# Conjure: A Computer Vision Controller Using Hand Gestures

Anthony Hunt

April 20, 2025

## 1 Introduction

Over the last 20 years, extraordinary progress in image processing for machine learning has brought about revolutionary methods of interacting with the world. From self-driving cars to early cancer detection, computer vision has become integral to interactions with technology and our environments. The usefulness of such software is self-evident in its pervasiveness within smartphones and personal computing; home buttons have been replaced with retina scanners and facial recognition, VR headsets no longer need dedicated controllers, and any device can read handwritten text directly from a photo.

In the gradual rollout of virtual hands-free computing, gesture-based interactions serve as a natural and intuitive platform for communicating with computers. This project, named Conjure, aims to take gesture-based interactions one step further by providing a human-computer interface with only a camera and hand recognition software. However, unlike the Xbox Kinect or Apple Vision Pro headset, which make use of depth sensors in addition to regular cameras, we attempt to use only a standard camera. Of course, the use case of Conjure is far simpler than that of VR headsets or game consoles, since the target platform of this program is user-facing laptop webcams. We can further assume that most people interact with webcams in an egocentric view, that is, facing the camera with one prominent subject and a mostly static background.

The rest of this report will outline the features of Conjure along with instructions to get started, the model architectures explored and used within the program, some key results, and difficulties throughout the project.

## 2 Usage

To get started, clone the GitHub repository and download all dependencies through `pip install -r requirements.txt`. Ensure the computer's webcam is plugged in and working, then run `python main.py`. Note that this project was tested with Python 3.12 on a Windows machine.

Upon startup, a GUI containing settings and other options should appear similar to Figure 1.

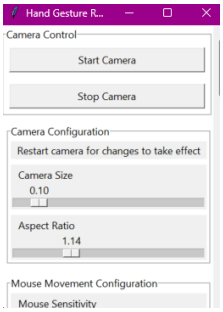



Figure 1: Startup configuration UI. Click the "Start/Stop Camera" button to activate the virtual controller.

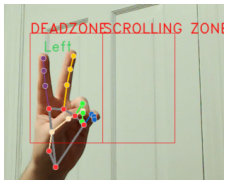
The main interactions of Conjure are as follows, with coinciding images in Figure 2:

- Hand movement in the camera's 2D projection of 3D space coincides with mouse movement across a computer's monitor. For fine-grained movements and other interactions, Conjure creates a deadzone at the center of the screen, where all hand movements are ignored. Then, as a hand moves out of the deadzone toward the borders of the screen, the mouse will move with increasing velocity in the hand's general direction.
- Simulating a left click can be done by making an "OK" gesture towards the camera. This gesture is closest to the natural pinch gestures of VR headsets while being freely available in many datasets [1, 2, 5, 10].
- A right click conversely uses a "peace" sign, with the index and middle fingers extended and pointing away from each other.
- A click-and-hold motion makes use of a closing fist gesture. As long as the fist remains closed, the mouse will continue holding down the button. Moving around the camera space while maintaining a closed fist allows for drag-and-drop behaviours.
- Exiting the program is done with a "reverse-stop" motion, where the hand is fully extended, fingers close to one another, and the user's palm is facing away from the camera.
- Scrolling coincides with a two-finger-raised gesture, with up and down behaviour following the palm's direction - up when facing towards and down when facing away from the camera respectively. From rudimentary testing, scrolling behaviours worked best if used only in a specific area of the screen. Thus, this gesture only works within the scroll zone (by default, this is the right side of the mouse deadzone).

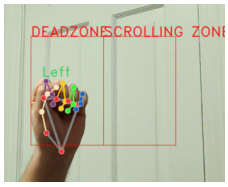
All gesture-specific actions may be changed with the GUI configuration, made in Tkinter. Other options, like movement sensitivity, deadzone sizes, etc., are also configurable through the UI.


(a) The left deadzone prevents hand movement within that area from controlling the mouse, however, any actionable gestures will still work. The right scroll zone prevents mouse movement but enables scrolling movement. Pointing two fingers up when the index finger is within this zone will perform a scroll-up action.

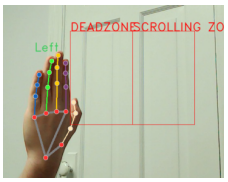

(b) OK symbol for left click.



