Università di Genova

Dipartimento di neuroscienze, riabilitazione, oftalmologia, genetica e scienze materno-infantili (DINOGMI)

Corso di Dottorato in Neuroscienze XXXIII ciclo

Curriculum: Scienze delle attività motorie e sportive

A.A. 2020-2021

Tesi di Dottorato

Sviluppo di un classificatore real-time per l'identificazione del pattern motorio del Sit-To-Stand

Candidato:

Mirko Job

Relatore:

Marco Testa

Coordinatore:

Angelo Schenone

Piero Ruggeri

DINOGMI 2018-2022 | DIPARTIMENTO DI ECCELLENZA MIUR

REHE Lab

Department of Neuroscience, Rehabilitation, Ophthalmology, Genetics and Maternal and Child Sciences (DINOGMI)

PhD Course in Neuroscience - XXXIII cycle

Curriculum: Sciences of motor and sports activities

A.A. 2020-2021

PhD Thesis

Development of a real-time classifier for the identification of the Sit-To-Stand motion pattern

Candidate:

Mirko Job

Tutor:

Marco Testa

Coordinator:

Angelo Schenone

Piero Ruggeri

Contents

Index Of Figures

Index Of Tables

Abstract

The Sit-to-Stand (STS) movement has significant importance in clinical practice, since it is an indicator of lower limb functionality. As an optimal trade-off between costs and accuracy, accelerometers have recently been used to synchronously recognise the STS transition in various Human Activity Recognition-based tasks. However, beyond the mere identification of the entire action, a major challenge remains the recognition of clinically relevant phases inside the STS motion pattern, due to the intrinsic variability of the movement. This work presents the development process of a deep-learning model aimed at recognising specific clinical valid phases in the STS, relying on a pool of 39 young and healthy participants performing the task under self-paced (SP) and controlled speed (CT). The movements were registered using a total of 6 inertial sensors, and the accelerometric data was labelised into four sequential STS phases according to the Ground Reaction Force profiles acquired through a force plate. The optimised architecture combined convolutional and recurrent neural networks into a hybrid approach and was able to correctly identify the four STS phases, both under SP and CT movements, relying on the single sensor placed on the chest. The overall accuracy estimate (median [95% confidence intervals]) for the hybrid architecture was 96.09 [95.37 - 96.56] in SP trials and 95.74 [95.39 – 96.21] in CT trials. Moreover, the prediction delays (≅ 33 ms) were compatible with the temporal characteristics of the dataset, sampled at 10 Hz (100 ms). These results support the implementation of the proposed model in the development of digital rehabilitation solutions able to synchronously recognise the STS movement pattern, with the aim of effectively evaluate and correct its execution.

Introduction:

The Sit-To-Stand

I.1 A key functionality in the light of a global demographic ageing

The United Nations organization (UN) recognise the increase in the average age of the world population as one of the main factors which influence and guide its 2030 agenda for sustainable development [1]. From the global report of 2019, 703 million people, with more than 65 years old, have been surveyed, with East and South-East Asia holding the primacy for the largest number of older adults (260 million) followed by Europe and North America (200 million). In adherence to the current trend, these numbers are destined to grow, increasing the percentage of "Over 65" in the world by 16% in the next three decades [Fig.I.1].

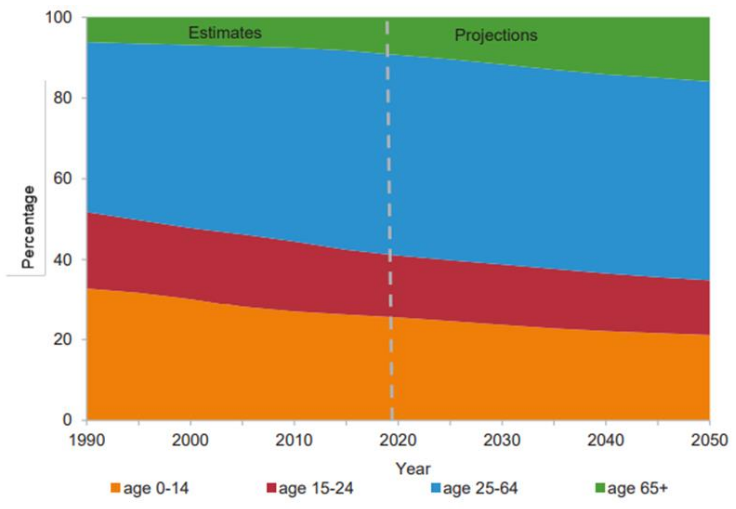


Figure I.1: Global population stratified by age groups, 1990-2050. [1]

If on the one hand, this phenomenon highlights the important goals achieved by health care in modern society, on the other hand, it emphasises the need to adopt economic, social, and technological strategies to guarantee an adequate quality of life for elderly people. Following the above statements, the World Health Organization (WHO) defines the principle of Healthy Aging as "the process of developing and maintaining the functional ability that enables wellbeing in older age". The expression functional ability describes the interaction between the psycho-physical abilities of a subject within the environment and the community in which he/she is inserted, considering his/her relationships, values, attitudes, and policies or services at support [2]. According to this definition, the quality of life in elderly populations implies the maintenance of their independence in the activities of daily living (ADLs) and their social engagement [3], [4]. Together, these factors promote physical and [5] mental health [6], survival and intellectual functioning [7], [8], and for this reason, they are considered objective indicators of successful ageing [9]. In this context, the transition from the sitting position to the upright stance, identified by the term Sit-To-Stand (STS), is an important element for the individual's independence in everyday life. Despite the apparent simplicity, the STS is the functional bridge that connects static and dynamic phases in human movement [10]. It represents a fundamental prerequisite for maintaining the vertical stance and for walking, which in turn are essential functions for the interaction of a person in the surrounding environment [11]. Furthermore, lower limb muscle force is correlated with functional status and ability [12]. From these statements, it is clear how eventual changes in the performance of the STS movement are directly related to the quality of life, being predictors of adverse events in presence of neuromotor diseases and/or in old age [13]–[16].

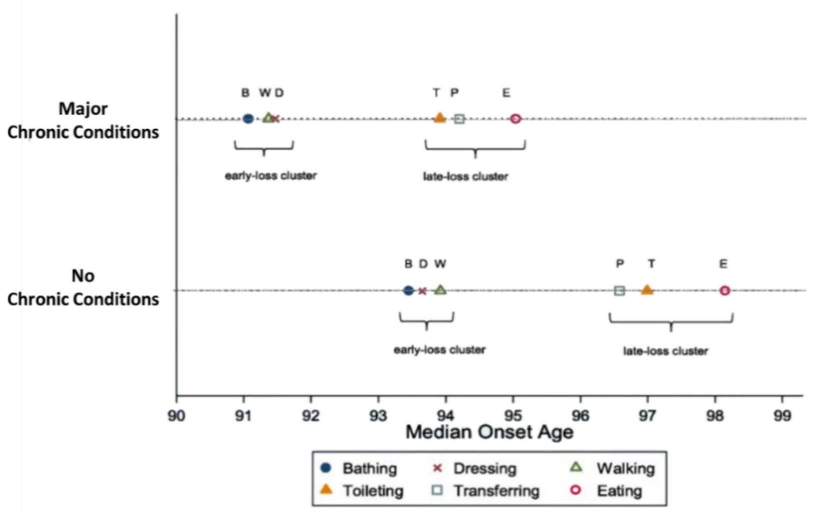


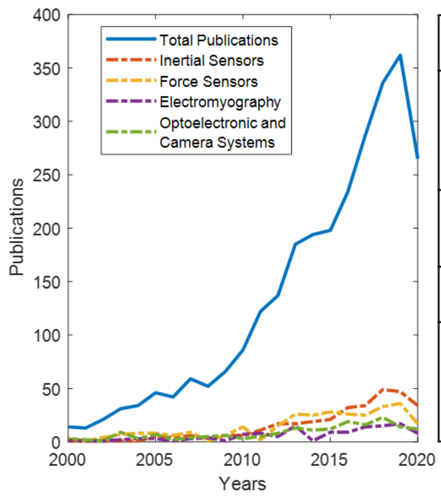
Figure I.2: Onset age of ADL disabilities for those with and without major chronic conditions. [17]

Specifically, it has been shown that STS (and in general 'Transferring') dysfunctions have a later onset than walking impairments [17], [18], manifesting majorly in elderly subjects [Fig.I.2]. Such a phenomenon is related to the inevitable functional decline faced by older adults as a physiological consequence of the aging process or a symptom of chronic diseases [11]. Munton and colleagues highlighted that 42% of a population of 379 older people reported difficulties associated with the STS movement across different neurological disorders [19]. A later survey identified the same outcome in 81% of 101 individuals with Parkinson’s disease [20]. The STS has been indicated as the most difficult and mechanically demanding motor task associated with ADLs [21]. For comparison purposes, the magnitude of moments (torques)[1] at the hip articulation is greater during the STS compared to other activities such as climbing stairs or walking [23]. The dynamic of the movement itself requires a considerable amount of muscle-force and joint torques in the lower limbs [21], [24], [25] as well as an optimal balance control to lift the centre of mass (COM) against the gravity effect [11], [26]. With the natural ageing process, these skills tend to decline after 50 years, with a muscle strength reduction of 1-4% per year, (i.e. sarcopenia), [27], [28]. These factors lead to increased difficulty in the STS task for older adults, which perform the movement close to their maximal functional level [29], [30]. The power[2] produced during STS significantly correlates with many established functional batteries and single measures of strength, speed, and balance [27], offering a versatile solution applicable in clinical settings, with advantages in terms of time and costs for the health-care system. Nonetheless, it is important to underline that like any complex movement, the STS is the result of an elaborate integration of physiological and psychological functions, rather than a simple measure of lower limb strength [32]. Hence, based on the global ageing trend and its possible impact on the ability to perform ADLs, the STS evaluation will play an important role in the coming years as an important index of institutionalization, disability, and reduced mobility in the elderly [24].

1 Joint moments describe the net sum of all internal moments delivered by all internal structures around a joint. Typically, joint moments are delivered by muscles and, toward the end range of motion, by ligamentous or bony tissue. [22]

2 The power is the product of force and speed. It has been proven that this parameter is much more important than strength for performing daily activities like rising from a chair, walking or stair climbing. [31]

I.2 The Sit-To-Stand movement in clinical practice



Total Publications	(sit-to-stand or "sit to stand")
Inertial Sensors	and (accelerometry or "accelerometry" or accelerometry [mh] or acceleromet* or inertial or inertial sensor* or "inertial sensor*" or actigraphy or "actigraphy"[mh] or wearable sensor* or "wearable sensor*" or wearable* or smart*)
Force Sensors	and ("posturography" or force plate or "force plate" or ground reaction force or "ground reaction force" or kinetics)
Electromyography	and (electromyography or "electromyography" [mh] or emg)
Optoelectronic and Camera Systems	and (mocap or motion capture or "motion capture" or (kinematic or motion or movement) and (analysis or assessment or measure*) and (camera or video or optoelectronic or stereophotogram* or rgb or marker*))

Figure I.3: Number of publications per year (left), for the researched terms (right), qualitatively portraying the portion of literature focused on the STS. (https://pubmed.ncbi.nlm.nih.gov/) (09-08-2020)

A qualitative search on MEDLINE outlined a continuously growing interest regarding the Sit-To-Stand movement in the last two decades [Fig.I.3], being adopted in clinic as a valid measure of patients' functionality [33], either as a single test or as a part of clinical scales and batteries. Simple temporal outcomes measured during the STS have shown significant correlations with many physical performance measures such as gait speed [34] and the 6-Minutes-Walk-Test [35] in healthy older adults and patients affected by Parkinson's disease, Stroke, and vestibular disorders [33]. In the following paragraphs, a summary of the main STS-based performance tests used in clinical practice is provided.

I.2.1 Five-Time Sit-To-Stand Test

In the Five Time Sit-To-Stand-Test (FTSTS) [36], the participant is asked to rise from a chair five consecutive times as quickly as possible. The time required for completing the task represents the main clinical outcome. First introduced just as a measure to quantify strength in lower limbs, the FTSTS displayed more general correlations with the overall level of patients' functionality.

Whitney and colleagues [37] highlighted that the FTSTS improves the discriminative ability of the Activities-specific Balance Confidence Scale[3] and the Dynamic Gate Test[4] in the identification of people with impaired balance. Furthermore, the correct execution of the FTSTS implies the ability to integrate visuospatial information [38], which ultimately associates with cognitive function assessed by the Pfeiffer Short Portable Mental Status Questionnaire[5]. In particular, individuals that complete the FTSTS within 15 s have a reduced probability to develop cognitive dysfunction. FTSTS is also used to assess fall risk in the elderly [39] and it is included in the Short Physical Performance Battery[6][40]. Unfortunately, the main limitation is the identification of the right time threshold. In this case, analysis of the Postural sway and the Jerk of the movement could be added as quality performance indicators.

I.2.2 30s Chair Test

The 30-second Chair Stand Test (30CST) [41] consists of counting the number of completed stands in 30 s with the hands crossed against the chest. The total number of stands completed is used as the main clinical outcome and a threshold of 8 repetitions can distinguish between patients with low and high physical performance. The 30CST was introduced to avoid the floor effect associated with the difficulty of many patients in executing chair stand tests [30] and it is used as an indicator of lower limb strength. Bruun et al. [42] highlighted strong associations between the de Morton Mobility Index[7] [43] and the 30CST, which presents the added value of being faster and easier to deliver, with the possibility to be implemented in acute settings. Moreover, the 30CST is included as a part of the

3 The Activities-specific Balance Confidence Scale is a self-administered questionnaire tool used to assess the confidence is performing various ambulatory activities without falling or experiencing sense of unsteadiness. BC scale consists of less and more challenging daily activities.

4 The Dynamic Gate Test is an 8-test to assess to quantify the dynamic balance abilities. It has been demonstrated to be very sensitive test, since it evaluates walking during challenging tasks.

5 The Pfeiffer's Short Portable Mental State Questionnaire is used for the assessment of organic brain deficit in elderly patients.

6 The Short Physical Performance Battery comprised the gait speed, chair stand and balance tests. It has been used as a predictive tool for possible disability, mortality. and institutionalization, allowing the functional monitoring in older people.

7 The de Morton Mobility Index is a test comprised of 51 items that measure patients' mobility across the spectrum from bed-bound to independent mobility. It is a standard procedure of the admission protocol in an emergency department short-stay unit.

Fullerton Advanced Balance Scale [44][8] for assessing the endurance and the strength of lower limbs.

I.2.3 Timed Up And Go Test

The Timed Up and Go (TUG) test was introduced in 1991 as a modification of the former Get-Up and Go test [45]. The TUG measures the time in seconds for a person to rise from a sitting position on a standard armchair, walk 3 meters, turn, walk back to the chair, and sit down. The person wears regular footwear and uses his/her customary walking aid. The original purpose of the TUG was to test the basic mobility skills of frail elderly people. The test has been used in other populations, including people with arthritis, stroke, and vertigo. The TUG may be particularly well suited for the quantification of those disorders characterised by a poor sequencing of well-learned motor skills, which is a common problem in people affected by Parkinson’s disease.

8 The Fullerton Test is mainly intended to identify highly-active older adults who are at an increased risk to experience fall-related injuries due to sensory impairments. The test uses both dynamic and static balance under different situations to identify balance deficits in older adults.

I.3 The instrumented analysis of the Sit-To-Stand

Despite being extensively used in practice, standard clinical assessments for mobility, walking, and balance depend on different contextual factors, in terms of patients, clinical staff, and environmental characteristics [46]. By requiring the presence of a trained assessor, these evaluation tools are affected by an intrinsic subjective bias that reduces their diagnostic power and the reliability of the measures [47]. This limit is reflected in a reduced capacity to monitor the evolution of pathology or the efficacy of a therapy [46], [48], [49]. Consequently, the technological progress of the last decades has led to the development of a broad variety of sensors, promoting the adoption of automated methods in clinical practice and research. Regarding this subject, many studies in the literature emphasize the added value of instrumented measures to better understand the biomechanics of the movement. Specifically, the STS has been analyzed since the late 1980s with the use of camera and optoelectronics systems, force plates, and electromyography (EMG). In a review of 2002, Janssen and colleagues [50] identified various elements that can affect expressly or implicitly (as confounding factors) how the STS movement is executed, by gathering evidence from experimental studies performed through the use of sensor-based technology. In their work, they divided these “determinants” into three main categories, presented in Table I.1.

Chair-Related Determinants	1. Height of chair seat 2.With armrests 3.Chair special type 4.With backrest
Subject-Related Determinants	1.Age 2.Disease (eg, stroke, arthritis, low back pain) 3.Muscle force 4.No footwear
Strategy-Related Determinants	1.Speed 2.Foot position 3.Trunk position/movement 4.Arm use with armrest 5.Terminal constraint 6. Arm movement 7.Dark versus light 8."Fixed" joints 9.Knee position 10.Attention 11.Training

Table I.1: Determinants of the Sit-To-Stand. [50]

The objective analysis of these aspects has revealed important insights into the dynamics of the STS movement and how this can change according to the specific characteristics of different populations.

I.3.1 Events and phases standardisation of the Sit-To-Stand

A large number of studies tried to establish a standard definition of the events occurring during the execution of the STS using sensor-based analysis. However, although it may seem easier to describe, compared to other movements (i.e. walking), the STS is the one that is characterised by the higher inter/intra-subject variability [51]. Even if temporally well-defined between the seated position and the standing stance, it implies a coordinated movement of flexion and extension involving the head, arms, trunk, and lower body segments. For this reason, the execution of the STS movement highly varies from one repetition to another, between and within-subjects. On top of that, different studies employ different types of instrumentation and measures, resulting in different descriptions of the same STS event. As an explanatory example, the beginning of the STS movement has been defined as the initiation of the trunk forward flexion and momentum [52], [53], or as the first vertical force deviation greater than 10 N [54]. Even the instant of rising from the chair, logically well-recognisable, has been determined with a plethora of various methods, by implementing a switch on the chair [55] or by identifying the peak of the horizontal [56] or the vertical [57] components of the Ground Reaction Force (GRF). In general, what emerges is a large variability in methodologies, terminologies, and biomechanic definitions of the movement that makes difficult the identification of a univocal standard for the categorisation of the STS motion pattern. The first attempts to approach this challenge exploited the data acquired from three-dimensional camera-based systems to divide the movement into two distinct phases [23], [58]:

- The flexion phase, representing the first 35% of the movement and characterised by the forward flexion of the head, neck, trunk, and pelvis;
- The extension phase, characterised initially by the backward movement of the neck toward the vertical axis, followed by the trunk and by the extension of the hips ankles, and knee;

Nonetheless, this subcategorisation appeared to be excessively simplistic as it did not model properly the transfer of the COM of the body [59]. To overcome this limit, new finer standardisations were subsequently introduced, focusing on the analysis of body segments' kinema, momentum and velocity events [51], [60]–[62].

For instance, Roebrock modeled the STS movement into three phases [Fig.I.4] relying on the velocity of the COM:

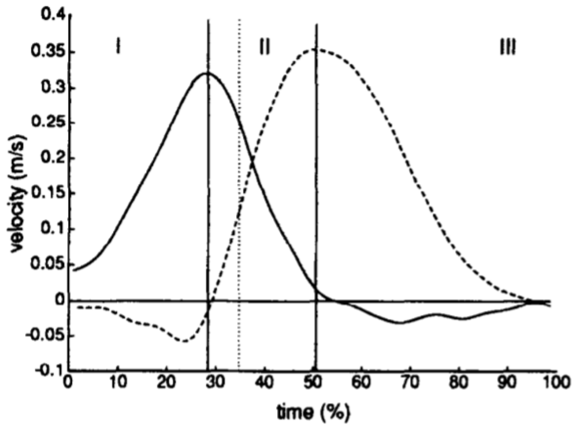


Figure I.4: Horizontal and vertical velocity of MCB during STS transfer. Mean curves in time (%) are given (n = 10). The curves refer to horizontal (solid line) and vertical (dashed line) velocity. The vertical lines indicate the limits between phases (solid) and the instant of seat-off (dotted). Phases: acceleration phase (I), transition phase (II), and deceleration phase (III). [62]

- The acceleration phase, starting from the beginning of the movement until the COM reach the maximal horizontal velocity;
- The transition phase, where the vertical acceleration increases until its maximum value at the expense of the horizontal acceleration which is gradually zeroed;
- The deceleration phase, characterised by the decrement of the vertical velocity decreased until the end of the movement;

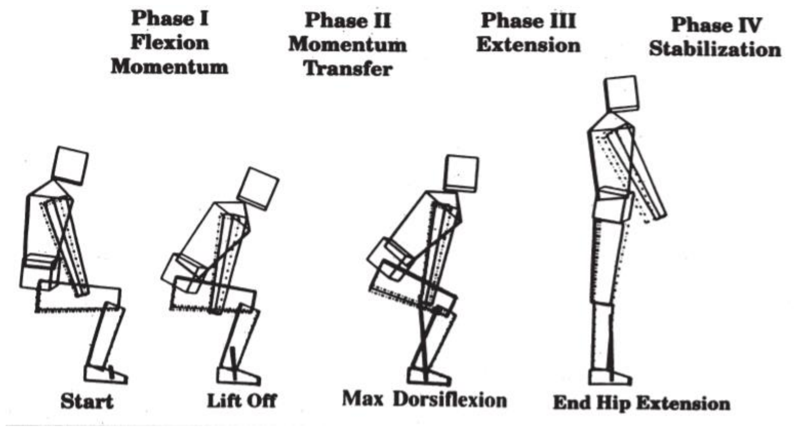


Figure I.5: Four phases of rising marked by four key, events. [52]

Schenkman and colleagues proposed another model, based on the movement of the trunk and the ankle dorsiflexion [52], [63], categorising the STS in 4 distinct phases [Fig.I.5]:

- The flexion momentum phase, temporally delimited by the beginning of the trunk movement and the moment preceding the raising from the chair;
- The momentum-transfer phase, starting with the lifting from the seated position to the reaching of the maximal dorsiflexion of the ankles;
- The extension phase, characterised by the complete extension of the hips;
- The stabilization phase, beginning with the completion of the hip extension and continuing until the reach of a stable upright stance;

Considering the intrinsical variability in the kinematics of the movement and the lack of an accepted standard to describe the STS motion pattern, a later work of Etnyre and Thomas [51] grounded its methods on a kinetic evaluation of the STS through the analysis of the GRF profiles.

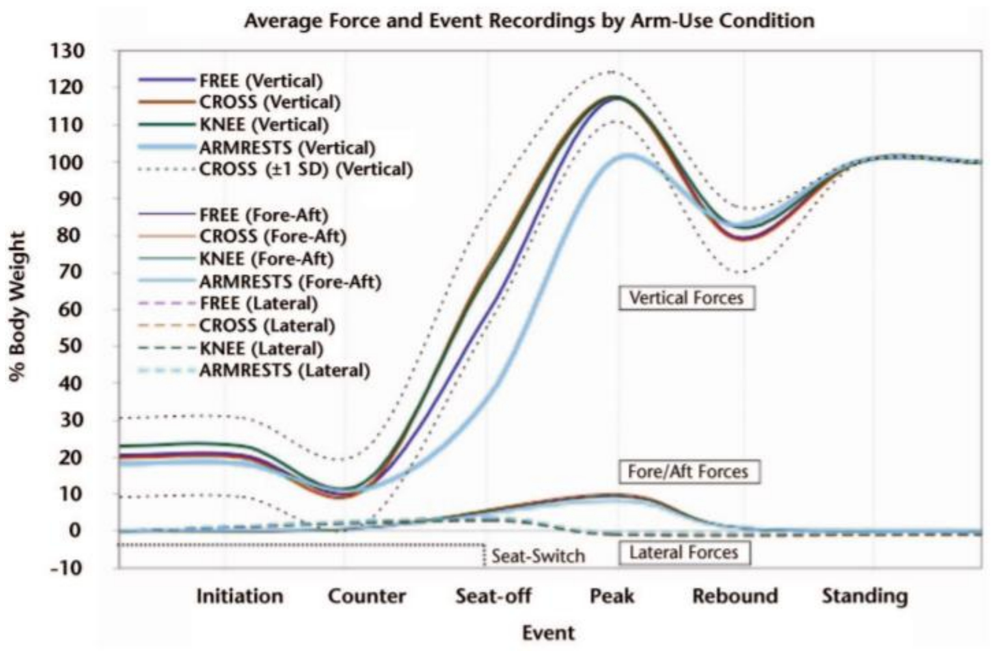


Figure I.6: Averaged normalised recordings over all trials and participants from the force platform and the seat-switch event for the 4 arm-use conditions by events. The arms-crossed condition was considered as a control condition, and the standard deviation was plotted for that condition. Positive deflections of the fore-aft traces represent backward force (ie, moving the body forward). Lateral force to the right is shown in the positive direction, left toward negative. FREE: arms-free condition, CROSS: arms-crossed condition, KNEE: hands-on-knees condition, and ARMRESTS: hands-on-armrests condition. [51]

Across a sample of healthy young people (age: 21.8 ± 4.6), they identified eleven essential events [Fig.I.6] under different raising conditions (i.e. arm free, arm crossed, hands on knees, hands on armrest) described as follows:

- 6 vertical GRF events, initiation of movement, a counterforce before seat off, seat-off, peak force, post-peak, rebound force, and final steady standing force;
- 3 horizontal events, the start of force, peak force, and end of force;
- 2 lateral events;

Despite the population (young, healthy) that did not portray the usual characteristics of patients in clinical practice, the recognition of invariable events during the STS, independently from the movement strategies, provided interesting intuitions for discretely identifying key elements in its motion pattern.

I.3.2 Raising strategies in the Sit-To-Stand movement

If, as a first contribution, the use of objective evaluations allowed the description of certain invariant elements in the STS, it also permitted the quantitative identification of specific movement strategies related to the underlying characteristics of different groups of patients. Often underestimated by clinicians, which focus majorly on the success of the physical task per se, the information on how the upright position is attained could improve the outcome of the treatment itself, favouring the determination of which components of motor control are impaired [64]. Studies on healthy younger people and older adults raising from a chair under different conditions outlined two main movement strategies [29], [52], [58], [64]–[67]. The momentum transfer strategy is characterised by a limited flexion of the trunk and an early lifting from the chair when the COM is still distant from the feet. This implies high balance control and greater momentum on the lower limbs, to lift the body mass to a stable upright stance. For this reason, this strategy is typically observed in younger people, who have more strength and perform this movement almost automatically. Conversely, the elderly often presents functional deficits, in strength, physical characteristics and in sensory-cognitive terms, which ultimately precludes them from the feasible execution of the momentum transfer strategy. For this reason, frailer populations generally perform a stabilisation strategy (or flexion strategy) [29], [52]. This manoeuvre aims at reducing the global instability of the movement and consists of an accentuated flexion of the trunk toward the knees, moving the COM over the feet.

I.3.3 Evidence from Force plates

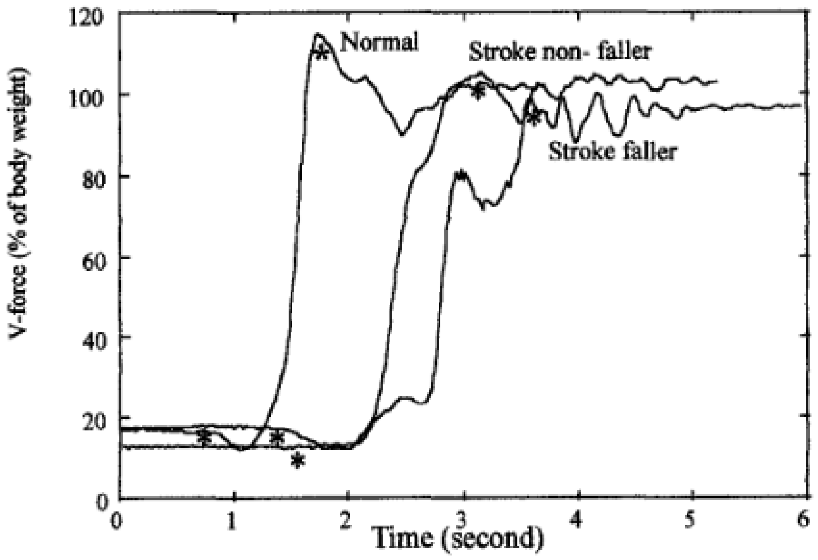


Figure I.7: Comparison of vertical force change from sitting to standing in healthy subjects, stroke non-fallers, and stroke fallers. The slope of the curve between the two asterisks on each of the three lines indicates the rate of rising in force (dF/dT) during rising. [68]

Most of the modern, full-featured force plates assess the kinetics of the movement by measuring simultaneously [69]:

- The three-dimensional components of the GRF;
- The centre of pressure of the body (COP);
- The centre of force;
- The moment (torque) around each of the axes;

In [68], the analysis of these parameters highlighted significant differences between stroke fallers and non-fallers patients. In general, stroke patients needed more time to stabilise the cop during the rising transition, with a loading asymmetry on the healthy leg. Stroke Fallers patients showed high medio-lateral COP sway, showing poorer stability and a higher risk of falls. Additionally, the between-group comparison of the vertical GRF profiles pointed out a lower rate of rising (dF/dT) in stroke fallers compared to non-fallers or healthy people [Fig.I.7]. Through the analysis of the COP displacement, Janssens et al. [70] explored the STS performance in Individuals with Chronic Obstructive Pulmonary Disease (COPD). The COPD group displayed greater difficulties in the phases that require greater postural control, i.e. the stand and the stand-to-sit phases compared to the healthy group [Fig.I.8].

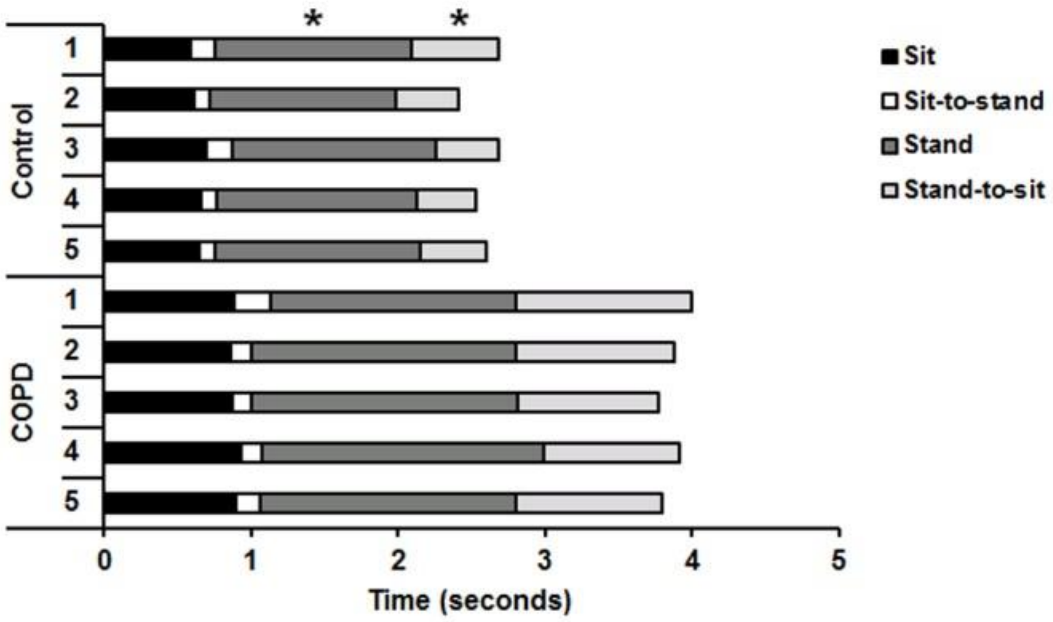


Figure I.8: Mean duration of the five sit, sit-to-stand, stand, and stand-to-sit phases. The phase durations of the sit-to-stand-to-sit (STSTS) task are displayed for the control group and COPD group. (=p<0.05 between both groups for five STSTS movements).* [70]

Recently, a similar evaluation of the COP underlined the possibility to differentiate functionally independent healthy older adults from younger populations [71]. The elderly displayed a greater global sway and higher velocity during the execution of the STS movement, suggesting the possibility to identify functional deterioration in the early stages.

I.3.4 Evidence from Electromyography

The analysis of the EMG patterns of activation under different initial postural settings allowed the identification of two main groups of muscles involved in the execution of the STS [72]. The tibialis anterior, the abdominal, the sternocleidomastoid, the soleus, and the trapezius showed different behaviors relative to the different experimental conditions under which the STS is performed. Such peculiarity is characteristic of the “postural” muscles [73] that must maintain balance throughout the movement, but not of the main “executor” muscles, involved in the effective execution.

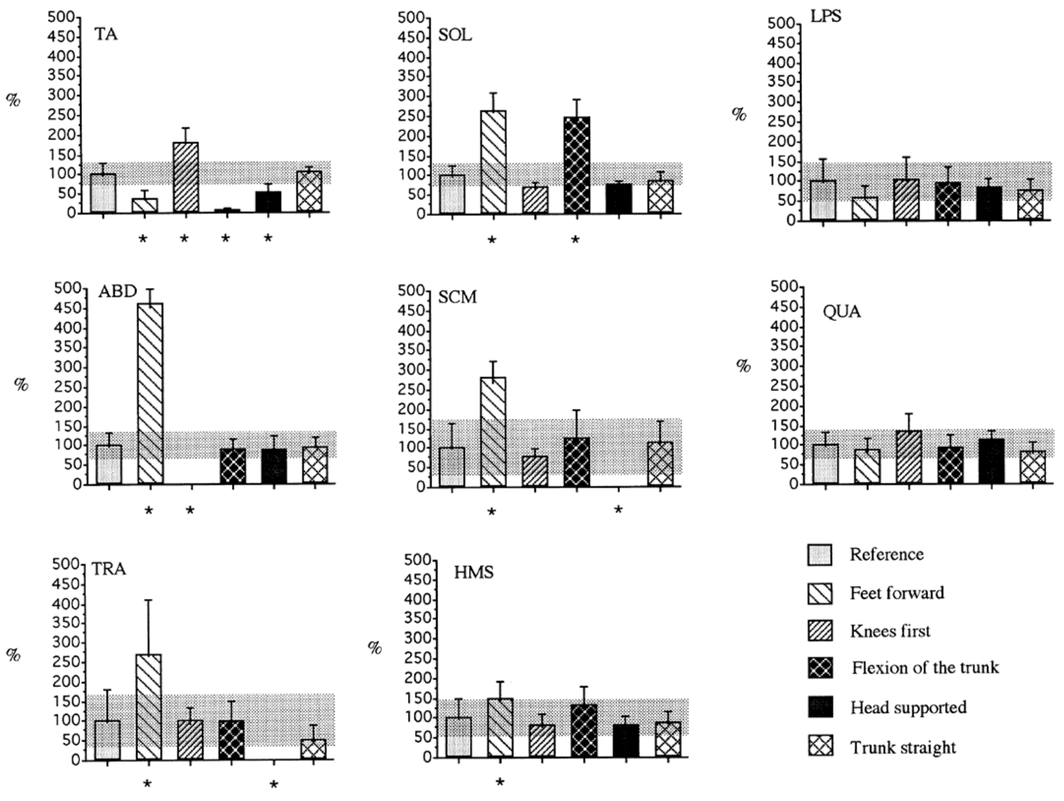


Figure I.9: Area (μV×ms) of the EMG activity of the muscles examined in each condition. The area in grey marks the limits of 1SD above and below the mean area calculated in the reference condition for the muscle specified in each graph. [72]

Accordingly, the hamstrings, the lumbar paraspinal, and the quadriceps were the real actuators of the STS, showing a consistent activation pattern independently from the postural condition [Fig.I.9]. By retaining very specific information on how the movement is performed, the study of the EMG signal can unveil condition-related characteristics in the STS execution. As an example, it has been pointed out that the lower limbs' activation patterns in hemiplegic patients differ significantly between fallers and non-fallers subjects [74]. In particular, as suggested by the authors, a limited (or absent) activity of the tibialis anterior and quadriceps muscles of the unaffected limb discriminated those individuals who are more prone to fall. This was accompanied by early or excessive activation of the soleus muscles in the hemiplegic limb, contributing to the instability and stiffness of the ankles. Similarly, in a recent publication [75] the analysis of the EMG pattern of the gastrocnemius lateralis revealed that fallers opted for a different strategy to maintain a certain condition of stability during the STS.

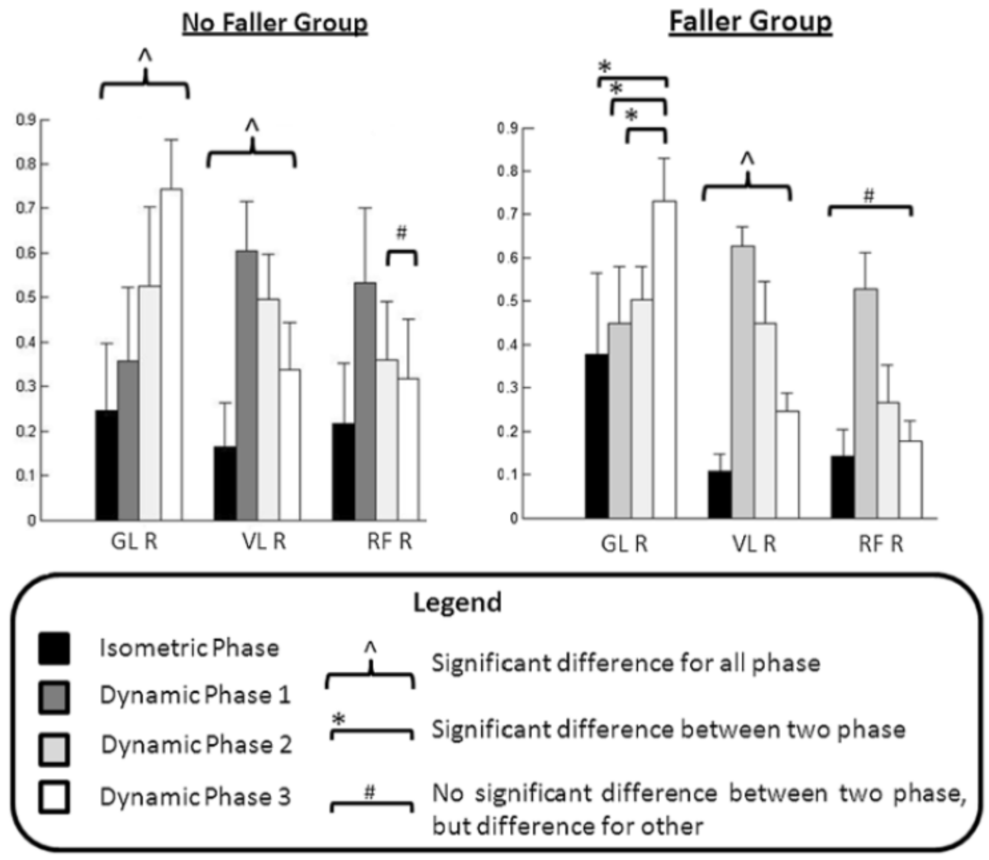


Figure I.10: Sequence of muscular activation movements in old subjects during sit-to-stand movement, RFR: rectus femoris right side, VLR: vastus lateralis right side, GLR: gastrocnemius lateralis right side. [75]

Thus, as displayed in Figure I.10, while in the non-faller group the activity of gastrocnemius lateralis increased significantly during each phase of the STS, in the faller group it remains quite identical during the first stages of the movement. From a clinical point of view, these results can be used to plan effective therapy to prevent falls, by reinforcing the compensatory effect of specific postural muscles and consequently enhancing balance control.

I.3.5 Evidence from Optical systems

Starting from the first experiment of three-dimensional gait analysis by Otto Fischer and Willhelm Braune in the late 19th century [76], optical systems have evolved to become the gold standard reference in human kinematic analysis. They include various technologies to determine the positions of predetermined points on the body. As for other types of sensors, the information collected through these systems has been used to primarily describe the STS movement patterns and strategies (as described in Chapters I.3.1 and I.3.2), and to describe the eventual difference in frailer individuals. As an example, it has been highlighted that obese patients use different motor strategies as an adaptation to the altered weight distribution [77]. This results in a negative overstress of the joints, especially during postural transitions. Compared to healthy subjects, during STS tasks obese people tend to limit the trunk flexion, moving the feet backward in respect to the initial position. This kinematic strategy leads to a minimisation of the hip joint torque (lowering at the same time the low backloading) but it exposes the knee to a higher load increasing the muscular fatigue.

I.4 Applications of wearable and smart sensors in the analysis of the Sit-To-Stand movement

One major drawback of the presented devices and instrumented measures is that they are not easily applicable outside the laboratory environment [12], [47]. Despite their advantages over performance assessments, force plates and optoelectronic systems need certain expertise for their correct use, and are extremely expensive, both in terms of costs and time [78], precluding their implementation in standard clinical practice. In the later period, the technological advances in the sensor industry led to the development of wearable sensors that can be implemented for the kinematic analysis of the movement [79]. The low production costs, together with the reduced dimensions, make the inertial measurement unit (IMU) sensors the perfect trade-off between accessibility, portability, and measurement accuracy, allowing their comprehensive use in everyday life and clinical routine. For these reasons, this chapter discusses more in-depth the principles and the clinical implications of the IMU sensors, with particular attention to their application in the analysis of the STS.

I.4.1 Principles of inertial sensing

Inertial sensors are embedded micro-electromechanical systems (MEMS) that measure the static and dynamic components of the force of acceleration, angular rate, and orientation of the device reference frame [80]. They encompass different technologies "borrowed" from aerospace, industrial, and robotic engineering [78].

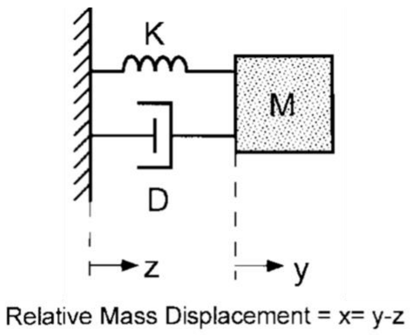


Figure I.11: Mechanical model of a general MEMS accelerometer.[81]

The two main components of an IMU system are accelerometers and gyroscopes. Accelerometers can be modeled as second-order mass-damper-spring systems [81] consisting of a mass (M), a spring constant (K), and a damping factor (D). The functioning principle is relatively simple, an external acceleration causes the movement of the device, which stretches the spring in the direction opposite to the movement. The acceleration is derived from the force applied to the spring element, by measuring the relative displacement between the external support and the mass [Fig.I.11]. Gyroscopes are specific sensors used to measure the angular velocity of the reference frame on which they are mounted [82]. In MEMS applications, gyroscopes are based on the transfer of energy between two vibration modes, exploiting the effect of the Coriolis acceleration[9]. The device is outlined by two orthogonal mechanical excitation directions [Fig.I.12]:

- The mass is vibrating in its natural frequency along its "drive direction";
- Under rotation, the Coriolis effect change the vibration in the "sense direction";

In this way, by completely characterising the drive axis, the displacement along the sense direction is proportional to the angular velocity.

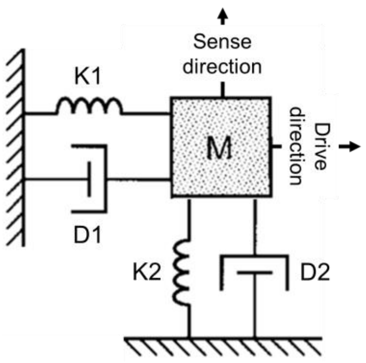
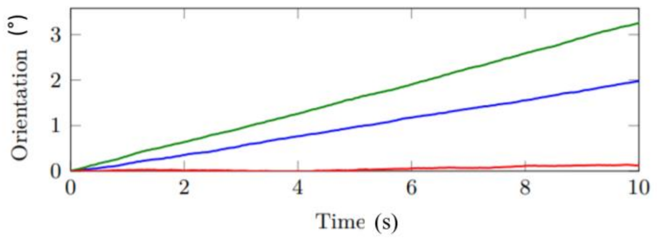


Figure I.12: Mechanical model of a general MEMS gyroscope. (modified by [81])

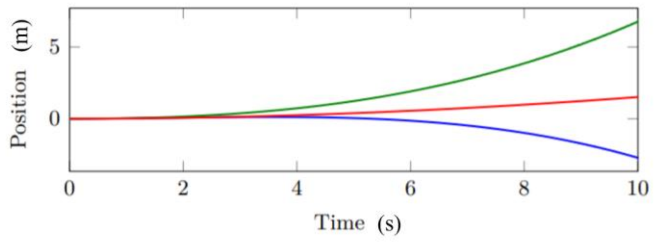
These variables are consequently integrated to estimate the position and the orientation of the IMU sensor through a process called dead reckoning. Theoretically, the orientation is obtained by single-integrating the angular velocity from the gyroscope. Starting from the pose of the IMU, the earth's gravity component is removed and the accelerometric measures are double integrated to obtain the position of the device [83]. However, the orientation estimate is affected by noise and a slowly time-varying bias, which adds an offset component to the

9 The Coriolis force is an apparent force to which a body is subjected if observed by a system in rotary motion with respect to an inertial reference.

output even in the absence of any input. These errors accumulate through integration, leading to a measurement drift in time, especially for the yaw angle.



(a) Integrated orientation for the position in x- (blue), y- (green) and z-direction (red).



(b) Integrated position for rotation around the x-axis (blue), the y-axis (green) and the z-axis (red).

Figure I.13: Position and orientation estimates based on dead-reckoning of the inertial sensors only. The data is collected with a Sony Xperia Z5 Compact smartphone that is lying stationary on a table [83].

This so defined bias drift is a complex phenomenon that depends on various factors related to temperature, operating time, and type of movement performed [84], [85]. To adjust its effect, the estimates of position and orientation are corrected with the use of wavelet analysis [86] or by exploiting prior knowledge about the movement [87] and the information from supportive systems [85], [88]. In the latter case, Kalman filters are used to fuse contributes from multiple devices and ultimately improve the accuracy by reducing the bias. As an example, modern inertial sensors incorporate internal magnetometers to improve the performance in the estimation of orientation and position [Fig.I.14] [89].

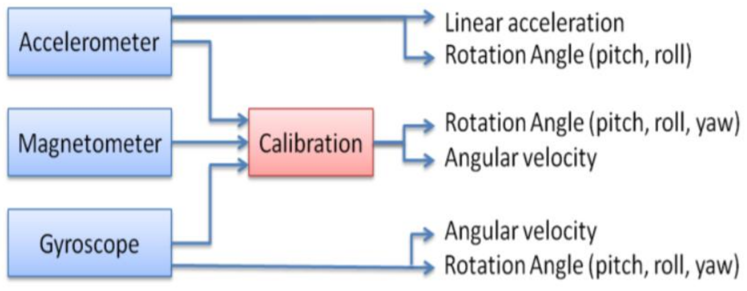


Figure I.14: Model of IMU sensor based on Accelerometers, Magnetometers, and Gyroscopes [90]

In absence of external magnetic interferences, magnetometers identify the local magnetic north and can be used to improve the estimation of the heading reference in IMU devices [91], with successful results in various fields. In this context, it is important to remind that all sensors must be properly calibrated to obtain optimal measures.

I.4.2 Towards an ecological and continuous assessment of the STS movement

Beyond the advantageous trade-off between cost and measurement accuracy, the most appreciable feature of IMU sensors is undoubtedly their portability. Starting from the late 1900s with the invention of the first integrated circuit, miniaturisation and manufacturing techniques evolved rapidly, affirming the importance of MEMS systems in the global market. Just in 2019, the MEMS market size was around \$11-\$12 billion, and it is forecasted to grow up between 15% and 18% in 2025 [92], [93] [Fig I.15]. In particular, the effect of COVID-19 pandemic in the medical field will accelerate the transformation of the healthcare organisation promoting a more patient-centric approach. Contextually, novel telehealth solutions, represented by more wearable and connected devices, will guide the technological development of the following years towards a continuous monitoring of the patients, to improve prevention and health outcomes.

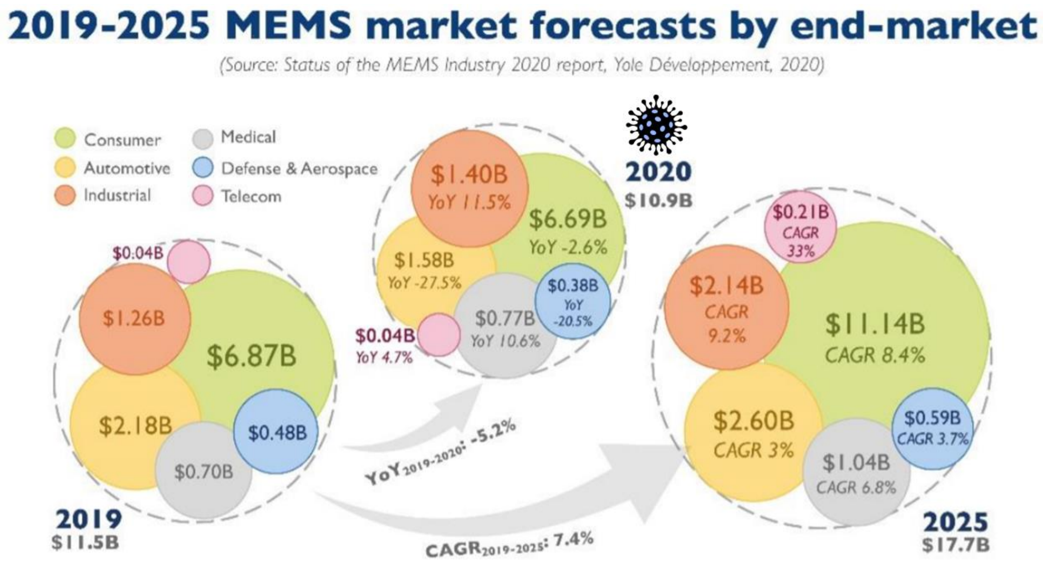


Figure I.15: The forecasted size of the market of the MEMS industry for 2020 (COVID-19) and 2025. MEMS=Microelectromechanical systems; CAGR=Compound Annual Growth Rate; YoY=Year-over-Year. [92]

Following this trend in the medium term, there will be a transition to more wearable ultra-sensitive devices packing a lot of sensors but also a move to more consumer healthcare [92]. Nowadays, inertial sensors are included in many devices we use daily. Consequently, the interest in health-related metrics collected from smartphones [94], [95], and activity monitors [96]–[98] is rapidly growing among researchers, as they allow to perform objective clinical evaluations outside the laboratory environment [94]. By not being constrained in terms of time, costs, and medical personnel, the measurements performed through wearable sensors provide the opportunity for a continuous assessment of the patients' health status in their own everyday life, potentially permitting the early detection of functional decline [99]. In this sense, it is important to distinguish the patient's ability to perform an exercise in front of a clinician from that of carrying out their daily activities outside the clinical environment. Functional evaluation strongly depends on the external factors in which they are applied, in terms of patients, clinical staff, and environmental characteristics [100], [101]. In the International Classification of Functioning, Disability, and Health (ICF) [102], the term functional capacity describes what individuals can do in a standardised environment, while the functional performance describes what they do in their daily environment. Similarly, Van Lummel and colleagues [103] distinguished physical function [Fig.I.16] into physical performance and physical activity.

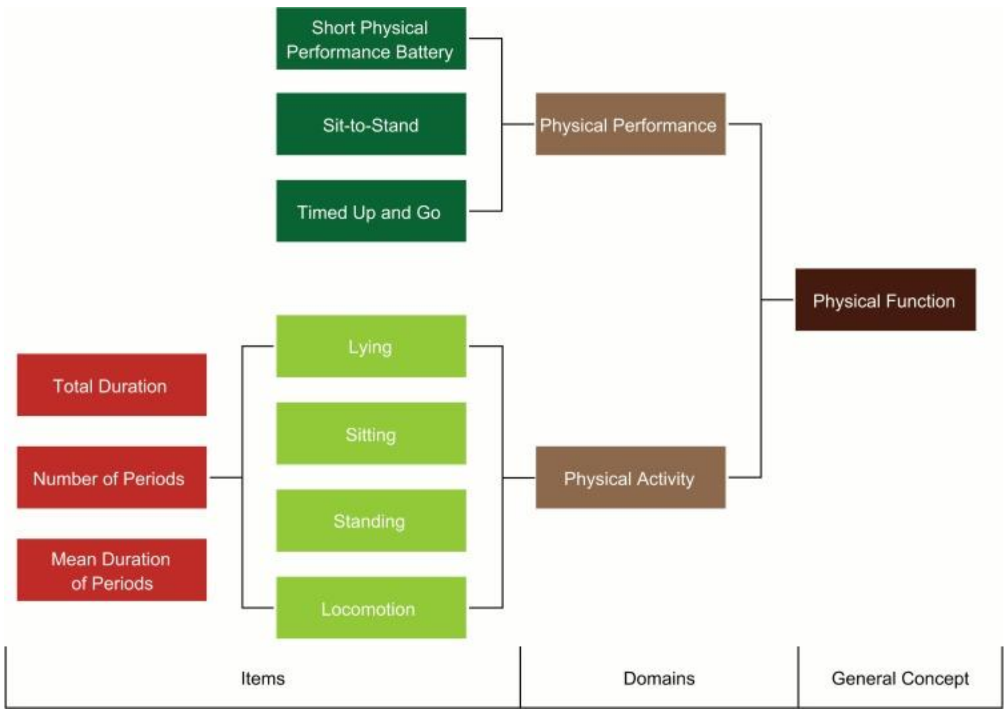


Figure I.16: Mobility measures presented in a framework with physical performance and physical activity as domains of physical function. [103]

In their work, the low correlations obtained underlined the clear distinction between these two domains. Therefore, an improvement in the clinical tests does not automatically reflect on the quality of ADLs, and continuous unsupervised screening of the patients in their daily routine (through inertial sensors and wearable devices) might be more representative of their health conditions.

I.4.3 Clinical evidence

Temporal duration and kinematic parameters characterising the STS motion pattern can be described using IMU sensors [104]–[107] and can be used to identify age-related differences [108], [109]. Moreover, since the STS is influenced by somatosensation, balance, and psychological status [32], it is used to evaluate the fall risk [110]–[114], and the frailty level [115], [116]. As an example, in [108] the acceleration and the angular velocity from a single sensor positioned on the lumbar area were used to derive the trunk angle and consequently to calculate the duration of the STS motion pattern. These parameters, together with the relative coefficients of variation (CV) and the maximum values of angular velocity in the different STS phases, showed particular clinical interest, being able to discriminate between young and older adults [Tab.I.2].

		Young adults			Older adults			p-Value
Phase	Duration (s)	Median	Min	Max	Median	Min	Max	
Sit-to-stand	Total	1.45	1.14	2.58	1.98	1.65	3.49	<0.001
	Flexion	0.73	0.63	0.88	1.06	0.74	1.64	<0.001
	Extension	0.72	0.49	1.74	1.1	0.82	1.94	<0.001
Standing	Total	0.33	0	0.74	1.35	0.57	6.57	<0.001
Stand-to-sit	Total	1.47	1.18	2.28	2.59	1.34	3.21	<0.001
	Flexion	0.69	0.46	0.91	1.31	0.65	1.87	<0.001
	Extension	0.79	0.71	1.37	1.06	0.69	1.68	0.024
Sitting	Total	0.33	0.06	0.7	3.1	0.36	9.71	<0.001
Phase	**Angular rates (°/s)**							
Sit-to-stand	ω max flexion	124.62	90.04	192.7	91.62	57.31	125.46	<0.001
	ω max extension	57.22	20.7	98.9	54.67	25.57	93.33	0.323
	ω max flexion	79.68	50.32	117.63	40.93	22.99	72.71	<0.001
Stand-to-sit	ω max extension	102.15	60.42	138.22	107.31	65.65	170.29	0.527
Phase	**CV Duration (%)**							
Sit-to-stand	Total	7	2	15	26	7	42	<0.001
	Flexion	8	5	16	19	9	41	<0.001
	Extension	11	3	33	40	7	85	0.003
Standing	Total	40	5	96	55	26	121	0.08
Stand-to-sit	Total	8	3	39	19	7	51	0.001
	Flexion	12	2	36	22	9	44	0.005
	Extension	10	3	61	18	11	79	<0.001
Sitting	Total	36	8	69	57	38	140	0.002

Table I.2: Durations (s), maximum angular velocity (ω max, in°/s), and coefficient of variation of durations (percentage) of the 5 repeated sit-to-stand cycles of the young and older adults. P-values compared young and older adults are calculated using the Mann–Whitney U-test ($p < 0.05$). [108]

By focusing on the analysis of the STS transitions, Ejupi and colleagues [113] collected and analysed data from a single wearable pendant device, with an embedded tri-axial accelerometer, to identify fallers and non-fallers in a population of 119 community-dwelling older adults. They found that the maximum magnitude of the acceleration vector, the maximum velocity, and the vertical peak power were significantly different between the two groups [Tab.I.3].

Measureement	Fallers (n=34)	Non-fallers (n=60)	p-Value
Duration (s)	1.6 ± 0.5	1.5 ± 0.3	0.59
Max acceleration (m/s^2)	2.4 ± 1.0	2.9 ± 1.1	0.04*†
Max velocity (m/s)	0.6 ± 0.2	0.73 ± 0.3	0.03*
Peak power (W)	464.1 ± 225.3	594.4 ± 292.9	0.03*
Max forward lean (°)	15.8 ± 8.8	12.2 ± 10.2	0.09

*Table I.3: Sensor-Based Sit-To-Stand Assessment for the Fallers and Non-fallers. [113] *< 0.05; † Non-parametric test*

In [116], the examination of the anteroposterior orientation and the vertical acceleration signals pointed out peculiar information on how people with different levels of frailty execute the STS movement. By collecting data from a single IMU sensor positioned over the lumbar area, frail subjects showed lower accelerations compared to pre-frail and healthy groups during the raising and the sitting transactions, displaying reduced strength and a more cautious movement strategy. Additionally, frail people spent more time in the preparation of the raising movement, displaying a greater movement range along the sagittal plane [Fig.I.17].

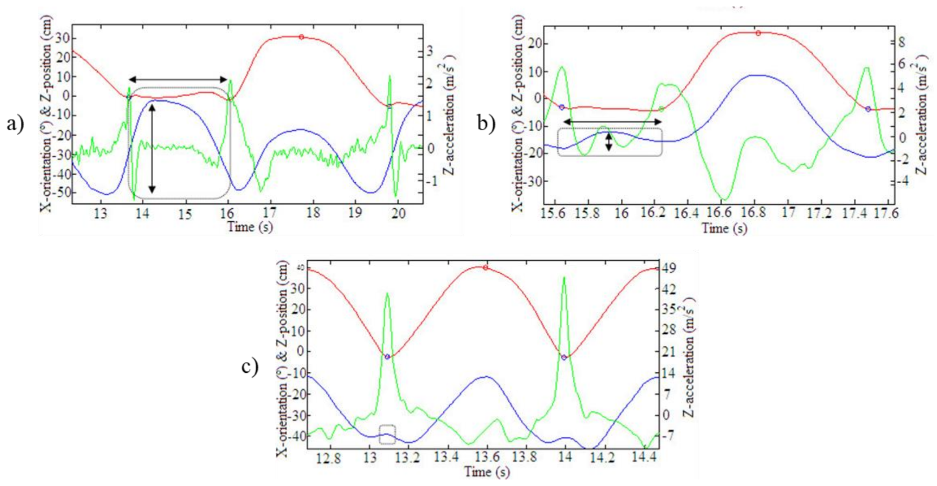


Figure I.17: Movement patterns from raw inertial sensors' signal for frail (a), pre-frail (b), and healthy subjects (c). Z-position, Z-acceleration, and X-orientation are displayed respectively in red, green, and blue. The circle outlines the extra forward and backward lean for more frail subjects and the arrows feature the time duration and X-orientation range.

The evidence presented promotes the utility and clinical applicability of the objective measures of the sit-to-stand through inertial sensors, however, they must be evaluated contextually to the wide variability of results and in the literature [51], [112]. As a matter of fact, by allowing continuous and unsupervised measurements, the outcomes from IMU-based analysis are deeply influenced by the task performed, sensor positioning, and data processing more than any other kinematic methods. Therefore, to develop effective clinical applications, it is necessary to take these factors into account for the identification of movement features related to a pathology or risk factor.

I.4.4 Human activity recognition

The last application of inertial sensors that will be introduced is paradoxically the one on which all the clinical evidence presented so far is based. Infact, the identification of condition-specific characteristics in a movement cannot be performed without first identifying the movement itself. The term Human Activity recognition (HAR) summarises all those techniques aimed at detecting human activity, relying on the information received from different sensors [117] in controlled and uncontrolled environments [118]. Research in this specific field has increased over the last twenty years, focusing on the development of applications for gesture and posture recognition, fall detection, and ambient assisted living [117], [119], [120]. HAR approaches consist generally of four sequential phases [117]: (i) the selection and deployment of the sensor-system, (ii) the data collection, and the (iv) pre-processing and feature selection phase, on which the (v) final inference process is based. This last step requires a decisional model that can be developed by exploiting knowledge of the phenomenon of interest, or by using various machine learning (ML) techniques [118]. Specifically, ML approaches have the advantages of learning directly on the recorded data, being able to handle its variability and inconsistency, and performing better than heuristics-methods based on determined decision rules [121]. The present state-of-the-art HAR methods reached an overall good to excellent identification rate in the IMU-based recognition of the STS movement transitions [Tab.I.4]. Still, wherever the categorisation of the STS into either static or dynamic phases may be acceptable in activity monitoring [120] and robotics implementations [122], it does not adequately model the movement, limiting the possible applications of HAR based clinical evaluation. This can be partially due to the intrinsic variability of the kinematic parameters of the STS, on which the majority of the works in the literature base their algorithms. Surprisingly, just a few studies explore the temporal performance of HAR techniques for the recognition of the STS movement in real-time assessments. Providing a fast, as well as reliable, recognition is fundamental to implement specific skill learning and assessment tasks, where movements need to be evaluated by a system able to give corrective feedback on the performance [133] with interesting applications in physical rehabilitation and sports training [134].

Ref.	Activities	Performance	Sensors	Real-Time
[123]	standing, sleeping, watching TV, walking, running, sweeping, stand-to-sit, sit-to-stand, stand-to-walk, walk-to-stand, lie-to-sit, sit-to-lie	ACC[10]: 95.5 (on sit-to-stand)	1 sensor on the wrist (acc+gyro)	no
[124]	walking, walking-upstairs, walking-downstairs, sitting, standing, lying, stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, lie-to-stand.	ACC: 99.6	Smartphone (acc+gyro)*	no
[125]	stair descent, standing, sitting down, sitting, from sitting to sitting on the ground, sitting on the ground, lying down, lying, from lying to sitting on the ground, standing up, walking, stair ascent	ACC: 83.3-97.7	3 sensors on the trunk and the thighs (acc+gyro)	no
[122]	sitting, standing, sit-to-stand	100	1 sensor on the tigh (acc+ gyro+ mag)	yes
[126] [127]	sit-to-stand	ACC: 92.9 ACC: 99	orthosis system (2 acc+ 2 gyro+2 potentiometers + 2 force sensors)	yes
[128]	sit-to-stand, stand-to-sit	ACC:90.6	1 sensor on the trunk (acc+gyro+bar)	no
[129]	sit-to.stand, stand-to-sit	PPV[11]: 80-100 SEN[12]: 80-100 (on sit-to-stand)	1 sensor on the trunk (acc+gyro)	no
[130]	sit-to-stand, stand-to-sit	ACC: 99 (on sit-to-stand)	1 sensor on the trunk (acc)	no
[131]	stepping, standing, sitting, sit-to-stand, stand-to-sit	ACC: 52 (on sit-to-stand)	1 sensor on the hips (acc)	no
[132]	sit-to-stand, stand-to-sit	ACC: 90-96 (on sit-to-stand)	5 sensors on the trunk, thighs and shanks (acc+gyro+mag)	no
[113]	sit-to-stand	SEN: 93 SPE[13]: 97 (on sit-to-stand)	1 sensor embedded in a pendant (acc+bar)	no

Table 4: Literature overview of the state of the art of human activity recognition methods for the identification of the sit-to-stand motor pattern. acc: accelerometer, bar: barometer, gyro: gyroscope, PPV: Positive Predictive Value, SEN: Sensitivity, SPE: Specitivity, ACC: Accuracy

10 The accuracy is generally defined as the percentage of correctly classified instances.

11 The precision also called positive predictive value (PPV) indicates the number of instances correctly identified as belonging to the positive class divided by the total number of elements identified as the positive class

12 The sensitivity (SEN) also called recall (RCL) measures the proportion of positives that are correctly identified.

13 The specificity (SPE) measures the proportion of negatives that are correctly identified.

I.5 Objectives of the Thesis

This thesis aims at the development of a ML algorithm able to classify the data acquired from a single IMU in accordance with the GRF signals registered from a force plate, for a finer synchronous classification of the STS movement. The use of a single reduced-sized sensor will limit the burden in terms of portability for possible implementation in the rehabilitation field. The major innovation of this work is represented by the sub-categorisation of a movement transition according to a validated clinical-relevant standardization [51]. The dissertation is structured in three chapters:

- In the first chapter, an off-line heuristic method based on GRF signals for the categorisation of the STS is presented together with a structured analysis of its performance against human visual assessments;
- Relying on the acquired evidence, the second chapter introduces and evaluates a simple ML method able to recognise in real-time the STS motion pattern from multiple inertial sensors;
- In the third chapter, more sophisticated ML models are explored to improve the overall accuracy in the STS phase recognition and reducing at the same time the number of sensors needed for the classification.

Chapter 1:

Categorisation of the Sit-To-Stand Motion Pattern. Human vs Automated Quantitative Assessments

1.1 Introduction

This chapter describes in detail the data collection and the subsequent processing aimed at creating the datasets on which the proposed ML routines were based. These two steps are essential as they lay the foundation for the success of any applications based on artificial intelligence. The reason behind this statement is to be found in the very main concepts of machine learning itself. This term defines the field of study which focuses on the development of computer algorithms that improve automatically through experience [135]. ML approaches can be divided mainly into two categories: unsupervised learning methods, which learn directly from the input by modelling undetected patterns in the data [136] and supervised learning methods, which, given a set of paired input-output training samples, learn their reciprocal relationship [137]. In this case, the output is defined by a label, and the method used to match the label with the respective input is called labelisation. Hence, by developing the present research on supervised methods, it is crucial to properly categorise the STS and consequently labelise the dataset to not negatively affect the performance of the final recognition algorithm. However, as already highlighted in the previous chapters, establishing a univocal definition of the STS motion pattern is a quite complex task which still depends too much on visual evaluations to obtain reliable results [51], [71]. Kinematic parameters proved to be strongly affected by the high within-between individuals' variability, limiting the performance of automated techniques to the recognition of dynamic transitions and static positions. On the contrary, kinetic variables seem to offer a finer discretisation of the movement [51] and could be used as ground truth for the data labelisation process. Therefore, to demonstrate the validity of an automated labelisation method, this chapter confronts the result obtained by human visual inspection of the GRF profiles with those obtained by a custom routine for the categorisation of the STS movement pattern.

1.2 Methods

1.2.1 Data Acquisition

Given the homogeneity in population and methodologies, this section will be used as a reference for the entire dissertation, describing the characteristics of the analysed participants and the general experimental protocol followed during the research. Three convenience groups of subjects (HAR1, HAR2, HAR3) have been recruited from students attending university at the Campus of Savona, (Via Magliotto 2, 17100, Savona, Italy). The inclusion criteria for eligible subjects were: good health, absence of musculoskeletal or neurological disorders and, the ability to easily raise from a chair. Each participant had to sign an informed consent. The summarising characteristics for the three groups are presented in Table 1.1. A two-tailed two-sample t-test was used to highlight possible between-group differences. No significant difference was found across the groups' demographics.

	HAR1	**HAR2**	**HAR3**
N° of+ subjects	19	20	20
Age (years)	26 ± 3	26 ±3.	27 ± 4
Gender (M/F)	[9/10]	[13/7]	[11/9]
Weight (kg)	63 ± 10	69 ± 12	67 ± 15
Height (cm)	171 ± 9	173 ±11	172 ± 9

Table 1.1: Demographic characteristics of the recruited groups of subjects

All subjects wore 6 inertial sensors (XSENS Technologies B.V. Enschede, The Netherlands) and performed the STS movement on a force plate (Kistler Winterthur, Switzerland). According to previous studies on accelerometry-based movement analysis [138], [139], sensors were placed on the trunk, the sacrum, upper legs (on the trochanteric area), and lower legs (on the fibular area) [Fig.1.1]. The sample frequencies for the inertial sensors and the force plate were settled respectively at 50 Hz and 1024 Hz.

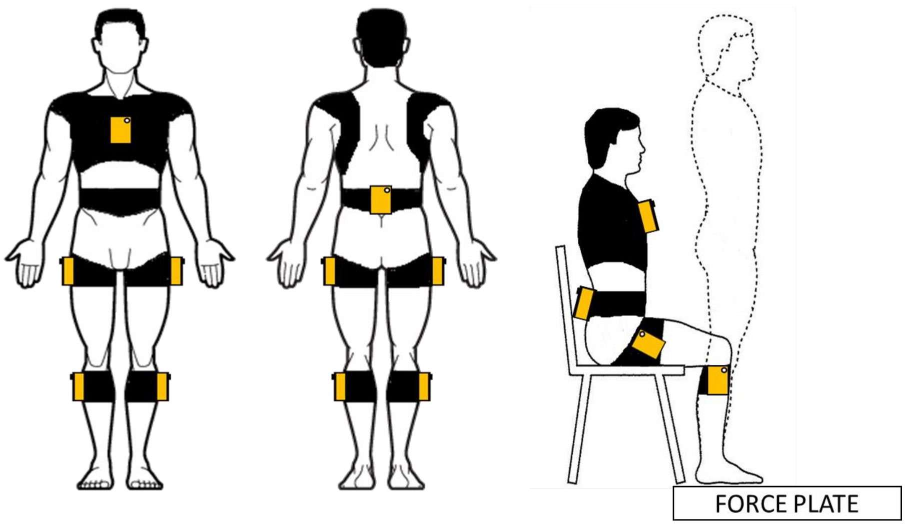


Figure 1.1: Experimental set-up

During the execution of the movement, a custom-made chair equipped with an electronic switch was used to record the time instants of rising (Seat Off) and sitting (Seat On). The chair was height-adjustable, to adapt each acquisition to the variability in the stature of the target population. The signal from the electronic switch was synchronized with the input from the force plate using a DAQ Hardware NIUSB 6343 (National Instruments, Austin, Texas, USA). Participants had to perform two different tasks:

- To perform a set of repetitions of a single STS transition at self-paced speed (SP);
- To perform a set of repetitions of a single STS transition at a controlled speed (CT) with duration marked by a repetitive 4 seconds acoustic feedback, composed by a succession of 3 tones and a pause.

The acoustic feedback was designed to uniform the duration of the individual phases of the STS, identified on the GRF profiles according to the standardization proposed by Etnyre [51]. In SP trials, an initial deflection from the baseline was observed (Initiation). After reaching the lowest level in the force recording (Peak-counter), the GRF raise to a global maximum (Peak) and subsequently levelled to a normal postural sway (Standing). Diversely, CT trials were characterised by a more gradual increase in the GRF following the progressive inclination of the trunk and the raising movement from the chair. Examples of force profiles for both SP and CT trials are displayed in Figure 1.2. Accordingly, the STS movement pattern was categorised in 4 sequential phases [Fig.1.3]:

- The Rest phase (RES), identified as the initial sitting position;
- The Trunk Leaning phase (TLN), starting with the initial deflection of the GRF from the Initiation event to the Seat Off instant;
- The Raising phase (RAI), delimited from the Seat Off to the Standing event;
- The Standing phase (STA), characterised by a stable upright position until the beginning of the sitting movement.

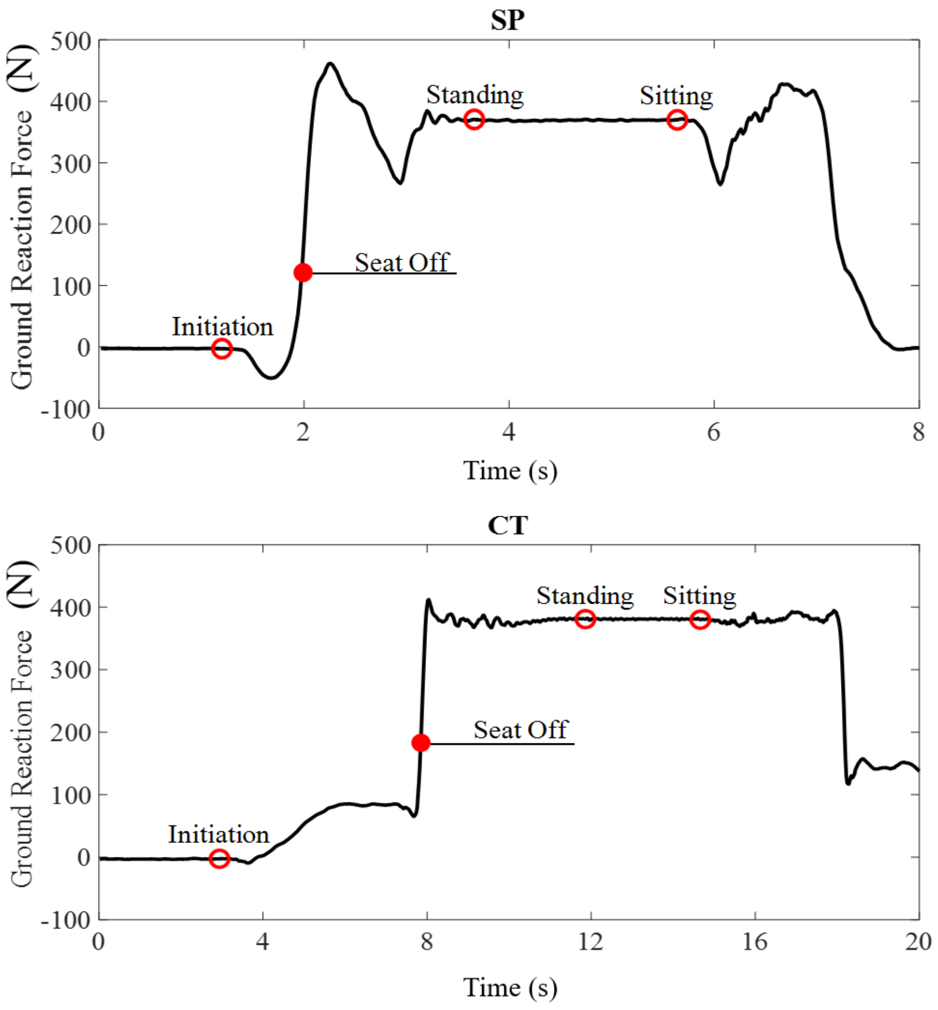
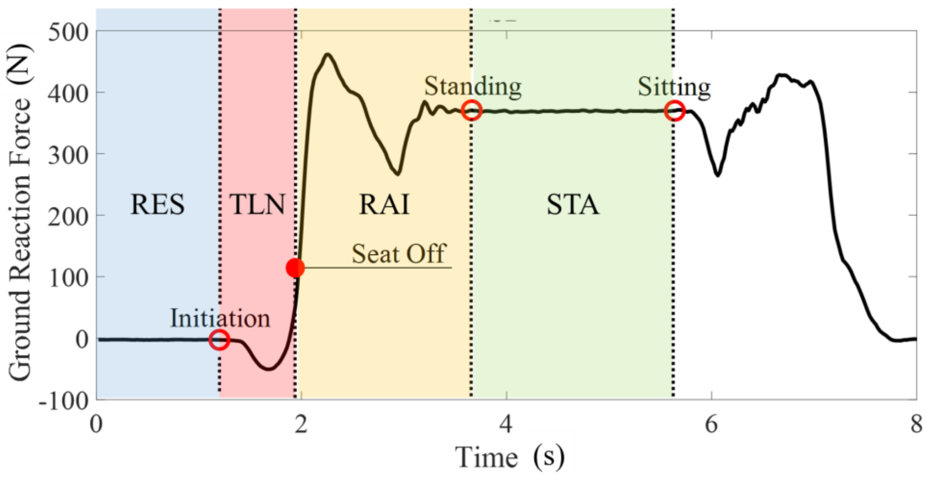


Figure 1.2: Examples of GRF force profiles for SP and CT trials



RES: Resting phase, TLN: Trunk Leaning phase, RAI: Raising phase, STA: Standing phase

Figure 1.3: Categorisation of the STS phases

Throughout the development and progress of this project, the experimental protocol has undergone minor modifications that must be taken considered and disclosed. These changes are due to the progressive methodological improvement of the general research, which relies on previous empirical evidence to reduce possible future bias. The summary of the differences in the data acquisition protocol for the three populations is reported in Table 1.2.

HAR1	• Participants performed 10 SP trials and 10 CT trials; • In SP trials some participants tended to start the movement too early not allowing the registration of an appropriate RES phase; • In SP trials some participants tended to sit-down without having reached a sufficient stable STA phase; • In CT trials participants considered the Stand-to-Sit transition as a single returning phase, starting from the beginning of the Sitting event until reaching the RES phase.
HAR2	• Participants performed 1 SP trials and 2 CT trials for two experimental sessions separately (see Chapter 2.2.2) • HAR1 and HAR2 groups were collected using the same experimental methodologies. However, data from HAR2 was collected through a different routine optimized for simultaneous acquisition of signals from sensors and real-time motion recognition.
HAR3	• Participants performed 10 SP trials and 10 CT trials • In SP trials participants had to wait 3 seconds from the start of the acquisition before starting the movement; • In SP trials participants had to wait 3 seconds in the STA phase to reach balance stability; • In CT trials participants were asked to control the descending movement, identifying two returning phases: a Sitting phase, starting from the beginning of the sitting movement until registration of the “seat-on” signal; a Trunk Raising phase, where subjects raise the trunk until reaching the RES phase. This was done in anticipation of future efforts in categorising the Stand-to-Sit transition.

Table 1.2: Protocol differences between HAR1, HAR2, and HAR3 datasets

1.2.2 Data Processing

To test the labelisation process, the analysis was carried out over the combined HAR1 and HAR3 datasets. The data acquired from the force plate was filtered at 64 Hz using a first-order low-pass Butterworth filter and subsequently resampled at 50 Hz. This step was automatically implemented to match the sampling frequency of the force plate with the one of the IMU sensors, however, in this context, only the signals from the force plate and the electronic switch were considered. Taking into account the possible sources of variability across different datasets, the similarity between movements was assessed using a correlation-based method, by comparing all the registered STS sequences with each other. This was done to evaluate the possible homogeneity of results obtainable from a combined analysis of the HAR1 and HAR3 datasets. For each pair of trials:

- Through a cross-correlation operation, the relative lags that maximised the similarity between the two GRF profiles were found;
- The Pearson's Correlation Coefficient (ρ) between the two STS trials, was calculated after shifting the respective GRF profiles according to the lags found in the first step.

This process was carried out separately for the SP and the CT trials and the results are graphically depicted in the correlograms below [Fig.1.4].

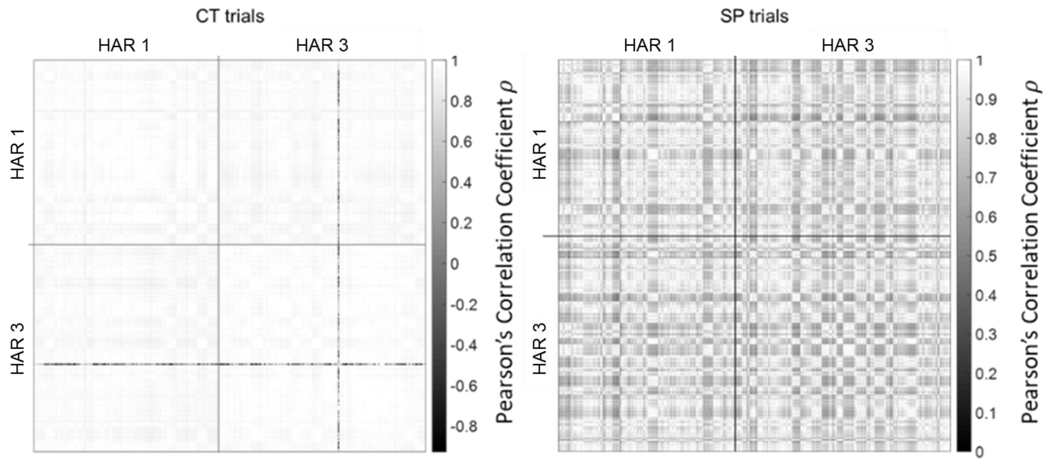


Figure 1.4: Correlograms of the Pearson's Correlation Coefficients for each pairwise comparison between the trials in HAR1 and HAR3.

In CT trials a specific sequence displayed a different pattern compared to all the other STS repetitions. A further exploration evidenced that this trial was characterised by an erroneous offset of the force plate and therefore it was not considered in the subsequent analysis. In general, the graphs highlighted strong correlations (≅1) across all CT trials comparing signals from the same (HAR1 vs HAR1, HAR3 vs HAR3) and different dataset (HAR1 vs HAR3). In SP trials. The correlations were moderate to strong, significantly lower than those obtained in CT trials, with a comparable behaviour for either within and between datasets comparisons. This result was influenced by the unconstrained speed of execution of the movement and the differences in the experimental protocol, however, the similarities between the GRF profile of the trials were sufficiently strong to presume the consistency of the results across the different sequences of the datasets. The categorisation software was implemented in MATLAB (MATLAB R2020a. Natick, Massachusetts: The MathWorks Inc.) and includes three main sub-functions able to identify the significant events in the STS motion pattern.

- The TrunkMOV.m function recognises the Initiation event, as the beginning of the Trunk Leaning phase. In a first step, the GRF profile is segmented into *n* time epochs of N_e samples. Epochs are temporally defined by their middle sample t_i, with *i* =1,…*n*. The software calculates the standard deviation of the force across each epoch as:

$$\sigma_{ep}(t_i) = \sqrt{\frac{1}{N_e - 1} \cdot \sum_{j=1}^{N_e} (F_{j,i} - \overline{F_i})^2} \tag{1.1}$$

where $F_{j,i}$ is the *j*th force sample of the *i*th epoch and $\overline{F_i}$ is the respective mean force. The resulting $\sigma_{ep}(t_i)$ values are then averaged using a moving mean of 10 epochs and normalized over the sequence baseline, B_{TL}:

$$\bar{\sigma}_{ep}(t_i) = E\{\sigma_{ep}(t_{i-5}), \dots, \sigma_{ep}(t_{i+4})\} \cdot \frac{1}{B_{TL}} \tag{1.2}$$

where:

$$B_{TL} = \frac{1}{10} \cdot \sum_{i=1}^{10} \sigma_{ep}(t_i) \tag{1.3}$$

Hence, the initiation event is defined as the first instant t_i that satisfies the condition:

$$\bar{\sigma}_{ep}(t_i) > T_{TL} \tag{1.4}$$

where T_{TL} is a user-defined threshold.

● The Bottom_Transition.m identifies voltage transitions of the electronic switch, from 0V to -5V and vice versa using a differential mask procedure.

● The Standing phase is recognised by retrieving the information from the electronic switch and by identifying the stable stance between the Seat Off and the Seat On events. The SteadyStandingPoints.m function recognise the balance stability in the upright stance, by identifying the Standing and Sitting events. Firstly, the force signal is trimmed and divided into n time epochs of N_e samples between the Seat Off and the Seat On instants. Subsequently, the standard deviation across all the epochs is calculated using equation (1.1). Relying on the symmetrical shape of the GRF profile, the middle epoch t_{mid} of the resulting signal is considered as the centre of the stable stance. Hence the stability baseline B_{ST} is calculated around t_{mid} as:

$$B_{ST} = E\{\sigma_{ep}(t_{mid-10}), \dots, \sigma_{ep}(t_{mid+10})\} \tag{1.5}$$

In a similar way to that described in equation (1.3) the B_{ST} value is used to normalise the moving average of $\sigma_{ep}(t)$. The resulting $\bar{\sigma}_{ep}(t)$ is used to identify the beginning and the ending instants of the stable stance as respectively the first and the last epochs that satisfy the condition:

$$\bar{\sigma}_{ep}(t_i) < T_{ST} \tag{1.6}$$

Where T_{ST} is a user-defined threshold. The default values for the described variables used in this evaluation are reported in Table 1.3. For the purpose of the present analysis T_{TL} and T_{ST} were selected empirically by tuning the values according to the results obtained from the STS profiles. Under this consideration, 2 and 1.5 were the values that assured the best categorisation of the STS across the majority of the force profiles.

	TrunkMOV.m	**SteadyStandingPoints.m**
GRF sample frequency	50 Hz	50 Hz
$\boldsymbol{Ne}$	5 epochs (0.1 s)	5 epochs (0.1 s)
$\boldsymbol{T_{TL}}$	2	N.A.
$\boldsymbol{T_{ST}}$	N.A.	1.5

Table 1.3: Default values of the categorisation software used for this study.

1.2.3 Data Analysis

In the lack of a proper gold standard approach, visual inspection remains the most reliable method to recognise STS events and phases based on GRF values [51]. Hence, simultaneously with the software analysis, five participants aged (30 ± 6) years visually identified the beginning of the trunk movement (Initiation event) and the limits of the stable stance (Standing and Sitting events) from the force profiles of 200 STS sequences. The ages of the assessors are reported in Table 8 together with their professional and academic background. As a measure of reliability, assessments were repeated in 2 distinct sessions, separated in time by a minimum interval of 1 hour. STS sequences were drawn randomly from a total pool of 742 acquisitions (HAR1+HAR3) and were maintained across the measurement trials. This information was not made explicit to avoid possible learning effects. Among the 200 pooled sequences, 100 referred to SP trials, and 100 referred to CT trials. Before the first session, subjects were briefly trained to recognise the onset of each event on five force profiles accordingly to the definitions given by Etnyre and colleagues, and an explanatory summary was always available in the form of MATLAB live script during all the measurements.

AGE	PROFESSIONAL BACKGROUND	ACADEMIC LEVEL
29	Bioengineer	MSc/Ph.D. student
27	Physicist	MSc/Ph.D. student
40	Physiotherapist	Ph.D.
25	Bioengineer	MSc/Research Fellow
30	Psychologist	MSc/Ph.D. student

Table 1.4: Assessors' description

All assessors were physiotherapists, bioengineers, and Ph.D. candidates with clinical experience in physiotherapy and expertise in movement analysis. Test-Retest reliability was considered as a quality index of the human measure, and it was compared with the performance of the software evaluated with the Inter-Rater reliability against participants' assessments in the first trial. For both analyses, the absolute agreement was measured using a non-parametric version of Bland-Altman statistics [140], as a consequence of the violation of the normality assumption of data. Normal distributions were tested using the Kolmogorov-Smirnov test and, furtherly investigated using the skewness and kurtosis indexes. To correct the analysis for possible outliers not directly connected to the evaluation of the assessors or the performance of the software, data observations that fell outside 1.5 interquartile ranges above the upper quartile (75th percentile) or below the lower quartile (25th percentile), were visually inspected. In particular, wrong identifications resulting from an erroneous mouse click, early movements, or a malfunction of the electronic switch were eliminated. The systematic bias and the limits of agreement was calculated as the median of the absolute differences between assessments and the respective 2.5th and 97.5th percentile scores. The upper Limits of Agreement (ULoA) represented the maximum estimated error between the measures. The 95% Confidence Intervals (CI) of the above-specified parameters were also calculated using a percentile bootstrap method based on 10k samples [141]. The percentile method was chosen for its conservative nature, as it tends to produce wider CI less sensitive to population value and sample size [142]. A two-tailed two-sample t-test [143] was used to compare the maximum errors made between the Test-Retest trials and the Human-Software evaluations, exploring the significant differences between measures. Every ULoAs and respective CI were approximated to normal distributions characterised by mean values μ_i and standard deviation σ_i [144], calculated as:

$$\mu_i = ULoA_i \tag{1.7}$$

$$\sigma_i = \begin{cases} \dfrac{|CI_{up} - (ULoA)_i|}{1,96} \cdot \sqrt{N} & if \quad |CI_{up} - (ULoA)_i| > |CI_{low} - (ULoA)_i| \\ \dfrac{|CI_{low} - (ULoA)_i|}{1,96} \cdot \sqrt{N} & if \quad |CI_{low} - (ULoA)_i| > |CI_{up} - (ULoA)_i| \end{cases} \tag{1.8}$$

The choice of the lower or higher confidence limit was guided by the need to calculate the largest standard error to obtain a more conservative approximation. The statistical analysis was stratified by considering the different STS events separately, dividing the results obtained in SP and CT trials. Bland-Altman statistics was implemented in MATLAB as a modified version of the BlandAltman.m function developed by Ran Klein from the Department of Nuclear Medicine of the Ottawa Hospital [145]. The two-tailed two-sample t-test was executed with the online “Comparison of means calculator” tool from Medcalc Statistical Software [146].

1.3 Results

The Bland-Altman plots related to the identification of the Initiation, Standing and Sitting events are shown respectively in Figure 1.5, Figure 1.6, and Figure 1.7 and the summary of the descriptive statistics for the bias and the ULoAs (CI) are reported in Table 1.5. In both SP and CT trials, for all STS events, participants showed significantly lower identification bias in the repeated assessments, compared to the difference displayed against the measurements performed by the software. Moreover, the Test-Retest Reliability in the identification of the STS events significantly decreases in CT trials with a consequent increase in the value of the systematic error.

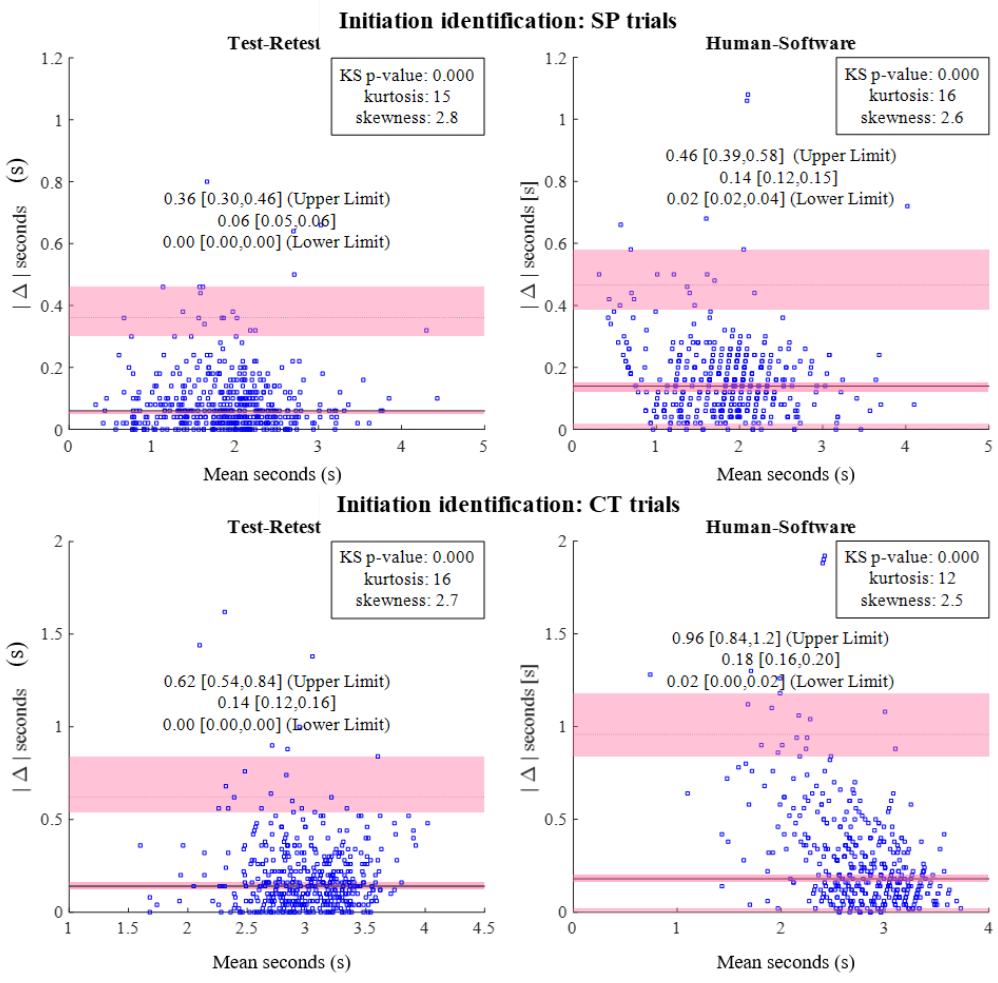


Figure 1.5: Bland-Altman plots depicting the Test-Retest Reliability (left) and the Inter-Rater Reliability (right) of the identification of the Initiation event in SP trials (upper plots) and CT trials (lower plots).

Events	SP trials						CT trials					
	Test-Retest			Human-Software			Test-Retest			Human-Software		
	Bias (s)	ULoA (s)	N	Bias (s)	ULoA (s)	N	Bias (s)	ULoA (s)	N	Bias (s)	ULoA (s)	N
Initiation	0.06 [0.05-0.06]	0.36 [0.30-0.46]	492	0.14 [0.12-0.15]	0.46 [0.39-0.58]	490	0.14 [0.12-0.16]	0.62 [0.54-0.84]	483	0.18 [0.16-0.20]	0.96 [0.84-1.20]	493
Standing	0.10 [0.08-0.10]	0.62 [0.48-0.74]	488	0.30 [0.28-0.32]	0.82 [0.74-0.86]	495	0.16 [0.16-0.20]	1.30 [1.10-1.70]	484	0.44 [0.40 0.48]	1.60 [1.40-1.70]	493
Sitting	0.08 [0.06-0.08]	0.42 [0.34-0.47]	497	0.20 [0.18-0.22]	0.48 [0.46-0.60]	494	0.14 [0.12-0.14]	0.69 [0.60-0.86]	491	0.20 [0.18-0.22]	0.81 [0.66-0.86]	490

Table 1.5: Descriptive statistics for Bias and ULoA for every comparison. (*) p<0.05 for not overlapping CI

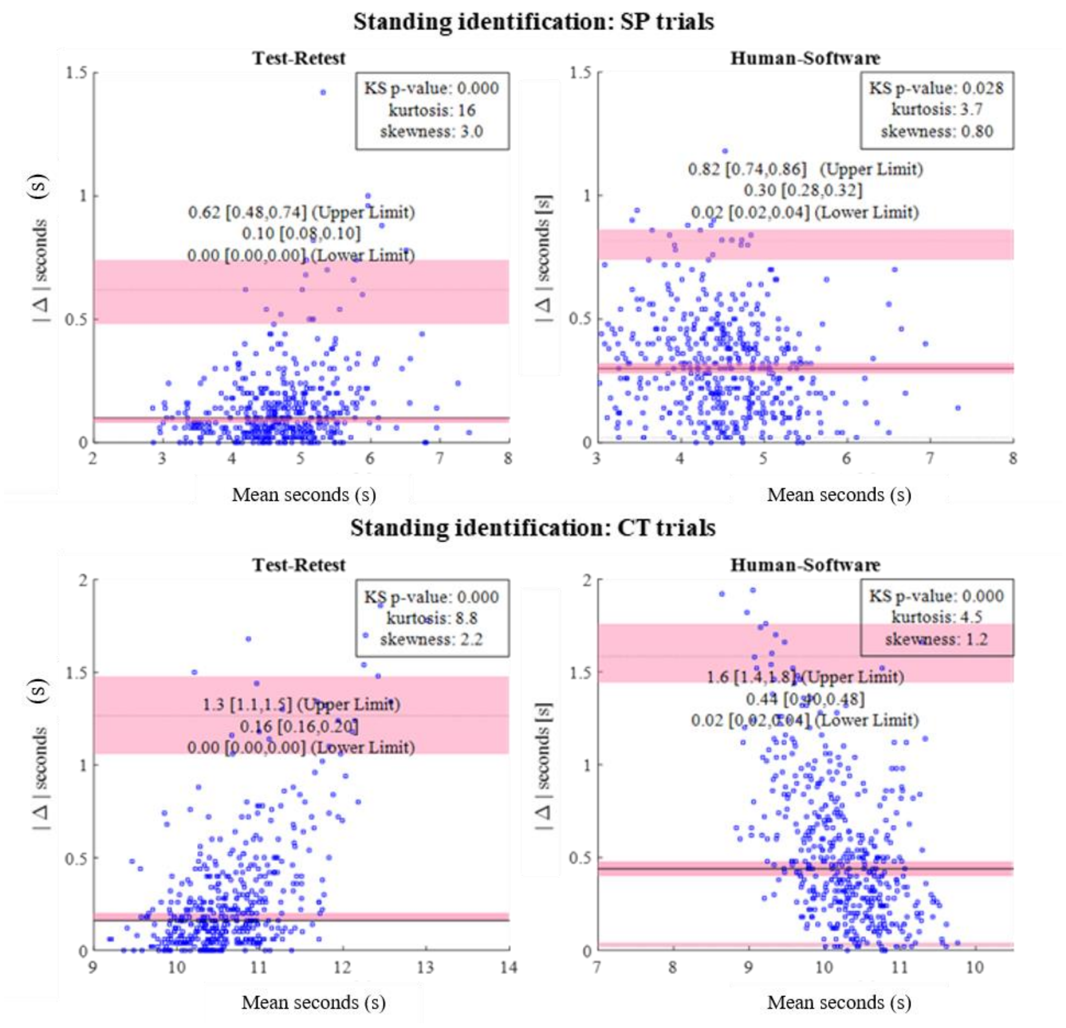


Figure 1.6: Bland-Altman plots depicting the Test-Retest Reliability (left) and the Inter-Rater Reliability (right) of the identification of the Standing event in SP trials (upper plots) and CT trials (lower plots).

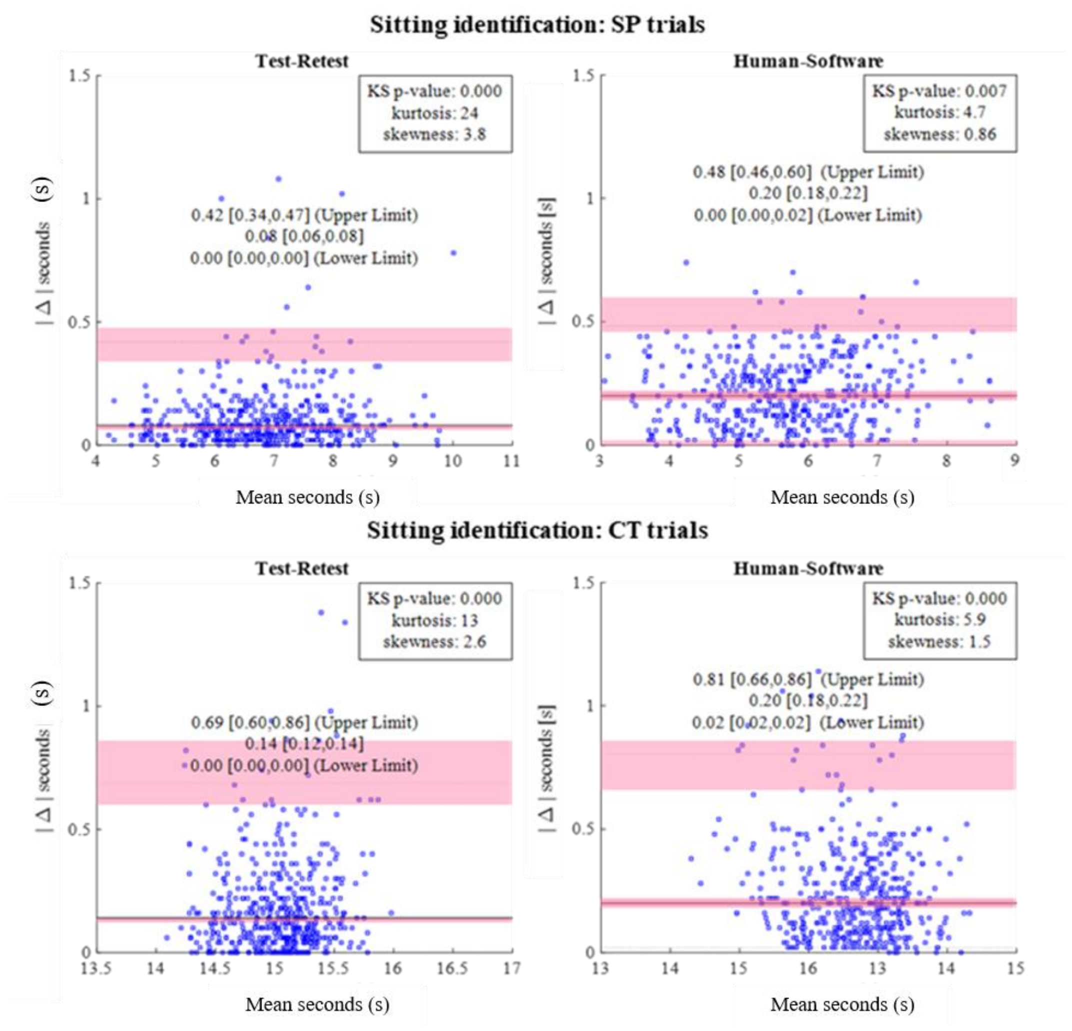


Figure 1.7: Bland-Altman plots depicting the Test-Retest Reliability (left) and the Inter-Rater Reliability (right) of the identification of the Sitting event in SP trials (upper plots) and CT trials (lower plots).

The results of the two-tailed two-sample t-test are summarised in Table 1.6.

	SP trials			CT trials		
	Initiation	**Standing**	**Sitting**	**Initiation**	**Standing**	**Sitting**
Difference (s)	0.100	0.200	0.060	0.340	0.300	0.120
95% CI (s)	0.056	0.039	-0.084	0.014	0.017	-0.107
	0.256	0.361	0.204	0.666	0.583	0.347
t-statistic	1.255	2.440	0.816	2.045	2.079	1.037
df	980	981	989	974	975	979
p-value	0.209	0.015 **(*)**	0.414	0.041 **(*)**	0.038 **(*)**	0.300

Table 1.6: Two-tailed two-sample t-test results. Differences with p-values <0.05 (*) were considered statistically significant

Significant differences in ULoAs between Test-Retest evaluation and Human-Software assessments were found in the Standing event identification for both SP (0.200 s [0.039; 0.361], p>0.05) and CT (0.300 s [0.017; 0.583], p>0.05) trials. A significant difference ($p>0.05$) of 0.340 s [0.014; 0.666] was also found in the Initiation event identification in CT trials.

1.4 Discussion and conclusions

This part of the project aimed at evaluating the performance of a custom-routine for the recognition of clinically relevant events in the STS from GRF profiles, comparing the results of the automated approach to human visual assessment. Despite the significantly lower systematic bias in human evaluations, the comparison between visual assessments and the proposed approach showed similar values of maximum absolute error, supporting the use of the presented method for the automated categorisation of the STS movement. More specifically no statistical differences were found in the identification of the Initiation and the Sitting events in SP trials and the identification of only the Sitting event in CT trials. The worsening of the observed agreement during CT movements was generally in line with our expectation, as ULoAs values could be affected by uncertainties due to the kinetic modifications resulting from the standardisation of the movement. For instance, the initial GRF deflection effect is highly dependent on each individual's movement strategy [51] and could be reduced by the lower quantity of momentum produced under constrained speed, complicating the identification of the Initiation event. Another important consideration highlights the intrinsic subjectiveness of human evaluations [147]. One could consider the slightest oscillation either as an extension of a contiguous static phase or as the limit of a movement transition. Moreover, whereas visual assessments can vary across repeated evaluation and differ in individuals, depending on their professional experience [148], [149], the use of an automated procedure ensures the repeatability of the assessments. Nonetheless, it is important to underline that, with an estimated maximal discrepancy of 0.200 s [0.039; 0.361] and 0.340 s [0.014; 0.666] respectively for normal and standardised speed, the proposed algorithm should not be considered as a replacement to visual clinical evaluations on a single patient, where the specific expertise of clinicians plays a key role in the diagnosis process, often requiring a high level of abstraction [147]. Indeed, such a method should be seen as a help to health professionals as it could be able to free them from the encumbrance of the data processing, with a reasonable margin of error. Previous works [147]–[151] evaluated the performances of various algorithms developed for the identification of Sit-to-Stand and Stand-to-Sit postural transitions using data acquired from inertial sensors. In particular, a recent paper of Atrsaei and colleagues [129] validated the accuracy of a new routine based on a single inertial device against visual assessments on-camera recordings of STS movements, obtaining levels of agreement above 94%, in terms of positive predictive values and sensitivity. As a direct comparison with the present study, the use of inertial sensors is usually preferable since they can be also applied in non-clinical environments [46]. However, their measurements are strongly influenced by the inter/intra-individual variability of the movement [51], [152], and the positioning of the sensors, limiting the recognition of the STS motion pattern to the simple discrimination of static and dynamic phases. Conversely, our choice to use a force plate has some doubtless limitations in terms of costs and portability but the strong value of providing easier interpretable results, on which it is possible to identify clinically significant movement events and phases. This major advantage can be exploited by ML algorithms, as valid ground-truth data to train specific supervised approaches in a finer recognition of the STS for HAR tasks [153], [154]. In this context, the results obtained offered a double contribution to the general work and prospective researches. First, we evaluated the human-level performance in the described recognition task, giving an estimate of the considerable reference value in terms of "optimal error" [155]. Second, we quantified the discrepancies between the categorisation software and human assessment, obtaining an overall good-agreement in the identification of significant STS events from values of vertical GRF. By providing objective measures, this method can be used to identify specific trends and patterns in the STS movement, which can be ultimately exploited for a reliable data labelisation in ML applications. With the support of such objective reference, the use of co-registration methods between different sensor modalities would allow the development of accessible, wearable rehabilitation tools, that combine the discriminating power of gold-standard instruments with the limited dimensions and costs of inertial sensors.

Chapter 2:

Real-Time Validation of Neural Networks for a Clinically Relevant Recognition of the Sit-to-Stand Movement

2.1 Introduction

This chapter takes up some notions previously introduced to clearly explain the basic principles of machine learning and explore the potential of a simple neural network for synchronously recognise the STS movement pattern.

