TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

DEEP LEARNING BASED AUTOMATED YOGA ASSISTANCE USING CNN AND LSTM

SUBMITTED BY:

AAGAB BHATTARAI (PUL077BCT001)

AAYUSHA ODARI (PUL077BCT006)

BIPUL NEUPANE (PUL077BCT018)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

April, 2025

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"Deep Learning based automated Yoga assistance using CNN and LSTM"** submitted by **Aagab Bhattarai**, **Aayusha Odari**, **Bipul Neupane** in partial fulfillment of the requirements for the Bachelor's degree in Electronics & Computer Engineering.

.....
Supervisor
Dr. Ganesh Gautam
Designation: Assistant Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

.....
External examiner
Bishnu Ram Neupane
CAO
Public Service Broadcasting Nepal

Date of approval:

Copyright

Acknowledgments

We extend our heartfelt gratitude to the **Department of Electronics and Computer Engineering** at Pulchowk Campus for providing us with the opportunity to undertake and execute this project. Our sincere appreciation goes to our project supervisor, **Dr. Ganesh Gautam**, for his unwavering support, expert guidance, and insightful feedback throughout the project. His mentorship has been invaluable in overcoming challenges and refining our approach.

We would also like to express our appreciation to our friends and seniors, for providing valuable insights and feedback for the project. We recognize and appreciate the collective efforts, insights, and contributions of everyone involved, directly or indirectly. Your support and collaboration have been pivotal. Furthermore, we would like to express our appreciation for the work previously done in this field and the resources we reference.

Sincerely,

Aagab Bhattarai (077BCT001)

Aayusha Odari (077BCT006)

Bipul Neupane (077BCT018)

Abstract

Fitness issues are highly prevalent, with motivation for engaging in fitness-related activities being generally low. This project aims to develop a yoga-assistance system that assists users in performing yoga-pose accurately by recognizing yoga posture and providing appropriate corrective feedback to improve the yoga posture. The system processes video frames, extracts key joint positions, and classifies yoga actions based on the action classification models trained on 3DYoga90 dataset. Three architectural variants were tested for yoga action classification: 1D CNN LSTM, 2D CNN LSTM, and Adaptive ST-GCN. Among them, the AGCN-STC-MTCN model demonstrated the best performance as per the test metrics and was selected for deployment. The system also integrates pose phase classification to distinguish between different phases of a yoga pose. Based on the classified action and extracted biomechanical features, it generates feedback for posture correction. Additionally, a web-based application was developed, incorporating a pipeline that simultaneously generates corrective feedback and visualizes a 3D human model for self-analysis. This feature allows users to evaluate their poses independently to a certain degree. This report provides a comparative analysis of the models for yoga pose recognition and details the system implementation for the yoga-assistance system. While the system shows effectiveness in providing yoga posture correction feedback, challenges such as dataset constraints and real-time processing persist. Future improvements include training the model on a more diverse yoga pose dataset, real-time optimization, and refining corrective feedback mechanisms.

Keywords: *Yoga Posture Recognition, Corrective Feedback Generation, Yoga Pose Phase Classification*

Contents

List of Figures

List of Tables

List of Abbreviations

2D	2 Dimension
3D	3 Dimension
HAR	Human Activity Recognition
YPR	Yoga Posture Recognition
CNN	Convolutional neural network
LSTM	Long Short-Term Memory
AGCN	Adaptive Graph Convolutional Network
MTCN	Multi-branch Temporal Convolutional Network
STSAM	Spatial–Temporal Self-Attention Module
STC	Spatial-Temporal-Channel Attention Module
ST-GCN	Spatial-Temporal Graph Convolutional Network
LSTM	Long Short-Term Memory
CNN	Convolution Neural Network
RTT	Round trip time
LLM	Large Language Model
API	Application Programming Interface

1. Introduction

1.1 Background

Yoga is an ancient practice that promotes the physical and mental well-being of individuals, promoting a harmonious balance between the body, mind, and soul. Yoga's ability to alleviate stress, enhance flexibility, and build muscle strength are just a few benefits that have contributed to its widespread adoption. However, incorrect postures can lead to inefficacy and even injury, making proper guidance essential for practitioners. While expert guidance from professional yoga instructors is ideal, access to such resources is often limited due to cost, availability, or location constraints.

Our proposed project aims to bridge this gap by utilizing the power of artificial intelligence to develop a yoga assistance system. This system utilizes deep learning techniques to analyze the user's position, assess their posture, and provide feedback, ensuring that practitioners perform yoga poses correctly and effectively. By offering guidance, our AI yoga assistance system aims to optimize yoga practice, helping users achieve better alignment, prevent injuries, and experience the full range of benefits that yoga offers.

1.2 Problem statements

Yoga offers numerous physical and mental health benefits, but achieving these benefits depends on performing asanas with proper technique. Incorrect posture and alignment can not only diminish the effectiveness of the practice but can also result in injury. Thus, recognizing and correcting yoga poses is a crucial aspect of the practice. However, access to qualified yoga trainers is often limited, and it is challenging for many individuals to receive the necessary guidance. Also, many practitioners rely on online videos which lack feedback and may not address the mistakes.

1.3 Objectives

The main objectives of the project are:

- To develop a system that can capture video of the user performing yoga poses, and by extracting the key points, track the user's body movements.
- To train deep learning models to classify various yoga poses and detect errors in the user's performance.

- To create a user-friendly interface that makes the application accessible.
- To perform a comparative study between different trained models.

1.4 Scope

The project involves developing a system that captures user videos during yoga practice and applies pose estimation techniques to extract keypoints for tracking body movements. It includes training a deep learning model to classify the user's yoga pose. The project also involves developing a component that detects errors in the user's posture by computing the deviation from an ideal pose, ultimately providing real-time feedback to help users correct their posture and improve their technique. Additionally, the project aims to create a user-friendly interface to ensure the application is accessible to a wide range of users and delivers a seamless, intuitive user experience.

2. Literature Review

2.1 Related work

Yoga Posture Recognition (YPR) is a specialized subset of the broader Human Acitivity Recognition (HAR) task, where the objective is to classify distinct yoga posture based on human acitivity data. HAR serves as a foundational component in the downstream task of fitness tracking and exercise monitoring. Within this context, YPR focuses on identifying fine-grained postural variations to support applications such as form correction and feedback generation.

For the task of YPR, [1] focused on developing an ensemble of deep learning models to accurately classify and estimate yoga poses. The study utilized the "Yoga Pose Classification" dataset from Kaggle [7], which includes 839 images categorized into five specific yoga poses. The author combined a novel residual convolutional neural network with three deep learning approaches: Xception, VGGNet, and SqueezeNet and created ensemble classifier to obtain 100% accuracy in the tested dataset. The author trained a custom residual CNN as the core model on the output features extracted from ImageNet-pretrained models: SqueezeNet, Xception, and VGGNet, each processed separately. Majority voting was used to obtain the final classification for the yoga pose.

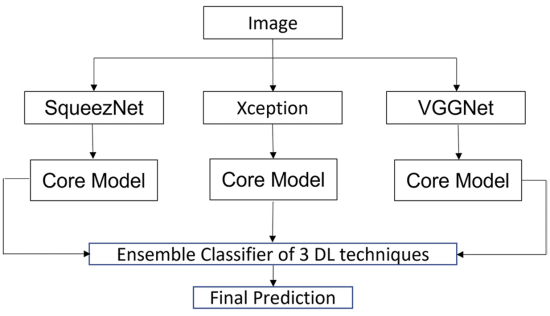


Figure 2.1: Yoga pose estimator algorithm as proposed by [1]

Wei et al. [8] developed a spatial–temporal self-attention enhanced graph convolutional network (STSAE-GCN) for yoga action recognition in video data. They created a custom Yoga10 dataset comprising 10 yoga poses, with a total of 960 videos. Pose estimation was performed on raw RGB video frames using HRNet [9], which, after human detection, predicts 2D keypoint coordinates to generate skeleton representations of yoga actions. By

incorporating a spatial-temporal self-attention module into ST-GCN++ [10], they proposed the STSAE-GCN model, which achieved a recognition accuracy of 93.83% on the Yoga10 dataset, outperforming the ST-GCN++ model.

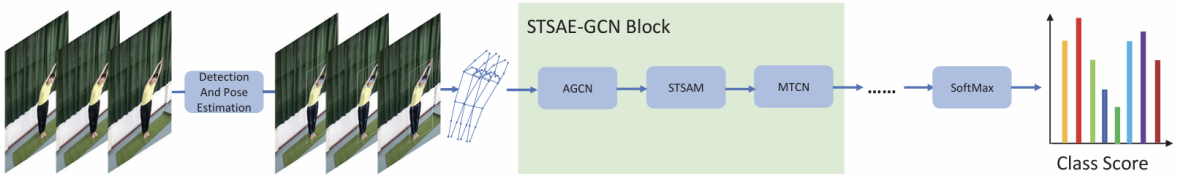


Figure 2.2: Framework of STSAE-GCN

Verma et al. [11] introduced Yoga-82, a dataset for fine-grained classification of human poses, featuring a three-level hierarchical label structure to overcome limitations of existing yoga pose datasets. The hierarchy categorizes poses by body positions, variations within those positions, and specific pose names, with data collected from the web using the Bing search engine. This organization allowed for the categorical error to be restricted to the specific subcategories and is more informative than the traditional flat N-way classification. The authors proposed a modified DenseNet architecture to utilize the hierarchical labels, enabling the network to learn features related to the hierarchical structure.

The 3DYoga90 dataset [12], a hierarchical video dataset for yoga pose understanding, comprises RGB videos and 3D skeleton sequences extracted from YouTube using BlazePose [13]. Unlike the Yoga-82 dataset [11], which is a 2D image dataset, 3DYoga90 provides video data with 3D skeleton information. This dataset covers 90 unique poses.

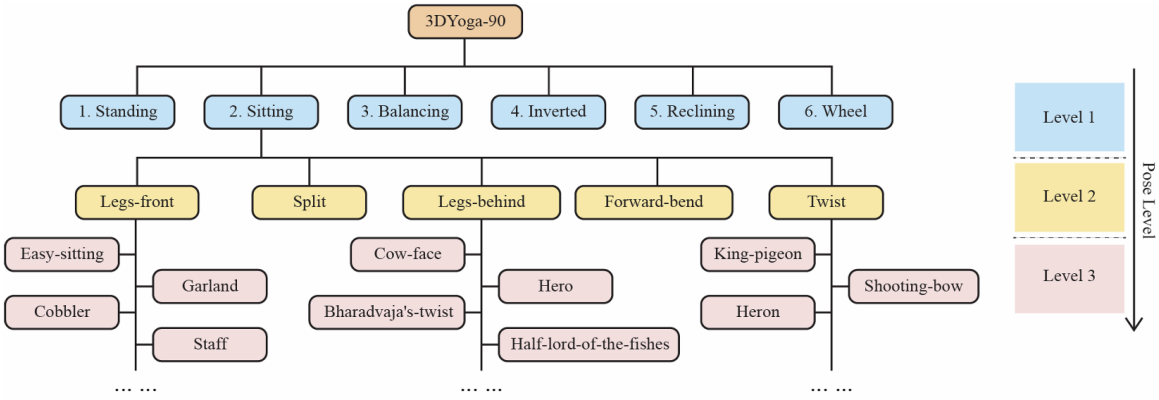


Figure 2.3: Three-level Hierarchical Structure of 3DYoga90 Dataset

To assess the utility of the dataset the author used the skeleton dataset and applied DNN model to predict the three levels of the proposed dataset. The author used three distinct DNN model variants which consistently demonstrated strong performance while drop in

performance metrics at level 3 was seen, due to higher number of classes(90) and subtle pose differences that DNN couldn't distinguish from the extracted feature vector.

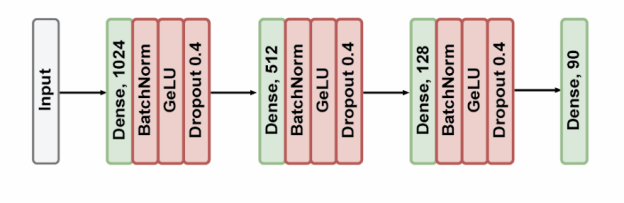


Figure 2.4: DNN architecture for Pose Classification with skeleton data

YogNet [14], made use of two-stream network, i.e. pose-based CNN-LSTM model alongside 3D convolutional networks (C3D). The result of two streams were fused to achieve higher accuracy for yoga posture classification. For posture grading, the authors utilized domain knowledge to establish thresholds for various postures, which allowed the generation of defined recommendations for posture corrections.

Jing Chen et al. [15] developed a CNN-LSTM model for action recognition in the context of the traditional Chinese exercise Baduanjin. By utilizing VGG16 to extract feature maps, they converted these into a sequence of feature vectors and fed into the LSTM layer, achieving 96.43% classification accuracy on their testing dataset for Baduanjin. In contrast, when they manually extracted angular features from skeleton data and inputted these directly into the LSTM layer, the model only reached a classification accuracy of 66.07%. This work highlighted effectiveness of end-to-end approach over feature engineering approach for improving classification accuracy in action recognition tasks.

In [16], the authors developed a new approach for yoga pose grading, where they trained a pose feature encoder using contrastive triplet examples. This method extracts human body skeleton keypoints from input images and uses these coordinates for grading poses. Notably, the proposed approach performs grading for a single image frame, which means they don't incorporate temporal information from video inputs.

A comprehensive review [2] detailed various approaches that have been proposed for yoga pose estimation and yoga pose grading for papers earlier than May 2021. The paper presented a brief look at various approaches tried in yoga posture recognition while also presenting the significance of each approach while highlighting key points such as the dataset that was used and methodologies that were tried out. The paper presented direct comparison in terms of accuracy that was achieved using various deep learning models for YPR; though across the papers different dataset was used with different number of yoga poses which were to be classified, approaches for YPR using hybrid CNN-LSTM model showed higher benchmark scores.

In [17], trained 3D human pose estimator on Human3.6M dataset and then fine-tuned on exercise-specific synthetic dataset which reduced the error in joint position estimation related to exercise-specific pose, then used joint angles to provide feedback to the user regarding their exercise pose.

For the task of HAR, ImageNet pretrained CNN is usually taken as feature extractor in video sequences and the extracted feature is fed into RNN for modeling sequential nature of the action [15, 18, 19]. The study in [20] tested alternative approach using Vision transformers(ViT) for spatial feature extraction and recurrent transformer(ReT) for sequence modeling, they tried out few different variation with ResNet50 or ViT as spatial feature extractor and ReT or LSTM for sequential modeling of the input video frames. The results indicated that ViT-ReT based models gave faster inference while being memory efficient than the CNN-LSTM based approach but offered lower accuracy and higher loss due to lower performance of ViT compared to ResNet50.

2.2 Related theory

2.2.1 Human Pose Estimation

Human pose estimation (HPE) involves estimating the configuration of human body parts from input data captured by sensors, such as images and videos. HPE provides geometric and motion information about the human body, which is utilized in various applications including human-computer interaction, motion analysis, augmented reality (AR), virtual reality (VR), and healthcare [21]. However, occlusion, lighting variability, etc, still pose major challenges in the accurate assessment of human pose when the input is just an image from a monocular camera with no depth information. Various pose estimation methods exist, like OpenPose, BlazePose, and High-Resolution Net (HRNet).

2.2.2 Human Body Modeling

From the estimated location of human body parts, a human body representation is constructed. This representation provides a holistic view by organizing the identified key points extracted from the input image and helps to interpret and analyze human posture and movement. Human body models are separated into the following categories:

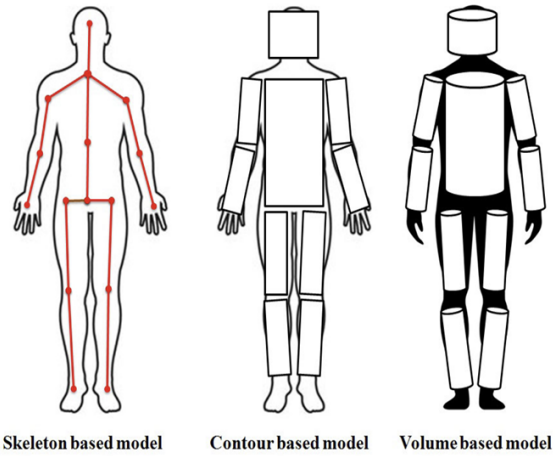


Figure 2.5: Types of models in human body modeling [2]

1. **Skeleton-Based Model** Also known as the kinematic model, this approach represents the human body as a skeletal structure composed of key landmarks through human pose estimation methods. The skeleton-based human action recognition needs less computation compared with methods relying on other modalities [8]. The kinematic model is used for both 2D and 3D pose estimation.

2. **Contour-Based Model** Also referred to as a planar model, the model depicts the human body shape by encircling it with a contour. Multiple rectangles approximate the person's body contours in this model.

3. **Volume-Based Model** Volumetric models focus on reconstructing a high-quality human mesh and provide extra shape information of the human body.[21].

2.2.3 Human Action Recognition

Human Action Recognition (HAR) is a branch of computer vision that deals with identifying and interpreting human actions at various levels. These activities can be grouped into four categories, i.e., gestures, actions, interactions, and group activities. HAR systems use advanced algorithms and deep learning models to analyze video frames and detect these varied human activities. The key aspects of HAR generally includes pose estimation, feature extraction, and classification. It has many applications, including surveillance, human-computer interaction, video indexing, and activity monitoring. However, HAR systems still face several challenges, such as handling viewpoint and scale variations, dealing with occlusions and background clutter, and capturing complex temporal dependencies in human actions.

2.2.4 Yoga Posture Recognition

Yoga Posture Recognition focuses on the identification and classification of yoga asanas from either static images or continuous video frames. It is essentially a domain-specific extension of general human pose recognition, adapted to interpret the unique and diverse postures associated with yoga practice. Human pose images for the yoga activity consist of some of the most diverse poses that a human body can perform, and some of these poses that are performed are too complex to be captured from a single point of view. The task becomes more difficult with the changes in image resolution and occlusions [11].

2.2.5 Yoga Posture Grading

Yoga posture grading involves automatically assessing the quality of yoga poses performed by practitioners. This area remains relatively underexplored, as highlighted in the survey, [2]. The primary methodology involves comparing between yoga posture done by the expert and the user.

Usually, a reference posture is defined, representing the ideal yoga pose the user aims to perform. To evaluate accuracy, the predicted key-point for user posture is compared to the target posture by analyzing angles and joint positions. The degree of similarity between these poses provides an estimate of how well the pose has been executed. One approach to identifying deviations is to measure the angles between the user's joints and check whether they fall within an acceptable range, as defined by yoga expertise, for performing the pose correctly [2]. Also, yoga-pose grading introduced in [16] is limited when working with sequential image frames, and moreover, the dataset required for training such a model couldn't be found in the public domain.

2.2.6 3D pose estimation with MediaPipe:

MediaPipe Pose is a machine learning solution designed for high-fidelity body pose tracking, which can infer 33 3D landmarks from RGB video frames. It also generates a background segmentation mask covering the entire body, making it suitable for applications, including activity recognition, fitness tracking, and augmented reality. It is built on the BlazePose architecture, which is optimized for real-time performance and capable of detecting 33 key landmarks, including the face, hands, and feet. MediaPipe Pose uses a two-step detector-tracker approach for accurate 3D pose estimation. It first detects the body region and then predicts 33 keypoints in 3D space.

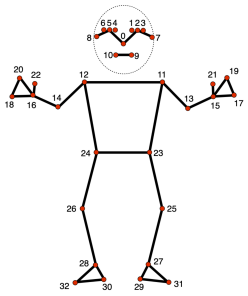


Figure 2.6: 33 pose landmarks

2.2.7 Transfer Learning

Transfer learning is a technique where a model pre-trained on a large dataset is reused or adapted to a different but related task. It's a way to reuse models and knowledge instead of starting from scratch. It helps to overcome some of the core challenges in machine learning, such as the need for large amounts of data, long training times, and high computational costs, which makes it an invaluable technique in modern AI research and applications. In transfer learning, the pre-trained model is often composed of two types of layers: frozen layers and trainable layers.

- Frozen Layers: These layers are kept unchanged during fine-tuning and retain general features learned from the original task. They extract universal patterns from input data, such as edges and textures.
- Trainable Layers: These layers are updated during fine-tuning to adapt to the specific characteristics of the new dataset. They help the model learn task-relevant patterns and features that are unique to the target application.

2.2.8 CNN

A Convolutional Neural Network (CNNs) is a class of deep learning models designed to process visual data, such as images and videos. It consists of multiple layers: convolutional, pooling, and fully connected layers, each playing a specific role in feature extraction and pattern recognition. The convolutional layers apply filters to extract essential features like edges, textures, and patterns. These filters move across the image and help the network to understand spatial hierarchies and local dependencies within the image. Next, pooling layers reduce the size of the feature maps created by the convolutional layers, which reduces the spatial dimensions of the feature maps. This step helps simplify the data, reduce computation, and make the model more resilient to small shifts or distortions in the input. Finally, fully connected layers use the extracted features to make predictions, such as classifying the input into predefined categories [22].

1D CNN

A 1D CNN is used for processing sequential data. It works by applying a convolutional filter that moves across the sequence, step by step. This filter consists of learnable weights that are updated during training. At each step, the filter multiplies its values with the corresponding input values in a sequence segment, summing the results to produce a single output value. This process is repeated along the entire sequence, generating a transformed output that captures important local patterns and dependencies.

2D CNN

A 2D CNN is designed to process data for image and video processing, where convolutional filters slide over both height and width dimensions. It captures spatial hierarchies and patterns, making it effective for tasks such as object detection, facial recognition, and medical image analysis. The combination of multiple convolutional layers, pooling, and activation functions enables deep feature extraction, allowing CNNs to learn complex representations of visual data.

2.2.9 LSTM

Long Short-Term Memory (LSTM) [23] networks are a specialized type of recurrent neural network (RNN) designed to capture patterns and dependencies in sequential data. Unlike traditional RNNs, they overcome the problem of long-term dependency by using a gating mechanism.

LSTMs maintain two separate types of memory: short-term memory, which handles recent inputs, and long-term memory, which preserves information over longer sequences. By selectively adding or removing data from their cells using input, forget, and output gates, LSTMs can learn and retain information over extended periods.

The mathematical operations of the input gate, forgetting gate, and output gate can be respectively written as follows:

$$i_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{2.1}$$

$$f_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{2.2}$$

$$o_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{2.3}$$

where $\mathbf{W}_i, \mathbf{W}_f$ and $\mathbf{W}_o$ are the weight matrices of input gate, forgetting gate and output gate respectively; $\mathbf{b}_i, \mathbf{b}_f$ and $\mathbf{b}_o$ are the offsets of input gate, forgetting gate and output gate respectively; σ represents the sigmoid function, whose output is between 0 and 1. The output of the LSTM cell is obtained by sequentially performing the calculations in the memory unit

and output gate as follows:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot \mathbf{h}_{t-1} + \mathbf{W}_c \cdot \mathbf{x}_t + \mathbf{b}_c) \tag{2.4}$$

$$\mathbf{c}_t = f_t * \mathbf{c}_{t-1} + i_t * \tilde{\mathbf{c}}_t \tag{2.5}$$

$$\mathbf{h}_t = o_t * \tanh(\mathbf{c}_t) \tag{2.6}$$

where $\tilde{\mathbf{c}}_t$ is the candidate state at time t; $\mathbf{W}_c$ is the weight matrix of the candidate states; $\mathbf{b}_c$ is the offset of the candidate states; $\mathbf{c}_t$ is the cell state of time t; and $\mathbf{h}_t$ is the final output at time t.

2.2.10 Cross Entropy Loss

Cross-entropy loss is a metric to measure the performance of a classification model. It compares the actual class with the predicted class probabilities. A lower cross-entropy loss indicates better performance.

For problems with multi-class classification, the cross-entropy loss for a single example is calculated as:

$$L = -\sum_{k=1}^{K} y_k \log(p_k) \tag{2.7}$$

where y_k is the true probability of class k, typically represented as 1 for the correct class and 0 for all other classes, and p_k is the predicted probability of class k.

This loss function penalizes incorrect predictions by assigning higher values when the predicted probability for the correct class is low, encouraging the model to improve its predictions during training.

2.2.11 SMPL-X

SMPL-X (Skinned Multi-Person Linear Model eXpressive)[3] is a unified 3D body model with shape parameters that represents realistic 3D human shapes, poses, and facial expressions. SMPL-X extends SMPL with fully articulated hands and an expressive face. The model is based on a statistical representation of the human body, constructed from a large dataset of 3D body scans. It uses linear blend skinning with learned corrective blend shapes. SMPL-X has 10,475 vertices and 54 joints, enabling detailed and accurate mesh representations. The model is defined by a function $M(\theta, \beta, \psi)$, where θ represents pose parameters, β represents shape parameters, and ψ represents facial expression parameters. Due to its detailed articulation, SMPL-X is widely used in motion capture, virtual avatars, and human-computer interaction applications.

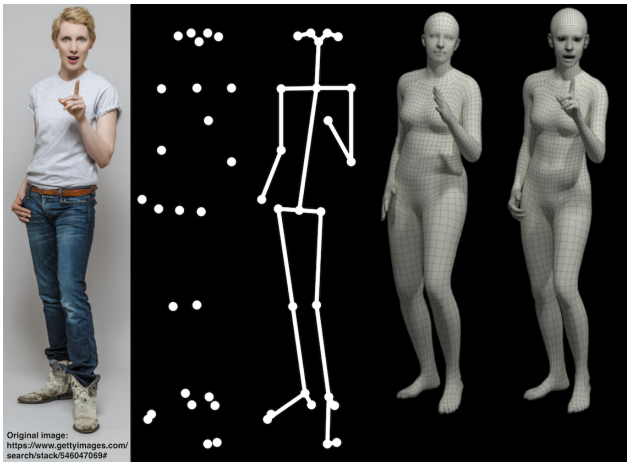


Figure 2.7: Pose Estimation and 3D Human Reconstruction. Image from [3]

The image demonstrates the stages of human pose estimation and reconstruction. From left to right, it shows the original RGB image, detected major joints, a skeletal representation, the SMPL model (female) for body shape and posture, and the SMPL-X model (female), which extends SMPL by incorporating facial expressions and hand articulation.

2.2.12 ResNet18

ResNet-18 is a lightweight deep learning model commonly used for tasks like image classification and feature extraction. It belongs to the ResNet (Residual Network) family [24], which utilizes skip connections (shortcuts) to enable smoother gradient flow, preventing vanishing gradients. ResNet-18 is built with 18 layers, making it an efficient and fast model suitable for real-time applications. With 18 layers, ResNet-18 is computationally less expensive than the deeper versions like ResNet-50 or ResNet-101, which makes it suitable for real-time applications, such as mobile devices or embedded systems.

The core idea behind ResNet-18 is the use of residual connections, which allow the model to learn deeper feature representations by bypassing some layers. Each residual unit includes two convolutional layers that focus on learning the difference (or residual) from the identity, rather than trying to learn the full mapping from scratch. This approach leads to faster convergence and better generalization.

ResNet Block

A residual block is the fundamental unit of ResNet, designed to solve the problem of vanishing gradients in deep networks. It includes a shortcut connection that skips one or more layers, allowing the network to learn identity mappings. This ensures better gradient flow and improves stability during training.

Each ResNet block consists of two or more convolutional layers, followed by batch normalization (BN) and ReLU activation. The key innovation is the addition of a shortcut connection that directly adds the input to the output of the convolutional layers, forming a residual mapping:

$$y = F(x) + x \tag{2.8}$$

where, x is the input to the block,
$F(x)$ represents the transformation applied by the convolutional layers,
The sum of $F(x)$ and x is passed through an activation function.

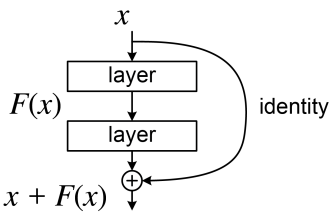


Figure 2.8: A residual block

2.2.13 VGG16

VGG16 is a deep convolutional neural network that consists of 16 weight layers, including 13 of which are convolutional layers and the remaining 3 are fully connected layers. It was developed by the Visual Geometry Group (VGG) at the University of Oxford. [4]

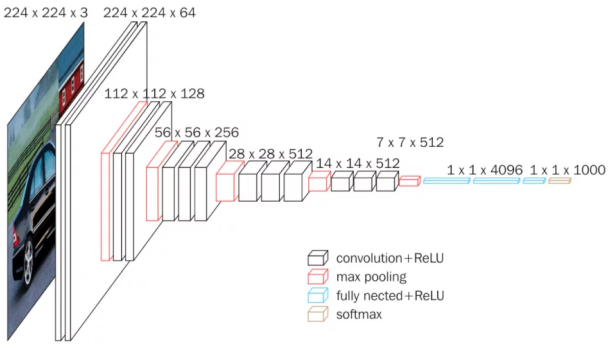


Figure 2.9: VGG Neural Network Architecture [4]

The architecture follows a stacked convolutional approach, using only 3×3 convolutional filters with ReLU activation. This uniform filter size enables the network to learn complex patterns while maintaining architectural simplicity. 2×2 max pooling layers are applied after certain convolutional blocks to reduce spatial dimensions and improve computational efficiency. After the final convolutional block, the output is flattened into a vector and passed through a series of fully connected layers. The final layer uses a softmax activation to classify images. VGG16 is widely used for transfer learning, object detection, and medical imaging.

2.2.14 DenseNet121

DenseNet-121 [5] introduces dense connectivity patterns to enhance feature propagation and improve training efficiency. Unlike traditional architectures, where each layer receives input only from the previous layer, DenseNet establishes direct connections between each layer and all subsequent layers. This allows feature reuse, reduces redundancy, and makes the network more efficient.

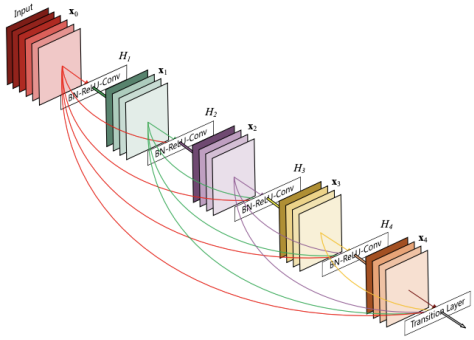


Figure 2.10: A 5-layer dense block with a growth rate of k = 4. [5]

DenseNet-121 consists of four dense blocks, each containing multiple convolutional layers. Each dense block is connected by transition layers that reduce the number of feature maps and the spatial dimensionality of the feature maps. Transition layers perform downsampling between these blocks using 1×1 convolutions followed by average pooling. The growth rate (k) determines how many new feature maps each layer contributes, promoting compact and efficient learning.

2.2.15 Adaptive Graph Convolutional Network

The Adaptive Graph Convolutional Network (AGCN) operates on a graph structure defined by a set of adjacency matrices that capture the connectivity between nodes based on physical or structural relationships [10, 25]. The adjacency matrices are initialized as per the following:

1. **Self-loop Matrix (A_1)**: This matrix represents connections of each node to itself, typically initialized as the identity matrix $A_1 = I$, where $I \in \mathbb{R}^{V \times V}$ and V is the number of nodes. It ensures that each node's own features are considered in the convolution process.

2. **Inward/Centripetal Matrix (A_2)**: This matrix encodes directed edges pointing inward, such as connections from peripheral nodes to a central node or sink, reflecting centripetal physical relationships. For an edge from node i to node j, $(A_2)_{ij} = 1$ if such a connection exists, and 0 otherwise.

3. **Outward/Centrifugal Matrix (A_3)**: This matrix captures directed edges pointing outward, such as connections from a central node to peripheral nodes, representing

centrifugal physical relationships. For an edge from node i to node j, $(A_3)_{ij} = 1$ if such a connection exists, and 0 otherwise.

These matrices, collectively forming a tensor $A \in \mathbb{R}^{K \times V \times V}$ with $K = 3$, are initialized based on the underlying physical topology and are made learnable to adapt during training.

The adjacency matrices are normalized so that nodes with higher number of neighbours don't dominate aggregated features. So, if $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix of a directed graph. Then, the in-degree matrix D is defined as:

$$D = \text{diag}(d_1, d_2, \ldots, d_N) \tag{2.9}$$

where the in-degree of node j is given by:

$$d_j = \sum_{i=1}^{N} A_{ij} \tag{2.10}$$

So, the inverse in-degree matrix D^{-1} is given as follows:

$$D^{-1} = \text{diag}(d_1^{-1}, d_2^{-1}, \ldots, d_N^{-1}) \tag{2.11}$$

where the individual components are given by,

$$d_i^{-1} = \begin{cases} \frac{1}{d_i}, & \text{if } d_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

Then, we normalize A to obtain the normalized adjacency matrix $\hat{A}$ as:

$$\hat{A} = AD^{-1} \tag{2.13}$$

Thus, $\hat{A}$ is a column-normalized adjacency matrix where each column sums to 1 if the original column sum was non-zero.

The AGCN processes input features $f_{\text{in}} \in \mathbb{R}^{C_{\text{in}} \times T \times V}$, where C_{in} is the number of input channels, T is the number of frames, and V is the number of nodes.

The output feature is defined as:

$$f_{\text{out}} = \delta\big(g(f_{\text{in}})\,\hat{A}\big) \tag{2.14}$$

where, $g(f_{\text{in}})$ represents a 1×1 2D convolution applied to f_{in} with $g(f_{\text{in}}) \in \mathbb{R}^{(C_{\text{out}} \cdot K) \times T \times V}$. Here, $\delta(z)$ represents non-linearity like ReLU activation function before which batch normalization is performed. At the end we get $f_{\text{out}} \in \mathbb{R}^{C_{\text{out}} \times T \times V}$; where C_{out} is the number of output channels.

2.2.16 Multi-Branch Temporal Convolution

Multi-Branch Temporal Convolutional Networks (MTCNs) extend the capabilities of traditional Temporal Convolutional Networks (TCNs) by incorporating multiple branches to capture multi-scale temporal dependencies in sequential data. This architectural approach enables the model to extract diverse temporal patterns from sequential data by processing it through different convolutional pathways simultaneously. Here, the variant shown in [26] is used instead of the varaint shown in [25].

The network takes in a sequential input, which is then processed through parallel convolutional pathways. Four pathway consists of a 1x1 convolution followed by a 3x1 convolution with different dilation rates (eg: 1, 2, 3, and 4), allowing the network to capture temporal dependencies at multiple scales. Additionally, there is a separate pathway that applies max pooling followed by a 1x1 convolution to extract prominent temporal context, while another 1x1 convolution branch helps retain important input channel information. The outputs from all these branches are concatenated, and a residual connection is added, that allows the input to be directly added to the concatenated output.

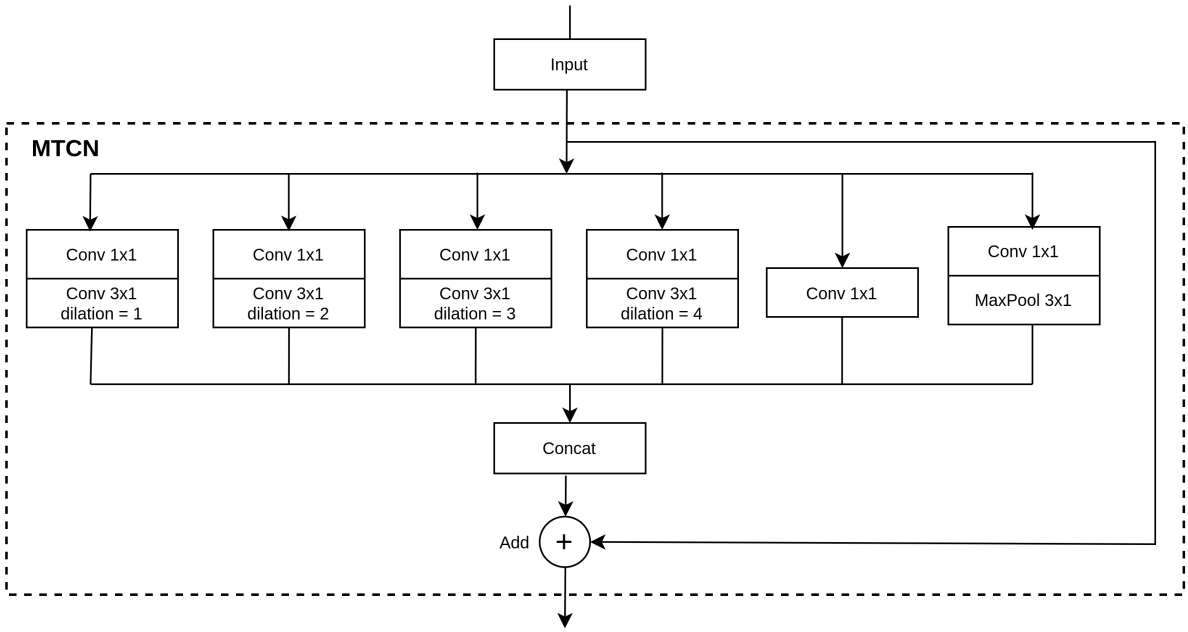


Figure 2.11: MTCN

2.2.17 Spatial Temporal Self Attention Module (STSAM)

The Spatio-Temporal Self-Attention Mechanism (STSAM) as introduced by [25] is used to enhance feature representation by modeling both spatial and temporal dependencies in se-

quential data. It processes input features through two attention branches: one that captures temporal dynamics and another that focuses on spatial relationships. Given an input feature map $F_{\text{in}} \in \mathbb{R}^{C \times T \times V}$, three 1×1 convolutions generate Query (Q), Key (K), and Value (V) feature maps. Temporal pooling extracts motion patterns, forming Q_t, K_t, V_t, while spatial pooling focuses on joint relationships, generating Q_s, K_s, V_s. The attention maps are computed using:

$$M_s = \text{softmax}\left(\frac{Q_s K_s^T}{\sqrt{d_k}}\right) V_s \tag{2.15}$$

$$M_t = \text{softmax}\left(\frac{Q_t K_t^T}{\sqrt{d_k}}\right) V_t \tag{2.16}$$

where $M_s \in \mathbb{R}^{C \times 1 \times V}$ and $M_t \in \mathbb{R}^{C \times T \times 1}$ represent the spatial and temporal attention maps, respectively. To ensure stability, the attention maps are refined using an additional 1×1 convolution and a sigmoid activation function:

$$M_{s_1} = \delta(W_s * M_s), \quad M_{t_1} = \delta(W_t * M_t) \tag{2.17}$$

where W_s and W_t are learnable weights. The final output is obtained by integrating both attention maps using residual connections:

$$F_{\text{out}} = (F_{\text{in}} + F_{\text{in}} \bigodot M_{s_1}) + (F_{\text{in}} + F_{\text{in}} \bigodot M_{t_1}) \tag{2.18}$$

where $\bigodot$ represents element-wise multiplication. This mechanism refines feature representations, ensuring effective learning of spatial and temporal dependencies in sequential data.

