



MediaPipe in Python


MediaPipe in Python

Ready-to-use Python Solutions MediaPipe on Google Colab MediaPipe Python Framework Building MediaPipe Python Package
MediaPipe offers ready-to-use yet customizable Python solutions as a prebuilt Python package. MediaPipe Python package is available on PyPI for Linux, macOS and Windows.

https://google.github.io/mediapipe/getting_started/python.html

Projects - Computer Vision Zone

Computer Vision Projects

 <https://www.computervision.zone/projects/>



▼ code

```
"""
Hand Tracking Module
By: Computer Vision Zone
Website: https://www.computervision.zone/
"""

import cv2
import mediapipe as mp
import math

class HandDetector:
    """
    Finds Hands using the mediapipe library. Exports the landmarks
    in pixel format. Adds extra functionalities like finding how
    many fingers are up or the distance between two fingers. Also
    provides bounding box info of the hand found.
    """
```

```

def __init__(self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5):
    """
    :param mode: In static mode, detection is done on each image: slower
    :param maxHands: Maximum number of hands to detect
    :param detectionCon: Minimum Detection Confidence Threshold
    :param minTrackCon: Minimum Tracking Confidence Threshold
    """
    self.mode = mode
    self.maxHands = maxHands
    self.detectionCon = detectionCon
    self.minTrackCon = minTrackCon

    self.mpHands = mp.solutions.hands
    self.hands = self.mpHands.Hands(self.mode, self.maxHands,
                                     self.detectionCon, self.minTrackCon)
    self.mpDraw = mp.solutions.drawing_utils
    self.tipIds = [4, 8, 12, 16, 20]
    self.fingers = []
    self.lmList = []

def findHands(self, img, draw=True, flipType=True):
    """
    Finds hands in a BGR image.
    :param img: Image to find the hands in.
    :param draw: Flag to draw the output on the image.
    :return: Image with or without drawings
    """
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)
    allHands = []
    h, w, c = img.shape
    if self.results.multi_hand_landmarks:
        for handType, handLms in zip(self.results.multi_handedness, self.results.multi_hand_landmarks):
            myHand={}
            ## lmList
            mylmList = []
            xList = []
            yList = []
            for id, lm in enumerate(handLms.landmark):
                px, py = int(lm.x * w), int(lm.y * h)
                mylmList.append([px, py])
                xList.append(px)
                yList.append(py)

            ## bbox
            xmin, xmax = min(xList), max(xList)
            ymin, ymax = min(yList), max(yList)
            boxW, boxH = xmax - xmin, ymax - ymin
            bbox = xmin, ymin, boxW, boxH
            cx, cy = bbox[0] + (bbox[2] // 2), \
                    bbox[1] + (bbox[3] // 2)

            myHand["lmList"] = mylmList
            myHand["bbox"] = bbox
            myHand["center"] = (cx, cy)

            if flipType:
                if handType.classification[0].label == "Right":
                    myHand["type"] = "Left"
                else:
                    myHand["type"] = "Right"
            else: myHand["type"] = handType.classification[0].label
            allHands.append(myHand)

            ## draw
            if draw:
                self.mpDraw.draw_landmarks(img, handLms,
                                           self.mpHands.HAND_CONNECTIONS)
                cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                              (bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20),
                              (255, 0, 255), 2)

