In [1]:
```
import os
import time
import numpy as np
from matplotlib import pyplot as plt
import cv2 as cv
import mediapipe as mp
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix ,classification_report
from gtts import gTTS
from playsound import playsound
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.1.0'
```

In [4]:
```
tf.test.is_gpu_available('gpu')
```

Out[4]:
```
True
```

In [39]:
```
input_types = ['palm', 'fist', 'thumbsup', 'gun', 'call']
```

In [40]:
```
description_types = ['01', '02', '03', '04', '05']
```

In [41]:
```
defination_types = ['a', 'b', 'c', 'd', 'e']
```

Load Model

In [6]:
```
cnn = models.load_model("cnn.h5")
```

In [7]:
```
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 96, 96, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 36864)             0
_________________________________________________________________
dense (Dense)                (None, 512)               18874880
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
Total params: 18,896,837
Trainable params: 18,896,837
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
l = []
for i in range(5):
    image_matrix = plt.imread('Inputs/call/' + 'call_' + str(i + 1) + '.jpg')
    l.append(image_matrix)
l = np.array(l)
l = l / 255
```

In [9]:
```
cnn.predict(l)
```

Out[9]:
```
array([[2.6413172e-10, 3.2406274e-13, 5.9979181e-11, 1.3836636e-07,
        9.9999988e-01],
       [4.7895166e-10, 3.6021318e-12, 5.4874799e-10, 3.0129550e-07,
        9.9999964e-01],
       [1.0681169e-10, 2.1262924e-13, 1.7447507e-11, 3.5530383e-07,
        9.9999964e-01],
       [3.1814351e-09, 3.2657731e-11, 6.4703887e-09, 5.8453224e-08,
        9.9999988e-01],
       [2.3002328e-10, 2.8322423e-12, 2.8186053e-10, 2.9070209e-08,
        1.0000000e+00]], dtype=float32)
```

In [10]:
```
l.shape
```

Out[10]:
```
(5, 96, 96, 3)
```

In [11]:
```
def Predict(img):
    class_ = np.argmax(cnn.predict(img))
    return input_types[class_]
```

In [12]:
```
Predict(np.array([l[0]]))
```

Out[12]:
```
'call'
```

New code

In [13]:
```
def saving_sounds():
    language = 'en'
    for types in input_types:
        obj = gTTS(text = types, lang = language, slow = False)
        if os.path.isfile(str(types) + ".mp3") is False:
            obj.save(str(types) + ".mp3")
```

In [14]:
```
saving_sounds()
```

In [15]:
```
types = 'thumbsup'
playsound(str(types) + ".mp3")
```

In [16]:
```
def playing_sounds():
    for types in input_types:
        playsound(str(types) + ".mp3")
```

In [18]:
```
playing_sounds()
```

linkers

In [42]:
```
def message_class(mirror1, class_):
    cv.putText(mirror1, str(class_), (100, 100), cv.FONT_HERSHEY_PLAIN, 2, (255,0,0), 1)
    return mirror1
```

In [43]:
```
def message_sound(class_):
#     if os.path.isfile(str(class_) + ".mp3") is False:
#         obj.save(str(class_) + ".mp3")
    playsound(str(class_) + ".mp3")
    block = True
```

In [44]:
```
message_sound("palm")
```

In [45]:
```
def message_description(black1, class_):
    text1 = description_types[input_types.index(class_)]
    cv.putText(black1, str(text1), (100, 100), cv.FONT_HERSHEY_PLAIN, 5, (255,0,0), 1)
    return black1
```

In [46]:
```
def message_defination(black2, class_):
    text2 = defination_types[input_types.index(class_)]
    cv.putText(black2, str(text2), (100, 100), cv.FONT_HERSHEY_PLAIN, 5, (255,0,0), 1)
    return black2
```

Real Time Acc

In [22]:
```
mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
```

In [23]:
```
holistic = mp_holistic.Holistic()
```

In [47]:

```
R = 25
thickness = 2


webcam = 0
capture = cv.VideoCapture(webcam)

pre_class = ''

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

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

while capture.isOpened():
    #time.sleep()
    if cv.waitKey(1) & 0xFF == 13:
        break

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

    black1 = np.zeros(shape = frame.shape)
    black2 = 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

            mirror1 = cv.flip(result1, 1)


            '''

            '''
            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 - thickness
]

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

            mirror2 = cv.flip(resized, 1)

            result2 = mirror2

            img_mat = np.array([result2])

            class_ = Predict(img_mat)

            # msg 01
            message_class(mirror1, class_)

            #msg 02
```

```
#             if pre_class != class_:
#                 message_sound(class_)
#                 pre_class = class_

            #msg 03
            black1 = message_description(black1, class_)
            result3 = black1

            #msg 04
            black2 = message_defination(black2, class_)
            result4 = black2

            cv.imshow("Frame2", result2)
            cv.imshow("Frame3", result3)
            cv.imshow("Frame4", result4)
    except:
        result1 = frame
        mirror1 = cv.flip(result1, 1)
        #result2 = black
        pass



    cv.imshow('frame1', mirror1)

capture.release()
cv.destroyAllWindows()
```

```
fps is 30
```

In [ ]: