In [1]:
```
#opencv - > BGR
#matplot-> RGB
#mediapipe->RGB
```

In [2]:
```
# stop the mission
# freeze your self
# call
# hold your fire
# down
# fire
# run
# silent
# message ack
```

In [3]:
```
'''
palm
fist
gun
call
thumbsup
'''
```

Out[3]:
```
'\npalm\nfist\ngun\ncall\nthumbsup\n'
```

In [4]:
```
import os
import time
import numpy as np
import cv2 as cv
import mediapipe as mp
```

In [14]:
```
delay = 50
input_types = ['palm', 'fist', 'thumbsup', 'gun', 'call']
min_input_count = 2000 + delay
```

In [6]:
```
def make_dirs():#use this in when the training part or testing
    # applying the checking of the folder for input type folders.
    global input_types

    # applying list dir on current dir.
    directory = './Inputs/'
    if not os.path.isdir(directory):
        os.mkdir(directory,mode=511)
    os.chdir(directory)
    # making the directory if doesn't exist
    for dirs in input_types:
        if not os.path.isdir(dirs):
            os.mkdir(dirs,mode=511)
    os.chdir('..')
```

In [7]:
```
#make_dirs()
```

In [8]:
```
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
```

In [9]:
```
holistic = mp_holistic.Holistic()
```

In [10]:
```
mp_holistic.HAND_CONNECTIONS
```

Out[10]:
```
frozenset({(<HandLandmark.WRIST: 0>, <HandLandmark.THUMB_CMC: 1>),
           (<HandLandmark.WRIST: 0>, <HandLandmark.INDEX_FINGER_MCP: 5>),
           (<HandLandmark.WRIST: 0>, <HandLandmark.PINKY_MCP: 17>),
           (<HandLandmark.THUMB_CMC: 1>, <HandLandmark.THUMB_MCP: 2>),
           (<HandLandmark.THUMB_MCP: 2>, <HandLandmark.THUMB_IP: 3>),
           (<HandLandmark.THUMB_IP: 3>, <HandLandmark.THUMB_TIP: 4>),
           (<HandLandmark.INDEX_FINGER_MCP: 5>,
            <HandLandmark.INDEX_FINGER_PIP: 6>),
           (<HandLandmark.INDEX_FINGER_MCP: 5>,
            <HandLandmark.MIDDLE_FINGER_MCP: 9>),
           (<HandLandmark.INDEX_FINGER_PIP: 6>,
            <HandLandmark.INDEX_FINGER_DIP: 7>),
           (<HandLandmark.INDEX_FINGER_DIP: 7>,
            <HandLandmark.INDEX_FINGER_TIP: 8>),
           (<HandLandmark.MIDDLE_FINGER_MCP: 9>,
            <HandLandmark.MIDDLE_FINGER_PIP: 10>),
           (<HandLandmark.MIDDLE_FINGER_MCP: 9>,
            <HandLandmark.RING_FINGER_MCP: 13>),
           (<HandLandmark.MIDDLE_FINGER_PIP: 10>,
            <HandLandmark.MIDDLE_FINGER_DIP: 11>),
           (<HandLandmark.MIDDLE_FINGER_DIP: 11>,
            <HandLandmark.MIDDLE_FINGER_TIP: 12>),
           (<HandLandmark.RING_FINGER_MCP: 13>,
            <HandLandmark.RING_FINGER_PIP: 14>),
           (<HandLandmark.RING_FINGER_MCP: 13>, <HandLandmark.PINKY_MCP: 17>),
           (<HandLandmark.RING_FINGER_PIP: 14>,
            <HandLandmark.RING_FINGER_DIP: 15>),
           (<HandLandmark.RING_FINGER_DIP: 15>,
            <HandLandmark.RING_FINGER_TIP: 16>),
           (<HandLandmark.PINKY_MCP: 17>, <HandLandmark.PINKY_PIP: 18>),
           (<HandLandmark.PINKY_PIP: 18>, <HandLandmark.PINKY_DIP: 19>),
           (<HandLandmark.PINKY_DIP: 19>, <HandLandmark.PINKY_TIP: 20>)})
```

In [11]:
```
#temp = result.right_hand_landmarks.landmark
```

In [12]:
```
#temp[9].x
```

In [13]:

```
R = 25
thickness = 2


webcam = 0
capture = cv.VideoCapture(webcam)

fps = int(capture.get(cv.CAP_PROP_FPS))
print("fps is "+str(fps))

_, frame = capture.read()
height, width, channel = frame.shape

for types in input_types:
    count = 0
    while capture.isOpened():
        #time.sleep()
        if (cv.waitKey(1) & 0xFF == 13) or (count >= min_input_count + 1):
            break

        black = np.zeros(shape = frame.shape)

        _, frame = capture.read()

        frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)

        result = holistic.process(frame_rgb)

        try:
            hand_landmarks = result.right_hand_landmarks.landmark
            if hand_landmarks:
                x_max = 0
                y_max = 0
                x_min = width
                y_min = height
                for i in range(0,21,1):
                    lm = hand_landmarks[i]
                    x, y = int(lm.x * width), int(lm.y * height)
                    if x > x_max:
                        x_max = x
                    if x < x_min:
```

```
                    x_min = x
                if y > y_max:
                    y_max = y
                if y < y_min:
                    y_min = y

            frame_bgr = cv.cvtColor(frame_rgb, cv.COLOR_RGB2BGR)

            mp_drawing.draw_landmarks(frame_bgr, result.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)

            cv.rectangle(frame_bgr, (x_min - R, y_min - R), (x_max + R, y_max + R), (0, 255, 0), thickness)

            result1 = frame_bgr


            '''
            work on black
            '''
            mp_drawing.draw_landmarks(black, result.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS)

            croped = black[y_min - R + thickness: y_max + R - thickness, x_min - R + thickness : x_max + R - thick
ness]


            resized =cv.resize(croped, (96, 96))

            mirror2 = cv.flip(resized, 1)

            result2 = mirror2

            cv.imshow("Frame2", result2)
        except:
            result1 = frame
            #result2 = black
            pass

        mirror1 = cv.flip(result1, 1)

        if count <= delay:
            delay_msg = "Class of " + types + " is going to start in " + str(delay - count)
            cv.putText(mirror1, delay_msg, (20, 20), cv.FONT_HERSHEY_PLAIN, 1, (0,0,255), 1)
        else:
            msg = "Class of " + types + " input no : " + str(count - delay)
```

```
            cv.putText(mirror1, msg, (50, 50), cv.FONT_HERSHEY_PLAIN, 1, (255,0,0), 1)
            try:
                path_mirror2 = "Inputs/" + types + "/"  + types + "_" + str(count - delay) + ".jpg"
                #cv.imwrite(path_mirror2, result2)
            except:
                pass
        count += 1

        cv.imshow('frame1', mirror1)
capture.release()
cv.destroyAllWindows()
```

```
fps is 30
```

In [ ]: