1. Introduction

1.1. Background

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a type of security measure known as challenge-response authentication. CAPTCHA helps protect you from spam and password decryption by asking you to complete a simple test that proves you are human and not a computer trying to break into a password-protected account.

A CAPTCHA test is made up of two simple parts: a randomly generated sequence of letters and/or numbers that appear as a distorted image, and a text box. To pass a test and prove your human identity, you must type the characters you see in the image into the text box. This could not be the case for some human beings with special needs. As a result, nowadays, the need for Multimodality systems is increasing to support people with special needs as well as increase the system flexibilities.

Multimodal interfaces are a class of intelligent multimedia systems that make use of multiple and natural means of communication (modalities), such as speech, handwriting, gestures, and gaze, to support human-machine interaction. More specifically, the term modality describes human perception as one of the three following perception channels: visual, auditive, and tactile. Multimodality qualifies interactions that comprise more than one modality either the input (from the human to the machine)or the output (from the machine to the human) and the use of more than one device on either side (e.g., microphone, camera, display, keyboard, mouse, pen, trackball, data glove). Some of the technologies used for implementing multimodal interaction come from speech processing and computer vision; for example, speech recognition, gaze tracking, recognition of facial expressions and gestures, perception of sounds for localization purposes, lip movement analysis (to improve speech recognition), and integration of speech and gesture information. In 1980, the put-that-there system (Bolt, 1980)was developed at the Massachusetts Institute of technology and was one of the first multimodal systems. In this system, users simultaneously could speak and point at a large-screen graphics display surface in order to manipulate simple shapes. In the1990s, multimodal interfaces started to depart from the rather simple speech-and-point paradigm to integrate more powerful modalities such as pen gestures and handwriting input (Vo, 1996) or haptic output. Currently, multimodal interfaces have started to understand 3D hand gestures, body postures, and facial expressions (Ko, 2003), thanks to recent progress in computer vision techniques.

1.2. Motivation & Goal

Our goal in this project is to have an application that is able to simulate the CAPTCHA functionalities with the use of multichannel interactions by considering different channels of the interaction in order to detect whether the system is communicating with a human or machine.

It is divided into **two** main sections that are somehow two levels of machine detection in which if you will not be able to pass the first challenge you would not be able to access the next challenge since you have been detected as a machine.
the first level is done automatically by the system but the second level needs a different type of interaction from the user/machine.

1. At the first level, we are trying to detect the possible spoofing (the machine is trying to log in with an image of the target user) the first step to preventing having access to the system. Actually, we are trying to support somehow the reliability of the authentication system. Suppose that a machine is able to access the true credential information of the genuine user, then once it is requested to communicate with our system, It must be able to present a real face of the genuine user, otherwise (presenting a picture), it would be detected as a machine that could not be able to access the system even with true credentials.
2. Once the user has been detected as a real user and not a machine, he/ she has been forced to do some challenges which have been generated completely randomly by the system. These challenges are combined with some human perceptions such as Gestures, Head poses, and Emotions in order to further detect a possible attack on the system. Suppose that a machine would be able to pass from the first section successfully, then we need to further stress the detection by the second (harder) level of challenges!

Since the application should be able to support different people and also the people with special needs we decided to provide multi-channel in the system. We considered several capabilities for the system to support different kinds of disabilities. to mention some:

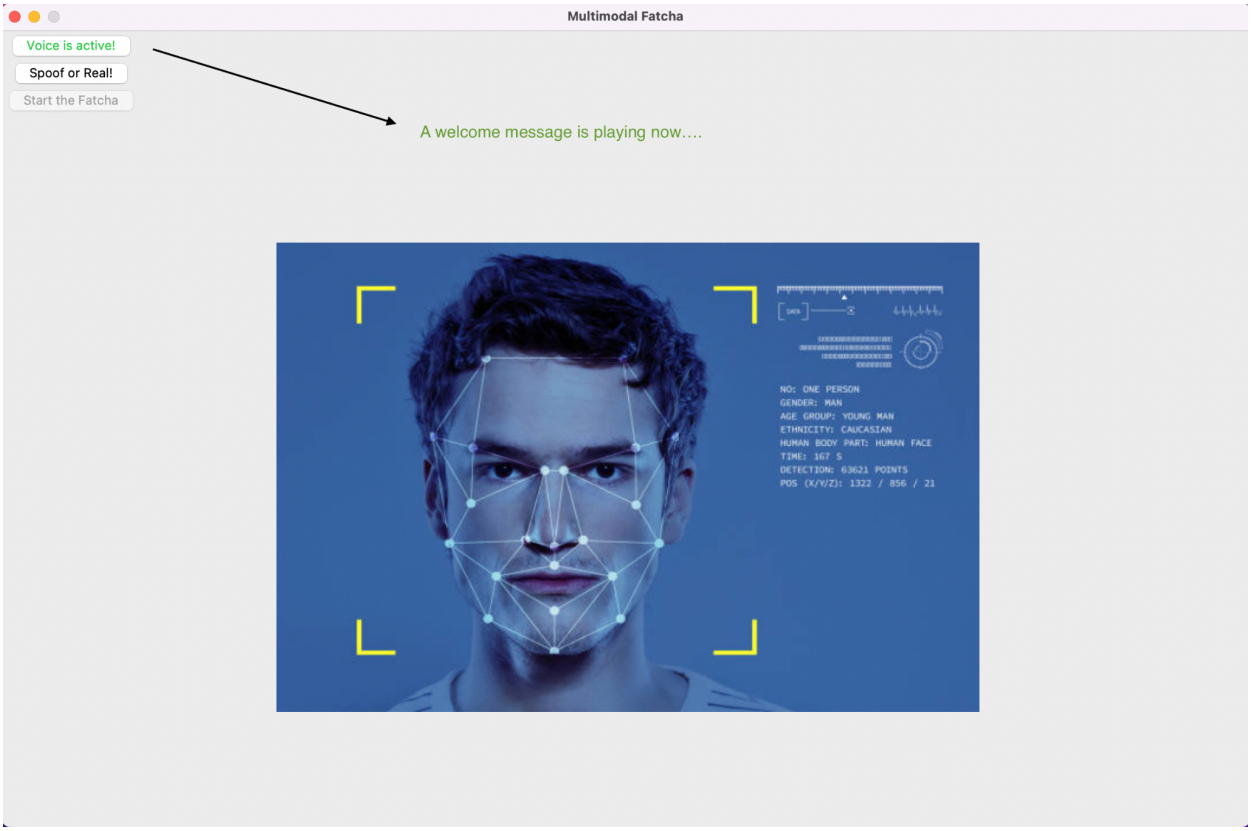
- the channel of voice interaction which can be switched on or off during the process.
- the textual information all over the system to provide the state of the system
- visual icons for specific kinds of gesture interaction (for people that unable to read a text)
- and many more.

1.3. System's Requirements

Since the application is based on a **Multimodal Interaction** (**3** different channels voice, visual, and gesture), the user should be able to meet some requirements such as a microphone, camera, display, keyboard, mouse, and speaker.

Also, all the codes and modules have been implemented by the latest version of python(3.10.5)
It has to be mentioned that, as we have trained the high accuracy models, the application is not sensitive to illumination problems.

