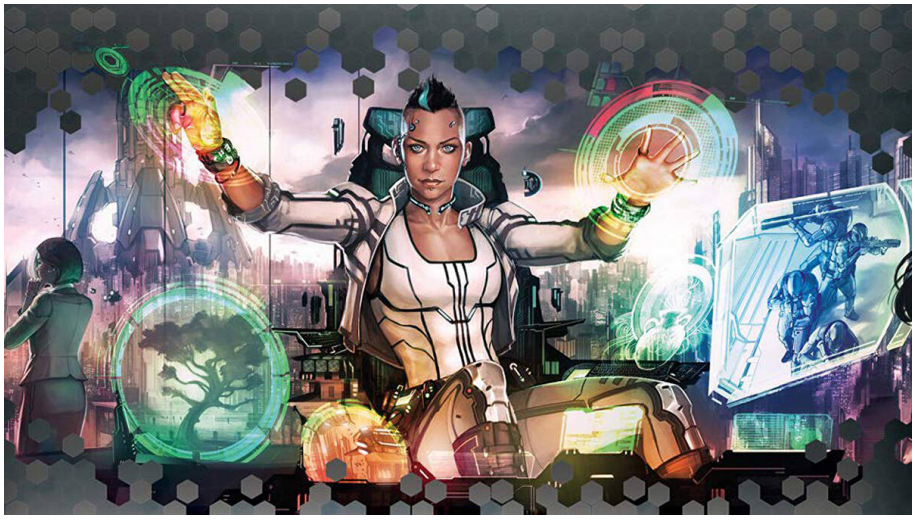
Team Member Names

Benjamin Hendel

Project Title

"Wave your hand to control your device"



Problem Statement

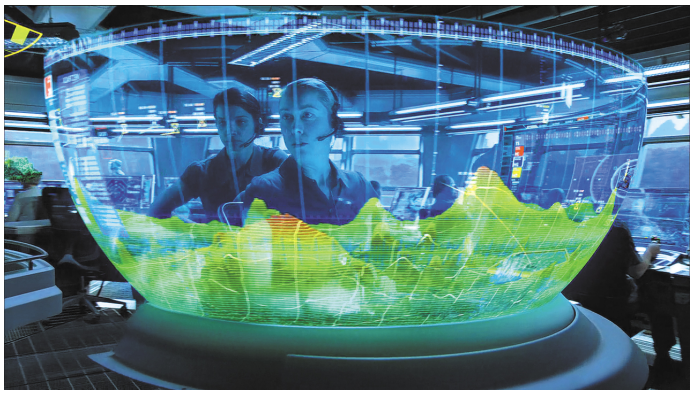
Can you build useful software that automates part of your routine on your computer by recognizing hand-waving inputs through your webcam?

Introduction

Science-fiction cinema teases us with flashy 3-dimensional computer interfaces, holographic blue screens where hacker geniuses wave their hands around and execute complex instructions. Are these merely impractical Hollywood trickery or could hand gestures in front of a screen actually be more convenient than the old mouse and keyboard? Believe it or not, the technology for recognizing hand gestures in the air has been around for some time, and recent advances in machine learning have made it easy for us to give it a crack. This kind of project

has been done before, but for whatever reason we do not see it in use, meaning there is room for improvement.

The goal of this project is to use computer vision and Convolutional Neural Networks to create useful automation software through hand gestures. Video capturing a sequence of images will use pattern detection and images represented as vectors to track fingers and classify hand gestures. This project will also connect them to modern popular automation software such as Keyboard Maestro, in order to vastly expand the range of what these gestures have been able to achieve.

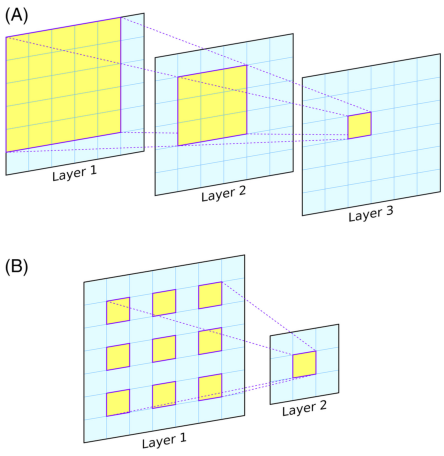


Source: 99%invisible.com

Since this is an existing problem that has been tackled many different times, I used pre-trained models instead of training myself. The training of these models was done on Kaggle's gesture recognition datasets using far more powerful computing power than my local machine. The tricky part was incorporating different sources together in code that can achieve the gestures and applications I wanted to test, with the same kinds of output.

The computer's webcam inputs each frame as an RGB image, from which it detects hands, locates their boundaries, and segments them into 20 important points based on the pose. These 20 points are fed into the Convolutional Neural Network model each frame.

A Convolutional Neural Network differs from a traditional neural network in that it has filters that group together nearby values (usually pixel intensities) which then pool down into fewer values that represent neighborhood effects such as edges.



Visualization of convolutional layers

The model itself is a CNN with three 3D layers of increasing sizes (starting at 16, one per frame and expanding then contracting) with pooling and normalization layers and Relu activation. This is pretty standard for CNNs, the pooling layers bring down all the large parameters into smaller sizes and the normalization/activation lets the model make complex conditions that go beyond simple linear combinations. The model then flattens into a basic 2D layer before a final layer uses a Softmax activation that lets the model "trigger" a positive or negative gesture if the final score is high enough.

After the model recognizes a gesture, a corresponding trigger is called and signaled to automation software known as Keyboard Maestro. With enough tuning, Keyboard Maestro can automate tasks, call parts of your OS, and accelerate a workflow (https://www.keyboardmaestro.com/). For whatever Keyboard Maestro couldn't trigger, I used AppleScript called by the python package osascript (https://pypi.org/project/osascript/). I was able to change brightness, volume, switch applications, send emails and texts, and much more.

```
result = osascript.osascript('get volume settings')
volInfo = result[1].split(',')
outputVol = volInfo[0].replace('output volume:', '')
if outputVol == 0:
    osascript.osascript('set theDialogText to "Volume is already at the minimum!"')
    osascript.osascript("display alert theDialogText with icon caution")
else:
    osascript.osascript("set volume output volume {}".format(int(outputVol) - 60))

os.system("""osascript -e 'tell application "Keyboard Maestro Engine" to do script "Spotify Macro 2"'""")
```

Example code that decreases volume, displays pop-up message if at minimum, opens music

Issues

What follows is a breakdown of the whole product chain and all the issues encountered, from camera input to machine learning to gesture incorporation into daily use and a conclusion on feasibility and future steps.

Camera Input: Mediapipe vs OpenCV's VideoCapture: I used a combination of existing OpenCV and Mediapipe frameworks. Mediapipe is Google's open-source platform for handling media input. Mediapipe was the clear winner here and resulted in higher accuracy when it was used, at the cost of much more computing power.

Lighting: Was a pain, I tested in multiple environments, the slightest unevenness caused complete chaos. Light from bulbs anywhere near the screen was mayhem, but it worked best with most lights off and natural light flooding in. Any further progress was made when light and motion was good and stable.

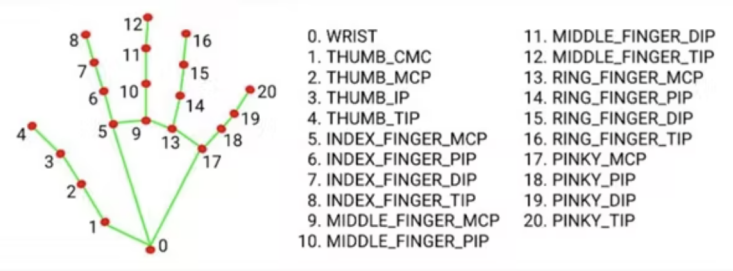
Motion: Moving my chair made me restart several times, someone moving behind me changed every value and it had to recalibrate. Any kind of swiveling my laptop screen and showing a different part of the room that changes hand angle and lighting was bad news, and any angle that wasn't head-on to the screen resulted in many false negatives, as hand gestures look completely different when viewed from the side. Like mentioned before, any further progress was made when light and motion was good and stable. I still have yet to see demos of this technology in crowded outdoor environments, and I think I know why.

Users: It was irrelevant whether the user was me or friends, family. In order to be wise about product accessibility, it may be worth testing with other skin tones and ages.

Webcams: It performed slightly better on 2021 Macbook Pro webcam vs a budget 2019 Windows HP Laptop, though it was hard to isolate and quantify. I expected that a higher megapixel camera would perform much better than it did, my theory is that the more detailed image had more room to get lost in small details.

Detection: The first step of the machine learning process: Is there a hand in the image or not? In my experience, once good and stable lighting was achieved, OpenCV hand detection worked around 95% of the time with minimal flicker. I imagine this is a fairly simple task given the unique shape of a hand in an image mask.

Segmentation: After a hand is detected, where is it? A bounding box was placed around the borders, fingers were identified, and joint locations are usually located depending on the task, in order to reduce the data for gestures into just 20 locations. This is the classic segmentation used in most applications.



Source:https://www.google.com/url?sa=i&url=https%3A%2F%2Fgoogle.github.io%2Fmediapipe%2Fsolutions%2Fhands.html&psig=AOvVaw2xboCpSol16KKEIHe987Ui&ust=1638377869062000&source=images&cd=vfe&ved=0CA0Q3YkBahcKEwiI_Y2qx8D0AhUAAAAAHQAAAAAQAw

The 20 joint locations of the hand used as input in the model

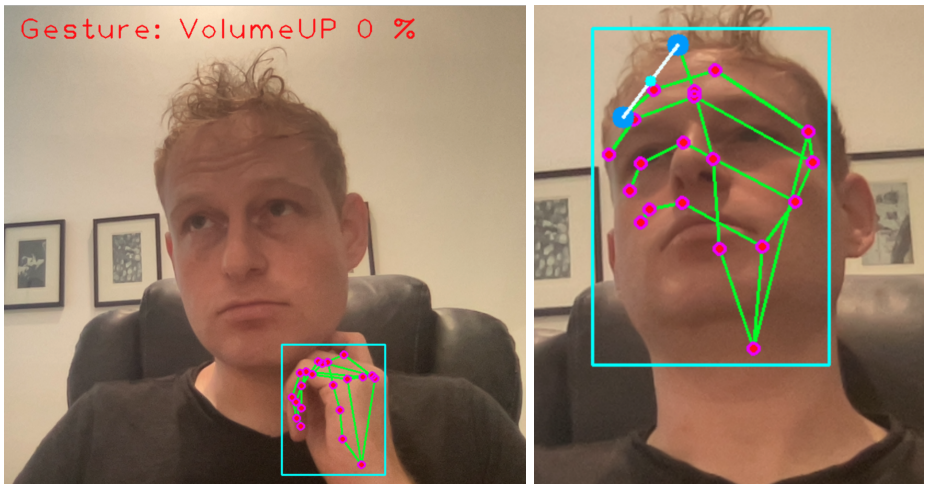
Once good stable lighting was achieved, the hand bounding box was always around the vicinity of the hand, but with lots of flicker as the hand moved. Delay between hand movement and bounding box/gesture response was negligible, which is promising. Of the points, there was what seemed to be random noise causing a mild jitter, which was usually not coordinated enough to cause false positives.

Hand in bounding box, true positive

Gestures, Continuous

The gestures I tried fell into two categories: continuous gestures (Pinching, Sliding, Waving, Dragging, etc) and Fixed Poses (thumbs up, peace sign, etc).

Most continuous gestures were limited in their success, since it's harder to get them to work as a sequence of frames, and there was a frame delay between hand motion and what was seen on camera, making them feel unintuitive. In addition ,these kinds of motions don't have the natural "end" point that happens with, say a thumbs-up. Predictably, there were many false positives. To take care of the false positives resulting from when your hand accidentally crosses the screen, I tried adding a fixed detection bounding box for your hand to be recognized in. This worked great but made gestures much slower and harder to incorporate into a day-to-day workflow, since you have to take some time to line up your hand with the camera.

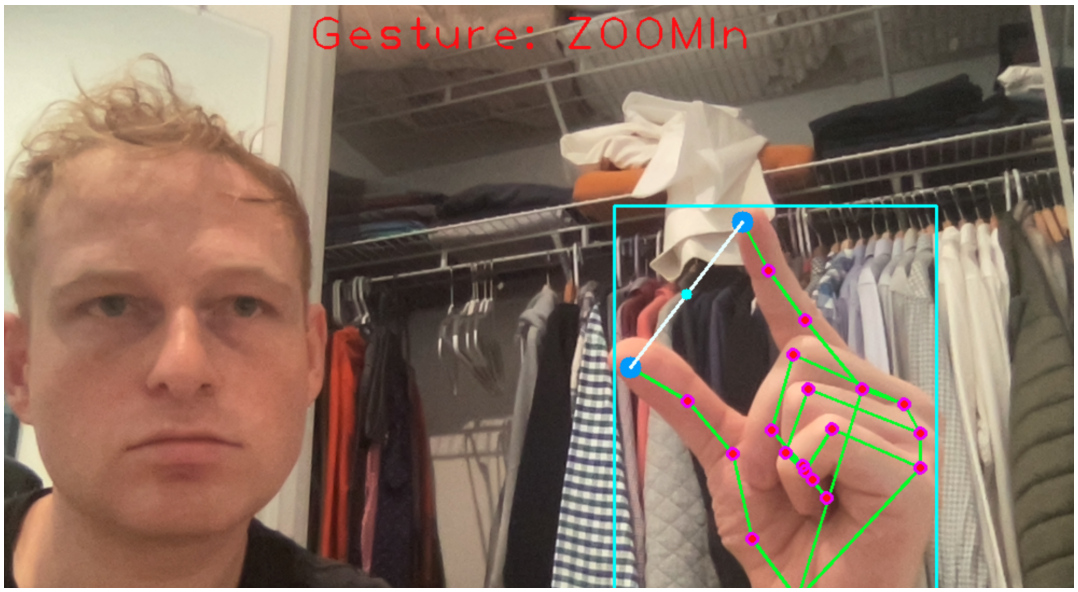


Two False Positive examples: One when scratching neck, one on face for no reason whatsoever

Dragging: Making a pulling motion of the hand. I tried to use this gesture to turn pages in documents and pan across a screen. It was a mess and did not succeed often. Like other continuous gestures it was very unclear as to when the gesture was over, so it was hard to get a good end point.

Waving: I attempted to use a wave-like gesture to delete text, it was unusably bad. Virtually any hand motion could set it off in the beginning, and I never got it to work.

Pinching: Pinching worked quite well, but also ran into the problem of not having a natural end point. The best I could achieve when testing out pinching to adjust screen brightness was to pinch to your desired brightness then immediately yank your hand out of frame before it had time to detect anything else. This usually worked, but was unpleasant and occasionally resulted in the screen dimming all the way to dark.



True Positive: Zoom Increase

Gestures, Fixed Pose

Raising fingers: One, two, or three fingers up were very easy to detect and worked well. Four and five were harder, and became easier when I started from three fingers and raised more. I found I could “store” functions in these well, like mapping two fingers to bringing one application to the front, three to bring another.

Thumbs Up/Down: These also were pretty reliable, especially thumbs down, since it’s such a unique pose that is not likely to happen by accident. I used these to boost and decrease the volume and brightness quite effectively.

Fist: A little less accurate since fists would get confused with hands at an angle, but still usable if not too many gestures were enabled. I mapped the fist and palm gestures to play and pause music and it worked well for a little while.

“OK”/Chef’s Kiss: This gesture is when you make a small circle with your first two fingers and raise your remaining fingers. It works when angled specifically head-on to the camera, and it was lacking some modifications that would be needed to make it more accurate. I mapped this to take a screenshot since it was mostly false negatives.

Accuracy

I conducted a very rough accuracy calculation across 45 gesture attempts in stable lighting (which would be ideal since lighting and environment gave so many issues). I tried three different rooms and three different lighting. A flicker was counted as a success as long as one of

the flickering values was the target. Enabling all gestures greatly increased false positives and decreased accuracy, so accuracy was calculated with five total possible gestures active at a time; this isn't perfect but a reasonable estimate of the amount of useful gestures a user would want to use at a time. Continuous gestures couldn't be assessed in the same way

One finger: 93%

Two fingers: 91%

Three fingers: 91%

Four fingers: 84%

Five fingers: 84%

Thumbs up: 84%

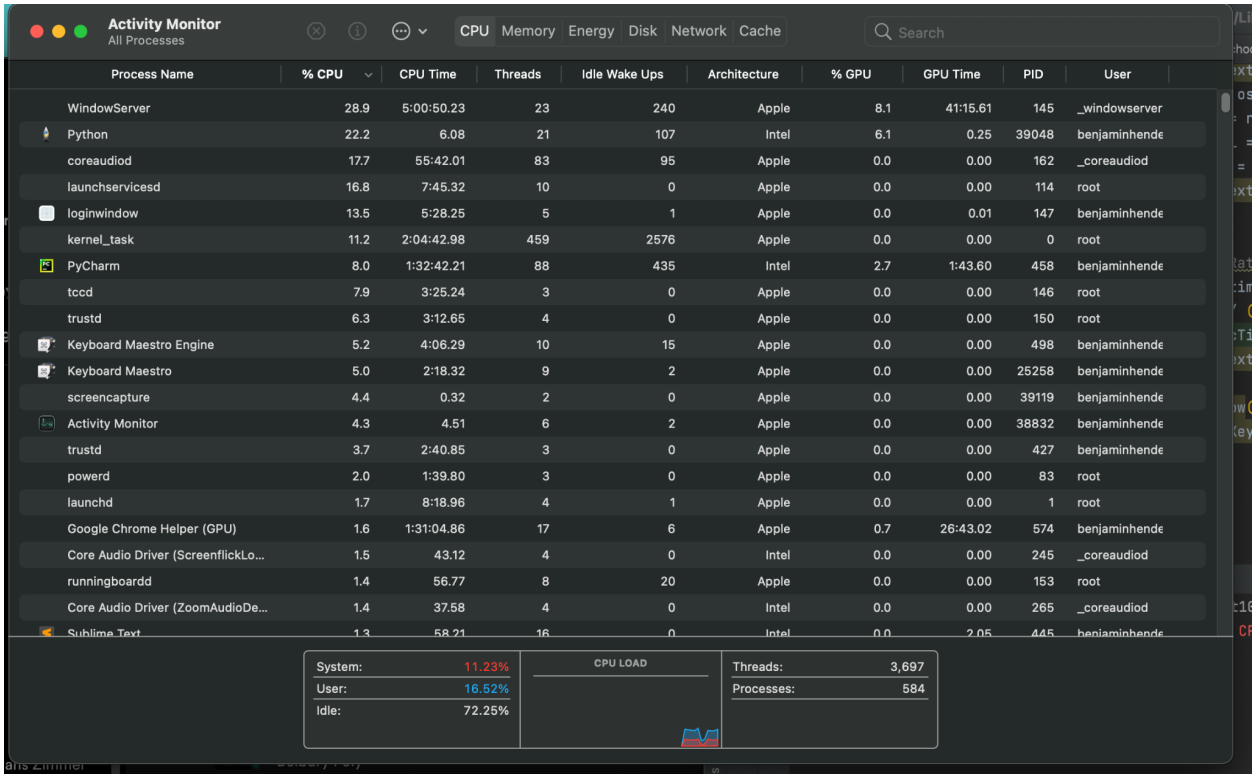
Thumbs down: 95%

Fist: 84%

"OK"/Chef's Kiss: 66%

Resource Usage

In order to assess feasibility, I took metrics on CPU usage screenshot and range on M1 2021 Macbook Pro, feasibility. Python and WindowServer jumped up to 22% and 28% each, with Keyboard Maestro related processes also consuming around 10%. These figures decreased if I didn't use Mediapipe.



Testing

The reasonably usable gestures were tested over the period of a week through a typical day to day routine. I had to be careful to have the same lighting and to keep my laptop in the same place in order to have a similar wall backdrop with no details or edges in frame.

In the end, the only ones worth doing were ones that were used infrequently and had a low false positive rate, also known as Specificity (TN/(TN + FP)). Pause/play, copy/paste, screenshot, continuous MIDI, are such examples. Since it's pretty infrequent that you want to take a screenshot, a user might be fine with a couple failed attempts being false negatives. They would not be fine with false positive screenshots going off, or seeing false negatives on more commonly used tasks. Volume and brightness adjustment using up/down was relatively decent to use once it calibrated, but lacked the continuity of the computer's slider bars. Using continuous gestures for these was too frustrating. Pausing, playing, and controlling music was easy to get used to and was nice to control from across the room. Switching between active applications was fast but happened too much by accident. Any precise movement and resizing of windows was unfeasibly bad, text manipulation was too frustrating. I never succeeded in incorporating any of the email and text messaging related functions into my day-to-day workflow.

Automations are useful if they:

Save time (faster than the alternative)

Are intuitive

Work reliably (> 80% Recall, higher precision)

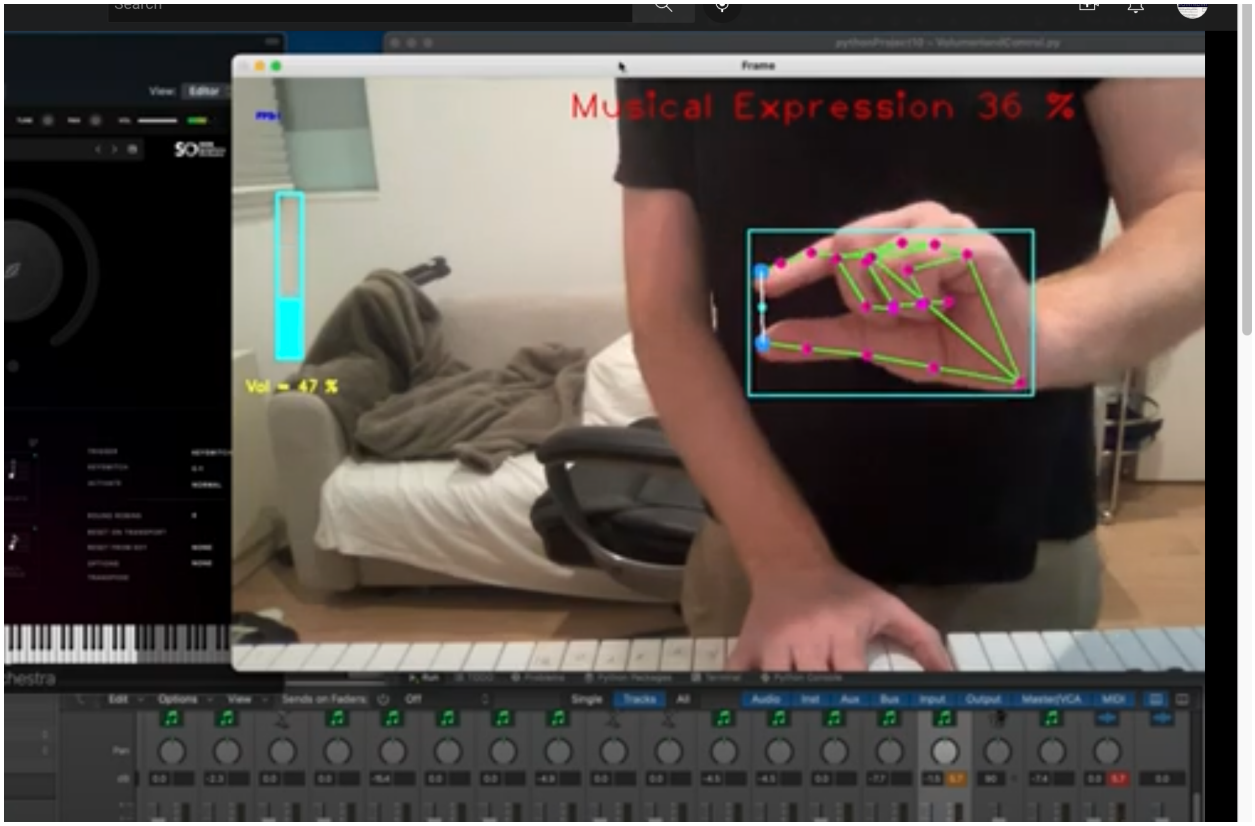
Can be incorporated/Are tested to be convenient for a user's day-to-day routine

I didn't find any of these functions to work better than a simple keyboard shortcut, and the most effective gestures still need tuning before they ship out to real users anytime soon. A more experienced purveyor of human-computer interfacing could likely find their way around many of the problems I encountered, and maybe even design specific new gestures suitable for recognition.

Here is a video of my favorite moment in this project, using gestures to control musical expression through keyboard MIDI CC control. Please watch this here:

https://youtu.be/xWYmvyXt2d0

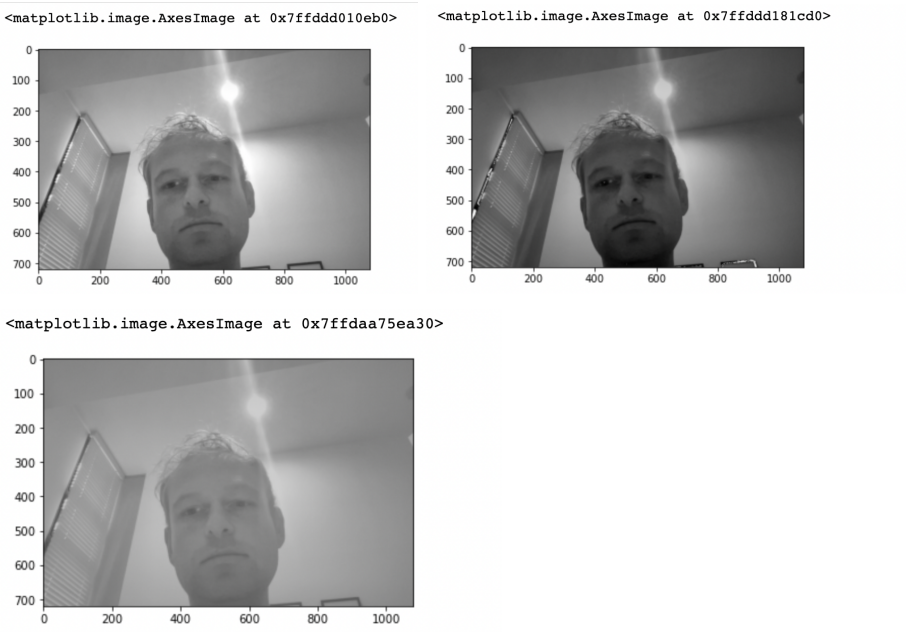
Overall, it was a decent way to control an instrument's intensity. The only thing it needed was less lag!



Room for improvement: Handling Different Data

It's clear that future improvements to tackle this problem will need to handle incoming light data better, far before it reaches any neural network model. Models have been trained to work in certain isolated environments, which will inevitably differ from your room. The presence of a candle or screen in the background can throw the whole thing off, so a commercially successful implementation model must be robust and trained in different rooms and outdoor environments.

Thus, the real challenge is to build image processing software that equips your camera to handle different light sources and still get a clear, crisp image. This is the exact problem that cell phone cameras try to tackle with Night Modes, and it still remains difficult with much room for improvement. In my implementation, I realized this when I tried out simple bits of code that attempted to adjust for harsh lighting, including doing operations such as scaling and limiting, but none of them was both effective and resulted in a natural looking image.



So it's clear that most of the improvement is in the processing and filtering of incoming data, normalizing what's on screen before any math is done. But of the potential improvements to the models, I have identified a few that may help it generalize to different environments. These are the following:

More training data

More complex architecture

Better training techniques

More training data was already described; A more complex architecture could include more layers/convolution, more neurons/parameters, or normalized/processed data. This would ideally reduce underfitting. Better training techniques would transform the data to make it generalize better and prevent overfitting. Examples of such techniques are Dropout (randomly disable neurons to make them generalize better and not overfit by making the same paths again and again), Data Augmentation (artificially generate more training data, adjust angle and lighting randomly in the hope that the model will be able to tackle more environments).

Postmortem and Future Steps

After extensive testing, it seems likely that overfitting is a bigger challenge for this specific task, and perhaps any task involving live video data. When every frame is blasting thousands of pixels, even small errors add up and false positives and negatives become numerous. How can a gesture detection application work if your hand's bounding box flickers and the operating system erroneously exits your current application? And, the most important:

How many users would tolerate having to redo gestures even 1% of the time when a wired mouse and keyboard are so foolproof?

Further attempts to make progress will need to solve the problems I encountered with an extremely low failure rate. My implementation ultimately didn't overcome account lighting variations and background motion, but it is possible someone has or will get past this hurdle. It's possible someone could design a method to make unmistakeable gestures that fit seamlessly in the day to day use of your devices, and better training and processing could make this once futuristic vision a reality.

Sources:

"Hand Gesture Recognition Database"
https://www.kaggle.com/datasets/gti-upm/leapgestrecog

"Gesture Recognition - Conv3D & Conv2D+RNN 35519d"
https://www.kaggle.com/code/anu2002/gesture-recognition-conv3d-conv2d-rnn-35519d

"Creating a hand tracking module"
https://www.section.io/engineering-education/creating-a-hand-tracking-module/

"Volume-control-using-hand-gesture-using-python-and-openCv"
https://github.com/Aaru77/Volume-control-using-hand-gesture-using-python-and-openCv

"Hand Gesture Recognition Using Mediapipe"
https://github.com/kinivi/hand-gesture-recognition-mediapipe

"Volume Hand Control"
https://github.com/Vibhugupta10616/Volume-Hand-control

"Real Time MIDI I/O For Python"
https://github.com/patrickkidd/pyrtmidi

"Control Mac Volume by Python"
https://dev.to/0xkoji/control-mac-sound-volume-by-python-h4g

Keyboard Maestro
https://www.keyboardmaestro.com

"Combining Python and Applescript"
https://leancrew.com/all-this/2013/03/combining-python-and-applescript/