AUTODESK Instructables

How to Train Custom Hand Gestures Using Mediapipe

By thomas9363 in CircuitsComputers
(cc) BY-NC-SA

Introduction: How to Train Custom Hand Gestures Using Mediapipe

One of the subsets of artificial intelligence, machine learning - particularly deep learning - has gained significant attention and success in recent years. By using deep learning algorithms, computers can interpret and understand visual information from the world. As a result, many applications have been developed to detect and recognize objects such as dogs, cars, flowers, and other items. There are various object detection methods and frameworks available. In this article, I will focus on a straightforward way to train computers to detect your own hand gestures and use them to control a robotic device.

Supplies

In this project, I am using the following hardware:

- A 16GB laptop with Windows 11 OS
- A 4GB Raspberry Pi 4 microprocessor with Debian 12 Bookworm OS (optional)
- A Coral USB edge TPU accelerator (optional)

The software I have installed on my Windows system includes Anaconda and PyCharm. I have created a virtual environment using conda and installed the project-specific software, including:

- TensorFlow 2.14
- OpenCV 4.8.1
- MediaPipe 0.10.8
- NumPy 1.26.4
- Jupyter Notebook

On the Raspberry Pi, I have also created a virtual environment and installed the required modules, including:

- TensorFlow 2.16.1
- OpenCV 4.9.0
- MediaPipe 0.10.9
- NumPy 1.26.4

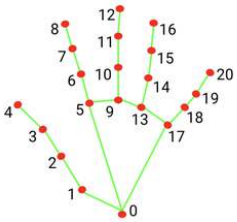
Other versions of these modules may also work. However, some modules may evolve faster than others. If you encounter any error messages, you can refer to the specified versions to ensure compatibility with my codes. You can download them from my GitHub repository.

Step 1: Common Approaches

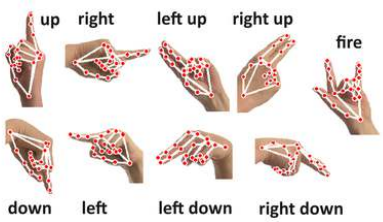
There are several ways to train your own hand gesture detection system. Two main approaches are: (1) using a large amount of photo data of hand gestures and (2) using the MediaPipe Hand landmark framework. These approaches differ significantly in terms of methodology, complexity, and application.

Using raw photo data requires substantial preprocessing, such as resizing, normalization, and sometimes augmentation. Training convolutional neural networks on large datasets takes significant time, computational resources, and powerful hardware for real-time performance. In contrast, the MediaPipe Hand landmark framework only needs key points, making the training and inference time much shorter. This allows for real-time performance even on less powerful edge devices such as Raspberry Pi. I will use the second approach for my training and detection.

Step 2: Hand Gestures



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



MediaPipe Hand is a powerful framework that can detect 21 landmarks on a hand, as shown. With this, you can write a simple program to identify fingers or count the number of fingers you pose. By assigning different functions to different fingers, you can easily control a robot. For example, you can use the thumb, index finger, middle finger, and ring finger to move a robot up, down, left, and right, respectively. However, this approach is not intuitive. As the number of gestures increases, you may not remember the purpose of some of the gestures.

A better way is to define each gesture in an intuitive manner. For instance, when you point your index finger up, down, left, or right, the robot moves up, down, left, or right accordingly. In this project, I have designed nine different gestures: "Up," "Down," "Left," "Right," "Left Up," "Left Down," "Right Down," "Right Up," and "Fire."

That said, writing code for these gestures can require significant effort and many lines of code. An easier approach is to use a neural network and train the computer to recognize the gestures by itself.

Step 3: Workflow

The following are the breakdown of each step using MediaPipe to extract landmark points and using them to train a gesture detection model with TensorFlow and Keras:

Data Collection:

- Using MediaPipe to detect and extract landmark points from hand gestures in video frames.
- Normalizing the landmark points to make the model training more effective.
- Storing the extracted landmarks along with their corresponding gesture labels in a CSV file for training.

Model Training:

- Splitting the collected data into training, validation, and test sets.
- Creating a neural network model to classify gestures based on the extracted landmarks.
- Using the Keras Sequential API to define the layers of the neural network, including input, hidden, and output layers.
- Training the neural network on the preprocessed landmark data using TensorFlow's training functionalities.
- Validating the model on the validation set to tune hyperparameters and prevent overfitting.

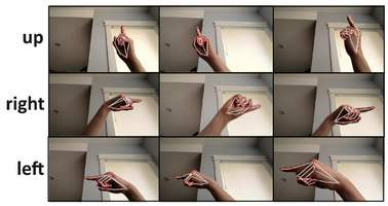
Model Deployment:

- Using the trained model to predict gestures in real-time based on new landmark data.
- Porting to a Raspberry Pi to control a robotic device (optional).

The following files are created during the execution of the above steps:

- hand_create_csv.py - Collecting hand gesture data for training
- hand_gesture_data.csv - Collected training data in CSV format
- hand_train.ipynb - Jupyter notebook file for model training
- hand_gesture_model.h5 - Trained model in H5 format
- hand_gesture_model.tflite - Trained model in TFLite format
- hand_gesture_model_quantized.tflite - Trained model converted to a quantized model
- hand_detect.py - Runs TFLite model and displays the result as text in the image window
- hand_detect_h5.py - Runs H5 model and displays the result as text in the image window
- hand_detect_quantized.py - Runs quantized TFLite model and displays the result as text in the image window
- hand_detect_move_ball.py - Uses gestures to move a ball and change its color
- hand_detect_robot.py - Uses gestures to control servo movement on a Raspberry Pi

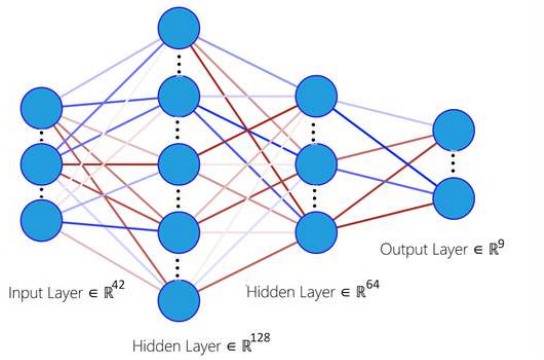
Step 4: Data Collection



label	x0	y0	x1	y1	x2	y2	·····	x19	y19	x20	y20
0	0	0	-0.03033	-0.144234	-0.087753	-0.266411		-0.169783	0.030573	-0.143914	0.036531
0	0	0	-0.035854	-0.158249	-0.064451	-0.319114		-0.114044	-0.015522	-0.076278	-0.004892
⋮											
1	0	0	-0.044397	0.050285	-0.104694	0.101315		-0.096277	-0.099296	-0.090418	-0.106566
1	0	0	-0.044356	0.051899	-0.104913	0.102884		-0.095228	-0.098684	-0.089563	-0.107195
⋮											
6	0	0	-0.041442	-0.008747	-0.092899	-0.066064		-0.014069	-0.185213	-0.006645	-0.159959
6	0	0	-0.04307	-0.009301	-0.095427	-0.063998		-0.017461	-0.184905	-0.008784	-0.158443
⋮											

hand_create_csv.py: This script uses MediaPipe Hand to detect hand landmarks in video frames. Up to a total of 26 gestures can be extracted. You move your hand in front of the camera in different positions and angles as shown above. By pressing keyboard keys a to z, defined as class numbers 0-25, the x and y coordinates of the landmarks of your hand at that frame are extracted and normalized to point 0 (the wrist point). The class number and the flattened coordinates are stored as a row in a CSV file. The count of the dataset in each class is displayed when you show your hand and press a letter. The data are sorted during execution and stored in CSV file format. I have collected 60 sets of data for each gesture in different positions and angles, totaling 540 datasets. The resulting file is called "hand_gesture_data.csv" and is structured as shown above.

Step 5: Model Training



hand_train.ipynb: This notebook file starts the neural network training. The data from the CSV file is fed into a neural network model built with TensorFlow and Keras. Model layers are defined using the Sequential API. I am using a simple ReLU activation function. Here is a snippet of the code:

```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(len(np.unique(y)), activation='softmax')
])
```

I am using Jupyter Notebook in PyCharm on my laptop to run the training. The training is fast and can be completed within a minute. After training, the model is saved. Several models in different formats are saved, including *.h5 and *.tflite files. The *.tflite is the trained model, which can be used on Windows OS or Raspberry Pi. Additionally, I convert the model to a TFLite model with quantization for later testing. At the end of this step, the following files are created: hand_gesture_model.h5, hand_gesture_model.tflite, and hand_gesture_model_quantized.tflite.

Step 6: Model Deployment and Inference

hand_detect.py: This script takes hand_gesture_model.tflite as input, performs inference, and displays the detected hand gesture as text on the screen. You press "Esc" to exit the script. Similarly, the other two scripts, hand_detect_h5.py and hand_detect_quantized.py, are used to run hand_gesture_model.h5 and hand_gesture_model_quantized.tflite, respectively. These scripts can be run on both Windows and Raspberry Pi.

hand_detect_move_ball.py: This file contains an interactive graphic routine that allows you to use your hand gestures to roll a ball on the screen in different directions. It also changes the color of the ball when you show a "Fire" gesture. This script can also be run on both Windows and Raspberry Pi.

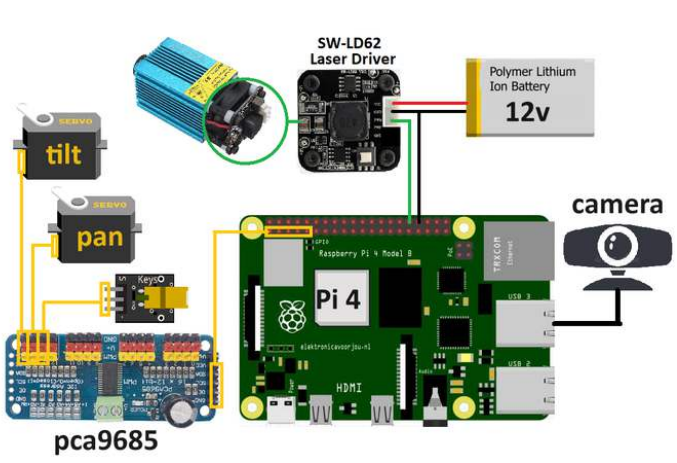
I have embedded the name of my gestures in the script:

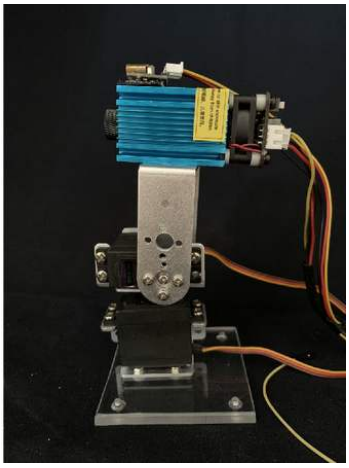
```
gesture_names = ["Up", "Down", "Left", "Right", "Left Up", "Left Down", "Right Down",
"Right Up", "Fire"]
```

If you train your own gestures with different name, you need to change them accordingly. Another thing to note is that you need to change cap = cv2.VideoCapture() to the number of camera you are using.

You can go further if you want to use your newly trained gesture to control a robotic device on Raspberry Pi.

Step 7: Robotic Device





The device, as shown above, contains a pan-tilt servo setup with two lasers mounted on top. The servos can move in the x and y directions according to the hand gestures. The small 5mW laser is always on so that the movement of the servos is visible. The larger 500mW laser turns on when the camera detects the “Fire” gesture. The two servos and the small laser are plugged into a servo driver (PCA9685) via an I2C connection, while the larger laser is controlled using a GPIO pin and powered by a 12V LiPo battery. The detection is done using a Logitech C920 USB camera.

hand_detect_robot.py is the script I used to control the device. To move the servos, you need to install the adafruit_servokit module and supply the servo driver with 5V power.

Here are the key points summarized:

- **Device Components:**
- Pan-tilt servo setup with two lasers.
- Small 5mW laser (always on).
- Larger 500mW laser (turns on with “Fire” gesture).
- Servos and small laser connected to PCA9685 via I2C.
- Large laser controlled by a GPIO pin and powered by a 12V LiPo battery.
- Detection performed using a Logitech C920 USB camera.
- **Script: hand_detect_robot.py**
- Controls the pan-tilt servos and lasers based on detected gestures.
- Requires installation of the adafruit_servokit module.
- Servo driver needs to be supplied with 5V power.

The script can be easily expanded to control a different robotic device on Raspberry Pi using your own gestures. This flexibility allows you to adapt the system for various applications, providing a customizable solution for gesture-based control of robotic devices.

Step 8: Conclusions

The video has two parts. The first part demonstrates using hand gestures to control a ball on a graphic interface. A red ball moves according to the direction of the hand gesture, and the ball changes its color when the camera detects the "Fire" gesture. The second part shows using hand gestures to move the pan-tilt setup, where the larger laser turns on upon detecting the "Fire" gesture.

The speed on the Windows system is around 30fps, but it drops to 6fps on the Pi4. I have a Coral Edge TPU accelerator that is supposed to bring accelerated ML inferencing. The Edge TPU is specifically optimized for 8-bit integer operations; therefore, the model weights and activations need to be converted from 32-bit floating point to 8-bit integers. I have done this during the last step in hand_train.ipynb to generate a quantized TFLite file (hand_gesture_model_quantized.tflite), and have used a web-based Google Colab compiler (compile_for_edgetpu.ipynb) to compile it to a TPU-capable tflite file, namely, hand_gesture_model_quantized_edgetpu.tflite.

To run the Edge TPU model on a Raspberry Pi, I have followed the instructions to set up the Edge TPU runtime, the necessary software, and modify my script to use the Edge TPU. The script is hand_detect_edgetpu.py

Unfortunately, a simple inference test shows that the Edge TPU doesn’t improve the speed. The Google site suggests that a library called PyCoral may increase the speed. However, after following its instructions and downgrading Python to 3.9, it still gives error messages. It seems to be a common problem in the community, and there seems to be no point in wasting more time searching for a solution.

In summary, MediaPipe's training for custom hand gestures has been straightforward and effective for real-time applications. If you encounter issues with gesture detection accuracy, adding more data and retraining the model can often help improve performance.