1.4. General View of the interface



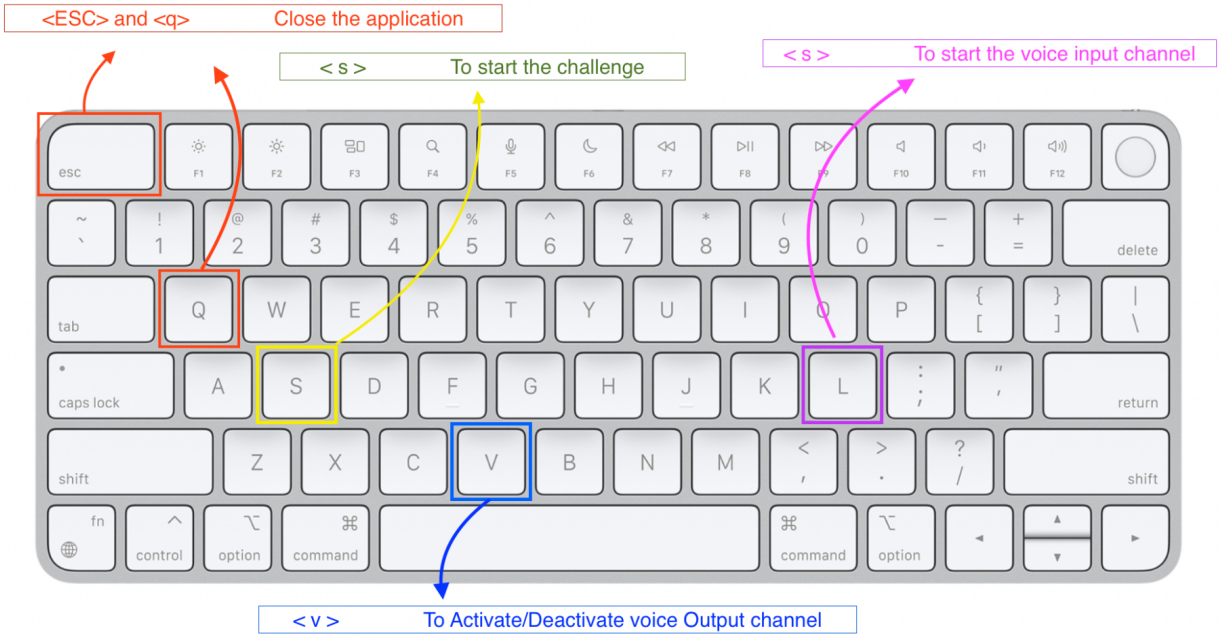
One thing to consider is that we are not trying to authenticate the user, but we are trying to detect whether the communication is done with a robot or a human being.

1.5. Multimodality

As mentioned we are going to use multiple and natural means of communication (modalities), in order to interact with machines or users. We use speech, gestures, and graphics interaction in order to support human-machine interaction. The term modality describes human perception as one of the three channels: visual, auditive, and tactile.
We have taken two of these channels in order to proceed with the communication. As mentioned in the introduction Multimodality qualifies interactions with more than one modality either the input or the output. We know that Mulimodality entails the use of more than one device on either side.
We are going to briefly talk about the channels and devices that we have used to perform the interaction.

1.5.1 Input Devices

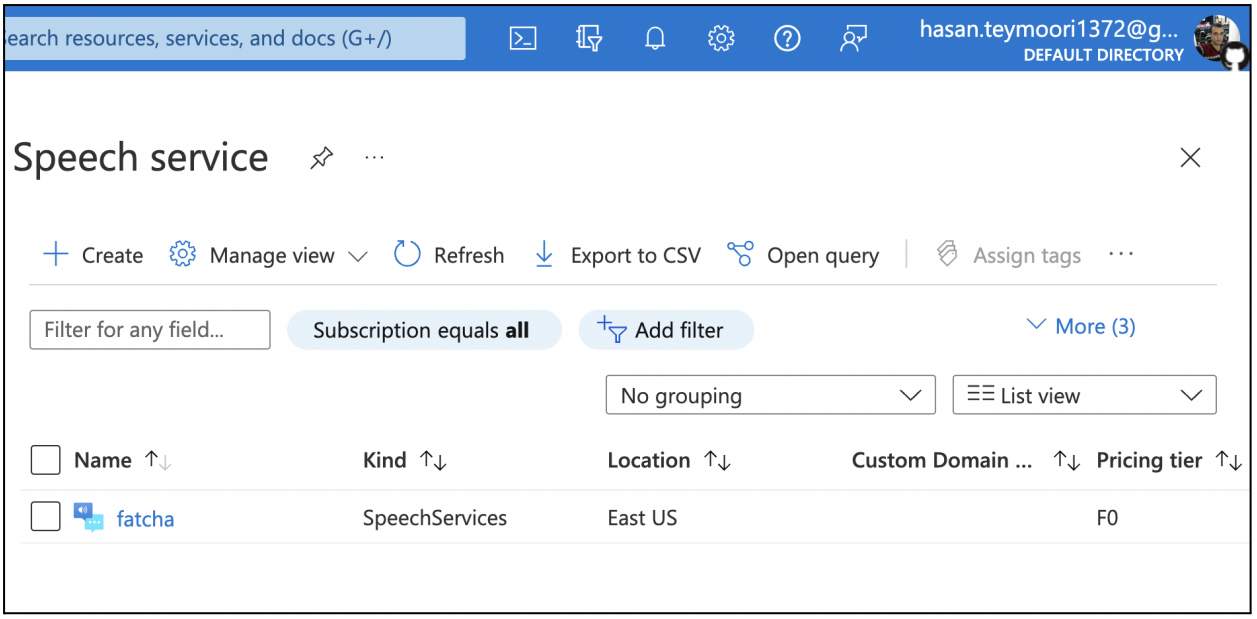
We have used some physical devices like a keyboard and voice speech recognition in order to give commands to the system. you can see a general overview of the keyboard interaction commands that can be given to the system.



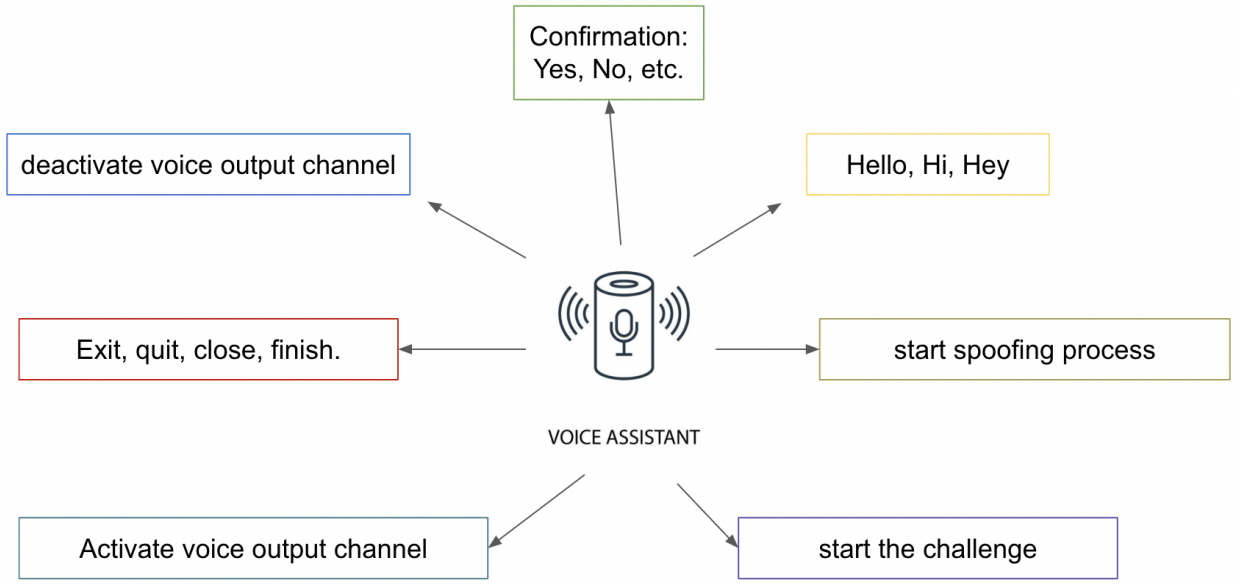

```
# Bind the ESC and q keys with the callback function
window.bind('<Escape>', lambda event: close_win(event))
window.bind('q', lambda event: close_win(event))

window.bind('s', lambda event: start_ch(event))
window.bind('v', lambda event: voice_channel(event))
window.bind('l', lambda event: listening(event))
```

The other device that we could take the advantage of during the interaction is the microphone. To be able to use the microphone and listen to the speech that comes to the system we decided to use Microsoft [Cognitive Services | Speech]service.

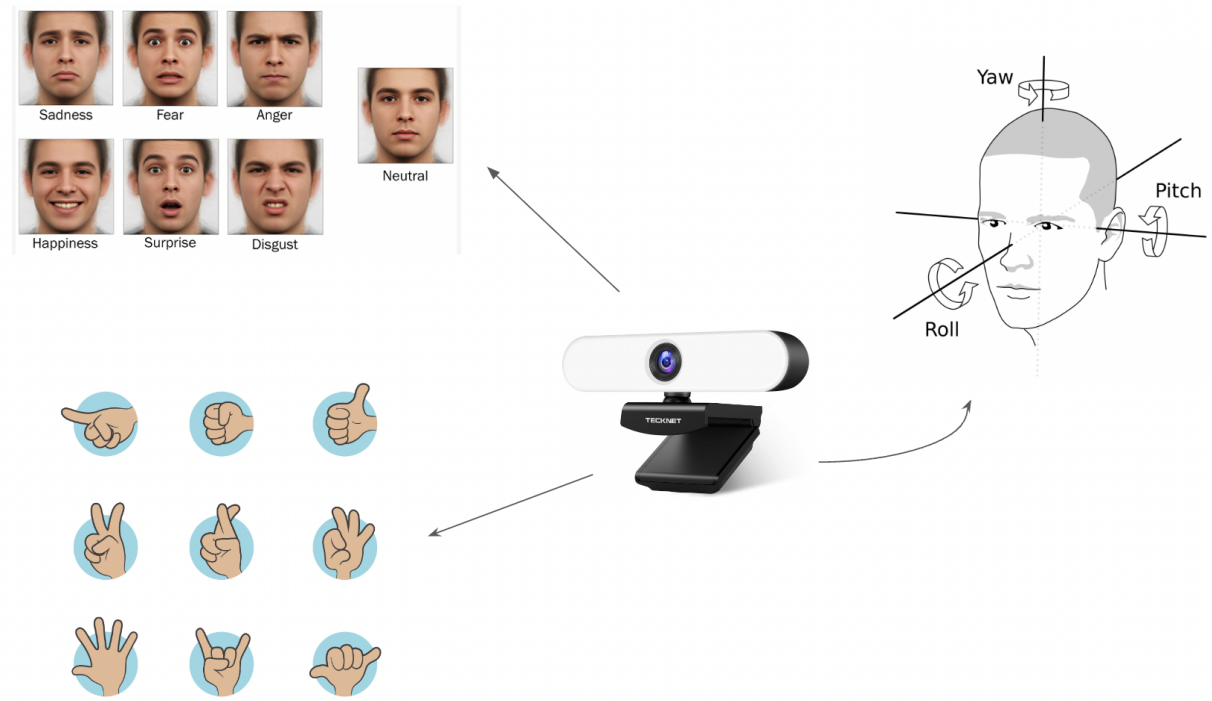
To be able to use the speechesdk we need to create a virtual resource on the cloud base Azure. Azure provides us with a specific credential to work with the resources. The credentials are stored in the config file in the project. Speechsdk sends the captured voice to the virtual machine in the cloud, processes it, and returns the result. During the process of speech recognition by the virtual machine, the execution of the project in our device is stopped for a moment. In the following, you can see our virtual machine.



The list of the command that can be performed by the microphone by the use of speechsdk is visualized in the figure below.



The other device that we have taken the most out of it were the camera. We use the camera to present different kinds of input the systems. The frame of the faces and hands are taken by the camera and processed during the interaction. The confirmation command has been used as a complementarity message in order to complete the process.
During the interaction using this device, facial expressions, gestural actions, and head pose detection have been considered for the application.
A general overview of this device is shown in the following.



We have tested the system according to the default webcam of the laptop and a separate 1080p external camera.

1.5.1 Output Devices

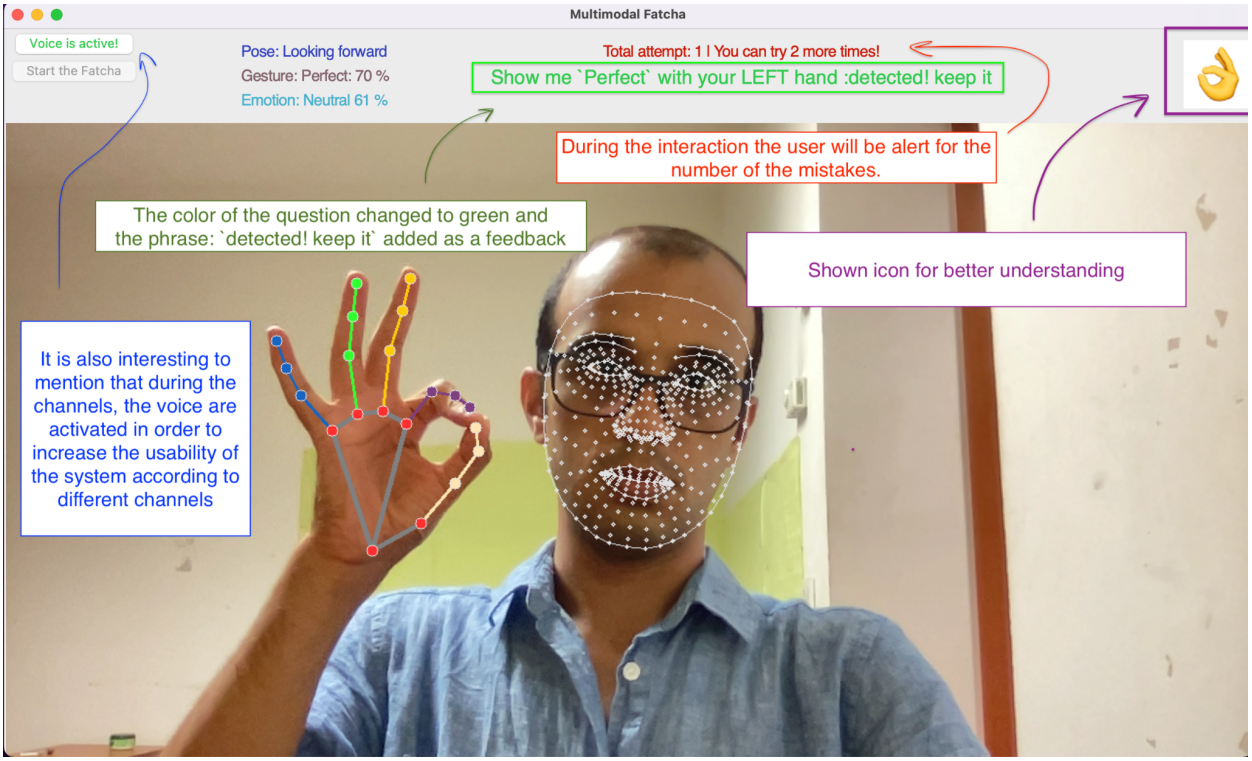
For the case of output devices, in order to be able to interact with the user or machine, we can first of all rely on the laptop default display as the main resource of graphical information. during the interaction from devices to the human side, we have used different emoji icons for the case of gestural interaction, and corresponding text representation to increase the flexibility of the system.

Even in output interaction (from system to human) we have taken to best practices. One was to think that we are somehow interacting with a person that is not able to read (no qualification), in this case, we thought that showing an emoji of the corresponding gestures would be good feedback from the system to the mentioned person in order to increase the understanding.

And second, we decided to support another kind of disability which is visual cases. In this way, the voice output interaction according to the system's speaker would be an ideal device to communicate with those people. Again we take into account the Microsoft speech synthesizer to synthesize the given text for better interaction with the people.

Many feedbacks either textual or graphical have been used during the interaction to increase the understanding of communication flow. Some of them are represented in the following:

We have used redundancy as one of the coordination modalities to interact with the user:



Now that we have a brief introduction about some of the important part of the project, let's start to introduce the technical parts and how we obtain those functionalities.

2. Real and Spoof Detection

The application will be started with an interface with 2 operation options: 1. **Spoof or Real!** and 2. **Start the Fatcha**. The latter option keeps deactivated until the system determined that it is communicating with a human and not a machine. By using the liveness detection technique we will be able to find out whether a frame of the camera is captured from a real human face or recaptured from a picture of the face. That was the place where an imaginary machine or robot has to present a real face not a printed picture or a picture from its monitor.

Once the user was detected as a real person(liveness detected), the second phase will be unlocked and goes for some interesting challenges.
For the case of real or spoof detection, we decided to train a deep neural network with binary classification.

Real and Spoof Detection is divided into **three** steps:

1. Providing a suitable dataset
2. Training a model by the given dataset
3. Recognition of spoofing or real

2.1. Providing DataSet

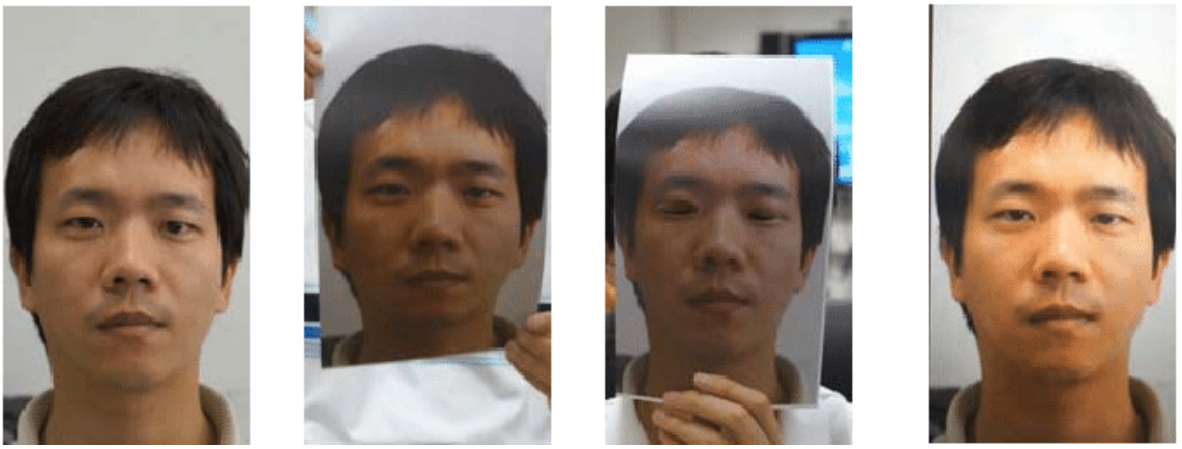
For the purpose of the dataset, we selected a number of datasets available for face anti-spoofing such as link, but unfortunately, the performance was not satisfactory. We have found another exciting approach that is used by a person from this repository.

He has provided this dataset from the **CASIA-FASD** datasets that consist of videos, each of which is made of 100 to 200 video frames. For each video, he captured 30 frames (with the same interval between each frame). Then, with the Haar_classifier, he was able to crop a person's face from an image. These images make up the datasets.
We took his dataset and use it in our training procedure. Of course, we could have done the same approaches ourselves from the CASIA dataset but due to lack of time and concentration of the project on another important task, we decided to use the ready-to-use data set.

The provided dataset does not have a large number of individuals. It consists of only 15 identities for the real case and 15 identities for the fake ones.

The be able to download the dataset into our google colab notebook, we uploaded it into our google drive as a zip file. Link

These are some of the samples from the dataset.



2.2. Training The Model

Once we have the dataset, we start our training with **MbileNetV2** because MobileNet is an architecture that is more suitable for mobile and embedded-based vision applications where there is a lack of computing power. This architecture was proposed by Google. As the system is already computationally expensive due to emotion, gesture, face, etc detection, we decided to decrease the pressure on the system and train a simple architecture.

We have binary classes of **'Spoof'** and **'Real'.** In the training set, we have **10092** images belonging to **2** classes .**2522** images were taken as the testing one.

for our model we are going to make the most use of some callback functions in the **Keras API**:

- `**ModelCheckpoint**` 🏁: callback is used in conjunction with training using `model.fit()` to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

- `**EarlyStopping**` 🚦: Assuming the goal of training is to minimize the loss. With this, the metric to be monitored would be `loss`, and the mode would be `min`. A `model.fit()` training loop will check at end of every epoch whether the `loss` is no longer decreasing, considering the `min_delta` and patience if applicable. Once it's found to no longer decrease, the training terminates.

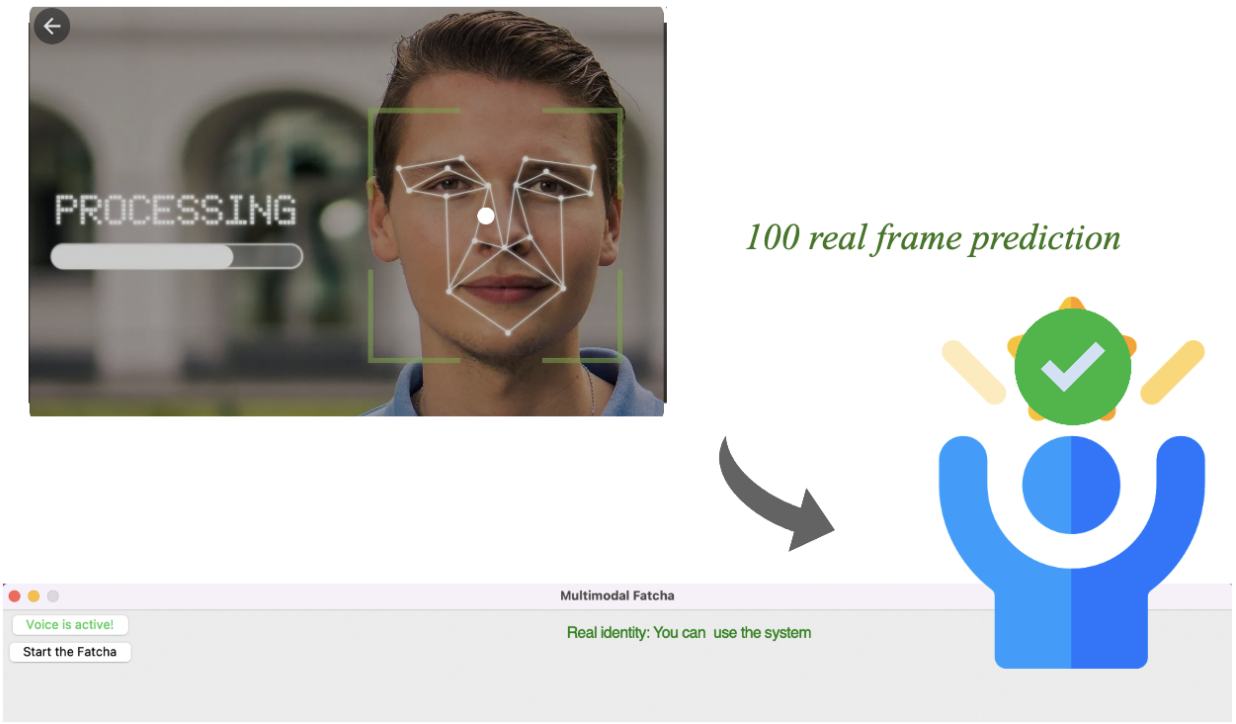
Finally, we saved the best model and its architecture(serialize model to JSON file) in order to use it in real-time.

