KEC Conference 2026

ISSN: 3021-940X (Print) DOI: 10.3126/injet.v3i2.95537

ExerLiteNet: Lightweight CNN-LSTM Architecture for Binary Exercise Recognition from Webcam RGB Video

Ranish Ghimire[1*], Rojin Maharjan[2], Riya Bhattarai[3], Sahaj Shakya[4]

[1]*Computer and Electronics Department, Kantipur Engineering College, Lalitpur, Nepal, ghimireranish504@gmail.com*
[2]*Computer and Electronics Department, Kantipur Engineering College, Lalitpur, Nepal, rojin.maharjan44@gmail.com*
[3]*Computer and Electronics Department, Kantipur Engineering College, Lalitpur, Nepal, wreeyab123@gmail.com*
[4]*Computer and Electronics Department, Kantipur Engineering College, Lalitpur, Nepal, sahaj@kec.edu.np*

Abstract

Exercise recognition from video is important for building digital workout tracking systems and automated fitness monitoring tools that can provide coaching without the need for expensive equipment. This paper presents ExerLiteNet, a lightweight deep learning model designed for resource-constrained devices, which uses standard RGB webcam video to classify two common resistance exercises, squats and bicep curls. The proposed model combines a fine-tuned MobileNetV3Small CNN wrapped in a Time Distributed architecture to extract spatial features from each video frame, and a Long Short-Term Memory (LSTM) network to capture the motion patterns across a sequence of 15 frames. The training data consisted of 299 videos across two exercise classes, each averaging 10 seconds in length, collected from both stock video sources and webcam recordings. The model achieved a classification accuracy of 92.08% on the test dataset after fine-tuning, outperforming a frozen ResNet50 baseline (82.31%) and an intermediate frozen MobileNetV3Small + LSTM configuration (87.08%). In terms of computational efficiency, the proposed model has a total size of approximately 178 MB and requires only 1.8 GFLOPs per inference sequence, which is significantly lower than VGG16+LSTM at approximately 553 MB and 15.43 GFLOPs, and ResNet50 at approximately 3.8 GFLOPs with no temporal modeling capability. The model also runs at approximately 15 frames per second with an end-to-end inference latency of 2.3 seconds per sequence on a standard webcam setup. These results show that combining a lightweight convolutional architecture with sequence modeling can achieve competitive accuracy while remaining practical for deployment on everyday hardware without any specialized sensors or depth cameras.

Keywords: Exercise Recognition, MobileNetV3Small, CNN-LSTM, Spatiotemporal Learning, Transfer Learning, RGB Video, Lightweight Deep Learning, Fine-tuning, Overfit

1. Introduction

Practitioners of physical fitness need to train properly and accurately assess their progress; however, many individuals lack the guidance to do so. Thus, the critical issue is how to analyze and interpret continuous motion from a video feed. Therefore, the primary goal is to develop algorithms that can process the instantaneous pose and the evolving pattern of motion [1]. Both vision-based methods and hardware-based methods have been used to solve this problem. Vision-based methods are limited to processing individual image frames; therefore, they cannot model temporal dynamics, which are essential for recognizing activities defined by transitional movement phases (i.e., how one moves from point A to point B). Sequential models such as RNNs can process image sequences but lack the spatial feature extraction ability of CNNs, making them insufficient on their own. Additionally, hardware-based methods use either attached sensors or special cameras to measure movement. These methods provide high accuracy in measuring movement, but they require equipment that is typically not found in a home environment.

The major contribution of this research is developing a new approach that combines the advantages of both visions-based (CNN + LSTM) methods and hardware (2D webcam for input) based methods to overcome some of the limitations of each method. In addition, the developed algorithm uses a mobile architecture that is designed for limited computer resources. The architecture uses MobileNetV3Small for per-frame spatial analysis, passing its outputs to an LSTM pipeline that analyzes the movement patterns over time. This approach has successfully demonstrated the ability to categorize exercises using common webcams, while

* *Corresponding author*

also providing a practical solution for embedded platforms, thus bridging the gap between accuracy and practicality.

2. Related Work

2.1 Vision Based Systems

Yadav et al. proposed YogNet, a two-stream deep spatiotemporal neural network for real-time yoga action recognition. The architecture integrates a pose stream, where 25 body keypoints from OpenPose are processed through TimeDistributed CNNs and regularized LSTMs to capture temporal patterns [2] to capture temporal patterns, alongside an RGB stream that uses modified C3D 3D-CNNs for spatiotemporal feature extraction. Both streams are combined using feature-level and decision-level fusion to enhance recognition performances. To handle multiple individuals, pseudo-tracking based on left-to-right bounding box sorting is applied. Evaluated on the YAR dataset with 1206 videos from 16 participants performing 20 asanas, the model achieved 96.31% accuracy, outperforming previous approaches that reported 75% and 85.47%.

Bang et al. created a Convolutional Neural Network (CNN) ensemble model that uses joint coordinates and angles obtained from the MediaPipe framework to classify ten home workout exercises in real time [3]. The framework processes video frames to find twelve hinge angles based on 66 landmarks that will then be used as inputs into a 1D CNN consisting of several convolution and max-pooling layers followed by densely connected layers. The top prediction is determined by applying a soft-voting ensemble across all 30 frames processed. The authors evaluated their model against the InfinityRep dataset and achieved a classification accuracy of 92.1% which greatly exceeded the classifications from other models such as Random Forest (72.68%), LSTM (79.5%), and CNN + LSTM (79.78%). A follow-up study was conducted with real-world validation of the model using videos of individuals performing exercises with an accuracy of 92.63%.

Ashraf et al. proposed a new CNN architecture named YoNet [4]. This model is designed to classify photographs of five different yoga postures in the Yoga-82 dataset (286 photographs) using primarily depth-wise separable convolutions along with standard convolutions as spatial features up until before the dense layer (layers after which global average pooling occurs). YoNet's accuracy rate of 94.91% was higher than ResNet50's accuracy rate (91.52%), InceptionV3's accuracy rate (86.44%), and Xception's accuracy rate (89.83%). The statistical analysis of the results indicated the significant difference between each of their accuracies. The research only included five static yoga postures rather than dynamic movements. Additionally, the study did not compare its results against pose estimation models such as MediaPipe.

Simonyan and Zisserman introduced VGGNet, scaling convolutional networks to 19 layers by stacking uniform 3x3 convolutions with stride 1 and padding to maintain spatial resolution, interspersed with 2x2 max-pooling, followed by three fully-connected layers and softmax [4]. By eliminating Local Response Normalization and using only max-pooling with small filters, their approach achieved 7.1% top-5 error on ImageNet, securing 1st in localization and 2nd in classification at ILSVRC-2014 according to Simonyan and Zisserman. VGG standardized modern CNN design, and its pretrained weights became foundational for transfer learning in applications like yoga pose recognition. With high memory demands exceeding 500MB and GPU-intensive training, they later used ResNet for optimizations.

A two-stream network combining pose keypoints with 2D CNNs and LSTMs for capturing temporal dependencies alongside RGB frames achieved high accuracy on yoga recognition.

2.2 Sensor Based Systems

Wu et al. introduced an IMU-based system using 11 sensors across the body to recognize and quantitatively evaluate 18 static yoga postures [1]. Their two-stage classifier combines Back Propagation-ANN for categorizing poses into 5 groups and Fuzzy C-Means for precise identification, achieving 95.39% instance accuracy. Bayesian networks model quaternion data as Gaussian variables to provide corrective feedback based on deviation from standard postures. This approach effectively resolves prior limitations, including

RGB camera occlusions, lighting issues, Electromyogram noise and limited posture coverage in previous methods However, the system requires 11 wearable sensors that may restrict natural movement, and it addresses only static poses without modeling dynamic transitions between asanas.

Chen et al. introduced a Kinect-based system for recognizing 10 yoga postures from 5 subjects using depth camera data [**6**]. They implemented a two-phase method where Dynamic Time Warping (DTW) first classifies postures via joint angle sequences, followed by Support Vector Machines (SVM) for refined matching with geometric features like inter-joint distances. While DTW alone achieves 87% accuracy and SVM achieves 89%, the combining of both DTW and SVM achieved 92.5% accuracy with real-time processing under 50ms per frame. This approach allows for accessible self-training without the need for guidance by addressing vision-based issues like lighting fluctuations and occlusions through depth sensing. However, hardware dependency on Kinect sensors limits home usability and may suffer from sensor drift. The small dataset restricts generalization and diversity in body types. Feature extraction using only joint angles and distances misses subtle misalignments like spinal curvatures and dynamic transitions between poses, which might remain unaddressed.

The major limitations of existing yoga pose recognition systems include inconvenient wearable sensors that restrict user movement and dependency on specialized depth-camera hardware, many systems struggle to capture smooth transitions between poses and fail to detect subtle postural misalignments, reducing their effectiveness in real-world practice.

2.3 Research Gaps

Current exercise recognition technologies have limitations. Traditional ML-based techniques rely on manual feature extraction, resulting in high computational complexity and limited generalization. Sensor-based methods using wearable sensors or using Kinect camera achieve high accuracy but require specialized hardware that is impractical for home use. CNNs are able to model spatial features but lack temporal modeling ability, which is critical for properly recognizing exercises. There is a need for architectures which can learn both spatial and temporal characteristics directly from RGB video input without requiring the use of specialized hardware and additional complexity.

3. Methodology

3.1 Data Collection and Preparation

Frames for squats and biceps curl were collected from stock videos on iStock and Pexels, yielding about 7,500 frames per class. This was supplemented by 4,590 webcam-recorded frames (10-second clips) per class. This resulted in approximately 12,090 frames per exercise, totaling over 24,180 frames. Each frame has been resized to 224×224×3 and then chunked into sequences of 15 frames each to make the data suitable for sequential processing. This produced 1,612 sequences per exercise, 806 from bicep curls and 806 from squats, resulting in a model input shape of (15, 224, 224, 3) to capture both temporal and spatial details. Data augmentation techniques such as horizontal flipping, random brightness and contrast adjustments, Gaussian blur, and motion blur were applied during training to improve generalization. All pixel values were normalized, and frames were grouped into sequences using a length of 15 frames. The dataset was randomly split into 70% training, 15% testing, and 15% validation sets to ensure a balanced distribution.

Figure 1. Clustering of Frames

3.2 Proposed System

The following diagram depicts the two main pipelines of the proposed system, beginning with the training pipeline. In the initial phase, the model was trained on the training set while the validation set was used to monitor performance. After identifying the optimal model, fine tuning was carried out using the same training and validation sets to improve feature representation and enhance classification performance. The fine-tuned model was then evaluated using the same testing set. Based on this evaluation, the final trained model was obtained and prepared for deployment in real-world scenarios. The second portion of the diagram shows the inference pipeline, where a video feed was provided as input to the system. The input video underwent preprocessing steps, including frame extraction, resizing, and normalization to ensure consistency with the training data. The processed frames were then organized into fixed length sequences to capture temporal motion information across consecutive frames. These sequences were passed into the trained classifier, where spatial features were extracted, and temporal dependencies were analyzed. The classifier then performed prediction based on learned patterns and produced the detected exercise as output, such as squat or bicep curl.

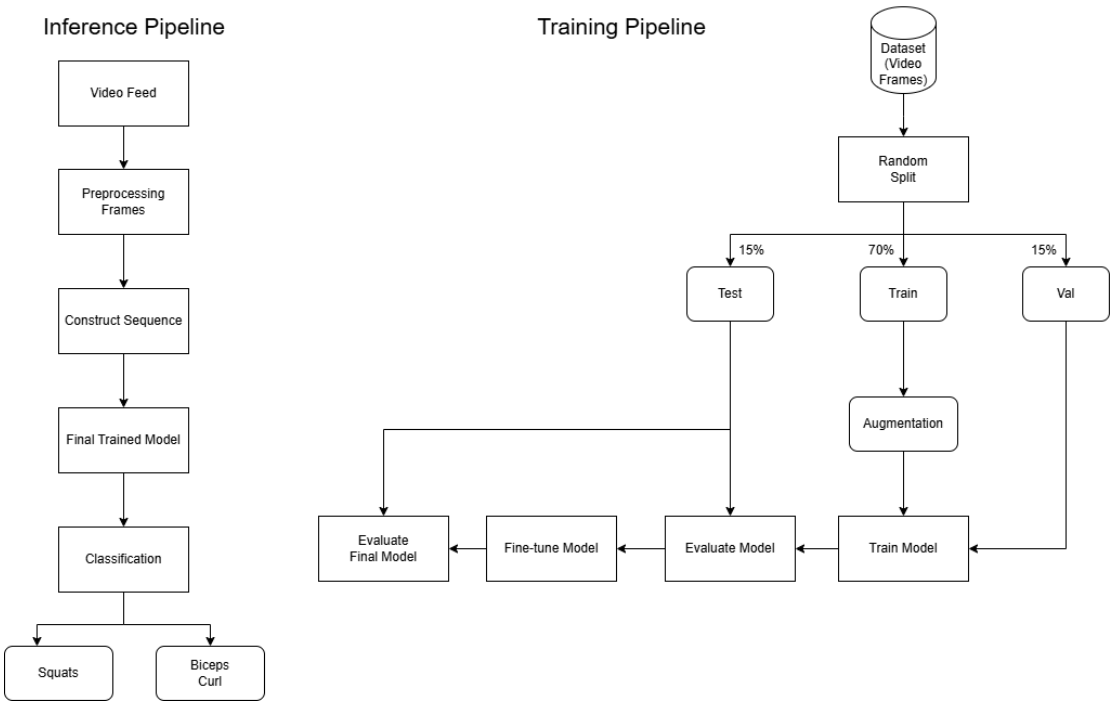

Figure 2. Training and Inference of System

3.3 Model Architecture

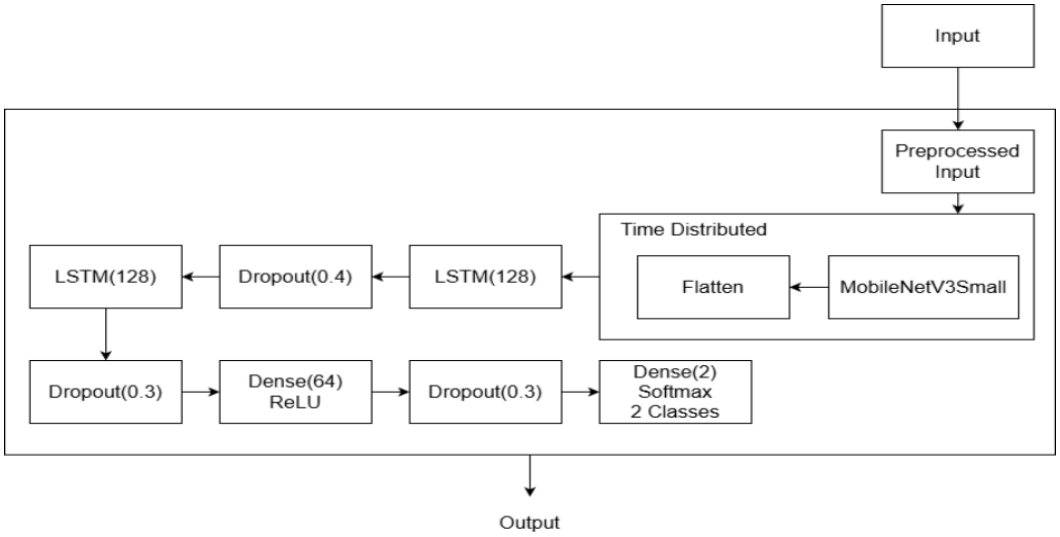

Figure 3. Model Architecture of System

Figure 3 illustrates the architecture of the proposed CNN–LSTM based exercise recognition system. The model takes processed sequences of RGB frames extracted from workout videos as input. Each frame was then processed separately using a lightweight backbone (MobileNetV3Small) to extract spatial features. Those frame-level features were then passed into the stacked LSTM layers to model the motion dynamics between frames. The classification head then predicted which category of exercise was being performed. The proposed model based on CNN, combined with LSTM, has been designed to extract both spatial and temporal information from video data of people performing two classes of physical exercise (biceps curl and squats). The input data consisted of 15 sequential RGB frames per sample; each frame resized to 224×224×3 and each pixel values normalized to [0, 1] before feeding it into the model. So, the input shape to the model was (15, 224, 224, 3).

MobileNetV3Small, a pre-trained CNN on ImageNet dataset, was used to extract the spatial features from each frame. In order to apply this same convolutional feature extractor to each frame in the sequence independently, the network was wrapped inside a TimeDistributed layer, which enabled the model to capture individual frame level visual cues (such as posture of the body, angle of joints, position of arms and legs) while preserving the temporal ordering of the sequence. The output shape after this layer became (15, 7, 7, 576). The feature maps resulting from the application of this convolutional feature extractor to each frame were flattened into a sequence of feature vectors of shape (15, 28224).

To model the dynamics of the motion across consecutive frames, the sequence of feature vectors (15, 28224) was passed through two stacked LSTM layers, each containing 128 units. The number of units in the LSTM layers was chosen to provide a balance between sufficient capacity to capture temporal dependencies within the 15-frame sequences and keeping computational complexity low. These layers enabled the model to learn motion patterns such as the upward and downward movement of the arms during a biceps curl or the flexion and extension of the legs during a squat. Dropout layers with rates of 0.4 and 0.3 were added right after an LSTM block to increase the model's ability to generalize new examples since the dataset size is relatively limited. These dropout rates were selected so that the model does not rely excessively upon specific frame patterns while at the same time encouraged the model to learn more robust temporal features. The first LSTM block outputs a sequence of shape (15, 128), which is fed into the second LSTM block. The second LSTM block returns a one-dimensional 128-unit feature vector, which is then passed to the dense layer.

The learned temporal representation is then fed into a fully connected dense layer with 64 neurons with ReLU activation that relates the spatial-temporal features extracted by the CNN-LSTM pipeline to learn higher-level discriminative patterns from the motion features that will enable the model to distinguish

the two exercises. Finally, the last two neurons of the Softmax output layer return the probabilities of the two types of exercises.

The model was then adjusted to fit the specific domain, so the last 20 layers of MobileNetV3Small were unfrozen and fine-tuned at a reduced learning rate than what was previously used to allow the model to learn the domain-specific patterns while still preserving the broad visual representations learned from ImageNet.

Overall, the CNN–LSTM architecture proposed in this study allows for both (spatial) visual information from exercise videos and (temporal) motion dynamics to be learned together. MobileNetV3Small provides a lightweight backbone for extracting visual features at the frame level. With the use of LSTMs to capture motion dynamics across time, this design allows for accurate classification of exercises using only standard webcam video input.

The proposed CNN-LSTM architecture can be seen in detail in the Figure 3 that comes directly from the model.summary() output. All layers are listed in order of occurrence, beginning with the input layer that takes in a series of frames from a video. In addition, the model utilizes a TimeDistributed wrapper around a convolutional backbone for extracting the spatial features of each video frame. Once extracted, these spatial features will be reshaped and passed on to the LSTM layers supplying the temporal dependencies within the sequence. Dropout is utilized after each LSTM and Dense layer as an effort to decrease the likelihood of overfitting and improve generalization. Dense layers will also further process the extracted features for classification purposes. Finally, the output of the model for the input sequence will be a predicted class from the last layer of the network. Each of these elements, thus, illustrates the output shape at each stage of development by demonstrating how the data is altered (i.e., dimensions change) throughout the entire model. In addition, the number of parameters contained within each of the layers will illustrate the degree of complexity and contribution level each component contributes to the final model.

Table 1: Hyperparameters used

Hyperparameters	**Value**
Input Shape	(15,224,224,3)
Sequence Length	15 frames
CNN backbone	MobileNetV3Small
Frozen Layers (pre-fine-tuning)	Frozen Layers (pre-fine-tuning)
Unfrozen Layers (fine-tuning)	Last 20 layers of backbone
Initial Learning Rate	0.0001
Fine-tuning Learning Rate	0.00001
LSTM units (each layer)	128
Dense Layer	64 neurons, ReLU
Dropout rates	0.4 (after LSTM 1), 0.3 (after LSTM 2 and Dense)
Optimizer	Adam
Loss function	Categorical Cross-Entropy
Batch size	8
Early stopping rate	5 epochs

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 15, 224, 224, 3)	0
time_distributed (TimeDistributed)	(None, 15, 7, 7, 576)	939,120
time_distributed_1 (TimeDistributed)	(None, 15, 28224)	0
lstm (LSTM)	(None, 15, 128)	14,516,736
dropout (Dropout)	(None, 15, 128)	0
lstm_1 (LSTM)	(None, 128)	131,584
dropout_1 (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130

Figure 4. Architecture Summary of the Proposed Model

Table 2: Parameter Distribution before and after Fine-tuning

Parameters	Before Unfreezing	After Unfreezing
Trainable parameters	14,656,706	15,007,442
Non-trainable Parameters	939,120	588,384
Total Parameters	15,595,826	15,595,826

3.4 Convolutional Neural Networks (CNN)

CNNs are deep learning models designed to process image data by automatically learning spatial features such as edges, textures, and shapes [7].

3.4.1 Convolutional Layer

This layer applies filters knowns as kernels containing learnable parameters over the input image to extract spatial features.

$$\frac{(n-f+2p)}{s} + 1 \qquad \text{(Equation 1)}$$

Where n represents input size, f represents kernel size, p represents padding and s represents strides.

3.4.2 Pooling Layer

This layer reduces the spatial dimensions of feature maps by summarizing regions (e.g., max or average values). It helps reduce computation and improves robustness to small spatial variations. It is a non-trainable layer.

3.4.3 Flatten Layer

This layer converts multidimensional feature maps into a one-dimensional vector so that they can be processed by fully connected layers.

3.4.4 Fully Connected Layer

This layer connects all extracted features to neurons and learns high-level representations used for final prediction. It is a trainable layer with weights and biases [7].

3.4.5 SoftMax Layer

This layer converts the output values of the network into probabilities for each class. The output values range between 0 and 1 and their sum equals 1.

$$[\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}] \quad \text{(Equation 2)}$$

where z is the input vector and j represents the class index.