2.1.1 A brief history of neural networks

The Neural Network concept is relatively old, dating back to the 1940s with the studies of Warren McCulloch and Walter Pitts [156]. In their work, they described the theoretical model of the nervous system as a set of nodes that are connected through synapses. If a node (representing a neuron) receives a discrete amount of input from the neighbouring nodes, surpassing a definite threshold, then it will initiate an impulse and the signal will propagate across the network. This idea has been applied later by Frank Rosenblatt [157] in the development of a supervised binary classifier[14]: the Perceptron.

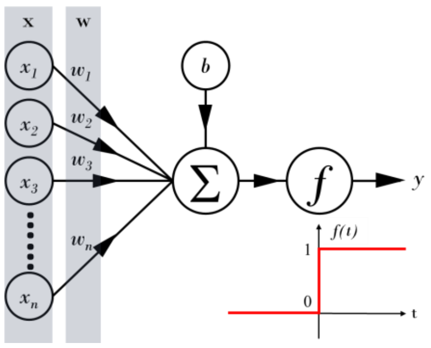


Figure 2.1: The Perceptron

[14] There are two type of predictive models:

- Classification models approximates a mapping function (f) from input variables (x) to discrete output variables (y) called labels or categories.
- Regression models approximates a mapping function (f) from input variables (x) to a real-value continuous output variable (y)

Figure 2.2: The Mark I Perceptron machine, The system was connected to a 20×20 cadmium sulfide photocells based camera to make a 400-pixel image (on the left). The inputs were randomly connected with wires (center panel) to an array of potentiometers that implemented the network weights (on the right)

The Perceptron is the simplest neural network architecture, it takes multiple inputs ($\mathbf{x}$) and computes a weighted sum accordingly to the vector ($\mathbf{w}$) of the weights. A non-linear activation function (in this case, a step-function *f*), is later applied to the weighted sum to produce the final output (*y*) [Fig.2.1].

$$f(x) = \begin{cases} 1, & if\ \mathrm{w} \cdot \mathrm{x} + b > 0 \\ 0, & otherwise \end{cases} \tag{2.1}$$

Although initial expectations were high, Perceptron could not be generalised to multinomial classifications. Furthermore, being based on linear combinations of fixed basic functions, the model could not solve non-linearly separable problems (i.e. learning the boolean function "XOR"[15]). These limitations, valid for single-layer networks, were erroneously generalized even to more complex architectures, leading the scientific community to underestimate the potential of neural networks with a consequent stall in research [158]. It is just in 1986 with the publishing of the backpropagation algorithm [159], a methodology for training[16] more complex architectures, that the interest in neural network revived, laying the bases for the later “deep-learning” expansion.

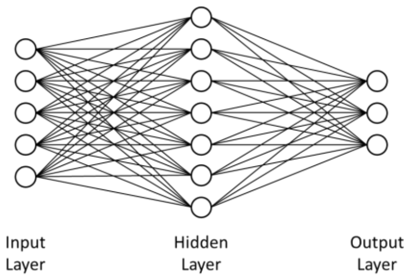


Figure 2.3: General structure of a simple feed-forward neural network

[15] The logical operator XOR outputs TRUE just when the inputs are differs between each other
[16] Training implies providing a model a set of data to learn from.The learning process allow the model to map the input intrinsic patterns to some outputs.

As the Perceptron can be considered a model for a single neuron, a neural network can be thought of as the connection of more Perceptrons and it is often schematised as displayed in figure 2.3. In the most general conception a network structure can be characterised by three types of layers [160], [161]:

- the input layer whose neurons encode a fixed-size input signal (i.e. an image 256x256 pixel);
- one (or more) hidden layer(s), which connects inputs and outputs;
- the output layer whose neurons represent the outputs of the model, for example the number of categories in a classification problem.

The complexity of a network is related to the depth of its layers, a “shallow” network is composed of a maximum of three layers including the inputs and the outputs, while a “deep” network has more than one hidden layer [162]. Finally, according to the direction of the connections networks can be categorised in *feed-forward*, if the information is fed uniquely from the inputs to the output, or *recurrent* if there are loops in the architecture to maintain a memory of the previous states (see Chapter 3) [160].

2.1.2 Multilayer Perceptron

Following the previous statements, the Multilayer Perceptron (MLP) is a feed-forward network that can be thought of as a composition of many Perceptron units organized into layers. However, the term “Perceptron” can be deceiving because the choice of the activation function for the single units is not limited to the step function but can be arbitrarly defined with real values outputs (usually ranging between 0 and 1or between -1 and 1), allowing probability-based classifications. The general functioning of a MLP can be summarised by the equations presented in Figure 2.4:

$$z_j = h(\sum_{i=1}^{M} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)})$$
$$y_k = g(\sum_{j=1}^{N} w_{k,j}^{(2)} z_j + w_{k,0}^{(2)})$$

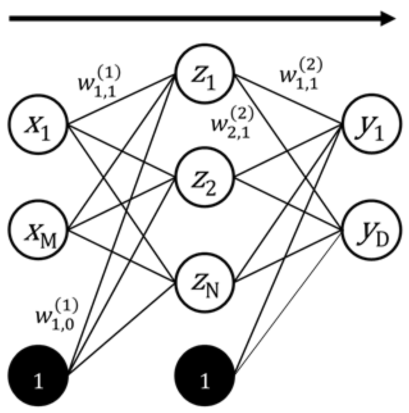


Figure 2.4: On the left the characteristic equation of a shallow MLP, the graphical representation of a shallow MLP network. Each layer is charachterised by the relative equation on the left. Biases elements are reported in black.

Where i = 1,...,M, j=1,...,N and k =1,...,D. The superscript (l) indicates that the corresponding parameters are relatives to the lth layer of the network. The first subscript indicates the destination unit of the connection in the lth layer while the second subscript is referred to the origin unit of the connection in the l-1th layer. In this way the parameters $w_{a,b}^{(l)}$ define the weights of the connections to the lth layer of the architecture and the parameters $w_{a,0}^{(l)}$ define the biases[17] of the lth layer. The activation functions should be chosen accordingly to the nature of the data and the specific task. For instance, for regression problems, the activation function is the identity so that $y_k = \sum_{j=1}^{N} w_{k,j}^{(2)} z_j + w_{k,0}^{(2)}$. and for multiclass problems a softmax activation function in the form of

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}. \qquad (2.2)$$

is used to obtain the probability of each ith choice among a total number of C possible classes. From this point of view, the MLP can be seen as a nonlinear function that maps a set of input variables X to some output variables Y controlled through a vector W of adjustable parameters. Hence starting from a set of training data, the learning procedure aims at optimising the weights of the connections to reduce the difference between errors committed in output compared to the expected results.

[17] Biases are simple constant value added to the weighthed sum of the afferent connections to a node in the subsequent layer. They are used to offset the output of the activation function.

3.1.3 The Backpropagation learning rule: from biological to artificial neural networks

The learning process described above recalls the mechanisms of neural plasticity, which is considered at the base of learning and memory function in the brain, and consist of the ability of biological Neural Networks and synapses to modify their properties depending on their activity [163], [164]. More specifically, synaptic plasticity is commonly referred to as the strengthening or weakening of synaptic weights and can be categorised into Short-Term Synaptic Plasticity (STSP) and Long-Term Synaptic Plasticity (LTSP) according to the duration of its effect. STSP lasts several minutes [165]. The STSP can manifest either as an enhancement or a depression. Among the enhancing effects, the two main typologies are the Paired Pulse Facilitation (PPF) and the Post Tetanic Potentiation (PTP) [164]. The PPF occurs when two action potentials arrive in short succession within tens of milliseconds due to the increased level of Ca^{++}, which allows a larger release of neurotransmitters in the presynaptic cell, resulting in an excitatory potential. PTP is characterised by a larger temporal window, occurring after a high-frequency train of presynaptic action potentials in association with Ca^{++} dependent protein kinases [166]. After PPF, STSP depression can occur due to the depletion of releasable pools of neurotransmitter vesicles. LTSP retains its effect for many hours (or longer) [167] with either enhancing (Long-Term Potentiation - LTP) or depowering synaptic effects (Long-Term Depression - LTD). The molecular mechanisms of LTP and LTD differ among the various types of synapsis and have been observed in numerous cerebral areas like the hippocampus, the cortex, and the cerebellum [163], [167]–[169]. The NMDA postsynaptic receptors[18] play a key role in LTP and LTD. These receptors can be activated by the neurotransmitter glutamate from the presynaptic cell, glycine and D-serine [163], [171], enabling the stream of Ca^{++} in the postsynaptic cell, which causes more AMPA receptors to be inserted into the postsynaptic membrane, with the consequent strengthening of the synaptic

[18] NMDA and AMPA receptors belong to a class of ionotropic glutamate receptors. Excitatory synaptic transmission in the brain is based on the release of L-glutamate from presynaptic terminals that diffuses across the synapsis and binds to postsynaptic AMPA and NMDA receptors. The activation of AMPA receptors is fast and transient, while NMDA receptors regulate functional and structural plasticity of individual synapses, dendrites, and neurons by allowing activation of specific calcium-dependent signaling cascades. Several unique properties of NMDA receptors prevent their activation by L-glutamate released during a single synaptic event [170]

transmission (LTP). Conversely, a lower influx of Ca^{++} provokes the internalization of AMPA receptors, weakening the synaptic transmission (LTD). Following a process akin to the biological counterpart, Artificial Neural Networks update the weights of their connections iteratively according to the feedback received by comparing their outputs with some reference target. This process is commonly carried out by the Backpropagation algorithm [159]. The method is divided into two main steps:

- the forward propagation of the input through the network to find the activations of all the hidden and output units;
- the backpropagation of the error from the outputs to the inputs to find its partial derivatives respective to all the weights and the bias of the network;

The forward propagation is completely defined by the previously introduced equations, which are reported and expanded more in-depth below [160], [172].

$a_i^{(0)} = x_i, \quad \forall i \in \mathbb{Z} \mid 1 \leq i \leq N^{input}$	Input units	(2.3)
$z_j^{(l)} = \sum_{i=1}^{M} w_{j,i}^{(l)} a_i^{(l-1)} + w_{j,0}^{(l)}$	Weighted Sum at the jth unit in the lth layer	(2.4)
$a_i^{(l)} = f(z_i^{(l)})$	Activations of the jth unit in the lth layer	(2.5)

$$\forall i \in \mathbb{Z} \mid 1 \leq i \leq N^{(l-1)}$$
$$\forall j \in \mathbb{Z} \mid 1 \leq j \leq N^{(l)}$$
$$\forall l \in \mathbb{Z} \mid 1 \leq l \leq L$$

Table 2.1: Forward propagation. The superscript (l) indicates the layer, L is the total number of layers, the subscript j refers to the target unit in the lth layer, the subscript i refers to the origin unit in the l-1th layer, The parameter $w_{j,0}^{(l)}$ is the bias term. In the output layer the activations are the outputs of the network.

The back propagation step starts with the calculation of the output error which is characterised by the cost function $J(\mathrm{W})$, where W is the vectorisation of all the weights $w_{j,i}^{(l)}$ of the network. Subsequently the partial derivatives of the cost function $\frac{\partial J}{\partial w_{j,i}^{(l)}}$ are calculated using a mathematical method called the chain rule. Every $\frac{\partial J}{\partial w_{j,i}^{(l)}}$ retain information on how much every single weight $w_{j,i}^{(l)}$ contribute to the overall output error, and are actually used to update the parameters of the network. Below are the equations that describe back propagation in detail [160]:

$\delta_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$	Error associated with the jth node of the Lth layer	(2.6)
$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \sum_{k=1}^{K} \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{K} w_{k,j}^{(l+1)} f'(z_j^{(l)}) \delta_k^{(l+1)}$	Error associated with the jth node of the lth layer	(2.7)
$\frac{\partial J}{\partial w_{j,i}^{(l)}} = a_i^{(l-1)} \delta_j^{(l)}$	Partial Derivatives calculation	(2.8)
$w_{j,i}^{(l)} = w_{j,i}^{(l)} - \lambda \frac{\partial J}{\partial w_{j,i}^{(l)}}$	Weight Update	(2.9)

$$\forall i \in \mathbb{Z} \mid 0 \le i \le N^{(l-1)}$$
$$\forall j \in \mathbb{Z} \mid 1 \le j \le N^{(l)}$$
$$\forall k \in \mathbb{Z} \mid 1 \le j \le N^{(l+1)}$$
$$\forall l \in \mathbb{Z} \mid 1 \le l \le L$$

Table 2.2: Backward propagation. The superscript l indicates the layer, L is the total number of layers, the subscript j refers to the target/origin unit in the lth layer, the subscript i refers to the origin unit in the l-1th layer, if equals to 0 refers to the bias connection, the subscript i refers to the target unit in the l+1th layer

The algorithm starts with a randomised set of initial weight W_0 and stops when the cost function reaches a sufficiently small value after a definite number of iterates $\{W_K\}_{K=0}^{N}$, essentially categorising the learning process as an optimisation problem, where a local minimum of the function $J(W)$ is sought [173].

Iterative techniques for unconstrained non-linear optimisation [19]can be categorised in two major classes: line search methods and trust-region methods [174]. Line search strategies search the data space along a direction p_k, by moving with a step α to find a new iterate W_{k+1}which decrease the cost function.

$$\min_{\alpha>0} J(W_K + \alpha p_k)$$

Following a dual concept, trust-region strategies reasonably approximate $J(W)$ with a simpler function $J_*(W_K)$ in a neighbourhood surrounding α, and minimise it by searching for an appropriate trial step p inside the so defined trust region. W_k is consequently updated as $W_k + p$ if $J(W_K + p) < J(W_K)$. If the condition is not satisfied, a smaller trust region is considered, and a new solution $W_k + p$ is evaluated.

$$\min_{p} J_k(W_K + p)$$

[19] Unconstrained minimization is the problem of finding a vector x that represent a local minimum of an objective function f(x)

$$\min_{x} f(x)$$

Where $x \in \mathbb{R}^n, n \geq 1$ and $f: \mathbb{R}^n \rightarrow \mathbb{R}$.
The term unconstrained means that no restriction is placed on the range of x.

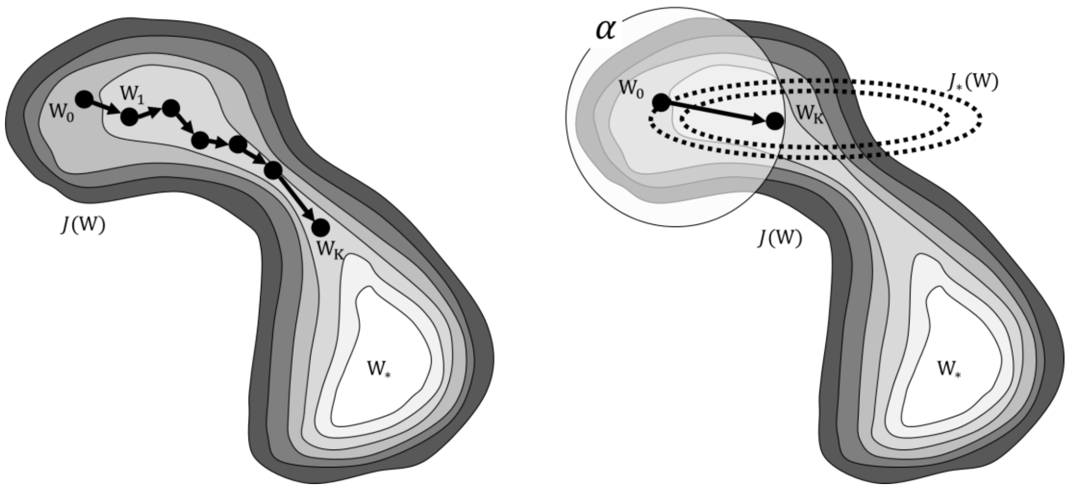


Figure 2.5: On the left: line search strategies search the data space along a defined direction and move along it by a defined step. On the right: trust region strategies approximate J(W) with a simpler function J (W) in a neighbourhood surrounding α, and minimise it inside the so defined trust region*

The basic Backpropagation approach applies a line search method, by finding the direction where the function J decreases more rapidly (i.e. gradient descent), however, it is important to underline that different variants of the algorithm exist. These alternative formulations can implement both line search and trust region strategies with advantages in terms of stability and/or speed of convergence.

2.1.3 Objective

By exploiting the aforementioned concepts, this chapter discusses the development of two shallow MLPs and evaluates their classification performance in the real-time recognition of the STS movement pattern.

2.2 Methods

This section is divided into two parts: the development of the MLPs (design phase) and their implementation in the real-time recognition of the STS movement (validation phase).

2.2.1 Design phase

The MLPs were trained on the data collected from the HAR1 population following the methodologies described in Chapter 1.2.1 for the STS registration. Concerning the specific supervised classification task, it is important to underline that the execution of the STS movement implies variable durations for the phases of its motion pattern (RES, TLN, RAI, STA see Chapter 1.2.1) resulting in an unequal distribution of occurrences in the different output classes. This inhomogeneity of the dataset can affect the final performance of the algorithm in multiple ways [175]:

- in imbalanced conditions, classifiers models provide suboptimal results, with a distortion in the overall performance pointing toward the most represented examples;
- performance metrics such as prediction accuracy are prone to bias in favour of the majority class;
- minority classes can be coded as noise and vice versa by the prediction algorithm since both are rare patterns in the data space;
- minority classes could overlap with another class in the same data space and with a similar number of samples, making their a priori distinction difficult;
- in general, small sample size and high dimensionality in the data space usually cause a failure in the detection of under-represented patterns;

To conveniently handle these problems, the two classifiers were trained on the CT dataset, exploiting the role of the acoustic feedback in standardising the duration of the STS phases. The two classifiers were trained on the inertial signals, composed by 3D accelerations, Roll, Pitch and Yaw values. An epoch of 5 samples from the inertial signal (50 Hz – 20 ms) was used to obtain a valid sample of features for classification at 10 Hz (100 ms). This sampling rate was selected to simulate the occurrence of a real-time human movement signal, which is characterised by spectral components below 20 Hz [176]. For instance the typical kinematic bandwidth of normal gait is between 4 and 6 Hz [177] and it has been demonstrated that the frequency range of ADLs, performed on a force platform, range between 0.3 and 3.5 Hz [178]. Hence, from each signal epoch, 7 temporal features were extracted: mean, root-mean-square (RMS)[20], maximum normalised to RMS, minimum normalised to RMS, standard deviation (SD), coefficient of variation (COV)[21] , and Jerk[22], calculated as the first derivative of the acceleration. The resulting dataset was subsequently segmented and labelled, based on the time events obtained from the GRF and the electronic switch using the labelisation software described in Chapter 1.2.2. During the execution of the automated labeling process of the dataset, data was visually investigated to exclude eventual inconsistencies due to acquisition errors. A supervised approach was then implemented in MATLAB to develop 2 separate models of MLPs, respectively trained over the data acquired from all the IMUs (see Chapter 1.2.1) or from just three sensors (situated on the trunk, and the upper legs). For each network, 100 possible alternative models were identified by defining 10 different topologies (number of hidden units) and 10 different starting sets of initial weights. As suggested by Heaton [180], the numbers of hidden neurons were chosen between the number of output units (4 phases of the movement) and the number of input units (252 features from 6-sensors, 126 features from 3 sensors). The scaled conjugate gradient (SCG) variation of the backpropagation algorithm [173] was used as training rule and implemented inside an 11*10 stratified nested cross-validation (SNCV) routine [181] with a simple early-stopping rule to regularise the training and limit the possible overfitting effect. The SCG combines line search and trust region techniques, removing the necessity of user-dependent parameters with an increment in speed of convergence and effectiveness of the algorithm. The

[20] The RMS of a time-varying quantity y(t) describes its average power. It is calculated as [179]:

$$y_{RMS} = \sqrt{\frac{1}{T}\int_{0}^{T}(y(t))^2 dt}$$

[21] The coefficient of variation (COV) is defined as the ratio of the standard deviation to the mean:

$$CV = \frac{\sigma}{\mu}$$

[22] The Jerk is the rate of change of acceleration, or the third derivative of position with respect to time.

SNCV is a technique that assesses the generalisation power of a statistical analysis over independent data groups, based on the multiple division of the entire dataset. It is composed of two nested cross-validation loops which are referred to as internal and external cross-validation loops [182]. Firstly, the dataset is randomly divided into K stratified folds, which are repetitively assigned to the Test and the Design set in the external loop. The Test set is exclusively used to obtain a reliable evaluation of the model performance. In the internal loop, the Design set is iteratively splitted in the Training and the Validation set to optimise the parameters of the model. In summary, the networks which displayed the lowest error in the internal loop are employed and assessed in the external loop, obtaining an unbiased estimation of its classification performance [183]. The categorical cross-entropy[23] function was used as loss metric to evaluate the different models across the validation process. Following the above described concept the training procedure adopted for the development of the two MLPs was the following:

```
STRATIFIED NESTED CROSS VALIDATION

PARAMETER DEFINITION
1- Define tuneable parameters, Hidden units: N
2- Define tuneable parameters, Initial weights: W
3- Define number of stratified folds: K
PERFORMANCE ESTIMATION
for k=1:K
  5- Define the present fold k as test set: T
  6- Define the remaining folds as design set: V
  for n=1:N
    for w=1:W
      for v=1:V
        7- Define the present fold v as validation set
        8- Train the model with parameters w and n on the remaining folds
        9- Test the model with parameters w and n on the validation set v
        10- Save the performance for the validation set v
      end
      11- Average the performance across V
      12- Save the validation result (matrix N x W x K)
    end
  end
  13- Average the results across the set of random initial weights
  14- Find the optimal number of hidden units n*
  15- Train the optimised model with n* hidden units on the design set V
  16- Test the optimised model on the test set T
  17- Save the performance for the test set T (matrix K x 1) < Estimated performance
  and the number of training iterations (matrix K x 1)
```

[23] For a multi-label classification, the loss is calculated using the following formula:

$$loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}(T_{i,j})\log(X_{i,j}) + (1 - T_{i,j})\log(1 - X_{i,j})$$

where $X_{i,j}$ is the network response for a given category, $T_{i,j}$ is the target value of that category, and C is the total number of categories. In this case, the cross-entropy loss is calculated as the probability of a given observation being assigned to a given category, summed over all categories and observations and normalized by the number of observations N.

```
end
18- Select the optimal number of hidden units n** that performed better across the majority of the folds
19- Train a network with n** hidden units on the entire dataset for the average number of training iterations across all the folds
```

2.2.2 Validation phase

In the validation phase, subjects executed the STS task in two experimental sessions, one for each MLP classifiers. For each session, participants performed 1 SP trial and 2 CT trials. Features from the inertial sensors were calculated and fed directly into the classifiers, synchronizing the output of the network with the force plate and the electronic switch. Subsequently, we compared the real-time classifications with the asynchronous categorisation of the movement based on the analysis of the GRF and the electronic switch signal (Chapter 2). To evaluate the classification performance of the two models, (2.10) the precision (PRC), (2.11) the recall (RCL), and (2.12) the F1 scores across all STS phases were extrapolated from the output of the networks. As overall-performance indexes, we also calculated the macro averages of these metrics together with (2.13) the general accuracy (ACC) achieved by each model.

$$PRC_p = \frac{TP_p}{TP_p + FP_p} \tag{2.10}$$

$$RCL_p = \frac{TP_p}{TP_p + FN_p} \tag{2.11}$$

$$F1_p = \frac{2 * PRC_p * RCL_p}{PRC_p + RCL_p} \tag{2.12}$$

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.13}$$

Additionally, the temporal distribution of the classification errors for each phase was evaluated by centring around its middle sample and comparing the network's outputs with the ground-truth categorisation over 300 ms time bins. To evaluate the temporal performance of each model, the acquisition time of every 100 ms sample (defined as the last recording instant of the signal from inertial sensors, see Figure 2.6) and the relative classification time were recorded.

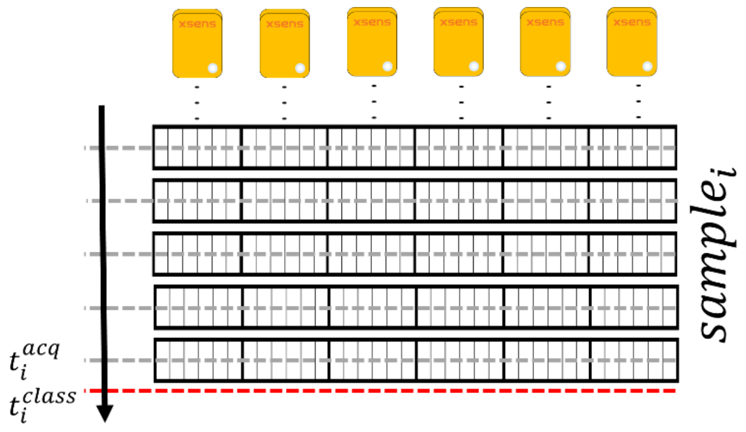


Figure 2.6: Real-time data acquisition workflow from the XSENS sensors. Data from each sensor are acquired sequentially until the completion of an entire datapacket at the user-defined sample frequency of 50 Hz. Five consecutive datapackets are used to define a sample of 100 ms in the analysed datasets. t_i^{acq} *is defined as the temporal instant in which the last datapacket of the current sample is acquired.* t_i^{class} *is defined as the temporal instant of classification after the current sample is acquired and fed to the MLP.*

Hence, (2.14) the average delay between data acquisition and classification; (2.15) and the average inter-classification time were calculated for each STS repetition. Finally, for both parameters 95% confidence intervals were calculated across all the specific datasets.

$$T_{delay} = \frac{\sum_{i=1}^{N} t_i^{class} - t_i^{acq}}{N} \tag{2.14}$$

$$T_{i-class} = \frac{\sum_{i=1}^{N} t_{i+1}^{class} - t_i^{class}}{N} \tag{2.15}$$

2.3 Results

The characterisation of the developed models is presented in Table 2.3.

Features	Sensors	Topology	Test Accuracy (%)
mean, RMS, max to RMS, min to RMS, SD, COV, jerk for each variable Roll, Pitch, Yaw, Acceleration on x-y-z axes	6 sensors (trunk, sacrum, trochanter Sx, trochanter Dx, fibula Sx, fibula Dx)	Input: 252 Hidden: 15 Output:4	89.2 [80.9 - 97.5]
	3 sensors (trunk, trochanter Sx, trochanter Dx)	Input: 126 Hidden: 90 Output: 4	92.2 [89.7 – 94.7]

Table 2.3: Characterisation of the developed MLPs

The final architectures for the two MLPs were identified in a 3 layer feed-forward topology with 15 hidden units for the 6-sensor-based network, and with 90 hidden units for the 3-sensor-based network. These models yielded the best validation performances (internal validation) and were therefore selected as optimal in the majority of the test folds (external validation), with accuracy values ranging around 90 % (3 sensors: 92.2 [89.7 – 94.7]; 6 sensors: 89.2 [80.9 – 97.5]).

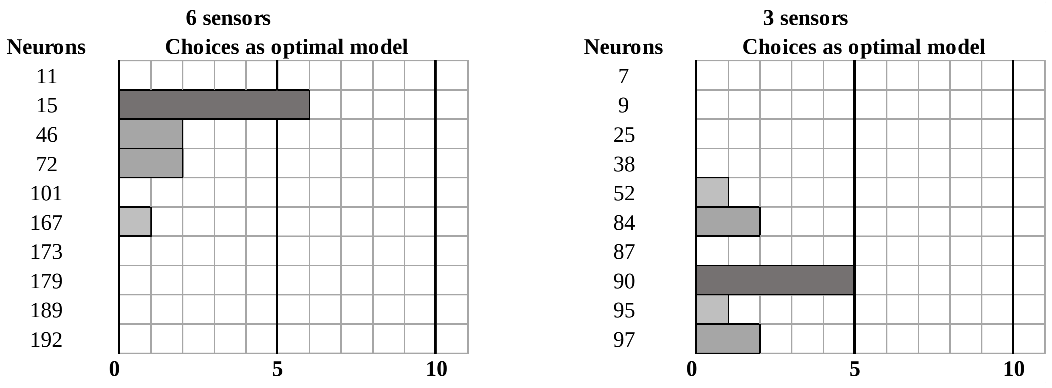
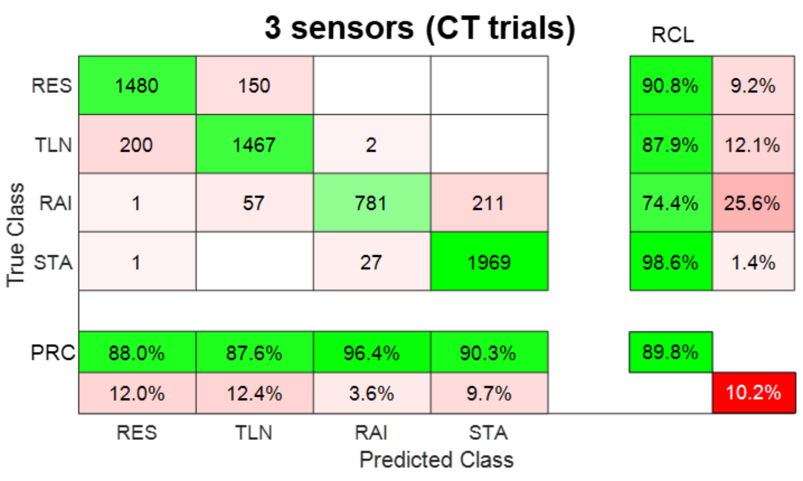
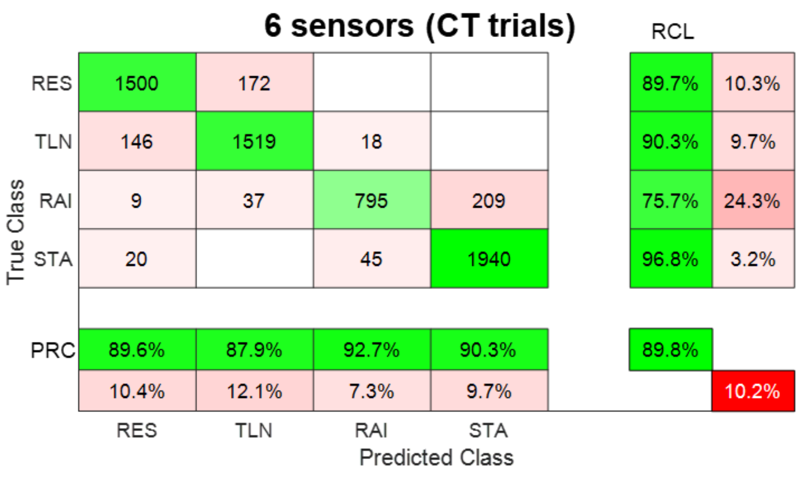


Figure 2.7: Frequencies of models' parameters tunings across the outer loops of the SNCV for the two developed models.



3 sensors (CT trials)

True Class \ Predicted Class	RES	TLN	RAI	STA	RCL	
RES	1480	150			90.8%	9.2%
TLN	200	1467	2		87.9%	12.1%
RAI	1	57	781	211	74.4%	25.6%
STA	1		27	1969	98.6%	1.4%
PRC	88.0%	87.6%	96.4%	90.3%	89.8%	
	12.0%	12.4%	3.6%	9.7%		10.2%



6 sensors (CT trials)

True Class \ Predicted Class	RES	TLN	RAI	STA	RCL	
RES	1500	172			89.7%	10.3%
TLN	146	1519	18		90.3%	9.7%
RAI	9	37	795	209	75.7%	24.3%
STA	20		45	1940	96.8%	3.2%
PRC	89.6%	87.9%	92.7%	90.3%	89.8%	
	10.4%	12.1%	7.3%	9.7%		10.2%

Figure 2.8: Confusion matrixes for the two developed MLPs tested on real-time CT trials.

Model	**Phase**	**PRC (%)**	**RCL (%)**	**F1 (%)**	**ACC (%)**	$T_{i-class}$ **(ms)**	T_{delay}**(ms)**
3 sensors	RES	88.0	90.8	89.4	89.8	100.1 [99.9 – 100.2]	26.0 [23.4 - 28.0]
	TLN	87.6	87.9	87.7			
	RAI	96.4	74.4	84.0			
	STA	90.3	98.6	94.3			
	Macro	90.6	87.9	89.2			
6 sensors	RES	89.6	89.7	89.6	89.8	99.9 [99.2 – 100.6]	27.5 [25.8 – 29.1]
	TLN	87.9	90.3	89.1			
	RAI	92.7	75.7	83.3			
	STA	90.3	96.8	93.4			
	Macro	90.1	88.1	89.1			

Table 2.4: Classification results for the two developed MLPs tested on real-time CT trials.

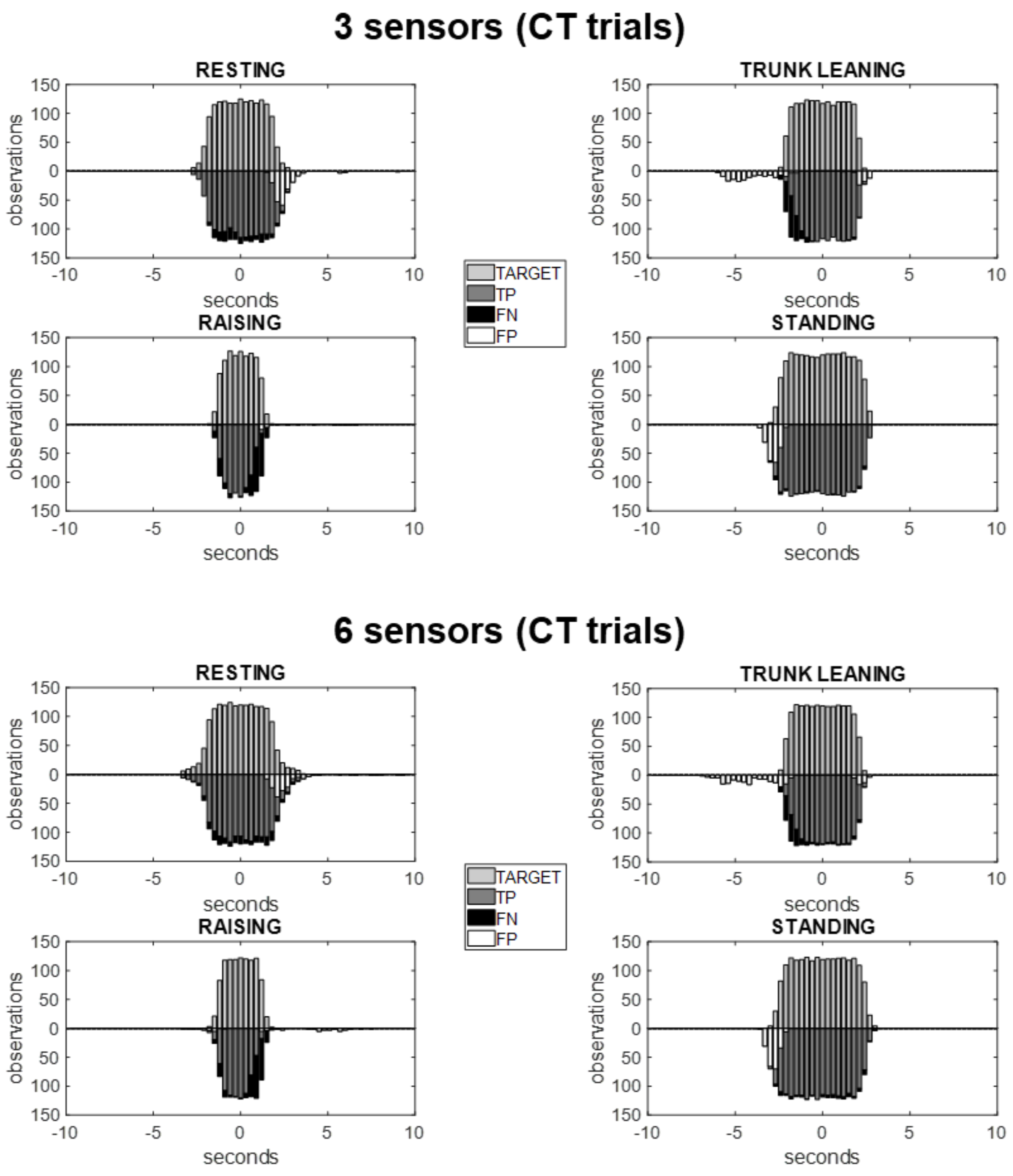


Figure 2.9: Temporal distribution of the classification outputs for the two developed MLPs tested on real-time CT trials.

The classification outputs over CT trials for both the developed MLPs are presented in Table 2.4 and graphically displayed in Figures 2.8 and 2.9. Both networks showed an 89.8 % overall accuracy in line with the average values of F1 scores (3 sensors: 89.2; 6 sensors: 89.1). Looking at the specific results of the individual phases of the movement, there was a generally slightly better performance in the recognition of static phases (RES, STA) than the dynamic ones (TLN, RAI), as highlighted by the relative F1 scores [Tab.2.4]. Also, the temporal performance metrics were very similar across the two models. The inter-classification times were close to the nominal 100 ms epoch necessary to acquired 5 samples at 50 Hz (3 sensors: 100.1 [99.9 – 100.2] ms; 6 sensors: 99.9 [99.2; 100.6] ms).

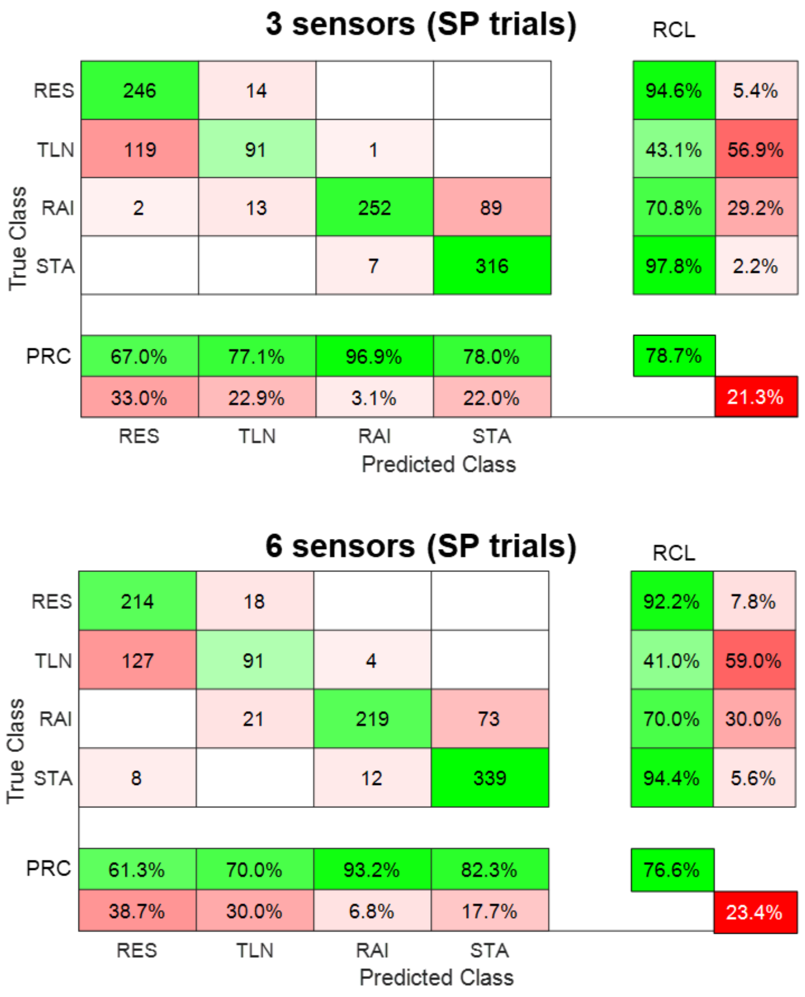


Figure 2.10: Confusion matrixes for the two developed MLPs tested on real-time SP trials.

Model	**Phase**	**PRC (%)**	**RCL (%)**	**F1 (%)**	**ACC (%)**	$T_{i-class}$ **(ms)**	T_{delay}**(ms)**
3 Sensors	RES	67.0	94.6	78.4	78.7	100.0 [99.8 – 100.2]	27.84 [24.1 –31.6]
	TLN	77.1	43.1	55.3			
	RAI	96.9	70.8	81.8			
	STA	78.0	97.8	86.8			
	Macro	79.8	76.6	78.1			
6 Sensors	RES	61.3	92.2	73.6	76.6	100.3 [99.9 – 100.7]	27.7 [24.1 – 31.4]
	TLN	70.0	41.0	51.7			
	RAI	93.2	70.0	80.0			
	STA	82.3	94.4	87.9			
	Macro	76.7	74.4	75.5			

Table 2.5: Classification results for the two developed MLPs tested on real-time SP trials.

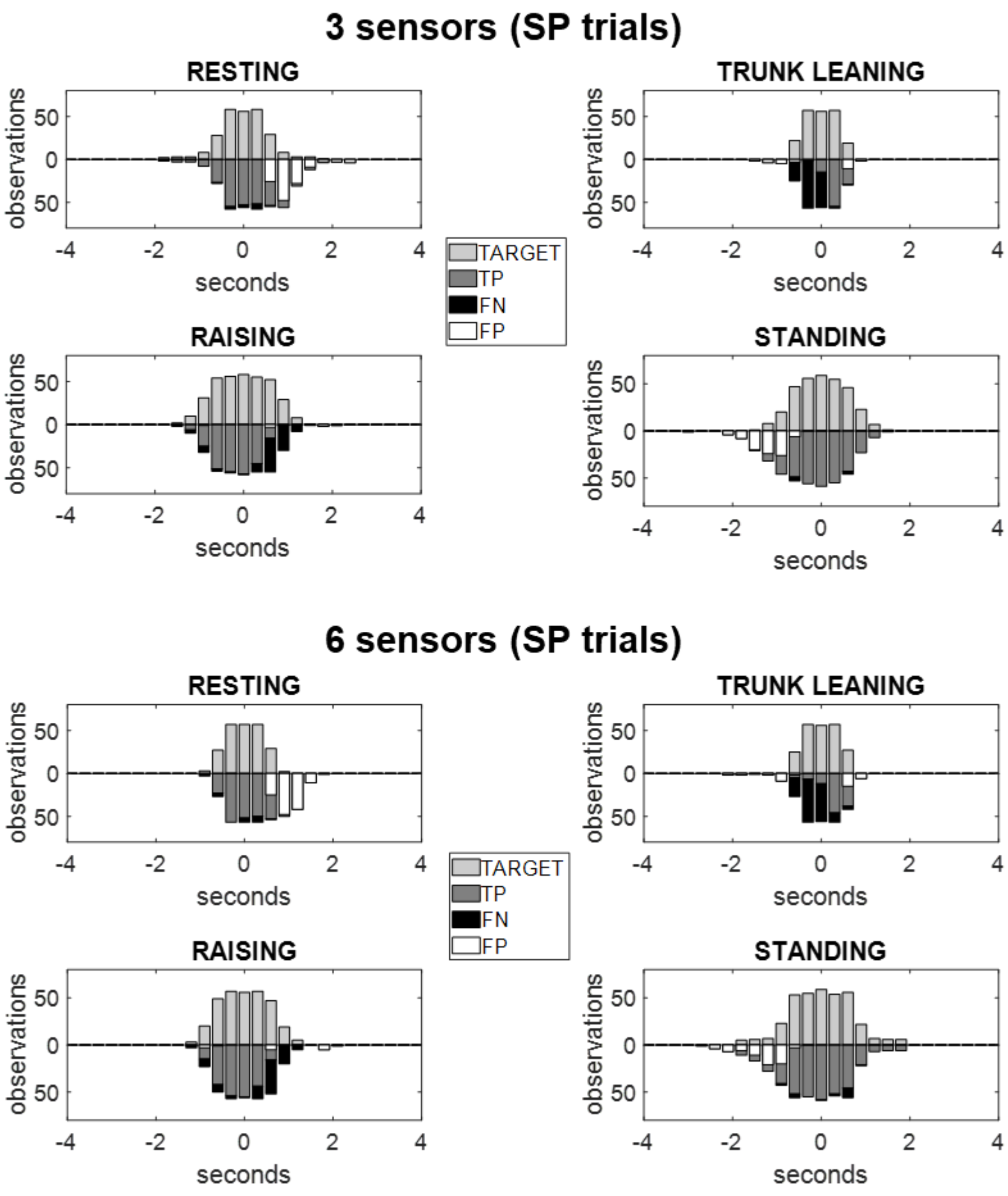


Figure 2.11: Temporal distribution of the classification outputs for the two developed MLPs tested on real-time SP trials.

The delays between acquisition and classification were also in line with these results respecting the 50 Hz (20 ms) between the end of one datapacket and the beginning of a new one (3 sensors: 26.0 [23.4 - 28.0] ms; 27.5 [25.8 – 29.1] ms). Moreover, focusing on the temporal distribution of the misclassified outputs [Fig.2.9] beyond the insights given by the confusion matrixes, it is evident how the major classification criticality lied in the transitions between the Rest phase and the Trunk Leaning phase and between the Raising phase and the Standing phase. A significant reduction in the classification performance was observed when the two models were experimented on the SP trials, as displayed in Figure 2.10-2.11 and Table 2.5. The two classifiers achieved an overall accuracy of 78.6 % for the 3-sensor-based implementation (F1: 78.1 %) and, 76.6 % for the 6-sensor-based (F1: 76.1 %). Similarly to what observed under controlled speed, in SP trials the temporal metrics were ranging narrowly around 100 ms for inter-classification times (3 sensors: 100.0 [99.8 – 100.2] ms; 100.3 [99.9 – 100.7] ms) and 20 ms for classification delays (3 sensors: 27.84 [24.1 – 31.6] ms; 6 sensors: 27.7 [24.1 – 31.4] ms). Also, in this case, the classification errors were concentrated between the Rest phase and the Trunk Leaning phase and between the Raising phase and the Standing phase. Finally, since each 100 ms epoch is processed from 5 consecutive samples at 50 Hz from the inertial sensors, the possible loss of information should be taken into account during the evaluation of the two classifiers. Such analysis is displayed in Figure 2.12, where the observations that lacked data from one or more IMU sensors across five consecutive are reported in red while the complete datapackets are shown in yellow.

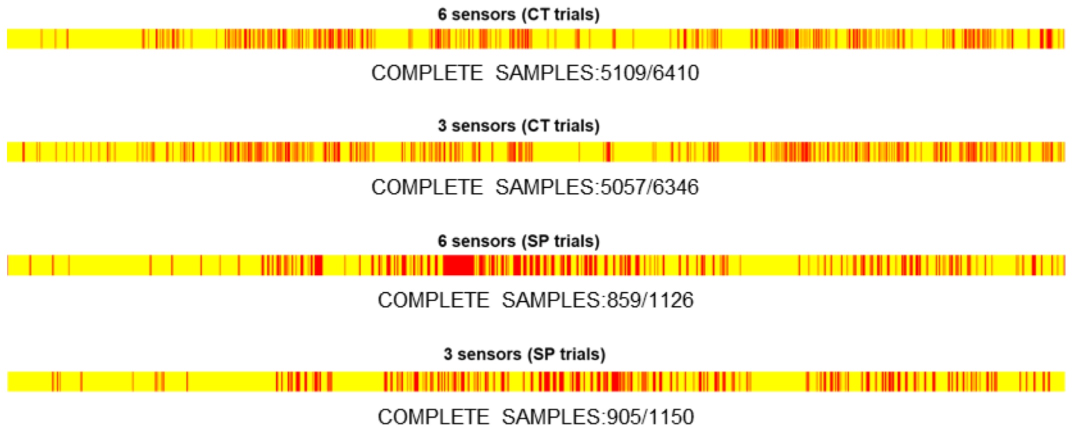


Figure 2.12: Analysis of the completeness of the data acquired in real-time. Complete samples are shown in yellow. Samples with one or more incomplete data packets are shown in red. The graph was obtained by aligning the real-time acquisitions of all the subjects

2.4 Discussion and conclusions

This work focused on the validation of 2 MLPs in a real-time classification task for the recognition of complex movement patterns in the STS. A similar paradigm was already exploited in a recent work by Doulah and colleagues [127], which applied a three-layer feed-forward network for the early identification of STS transitions. Although they did not test their algorithm over real-time data, the results they obtained suggested that MLPs were able to identify STS transitions from synchronous data acquired with different types of sensors (IMU, potentiometers, and force sensors). With a classification delay of 26.0 [23.4 - 28.0] ms, 27.5 [25.8 – 29.1] ms over CT trials, and 27.84 [24.1 – 31.6] ms, 27.7 [24.1 – 31.4] ms over SP trials this work confirmed the potential of simple architecture neural networks for the online detection of STS movement patterns. In terms of classification performance, the state-of-the-art methods for the STS pattern recognition reach a general accuracy above 90%. Yang and Hsu [184] implemented a rule-based algorithm to detect the standing and the sitting movements from a single wearable motion sensor with an accuracy of 92.2% and 95.6%. In their respective works, Bannerjee, Doulah and Martinez-Hernandez reached 94.6% [185], 92.9% [126], 99.4% [127] and 100% [122] accuracy in the identification of STS transitions. In this scenario, our models failed to compete against the state-of-the-art approaches with greater accuracy in CT trials (89.8% for both the 3 sensor-based and the 6 sensor-based networks) compared to SP trials (78.7% for the 3 sensor-based model and 76.6% for the 6 sensor-based network). To identify the next steps in the development of the project, the results obtained must be evaluated in light of some basic considerations. First, the main objective of this work was to develop a classifier model able to categorise the STS in four clinically relevant stages, characterising the movement beyond the identification of the simple transition of previous studies. The specificity of this task takes into account a greater level of uncertainty in the source dataset from which the neural networks were developed and trained. This was reflected by the large percentage of classification errors concentrated in the transitions between the Rest and the Trunk leaning phase and between the Raising and the Standing phase. This evidence was in accordance with what has been highlighted in Chapter 1, where the beginning of the trunk movement and the reaching of the stable upright stance proved to be the most challenging tasks for the labelisation software with a difference of approximately 300 ms compared to the optimal visual categorisation made by a human rater. The intrinsic movement variability in these two particular instants could have affected the training of our models, reflecting in a relatively poor classification performance during the real-time validation. Moreover, the lacks in the datastream (Fig.2.12) could also have negatively affected the performance of the MLPs, since in the general architecture no compensation for any data loss has been introduced to improve generalisation and avoid overfitting. The poor generalisation was also displayed in the low accuracy obtained over SP trials, which were somehow expected since the two models were trained over CT movements to avoid class imbalance. A partial solution to some of the above presented critical issues could be found in the work of Martinez-Hernandez and Abbas [122] where a probabilistic method was used to identify STS transitions with a 100% classification accuracy. In their study they presented a method based on the Bayesian theory and sequential analysis, comparing their results with other related work on the STS movement recognition. The Bayesian method could classify data relying on its intrinsic dependancies in time [186]. Time series data is ubiquitous and it is used to represent weather readings, financial recordings, industrial observations, psychological signals, electronic health records, and human activity recognition [187], [188]. Under this consideration, the recognition of the STS translates into a time-series classification problem, where the temporal relations retained in the ordered sequence of movements can be exploited to improve the overall classification performance. Moreover, this information is essential to reduce the number of sensors without compromising the classification accuracy

Chapter 3: A Hybrid Convolutional-Recurrent Deep Learning Approach for a Clinically Relevant Real-Time Classification of the Sit-to-Stand Movement.

3.1 Introduction

Starting from the evidence presented in the previous sections, this chapter addresses the topic of time-series analysis, by introducing more advanced deep architectures and applying them to the classification of the STS motion pattern.

3.1.1 Time series data and sequential predictive problems

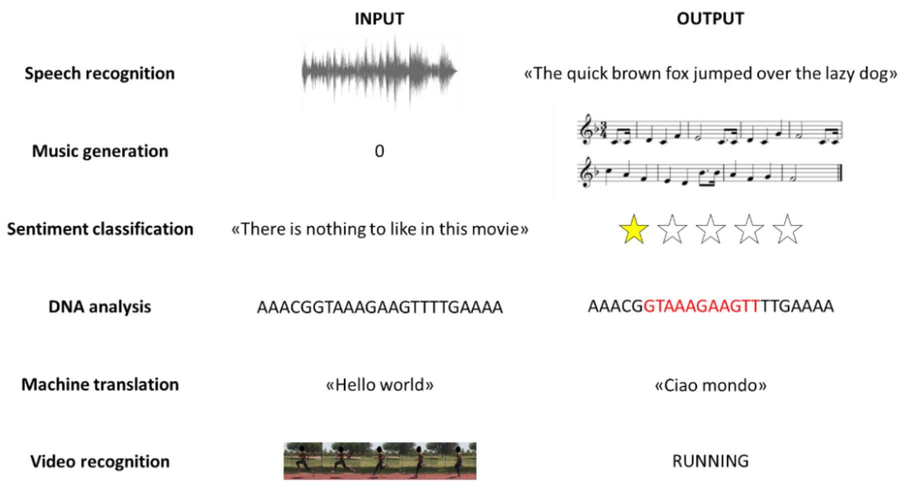


Figure 3.1: Example of sequential data and applications. In speech recognition, an input audio clip (sequence) is elaborated to obtain text transcript in output. In music generation, the output is a music sequence and the input can be either an integer value or an empty set or a sequence of notes to start the generation. In sentiment classification, a text is processed to extract a label (i.e good, bad, excellent, awful). In DNA analysis it is possible to identify, within a sequence of nucleotides, the portion that codifies a specific protein. In machine translation text sequences are automatically translated across different languages. In video recognition, a video clip (or each frame) can be classified in a label (i.e. activity recognition). [189]

Time-series defines a type of data where the samples are temporally ordered. In a broader conception, this definition refers to a specific subset of a larger domain of sequential data, which includes any type of ordered information that shows a significant sequential correlation [190]. That is, nearby samples are likely to be related to each, rather than being drawn from an independent and identically distributed (IID) random variable. Sequential predictive methods exploit the knowledge retained within these data patterns to improve the prediction outcome.

As an example, it is possible to detect if a telephone is stolen by looking at the distribution of legitimate phone calls and then checking its variations to detect fraudulent activity. In text analysis, it is possible to identify the style that characterise an author or a section related to a specific subject [191]. However, despite the variety of applications, a sequential predictive task can be categorised into four major groups:

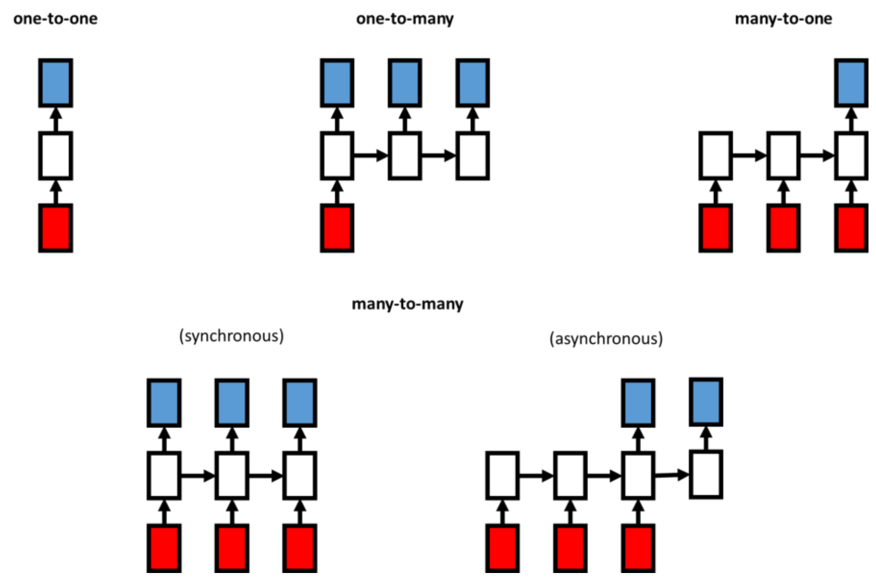


Figure 3.2: Type of sequential predictive problems. The inputs are represented by the red squares and the outputs are represented by the blue squares. The processing steps are represented in white.

- one-to-one, characterised by one fixed-sized input and one fixed-sized output;
- one-to-many, characterised by one fixed-sized input and an output sequence;
- many-to-one, characterised by an input sequence and a one fixed-sized output (i.e. in sentiment classification, a text can be classified as expressing positive or negative sentiment).
- many-to-many, characterised by sequences both in input and in output which can be elaborated synchronously (i.e. in video recognition each frame of the video is classified and labelled) or asynchronously (i.e. in machine translation, a sentence is first read and then translated in different languages);

The cardinality between inputs and outputs, and the specific setting of the predictive problem impose a constraint over the model architecture [192]. In the context of the present work, the STS movement pattern recognition can be seen as a many-to-many synchronous sequential classification task, that can be expressed as follows. Let $\{(x_i, y_i)\}_{i=1}^{N}$ be a set of N training examples, where each element is represented by a pair of sequences (x_i, y_i) where $x_i = \langle x_{i,1}, x_{i,2}, x_{i,3} \dots \dots \dots x_{i,T} \rangle$ represent the ith STS movement expressed as a time-series and $y_i = \langle y_{i,1}, y_{i,2}, y_{i,3} \dots \dots \dots y_{i,T} \rangle$ represent the respective sequence of STS phase labels at each time-stamp. The aim is to develop a classifier c that can classify the movement pattern $y = c(x)$ given an input sequence x [190]. However, while the MLPs architectures introduced so far (see Chapter 2) are naturally fitted to solve one-to-one predictive problems, they present some major limitations when dealing with sequential data. Feedforward networks take into account the assumption of a pre-defined context length with fixed-sized inputs and outputs [193], [194] which contrasts with the intrinsic variability in length of sequential data. Moreover, by considering each element of the input sequence independently (every single input has its weight), MLPs do not exhibit any spatial sequential invariance and the temporal information is lost [188]. Starting from these assumptions, the following sections explores more complex neural network models, able to deal with the dynamic nature of the data characterising the STS movement pattern.

3.1.2 Recurrent neural networks

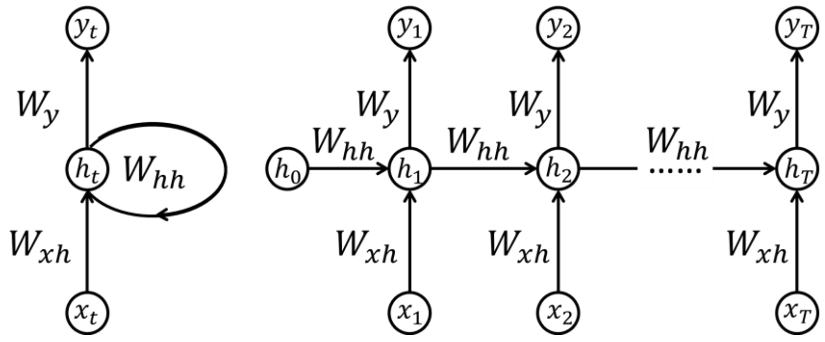


Figure 3.3: Simple recurrent neural network from Elman. The hidden layer has a feedback connection as part of its inputs. On the left, the recurrent model, on the right the unfolded architecture in time [195].

Recurrent neural networks (RNN) can be easily defined as specific types of architectures that maintain a “memory” of the previous inputs within a sequence to influence the present output [196]. This is achieved through internal loop connections in the network architecture that introduce recursive dynamics across its processing elements [197]. Elman [195] introduced a simple RNN model, where the hidden layer has feedback connections to the inputs, as a form of short term memory of the previous activations of the hidden nodes (i.e. hidden state). [198]. That is, the parameters of the network are shared over time [Fig.3.4].

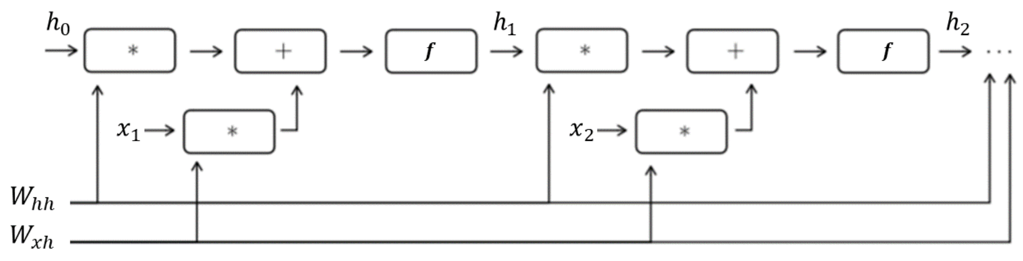


Figure 3.4: A computation graph corresponding to a simple RNN. Two time-steps are shown, but the computation graph can be unrolled indefinitely. The symbol $*$ denotes matrix multiplication. [193]

More generally, RNNs maps a sequence of inputs e $x = \langle x_1, x_2, x_3 \dots x_T \rangle$ to a sequence of hidden states $h = \langle h_1, h_2, h_3 \dots h_T \rangle$ through a set of parameters θ:

$$h_t = f(h_{t-1,} x_t, \theta) \tag{3.1}$$

This formulation allows to describe RNNs as actual computational blocks, where f can represent different types of nonlinearities, and x_t and h_t does not need to be directly related to input and output data. In this way, it is possible to feed an input sequence to a recursive block, and the resulting hidden states are used by another RNN as inputs in deeper architectures [193].

3.1.3 Long short-term memory networks: Beyond the limits of simple RNNs

As discussed above, RNNs overcome the limitations of the feedforward neural network since no specific constraints are imposed on either the inputs or the outputs [194]. Hence, with no dependence on the context dimensions, it is theoretically possible to capture long-term relationships within a sequence. However, while the RNN architecture itself naturally motivates the modelling of temporally distant dependencies, these are not effectively learned by standard gradient-based training algorithms [199]. As data relationships become more distant, the gradients become unstable, increasing or shrinking exponentially with the lengths of the sequences. To understand this mechanic, it is necessary to consider the cost function of a general RNN (Fig.3.3) [193], [200]:

$$J = \sum_{t=1}^{T} L_t \tag{3.2}$$

Where L_t represent the loss of the network performance at the time t. One classical training procedure is carried out by following the same principles of the backpropagation, where the RNN is expanded over time in a multi-layer architecture [Fig.3.3 (left)], and the algorithm is applied on the unrolled model. This temporal variant of the routine defines the backpropagation through time (BPTT). By applying the chain rule to obtain the gradients of the cost function with respect to the parameters of the network θ :

$$\frac{dJ}{d\theta} = \sum_{t=1}^{T} \frac{dL_t}{d\theta} \tag{3.3}$$

$$\frac{dL_t}{d\theta} = \frac{\partial L_t}{\partial h_t} \frac{dh_t}{d\theta} \tag{3.4}$$

It is important to note that h_t depends from h_{t-1}which in turn depends from h_{t-2} and θ, hence by applying the chain rule again and summing the contribution of every time step:

$$\frac{dL_t}{d\theta} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{dh_k}{d\theta} \tag{3.5}$$

Here $\frac{\partial h_t}{\partial h_k}$ transport the error in time from step t back to step k. If its spectral norm is extremely small, then the effect of $h_k$over the gradient computation is limited, and the network find very difficult to learn from events that occurred at time k. Specifically, since in simple RNNs there is one only possible "temporal path" from h_k to h_t, the term can be further expanded in:

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k}^{t} \frac{\partial h_i}{\partial h_{i-1}} \tag{3.6}$$

Which, by considering the spectral norms, can be written as [193]:

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq \left\| \frac{\partial h_t}{\partial h_{t-1}} \right\| \left\| \frac{\partial h_{t-1}}{\partial h_{t-2}} \right\| \cdots \left\| \frac{\partial h_{k+1}}{\partial h_k} \right\| \tag{3.7}$$

Each partial is a Jacobian result of the product of two matrixes, one related to the specific function f(diagonal) (Fig.3.4) and one related to the recurrent weights W_{hh}. Intuitively, if the largest spectral norm of any partial is lower than the unity, the contribution of the gradients fall exponentially with $t - k$ (vanishing gradients). Conversely, if the lowest spectral norm of any partial is larger than the unity the gradients grow exponentially with $t-k$ (exploding gradients). The problem of exploding gradients is generally handled with the gradient clipping technique, which limits the derivative so that it lies inside a specified range. However, a more difficult challenge is mitigating the vanishing gradients issue. Among different alternatives proposed in the literature [201]–[203], the solution that gained major success is the Long Short-Term Memory (LSTM) architecture [204], which stabilise the spectral norms across the gradients with additional paths between h_k to h_t [193].

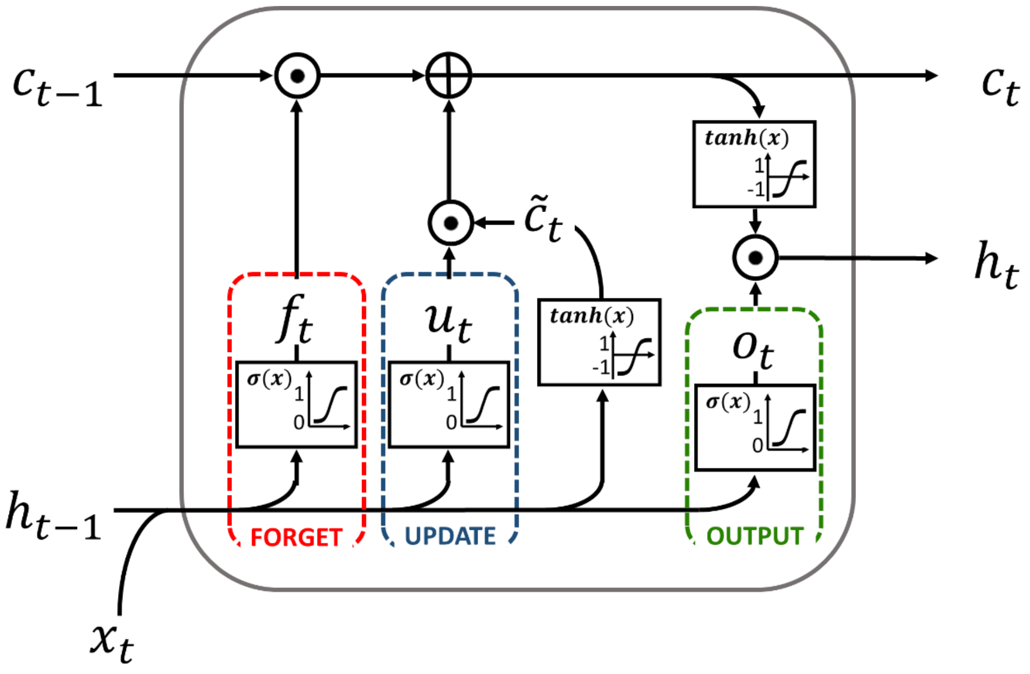


Figure 3.5: Graphical depiction of the LSTM

LSTM networks are distinguished by the ability to manage a memory cell c_t at time t, by resetting, writing, and reading it using three sub-architectures called respectively the forget gate, the update gate, and the output gate. The structure of a generic LSTM is depicted in figure 40 and functionally described by the equations below:

$$f_t = \sigma\left(W_{fh}h_{t-1} + W_{fx}x_t + b_f\right) \tag{3.8}$$

$$u_t = \sigma(W_{uh}h_{t-1} + W_{ux}x_t + b_u) \tag{3.9}$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \tag{3.10}$$

$$\tilde{c}_t = tanh(W_{ch}h_{t-1} + W_{cx}x_t + b_c) \tag{3.11}$$

$$c_t = f_t \odot c_{t-1} + u_t \odot \tilde{c}_t \tag{3.12}$$

$$h_t = o_t \odot \tanh\left(c_t\right) \tag{3.13}$$

Each gate is a simple RNN characterised by a sigmoid activation function, with outputs limited in the range between 0 and 1, controlling the flow of information. A new candidate update for the memory cell $\tilde{c}_t$ is combined with c_{t-1}, according to the forget gate and the update gate, to obtain the new memory cell c_t. Finally, the hidden state h_t is formed by applying the tanh activation function to the present memory cell and weighting the result according to the output gate. This allows the LSTM to better control the behaviour of the gradients, deciding at each time step what information should be retained and updating the model parameters accordingly [204]–[206].

3.1.4 Convolutional neural networks

As we highlighted in the previous section, RNNs are commonly considered the starting point (if not the favourite solution) for sequential predictive tasks [207]. However, another type of network model, the convolutional neural networks (CNN), proved to be capable of handling sequential data, obtaining optimal results in various challenging applications like audio generation, language processing, and machine translation [208]–[211]. As the name suggests, CNNs are a specific type of network that employs a convolution instead of straightforward matrix multiplications in at least one layer [212]. The convolution operation can be defined as:

$$f(x) = \int i(a)k(x-a)\,da = (i * k)(x) \tag{3.14}$$

The term $i(x)$ is referred to as the input function, the term k(x) is the kernel and the output $f(x)$ is generally called feature map. In the context of deep-learning applications, CNN operates on multidimensional arrays of data including time-series data (1-D array) and images (2-D array), hence, the previous formula can be discretised as:

$$f(x) = \sum_n i(n)k(x-n) \tag{3.15}$$

$$f(x) = \sum_m \sum_n i(m,n)k(x-m, y-n) \tag{3.16}$$

Respectively for 1-D and 2-D inputs. Theoretically, by looking at the formulation of the convolution, the kernel is flipped in respect to the input. As m or n increases the indexes in the input increase accordingly, while the ones in the output decrease. Nonetheless, it is more common in practice to implement the cross-correlation function, which does not imply the flipping of the kernel term:

$$f(x) = \sum_n i(x+n)k(n) \tag{3.17}$$

$$f(x) = \sum_m \sum_n i(x+m, y+n)k(m,n) \tag{3.18}$$

Despite this slight variation, the terminology does not change, and the definition of convolution is overall accepted and used for both convolution and cross-correlation functions. The parameters that describe the convolutional operations are:

- the kernel size (d_{kernel}), which defines the receptive field of the convolution;
- the stride (s), which defines the step size of the kernel when spanning the input;
- the padding ($d_{padding}$), which defines how the limits of the input are handled;

Unpadded convolutions reduce the dimensions of the sequence in the output according to the following equation:

$$dim_{output} = \frac{(dim_{input} - dim_{kernel} + dim_{padding})}{s} + 1 \tag{3.19}$$

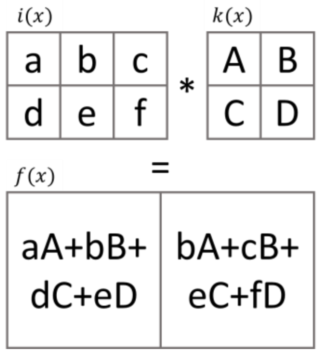


Figure 3.6: Graphical example of a 2-D convolution

Several reasons are motivating the practical success of CNNs in sequential processing and general machine learning applications [212]. The most immediate one is that, like RNNs, convolutional networks do not require a pre-defined context with fixed-dimensions. However, the real added value of the convolutional structure is its intrinsic efficiency in terms of memory requirements. CNNs can operate on very large images, detecting very specific features with a relatively small kernel. This is due to CNNs' properties of sparse connectivity and parameter sharing [Fig.3.7], which limit the number of variables that need to be stored.

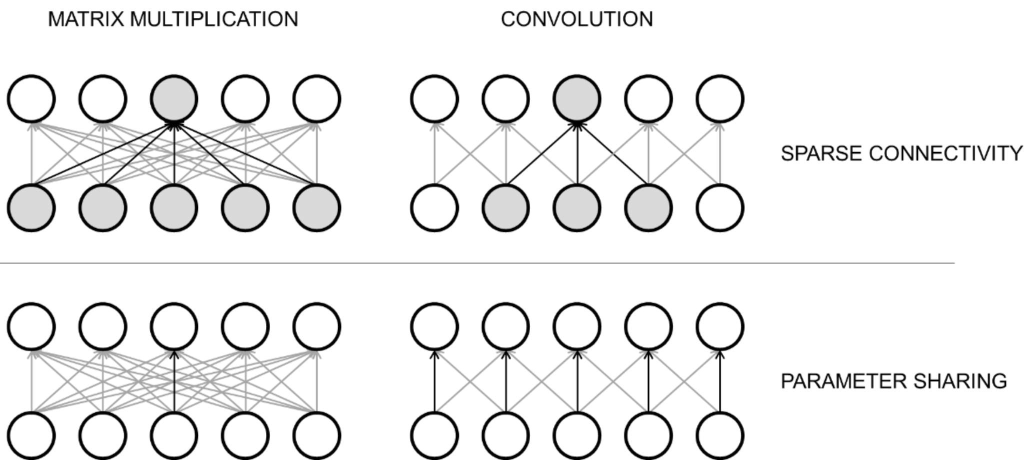


Figure 3.7: (Top) In matrix multiplication, all the inputs affect every single output. In the convolution, with a kernel of width 3, only three inputs affect the output unit (sparse connectivity). (Bottom) In matrix multiplication, each parameter (black arrows) is used only once. In the convolution, the central parameter of a 3-element kernel in a convolutional model is used at all input locations. (parameter sharing) [212].

Moreover, sharing the parameter across the inputs implies another useful property of the convolution, which defines itself as invariant to translation. Focusing on time-series data, if the convolution produces a representation of an event at time t and the same event repeats itself at time t+k, the same representation will be also found in output at time t+k. These concepts are at the base of CNNs' capacity to model the translation-invariant characteristics of human activities and capture the local dependencies of temporal sequences [213].

3.1.5 Temporal convolutional neural networks

Relying on these concepts and the best practices in deep learning design, Bai [207] used a terminology previously introduced by Lea and colleagues [214] to describe a convolutional architecture specifically optimised for sequential predictive problems. In this sense, temporal convolutional neural network (TCN) have two important characteristics:

- the convolution operations in the architecture are “causal”, that is the output at time T does not depend on inputs at t >T;
- the network can operate on input sequences of any length and process an output sequence of the same dimension;

The general TCN model described by Bai in [207] is reported in figure 3.8. The TCN is designed as a fully convolutional network model [215], where every hidden layer has the same size of the input. This is accomplished by using a zero-padding technique, adding at the left of the temporal sequence k-1 zero elements, where k represents the kernel size.

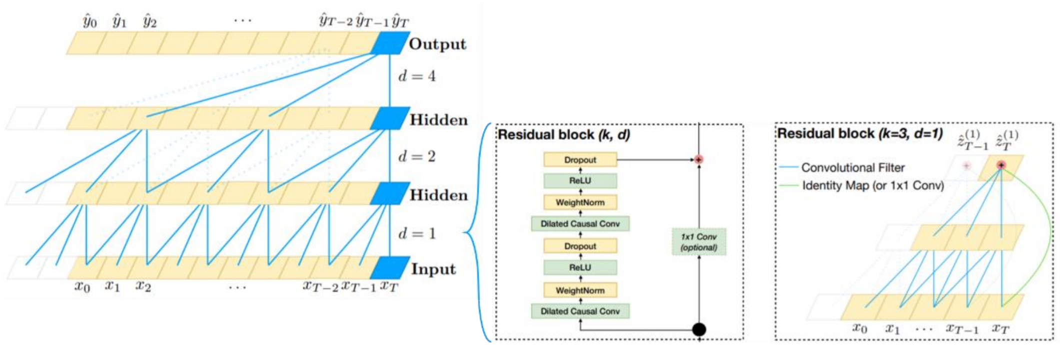

$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
Residual block (k=3, d=1)
$\hat{z}^{(1)}_{T-1}$ $\hat{z}^{(1)}_T$
Convolutional Filter
Identity Map (or 1x1 Conv)
x_0 x_1 ... x_{T-1} x_T

Figure 3.8: A general architecture of a TCN, formed by 3 residual blocks characterised by dilated causal convolutions with dilation factors d = 1, 2, 4 and filter size k = 3. Each residual block comprises two sets of dilated convolutions followed by a weight normalisation, a non-linear activation, and a dropout function, which prevents overfitting during training. A 1x1 convolution is added when residual input and output have different dimensions. On the far right, an example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are skip connections. [207]

However, the use of simple causal convolution allows retaining limited information in time [207], hindering the performance of the network on all those tasks that require the modelling of long-distance relations within a sequence. Hence, to increase the memory capacity, TCNs implement a variant of the normal convolution, able to inflate by a dilation factor the receptive field of the kernel function [216]. In this way, TCNs allows to effectively control their memory requirements, by choosing larger filter sizes k and/or increasing the dilation factor d. Finally, since the receptive field depends also on the depth of the network, the stabilisation of deepest and larger structures becomes a key feature in the TCN design. For this reason, convolutional layers are embedded inside a specific building module called residual block, which ease the training of more complex architectures [217] and defines the network topology. The residual blocks are characterised by skip connections, that allows the information to flow directly from one layer to another, and comprehends two dilated convolutional layer with non-linear activations (ReLU) [218].

3.1.5 Deep-learning in HAR applications

Conventional approaches for pattern recognition problems follow a well-defined general scheme [Fig.3.9] [219]. Information from different sensor modalities is acquired and processed to extract significant features, which are ultimately used by a decision model to make inferences on the collected data. These methodologies reached impressive progress in the HAR field, but their implementation is highlighted by some important drawbacks. The feature extraction process is heavily based on personal experience and knowledge relating to the specific domain of application. Information processed through human expertise using heuristic methods often refers to some statistical information of the signals (i.e. mean, variance, frequency, amplitude). These features can be used to identify low-level activities (i.e walking, running), but are not sufficient to recognise high-level or context-aware activities [220], [221].

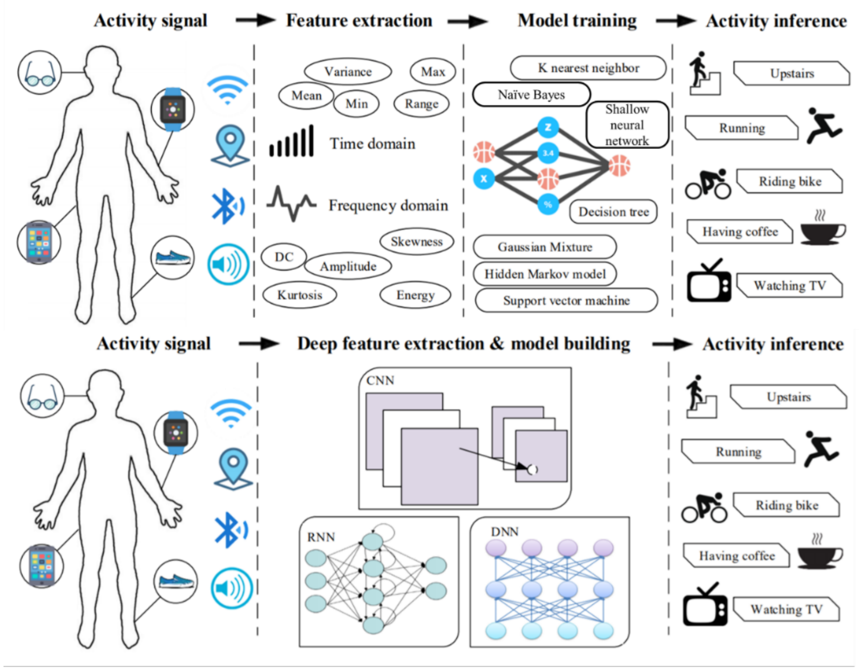
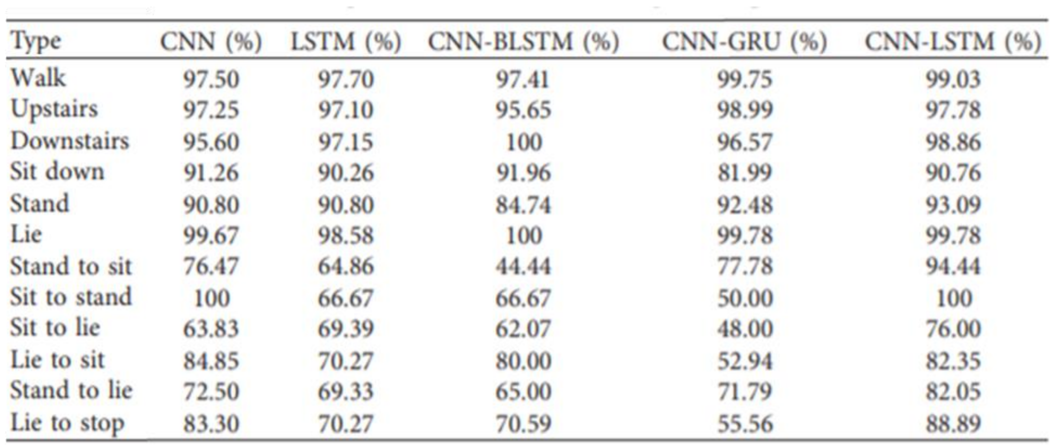


Figure 3.9: HAR applications using conventional (top) and deep-learning (bottom) approaches [219]

Moreover, the majority of conventional approaches are based on static data, while every aspect of the real world is characterised by a continuous stream of information, requiring robust and synchronous incremental learning. Deep-learning approaches overcome these limitations, as the feature extraction process is carried out automatically rather than be manually designed. Through training, the network learns the significant features of the data, and extract higher-level representations along with the depth of the model. In this context, the use of both RNNs and CNNs in HAR applications has been extensively investigated in the latest years [124], [213], [222]–[227]. By exploiting the information of the time-order relationship RNNs are recommended to recognize ordered short activities. On the other hand, CNNs are better at inferring long-term repetitive activities by learning deep features contained in recursive patterns. However, recent studies pointed out how hybrid implementations of recurrent and convolutional architectures can achieve better results in activity recognition tasks, beyond the one obtained using the single models [228]–[231]. In particular, Wang and colleagues [231] showed that the combination of CNNs and LSTM outperformed other deep-learning solutions in the recognition of basic and transitions movements (i.e. sitting to standing, standing to sitting, sitting to lying, and standing to lying).



Type	CNN (%)	LSTM (%)	CNN-BLSTM (%)	CNN-GRU (%)	CNN-LSTM (%)
Walk	97.50	97.70	97.41	99.75	99.03
Upstairs	97.25	97.10	95.65	98.99	97.78
Downstairs	95.60	97.15	100	96.57	98.86
Sit down	91.26	90.26	91.96	81.99	90.76
Stand	90.80	90.80	84.74	92.48	93.09
Lie	99.67	98.58	100	99.78	99.78
Stand to sit	76.47	64.86	44.44	77.78	94.44
Sit to stand	100	66.67	66.67	50.00	100
Sit to lie	63.83	69.39	62.07	48.00	76.00
Lie to sit	84.85	70.27	80.00	52.94	82.35
Stand to lie	72.50	69.33	65.00	71.79	82.05
Lie to stop	83.30	70.27	70.59	55.56	88.89

Table 3.1: Average accuracy of different activities with five deep learning models [231]

3.1.6 Objective

By exploiting the peculiar characteristics of the deep learning approaches above described, this chapter aims at investigating the performance of a hybrid-design model that combines the TCN [207] and LSTM [204] architectures for the recognition of the STS movement pattern.

3.2 Methods

3.2.1 Deep-learning architectures

The hybrid (HYB) model was assembled from the individual basic networks (LSTM and TCN). The LSTM architecture was characterised by a simple design (Fig.3.10), with the main recurrent core of the model followed by a fully connected layer and a softmax activation function in output.

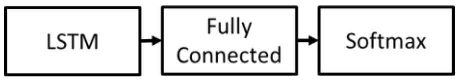


Figure 3.10: The LSTM network architecture.

The TCN model was designed upon the model described in the work of Bai [207], [232] as 4 residual blocks followed, also in this case, by a fully connected layer and a softmax activation function (Fig.3.11). A single residual block was defined by two dilated convolutions, followed by a normalisation and a ReLU activation function. At the end of the residual block, the inputs were added to the output of the dilated convolutions. An optional 1-by-1 convolution was implemented in case of different dimensions between inputs and outputs. The dilation factor of subsequent residual blocks was increased exponentially to expand the receptive field of the network. Let the dilation factor d of the kth block be defined as:

$$d = 2^{(k-1)} \tag{3.20}$$

then the receptive field size of such a network can be computed as

$$R = (f-1)(2^K - 1) + 1 \tag{3.21}$$

where f is the filter size and K is the number of convolutional layers. At the end of the residual block, the inputs were added to the output of the dilated convolutions.

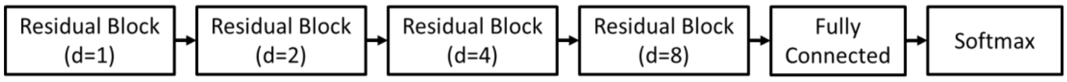


Figure 3.11: The TCN network architecture

Finally, the HYB network was implemented as a concatenation of the previously described architectures: with the LSTM network connected in-chain with the four residual blocks of the TCN. The complete model is represented in detail in figure 3.12.

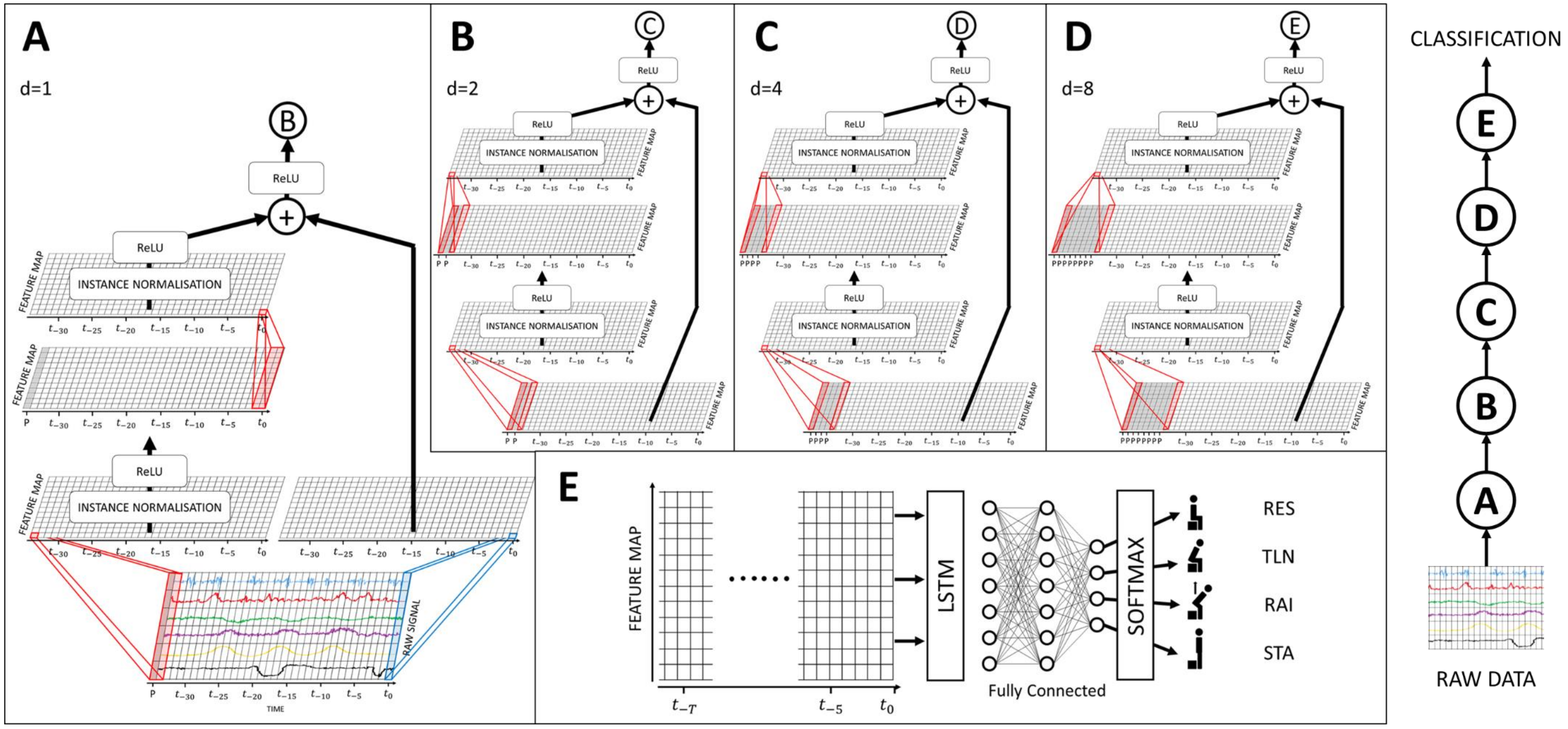


Figure 3.12: The hybrid network architecture

3.2.2 Model tuning and evaluation

The tuning of the HYB network followed a two-step process. In the first part, the LSTM and the TCN networks were singularly designed and evaluated on the combined data from HAR1 and HAR3 datasets. For each architecture, two separate models were developed, specifically focused on the recognition of the two different modalities of execution of the STS movement, under controlled (CT trials) and self-paced speed (SP trials). The models were directly trained on the signals acquired by the single IMU sensor placed on the chest, (namely the 3-D accelerations and orientations) without calculating any complex features. The only pre-processing operation on the raw data was to average 5 consecutive samples at 50 Hz (20 ms) to obtain a resampled version of the original signal at 10 Hz (100 ms) with T time steps. In this way, each STS acquisition was represented as a 2-D array of 6-by-T elements and labelled according to the values of the GRF through the software described in Chapter 1.2.2. To improve the statistical power of the estimated performance, each model was trained and tested using a 5-repeated 11*10 stratified nested cross-validation routine to tune a set of parameters related to the specific architecture [Tab.3.2].

Model	Parameter
LSTM	Number of hidden units Learning rate of the training algorithm
TCN	Size of the convolutional filters Learning rate of the training algorithm

Table 3.2: List of parameters chosen for model optimization related to TCN and LSTM networks.

Essentially, the algorithm repeats the subdivision of the dataset in 11 folds 5 times, altering the order of the subjects in the original dataset. In this way the composition of the folds is varied at each iteration, reducing the noisiness of the performance estimate obtained by the validation [233], [234]. The described procedure is schematically reported below:

```
STEP 1) TRAINING AND EVALUATION OF THE TCN AND LSTM NETWORKS
PARAMETER DEFINITION
1- Define tuneable parameters: P
2- Define number of folds: K
3- Define number of repetitions: R
PERFORMANCE ESTIMATION
for r=1:R
  4- Define the composition (randomisation) of the K folds: r
  for k=1:K
    5- Define the present fold k as test set: T
    6- Define the remaining folds as design set: V
    for p=1:P
      for v=1:V
        7- Define the present fold v as validation set
        8- Train the model with parameter p on the remaining folds
        9- Test the model with parameter p on the validation set v
        10- Save the performance for the validation set v
      end
      11- Average the performance across V
      12- Save the validation result (matrix P x K x R)
    end
    13- Find the optimal p*
    14- Train the optimised model with p* on the design set V<general train performance
    15- Save the performance for the train set T (matrix K x R)
    for each sequence in T
      16- Test the complete model on the sequence in T < test performance
    end
    17- Average the performance across the entire test set T
    18- Save the performance for the test set T (matrix K x R) < general test performance
  end
FINAL PARAMETER TUNING ON THE ENTIRE DATASET
  for p=1:P
    for k=1:K
      19- Define the present fold k as test set
      20- Train the model with parameter p on the remaining folds
      21- Test the model with parameter p on the test set k
      22- Save the performance for the test set k
    end
    23- Average the performance across K
    24- Save the test result (matrix P x R)
  end
end
FINAL TRAINING OF THE MODEL
25- Choose the best parameter p** with the best average performance across all the repetitions R
26- Choose the best randomisation r* of the dataset which achieved the best result across all the parameters P
27- Train the final model on the entire dataset randomised r* with the optimal parameter p**
```

The backpropagation algorithm with Adam optimisation was used as a general learning rule in the training routine (defined by the parameters in Table 3.3).

Parameter	Value	Description
Max epochs	30	The maximum number of epochs to use for training. An epoch is the full pass of the training algorithm over the entire training set
Mini-batch size	20	Size of the mini-batch to use for each training iteration. An iteration is one step taken in the gradient descent algorithm towards minimizing the loss function. The mini-batch size defines the portion of the dataset used to train the network at each iteration.
Learn-rate drop factor	0.1	Multiplicative factor to apply to the learning rate every time a certain number of epochs passes.
Learn-rate drop period	12	The number of epochs for dropping the learning rate.
Gradient threshold	1	Clipping value, which limits the gradient from exploding above a certain threshold.
L2 coefficient	0.0001	The factor for L2 regularisation (weight decay). Regularisations are techniques used to reduce the prediction error by tailoring an appropriate model on the data to avoid overfitting.

Table 3.3: General parameters used in the training routine.

The Adam optimiser [235] is one of the most popular solutions in the field of deep learning, outperforming, by a big margin, other used methods for an optimised gradient descent [236], [237]. During training, a dropout function was added both in the LSTM and TCN models, respectively between recurrent and the fully connected layer, and between the two consecutive dilated convolutions. The dropout function [230] is a form of regularisation which prevents overfitting by randomly zeroing some portions of the input data with a defined probability. As a second step, for each randomisation, the trained TCN was applied to extract, at the end of the residual blocks, the elaborated features from all the STS sequences, maintaining at the same the fold composition of the dataset (and every randomisation of it). These features were used as a base to tune the recurrent part of the hybrid model, and the resulting implementation was combined with the optimal TCN obtained in the first step to compose the final architecture, as described in the pseudo-code below:

```
STEP 2) TRAINING AND EVALUATION OF THE HYBRID ARCHITECTURE
PARAMETER DEFINITION
1- Define tuneable parameters of the recurrent model: P
2- Define number of folds: K (from STEP 1)
3- Define number of repetitions: R (from STEP 1)
4- Required the optimised TCN model (from STEP 1)
for r=1:R
  for k=1:K
    for each sequence in k
      5- Applying the TCN model on the sequence in the fold k and randomisation r of the raw dataset
      6- Extract the elaborated features from the final residual block (RES):
      INPUT>RES1>RES2>RES3>elaborated features
    end
  end
end
PERFORMANCE ESTIMATION
for r=1:R
  for k=1:K
    7- Define the present fold k as test set (raw data): T
    8- Define the remaining folds as design set: V
    for p=1:P
      for v=1:V
        9- Define the present fold v as validation set (raw data)
        10- Train the recurrent model with parameter p on the remaining folds (elaborated features)
        11- Combine the optimised TCN with the trained recurrent model
        INPUT>RES1>RES2>RES3>LSTM>Fully connected>Softmax>OUTPUT
        for each sequence in v
          12- Test the complete model with parameter p on the sequence
        end
        13- Save the performance for the entire validation set v
      end
      14- Average the performance across V
      15- Save the validation result (matrix P x K x R)
    end
    16- Find the optimal p* for the recurrent model for the iteration k and repetition r
    17- Train the optimised recurrent model with parameter p* on the design set V (elaborated features)
    18- Save the performance for the train set T (matrix K x R) < general train performance
    19- Combine the optimised TCN with the optimised recurrent model
    INPUT>RES1>RES2>RES3>LSTM>Fully connected>Softmax>OUTPUT
    for each sequence in T
    🕒 START
      20- Test the complete model on the sequence in T<test performance
    🕒 STOP< temporal cost
      21- Save the computational time for the sequence in T
    end
    22- Average the performance across the entire test set T
    23- Average the computational time for the entire test set T
    end
    24- Save the performance for the test set T (matrix K x R) < general test performance
    25- Save the average computational time for the test set T (matrix K x R) < general temporal cost
  end
FINAL PARAMETER TUNING ON THE ENTIRE DATASET
  for p=1:P
    for k=1:K
      26- Define the present fold k as test set (raw data)
      27- Train the recurrent model with parameter p on the remaining folds (elaborated features)
      28- Combine the optimised TCN with the trained recurrent model
      INPUT>RES1>RES2>RES3>LSTM>Fully connected>Softmax>OUTPUT
      for each sequence in k
        29- Test the complete model on the sequence in k
      end
      30- Save the performance for the entire test set k
    end
    31- Average the performance across K
    32- Save the test result (matrix P x R)
  end
end
FINAL TRAINING OF THE MODEL
33- Choose the best parameter p** with the best average performance across all the repetitions R
34- Choose the best randomisation r* of the dataset which achieved the best result across all the
parameters P
35- Train the final recurrent model on the entire dataset (elaborated features) randomised r* with the
optimal parameter p**
36- Combine the optimised TCN with the final recurrent model
```

3.2.3 Evaluation of the networks' performance and statistical analysis.

The results obtained by the three models (LSTM, TCN, HYB) have been analysed separately for SP and CT trials. For every repetition, the expected and predicted outputs obtained from all the STS sequences across the different test folds were serialised and used to build the respective confusion matrixes. Concurrently, the overall accuracies obtained in the performance estimation on the test sets were registered during both the final training and the testing of the models, together with the precision and recall performance metrics obtained for each STS phase. Descriptive statistics for the accuracy values were reported as medians and interquartile ranges (IQR), and their distributions were graphically displayed with histogram plots. The degree of difference between distributions was tested using a two-sided Wilcoxon rank-sum test and the direct comparison between the three models across all the performance metrics was displayed using boxplots. Outliers were defined as those values that were more than 1.5 IQR distant from the 75th and 25th percentiles. Since the TCN and the HYB models were trained and tested on the same dataset (in terms of composition of folds for every repetition), the McNemar[24] test was used to assess two hypotheses:

"The HYB model is a more accurate classifier than the TCN model in predicting the STS motion patterns across the test folds T"

"The HYB model is a less accurate classifier than the TCN model in predicting the STS motion patterns across the test fold T"

For every coupled prediction in the T fold across every randomisation of the dataset. The test was implemented using the pre-defined function included in the MATLAB suite [239]. Moreover, to evaluate the real-time applicability of the HYB architecture, the prediction time required for the classification of one STS sequence was averaged across the entire fold for all the test sets.

[24] The McNemar's test is a well-known statistical test to analyze statistical significance of the differences in classifier performances [238]. It is applied to a contingency table, the cells of which include the number of samples correctly and incorrectly identified by both methods, the number of samples only classified correctly by one method.

		Model 2		
		Correct	Incorrect	
Model 1	Correct	n_{11}	n_{12}	$n_{1\bullet}$
	Incorrect	n_{21}	n_{22}	$n_{2\bullet}$
		$n_{\bullet 1}$	$n_{\bullet 2}$	n_{test}

A two-sided test for comparing the accuracy of the two models evaluate the null hypothesis that the two marginal probabilities for each outcome are the same $n_{1\blacksquare} = n_{\blacksquare 1}$and $n_{2\blacksquare} = n_{\blacksquare 2}$. Thus the null hypothesis can be written as:

$$H_0 \colon \frac{n_{12}}{n} = \frac{n_{21}}{n}$$

Where n_{ij}indicates the number of observation misclassified by method j but classified correctly by method i. For two-sided tests, the test statistic is defined as:

$$t_{\alpha 2}^{*} \colon \frac{(n_{12} - n_{21})^2}{n_{12} + n_{21}}$$

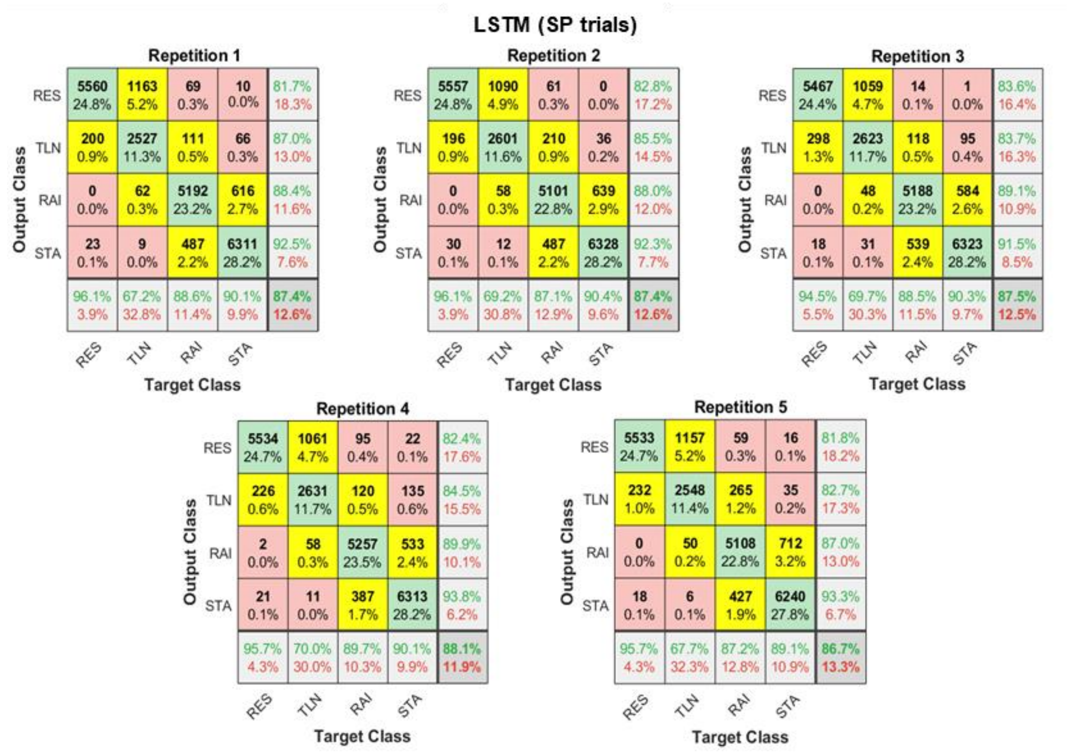
If $1 - F_{\chi 2}(t_2^*; m) < \alpha$, where F is the χ_m^2 cdf evaluated at x, then H_0 is rejected.

3.3 Results

Model	Dataset	Parameter	Optimised Value
LSTM	SP	Number of hidden units	165
		Learning rate of the training algorithm	0.028
	CT	Number of hidden units	130
		Learning rate of the training algorithm	0.007
TCN	SP	Size of the convolutional filters	5
		Learning rate of the training algorithm	0.002
	CT	Size of the convolutional filters	4
		Learning rate of the training algorithm	0.136
HYB	SP	Size of the convolutional filters	5
		Number of hidden units	350
		Learning rate of the training algorithm	0.006
	CT	Size of the convolutional filters	4
		Number of hidden units	178
		Learning rate of the training algorithm	0.002

Table 3.4: Optimised parameters for every architecture tuned over the SP and the CT trials.

The final implementations of the LSTM, the TCN, and the HYB architecture with the respective optimised parameters are reported in table 3.4. The tuning of the convolutional networks defined a filter size of 5 time-steps with a resulting causal receptive field of 6.1 s for the SP trials, and 4 time-steps with a causal receptive field of 4.6 s for the CT trials. The average prediction times necessary for the classification of an entire STS movement sequence, obtained across all the test folds were 29 ms [28 ms - 33 ms] for the SP trials and 30 ms [28 ms - 34 ms] for the CT trials. The confusion matrixes for every repetition of the SNCV process are displayed in Figures 3.13 to 3.18 and the accuracy values obtained during the train and test performance estimation of the selected architectures are described in Table 3.5, with their distributions portrayed in figure 3.19 to 3.24. For both CT and SP trials the HYB model was significantly different, in terms of accuracy, compared to the other two models ($p<0.001$). Conversely, the results obtained by the TCN and the LSTM models were not dissimilar between each other in CT trials, however, in SP trials, their performance differed significantly ($p<0.001$) [Tab3.6].


