POLITECNICO DI TORINO

Master Degree Course in Computer Engineering

Master Thesis

LIS2SPEECH

LIS translation in written text and spoken language

Supervisor
prof. Maurizio Morisio

Candidate
Giuseppe Mercurio

Company Tutor
ORBYTA Tech
Eng. Carla Melia

Academic Year 2020-2021

Abstract

Deaf and hard-of-hearing people can communicate with each other using Sign Language, but they may have difficulties in connecting with the rest of society. Sign Language Recognition is a field of study that started to be analysed back in 1983, but only in the last decade this task gained more attention. Most of the published works are related to American, Chinese and German sign languages. On the other hand, the number of studies on the Italian Sign Language (LIS) is still scarce. Thus, this work aims to offer a novel mechanism to translate isolated LIS signs into Italian written text and speech.

In order to solve the expressed problem, Neural Networks, Deep Learning and Computer Vision have been exploited to create an application, called *LIS2Speech* (LIS2S), capable of returning the Italian translation of a LIS sign, performed within a recorded video. The method relies on hands, body and face skeletal features extracted from RGB videos without the need for any additional equipment, such as colour gloves. Since the goal is to embrace as many people as possible, LIS2S has been developed as a Progressive Web App, which is able to be run on any device, being it a computer or a smartphone, equipped with a camera.

The results obtained with the described approach are in line with those obtained by automatic tools that have been developed for other sign languages, allowing the model to correctly understand and discriminate between signs belonging to a vocabulary of 50 words, which is in accord with the size of other corpora for isolated sign language recognition. In addition, a new dataset for Continuous Sign Language Recognition (CSLR) has been created and is being constantly expanded, to create a publicly available benchmark for this kind of task.

Finally, although the conducted experiments yielded promising results, this work has just scratched the surface of the problem. The need for a corpus able to tackle CSLR tasks has emerged, since the proposed solution can translate only a single sign at a time. Other future works may examine the possibility of performing sentence segmentation, so that the obtained isolated signs can be translated by the actual model; moreover, to produce a very useful application for real-life purposes, it is necessary to convert the present prototypes into real-time instruments. Finally, another improvement concerns the extension of the number of signs the proposed design can translate, to enlarge the application fields of LIS2S.

Acknowledgements

I would like to offer my special thanks to the entire ORBYTA development team, whose help and support during the writing of this work have been really much appreciated.

I am particularly grateful for the assistance given by my company tutor, Eng. Carla Melia, for her patient guidance, enthusiastic encouragement and extremely useful suggestions for this research work.

Finally, I wish to thank my family and friends for their support and inspiration throughout my study.

Contents

List of Figures

Introduction

Spoken languages and sign languages are different in a number of important ways: the former make use of the "vocal - auditory" channel, since the sound is produced with the mouth and perceived with the ear; the latter instead use the "corporal - visual" channel, signs are produced with the body (hands, arms, facial expressions) and perceived with the eyes.

There are several flavours of sign languages, due to the fact that they are not international, and even inside a national sign language different dialects are present. They are natural languages, since they evolved spontaneously wherever communities of deaf people had the possibility of communicating mutually, and are not derived from spoken languages, because they have their own vocabulary and grammatical structures [1].

The fundamental building block of a sign language is a gloss, which combines manual and non-manual features and represents the closest meaning of a sign [2]. Based on the context, a specific feature can be the most important factor in the interpretation of a gloss: it may change the meaning of a verb, provide spatial or temporal information and discriminate between objects or people.

As known, there is an intrinsic difficulty in the communication between the deaf community and the rest of the society (according to ANSA, in 2018 there were more than 72 million people all over the world using sign languages), so the design of robust systems for automatic sign language recognition would largely reduce this gap.

The definition of Sign Language Recognition (SLR) can be expressed as the task of deducing glosses performed by a signer from video recordings. It can be considered some way related to human action or gesture recognition, but automatic SLR exhibits the following key additional challenges:

- The interpretation of sign language is highly affected by the exact position in the surrounding space and context. For example, there are no personal pronouns (e.g., "he", "she" etc.), because the signer points directly to any involved actor.

- Many glosses are only discernible by their component non-manual features and they are usually difficult to be detected [3].

- Based on the execution speed of a given gloss, it may have a different meaning. For instance, signers would not use two glosses to express "run quickly", but they would simply accelerate the execution of the involved signs [3].

Machine Learning (ML) and Deep Learning (DL) mechanisms are the base of the so-called Computer Vision (CV); it is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. SLR is a task extremely related to CV; it takes advantage from the significant improvement in performance gained by many video-related tasks, thanks to the rise of deep networks.

In this work, a new approach to the Italian Sign Language (LIS - *Lingua Italiana dei Segni*) is proposed; the goal is to produce a tool useful for improving the integration of deaf people with the rest of the society; this tool should be easily accessible by anyone, and this is why the choice of developing an application able to be run on laptop or even smartphones has been taken. The name of the application described in this dissertation is *LIS2Speech* (LIS2S).

The main contributions of this work can be summarized as follows:

- An improved version of the A3LIS-147 dataset [4]; this dataset has been used as the initial training dataset for the neural network which is the core of LIS2S application, but then it has been expanded with lots of new words and videos.
- A new pipeline to extract meaningful information from videos of people using LIS; in particular, the Mediapipe framework [5] and its Python implementation have been used to extract 3D coordinates of hands [6], face [7] and pose [8].
- An improved implementation of the neural network that has been proposed by the DeepGRU project [9], whose focus was to obtain an easy to train neural network capable of analysing temporal sequences of data; the main element of this architecture is the Gated Recurrent Unit (GRU) [10], a gating mechanism used in Recurrent Neural Networks (RNN) [11].
- A complete Progressive Web Application (PWA), created from scratch, that can be used to interface with the neural network briefly introduced previously. This application is intended to support deaf people in communicating with hearing people, producing a vocal and textual interpretation of the signs which have been executed by the user.

In order to ease the reading of this work, this is how it is organized: in chapter 1 the state-of-the-art and the related works are exposed, together with an introduction to the PWAs and the technologies adopted in the project. In the chapter 2, first the dataset structure is explained; subsequently, the selected pipeline for the final application is described, together with the trained models exploited to accomplish the sign recognition. Then, in chapter 3 interpretations and insights of the

conducted experiments are discussed. Finally, conclusions are drawn and future research directions are highlighted.

Chapter 1

Related work and technologies

In the following sections, a summary of the main findings, together with a brief overview of the technologies used in the work correlated to this dissertation, are provided. First, the related works are exposed in order to set a starting point for this research; then follows a general introduction to used tools and frameworks, so that the successive chapters can be easily read and understood.

1.1 State of the art

The past decade has seen the rapid expansion of DL techniques in many applications involving spatio-temporal inputs. The CV is an incredible promising research area, with ample room for improvement; in fact video-related tasks such as human action recognition [12], gesture recognition [13], motion capturing [14] etc. have seen considerable progress in their development and performance. SLR is extremely related to CV, since it requires the analysis and processing of video chunks or sequences to extract meaningful information; this is the reason most approaches tackling SLR have adjusted to this direction.

SLR can play an important role in addressing the issue of deaf people integration with the rest of the society. The attention paid by the international community to this particular problem has been growing during the last years; the number of published studies, but also the quantity of available data sets is increasing. There are different automatic SLR tasks, depending on the detail level of the modelling and the subsequent recognition step; these can be approximately divided in (Figure 1.1):

- *Isolated SLR*: in this category, most of the methods aim to address video segment classification tasks, given the fundamental assumption that a single gloss is present [15], [16], [17].

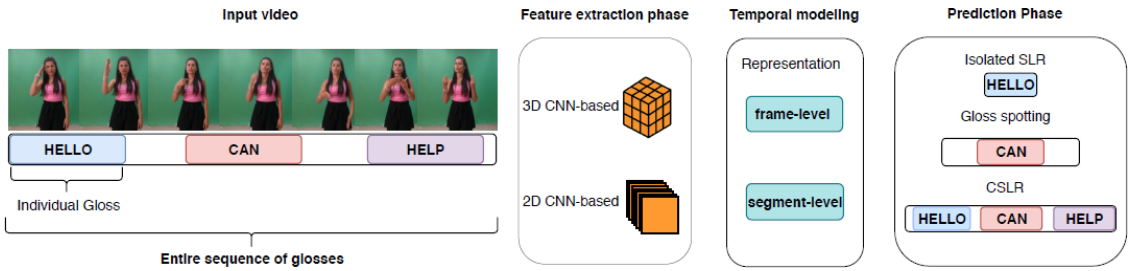


Figure 1.1: Overview of SLR categories

Adapted from [26].

- *Sign detection in continuous streams*: the objective of these approaches is to recognize a set of predefined glosses in a continuous video flow [18], [19], [20].
- *Continuous SLR* (CSLR): the goal of this methods is to identify the sequence of glosses present in a continuous or non-segmented video sequence [21], [22], [23]. The characteristics of this particular category of mechanisms are most suitable for the requirements of real-life SLR applications [24].

This distinction is necessary to understand the different kinds of problems present for each task; historically, before the advent of the deep learning methods, the focus was on identifying isolated glosses and gesture spotting, so this is why studies on isolated SLR are more common. In Figure 1.2 the trend of isolated and continuous recognition studies can be observed in blocks of five years up until 2020; the growth looks exponential for isolated studies, while it is close to linear for continuous studies. This can reflect the difficulty of the continuous recognition scenario and the scarcity of available training datasets. In fact, on average it can be observed that there are at least twice as many studies published using isolated sign language data.

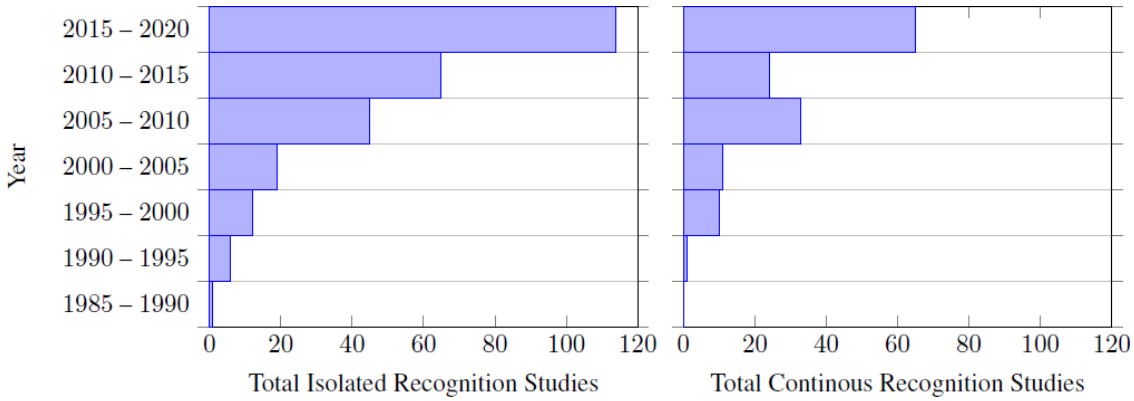


Figure 1.2: SLR studies in time

Showing the number of published recognition results between 1983 and 2020. Adapted from [27].

In terms of vocabulary size, the majority of isolated SLR works model a very limited amount of signs (i.e., below 50 signs), while this is not the case when comparing CSLR, where the overall studies are more or less evenly spread across all sign vocabularies. This trend can be observed in Figure 1.3.

1.1.1 Type of input data

There are different kinds of input data used in the literature; these data are consumed by the recognition algorithms to extract features and perform computation. Table 1.1 presents an overview of the input data: as can be seen from the table, RGB is the most popular type of input data both for small and larger scale vocabulary ranges. Coloured and Electronic Gloves have been applied only to small and medium vocabulary tasks and did never get significant attention over the years. The Depth input data became popular only after the release of the Kinect sensor [25] in 2010.

Input data could be aggregated into "non-intrusive" and "intrusive" categories, meaning that in the intrusive methods the recognition subject needs to be modified to perform body pose estimation and general feature extraction. This said, "RGB" and "Depth" are non-intrusive capturing methods, while "Colour Glove", "Electronic Glove", and "Motion Capturing" are intrusive techniques. Focusing on the temporal evolution of the studies, it can be seen in Table 1.2 that there is a clear paradigm shift after 2005, when the dominating intrusive capturing methods started to be less and less used: in fact, their adoption trend decreased from around 70% to less than 30% and it is going to constantly reduce over time, thanks to the notable improvements of Deep Learning techniques.

Considering the number of recognition results per sign language and employed

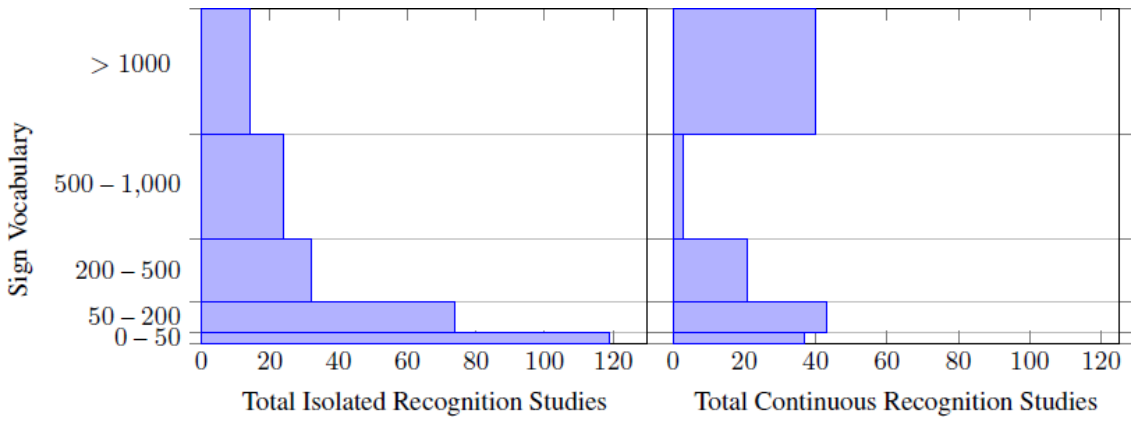


Figure 1.3: SLR studies by vocabulary size

Showing the number of published results between 1983 and 2020 and the size of their modelled sign vocabulary. Adapted from [27].

Vocabulary	RGB	Depth	Colour Glove	Elect. Glove	Motion Capt.
> 1000	85	4	0	17	13
500 - 1000	93	41	0	4	4
200 - 500	77	23	6	12	12
50 - 200	73	11	6	16	15
0 - 50	72	24	13	10	8

Table 1.1: Shows the fraction in [%] of published SLR results that make use of a specific input data type (e.g. "RGB", "Depth", etc.) relative to all published results that fall in the same modelled vocabulary range. E.g. this table reads like: "85% of all results with a modelled vocabulary above 1000 signs employ RGB input data".

type of input data, it is worth noting that, as depicted in Table 1.3, experiments recognizing American Sign Language (ASL) are dominated by RGB data; Chinese Sign Language (CSL), instead, may count on a larger number of results involving RGB-D (colour with depth) data, together with just RGB data. German Sign Language (Deutsche Gebärdensprache) (DSG) and most other sign languages focus mainly on RGB based recognition. In this work a new approach for LIS recognition will be introduced, based on 3D skeletal data, which are supposed to require less computational power than NN involving convolutional operations over images and frames. However, as we will see later on, few studies have investigated LIS recognition in any systematic way; this may reflect the different perception of the problem from the Italian society with respect to other ones, like USA or China, which started to tackle SLR difficulties years ago.

Years	non-Intrusive	Intrusive
> 2015	89	11
2010 - 2015	87	13
2005 - 2010	72	28
2000 - 2005	30	70
1995 - 2000	27	73
1990 - 1995	29	71
< 1990	50	50

Table 1.2: Shows the fraction in [%] of published SLR results that make use of non-intrusive data input capturing methods (i.e. "RGB" or "Depth") and those that are intrusive (i.e. "Color Glove", "Elect. Glove" or "Motion Capt.") relative to a year range.

Input Data	ASL	CSL	DGS	LIS
RGB	93	53	59	4
Depth	14	38	3	0
Colour Glove	4	1	4	0
Elect. Glove	16	17	1	1
Motion Capt.	14	15	0	0

Table 1.3: Shows the number of published SLR results per sign language and type of input data.

1.1.2 Sign language parameters

The previous section has analysed the different kinds of input data used for SLR. The following one will investigate the sign language parameters and features that are extracted based on the input data. A distinction can be made between manual parameters, such as hand shape, movement, location and orientation; and non-manual parameters, for example, head, mouth, eyes and eyebrows. In the majority of studies on SLR, the hand shape is the most covered parameter, while location and movement are the next popular ones across all vocabulary sizes below 1000 signs. However, using recent DL based feature extractors, it is possible to infer hand posture and orientation parameters starting only from the hand shape.

In Table 1.4 the hand location, movement, shape and orientation are aggregated into manual parameters; head, mouth, eyes and eyebrows are instead indicated as non-manual parameters. In addition to these parameters, other kind of information can be extracted from the entire RGB image, for example, body joints or motion estimation: these are referred to as global features. What is interesting in this table is the shifting from manual to global features when considering larger modeled vocabularies: while the former usage is dominant for vocabularies of up to 50 signs, the latter takes the lead for large vocabularies above 1000 signs. The increasing trend of global features may have two reasons:

- The availability of body joints and full depth image features with the realise of the Kinect sensor in 2010.
- The shift towards DL techniques, which allowed to input fullframes images and videos instead of manual feature engineering.

Both hypotheses can be confirmed by looking at the Table 1.5: global features, in fact, started to stand out just after 2010 (when Kinect sensor was released) and also coincides with the took off of DL for SLR, which is traced back to 2015. According to [27], the ASL has the most published results overall, but non-manual parameters are most frequently included in studies on DGS. In addition, despite the fact that CSL is the second most frequently researched sign language, there is only a single study that includes non-manual parameters like the face [28]. Eye and specifically

Vocabulary	Manual	non-Manual	Global
> 1000	49	15	64
500 - 1000	67	0	62
200 - 500	77	23	52
50 - 200	74	7	35
0 - 50	90	7	20

Table 1.4: Shows the fraction in [%] of published sign language recognition results that employ manual, non-manual or global features relative to all published results that fall in the same vocabulary range. E.g. this table reads like: "49% of all results with a modeled vocabulary above 1000 signs include manual parameters".

eyebrows have only been tackled in few studies [29], [30], [31], [32], [33], [34], [35], while eye gaze or eye blinks have not being taken under account for SLR until now. In this work, both manual, non-manual and global parameters have been considered: in fact, the 3D skeletal data which was introduced in the previous section are a combination of coordinates extracted from hands, head, mouth, eyebrows and body joints.

