Hand Gesture Recognition Project

SRS Report

Shashank

[Company Name]

Overview:

The aim of this project is to help deaf people communicate easily with other people. Hand gesture recognition is a technology that enables a computer to interpret and understand the gestures made by a person using their hands. This project covers various domains, including human-computer interaction, virtual reality, gaming, robotics and healthcare.
The goal of this project is to develop algorithms and systems that can accurately identify and interpret hand movements, allowing for intuitive and natural interaction between humans and machines.

- **Data Collection:**

➔ Hand gesture recognition systems often start with the collection of a dataset containing images or videos of hand gestures.
➔ This dataset is very crucial for training machine learning models.

- **Image Acquisition:**

➔ This system captures images or video frames of the hand using cameras or other imaging devices.
➔ These images serve as input data for the gesture recognition algorithm.

- **Pre-processing;**

➔ Raw image data may undergo pre-processing techniques such as normalization, resizing and noise reduction to enhance the quality of input data and make it suitable for further analysis.

• **Feature extraction:**
➔ Extracting relevant features from the input data is

essential for representing the key characteristics of hand

gestures.
➔ Features may include hand shape, finger position and

motion trajectories.

• **Gesture Recognition Algorithms:**
➔ Various machine learning and computer vision algorithms

are employed for recognizing hand gestures. These are approaches used:

I. II.

Deep Learning: Convolution Neural Networks(CNNs), Recurrent Neural Networks(RNNs), or combinations of both.
Traditional Computer Vision Techniques: Histogram of Oriented Gradients(HOG), Haar Cascades and other feature-based methods.

• **Training:**
➔ The selected model is trained on the labelled dataset to

learn the patterns and relationships between input

features and corresponding gestures.
➔ This phase involves adjusting the model's parameters to

minimize errors.

• **Validation and Testing:**
➔ The trained model is validated on a separate dataset to

ensure its generalization capabilities. Subsequently, it undergoes testing to evaluate its performance on new, unseen data.

- **Real-time Processing:**

➔ Once trained and validated, the model can be deployed for real-time hand gesture recognition.
➔ This involves processing incoming data(video frames) and predicting the corresponding gestures.

- **Application Integration:**

➔ Hand gesture recognition systems are integrated into applications and devices, such as smart TVs, virtual reality systems, or robotics, to enable users to interact naturally using hand movements.

- **Challenges and considerations:**

➔ Challenges in this project include variation in lighting conditions, occlusions, and the need for robustness to different hand shapes and sizes.
➔ Continuous improvement and adaptation to diverse user scenarios are crucial.

❖ Requirements

- **Software Requirements:**

1. Development Environment:

Any operating system like Windows or Linux that can support the hand gesture recognition software. Integrated development environment(IDEs) eg.Visual Studio code, Pycharm and Jupyter Notebooks.

2. Programming language:

This project will be done on Python due to its extensive libraries and frameworks such as TensorFlow nad OpenCV.

3. Computer Vision Libraries:

Hand gesture detection requires a variety of computer vision features, which libraries like OpenCV oer. Among these are picture processing and feature extraction.

4. Machine learning frameworks:

Machine learning frameworks such as TensorFlow, PyTorch, or Keras are crucial for developing and training gesture recognition models, especially deep learning models.

5. Gesture Recognition SDKs:

Some companies provide software development kits (SDKs) specifically designed for hand gesture recognition. These SDKs may include pre-trained

models, APIs, and documentation to facilitate the integration of gesture recognition into applications.

---

6. User Interface (UI) Development tools.

- **Hardware requirements: 1. Basics:**

A powerful CPU and in cases also a GPU are required for real-time processing of image data and running complex machine learning models.

Random access memory(RAM) us necessary for handling the large datasets and computational requirements of the project

Adequate storage space for storing the datasets, models and application related files.

Connectivity is optionally needed inorder to establish connection between devices for cloud-based processing.

A continuous and stable power supply is a must especially for applications in which the hand gesture recognition system is embedded in a device or integrated into a larger system.

2. Cameras and/or sensors:

High-quality cameras or sensors capable of capturing hand movements accurately are essential to produce accurate results. Depth-sensing cameras are often used for three-dimensional gesture recognition (optional).

❖ Suitable process models:

There are two process models which are most appropriate for this project out of which any one can be implemented depending upon the complexity of the project.

The two process models are:

1.The Waterfall Model

A methodical and systematic approach to software development is the Waterfall Model. It is divided into discrete stages, each of which needs to be finished before going on to the next. The following are the phases:

1. Acquisition of Requirements:

Specify the motions to be identified, as well as the system's goals and user requirements for hand gesture recognition.

2. System Architecture:

Provide a thorough system design that includes the software architecture, gesture detection algorithms, and hardware components (cameras, sensors).

3. Execution:

Based on the design parameters, write the code for the hand gesture recognition system. This involves developing the software, putting the hardware components together, and implementing the selected gesture recognition algorithms into practice.

4. Testing

To make sure the system recognizes hand gestures accurately under varied conditions, thoroughly test the system. Test for robustness, accuracy, and real-time performance.

5. Implementation:

Install the hand gesture recognition system in the designated environment or incorporate it into the targeted device or application.

6. Upkeep/Maintenance:

Regularly help users, resolve problems with support, and create improvements in response to the customers feedback and evolving needs.

7. Benefits

Straightforward and simple to comprehend.
Ideal for projects with certain and consistent criteria.

8. Drawbacks:

Restricted ability to adjust during development to changes.

Results might not be visible to users until the project is finished.

This repository contains the following contents.

- Sample program
- Hand sign recognition model(TFLite)
- Finger gesture recognition model(TFLite)
- Learning data for hand sign recognition and notebook for learning
- Learning data for finger gesture recognition and notebook for learning

Requirements

- mediapipe 0.8.1
- OpenCV 3.4.2 or Later
- Tensorflow 2.3.0 or Later
  tf-nightly 2.5.0.dev or later (Only when creating a TFLite for an LSTM model)
- scikit-learn 0.23.2 or Later (Only if you want to display the confusion matrix)

- matplotlib 3.3.2 or Later (Only if you want to display the confusion matrix)

app.py

This is a sample program for inference.
In addition, learning data (key points) for hand sign recognition,
You can also collect training data (index finger coordinate history) for finger gesture recognition.

keypoint_classification.ipynb

This is a model training script for hand sign recognition.

point_history_classification.ipynb

This is a model training script for finger gesture recognition.

model/keypoint_classifier

This directory stores files related to hand sign recognition.
The following files are stored.

- Training data(keypoint.csv)
- Trained model(keypoint_classifier.tflite)
- Label data(keypoint_classifier_label.csv)
- Inference module(keypoint_classifier.py)

model/point_history_classifier

This directory stores files related to finger gesture recognition.
The following files are stored.

- Training data(point_history.csv)
- Trained model(point_history_classifier.tflite)
- Label data(point_history_classifier_label.csv)
- Inference module(point_history_classifier.py)

utils/cvfpscalc.py

This is a module for FPS measurement.

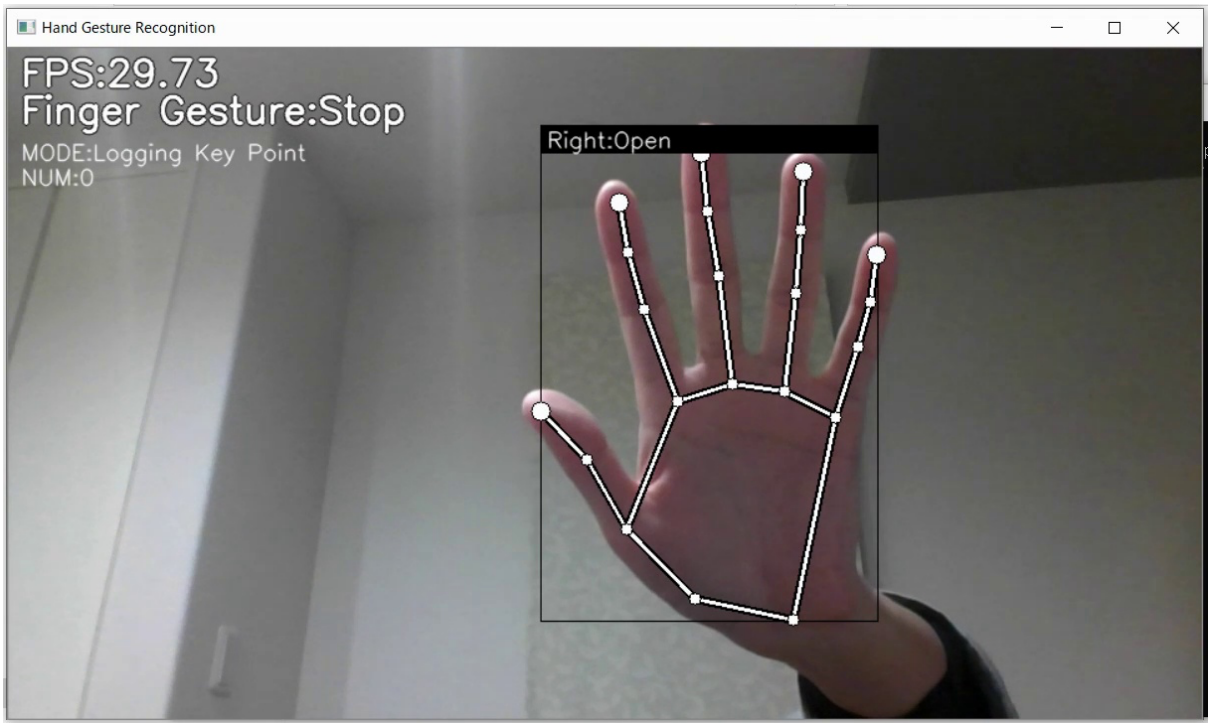
Training

Hand sign recognition and finger gesture recognition can add and change training data and retrain the model.

Hand sign recognition training

1.Learning data collection

Press "k" to enter the mode to save key points（displayed as 「MODE:Logging Key Point」）

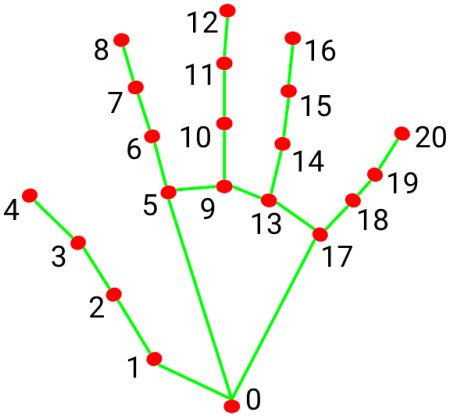


If you press "0" to "9", the key points will be added to "model/keypoint_classifier/keypoint.csv" as shown below.
1st column: Pressed number (used as class ID), 2nd and subsequent columns: Key point coordinates