```

```

        cv2.putText(img, myHand["type"], (bbox[0] - 30, bbox[1] - 30), cv2.FONT_HERSHEY_PLAIN,
                    2, (255, 0, 255), 2)

    if draw:
        return allHands, img
    else:
        return allHands

def fingersUp(self, myHand):
    """
    Finds how many fingers are open and returns in a list.
    Considers left and right hands separately
    :return: List of which fingers are up
    """
    myHandType = myHand["type"]
    myLmList = myHand["lmList"]
    if self.results.multi_hand_landmarks:
        fingers = []
        # Thumb
        if myHandType == "Right":
            if myLmList[self.tipIds[0]][0] > myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)
        else:
            if myLmList[self.tipIds[0]][0] < myLmList[self.tipIds[0] - 1][0]:
                fingers.append(1)
            else:
                fingers.append(0)

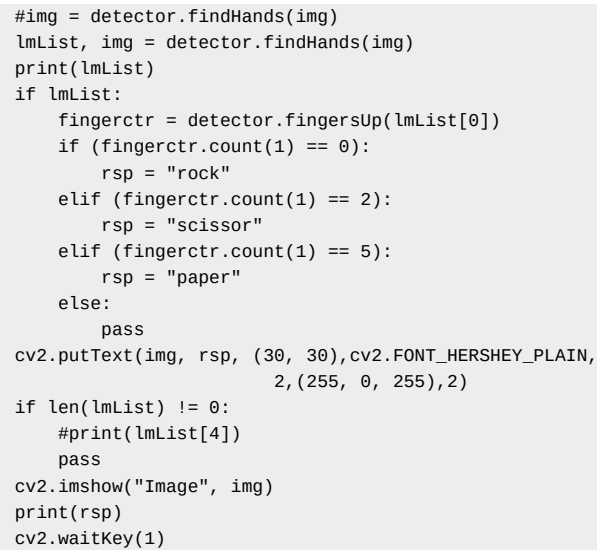
        # 4 Fingers
        for id in range(1, 5):
            if myLmList[self.tipIds[id]][1] < myLmList[self.tipIds[id] - 2][1]:
                fingers.append(1)
            else:
                fingers.append(0)
    return fingers

def findDistance(self, p1, p2, img=None):
    """
    Find the distance between two landmarks based on their
    index numbers.
    :param p1: Point1
    :param p2: Point2
    :param img: Image to draw on.
    :param draw: Flag to draw the output on the image.
    :return: Distance between the points
             Image with output drawn
             Line information
    """
    x1, y1 = p1
    x2, y2 = p2
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
    length = math.hypot(x2 - x1, y2 - y1)
    info = (x1, y1, x2, y2, cx, cy)
    if img is not None:
        cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
        return length, info, img
    else:
        return length, info

if __name__ == "__main__":
    rsp = None
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)
    detector = HandDetector()
    while True:
        success, img = cap.read()

```

```

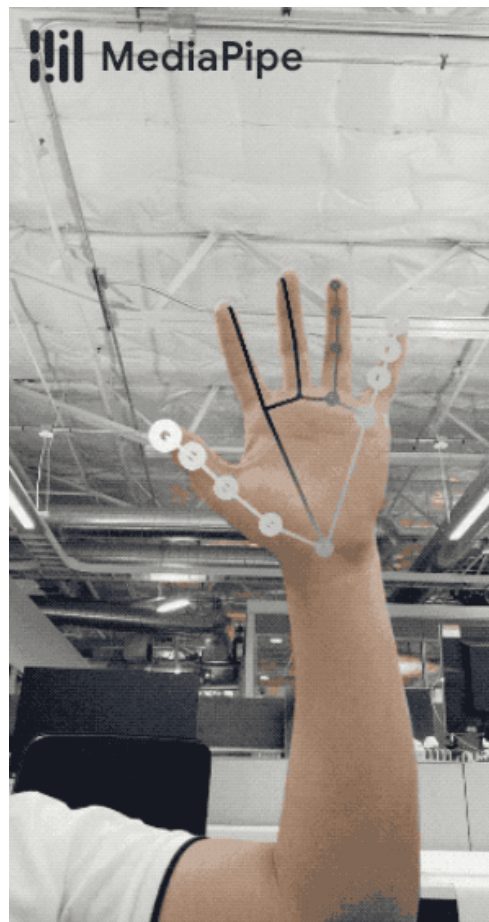

#img = detector.findHands(img)
lmList, img = detector.findHands(img)
print(lmList)
if lmList:
    fingerctr = detector.fingersUp(lmList[0])
    if (fingerctr.count(1) == 0):
        rsp = "rock"
    elif (fingerctr.count(1) == 2):
        rsp = "scissor"
    elif (fingerctr.count(1) == 5):
        rsp = "paper"
    else:
        pass
cv2.putText(img, rsp, (30, 30), cv2.FONT_HERSHEY_PLAIN,
            2, (255, 0, 255), 2)
if len(lmList) != 0:
    #print(lmList[4])
    pass
cv2.imshow("Image", img)
print(rsp)
cv2.waitKey(1)

```

MediaPipe Hands

Overview

MediaPipe Hands는 충실도가 높은 손 및 손가락 추적 솔루션이다. 머신러닝을 사용하여 단일 프레임에서 손의 21개 3D 랜드마크를 추론한다.



ML Pipeline

MediaPipe Hands는 함께 작동하는 여러 모델로 구성된 머신러닝 파이프라인을 활용한다. 전체 이미지에서 작동하고 hand bounding box를 반환하는 palm detection model이다. hand landmark model은 palm detection model에 의해 잘린 이미지 영역에서 고성능 3D hand keypoints을 반환한다.

hand landmark model에 input으로 정확하게 잘라낸 손 이미지를 주면 네트워크가 대부분의 용량을 좌표 예측 정확도에 할애한다. landmark model이 손을 식별할 수 없는 경우 palm detection이 재 작동하여 손의 위치를 재 지정한다.

파이프라인은 핸드 랜드마크 모듈의 hand landmark tracking subgraph를 사용하는 MediaPipe 그래프로 구현되며 hand renderer subgraph를 사용하여 렌더링된다. hand landmark tracking subgraph는 내부적으로 동일한 모듈의 hand landmark tracking subgraph와 palm detection subgraph의 palm detection module을 사용한다.

Models

Palm Detection Model

이미지 프레임에 비해 큰 스케일 범위(~20x)로 다양한 손 크기에서 작동해야 하며 가려진 손과 자체 폐쇄된 손을 감지할 수 있어야 한다. 얼굴은 예를 들어 눈과 입 영역에서 고대비 패턴을 갖는 반면, 손에는 이러한 특징이 없기 때문에 시각적 특징만으로는 이를 안정적으로 감지하기가 상대적으로 어렵다. 대신 팔, 신체 또는 사람 특징과 같은 추가 컨텍스트를 제공하면 정확한 손 위치 파악에 도움이 된다.

mediapipe의 **Palm Detection Model**은 다른 전략을 사용하여 위의 문제를 해결했다.

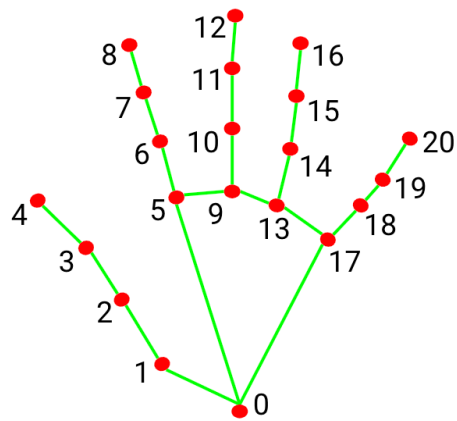
1. 손바닥과 주먹과 같은 단단한 물체의 경계 상자를 추정하는 것이 관절 손가락으로 손을 감지하는 것보다 훨씬 간단하기 때문에 손 감지기 대신 손바닥 감지기를 훈련.
2. encoder-decoder feature extractor는 작은 개체에 대해서도 더 큰 장면 컨텍스트 인식을 위해 사용된다.

위의 기술을 통해 손바닥 감지에서 평균 95.7%의 정확도를 달성했다.

Hand Landmark Model

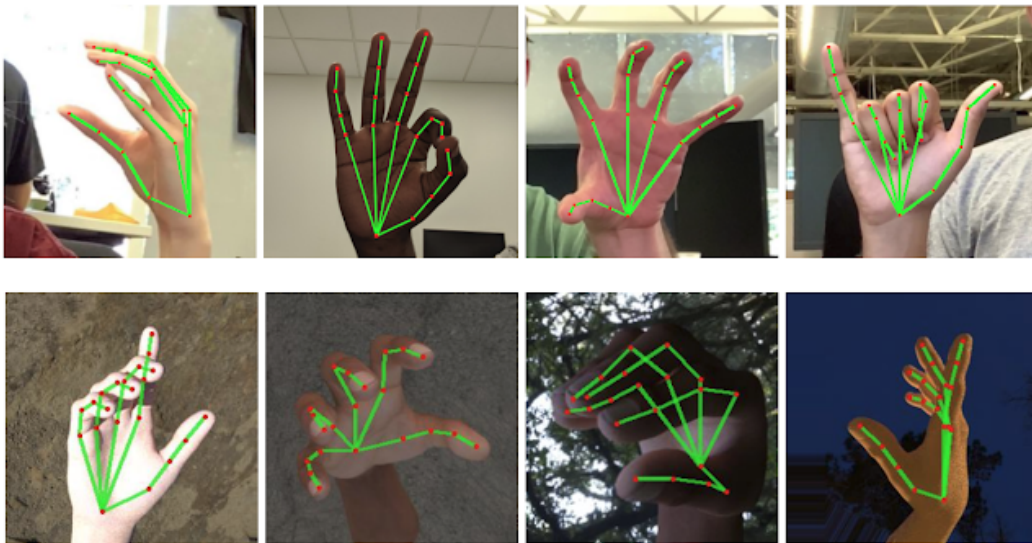
전체 이미지에 대한 palm detection 후 subsequent hand landmark model은 회귀, 즉 직접 좌표 예측을 통해 감지된 손 영역 내부의 21개의 3D 손 관절 좌표의 정확한 keypoint 위치를 찾아낸다. 이 model은 일관된 내부 손 포즈 표현을 학습하고 부분적으로 보이는 손과 자가 교합에도 견고하다.

실측 데이터를 얻기 위해 아래와 같이 21개의 3D 좌표로 ~30K 실제 이미지에 수동으로 주석을 달았다(해당 좌표별로 존재하는 경우 이미지 깊이 맵에서 Z 값을 가져옴). 가능한 손 포즈를 더 잘 커버하고 손 기하학의 특성에 대한 추가 감독을 제공하기 위해 다양한 배경 위에 고품질 합성 손 모델을 렌더링하고 해당 3D 좌표에 매핑한다.



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



Model Details

- A palm detection model (3.7MB size)
- A hand landmark model (3.7MB size)
- To detect palm and predict hand landmarks within an image on a smartphone.
- Palm detector returns bounding boxes for each palm
- Hand landmark model predicts keypoints for each hand from the cropped image.

Model Types

- Convolutional Neural Network

Model Architecture

- Palm detector: Adapted SSD with a custom encoder
- Hand landmark model: regression model

Input

- Palm detection modle
 - A frame of video or an image, represented as a 128x128x3 tensor
 - channels order: RGB with values in [-1.0, -1.0].
- Hand landmark model
 - A frame of video or an image, represented as a 224 x 224 x 3 tensor
 - Channels order: RGB with values in [0.0, 1.0].

Output

- Palm detection modle
 - Predicted offset of predefined anchors represented as a 1 x 896 x 18 tensor
 - Predicted detection confidence score of each anchor represented as a 1 x 896 tensor.
- Hand landmark model
 - A float scalar represents the presence of a hand in the given input image.
 - 213-dimensional landmarks represented as a 1 x 63 tensor and are normalized by image size.
This output should only be considered valid when the presence score is higher than a threshold.
 - A float scalar represents the handedness of the predicted hand. This output should only be considered valid when the presence score is higher than a threshold.