# Anglia Ruskin University

# Final Project

## British sign language translator for medical services

BSLT

*Course Code: MOD002691*

---

*SID:*
1841631

*Supervisor:*
Dr Abdul Adamu

April 25, 2022

# Abstract

Machine learning has been widely implemented in a number of applications ranging from social media and communication to medical sciences and economics. This extensive use of such a technology is mainly due to two factors: its versatility, and its ability to produce results in a relatively efficient manner. One of the most recent and exciting advances in machine learning technologies, known as deep learning, has proven to be particularly effective when it comes to tasks linked to recognition of patterns thanks to its human-like learning method (learning by example). This report analyses the use of this technology in the field of sign language translation, and explores a new approach to the problem relying on a special type of neural networks (LSTM) combined with the mediapipe framework instead of a more traditional approach that requires inserting large pools of labelled images in a CNN (Convolutional Neural Network) or ANN (Artificial Neural Network) for the training. For the training, short videos are labelled to mimic real-life scenarios and enhance the signs prediction by relying on a "frame" average (frames collected from short videos) instead of a single labelled static image. The Mediapipe framework was added to further refine these prediction by extracting human key-points (face, shoulders, arms and hands) to get rid off any inconsistencies due to image variance (background, person, lighting etc...).
Note that the BSLT for medical services is both a research and a software development project which covers existing methods for sign language translation in machine learning while including a new approach that aims at producing a decent SLT model with, considerably, smaller data-sets.

# Contents

# 1 Introduction

As many as one in six people are deaf or hard of hearing, however, hospitals, police and most public services are not legally required and do not provide qualified sign language interpreters.This issue is either due to the lack of financial resources reserved for this cause or due to a failure to make sufficient attempts to obtain qualified interpreters. Moreover, while there are many sign language translating technologies proposed online, none have been adapted or implemented to the field of public services in general and medical services specifically.

This report describes a new approach in machine learning to sign language translation that relies on a different network system for the training than the ones usually adopted for this case scenario.

The following pages will cover, in depth, an attempt at training a sign language translation model for British sign language specific to the medical field along with the process and the technologies involved in the development, such as LSTM networks and mediapipe, along with the explanation for such choices. Some of the existing alternatives and the different approaches that have been proposed for the same problem will be discussed as well.

## 2 Aims

The main objective for this project is making sign language translation available and affordable for hospitals and medical centers through the development of a deep learning model able to translate British Sign Language words related to emergencies, illnesses and medical assistance in general.
This model will be developed by exploring a new method in SLT that relies on extracting human key-points from video frames using the MediaPipe framework, then collecting their coordinates (of the key-points) for each sign and have them be the training dataset. The language used for the development of the model is Python given its flexibility thanks to the various libraries available and useful to this case scenario. Due to the nature of the project, we will rely on Jupyter Notebook as a development environment.

# 3 Background

To develop a deep learning model, two components are required and influence greatly the outcome of the training: the data-set and the neural network. These elements represent, respectively, the inputs and the processes of the training: a neural network include weights of its different nodes and the mathematical formulas that influence the statistics of a model when learning to identify specific patterns. These two components, with the model accuracy, are essential in evaluating the distinctions between the previously proposed approaches for sign language translation and the one presented in this document.
After analysing dozens of solutions offered online for sign language translation, the redundant themes were: the focus on the **American sign language**, and the use of data-sets of **single static images per sign**, resulting from image segmentation, for training and recognition which falls in the category of object detection in the realm of deep learning. Moreover, the majority of the previous work done on sign language recognition focuses exclusively on fingerspelling which solely is not enough to construct an SLT system that is of use to the deaf communities. Furthermore, most models rely on focusing the camera on the hands of the signer alone, with the aim of obtaining clearer images of the signs. A more practical approach should, ideally, be capable of recognizing signs based on hand position, posture, and movements, which are involved in the meaning of many signs, and mix fingerspelling with words signs.

Some of the most unique models that have been proposed so far are:

## 3.1 Real-Time Malaysian Sign Language Translation using Colour Segmentation and Neural Network
## By Rini Akmeliawatil, Melanie Po-Leen Ooi. Ye Chow Kuang
## Monash University

Finger spelling is used in sign language to express words that can't be expressed by signs. This project focuses mainly on finger spelling for the Malaysian sign language while including a few Malaysian words. Regarding the methodology, a camera is used to capture signs performed by a person, before going through video processing to recognise the sign:

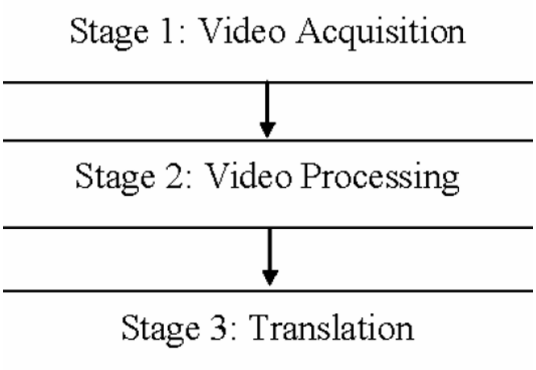


Figure 1: Vision-based Sign Language Translation

It is worth mentioning that most SLT models follow this flow action which serves as a generic plan for all sign language translators rather than being specific to this project.

The diagram specific to this project is presented below:

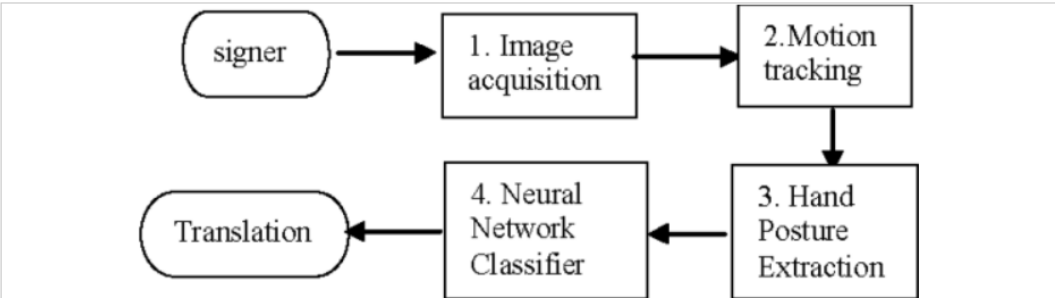


Figure 2: Malaysian Sign Language Translator

In order for the translation to happen, images of the hands are extracted using image segmentation (color segmentation) techniques before being processed to recognise the sign:

