Signs in Music

Lelio Casale (10582124), Alice Sironi (10699219), Cecilia Raho (10669712)

July 8, 2024

1 Introduction

1.1 Presentation of the project

When hearing people speak, they never think about the meaning of the words they are pronouncing and what they represent. Sign language, on the other hand, is a depiction of these words through the use of gestures and expressions directly associated with their meaning. We have created this project to develop an emotional bond between hearing and deaf people, through their way to expressing themselves. The site guides the user through an interface that recognizes gestures from the sign language using a neural network and searches for a song with a name related to the gesture with a query to the Spotify database using the **Spotify API for developers**. Then this song is played and some audio features are extracted and sent to the graphical part of the project, which is a particle system whose parameters depend on the audio features extracted from the song. This allows a deaf user to visualize the played song.

1.2 Structure

Our project has four different main blocks:

- An interactive site that explains the main purpose of the project and helps the user learn the American Sign Language (ASL).
- A **supervised trained neural network** whose goal is to recognize ten gestures of the sign language.
- A Python script that searches for a song whose title is related to the gesture done by the user using the **Spotify API for developers**. This section of the project also extracts from the **Spotify** database some of the audio features of the song, while the instantaneous energy is directly computed from the audio file of the song.
- A **particle system** that translates the song into a graphical interface. The parameters of the particle system are set based on the audio features extracted. For example, the **instantaneous energy** is mapped to the particles speed.

2 Implementation

2.1 Interactive Site

The site is divided into two pages: the introduction and the dictionary.
Professor Ludwig Von Drake, a magnificent know-it-all, is the character we chose to explain to users the purpose of *Signs in Music* and how it works (fig, 5).

This page is created using HTML and CSS, for the graphical interface, and JavaScript, in which the function to scroll the text by pressing the space bar is implemented. The last speech bubble present the *start* button, that links the introduction with the main part of the web site: the dictionary.

Figure 1: Web site introduction

By clicking on the dictionary cover, the book opens, and the user can find ten different signs, listed in an alphabetic order, with a brief explanation of how to correctly make the gesture and the corresponding image. The signs are: *Art, Baby, Cherry, Hello, I love you, Mommy, Music, Party, Sun* and *Thanks*. By pressing the *TRY NOW* button, the user is directed to the gesture recognition part.

Also the dictionary is implemented in HTML and CSS, to create the graphical part, and JavaScript, in which are implemented two functions: the first one allows the user to turn pages, going forward (clicking the right page of the book) and backward (clicking the left page of the book), the second one hides the character and the speech bubble when the dictionary is opened.

Figure 2: Dictionary

2.2 Gesture recognition

2.2.1 Dataset Creation

The first step involves creating a comprehensive dataset comprising various hand signs. Ten distinct actions are selected: *'art'*, *'baby'*, *'cherry'*, *'hello'*, *'iloveyou'*, *'mommy'*, *'music'*, *'party'*, *'sun'*, and *'thanks'*. For each action, 30 sequences of videos are recorded, each containing 40 frames.

To capture real-time video feed through a webcam, **MediaPipe's holistic model** is used to obtain accurate estimations of various body parts. This model employs state-of-the-art machine learning algorithms to identify and track 33 body landmarks, 21 hand landmarks, and 468 facial landmarks in real-time.

For each frame, keypoints representing the positions of the hand landmarks are extracted, which were further augmented with random rotations to enhance the dataset's diversity. These keypoints were then saved as *numpy arrays*, systematically organized into respective directories based on the action and sequence number.

2.2.2 Model Development

The next step is to develop a neural network model capable of recognizing these signs. The model architecture consists of a combination of *convolutional* and *LSTM layers* to effectively capture both spatial and temporal features.

The architecture begins with two *Conv1D layers* with *ReLU activation* and *L2 regularization*, followed by *MaxPooling* and *BatchNormalization layers* to reduce overfitting and improve generalization. A Dropout layer is also added for further regularization. The output of the convolutional layers is fed into an *LSTM layer* to capture the temporal dynamics of the sequences. Finally, *Dense layers* with *ReLU activation* and dropout are added before the output layer, which uses a *softmax activation function* to classify the input sequences into one of the ten action categories.