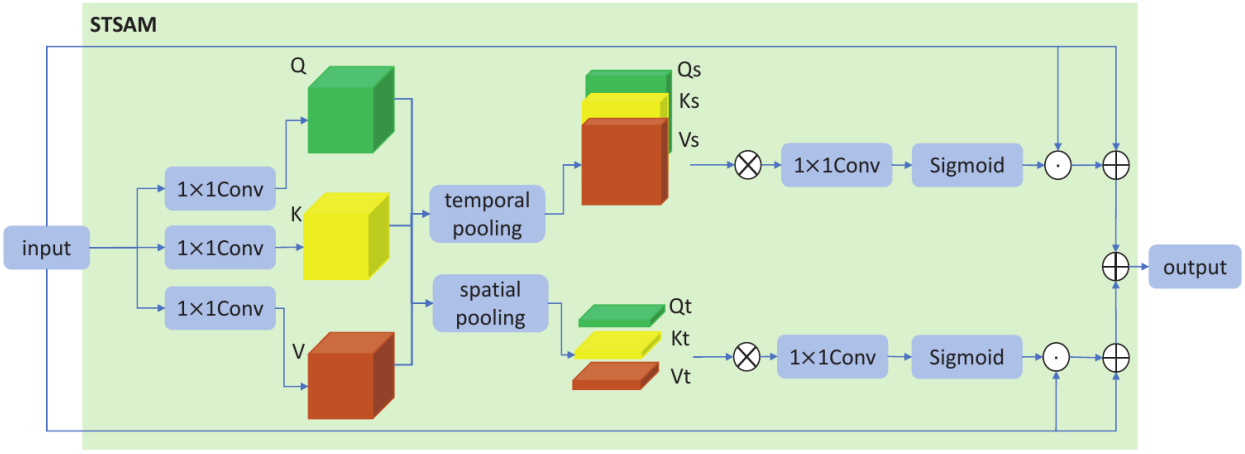


Figure 2.12: STSAM

2.2.18 STC-Attention Module Formulation

The Spatio-Temporal-Channel (STC) attention module, introduced in MS-AGCN [6], enhances the skeleton-based action recognition by adaptively capturing spatial, temporal, and channel-wise dependencies. The STC module has the following structure:

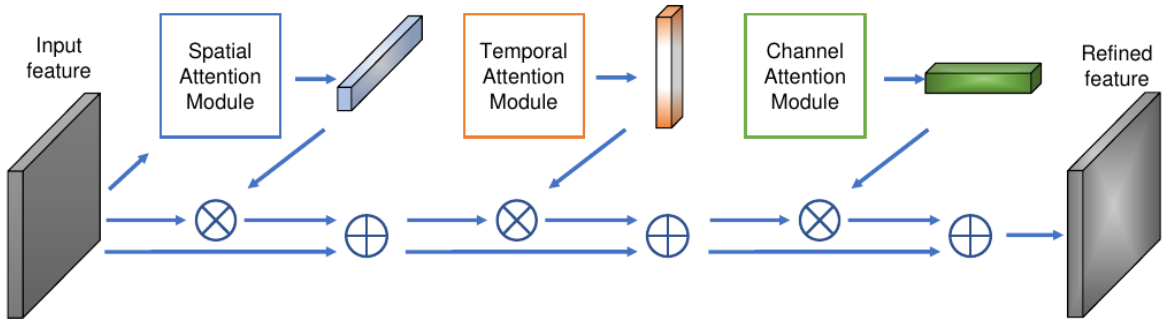


Figure 2.13: Illustration of the STC-attention module. Image from [6]

Here, three sub-modules are arranged sequentially in the order of SAM, TAM, and CAM. In the figure, $\otimes$ denotes the element-wise multiplication. $\oplus$ denotes the element-wise addition. It contains three sub-modules: spatial attention module, temporal attention module, and channel attention module.

SAM(Spatial Attention Module)

SAM focuses on important joints (nodes) by dynamically adjusting the importance of spatial connections. It helps capture key body parts involved in an action. It computes the following:

$$\mathbf{M_s} \in \mathbb{R}^{1\times1\times N}, \quad \mathbf{M_s} = \sigma\big(g_s(\text{AvgPool}(f_{in}))\big) \tag{2.19}$$

where g_s is a 1D convolution with kernel size set to number of joints and σ is the sigmoid function.

TAM(Temporal Attention Module)

TAM addresses the temporal dimension of the data. It identifies and prioritizes important time steps in the sequence, such as keyframes in a video or critical moments in a time series. It is computed as:

$$\mathbf{M_t} \in \mathbb{R}^{1\times T\times1}, \quad \mathbf{M_t} = \sigma\big(g_t(\text{AvgPool}(f_{in}))\big) \tag{2.20}$$

where g_t is a 1D convolution over temporal dimension and σ is the sigmoid function.

CAM(Channel Attention Module)

CAM operates on the feature channels of the data. It learns to highlight the most relevant features for the task by assigning adaptive weights to different channels. It generates the attention maps as follows:

$$\mathbf{M_c} \in \mathbb{R}^{C\times 1\times 1}, \quad \mathbf{M_c} = \sigma\big(\mathbf{W_2}(\delta(\mathbf{W_1}(\text{AvgPool}(f_{in}))))\big) \tag{2.21}$$

where δ is a ReLU activation function.

2.2.19 ReLU

ReLU (Rectified Linear Unit) is a commonly used activation function in deep neural networks that works by outputting the input directly if it is positive and outputting zero if the input is negative. It introduces non-linearity without adding much computational complexity and helps avoid issues like vanishing gradients, which improves the training of deep networks. Mathematically, it is defined as:

$$f(x) = \max(0, x) \tag{2.22}$$

and is shown in Figure 2.14.

2.2.20 Tanh

The Hyperbolic Tangent (Tanh) is a non-linear activation function in neural networks whose output values lie in the range from -1 to 1, making it zero-centered as shown in Figure 2.15. Tanh has a stronger gradient for inputs around zero, improving learning efficiency. However, for very large or very small inputs, tanh can suffer from the vanishing gradient problem. It is defined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.23}$$

2.2.21 Sigmoid Function

The Sigmoid activation function is defined as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.24}$$

and maps inputs to a range of 0 to 1. Its graph has a characteristic S-shaped curve or sigmoid curve as shown in Figure 2.16.

2.2.22 Evaluation Metrices

For the evaluation of our model, macro-averaged Precision, Recall, and F1-score were used to ensure that each class contributes equally to the final score, regardless of its frequency in the dataset.

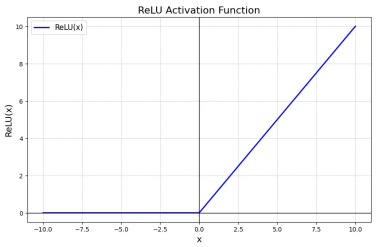


Figure 2.14: ReLU

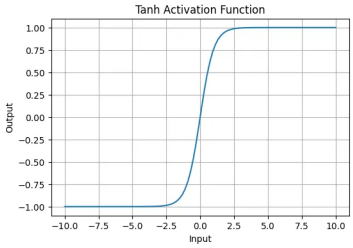


Figure 2.15: Tanh

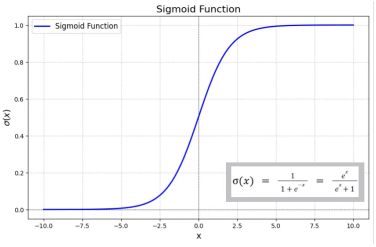


Figure 2.16: Sigmoid function

Macro Precision

Macro Precision is the arithmetic mean of the precision values computed for each class individually. Precision is the ratio of true positive predictions to the sum of true positive and false positive predictions. A high macro precision indicates that the model makes fewer false positive predictions across all classes. Mathematically, macro-averaged Precision is given by:

$$\text{Macro Precision} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FP_i} \tag{2.25}$$

where TP_i and FP_i represent the number of true positives and false positives for class i, and N is the total number of classes.

Macro Recall

Macro recall is the average recall score computed separately for each class. It reflects how well the model identifies actual positive instances across all classes, giving equal weight to each class. It is formulated as:

$$\text{Macro Recall} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FN_i} \tag{2.26}$$

where FN_i represents the false negatives for class i.

Macro F1 Score

The macro F1-score is the average of F1-scores calculated for each class, treating all classes equally regardless of their size. Since the F1-score combines both precision and recall, a high macro F1 value indicates that the model performs well across all classes in terms of both metrics. It is computed as:

$$\text{Macro F1} = \frac{1}{N}\sum_{i=1}^{N} 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{2.27}$$

2.2.23 Technologies Used

Pytorch

PyTorch is an open-source deep learning library widely used in research and production. It provides support for dynamic computational graphs, which is especially useful for models that require frequent adjustments, such as neural networks. It is based on the Torch library and provides an efficient way to perform numerical computing and build deep learning models.

GoogleColab

GoogleColab is a free, cloud-based platform that provides a collaborative environment for data science and machine learning projects. It offers access to GPUs and TPUs, which accelerate model training and experimentation. Google Colab allows users to write and execute Python code in an interactive notebook format and is integrated with Google Drive for easy file management and sharing.

Kaggle

Kaggle provides a community-driven environment for data science and machine learning projects where users can collaborate, share insights, and compete in various predictive modeling tasks. Kaggle also offers free access to computing resources, including GPUs, making it an excellent platform for training machine learning models efficiently and cost-effectively.

React

React is an open-source JavaScript library that enables developers to create reusable UI components that efficiently update in response to changing data, and is effective for building dynamic and responsive web applications. React uses a virtual DOM to ensure that only the necessary parts of the UI are updated, improving performance. It is highly extensible, allowing for seamless integration with other libraries and frameworks.

FFmpeg

FFmpeg is a comprehensive multimedia framework for handling video, audio, and other multimedia files and streams. It supports a vast array of formats and codecs, making it a versatile tool for processing and manipulating media content. FFmpeg is used for tasks like video and audio conversion, streaming, editing, and extracting metadata.

3. Methodology

3.1 Feasibility Study

3.1.1 Technical Feasibility

The algorithms and frameworks selected for this project are well-established and widely used in the industry. The system is designed to process and analyze human poses effectively using different deep learning and machine learning techniques. To ensure efficient computation, GoogleColab and Kaggle Notebook are used. No additional hardware resources are required except commonly available devices such as smartphones or low-cost cameras, ensuring technical feasibility.

3.1.2 Economic Feasibility

The project is economically viable due to the use of open-source frameworks and libraries, eliminating licensing costs. Additionally, the availability of free, time-limited GPUs in Google Colab and Kaggle Notebooks enables cost-effective training of deep learning models, reducing the need for expensive hardware.

3.2 Requirement Analysis

3.2.1 Functional Requirements

The functional requirements for the project include:

1. **Pose detection and Classification:** The system should accurately detect and track the user's body positions and movements from video input and should classify different yoga postures from a predefined set of poses.

2. **Pose Phase Classification:** The system should identify different movement phases of a yoga pose (e.g., transition phase, hold phase).

3. **Feedback:** After each session, the system should provide corrective feedback on the user's performance. This feedback should be instructor-like feedback in concise phrases.

4. **Speech Based Interaction:** The system shall provide real-time corrective feedback through speech output for a better user experience.

5. **Web based User Interface:** It should offer a user-friendly web-based platform for pose analysis.

6. **3D Visualization for Self-Evaluation:** The system should integrate a 3D human model visualization to help users analyze their posture and allow users to compare their pose with an ideal pose for self-assessment.

3.2.2 Non-functional Requirements

Non-functional requirements describe the quality characteristics or attributes of the system that are not directly related to the system's functionality. They specify how well the system should perform, rather than what it should do.

1. **Reliability:** The system should be robust, consistently providing reliable results across a wide range of input scenarios, including different body types, postures, and lighting conditions.

2. **Usability:** The system should offer a user-friendly and intuitive interface, ensuring that users, regardless of technical expertise, can easily navigate and understand the analysis results.

3. **Accuracy:** The AI models used for pose estimation should be highly accurate and robust, achieving optimal results for a wide variety of yoga poses, body types, and movement patterns.

4. **Responsiveness:** The system's interface must be responsive, ensuring it works seamlessly.

3.3 System Design and Implementation

3.3.1 Data Preprocessing

Skeleton Data

For skeleton data for each sample in the dataset 20 frames are sampled. Since, number of samples in the dataset is limited, data augmentation techniques were applied. Various spatial augmentation techniques were applied as suggested in [27] to improve the model generalization by exposing it to more diverse data sample spaces. The data augmentations that was used were:

1. **Additive Gaussian Noise**

   Gaussian noise was added to the input data to introduce slight variations to mimic errors caused during joint pose estimation. Given an input skeleton sequence X, the noisy version $\tilde{X}$ is defined as:

$$\tilde{X} = X + \mathcal{N}(\mu, \sigma^2) \tag{3.1}$$

   where $\mathcal{N}(\mu, \sigma^2)$ represents Gaussian noise with mean μ and standard deviation σ. The value of μ was taken as 0, and the standard deviation was set to 0.01 to avoid heavily distorting the skeleton data.

2. **Random Rotation**

   Here, the skeleton was rotated around the vertical axis (Y-axis) by a random angle θ sampled from a uniform distribution in the range $[-\theta_{\max}, \theta_{\max}]$. This was done to mimic different perspectives from which the human pose may be captured. The rotation matrix $R_y(\theta)$ is given by

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2}$$

   The transformed skeleton $\tilde{X}$ was then computed as:

$$\tilde{X} = R_y(\theta)\, X \tag{3.3}$$

   The angle θ was sampled uniformly from $\left[-30^\circ, 30^\circ\right]$.

3. **Random Scaling**

   To introduce some variability in the skeleton size, a random scale factor s was sampled from a uniform range $[s_{\min}, s_{\max}]$, and then the skeleton was scaled as:

$$\tilde{X} = sX \tag{3.4}$$

   where $s \sim \mathcal{U}(0.9, 1.1)$.

4. **Random Translation**

   Each joint in the skeleton was translated by a random offset T which also sampled from a uniform distribution in the range $[-t_{\max}, t_{\max}]$ along each axis:

$$T = (t_x, t_y, t_z), \quad t_x, t_y, t_z \sim \mathcal{U}(-0.1, 0.1) \tag{3.5}$$

   The transformed skeleton is computed as:

$$\tilde{X} = X + T \tag{3.6}$$

Video Data

For video data, we sourced the required video samples from the URL provided by the 3DYoga90 dataset. Using the `frame_start` and `frame_end` values defined for each video in the dataset, we trimmed the videos to obtain the required pose sequences. Since the sourced videos were in 720p resolution, they were resized to 224×224. Additionally, 30 frames were sampled at equal intervals from the total available frames in a video sample and stored as tensors to improve training efficiency. The various data augmentation strategies used were: Gaussian Blur, Random Horizontal Flip, Color Jitter, and Random Rotation.

3.3.2 Model Architectures

1D CNN LSTM

Layer	Description
Conv1	$K = 3$, 32 output channels, stride 1
Block1	64 output channels, stride 2
Block2	128 output channels, stride 2
Block3	256 output channels, stride 2
Block4	512 output channels, stride 2
LSTM	Hidden size 128
FC	Output size 13

Table 3.1: 1D CNN LSTM Model Architecture.

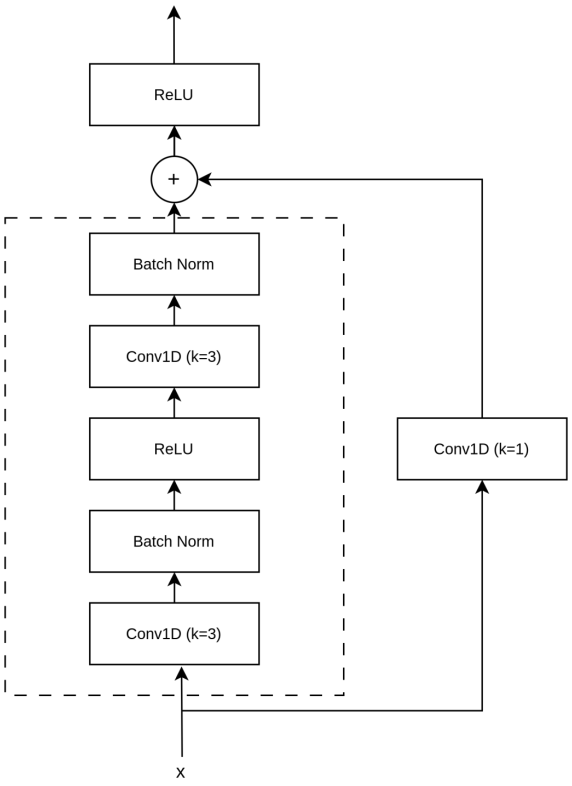


Figure 3.1: ResNet block for 1D-Convolution

Here, Conv1 represents 1D Convolutional operation with kernel size 3. After Conv1, an additional BatchNorm1D layer is applied, followed by a ReLU activation function. The output from this is passed into the four ResNet Blocks marked as Blk as shown in 3.1. The output from Block4 is given to the LSTM layer for temporal feature extraction. Hidden state output from last time step of LSTM is fed into FC layer. This FC layer produces 13 logits, each corresponding to one of the 13 classes.

1D CNN LSTM-Attention

As proposed in [28, 29], instead of taking only the last time step hidden state output from LSTM to feed into the fully connected layer, attention pooling is added that weighs hidden state output from all time steps and aggregates it to create the context vector, then this context vector is fed into the fully connected layer.

$$u_t = \tanh(W \cdot h_t + b) \tag{3.7}$$

$$e_t = v^\top \cdot u_t \tag{3.8}$$

$$\alpha_t = \frac{\exp(e_t)}{\sum_{j=1}^{T} \exp(e_j)} \tag{3.9}$$

$$c = \sum_{t=1}^{T} \alpha_t \cdot h_t \tag{3.10}$$

where,

$h_t \in \mathbb{R}^d$ (hidden state output from LSTM at time step t)

$W \in \mathbb{R}^{d \times d}$ (weight matrix)

$b \in \mathbb{R}^d$ (bias vector)

$u_t \in \mathbb{R}^d$ (hidden representation of h_t)

$v \in \mathbb{R}^{d \times 1}$ (learnable parameter vector)

$e_t \in \mathbb{R}$ (scalar attention score for time step t)

$\alpha_t \in \mathbb{R}$ (scalar attention weight for time step t)

$T \in \mathbb{N}^+$ (total number of time steps)

$c \in \mathbb{R}^d$ (context vector)

Adaptive Spatial Temporal GCN

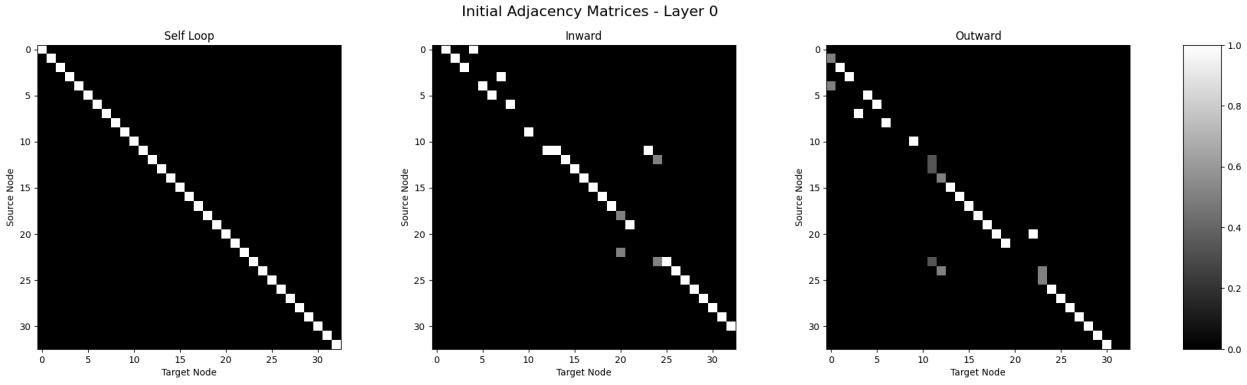


Figure 3.2: Initial Adjacency Matrices

We initialize the adjacency matrices based on the physical connections between joints as defined by Mediapipe's 3D pose estimator. We designate Joint 11 (left shoulder) as the sink node. The inward adjacency matrix represents directed edges pointing toward Joint 11,

while the outward adjacency matrix represents directed edges extending away from Joint 11. Figure 3.2 shows the normalized adjacency matrices.