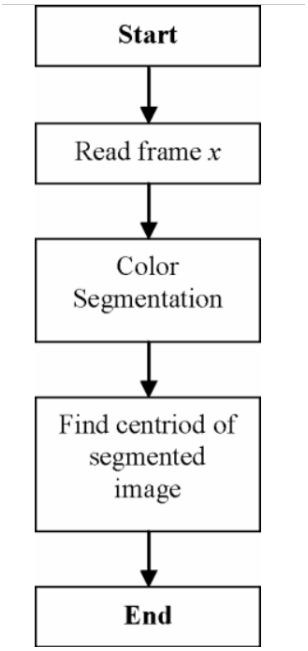

Figure 3: Flow of image segmentation

**Specificity:**
The vast majority of Vision-based Sign Language Translation models require some form of image segmentation (e.g to isolate the hands), usually, through many processes that include color segmentation (only preserve pixels in a specific range of RGB/HSV values). The more effective this first color segmentation, the less processes are needed afterwards to produce an image usable either for training or testing. This project proposes a clever idea to enhance this color segmentation process: by making users wear gloves of a specific color(red or yellow) the choice for color threshold values to apply is normalised and the issues caused by different skin tones across people are no longer relevant (regardless of their skin color, all users are going to be wearing the same gloves).

Regarding the neural network system implemented, this method relies on an Artificial Neural Network (ANN) for the training.
ANN is a network that has multiple neurons at each layer and is also known as a Feed-Forward Neural network due to inputs only being processed in the forward direction(from the input layer to the output layers). This single direction passing of information makes ANN one of the simplest neural network variants. The **advantages ANN** include the ability to store information on the entire network and the ability to work with this in-

formation even if incomplete. ANN networks also offer some form of fault tolerance.However, ANN have some flaws as well such as a huge hardware dependency and they also might display unexplained/unpredictable behaviours sometimes (causing unsatisfactory results). Figure 4 shows the decision process in recognising the signs using ANN (simplified); the sign, when detected (after passing through the word network), is fed to the Alphabet and number decider in order to classify it either as a letter (fingerspelling) or a number before being returned to the word network (figure 4).

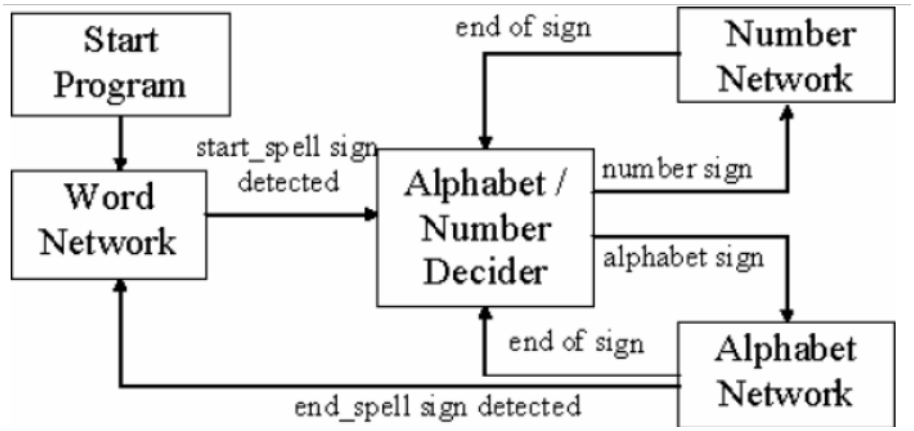


Figure 4: Decision Process using Artificial Neural Network

### 3.2 Statistical Sign Language Translation By Jan Bungeroth, Hermann Ney RWTH Aachen University

In the field of machine translation, significant progress has been made by using statistical methods. In the Statistical Sign Language Translation paper, a statistical machine translation system for Sign Language and written language is proposed for the language pair German Sign Language (DGS) and German. The idea behind this approach is summarized in a model that relies on statistics to predict the meaning of a sign based on the sign preceding it, in order for the sentence to be, to some extent, "coherent". A similar method is used by google to suggest words to add at the end of the text in the search bar or by IOS through predictive text (words are suggested to complete the sentence based on probability). The choice for the sentence with the highest probability is decided by Bayes' decision rule.

Figure 5 shows the general architecture of the statistical translation approach where "e" represents the resulting sentence and f the input sentence (before the prediction).

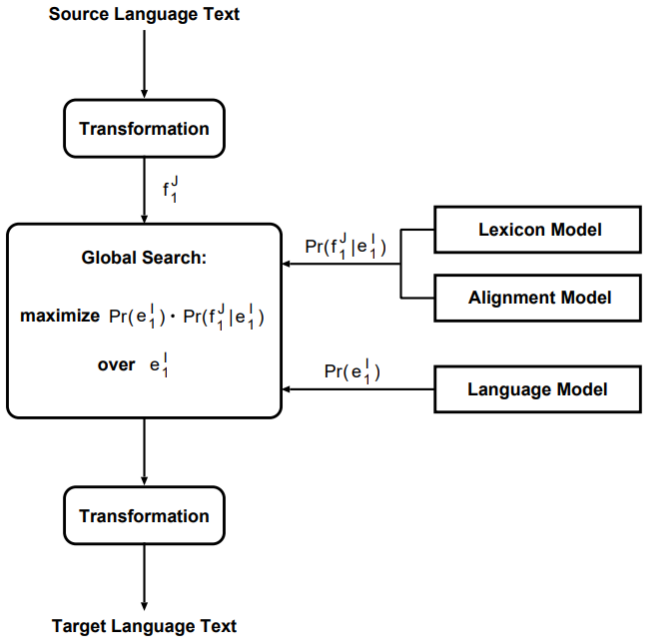


Figure 5: : Architecture of the translation approach based on Bayes' decision rule as proposed by Jan Bungeroth, Hermann Ney

**Specificity:**
This research doesn't discuss in details the process of creating a model for sign language translation from scratch but rather proposes a way to enhance and increase the reliability of an existing one. Moreover, unlike the previous example where a solution for translation only in one way is put forward (from signs to written words), this research explores the possible use and effectiveness of statistical methods for a translation from written German to German Sign Language (images of the corresponding sign are produced). While the use of statistical methods in language translation seems promising, this approach suffers from a serious limitation, the lack of large corpora; in written language, the Hansards corpus with French and English sentences from debates of the Canadian Parliament contains about 1,470,000 sentences while Sign Language corpus doesn't propose more than 2000 sentences influencing directly the prediction accuracy.

### 3.3 Korean Sign Language Translation Using Machine Learning

The Korean Sign Language Translation Using Machine Learning is a project that offers an approach extremely similar to the one proposed in this paper in terms of the sign identification process; they both require the extraction of human keypoints through some third party piece of software (OpenPose library for the Korean SLT and the Mediapipe framework for the British SLT presented in this paper) to overcome, to some extent, the issues caused by relatively small data-sets like the large visual variance as the same sign can look very different depending on the signer.

Figure 6: Example of extracted human keypoints source: the Korean Sign Language Translation Using Machine Learn- ing

After extracting the keypoints from frames on the data-set, they are normalised and their coordinates (for each sign) are inserted as inputs in a RNN network for the training.

**Specificity:**
The strength of this Sign Translation method lies in the use of the OpenPose library for the human keypoint extraction that allows avoiding any issue related to how change in scenery, background or even the signer affect the accuracy of the translation given that anything other than the keypoints, representing limbs and face, is ignored by the computer. It is also worth mentioning that, unlike the majority of Visual-Based models (SLT precisely) where a CNN is the norm when dealing with standard SLT data-sets (thousands of labelled images of signs), an RNN (Recurrent Neural Network see the neural networks section) is used instead for this approach.
Unlike Artificial Neural Networks, RNN's do not pass the information in one direction only: they save the output of processing nodes and feed the result back into the model which is then said to learn to predict the outcome of a layer (see figure 7).

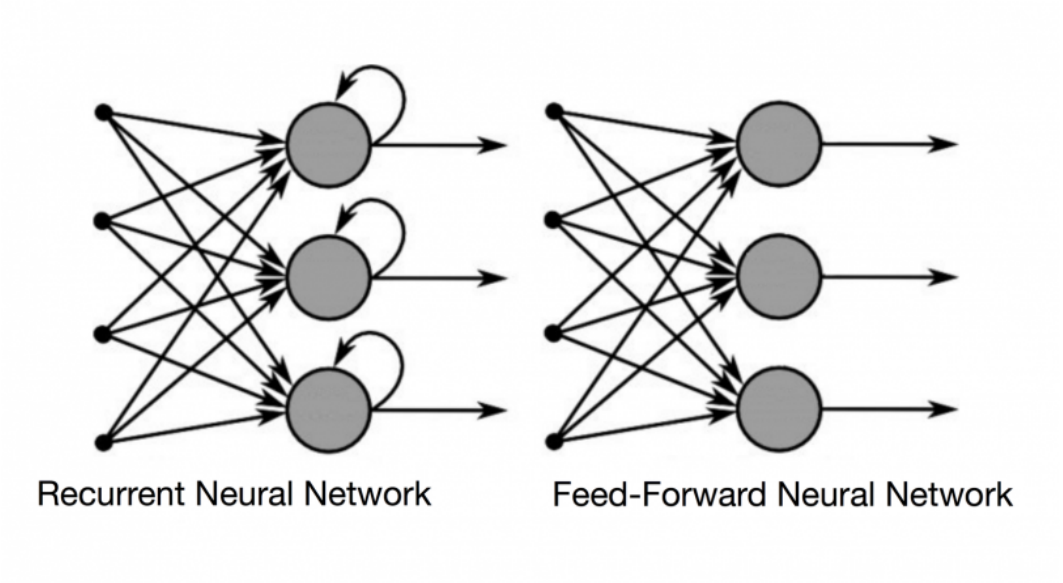


Figure 7: Comparison between a RNN(left) and an ANN(right) information flow

Each node in the RNN model acts as a memory cell continuing the computation and implementation of operations. Moreover, when the prediction of the network is incorrect, the system learns by itself (self-learning) and carries on searching for a correct prediction during backpropagation (when information flows in the opposite direction ie from output to input). The advantages of RNN include the network's ability to remember information through time along with the possibility of combining it with convolutional neural networks layers to extend the effective pixel neighborhood in computer vision. Some of the disadvantages of RNNs are the training being more complex than in the case of other neural networks.Some activation functions (Tahn and relu) do not allow processing long sequences.

# 4 Development of the BSLT for medical services

The BSLT for medical services is a sign language translation model that relies on a relatively unique approach; while most sign language translators use a dataset of labeled images(usually a CSV file) for training through a Convolutional Neural Network which is the most popular type of network for sign language translation and any model that requires processing of images, the BSLT for medical services uses a different kind of dataset and Neural Network.

## 4.1 The Dataset:

The BSLT for medical services main objective is helping the deaf and hard of hearing, in a situation of health emergencies, communicate in a quick and efficient manner to avoid any complications, potentially save anyone from a life-threatening situation or just to allow a group of marginalised members of society to have access to their most basic rights related to medical services without the need of an interpreter who is, usually, not available in this scenarios. Therefore, the BSLT for medical services was developed with the intention of breaking communication barriers and, to achieve this, 91 signs including the 26 alphabet letters and words from the Oxford 3000 emergency word list were originally chosen to create a training and testing dataset. A total of 2275 videos was recorded and a total of 56875 frames were produced (25 frames per video, 25 videos per sign and approximately 5 hours of recording). However, due to the heavy workload that training causes with large datasets, the limitations of the hardware used by the student and the initial accuracy being extremely low and requiring more training time, the list was shortened to 10 words as a sample training to test the reliability of this method before extending it over a larger set of signs when more powerful hardware or more training time is available (the better the hardware and precisely the GPU the less time is required for training and vice-versa). The selected 10 signs are:'doctor','nurse','hospital','ambulance', 'heart attack','accident','weak', 'cough','tablet',and 'it hurts'. Figure 8 summarizes the numerical details of the dataset used for this project.

| Metric | Training | Testing | Total |
|---|---|---|---|
| Number of Signs Collected | - | - | 91 |
| Number of Signs used in the development | 10 | 10 | 10 |
| Number of videos | 126 | 124 | 250 |
| Number of frames | 3150 | 3100 | 6250 |
| Number of signers | 1 [student] | 1 [student] | 1 [student] |

Figure 8: Details of the BSLT for medical services dataset

### 4.2 Methodology

Since this is a deep learning project that relies on a unique approach that requires human keypoints extraction from a set of recorded videos, no open-source dataset was available online and, therefore, the first step is to prepare it. To do so, the signer (in this case the student) records videos of him performing the different signs (from the BSLT) while using the Mediapipe framework offered by google to have the human keypoints extracted and stored in real-time instead of having to process the videos after recording. Mediapipe is an open-source framework offered by google as an efficient tool in Machine Learning for live and streaming media that allows for detecting human features. The main advantages of Mediapipe are its availability to the wide public (free and open source), its end-to-end acceleration (built-in fast ML inference and processing is accelerated even on common hardware which makes it a great choice for development on average or old computers) and being a cross-platform solution (opens doors to a potential deployment of the model on Android and IOS instead of being limited to Windows).
As explained earlier, originally, 2275 videos for 91 signs were recorded, with each video having a length of 25 frames, however, only 10 signs were selected out the 91 leaving us with 250 videos in total (see figure 8).
Once extracted, the keypoints' positions are labelled and organised according to each frame of the video of the sign they represent: that's our dataset. The figure 9 shows an example of some frames of 5 out the 10 sings being recorded while using mediapipe for the extraction of keypoints represented

by the colorful landmarks on the face, the hands, the arms and the shoulders.

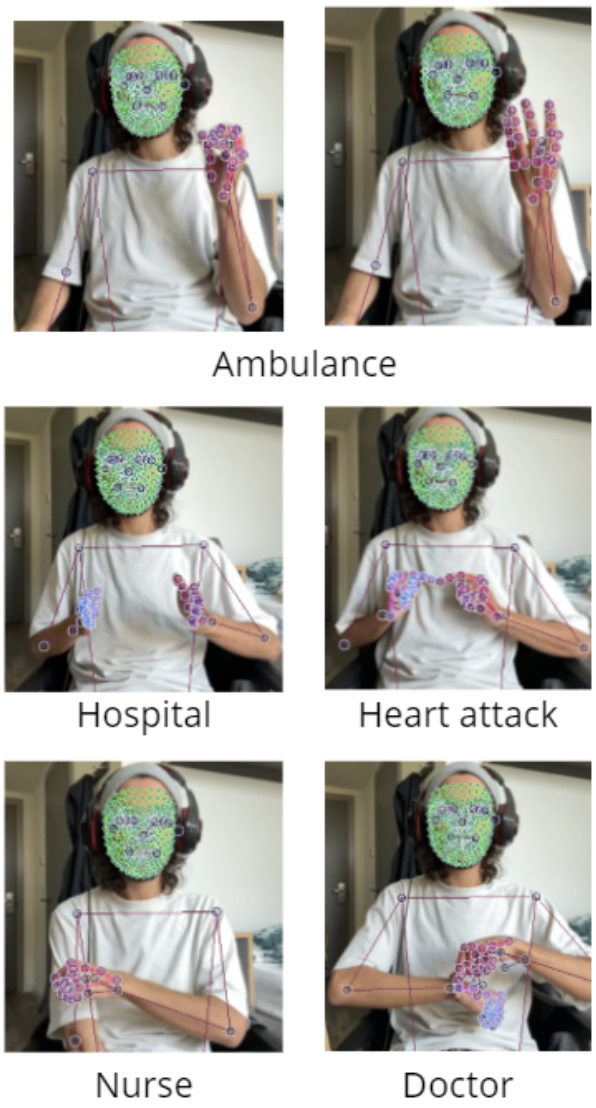


Figure 9: Recorded signs with Mediapipe landmarks (human keypoints)

After obtaining a usable dataset, the next step is to use the Train-Test Split method to have to sub-datasets, one for the training and the other one for the testing (the number of signs is the same of both but the number of videos is halved approximately see figure 8). Now to use this sub-dataset to train the actual model, a neural network is required. The Neural network

built for this project is composed of three LSTM layers and three Dense layers; the Dense layers are just normal fully connected layers, we will therefore ignore them and focus on the LSTM layers where the action detection is happening. Figure 10 summarizes the Neural Network structure, Note that the 25 in the output shape column represents the number of frames and the 128,64,32 and 10 represent the number of units per layer where 10 is the number of signs of our dataset to be precise.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm_42 (LSTM) | (None, 25, 64) | 442112 |
| lstm_43 (LSTM) | (None, 25, 128) | 98816 |
| lstm_44 (LSTM) | (None, 64) | 49408 |
| dense_42 (Dense) | (None, 64) | 4160 |
| dense_43 (Dense) | (None, 32) | 2080 |
| dense_44 (Dense) | (None, 10) | 330 |

Total params: 596,906
Trainable params: 596,906
Non-trainable params: 0

Figure 10: BSLT Neural Network Summary table

It's also worth mentioning that the total number of parameters (596 906) is unimaginably low in comparison with any CNN network that trains a similar type of model, which proves the efficiency of LSTM networks over other of neural networks for this specific case and this is due to the nature of the LSTM networks itself.
LSTM networks are a special type of Recurrent Neural Networks (RNN) which are already extremely appealing as an alternative for this kind of project given that, unlike Artificial Neural Networks for example, they are able to connect previous information to the present task, such as using previous video frames to inform the understanding of the present frame and, potentially, recognise the sign. The issue however, is that RNNs cannot always do this; sometimes, the last information (or last video frame in this

case) is not the most relevant or not sufficient for recognising the sign but rather an information further in the past might be needed, this makes RNNs unable to connect information harming the learning process. This problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found fundamental reasons behind this learning difficulty in RNNs. Long Short Term Memory networks (LSTMs) however, were created to avoid the long-term dependency problem: remembering information for long periods of time is the specificity of their design. Therefore, in our case scenario where our dataset is composed of videos with many frames, RNNs might fall short given that only the frame preceding the one at hand is used for learning, which is not enough to accurately predict the sign. Figure 11 and 12 Represent a RNN a LSTM structure where X is the input h the output (hidden state).

