# Yoga Pose detection and correction

## Approach

The core objective of this project is to build a system that recognizes and corrects yoga poses using keypoint extraction and classification techniques. Here's a breakdown of the approach:

### 1) Keypoint Extraction (Using MediaPipe):

- **MediaPipe** is a powerful framework that allows for efficient and accurate pose detection. It uses a pre-trained deep learning model to detect and extract key points from input images, focusing on key joints such as elbows, knees, shoulders, wrists, and ankles. These keypoints represent the critical landmarks needed to analyze the user's posture.
- **Pose Representation**: These extracted keypoints, typically in the form of 2D coordinates (x, y), define the body structure of the person performing the yoga pose. MediaPipe provides 33 keypoints per image, each corresponding to specific body parts, making it ideal for capturing detailed pose information.

### 2) Keypoint Normalization:

- **Normalization** is a crucial step to ensure consistency across different input images. Since keypoint positions are relative to the image dimensions, normalization removes the effect of variations in image size or scale.
- The process involves calculating the minimum and maximum x and y coordinates from the extracted keypoints and then transforming the keypoints so that they fit within a normalized range, typically between 0 and 1. This is achieved using the following formula:

$$\text{Normalized Keypoint} = \frac{\text{Keypoint} - [\min_x, \min_y]}{[\max_x - \min_x, \max_y - \min_y]}$$

  The z-coordinate (if present) is typically normalized independently. This ensures that the input data is consistent and scale-invariant, helping the model generalize better across different body sizes and angles.

### 3) Deep learning Model for Pose Classification (MLP):

- The **normalized keypoints** are then flattened into a one-dimensional array and fed into a **Multilayer Perceptron (MLP)** for classification. The MLP is a fully connected neural network consisting of multiple dense layers. It learns to map the input keypoints to predefined pose categories (such as Tree pose, Downward Dog, etc.).
- The architecture typically includes several hidden layers with ReLU activations, ensuring that the model can capture complex patterns in the data. Dropout layers are used to prevent overfitting, especially when the model is exposed to a limited amount of training data.

**4) Pose Feedback Generation:**

- Once the model classifies the pose, feedback is generated by comparing the current pose's keypoints to an ideal set of keypoints for that specific yoga pose.
- This ideal set is based on reference data of a correctly performed yoga pose, and it serves as a standard for comparison. The difference between the predicted pose and the ideal pose is measured to identify areas that need correction.
- The feedback mechanism could be designed to indicate whether the user's pose is correct or if adjustments are needed, providing guidance on how to align better with the ideal posture. This can be done through visual feedback or corrective suggestions.

## Data Preprocessing

1) Keypoint Extraction:
    - MediaPipe Pose module is used to extract 33 keypoints (x, y, and visibility) from yoga pose images.

2) Normalisation
    - Extracted keypoints are normalized to account for variations in scale and position.
    - The x and y coordinates of the keypoints are extracted separately
    - Then we normalise using the formula

$$x' = \frac{x - x_{\text{min}}}{\text{width}} \quad \text{and} \quad y' = \frac{y - y_{\text{min}}}{\text{height}}$$

Where:
x_min and y_min are the minimum x and y values of the keypoints, respectively and width and height are the dimensions of the bounding box calculated from the min and max coordinates.

3) Flattening:
    - Finally, after the keypoints are normalized, they are flattened into a 1D array

## Model Architecture:

The model is a Sequential Neural Network designed for pose classification:

- **Input Layer**: Accepts a 99-dimensional flattened keypoint array. This structure ensures that the model can process a significant number of keypoints efficiently for pose classification.
- **Hidden Layers**:
    - **Dense layer with 128 units and ReLU activation**. The large number of neurons helps the model capture complex relationships between the input keypoints. ReLU activation enables non-linear transformations, essential for accurate pose recognition.
    - **Dropout layer (20%)**Dropout prevents overfitting by randomly disabling 20% of the neurons during training. This regularization technique ensures better generalization to unseen data.
    - **Dense layer with 64 units and ReLU activation.** This smaller layer forces the model to learn more compact, relevant representations of the input data. ReLU ensures that the model can still capture non-linear features.
    - **Dropout layer (20%).** A second dropout layer further prevents overfitting by reducing model dependency on specific neurons. This improves the robustness and generalization of the model.
    - **Dense layer with 32 units and ReLU activation.**The final dense layer with 32 units abstracts the learned features even further, refining the representation for classification. ReLU continues to handle non-linearities effectively.

- **Output Layer**: Dense layer with softmax activation, outputting probabilities for each pose class. This ensures that the final predictions are in the form of a probability distribution over all possible poses.

**Compilation**:

- Optimizer: Adam. It adjusts the learning rate dynamically, leading to faster convergence. It combines the advantages of both Adagrad and RMSProp, improving efficiency in training.
- Loss Function: Sparse Categorical Crossentropy. This loss function is ideal for multi-class classification tasks with integer labels. It computes the difference between predicted probabilities and the actual class labels efficiently.
- Metric: Accuracy. It measures the proportion of correct predictions, providing a clear indication of the model's performance. It is straightforward and effective for classification tasks.

**Training**:

- Data: Split into training and validation sets. By splitting the data into training and validation sets, the model is evaluated on unseen data during training. This helps prevent overfitting and ensures that the model generalizes well.
- Epochs: 30
- Batch Size: 32
- Training for 30 epochs allows the model to learn effectively while avoiding overfitting. A batch size of 32 strikes a balance between memory efficiency and model convergence speed.

## Result:

The results of the model, with a test loss of **1.0274** and a test accuracy of **74.62%,** indicate that the model is performing reasonably well, but there is potential for improvement.

- The test loss represents how well the model's predictions match the actual labels. A loss of 1.0274 suggests that while the model is making correct predictions in some cases, there is still significant room for improvement. Lowering the loss further can help the model improve its accuracy by making its predictions closer to the true values.

- An accuracy of 74.62% means that approximately three-quarters of the predictions made by the model are correct. This is a solid starting point for a pose classification task, where the input data (keypoints) can be complex and varied.

## Next Steps:

1. **Real-Time Correction:** Implementing real-time feedback will allow users to receive immediate guidance as they perform yoga poses. Using video streams can help the model track and correct posture dynamically, offering timely suggestions during practice.

2. **Improved Feedback:** Add visual overlays to guide pose adjustments. Visual overlays, such as color-coded skeletons or arrows indicating joint movements, can provide clear, intuitive guidance for users. These overlays would visually represent the alignment and areas needing improvement.

3. **Dataset Expansion:** Include a broader range of yoga poses to enhance classification diversity.Expanding the dataset with more diverse poses from various yoga styles will help the model learn a broader spectrum of movements. This would enhance the model's ability to classify new, previously unseen poses accurately.

4. **Deployment:** Create a user-friendly interface for real-time yoga practice assistance. Developing a user-friendly interface, possibly as a mobile app or desktop application, will make the system more accessible to users. It should include an easy-to-navigate design for seamless interaction and feedback during practice.

## Conclusion:

The yoga pose recognition system demonstrates solid performance in classifying poses with a test accuracy of 74.62%. While the current model provides a good starting point, there are clear opportunities for improvement through data augmentation, architectural optimization, and real-time feedback integration. Expanding the dataset to include a broader range of poses, experimenting with advanced architectures like CNN-LSTMs, and developing a user-friendly deployment interface will further enhance the model's capabilities. These steps will not only improve classification accuracy but also create a more interactive and effective tool for users, offering valuable guidance during real-time yoga practice.