Simon Fraser University, CMPT 310
Written by: Simran Vig
Project by: Junghun Byun, Ines Machinho Rodrigues, Simran Vig

How To Build a Hand Gesture Detection Dataset

Introduction

This guide details the steps to build a hand gesture detection dataset using OpenCV and MediaPipe. It intends to make the process of creating a dataset with hundreds of entries efficient and accessible with limited technology. The resulting dataset is fit to use for training an AI classification model.

The example gestures used are “gun” and “fist,” but any hand gestures could be used. To see an application of this guide in real-time, see our Space Invaders Gesture Detection Game, whose AI was trained off the dataset used for this guide’s examples.

This guide is intended for creating a dataset containing information for detecting a single hand in the frame.

Requirements

Language: Python (version 3.10 or earlier)

External Libraries: OpenCV, MediaPipe

Standard Libraries: CSV, OS

Hardware: Webcam (must be granted permission by your IDE)

Operating System: MacOS, Linux, or Windows

Background

MediaPipe has two modules that are used in this program: `hands` and `drawing_utils`. `hands` is used to access the landmarks of the hand being tracked. These landmarks are 21 (x,y,z) tuples that map to various parts of the hand.

Simon Fraser University, CMPT 310
Written by: Simran Vig
Project by: Junghun Byun, Ines Machinho Rodrigues, Simran Vig

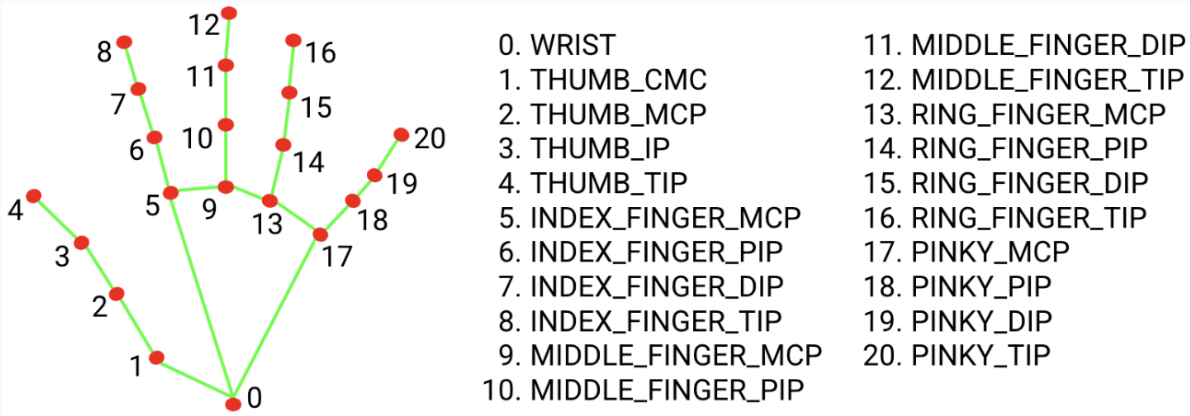

Figure 1: MediaPipe documentation of the landmarks in the Hands module.

As such, recording the data of one hand should yield 63 data points in total. `drawing_utils` is used to visualize the 21 landmarks on the screen, so that the user can see how their hand is being tracked in real-time.

OpenCV is used to capture images of the hand on the screen, and optionally, to write text on the screen.

Examples of how both libraries are used are shown in the steps and examples below.

If you are planning to use this dataset to train a classification model, we recommend taking an equal number of captures for each gesture in your dataset. In addition, we recommend taking captures under various lightings, using different hand sizes, and from various distances from the camera. This will help ensure that your model is being trained on balanced data.

Steps

1) Create a new .py file.

- (Recommendation: Name the file dataset.py, dataset_creator.py)

2) Import the external `cv2` and `mediapipe` modules to your .py file.
(These may need to be installed first, as they are external libraries.)

- (Recommendation: `import mediapipe as mp`)

- For issues with installing MediaPipe, see **Tip 1** in **Troubleshooting**.

3) Import the internal `os` and `csv` modules to your .py file.

4) Access MediaPipe's `hands` module by assigning a variable to `mp.solutions.hands`.

- Ex: `mp_hands = mp.solutions.hands`

5) Declare an instance of the `hands` class.
By default, this instance will track up to two hands, with up to 21 landmarks per hand.

- Ex: `hands = mp_hands.Hands()`

Note: Optionally, you can pass in arguments for the maximum number of hands to track (up to 2), the minimum detection confidence (from 0 to 1), and the minimum tracking confidence (from 0 to 1). By default, these arguments will be 2, 0.5, and 0.5.

6) Access MediaPipe's draw module by assigning a variable to `mp.solutions.drawing_utils`.
This will later allow you to see the landmarks of the hands on the screen.

- Ex: `mp_draw = mp.solutions.drawing_utils`

By this point, your .py file should look something like this:

```
import cv2
import mediapipe as mp
import csv
import os

mp_hands = mp.python.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.python.solutions.drawing_utils
```

7) Create (or open, if it already exists) the CSV file that will store the dataset.
Label each column from right to left as "label," "x1," "y1", "z1", "x2", "y2", "z2", … . "x21", "y21", "z21".

- Ex:

```
csv_file = 'dataset.csv'
if not os.path.exists(csv_file):
    with open(csv_file, 'w', newline='') as file:
        writer = csv.writer(file)
        header = ["label"]
```

```
        for i in range(21):
            header.extend([f"x{i}", f"y{i}", f"z{i}"])
        writer.writerow(header)
```

8) Access your webcam by assigning a variable to `cv2.VideoCapture(0)`.
The (0) argument is issued to indicate that your program will use the default webcam on your system.
Also, initiate two flags that will be used later to indicate whether the capture is on, and whether there is already a label assigned to the current gesture.

- Ex:
  ```
  capture = cv2.VideoCapture(0)
  current_label = None
  recording_enabled = False
  ```

9) Create the main capture loop to start tracking your hand in real-time.
This should be a `while True` loop which:

- Reads frames from the webcam
  ```
  ret, frame = capture.read()
  ```
- Flips the image horizontally to mirror the camera feed
  ```
  flipped_frame = cv2.flip(frame, 1)
  ```
- Changes the capture from OpenCV's default colour formatting to RGB so that MediaPipe can access it.
  ```
  rgb_image = cv2.cvtColor(flipped_frame, cv2.COLOR_BGR2RGB)
  ```
- Detects the hand(s) in the capture.
  ```
  result = hands.process(rgb_image)
  ```

10) Within the capture loop, process the first detected hand on the screen.
To do this, check if your result variable is detecting at least one hand on the screen.
If so, store the 21 landmarks of this hand in a variable.

- Ex:

```
if result.multi_hand_landmarks:
        first_hand = result.multi_hand_landmarks[0]
```

The [0] parameter specifies that only the first hand that is detected will be processed.
The `multi_hand_landmarks` attribute fetches the 21 (x,y,z) tuples we want to store in the dataset.

11) After checking if recording is enabled and that there is a label on the screen for the current gesture, write the 63 data points to your csv file.

- Ex:

```
if recording_enabled and current_label is not None:
            new_row = [current_label]
            for point in first_hand.landmark:
                new_row.extend([point.x, point.y, point.z])
            with open(csv_file, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(new_row)
```

12) Optionally, display a message on the screen to show that a capture of the current hand was taken.
Then, set your `recording_enabled` flag to False to prevent an infinite loop.

- Ex:

```
message = "RECORDED: " + current_label
cv2.putText(flipped_frame, message, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
recording_enabled = False
```

13) Optionally, show the current label on the screen so that the user knows what they chose to assign the given gesture to.

- Ex:

```
if current_label is not None:
        label_text = "Label: " + current_label
        cv2.putText(flipped_frame, label_text, (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 0), 2)
```

14) Display the video frame (which you can optionally annotate), and initialize a key using `cv2.waitKey()` so that the capture can receive keyboard input.

- Ex:
```
cv2.imshow("Gesture Recorder", flipped_frame)
key = cv2.waitKey(1) & 0xFF
```

The (1) argument is optional, but introduces a 1 millisecond delay to wait for a key to be pressed.
However, `& 0xFF` is a **necessary** bitmask to keep only the lowest 8 digits of the key code returned by `cv2.waitKey(1)` . By keeping only the lowest 8 digits, the integer is converted to a standard ASCII key code that can be read by and translated to most keyboards/systems.

15) Assign keys to stop the recording, assign a label to the gesture on screen, or take a capture of the current screen.

- Ex:
```
if key == ord('q'):
     break
elif key == ord('r'):
     recording_enabled = True
elif key == ord('1'):
     current_label = 'fist'
elif key == ord('2'):
     current_label = 'gun'
```

Any keys can be chosen, and any number of labels can be added beyond just “fist” and “gun.”

16) Finally, outside of the `while True` loop, release the capture and destroy the recording window.

- Ex:
```
capture.release()
cv2.destroyAllWindows()
```

17) Run your program. If you encounter issues, see the **Troubleshooting** section.

Simon Fraser University, CMPT 310
Written by: Simran Vig
Project by: Junghun Byun, Ines Machinho Rodrigues, Simran Vig

Sample Run

1) The user starts the program, and the feed begins.

Figure 2: Example of MediaPipe visualizing hand landmarks on webcam feed.

2) The user imitates their desired hand gesture, and assigns it the correct label based on the key they initialized (in the example code, the ‘2’ key). In this sample, the hand gesture is “gun.”

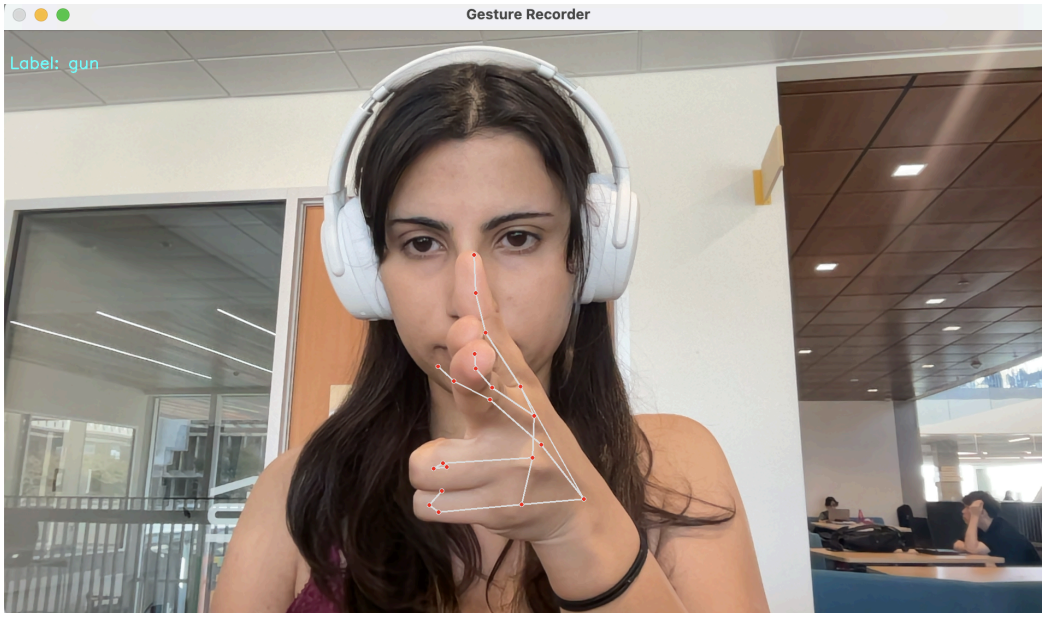

Figure 3: Example of MediaPipe tracking a “gun” hand gesture on webcam feed.

3) The user takes a capture by pressing the key they initialized (in the example code, the ‘r’ key).

4) When the user is done recording gestures, they exit the capture by pressing the key they initialized (in the example code, the ‘q’ key).

5) The user checks their dataset.csv file and sees that their information has been stored in new rows, like so:

dataset.csv

```
label,x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4,x5,y5,
gun,0.6407729983329773,0.7238888144493103,-2.672981906925
gun,0.6663428544998169,0.7145697474479675,-1.218818823645
gun,0.5973020792007446,0.6911460161209106,-2.594738930383
gun,0.5462788343429565,0.6839188933372498,-2.274107657740
gun,0.530208945274353,0.7899895906448364,-9.6578816055625
gun,0.46679356694221497,0.7405903935432434,-1.44710412541
```

Figure 4: Example of the csv file that stores the landmark data.

Troubleshooting

Tip 1) Issues installing MediaPipe

A common issue with installing the MediaPipe library, especially for Python users, is that it is only compatible with versions of Python up to 3.10. As such, we recommend to either:

a) change the interpreter on your IDE to Python 3.10, or

b) create a virtual environment that will use Python 3.10 instead.

To do so, navigate to your current project folder in the terminal, and then enter
`python3.10 -m venv myenv`
in the terminal to create a new virtual environment called my env. Then, enter
`source myenv/bin/activate`
to activate the virtual environment.

To ensure that you are now working with Python 3.10, enter
`python3 --version`
in the terminal, and ensure that 3.10 is returned.

Tip 2) Issue accessing webcam

When you first run your .py file, your IDE might be reluctant to access your webcam. Give your IDE permission to access the webcam in your system's settings. Then, restart the program and try running your .py file again.

Tip 3) OpenCV is not capturing the webcam

If you run your program, but it opens a blank window, ensure that only one application is using the webcam. Also see Tip 2) in case there is an issue with webcam permissions.

Tip 4) No landmarks shown on screen

If you run the program, and the feed starts, but you see no landmarks on the screen:

a) Make sure your hand is fully visible in the frame, and well-lit
b) Try only including one person in the frame at a time to help MediaPipe detect your hand
c) Ensure that you follow the steps in the **Steps** section for accessing the `drawing_utils` module and showing the landmarks on the screen. (Namely, Step 6).

Potential Improvements

1. The program could include the option for the user to delete the most recent capture. This would be useful if the user accidentally uses the wrong label for the gesture on the screen.

2. The program is extended to track the coordinates of multiple hands

3. The user interface of the program is made more visually appealing/accessible, or extended to include more information (ie, displaying coordinates on screen, showing errors, etc).

References

1. The following GIT repository was used to see a working example of how MediaPipe is used to track hands.
https://github.com/Sousannah/hand-tracking-using-mediapipe/blob/main/hand_tracking.py

2. The following MediaPipe documentation was used for the diagram of the landmarks used in this report, **Figure 1,** and to help explain the reasoning behind the code snippets shown in the **Steps** section.
https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

3. To see how a dataset we created was used to train a linear classification model and create a game, see this repository: https://github.com/HoonieB/310_project/tree/feature/game-interface

4. **Figure 2, Figure 3,** and **Figure 4** were taken by Simran Vig using the program in the above repository.