Hand Gesture Recognition for Interaction with Computers

Sukanya Baichwal Hailiang Dong Hasmitha Jalla Ananya Reddy Katpally
University of Texas at Dallas [*]
800 W Campbell Rd, Richardson, TX 75080
{ Sukanya.Baichwal, Hailiang.Dong, Hasmitha.Jalla, Ananyareddy.Katpally} @utdallas.edu

Abstract

This project aims to develop a real-time computer vision-based system that can recognize hand gestures and utilize them to control the computer. The system can identify the position and type of the gesture, enabling a range of interactions with the computer. The project is implemented in two phases. The first phase involves fine-tuning a pre-trained object detection model using the HAGRID dataset, enabling recognition and localization of hand gestures in given images. In the second phase, the system captures live images from a camera and utilizes the previously trained model to identify gestures in real-time. Based on the identified gesture and its location across time, a program is developed to control the mouse and keyboard, allowing for seamless interaction with the computer. The codes of this project are publicly available at https://github.com/LeonDong1993/Air-Gesture-Control.

1. Introduction

Computer vision systems have significantly advanced over the past decade, leveraging the rapid development of machine learning and deep learning algorithms to analyze images and video. These systems have become increasingly prevalent in various industries, providing accurate and efficient solutions to real-world problems. For example, these systems can count the number of free spots in a parking lot, identify objects in a retail store, or diagnose diseases in medical imaging. With the advances of computer vision technology, the scope of its applications is ever-widening, enabling more efficient and effective solutions to numerous challenges.

Despite the usefulness of computer vision systems, the most prevalent methods of interacting with computers remain the mouse and keyboard. While these tools are highly effective and efficient for many tasks, they can be cumbersome and inconvenient when a user is not sitting in front of or around the computer. For example, if a presenter wants to switch slides during a lecture that also uses board to write detailed formulas or draw pictures to help the understanding of the material, they need to physically walk over to the computer to use the mouse or keyboard. Similarly, if someone is watching a movie with friends and receives an important phone call, they might need to pause the movie by walking over to the computer and pressing a button on the keyboard. These examples illustrate the limitations of traditional computer interaction methods when it comes to hands-free or remote access cases. As such, there is a need for alternative methods that enable more efficient and convenient interactions with computers.

Motivated by the success of the Microsoft Kinect system, which utilized human body movements to interact with game consoles, this project aims to develop a computer vision system that uses human hand gestures to control computers. Human hands are incredibly flexible and capable of performing a wide range of gestures, making them an ideal tool for controlling computers in various contexts. By leveraging the power of computer vision, the system can recognize and interpret these hand gestures to perform various computer operations, such as switching between slides, control the playing of music at background, or performing other tasks. Ultimately, this system aims to enable more natural and seamless interactions between humans and computers, bridging the gap between the physical and digital worlds.

Our system consists of two parts. The first part is a vision model that can identify hand gestures and their locations in images. We treat each hand gesture as a type of object, and fine-tune a Faster RCNN model on the HAGRID dataset using the Detectron2 [] framework to achieve this goal. The second part is a client program that continuously captures frames from a live camera and uses the above model to infer the hand gesture and its location in each frame. Based on the recognized gesture or its movement direction, the program sends mouse or keyboard signals to the computer, allowing the user to interact with the computer, without the need to be physically close to it. For instance, a teacher or presenter could show an ”OK” gesture to the camera, and the program would automatically send a ”Down” key sig-

