Universitat Politècnica de Catalunya

Master in Artificial Intelligence

Assistive And Health-Care Technologies

Final AHCT Project Document

Team 2

Authors:
Raquel Pérez
Gaspard Debussche
Jorge Sierra

January 15, 2020

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Contents

1 Introduction

According to the World Health Organization (WHO), 466 million people across the world have disabling hearing loss and statistics show that this figure will rise to 900 million in 2050. It has been noted that the majority of people with hearing or speech impairments are located in low-income parts of the world and therefore in countries where the infrastructure to enable them to integrate well has not been put in place.

Wherever communities of deaf people exist, sign languages have developed as handy means of communication. Hands gesture even accompany normal conversation as part of body language (however this should not be confused with sign language).

Although signing is used primarily by the deaf and hard of hearing, it is also used by hearing individuals, such as those unable to physically speak, those who have trouble with spoken language due to a disability or condition (augmentative and alternative communication), or those with deaf family members, such as children of deaf adults.

Linguists consider both spoken and signed communication to be types of natural language, sign languages are expressed through manual articulations in combination with non-manual elements. Sign languages are full-fledged natural languages with their own grammar and lexicon

It's important to note that **sign languages are not universal and they are not mutually intelligible with each other**. For this reason the project will be based on the most widely used American Sign Language (ASL).

2 Solution proposal

Although their lives may seem very quiet, deaf people have no different mental state from common individuals. More and more organizations and educational institutions are trying to include them in the society without any mayor set back due to communication problems. In order to achieve this, institutions need to equip themselves with tools that are accessible and easy to use in order to allow fluid communication.

In order to meet the needs of the deaf and muted or their entourage, we want to offer a solution that makes it easier to interact with people. The idea of our solution is to respond to the problem of learning sign language. Indeed, very often deaf and dumb people are not able to express themselves in the common language of their surroundings and their wider environment does not know sign language. We therefore thought of developing a mobile application that solves the problem of this language barrier: SignReco (figure 1).

Figure 1: SignReco Logo

Our goal is to develop an application that will be able to translate from text to sign language and vice versa using the phone camera. It is also needed to add the standards text-to-speech and speech-to-text functionality in order to make the application more ergonomic.

The project will be centered around **ASL spelling to text** (also called fingerspelling), excluding ASL words to text translation which includes all possible words that may be created with just one gesture (cf. Figure 2), this would increase the complexity beyond the current state of the art levels. However, a lot of research has been made in the field of automatic translation of sign language video into text (e.g. [4] [3]) but not even a handful of alternatives exists as a mobile application.

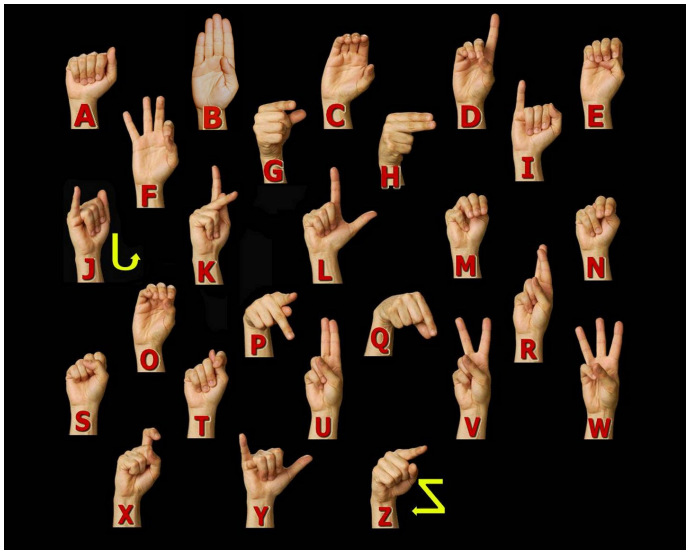


Figure 2: ASL fingerspelling.

The variability of the circumstances in which the real time video may be taken may affect the robustness of the system, however with only a small fixed value of possible hand positions to represent spelling, an intelligent system to recognize them shall be found.

