CSCI 1430 Final Project Report: Gesture-Controlled Special Effects

TEAM STRANGE: Kevin Hsu, Mandy He, Sophia Liu, and Tianran Zhang
Brown University

Abstract

Inspired by the movie Dr. Strange*, we created a program that uses hand gestures to trigger special effects using real-time camera video. Our project contains three main components: 1) tuning a pre-trained model to recognize desired gestures and 2) rendering video frames captured through OpenCV with special effects and 3) mapping the special effects to the gestures. We hope to provide a potential solution to improve film effect production efficiency, as well as provide the general public with an interactive virtual experience where they could navigate through different modes using their hands.*

1. Introduction

We are inspired by the movie *Dr. Strange*, where the characters have special power to manipulate the surroundings with their hands. Film producers and directors may seek to overlay cinematic effects on the real-time film captured during production to assist them in determining how the film would look like post-production. These overlays could include applying a tint or a stylistic effect, duplicating an item in the scene, or moving a prop in the scene. The film crew may wish to test out different scene layouts and aesthetics during production but may be constrained by time and budget. Given this problem, our group decided to make an application where hand gestures would trigger a variety of visual effects on real-time video. This would allow directors and producers to temporarily change the scene in the film without touching their finely tuned cameras or the physical scene they are filming until they are sure of the change they want to make. Furthermore, our application gives access to people without film production background to an interactive virtual experience, where they could navigate through different filters and effects with their hands. .

2. Related Work

We began by researching about MediaPipe [3], a library that supports real-time hands and facial features. As we searched on Github about projects built upon MediaPipe, we were interested in a pre-trained CNN model that provides classification methods for finger landmarks as well as point history. [4]. Hence, we decided to build upon the model by Kazuhito Takahashi to recognize gestures and train it with our own gesture point data. In addition, to implement the special effects, we followed the pointilism paper[2], TensorFlow style transformation [5], and image segmentation[1] guides.

3. Method

We began with two possible paths: building and training a CNN model to recognize gestures by ourselves, or tuning a pre-trained model to recognize gestures we chose. Through experimentation, we realized that feeding images of hands through a CNN for gesture recognition would not produce ideal results. Rather than training with images, we modified a pre-existing model [4] to detect finger landmarks and trained our finger point coordinate data on the CNN.

However, even with fairly decent accuracy of gesture detection, there is a limited set of gestures that are easily distinguishable due to the complex nature of gestures. Hence, we established a "nested" gesture system that reuses the signs to trigger effects.

A selfie segmentation special effect (cropping the background from the video frame so only the person can be seen), for example, would be triggered following a sequence of gestures:

1. The gesture for "6" would pull up the special effects menu (listing possible effects, triggered by the gestures for "1", "2", "3", "4", and "5").
2. Then, the gesture for "3" would select "segmentation" and pull up the segmentation menu (listing the two types of segmentation, triggered by the gestures for "2" or "4").
3. Next, the gesture for "2" would select "selfie segmentation".
4. The user would use the gesture for "1" to drag the cropped selfie image around the video frame.
5. The user would use the gesture for "5" to temporarily place the cropped selfie image on top of the video frame.

6. The user would use the gesture for “6” to return to the special effects menu.

We implemented seven special effects in total. The following section introduces the implementation of each effect in detail.

1. Point Art Filter
   First, the video frame is preprocessed by applying a low pass filter (Gaussian filter) to the video frame. Then, the ten main colors of the frame are determined. Finally, the algorithm loops through the clusters of pixels in the video frame, painting a dot in the color that most represents that current cluster of pixels (shortest distance/difference using cdist).

2. Segmentation
   In our project we also utilize 2 different segmentation models, MediaPipe Selfie Segmentation (which is based on MobileNetV3) and a pre-trained semantic segmentation model (PSPNet), to segment frames from the video-stream. Color-maps, masks, and Numpy functions were then used to be able to shift the segmented image around the real-time video stream.

3. Cartoon
   We use a bilateral filter to smooth the colors and whittle down the gradients, then used edge detection for dark, thick borders that replicate line art in animations and cartoons.

4. Panorama
   We took a panorama picture from an iPhone, then displayed a subset of it. Panning around the width subset gives off the illusion that a person is turning in a room as they shift along the image.

5. Mural
   For this section, an image stylization model from Magenta was used to transfer style and color from a different image to the live camera feed. This model was designed to be used on arbitrary images, and so we were able to successfully conduct style transfer on most elements that could be provided in the camera frame, instead of being limited to specific items.

