UCLouvain

LOUVAIN
School of Management

Louvain School of Management

Investigating the effect of dimensionality reduction in a sign language recognition task

Author(s): Jérémy de Bodt, Philippe Palumbo
Supervisor(s): Pr. Marco Saerens
Academic year 2023-2024
Dissertation for the master of Business Engineering
Master subject and focus: Investigating the effect of dimensionality reduction in a sign language recognition task

Declaration Regarding AI Tool Usage in Master's Thesis

During the preparation of this master's thesis, Palumbo Philippe & de Bodt Jérémy utilized CHATGPT for the following purpose:

1. REASON : I frequently use ChatGPT to help coding and rewrite my text. This AI tool offer clear explanations which allow me to debug and improve my code. Additionally, it assists in rewriting my master thesis to improve the vocabulary, clarity and coherence. I ensure that my communication is way more effective and fluid.
2. After using [CHATGPT], de Bodt Jérémy & Palumbo Philippe diligently reviewed and edited the content produced by the tool. We take full responsibility for the final content presented in this thesis.

Signature :

de Bodt Jérémy
Palumbo Philippe

List of contributions

The main contributions of this dissertation are listed here under.

- First of all, a key point of our master thesis is to understand which body parts provide the most information in sign language recognition using machine learning models. We indeed have four different sources of data: the face, the pose, the right hand, and the left hand. The aim is to understand which of these anatomical parts, or fusion of these, provides the best information to create a whole system to recognize different signs.
- We introduce a new approach to transform coordinates coming from our dataset into an image. Our goal is to check whether this transformation performs well in sign language recognition.
- Another goal of this master thesis is to test if using singular value decomposition (SVD) to decompose and recompose our inputs helps remove noise from input data. Additionally, we explore whether alternative dimensionality reduction techniques display better performances. Indeed, the size of our dataset is very large, we need to sort our inputs because the computation time and complexity of the various algorithms have increased considerably. This is the reason that explains our interest in dimensionality reduction methods, which should help to overcome this problem while keeping as much information as possible.
- We finally explore various machine learning models and investigate which of these models perform best in a sign language recognition. Moreover, we study whether algorithms that are able to take into account the time dimension obtain better accuracy than those that are not.

Abstract

In today's world, communication is one of the most important aspects of our society. With the rise of new technologies, communication between people has become even easier. However, for a certain portion of the population, it is not the case. Indeed, the deaf community still encounters great difficulty communicating with other people, as few people speak sign language. Our master thesis's objective is to study the effect of dimensionality reductions on machine learning models for isolated sign language recognition. Three dimensionality reduction techniques were investigated: principal component analysis, auto-encoder, and singular value decomposition. Then, for each of these techniques, we explore the performance of several models, such as support vector machine, artificial neural network, recurrent neural network, gated recurrent unit, and long short-term memory, to see which model works best with which dimensionality reduction technique. Lastly, we investigate the transformation of our data into images in order to apply convolutional neural network.

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Prof. Saerens for his exceptional follow-up, continuous support, discussions, advice, patience, time and ideas. His guidance helped us all year, and this master thesis could not have been undertaken without his insightful comments and constant motivation.

We also thank researchers at the University of Namur, especially Jérôme Fink for giving us access to large datasets and computing resources.

We thank Prof. Kolp for accepting to read this dissertation.

We thank our brothers and sisters for their help, time, and kindness. We especially thank Marie, Cyril, and Dounia for their remarks and corrections.

Finally, we thank our parents for their support, insightful discussions, and proofreadings, which were certainly quite painful at some points.

Contents

Introduction

" Communication is the key to any successful relationship. (John C. Maxwell) "

Our master thesis is dedicated to the study of modern techniques used to automatically interpret sign language communication between humans. This thought of John C. Maxwell, a famous American author and speaker, summarize nicely why we decided to work on this subject. Communication is the central element of human interaction and an essential aspect of our everyday lives. It is much more than a simple exchange of words or gestures; it serves as the conduit through which emotions and ideas are shared between people. From the first moments of our lives, we need to communicate. This shapes our experiences, decisions, and connections with others.

However, even if communication is one of the most important elements in our lives, few people realize how lucky we are to be able to communicate as we wish, using multiple channels (vision, hearing, speaking). But many many people suffers from earing disabilities. According to the 2020 World Health Organization (WHO) report, there are over 466 million individuals worldwide who are deaf and use various sign languages, such as American Sign Language (ASL), Argentine Sign Language, Polish Sign Language, German Sign Language, Greek Sign Language, Spanish Sign Language, Chinese Sign Language, Korean Sign Language, and Persian Sign Language [1] [2]. These numbers underline the importance of developing a system that will enable hard-of-hearing people to communicate like any other person. How do these people communicate? Sign language is their dominant form of communication people (see [1]). But despite sign language is widely used in the deaf community, very few people in the hearing community know that language, making communication between the two communities almost impossible. For example, imagine a deaf person asking you for help in a store, but you don't speak sign language. Only written communication is possible, except if you have at disposal a device or an application that translates signs into text.

Sign language is definitively a vital channel of communication for the deaf community. Despite its critical role, the integration of sign language into mainframe communication systems remains limited. This gap highlights the need for technological solutions that can reduce the communication barriers that face numerous people in the world.

In recent years, the fields of machine learning and artificial intelligence have seen many major advances, which has spurred the interest of researchers in the field of sign language recognition. The creation of systems that can translate sign language into text or spoken language represents, however, a huge challenge. It is an incredibly difficult task with many different problems to overcome, such as movement epenthesis, co-articulation, etc.

In this Master's thesis, we explore and implement machine learning models that can efficiently help interpret sign language for isolated signs. Various machine learning models will be investigated such as support vector machines and deep learning techniques like convolutional neural networks, recurrent neural networks, and artificial neural networks. We will also explore other techniques that use distances between coordinates as information to classify signs. To increase the accuracy of models, we will show that dimension reduction techniques, such as principal component analysis (PCA), will play a crucial role.

With this master thesis, we hope to contribute to the growing interest in machine learning-based techniques to interpret sign language. This is a fundamental endeavor to promote deaf community integration within our society. As business engineering students, we are aware of the importance of effective communication in any organizational context. However, many hard-of-hearing people face significant problems in workplace environments, and managers struggle to communicate with them. Advances in automated sign language interpretation have the potential to significantly improve the quality of life in the deaf community. Such systems fit in various domains, such as education, healthcare, and any public or private institution. For example, in education, sign language recognition systems can provide assistance to deaf students to provide them with a better learning experience.

This master thesis is divided into five chapters. We will start with the state of the art to provide a broad overview of the literature on sign language recognition. We focus on identifying the current knowledge landscape, emphasizing the trends, and pointing out the main challenges. Then, the second chapter provides a theoretical and synthetic presentation of modern machine learning models and dimension reduction techniques we use in our work. The third chapter introduces the data sources that we use to train our models and the data transformations that are done. The fourth chapter describes our implementation of the used models; each step needed to perform them will be explained. Finally, we will report our results.

Chapter 1

State Of The Art

This first chapter will provide a general overview of the current state of the art in sign language recognition (SLR). By reviewing the recent literature addressing this topic, this chapter will provide important information about current knowledge, with the goal of identifying what appear to be the most significant current issues.

In recent years, with the emergence of big data, new opportunities have arisen in the SLR field. SLR has made many advancements recently due to the possibility of using techniques in deep learning to overcome up-to-now unresolved difficulties. As a consequence, the SLR landscape has undergone many changes recently, not only due to the evolution of the technology but also because of the growing interest of researchers in this field.

This state of the art section will be structured into four sub-sections. The first one will be a general presentation of sign language, which includes three different items: an historical perspective on SLR, SLR applications, the definition of isolate, and continuous SLR. The second sub-section will describe features that are used in SLR. The third sub-section will explain the different approaches that the researcher can use to recognize signs, and lastly, we will present the main challenges of this field.

The principal limitation of this state of the art section is that it cannot pretend to exhaustion, as the SLR field is nowadays particularly vast and actively growing.

1.1 Sign Language Recognition

In this section, we will address three different points to give a first insight on when researchers began studying the field of sign recognition, in what applications we can use SLR, and the difference between isolated and continuous SLR.

1.1.1 History Of Sign Language Recognition

Research into sign language recognition (SLR) commenced in the late 20th century, since human-computer interaction has become more commonly available. Research in this field was highly motivated by the need for technologies that could facilitate communication between the deaf and hearing communities [3]. The foundation of SLR is to translate sign language, which means that gesture recognition plays a central role in SLR. It is a domain that poses significant technological challenges due to the nature of sign languages, which are complex gestural languages with a unique grammar and syntax. Gesture and sign language recognition include: “the whole process of tracking and identifying the signs performed and converting into a semantically meaningful words and expressions” (Cheok et al,2020).

Early efforts in SLR can be dated back to 1993, when Darrel and Pentland adapted techniques from speech and handwriting recognition for gesture recognition. To recognize the dynamic gestures of a sign language, they used the dynamic time-warping method [4]. Another example of early work is the first vision-based SLR systems, which depend on handcrafted features, like the work of Huang and Huang in 1998 [5].

Then, a new model that was very successful in automatic speech recognition was tested in SLR: Hidden Markow Models (HMM). The special properties of this model make it particularly effective in gesture recognition. In a study with 262 signs, the HMM obtained an average accuracy of 94% [6]. The use of HMM resulted in the development of a system capable of recognizing a broader sign vocabulary [7].

In recent years, the true breakthrough in SLR arrived when the domain moved towards deep learning techniques with the advent of large, publicly available databases. Its performance in other domains, such as computer vision and speech recognition, has motivated its use in SLR. Their ability to learn important features from raw data, to improve as more data becomes available, and to integrate the temporal part of sign language into their architecture has led to excellent results for these techniques. For example, the use of convolutional neural networks (CNN) with large datasets has led to the creation of far more robust algorithms. It means that these algorithms can be used in very different conditions [7]. Recurrent neural networks such as LSTM are also widely used because they are one of the best models to recognize a sequence of signs and not just one sign at a time.

Despite significant improvements over the past few years, SLR still encounters various challenges, such as movement epenthesis, co-articulation, and the and the need for an even bigger dataset. The transition from the first techniques to the use of deep learning methods represents a considerable step, but the researchers continue to push even further to improve communication between the deaf and hearing communities.

1.1.2 Applications Of Sign Language Recognition

The prime impact of SLR is clearly to translate signs into text or speech to provide the deaf community with the opportunity to communicate with hearing people without an interpreter. But the importance of SLR goes beyond that, and many applications can benefit from SLR engineering.

In [8], the authors present different useful SLR applications:

- Sign-to-text speech translation systems

  The first application is obviously the creation of a system that can translate complex sign sentences into text or spoken language. Many public representatives can use this translation system to help them interact with the deaf community in places such as post offices, hospitals, airports, police stations, and so on.

- Bandwidth conserving systems

  A bandwidth-conserving system plays the opposite role to the sign-to-speech translation one. In this case, we use SLR to translate live video into recognized signs that are animated by avatars at the receiving end. This application allows the deaf community to understand and participate in any live video.

- Video annotation

  In linguistic research, it is important to have sign videos that are annotated. SLR allows you to save significant effort by annotating videos automatically rather than manually.

- Tele presence

  Telepresence is a technology that allows individuals to participate in remote environments as if they were physically present. It is highly useful when manual operations are necessary but physical proximity is impossible or costly, especially

in the case of an emergency or system failure. SLR plays a crucial role in this case because it recognizes and interprets the hand's gestures and transforms them into actionable commands for the machinery. In other words, SLR allows for the efficient use of machines to fix dangerous situations such as undersea missions or the maintenance of nuclear power reactors.

There are other applications that benefit from SLR [9]. Among them, we can cite translation systems, video remote human interpretations, human-computer interactions, online hand tracking of human communications in desktop environments, real-time multi-person recognition systems, games, virtual reality environments, robot controls, and natural language communication.

Many large companies like Google, Microsoft, and Facebook work on projects involving augmented reality, virtual reality, and mixed reality, which has the effect of extending the applicability of SLR. Increasing the performance of SLR can typically be very useful in these fields.

It is clear, then, that beyond the objective of helping the deaf community by making their communication easier, any advances in this field will have an impact in many other areas.

1.1.3 Isolate And Continuous Sign Language Recognition

The literature on SLR can be grouped into two categories: isolated SLR and continuous SLR [8]. The section delves into the inner- workings of both approaches, clarifying their applications and challenges, and emphasizing their importance in improving SLR.

- Isolated Sign Language Recognition

  Isolated sign language recognition concerns the recognition of signs that are performed alone, which means that there is no sign before or after the performed sign. In [8], the authors explain that Dreuw says that the main characteristic of this form of SLR is that the performed sign is not affected by the preceding or succeeding sign. According to the authors of the book, "Visual analysis of humans" [10], The vast majority of SLR research has been concentrated on isolated SLR because it is simpler than continuous SLR. Even though it's important to develop high-performance isolated models because they enable better techniques that make the best use of the features representing the performed signs, Isolated models are, however, not applicable to a real-world SLR system because, since the goal of this category is to translate one sign after another, this system cannot translate all sentences.

  In the article of Rastgoo et al [1],the authors explain that there are still challenges in isolated SLR that need to be managed, such as : "high occlusions of hands, fast hands movement, background complexity, inexistence of the large and diverse datasets, varying illumination conditions, different hand gestures, and complex interactions between hands and objects.".

- Continuous Sign Language Recognition

  Continuous SLR concerns the recognition of a sequence of signs. The user performs one sign after the other, generating a sequence. It is way more challenging since a performed sign is now affected by the previous and the next ones. This effect is called "co-articulation" and can also be found in speech recognition.

  The main challenge is explained by the authors of the book "Visual analysis of humans" [10], and bears on the transition between signs. In a sequence of signs, it is difficult to identify the boundaries between signs in the sequence; in other words, the issue is detecting when the sign starts and ends. This challenge is called "movement epenthesis" or "segmentation" in speech recognition.

  Both challenges will be explained later in the section "Challenges of SLR.".

  Another challenge in continuous SLR is the fact that the time to make a sign can vary due to the context of the sentence. These variations make it harder for the

system to recognize a given sign since the time varies, so the data for the same sign varies, which complicates the training of the algorithm.

Continuous SLR is now widely studied by researchers thanks to the increasing number of large databases available containing many different examples. In order to develop a system for translating sign language into text or spoken language, it is necessary to work on continuous SLR because this is the way sign language is used in real life.

1.2 Features Extraction For Sign Language Recognition

In this section, we describe which types of features are used in SLR and their origin. Joksimoski in his article [11] explains that SLR features can come from three types of sources.

The first type is an image, a sequence of images, or videos that are created using a camera. The second type is the transformed data that comes from specialized equipment such as data gloves that use motion-tracking techniques. The gloves allow for the retrieval of information about movements of the hands and arms, like rotation or the location of joints. The last type of data source is depth data, which uses depth sensors to extract the required information. This technique uses the depth in the scene; in other words, it captures the distance from the sensor to the object. The most common devices are the Microsoft R KenectTM and Intel R RealSenseTM series of sensors.

We now turn our attention to the different features extracted from the data that are used to recognize signs. A sign is composed of two types of features: manual features such as hand shape and position and non-manual features such as facial expression, facial movement, and so on. Elements that make up the manual and non-manual features are shown in Figure 1.1.

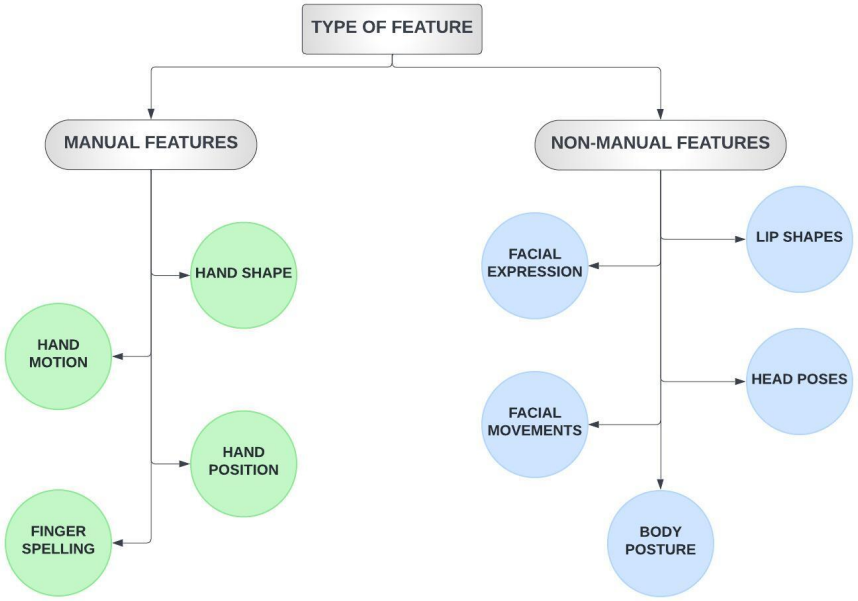

FIGURE 1.1: Type of features in SLR

A feature can come from different points of view [12]. The first simply concerns the part of the body used to sign: hands, head, facial features, and body/shoulder motion. The second aspect is whether a sign provides the main message or a paralinguistic message. In sign language, the main message is composed of a manual (hand-based) and a non-manual feature, where the non-manual one is used to complement the manual

meaning. The last aspect is whether or not a feature has a meaning by itself or not. For example, the non-manual components can alter the meaning of a sentence, but the manual components have, most of the time, a meaning by themselves.

1. Manuel features

   Manual features are the fundamental elements of sign languages that are used to convey linguistic information. They are really the primary means of communication in sign language; it is therefore extremely important to analyze them in order to be able to identify and recognize signs. Of course, the analysis of manual features is highly related to the recognition of gestures but has its own challenges [12].

   - Hand shape

     In gesture languages such as sign language, hand shape is one of the primary features. This feature is widely used in gesture-controlled computing to perform specific commands within an operational system or application. As it is explained in [10], it is challenging to analyze hand shapes due to an image resolution issue. When the whole signer occupies the field of the camera, most often the resolution of the video is not high enough to contain all the specific details about the hand shape. Sensors such as data gloves allow us to collect information more efficiently.

     The other aspect that makes hand shape a challenging feature to apprehend is the fact that in sign language, there are a wide variety of hand shapes. In [12], there are 150 hand shapes in the American Sign Language and 57 variants in the British Sign Language. This great variety makes it more difficult to set up a system capable of recognizing all hand shapes.

   - Hand motion

     The second manual feature is hand motion, which is, with the hand shape and position, one of the most important features of signs. It refers to the specific configuration, especially the rotation and the trajectory, of the hands that perform the sign.

     In sign language, hand trajectories can vary considerably due to the large number of signs, which means that hand movement analysis is complex and requires different approaches. There are two types of signs in sign language [12]: the dynamic ones and the static ones. For the static ones, you don't need to study the trajectory because, as their name suggests, they are static. On the contrary, dynamic signs need a trajectory analysis to be studied. They can be examined in two ways:

     (a) Signs with global hand motion
         This refers to signs with movements of the entire hand or arm during sign execution. In this case, we use the hand center of mass to analyze the trajectory.

     (b) Signs with local hand motion
         It refers to the signs with movement or changes in positions of individuals fingers or hand shapes within a sign. In this case, it is mandatory to analyze every part of the hand.

     The trajectory can be difficult to analyze because it is generally noisy due to bad illumination or occlusions [12]. It is an extremely costly feature in terms of computation time. Segmentation of the hand motion can therefore not be done in every frame, which reduces the system performance.

     The last important information in this feature is the relative position of the hand with respect to the other hand, especially in sign language. For each sign, the hands can perform different actions. For example, the two hands can synchronize their movements entirely, or one hand can stay still while the other is in motion, or else they can approach or move away from each other.

   - Hand position

     Another important manual feature is the hand position in space; the exact coordinates of the fingers and of the overall hand are necessary information. In gesture recognition, the hand's center of mass is occasionally used due

to the difficulty of acquiring these coordinates, but in SLR, this information alone is not sufficient. It is mandatory to make an in-depth analysis of the hands to retrieve the relative location and the global motion of the hands [12]. In addition, these analyses must be performed in three dimensions.

- Finger spelling
  The manual features also encompass finger spelling, even if it is an ancient form of communication. As defined in [13], finger spelling is : "the method of spelling worlds using hands' movements".Finger spelling has two purposes in sign language: the first is to spell names, cities, or other words that have no sign in their own. The second purpose of finger spelling is to clarify a sign when a person doesn't understand the signer.
  Each sign language has its own finger alphabet. For example, in Appendix 1, we have the finger alphabet of the American sign language and the finger alphabet of the British sign language.

2. Non-Manual Features
   Manual features are not the only information that is present in a sign. There are also non-manual features, which are features that do not contain hand'-based signs. The most well-known are the facial expressions, facial movements, lip shapes, head poses, and body postures. These features in sign language serve to either reinforce, weaken, or alter the meaning of signs. For example, facial expressions are used to show emotional nuances or sentence forms (questions or claims, for example).

   In [14], the authors explain that non-manual features provide a significant help for sign language recognition tasks since they bring grammatical and prosodic information. Prosodic information refers to the overarching elements of speech, including intonation, rhythm, stress, and tempo. Even though the importance of facial features is nowadays acknowledged, it took a long time to realize their significance for automatic SLR. It was uncovered in Ulrich von Agris et al. (2008).

   The non-manual feature that allows the best distinction between signs is the lip shape [14]. This feature addresses ambiguities between signs and uses redundant information to strengthen signs' differentiation. Another extremely helpful feature is the head pose, because it communicates a large amount of information, such as whether the signer raises questions, states affirmations, denials, or conditional clauses. Concerning body posture, limited attention has been devoted to this non-manual feature up until now, even though it plays probably an important role in sign language interpretation.

   In order to build an efficient SLR system, it is therefore of first-order importance to take into account the information given by the non-manual features. In [12], The authors explain that two signs with the same manual components can have different meanings depending on non-manual features. In Figure 1.2 come from [14].

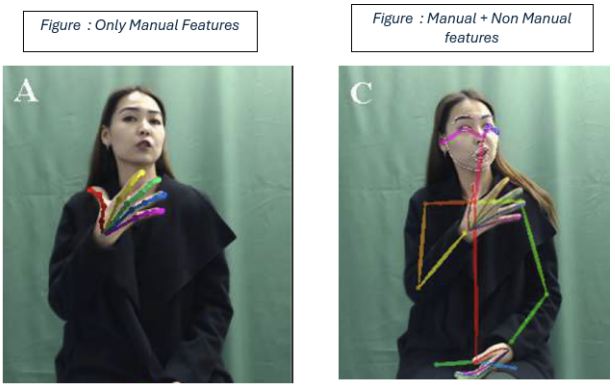

FIGURE 1.2: Manual Features and non Manual Features

In the literature, there are a limited number of works that use both manual and non-manual features to recognize signs. Despite that, when it comes to continuous SLR, it is essential to use both sources of information. It is particularly difficult to combine them because non-manual features are not always synchronized (in time) with the sign itself. This means that the analysis must be performed according to different time scales. In Figure 1.2, the left image display an example into which we only use the manual features, while the right image shows an example where we use manual and non-manual features are used.

1.3 Type Of Sign Language Recognition Approach

In this section, we describe three approaches to capture the valuable elements of hands' configurations to translate signs in natural language. These are shown in Figure 1.3.

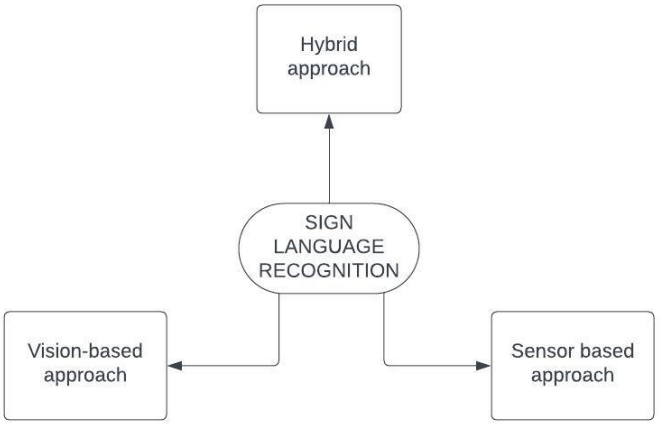

FIGURE 1.3: Type of approach for Sign language recognition

1.3.1 Vision Based Sign Language Recognition

The first approach that we will describe is the vision-based approach, which accords with [15] focus on features' extraction from an image or video. To obtain these images, the main tool that we need is a camera. This is precisely the advantage of this method: the required material is limited, which greatly reduces research costs. Of course, the better the quality of the camera, the sharper the image and the better the recognition system will be. However, there are limitations associated with this technique that cannot be eliminated, even when using a high-precision camera. In [15], the authors list three of them :

1. Limited field of view of the capturing device
2. High computational cost
3. The need for multiple cameras to obtain robust results due to problems of depth and occlusions

Figure 1.4 from [15] shows processing steps needed in vision-based system for SLR. In vision-based methodology, there are four steps to perform :

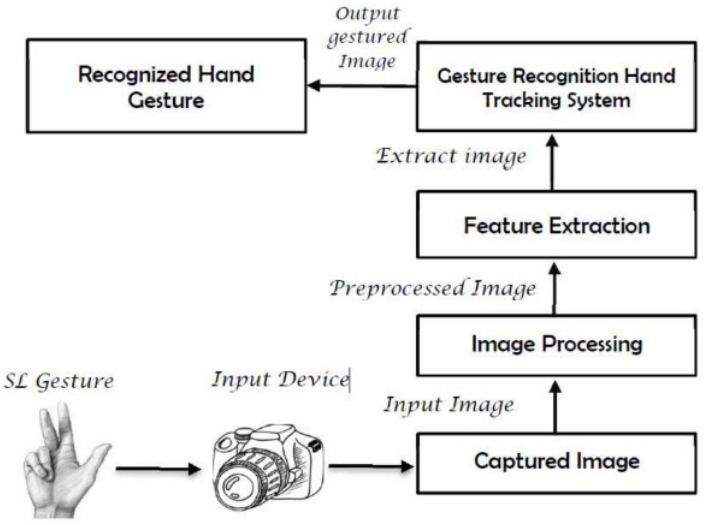

FIGURE 1.4: Vision-based approach,[15]

- First step: Data acquisition which corresponds to the "SL gesture", "Input Device" and "Captured Image".

  The first step, as its name suggests, concerns the acquisition of necessary data. At this step, we need a large number of images of each specific sign—images that have been performed under different conditions—to make the algorithm more resilient to change. As shown in Figure 1.4, there is the sign language gesture recorded by the input device (camera), and then we have the captured image.

- Second step: Image processing and Segmentation.

  The second step is used to prepare the images recorded in the first step. This can involve a number of different tasks, such as standardizing the images so that they have the same format or reducing image noise (i.e., removing anything that prevents the right information from being extracted). The goal here is to improve the robustness of the method that is implemented.

  We also perform the segmentation of the image in this step. In an image, there is a lot of unnecessary information. It is of first-order importance to identify which part of the images contains this information. The goal is to extract relevant parts of the image, especially the once's that are focusing on the gestures [16].

- Third step: Feature extraction.

  The aim is to create a manageable data set from the raw data—the images. Storage and computing power constraints often limit the quantity of information that can be used. Too much data will have negative effects, and in particular, the overall system will be too slow to operate. Dealing with this issue refers to "dimensionality reduction" techniques aimed at reducing the number of dimensions into which the raw information is mapped.

- Fourth step: Classification

  This last step, which is the two last points in Figure 1.4, concerns the algorithm training, which corresponds to the "Gesture Recognition Hand Tracking System" in Figure 1.4, and the classification of the gesture into specific classes, which corresponds to the "Recognized Hand Gesture" in Figure 1.4 There are two types of classification techniques: supervised and unsupervised. The difference between the two is that supervised machine learning takes known data as input in order to train the system to recognize that data, whereas unsupervised machine learning uses unlabeled data as input. Unsupervised techniques do not therefore expect any desired output to be predicted as a result, but rather to find patterns and relationships between the data. It is important to notice that most systems are based on supervised learning.

1.3.2 Sensor Based Sign Language Recognition

The second approach to learning is the sensor-based approach, which, according to [15], uses mechanical gloves that contain sensors to record hand or other body parts gestures. Different types of sensors are used to capture this information, such as flexion (or bend) sensors, proximity sensors, accelerometers (ACC), and abduction sensors. There are multiple parts of the hand that can be measured with these sensors, such as the bend angles for fingers, the abduction between fingers, and the orientation of the wrist. The important difference and advantage compared to the first approach is that by using gloves with sensors, there is no need to process the raw data into usable data. Indeed, the gloves can directly report the relevant information. In the vision-based approach, we need to perform a feature extraction from the raw data, which increases the complexity of the overall system[15].

Figure 1.5 made by [15] show the processing steps needed in a sensor-based system for SLR. It is exactly the same as in the vision-based approach, except for the first step, "data acquisition," since the input device is no longer a camera but gloves with sensors.

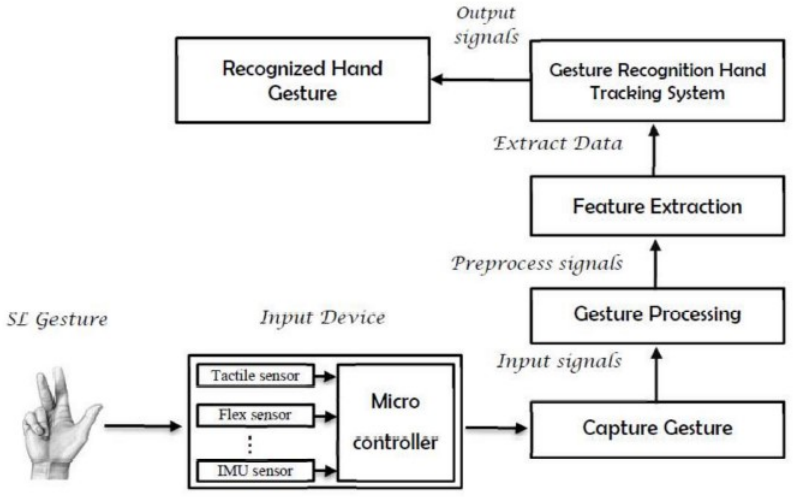

FIGURE 1.5: Sensor-based approach, [15]

In [6], the authors explain that there are three types of technology in the sensor-based approach: data glove, electromyography (EMG), WiFi, and radar.

- Data Glove The first technology used to recognize sign language was the data glove. As we just said before, they use different sensors to capture the necessary information.
- Electromyography (EMG)

  EMG involves capturing the electrical signals produced by the muscle tissues through electrodes either placed on the skin or directly attached to the muscles.
- WiFi and Radar

  One of the possible WiFi technologies is the Wisee technology, which uses multiple antennas to detect the user's gestures through changes in the frequency of a wave. It is considered much simpler than Kinect technology because WiFi does not require you to see the sign directly and can go through walls. In comparison, radar technology uses the difference in magnitude and doppler shite to extract features. This technology offers way more flexibility in terms of orientation than the vision-based approach because it is not mandatory to face the camera.

Regardless of the type of technology used, the sensor-based approach has several benefits [15] :

- The first and most important one is that it is much simpler to obtain the relevant data.
- The second one is that gloves are also way more comfortable for the user and have high mobility, which makes them easy to carry.
- Sensor-based data acquisition is, in addition, an extremely active area of research because this technology can be useful in various fields such as SLR, substitutional computer interfaces, socially assistive robotics, immersive gaming, virtual objects, remote control, medicine-health care, and many more, and this guarantees rapid advances in the field.

This approach also has its drawbacks, and we can divide them into two categories: problems linked to devices and those linked to users.

- Drawbacks linked to devices

  The main drawback related to devices is simply the price of the gloves. Many of them are on the market, but prices can vary from $1,000 to $20,000, which is very expensive. The second problem is the portability of the devices: to translate signs with gloves, you need a PC, which makes the whole thing difficult to transport. Even though it is easy to gather hand-based information, it is not possible to have access to some important parts of the body, such as the arms and elbows. This can cause an inability to recognize certain signs.

  Another problem is related to the quality and quantity of sensors. It is much better to use more and better-quality sensors because bad sensors will generate noise. With poor-quality sensors, a lot of relevant information will be lost. This can negatively affect the overall performance of the system.

  The final device-related problem is what is called "calibration.". This issue is due to the fact that every person has a different anatomy: finger extent, size of the hand, and thickness. Each glove needs therefore to be "calibrated" '(adapted) for users.

- Drawbacks linked to users

  There are two limitations in this category. The first one is due to variations in sign execution by signers. These variations affect the angles of finger joints or simply the positions of the hand and fingers. These variations impact the performance of the SLR system again.

  The second limitation is, like for devices, the need for calibration. Each person has different hand sizes, which affects the performance of gloves equipped with sensors. If gloves aren't suitable or don't suit the user, this can have a serious impact on acquired data and, therefore, on overall system performance.

1.3.3 Hybrid Based Sign Language Recognition

The last approach employs a combination of the first two approaches to collect the data by combining a camera and glove system, as explained in [15]. This technique was created to optimize the use of inertial sensor measurements, thereby improving the utility of the collected visual data. Since this approach is not used frequently due to the high cost of the necessary equipment, not much can be said about it. This is, however, perhaps a path for future developments.

1.4 Challenges In Sign Language Recognition

In this section, we present challenges that vision-based gesture recognition faces and, therefore, with which SLR systems have to deal. SLR systems face, in addition, their own challenges, which we will also describe.

In the literature review written by Al-Shamayleh et al [17], the authors propose a typology of challenges that vision-based gesture recognition faces. Figure 1.6 summarizes this typology. The different challenges are divided into three categories: system, environment, and gesture challenges. and each category has its own characteristics.

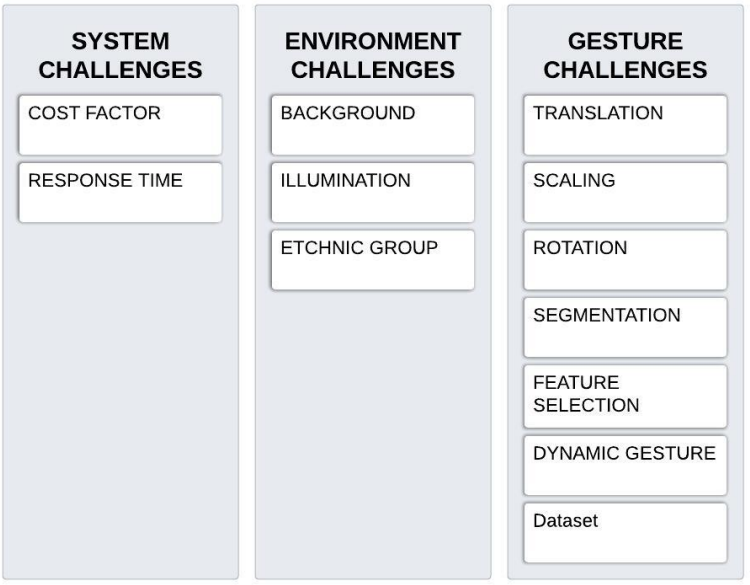

FIGURE 1.6: Vision-based Gesture Recognition Challenges

1.4.1 System Challenges

The first type of challenge is denominated "system challenges" and concerns all requirements and expectations that the users might have towards the performance of the developed system. We have two challenges in this category:

1. Cost factor

   In order to perform gesture recognition efficiently, special equipment is required. For example, in the case of the vision-based approach, it is necessary to use an excellent camera to benefit from high image quality. In the case of the sensor-based approach, data gloves and sensors are indispensable to having the necessary input. All of this can lead to high costs, which can limit the diffusion of the technology.

2. Response Time

   In order to have an efficient gesture algorithm, it's important that the response time be fast (at least fast enough to be usable). The global time to execute the algorithm and to obtain the meaning of the gesture must suit real-world use.

1.4.2 Environment Challenges

This type of challenge is particularly present during the creation phase of a dataset. It concerns, more specifically, anything that can alter the environment during video recording, such as the illumination, background, and ethnicity of the signer.

1. Illumination

   It concerns natural or artificial changes in lighting due to internal or external factors.

2. Background

   The background includes everything that surrounds the person signing during the video recording.

3. Ethnicity of the signers

   Ethnicity is an important challenge to tackle in gesture recognition. The algorithm must be indifferent to a person's ethnicity, while morphology undergoes significant variations between ethnic groups.

Since most of them are present during the creation phase of the dataset, they can significantly affect the quality of the dataset, which in turn affects algorithm performance. A high-quality dataset is one that contains videos with many different signers and that contains light and background changes [7]. There is really a need for numerous videos to increase the reliability of the algorithm.

1.4.3 Gesture Challenges

The gesture challenges concern all the things around the gesture itself, its structure, and its variations.

1. Translation

   The translation challenge pertains to the spatial orientation of hands within an image. These shifts can occur through the adjustment of the camera or just by changing hands locations.

2. Scaling

   In the vision-based approach, an essential requirement is to make sure that the camera can properly capture the hands. If the hands are too far away from the camera, it can greatly affect the overall recognition rate.

3. Rotation

   Rotation movements can be very difficult to manage, whether it is hands or any gesture rotation.

4. Segmentation

   In vision-based gesture recognition, it is important to be able to extract precisely the portion of the image of interest. This is segmentation: if you're interested in the position of hands or faces, you must be able to perform an accurate segmentation of these elements. Segmentation can directly affect the overall performance of the SLR system and, especially, the classification rate of signs. Without enough information, the system will underperform; with too much information, it will struggle to extract the correct features. The best way to address this challenge is, as indicated in [17], to be able to recognize the foreground from the background.

5. Feature Selection

   As its name suggests, the feature selection challenge is about the selection of the desired features. In an image, there's a large amount of information, but as we said in the "Segmentation" challenge, dealing with too much information is detrimental to the overall performance of the SLR system. It is particularly important to choose the right features to acquire and, simultaneously, to be able to run the system in a reasonable time frame.

6. Dynamic Gesture

   In [7], dynamic gesture as defined as "a gesture whose semantics do not only rely on the hand and arm configurations but also in the movements performed by the signers.". Dynamic gestures are much more complicated to manage because you have to take into account the movement of the gesture. This adds a time dimension and, therefore, much more information to process. In sign language, most signs are dynamic gestures by nature. The SLR system must therefore take into account movement epenthesis and co-articulation, as explained hereafter.

7. Dataset

   The last challenge in the gesture category is the dataset. As we explained just before, the size and variety (example of a sign with different background, illumination, etc.) of the dataset play a crucial role in the robustness and performance of the overall system. Obtaining a suitable dataset is the first and most important step in gesture recognition. Without it, nothing can be achieved.

   Bragg et al. [18] explain that there are three problems that need to be addressed to have a good dataset. The first two, size and variety, are also pinpointed in Fink's article [7]. Bragg stresses that the need for size and variety is due to the fact that modern machine learning techniques work better on large datasets. That contains a lot of examples. Variety can be based on gender, age, clothing, geography, culture, skin tone, body proportions, disability, fuency, background scenery, lighting conditions, camera quality, and camera angles [18] to depict precisely the situations met in the real world.

   The third issue is the use of novice signers in the dataset and of professional signers who are not native signers. They often change the execution of signs by simplifying style and vocabulary, which prevents the system from being used in real life.

1.4.4 Challenges Specific to SRL

We have seen in the previous section what challenges are present in both SLR and gesture recognition, but SRL has its own challenges. We can cite three of them: movement epenthesis, co-articulation, and the numerous types of signs. We describe each of them in detail.

1. Movement Epenthesis

   In [19], the authors explain that sign languages are "a sequence of gestures performed one after another without any pauses, like spoken language." A sign is thus a sequence of gestures divided into different phases: three for isolated signs and four in the case of continuous signing. The three common phases are:

   - Preparation phase
     The preparation phase is defined as "the movement of the hand from rest position to the signing area in front of the body" [19].
   - Stroke phase
     The stroke phase is defined as "the movement describing the word (original sign)" [19].
   - Retraction phases
     The retraction phase is defined as "the movement of the hand back to rest state" [19].

In continuous signing, the fourth phase is named "transition" and lies between the stroke and retraction phases.

- Transition phases
  The transition phase is defined as :"the hand movement after the ending of one sign to the position from which the subsequent sign will be performed" [19].

The transition phase represents exactly the challenge of "movement epenthesis." To determine the start and end of a sign is a very difficult task and one of the most difficult challenges in SLR. Those movements are not signs; they have no meaning, but automated sign language recognition systems mistake them for signs. There is really a need for new techniques that allow us to identify and ignore movement epenthesis to increase sign language recognition.

2. Co-articulation

   The second big challenge of sign language recognition is co-articulation, which is due to the starting and ending positions of a sign changing depending on the context.

   In [7] [20], the authors explain that co-articulation comes from the fact that signers tend to change the ending position of the current sign to anticipate the next sign. These changes can affect the hand shape, movement, and position of the sign, thereby making the recognition of signs within sentences more complex.

3. Numerous types of signs

   In sign language, as we have seen before, signs are divided into multiple classes: the fully lexical sign, the partly lexical sign, which contains pointing signs and depicting signs, and finally the non-lexical sign.

   In [21], the authors report a study done on Australian sign language (Auslan corpus), which showed that in this database, there were 70.2% lexical signs, 12.3% pointing signs, 11% depicting signs, and 6.5% non-lexical signs. French sign language conversations are more or less composed of 75% lexical signs and 25% of depicting signs.

   The challenge here is to be able to create a tool capable of recognizing any type of sign, where each has its own characteristics and likelihood of being used in real life. Lexical signs are the most common of all, and they are therefore easier to recognize as datasets are often filled with more examples. On their side, depicting signs is really tricky to deal with because instead of using lexical signs that are associated with a unique label, they are used to describe a situation that highly depends on the context. Even lexical signs can be difficult to handle because a slight change in the manual feature can totally change the meaning of the sign. In [7], they give this example: "signers would rather execute the sign walk quickly instead of using sequentially the signs walk and fast."

In the next chapter, we will describe the theory behind each of the models and dimensionality reduction techniques used in this work.

Chapter 2

Theory And Models

Chapter two presents the theory behind the techniques we will be using in this master thesis. This chapter will contain two sections, The first of which will present the deep learning technique that enables sign recognition, or more precisely, the classification of the sign into the correct class, and the second section will present dimension reduction techniques.

2.1 Deep Learning Methods

In this first section, we will explain all the techniques we use in this master thesis that belong to the class of machine learning techniques that use artificial neural networks to automatically learn the data. These techniques had a hard time getting off the ground in the early days, as they required very large amounts of data to operate, but since the advent of large, publicly available datasets, they have become extremely popular. Deep learning methods are built to model relationships within large datasets and extract the right features through multiple layers of neurons. These methods are used in multiple domains, such as computer vision, natural language processing, speech recognition, reinforcement learning, and so on. In the case of our master thesis, we use these methods to perform the task of classification.

2.1.1 Artificial Neural Network

The artificial neural network (ANN) is a model based on the biological functioning of a neuron network. In machine learning, this network is represented in the form of an oriented, weighted graph. It has many layers of neurons (nodes) linked with each other thanks to some weighted connections (the edges of the graph). The first layer is an input layer, usually composed of one node per variable. The last layer is the output layer. It can either be one single node, mostly for binary output, or multiple nodes in the case where there are many classes that can be predicted. The layers in between are called hidden layers. They are responsible for most of the transformations. This model is generally used for class recognition [22]. Figure 2.1 shows the architecture of an ANN with multiple inputs and hidden layers.

The first step of ANN is the feed-forward network. This is the forward propagation of the data. It happens mainly in the hidden layers. There are two phases [24], the first phase is the pre-activation phase. This is where they compute the weighted sum of all the inputs of the node, to which we add a bias term. If the pre-activation passes the threshold, then the activation phase begins. The activation is a non-linear function that transforms the data. There are many possible functions, but we use two of them: the sigmoid and Rectified Linear Unit (ReLu) functions. The equation of the sigmoid

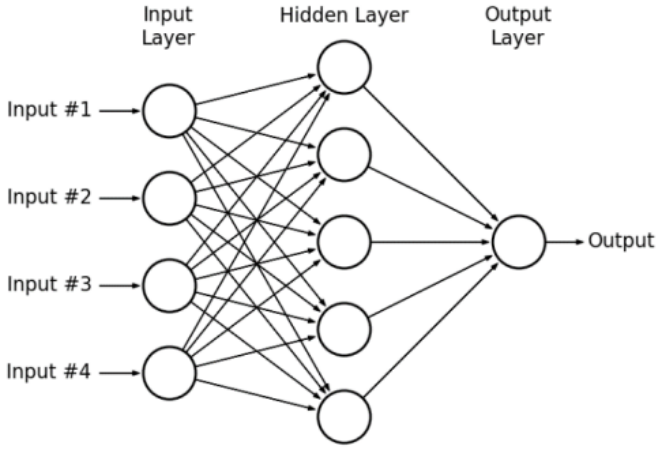

FIGURE 2.1: Architecture of an ANN from [23]

function is [22] :

$$Sigmoid(a_i) = \frac{1}{1 + e^{-a_i}} \tag{2.1}$$

where a_i represents the pre-activation function of the layer i

The equation of the ReLU function is [25]:

$$ReLu = max(0, z) \tag{2.2}$$

where z is the input value to the neuron.

During this first step, each connection in the network, also called an edge [26], is associated with a weight that determines the strength of the connection between the two neurons. It means that for $h_{l,i}$, the output of the ith neuron of the lth layer, its value is then given by [26]:

$$h_{l+1,i} = \sigma \left(\sum_{j=1}^{N_l} w_{l,j,i} h_{l,j} + b_{l,i} \right) \tag{2.3}$$

Where :

- $w_{l,j,i}$ is the weight of the edge between node j of layer l and node i of layer $l + 1$
- $b_{l,i}$ is the bias term of the neuron
- σ is the activation function

The next step is the improvement of the results. The aim of this is to minimize the error function in order to achieve the best accuracy of prediction.

This directly leads to the third step of the ANN model: back propagation. The aim of this mechanism is to update the neural network based on his performance. It is a tuning of the weight based on the error rate. To compute this, the derivative of the activation function is used [22].

The first forward propagation is done with random weights that are then updated thanks to the back propagation based on the results obtained. This is called the gradient descent method. But the weight is dependent on the output function and the last hidden layer.

2.1.2 Convolutional Neural Network

The Convolutional Neural Network (CNN) is a neural network particularly efficient with image, speech, or audio signal inputs. It is based on three main types of layers: the convolutional layer, the pooling layer, and the fully connected (FC) layer.

The convolutional layer is the core of the CNN. It is composed of three elements: the input data, a filter, and a feature map. In the case of a color image, the input data is a 3D matrix of pixels with the depth corresponding to the RGB of the image. The filter, also called kernel, is a 2-dimensional matrix (usually of size 3×3) [27]. This filter moves across the input matrix, from left to right and top-down [28]. The filter is applied to an area of the input matrix, and a dot product is computed between the input and the weights of the kernel [22]. The result is stored in an output array. This is called convolution [27]. Then the filter moves by a stride whose length is determined in advance. Each layer of the depth of the input matrix uses its own kernel. The output is the sum of the weighted input plus a bias term [28]. When all the input data has been treated, the final output array of dot products is the feature map. After each convolution, the CNN applies a ReLU (Rectified Linear Unit) activation function to the feature map, introducing non-linearity in the model. The first convolutional layer will capture low-level features. A convolutional layer can be followed by another one. The more we add convolutional layers, the more we will capture high-level features.

The pooling layer, also working with the ReLU function, has a functioning similar to the convolutional layer [22]. The principal action of the pooling layer is dimensionality reduction, which reduces the parameters in the input. The pooling layer also works with a filter, but it is not weighted this time. An aggregation function is applied to the value instead. There are two main types of pooling: max pooling and average pooling [27]. We decided to use the Max pooling function, which selects the highest value of the area. The advantage of Max pooling is that it reduces noise. Though a lot of information is lost in the process, it reduces the complexity, improves the efficiency, and avoids overfitting by keeping only the dominant features [27].

Finally, the FC layer is always the last layer of a CNN, as it is the output layer. It is fully connected to all the nodes in the previous layer. The FC layer flattens the features extracted through all the previous layers into a column vector and can feed this into a classification method [22], [27]. The classification process can be seen as an ANN since it usually uses a Softmax function and a feed-forward and backpropagation system [27]. Figure 2.2 shows the architecture of a typical CNN.

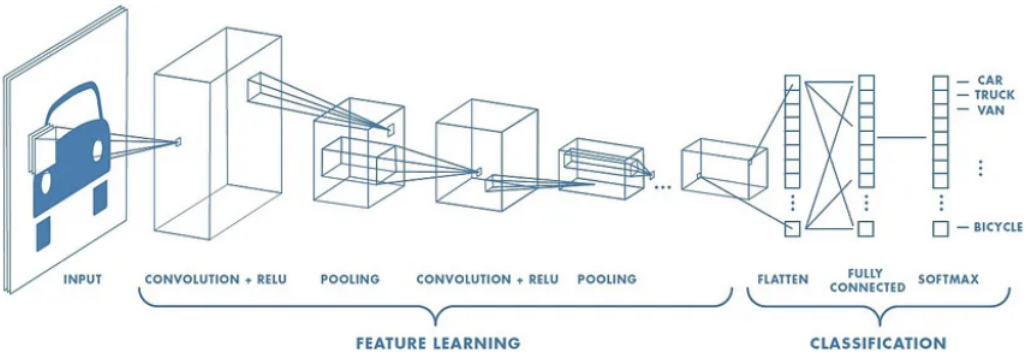

FIGURE 2.2: Architecture of a CNN from [29]

The neural networks often suffer from an increased volume of space as the number of dimensions increases, also named the "curse of dimensionality" problem [30]. However, thanks to pooling and convolution, the CNN only works with abstract representations of the features, which decreases the dimensionality [30].

Next to this, the drop-out layer prevents the model from overfitting. A drop-out layer is a sort of "mask that randomly nullifies the contribution of some neurons towards the next layer" [30]. It can be applied to any layer except the output layer. The dropout rate usually varies between 0.5 (hidden layers) and 0.2 (input layer) [31]. It is particularly important during the first batch of training, as this batch has a great influence on determining the weight of the neurons [30]. During the prediction phase, the dropout layers are not used. This leads to the final weights being larger than expected. To deal with that, weights are first scaled by the dropout rate [31].

2.1.3 Recurent Neural Network

Recurrent neural networks are a type of neural network used in supervised machine learning models that are made of artificial neurons with one or more feedback loops [32]. The main difference between a basic artificial neural network is the presence of these feedback loops, which are defined in [32] as "recurrent cycles over time or sequence." As for any neural network, it is necessary to have a training dataset to train a recurrent neural network that contains a set of input-target pairs. Of course, the goal is to minimize the disparity between the network output and the target values by adjusting the weights of the artificial neurons contained in the neural network.

Recurrent neural networks have become increasingly popular over the years for their ability to effectively model sequences of varying lengths. It means that this type of neural network is really suited to recognize patterns in sequences of data such as video, speech, language, and time-series data [33]. Recurrent neural networks are also used in other tasks such as language modeling and learning word embeddings [34].

The baseline architecture recurrent neural network is composed of three layers, as in an artificial neural network: an input, hidden, and output layer. Figure 2.3 that comes from [33], the input layer is represented by the node with the letter "x", the hidden layer by the nodes with the letter "h", and the output layer by the node with the letter "y". As explained in [33], the difference between a RNN and a classic ANN is that in this architecture, there are some loops. In other words, the nodes in the hidden layer are interconnected, which is not the case with a classical neural network. It means that the previous node in the hidden layer has an influence on the next node in the hidden layer.

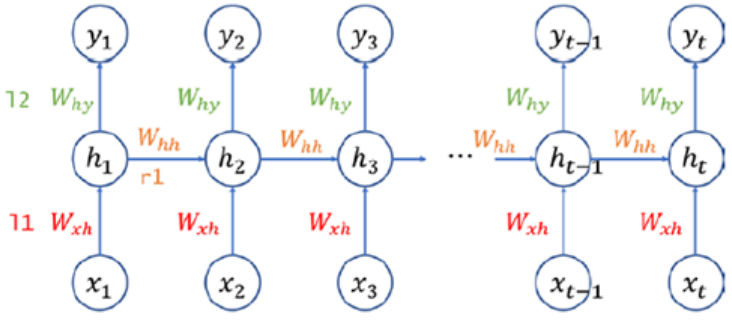

FIGURE 2.3: Architecture of a basic RNN [33]

This approach guarantees that the output at time n depends on the inputs from time $t = 1$ to time $t = n$. In other words, it is a sequence of data that determines the output and not only an individual data point. For example, the output1 depends on the input1, and the output3 depends on the inputs1, input2, and input3. For the hidden layer, it is a bit different because a hidden layer node depends only on the previous hidden node and its own input layer node. For example, the hidden2 node depends on the input2 node and the hidden1 node [33].

In terms of equations, to compute the value of a hidden layer of the output layer node, we use these formulas [33]:

$$h_t = tan((L_1(x_t) + r1(h_t - 1)) \tag{2.4}$$

$$y_t = L_2(h_t) \tag{2.5}$$

where L_1, L_2, and r_1 represent the weights of the nodes in the input, hidden, and output layers.

We are now going to present the two types of RNN that we will use in this master thesis.

2.1.3.1 Long Short-Term Neural Networks (LSTM)

The first type of recurrent neural network that will be explained is long-short-term memory. This model was created to address the issues of vanishing, exploding gradients, and the problems of training over a long sequence and retaining memory.

As we said earlier, a recurrent neural network improves the performance of a neural network by incorporating the ability to understand sequential dependencies [32]. But recurrent neural networks have one big problem that appears during backpropagation. It is the feedback loops that allow a RNN to retain information over time; the training of a RNN can be very difficult when the model has to face lengthy sequences that require long-term temporal dependencies. This problem is called the "vanishing gradient problem" [33]. In other words, it means that the weights calculated during the backpropagation can, if they are dealing with lengthy sequences, vanish or explode. If they vanish, it results from the multiplication of small values (less than 1). On the contrary, if the weights explode, it results from the multiplication of numerous large values (exceeding 1). Since all classical RNN failed to address these problems, in order to lean long-term sequential data, the long-short-term memory was designed [33].

The architecture of an LSTM is thus different from that of a classical RNN. The LSTM contains "memory cells," whose role, as explained in (3), is to "maintain the information in memory for long periods of time." The inputs and outputs of these memory cells are controlled by what we called "gates." Their objective is to "control the flow of information to hidden neurons and preserve extracted features from previous timesteps." [32]. As shown in Figure 2.4 that come from [32], a typical LSTM is made of an input, output, and forget gate in each memory cell. It is impossible for the network to remember all the information and thus need to decide which becomes irrelevant at some point, but the network cannot detect by itself the unnecessary information. This is the role of the "forget gate.", this gate as it is there to "control the rate at which the value stored in the memory cell decays" [32]. When the input and output gates are not working and the forget gates do not cause decay, the memory cells keep their value in memory over time.

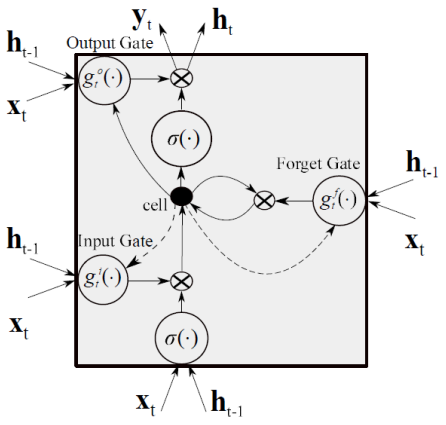

FIGURE 2.4: Architecture of a memory cell in a LSTM

A LSTM can have four different architectures: one to one, one to many, many to one, and many to many models. In the case of our master thesis, we use the many-to-one model, and this is the one we are going to explain in a little more detail [33]. The authors of the book "Deep learning with applications using Python" [33] explain that this model uses many inputs to create only one output. In Figure 2.5 that comes from [33], we can see that there are many inputs represented by the letter x and only one output represented by the letter y.

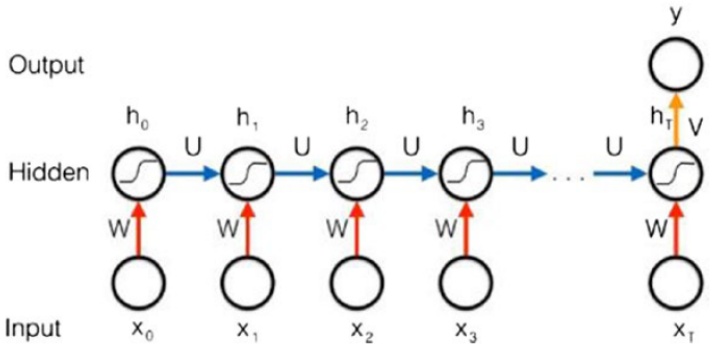

FIGURE 2.5: Architecture of many to one LSTM model

Even if the LSTM helps solve RNN problems, it has his disadvantages. Sankar in his article [32] explains that the architecture of the hidden layer of a LSTM is highly complex, there are four times more parameters in a LSTM than a classical RNN. The fact that there are so many more parameters makes it much more time-consuming to train.

2.1.3.2 Gated Recurrent Unit (GRU)

The second type of RNN that we use in this master thesis is the Gated Recurrent Unit. In comparison to the LSTM, which has a complex architecture with numerous memory cells, the GRU contains much fewer parameters. Like the LSTM, a GRU captures the information of a sequence unit with gating units that modulate the flow of information but without the presence of distinct memory cells [32]. Figure 2.6 that comes from [32], we can see that in a GRU, the gate z decides whether the hidden state should incorporate a new hidden state h, and the gate r decides if we ignore or not the prior hidden state.

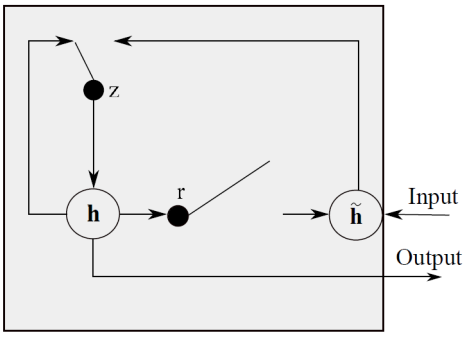

FIGURE 2.6: Architecture of GRU

2.2 Support Vector Machine

The third model that we will present in this part is the support vector machine (SVM). We will first describe the baseline model, the linear SVM, then the SVM we use. The SVM is a well-known model used in supervised binary classification. This model was built to "construct an optimal separating hyperplane between two perfectly separated classes." [35]. His approach is to use the concept of margin, which is "the smallest distance between the decision boundary and any of the samples." [22]. The goal is to maximize the margin, and to compute it, we use the Euclidean distance between the closest observations (called the support vector) and their orthogonal projection on the hyperplane, as shown in Figure 2.7.