3 Schedule

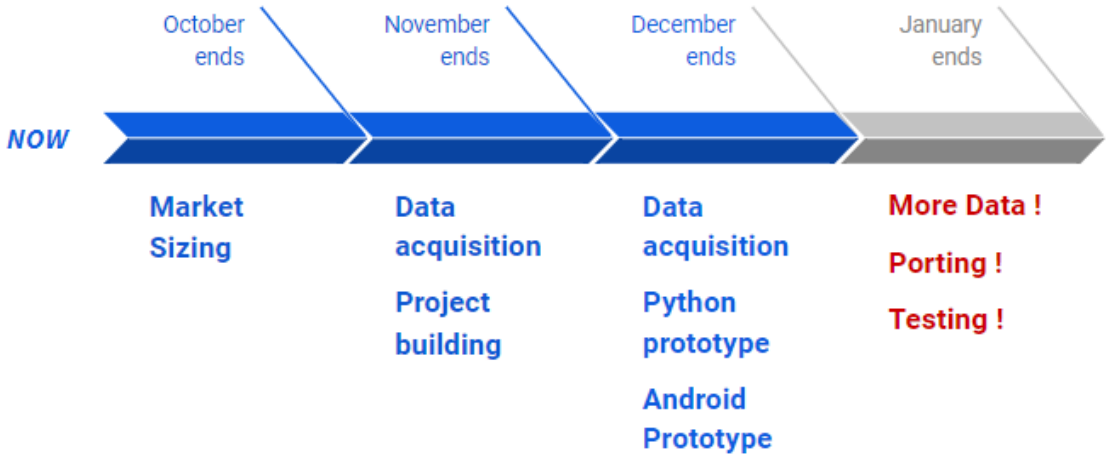


Figure 3: Original schedule.

Our schedule foresees the release of a prototype at the end of January with four steps of development. First of all, the market study that will allow us to start the product design in October. At the same time it is planned to start building our models and train them until November. In December, the different modules will be assembled: model and interface; at this point, a Python and Android prototype are

finished as a first version.

The final step in January includes testing with beta users, however due to different setbacks, more data acquisition is recommended and porting the Python code from the prototype to Java is work in progress. This step will allow us to get feedback and improve the application. We plan to apply the Agile methodology to obtain a final product that meets the needs of users. That is why we will probably do a new test phase with the improved application.

4 Mockup

Here is an idea of the final application. The application has three parts: speech to sign language, text to sign language, sign language to text.

Figure 4: Home screen, Menu and Sign language to text.

Figure 5: Speech and Text to Sign laguage.

5 Global Architecture

We are build a pretty common architecture with 3 layers (Figure 6).

- **Backend**
  Coded in Java (Kotlin) for Adnroid links all the modules together and execute other secondary functions.
- **User Interface**
  The graphical interface will also be coded in Java (Kotlin) for Android. It contains all the direct user interaction and graphical design content.
- **Modules and database**
  The database coded in SQL will store all the data needed by the app. In this database there will be all the images for the part of the application that concerns the courses to learn to speak sign

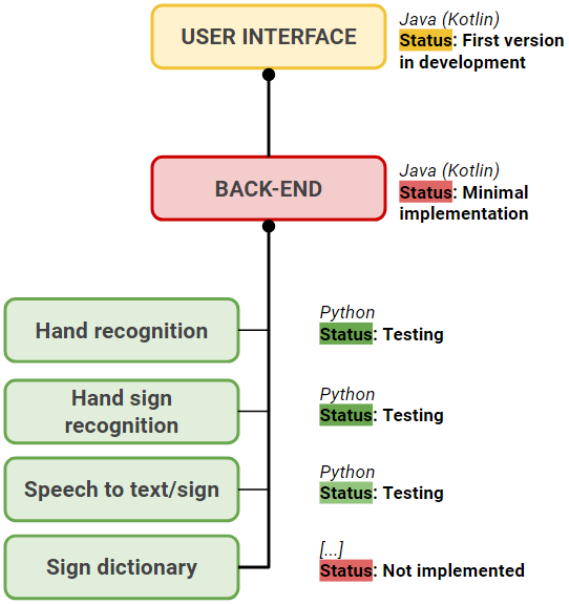


Figure 6: Global Architecture.

languages. Four different modules are used: hand recognition, hand sign recognition, speech to text/sign and the sign dictionary. All modules are explained in the following sections; each one represents a model or a group of models destinated to fulfill the same task.

5.1 Speech-to-text/sign module

It seemed obvious to us to develop a module that allows the user to talk to the application and have his words automatically translated into sign language.

This module was developed in Python. It uses the SpeechRecognition library as a speech-to-text model where the transcription into sign is used later. The tool, developed by the community can be used around the google wrapper which is the most powerful free tool. We will then be able to use a more powerful and paying service depending on our needs.

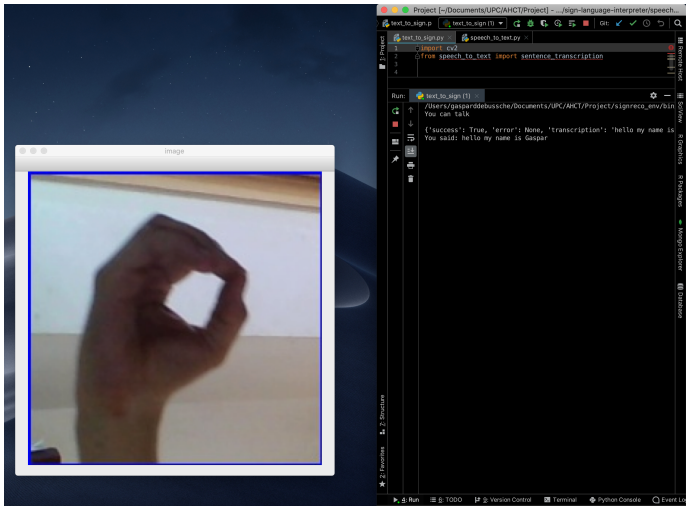


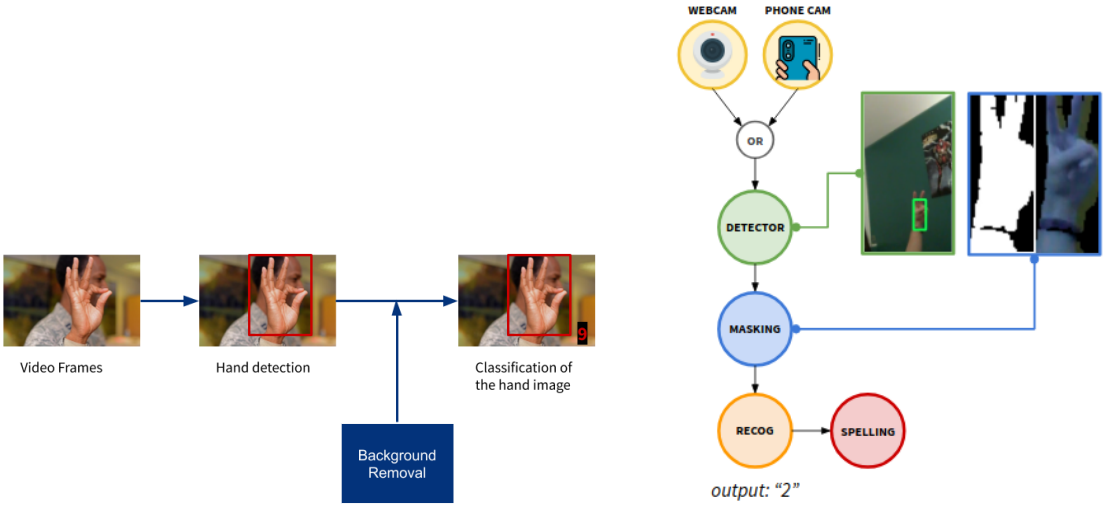
Figure 7: Speech-to-sign in action

The current tool can be launched by running the textbftext-to-sign.py program. Once the text "You can talk" is displayed, you can say a sentence. A Python OpenCV window will open and spell out all the letters of the sentence including spaces.

You can see the current result on the Figure 7.

5.2 Hand recognition

The next two sections cover the computer vision part of the project. During the development of the project, we designed two different pipelines for the computer vision part. Figure 8a shows the first pipeline and Figure 8b the final one. At first a simple enough model was expected to work given enough data, but as the experience dictates, after a handful of tries a more robust pipeline was needed to recognize images in more sophisticated ways. The final pipeline has three main parts; Hand detection, hand masking and sign recognition.



(a) Original pipeline planed.

(b) Final pipeline implemented.

Figure 8: Comparison between the planed pipeline and the one implemented at the end.

In the first part, the app gets an image, coming either from a webcam (using a desktop implementation, for testing and debugging purposes) or from the phone camera. Then, a pre-trained SSD network model [5] for hand detection looks for all the hands present on it in real time. In a previous project version our own CNN model was tested and trained for hand detection using the COCO dataset [2] (400 hand images where extracted and manually segmented from the dataset) but the data quantity and training time required for this first step with really no accuracy improvement over the pre-trained SSD network so it was directly used at the end.

The SSD network outputs a big number of possible bounding boxes that may or may not contain hands with a given confidence score. Some steps are followed here in order to maintain temporal, color and spatial information coherent while selecting which of these boxes correspond to a hand:

1. Discard boxes with abnormal (too big or too small) normalized area
2. If no boxes are found, pick last found box within time
3. Thresholding score for best boxes
4. Pick box above threshold with best manhattan distance to the last known box
5. If that manhattan distance is too big, try again but with all boxes (don't exclude by score threshold)

6. Execute a KMeans on the box region to get a list of the predominant colors and check whether those are representative of a hand
7. Assert the average scores of the last N boxes are bigger than a threshold

At this point the bounding box or *roi* of the hand has been selected. Now all it is needed is to mask the hand in that region so it can be later classified. This way the hands are easier to classify, as they have less noise and are more similar to the dataset used to train the classifier. The steps following in masking are described as follow (in figure 8b an example of masking is shown):

1. Expand *roi* by factor k ($k = 150\%$)
2. Transform *roi* color space to HSV, mask colors in range in a binary image:
   - H: 5°, S: 19%, V: 19% (Red, lighter)
   - H: 50°, S: 58%, V: 58% (Yellow, darker)
3. Morpholohy operations
   - Open (Erode + Dilate)
   - Dilate
4. Calculate all contours for each segmentated area
5. From all the contours, pick only the one with the closest manhattan distance to the center of the *roi*.
6. Fill completely that contour

Finally, the hand images without background are passed to a model, that classifies them into the sign that they contain.

5.3 Hand sign classifier

No pre-trained models were found for the hand sign classifier, for this reason we trained a classifier from scratch. This includes, searching for a proper dataset and train different architectures until one that adapts well to the particularities of the data is found.

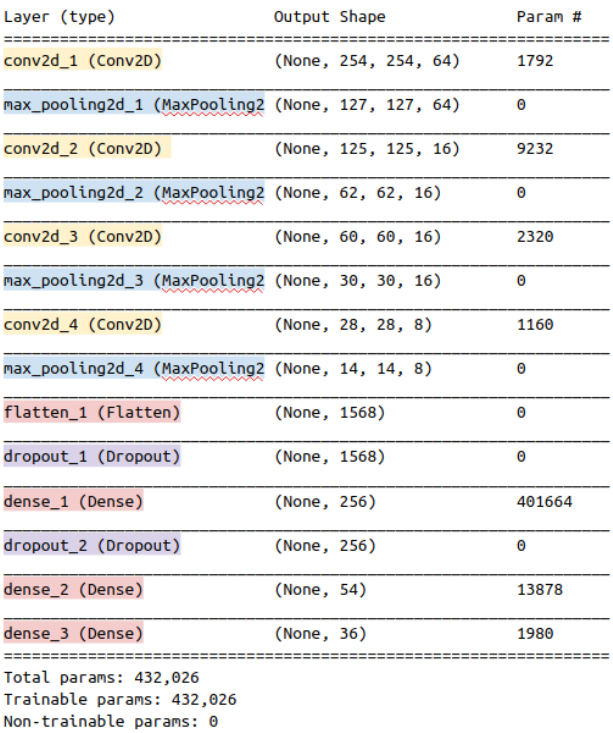
5.3.1 Architecture

We tried different architectures. The one that showed the best performance is medium sized (around 400k parameters) with 4 convolutional layers, all followed by max-pooling, three dense layers and two 0.2-dropout layers. We decided to add a bit of dropout to avoid doing over-fitting, as the dataset is quite small. Figure 9 shows a representation of the architecture.

5.3.2 Dataset

Originally, we planned to use the sign-language MNIST to train our model, but when we started to work with it, we discovered that this dataset was overly processed. The images contained on it where too small, without many variability and stored in a CSV instead of raw images. All these characteristics made it useless to us, as we planned to apply the model to real-world images taken from a phone. So we needed to find a better dataset. Figure 11 shows some of the sign language MNIST samples.

The chosen dataset is the one provided by [1] from Massey University. It contains a total of 2589 images, from 36 different classes. The classes correspond to the American Sign Language alphabet and the 10 numbers. Figure 13 shows an example of the images in the dataset. This dataset contains images from different hands, different illuminations and different sizes. Having variability in the training dataset will help the model work with the real images. We decided to split the dataset into 2,123 samples for training, 258 for validation and 208 for testing.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 254, 254, 64)	1792
max_pooling2d_1 (MaxPooling2	(None, 127, 127, 64)	0
conv2d_2 (Conv2D)	(None, 125, 125, 16)	9232
max_pooling2d_2 (MaxPooling2	(None, 62, 62, 16)	0
conv2d_3 (Conv2D)	(None, 60, 60, 16)	2320
max_pooling2d_3 (MaxPooling2	(None, 30, 30, 16)	0
conv2d_4 (Conv2D)	(None, 28, 28, 8)	1160
max_pooling2d_4 (MaxPooling2	(None, 14, 14, 8)	0
flatten_1 (Flatten)	(None, 1568)	0
dropout_1 (Dropout)	(None, 1568)	0
dense_1 (Dense)	(None, 256)	401664
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 54)	13878
dense_3 (Dense)	(None, 36)	1980

Total params: 432,026
Trainable params: 432,026
Non-trainable params: 0

Figure 9: Visual representation of the architecture chosen.

Notice that this task has some characteristic limitations that might affect the results. First of all, some sign letters are very similar. We can see on Figure 12 some examples of letters that could be mistaken even by a human. The second main difficulty is that some sign letters contain movement, and we are loosing that information by classifying static images. Finally, we are only classifying letters from sign language, but usually people communicates using signs that represent whole words. This could be a big limitation in real usages.

5.3.3 Results

The results obtained by classifying the test set (containing around an 8% of the total dataset) are presented on Table 1 and Figure 14. We obtained around an 0.82 of accuracy, and a 0.79 of f1-score. These are quite good results for a dataset with 32 classes. In Figure 14 we can see the confusion matrix of the test images. Most of the classes where correctly identified, but some of them where completely confused by the model. To try to understand why this happened we observed the images of given classes. We found out that the miss-classified samples where very similar to the class where the model classified them. Figure 12 shows some examples; the model confused the R and the U, the M and the N and the V and the K. As we can see on the figure these letters are very similar, so even a human could confuse them sometimes.

	Precision	Recall	F1-score
Macro avg.	0.82	0.81	0.79
Weighted avg.	0.82	0.81	0.79

Table 1: Results.

6 Prototype viability

It's important to test whether the prototype that was made is capable of running all the inference models in real time. For a very early version test without porting the Python code to Java (Kotlin),

Figure 10: Example of the images present on the sign language MNIST dataset.

