PERSONAL AI YOGA TRAINER

Abstract:

Yoga has gained global recognition due to the rising levels of stress in modern lifestyles. There are numerous resources available today for learning yoga, which emphasizes the deep connection between the mind and body. Extensive medical and scientific evidence now supports the idea that practicing various forms of yoga can bring about significant changes in our brain activity, body chemistry, and even genetic makeup.

Our main aim was to emphasize the importance of Practicing Yoga

To analyze yoga postures, the Mediapipe Library is imported. Advanced software is employed to detect the standing posture in real-time as individuals perform yoga poses in front of a camera.

Code Overview

Dependencies:

The following libraries are used to run the code:

- `mediapipe`: Used for pose detection.
- `cv2` (OpenCV): Used for image and video processing.
- `numpy`: Used for numerical operations.
- `pandas`: Used for data manipulation.
- `seaborn`: Used for data visualization.
- `csv`: Used for reading and writing CSV files.
- `os`: Used for file operations.
- `pickle`: Used for serializing and deserializing Python objects.

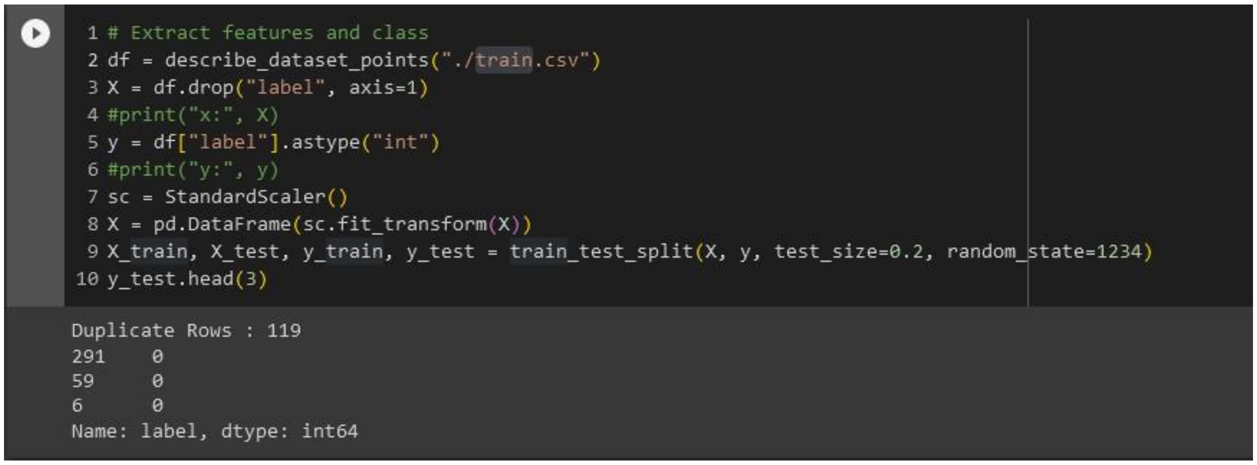
- `scikit-learn`: Used for machine learning models and utilities.

Usage:

To use this code:

1. Install the required dependencies mentioned above.
2. Sequence of the files:
    - Getting_data_in_csv
    - Training_model
    - Live_Pose_Test__final
3. Make sure you have a video file that contains yoga pose sequences. Update the `cap = cv2.VideoCapture("goddess-pose.mp4")` line to specify the path to your video file.
4. Run the code.

    - Training



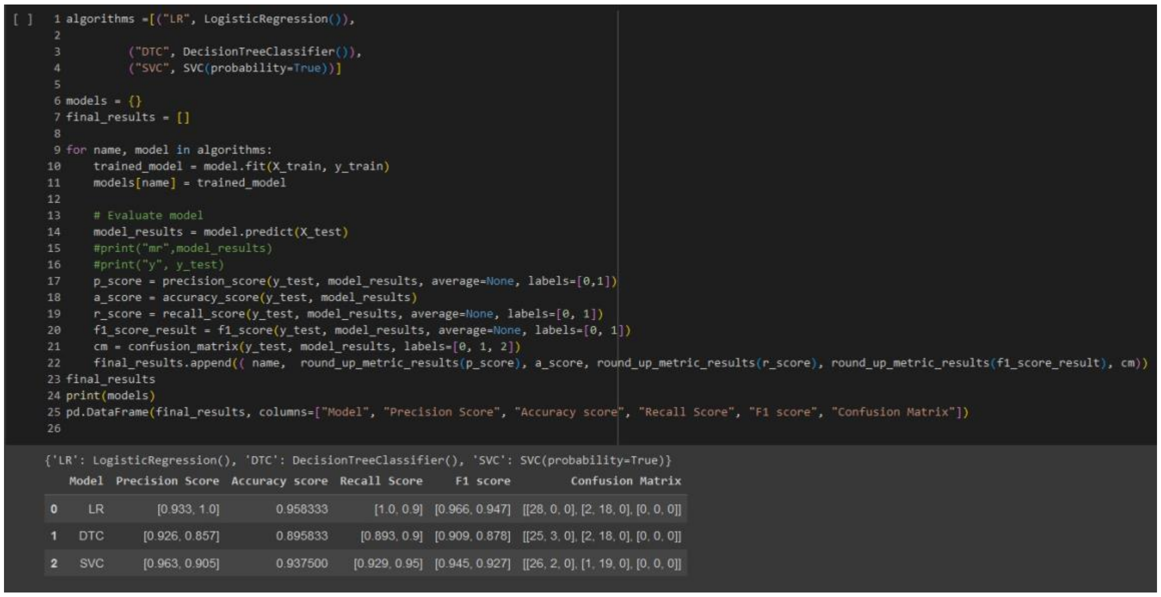
```
# Extract features and class
df = describe_dataset_points("./train.csv")
X = df.drop("label", axis=1)
#print("x:", X)
y = df["label"].astype("int")
#print("y:", y)
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
y_test.head(3)
```

```
Duplicate Rows : 119
291    0
59     0
6      0
Name: label, dtype: int64
```

This function takes a file path as input and returns a DataFrame containing the descriptive statistics of the dataset points.

- The train_test_split() function from the scikit-learn library is called to split the data into training and testing sets. The feature matrix X and the label vector y are split into X_train, X_test, y_train, and y_test with a test size of 0.2 and a random state of 1234.

- Finally, the head() method is called on y_test to display the first three rows of the test labels.



```python
algorithms =[("LR", LogisticRegression()),

        ("DTC", DecisionTreeClassifier()),
        ("SVC", SVC(probability=True))]

models = {}
final_results = []

for name, model in algorithms:
    trained_model = model.fit(X_train, y_train)
    models[name] = trained_model

    # Evaluate model
    model_results = model.predict(X_test)
    #print("mr",model_results)
    #print("y", y_test)
    p_score = precision_score(y_test, model_results, average=None, labels=[0,1])
    a_score = accuracy_score(y_test, model_results)
    r_score = recall_score(y_test, model_results, average=None, labels=[0, 1])
    f1_score_result = f1_score(y_test, model_results, average=None, labels=[0, 1])
    cm = confusion_matrix(y_test, model_results, labels=[0, 1, 2])
    final_results.append(( name,  round_up_metric_results(p_score), a_score, round_up_metric_results(r_score), round_up_metric_results(f1_score_result), cm))
final_results
print(models)
pd.DataFrame(final_results, columns=["Model", "Precision Score", "Accuracy score", "Recall Score", "F1 score", "Confusion Matrix"])

```

```
{'LR': LogisticRegression(), 'DTC': DecisionTreeClassifier(), 'SVC': SVC(probability=True)}
```

	Model	Precision Score	Accuracy score	Recall Score	F1 score	Confusion Matrix
0	LR	[0.933, 1.0]	0.958333	[1.0, 0.9]	[0.966, 0.947]	[[28, 0, 0], [2, 18, 0], [0, 0, 0]]
1	DTC	[0.926, 0.857]	0.895833	[0.893, 0.9]	[0.909, 0.878]	[[25, 3, 0], [2, 18, 0], [0, 0, 0]]
2	SVC	[0.963, 0.905]	0.937500	[0.929, 0.95]	[0.945, 0.927]	[[26, 2, 0], [1, 19, 0], [0, 0, 0]]

In this step, each algorithm is trained on the provided training data (X_train and y_train). The trained models are stored in the models dictionary, with the algorithm names as keys and the trained models as values.

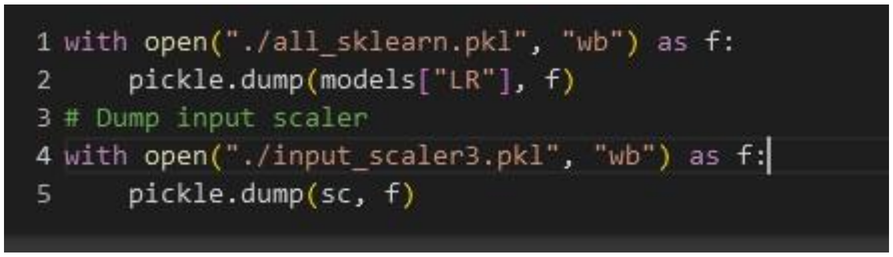
Next, the trained models are used to make predictions on the test data (X_test). Various evaluation metrics are computed using scikit-learn functions:

- Precision Score (p_score): Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The precision_score function is used to calculate this metric for each class (0 and 1) separately, and the results are stored.
- Accuracy Score (a_score): Accuracy is the ratio of correctly predicted observations to the total number of observations. The

accuracy_score function is used to calculate this metric, and the result is stored.

- Recall Score (r_score): Recall is the ratio of correctly predicted positive observations to the total actual positive observations. The recall_score function is used to calculate this metric for each class (0 and 1) separately, and the results are stored.
- F1 Score (f1_score_result): F1 score is the harmonic mean of precision and recall. The f1_score function is used to calculate this metric for each class (0 and 1) separately, and the results are stored.
- Confusion Matrix (cm): The confusion matrix is a table that shows the counts of true positive, true negative, false positive, and false negative predictions. The confusion_matrix function is used to calculate this matrix, and the result is stored.

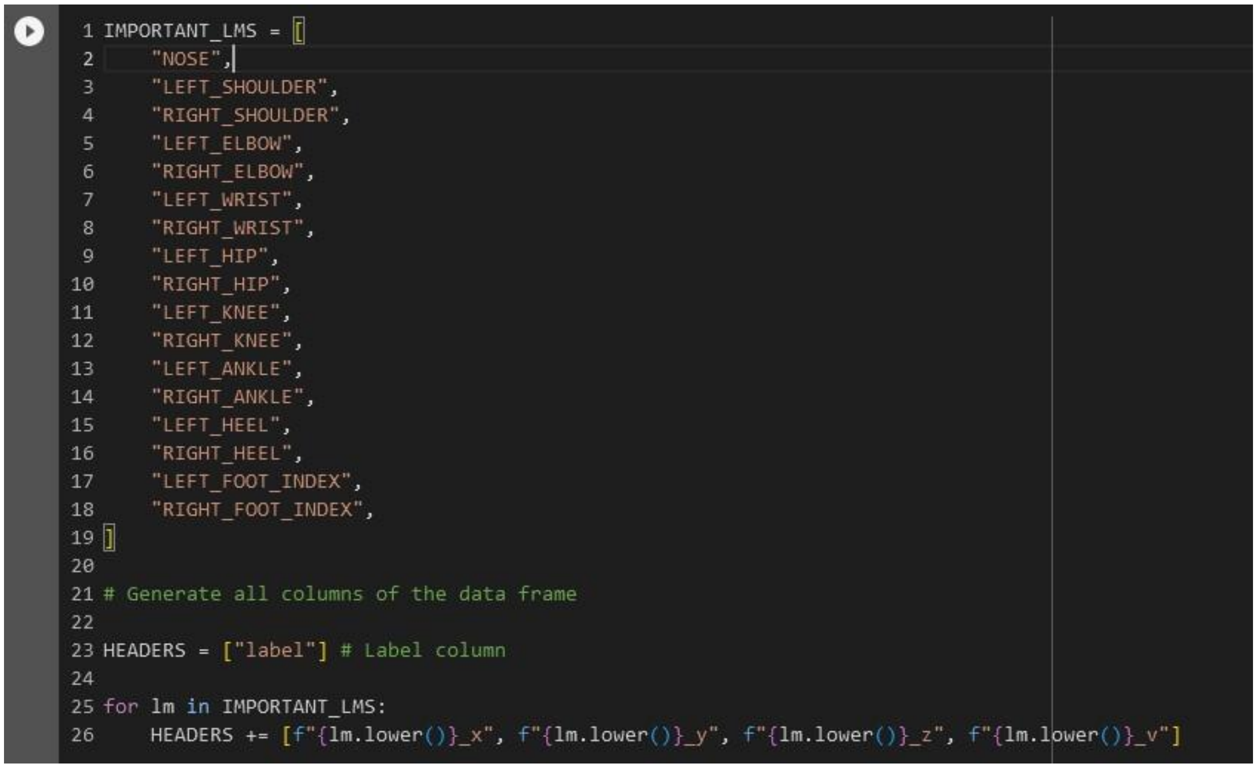
The computed evaluation metrics, along with the confusion matrix, are appended to the final_results list as a tuple for each algorithm.



```
with open("./all_sklearn.pkl", "wb") as f:
    pickle.dump(models["LR"], f)
# Dump input scaler
with open("./input_scaler3.pkl", "wb") as f:
    pickle.dump(sc, f)
```

Using pickle, train file is Dumped for our Live_Pose_Test__final.ipynb to use.

- Testing



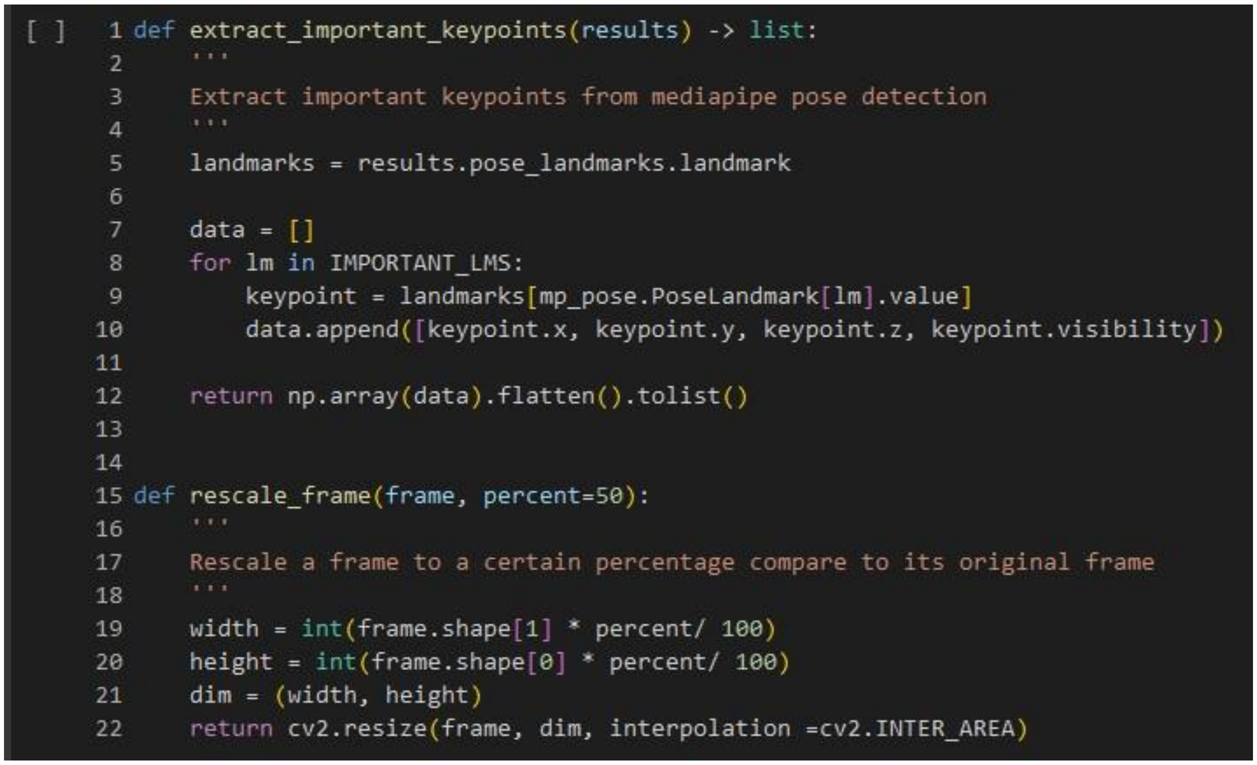
```python
IMPORTANT_LMS = [
    "NOSE",
    "LEFT_SHOULDER",
    "RIGHT_SHOULDER",
    "LEFT_ELBOW",
    "RIGHT_ELBOW",
    "LEFT_WRIST",
    "RIGHT_WRIST",
    "LEFT_HIP",
    "RIGHT_HIP",
    "LEFT_KNEE",
    "RIGHT_KNEE",
    "LEFT_ANKLE",
    "RIGHT_ANKLE",
    "LEFT_HEEL",
    "RIGHT_HEEL",
    "LEFT_FOOT_INDEX",
    "RIGHT_FOOT_INDEX",
]

# Generate all columns of the data frame

HEADERS = ["label"] # Label column

for lm in IMPORTANT_LMS:
    HEADERS += [f"{lm.lower()}_x", f"{lm.lower()}_y", f"{lm.lower()}_z", f"{lm.lower()}_v"]
```

A list of important landmarks is defined. These landmarks correspond to specific body parts detected by Mediapipe, such as nose, shoulders, elbows, wrists, etc. These landmarks will be used to extract keypoints for classification.



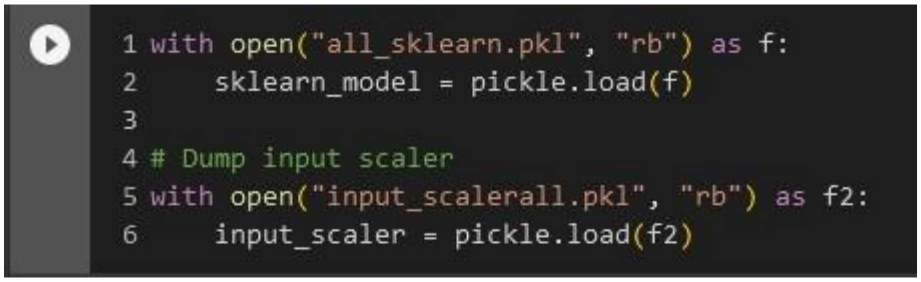
```python
def extract_important_keypoints(results) -> list:
    '''
    Extract important keypoints from mediapipe pose detection
    '''
    landmarks = results.pose_landmarks.landmark

    data = []
    for lm in IMPORTANT_LMS:
        keypoint = landmarks[mp_pose.PoseLandmark[lm].value]
        data.append([keypoint.x, keypoint.y, keypoint.z, keypoint.visibility])

    return np.array(data).flatten().tolist()


def rescale_frame(frame, percent=50):
    '''
    Rescale a frame to a certain percentage compare to its original frame
    '''
    width = int(frame.shape[1] * percent/ 100)
    height = int(frame.shape[0] * percent/ 100)
    dim = (width, height)
    return cv2.resize(frame, dim, interpolation =cv2.INTER_AREA)
```

The extract_important_keypoints(results) function takes the results of pose detection and extracts the important keypoints for classification. It loops through the list of important landmarks and retrieves the x, y, z coordinates, and visibility of each landmark. The keypoints are returned as a flattened list.



```python
with open("all_sklearn.pkl", "rb") as f:
    sklearn_model = pickle.load(f)

# Dump input scaler
with open("input_scalerall.pkl", "rb") as f2:
    input_scaler = pickle.load(f2)
```