LSTM (SP trials)

Repetition 1

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	5560 24.8%	1163 5.2%	69 0.3%	10 0.0%	81.7% 18.3%
TLN	200 0.9%	2527 11.3%	111 0.5%	66 0.3%	87.0% 13.0%
RAI	0 0.0%	62 0.3%	5192 23.2%	616 2.7%	88.4% 11.6%
STA	23 0.1%	9 0.0%	487 2.2%	6311 28.2%	92.5% 7.6%
	96.1% 3.9%	67.2% 32.8%	88.6% 11.4%	90.1% 9.9%	87.4% 12.6%

Repetition 2

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	5557 24.8%	1090 4.9%	61 0.3%	0 0.0%	82.8% 17.2%
TLN	196 0.9%	2601 11.6%	210 0.9%	36 0.2%	85.5% 14.5%
RAI	0 0.0%	58 0.3%	5101 22.8%	639 2.9%	88.0% 12.0%
STA	30 0.1%	12 0.1%	487 2.2%	6328 28.2%	92.3% 7.7%
	96.1% 3.9%	69.2% 30.8%	87.1% 12.9%	90.4% 9.6%	87.4% 12.6%

Repetition 3

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	5467 24.4%	1059 4.7%	14 0.1%	1 0.0%	83.6% 16.4%
TLN	298 1.3%	2623 11.7%	118 0.5%	95 0.4%	83.7% 16.3%
RAI	0 0.0%	48 0.2%	5188 23.2%	584 2.6%	89.1% 10.9%
STA	18 0.1%	31 0.1%	539 2.4%	6323 28.2%	91.5% 8.5%
	94.5% 5.5%	69.7% 30.3%	88.5% 11.5%	90.3% 9.7%	87.5% 12.5%

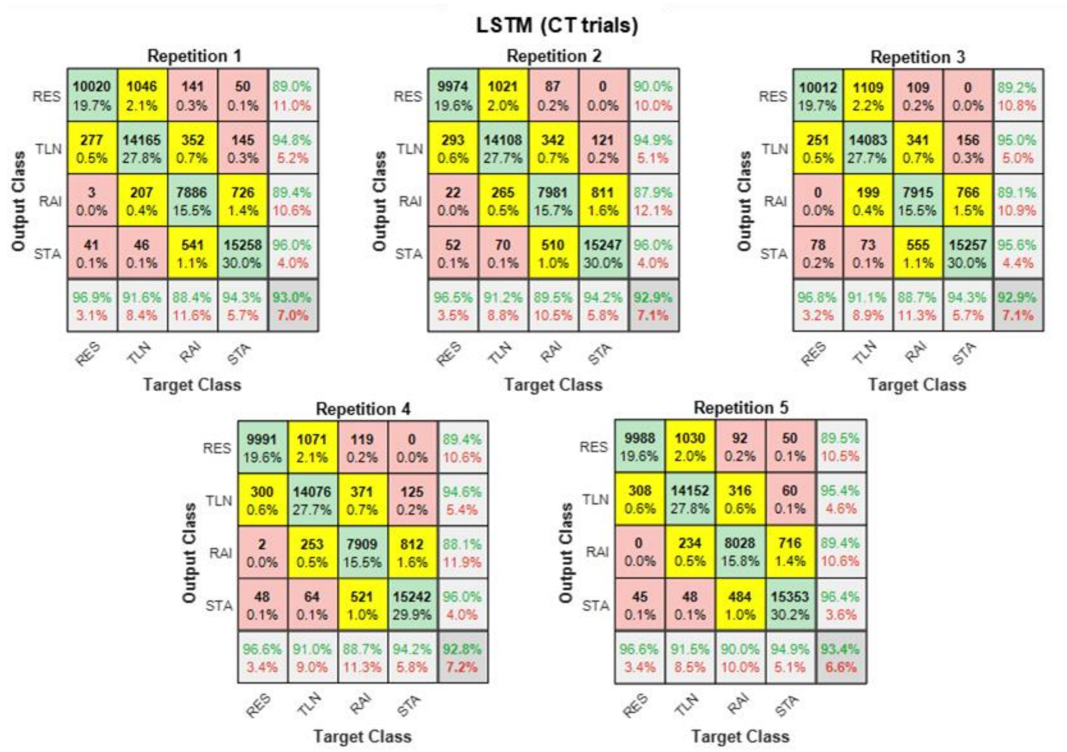
Repetition 4

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	5534 24.7%	1061 4.7%	95 0.4%	22 0.1%	82.4% 17.6%
TLN	226 0.6%	2631 11.7%	120 0.5%	135 0.6%	84.5% 15.5%
RAI	2 0.0%	58 0.3%	5257 23.5%	533 2.4%	89.9% 10.1%
STA	21 0.1%	11 0.0%	387 1.7%	6313 28.2%	93.8% 6.2%
	95.7% 4.3%	70.0% 30.0%	89.7% 10.3%	90.1% 9.9%	88.1% 11.9%

Repetition 5

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	5533 24.7%	1157 5.2%	59 0.3%	16 0.1%	81.8% 18.2%
TLN	232 1.0%	2548 11.4%	265 1.2%	35 0.2%	82.7% 17.3%
RAI	0 0.0%	50 0.2%	5108 22.8%	712 3.2%	87.0% 13.0%
STA	18 0.1%	6 0.1%	427 1.9%	6240 27.8%	93.3% 6.7%
	95.7% 4.3%	67.7% 32.3%	87.2% 12.8%	89.1% 10.9%	86.7% 13.3%

Figure 3.13: Confusion matrixes for the LSTM architecture obtained by the prediction on SP trials across all the test folds, for each repetition of the SNCV


LSTM (CT trials)

Repetition 1

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	10020 19.7%	1046 2.1%	141 0.3%	50 0.1%	89.0% 11.0%
TLN	277 0.5%	14165 27.8%	352 0.7%	145 0.3%	94.8% 5.2%
RAI	3 0.0%	207 0.4%	7886 15.5%	726 1.4%	89.4% 10.6%
STA	41 0.1%	46 0.1%	541 1.1%	15258 30.0%	96.0% 4.0%
	96.9% 3.1%	91.6% 8.4%	88.4% 11.6%	94.3% 5.7%	93.0% 7.0%

Repetition 2

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	9974 19.6%	1021 2.0%	87 0.2%	0 0.0%	90.0% 10.0%
TLN	293 0.6%	14108 27.7%	342 0.7%	121 0.2%	94.9% 5.1%
RAI	22 0.0%	265 0.5%	7981 15.7%	811 1.6%	87.9% 12.1%
STA	52 0.1%	70 0.1%	510 1.0%	15247 30.0%	96.0% 4.0%
	96.5% 3.5%	91.2% 8.8%	89.5% 10.5%	94.2% 5.8%	92.9% 7.1%

Repetition 3

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	10012 19.7%	1109 2.2%	109 0.2%	0 0.0%	89.2% 10.8%
TLN	251 0.5%	14083 27.7%	341 0.7%	156 0.3%	95.0% 5.0%
RAI	0 0.0%	199 0.4%	7915 15.5%	766 1.5%	89.1% 10.9%
STA	78 0.2%	73 0.1%	555 1.1%	15257 30.0%	95.6% 4.4%
	96.8% 3.2%	91.1% 8.9%	88.7% 11.3%	94.3% 5.7%	92.9% 7.1%

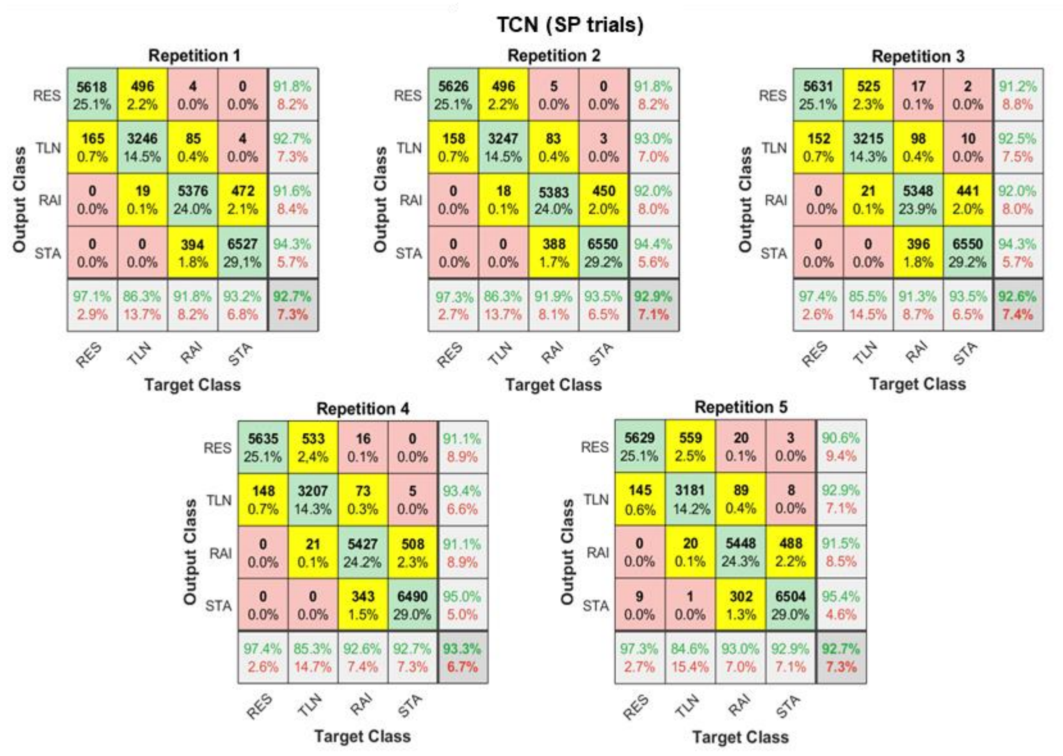
Repetition 4

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	9991 19.6%	1071 2.1%	119 0.2%	0 0.0%	89.4% 10.6%
TLN	300 0.6%	14076 27.7%	371 0.7%	125 0.2%	94.6% 5.4%
RAI	2 0.0%	253 0.5%	7909 15.5%	812 1.6%	88.1% 11.9%
STA	48 0.1%	64 0.1%	521 1.0%	15242 29.9%	96.0% 4.0%
	96.6% 3.4%	91.0% 9.0%	88.7% 11.3%	94.2% 5.8%	92.8% 7.2%

Repetition 5

Output Class \ Target Class	RES	TLN	RAI	STA	
RES	9988 19.6%	1030 2.0%	92 0.2%	50 0.1%	89.5% 10.5%
TLN	308 0.6%	14152 27.8%	316 0.6%	60 0.1%	95.4% 4.6%
RAI	0 0.0%	234 0.5%	8028 15.8%	716 1.4%	89.4% 10.6%
STA	45 0.1%	48 0.1%	484 1.0%	15353 30.2%	96.4% 3.6%
	96.6% 3.4%	91.5% 8.5%	90.0% 10.0%	94.9% 5.1%	93.4% 6.6%

Figure 3.14: Confusion matrixes for the LSTM architecture obtained by the prediction on CT trials across all the test folds, for each repetition of the SNCV



TCN (SP trials)

Repetition 1

	RES	TLN	RAI	STA	
RES	5618 25.1%	496 2.2%	4 0.0%	0 0.0%	91.8% 8.2%
TLN	165 0.7%	3246 14.5%	85 0.4%	4 0.0%	92.7% 7.3%
RAI	0 0.0%	19 0.1%	5376 24.0%	472 2.1%	91.6% 8.4%
STA	0 0.0%	0 0.0%	394 1.8%	6527 29,1%	94.3% 5.7%
	97.1% 2.9%	86.3% 13.7%	91.8% 8.2%	93.2% 6.8%	92.7% 7.3%

Repetition 2

	RES	TLN	RAI	STA	
RES	5626 25.1%	496 2.2%	5 0.0%	0 0.0%	91.8% 8.2%
TLN	158 0.7%	3247 14.5%	83 0.4%	3 0.0%	93.0% 7.0%
RAI	0 0.0%	18 0.1%	5383 24.0%	450 2.0%	92.0% 8.0%
STA	0 0.0%	0 0.0%	388 1.7%	6550 29.2%	94.4% 5.6%
	97.3% 2.7%	86.3% 13.7%	91.9% 8.1%	93.5% 6.5%	92.9% 7.1%

Repetition 3

	RES	TLN	RAI	STA	
RES	5631 25.1%	525 2.3%	17 0.1%	2 0.0%	91.2% 8.8%
TLN	152 0.7%	3215 14.3%	98 0.4%	10 0.0%	92.5% 7.5%
RAI	0 0.0%	21 0.1%	5348 23.9%	441 2.0%	92.0% 8.0%
STA	0 0.0%	0 0.0%	396 1.8%	6550 29.2%	94.3% 5.7%
	97.4% 2.6%	85.5% 14.5%	91.3% 8.7%	93.5% 6.5%	92.6% 7.4%

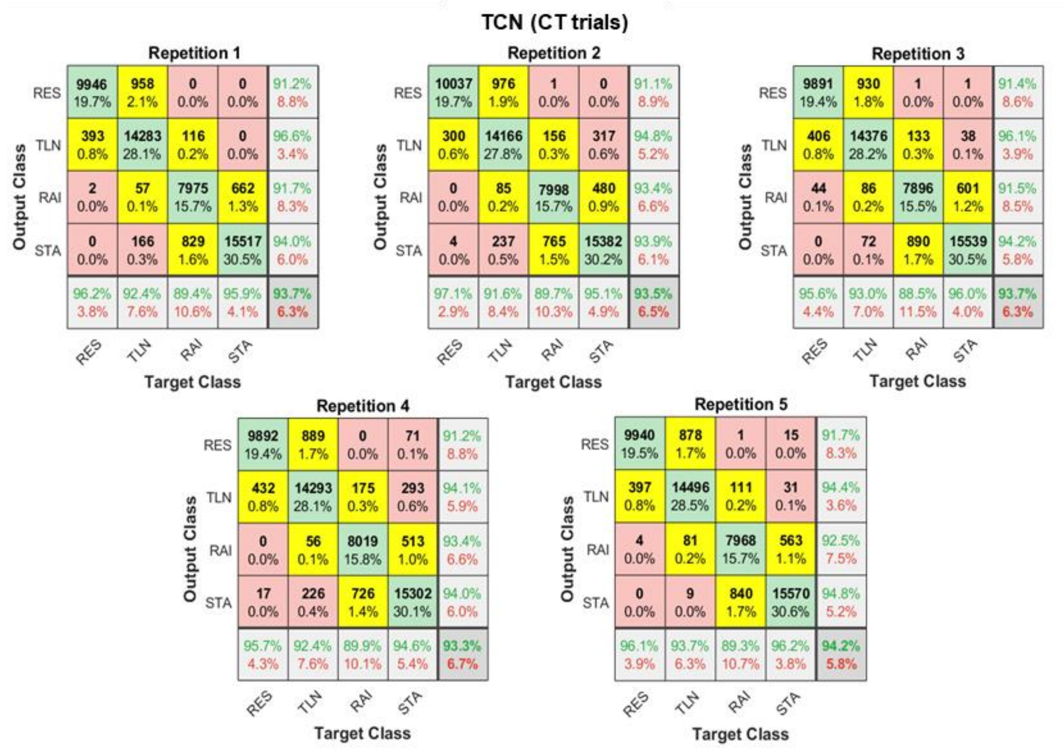
Repetition 4

	RES	TLN	RAI	STA	
RES	5635 25.1%	533 2,4%	16 0.1%	0 0.0%	91.1% 8.9%
TLN	148 0.7%	3207 14.3%	73 0.3%	5 0.0%	93.4% 6.6%
RAI	0 0.0%	21 0.1%	5427 24.2%	508 2.3%	91.1% 8.9%
STA	0 0.0%	0 0.0%	343 1.5%	6490 29.0%	95.0% 5.0%
	97.4% 2.6%	85.3% 14.7%	92.6% 7.4%	92.7% 7.3%	93.3% 6.7%

Repetition 5

	RES	TLN	RAI	STA	
RES	5629 25.1%	559 2.5%	20 0.1%	3 0.0%	90.6% 9.4%
TLN	145 0.6%	3181 14.2%	89 0.4%	8 0.0%	92.9% 7.1%
RAI	0 0.0%	20 0.1%	5448 24.3%	488 2.2%	91.5% 8.5%
STA	9 0.0%	1 0.0%	302 1.3%	6504 29.0%	95.4% 4.6%
	97.3% 2.7%	84.6% 15.4%	93.0% 7.0%	92.9% 7.1%	92.7% 7.3%

Figure 3.15: Confusion matrixes for the TCN architecture obtained by the prediction on SP trials across all the test folds, for each repetition of the SNCV



TCN (CT trials)

Repetition 1

	RES	TLN	RAI	STA	
RES	9946 19.7%	958 2.1%	0 0.0%	0 0.0%	91.2% 8.8%
TLN	393 0.8%	14283 28.1%	116 0.2%	0 0.0%	96.6% 3.4%
RAI	2 0.0%	57 0.1%	7975 15.7%	662 1.3%	91.7% 8.3%
STA	0 0.0%	166 0.3%	829 1.6%	15517 30.5%	94.0% 6.0%
	96.2% 3.8%	92.4% 7.6%	89.4% 10.6%	95.9% 4.1%	93.7% 6.3%

Repetition 2

	RES	TLN	RAI	STA	
RES	10037 19.7%	976 1.9%	1 0.0%	0 0.0%	91.1% 8.9%
TLN	300 0.6%	14166 27.8%	156 0.3%	317 0.6%	94.8% 5.2%
RAI	0 0.0%	85 0.2%	7998 15.7%	480 0.9%	93.4% 6.6%
STA	4 0.0%	237 0.5%	765 1.5%	15382 30.2%	93.9% 6.1%
	97.1% 2.9%	91.6% 8.4%	89.7% 10.3%	95.1% 4.9%	93.5% 6.5%

Repetition 3

	RES	TLN	RAI	STA	
RES	9891 19.4%	930 1.8%	1 0.0%	1 0.0%	91.4% 8.6%
TLN	406 0.8%	14376 28.2%	133 0.3%	38 0.1%	96.1% 3.9%
RAI	44 0.1%	86 0.2%	7896 15.5%	601 1.2%	91.5% 8.5%
STA	0 0.0%	72 0.1%	890 1.7%	15539 30.5%	94.2% 5.8%
	95.6% 4.4%	93.0% 7.0%	88.5% 11.5%	96.0% 4.0%	93.7% 6.3%

Repetition 4

	RES	TLN	RAI	STA	
RES	9892 19.4%	889 1.7%	0 0.0%	71 0.1%	91.2% 8.8%
TLN	432 0.8%	14293 28.1%	175 0.3%	293 0.6%	94.1% 5.9%
RAI	0 0.0%	56 0.1%	8019 15.8%	513 1.0%	93.4% 6.6%
STA	17 0.0%	226 0.4%	726 1.4%	15302 30.1%	94.0% 6.0%
	95.7% 4.3%	92.4% 7.6%	89.9% 10.1%	94.6% 5.4%	93.3% 6.7%

Repetition 5

	RES	TLN	RAI	STA	
RES	9940 19.5%	878 1.7%	1 0.0%	15 0.0%	91.7% 8.3%
TLN	397 0.8%	14496 28.5%	111 0.2%	31 0.1%	94.4% 3.6%
RAI	4 0.0%	81 0.2%	7968 15.7%	563 1.1%	92.5% 7.5%
STA	0 0.0%	9 0.0%	840 1.7%	15570 30.6%	94.8% 5.2%
	96.1% 3.9%	93.7% 6.3%	89.3% 10.7%	96.2% 3.8%	94.2% 5.8%

Figure 3.16: Confusion matrixes for the TCN architecture obtained by the prediction on CT trials across all the test folds, for each repetition of the SNCV

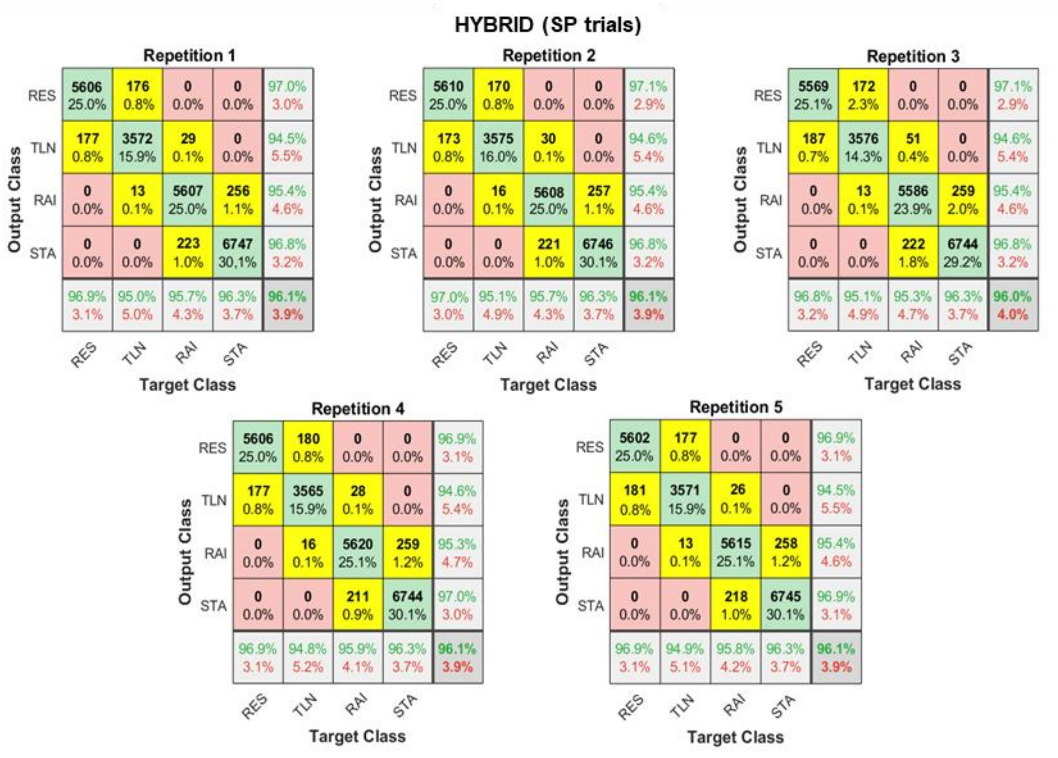


Figure 3.17: Confusion matrixes for the HYB architecture obtained by the prediction on SP trials across all the test folds, for each repetition of the SNCV

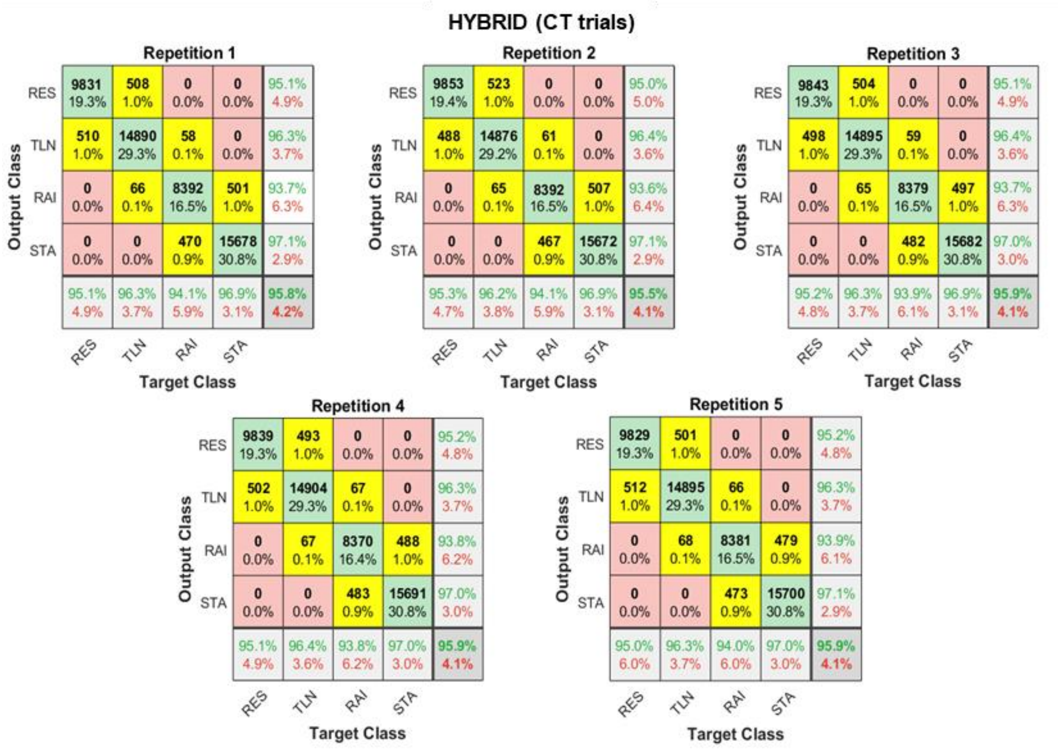


Figure 3.18: Confusion matrixes for the HYB architecture obtained by the prediction on CT trials across all the test folds, for each repetition of the SNCV

Model	Dataset	Training		Test	
		Median	IQR	Median	IQR
LSTM	SP	94.83	93.93 - 95.48	89.10	86.78 - 90.24
	CT	96.28	96.12 – 96.85	94.40	93.15 – 95.51
TCN	SP	97.44	96.96 – 97.85	92.77	91.31 – 93.86
	CT	96.43	96.14 – 96.94	94.84	94.25 – 95.67
HYB	SP	97.56	97.06 - 97.91	96.09	95.37 - 96.56
	CT	92.28	91.68 – 93.58	95.74	95.39 – 96.21

Table 3.5: Train and test Accuracy values (median, IQR) obtained by the three architectures on SP and CT trials in the performance estimation of the repeated SNCV.

This was supported also by the boxplots, which underlined how the HYB architecture yielded the best results in terms of performance metrics, confidence interval, and the number of outliers.

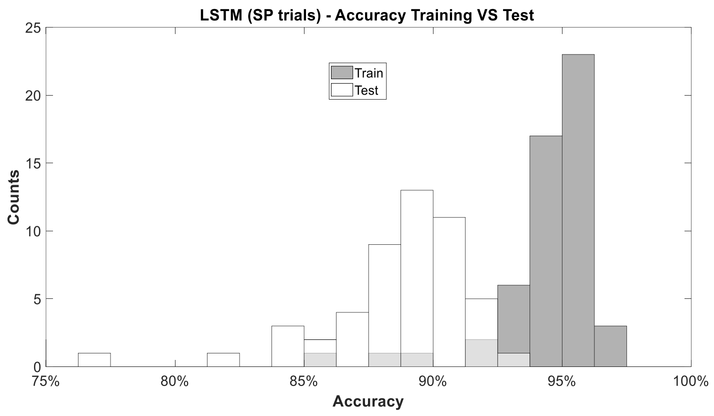


Figure 3.19: Accuracy values for the SP trials obtained by the LSTM architecture in the train and test performance estimation

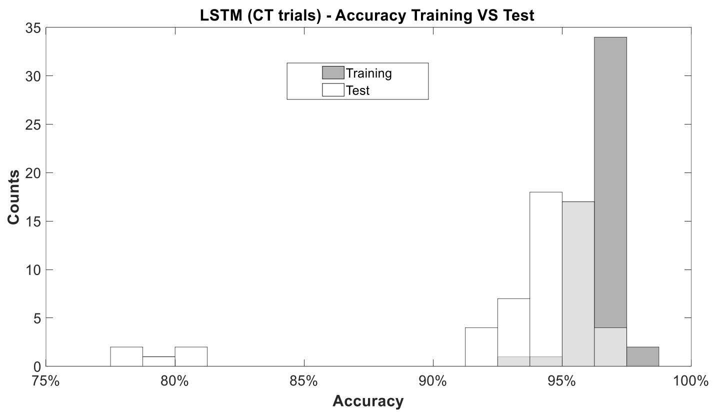


Figure 3.20: Accuracy values for the CT trials obtained by the LSTM architecture in the train and test performance estimation

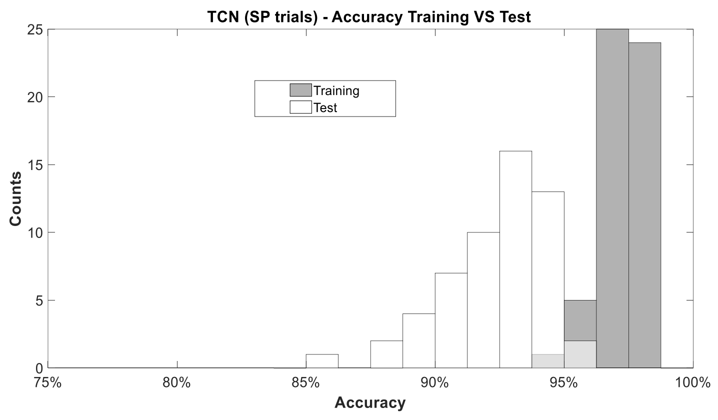


Figure 3.21: Accuracy values for the SP trials obtained by the TCN architecture in the train and test performance estimation

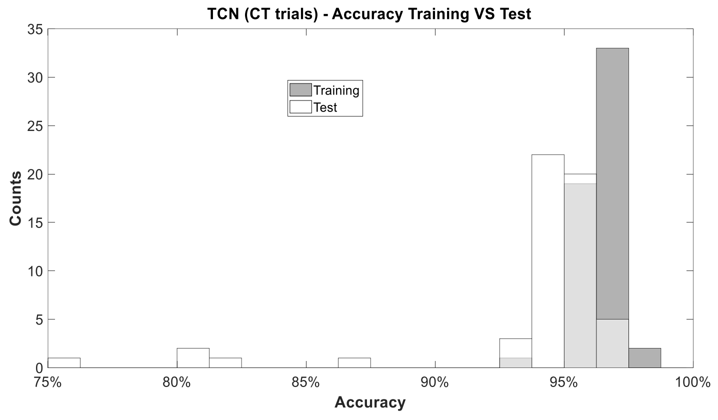


Figure 3.22: Accuracy values for the CT trials obtained by the TCN architecture in the train and test performance estimation

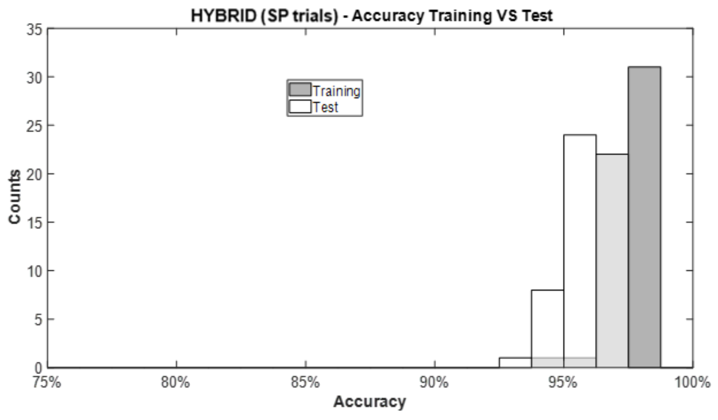


Figure 3.23: Accuracy values for the SP trials obtained by the HYB architecture in the train and test performance estimation

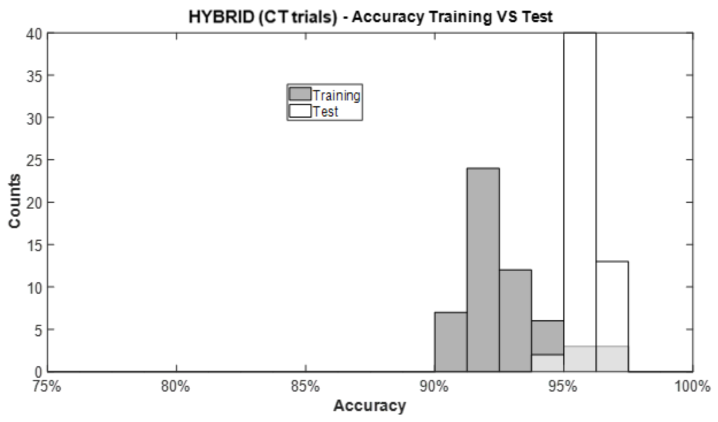


Figure 3.24: Accuracy values for the CT trials obtained by the HYB architecture in the train and test performance estimation

	HYB	TCN	LSTM
HYB		p=4.2e-08	p=9.9e-09
TCN	p=5.4e-18		p=0.2
LSTM	p=1.6e-19	p=4.7e-13	

Table 3.6: Results of the two-sided Wilcoxon rank-sum test. The shaded area represents the values obtained from the SP trials. The white area represents the values from the CT trials.

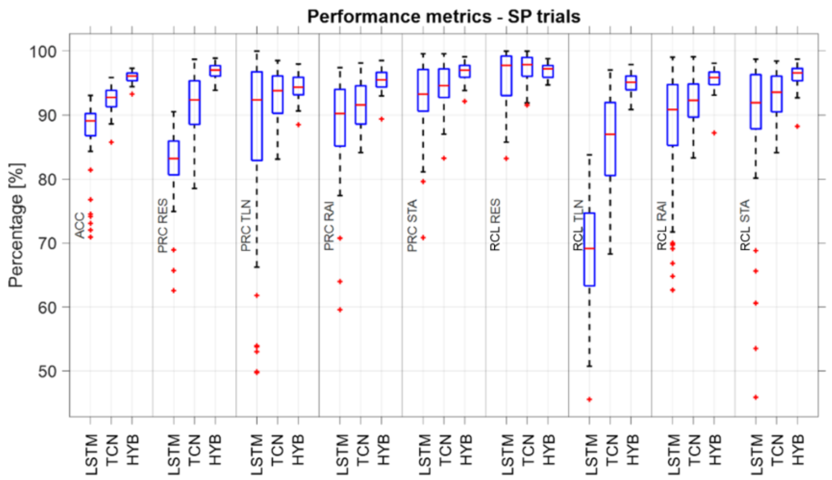


Figure 3.25: Box plots of the distributions of the performance metrics obtained over the SP trials from the HYB, TCN, and LSTM models across all the test folds

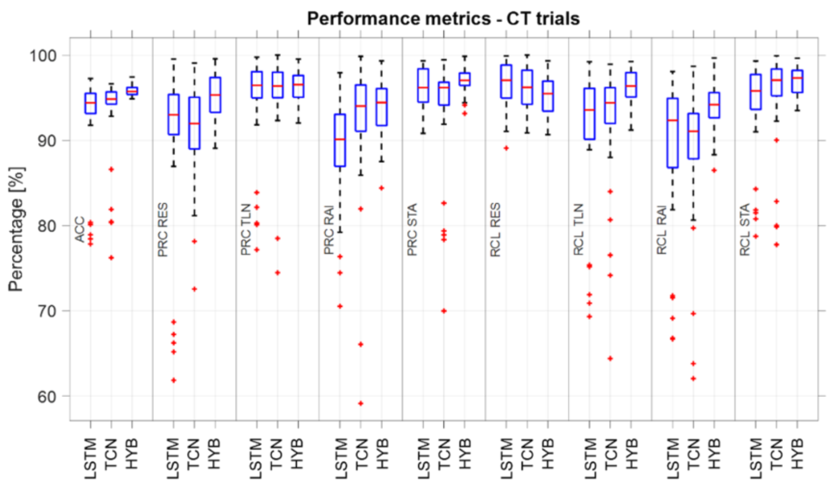


Figure 3.26: Box plots of the distributions of the performance metrics obtained over the CT trials from the HYB, TCN, and LSTM models across all the test folds

Fold	**Repetition 1**		**Repetition 2**		**Repetition 3**		**Repetition 4**		**Repetition 5**	
	p+	**p-**	**p+**	**p-**	**p+**	**p-**	**p+**	**p-**	**p+**	**p-**
1	7.4e-09‡	1.0	3.7e-11‡	1.0	8.8e-07‡	1.0	7.1e-17‡	1,0	1.1e-08‡	1.0
2	2.2e-22‡	1.0	5.1e-21‡	1.0	4.4e-22‡	1.0	2.7e-15‡	1,0	1.1e-08‡	1.0
3	1.2e-10‡	1.0	1.2e-04‡	1.0	1.8e-21‡	1.0	4.1e-06‡	1,0	6.5e-32‡	1.0
4	7.3e-06‡	1.0	1.6e-10‡	1.0	2.1e-07‡	1.0	6.1e-25‡	1,0	4.7e-16‡	1.0
5	6.9e-12‡	1.0	9.3e-10‡	1.0	1.5e-12‡	1.0	1.1e-07‡	1,0	2.6e-09‡	1.0
6	5.8e-21‡	1.0	6.2e-13‡	1.0	9.6e-06‡	1.0	2,5e-05‡	1,0	3.8e-12‡	1.0
7	3.4e-18‡	1.0	6.9e-09‡	1.0	3.3e-13‡	1.0	9.3e-10‡	1,0	4.1e-05‡	1.0
8	1.5e-10‡	1.0	2.5e-09‡	1.0	7.8e-11‡	1.0	6.9e-13‡	1,0	1.2e-12‡	1.0
9	1.4e-09‡	1.0	2.2e-10‡	1.0	1.7e-11‡	1.0	3.8e-10‡	1,0	9.5e-16‡	1.0
10	3.8e-04‡	1.0	3.2e-10‡	1.0	2.2e-16‡	1.0	3.7e-11‡	1,0	2.2e-12‡	1.0
11	6.4e-16‡	1.0	1.6e-07‡	1.0	1.1e-11‡	1.0	2.0e-19‡	1,0	1.3e-07‡	1.0

Table 3.7: Results of the McNemar test in the comparisons between the HYB and the TCN architectures over the SP trials. The columns p+ test the hypothesis HYB > TCN. The columns p- test the hypothesis HYB < TCN. A p-value less than 0,05 confirms the relative hypothesis. The superscripts ‡indicates a value of p< 0,01. The superscripts †indicates a value of p< 0,05.

Fold	**Repetition 1**		**Repetition 2**		**Repetition 3**		**Repetition 4**		**Repetition 5**	
	p+	**p-**	**p+**	**p-**	**p+**	**p-**	**p+**	**p-**	**p+**	**p-**
1	2.1e-09‡	0.9	1.1e-06‡	1.0	1.2e-12‡	1.0	0.1	0.8	1.3e-09‡	1.0
2	4.3e-07‡	0.9	1.4e-08‡	1.0	1.2e-14‡	1.0	5.2e-17‡	1.0	1.0e-02†	1.0
3	0.9	0.1	1.1e-07‡	1.0	4.6e-07‡	1.0	3.2e-07‡	1.0	0.8	1.0
4	6.9e-02	0.9	8.7e-02	1.0	1.3e-02†	1.0	2.1e-02†	1.0	3.3e-06‡	1.0
5	2.8e-04‡	1.0	1.6e-08‡	1.0	1.1e-03‡	1.0	3.2e-02†	1.0	1.7e-05‡	1.0
6	6.2e-02	0.9	1.6e-206‡	1.0	3.5e-02†	1.0	3.5e-05‡	1.0	2.3e-04‡	1.0
7	2.2-02†	1.0	1.3e-04‡	1.0	0.9	9.4e-04‡	9.3e-04‡	1.0	5.4e-05‡	1.0
8	6.8e-02	0.9	0.6	0.4	6.9e-04‡	1.0	2.1e-04‡	1.0	1.9e-113‡	1.0
9	2.2e-02†	1.0	8.2e-09‡	1.0	1.4e-154‡	1.0	9.9e-204‡	1.0	0.8	0.2
10	1.3e-205‡	1.0	0.9	0.1	4.4e-09‡	1.0	5.5e-04‡	1.0	0.1	0.9
11	1.6e-08‡	1.0	0.1	0.8	6.3e-23‡	1.0	1,.7e-18‡	1.0	3.3e-09‡	1.0

Table 3.8: Results of the McNemar test in the comparisons between the HYB and the TCN architectures over the CT trials. The columns p+ test the hypothesis HYB > TCN. The columns p- test the hypothesis HYB < TCN. A p-value less than 0,05 confirms the relative hypothesis. The superscripts ‡indicates a value of p< 0,01. The superscripts †indicates a value of p< 0,05.

Also, the evidence from the McNemar test was in accordance with the above-described results. On the one hand, the HYB architecture was more accurate than the TCN model across all the test folds in SP trials [Tab.3.7] and the majority of the test folds in CT trials [Tab.3.8]. On the other hand, the HYB model did not perform worse than the TCN (except for one fold in a singular randomisation of the dataset). This was evident in the analysis of the classification outputs related to those STS sequences which reached a low level of performance in the test folds: whereas the LSTM and TCN models poorly classified the STS phases, the HYB model was visibly more accurate in respect to the ground truth data.

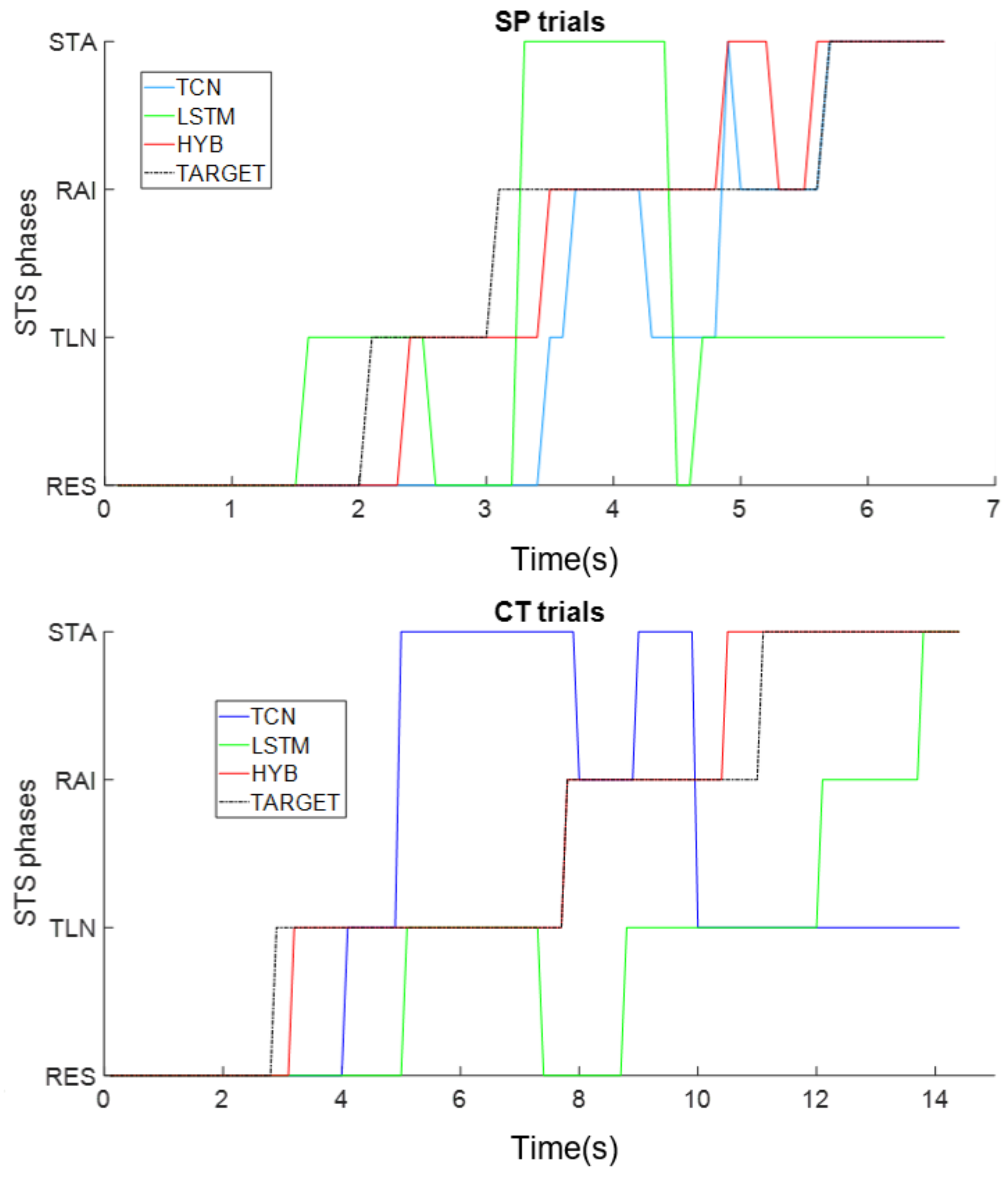


Figure 3.27: Classification output of the HYB model compared to the LSTM and TCM models over two critical STS sequences (one for each movement paradigm: SP and CT trials)

3.4 Discussion and Conclusions

A hybrid-approach architecture based on deep-learning was evaluated to achieve a finer real-time categorisation of the STS movement, relying on the data acquired from a single inertial sensor placed on the chest. The temporal dependencies within the sequence of movements were exploited to improve the predictive accuracy across all the STS phases, using both convolutional and recurrent approaches. Specifically, convolutional networks have been already used successfully in literature as feature filters able to extract more significant characteristics from raw data improving the performance of recurrent models in daily activity identification [231], hyperspectral image classification [240], and hand gestures recognition [241]. In this work, the TCN architecture described by Bai [207] was implemented to extract more discriminant temporal features from the raw accelerometric signal, to enhance the predictive ability of the LSTM network. Thus, a comparative study was carried on to highlight the advantages of the proposed architecture over the single models. The results obtained pointed out the better accuracy and stability of the HYB architecture across all the phases of the STS motion pattern. More importantly, the number of outliers in the boxplots and the distributions of accuracy obtained across the different test folds pointed out the better generalisation performance of the HYB model. Whereas TCN and LSTM model, separately, showed low accuracy on some specific test folds, their combination proved to be significantly more robust against the intrinsic between/within-subjects variability of the STS and the uncertainty related to the possible systematic errors due to the experimental setting. This result represents a key factor in the context of a possible clinical implementation of the proposed architecture, as it estimates the reliability of the system in a common-use scenario: where the movement and the mounting position of the inertial sensor can slightly vary across different executions [242]. This evidence was found in both SP and CT trials, suggesting that a hybrid approach, based on convolutional feature extractors, could be a viable solution to classify the STS motion pattern independently from the condition of execution of the task. In this sense, such architecture might provide reliable and finer movement classification also in all those populations that are characterised by different motor strategies from those expressed by the healthy young people recruited for this work.

Final remarks and future developments

The recent technological advances are leading the global digital development in the health-care field, opening novel solutions for both clinicians and patients [243]. Such innovations aim at maximizing the efficacy of health treatments, allowing to overcome the constraints of distance, location, and time through the use of different technology modalities like smartphone apps, sensors, artificial intelligence, video, social media, and messenger platforms. It has been estimated that, just in the United States, applying digital solutions to the annual practice of every primary care physician would result in a saving of 5 minutes per patient encounter, which translates into a general annual value of more than 7 billion $ [243]. Nowadays, digital health-care is starting to be considered more than a simple accessory, and several specific applications have already been approved and prescribed by doctors in their clinical practice. For instance, real-time data analytic, artificial intelligence, and sensor technology are changing the prospect of health and biomedical research obtaining successful results in image-based diagnosis, genome interpretation, patient monitoring, clinical outcome prediction, and in a plethora of applications aimed at inferring the patients' status from wearable technology [244]. The results highlighted in the present research represent a first step toward the development of an automated coaching system for the unsupervised rehabilitation of the STS movement. Taking inspiration from the real world, human coaches exhibit unique online behaviours which must be taken into account in the development of a motor skill learning applications. In a study from 2015, de Kok and colleagues [133] presented a virtual reality environment capable of reading and analysing users' movements and comprised of a coaching avatar that can generate appropriate instructions as the motor skill is performed. Based on the evidence from real coaching interactions, their study identified four major attributes that define a realistic virtual coach:

- Intrinsic multimodality in the understanding of coachee's movements and the generation of a proper demonstration of the motor skill;
- Embedded detailed science-informed online movement analysis;
- Motivational and relevant feedback generation to maximise the learning gain and naturalness of the coach;

- Closed-loop functioning framework with low latency and incremental processing components from input to output to instruct on and correct a problematic phase of the skill at the relevant time.

While a multimodal approach would necessitate different types of sensors (increasing the overall costs and encumbrance of the system), the presented work demonstrated the possibility to accurately and reliably recognise the STS movement pattern online, relying on a single inertial sensor placed on the chest. In this way, the limited hardware requirements would ease the software development process by reducing the computational burden due to the synchronization of different devices, and at the same time, facilitate the clinical implementation of the system by exploiting the unobtrusive nature of wearable technology. Starting from the positive results achieved, future efforts will be focused on the further evaluation of the synchronous prediction capabilities of the hybrid model. The estimated classification time obtained for an entire STS movement sequence (~30 ms) must be empirically tested through an online classification task, following the same protocol presented in Chapter 2. Eventual positive results would confirm the temporal performance of the proposed architecture supporting its implementation in a closed-loop framework able to evaluate the execution of the STS and synchronously and effectively correcting the patients' movement. In this sense, there is extensive evidence in the literature that promotes the use of artificially produced sounds cues as an effective feedback mechanism to support motor skill learning [245]. From a neuroanatomical aspect, the auditory and the motor system are strictly connected, with reciprocal interactions at the spinal cord, subcortical and cortical levels [246]. It has been demonstrated that rhythmic auditory stimuli could increase the efficiency of the motor system [247], by entraining the activity of the auditory neurons with the firing pattern of the motor cortex, with a key role in movement anticipation and motor preparation [248]. More importantly, synchronous auditory feedback ameliorates error-correction mechanisms [249]–[251] and it is hypothesised that mapping of movements parameters onto different sound components through a multi-perceptual integration of congruent information enhance the internal representation of the movement [252], [253] improving its quality and the re-learning of motor skills in stroke rehabilitation [254]–[256]. An additional investigation must be aimed at understanding and design the correct auditory stimulus to be used as effective feedback for the STS movement execution. For instance, rhythmic auditory stimulation techniques are used in rehabilitation to solicit auditory-motor synchronization and promote sustained functional changes to movement by providing continuous-time references [257]–[259]. In particular, the repetitive pattern of the temporal cues generates expectation regarding the advent of a subsequent sound, allowing eventual anticipation and motor preparation with an increase in quality and precision of the movements [248]. Another implementable solution is represented by the use of sonification techniques, by transferring movement features into non-speech audio signals. Sonification refers to the mapping of physiological and physical characteristics onto psychoacoustic parameters to provide access to biomechanical information otherwise not available [260]. Sonification promotes movement control and planning by improving self-awareness of the physiological processes underlying its execution [261]. For this reason, this method has already been applied as effective feedback in sports training, to inform athletes about performance error/deviation during the execution of movements. Literature evidence indeed suggests that the availability of real-time auditory feedback enhances online error-correction mechanisms during movement execution and facilitates the learning of a new motor skill [245], [262]–[264]. Such properties could be effectively exploited in a closed-loop system for the coaching of the STS to contrast avoidance and maladaptive behaviours in the execution of the movement under pain or fatigue conditions. For instance, the different movement strategies adopted for the STS manoeuvre in chronic low back pain can be characterised by a cognitive-behavioral fear-avoidance paradigm [265], [266]. This model is based on the concept that pain and pain-related fear levels might lead to avoidance behaviour, which is ultimately transferred into a reduction of mobility, preventing the further incitement of pain in damaged tissues [267]. If retained, these habits may evolve into a chronic pain syndrome, negatively affecting patients' everyday functioning and mental state [268]. The maintenance of altered movement strategies in the STS transition has also been evidenced in patients who have undergone unilateral total knee arthroplasty (TKA) [25]. In their cross-sectional study, Farquhar and colleagues investigated the changes in STS performance after surgery

[25] Total knee arthroplasty (TKA) is one of the most cost-effective and successful surgeries performed in orthopedics. It involves different possible approaches [269] and it is often is performed to relieve the pain of end-stage knee OA following the failure of nonsurgical management .

[270]. After 3 months follow-up, the TKA group displayed an altered raising strategy, consisting of the unloading of the affected limb with the use of greater hip flexion, which resulted in higher hip extensor moments. In an early phase, this manoeuvre represents a reasonable solution to compensate for muscle weakness and pain. However, the altered strategy persisted and increased further after 1 year follow-up, despite the normalization of weight-bearing and strength, with a consequent over-stress on the uninvolved hip joints. A large hip extensor moment contributes to increased wear on the anterior portion of the femur and has been implicated in the development of hip osteoarthritis [271]. Relying on the presented observations, an unobtrusive and automated STS coaching system would be a valuable contribute to the clinical practice for treatment and prevention. Nonetheless, the use of technology in health-care remains fragmented at present due to a lack of supportive policy and regulation, unsustainable reimbursement, inefficient business models, and concerns regarding data security and privacy. A "human-machine" synergy, via a closer cooperation between clinicians and data-scientists, would allow the great amount of available data to be an efficient enabler for new knowledge and intelligence in biomedicine and healthcare [272].

Abbreviations

30CST	30-Second Chair Stand Test
ACC	Accuracy
ADL(s)	Activit(y/ies) Of Daily Living
BPTT	Backpropagation Through Time
CI	Confidence Intervals
CNN(s)	Convolutional Neural Network(s)
COM	Centre Of Mass
COP	Centre Of Pressure Of The Body
COPD	Chronic Obstructive Pulmonary Disease
COV	Coefficient Of Variation
CT	Controlled Speed
EMG	Electromyography
FTSTS	Five Time Sit-To-Stand-Test
GRF	Ground Reaction Force
HAR	Human Activity Recognition
HYB	Hybrid
ICF	International Classification Of Functioning, Disability, And Health
IID	Independent And Identically Distributed
IMU	Inertial Measurement Unit
IQR	Interquartile Range
LSTM	Long Short-Term Memory
LTD	Long-Term Depression
LTP	Long-Term Potentiation
LTSP	Long-Term Synaptic Plasticity
MEMS	Micro-Electromechanical Systems
ML	Machine Learning
MLP	Multilayer Perceptron
PPF	Paired Pulse Facilitation
PPV	Positive Predictive Value
PRC	Precision
PTP	Post Tetanic Potentiation
RAI	Raising Phase

RCL	Recall
RES	Rest Phase
RMS	Root-Mean-Square
RNN(s)	Recurrent Neural Network(s)
SCG	Scaled Conjugate Gradient
SD	Standard Deviation
SEN	Sensitivity
SNCV	Stratified Nested Cross-Validation
SP	Self-Paced Speed
SPE	Specitivity
STA	Standing Phase
STS	Sit-To-Stand
STSP	Short-Term Synaptic Plasticity
TCN	Temporal Convolutional Neural Network
TKA	Total Knee Arthroplasty
TLN	Trunk Leaning Phase
TUG	Timed Up And Go
ULoA	Upper Limits Of Agreement
UN	United Nations Organization
WHO	World Health Organization

Supplement Materials: MATLAB code and routines

In this section, the main routines are reported and commented. All the comments are introduced by a % character and include:

- A brief description of the script/function;
- Eventual input and outputs;
- Pseudocode

For the main functions a user-friendly description of the routine has been added, displaying the main inputs and outputs to provide an intuitive understanding of its processes.

S.1 TrunkMOV.m function

The following code implements a heuristic routine based on the standard deviation thresholding of the GRF to identify the initial instant of forward bending of the trunk. It takes as input the threshold constant (T) and the STS force signal (varargin) registered through a force plate and divided into 100 ms epochs (i.e. the rows of the data structure represent the number of samples for each epoch, the columns of the data structure represent the number of epochs). It gives as output the index of the sample that identifies the beginning of the movement (IndexMov).

```
% Function to detect the initiation of the Trunk movement (Initiation event). It calls the sub-function stdonepoch.m, which divide
the Signal in % epochs of 100 ms and calculates the standard deviation for each epoch.

function IndexMov=TrunkMOV(T,varargin)

% INPUT      T: Experimental threshold;
%            Varargin: Input signal divided into epochs MxN matrix where M is the number of samples in each epoch and N is the %
             number of epochs;
% OUPUT      IndexMov: Initiation event;

Signal=varargin{1,1}; % Assign the input parameter to Signal
std_arr=stdonepoch(Signal); % For each epoch calculate the standard deviation
std_arr(2,:)=movmean(std_arr(2,:),10); % Moving average of the standard deviation across epochs
BsLn=abs(mean(std_arr(2,1:10))); % Baseline reference = standard deviation on first ten epochs
std_arr(2,:)=std_arr(2,:)./BsLn; % Normalisation of the averaged standard deviation in respect to the baseline
IndexMov=std_arr(1,find(std_arr(2,:)>=T,1)); % The first index surpassing the threeshold identify the beginning of the movement
```

S.2 stdonepoch.m function

```
% Function to calculate the standard deviation across each epochs. Save the values of standard deviation and the central indexes of
% the epochs.

function std_arr=stdonepoch(Signal)

% INPUT      Signal: Input signal divided into epochs. MxN matrix where M is the number of samples in each epoch and N is the
%            number of epochs;
% OUTPUT     stdEp: Output matrix describing how the standard deviation range across the different epochs. 2xM matrix
%            , where M is the number of epochs, the first row represent the epochs indexes (number of the middle
%            sample), the second row represent the standard deviation for each epoch;
%            |__|__|__|... references indexes
%            |__|__|__|... standard deviations

stdEp=NaN(2,size(Signal,2)); % Declaration and pre-allocation of the variable
nsamples=size(Signal,1); % N°of samples per epochs
stdEp(2,:)=std(Signal,0,1); % Standard deviations per epochs
stdEp(1,:)=(1:length(Signal))*Nsamples+ceil(Nsamples/2); % Indexes per epochs
```

S.3 Bottom_Transition.m function

The following code implements a heuristic routine based on the masking of the signal from the electronic switch on the chair to identify the instants of seat-off and seat-on. It takes as input the signal from the electronic switch (Bott). It gives as output the indexes of seat-off (Bottomup) and seat-on (Bottomdown).

```
% Function to identify the SeatOFF and SeatON moments.

function [Bottomup,Bottomdown]=Bottom_Transition(Bott)

% INPUT      Bott: Filtered and sampled electronic switch signal
% OUTPUT     Bottomup/Bottomdown: SeatOFF / SeatON

Bott(Bott>-1)=0; % Signal over threshold is setted at 0
Bott(Bott<-1)=-5; % Signal under threshold is setted at -5
DerBott=diff(Bott); % Differentiation of the masked Signal
Bottomup=find(DerBott<=-1,1,'first')+1; % Find transitions
Bottomdown=find(DerBott>-(-1),1,'last');
```

S.4 SteadyStandingPoints.m function

The following code implements a heuristic routine based on the standard deviation thresholding of the GRF between the seat-off and seat-on instants to identify the period of stable upright stance. It takes as input the threshold constant (T), the STS force signal (varargin) registered through a force plate (F), the sampling rate of the force signal (fs), the temporal length (in seconds) of the epochs (ep) with which to divide the force signal and the seat-off/seat-on instants (Bottomup/Bottomup). It gives as outputs the indexes of the samples that identify the beginning and the ending of the upright stable stance (Standing/Sitting).

```
% Function to extrapolate the Standing and Sitting event from the GRF profile. It calls

function [Standing,Sitting]=SteadyStandingPoints(F,fs,ep,Bottomup,Bottomdown,T)

% INPUT      F: Force signal resampled at 50 Hz;
%            fs: Sampling frequency (50 Hz)
%            ep: 0.1 [s]
%            Bottomup: SeatOFF
%            Bottomdown: SeatON
%            T: threshold
% OUTPUT     Standing/Sitting

samples=fix(fs*ep); % N°samples x epoch
t=Bottomup:Bottomdown; % Consider just the samples between Bottomup and Bottomdown
stand2sit=F(stand:sit); % Consider just the GRF between Bottomup and Bottomdown
bstdF=std(buffer(stand2sit,samples,0,'nodelay')); % Divide the GRF in epochs
bavgT=mean(buffer(t,samples,0,'nodelay')); % Divide the samples in epochs
midpoint=round(length(bstdF)/2); % The midpoint between Bottomup and Bottomdown is considered the centre of stability
baseline=mean(bstdF(midpoint-10:midpoint+10)); % Calculate the baseline
bstdF=bstdF/baseline; % Normalise the force signal
mask=movmean(bstdF,10)>T; % Mask the force signal
Standing= bavgT(find(mask==0,1,'first')); % The first sample below the threshold is the Standing event
Sitting= bavgT(find(mask==0,1,'last')); % The last sample below the threshold is the Sitting event
```

S.5 NeuralSNCV.m script

The following script implements the Stratified Nested Cross Validation in the tuning and evaluation of a Multi-Layer Perceptron.

```
% Script to implement the Stratified Nested Cross Validation in the tuning and evaluation of a Multi-Layer Perceptron. It calls the
sub-functions Redistribute.m and ValidationLoop.m which respectively distribute the observations of the dataset equally in the
different folds, and implement the cross validation loops.

% INPUT:     The file containing:
%            - the variable DataSET with all the 0.1 s observations from the IMU sensors
%            - the variable DIVISOR with the definition of the subjects for all the 0.1 s observations in the dataset
```

```
load(fullfile(cd,'Table_Classifier.mat')); % Load of the variable DataSET, matrix (N°observations X [Features+Labels])
x=table2array(DataSET(:,1:end-1))'; % Input matrix F X N, where F is the number of features and N is the number of observations
tc=table2array(DataSET(:,end)); % Label matrix N x 1, where N is the number of observations
targets=[ ... % Creation of the dummy variable targets O X N, where O is the number of labels in the label matrix
    tc==1,... % Boolean vector defining which observation correspond to the label 1 = REST phase
    tc==2,... % Boolean vector defining which observation correspond to the label 2 = TRUNK LEANING phase
    tc==3,... % Boolean vector defining which observation correspond to the label 3 = RAISING phase
    tc==4,... % Boolean vector defining which observation correspond to the label 4 = STANDING phase
    ]';
[I, N]=size(x); % Dimensionality of DataSET
[O, ~]=size(targets);
Sbj=unique(DIVISOR); % Identifying the subjects in the dataset
loop=0; % Flag for the loop for the definition of the nested cross validation modality
while loop==0
    flag=input(['What validation model you would like to implement?\n 1 - 5 folds\n   2 - 10 folds\n   3 - LOSOCV\n\n']);
    switch flag
        case 1 % CrossValidation 5 fold
            folds = 6; % folds are +1 the definition of the cross validation to implement a nested routine:
            loop = 1; % an outer loop will consider one fold as test, the inner loop will implement the cross validation
        case 2 % CrossValidation 10 fold
            folds = 11;
            loop = 1;
        case 3 % LOSOCV (Leave One Subject Out CrossValidation
            folds = length(SBJ);
            loop = 1;
        otherwise
            loop = 0;
    end
end

% The following step is implemented to separate different subjects in every fold (avoid overfitting)
Shuffled=Sbj(randperm(numel(Sbj))); % Randomize the order of the subjects
step=floor(numel(Sbj)/folds); % Minimum number of subject per fold
division=1:step:(step*folds); % Indexes to distribute subjects in the folds
R=rem(numel(Sbj),folds); % Remaining subjects to distribute
subgroup=cell(folds,1); % Folds declaration
for t=1:folds
    subgroup{t}=Shuffled(division(t):division(t)+step-1); % Folds filling
```

```
end

% The following step is implemented to redistribute the remaining subjects according to the number of observations
if R~=0
    Remain=Shuffled(end-(R-1):end); % Remaining subjects
    subgroup=Redistribute(DIVISOR,subgroup,Remain); % Redestribute function
end

% Definition of the performance metrics
xentrval=cell(folds,1);
xentrtrn=cell(folds,1);
xentrtst=zeros(folds,1);
xentrtstT=zeros(folds,1);
best_epval=cell(folds,1);
best_eptst=zeros(folds,1);
TSToutputs=cell(folds,1);
TSTtargets=cell(folds,1);
accuracy=zeros(folds,1);
recall=zeros(folds,1);
precision=zeros(folds,1);
f1=zeros(folds,1);
OptimalN=zeros(folds,1);

Neurons=sort(randi([O,I],10,1)); % Ten different values of hidden units between input and outputs
S=cell(1,10); % Ten random states to initialise ten different set of weights
rng(0);
for s=1:10
    S{s}=rng;
    rand;
end
rng(0);

for t=1:folds
    logicalindext=cellfun(@(x)contains(DIVISOR,x), subgroup{t},'un',0); % Definition of the TestSET.
    ITST=find(any(horzcat(logicalindext{:}),2)==1) % TestSET Indexes
    IVAL=cell(1,folds-1);% Declaration of validation indexes for each validation loop
    ITRN=cell(1,folds-1);% Declaration of training indexes for each validation loop
```

```
count=1;
for v=1:folds % For each fold ...
    if t~=v % ... different from the TestSET
        logicalindexv=cellfun(@(x)contains(DIVISOR,x),subgroup{v},'un',0);
        IVAL{1,count}=find(any(... % Definition of the validation indexes
            horzcat(logicalindexv{:}),2)==1);
        ITRN{1,count}=find(~any([any(... % Definition of the training indexes
            horzcat(logicalindext{:}),2),any(...
            horzcat(logicalindexv{:}),2)],2)==1);
        count=count+1;
    end
end

perf_xentrval=zeros(10,10);
perf_xentrtrn=zeros(10,10);
perf_best_epval=zeros(10,10);

for n=1:10 % For each model described by a different number of neurons
    H=Neurons(n);
    parfor i=1:10% For each random state (Parallel multi-core processing)
        fprintf(['Validation for Model with: ',num2str(H),' neurons and randomization ',num2str(i),'\n']);
        % Validation for Model with: H neurons and randomization i
        [val_xentrval,val_xentrtrn,val_epochs]=ValidationLoops(S{i},folds,x,targets,H,ITRN,IVAL);
        perf_xentrval(n,i)=mean(val_xentrval);
        perf_xentrtrn(n,i)=mean(val_xentrtrn);
        perf_best_epval(n,i)=mean(val_epochs);
    end
end

% The following step implement the final architecture and test it on the test set
[~,optind]=min(mean(perf_xentrval,2)); % Found the optimal number of hidden units as the topology that minimise the ...
OptimalN(t)=Neurons(optind); % ... validation crossentropy across all the randomisation of the initial weights
net=patternnet(OptimalN(t),'trainscg'); % Definition of the net
net.performFcn = 'crossentropy'; % Cost function
net.divideFcn='divideind'; % Division by indexes
ITRNcom=1:length(DIVISOR); % All the dataset is in the training
ITRNcom(ITST)=[]; % Except for the hold out portion
net.divideParam.trainInd=ITRNcom;
```

```
    net.divideParam.valInd=ITST;
    net.divideParam.testInd=[];
    net=configure(net,x,targets); % Net configuration
    [net,tr,y,e]=train(net,x,targets); % Training of the net

    best_eptst(t)=tr.best_epoch; % Saving of the performance metrics
    xentrtst(t)=crossentropy(net,targets(:,ITST),y(:,ITST));
    xentrtstT(t)=crossentropy(net,targets(:,ITRNcom),y(:,ITRNcom));
    TSToutputs{t}=y(:,ITST);
    TSTtargets{t}=targets(:,ITST);
    % Accuracy
    [nanO,out]=max(y(:,ITST));
    [nanT,tar]=max(targets(:,ITST));
    cmat=confusionmat(out(~isnan(nanO)),tar(~isnan(nanT)));
    accuracy(t)=trace(cmat)/sum(cmat(:));
    for c=1:size(cmat,1)
        TP = cmat(c,c);
        FP = sum(cmat(c,:))-TP;
        FN = sum(cmat(:,c))-TP;
        recall(t)=recall(t)+(TP/(TP+FN));
        precision(t)=precision(t)+(TP/(TP+FP));
    end
    recall(t)=recall(t)/size(cmat,1);
    precision(t)=precision(t)/size(cmat,1);
    f1(t)=2*(precision(t)*recall(t))/(precision(t)+recall(t));
    xentrval{t}=perf_xentrval;
    xentrtrn{t}=perf_xentrtrn;
    best_epval{t}=perf_best_epval;
end
delete(gcp('nocreate')) % Close the Parallel Multicore processing
```

S.6 ValidationLoop.m function

The following code implements the inner loop of the SNCV routine given a determinate neural network model to validate (identified by the number of hidden units and the initial random weights). It takes as inputs the random state (S) to initialise the initial weights of the network, the number of folds in the outside loop (folds), the validation training data (the Design set, x) and the respective targets (targets), and the data structure containing the indexes of training and validation for each folds-1 validation loops (ITRN/IVAL). That is, the ith elements of ITRN and IVAL represent respectively the ith indexes of training and validation of the ith validation loop. It gives as outputs the values of crossentropy for the validation (val_xentrval) and training (val_xentrtrn) sets, and the number of training epochs (val_epochs) for each validation loop.

```
function [val_xentrval,val_xentrtrn,val_epochs]=ValidationLoops(S,folds,x,targets,H,ITRN,IVAL)

% INPUT       S: Random state
%             folds: Number of folds
%             x: Input data
%             targets: Targets
%             ITRN/IVAL: Indexes of training and validation, cell arrays with fold-1 elements containing the indexes of the
%             different validation cycles
% OUTPUT      val_xentrval: Validation crossentropy
%             val_xentrtrn: Training crossentropy
%             val_epochs: Total epochs of training

val_xentrval = zeros(1,folds-1);
val_xentrtrn = zeros(1,folds-1);
val_epochs = zeros(1,folds-1);
for v=1:folds-1 % For each validation fold
    net=patternnet(H,'trainscg'); % Definition of the net with learning model: Scaled Conjugant Gradient
    net.performFcn = 'crossentropy'; % Cost Function
    net.divideFcn='divideind'; % Define Sets by indexes
    net.divideParam.trainInd=ITRN{v}; % Training set
    net.divideParam.valInd=IVAL{v}; % Validation set
    net.divideParam.testInd=[];
    rng(S); % Reset the random state
    net=configure(net,x,targets); %Configuration of the initial parameter of the net
    [net,tr,y,e]=train(net,x,targets); % Training
    val_xentrval(v) = crossentropy(net,targets(:,IVAL{v}),...%------- Crossentropia
        y(:,IVAL{v}));
    val_xentrtrn(v) = crossentropy(net,targets(:,ITRN{v}),...
        y(:,ITRN{v}));
    val_epochs(v) = tr.best_epoch;
end
```

S.7 Redistribute.m function

```
% The function redistribute the remaining data observations to the less populated folds of the dataset

function [folds]=Redistribute(DIVISION,folds,remains)
% INPUT       DIVISION: cell array D x 1 where D is the entire dimensionality of the dataset
%             folds: cell array F x 1 where F is the number of folds containing the name of the included subjects
%             remains: cell array R x 1 where R is the number of subjects that remained undistributed from the main categorisation
% OUTPUT      folds

% Count the observations for the already defined folds sorting them in ascending order
occurencesG=sortrows([cell2mat(cellfun(@(x)sum(ismember(DIVISION,x)),folds,'un',0)),(1:1:length(folds))'],1,'ascend');
% Count the observations for the remaining subjects
occurencesR=sortrows([cell2mat(cellfun(@(x)sum(ismember(DIVISION,x)),remains,'un',0)),(1:1:length(R))'],1,'descend');
% the less populated folds are filled with the subjects with more observations
for k=1:length(remains)
    folds{occurencesG(k,2)}=[folds{occurencesG(k,2)};R{occurencesR(k,2)}];
end
```

S.8 ImplementTCN.m function

The following script tune evaluates and implements a Temporal Convolutional Neural Network according to the model proposed in [207].

```
% The script tune evaluates and implements a Temporal Convolutional Neural Network according to the model proposed in [207]. It
calls the function TCN which builds the network architecture.

load("DataSTS_NOTimed.mat",'DataSEQ');
% DataSEQ is a version of the dataset where all the subjects are divided with their respective STS sequence.
% It is a cell array S x 1 where S is the number of subjects. Each element is a T x 2 cell array where T is the number of trials for
% each subject: The first column contains the features from the accelerometers, a F x L matrix where F is the number of features and
% L the length of the sequence. The second column represents the label of the sequence, a matrix 1 x L.

loop=true; % This part of the code limits the choice of the phases to analyse. Just Sit-To-Stand, not Stand-to-Sit
while loop
    p=input('Phase of movement considered 1-REST, 2-TRUNKLEANING, 3-RAISING, 4-STANDING, 5-SITTING, 6-TRUNKRAISING, 7-REST:');
    if (0<p) && (p<8)
        loop=false;
    end
end

loop=true; % Choice of the number of sensors to consider
while loop
 s=input('Number of sensors considered :');
 if (0<s) && (s<7)
     loop=false;
 end
end
while loop==0 % Choice of the validation process
    flag=input('What validation model you would like to implement?\n   1 - 5 folds\n   2 - 10 folds\n   3 - LOSOCV\n\n');
    switch flag
        case 1      % 6*5 Nested CV
            f = 6; loop = 1;
        case 2      % 11*10 Nested CV
            f = 11; loop = 1;
        case 3      % LOSOCV
            f = length(SBJ); loop = 1;
        otherwise   % Il
            loop = 0;
    end
end

X=cell(f,Nrep); % Declaration of inputs
Y=cell(f,Nrep); % Declaration of outputs
LearnRates=sort(10.^(3.*rand(10,1)-3)); % hyperparameters to tune: learn rates
SizeF=[2 3 4 5]; % hyperparameters to tune: size of the filters
Nrep = 5; % number of repetitions
ValidationAccV=zeros(numel(LearnRates),numel(SizeF),f,Nrep); % Accuracy in validation (Tuning of the model)
ValidationAccT=zeros(numel(LearnRates),numel(SizeF),f,Nrep); % Accuracy in training (Tuning of the model)
ValidationTime=zeros(numel(LearnRates),numel(SizeF),f,Nrep); % Prediction time (Tuning of the model)
TestAccV=zeros(f,Nrep); % Accuracy in test (Estimate of the model)
TestAccT=zeros(f,Nrep); % Accuracy in training (Estimate of the model)
TestTime=zeros(f,Nrep); % Prediction time (Estimate of the model)
TuneAccV=zeros(numel(LearnRates),f,Nrep); % Accuracy in validation (Final tuning of the model)
TuneAccT=zeros(numel(LearnRates),f,Nrep); % Accuracy in training (Final tuning of the model)
TuneTime=zeros(numel(LearnRates),f,Nrep); % Prediction timme (Final tuning of the model)
Hypar_choice=cell(f,Nrep); % Hyper-parameter choice (Estimate of the model)
YP=cell(f,Nrep); % Prediction (Estimate of the model)
YT=cell(f,Nrep); % Targets (Estimate of the model)

for r=1:5 % Repeated Stratified Cross-Validation

    idx=randperm(numel(DataSEQ));    % Randomisation of the subjects
    DataSEQR=DataSEQ(idx);

    step=floor(numel(DataSEQR)/f); % Preparation of the folds (as seen NeuralSNCV.m)
    division=1:step:(step*f);
    R=rem(numel(DataSEQR),f);
    folds=cell(f,1);
```

```
for i=1:f
    folds{i}=DataSEQR(division(i):division(i)+step-1);
end

if R~=0 % Distribution of remaining subjects
    folds=Redistribute(folds,DataSEQR(end-(R-1):end));
end
folds=cellfun(@(x)vertcat(x{:}),folds,'UniformOutput',false);

for i=1:numel(folds) % Preparation of dataset according to the user choices
    X{i,r}=cellfun(@(x,y)x(1:7:s*6*7,y<=p),folds{i}(:,1),folds{i}(:,2),'UniformOutput',0);
    Y{i,r}=cellfun(@(y)categorical(y(y<=p)),folds{i}(:,2),'UniformOutput',0);
end

for t=1:numel(folds) % Outer loop Estimate of the model
    idev=1:numel(folds); % Development set (Training+Validation)
    idev(t)=[]; % Except the present fold as Test set
    Xtest=X(t,r);     % Test Data
    Ytest=Y(t,r);     % Test Response
    Xdev=X(idev,r);   % Development Data
    Ydev=Y(idev,r);   % Development Response

    for ln=1:numel(LearnRates) % For each set of parameters
        for sz=1:numel(SizeF)
            % Validation and parameter tuning on the remaining folds: Inner loop
            [ValidationAccV(ln,sz,t,r),ValidationAccT(ln,sz,t,r),ValidationTime(ln,sz,t,r)]=TCN(...
               Xdev,...                 % X
               Ydev,...                 % Y
               SizeF(sz),...            % filterSize
               30,...                   % maxEpochs
               20,...                      % miniBatchSize,
               LearnRates(ln),...          % initialLearnRate
               0.1,...                     % learnRateDropFactor
               12,...                      % learnRateDropPeriod
               1,...                       % gradientThreshold
               1,...                       % validationFrequency
               0.0001,...                  % l2Regularization
               "auto",...                  % executionEnvironment
               "none",...              % plots
               "validation"...             % mode
            );
        end
    end

    [OptimalLR,OptimalSZ]=find(ValidationAccV(:,:,t,r)==max(ValidationAccV(:,:,t,r),[],'all')); % Find the Optimal parameters
    Hypar_choice{t,r}=[LearnRates(OptimalLR),SizeF(OptimalSZ)];
   % Estimate of the model on the test set
    [TestAccV(t,r),TestAccT(t,r),TestTime(t,r),YP{t,r},YT{t,r}]=TCN(...
           [Xdev;Xtest],...            % X
           [Ydev;Ytest],...            % Y
           SizeF(OptimalSZ),...        % filterSize
           30,...                      % maxEpochs
           20,...                      % miniBatchSize,
           LearnRates(OptimalLR),...       % initialLearnRate
           0.1,...                     % learnRateDropFactor
           12,...                      % learnRateDropPeriod
           1,...                       % gradientThreshold
           1,...                       % validationFrequency
           0.0001,...                  % l2Regularization
           "auto",...                  % executionEnvironment
```

```
                "training-progress",...  % plots
                "test"...          % mode
            );
        figure('Name',['Test ',num2str(t)]);
        plotconfusion(horzcat(YT{t,r}{:}),horzcat(YP{t,r}{:}));
    end

    % The below section reiterates the tuning process (validation) using all the folds in the datasets
    for ln=1:numel(LearnRates)
        for sz=1:numel(SizeF)
            [TuneAccV(ln,sz,r),TuneAccT(ln,sz,r),TuneTime(ln,sz,r)]=TCN(...
                X(:,r),...                 % X
                Y(:,r),...                 % Y
                SizeF(sz),...             % filterSize
                30,...                    % maxEpochs
                20,...                    % miniBatchSize,
                LearnRates(ln),...        % initialLearnRate
                0.1,...                   % learnRateDropFactor
                12,...                    % learnRateDropPeriod
                1,...                     % gradientThreshold
                1,...                     % validationFrequency
                0.0001,...                % l2Regularization
                "auto",...                % executionEnvironment
                "none",...  % plots
                "validation"...           % mode
            );
        end
    end
end
% From the final tuning process, averaging across the different repetitions to find the best hyperparameters
```

```
globalAcc=mean(TuneAccV,3);
[OptimalLR,OptimalSZ]=find(globalAcc==max(globalAcc,[],'all'));
% The randomisation of the dataset that yielded the best results is used as training data
[~,OptimalRep]=max(squeeze(mean(mean(TuneAccV,1),2)))
[~,~,~,~,~,parameters,hyperparameters,~,~]=TCN(...
    X(:,OptimalRep),...              % X
    Y(:,OptimalRep),...              % Y
    SizeF(OptimalSZ),...     % filterSize
    30,...                   % maxEpochs
    20,...                   % miniBatchSize,
    LearnRates(OptimalLR),...% initialLearnRate
    0.1,...                  % learnRateDropFactor
    12,...                   % learnRateDropPeriod
    1,...                    % gradientThreshold
    1,...                    % validationFrequency
    0.0001,...               % l2Regularization
    "auto",...               % executionEnvironment
    "training-progress",...  % plots
    "training"...            % mode
);

XF=cell(size(X)); % Declaration of the transformed dataset (and all its randomisations)
YF=cell(size(Y));
% In features mode the TCN function extrapolate the elaborated feature map from the initial dataset
for r=1:Nrep
    [~,~,~,~,~,~,~,XF(:,r),YF(:,r)]=TCN(...
               X(:,r),...                % X
               Y(:,r),...                % Y
               SizeF(OptimalSZ),...      % filterSize
               30,...                    % maxEpochs
               20,...                   % miniBatchSize,
               LearnRates(OptimalLR),...% initialLearnRate
               0.1,...                  % learnRateDropFactor
               12,...                   % learnRateDropPeriod
               1,...                    % gradientThreshold
               1,...                    % validationFrequency
               0.0001,...               % l2Regularization
               "auto",...               % executionEnvironment
               "training-progress",...  % plots
               "features",...           % mode
               parameters,...
               hyperparameters...
            );
end
```

S.9 TCN.m function

The following code implements the Temporal Convolutional Network (TCN) according to the parameters passed from the script ImplementTCN.m. It can take as inputs the training data (X), the relative target labels (Y), the size of the convolutional filters (filterSize), the maximum number of training epochs allowed (maxEpochs), the number of training samples for each mini-batch subset (miniBatchSize), the initial learning rate (initialLearnRate), the learning drop factor (learnRateDropFactor), the period that marks the dropping of the learning rate (learnRateDropPeriod), the threshold for gradient clipping (gradientThreshold), the period that marks the validation checks (validationFrequency), the L2 regularisation constant (l2Regularization), the execution environment (CPU/GPU) (executionEnvironment), the plotting option (plot – chose whenever to plot the training progress), the set of parameters of the convolutional network (learnables) the set of hyperparameters of the convolutional network (hyperparameters) and the functioning mode (mode). It can give as outputs the overall accuracy on the validation (or either test) set (validationAccuracy), the overall accuracy on the training set (trainingAccuracy), the average prediction time on the validation set (TimePred), the predictions of the network with the relative labelled targets (pred/targ), the set of parameters of the convolutional network (learnables) the set of hyperparameters of the convolutional network (hyperparameters), the elaborated parameters of the network with the relative labelled targets (transformedInput, transformedOutput). The mode parameter controls the four different work modalities of the function:

- VALIDATION: In validation mode, the TCN is trained and validated across the different folds of the Design set, the function takes all inputs except for the "learnables" and the "hyperparameters" parameters and it gives as outputs the "validationAccuracy", the "trainingAccuracy", and the "TimePred" parameters to evaluate the validation performance;
- TEST: In test mode, the TCN is trained following the optimal model characteristics (from the validation mode) over the entire Design test and validated over the Test set, the function takes all inputs except for the "learnables" and the "hyperparameters" parameters and it gives as outputs the "validationAccuracy", the "trainingAccuracy", the "TimePred", the "pred", and the "targ" parameters to evaluate the test performance;
- TRAINING: In training mode, the TCN is trained following the optimal model characteristics (from the test mode) over the entire Design test and validated over the Test set, the function takes all inputs except for the "learnables" and the "hyperparameters" parameters and it gives as outputs the "learnables", and the "hyperparameters" parameters to reproduce the optimal neural network;
- FEATURES: In features mode, the optimal TCN obtained from the training mode is used to extract elaborated features from the entire dataset using the convolutional layers. The function takes all the input parameters and returns the elaborated dataset ("XF" and "YF" parameters.

```
% This function implements a Temporal Convolutional Network in training validation test and feature mode. It works in different
% modalities, depending on the call from implementTCN.m. The supporting functions called by this algorithm are reported and briefly
% described after the main routine.

function [validationAccuracy,trainingAccuracy,TimePred,pred,targ,learnables,hyperparameters,transformedInput,transformedOutput] =
TCN(X,Y,filterSize,maxEpochs,miniBatchSize,initialLearnRate,learnRateDropFactor,learnRateDropPeriod,gradientThreshold,validationFreq
uency,l2Regularization,executionEnvironment,plots,mode,learnables,hyperparameters)

% INPUT      X: Accelerometric data [F x 1] cell array containing [S x 1] cell arrays containing [Feat x T] elements.
%            Where F is the number of features, S is the number of sequences, Feat is the number of features and T is the number of
%            time steps
%            Y: Response data [F x 1] cell array containing [S x 1] cell arrays containing [1 x T] elements.
%            maxEpochs: Number of training epochs.
%            miniBatchSize: Number of sequences inside a batch of training.
%            initialLearnRate: Initial learning rate.
%            learnRateDropFactor,learnRateDropPeriod: These two parameters describe the learning rate decay process.
%            gradientThreshold: Threshold for gradient clipping.
%            validationFrequency: Number of epochs after which the trained network is tested on validation data.
%            l2Regularization: l2 regularization rate.
%            executionEnvironment: 'gpu' or 'cpu'.
%            plots: 'training-progress' or 'none'.
%            mode: 'validation','test','training'
% OUTPUT     validationAccuracy: The accuracy of the network on the set of data held out from the training.
%            trainingAccuracy: The accuracy of the network on the set of training.
%            TimePred: Prediction time of the network on the set of data held out from the training
%            pred: Predicted outputs.
%            targ: Target outputs
%            learnables: Parameters of the model
%            hyperparameters: Hyperparameters of the model
%            transformedInput:
%            transformedOutput:

% VALIDATION MODE: Iteration of the training-validation process across the folds of X,Y
% INPUTS: All
% OUTPUTS: validationAccuracy, trainingAccuracy, TimePred
```

```
if mode == "validation"
    learnables=[]; % Declaration of the learnables parameters structured as a table
    hyperparameters=[]; % Hyper-parameters
    VvalidationAccuracy = ones(numel(X),1); % Validation accuracy on the dev set
    VtrainingAccuracy = ones(numel(X),1); % Training accuracy on the dev set

    for v = 1:numel(X) % Validation loop
        idt = 1:numel(X); % training folds
        idt(v) = []; % held out the current fold as validatiomn
        XT=vertcat(X{idt});YT = vertcat(Y{idt}); % grouping all the sequences in the training and validation folds
        XV=vertcat(X{v});YV = vertcat(Y{v});
        % Dimensionality of the dataset
        numSequences = numel(XT); % Number of sequences
        numInputChannels = size(XT{1},1); % Number of features of the signal
        classes = categories(YT{1});
        numClasses = numel(classes); % Number of the output classes

        hyperparameters = struct; % hyperparameter definition
        numBlocks = 4;              % residual blocks (default: 4)
        numFilters = 175;           % number of filters (default: 175)
        dropoutFactor = 0.05;       % dropout (default: 0.05)
        hyperparameters.NumBlocks = numBlocks;
        hyperparameters.DropoutFactor = dropoutFactor;
        hyperparameters.sizeFilters = filterSize;

        learnables = table([],[],'VariableNames',{'Parameter','Value'});% learnable parameter definition for each block
        numChannels = numInputChannels;
        for k = 1:numBlocks
            blockName = "Block"+k
            par = {...
        blockName+".Conv1.Weights";... % Parameters name for convolution 1
        blockName+".Conv1.Bias";...
        blockName+".Conv2.Weights";... % Parameters name for convolution 2
        blockName+".Conv2.Bias"...
        };
    var = {...
        dlarray(HeinitializeGaussian([filterSize, numChannels, numFilters]));... % Value for convolution 1
        dlarray(zeros(numFilters, 1, 'single'));...
        dlarray(HeinitializeGaussian([filterSize, numFilters, numFilters]));... % Value for convolution 2
        dlarray(zeros(numFilters, 1, 'single'))...
        };
    learnables = [learnables;par,var]; compose the learnables table
    if numChannels ~= numFilters % If inputs and outputs of the residual block are not the same
        par = {...
            blockName+".Conv3.Weights";... % Parameters name for convolution 3
            blockName+".Conv3.Bias"...
            };
        var = {...
            dlarray(HeinitializeGaussian([1, numChannels, numFilters]));... % Value for convolution 3
            dlarray(zeros(numFilters, 1, 'single'))...
            };
        learnables = [learnables;par,var]; % compose the learnables table

    end
    numChannels = numFilters; % For next block, update number of channels
end


par = {... % fully connect parameters names
    "FC.Weights";...
    "FC.Bias"...
    };
var = {...
    dlarray(HeinitializeGaussian([numClasses,numChannels]));... % fully connect parameters values
    dlarray(zeros(numClasses,1,'single'))...
    };
learnables = [learnables;par,var]; compose the learnables table
learnRate = initialLearnRate;
trailingAvg = []; % moving average of the parameters
trailingAvgSq = []; % element-wise squares of the gradients used by the Adam optimizer.

if plots == "training-progress" % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    lineAccuracyValid =  animatedline(gca,'Color',[0.85 0.325 0.098]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on
    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    lineLossValid = animatedline('Color',[0.85 0.325 0.098]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on
    if exist('conf','var') == 0;conf=figure;end % If the confusion plot does not exist create it
end
iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize); % Number of iterations per epochs
```

```
for epoch = 1:maxEpochs % for each epoch
    idx = randperm(numSequences);
    XT = XT(idx);
    YT = YT(idx);

    for i = 1:numIterationsPerEpoch % for each iteration

        iteration = iteration + 1;
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;

        % This section calls at the function transformSequences.m for pre-processing
        [DataTrain,ResponseTrain,numTimeSteps] = transformSequences(XT(idx),YT(idx)); % Training
        [DataValid,ResponseValid,numTimeStepsV] = transformSequences(XV,YV); % Validation
        dlX = dlarray(DataTrain); % Training data
        dlXV = dlarray(DataValid); % Validation data

        % Gradient calculation
        [gradients, loss, accuracy] = dlfeval(...
        @modelGradients, ...
        dlX, ...
        ResponseTrain, ...
        learnables.Value, ...
        learnables.Parameter, ...
        hyperparameters, ...
        numTimeSteps);
        % Regularization
        idx = contains(learnables.Parameter,"Weights"); % Consider the weights parameters
        % Update the gradients
        gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), learnables.Value(idx,:));
```

```
% Gradients clipping
gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
% Adam optimizer
[learnables.Value,trailingAvg,trailingAvgSq] = adamupdate(learnables.Value,gradients, ...
    trailingAvg, trailingAvgSq, iteration, learnRate);
% Validation of the model (at first iteration and at a defined frequency
if iteration == 1 || mod(iteration,validationFrequency) == 0
    [lossV, accuracyV,targ,pred,TimePred] = modelValidation(...
    dlXV, ...
    ResponseValid, ...
    learnables.Value, ...
    learnables.Parameter, ...
    hyperparameters,numTimeStepsV);

    if plots == "training-progress" % Validation progress (if plot is enabled)
        figure(lossacc);
        addpoints(lineLossValid,iteration, lossV);
        addpoints(lineAccuracyValid,iteration, accuracyV);
        figure(conf);
        plotconfusion(targ,pred)
    end
end
if plots == "training-progress" % Training progress (if plot is enabled)
    loss = mean(loss ./ numTimeSteps); % Normalize the loss over the sequence lengths
    loss = double(gather(extractdata(loss)));
    loss = mean(loss);
    figure(lossacc);
    addpoints(lineLossTrain,iteration, mean(loss));
    addpoints(lineAccuracyTrain,iteration, accuracy);
    title("Epoch: " + epoch)
```

```
                    drawnow
                end
            end
            if mod(epoch,learnRateDropPeriod) == 0 % Drop the learning rate after a defined period
                learnRate = learnRate*learnRateDropFactor;
            end
            fprintf(['EPOCH',num2str(epoch),' Validation = %f | Training = %f\n'],accuracyV,accuracy);
        end
        VvalidationAccuracy(v)=accuracyV;
        VtrainingAccuracy(v)=accuracy;
        fprintf('Validation accuracy on fold = %f \n',VvalidationAccuracy(v));
        fprintf('Training accuracy on fold = %f \n',VtrainingAccuracy(v));
    end
    validationAccuracy=mean(VvalidationAccuracy);
    trainingAccuracy=mean(VtrainingAccuracy);
    fprintf('Validation accuracy = %f \n',mean(validationAccuracy));
    fprintf('Training accuracy = %f \n',mean(trainingAccuracy));

% TEST MODE: training the net on the entire train+validation sets and evaluating on the test set
% INPUTS: All
% OUTPUTS: validationAccuracy, trainingAccuracy, TimePred


  elseif mode == "test" % The first part follow the same rules of the VALIDATION MODE, but it is not included in a for loop
    learnables=[]; % Declaration of the learnables parameters structured as a table
    hyperparameters=[]; % Hyper-parameters
    XT=vertcat(X{1:end-1});YT=vertcat(Y{1:end-1}); % grouping all the sequences in the training and validation folds
    numSequences = numel(XT);
    numInputChannels=size(XT{1},1); % Number of features of the signal
    classes = categories(YT{1}); % Number of the output classes
```

```
numClasses = numel(classes);
% hyperparameter definition
hyperparameters = struct;
numBlocks = 4;              % residual blocks (default: 4)
numFilters = 175;           % number of filters (default: 175)
dropoutFactor = 0.05;       % dropout (default: 0.05)
hyperparameters.NumBlocks = numBlocks;
hyperparameters.DropoutFactor = dropoutFactor;
hyperparameters.sizeFilters = filterSize;

learnables = table([],[],'VariableNames',{'Parameter','Value'}); % learnable parameter definition for each block
numChannels = numInputChannels;

for k = 1:numBlocks
    blockName = "Block"+k;

    par = {...
        blockName+".Conv1.Weights";... % Parameters name for convolution 1
        blockName+".Conv1.Bias";...
        % convolution2
        blockName+".Conv2.Weights";... % Parameters name for convolution 2
        blockName+".Conv2.Bias"...
        };
    var = {...
        dlarray(HeinitializeGaussian([filterSize, numChannels, numFilters]));... % Value for convolution 1
        dlarray(zeros(numFilters, 1, 'single'));...
        dlarray(HeinitializeGaussian([filterSize, numFilters, numFilters]));... % Value for convolution 2
        dlarray(zeros(numFilters, 1, 'single'))...
        };
    learnables = [learnables;par,var]; % compose the learnables table
    if numChannels ~= numFilters
        par = {...
            blockName+".Conv3.Weights";... % Parameters name for convolution 3
            blockName+".Conv3.Bias"...
            };
        var = {...
            dlarray(HeinitializeGaussian([1, numChannels, numFilters]));... % Value for convolution 3
            dlarray(zeros(numFilters, 1, 'single'))...
            };
        learnables = [learnables;par,var]; % compose the learnables table
    end

    numChannels = numFilters; % For next block, update number of channels

end

par = {... % fully connect parameters names
    "FC.Weights";...
    "FC.Bias"...
    };
var = {...
    dlarray(HeinitializeGaussian([numClasses,numChannels]));... % fully connect parameters values
    dlarray(zeros(numClasses,1,'single'))...
    };
learnables = [learnables;par,var]; % compose the learnables table
learnRate = initialLearnRate;
trailingAvg = []; % moving average of the parameters
trailingAvgSq = []; % element-wise squares of the gradients used by the Adam optimizer.
```

```
if plots == "training-progress" % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end  % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on
    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on
end
iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize); % Number of iterations per epochs

for epoch = 1:maxEpochs % for each epoch
    idx = randperm(numSequences);
    XT = XT(idx);
    YT = YT(idx);

    for i = 1:numIterationsPerEpoch % for each iteration

        iteration = iteration + 1;
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;

        % This section calls at the function transformSequences.m for pre-processing
        [DataTrain,ResponseTrain,numTimeSteps] = transformSequences(XT(idx),YT(idx)); % Training
        dlX = dlarray(DataTrain); % Training data
```

```
% gradient calculation
[gradients, loss, trainingAccuracy] = dlfeval(...
 @modelGradients, ...
 dlX, ...
 ResponseTrain, ...
 learnables.Value, ...
 learnables.Parameter, ...
 hyperparameters, ...
 numTimeSteps);

% Regularization
idx = contains(learnables.Parameter,"Weights");
 % Update the gradients
gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), learnables.Value(idx,:));
% Gradient clipping
gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
% Adam optimizer
[learnables.Value,trailingAvg,trailingAvgSq] = adamupdate(learnables.Value,gradients, ...
    trailingAvg, trailingAvgSq, iteration, learnRate);
if plots == "training-progress" % Training progress (if plot is enabled)
    loss = mean(loss ./ numTimeSteps); % Normalize the loss over the sequence lengths
    loss = double(gather(extractdata(loss)));
    loss = mean(loss);
    figure(lossacc);
    addpoints(lineLossTrain,iteration, mean(loss));
    addpoints(lineAccuracyTrain,iteration, trainingAccuracy);
    title("Epoch: " + epoch)
    drawnow
end
```

```
    end
    if mod(epoch,learnRateDropPeriod) == 0 % Drop the learning rate after a defined period
        learnRate = learnRate*learnRateDropFactor;
    end
end
fprintf('Training accuracy = %f \n',trainingAccuracy);

Xt=vertcat(X{end});Yt=vertcat(Y{end}); % Test data
numObservationsTest = numel(Xt); % Number of test observations
doTraining = false; % test phase

% Test data and response
dlXTest=cellfun(@(x)dlarray(reshape(x,[size(x,1),1,size(x,2)])),Xt,'UniformOutput',false);
Ytest=cellfun(@(x)reshape(single(full(ind2vec(double(x),numClasses))),[numClasses,1,size(x,2)]),Yt,'UniformOutput',false);

accuracy = zeros(1,numObservationsTest); % Accuracy for every test sequence
Times = zeros(1,numObservationsTest); % Prediction time for every test sequence
pred = cell(1,numObservationsTest); % Predictions for every test sequence
targ = cell(1,numObservationsTest); % Targets for every test sequence
for obs=1:numObservationsTest % For each sequence in the test set
    start=tic;
    dlYPred = model(dlXTest{obs},learnables.Value,learnables.Parameter,hyperparameters,doTraining); % Prediction
    Times(obs) = seconds(duration(0,0,toc(start),'Format',"mm:ss.SSS")); % Time
    YPred = gather(extractdata(dlYPred));
    pred{obs} = squeeze(YPred); % Predictions
    targ{obs} = squeeze(Ytest{obs}); % Targets
    [~,idxPred] = max(pred{obs},[],1);
    [~,idxTest] = max(targ{obs},[],1);
    accuracy(obs) = mean(idxPred == idxTest); % Accuracy for the sequence
end
```

```
        validationAccuracy=mean(accuracy);
        TimePred=mean(Times);
% TRAINING MODE: training the net on the entire dataset
% INPUTS: All
% OUTPUTS: learnables, hyperparameters
    elseif mode == "training"
        learnables=[]; % Declaration of the learnables parameters structured as a table
        hyperparameters=[]; % Hyper-parameters
        XT=vertcat(X{1:end});YT=vertcat(Y{1:end}); % grouping all the sequences
        numSequences = numel(XT);
        numInputChannels=size(XT{1},1); % Number of features of the signal
        classes = categories(YT{1}); % Number of the output classes
        numClasses = numel(classes);
        % hyperparameter definition
        hyperparameters = struct;
        numBlocks = 4;              % residual blocks (default: 4)
        numFilters = 175;           % number of filters (default: 175)
        dropoutFactor = 0.05;       % dropout (default: 0.05)
        hyperparameters.NumBlocks = numBlocks;
        hyperparameters.DropoutFactor = dropoutFactor;
        hyperparameters.sizeFilters = filterSize;

        learnables = table([],[],'VariableNames',{'Parameter','Value'}); % learnable parameter definition for each block
        numChannels = numInputChannels;

        for k = 1:numBlocks
            blockName = "Block"+k;

            par = {...
                blockName+".Conv1.Weights";... % Parameters name for convolution 1
                blockName+".Conv1.Bias";...
                % convolution2
                blockName+".Conv2.Weights";... % Parameters name for convolution 2
                blockName+".Conv2.Bias"...
                };
            var = {...
                dlarray(HeinitializeGaussian([filterSize, numChannels, numFilters]));... % Value for convolution 1
                dlarray(zeros(numFilters, 1, 'single'));...
                dlarray(HeinitializeGaussian([filterSize, numFilters, numFilters]));... % Value for convolution 2
                dlarray(zeros(numFilters, 1, 'single'))...
                };
            learnables = [learnables;par,var]; % compose the learnables table

            if numChannels ~= numFilters
                par = {...
                    blockName+".Conv3.Weights";... % Parameters name for convolution 3
                    blockName+".Conv3.Bias"...
                    };
                var = {...
                    dlarray(HeinitializeGaussian([1, numChannels, numFilters]));... % Value for convolution 3
                    dlarray(zeros(numFilters, 1, 'single'))...
                    };
                learnables = [learnables;par,var]; % compose the learnables table
            end

            numChannels = numFilters; % For next block, update number of channels

        end

        par = {... % fully connect parameters names
```

```
    "FC.Weights";...
    "FC.Bias"...
    };
var = {...
    dlarray(HeinitializeGaussian([numClasses,numChannels]));... % fully connect parameters values
    dlarray(zeros(numClasses,1,'single'))...
    };
learnables = [learnables;par,var]; % compose the learnables table
learnRate = initialLearnRate;
trailingAvg = [];               % moving average of the parameters
trailingAvgSq = [];             % element-wise squares of the gradients used by the Adam optimizer.

if plots == "training-progress" % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end  % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on
    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on
end
iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize); % Number of iterations per epochs

for epoch = 1:maxEpochs % for each epoch
```

```
idx = randperm(numSequences);
XT = XT(idx);
YT = YT(idx);

for i = 1:numIterationsPerEpoch % for each iteration

    iteration = iteration + 1;
    idx = (i-1)*miniBatchSize+1:i*miniBatchSize;

    % This section calls at the function transformSequences.m for pre-processing
    [DataTrain,ResponseTrain,numTimeSteps] = transformSequences(XT(idx),YT(idx)); % Training
    dlX = dlarray(DataTrain); % Training data

    % gradient calculation
    [gradients, loss, trainingAccuracy] = dlfeval(...
     @modelGradients, ...
     dlX, ...
     ResponseTrain, ...
     learnables.Value, ...
     learnables.Parameter, ...
     hyperparameters, ...
     numTimeSteps);

    % Regularization
    idx = contains(learnables.Parameter,"Weights");
     % Update the gradients
    gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), learnables.Value(idx,:));
    % Gradient clipping
    gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
    % Adam optimizer
```

```
                [learnables.Value,trailingAvg,trailingAvgSq] = adamupdate(learnables.Value,gradients, ...
                    trailingAvg, trailingAvgSq, iteration, learnRate);
                if plots == "training-progress" % Training progress (if plot is enabled)
                    loss = mean(loss ./ numTimeSteps); % Normalize the loss over the sequence lengths
                    loss = double(gather(extractdata(loss)));
                    loss = mean(loss);
                    figure(lossacc);
                    addpoints(lineLossTrain,iteration, mean(loss));
                    addpoints(lineAccuracyTrain,iteration, trainingAccuracy);
                    title("Epoch: " + epoch)
                    drawnow
                end
            end
            if mod(epoch,learnRateDropPeriod) == 0 % Drop the learning rate after a defined period
                learnRate = learnRate*learnRateDropFactor;
            end
        end
        fprintf('Training accuracy = %f \n',trainingAccuracy);
% FEATURE MODE: Giving a trained model extract the complex features from the entire dataset
% INPUTS: All
% OUTPUTS: transformedInput; transformedOutput;
    elseif mode == "features"
        Xtransf=vertcat(X{1:end});Ytransf=vertcat(Y{1:end});
        numObservations = numel(Xtransf);
        transformedInput=cell(numObservations,1); % transformed dataset
        ends=cumsum(cellfun(@(x)length(x),X)); % ends of folds subdivision
        starts=[1;ends(1:end-1)+1]; % starts  of folds subdivision
        for obs=1:numObservations % For each sequence
            [XF,~]=transformSequences(Xtransf(obs),Ytransf(obs));
            dlX = dlarray(XF);
            doTraining = false;
            getFeatures = true;
            transformedInput{obs} = squeeze(extractdata(model(... extract the elaborated features from the convolutional layers
            dlX, ...
            learnables.Value, ...
            learnables.Parameter, ...
            hyperparameters, ...
            doTraining, ...
            getFeatures)));
        end

        transformedInput=cellfun(@(x,y)transformedInput(x:y),num2cell(starts),num2cell(ends),'UniformOutput',false);
        transformedOutput=Y;
    end
    switch mode % Outputs according to the function mode
    case "validation"
        pred=[];targ=[];learnables=[];hyperparameters=[];transformedInput=[];transformedOutput=[];
    case "test"
        learnables=[];hyperparameters=[];transformedInput=[];transformedOutput=[];
    case "training"
        validationAccuracy=[];TimePred=[];pred=[];targ=[];transformedInput=[];transformedOutput=[];
    case "features"
        validationAccuracy=[];trainingAccuracy=[];TimePred=[];pred=[];targ=[];learnables=[];hyperparameters=[];
    end
end
```

S.9.1 Supporting functions

initializeGaussian.m. function

```
% The initializeGaussian function samples weights from a Gaussian distribution with mean 0 and standard deviation 0.01.
```

```
function parameter = initializeGaussian(sz)
    parameter = randn(sz,'single') .* 0.01;
end
```