Vocabulary	Manual	non-Manual	Global
> 2015	47	8	66
2010 - 2015	100	17	37
2005 - 2010	99	13	1
2000 - 2005	100	0	0
1995 - 2000	100	5	0
1990 - 1995	100	0	0
< 1990	100	0	0

Table 1.5: Shows the fraction in [%] of published sign language recognition results that employ manual, non-manual or global features relative to all published results that fall in the same range of years. E.g. this table reads like: "47% of all results released after 2015 include manual parameters".

1.1.3 Analysis by sign language

Having discussed the different kind of input data, lets now consider the results divided by sign language over time and per vocabulary range. Standing to [27], ASL has usually been the language with the most results published; in practice this is only true for vocabularies below 200 signs. On larger vocabularies CSL is leading and, on vocabularies above 1000 signs, DGS has significantly more research published; in particular, the RWTH-PHOENIX-Weather [30] dataset used for DGS

consists of a vocabulary of more than 1000 signs and represents the only resource for large-scale CSLR worldwide. This can partly be explained by the public availability of sign language datasets, which is extremely low for most of the sign languages. All these information can be summarized in Table 1.6 and Table 1.7.

Year	ASL	CSL	DGS	LIS
> 2015	46	35	40	1
2010 - 2015	32	21	17	2
2005 - 2010	30	4	4	2
2000 - 2005	11	10	2	0
1995 - 2000	5	2	1	0
1990 - 1995	5	0	0	0
< 1990	1	0	0	0

Table 1.6: Shows the number of published SLR results per sign language and year. This table reads like: "There are 46 results published after 2015 that use ASL".

Concerning the Italian Sign Language, it is worth citing the A3LIS-147 dataset [4], realised in 2012 from the A3Lab research group based at the *Università Politecnica delle Marche, Ancona*. The dataset is composed of 147 isolated signs, performed by 10 different signers; the signs have been organized in six categories, related to different situations of the common life, as it is shown in Table 1.8. These likely represent the domains where automatic tools for social inclusion of deaf people could be effectively applied. Each video presents a single sign which is preceded and succeeded by the occurrence of the "silence" sign.

The work done by A3Lab research group is commendable, but it is focused only on isolated signs; this means it can be only used to conduct experiments on Isolated SLR. Because of this, this dissertation concerns Isolated SLR, since the NN described and used in this work, as explained later on, is trained based on this dataset.

In contrast to this A3LIS-147 dataset, however, the RWTH-PHOENIX-Weather

Vocabulary	ASL	CSL	DGS	LIS
> 1000	4	11	36	0
500 - 1000	7	10	2	0
200 - 500	8	17	16	0
50 - 200	43	20	3	2
0 - 50	57	14	7	3

Table 1.7: Shows the number of published SLR results per sign language and modeled vocabulary. This table reads like: "There are 4 published results of ASL that use a vocabulary bigger than 1000 signs".

Scenario	Signs per scenario	Vocabulary examples
Public institute	39	*employee, municipality, timetable*
Railway station	35	*train, ticket, departure*
Hospital	19	*emergency, doctor, pain*
Highway	8	*traffic, toll booth, delays*
Common life	16	*water, telephone, rent*
Education	30	*school, teacher, exam*

Table 1.8: The proposed real-life scenarios, number of signs per scenario and vocabulary examples for the A3LIS-147 dataset.

is made of videos concerning sequences of signs. This dataset required a period of three years (from 2009 to 2011) to record daily news and weather forecast airings of the German public TV-station PHOENIX featuring sign language interpretation. It consists of more than 1080 different signs. In the end, only the weather forecasts of a subset of 386 editions have been transcribed using gloss notation. The transcriptions have been carried out by deaf and hard-of-hearing native speakers of DGS, and an additional translation of the glosses into spoken German has been created to capture allowable translation variability. A structure and dataset like this is actually missing for the LIS, and would be a remarkable step towards real-life SLR applications usable by Italian deaf people, allowing researchers to investigate solutions for CSLR.

1.2 Technologies overview

In the next section the most important tools used during the development of LIS2S application will be briefly explained. Going deeper into the details, LIS2S is made previously of two parts: on the client-side there is a Progressive Web Application (PWA), which will be used by the admins and users to access the functionality provided by the software; and the Back-end, which will be managed by a server process constantly listening for requests coming from the application. Whenever a new request is received from the server, it will run a new instance of a Docker container: this will be in charge of processing the data coming with the request and returning back the translation to the client.

1.2.1 Progressive Web Apps

Progressive Web Apps (PWA) are web apps that use emerging web browser APIs and features along with traditional progressive enhancement strategies to bring a native app-like user experience to cross-platform web applications. PWAs are a useful design pattern, although they are not a formalized standard. The term

"Progressive Web App" is not a formal official name, but just a shorthand used initially by Google for the concept of creating a flexible, adaptable app using only web technologies. They can be thought as similar to AJAX or other similar patterns that involve a set of application attributes, including use of specific web technologies and techniques.

PWAs are web apps developed using a number of specific technologies and standard patterns to allow them to take advantage of both web and native app features. For example, web apps are more discoverable than native apps; it is a lot easier and faster to visit a website than to install an application, and it is also possible to share web apps by sending a link. On the other hand, native apps are better integrated with the operating system and therefore offer a more seamless experience for the users. Native apps can be installed so that they work offline, and users love tapping their icons to easily access their favourite apps, rather than navigating to them using a browser. PWAs allow to create web apps that can enjoy these same advantages, as depicted in Figure 1.4.

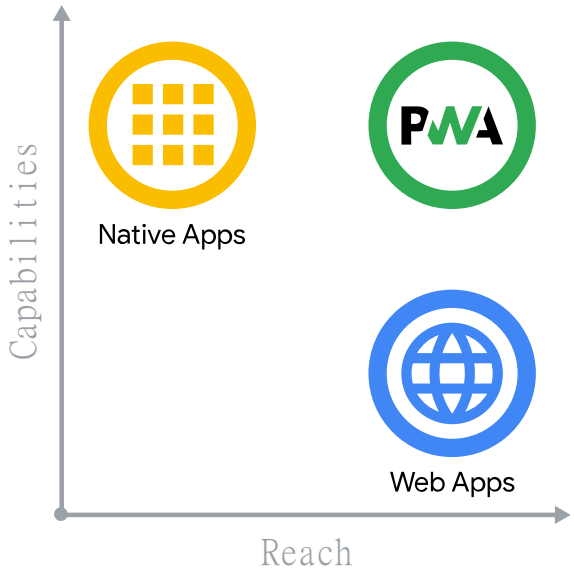


Figure 1.4: Native vs. Web vs. PWA

Showing capabilities vs. reach of platform-specific apps, web app, and progressive web apps.

There are some key principles a web app should try to observe to be identified as a PWA. It should be:

- *Discoverable*, so the contents can be found through search engines.
- *Installable*, so it can be available on the device's home screen or app launcher.
- *Linkable*, so that it can be shared by sending a URL.

- *Network independent*, so it works offline or with a poor network connection.
- *Progressive*, so it is still usable on a basic level on older browsers, but fully-functional on the latest ones.
- *Re-engageable*, so it is able to send notifications whenever there is new content available.
- *Responsive*, so it is usable on any device with a screen and a browser (mobile phone, tablet, laptop, TV, etc.).
- *Safe*, so the connections between the user, the app, and the server are secured against any third parties trying to get access to sensitive data.

There are many success stories of companies trying the PWA route, opting for an enhanced website experience rather than a native app, and seeing significant measurable benefits as a result. Examples could be *Pinterest*, which rebuilt their mobile site as a PWA and observed a core engagement increase by 60%, together with a 44% increase in user-generated ad revenue; or AliExpress PWA, which has also seen much better results than either web or native app, with 104% increase in conversion rates for new users.

Since the goal of the LIS2S application is to reach as many people as possible, to help deaf and hard-of-hearing people integrating in the society, the decision to use a PWA was taken. In this way it is possible to exploit physical and portable devices of whatever form factor, starting from smartphones and going to laptops or multimedia totems. With this also come all the different advantages of PWAs, such as the security for the exchanged data and the responsiveness of the application.

1.2.2 Ionic Framework

The PWA is the core concept of the client-side application, but what is more important for the end user is the User Interface (UI). Concerning this topic, the Ionic Framework has been selected in order to exploit an open-source UI toolkit for building performant, high-quality mobile and desktop apps using web technologies (HTML, CSS, and JavaScript) with integration for popular frameworks like Angular, React and Vue.

Ionic is a cross-platform framework and its goal is to allow developers to use technology that they already know to build apps in technology they are not so familiar with; for example, Ionic allows to use HTML/CSS/JS to build iOS apps, Android apps, PWAs, desktop apps, or apps for any other platform equipped with a browser. In the same way, for instance, React Native utilizes the JS framework React, but renders native UI elements at runtime, making it possible to build iOS and Android apps. In general, all hybrid app development frameworks allow access to native device functionality like the camera, biometrics, geo-localization

and offline storage. Still, there are some philosophical differences between these technologies. They could be divided into two major categories: Hybrid-Native and Hybrid-Web, whose most important differences are depicted in Table 1.9.

Hybrid-Native

React Native, along with Xamarin, NativeScript and other frameworks like Flutter, allows to program the UI in one language that then organizes native UI controls at runtime. In this case, cross-platform means that React Native has got JavaScript functions that are mapped to system calls, allowing to manage the native UI for specific mobile platforms. For example, a React Native component that renders text on a mobile app will be converted into two different components: *TextView* for Android and *UIView* for iOS. Thus, not components but code is shared across platforms. This means that the underlying native UI component and any customization of that component must be supported by React Native.

The amount of code reuse in React Native projects will vary depending on how much the application will be customized at the native layer. If only fundamental UI elements, like View, Text and Image are used, then the code will be able to run generally on any different platform. On the other hand, if native customizations are required, the project will need three separate codebases: two necessary to manage Android and iOS UI, and another shared codebase where the controller code will settle.

Finally, React Native and other Hybrid-Native frameworks are very good solutions with lots of advantages: the possibility to build a real native UI using mostly JavaScript, still being able to share a big part of the code, but at least a basic knowledge of native specific languages and features is required to fully customize the application.

Hybrid-Web

These kind of frameworks, among which Ionic stands, take a different stance. UI components used in applications are actually running across all platforms, instead of having JavaScript code that acts as a bridge between them and native IU elements. In a mobile app, these components are executed in a web-view container; in a PWA they run in the browser; in a desktop app they are rendered in a desktop container like Electron.

These characteristics are important because they ensure a unique and consistent UX across different platforms, together with the possibility to reuse a particular component over any project. In addition, frameworks like Ionic are based on HTML, CSS and JavaScript, allowing the UI to run on a portable and standardized layer. Thanks to the CSS styling properties, it is easier to customize an application's design based on each platform. Lastly, the biggest benefit of building on the web

is the stability of the platform: with open web standards supported in all modern browsers, whatever is being developed today is going to work also in the future.

Using a Hybrid-Web approach, the UI is built with HTML/CSS/JS, and native functionalities can be accessed through portable APIs that abstract the underlying platform dependencies. Only certain features, like the camera, depend on the platform. This allows the app to run on any system with web support: iOS, Android, browsers, Desktop, PWAs etc. The most important aspect is that the UI layer can be shared between different platforms: even if the look of the app is customized over Ionic's default platform-specific design, the app will never be split into multiple codebases.

Concerning performance, for the vast majority of cases, hybrid frameworks (both Hybrid-Native and Hybrid-Web) will have similar behaviour. Nevertheless, if performance is the main goal of the developed application, then a fully native approach would be preferable: using the Native SDKs will surely be worth the trade-off in terms of cost and development time.

In the end, the Ionic framework based on the React library, formerly known as ReactJS or Ionic React, was chosen to build the LIS2S application, because it satisfies the main principles of reachability and pleasantness of use needed for its goals. Thanks to this framework, it was possible to obtain an application ready for use both in Android and iOS devices, as well as browsers or desktops. In addition, thanks to the ORBYTA's team experience in React development, the final product is designed to be as simple and powerful as possible.

1.2.3 Firebase

Firebase is the Google's toolset intended to build, improve and grow an application. It covers a large portion of the services that developers would normally build themselves, including analytics, authentication, file storage, push messaging and so on. The services have back-end components which are fully maintained and operated by Google, and scale with little to no effort on the part of the developer. All these services are available through the Firebase Console associated with the developed project.

For this particular application, Firebase Authentication and Cloud Storage have been used: using the former, the user is able to authenticate himself in LIS2S application, to have access to his past translation and feedback; the latter, instead, is used as temporary storage for user uploaded videos: whenever a new translation is required by the user, the application will upload the recorded video to Firebase Cloud Storage and the back-end service will process it in order to return back the translation to the user. Once the translation operation is completed, the video is deleted.

Features	Hybrid-Native	Hybryd-Web
Performance	Excellent, as there is no web-view	Good, as it uses web-view
Code reusability	Same code for Android and iOS apps	Same code for Android and iOS apps, desktop and PWA
Ease of development	Learned once, written anywhere	Written once, reused anywhere
Learning	Very few pre-developed components	Good amount of pre-developed and pre-styled components
Code testing	Needs emulator or mobile device	Can be done on any browser

Table 1.9: The most important differences between Hybrid-Native and Hybrid-Web frameworks.

1.2.4 Git

The previous sections have shown the frameworks used to build and deploy a multi-platform application; to manage the code needed to effectively compose the project, the development team required an easy to use platform to handle code changes and improvements: nowadays, Git is the most used platform for this purpose.

Git is officially defined as a *distributed version control system* (VCS). It allows to track the changes made to project files over time. It is possible to record project changes and return back to a specific version of the tracked files. This system has been developed in order to let different people efficiently work together and collaborate on team projects: each developer can have their own version of the project, and later on these versions can be merged and adapted into the main version of the project.

Git is primarily used via the command-line interface (CLI), accessible using system terminals. The basic container for a project is called *repository*, and it can be either local or remote. Remote repositories are especially useful when working in teams, since people working together will be able to share the project code, see other people's code and integrate it into their local version of the project.

Once a repository is initialized, it's possible to modify the project it contains by *staging* and *committing* code:

- *Committing* is the process in which the changes are "officially" added to the

Git repository;

- *Staging*, instead, consists in adding the changes made to project files into the staging area: initially, it is necessary to start tracking a file; then, once it is modified, Git will consider that file as something that should be added to the next commit.

It's possible to monitor the files that have been modified by checking the status of the repository. Git also allows to move or delete files. The peculiar thing of using a distributed VCS is that it allows to move back and forth from a commit to another one, like a time machine; in addition, it's also possible to create new branches, which can be interpreted as an individual timeline of the projects commits. Many of these alternative environments can be created, so this means that different versions of the project code can exist and be tracked in parallel. Exploiting this mechanism, Git allows to add new features in separate branches, without touching the stable version of the project, which is usually kept in the master branch. Once changes done in a particular branch are satisfactory, it can be merged into a different one, for example the master branch.

During the work carried out for this dissertation, particular attention was paid to versioning, making the best of Git in order to efficiently collaborate.

1.2.5 Python

Moving on towards the back-end section of the LIS2S application, the core of the translation mechanism is performed by the processing of the received videos. This particular task has been realized exploiting the potentiality of the Python programming language, which offers a complete set of tools and libraries extremely useful for Computer Vision and data manipulation.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers prefer using Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables,

evaluation of arbitrary expressions, setting breakpoints, stepping through the code a single line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

During the development of the LIS2S application, lots of packages and libraries have been used to achieve the final translation goal. In the following sections, a brief overview of the most important modules follows.

Open CV

OpenCV is a cross-platform library that can be used to develop real-time computer vision applications. It mostly focuses on image processing, video capture and analysis including features like face detection and object detection.

First, an introduction to CV is needed. It explains how to reconstruct and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modelling and replicating human vision using computer software and hardware. It can also be intended as the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scenes. CV is heavily used in the most disparate domains, such as robotics, medicine, security and so on.

Using the OpenCV library, it is possible to read and write images, capture and save videos, process images (filter or transform them), perform feature detection, detect specific objects such as faces, eyes etc. in videos or images and further more. This library was originally developed in C++; in addition, Python and Java bindings were later provided. It can run on different Operating Systems such as Windows, Linux, FreeBSD and so on.

In particular, OpenCV was used during the development of the LIS2S application in order to manipulate videos of people using LIS. During the training phase, that will be later covered, this module is the key element for the feature extraction from videos; in addition, it is used to process the videos received by the users in the final implementation of the application.

MediaPipe

MediaPipe is a framework which offers cross-platform, customizable ML solutions for live and streaming media. As reported in [5], with MediaPipe a *perception pipeline* can be built as a graph of modular components, including model inference, media processing algorithms and data transformations. Sensory data such as audio and video streams enter the graph, and perceived descriptions such as object-localization and face-landmark streams exit the graph. An example is shown in Figure 1.5.

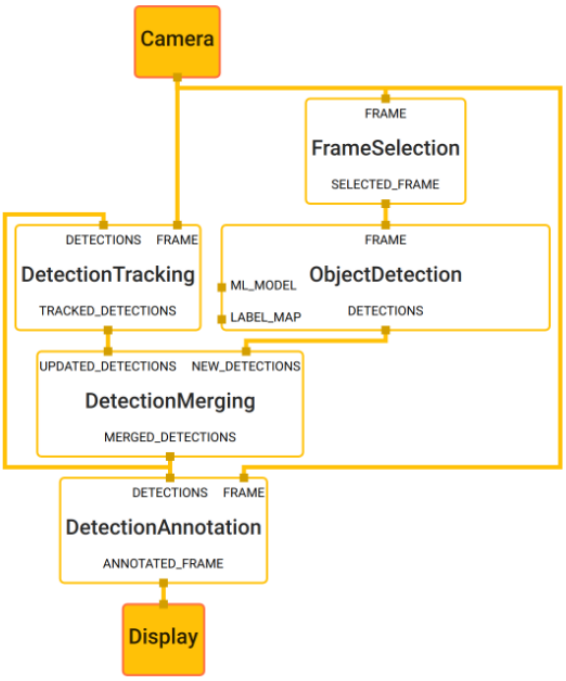


Figure 1.5: Object detection using MediaPipe

The transparent boxes represent computation nodes (calculators) in a MediaPipe graph, solid boxes represent external input/output to the graph, and the lines entering the top and exiting the bottom of the nodes represent the input and output streams respectively. Adapted from [5].

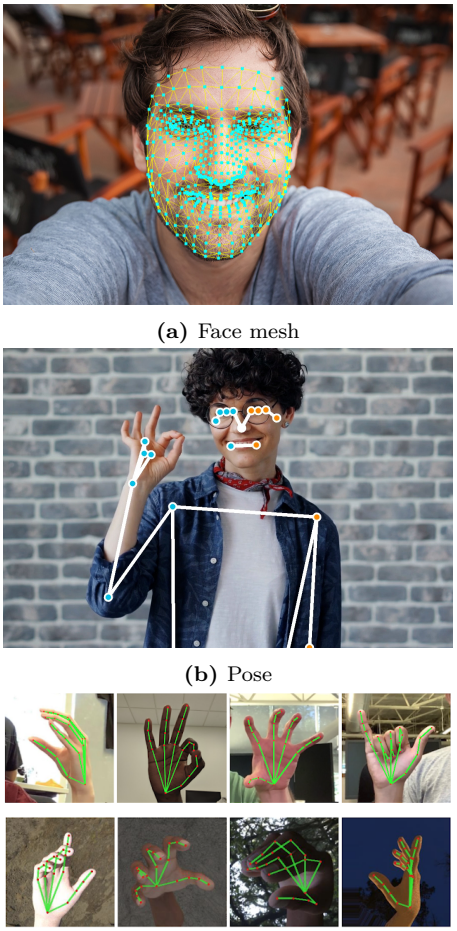
MediaPipe can exploit build-in fast ML inference and processing accelerated even on common hardware, such as smartphones and tablets; since it is a cross-platform framework, it allows to build once and deploy anywhere: unified solutions are available, working across Android, iOS, desktop/cloud, web and IoT. At the time of writing, there are multiple ready-to-use solutions, cutting-edge ML projects demonstrating the full power of the framework. Finally, MediaPipe is free and open source, it is fully extensible and customizable.

MediaPipe offers customizable Python solutions as a prebuilt Python package. It also provides tools for users to build their own solutions. MediaPipe on the Web is an effort to run the same ML solutions built for mobile and desktop also in web browsers; the project is constantly evolving, and a JavaScript version is being developed at a sustained pace.

For the purpose of LIS2S, the Python implementation of some solutions has been used; these particular solutions are the Hands, Face Mesh and Pose solution; examples of what is possible to do with these solutions are provided in Figure 1.6.

These solutions are particularly helpful for the developed application goal, since they allow to extracts important information by images and videos in near real-time; the information extracted are *skeletal data*, which will be used by the NN to

train and understand what kind of sign has been performed by the user.



(a) Face mesh

(b) Pose

(c) Hand crops

Figure 1.6: Three solutions from MediaPipe

Numpy

Going deeper into the video processing task, attention shifts from original videos to the features extracted from them. In order to manipulate all these numbers, arrays and values, the Numpy package has been included in the developed Python scripts used by LIS2S application.

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in the compiled code for performance. NumPy arrays facilitate advanced mathematical and other types of operations on large amounts of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.

The points about sequence size and speed are particularly important in scientific computing. Consider the case of multiplying each element in a 1-D sequence with the corresponding element in another sequence of the same length: if the two sequences contain millions of numbers, the time needed to accomplish the calculation will not be negligible, due to the inefficiencies of looping in Python. Exploiting Numpy, instead, element-by-element operations are speedily executed by precompiled C code. The power of Numpy is in the *vectorization*: it describes the absence of any explicit looping, indexing etc. in the code, and has many advantages (it is more concise, needs fewer lines of code etc.).

Pandas

The data extracted during the previous phases of video processing need to be saved in reliable data structures for further manipulation and analysis, in particular to be used in Neural Network training. This is the reason the Pandas package has been used: it provides fast, flexible, and expressive data structures designed to make working with "relational" or "labelled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

Pandas is well suited for many kinds of data, such as tabular data with heterogeneously typed columns, as in an SQL table or Excel spreadsheet, ordered and unordered (not necessarily fixed-frequency) time series data etc.; for all these reasons, it is the perfect package to be used in LIS2S application to manage and save the time series features extracted from video processing into an SQL database, based on Microsoft Azure.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. Pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

PyTorch

The core of the LIS2S application stands in the ability of recognizing the sign performed by the user in his video. To accomplish this not trivial goal, Deep Learning (DL) mechanisms and methodologies need to be used; DL is part of a broader family of machine learning methods based on Artificial Neural Networks (ANNs). This

kind of structures was inspired by information processing and distributed communication nodes in biological systems (hence the name "Neural Network"), but there are various differences between these two systems. The adjective "deep" in DL refers to the use of multiple layers in a single network. Historically, the first type of classifier was the *linear perceptron* [36], but early work showed that it cannot be a universal classifier, while a network with a non-polynomial activation function with one hidden layer of unbounded width can on the other hand so be. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions.

As the years go on, DL improvements allowed to build technologies that previously were not even imaginable. To make use of these technologies, several frameworks have been developed: among them, the most known are Keras[1], PyTorch[2] and TensorFlow[3].

For many scientists, engineers and developers, TensorFlow was their first DL framework, released back in 2017 but not so user friendly. The difficulties in the usage of this framework are due to the complexity of TensorFlow Execution Engine, in which the actual graph of the NN needs to be compiled first, together with concepts such as Variable scoping, placeholders and sessions, which would lead to boilerplate code.

Over the past couple of years, the two major DL libraries that have gained massive popularity, mainly due to how much easier to use they are over TensorFlow, are Keras and Pytorch. Keras is not a framework on its own, but actually a high-level API that stands on top of other DL frameworks (currently it supports TensorFlow, for instance); its main strength is the ease of use, since it is considered the easiest framework to get up and running fast. Defining NNs is intuitive, thanks to its Functional API which allows to define layers of NNs as functions. PyTorch is per se a DL framework, developed by Facebook's AI research group. Like Keras, it also abstracts away much of the messy parts of programming deep networks. PyTorch lies somewhere between Keras and TensorFlow, since it allows a greater level of flexibility than the former, but at the same time it is not so blundering like the latter. The main differences between these two frameworks are:

- *Classes vs. Functions for defining models*: Keras exploits the Functional API to create models as a set of sequential functions, applied one after the other; in PyTorch, instead, networks are defined as classes and layers are defined as attributes of the class;

[1] `www.keras.io`

[2] `www.pytorch.org`

[3] `www.tensorflow.org`

- *Tensors vs. standard arrays*: a tensor is a multi-dimensional matrix containing elements of a single data type; it differs from a Numpy array because it is immutable and can reside in accelerator's memory (like GPUs). In PyTorch it is very easy to move back and forth between tensors and arrays, thanks to built-in methods, while in Keras it is a little more tricky, since it requires the programmer to have a solid understanding of TensorFlow sessions.

- *Training models*: in Keras it is possible to train a model by simply calling the *fit* method on it; in PyTorch, instead, there are some steps that need to be executed (initialise gradients, run forward and backward pass, compute loss and update weights);

- *Controlling CPU and GPU mode*: in Keras all the computations will be executed on the GPU by default, while in PyTorch it is required to explicitly specify on which device to execute the code; this can be a bit error prone if it is needed to move back and forth between CPU and GPU for different operations.

In general, the advice is to start with Keras, but for this dissertation PyTorch has been chosen, since it provides a perfect balance between ease of use and control over the model training and testing.

1.2.6 Docker

In the course of the LIS2S development, the possibilities offered by the Docker platform have been exploited to generate programs able to run in a separated environment; some of these processes, for example, required to be executed on an OS different from the one actually used for the development, which is Windows 10. For instance, at the beginning of the project, the MediaPipe library for Python was only available on macOs and Linux: this is the reason the feature extraction process was executed in a Docker container which was based on a Linux image.

Docker is an open platform for developing, shipping, and running applications. It allows to separate applications from the underlying infrastructure so that the software can be quickly delivered. Taking advantage of Docker's methodology for shipping, testing and deploying code, it is possible to greatly reduce the delay between writing the code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow to run many containers simultaneously on a given host. Docker provides tools and a mechanisms to manage the lifecycle of containers; once the application is ready, it can be deployed in the production environment as a container.

The core of Docker technology is the Docker Engine; it is a client-server application with these major components:

- A *server* which is a type of long-running program called a daemon process;
- A *REST API* which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A *command line interface* (CLI) client (docker command).

The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Docker speeds up the development lifecycle by allowing developers to work in standardized environments using local containers which provide applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows; in fact, they were combined with the pipeline offered by GitLab to create a new Docker image each time a modification was committed on the remote repository.

Docker's container-based platform allows for highly portable workloads. Containers can run on a developer's local laptop, on physical or virtual machines in a data centre, on cloud providers, or in a mixture of environments. Basically, the idea was to execute the containers on developer's machines during the prototyping of LIS2S, and then move them to cloud providers once the application was ready for production.

Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate in near real time. This feature is essential for the purpose of this project, since the application should be easily reachable by anyone who requests it at any time.

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, it is possible to build an image which is based on the *ubuntu* image, but installs the Apache web server and the developed application, as well as the configuration details needed to make the application run.

A container is a runnable instance of an image. It is possible to create, start, stop, move or delete a container using the Docker API or CLI. By default, a container is relatively well isolated from other containers and its host machine. A container is defined by its image as well as any configuration option which comes with it when it is created or started. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Finally, this chapter has described the technologies and related works which characterize the SLR, in particular focusing on the isolated SLR and all the tools used to face its difficulties.

The next chapter will describe more in the details the dataset used for NN training, together with the pipeline exploited to extract features from videos, feed them to the NN and obtain, given a video, the prediction of the LIS sign which has been performed by the signer.

Chapter 2

LIS2Speech architecture

In the current chapter, the procedures and methods used in this investigation will be presented. First, the complete dataset used for the sign recognition task is going to be described, introducing the improvements applied to the initial corpus; then, the proposed pipeline will be illustrated, explaining how to start from raw videos, extract meaningful information and then feed them to the Neural Network, which is the core element of the sign recognition task.

2.1 Dataset and Database

The main goal of the LIS2S application is the creation of a signer-independent automatic sign recognition system, exploiting the technologies and tools exposed in the previous chapter. For this purpose, a suitable corpus for training the NN responsible for this automatic recognition is needed. Since this is an isolated sign recognition task, the corpus should contain videos with many different signs executed by multiple signers.

In the past decade, only the A3Lab tried to lay the foundations of a dataset like this, creating from scratch the A3LIS-147 database. This corpus is freely available[1] and has been presented in [4]. As previously exposed, it consists of 147 distinct isolated signs executed by 10 different people (7 males and 3 females). Each signer executed all the 147 signs, for a total of 1470 recorded videos. The ENS[2] (*"Ente Nazionale Sordi"*) supported the authors of that work to suitably pre-train the subjects and to choose a meaningful and unambiguous set of signs to be executed. In each of these videos, the person executes a single sign, preceded and ensued by the "silence" sign shown in Figure 2.1. This sign represents a common "rest" position in every-day conversations, thus it has been chosen for this reason.

[1] `www.a3lab.dii.univpm.it/projects/a3lis`

[2] `www.ens.it`

Figure 2.1: The "silence" position

An example of a sign of the A3LIS-147 dataset. Adapted from [4]

In addition to this egregious work, the Analytics team at ORBYTA's *Area 51* software house started to collect new videos to expand the pre-existing A3LIS-147 database. This process was necessary in order to increase as much as possible the performance and accuracies of the proposed model, but also to take some steps toward a new approach for the Italian SLR: in fact, not only videos containing isolated signs have been gathered, but also samples containing phrases. These new videos will be extremely useful in future works, because the aim is to create a database which resembles the RWTH-PHOENIX-Weather one, described in [30]. This means that progress towards CSLR mechanisms are being done for the Italian Sign Language, so that a real-time translation application could be developed in future.

Particular attention has been paid to inserting commonly used phrases in the expanded database, together with phrases which contained signs similar to the ones included in the initial dataset. Each phrase has been subdivided in glosses, little portions of the initial video, each one associated with its meaning. An example can be found in Figure 2.2 For the purpose of this dissertation, only these isolated glosses have been used, but they would be really helpful for further researches and

experimentation in CSLR field.

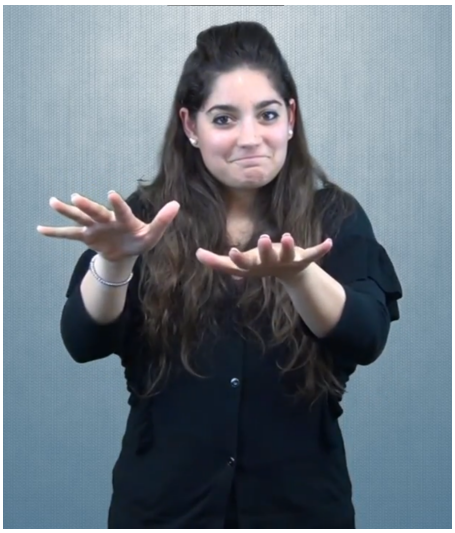


Figure 2.2: The "diffusion" sign

An example of a sign added to the A3LIS-147 dataset.

Moving on towards the actual implementation of the sign recognition mechanisms, an approach similar to the "isolated word recognition" task in speech recognition has been used, so the final system will perform isolated sign recognition; this means that it recognizes a single sign present in an input video. To accomplish this task, the LIS2S application will follow a specific pipeline, studied so that features can be extracted from videos and fed to a Neural Network, which will perform the actual Sign Recognition. In the following sections, the steps taken for training this network will be extensively illustrated.

2.2 Description of the proposed application

The wide field of study this particular project belongs to is extremely volatile and different technologies are released very often. For this reason the final application developed together with this dissertation will provide a Proof of Concept (PoC), in order to prove that steps towards deaf and hard-of-hearing are becoming to be taken. In the previous chapter, a brief introduction to the main technologies used in the project was given. Now let's concentrate on how they have been used and how they work together. In Figure 2.3 a diagram showing the main components of the application is provided.

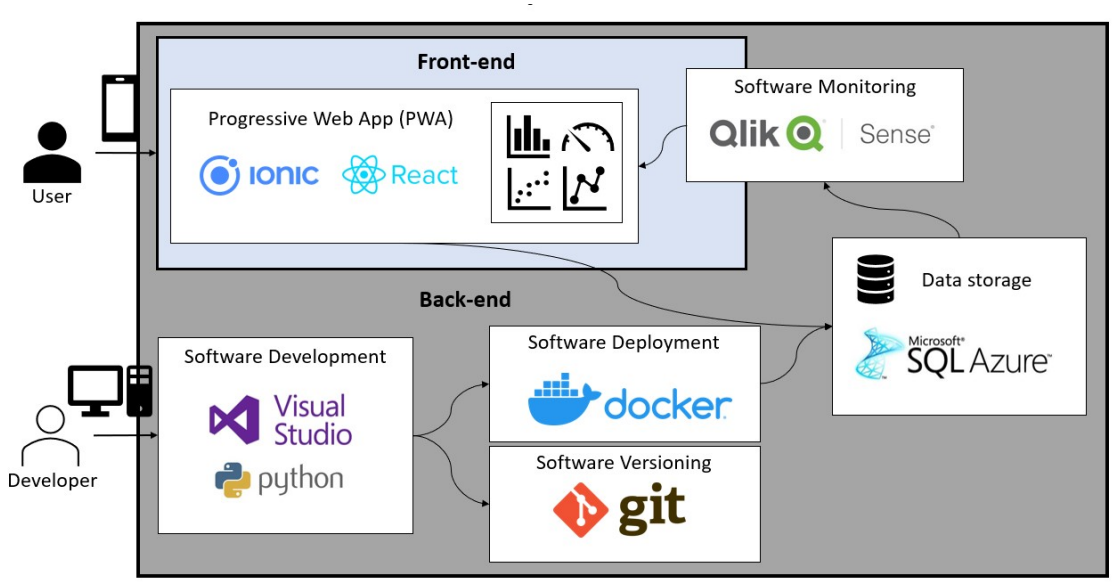


Figure 2.3: The LIS2S architecture diagram

2.2.1 Front-End

As previously mentioned, LIS2S application has been thought as a PWA, specifically taking advantage of the reachability and flexibility of Ionic React framework.

The UI allows for different activities depending on the type of user: general users can record a new video and get the translation of the performed sign, while admins can also insert new sign language videos, to increase the number of samples belonging to the dataset used by the network, and, in addition, they can access a specific dashboard to monitor the performance of the application.

An example of the proposed interface is depicted in Figure 2.4. The different operations that can be executed once the app is used are:

- *Select right-handed/left-handed mode*: LIS is really dependant on the orientation of the executed sign, so the user, before executing any kind of operation, should select whether he is left or right handed; if this is not specified, the algorithm could get confuse and return wrong translations (e.g. "16" instead of "61"), so the other voices of the UI would be not accessible to the user.

- *LIS2Speech*: this option starts a new video translation session; initially, the application will inform the user to place the camera at the correct distance, in order to fit correctly inside the recording space. Subsequently, the user can start the video recording session by clicking the button: a countdown will be shown to the user, so that he can take position and understand when the record has been started. To stop the recording, a specific sign has to be executed (usually, the "silence" sign would work perfectly): this has been decided to

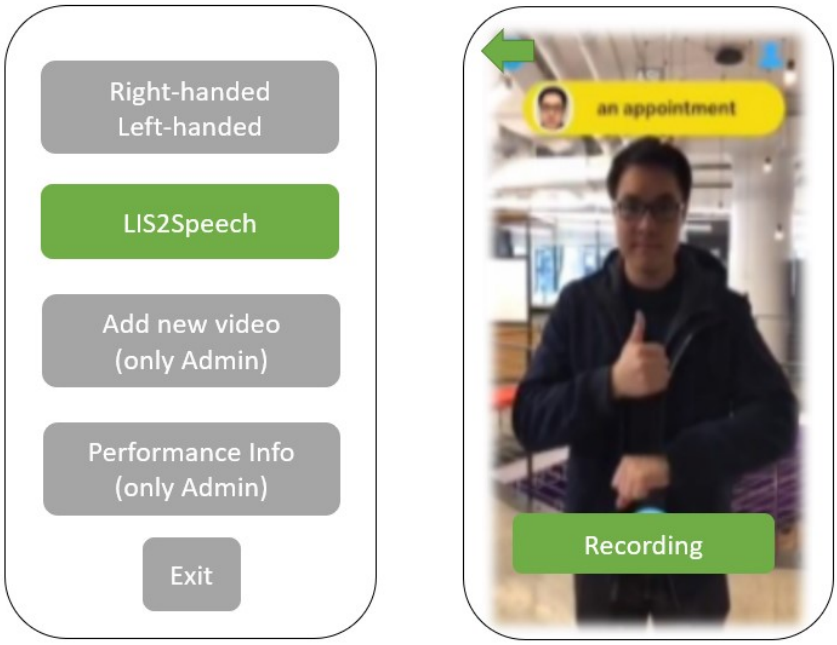


Figure 2.4: Application UI

On the left side, the options that will be available in the main section of the application. On the right side, an example of the UI during video recording.

avoid recording also video chunks in which no sign has been executed, and in addition this would assimilate the new video to the ones already present in the original database (which are executed and then followed by the "silence" sign). Once the video has been processed by the back-end process, composed by the neural network, the proposed translation will be printed on the screen. Finally, the user will be prompted to provide a feedback, if willing so, describing his experience using the application.

- *Add new video*: this option will only be available for admins; in this part of the application the process will be the same as for video translation, but, at the end of the recording, instead of receiving the translation of the video, the admin can insert the meaning of the sign he performed and save or cancel the operation. Then, a pop-up message will appear to confirm the chosen action.

- *Performance info*: also this option will only be available for admins, that can access the dashboard to obtain information about the status and performance of the application; the dashboard of the application is designed using *QlikSense* software, and it shows details about the current dataset (number of videos in the current dataset, dimension of the vocabulary etc.) and the performance of the translation process (accuracy, errors etc.). A mock-up of this dashboard

for the smartphone version is presented in Figure 2.5; instead, a web version of the application dashboard is provided in Figure 2.6.

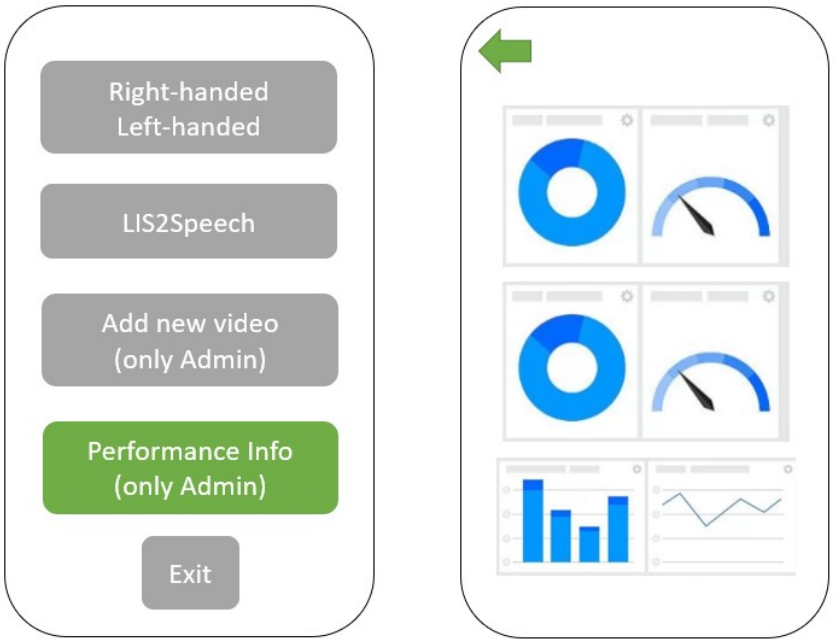


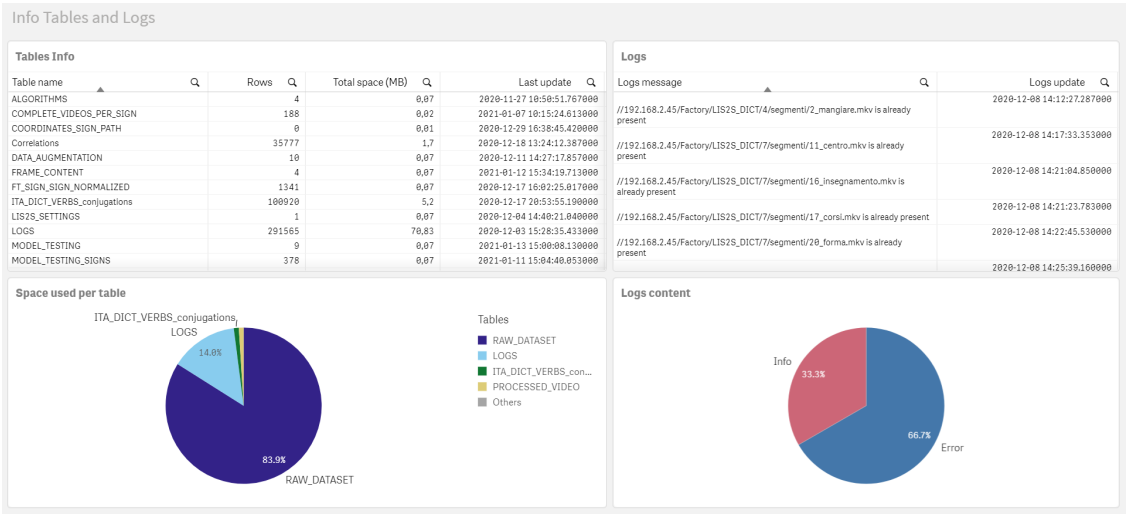
Figure 2.5: UI mock-up for performance monitoring

On the right side, a mock-up of the dashboard inside the application.

2.2.2 Back-End

In the computer world, the "back-end" refers to any part of a website or software program that users do not see. It contrasts with the front-end, which refers to a program's or website's UI. In programming terminology, the back-end is the "data access layer", while the front-end is the "presentation layer".

For what concerns the LIS2S application, the back-end has been developed exploiting the *Python* programming language, its flexibility and the great quantity of modules ready to be used (such as Numpy, OpenCV and so on); in particular, *Visual Studio* was used as Integrated Development Environment (IDE), thanks to its integration with *Git*, which has been used for software versioning. Specifically, the development team has made use of the *GitLab* DevOps platform to manage both software versioning and Continuous Integration/Continuous Deployment (CI/CD). This particular mechanism embodies a culture, a set of operating principles and collection of practices that enable application development teams to deliver code changes more frequently and reliably.



Information about videos and signs

Figure 2.6: QlikSense dashboard

CI/CD

Continuous integration is a coding philosophy and set of practices that drive development teams to implement small changes and check in code to version control repository frequently. Because most modern applications require developing code in different platforms and tools, the team needs a mechanism to integrate and validate its changes. The technical goal of CI is to establish a consisted and automated way to build, package and test applications.

Continuous delivery picks up where CI ends. CD automates the delivery of applications to selected infrastructure environments. Most teams work with multiple environments other than the production, such as development and testing environments, and CD ensures there is an automated way to push code changes to them.

In particular, the software deployment is obtained using Docker; a *Dockerfile* is defined in the project, containing the commands a user could call on the command line to assemble an image, executing several command line instructions in succession. In the Docker image also a *context* is present, containing the set of files at the specified location path or URL.

The GitLab platform allows to create a pipeline for CI/CD and has an integrated Container registry which can contain the images created by the continuous integration. Specifically, the pipeline is implemented using an *YAML* file containing the instructions that need to be executed in order to create the image and save it onto the GitLab Container registry. The list of instructions contained in the YAML file, which is called `gitlab-ci.yml`, can be found in Appendix A. What is

done in practice is: whenever a new commit is received by GitLab, the platform will log into the Container Registry associated with that particular repository, will read the Dockerfile that will be found in the `./LIS2Speech` folder, build the image defined inside it and then push the image on the Container Registry, so that it can be used and deployed on external services (like Kubernetes).

Data storage

To manage the videos collected (both from the A3LIS-147 corpus and selected by the development team), their processing and all the information associated with them, the *Microsoft Azure SQL* combined with a proprietary Orbyta's NAS have been used. The actual videos are stored in the NAS drive, while the information extracted by processing videos (number of signs in the vocabulary, number of videos etc.) has been stored in the Azure database.

Multiple tables where necessary to successfully manage and organize the data needed to perform the neural network training and to monitor the performance of the application. In Figure 2.7 a diagram of the database organization is provided. The videos making part of the used dataset are stored only for reference or for future improvements, since they are not necessary any more once they have been processed and the information have been stored on the Azure database. In this way, the development team can manage a considerably lower amount of data, because, as known, videos are much more relevant in terms of disk space with respect to numbers and strings extracted from videos.

As we can see from Figure 2.7, a great number of tables was necessary to correctly manage the elements which are part of the dataset. In order to make the organization clearer and the application more responsive, using QlikSense it is also possible to reorganize the structure of the Database: by doing so, it is easier to read and the performance are better, because a well done data model is able to fulfill the requested query in a smaller amount of time; in fact, the difference can be seen in Figure 2.8, allowing to better understand the dependencies between the different tables present in the database.

2.3 Pipeline for Sign Recognition

Moving on towards the actual Sign Language recognition, the development team, after an accurate analysis, highlighted the need for four different models. Actually, the developed prototype focuses only on the first two models, which deal with the extraction of skeletal data and the isolated sign recognition; the last two are reported to give indications for future improvements, in order to expand the use cases to which this application could be applied.

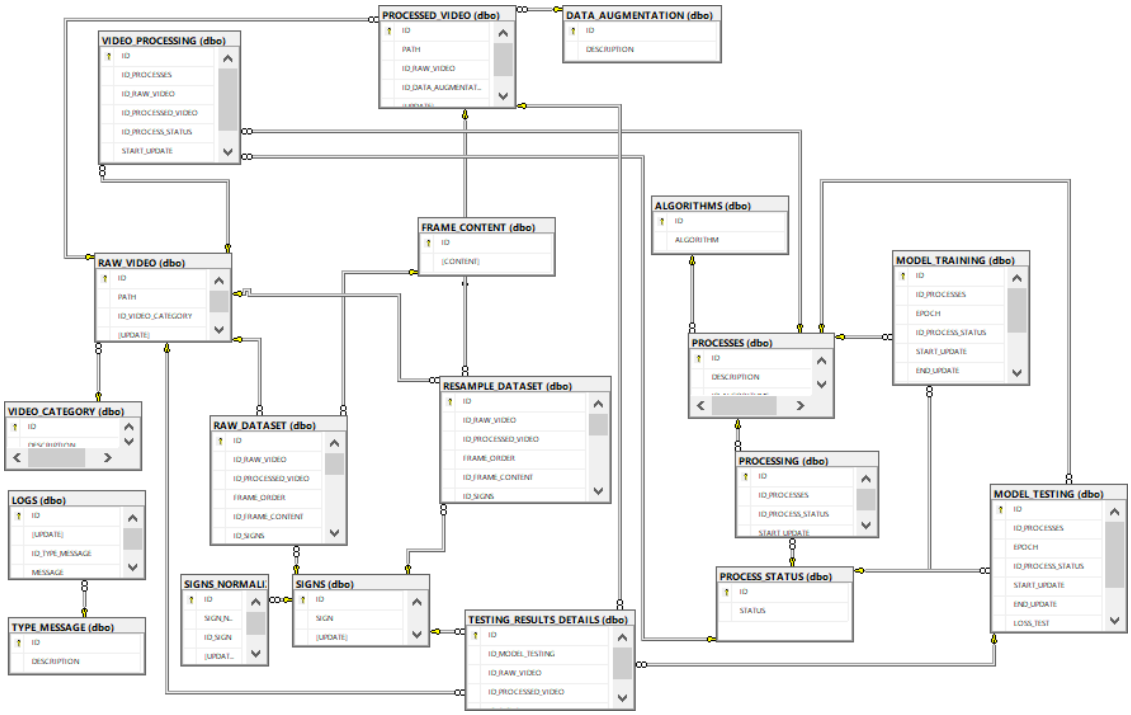


Figure 2.7: Database organization on Microsoft Azure SQL

Figure 2.8: Database organization on QlikSense

2.3.1 From videos to skeletal data

Initially, the idea was to start building a new dataset from scratch exploiting the possibilities offered by the Kinect sensor: it would have allowed to extract skeletal data and depth information from captured videos. Subsequently, it was decided to use the A3LIS-147 dataset as a starting basis and expand it to be future-proof. This decision was due to the reduced time availability to complete this study, together with the difficulties of importing a new Kinect sensor in Italy, since it is not directly available for our market.

However, the willingness to use skeletal data persisted, instead of using videos directly: this was decided in order to reduce the dimensionality of the data to manipulate, and to demonstrate that using only lighter data like skeletal data it is possible to obtain state-of-the-art results. To extract these information from the videos, a first model has been implemented; its main goal was to extract these skeletal data to obtain the coordinates of the principal joints of the human body. In particular, as previously seen in the related works, the most important information for SLR are contained in hands, face and upper body, so the extraction of coordinates mainly focuses on these three areas. Thus, the MediaPipe Python library has been used to extract:

- 3D coordinates of 21 joints for each hand;
- 2D coordinates of 25 joints for the upper body;
- 3D coordinates of 22 joints for the face mesh.

As it can be noticed, only the upper body landmarks are two-dimensional, and this is due to the fact that the Pose solution model [8] is not fully trained to predict depth, but this is something on the roadmap of the MediaPipe team; anyway, the third dimension of upper body joints is zero-padded (filled with zeros) only to obtain three-dimensional matrices that can later be combined.

In total, the number of coordinates extracted is given by the number of landmarks, which is 89, multiplied by 3 dimensions; so in the end, for each frame of a single video, 267 numerical values (features) are extracted. At the beginning of the development, the number of values extracted was much greater, because the number of joints that are detected exploiting the Face Mesh solution of MediaPipe [7] is equal to 468. The development team decided to reduce the number of considered landmarks, since this would have led to a disproportion in the relationship between joints belonging to hands and upper body and joints belonging to the face mesh; in addition, this imbalance could have led the network to focus mainly on the information held in face mesh joints and ignore the broader picture of the situation, since the most important information can be extracted only considering all movements made by the signer.

Finally, only 22 joints were considered out of the initial 468; these joints describe the mouth, eyes and eyebrows inclination and position, and are considered the most relevant since it is still possible to extract meaningful information only by these points, as proved by different studies [37], [38]. An example of the coordinates extracted in this process is provided in Figure 2.9.

Figure 2.9: Example of skeletal data extraction

The sign that is performed in this video is "Torino".

It is important noticing that the precision obtained using this system is comparable, but not at the same level of the accuracy that can be obtained using the Kinect sensor. In fact, the MediaPipe framework has been developed with mobile platforms in mind, so it needs to be extremely portable and power efficient; a trade-off between accuracy and performance was made, but the results are still quite impressive, considering that these solutions are able to run at optimal frame rates even on not so updated smartphones. In addition, also browsers are supported, so this is really important in terms of reachability.

Nevertheless, the tracking and detecting accuracy can be tweaked for hands, face mesh and pose. Thanks to implementation choices, it is possible to set a *minimum detection confidence* and also a *minimum tracking confidence*: these are float values, between 0.0 and 1.0, which specify, respectively, the minimum value from the hand, face or pose detection model for the detection to be considered successful, and the minimum value from the landmark-tracking model for the hand, face or pose landmarks to be considered tracked successfully. Setting these values to higher values can increase the robustness of the solution, avoiding errors in detection and tracking, but at the cost of higher latency, so worse performance. The default values are 0.5 for both; during our processing, instead, the minimum detection confidence was set to 0.7 to avoid errors detecting hands, face or upper body joints and obtain a more reliable system.

Another important point to highlight is that the Hands MediaPipe solution offers a *multi handedness* option which allows to keep track of both hands, understanding, in the meantime, which one is the right hand and the left hand. This is extremely important for sign recognition, since the meaning of a gesture may change according to how it is performed. Specifying the multi-handedness option, the solution returns, for each frame, the predicted label for the detected hand (i.e., if it is a left or right hand) together with the score, which is the estimated probability of the predicted handedness: this value is always greater than or equal to 0.5, but sometimes the model may wrongly detect two left hands or two right hands; this is not possible, since the processed videos always contain a single subject, and this means that the detected hands should always be right and left.

In order to overcome this problem, a simple algorithm has been developed so that, for each frame, it is always guaranteed to obtain data for left and right hands. The algorithm is based on the score of each hand: if two equal hands are detected, the one with the highest score is supposed to be the correct one, while the remaining one will be considered as the opposite of the previous one. Finally, if no hand has been detected, or if only a single hand has been detected, the algorithm will accordingly return some coordinates which are averaged from the coordinates of previous time frames. A possible case of study could be considering to return zero-filled arrays in case one or both hands have not been detected.

Once all these information are collected, the coordinates of hands, face and pose are stacked together in a single array, and this element, which represents the coordinates extracted from the i^th frame, is added to the numpy array containing all the frames relative to the actual processed video. Summing up, if a video is composed of t frames, the associated array would have shape $(t, 267)$, where 267 is the number of features extracted for a single frame. These information, together with the sign performed in the video, will be stored in the Azure database, in the RAW_DATASET table to which t rows will be added.

In the next section, these features will be fed to the neural network, which is the core of the SLR. The model of the network will be discussed and illustrated.

2.3.2 From skeletal data to gloss

In the proposed architecture, after the features extraction it is possible to find the actual neural network, which is in charge of understanding the temporal information held inside features it is fed with, and return the predicted sign.

To do so, first let introduce Recurrent Neural Networks: these are particular networks which are designed to take a series of input with no predetermined limit on size; in this way, the input is considered as a series of information, which can held additional meaning to what the network is training on. A single input item from the series is related to others and likely will have an influence on its neighbours; RNNs are able to capture this relationship across inputs meaningfully. In fact, they are able to "remember" the past and take decisions based on what they have learnt from the past. In particular, they can remember the information learnt from prior input(s) while generating output(s). RNNs can take one or more input vectors and produce one or more output vectors, and the output(s) will be influenced not just by weights applied on inputs like a regular NN, but also by a hidden state vector representing the context based on prior input(s)/output(s). In other words, the same input could produce a different output depending on the previous inputs in the series.

In addition, RNNs can also be bidirectional: this means that they can extract information from past and also future temporal information held in the series. For example, in speech recognition and handwriting recognition tasks, where there could be considerable ambiguity given just one part of the input, it is often needed to know what is coming next to better understand the context and detect the present.

In the literature, there are different works that use RNNs for action and gesture recognition, for example, in [39] Shahroudy *et al.* showed the power of recurrent architectures and long short-term memory (LSTM) units [40] for large-scale gesture recognition; Liu *et al.* [41] incorporated the spatio-temporal and contextual dependencies to recognize actions from 3D skeletons. In contrast, the model proposed for the sign recognition task is based on GRU (Gated Recurrent Units) [10]. GRUs have been preferred because they are faster to train and produce better results. The chosen method, in addition, is designed to be general and not specific to a particular device, gesture modality or feature representation.

Previously, it was outlined the importance of the information held inside temporal sequences; anyway, the sub-parts of a temporal sequence may not all be equally important: some subsequences may be more useful for the task at hand than others. Thus, it is considered useful to learn a representation that can identify these important subsequences and leverage them to deal with the subject matter. This is the key intuition behind the attention model [42]. Even though the attention model was initially proposed for neural machine translation, it has been adapted to the task of gesture, action and sign recognition [43].

For the network architecture, we took inspiration from DeepGru [9], VGG-16

[44] and the attention models [42]. The model, depicted in Figure 2.10, is made of three main components: an encoder network, the attention model and two fully-connected (FC) layers fed to softmax producing the probability distribution of the class labels.

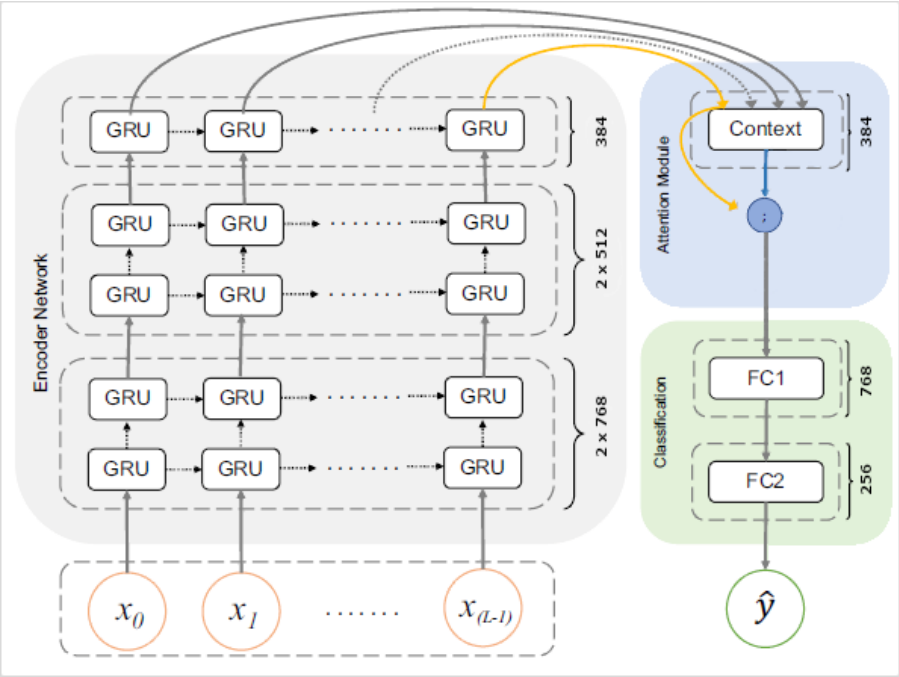


Figure 2.10: LIS2S Neural Network

The proposed recurrent model consists of an *encoder network* of stacked gated recurrent units (GRU), the *attention module* and the *classification* layers. The input $\mathbf{x} = (x_0, x_1, \ldots, x_{L-1})$ is a sequence of vector data of fixed length (100 time frames) and the output is the predicted class label $\hat{y}$. The number of the hidden units for each layer is displayed next to every component.

Input Data

The input data for the network is obtained from the features extracted by the previous model, represented as a temporal sequence of the underlying sign data (3D joint positions). At the time step t, the input data is the row vector $x_t^\intercal \in \mathbb{R}^N$, where N is the dimensionality of the feature vector. Thus, the input data of the entire temporal sequence of a single gesture sample is the matrix $\mathbf{x} \in \mathbb{R}^{L \times N}$, where L is the length of the sequence of time steps.

The dimensionality N depends on the device that generated the data and how the data was represented. In the previous section it was described how the extracted data had a dimensionality $(t, 267)$, since the number of features of each time frame is equal to 267; anyway, this is a fixed value only for this particular use case: if

the features were extracted with a different mechanism, the model would have been agnostic about to the input representation, and would have accepted any kind of dimensionality. Later on, it will be shown how the number of features fed to the model is smaller than the actual 267, since this high value of features would bring with it a whole series of problem, that will be tackled in the next chapter.

In addition, note that various input example sequences could have different number of time steps. During the development of the model it was decided to perform the *subsampling* of the input data, so that each sample would have the same number of time steps, which is 100. The original model offered the possibility to use the temporal sequences as-is, without subsampling or clipping, but the development team decided to perform subsample in order to equalize the samples which compose the dataset. When training on mini-batches, so, the i^{th} mini-batch is represented as the tensor $\mathbf{X}_i \in \mathbb{R}^{B \times L \times N}$, where B is the mini-batch size and L is set equal to 100.

Encoder Network

The encoder network in the depicted model is fed with data from the training samples and serves as the feature extractor. The encoder network consists of five stacked unidirectional GRUs. Although LSTMs units are more prevalent in the literature, the development team decided to use GRUs since they have a smaller number of parameters, so these units are simpler to utilize, generally faster to train and less prone to overfitting. At time step t, given an input vector x_t and the hidden state vector of the previous time step h_{t-1}, a GRU computes h_t, the hidden output at time step t, as $h_t = \Gamma(x_t, h_{(t-1)})$ using the following transition equations:

$$
\begin{aligned}
r_t &= \sigma\Big(\Big(W_x^r x_t + b_x^r\Big) + \Big(W_h^r h_{(t-1)} + b_h^r\Big)\Big) \\
u_t &= \sigma\Big(\Big(W_x u x_t + b_x^u\Big) + \Big(W_h^u h_{(t-1)} + b_h^u\Big)\Big) \\
c_t &= \tanh\Big(\Big(W_x^c x_t + b_x^c\Big) + r_t\Big(W_h^c h_{(t-1)} + b_h^c\Big)\Big) \\
h_t &= u_t \circ h_{(t-1)} + \Big(1 - u_t\Big) \circ c_t
\end{aligned}
\tag{2.1}
$$

where σ is the sigmoid function, $\circ$ denotes the Hadamard product, r_t, u_t and c_t are reset, update and candidate gates respectively and W_p^q and b_p^q are the trainable weights and biases. In the encoder network, h_0 of all the GRUs are initialized to zero.

Given a gesture example $\mathbf{x} \in \mathbb{R}^{L \times N}$, the encoder network uses Equation 2.1 to output $\bar{h} \in \mathbb{R}^{100 \times 384}$, where $\bar{h}$ is the result of the concatenation $\bar{h} = [h_0; h_1; \cdots; h_{L-1}]$. This output, which is a compact encoding of the input matrix $\mathbf{x}$, is then fed to the attention module.

Attention Module

The output of the the encoder network, which is a compressed representation of the input sign sample, can provide a plausible set of features to perform classification. During the development of the network, it was decided also to refine this set of features by extracting the most informative parts of the sequence using the attention model. A proper adaptation of the global attention model [42], suitable for the sign recognition task, has been developed.

Given all the hidden states $\bar{h}$ of the encoder network, the attention module computes the attentional context vector $c \in \mathbb{R}^{384}$ using the trainable parameters W_c as:

$$c = \bar{h}\left(\frac{\exp(h_{L-1}^{\intercal} W_c \bar{h})}{\sum_{t=0}^{L-1} \exp(h_{L-1}^{\intercal} W_c h_t)}\right) \tag{2.2}$$

The hidden states of the encoder network are used only to compute the attentional context vector, as can be seen in Equation 2.2. The main component in the context computation and attentional output is the the hidden state of the last time step h_{L-1} of the encoder network (the yellow arrow in Figure 2.10); this is because h_{L-1} can capture a great amount of information from the entire gesture sample sequence. The concatenation $[c; h_{L-1}]$ has been used to form the contextual feature vector and perform classification; in the original project, the inputs to the network could have arbitrary lengths, while in the analysed case the inputs have been resampled so that they all have the same temporal length of 100 time frames. In this way, the amount of information that is captured by h_{L-1} should not differ among sequences which may have different length, thus the model is not considered susceptible to variations in sequence lengths. Summing up, the attention model relies only on the hidden state of the last time step h_{L-1}, which reduces complexity.

Classification

The final layers of the model are composed by two FC layers (F_1 and F_2) with ReLU activation functions that take the attention module's output and produce the probability distribution of the class labels using a softmax classifier:

$$\hat{y} = \text{softmax}\left(\mathrm{F}_2\left(\text{ReLU}\left(\mathrm{F}_1\left(o_{attn}\right)\right)\right)\right) \tag{2.3}$$

The batch normalization [45] followed by dropout [46] are used on the input of both F_1 and F_2 in Equation Equation 2.3. During training, the cross-entropy loss in minimized to reduce the difference between the predicted class labels $\hat{y}$ and the ground truth labels y.

More implementation details are discussed shortly in chapter 3.

2.3.3 From glosses to sentences

The network depicted in the previous section, still being very interesting, has some limitations; the most important one is that the input must be segmented, although adding support for unsegmented data is straightforward, requiring a change in the training protocol as demonstrated in [47].

In general, this work will not try to solve the problem of translating sign language sentences, but, as previously written, is focused on isolated sign language recognition. Nevertheless, during the development of the project, another model has been considered, to effectively switch from isolated to continuous sign language recognition. This model will hopefully be developed in the future, or can be an inspiration for future works.

Specifically, what the third model should perform is the sign segmentation of sign language sentences; defining when a sign starts and finishes is a quite complicated task, considering the scarcity of corpus for the Italian Sign Language and also the complexity of the sign language itself. Detecting when a sign ends and another one starts is not a trivial operation, but there are some solutions that may be considered and that can be useful to solve this task.

A first solution could be to actually divide a given video, containing a sign language sentence, into subvideos which are made of a single sign; in order to understand when a sign ends and another one starts, different techniques can be used. For instance, in [48] the greedy similarity measure is used to segment long spatial-temporal video sequences; first, a principal curve of the motion region along the frames of a video sequence is constructed to represent the trajectory; then from the constructed principal curves, Hidden Markov Models (HMMs) are applied to model gestures, or signs in this case.

Another solution to overcome this issue and to get a sort of real-time, step-by-step classification, could be the usage of non-overlapping short time windows. The recognition model would emit a classification of the gesture data inside each window. Finally, the use of an objective function such as the Connectionist Temporal Classification Loss (CTC) [49] could allow an alignment of the classes obtained from the time windows with the desired (actual) ones.

Nevertheless, these solutions are to be intended as possible improvements for this work, and do will be briefly summarized in the conclusions.

2.3.4 From LIS to proper Italian

In this section, another model will be quickly described. This model is considered not so essential, but can be very helpful during translation from LIS to Italian. The grammar and the structure of sentences in LIS and in Italian are strongly different.

For instance, lets consider a simple sentence which is present in the constructed LIS2S dataset: in LIS, in order to say "*Oggi c'è il sole*", which translates to English "*Today it is sunny*", the signer will perform, in sequence, the signs *oggi sole c'è*,

which translates to English *today sunny it is*. As evident, the order of subjects and verbs is different from correct Italian, because, generally, in LIS the verbs are placed at the end of the sentence.

Another example could be the expression "*Ciao a tutti, il mio nome è Chiara*", which translates to English "*Hi everybody, my name is Chiara*": in this case, the signer will perform, in sequence, the signs *ciao tutti nome mio Chiara*, which translates to English *hello everybody name my Chiara*. Moreover, here it is possible to notice how the structure of the sequence is different, with the possessive adjective immediately following the subject it is referred to.

In order to make the translation more readable by non-deaf users, the model that has been thought as the final one should perform a type of rephrasing, manipulating the raw translation and converting it into a correct Italian sentence. Furthermore, this model is not faced in this work, but just reported for completeness and to be considered as a starting point for further improvements of this application.

This kind of operations should fall back in the Natural Language Processing (NLP) field [50], which has very ancient roots in Computer Science history, starting from Alan Turing back in 1950s up to the present days. To return complete and meaningful translation, the model should be able to understand the language syntax and structure.

In NLP, there is an interesting technique which is called Part of Speech (PoS) Tagging: words can be grouped into classes referred to as PoS or morphological classes (noun, verb, adjective, preposition, adverb, conjunction etc.); the word PoS provides crucial information to determine the roles of the word itself and of the words close to it in the sentence: knowing if a word is a personal pronoun or a possessive pronoun allows a more accurate selection of the most probable words that appear in its neighbourhood (the syntactic rules are often based on the PoS words).

In addition, language models can be very useful to solve the addressed problem, since they represent the probability distribution over sequences of words: given a sequence of length m, the language model assigns $P(w_1, \ldots, w_m)$ to the whole sequence. The language model provides context to distinguish between words and phrases that sound similar; estimating the relative likelihood of different phrases is useful in many NLP applications, especially those that generate text as an output.

For what concerns this work, it would be useful to find a simple text corpus for the Italian language, better if it has been tagged with PoS tags; exploiting this corpus, it would be possible to learn a language model, either on the basis of the words or on the basis of words and PoS tags. Once the raw translation has been returned by the previous model, the actual one would generate all possible sequences of the target words and use the language model to compute the probabilities of all those sequences. In the end, the model would pick the sequence with the highest probability, which hopefully would have the correct grammar order.

Finally, another hint could be the possibility to add conjunctions and prepositions to improve the quality and correctness of the output sentence. All these possible improvements will be recalled in the conclusions of this dissertation, to give some ideas for next works starting from this project.

This chapter began by describing the dataset and database used in this application; it went on to report the different elements which are part of the project, the front-end, the back-end and the different models which are vital for the correct working of the application itself. In the next chapter more precise implementation details will be provided, together with an extensive overview of the experiments that was conducted and the results that have been obtained.

Chapter 3

Experiments and results

The following part of this dissertation moves on to describe in greater detail the experiments that have been performed. Firstly, the implementation details of the adopted solution will be listed; then, there follows an overview about the different techniques utilized to select the most useful and meaningful set of features coming from the skeletal data. In the subsequent sections, the problem of overfitting will be discussed and investigated. Finally, the training and testing results of the network involved in the LIS2S application will be disclosed.

3.1 Implementation details

The network structure which was depicted in the previous chapter has been implemented using the PyTorch [51] framework. To understand which were the best possible hyper-parameters to use on the LIS2S dataset, the *Optuna*[1] hyper-parameter optimization framework has been used. Optuna is framework agnostic, so it can be used with any machine learning or deep learning framework, such as TensorFlow or Keras, and needs only little effort to work: an `objective` function needs to be defined, and that will be the function to be optimized; then it is possible to suggest the hyper-parameter values to try using the `trial` object. Finally, a `study` object must be created and the `optimize` method needs to be invoked on it, defining the number of trials to execute. The code which has been produced for LIS2S application can be found in Appendix B: the objective function is called `hyperparams_tuning` and the suggested hyper-parameter values are explained later on.

For what concerns the purposes of this dissertation, the hyper-parameters that have been tuned are:

[1] `www.optuna.org`

- The *learning rate*, which in an optimization algorithm determines the step size at each iteration while moving toward a minimum of a loss function (the loss function measures how wrong the final predictions are). Since it influences to what extent newly acquired information overrides old information, it metaphorically represents the speed at which a machine learning model "learns". In setting the learning rate, there is a trade-off between the rate of convergence and overshooting. While the descent direction is usually determined from the gradient of the loss function, the learning rate determines how big a step is taken in that direction. A too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum. During the hyper-parameters tuning, the learning rate is sampled from $[1 \times 10^{-4}, 1 \times 10^{-2}[$ in the log domain, as it can be seen in Appendix B at row 13.

- The *optimizer*, which shape the model into its most accurate possible form by tweaking and changing the parameters (weights) of the model itself in order to minimize the loss function. So, the loss function specify to the optimizer when it is moving in the right or wrong direction. During the tuning of hyper-parameters, the considered optimizers are the Stochastic Gradient Descent (SGD) [52], the RMSProp [53] and the Adam [54] optimizer, as it can bee seen in Appendix B at row 15.

- The *weight decay*, which is an additional term in the weight update rule that causes the weights to exponentially decay to zero, if no other update is scheduled, in order to limit the complexity of the network. This argument will be better discussed later on, talking about overfitting in section 3.2. As described in Appendix B at row 18, the weight decay value is sampled from $[0, 1 \times 10^{-1}[$ in the log domain.

- The *model* itself has been evaluated with the same model structure, but with LSTM cells instead of GRU ones.

At the end of the hyper-parameter optimization process, the best initial learning rate was found to be 5×10^{-4}, the best optimizer is Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$), the selected model was based on GRU cells and the weight decay was set to 2×10^{-3}.

Another important parameter to set is the *mini-batch size*: using the gradient descent optimization algorithm, the current state of the model will be used to make predictions, compare them to the expected values and use the difference as an estimate of the error gradient, which will be then used to update the model weights; the process is then repeated. The error gradient is a statistical estimate: the more training examples are used in the estimate, the more accurate the estimate will be and the more likely that the weights of the network will be adjusted in a way that will improve the performance of the model. This comes at the cost of computation and time, since the model needs to make many predictions before the

estimate can be calculated. On the other hand, using fewer examples results in a less accurate estimate of the error gradient, which is highly dependent on the specific training examples used, so the results can lead to a noisy estimate and noisy updates.

The number of training examples used in the estimate of the error gradient is called the *batch size*: a batch size of 32 means that 32 samples from the training dataset will be used to estimate the error gradient before the model weights are updated. Historically, there are three distinctions about the training algorithm:

- It is called *Batch Gradient Descent* if the batch size is set to the total number of examples in the training set.
- It can be the *Stochastic Gradient Descent* if the batch size is composed by one single sample.
- Finally, it may be the *Mini-batch Gradient Descent* if the batch size is set to more than one and less than the total number of examples in the training set.

Usually, smaller batch sizes are used because they are noisy and offer a regularizing effect and lower generalization error, and because the smaller size makes it easier to fit one batch worth of training data in memory (i.e. when using a GPU). For this project, the batch size is set to 64. This is also due to the available amount of memory on the GPU used for training, which is limited. Training has been done on a machine equipped with an NVIDIA GeForce GTX 1650 Super GPU, having 4 GB of VRAM, AMD Ryzen 5 2600X processor and 16 GB of RAM.

Finally, the training of the network is maintained for 150 epochs, but *early stopping* mechanism has been implemented and will be examined in depth later on when talking about overfitting. The number of epochs defines the number of times the learning algorithm will work through the training dataset: on a single epoch, the samples belonging to a particular batch will be used by the optimization algorithm to update the internal model parameters and minimize the loss function. The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error of the model has been sufficiently minimized.

3.2 Overfitting and Regularization

Supervised machine learning algorithms, like classification, can be considered as models trying to approximate a target function f that maps input variables X to an output variable Y: $Y = f(X)$. A critical factor when learning the target function from the training data is how well the model generalizes to new data.

In machine learning, the inferring of a target function from training data is described as inductive learning. Supervised machine learning models try to learn

general concepts from specific examples, and this is exactly the induction mechanism. It is different from deduction, which seeks to learn specific concepts from general rules. Generalization refers to how well the concepts learnt by a machine learning model apply to examples not yet seen by the model. The goal of a good machine learning model is to successfully generalize from the training data to any data from the problem domain. This allows to make future predictions on data the model has never seen. When talking about how well a machine learning model learns and generalizes to new data, *overfitting* and *underfitting* come into play: poor performance of machine learning algorithms is typically due to these phenomena. Examples of overfitting, underfitting and ideal models are provided in Figure 3.1.

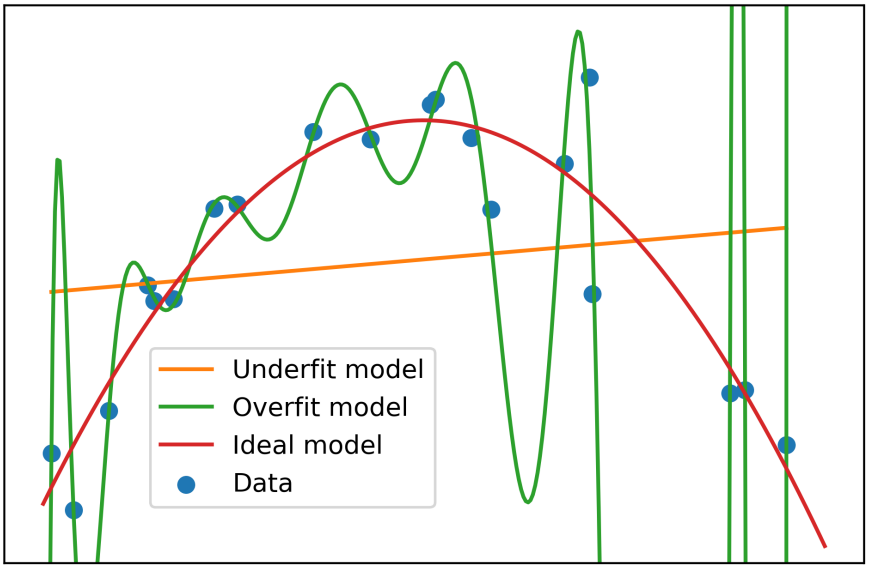


Figure 3.1: Overfitting, underfitting and ideal models

While the green line best follows the training data, it is too dependent on that data and it is likely to have a higher error rate on unseen data, compared to the red line. The orange line, instead, is not able to model the data in the right way, and will have poor performance both on training and test set.

Overfitting concerns neural networks that model the training data too well. It happens when a model learns useless details and noise held in the training data. Usually, this will negatively impact the performance of the model on unseen data. The noise or random fluctuations in the training data are learnt as something relevant for the model. These concepts do not apply to new data and negatively impact the model ability to generalize. Overfitting is more likely with non-parametric and non-linear models that have more flexibility when learning a target function. As such, there are different techniques to limit and constraint how many details the model learns.

Underfitting, instead, refers to a model that is not able to correctly model the training data, and in addition cannot generalize to new data. An underfit machine learning model is not a suitable model: it will have poor performance on the training data, and, as a consequence, it will perform even more poorly on the test set. Underfitting is often not discussed as it is easy to detect, given a good performance metric. As a remedy, it is possible to try alternative machine learning algorithms or modify the existing models to create more complex ones.

Ideally, the best model lays at the sweet spot between underfitting and overfitting. To understand the real performance of the model, it is possible to check, over different epochs, both the accuracy and loss over the training and the test set. Over time, as the algorithm learns, the error on the training data should decrease, and so does the error on the test set. If the model trains for too long, the performance on the training dataset will increase indefinitely, because the model is learning all irrelevant details and noise in the training dataset. At the same time, the error on the test set will grow up, while the model's generalization skills will get worse. The best model can be found just before the error on the test dataset starts to increase; at that time the model has good skill on both the training set and the unseen test set.

3.2.1 K-Fold Cross Validation

One of the most used techniques used to tackle overfitting is the *K-Fold Cross Validation*: this mechanism can allow to train and test the model k times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data. The procedure has a single parameter called k that refers to the number of groups that a given dataset is to be split into. This technique has been used in the development of the LIS2S application, setting the value of k to 5, so defining 5 different folds. The general procedure works as follows:

- Shuffle the dataset randomly;
- Split the dataset into k groups;
- For each unique group i:
  - Take the i^{th} group as the validation set
  - Take the remaining groups as training set
  - Fit the model in the training set and evaluate it over the validation set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the validation set 1 time and used to train the model $k - 1$ times. It is also important that any preparation of the data prior to fitting the model will occur on the CV-assigned training dataset within the loop rather than on the broader dataset. A failure to perform these operations within the loop may result in data leakage and an optimistic estimate of the model skill. The results of a K-Fold Cross Validation run are often summarized with the mean of the model skill scores.

3.2.2 Avoid Overfitting

The goal of neural networks is to obtain a final model capable of gaining good performance both on the data that have been used to train it, which compose the training set, and on the unseen data on which the model will be tested to make predictions. The ability to perform well on previously unobserved inputs is called *generalization*. The model needs to learn from known examples, extracting meaningful information that can help it generalize to new samples in the future. There are different methods, like train-test split or k-fold cross validation, that can be used to estimate how the model is able to generalize to new data.

Learning and generalizing to new cases is not trivial: if the learning phase is not performed correctly, the model will have poor performance on the training dataset and on new data, so it will underfit the problem. On the other hand, if the model learns *too much* it will perform well on the training dataset but poorly on new data, and this means that it will overfit. In both cases, the model is not able to generalize well. To do so, the system needs to be sufficiently powerful to approximate the target function. Usually, underfitting can be addressed by increasing the capacity of the model, which means improving the ability of the model to fit a larger set of functions: change the architecture of the model, add more layers or even more nodes to layers.

While an underfit model is really easy to be addressed, it is more common to have an overfit model. A model like this is easily recognizable by monitoring the performance of the model during training. The most effective mechanism consists in evaluating the model on both the training dataset and an holdout validation dataset. An easy way to check if the model is overfitting is to plot the accuracy or loss curves over these datasets, because they are good indicators of the model performance. An example of these curves for an overfitting model is provided in Figure 3.2.

Typically, there are two main approaches to an overfitting model: training the network with more examples, if possible, or changing the complexity of the network. An essential advantage of deep neural networks is that their performance improves by using larger datasets. A model with a near-infinite number of examples will

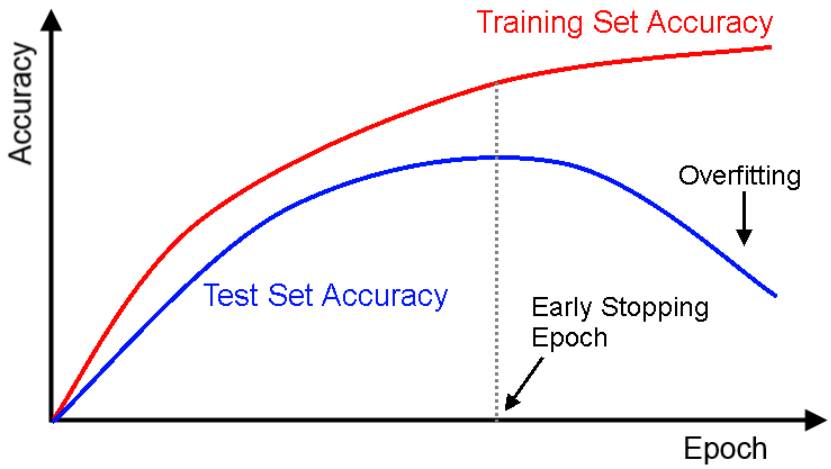


Figure 3.2: Accuracies over epochs for overfitting model

The red line specifies the training set accuracy, while the blue one points out the test set accuracy. As evident, over a certain number of epochs, the model performance on the test set starts to degrade and the error, instead, increase. The Early Stopping Epoch is the epoch in which the model get the best performance on the test set, so the training can be stopped.

eventually stabilize in terms of what the capacity of the network is capable of learning.

Focusing on the complexity of the network, a model can overfit a training dataset because it is powerful enough to do so. To reduce the probability of the model overfitting the training dataset, the capacity of the model can be diminished. The complexity of a neural network model defines its capacity, and it is delineated by the nodes and layers composing the model, but also by the parameters (weights) which are part of these nodes. Therefore, to limit overfitting, the complexity of the neural network can be reduced in two ways: changing the network structure (number of weights) or changing the network parameters (values of weights).

The structure of the model can be tuned using a grid search, trying different number of nodes and/or layers to reduce or remove overfitting. Alternately, nodes can be removed until the model achieves suitable performance on a validation dataset. However, the most common mechanism consists in ensuring that the parameters (weights) of the model remain small. This will lead to a less complex and, in turn, a more stable model that is less affected by statistical fluctuations in the input data.

Techniques that try to reduce overfitting (and the generalization error) by keeping network weights small are known as *regularization methods*.

Weight Decay

The simplest and perhaps most common regularization method is to add a penalty to the loss function in proportion to the size of the weights in the model. This is the *weight decay* which has been introduced in the previous sections: it consists in penalizing the model during training based on the magnitude of the weights. This will encourage the model to map the inputs to the outputs of the training dataset in such a way that the weights of the model are kept small.

For instance, if the loss function is $L(\mathbf{w})$, where $\mathbf{w}$ are the weights of our model, the gradient descent mechanism specifies how to modify these weights in the direction of steepest descent in L:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} \tag{3.1}$$

where η is the learning rate, w_i is the i^th weight of the model and $\leftarrow$ indicates the assignment statement. In general, if the learning rate is too big, it follows also a large modification of the weights $\mathbf{w}$. To effectively limit the number of free parameters in the model, it is possible to apply *regularization* to the cost function. Essentially, the model becomes too much complex and is not able to correctly classify unseen data. To apply regularization, usually a zero mean Gaussian prior is introduced over the weights, and this is equivalent to changing the loss function to $\tilde{E}(\mathbf{w}) = E(\mathbf{w})\frac{\lambda}{2}\mathbf{w}^2$. In practice, this penalizes large weights and effectively limits the freedom of the model. The regularization parameter λ determines the trade-off between the original loss function L and large weight penalization. Applying gradient descent to this new loss function results in:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} - \eta\lambda w_i \tag{3.2}$$

where the new term $-\eta\lambda w_i$, coming from the regularization, causes the weights to decay in proportion to their size. This approach has proven to be very effective for decades both for simpler linear models and neural networks.

Dropout

As previously introduced, a large neural network can overfit when they are trained on relatively small datasets. In this case, the model will learn the intrinsic statistical noise present in the training data, resulting in poor performance when the model is evaluated on new data. To reduce overfitting, once a well-defined architecture has been chosen, an effective method could be to consider different neural networks, obtained by a combination of the nodes and layers present in the initial structure, and train them on the same dataset . Then, the average performance of the starting model would be given by the predictions mean from each model.

This is not feasible in practice and can be approximated using a small collection of different models, called an *ensemble*. Even with the ensemble approximation, anyway, multiple models need to be fit and stored, and this can be a problem if the models are large, requiring days or weeks to train and tune.

Dropout is a regularization method by which the training of a large number of neural networks with different architectures is approximated. During training, some number of layer outputs are randomly ignored or "*dropped out*". An example can be seen in Figure 3.3. In this way, the layer will look like and will be treated like a layer with a different number of nodes and connectivity to the previous layer.

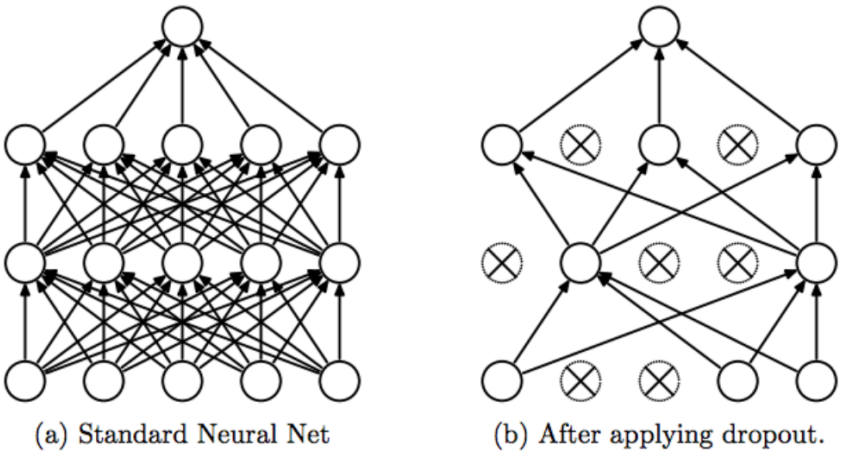


Figure 3.3: Dropout example

The mechanism of Dropout has the effect of making the training process noisy, forcing network layers to collaborate to correct mistakes coming from previous layers, in turn making the model more robust. Since the outputs of a layer under dropout are randomly reduced, this leads to a consequent reduction of the network capacity during training.

Dropout can be applied independently to each layer inside a neural network. This means that a new hyper-parameter is introduced: it specifies the probability of a layer node to be dropped out. The most common value is a probability of 0.5, which has been used also for this project. During the testing phase, Dropout is not used, so that the network can be exploited to its full potential. Like other regularization methods, dropout is more effective when there is a limited amount of training data, such as the case described in this dissertation.

Early Stopping

Another important aspect in training neural networks concerns how long to train them. If the model is not trained enough, then it will underfit the train and the test sets. Instead, training it too much means that the model will overfit the training

dataset and perform poorly on the test set. The sweet spot in this case consists in training the model on the training set and stopping the process whenever the performance on the validation dataset starts to degrade. This simple, effective, and largely used approach is called *early stopping*.

After each epoch, the model is evaluated on a holdout validation dataset. If the performance of the model on the validation set starts to degrade (e.g., loss increases or accuracy decreases), then the training process is stopped. At the time of stop, the model will be saved and then used for final testing, because it should have good generalization performance.

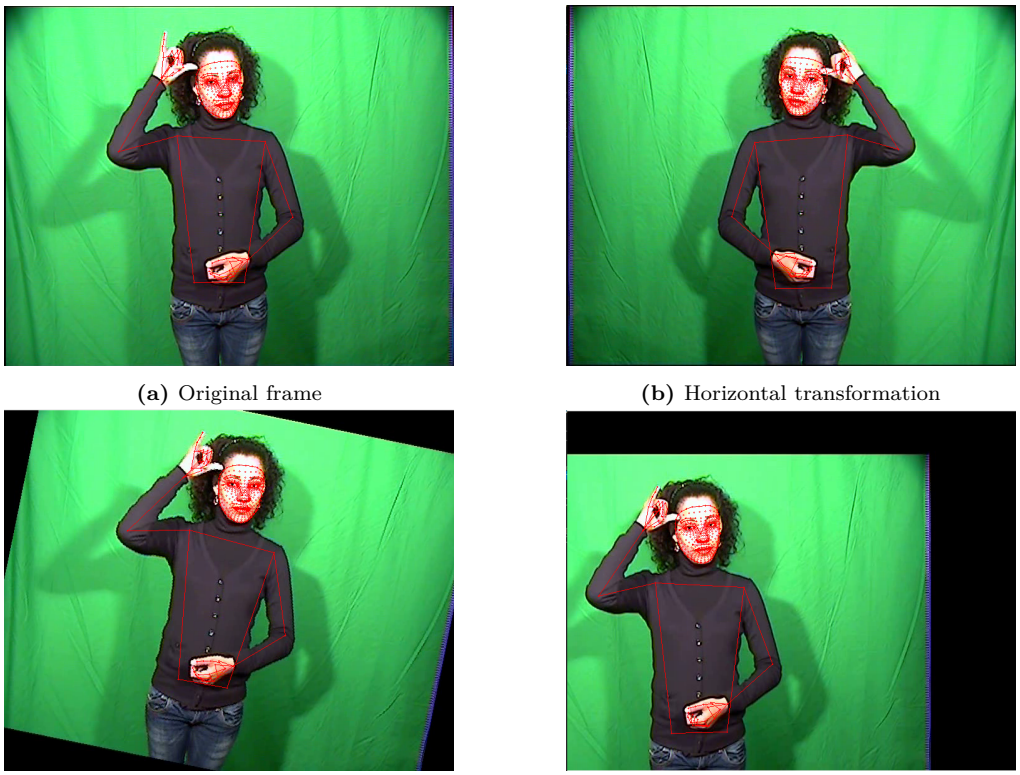
To implement the early stopping mechanism, a particular metric needs to be monitored during training and a trigger for stopping the training process must be defined. Once a certain threshold is reached, or once the performance on the validation set decreases with respect to the previous epochs, the training is stopped. In this project, the metrics that can trigger the early stopping are the accuracy and loss on the validation set: if the loss does not decrease or if the accuracy does not improve for a certain number of epochs, then the training is stopped.

Data Augmentation

The last method adopted in this project to tackle the overfitting problem is the Data Augmentation technique. Previously, it has been said that there are two approaches to face an overfitting model: the previous sections explained the more common methods to reduce or limit the complexity of the network. Data Augmentation, instead, focuses on the other approach, which consists on training the network with more examples. It can be used to artificially expand the size of a training dataset by creating modified versions of images or videos in the dataset.

Training deep learning neural network models on more data can result in more skilful models, and the augmentation techniques can create variations of the videos that can improve the ability of the fit models to generalize what they have learned to new samples. Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. For videos the argument is more complicated, but substantially what has been done for this project is to apply the techniques available for image data augmentation on the different frames of the videos in the dataset.

Going more into the details, four different data augmentation transformations have been applied: random rotate, translate, centre crop and horizontal flip. An example of these different techniques is available in Figure 3.4. The intent is to expand the training dataset with new, plausible examples. This means variations of the training set images that are likely to be seen by the model. These transformations have been selected to help the model to better generalize and because the original dataset was not large enough to accomplish the goal fixed for this project.



(a) Original frame

(b) Horizontal transformation

(c) Rotate transformation

(d) Translate transformation

Figure 3.4: Data Augmentation example on the "Torino" sign

For instance, a vertical flip does not make sense and would probably not be appropriate given that the model is very unlikely to see a video which is recorded upside down.

3.2.3 Text mining techniques

During the extraction of new word samples from videos which are not part of the original A3LIS-147 dataset, it was noticed that a relevant number of words have a common root, but a different final form. For this reason, text mining mechanisms need to be used to reduce these words to their common roots and assign these new samples to their correct labels.

For doing so, the NTLK Python package has been used. It provides a set of diverse natural language algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis and so on. NLTK helps the computer to analysis, preprocess and understand the written text.

Since the sign performed in a single video is just a single one, the tokenization mechanism, which is usually the first step in text analytics and consists in the process of breaking down a text paragraph into smaller chunks such as words or

sentences, is not needed. Instead, lexicon normalization is really helpful for the proposed task, since it reduces derivationally related forms of a word to a common root word. In particular, the *stemming* technique has been used: it is a process of linguistic normalization which reduces words to their word root or chops off the derivational affixes. For example, *connection, connected, connecting* words can be reduced to a single common word, *connect*.

3.3 Feature selection

The dimensionality of the data fed to a neural network during training is a critical factor for the correct resolution of the problem. Usually, the higher the dimensionality, the higher will be the computational cost of modelling, and in some cases the model could suffer from this abundance of features, learning irrelevant information which can deteriorate its generalization capabilities. For this set of reasons, the feature selection technique is really helpful and can be crucial for the proper training of the model.

Feature selection methods have been developed to decrease the dimensionality of input variables. The goal is to choose only those elements which are the most useful for the model, so that it is able to correctly predict the target variable. As said in [55]: 'Feature selection is primarily focused on removing non-informative or redundant predictors from the model'. Since the system on which the model has been trained has a limited amount of memory, this sort of technique is twice useful, because in this way less resources are required and, in addition, the performance of the model will increase, since it will not be affected by input elements that are not relevant to the target variable.

The feature selection mechanisms that have been used in this project use statistical techniques, like correlation, to evaluate the relationship between each input feature and the target variable. These values are then used to make decisions and filter those input features that will be used by the neural network.

Feature selection and dimensionality reduction are related, but they are not the same. Both are based on the concept of retaining fewer input features starting from the original ones, but the difference stands in the methodology: feature selection chooses the features to keep or remove from the dataset, while dimensionality reduction creates a projection of the data onto a lower-dimensional feature space, and this results in a whole new set of input features. This is why they are considered as alternatives.

Usually, correlation type statistical measures between input and output are used to filter the features fed to the network. The choice of which kind of statistical measure to consider is highly dependent on the data type under analysis. The most common data types are numerical (integer or float) and categorical (labels, for instance). The input features are provided to the model, and they are the target to be reduced in feature selection mechanisms. Output variables, instead,

are going to be predicted by the model, and their type determines the kind of predictive modelling problem being performed. For instance, if the model tries to predict numerical values, then it is a regression problem, while a categorical output variable indicates a classification task. The case considered in this dissertation, as said, is part of this last problem family.

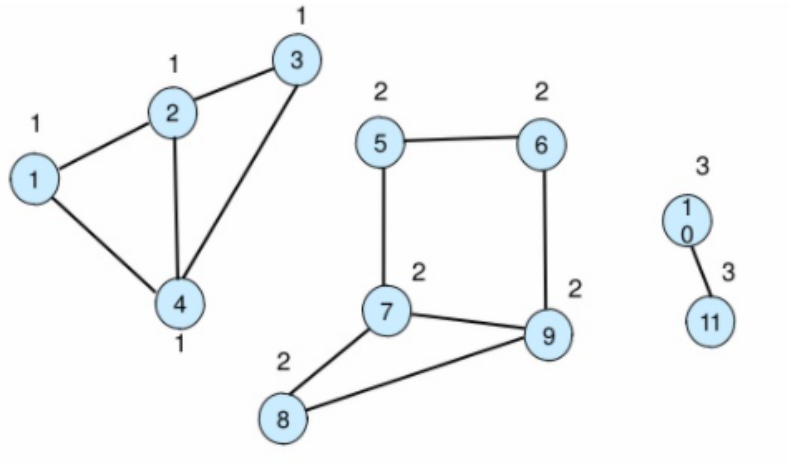


Figure 3.5: Example of graph with connected components

Based on the type of input and output features, different feature selection techniques can be chosen. For the classification task, which considers numerical inputs and categorical outputs, the most common methods are based on correlation. Among them, the ANOVA correlation coefficient and the Kendall's rank coefficient are some of the most used tools for feature selection. In general, a correlation coefficient is a numerical measure of a type of correlation, which means that there is a statistical relationship between two variables.

In this work, however, it was decided not to rely on this general coefficient, but to manage a correlation analysis based on *graph theory*. The graph theory is the study of graphs, which are defined as mathematical structures to model pairwise relations between distinct elements. In general, a graph (or network) G is a pair of elements (V, E), where V is a finite set whose elements are called vertices (or nodes) and E is a subset whose elements, called edges (or links), are couples of elements belonging to V. The edge which goes from vertex i to vertex j is indicated with (i, j).

Routes can be defined in a graph, selecting a starting node (source), a destination node (destination) and a set of edges capable of connecting them, eventually through intermediate nodes. A network is a directed graph (or digraph) if it contains at least one oriented link, which is an edge characterized by an orientation, thus it cannot be traversed in the opposite direction. Otherwise, the graph is undirected.

A subgraph of a graph G is composed by a set of vertices which is a subset of G

ones, and whose adjacency relation is a subset of that of G restricted to this subset. Finally, a connected component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the rest of the graph. An example of a graph with three connected components can be found in Figure 3.5.

An ensemble of nodes which are part of a connected component is characterized by an high correlation coefficient, so these elements are greatly connected and depend on each other. For this reason, the connected component tool has been used to find the features that retain the biggest amount of correlation. First, groups of highly correlated variables have been found and they have been considered as nodes in a graph; then, their correlation has been calculated, and, if the value of correlation between two nodes was higher than 0.8, then it was considered as an undirectional link between those two elements. Subsequently, the connected components of the graph have been found, that are groups of features highly correlated. Finally, for each connected component, the feature with the highest variance has been retained, considering it to be the most representative for that group of connected components.

By using the previously exposed methodologies, starting from 238 features characterizing each single temporal frame, the development team extracted only 38 features which are the most important and most representative of the information contained inside each video sample. Applying this kind of technique allows the network to not focus on irrelevant data and to better generalize, obtaining better overall results for the proposed goal.

3.4 Training and testing results

At the very beginning of the training process, the whole A3LIS-147 dataset was used to tune the neural network presented in the previous sections. The results obtained, without applying any kind of hyper-parameter tuning or technique to avoid overfitting and considering the complete ensemble of features, where not satisfying, since the model reached to obtain an acceptable accuracy only on the training set, but behaved very poorly on unseen data, with a final accuracy of more or less 20%.

Given this situation, the development team started to apply the techniques and methods depicted earlier in this work, to obtain better results and find a sweet spot in which the neural network was able not to focus on irrelevant information, but to really understand the temporal information included in the input data.

A relevant amount of time is needed to reach an acceptable level of performance, since, as it is known from experience, when dealing with neural networks, it is mainly a question of trying and trying again with slightly different changes in parameters, to find an adequate solution.

Together with these techniques, the team also decided to reduce the number of signs which would be recognized by the neural network; to enlarge the original dataset, a great number of different signs was added to the vocabulary, reaching more than a thousand words. This great amount of different classes was not feasible to understand for a relatively simple neural network architecture like the one used for this application, so the decision of shrinking the vocabulary to 100 or 50 different signs was taken.

The team decided to focus mainly on words relative to a specific domain, mainly regarding the instruction field, so signs like "school", "exam" or "diploma" were maintained inside the vocabulary. This decision leads to a significant improvement in performance, since the network needs only to focus on understanding information to discriminate between a lower number of classes.

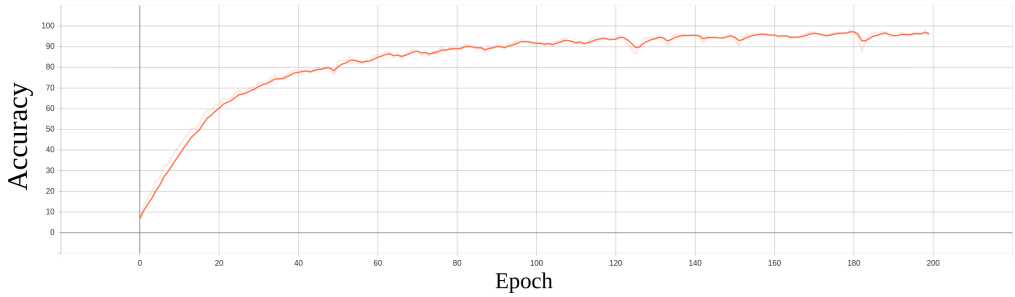
In each of the different training process which were executed on the network, the model was always able to obtain an overall accuracy greater than 80% on the validation set, as can be observed in Figure 3.6. In the previous chapters, it was said that K-Fold Cross Validation technique was used for the training, and the figure shows only one of the different folds, as an example. As visible in the indicated figure, the accuracy value over the train set almost reaches the maximum value after approximately 100 epochs, and after the same amount of computation, the validation accuracy is able to settle on its best value, around 80%.

Instead, concerning the loss function, the Categorical Cross-Entropy Loss is the one used by the algorithm to measure and reduce the error. It is also called Softmax Loss, since it requires a Softmax activation plus a Cross-Entropy loss. Using this kind of loss, the neural network will be trained to output a probability over the C classes for each sign; it is used for multi-classification. In Figure 3.7 it is possible to see an explanation of how it is computed. In particular, t_i and s_i are, respectively, the ground-truth and the neural network score for each class $_i$ in C.

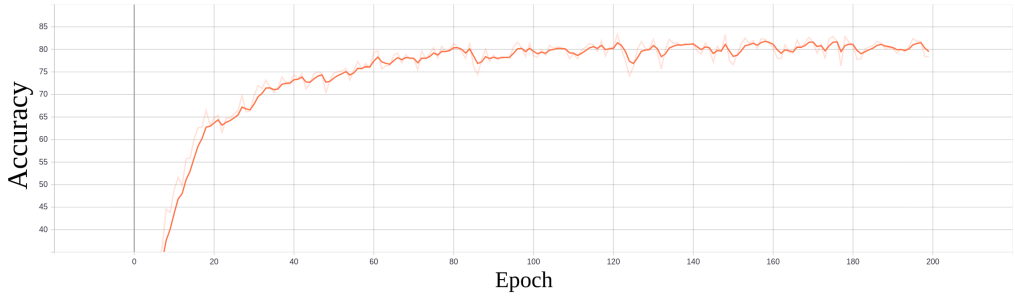
As can be expected for this kind of task, the loss, both for train and validation, starts at a very high value and quickly decrements, until it reaches its best and lower value — remember that the goal of the neural network is to minimize the loss function on the train set, and hopefully this will lead to a minimization also on the validation set, if overfitting has been avoided. An example of the envelop of losses on train and validation datasets is provided in Figure 3.8.

As can be seen in the loss envelops, the train loss could still decrease, considering to run the training task longer, but the validation loss reaches its best and lower value after more or less 100 epochs. Later on, it starts to slightly increase due to the overfitting phenomenon, so this is why the early stopping mechanism has been implemented.

Considering the results that have been obtained in the literature, it is worth considering the first Italian sign language recognition work, proposed by [56]. The mentioned framework deals with a continuous sign case study, using a vocabulary of 40 signs. No specific distinction is made among the signers which are involved in the



(a) Accuracy on train set



(b) Accuracy on validation set

Figure 3.6: Accuracy envelops over train and validation sets

training and testing of the network, in contrast to the proposed approach; anyway, the accuracy for correct translated sentences refers to two different datasets: in the first, the test set is composed of 30 sentences with 20 distinct signs, with an accuracy of 83.30%. In the second, the test set is composed of 80 sentences and 40 distinct signs, and the accuracy value is 82.50%. As said, the obtained results of the current work are in line with the state of the art results.

In other international works, such as [57] and [58], coloured gloves were used and frames were deleted whenever hand tracking was lost; the accuracy obtained in this manner is really high, but the method used is still based on an intrusive capture

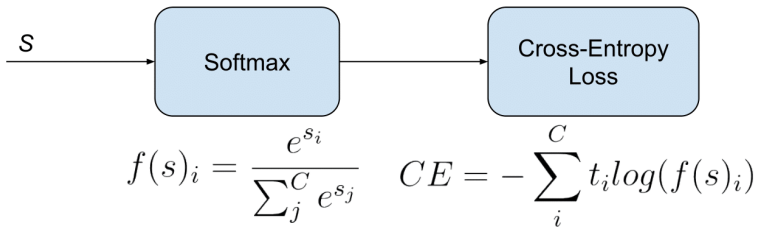
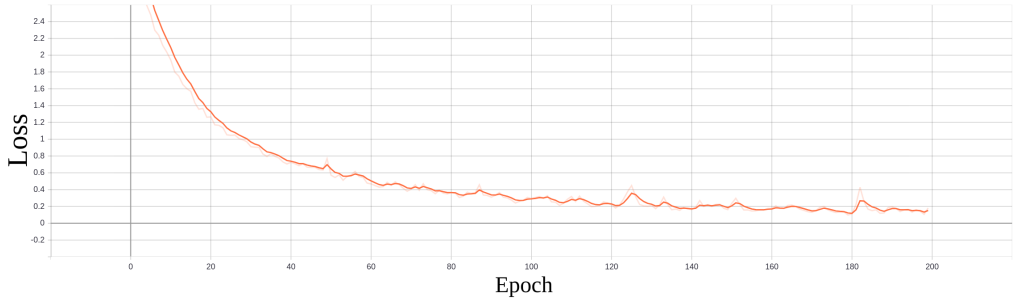
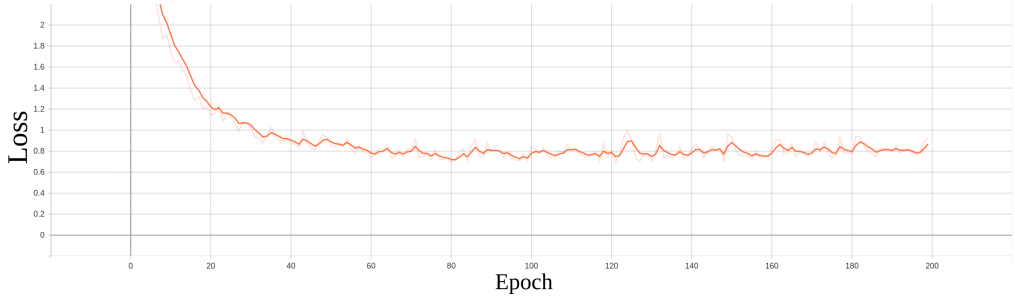


Figure 3.7: Categorical Cross-Entropy Loss



(a) Loss on train set



(b) Loss on validation set

Figure 3.8: Loss envelops over train and validation sets

methodology, which is not in line with the arguments discussed in this thesis. In general, the takeout from these works is that this kind of systems are able to perform well when they are restricted by grammatical rules. These kind of studies show that the use of grammar notions in sign language recognition can be really helpful, and this is why a modern and complete new vocabulary for continuous sign language recognition is being formed by the ORBYTA team.

3.4.1 Discussion about timing

Unfortunately, the development team was not able to carry out a complete experiment flow, meaning that there is still work to do to obtain a successfully usable application. Thus, the experiments that have been executed would not allow to make meaningful observations on the time needed by the application to translate a single sign.

At the actual prototype stage, anyway, the application needs to receive a complete video to start the translation process. This means that, once the video has been recorded, it has to be sent to the server, which will process the entire video and then send back the response to the application. This workflow clearly introduces some latency, which is to be mainly attributed to the video processing mechanism:

extracting the coordinates of hands, face and upper body is really a computational expensive task, and actually the work is completely in charge of the CPU. Indeed, parallelizing the process using GPU would lead to great improvements, but at the actual stage this is not available on the MediaPipe Python implementation. Then, the coordinates need to be fed to the neural network, which in the end will produce the translation. Furthermore, this mechanism has an high computational cost, which translates to an increased execution time.

