Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Image Processing and Computer Vision - II Laboratory (DJ19DSL702)

AY: 2024-25

Experiment 6

(Spatio-Temporal Analysis)

Jhanvi Parekh

60009210033

CSE (Data Science)

Aim: Analyse body postures/ keypoints using MoveNet.

Theory:

1. Introduction

There are several methods for body posture detection using spatio-temporal analysis. These methods leverage the changes in body keypoints and their relationships over time to infer different postures. Here are some common approaches:

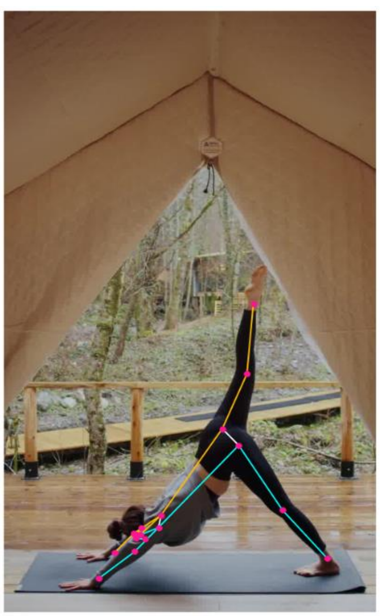
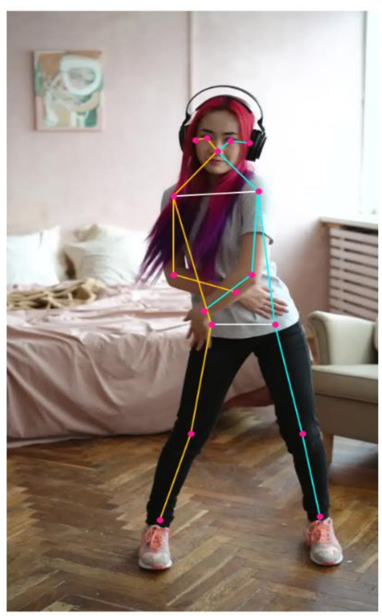
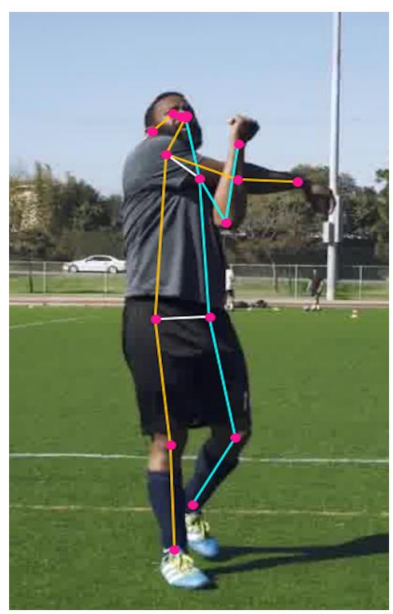

Figure 1. Keypoints of the body

1. Optical Flow Analysis:

Optical flow is a method that tracks the movement of pixels between consecutive frames in a video. By applying optical flow techniques to human body keypoints, you can estimate how these keypoints move over time. This information can be used to determine body posture changes and gestures.

2. Recurrent Neural Networks (RNNs):

RNNs are a type of neural network architecture designed to handle sequences of data. By treating the sequence of body keypoints as a time-series data, you can use RNNs to capture temporal dependencies and patterns in body posture changes. Long Short-Term Memory (LSTM) networks, a type of RNN, are commonly used for this purpose.

3. Convolutional Neural Networks (CNNs) + Temporal Layers:

You can combine the power of CNNs for spatial analysis with temporal layers for tracking changes over time. This approach involves creating a 3D convolutional neural network (CNN) that takes a sequence of video frames as input. The 3D CNN can learn to extract both spatial and temporal features from the video frames, aiding in posture detection.

4. Graph Convolutional Networks (GCNs):

GCNs are specialized neural networks designed for analyzing graph-structured data, where the relationships between elements matter. In the context of body posture detection, you can represent body keypoints as nodes in a graph and edges between them representing skeletal connections. GCNs can then analyze how these connections change over time to infer posture.

5. Hidden Markov Models (HMMs):

6. 2D Pose Matching:

This approach involves comparing the detected 2D poses in each frame with predefined templates of various postures. By matching the detected pose to the closest template, you can identify the corresponding body posture.

7. 3D Pose Estimation:

If you have access to depth information, you can perform 3D pose estimation. This approach captures the spatial relationships between keypoints in a three-dimensional space, allowing for a more accurate analysis of body posture changes over time.

