N° d'ordre :
N° de série :

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA Ministry of Higher Education and Scientific Research

UNIVERSITY ECHAHID HAMMA LAKHDAR
EL OUED FACULTY OF EXACT SCIENCES
Computer Science department
End of study project
Presented for the diploma of

ACADEMIC MASTER

Domain : **Mathematics and Computer Science**
Industry : **Computer Science**
Speciality : **Distributed Systems and Artificial Intelligence**

Presented by :

- **Laklouka Ben Salem**
- **Cherfi Hayder Seif Eddine**

Theme

Video Transcribing for Sign Language

Sustained on ..-..- 2021 For the jury:

Miss.	Guettas Chourouk	MAA	Supervisor Univ. El Oued
M.	……	MCA	President Univ. El Oued
M.	……	MAA	Reporter Univ. El Oued

Academic year: 2020/2021

Abstract

Communication is an interaction between two or more peoples through which knowledge is exchanged, sending and receiving information between the communicating peoples, the importance of communication lies in societies due to its great importance in the formation of understanding and harmonious societies, but for the deaf-mute community, communication is difficult or almost impossible. With the rest of the people, except with people of their own category, because they use sign language, which is impossible for those who do not master it to understand them. We aim in this project and by using artificial intelligence and machine learning techniques and exploiting object detection techniques, to develop a program that can translate sign language into written words to facilitate communication between the deaf-mute community and the rest of the people without the need to learn sign language.

Key words: machine learning, object detection, sign language.

الملخص

التواصل هو تفاعل يكون بين طرفين او اكثر يتم عن طريقه تبادل معارف, ارسال و استقبال معلومات بين الاطراف المتواصلة, و تكمن اهمية التواصل في المجتمعات نظرا لأهميته البالغة في تكوين مجتعمات متفاهمة و متناغمة, و لكن بالنسبة لفئة الصم البكم فالتواصل يكون صعبا او شبه مستحيل مع بقية الناس الا مع اناس من فئتهم و ذلك لأنهم يستخدمون لغة الاشارة و التي يستحيل على من لا يتقنها فهم الطرف الاخر,

نهدف في هذا المشروع و باستخدام الذكاء الاصطناعي و تقنيات تعلم الالة و استغلال تقنيات الكشف و متابعة الاشياء, ان نطور برنامج بأمكانه ترجمة لغة الاشارة الى كلمات مكتوبة ليسهل بذلك التواصل بين فئة الصم البكم و بقية الناس بدون داعي لتعلم لغة الاشارة.

الكلمات المفتاحية تعلم الالة استغلال تقنيات الكشف و متابعة الاشياء لغة الاشارة

Abstract

La communication est une interaction entre deux ou plusieurs peuples à travers laquelle des connaissances sont échangées, envoyant et recevant des informations entre les peuples communicants, l'importance de la communication réside dans les sociétés en raison de sa grande importance dans la formation de sociétés compréhensives et harmonieuses, mais pour les sourds- communauté muette, la communication est difficile ou presque impossible. Avec le reste du peuple, sauf avec les gens de leur catégorie, car ils utilisent la langue des signes, ce qui est impossible pour ceux qui ne la maîtrisent pas de les comprendre. Nous visons dans ce projet et en utilisant l'intelligence artificielle et les techniques d'apprentissage automatique et en exploitant les techniques de détection d'objets, à développer un programme qui peut traduire la langue des signes en mots écrits pour faciliter la communication entre la communauté des sourds-muets et le reste de la population sans avoir besoin pour apprendre la langue des signes.

Les mots clés: apprentissage automatique, détection d'objets, langue des signes

Contents

List of Figures

General introduction

During the last celebration of the International Day of the Deaf and Mute, corresponding to September 23, 2020, the World Organization of the Deaf announced, according to its latest statistics, that there are approximately 72 million deaf people around the world, more than 80 percent of whom live in developing countries, and they use more than 300 different sign languages.

This large number of different sign languages around the world makes the way of communication between different people who suffer from deafness difficult, and therefore the existence of a unified language is very important for this group, which is considered marginalized, especially in developing countries, for the purpose of facilitating communication and exchanging ideas and experiences with their counterparts In developed countries and those who receive special care in all areas of life

In our project, we considered it a moral and humanitarian duty to help this category by creating a program that translates the sign language into readable text through video analysis with the help of artificial intelligence and machine learning technology.

We chose English to be the language used in our project due to its wide spread around the world. We also preferred to choose word-based sign language translation instead of letter-based sign language translation in order to facilitate and speed up the translation process for users.

This report will contain four chapters

Chapter 1: will explain in details the sign language, and our choice the WLASL (Word-Level American Sign Language) and our motivation.

Chapter 2: will contain the evaluation of other similar works

Chapter 3: Explain the architecture of the system and the tools used to implement this project

Chapter 4: The last chapter contains the description of the implementation of the system and discussing the obtained results.

Chapter I

Introduction to the American Sign Language

I.1 Introduction: what is ASL?

American Sign Language (ASL) is a visual/gestural language. It is a natural language, meaning that it has developed naturally over time by its users, Deaf people. ASL has all of the features of any language; that is, it is a rule governed system using symbols to represent meaning. In ASL, the symbols are specific hand movements and configurations that are modified by facial expressions to convey meaning. These gestures or symbols are called signs [1].

Contrary to common belief, ASL is not derived from any spoken language, nor is it a visual code representing English. It is a unique and distinct language, one that does not depend on speech or sound. ASL has its own grammar, sentence construction, idiomatic usage, slang, style, and regional variations-the characteristics that define any language.[1]

American Sign Language is the shared language that unites Deaf people in what is known as the Deaf community. Deaf with a capital D is used in publications to recognize the cultural and linguistic affiliation of Deaf people who are members of the Deaf community, whereas deaf with a lowercase d is used to refer to deaf people who do not embrace ASL or involve themselves with the values, organizations, and events that are heralded by signing Deaf people.[1]

The Deaf community is not bound by geographic borders, but rather comprises those people who elect to become members by using ASL as their preferred mode of communication and by accepting the cultural identity of Deaf people. It is difficult to give an accurate number of how many people are in the Deaf community because census takers typically lump together all people who have a hearing loss. Many researchers believe that approximately 10 percent of the general population has some degree of hearing loss and that 1 percent of that number represents Deaf people, for a total of about half a million people in the Deaf community. The people most likely to be native users of ASL are those who have Deaf parents. People who lose their hearing as infants, before they begin to speak, may become native signers if they are exposed to ASL at an early age. These people, who are unable to hear English and learn it naturally, must be taught English through formal means. Hearing children of Deaf parents also acquire ASL as a first language. However, their enculturation tends to cross the cultures of the Deaf and hearing worlds. These children, like their Deaf counterparts, are often referred to as bicultural and bilingual [1].

I.1.1 Where Did American Sign Language Originate?

The precise beginnings of ASL aren't clean. Many human beings agree with that ASL came mostly from French sign Language (FSL). Others claim that the muse for ASL existed earlier than FSL became added in the us in 1817. It changed into in that year that a French instructor named Laurent Clerc, added to the usa by way of Thomas Gallaudet, based the primary college for the deaf in Hartford, Connecticut. Clerc started teaching FSL to americans, though many of his students have been already fluent in their personal types of nearby, natural sign language. contemporary ASL probably contains some of this early American signing. Which language had extra to do with the formation of current ASL is tough to show. present day ASL and FSL percentage a few factors, together with a significant amount of vocabulary. but, they're now not jointly understandable. [1]

I.1.2 How Does American Sign Language Compare With Spoken Language?