HeinitializeGaussian.m function

```
% The HeinitializeGaussian function samples weights using the He initialization.
```

```
function parameter = HeinitializeGaussian(sz,scale)
    if nargin < 2
    scale = 0.1;
    end
    numIn = prod(sz);
    varWeights = 2 / ((1 + scale^2) * numIn);
    parameter = randn(sz) * sqrt(varWeights);
end
```

transformSequences.m function:

```
% The transformSequence function takes a cell array of N sequences and returns a C-by-N-by-S numeric array of left-padded 1-D
% sequences and the number of time steps in each sequence, where C corresponds to the number of features of the sequences and S
% corresponds to the number of time steps of the longest sequence.
```

```
function [XTransformed, YTransformed, numTimeSteps] = transformSequences(X,Y)
numTimeSteps = cellfun(@(sequence) size(sequence,2),X);
miniBatchSize = numel(X); % Size of the mini-batch
numFeatures = size(X{1},1); % Number of features
sequenceLength = max(cellfun(@(sequence) size(sequence,2),X)); % Maximum lenght in the mini-batch.
classes = categories(Y{1});     % classes in the sequences
```

```
numClasses = numel(classes);    % number of classes

sz = [numFeatures miniBatchSize sequenceLength]; % Definition of the Input and Output transformed sequences
XTransformed = zeros(sz,'single');
sz = [numClasses miniBatchSize sequenceLength];
YTransformed = zeros(sz,'single');

for i = 1:miniBatchSize For each sequence.
    predictors = X{i};
    responses = zeros(numClasses, numTimeSteps(i), 'single');
    for c = 1:numClasses
        responses(c,Y{i}==classes(c)) = 1;
    end
    XTransformed(:,i,:) = leftPad(predictors,sequenceLength); % Left padding.
    YTransformed(:,i,:) = leftPad(responses,sequenceLength);
end
end
```

leftPad.m function

```
% The leftPad function takes a sequence and left-pads it with zeros to have the specified sequence length.

function sequencePadded = leftPad(sequence,sequenceLength)
[numFeatures,numTimeSteps] = size(sequence);
paddingSize = sequenceLength - numTimeSteps;
padding = zeros(numFeatures,paddingSize);
sequencePadded = [padding sequence];
end
```

modelGradients.m function

```
% The modelGradients function takes a mini-batch of input data dlX, the corresponding target sequences T, the learnable parameters,
% and the hyperparameters, and returns the gradients of the loss with respect to the learnable parameters and the corresponding loss
% and applying, if setted the the L2 regularization using the RegularizationFunction function. To compute the gradients, evaluate
% the modelGradients function using the dlfeval function in the training loop.

function [gradients,loss,accuracy] = modelGradients(dlX,T,learnables,labels,hyperparameters,numTimeSteps)

dlY = model(dlX,learnables,labels,hyperparameters,true); Application of the model using the defined parameters and hyperparameters.
dlT = dlarray(T,'CBT');
numObservationsTest=size(T,2);  % Number of observations for the mini-batch
YP = gather(extractdata(dlY));  % Output of the model
acc = zeros(1,numObservationsTest);
for i = 1:numObservationsTest
    [~,idxP] = max(YP(:,i,:),[],1);
    [~,idxT] = max(T(:,i,:),[],1);
    acc(i) = mean(idxP == idxT);
end
accuracy=mean(acc); % Calculation of the accuracy on the training set.
loss = maskedCrossEntropyLoss(dlY, dlT, numTimeSteps); % Calculation of the loss on the training set.
gradients = dlgradient(mean(loss),learnables); Calculation of the gradients and Regularization.
end
```

maskedCrossEntropyLoss.m function

```
% The maskedCrossEntropyLoss function computes the cross-entropy loss for mini-batches of sequences, where the sequences are
% different lengths.

function loss = maskedCrossEntropyLoss(dlY, dlT, numTimeSteps)
```

```
numObservations = size(dlY,2);
loss = dlarray(zeros(1,numObservations,'like',dlY));
for i = 1:numObservations
    idx = (size(dlY,3)-numTimeSteps(i)+1):size(dlY,3);
    loss(i) = crossentropy(dlY(:,i,idx),dlT(:,i,idx),'DataFormat','CBT');
end
end
```

thresholdL2Norm.m function

% The thresholdL2Norm function scales the gradient g so that its L_2norm equals gradientThreshold when the L_2norm of the gradient is
% larger than gradientThreshold.

```
function g = thresholdL2Norm(g,gradientThreshold)
gradientNorm = sqrt(sum(g.^2,'all'));
if gradientNorm > gradientThreshold
    g = g * (gradientThreshold / gradientNorm);
end
end
```

modelValidation.m function

```
% The modelValidation function takes a mini-batch of validation input data dlXV, the corresponding target sequences TV, the
% parameters, and the hyperparameters, and returns the accuracy and the loss.

function [lossV,accuracyV,targ,pred,TimePred] = modelValidation(dlXV,TV,learnables,labels,hyperparameters,numTimeSteps)
numObservationsTest=size(TV,2); % number of observations in validation set

startV=tic;             % timer start
dlY = model(dlXV,learnables,labels,hyperparameters,false); % prediction
TimePred = duration(0,0,toc(startV),'Format',"mm:ss.SSS")/numObservationsTest; % Total time divided by the number of sequences
TimePred = seconds(TimePred);   % seconds
```

```
dlT = dlarray(TV,'CBT');
YP = gather(extractdata(dlY));
acc = zeros(1,numObservationsTest);
pred=zeros(size(YP,1),size(YP,2)*size(YP,3));
targ=zeros(size(TV,1),size(TV,2)*size(TV,3));

for i = 1:numObservationsTest % Calculation of the accuracy on the validation set
    pred(:,((i-1)*size(YP,3))+1:(i*size(YP,3)))=squeeze(YP(:,i,:));
    targ(:,((i-1)*size(TV,3))+1:(i*size(TV,3)))=squeeze(TV(:,i,:));
    [~,idxP] = max(YP(:,i,:),[],1);
    [~,idxT] = max(TV(:,i,:),[],1);
    acc(i) = mean(idxP == idxT);
end
accuracyV=mean(acc);
lossV = maskedCrossEntropyLoss(dlY, dlT, numTimeSteps); % Calculation of the loss on the validation set
lossV = mean(lossV ./ numTimeSteps);
lossV = double(gather(extractdata(lossV)));
lossV = mean(lossV);
end
```

model.m function

```
% The function model takes the input data dlX, the learnable model parameters, the model hyperparameters, and the flag doTraining
% which specifies whether the model should return outputs for training or prediction. The network outputs the predictions for the
% labels at each time step of the input sequence. The model consists of multiple residual blocks with exponentially increasing
% dilation factors. After the last residual block, a final fullyconnect operation maps the output to the number of classes in the
% target data.

function dlY = model(dlX,learnables,labels,hyperparameters,doTraining,getFeat)
```

```
if nargin<6 % If not specified does not extract features
    getFeat=false;
end
numBlocks = hyperparameters.NumBlocks;          % Number of residual blocks
dropoutFactor = hyperparameters.DropoutFactor;  % Dropout factor
dlY = dlX;

for k = 1:numBlocks % Definition of residual blocks
    dilationFactor = 2^(k-1); % The dilation factor increase at each consecutive layer
    nameBlock = "Block"+k;   % Parameter of the block
    dlY = residualBlock(dlY,dilationFactor,dropoutFactor,nameBlock,learnables,labels,doTraining);   % Residual block function
end

if getFeat
    Features=dlY; % Extraction of the features
end

weights = learnables{labels=="FC.Weights"}; Definition of the fully connected layer.
bias = learnables{labels=="FC.Bias"};
dlY = fullyconnect(dlY,weights,bias,'DataFormat','CBT');

dlY = softmax(dlY,'DataFormat','CBT'); % Softmax activation function.

if getFeat
    dlY=Features; % If the feature mode is activated the output are the complex features
end
end
```

residualBlock.m function

```
% The function residualBlock implements the core building block of the temporal convolutional network.

function dlY = residualBlock(dlX,dilationFactor,dropoutFactor,nameBlock,learnables,labels,doTraining)

filterSize = size(weights,1); % Size of the filter
paddingSize = (filterSize - 1) * dilationFactor;    % Padding of the convolution as (size of the filter-1)*dilation

weights = learnables{labels==nameBlock+".Conv1.Weights"};
bias = learnables{labels==nameBlock+".Conv1.Bias"};


dlY = dlconv(dlX,weights,bias, ... % Convolution
    'DataFormat','CBS', ...
    'Stride', 1, ...
    'DilationFactor', dilationFactor, ...
    'Padding', [paddingSize; 0] );

dlY = instanceNormalization(dlY,'CBS');     % Instance normalization function
dlY = relu(dlY); % Relu
dlY = spatialDropout(dlY,dropoutFactor,'CBS',doTraining);   % Spatial dropout function

weights = learnables{labels==nameBlock+".Conv2.Weights"};
bias = learnables{labels==nameBlock+".Conv2.Bias"};

dlY = dlconv(dlY,weights,bias, ... % Convolution 2
    'DataFormat','CBS', ...
    'Stride', 1, ...
    'DilationFactor', dilationFactor, ...
    'Padding',[paddingSize; 0] );
```

```
dlY = instanceNormalization(dlY,'CBS');
dlY = relu(dlY);
dlY = spatialDropout(dlY,dropoutFactor,'CBS',doTraining);

if ~isequal(size(dlX),size(dlY)) % In case channel input are not equal to channel output, optional 1-by-1 convolution.
    weights = learnables{labels==nameBlock+".Conv3.Weights"};
    bias = learnables{labels==nameBlock+".Conv3.Bias"};

    dlX = dlconv(dlX,weights,bias,'DataFormat','CBS');
end

dlY = relu(dlX+dlY); % Addition and ReLU

end
```

instanceNormalization.m function

% The instanceNormalization function normalizes the input dlX by first calculating the mean μ and the variance σ^2 for each

% observation over each input channel. Then it calculates the normalized activations as

$$\widehat{X} = \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}}.$$

% In comparison to batch normalization, the mean and variance is different for each observation in the mini-batch. Use

% normalization, such as instance normalization, between convolutional layers and nonlinearities to speed up training of

% convolutional neural networks and improve convergence.

```
function dlY = instanceNormalization(dlX,fmt)

reductionDims = find(fmt == 'S');
mu = mean(dlX,reductionDims);
```

```
sigmaSq = var(dlX,1,reductionDims);
epsilon = 1e-5;
dlY = (dlX-mu) ./ sqrt(sigmaSq+epsilon);
end
```

spatialDropout.m function

```
The spatialDropout function performs spatial dropout [3] on the input dlX with dimension labels fmt when the doTraining flag is
true. Spatial dropout drops an entire channel of the input data. That is, all time steps of a certain channel are dropped with the
probability specified by dropoutFactor. The channels are dropped out independently in the batch dimension.
```

```
function dlY = spatialDropout(dlX,dropoutFactor,fmt,doTraining)
if doTraining
    maskSize = size(dlX);
    maskSize(fmt=='S') = 1;
    dropoutScaleFactor = single(1 - dropoutFactor);
    dropoutMask = (rand(maskSize,'like',dlX) > dropoutFactor) / dropoutScaleFactor;

    dlY = dlX .* dropoutMask;
else
    dlY = dlX;
end
end
```

S.10 HybridNET.m function

The following code implements the Hybrid Convolutional-Recurrent Network (HYB) according to the parameters passed from the user. It can take as inputs the raw (i.e. directly from the IMU sensors) training data (R), the elaborated (see S.9 – feature mode) training data from the previously trained TCN (E), the relative target labels (Y), the optimised TCN parameters/hyperparameters (TCNpar, TCNhypar) the number of hidden neurons of the recurrent LSTM part of the network to be trained (Neurons), the maximum number of training epochs allowed (maxEpochs), the number of training samples for each mini-batch subset (miniBatchSize), the initial learning rate (initialLearnRate), the learning drop factor (learnRateDropFactor), the period that marks the dropping of the learning rate (learnRateDropPeriod), the threshold for gradient clipping (gradientThreshold), the period that marks the validation checks (validationFrequency), the L2 regularisation constant (l2Regularization), the execution environment (CPU/GPU) (executionEnvironment), the plotting option (plot – chose whenever to plot the training progress), and the functioning mode (mode). It can give as outputs the overall accuracy on the validation (or either test) set (validationAccuracy), the overall accuracy on the training set (trainingAccuracy), the average prediction time on the validation set (TimePred), the predictions of the network with the relative labelled targets (pred/targ), the set of parameters of the trained recurrent LSTM part of the HYB network (LSTMpar), and the set of hyperparameters of the trained recurrent LSTM part of the HYB network (LSTMhypar). The mode parameter controls the four different work modalities of the function:

- VALIDATION: In validation mode, the HYB is trained and validated across the different folds of the Design set, the function takes all inputs and it gives as outputs the “validationAccuracy”, the “trainingAccuracy”, and the “TimePred” parameters to evaluate the validation performance;
- TEST: In test mode, the HYB is trained, following the optimal model characteristics (from the validation mode), over the entire Design test and validated over the Test set, and it gives as outputs the “validationAccuracy”, the “trainingAccuracy”, the “TimePred”, the “pred”, and the “targ” parameters to evaluate the test performance;
- TRAINING: In training mode, the HYB is trained following the optimal model characteristics (from the test mode) over the entire Design test and validated over the Test set, the function takes all inputs parameters and it gives as outputs the “LSTMpar”, and the “LSTMhypar” parameters to reproduce the optimal combined neural network;

```
% This function implement a hybrid network composed by a Temporal Convolutional and LSTM network in training validation and test
mode. The supporting functions called by this algorithm are reported and briefly described after the main routine. Some are
equivalent to the one called by TCN.m and for this reason are omitted.


function [validationAccuracy,trainingAccuracy,TimePred,pred,targ,LSTMpar,LSTMhypar] =
HybridNET(R,E,Y,TCNpar,TCNhypar,Neurons,maxEpochs,miniBatchSize,initialLearnRate,learnRateDropFactor,learnRateDropPeriod,gradientThr
eshold,validationFrequency,l2Regularization,executionEnvironment,plots,mode)
% INPUT

%             R: Accelerometric data extracted from IMU [F x 1] cell array containing [S x 1] cell arrays containing [Feat x T]
%             elements.Where F is the number of features, S is the number of sequences, Feat is the number of features and T is the
%             number of time steps
%             E: Elaborated data extracted from the Pre-Trained TCN [F x 1] cell array containing [S x 1] cell arrays containing
%             [Feat x T] elements.Where F is the number of features, S is the number of sequences, Feat is the number of features
%             and T is the number of time steps
%             Y: Response data [F x 1] cell array containing [S x 1] cell arrays containing [1 x T] elements.
%             TCNpar= parameter of the pre-trained TCN architecture
%             TCNhypar= hyparameter of the pre-trained TCN architecture
%             Neurons= Neurons of the LSTM network
%             maxEpochs: Number of training epochs.
%             miniBatchSize: Number of sequences inside a batch of training.
%             initialLearnRate: Initial learning rate.
%             learnRateDropFactor,learnRateDropPeriod: These two parameters describe the learning rate decay process.
%             gradientThreshold: Threshold for gradient clipping.
%             validationFrequency: Number of epochs after which the trained network is tested on validation data.
%             l2Regularization: l2 regularization rate.
%             executionEnvironment: 'gpu' or 'cpu'.
%             plots: 'training-progress' or 'none'.
%             mode: 'validation','test','training'
% OUTPUT      validationAccuracy: The accuracy of the network on the set of data held out from the training.
%             trainingAccuracy: The accuracy of the network on the set of training.
%             TimePred: Prediction time of the network on the set of data held out from the training
%             pred: Predicted outputs.
%             targ: Target outputs
%             LSTMpar: Parameters of the model
%             LSTMhypar: Hyperparameters of the model




% VALIDATION MODE: Iteration of the training-validation process across the dataset folds
% INPUTS: All
% OUTPUTS: validationAccuracy, trainingAccuracy, TimePred

    if mode=="validation"
        Loop_Val_Acc = ones(numel(E),1); % Performance metrics, vectors of folds-1 elements
        Loop_Train_Acc = ones(numel(E),1);

        for v=1:numel(E)        % Validation loops
            idt=1:numel(E);
            idt(v)=[];

            ETrn=vertcat(E{idt});YTrn=vertcat(Y{idt});  % Training set (Elaborated features)
            RVal=vertcat(R{v});                         % Validation set (Raw data)
            EVal=vertcat(E{v});                         % Validation set (Elaborated data)
            YVal=vertcat(Y{v});

            numSequences = numel(ETrn);                 % Number of sequences
            numInputFeatures=size(ETrn{1},1);           % Number of features
            classes = categories(YTrn{1});
            numClasses = numel(classes);                % Number of classes

            LSTMhypar = struct;                         % Hyperparameters of the LSTM network
            dropoutFactor = 0.05;
```

```
LSTMhypar.DropoutFactor = dropoutFactor;    % Dropout factor
LSTMhypar.numHiddenN = Neurons;             % Hidden neurons
LSTMhypar.numClasses = numClasses;          % Number of classes

LSTMpar = table([],[],'VariableNames',{'Parameter','Value'});   % Initialise the learnable parameters for the LSTM
par = {...                          % Labels of the parameters
    "LSTM.Weights";...
    "LSTM.recurrentWeights";...
    "LSTM.Bias";...
    "FC.Weights";...
    "FC.Bias"...
    };
var = {...                          % Values of parameters
    dlarray(HeinitializeGaussian([4*Neurons,numInputFeatures]),'CU');...
    dlarray(HeinitializeGaussian([4*Neurons,Neurons]),'CU');...
    dlarray(zeros(4*Neurons,1,'single'),'CU');...
    dlarray(HeinitializeGaussian([numClasses,Neurons]),'CU');...
    dlarray(zeros(numClasses,1,'single'),'CU')...
    };
LSTMpar = [LSTMpar;par,var];        % Compose the learnables table

learnRate = initialLearnRate;       % Variables for adaam optimizer
trailingAvg = [];
trailingAvgSq = [];

if plots == "training-progress"     % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    lineAccuracyValidE =  animatedline('Color',[1 0 0]);
    lineAccuracyValidR =  animatedline('Color',[0 0 1]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on
    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    lineLossValid = animatedline('Color',[1 0 0]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on

    %if exist('conf','var') == 0;conf=figure;end             % If the accuracy-loss plot does not exist create it
end

iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize); % Number of iterations per epochs

for epoch = 1:maxEpochs                 % for each epoch
    idx = randperm(numSequences);       % shuffle data at each loop
    ETrn = ETrn(idx);                   % training data on elaborated features
    YTrn = YTrn(idx);                   % training response

    for i = 1:numIterationsPerEpoch                     % for each iteration
        iteration = iteration + 1;                      % increment iteration counter
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;    % indexes of observation

        % This section calls at the function transformSequences.m for pre-processing
        [TrainX,TrainY,Train_TSteps] = transformSequences(ETrn(idx),YTrn(idx));
        %[ValXE,ValY,ValE_TSteps] = transformSequences(EVal,YVal);
        dlTrainX = dlarray(TrainX,'CBT');
```

```
%dlValXE = dlarray(ValXE,'CBT');
dlValXR = cellfun(@(x)dlarray(x),RVal,'UniformOutput',false);

% Gradient calculation
[gradients, Train_Loss, Train_Acc] = dlfeval(...
@modelGradients, ...
dlTrainX, ...
TrainY, ...
LSTMpar.Value, ...
LSTMpar.Parameter, ...
LSTMhypar, ...
Train_TSteps);
% Regularization
idx = contains(LSTMpar.Parameter,"Weights");
% Update the gradients
gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), LSTMpar.Value(idx,:));
% Gradient clipping
gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
% Adam Optimizer
[LSTMpar.Value,trailingAvg,trailingAvgSq] = adamupdate(LSTMpar.Value,gradients, ...
       trailingAvg, trailingAvgSq, iteration, learnRate);
% Validation of the model (at first iteration and at a defined frequency
if iteration == 1 || mod(iteration,validationFrequency) == 0
    [Val_Loss, ValE_Acc,~,~,~] = modelValidation(... % Validation on elaborated inputs
    dlValXE, ...
    ValY, ...
    LSTMpar.Value, ...
    LSTMpar.Parameter, ...
    LSTMhypar, ...
    ValE_TSteps);
```

```
        [ValR_Acc,targ,pred,TimePred] = modelCValidation(... % Validation on raw inputs
        dlValXR, ...
        YVal, ...
        TCNpar, ...
        TCNhypar, ...
        LSTMpar.Value, ...
        LSTMpar.Parameter,
        LSTMhypar);

        if plots == "training-progress"     % Plot validation progress
            figure(lossacc);
            addpoints(lineLossValid,iteration, Val_Loss);
            addpoints(lineAccuracyValidE,iteration, ValE_Acc);
            addpoints(lineAccuracyValidR,iteration, ValR_Acc);
            %figure(conf);
            %plotconfusion(targ,pred)
        end
    end
    if plots == "training-progress" % Training progress (if plot is enabled)
        Train_Loss = mean(Train_Loss ./ Train_TSteps); % Normalize the loss over the sequence lengths
        Train_Loss = double(gather(extractdata(Train_Loss)));
        Train_Loss = mean(Train_Loss);
        figure(lossacc);
        addpoints(lineLossTrain,iteration, mean(Train_Loss));
        addpoints(lineAccuracyTrain,iteration, Train_Acc);
        title("Epoch: " + epoch)
        drawnow
    end
end
if mod(epoch,learnRateDropPeriod) == 0
```

```
                    learnRate = learnRate*learnRateDropFactor;
                end
                fprintf(['EPOCH ',num2str(epoch),' Validation = %f | Training = %f\n'],ValR_Acc,Train_Acc);
            end

            Loop_Val_Acc(v)=ValR_Acc;
            Loop_Train_Acc(v)=Train_Acc;
            fprintf(['FOLD ',num2str(v),' Validation = %f | Training = %f\n'],Loop_Val_Acc(v),Loop_Train_Acc(v));
        end

        validationAccuracy=mean(Loop_Val_Acc);
        trainingAccuracy=mean(Loop_Train_Acc);
        fprintf('Validation = %f | Training = %f\n',validationAccuracy,trainingAccuracy);

% TEST MODE: training the net on the entire train+validation sets and evaluating on the test set
% INPUTS: All
% OUTPUTS: validationAccuracy, trainingAccuracy, TimePred, pred, targ

    elseif mode=="test"
        ETrn=vertcat(E{1:end-1});YTrn=vertcat(Y{1:end-1});  % Training on elaborated features
        numSequences = numel(ETrn);                          % Number of sequences
        numInputFeatures=size(ETrn{1},1);                    % Number of features
        classes = categories(YTrn{1});                       % Number of classes
        numClasses = numel(classes);
        LSTMhypar = struct;                          % Hyperparameters of the LSTM network
        dropoutFactor = 0.05;                        % Dropout factor
        LSTMhypar.DropoutFactor = dropoutFactor;
        LSTMhypar.numHiddenN = Neurons;              % Hidden neurons
        LSTMhypar.numClasses = numClasses;           % Number of classes

        LSTMpar = table([],[],'VariableNames',{'Parameter','Value'}); % Initialise the learnable parameters for the LSTM
```

```
par = {...                                  % Labels of the parameters
    "LSTM.Weights";...
    "LSTM.recurrentWeights";...
    "LSTM.Bias";...
    "FC.Weights";...
    "FC.Bias"...
    };
var = {...                                  % Values of parameters
    dlarray(HeinitializeGaussian([4*Neurons,numInputFeatures]),'CU');...
    dlarray(HeinitializeGaussian([4*Neurons,Neurons]),'CU');...
    dlarray(zeros(4*Neurons,1,'single'),'CU');...
    dlarray(HeinitializeGaussian([numClasses,Neurons]),'CU');...
    dlarray(zeros(numClasses,1,'single'),'C')...
    };
LSTMpar = [LSTMpar;par,var];                % Compose the learnables table

learnRate = initialLearnRate;   % Variables for adaam optimizer
trailingAvg = [];
trailingAvgSq = [];
if plots == "training-progress"     % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end  % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on

    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on
end

iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize);     % Number of iterations per epochs

for epoch = 1:maxEpochs                         % for each epoch
    idx = randperm(numSequences);               % shuffle data at each loop
    ETrn = ETrn(idx);                           % training data on elaborated features
    YTrn = YTrn(idx);                           % training response% training response

    for i = 1:numIterationsPerEpoch                     % for each iteration
        iteration = iteration + 1;                      % increment iteration counter
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;    % indexes of observation

        % This section calls at the function transformSequences.m for pre-processing
        [TrainX,TrainY,Train_TSteps] = transformSequences(ETrn(idx),YTrn(idx));
        dlTrainX = dlarray(TrainX,'CBT');

        % Gradient calculation
        [gradients, Train_Loss, trainingAccuracy] = dlfeval(...
        @modelGradients, ...
        dlTrainX, ...
        TrainY, ...
        LSTMpar. ...
        Value, ...
        LSTMpar.Parameter, ...
        LSTMhypar, ...
        Train_TSteps);
        % Regularization
        idx = contains(LSTMpar.Parameter,"Weights");
        % Update the gradients
        gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), LSTMpar.Value(idx,:));

        % Gradient clipping
        gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
        % Adam optimizer
        [LSTMpar.Value,trailingAvg,trailingAvgSq] = adamupdate(LSTMpar.Value,gradients, ...
            trailingAvg, trailingAvgSq, iteration, learnRate);

        if plots == "training-progress"         % Plot training progress
            Train_Loss = mean(Train_Loss ./ Train_TSteps);
            Train_Loss = double(gather(extractdata(Train_Loss)));
            Train_Loss = mean(Train_Loss);
            figure(lossacc);
            addpoints(lineLossTrain,iteration, mean(Train_Loss));
            addpoints(lineAccuracyTrain,iteration, trainingAccuracy);
            title("Epoch: " + epoch)
            drawnow
        end
    end
    if mod(epoch,learnRateDropPeriod) == 0
        learnRate = learnRate*learnRateDropFactor;
    end
end

fprintf('Training accuracy = %f \n',trainingAccuracy);

Etst=vertcat(E{end});Ytst=vertcat(Y{end}); % Test set definition
Rtst=vertcat(R{end});
```

```
    numObservationsTest = numel(Etst);
    doTraining = false;

    %dlTestXE=cellfun(@(x)dlarray(reshape(x,[size(x,1),1,size(x,2)]),'CBT'),Etst,'UniformOutput',false);
    dlTestXR = cellfun(@(x)dlarray(x),Rtst,'UniformOutput',false);
    TestY=cellfun(@(x)reshape(single(full(ind2vec(double(x),numClasses))),[numClasses,1,size(x,2)]),Ytst,'UniformOutput',false);

    %Test_AccE = zeros(1,numObservationsTest);
    Test_AccR = zeros(1,numObservationsTest);
    Times = zeros(1,numObservationsTest);
    pred = cell(1,numObservationsTest);
    targ = cell(1,numObservationsTest);

    for obs=1:numObservationsTest
        %dlYPredE = model(dlTestXE{obs},LSTMpar.Value,LSTMpar.Parameter,LSTMhypar,doTraining);
        start=tic;
        dlYPredR = modelC(dlTestXR{obs},TCNpar,TCNhypar,LSTMpar.Value,LSTMpar.Parameter,LSTMhypar,doTraining);
        Times(obs) = seconds(duration(0,0,toc(start),'Format',"mm:ss.SSS"));
        YPred = gather(extractdata(dlYPredR));
        pred{obs} = YPred;
        targ{obs} = squeeze(TestY{obs});
        [~,idxPred] = max(pred{obs},[],1);
        [~,idxTest] = max(targ{obs},[],1);
        Test_AccR(obs) = mean(idxPred == idxTest);
    end

    validationAccuracy=mean(Test_AccR);
    TimePred=mean(Times);

% TRAINING MODE: training the net on the entire train+dev sets and evaluating on the test set
```

```
elseif mode=="training"

    ETrn=vertcat(E{1:end});YTrn=vertcat(Y{1:end});  % Training on elaborated features
    numSequences = numel(ETrn);                         % Number of sequences
    numInputFeatures=size(ETrn{1},1);                   % Number of features
    classes = categories(YTrn{1});                      % Number of classes
    numClasses = numel(classes);
    LSTMhypar = struct;                         % Hyperparameters of the LSTM network
    dropoutFactor = 0.05;                       % Dropout factor
    LSTMhypar.DropoutFactor = dropoutFactor;
    LSTMhypar.numHiddenN = Neurons;             % Hidden neurons
    LSTMhypar.numClasses = numClasses;          % Number of classes

    LSTMpar = table([],[],'VariableNames',{'Parameter','Value'}); % Initialise the learnable parameters for the LSTM
    par = {...                                  % Labels of the parameters
        "LSTM.Weights";...
        "LSTM.recurrentWeights";...
        "LSTM.Bias";...
        "FC.Weights";...
        "FC.Bias"...
        };
    var = {...                                  % Values of parameters
        dlarray(HeinitializeGaussian([4*Neurons,numInputFeatures]),'CU');...
        dlarray(HeinitializeGaussian([4*Neurons,Neurons]),'CU');...
        dlarray(zeros(4*Neurons,1,'single'),'CU');...
        dlarray(HeinitializeGaussian([numClasses,Neurons]),'CU');...
        dlarray(zeros(numClasses,1,'single'),'C')...
        };
    LSTMpar = [LSTMpar;par,var];                % Compose the learnables table

    learnRate = initialLearnRate;   % Variables for adaam optimizer
```

```
trailingAvg = [];
trailingAvgSq = [];
if plots == "training-progress"     % training and validation progress
    if exist('lossacc','var') == 0;lossacc=figure;end  % If the accuracy-loss plot does not exist create it
    subplot(2,1,1)
    lineAccuracyTrain =  animatedline('Color',[0 0 0]);
    ylim([0 inf])
    ylabel("Accuracy")
    grid on

    subplot(2,1,2)
    lineLossTrain = animatedline('Color',[0 0 0]);
    ylim([0 inf])
    xlabel("Iteration")
    ylabel("Loss")
    grid on
end

iteration = 0; % Iteration counter
numIterationsPerEpoch = floor(numSequences./miniBatchSize);     % Number of iterations per epochs

for epoch = 1:maxEpochs                              % for each epoch
    idx = randperm(numSequences);                    % shuffle data at each loop
    ETrn = ETrn(idx);                                % training data on elaborated features
    YTrn = YTrn(idx);                                % training response% training response

    for i = 1:numIterationsPerEpoch                          % for each iteration
        iteration = iteration + 1;                           % increment iteration counter
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;    % indexes of observation
```

```
% This section calls at the function transformSequences.m for pre-processing
[TrainX,TrainY,Train_TSteps] = transformSequences(ETrn(idx),YTrn(idx));
dlTrainX = dlarray(TrainX,'CBT');

% Gradient calculation
[gradients, Train_Loss, trainingAccuracy] = dlfeval(...
@modelGradients, ...
dlTrainX, ...
TrainY, ...
LSTMpar. ...
Value, ...
LSTMpar.Parameter, ...
LSTMhypar, ...
Train_TSteps);
% Regularization
idx = contains(LSTMpar.Parameter,"Weights");
% Update the gradients
gradients(idx,:) = dlupdate(@(g,w) g + l2Regularization*w, gradients(idx,:), LSTMpar.Value(idx,:));
% Gradient clipping
gradients = dlupdate(@(g) thresholdL2Norm(g,gradientThreshold),gradients);
% Adam optimizer
[LSTMpar.Value,trailingAvg,trailingAvgSq] = adamupdate(LSTMpar.Value,gradients, ...
    trailingAvg, trailingAvgSq, iteration, learnRate);

if plots == "training-progress"         % Plot training progress
    Train_Loss = mean(Train_Loss ./ Train_TSteps);
    Train_Loss = double(gather(extractdata(Train_Loss)));
    Train_Loss = mean(Train_Loss);
    figure(lossacc);
    addpoints(lineLossTrain,iteration, mean(Train_Loss));
```

```
                    addpoints(lineAccuracyTrain,iteration, trainingAccuracy);
                    title("Epoch: " + epoch)
                    drawnow
                end
            end
            if mod(epoch,learnRateDropPeriod) == 0
                learnRate = learnRate*learnRateDropFactor;
            end
        end

        fprintf('Training accuracy = %f \n',trainingAccuracy);

    end
    switch mode
    case "validation"
        pred=[];targ=[];LSTMpar=[];LSTMhypar=[];
    case "test"
        LSTMpar=[];LSTMhypar=[];
    case "training"
        validationAccuracy=[];TimePred=[];pred=[];targ=[];
    end
end
```

S.10.1 Supporting functions

modelValidation.m function

```
% This function implement the validation of the neural network relying on the elaborated data extracted from the previous training
% of a TCN net. Hence, the model function used in this context have to build just the LSTM part of the hybrid implementation.

function [Val_Loss,Val_Acc,Targ,Pred,TimePred] = modelValidation(dlXV,YVal,LSTMvalue,LSTMlabels,LSTMhypar,TimeSteps)
```

```
numObsv_Val=size(YVal,2);   % number of observation for the validation set
startV=tic;                 % start timer prediction
dlYpred = model(dlXV,LSTMvalue,LSTMlabels,LSTMhypar,false);     % model based on elaborated features
TimePred = duration(0,0,toc(startV),'Format',"mm:ss.SSS")/numObsv_Val; % stop timer prediction
TimePred = seconds(TimePred); % Save the result in seconds
dlYVal = dlarray(YVal,'CBT');           % formatting y responses
YPred = gather(extractdata(dlYpred));   % gather prediction values
Accuracy = zeros(1,numObsv_Val);        % accuracy
Pred=zeros(size(YPred,1),size(YPred,2)*size(YPred,3));
Targ=zeros(size(YVal,1),size(YVal,2)*size(YVal,3));
for i = 1:numObsv_Val % accuracy for each observation
    Pred(:,((i-1)*size(YPred,3))+1:(i*size(YPred,3)))=squeeze(YPred(:,i,:));
    Targ(:,((i-1)*size(YVal,3))+1:(i*size(YVal,3)))=squeeze(YVal(:,i,:));
    [~,idxP] = max(YPred(:,i,:),[],1);
    [~,idxT] = max(YVal(:,i,:),[],1);
    Accuracy(i) = mean(idxP == idxT);
end
Val_Acc=mean(Accuracy);                 % average value of accuracy
Val_Loss = maskedCrossEntropyLoss(dlYpred, dlYVal, TimeSteps);
Val_Loss = mean(Val_Loss ./ TimeSteps);
Val_Loss = double(gather(extractdata(Val_Loss)));
Val_Loss = mean(Val_Loss);
end
```

modelCValidation.m function

```
% This function implement the validation of the neural network relying on raw accelerometric data. Hence the function have to build
% the entire model comprised of convolutional network.

function [accuracyV,targ,pred,TimePred] = modelCValidation(dlValXR,YVal,TCNpar,TCNhypar,learnables,labels,hyperparameters)
```

```
numObservationsTest=numel(YVal);    % number of observation for the validation set
dlY=cell(size(YVal));
TimePred=0;
for obs=1:numObservationsTest
    startV=tic;                               % start timer prediction
    dlY{obs} = modelC(dlValXR{obs},TCNpar,TCNhypar,learnables,labels,hyperparameters,false);
    TimePred = TimePred+seconds(duration... % stop timer
        (0,0,toc(startV),'Format',"mm:ss.SSS"));
end
TimePred=TimePred/numObservationsTest;      % average prediction time
YVal=cellfun(@(x)single(full(ind2vec(double(x),4))),YVal,'UniformOutput',false);
YP=cellfun(@(x)gather(extractdata(x)),dlY,'UniformOutput',false);
acc = zeros(1,numObservationsTest);         % accuracy
for i = 1:numObservationsTest
    [~,idxP] = max(YP{i},[],1);
    [~,idxT] = max(YVal{i},[],1);
    acc(i) = mean(idxP == idxT);
end
targ=horzcat(YVal{:});
pred=horzcat(YP{:});
accuracyV=mean(acc);
end
```

model.m Function

```
% This function implement the hybrid model relying on pre-elaborated features, hence it defines just the LSTM part of the
% architecture

function dlY = model(dlX,LSTMvalues,LSTMlabels,LSTMhypar,doTraining)
dropoutFactor = LSTMhypar.DropoutFactor;    % dropoutfactor
```

```
H0 = zeros(LSTMhypar.numHiddenN,1);         % hiddenstates
C0 = zeros(LSTMhypar.numHiddenN,1);
weights = LSTMvalues{LSTMlabels=="LSTM.Weights"};  % weights
recweights = LSTMvalues{LSTMlabels=="LSTM.recurrentWeights"};  % recurrent weights
bias = LSTMvalues{LSTMlabels=="LSTM.Bias"};        % bias
[dlY,~,~] = lstm(dlX,H0,C0,weights,recweights,bias);    % lstm layer
dlY = Dropout(dlY,dropoutFactor,doTraining);    % dropout function (just in training)
weights = LSTMvalues{LSTMlabels=="FC.Weights"}; %
bias = LSTMvalues{LSTMlabels=="FC.Bias"};
dlY = fullyconnect(dlY,weights,bias);
dlY = softmax(dlY);
end
```

modelC.m function

```
% This function implement the hybrid model relying on raw accelerometric data, hence it has to implement both the pre-trained TCN
% and the LSTM architecture

function dlY = modelC(dlX,TCNpar,TCNhypar,learnables,labels,hyperparameters,doTraining)
dropoutFactor = TCNhypar.DropoutFactor;  % Fattore di dropout
numBlocks = TCNhypar.NumBlocks;          % Number of residual blocks
dlY = dlX;
for k = 1:numBlocks
    dilationFactor = 2^(k-1); % The dilation factor increase at each consecutive layer
    dlY = residualBlock(dlY,dilationFactor,dropoutFactor,TCNpar,k,doTraining);   % Residual block function
end
H0 = zeros(hyperparameters.numHiddenN,1);
C0 = zeros(hyperparameters.numHiddenN,1);
weights = learnables{labels=="LSTM.Weights"};
recweights = learnables{labels=="LSTM.recurrentWeights"};
bias = learnables{labels=="LSTM.Bias"};
```

```
[dlY,~,~] = lstm(dlY,H0,C0,weights,recweights,bias,'DataFormat','CT');
dlY = Dropout(dlY,dropoutFactor,doTraining);
weights = learnables{labels=="FC.Weights"};
bias = learnables{labels=="FC.Bias"};
dlY = fullyconnect(dlY,weights,bias,'DataFormat','TC');
dlY = softmax(dlY,'DataFormat','CT');
end
```

Dropout.m function

The Dropout function performs spatial dropout [3] on the input dlX with dimension labels fmt when the doTraining flag is true. Dropout drops random samples of the input data.

```
function dlY = Dropout(dlX,dropoutFactor,doTraining)
if doTraining
    maskSize = size(dlX);
    dropoutScaleFactor = single(1 - dropoutFactor);
    dropoutMask = (rand(maskSize,'like',dlX) > dropoutFactor) / dropoutScaleFactor;

    dlY = dlX .* dropoutMask;
else
    dlY = dlX;
end
end
```

References

[1] United Nations, Department of Economic and Social Affairs, Population Division, *World population ageing, 2019 highlights.* 2020.

[2] «WHO | What is Healthy Ageing?», *WHO*. http://www.who.int/ageing/healthy-ageing/en/ (consultato ago. 07, 2020).

[3] H.-C. Hsu, «Exploring elderly people's perspectives on successful ageing in Taiwan», *Ageing & Society*, vol. 27, n. 1, pagg. 87–102, gen. 2007, doi: 10.1017/S0144686X06005137.

[4] J. F. D. de Moraes e V. B. de A. e Souza, «Factors associated with the successful aging of the socially-active elderly in the metropolitan region of Porto Alegre», *Brazilian Journal of Psychiatry*, vol. 27, n. 4, pagg. 302–308, dic. 2005, doi: 10.1590/S1516-44462005000400009.

[5] K. M. Bennett, «Social engagement as a longitudinal predictor of objective and subjective health», *Eur J Ageing*, vol. 2, n. 1, pagg. 48–55, mar. 2005, doi: 10.1007/s10433-005-0501-z.

[6] R. S. Wilson *et al.*, «Participation in cognitively stimulating activities and risk of incident Alzheimer disease», *JAMA*, vol. 287, n. 6, pagg. 742–748, feb. 2002, doi: 10.1001/jama.287.6.742.

[7] C. Schooler e M. S. Mulatu, «The reciprocal effects of leisure time activities and intellectual functioning in older people: a longitudinal analysis», *Psychol Aging*, vol. 16, n. 3, pagg. 466–482, set. 2001, doi: 10.1037//0882-7974.16.3.466.

[8] N. Nakanishi e K. Tatara, «Correlates and Prognosis in Relation to Participation in Social Activities Among Older People Living in a Community in Osaka, Japan», *Journal of Clinical Geropsychology*, vol. 6, n. 4, pagg. 299–307, ott. 2000, doi: 10.1023/A:1009538913527.

[9] E. Rubio, A. Lázaro, e A. Sánchez-Sánchez, «Social participation and independence in activities of daily living: a cross sectional study», *BMC Geriatrics*, vol. 9, n. 1, pag. 26, lug. 2009, doi: 10.1186/1471-2318-9-26.

[10] Y. R. Mao *et al.*, «The Crucial Changes of Sit-to-Stand Phases in Subacute Stroke Survivors Identified by Movement Decomposition Analysis», *Front Neurol*, vol. 9, mar. 2018, doi: 10.3389/fneur.2018.00185.

[11] W. Saensook, L. Mato, N. Manimmanakorn, P. Amatachaya, T. Sooknuan, e S. Amatachaya, «Ability of sit-to-stand with hands reflects neurological and functional impairments in ambulatory individuals with spinal cord injury», *Spinal Cord*, vol. 56, n. 3, Art. n. 3, mar. 2018, doi: 10.1038/s41393-017-0012-8.

[12] R. C. van Lummel, S. Walgaard, A. B. Maier, E. Ainsworth, P. J. Beek, e J. H. van Dieën, «The Instrumented Sit-to-Stand Test (iSTS) Has Greater Clinical Relevance than the Manually Recorded Sit-to-Stand Test in Older Adults», *PLoS One*, vol. 11, n. 7, lug. 2016, doi: 10.1371/journal.pone.0157968.

[13] J. Howcroft, J. Kofman, e E. D. Lemaire, «Review of fall risk assessment in geriatric populations using inertial sensors», *J Neuroeng Rehabil*, vol. 10, n. 1, pag. 91, ago. 2013, doi: 10.1186/1743-0003-10-91.

[14] S. R. Lord, C. Sherrington, H. B. Menz, e J. C. T. Close, *Falls in Older People: Risk Factors and Strategies for Prevention*, 2ª ed. Cambridge: Cambridge University Press, 2007. doi: 10.1017/CBO9780511722233.

[15] M. A. Habib, M. S. Mohktar, S. B. Kamaruzzaman, K. S. Lim, T. M. Pin, e F. Ibrahim, «Smartphone-Based Solutions for Fall Detection and Prevention: Challenges and Open Issues», *Sensors (Basel)*, vol. 14, n. 4, pagg. 7181–7208, apr. 2014, doi: 10.3390/s140407181.

[16] D. W. Vander Linden, D. Brunt, e M. U. McCulloch, «Variant and invariant characteristics of the sit-to-stand task in healthy elderly adults», *Arch Phys Med Rehabil*, vol. 75, n. 6, pagg. 653–660, giu. 1994, doi: 10.1016/0003-9993(94)90188-0.

[17] J. H. Fong, «Disability incidence and functional decline among older adults with major chronic diseases», *BMC Geriatr*, vol. 19, nov. 2019, doi: 10.1186/s12877-019-1348-z.

[18] D. D. Dunlop, S. L. Hughes, e L. M. Manheim, «Disability in activities of daily living: patterns of change and a hierarchy of disability.», *Am J Public Health*, vol. 87, n. 3, pagg. 378–383, mar. 1997.

[19] J. S. Munton, M. I. Ellis, M. A. Chamberlain, e V. Wright, «An investigation into the problems of easy chairs used by the arthritic and the elderly», *Rheumatol Rehabil*, vol. 20, n. 3, pagg. 164–173, ago. 1981, doi: 10.1093/rheumatology/20.3.164.

[20] M. Brod, G. A. Mendelsohn, e B. Roberts, «Patients' experiences of Parkinson's disease», *J Gerontol B Psychol Sci Soc Sci*, vol. 53, n. 4, pagg. P213-222, lug. 1998, doi: 10.1093/geronb/53b.4.p213.

[21] G. Yamako, E. Chosa, K. Totoribe, Y. Fukao, e G. Deng, «Quantification of the sit-to-stand movement for monitoring age-related motor deterioration using the Nintendo Wii Balance Board», *PLoS ONE*, vol. 12, n. 11, pag. e0188165, 2017, doi: 10.1371/journal.pone.0188165.

[22] L. H. Sloot e M. M. van der Krogt, «Interpreting Joint Moments and Powers in Gait», in *Handbook of Human Motion*, Cham: Springer International Publishing, 2018, pagg. 625–643. doi: 10.1007/978-3-319-14418-4_32.

[23] M. W. Rodosky, T. P. Andriacchi, e G. B. J. Andersson, «The influence of chair height on lower limb mechanics during rising», *Journal of Orthopaedic Research*, vol. 7, n. 2, pagg. 266–271, 1989, doi: 10.1002/jor.1100070215.

[24] U. P. Arborelius, P. Wretenberg, e F. Lindberg, «The effects of armrests and high seat heights on lower-limb joint load and muscular activity during sitting and rising», *Ergonomics*, vol. 35, n. 11, pagg. 1377–1391, nov. 1992, doi: 10.1080/00140139208967399.

[25] F. Bahrami, R. Riener, P. Jabedar-Maralani, e G. Schmidt, «Biomechanical analysis of sit-to-stand transfer in healthy and paraplegic subjects», *Clin Biomech (Bristol, Avon)*, vol. 15, n. 2, pagg. 123–133, feb. 2000, doi: 10.1016/s0268-0033(99)00044-3.

[26] S. Ng, «Balance ability, not muscle strength and exercise endurance, determines the performance of hemiparetic subjects on the timed-sit-to-stand test», *Am J Phys Med Rehabil*, vol. 89, n. 6, pagg. 497–504, giu. 2010, doi: 10.1097/PHM.0b013e3181d3e90a.

[27] J. M. Glenn, M. Gray, e A. Binns, «Relationship of Sit-to-Stand Lower-Body Power With Functional Fitness Measures Among Older Adults With and Without Sarcopenia», *Journal of Geriatric Physical Therapy*, vol. 40, n. 1, pagg. 42–50, mar. 2017, doi: 10.1519/JPT.0000000000000072.

[28] T. W. Buford *et al.*, «Models of Accelerated Sarcopenia: Critical Pieces for Solving the Puzzle of Age-Related Muscle Atrophy», *Ageing Res Rev*, vol. 9, n. 4, pagg. 369–383, ott. 2010, doi: 10.1016/j.arr.2010.04.004.

[29] M. A. Hughes, B. S. Myers, e M. L. Schenkman, «The role of strength in rising from a chair in the functionally impaired elderly», *Journal of Biomechanics*, vol. 29, n. 12, pagg. 1509–1513, dic. 1996, doi: 10.1016/S0021-9290(96)80001-7.

[30] J. M. Guralnik *et al.*, «A short physical performance battery assessing lower extremity function: association with self-reported disability and prediction of mortality and nursing home admission», *J Gerontol*, vol. 49, n. 2, pagg. M85-94, mar. 1994, doi: 10.1093/geronj/49.2.m85.

[31] E. J. Bassey, M. Fiatarone, E. O'Neill, M. Kelly, W. Evans, e L. Lipsitz, «Leg extensor power and functional performance in very old men and women.», *Clinical science*, 1992, doi: 10.1042/CS0820321.

[32] S. R. Lord, S. M. Murray, K. Chapman, B. Munro, e A. Tiedemann, «Sit-to-stand performance depends on sensation, speed, balance, and psychological status in addition to strength in older people», *J. Gerontol. A Biol. Sci. Med. Sci.*, vol. 57, n. 8, pagg. M539-543, ago. 2002, doi: 10.1093/gerona/57.8.m539.

[33] S. E. Jones *et al.*, «The five-repetition sit-to-stand test as a functional outcome measure in COPD», *Thorax*, vol. 68, n. 11, pagg. 1015–1020, nov. 2013, doi: 10.1136/thoraxjnl-2013-203576.

[34] K. L. Schaubert e R. W. Bohannon, «Reliability and validity of three strength measures obtained from community-dwelling elderly persons», *J Strength Cond Res*, vol. 19, n. 3, pagg. 717–720, ago. 2005, doi: 10.1519/R-15954.1.

[35] S. Ozalevli, A. Ozden, O. Itil, e A. Akkoclu, «Comparison of the Sit-to-Stand Test with 6 min walk test in patients with chronic obstructive pulmonary disease», *Respir Med*, vol. 101, n. 2, pagg. 286–293, feb. 2007, doi: 10.1016/j.rmed.2006.05.007.

[36] R. W. Bohannon, «Sit-to-stand test for measuring performance of lower extremity muscles», *Percept Mot Skills*, vol. 80, n. 1, pagg. 163–166, feb. 1995, doi: 10.2466/pms.1995.80.1.163.

[37] S. L. Whitney, D. M. Wrisley, G. F. Marchetti, M. A. Gee, M. S. Redfern, e J. M. Furman, «Clinical measurement of sit-to-stand performance in people with balance disorders: validity of data for the Five-Times-Sit-to-Stand Test», *Phys Ther*, vol. 85, n. 10, pagg. 1034–1045, ott. 2005.

[38] C. Weng, N. Wang, e L. Liu, «The five times sit to stand test: a useful assessment tool for predicting falls in elderly», *Chinese Journal of Rehabilitation Medicine.*, vol. 27, pagg. 908–912, 2012, doi: 10.3969/j.issn.1001-1242.2012.10.004.

[39] S. Buatois *et al.*, «A simple clinical scale to stratify risk of recurrent falls in community-dwelling adults aged 65 years and older», *Phys Ther*, vol. 90, n. 4, pagg. 550–560, apr. 2010, doi: 10.2522/ptj.20090158.

[40] J. M. Guralnik *et al.*, «Lower extremity function and subsequent disability: consistency across studies, predictive models, and value of gait speed alone compared with the short physical performance battery», *J. Gerontol. A Biol. Sci. Med. Sci.*, vol. 55, n. 4, pagg. M221-231, apr. 2000, doi: 10.1093/gerona/55.4.m221.

[41] C. J. Jones, R. E. Rikli, e W. C. Beam, «A 30-s chair-stand test as a measure of lower body strength in community-residing older adults», *Res Q Exerc Sport*, vol. 70, n. 2, pagg. 113–119, giu. 1999, doi: 10.1080/02701367.1999.10608028.

[42] I. Bruun, B. Nørgaard, B. Schiøttz-Christensen, C. B. Mogensen, e T. Maribo, «The 30-second chair stand test and habitual mobility predict reduced physical ability after acute admission», *Physiotherapy*, vol. 102, pagg. e224–e225, nov. 2016, doi: 10.1016/j.physio.2016.10.277.

[43] N. A. de Morton, M. Davidson, e J. L. Keating, «The de Morton Mobility Index (DEMMI): an essential health index for an ageing world», *Health Qual Life Outcomes*, vol. 6, pag. 63, ago. 2008, doi: 10.1186/1477-7525-6-63.

[44] D. J. Rose, N. Lucchese, e L. D. Wiersma, «Development of a multidimensional balance scale for use with functionally independent older adults», *Arch Phys Med Rehabil*, vol. 87, n. 11, pagg. 1478–1485, nov. 2006, doi: 10.1016/j.apmr.2006.07.263.

[45] D. Podsiadlo e S. Richardson, «The timed "Up & Go": a test of basic functional mobility for frail elderly persons», *J Am Geriatr Soc*, vol. 39, n. 2, pagg. 142–148, feb. 1991, doi: 10.1111/j.1532-5415.1991.tb01616.x.

[46] M. Job, A. Dottor, A. Viceconti, e M. Testa, «Ecological Gait as a Fall Indicator in Older Adults: A Systematic Review», *Gerontologist*, vol. 60, n. 5, pagg. e395–e412, lug. 2020, doi: 10.1093/geront/gnz113.

[47] J. J. Craig, A. P. Bruetsch, S. G. Lynch, F. B. Horak, e J. M. Huisinga, «Instrumented balance and walking assessments in persons with multiple sclerosis show strong test-retest reliability», *Journal of NeuroEngineering and Rehabilitation*, vol. 14, n. 1, pag. 43, mag. 2017, doi: 10.1186/s12984-017-0251-0.

[48] M. H. Cameron, E. Thielman, R. Mazumder, e D. Bourdette, «Predicting falls in people with multiple sclerosis: fall history is as accurate as more complex measures», *Mult Scler Int*, vol. 2013, pag. 496325, 2013, doi: 10.1155/2013/496325.

[49] Y. Nilsagård, C. Lundholm, E. Denison, e L.-G. Gunnarsson, «Predicting accidental falls in people with multiple sclerosis -- a longitudinal study», *Clin Rehabil*, vol. 23, n. 3, pagg. 259–269, mar. 2009, doi: 10.1177/0269215508095087.

[50] W. G. Janssen, H. B. Bussmann, e H. J. Stam, «Determinants of the Sit-to-Stand Movement: A Review», *Phys Ther*, vol. 82, n. 9, pagg. 866–879, set. 2002, doi: 10.1093/ptj/82.9.866.

[51] B. Etnyre e D. Q. Thomas, «Event standardization of sit-to-stand movements», *Phys Ther*, vol. 87, n. 12, pagg. 1651–1666, dic. 2007, doi: 10.2522/ptj.20060378.

[52] M. Schenkman, R. A. Berger, P. O. Riley, R. W. Mann, e W. A. Hodge, «Whole-body movements during rising to standing from sitting», *Phys Ther*, vol. 70, n. 10, pagg. 638–651, ott. 1990, doi: 10.1093/ptj/70.10.638.

[53] C. A. Doorenbosch, J. Harlaar, M. E. Roebroeck, e G. J. Lankhorst, «Two strategies of transferring from sit-to-stand; the activation of monoarticular and biarticular muscles», *J Biomech*, vol. 27, n. 11, pagg. 1299–1307, nov. 1994, doi: 10.1016/0021-9290(94)90039-6.

[54] J. H. Carr e A. M. Gentile, «The effect of arm movement on the biomechanics of standing up», *Human Movement Science*, vol. 13, n. 2, pagg. 175–193, giu. 1994, doi: 10.1016/0167-9457(94)90035-3.

[55] P. J. Millington, B. M. Myklebust, e G. M. Shambes, «Biomechanical analysis of the sit-to-stand motion in elderly persons», *Archives of Physical Medicine and Rehabilitation*, vol. 73, n. 7, pagg. 609–617, lug. 1992, doi: 10.5555/uri:pii:000399939290124F.

[56] M. M. Gross, P. J. Stevenson, S. L. Charette, G. Pyka, e R. Marcus, «Effect of muscle strength and movement speed on the biomechanics of rising from a chair in healthy elderly and young women», *Gait Posture*, vol. 8, n. 3, pagg. 175–185, dic. 1998, doi: 10.1016/s0966-6362(98)00033-2.

[57] B. K. Kaya, D. E. Krebs, e P. O. Riley, «Dynamic stability in elders: momentum control in locomotor ADL», *J. Gerontol. A Biol. Sci. Med. Sci.*, vol. 53, n. 2, pagg. M126-134, mar. 1998, doi: 10.1093/gerona/53a.2.m126.

[58] S. Nuzik, R. Lamb, A. VanSant, e S. Hirt, «Sit-to-stand movement pattern. A kinematic study», *Phys Ther*, vol. 66, n. 11, pagg. 1708–1713, nov. 1986, doi: 10.1093/ptj/66.11.1708.

[59] E. R. Ikeda, M. L. Schenkman, P. O. Riley, e W. A. Hodge, «Influence of Age on Dynamics of Rising from a Chair», *Phys Ther*, vol. 71, n. 6, pagg. 473–481, giu. 1991, doi: 10.1093/ptj/71.6.473.

[60] K. Kerr, J. White, D. Barr, e R. Mollan, «Analysis of the sit-stand-sit movement cycle in normal subjects», *Clinical Biomechanics*, vol. 12, n. 4, pagg. 236–245, giu. 1997, doi: 10.1016/S0268-0033(96)00077-0.

[61] A. Kralj, R. J. Jaeger, e M. Munih, «Analysis of standing up and sitting down in humans: definitions and normative data presentation», *J Biomech*, vol. 23, n. 11, pagg. 1123–1138, 1990, doi: 10.1016/0021-9290(90)90005-n.

[62] M. E. Roebroeck, C. A. Doorenbosch, J. Harlaar, R. Jacobs, e G. J. Lankhorst, «Biomechanics and muscular activity during sit-to-stand transfer», *Clin Biomech (Bristol, Avon)*, vol. 9, n. 4, pagg. 235–244, lug. 1994, doi: 10.1016/0268-0033(94)90004-3.

[63] P. O. Riley, M. L. Schenkman, R. W. Mann, e W. A. Hodge, «Mechanics of a constrained chair-rise», *J Biomech*, vol. 24, n. 1, pagg. 77–85, 1991, doi: 10.1016/0021-9290(91)90328-k.

[64] D. M. Scarborough, C. A. McGibbon, e D. E. Krebs, «Chair rise strategies in older adults with functional limitations», *J Rehabil Res Dev*, vol. 44, n. 1, pagg. 33–42, 2007, doi: 10.1682/jrrd.2005.08.0134.

[65] N. B. Alexander, D. J. Koester, e J. A. Grunawalt, «Chair Design Affects How Older Adults Rise from a Chair», *Journal of the American Geriatrics Society*, vol. 44, n. 4, pagg. 356–362, 1996, doi: 10.1111/j.1532-5415.1996.tb06402.x.