The model is compiled using the *Adam optimizer* and *categorical cross-entropy loss function*. Training is conducted with *early stopping* and *TensorBoard callbacks* for better monitoring and optimization. The final model achieve good performance metrics, including *high categorical accuracy* and a *balanced confusion matrix* across all classes.

The trained model is saved for future use in real-time sign recognition applications.

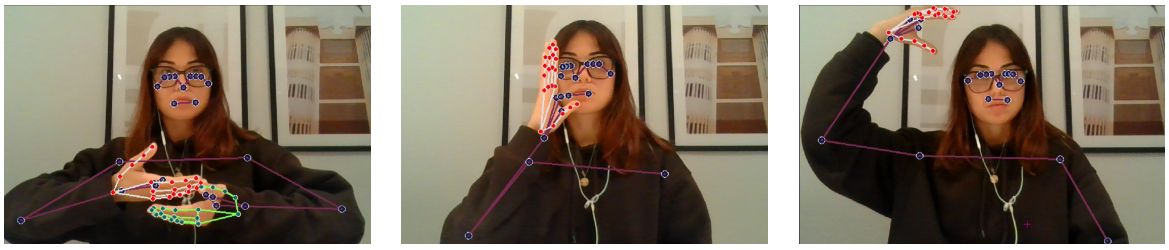
2.2.3 Real-time Analysis Visualization

In the final step of the gesture recognition, the trained neural network model is integrated into a real-time application to test its performance.

The application is developed using **Flask** to create a web interface, and **OpenCV** in conjunction with **MediaPipe** to capture and process video input from a webcam. **Flask** is a lightweight and flexible web framework for building web applications in Python. It is utilized to build a web-based interface that allows users to interact with the sign recognition system. The Flask application serves as the backbone of the system, handling incoming requests from users and providing appropriate responses.

The pre-trained model, loaded from a saved file, is utilized to predict the hand signs in real-time. As the user performs gestures in front of the camera, the application captures the video frames, detects hand landmarks, and extracts keypoints. These keypoints are then passed to the neural network model, which predicts the corresponding sign.

To enhance the user experience, the application includes features such as displaying the predicted sign on the screen, providing feedback for correct or incorrect signs, and even integrating with Spotify to play music based on recognized gestures.



"Baby" "Mommy" "Sun"

Figure 3: Landmarks recognized on the signs

2.3 Song search

The **Spotify API for developers** is a powerful tool that allows to search for a song given its name, its artist, the album of origin and eventually other parameters. In this case, the code written in *Python* uses the name of the detected gesture to search for a song that contains this name in its title. This

search is randomized over the ten most popular results, in order to obtain a different song given the same gesture, and all the song that do not have a *preview URL* are excluded. This is because from the *preview URL*, it's possible to download a 30 seconds preview of the song (not the entire song because of copyright reasons), that is then written to an audio file. **Spotify** allows also to download some precomputed features of the song, like the key and mode, the overall tempo, the average energy of the song, the average loudness, instrumentalness, speechness and much more.
Another part of the code computes the **instantaneous energy** of the song using windows of 512 samples with an hop length of 512 samples. Using a sampling frequency of 22050, this means that a window is computed every 23.22 ms. The audio features and the array with all the instantaneous energies are put together in a ***json* file** that acts as an intermediary between this part of the project written in *Python* and the particle system written in *HTML* and *JavaScript*. This part of the code then opens the *HTML* code that implements the particle system.

2.4 Particle system

The graphical part is realized through a **Particle System** that help to "visualize" the song selected from the *Python* code. Some parameters of the Particle System are fixed, like the particle shape, some are randomized, like the particle dimension and the particle opacity, while some others are selected using the song parameters. The Particle System is implemented using a lightweight external library written in *JavaScript* called *particles.js*. This library was then strongly modified in order to change the Particle System dynamically depending on the song parameters written in the *json* file.

