```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!pip install mediapipe
!pip install opencv-python
import cv2
```

Show hidden output

```
import os
import sys
root_path='/content/drive/My Drive'
os.chdir(root_path)
sys.path.append('/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms')
```

```
import csv
#import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Train and Test angle CSV data reading

```
test_angle=pd.read_csv("/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/test_ang
test_angle
train_angle=pd.read_csv("/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/train_a
train_angle
```

	left_wrist_angle	right_wrist_angle	left_elbow_angle	right_elbow_angle	left_shoulder_angle	right_shoulder_angle	left_knee
0	209.124053	198.305613	190.681518	192.153146	167.552376	188.956532	183
1	210.963757	211.429566	197.710743	196.099713	182.366604	171.215813	176
2	213.332197	213.164652	181.746842	185.259076	184.486644	173.135858	132
3	210.972835	191.259688	189.751718	197.366763	180.097624	177.883081	180
4	158.601281	144.855757	170.407834	174.575282	168.859375	190.196421	183
...	...	...	...	...	...	...	
1040	129.973890	193.730216	123.986167	201.989739	50.666155	62.792578	175
1041	187.594643	177.709390	174.090041	178.054215	89.120914	106.765203	170
1042	191.309932	176.143199	174.289407	185.133884	91.414423	114.095335	173
1043	177.881082	181.647029	179.910110	182.620865	93.111923	107.836581	176
1044	173.586696	177.604994	175.793649	176.054814	264.892651	266.478547	185

1045 rows × 13 columns

```
from sklearn.svm import SVC
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

:Training a machine learning model, specifically a Support Vector Classifier

1. *List item*
2. List item

(SVC), for a classification task.

```
data_train = pd.read_csv("/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/train_a
data_test = pd.read_csv("/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/test_ang

X, Y = data_train.iloc[:, :data_train.shape[1] - 1], data_train['target']

model = SVC(kernel='rbf', decision_function_shape='ovo',probability=True)
```

```
model.fit(X, Y)
```

```
SVC
SVC(decision_function_shape='ovo', probability=True)
```

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


# Separate features (X) and target (Y) from training data
X_train = data_train.iloc[:, :data_train.shape[1] - 1]
Y_train = data_train['target']

# Separate features (X) and target (Y) from testing data
X_test = data_test.iloc[:, :data_test.shape[1] - 1]
Y_test = data_test['target']

# Initialize SVM model
model = SVC(kernel='rbf', decision_function_shape='ovo', probability=True)

# Train the model
model.fit(X_train, Y_train)

# Predict on the test data
Y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 97.85%
```

Performing the evaluation of a machine learning model using a test dataset and visualizing the results using a confusion matrix.

```
import mediapipe as mp
import cv2
import pandas as pd
import os
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

mp_pose = mp.solutions.pose
pose = mp_pose.Pose()
points = mp_pose.PoseLandmark  # Landmarks
mp_drawing = mp.solutions.drawing_utils # For drawing keypoints


def calculate_angle(landmark1, landmark2, landmark3):
    x1, y1, _ = landmark1.x, landmark1.y, landmark1.z
    x2, y2, _ = landmark2.x, landmark2.y, landmark2.z
    x3, y3, _ = landmark3.x, landmark3.y, landmark3.z

    angle = np.degrees(np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2))

    # Check if the angle is less than zero.
    if angle < 0:
        # Add 360 to the found angle.
        angle += 360

    return angle


def extract_pose_angles(results):
    angles = []

    if results.pose_landmarks is not None:
        landmarks = results.pose_landmarks.landmark
        # Get the angle between the left elbow, wrist and left index points.
        left_wrist_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                           landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value],
                                           landmarks[mp_pose.PoseLandmark.LEFT_INDEX.value])
        angles.append(left_wrist_angle)
        # Get the angle between the right elbow, wrist and left index points.
```

```
            right_wrist_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value],
                                                landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value],
                                                landmarks[mp_pose.PoseLandmark.RIGHT_INDEX.value])
            angles.append(right_wrist_angle)


            # Get the angle between the left shoulder, elbow and wrist points.
            left_elbow_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                               landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                               landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value])
            angles.append(left_elbow_angle)
            # Get the angle between the right shoulder, elbow and wrist points.
            right_elbow_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                                                landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value],
                                                landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value])
            angles.append(right_elbow_angle)
            # Get the angle between the left elbow, shoulder and hip points.
            left_shoulder_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                                  landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                                  landmarks[mp_pose.PoseLandmark.LEFT_HIP.value])
            angles.append(left_shoulder_angle)

            # Get the angle between the right hip, shoulder and elbow points.
            right_shoulder_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                                   landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                                                   landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value])
            angles.append(right_shoulder_angle)

            # Get the angle between the left hip, knee and ankle points.
            left_knee_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                              landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value],
                                              landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value])
            angles.append(left_knee_angle)

            # Get the angle between the right hip, knee and ankle points
            right_knee_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                               landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value],
                                               landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value])
            angles.append(right_knee_angle)

            # Get the angle between the left hip, ankle and LEFT_FOOT_INDEX points.
            left_ankle_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                               landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value],
                                               landmarks[mp_pose.PoseLandmark.LEFT_FOOT_INDEX.value])
            angles.append(left_ankle_angle)

            # Get the angle between the right hip, ankle and RIGHT_FOOT_INDEX points
            right_ankle_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                               landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value],
                                               landmarks[mp_pose.PoseLandmark.RIGHT_FOOT_INDEX.value])
            angles.append(right_ankle_angle)

            # Get the angle between the left knee, hip and right hip points.
            left_hip_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value],
                                             landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                             landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value])
            angles.append(left_hip_angle)

            # Get the angle between the left hip, right hip and right kneee points
            right_hip_angle = calculate_angle(landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                              landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                              landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value])
            angles.append(right_hip_angle)
        return angles




    def evaluate(data_test, model, show=False):
            target = data_test.loc[:, "target"]  # list of labels
            target = target.values.tolist()
            predictions = []
            for i in range(len(data_test)):
                    tmp = data_test.iloc[i, 0:len(data_test.columns) - 1]
                    tmp = tmp.values.tolist()
                    predictions.append(model.predict([tmp])[0])
            if show:
                    print(confusion_matrix(predictions, target), '\n')
                    print(classification_report(predictions, target))
            return predictions
```

```
# Test phase : build test dataset then evaluate
predictions = evaluate(data_test, model, show=True)

#Create a confusion matrix
cm = confusion_matrix(data_test['target'], predictions)

# Display the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Show hidden output

Prediction of image

```
# Predict the name of the poses in the image
def predict(img, model, show=False):
        img = cv2.imread(img)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #Resize the image to 50% of the original size
        scale_factor = 0.5
        img = cv2.resize(img, None, fx=scale_factor, fy=scale_factor)
        results = pose.process(img)

        if results.pose_landmarks:
                list_angles = []
                list_angles = extract_pose_angles(results)
                y = model.predict([list_angles])

                if show:
                        mp_drawing.draw_landmarks(img, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
                        cv2.putText(img, str(y[0]), (50,50), cv2.FONT_HERSHEY_SIMPLEX,1,(215,215,0),3)
                        plt.imshow(img) #cv2_imshow("image", img)
                        #cv2.waitKey(0)
```

```
predict('/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/teacher_yoga/goddess.jpe
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but SVC
  warnings.warn(
```

Yoga pose Prediction of the video

```
from google.colab.patches import cv2_imshow
import os
from moviepy.editor import ImageSequenceClip
import os
!pip install Pillow
from PIL import Image


def predict_video02(model, video="0", show=False):
    m = 0
    cap = cv2.VideoCapture(video)
```

```
output_folder_path = '/content/drive/My Drive/DETECTION-OUTPUT'  # Define output folder path
video_name = os.path.splitext(os.path.basename(video))[0]  # Extract video file name without extension
sc='/'

# Create a unique folder for each video
video_folder_path = os.path.join(output_folder_path, video_name)
if not os.path.exists(video_folder_path):
    os.makedirs(video_folder_path)
img_dir2 = output_folder_path+sc+video_name
print(video_folder_path)

while cap.isOpened():
    angles = []
    success, img = cap.read()
    if not success:
        print("Ignoring empty camera frame.")
        break
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = pose.process(img)
    if results.pose_landmarks:
        list_angles = extract_pose_angles(results)
        y = model.predict([list_angles])
        name = str(y[0])
        if show:
            mp_drawing.draw_landmarks(img, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
            (w, h), _ = cv2.getTextSize(name, cv2.FONT_HERSHEY_SIMPLEX, 1, 1)
            cv2.rectangle(img, (40, 40), (40+w, 60), (255, 255, 255), cv2.FILLED)
            cv2.putText(img, name, (40, 60), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 3)
            # cv2_imshow(img)
            if cv2.waitKey(5) & 0xFF == 27:
                break
        if m > 0:
            i_formatted = '{:03d}'.format(m)
            img_name1 = f"output_image_{i_formatted}.jpg"
            img_path1 = os.path.join(video_folder_path, img_name1)  # Use video_folder_path
            cv2.imwrite(img_path1, img)
        m = m + 1
        print(m)
        # if m==10:
        #   break

# folder=folder
# os.makedirs(video_folder_path, exist_ok=True)
video_folder_path1 = os.path.dirname(video_folder_path)
def compute_average_dimensions(video_folder_path1):
    total_width = 0
    total_height = 0
    img_count = 0
    print(video_folder_path1)

    for img_file in os.listdir(video_folder_path1):
        if img_file.endswith((".jpg", ".jpeg", ".png")):

            image = Image.open(os.path.join(video_folder_path1, img_file))
            w, h = image.size
            total_width += w
            total_height += h
            img_count += 1

    avg_width = int(total_width / img_count)
    avg_height = int(total_height / img_count)
    return avg_width, avg_height

# Calculate average dimensions of images
avg_width, avg_height = compute_average_dimensions(img_dir2)

def create_video_from_images(folder, video_filename='video_name.mp4'):
    # Get list of image files in the folder

    valid_images = [i for i in os.listdir(folder) if i.endswith((".jpg", ".jpeg", ".png"))]

    # Sort images based on filename
    valid_images.sort()

    # Create list of image paths
    image_paths = [os.path.join(folder, img_name) for img_name in valid_images]

    # Create video from image sequence
    clip = ImageSequenceClip(image_paths, fps=12)  # Adjust fps as needed

    # Get the first frame
    first_frame = image_paths[0]
```

```
        # Duplicate the first frame to match the desired duration
        num_duplicates = max(1, int(clip.fps * (1 - len(image_paths) / clip.fps)))
        image_paths = [first_frame] * num_duplicates + image_paths

        # Create video from image sequence
        clip = ImageSequenceClip(image_paths, fps=12)  # Adjust fps as needed

        # Save the video
        video_path = os.path.join(folder, video_filename)
        clip.write_videofile(video_path, codec='libx264')

        # Print the name of each image appended in the video
        print("Names of images appended in the video:")
        for img_path in image_paths[num_duplicates:]:
            img_name = os.path.basename(img_path)
        return video_path
        # Replace 'img_dir' with the directory containing your images
    video_path = create_video_from_images(img_dir2)
```

```
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (11.1.0)
```

```
predict_video02(model,'/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/video4.mp4
```

Show hidden output

Yoga Pose Correction of the image

```
import cv2
import mediapipe as mp
import numpy as np
import csv
import os

# Create a pose instance
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

def calculate_angle(landmark1, landmark2, landmark3, select=1):
    angle=None
    if select == '1':
        x1, y1, _ = landmark1.x, landmark1.y, landmark1.z
        x2, y2, _ = landmark2.x, landmark2.y, landmark2.z
        x3, y3, _ = landmark3.x, landmark3.y, landmark3.z

        angle = np.degrees(np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2))
    # else:
    #     x1, y1 = landmark1.x, landmark1.y
    #     x2, y2 = landmark2.x, landmark2.y
    #     x3, y3 = landmark3.x, landmark3.y

    #     radians = np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2)
    #     angle = np.abs(np.degrees(radians))
    if angle is None:
      return 0
    angle_calc = angle + 360 if angle < 0 else angle
    return angle_calc

def correct_feedback_image(model, image_path, input_csv='0'):
    angle_name_list = ["L-wrist","R-wrist","L-elbow", "R-elbow","L-shoulder", "R-shoulder", "L-knee", "R-knee","L-ankle","R-ankle","L-h
    angle_coordinates = [[13, 15, 19], [14, 16, 18], [11, 13, 15], [12, 14, 16], [13, 11, 23], [14, 12, 24], [23, 25, 27], [24, 26, 28]
    correction_value = 30
    image = cv2.imread(image_path)

    target_width=1500
    target_height=1200
    # Resize image to target dimensions
    image = cv2.resize(image, (target_width, target_height))


    #image = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)  # Reduce image size by 50%
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = pose.process(image_rgb)

    if results.pose_landmarks is not None:
```

```
            landmarks = results.pose_landmarks.landmark
            angles = []

            for itr in range(12):
                point_a = landmarks[angle_coordinates[itr][0]]
                point_b = landmarks[angle_coordinates[itr][1]]
                point_c = landmarks[angle_coordinates[itr][2]]

                angle_obtained = calculate_angle(point_a, point_b, point_c)

                angles.append(angle_obtained)

            y = model.predict([angles])
            Name_Yoga_Classification = str(y[0])

            probabilities = model.predict_proba([angles])
            class_labels = model.classes_
            check_accry_class = False

            for i, class_label in enumerate(class_labels):
                probability = probabilities[0, i]
                if probability > 0.5:
                    check_accry_class = True
                else:
                    continue

            with open(input_csv, 'r') as inputCSV:
                for row in csv.reader(inputCSV):
                    if row[12] == Name_Yoga_Classification:
                        accurate_angle_lists = [float(row[0]), float(row[1]), float(row[2]), float(row[3]), float(row[4]), float(row[5]), f

            if check_accry_class:
                (w, h), _ = cv2.getTextSize(Name_Yoga_Classification, cv2.FONT_HERSHEY_SIMPLEX, 2, 4)
                cv2.rectangle(image, (10, image.shape[0] - 60), (10 + w, image.shape[0] - 20), (255, 255, 255), cv2.FILLED)
                cv2.putText(image, Name_Yoga_Classification, (10, image.shape[0] - 30), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 4)

            else:
                (w, h), _ = cv2.getTextSize('None', cv2.FONT_HERSHEY_SIMPLEX, 2, 4)
                cv2.rectangle(image, (10, image.shape[0] - 60), (10 + w, image.shape[0] - 20), (255, 255, 255), cv2.FILLED)
                cv2.putText(image, 'None', (10, image.shape[0] - 30), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 4)

            correct_angle_count = 0
            for itr in range(12):
                point_a = landmarks[angle_coordinates[itr][0]]
                point_b = landmarks[angle_coordinates[itr][1]]
                point_c = landmarks[angle_coordinates[itr][2]]

                angle_obtained = calculate_angle(point_a, point_b, point_c, '1')

                if angle_obtained < accurate_angle_lists[itr] - correction_value:
                    status = "more"
                elif accurate_angle_lists[itr] + correction_value < angle_obtained:
                    status = "less"
                else:
                    status = "OK"
                    correct_angle_count += 1

                status_position = (int(point_b.x * image.shape[1]) - int(image.shape[1] * 0.03), int(point_b.y * image.shape[0]) + int(image
                cv2.putText(image, f"{status}", status_position, cv2.FONT_HERSHEY_PLAIN, 3, (0, 255, 0), 6)
                cv2.putText(image, f"{angle_name_list[itr]}", (int(point_b.x * image.shape[1]) - 100, int(point_b.y * image.shape[0]) - 30),

            mp_drawing = mp.solutions.drawing_utils
            annotated_image = image.copy()
            mp_drawing.draw_landmarks(annotated_image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

            image1 = cv2.resize(annotated_image, (0, 0), fx=0.5, fy=0.5)
            cv2_imshow(image1)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        else:
            print("No pose landmarks found in the image.")
```

```
input_csv = '/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/teacher_yoga/angle_t
```

```
image_path = '/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/teacher_yoga/warri

correct_feedback_image(model, image_path, input_csv)
```

Show hidden output

Yoga Pose Correction for Video

```
import cv2
import mediapipe as mp
import numpy as np
import time
import csv
import os

# Create a pose instance
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

from google.colab import drive
import cv2
import os

# Mount Google Drive
#drive.mount('/content/drive')

# Define output folder path
output_folder_path1 = '/content/drive/My Drive/CORRECTION-OUTPUT'

# Function to calculate angle between three points
def calculate_angle(landmark1, landmark2, landmark3, select = 1):
    angle = None
    if select == '1':
        x1, y1, _ = landmark1.x, landmark1.y, landmark1.z
        x2, y2, _ = landmark2.x, landmark2.y, landmark2.z
        x3, y3, _ = landmark3.x, landmark3.y, landmark3.z

        angle = np.degrees(np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2))
    # else:
    #     x1, y1 = landmark1.x, landmark1.y
    #     x2, y2 = landmark2.x, landmark2.y
    #     x3, y3 = landmark3.x, landmark3.y

    #     radians = np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2)
    #     angle = np.abs(np.degrees(radians))
    if angle is None:
      return 0

    angle_calc = angle + 360 if angle < 0 else angle
    return angle_calc

def correct_feedback(model, video='0', input_csv='0'):
    # Load video
    cap = cv2.VideoCapture(video)  # Replace with your video path
    output_folder_path2 = '/content/drive/My Drive/CORRECTION-OUTPUT'  # Define output folder path
    video_name1 = os.path.splitext(os.path.basename(video))[0]
    sc='/' # Extract video file name without extension
    video_folder_path2 = os.path.join(output_folder_path2, video_name1)
    if not os.path.exists(video_folder_path2):
        os.makedirs(video_folder_path2)
    img_dir3 = output_folder_path2+sc+video_name1


    if cap.isOpened() is False:
        print("Error opening video stream or file")

    accurate_angle_lists = []

    angle_name_list = ["L-wrist","R-wrist","L-elbow", "R-elbow","L-shoulder", "R-shoulder", "L-knee", "R-knee","L-ankle","R-ankle","L-h
    angle_coordinates = [[13, 15, 19], [14, 16, 18], [11, 13, 15], [12, 14, 16], [13, 11, 23], [14, 12, 24], [23, 25, 27], [24, 26, 28]
    correction_value = 30

    fps_time = 0
    k=0
    while cap.isOpened():
        ret_val, image = cap.read()
        # print(image.shape)

        if not ret_val:
            break
```

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
resize_rgb = cv2.resize(image_rgb, (0, 0), None, .50, .50)

results = pose.process(image_rgb)

angles = []

if results.pose_landmarks is not None:
    landmarks = results.pose_landmarks.landmark

    for itr in range(12):
        point_a = landmarks[angle_coordinates[itr][0]]
        point_b = landmarks[angle_coordinates[itr][1]]
        point_c = landmarks[angle_coordinates[itr][2]]

        angle_obtained = calculate_angle(point_a, point_b, point_c)

        angles.append(angle_obtained)

    y = model.predict([angles])

    Name_Yoga_Classification = str(y[0])

    probabilities = model.predict_proba([angles])

    class_labels = model.classes_
    check_accry_class = False

    for i,class_label in enumerate(class_labels):
        probability = probabilities[0, i]
        if probability > 0.5 :
            check_accry_class = True
        else:
            continue

    with open(input_csv, 'r') as inputCSV:
        for row in csv.reader(inputCSV):
            if row[12] == Name_Yoga_Classification:
                accurate_angle_lists = [float(row[0]), float(row[1]), float(row[2]), float(row[3]), float(row[4]), float(row[5])

    if check_accry_class == True :
        (w, h), _ = cv2.getTextSize(Name_Yoga_Classification, cv2.FONT_HERSHEY_SIMPLEX, 1, 1)
        cv2.rectangle(image, (10, image.shape[0] - 30), (10 + w, image.shape[0] - 10), (255, 255, 255), cv2.FILLED)
        cv2.putText(image, Name_Yoga_Classification, (10, image.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 3)

    else :
        (w, h), _ = cv2.getTextSize('None', cv2.FONT_HERSHEY_SIMPLEX, 1, 1)
        cv2.rectangle(image, (10, image.shape[0] - 30), (10 + w, image.shape[0] - 10), (255, 255, 255), cv2.FILLED)
        cv2.putText(image, 'None', (10, image.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 3)


    correct_angle_count = 0
    for itr in range(12):
        point_a = landmarks[angle_coordinates[itr][0]]
        point_b = landmarks[angle_coordinates[itr][1]]
        point_c = landmarks[angle_coordinates[itr][2]]

        angle_obtained = calculate_angle(point_a, point_b, point_c, '1')

        if angle_obtained < accurate_angle_lists[itr] - correction_value:
            status = "more"
        elif accurate_angle_lists[itr] + correction_value < angle_obtained:
            status = "less"
        else:
            status = "OK"
            correct_angle_count += 1

        status_position = (int(point_b.x * image.shape[1]) - int(image.shape[1] * 0.03), int(point_b.y * image.shape[0]) + int(
        cv2.putText(image, f"{status}", status_position, cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)

        cv2.putText(image, f"{angle_name_list[itr]}", (int(point_b.x * image.shape[1]) - 50, int(point_b.y * image.shape[0]) - 1

    mp_drawing = mp.solutions.drawing_utils
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

    posture = "CORRECT" if correct_angle_count > 9 else "WRONG"
    posture_color = (0, 255, 0) if posture == "CORRECT" else (0, 0, 255)

    posture_position = (10, 30)
    cv2.putText(image, f"Yoga movements: {posture}", posture_position, cv2.FONT_HERSHEY_PLAIN, 1.5, posture_color, 2)
```

```
            fps_text = f"FPS: {1.0 / (time.time() - fps_time):.3f}"
            fps_position = (10, 60)
            cv2.putText(image, fps_text, fps_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            #cv2_imshow(image)

            fps_time = time.time()

        if cv2.waitKey(1) == 27:
            break
        #for i, img in enumerate(image):
        # print(image.shape)

        if k > 0:
          i_formatted = '{:03d}'.format(k)
          img_name = f"output_image_{i_formatted}.jpg"
          img_path = os.path.join(video_folder_path2, img_name)
          cv2.imwrite(img_path, image)
        k=k+1
        print(k)
    cap.release()
    cv2.destroyAllWindows()
    video_folder_path2 = os.path.dirname(video_folder_path2)
    def compute_average_dimensions(video_folder_path2):
        total_width = 0
        total_height = 0
        img_count = 0
        print(video_folder_path2)

        for img_file in os.listdir(video_folder_path2):
            if img_file.endswith((".jpg", ".jpeg", ".png")):

                image = Image.open(os.path.join(video_folder_path2, img_file))
                w, h = image.size
                total_width += w
                total_height += h
                img_count += 1

        avg_width = int(total_width / img_count)
        avg_height = int(total_height / img_count)
        return avg_width, avg_height

    # Calculate average dimensions of images
    avg_width, avg_height = compute_average_dimensions(img_dir3)

    def create_video_from_images(folder, video_filename='video_name.mp4'):
        # Get list of image files in the folder

        valid_images = [i for i in os.listdir(folder) if i.endswith((".jpg", ".jpeg", ".png"))]

        # Sort images based on filename
        valid_images.sort()

        # Create list of image paths
        image_paths = [os.path.join(folder, img_name) for img_name in valid_images]

        # Create video from image sequence
        clip = ImageSequenceClip(image_paths, fps=12)  # Adjust fps as needed

        # Get the first frame
        first_frame = image_paths[0]

        # Duplicate the first frame to match the desired duration
        num_duplicates = max(1, int(clip.fps * (1 - len(image_paths) / clip.fps)))
        image_paths = [first_frame] * num_duplicates + image_paths

        # Create video from image sequence
        clip = ImageSequenceClip(image_paths, fps=12)  # Adjust fps as needed

        # Save the video
        video_path = os.path.join(folder, video_filename)
        clip.write_videofile(video_path, codec='libx264')

        # Print the name of each image appended in the video
        print("Names of images appended in the video:")
        for img_path in image_paths[num_duplicates:]:
            img_name = os.path.basename(img_path)
        return video_path
        # Replace 'img_dir' with the directory containing your images
    video_path = create_video_from_images(img_dir3)
```

```
correct_feedback(model,'/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/downdog_
```

Show hidden output

END

```
# import cv2
# import mediapipe as mp
# import numpy as np
# import csv
# import os

# # Create a pose instance
# mp_pose = mp.solutions.pose
# pose = mp_pose.Pose()

# def calculate_angle(landmark1, landmark2, landmark3, select=1):
#     angle=None
#     if select == '1':
#         x1, y1, _ = landmark1.x, landmark1.y, landmark1.z
#         x2, y2, _ = landmark2.x, landmark2.y, landmark2.z
#         x3, y3, _ = landmark3.x, landmark3.y, landmark3.z

#         angle = np.degrees(np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2))
#     # else:
#     #     x1, y1 = landmark1.x, landmark1.y
#     #     x2, y2 = landmark2.x, landmark2.y
#     #     x3, y3 = landmark3.x, landmark3.y

#     #     radians = np.arctan2(y3 - y2, x3 - x2) - np.arctan2(y1 - y2, x1 - x2)
#     #     angle = np.abs(np.degrees(radians))
#     if angle is None:
#       return 0
#     angle_calc = angle + 360 if angle < 0 else angle
#     return angle_calc

# def correct_feedback_image(model, image_path, input_csv='0'):
#     angle_name_list = ["L-wrist","R-wrist","L-elbow", "R-elbow","L-shoulder", "R-shoulder", "L-knee", "R-knee","L-ankle","R-ankle","L-
#     angle_coordinates = [[13, 15, 19], [14, 16, 18], [11, 13, 15], [12, 14, 16], [13, 11, 23], [14, 12, 24], [23, 25, 27], [24, 26, 28
#     correction_value = 30
#     image = cv2.imread(image_path)

#     target_width=1500
#     target_height=1200
#     # Resize image to target dimensions
#     image = cv2.resize(image, (target_width, target_height))


#     #image = cv2.resize(image, (0, 0), fx=0.5, fy=0.5)  # Reduce image size by 50%
#     image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
#     results = pose.process(image_rgb)

#     if results.pose_landmarks is not None:
#         landmarks = results.pose_landmarks.landmark
#         angles = []

#         for itr in range(12):
#             point_a = landmarks[angle_coordinates[itr][0]]
#             point_b = landmarks[angle_coordinates[itr][1]]
#             point_c = landmarks[angle_coordinates[itr][2]]

#             angle_obtained = calculate_angle(point_a, point_b, point_c)

#             angles.append(angle_obtained)

#         y = model.predict([angles])
#         Name_Yoga_Classification = str(y[0])

#         probabilities = model.predict_proba([angles])
#         class_labels = model.classes_
#         check_accry_class = False

#         for i, class_label in enumerate(class_labels):
#             probability = probabilities[0, i]
```

```
#             if probability > 0.5:
#                 check_accry_class = True
#             else:
#                 continue

#         with open(input_csv, 'r') as inputCSV:
#             for row in csv.reader(inputCSV):
#                 if row[12] == Name_Yoga_Classification:
#                     accurate_angle_lists = [float(row[0]), float(row[1]), float(row[2]), float(row[3]), float(row[4]), float(row[5]),

#         if check_accry_class:
#             (w, h), _ = cv2.getTextSize(Name_Yoga_Classification, cv2.FONT_HERSHEY_SIMPLEX, 2, 4)
#             cv2.rectangle(image, (10, image.shape[0] - 60), (10 + w, image.shape[0] - 20), (255, 255, 255), cv2.FILLED)
#             cv2.putText(image, Name_Yoga_Classification, (10, image.shape[0] - 30), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 4)

#         else:
#             (w, h), _ = cv2.getTextSize('None', cv2.FONT_HERSHEY_SIMPLEX, 2, 4)
#             cv2.rectangle(image, (10, image.shape[0] - 60), (10 + w, image.shape[0] - 20), (255, 255, 255), cv2.FILLED)
#             cv2.putText(image, 'None', (10, image.shape[0] - 30), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 0), 4)

#         correct_angle_count = 0
#         for itr in range(12):
#             point_a = landmarks[angle_coordinates[itr][0]]
#             point_b = landmarks[angle_coordinates[itr][1]]
#             point_c = landmarks[angle_coordinates[itr][2]]

#             angle_obtained = calculate_angle(point_a, point_b, point_c, '1')

#             if angle_obtained < accurate_angle_lists[itr] - correction_value:
#                 status = "more"
#             elif accurate_angle_lists[itr] + correction_value < angle_obtained:
#                 status = "less"
#             else:
#                 status = "OK"
#                 correct_angle_count += 1

#             status_position = (int(point_b.x * image.shape[1]) - int(image.shape[1] * 0.03), int(point_b.y * image.shape[0]) + int(ima
#             cv2.putText(image, f"{status}", status_position, cv2.FONT_HERSHEY_PLAIN, 3, (0, 255, 0), 6)
#             cv2.putText(image, f"{angle_name_list[itr]}", (int(point_b.x * image.shape[1]) - 100, int(point_b.y * image.shape[0]) - 3(

#         mp_drawing = mp.solutions.drawing_utils
#         annotated_image = image.copy()
#         mp_drawing.draw_landmarks(annotated_image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

#         image1 = cv2.resize(annotated_image, (0, 0), fx=0.5, fy=0.5)
#         cv2_imshow(image1)
#         cv2.waitKey(0)
#         cv2.destroyAllWindows()
#     else:
#         print("No pose landmarks found in the image.")




# input_csv = '/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/teacher_yoga/angle
# image_path = '/content/drive/MyDrive/Detect-Yoga-Poses-And-Correction-In-Real-Time-Using-Machine-Learning-Algorithms/teacher_yoga/warr
# correct_feedback_image(model, image_path, input_csv)
```