Let suppose we have a set of data of N pairs :

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

where $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$.

The equation of a hyperplane is, as shown in Figure 2.7:

$$f(x) = x^T\beta + \beta_0 = 0 \tag{2.6}$$

where β is a unit vector.

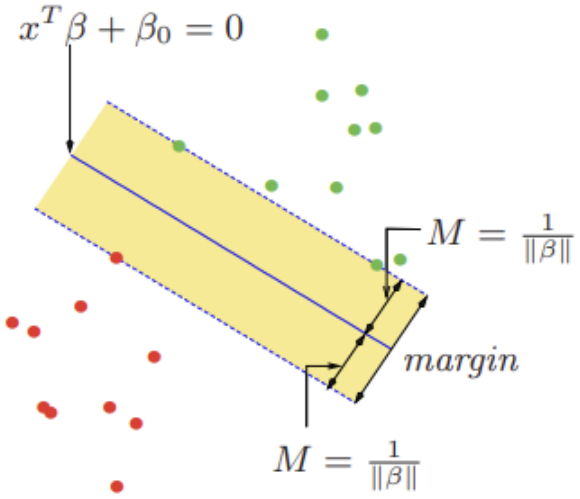

FIGURE 2.7: Architecture of a SVM from [35]

The problem can be formulated as this [35]:

$$\max_{\beta, \beta_0, \|\beta\|=1} M \tag{2.7}$$

$$\text{subject to} \quad y_i(x_i^T\beta + \beta_0) \geq M, \quad i = 1, \ldots, N \tag{2.8}$$

In our master thesis, we have two problems . that make it impossible to use linear SVM. The first is that our data are not linearly separable, and the second is that we have much more than two classes.

To solve the non-linearity problem, the dataset is projected into a higher-dimensional space in which it is linearly separable. This is done by using the kernel trick. That trick allows you to work in a higher-dimensional space without ever computing the coordinates of the datapoint in the higher-dimensional space. Then, with the feature space and the kernel trick, we can work in higher-dimensional space ϕ but this mapping ϕ is unknown

and we don't need to know it. In this work, the kernel function used is the Gaussian RBF function because this kernel function works with pairwise distance [22] :

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{2.9}$$

where σ represents the standard deviation of the RBF kernel.

To solve the problem of multi-class classification, there are two techniques :

1. One versus One (OVO) : In this technique, one class is compared to each of the other classes. The goal is to find the hyperplane that best separates the two classes, but this is done with all the possible combinations. It means that if we have 10 classes, we will have 45 hyperplanes.
2. One versus Rest (OVR) : In this technique, one class is compared to all the others. The goal is to find the hyperplane that best separates one class from all the others. It means that if we have 10 classes, we will have 9 hyperplanes.

In Appendix 3, we can find an example of the OVO and OVR technique.

2.3 K Nearest Neighbors (KNN)

KNN is a supervised learning model that predicts the class of a new observation based on its k nearest neighbors. To choose which class to give to a new observation, KNN uses a majority voting system to allow the classes of the k observations closest to the new observation to be selected. To find out which are the closest observations, there are several ways of calculating distances, such as the Euclidean distance, which is the one that we use in our work because it is one of the most used, the Manhattan distance, the Minkowski distance, the Hamming distance,...

Figure 2.8 shows how a KNN works.

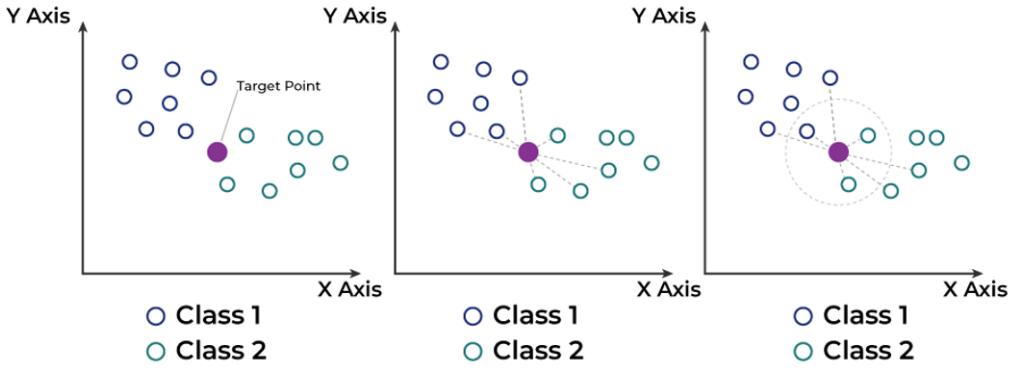

FIGURE 2.8: Classification with a KNN [36]

The equation of the Euclidean distance is :

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \tag{2.10}$$

2.4 Dimensionality Reduction Techniques

Dimensionality reduction plays an essential role in examining data with numerous dimensions.The goal of these technique is [37] "to reduce the distance in a latent space between distributions of different data sets to allow efficient transfer learning". In other words, this means that it consists of reducing the number of features related to a certain class without losing the capability to recognize this class. In the literature, many studies support the fact that classifiers (models used to classify) are less preferable when no reduction dimension has been made because the reduced-dimensional representation of the original data helps mitigate the challenges posed by the curse of dimensionality.

There are many different dimension reduction techniques, such as principal component analysis, non-negative matrix factorization, isomaps, t-distributed stochastic neighbor embedding, uniform manifold approximation and projection for dimension reduction, autoencoders, multidimensional scaling, and so on. Each of these techniques is based on one or more criteria to reduce the dimensions [38].

There are many advantages to using dimension reduction techniques, we list seven of them [37]:

- "Decrease the number of dimensions and data storage"
- "It requires less time to compute"
- "Irrelevant, noisy, and redundant data can be deleted"
- "Data quality may well be optimized"
- "Allow to visualize data"
- "It simplifies the classifications and increases performance as well"
- "Help an algorithm to work efficiently and improves accuracy"

All size reduction techniques fall into two categories : feature selection and feature extraction. The purpose of the first type is to address dimensionality issues by minimizing redundancy in the data, deleting unnecessary data, and improving the interpretability of the outcomes. On the other hand, feature extraction techniques are not used to remove dates but to find the most relevant features, the ones that contain the most information, to improve the efficiency of the algorithm [37].

2.4.1 Principal Components Analysis

The first technique of dimension reduction that we will present is principal component analysis (PCA), which is one of the most widely used techniques in dimension reduction. It can be define as "the orthogonal projection of the data onto a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized" [22]. In other words, the goal is to reduce the dimensionality of a dataset while retaining the most important information. By converting the data into a new set of variables called the principal components, which are uncorrelated, PCA preserves the variability (variance) of the original data for as long as possible.

Consider a set of observations $\{x_n\}$ where $n = 1, \ldots, N$ with dimensionality D, the goal is to project our data into a sub space dimensionality $M < D$ while maximize the variance of the projected data.

Firstly, consider projecting onto a one-dimensional space. To define the direction of this space using a D-dimensional vector u_1 which is a vector unit $u_1^\top u_1 = 1$. Then, each data

of the dataset x_n is projected onto the scalar value $u_1^\top x_n$. As we said before, the goal is to maximize the variance of projected data which is :

$$\frac{1}{N}\sum_{n=1}^{N}\left(u_1^\top x_n - u_1^\top \bar{x}\right)^2 = u_1^\top S u_1 \tag{2.11}$$

where :

- $\bar{\mathbf{x}}$ is the mean of the projected data given by this formula [22]:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{2.12}$$

- $\mathbf{S}$ is the data covariance matrix given by [22]:

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{x}_n - \bar{\mathbf{x}}\right)\left(\mathbf{x}_n - \bar{\mathbf{x}}\right)^\top \tag{2.13}$$

To maximize the variance of the projected data, we use a Lagrange multiplier λ_1 to obtain a unconstrained maximization of :

$$u_1^\top S u_1 + \lambda_1(1 - u_1^\top u_1) \tag{2.14}$$

By solving this maximization, we find this stationary point [22]:

$$S u_1 = \lambda_1 u_1 \tag{2.15}$$

where u_1 is the eigenvector of S.

By pre-multiplying the previous equation by $u_1^\top$, we can rewrite this equation like this, which give the value of the variance [22]:

$$u_1^\top S u_1 = \lambda_1 \tag{2.16}$$

Finally, the variance reaches its maximum when $\mathbf{u}_1$ is equal to the eigenvector corresponding to the largest eigenvalue. The eigenvectors are the principal components. In the general case of an M-dimensional projection space, the ideal linear projection is determined by the M eigenvector $u_1, \ldots, u_M$ of the data covariance matrix S corresponding to the M largest eigenvalues $\lambda_1, \ldots, \lambda_M$ [22].

2.4.2 Auto-Encoder

Auto-encoder, also known as auto-association, is a dimension reduction technique that uses the implementation of an artificial neural network to reduce the number of dimensions.

Figure 2.9 that come from [39] shows the architecture of a auto-encoder. It is composed of two parts: an encoder and a decoder. The purpose of the encoder is to transform the high-dimensional data into low-dimensional data, whereas the decoder does the exact opposite, transforming the low-dimensional data into its original high-dimensional data. In the figure, the compressed layer d contains the most important information that comes from the original data. The overall goal of the auto-encoder is to adjust the weights for each unit to ensure that the outputs are as close as possible to the inputs [39].

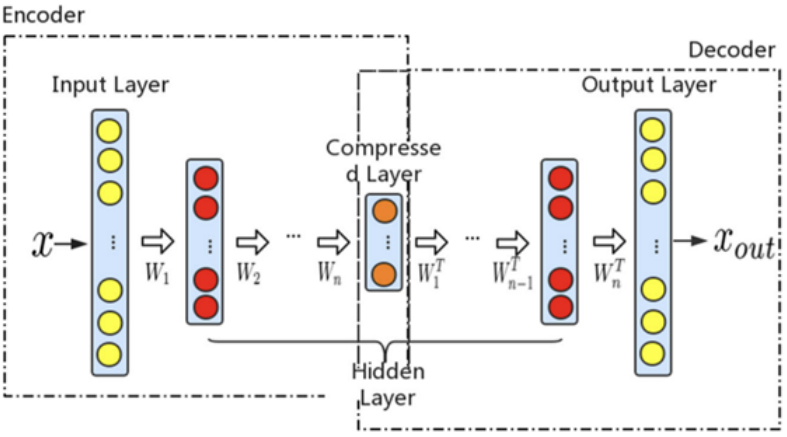

FIGURE 2.9: Architecture of an auto-encoder [39]

In terms of equations, we obtain the value of the encoder and decoder parts with these equations [39] :

$$E(x) : W(x) + b \tag{2.17}$$

$$D(x) : W^T(x) + b' \tag{2.18}$$

W and b represent the weight and bias matrix of the encoder in equation 2.17, while in equation 2.18, W^T and b' represent the weight and bias matrix of the decoder.

To identify the difference between the x and the x_{out}, an auto-encoder employs a loss function such as the mean absolute error (MAE) or the mean square error (MSE). While the MAE represents the mid-value data, the MSE reflects the mean value of the data.

In order to find the best low-dimensional space, the method that the auto-encoder uses to find its weights is extremely important. The weights can be performed randomly, but often, the results deviate considerably from the desirable outcomes. There are other methods that an auto-encoder can use to optimize its weights, such as the restricted Boltzmann machine (RBM). This method uses the underlying data structure to perform backpropagation to find the best possible weights [39].

2.4.3 Singular Value Decomposition

Singular value decomposition (SVD) is a dimension reduction technique that involves the decomposition of a matrix into three other matrices. SVD provides reduced-rank approximations for both dimensions of the original matrix (m, n, for the m rows and n columns). Here is the SVD equation :

$$A = U\Sigma V^T \tag{2.19}$$

In equation 2.20, A is the original matrix of dimension (m, n).

U is an orthogonal matrix of dimensions $m * m$. The columns of this matrix define the left singular vectors of the original matrix A. The matrix U is obtained from the eigenvector matrix of $A^T A$

The matrix Σ is of dimensions $m * n$, where the diagonal contains the singular values of the original matrix A, and the rest of the matrix is filled with zeros. The singular values are placed in decreasing order on the diagonal of the matrix Σ such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$

V^T is an orthogonal matrix of dimensions $n * n$. The columns of this matrix define the right singular vectors of the original matrix A. The matrix V^T is obtained with the eigenvector matrix $A^T A$.

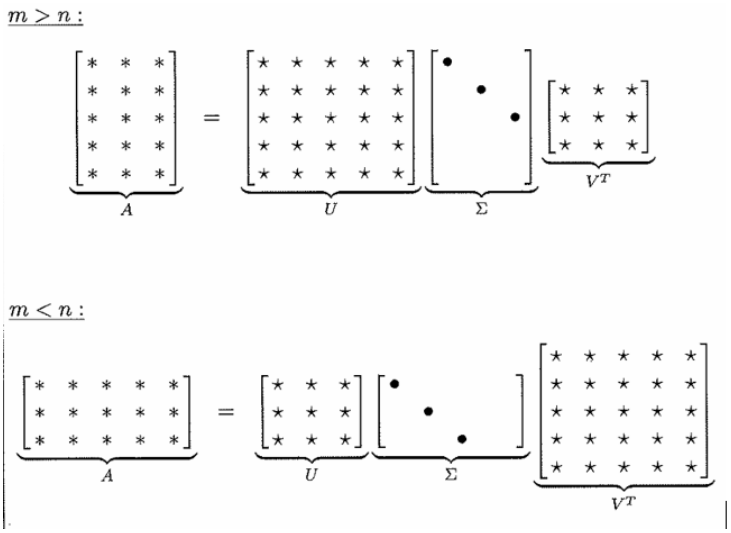

FIGURE 2.10: Architecture of SVD

Figure 2.10 that come from [40], we can see how the matrix Σ is filled. When $m > n$, rows beyond the value of n will be filled with zeros. In the opposite case, when $n > m$, the columns beyond the value of m will be filled with zeros [40].

To approximate the original matrix with the SVD, we simply keep the first k singular values of the Σ matrix and set the other singular values of the diagonal equal to zero, then use the original formula with the new Σ matrix called Σ'. We have thus this equation [40]:

$$A' = U\Sigma' V^T \tag{2.20}$$

In the next chapter, we will present the dataset used in this word and the pre-processing steps.

Chapter 3

Data Presentation & Transformation

Chapter three is divided into two sections. The first section presents the existing datasets and the University of Namur (UN) dataset we will be using. The second section presents the pre-processing we have carried out on our dataset so that we can apply our models to it.

3.1 Used Dataset

In this second chapter, we present existing datasets that are used for SLR system building. As we have seen before, the availability and quality of datasets are major challenges in this field, given the time and difficulty involved in creating quality datasets.

3.1.1 Existing Datasets

As it is explained in [7], the need for large datasets of quality for SLR has increased greatly in recent years. Existing datasets can be either isolated or continuous and typically consist of videos of people signing. These datasets are not necessarily video-based but can also be built using the sensor gloves. In the case of datasets containing videos, the format of the video can vary depending on the type of device used to record the videos.

A dataset that has been built using sensors is called a "motion-capture dataset" (Bragg 2019). To build such a dataset, sensors are attached to a signer to obtain the information. In addition, depth cameras can be used to capture 3D positioning. For example, the device "Kinect," developed by Microsoft, contains a depth sensor and is used to create a sign language dataset.

Many parameters can change from one dataset to another. Typically, some datasets are created with the help of professional interpreters who translate spoken language into signs. Some people have had the idea of building datasets based on videos available on YouTube in order to obtain data that is closer to "real life.". The goal is really to obtain more self-generated signs, because there is always a difference between a sign that is made automatically and a sign generated on request. The geographic region in which the dataset is built has considerable importance too, since sign language is country-specific. For example, most datasets are constructed using the American sign language since it is one of the most used sign languages [18].

Different existing datasets that have been built using RGB videos are explained in the article [7]. As explained above, datasets are divided into two categories: isolated and continuous. A detailed description of these datasets can be found in Appendix 2.

1. Continuous Signs Datasets
    - The Phoenix Weather Dataset
    - The Greek Sign Language (GSL) Dataset
2. Isolated Signs Datasets
    - The American Sign Language Lexicon Video Dataset (ASLLVD)
    - The DEVISIGN-L Dataset
    - Microsoft American Sign Language (MS-ASL) dataset

3.1.2 French Belgian Sign Language laboratory Corpus (LSFB lab)

The French-Belgian Sign Language Laboratory is the department of the University of Namur (UN), which built the dataset we use for this master thesis. The corpus was created to study Belgian sign language in greater detail [41] [7].

The researchers at the University of Namur have gathered and annotated many conversations since 2012 to build the dataset. It contains more than ninety hours of videos, with twenty-five of them annotated, making it one of the largest publicly available datasets. Given the complexity and variety of the Belgian sign language grammar, one hundred signers were used to build this dataset to reflect diverse age groups or geographical regions of the country. To expand the variety included in the dataset even more, UN researchers have decided to use native and non-native speakers. Figure 3.1 provides information on the gender, dominant hand, and age distributions of signers within the dataset.

FIGURE 3.1: Information on LSFB datasets

Videos in this dataset consist of nineteen tasks that the person was asked to perform to analyze a large number of discussion types, such as argumentation, narration, and explanation. The main weakness of the LSFB corpus database is that all the videos were recorded in the same controlled space without any change between videos. The result is a lack of variety in the background and lightning. This negatively affects the robustness of systems using this dataset for training. It is important to notice that the vast majority of the existing databases are built in the same way as the LSFB dataset, which means that the models that work well on the LSFB dataset can also work on other datasets.

To record the ninety hours of videos, two cameras positioned in front of the signer were used, capturing 50 frames per second. The process of annotation and translation of the video is a very time-consuming one, and so far, twenty-five hours have been annotated and five hours translated into French. To annotate videos, a separation for each hand is made, and each lexical sign (LS) has its own gloss (meaning). For example, the signs that denote proper nouns, such as the city of Bastogne, are prefixed with the suffix "NS" (name sign).

The LSFB corpus dataset is a great source of information but is hard to use with computer-automated systems. To solve this problem, researchers in Namur have decided to build two more user-friendly sub-datasets. One for continuous SLR and the other for isolated SLR[7].

3.1.2.1 LSFB-CONT

The LSFB continuous dataset contains the following information :

- All the LSFB corpus videos
- The original XML annotation files for the videos
- A preprocessed version of these annotations is stored in a CSV format.

This dataset is the largest available dataset for continuous SLR and contains 6,883 different signs.

3.1.2.2 LSFB-ISOL

The LSFB-CONT dataset contains a large number of different signs, but it is heavily unbalanced. The majority of signs are indeed only performed a few times, while the most frequently used signs are repeated hundreds of times. As UN researchers wanted to be able to study signs one by one and not together in a sentence, they created the LSFB-ISOL dataset, which is the second sub-dataset. It contains all signs that have been repeated at least forty times in the LSFB-CONT dataset.

An important piece of information about this dataset is that since the recorded signs come from a continuous dataset, they are performed faster than when signs come from an isolated dataset. Since the speakers are talking freely and are not following a script or don't have to perform signs one by one, they tend to sign faster than in other isolated datasets.

The LSFB-ISOL is the dataset that we use in our master thesis, so it's important to describe it in more detail to understand it fully. It contains 4,657 signs with a total of 120,739 observations. For each observation, we have the coordinates (x, y, z) of a certain number of points that are located in four different parts of the body. We have 21 points in the left hand, 21 points in the right hand, 478 points in the face, and 33 points in the body. It means that one observation gives us 553 (21+21+478+33) sets of coordinates. One last important piece of information about this dataset is that an observation is composed of a set of frames (images). In other words, each observation has a certain number of frames, which varies for each observation.

3.2 Pre-processing of the data

Before explaining the implementation and results of our models, it is imperative to understand the pre-processing of our data, which is a mandatory step for the proper functioning of the models. The goal is to refine and prepare our raw data, transforming it into a format that is suitable for us. In this chapter, we will explain each step that we go through to clean our data and set the stage for robust analysis and meaningful interpretations.

3.2.1 Reduction of the dataset

The first step is to make the dataset smaller. Indeed, our data consists of 120,739 observations with 4657 classes; one class is equal to one sign. We decide to use only a part of the dataset at our disposal for multiple reasons. The first one is that, given the large size of the dataset and therefore the computation time, it would be considerable if used in its entirety. The second reason is the fact that some classes are represented less than 20 times, or even just once, which is not enough for most models to recognize this sign. In consequence, we arbitrarily decided to take only part of it. The best decision given the time and material at our disposal was to use the 20 most represented classes, with a total of 26,492 observations.

3.2.2 Removing the third dimension

The data are three-dimensional arrays, with time frames on the first axis, mask points on the second, and spatial coordinates (x, y, z) on the final axes. We decided to delete the z-coordinates because of their low precision, and as the gestures were recorded facing the camera, depth information is not very useful. Figure 3.2 represents the different dimensions that we have in our data. Figure 3.3 represents the mask points that we have on the hand.

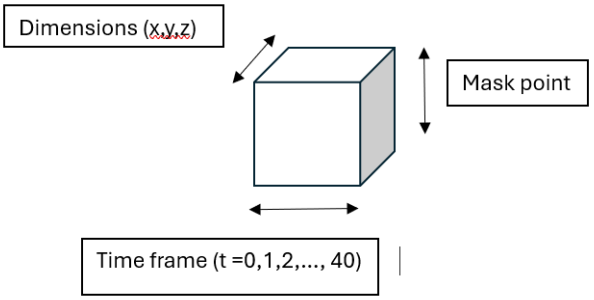

FIGURE 3.2: Representation of dimension

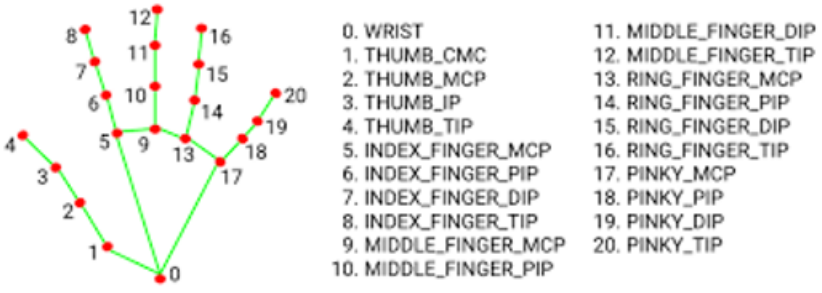

FIGURE 3.3: Coordinate of the hand

3.2.3 Uniformization of the number of frames

The different observations have been recorded with a different number of frames. To facilitate the use of the data set in the different algorithms, the sequences should have the same number of frames. To find an optimal common frame number for all observations, we calculated the average number of frames per observation, which is 23. Furthermore, in various articles, we have seen that they set the number of common frames at 40. We arbitrarily decided to follow his lead and set the number of frames at 40. It's easier to create intermediate frames than to delete frames because, when we delete frames, we lose information. Here are the formulas to normalize the number of frames.

- Formula for calculating the step between points in the new sequence :

$$step = (T-1)/(Fr-1) \tag{3.1}$$

- Calculation of the new sequence for each point :

$$Frac = step * i \tag{3.2}$$

$$Inf = int(frac) \tag{3.3}$$

$$Supp = inf + 1 \tag{3.4}$$

$$Reste = frac–inf \tag{3.5}$$

$$Nseq(i) = seq(inf) * (1 - Reste) + seq(sup) * reste \tag{3.6}$$

where Fr is the desired number of frames, Seq is the sequence to be transformed, T is the number of frames of the sequence to be transformed, and $Nseq$ is the new sequence. An example is shown below, where we go from a 5-frame sequence to a 7-frame sequence and look for the value of the third frame ($i = 2$ because we start with 0) in the new sequence.

0	1	2	3	4
2.7	3.0	3.2	3.3	2.9

TABLE 3.1: Example of frames

$$T = 5 \tag{3.7}$$

$$Fr = 7 \tag{3.8}$$

$$step = (5-1)/(7-1) = 0.67 \tag{3.9}$$

$$i = 2 \tag{3.10}$$

$$Frac = 0.67 * 2 = 1.33 \tag{3.11}$$

$$Inf = 1 \tag{3.12}$$

$$Sup = 2 \tag{3.13}$$

$$Reste = 0.33 \tag{3.14}$$

$$Nseq(2) = seq(1) * (1 - 0.33) + seq(2) * 0.33 \tag{3.15}$$

$$Nseq(2) = 3 * 0.67 + 3.2 * 0.33 = 3.06 \tag{3.16}$$

3.2.4 Structure of The Data

The datasets are separated into four parts: the points of the right hand (R), the left hand (L), the body (P), and the face (F). We have therefore created several datasets: separate datasets, R, L, P, F; datasets merged by two, FP, FL, FR, PL, PR, LR; datasets merged by three, FPL, FPR, FLR, PLR; and datasets merged all together, FPLR.

In the next chapter, we will provide a detailed explanation of the implementation process for each of the models. We will describe each step to be able to use the algorithms and also which library has been used to code it.

Chapter 4

Implementation

Chapter four presents the implementation of the different models we have used in this work. We describe in detail the many choices that we have made.

4.1 Implementation Of The Dimensionality Reduction Techniques

We first focus on dimension reduction techniques.

Datasets can be reduced in four different ways. In the first case, one can reduce the dimensions of the datasets taken together (right hand, left hand, pose, and face) with x and y coordinates taken collectively. In the second case, one can reduce the dimensions of the datasets together, but x and y coordinates taken separately. This means that there is a reduction for x and a reduction for y. In the third case, the dimension reduction can be applied to separate datasets but keep x and y coordinates taken together. In the last case, the dimension reduction is applied to the separate datasets and on the separate x and y coordinates. This means that there will be a reduction for each data point, each x, and each y.

4.1.1 Principal Component Analysis

To carry out our PCA, we first standardized the data and then performed the PCA. Here is how it is implemented practically speaking:

The data is stored on a CSV file, where the rows are the observations and the columns are the flattened coordinate matrix. As a reminder, the coordinate matrix is a 3-dimensional matrix with the temporal aspect (frames) on its first axis, the mask points on its second axis, and the spatial coordinates (x, y) on its last axis.

In order to carry out standardization and later PCA correctly, we need to change the layout of the data matrix. We need to move from a matrix with observations in rows and temporality, mask points, and coordinates in columns, to a matrix with observations and temporality in rows and mask points and coordinates in columns.

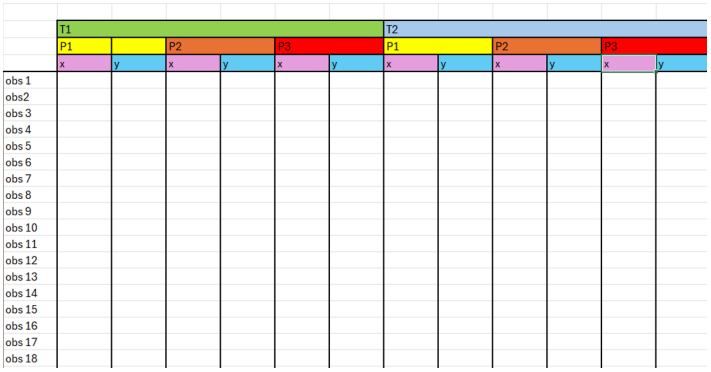

FIGURE 4.1: Orignal layout of the data

Figure 4.1 represents the original layout of our data and Figure 4.2 represents the transformed layout of the data used to perform the PCA. Figure 4.1, the T corresponds the time, and thus T1 is the first frame, T2 is the second frame (we have 40 frames by observations). The letter P is associated with the point of the mask, which means that P1 is the first point (we have in total 21 for the left and right hands, 33 for poses, and 478 for faces).

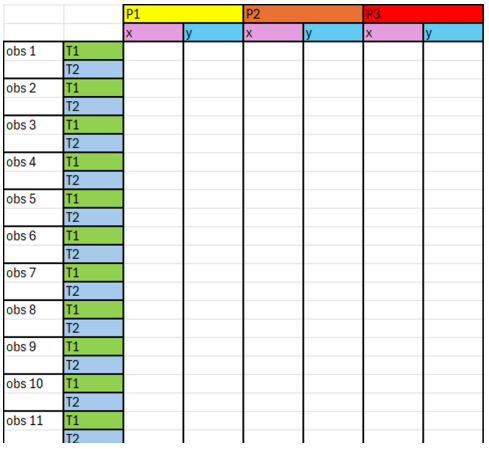

FIGURE 4.2: Transformation layout of the data

If we take, for example, the data from the right hand, which includes 26.492 observations, 40 frames, 21 points, (x, y), The data matrix would go from a dimension of (26.492, $40 \times 21 \times 2$) or (26.492, 1680) to a matrix of dimension (26492×40, 21×2) or (1.059.680, 42).

4.1.2 Auto-encoder

Before applying the auto-encoder, the data must be standardized. To do this, we perform the same data transformation as for the PCA. In other words, we go from a matrix with observations in rows and frames, mask points, and x and y coordinates in columns, to a matrix with observations and frames in rows and mask points and (x, y) coordinates in columns.

Once the data has been standardized, the auto-encoder is used for dimension reduction. The auto-encoder can adopt several different structures, depending on the type of reduction required.

Finally, we recompose the data in their original form, i.e., on the first axis, the observations, and on the second axis, the frames, mask points, and (x, y) coordinates.

4.1.3 Singular Value Decomposition

The SVD dimension reduction method must be applied to each observation. To do this, we resize each observation into a matrix that takes the frames in rows and the mask points and (x, y) coordinates in columns. We then apply the SVD, which consists of decomposing this matrix into three sub-matrices :

$$A = U\Sigma V^T \tag{4.1}$$

Next, we recompose our matrix, choosing the number of singular values needed to use in the reconstruction. The number of singular values can range from 1 to 40. We then flatten the matrix into a vector.

Finally, we apply a standardization and a PCA. For these operations, we follow the same procedure as in the PCA section.

4.2 Images processing

We now turn to images processing required to run specific models. Four types of images have been created: single-layer, double-layer, four-layer, and multiple-layer.

1. Single-layer image.
   For this first method of transforming data into an image, the aim is to represent the graph of all points at all times on the same image. To do this, we first normalize each dataset independently. Next, we create our image, which is an 80 by 80 matrix (where a matrix cell is equivalent to a pixel). Then we multiply each coordinate (which is between 0 and 1 because of normalization) by 80 to place it on the image. For each coordinate of the image, pixels that are at the right, left, top, and bottom are filled with a specific color, and pixels that are on the diagonals are filled with an other color. We do this because models like CNN that use images work best with color variations. One drawback of this approach is that, we lose the time dimension information. Figure 4.3 provides an example of the results obtained when we apply this transformation to a given sign.

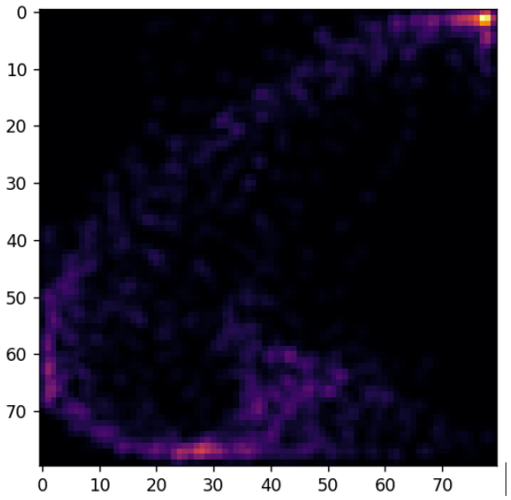

FIGURE 4.3: Single-layer image

2. Double-layer image.

   For this second method, the goal is to keep the time information. To this end, we construct two images. For the first image, its abscissa will be the x coordinate, and its ordinate will be the time dimension. For the second image, its abscissa is the y coordinates, and the ordinate is the time dimension. To implement this processing, we must first normalize all the data taken togheter. Next, we create the image, with each column representing a time frame made up of all the points in the mask. Apply this image transformation to the LR data set. Figure 4.4 provides an example of the results obtained when we apply this transformation to a given sign.

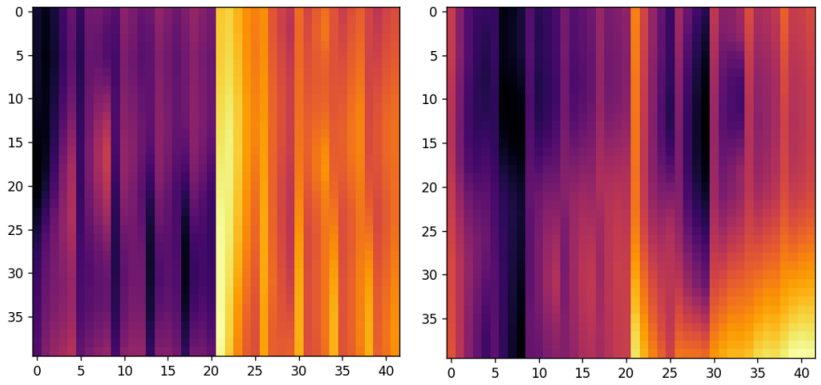

FIGURE 4.4: Double-layer image

3. Four-layer image

   This method is the same as the previous one, except that we divide the images into two, which gives us four layers. We separate the images into two because on the original one, we have both hands (in the vertical cut we see on both images in Figure 4.4 in the middle is when we switch to the other hand). This leads to the generation of an image for each hand. The outcome is four different images:

   (a) Right-hand with the time in ordinate and the x coordinate in abscissa.

   (b) Right-hand with the time in ordinate and the y coordinate in abscissa.

   (c) Left-hand with the time in ordinate and the y coordinate in abscissa.

   (d) Left-hand with the time in ordinate and the x coordinate in abscissa.

4. Image with multiple layers

   The aim of this procedure is to create a layer for each time frame. To this end, we first normalize all the data taken together in the same way as for dimension reduction . Next, we create an 80 by 80 matrix (our image) filled with zeros each time frame. We fill these matrices by changing the zeros to one at the coordinates obtained by multiplying the normalized values by 80. In the end, each observation is transformed into an image of 10 layers with dimensions 80 by 80. We are limited to 10 layers while having 40 frames due to a data storage constraints. With these 80 by 80 by 10 dimensions, images weigh 15 GB. If we used 40 frames, the file size be 193 GB, which start to be hard to store and adds enormous computing time to the various algorithms.

   When applying this image transformation to the LR data set, we obtain Figure 4.5.

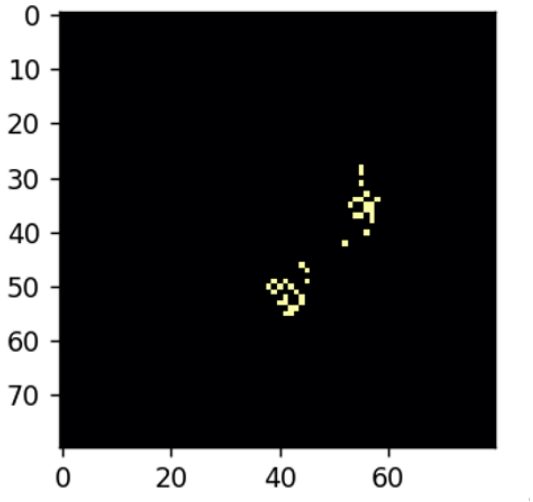

FIGURE 4.5: Multiple layers image at frame 17

4.3 Computation of the Distance Matrix

To create our distance matrix for the KNN, we calculate the Euclidean distance between each dataset. The distance can be calculated using standardized or non-standardized data. We also create two other matrices, one where we removed the first 5 and last 5 frames and one another where we removed the first 10 and last 10 frames. We do this frame removal because when a sign begins or ends, the position of the hands varies according to the preceding or following sign. Removing these frames avoids biasing our distance computations.

4.4 Implementation of the models

In this section, we present the inputs used for the models that we test as well as, the hyper-parameters to be selected to control the learning process.

4.4.1 Artificial Neural Networks

The ANN takes as input the results of the PCA, autoencoder, or SVD. Each observation is a vector (1-dimensional). The 2-dimensional matrix (time, reduced points) is indeed flattened initially.

In this model, there are six hyper-parameters to choose:

- The number of layers.
- The number of nodes in each layer.
- The dropout is the percentage of neurons switched off during the learning iteration. We use it to spread the learning over all nodes, avoiding having one node take all the weight.
- The number of iterations
- The batch size is the number of times the model is trained before the gradient descent is started, i.e., before the model is updated.

4.4.2 Convolutional Neural Network

The input for a CNN is the one-layer images that we have built. There are seven hyper-parameters to choose:

- The number of layers.
- The number of nodes in each layer.
- The dropout.
- Number of convolution layers and kernel size.
- The batch size.
- The number of iteration.
- The number of pooling layers and reduction size.

4.4.3 Convolutional Neural Network 3D

The CNN 3D is similar to classical CNN but uses a 3D kernel instead of a 2D kernel. The input for this model is the images with multiple layers (our double, four, or multiple layers image).

The hyperparameters are the same as for the CNN.

4.4.4 Support Vector Machine

The support vector machine uses as input the data output by the PCA, auto-encoder, or SVD. An observation is a vector (1-dimensional)

In this model, we have four hyper-parameters:

- Decision function shape, which is composed of OVO and OVR (see Section 2.2).
- The type of kernel (see Section 2.2).
- The gamma represents a value inversely proportional to the distance. A higher value means that the closest points are considered for the boundary decision. While a lower value means that we take into account way more points to construct the boundary decision.
- C that controls the penalty for training errors.

4.4.5 Recurrent Neural Network

As for the other neural network model, a RNN takes as input the data produced by the PCA, SVD, or auto-encoder. An observation is a two-dimensional matrix composed of time and a reduced number of points. The difference with the other neural network model is that, we do not have to flatten our matrix. We need to keep the time dimension as information since RNN are specialized in time sequence analysis.

In this model, there are four hyper-parameters to choose:

- The number of RNNs we make in a row.
- The numbers of nodes.

- The number of iterations.
- The batch size.

The LSTM and GRU use the same input as the classic RNN and have the same hyper-parameters.

4.4.6 K-Nearest Neighbor

This model uses the distance matrix that we have constructed as input. This is the only data the model needs to function. Only one hyper-parameter needs to be optimized here: the number of neighbors.

4.5 Manually Coded Solutions And Used Packages

In this section, we indicate models that have been implemented by ourselves and models for which have used an existing package.

The data set filtering to take only the first 20 most representative classes was hand-coded in Python. Removing the third dimension was hand-coded using the library "Numpy."Setting the number of frames to the same number was hand-coded in Python.

The building of all different datasets (the merging of the L, R, P and F datasets) was hand-coded using the library "Numpy."

For the dimension reduction techniques, we use the library "Scikit-learn" to perform the PCA, "Tensorflow" for the auto-encoder, and we hand-code the SVD using the "Numpy" library.

The image processing has been hand-coded, using again the library "Numpy" and distance matrix computation is also hand-coded, using this time the "Numba" package.

We use the library "Scikit-learn" for the decision tree, random forest, and support vector machine models, and the package "Tensorflow" for the ANN, CNN, CNN3D, RNN, GRU, and LSTM models. The KNN was hand-coded.

In the next chapter, we will present and analyze the results obtained from the different models detailed in chapter 2. We will compare the performance the performance of each of them, and try to explain why some work better than others.

Chapter 5

Results And Discussions

In this chapter of our master thesis, we present the results that we have obtained using all the models introduced in Chapter 2.

This chapter is divided into seven sections. The first section will try to respond to the question, "Which body part brings most of the information?". To do this, we have tested all possible combinations of the four datasets available to us. The second section will investigate which approach is the best to reduce the dimensions of our data. Then, we will test the performance of the several dimensionality reduction techniques used in this work. In the fourth, fifth, and sixth sections, we will present our results for the static, dynamic, and distance models. Finally, the conclusion of the chapter will be a summary of all our results.

It is important to know that each of the results that will be presented has been obtained with 5-fold cross-validation. We do this to strengthen our results and avoid overfitting.

5.1 Best Combination Of Data Sources

In this first section, we will investigate which data combinations are best. The goal is to see which part of the body brings the most information among the hands, face, and body. By knowing which information is the best, we can also reduce the amount of data we work with, which is essential in our case.

To this end, we have had first to run a PCA on our different datasets to reduce the quantity of data. Table 5.1 shows the PCA that we decide to implement for the different datasets. F stands for face, P for pose, L for left hand, and R for right hand. We decided to do a PCA10 for P, L, and R, meaning that we reduce L and R from 42 dimensions to 10 dimensions and P from 66 to 10. Concerning F, we performed a PCA20 (we keep 20 dimensions) because it contains much more information than the others, but with 20 variables, we kept a significant amount of information. These two decisions were based on the choice to keep 97% of the explained variance while reducing the data. For the dataset combinations, the PCA performed corresponds to the sum of the number of variables kept when there is no combination.

	F	P	L	R	FP	FL	FR	PL	PR	LR	FPL
PCA	20	10	10	10	30	30	30	20	20	20	40

Table 5.1: PCA for the Different Dataset

We then decided to use three different models to judge which combination was the best: a decision tree, a random forest, and an artificial neural network.

Concerning the architecture of the decision tree, we decided to max out at 400 trees. For the random forest, the number of trees is 270 with a maximum depth of 350. This model

takes a significant amount of processing time to be estimated on large datasets. In the ANN, we have four layers, each with 512 nodes; we perform a total of 100 iterations, and we have a batch size (the number of information that enter the neural network at the same time) of 32.

Table 5.2 shows all the results, each percentage represent an accuracy (percentage of time the model has the right answer). Several conclusions can be drawn :

- Most of the information is in R, which means that the majority of signs are executed with the right hand and that the left hand just adds a few details. Pose contains a little information, which is normal, as it contains points on both hands, on the face, and on the body, but there aren't enough dots on these masks. Signals are therefore,not precise enough. Finally, the face doesn't provide much information, given the number of points it contains. This can be explained by the fact that faces are used to add intonation and expression to sentences in sign language. But here, we're analyzing independent signs, not sentences, so intonation is not significantly important.
- Decision trees deliver deceptive results. Given the complexity of the input data, one explanation could be that the depth of our decision tree was not enough. But increasing the depth also increases the risk of overfitting.
- Random forest performs quite well, but we should add more trees and depth. The main issue is that calculation time will then be far too long.
- The ANN works well and is fast.

	Decision tree	Random forest	ANN
F	10.14%	23.01%	17.81%
P	21.15%	38.31%	36.11%
L	22.89%	37.80%	34.15%
R	48.55%	69.25%	69.39%
FP	18.34%	37.31%	40.05%
FL	21.62%	38.79%	37.97%
FR	44.29%	68.20%	71.21%
PL	22.39%	42.50%	43.88%
PR	36.40%	59.80%	66.18%
LR	51.28%	72.21%	76.76%
FPL	22.80%	42.58%	50.02%
FPR	36.22%	63.14%	72.54%
FLR	45.04%	69.43%	77.52%
PLR	41.09%	65.19%	73.95%
FLPR	38.02%	65.03%	76.85%

TABLE 5.2: Accuracy of the different combination

We decided to keep the dataset LR, which is composed of the left and right hands, for two reasons:

- LR has equally good results as FPLR but with much less information. It is therefore easier to manipulate the data and run algorithms. Another advantage of LR is that, with some algorithms such as random forest or decision tree, having more input makes the algorithm more complex, with more parameters to optimize and more possible combinations, which makes it much harder to maximize accuracy. LR, with its limited number of inputs, simplifies the task.

- We can see that F alone does not work well, which means that it doesn't provide much information. By including it in our data, we're much more likely to add noise than anything else.

There are several things to bear in mind with these results :

- The first one bears on the choice of the PCA preprocessing that we implement The choice is based on keeping a certain quantity of information, in our case 97% of the variance but this choice is arbitrary.
- It is impossible to optimize fully the different models because, using the hardware that we have at disposal, computing time is too long.
- ANN faces convergence issues. In other words, during the optimization process, the model will try to find the minimum of the loss-function but the gradient descent starts to oscillate as we see in Figure 5.1. This can lead to one ANN performing better than one another just because learning has been stopped during at a better point in time of this oscillatory process.

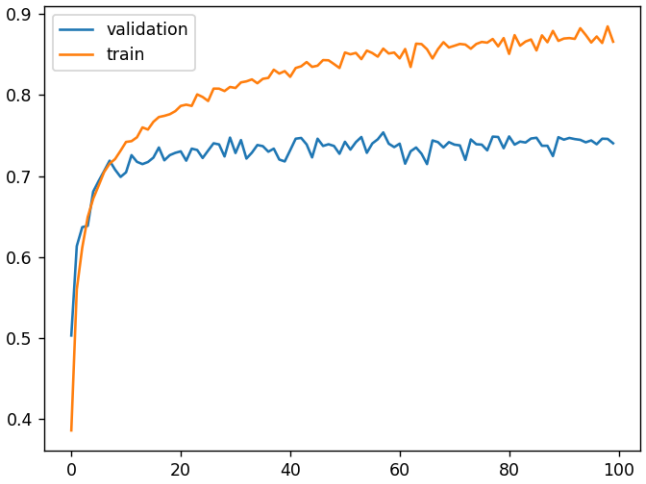

FIGURE 5.1: Gradient descent problem

5.2 The Best Approach To Reduce Dimensions

In Chapter four, we explained that dimension reduction can be performed in four different ways. In this section, we present the results obtained in each case. In order to identify the best approach, we use an ANN to test accuracy. We have had to reduce the amount of data due to computation time constraints and have therefore decided to implement several PCAs for each case as well. Table 5.3 shows the architecture of the ANN used for each PCA. The four first numbers in each row represent the numbers of nodes in the four layers of the neural network. Then, we have the dropout rate, which is a regularization technique used to prevent overfitting (a dropout rate of 0.2 means that we have a 20% chance that each neuron in a layer is ignored).

	Architecture
PCA 4	160/200/100/20—Dropout = 0.2
PCA 10	400/400/200/20—Dropout = 0.2
PCA 20	800/512/256/128/20—Dropout= 0.2
PCA 30	1200/800/256/128/20—Dropout = 0.2
PCA 40	1600/1024/512/128/20—Dropout = 0.2
PCA 50	2000/1024/256/128/20—Dropout = 0.2
PCA 60	2400/1024/512/256/20—Dropout = 0.2
PCA 70	2800/800/512/256/20—Dropout = 0.2
PCA 80	3200/1024/512/128/20—Dropout = 0.2
NO PCA	3360/1024/512/512/256/20—Dropout = 0.2

TABLE 5.3: Architecture of the ANN used to test each PCA

Table 5.4 shows the results for the four cases (see section 4.1 for explanation about the fours cases).

	Case 1	Case 2	Case 3	Case 4
PCA 4	42.85%	43.33%	43%	42.36%
PCA 10	68.33%	65.84%	67.80%	42.36%
PCA 20	79.23%	79.28%	78.61%	78.29%
PCA 30	80.09%	80.71%	80.09%	80.56%
PCA 40	80.68%	80.94%	80.84%	80.67%
PCA 50	81.32%	81.05%	81.33%	81.32%
PCA 60	81.74%	81.25%	81.10%	80.78%
PCA 70	81.05%	80.94%	81.28%	80.84%
PCA 80	81.31%	80.88%	81.07%	81.14%
NO PCA	74.12%	74.12%	74.12%	74.12%

TABLE 5.4: Results for the four cases

Several conclusions can be drawn from the results:

- There is no difference between the different cases. Case 1 was therefore arbitrarily chosen for the sequel of the analysis.
- We can see that the accuracy varies greatly with the different PCAs and is close to 80% from PCA with 20 dimensions. Additional dimensions bring a limited improvement of the results.

Note that the results reported here rely on ANN architectures not fully optimized. Also, since architectures of the ANN vary between the PCAs, results comparison can be driven by the change in ANN architecture and not the number of dimensions kept from the PCA.

5.3 Comparisons between PCA, SVD & Auto-encoder

In this third section, we study which of the three dimensions reduction techniques obtains the best results. In order to compare their results, we use an ANN to obtain accuracy.

It is important to remember that, from now on, we are working now only with case 1 and the dataset LR.

5.3.1 Principal Component Analysis

The first technique that we analyze is the PCA. The architecture of the ANN is the same as in the previous section, and the results in Table 5.5 are the same as in Table 5.4.

	PCA 4	PCA 10	PCA 20	PCA 30	PCA 40	PCA 50	PCA 60	PCA 70	PCA 80
Case 1	42.85%	68.33%	79.23%	80.09%	80.68%	81.32%	81.74%	81.05%	81.31%

TABLE 5.5: Results for the PCA

5.3.2 Singular Value Decomposition

The second technique used is the singular value decomposition (SVD). In this case, we use SDV to decompose the data and then partially recompose them with the selected value k.

Note that in this case, we first use the SVD, then we apply a PCA to the reconstructed data given by the SVD. Table 5.6 reports the architecture of ANN used to obtain our accuracy.

	Architecture
PCA 20	800/512/256/128/20—Dropout = 0.2
PCA 40	1600/1024/512/128/20—Dropout = 0.2
PCA 60	2400/1024/512/256/20—Dropout= 0.2
NO PCA	3360/1024/512/512/256/20—Dropout = 0.2

TABLE 5.6: Architecture of the ANN used to test each PCA

Table 5.7 shows the results for each k that we have tested.We can draw several conclusions from these results:

- Using the SDV does not improve accuracy. In our opinion, this is because SVD cannot filter efficiently noise. A second possible cause is that SVD drops too much information, which negates accuracy gains due to noise filtering. Finally, one can not exclude that some specific value of k between intervals we have tested increases accuracy.
- A surprising result with SVD is that with only four singular values (k = 4), we reach already almost constant levels of accuracy This can be seen clearly in Figure 5.2, which displays the results. We see two possible explanations for this.First, only four singular values are needed to obtain the vast majority of the information. The second, the PCA that performed after the SVD compensates and increases the accuracy. Figure 5.2 supports the second explanation since, without PCA, the accuracy plateau is reached at $k = 15$.
- The results for the three PCAs are more or less at the same level, with a slight dominance of PCA 2O, followed by PCA 40 and PCA 60.
- Another interesting result is that performance obtained after PCA is far superior to that obtained without PCA dimension reduction, as we can see in Figure 5.2. We think this is because with the PCA, the information is concentrated in certain features, making it easier for the ANN to place the weights in the neuron.

	PCA 20	PCA 40	PCA 60	No PCA
K1	72.04%	72.15%	71.61%	65.72%
K2	75.65%	75.57%	75.35%	70.76%
K3	77.43%	77.51%	77.05%	73.11%
K4	78.08%	77.70%	77.38%	73.51%
K5	77.96%	77.90%	77.28%	72.48%
K6	77.84%	77.43%	77.67%	72.86%
K7	78.04%	77.61%	77.19%	73.03%
K8	77.99%	78.11%	76.96%	72.71%
K9	77.88%	77.73%	77.03%	73.41%
K10	78.06%	77.69%	77.25%	73.24%
K15	77.71%	77.55%	77.17%	74.38%
K20	78.04%	77.89%	77.21%	74.50%
K25	78.07%	77.93%	76.97%	74.50%
K30	78.13%	77.66%	77.13%	74.59%
K35	77.61%	78.18%	77.72%	74.61%
K40	77.86%	77.97%	77.19%	74.15%

TABLE 5.7: Results for the different k

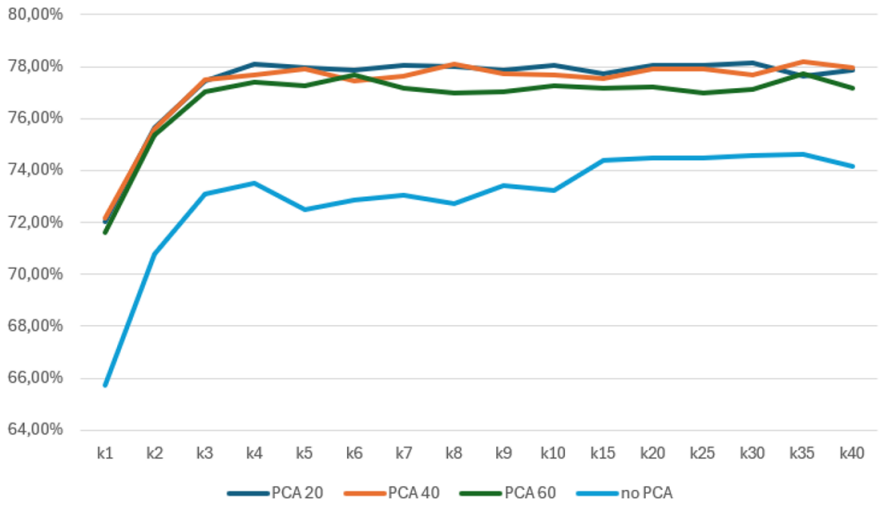

FIGURE 5.2: SDV results

Interpretation of these results are however subject to limitations:

- We use different architectures for the ANN and this potentially affect the comparability of the results
- The ANNs are not fully optimized.
- There may be variation in the results due to the gradient descent convergence issue.
- • A limited number of combinations of parameters have been explored due to time constraints.

5.3.3 Auto-encoder

The last dimension reduction technique that we use is the auto-encoder. Table 5.8 shows the architecture of the ANN used to test the performance of the auto-encoder.

	Architecture
PCA 4	160/200/100/20—Dropout = 0.2
PCA 10	400/400/200/20—Dropout = 0.2
PCA 20	800/512/ 256/128/20—Dropout= 0.2
PCA 30	1200/800/256/128/20—Dropout = 0.2
PCA 40	1600/1024/512/128/20—Dropout = 0.2
PCA 50	2000/1024/256/128/20—Dropout = 0.2
PCA 60	2400/1024/512/256/20—Dropout = 0.2
PCA 70	2800/800/512/256/20—Dropout = 0.2
PCA 80	3200/1024/512/128/20—Dropout = 0.2

TABLE 5.8: Architecture of the ANN to test the auto-encoder

Table 5.9 shows the results for different versions of the auto-encoders. Each column corresponds to a given architecture of the auto-encoder, where FL means the first layer, SL is the second layer, and TL is the third layer of the auto-encoder. Next to these abbreviations, we have the number of nodes in each of these layers. Each row corresponds to the number of dimensions that we keep; for example, Auto 4 means that we perform an auto-encoder and we keep four dimensions.

	FL :64/ SL :32	FL :64/ SL 64	FL :128/ SL :64	FL :64/ SL : 64/ TL :64
Auto 4	33.27%	33.71%	33.97%	34.18%
Auto 10	52.74%	52.80%	50.02%	49.33%
Auto 20	61.28%	64.75%	66.58%	62.35%
Auto 30	70.55%	69.66%	71.26%	66.24%
Auto 40	70.05%	71.23%	71.21%	67.24%
Auto 50	72.97%	72.70%	72.60%	68.73%
Auto 60	71.03%	71.82%	71.73%	67.63%
Auto 70	71.38%	72.75%	70.77%	67.85%
Auto 80	71.08%	71.89%	71.13%	68.05%
No Auto	74.12%	74.12%	74.12%	74.12%

TABLE 5.9: Results of the Auto-encoder

Looking at these results, we see no significant difference between the first three architectures, and the last one displays clearly the worse performance. To our opinion, the first architecture dominates but we acknowledge that this choice is rather arbitrary at this stage.

We can draw other conclusions from these results:

- We can see that if we reduce to thirty dimensions, the accuracy reaches a plateau, as we can see in Figure 5.3.
- A two-layer auto-encoder outperforms a three-layer auto-encoder for this type of data.
- We can observe a link between the number of neurons on the last layer and the number of dimensions reduced: The first architecture is better when reduced below thirty, while the second and third architectures are better when reduced to higher than 30 dimensions.

We have again to call for caution for the following reasons :

- The architecture of ANNs that we use to test the different auto-encoders varies again, making comparison of accuracy between auto encoder potentially misleading.
- ANN are not fully optimized, which can affect our results.
- Some results might be driven by gradient descent convergence issue.

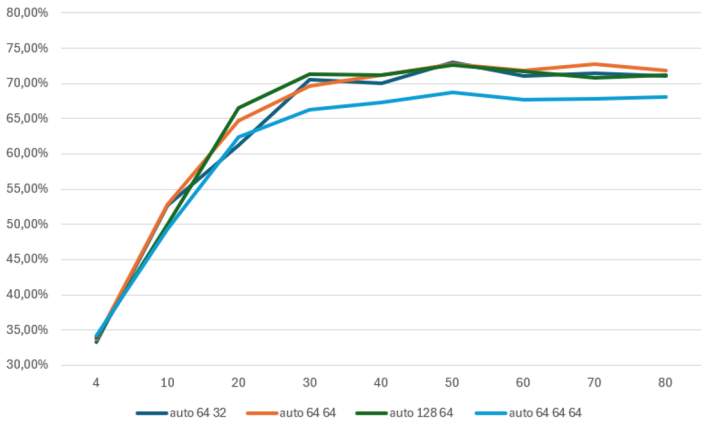

FIGURE 5.3: Results of Auto-encoder

5.3.4 Comparison of the Three Dimensionality Reduction Techniques

In this section, we compare the performance of the PCA, SVD, and auto-encoder. Table 5.10 shows the results of the best PCA, SVD, and auto-encoder that we have reported in previous sections. Each row corresponds to the number of dimensions that we keep, and each column to a method.

We can draw two conclusions from these results:

- The performance of PCA is far better than that of the auto-encoder. Table 5.10 highlights clearly this difference in performance.
- We also notice that the performance of the PCA is slightly better than the SVD, which means doing only a PCA is better.

	PCA (60)	SVD (k=40)	Autoencoder (64_64)
5	42.85%	76.76%	33.71%
10	68.33%	77.81%	52.80%
20	79.23%	77.86%	64.75%
30	80.09%	77.91%	69.66%
40	80.68%	77.97%	71.23%
50	81.32%	77.71%	72.70%
60	81.74%	77.19%	71.82%
70	81.05%	77.12%	72.75%
80	81.31%	71.89%	71.89%

TABLE 5.10: Comparison of best result of PCA, SVD and auto-encoder

5.4 Static Models

In this fourth section, we present results for the static models that we have introduced in Chapter 2: Models Theory. A static model does not evolve over time as a new data is used.

It is important to remember that these results were made on the dataset LR under case 1.

5.4.1 Artificial Neural Networks

Table 5.11 reports results of the ANN. Each row corresponds to a specific architecture based on the number of neurons in each layer. We test each structure on our reduced data using two dimension reduction techniques: auto-encoder, PCA, and on the original data.

We can draw conclusion from these results :

- The use the data preprocessed with PCA improves ANN performance. It is also better to reduce the number of dimensions since the results of the auto-encoder are better than those without reduction.
- Architectures with 2048 neurons on one layer have difficulty achieving better results than architectures with fewer neurons. Yet, theoretically, they should have equivalent or even better results because they have more neurons. This probably signals numerical convergence issues
- Interestingly, the third-layer architecture delivers the best results. Adding a fourth layer does not necessarily improve performance significantly.

Architecture	**AUTO**	**PCA**	**NO reduction**
1024—512	71.51%	79.17%	70.18%
2048—512	72.13%	77.98%	67.01%
1024—512—256	71.63%	81.14%	72.77%
1024—512—512	71.47%	80.49%	71.48%
1024—1024—512	70.32%	79.95%	72.22%
2048—512—256	72.65%	80.59%	70.86%
1024—256—256—128	71.76%	81.22%	70.22%
1024—512—256—128	72.35%	81.94%	73.36%
1024—512—512—256	72.23%	81.63%	73.00%
1024—512—512—512—256	71.54%	81.59%	73.65%

Table 5.11: Results of ANN

While we have tested several architectures, this still represents a limited choice. Alternative architectures might deliver better performance and the use of model selection techniques could help from this regard.

5.4.2 Convolutional Neural Network

In this section, we present the results of CNN. This model was applied on the one-layer images. Table 5.12 shows the architecture of the eight CNNs that we have tested.

Each column describes the structure of the CNN from start to end. The C2D is the 2D convolution, where the first number indicates the number of layers created and the number pair is the kernel size. M2D refers to the 2D pooling, where the number pair is the size of the pooling filter. Flatten means that we have flattened our data matrix into a vector. Finally, the isolated numbers are the number of neurons in the layer.

1	2	3	4	5	6	7	8
C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (64,(3,3))	C2D (64,(3,3))
M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)
C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (64,(3,3))	C2D (64,(3,3))
M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)
Flatten	Flatten	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (32,(3,3))	C2D (64,(3,3))	C2D (64,(3,3))
512	1024	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)	M2D(2,2)
	512	Flatten	Flatten	C2D (32,(3,3))	C2D (32,(3,3))	C2D (64,(3,3))	C2D (64,(3,3))
		512	1024	Flatten	Flatten	Flatten	Flatten
			512	512	1024	512	1024
					512		512

TABLE 5.12: Architecture of the CNN

Table 5.13 shows the results of the eight CNN structure.

	CNN 1	**CNN 2**	**CNN 3**	**CNN 4**	**CNN 5**	**CNN 6**	**CNN 7**	**CNN 8**
accuracy	19.97%	23.14%	20.79%	21.76%	23.14%	23.10%	23.21%	23.43%

TABLE 5.13: Accuracy of different architectures.

We can draw several conclusions looking to these results:

- First, we see that CNN does not perform very well. Indeed, we obtain an low accuracy levels, between 20% and 25%.
- Having one or two layers of neurons at the output of our model sometimes improves our accuracy (for example, between architectures 3 and 4). But it is not the case all the time, for example, between architecture 1 and 2, 5, 6, or 7, and 8.
- We can also see that increasing the number of convolution and pooling layers increases accuracy. However, we are limited in the number of layers because images are too small.
- Having convolutions with more layers increases accuracy.

Figure 5.4 gives us a much clearer view of our results.

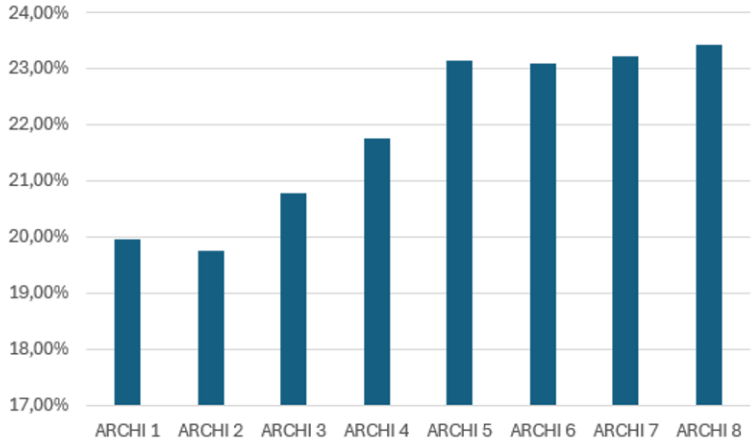

FIGURE 5.4: CNN Results

It is important to keep this in mind when looking at these results that we are limited by computing time constraint in the number of tested architecture and that our results are potentially affected by gradient descent convergence issues.

5.4.3 Convolutional Neural Network : 3D

In this section, we present the results of the CNN 3D that we use on the images that have at least two layers.

The hyper-parameters of the CNN3D are the same as those of the CNN in the previous section, except that we now have a convolution 3D and a pooling 3D.

1. Double-layers images
   Table 5.14 shows the architecture for the eights CNN3D that we have tested.

1	2	3	4	5	6	7	8
C3D (32,(2,3,3))	C3D (32,(1,3,3))	C3D (64,(2,3,3))	C3D (32,(1,3,3))	C3D (64,(2,3,3))	C3D (32,(1,3,3))	C3D (64,(2,3,3))	C3D (32,(1,3,3))
M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)	M3D(1,2,2)
R(-1,19,20,32)	C3D (32,(1,3,3))	R(-1, 19,20,32)	C3D (32,(1,3,3))	C3D (64,(1,3,3))	C3D (32,(1,3,3))	C3D (64,(1,3,3))	C3D (32,(1,3,3))
C2D(32(3,3))	M3D(2,2,2)	C2D(64,(3,3))	M3D(2,2,2)	M3D(1,2,2)	M3D(2,2,2)	M3D(1,2,2)	M3D(2,2,2)
M3D(1,2,2)	Flatten	M3D(1,2,2)	R(-1,8,9,32)	Flatten	R(-1,8,9,32)	C3D (64,(1,3,3))	R(-1,8,9,32)
Flatten	512	R(-1,8,9,32)	C2D(32,(3,3))	1024	C2D(32,(3,3))	C3D(64,(1,3,3))	C2D(32,(3,3))
512	512	C2D(32,(3,3))	Flatten	512	Flatten	Flatten	C2D(32,(3,3))
512		Flatten	512		1024	1024	Flatten
		512	512		512	512	1024
		512					512

TABLE 5.14: Architecture of the CNN3D for 2-layers images

Table 5.15 shows the results for the 8 CNN3D.

	CNN3D 1	CNN3D 2	CNN3D 3	CNN3D 4	CNN3D 5	CNN3D 6	CNN3D 7	CNN3D 8
accuracy	78.99%	78.39%	77.88%	77.92%	78.60%	77.98%	76.35%	75.29%

TABLE 5.15: Accuracy of different architectures.

A number of conclusions can be drawn from these results:

- A very positive observation is that the image transformation technique performs well. All the architectures have an accuracy higher than 75%, which is well above the previous CNN result.
- An interesting observation is that architectures with 2 or 3 convolution layers give better results than those with 4 convolution layers. It could be explained by the fact that the fourth layer is not able to provide additional information.

2. Four-layers images
   Table 5.16 shows the architecture of the eights CNN3D that we have tested.

1	2	3	4	5	6	7	8
C3D (32,(2,3,3))	C3D (32,(1,3,3))	C3D (32,(2,3,3))	C3D (32,(1,3,3))	C3D (64,(2,3,3))	C3D (64,(1,3,3))	C3D (64,(2,3,3))	C3D (64,(1,3,3))
M3D(1,2,2)	M3D(2,2,2)	M3D(1,2,2)	M3D(2,2,2)	M3D(1,2,2)	M3D(2,2,2)	M3D(1,2,2)	M3D(2,2,2)
C3D (32,(2,3,3))	C3D (32,(1,3,3))	C3D (32,(2,3,3))	C3D (32,(1,3,3))	C3D (64,(2,3,3))	C3D (64,(1,3,3))	C3D (64,(2,3,3))	C3D (64,(1,3,3))
Flatten	M3D(1,2,2)	C3D(32,(2,3,3))	M3D(1,2,2)	C3D(64,(2,3,3))	M3D(1,2,2)	C3D(64,(2,3,3))	M3D(1,2,2)
512	C3D(32,(1,3,3))	Flatten	Flatten	Flatten	Flatten	C3D (64,(1,3,1))	C3D (64,(1,3,1))
512	Flatten	1024	1024	1024	1024	Flatten	Flatten
	512					1024	1024
	512					512	512

TABLE 5.16: Architecture of the CNN3D for 4-layers images

Table 5.17 shows the results of the eights CNN3D.

	CNN3D 1	CNN3D 2	CNN3D 3	CNN3D 4	CNN3D 5	CNN3D 6	CNN3D 7	CNN3D 8
accuracy	78.97%	74.87%	77.72%	76.79%	77.81%	77.44%	76.09%	74.97%

TABLE 5.17: Accuracy of different architectures.

It is clear that these image transformation techniques perform just as well as double-layer images. All the results are above or really close to 75% accuracy. But this time, we did not find a trend in these results, it is not clear is a particular architecture is better than another.

3. Multiple layers images
   Table 5.18 shows the architecture of the 6 CNN3D that we have tested.

1	**2**	**3**	**4**	**5**	**6**
C3D (32,(3,3,3))	C3D (32,(3,3,3))	C3D (64,(3,3,3))	C3D (64,(3,3,3))	C3D (64,(3,3,3))	C3D (64,(3,3,3))
M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)
C3D (32,(3,3,3))	C3D (32,(3,3,3))	C3D (64,(3,3,3))	C3D (64,(3,3,3))	C3D (32,(3,3,3))	C3D (64,(3,3,3))
M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)	M3D(2,2,2)
C3D (32,(1,3,3))	C3D (32,(1,3,3))	C3D (64,(1,3,3))	C3D (64,(1,3,3))	C3D (32,(1,3,3))	C3D (64,(1,3,3))
C3D (32,(1,3,3))	M3D(1,2,2)	M3D(1,2,2)	C3D (64,(1,3,3))	M3D(1,2,2)	M3D(1,2,2)
Flatten	Flatten	Flatten	Flatten	Flatten	Flatten
1024	1024	512	1024	512	1024
512	512	512	512	512	512

TABLE 5.18: Architecture of the CNN3D for multiple layers images

Table 5.19 shows the results for the six CNN3D.

	CNN3D 1	**CNN3D 2**	**CNN3D 3**	**CNN3D 4**	**CNN3D 5**	**CNN3D 6**
accuracy	56.02%	57.83%	59.66%	57.31%	58.12%	58.88%

TABLE 5.19: Accuracy of different architectures.

It is obvious by looking at these results that this technique performs much worse than the previous two. Indeed, the maximum accuracy is 58.88%.

Summary of performance CNN2D & CNN3D The first evident conclusion is that CNN2D is performing very badly. It can be explained by the fact that the time dimensions have been completely removed. Even more, the fact that gestures start and end at different points creates a great deal of difference between images in the same class.

Concerning CNN3D, the first two methods work much better than CNN2D. The interesting thing about the fact that there is no difference between the performance of these two methods is that it had been hypothesized that separating the image into right and left hands in the four-layer image method would have resulted in better accuracy than the double-layer image method, but this is not the case. Finally, for the last method, the accuracy is more or less 55%, which is much worse than the first two methods in CNN3D. This could be explained by the fact that only 10 frames were kept. What's more, using images of 80 by 80 pixels still means a loss of precision. With more pixels, we could have maintained better image quality.

5.4.4 Support vector machine

The third static model used is SVM. There are numerous hyper-parameters in this model: *c*, *gamma*, decision function shape (OVO/OVR), and the type of kernel. We decide to use an RBF kernel, and we also test multiple *gamma* and *c*. Table 5.20 reports results according to specific hyper-parameter values. These results indicates that the hyper-parameter range and the decision shape function (OVO/OVR) have no material impact on accuracy. Only *c* matters, with a limited impact. It is obvious to say that the model does not work at all. The reason is unknown, but we have seen that the model is incapable of correctly separating the different classes. Given the limited time at our disposal and such deceptive results, we have decided not to explore this avenue any further.

	G = 0.5	G = 0.7	G = 0.8	G = 1	G = 1.2
Ovr, c = 0.8	12.43%	12.43%	12.43%	12.43%	12.43%
Ovr, c = 1	12.47%	12.47%	12.47%	12.47%	12.47%
Ovr, c = 1.2	12.47%	12.47%	12.47%	12.47%	12.47%
Ovo, c = 0.8	12.43%	12.43%	12.43%	12.43%	12.43%
Ovo, c = 1	12.47%	12.47%	12.47%	12.47%	12.47%
Ovo, c = 1.2	12.47%	12.47%	12.47%	12.47%	12.47%

TABLE 5.20: SVM results

5.5 Dynamic models

In this section, we present results for dynamic models, which are models that can take account of changes over time.

5.5.1 Recurrent Neural Network

Table 5.21 shows the results for a classic recurrent neural network (RNN). Each row corresponds to a specific architecture based on the number of neurons in each layer. We decided to test RNN for each of reduction techniques except the SVD. We exclude SVD because since we did a PCA after the SVD and the results were not as good, there's no point in doing it. We also test it on data without dimension reduction.

There are several conclusions that emerge from these results:

- With RNN, the best results are obtained after PCA dimension reduction. Autoencoder doesn't a performance improvement compared to the use of raw data.
- The architecture that contains the highest number of neurons in its layer performs best.
- Architectures that contain at least two layers work better if they contain enough neurons in each of their layers

Architecture	**Autoencoder 60**	**PCA 60**	**No reduction**
64	36.32%	46.27%	36.15%
128	35.76%	52.07%	37.04%
256	35.08%	56.72%	36.26%
512	29.05%	56.50%	29.74%
64-—4	51.51%	59.08%	52.22%
128—64	45.73%	65.59%	48.31%
256—64	39.95%	66.11%	55.41%
128—128	44.22%	65.20%	47.02%
256—256	42.43%	64.55%	53.56%
64—64—64	54.15%	64.20%	38.58%

TABLE 5.21: Results of the RNN

5.5.2 Gated Recurrent Unit

Table 5.22 shows the results for the different gated recurrent unit (GRU) architectures. The table has the same structure as the RNN results table (see Table 5.21). Results in

Architecture	Autoencoder 60	PCA 60	No reduction
64	72.90%	79.17%	76.68%
128	73.13%	80.17%	77.60%
256	72.96%	80.83%	77.21%
512	73.53%	81.16%	75.83%
64—64	73.55%	79.83%	78.44%
128—64	73.60%	80.09%	77.53%
256—64	73.65%	80.74%	77.02%
128—128	73.62%	80.84%	77.46%
256—256	73.71%	80.59%	77.21%
64—64—64	73.14%	80.23%	77.53%

TABLE 5.22: Results of GRU

Table 5.22 deliver two conclusions :

- With a GRU, PCA dimension reduction delivers the best results and the use of the auto-encoder, the worse ones.
- We also observe that the architectures with the highest number of neurons in their layers perform best.

5.5.3 Long Short Term Memory

Table 5.23 shows the results for the different LSTM architectures. The table has the same structure as the GRU and RNN results tables (see Table 5.22 and Table 5.21). We can draw several conclusions from these results:

- With an LSTM, PCA dimension reduction delivers again the best results and the auto-encoder based one the worse.
- Architectures with the highest number of neurons in their layers perform best.
- Architectures containing more layers perform better, except for the architecture with 3 layers. This probably due to a too limited number of neurons in each layer

Architecture	Autoencoder 60	PCA 60	No reduction
64	72.43%	78.71%	74.14%
128	72.71%	79.52%	74.26%
256	74.06%	79.86%	73.77%
512	74.16%	80.34%	71.94%
64—64	73.51%	79.57%	77.24%
128—64	73.61%	79.72%	77.22%
256—64	73.95%	80.34%	75.85%
128—128	72.79%	80.61%	77.28%
256—256	75.66%	81.03%	75.64%
64—64—64	74.26%	80.31%	77.47%

TABLE 5.23: Results of LSTM

5.6 K-Nearest Neighbor

In this section, we present result of a simple K-Nearest Neighbor algoritm (KNN), that uses the Euclidean distance matrix as input. The results will be reported witout and with data standardization.

5.6.1 KNN without Standardization

The only hyper-parameter that we had to optimize with KNN is the number of neighbors, which is provided in each row of Table 5.24. For example, K15 means that to classify new data, we take into account the 15 closest neighbors.

As hands do not start at the same place for each observation for a specific sign due to the preceding and following signs, to implement KNN, we remove a given number of frames at the beginning and end of the sequence. We have chosen to remove the first and last 5/10/15 frames to test if these frames bring noise or information.

The results are shown in Table 5.24 Each column corresponds to the number of frames we use. We remove 10/20/30 frames in the first, second, and third columns.

TABLE 5.24: Results for non standardized KNNs

	Number of Frames			
number of neighbors	40 frames	30 frames	20 frames	10 frames
K1	41.95%	44.31%	45.48%	45.13%
K5	45.53%	47.14%	48.08%	47.74%
K10	45.71%	46.51%	48.69%	48.09%
K15	45.69%	47.44%	48.20%	47.71%
K20	44.96%	46.65%	47.45%	46.86%
K30	44.00%	45.58%	45.96%	45.42%
K40	43.11%	44.44%	44.90%	44.37%
K50	42.09%	43.74%	43.85%	43.49%
K60	41.23%	42.84%	42.99%	42.55%

By looking at these results, we can draw several conclusions:

- Deleting frames increases accuracy, as we can clearly see in Figure 5.24. This is due to the fact that, in the first and last frames, the signs do not start at the same place, or are even located at completely opposite positions. As a result, the Euclidean distance between these first points is very high, significantly increasing the total cost of the distance between two observations.
- The best number of neighbors seems to be 10.
- It is preferable to remove 10 frames at the beginning and end of each sequence to increase accuracy. Removing 5 frames at the beginning and end of each sequence doesn't remove enough noise, and removing 15 frames at the beginning and end of each sequence removes too much information.

It is important to notice that we did not explore all possible neighbors because it would take too much time. We also didn't explore all possible frame removals.

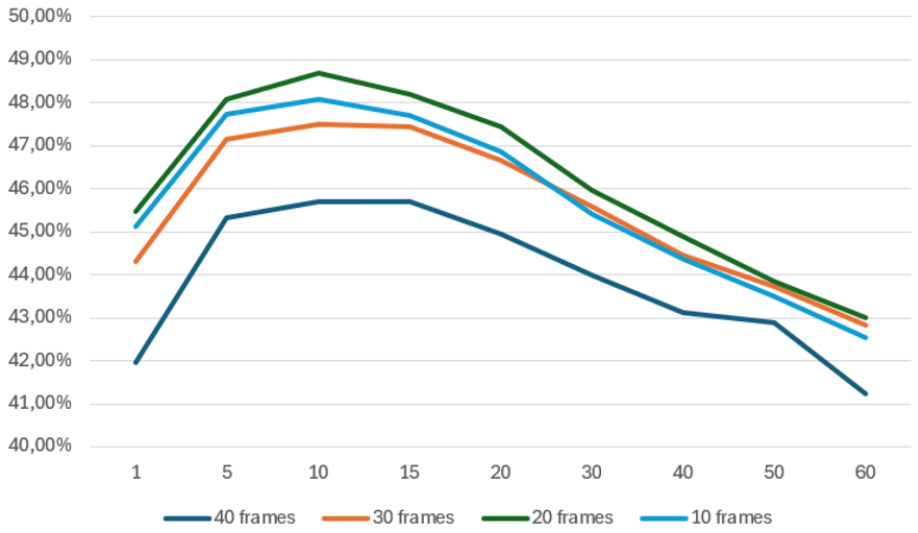

FIGURE 5.5: Results of non standardized KNN

5.6.2 KNN Standardized

For the standardized KNN, the results are presented in the same form as for the non-standardized KNN. Table 5.25 shows the results for the different numbers of neighbors and frames we use.