AGCN-MTCN Block

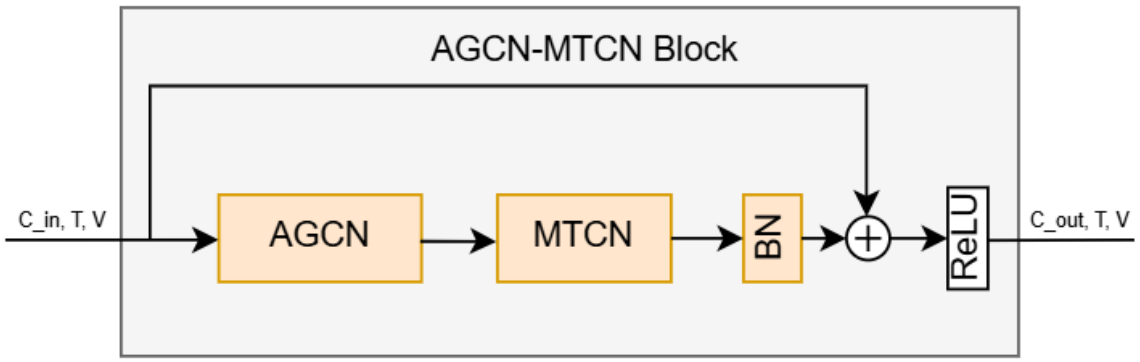


Figure 3.3: AGCN-MTCN Block

The AGCN-MTCN Block, as shown in Figure 3.3, processes input data of the form $f_{\text{in}} \in \mathbb{R}^{C_{\text{in}} \times T \times V}$, where C_{in} represents the number of input channels, T denotes the number of frames, and V indicates the number of nodes.
The block comprises of AGCN component responsible for spatial feature extraction (described in Section 2.2.15), followed by MTCN component responsible for processing temporal dependencies (described in 2.2.16). The MTCN's output is batch-normalized, combined with a residual connection, and passed through a ReLU activation function.

STSAE-GCN Block

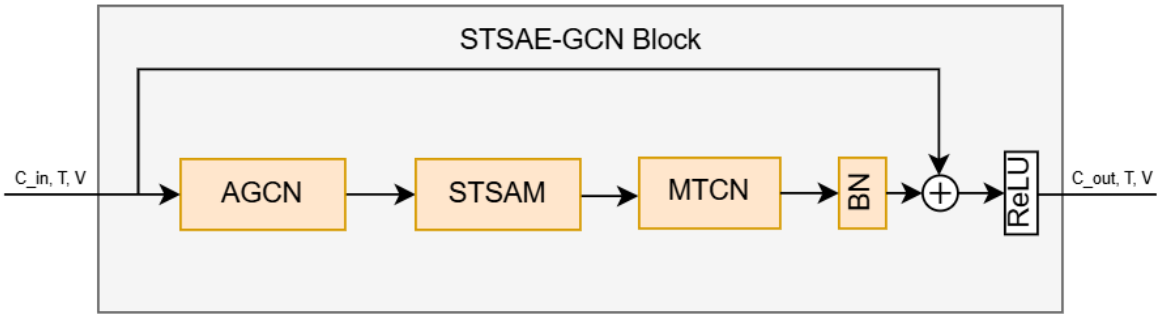


Figure 3.4: STSAE-GCN Block

STSAE-GCN block just inserts STSAM module (described in section 2.2.15) between AGCN and MTCN module.The STSAM consists of spatial self-attention module (SSAM) and temporal self-attention module (TSAM) applied parallely.

AGCN-STC-MTCN Block

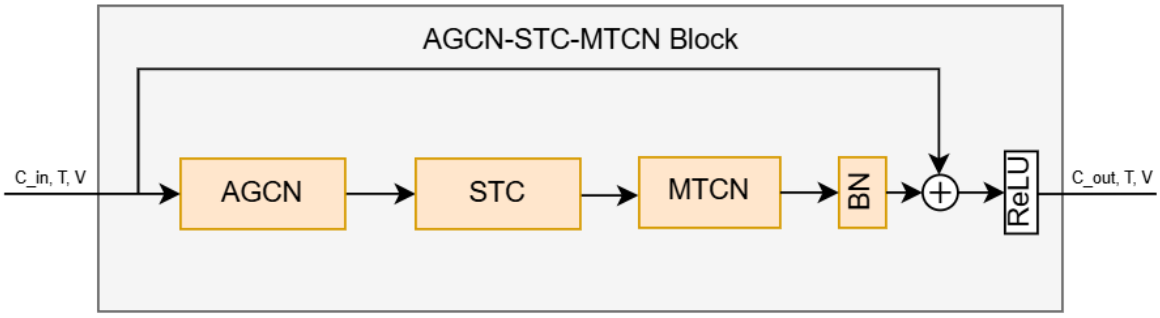


Figure 3.5: AGCN-STC-MTCN Block

Similar to STSAE-GCN block, AGCN-STC-MTCN block shown in Figure 3.5 is the addition of STC-attention module between AGCN and MTCN modules. The STC-attention module consists of SAM (Spatial Attention Module), TAM (Temporal Attention Module), and CAM (Channel Attention Module) applied sequentially with hopes of capturing strengthening feature representation of important joints, frames, and channels respectively.

Overall Architecture

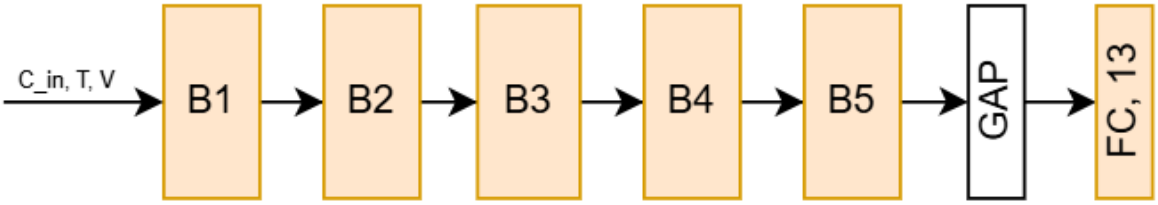


Figure 3.6: Overall Architecture for Adaptive ST-GCN

The overall architecture consists of 5 blocks/layers applied one after another with no residual connection in the input Block, B1. The output channel size of each block is taken as 120. The input skeleton features are successively processed by each layer, then for the output of Block B5, global average pooling is done to reduce the temporal and spatial feature before passing onto the fully-connected layer, which outputs 13 logits corresponding to 13 classes.

2D CNN LSTM

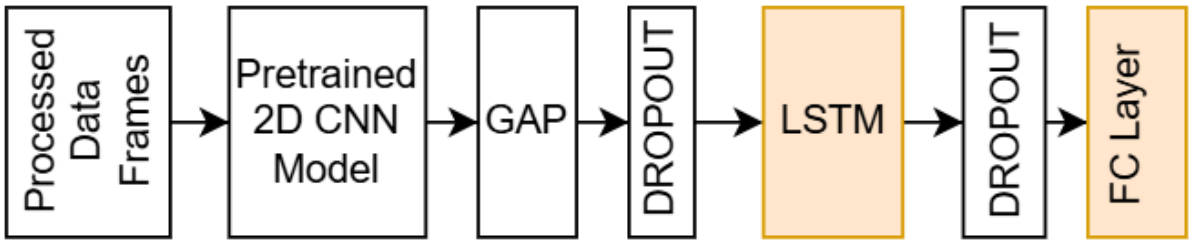


Figure 3.7: 2D CNN LSTM Architecture

For the 2D CNN LSTM architecture shown in Figure 3.7, the input consists of data frames from video sequences, which are fed into a pretrained 2D CNN model(VGG-16, DenseNet-121, or ResNet-18), with its classification head removed, and serves as a spatial feature extractor. The extracted feature maps are then processed using Global Average Pooling (GAP) to reduce spatial dimensions into a fixed-size feature vector. A dropout layer is applied before passing the feature vectors into an LSTM layer, which captures temporal dependencies across frames. Another dropout layer is used before the fully connected (FC) layer to help with regularization. Finally, the FC layer takes the LSTM's hidden state as input and outputs 13 logits, corresponding to 13 classes for classification.

3.3.3 Pose Phase Classification

A yoga pose evolves through three phases:

1. **Preparation Phase:** This phase involves the initial movement and positioning required to transition into the hold phase. It typically consists of dynamic actions such as stretching and bending.

2. **Hold Phase:** In this phase, the body is held in a stable position, maintaining the final form of the yoga pose. The focus is on breath control, balance, and maintaining the correct alignment of the body. This phase can last anywhere from a 10-15 seconds to several minutes, depending on the type of pose and the practitioner's ability to sustain the posture. The hold phase is crucial for building strength and flexibility, as well as for the therapeutic benefits of the pose.

3. **Exit Phase:** The exit phase involves transitioning out of the yoga pose in a controlled and fluid manner.

The dataset samples in 3DYoga90 dataset captures the data related to the preparation phase of a yoga pose, meaning each sample captures the movement required to transition into the hold phase. To correctly classify the yoga pose for our system, we needed to identify which video frames correspond to the movement (preparation) phase which also allows the system to determine when the user transitions into the hold phase.

To achieve this, we defined a three-state state machine, as illustrated in Figure 3.8.

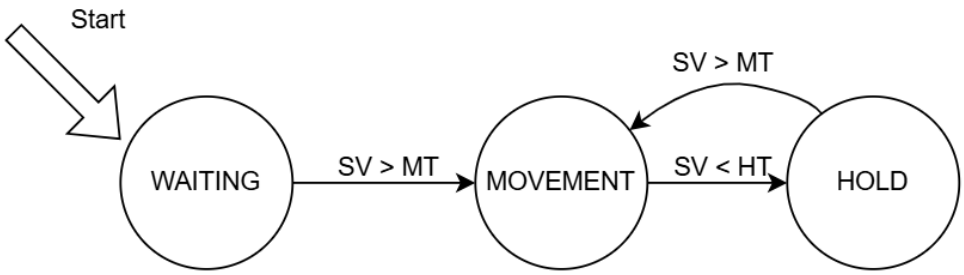


Figure 3.8: State machine for pose phase detection.

SV stands for **Smoothed Velocity**, which represents the aggregated velocity magnitude for all joints in a frame after applying a smoothing function. MT stands for **Movement Threshold**, which defines the minimum velocity required to classify a motion as active movement. HT stands for **Hold Threshold Velocity**, which determines the velocity below which a pose is considered to be in a hold phase.

Here, WAITING state corresponds to the initial time frame where no significant movement has yet occurred. MOVEMENT state corresponds to the preparation phase of a yoga action, while HOLD state corresponds to the hold phase of the yoga action.

The state machine operates on the following formulation. With velocity computation defined as:

$$\mathbf{v}_t^j = \mathbf{p}_{t+1}^j - \mathbf{p}_t^j, \quad \forall\, t \in \{1, 2, \ldots, T-1\} \tag{3.11}$$

where,
$\mathbf{p}_t^j \in \mathbb{R}^3$ is the position of joint j at frame t, where $j = 1, 2, \ldots, N$.
$\mathbf{v}_t^j \in \mathbb{R}^3$ is the velocity of joint j at frame t.
T is the total number of frames.

For the last frame ($t = T$), since there is no next frame, the velocity is set equal to the velocity of the second-to-last frame:

$$\mathbf{v}_T^j = \mathbf{v}_{T-1}^j \tag{3.12}$$

The smoothed velocity at frame t; $V_{\text{smoothed}}(t)$, is computed as per the following formula:

$$V_{\text{smoothed}}(t) = \frac{1}{5} \sum_{\tau=t-4}^{t} \min \left(1.5, \sum_{i=1}^{N} \sqrt{v_{i,x}(\tau)^2 + v_{i,y}(\tau)^2 + v_{i,z}(\tau)^2} \right) \tag{3.13}$$

Here, N represents the total number of joints. For each joint i, we have velocity components along the x, y, and z coordinates at a given frame τ, denoted as $v_{i,x}(\tau)$, $v_{i,y}(\tau)$, and $v_{i,z}(\tau)$, respectively.

If $V_{\text{smoothed}}(t)$ meets the state transition threshold for a specified count of frames, then the state transition occurs. While transitioning from the MOVEMENT state to the HOLD state, frames collected during the MOVEMENT state are given to the action classification module.

3.3.4 Biomechanical Feature Extraction

Joint angles are the primary features extracted from the user's skeleton representation by this module, along with joint velocity as described in the previous section. These joint angles help quantify the spatial relation between connected body segments, which is later used to provide corrective feedback.

For each joint of interest, the angle is calculated based on the joint positions of a triplet of joints, with the central joint acting as the vertex of the angle. These triplets define two vectors originating from the central joint, and the angle between these vectors represents the joint angle for the central joint.

For example, the right shoulder angle involves three joints: the right hip (Rh), right shoulder (Rs), and right elbow (Re). The two vectors with the right shoulder as the origin are defined as:

$$\vec{v_1} = p_{Rh} - p_{Rs}, \quad \text{(vector from right shoulder to right hip)} \tag{3.14}$$

$$\vec{v_2} = p_{Re} - p_{Rs}, \quad \text{(vector from right shoulder to right elbow)} \tag{3.15}$$

where p_{Rh}, p_{Rs}, and p_{Re} are the joint positions of the right hip, right shoulder (vertex), and right elbow, respectively.

The right shoulder angle, denoted as θ_{Rs}, is the angle between $\vec{v_1}$ and $\vec{v_2}$, computed as:

$$\theta_{Rs} = \cos^{-1} \left(\frac{\vec{v_1} \cdot \vec{v_2}}{|\vec{v_1}|\,|\vec{v_2}|} \right) \tag{3.16}$$

where, $\vec{v_1} \cdot \vec{v_2}$ is the dot product of vectors $\vec{v_1}$ and $\vec{v_2}$, $|\vec{v_1}|$ is the magnitude of $\vec{v_1}$, $|\vec{v_2}|$ is the magnitude of $\vec{v_2}$.

The same formulation is applied similarly to other joints (e.g., left shoulder, right knee) using their respective joint triplets to get the respective joint angles, giving a comprehensive set of biomechanical features for analysis to provide corrective feedback.

3.3.5 Corrective Feedback Generation

The corrective feedback generation module utilizes biomechanical features specifically joint angles to generate instructor-like guidance for yoga pose correction. Focusing more on extensibility and flexibility for generation of corrective feedback instead of rule based approach to generate textual feedback generation as done in AIFit [30], taking inspiration from UbiPhysio [31], this module was designed such that the biomechanical features are formatted in a template and processed by a large language model (LLM), specifically the LLaMA-3.1-8B-Instant variant, configured to emulate a yoga coach providing short and actionable feedback. The system is designed to identify the most significant misalignment in the user's pose and articulate a concise correction, constrained to 5-10 words, mirroring short verbal instructions that a instructor would give.

System and User Prompt Template

```
System Content: "You are a helpful yoga coach providing precise,
                constructive feedback."

User Content:
Action: <ACTION-TYPE>

Current Joint Analysis:
- <JOINT_ANGLE>:
  Current: <VALUE>
  Target: <VALUE>
  Involved parts: <JOINT_1, JOINT_2, JOINT_3>
...

Context:
- This is <ACTION-TYPE> yoga-pose.
- Focus on angles representing proper body alignment.
Task:
Respond with the feedback in 5-10 words only like a yoga instructor.
Avoid any extra explanations or numbers.
Only focus on most significant flaw and give output like the user
is listening to the feedback while doing the pose.
```

Here, the `<ACTION-TYPE>` represents the pose name, determined by the action classi-

fication module, while the <JOINT_ANGLE> denotes a specific joint angle, with associated parameters, such as current angles and involved joints; obtained from the biomechanical feature extraction module. For a given <ACTION-TYPE>, relevant <JOINT_ANGLE>s are selected using a simplified rule-based approach, defined by conditional logic that assigns target joint angles based on the requirements of the corresponding yoga pose for proper execution.

3.3.6 Implementation

The overall system was developed using Python for the backend and JavaScript for the frontend. A very minimalistic application using 'React' was developed for using as the frontend. The communication between the frontend and the backend occurs through websockets, where the 'FastAPI' web framework was utilized as a facilitator. There is also an optional component that is used to generate the 3D human mesh using the SMPLR-x project that runs out of the usual backend services, and whose api has to be explicitly specified in the frontend if one also needs the 3D mesh to be available during usage of the application.

The frontend consists of a mechanism to utilize both past video sources as well as a live webcam feed as input to the overall pipeline. For each frame sampled currently at a 10 fps rate, the backend also replies with the mediapipe keypoints it has generated, which the frontend illustrates on a separate image canvas. After a yoga pose has been identified and the feedback has been generated, the frontend plays the audio feedback live immediately. After the user ends the streaming job, the user can still review the yoga poses that have been performed in the current session, along with the classified pose name and the feedbacks provided for those poses.

The backend follows a simple request-reply protocol, but is built upon the persistent WebSocket protocol. JSON format has been used to send both request as well as replies, except for the streaming binary data that is sent directly for each frame. The backend calls mediapipe upon each incoming frame and generates a set of keypoints, those keypoints are then subsequently sent to a pose phase detection state machine, which can identify the end of a yoga pose based on the fact that 'all yoga poses have a slight hold-still-and-breathe phase after each major pose'.

After the end of the yoga pose has been identified, the frames involved with the yoga pose are sent to the classifier. The classifier selected is currently a graph neural network-based, described in the sections above, which has been written using the PyTorch framework. The classifier after obtaining the keypoints of the sampled frames, produces a yoga class. The detected yoga class and the keypoints from the last frame are used to generate feedback. Currently the textual feedback generation is done using an LLaMA model hosted through Groq, so a Groq API key has to be fed into the backend as an environment variable. When

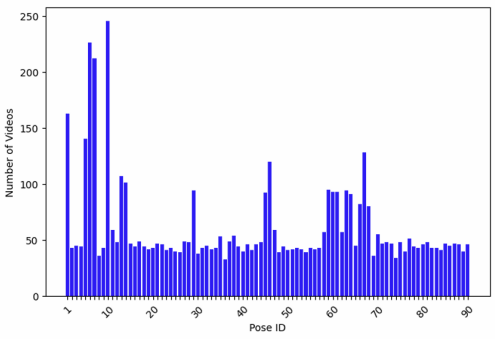
the said textual feedback is generated, then the 'Piper Text to Speech' tool installed locally in the system with the backend is utilized through its python bindings to generate a human voice of the generated feedback. All of these are replied to the frontend as soon as they are available, although some anomalies in the replies can be expected, as sometimes the CPU usage can get unpredictable.

4. Experimental Setup

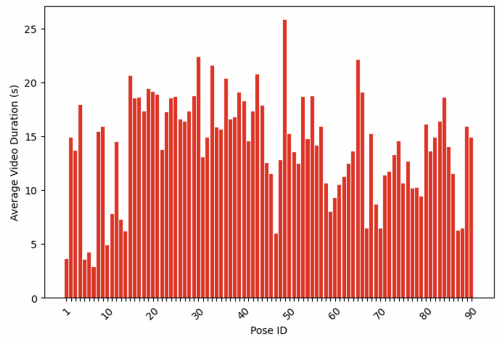
4.1 Dataset

The dataset that is primarily used in the project is the one provided by 3DYoga90: A Hierarchical Video Dataset for Yoga Pose Understanding [12]. The dataset comprises 3 levels of hierarchy as seen in 2.3, for the three levels, the categorical labels are assigned: level 1, level 2, and level 3. In level 1, there are six classes; in level 2, twenty classes; and in level 3, ninety classes are present. JSON files containing web-based resources such as YouTube links or video identifiers are provided by the dataset, which was curated by six individuals. The primary data criterion ensured by the dataset was the visibility of all 33 skeleton points. The dataset comprises of yoga postures where a single person performs the posture in an uninterrupted sequence.