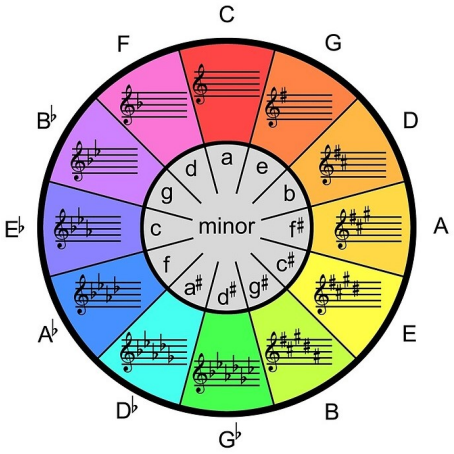


Figure 4: The Musical Color Wheel

In particular, the following parameters control the Particle System graphic:

- The **average energy** of the song controls the number of particles in the screen. A higher energy means more particles in the screen.
- The **average loudness** of the song controls the width of the borders of the particles and the width of the lines that connect the particles. A higher loudness results in a higher width of these two parameters.
- The **key** and the **mode** are used to setup the colors of the Particle System. Each couple *key, mode* is mapped to a different **background color** and **color of the lines** between the particles. This mapping is done following the **Musical Color Wheel** (fig. 4).
- The **instantaneous energy** controls frame by frame the particles speed, which is updated every 23.22ms. The result is that the particles move synchronously with the played song.

The **Particle System** starts with a static screen, with all the particles not moving. The movement and the song can be started using a button in the top-right corner: by clicking it, the particles start moving with a speed that depends on the instantaneous energy of the song.

Figure 5: Particle System graphic

2.5 Additive Synthesizer

In addition to the playing song, an **additive synthesizer** was implemented in *javascript* to add a modifiable pad sound to the selected song. The synthesizer has many parameters that can be modified by the user through the **Graphical User Interface** on the particle system:

- The single gains of the four base oscillators implemented, which are a **triangle** waveform, a **square** waveform, a **sawtooth** waveform and a **wavetable-based** waveform implemented to be the most suitable for a pad-like sound. Each oscillator is used three times at different frequencies in order to play a chord which is based on the key and mode of the selected song (e.g. if the song is in C major, the oscillators will play the C major chord, composed by C, E and G)
- The **gain of the song**, that can be modified in order to obtain a louder volume for the song, or even to mute it if the user wants to hear only the pad.
- The **cutoff frequency** of the two **Biquad filters** implemented, an high-pass and a low-pass.
- The **octave** of the first note of the chord: in this way the user can play a higher or lower chord based on his preferences and on the song needs.

To the output gain there is also a **Low Frequency Oscillator** applied, whose frequency is a fraction of the song bpm. In this way, the amplitude oscillation of the pad sound will be synced with the song.

3 Conclusions and Future Improvements

Signs in Music is created to explain the American Sign Language (ASL) to hearing people, in order to create an emotional bond with deaf people, with the aid of audio and graphic to generate an immersive experience. We have created a site with some features, useful to learn the sign, replicate it in the gesture recognition to produce the graphical interface, but there are some useful ideas for possible future improvements:

1. We can insert a new section in the web site, where the user is free to perform all the sign learned, in the order and as many times as he wants.
2. The gesture recognition system works only with ten signs (written in section 2.1). The idea is to expand the dictionary by inserting new signs of the ASL.

3. According to the previous point, if we expand the dictionary, we must also improve the gesture recognition system, in order to map body movements, as well as hand movements and facial expressions. This will make the recognition system more effective in distinguishing similar signs from each other.

4. As mentioned in section 2.4, the particles shape is fixed to a circle. Another idea for possible improvements could be to give to the particles the shape of an emoji related to the done gesture.

5. The system can be integrated with more applications, such as virtual assistants and educational tools, to increase its utility and accessibility for promoting inclusivity and language learning.