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1519	0	0	0	-0.21659	0.073733	-0.34101	0.253456	-0.40553	0.419355	-0.40092	0.552995	-0.28571	0.198157	-0.35945	0.479263	-0.37327	0.645161	-0.36866
1520	0	0	0	-0.2287	0.080717	-0.33632	0.255605	-0.38565	0.426009	-0.36323	0.565022	-0.26906	0.179372	-0.33184	0.452915	-0.34978	0.627803	-0.35426
1521	0	0	0	-0.16889	0.048889	-0.21778	0.217778	-0.24444	0.4	-0.24889	0.551111	-0.08	0.151111	-0.06222	0.377778	-0.02667	0.524444	0.013333
1522	0	0	0	-0.16114	0.066351	-0.22275	0.236967	-0.25118	0.417062	-0.24171	0.559242	-0.19431	0.194313	-0.2654	0.469194	-0.2891	0.649289	-0.3128
1523	1	0	0	-0.3	-0.18667	-0.44667	-0.48	-0.46667	-0.76	-0.46667	-1	-0.3	-0.67333	-0.29333	-0.90667	-0.31333	-0.66667	-0.31333
1524	1	0	0	-0.32432	-0.17568	-0.5	-0.4527	-0.5473	-0.74324	-0.56757	-1	-0.41216	-0.65541	-0.39865	-0.91892	-0.38514	-0.67568	-0.38514
1525	1	0	0	-0.33803	-0.16901	-0.54225	-0.43662	-0.59859	-0.73239	-0.61972	-1	-0.4507	-0.67606	-0.43662	-0.93662	-0.41549	-0.6831	-0.41549
1526	1	0	0	-0.34286	-0.15	-0.55	-0.42857	-0.62857	-0.72857	-0.66429	-1	-0.49286	-0.66429	-0.47857	-0.93571	-0.44286	-0.67857	-0.43571

The key point coordinates are the ones that have undergone the following

preprocessing up to ④.



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

① 各キーポイント座標（Landmark coordinates）

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[551, 465]	[485, 428]	[439, 362]	[408, 307]	……	[633, 315]	[668, 261]	[687, 225]	[702, 188]

② ID:0からの相対座標に変換（Convert to relative coordinates from ID:0）

ID：0	ID：1	ID：2	ID：3	……	ID：17	ID：18	ID：19	ID：20
[0, 0]	[-66, -37]	[-112, -103]	[-143, -158]	……	[82, -150]	[117, -204]	[136, -240]	[151, -277]

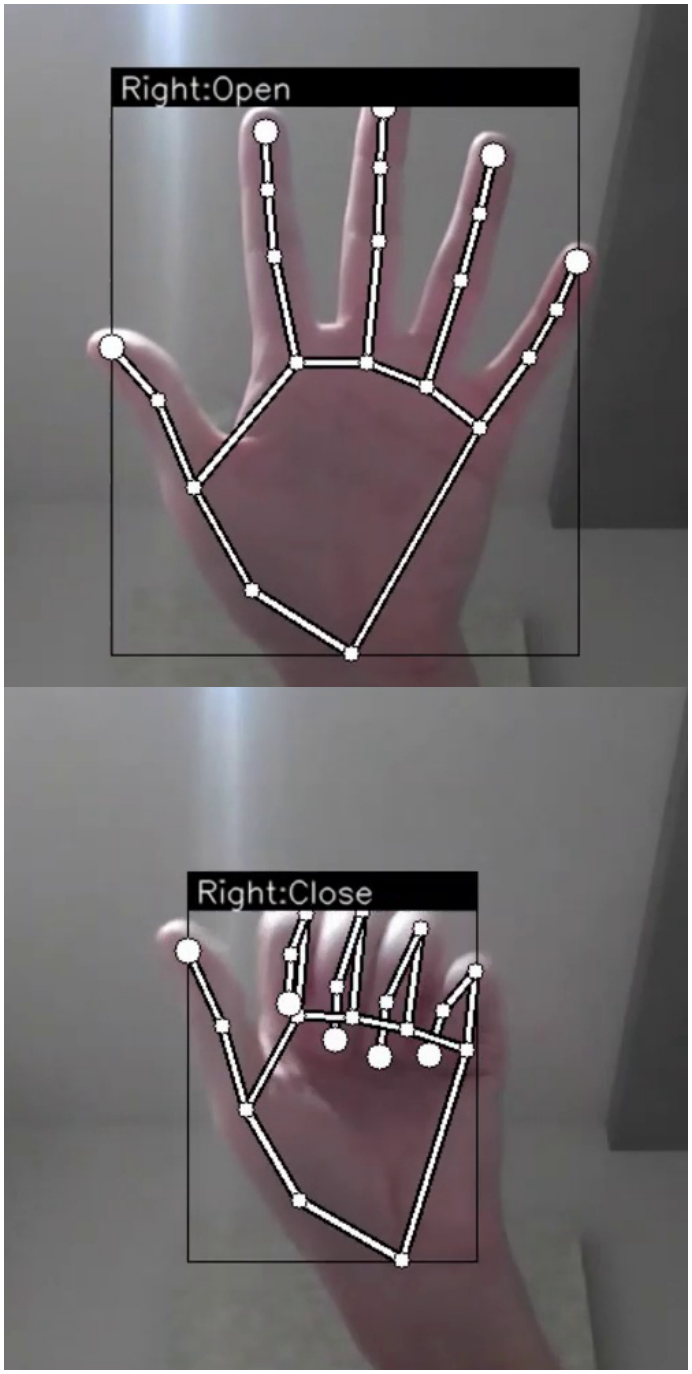
③ 1次元配列に変換（Flatten to a one-dimensional array）