2.3. Inference of the Model In Real-Time

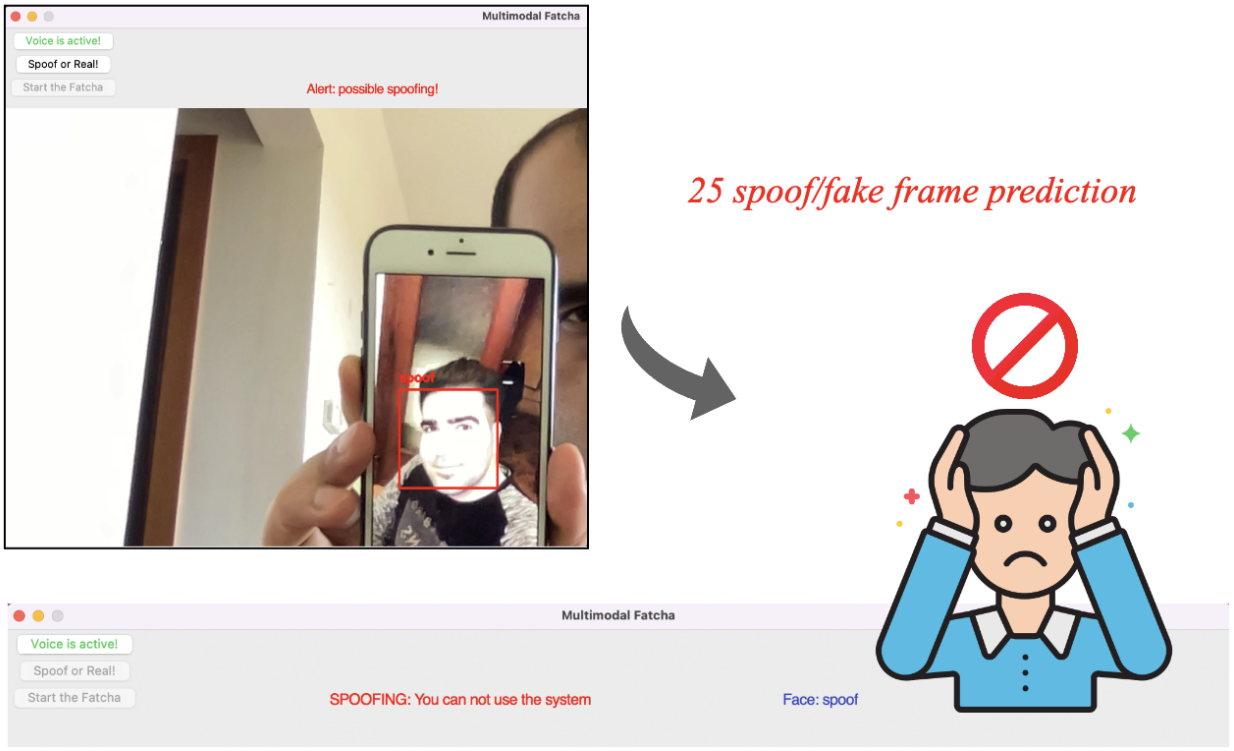
In this stage, we load our trained model and its architecture.
To be able to detect whether we have a real user in front of the camera or a fake one, we need to define some design choices.

- 25 continuous spoof detection frames show a spoofing attack.
- 100 continuous real detection frames would show a real identity.

By implementing some functions for these design choices we somehow create a threshold on the number of frames that have been detected. If we had **100** continuous real detected frames, the user would be recognized as real, but if we had **25** continuous spoof detected frames, it would return as a spoofed user, and access would be denied.



In the above figure, you can see a real user procedure. I the user/machine pass the challenge they would be able to access the second level of the procedure. Thus we activate the `start the Fatcha ` button for them.



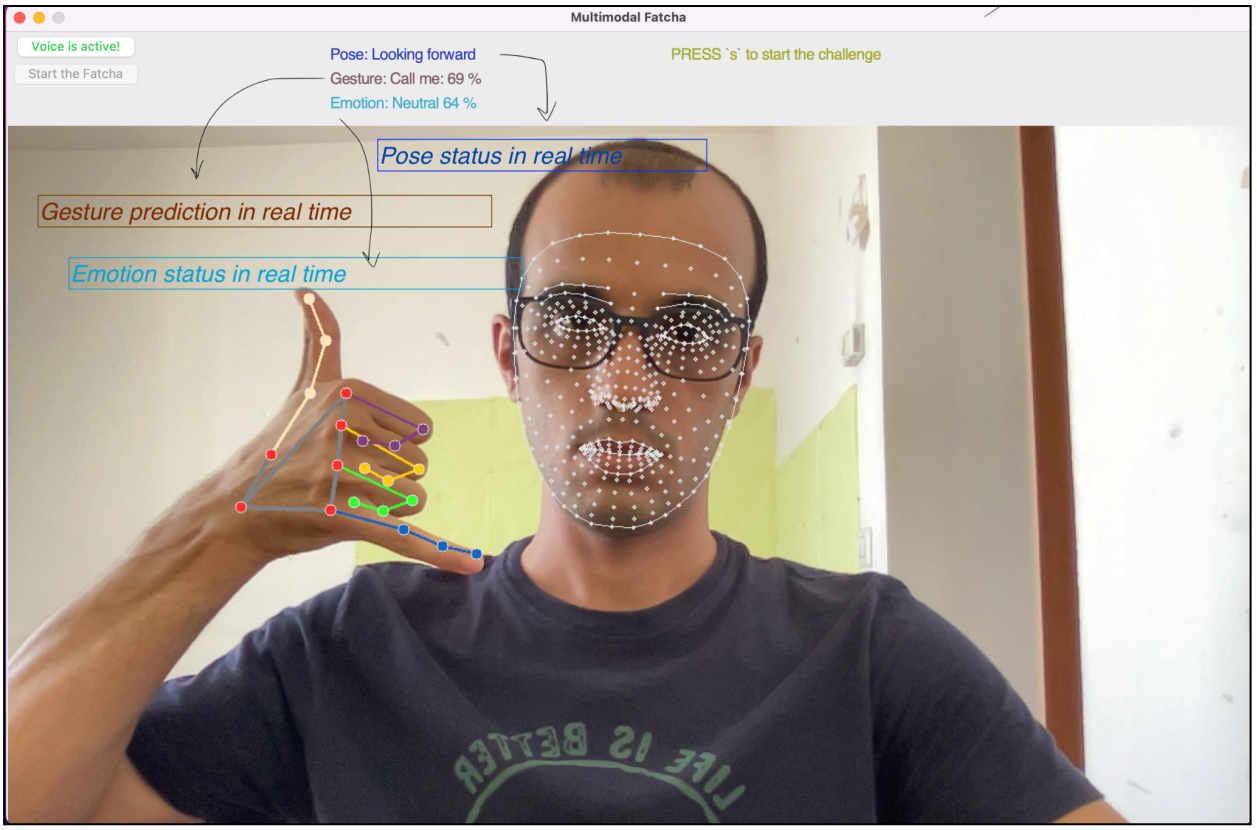
On the other hand, in case the system detects a possible fake or spoofing frame that is the case when a machine or robot tries to present an image of a user in order to pass as a human being. Thus a 25 negative frame would block access to the system.

