

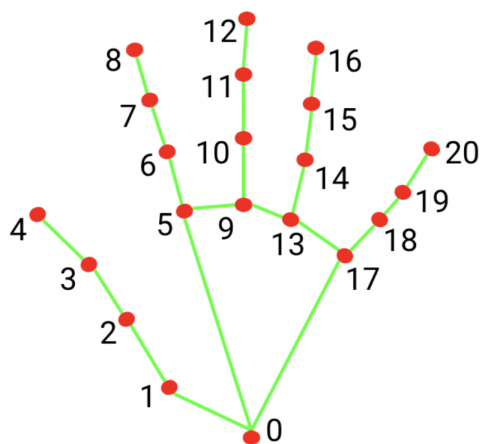
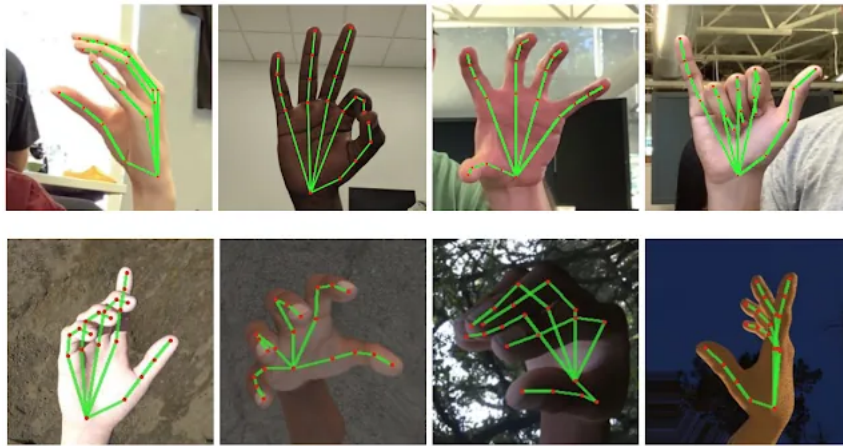
# AI VIET NAM – COURSE 2024

## Light Controlling Using Hand Gestures

Tho-Anh-Khoa Nguyen, Duc-Thang Nguyen, Le-Phi-Long Bui

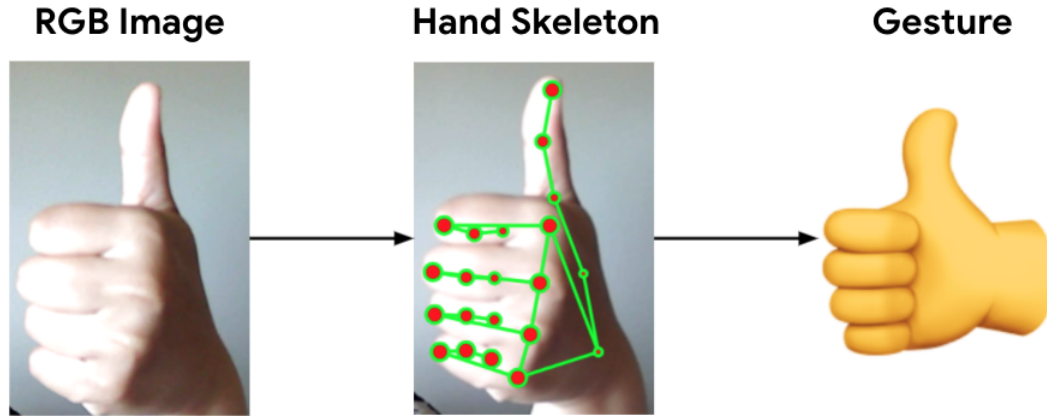
Ngày 28 tháng 11 năm 2024

### PHẦN I: Giới Thiệu



- |                       |                       |
|-----------------------|-----------------------|
| 0. WRIST              | 11. MIDDLE_FINGER_DIP |
| 1. THUMB_CMC          | 12. MIDDLE_FINGER_TIP |
| 2. THUMB_MCP          | 13. RING_FINGER_MCP   |
| 3. THUMB_IP           | 14. RING_FINGER_PIP   |
| 4. THUMB_TIP          | 15. RING_FINGER_DIP   |
| 5. INDEX_FINGER_MCP   | 16. RING_FINGER_TIP   |
| 6. INDEX_FINGER_PIP   | 17. PINKY_MCP         |
| 7. INDEX_FINGER_DIP   | 18. PINKY_PIP         |
| 8. INDEX_FINGER_TIP   | 19. PINKY_DIP         |
| 9. MIDDLE_FINGER_MCP  | 20. PINKY_TIP         |
| 10. MIDDLE_FINGER_PIP |                       |

1. **Tổng quan project** Project Light Controlling Using Hand Gestures nhằm phát triển một hệ thống cho phép người dùng điều khiển đèn thông qua các cử chỉ tay. Project kết hợp công nghệ nhận diện cử chỉ Google với deep learning model để tạo ra một hệ thống nhận diện cử chỉ tay và thực thi lệnh tương ứng với các loại cử chỉ của tay đã được định nghĩa trước. Project được triển khai qua ba giai đoạn chính:



Hình 1: Hand Gesture Recognition

(a) **Step 0 (Chuẩn Bị Data):** Trong giai đoạn này, chúng tôi sử dụng MediaPipe Gesture Recognizer của Google kết hợp với camera để thu thập dữ liệu về các cử chỉ tay theo các class (class) do chúng tôi định nghĩa. Quá trình này bao gồm:

- Thu thập dữ liệu: Ghi lại các cử chỉ tay tương ứng với từng class.
- Xử lý dữ liệu: Sử dụng MediaPipe để nhận diện và trích xuất đặc trưng của các cử chỉ
- Lưu trữ dữ liệu: Chia dữ liệu thành bộ train, validation, và test, sau đó lưu vào các file CSV để sử dụng trong quá trình huấn luyện mô hình.

(b) **Step 1 (Huấn Luyện Mô Hình Phân Loại):** Ở giai đoạn này, chúng tôi xây dựng một mô hình MLP (Multi-Layer Perceptron) để huấn luyện trên bộ dữ liệu đã chuẩn bị:

- Thiết kế mô hình: Xác định kiến trúc của MLP phù hợp với dữ liệu cử chỉ tay.
- Huấn luyện mô hình: Sử dụng bộ dữ liệu train và validation để huấn luyện và tinh chỉnh mô hình.
- Đánh giá mô hình: Kiểm tra hiệu suất của mô hình trên bộ dữ liệu test để đảm bảo độ chính xác trong việc phân loại các cử chỉ.

(c) **Step 2 (Triển Khai Hệ Thống Thực Tế):** Trong giai đoạn cuối, chúng tôi triển khai hệ thống điều khiển đèn theo thời gian thực:

- Nhận diện cử chỉ: Sử dụng webcam để nhận hình ảnh real-time, sau đó áp dụng MediaPipe Gesture Recognizer để trích xuất cử chỉ tay.
- Phân loại cử chỉ: Đưa dữ liệu cử chỉ vào mô hình MLP đã huấn luyện để xác định class tương ứng.
- Điều khiển đèn: Dựa trên class cử chỉ nhận được, thực hiện thao tác tắt/mở đèn trong mô phỏng. (Tùy chọn) Hệ thống có thể điều khiển thiết bị đèn thực tế bằng cách sử dụng module 4 relay giao tiếp qua Modbus RTU RS485.

2. **Giới Thiệu Hand Gestures Recognition** Hand Gestures Recognition là một lĩnh vực quan trọng trong tương tác người và máy, cho phép hệ thống hiểu và phản hồi lại các động tác tay của người dùng. Công nghệ này dựa trên việc phân tích hình ảnh hoặc video để xác định vị trí và hình dạng của bàn tay, từ đó nhận diện các cử chỉ cụ thể.

Các bước chính trong nhận diện cử chỉ tay:

- Phát hiện bàn tay: Xác định vị trí của bàn tay trong khung hình bằng các kỹ thuật xử lý ảnh.

- Trích xuất đặc trưng: Thu thập thông tin về hình dạng, vị trí các ngón tay, góc độ, v.v.
- Phân loại cử chỉ: Sử dụng các mô hình học máy để phân loại cử chỉ dựa trên các đặc trưng đã trích xuất.

Ứng dụng của nhận diện cử chỉ tay:

- Điều khiển thiết bị: Tương tác với máy tính, điện thoại, hoặc các thiết bị thông minh mà không cần chạm.
- Thực tế ảo và tăng cường: Cải thiện trải nghiệm người dùng trong môi trường ảo.
- Hỗ trợ người khuyết tật: Giúp người khuyết tật vận động hoặc giao tiếp dễ dàng hơn.

Công nghệ MediaPipe của Google: MediaPipe là một framework mã nguồn mở cung cấp các giải pháp tiên tiến cho xử lý đa phương tiện thời gian thực. MediaPipe Gesture Recognizer là một trong những giải pháp mạnh mẽ cho nhận diện cử chỉ tay, với khả năng:

- Nhận diện chính xác: Cung cấp mô hình tiên tiến với độ chính xác cao.
- Thực tế ảo và tăng cường: Cải thiện trải nghiệm người dùng trong môi trường ảo.
- Hỗ trợ người khuyết tật: Giúp người khuyết tật vận động hoặc giao tiếp dễ dàng hơn.

## PHẦN II: Thực Hành

### Step 0: Chuẩn Bị Data và Môi Trường

Trong bước đầu tiên của dự án "Light Controlling Using Hand Gestures", chúng ta sẽ tập trung vào việc chuẩn bị dữ liệu cần thiết để huấn luyện mô hình nhận diện cử chỉ tay. Bước này bao gồm việc thiết lập môi trường làm việc, cài đặt các thư viện cần thiết, cấu hình các class cho các cử chỉ, và thu thập dữ liệu thông qua việc ghi lại các cử chỉ tay sử dụng MediaPipe Gesture Recognizer của Google.

#### 1. Thiết Lập Môi Trường Làm Việc

**1.1. Sử Dụng Conda Với Python 3.10:** Chúng ta sẽ sử dụng Conda để tạo môi trường ảo cho project với yêu cầu sử dụng Python phiên bản 3.10

- Cài đặt Anaconda hoặc Miniconda: Nếu bạn chưa có Conda trên máy tính, hãy tải và cài đặt từ trang chủ:

- Anaconda: <https://www.anaconda.com/download>
- Miniconda: <https://docs.anaconda.com/miniconda/>

- Tạo môi trường mới với Python 3.10:

```
1 conda create -n gesture_env python=3.10.0
```

- Kích hoạt môi trường:

```
1 conda activate gesture_env
```

**1.2. Cài Đặt Các Thư Viện Từ requirements.txt:** Sau khi môi trường được activate, chúng ta tiến hành cài đặt các thư viện cần thiết được liệt kê trong file requirements.txt.

```
1 pip install -r requirements.txt
```

#### 2. Cấu Hình Các Class Cử Chỉ Trong hand\_gesture.yaml

##### 2.1. hand\_gesture.yaml:

File hand\_gesture.yaml được sử dụng để định nghĩa các class cử chỉ tay sẽ được nhận diện và liên kết với các hành động điều khiển đèn tương ứng. Việc sử dụng file YAML giúp chúng ta dễ dàng chỉnh sửa và mở rộng danh sách các cử chỉ mà không cần thay đổi code trực tiếp.


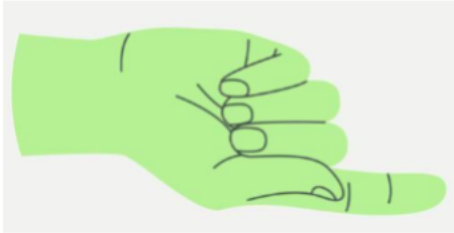



Nội dung file:

```
1 gestures:
2   0: "turn_off"
3   1: "light1"
4   2: "light2"
5   3: "light3"
6   4: "turn_on"
```

Giải thích:

- **gestures:** Đây là key, chứa danh sách các cử chỉ.
- **0, 1, 2, 3, 4:** Các số đại diện cho class của từng loại cử chỉ.
- **"turn\_off", "light1", "light2", "light3", "turn\_on":** Tên của các cử chỉ, đồng thời cũng là hành động điều khiển đèn tương ứng.

**2.2. Cách Chỉnh Sửa và Mở Rộng:** Nếu bạn muốn thêm hoặc chỉnh sửa các cử chỉ, chỉ cần thay đổi nội dung trong file hand\_gesture.yaml. Ví dụ, nếu muốn thêm cử chỉ cho "light4":

Class	Action	Hand Gesture
0	Turn All the Lights Off	
1	Turn the Lights 1 On	
2	Turn the Lights 2 On	
3	Turn the Lights 3 On	
4	Turn All the Lights On	

