MoveNet Refined

October 17, 2024

1 0. Install and Import Dependencies

[23]:
```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in /opt/anaconda3/lib/python3.8/site-packages
(24.2)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
!pip install tensorflow==2.4.1 opencv-python matplotlib
```

[1]:
```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import cv2
```

2 1. Load Model

[2]:
```
interpreter = tf.lite.
 ↪Interpreter(model_path='lite-model_movenet_singlepose_lightning_3.tflite')
interpreter.allocate_tensors()
```

3 2. Make Detections

[ ]:
```
# Initialize video capture object to access the webcam (change index if␣
 ↪necessary)
cap = cv2.VideoCapture(0)

# Loop runs continuously as long as the webcam is open
while cap.isOpened():
    # Capture frame-by-frame from the webcam
    ret, frame = cap.read()

    # Check if the frame was captured successfully
    if not ret:
        print("Failed to capture frame from webcam.")
        break
```

```
    # Make a copy of the frame to perform image transformations
    img = frame.copy()

    # Resize the image to match the input size expected by the model (192x192)
    # Expanding dimensions to fit the model input format (batch size of 1)
    img = tf.image.resize_with_pad(np.expand_dims(img, axis=0), 192, 192)

    # Convert image to float32 to match the expected model input data type
    input_image = tf.cast(img, dtype=tf.float32)

    # Get input and output tensor details from the TFLite model interpreter
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    # Set the input tensor for the model (feed the preprocessed image)
    interpreter.set_tensor(input_details[0]['index'], np.array(input_image))

    # Run inference (make predictions using the model)
    interpreter.invoke()

    # Extract the output predictions (keypoints and their associated scores)
    keypoints_with_scores = interpreter.get_tensor(output_details[0]['index'])

    # Drawing detected keypoints and connecting lines on the original frame
    # `EDGES` defines body part connections, 0.4 is the confidence threshold
    draw_connections(frame, keypoints_with_scores, EDGES, 0.4)
    draw_keypoints(frame, keypoints_with_scores, 0.4)

    # Display the processed frame with the keypoints and connections overlaid
    cv2.imshow('Capstone AI Running Coach: Move // Run', frame)

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

# Release the webcam resource when done
cap.release()

# Close all OpenCV windows
cv2.destroyAllWindows()
```

4 3. Draw Keypoints

[5]:
```
def draw_keypoints(frame, keypoints, confidence_threshold):
    # Get the dimensions of the frame (height, width, and channels)
    y, x, c = frame.shape

    # Squeeze the keypoints array to remove unnecessary dimensions and scale
    # the keypoint coordinates to the image size (multiplying y and x)
    shaped = np.squeeze(np.multiply(keypoints, [y, x, 1]))

    # Loop through each keypoint (ky for y-coordinate, kx for x-coordinate,␣
 ↪kp_conf for confidence)
    for kp in shaped:
        ky, kx, kp_conf = kp

        # Check if the keypoint's confidence is above the threshold before␣
 ↪drawing
        if kp_conf > confidence_threshold:
            # Draw a small green circle at the keypoint's (x, y) coordinates on␣
 ↪the frame
            cv2.circle(frame, (int(kx), int(ky)), 4, (0, 255, 0), -1)  # Radius␣
 ↪4, filled circle (-1)
```

5 4. Draw Edges

[4]:
```
# Define edges (connections) between body keypoints and the color to draw each␣
 ↪connection
# The key is a tuple (point A, point B) representing the indices of two␣
 ↪connected keypoints
# The value is a string representing the color used to draw the edge (m for␣
 ↪magenta, c for cyan, y for yellow)
EDGES = {
    (0, 1): 'm',   # Nose to left eye
    (0, 2): 'c',   # Nose to right eye
    (1, 3): 'm',   # Left eye to left ear
    (2, 4): 'c',   # Right eye to right ear
    (0, 5): 'm',   # Nose to left shoulder
    (0, 6): 'c',   # Nose to right shoulder
    (5, 7): 'm',   # Left shoulder to left elbow
    (7, 9): 'm',   # Left elbow to left wrist
    (6, 8): 'c',   # Right shoulder to right elbow
    (8, 10): 'c',  # Right elbow to right wrist
    (5, 6): 'y',   # Left shoulder to right shoulder
    (5, 11): 'm',  # Left shoulder to left hip
    (6, 12): 'c',  # Right shoulder to right hip
    (11, 12): 'y', # Left hip to right hip
```

```
    (11, 13): 'm', # Left hip to left knee
    (13, 15): 'm', # Left knee to left ankle
    (12, 14): 'c', # Right hip to right knee
    (14, 16): 'c'  # Right knee to right ankle
}
```

[8]:
```
def draw_connections(frame, keypoints, edges, confidence_threshold):
    # Get the dimensions of the frame (height, width, channels)
    y, x, c = frame.shape

    # Reshape and scale the keypoints to match the frame dimensions
    shaped = np.squeeze(np.multiply(keypoints, [y, x, 1]))

    # Iterate through the edges dictionary, which contains pairs of keypoints␣
 ↪and their assigned color
    for edge, color in edges.items():
        # Unpack the keypoint indices from the edge tuple
        p1, p2 = edge
        print(p1,p2) #printing joint connection

        # Get the coordinates and confidence scores for both keypoints in the␣
 ↪edge
        y1, x1, c1 = shaped[p1]
        y2, x2, c2 = shaped[p2]
        print(int(x2), int(y2))

        # Check if both keypoints have confidence scores above the threshold
        if (c1 > confidence_threshold) & (c2 > confidence_threshold):
            # Draw a line connecting the two keypoints on the frame
            # Using a blue color (255,0,0) and a line thickness of 2
            cv2.line(frame, (int(x1), int(y1)), (int(x2), int(y2)), (255, 0,␣
 ↪0), 2)
```