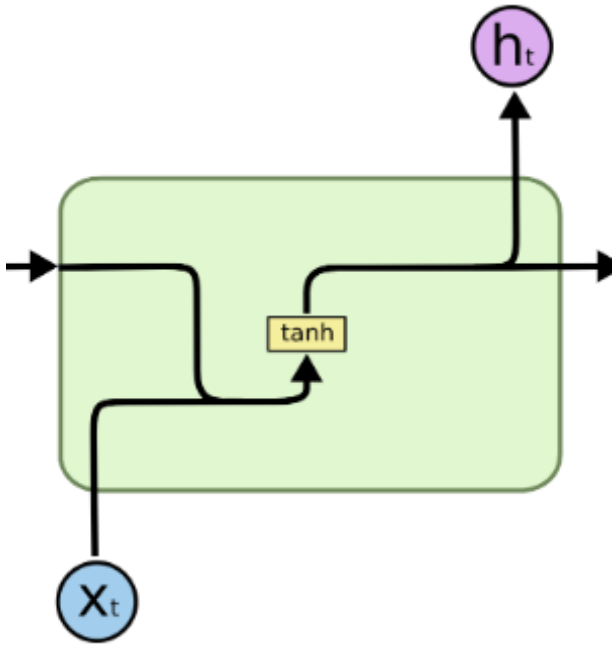


Figure 11: Standard RNN structure (Source : http://colah.github.io/posts/2015-08-Understanding-LSTMs)

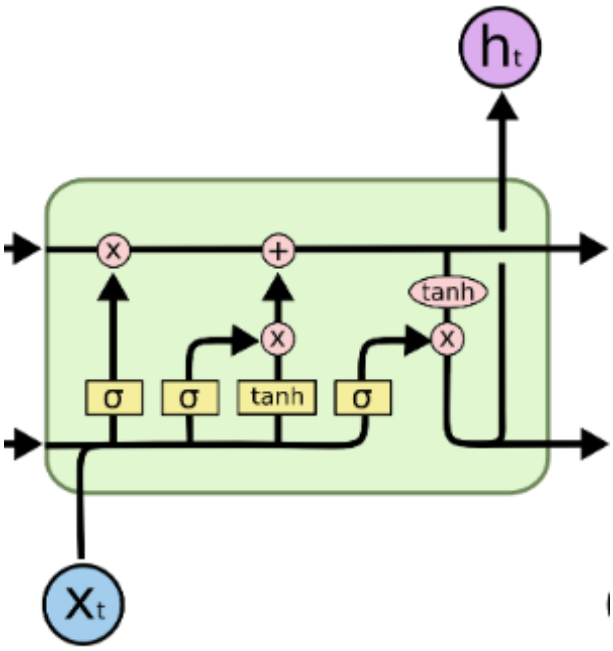


Figure 12: Standard LSTM structure (Source : http://colah.github.io/posts/2015-08-Understanding-LSTMs)

Notice that in the case of the RNN only one simple tahn layer is present unlike with the LSTM where there are many gates (a sigmoid neural net layer attached to a pointwise multiplication operation is referred to as a gate). These gates are what controls the flow of information and what offers LSTMs the advantage over RNNs (sigmoid can output 0 or 1, it can be used to forget or remember the information).
As displayed in figure 13, which is a simplified representation of the Neural network, the return sequences is set to true for the first 2 LSTM layers but not for the third one, resulting in producing the outputs h1 and h2. The reason behind this configuration is the fact that these outputs are used for the learning process inside as inputs for the following LSTM layers (remembering previous information and base the present one on it in LSTM structures). However, the last LSTM layer is connected to a Dense layer which has no use for the output information. Therefore, the return sequences has to be set to false.

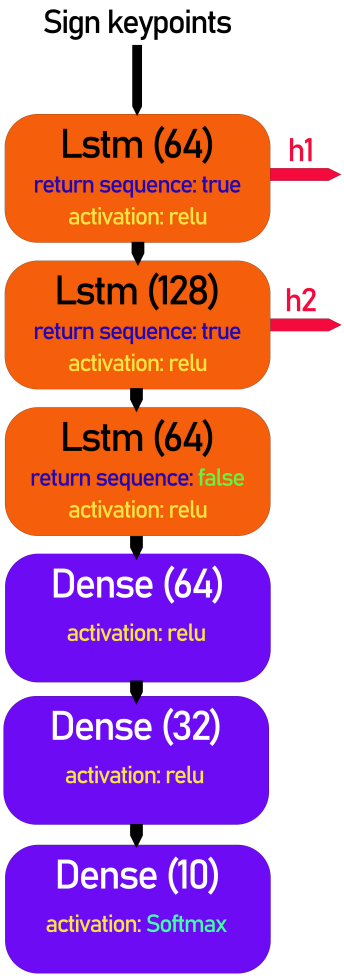


Figure 13: Simplified BSLT for medical Services Neural Network

Note that the Softmax activation is only used in the last layer to convert the output scores to a probabilistic value in order to facilitate signs recognition (picks the sign with the highest probability score). Figure 14 shows a simple example of softmax application for a small dataset of 3 signs.

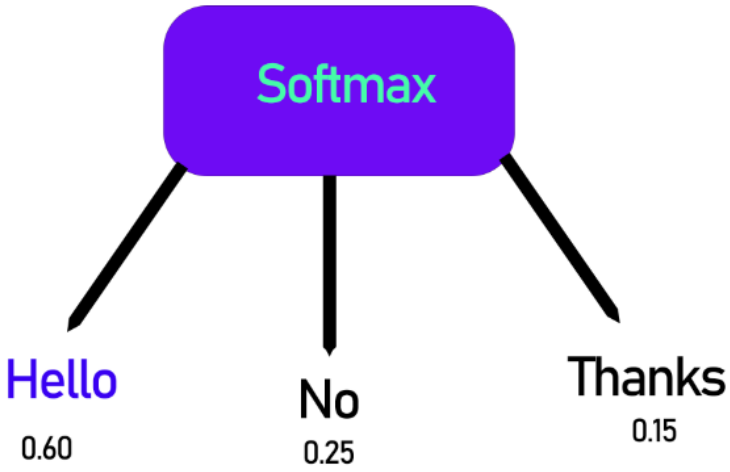


Figure 14: Softmax example

In figure 14, "Hello" has the highest score so its the word guessed by the model (note that the total of all the probability scores should add up to 1: 0.60+0.25+0.15=1).

