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: 2022-23

Experiment 6

(Spatio-Temporal Analysis)

Bhuvi Ghosh
60009210191

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.

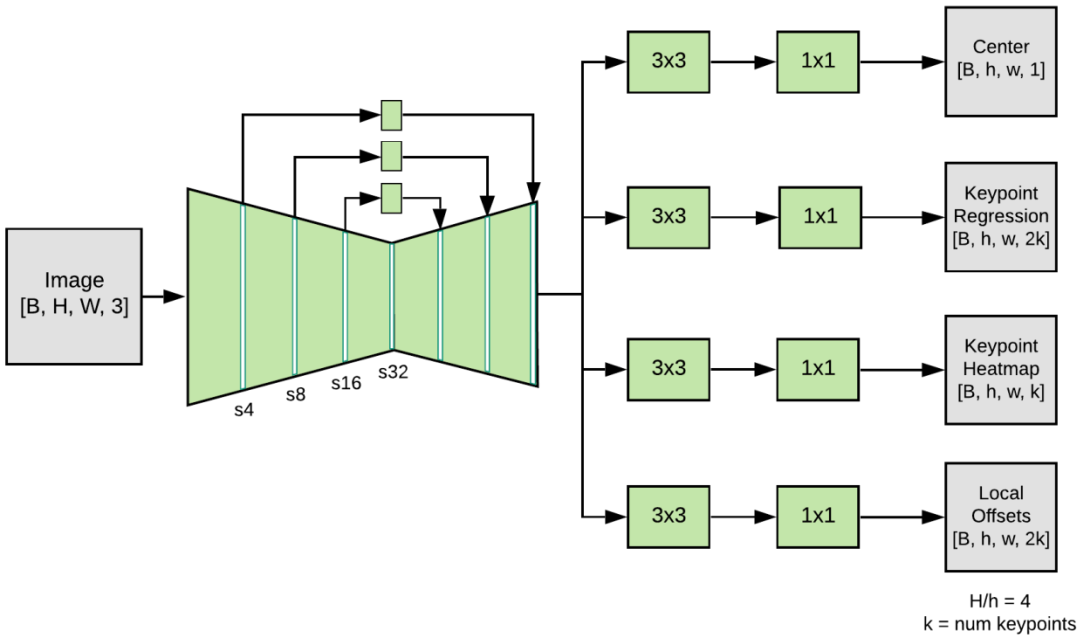


Figure 2. Architecture of MoveNet

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.

Libraries

```
[1] # Computer vision/graphics library
    import cv2

    # Gif writer
    import imageio

    # Display libraries
    import matplotlib.pyplot as plt
    from IPython.display import HTML, display

    # Calculations and Deep Learning library
    import numpy as np
    import tensorflow as tf
    import tensorflow_hub as hub
```

```
! pip install -q git+https://github.com/tensorflow/docs
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for tensorflow-docs (setup.py) ... done
```

Setup

Map the bones (keypoint edges) to a matplotlib color name

```
[3]  cyan = (255, 255, 0)
     magenta = (255, 0, 255)
```

```
[4]  EDGE_COLORS = {
         (0, 1): magenta,
         (0, 2): cyan,
         (1, 3): magenta,
         (2, 4): cyan,
         (0, 5): magenta,
         (0, 6): cyan,
         (5, 7): magenta,
         (7, 9): cyan,
         (6, 8): magenta,
         (8, 10): cyan,
         (5, 6): magenta,
         (5, 11): cyan,
         (6, 12): magenta,
         (11, 12): cyan,
         (11, 13): magenta,
         (13, 15): cyan,
         (12, 14): magenta,
         (14, 16): cyan
     }
```

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

```
[6]  #initial_width, initial_height = (461,250)
     WIDTH = HEIGHT = 256
```

Inference

Download the test gif

```
[7] ! wget -O ngannou.gif https://raw.githubusercontent.com/Justsecret123/Human-pose-estimation/main/Test%20gifs/Ngannou_3.gif
```

```
--2024-11-09 15:41:30--  https://raw.githubusercontent.com/Justsecret123/Human-pose-estimation/main/Test%20gifs/Ngannou_3.gif
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7431914 (7.1M) [application/octet-stream]
Saving to: 'ngannou.gif'

ngannou.gif         100%[===================>]   7.09M  --.-KB/s    in 0.07s

2024-11-09 15:41:30 (109 MB/s) - 'ngannou.gif' saved [7431914/7431914]
```

```
[8] def loop(frame, keypoints, threshold=0.11):
        """
        Main loop : Draws the keypoints and edges for each instance
        """

        # Loop through the results
        for instance in keypoints:
            # Draw the keypoints and get the denormalized coordinates
            denormalized_coordinates = draw_keypoints(frame, instance, threshold)
            # Draw the edges
            draw_edges(denormalized_coordinates, frame, EDGE_COLORS, threshold)
```

```
def draw_keypoints(frame, keypoints, threshold=0.11):
    """Draws the keypoints on a image frame"""

    # Denormalize the coordinates : multiply the normalized coordinates by the input_size(width,height)
    denormalized_coordinates = np.squeeze(np.multiply(keypoints, [WIDTH,HEIGHT,1]))
    #Iterate through the points
    for keypoint in denormalized_coordinates:
        # Unpack the keypoint values : y, x, confidence score
        keypoint_y, keypoint_x, keypoint_confidence = keypoint
        if keypoint_confidence > threshold:
```

```python
[9] def draw_keypoints(frame, keypoints, threshold=0.11):
        """Draws the keypoints on a image frame"""

        # Denormalize the coordinates : multiply the normalized coordinates by the input_size(width,height)
        denormalized_coordinates = np.squeeze(np.multiply(keypoints, [WIDTH,HEIGHT,1]))
        #Iterate through the points
        for keypoint in denormalized_coordinates:
            # Unpack the keypoint values : y, x, confidence score
            keypoint_y, keypoint_x, keypoint_confidence = keypoint
            if keypoint_confidence > threshold:
                """"""
                Draw the circle
                Note : A thickness of -1 px will fill the circle shape by the specified color.
                """"""
                cv2.circle(
                    img=frame,
                    center=(int(keypoint_x), int(keypoint_y)),
                    radius=4,
                    color=(255,0,0),
                    thickness=-1
                )
        return denormalized_coordinates
```

```python
def draw_edges(denormalized_coordinates, frame, edges_colors, threshold=0.11):
    """
    Draws the edges on a image frame
    """

    # Iterate through the edges
    for edge, color in edges_colors.items():
        # Get the dict value associated to the actual edge
        p1, p2 = edge
        # Get the points
        y1, x1, confidence_1 = denormalized_coordinates[p1]
        y2, x2, confidence_2 = denormalized_coordinates[p2]
        # Draw the line from point 1 to point 2, the confidence > threshold
        if (confidence_1 > threshold) & (confidence_2 > threshold):
```

```
[10] def draw_edges(denormalized_coordinates, frame, edges_colors, threshold=0.11):
        """
        Draws the edges on a image frame
        """

        # Iterate through the edges
        for edge, color in edges_colors.items():
            # Get the dict value associated to the actual edge
            p1, p2 = edge
            # Get the points
            y1, x1, confidence_1 = denormalized_coordinates[p1]
            y2, x2, confidence_2 = denormalized_coordinates[p2]
            # Draw the line from point 1 to point 2, the confidence > threshold
            if (confidence_1 > threshold) & (confidence_2 > threshold):
                cv2.line(
                    img=frame,
                    pt1=(int(x1), int(y1)),
                    pt2=(int(x2), int(y2)),
                    color=color,
                    thickness=2,
                    lineType=cv2.LINE_AA # Gives anti-aliased (smoothed) line which looks great for curves
                )
```

```
def progress(value, max=100):
    """
    Returns an HTML progress bar with a certain value. Used within each step
    """


    return HTML("""
      <progress
          value='{value}'
          max='{max}',
          style='width: 100%'
      >
          {value}
      </progress>
  """.format(value=value,
                  max=max))
```

```
    return HTML("""
      <progress
          value='{value}'
          max='{max}',
          style='width: 100%'
      >
          {value}
      </progress>
  """.format(value=value,
             max=max))
```

```
def load_gif():
    """
    Loads the gif and return its details
    """

    # Load the gif
    gif = cv2.VideoCapture("/content/ngannou.gif")
    # Get the frame count
    frame_count = int(gif.get(cv2.CAP_PROP_FRAME_COUNT))
    # Display parameter
    print(f"Frame count: {frame_count}")

    """""""
    Initialize the video writer
    We'll append each frame and its drawing to a vector, then stack all the frames to obtain a sequence (video).
    """
    output_frames = []

    # Get the initial shape (width, height)
    initial_shape = []
    initial_shape.append(int(gif.get(cv2.CAP_PROP_FRAME_WIDTH)))
    initial_shape.append(int(gif.get(cv2.CAP_PROP_FRAME_HEIGHT)))

    return gif, frame_count, output_frames, initial_shape
```

```
def run_inference():
    """
    Runs inferences then starts the main loop for each frame
    """

    # Load the gif
    gif, frame_count, output_frames, initial_shape = load_gif()
    # Set the progress bar to 0. It ranges from the first to the last frame
    bar = display(progress(0, frame_count-1), display_id=True)

    # Loop while the gif is opened
    while gif.isOpened():

        # Capture the frame
        ret, frame = gif.read()

        # Exit if the frame is empty
        if frame is None:
            break

        # Retrieve the frame index
        current_index = gif.get(cv2.CAP_PROP_POS_FRAMES)

        # Copy the frame
        image = frame.copy()
        image = cv2.resize(image, (WIDTH,HEIGHT))
        # Resize to the target shape and cast to an int32 vector
        input_image = tf.cast(tf.image.resize_with_pad(image, WIDTH, HEIGHT), dtype=tf.int32)
        # Create a batch (input tensor)
        input_image = tf.expand_dims(input_image, axis=0)

        # Perform inference
        results = movenet(input_image)
        """
        Output shape :  [1, 6, 56] ---> (batch size), (instances), (xy keypoints coordinates and score from [0:50]
        and [ymin, xmin, ymax, xmax, score] for the remaining elements)
        First, let's resize it to a more convenient shape, following this logic :
        - First channel ---> each instance
        - Second channel ---> 17 keypoints for each instance
        - The 51st values of the last channel ----> the confidence score.
```

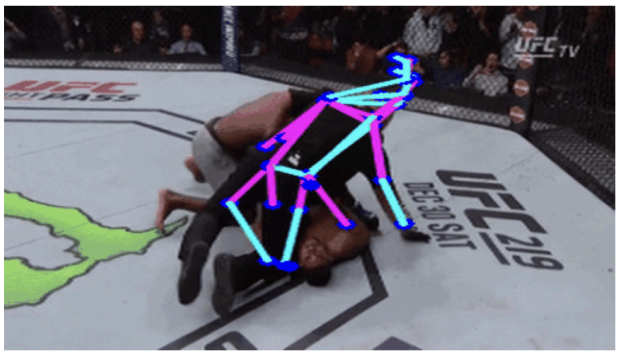
```
        """
        Output shape :  [1, 6, 56] ---> (batch size), (instances), (xy keypoints coordinates and score from [0:50]
        and [ymin, xmin, ymax, xmax, score] for the remaining elements)
        First, let's resize it to a more convenient shape, following this logic :
        - First channel ---> each instance
        - Second channel ---> 17 keypoints for each instance
        - The 51st values of the last channel ----> the confidence score.
        Thus, the Tensor is reshaped without losing important information.
        """

        keypoints = results["output_0"].numpy()[:,:,:51].reshape((6,17,3))

        # Loop through the results
        loop(image, keypoints, threshold=0.11)

        # Get the output frame : reshape to the original size
        frame_rgb = cv2.cvtColor(
            cv2.resize(
                image,(initial_shape[0], initial_shape[1]),
                interpolation=cv2.INTER_LANCZOS4
            ),
            cv2.COLOR_BGR2RGB # OpenCV processes BGR images instead of RGB
        )

        # Add the drawings to the output frames
        output_frames.append(frame_rgb)

        # Update the progress bar
        bar.update(progress(current_index, frame_count-1))

    # Release the object
    gif.release()

    print("Completed !")

    return output_frames
```

```
[ ] output_frames = run_inference()
```

```
Frame count: 113
Completed !
```

```
[ ] from tensorflow_docs.vis import embed
```

```
[ ] # Stack the output frames horizontally to compose a sequence
    output = np.stack(output_frames, axis=0)
    # Write the sequence to a gif
    imageio.mimsave("./animation.gif", output, fps=15)
    # Embed the output to the notebook
    embed.embed_file("./animation.gif")
```



On an image

```
[ ] def load_gif():
        """
        Loads the gif and return its details
        """

        # Load the gif
        gif = cv2.VideoCapture("/content/sample_data/image.jfif")
        # Get the frame count
        frame_count = int(gif.get(cv2.CAP_PROP_FRAME_COUNT))
        # Display parameter
        print(f"Frame count: {frame_count}")

        """""""
        Initialize the video writer
        We'll append each frame and its drawing to a vector, then stack all the frames to obtain a sequence (video
        """
        output_frames = []

        # Get the initial shape (width, height)
        initial_shape = []
        initial_shape.append(int(gif.get(cv2.CAP_PROP_FRAME_WIDTH)))
        initial_shape.append(int(gif.get(cv2.CAP_PROP_FRAME_HEIGHT)))

        return gif, frame_count, output_frames, initial_shape
```

```
[ ] output_frames = run_inference()
```

```
[ ] # Stack the output frames horizontally to compose a sequence
    output = np.stack(output_frames, axis=0)
    # Write the sequence to a gif
    imageio.mimsave("./image.jfif", output, fps=15)
    # Embed the output to the notebook
    embed.embed_file("./image.jfif")
```