3. Challenge-Response

Now that we somehow ensure that there is a possible user with a high probability that we interact with, it is time to start the second level of challenge. The user/machine must enter into a new section which is called challenge-response Fatcha. In this stage like the previous one, users should respond to some of the challenges generated entirely at random based on different human nature actions.

All challenges are combined into **3** different challenges:

1. **Head pose (keeping the head up, down, right, left)**
2. **Gesture Recognition (illustrating victory, like, dislike, perfect, etc shapes with hands)**
3. **Emotion Recognition (pretending some demanded emotions e,g, happy, angry, surprised, etc)**



Each of the mentioned functionalities requires a specific kind of implementation, then we need to process them and inject them into a challenge-response loop. Each of the users has 3 chances in total to try to be recognized as a human otherwise they will be rejected by the system.

Our design choices here are the following:

- In general, the user is allowed to make mistakes 3 times in total!. In other words, each user can attempt to be recognized 3 times. (this number is manageable)
- When the challenge started, a subset of the total challenges are selected for the upcoming challenge process at random (the number of the questions is manageable)
- Each question has a limited time to be completed, there is a timer for all the questions.
- To successfully pass a specific question, you need to present the requested action for 25 continuous frames. (the `detected` feedback is shown to help the user the current state of detection)
- To successfully pass the challenge, you need to pass all the questions correctly.

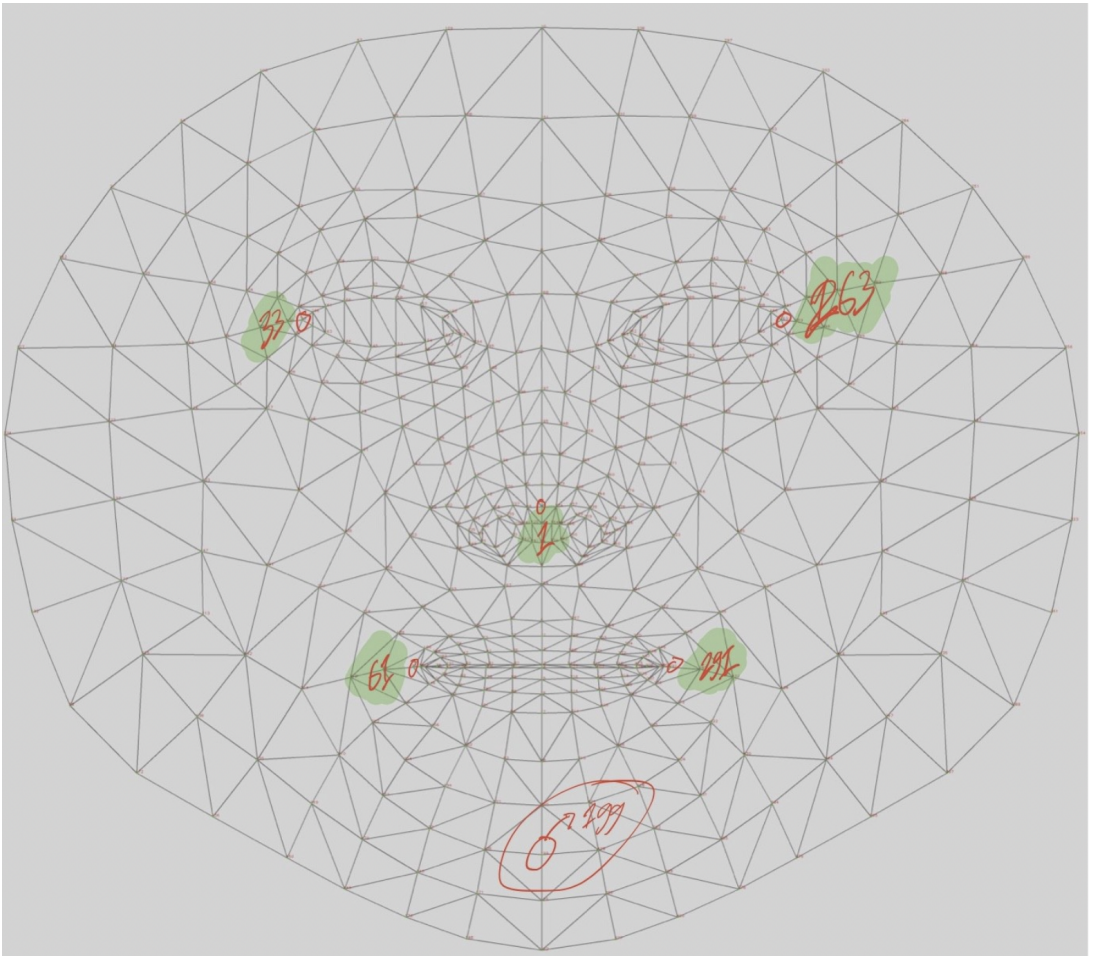
```
challenge = {
    'number_of_challenges': 7,
    'time_per_question': 10,
    'consecutive': 25,
    'allowed_attempt': 3
}
```

Let us now explain the detail for each type of the challenge and the way we implement their functionalities.

3.1. Head Pose

In this stage, we used the **MediaPipe** framework to have to access the face geometry landmarks.
MediaPipe is an open-source framework from **Google** for building **multimodal** (eg. video, audio, any time series data), cross-platform (i.e Android, iOS, web, edge devices) applied ML pipelines. It is performance optimized with end-to-end on-device inference in mind.
MediaPipe **Face Mesh** is a solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor. Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance-critical for live experiences.
Additionally, the solution is bundled with the Face Transform module that bridges the gap between face landmark estimation and useful real-time augmented reality (AR) applications. It establishes a metric 3D space and uses the face landmark screen positions to estimate a face transform within that space. The face transform data consists of common 3D primitives, including a face pose transformation matrix and a triangular face mesh. Under the hood, a lightweight statistical analysis method called Procrustes Analysis is employed to drive a robust, performant, and portable logic. The analysis runs on a CPU and has a minimal speed/memory footprint on top of the ML model inference.

To create this kind of challenge, we define a function that the pose is looking forward to by default. Thanks to MediaPipe Face Mesh, it extracts the special points on the face, then by using some definition and threshold it detects the pose of the head on the sides of up, down, right, and left. These lead to a suitable challenge for the user to keep his/her head on the randomly demanded side. In this type of challenge, the voice assistant will be activated automatically to guide the user when the challenge has been passed.
To estimate the head pose we define a statistical function that according to the 6 important landmarks that were extracted from the total landmarks in faceless, we could estimate the head pose.



The decision would be according to the position of the 6 landmarks as shown in the figure above.

The idea behind the concept was initially taken from [this article] that is trying to estimate the pose by 3D reconstruction and Perspective-n-Point. We have used the same approaches to be able to estimate the head pose.

The PnP problem equation looks like this:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

From this equation, we can retrieve the rotational and the translational matrix. But before we get those matrices, this equation needs to take three inputs, such as:

- The 2D coordinates in the image space
- The 3D coordinates in the world space
- The camera parameters, such as the focal points, the center coordinate, and the skew parameter

Once we have all the inputs, ranging from the 2D coordinates, the 3D coordinates, and the camera parameters matrix:

```
    # Convert it to the NumPy array
    face_2d = np.array(face_2d, dtype=np.float64)

    # Convert it to the NumPy array
    face_3d = np.array(face_3d, dtype=np.float64)

    # The camera matrix
    focal_length = 1 * frame_width

    cam_matrix = np.array([
        [focal_length, 0, frame_height / 2],
        [0, focal_length, frame_width / 2],
        [0, 0, 1]
    ])
```

We could apply the PnP to our problem with help of the OpenCV library.

```
# Solve PnP
success, rot_vec, trans_vec = cv2.solvePnP(face_3d, face_2d, cam_matrix, dist_matrix)

# Get rotational matrix
rotational_matrix, jac = cv2.Rodrigues(rot_vec)

# Get angles
results_rq = cv2.RQDecomp3x3(rotational_matrix)
angles = results_rq[0]
```

The outcome of this equation would be the rotational vector that need to be changed to a matrix. by having the matrix we could determine the Euler Angles (Roll, Pitch, and Yaw) angle and according to that angle, we could estimate the head pose and we are done.