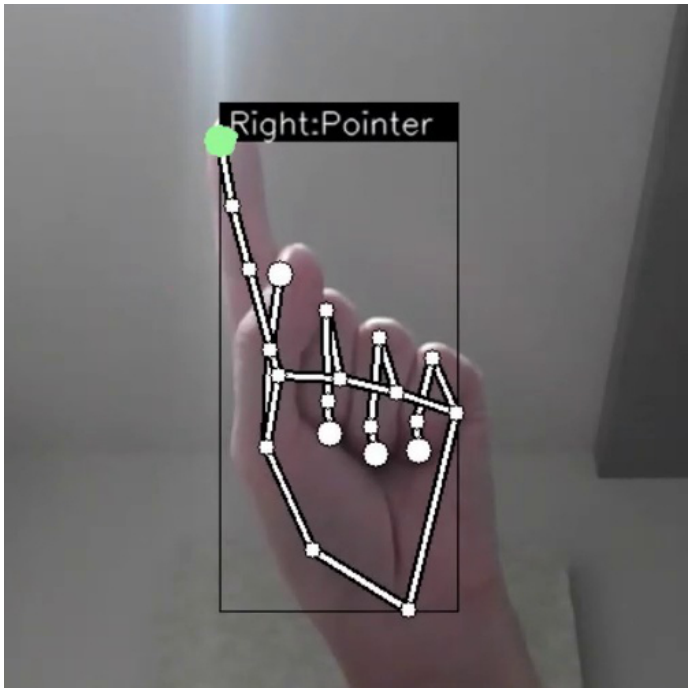
ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-66	-37	-112	-103	-143	-158	……	82	-150	117	-204	136	-240	151	-277

④ 最大値(絶対値)に合わせて正規化（Normalized to the maximum value(absolute value)）

ID：0		ID：1		ID：2		ID：3		……	ID：17		ID：18		ID：19		ID：20	
0	0	-0.24	-0.13	-0.4	-0.37	-0.52	-0.57	……	0.296	-0.54	0.422	-0.74	0.491	-0.87	0.545	-1

In the initial state, three types of learning data are included: open hand (class ID: 0), close hand (class ID: 1), and pointing (class ID: 2).
If necessary, add 3 or later, or delete the existing data of csv to prepare the training data.


Right:Open
Right:Close

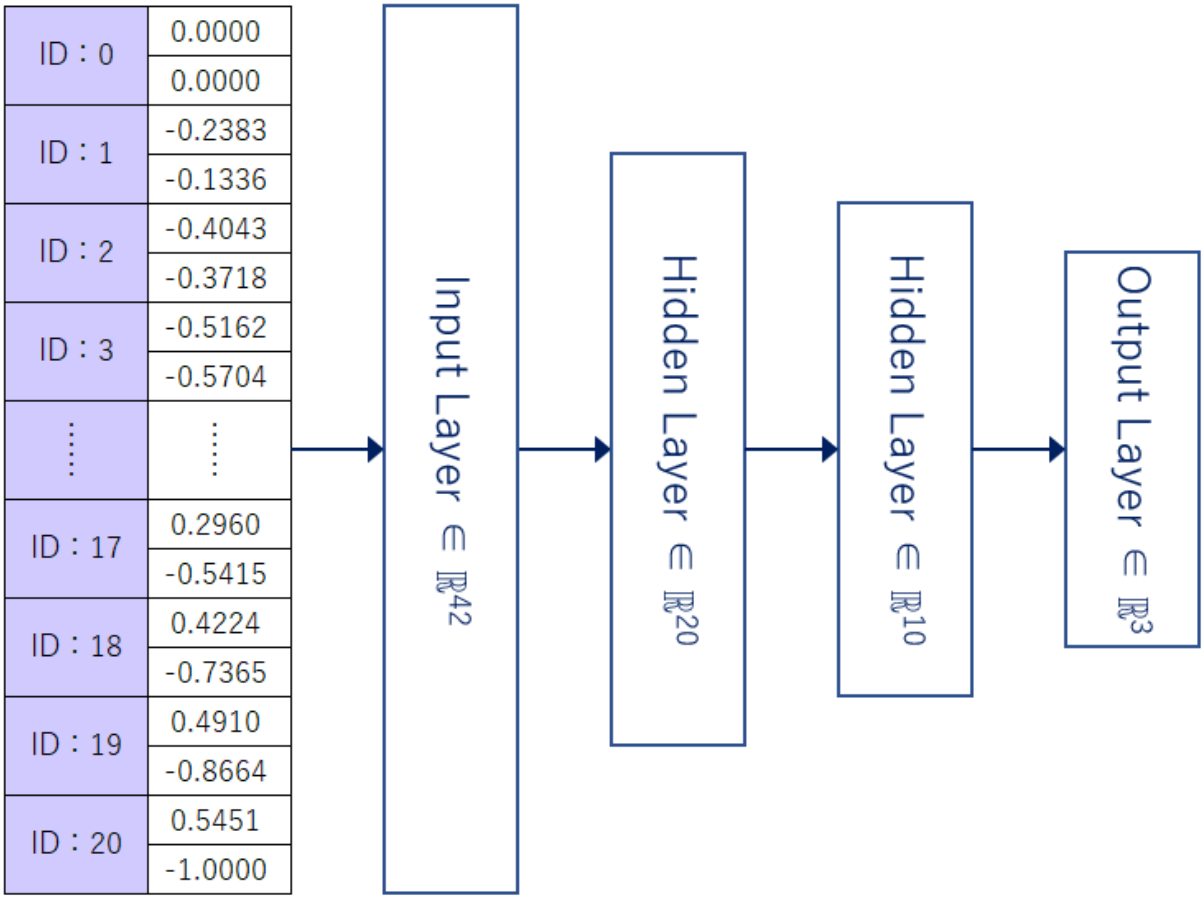


2.Model training

Open "keypoint_classification.ipynb" in Jupyter Notebook and execute from top to bottom.
To change the number of training data classes, change the value of "NUM_CLASSES = 3"
and modify the label of "model/keypoint_classifier/keypoint_classifier_label.csv" as appropriate.

X.Model structure

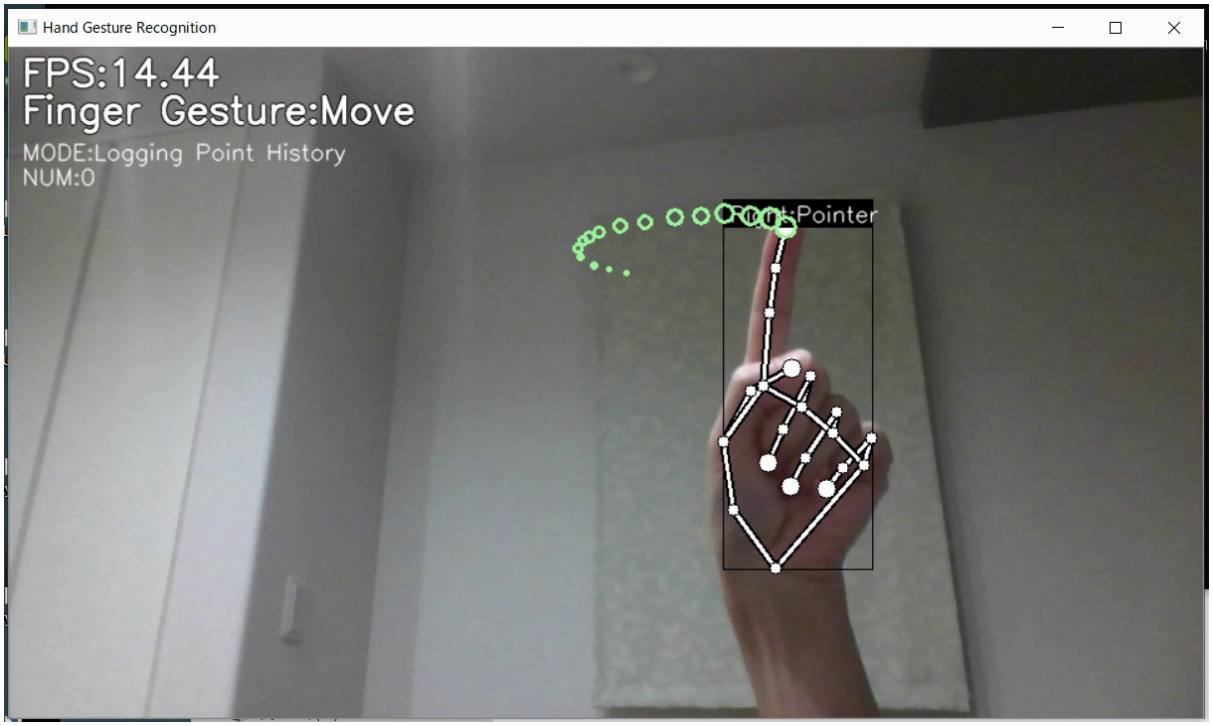
The image of the model prepared in "keypoint_classification.ipynb" is as follows.



Finger gesture recognition training

1.Learning data collection

Press "h" to enter the mode to save the history of fingertip coordinates (displayed as "MODE:Logging Point History").



If you press "0" to "9", the key points will be added to "model/point_history_classifier/point_history.csv" as shown below.
1st column: Pressed number (used as class ID), 2nd and subsequent columns: Coordinate history

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1478	0	0	0	0.001042	0	0	-0.00185	0	-0.0037	0	-0.0037	0	-0.0037	0	-0.0037	0	-0.0037	0
1479	0	0	0	-0.00104	-0.00185	-0.00104	-0.0037	-0.00104	-0.0037	-0.00104	-0.0037	-0.00104	-0.0037	-0.00104	-0.0037	-0.00104	-0.00556	-0.00104
1480	0	0	0	0	-0.00185	0	-0.00185	0	-0.00185	0	-0.00185	0	-0.00185	0	-0.0037	0	-0.0037	0
1481	0	0	0	0	0	0	0	0	0	0	0	0	-0.00185	0	-0.00185	0	-0.00185	0
1482	1	0	0	-0.00313	-0.02778	0.002083	-0.05741	0.010417	-0.08148	0.020833	-0.09444	0.0375	-0.10741	0.054167	-0.10556	0.073958	-0.09444	0.088542
1483	1	0	0	-0.01354	-0.02037	-0.02292	-0.04815	-0.02292	-0.08519	-0.01875	-0.11481	-0.00938	-0.13704	0.002083	-0.15741	0.017708	-0.17037	0.034375
1484	1	0	0	-0.00938	-0.02778	-0.00938	-0.06481	-0.00521	-0.09444	0.004167	-0.11667	0.015625	-0.13704	0.03125	-0.15	0.047917	-0.1537	0.066667
1485	1	0	0	0	-0.03704	0.004167	-0.06667	0.013542	-0.08889	0.025	-0.10926	0.040625	-0.12222	0.057292	-0.12593	0.076042	-0.12037	0.09375

The key point coordinates are the ones that have undergone the following preprocessing up to ④.