### 4.3 Implementation

For the implementation, Jupyter (Julia, Python and R) notebook is the IDE adopted because of its numerous advantages when it comes to data science and machine learning projects specifically: Jupyter allows users to view the results of the code in-line without the dependency of other parts of the code (similar to the sections in matlab scripts), it is also able to cache the results of every cell that is running ( a code that is training a model ie), and the code can be edited by users and can also be sent for a re-run, making Jupyter's code non-static which allows users to control input sources for code and provide feedback directly on the browser. Another appealing advantages of Jupyter notebook in machine learning is the data visualisation: Jupyter supports rendering some of the data sets like graphics and charts, which are generated from codes with the help of some modules (Matplotlib for example). The programming language used is Python given that it is the most common programming language for machine learning thanks to the variety of useful modules and libraries it offers for this computing field. Speaking of libraries, the first step is importing the dependencies required for our

project: Tensorflow (version 2.4.1), openCV, "cv2" for computer visualisation (for the recording of the videos), "numpy" (for arrays and some basic functions), "matplotlib" (for graph representations) and the "Mediapipe" package (for keypoints extraction). Figure 15 displays the dependencie importation section of the code.

```
n [ ]: !pip install tensorflow==2.4.1 tensorflow-gpu==2.4.1 opencv-python mediapipe sklearn matplotlib

n [1]: import cv2
       import numpy as np
       import os
       from matplotlib import pyplot as plt
       import time
       import mediapipe as mp
```

Figure 15: Importing dependencies code

Next, we define functions to recognise the human keypoints (face, arms, hands and shoulders) and draw their landmarks on on live streaming:

```
In [2]: mp_holistic = mp.solutions.holistic # Holistic model
        mp_drawing = mp.solutions.drawing_utils # Drawing utilities

In [3]: def mediapipe_detection(image, model):
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
            image.flags.writeable = False                  # Image is no longer writeable
            results = model.process(image)                 # Make prediction
            image.flags.writeable = True                   # Image is now writeable
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
            return image, results

In [5]: def draw_landmarks(image, results):
            # Draw face connections
            mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION,
                                      mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                                      mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                                      )
            # Draw pose connections
            mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                                      mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                                      mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                                      )
            # 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 16: recognise human keypoints and draw landmarks (comments in green explain each line)

```
In [7]: cap = cv2.VideoCapture(0)
        # Set mediapipe model
        with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
            while cap.isOpened():

                # Read camera feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)
                print(results)

                # Draw landmarks
                draw_landmarks(image, results)

                # Show to screen
                cv2.imshow('OpenCV Feed', image)

                # Stops if s is pressed
                if cv2.waitKey(10) & 0xFF == ord('s'):
                    break
            cap.release()
            cv2.destroyAllWindows()
```

Figure 17: code to start the camera and apply the detection and drawing function to the live feed (comments in green explain each line)

Figure 18: Camera output)

Note that there's 1662 keypoints in total (points or circles covering the face,

shoulders, arms and hands).
The Next step is defining a function to extract these keypoints (for face, shoulders, arms and hands) and store them in an array once we start recording the different signs for our dataset, the extract$_k$*eypointsfunctionthattakesasaparameterthemediapi*

```
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if results.pos
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if results.face_landmarks else
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if results.left_hand_landma
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if results.right_hand_landm
    return np.concatenate([pose, face, lh, rh])
```

Figure 19: extractkeypoints function

Then a path is defined to store the extracted keypoints of each frame along with the array storing the signs to record and the number of frames and videos:

```
#Data for exported date, numpy array
DATA_PATH= os.path.join('Data_Collected')

# signs to Record
signs_emergency1 =np.array(['doctor','nurse','hospital','ambulance', 'heart attack','accident','weak', 'cough','tablet','it hurts

# 25 videos worht of data
no_sequences= 25

# videos are going to be 25 frames in length
sequence_length= 25
```

Figure 20: Path and signs

For each frame of each video of each sign, a folder is created (if it doesn't already exist) in the previously defined path to store the keypoints that are going to be collected during the recording:

```
for sign in signs:

    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, sign, str(sequence)))
        except:
            pass
```

Figure 21: Keypoints folder creation

The next step is the actual recording of the signs, to do so, we will loop through each sign in the signs Emergency array and through each frame (out of the 25 frames) of each video(out of the 25 videos) to record the

signs and create the dataset, we will be using the drawing landmarks function as well as the extract keypoints function to extract and store the keypoints while recording instead of going through some form of image processing after the videos are collected allowing us to save time and be more efficient. There is a wait time of 2 sec between the recording of each sign and the recording stops if 's' is pressed or if all the signs were recorded (traversed the whole sign array):

```
# set the camera to use for the recording
cap = cv2.VideoCapture(1)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through signs
    for sign in signs_emergency1:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(no_sequences):

                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)

                # Draw Landmarks
                draw_landmarks(image, results)

                # Create a starting point for recording
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                               cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(sign, sequence), (15,12),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    #wait 2 sec between the recording of each 2 signs
                    cv2.waitKey(2000)
                else: q
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(sign, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)

                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, sign, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                # Break
                if cv2.waitKey(10) & 0xFF == ord('s'):
                    break
    #closes recording window
    cap.release()
    cv2.destroyAllWindows()
```

Figure 22: Data collection for the training

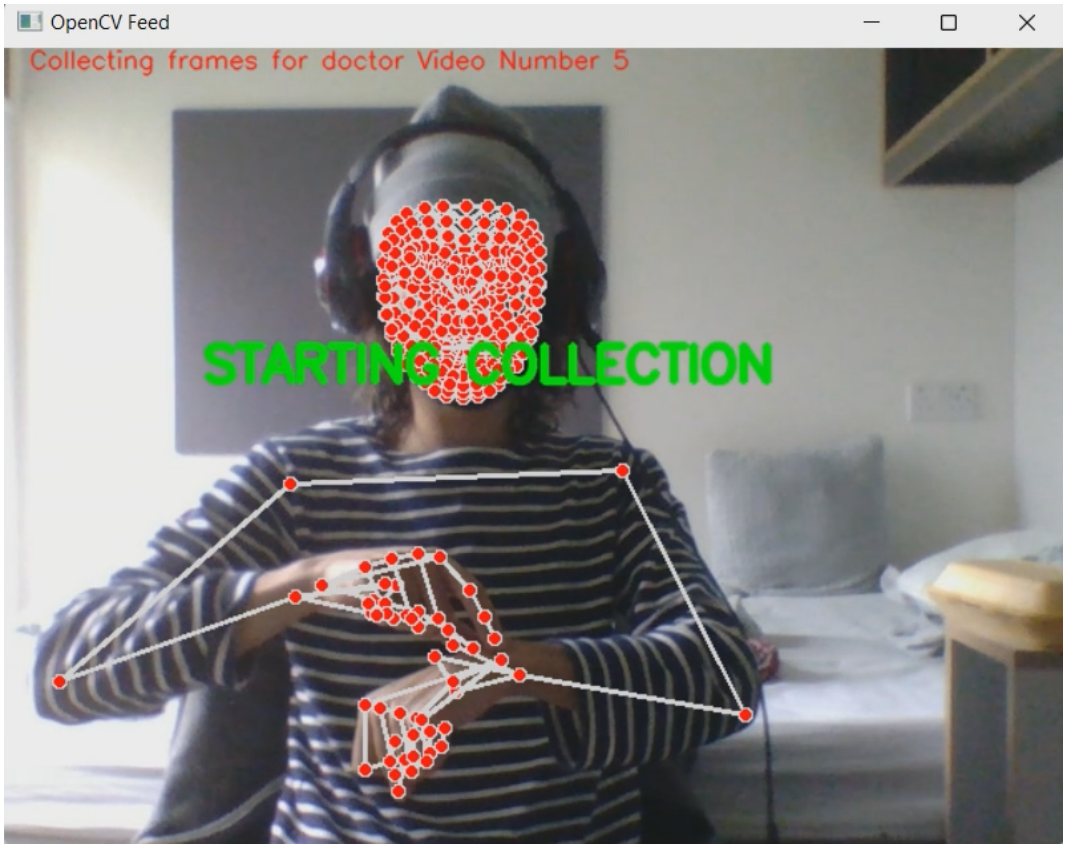


Figure 23: Camera output during the recording

Now that data is collected, we have to preprocess it by creating a label map and storing the frames in their folders according to the sign names. The dataset is then split into two sub-sets (training and testing) using the $\text{train}_t est_s plit function$ :

```
# importing method for splitting data for training and testing
from sklearn.model_selection import train_test_split

# importing method for visualising data during training
from tensorflow.keras.utils import to_categorical

# creating a labelling map for the data and adding each frame to its reserved folder
label_map = {label:num for num, label in enumerate(signs_emergency1)}
sequences, labels = [], []
for sign in signs_emergency1:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, sign))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, sign, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[sign])


#splitting data for the training and testing
X = np.array(sequences)
y = to_categorical(labels).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

Figure 24: Data preprocessing + train test splitting

The next, and most important, step is the creation of the Neural Network as defined by the structure described in the methodology section:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense



model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=(25,1662)))
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(signs_emergency1.shape[0], activation='softmax'))
```

Figure 25: Neural Network creation

For the training we will be using 'Adam' optimiser (default algorithm for stochastic gradient descent), a categorical crossentropy loss and we will start with a 1000 epochs:

```
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])
model.fit(X_train, y_train, epochs=2000, callbacks=[tb_callback])
```

Figure 26: Training parameters

After starting the training, we notice that at approximately the 800 epochs the categorical accuracy starts to stabilize around 1 (100 percent accuracy), we can stop the training of the model there:

```
8/8 [==============================] - 1s 113ms/step - loss: 0.0110 - categorical_accuracy: 1.0000
Epoch 770/2000
8/8 [==============================] - 1s 109ms/step - loss: 0.0099 - categorical_accuracy: 0.9935
Epoch 771/2000
8/8 [==============================] - 1s 113ms/step - loss: 0.0059 - categorical_accuracy: 1.0000
Epoch 772/2000
8/8 [==============================] - 1s 111ms/step - loss: 0.0074 - categorical_accuracy: 0.9964
Epoch 773/2000
8/8 [==============================] - 1s 118ms/step - loss: 0.0060 - categorical_accuracy: 1.0000
Epoch 774/2000
8/8 [==============================] - 1s 109ms/step - loss: 0.0043 - categorical_accuracy: 1.0000
Epoch 775/2000
8/8 [==============================] - 1s 110ms/step - loss: 0.0076 - categorical_accuracy: 0.9973
Epoch 776/2000
8/8 [==============================] - 1s 112ms/step - loss: 0.0078 - categorical_accuracy: 1.0000
```

Figure 27: categorical accuracy between epoch 770 and 776

After achieving satisfying training results, the model is saved :

```
model.save('signEm10.h5')
```

Figure 28: model Saving

Let's test the model next; the following code starts the camera and starts identifying the signs done by the signer (the student), figure 30 shows the output of this code when the sign hospital is done in front of the camera:

```
In [30]: # 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

cap = cv2.VideoCapture(1)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
        print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        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(signs_emergency1[np.argmax(res)])
            predictions.append(np.argmax(res))


        #3. Viz logic, display the words with higher probability
            if np.unique(predictions[-10:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:

                    if len(sentence) > 0:
                        if signs_emergency1[np.argmax(res)] != sentence[-1]:
                            sentence.append(signs_emergency1[np.argmax(res)])
                    else:
                        sentence.append(signs_emergency1[np.argmax(res)])

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

                # Viz probabilities
               # image = prob_viz(res, signs_simple1, image, colors)

        cv2.rectangle(image, (0,450), (640, 400), (255, 0, 255), -1)
        cv2.putText(image, ' '.join(sentence), (3,430),
        cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
```

Figure 29:code experimental testing of the model

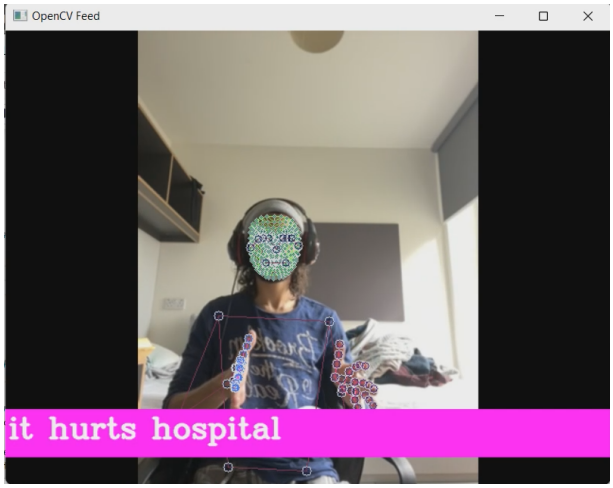


Figure 30:Hospital sign recognition

# 5 Testing

There are two types of approaches for testing: an experimental one where the user shows signs in front of the camera and evaluates the accuracy of the model or, a more methodical and precise approach that relies on a mathematical function, a dataset and which returns and tangible result. Given the advantage of precision and technicality (results can be generalised), we will opt for the second approach for this section by calculating the accuracy of this model for the sub-dataset we prepared. We have used earlier the train test split function offered by the Sklearn Library in python to split the original Dataset into two sub-datasets, one for testing and another one for training. The reason behind that being that all accuracy calculating algorithms work on a labelled dataset by using a function that compares the number of correct guesses made by the model with the total number of elements in the dataset, and to identify whether this guesses were correct or not in the first place, we need a processed, ready to use dataset. However, we can't test the training dataset; the accuracy will be biased. Hence the need for the train/test split method which, while being criticized by Jimin Tan, Jianan Yang, Sai Wu, Gang Chen, and Jake Zhao in their paper (*A critical look at the current train/test split in machine learning*), was still called " *the gold standard of machine learning*" in that same paper.
Now regarding the calculation of the accuracy, we will be using the function accuracy score offered by the SKLearn library in Python. This method calculates the accuracy of a model according to the following function:

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

$\hat{y}_i$ is the predicted value of the $i$-th sample and $y_i$ is the corresponding true value

In a simpler language, the sum of all correct guesses is divided by the total number of elements in the set (n), therefore, if all the elements are identified correctly by the model, the accuracy is one (100 percent) and in the opposite case, the accuracy is 0. After applying the accuracy score method to our testing dataset, we obtain the following result: 0.923 (92,3 percent).

```
accuracy_score(ytrue, yhat)
```

```
: 0.9230769230769231
```

Figure 29: model accuracy score where yhat is the model guessed value (sign identified by the model) and ytrue is the true value (the real sign)

# 6 Discussion and conclusion

The time invested for the development of the BSLT for medical services was mainly spent doing research, learning new technologies and experimenting with different approaches instead of developing the final product in itself(the model). This research has brought new perspectives to adopt for different socio-computational problems but has also proven that computer scientist may think of similar ways to tackle these problems as shown by the KSLT project.
As stated earlier, the **Korean Sign Language Translation Using Machine Learning model** has a similar approach to the one adopted for this project: relying on keypoints for sign language translation. However, there are some important differences, the obvious ones are the language (korean sign language in opposition to british sign language), the size of the dataset (larger for KSLT) and the use of the OpenPose Library by the KSLT while Mediapipe was utilized for BSLT. However, more subtle differences are also present: for the KSLT, keypoints are extracted later on after the recording is finished which means additional work, but in the case of BSLT, keypoints are extracted and stored automatically during the recording of the signs to increase efficiency. Moreover, while they both use recurrent networks, the overall structure of the networks are different.
Regarding the results, the accuracy of the BSLT for the sample of 10 signs is phenomenal (92.3 percent) considering the size of the dataset and the number of parameters of the Neural Network. This proves the efficiency of the LSTM approach for sign language translation, thus, allowing us to successfully achieve our goal of developing an operational Sign language Translator for medical services.Of course, improvement is still required in order to bring the model closer to a hundred percent for sign prediction either through some modifications to the neural network or by using larger datasets. Moreover more words should be added to the dataset in the future to allow for a more flexible and rich communication. These results are still,however, encouraging and prove that a powerful Sign Language Translator that would allow us to shatter the communication barriers is not out of reach.

# 7 Future Work

Thanks to these great results, LSTM networks seem to be a reliable tool for Sign Language Translation. The natural next step to take is, first, deploying this model in order for it to fulfill its purpose by being of use to somebody and also to receive feedback for areas of improvement from users. Then working on refining it by adding more data and words to the dataset but also by combining a more interesting concept with SLT: word prediction. If we can implement word prediction in SLT, we will be able to not only build models that translate isolated words but, complete phrases and even produce translation that sound natural. LSTM networks seem to be the best approach for this kind of problem given the advantage of their information remembering which will be vital for the development of Sign Language Word Predicting Translating models.

# 8 References

-Sang-Ki Ko , Chang Jo Kim, Hyedong Jung and Choongsang Cho, Korea Electronics Technology Institute, *Neural Sign Language Translation Based on Human Keypoint Estimation* PDF [online] Available at: https://www.mdpi.com/2076-3417/9/13/2683 [Accessed 10 April 2022]
-Rini Akmeliawatil, Melanie Po-Leen Ooi, Ye Chow Kuang *Real-Time Malaysian Sign Language Translation using Colour Segmentation and Neural Network* PDF [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=4258110 [Accessed 14 January 2022]
-Jan Bungeroth, Hermann Ney *Statistical Sign Language Translation* PDF [online] Available at: https://www.sign-lang.uni-hamburg.de/lrec/pub/04021.pdf [Accessed 23 December 2021]
-Nimesh Sinha *understanding lstm and its quick implementation in keras for sentiment analysis 2018* [online] Available at: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47 [Accessed 18 October 2021]
-FreeCodeCamp.org Channel *Deep Learning Crash Course for Beginners* Youtube [online] Available at:https://www.youtube.com/watch?v=VyWAvY2CF9c [Accessed 10 October 2021]
-CommendingHands Channel *Health and Medical Terminology in British Sign Language (BSL)* Youtube [online] Available at:https://www.youtube.com/watch?v=07KG48ADhO8 [Accessed 6 October 2021]
-Jimin Tan, Jianan Yang, Sai Wu, Gang Chen, Jake Zhao *A critical look at the current train/test split in machine learning* PDF [online] Available at: hhttps://arxiv.org/abs/2106.04525 [Accessed 28 December 2021]