TABLE 5.25: Results for standardized KNN

	Number of Frames			
number of neighbors	40 frames	30 frames	20 frames	10 frames
K1	42.68%	44.65%	45.37%	44.91%
K5	45.37%	47.33%	47.89%	47.21%
K10	46.17%	47.72%	48.27%	47.40%
K15	45.49%	47.01%	47.68%	47.01%
K20	44.91%	46.68%	47.02%	46.11%
K30	44.05%	45.21%	45.71%	45.11%
K40	42.94%	44.27%	44.64%	44.11%
K50	42.18%	43.65%	43.85%	43.18%
K60	41.48%	42.84%	42.98%	42.52%

The analysis of these results is the same as in non-standardized KNN (see Section 7.6.1). The only difference is that there is a slight bias with the standardization because we perform it before we divide the train and test sets.

5.7 Summary And Discussion Of The Results

In this section, we will present a brief summary of all our results. Table 5.26 shows the best accuracy for each of the models that we have tested.

Model	Accuracy
SVM with PCA	12.47%
KNN	48.69%
ANN	81.94%
RNN with PCA	66.11%
GRU with PCA	81.16%
LSTM with PCA	81.03%
CNN2D	23.43%
CNN3D double-layers images	78.99%
CNN3D four-layers images	78.97%
CNN3D multiple layers images	59.66%

TABLE 5.26: Accuracy of different models.

All the results present in Table 5.26 have been carried out on the datasets LR (combination of right and left hand) under Case 1, which is when we perform dimension reduction on x,y together, left and right hand together. This combination of dataset and case is the best one.

We notice that among the static models, which are ANN and SVM, Only one performs well, which is the ANN. The SVM is incapable of separating correctly the classes. For the KNN, standardize or not the data does not change significantly the results. However, an encouraging result is the fact that under our hypothesis to delete the 10 first and last frames, KNN accuracy increases by an average of 2.5%.

Concerning the dynamic models, both LSTM and GRU perform well and are quite similar, but we cannot say the same for the RNN, which has more difficulty recognizing the signs.

Finally, as we know, the only transformations in the image that work are the CNN3D with double and four-layer images.

What is surprising is that we thought that dynamic models would have better accuracy than static models because they have the particularity of being able to handle the temporal aspect. But we can clearly see that this is not the case either and that they perform equally well. This could be explained by the fact that the sequences of our observations are not long enough.

Another interesting analysis is that the only dimensionality reduction technique that is present in 5.26 is the PCA. This technique is clearly the best one compared to the SVD and auto-encoder. It means that our hypothesis about the SVD could improve the performance of the model by reducing the noise in our data turns out to be wrong.

Conclusion and Future Work

This master thesis investigates the performance of machine learning (ML) models for sign language (SL) recognition with a particular focus on dimensionality reduction techniques (DRT).

A first objective was to improve the performance of these ML models by addressing the high-dimensional nature of SL data thanks to DRT. The large number of dimensions poses indeed numerous challenges, such as the curse of dimensionality, computational complexity and model overfitting. To this end, we have carried out numerous tests and analyses with several prominent dimensionality reduction techniques, including principal component analysis, singular value decomposition, and auto-encoder. For each of these techniques, various architectures and set of parameters have been tested to conduct in-depth investigation performance improvement potential.

A second objectif was to compare the performance of ML models, such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Artificial Neural Networks (ANN), and several deep learning frameworks, including Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), after DRT preprocessing

Our results indicates that performing any DRT on any ML model always improves sign language recognition precision on the investigate dataset. By reducing noise and redundancy in the data, ML models can learn more efficiently the features that are necessary to identify signs. In addition, our results indicate that PCA is the best performing DRT, followed by auto-encoder, and finally SVD.

Among ML models, deep learning models dominates clearly other alternatives. One of the most surprising findings is the poor performance of RNN. Despite the capacity of RNNs to take into account the time dimension when analyzing sequential data, RNN (at least as we did implement them) did not outperform simple ANN. One probable explanation is that the length of our sequences that we analyze was too short for RNN to benefit from taking into account the time dimension. Further investigation is needed to fully understand this issue.

We also investigate the potential contribution of image transformation methods and show that some display very encouraging results. In particular, CNN3D used with the double- and four-layers images contributes significantly to performance improvements. These results highlight the capacity of the convolutional structure to capture the right feature to recognize sign language. Noteworthy, the ability of CNN3D to incorporate the temporal dimension explains differences of performance with respect to CNN2D.

Several avenues are opened at this stage for further investigations. First, we faced limits of storage capacity for image transformations, especially for the double, four and multiple layers images. These are indeed extremely heavy, and we had to limit ourselves to 10 frames. This limit means that we lose a significant fraction of information contained in image sequences. With more time and more powerful hardware, we believe that we would have been in position to report better results.

Secondly, we assumed that the first and last frames bring noise since the start and the end for a same sign is always varying in our dataset. We have tested this assumption only the KNN model due to time limitation. It would be interesting to test it for every model to see if the performance can be improved. In particular, it would be very interesting to see whether neural networks underweight the first and last frames.

Thirdly, due to limited computing power, we have had to restrain ourselves to quite simple neural network architecture. With more computing power, we would be in position to test more complex architectures (adding more layers and neurons).

Finally, this work focuses on three specific dimensionality reduction technique : PCA, SVD and auto-encoder. It would been interesting to explore more techniques in depth to see whether they are more adapted to our SL recognition tasks.

In conclusion, this master thesis contributes to the field of sign language recognition by (i) investigating how dimensionality reduction techniques have an impact on machine learning models and (ii) comparing the performance of several machine learning approaches. We hope that this will be helpful to the development of efficient and accurate sign language recognition systems.

Bibliography

[1] Rastgoo, Kiani, and Escalera. Word separation in continuous sign language using isolated signs and post-processing. *Cornell University*, 2022.

[2] World Health Organization. Deafness and hearing loss, 2021. URL http://www.who.int/mediacentre/factsheets/fs300/en/.

[3] Zheng, Liang, and Jiang. Recent advances of deep learning for sign language recognition. *International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–17, 2017.

[4] Darren and Pentland. Comput vis pattern recognit. proceedings cvpr'93. *IEEE computer society conference*, pages 335–340, 1993.

[5] Huand and Huang. Sign language recognition using model-based 563 tracking and a 3d hopfield neural network. *Machine Vi- 564 sion and Applications*, 10(5-6):292–307, 1998.

[6] Cheok, Zaid, and Hisham Jaward. A review of hand gesture and sign language recognition techniques. *International Journal of Machine Learning and Cybernetics*, 2017.

[7] Fink, Frénay, Meurant, and Cleve. Lsfb-cont and lsfb-isol: Two new datasets for vision-based sign language recognition. *In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN 2021) IEEE Computer Society Press*, 2021.

[8] Agrawal, Jalal, and Tripathi. A survey on manual and non-manual sign language recognition for isolated and continuous sign. *International Journal of Applied Pattern Recognition*, 3:99, 2016.

[9] Rastgoo, Kiani, and Escalera. Sign language recognition: A deep survey. *Expert Systems with Applications*, 2020.

[10] Moeslund, Hilton, Krüger, and Sigal. *Visual Analysis of Humans*. Springer eBooks, 2011.

[11] Joksimoski, Zdravevski, Lameski, Pires, Melero, Martinez, Garcia, Mihajlow, and d trajkovik Chorbev a. Technological solutions for sign language recognition : A scoping review of research trends, challenges, and opportunities. *IEEE Access*, 2022.

[12] Oya Aran. *Vision based sign language recognition: modelling and recognizing isolated signs with manual and non-manual components*. PhD thesis, Bogazici University, 2008.

[13] Lead Academy. What is finger spelling. URL https://lead-academy.org/blog/what-is-fingerspelling/. Accessed on: 2024-03-21.

[14] Kimmelman, Sabyrov, Mukushev, Imashev, Koishybay, and Sandygulova. Towards real-time sign language interpreting robot: Evaluation of non-manual components on recognition accuracy. *Computer Vision and Pattern recognition (CVPR) Workshops*, 2019.

[15] Ahmed, Zaidan, Zaidan, Salih, and Lakulu. A review on systems-based sensory gloves for sign language recognition state of the art between 2007 and 2017. *Sensors*, 18:2208, 2018.

[16] Joksimoski, Zdravevski, Lameski, Pires, Melero, Martinez, García, Mihajlov, Chorbev, and Trajkovik. Technological solutions for sign language recognition : A scoping review of research trends, challenges, and opportunities. *IEEE Access*, 2022.

[17] Al-Shamayleh, Ahmad, Abushariah, Alam, and Jomhari. A systematic literature review on vision based gesture recognition techniques. *Multimedia Tools And Applications,*, 77(21):28121 28184, 2018.

[18] Bragg, Verhoef adnd Vogler, Ringel Morris abdKoller, Bellard, and Kacorri. Sign language recognition, generation, and translation. *The 21st International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS*, 2019.

[19] Ibrahim, Zayed, and Selim. Advances, challenges and opportunities in continuous sign language recognition. *Journal Of Engineering And Applied Sciences*, 15(5):1205–1227, 2019.

[20] Ruiduo Yang and Sarkar. Detecting coarticulation in sign language using conditional random fields. *IEEE 18th International Conference on Pattern Recognition (ICPR'06) - Hong Kong, China ()] 18th International Conference on Pattern Recognition (ICPR'06)*, pages 108–112, 2006.

[21] Hodge and Johnston. Points, depictions, gestures and enactment : Partly lexical and non-lexical signs as core elements of single clause-like units in auslan (australian sign language). *Australian Journal Of Linguistics*, 34:262-291, 2014.

[22] Christophr M.Bishop. *Pattern Recognition and Machine Learning*. SPRINGER, 2006.

[23] Hassan Hassan and al. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. *International Water Technology Journal*, 2015.

[24] Vikashraj Luhaniwal. Analyzing different types of activation functions in neural networks— which one to prefer. *Towards data science*, 2019.

[25] Gaurav Nair. The spark your neural network needs: Understanding the significance of activation functions, 2023. URL https://medium.com/@gauravnair/the-spark-your-neural-network-needs-understanding-the-significance-of-activation-functions-6b82d5f8c65.

[26] De Plaen Pierre-François. Study of convolutional models for semi- supervised classification on graph-structured data. Master's thesis, University Catholique of Louvain-La-Neuve, 2021.

[27] IBM. Convolutional neural networks, n.d. URL https://www.ibm.com/topics/convolutional-neural-networks. Accessed on: 2024-01-4.

[28] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. *Towards data science*, 2018.

[29] Saha Sumit. A comprehensive guide to convolutional neural networks - the eli5 way, 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[30] Baeldung. How relu and dropout layers work in cnns, 2022. URL https://www.baeldung.com/cs/ml-relu-dropout-layers. Accessed on: 2024-01-4.

[31] Harsh Yadav. Dropout in neural networks. *Towards data science*, 2022.

[32] Sankar, Barfett, Colak, and Valaee. Recent advances in recurrent neural networks. *https://arxiv.org/abs/1801.01078*, 2018.

[33] N. K Manaswi. *Deep Learning with Applications Using Python*. Apress, 2018.

[34] Pascanu1, Gulcehre1, Cho2, and Bengio1. How to construct deep recurrent neural networks. *https://arxiv.org/abs/1801.01078*, 2014.

[35] Hastie, Tibshirani, and Friedman. *The Element of Statistical Learning*. SPRINGER, 2009.

[36] geeksforgeeks. K-nearest neighbor(knn) algorithm, 2024. URL https://www.geeksforgeeks.org/k-nearest-neighbours/.

[37] Bharadiva. A tutorial on principal component analysis for dimensionality reduction in machine learning. *Zenodo (CERN European Organization For Nuclear Research*, 2023.

[38] Bruni, Cardinalu, and Vitulano. A short review on minimum description length : An application to dimension reduction in pca. *ENtropy*, 24(2):269, 2022.

[39] Li, Zhand, Zhao, and Yi. Guided autoencoder for dimensionality reduction of pedestrian features. *Applied Intelligence*, 50(12):4557-4567, 2020.

[40] Berry and Browne. *Understanding search engines : mathematical modeling and text retrieval*. SIAM, 2000.

[41] Meurant. Corpus lsfb. first digital open access corpus of movies and annotations of french belgian sign language (lsfb). URL http://www.corpus-lsfb.be. Accessed on: 2024-04-01.

Appendix

Appendix 1: Fingerspelling

FIGURE 6: American sign language [13]

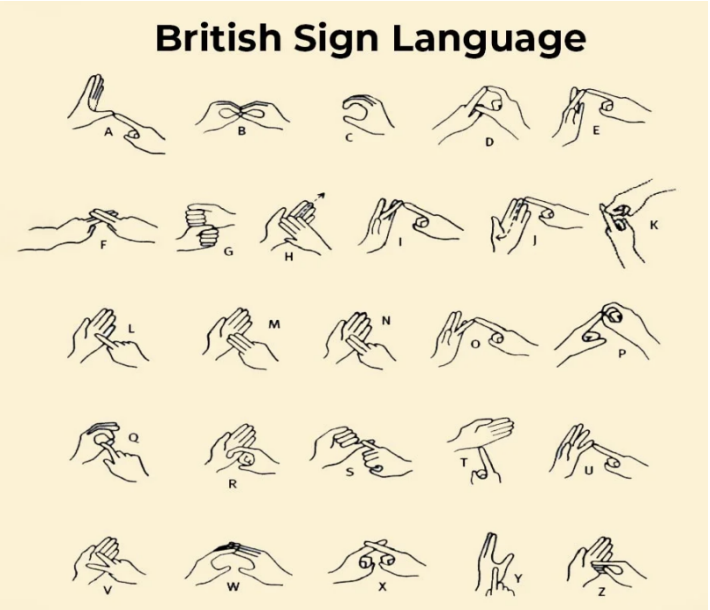

FIGURE 7: British sign language [13]

Appendix 2: Description Of The Existing Datasets

- The Phoenix Weather Dataset

  The phoenix weather dataset is one of the first dataset for continuous SLR. It was built with videos of 9 different signers that translate German weather forecasts. It contains 1080 classes (one class equals one sign) which makes of it one of the largest dataset available for continuous SLR. The main drawback of this dataset is that the signers tend to sign differently when they translate something or when they speak naturally as in everyday life. The consequence is that this dataset is not the best one to recognize natural sign language.

- The Greek Sign Language (GSL) Dataset

  The Greek Sign Language is a dataset that contains five different scripted scenarios that are signed by seven signers. The scenarios are constructed based on interaction with public service employees such as police officers, train station agents, etc. The dataset revealed to be extremely useful when we have to implement a system that helps impaired people to communicate with public services but it suffers from two drawbacks. The first one is that since only five situations are used, there is a lack of diversity in terms of co-articulation. The second one is that since it is based on the use of scripts, signers tend to slow down their movements, which distorts the data a little. Videos are recorded in 30 frames per second, and there is also an isolated dataset version.

- The American Sign Language Lexicon Video Dataset (ASLLVD)

  The American Sign Language Lexicon Video Dataset was created in 2008 and is one of the first large dataset for isolated sign language. Six signers were used to build it and each of them performed each sign one time from a neutral position. The main quality of this dataset is that four cameras are used to capture signs which brings much more information about the sign's spatial position. The main weakness is that there are very few repetitions of each sign, which is problematic in deep learning since it requires numerous examples to be trained.

- The DEVISIGN-L Dataset

  The DEVISIGN-L dataset is a dataset of the Chinese sign language that contains two thousand classes with twelve examples for each class signed by eight different signers. It was created in 2016 and uses a Kinect device to provide depth information for each video. The weaknesses are the same as the American Sign Language Lexicon Video Dataset.

- Microsoft American Sign Language (MS-ASL) dataset

  The Microsoft American Sign Language is the dataset with the greatest variety in terms of backgrounds, signers, and lighting which makes it one of the best dataset to implement a SLR system performing well in any situation. It was created in 2019 with more than two hundred different signers, who have signed one thousand signs. This dataset was created by combining videos of the American sign language available on YouTube. Another quality of this dataset is that the authors have created four sub- e, each with a different number of classes: two hundred, four hundred, five hundred and one thousand. However, only URL and times of the videos are provided, and as time length goes by, the videos disappear from YouTube, diminishing the quality of the dataset.

These two figures resume the information about all these datasets.

Continuous Signs Datasets	**Class**	**Signers**	**FPS**	**Videos**	**Position of the Camera**
Phoenix weather	1080	9	30	6841	1 controlled camera
GSL	310	7	30	40826	little variation

FIGURE 8: Continuous sign dataset specification

Isolated Signs Datasets	**Class**	**Signers**	**FPS**	**Videos**	**Position of the Camera**
ASLLVD	3300	6	30	9800	4 controlled cameras
DEVISIGN-L Dataset	2000	8	30	2400	1 controlled camera
MS-ASL dataset	1000	222	vary	25513	great variation

FIGURE 9: Isolated sign dataset specification

Appendix 3 : One Versus One and One Versus Rest

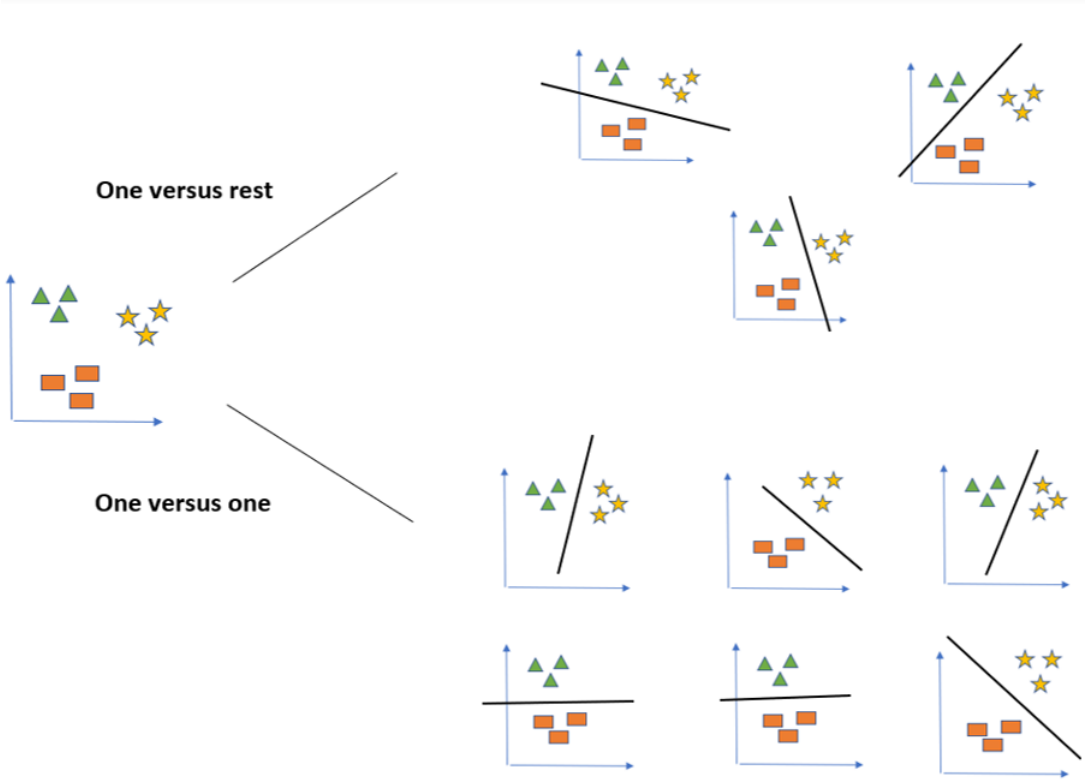

FIGURE 10: OVR and OVO