Department of Computer Science

SAPIENZA
UNIVERSITÀ DI ROMA

Hand Gesture Recognition.
AI Lab: Computer Vision and NLP
Applied Computer Science and Artificial Intelligence

Olha Biziura
Lin Can
Aruta Srymova

Professor:
Daniele Pannone

Contents

1. Introduction

Analysis of hand gestures is one of the prominent areas of research within computer vision. Gesture-based control is a method of controlling devices or software with hand signs or body movements as an alternative to traditional physical inputs such as keyboards, buttons, or touchscreens. Cameras, deployed as sensors, tend to recognize and extract necessary data to identify and classify different gestures. This paper presents a Python-based project that leverages computer vision techniques and libraries such as OpenCV, Scikit-learn and Mediapipe to analyse hand gestures in real-time using a webcam and develop a system capable of subsequent action triggering, that is executing various commands on the PC. This would allow users to interact with technology in a more intuitive, natural way, making it more accessible.

Throughout this paper, we will delve into the technical details of our project, exploring the methodologies and algorithms employed to process video frames and extract meaningful hand gesture information. We will also discuss the challenges faced in achieving accurate gesture recognition, including background subtraction, hand segmentation, feature extraction, and classification techniques.

2. Background

In recent years, several methods have been proposed to tackle the challenges associated with hand gesture analysis. With the advent of deep learning and the availability of large-scale annotated datasets, researchers have made significant progress in developing more robust and accurate hand gesture recognition systems. In our project, the major part in detecting the hand gestures depends on Mediapipe, and to predict the hand sign corresponding to the analysed hand gestures, we employed the RandomForestClassification algorithm.

2.1. Mediapipe

Mediapipe is being developed by Google. It is an open-source framework that allows the creation of customizable pipelines for applications processing real-time multimedia. It has a wide range of functionalities, such as video processing, object detection, hand gesture and facial recognition. Computer vision pipelines themselves are designed to process the sensory

data and generate perceptual outputs - object localization or hand keypoint streams. They involve various components, including model inference, media processing algorithms, and data transformations. Pipeline graphs receive input data and output processed information. Currently, Mediapipe is in its alpha version, v0.7. API changes, breaking compatibility, are expected to become available by the time it reaches v1.0.

2.2. Pickle module

In our project, alongside mediapipe we used Python's pickle module to save the extracted hand landmarks and the corresponding labels in a serialised format. Pickling is a process of serialisation and deserialisation of the objects. By saving the data and labels as a pickle file, we were able to persistently store the extracted landmarks and labels on disk, and later easily load and reuse the data in future sessions without needing to reprocess the images. Pickle

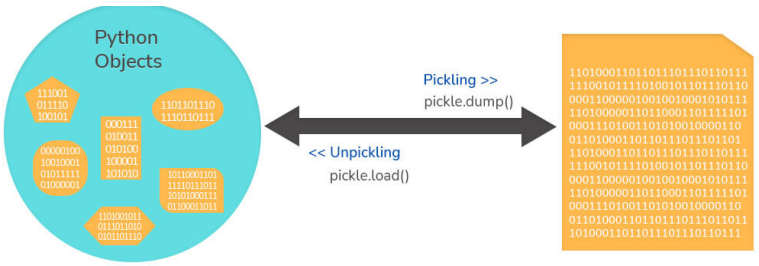


Figure 1. The Pickle Module.

files are binary files [Fig.1], which means they are more efficient in terms of storage space compared to other text-based formats. Since the data can potentially contain a large number of hand landmarks, storing them as a pickle file helps save disk space.

2.3. Random Forest Classification

RandomForestClassifier is a machine learning technique widely used for classification tasks. RandomForestClassification belongs to the ensemble learning family, which combines multiple decision trees to create a robust and accurate predictive model. It goes through different steps, which are data preparation, building of decision trees with random subsets of the training data and features, training the trees by splitting the data based on different feature

values, voting for predictions of all decision trees in the ensemble, handling new examples, and, lastly, model evaluation based on accuracy and precision to understand how well the model is performing on the data[1][2].
In our project, the algorithm leverages a training dataset consisting of labelled hand gesture samples to learn the patterns and characteristics of different signs. The random selection of features and bootstrap aggregating of decision trees within the random forest framework ensure diversity and reduce overfitting. By using the trained RandomForestClassification model, we were able to effectively classify and predict the hand sign represented by the analysed gestures in real-time, enabling seamless and accurate interaction with the system.