In spoken language, the specific sounds created by using phrases and tones of voice (intonation) are the most crucial gadgets used to speak. sign language is based totally at the idea that sight is the maximum useful tool a deaf individual has to talk and get hold of information. as a result, ASL uses hand form, role, and motion; frame actions; gestures; facial expressions; and different visible cues to shape its phrases. Like another language, fluency in ASL takes place most effective after a long period of examine and practice.[1]

even though ASL is used in america, it is a language absolutely become independent from English. It contains all the fundamental capabilities a language needs to function on its very own–it has its own policies for grammar, punctuation, and sentence order. ASL evolves as its customers do, and it also lets in for local utilization and jargon. each language expresses its functions otherwise; ASL isn't any exception. whereas English audio system frequently sign a question through using a selected tone of voice, ASL customers do so by means of elevating the eyebrows and widening the eyes. every so often, ASL customers might also ask a question by way of tilting their our bodies forward whilst signaling with their eyes and eyebrows. just as with other languages, unique approaches of expressing thoughts in ASL vary as plenty as ASL users themselves do. ASL users may select from synonyms to express not unusual words. ASL also changes regionally, simply as certain English phrases are spoken otherwise in distinctive elements of the u . s . a .. Ethnicity, age, and gender are some extra elements

that affect ASL usage and contribute to its range.[1]

I.1.3 Why Does American Sign Language Become A First Language For Many Deaf People?

Parents are regularly the supply of a child's early acquisition of language. A deaf child who is born to deaf mother and father who already use ASL will start to collect ASL as clearly as a listening to child choices up spoken language from listening to dad and mom. but, language is received differently by a deaf toddler with listening to parents who've no previous enjoy with ASL. a few hearing dad and mom pick out to introduce sign language to their deaf kids. hearing dad and mom who pick to research signal language often study it along side their child. nine out of every ten children who are born deaf are born to parents who hear. other communication models, primarily based in spoken English, exist aside from ASL, together with oral, auditory-verbal, and cued speech. as with every language, interplay with different kids and adults is likewise a considerable element in acquisition.[1]

I.2 Sign language (overview/general context)

All living human languages, are constantly changing, and ASL is no exception. In this dictionary, we offer a description of ASL as it now exists. Loan words to describe new processes and technologies, borrowed terms from sign languages elsewhere in the world, and the inventiveness of its native speakers will doubtless alter and enrich ASL in the coming years, and we look forward to keeping up with that growth.[2]

I.2.1 General Arrangement of the Dictionary

This dictionary is divided into three major sections, each with a specific purpose. Witch are :

I.2.2 Organization of the Handshape Sections

American Sign Language, unlike many spoken languages, does not have a written form. A sign conveys a concept, not an English word, and the production of a sign involves five parameters that need to be described: the handshape, the palm orientation, the location, the movement, and the nonmanual signals. The signs in a dictionary have to be ordered by one of these components so they can be easily located.[2]

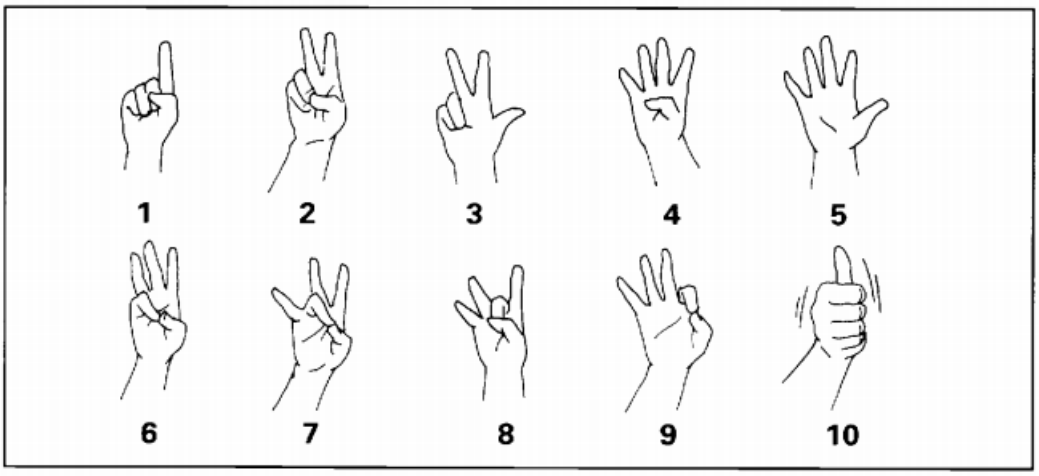
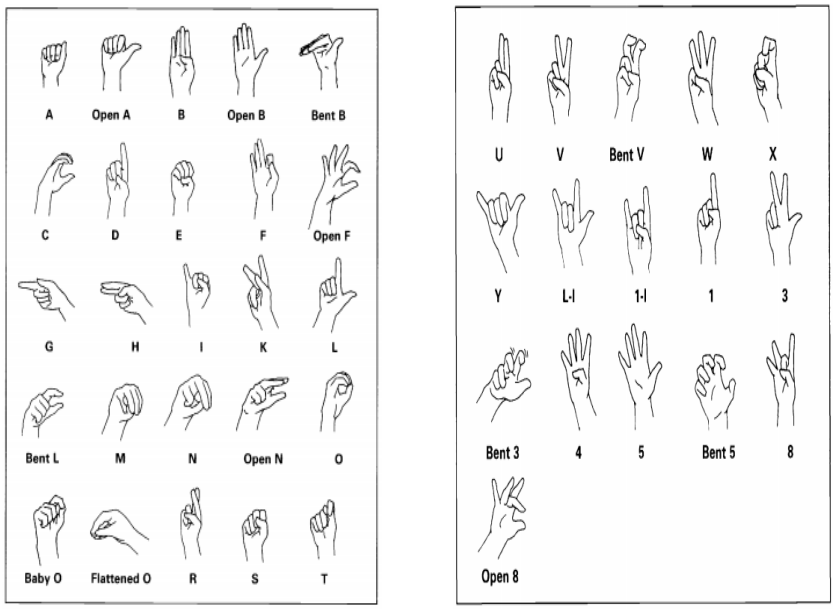


Figure I.1: The manuel numbers

Some letters of the American Manual Alphabet and some number handshapes are not included among the handshapes. The reason for this is that some handshapes are identical except for palm orientation (for example, H. and U, K and P, 2 and V). In the context of forming a sign, palm orientation is determined by the



(a) (b)

Figure I.2: The handshapes used in this dictionary are presented in morphological order

sign, not by the handshape. The decision as to which of two identical handshapes was selected and which was eliminated was arbitrary.[2]

I.2.3 Ordering of Signs within Handshape Sections

The number of hands used to make a sign is one of the easiest things to observe about sign production. This dictionary is divided into two handshape sections: the one-hand signs and the two-hand signs. This organization helps narrow the search for a sign to one of two large locations in the book. The handshape at the top of the even-numbered pages indicates the handshape of the first sign on the page. The handshape at the top of odd-numbered pages indicates the handshape of the last sign on the page.[2]

One-Hand Signs The one-hand signs have one more level of ordering-signs that are formed in space and do not touch the body appear before those signs that do:

Two-Hand Signs the two-hand signs are presented in the following sequence:

1. Hands have the same handshapes and move simultaneously
2. Hands have the same handshapes and move alternatingly
3. Hands have the same handshapes, but only the dominant hand moves

4. Hands have different handshapes, and the hands are ordered first by the handshape and movement of the dominant hand and second by the handshape of the passive hand. [2]

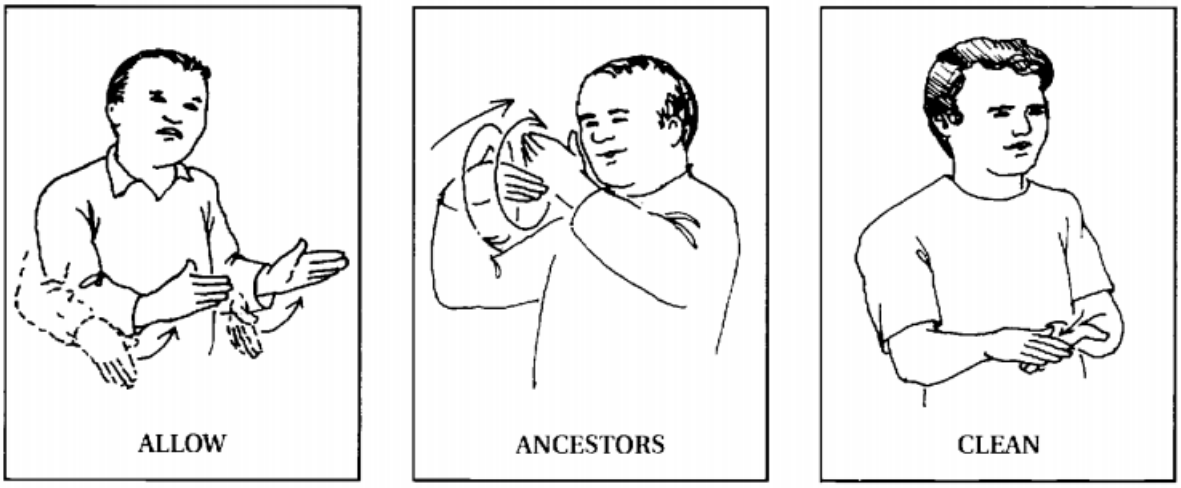
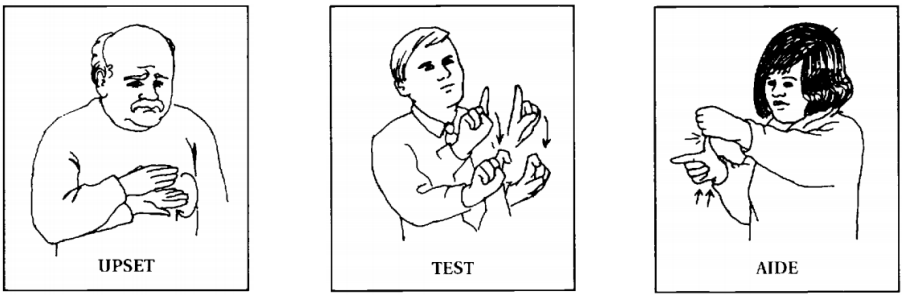


Figure I.3: This order is important for locating signs in very large handshape categories



(a) (b) (c)

Figure I.4: Techniques for Using the Handshape Sections

I.3 Conclusion

In this chapter, we presented the Sign Language generally and with details and it's basics and it's main types and properties, and we dived into the American Sign Language also because it is the one we are going to use in our project, in the next chapter, we will review some similar projects.

Chapter II

Literature review

II.1 Introduction

The care and interesting became greater in last decades from all over the world towards people with disabilities, The existence of a program for the deaf-mute community to help them communicating easily with others becomes more and more necessary.
In this chapter we will discuss some related works, trying to demonstrate the points that others failed to cover and the strong points in those systems. Explaining the differences between them, and what will our system include eventually.

II.2 Sign Language transcribing systems and solutions

In the recent years, with the evolution of AI (Artificial Intelligence), and it's techniques such as Machine learning, deep learning and object detection, it became easier to realize a system on any platform (Mobile apps, Desktop apps,...) for the community of deaf-mute, because of the unlimited abilities and potentials that we can achieve with AI, object detection and machine and deep learning.
To create a sign language transcriber, there are two main approaches or solutions:

- Software based solutions
- Hardware based solutions

Software based solutions: This type is totally based on creating a solid and efficient system based on the power of AI.

Hardware based solutions: In this type the capturing of gestures and detecting them, is based on a specific type of hard wares.

II.3 Related works

We are going present some of the similar works that have been realized, in order to get more understanding on the methods used to get to our destination.

II.3.1 Software based solutions

In the software based solutions, there are so many different methods used, we are going to mention two different ones to show the variability in options.

Sign Language to Text Conversion for Dumb and Deaf

This project is presented in the Indian Institute of Information Technology based in Allahabad. realized by:

- Luv
- Nitin Raj Singh
- Ravi Chandra
- Sheldon Tauro
- Siddhant Gautam

Under the supervision of **Dr. Vrijendra Singh**.
The system is a vision based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.
To create the data set they have used OpenCV to take pictures of different gestures of sign language, they have used fingerspelling as a choice for their system. They capture each frame shown by the webcam of their machine. In each frame they have define a region of interest (ROI).
From this whole image they extract their ROI which is RGB and convert it into gray scale Image.
Finally they apply a gaussian blur filter to image which helps extracting various features of image.

Gesture classification: The approach uses two layers of algorithm to predict the final symbol of the user:

1. Algorithm layer 1:
   (a) Apply gaussian blur filter and threshold to the frame taken with opencv to get the processed image after feature extraction
   (b) This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word
   (c) Space between the words are considered using the blank symbol.
2. Algorithm layer 2:
   (a) detect various sets of symbols which show similar results on getting detected.
   (b) then classify between those sets using classifiers made for those sets only.

Discussion: After the study we made on this project, we noticed that using the fingerspelling in such a like system will do no use, and will be time consuming for the users, along side the need of a huge number of images with good to great quality to get good accuracy. Also we can mention the problem of detecting letters with motions like "Z" and "J".

Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison

This article from The Australian National University, Australian Centre for Robotic Vision (ACRV), presented by:

- Dongxu Li
- Cristian Rodriguez Opazo
- Xin Yu
- Hongdong Li

In this article, the goal was to create the largest dataset for Word Level American Sign Language, with numbering 2000 words.
To create the dataset, they have worked with actors and volunteers from all over the world to record short videos of ASL, and also collecting videos from the internet from specialized websites.
The number of collected videos were more than 21000 videos, their intention was to prevent the problem of detecting words with motion by using videos instead of images.
The methods they used to train and test the dataset were:

- Pose-GRU
- Pose-TGCN
- VGG-GRU
- I3D

In the figure II.1, we see the comparison of accuracy for those models:

Method	WLASL100			WLASL300			WLASL1000			WLASL2000		
	top-1	top-5	top-10	top-1	top-5	top-10	top-1	top-5	top-10	top-1	top-5	top-10
Pose-GRU	46.51	76.74	85.66	33.68	64.37	76.05	30.01	58.42	70.15	22.54	49.81	61.38
Pose-TGCN	55.43	78.68	87.60	38.32	67.51	79.64	34.86	61.73	71.91	23.65	51.75	62.24
VGG-GRU	25.97	55.04	63.95	19.31	46.56	61.08	14.66	37.31	49.36	8.44	23.58	32.58
I3D	**65.89**	**84.11**	**89.92**	**56.14**	**79.94**	**86.98**	**47.33**	**76.44**	**84.33**	**32.48**	**57.31**	**66.31**

Figure II.1: Comparison of the models [3]

II.3.2 Hardware based solutions

We are going to talk briefly about this approach, because simply the used hard wares are expensive and not available for anyone such as electric gloves and the microtouch sensor.
we can mention the project of **Design And Implementation Of Sign Language Translator Using Microtouch Sensor** made by **Nithyakalyani.K, S.Ramkumar, K.Manikandan** as an example of this kind of solutions.

Discussion: Using hard wares if not eeficient at all in public places, time consuming to get ready to use and expensive to be used everywhere.

II.4 Conclusion

In this chapter, we presented an overview of the most important methods used to the transcribing of sign lanuage, as well as some effective systems based on both software and hardware. We also categorized similar works to represent their methods and approaches to show their results and try to criticize them. In the next chapter, we will describe the general architecture of our system, the used methods and approaches.

Chapter III

System Architecture

III.1 Introduction

The design phase is a important step in determining the technical picks. in this chapter,we will suggest an architecture of an integrated system with two elements; hardware and software program, give an explanation for in details the interrelation between them and the function of every module of our system.

III.2 Architecture

To identify the system and the way it works, we are going through Architecture of the different parts that create the system.

III.2.1 Data collection

In this script which is a part of the general script, this part is responsible on collecting data for the entire system. In the figure III.1 an explanation of the steps taken in collecting data phase.

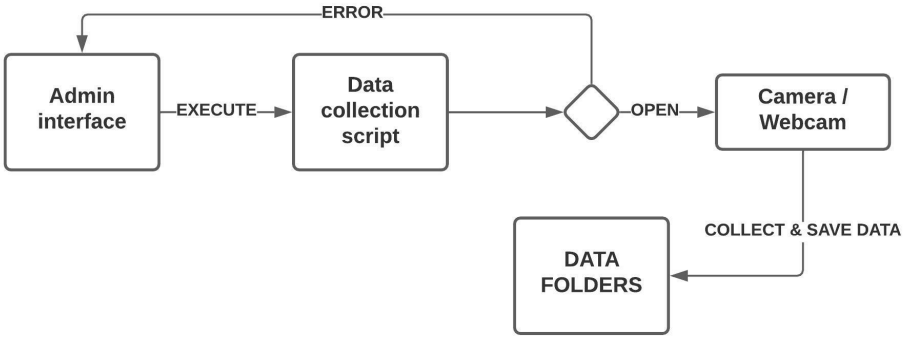


Figure III.1: Data collectinon phase

This script will open the camera of the used device, while reading the feed from the camera, while then the detection of keypoints (right hand and left hand) starts from the feed, for each word we are going to collect 20 sequences (videos), and every sequence is formed by collecting 30 frames.