Convolution Operation:

$$conv(I, K) = I * rot^{180}(K) \quad \text{(Equation 3)}$$

Where,
I: Input image or feature map,
K: Kernel/filter matrix,
$rot^{180}(K)$: Kernel rotated by 180° (used in mathematical convolution),
$*$: Convolution operation

Forward Propagation:

$$Y_i = B_i + \sum_{j=1}^{n} X_j * k_{ij} (i = 1, \dots, d) \quad \text{(Equation 4)}$$

Where Y_i represents output feature map at channel i, B_i represents bias term for the ith output channel, X_j represents input feature map at channel j, k_{ij} represent kernel connecting input channel j to output channel i, n represents number of input channels and d represents number of output channels.

Backward Propagation:

$$\frac{\partial E}{\partial K_{i}} = X * \frac{\partial E}{\partial Y_i} \quad \text{(Equation 5)}$$

$$\frac{\partial E}{\partial B_i} = \frac{\partial E}{\partial Y_i} \quad \text{(Equation 6)}$$

$$\frac{\partial E}{\partial X} = \sum_{i=1}^{n} \frac{\partial E}{\partial Y_i} * K_i \quad \text{(Equation 7)}$$

Where,
E: Loss function,
$\frac{\partial E}{\partial Y_i}$: Gradient of loss relative to output feature map,
$\frac{\partial E}{\partial K_i}$: Gradient relative to kernel weights,
$\frac{\partial E}{\partial B_i}$: Gradient relative to bias,
$\frac{\partial E}{\partial X}$: Gradient relative to input feature map.

3.5 MobileNetV3Small

MobileNetV3 Small is a lightweight convolutional neural network (CNN) optimized for speed and efficiency, especially for mobile devices and embedded systems [8].

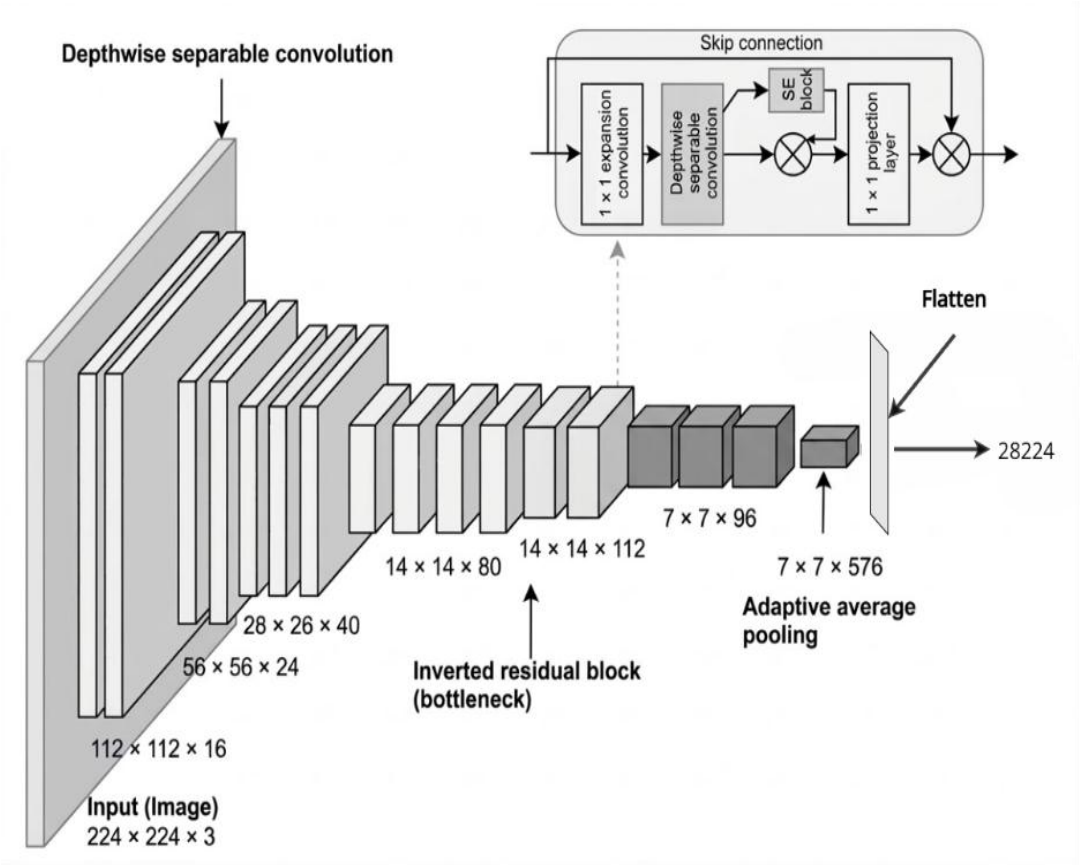

Figure 5. MobileNetV3Small

3.5.1 Key Features of MobileNetV3Small

MobileNetV3Small uses depth-wise separable convolutions to reduce computational cost by separating spatial and channel-wise filtering. It also employs inverted residual blocks, which improve efficiency by expanding and compressing feature representations. Additionally, Squeeze-and-Excitation (SE) layers help the network emphasize important feature channels while suppressing less useful ones [**9**].