3. Implementation

3.1. Gathering data

The first step of the implementation of our project was to create datasets, that is to gather data for our model. Gesture recognition is considered as one of the solutions of Mediapipe. Regarding vision tasks, the library can provide different functionalities, like object detection, image segmentation, face-, hand-, pose landmark detections and holistic landmarks detection [3]. For our project, we took on the holistic approach to collecting data with mediapipe.

3.1.1. Holistic model

The MediaPipe Holistic Landmarker allows combining components of the pose, face, and hand landmarkers to create a complete landmarker for the human body. We can use this task to analyse full-body gestures, poses, and actions. This task uses a machine learning (ML) model on a continuous stream of images. The task outputs a total of 543 landmarks (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand) in real-time [4]. For the camera to recognise the gestures presence of the hand only is not enough. In this case, the camera has to capture both the face and the hand [Fig. 2 & 3].

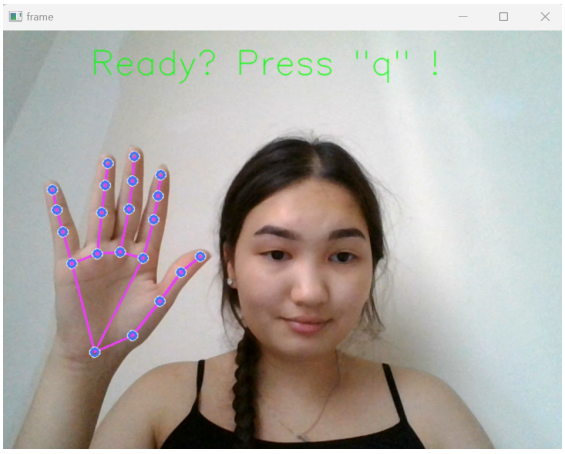


Figure 2. Frame with both face and hand.

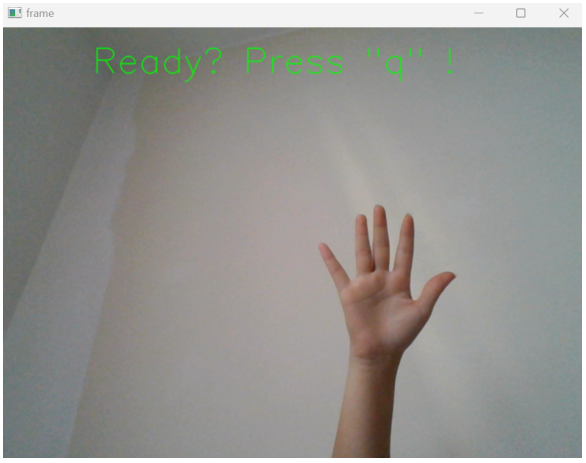


Figure 3. Frame without face, hand only.

To collect a dataset consisting of 300 pictures, the user has to press the key “Q”. In each iteration of the loop, the program captures a frame, processes it using the Mediapipe model, and displays the processed image with hand landmarks overlaid. It saves the frame as an image in the corresponding directory for the current sign. After saving the specified number of frames [Fig. 4], it displays an option to either accept or reject the current dataset. Once all the necessary signs’ landmarks have been processed, it releases the video capture object and the program ends.

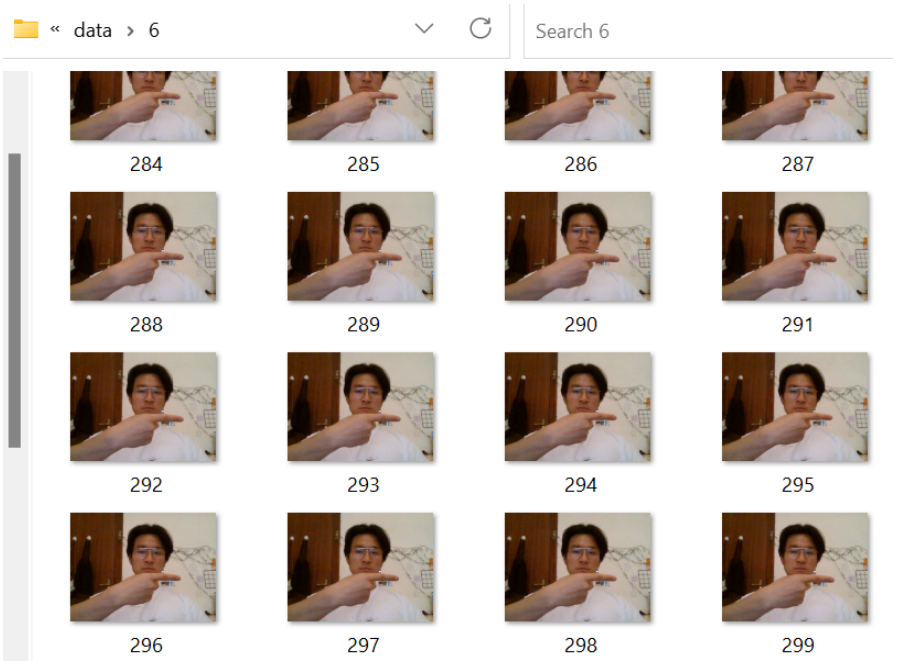


Figure 4. 300 frames saved as the dataset.

3.2. Classifying data

The next step is classification of the dataset. At this step, the code performs the capture of hand landmarks, their coordinates, and normalises them by subtracting the minimum value from each coordinate. The values are temporarily saved in the list. Later we create a file called “data.pickle” and open it in write binary mode. This way we serialise the data. Overall, this code loads images from a specified directory, processes them using the MediaPipe Hands module to extract hand landmarks, and stores the extracted landmarks along with their labels in a pickle file. The reasoning behind using the pickle file is based on the efficient storage and retaining the data structure. Pickle files are binary files, which means they are more efficient in terms of storage space compared to other text-based formats. Since the data can potentially contain a large number of hand landmarks, storing them as a pickle file helps save disk space. Also, pickle preserves the original structure of the Python objects, including lists, dictionaries, and their nested elements. By using pickle, we can store the data and label lists as a dictionary object, making it easy to load and access the data in its original structure. In addition to this, it’s necessary to mention that pickle files can be easily transferred between different systems and platforms while retaining the integrity of the data. Once the data is saved as a pickle file, it can be loaded and used on any system with Python and the required dependencies.

3.3. Training the model

During the training stage, we unpickle the file and use the scikit-learn library to train a RandomForestClassifier on a dataset of hand sign images. We also utilise train_test_split from sklearn.model_selection module for splitting the data into training and testing sets. 20% of the dataset is used for testing, and 80% for the training. Choosing the test size involves finding a balance between having enough data for effective model training and having a suitable amount for evaluating its performance. If a higher test size were chosen, it would allocate a larger portion of the data for testing, reducing the amount available for training. This can result in a less robust model that may struggle to generalise well to new, unseen data. Conversely, a lower test size would provide more data for training, but the evaluation of the model's performance would be based on a smaller test set. This smaller sample may lead to a less reliable estimate of how well the model performs on unseen data. To strike a

reasonable balance, a test size of 0.2, or 20%, is used as it provides a satisfactory amount of data for both training and evaluating the model.

Then, we calculate the accuracy score by comparing the predicted labels with the true labels using accuracy_score() and print it as a percentage.

```
99.86101459346769% of samples were classified correctly !
```

3.3.1. Confusion matrix

In addition, we construct a confusion matrix using confusion_matrix() with the true labels and predicted labels by creating a heatmap visualisation of the confusion matrix using seaborn.heatmap() and matplotlib.pyplot [Fig. 5]. Through the confusion matrix, we highlight the main diagonal, where values close to 100% indicate successful and precise classifications. The high percentages on the main diagonal reveal that our model exhibits outstanding accuracy in correctly predicting instances for each class. This level of precision is crucial in applications where reliable and accurate classification is paramount.

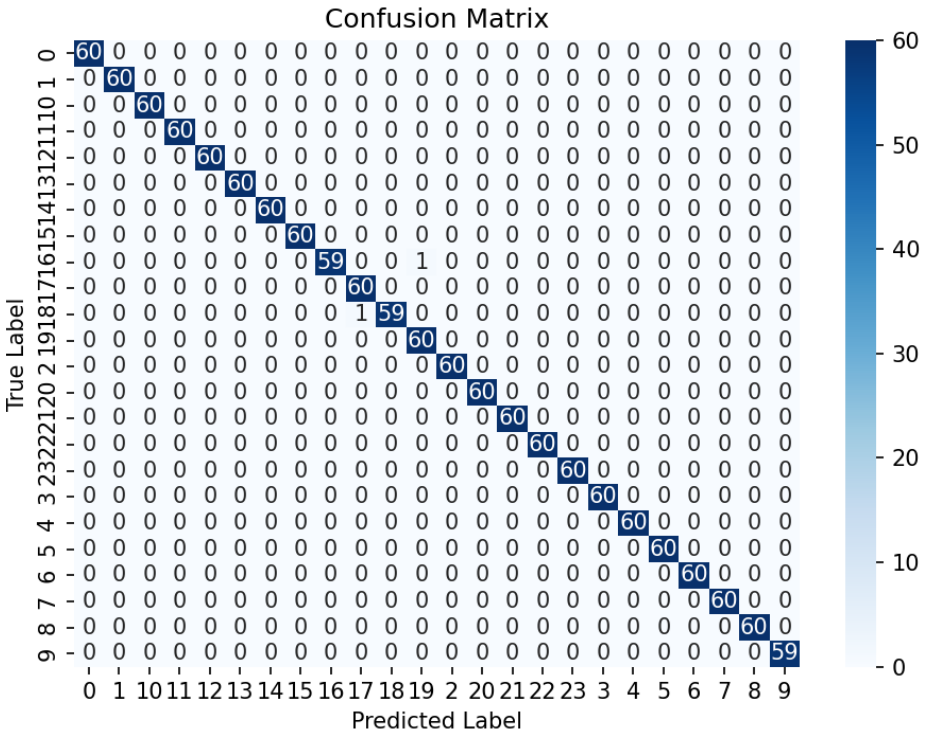


Figure 5. Confusion matrix.

3.4. Real time prediction and execution of the commands

During the execution of the main program, we go through the above-mentioned steps of hand landmarks detection, processing, drawing connections, and consequent data extraction. The pre-trained model is used to make a prediction.
Then based on the result of the prediction, it's going to execute certain commands on the PC [Table 1].

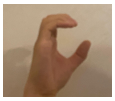
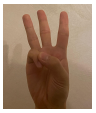
SIGN	ACTION
	CREATES A NOTEBOOK FILE
	IT STOPS RECOGNIZING SIGNS
	OPENS THE CALCULATOR
	OPENS THE TIMER
	RESTART RECOGNIZING SIGNS
	SCREENSHOTS THE CURRENT STATE OF THES SCREEN AND SAVES IT
	LISTENS TO YOUR REQUEST AND IT SEARCHES IT ON GOOGLE
	OPENS GOOGLE TRANSLATE
	LISTENS TO YOUR REQUEST AND PLAYS THE SONG ON YOUTUBE
	LISTENS TO YOUR REQUEST AND SHOWS YOU THE WEATHER OF THE CITY

Table 1. List of commands that a user can execute with Sign Assistant.