To organize the data collecting phase, we have made breaks between every word and another, to give time to the user to be prepared to take another word.

The results will be stored in files, each file is labeled with the word label, and in every file there will be 20 file numbered from 0 to 19 (number of the sequences for

each word), the results are numpy arrays which contains the coordinates (x, y and z) of the hands landmarks collected from the frames, so as we took the sequences with 30 frames per sequence, we will get 30 npy arrays for each sequence.

III.2.2 Build the model

This phase is important and very critical, in the figure III.2 the explanation of this phase.

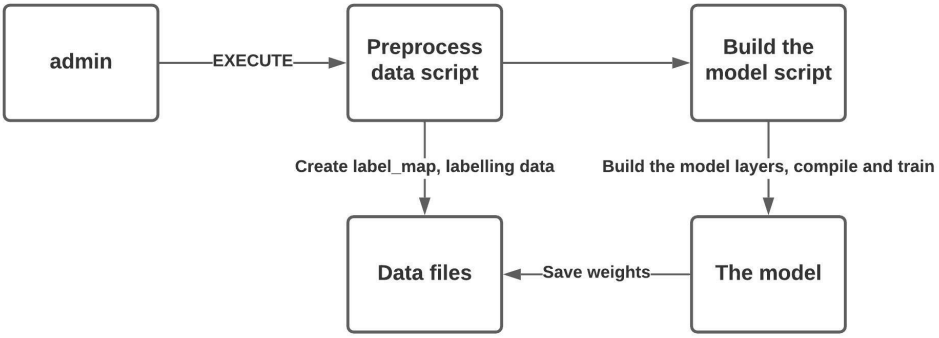


Figure III.2: Build model phase

In the figure III.3 we show the 6 layers of the model we builted

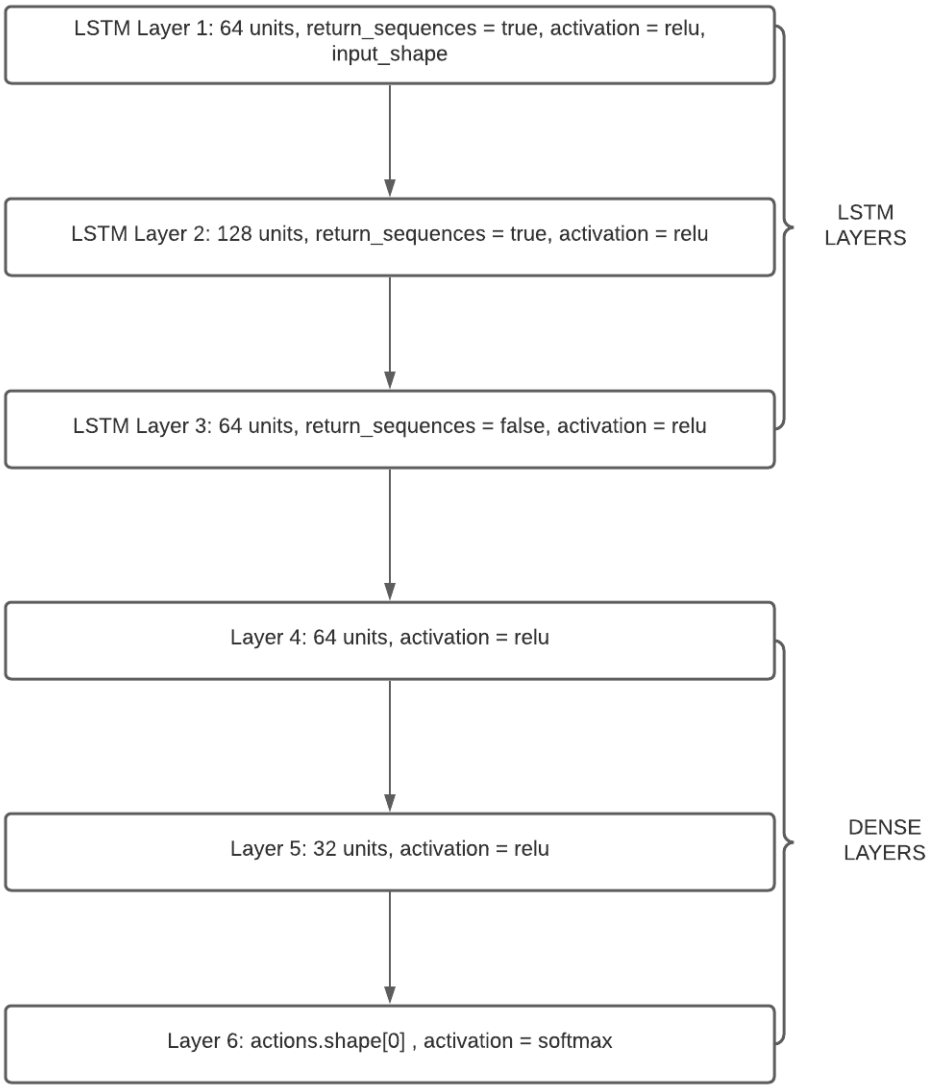


Figure III.3: Model layers

Explaining the layers parameters: In the first two LSTM layers we made the "return sequence' value "true", because when we use LSTM layers with tensorflow we need to pass the results to next layer(results here are in form of sequences), for the "input shape" we defined it only in the first layer, it contains and defines the number of frames and the number of landmarks collected for sequence.
In the third layer we have changed the "return sequence" value to "false" because the next layer is a dense layer and we dont need to pass the results to it. In the last layer the "actions.shape[0]" is the number of actions (words) collected in the first phase.
For the selection of the "relu" and "softmax" activation were based on recom-

mendations in the site of stackoverflow, with a notice that we can change them to other types.

After building the model, we are going to compile and train the model, and also see the progress of the training in real time via Tensorboard, and finally saving the weights. we can evaluate the model and it's results when we done training the model

III.2.3 Real time detection

In this phase which is the last one, Firstly loading the weights saved from after the training, and then launch the system, the figure explain the steps.

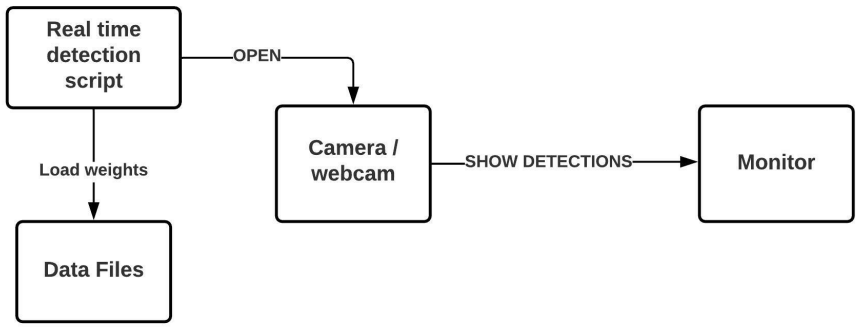


Figure III.4: real time detection

III.3 The sign language transcribing system

The sign language system works mainly in two main parts, the data collection part and then the real time detection part, as for building the model is considered as a one time operation, in case of we wanted to add more words via the data collection script, we just need to train the model using the newely collected data. In the figure III.5 is the general architecture of the system.

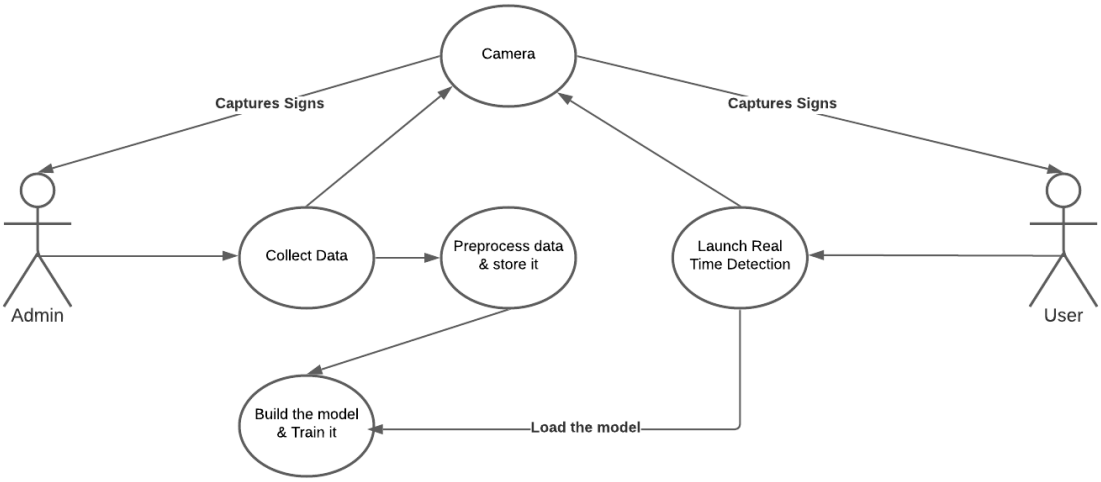


Figure III.5: The general architecture

III.3.1 Improvements

After studying and trying some approaches in order to get to the best results from our program, we noticed that there is two factors affects directly in the final results and the acccuracy of the transcriber, so in our system we have aimed to optimize two main factors:

1. Images quality (Sensors)
2. Faster Training

Image quality: Simply in the phase of collecting data, we depend on the sensor (camera), we have used a laptop webcam, which provides a low quality images and videos, and also tried attaching a phone camera and use it instead of the laptop's webcam, we had better image quality, but it remains a concern, but when using mediapipe and tracking hands landmarks, the image's quality becomes unnecessary because it does not require a high quality image to track hands, on the opposite side when we tried working with the "ssdMobileNet" we had a lot of issues just because of the image quality.

Faster training: As we mentioned before, we have tried to work with "Tensorflow" and the "SSD MobileNet V2" combined, but this combination was very time and hardware consuming, on the other side working with "Tensorflow" combined with "LSTM and Mediapipe" was a difference maker, much less time in training the model with less computional power.
Also getting a better accuracy after evaluating and comparing the two approaches.

III.4 Conclusion

In this chapter, we have presented how the transcribing system works and how its different parts interacts and the relation between them. In the next chapter, we will provide defintions of the languages and the environements we used to implement the transcribing system, also show the evaluation and final results.

Chapter IV

Experiment and results

IV.1 Introduction

In this chapter we are going to define the objectives and requirements of the project.Besides of that we are going to define and explain most of the software we are using in this project. Along with a description of the algorithms implementation and the obtained results, followed with a discussion.

Requirements The minimum requirements for this project are:

1. Python
2. Tensorflow
3. LSTM (Long Short Term Memory)
4. Mediapipe
5. Computer/ laptop
6. Camera

IV.2 Definitions

IV.2.1 Tensorflow

Tensorflow is an open-source library made for numerical computation, also large-scale machine learning that ease Google Brain Tensorflow, such as processes of:

- Acquiring data
- Training models
- Serving predictions
- Refining future results

Tensorflow uses Python as a convenient front-end and runs it in optimized C++, also bundles Machine learning and Deep learning models and algorithms, Tensorflow permit to developers to create and manipulate graphs of computations to perform. [4]

Each node in the middle row represents the arithmetic and each connection represents the data. This allows the developer to carefully consider all the assumptions of the application instead of interfering with the content like figuring out the necessary way to connect to the output of a single smile work on the idea of another. [4]

Google Brain, Google's deep learning artificial intelligence research team, developed TensorFlow for internal use by Google in 2015. This open library is being used by research teams to do a lot of important work. TensorFlow is the most popular software site today. There are many applications in many areas that make TensorFlow popular. TensorFlow, an open source library for deep learning and machine learning, confirms its role in text-based applications, image recognition, voice search, and more. DeepFace, Facebook's image recognition system, uses TensorFlow for image recognition. Used by Apple's Siri for voice recognition. Every Google app we use makes good use of TensorFlow to improve our experience.[4]

IV.2.2 LSTM (Long Short Term Memory)

Introduction: long short term memory community is an advanced RNN, a sequential network, that permits information to persist. it is able to dealing with the vanishing gradient trouble faced by using RNN. A recurrent neural network is also known as RNN is used for persistent memory.[5]

LSTM Architecture At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM consists of three parts, as shown in the figure IV.1 below and each part performs an individual function.[5]

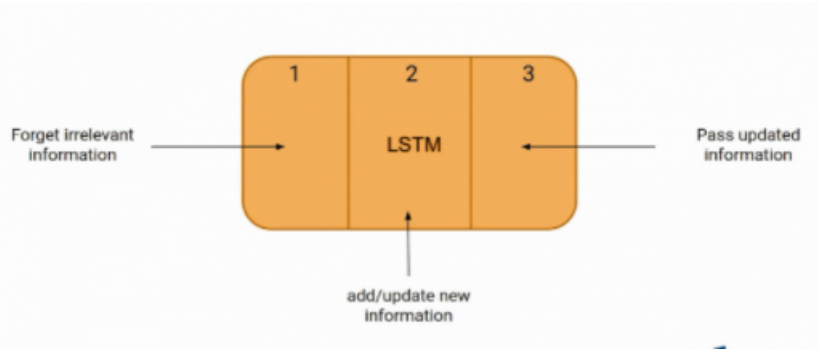


Figure IV.1: explanation of LSTM Architecture (1) [5]

The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp.
These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate. [5]

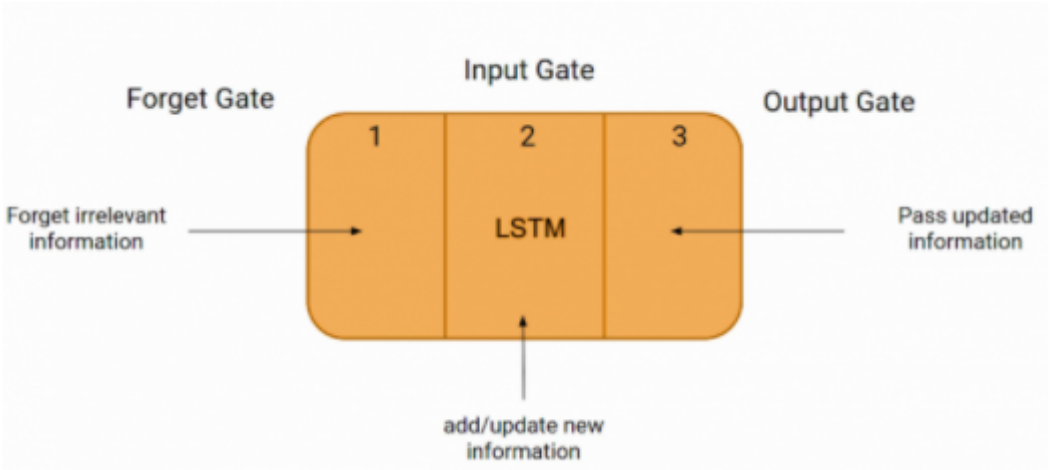


Figure IV.2: explanation of LSTM Architecture (2) [5]

Just like a simple RNN, an LSTM also has a hidden state where H(t-1) represents the hidden state of the previous timestamp and Ht is the hidden state of the current timestamp. In addition to that LSTM also have a cell state represented by C(t-1) and C(t) for previous and current timestamp respectively.
Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following figure IV.3. [5]

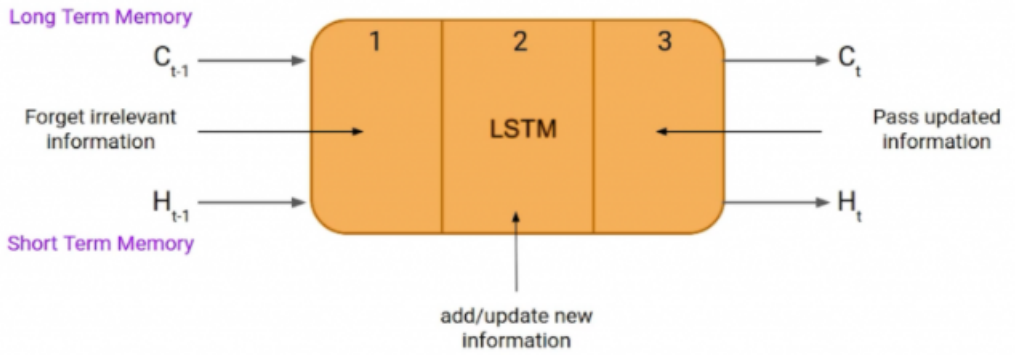


Figure IV.3: explanation of LSTM Architecture (3) [5]

IV.2.3 Mediapipe Holistic

Overview: Live perception of simultaneous human pose, face landmarks, and hand tracking in real-time on mobile devices can enable various modern life applications: fitness and sport analysis, gesture control and sign language recognition, augmented reality try-on and effects. MediaPipe already offers fast and accurate, yet separate, solutions for these tasks. Combining them all in real-time into a semantically consistent end-to-end solution is a uniquely difficult problem requiring simultaneous inference of multiple, dependent neural networks [6].

ML Pipeline: The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. However, because of their different specializations, the input to one component is not well-suited for the others. The pose estimation model, for example, takes a lower, fixed resolution video frame (256x256) as input. But if one were to crop the hand and face regions from that image to pass to their respective models, the image resolution would be too low for accurate articulation. Therefore, we designed MediaPipe Holistic as a multi-stage pipeline, which treats the different regions using a region appropriate image resolution [6].
First, we estimate the human pose (top of Fig 2) with BlazePose's pose detector and subsequent landmark model. Then, using the inferred pose landmarks we

derive three regions of interest (ROI) crops for each hand (2x) and the face, and employ a re-crop model to improve the ROI. We then crop the full-resolution input frame to these ROIs and apply task-specific face and hand models to estimate their corresponding landmarks. Finally, we merge all landmarks with those of the pose model to yield the full 540+ landmarks [6].

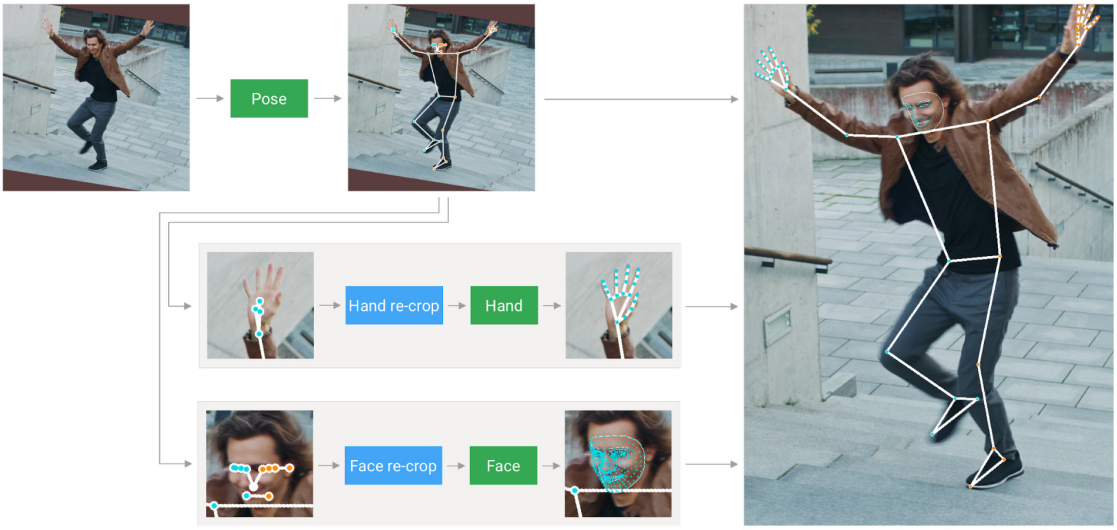


Figure IV.4: MediaPipe Holistic Pipeline Overview [6]

To streamline the identification of ROIs for face and hands, we utilize a tracking approach similar to the one we use for standalone face and hand pipelines. It assumes that the object doesn't move significantly between frames and uses estimation from the previous frame as a guide to the object region on the current one. However, during fast movements, the tracker can lose the target, which requires the detector to re-localize it in the image. MediaPipe Holistic uses pose prediction (on every frame) as an additional ROI prior to reduce the response time of the pipeline when reacting to fast movements. This also enables the model to retain semantic consistency across the body and its parts by preventing a mixup between left and right hands or body parts of one person in the frame with another (?, ?).

In addition, the resolution of the input frame to the pose model is low enough that the resulting ROIs for face and hands are still too inaccurate to guide the re-cropping of those regions, which require a precise input crop to remain lightweight. To close this accuracy gap we use lightweight face and hand re-crop models that play the role of spatial transformers and cost only 10 per cent of corresponding model's inference time [6].

The pipeline is implemented as a MediaPipe graph that uses a holistic landmark

subgraph from the holistic landmark module and renders using a dedicated holistic renderer subgraph. The holistic landmark subgraph internally uses a pose landmark module , hand landmark module and face landmark module. Please check them for implementation details [6].

IV.3 Platforms

To Create any system choosing the right platform is a must, and for our system, we have chosen:

IV.3.1 Anaconda

What is Anaconda? From the official blog of Anaconda we quote:
"Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux." "Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer." "It was created for Python programs, but it can package and distribute software for any language." [7]

Why Anaconda? We can answer this question in into two words "Dependencies and Environments" Anaconda takes care automatically of both of them so you will not have to worry about them, that means Anaconda will save you time honestly a lot of time.[8] The more explaining answer is that open source software often needs to be compiled from your system's source code, often from scratch. The process of making the source code one that can be copied to your system is called a compilation, which can be a tedious and time-consuming process.[8] Anaconda is the distribution of packages built for data science. Comes with Conda, package and environment manager. We typically use conda to create isolated environments for our projects that used different versions of Python and / or different package versions. We also use it to install, uninstall and update packages in our project environments. When you first download Anaconda, it comes with conda, Python, and more than 150 scientific packages and their dependencies. [8]

To write our code we used Jupyter Notebook, which is a Python IDE provided in Anaconda

Jupyter Notebook Jupyter notebook is one the top 5 IDEs for Python, Jupyter notebook was founded in 2014 out of IPython, is defined as a web application

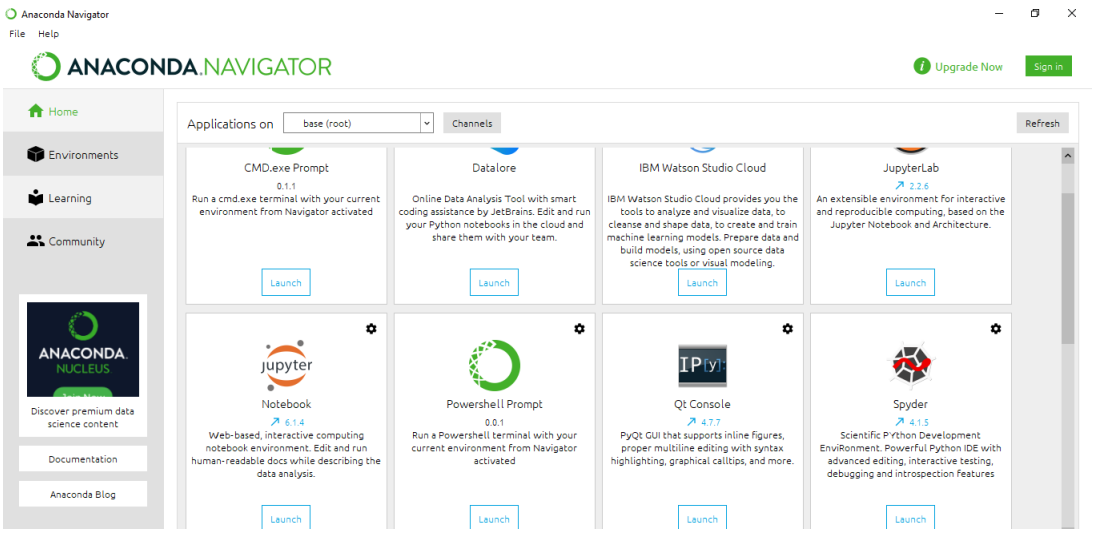


Figure IV.5: Anaconda Navigator Interface

structured on the Server-Client structure, and it gives you the ability to create manipulate notebook documents.
Jupyter notebook provides you with easy-to-use interactive learning materials (data science) across a variety of programming languages. It is perfect for people who started their journey in data science newely.[9]