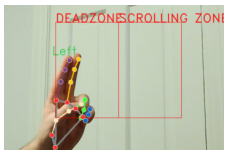

(c) Peace symbol for right click.




(d) Clenching a hand will mirror click-and-hold behaviour.




(e) A stop symbol with the palm facing away from the camera will exit the camera control program.




(f) Two fingers pointing up will scroll up (only in the right side scroll zone).


(g) Showing the back of the scroll-up gesture will scroll down (only in the right side scroll zone).

Figure 2: A collection of gestures described in section 2.

# 3 Model and Method

When looking for similar projects in hand recognition and gesture detection, Google's Mediapipe suite of models [2] is among the first and most performant entries. A sizeable portion of this project uses the Mediapipe hand landmarking model to track hands in 3D space, along with a fine-tuned version of their gesture classification model retrained on the HaGRID dataset [1]. Mediapipe also provides convenient functions for projecting hand skeletons on the webcam's video feed (as seen in Figure 2).

Originally, I planned to create my own CNN model from scratch (as seen in the `model/old_attempts` folder), training on the Rendered Handpose Dataset [10]. This process would have included a CNN landmarking model to project key hand joints in 2D space alongside a second fully-connected network to classify the points into useful gestures. However, a combination of hardware limitations, data preprocessing, and overly simple model architecture failed to capture the objective hand location data, resulting in extreme overfitting when compared with a validation set.

After spending a few days investigating the poor performance of my own model and researching CNNs for object detection, I came across an old Mediapipe blog post [3] detailing a data pipeline for the entire hand recognition process. With a bit of simplification, the general process of classification is as follows:

1. A Single Shot Detector (SSD) model [4] is first used to recognize the bounding boxes of palms within the image. Generally, SSD models are specialized CNNs without final fully-connected layers that make use of bounding box anchors to simultaneously predict the location and classification of objects. Training data passed to this model must be prepared with hundreds of anchor points that enable the model to pinpoint the exact contributions of anchors to the object's location. Although data preparation is intense, SSDs generally perform far better and more efficiently than CNN counterparts, contributing to the feasibility of real-time livestream hand detection.

2. After the general location of a hand has been processed through the SSD, the original image is cropped to relevant regions and passed through a regular CNN. By virtually eliminating all non-hand objects from the image, this model can focus on finding landmarks within the hand, rather than finding the hand within an entire image. Key points can then be converted back into the original image's coordinates and prepared for classification. The current version of the gesture recognizer [2] provides further computational benefits in tracking hand motion over multiple frames when set to "video" or "livestream" mode.

3. From the relative locations of landmarks, Mediapipe uses a simple fully-connected model to finally classify the gesture.

Although I have some experience making CNNs in an academic setting, creating a full SSD model from scratch fell far beyond the scope of this project, especially with limited time, processing power, and overall lack of familiarity in manipulating image data. An alternative scheme wherein a pre-made SSD model is used in conjunction with custom models for stages two and three of the pipeline would have been manageable, however there were understandably few resources on the topic. Mediapipe only offers end-to-end complete models through their API and GitHub, and the only similar project [9] contained several dependency-related issues between tensorflow v1 and v2, preventing access to model weights in a usable form. Because Mediapipe offered a complete, highly performant and customizable classification pipeline, I opted to use their API and architecture, only finetuning hyperparameters and providing training data for the third-stage model.

Therefore, instead of creating deep learning models from source, this project takes a more application-focused approach to machine learning. While imperfect, the live video feed, GUI, and extensive configuration all attempt to make the project more usable for its end goal: providing a virtual trackpad with only camera input.

# 4 Results

The retrained stage three classifier model performed well fairly quickly, only requiring 20 epochs of training on a 30,000-image dataset [1] to achieve 90% accuracy. The classifier model contained two relatively small units of fully connected layers, performing best with a learning rate of 0.005 and dropout rate of 0.1. Steady improvements in both training and validation over 50 epochs can be seen in Figure 3.

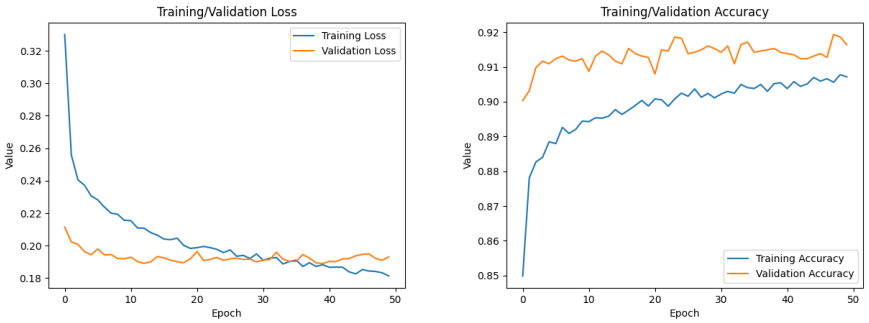



Figure 3: Results of training and validation of the stage three classifier model on 30,000 samples of the HaGRID [1] dataset.

On the interaction side of Conjure, the application is separated into two parts. First, the main GUI controls and updates all configuration changes in

real-time. Sliders and dropdown menus are provided to change deadzone properties, gestures, and other similar settings. Then, two buttons control the camera livestream component, which performs classification predictions and gesture-to-action mouse movement.

In the initial stages of testing, the classifier model, although highly accurate for still images, sometimes had trouble identifying frames involving rapid swaps between two similar gestures. For example, a "peace" sign may be mistaken for a "two-fingers-up" gesture when transitioning from "peace" to a closed fist, and vice versa. This could introduce undesired scrolling or clicking behaviours, reducing the reliability of computer control. However, requiring around three consecutive frames of input before performing an action seemed to resolve a large amount of such cases. Explicit landmark calculations for identifying the front/back of hands also helped to resolve differences in, for example, identifying a left-hand's stop versus a right-hand's inverted stop gestures.

# 5 Conclusion

While CNNs are a mature concept in the fast-paced field of machine learning, modern interpretations on convolutions may continue to increase the usefulness and viability of computer vision in ever-critical applications. Among the burst of popularity of generative text-based models and image creation, computer vision in particular shows great promise in shaping a more accessible and convenient future. For instance, projects like this one could be easily extended with gestures to enable speech-to-text input or simplified video game controls for those with limited finger mobility, without the need for dedicated hardware. Although the process of making Conjure did not follow the original plan exactly, the final product is more than satisfactory in providing users with a proof-of-concept virtual trackpad interface.

# References


[1] Kapitanov Alexander, Kvanchiani Karina, Nagaev Alexander, Kraynov Roman, and Makhliarchuk Andrei. "HaGRID - HAnd Gesture Recognition Image Dataset". In: *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2024. DOI: 10.1109/wacv57701.2024.00451.

[2] *Gesture recognition task guide*. 2025. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer (visited on 04/19/2025).

[3] *Hands parent: MediaPipe Legacy Solutions*. URL: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html (visited on 04/19/2025).

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-3-319-46448-0_2.

[5] Anton Nuzhdin, Alexander Nagaev, Alexander Sautin, Alexander Kapitanov, and Karina Kvanchiani. *HaGRIDv2: 1M Images for Static and Dynamic Hand Gesture Recognition*. 2024. arXiv: 2412.01508 [cs.CV]. URL: https://arxiv.org/abs/2412.01508.

[6] *OpenCV*. 2025. URL: https://opencv.org/ (visited on 04/19/2025).

[7] *PyAutoGUI*. 2019. URL: https://pyautogui.readthedocs.io/en/latest/ (visited on 04/19/2025).

[8] Usman Rizwan. *Implementing a Single Shot Detector Model in TensorFlow 2.0*. 2025. URL: https://usmanr149.github.io/urmlblog/computer%20vision/2022/09/10/Implenting-SSD-TF2.html (visited on 04/19/2025).

[9] Dibia Victor. "HandTrack: A Library For Prototyping Real-time Hand TrackingInterfaces using Convolutional Neural Networks". In: *GitHub repository* (2017). URL: https://github.com/victordibia/handtracking/tree/master/docs/handtrack.pdf.

[10] Christian Zimmermann and Thomas Brox. *Learning to Estimate 3D Hand Pose from Single RGB Images*. 2017. arXiv: 1705.01389 [cs.CV]. URL: https://arxiv.org/abs/1705.01389.