The dataset comprises a total of 6,177 original video clips collected from 2,170 different sources spanning various countries. From these clips, a total of 5,482 skeleton sequences were extracted. The distribution of level 3 poses is presented in 4.1a. The overall mean sequence duration is approximately 12.16 seconds, with sequence duration varying from a minimum of 2.83 seconds to a maximum of 25.84 seconds. Detailed information on the distribution of level 3 pose class duration is presented in 4.1b.



(a) Pose ID vs Number of videos



(b) Pose ID vs Average Video Duration

Figure 4.1: Dataset Statistics

Pose ID refers to the pose identification at level 3.

4.1.1 Selected Yoga Poses

For the model training and evaluation, a set of yoga poses was chosen from the 3DYoga90 dataset. These poses were selected with a focus on achieving full-body coverage, while also considering the availability of videos in the dataset to ensure a sufficient amount of data for each pose. The aim was to incorporate poses that span a range of body movements and alignments. The selected poses are as follows:

1. Downward Dog (Adho Mukha Svanasana)
2. Standing Forward Bend (Uttanasana)
3. Half-Way Lift (Ardha Uttanasana)
4. Mountain (Tadasana)
5. Chair (Utkatasana)
6. Cobra (Bhujangasana)
7. Cockerel (Kukkutasana)
8. Extended Triangle (Utthita Trikonasana)
9. Extended Side Angle (Utthita Parshvakonasana)
10. Corpse (Savasana)
11. Staff (Dandasana)
12. Wind-Relieving (Pavanamuktasana)
13. Fish (Matsyasana)

Figure 4.2: Downward Dog

Figure 4.3: Standing Forward Bend

Figure 4.4: Half-Way Lift

Figure 4.5: Mountain

Figure 4.6: Chair

Figure 4.7: Wind-Relieving

Figure 4.8: Cockerel

Figure 4.9: Extended Triangle

Figure 4.10: Extended Side Angle

Figure 4.11: Corpse

Figure 4.12: Staff

Figure 4.13: Cobra

Figure 4.14: Fish

Figure 4.15: Selected Yoga Poses

4.1.2 Data Distribution of Selected Poses

	Video Data		Skeleton Data	
Pose Name	Training	Testing	Training	Testing
Chair	118	22	118	22
Cobra	57	15	109	19
Cockerel	102	18	102	18
Corpse	68	14	81	14
Downward Dog	205	37	208	37
Extended Side Angle	85	16	85	16
Extended Triangle	88	17	90	17
Fish	63	14	79	14
Half-Way Lift	180	32	180	32
Mountain	137	25	138	25
Staff	48	15	79	15
Standing Forward Bend	191	34	192	34
Wind-Relieving	60	15	79	15

Table 4.1: Dataset Statistics

4.2 Training Details

All experiments were conducted using the PyTorch deep learning framework.

1D CNN LSTM

15% of the training dataset was used as the validation set. The model was trained for 50 epochs with a batch size of 16. The Adam optimizer was used with an initial learning rate of 0.001 and a weight decay of 0.001. The LSTM hidden channel size was set to 128. Cross-entropy loss was used as the loss function. The learning rate was reduced by a factor of 0.1 when the validation loss did not improve for 5 consecutive epochs.

Adaptive ST-GCN

For training all variants of Adaptive ST-GCN, five blocks/layers were used, with the output channel of each block set to 120. A weight decay of 0.005 was used, with a batch size of 32, and the model was trained for 50 epochs. The Adam optimizer was used, with the initial learning rate set to the same value as the 1D CNN LSTM model (0.001). The learning rate

was reduced by a factor of 0.1 when the validation loss didn't improve for 5 consecutive epochs. However, the number of training epochs was not explicitly mentioned.

2D CNN LSTM

Three variants of pretrained 2D CNN models (ResNet-18, VGG-16, and DenseNet-121) were tested. Training was performed with a batch size of 16, weight decay set to 0.01, and a dropout rate of 0.5 for both dropout layers. The LSTM hidden size was set to 64 for ResNet-18 and DenseNet-121, while for VGG-16 it was set to 32. AdamW optimizer was used with initial learning rate set to 0.0005, which was reduced by a factor of 0.5 if the validaiton loss didn't improve for 5 consecutive epochs. The configuration of the fully connected layer was simple, as it took an LSTM hidden size input and produced 13 outputs.

5. System Design

The architecture for the system consists of 3 main components:

1. **Pose Classifier**
2. **Statistical Coach**
3. **User Interaction and Post Analysis**

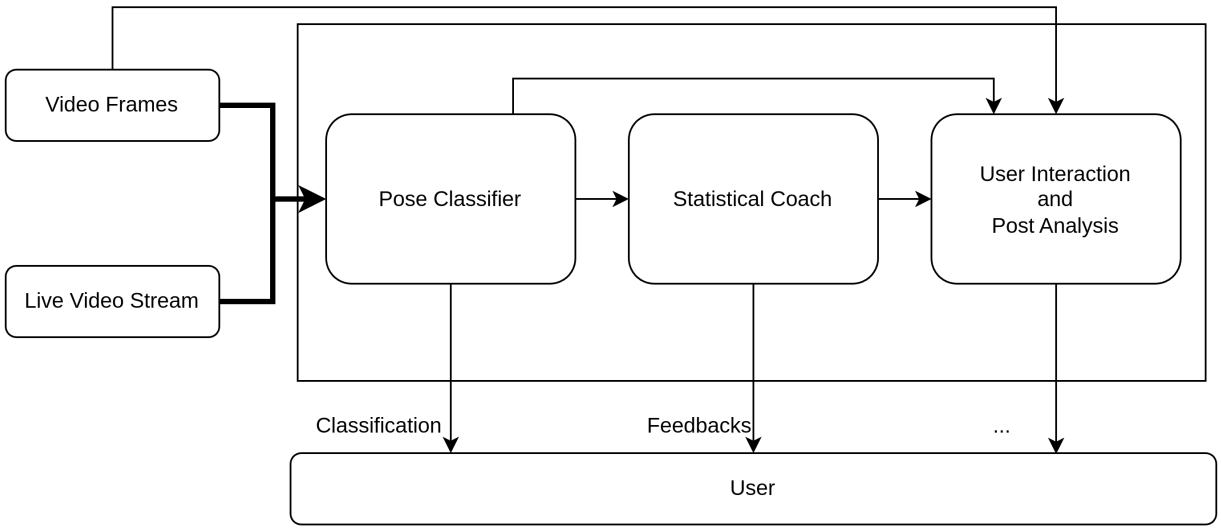


Figure 5.1: Overall System Diagram

Each of the three major components is specially dependent upon the results produced by the preceding component. In overall, these three sub-systems work together and process a live sequence of image frames, i.e, in a streaming fashion, classify the human yoga pose being performed in those images, produce the feedback upon the particular yoga pose being performed and finally store and allow reviewing of the poses encountered during the yoga session.

5.1 Sub-system Descriptions

5.1.1 Pose Classifier

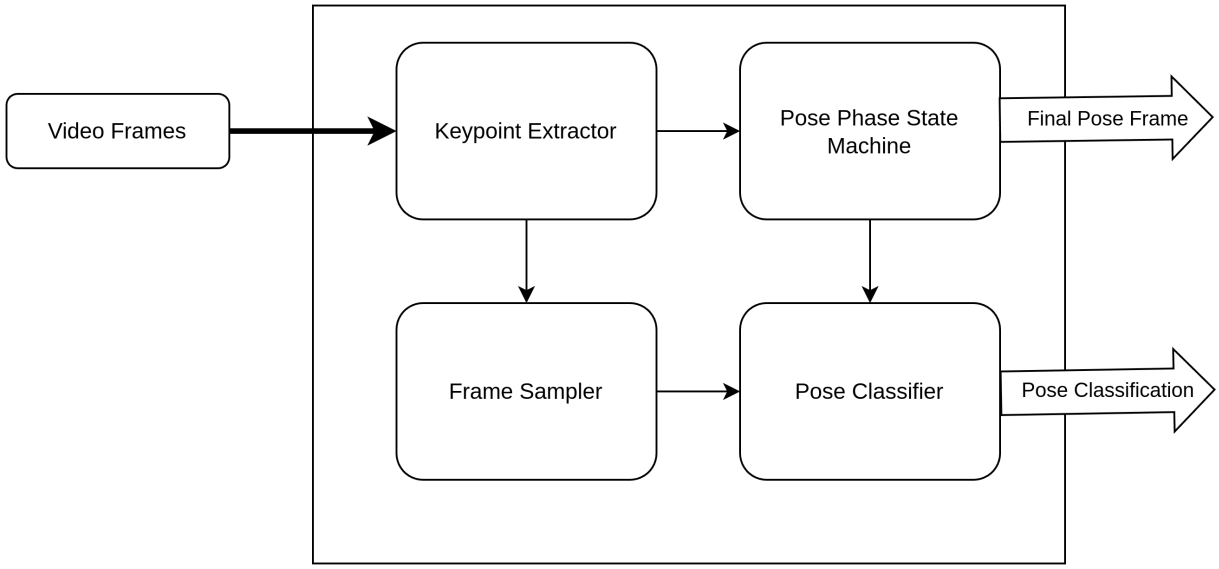


Figure 5.2: Pose Classifier System Diagram

This diagram shows us the generic structure of our skeleton-based yoga pose classification system. As an input it takes in a sequence of video frames, which is then passed through a keypoint extractor, which in other words shows a skeletal structure of the person appearing in the video sequence. Then a state machine is used to find out the end of a yoga pose in the sequence of video frames streaming into the system. Only when the state changes in the state machine indicates the end of the yoga pose, is the pose classification component is triggered. The pose classification take in a sampled sequence of some past frames and identifies the class of the latest pose being performed by the suer.

5.1.2 Statistical Coach

This diagram shows the formulation of the simple statistical coaching system in place for suggesting the user with improvements upon the yoga pose currently being performed. It is also triggered after the pose classification task is completed. As input, it takes in the last frame keypoints of the video sequence for the latest identified pose and compares it with the ideal keypoints for that yoga pose, and finds the crucial differences. It is then fed into an LLB-based component that translates the numerical differences into human-understandable language, which is then fed through a text-to-speech (TTS) system and finally returned to the user.

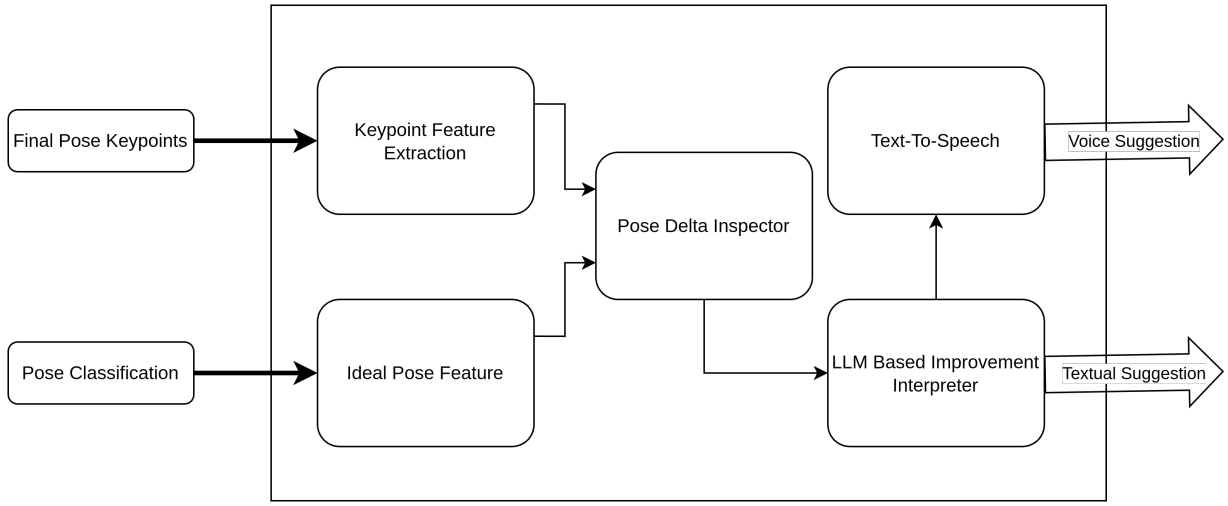


Figure 5.3: Statistical Coach System Diagram

5.1.3 User Interaction and Post Analysis

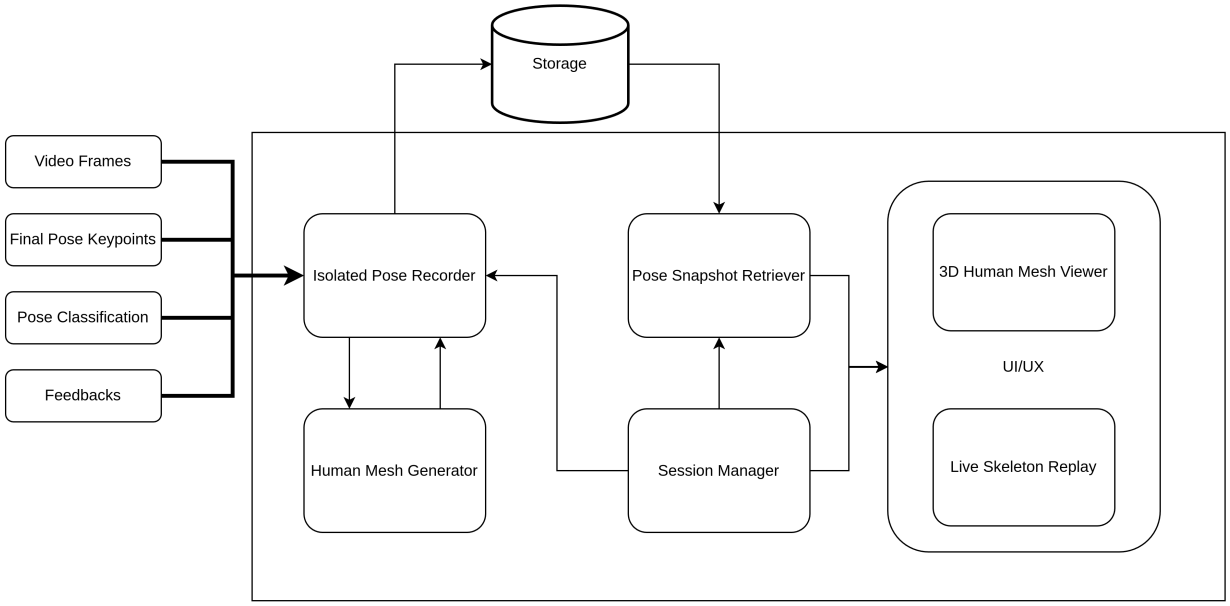


Figure 5.4: User Interaction and Post Analysis System Diagram

The system diagram above covers the post-analysis portion of the overall system, along with how UI/UX is structured in the web application. For this project, there exists only a simple session storage that stores the list of all the major frames in which the yoga poses have been detected, along with the classification and the feedback obtained at those frames. There also exists a pair of components that helps the user generate as well as view a 3D mesh of the yoga pose that the user had performed at that time instant, which would allow the user to

understand the error in their performance more intuitively and from all directions.

5.2 Sub-system Interactions

The following behavioral diagrams represent the overview of the interactions that happens among those components that have been described above.

5.2.1 System-Sequence Diagram

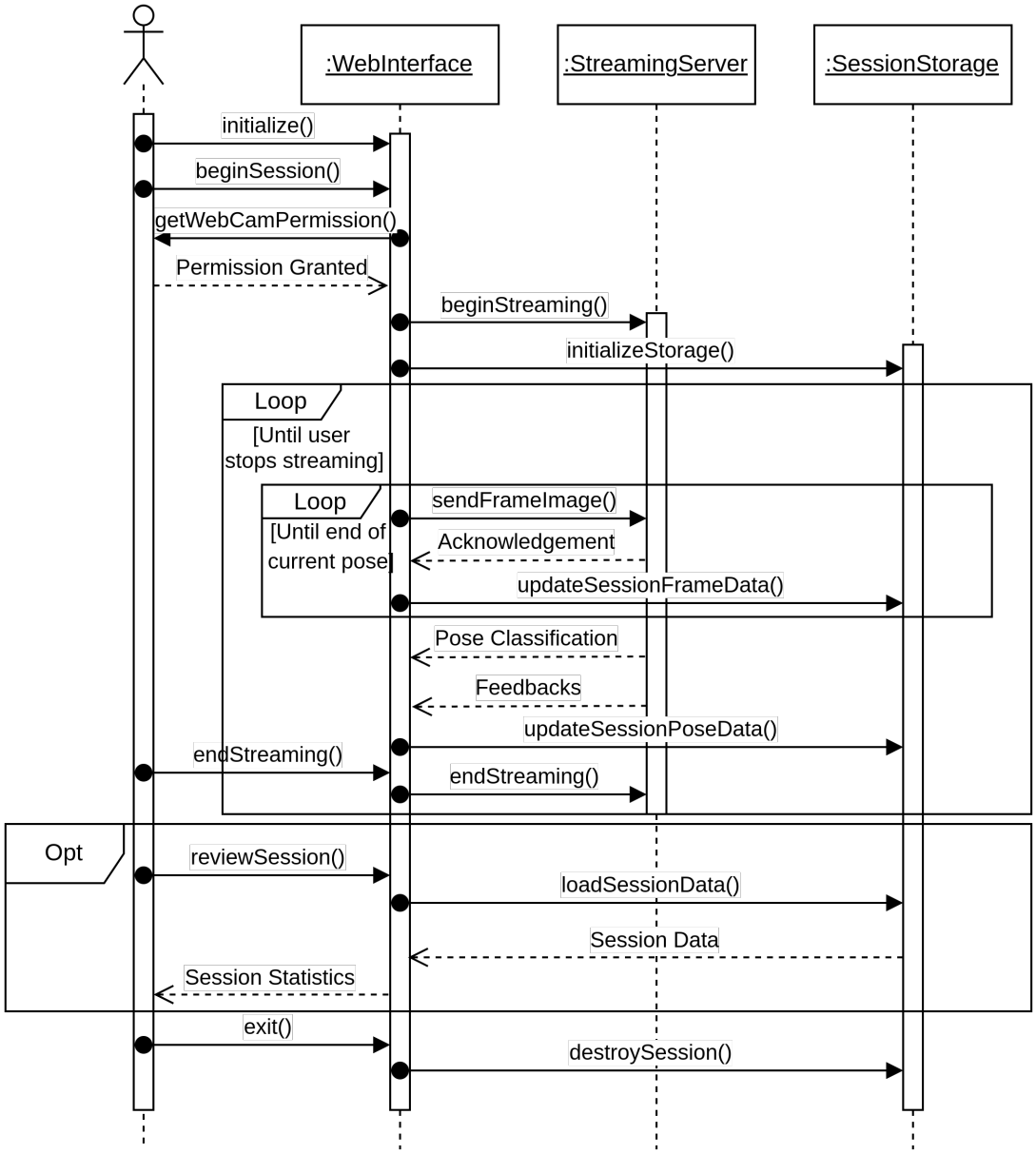


Figure 5.5: System-Sequence Diagram

This sequence diagram shows us the complete lifecycle of our web-based streaming yoga session classification and coaching system, detailing interactions between a user, the web interface, and a streaming server backed by session storage. From the diagram, we can see the key processes, including session initialization, webcam permission management, streaming, pose classification, session data handling, and session termination.

5.2.2 Classification and Coaching Sequence Diagram

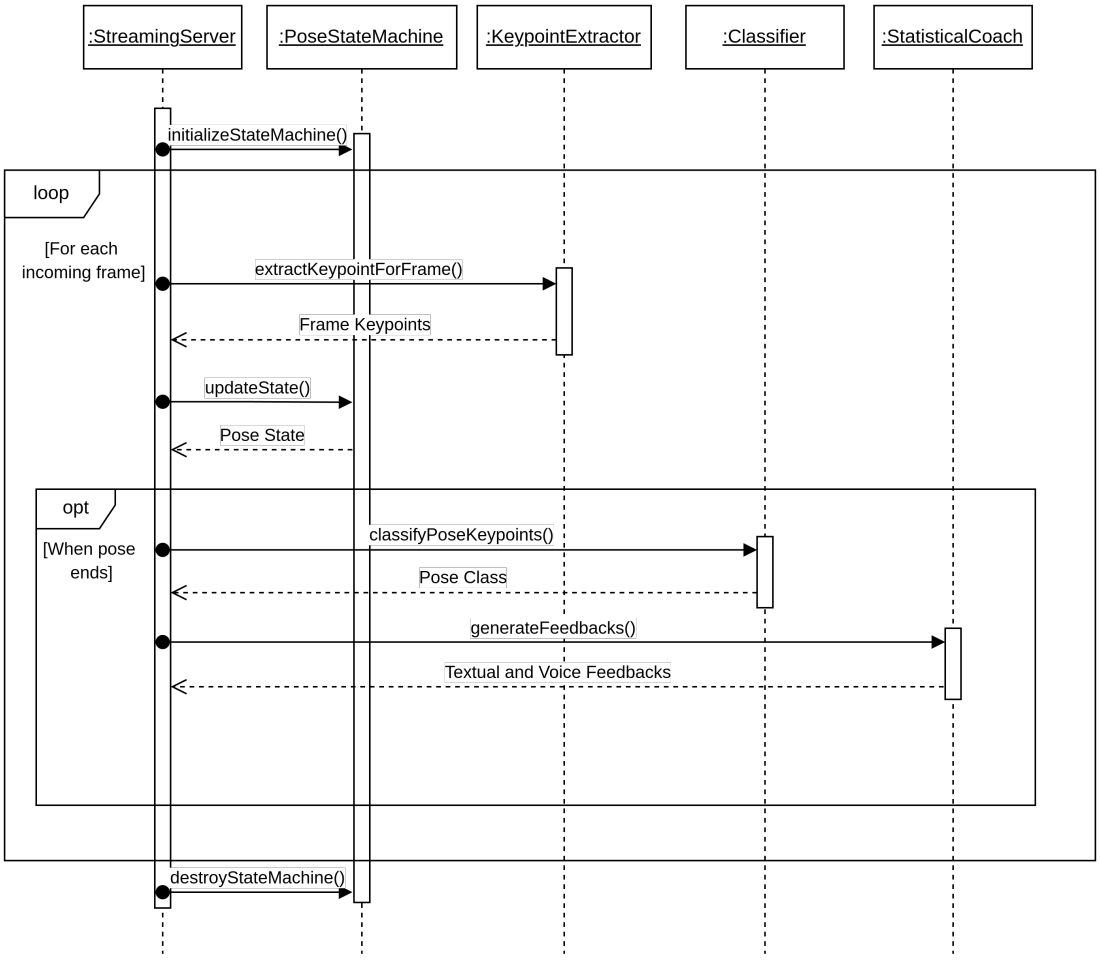


Figure 5.6: Classification and Coaching Sequence Diagram

This sequence diagram shows the workflow of our yoga pose recognition system, showing interactions between key components of our system, as StreamingServer, PoseStateMachine, KeypointExtractor, Classifier, and StatisticalCoach. The diagram depicts the process of extracting keypoints from incoming frames, classifying pose states, and generating feedback, with a loop for processing each incoming frame and an optional section for handling pose-related operations.

6. Results & Discussion

6.1 Results

6.1.1 1D CNN LSTM

1D CNN LSTM Without Attention Pooling

Training and Validation Performance

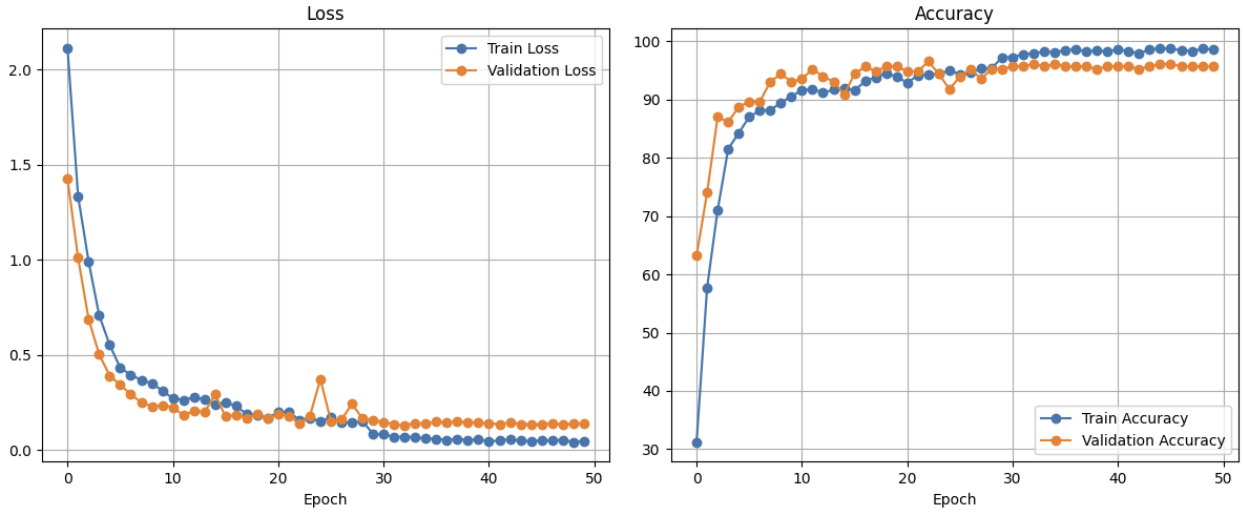


Figure 6.1: Training and validation loss curves for 1D CNN LSTM

Confusion Matrix and Test Metrics

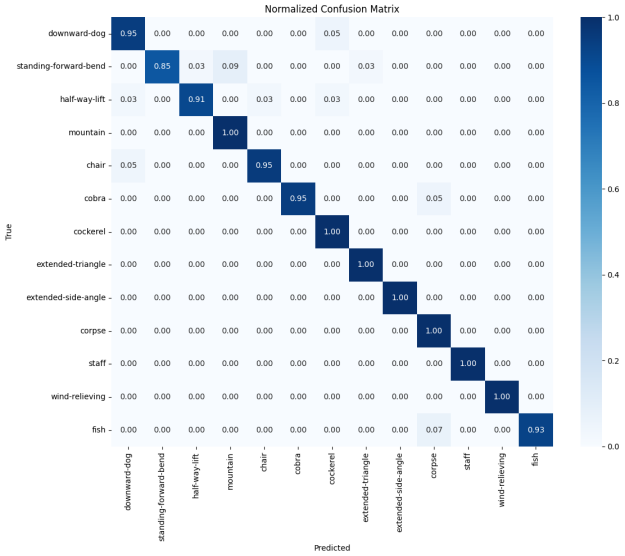


Figure 6.2: Confusion Matrix of 1D CNN LSTM model on test data.

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.17797	95.32	0.9568	0.9640	0.9587

Table 6.1: Test results for 1D CNN LSTM model evaluation

1D CNN LSTM with Attention Pooling

Training and Validation Performance

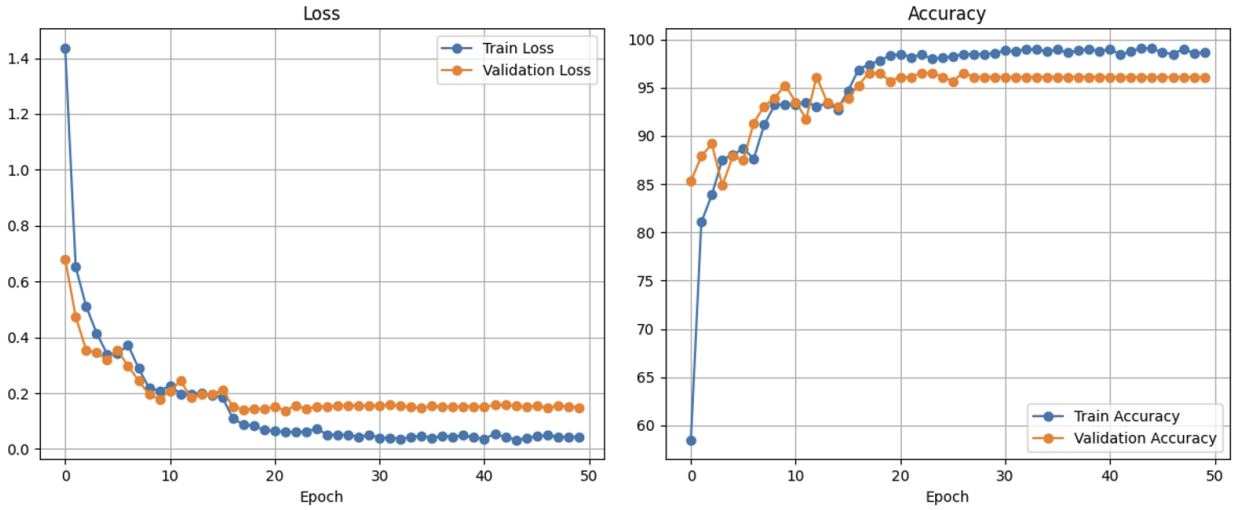


Figure 6.3: Training and validation loss curves for 1D CNN LSTM-Attention

Confusion Matrix and Test Metrics

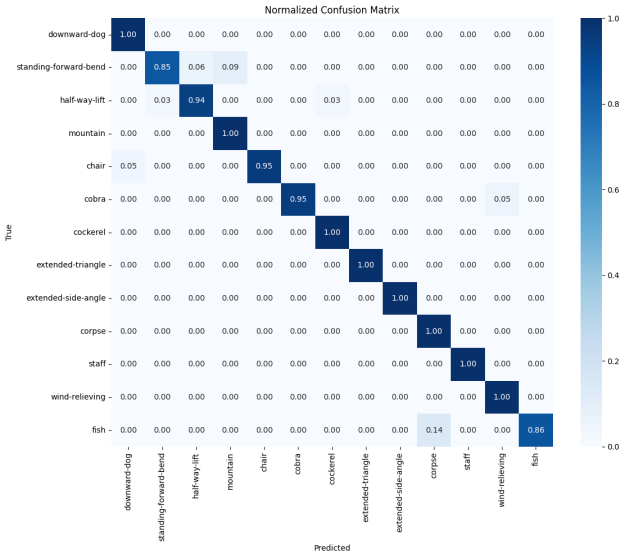


Figure 6.4: Confusion Matrix of 1D CNN LSTM-Attention

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.2233	96.043	0.9638	0.9650	0.9629

Table 6.2: Test results for 1D CNN LSTM-Attention

6.1.2 2D CNN LSTM

ResNet-18 LSTM

Training and Validation Performance

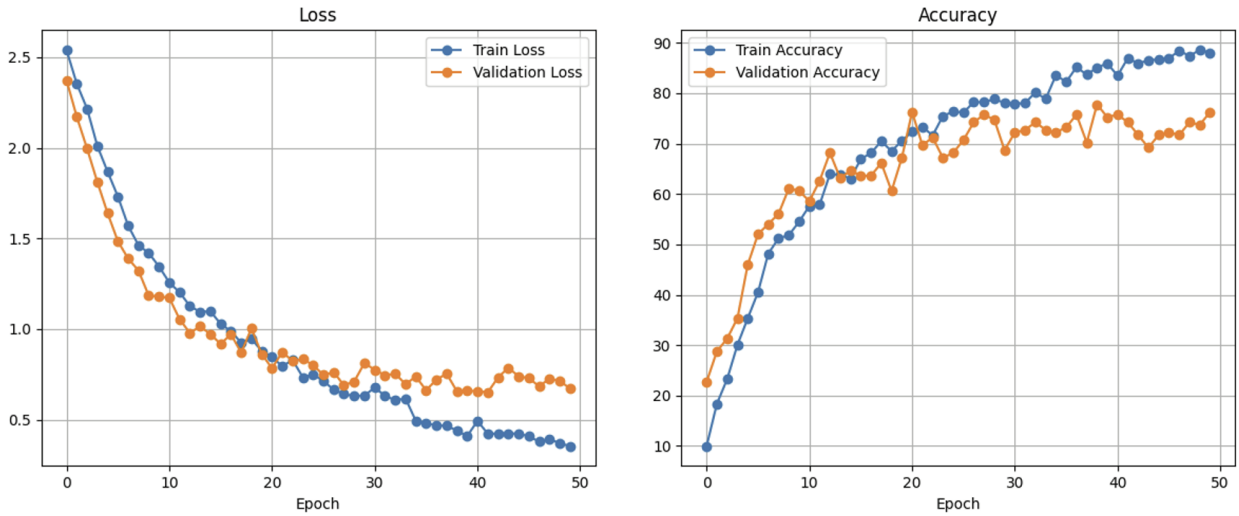


Figure 6.5: Training and validation loss curves for ResNet-18 LSTM

Confusion Matrix and Test Metrics

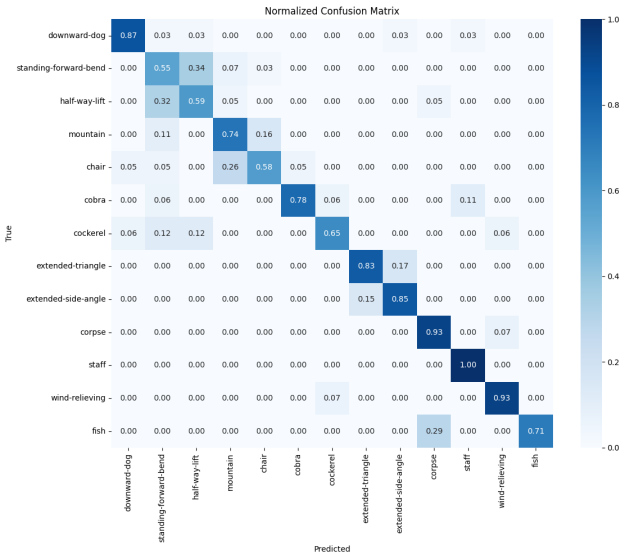


Figure 6.6: Confusion Matrix of ResNet-18 LSTM model on test data.

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.7605	75.11	0.7816	0.7697	0.7691

Table 6.3: Test results for ResNet-18 LSTM model evaluation.

DenseNet-121 LSTM

Training and Validation Performance

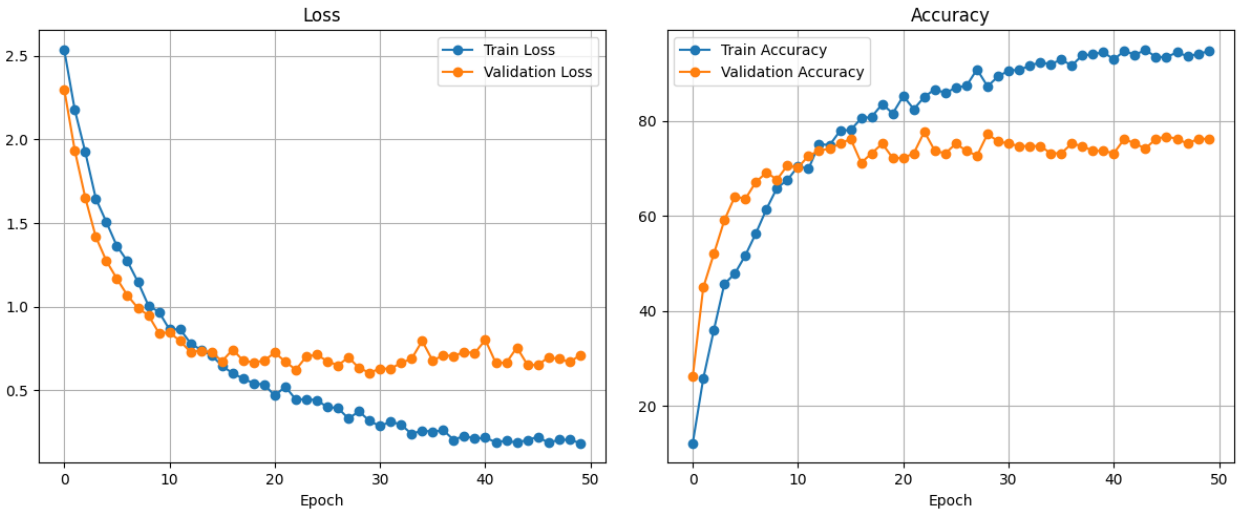


Figure 6.7: Training and validation loss curves for DenseNet-121 LSTM model.

Confusion Matrix and Test Metrics

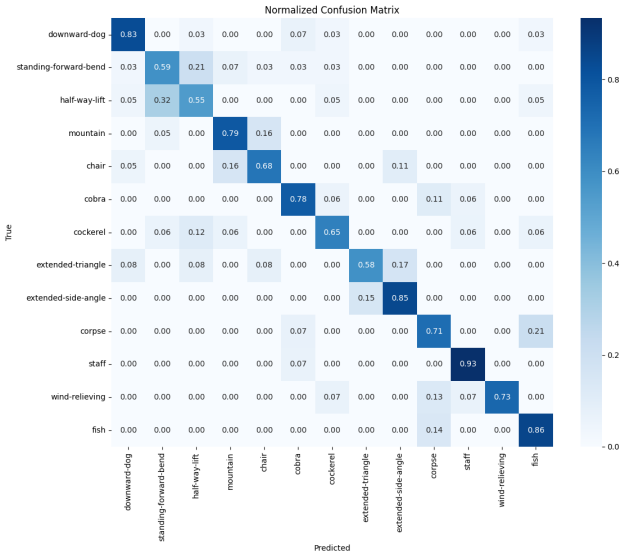


Figure 6.8: Confusion Matrix of DenseNet-121 LSM model on tes data.

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.8354	72.57	0.7345	0.7332	0.7290

Table 6.4: Test results for DenseNet-121 LSTM model evaluation.

VGG-16 LSTM

Training and Validation Performance

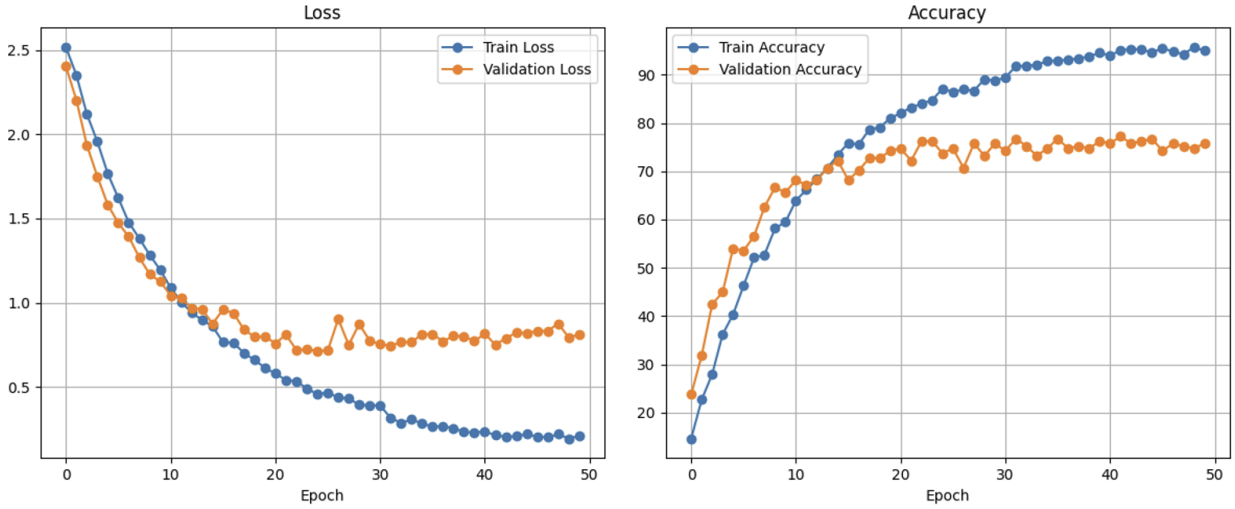


Figure 6.9: Training and validation loss curves for VGG-16 LSTM model

Confusion Matrix and Metrics

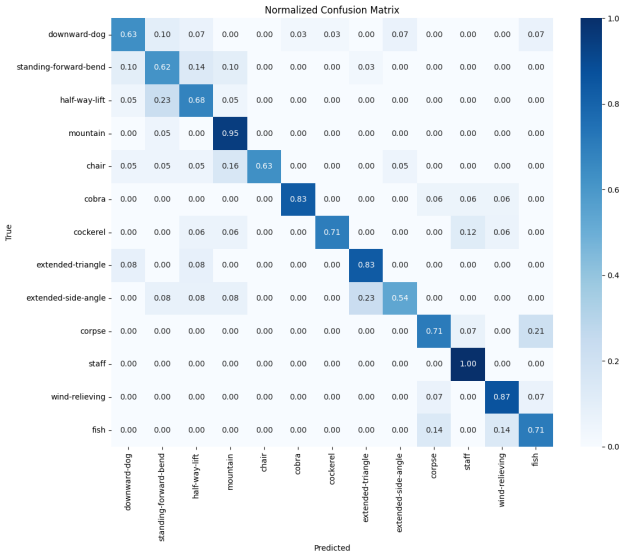


Figure 6.10: Confusion Matrix of VGG-16 LSTM model on test data

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.8808	73.42	0.7551	0.7478	0.7418

Table 6.5: Test results for VGG-16 LSTM model evaluation.

6.1.3 Adaptive ST-GCN Architecture

STSAE-GCN

Training and Validation Performance

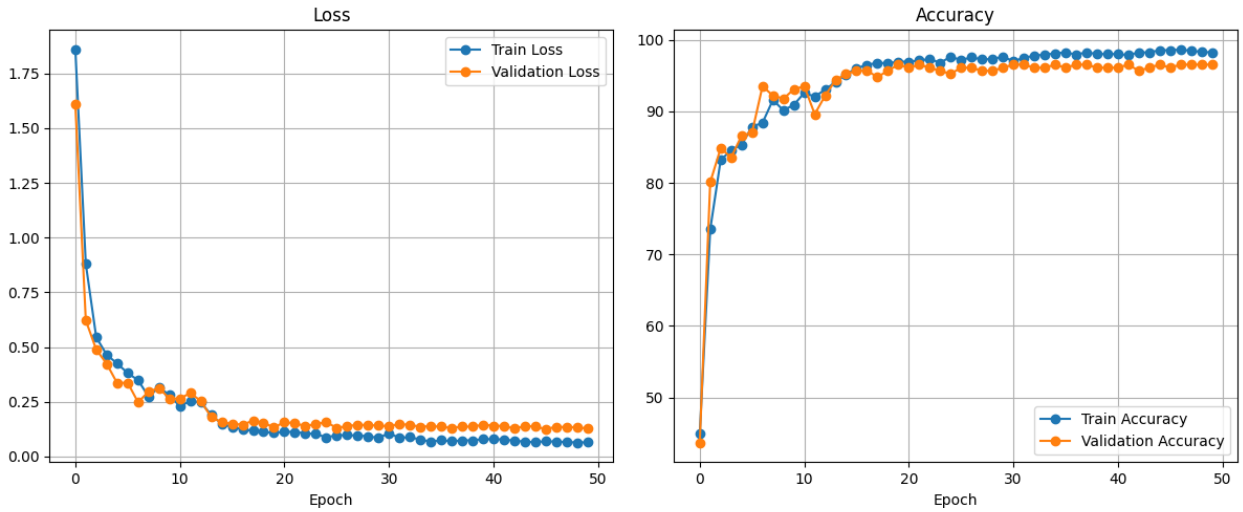


Figure 6.11: Training and validation loss curves for STSAE-GCN

Confusion Matrix and Metrics

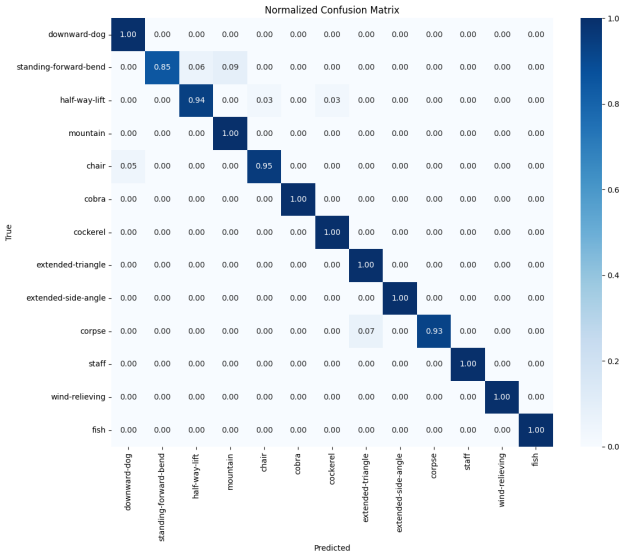


Figure 6.12: Confusion Matrix of STSAE-GCN model on test data

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.1409	96.76	0.9732	0.9745	0.9729

Table 6.6: Test results for STSAE-GCN model evaluation

AGCN-MTCN

Training and Validation Performance

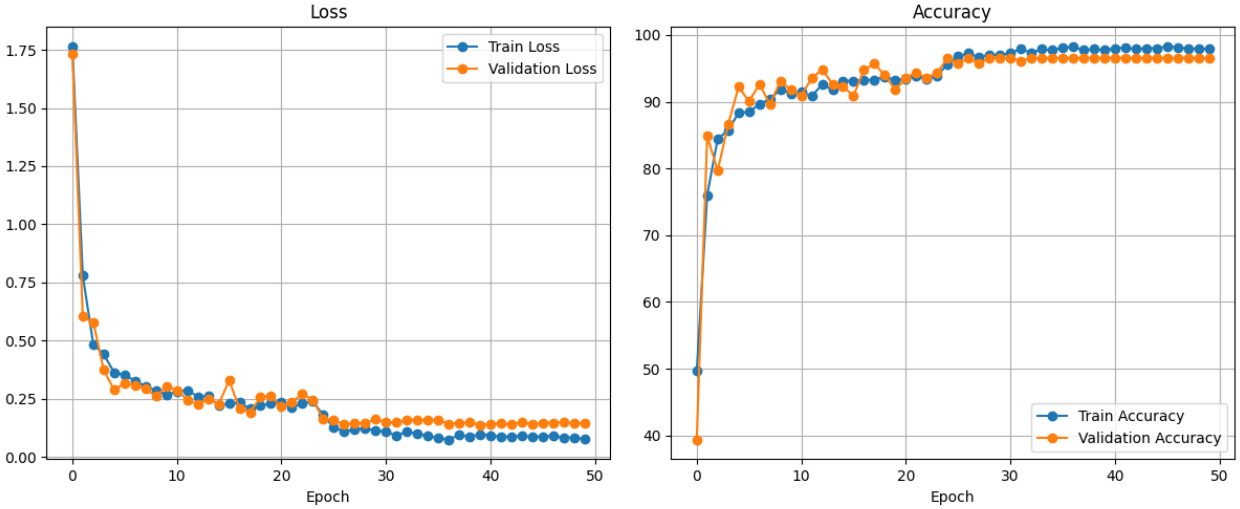


Figure 6.13: Training and validation loss curves for AGCN-MTCN model

Confusion Matrix and Metrics

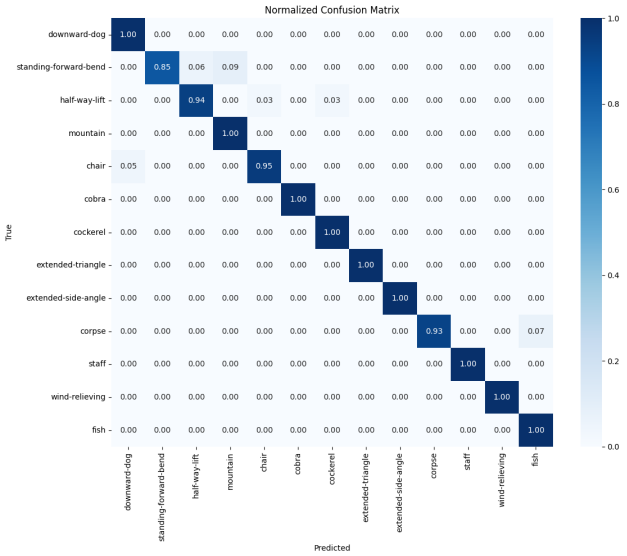


Figure 6.14: Confusion Matrix of AGCN-MTCN model on test data.

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.1549	96.76	0.9723	0.9745	0.9725

Table 6.7: Test results for AGCN-MTCN model evaluation.

AGCN-STC-MTCN

Training and Validation Performance

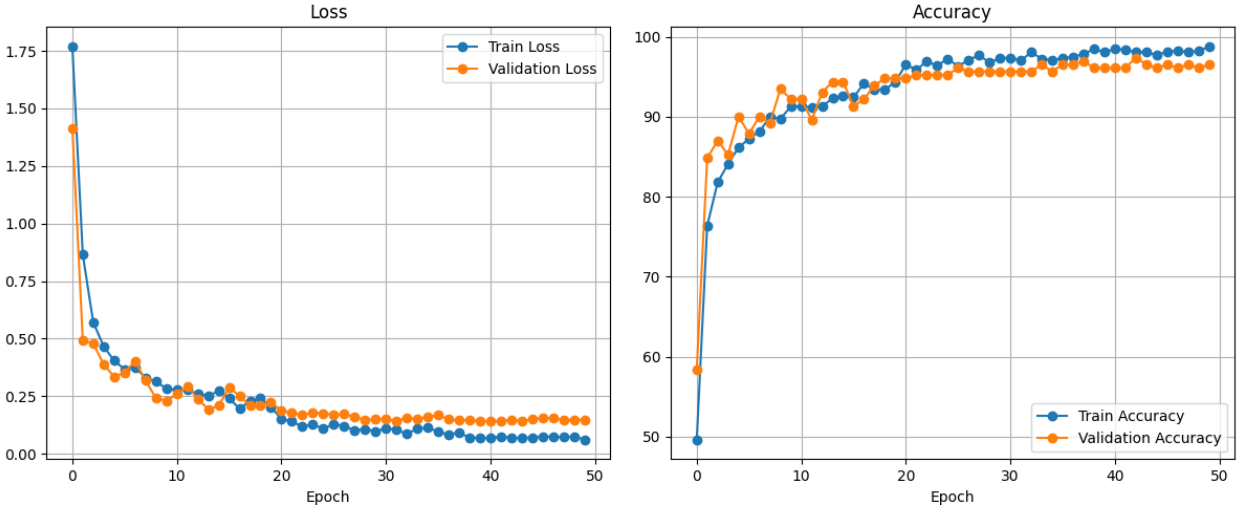


Figure 6.15: Training and validation loss curves for AGCN-STC-MTCN model

Confusion Matrix and Metrics

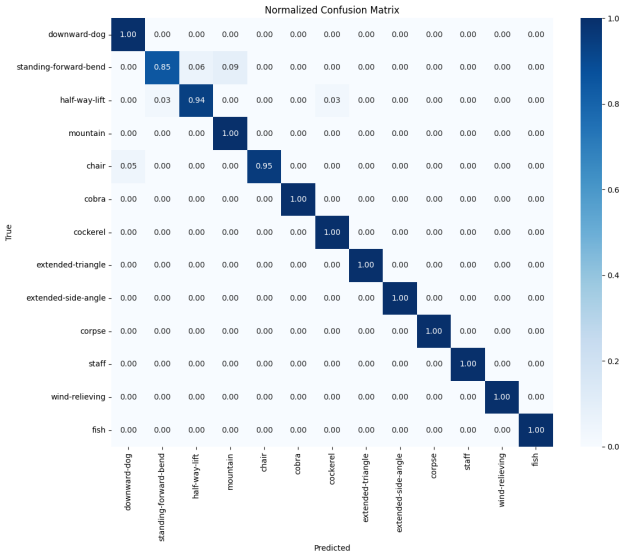
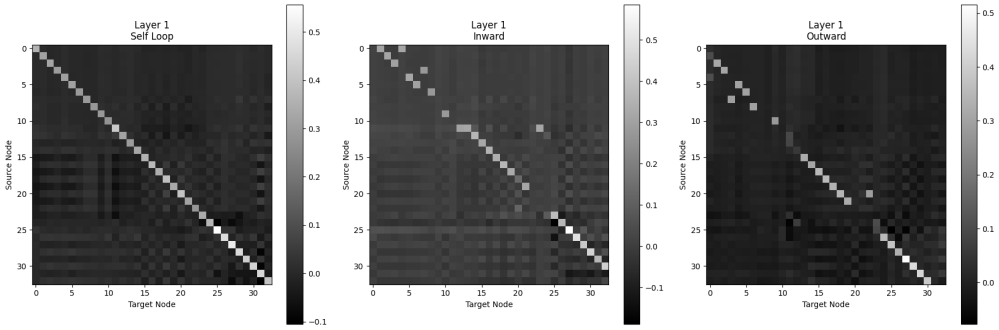


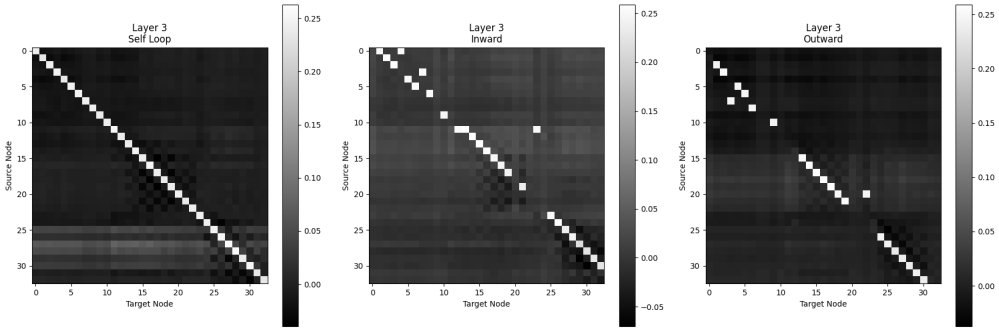
Figure 6.16: Confusion Matrix of AGCN-STC-MTCN model on test data.

	Loss	Accuracy (%)	Precision	Recall	F1 Score
Test Results	0.1420	97.12	0.9783	0.9800	0.9785

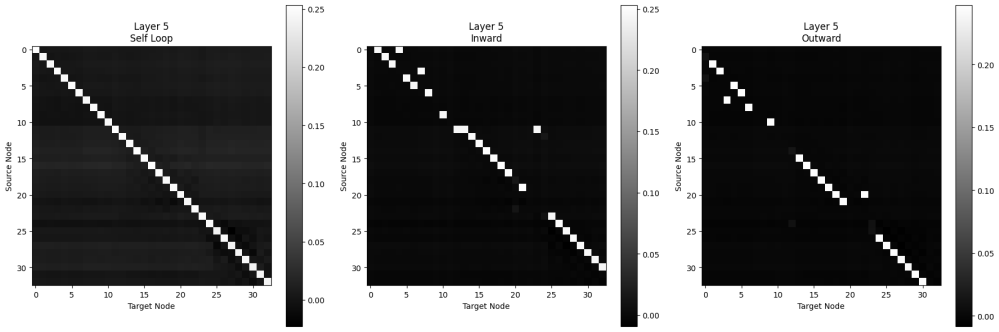
Table 6.8: Test results for AGCN-STC-MTCN model evaluation



(a) Learned adjacency matrices for layer 1



(b) Learned adjacency matrices for layer 3



(c) Learned adjacency matrices for layer 5

Figure 6.17: Learned adjacency matrices for different layers of the AGCN-MTCN model.

6.2 Discussion

6.2.1 1D CNN LSTM

As we can see from the test metrics for the 1D CNN LSTM model with and without attention pooling, the addition of attention pooling layer helped increase the accuracy by a slight amount but the loss increased by a significant amount which indicates that additional model complexity from the addition of attention pooling doesn't help generalize our model, this maybe because the input frames given to the model is only of sequence length 20 which isn't

a very long sequence so LSTM's hidden state output from the last time step can encode all the required information from previous time step inputs.

6.2.2 2D CNN LSTM

The primary challenge encountered during 2D CNN LSTM model training was dataset shrinkage due to link rot, which resulted in the unavailability of certain video samples. This further constrained an already limited dataset, reducing the diversity of training examples and making it more difficult for models to generalize effectively. To mitigate the risk of overfitting given the reduced dataset size, multiple 2D CNN LSTM architectures were evaluated, with a focus on selecting models that balanced complexity and generalization. Among the tested architectures, the ResNet-18 LSTM model demonstrated the best performance on test metrics. The ResNet-18 LSTM model, which had half the number of parameters compared to the DenseNet-121 LSTM model but the same LSTM hidden state size, achieved better results. VGG-16 LSTM model, which had half the LSTM hidden size compared to the ResNet-18 model, showed greater loss. This suggests that the primary limitation in the model performance is not so much related to CNN feature extraction as it is in the LSTM's ability to capture fine-grained temporal dependencies.

The confusion matrices, as shown in 6.6, 6.8, and 6.10 for the tested models, further highlight these limitations. The ResNet-18 LSTM model, which had the LSTM hidden state size of 64, struggled to differentiate poses with subtle movement differences, such as between Half-Way Lift and Standing Forward Bend, as well as between Extended Triangle and Extended Side Triangle. This indicates that the model's ability to learn nuanced motion patterns was insufficient, leading to frequent misclassifications. The restricted LSTM hidden state size likely reduced the model's capacity to retain crucial temporal dependencies, weakening its ability to distinguish between visually similar but contextually distinct actions.

Additionally, even with an aggressive dropout rate of 0.5 to combat overfitting, generalization remained a challenge. The small dataset size exacerbated this issue, limiting the model's capacity to effectively capture diverse movement patterns in yoga action classification.

2D CNN LSTM model proved to be suboptimal under constrained dataset size conditions. The results suggest that, with a limited dataset size, alternative architectures seem to be more effective in capturing subtle movement variations and improving generalization in yoga action recognition.

6.2.3 Adaptive ST-GCN

The observed test losses for both the STSAE-GCN and AGCN-STC-MTCN models indicate that the addition of STSAM or STC Attention module to the AGCN-MTCN model improves performance. Additionally, all adaptive ST-GCN models outperformed the 1D CNN-LSTM model. This highlights the efficacy of graph convolutional networks (GCNs) in human action recognition, particularly in capturing complex spatiotemporal dependencies within skeletal motion data, since the human skeleton can naturally be represented as a graph structure.

Furthermore, the learned adjacency matrix, as shown in 6.17, reveals information flow between nodes that are not physically connected. This adaptation extends beyond the initial adjacency matrix configuration, as shown in 3.2, highlighting the model's ability to dynamically capture relationships between physically disconnected but semantically related nodes within the skeletal graph structure.

6.3 System Performace Metrics

Table 6.9: Descriptive Statistics of Performance Metrics

Statistic	Metrics				
	Yoga Pose RTT	Text Feedback RTT	Voice Feedback RTT	Text Feedback Delay	Voice Feedback Delay
Mean	1.051	2.337	2.742	1.286	0.405
Std Dev	1.514	1.946	1.938	0.892	0.239
Minimum	0.079	0.858	0.525	-0.127	-0.512
Median	0.160	1.524	1.945	1.051	0.422
Maximum	5.550	7.938	7.976	4.768	0.680

Above lies a table illustrating the time performance metrics of various components used in this yoga classification and statistical coaching system. All the above round-trip time (RTT) metrics are measured from the time the last frame for the relevant yoga pose was sent from the client side, whereas the delays represent the closest assumed time taken solely by the particular stage. The various metrics used in the table are slightly described below:

1. **Yoga Pose Round Trip Time**: It represents the time taken to get the result of the yoga pose classification system at the client side. It is also taken as the delay time for pose classification.

2. **Text Feedback Round Trip Time**: It represents the time taken to generate and obtain the textual feedback back at the client side.

3. **Voice Feedback Round Trip Time**: It represents the time taken to convert the textual suggestion to sound and start playing it at the client side.

4. **Text Feedback Delay**: It is calculated as the difference between the RTT of text feedback and pose classification. This value is affected by the asynchronous nature of the underlying communication.

5. **Voice Feedback Delay**: It is calculated as the difference of the RTT of voice feedback and text classification. This value is also affected by the asynchronous nature of the underlying communication.

The data as obtained above was collected through a sample of 24 independent and some interdependent experiments over various related as well as unrelated videos. From the table above, it can be observed that on average, the whole pipeline takes about 2.742 seconds, which is actually an acceptable level of delay when a calming activity such as yoga is considered, in which the same poses can last for even longer at a single continuous time frame. Looking at the specifics, we can observe that the text feedback process takes the longest time of all in average, which is accounted by the fact that the text feedback generation took place outside of our local system through the use of an LLM API, and the effects of network transfer delays were obviously felt. Although on average, yoga pose classification doesn't take the longest time, sometimes the observed RTT of yoga pose classification exceeds even that of the text and voice feedback (causing some impossible metrics like the negative delays in the feedbacks), which can be attributed to some sudden instabilities in the underlying asynchronous proccessing of the message packets.

7. Conclusions

This project successfully developed an AI-based yoga assistance system capable of recognizing and assessing yoga postures using deep learning techniques. Its objective was to help bridge the gap between yoga practitioners and expert guidance by providing real-time posture analysis and corrective feedback.

Action classification was performed based on the 3DYoga90 dataset, achieving strong results on test metrics. Different architectures were tested for action classification, including 1D CNN LSTM, 1D CNN LSTM-Attention, AGCN-STC-MTCN, AGCN-MTCN, STSAE-GCN and 2D CNN LSTM (ResNet-18, DenseNet-121, and VGG-16). The AGCN-STC-MTCN model seemed to perform best based on the test metric, which highlights its capability in strengthening important spatial, temporal, and channel-wise features.

Additionally, pose phase classification was implemented to assist action classification and biomechanical feature extraction. This enabled joint angle analysis and allowed for a conditional rule-based approach to create prompt templates, which were processed by llama-3.1 8B for natural language feedback.

A web-based application was also developed, featuring pose-to-corrective speech generation and 3D human model visualization for self-evaluation, enhancing user experience.

While the project has successfully built a system that can be utilized to improve yoga practice experience through accessible guidance, challenges still remain, including dataset limitation to expand to more number of poses and computational constraints for higher throughput from the system. Future work can focus on refining the feedback mechanism, expanding datasets with more diverse samples, and optimizing the system to overcome performance bottlenecks.

8. Limitations and Future Enhancement

The following section outlines the possible further enhancements that can be done in relation to the yoga-assistance system that overcomes some limitations of the project, particularly regarding regards more robust corrective feedback generation system for the yoga poses.

1. **Improvements on the UI/UX design:**
   The user interface portion of this project has been completed just enough to present a strong case for the usability of the system proposed in this report, as well as an opportunity for the real-world testing of the developed system. Still, to be easy for common people to utilize outside of academic research purposes, user interaction and experience need to be developed to a level that can be regarded as more than just satisfactory.

2. **Usage of more robust body pose extractor**:
   Our current system utilizes MediaPipe as a crucial component to estimate the human body landmarks from the given input image frames. One can get even better performance in a project such as ours by utilizing a better and more computationally expensive system that generates human body pose information of greater quality.

3. **Incorporating a broader set of biomechanical feature for feedback generation**:
   The current methodology for determining the best way to perform the given pose relies on a simple mathematical formula based on the deviation from the ideal target angle between different body parts. This can be improved by utilizing other features, such as the separation distance between joints of the human body. Another approach would be to directly use an end-to-end pipeline for textual feedback generation using a 3D human body model for source pose and target pose as proposed in [32], removing the need to manually select and identify the most relevant biomechanical features for achieving a target pose.

4. **Integrating knowledge of experts in feedback**:
   Our feedback generation model uses a conditional rule-based approach to retrieve the ideal pose orientation based on the pose type. But for more accurate and robust yoga feedback generation, integrating domain knowledge of yoga experts would be more helpful for end users.

5. **Utilizing better hardware resources**:

   As demonstrated in the RTT metrics section, real-time deployment is feasible even with limited hardware. However, with a more powerful GPU backend, we could integrate additional features such as more human-like speech synthesis, which can be achieved with models like Kokoro-82M, and faster human mesh generation, as demonstrated in the SMPLer-X repository [33] (with a single Tesla V100 GPU achieving an average inference speed suitable for real-time mesh generation).

6. **Obtaining an actual deployment environment**:

   So far, all testing has been conducted on personal computers with limited processing power. A dedicated deployment environment with higher computational capacity could significantly reduce system strain, addressing some of the performance bottlenecks observed in the RTT metrics.

References

[1] Amira Samy Talaat. Novel deep learning models for yoga pose estimator. *SN Applied Sciences*, 5(12):341, 2023.

[2] Arun Kumar Rajendran and Sibi Chakkaravarthy Sethuraman. A survey on yogic posture recognition. *IEEE Access*, 11:11183–11223, 2023.

[3] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive body capture: 3D hands, face, and body from a single image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 10975–10985, 2019.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[6] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with multi-stream adaptive graph convolutional networks. *IEEE Transactions on Image Processing*, 29:9532–9545, 2020.

[7] Ujjwal Chowdhury. Yoga Pose Classification Dataset. `https://www.kaggle.com/datasets/ujjwalchowdhury/yoga-pose-classification`, 2020. Accessed: 2025-04-17.

[8] Guixiang Wei, Huijian Zhou, Liping Zhang, and Jianji Wang. Spatial–temporal self-attention enhanced graph convolutional networks for fitness yoga action recognition. *Sensors*, 23(10):4741, 2023.

[9] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition, 2020.

[10] Haodong Duan, Jiaqi Wang, Kai Chen, and Dahua Lin. Pyskl: Towards good practices for skeleton action recognition, 2022.

[11] Manisha Verma, Sudhakar Kumawat, Yuta Nakashima, and Shanmuganathan Raman. Yoga-82: a new dataset for fine-grained classification of human poses. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 1038–1039, 2020.

[12] Seonok Kim. 3dyoga90: A hierarchical video dataset for yoga pose understanding. *arXiv preprint arXiv:2310.10131*, 2023.

[13] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. Blazepose: On-device real-time body pose tracking, 2020.

[14] Santosh Kumar Yadav, Aayush Agarwal, Ashish Kumar, Kamlesh Tiwari, Hari Mohan Pandey, and Shaik Ali Akbar. Yognet: A two-stream network for realtime multiperson yoga action recognition and posture correction. *Knowledge-Based Systems*, 250:109097, 2022.

[15] Jing Chen, Jiping Wang, Qun Yuan, and Zhao Yang. Cnn-lstm model for recognizing video-recorded actions performed in a traditional chinese exercise. *IEEE Journal of Translational Engineering in Health and Medicine*, 11:351–359, 2023.

[16] Yubin Wu, Qianqian Lin, Mingrun Yang, Jing Liu, Jing Tian, Dev Kapil, and Laura Vanderbloemen. A computer vision-based yoga pose grading approach using contrastive skeleton feature representations. *Healthcare*, 10(1), 2022.

[17] Ben Alexander and Carlota Par´es-Morlans. Correcting fitness poses with monocular 3d human pose estimation from a single image.

[18] Najmul Hassan, Abu Saleh Musa Miah, and Jungpil Shin. A deep bidirectional lstm model enhanced by transfer-learning-based feature extraction for dynamic human activity recognition. *Applied Sciences*, 14(2), 2024.

[19] Santosh Yadav, Amitojdeep Singh, Abhishek Gupta, and Jagdish Raheja. Real-time yoga recognition using deep learning. *Neural Computing and Applications*, 31:https://link.springer.com/article/10.1007/s00521–019, 12 2019.

[20] James Wensel, Hayat Ullah, and Arslan Munir. Vit-ret: Vision and recurrent transformer neural networks for human activity recognition in videos. *IEEE Access*, 11:72227–72249, 2022.

[21] Ce Zheng, Wenhan Wu, Chen Chen, Taojiannan Yang, Sijie Zhu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. Deep learning-based human pose estimation: A survey. *ACM Computing Surveys*, 56(1):1–37, 2023.

[22] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Guixiang Wei, Huijian Zhou, Liping Zhang, and Jianji Wang. Spatial–temporal self-attention enhanced graph convolutional networks for fitness yoga action recognition. *Sensors*, 23(10), 2023.

[26] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. Disentangling and unifying graph convolutions for skeleton-based action recognition, 2020.

[27] Chu Xin, Seokhwan Kim, Yongjoo Cho, and Kyoung Shin Park. Enhancing human action recognition with 3d skeleton data: A comprehensive study of deep learning and data augmentation. *Electronics*, 13(4), 2024.

[28] HaoJie Ma, Wenzhong Li, Xiao Zhang, Songcheng Gao, and Sanglu Lu. Attnsense: Multi-level attention mechanism for multimodal human activity recognition. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3109–3115. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[29] M. Kumar, A. K. Patel, M. Biswas, et al. Attention-based bidirectional-long short-term memory for abnormal human activity detection. *Scientific Reports*, 13:14442, 2023.

[30] Mihai Fieraru, Mihai Zanfir, Silviu Cristian Pirlea, Vlad Olaru, and Cristian Sminchisescu. Aifit: Automatic 3d human-interpretable feedback models for fitness training. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9914–9923, 2021.

[31] Chongyang Wang, Yuan Feng, Lingxiao Zhong, Siyi Zhu, Chi Zhang, Siqi Zheng, Chen Liang, Yuntao Wang, Chengqi He, Chun Yu, and Yuanchun Shi. Ubiphysio: Support daily functioning, fitness, and rehabilitation with action understanding and feedback in natural language, 2024.

[32] Ginger Delmas, Philippe Weinzaepfel, Francesc Moreno-Noguer, and Grégory Rogez. Posefix: Correcting 3d human poses with natural language, 2024.

[33] Zhongang Cai, Wanqi Yin, Ailing Zeng, Chen Wei, Qingping Sun, Yanjun Wang, Hui En Pang, Haiyi Mei, Mingyuan Zhang, Lei Zhang, Chen Change Loy, Lei Yang, and Ziwei Liu. SMPLer-X: Scaling Up Expressive Human Pose and Shape Estimation. https://github.com/caizhongang/SMPLer-X#pretrained-models, 2025. Accessed: 2025-03-05.

APPENDIX A

Output and Inference

Some outputs obtained through this project are shown below.

These two were from internet sourced videos:

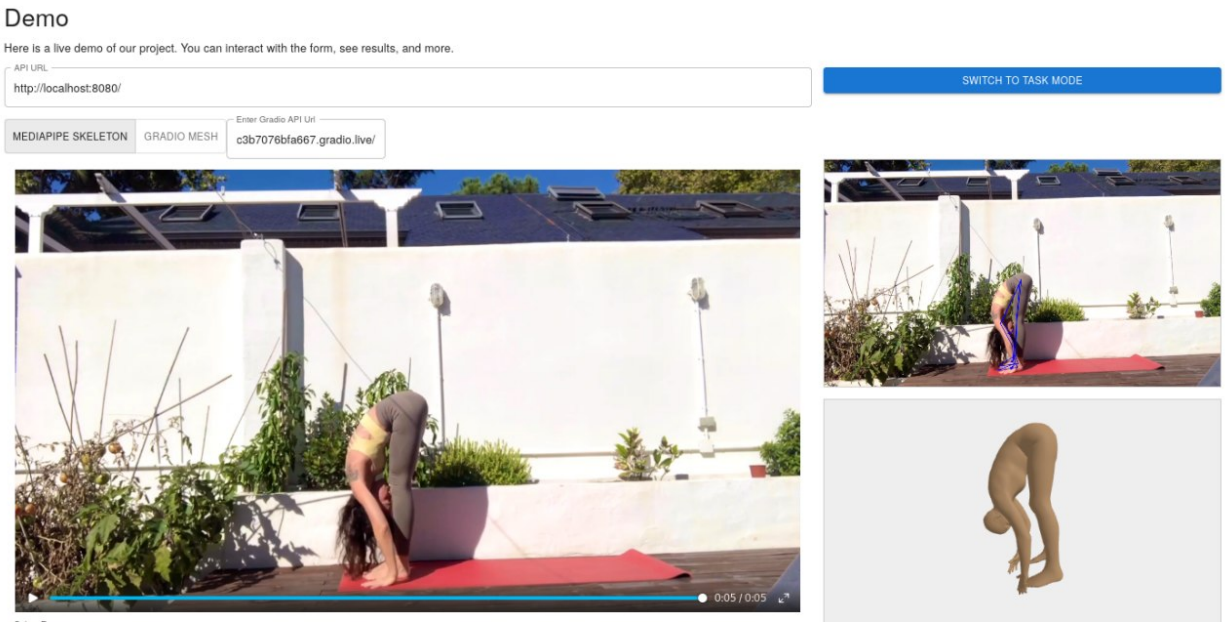


Figure 8.1: Streaming Output of a Standing Forward Bend Pose

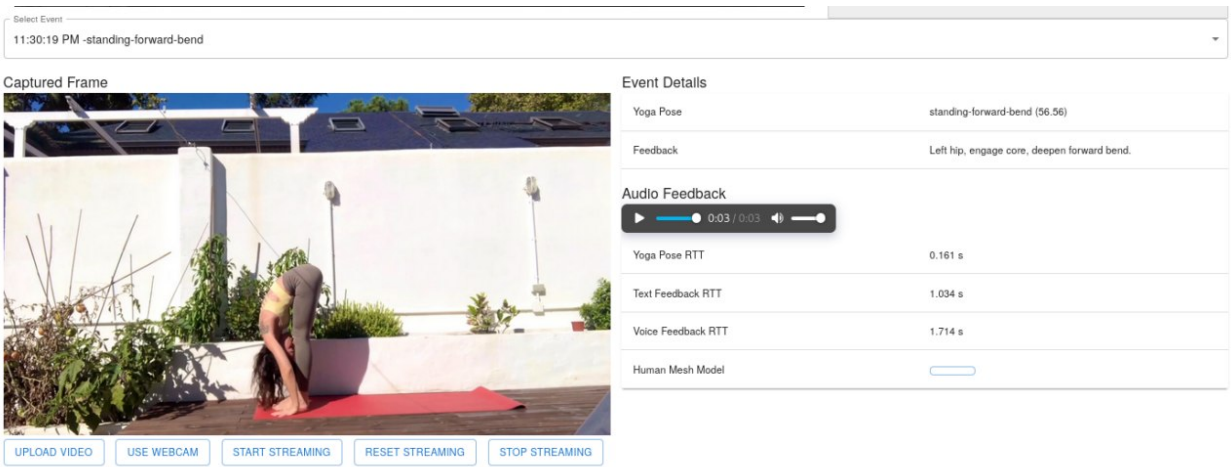


Figure 8.2: Detailed Output of Standing Forward Bend Pose

These two were from self performed yoga poses:

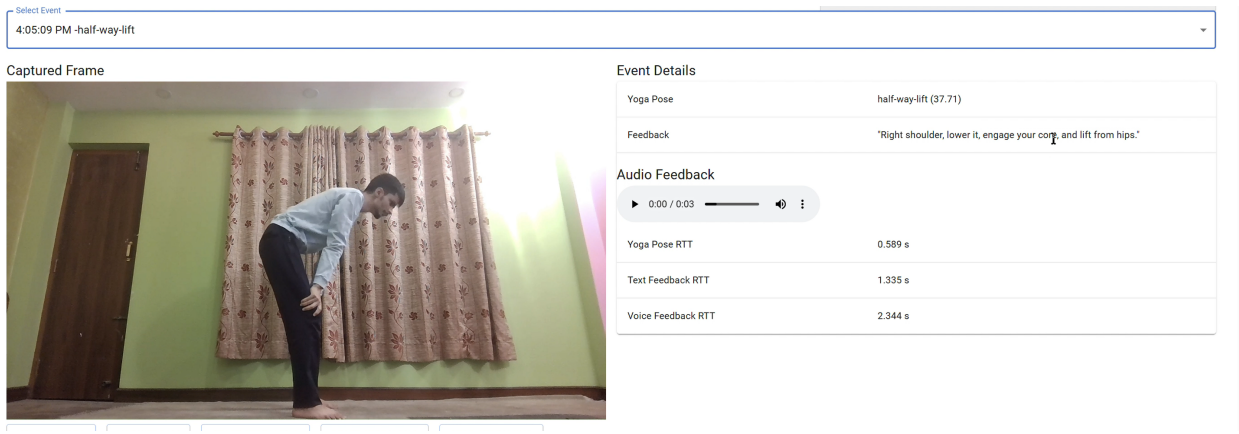


Figure 8.3: Sample Output of Half Way Lift

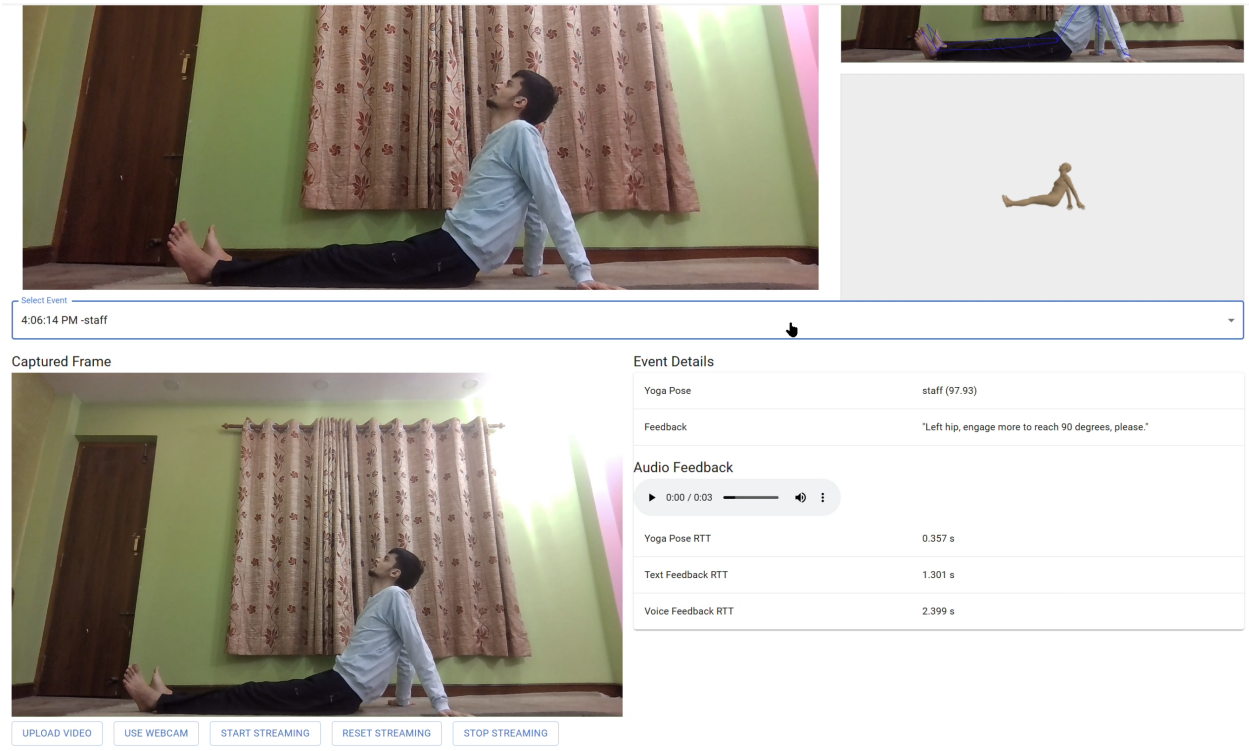


Figure 8.4: Sample Output of Staff Pose