Cursor and Keyboard Control via Hand-Tracking

By

Rushit N. Shah
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607

Course Project Report

Submitted as partial fulfillment of the requirements for the
Computer Vision (CS415) course in Fall 2019 in the
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois

Course Instructors:
Dr. Wei Tang
Nikolaos Agadakos (TA)

Table of Contents

1. Introduction

A Brief History of Computer Vision

One of the most powerful and compelling types of AI is computer vision. Computer vision is the field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labeling objects [1] [2].

Figure 1: YOLO Multi-Object Detection And Classification.

One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better. Along with a tremendous amount of visual data (more than 3 billion images are shared online every day), the computing power required to analyze the data is now accessible. As the field of computer vision has grown with new hardware and algorithms so has the accuracy rates for object identification. In less than a decade, today's systems have reached 99 percent accuracy from 50 percent making them more accurate

than humans at quickly reacting to visual inputs. Early experiments in computer vision started in the 1950s and it was first put to use commercially to distinguish between typed and handwritten text by the 1970s, today the applications for computer vision have grown exponentially. Indeed, by 2022, he computer vision and hardware market is expected to reach $48.6 billion [3].

The Evolution

In its early days , the tasks that computer vision could perform were very limited and required a lot of manual coding and effort by developers and human operators. After all this manual work, the application would finally be able to compare the measurements in the new image with the ones stored in its database and tell you whether it corresponded with any of the profiles it was tracking. In fact, there was very little automation involved and most of the work was being done manually. And the error margin was still large.

Machine learning provided a different approach to solving computer vision problems. With machine learning, developers no longer needed to manually code every single rule into their vision applications. Instead they programmed “features,” smaller applications that could detect specific patterns in images. They then used a statistical learning algorithm such as linear regression, logistic regression, decision trees or support vector machines (SVM) to detect patterns and classify images and detect objects in them.

Deep learning provided a fundamentally different approach to doing machine learning. Deep learning relies on neural networks, a general-purpose function that can solve any problem representable through examples. When you provide a neural network with many labeled examples

of a specific kind of data, it'll be able to extract common patterns between those examples and transform it into a mathematical equation that will help classify future pieces of information.
For instance, creating a facial recognition application with deep learning only requires you to develop or choose a preconstructed algorithm and train it with examples of the faces of the people it must detect. Given enough examples (lots of examples), the neural network will be able to detect faces without further instructions on features or measurements.

Deep learning is a very effective method to do computer vision. In most cases, creating a good deep learning algorithm comes down to gathering a large amount of labeled training data and tuning the parameters such as the type and number of layers of neural networks and training epochs. Compared to previous types of machine learning, deep learning is both easier and faster to develop and deploy.

Most of current computer vision applications such as cancer detection, self-driving cars and facial recognition make use of deep learning. Deep learning and deep neural networks have moved from the conceptual realm into practical applications thanks to availability and advances in hardware and cloud computing resources.

What You Can Do With It?

Computer vision is more than just the recognition of objects in images. The area is extremely rich in the problems available at hand to solve. Some of these problems are

- Image segmentation

- Object detection
- Facial recognition
- Edge detection
- Pattern detection
- Image classification
- Feature matching

While these problems by themselves might tackle only a small problem, their solutions, when used in conjunction, enable the development of very rich applications with a massive variety of applications and potential.

2. Problem Statement

Goals

As highlighted in the previous section, computer vision has a wide variety of applications in everyday life. Some of these applications may be advanced and quite futuristic, for example the problem of detecting a multitude of objects by a self-driving car; other applications may be more humanitarian in nature, for example a device to narrate to a partially-blind person the object he/she was looking at or pointing towards.

In choosing the problem statement for this study, it was desired that the choice be a good combination of both aforementioned classes of applications—one that not only reflects the current and future state of the field of computer vision, but also has some potential humanitarian benefit. Consequently, the problem statement of this study is framed as follows

"To enable the movement of the cursor on-screen via tracking the user's hand position. Additionally, it is also desired to enable some keyboard operations via such hand tracking."

Basically, this project involves the user moving their hands in front of a camera, which detects and maps those movements to mouse positions on-screen. Further, this hand-tracking functionality is also leveraged to allow the user to press certain keys via hand-positions. This project is meant to serve as a prototype for applications in the future, which are built to be more robust, and tailored to the needs of specific individuals.

Breakdown

The task of controlling the cursor/keyboard can be broken down into two major phases/parts. The first phase involves the actual detection of the hand using the webcam and tracking it, followed by the second phase which involves mapping the hand positions to actual cursor/keyboard operations. The following sections detail the steps involved in each of these phases, the approaches used, and the challenges involved in the successful implementation. Further details can be found in the respective chapters.

3. Phase 1: Hand Tracking

The first phase, the hand-tracking, forms the crux of the project and makes-up ~70% of the actual work involved in the successful implementation of the tasks outlined in the problem statement. Specifically, this involves detecting the outline and the position of any hand(s) in the frame. The successful implementation of this step plays a major role in the quality of the final output (the cursor movement/keyboard button presses). This is primarily due to the fact that if the algorithm used cannot identify the outline/position of the hands in a given frame, then this introduces latency in the pipeline leading to undesirable/jerky cursor movements or button presses. The process of hand-tracking in a live video can be further broken down into the following two steps

- Retrieval of the image frame from the live video stream
- Treating the frame as a single image to retrieve the outlines/contours of any hands in the retrieved image frame.

The first of these steps—the retrieval of image frames from the live video stream—is trivial since this enabled by computer vision libraries such as OpenCV. A code snippet to do this is shown below

```
vid  = cv2.VideoCapture(0)
while True:
    ret, frame = vid.read()
```

This loop is executed continuously until the program is running. The frame captured in each iteration is displayed in a window and acts as a live-streaming video.

The second step—that of hand-tracking—as stated before is more challenging of the two. There are several approaches to this part, but one must be cautious; an easier approach is not necessarily better since incorrect hand detections undoubtedly result in lagged cursor movement. On the other

hand, an approach that is extremely good at detecting hands but makes use of a complex model is not desirable either, since the higher detection times would also introduce latency in the pipeline and lead to lagged cursor movements.

Following the evolution of the field of computer vision described in the previous section, several approaches were explored for the task of hand detection in this project; these methods ranged in their complexity from very rudimentary to significantly more complex. The detailed design of these approaches is described in the following subsections.

a. Using Color Markers

The simplest approach to detecting any objects is to use specific color markers on the object of interest to make for easy detection. An example of this is shown in Figure 2 below. In this example, green paper rings were first worn around the thumb and index fingers. Each frame received from the video stream was then colored to HSV color space. Then using an upper and lower threshold for the green color in the HSV color space, the regions of interest in each frame were segmented out. The thresholds used are shown below

```
lower_threshold = np.array([33,80,40])
upper_threshold = np.array([102,255,255])
```

This process resulted in two contours being obtained—one corresponding to each of the green markers. The position of centroids of the contours thus obtained, and the position of the midpoint (red dot) of those centroids in the frame then had the potential to be mapped to a position on the screen for the cursor. Thus, as the midpoint moved with the hand position, so would the cursor.

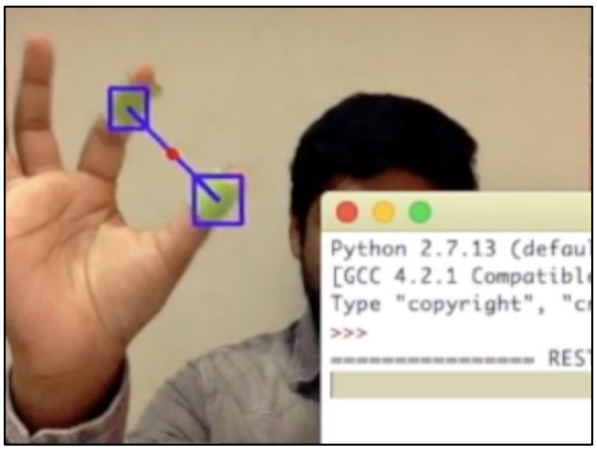

Figure 2: Example of the use of green finger markers to localize position of interest in the frame (red dot), eventually mapped to a mouse position on the screen.

While this approach was extremely straightforward, it suffered from its own set of drawbacks. First, this approach required the use of additional accessories to work at all, and this left much to be desired; after all, an elegant approach would be one that would work even in the absence of such accessories. Also, on the technical side, this approach left much to be desired in terms of its applicability in different circumstance. In other words, it was found to be sensitive to the specific color being sought—thus if the background or any other object in the frame was a shade of green, it would render this approach completely useless. This approach was also found to not work too well under different lighting conditions.

b. Skin Segmentation using Color Thresholds

Given the aforementioned drawbacks of the color marker approach, it was concluded that the approach finally used would need to be independent of any additional accessories. Thus it was decided that the detection of the hand, rather than color markers, would be pursued.

To this end, as a first step, color thresholds were applied. Following from the color marker approach, color threshold were set to segment out the skin in the image frame, rather than color

markers. With the difference being that rather than using multiple markers, the centroid of the hand thus detected would be used to track the cursor position on the screen.

Upon actual implementation, this approach was found to perform even worse than the color marker approach. This was mainly due to the fact that simple thresholding of the skin color yielded widely varying results in even marginally different lighting conditions. Also, as the HSV thresholds were set to be less sensitive to lighting conditions, in most actual experiments, the actual background was detected as part of the skin, since that particular color of skin is easy to find in most frames and backgrounds even in the absence of actual skin.

c. Skin Segmentation using Gaussian Color Model

Figure 3: Example of the issues faced when using skin segmentation using a Gaussian color model.

Following the unsuccessful use of color thresholds for skin segmentation, it was hypothesized that potentially using a more sophisticated model for skin segmentation would yield better results. Consequently, using a hand gesture recognition data set [4] [5] [6], a Gaussian color model was trained. Using a minimum probability threshold on the HSV value of a color in the model, the skin was segmented out.

Upon implementation, This approach worked better in terms of detection than the previous two color threshold-based approaches. An example of what the result looked like is shown in Figure 3 above.

This approach was particularly interesting since the implementation of Gaussian color models used here was the one that was implemented for one of the homework assignments for this course. Unfortunately, it is for this same reason that this approach did not work out too well—the implementation was pixel-level. The algorithm would visit each pixel in the frame inside a double `for-loop` and classify it as skin/non-skin which made the implementation prohibitively slow. While this approach would work well on single images, it was extremely slow for segmenting skin in a video stream. Another drawback of this approach was that due to the lack of use of libraries, it was not particularly robust and failed to detect sudden changes in skin tone—making the overall approach fail in regions of shadows and darker skin tone. An example of the type of discontinuities encountered are shown in Figure 3 above.

d. Using Convolutional Neural Networks

With most other approaches exhausted, deep neural networks were resorted to. Indeed, in the age of deep learning, it is hard for any approach to compete with the power of a deep convolutional neural network (CNN) [7] [8]. Their expressive power is unparalleled, and so is generalization power—indeed a well-trained CNN is invariant to various geometrical transformations such as scaling and rotation.

However, the explosion of the research area of deep learning has led to the availability of a multitude of different CNN architectures (AlexNet [9], GoogLeNet [10], Faster R-CNN [11]), with each architecture reporting cutting-edge performance. With so many architectures

available at one's disposal, one must choose an architecture based on his/her specific requirements.

The goal of this project was the fast detection of hand(s) in incoming frames from a video stream—the speed was a priority. With this in mind, a CNN which would allow the quickest classification was sought. Tensorflow's model zoo, provides several different architectures already implemented out-of-the-box. These models are pre-trained on Microsoft's COCO dataset [12], and have their detection speeds reported.

Based, on the detection speeds, the fastest available detection model was chosen—the ssd_mobilenet_v1_coco with a reported speed of 30ms. This architecture is a Single-shot Multi Box Detector [13] with MobileNet [14] as its base architecture. A detailed explanation of this architecture may be found in [15].

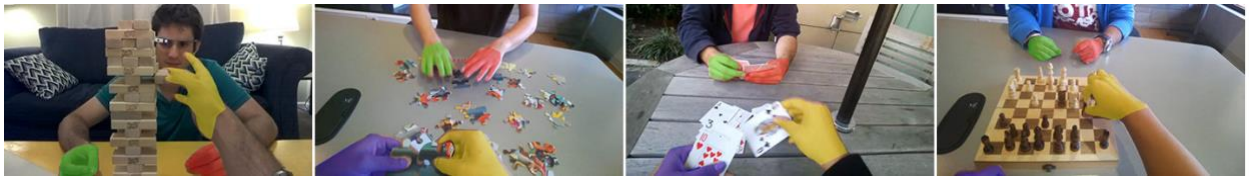

Figure 4: Example images from the EgoHands [16] dataset.

Since the Tensorflow implementation of the model is pre-trained on the COCO dataset, the next step involved re-training the pre-trained model on a more relevant dataset which would enable detection of hands by the process of transfer learning [17]. To this end, the EgoHands dataset [16] was chosen. This dataset contains 4800 Google Glass images of complex, first-person interactions between two people. The images span 48 different environments (indoor, outdoor) and activities (playing cards, chess, jenga, solving puzzles etc).

Having retrained the *ssd_mobilenet_v1_coco* architecture using the EgoHands data set, the results were indeed significantly better than any of the previous approaches. The hand detection approach performed impeccably under almost every different lighting condition it was tested in. It was also able to seamlessly detect multiple hands in a given frame. However, the maximum number of hands to detect in every frame was set to two—this was done in order to prevent false detections misleading the algorithm.

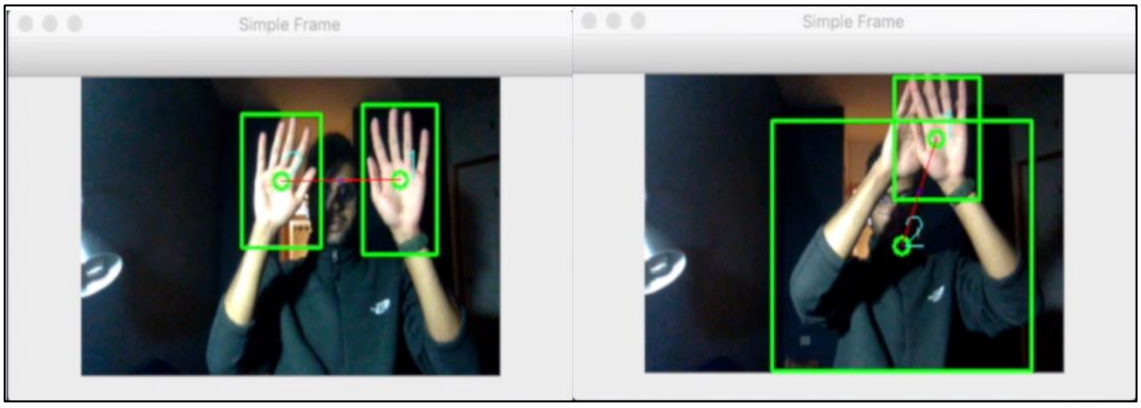

Figure 5: Example of hand detection using CNNs. Left-successful detection of two hands. Right-example of false hand detection.

Bounding boxes were constructed around the detected hands, and the centroids of the bounding boxes, and the midpoints of the centroids on the frame was then used to determine the position of the mouse on the screen. Examples are shown in Figure 5. However, there was an occasional misclassified hand when the two hands in the frame were brought very close together as seen in Figure 5 (right)—this may be attributed to issues raised by occlusion of objects. But overall, the performance of the CNNs was commendable and was chosen as the final approach for hand detection.

4. Phase 2: Mapping Hand Positions to Inputs

With a reliable method for hand detection and localization in place with the successful implementation of CNNs, the next phase of project pipeline involved converting various hand positions to inputs to the mouse and keyboard. This task was made fairly straightforward with the availability of Pynput library [18]. This Python library allows the creation of mouse/keyboard objects via which the respective devices can be controlled. A code snippet is shown in Figure 6 below.

```
from pynput.mouse import Button, Controller
mouse = Controller()
mouse.press(Button.left)
mouse.position=(xPos,yPos)
```

Figure 6: An example of how the mouse can be controlled using the Pynput Python library.

This phase of the project can be further divided into two parts

- Cursor Control
- Keyboard Control

Each of these will be discussed in some detail in the next subsections.

a. Cursor Control

Implementing the full functionality of a mouse was beyond the scope of this project—the main goal was prototyping. To this end, a few main mouse functionalities to be enabled via hand detection were laid out as objectives. These were

- Cursor movement
- Single (left) click
- Combination of the above two (left click + movement)

Of these, the cursor movement was dictated by the position of the midpoint of the centroid of the bounding boxes of the detected hands—the cursor could thus be moved by holding two hands in the frame and then moving them in conjunction.

A single left click was registered when the number of hands detected went from two to one i.e. the hands were brought close together and one was shielded by the other as shown in Figure 5 (right). Lastly, if the number of hands detected remained one over consecutive frames, the left click would be prolonged—until the detected number of hands went from one to two again i.e. the hands were separated.

b. Keyboard Control

Like the mouse, implementing every key press on a keyboard using just two detected hands was beyond the scope of this project. Instead, for the purpose of prototyping, three main button presses were laid out as objectives to be achieved via different combinations of hand patterns. These were as follows

- Left arrow key
- Right arrow key
- Space bar

The prototyping objective in this study was to successfully play the popular Atari game SpaceInvaders to demonstrate the developed methodology—this would require using the aforementioned three buttons.

The left and right arrow key presses were enabled by the position of a single hand in the frame—these would be activated when only one hand was detected on screen. Which of the two was pressed was determined by a simple rule—a hand detected in the left 30% of the frame

Figure 7: Examples of detected hands, and corresponding key activations annotated on the frame.

would activate the left arrow key, whereas a hand detected in the right 30% of the frame would activate the right arrow key. Any hand detection in the middle 30-70% of the frame would result in no click. The space bar was activated by the detection of two hands at *any* position in the frame. Some examples of these detections and key activations are shown in Figure 7.

5. Results

While it is obviously not possible to show the results of this work in pictures, some results have been included as images in Figure 5 and Figure 7. However, full-length demonstrations of both, the cursor and keyboard control, functionalities are available on YouTube at the following links

Cursor Control: https://youtu.be/WkfXd6S-njY

Keyboard Control: https://youtu.be/e6Cu1omx6Y0

6. Analysis and Future Work

While this projects meets the goals that had been set, it is only a prototype of a fully-functional system and a lot more functions can potentially be included to extend this work. But even within the scope of the current work, there is room for a lot of improvement. Specific improvements that can be made will be covered now in three parts—hand detection, cursor control, and keyboard control respectively.

Hand Detection: While the current CNN model performs extremely well in terms of hand detection, compared to the other three approaches tried, there is still room for improvement. The current Keyboard control relies on simple rules which detect a hand in different parts of the screen and lead to key activations. While this works, it is a very rudimentary solution. A more sophisticated solution would involve the model being able to tell the left and right hands apart from each other and determine key presses accordingly. However, this approach is significantly more complex given that it would be extremely difficult for CNNs to understand the hand orientation, given that they are specifically tailored to be invariant to such spatial transformations.

Cursor Control: The current cursor movement is only satisfactory, as evidenced by the demonstration video—the movement is not smooth enough for any practical use. The inclusion of a damping mechanism to the cursor movement would greatly smooth this movement—such a mechanism can be compared to the sensitivity control available out-of-the-box on all modern computers. This would potentially moving the cursor only by a certain magnitude every frame. In other words, even if the user moved their hands by a large distance between given frames, the cursor would actually only move by a certain percentage of that movement. Specific damping factors would have to be experimented with to arrive at a number that works well—much like hyperparameter tuning.

Keyboard Control: While the keyboard control functions much smoother than the cursor control, it is very rudimentary in its overall functionality. The keyboard offers a wide variety of keys and their combinations, but not all of those can be mapped to unique hand positions. However, by changing the hand detection model to a gesture recognition model, it is certainly possible to cover a larger number of keys than being done at present. For example, it is actually not very difficult to build a hand detection model, which, based on the deformities in the convex hull of the detected hand contour can judge how many fingers a user is showing in an image; these detections can then directly be mapped to the numeric keys on the keyboard. Going further, models trained to recognize specific gestures can even be used to type out text—this can be extremely useful for handicapped persons who cannot leverage the speech-to-text functionality on most modern devices.

7. Key Takeaways and Course Feedback

The course CS415 covered a wide variety of topics from the field of Computer Vision in a well-structured manner. The topics covered over the course of the semester included introductory concepts of image understanding such as light, geometry and cameras. This was followed by basic concepts of image manipulations such as geometric transformations and filtering. As the course progressed more sophisticated concepts of edge/line/corner detection, feature extraction/matching, were covered. Having learnt how to extract fractures from images, image classification, recognition, and detection were taught.

What stood out in this course was the course structure, and delivery. The concepts were made easy to understand, and the order in which the various topics were covered provided a sense of continuity between topic. This was found to be extremely useful in understanding the overall structure of the field of computer vision—indeed, such a high-level picture of the field helps students situate themselves within in, which consequently makes approaching specific research topics easier. For example, the computer vision application which is most relevant to my own research endeavors is that of human pose estimation; the logical structure of the course benefitted me by allowing me to narrow my focus to the specific areas closest to my area of interest—recognition, detection, and action recognition.

In summary, the following aspects of the course were found to be very beneficial and should, in my humble opinion, not change across semesters

- The organization of the course

- **The mode of delivery of lectures—slides combined with derivations being shown LIVE on the document camera**
- The grading format—six manageable machine problems of appropriate difficulty and a semester-long project. The machine problems were extremely useful in providing insights into the actual operation of the learnt algorithms.

References

[1] A. Hern, "Computers now better than humans at recognising and sorting images," 13tht May 2015. [Online]. Available: https://www.theguardian.com/global/2015/may/13/baidu-minwa-supercomputer-better-than-humans-recognising-images. [Accessed 9th December 2019].

[2] A. Sergeenkov, "Artificial Intelligence is Becoming Better than Human Expertise," 28th February 2019. [Online]. Available: https://hackernoon.com/artificial-intelligence-is-becoming-better-than-human-expertise-16903f4fc3c0. [Accessed 9tht December 2019].

[3] I. Mihajlovic, "Everything You Ever Wanted To Know About Computer Vision.," 25th April 2019. [Online]. Available: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e. [Accessed 9th December 2019].

[4] "Database for Hand Gesture Recognition," [Online]. Available: http://sun.aei.polsl.pl/~mkawulok/gestures/.

[5] M. Kawulok, J. Kawulok, J. Nalepa and B. Smolka, "Self-adaptive algorithm for segmenting skin regions," *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 170, 2014.

[6] T. Grzejszczak, M. Kawulok and G. A., "“ Hand landmarks detection and localization in color images,," *Multimedia Tools and Applications*, vol. 75, no. 23, pp. 16363-16387, 2016.

[7] H. W. Lin, M. Tegmark and D. Rolnick, "Why does deep and cheap learning work so well?," *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223-1247, 2017.

[8] M. Kana, "Why Deep Learning Works: solving a farmer’s problem," 18th July 2019. [Online]. Available: https://towardsdatascience.com/why-deep-learning-works-289f17cab01a.

[9] A. Krizhevsky, I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks.," in *Advances in Neural Information Processing Systems*, 2012.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[11] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards eal-time object detection with region proposal networks.," in *Advances in Neural Information Processing Systems*, 2015.

[12] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. Zitnick, "Microsoft coco: Common objects in context.," in *European conference on computer vision* , 2014.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu and A. C. Berg, "SSD: Single shot multibox detector.," in *European Conference on Computer Vision*, 2016.

[14] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications.," *arXiv preprint arXiv:1704.04861*, 2017.

[15] H. Gao, "Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch," 6th June 2018. [Online]. Available: https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad.

[16] S. Bambach, S. Lee, D. J. Crandall and C. Yu, "Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions.," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[17] S. J. Pan and Q. Yang, "A survey on transfer learning.," *IEEE Transactions on Knowledge and Data Engineering.*, vol. 22, no. 10, pp. 1345-1359, 2009.