MoveNet:

MoveNet is a deep learning model developed by Google that focuses on human pose estimation and tracking. It's designed to accurately estimate 2D and 3D human body keypoints and poses in real time. The model uses a lightweight architecture that makes it suitable for real-time applications on mobile devices and embedded systems.

MoveNet operates in a spatio-temporal manner by analyzing the changes in body keypoints across frames of a video sequence. This enables it to track the movement of body parts over time, allowing for a more comprehensive understanding of body posture and gestures.

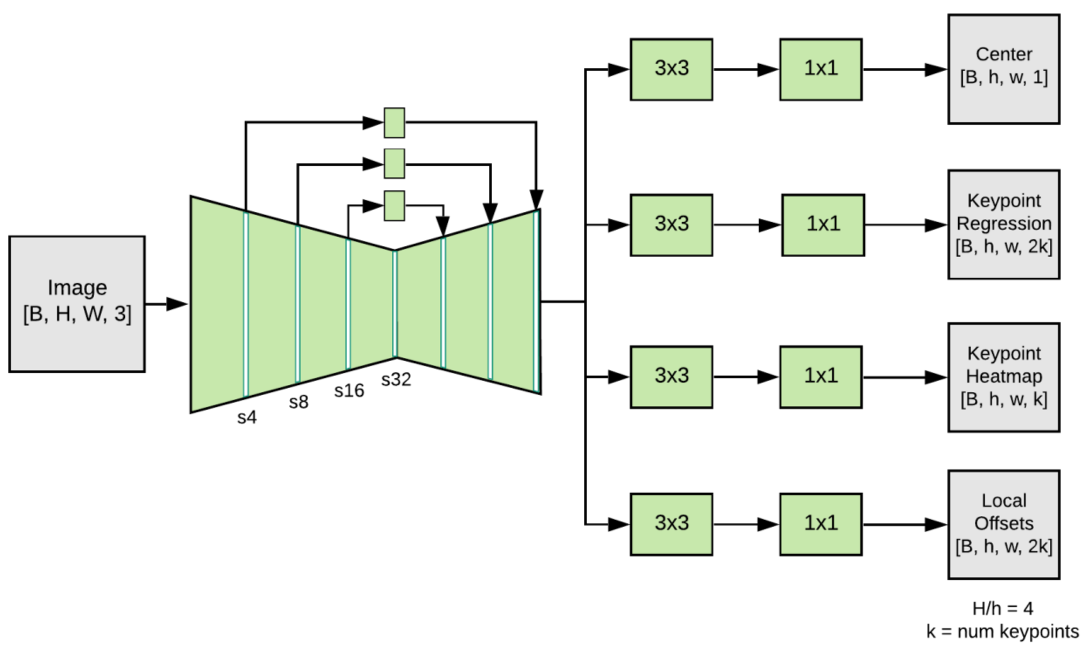

Figure 2. Architecture of MoveNet

Spatio-Temporal Analysis:

In the context of body posture analysis, spatio-temporal analysis involves studying how the positions of various body keypoints change over consecutive frames of a video. This analysis can provide insights into the dynamics of body movements, helping to identify actions and gestures.

Experimental Setup:

1. Data Collection: For this experiment, you'll need a video dataset that includes individuals performing various actions or movements. These videos should ideally capture different angles and perspectives of the subjects.

2. Preprocessing: Before applying MoveNet, the videos need to be preprocessed. This involves tasks such as resizing frames, converting videos to image frames, and normalizing pixel values.

3. MoveNet Implementation: Implement the MoveNet model using a deep learning framework like TensorFlow or PyTorch. Load pre-trained weights for MoveNet so that you can utilize its pose estimation capabilities.

4. Spatio-Temporal Analysis: Apply MoveNet to the video frames and obtain human body keypoints for each frame. Track the movement of these keypoints across consecutive frames to analyze how they change over time.

5. Posture Analysis: Based on the spatio-temporal analysis, you can infer various insights about body posture, such as identifying specific actions, gestures, or anomalies. For example, you can detect actions like walking, waving, or sitting down based on the movement of keypoints.

6. Visualization: Visualize the results of your analysis by overlaying the estimated keypoints onto the video frames. This will provide a clear representation of how the body postures change over time.

Conclusion:

Spatio-temporal analysis using MoveNet offers a powerful approach to understanding human body movements and postures. It enables the identification of actions and gestures, making it useful for applications like video surveillance, sports analysis, and healthcare monitoring.

Lab Assignment to complete after this session:

Objective:

The objective of this lab assignment is to utilize MoveNet to perform spatio-temporal analysis on a given video, tracking body keypoints and identifying patterns of body postures over time.

Requirements:

1. Python environment with TensorFlow and OpenCV installed.

2. Pre-trained MoveNet model weights.

3. Sample video containing human body movements.

Steps:

1. Load the MoveNet model with pre-trained weights.

2. Read and preprocess the input video frames.

3. For each frame, perform pose estimation using MoveNet.

4. Store the keypoints and their corresponding timestamps.

5. Analyze the temporal sequence of keypoints to identify patterns and movements for any video input.

6. Visualize the results by overlaying keypoints on the video frames.

```
import cv2
import numpy as np
import tensorflow as tf
import imageio
from tensorflow_docs.vis import embed
from IPython.display import display, Javascript
from google.colab.output import eval_js
# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
import base64
import cv2
import numpy as np
import PIL
```

```
import io
import html
import time
from google.colab.patches import cv2_imshow

model = hub.load("https://tfhub.dev/google/movenet/multipose/lightning/1")
movenet = model.signatures["serving_default"]

def load_gif():
    """Loads the GIF and returns its details."""
    gif = cv2.VideoCapture("/content/ngannou.gif")
    frame_count = int(gif.get(cv2.CAP_PROP_FRAME_COUNT))
    print(f"Frame count: {frame_count}")

    output_frames = []
    initial_shape = [int(gif.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(gif.get(cv2.CAP_PROP_FRAME_HEIGHT))]
    return gif, frame_count, output_frames, initial_shape

def run_inference_on_frame(frame):
    """Run Movenet inference on a single frame."""
    input_image = tf.image.resize_with_pad(tf.expand_dims(frame, axis=0),
192, 192)
    input_image = tf.cast(input_image, dtype=tf.int32)

    # Run inference and get the keypoints
    results = movenet(input_image)
    keypoints = results["output_0"].numpy()[:,:,:51].reshape((6,17,3))

    return keypoints

# Define the skeleton edges (keypoints indices for each limb) and their
colors
EDGES = {
    (0, 1): (255, 0, 0),  # Nose to Left Eye
    (0, 2): (255, 0, 0),  # Nose to Right Eye
    (1, 3): (0, 255, 0),  # Left Eye to Left Ear
    (2, 4): (0, 255, 0),  # Right Eye to Right Ear
    (5, 6): (0, 255, 255),  # Left Shoulder to Right Shoulder
    (5, 7): (255, 255, 0),  # Left Shoulder to Left Elbow
    (7, 9): (255, 255, 0),  # Left Elbow to Left Wrist
```

```
    (6, 8): (0, 255, 255),  # Right Shoulder to Right Elbow
    (8, 10): (0, 255, 255),  # Right Elbow to Right Wrist
    (11, 12): (255, 0, 255),  # Left Hip to Right Hip
    (5, 11): (255, 255, 255),  # Left Shoulder to Left Hip
    (6, 12): (255, 255, 255),  # Right Shoulder to Right Hip
    (11, 13): (255, 0, 0),  # Left Hip to Left Knee
    (13, 15): (255, 0, 0),  # Left Knee to Left Ankle
    (12, 14): (0, 255, 0),  # Right Hip to Right Knee
    (14, 16): (0, 255, 0)   # Right Knee to Right Ankle
}

def draw_edges(keypoints, frame, edges_colors, threshold=0.5):
    """Draws edges between keypoints on the frame."""
    height, width, _ = frame.shape

    # Iterate through the edges
    for edge, color in edges_colors.items():
        p1, p2 = edge  # Get keypoint indices for the edge
        y1, x1, c1 = keypoints[p1]  # Extract (y1, x1) and confidence for
first keypoint
        y2, x2, c2 = keypoints[p2]  # Extract (y2, x2) and confidence for
second keypoint

        # Draw line only if both keypoints have confidence > threshold
        if c1 > threshold and c2 > threshold:
            cv2.line(
                frame,
                (int(x1 * width), int(y1 * height)),
                (int(x2 * width), int(y2 * height)),
                color=color,
                thickness=2,
                lineType=cv2.LINE_AA  # Anti-aliased line for smoother
look
            )

def draw_keypoints(frame, keypoints):
    """Draws keypoints detected by Movenet on the frame."""
    height, width, _ = frame.shape

    # Iterate over each person detected
    for person in keypoints:
```

```
        # Iterate over all 17 keypoints for the person
        y, x, conf = person  # Extract (y, x) coordinates and confidence
        if conf > 0.5:  # Only draw if confidence is high
            cv2.circle(frame, (int(x * width), int(y * height)), 5, (0,
255, 0), -1)

    return frame

def video_stream():
    """JavaScript to properly create a live video stream using the
webcam."""
    js = Javascript('''
        // JavaScript code to create a video stream
        var video;
        var div = null;
        var stream;
        var captureCanvas;
        var imgElement;
        var labelElement;

        var pendingResolve = null;
        var shutdown = false;

        function removeDom() {
           stream.getVideoTracks()[0].stop();
           video.remove();
           div.remove();
           video = null;
           div = null;
           stream = null;
           imgElement = null;
           captureCanvas = null;
           labelElement = null;
        }

        function onAnimationFrame() {
          if (!shutdown) {
            window.requestAnimationFrame(onAnimationFrame);
          }
          if (pendingResolve) {
            var result = "";
```

```
            if (!shutdown) {
              captureCanvas.getContext('2d').drawImage(video, 0, 0, 640,
480);
              result = captureCanvas.toDataURL('image/jpeg', 0.8)
            }
            var lp = pendingResolve;
            pendingResolve = null;
            lp(result);
          }
        }

        async function createDom() {
          if (div !== null) {
            return stream;
          }

          div = document.createElement('div');
          div.style.border = '2px solid black';
          div.style.padding = '3px';
          div.style.width = '100%';
          div.style.maxWidth = '600px';
          document.body.appendChild(div);

          const modelOut = document.createElement('div');
          modelOut.innerHTML = "<span>Status:</span>";
          labelElement = document.createElement('span');
          labelElement.innerText = 'No data';
          labelElement.style.fontWeight = 'bold';
          modelOut.appendChild(labelElement);
          div.appendChild(modelOut);

          video = document.createElement('video');
          video.style.display = 'block';
          video.width = div.clientWidth - 6;
          video.setAttribute('playsinline', '');
          video.onclick = () => { shutdown = true; };
          stream = await navigator.mediaDevices.getUserMedia(
              {video: { facingMode: "environment"}});
          div.appendChild(video);

          imgElement = document.createElement('img');
```

```
          imgElement.style.position = 'absolute';
          imgElement.style.zIndex = 1;
          imgElement.onclick = () => { shutdown = true; };
          div.appendChild(imgElement);

          const instruction = document.createElement('div');
          instruction.innerHTML =
              '<span style="color: red; font-weight: bold;">' +
              'When finished, click here or on the video to stop this
demo</span>';
          div.appendChild(instruction);
          instruction.onclick = () => { shutdown = true; };

          video.srcObject = stream;
          await video.play();

          captureCanvas = document.createElement('canvas');
          captureCanvas.width = 640;
          captureCanvas.height = 480;
          window.requestAnimationFrame(onAnimationFrame);

          return stream;
        }
        async function stream_frame(label, imgData) {
          if (shutdown) {
            removeDom();
            shutdown = false;
            return '';
          }

          var preCreate = Date.now();
          stream = await createDom();

          var preShow = Date.now();
          if (label != "") {
            labelElement.innerHTML = label;
          }

          if (imgData != "") {
            var videoRect = video.getClientRects()[0];
            imgElement.style.top = videoRect.top + "px";
```

```
          imgElement.style.left = videoRect.left + "px";
          imgElement.style.width = videoRect.width + "px";
          imgElement.style.height = videoRect.height + "px";
          imgElement.src = imgData;
        }

        var preCapture = Date.now();
        var result = await new Promise(function(resolve, reject) {
          pendingResolve = resolve;
        });
        shutdown = false;

        return {'create': preShow - preCreate,
                'show': preCapture - preShow,
                'capture': Date.now() - preCapture,
                'img': result};
      }
    ''')
    display(js)

def video_frame(label, bbox):
    """Captures a frame from the webcam stream."""
    data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
    return data

def js_to_image(js_reply):
    """Converts JavaScript response to OpenCV image."""
    img_bytes = base64.b64decode(js_reply.split(',')[1])
    np_img = np.frombuffer(img_bytes, dtype=np.uint8)
    return cv2.imdecode(np_img, cv2.IMREAD_COLOR)

def process_frame(frame, keypoints):
    """Draws both keypoints and edges on the frame."""
    for person_keypoints in keypoints:
        draw_keypoints(frame, person_keypoints)  # Draw keypoints
        draw_edges(person_keypoints, frame, EDGES)   # Draw edges
(skeleton)
    return frame

# Start video stream
video_stream()
```

```
# Real-time pose estimation loop
label_html = 'Capturing...'
bbox = ''
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # Convert JavaScript response to OpenCV image
    frame = js_to_image(js_reply["img"])

    # Run inference and get keypoints
    keypoints = run_inference_on_frame(frame)

    # Process the frame to draw keypoints and edges
    frame_with_skeleton = process_frame(frame, keypoints)

    # Display the frame with skeleton
    cv2_imshow(frame_with_skeleton)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```