The detailed representation of angles and corresponding head pose is the following screenshot.

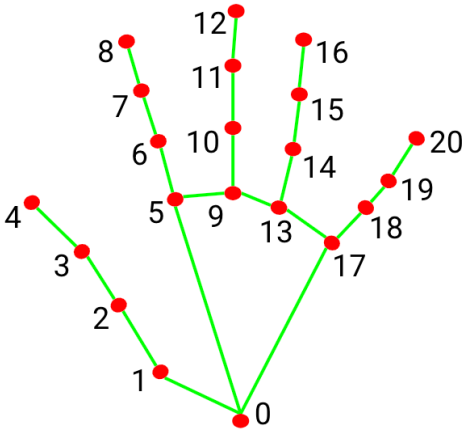
```
        # Get the y rotation degree
        x = angles[0] * 360
        y = angles[1] * 360

        # See where the user's head tilting
        if y < -10:
            head_pose_class = 1  # Looking left
        elif y > 10:
            head_pose_class = 2  # Looking right
        elif x < -10:
            head_pose_class = 3  # Looking down
        elif x > 10:
            head_pose_class = 4  # Looking up
        else:
            head_pose_class = 5  # Looking forward
```

3.2. Gesture Recognition

Another challenge will be generated based on the user gesture. Similar to the Head Pose, this kind of challenge has been implemented by the MediaPipe Hands library.
The hand's landmarks model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

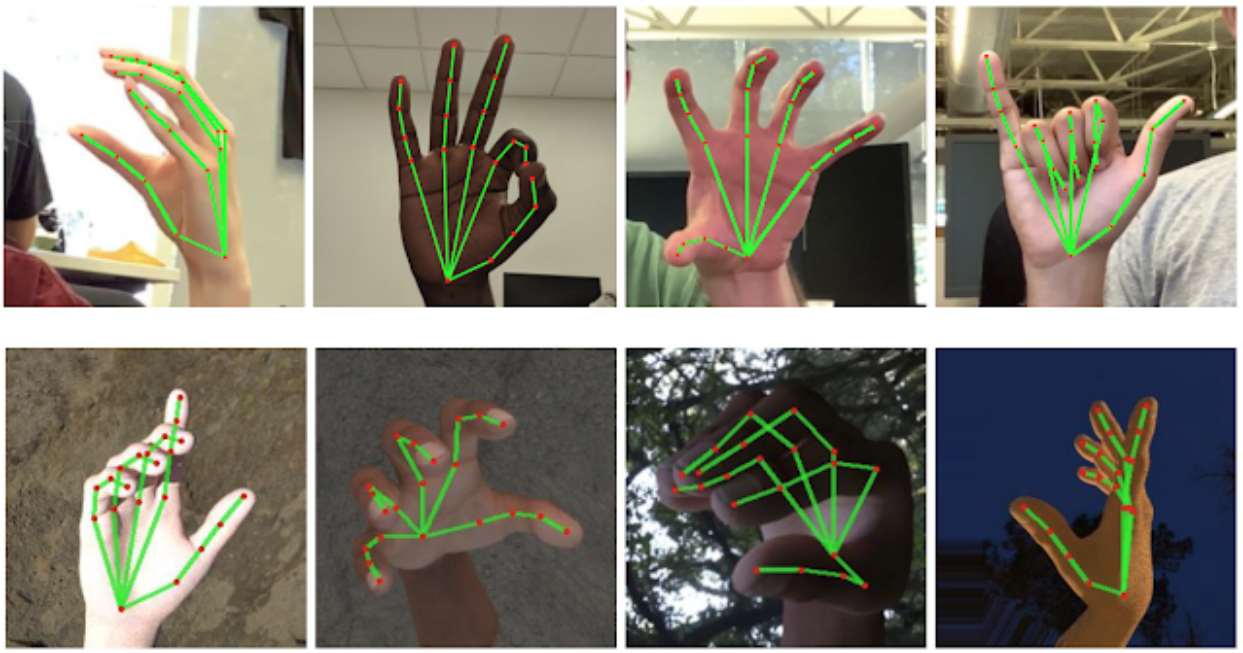
To obtain ground truth data, google has manually annotated ~30K real-world images with 21 3D coordinates, as shown below (they take Z-value from the image depth map if it exists per corresponding coordinate).
To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, they also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

In the following figures that are taken from the original resource of Mediapipe, you can see detailed information about the 21 landmark points and their ground truth annotations information.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

21 hand landmarks



Top: Aligned hand crops passed to the tracking network with ground truth annotation.
Bottom: Rendered synthetic hand images with ground truth annotation.

Having access to the hand landmarks was just the first step of our approach. We thought that the distance between the landmarks for each gesture can somehow represent a specific kind of pattern. We decided the gather all the landmarks while presenting those gestures and train a classification model in order to predict each of the gestures.

The first step, as mentioned above was to create a custom dataset of the landmarks. We implemented a module that accessed the camera and ask you to present a kind of gesture and start recording the landmarks to create the dataset.

In summary, the Implementation of this challenge is divided into 3 different phases:

1. **Dataset Providing**
2. **Training a Model**
3. **Recognizing the gesture using the trained model**

3.2.1 Dataset Generating

First of all, we extracted the coordinates of hand landmarks using the hand-made module of key_point_logging.py in the gesture folder. This function gets as many frames as you want per each gesture shape and writes the coordinates of its points in a CSV file. (after applying a normalization) Therefore, our dataset is a CSV file containing handmarks coordinates and is ready for training.

The way in which we normalize the landmarks is represented as follows:

This is the plain landmarks that we have initially

ID : 0	ID : 1	ID : 2	ID : 3	·····	ID : 17	ID : 18	ID : 19	ID : 20
[551, 465]	[485, 428]	[439, 362]	[408, 307]	·····	[633, 315]	[668, 261]	[687, 225]	[702, 188]

Change the landmark coordinates into a relative coordinates w.r.t ID:0

ID : 0	ID : 1	ID : 2	ID : 3	·····	ID : 17	ID : 18	ID : 19	ID : 20
[0, 0]	[-66, -37]	[-112, -103]	[-143, -158]	·····	[82, -150]	[117, -204]	[136, -240]	[151, -277]

Flatten from 2D-array to one dimensional array

ID : 0		ID : 1		ID : 2		ID : 3		·····	ID : 17		ID : 18		ID : 19		ID : 20	
0	0	-66	-37	-112	-103	-143	-158	·····	82	-150	117	-204	136	-240	151	-277

Normalize to the maximum value

ID : 0		ID : 1		ID : 2		ID : 3		·····	ID : 17		ID : 18		ID : 19		ID : 20	
0	0	-0.24	-0.13	-0.4	-0.37	-0.52	-0.57	·····	0.296	-0.54	0.422	-0.74	0.491	-0.87	0.545	-1

3.2.2 Training Model

Once we prepared the dataset, we went to train a model. We separated the target feature from the features and pass to a deep architecture.

As our task is based on a security measurement, we have to have a highly accurate model.