Hình 2: Class và action tương ứng với các cử chỉ để điều khiển đèn

```

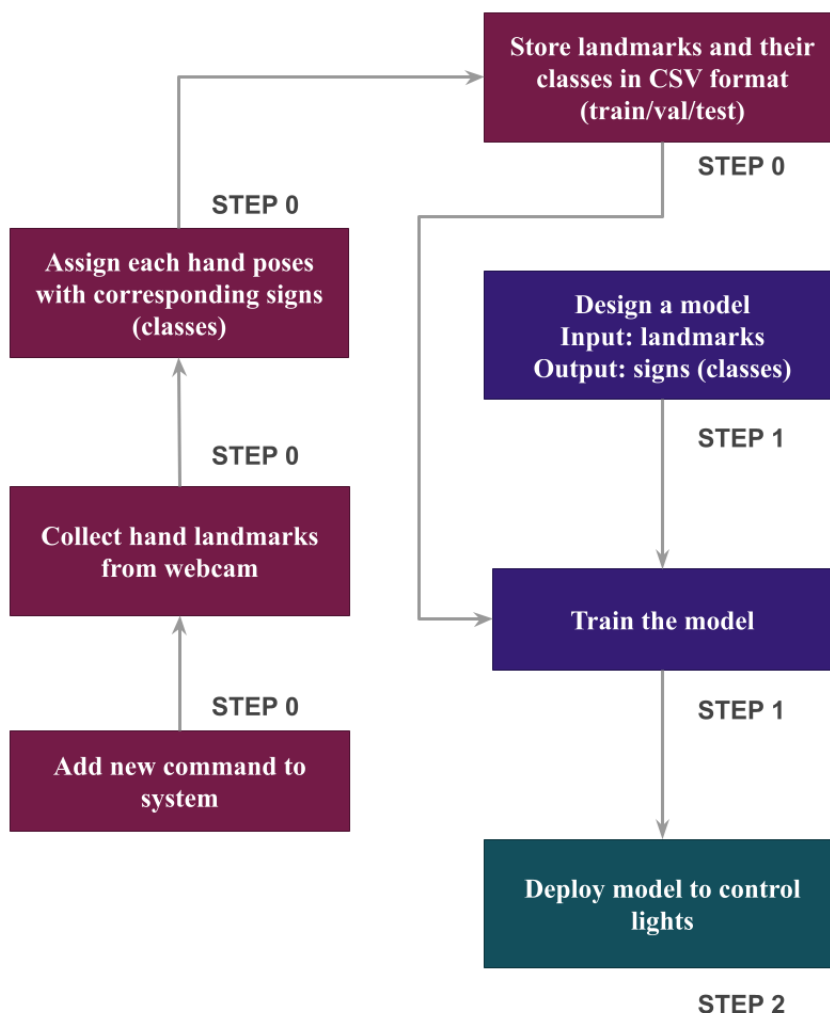
1 gestures:
2   ...
3   5: "light4"

```

Sau đó, các bạn cần thu thập dữ liệu cho cử chỉ mới và cập nhật mô hình.

### 3. Giới Thiệu Về File generate\_landmark\_data.py

File generate\_landmark\_data.py là một script Python quan trọng trong dự án, chịu trách nhiệm thu thập dữ liệu về các landmarks của bàn tay khi thực hiện các cử chỉ đã định nghĩa. Dữ liệu này sau đó sẽ được sử dụng để huấn luyện mô hình phân loại trong bước tiếp theo.



Hình 3: Các bước thực hiện project

#### 3. 1. Tổng Quan các bước thực hiện thu thập cử chỉ data:

- **Ghi lại các cử chỉ tay qua webcam:** Người dùng thực hiện các class cử chỉ trước camera, và sử dụng các phím trên keyboard để ghi lại dữ liệu landmarks của bàn tay tương ứng với các class cử chỉ.
- **Gắn nhãn cho dữ liệu:** Mỗi cử chỉ sẽ được gắn với một nhãn tương ứng dựa theo cấu hình trong file hand\_gesture.yaml.
- **Lưu trữ dữ liệu:** Dữ liệu được lưu vào các file CSV để sử dụng cho việc huấn luyện mô hình.

### 3.2. Import Các Thư Viện Cần Thiết:

```
1 import os
2 import cv2
3 import csv
4 import yaml
5 import numpy as np
6 import mediapipe as mp
```

- **os:** Tương tác với hệ điều hành.
- **cv2:** Thư viện OpenCV để xử lý hình ảnh và video.
- **csv:** Đọc và ghi các file CSV.
- **yaml:** Đọc và ghi các file YAML.
- **numpy:** Xử lý mảng và tính toán số học.
- **mediapipe:** Framework của Google để nhận diện bàn tay và các landmarks.

### 3.3. is\_handsign\_character function:

```
1 def is_handsign_character(char:str):
2     return ord('a') <= ord(char) <ord("q") or char == " "
```

- **Chức năng:** Kiểm tra xem ký tự nhập vào có phải là một chữ cái thường từ 'a' đến 'z' hoặc khoảng trắng không.  
**NOTE:** Chúng ta sẽ assign class như trong file yaml config từ 0 trở đi tương ứng với phím 'a' trên keyboard. Ví dụ ta nhấn 'a' tương đương class 0, 'b' tương đương class 1, ... cho đến trước 'q'.
- **Mục đích:** Xác định các phím bấm hợp lệ để bắt đầu hoặc kết thúc việc ghi dữ liệu cho một class cử chỉ.

### 3.3. label\_dict\_from\_config\_file function:

```
1 def label_dict_from_config_file(relative_path):
2     with open(relative_path,"r") as f:
3         label_tag = yaml.full_load(f)["gestures"]
4     return label_tag
```

- **Chức năng:** Đọc file hand\_gesture.yaml và trả về một từ điển chứa các nhãn cử chỉ.
- **Mục đích:** Sử dụng các nhãn này để gán nhãn cho dữ liệu thu thập.

### 3.4. HandDatasetWriter class:

```
1 class HandDatasetWriter():
2     def __init__(self,filepath) -> None:
3         self.csv_file = open(filepath,"a")
4         self.file_writer = csv.writer(self.csv_file,delimiter=',',quotechar='|',
5         quoting=csv.QUOTE_MINIMAL)
6         def add(self,hand,label):
7             self.file_writer.writerow([label,*np.array(hand).flatten().tolist()])
8         def close(self):
9             self.csv_file.close()
```

- **Chức năng:** Ghi dữ liệu landmarks của bàn tay và nhãn tương ứng vào file CSV.
- **Method `__init__`:** Khởi tạo và mở file CSV để ghi dữ liệu.
- **Method `add`:** Thêm một dòng dữ liệu mới vào file CSV, bao gồm nhãn và tọa độ các landmarks.
- **Method `close`:** Đóng file CSV sau khi hoàn tất ghi dữ liệu.