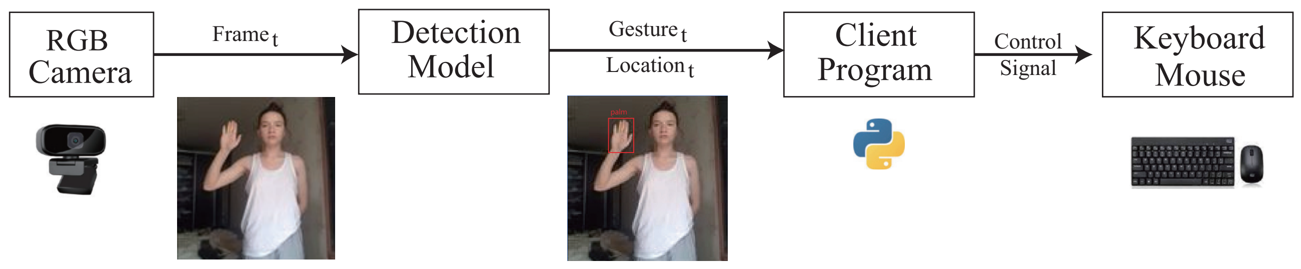

Figure 1. Overview of our system architecture.

nal to the system, enabling them to switch to the next slides (The mapping between gesture and the mouse-keyboard action can be arbitrarily defined in the program). We will use Python along with the pyautoGUI package to develop this program. This system has many potential use cases, such as in education, entertainment, and other fields where hands-free computer interaction would be beneficial.

2. Related Work

The advances in machine learning and modern technologies have made it possible for researchers and developers to exploring different ways to interact and control with the computers beyond keyboards and mouses.

One of these approaches is to use speech recognition techniques [1]. However, it has limitations with respect to how different people pronounce the same word differently and noisy environments that can hinder the correct recognition of a particular word.

Another alternate way of achieving the interaction with computers is to employ gloves which are fitted with sensors that capture details of the hand's orientation and the degree of the fingers. However, this approach is limited to the comfort and size of the gloves to the person wearing it, along with expense incurred for the sensors.

With the rapid development in computer vision these years, systems using traditional RGB cameras can prove a alternative way for interacting with or controlling the computers. In this work, we explore one such possibilities of using hand gestures to interact with the computers use just normal RGB camera. Our system is independent of extra sensor and is robust to various real life noise environments, and provide a new as well as efficient way of interacting with computers.

3. Method

3.1. System Overview

Figure. 1 shows the overall architecture of our system. It consists of two major components: (1) an object detection model for hand gesture recognition and location identification and (2) a client program that control the keyboard and mouse by sending signals based on the gesture identified. The overall flow of system is as follows: first, the RGB camera keeps reading the live frame from the environment and the frame at current time t is passed to the object detection model. Then, the model performs object detection on the input frame and output the location and gesture type of hands. After that, the client program sends keyboard and mouse control signals to the system by looking the the current gesture and pre-defined action for that gesture. Finally, the system will respond to the signal and we have completed our interaction with the computers.

In the next two sections, we will introduce how to develop the two major components.

3.2. Object Detection

In this project, we will employ the Detectron2 framework [7] to train a customised object detection model. Figure. 2 shows the detailed architecture of our model. It is mainly a Faster RCNN architecture [6] augmented with Feature Pyramid Networks (FPN) [4] techniques. In this case, we can handle objects with different size more easily. Note that ResNet [2] with 101 layers as the backbone in FPN to extract the features from the input images.

To fine-tune the model for hand gesture recognition, we need to prepare a dataset of hand gesture images that can be used to train the model. In this project, we use the HAGRID [3] dataset, which contains images of hand gestures labeled with their corresponding class and bounding box coordinates. Specifically, we start with a Faster RCNN + RPN model that is pre-trained on the COCO dataset [5]. We then fine-tune this model on the HAGRID dataset. Note that the number of classes of HAGRID and COCO dataset is different and therefore the last classification head of the model is changed accordingly. During the fine-tuning process, we adjust the hyperparameters of the model, such as the learning rate and batch size, and weight decay, to optimize its performance on the HAGRID dataset.

Once the fine-tuning process is complete, we will have a model that is trained to recognize hand gestures in images with high accuracy. We can then use this model in the second part of our system, where it will be used to identify hand gestures and movements in live camera feeds and

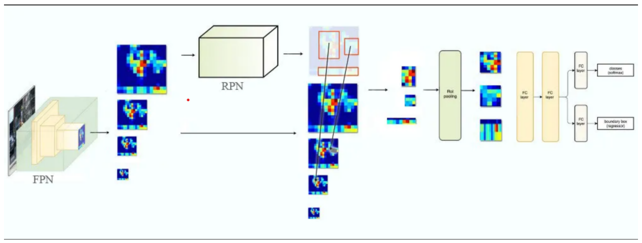

Figure 2. Architecture of our object detection model.

control the computer accordingly.

3.3. Client Program

The client program is responsible for mainly two tasks: (1) keep reading the live frame from the RGB camera, and (2) takes in the identified gesture and location from the above object detection model and use it to control the mouse/keyboard. We will use Python along with opencv and pyautoGUI package to implemented the those two tasks. Specifically, we will use opencv to keep reading the live image frame and pass it to our Faster RCNN model. Once the gesture or movement is identified, we will send the keyboard mouse control signal corresponding to this gesture using pyautoGUI package. There are two things worth noting here. First, we will keep track of the gesture and location history in this program, this enable us to better control the frequency we send control signals. Second, the mapping between gesture and control signal can be arbitrary defined by users in a JSON configuration file. Our client program will read the configuration first when started.

The challenges of designing such program is that the labels and bounding boxes generated from the object detection model can be noisy across frame and time.

Identify gesture: as the input labels can be noisy, we cannot execute the action immediately once we detect a certain gesture in one frame. This is because (1) the label might be wrong from the model; (2) the gesture is accidentally presented by the user; (3) the action associated with gesture is just executed and the user is going to cancel the gesture. For the above reasons, we developed the following procedures to confidently and accurately identify the actual gestures presented by the users. Specifically, we keep track of the detection history and once the gesture is *consecutively* presented for approximately 0.9 seconds, we think that gesture is actually presented by the user and we execute the corresponding action. After that we clear the history and collect them again. In this way, there won't be another action being executed in 0.9s after this one.

Identify the movement: for identifying the moving gestures, there is high possibility that the model will detect no object in one of the frame or the label is simply wrong. Therefore, in this case, we cannot enforce that the gesture is *consecutively* presented. In stead, we will collected the history of movement gestures, put it in a list and this list will not be cleared (for previous case, the history will be removed when we detect a new gesture) even we detect nothing or detect a wrong object. Once we have collected enough history (about 1.2s), we use the center of the bounding boxes as the position of hands and compute the movement direction base on this set of positions.

The algorithm works as follows:

1. We identify the starting position $x = (x_0, y_0)$.
2. We identify the largest, smallest x or y coordinates in the history, denoted as $x_{max}, x_{min}, y_{min}, y_{max}$.
3. We compute the movement strength along four direction as (1) $|x_{min} - x_0|$; (2) $|x_{max} - x_0|$; (3) $|y_{min} - y_0|$ and (4) $|y_{max} - y_0|$.
4. the largest one indicated the direction we are moving. For example, if (1) is the largest, this means we are moving towards the left (the direction of negative x axis).

4. Experiments

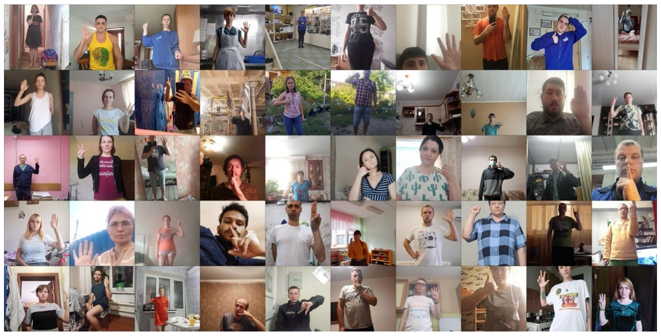

Figure 3. Examples from the HAGRID hand gesture dataset.

4.1. Dataset

In this project, we will use HAnd Gesture Recognition Image (HAGRID) dataset to fine-tune the object detection model. The HAGRID dataset is a large-scale dataset designed to address the need for a comprehensive and diverse dataset for hand gesture recognition tasks. It contains over 552990 RGB images and 18 gesture classes, which include commonly used hand gestures such as "like", "ok", "one" and more. The images in the dataset are FullHD resolution with 1920 x 1080 pixels, providing a high level of detail for the hand gestures. Some examples of this dataset is shown in Figure 3.

One of the notable features of the HAGRID dataset is its diversity. The dataset contains images of different skin

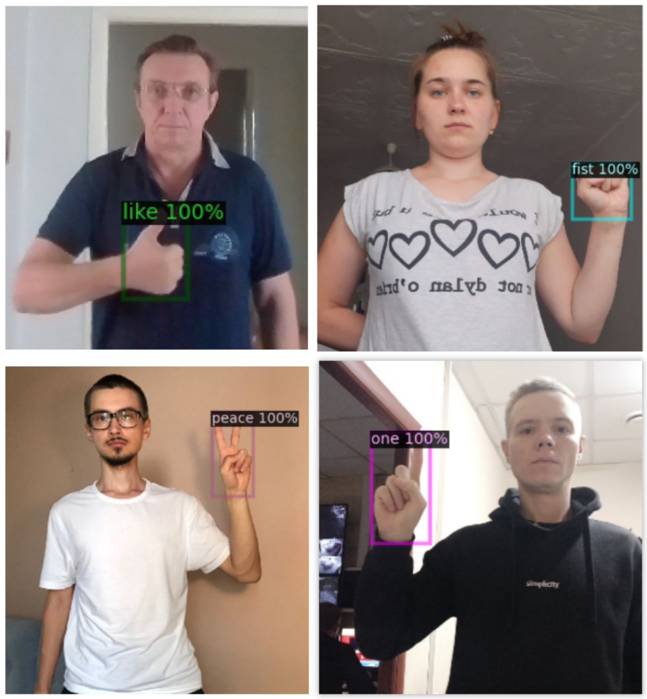

Figure 4. Hand gesture detection results in test set.

tones, hand shapes, and sizes, and are collected from real-life scenes. This is important for developing a model that can generalize to different users. In addition, the dataset contains *images captured from different distances to the camera*, ranging from 50 centimeters to 4 meters. This is a very important property to our task as we would to interact with computer even we are not around it.

As this dataset is very large (adds up to almost 800GB of space) and our computation resource is limited, we pre-process the dataset as follows. First, we choose only 6 categories out of the 18 classes, which are 'stop', 'fist', 'one', 'peace', 'like', and 'ok'. Then, for each of the 6 classes, we randomly pick 1100 images along with their annotations from the full dataset. Note that we choose in a way such that all 1100 images are from different users to maximize the diversity of our dataset while reducing fine-tuning overhead. After that, we spilt our data subset into train and test portion, specifically, we randomly choose 1000 of the images as training split and the rest of 100 images as test split for each of the 6 classes.

At the last step, we convert the annotation format of HAGRID dataset into the COCO format, which enable us to reuse the existing COCODetection dataset module in the torchvision library.

4.2. Evaluation Metric

We can evaluate our system from two aspects. Firstly, we will evaluate our object detection model using (1) Standard

Table 1. Average precision results for our trained object detection model on test dataset.

mAP	AP-50	AP-75	AP-small	AP-medium	AP-large
83.4	97.5	96.7	70.1	79.2	85.7

Table 2. The classification performance of our model.

Detection Rate	Avg. Precision	Avg. Recall	Avg. F1
99.33 %	98%	98%	98%

d object detection metrics including mean Average Precision (mAP) over differnt IoU threshold; (2) standard classification metrics, such as F1 score, to evaluate the ability of the model to correctly identify the gestures inside the bounding box. Secondly, for the whole system, as it is hard to evaluate numerically, we will use human-evaluation and see how well it can interact with computers in real-time. This evaluation will provide us with insight into the practicality and usability of the hand gesture recognition system.

4.3. Fine-tuning Details

We initialized our object detection model directly from the checkpoint `faster_rcnn_R_101_FPN_3x.yaml`, which uses resnet-101 as the backbone structure to extract the feature vector from the input image. As this model is pre-trained on the COCO dataset that has different number of classes when compared to our dataset, we change the classification head in the model with output dimension 7 (6 classes + 1 default no-object class).

We pre-process all images by downsample to width and height to half of the original image size. The detecctron2 framework will also augment our input by resizes the image to the specified size, and pads the image if necessary to make it the exact size needed by the model. It also applies normalization to the pixel values of the image to ensure that they are within a specific range, which is a common step in training deep learning models. Finally, it converts the image to PyTorch tensors.

We fine tune the model using the following configurations. The number of epochs is 128, with a batch size of 8. The learning rate is 2×10^{-5}, and weight decay is 10^{-4}. We use small learning rate and large number of epoches here to ensure the the finetuning process won't diverge from the original weight too much but also gurantee the model are well trained.

4.4. Results

We first evaluate the quality of bounding boxes generated from our model using metric mean Average Precision (mAP), and we show the results in Table. 1.

As we can see that, our model achieves the mAP of 83.4, which means the predicted bounding box position is quite close the truth position. In addition, our model achieves over 95 on the AP-5O and AP-75 metrics. For objects with different size, it is not surprising that our model works better on large objects, achieving 85.7 of AP performance. For smaller objects, the performance is degraded but still acceptable for most cases.

In addition, since the images in our dataset only contains one hand gesture (i.e. one object), we can also evaluate the classification performance of our model. Specifically, we achieved the precision 98% and recall 98% as show in Table 2. This means for objects that are identified, it is correct in almost all cases, which means our model is well trained. In addition, the detection rate of our model is 99.33%, this means that out of all images that have objects, 99.33% of them are being detected.

To better illustrate the performance of our trained model, we visualize some identification results on the test set as shown Figure. 4. From the figure, we can see that the detected label are correct in almost all cases, with high confidence. In addition, the location of the gesture are very accurate.

4.5. Real-time Evaluation

We have also conducted an human-evaluation of our whole system using a webcam in real time. Specifically, we evaluate our system through a simple example of controlling music playing along with the volume level. The demo video can be accessed at https://drive.google.com/file/d/1b2qHICTMi3W8hW2be9iSXtXbFZEKMITk/view?usp=share_link (The recording software records the audio of music even after the system is muted, thus you can still hear sounds in this demo video, but in reality you won't hear anything).

From the video, you can see that our system performs exactly as expected. The various gesture can be identified in high accuracy. In addition, for movement gestures, you can see that the system will fail to track the gesture in some frame, but the overall system can still correctly identify your movement direction and therefore execute the correct action.

5. Conclusion

In this work, we developed a system that can be used to interact with computers (with normal RGB camera) using hand gestures in real time. Specifically, we first trained an object detection model that can accurately identify the gesture presented in the image. After that, we developed a python client program that make use above model and achieve real-time interact with computers. Our experimental results and demo video have shown that our system can work seamlessly when being used for controlling music playing along with the volume.

Note that controlling the music playing is just one of the simple applications of our system. In this work, we mainly focus on keyboard and mouse based actions. However, these pre-defined computer controls can be extended to include any peripheral of the computer that can be issued in python, making it compatible with our project framework.

Our system shares the certain limitations of the standard RGB cameras such as lighting issues or complex backgrounds. These issues can be overcame by using depth cameras, which over the recent years, have evolved to be more advanced and at lowered costs.

References

[1] Santosh K Gaikwad, Bharti W Gawali, and Pravin Yannawar. A review on speech recognition technique. *International Journal of Computer Applications*, 10(3):16–24, 2010. 2

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[3] Alexander Kapitanov, Andrew Makhlyarchuk, and Karina Kvanchiani. Hagrid-hand gesture recognition image dataset. *arXiv preprint arXiv:2206.08219*, 2022. 2

[4] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017. 2

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 2

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 2

[7] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 2