We load trained model using pickle

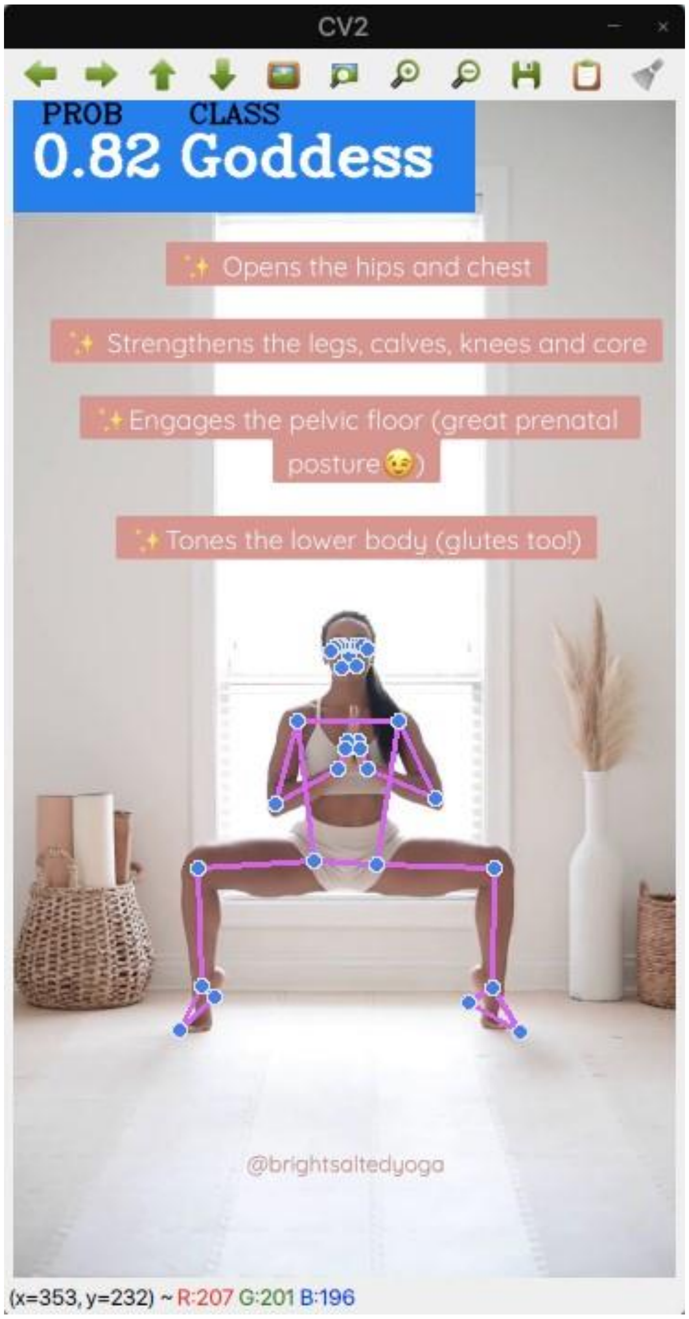
Classifying Poses

The code initializes the video capture object cap with the path to the video file. It then enters a loop to process each frame of the video.

Inside the loop, it reads the next frame from the video and performs the following operations:

- Rescales the frame to a smaller size.
- Converts the color space of the frame from BGR to RGB for Mediapipe.
- Processes the frame with Mediapipe pose detection.
- Checks if any pose landmarks are detected in the frame. If not, it continues to the next frame.
- Converts the color space of the frame back to BGR for display purposes.
- Draws the landmarks and connections on the frame using Mediapipe.
- Extracts the important keypoints from the pose detection results.
- Creates a pandas DataFrame from the keypoints.
- Makes a prediction using the pre-trained machine learning model.
- Evaluates the prediction and sets the current pose label based on the prediction probability and threshold.
- Displays the current pose label and prediction probability on the frame.
- Shows the frame in a window.
- Waits for the 'q' key to be pressed to exit the loop and close the window.
- After processing all frames, the video capture object is released, and all windows are closed.

Conclusion:


CV2
PROB CLASS
0.82 Goddess
✨ Opens the hips and chest
✨ Strengthens the legs, calves, knees and core
✨ Engages the pelvic floor (great prenatal posture 😉)
✨ Tones the lower body (glutes too!)
@brightsaltedyoga
(x=353, y=232) ~ R:207 G:201 B:196

This code demonstrates how to perform real-time yoga pose classification using pose detection and machine learning models. By extracting important Landmarks from the detected pose, it uses a pre-trained model to predict and classify the pose into specific categories. You can customize the code to use your own video files and adapt it to different pose classification tasks.

The output above is showing how our model is detecting the appropriate pose and the corresponding probability.

Our Github repository : https://github.com/riaprasad/Personal-Yoga-Trainer/

Credits:

1. kumar.1985864@studenti.uniroma1.it
2. prasad.1968913@studenti.uniroma1.it
3. khabbazian.1981092@studentiroma1.it
4. manku.1938352@studenti.uniroma1.it