### 3.5. HandLandmarksDetector class:

```

1 class HandLandmarksDetector():
2     def __init__(self) -> None:
3         self.mp_drawing = mp.solutions.drawing_utils
4         self.mp_drawing_styles = mp.solutions.drawing_styles
5         self.mp_hands = mp.solutions.hands
6         self.detector = self.mp_hands.Hands(False, max_num_hands=1,
7 min_detection_confidence=0.5)
8
9     def detectHand(self, frame):
10        hands = []
11        frame = cv2.flip(frame, 1)
12        annotated_image = frame.copy()
13        results = self.detector.process(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
14        if results.multi_hand_landmarks is not None:
15            for hand_landmarks in results.multi_hand_landmarks:
16                hand = []
17                self.mp_drawing.draw_landmarks(
18                    annotated_image,
19                    hand_landmarks,
20                    self.mp_hands.HAND_CONNECTIONS,
21                    self.mp_drawing_styles.get_default_hand_landmarks_style(),
22                    self.mp_drawing_styles.get_default_hand_connections_style())
23                for landmark in hand_landmarks.landmark:
24                    x, y, z = landmark.x, landmark.y, landmark.z
25                    hand.extend([x, y, z])
26                hands.append(hand)
27        return hands, annotated_image

```

- **Chức năng:** Phát hiện bàn tay trong khung hình và trích xuất các landmarks.
- **Method `__init__`:** Khởi tạo các thành phần cần thiết của MediaPipe để nhận diện bàn tay.
- **Method `detectHand`:**
  - Chuyển đổi frame từ BGR sang RGB.
  - Sử dụng MediaPipe để nhận diện bàn tay và trích xuất các landmarks.
  - Vẽ các landmarks và kết nối lên hình ảnh để hiển thị.
  - Trả về danh sách các landmarks và hình ảnh đã vẽ.

### 3.6. run function:

```

1 def run(data_path, sign_img_path, split="val", resolution=(1280, 720)):
2
3     hand_detector = HandLandmarksDetector()
4     cam = cv2.VideoCapture(0)
5     cam.set(3, resolution[0])
6     cam.set(4, resolution[1])
7

```



```

8     os.makedirs(data_path, exist_ok=True)
9     os.makedirs(sign_img_path, exist_ok=True)
10    print(sign_img_path)
11    dataset_path = f"./{data_path}/landmark_{split}.csv"
12    hand_dataset = HandDatasetWriter(dataset_path)
13    current_letter= None
14    status_text = None
15    cannot_switch_char = False
16
17
18    saved_frame = None
19    while cam.isOpened():
20        _, frame = cam.read()
21        hands, annotated_image = hand_detector.detectHand(frame)
22
23        if(current_letter is None):
24            status_text = "press a character to record"
25
26        else:
27            label = ord(current_letter)-ord("a")
28            if label == -65:
29                status_text = f"Recording unknown, press spacebar again to stop"
30                label = -1
31            else:
32                status_text = f"Recording {LABEL_TAG[label]}, press {current_letter}
again to stop"
33
34            key = cv2.waitKey(1)
35            if(key == -1):
36                if(current_letter is None ):
37                    # no current letter recording, just skip it
38                    pass
39                else:
40                    if len(hands) != 0:
41                        hand = hands[0]
42                        hand_dataset.add(hand=hand, label=label)
43                        saved_frame = frame
44                    # some key is pressed
45                else:
46                    # pressed some key, do not push this image, assign current letter to the
key just pressed
47                    key = chr(key)
48                    if key == "q":
49                        break
50                    if (is_handsign_character(key)):
51                        if(current_letter is None):
52                            current_letter = key
53                        elif(current_letter == key):
54                            # pressed again?, reset the current state
55                            if saved_frame is not None:
56                                if label >=0:
57                                    cv2.imwrite(f"./{sign_img_path}/{LABEL_TAG[label]}.jpg",
saved_frame)
58
59                                cannot_switch_char=False
60                                current_letter = None
61                                saved_frame = None
62                        else:
63                            cannot_switch_char = True
64                            # warned user to unbind the current_letter first

```

```

65         if (cannot_switch_char):
66             cv2.putText(annotated_image, f"please press {current_letter} again to
unbind", (0,450), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
67
68             cv2.putText(annotated_image, status_text, (5,20), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2, cv2.LINE_AA)
69             cv2.imshow(f"{split}", annotated_image)
70             cv2.destroyAllWindows()

```

- **Chức năng:** Vòng lặp chính để thu thập dữ liệu cử chỉ.
- **Tham số:**
  - *data\_path*: Đường dẫn lưu trữ dữ liệu CSV.
  - *sign\_img\_path*: Đường dẫn lưu trữ hình ảnh cử chỉ. **NOTE:** Chỉ có 1 ảnh cuối cùng của mỗi class sẽ được lưu vào đường dẫn để người dùng kiểm tra xem ảnh và class có đúng không.
  - *split*: Loại dữ liệu (train, val, test).
  - *resolution*: Độ phân giải của webcam.
- **Flow:**
  - Khởi tạo HandLandmarksDetector() để nhận diện bàn tay và mở webcam.
  - Tạo các thư mục cần thiết.
  - Khởi tạo các biến trạng thái.
  - Main loop:
    - \* Đọc frame từ webcam.
    - \* Nhận diện bàn tay và trích xuất landmarks.
    - \* Xử lý trạng thái ghi dữ liệu dựa trên phím bấm.  
**NOTE::** Các landmarks và connection giữa các landmarks sẽ được vẽ lên frame và chúng ta có thể quan sát được các landmark này. Từ đó, chúng ta sẽ sử dụng các phím để ghi nhận các class cử chỉ tương ứng. Ví dụ, với 5 class cử chỉ, chúng ta bắt đầu bằng cử chỉ class 0 và nhấn phím 'a'. Sau đó, di chuyển bàn tay nhưng giữ nguyên dáng cử chỉ class 0 để camera liên tục chụp ảnh và thu thập các điểm đánh dấu của cử chỉ này vào file CSV. Quá trình này thường kéo dài từ 10 đến 30 giây cho mỗi class. Khi đã thu thập đủ dữ liệu cho class 0, nhấn 'a' một lần nữa để thoát chế độ thu thập dữ liệu của class đó. Tiếp theo, nhấn 'b' để bắt đầu thu thập dữ liệu cho cử chỉ của class 1 và lặp lại quy trình tương tự cho các class còn lại. Sau khi thu thập xong ta sẽ nhấn phím 'q' để thoát chương trình
    - \* Ghi dữ liệu khi cần thiết.
    - \* Hiển thị thông tin lên màn hình.
  - Kết thúc và giải phóng tài nguyên.

### 3.7. main function:

```

1 if __name__ == "__main__":
2     LABEL_TAG = label_dict_from_config_file("hand_gesture.yaml")
3     data_path = './data2'
4     sign_img_path = './sign_imgs2'
5     run(data_path, sign_img_path, "train", (1280, 720))
6     run(data_path, sign_img_path, "val", (1280, 720))
7     run(data_path, sign_img_path, "test", (1280, 720))

```

- **Chức năng:** Điểm bắt đầu của chương trình.
- **Flow:**
  - Đọc class cử chỉ từ file `hand_gesture.yaml`.
  - Thiết lập đường dẫn lưu trữ dữ liệu và hình ảnh.
  - Gọi hàm `run` 3 lần để xây dựng `train`, `val`, `test data`.

