```
!pip install opencv-python-headless
!pip install mediapipe

Requirement already satisfied: opencv-python-headless in
/usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in
/usr/local/lib/python3.10/dist-packages (from opencv-python-headless)
(1.25.2)
Collecting mediapipe
  Downloading mediapipe-0.10.14-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (35.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.7/35.7 MB 28.9 MB/s eta
0:00:00
ent already satisfied: absl-py in /usr/local/lib/python3.10/dist-
packages (from mediapipe) (1.4.0)
Requirement already satisfied: attrs>=19.1.0 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (23.2.0)
Requirement already satisfied: flatbuffers>=2.0 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (24.3.25)
Requirement already satisfied: jax in /usr/local/lib/python3.10/dist-
packages (from mediapipe) (0.4.26)
Requirement already satisfied: jaxlib in
/usr/local/lib/python3.10/dist-packages (from mediapipe)
(0.4.26+cuda12.cudnn89)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (3.7.1)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (1.25.2)
Requirement already satisfied: opencv-contrib-python in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (4.8.0.76)
Collecting protobuf<5,>=4.25.3 (from mediapipe)
  Downloading protobuf-4.25.3-cp37-abi3-manylinux2014_x86_64.whl (294
kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 294.6/294.6 kB 23.3 MB/s eta
0:00:00
 mediapipe)
  Downloading sounddevice-0.4.7-py3-none-any.whl (32 kB)
Requirement already satisfied: CFFI>=1.0 in
/usr/local/lib/python3.10/dist-packages (from sounddevice>=0.4.4-
>mediapipe) (1.16.0)
Requirement already satisfied: ml-dtypes>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (0.2.0)
Requirement already satisfied: opt-einsum in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (3.3.0)
Requirement already satisfied: scipy>=1.9 in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (1.11.4)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(1.2.1)
Requirement already satisfied: cycler>=0.10 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(2.8.2)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.10/dist-packages (from CFFI>=1.0-
>sounddevice>=0.4.4->mediapipe) (2.22)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->mediapipe) (1.16.0)
Installing collected packages: protobuf, sounddevice, mediapipe
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.3
    Uninstalling protobuf-3.20.3:
      Successfully uninstalled protobuf-3.20.3
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
tensorflow-metadata 1.15.0 requires protobuf<4.21,>=3.20.3;
python_version < "3.11", but you have protobuf 4.25.3 which is
incompatible.
Successfully installed mediapipe-0.10.14 protobuf-4.25.3 sounddevice-
0.4.7
```

```
import cv2
import mediapipe as mp
from google.colab.patches import cv2_imshow

import matplotlib.pyplot as plt



from IPython.display import display, Javascript
from google.colab.output import eval_js
```

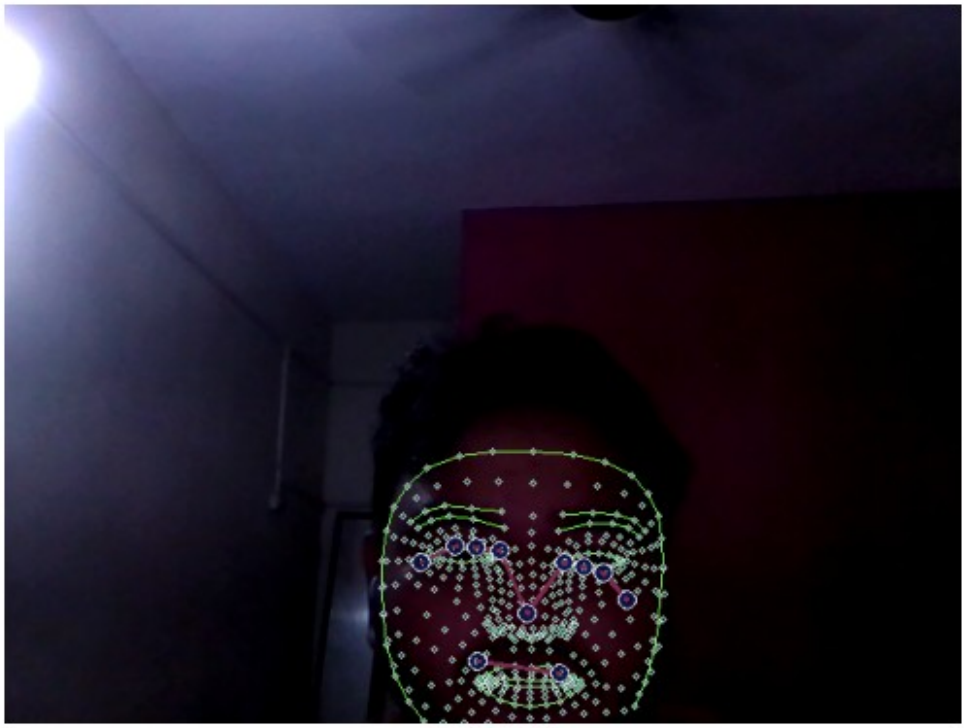
```
from base64 import b64decode
import cv2
import numpy as np
import mediapipe as mp
import matplotlib.pyplot as plt

def js_to_image(js_reply):
    """
    Convert the JavaScript object to an OpenCV image.
    """
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    return img

def video_stream():
    """
    Launch the video stream in the browser and capture an image.
    """
    js = Javascript('''
    async function stream() {
        const video = document.createElement('video');
        video.style.display = 'block';
        const stream = await
navigator.mediaDevices.getUserMedia({video: true});
        document.body.appendChild(video);
        video.srcObject = stream;
        await video.play();
        // Resize output to fit the screen.

google.colab.output.setIframeHeight(document.documentElement.scrollHei
ght, true);

        // Create a canvas element.
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');

        // Draw the video frame to the canvas.
        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        // Convert the canvas image to a base64 string.
        const image = canvas.toDataURL('image/png');
        stream.getTracks()[0].stop();
        return image;
    }
    ''')
    display(js)
    data = eval_js('stream()')
```

```
    return data

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # COLOR CONVERSION
BGR 2 RGB
    image.flags.writeable = False  # Image is no longer writeable
    results = model.process(image)  # Make prediction
    image.flags.writeable = True  # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # COLOR COVERSION
RGB 2 BGR
    return image, results

def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACE_CONNECTIONS)  # Draw face connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS)  # Draw pose connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)  # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS)  # Draw right hand connections

def draw_styled_landmarks(image, results):
    # Draw face connections
    if results.face_landmarks:
        mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,

mp_drawing.DrawingSpec(color=(80,110,10), thickness=1,
circle_radius=1),

mp_drawing.DrawingSpec(color=(80,256,121), thickness=1,
circle_radius=1)
                                  )

    # Draw pose connections
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,

mp_drawing.DrawingSpec(color=(80,22,10), thickness=2,
circle_radius=4),

mp_drawing.DrawingSpec(color=(80,44,121), thickness=2,
circle_radius=2)
                                  )

    # Draw left hand connections
    if results.left_hand_landmarks:
        mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
```

```
mp_holistic.HAND_CONNECTIONS,

mp_drawing.DrawingSpec(color=(121,22,76), thickness=2,
circle_radius=4),

mp_drawing.DrawingSpec(color=(121,44,250), thickness=2,
circle_radius=2)
                                 )

    # Draw right hand connections
    if results.right_hand_landmarks:
        mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,

mp_drawing.DrawingSpec(color=(245,117,66), thickness=2,
circle_radius=4),

mp_drawing.DrawingSpec(color=(245,66,230), thickness=2,
circle_radius=2)
                                 )


mp_holistic = mp.solutions.holistic  # Holistic model
mp_drawing = mp.solutions.drawing_utils  # Drawing utilities

# Capture an image from the webcam
data = video_stream()
image = js_to_image(data)

# Display the captured image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

# Perform landmark detection
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    image, results = mediapipe_detection(image, holistic)
    draw_styled_landmarks(image, results)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
/usr/local/lib/python3.10/dist-packages/google/protobuf/
symbol_database.py:55: UserWarning: SymbolDatabase.GetPrototype() is
deprecated. Please use message_factory.GetMessageClass() instead.
SymbolDatabase.GetPrototype() will be removed soon.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
```

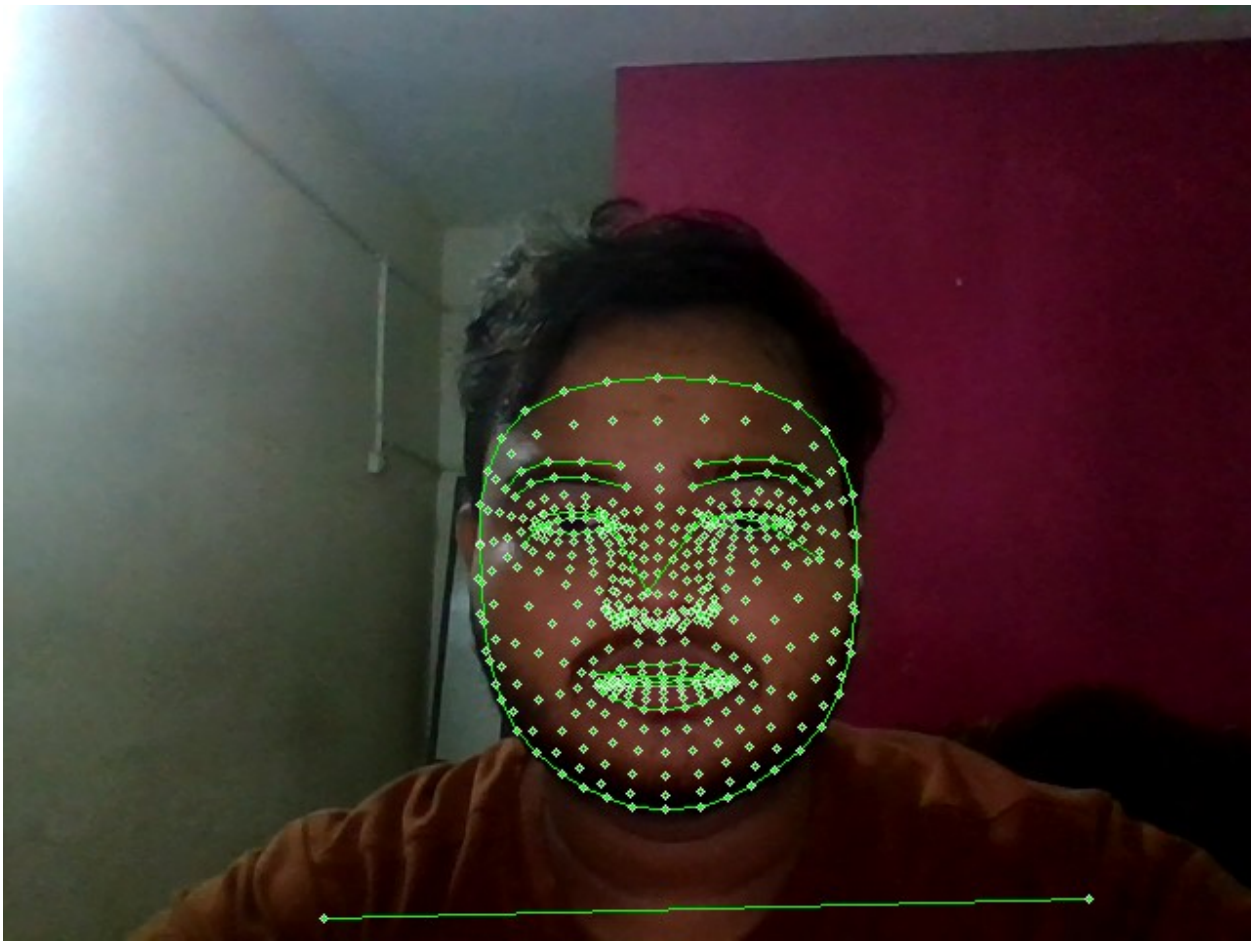


```
import cv2
import mediapipe as mp
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np

# Function to convert JavaScript object to OpenCV image
def js_to_image(js_reply):
    """
    Convert the JavaScript object to an OpenCV image.
    """
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    return img

# Function to stream video from webcam
def video_stream():
    """
    Launch the video stream in the browser and capture an image.
    """
    js = Javascript('''
    async function stream() {
        const video = document.createElement('video');
        video.style.display = 'block';
        const stream = await
```

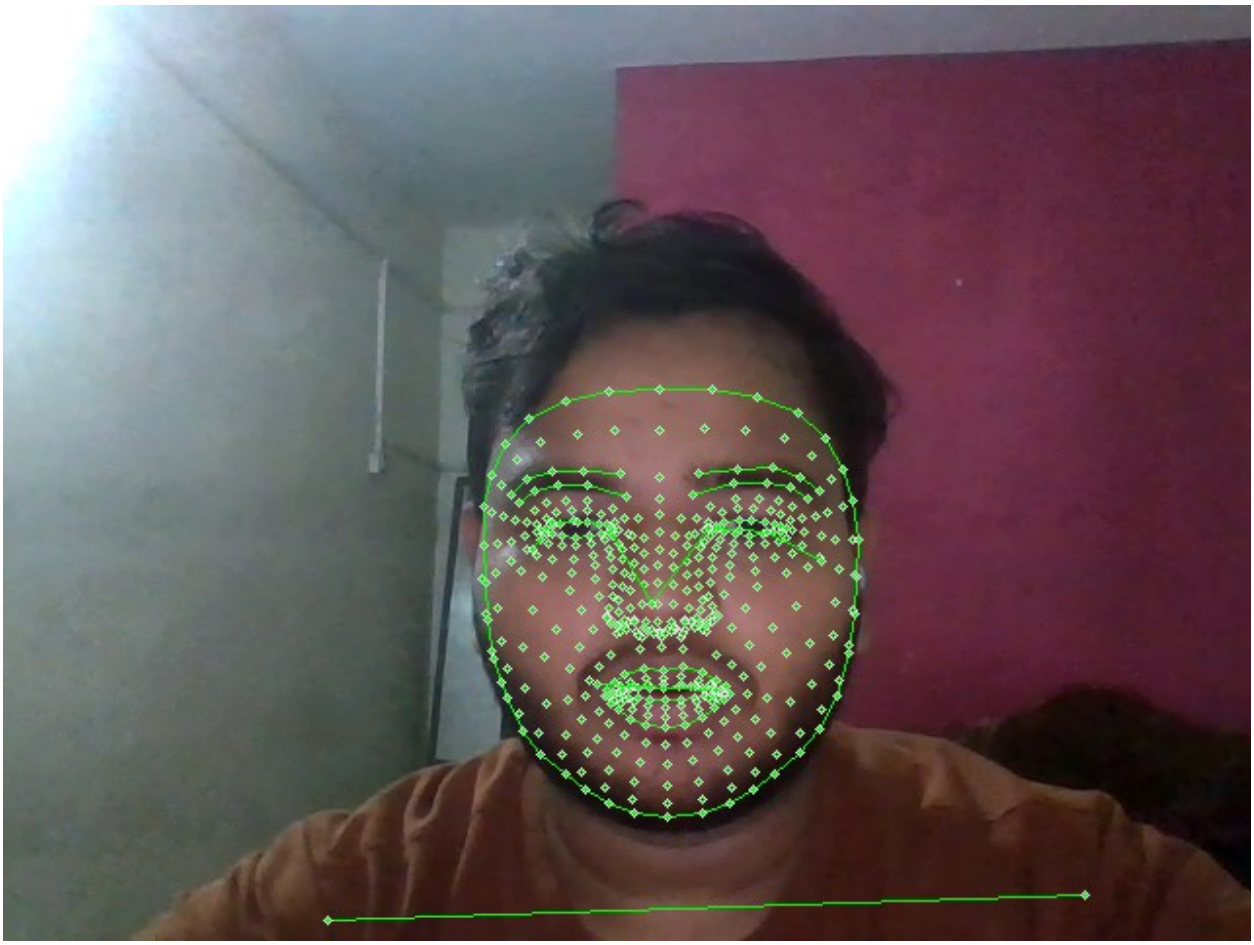
```
navigator.mediaDevices.getUserMedia({video: true});
        document.body.appendChild(video);
        video.srcObject = stream;
        await video.play();
        // Resize output to fit the screen.

google.colab.output.setIframeHeight(document.documentElement.scrollHei
ght, true);

        // Create a canvas element.
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');

        // Draw the video frame to the canvas.
        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        // Convert the canvas image to a base64 string.
        const image = canvas.toDataURL('image/png');
        stream.getTracks()[0].stop();
        return image;
    }
    ''')
    display(js)
    data = eval_js('stream()')
    return data

# Function to perform landmark detection using Mediapipe
def detect_landmarks(image, holistic):
    """
    Perform landmark detection on the given image using Mediapipe.
    """
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = holistic.process(image_rgb)
    return results

# Function to draw landmarks on the image
def draw_landmarks(image, results, holistic):
    """
    Draw landmarks (face, pose, hands) on the image.
    """
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
```

```
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Start streaming video from webcam
data_url = video_stream()

# Convert data URL to OpenCV image
image = js_to_image(data_url)

# Main loop for processing video stream
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    try:
        while True:
            # Perform landmark detection on the frame
            results = detect_landmarks(image, holistic)

            # Draw landmarks on the image
            draw_landmarks(image, results, holistic)

            # Display the image with landmarks
            cv2_imshow(image)

            # Capture next frame
            data_url = video_stream()
            image = js_to_image(data_url)

    except KeyboardInterrupt:
        print("Video stream interrupted.")
```

```
<IPython.core.display.Javascript object>
```



```
<IPython.core.display.Javascript object>
```



```
<IPython.core.display.Javascript object>

Video stream interrupted.

len(results.left_hand_landmarks.landmark)

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-db886034048d> in <cell line: 1>()
----> 1 len(results.left_hand_landmarks.landmark)

AttributeError: 'NoneType' object has no attribute 'landmark'

pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)

pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks
```

```
else np.zeros(132)
face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(1404)
lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)

face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(1404)

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks
else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

result_test = extract_keypoints(results)

result_test

array([ 0.51938146,  0.64024031, -1.87879813, ...,  0.        ,
        0.        ,  0.        ])

np.save('0', result_test)

np.load('0.npy')

array([ 0.51938146,  0.64024031, -1.87879813, ...,  0.        ,
        0.        ,  0.        ])



import os
import numpy as np

DATA_PATH = os.path.join('MP_Data')
actions = np.array(['hello', 'thanks', 'iloveyou'])
no_sequences = 30
```

```
sequence_length = 30
start_folder = 30

for action in actions:
    action_path = os.path.join(DATA_PATH, action)
    if not os.path.exists(action_path):
        os.makedirs(action_path)  # Create the action directory if it
doesn't exist

    dir_list = os.listdir(action_path)
    if dir_list:  # Check if the directory list is not empty
        dirmax = np.max(np.array(dir_list).astype(int))
    else:
        dirmax = 0  # If no directories found, start with 0

    for sequence in range(1, no_sequences + 1):
        try:
            os.makedirs(os.path.join(action_path, str(dirmax +
sequence)))
        except FileExistsError:
            pass
```

```
import cv2
import mediapipe as mp
from google.colab.patches import cv2_imshow
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
import os
import time

# Function to convert JavaScript object to OpenCV image
def js_to_image(js_reply):
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    return img

# Function to stream video from webcam
def video_stream():
    js = Javascript('''
    async function stream() {
        const video = document.createElement('video');
        video.style.display = 'block';
```

```
        const stream = await
navigator.mediaDevices.getUserMedia({video: true});
        document.body.appendChild(video);
        video.srcObject = stream;
        await video.play();
        // Resize output to fit the screen.

google.colab.output.setIframeHeight(document.documentElement.scrollHei
ght, true);

        // Create a canvas element.
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');

        // Draw the video frame to the canvas.
        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        // Convert the canvas image to a base64 string.
        const image = canvas.toDataURL('image/png');
        stream.getTracks()[0].stop();
        return image;
    }
    ''')
    display(js)
    data = eval_js('stream()')
    return data

# Function to perform landmark detection using Mediapipe
def mediapipe_detection(image, model):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_rgb.flags.writeable = False
    results = model.process(image_rgb)
    image_rgb.flags.writeable = True
    image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
    return image, results

# Function to draw landmarks on the image
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
```

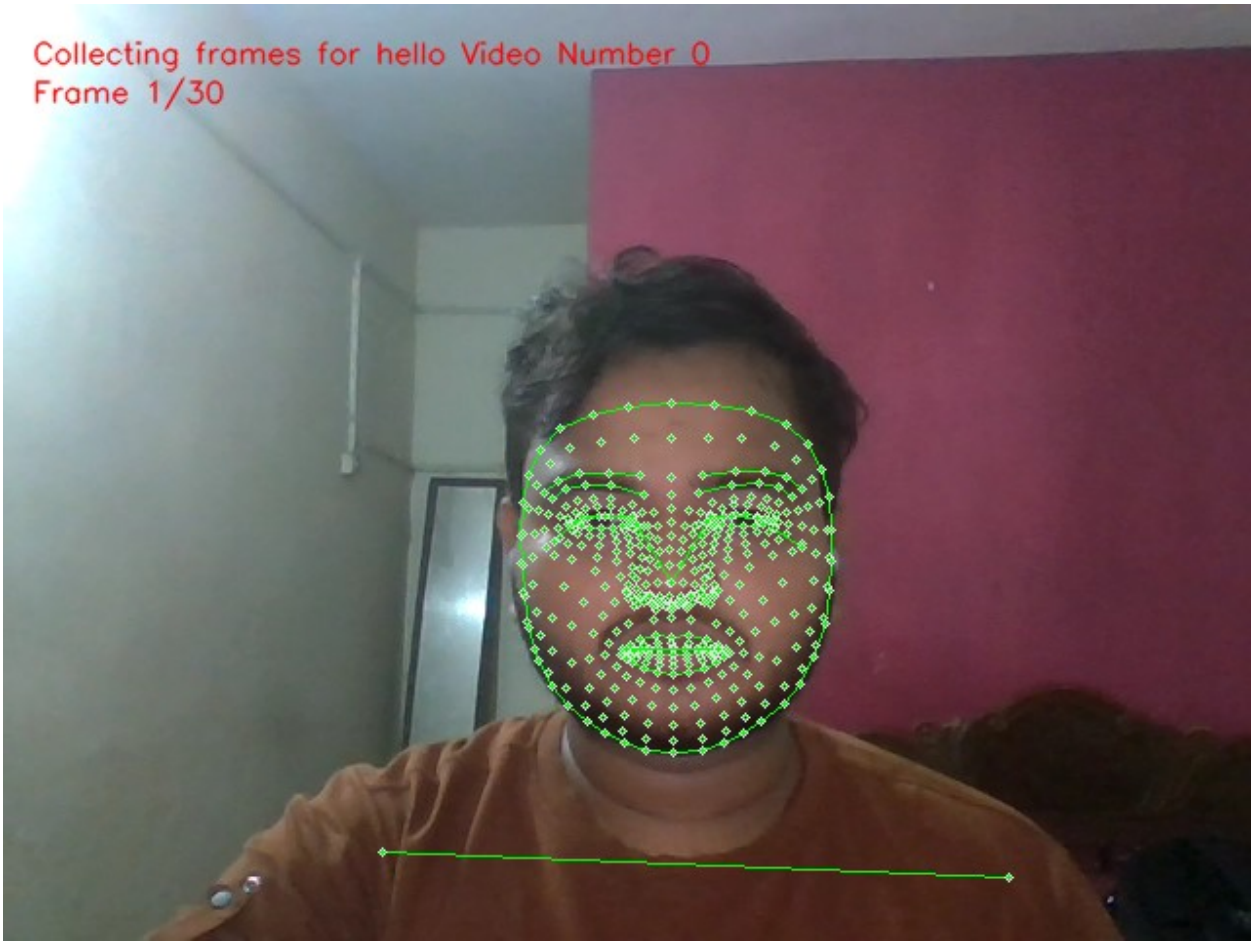
```
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.array(['hello', 'thanks', 'iloveyou'])

# Thirty videos worth of data
no_sequences = 30

# Videos are going to be 30 frames in length
sequence_length = 30

# Folder start
start_folder = 0

# Ensure directories exist
for action in actions:
    for sequence in range(start_folder, start_folder + no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action,
str(sequence)))
        except:
            pass

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks
else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
```

```
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

# Initialize Mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    # Loop through actions
    for action in actions:
        print(f"Collecting frames for action: {action}")
        # Loop through sequences aka videos
        for sequence in range(start_folder, start_folder +
no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):
                # Capture frame from webcam
                data_url = video_stream()
                frame = js_to_image(data_url)

                # Check if frame is captured successfully
                if frame is None:
                    print(f"Error: Frame {frame_num} not captured.")
                    continue

                # Make detections
                image, results = mediapipe_detection(frame, holistic)

                # Draw landmarks
                draw_styled_landmarks(image, results)

                # Apply wait logic and display frame info
                cv2.putText(image, f'Collecting frames for {action}
Video Number {sequence}', (15, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
255), 1, cv2.LINE_AA)
                cv2.putText(image, f'Frame {frame_num +
1}/{sequence_length}', (15, 50),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
255), 1, cv2.LINE_AA)
                cv2_imshow(image)

                # Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action,
str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                # Reduce wait time
```

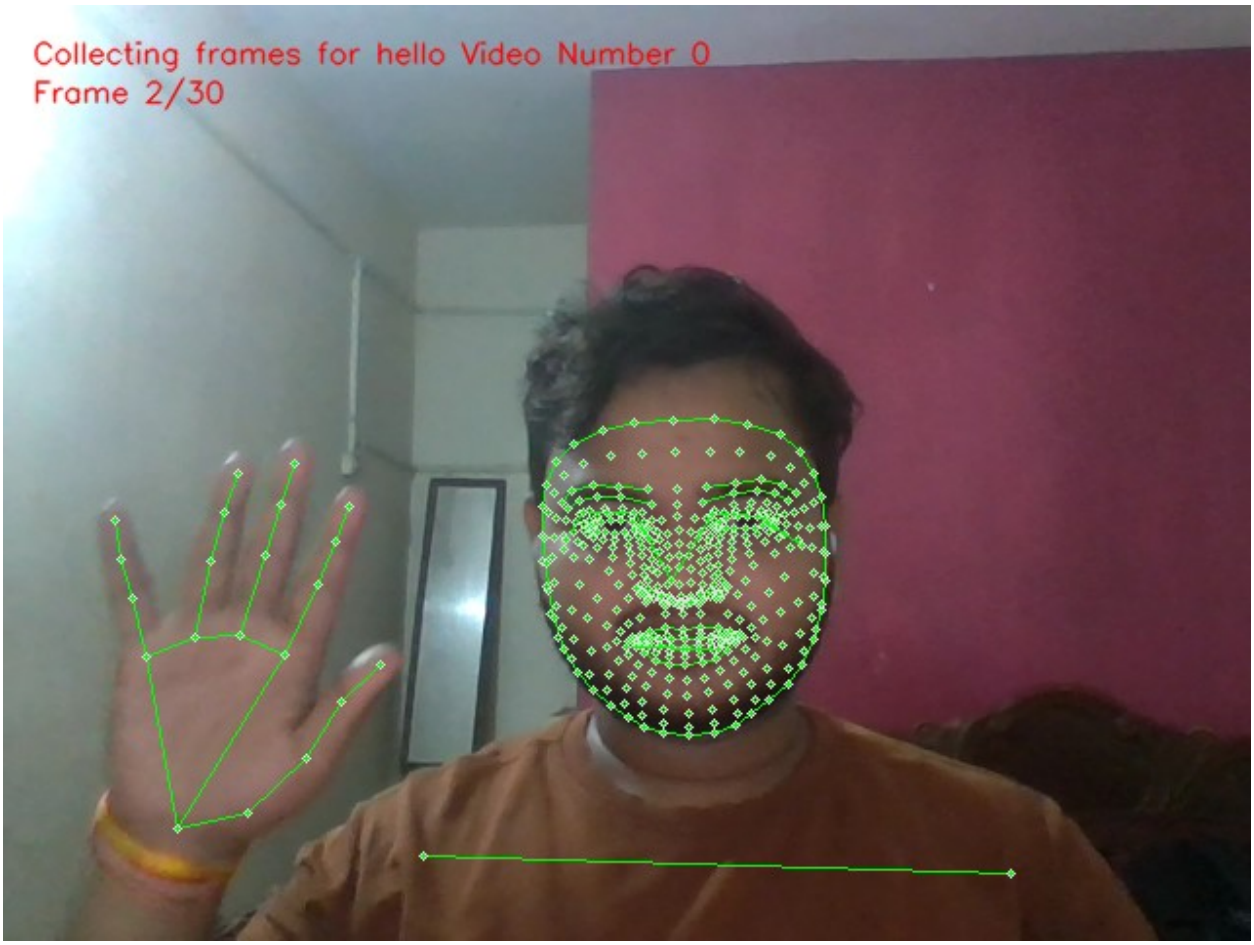
```
                time.sleep(0.1)  # Slight delay to avoid overloading
the system

print("Data collection complete.")
```

```
Collecting frames for action: hello
```

```
<IPython.core.display.Javascript object>
```



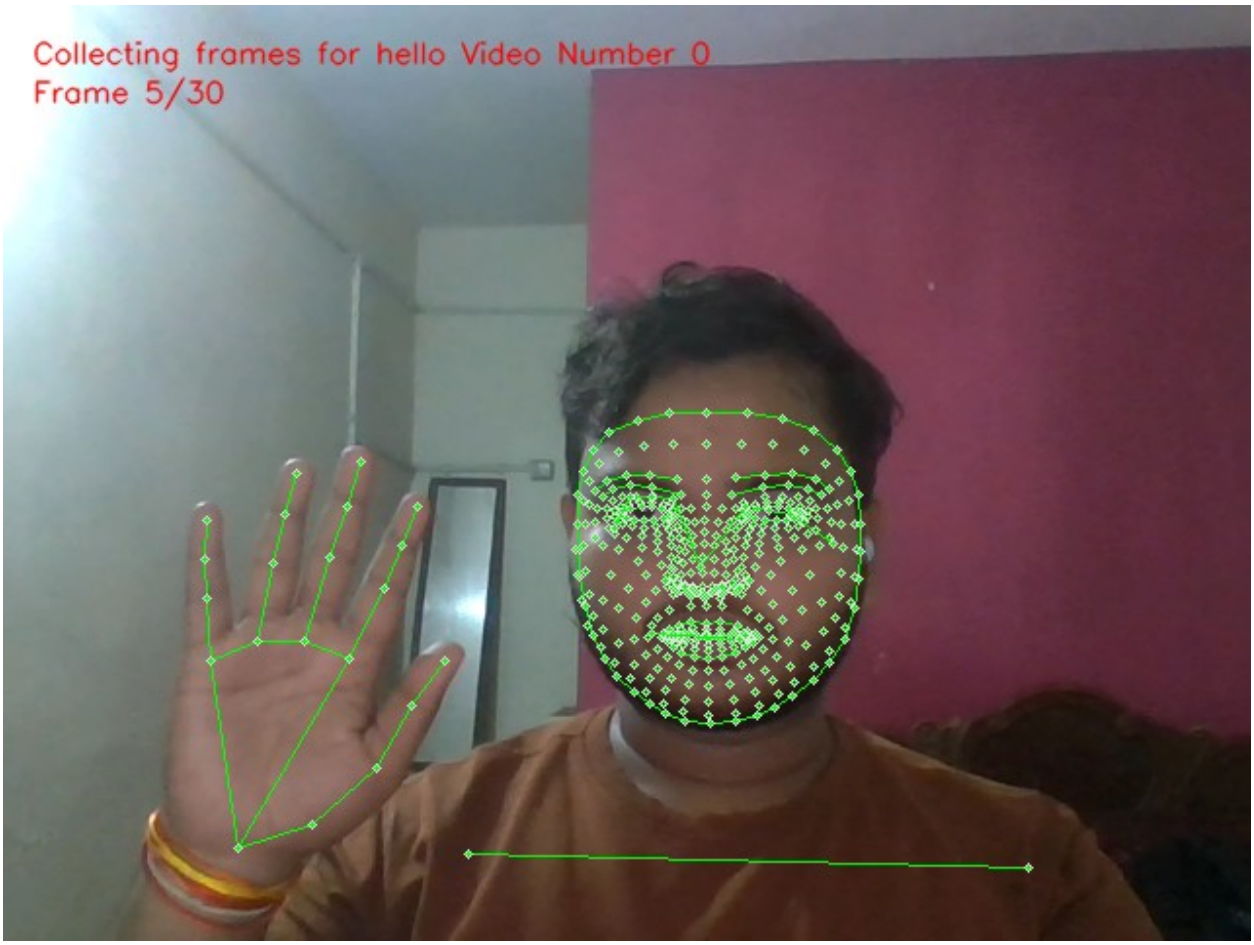

```
<IPython.core.display.Javascript object>
```



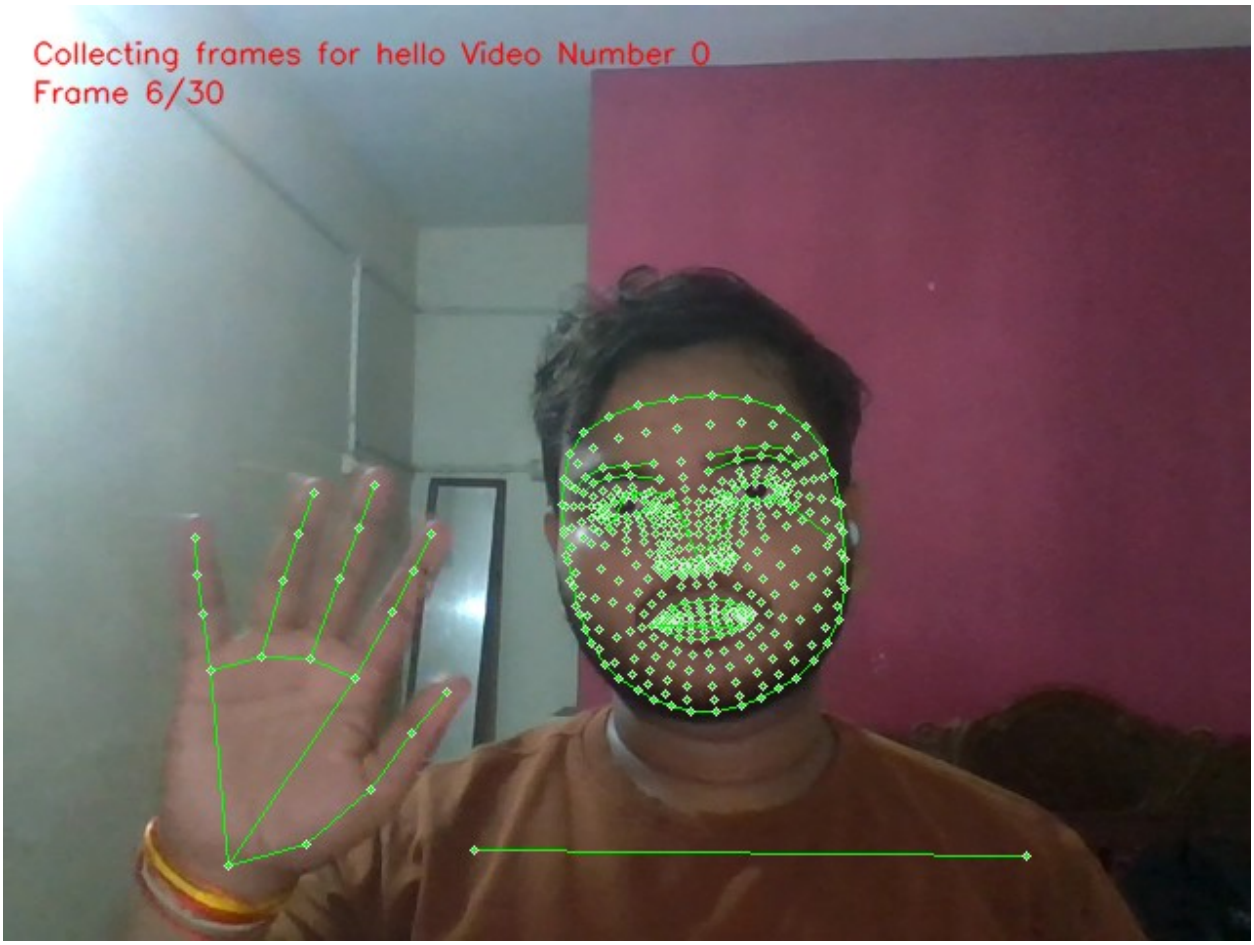

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

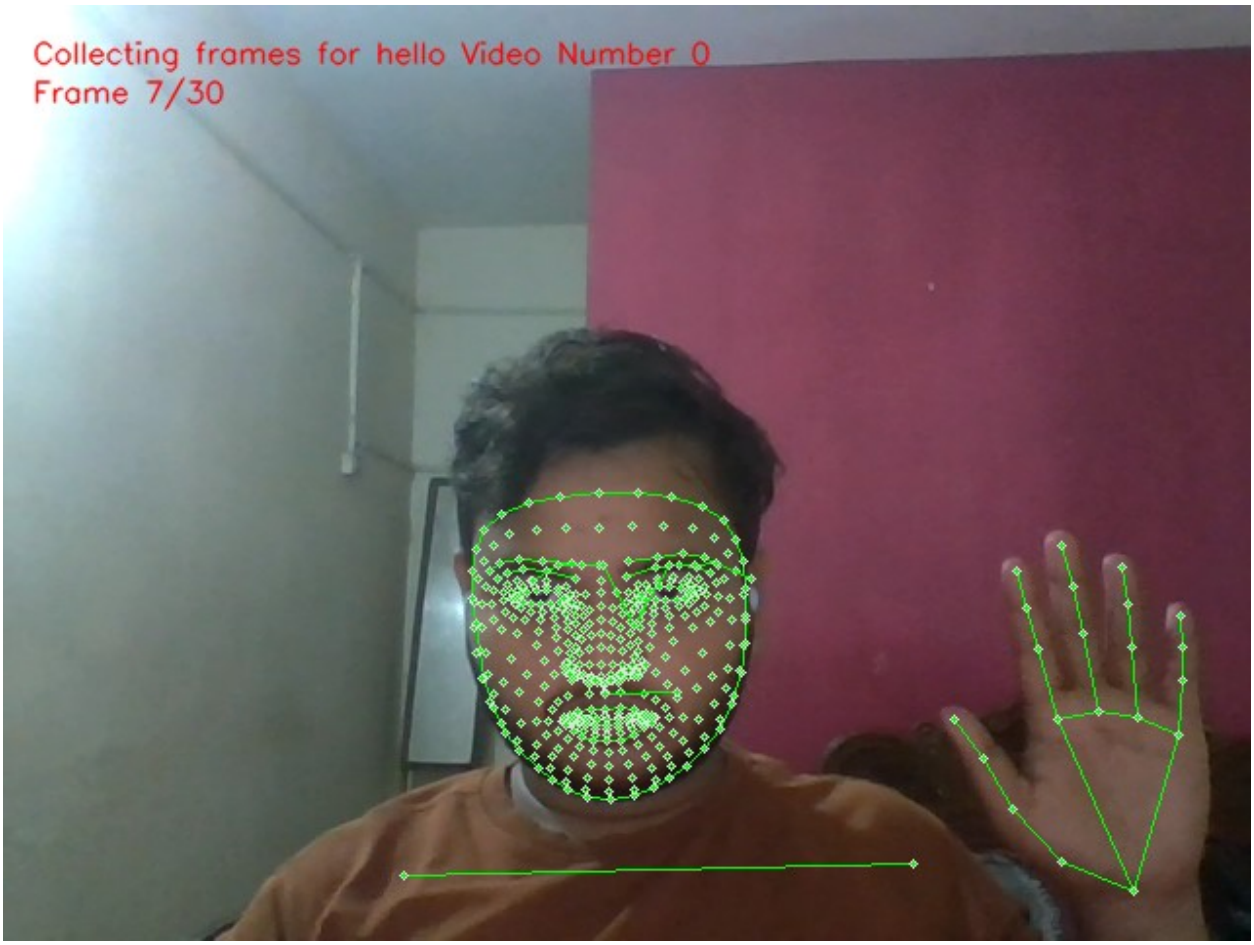

```
<IPython.core.display.Javascript object>
```



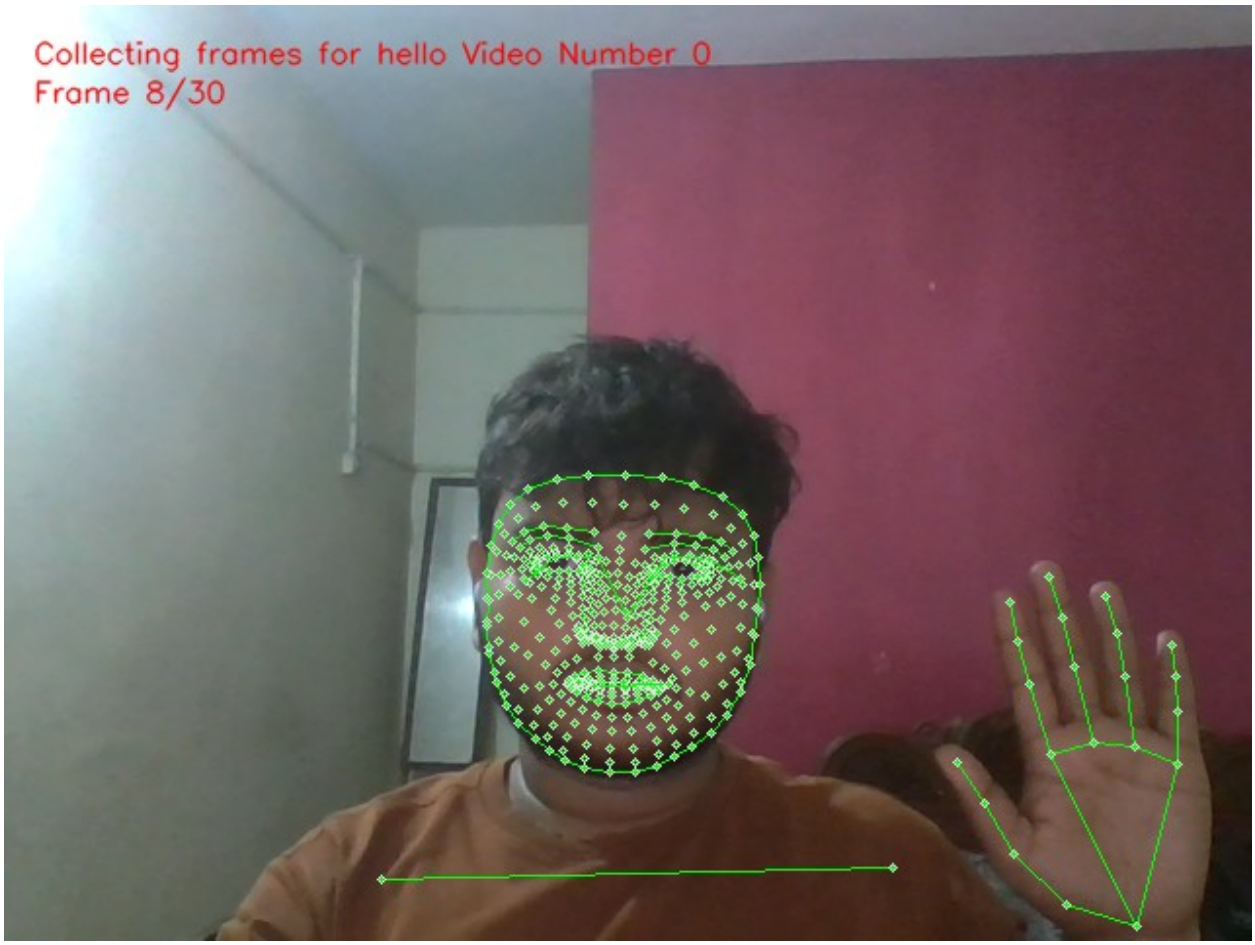

```
<IPython.core.display.Javascript object>
```



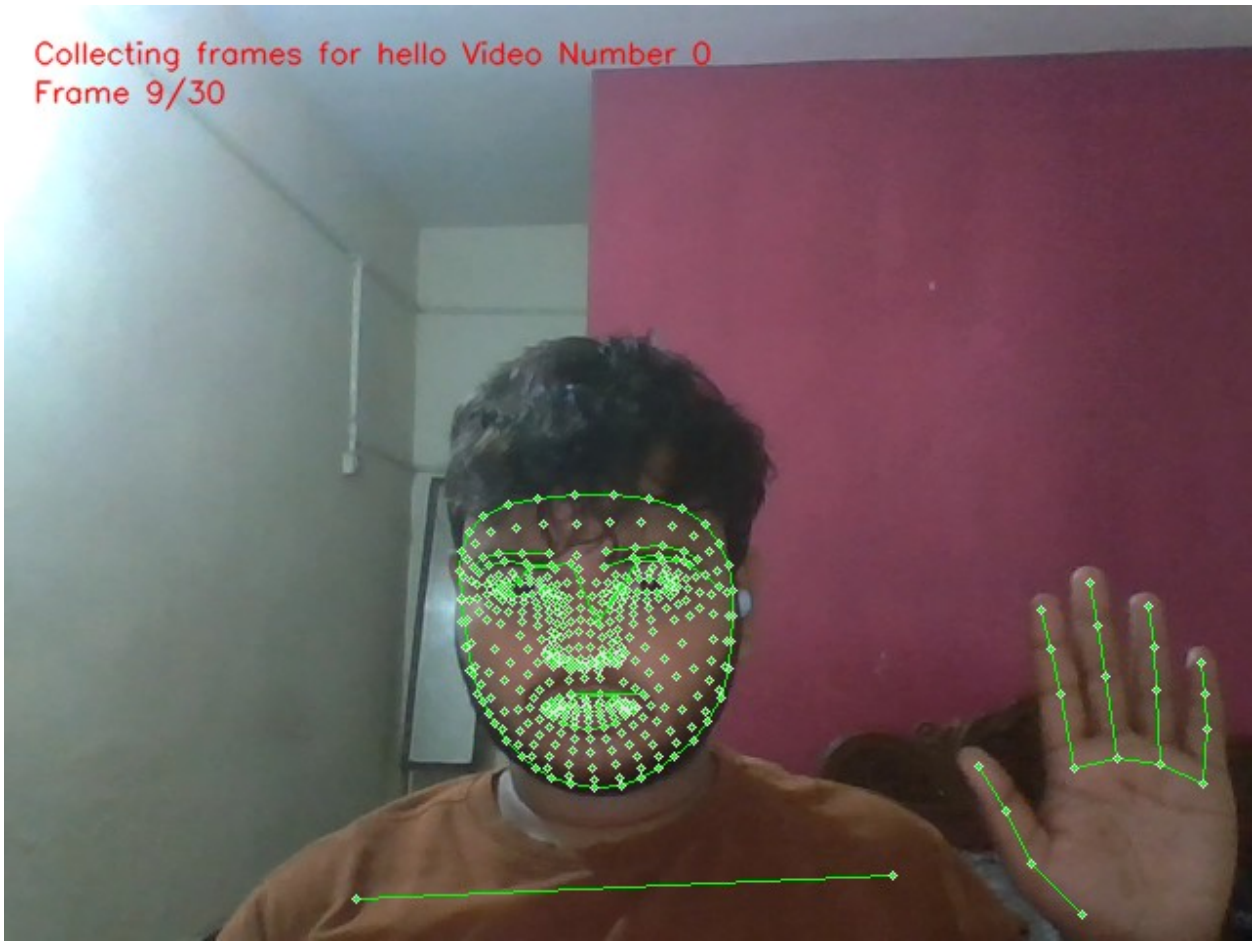

```
<IPython.core.display.Javascript object>
```



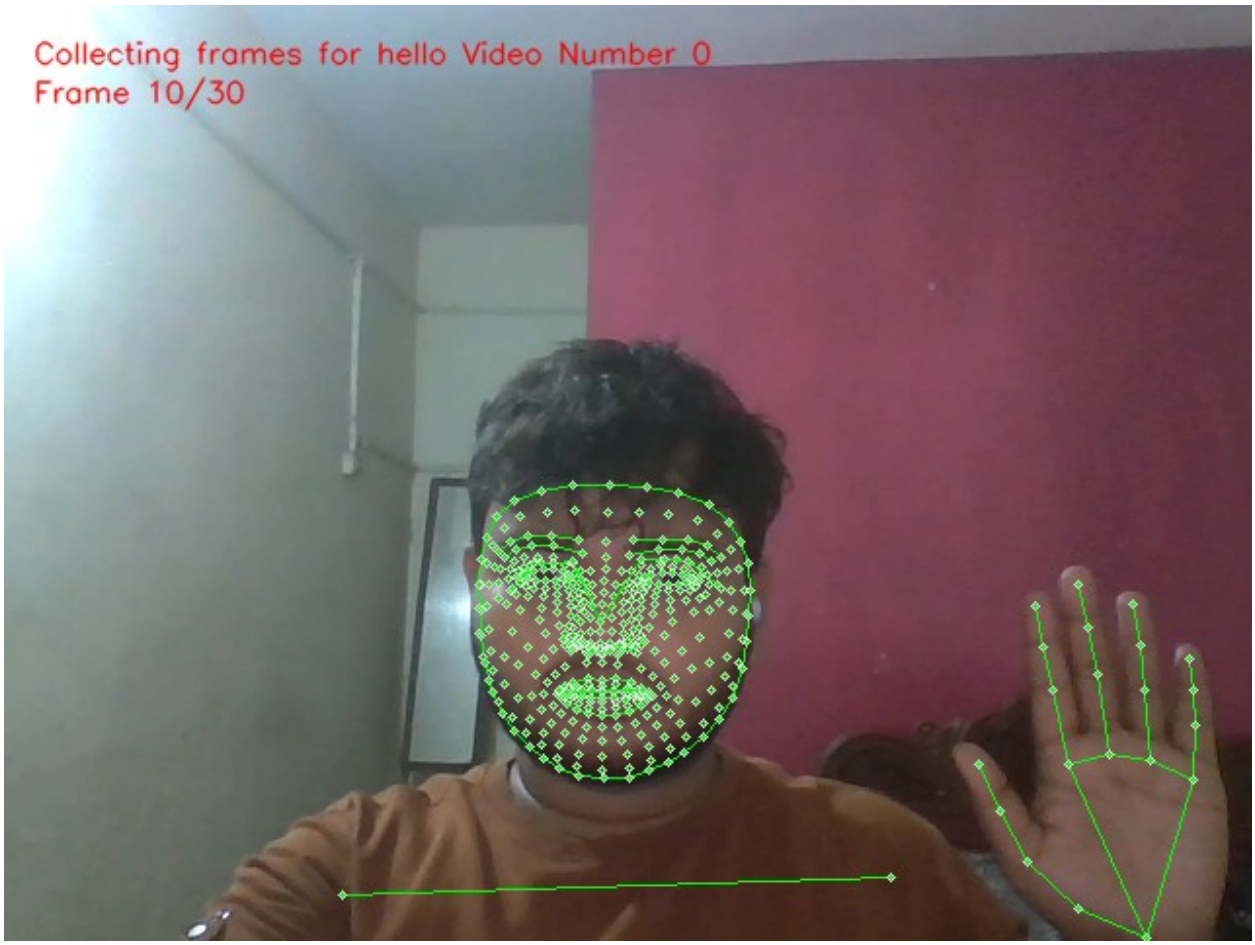

```
<IPython.core.display.Javascript object>
```



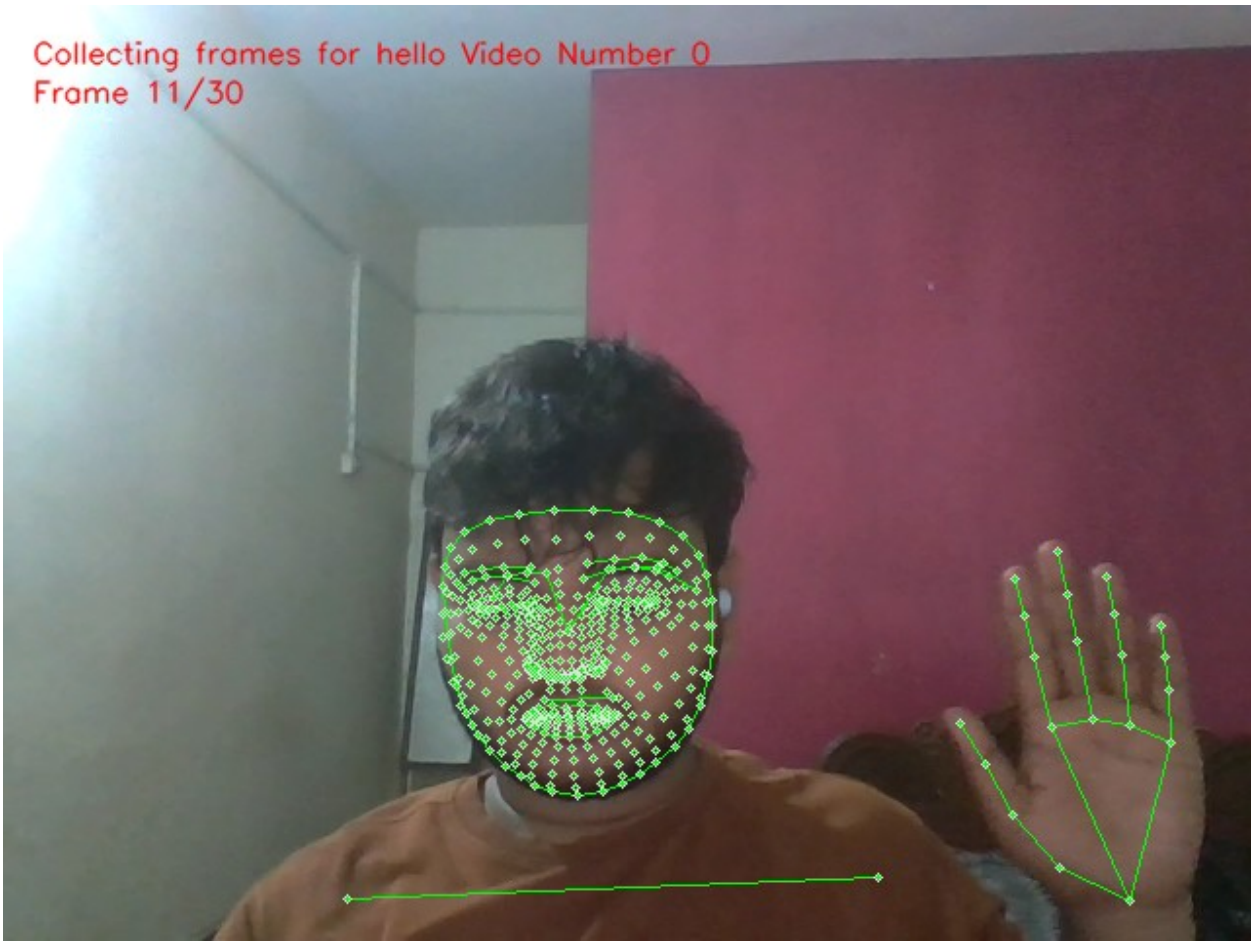

```
<IPython.core.display.Javascript object>
```



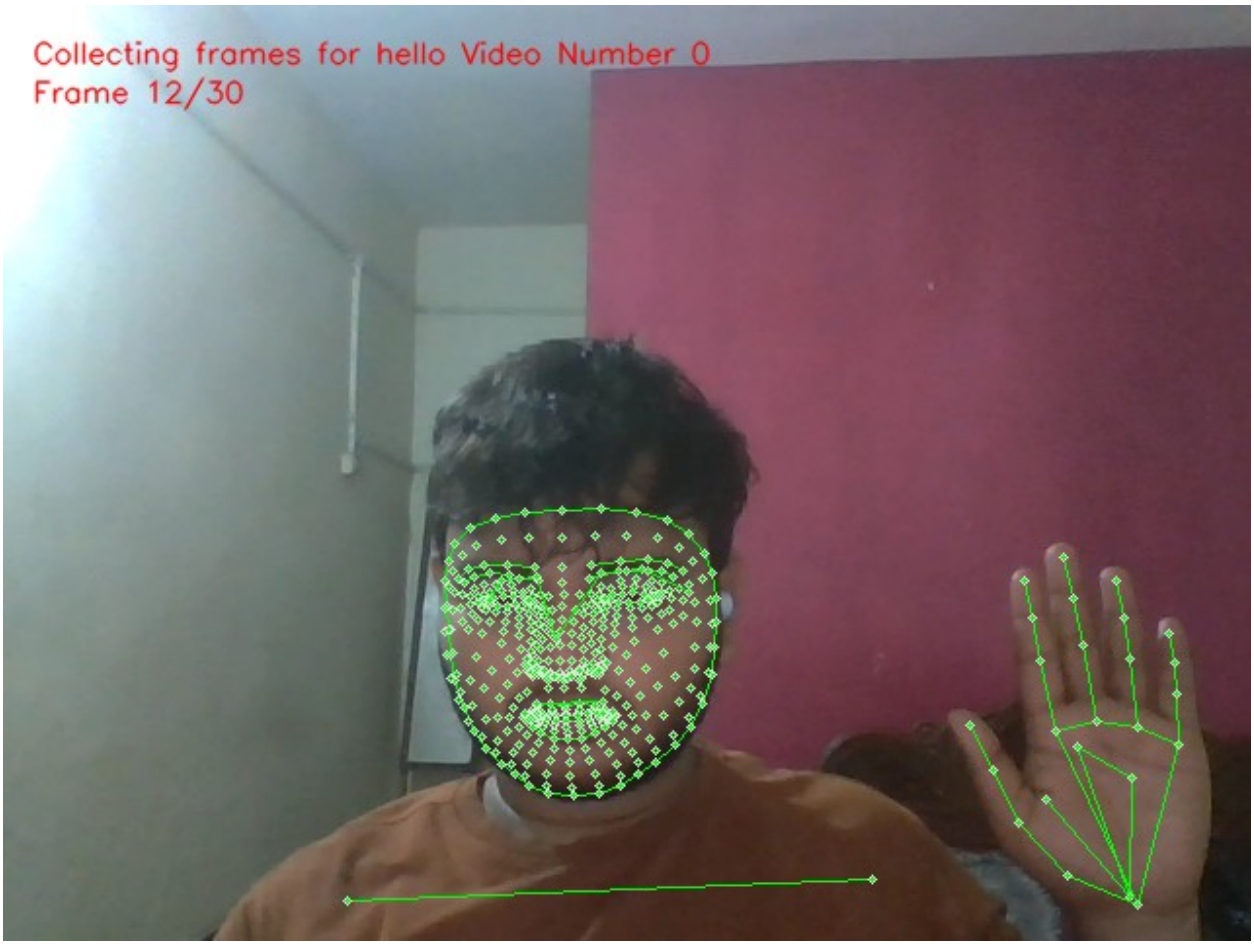

```
<IPython.core.display.Javascript object>
```



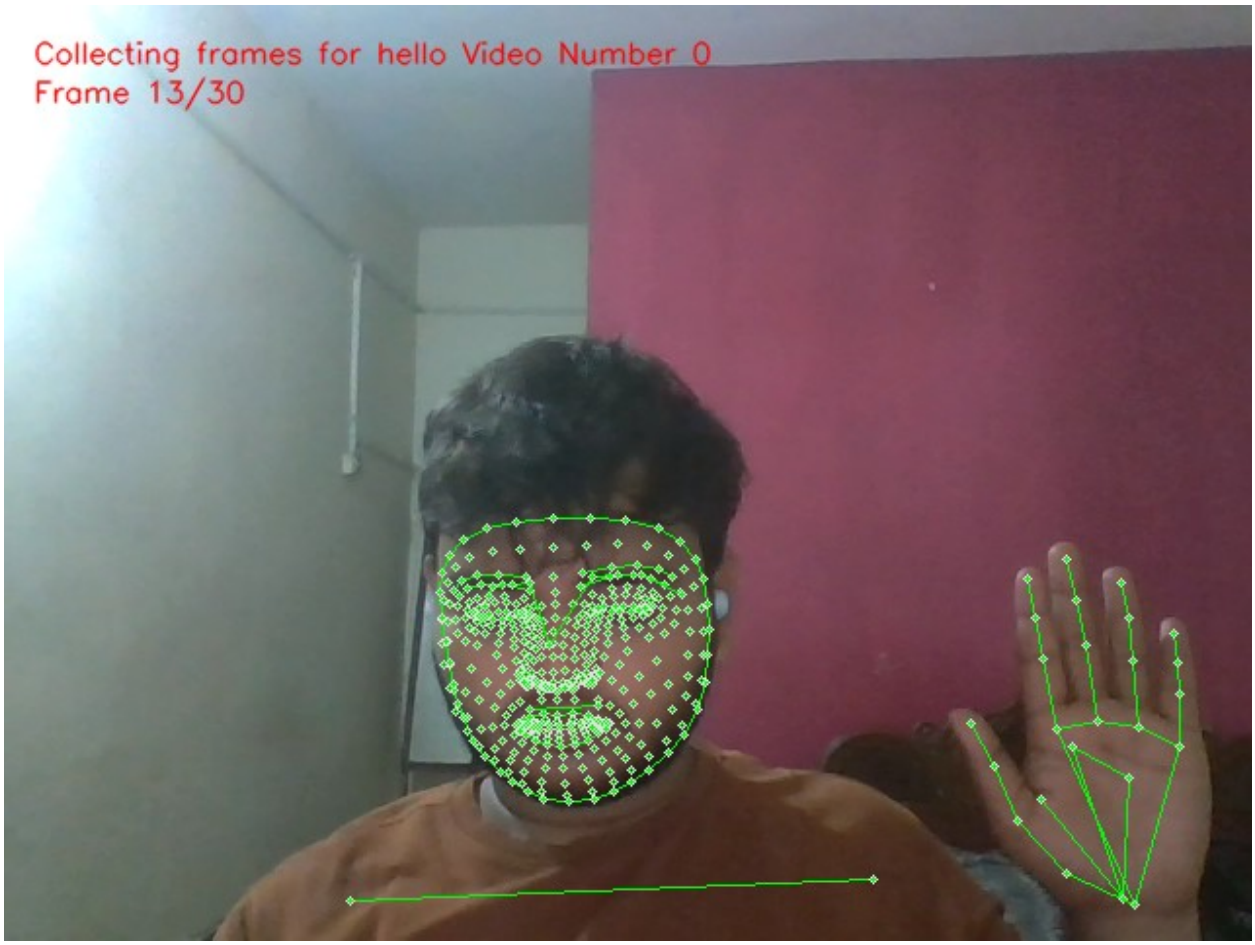

```
<IPython.core.display.Javascript object>
```



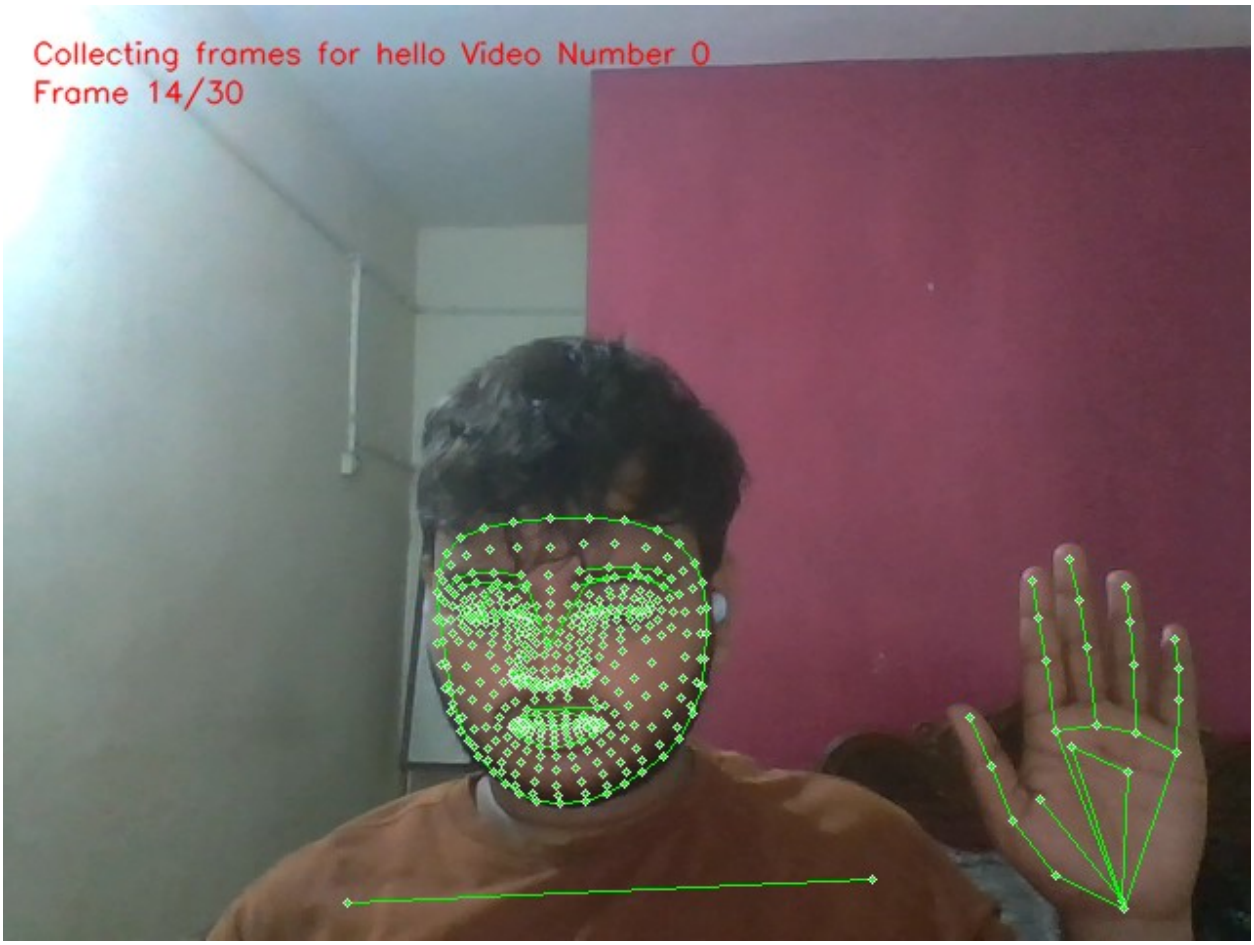

```
<IPython.core.display.Javascript object>
```



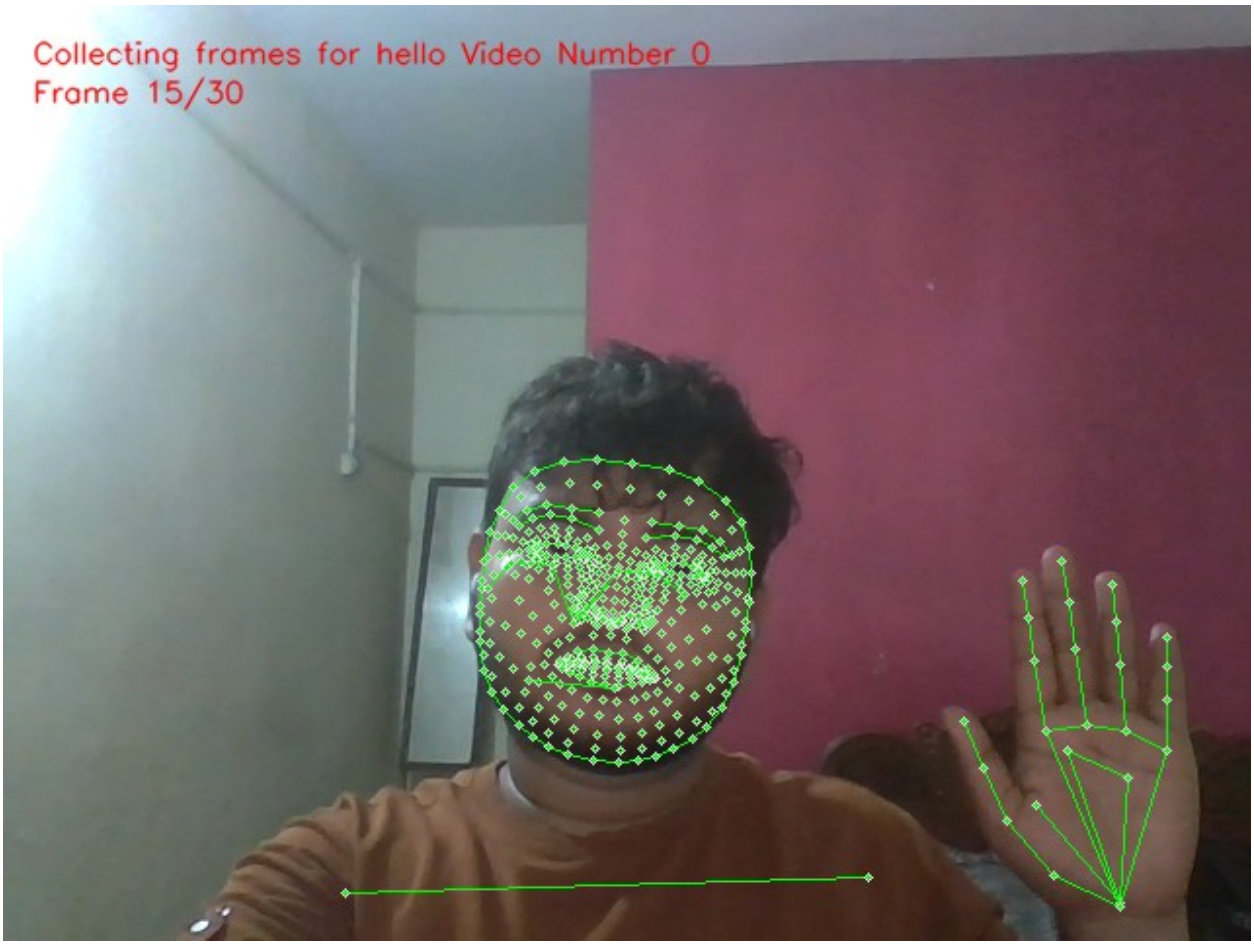

```
<IPython.core.display.Javascript object>
```



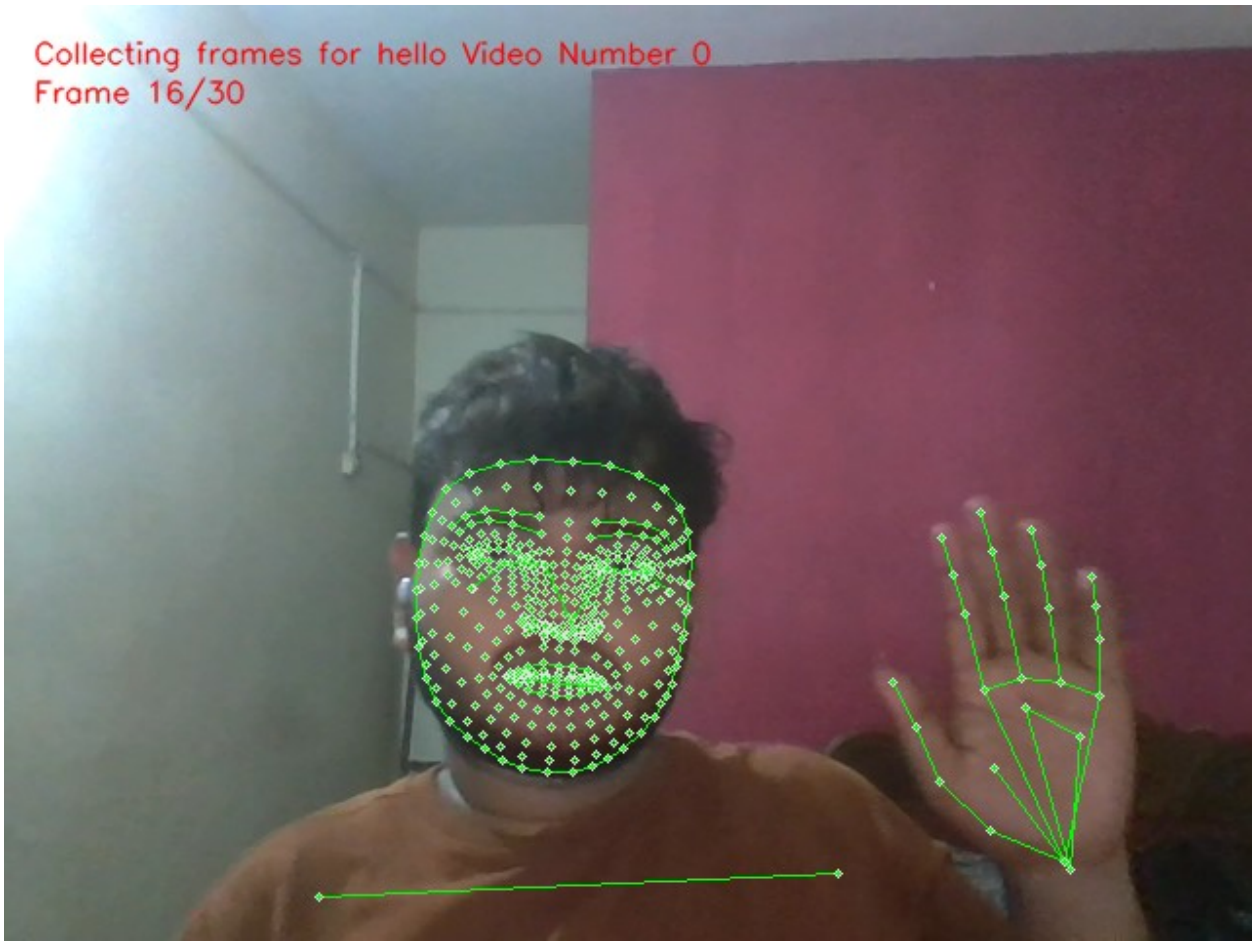

```
<IPython.core.display.Javascript object>
```



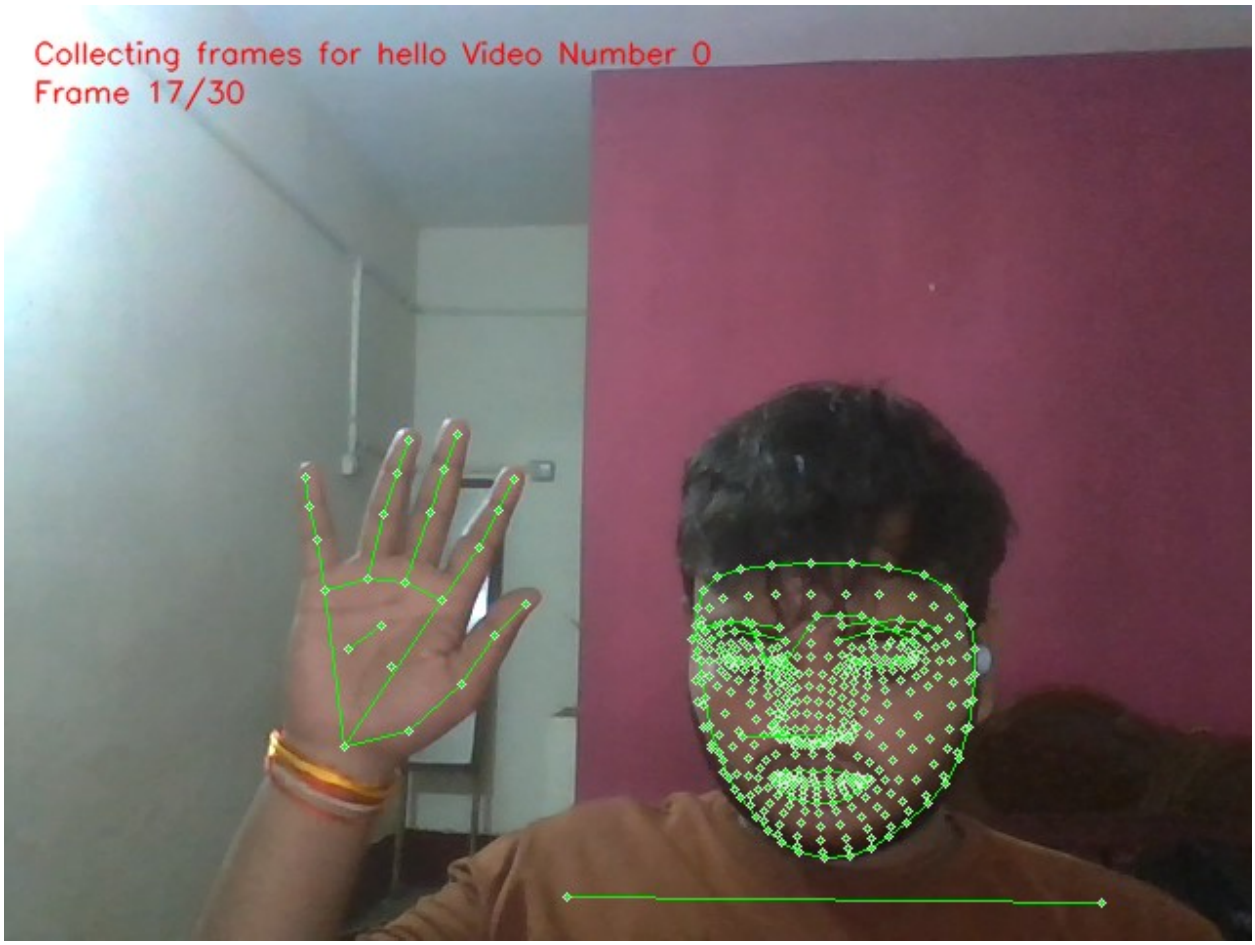

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```



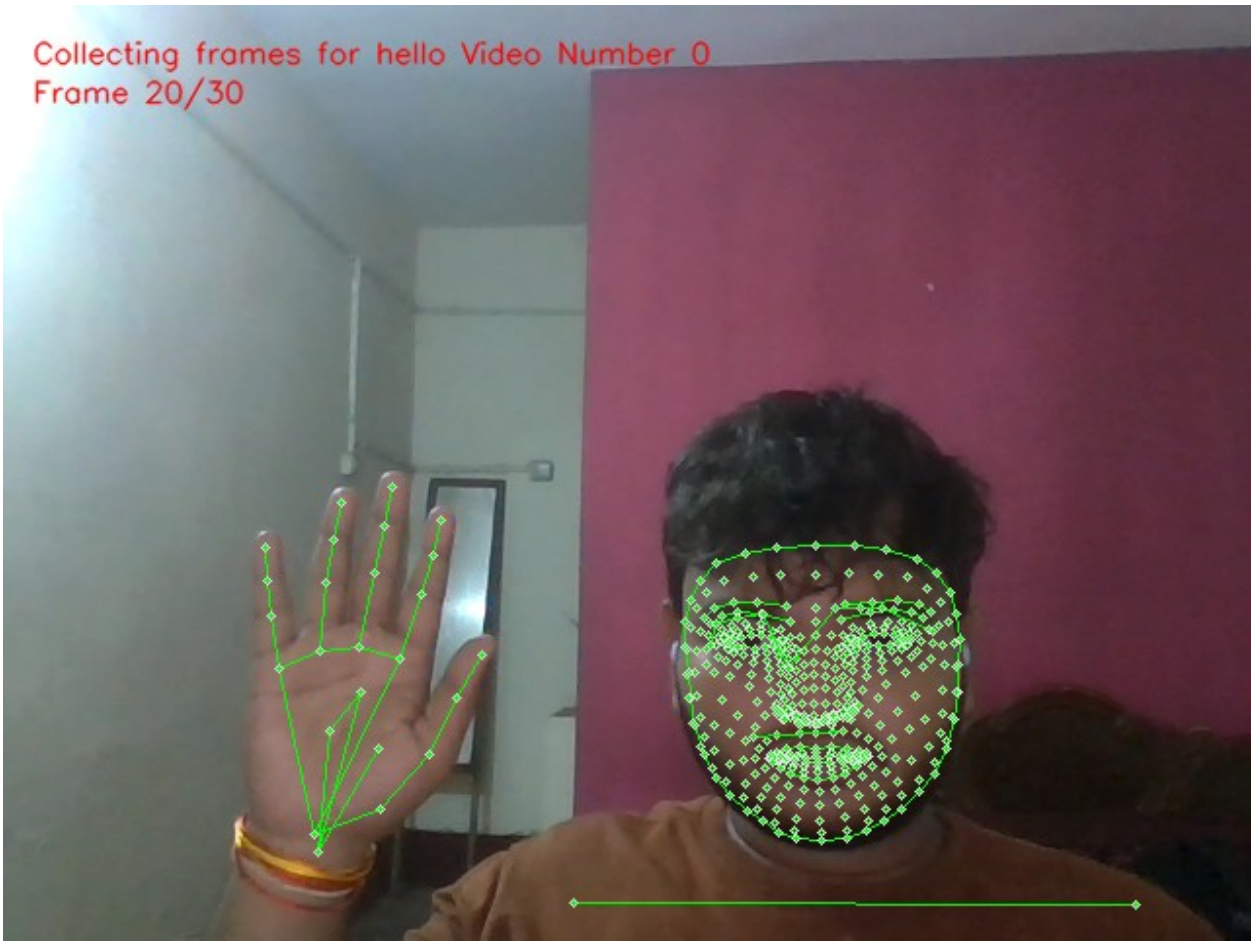

```
<IPython.core.display.Javascript object>
```



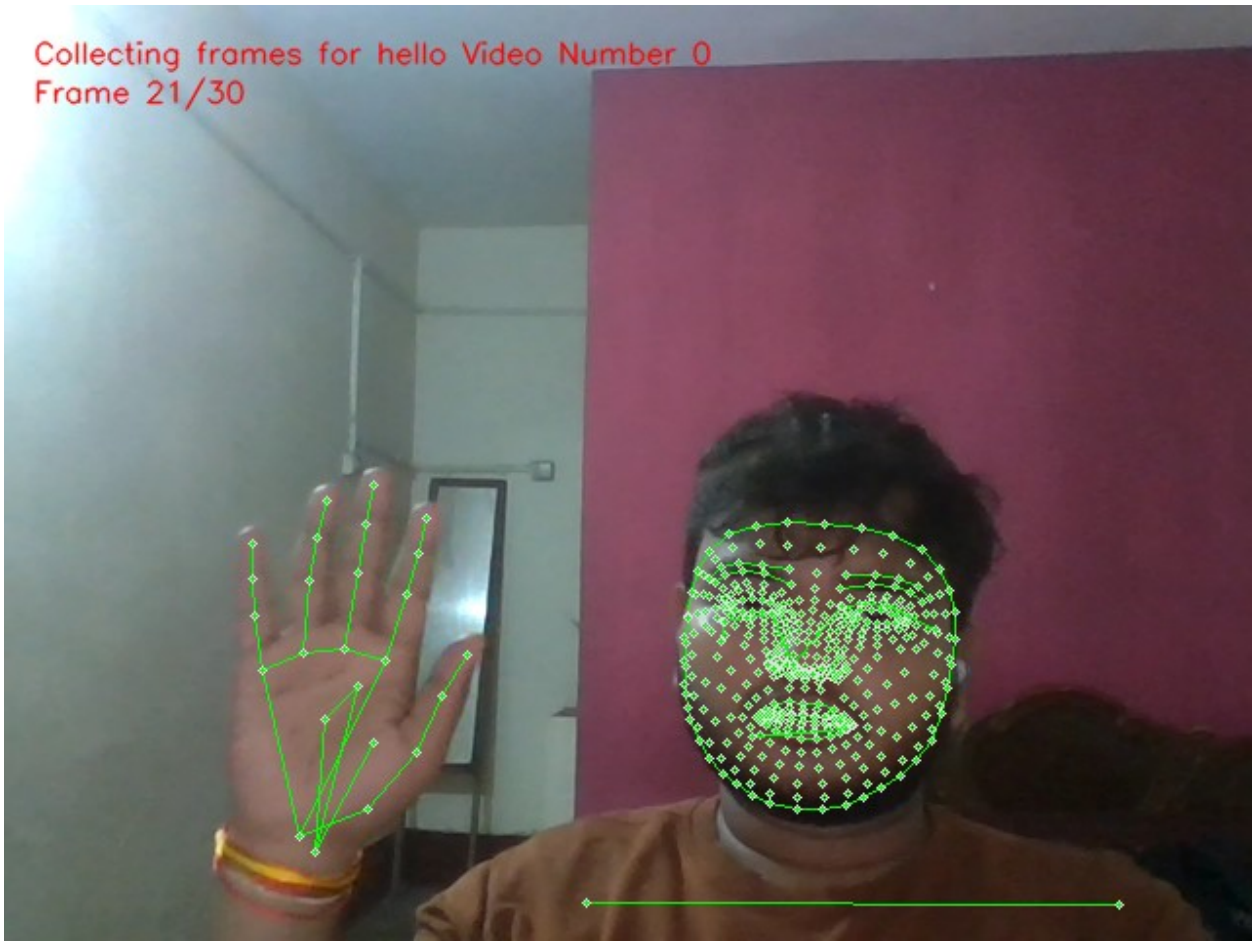

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

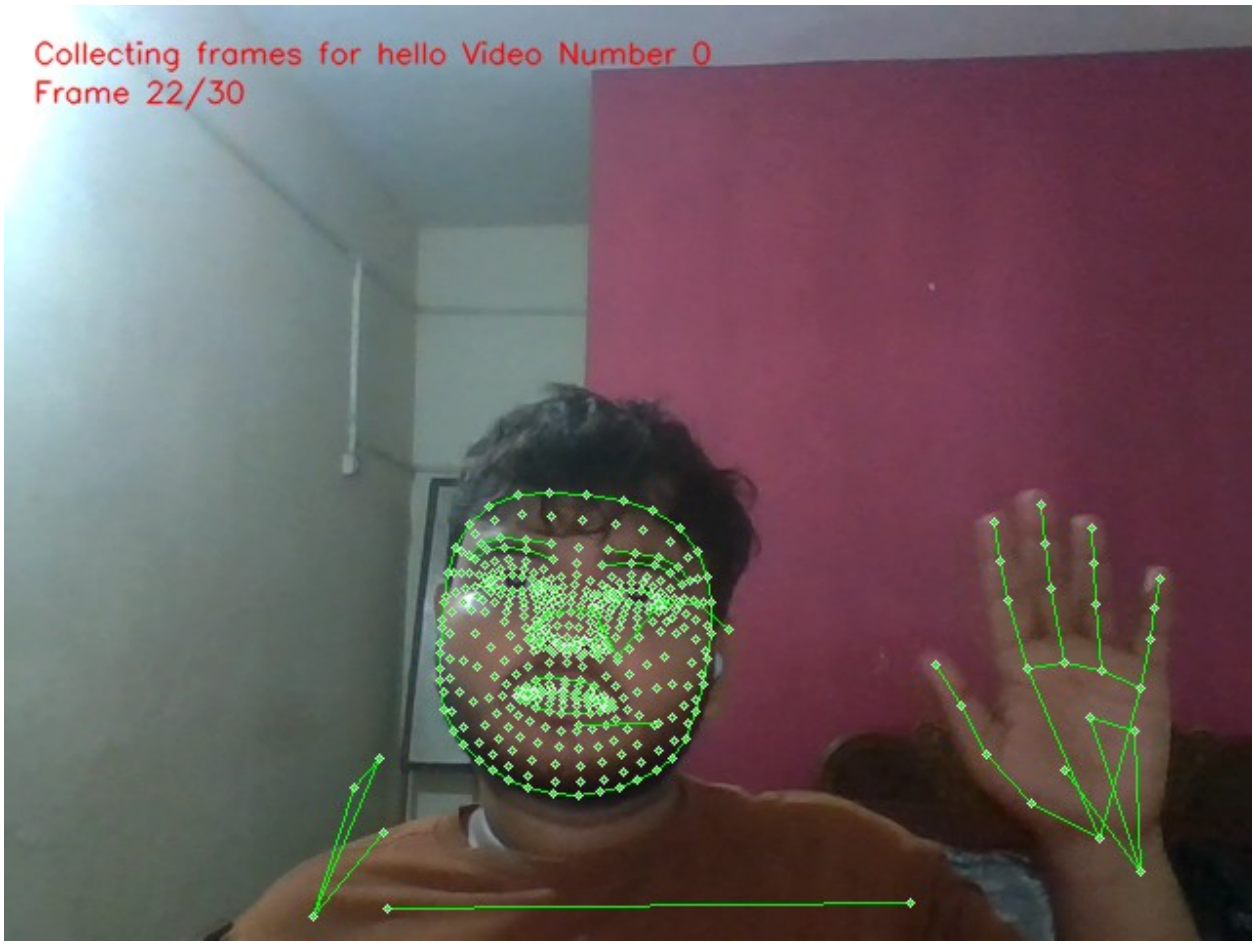

```
<IPython.core.display.Javascript object>
```



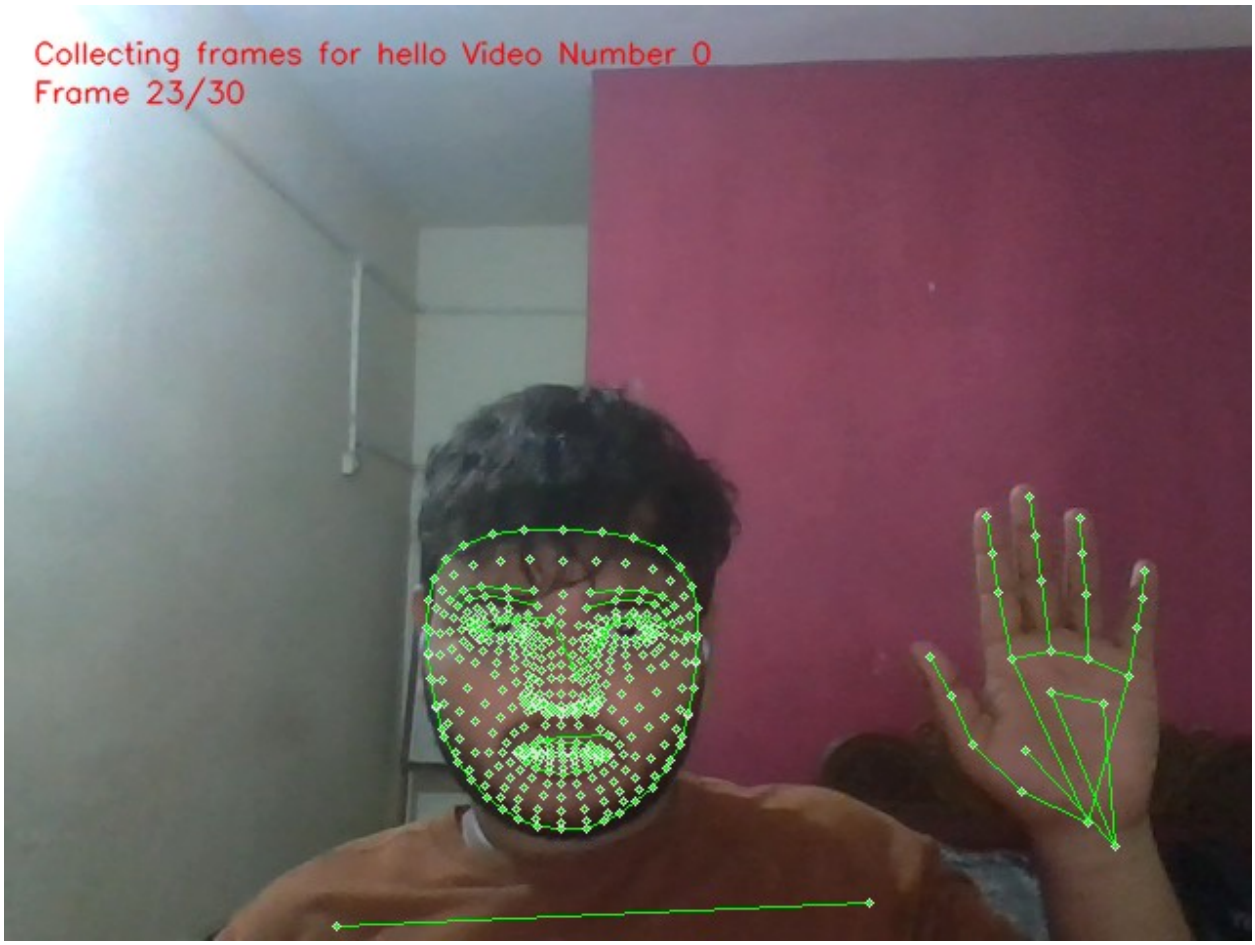

```
<IPython.core.display.Javascript object>
```



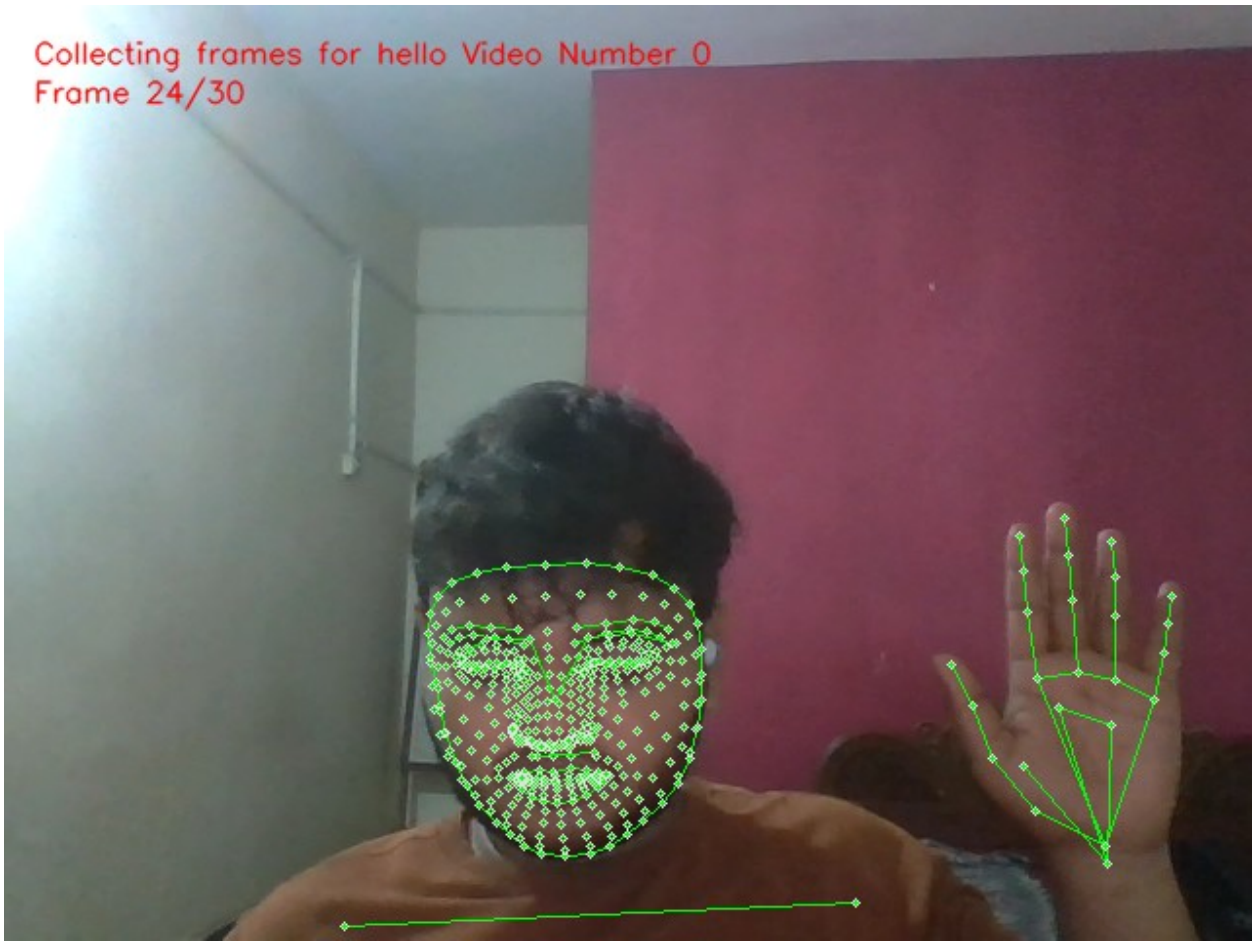

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

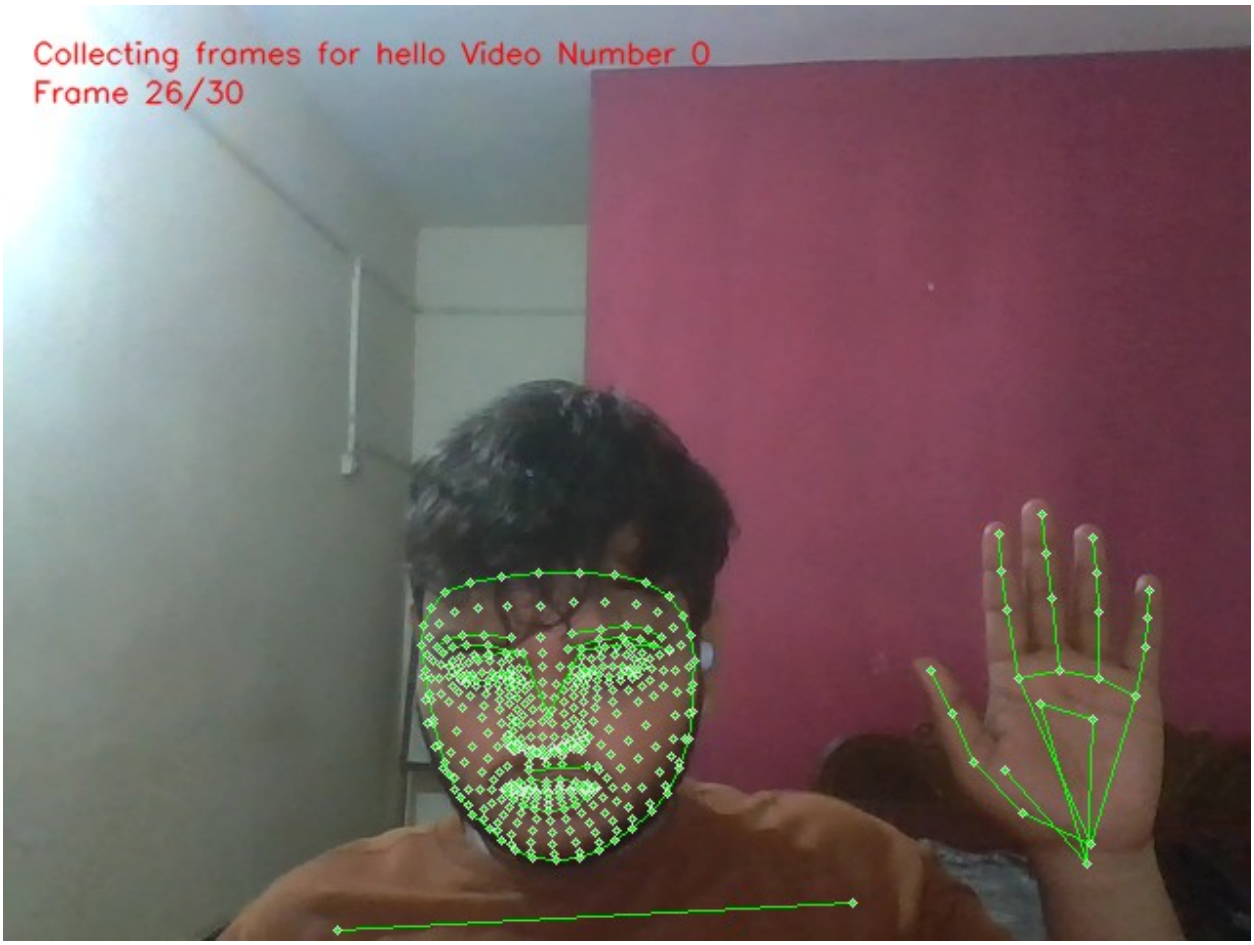



```
<IPython.core.display.Javascript object>
```



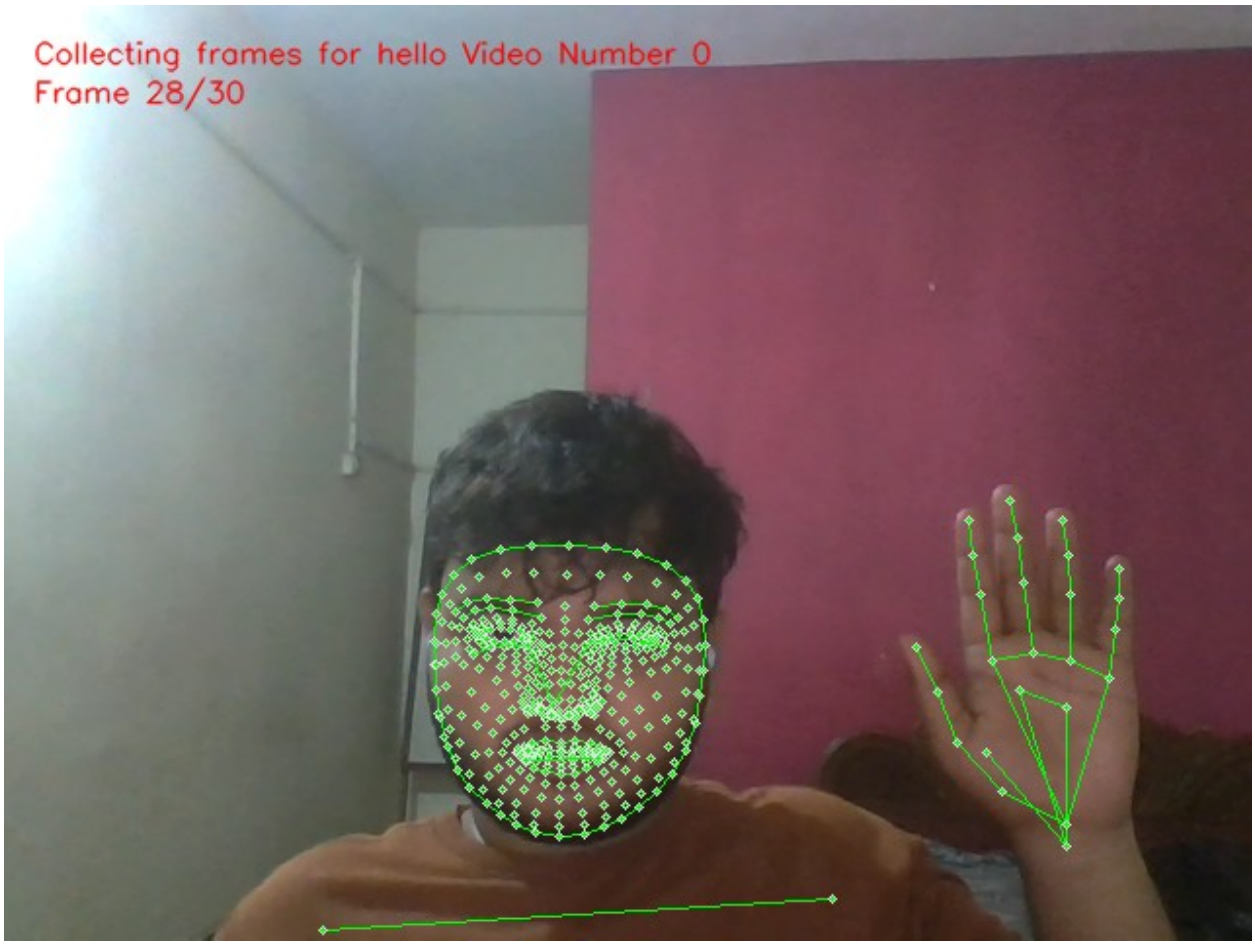

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

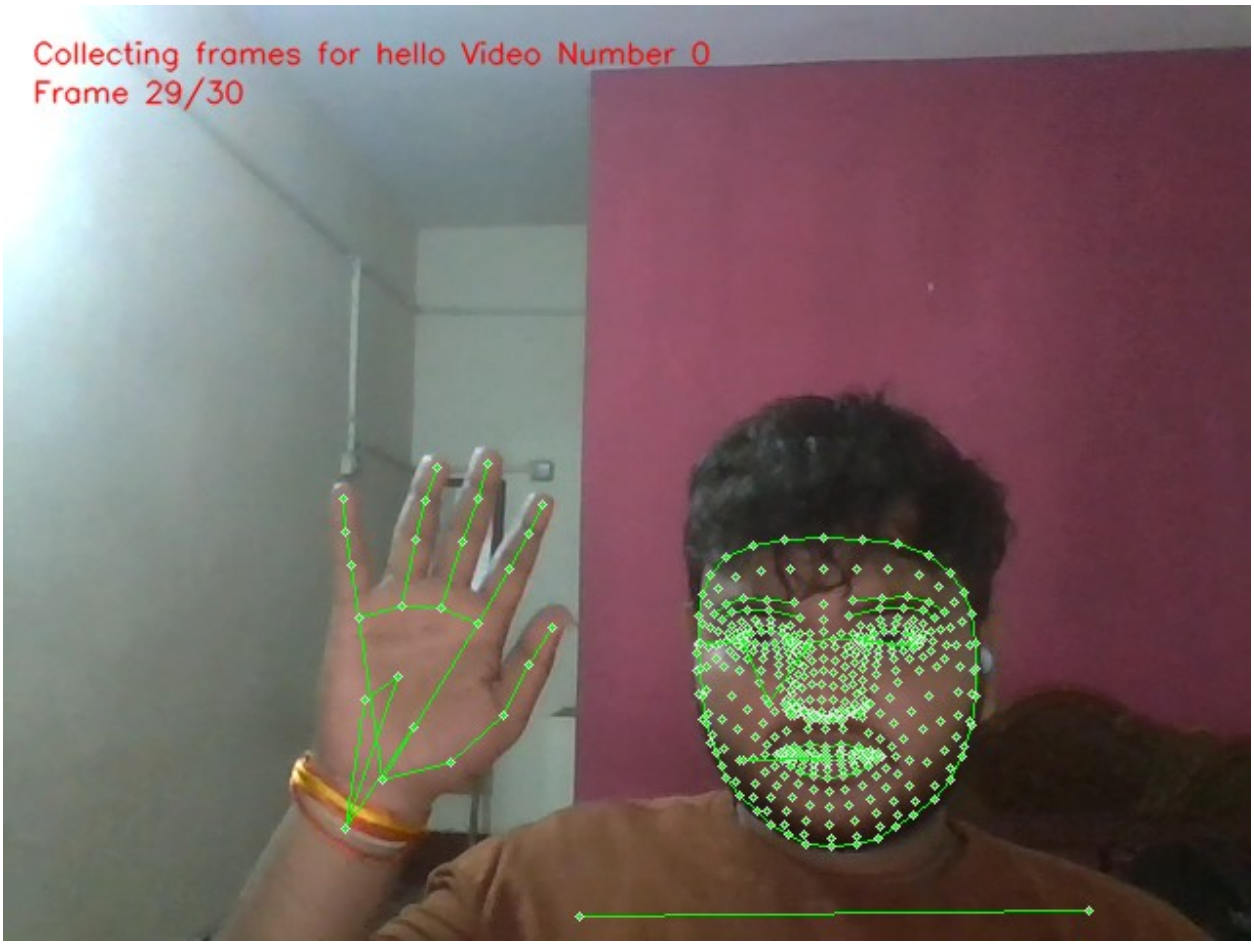



```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

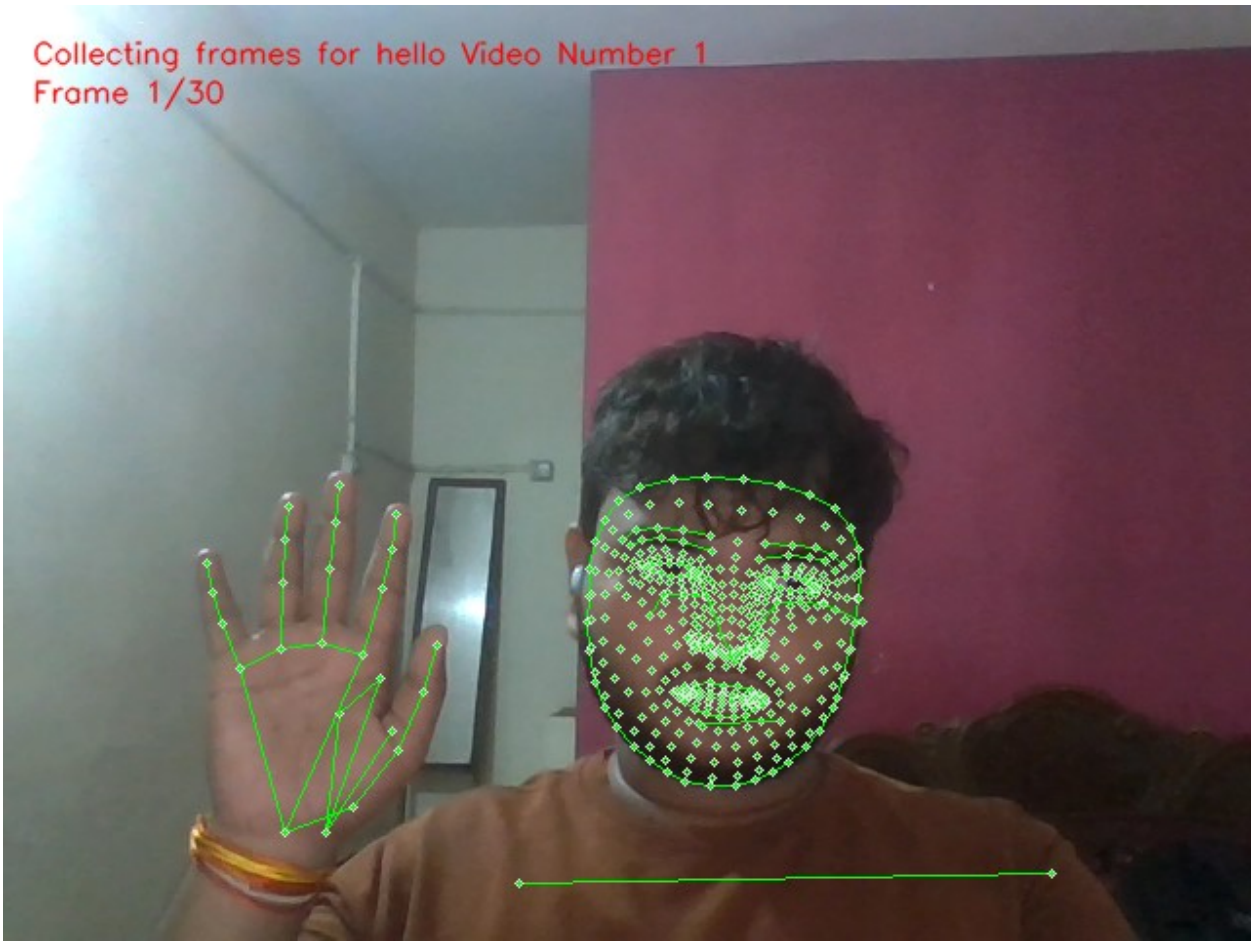



```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-18-058ccae7f569> in <cell line: 109>()
    116             for frame_num in range(sequence_length):
    117                 # Capture frame from webcam
--> 118                 data_url = video_stream()
    119                 frame = js_to_image(data_url)
    120

<ipython-input-18-058ccae7f569> in video_stream()
     45     ''')
     46     display(js)
---> 47     data = eval_js('stream()')
     48     return data
     49

/usr/local/lib/python3.10/dist-packages/google/colab/output/_js.py in
eval_js(script, ignore_result, timeout_sec)
```

```
     38   if ignore_result:
     39     return
---> 40   return _message.read_reply_from_input(request_id,
timeout_sec)
     41
     42

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
     94     reply = _read_next_input_message()
     95     if reply == _NOT_READY or not isinstance(reply, dict):
---> 96       time.sleep(0.025)
     97       continue
     98     if (

KeyboardInterrupt:
```

```
import shutil
from google.colab import files

# Define the directory to be zipped and the name of the output zip
file
directory_to_zip = '/content/MP_Data/hello'  # Example directory
output_filename = 'MP_Data_backup.zip'

# Zip the directory
shutil.make_archive(output_filename.replace('.zip', ''), 'zip',
directory_to_zip)

# Download the zip file
files.download(output_filename)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

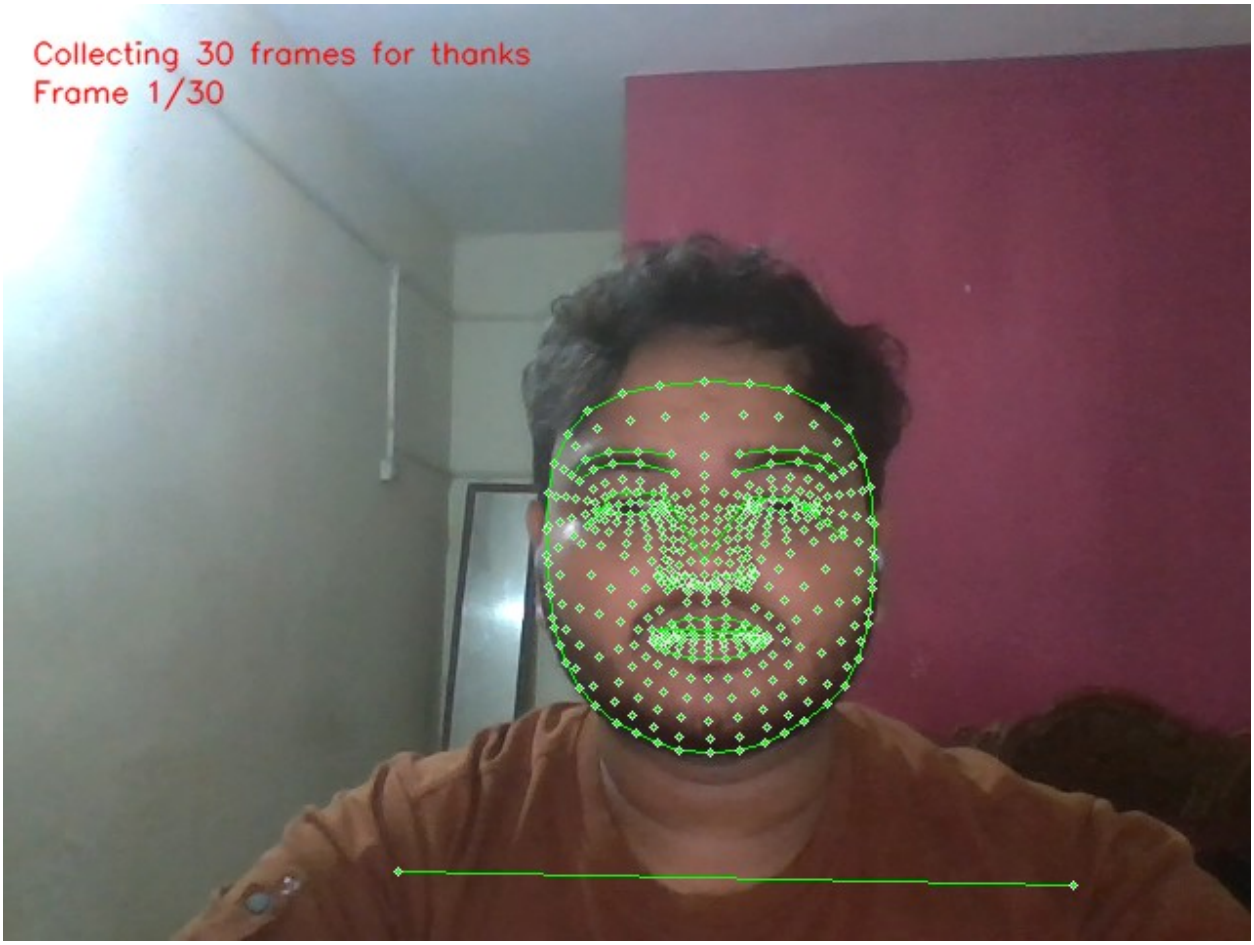
```
import cv2
import mediapipe as mp
from google.colab.patches import cv2_imshow
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
import os
import time

# Function to convert JavaScript object to OpenCV image
def js_to_image(js_reply):
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
```

```
    return img

# Function to stream video from webcam
def video_stream():
    js = Javascript('''
    async function stream() {
        const video = document.createElement('video');
        video.style.display = 'block';
        const stream = await
navigator.mediaDevices.getUserMedia({video: true});
        document.body.appendChild(video);
        video.srcObject = stream;
        await video.play();
        // Resize output to fit the screen.

google.colab.output.setIframeHeight(document.documentElement.scrollHei
ght, true);

        // Create a canvas element.
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');

        // Draw the video frame to the canvas.
        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        // Convert the canvas image to a base64 string.
        const image = canvas.toDataURL('image/png');
        stream.getTracks()[0].stop();
        return image;
    }
    ''')
    display(js)
    data = eval_js('stream()')
    return data

# Function to perform landmark detection using Mediapipe
def mediapipe_detection(image, model):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_rgb.flags.writeable = False
    results = model.process(image_rgb)
    image_rgb.flags.writeable = True
    image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
    return image, results

# Function to draw landmarks on the image
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
```

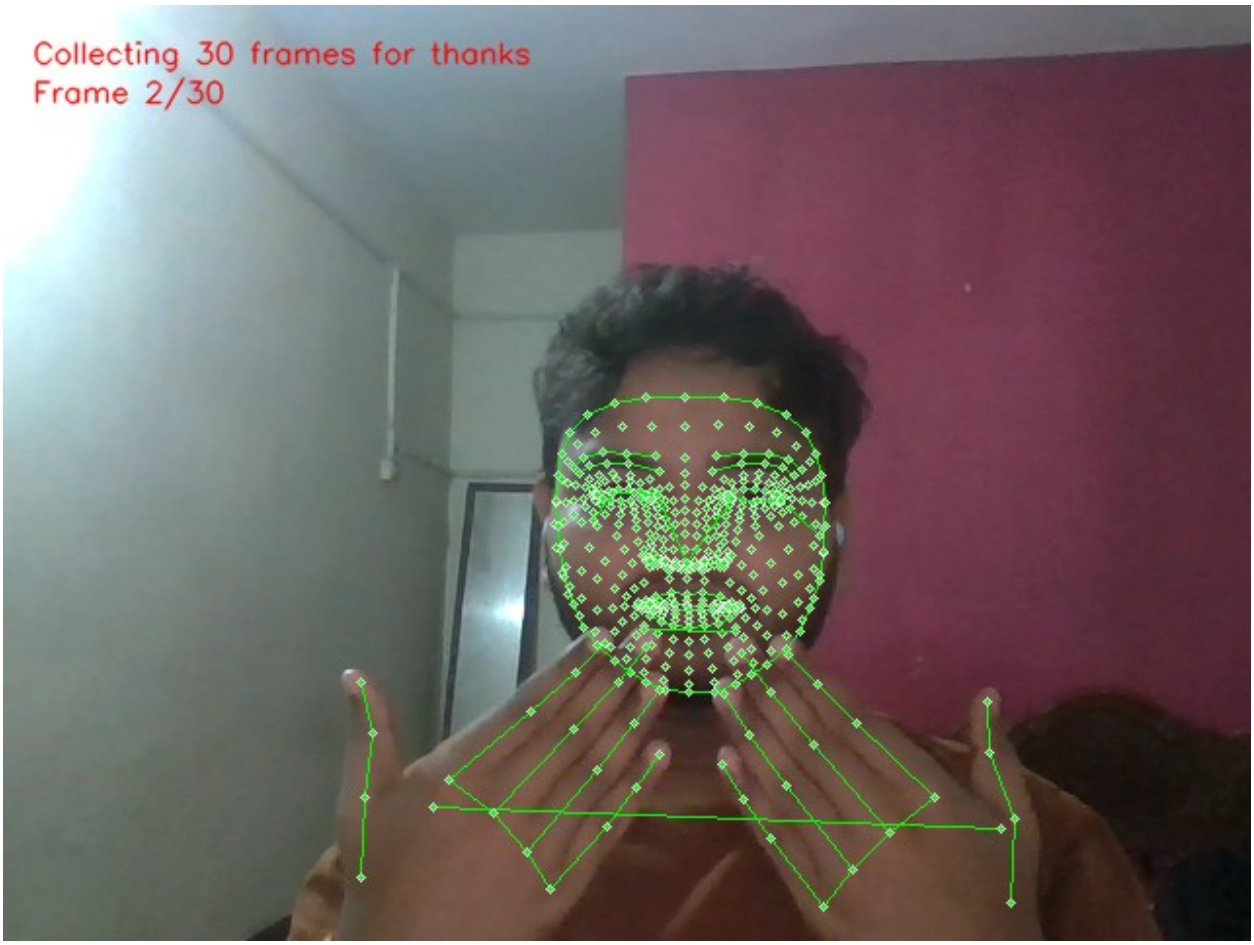
```
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.array(['thanks', 'iloveyou'])

# Number of frames to capture per action
frames_per_action = 30

# Ensure directories exist
for action in actions:
    try:
        os.makedirs(os.path.join(DATA_PATH, action))
    except:
        pass

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks
else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
```

```
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

# Initialize Mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    for action in actions:
        print(f"Collecting {frames_per_action} frames for action:
{action}")
        for frame_num in range(frames_per_action):
            # Capture frame from webcam
            data_url = video_stream()
            frame = js_to_image(data_url)

            # Check if frame is captured successfully
            if frame is None:
                print(f"Error: Frame {frame_num} not captured.")
                continue

            # Make detections
            image, results = mediapipe_detection(frame, holistic)

            # Draw landmarks
            draw_styled_landmarks(image, results)

            # Display frame info
            cv2.putText(image, f'Collecting {frames_per_action} frames
for {action}', (15, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
            cv2.putText(image, f'Frame {frame_num +
1}/{frames_per_action}', (15, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
            cv2_imshow(image)

            # Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(DATA_PATH, action,
f'{action}_{frame_num}')
            np.save(npy_path, keypoints)

            # Reduce delay between frames
            time.sleep(0.05)  # Adjust this value if needed

print("Data collection complete.")
```

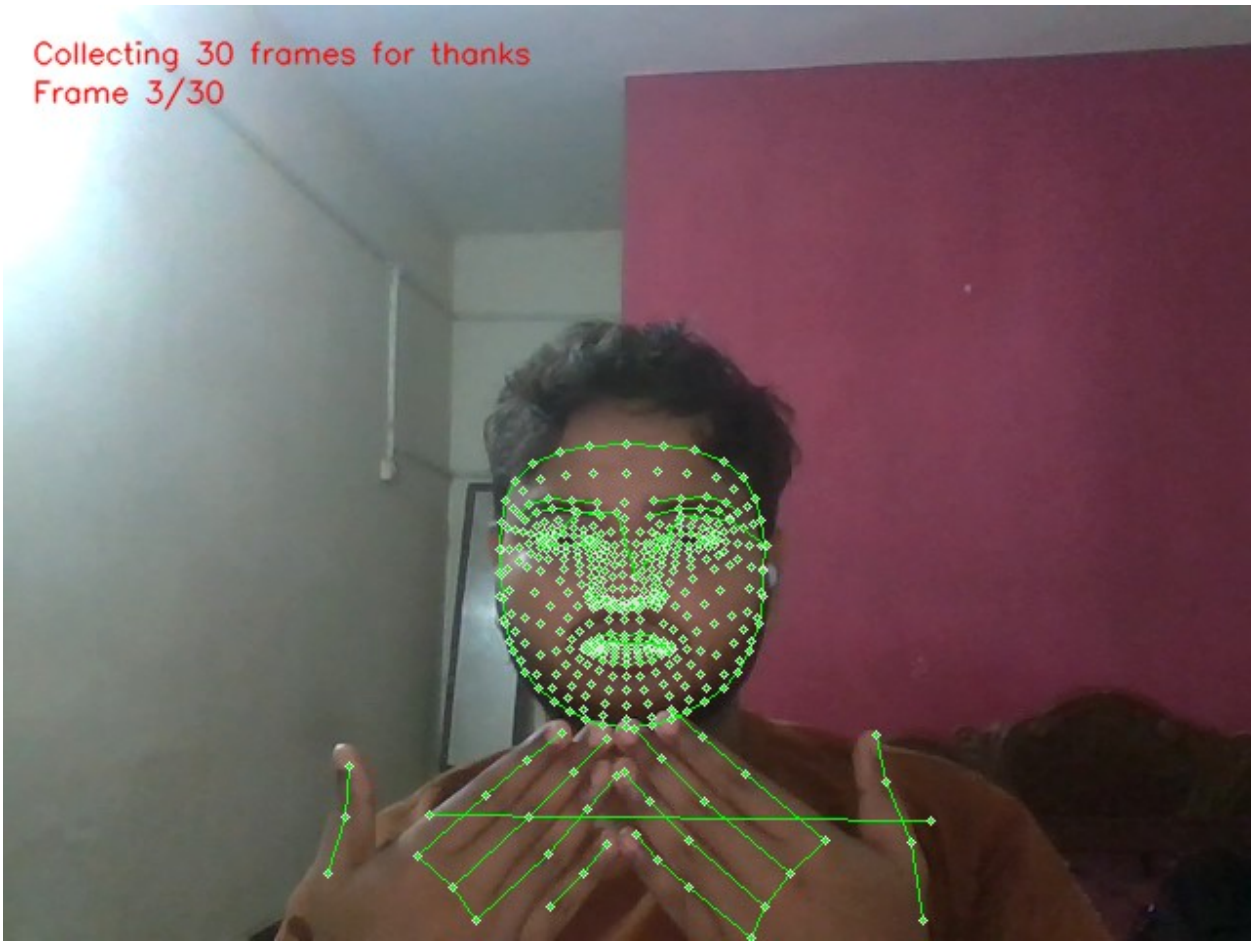
```
Collecting 30 frames for action: thanks
<IPython.core.display.Javascript object>
```



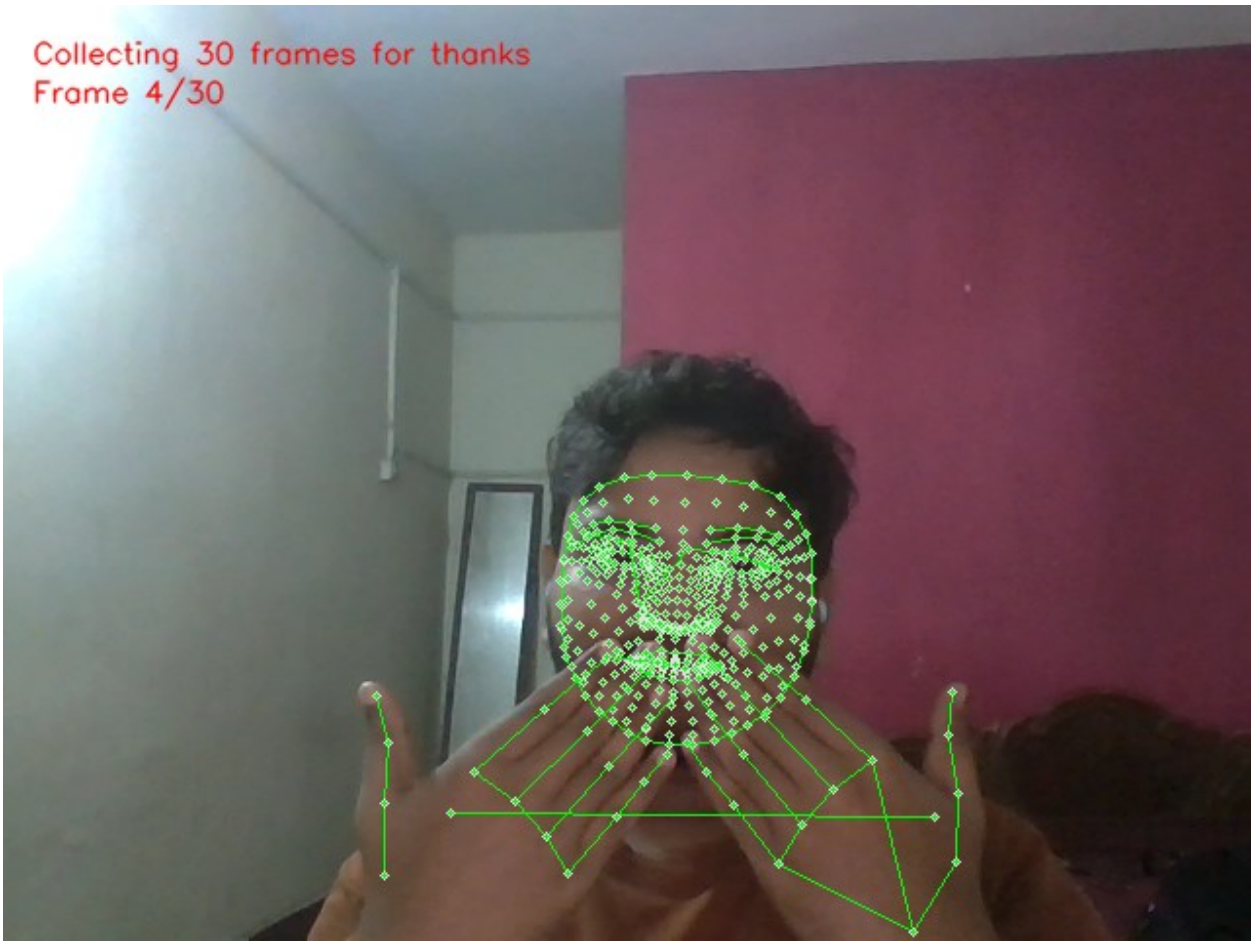

```
<IPython.core.display.Javascript object>
```



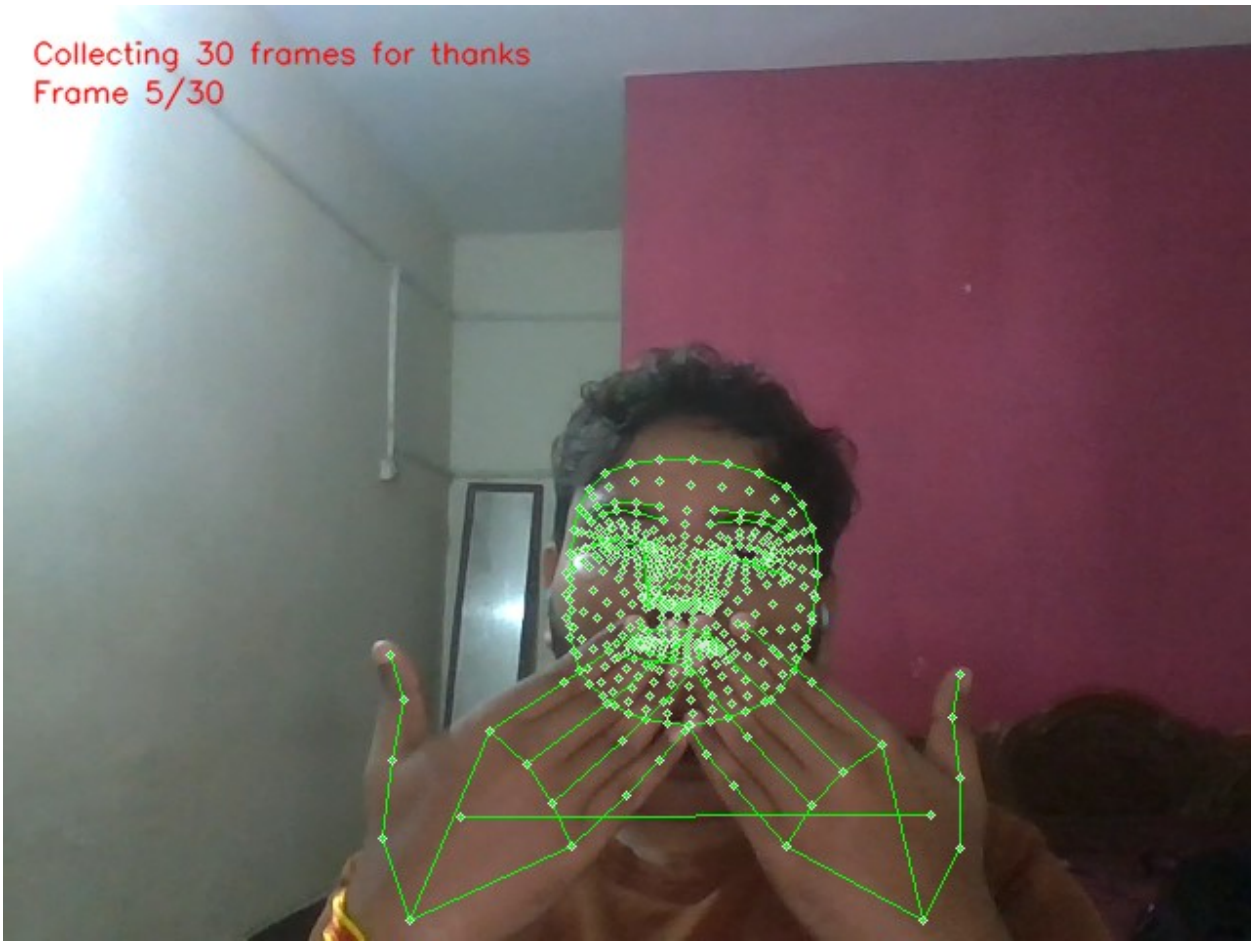

```
<IPython.core.display.Javascript object>
```



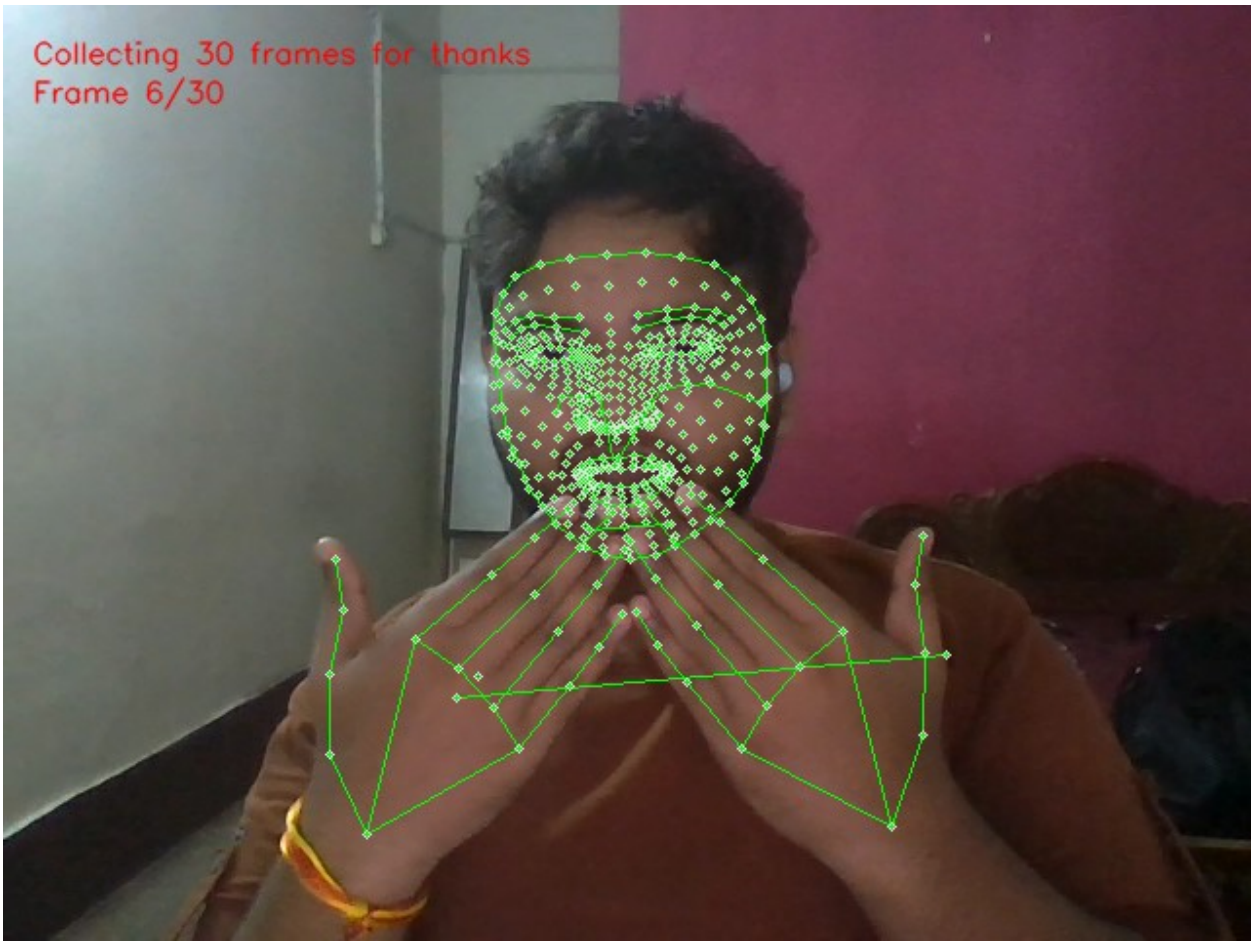

```
<IPython.core.display.Javascript object>
```



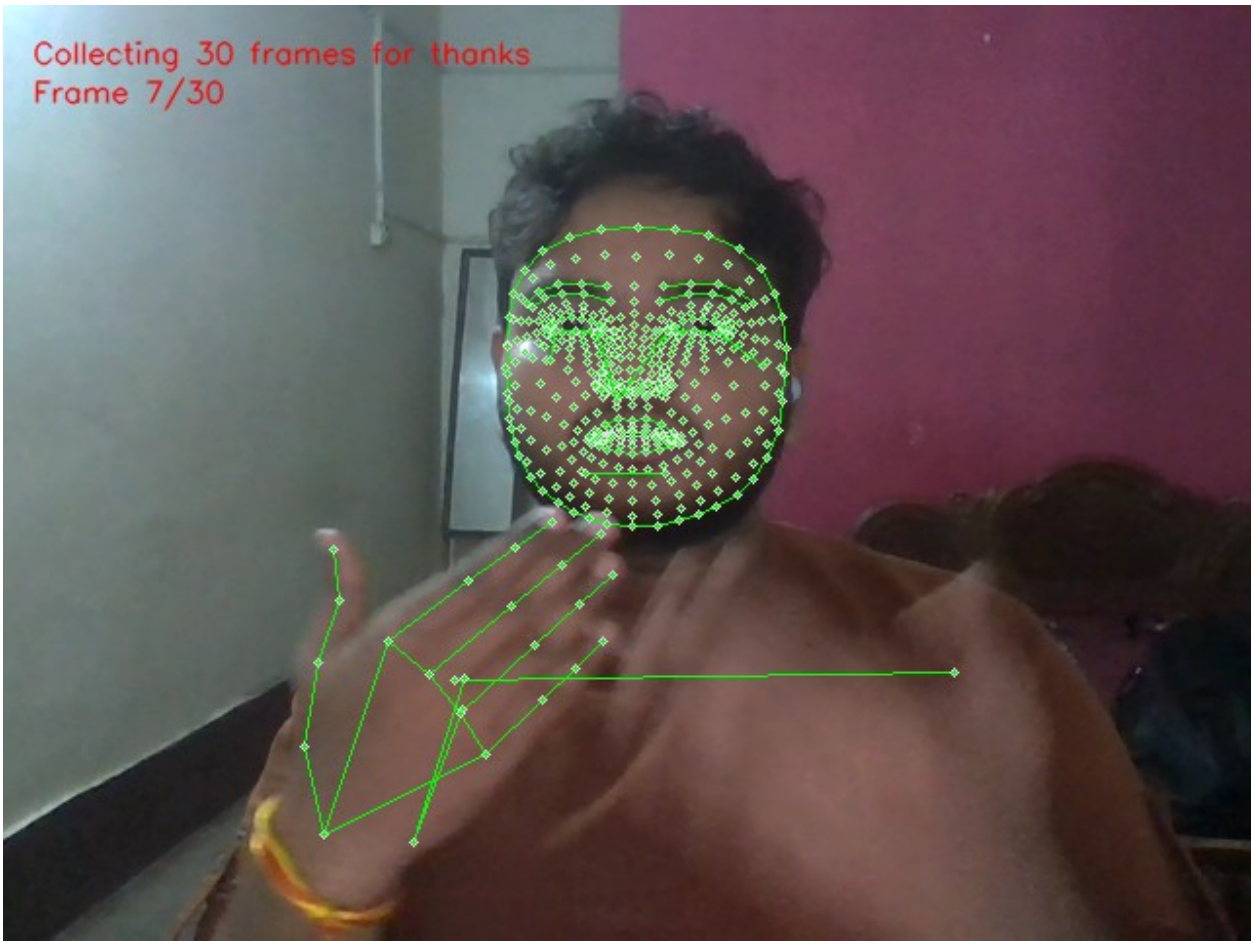

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

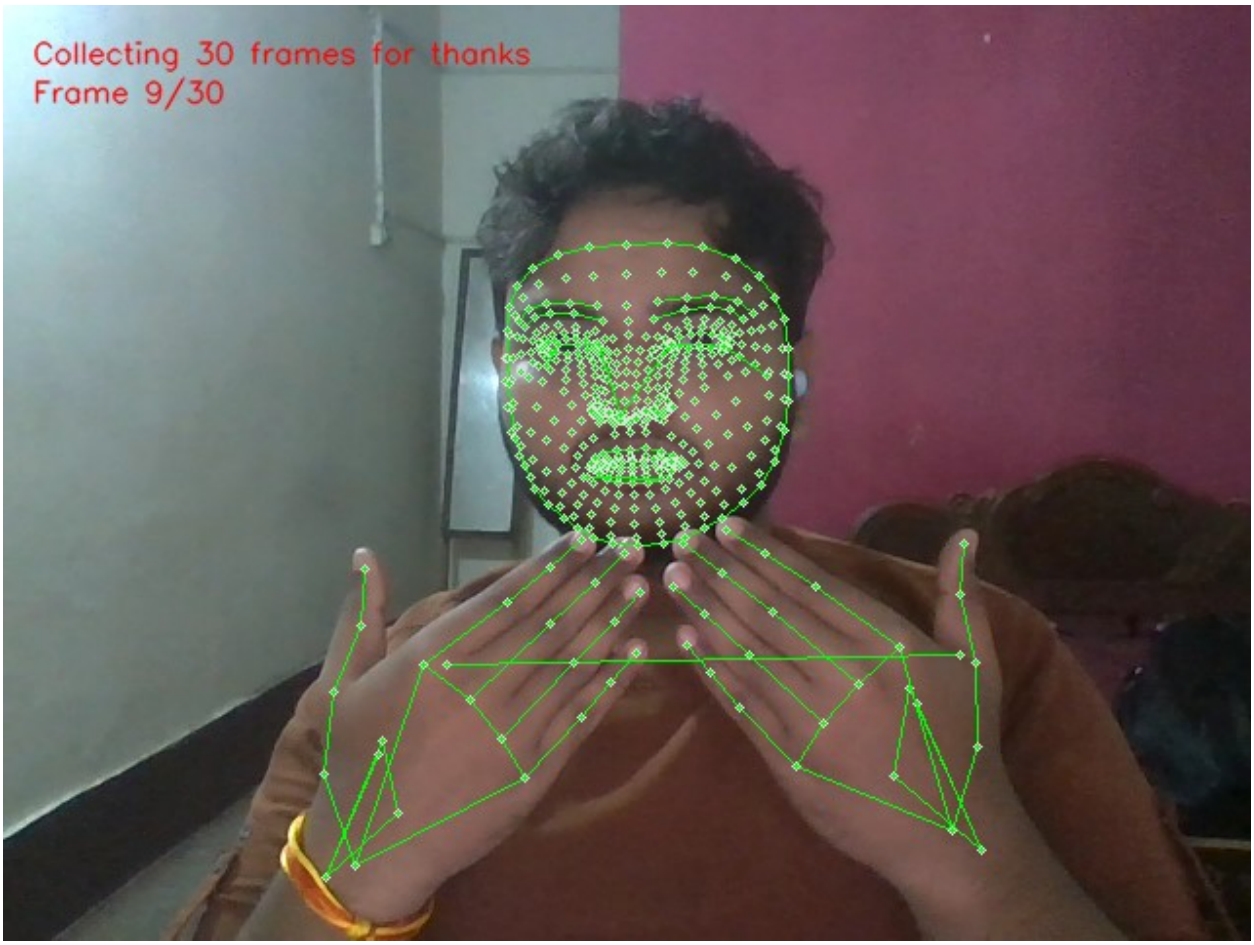



```
<IPython.core.display.Javascript object>
```

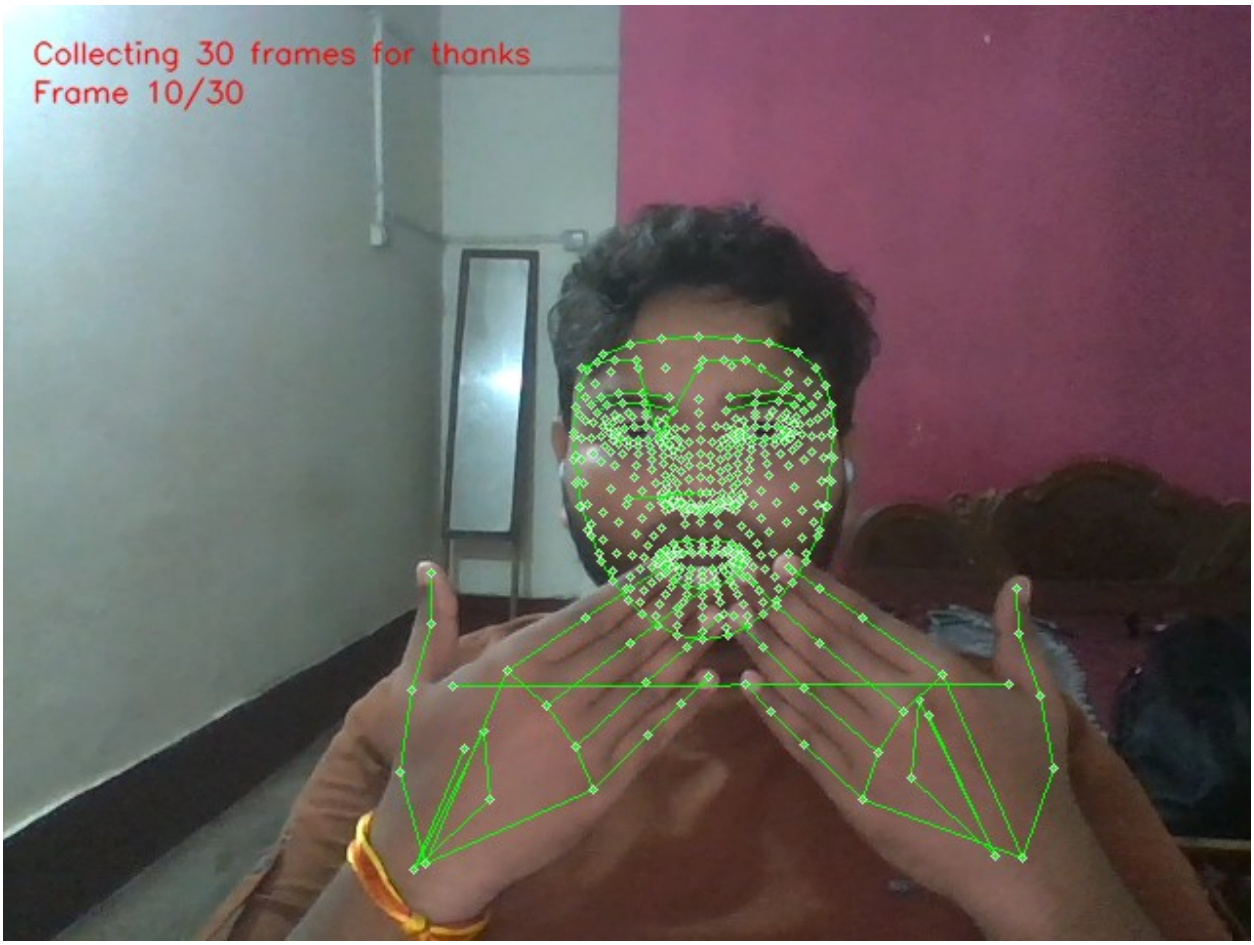



```
<IPython.core.display.Javascript object>
```

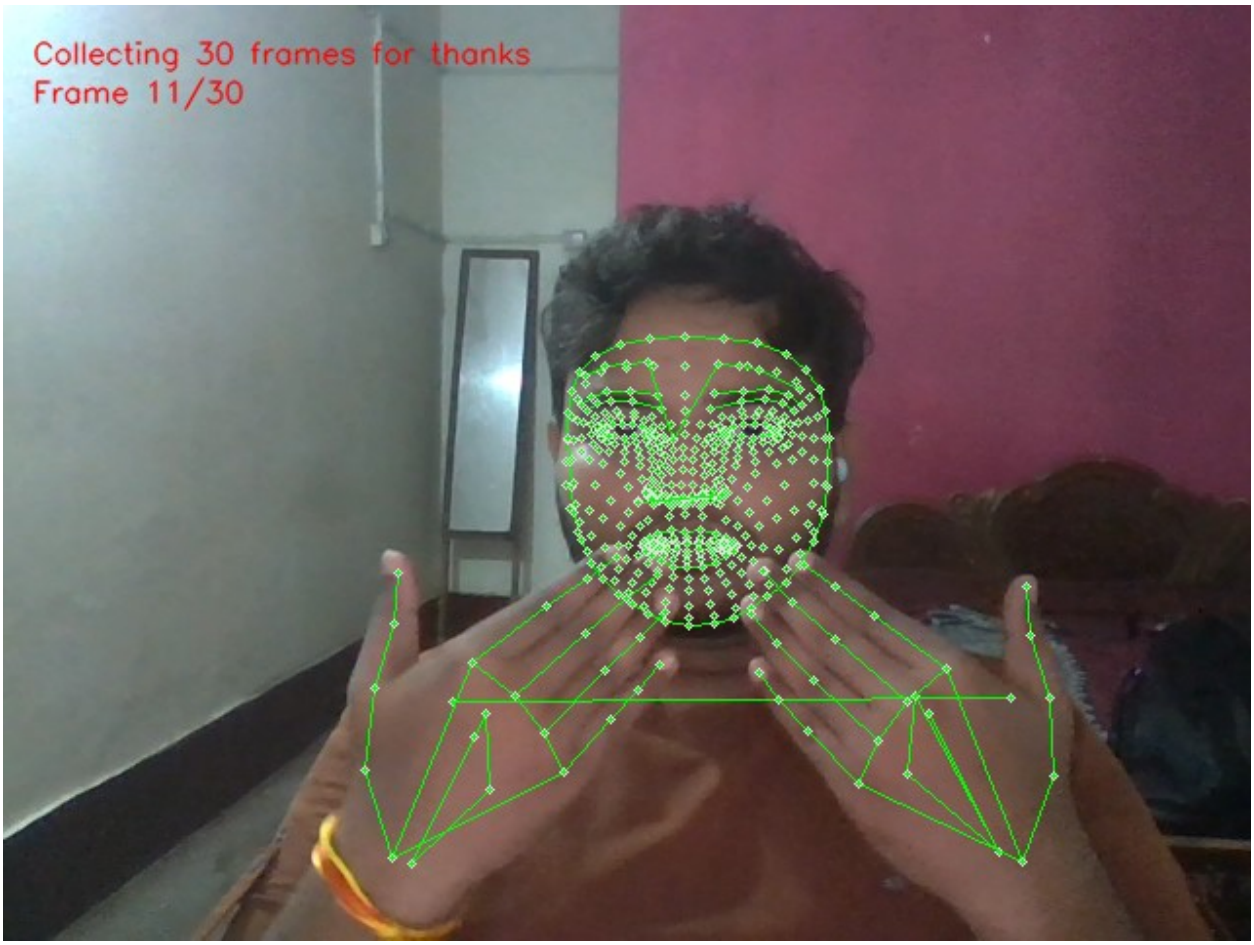

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-c5a40e7296dd> in <cell line: 102>()
    105         for frame_num in range(frames_per_action):
    106             # Capture frame from webcam
--> 107             data_url = video_stream()
    108             frame = js_to_image(data_url)
    109

<ipython-input-21-c5a40e7296dd> in video_stream()
     45     ''')
     46     display(js)
---> 47     data = eval_js('stream()')
     48     return data
     49

/usr/local/lib/python3.10/dist-packages/google/colab/output/_js.py in eval_js(script, ignore_result, timeout_sec)
```

```
     38   if ignore_result:
     39     return
---> 40   return _message.read_reply_from_input(request_id,
timeout_sec)
     41
     42

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
     94     reply = _read_next_input_message()
     95     if reply == _NOT_READY or not isinstance(reply, dict):
---> 96       time.sleep(0.025)
     97       continue
     98     if (

KeyboardInterrupt:
```

```
import shutil
from google.colab import files

# Define the directory to be zipped and the name of the output zip
file
directory_to_zip = '/content/MP_Data/'  # Example directory
output_filename = 'MP_Data_backup.zip'

# Zip the directory
shutil.make_archive(output_filename.replace('.zip', ''), 'zip',
directory_to_zip)

# Download the zip file
files.download(output_filename)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
import cv2
import mediapipe as mp
from google.colab.patches import cv2_imshow
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
import os
import time

# Function to convert JavaScript object to OpenCV image
def js_to_image(js_reply):
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
```

```
    return img

# Function to stream video from webcam
def video_stream():
    js = Javascript('''
    async function stream() {
        const video = document.createElement('video');
        video.style.display = 'block';
        const stream = await
navigator.mediaDevices.getUserMedia({video: true});
        document.body.appendChild(video);
        video.srcObject = stream;
        await video.play();
        // Resize output to fit the screen.

google.colab.output.setIframeHeight(document.documentElement.scrollHei
ght, true);

        // Create a canvas element.
        const canvas = document.createElement('canvas');
        canvas.width = video.videoWidth;
        canvas.height = video.videoHeight;
        const context = canvas.getContext('2d');

        // Draw the video frame to the canvas.
        context.drawImage(video, 0, 0, canvas.width, canvas.height);

        // Convert the canvas image to a base64 string.
        const image = canvas.toDataURL('image/png');
        stream.getTracks()[0].stop();
        return image;
    }
    ''')
    display(js)
    data = eval_js('stream()')
    return data

# Function to perform landmark detection using Mediapipe
def mediapipe_detection(image, model):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_rgb.flags.writeable = False
    results = model.process(image_rgb)
    image_rgb.flags.writeable = True
    image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
    return image, results

# Function to draw landmarks on the image
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
```

```
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                                landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Action to detect
action = 'iloveyou'

# Number of frames to capture
frames_per_action = 30

# Ensure directories exist
try:
    os.makedirs(os.path.join(DATA_PATH, action))
except:
    pass

def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks
else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in
results.face_landmarks.landmark]).flatten() if results.face_landmarks
else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in
results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
```

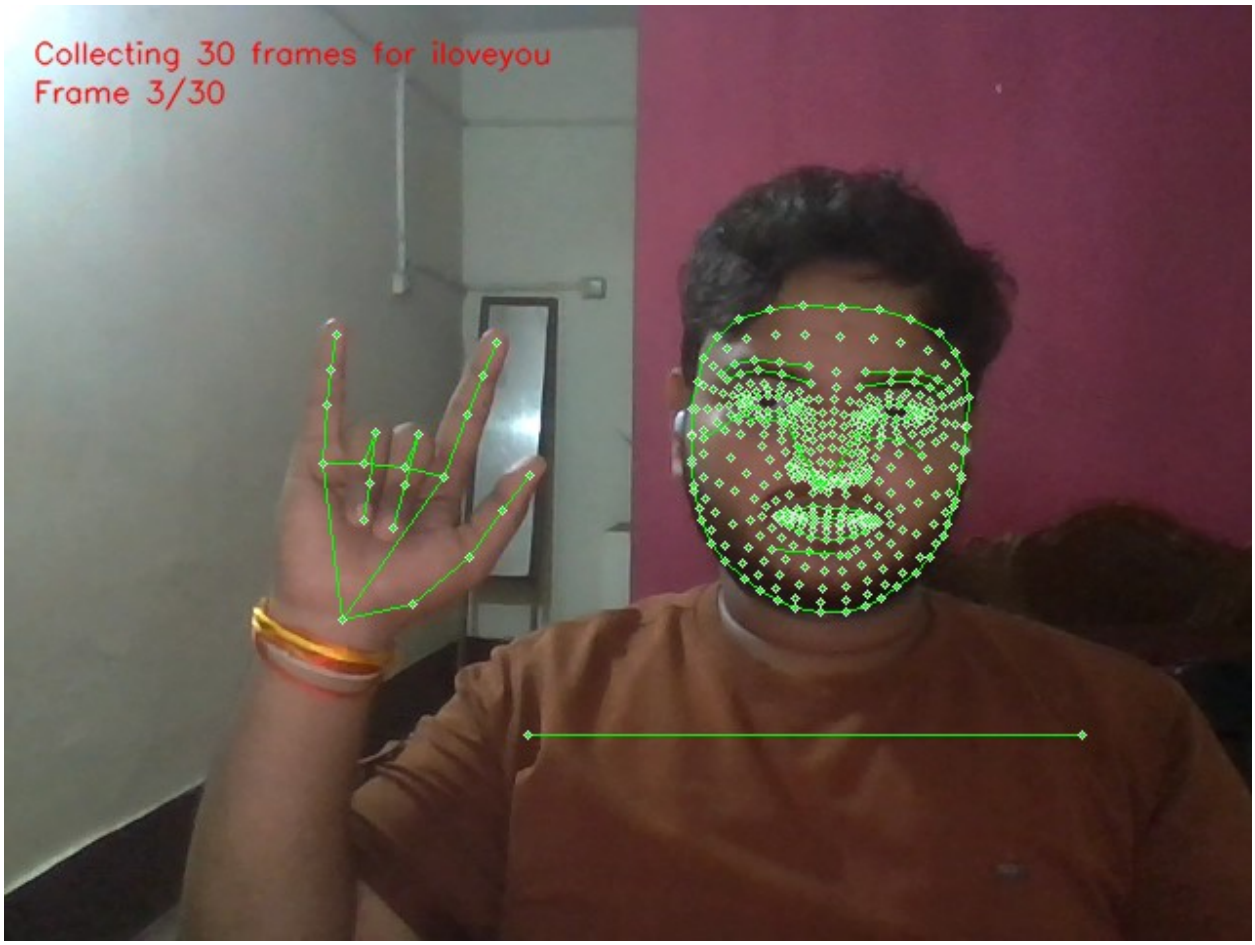
```
    rh = np.array([[res.x, res.y, res.z] for res in
results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])

# Initialize Mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    print(f"Collecting {frames_per_action} frames for action:
{action}")
    for frame_num in range(frames_per_action):
        # Capture frame from webcam
        data_url = video_stream()
        frame = js_to_image(data_url)

        # Check if frame is captured successfully
        if frame is None:
            print(f"Error: Frame {frame_num} not captured.")
            continue

        # Make detections
        image, results = mediapipe_detection(frame, holistic)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # Display frame info
        cv2.putText(image, f'Collecting {frames_per_action} frames for
{action}', (15, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
        cv2.putText(image, f'Frame {frame_num +
1}/{frames_per_action}', (15, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1,
cv2.LINE_AA)
        cv2_imshow(image)

        # Export keypoints
        keypoints = extract_keypoints(results)
        npy_path = os.path.join(DATA_PATH, action,
f'{action}_{frame_num}')
        np.save(npy_path, keypoints)

        # Reduce delay between frames
        time.sleep(0.05)  # Adjust this value if needed

print("Data collection complete.")
```

```
Collecting 30 frames for action: iloveyou
```

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```



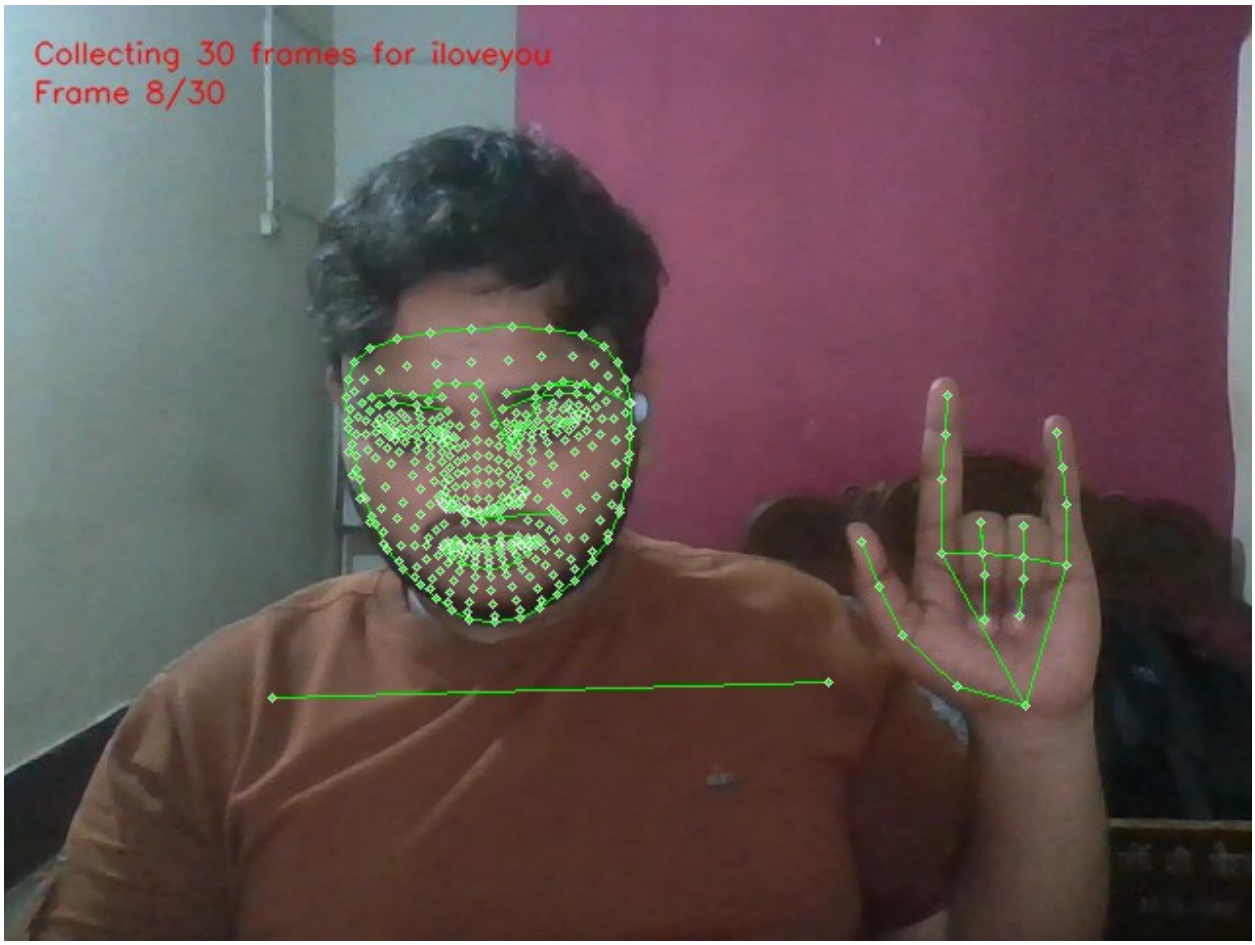

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

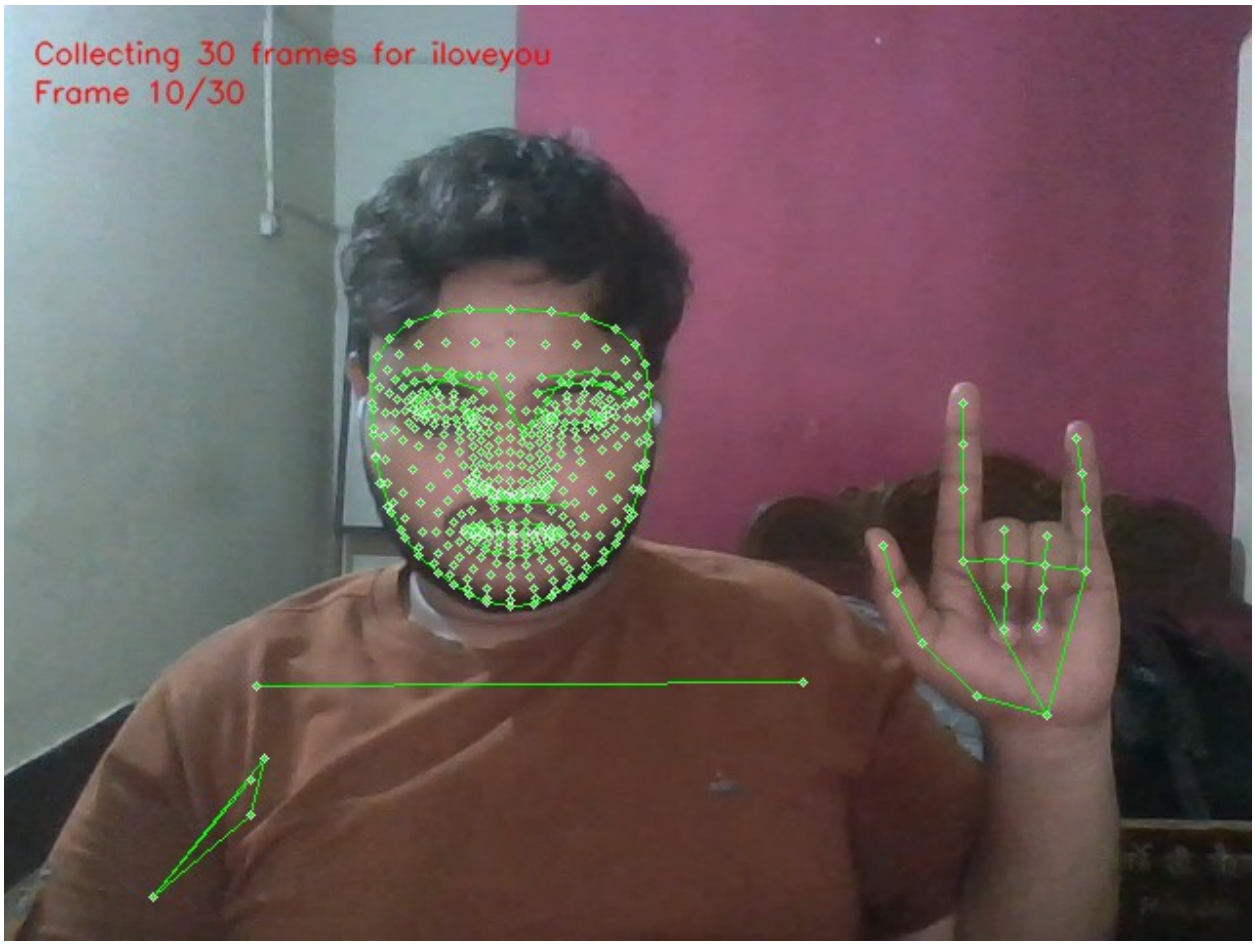

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

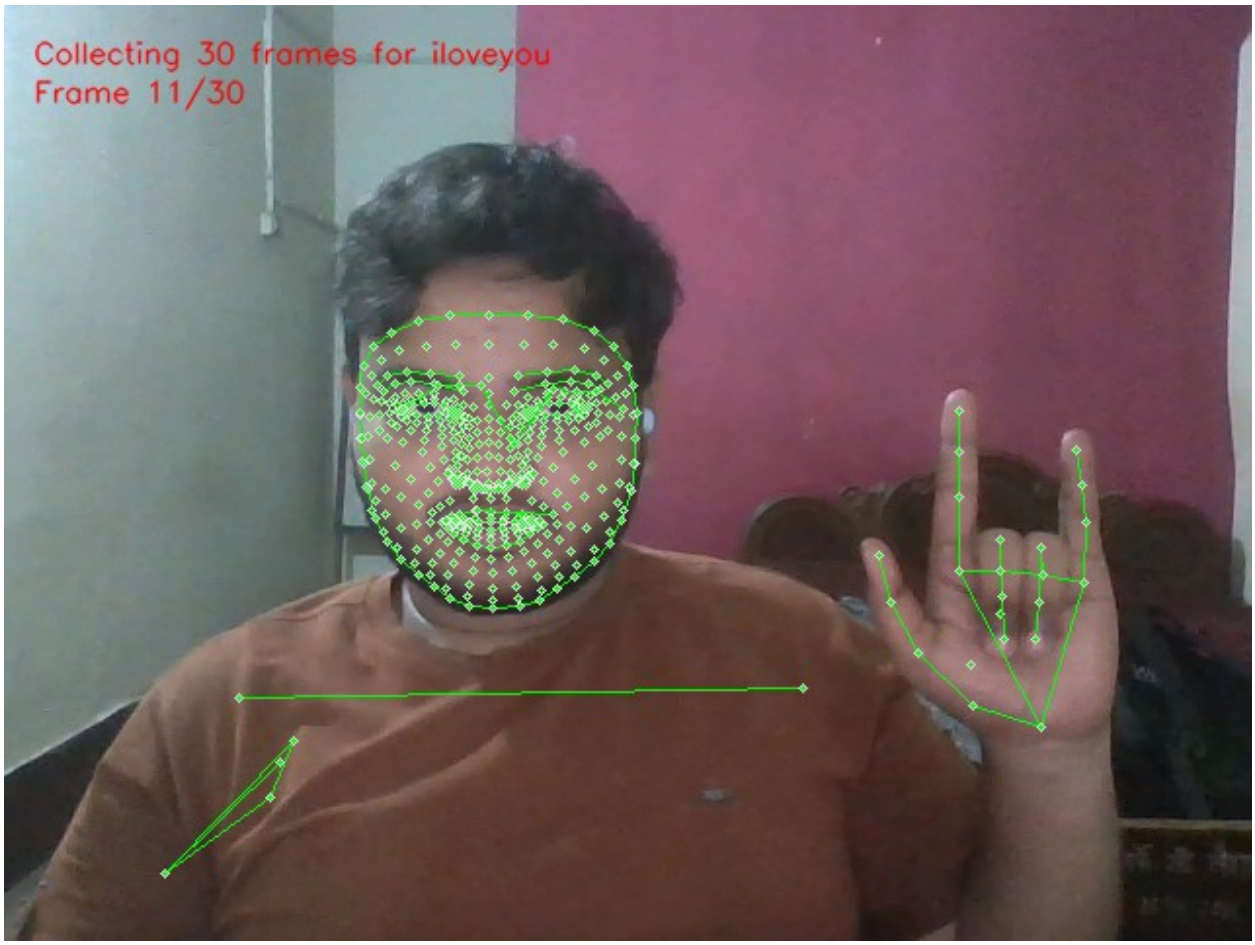



```
<IPython.core.display.Javascript object>
```

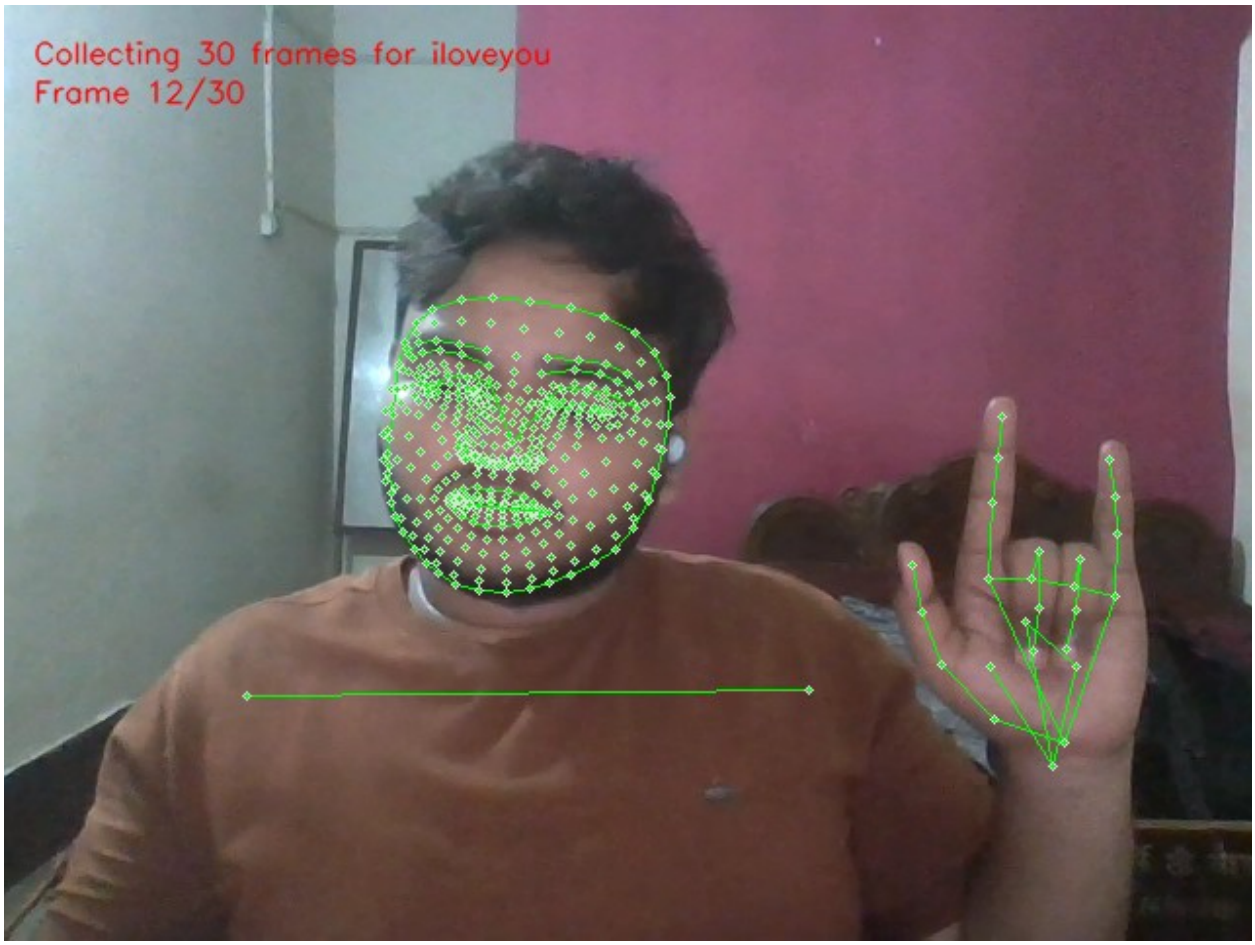

```
<IPython.core.display.Javascript object>
```



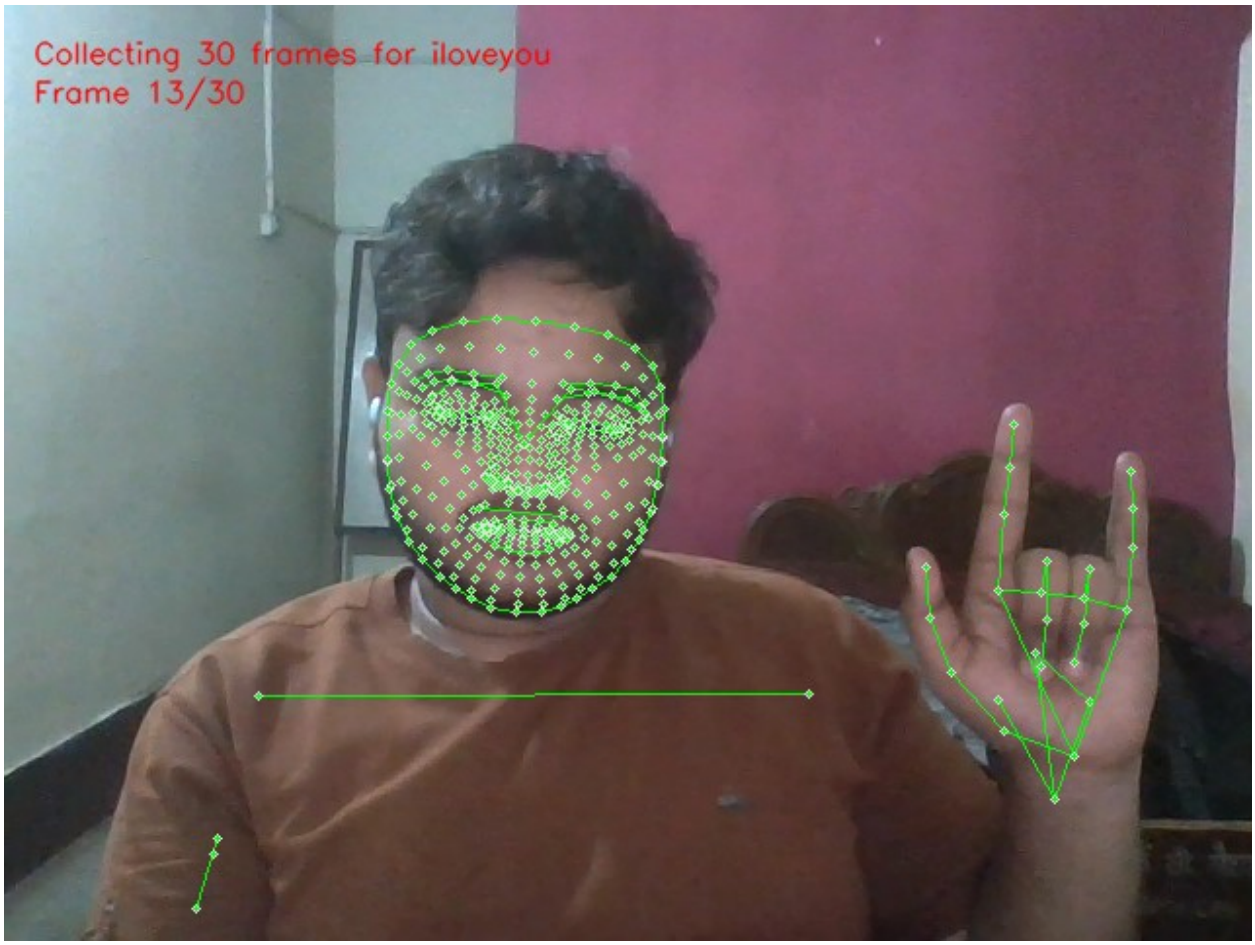

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```

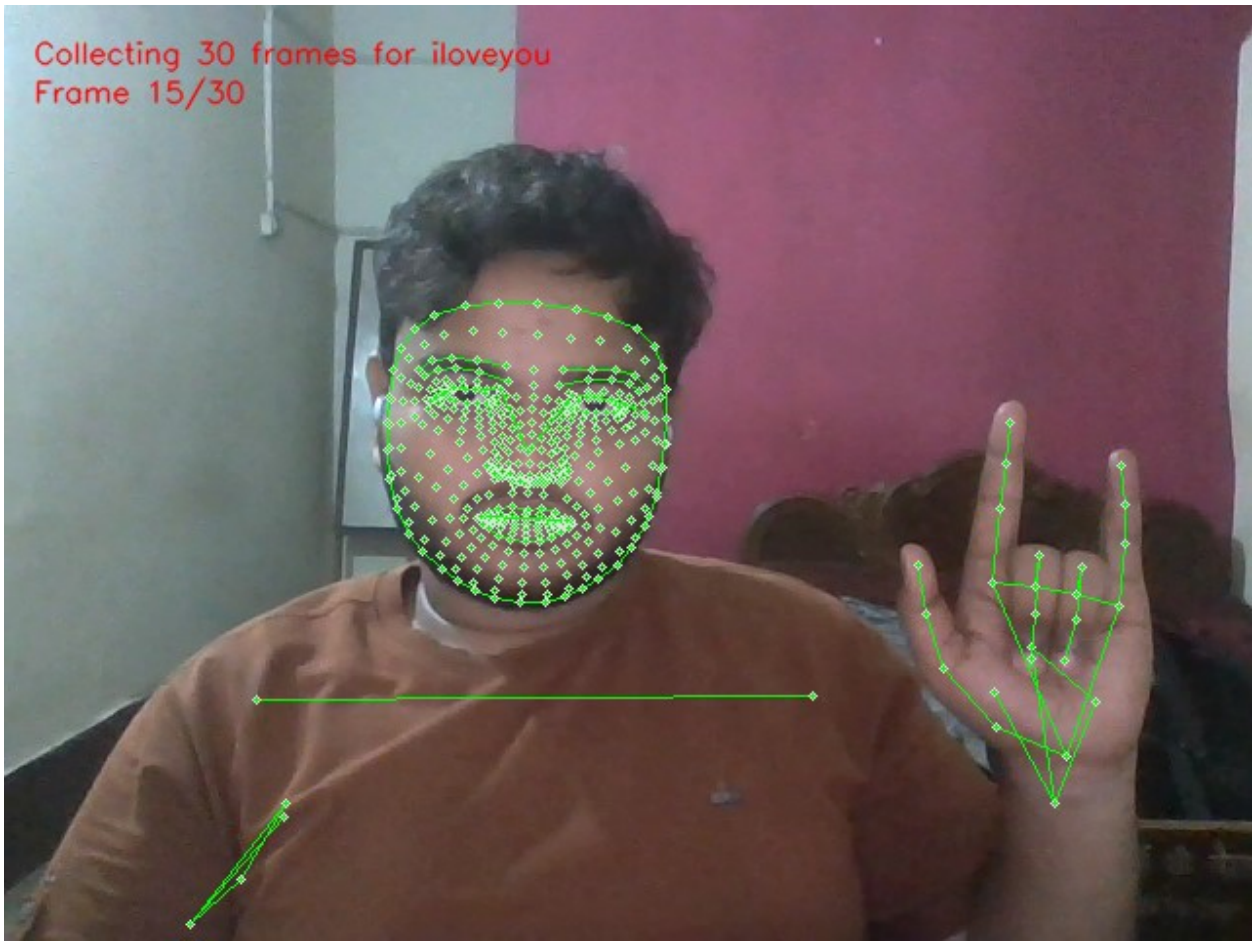



```
<IPython.core.display.Javascript object>
```



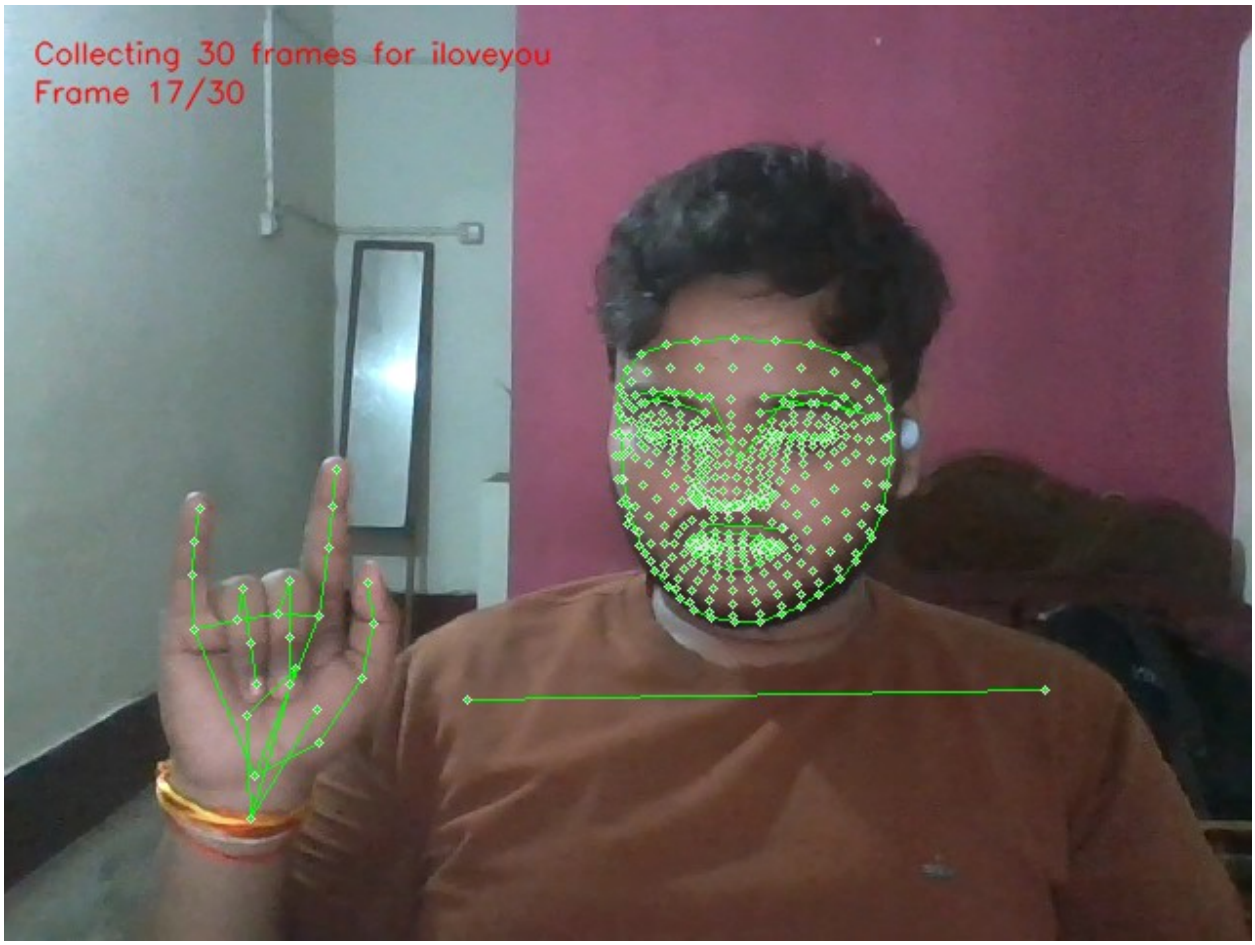

```
<IPython.core.display.Javascript object>
```


```
<IPython.core.display.Javascript object>
```

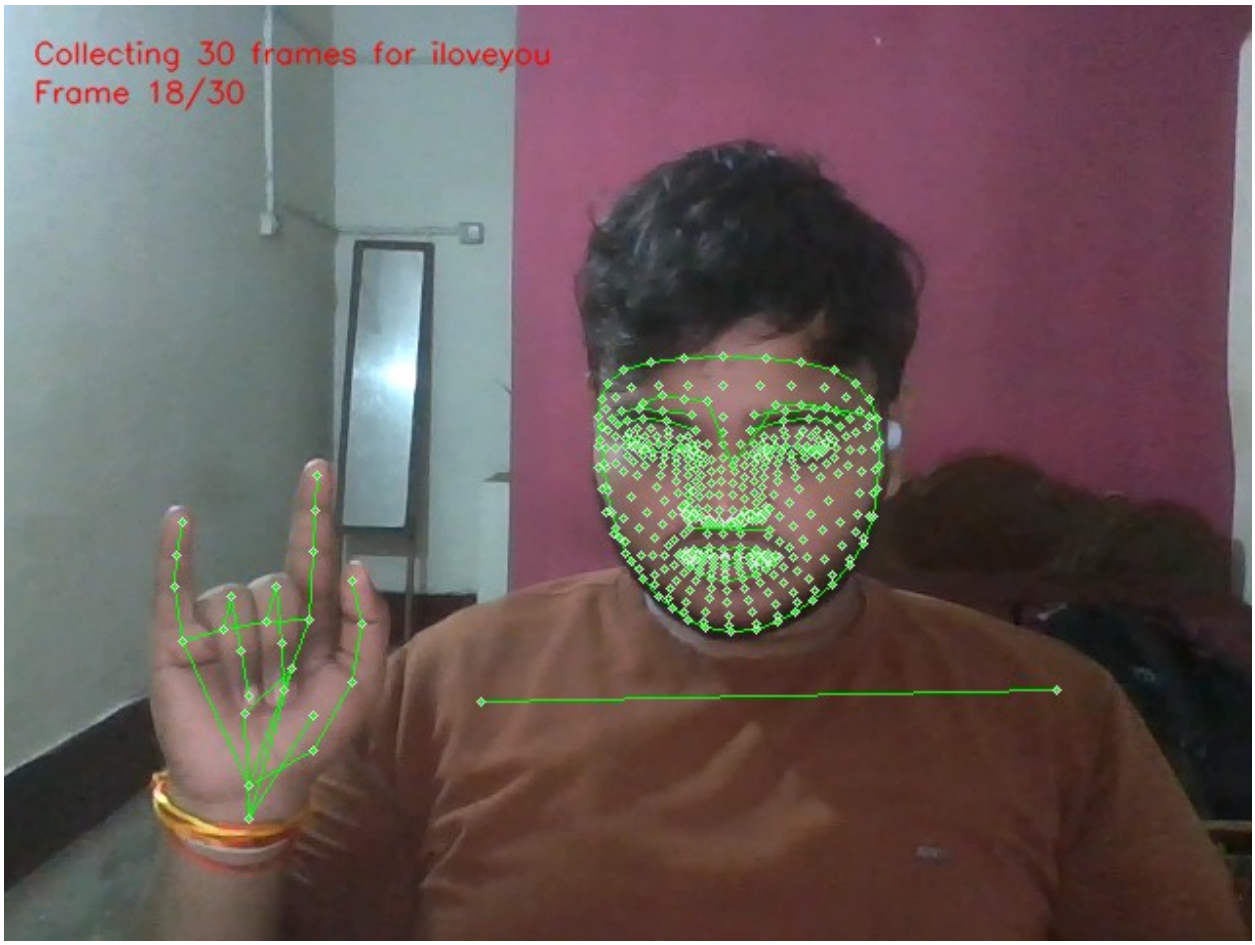



```
<IPython.core.display.Javascript object>
```

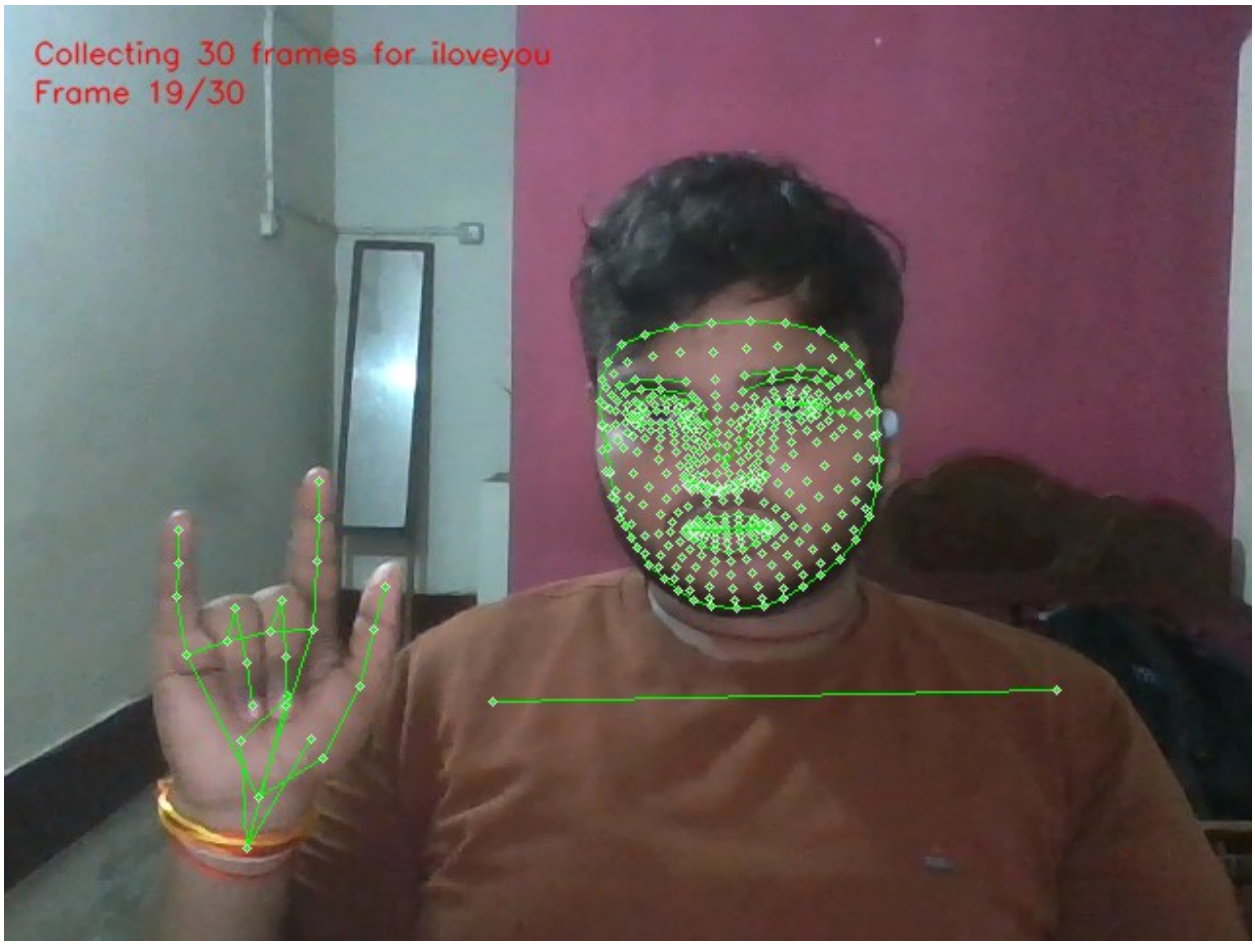

```
<IPython.core.display.Javascript object>
```



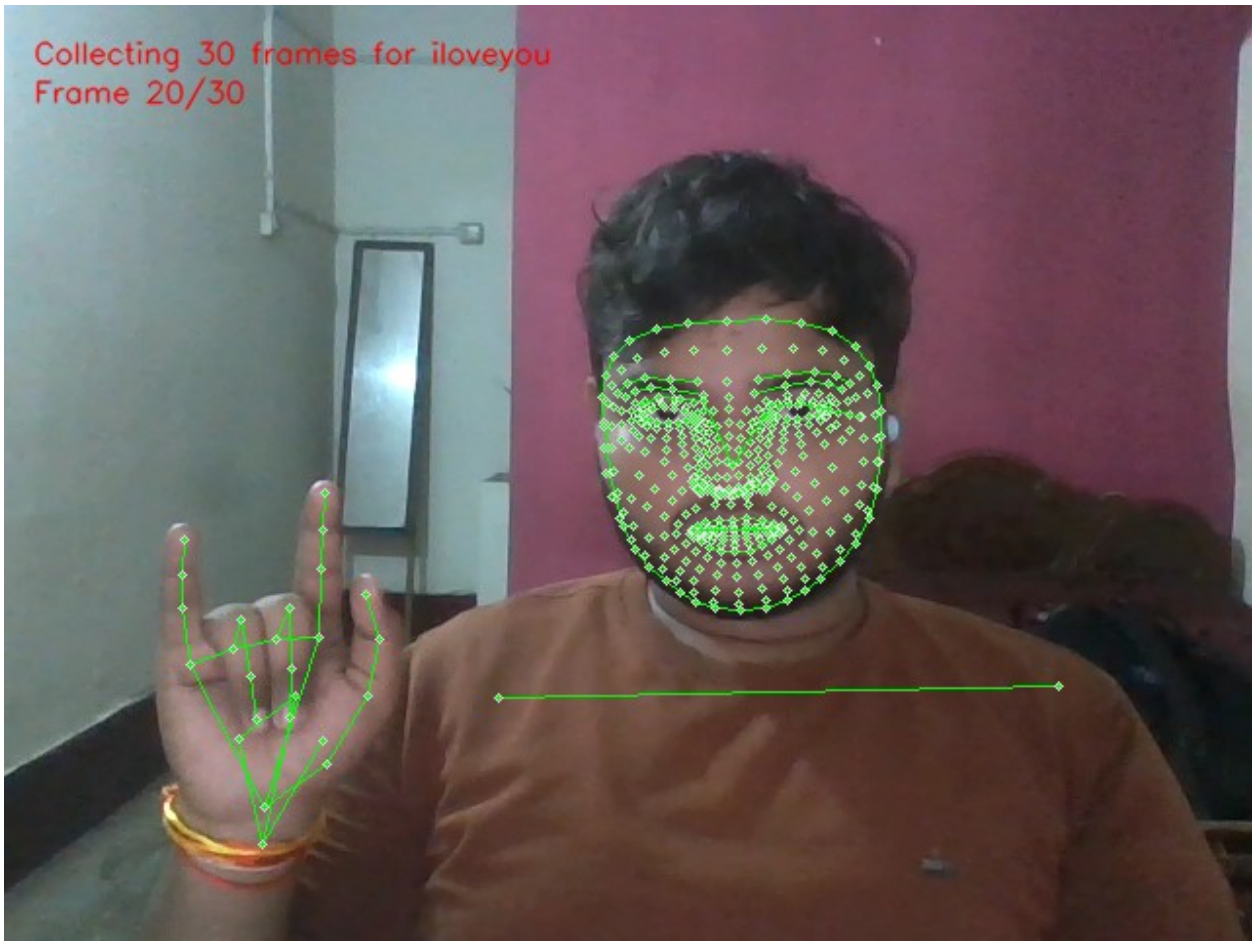

```
<IPython.core.display.Javascript object>
```



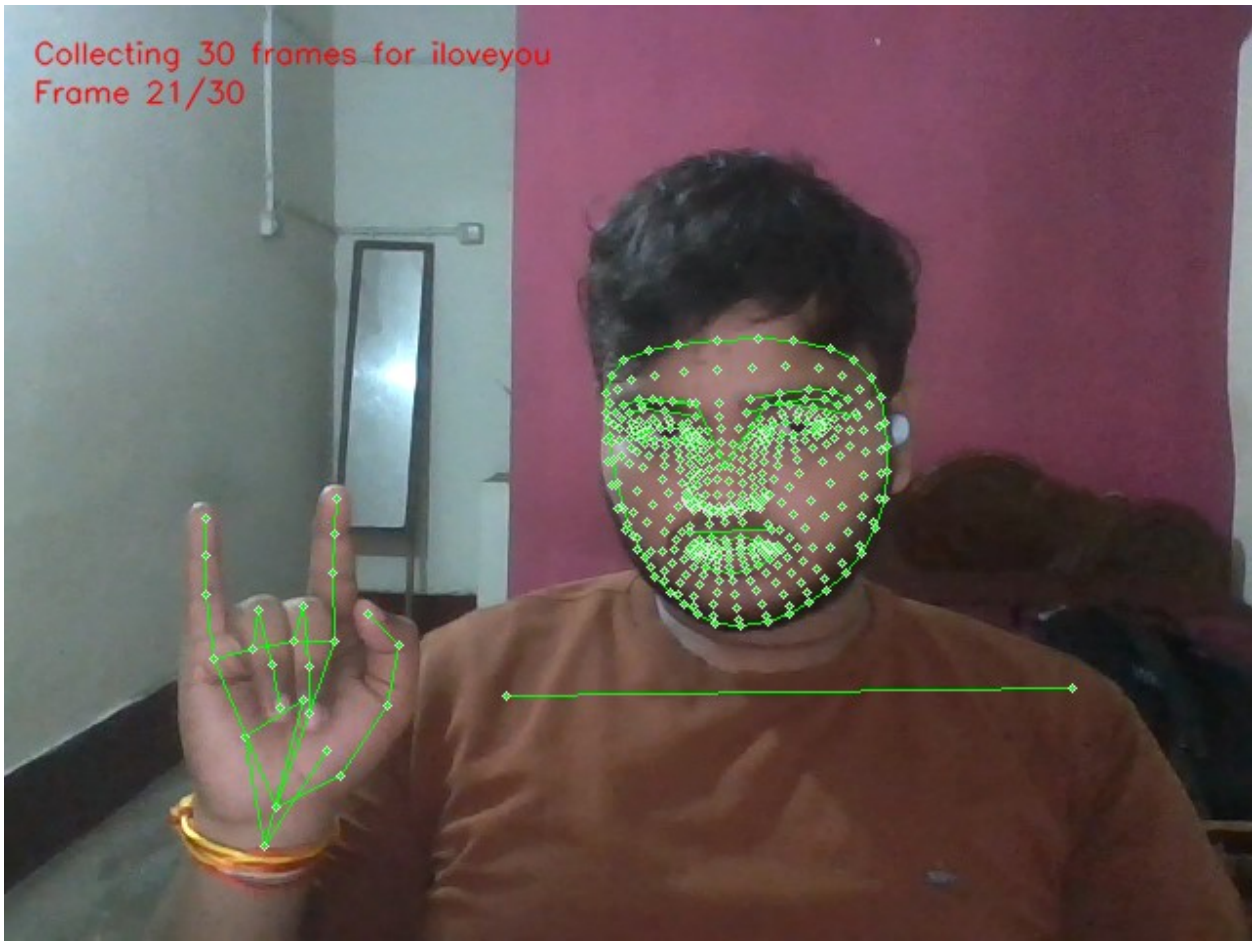

```
<IPython.core.display.Javascript object>
```



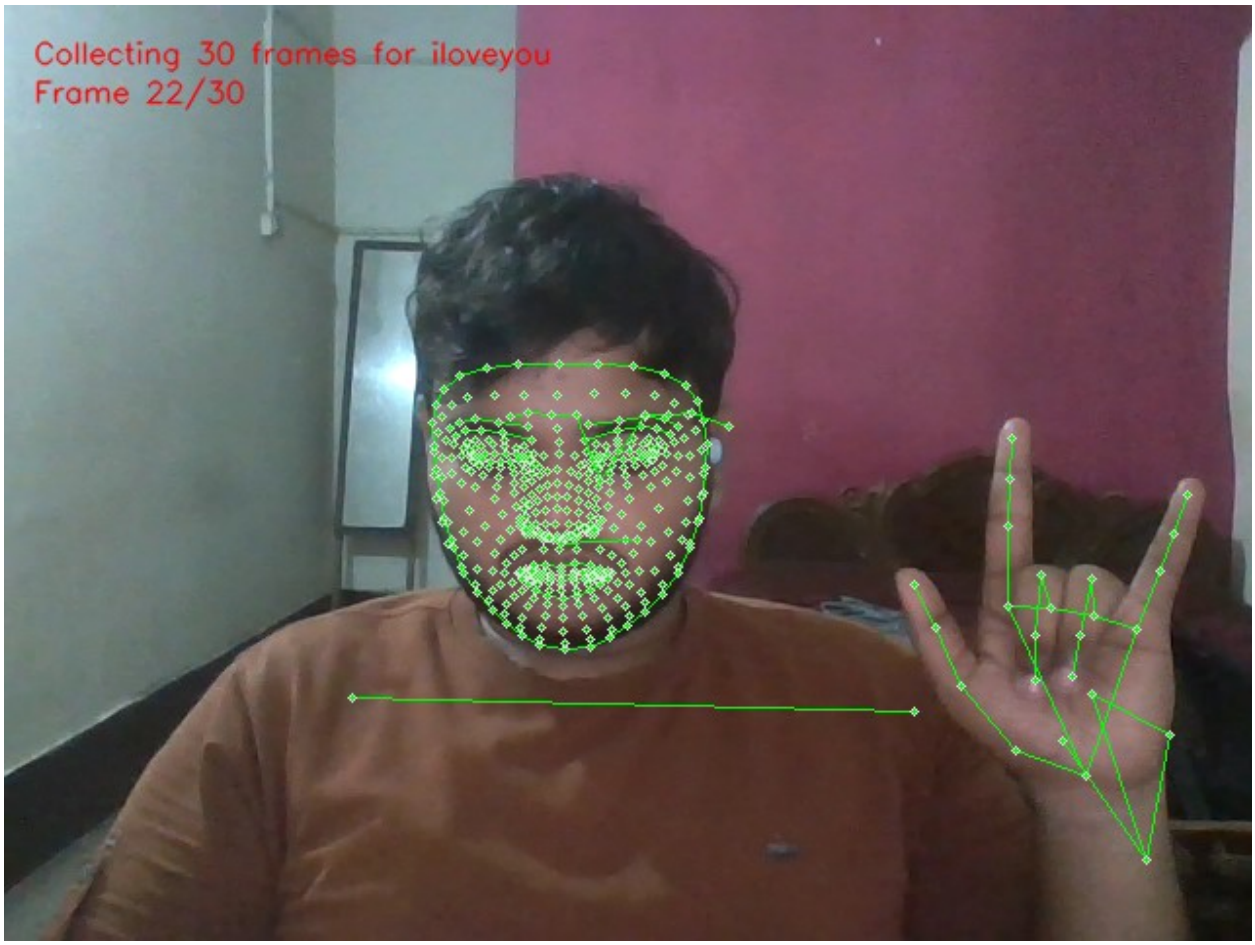

```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>
```




```
<IPython.core.display.Javascript object>

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-23-dfeb24ca1167> in <cell line: 101>()
    103     for frame_num in range(frames_per_action):
    104         # Capture frame from webcam
--> 105         data_url = video_stream()
    106         frame = js_to_image(data_url)
    107

<ipython-input-23-dfeb24ca1167> in video_stream()
     45     ''')
     46     display(js)
---> 47     data = eval_js('stream()')
     48     return data
     49

/usr/local/lib/python3.10/dist-packages/google/colab/output/_js.py in eval_js(script, ignore_result, timeout_sec)
```

```
     38   if ignore_result:
     39     return
---> 40   return _message.read_reply_from_input(request_id,
timeout_sec)
     41
     42

/usr/local/lib/python3.10/dist-packages/google/colab/_message.py in
read_reply_from_input(message_id, timeout_sec)
     94     reply = _read_next_input_message()
     95     if reply == _NOT_READY or not isinstance(reply, dict):
---> 96       time.sleep(0.025)
     97       continue
     98     if (

KeyboardInterrupt:
```

```
import os
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.array(['hello', 'thanks', 'iloveyou'])

# Number of sequences and sequence length
no_sequences = 30
sequence_length = 30

# Load data
label_map = {label: num for num, label in enumerate(actions)}

sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        window = []
        for frame_num in range(sequence_length):
            if action == 'hello':
                res = np.load(os.path.join(DATA_PATH, 'MP_Data',
action, f"{frame_num}.npy"))
            else:
                res = np.load(os.path.join(DATA_PATH, 'MP_Data',
action, f"{action}_{frame_num}.npy"))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

```
# Convert to numpy arrays
X = np.array(sequences)
y = to_categorical(labels).astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.05)

# Build the model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard

log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)

model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu',
input_shape=(30, 1662)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=1000, callbacks=[tb_callback])

model.summary()
res = model.predict(X_test)

actions[np.argmax(res[4])]

actions[np.argmax(y_test[4])]
model.save('action.h5')

model.load_weights('action.h5')
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

```
Epoch 1/1000
3/3 [==============================] - 4s 115ms/step - loss: 1.9999 -
categorical_accuracy: 0.3176
Epoch 2/1000
3/3 [==============================] - 0s 113ms/step - loss: 5.2222 -
categorical_accuracy: 0.3176
Epoch 3/1000
3/3 [==============================] - 0s 129ms/step - loss: 3.9298 -
categorical_accuracy: 0.2353
Epoch 4/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.9222 -
categorical_accuracy: 0.3412
Epoch 5/1000
3/3 [==============================] - 0s 113ms/step - loss: 1.3823 -
categorical_accuracy: 0.4235
Epoch 6/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.1243 -
categorical_accuracy: 0.4588
Epoch 7/1000
3/3 [==============================] - 1s 342ms/step - loss: 1.0755 -
categorical_accuracy: 0.6471
Epoch 8/1000
3/3 [==============================] - 1s 281ms/step - loss: 0.8273 -
categorical_accuracy: 0.6471
Epoch 9/1000
3/3 [==============================] - 1s 256ms/step - loss: 0.5317 -
categorical_accuracy: 0.6941
Epoch 10/1000
3/3 [==============================] - 1s 222ms/step - loss: 0.4567 -
categorical_accuracy: 0.7294
Epoch 11/1000
3/3 [==============================] - 1s 339ms/step - loss: 0.9274 -
categorical_accuracy: 0.6706
Epoch 12/1000
3/3 [==============================] - 1s 288ms/step - loss: 2.6763 -
categorical_accuracy: 0.4824
Epoch 13/1000
3/3 [==============================] - 1s 203ms/step - loss: 3.3835 -
categorical_accuracy: 0.3294
Epoch 14/1000
3/3 [==============================] - 1s 244ms/step - loss: 7.9633 -
categorical_accuracy: 0.3294
Epoch 15/1000
3/3 [==============================] - 1s 182ms/step - loss: 1.1386 -
categorical_accuracy: 0.5647
Epoch 16/1000
3/3 [==============================] - 0s 117ms/step - loss: 10.9172 -
categorical_accuracy: 0.4471
Epoch 17/1000
3/3 [==============================] - 0s 111ms/step - loss: 18.8138 -
```

```
categorical_accuracy: 0.2000
Epoch 18/1000
3/3 [==============================] - 0s 122ms/step - loss: 49.2106 -
categorical_accuracy: 0.1294
Epoch 19/1000
3/3 [==============================] - 0s 116ms/step - loss: 43.5529 -
categorical_accuracy: 0.1059
Epoch 20/1000
3/3 [==============================] - 0s 115ms/step - loss: 36.1457 -
categorical_accuracy: 0.2706
Epoch 21/1000
3/3 [==============================] - 0s 127ms/step - loss: 17.7975 -
categorical_accuracy: 0.3529
Epoch 22/1000
3/3 [==============================] - 0s 114ms/step - loss: 33.3642 -
categorical_accuracy: 0.5294
Epoch 23/1000
3/3 [==============================] - 0s 120ms/step - loss: 78.3029 -
categorical_accuracy: 0.3176
Epoch 24/1000
3/3 [==============================] - 0s 119ms/step - loss: 11.6654 -
categorical_accuracy: 0.7647
Epoch 25/1000
3/3 [==============================] - 0s 117ms/step - loss: 13.8966 -
categorical_accuracy: 0.6118
Epoch 26/1000
3/3 [==============================] - 0s 119ms/step - loss: 20.3308 -
categorical_accuracy: 0.3647
Epoch 27/1000
3/3 [==============================] - 0s 126ms/step - loss: 12.7889 -
categorical_accuracy: 0.3294
Epoch 28/1000
3/3 [==============================] - 0s 114ms/step - loss: 8.0361 -
categorical_accuracy: 0.1647
Epoch 29/1000
3/3 [==============================] - 0s 111ms/step - loss: 21.2982 -
categorical_accuracy: 0.3412
Epoch 30/1000
3/3 [==============================] - 0s 129ms/step - loss: 87.2996 -
categorical_accuracy: 0.4000
Epoch 31/1000
3/3 [==============================] - 0s 112ms/step - loss: 98.1753 -
categorical_accuracy: 0.3412
Epoch 32/1000
3/3 [==============================] - 0s 114ms/step - loss: 27.9521 -
categorical_accuracy: 0.2588
Epoch 33/1000
3/3 [==============================] - 0s 128ms/step - loss: 14.8363 -
categorical_accuracy: 0.4353
Epoch 34/1000
```

```
3/3 [==============================] - 0s 119ms/step - loss: 39.4111 -
categorical_accuracy: 0.3294
Epoch 35/1000
3/3 [==============================] - 0s 112ms/step - loss: 96.3224 -
categorical_accuracy: 0.1882
Epoch 36/1000
3/3 [==============================] - 0s 137ms/step - loss: 32.2102 -
categorical_accuracy: 0.4000
Epoch 37/1000
3/3 [==============================] - 0s 115ms/step - loss: 85.6281 -
categorical_accuracy: 0.4824
Epoch 38/1000
3/3 [==============================] - 0s 109ms/step - loss: 110.9300
- categorical_accuracy: 0.1059
Epoch 39/1000
3/3 [==============================] - 0s 129ms/step - loss: 83.3360 -
categorical_accuracy: 0.3529
Epoch 40/1000
3/3 [==============================] - 0s 113ms/step - loss: 40.5483 -
categorical_accuracy: 0.2941
Epoch 41/1000
3/3 [==============================] - 0s 113ms/step - loss: 28.6116 -
categorical_accuracy: 0.3176
Epoch 42/1000
3/3 [==============================] - 0s 124ms/step - loss: 26.0217 -
categorical_accuracy: 0.1765
Epoch 43/1000
3/3 [==============================] - 0s 135ms/step - loss: 38.1958 -
categorical_accuracy: 0.3529
Epoch 44/1000
3/3 [==============================] - 1s 227ms/step - loss: 19.3307 -
categorical_accuracy: 0.3059
Epoch 45/1000
3/3 [==============================] - 1s 237ms/step - loss: 12.3476 -
categorical_accuracy: 0.3294
Epoch 46/1000
3/3 [==============================] - 1s 197ms/step - loss: 14.6712 -
categorical_accuracy: 0.3176
Epoch 47/1000
3/3 [==============================] - 1s 226ms/step - loss: 6.3911 -
categorical_accuracy: 0.4235
Epoch 48/1000
3/3 [==============================] - 0s 119ms/step - loss: 7.5245 -
categorical_accuracy: 0.1882
Epoch 49/1000
3/3 [==============================] - 0s 117ms/step - loss: 3.1888 -
categorical_accuracy: 0.5882
Epoch 50/1000
3/3 [==============================] - 0s 115ms/step - loss: 4.3452 -
categorical_accuracy: 0.5294
```

```
Epoch 51/1000
3/3 [==============================] - 0s 128ms/step - loss: 2.0670 -
categorical_accuracy: 0.3765
Epoch 52/1000
3/3 [==============================] - 0s 122ms/step - loss: 3.0598 -
categorical_accuracy: 0.3176
Epoch 53/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.2701 -
categorical_accuracy: 0.5059
Epoch 54/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.7293 -
categorical_accuracy: 0.6706
Epoch 55/1000
3/3 [==============================] - 0s 143ms/step - loss: 0.8819 -
categorical_accuracy: 0.6471
Epoch 56/1000
3/3 [==============================] - 0s 115ms/step - loss: 1.0809 -
categorical_accuracy: 0.5765
Epoch 57/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.7322 -
categorical_accuracy: 0.8235
Epoch 58/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.2670 -
categorical_accuracy: 0.5412
Epoch 59/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.3788 -
categorical_accuracy: 0.8706
Epoch 60/1000
3/3 [==============================] - 0s 112ms/step - loss: 0.2769 -
categorical_accuracy: 0.8941
Epoch 61/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.3740 -
categorical_accuracy: 0.8118
Epoch 62/1000
3/3 [==============================] - 0s 109ms/step - loss: 0.2749 -
categorical_accuracy: 0.9176
Epoch 63/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.4310 -
categorical_accuracy: 0.8471
Epoch 64/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.3961 -
categorical_accuracy: 0.9059
Epoch 65/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.4549 -
categorical_accuracy: 0.7529
Epoch 66/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.6241 -
categorical_accuracy: 0.5529
Epoch 67/1000
3/3 [==============================] - 0s 112ms/step - loss: 0.2511 -
```

```
categorical_accuracy: 0.8588
Epoch 68/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.2378 -
categorical_accuracy: 1.0000
Epoch 69/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.3641 -
categorical_accuracy: 0.8941
Epoch 70/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.4523 -
categorical_accuracy: 0.8706
Epoch 71/1000
3/3 [==============================] - 0s 111ms/step - loss: 1.1152 -
categorical_accuracy: 0.5176
Epoch 72/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.6781 -
categorical_accuracy: 0.7647
Epoch 73/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.3690 -
categorical_accuracy: 0.7412
Epoch 74/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.7345 -
categorical_accuracy: 0.5529
Epoch 75/1000
3/3 [==============================] - 0s 151ms/step - loss: 0.2964 -
categorical_accuracy: 0.8235
Epoch 76/1000
3/3 [==============================] - 1s 227ms/step - loss: 0.5032 -
categorical_accuracy: 0.8706
Epoch 77/1000
3/3 [==============================] - 1s 227ms/step - loss: 0.4146 -
categorical_accuracy: 0.7412
Epoch 78/1000
3/3 [==============================] - 1s 220ms/step - loss: 0.4272 -
categorical_accuracy: 0.6824
Epoch 79/1000
3/3 [==============================] - 1s 251ms/step - loss: 0.5322 -
categorical_accuracy: 0.8588
Epoch 80/1000
3/3 [==============================] - 1s 162ms/step - loss: 0.7113 -
categorical_accuracy: 0.6471
Epoch 81/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.4886 -
categorical_accuracy: 0.6471
Epoch 82/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.4993 -
categorical_accuracy: 0.7647
Epoch 83/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.3401 -
categorical_accuracy: 0.8941
Epoch 84/1000
```

```
3/3 [==============================] - 0s 125ms/step - loss: 0.2531 -
categorical_accuracy: 0.8471
Epoch 85/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.1773 -
categorical_accuracy: 0.8941
Epoch 86/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.2355 -
categorical_accuracy: 0.8353
Epoch 87/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.1208 -
categorical_accuracy: 1.0000
Epoch 88/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.1388 -
categorical_accuracy: 1.0000
Epoch 89/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.1040 -
categorical_accuracy: 1.0000
Epoch 90/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.1040 -
categorical_accuracy: 1.0000
Epoch 91/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0962 -
categorical_accuracy: 1.0000
Epoch 92/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0825 -
categorical_accuracy: 1.0000
Epoch 93/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.0808 -
categorical_accuracy: 1.0000
Epoch 94/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0732 -
categorical_accuracy: 1.0000
Epoch 95/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0605 -
categorical_accuracy: 1.0000
Epoch 96/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0621 -
categorical_accuracy: 1.0000
Epoch 97/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0550 -
categorical_accuracy: 1.0000
Epoch 98/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0524 -
categorical_accuracy: 1.0000
Epoch 99/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0524 -
categorical_accuracy: 1.0000
Epoch 100/1000
3/3 [==============================] - 0s 109ms/step - loss: 0.0480 -
categorical_accuracy: 1.0000
```

```
Epoch 101/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0479 -
categorical_accuracy: 1.0000
Epoch 102/1000
3/3 [==============================] - 0s 152ms/step - loss: 0.0453 -
categorical_accuracy: 1.0000
Epoch 103/1000
3/3 [==============================] - 0s 138ms/step - loss: 0.0429 -
categorical_accuracy: 1.0000
Epoch 104/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0412 -
categorical_accuracy: 1.0000
Epoch 105/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.0391 -
categorical_accuracy: 1.0000
Epoch 106/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0373 -
categorical_accuracy: 1.0000
Epoch 107/1000
3/3 [==============================] - 0s 112ms/step - loss: 0.0354 -
categorical_accuracy: 1.0000
Epoch 108/1000
3/3 [==============================] - 1s 205ms/step - loss: 0.0314 -
categorical_accuracy: 1.0000
Epoch 109/1000
3/3 [==============================] - 1s 243ms/step - loss: 0.0294 -
categorical_accuracy: 1.0000
Epoch 110/1000
3/3 [==============================] - 1s 187ms/step - loss: 0.0288 -
categorical_accuracy: 1.0000
Epoch 111/1000
3/3 [==============================] - 1s 243ms/step - loss: 0.0276 -
categorical_accuracy: 1.0000
Epoch 112/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0267 -
categorical_accuracy: 1.0000
Epoch 113/1000
3/3 [==============================] - 0s 140ms/step - loss: 0.0257 -
categorical_accuracy: 1.0000
Epoch 114/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0246 -
categorical_accuracy: 1.0000
Epoch 115/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0236 -
categorical_accuracy: 1.0000
Epoch 116/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0228 -
categorical_accuracy: 1.0000
Epoch 117/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0222 -
```

```
categorical_accuracy: 1.0000
Epoch 118/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0213 -
categorical_accuracy: 1.0000
Epoch 119/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0207 -
categorical_accuracy: 1.0000
Epoch 120/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0200 -
categorical_accuracy: 1.0000
Epoch 121/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0194 -
categorical_accuracy: 1.0000
Epoch 122/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0192 -
categorical_accuracy: 1.0000
Epoch 123/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0189 -
categorical_accuracy: 1.0000
Epoch 124/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0187 -
categorical_accuracy: 1.0000
Epoch 125/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0181 -
categorical_accuracy: 1.0000
Epoch 126/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0179 -
categorical_accuracy: 1.0000
Epoch 127/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0176 -
categorical_accuracy: 1.0000
Epoch 128/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0173 -
categorical_accuracy: 1.0000
Epoch 129/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0170 -
categorical_accuracy: 1.0000
Epoch 130/1000
3/3 [==============================] - 0s 140ms/step - loss: 0.0167 -
categorical_accuracy: 1.0000
Epoch 131/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0165 -
categorical_accuracy: 1.0000
Epoch 132/1000
3/3 [==============================] - 0s 153ms/step - loss: 0.0162 -
categorical_accuracy: 1.0000
Epoch 133/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0160 -
categorical_accuracy: 1.0000
Epoch 134/1000
```

```
3/3 [==============================] - 0s 124ms/step - loss: 0.0157 -
categorical_accuracy: 1.0000
Epoch 135/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0155 -
categorical_accuracy: 1.0000
Epoch 136/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0152 -
categorical_accuracy: 1.0000
Epoch 137/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0149 -
categorical_accuracy: 1.0000
Epoch 138/1000
3/3 [==============================] - 0s 134ms/step - loss: 0.0147 -
categorical_accuracy: 1.0000
Epoch 139/1000
3/3 [==============================] - 0s 150ms/step - loss: 0.0144 -
categorical_accuracy: 1.0000
Epoch 140/1000
3/3 [==============================] - 1s 238ms/step - loss: 0.0142 -
categorical_accuracy: 1.0000
Epoch 141/1000
3/3 [==============================] - 1s 228ms/step - loss: 0.0139 -
categorical_accuracy: 1.0000
Epoch 142/1000
3/3 [==============================] - 1s 262ms/step - loss: 0.0137 -
categorical_accuracy: 1.0000
Epoch 143/1000
3/3 [==============================] - 1s 239ms/step - loss: 0.0134 -
categorical_accuracy: 1.0000
Epoch 144/1000
3/3 [==============================] - 1s 172ms/step - loss: 0.0133 -
categorical_accuracy: 1.0000
Epoch 145/1000
3/3 [==============================] - 0s 136ms/step - loss: 0.0130 -
categorical_accuracy: 1.0000
Epoch 146/1000
3/3 [==============================] - 0s 131ms/step - loss: 0.0127 -
categorical_accuracy: 1.0000
Epoch 147/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0126 -
categorical_accuracy: 1.0000
Epoch 148/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0124 -
categorical_accuracy: 1.0000
Epoch 149/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0121 -
categorical_accuracy: 1.0000
Epoch 150/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0120 -
categorical_accuracy: 1.0000
```

```
Epoch 151/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.0117 -
categorical_accuracy: 1.0000
Epoch 152/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0115 -
categorical_accuracy: 1.0000
Epoch 153/1000
3/3 [==============================] - 0s 110ms/step - loss: 0.0113 -
categorical_accuracy: 1.0000
Epoch 154/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0111 -
categorical_accuracy: 1.0000
Epoch 155/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0109 -
categorical_accuracy: 1.0000
Epoch 156/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0107 -
categorical_accuracy: 1.0000
Epoch 157/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0105 -
categorical_accuracy: 1.0000
Epoch 158/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.0103 -
categorical_accuracy: 1.0000
Epoch 159/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0101 -
categorical_accuracy: 1.0000
Epoch 160/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0099 -
categorical_accuracy: 1.0000
Epoch 161/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0097 -
categorical_accuracy: 1.0000
Epoch 162/1000
3/3 [==============================] - 0s 131ms/step - loss: 0.0095 -
categorical_accuracy: 1.0000
Epoch 163/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0092 -
categorical_accuracy: 1.0000
Epoch 164/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0090 -
categorical_accuracy: 1.0000
Epoch 165/1000
3/3 [==============================] - 0s 140ms/step - loss: 0.0088 -
categorical_accuracy: 1.0000
Epoch 166/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0085 -
categorical_accuracy: 1.0000
Epoch 167/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0083 -
```

```
categorical_accuracy: 1.0000
Epoch 168/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0081 -
categorical_accuracy: 1.0000
Epoch 169/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0078 -
categorical_accuracy: 1.0000
Epoch 170/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0076 -
categorical_accuracy: 1.0000
Epoch 171/1000
3/3 [==============================] - 0s 138ms/step - loss: 0.0074 -
categorical_accuracy: 1.0000
Epoch 172/1000
3/3 [==============================] - 1s 356ms/step - loss: 0.0071 -
categorical_accuracy: 1.0000
Epoch 173/1000
3/3 [==============================] - 1s 240ms/step - loss: 0.0068 -
categorical_accuracy: 1.0000
Epoch 174/1000
3/3 [==============================] - 1s 247ms/step - loss: 0.0065 -
categorical_accuracy: 1.0000
Epoch 175/1000
3/3 [==============================] - 1s 225ms/step - loss: 0.0062 -
categorical_accuracy: 1.0000
Epoch 176/1000
3/3 [==============================] - 0s 135ms/step - loss: 0.0059 -
categorical_accuracy: 1.0000
Epoch 177/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0055 -
categorical_accuracy: 1.0000
Epoch 178/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0053 -
categorical_accuracy: 1.0000
Epoch 179/1000
3/3 [==============================] - 0s 136ms/step - loss: 0.0049 -
categorical_accuracy: 1.0000
Epoch 180/1000
3/3 [==============================] - 0s 112ms/step - loss: 0.0046 -
categorical_accuracy: 1.0000
Epoch 181/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.0044 -
categorical_accuracy: 1.0000
Epoch 182/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0042 -
categorical_accuracy: 1.0000
Epoch 183/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0039 -
categorical_accuracy: 1.0000
Epoch 184/1000
```

```
3/3 [==============================] - 0s 126ms/step - loss: 0.0038 -
categorical_accuracy: 1.0000
Epoch 185/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0036 -
categorical_accuracy: 1.0000
Epoch 186/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0035 -
categorical_accuracy: 1.0000
Epoch 187/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0033 -
categorical_accuracy: 1.0000
Epoch 188/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0032 -
categorical_accuracy: 1.0000
Epoch 189/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0031 -
categorical_accuracy: 1.0000
Epoch 190/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0030 -
categorical_accuracy: 1.0000
Epoch 191/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0028 -
categorical_accuracy: 1.0000
Epoch 192/1000
3/3 [==============================] - 0s 149ms/step - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 193/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0026 -
categorical_accuracy: 1.0000
Epoch 194/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0025 -
categorical_accuracy: 1.0000
Epoch 195/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 196/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 197/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 198/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0022 -
categorical_accuracy: 1.0000
Epoch 199/1000
3/3 [==============================] - 0s 111ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 200/1000
3/3 [==============================] - 0s 107ms/step - loss: 0.0020 -
```

```
categorical_accuracy: 1.0000
Epoch 201/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 202/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 203/1000
3/3 [==============================] - 1s 231ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 204/1000
3/3 [==============================] - 1s 186ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 205/1000
3/3 [==============================] - 1s 241ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 206/1000
3/3 [==============================] - 1s 266ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 207/1000
3/3 [==============================] - 1s 164ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 208/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 209/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 210/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 211/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 212/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 213/1000
3/3 [==============================] - 0s 144ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 214/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 215/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 216/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
```

```
Epoch 217/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 218/1000
3/3 [==============================] - 0s 129ms/step - loss: 9.6216e-
04 - categorical_accuracy: 1.0000
Epoch 219/1000
3/3 [==============================] - 0s 115ms/step - loss: 8.8435e-
04 - categorical_accuracy: 1.0000
Epoch 220/1000
3/3 [==============================] - 0s 121ms/step - loss: 7.9473e-
04 - categorical_accuracy: 1.0000
Epoch 221/1000
3/3 [==============================] - 0s 135ms/step - loss: 7.5704e-
04 - categorical_accuracy: 1.0000
Epoch 222/1000
3/3 [==============================] - 0s 116ms/step - loss: 7.2258e-
04 - categorical_accuracy: 1.0000
Epoch 223/1000
3/3 [==============================] - 0s 155ms/step - loss: 6.7432e-
04 - categorical_accuracy: 1.0000
Epoch 224/1000
3/3 [==============================] - 1s 230ms/step - loss: 6.3397e-
04 - categorical_accuracy: 1.0000
Epoch 225/1000
3/3 [==============================] - 0s 116ms/step - loss: 6.1357e-
04 - categorical_accuracy: 1.0000
Epoch 226/1000
3/3 [==============================] - 0s 112ms/step - loss: 5.9147e-
04 - categorical_accuracy: 1.0000
Epoch 227/1000
3/3 [==============================] - 0s 121ms/step - loss: 5.6651e-
04 - categorical_accuracy: 1.0000
Epoch 228/1000
3/3 [==============================] - 0s 119ms/step - loss: 5.4742e-
04 - categorical_accuracy: 1.0000
Epoch 229/1000
3/3 [==============================] - 0s 112ms/step - loss: 5.3337e-
04 - categorical_accuracy: 1.0000
Epoch 230/1000
3/3 [==============================] - 0s 124ms/step - loss: 5.1866e-
04 - categorical_accuracy: 1.0000
Epoch 231/1000
3/3 [==============================] - 1s 235ms/step - loss: 5.0430e-
04 - categorical_accuracy: 1.0000
Epoch 232/1000
3/3 [==============================] - 1s 425ms/step - loss: 4.9353e-
04 - categorical_accuracy: 1.0000
Epoch 233/1000
```

```
3/3 [==============================] - 1s 420ms/step - loss: 4.8414e-
04 - categorical_accuracy: 1.0000
Epoch 234/1000
3/3 [==============================] - 1s 552ms/step - loss: 4.7290e-
04 - categorical_accuracy: 1.0000
Epoch 235/1000
3/3 [==============================] - 1s 330ms/step - loss: 4.6179e-
04 - categorical_accuracy: 1.0000
Epoch 236/1000
3/3 [==============================] - 1s 185ms/step - loss: 4.5231e-
04 - categorical_accuracy: 1.0000
Epoch 237/1000
3/3 [==============================] - 1s 310ms/step - loss: 4.4402e-
04 - categorical_accuracy: 1.0000
Epoch 238/1000
3/3 [==============================] - 1s 262ms/step - loss: 4.3470e-
04 - categorical_accuracy: 1.0000
Epoch 239/1000
3/3 [==============================] - 1s 277ms/step - loss: 4.2405e-
04 - categorical_accuracy: 1.0000
Epoch 240/1000
3/3 [==============================] - 1s 199ms/step - loss: 4.1563e-
04 - categorical_accuracy: 1.0000
Epoch 241/1000
3/3 [==============================] - 0s 114ms/step - loss: 4.0765e-
04 - categorical_accuracy: 1.0000
Epoch 242/1000
3/3 [==============================] - 0s 120ms/step - loss: 3.9939e-
04 - categorical_accuracy: 1.0000
Epoch 243/1000
3/3 [==============================] - 0s 128ms/step - loss: 3.9023e-
04 - categorical_accuracy: 1.0000
Epoch 244/1000
3/3 [==============================] - 0s 118ms/step - loss: 3.8219e-
04 - categorical_accuracy: 1.0000
Epoch 245/1000
3/3 [==============================] - 0s 122ms/step - loss: 3.7401e-
04 - categorical_accuracy: 1.0000
Epoch 246/1000
3/3 [==============================] - 0s 128ms/step - loss: 3.6633e-
04 - categorical_accuracy: 1.0000
Epoch 247/1000
3/3 [==============================] - 0s 125ms/step - loss: 3.5815e-
04 - categorical_accuracy: 1.0000
Epoch 248/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.5026e-
04 - categorical_accuracy: 1.0000
Epoch 249/1000
3/3 [==============================] - 0s 111ms/step - loss: 3.4228e-
```

```
04 - categorical_accuracy: 1.0000
Epoch 250/1000
3/3 [==============================] - 0s 118ms/step - loss: 3.3453e-
04 - categorical_accuracy: 1.0000
Epoch 251/1000
3/3 [==============================] - 0s 115ms/step - loss: 3.2656e-
04 - categorical_accuracy: 1.0000
Epoch 252/1000
3/3 [==============================] - 0s 114ms/step - loss: 3.1894e-
04 - categorical_accuracy: 1.0000
Epoch 253/1000
3/3 [==============================] - 0s 124ms/step - loss: 3.1132e-
04 - categorical_accuracy: 1.0000
Epoch 254/1000
3/3 [==============================] - 1s 331ms/step - loss: 3.0346e-
04 - categorical_accuracy: 1.0000
Epoch 255/1000
3/3 [==============================] - 1s 365ms/step - loss: 2.9554e-
04 - categorical_accuracy: 1.0000
Epoch 256/1000
3/3 [==============================] - 1s 266ms/step - loss: 2.8802e-
04 - categorical_accuracy: 1.0000
Epoch 257/1000
3/3 [==============================] - 1s 214ms/step - loss: 2.8073e-
04 - categorical_accuracy: 1.0000
Epoch 258/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.7309e-
04 - categorical_accuracy: 1.0000
Epoch 259/1000
3/3 [==============================] - 0s 118ms/step - loss: 2.6604e-
04 - categorical_accuracy: 1.0000
Epoch 260/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.5867e-
04 - categorical_accuracy: 1.0000
Epoch 261/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.5123e-
04 - categorical_accuracy: 1.0000
Epoch 262/1000
3/3 [==============================] - 0s 142ms/step - loss: 2.4354e-
04 - categorical_accuracy: 1.0000
Epoch 263/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.3648e-
04 - categorical_accuracy: 1.0000
Epoch 264/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.2947e-
04 - categorical_accuracy: 1.0000
Epoch 265/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.2300e-
04 - categorical_accuracy: 1.0000
```

```
Epoch 266/1000
3/3 [==============================] - 0s 114ms/step - loss: 2.1654e-
04 - categorical_accuracy: 1.0000
Epoch 267/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.0995e-
04 - categorical_accuracy: 1.0000
Epoch 268/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.0446e-
04 - categorical_accuracy: 1.0000
Epoch 269/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.9868e-
04 - categorical_accuracy: 1.0000
Epoch 270/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.9339e-
04 - categorical_accuracy: 1.0000
Epoch 271/1000
3/3 [==============================] - 0s 113ms/step - loss: 1.8833e-
04 - categorical_accuracy: 1.0000
Epoch 272/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.8327e-
04 - categorical_accuracy: 1.0000
Epoch 273/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.7856e-
04 - categorical_accuracy: 1.0000
Epoch 274/1000
3/3 [==============================] - 0s 114ms/step - loss: 1.7422e-
04 - categorical_accuracy: 1.0000
Epoch 275/1000
3/3 [==============================] - 0s 115ms/step - loss: 1.6963e-
04 - categorical_accuracy: 1.0000
Epoch 276/1000
3/3 [==============================] - 0s 125ms/step - loss: 1.6543e-
04 - categorical_accuracy: 1.0000
Epoch 277/1000
3/3 [==============================] - 0s 111ms/step - loss: 1.6133e-
04 - categorical_accuracy: 1.0000
Epoch 278/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.5769e-
04 - categorical_accuracy: 1.0000
Epoch 279/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.5391e-
04 - categorical_accuracy: 1.0000
Epoch 280/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.5016e-
04 - categorical_accuracy: 1.0000
Epoch 281/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.4693e-
04 - categorical_accuracy: 1.0000
Epoch 282/1000
```

```
3/3 [==============================] - 0s 115ms/step - loss: 1.4353e-
04 - categorical_accuracy: 1.0000
Epoch 283/1000
3/3 [==============================] - 0s 111ms/step - loss: 1.4023e-
04 - categorical_accuracy: 1.0000
Epoch 284/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.3749e-
04 - categorical_accuracy: 1.0000
Epoch 285/1000
3/3 [==============================] - 1s 182ms/step - loss: 1.3429e-
04 - categorical_accuracy: 1.0000
Epoch 286/1000
3/3 [==============================] - 1s 245ms/step - loss: 1.3142e-
04 - categorical_accuracy: 1.0000
Epoch 287/1000
3/3 [==============================] - 1s 204ms/step - loss: 1.2888e-
04 - categorical_accuracy: 1.0000
Epoch 288/1000
3/3 [==============================] - 1s 226ms/step - loss: 1.2614e-
04 - categorical_accuracy: 1.0000
Epoch 289/1000
3/3 [==============================] - 1s 269ms/step - loss: 1.2362e-
04 - categorical_accuracy: 1.0000
Epoch 290/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.2111e-
04 - categorical_accuracy: 1.0000
Epoch 291/1000
3/3 [==============================] - 0s 132ms/step - loss: 1.1881e-
04 - categorical_accuracy: 1.0000
Epoch 292/1000
3/3 [==============================] - 0s 129ms/step - loss: 1.1658e-
04 - categorical_accuracy: 1.0000
Epoch 293/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.1432e-
04 - categorical_accuracy: 1.0000
Epoch 294/1000
3/3 [==============================] - 0s 125ms/step - loss: 1.1225e-
04 - categorical_accuracy: 1.0000
Epoch 295/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.1015e-
04 - categorical_accuracy: 1.0000
Epoch 296/1000
3/3 [==============================] - 0s 112ms/step - loss: 1.0816e-
04 - categorical_accuracy: 1.0000
Epoch 297/1000
3/3 [==============================] - 0s 138ms/step - loss: 1.0627e-
04 - categorical_accuracy: 1.0000
Epoch 298/1000
3/3 [==============================] - 0s 145ms/step - loss: 1.0446e-
```

```
04 - categorical_accuracy: 1.0000
Epoch 299/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.0260e-
04 - categorical_accuracy: 1.0000
Epoch 300/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.0092e-
04 - categorical_accuracy: 1.0000
Epoch 301/1000
3/3 [==============================] - 0s 122ms/step - loss: 9.9201e-
05 - categorical_accuracy: 1.0000
Epoch 302/1000
3/3 [==============================] - 0s 130ms/step - loss: 9.7501e-
05 - categorical_accuracy: 1.0000
Epoch 303/1000
3/3 [==============================] - 0s 116ms/step - loss: 9.5995e-
05 - categorical_accuracy: 1.0000
Epoch 304/1000
3/3 [==============================] - 0s 116ms/step - loss: 9.4470e-
05 - categorical_accuracy: 1.0000
Epoch 305/1000
3/3 [==============================] - 0s 125ms/step - loss: 9.2944e-
05 - categorical_accuracy: 1.0000
Epoch 306/1000
3/3 [==============================] - 0s 111ms/step - loss: 9.1462e-
05 - categorical_accuracy: 1.0000
Epoch 307/1000
3/3 [==============================] - 0s 126ms/step - loss: 9.0125e-
05 - categorical_accuracy: 1.0000
Epoch 308/1000
3/3 [==============================] - 0s 120ms/step - loss: 8.8690e-
05 - categorical_accuracy: 1.0000
Epoch 309/1000
3/3 [==============================] - 0s 113ms/step - loss: 8.7302e-
05 - categorical_accuracy: 1.0000
Epoch 310/1000
3/3 [==============================] - 0s 120ms/step - loss: 8.6020e-
05 - categorical_accuracy: 1.0000
Epoch 311/1000
3/3 [==============================] - 0s 126ms/step - loss: 8.4639e-
05 - categorical_accuracy: 1.0000
Epoch 312/1000
3/3 [==============================] - 0s 134ms/step - loss: 8.3440e-
05 - categorical_accuracy: 1.0000
Epoch 313/1000
3/3 [==============================] - 0s 125ms/step - loss: 8.2323e-
05 - categorical_accuracy: 1.0000
Epoch 314/1000
3/3 [==============================] - 0s 116ms/step - loss: 8.1099e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 315/1000
3/3 [==============================] - 0s 115ms/step - loss: 7.9942e-
05 - categorical_accuracy: 1.0000
Epoch 316/1000
3/3 [==============================] - 0s 126ms/step - loss: 7.8831e-
05 - categorical_accuracy: 1.0000
Epoch 317/1000
3/3 [==============================] - 1s 204ms/step - loss: 7.7784e-
05 - categorical_accuracy: 1.0000
Epoch 318/1000
3/3 [==============================] - 1s 229ms/step - loss: 7.6811e-
05 - categorical_accuracy: 1.0000
Epoch 319/1000
3/3 [==============================] - 1s 212ms/step - loss: 7.5675e-
05 - categorical_accuracy: 1.0000
Epoch 320/1000
3/3 [==============================] - 1s 207ms/step - loss: 7.4731e-
05 - categorical_accuracy: 1.0000
Epoch 321/1000
3/3 [==============================] - 1s 267ms/step - loss: 7.3765e-
05 - categorical_accuracy: 1.0000
Epoch 322/1000
3/3 [==============================] - 0s 120ms/step - loss: 7.2834e-
05 - categorical_accuracy: 1.0000
Epoch 323/1000
3/3 [==============================] - 0s 116ms/step - loss: 7.1831e-
05 - categorical_accuracy: 1.0000
Epoch 324/1000
3/3 [==============================] - 0s 121ms/step - loss: 7.0951e-
05 - categorical_accuracy: 1.0000
Epoch 325/1000
3/3 [==============================] - 0s 125ms/step - loss: 7.0066e-
05 - categorical_accuracy: 1.0000
Epoch 326/1000
3/3 [==============================] - 0s 112ms/step - loss: 6.9229e-
05 - categorical_accuracy: 1.0000
Epoch 327/1000
3/3 [==============================] - 0s 113ms/step - loss: 6.8349e-
05 - categorical_accuracy: 1.0000
Epoch 328/1000
3/3 [==============================] - 0s 114ms/step - loss: 6.7552e-
05 - categorical_accuracy: 1.0000
Epoch 329/1000
3/3 [==============================] - 0s 111ms/step - loss: 6.6719e-
05 - categorical_accuracy: 1.0000
Epoch 330/1000
3/3 [==============================] - 0s 111ms/step - loss: 6.5867e-
05 - categorical_accuracy: 1.0000
Epoch 331/1000
```

```
3/3 [==============================] - 0s 117ms/step - loss: 6.5064e-
05 - categorical_accuracy: 1.0000
Epoch 332/1000
3/3 [==============================] - 0s 124ms/step - loss: 6.4300e-
05 - categorical_accuracy: 1.0000
Epoch 333/1000
3/3 [==============================] - 0s 126ms/step - loss: 6.3606e-
05 - categorical_accuracy: 1.0000
Epoch 334/1000
3/3 [==============================] - 0s 117ms/step - loss: 6.2836e-
05 - categorical_accuracy: 1.0000
Epoch 335/1000
3/3 [==============================] - 0s 122ms/step - loss: 6.2252e-
05 - categorical_accuracy: 1.0000
Epoch 336/1000
3/3 [==============================] - 0s 126ms/step - loss: 6.1619e-
05 - categorical_accuracy: 1.0000
Epoch 337/1000
3/3 [==============================] - 0s 119ms/step - loss: 6.0825e-
05 - categorical_accuracy: 1.0000
Epoch 338/1000
3/3 [==============================] - 0s 122ms/step - loss: 6.0164e-
05 - categorical_accuracy: 1.0000
Epoch 339/1000
3/3 [==============================] - 0s 119ms/step - loss: 5.9428e-
05 - categorical_accuracy: 1.0000
Epoch 340/1000
3/3 [==============================] - 0s 113ms/step - loss: 5.8766e-
05 - categorical_accuracy: 1.0000
Epoch 341/1000
3/3 [==============================] - 0s 126ms/step - loss: 5.8204e-
05 - categorical_accuracy: 1.0000
Epoch 342/1000
3/3 [==============================] - 0s 110ms/step - loss: 5.7527e-
05 - categorical_accuracy: 1.0000
Epoch 343/1000
3/3 [==============================] - 0s 116ms/step - loss: 5.6908e-
05 - categorical_accuracy: 1.0000
Epoch 344/1000
3/3 [==============================] - 0s 121ms/step - loss: 5.6448e-
05 - categorical_accuracy: 1.0000
Epoch 345/1000
3/3 [==============================] - 0s 135ms/step - loss: 5.5902e-
05 - categorical_accuracy: 1.0000
Epoch 346/1000
3/3 [==============================] - 0s 149ms/step - loss: 5.5193e-
05 - categorical_accuracy: 1.0000
Epoch 347/1000
3/3 [==============================] - 0s 138ms/step - loss: 5.4623e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 348/1000
3/3 [==============================] - 0s 114ms/step - loss: 5.4098e-
05 - categorical_accuracy: 1.0000
Epoch 349/1000
3/3 [==============================] - 1s 213ms/step - loss: 5.3526e-
05 - categorical_accuracy: 1.0000
Epoch 350/1000
3/3 [==============================] - 1s 195ms/step - loss: 5.3007e-
05 - categorical_accuracy: 1.0000
Epoch 351/1000
3/3 [==============================] - 1s 201ms/step - loss: 5.2507e-
05 - categorical_accuracy: 1.0000
Epoch 352/1000
3/3 [==============================] - 1s 236ms/step - loss: 5.1967e-
05 - categorical_accuracy: 1.0000
Epoch 353/1000
3/3 [==============================] - 1s 264ms/step - loss: 5.1474e-
05 - categorical_accuracy: 1.0000
Epoch 354/1000
3/3 [==============================] - 1s 175ms/step - loss: 5.0989e-
05 - categorical_accuracy: 1.0000
Epoch 355/1000
3/3 [==============================] - 0s 118ms/step - loss: 5.0545e-
05 - categorical_accuracy: 1.0000
Epoch 356/1000
3/3 [==============================] - 0s 112ms/step - loss: 5.0053e-
05 - categorical_accuracy: 1.0000
Epoch 357/1000
3/3 [==============================] - 0s 110ms/step - loss: 4.9520e-
05 - categorical_accuracy: 1.0000
Epoch 358/1000
3/3 [==============================] - 0s 135ms/step - loss: 4.9054e-
05 - categorical_accuracy: 1.0000
Epoch 359/1000
3/3 [==============================] - 0s 115ms/step - loss: 4.8612e-
05 - categorical_accuracy: 1.0000
Epoch 360/1000
3/3 [==============================] - 0s 116ms/step - loss: 4.8143e-
05 - categorical_accuracy: 1.0000
Epoch 361/1000
3/3 [==============================] - 0s 136ms/step - loss: 4.7713e-
05 - categorical_accuracy: 1.0000
Epoch 362/1000
3/3 [==============================] - 0s 116ms/step - loss: 4.7286e-
05 - categorical_accuracy: 1.0000
Epoch 363/1000
3/3 [==============================] - 0s 114ms/step - loss: 4.6847e-
05 - categorical_accuracy: 1.0000
Epoch 364/1000
```

```
3/3 [==============================] - 0s 128ms/step - loss: 4.6456e-
05 - categorical_accuracy: 1.0000
Epoch 365/1000
3/3 [==============================] - 0s 124ms/step - loss: 4.6085e-
05 - categorical_accuracy: 1.0000
Epoch 366/1000
3/3 [==============================] - 0s 118ms/step - loss: 4.5662e-
05 - categorical_accuracy: 1.0000
Epoch 367/1000
3/3 [==============================] - 0s 122ms/step - loss: 4.5261e-
05 - categorical_accuracy: 1.0000
Epoch 368/1000
3/3 [==============================] - 0s 115ms/step - loss: 4.4893e-
05 - categorical_accuracy: 1.0000
Epoch 369/1000
3/3 [==============================] - 0s 119ms/step - loss: 4.4499e-
05 - categorical_accuracy: 1.0000
Epoch 370/1000
3/3 [==============================] - 0s 121ms/step - loss: 4.4150e-
05 - categorical_accuracy: 1.0000
Epoch 371/1000
3/3 [==============================] - 0s 117ms/step - loss: 4.3737e-
05 - categorical_accuracy: 1.0000
Epoch 372/1000
3/3 [==============================] - 0s 124ms/step - loss: 4.3389e-
05 - categorical_accuracy: 1.0000
Epoch 373/1000
3/3 [==============================] - 0s 129ms/step - loss: 4.3026e-
05 - categorical_accuracy: 1.0000
Epoch 374/1000
3/3 [==============================] - 0s 128ms/step - loss: 4.2685e-
05 - categorical_accuracy: 1.0000
Epoch 375/1000
3/3 [==============================] - 0s 133ms/step - loss: 4.2334e-
05 - categorical_accuracy: 1.0000
Epoch 376/1000
3/3 [==============================] - 0s 119ms/step - loss: 4.1992e-
05 - categorical_accuracy: 1.0000
Epoch 377/1000
3/3 [==============================] - 0s 134ms/step - loss: 4.1633e-
05 - categorical_accuracy: 1.0000
Epoch 378/1000
3/3 [==============================] - 0s 126ms/step - loss: 4.1333e-
05 - categorical_accuracy: 1.0000
Epoch 379/1000
3/3 [==============================] - 0s 114ms/step - loss: 4.0984e-
05 - categorical_accuracy: 1.0000
Epoch 380/1000
3/3 [==============================] - 0s 124ms/step - loss: 4.0717e-
```

```
05 - categorical_accuracy: 1.0000
Epoch 381/1000
3/3 [==============================] - 0s 168ms/step - loss: 4.0392e-
05 - categorical_accuracy: 1.0000
Epoch 382/1000
3/3 [==============================] - 1s 258ms/step - loss: 4.0072e-
05 - categorical_accuracy: 1.0000
Epoch 383/1000
3/3 [==============================] - 1s 231ms/step - loss: 3.9758e-
05 - categorical_accuracy: 1.0000
Epoch 384/1000
3/3 [==============================] - 1s 217ms/step - loss: 3.9466e-
05 - categorical_accuracy: 1.0000
Epoch 385/1000
3/3 [==============================] - 1s 256ms/step - loss: 3.9162e-
05 - categorical_accuracy: 1.0000
Epoch 386/1000
3/3 [==============================] - 1s 127ms/step - loss: 3.8879e-
05 - categorical_accuracy: 1.0000
Epoch 387/1000
3/3 [==============================] - 0s 124ms/step - loss: 3.8573e-
05 - categorical_accuracy: 1.0000
Epoch 388/1000
3/3 [==============================] - 0s 121ms/step - loss: 3.8274e-
05 - categorical_accuracy: 1.0000
Epoch 389/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.7993e-
05 - categorical_accuracy: 1.0000
Epoch 390/1000
3/3 [==============================] - 0s 117ms/step - loss: 3.7715e-
05 - categorical_accuracy: 1.0000
Epoch 391/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.7465e-
05 - categorical_accuracy: 1.0000
Epoch 392/1000
3/3 [==============================] - 0s 121ms/step - loss: 3.7179e-
05 - categorical_accuracy: 1.0000
Epoch 393/1000
3/3 [==============================] - 0s 120ms/step - loss: 3.6887e-
05 - categorical_accuracy: 1.0000
Epoch 394/1000
3/3 [==============================] - 0s 117ms/step - loss: 3.6693e-
05 - categorical_accuracy: 1.0000
Epoch 395/1000
3/3 [==============================] - 0s 119ms/step - loss: 3.6413e-
05 - categorical_accuracy: 1.0000
Epoch 396/1000
3/3 [==============================] - 0s 125ms/step - loss: 3.6176e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 397/1000
3/3 [==============================] - 0s 120ms/step - loss: 3.5895e-
05 - categorical_accuracy: 1.0000
Epoch 398/1000
3/3 [==============================] - 0s 114ms/step - loss: 3.5663e-
05 - categorical_accuracy: 1.0000
Epoch 399/1000
3/3 [==============================] - 0s 115ms/step - loss: 3.5412e-
05 - categorical_accuracy: 1.0000
Epoch 400/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.5178e-
05 - categorical_accuracy: 1.0000
Epoch 401/1000
3/3 [==============================] - 0s 112ms/step - loss: 3.4934e-
05 - categorical_accuracy: 1.0000
Epoch 402/1000
3/3 [==============================] - 0s 122ms/step - loss: 3.4819e-
05 - categorical_accuracy: 1.0000
Epoch 403/1000
3/3 [==============================] - 0s 119ms/step - loss: 3.4526e-
05 - categorical_accuracy: 1.0000
Epoch 404/1000
3/3 [==============================] - 0s 126ms/step - loss: 3.4383e-
05 - categorical_accuracy: 1.0000
Epoch 405/1000
3/3 [==============================] - 0s 115ms/step - loss: 3.4086e-
05 - categorical_accuracy: 1.0000
Epoch 406/1000
3/3 [==============================] - 0s 136ms/step - loss: 3.4060e-
05 - categorical_accuracy: 1.0000
Epoch 407/1000
3/3 [==============================] - 0s 122ms/step - loss: 3.4231e-
05 - categorical_accuracy: 1.0000
Epoch 408/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.3853e-
05 - categorical_accuracy: 1.0000
Epoch 409/1000
3/3 [==============================] - 0s 116ms/step - loss: 3.3516e-
05 - categorical_accuracy: 1.0000
Epoch 410/1000
3/3 [==============================] - 0s 134ms/step - loss: 3.3349e-
05 - categorical_accuracy: 1.0000
Epoch 411/1000
3/3 [==============================] - 0s 131ms/step - loss: 3.3129e-
05 - categorical_accuracy: 1.0000
Epoch 412/1000
3/3 [==============================] - 0s 114ms/step - loss: 3.3063e-
05 - categorical_accuracy: 1.0000
Epoch 413/1000
```

```
3/3 [==============================] - 0s 186ms/step - loss: 3.2981e-
05 - categorical_accuracy: 1.0000
Epoch 414/1000
3/3 [==============================] - 1s 639ms/step - loss: 3.2724e-
05 - categorical_accuracy: 1.0000
Epoch 415/1000
3/3 [==============================] - 1s 266ms/step - loss: 3.2577e-
05 - categorical_accuracy: 1.0000
Epoch 416/1000
3/3 [==============================] - 1s 257ms/step - loss: 3.2194e-
05 - categorical_accuracy: 1.0000
Epoch 417/1000
3/3 [==============================] - 0s 131ms/step - loss: 3.1888e-
05 - categorical_accuracy: 1.0000
Epoch 418/1000
3/3 [==============================] - 0s 116ms/step - loss: 3.1723e-
05 - categorical_accuracy: 1.0000
Epoch 419/1000
3/3 [==============================] - 0s 119ms/step - loss: 3.1758e-
05 - categorical_accuracy: 1.0000
Epoch 420/1000
3/3 [==============================] - 0s 114ms/step - loss: 3.1497e-
05 - categorical_accuracy: 1.0000
Epoch 421/1000
3/3 [==============================] - 0s 126ms/step - loss: 3.1059e-
05 - categorical_accuracy: 1.0000
Epoch 422/1000
3/3 [==============================] - 0s 113ms/step - loss: 3.0773e-
05 - categorical_accuracy: 1.0000
Epoch 423/1000
3/3 [==============================] - 0s 118ms/step - loss: 3.0668e-
05 - categorical_accuracy: 1.0000
Epoch 424/1000
3/3 [==============================] - 0s 129ms/step - loss: 3.0423e-
05 - categorical_accuracy: 1.0000
Epoch 425/1000
3/3 [==============================] - 0s 120ms/step - loss: 3.0075e-
05 - categorical_accuracy: 1.0000
Epoch 426/1000
3/3 [==============================] - 0s 114ms/step - loss: 2.9897e-
05 - categorical_accuracy: 1.0000
Epoch 427/1000
3/3 [==============================] - 0s 129ms/step - loss: 2.9779e-
05 - categorical_accuracy: 1.0000
Epoch 428/1000
3/3 [==============================] - 0s 117ms/step - loss: 2.9609e-
05 - categorical_accuracy: 1.0000
Epoch 429/1000
3/3 [==============================] - 0s 123ms/step - loss: 2.9353e-
```

```
05 - categorical_accuracy: 1.0000
Epoch 430/1000
3/3 [==============================] - 0s 137ms/step - loss: 2.9175e-
05 - categorical_accuracy: 1.0000
Epoch 431/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.9027e-
05 - categorical_accuracy: 1.0000
Epoch 432/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.8902e-
05 - categorical_accuracy: 1.0000
Epoch 433/1000
3/3 [==============================] - 0s 131ms/step - loss: 2.8703e-
05 - categorical_accuracy: 1.0000
Epoch 434/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.8559e-
05 - categorical_accuracy: 1.0000
Epoch 435/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.8375e-
05 - categorical_accuracy: 1.0000
Epoch 436/1000
3/3 [==============================] - 0s 118ms/step - loss: 2.8096e-
05 - categorical_accuracy: 1.0000
Epoch 437/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.7939e-
05 - categorical_accuracy: 1.0000
Epoch 438/1000
3/3 [==============================] - 0s 128ms/step - loss: 2.7814e-
05 - categorical_accuracy: 1.0000
Epoch 439/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.7666e-
05 - categorical_accuracy: 1.0000
Epoch 440/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.7501e-
05 - categorical_accuracy: 1.0000
Epoch 441/1000
3/3 [==============================] - 0s 123ms/step - loss: 2.7347e-
05 - categorical_accuracy: 1.0000
Epoch 442/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.7171e-
05 - categorical_accuracy: 1.0000
Epoch 443/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.7012e-
05 - categorical_accuracy: 1.0000
Epoch 444/1000
3/3 [==============================] - 0s 166ms/step - loss: 2.6997e-
05 - categorical_accuracy: 1.0000
Epoch 445/1000
3/3 [==============================] - 1s 213ms/step - loss: 2.6884e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 446/1000
3/3 [==============================] - 1s 222ms/step - loss: 2.6757e-
05 - categorical_accuracy: 1.0000
Epoch 447/1000
3/3 [==============================] - 1s 253ms/step - loss: 2.6741e-
05 - categorical_accuracy: 1.0000
Epoch 448/1000
3/3 [==============================] - 1s 239ms/step - loss: 2.6277e-
05 - categorical_accuracy: 1.0000
Epoch 449/1000
3/3 [==============================] - 1s 160ms/step - loss: 2.6138e-
05 - categorical_accuracy: 1.0000
Epoch 450/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.5987e-
05 - categorical_accuracy: 1.0000
Epoch 451/1000
3/3 [==============================] - 0s 124ms/step - loss: 2.5827e-
05 - categorical_accuracy: 1.0000
Epoch 452/1000
3/3 [==============================] - 0s 115ms/step - loss: 2.5746e-
05 - categorical_accuracy: 1.0000
Epoch 453/1000
3/3 [==============================] - 0s 123ms/step - loss: 2.5570e-
05 - categorical_accuracy: 1.0000
Epoch 454/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.5425e-
05 - categorical_accuracy: 1.0000
Epoch 455/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.5249e-
05 - categorical_accuracy: 1.0000
Epoch 456/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.5161e-
05 - categorical_accuracy: 1.0000
Epoch 457/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.5141e-
05 - categorical_accuracy: 1.0000
Epoch 458/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.4938e-
05 - categorical_accuracy: 1.0000
Epoch 459/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.4963e-
05 - categorical_accuracy: 1.0000
Epoch 460/1000
3/3 [==============================] - 0s 123ms/step - loss: 2.4636e-
05 - categorical_accuracy: 1.0000
Epoch 461/1000
3/3 [==============================] - 0s 126ms/step - loss: 2.4454e-
05 - categorical_accuracy: 1.0000
Epoch 462/1000
```

```
3/3 [==============================] - 0s 122ms/step - loss: 2.4380e-
05 - categorical_accuracy: 1.0000
Epoch 463/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.4279e-
05 - categorical_accuracy: 1.0000
Epoch 464/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.4175e-
05 - categorical_accuracy: 1.0000
Epoch 465/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.3986e-
05 - categorical_accuracy: 1.0000
Epoch 466/1000
3/3 [==============================] - 0s 113ms/step - loss: 2.4043e-
05 - categorical_accuracy: 1.0000
Epoch 467/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.3749e-
05 - categorical_accuracy: 1.0000
Epoch 468/1000
3/3 [==============================] - 0s 113ms/step - loss: 2.3691e-
05 - categorical_accuracy: 1.0000
Epoch 469/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.3550e-
05 - categorical_accuracy: 1.0000
Epoch 470/1000
3/3 [==============================] - 0s 116ms/step - loss: 2.3451e-
05 - categorical_accuracy: 1.0000
Epoch 471/1000
3/3 [==============================] - 0s 126ms/step - loss: 2.3290e-
05 - categorical_accuracy: 1.0000
Epoch 472/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.3122e-
05 - categorical_accuracy: 1.0000
Epoch 473/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.3010e-
05 - categorical_accuracy: 1.0000
Epoch 474/1000
3/3 [==============================] - 0s 125ms/step - loss: 2.2899e-
05 - categorical_accuracy: 1.0000
Epoch 475/1000
3/3 [==============================] - 0s 123ms/step - loss: 2.2777e-
05 - categorical_accuracy: 1.0000
Epoch 476/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.2719e-
05 - categorical_accuracy: 1.0000
Epoch 477/1000
3/3 [==============================] - 1s 209ms/step - loss: 2.2614e-
05 - categorical_accuracy: 1.0000
Epoch 478/1000
3/3 [==============================] - 1s 223ms/step - loss: 2.2516e-
```

```
05 - categorical_accuracy: 1.0000
Epoch 479/1000
3/3 [==============================] - 1s 221ms/step - loss: 2.2425e-
05 - categorical_accuracy: 1.0000
Epoch 480/1000
3/3 [==============================] - 1s 255ms/step - loss: 2.2325e-
05 - categorical_accuracy: 1.0000
Epoch 481/1000
3/3 [==============================] - 1s 232ms/step - loss: 2.2230e-
05 - categorical_accuracy: 1.0000
Epoch 482/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.2087e-
05 - categorical_accuracy: 1.0000
Epoch 483/1000
3/3 [==============================] - 0s 136ms/step - loss: 2.2021e-
05 - categorical_accuracy: 1.0000
Epoch 484/1000
3/3 [==============================] - 0s 129ms/step - loss: 2.1956e-
05 - categorical_accuracy: 1.0000
Epoch 485/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.1815e-
05 - categorical_accuracy: 1.0000
Epoch 486/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.1707e-
05 - categorical_accuracy: 1.0000
Epoch 487/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.1635e-
05 - categorical_accuracy: 1.0000
Epoch 488/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.1533e-
05 - categorical_accuracy: 1.0000
Epoch 489/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.1449e-
05 - categorical_accuracy: 1.0000
Epoch 490/1000
3/3 [==============================] - 0s 134ms/step - loss: 2.1363e-
05 - categorical_accuracy: 1.0000
Epoch 491/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.1265e-
05 - categorical_accuracy: 1.0000
Epoch 492/1000
3/3 [==============================] - 0s 114ms/step - loss: 2.1187e-
05 - categorical_accuracy: 1.0000
Epoch 493/1000
3/3 [==============================] - 0s 119ms/step - loss: 2.1093e-
05 - categorical_accuracy: 1.0000
Epoch 494/1000
3/3 [==============================] - 0s 118ms/step - loss: 2.1011e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 495/1000
3/3 [==============================] - 0s 129ms/step - loss: 2.0895e-
05 - categorical_accuracy: 1.0000
Epoch 496/1000
3/3 [==============================] - 0s 121ms/step - loss: 2.0836e-
05 - categorical_accuracy: 1.0000
Epoch 497/1000
3/3 [==============================] - 0s 133ms/step - loss: 2.0753e-
05 - categorical_accuracy: 1.0000
Epoch 498/1000
3/3 [==============================] - 0s 134ms/step - loss: 2.0668e-
05 - categorical_accuracy: 1.0000
Epoch 499/1000
3/3 [==============================] - 0s 138ms/step - loss: 2.0541e-
05 - categorical_accuracy: 1.0000
Epoch 500/1000
3/3 [==============================] - 0s 125ms/step - loss: 2.0464e-
05 - categorical_accuracy: 1.0000
Epoch 501/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.0387e-
05 - categorical_accuracy: 1.0000
Epoch 502/1000
3/3 [==============================] - 0s 115ms/step - loss: 2.0303e-
05 - categorical_accuracy: 1.0000
Epoch 503/1000
3/3 [==============================] - 0s 125ms/step - loss: 2.0220e-
05 - categorical_accuracy: 1.0000
Epoch 504/1000
3/3 [==============================] - 0s 125ms/step - loss: 2.0167e-
05 - categorical_accuracy: 1.0000
Epoch 505/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.0030e-
05 - categorical_accuracy: 1.0000
Epoch 506/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.9961e-
05 - categorical_accuracy: 1.0000
Epoch 507/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.9894e-
05 - categorical_accuracy: 1.0000
Epoch 508/1000
3/3 [==============================] - 0s 166ms/step - loss: 1.9882e-
05 - categorical_accuracy: 1.0000
Epoch 509/1000
3/3 [==============================] - 1s 199ms/step - loss: 1.9791e-
05 - categorical_accuracy: 1.0000
Epoch 510/1000
3/3 [==============================] - 1s 185ms/step - loss: 1.9704e-
05 - categorical_accuracy: 1.0000
Epoch 511/1000
```

```
3/3 [==============================] - 1s 221ms/step - loss: 1.9649e-
05 - categorical_accuracy: 1.0000
Epoch 512/1000
3/3 [==============================] - 1s 247ms/step - loss: 1.9558e-
05 - categorical_accuracy: 1.0000
Epoch 513/1000
3/3 [==============================] - 1s 245ms/step - loss: 1.9469e-
05 - categorical_accuracy: 1.0000
Epoch 514/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.9400e-
05 - categorical_accuracy: 1.0000
Epoch 515/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.9297e-
05 - categorical_accuracy: 1.0000
Epoch 516/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.9232e-
05 - categorical_accuracy: 1.0000
Epoch 517/1000
3/3 [==============================] - 0s 125ms/step - loss: 1.9177e-
05 - categorical_accuracy: 1.0000
Epoch 518/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.9075e-
05 - categorical_accuracy: 1.0000
Epoch 519/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.9030e-
05 - categorical_accuracy: 1.0000
Epoch 520/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.8937e-
05 - categorical_accuracy: 1.0000
Epoch 521/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.8840e-
05 - categorical_accuracy: 1.0000
Epoch 522/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.8765e-
05 - categorical_accuracy: 1.0000
Epoch 523/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.8672e-
05 - categorical_accuracy: 1.0000
Epoch 524/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.8619e-
05 - categorical_accuracy: 1.0000
Epoch 525/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.8554e-
05 - categorical_accuracy: 1.0000
Epoch 526/1000
3/3 [==============================] - 0s 127ms/step - loss: 1.8484e-
05 - categorical_accuracy: 1.0000
Epoch 527/1000
3/3 [==============================] - 0s 127ms/step - loss: 1.8424e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 528/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.8338e-
05 - categorical_accuracy: 1.0000
Epoch 529/1000
3/3 [==============================] - 0s 115ms/step - loss: 1.8281e-
05 - categorical_accuracy: 1.0000
Epoch 530/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.8191e-
05 - categorical_accuracy: 1.0000
Epoch 531/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.8128e-
05 - categorical_accuracy: 1.0000
Epoch 532/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.8089e-
05 - categorical_accuracy: 1.0000
Epoch 533/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.8027e-
05 - categorical_accuracy: 1.0000
Epoch 534/1000
3/3 [==============================] - 0s 137ms/step - loss: 1.7939e-
05 - categorical_accuracy: 1.0000
Epoch 535/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.7888e-
05 - categorical_accuracy: 1.0000
Epoch 536/1000
3/3 [==============================] - 0s 111ms/step - loss: 1.7835e-
05 - categorical_accuracy: 1.0000
Epoch 537/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.7772e-
05 - categorical_accuracy: 1.0000
Epoch 538/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.7675e-
05 - categorical_accuracy: 1.0000
Epoch 539/1000
3/3 [==============================] - 0s 139ms/step - loss: 1.7615e-
05 - categorical_accuracy: 1.0000
Epoch 540/1000
3/3 [==============================] - 0s 129ms/step - loss: 1.7588e-
05 - categorical_accuracy: 1.0000
Epoch 541/1000
3/3 [==============================] - 1s 222ms/step - loss: 1.7490e-
05 - categorical_accuracy: 1.0000
Epoch 542/1000
3/3 [==============================] - 1s 235ms/step - loss: 1.7399e-
05 - categorical_accuracy: 1.0000
Epoch 543/1000
3/3 [==============================] - 1s 228ms/step - loss: 1.7348e-
05 - categorical_accuracy: 1.0000
Epoch 544/1000
```

```
3/3 [==============================] - 1s 217ms/step - loss: 1.7272e-
05 - categorical_accuracy: 1.0000
Epoch 545/1000
3/3 [==============================] - 1s 141ms/step - loss: 1.7192e-
05 - categorical_accuracy: 1.0000
Epoch 546/1000
3/3 [==============================] - 0s 132ms/step - loss: 1.7129e-
05 - categorical_accuracy: 1.0000
Epoch 547/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.7076e-
05 - categorical_accuracy: 1.0000
Epoch 548/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.7041e-
05 - categorical_accuracy: 1.0000
Epoch 549/1000
3/3 [==============================] - 0s 137ms/step - loss: 1.6950e-
05 - categorical_accuracy: 1.0000
Epoch 550/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.6915e-
05 - categorical_accuracy: 1.0000
Epoch 551/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.6817e-
05 - categorical_accuracy: 1.0000
Epoch 552/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.6749e-
05 - categorical_accuracy: 1.0000
Epoch 553/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.6697e-
05 - categorical_accuracy: 1.0000
Epoch 554/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.6653e-
05 - categorical_accuracy: 1.0000
Epoch 555/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.6554e-
05 - categorical_accuracy: 1.0000
Epoch 556/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.6507e-
05 - categorical_accuracy: 1.0000
Epoch 557/1000
3/3 [==============================] - 0s 140ms/step - loss: 1.6434e-
05 - categorical_accuracy: 1.0000
Epoch 558/1000
3/3 [==============================] - 0s 115ms/step - loss: 1.6345e-
05 - categorical_accuracy: 1.0000
Epoch 559/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.6308e-
05 - categorical_accuracy: 1.0000
Epoch 560/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.6218e-
```

```
05 - categorical_accuracy: 1.0000
Epoch 561/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.6165e-
05 - categorical_accuracy: 1.0000
Epoch 562/1000
3/3 [==============================] - 0s 156ms/step - loss: 1.6125e-
05 - categorical_accuracy: 1.0000
Epoch 563/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.6044e-
05 - categorical_accuracy: 1.0000
Epoch 564/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.6009e-
05 - categorical_accuracy: 1.0000
Epoch 565/1000
3/3 [==============================] - 0s 134ms/step - loss: 1.5940e-
05 - categorical_accuracy: 1.0000
Epoch 566/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.5862e-
05 - categorical_accuracy: 1.0000
Epoch 567/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.5794e-
05 - categorical_accuracy: 1.0000
Epoch 568/1000
3/3 [==============================] - 0s 127ms/step - loss: 1.5755e-
05 - categorical_accuracy: 1.0000
Epoch 569/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.5703e-
05 - categorical_accuracy: 1.0000
Epoch 570/1000
3/3 [==============================] - 0s 129ms/step - loss: 1.5627e-
05 - categorical_accuracy: 1.0000
Epoch 571/1000
3/3 [==============================] - 0s 131ms/step - loss: 1.5641e-
05 - categorical_accuracy: 1.0000
Epoch 572/1000
3/3 [==============================] - 1s 229ms/step - loss: 1.5557e-
05 - categorical_accuracy: 1.0000
Epoch 573/1000
3/3 [==============================] - 1s 203ms/step - loss: 1.5494e-
05 - categorical_accuracy: 1.0000
Epoch 574/1000
3/3 [==============================] - 1s 227ms/step - loss: 1.5441e-
05 - categorical_accuracy: 1.0000
Epoch 575/1000
3/3 [==============================] - 1s 252ms/step - loss: 1.5407e-
05 - categorical_accuracy: 1.0000
Epoch 576/1000
3/3 [==============================] - 1s 209ms/step - loss: 1.5357e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 577/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.5303e-
05 - categorical_accuracy: 1.0000
Epoch 578/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.5221e-
05 - categorical_accuracy: 1.0000
Epoch 579/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.5176e-
05 - categorical_accuracy: 1.0000
Epoch 580/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.5114e-
05 - categorical_accuracy: 1.0000
Epoch 581/1000
3/3 [==============================] - 0s 132ms/step - loss: 1.5038e-
05 - categorical_accuracy: 1.0000
Epoch 582/1000
3/3 [==============================] - 0s 127ms/step - loss: 1.4972e-
05 - categorical_accuracy: 1.0000
Epoch 583/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.4935e-
05 - categorical_accuracy: 1.0000
Epoch 584/1000
3/3 [==============================] - 0s 114ms/step - loss: 1.4891e-
05 - categorical_accuracy: 1.0000
Epoch 585/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.4821e-
05 - categorical_accuracy: 1.0000
Epoch 586/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.4764e-
05 - categorical_accuracy: 1.0000
Epoch 587/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.4716e-
05 - categorical_accuracy: 1.0000
Epoch 588/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.4670e-
05 - categorical_accuracy: 1.0000
Epoch 589/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.4606e-
05 - categorical_accuracy: 1.0000
Epoch 590/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.4548e-
05 - categorical_accuracy: 1.0000
Epoch 591/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.4503e-
05 - categorical_accuracy: 1.0000
Epoch 592/1000
3/3 [==============================] - 0s 139ms/step - loss: 1.4451e-
05 - categorical_accuracy: 1.0000
Epoch 593/1000
```

```
3/3 [==============================] - 0s 123ms/step - loss: 1.4374e-
05 - categorical_accuracy: 1.0000
Epoch 594/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.4325e-
05 - categorical_accuracy: 1.0000
Epoch 595/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.4284e-
05 - categorical_accuracy: 1.0000
Epoch 596/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.4222e-
05 - categorical_accuracy: 1.0000
Epoch 597/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.4152e-
05 - categorical_accuracy: 1.0000
Epoch 598/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.4153e-
05 - categorical_accuracy: 1.0000
Epoch 599/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.4089e-
05 - categorical_accuracy: 1.0000
Epoch 600/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.4026e-
05 - categorical_accuracy: 1.0000
Epoch 601/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.3987e-
05 - categorical_accuracy: 1.0000
Epoch 602/1000
3/3 [==============================] - 0s 124ms/step - loss: 1.3911e-
05 - categorical_accuracy: 1.0000
Epoch 603/1000
3/3 [==============================] - 0s 164ms/step - loss: 1.3855e-
05 - categorical_accuracy: 1.0000
Epoch 604/1000
3/3 [==============================] - 1s 202ms/step - loss: 1.3824e-
05 - categorical_accuracy: 1.0000
Epoch 605/1000
3/3 [==============================] - 1s 259ms/step - loss: 1.3744e-
05 - categorical_accuracy: 1.0000
Epoch 606/1000
3/3 [==============================] - 1s 241ms/step - loss: 1.3692e-
05 - categorical_accuracy: 1.0000
Epoch 607/1000
3/3 [==============================] - 1s 248ms/step - loss: 1.3636e-
05 - categorical_accuracy: 1.0000
Epoch 608/1000
3/3 [==============================] - 1s 185ms/step - loss: 1.3587e-
05 - categorical_accuracy: 1.0000
Epoch 609/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.3520e-
```

```
05 - categorical_accuracy: 1.0000
Epoch 610/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.3468e-
05 - categorical_accuracy: 1.0000
Epoch 611/1000
3/3 [==============================] - 0s 120ms/step - loss: 1.3406e-
05 - categorical_accuracy: 1.0000
Epoch 612/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.3353e-
05 - categorical_accuracy: 1.0000
Epoch 613/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.3316e-
05 - categorical_accuracy: 1.0000
Epoch 614/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.3238e-
05 - categorical_accuracy: 1.0000
Epoch 615/1000
3/3 [==============================] - 0s 130ms/step - loss: 1.3183e-
05 - categorical_accuracy: 1.0000
Epoch 616/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.3148e-
05 - categorical_accuracy: 1.0000
Epoch 617/1000
3/3 [==============================] - 0s 117ms/step - loss: 1.3085e-
05 - categorical_accuracy: 1.0000
Epoch 618/1000
3/3 [==============================] - 0s 146ms/step - loss: 1.3023e-
05 - categorical_accuracy: 1.0000
Epoch 619/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.2987e-
05 - categorical_accuracy: 1.0000
Epoch 620/1000
3/3 [==============================] - 0s 121ms/step - loss: 1.2942e-
05 - categorical_accuracy: 1.0000
Epoch 621/1000
3/3 [==============================] - 0s 123ms/step - loss: 1.2900e-
05 - categorical_accuracy: 1.0000
Epoch 622/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.2860e-
05 - categorical_accuracy: 1.0000
Epoch 623/1000
3/3 [==============================] - 0s 139ms/step - loss: 1.2897e-
05 - categorical_accuracy: 1.0000
Epoch 624/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.4001e-
05 - categorical_accuracy: 1.0000
Epoch 625/1000
3/3 [==============================] - 0s 120ms/step - loss: 2.1241e-
05 - categorical_accuracy: 1.0000
```

```
Epoch 626/1000
3/3 [==============================] - 0s 124ms/step - loss: 5.6413 -
categorical_accuracy: 0.8235
Epoch 627/1000
3/3 [==============================] - 0s 116ms/step - loss: 9.5047 -
categorical_accuracy: 0.2824
Epoch 628/1000
3/3 [==============================] - 0s 128ms/step - loss: 6.5344 -
categorical_accuracy: 0.3647
Epoch 629/1000
3/3 [==============================] - 0s 138ms/step - loss: 6.3492 -
categorical_accuracy: 0.2824
Epoch 630/1000
3/3 [==============================] - 0s 122ms/step - loss: 2.5020 -
categorical_accuracy: 0.4235
Epoch 631/1000
3/3 [==============================] - 0s 134ms/step - loss: 3.9034 -
categorical_accuracy: 0.2471
Epoch 632/1000
3/3 [==============================] - 0s 124ms/step - loss: 5.0925 -
categorical_accuracy: 0.2118
Epoch 633/1000
3/3 [==============================] - 0s 115ms/step - loss: 3.2650 -
categorical_accuracy: 0.3294
Epoch 634/1000
3/3 [==============================] - 0s 156ms/step - loss: 6.2132 -
categorical_accuracy: 0.3294
Epoch 635/1000
3/3 [==============================] - 1s 211ms/step - loss: 2.7572 -
categorical_accuracy: 0.4588
Epoch 636/1000
3/3 [==============================] - 1s 221ms/step - loss: 3.7301 -
categorical_accuracy: 0.3647
Epoch 637/1000
3/3 [==============================] - 1s 182ms/step - loss: 3.4087 -
categorical_accuracy: 0.3176
Epoch 638/1000
3/3 [==============================] - 1s 230ms/step - loss: 2.8949 -
categorical_accuracy: 0.3529
Epoch 639/1000
3/3 [==============================] - 1s 254ms/step - loss: 1.7054 -
categorical_accuracy: 0.1882
Epoch 640/1000
3/3 [==============================] - 0s 128ms/step - loss: 2.0489 -
categorical_accuracy: 0.3294
Epoch 641/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.5560 -
categorical_accuracy: 0.4353
Epoch 642/1000
```

```
3/3 [==============================] - 0s 119ms/step - loss: 1.4283 -
categorical_accuracy: 0.4118
Epoch 643/1000
3/3 [==============================] - 0s 133ms/step - loss: 1.2565 -
categorical_accuracy: 0.3176
Epoch 644/1000
3/3 [==============================] - 0s 125ms/step - loss: 1.1366 -
categorical_accuracy: 0.3412
Epoch 645/1000
3/3 [==============================] - 0s 126ms/step - loss: 1.2942 -
categorical_accuracy: 0.2706
Epoch 646/1000
3/3 [==============================] - 0s 128ms/step - loss: 1.2441 -
categorical_accuracy: 0.2706
Epoch 647/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.1960 -
categorical_accuracy: 0.4235
Epoch 648/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.9580 -
categorical_accuracy: 0.4941
Epoch 649/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.9676 -
categorical_accuracy: 0.3059
Epoch 650/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.9287 -
categorical_accuracy: 0.4471
Epoch 651/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.8997 -
categorical_accuracy: 0.5647
Epoch 652/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.8782 -
categorical_accuracy: 0.5412
Epoch 653/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.8870 -
categorical_accuracy: 0.6706
Epoch 654/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.8765 -
categorical_accuracy: 0.5412
Epoch 655/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.8218 -
categorical_accuracy: 0.5059
Epoch 656/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.8544 -
categorical_accuracy: 0.5294
Epoch 657/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.8658 -
categorical_accuracy: 0.5176
Epoch 658/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.8428 -
```

```
categorical_accuracy: 0.5412
Epoch 659/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.7956 -
categorical_accuracy: 0.6706
Epoch 660/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.7443 -
categorical_accuracy: 0.5176
Epoch 661/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.7533 -
categorical_accuracy: 0.6118
Epoch 662/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.6950 -
categorical_accuracy: 0.6471
Epoch 663/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.6610 -
categorical_accuracy: 0.6471
Epoch 664/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.6348 -
categorical_accuracy: 0.6706
Epoch 665/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.6235 -
categorical_accuracy: 0.5529
Epoch 666/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.6056 -
categorical_accuracy: 0.6706
Epoch 667/1000
3/3 [==============================] - 1s 531ms/step - loss: 0.5926 -
categorical_accuracy: 0.6353
Epoch 668/1000
3/3 [==============================] - 1s 202ms/step - loss: 0.5624 -
categorical_accuracy: 0.6471
Epoch 669/1000
3/3 [==============================] - 1s 248ms/step - loss: 0.5492 -
categorical_accuracy: 0.6706
Epoch 670/1000
3/3 [==============================] - 1s 227ms/step - loss: 0.5152 -
categorical_accuracy: 0.6706
Epoch 671/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.3897 -
categorical_accuracy: 0.7765
Epoch 672/1000
3/3 [==============================] - 0s 138ms/step - loss: 0.1885 -
categorical_accuracy: 1.0000
Epoch 673/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.1099 -
categorical_accuracy: 1.0000
Epoch 674/1000
3/3 [==============================] - 0s 123ms/step - loss: 6.8027 -
categorical_accuracy: 0.6471
```

```
Epoch 675/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.4034 -
categorical_accuracy: 0.8235
Epoch 676/1000
3/3 [==============================] - 0s 122ms/step - loss: 1.3157 -
categorical_accuracy: 0.7176
Epoch 677/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.7238 -
categorical_accuracy: 0.6824
Epoch 678/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.1010 -
categorical_accuracy: 1.0000
Epoch 679/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.6572 -
categorical_accuracy: 0.7765
Epoch 680/1000
3/3 [==============================] - 0s 130ms/step - loss: 12.8492 -
categorical_accuracy: 0.4706
Epoch 681/1000
3/3 [==============================] - 0s 119ms/step - loss: 8.3140 -
categorical_accuracy: 0.3647
Epoch 682/1000
3/3 [==============================] - 0s 121ms/step - loss: 3.7982 -
categorical_accuracy: 0.3529
Epoch 683/1000
3/3 [==============================] - 0s 132ms/step - loss: 6.4132 -
categorical_accuracy: 0.5059
Epoch 684/1000
3/3 [==============================] - 0s 116ms/step - loss: 7.6792 -
categorical_accuracy: 0.4706
Epoch 685/1000
3/3 [==============================] - 0s 119ms/step - loss: 6.9590 -
categorical_accuracy: 0.4471
Epoch 686/1000
3/3 [==============================] - 0s 125ms/step - loss: 14.2349 -
categorical_accuracy: 0.2706
Epoch 687/1000
3/3 [==============================] - 0s 120ms/step - loss: 7.3300 -
categorical_accuracy: 0.2353
Epoch 688/1000
3/3 [==============================] - 0s 123ms/step - loss: 3.2898 -
categorical_accuracy: 0.6235
Epoch 689/1000
3/3 [==============================] - 0s 119ms/step - loss: 1.4244 -
categorical_accuracy: 0.4588
Epoch 690/1000
3/3 [==============================] - 0s 116ms/step - loss: 1.9548 -
categorical_accuracy: 0.6235
Epoch 691/1000
```

```
3/3 [==============================] - 0s 130ms/step - loss: 0.3070 -
categorical_accuracy: 0.9176
Epoch 692/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.3636 -
categorical_accuracy: 0.7176
Epoch 693/1000
3/3 [==============================] - 0s 112ms/step - loss: 20.6109 -
categorical_accuracy: 0.2118
Epoch 694/1000
3/3 [==============================] - 0s 130ms/step - loss: 38.3568 -
categorical_accuracy: 0.2000
Epoch 695/1000
3/3 [==============================] - 0s 121ms/step - loss: 7.6794 -
categorical_accuracy: 0.5765
Epoch 696/1000
3/3 [==============================] - 0s 123ms/step - loss: 7.0720 -
categorical_accuracy: 0.3765
Epoch 697/1000
3/3 [==============================] - 0s 129ms/step - loss: 1.1291 -
categorical_accuracy: 0.7294
Epoch 698/1000
3/3 [==============================] - 1s 226ms/step - loss: 2.3835 -
categorical_accuracy: 0.5647
Epoch 699/1000
3/3 [==============================] - 1s 209ms/step - loss: 1.7717 -
categorical_accuracy: 0.3059
Epoch 700/1000
3/3 [==============================] - 1s 252ms/step - loss: 3.0335 -
categorical_accuracy: 0.3176
Epoch 701/1000
3/3 [==============================] - 1s 235ms/step - loss: 3.3823 -
categorical_accuracy: 0.3529
Epoch 702/1000
3/3 [==============================] - 1s 225ms/step - loss: 2.2736 -
categorical_accuracy: 0.1647
Epoch 703/1000
3/3 [==============================] - 0s 118ms/step - loss: 1.0447 -
categorical_accuracy: 0.4000
Epoch 704/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.5480 -
categorical_accuracy: 0.7176
Epoch 705/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.4068 -
categorical_accuracy: 0.7294
Epoch 706/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.1268 -
categorical_accuracy: 1.0000
Epoch 707/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.3953 -
categorical_accuracy: 0.7412
```

```
Epoch 708/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.5813 -
categorical_accuracy: 0.8471
Epoch 709/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.4300 -
categorical_accuracy: 0.7294
Epoch 710/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.4401 -
categorical_accuracy: 0.6471
Epoch 711/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.3747 -
categorical_accuracy: 0.7765
Epoch 712/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.2414 -
categorical_accuracy: 1.0000
Epoch 713/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.1936 -
categorical_accuracy: 1.0000
Epoch 714/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.1853 -
categorical_accuracy: 1.0000
Epoch 715/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.1481 -
categorical_accuracy: 1.0000
Epoch 716/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.1264 -
categorical_accuracy: 1.0000
Epoch 717/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.1054 -
categorical_accuracy: 1.0000
Epoch 718/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0866 -
categorical_accuracy: 1.0000
Epoch 719/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0782 -
categorical_accuracy: 1.0000
Epoch 720/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0689 -
categorical_accuracy: 1.0000
Epoch 721/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0601 -
categorical_accuracy: 1.0000
Epoch 722/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0530 -
categorical_accuracy: 1.0000
Epoch 723/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0472 -
categorical_accuracy: 1.0000
Epoch 724/1000
```

```
3/3 [==============================] - 0s 125ms/step - loss: 0.0414 -
categorical_accuracy: 1.0000
Epoch 725/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0366 -
categorical_accuracy: 1.0000
Epoch 726/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0321 -
categorical_accuracy: 1.0000
Epoch 727/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0276 -
categorical_accuracy: 1.0000
Epoch 728/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0238 -
categorical_accuracy: 1.0000
Epoch 729/1000
3/3 [==============================] - 1s 212ms/step - loss: 0.0205 -
categorical_accuracy: 1.0000
Epoch 730/1000
3/3 [==============================] - 1s 208ms/step - loss: 0.0183 -
categorical_accuracy: 1.0000
Epoch 731/1000
3/3 [==============================] - 0s 148ms/step - loss: 0.0178 -
categorical_accuracy: 1.0000
Epoch 732/1000
3/3 [==============================] - 1s 261ms/step - loss: 0.0170 -
categorical_accuracy: 1.0000
Epoch 733/1000
3/3 [==============================] - 1s 244ms/step - loss: 0.0155 -
categorical_accuracy: 1.0000
Epoch 734/1000
3/3 [==============================] - 1s 188ms/step - loss: 0.0143 -
categorical_accuracy: 1.0000
Epoch 735/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0135 -
categorical_accuracy: 1.0000
Epoch 736/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0128 -
categorical_accuracy: 1.0000
Epoch 737/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0122 -
categorical_accuracy: 1.0000
Epoch 738/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.0116 -
categorical_accuracy: 1.0000
Epoch 739/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0110 -
categorical_accuracy: 1.0000
Epoch 740/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0103 -
```

```
categorical_accuracy: 1.0000
Epoch 741/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.0098 -
categorical_accuracy: 1.0000
Epoch 742/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0093 -
categorical_accuracy: 1.0000
Epoch 743/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0088 -
categorical_accuracy: 1.0000
Epoch 744/1000
3/3 [==============================] - 0s 139ms/step - loss: 0.0084 -
categorical_accuracy: 1.0000
Epoch 745/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0079 -
categorical_accuracy: 1.0000
Epoch 746/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0074 -
categorical_accuracy: 1.0000
Epoch 747/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0070 -
categorical_accuracy: 1.0000
Epoch 748/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0066 -
categorical_accuracy: 1.0000
Epoch 749/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0063 -
categorical_accuracy: 1.0000
Epoch 750/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0060 -
categorical_accuracy: 1.0000
Epoch 751/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0058 -
categorical_accuracy: 1.0000
Epoch 752/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0056 -
categorical_accuracy: 1.0000
Epoch 753/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0054 -
categorical_accuracy: 1.0000
Epoch 754/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0052 -
categorical_accuracy: 1.0000
Epoch 755/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0050 -
categorical_accuracy: 1.0000
Epoch 756/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0048 -
categorical_accuracy: 1.0000
```

```
Epoch 757/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0047 -
categorical_accuracy: 1.0000
Epoch 758/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0045 -
categorical_accuracy: 1.0000
Epoch 759/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0044 -
categorical_accuracy: 1.0000
Epoch 760/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0043 -
categorical_accuracy: 1.0000
Epoch 761/1000
3/3 [==============================] - 1s 206ms/step - loss: 0.0041 -
categorical_accuracy: 1.0000
Epoch 762/1000
3/3 [==============================] - 1s 238ms/step - loss: 0.0040 -
categorical_accuracy: 1.0000
Epoch 763/1000
3/3 [==============================] - 1s 206ms/step - loss: 0.0039 -
categorical_accuracy: 1.0000
Epoch 764/1000
3/3 [==============================] - 1s 215ms/step - loss: 0.0038 -
categorical_accuracy: 1.0000
Epoch 765/1000
3/3 [==============================] - 1s 258ms/step - loss: 0.0038 -
categorical_accuracy: 1.0000
Epoch 766/1000
3/3 [==============================] - 0s 113ms/step - loss: 0.0037 -
categorical_accuracy: 1.0000
Epoch 767/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0036 -
categorical_accuracy: 1.0000
Epoch 768/1000
3/3 [==============================] - 0s 131ms/step - loss: 0.0035 -
categorical_accuracy: 1.0000
Epoch 769/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0035 -
categorical_accuracy: 1.0000
Epoch 770/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0034 -
categorical_accuracy: 1.0000
Epoch 771/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0034 -
categorical_accuracy: 1.0000
Epoch 772/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0033 -
categorical_accuracy: 1.0000
Epoch 773/1000
```

```
3/3 [==============================] - 0s 133ms/step - loss: 0.0032 -
categorical_accuracy: 1.0000
Epoch 774/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0032 -
categorical_accuracy: 1.0000
Epoch 775/1000
3/3 [==============================] - 0s 148ms/step - loss: 0.0031 -
categorical_accuracy: 1.0000
Epoch 776/1000
3/3 [==============================] - 0s 135ms/step - loss: 0.0031 -
categorical_accuracy: 1.0000
Epoch 777/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0030 -
categorical_accuracy: 1.0000
Epoch 778/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0030 -
categorical_accuracy: 1.0000
Epoch 779/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0030 -
categorical_accuracy: 1.0000
Epoch 780/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0029 -
categorical_accuracy: 1.0000
Epoch 781/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0029 -
categorical_accuracy: 1.0000
Epoch 782/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.0028 -
categorical_accuracy: 1.0000
Epoch 783/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0028 -
categorical_accuracy: 1.0000
Epoch 784/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0028 -
categorical_accuracy: 1.0000
Epoch 785/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 786/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 787/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0027 -
categorical_accuracy: 1.0000
Epoch 788/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0026 -
categorical_accuracy: 1.0000
Epoch 789/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0026 -
```

```
categorical_accuracy: 1.0000
Epoch 790/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0026 -
categorical_accuracy: 1.0000
Epoch 791/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0025 -
categorical_accuracy: 1.0000
Epoch 792/1000
3/3 [==============================] - 1s 202ms/step - loss: 0.0025 -
categorical_accuracy: 1.0000
Epoch 793/1000
3/3 [==============================] - 1s 216ms/step - loss: 0.0025 -
categorical_accuracy: 1.0000
Epoch 794/1000
3/3 [==============================] - 1s 228ms/step - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 795/1000
3/3 [==============================] - 1s 243ms/step - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 796/1000
3/3 [==============================] - 1s 257ms/step - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 797/1000
3/3 [==============================] - 1s 174ms/step - loss: 0.0024 -
categorical_accuracy: 1.0000
Epoch 798/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 799/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 800/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 801/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0023 -
categorical_accuracy: 1.0000
Epoch 802/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0022 -
categorical_accuracy: 1.0000
Epoch 803/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0022 -
categorical_accuracy: 1.0000
Epoch 804/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0022 -
categorical_accuracy: 1.0000
Epoch 805/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0022 -
categorical_accuracy: 1.0000
```

```
Epoch 806/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 807/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 808/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 809/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 810/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0021 -
categorical_accuracy: 1.0000
Epoch 811/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 812/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 813/1000
3/3 [==============================] - 0s 136ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 814/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 815/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0020 -
categorical_accuracy: 1.0000
Epoch 816/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 817/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 818/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 819/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 820/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0019 -
categorical_accuracy: 1.0000
Epoch 821/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 822/1000
```

```
3/3 [==============================] - 0s 117ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 823/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 824/1000
3/3 [==============================] - 1s 195ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 825/1000
3/3 [==============================] - 1s 234ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 826/1000
3/3 [==============================] - 1s 238ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 827/1000
3/3 [==============================] - 1s 258ms/step - loss: 0.0018 -
categorical_accuracy: 1.0000
Epoch 828/1000
3/3 [==============================] - 1s 252ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 829/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 830/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 831/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 832/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 833/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 834/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.0017 -
categorical_accuracy: 1.0000
Epoch 835/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 836/1000
3/3 [==============================] - 0s 112ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 837/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 838/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0016 -
```

```
categorical_accuracy: 1.0000
Epoch 839/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 840/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 841/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0016 -
categorical_accuracy: 1.0000
Epoch 842/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 843/1000
3/3 [==============================] - 0s 115ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 844/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 845/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 846/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 847/1000
3/3 [==============================] - 0s 129ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 848/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 849/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 850/1000
3/3 [==============================] - 0s 120ms/step - loss: 0.0015 -
categorical_accuracy: 1.0000
Epoch 851/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 852/1000
3/3 [==============================] - 0s 114ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 853/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 854/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
```

```
Epoch 855/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 856/1000
3/3 [==============================] - 1s 233ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 857/1000
3/3 [==============================] - 1s 232ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 858/1000
3/3 [==============================] - 1s 228ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 859/1000
3/3 [==============================] - 1s 264ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 860/1000
3/3 [==============================] - 1s 265ms/step - loss: 0.0014 -
categorical_accuracy: 1.0000
Epoch 861/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 862/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 863/1000
3/3 [==============================] - 0s 131ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 864/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 865/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 866/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 867/1000
3/3 [==============================] - 0s 132ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 868/1000
3/3 [==============================] - 0s 134ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 869/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 870/1000
3/3 [==============================] - 0s 118ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 871/1000
```

```
3/3 [==============================] - 0s 136ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 872/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0013 -
categorical_accuracy: 1.0000
Epoch 873/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 874/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 875/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 876/1000
3/3 [==============================] - 0s 141ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 877/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 878/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 879/1000
3/3 [==============================] - 0s 133ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 880/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 881/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 882/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 883/1000
3/3 [==============================] - 0s 130ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 884/1000
3/3 [==============================] - 0s 140ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 885/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0012 -
categorical_accuracy: 1.0000
Epoch 886/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 887/1000
3/3 [==============================] - 1s 222ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
```

```
Epoch 888/1000
3/3 [==============================] - 1s 205ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 889/1000
3/3 [==============================] - 1s 237ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 890/1000
3/3 [==============================] - 1s 240ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 891/1000
3/3 [==============================] - 1s 229ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 892/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 893/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 894/1000
3/3 [==============================] - 0s 128ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 895/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 896/1000
3/3 [==============================] - 0s 124ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 897/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 898/1000
3/3 [==============================] - 0s 125ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 899/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 900/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0011 -
categorical_accuracy: 1.0000
Epoch 901/1000
3/3 [==============================] - 0s 122ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 902/1000
3/3 [==============================] - 0s 127ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 903/1000
3/3 [==============================] - 0s 121ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 904/1000
```

```
3/3 [==============================] - 0s 119ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 905/1000
3/3 [==============================] - 0s 117ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 906/1000
3/3 [==============================] - 0s 126ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 907/1000
3/3 [==============================] - 0s 123ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 908/1000
3/3 [==============================] - 0s 119ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 909/1000
3/3 [==============================] - 0s 116ms/step - loss: 0.0010 -
categorical_accuracy: 1.0000
Epoch 910/1000
3/3 [==============================] - 0s 128ms/step - loss: 9.9436e-
04 - categorical_accuracy: 1.0000
Epoch 911/1000
3/3 [==============================] - 0s 126ms/step - loss: 9.8880e-
04 - categorical_accuracy: 1.0000
Epoch 912/1000
3/3 [==============================] - 0s 120ms/step - loss: 9.8342e-
04 - categorical_accuracy: 1.0000
Epoch 913/1000
3/3 [==============================] - 0s 121ms/step - loss: 9.7811e-
04 - categorical_accuracy: 1.0000
Epoch 914/1000
3/3 [==============================] - 0s 127ms/step - loss: 9.7235e-
04 - categorical_accuracy: 1.0000
Epoch 915/1000
3/3 [==============================] - 0s 122ms/step - loss: 9.6694e-
04 - categorical_accuracy: 1.0000
Epoch 916/1000
3/3 [==============================] - 0s 132ms/step - loss: 9.6177e-
04 - categorical_accuracy: 1.0000
Epoch 917/1000
3/3 [==============================] - 0s 126ms/step - loss: 9.5653e-
04 - categorical_accuracy: 1.0000
Epoch 918/1000
3/3 [==============================] - 0s 133ms/step - loss: 9.5109e-
04 - categorical_accuracy: 1.0000
Epoch 919/1000
3/3 [==============================] - 1s 214ms/step - loss: 9.4606e-
04 - categorical_accuracy: 1.0000
Epoch 920/1000
3/3 [==============================] - 1s 232ms/step - loss: 9.4115e-
```

```
04 - categorical_accuracy: 1.0000
Epoch 921/1000
3/3 [==============================] - 1s 230ms/step - loss: 9.3579e-
04 - categorical_accuracy: 1.0000
Epoch 922/1000
3/3 [==============================] - 1s 258ms/step - loss: 9.3085e-
04 - categorical_accuracy: 1.0000
Epoch 923/1000
3/3 [==============================] - 1s 243ms/step - loss: 9.2588e-
04 - categorical_accuracy: 1.0000
Epoch 924/1000
3/3 [==============================] - 0s 120ms/step - loss: 9.2065e-
04 - categorical_accuracy: 1.0000
Epoch 925/1000
3/3 [==============================] - 0s 131ms/step - loss: 9.1608e-
04 - categorical_accuracy: 1.0000
Epoch 926/1000
3/3 [==============================] - 0s 126ms/step - loss: 9.1112e-
04 - categorical_accuracy: 1.0000
Epoch 927/1000
3/3 [==============================] - 0s 118ms/step - loss: 9.0650e-
04 - categorical_accuracy: 1.0000
Epoch 928/1000
3/3 [==============================] - 0s 134ms/step - loss: 9.0168e-
04 - categorical_accuracy: 1.0000
Epoch 929/1000
3/3 [==============================] - 0s 118ms/step - loss: 8.9699e-
04 - categorical_accuracy: 1.0000
Epoch 930/1000
3/3 [==============================] - 0s 120ms/step - loss: 8.9215e-
04 - categorical_accuracy: 1.0000
Epoch 931/1000
3/3 [==============================] - 0s 133ms/step - loss: 8.8764e-
04 - categorical_accuracy: 1.0000
Epoch 932/1000
3/3 [==============================] - 0s 127ms/step - loss: 8.8291e-
04 - categorical_accuracy: 1.0000
Epoch 933/1000
3/3 [==============================] - 0s 124ms/step - loss: 8.7843e-
04 - categorical_accuracy: 1.0000
Epoch 934/1000
3/3 [==============================] - 0s 120ms/step - loss: 8.7410e-
04 - categorical_accuracy: 1.0000
Epoch 935/1000
3/3 [==============================] - 0s 123ms/step - loss: 8.6966e-
04 - categorical_accuracy: 1.0000
Epoch 936/1000
3/3 [==============================] - 0s 125ms/step - loss: 8.6503e-
04 - categorical_accuracy: 1.0000
```

```
Epoch 937/1000
3/3 [==============================] - 0s 121ms/step - loss: 8.6038e-
04 - categorical_accuracy: 1.0000
Epoch 938/1000
3/3 [==============================] - 0s 124ms/step - loss: 8.5611e-
04 - categorical_accuracy: 1.0000
Epoch 939/1000
3/3 [==============================] - 0s 135ms/step - loss: 8.5160e-
04 - categorical_accuracy: 1.0000
Epoch 940/1000
3/3 [==============================] - 0s 125ms/step - loss: 8.4732e-
04 - categorical_accuracy: 1.0000
Epoch 941/1000
3/3 [==============================] - 0s 122ms/step - loss: 8.4302e-
04 - categorical_accuracy: 1.0000
Epoch 942/1000
3/3 [==============================] - 0s 126ms/step - loss: 8.3887e-
04 - categorical_accuracy: 1.0000
Epoch 943/1000
3/3 [==============================] - 0s 119ms/step - loss: 8.3481e-
04 - categorical_accuracy: 1.0000
Epoch 944/1000
3/3 [==============================] - 0s 123ms/step - loss: 8.3052e-
04 - categorical_accuracy: 1.0000
Epoch 945/1000
3/3 [==============================] - 0s 119ms/step - loss: 8.2628e-
04 - categorical_accuracy: 1.0000
Epoch 946/1000
3/3 [==============================] - 0s 119ms/step - loss: 8.2218e-
04 - categorical_accuracy: 1.0000
Epoch 947/1000
3/3 [==============================] - 0s 128ms/step - loss: 8.1801e-
04 - categorical_accuracy: 1.0000
Epoch 948/1000
3/3 [==============================] - 0s 117ms/step - loss: 8.1392e-
04 - categorical_accuracy: 1.0000
Epoch 949/1000
3/3 [==============================] - 0s 116ms/step - loss: 8.1016e-
04 - categorical_accuracy: 1.0000
Epoch 950/1000
3/3 [==============================] - 1s 185ms/step - loss: 8.0629e-
04 - categorical_accuracy: 1.0000
Epoch 951/1000
3/3 [==============================] - 1s 227ms/step - loss: 8.0213e-
04 - categorical_accuracy: 1.0000
Epoch 952/1000
3/3 [==============================] - 1s 240ms/step - loss: 7.9822e-
04 - categorical_accuracy: 1.0000
Epoch 953/1000
```

```
3/3 [==============================] - 1s 217ms/step - loss: 7.9433e-
04 - categorical_accuracy: 1.0000
Epoch 954/1000
3/3 [==============================] - 1s 237ms/step - loss: 7.9051e-
04 - categorical_accuracy: 1.0000
Epoch 955/1000
3/3 [==============================] - 1s 190ms/step - loss: 7.8675e-
04 - categorical_accuracy: 1.0000
Epoch 956/1000
3/3 [==============================] - 0s 121ms/step - loss: 7.8294e-
04 - categorical_accuracy: 1.0000
Epoch 957/1000
3/3 [==============================] - 0s 135ms/step - loss: 7.7914e-
04 - categorical_accuracy: 1.0000
Epoch 958/1000
3/3 [==============================] - 0s 115ms/step - loss: 7.7540e-
04 - categorical_accuracy: 1.0000
Epoch 959/1000
3/3 [==============================] - 0s 138ms/step - loss: 7.7180e-
04 - categorical_accuracy: 1.0000
Epoch 960/1000
3/3 [==============================] - 0s 123ms/step - loss: 7.6804e-
04 - categorical_accuracy: 1.0000
Epoch 961/1000
3/3 [==============================] - 0s 144ms/step - loss: 7.6459e-
04 - categorical_accuracy: 1.0000
Epoch 962/1000
3/3 [==============================] - 0s 128ms/step - loss: 7.6082e-
04 - categorical_accuracy: 1.0000
Epoch 963/1000
3/3 [==============================] - 0s 131ms/step - loss: 7.5747e-
04 - categorical_accuracy: 1.0000
Epoch 964/1000
3/3 [==============================] - 0s 132ms/step - loss: 7.5374e-
04 - categorical_accuracy: 1.0000
Epoch 965/1000
3/3 [==============================] - 0s 131ms/step - loss: 7.5009e-
04 - categorical_accuracy: 1.0000
Epoch 966/1000
3/3 [==============================] - 0s 130ms/step - loss: 7.4669e-
04 - categorical_accuracy: 1.0000
Epoch 967/1000
3/3 [==============================] - 0s 118ms/step - loss: 7.4320e-
04 - categorical_accuracy: 1.0000
Epoch 968/1000
3/3 [==============================] - 0s 120ms/step - loss: 7.3975e-
04 - categorical_accuracy: 1.0000
Epoch 969/1000
3/3 [==============================] - 0s 123ms/step - loss: 7.3638e-
```

```
04 - categorical_accuracy: 1.0000
Epoch 970/1000
3/3 [==============================] - 0s 127ms/step - loss: 7.3284e-
04 - categorical_accuracy: 1.0000
Epoch 971/1000
3/3 [==============================] - 0s 120ms/step - loss: 7.2961e-
04 - categorical_accuracy: 1.0000
Epoch 972/1000
3/3 [==============================] - 0s 116ms/step - loss: 7.2623e-
04 - categorical_accuracy: 1.0000
Epoch 973/1000
3/3 [==============================] - 0s 130ms/step - loss: 7.2268e-
04 - categorical_accuracy: 1.0000
Epoch 974/1000
3/3 [==============================] - 0s 137ms/step - loss: 7.1960e-
04 - categorical_accuracy: 1.0000
Epoch 975/1000
3/3 [==============================] - 0s 119ms/step - loss: 7.1616e-
04 - categorical_accuracy: 1.0000
Epoch 976/1000
3/3 [==============================] - 0s 123ms/step - loss: 7.1290e-
04 - categorical_accuracy: 1.0000
Epoch 977/1000
3/3 [==============================] - 0s 133ms/step - loss: 7.0981e-
04 - categorical_accuracy: 1.0000
Epoch 978/1000
3/3 [==============================] - 0s 126ms/step - loss: 7.0660e-
04 - categorical_accuracy: 1.0000
Epoch 979/1000
3/3 [==============================] - 0s 123ms/step - loss: 7.0323e-
04 - categorical_accuracy: 1.0000
Epoch 980/1000
3/3 [==============================] - 0s 122ms/step - loss: 7.0000e-
04 - categorical_accuracy: 1.0000
Epoch 981/1000
3/3 [==============================] - 0s 168ms/step - loss: 6.9684e-
04 - categorical_accuracy: 1.0000
Epoch 982/1000
3/3 [==============================] - 1s 210ms/step - loss: 6.9356e-
04 - categorical_accuracy: 1.0000
Epoch 983/1000
3/3 [==============================] - 1s 255ms/step - loss: 6.9056e-
04 - categorical_accuracy: 1.0000
Epoch 984/1000
3/3 [==============================] - 1s 220ms/step - loss: 6.8761e-
04 - categorical_accuracy: 1.0000
Epoch 985/1000
3/3 [==============================] - 1s 244ms/step - loss: 6.8431e-
04 - categorical_accuracy: 1.0000
```

```
Epoch 986/1000
3/3 [==============================] - 1s 248ms/step - loss: 6.8134e-
04 - categorical_accuracy: 1.0000
Epoch 987/1000
3/3 [==============================] - 0s 123ms/step - loss: 6.7830e-
04 - categorical_accuracy: 1.0000
Epoch 988/1000
3/3 [==============================] - 0s 136ms/step - loss: 6.7527e-
04 - categorical_accuracy: 1.0000
Epoch 989/1000
3/3 [==============================] - 0s 120ms/step - loss: 6.7228e-
04 - categorical_accuracy: 1.0000
Epoch 990/1000
3/3 [==============================] - 0s 125ms/step - loss: 6.6924e-
04 - categorical_accuracy: 1.0000
Epoch 991/1000
3/3 [==============================] - 0s 135ms/step - loss: 6.6631e-
04 - categorical_accuracy: 1.0000
Epoch 992/1000
3/3 [==============================] - 0s 127ms/step - loss: 6.6333e-
04 - categorical_accuracy: 1.0000
Epoch 993/1000
3/3 [==============================] - 0s 142ms/step - loss: 6.6048e-
04 - categorical_accuracy: 1.0000
Epoch 994/1000
3/3 [==============================] - 0s 118ms/step - loss: 6.5764e-
04 - categorical_accuracy: 1.0000
Epoch 995/1000
3/3 [==============================] - 0s 126ms/step - loss: 6.5467e-
04 - categorical_accuracy: 1.0000
Epoch 996/1000
3/3 [==============================] - 0s 130ms/step - loss: 6.5179e-
04 - categorical_accuracy: 1.0000
Epoch 997/1000
3/3 [==============================] - 0s 120ms/step - loss: 6.4903e-
04 - categorical_accuracy: 1.0000
Epoch 998/1000
3/3 [==============================] - 0s 145ms/step - loss: 6.4618e-
04 - categorical_accuracy: 1.0000
Epoch 999/1000
3/3 [==============================] - 0s 130ms/step - loss: 6.4345e-
04 - categorical_accuracy: 1.0000
Epoch 1000/1000
3/3 [==============================] - 0s 123ms/step - loss: 6.4057e-
04 - categorical_accuracy: 1.0000
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 lstm_15 (LSTM)              (None, 30, 64)            442112

 lstm_16 (LSTM)              (None, 30, 128)           98816

 lstm_17 (LSTM)              (None, 64)                49408

 dense_14 (Dense)            (None, 64)                4160

 dense_15 (Dense)            (None, 32)                2080

 dense_16 (Dense)            (None, 3)                 99

=================================================================
Total params: 596675 (2.28 MB)
Trainable params: 596675 (2.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at
0x788660cfc280> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

1/1 [==============================] - 0s 388ms/step

/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3103: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

actions[np.argmax(res[4])]

{"type":"string"}

actions[np.argmax(y_test[4])]

{"type":"string"}

model.save('action.h5')

model.load_weights('action.h5')
```

```
from sklearn.metrics import multilabel_confusion_matrix,
accuracy_score
yhat = model.predict(X_test)
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()
multilabel_confusion_matrix(ytrue, yhat)
```

```
1/1 [==============================] - 0s 41ms/step
```

```
array([[[2, 0],
        [0, 3]],

       [[3, 0],
        [0, 2]]])
```

```
accuracy_score(ytrue, yhat)
```

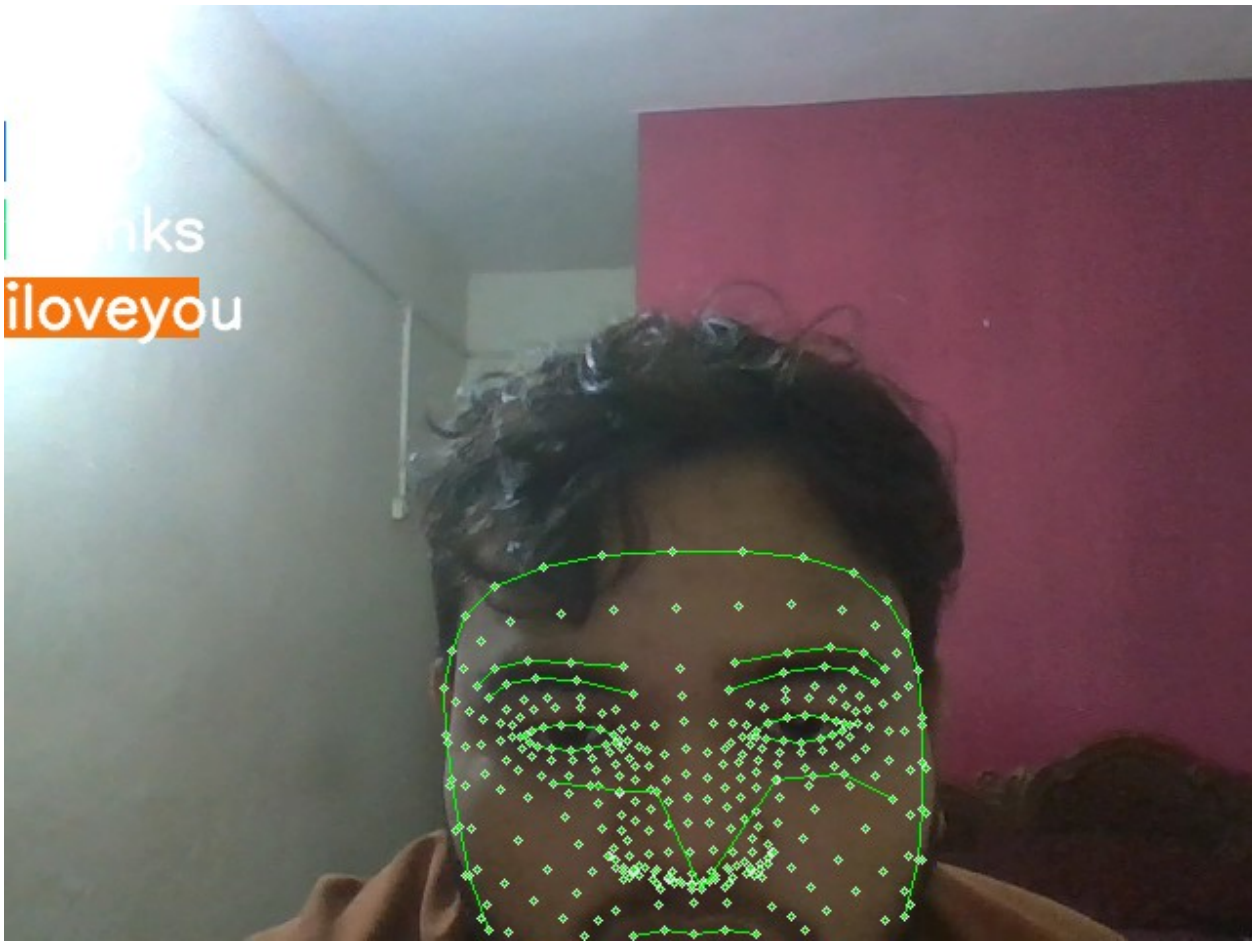
```
1.0
```

```
from scipy import stats

colors = [(245,117,16), (117,245,16), (16,117,245)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100),
90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

    return output_frame

plt.figure(figsize=(18,18))
plt.imshow(prob_viz(res, actions, image, colors))
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-74-0248883a8ffa> in <cell line: 2>()
      1 plt.figure(figsize=(18,18))
----> 2 plt.imshow(prob_viz(res, actions, image, colors))

<ipython-input-72-e9bc22b256b7> in prob_viz(res, actions, input_frame,
colors)
      3     output_frame = input_frame.copy()
      4     for num, prob in enumerate(res):
----> 5         cv2.rectangle(output_frame, (0,60+num*40),
(int(prob*100), 90+num*40), colors[num], -1)
      6         cv2.putText(output_frame, actions[num], (0,
85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2,
```

```
cv2.LINE_AA)
      7

TypeError: only length-1 arrays can be converted to Python scalars

<Figure size 1800x1800 with 0 Axes>
```

```python
import cv2
from google.colab.patches import cv2_imshow

colors = [(245,117,16), (117,245,16), (16,117,245)]

def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num in range(len(actions)):  # Iterate over number of actions
        prob = res[0, num]  # Access probability for action 'num'
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob *
100), 90 + num * 40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85 + num * 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    return output_frame

# Example usage in Colab (assuming 'res', 'actions', and 'image' are
defined)
image_with_probs = prob_viz(res, actions, image, colors)
cv2_imshow(image_with_probs)
```




```
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from scipy import stats
from base64 import b64decode

# Function to convert JavaScript object to OpenCV image
def js_to_image(js_reply):
    image_bytes = b64decode(js_reply.split(',')[1])
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    return img

# Function to perform landmark detection using Mediapipe
def mediapipe_detection(image, model):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_rgb.flags.writeable = False
    results = model.process(image_rgb)
    image_rgb.flags.writeable = True
    image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
```

```
    return image, results

# Function to draw landmarks on the image
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Load your trained model
model = load_model('action.h5')  # Replace with your model path

# Actions that your model can detect
actions = np.array(['hello', 'thanks', 'iloveyou'])

# Colors for visualization
colors = [(245,117,16), (117,245,16), (16,117,245)]

# Function to visualize probabilities
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob *
100), 90 + num * 40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85 + num * 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    return output_frame
```

```
# Function to handle real-time prediction
def run_prediction(image_data):
    # Convert base64 image to OpenCV format
    image = js_to_image(image_data)

    # Perform landmark detection
    image, results = mediapipe_detection(image, mp_holistic)

    # Extract keypoints
    keypoints = extract_keypoints(results)

    # Reshape keypoints for LSTM input (assuming your model expects
input_shape=(30, 1662))
    keypoints = np.array(keypoints).reshape((1, 30, 1662))

    # Make predictions
    res = model.predict(keypoints)

    # Visualize predictions on the frame
    image_with_probs = prob_viz(res[0], actions, image, colors)
    cv2_imshow(image_with_probs)  # Display the frame with predictions

# Example usage in Colab (assuming video streaming setup with
JavaScript)
# JavaScript setup should send each frame's base64 data to this
function
def handle_image_data(image_data):
    run_prediction(image_data)

# Call this function with JavaScript in your Colab notebook to start
streaming
# Example: handle_image_data(image_data_from_js)
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

```
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from scipy import stats
from base64 import b64decode
from google.colab.output import eval_js
```

```
from IPython.display import display, Javascript
from google.colab.patches import cv2_imshow

# Function to perform landmark detection using Mediapipe
def mediapipe_detection(image, model):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_rgb.flags.writeable = False
    results = model.process(image_rgb)
    image_rgb.flags.writeable = True
    image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
    return image, results

# Function to draw landmarks on the image
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks,
mp_holistic.FACEMESH_CONTOURS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.pose_landmarks,
mp_holistic.POSE_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              landmark_drawing_spec,
connection_drawing_spec)

# Initialize Mediapipe Holistic model and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Drawing specification for landmarks
landmark_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)
connection_drawing_spec = mp_drawing.DrawingSpec(color=(0, 255, 0),
thickness=1, circle_radius=1)

# Load your trained model
model = load_model('action.h5')  # Replace with your model path

# Actions that your model can detect
actions = np.array(['hello', 'thanks', 'iloveyou'])

# Colors for visualization
colors = [(245,117,16), (117,245,16), (16,117,245)]
```

```
# Function to visualize probabilities
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob *
100), 90 + num * 40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85 + num * 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    return output_frame

# Initialize webcam capture
cap = cv2.VideoCapture(0)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Perform landmark detection
    image, results = mediapipe_detection(frame, mp_holistic)

    # Extract keypoints
    keypoints = extract_keypoints(results)

    # Reshape keypoints for LSTM input (assuming your model expects
input_shape=(30, 1662))
    keypoints = np.array(keypoints).reshape((1, 30, 1662))

    # Make predictions
    res = model.predict(keypoints)

    # Visualize predictions on the frame
    image_with_probs = prob_viz(res[0], actions, image, colors)
    cv2.imshow('Real-time Detection', image_with_probs)

    # Exit loop on 'q' press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it
```

```
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.

from matplotlib import pyplot as plt
import cv2
import numpy as np

def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        # Ensure colors[num] is converted to tuple if it's not already
        color = tuple(colors[num]) if not isinstance(colors[num],
tuple) else colors[num]

        # Draw filled rectangle based on probability
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob *
100), 90 + num * 40), color, -1)

        # Add text label for action
        cv2.putText(output_frame, actions[num], (0, 85 + num * 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    return output_frame

# Example usage in Colab (assuming 'res', 'actions', and 'image' are
defined)
image_with_probs = prob_viz(res, actions, image, colors)

# Convert BGR image to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image_with_probs, cv2.COLOR_BGR2RGB)

# Display the image using matplotlib
plt.figure(figsize=(12, 8))
plt.imshow(image_rgb)
plt.axis('off')  # Hide axis
plt.show()

---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-109-b77c1e0decb2> in <cell line: 20>()
     18
     19 # Example usage in Colab (assuming 'res', 'actions', and
'image' are defined)
---> 20 image_with_probs = prob_viz(res, actions, image, colors)
     21
     22 # Convert BGR image to RGB for displaying with matplotlib

<ipython-input-109-b77c1e0decb2> in prob_viz(res, actions,
```

```
input_frame, colors)
     10
     11         # Draw filled rectangle based on probability
---> 12         cv2.rectangle(output_frame, (0, 60 + num * 40),
(int(prob * 100), 90 + num * 40), color, -1)
     13
     14         # Add text label for action

TypeError: only length-1 arrays can be converted to Python scalars
```

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import numpy as np
import cv2

# Function to capture a frame from webcam using JavaScript
def capture_frame():
    js = Javascript('''
        async function captureFrame() {
            const video = document.createElement('video');
            video.style.display = 'block';
            const stream = await
navigator.mediaDevices.getUserMedia({video: true});
            document.body.appendChild(video);
            video.srcObject = stream;
            await video.play();

            // Capture a frame from the video
            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);
            stream.getVideoTracks()[0].stop();
            return canvas.toDataURL('image/jpeg');
        }
        ''')
    display(js)
    data_url = eval_js('captureFrame()')
    binary = b64decode(data_url.split(',')[1])
    return np.asarray(bytearray(binary), dtype="uint8")

# Capture a frame from webcam
frame = capture_frame()
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert to OpenCV
format (RGB)
# 1. New detection variables
sequence = []
sentence = []
predictions = []
```

```
threshold = 0.5

# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    while True:
        # Make detections
        image, results = mediapipe_detection(frame, holistic)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))

            # 3. Action recognition logic
            if np.unique(predictions[-10:])[0] == np.argmax(res) and
res[np.argmax(res)] > threshold:
                if len(sentence) > 0 and actions[np.argmax(res)] !=
sentence[-1]:
                    sentence.append(actions[np.argmax(res)])
                elif len(sentence) == 0:
                    sentence.append(actions[np.argmax(res)])

                if len(sentence) > 5:
                    sentence = sentence[-5:]

            # 4. Visualization logic
            image = prob_viz(res, actions, image, colors)

        # Display sentence on image
        cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show image
        cv2_imshow(cv2.cvtColor(image, cv2.COLOR_RGB2BGR))

        # Capture next frame
        frame = capture_frame()

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
            break

# Clean up
cv2.destroyAllWindows()


<IPython.core.display.Javascript object>

---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-1-78904b61b22a> in <cell line: 42>()
     40
     41 # Set mediapipe model
---> 42 with mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
     43     while True:
     44         # Make detections

NameError: name 'mp_holistic' is not defined

!pip install tensorflow  # Install TensorFlow (adjust version as
needed)
!pip install mediapipe opencv-python-headless  # Install Mediapipe and
OpenCV
!pip install matplotlib  # Install matplotlib for visualization

Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
```

```
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.25.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
```

```
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2024.6.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5-
>tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
Requirement already satisfied: mediapipe in
/usr/local/lib/python3.10/dist-packages (0.10.14)
Requirement already satisfied: opencv-python-headless in
/usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (1.4.0)
Requirement already satisfied: attrs>=19.1.0 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (23.2.0)
Requirement already satisfied: flatbuffers>=2.0 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (24.3.25)
Requirement already satisfied: jax in /usr/local/lib/python3.10/dist-
packages (from mediapipe) (0.4.26)
Requirement already satisfied: jaxlib in
/usr/local/lib/python3.10/dist-packages (from mediapipe)
(0.4.26+cuda12.cudnn89)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (3.7.1)
Requirement already satisfied: numpy in
```

```
/usr/local/lib/python3.10/dist-packages (from mediapipe) (1.25.2)
Requirement already satisfied: opencv-contrib-python in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (4.8.0.76)
Requirement already satisfied: protobuf<5,>=4.25.3 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (4.25.3)
Requirement already satisfied: sounddevice>=0.4.4 in
/usr/local/lib/python3.10/dist-packages (from mediapipe) (0.4.7)
Requirement already satisfied: CFFI>=1.0 in
/usr/local/lib/python3.10/dist-packages (from sounddevice>=0.4.4-
>mediapipe) (1.16.0)
Requirement already satisfied: ml-dtypes>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (0.2.0)
Requirement already satisfied: opt-einsum in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (3.3.0)
Requirement already satisfied: scipy>=1.9 in
/usr/local/lib/python3.10/dist-packages (from jax->mediapipe) (1.11.4)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->mediapipe)
(2.8.2)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.10/dist-packages (from CFFI>=1.0-
>sounddevice>=0.4.4->mediapipe) (2.22)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->mediapipe) (1.16.0)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib) (1.16.0)
```

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from IPython.display import display, Javascript, Image
import PIL.Image
from io import BytesIO
import mediapipe as mp

# Function to capture a frame from webcam using JavaScript in Colab
def capture_frame():
    js = Javascript('''
        async function captureFrame() {
            const video = document.createElement('video');
            video.style.display = 'block';
            const stream = await
navigator.mediaDevices.getUserMedia({video: true});
            document.body.appendChild(video);
            video.srcObject = stream;
            await video.play();

            // Capture a frame from the video
            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);
            stream.getVideoTracks()[0].stop();
            return canvas.toDataURL('image/jpeg');
        }
        ''')
```

```
    display(js)
    data_url = eval_js('captureFrame()')
    binary = b64decode(data_url.split(',')[1])
    return np.asarray(bytearray(binary), dtype="uint8")

# Function to visualize probabilities on the frame
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0, 60 + num * 40), (int(prob *
100), 90 + num * 40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85 + num * 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    return output_frame

from tensorflow.keras.models import load_model

# Load your model
model = load_model('action.h5')  # Replace with your model path
actions = ['hello', 'thanks', 'iloveyou']  # Replace with your action
labels
colors = [(245, 117, 16), (117, 245, 16), (16, 117, 245)]  # Colors
for visualization
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
WARNING:tensorflow:Layer lstm_17 will not use cuDNN kernels since it
doesn't meet the criteria. It will use a generic GPU kernel as
fallback when running on GPU.
```

```
# 1. New detection variables
import mediapipe as mp
sequence = []
sentence = []
predictions = []
threshold = 0.5

# Set mediapipe model
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Function to process Mediapipe detections
def mediapipe_detection(frame, holistic):
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert to RGB
for Mediapipe
    image.flags.writeable = False
```

```
    results = holistic.process(image)  # Make detection
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # Convert back to
BGR
    return image, results

# Initialize Mediapipe Holistic model
holistic = mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5)

# Main loop for real-time action recognition
while True:
    # Capture frame from webcam
    frame = capture_frame()

    # Make detections with Mediapipe
    image, results = mediapipe_detection(frame, holistic)

    # Extract keypoints and update sequence
    keypoints = extract_keypoints(results)  # Implement
'extract_keypoints' function as per your requirement
    sequence.append(keypoints)
    sequence = sequence[-30:]  # Keep only the last 30 frames

    # Perform action recognition if sequence length is 30
    if len(sequence) == 30:
        res = model.predict(np.expand_dims(sequence, axis=0))[0]
        print(actions[np.argmax(res)])  # Print predicted action label
        predictions.append(np.argmax(res))

        # Update sentence based on predictions
        if np.unique(predictions[-10:])[0] == np.argmax(res) and
res[np.argmax(res)] > threshold:
            if len(sentence) > 0 and actions[np.argmax(res)] !=
sentence[-1]:
                sentence.append(actions[np.argmax(res)])
            elif len(sentence) == 0:
                sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

        # Visualize probabilities on the image
        image = prob_viz(res, actions, image, colors)

    # Display sentence on image
    cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
    cv2.putText(image, ' '.join(sentence), (3, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
```

```
    # Display image with OpenCV
    cv2.imshow('OpenCV Feed', image)

    # Exit loop if 'q' is pressed
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

# Clean up
cap.release()
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-4fde0edda79d> in <cell line: 25>()
     25 while True:
     26     # Capture frame from webcam
---> 27     frame = capture_frame()
     28
     29     # Make detections with Mediapipe

NameError: name 'capture_frame' is not defined
```

```
# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

# Set mediapipe model
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Initialize Mediapipe Holistic model
holistic = mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5)

# Main loop for real-time action recognition
while True:
    # Capture frame from webcam
    frame = capture_frame()

    # Make detections with Mediapipe
    image = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)  # Convert to BGR
for OpenCV
    results = holistic.process(image)  # Make detection

    # Extract keypoints and update sequence
    keypoints = extract_keypoints(results)  # Implement
```

```
'extract_keypoints' function as per your requirement
    sequence.append(keypoints)
    sequence = sequence[-30:]  # Keep only the last 30 frames

    # Perform action recognition if sequence length is 30
    if len(sequence) == 30:
        res = model.predict(np.expand_dims(sequence, axis=0))[0]
        print(actions[np.argmax(res)])  # Print predicted action label
        predictions.append(np.argmax(res))

        # Update sentence based on predictions
        if np.unique(predictions[-10:])[0] == np.argmax(res) and
res[np.argmax(res)] > threshold:
            if len(sentence) > 0 and actions[np.argmax(res)] !=
sentence[-1]:
                sentence.append(actions[np.argmax(res)])
            elif len(sentence) == 0:
                sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

        # Visualize probabilities on the image
        image = prob_viz(res, actions, image, colors)

    # Display sentence on image
    cv2.rectangle(image, (0, 0), (640, 40), (245, 117, 16), -1)
    cv2.putText(image, ' '.join(sentence), (3, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    # Display image with OpenCV
    cv2.imshow('OpenCV Feed', image)

    # Exit loop if 'q' is pressed
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

# Clean up
cap.release()
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-3-b3dd031fa6f0> in <cell line: 15>()
     15 while True:
     16     # Capture frame from webcam
---> 17     frame = capture_frame()
     18
```

```
     19     # Make detections with Mediapipe

NameError: name 'capture_frame' is not defined
```