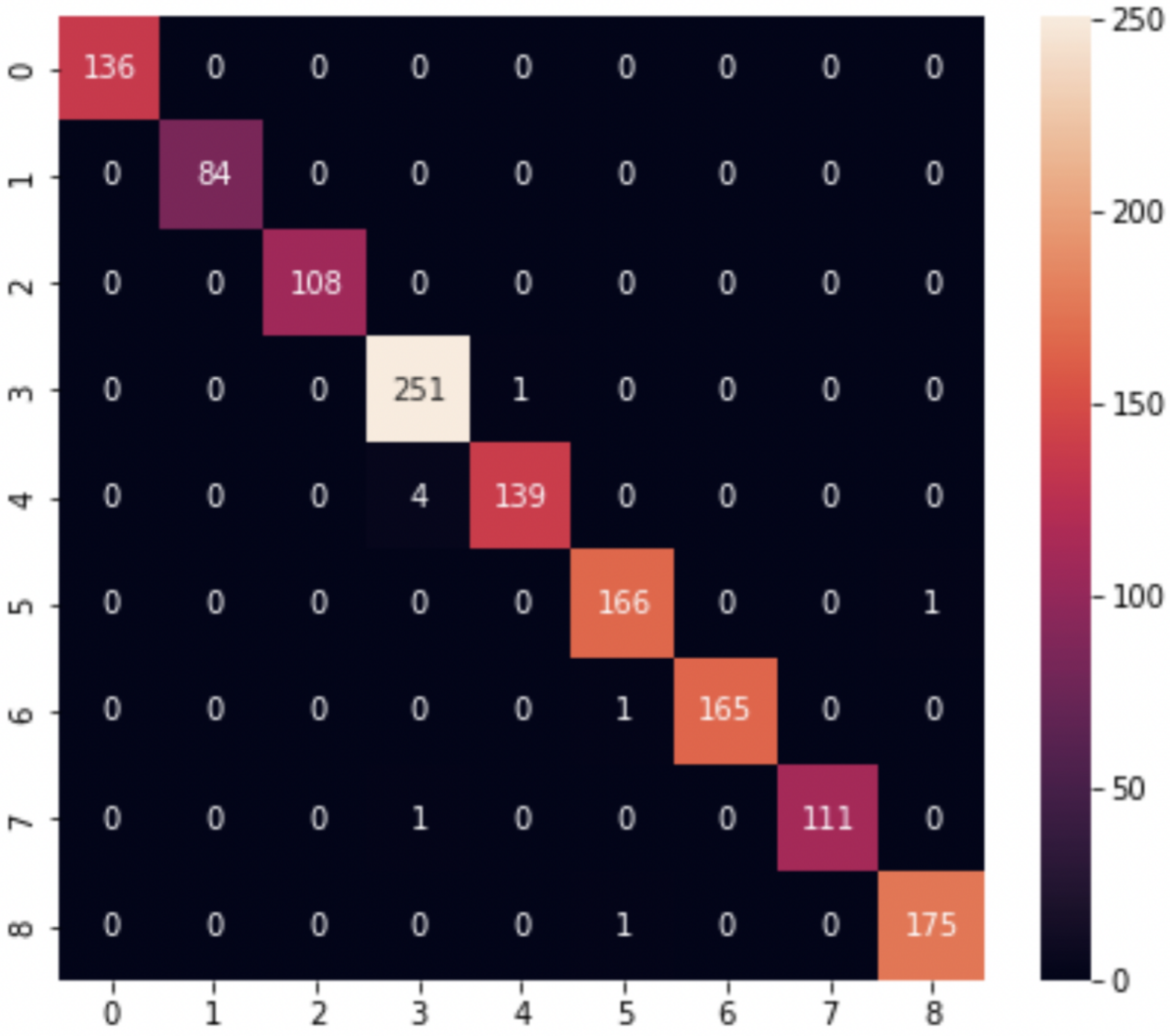
For training our model we had the following constants and configurations:

- ➔ BATCH_SIZE = 128
- ➔ N_CLASSES = len(class_names) = 10 different gestures
- ➔ The gestures are 'Like', 'Dislike', 'Call me', 'Four', 'Hi-five', 'Victory', 'Perfect', 'Mamma Mia!', 'Finger crossed'.
- ➔ **ModelCheckpoint** to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved
- ➔ **EarlyStopping** to stop training when there would be no improvements in loss
- ➔ **ReduceLROnPlateau** to monitor a quantity and if no improvement is seen for a patience number of epochs, the learning rate is reduced
- ➔ optimizer='adam'

The output model is saved as **'key_points_classifier.hdf5'** in the same directory and will be used for gesture detection in the main application.

We have not used any popular well-known architecture for the training as it was not an image-based task. Therefore, since we were not dealing with an image-based task but simple statistical CSV files, we decided to use a simple architecture to train the model. Indeed the result was absolutely perfect. You can see the confusion matrix and the performance of the model in the following:

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       136
           1       1.00      1.00      1.00        84
           2       1.00      1.00      1.00       108
           3       0.98      1.00      0.99       252
           4       0.99      0.97      0.98       143
           5       0.99      0.99      0.99       167
           6       1.00      0.99      1.00       166
           7       1.00      0.99      1.00       112
           8       0.99      0.99      0.99       176

    accuracy                           0.99      1344
   macro avg       1.00      0.99      0.99      1344
weighted avg       0.99      0.99      0.99      1344
```



These results clearly have shown us that our first interpretation of gestures was correct and as perfect as possible. So we have a good classification of the hand gestures according to the distance between the hand landmarks.

3.2.3 Inference of the Model to Recognize the Gestures

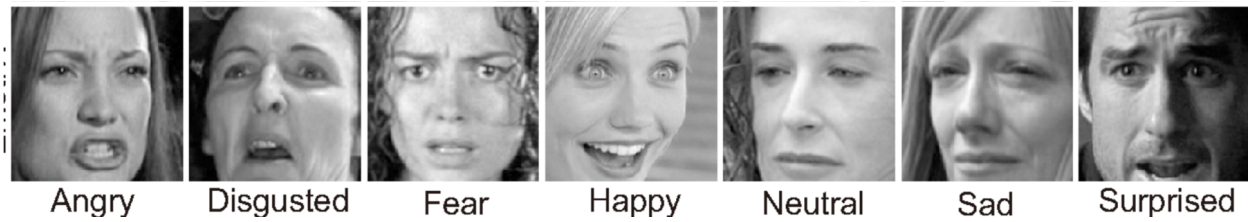
Now we have the model and we can recognize the gesture that the user has shown on the camera. The demanded gesture will be chosen randomly by the system and it would be combined with the head pose and the Emotion recognition.

3.3. Emotion Recognition

For the last type of challenge we decided to add emotional challenges to the procedure. Emotion recognition is the process of identifying human emotion. People vary widely in their accuracy at recognizing the emotions of others. Generally, the technology works best if it uses multiple modalities in context.

3.3.1 Training Model

First of all, we trained a model based on facial emotion recognition. In order to do this, we used the **FER2013** dataset (**Facial Expression Recognition 2013 Dataset**) given by Kaggle. Dataset
The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image.
The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the validation set consists of 7178 examples. The following image is a sample of each face's emotional mood.



We have chosen again the same familiar callback function to handle the training process effectively.

1. **ModelCheckpoint**
2. **EarlyStopping**
3. **ReduceLROnPlateau**

Then, we passed the input into the deep CNN and obtained a classification model. This model helped us to predict the emotion during the challenge process.

Libraries and References

Gdown: Used to download the raw files from google drive.

OpenCV-Python: is used for some image preprocessing.

Pandas: used in order to work with the annotation information

Tensorflow: is used to train the model as well as image augmentation

Keras: Keras is an open-source software library that provides a Python interface for artificial neural networks.

Numpy: is used to perform some of the functionalities like correlation

Pillow: Python Imaging Library

mediapipe: offers cross-platform, customizable ML solutions for live and streaming media.

azure.cognitiveservices.speech: for handling the speech recognition and synthesizing