### 3.8. Cách sử dụng `generate_landmark_data.py` để xây dựng `train/val/test data`:

#### 1. Khởi Chạy Chương Trình:

- Mở terminal và chạy lệnh

```
1 python generate_landmark_data.py
```

#### 2. Thu Thập Dữ Liệu

- **Bắt đầu ghi dữ liệu cho một cử chỉ:**
  - Nhấn phím tương ứng với cử chỉ muốn ghi (từ 'a' đến trước 'q').
  - Ví dụ: Nhấn phím 'a' để bắt đầu ghi dữ liệu cho cử chỉ có class 0 như đã định nghĩa ở file config `hand_gesture.yaml`.
- **Thực hiện cử chỉ trước camera:**
  - Đưa tay vào khung hình và thực hiện cử chỉ tương ứng.
  - Dữ liệu sẽ được tự động ghi khi cử chỉ đang được ghi và bàn tay được phát hiện.
- **Kết thúc ghi dữ liệu cho một cử chỉ:**
  - Nhấn lại phím đã chọn để dừng ghi dữ liệu cho cử chỉ đó.
- **Chuyển sang cử chỉ khác:**
  - Lặp lại quá trình trên cho các cử chỉ khác.
- **Kết thúc quá trình thu thập dữ liệu cho `train` hoặc `val` hoặc `test`:**
  - Sau khi thu thập đúng theo số lượng class trong file config `hand_gesture.yaml`. Các bạn nhấn phím 'q' kết thúc.
  - Chúng ta sẽ thực công việc như trên 3 lần tương ứng với `train`, `val` và `test data`. Khi `train` kết thúc sẽ tự động đến `val`, rồi đến `test` sau đó sẽ tự thoát chương trình và ta thu được 3 file CSV của 3 tập data.

#### 3. Lưu Ý Khi Thu Thập Dữ Liệu

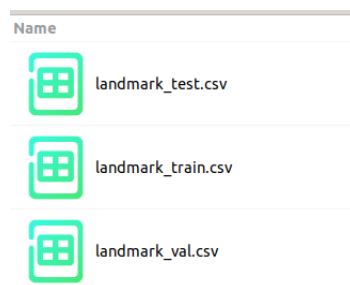
- **Không thể chuyển đổi cử chỉ khi đang ghi:**
  - Nếu muốn chuyển sang cử chỉ khác, cần nhấn lại phím hiện tại để dừng ghi, sau đó nhấn phím mới.
- **Hiển thị thông tin trạng thái:**
  - Trạng thái hiện tại sẽ được hiển thị trên màn hình, giúp người dùng biết đang ghi cử chỉ nào.
- **Lưu hình ảnh mẫu của cử chỉ:**
  - Hình ảnh cuối cùng của cử chỉ sẽ được lưu lại trong thư mục `sign_imgs` để tham khảo.
- **Trong trường hợp muốn thoát nhanh chương trình:**

- Nhấn 'q' liên tục cho đến khi thoát khỏi chương trình và xóa các file data để thực hiện lại từ đầu.

#### 4. Dữ Liệu Được Lưu Trữ

- **File CSV:**

- Dữ liệu landmarks và nhãn cử chỉ được lưu trong các file CSV trong thư mục data.
- Có ba file tương ứng với các loại dữ liệu: landmark\_train.csv, landmark\_val.csv, landmark\_test.csv



Hình 4: Sau khi chạy chương trình sẽ thu được train, val và test CSV file

- **Hình ảnh mẫu:**

- Hình ảnh của cử chỉ được lưu trong thư mục sign\_imgs.

File generate\_landmark\_data.py là công cụ quan trọng giúp chúng ta thu thập và chuẩn bị dữ liệu cần thiết cho việc huấn luyện mô hình nhận diện cử chỉ tay. Bằng cách sử dụng script này, chúng ta có thể:

- Tự động hóa quá trình thu thập dữ liệu: Giảm thiểu công sức và thời gian so với việc ghi dữ liệu thủ công.
- Đảm bảo tính nhất quán của dữ liệu: Dữ liệu được thu thập và lưu trữ theo một định dạng chuẩn, dễ dàng cho việc tiền xử lý và huấn luyện mô hình.
- Dễ dàng mở rộng và tùy chỉnh: Có thể thêm hoặc chỉnh sửa các cử chỉ bằng cách cập nhật file hand\_gesture.yaml và thu thập dữ liệu mới.

Tiếp theo, sau khi đã thu thập và chuẩn bị dữ liệu, chúng ta sẽ chuyển sang Bước 1 để xây dựng và huấn luyện model phân loại cử chỉ tay bằng cách xây dựng MLP (Multi-Layer Perceptron) model đơn giản. Model này sẽ học từ dữ liệu chúng ta đã thu thập

### Step 1: Xây dựng và huấn luyện mô hình phân loại cử chỉ tay

Sau khi đã hoàn thành việc chuẩn bị dữ liệu ở bước 0, chúng ta chuyển sang bước tiếp theo là xây dựng và huấn luyện mô hình phân loại cử chỉ tay sử dụng MLP (Multi-Layer Perceptron). Mục tiêu của bước này là tạo ra một mô hình có khả năng nhận diện chính xác các cử chỉ tay dựa trên dữ liệu landmarks thu thập được. File notebook hand\_gesture\_recognition huấn luyện mô hình MLP dựa trên cấu hình các class cử chỉ trong hand\_gesture.yaml và dữ liệu landmarks từ các file CSV để học cách phân loại cử chỉ tay.

#### 1. Quy Trình Huấn Luyện

##### 1. Load dữ liệu

- **Dữ liệu được chia thành ba tập:** train, val, và test đã được thu thập ở step 0. Nếu các bạn không thực hiện được thì có thể sử dụng data được cung cấp link download ở trong notebook
- Sử dụng class CustomImageDataset để đọc dữ liệu từ các file CSV và tạo bộ dữ liệu cho PyTorch.

## 2. Xây Dựng Mô Hình MLP

- Mô hình gồm nhiều layer Linear và ReLU để học các đặc trưng phức tạp từ dữ liệu.
- Sử dụng BatchNorm và Dropout để cải thiện hiệu suất và ngăn overfitting.

## 3. Huấn Luyện Mô Hình

- Sử dụng hàm loss CrossEntropyLoss
- Sử dụng bộ Adam optimizer
- Sử dụng Early Stopping để dừng huấn luyện khi mô hình không còn cải thiện trên tập val.
- Theo dõi accuracy của train và val sau mỗi epoch.

## 4. Đánh Giá Mô Hình

- Sau khi huấn luyện, sử dụng mô hình tốt nhất để đánh giá trên tập test.