① 時系列座標（Time series coordinates）

T-15	T-14	T-13	……	T-2	T-1	T
[550, 165]	[526, 176]	[509, 188]	……	[644, 219]	[644, 196]	[642, 178]

② [T-15]からの相対座標に変換（Convert to relative coordinates from [T-15]）

T-15	T-14	T-13	……	T-2	T-1	T
[0, 0]	[-24, 11]	[-17, 12]	……	[5, -16]	[0, -23]	[-2, -18]

③ 画面幅と高さに合わせて正規化（Normalized to fit screen width and height）
一画面幅960、画面高540の場合、T-14は[-24/960, 11/540]（Width of 960 and height of 540, T-14 is [-24/960, 11/540]）

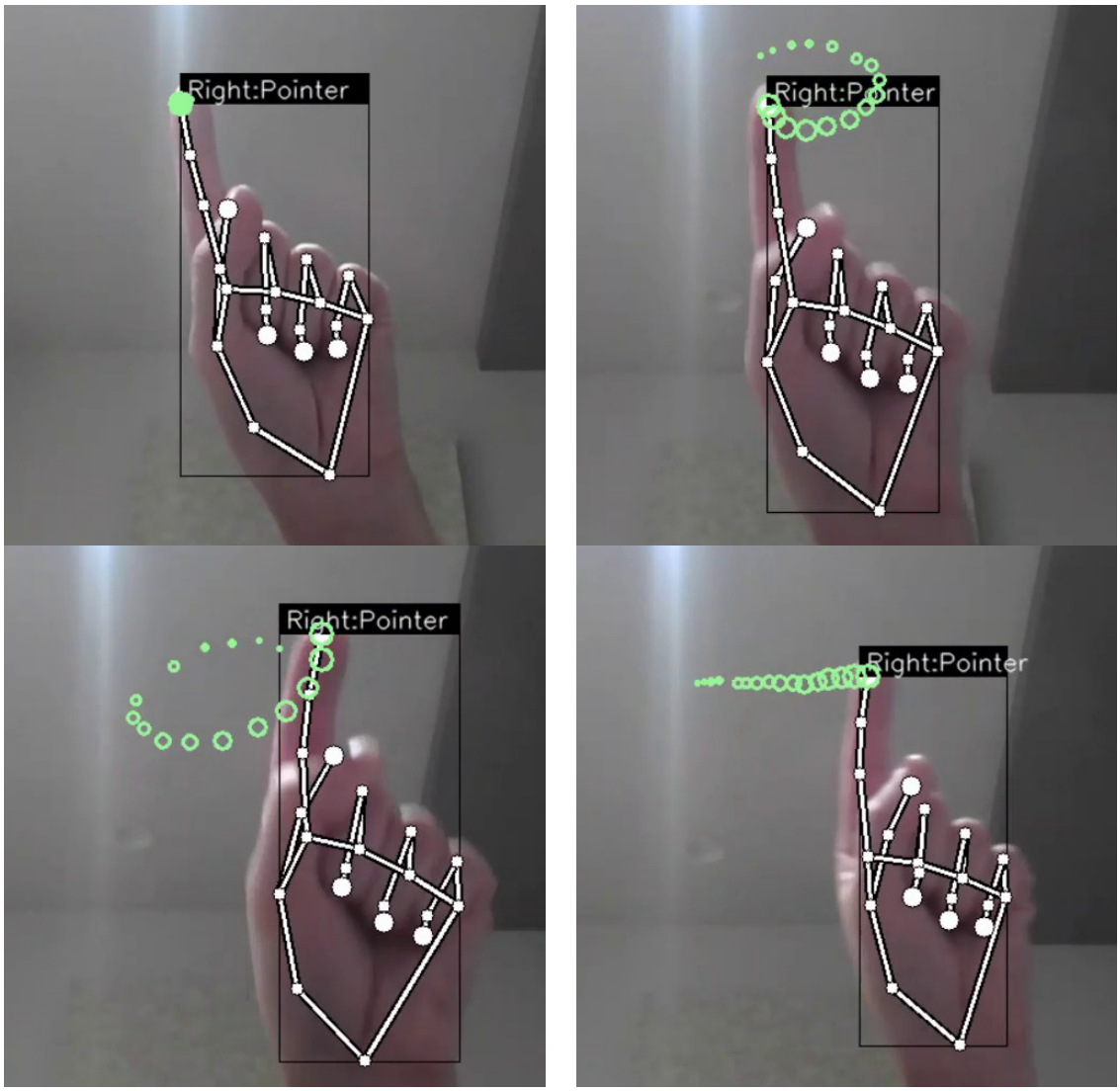
T-15	T-14	T-13	……	T-2	T-1	T
[0.0, 0.0]	[-0.025, 0.0204]	[-0.0427, 0.0426]	……	[0.0979, 0.1]	[0.0979, 0.0574]	[0.0958, 0.024]

④ 1次元配列に変換（Flatten to a one-dimensional array）

T-15		T-14		T-13		……	T-2		T-1		T	
0.0000	0.0000	-0.0250	0.0204	-0.0427	0.0426	……	0.0979	0.1000	0.0979	0.0574	0.0958	0.0241

In the initial state, 4 types of learning data are included: stationary (class ID: 0),

clockwise (class ID: 1), counterclockwise (class ID: 2), and moving (class ID: 4).
If necessary, add 5 or later, or delete the existing data of csv to prepare the training data.

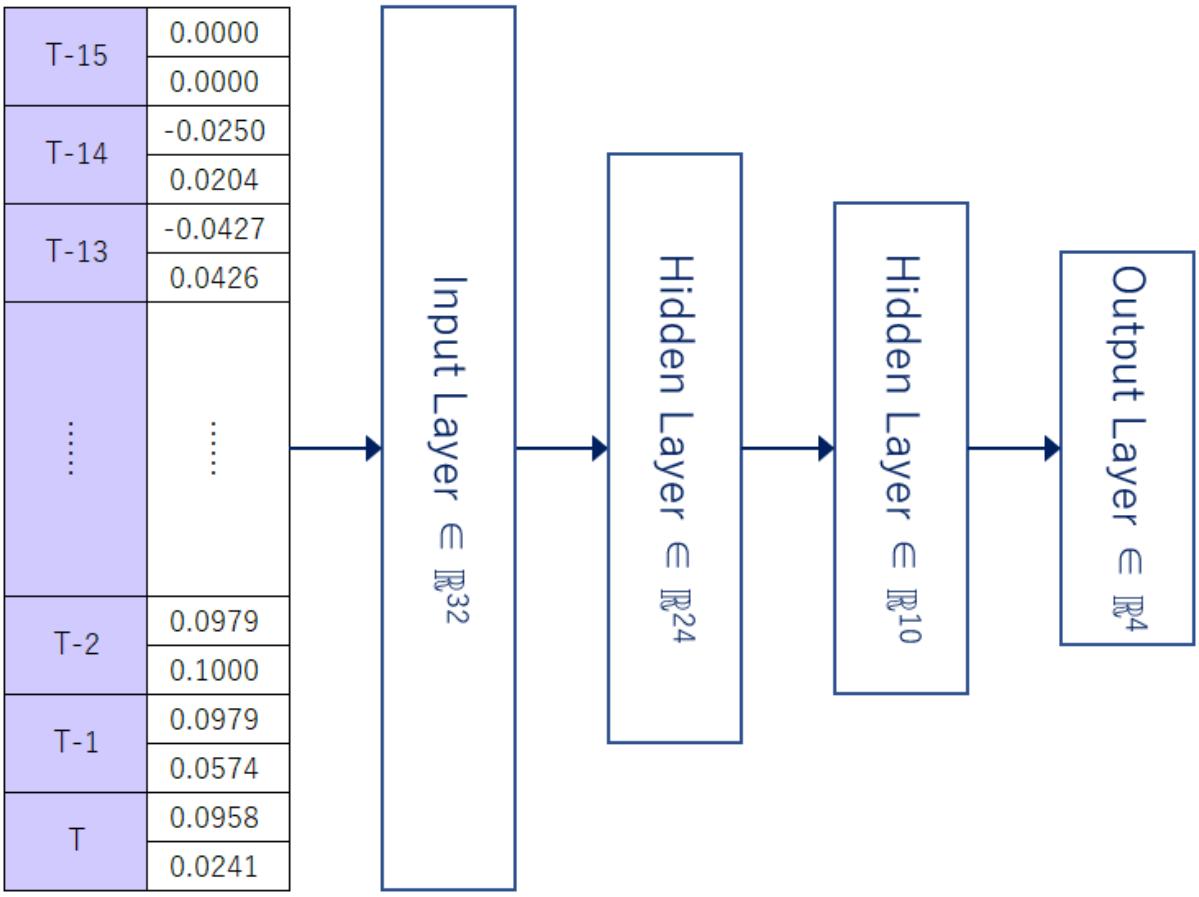


2.Model training

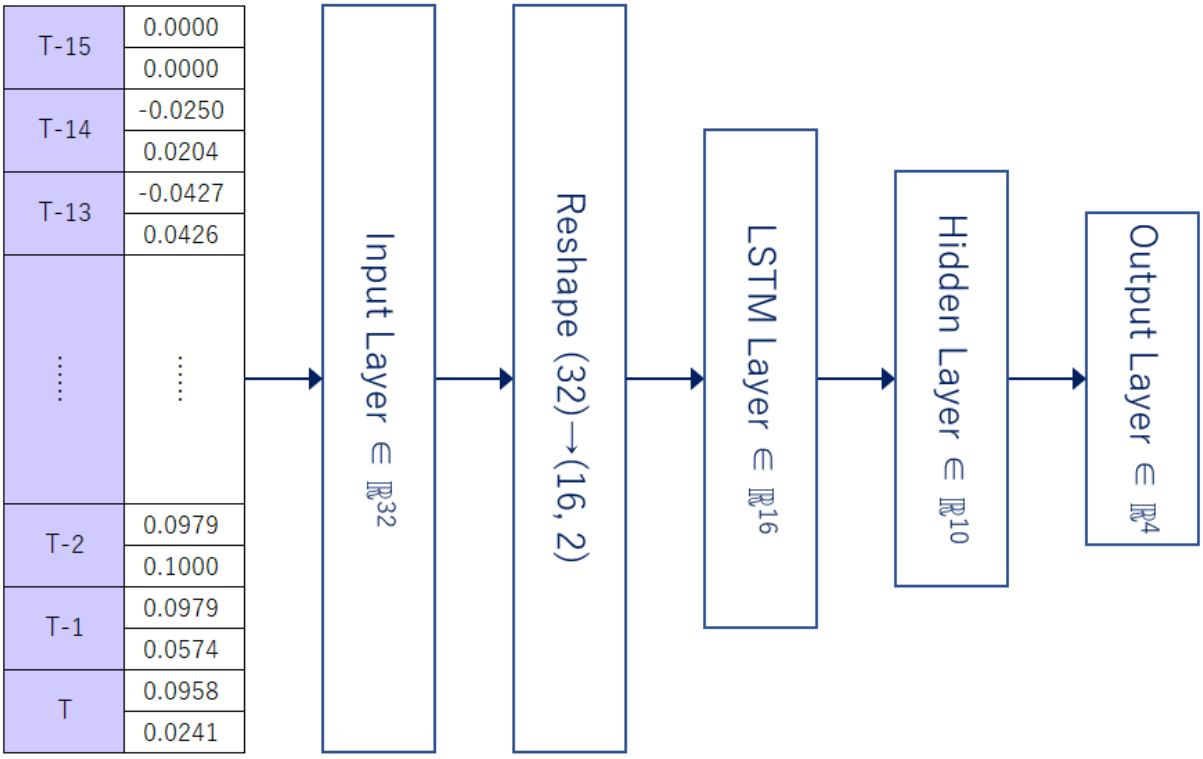
Open "point_history_classification.ipynb" in Jupyter Notebook and execute from top to bottom.
To change the number of training data classes, change the value of "NUM_CLASSES = 4" and
modify the label of "model/point_history_classifier/point_history_classifier_label.csv" as appropriate.

X.Model structure

The image of the model prepared in "point_history_classification.ipynb" is as follows.



The model using "LSTM" is as follows.
Please change "use_lstm = False" to "True" when using (tf-nightly required (as of 2020/12/16))



1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements and specifications for the development of a Hand Gesture Recognition System using the MediaPipe framework. This system aims to accurately estimate hand poses, recognize hand signs, and detect finger gestures in real-time.

1.2 Scope

The Hand Gesture Recognition System will utilize computer vision and machine learning techniques to analyze video input from a webcam or similar device. It will provide the following functionalities:

- Hand pose estimation: Estimating the 3D coordinates of hand landmarks in the video stream.

- Hand sign recognition: Identifying and classifying specific hand signs or gestures made by users.

- Finger gesture recognition: Recognizing dynamic finger movements and gestures performed by users.

1.3 Definitions, Acronyms, and Abbreviations

- MediaPipe: An open-source framework developed by Google for building multimodal machine learning pipelines.

- MLP: Multilayer Perceptron, a type of feedforward artificial neural network.

- TFLite: TensorFlow Lite, a lightweight machine learning framework for mobile and embedded devices.

1.4 References

- MediaPipe Documentation: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

- TensorFlow Lite Documentation: https://www.tensorflow.org/

2. Overall Description

2.1 Product Perspective

The Hand Gesture Recognition System will serve as a standalone application, capable of running on various platforms including desktop computers and mobile devices. It will interact with external hardware (webcams) for capturing video input and may integrate with other software systems through APIs or libraries.

2.2 Product Functions

- Hand Pose Estimation: Analyzing the video stream to estimate the spatial coordinates of key landmarks on the user's hand.

- Hand Sign Recognition: Classifying static hand gestures or signs based on the detected hand pose.

- Finger Gesture Recognition: Identifying dynamic finger movements and gestures performed by the user.

2.3 User Classes and Characteristics

The primary users of the system include developers, researchers, and end-users interested in gesture-based interaction systems. Users should have basic knowledge of computer vision and machine learning concepts to effectively utilize the system.

2.4 Operating Environment

The system is designed to operate in a standard computing environment with the following requirements:

- Operating System: Windows, macOS, Linux, Android, iOS

- Hardware: Webcam or built-in camera for video input

- Software Dependencies: Python 3.x, MediaPipe, TensorFlow, OpenCV

2.5 Design and Implementation Constraints

- Real-time Performance: The system should process video input in real-time with minimal latency to provide a seamless user experience.

- Computational Resources: The system's resource requirements should be reasonable to ensure compatibility with a wide range of devices.

2.6 Assumptions and Dependencies

The development of the Hand Gesture Recognition System assumes access to appropriate hardware (webcams) and software libraries (MediaPipe, TensorFlow) for implementing the required functionalities.

3. System Features

3.1 Feature 1: Hand Pose Estimation

- Description: Utilizes MediaPipe's hand tracking model to estimate the 3D coordinates of key landmarks on the user's hand.

- Inputs: Video stream from the webcam.

- Outputs: 3D coordinates of hand landmarks (e.g., fingertips, palm center).

3.2 Feature 2: Hand Sign Recognition

- Description: Classifies static hand gestures or signs based on the detected hand pose.

- Inputs: Hand landmarks detected by the system.

- Outputs: Predicted hand sign labels corresponding to predefined gesture classes.

3.3 Feature 3: Finger Gesture Recognition

- Description: Identifies dynamic finger movements and gestures performed by the user.

- Inputs: Hand landmarks and their temporal sequence.

- Outputs: Predicted finger gesture labels indicating the recognized hand movement patterns.

4. External Interface Requirements

4.1 User Interfaces

- Command-Line Interface (CLI): Provides options for running the system from the command line with configurable parameters.

- Graphical User Interface (GUI): Offers a user-friendly interface for interacting with the system, displaying video input and recognized gestures.

4.2 Hardware Interfaces

- Webcam Interface: Captures video input from the connected webcam device.

- Display Interface: Renders the video stream and recognized gestures on the user's display device.

4.3 Software Interfaces

- MediaPipe Integration: Utilizes the MediaPipe library for hand tracking and pose estimation.

- TensorFlow Integration: Incorporates TensorFlow models for hand sign and finger gesture recognition.

5. Non-functional Requirements

5.1 Performance Requirements

- Frame Rate: The system should achieve a minimum frame rate of 30 frames per second (FPS) for real-time processing.

- Latency: The processing latency from video input to gesture recognition should be minimal (<100 milliseconds).

5.2 Security Requirements

- Data Privacy: The system should not store or transmit sensitive user data captured by the webcam without explicit user consent.

- Access Control: Access to the system's functionality and data should be restricted based on user roles and permissions.

5.3 Reliability Requirements

- System Availability: The system should be available for use during normal operating hours without frequent downtime.

- Error Handling: The system should gracefully handle errors and exceptions to prevent crashes or unexpected behavior.

5.4 Maintainability Requirements

- Code Modularity: The system's codebase should be modular and well-organized to facilitate future updates and maintenance.

- Documentation: Comprehensive documentation should be provided for developers and users to understand the system's architecture and usage.

5.5 Portability Requirements

- Cross-Platform Compatibility: The system should be compatible with multiple operating systems (Windows, macOS, Linux) and hardware configurations.

- Mobile Support: The system should be optimized for deployment on mobile platforms (Android, iOS) with minimal resource overhead.

6. References

https://mediapipe.dev/

https://github.com/kinivi/hand-gesture-recognition-mediapipe/blob/main/README.md

https://www.youtube.com/watch?v=a99p_fAr6e4