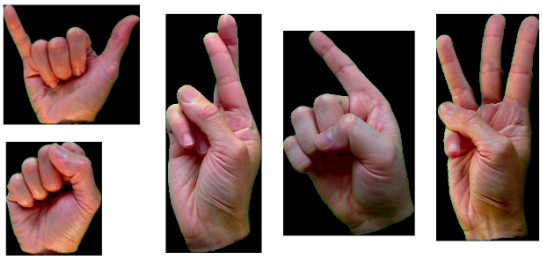


Figure 11: Example of the images present on the Massey University dataset.

the code was run in a single CPU core at $3.40GHz$ (no multithreading, no multiprocessing, no GPU) with the following results (a modern phone is expected to fulfill these requirements) using no more than 100MB of RAM:

Task	Speed (ms)
Frame reading (Webcam)	25
Pre-processing	$<$0
SSD Detector	30
Post-processing	10
Classifier	10
Speller (expected)	$<$5
Total processing	55 (18 FPS)
Total	85 (11 FPS)

Table 2: Processing speed.

7 Difficulties Found

Is common that when you do a project with a bit of complexity and size difficulties arise. We found some during the development of the project. During the planning phase of the project we predict some difficulties and thought about contingency plans for them, but we also found some unexpected problems. In this section we want to explain them and how did we overcome them.

We planned the project as a set of independent parts, with the objective of merge them at the end. This way it would be easier to work in them in parallel, but one risk of working this way is that you cannot test the whole program until all the parts are finished and integrated. Also, when integrating code from different programming languages you can have compatibility problems. At the end, the integration was easier than expected, but in the case it would not, we would have trained again the model in a java-based language, as the training process was quite fast, and once you know the proper

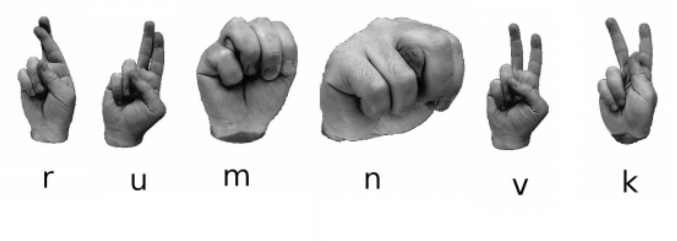


Figure 12: Example of some sign letters that are very similar between them.

architecture it is easy to implement the model again in another language.

With respect to the testing, we expected that the accuracy of the CNN would drop with the real images, and we planned some measures to avoid it (like fine tune the model with some real-life images or improve the background removal). The main problem in this case was that we got out of time to properly test, so we could not fix this problem.

Another difficulty that we found was implementing the *voice to text* part. As our main programming languages for the project (Java and Kotlin) were thought for doing Android apps, we thought it would be easy to find a library already implemented with this functionality. At the end this did not work properly, so we had to do this part on python. Contrary to the CNN (there are tools to read Keras models from android code), we had problems integrating this code in the whole project.

Last but not least, we had time-coordination problems to work on the project, as the three of us had other responsibilities a part from the master. This was a strong limitation for us, so we had to limit the functionalities of the final delivery.

8 Future Work

The current state of the application does not allow us to put it into production. Before reaching this stage, we will first have to group together the various modules we have created (speech-to-text-to-sign, sign-to-text), develop the graphical interface, and deposit the application on the distribution platforms. This will probably involve migrating our algorithms developed in Python to Java.

As we said before, our application currently only works with the letters of the alphabet. Of course, in order for the application to be practical and allow a fluid and fast language, we will have to integrate sign language for words. This can be done by applying the same models and algorithms as before and adapting them to the new images. For this reason, new datasets will have to be found that group together images with the signs corresponding to the words in the vocabulary of the English language.

It will also be necessary to integrate different languages so that the application can be used in different countries. According to the list of the most spoken languages in the world, we will have to develop the application for the following languages: Chinese, Spanish, Hindi, Arabic, Portuguese, Bengali and Russian. It should be noted that there are a multitude of sign languages throughout the world. It will therefore be necessary to be able to collect datasets for these different languages and their associated sign languages.

9 Software

All the software can be found in https://github.com/RaquelLeandra/sign-language-interpreter

References

[1] ALC Barczak, NH Reyes, M Abastillas, A Piccio, and T Susnjak. A new 2d static hand gesture colour image dataset for asl gestures. 2011.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. pages 740–755, 2014.

[3] Subha Rajam and Balakrishnan Ganesan. Real time indian sign language recognition system to aid deaf-dumb people. *International Conference on Communication Technology Proceedings, ICCT*, 09 2011.

[4] V. N. T. Truong, C. Yang, and Q. Tran. A translator for american sign language to text and speech. pages 1–2, Oct 2016.

[5] Dibia Victor. Handtrack: A library for prototyping real-time hand trackinginterfaces using convolutional neural networks. *GitHub repository*, 2017.

A Extra images

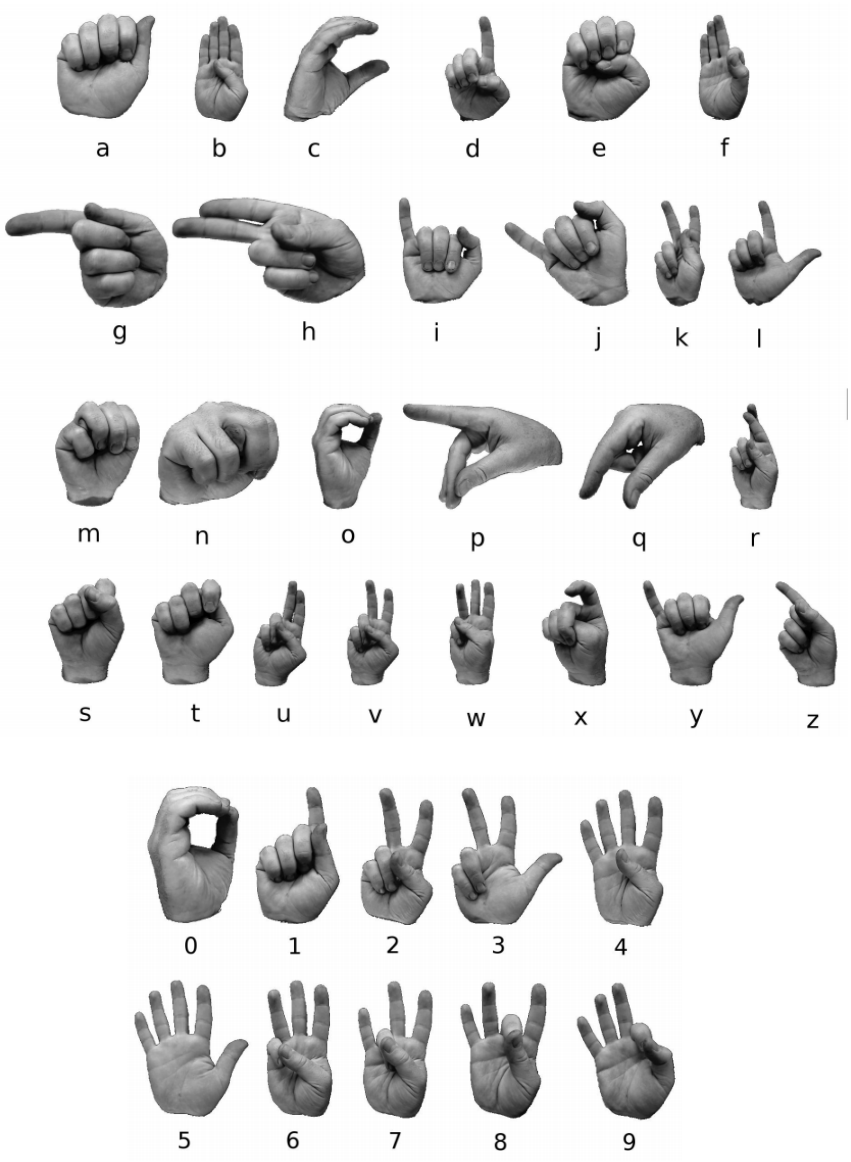


Figure 13: Example of the classes in the dataset.

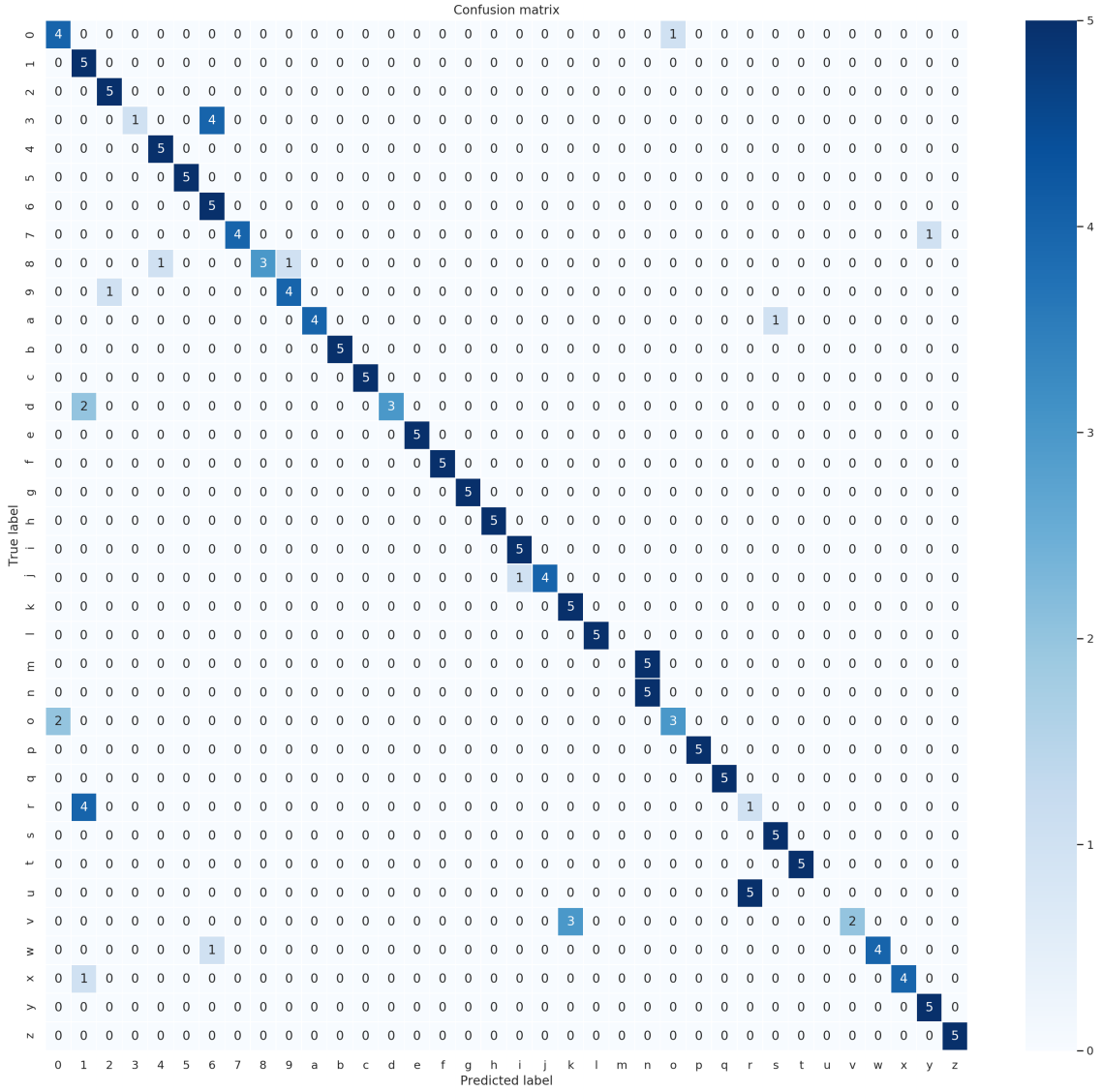


Figure 14: Original schedule.