Features of Jupyter notebook We are going to name some of many important features offered by Jupyter notebook [9] such as:

1. Supports Markdowns
2. Allow you to add HTML components from images to videos
3. See and edit your code in order to create compelling presentations (use data visualization libraries such as Matplotlib to show graphs in the same document where your code is)
4. Export you work to PDF and HTML files, or just export it as a py file
5. Create blogs and presentations from notebooks

IV.4 Used Materials

We have used two main computers in this project, a laptops and a desktop computer

1. Desktop computer was used to write the main scripts such as the main code and other scripts:
    - Processor AMD Ryzen 3 3200G with Radeon Vega Graphics, 3.60 GHz
    - Installed RAM 8.00 GB (5.95 GB usable)
    - Operating System: Windows 10 64-bit, x64-based processor
2. The second unit, a gaming laptop MSI GF65, we used it to train the model because it is the most powerful one:
    - Processor: Intel® Core™ i5-9300H CPU @2.40GHz
    - Installed memory (RAM): 8.00 GB (7.85 GB usuable)
    - Operating System: Windows 10 64-bit, x64-based processor
    - Graphic Card GPU: Nvidia GeForce RTX 2060

IV.5 Implementation

After going through the platform and defining it and the requirement of the project, now we will explain the implementation of the system. Our system is basically a software based solution, in which we dont need any programmed hardware, we just need a camera.

IV.5.1 Software

As we said before our system is pure software based solution, we are going explain the main scripts in our system.

Detection function

This function is to detect the landmarks from the images (the feed), in figure IV.6 the function's script

```
#detection Function
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results
```

Figure IV.6: Mediapipe detection function

In this function return two variables, the image "frame" after processing it, and the results which are the landmarks.

Drawing functions

we will start with the drawing functions, we made them just to visualize the hands landmarks during collecting data, to make sure in every sequence we captured the signs in the right position. in the figures IV.7, IV.8 their scripts:

```
def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)
```

Figure IV.7: Draw landmarks function

```
def draw_styled_landmarks(image, results):

    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
```

Figure IV.8: Draw styled landmarks function

Extracting points function

Tihs function's role is to extract (x,y,z) coordinates values from the frames captured, and concatinates them in a single array. script in figure IV.9

```
def extract_keypoints(results):
    left_hand = np.array([[res.x, res.y, res.z]
                          for res in results.left_hand_landmarks.landmark]).flatten():
                                if results.left_hand_landmarks else np.zeros(21*3)
    right_hand = np.array([[res.x, res.y, res.z]
                          for res in results.right_hand_landmarks.landmark]).flatten():
                                if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([left_hand, right_hand])
```

Figure IV.9: Extracting points function

Seting up folders

this part is to create folders to store collected data in them, see figure IV.10

```
# path of the collected data
DATA_PATH = os.path.join('MP_Data')

#Actions
actions = np.array(['hello','thank you','i love u'])

# number of videos for every word/action
nbr_sequences = 5

# number of frames per video we can add more frames here to take a longer sequence
sequence_length = 30
```

```
for action in actions:
    for sequence in range (nbr_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

Figure IV.10: Seting up folders

Collecting data for trainig and testing

This part is one of the three main parts, in the figure **??** the code of this part.

```
cap = cv2.VideoCapture(0)
# Access to the mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    # loop in actions
    for action in actions:
        # loop through sequences/videos
        for sequence in range(nbr_sequences):
            # loop through video length
            for frame_num in range(sequence_length):


                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)
                draw_styled_landmarks(image, results)

                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (120, 255, 50), 4, cv2.LINE_AA)
                    cv2.putText(image, 'COLLECTING FRAMES FOR {} video number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1, cv2.LINE_AA)
                    cv2.waitKey(5000)
                else:
                    cv2.putText(image, 'COLLECTING FRAMES FOR {} video number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)

                cv2.imshow('Feed', image)

                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)



                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break
    cap.release()
    cv2.destroyAllWindows()
```

Figure IV.11: Data collection script

In this part of the code, we launch the camera using opencv library, reading the feed (frames) from the camera processing each frame and make some modifications on the input frames and show then show the outputs as a final results. To be more explicatif, we take each frame, apply the "Mediapipe Detection function" to get the hands keypoints, while also decorating the frames with the "Drawing function", in this step we intend to take sequence of 30 frames of a specific sign, so we needed to apply some logic like time breaks between each sequence to give the time for the user to adjust his hands and body position for the next

sequence or the next word, also showing on the screen messages of name and the countdown of the word's sign and the sequence number, to organize to data collection.
At the end we store the keypoints in the folders specified in form of numpy arrays.

Preprocess data and labeling

Here just simple preprocessing operation, which is labeling the actions we captured and stored as numpy arrays, and create a label map, see figure IV.12

```
label_map = {label:num for num, label in enumerate(actions)}
```

```
sequences, labels = [],[]
for action in actions:
    for sequence in range(nbr_sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

Figure IV.12: Preprocessing and Labelling

Build and train the model

In this part, we have initialized and builded the model which contain 6 layers, then compiled it, and finally trained it for 5000 epochs(steps), see figure IV.13, IV.14.

```
model = Sequential()
model.add(LSTM(64, return_sequences = True, activation = 'relu', input_shape = (30,126)))
model.add(LSTM(128, return_sequences = True, activation = 'relu'))
model.add(LSTM(64, return_sequences = False, activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(actions.shape[0], activation = 'softmax'))
```

Figure IV.13: Build and train the model

After the training is done we save the weights in an ".h5" file.

```
model.compile( optimizer='Adam', loss= 'categorical_crossentropy', metrics=['categorical_accuracy'])
```

(a)

```
model.fit(x_train, y_train, epochs=2000, callbacks=[tb_callback])
```

(b)

Figure IV.14: Compile and Train the model

Real time detector

This is the final part in the code, here we test the system, see figures IV.15.

```
sequence = []
sentence = []
threshold = 0.8

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1080);
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 640);
```

Figure IV.15: Initialize new variables

Here we initialized new variables such as "sentence", which is an empty array to carry the words extracted as results of detection, and showed on the screen. also we defined the dimensions of the screen.
Then, we launch the camera, and extracting keypoints using the mediapipe detection and extracting keypoints functions, putting the results in the sequence array, this way we are taking the last 30 frames of the feed to process them.
The next step will be processing the frames and detect the actions, if there is any action detected we are going to append it's label to the sentence array if the sentence array has less than 5 words and we made a condition to not repeat the same word two times in a row. Finally showing the array sentence in form of a tape in the screen.see figure IV.16.

```
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        ret, frame = cap.read()


        image, results = mediapipe_detection(frame, holistic)

        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]

            print(actions[np.argmax(res)])

            if res[np.argmax(res)] > threshold:
                if len(sentence) > 0:
                    if actions[np.argmax(res)] != sentence[-1]:
                        sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]
```

Figure IV.16: Real time detector

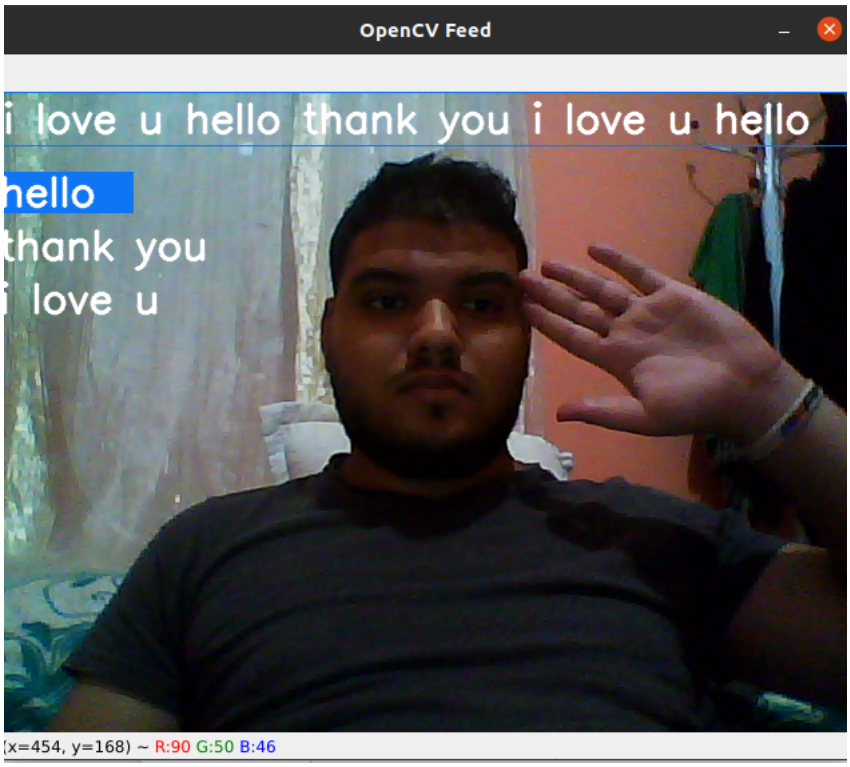


Figure IV.17: Example of Real time detection

IV.6 Results and Discussion

Results

The desired objectives we wanted to achieve, are:

- Make a system not relying on the image's quality, or the environment surronding when using the system
- make a fast and efficient detecting system.

The using of LSTM with Tensorflow, combined with the Holistic Mediapipe, enabled us to achieve what wanted, the training of the model was super fast within minutes we done training, the results were impeccable, All that without the need of using powerful machines, Although we had one issue which is when detecting words and after the system detects a word there is always a word keeps repeating. In figure IV.18, the diagram from Tensorboard of the accuracy value while training, in same time the figure IV.19 shows the epoch loss diagram.

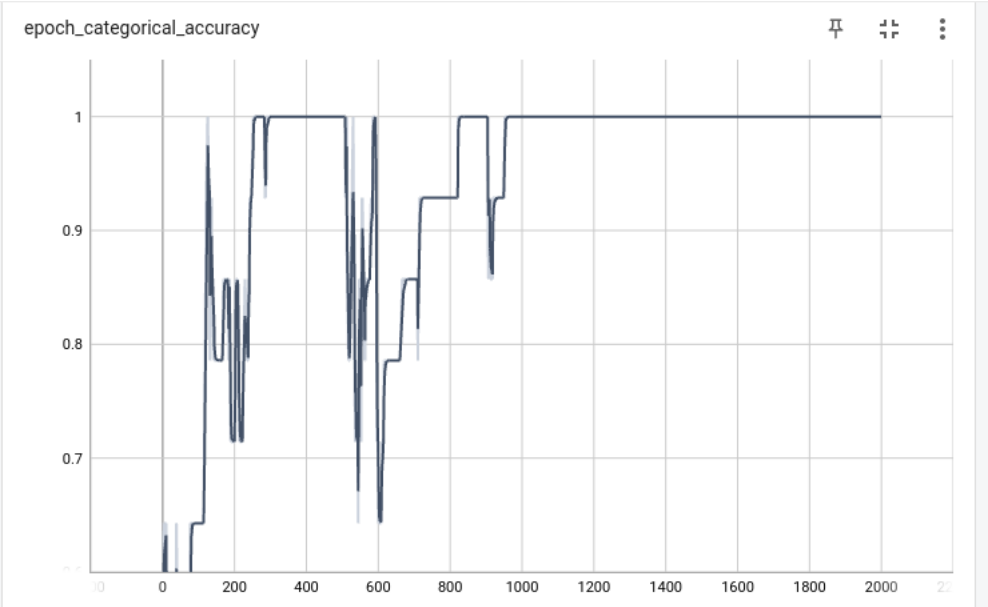


Figure IV.18: Epoch categorical accuracy

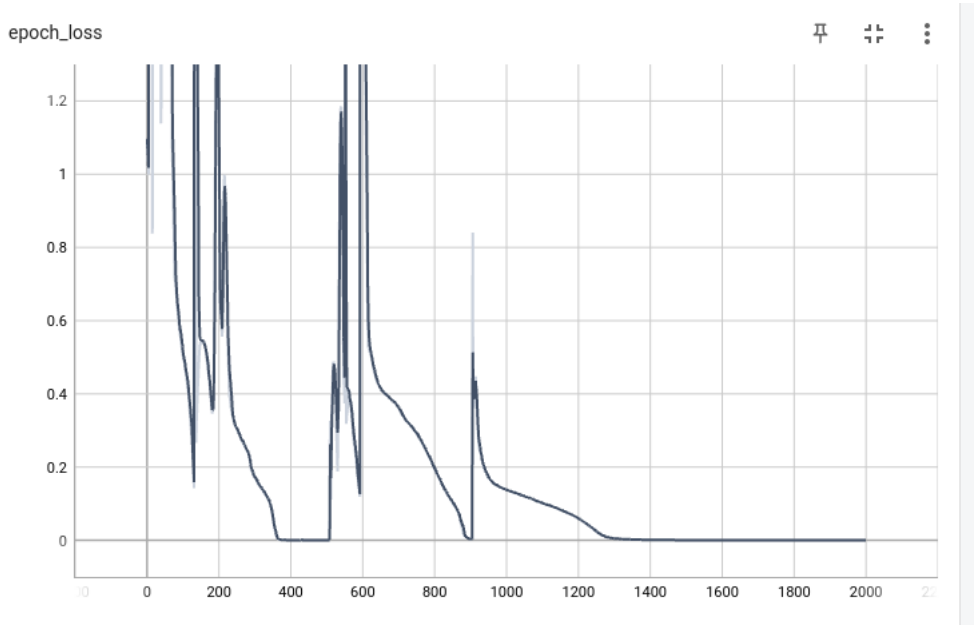


Figure IV.19: Epoch Loss

Discussion

We can notice that the proposed methods succeeded to help achieve the main goals, we this system we can detect the signs with efficiency, regardless to image quality, also for the signs with motions there is no major problems, taking sequences of 30 frames and train the model on them was very helpful.

Future scoope

We intend to make this system adaptable mobiles because it is light and does not require heavy hardware. Also developing a user friendly interface for users. Finally, we want to develop in this same system a vice versa, in mean that starting from a text to a sign, using animated characters.

IV.7 Conclusion

In this chapter, we discussed the platform used and the reason for choosing it. After that, we did go through the implementation. Passing by the result ending with discussion, clarifying some of the advantages and inconveniences of the implemented system.

General Conclusion

In this thesis, we tried to create a word level American sign language transcriber to fulfill the lack of alike systems in our country for the deaf-mute community. The system implementation was made with Anaconda Environment, Python, Mediapipe, Tensorflow and OpenCV. We achieved a system capable of :

- Fast and real time detection.
- Light for low end devices.
- Fast training model
- Accurate and efficient system.

What distinguishes our system from other systems is the low cost and the good accuracy rate. Walking through the different chapters we described and clarify the system architecture, faced limitations, some hypothesis and achieved objectives.

References

[1]: https://www.nidcd.nih.gov/health/american-sign-language.
[2]: The American Sign Language Handshape Dictionary, Richard A.Tennant, Marianne Gluszak Brown, Clerc Books, Gallaauder Universirt Press, Washington DC.
[3]: https://dxli94.github.io/WLASL/
[4]: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html
[5]: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/
[6]: https://google.github.io/mediapipe/solutions/holistic
[7]: https://docs.conda.io/projects/conda/en/latest/
[8]: https://www.datacamp.com/community/tutorials/data-science-python-ide
[9]: https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what-is-jupyter.html