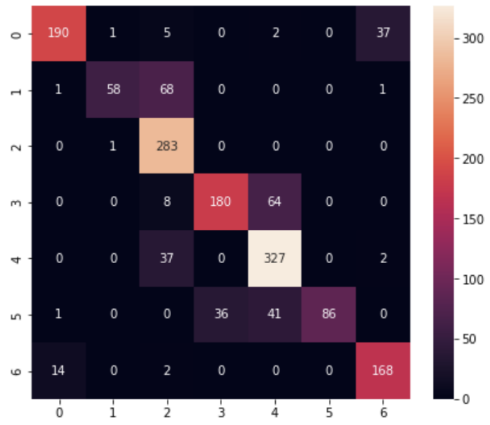
6. Drawing
   The drawing effect is implemented through OpenCV library. Our model records the point history of index finger. Based on the coordinates, we apply OpenCV to draw lines on an empty canvas, denoted by an array of zeros, and combine the canvas with the video frame image.

7. Light tunnel
   The light tunnel effects tracks the hand as the center and uses an numpy identity map to project the pixels outside the center circle into radiating lines.

8. Avatar filter
   The avatar filter, like the cartoon filter, is created using a bilateral filter and edge detection. However, the avatar filter also switches the video frames between BGR and RGB color space, giving the user’s skin a blue tint in the video frame.

4. Results

1. Gesture Recognition Model
   We tuned the pre-trained model to recognize seven gestures: zero, one, two, three (originally three fingers up), four, five, and six (“yolo” gesture). As we were training the hand gestures, we noticed that the gestures for “3” and “4” had low accuracy, as determined by experimental data. Specifically, the model often detected a “4” as a “5”, and a “3” as a “4”, most likely because of the similar shapes of these gestures. We decided to change the hand gesture for “3” to the “ok” sign, which included the third, fourth, and fifth fingers extended. We noted that the model was able to differentiate the “ok” gesture from the “4” gesture much better. The following chart shows our final recognition accuracy for each gesture.



Classification Report

	precision	recall	f1-score	support
0	0.92	0.81	0.86	235
1	0.97	0.45	0.62	128
2	0.70	1.00	0.82	284
3	0.83	0.71	0.77	252
4	0.75	0.89	0.82	366
5	1.00	0.52	0.69	164
6	0.81	0.91	0.86	184
accuracy			0.80	1613
macro avg	0.86	0.76	0.78	1613
weighted avg	0.83	0.80	0.79	1613

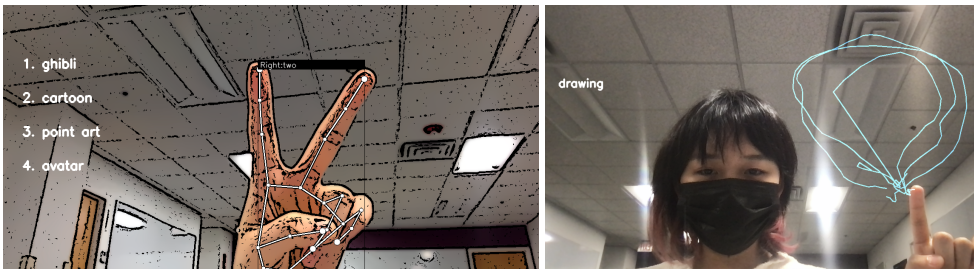


Figure 1. *Left:* Cartoon effect. *Right:* Drawing.

Figure 2. *Left:* Environment segmentation. *Right:* Selfie segmentation

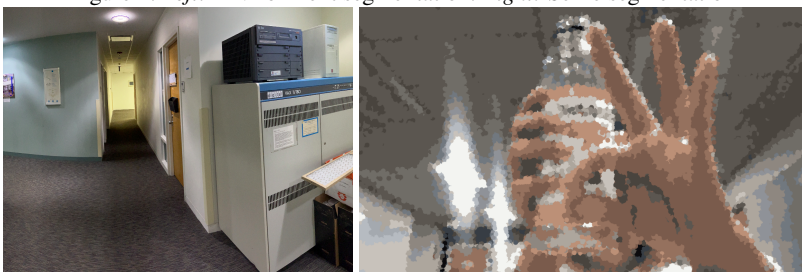


Figure 3. *Left:* Panorama. *Right:* Point art.

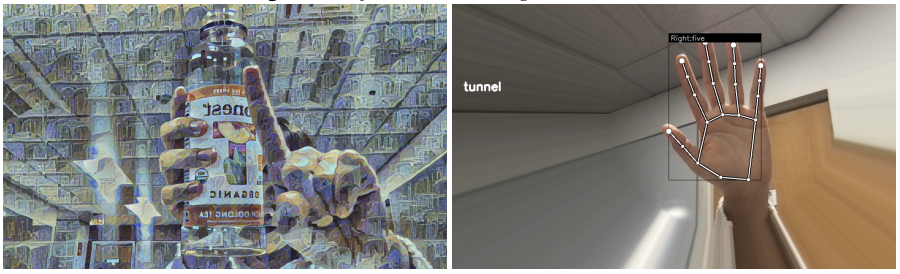


Figure 4. *Left:* Mural. *Right:* Light tunnel.

2. Special Effects Implementation
   We use the OpenCV library to process each video frame during the real time rendering. After the finger gesture of the user is captured, we mapped the label of gestures to different modes, each mode as an individual or a collection of special effects. The following figures present an overview to the special effects we have implemented.

4.1. Technical Discussion

As we added more features, especially more complex features that required multiple gestures, we found the need to add increasing amounts of instructions to aid the user in using our software. In other words, more features often meant less intuitive gestures, especially since the model we used limited the number of different gestures we could train.

As we increased the number of gestures to train the model for, the accuracy of each gesture recognition decreases. To maintain the accuracy, we limited the labels to seven gestures in total. This led to us often having to reuse gestures even when they didn't seem fitting for the feature/filter.

Furthermore, running two of the more complex special

effects, namely point art (which used a rather complex algorithm on thousands of pixels) and mural (which utilizes a real-time, arbitrary neural artistic stylization network) caused a severe video lag. We solve the issue by producing popup windows in the background that display the (slowly rendering) style-transformed images of the key frames while the original video can still run in the foreground and the user can still gesture. We hope to further simplify our model and improve the processing speed of the filters in the future.

The usage of the pre-trained semantic segmentation model also caused some lag. Unlike the speedy MediaPipe Selfie Segmentation which was specifically designed to be used on mobile applications, the model used for general/environment segmentation was much slower. Hence, we decided to limit the number of times we used the model by having the user activate the calling of the model only once each time the segmentation feature was chosen. Moreover, since this model was trained on a dataset consisting more of scenes than profile shots of people, it performs better on general scenes (bedrooms, suburban scenes, etc.) and was meant for our reach goal of taking input from a second camera.

4.2. Societal Discussion

1. Comment: We feel that it would be beneficial to highlight the importance of identifying discriminative gestures at an early stage of the model in order to replace or ignore them.

   Response: We carefully chose the gestures that the model was trained on. Our criteria for this selection were 1) differentiable from other gestures, 2) intuitive, 3) easy to remember, and 4) nondiscriminative. Our model is currently trained on the following hand signs: the gestures for the numbers "1", "2", "4", and "5", the "ok" gesture, and the "yolo" gesture.

2. Comment: We think that the team missed an important group of people that might be affected by the invention: the disabled, especially those who are unable to hear or speak.

   Response: Our application benefits those who are unable to hear and/or speak because voice commands are not necessary to control the application, unlike Siri or Alexa.

3. Comment: We should put an age limit to prevent young children from making potentially harmful edits to others' works.

   Response: Our application already does not support such actions as we do not store any images.

4. Comment: When using webcams in applications to capture hand gestures, some facial features of the user might be included. It would be great to highlight how the system is going to process and deal with the storage of those images to protect user privacy.

   Response: Our system will not store any images. At most, some images are stored in a global variable until the program is closed/exited, but once the program is closed, these images are deleted.

5. Comment: There should be a control on the choice of elements of special effects that can be triggered by hand gestures, especially those that include jump scares and violence. There should also be a filter on age group since younger kids might be more sensitive to certain elements than adults.

   Response: We've chosen the features of our application to ensure there are no popups that could be intimidating or trigger panicking emotions for the general public. Most of the features are similar to operations you could piece together in pixel-manipulating software like Photoshop.

6. Comment: In addition to serving influencers, directors, and animators, our group feels that this project can contribute to educational projects that use special effects to spark interest among

   Response: Our group agrees with this statement and also sees the potential benefits our project could bring to educational projects/efforts. For instance, our project could potentially be used to teach young children body parts. The segmentation feature could potentially be expanded so that young children could point at their own features given a prompt on the screen.

5. Conclusion

Our group implemented gesture-controlled special effects. This could save people like film producers and producers a lot of time while offering them flexibility. Going forward, this feature could potentially be incorporated in phone and/or laptop cameras as well as potentially professional cameras to further facilitate the creation of art and media.

References

[1] divamgupta. Image segmentation keras : Implementation of segnet, fcn, unet, pspnet and other models in keras, 2021. https://github.com/divamgupta/image-segmentation-keras. 1

[2] Hong Liu. Create pointillism art from digital images. https://web.stanford.edu/class/ee368/Project_Autumn_1516/Reports/Hong_Liu.pdf. 1

[3] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines. 2019. https://arxiv.org/abs/1906.08172. 1

[4] Kazuhito Takahashi. Hand gesture recognition using mediapipe, 2021. https://github.com/Kazuhito00/hand-gesture-recognition-using-mediapipe. 1

[5] TensorFlow. Tensorflow neural style transfer, 2022. https://www.tensorflow.org/lite/examples/style_transfer/overview. 1

Appendix

Team contributions

Sophia Liu Implemented the point art filter and created the avatar filter via the cartoon filter.

Mandy He Implemented the selfie-segmentation and general/environement segmentation features/effects.

Tianran Zhang Implemented the drawing mode and the light tunnel effect, collected gesture data, and trained the pre-trained model.

Kevin Hsu Implemented the mode selection system with feature hierarchies, in addition to creating panorama mode, a cartoon filter, and style transfer.