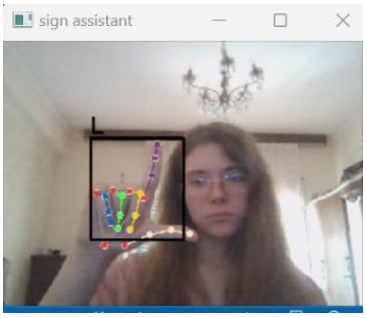


Figure 6. Frame prints the letter of the gesture.

When developing the project and creating the signs for the commands we used American Sign Language as a reference [5]. Predicted gestures are mapped to characters using the ‘alphabet’ dictionary. To trigger actions with the Sign Assistant user had to keep the same sign for 1.5 seconds. We implemented this approach with a timer because the frame is responsive to any change in the gestures. In addition to this, Assistant prints the corresponding letter of the gesture according to ASL, whether it has an assigned command or not [Fig. 6].

During the execution of the commands the program engages SpeechRecognition and gTTS to make the interaction with technology more natural and accessible. To search in Google, play music, or show weather, a function 'convert' that takes an audio parameter is defined in the code. Within this function, the code uses the Recognizer object to convert the audio to text using r.recognize_google(audio). The response to the command from the model is saved as an audio file using the Google Text-to-Speech library, gTTS. In case the Assistant does not recognise a speech, the “repeat” MP3 file is played. This ensures the consistent communication between the system and a user. When the various actions are triggered, the program also verbalises the commands. Corresponding audio files were pre-created by means of gTTS [Fig. 7].

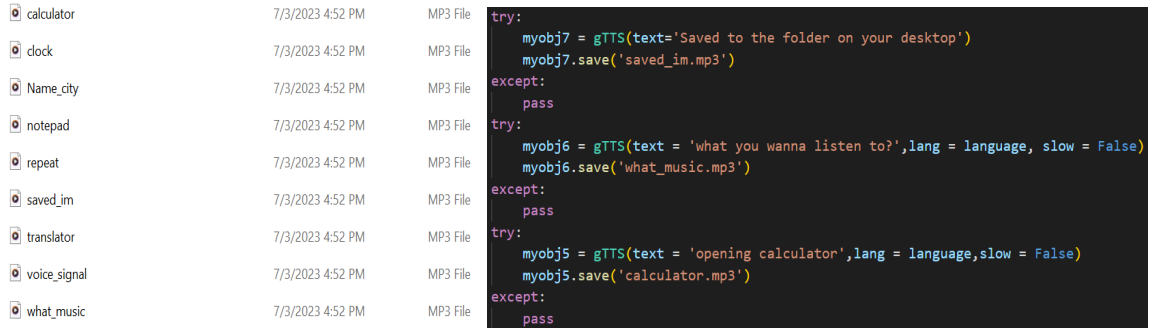


Figure 7. Audio files.

The requirements for the project are:

- OpenCv-python
- numpy
- mediapipe
- pickle

- scikit-learn
- matplotlib
- seaborn
- playsound
- gtts
- webbrowser
- speech_recognition
- requests
- PathLib
- PIL

4. Conclusion

The goal of this project was to create a gesture recognition based assistant that captures the signs and executes commands without the user having to rely on a keyboard. This objective was reached by extensive research on machine learning techniques. Moreover, the proposed project could be enhanced by adding more commands and engaging the full human body. Hand gesture recognition has the potential to serve biometric identification and authentication objectives. Our project possesses the ability to restrict hand recognition without the face being included within the frame, therefore it has a potential in constituting the security. Through the analysis and recognition of distinct hand gestures, it can furnish an extra level of safety for access control systems, thereby preventing unauthorised entry to sensitive locations or devices.

References

1. Shafi, A. (2023). *Random Forest Classification with Scikit-Learn*. DataCamp https://www.datacamp.com/tutorial/random-forests-classifier-python
2. Scikit-learn. *Sklearn.ensemble.RandomForestClassifier* https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. Google Developers. (2023). *MediaPipe Solutions Guide.* https://developers.google.com/mediapipe/solutions/guide
4. Fernandez, J. *MediaPipe.* https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md
5. National Institute on Deafness and Other Communication Disorders (2021). *American Sign Language*. https://www.nidcd.nih.gov/health/american-sign-language