As previously said, at the moment the coordinates extraction task is performed on the server side, but it could be implemented on the client side: by doing so, the server computational workload would be greatly reduced (considering also scalability problems, since it should serve different clients at the same time) and the time needed to obtain the translation would sharply decrease, since half of the work is already done by the client-side application. This is an essential and helpful future improvement that can be a game-changer for this particular application. After having recorded the video, the time needed to process it, extracting coordinates and obtaining the translation, is slightly equal to the the duration of the video itself. This makes the processing time too long for a daily usage of the application and may return the feeling of an unresponsive software. To improve the user experience and overall satisfaction, these problems are actually being tackled by the development team and are considered as essential future improvements.

To conclude this chapter, the conducted experiments show how the proposed novel mechanism is in line with the state-of-the-art results, obtaining an overall accuracy on the selected training and validation sets which is always greater than 80%. The novelty of the proposed architecture is that it is completely independent from the recording system used to capture the video that will be processed, and in practice this makes the application easily accessible by anyone, especially people with accessibility needs. The aim of the application is to embrace as many people as possible, to reduce the gap between deaf people and the rest of the society, and at the actual stage it can be said that it could have been completely fulfilled.

Conclusions

A novel application to approach sign language recognition has been exposed in this work. The main goal of the current study was to develop an application capable of helping deaf and hard-of-hearing people in relating with the rest of the society. Throughout this dissertation, different aspects of the development process have been reported: the difference between continuous and isolated sign language recognition is the most important one, stating the principal limitation of this work, that is, it is not meant to be a real-life, ready-to-use application, but rather a prototype which settles the groundwork for future researches.

This study has shown that it is possible to obtain a good accuracy sign recognition mechanism starting from a relatively small dataset of signs performed by different signers, reaching the state-of-the-art performance. This sign recognition tool could be used in any context, from school to hospitals, in video conferences and so on, because it needs only a simple camera to work, and so it should be easily accessible by a very good turnout. This was the second aim of the work, which is to embrace as many people as possible in a simple, yet effective way.

The second major finding was that a new dataset for continuous sign language recognition is needed to produce a real-life and real-time application. Isolated sign language recognition can be considered as a starting point, while the real-time translation of sign language should be the final goal. On the example of the German sign language vocabulary [30], a similar process has been started by the ORBYTA R&D team, in order to obtain a ready-to-use continuous language recognition dataset that could also be used as an international benchmark for testing other automatic recognition systems.

The results of this research support the idea that a real and effective inclusion of deaf and hard-of-hearing is possible, but there is still a great amount of work to do. There are several possible improvements of this work: first of all, it could be extended and become a continuous sign language recognition system implementing the two extra models that have been previously introduced. The first model consists in an automatic segmentation software, based on neural network and deep learning: given a video containing a sign language sentence, it should be able to precisely segment the video into glosses, containing a single sign. By doing so, the obtained glosses could be subsequently passed to the network already analyzed in this work,

which would return the translation of the sign. In the end, the sentence translation should be returned, but, as seen in the last chapter, applying also grammatical constraints to the model it should perform better, so also this aspect could be implemented as a future improvement.

Another limitation of the proposed study, even with the segmentation model, consists in the final translation: even constraining the overall model with sign language grammar rules, the returned sentence would not satisfy the Italian grammatical rules, due to the fact that they are different from LIS ones. For this reason the fourth and last model has been recognised as necessary: it should consist in a network capable of taking a LIS sentence and convert it into a correct Italian phrase. This last model could be implemented by taking advantage of the improvements obtained in Natural Language Recognition processes.

Finally, another problem of the proposed architecture could be inherent to the time needed to process videos and return the correct translation. As shown in the previous chapter, most of the work is handled on the server side: both the coordinates extraction and the translation processes are performed by the server, but this is only feasible in the prototype phase. It is not imaginable to put a system like this in production for obvious scalability issues, and so a further improvement could consist in moving some of the computation to the client side: it should be possible to perform the coordinates extraction process on the client (usually a smartphone or PC), since the MediaPipe project has been developed with the aim of exploiting mobile resources in the most effective way. In this way, the total amount of time needed to obtain the final translation should be definitely reduced, if not halved.

Notwithstanding the relatively limited prototype that has been developed, this work offers valuable insights into the automatic sign recognition field of study. A natural progression of this work is to analyse which results can be achieved by increasing the size of the dataset and of the vocabulary; this should be done hand in hand with an extension of the neural network used to translate signs, since increasing the number of words of the vocabulary also increases the computational workload on the network itself.

A further study could assess the possibility of transforming the proposed architecture into a real-time application: firstly, at the current state, the input needs to be segmented, but adding support for unsegmented data should be straightforward, requiring a change in the training protocol as demonstrated in [47]. Finally, since the system only works on complete sequences, to get a real-time, step by step, classification, the usage of non-overlapping short time windows could be helpful. As stated in [59], the recognition model can emit a classification of the sign performed inside each window; then, the use of an objective function, such as the Connectionist Temporal Classification Loss (CTC) [49] could allow an alignment of the classes obtained from the time windows with the desired (actual) ones.

Greater efforts are needed to ensure a proper integration of deaf and hard-of-hearing people with the rest of the society. More research and innovation are needed to improve the quality of communications between these different, but yet linked, worlds; and this can obviously lead to great benefits for both parties.

Appendix A

Listing of CI instructions in `gitlab-ci.yml`

```
docker-build-master:
  image: docker:latest
  stage: build
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
        $CI_REGISTRY
  script:
    - docker build --pull -t "$CI_REGISTRY_IMAGE" ./LIS2Speech
    - docker push "$CI_REGISTRY_IMAGE"
  only:
    - master

docker-build:
  image: docker:latest
  stage: build
  services:
    - docker:dind
  before_script:
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD"
        $CI_REGISTRY
  script:
    - docker build --pull -t "$CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG"
        ./LIS2Speech
    - docker push "$CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG"
  except:
    - master
```

Appendix B

Listing of Python script for hyper-parameters tuning

```
import torch
import torch.nn as nn
import optuna


def hyperparams_tuning(trial):
    # Set the suggested hyperparams
    cfg = {
        "device": "cuda" if torch.cuda.is_available() else "cpu",
        "n_epochs": 2,
        "seed": 0,
        "log_interval": 100,
        "lr": trial.suggest_loguniform("lr", 1e-4, 1e-2),
        "optimizer": trial.suggest_categorical(
            "optimizer", [torch.optim.SGD, torch.optim.RMSprop, torch.optim.
    Adam]
        ),
        "model": trial.suggest_categorical("model", [DeepGRU, DeepLSTM]),
        "weight_decay": trial.suggest_loguniform("weight_decay", 0, 1e-1),
        "criterion": nn.CrossEntropyLoss(),
    }

    # Load the dataset
    log.set_dataset_name("a3lis")
    dataset = DataFactory.instantiate(dataset_name="lis2s", num_synth=0)

    torch.manual_seed(cfg["seed"])
    train_loader, test_loader = dataset.get_data_loaders(
        fold_idx=0, shuffle=True, random_seed=cfg["seed"], normalize=True
    )

```

```
31      # Create the model with the suggested hyperparams
32      model = cfg["model"](dataset.num_features, dataset.num_classes).to(cfg["
        device"])
33
34      # Set the optimizer with the suggested hyperparams
35      optimizer = cfg["optimizer"](
36          model.parameters(), lr=cfg["lr"], weight_decay=cfg["weight_decay"]
37      )
38
39      # Train and compute test_accuracy, which is the goal to optimize
40      for epoch in range(1, cfg["n_epochs"] + 1):
41          train(
42              cfg["log_interval"], model, train_loader, optimizer, epoch, cfg["
        criterion"]
43          )
44          test_accuracy = test(model, test_loader, cfg["criterion"])
45
46      return test_accuracy
47
48
49  if __name__ == "__main__":
50      sampler = optuna.samplers.TPESampler()
51
52      # Create the study object, maximize the value returned by the objective
        function
53      study = optuna.create_study(sampler=sampler, direction="maximize")
54
55      # Start the hyperparams tuning
56      study.optimize(hyperparams_tuning, n_trials=20)
57
58      # Print the best hyperparams
59      print(study.best_trial)
```

Bibliography

[1] W. Sandler and D. Lillo-Martin, "Sign language and linguistic universals," in *Cambridge University Press*, 2006

[2] Z. Yang, Z. Shi, X. Shen, and Y.-W. Tai, "Sf-net: Structured feature network for continuous sign language recognition," *arXiv preprint arXiv:1908.01341*, 2019

[3] B. Cooper and R. Bowden, "Sign language recognition," in *Visual Analysis of Humans*, pp. 539–562, 2011

[4] M. Fagiani, E. Principi, S. Squartini and F. Piazza, "A New Italian Sign Language Database" in *Advances in Brain Inspired Cognitive Systems*, pp. 164–173, 2012

[5] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C. Chang, M. G. Yong, J. Lee, W. Chang, W. Hua, M. Georg, M. Grundmann, "MediaPipe: A Framework for Building Perception Pipelines," *arXiv preprint arXiv:1906.08172*, 2019

[6] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C. Chang, M. Grundmann, "MediaPipe Hands: On-device Real-time Hand Tracking," *arXiv preprint arXiv:2006.10214*, 2020

[7] Y. Kartynnik, A. Ablavatski, I. Grishchenko, M. Grundmann, "Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs," *arXiv preprint arXiv:1907.06724*, 2019

[8] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, M. Grundmann, "BlazePose: On-device Real-time Body Pose tracking," *arXiv preprint arXiv:2006.10204*, 2020

[9] M. Maghoumi, J. J. LaViola Jr, "DeepGRU: Deep Gesture Recognition Utility," *arXiv preprint arXiv:1810.12514*, 2019

[10] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv preprint arXiv:2006.10204*, 2014

[11] S. Dupond, "A thorough review on the current advance of neural network structures," in *Annual Reviews in Control*, pp. 200–230, 2019

[12] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015

[13] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recogni-tion with 3d convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1–7, 2015

[14] D. S. Alexiadis, A. Chatzitofis, N. Zioulis, O. Zoidi, G. Louizis,D. Zarpalas, and P. Daras, "An integrated platform for live 3d human reconstruction and motion capturing," *IEEE Transactions on Circuits and Systems for Video Technology*, v. 27, n. 4, pp. 798–813, 2016

[15] M. W. Kadouset al., "Machine recognition of auslan signs using powergloves: Towards large-lexicon recognition of sign language," in *Proceedings of the Workshop on the Integration of Gesture in Languageand Speech*, v. 165, 1996

[16] H. Cooper, E.-J. Ong, N. Pugeault, and R. Bowden, "Sign language recognition using sub-units," *Journal of Machine Learning Research*, v. 13, n. Jul, pp. 2205–2231, 2012

[17] C. Camgoz, S. Hadfield, O. Koller, and R. Bowden, "Using convolutional 3d neural networks for user-independent continuous gesture recognition," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 49–54, 2016

[18] G. D. Evangelidis, G. Singh, and R. Horaud, "Continuous gesture recognition from articulated poses," in *European Conference on Computer Vision*, pp. 595–607, 2014

[19] N. Neverova, C. Wolf, G. Taylor, and F. Nebout, "Moddrop: adaptive multimodal gesture recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 38, n. 8, pp. 1692–1706, 2015

[20] D. Wu, L. Pigou, P.-J. Kindermans, N. D.-H. Le, L. Shao, J. Dambre, and J.-M. Odobez, "Deep dynamic neural networks for multimodal gesture segmentation and recognition," *IEEE transactions on pattern analysis and machine intelligence*, v. 38, n. 8, pp. 1583–1597, 2016

[21] O. Koller, J. Forster, and H. Ney, "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers," *Computer Vision and Image Understanding*, v. 141, pp. 108–125, 2015

[22] O. Koller, C. Camgoz, H. Ney, and R. Bowden, "Weakly supervised learning with multi-stream cnn-lstm-hmms to discover sequential parallelism in sign language videos," *IEEE transactions on pattern analysis and machine intelligence*, 2019

[23] R. Cui, H. Liu, and C. Zhang, "A deep neural framework for continuous sign language recognition by iterative training," *IEEE Transactions on Multimedia*, 2019

[24] D. Bragg, O. Koller, M. Bellard, L. Berke, P. Boudrealt, A. Braffort, N. Caselli, M. Huenerfauth, H. Kacorri, T. Verhoef et al., "Sign language recognition,

generation, and translation: An interdisciplinary perspective," *arXiv preprint arXiv:1908.08597*, 2019

[25] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," in *IEEE Multimedia - IEEEMM*, v. 19, pp. 4–10, 2012

[26] N. Adaloglou, T. Chatzis, I. Papastratis, A. Stergioulas, G. T. Papadopoulos, V. Zacharopoulou, G. J. Xydopoulos, K. Atzakas, D. Papazachariou, P. Daras, "A Comprehensive Study on Sign Language Recognition Methods," *arXiv preprint arXiv:2007.12530v1*, 2020

[27] O. Koller, "Quantitative Survey of the State of the Art in Sign Language Recognition," *arXiv preprint arXiv:2008.09918v2*, 2020

[28] H. Zhou, W. Zhou, Y. Zhou and H. Li, "Spatial-Temporal Multi-Cue Network for Continuous Sign Language Recognition," *arXiv:2002.03187 [cs]*, 2020.

[29] O. Koller, H. Ney and R. Bowden, "Automatic Alignment of HamNoSys Subunits for Continuous Sign Language Recognition," in *LREC Workshop on the Representation and Processing of Sign Languages*, pp. 121–128, 2016

[30] O. Koller, J. Forster, and H. Ney, "Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers," in *Computer Vision and Image Understanding (CVIU)*, v. 141, pp. 108–125, 2015

[31] O. Koller, H. Ney, and R. Bowden, "Deep Hand: How to Train a CNN on 1 Million Hand Images When Your Data Is Continuous and Weakly Labelled," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 3793–3802, 2016

[32] M. Mukushev, A. Sabyrov, A. Imashev, K. Koishybay, V. Kimmelman, and A. Sandygulova, "Evaluation of Manual and Non-manual Components for Sign Language Recognition," in *Proceedings of The 12th Language Resources and Evaluation Conference*, pp. 6073–6078, 2020

[33] A. Sabyrov, M. Mukushev, and V. Kimmelman, "Towards Real-time Sign Language Interpreting Robot: Evaluation of Non-manual Components on Recognition Accuracy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 75–82, 2019

[34] H.-D. Yang and S.-W. Lee, "Combination of manual and non-manual features for sign language recognition based on conditional random field and active appearance model," in *Int. Conf. on Machine Learning and Cybernetics (ICMLC)*, v. 4, pp. 1726–1731, 2011

[35] C. Zhang, Y. Tian, and M. Huenerfauth, "Multi-modality American Sign Language recognition," in *Proc. IEEE Int. Conf. on Image Processing (ICIP)*, pp. 2881–2885, 2016

[36] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," in *Psychological Review*, pp. 65–386, 1958

[37] L. Zhang, S. Chen, T. Wang, and Z. Liu, "Automatic Facial Expression Recognition Based on Hybrid Features," in *Energy Procedia*, v. 17, pp. 1817–1823,

2012

[38] A. J. Logan, G. E. Gordon, and G. Loffler, "Contributions of individual face features to face discrimination," in *Vision Research*, v. 137, pp. 29–39, 2017

[39] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016

[40] S. Hochreiter and J.Schmidhuber, "Long Short-Term Memory," in *Neural computation*, v. 9(8), pp. 1735–1780, 1997

[41] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-Temporal LSTM with Trust Gates for 3D Human Action Recognition," in *Computer Vision – ECCV 2016*, pp. 816–833, 2016

[42] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015

[43] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, "An End-to-End Spatio-temporal Attention Model for Human Action Recognition from Skeleton Data," in *AAAI*, v. 1, pp. 4263–4270, 2017

[44] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," in *CoRR, abs/1409.1556*, 2014

[45] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, v. 37, pp. 448–456, 2015

[46] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," in *arXiv preprintarXiv:1207.0580*, 2012

[47] F. M. Caputo, S. Burato, G. Pavan, T. Voillemin, H. Wannous, J. P. Vandeborre, M. Maghoumi, E. M. Taranta II, A. Razmjoo, J. J. LaViola Jr., F. Manganaro, S. Pini, G. Borghi, R. Vezzani, R. Cucchiara, H. Nguyen, M. T. Tran, and A. Giachetti, "Online Gesture Recognition," in *Eurographics Workshop on 3D Object Retrieval*, 2019

[48] D. Qiulei, W. Yihong and H. Zhanyi, "Gesture Segmentation from a Video Sequence Using Greedy Similarity Measure," in *10.1109/ICPR.2006.608*, pp. 331–334, 2006

[49] A. Graves and N. Jaitly, "Towards End-to-End Speech Recognition With Recurrent Neural Networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1764–1772, 2014

[50] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf and E. A. Fox, "Natural Language Processing Advancements ByDeep Learning: A Survey," in *arXiv:2003.01200*, 2020

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. De-Vito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch,"

in *NIPS-W*, 2017

[52] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function", in *The Annals of Mathematical Statistics*, v. 23, n. 3, pp. 462–466, 1952

[53] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6a: Overview of mini-batch gradient descent", in *Neural Networks for Machine Learning*, 2012

[54] D. P. Kingma and J. Ba. Adam, "Adam: A Method for Stochastic Optimization," in *arXiv preprint arXiv:1412.6980*, 2014

[55] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, ed. 1, New York, Springer, 2013

[56] I. Infantino, R. Rizzo, and S. Gaglio, "A framework for sign language sentence recognition by commonsense context," in *IEEE Trans Syst Man Cybern C Appl Rev*, v. 37, pp. 1034–1039, 2007

[57] T. Starner, A. Pentland, "Real-time American Sign Language recognition from video using hidden Markov models," in *Proceedings of the international symposium on computer vision*, pp. 265–270, 1995

[58] T. Starner, J. Weaver, A. Pentland, "Real-time American Sign Language recognition using desk and wearable computer based video," in *IEEE Trans Pattern Anal Mach Intell*, v. 20, pp. 1371–1375, 1998

[59] G. Devineau, W. Xi, F. Moutarde, J. Yang, "Deep Learning for Hand Gesture Recognition on Skeletal Data," in *13th IEEE Conference on Automatic Face and Gesture Recognition (FG'2018)*, 2018