[66] E. Papa e A. Cappozzo, «Sit-to-stand motor strategies investigated in able-bodied young and elderly subjects», *Journal of Biomechanics*, vol. 33, n. 9, pagg. 1113–1122, set. 2000, doi: 10.1016/S0021-9290(00)00046-4.

[67] D. Moxley Scarborough, D. E. Krebs, e B. A. Harris, «Quadriceps muscle strength and dynamic stability in elderly persons», *Gait Posture*, vol. 10, n. 1, pagg. 10–20, set. 1999, doi: 10.1016/s0966-6362(99)00018-1.

[68] P.-T. Cheng, M.-Y. Liaw, M.-K. Wong, F.-T. Tang, M.-Y. Lee, e P.-S. Lin, «The sit-to-stand movement in stroke patients and its correlation with falling», *Archives of Physical Medicine and Rehabilitation*, vol. 79, n. 9, pagg. 1043–1046, set. 1998, doi: 10.1016/S0003-9993(98)90168-X.

[69] G. Beckham, T. J. Suchomel, e S. Mizuguchi, «Force Plate Use in Performance Monitoring and Sport Science Testing», *New Studies in Athletics*, vol. 29, n. 3, pagg. 25–37, 2014.

[70] L. Janssens *et al.*, «Impaired Postural Control Reduces Sit-to-Stand-to-Sit Performance in Individuals with Chronic Obstructive Pulmonary Disease», *PLoS One*, vol. 9, n. 2, feb. 2014, doi: 10.1371/journal.pone.0088247.

[71] L. Piano, T. Geri, e M. Testa, «Raising and stabilization phase of the sit-to-stand movement better discriminate healthy elderly adults from young

subjects: a pilot cross-sectional study», *Arch Physiother*, vol. 10, apr. 2020, doi: 10.1186/s40945-020-00078-8.

[72] F. R. Goulart e J. Valls-Solé, «Patterned electromyographic activity in the sit-to-stand movement», *Clin Neurophysiol*, vol. 110, n. 9, pagg. 1634–1640, set. 1999, doi: 10.1016/s1388-2457(99)00109-1.

[73] W. G. Friedli, M. Hallett, e S. R. Simon, «Postural adjustments associated with rapid voluntary arm movements 1. Electromyographic data.», *J Neurol Neurosurg Psychiatry*, vol. 47, n. 6, pagg. 611–622, giu. 1984.

[74] P.-T. Cheng, C.-L. Chen, C.-M. Wang, e W.-H. Hong, «Leg muscle activation patterns of sit-to-stand movement in stroke patients», *Am J Phys Med Rehabil*, vol. 83, n. 1, pagg. 10–16, gen. 2004, doi: 10.1097/01.PHM.0000104665.34557.56.

[75] F. Chorin, C. Cornu, B. Beaune, J. Frère, e A. Rahmani, «Sit to stand in elderly fallers vs non-fallers: new insights from force platform and electromyography data», *Aging Clin Exp Res*, vol. 28, n. 5, pagg. 871–879, ott. 2016, doi: 10.1007/s40520-015-0486-1.

[76] R. Baker, «The history of gait analysis before the advent of modern computers», *Gait & Posture*, vol. 26, n. 3, pagg. 331–342, set. 2007, doi: 10.1016/j.gaitpost.2006.10.014.

[77] F. Sibella, M. Galli, M. Romei, A. Montesano, e M. Crivellini, «Biomechanical analysis of sit-to-stand movement in normal and obese subjects», *Clinical Biomechanics*, vol. 18, n. 8, pagg. 745–750, ott. 2003, doi: 10.1016/S0268-0033(03)00144-X.

[78] A. I. Cuesta-Vargas, A. Galán-Mercant, e J. M. Williams, «The use of inertial sensors system for human motion analysis», *Phys Ther Rev*, vol. 15, n. 6, pagg. 462–473, dic. 2010, doi: 10.1179/1743288X11Y.0000000006.

[79] W. G. M. Janssen, J. B. J. Bussmann, H. L. D. Horemans, e H. J. Stam, «Validity of accelerometry in assessing the duration of the sit-to-stand movement», *Med Biol Eng Comput*, vol. 46, n. 9, pagg. 879–887, set. 2008, doi: 10.1007/s11517-008-0366-3.

[80] Rong Zhu e Zhaoying Zhou, «A real-time articulated human motion tracking using tri-axis inertial/magnetic sensors package», *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 12, n. 2, pagg. 295–302, giu. 2004, doi: 10.1109/TNSRE.2004.827825.

[81] N. Yazdi, F. Ayazi, e K. Najafi, «Micromachined inertial sensors», *Proceedings of the IEEE*, vol. 86, n. 8, pagg. 1640–1659, ago. 1998, doi: 10.1109/5.704269.

[82] V. M. N. Passaro, A. Cuccovillo, L. Vaiani, M. De Carlo, e C. E. Campanella, «Gyroscope Technology and Applications: A Review in the Industrial Perspective», *Sensors (Basel)*, vol. 17, n. 10, ott. 2017, doi: 10.3390/s17102284.

[83] M. Kok, J. D. Hol, e T. B. Schön, *Using Inertial Sensors for Position and Orientation Estimation*. now, 2017. Consultato: set. 06, 2020. [Online]. Disponibile su: https://ieeexplore.ieee.org/document/8187588

[84] F. Gulmammadov, «Analysis, modeling and compensation of bias drift in MEMS inertial sensors», in *2009 4th International Conference on Recent Advances in Space Technologies*, giu. 2009, pagg. 591–596. doi: 10.1109/RAST.2009.5158260.

[85] D. Titterton, J. L. Weston, e J. Weston, *Strapdown Inertial Navigation Technology*. IET, 2004.

[86] S. Nassar e N. El-Sheimy, «Wavelet Analysis For Improving INS and INS/DGPS Navigation Accuracy», *The Journal of Navigation*, vol. 58, n. 1, pagg. 119–134, gen. 2005, doi: 10.1017/S0373463304003005.
[87] W. T. Latt, K. C. Veluvolu, e W. T. Ang, «Drift-Free Position Estimation of Periodic or Quasi-Periodic Motion Using Inertial Sensors», *Sensors*, vol. 11, n. 6, Art. n. 6, giu. 2011, doi: 10.3390/s110605931.
[88] Y. K. Thong, M. S. Woolfson, J. A. Crowe, B. R. Hayes-Gill, e D. A. Jones, «Numerical double integration of acceleration measurements in noise», *Measurement*, vol. 36, n. 1, pagg. 73–92, lug. 2004, doi: 10.1016/j.measurement.2004.04.005.
[89] D. Won, J. Ahn, S. Sung, M. Heo, S.-H. Im, e Y. J. Lee, «Performance Improvement of Inertial Navigation System by Using Magnetometer with Vehicle Dynamic Constraints», *Journal of Sensors*, ago. 10, 2015. https://www.hindawi.com/journals/js/2015/435062/ (consultato set. 07, 2020).
[90] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, e V. Kasi, «Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications», vol. 1, n. 2, pag. 7, 2013.
[91] M. Kok e T. B. Schön, «Magnetometer Calibration Using Inertial Sensors», *IEEE Sensors Journal*, vol. 16, n. 14, pagg. 5679–5689, lug. 2016, doi: 10.1109/JSEN.2016.2569160.
[92] «Status of the MEMS Industry 2020», *i-Micronews*. https://www.i-micronews.com/products/status-of-the-mems-industry-2020/ (consultato set. 09, 2020).
[93] «MEMS Market Research Report: Market size, Industry outlook, Market Forecast, Demand Analysis,Market Share, Market Report 2019-2025». https://www.industryarc.com/Report/15575/mems-market.html (consultato set. 09, 2020).
[94] M. A. Brodie *et al.*, «Big data vs accurate data in health research: Large-scale physical activity monitoring, smartphones, wearable devices and risk of unconscious bias», *Medical Hypotheses*, vol. 119, pagg. 32–36, ott. 2018, doi: 10.1016/j.mehy.2018.07.015.
[95] H. Bragança, J. G. Colonna, W. S. Lima, e E. Souto, «A Smartphone Lightweight Method for Human Activity Recognition Based on Information Theory», *Sensors (Basel)*, vol. 20, n. 7, mar. 2020, doi: 10.3390/s20071856.
[96] Y.-T. Chang, «Physical Activity and Cognitive Function in Mild Cognitive Impairment», *ASN Neuro*, vol. 12, pag. 1759091419901182, dic. 2020, doi: 10.1177/1759091419901182.
[97] A. G. Kubala, B. Barone Gibbs, D. J. Buysse, S. R. Patel, M. H. Hall, e C. E. Kline, «Field-based Measurement of Sleep: Agreement between Six Commercial Activity Monitors and a Validated Accelerometer», *Behav Sleep Med*, vol. 18, n. 5, pagg. 637–652, ott. 2020, doi: 10.1080/15402002.2019.1651316.
[98] J. Xie, D. Wen, L. Liang, Y. Jia, L. Gao, e J. Lei, «Evaluating the Validity of Current Mainstream Wearable Devices in Fitness Tracking Under Various Physical Activities: Comparative Study», *JMIR Mhealth Uhealth*, vol. 6, n. 4, pag. e94, apr. 2018, doi: 10.2196/mhealth.9754.
[99] S. Fudickar, S. Hellmers, S. Lau, R. Diekmann, J. M. Bauer, e A. Hein, «Measurement System for Unsupervised Standardized Assessment of Timed "Up & Go" and Five Times Sit to Stand Test in the Community—A Validity Study», *Sensors (Basel)*, vol. 20, n. 10, mag. 2020, doi: 10.3390/s20102824.

[100] M. Vassallo, L. Poynter, J. C. Sharma, J. Kwan, e S. C. Allen, «Fall risk-assessment tools compared with clinical judgment: an evaluation in a rehabilitation ward», *Age Ageing*, vol. 37, n. 3, pagg. 277–281, mag. 2008, doi: 10.1093/ageing/afn062.

[101] M. Vassallo, R. Stockdale, J. C. Sharma, R. Briggs, e S. Allen, «A comparative study of the use of four fall risk assessment tools on acute medical wards», *J Am Geriatr Soc*, vol. 53, n. 6, pagg. 1034–1038, giu. 2005, doi: 10.1111/j.1532-5415.2005.53316.x.

[102] «WHO | International Classification of Functioning, Disability and Health (ICF)», *WHO*. http://www.who.int/classifications/icf/en/ (consultato set. 10, 2020).

[103] R. C. van Lummel *et al.*, «Physical Performance and Physical Activity in Older Adults: Associated but Separate Domains of Physical Function in Old Age», *PLoS ONE*, vol. 10, n. 12, pag. e0144048, 2015, doi: 10.1371/journal.pone.0144048.

[104] J. Hellec, F. Chorin, A. Castagnetti, e S. S. Colson, «Sit-To-Stand Movement Evaluated Using an Inertial Measurement Unit Embedded in Smart Glasses—A Validation Study», *Sensors*, vol. 20, n. 18, Art. n. 18, gen. 2020, doi: 10.3390/s20185019.

[105] M. H. Tran, P. Kmecl, Y. Regmi, B. Dai, M. Gorsic, e D. Novak, «Toward real-world evaluations of trunk exoskeletons using inertial measurement units», *IEEE Int Conf Rehabil Robot*, vol. 2019, pagg. 483–487, 2019, doi: 10.1109/ICORR.2019.8779517.

[106] M. C. Boonstra, R. M. A. van der Slikke, N. L. W. Keijsers, R. C. van Lummel, M. C. de Waal Malefijt, e N. Verdonschot, «The accuracy of measuring the kinematics of rising from a chair with accelerometers and gyroscopes», *J Biomech*, vol. 39, n. 2, pagg. 354–358, 2006, doi: 10.1016/j.jbiomech.2004.11.021.

[107] S. a. a. N. Bolink *et al.*, «Validity of an inertial measurement unit to assess pelvic orientation angles during gait, sit-stand transfers and step-up transfers: Comparison with an optoelectronic motion capture system», *Med Eng Phys*, vol. 38, n. 3, pagg. 225–231, mar. 2016, doi: 10.1016/j.medengphy.2015.11.009.

[108] R. C. Van Lummel *et al.*, «Automated approach for quantifying the repeated sit-to-stand using one body fixed sensor in young and older adults», *Gait Posture*, vol. 38, n. 1, pagg. 153–156, mag. 2013, doi: 10.1016/j.gaitpost.2012.10.008.

[109] R. Zarzeczny *et al.*, «Aging effect on the instrumented Timed-Up-and-Go test variables in nursing home women aged 80–93 years», *Biogerontology*, vol. 18, n. 4, pagg. 651–663, 2017, doi: 10.1007/s10522-017-9717-5.

[110] T. Pozaic, U. Lindemann, A.-K. Grebe, e W. Stork, «Sit-to-Stand Transition Reveals Acute Fall Risk in Activities of Daily Living», *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 4, 2016, doi: 10.1109/JTEHM.2016.2620177.

[111] L. J. Tulipani, B. Meyer, D. Larie, A. J. Solomon, e R. S. McGinnis, «Metrics extracted from a single wearable sensor during sit-stand transitions relate to mobility impairment and fall risk in people with multiple sclerosis», *Gait & Posture*, vol. 80, pagg. 361–366, lug. 2020, doi: 10.1016/j.gaitpost.2020.06.014.

[112] L. Montesinos, R. Castaldo, e L. Pecchia, «Wearable Inertial Sensors for Fall Risk Assessment and Prediction in Older Adults: A Systematic Review

and Meta-Analysis», *IEEE Trans Neural Syst Rehabil Eng*, vol. 26, n. 3, pagg. 573–582, 2018, doi: 10.1109/TNSRE.2017.2771383.

[113] A. Ejupi, M. Brodie, S. R. Lord, J. Annegarn, S. J. Redmond, e K. Delbaere, «Wavelet-Based Sit-To-Stand Detection and Assessment of Fall Risk in Older People Using a Wearable Pendant Device», *IEEE Transactions on Biomedical Engineering*, vol. 64, n. 7, pagg. 1602–1607, lug. 2017, doi: 10.1109/TBME.2016.2614230.

[114] E. P. Doheny *et al.*, «Falls classification using tri-axial accelerometers during the five-times-sit-to-stand test», *Gait Posture*, vol. 38, n. 4, pagg. 1021–1025, set. 2013, doi: 10.1016/j.gaitpost.2013.05.013.

[115] S. Parvaneh, J. Mohler, N. Toosizadeh, G. S. Grewal, e B. Najafi, «Postural Transitions during Activities of Daily Living Could Identify Frailty Status: Application of Wearable Technology to Identify Frailty during Unsupervised Condition», *Gerontology*, vol. 63, n. 5, pagg. 479–487, 2017, doi: 10.1159/000460292.

[116] N. Millor, P. Lecumberri, M. Gómez, A. Martínez-Ramírez, e M. Izquierdo, «An evaluation of the 30-s chair stand test in older adults: frailty detection based on kinematic parameters from a single inertial unit», *J Neuroeng Rehabil*, vol. 10, pag. 86, ago. 2013, doi: 10.1186/1743-0003-10-86.

[117] Z. Hussain, Q. Z. Sheng, e W. Zhang, «Different Approaches for Human Activity Recognition: A Survey», *ArXiv*, 2019.

[118] F. Demrozi, G. Pravadelli, A. Bihorac, e P. Rashidi, «Human Activity Recognition using Inertial, Physiological and Environmental Sensors: a Comprehensive Survey», *arXiv:2004.08821 [cs, eess]*, apr. 2020, Consultato: set. 16, 2020. [Online]. Disponibile su: http://arxiv.org/abs/2004.08821

[119] M. Vrigkas, C. Nikou, e I. A. Kakadiaris, «A Review of Human Activity Recognition Methods», *Front. Robot. AI*, vol. 2, 2015, doi: 10.3389/frobt.2015.00028.

[120] F. Attal, S. Mohammed, M. Dedabrishvili, F. Chamroukhi, L. Oukhellou, e Y. Amirat, «Physical Human Activity Recognition Using Wearable Sensors», *Sensors (Basel)*, vol. 15, n. 12, pagg. 31314–31338, dic. 2015, doi: 10.3390/s151229858.

[121] D. Hendry, K. Chai, A. Campbell, L. Hopper, P. O'Sullivan, e L. Straker, «Development of a Human Activity Recognition System for Ballet Tasks», *Sports Medicine - Open*, vol. 6, n. 1, pag. 10, feb. 2020, doi: 10.1186/s40798-020-0237-5.

[122] U. Martinez-Hernandez e A. A. Dehghani-Sanij, «Probabilistic identification of sit-to-stand and stand-to-sit with a wearable sensor», *Pattern Recognition Letters*, vol. 118, pagg. 32–41, feb. 2019, doi: 10.1016/j.patrec.2018.03.020.

[123] Q. Ni *et al.*, «Leveraging Wearable Sensors for Human Daily Activity Recognition with Stacked Denoising Autoencoders», *Sensors*, vol. 20, n. 18, Art. n. 18, gen. 2020, doi: 10.3390/s20185114.

[124] J.-H. Li, L. Tian, H. Wang, Y. An, K. Wang, e L. Yu, «Segmentation and Recognition of Basic and Transitional Activities for Continuous Physical Human Activity», *IEEE Access*, vol. 7, pagg. 42565–42576, 2019, doi: 10.1109/ACCESS.2019.2905575.

[125] K. Safi, S. Mohammed, F. Attal, M. Khalil, e Y. Amirat, «Recognition of different daily living activities using hidden Markov model regression», in

2016 3rd Middle East Conference on Biomedical Engineering (MECBME), ott. 2016, pagg. 16–19. doi: 10.1109/MECBME.2016.7745398.

[126] A. Doulah, X. Shen, e E. Sazonov, «A method for early detection of the initiation of sit-to-stand posture transitions», *Physiol. Meas.*, vol. 37, n. 4, pagg. 515–529, apr. 2016, doi: 10.1088/0967-3334/37/4/515.

[127] A. Doulah, X. Shen, e E. Sazonov, «Early Detection of the Initiation of Sit-to-Stand Posture Transitions Using Orthosis-Mounted Sensors», *Sensors (Basel)*, vol. 17, n. 12, nov. 2017, doi: 10.3390/s17122712.

[128] F. Masse, R. Gonzenbach, A. Paraschiv-Ionescu, A. R. Luft, e K. Aminian, «Wearable barometric pressure sensor to improve postural transition recognition of mobility-impaired stroke patients», *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 24, n. 11, Art. n. 11, nov. 2016, doi: 10.1109/TNSRE.2016.2532844.

[129] A. Atrsaei *et al.*, «Postural transitions detection and characterization in healthy and patient populations using a single waist sensor», *J Neuroeng Rehabil*, vol. 17, n. 1, pag. 70, 03 2020, doi: 10.1186/s12984-020-00692-4.

[130] N. Milad e R. Hossein, «Detection of daily postures and walking modalities using a single chest-mounted tri-axial accelerometer», *Medical Engineering and Physics*, vol. 57, pagg. 75–81, 2018.

[131] J. KERR, J. CARLSON, S. GODBOLE, L. CADMUS-BERTRAM, J. BELLETTIERE, e S. HARTMAN, «Improving Hip-Worn Accelerometer Estimates of Sitting Using Machine Learning Methods», *Med Sci Sports Exerc*, vol. 50, n. 7, pagg. 1518–1524, lug. 2018, doi: 10.1249/MSS.0000000000001578.

[132] S. Hemmati e E. Wade, «Detecting postural transitions: A robust wavelet-based approach», in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, ago. 2016, pagg. 3704–3707. doi: 10.1109/EMBC.2016.7591532.

[133] I. de Kok, J. Hough, F. Hülsmann, M. Botsch, D. Schlangen, e S. Kopp, «A Multimodal System for Real-Time Action Instruction in Motor Skill Learning», in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, New York, NY, USA, nov. 2015, pagg. 355–362. doi: 10.1145/2818346.2820746.

[134] A. Khan *et al.*, «Beyond activity recognition: skill assessment from accelerometer data», in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, New York, NY, USA, set. 2015, pagg. 1155–1166. doi: 10.1145/2750858.2807534.

[135] T. Mitchell, *Machine learning*. New York, NY: McGraw-Hill Education, 2019.

[136] Z. Ghahramani, «Unsupervised Learning», in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, e G. Rätsch, A c. di Berlin, Heidelberg: Springer, 2004, pagg. 72–112. doi: 10.1007/978-3-540-28650-9_5.

[137] Q. Liu e Y. Wu, «Supervised Learning», in *Encyclopedia of the Sciences of Learning*, N. M. Seel, A c. di Boston, MA: Springer US, 2012, pagg. 3243–3245. doi: 10.1007/978-1-4419-1428-6_451.

[138] I. Cleland *et al.*, «Optimal placement of accelerometers for the detection of everyday activities», *Sensors (Basel)*, vol. 13, n. 7, pagg. 9183–9200, lug. 2013, doi: 10.3390/s130709183.

[139] L. Atallah, B. Lo, R. King, e G.-Z. Yang, «Sensor Positioning for Activity Recognition Using Wearable Accelerometers», *IEEE Transactions on Biomedical Circuits and Systems*, vol. 5, n. 4, pagg. 320–329, ago. 2011, doi: 10.1109/TBCAS.2011.2160540.

[140] J. M. Bland e D. G. Altman, «Statistical methods for assessing agreement between two methods of clinical measurement», *Lancet*, vol. 1, n. 8476, pagg. 307–310, feb. 1986.

[141] A. C. Davison e D. V. Hinkley, *Bootstrap Methods and their Application*, 1 edizione. Cambridge University Press, 1997.

[142] K. Jung, J. Lee, V. Gupta, e G. Cho, «Comparison of Bootstrap Confidence Interval Methods for GSCA Using a Monte Carlo Simulation», *Front Psychol*, vol. 10, ott. 2019, doi: 10.3389/fpsyg.2019.02215.

[143] D. G. Altman, *Practical Statistics for Medical Research*. CRC Press, 1990.

[144] «Chapter 6: Choosing effect measures and computing estimates of effect | Cochrane Training». https://training.cochrane.org/handbook/current/chapter-06 (consultato lug. 20, 2020).

[145] «Bland-Altman and Correlation Plot». https://it.mathworks.com/matlabcentral/fileexchange/45049-bland-altman-and-correlation-plot (consultato lug. 05, 2020).

[146] F. Schoonjans, «MedCalc's Comparison of means calculator», *MedCalc*. https://www.medcalc.org/calc/comparison_of_means.php (consultato lug. 05, 2020).

[147] B. Najafi, K. Aminian, A. Paraschiv-Ionescu, F. Loew, C. J. Büla, e P. Robert, «Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly», *IEEE Trans Biomed Eng*, vol. 50, n. 6, pagg. 711–723, giu. 2003, doi: 10.1109/TBME.2003.812189.

[148] A. Godfrey, A. K. Bourke, G. M. Olaighin, P. van de Ven, e J. Nelson, «Activity classification using a single chest mounted tri-axial accelerometer», *Med Eng Phys*, vol. 33, n. 9, pagg. 1127–1135, nov. 2011, doi: 10.1016/j.medengphy.2011.05.002.

[149] A. Salarian, H. Russmann, F. J. G. Vingerhoets, P. R. Burkhard, e K. Aminian, «Ambulatory monitoring of physical activities in patients with Parkinson's disease», *IEEE Trans Biomed Eng*, vol. 54, n. 12, pagg. 2296–2299, dic. 2007, doi: 10.1109/tbme.2007.896591.

[150] M. H. Pham *et al.*, «Validation of a Lower Back "Wearable"-Based Sit-to-Stand and Stand-to-Sit Algorithm for Patients With Parkinson's Disease and Older Adults in a Home-Like Environment», *Front. Neurol.*, vol. 9, 2018, doi: 10.3389/fneur.2018.00652.

[151] D. Rodríguez-Martín, A. Samà, C. Pérez-López, e A. Català, «Identification of sit-to-stand and stand-to-sit transitions using a single inertial sensor», *Stud Health Technol Inform*, vol. 177, pagg. 113–117, 2012.

[152] M. Kristiansen, G. H. F. Rasmussen, M. E. Sloth, e M. Voigt, «Inter- and intra-individual variability in the kinematics of the back squat», *Human Movement Science*, vol. 67, pag. 102510, ott. 2019, doi: 10.1016/j.humov.2019.102510.

[153] T. Li *et al.*, «Automatic Timed Up-and-Go Sub-Task Segmentation for Parkinson's Disease Patients Using Video-Based Activity Classification», *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, n. 11, pagg. 2189–2199, nov. 2018, doi: 10.1109/TNSRE.2018.2875738.

[154] B. R. Greene, E. P. Doheny, R. A. Kenny, e B. Caulfield, «Classification of frailty and falls history using a combination of sensor-based mobility assessments», *Physiol Meas*, vol. 35, n. 10, pagg. 2053–2066, ott. 2014, doi: 10.1088/0967-3334/35/10/2053.

[155] A. Ng, «Structuring Machine Learning Projects», 2017. Consultato: mar. 10, 2021. [Online]. Disponibile su: https://www.coursera.org/learn/machine-learning-projects?specialization=deep-learning

[156] W. S. McCulloch e W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *Bulletin of Mathematical Biophysics*, vol. 5, n. 4, pagg. 115–133, dic. 1943, doi: 10.1007/BF02478259.

[157] C. Van Der Malsburg, «Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms», in *Brain Theory*, Berlin, Heidelberg, 1986, pagg. 245–248. doi: 10.1007/978-3-642-70911-1_20.

[158] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[159] Y. C. (Ph D.) e D. E. Rumelhart, *Backpropagation: Theory, Architectures, and Applications*. Psychology Press, 1995.

[160] M. Nielsen, «Neural Networks and Deep Learning», pag. 224.

[161] Y. LeCun, Y. Bengio, e G. Hinton, «Deep learning», *Nature*, vol. 521, n. 7553, Art. n. 7553, mag. 2015, doi: 10.1038/nature14539.

[162] «AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?», set. 01, 2020. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks (consultato ott. 04, 2020).

[163] T. V. P. Bliss e G. L. Collingridge, «A synaptic model of memory: long-term potentiation in the hippocampus», *Nature*, vol. 361, n. 6407, Art. n. 6407, gen. 1993, doi: 10.1038/361031a0.

[164] J. Tang *et al.*, «Bridging Biological and Artificial Neural Networks with Emerging Neuromorphic Devices: Fundamentals, Progress, and Challenges», *Advanced Materials*, vol. 31, n. 49, pag. 1902761, 2019, doi: 10.1002/adma.201902761.

[165] D. Purves, K. S. LaBar, M. L. Platt, M. Woldorff, R. Cabeza, e S. A. Huettel, *Principles of Cognitive Neuroscience*, Second Edition. Oxford, New York: Oxford University Press, 2012.

[166] R. S. Zucker e W. G. Regehr, «Short-Term Synaptic Plasticity», *Annu. Rev. Physiol.*, vol. 64, n. 1, pagg. 355–405, mar. 2002, doi: 10.1146/annurev.physiol.64.092501.114547.

[167] R. C. Malenka e M. F. Bear, «LTP and LTD: An Embarrassment of Riches», *Neuron*, vol. 44, n. 1, pagg. 5–21, set. 2004, doi: 10.1016/j.neuron.2004.09.012.

[168] T. V. Bliss e T. Lomo, «Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path», *The Journal of Physiology*, vol. 232, n. 2, pagg. 331–356, lug. 1973, doi: 10.1113/jphysiol.1973.sp010273.

[169] G. Bi e M. Poo, «Synaptic modification by correlated activity: Hebb's postulate revisited», *Annual Review of Neuroscience*, vol. 24, pagg. 139–166, 2001, doi: 10.1146/annurev.neuro.24.1.139.

[170] M. L. Blanke e A. M. J. VanDongen, *Activation Mechanisms of the NMDA Receptor*. CRC Press/Taylor & Francis, 2009. Consultato: feb. 10, 2021. [Online]. Disponibile su: https://www.ncbi.nlm.nih.gov/books/NBK5274/

[171] S. Ben Achour e O. Pascual, «Glia: The many ways to modulate synaptic plasticity», *Neurochemistry International*, vol. 57, n. 4, pagg. 440–445, nov. 2010, doi: 10.1016/j.neuint.2010.02.013.

[172] R. Rojas, «The Backpropagation Algorithm», in *Neural Networks*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pagg. 149–182. doi: 10.1007/978-3-642-61068-4_7.

[173] M. F. Møller, «A scaled conjugate gradient algorithm for fast supervised learning», *Neural Networks*, vol. 6, n. 4, pagg. 525–533, gen. 1993, doi: 10.1016/S0893-6080(05)80056-5.

[174] J. Nocedal e S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[175] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, e G. Bing, «Learning from class-imbalanced data: Review of methods and applications», *Expert Systems with Applications*, vol. 73, pagg. 220–239, mag. 2017, doi: 10.1016/j.eswa.2016.12.035.

[176] R. Khusainov, D. Azzi, I. E. Achumba, e S. D. Bersch, «Real-Time Human Ambulation, Activity, and Physiological Monitoring: Taxonomy of Issues, Techniques, Applications, Challenges and Limitations», *Sensors (Basel)*, vol. 13, n. 10, pagg. 12852–12902, set. 2013, doi: 10.3390/s131012852.

[177] H. Jagos, J. Oberzaucher, M. Reichel, W. L. Zagler, e W. Hlauschek, «A multimodal approach for insole motion measurement and analysis», *Procedia Engineering*, vol. 2, n. 2, pagg. 3103–3108, giu. 2010, doi: 10.1016/j.proeng.2010.04.118.

[178] G. Plasqui, A. G. Bonomi, e K. R. Westerterp, «Daily physical activity assessment with accelerometers: new insights and validation studies», *Obes Rev*, vol. 14, n. 6, pagg. 451–462, giu. 2013, doi: 10.1111/obr.12021.

[179] L. L. Beranek e T. J. Mellow, «Chapter 1 - Introduction and terminology», in *Acoustics: Sound Fields and Transducers*, L. L. Beranek e T. J. Mellow, A c. di Academic Press, 2012, pagg. 1–19. doi: 10.1016/B978-0-12-391421-7.00001-4.

[180] J. Heaton, *Introduction to Neural Networks for Java, 2Nd Edition*, 2nd ed. Heaton Research, Inc., 2008.

[181] G. C. Cawley e N. L. C. Talbot, «On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation», *J. Mach. Learn. Res.*, vol. 11, pagg. 2079–2107, 2010.

[182] D. Baumann e K. Baumann, «Reliable estimation of prediction errors for QSAR models under model uncertainty using double cross-validation», *Journal of Cheminformatics*, vol. 6, n. 1, pag. 47, nov. 2014, doi: 10.1186/s13321-014-0047-1.

[183] S. Varma e R. Simon, «Bias in error estimation when using cross-validation for model selection», *BMC Bioinformatics*, vol. 7, n. 1, pag. 91, feb. 2006, doi: 10.1186/1471-2105-7-91.

[184] C. C. Yang e Y. L. Hsu, «Development of a wearable motion detector for telemonitoring and real-time identification of physical activity», *Telemed J E Health*, vol. 15, n. 1, pagg. 62–72, gen. 2009, doi: 10.1089/tmj.2008.0060.

[185] T. Banerjee, J. M. Keller, M. Skubic, e C. Abbott, «Sit-to-stand detection using fuzzy clustering techniques», in *International Conference on Fuzzy Systems*, lug. 2010, pagg. 1–8. doi: 10.1109/FUZZY.2010.5584843.

[186] B. Esmael, A. Arnaout, R. K. Fruhwirth, e G. Thonhauser, «Improving time series classification using Hidden Markov Models», in *2012 12th*

International Conference on Hybrid Intelligent Systems (HIS), dic. 2012, pagg. 502–507. doi: 10.1109/HIS.2012.6421385.

[187] F. Karim, S. Majumdar, H. Darabi, e S. Chen, «LSTM Fully Convolutional Networks for Time Series Classification», *IEEE Access*, vol. 6, pagg. 1662–1669, 2018, doi: 10.1109/ACCESS.2017.2779939.

[188] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, e P.-A. Muller, «Deep learning for time series classification: a review», *Data Min Knowl Disc*, vol. 33, n. 4, pagg. 917–963, lug. 2019, doi: 10.1007/s10618-019-00619-1.

[189] A. Ng, «Recurrent Networks: Why Sequence Models?», 2017. Consultato: dic. 16, 2020. [Online]. Disponibile su: https://www.coursera.org/learn/nlp-sequence-models

[190] T. G. Dietterich, «Machine Learning for Sequential Data: A Review», in *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 2396, T. Caelli, A. Amin, R. P. W. Duin, D. de Ridder, e M. Kamel, A c. di Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pagg. 15–30. doi: 10.1007/3-540-70659-3_2.

[191] G. Ritschard, «Exploring Sequential Data», in *Discovery Science*, Berlin, Heidelberg, 2012, pagg. 3–6. doi: 10.1007/978-3-642-33492-4_3.

[192] P. Murugan, «Learning The Sequential Temporal Information with Recurrent Neural Networks», *arXiv:1807.02857 [cs, stat]*, lug. 2018, Consultato: dic. 21, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1807.02857

[193] R. DiPietro e G. D. Hager, «Deep learning: RNNs and LSTM», *Handbook of Medical Image Computing and Computer Assisted Intervention*, pagg. 503–519, gen. 2019, doi: 10.1016/B978-0-12-816176-0.00026-0.

[194] M. Sundermeyer, H. Ney, e R. Schlüter, «From feedforward to recurrent LSTM neural networks for language modeling», *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 23, n. 3, pagg. 517–529, mar. 2015, doi: 10.1109/TASLP.2015.2400218.

[195] J. L. Elman, «Finding Structure in Time», *Cognitive Science*, vol. 14, n. 2, pagg. 179–211, 1990, doi: https://doi.org/10.1207/s15516709cog1402_1.

[196] IBM Cloud Education, «What are Recurrent Neural Networks?», *ibm.com*, set. 14, 2020. https://www.ibm.com/cloud/learn/recurrent-neural-networks (consultato dic. 22, 2020).

[197] S. A. Marhon, C. J. F. Cameron, e S. C. Kremer, «Recurrent Neural Networks», in *Handbook on Neural Information Processing*, M. Bianchini, M. Maggini, e L. C. Jain, A c. di Berlin, Heidelberg: Springer, 2013, pagg. 29–65. doi: 10.1007/978-3-642-36657-4_2.

[198] R. Khaldi, R. Chiheb, e A. El Afia, «Feedforward and Recurrent Neural Networks for Time Series Forecasting: Comparative Study», in *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications*, New York, NY, USA, mag. 2018, pagg. 1–6. doi: 10.1145/3230905.3230946.

[199] Y. Bengio, P. Simard, e P. Frasconi, «Learning long-term dependencies with gradient descent is difficult», *IEEE Transactions on Neural Networks*, vol. 5, n. 2, pagg. 157–166, mar. 1994, doi: 10.1109/72.279181.

[200] R. Pascanu, T. Mikolov, e Y. Bengio, «On the difficulty of training Recurrent Neural Networks», *arXiv:1211.5063 [cs]*, feb. 2013, Consultato: dic. 26, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1211.5063

[201] M. Henaff, A. Szlam, e Y. LeCun, «Recurrent Orthogonal Networks and Long-Memory Tasks», *arXiv:1602.06662 [cs, stat]*, mar. 2017, Consultato: dic. 27, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1602.06662

[202] M. Arjovsky, A. Shah, e Y. Bengio, «Unitary Evolution Recurrent Neural Networks», *arXiv:1511.06464 [cs, stat]*, mag. 2016, Consultato: dic. 27, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1511.06464

[203] Tsungnan Lin, B. G. Horne, P. Tino, e C. L. Giles, «Learning long-term dependencies in NARX recurrent neural networks», *IEEE Transactions on Neural Networks*, vol. 7, n. 6, pagg. 1329–1338, nov. 1996, doi: 10.1109/72.548162.

[204] S. Hochreiter e J. Schmidhuber, «Long Short-Term Memory», *Neural Computation*, vol. 9, n. 8, pagg. 1735–1780, nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[205] N. Arbel, «How LSTM networks solve the problem of vanishing gradients», *Medium*, 2018. https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577 (consultato dic. 28, 2020).

[206] F. Gers, «Long Short-Term Memory in Recurrent Neural Networks», mag. 2001, doi: 10.5075/epfl-thesis-2366.

[207] S. Bai, J. Z. Kolter, e V. Koltun, «An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling», *arXiv:1803.01271 [cs]*, apr. 2018, Consultato: dic. 28, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1803.01271

[208] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, e K. Kavukcuoglu, «Neural Machine Translation in Linear Time», *arXiv:1610.10099 [cs]*, mar. 2017, Consultato: dic. 29, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1610.10099

[209] Y. N. Dauphin, A. Fan, M. Auli, e D. Grangier, «Language Modeling with Gated Convolutional Networks», *arXiv:1612.08083 [cs]*, set. 2017, Consultato: dic. 29, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1612.08083

[210] A. van den Oord *et al.*, «WaveNet: A Generative Model for Raw Audio», *arXiv:1609.03499 [cs]*, set. 2016, Consultato: dic. 29, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1609.03499

[211] J. Gehring, M. Auli, D. Grangier, e Y. N. Dauphin, «A Convolutional Encoder Model for Neural Machine Translation», *arXiv:1611.02344 [cs]*, lug. 2017, Consultato: dic. 29, 2020. [Online]. Disponibile su: http://arxiv.org/abs/1611.02344

[212] I. Godfellow, Y. Bengio, e A. Courville, *Deep Learning*. The MIT Press, 2016. Consultato: dic. 28, 2020. [Online]. Disponibile su: https://mitpress.mit.edu/books/deep-learning

[213] C. A. Ronao e S.-B. Cho, «Deep Convolutional Neural Networks for Human Activity Recognition with Smartphone Sensors», in *Neural Information Processing*, Cham, 2015, pagg. 46–53. doi: 10.1007/978-3-319-26561-2_6.

[214] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, e G. D. Hager, «Temporal Convolutional Networks for Action Segmentation and Detection», *arXiv:1611.05267 [cs]*, nov. 2016, Consultato: gen. 01, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1611.05267

[215] J. Long, E. Shelhamer, e T. Darrell, «Fully Convolutional Networks for Semantic Segmentation», *arXiv:1411.4038 [cs]*, mar. 2015, Consultato: gen. 02, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1411.4038

[216] F. Yu e V. Koltun, «Multi-Scale Context Aggregation by Dilated Convolutions», *arXiv:1511.07122 [cs]*, apr. 2016, Consultato: gen. 02, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1511.07122

[217] K. He, X. Zhang, S. Ren, e J. Sun, «Deep Residual Learning for Image Recognition», *arXiv:1512.03385 [cs]*, dic. 2015, Consultato: gen. 02, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1512.03385

[218] V. Nair e G. E. Hinton, «Rectified linear units improve restricted boltzmann machines», in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Madison, WI, USA, giu. 2010, pagg. 807–814.

[219] J. Wang, Y. Chen, S. Hao, X. Peng, e L. Hu, «Deep Learning for Sensor-based Activity Recognition: A Survey», *Pattern Recognition Letters*, vol. 119, pagg. 3–11, mar. 2019, doi: 10.1016/j.patrec.2018.02.010.

[220] Q. Yang, *Activity Recognition: Linking Low-level Sensors to High-level Intelligence*. 2009, pag. 25.

[221] J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, e S. Krishnaswamy, «Deep convolutional neural networks on multichannel time series for human activity recognition», in *Proceedings of the 24th International Conference on Artificial Intelligence*, Buenos Aires, Argentina, lug. 2015, pagg. 3995–4001.

[222] N. Y. Hammerla, S. Halloran, e T. Ploetz, «Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables», *arXiv:1604.08880 [cs, stat]*, apr. 2016, Consultato: gen. 05, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1604.08880

[223] Y. Tang, Q. Teng, L. Zhang, F. Min, e J. He, «Layer-Wise Training Convolutional Neural Networks With Smaller Filters for Human Activity Recognition Using Wearable Sensors», *IEEE Sensors Journal*, vol. 21, n. 1, pagg. 581–592, gen. 2021, doi: 10.1109/JSEN.2020.3015521.

[224] A. Ignatov, «Real-time human activity recognition from accelerometer data using Convolutional Neural Networks», *Applied Soft Computing*, vol. 62, pagg. 915–922, gen. 2018, doi: 10.1016/j.asoc.2017.09.027.

[225] D. Ortega Anderez, A. Lotfi, e A. Pourabdollah, «A deep learning based wearable system for food and drink intake recognition», *Journal of Ambient Intelligence and Humanized Computing*, nov. 2020, doi: 10.1007/s12652-020-02684-7.

[226] F. Luo, S. Poslad, e E. Bodanese, «Temporal Convolutional Networks for Multiperson Activity Recognition Using a 2-D LIDAR», *IEEE Internet of Things Journal*, vol. 7, n. 8, pagg. 7432–7442, ago. 2020, doi: 10.1109/JIOT.2020.2984544.

[227] N. Nair, C. Thomas, e D. B. Jayagopi, «Human Activity Recognition Using Temporal Convolutional Network», in *Proceedings of the 5th international Workshop on Sensor-based Activity Recognition and Interaction*, New York, NY, USA, set. 2018, pagg. 1–8. doi: 10.1145/3266157.3266221.

[228] W. Ahmad, B. M. Kazmi, e H. Ali, «Human Activity Recognition using Multi-Head CNN followed by LSTM», in *2019 15th International Conference on Emerging Technologies (ICET)*, dic. 2019, pagg. 1–6. doi: 10.1109/ICET48972.2019.8994412.

[229] R. Mutegeki e D. S. Han, «A CNN-LSTM Approach to Human Activity Recognition», in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, feb. 2020, pagg. 362–366. doi: 10.1109/ICAIIC48513.2020.9065078.

[230] J. Li, H. Zhang, X. Zhang, e C. Li, «Single Channel Speech Enhancement Using Temporal Convolutional Recurrent Neural Networks», feb. 2020, Consultato: gen. 05, 2021. [Online]. Disponibile su: https://arxiv.org/abs/2002.00319v1

[231] H. Wang *et al.*, «Wearable Sensor-Based Human Activity Recognition Using Hybrid Deep Learning Techniques», *Security and Communication Networks*, vol. 2020, pagg. 1–12, lug. 2020, doi: 10.1155/2020/2132138.

[232] MATLAB & Simulink - MathWorks, «Sequence-to-Sequence Classification Using 1-D Convolutions». https://it.mathworks.com/help/deeplearning/ug/sequence-to-sequence-classification-using-1-d-convolutions.html (consultato gen. 07, 2021).

[233] J. Brownlee, «Repeated k-Fold Cross-Validation for Model Evaluation in Python», *Machine Learning Mastery*, ago. 02, 2020. https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/ (consultato gen. 10, 2021).

[234] M. Kuhn e K. Johnson, «Over-Fitting and Model Tuning», in *Applied Predictive Modeling*, M. Kuhn e K. Johnson, A c. di New York, NY: Springer, 2013, pagg. 61–92. doi: 10.1007/978-1-4614-6849-3_4.

[235] D. P. Kingma e J. Ba, «Adam: A Method for Stochastic Optimization», *arXiv:1412.6980 [cs]*, gen. 2017, Consultato: gen. 09, 2021. [Online]. Disponibile su: http://arxiv.org/abs/1412.6980

[236] V. Bushaev, «Adam — latest trends in deep learning optimization.», *Medium*, ott. 24, 2018. https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c (consultato gen. 11, 2021).

[237] J. Brownlee, «Gentle Introduction to the Adam Optimization Algorithm for Deep Learning», *Machine Learning Mastery*, lug. 02, 2017. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (consultato gen. 11, 2021).

[238] T. Kavzoglu, «Chapter 33 - Object-Oriented Random Forest for High Resolution Land Cover Mapping Using Quickbird-2 Imagery», in *Handbook of Neural Computation*, P. Samui, S. Sekhar, e V. E. Balas, A c. di Academic Press, 2017, pagg. 607–619. doi: 10.1016/B978-0-12-811318-9.00033-8.

[239] «Compare predictive accuracies of two classification models - MATLAB testcholdout - MathWorks Italia». https://it.mathworks.com/help/stats/testcholdout.html#bup0p8g-1 (consultato feb. 16, 2021).

[240] H. Wu e S. Prasad, «Convolutional Recurrent Neural Networks forHyperspectral Data Classification», *Remote Sensing*, vol. 9, n. 3, Art. n. 3, mar. 2017, doi: 10.3390/rs9030298.

[241] G. Yuan, X. Liu, Q. Yan, S. Qiao, Z. Wang, e L. Yuan, «Hand Gesture Recognition Using Deep Feature Fusion Network Based on Wearable Sensors», *IEEE Sensors Journal*, vol. 21, n. 1, pagg. 539–547, gen. 2021, doi: 10.1109/JSEN.2020.3014276.

[242] C. Min, A. Mathur, A. Montanari, e F. Kawsar, «An early characterisation of wearing variability on motion signals for wearables», in *Proceedings of the 23rd International Symposium on Wearable Computers*, New York, NY, USA, set. 2019, pagg. 166–168. doi: 10.1145/3341163.3347716.

[243] S. Kataria e V. Ravindran, «Digital health: a new dimension in rheumatology patient care», *Rheumatol Int*, vol. 38, n. 11, pagg. 1949–1957, nov. 2018, doi: 10.1007/s00296-018-4037-x.
[244] K.-H. Yu, A. L. Beam, e I. S. Kohane, «Artificial intelligence in healthcare», *Nature Biomedical Engineering*, vol. 2, n. 10, Art. n. 10, ott. 2018, doi: 10.1038/s41551-018-0305-z.
[245] N. Schaffert, T. B. Janzen, K. Mattes, e M. H. Thaut, «A Review on the Relationship Between Sound and Movement in Sports and Rehabilitation», *Front. Psychol.*, vol. 10, 2019, doi: 10.3389/fpsyg.2019.00244.
[246] J. K. Bizley, «Chapter 26 - Audition», in *Conn's Translational Neuroscience*, P. M. Conn, A c. di San Diego: Academic Press, 2017, pagg. 579–598. doi: 10.1016/B978-0-12-802381-5.00042-7.
[247] J. E. Crasta, M. H. Thaut, C. W. Anderson, P. L. Davies, e W. J. Gavin, «Auditory priming improves neural synchronization in auditory-motor entrainment», *Neuropsychologia*, vol. 117, pagg. 102–112, ago. 2018, doi: 10.1016/j.neuropsychologia.2018.05.017.
[248] M. H. Thaut, G. C. McIntosh, e V. Hoemberg, «Neurobiological foundations of neurologic music therapy: rhythmic entrainment and the motor system», *Front. Psychol.*, vol. 5, 2015, doi: 10.3389/fpsyg.2014.01185.
[249] E.-J. Hossner, F. Schiebl, e U. Göhner, «A functional approach to movement analysis and error identification in sports and physical education», *Front. Psychol.*, vol. 6, 2015, doi: 10.3389/fpsyg.2015.01339.
[250] J. F. Dyer, P. Stapleton, e M. W. M. Rodger, «Sonification as Concurrent Augmented Feedback for Motor Skill Learning and the Importance of Mapping Design», *The Open Psychology Journal*, vol. 8, n. 1, dic. 2015, doi: 10.2174/1874350101508010192.
[251] R. Sigrist, G. Rauter, L. Marchal-Crespo, R. Riener, e P. Wolf, «Sonification and haptic feedback in addition to visual feedback enhances complex motor task learning», *Exp Brain Res*, vol. 233, n. 3, pagg. 909–925, mar. 2015, doi: 10.1007/s00221-014-4167-7.
[252] L. Shams e A. R. Seitz, «Benefits of multisensory learning», *Trends in Cognitive Sciences*, vol. 12, n. 11, pagg. 411–417, nov. 2008, doi: 10.1016/j.tics.2008.07.006.
[253] A. O. Effenberg, U. Fehse, G. Schmitz, B. Krueger, e H. Mechling, «Movement Sonification: Effects on Motor Learning beyond Rhythmic Adjustments», *Front. Neurosci.*, vol. 10, 2016, doi: 10.3389/fnins.2016.00219.
[254] D. S. Scholz, S. Rhode, M. Großbach, J. Rollnik, e E. Altenmüller, «Moving with music for stroke rehabilitation: a sonification feasibility study», *Annals of the New York Academy of Sciences*, vol. 1337, n. 1, pagg. 69–76, 2015, doi: https://doi.org/10.1111/nyas.12691.
[255] A. O. Effenberg e G. Schmitz, «Acceleration and deceleration at constant speed: systematic modulation of motion perception by kinematic sonification», *Annals of the New York Academy of Sciences*, vol. 1425, n. 1, pagg. 52–69, 2018, doi: https://doi.org/10.1111/nyas.13693.
[256] S. Ghai, G. Schmitz, T.-H. Hwang, e A. O. Effenberg, «Auditory Proprioceptive Integration: Effects of Real-Time Kinematic Auditory Feedback on Knee Proprioception», *Front. Neurosci.*, vol. 12, 2018, doi: 10.3389/fnins.2018.00142.
[257] M. H. Thaut e P. M. Thaut, *Rhythm, Music, and the Brain: Scientific Foundations and Clinical Applications*. Routledge, 2005.

[258] M. Thaut e V. Hoemberg, *Handbook of Neurologic Music Therapy*. Oxford University Press, 2014.
[259] M. Murgia *et al.*, «Rhythmic Auditory Stimulation (RAS) and Motor Rehabilitation in Parkinson's Disease: New Frontiers in Assessment and Intervention Protocols», *The Open Psychology Journal*, vol. 8, n. 1, dic. 2015, doi: 10.2174/1874350101508010220.
[260] A. Effenberg, U. Fehse, e A. Weber, «Movement Sonification: Audiovisual benefits on motor learning», *BIO Web of Conferences*, vol. 1, pag. 00022, 2011, doi: 10.1051/bioconf/20110100022.
[261] A. O. Effenberg, «Movement sonification: Effects on perception and action», *IEEE MultiMedia*, vol. 12, n. 2, pagg. 53–59, apr. 2005, doi: 10.1109/MMUL.2005.31.
[262] O. Höner, T. Hermann, e C. Grunow, «Sonification of Group Behavior for Analysis and Training of Sports Tactics», 2004. Consultato: feb. 25, 2021. [Online]. Disponibile su: https://pub.uni-bielefeld.de/record/2017476
[263] A. Godbout, C. Thornton, e J. E. Boyd, «Mobile Sonification for Athletes: A Case Study in Commercialization of Sonification», giu. 2014, Consultato: feb. 25, 2021. [Online]. Disponibile su: https://smartech.gatech.edu/handle/1853/52048
[264] H. Ramezanzade, B. Abdoli, A. Farsi, e M. A. Sanjari, «The Effect of Audiovisual Integration on Performance Accuracy and Learning in Motor Task», *Research in Rehabilitation Sciences*, vol. 11, pag. 2015, ago. 2015, doi: 10.22122/jrrs.v11i1.2176.
[265] T. Sipko, E. Glibowski, K. Barczyk-Pawelec, e M. Kuczyński, «The Effect of Chronic Pain Intensity on Sit-to-Stand Strategy in Patients With Herniated Lumbar Disks», *Journal of Manipulative & Physiological Therapeutics*, vol. 39, n. 3, pagg. 169–175, mar. 2016, doi: 10.1016/j.jmpt.2016.02.014.
[266] J. W. S. Vlaeyen e S. J. Linton, «Fear-avoidance and its consequences in chronic musculoskeletal pain: a state of the art», *Pain*, vol. 85, n. 3, pagg. 317–332, apr. 2000, doi: 10.1016/S0304-3959(99)00242-0.
[267] P. W. Hodges e G. L. Moseley, «Pain and motor control of the lumbopelvic region: effect and possible mechanisms», *Journal of Electromyography and Kinesiology*, vol. 13, n. 4, pagg. 361–370, ago. 2003, doi: 10.1016/S1050-6411(03)00042-7.
[268] I. P. J. Huijnen *et al.*, «Differences in activity-related behaviour among patients with chronic low back pain», *European Journal of Pain*, vol. 15, n. 7, pagg. 748–755, ago. 2011, doi: 10.1016/j.ejpain.2010.11.015.
[269] M. Varacallo, T. D. Luo, e N. A. Johanson, *Total Knee Arthroplasty Techniques*. StatPearls Publishing, 2020. Consultato: feb. 27, 2021. [Online]. Disponibile su: https://www.ncbi.nlm.nih.gov/books/NBK499896/
[270] S. J. Farquhar, D. S. Reisman, e L. Snyder-Mackler, «Persistence of Altered Movement Patterns During a Sit-to-Stand Task 1 Year Following Unilateral Total Knee Arthroplasty», *Physical Therapy*, vol. 88, n. 5, pagg. 567–579, mag. 2008, doi: 10.2522/ptj.20070045.
[271] S. J. Hampton, T. P. Andriacchi, e J. O. Galante, «Three dimensional stress analysis of the femoral stem of a total hip prosthesis», *Journal of Biomechanics*, vol. 13, n. 5, pagg. 443–448, gen. 1980, doi: 10.1016/0021-9290(80)90038-X.
[272] A. Chang, «The Role of Artificial Intelligence in Digital Health», in *Digital Health Entrepreneurship*, S. Wulfovich e A. Meyers, A c. di Cham:

Springer International Publishing, 2020, pagg. 71–81. doi: 10.1007/978-3-030-12719-0_7.