**NOTE:** Các bạn phải hoàn thành được các câu hỏi bên dưới thì sẽ thực hiện được viên train mô hình để phân loại cử chỉ

**Câu hỏi 1:** Hoàn thành đoạn code bên dưới để xây dựng một model gồm có 4 hidden layer, lần lượt input và output là (63, 128), (128, 128), (128, 128), (128, 128). Layer đầu tiên được theo sau bởi một Relu và Batchnorm1d. Layer thứ 2, 3, và 4 được theo sau bởi Relu và Dropout với rate lần lượt là 0.4, 0.4, 0.6. Output layer có nhiệm vụ phân loại với input là 128 và output là số lượng class cử chỉ:

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        list_label = label_dict_from_config_file("hand_gesture.yaml")

        self.linear_relu_stack = nn.Sequential(
            ##### Your Code Here #####
            '''Hoàn thành đoạn code để xây dựng một model gồm có 4 hidden layer,
            lần lượt input và output là (63, 128), (128, 128), (128, 128), (128, 128).
            Layer đầu tiên được theo sau bởi một Relu và Batchnorm1d.
            Layer thứ 2, 3, và 4 được theo sau bởi Relu và Dropout với rate lần lượt là 0.4, 0.4, 0.6.
            Output layer có nhiệm vụ phân loại với input là 128 và output là số lượng class cử chỉ
            '''
            #####
        )
```

(A)

```
1 nn.Linear(63, 128),
2 nn.ReLU(),
3 nn.BatchNorm1d(128),
4 nn.Linear(128, 128),
5 nn.ReLU(),
```

```

6 nn.Dropout(p=0.4),
7 nn.Linear(128, 128),
8 nn.ReLU(),
9 nn.Dropout(p=0.4),
10 nn.Linear(128, 128),
11 nn.ReLU(),
12 nn.Dropout(p=0.6),
13 nn.Linear(128, len(list_label)),

```

(B)

```

1 nn.Linear(63, 128),
2 nn.ReLU(),
3 nn.Linear(128, 128),
4 nn.ReLU(),
5 nn.BatchNorm1d(128),
6 nn.Dropout(p=0.5),
7 nn.Linear(128, 128),
8 nn.ReLU(),
9 nn.Linear(128, len(list_label)),
10 nn.Dropout(p=0.3),

```

(C)

```

1 nn.Linear(63, 128),
2 nn.ReLU(),
3 nn.BatchNorm1d(128),
4 nn.Linear(128, 128),
5 nn.Dropout(p=0.3),
6 nn.ReLU(),
7 nn.Linear(128, 128),
8 nn.ReLU(),
9 nn.Dropout(p=0.5),
10 nn.Linear(128, 128),
11 nn.ReLU(),
12 nn.Linear(128, len(list_label)),

```

(D)

```

1 nn.Linear(63, 256),
2 nn.ReLU(),
3 nn.BatchNorm1d(256),
4 nn.Linear(256, 256),
5 nn.ReLU(),
6 nn.Dropout(p=0.4),
7 nn.Linear(256, 128),
8 nn.ReLU(),
9 nn.Dropout(p=0.4),
10 nn.Linear(128, 128),
11 nn.ReLU(),
12 nn.Dropout(p=0.6),
13 nn.Linear(128, len(list_label)),

```

**Câu hỏi 2:** Hoàn thành code để thực hiện forward dự đoán chỉ với input x. Thực hiện flatten x. Pass x vừa flatten vào linear\_relu\_stack. Return logits (outputs từ layer cuối cùng)

(A)

```

1 x = self.flatten(x)
2 x = self.linear_relu_stack(x)
3 return x

```

(B)

```
def forward(self, x):
    ##### Your Code Here ##### Q2
    ''' Hoàn thành code để thực hiện forward dự đoán cu' chỉ với input x.
    Thực hiện flatten x
    Pass x vừa flatten vào linear_relu_stack
    Return logits (outputs từ layer cuối cùng)
    '''
    #####
```

```
1 x = self.linear_relu_stack(x)
2 return self.flatten(x)
```

(C)

```
1
2 return self.linear_relu_stack(x)
```

(D)

```
1 x = self.flatten(x)
2 self.linear_relu_stack.forward(x)
3 return
```

**Câu hỏi 3:** Hoàn thành code để thực hiện khởi tạo DataLoader cho trainset với batch\_size 40 và cho phép xáo trộn

```
trainset = CustomImageDataset(train_path)
##### Your Code Here ##### Q3
''' Hoàn thành code để thực hiện khởi tạo DataLoader cho trainset với
batch_size 40 và cho phép xáo trộn
'''
trainloader =
#####
```

(A)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset, batch_size=40, shuffle=True)
```

(B)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset, batch_size=50, shuffle=True)
```

(C)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=False)
```

(D)

```
1
2 trainloader = torch.utils.data.DataLoader(dataset=trainset, batch_size=40, shuffle=False)
```

**Câu hỏi 4:** Hoàn thành code để thực hiện cấu hình Adam optimizer cho các tham số của model với tốc độ học là 0.0001

(A)

```
1
2 optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

```
##### Your Code Here #####
'''Hoàn thành code để thực hiện câu hình Adam optimizer cho các tham số của
model với tốc độ học là 0.0001
'''
optimizer =
#####
```

(B)

```
1
2 optimizer = optim.Adam(model, lr=0.0001)
```

(C)

```
1
2 optimizer = optim.Adam(model.parameters, lr=0.0001)
```

(D)

```
1
2 optimizer = optim.Adam([model.parameters], lr=0.0001)
```

**Câu hỏi 5:** Hoàn thành đoạn code bên dưới để print ra epoch hiện tại và minimum watched metric và thoát loop (A)

```
if early_stopper.early_stop(avg_vloss):
    ##### Your Code Here ##### Q5
    ''' Hoàn thành đoạn code bên dưới để print ra epoch hiện tại và
    minimum watched metric và thoát loop
    '''

    #####
```

```
1
2 print(f"stopping at epoch {epoch}, minimum: {early_stopper.watched_metrics}")
3 break
```

(B)

```
1
2 print("Early stopping triggered")
3 break
```

(C)

```
1
2 print(f"Stopping training at epoch {epoch} due to minimal loss change")
```

(D)

```
1
2 print(f"stopping at epoch {epoch}, minimum: {early_stopper.watched_metrics}")
```

**Câu hỏi 6:** Hoàn thành code để thực hiện khởi tạo DataLoader cho testset với batch\_size 20 và không cho phép xáo trộn

(A)

```
1
2 test_loader = torch.utils.data.DataLoader(testset, batch_size=20, shuffle=False)
```



```
def forward(self, x):
    ##### Your Code Here ##### Q2
    ''' Hoàn thành code để thực hiện forward dự đoán cu'chỉ với input x.
    Thực hiện flatten x
    Pass x vừa flatten vào linear_relu_stack
    Return logits (outputs từ layer cuối cùng)
    '''
    #####
```

(B)

```
1
2 test_loader = torch.utils.data.DataLoader(testset, batch_size=20, shuffle=True)
```

(C)

```
1
2 test_loader = torch.utils.data.DataLoader(valet, batch_size=20, shuffle=False)
```

(D)

```
1
2 test_loader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=False)
```

**Câu hỏi 7:** Hoàn thành code bên dưới để predict class của cu'chỉ và update accuracy với kết quả predict và true labels

```
##### Your Code Here ##### Q7
''' Hoàn thành code bên dưới để predict class của cu'chỉ và update accuracy
với with kết quả predict và true labels
'''

preds =

#####
```

(A)

```
1
2 preds = network(test_input)
3 acc_test.update(preds, test_label)
```

(B)

```
1
2 preds = network(train_input)
3 acc_test.update(preds, test_label)
```

(C)

```
1
2 preds = network(val_input)
3 acc_test.update(preds, test_label)
```

(D)

```
1
2 preds = network(train_input)
3 acc_test.update(preds, val_label)
```

**Câu hỏi 8:** Hoàn thành code để thực hiện khởi tạo NeuralNetwork model đã xây dựng ở trên, khởi tạo hàm loss sử dụng CrossEntropyLoss và khởi tạo early stopper với patience là 30 và min\_delta là 0.01

(A)

```
##### Your Code Here ##### Q8
''' Hoàn thành code để thực hiện khởi tạo NeuralNetwork model đã xây dựng ở trên,
khởi tạo hàm loss sử dụng CrossEntropyLoss và khởi tạo early stopper với patience
là 30 và min_delta là 0.01
'''
model =
loss_function =
early_stopper =
#####
```

```
1
2 model = NeuralNetwork()
3 loss_function = nn.CrossEntropyLoss()
4 early_stopper = EarlyStopper(patience=30,min_delta=0.01)
```

(B)

```
1 model = nn.CrossEntropyLoss()
2 loss_function = NeuralNetwork()
3 early_stopper = EarlyStopper(patience=30,min_delta=0.01)
```

(C)

```
1 model = NeuralNetwork()
2 loss_function = EarlyStopper(patience=30,min_delta=0.01)
3 early_stopper = nn.CrossEntropyLoss()
```

(D)

```
1 model = EarlyStopper(patience=30,min_delta=0.01)
2 loss_function = nn.CrossEntropyLoss()
3 early_stopper = NeuralNetwork()
```

**Câu hỏi 9:** Hoàn thành code để thực hiện reset gradients và dự đoán class cử chỉ của inputs

```
##### Your Code Here ##### Q9
''' Hoàn thành code để thực hiện reset gradients và dự đoán class cử
chỉ của inputs
'''

preds =
#####
```

(A)

```
1
2 optimizer.zero_grad()
3 preds = model(inputs)
```

(B)

```
1
2 optimizer.clear()
3 preds = model(inputs)
```

(C)

```
1
2 optimizer.clean()
3 preds = model(inputs)
```

(D)

```

1
2 optimizer.reset()
3 preds = model(labels)

```

**Câu hỏi 10:** Hoàn thành code để thực hiện tính loss dựa vào kết quả dự đoán và labels, sau đó thực hiện backward và update parameters thông qua optimizer

```

##### Your Code Here ##### Q10
''' Hoàn thành code để thực hiện tính loss dựa vào kết quả dự đoán
và labels, sau đó thực hiện backward và update parameters thông qua
optimizer
'''

loss =

#####

```

(A)

```

1
2 loss = loss_function(preds, labels)
3 loss.backward()
4 optimizer.step()

```

(B)

```

1
2 loss.backward()
3 loss = loss_function(preds, labels)
4 optimizer.update()

```

(C)

```

1
2 loss = loss_function(preds, labels)
3 optimizer.backward()
4 loss.step()

```

(D)

```

1
2 optimizer.update()
3 loss = loss_function(preds, labels)
4 loss.backward()

```

Việc huấn luyện model thành công là nền tảng để chúng ta chuyển sang bước 2, nơi mô hình sẽ được triển khai để nhận diện cử chỉ tay trong thời gian thực và điều khiển đèn theo yêu cầu.

## Step 2: Triển khai model và điều khiển đèn bằng cử chỉ tay

Sau khi đã huấn luyện thành công mô hình phân loại cử chỉ tay ở bước 1, chúng ta chuyển sang bước cuối cùng của dự án: triển khai hệ thống nhận diện cử chỉ tay trong thời gian thực và điều khiển đèn theo cử chỉ. Bước này bao gồm việc sử dụng webcam để thu nhận hình ảnh, nhận diện cử chỉ tay thông qua mô hình đã huấn luyện, và thực hiện điều khiển đèn tương ứng.

Trong phần này, chúng ta sẽ phân tích chi tiết hai file chính:

- `detect_simulation.py`: File chính để nhận diện cử chỉ và điều khiển đèn.
- `controller.py`: Module hỗ trợ việc điều khiển thiết bị phần cứng thông qua giao tiếp Modbus RTU RS485.

- Mục Tiêu của Bước 2

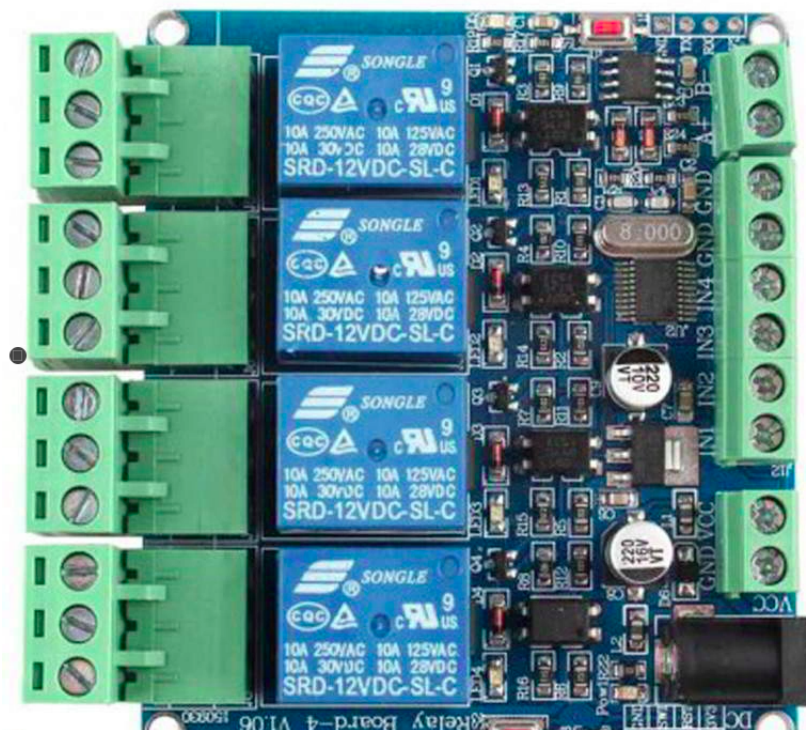
- **Nhận diện cử chỉ tay trong thời gian thực:** Sử dụng webcam để thu nhận hình ảnh và áp dụng mô hình đã huấn luyện để nhận diện cử chỉ.
- **Điều khiển đèn dựa trên cử chỉ:** Thực hiện thao tác tắt/mở đèn trong mô phỏng hoặc điều khiển thiết bị đèn thực tế thông qua module relay và giao tiếp Modbus RTU RS485.
- **Cung cấp tùy chọn cho người dùng:** Cho phép chạy ở chế độ mô phỏng (không cần phần cứng) hoặc chế độ thực tế (có phần cứng).

- Tùy Chọn Chế Độ Hoạt Động

- **Chế độ mô phỏng** (device=False): Nếu bạn không có phần cứng, chương trình sẽ hiển thị mô phỏng ba đèn trên khung hình webcam.
- **Chế độ thực tế** (device=True): Nếu bạn có phần cứng (module 4 relay giao tiếp Modbus RTU RS485), chương trình sẽ điều khiển đèn thực tế thông qua module này.

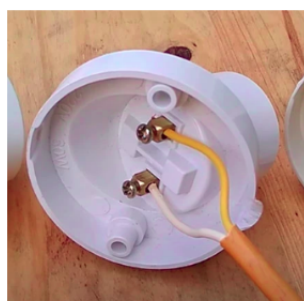
- Phần cứng (trong trường hợp các bạn muốn chạy chế độ thực tế)

- **4 relay giao tiếp Modbus RTU RS485:** Dùng để nhận tín hiệu từ laptop/PC chạy model để điều khiển 3 đèn tắt/mở



- **Mạch chuyển đổi giao tiếp USB to RS485:** Giúp giao tiếp giữa laptop/PC với module 4 relay giao tiếp Modbus RTU RS485
- **3 bóng đèn và chuỗi đèn:** dùng chuỗi đèn gắn vào bóng đèn và nối dây vào module 4 relay giao tiếp Modbus RTU RS485

Trong file detect\_simulation.py thì class LightGesture có nhiệm vụ nhận diện ra cử chỉ của tay và điều khiển đèn



### • Thuộc tính:

- device: Biến boolean xác định chế độ hoạt động (mô phỏng hoặc thực tế).
- detector: Đối tượng HandLandmarksDetector để nhận diện bàn tay.
- signs: Dictionary chứa các nhãn cử chỉ từ file hand\_gesture.yaml.
- classifier: Mô hình NeuralNetwork đã được tải trọng số từ quá trình huấn luyện.
- controller: Đối tượng ModbusMaster để điều khiển phần cứng (chỉ khởi tạo khi device=True).
- light1, light2, light3: Trạng thái của các đèn (True là bật, False là tắt).

### • Method run

- Khởi tạo webcam và thiết lập độ phân giải.
- Main Loop:
  - \* Đọc frame từ webcam.
  - \* Sử dụng detector để nhận diện bàn tay và trích xuất landmarks.
  - \* Nếu phát hiện bàn tay:
    - Chuyển landmarks thành tensor và đưa vào mô hình classifier để dự đoán cử chỉ.
    - Dựa trên class cử chỉ dự đoán, cập nhật trạng thái các đèn trên mô phỏng và điều khiển phần cứng (nếu có).
    - Cập nhật status\_text để hiển thị cử chỉ hiện tại.
  - \* Nếu không phát hiện bàn tay: Đặt status\_text là None.
  - \* Gọi method light\_device để vẽ mô phỏng đèn lên khung hình.
  - \* Hiển thị khung hình và status\_text

\* Kiểm tra phím bấm để thoát chương trình (nhấn phím "q" để thoát).

Hàm main: Tạo đối tượng LightGesture và chạy phương thức run.

- Thay thế model\_path bằng đường dẫn tới mô hình đã huấn luyện và lưu trữ ở bước 1.
- Đặt device=False để chạy ở chế độ mô phỏng, device=True để chạy ở chế độ thực tế.

## Phần III: Phụ Lục:

1. Code để thực hiện **step 0** [Link](#), Video hướng dẫn sử dụng generate\_landmark\_data.py để tạo train/val/test data [Link](#)
2. Code notebook (hint) [Link](#) để thực hiện **step 1** các bạn cần upload file data chạy được ở step 1 (nếu không chạy được ở step 0 các bạn có thể download data ở [Link](#)) và file config hand\_gesture.yaml để chạy với notebook.
3. Code để thực hiện **step 2** . [Link](#). Video demo kết quả cuối cùng với thiết bị sử dụng file detect\_simulation.py (cùng là một file với main.py trong demo) [Link](#)
4. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
5. **Rubric:**

Rubric Dự Án Điều Khiển Đèn Bằng Cử Chỉ Tay		
Bước	Kiến Thức	Đánh Giá
0	<ul style="list-style-type: none"> <li>- Thu thập và chuẩn bị dữ liệu cử chỉ tay với MediaPipe</li> <li>- Lưu trữ dữ liệu landmarks vào file CSV</li> <li>- Cấu hình các class cử chỉ trong file YAML</li> <li>- Viết code Python để tự động hóa quá trình thu thập dữ liệu</li> </ul>	<ul style="list-style-type: none"> <li>- Áp dụng kiến thức xử lý hình ảnh để thu thập dữ liệu</li> <li>- Sử dụng OpenCV và MediaPipe để nhận diện bàn tay</li> <li>- Quản lý dữ liệu bằng YAML và CSV</li> <li>- Viết và chạy được code thu thập dữ liệu hiệu quả</li> </ul>
1	<ul style="list-style-type: none"> <li>- Xây dựng và huấn luyện mô hình MLP phân loại cử chỉ tay</li> <li>- Sử dụng PyTorch để xây dựng mô hình</li> <li>- Đánh giá mô hình qua độ chính xác trên tập validation và test</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu và xây dựng mô hình MLP cho phân loại cử chỉ</li> <li>- Sử dụng PyTorch để huấn luyện và đánh giá mô hình</li> </ul>
2	<ul style="list-style-type: none"> <li>- Triển khai hệ thống nhận diện cử chỉ thời gian thực</li> <li>- Kết hợp mô hình với luồng video từ webcam</li> <li>- Điều khiển đèn qua giao tiếp Modbus RTU RS485</li> <li>- Tùy chỉnh chương trình cho chế độ mô phỏng hoặc thực tế</li> </ul>	<ul style="list-style-type: none"> <li>- Áp dụng lập trình xử lý video</li> <li>- Kết nối và điều khiển thiết bị phần cứng</li> </ul>

— Hết —