3.6 Computational Cost Comparison

3.6.1 Normal Convolution

Computational Cost = #filter params×#filter positions×#filters

$$= (3 \times 3 \times 3) \times (222 \times 222) \times 5 = 6653340$$

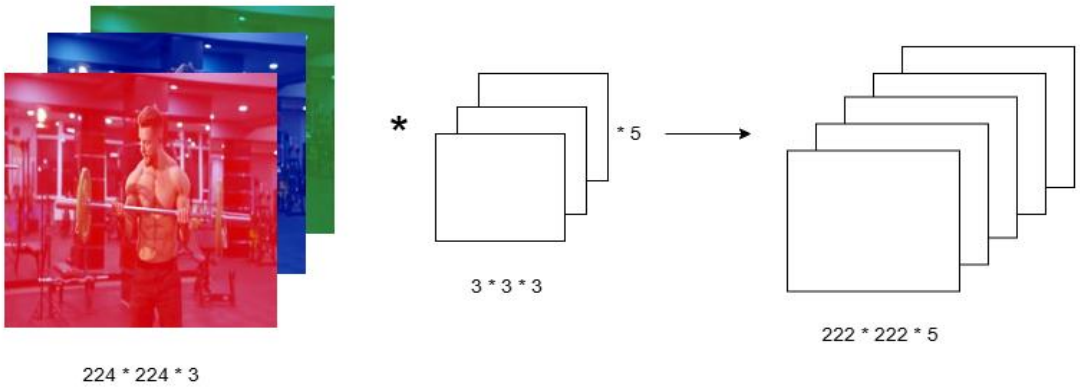

Figure 6. Normal Convolution

3.6.2 Depth-wise Convolution

Computational Cost = $(3 \times 3) \times (222 \times 222) \times 3 = 1330668$

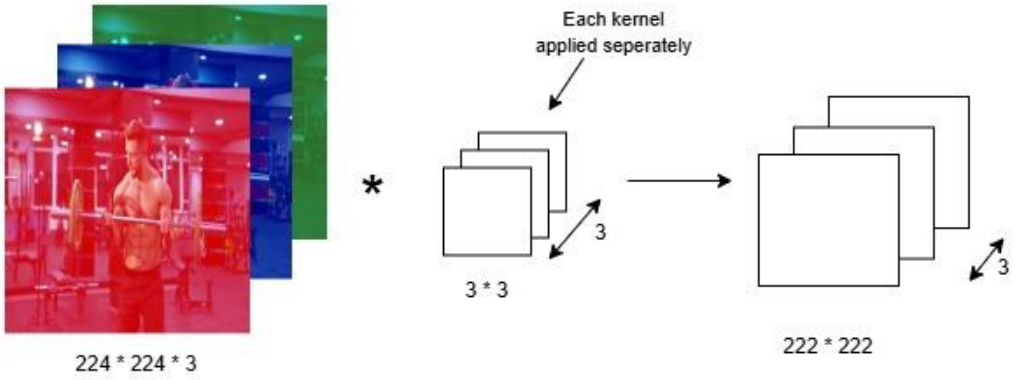

Figure 7. Depth-wise Convolution

3.6.3 Point-wise Convolution

Computational Cost = $(1 \times 1 \times 3) \times (222 \times 222) \times 5 = 739260$

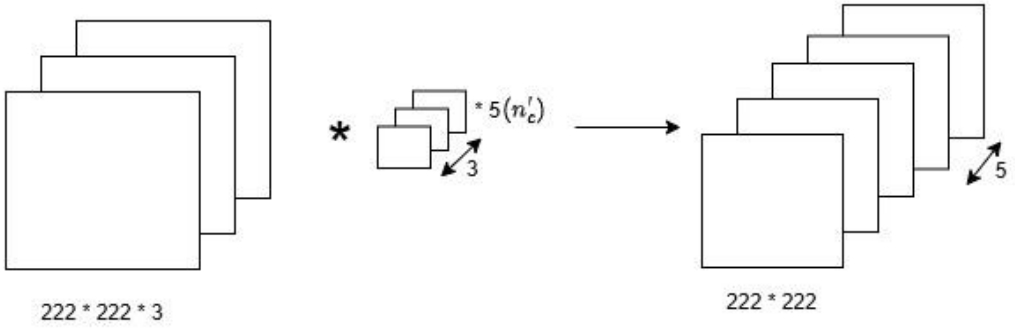

Figure 8. Point-wise Convolution

3.6.4 Total Computational Cost

Computational cost of Depth-wise + Computational cost of Pointwise = $1330668 + 739260 = 2069928$

$$\text{Cost Reduction: } = 1 - \frac{Computational\ Cost\ Of\ Depthwise\ Seperable}{Computational\ Cost\ Of\ Normal\ Convolution} = 1 - \frac{2069928}{6653340} = 0.689$$

Ratio of cost of Depth-wise-Separable to Normal Convolution = $\left(\frac{1}{n_c'}\right) + \left(\frac{1}{f^2}\right) = \left(\frac{1}{5}\right) + \left(\frac{1}{3^2}\right) = 0.311$

The cost reduction ratio 0.311 means that the depth-wise separable convolution requires only 31.1% of the computational cost compared to a normal convolution.

3.7 Hard Swish Activation Function

It is a computationally efficient activation function. The function helps in smoothly transitioning from negative to positive values and provides a non-linear activation effect which helps improve the stability of training and performance It is also less computationally intensive than previous types of activation functions such as Swish or GELU [**8**].

$$\text{Hard Swish(z): } = z * \big(ReLU\big(6(z+3)\big)/6\big) \qquad \text{(Equation 8)}$$

$$\text{Where, Relu6} = \min(\max(0, x), 6) \quad \text{(Equation 9)}$$

$$\text{HardSwish(x)} = \begin{cases} 0 & if\ x \leq -3 \\ x & if\ x \geq 3 \\ \frac{x(x+3)}{6} & otherwise \end{cases} \quad \text{(Equation 10)}$$

3.8 Long Short-Term Memory (LSTM)

LSTMs are designed for processing sequential data. The architecture addresses the vanishing gradient problem through memory cells and gating mechanisms. Memory cells remember previous values and pass the latest value through the gates to allow for both short-term and long-term dependencies to be learned from the sequential data.

3.8.1 LSTM Gates

Forget Gate: Decides what information to discard from the memory cell.

$$\mathrm{f_t} = \sigma\left(w_f \cdot [h_{t-1}, x_t] + b_f\right) \quad \text{(Equation 11)}$$

$$c_t = f_t \odot c_{t-1} \quad \text{(Equation 12)}$$

Where,

$\mathrm{f_t}$: Forget gate vector (controls what information to discard from c_{t-1}),

h_{t-1}: Hidden state from the previous time step (short-term memory),

c_{t-1}: Previous cell state (long-term memory),

c_t: Updated cell state at time step t,

x_t: Input vector at time step t (current frame/feature input),

b_f: Bias vector,

w_f: Weight matrices for forget gate,

σ: Sigmoid activation function (outputs values between 0 and 1, acting as a gate)

⊙: Element-wise (Hadamard) multiplication

Input Gate: Determines what new information to store in the memory cell.

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(Equation 13)}$$

$$\tilde{c}_t = tanh(w_c \cdot [h_{t-1}, x_t] + b_c) \quad \text{(Equation 14)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{(Equation 15)}$$

Where,

i_t: Input gate vector,

w_i: Weight matrices for input gate,

b_i, b_c: Bias vectors,

$\tilde{c}_t$: Candidate cell state (new potential information to be added),

tanh: Hyperbolic tangent activation (outputs values between -1 and 1),

w_c: Weight matrices for candidate gate,

Output Gate: Controls the output based on the memory and the current input.

$$o_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(Equation 16)}$$

Where,

o_t: Output gate vector (controls what part of the cell state is output),

w_o: Weight matrices for output gate,

b_o: Bias vector,

$[h_{t-1}, x_t]$: Concatenation of previous hidden state and current input.

Cell State Update:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{(Equation 17)}$$

$$h_t = o_t \odot tanh(c_t) \quad \text{(Equation 18)}$$

Where c_t represents updated cell state at time step t, c_{t-1} represents previous cell state (long-term memory), f_t represents forget gate vector (controls what information to discard from c_{t-1}), i_t represents input gate vector, $\tilde{c}_t$ represents candidate cell state (new potential information to be added), h_t represents hidden state from the current timestamp, o_t represents output gate vector (controls what part of the cell state is output) and tanh represents hyperbolic tangent activation (outputs values between -1 and 1).

4. Results and Analysis

4.1 Results

The proposed CNN–LSTM model was trained on frames sourced from online exercise videos and from gym sessions recorded via webcam. This method This approach provides a more realistic representation of exercise performance in real-world conditions. The spatial extractor of the model is MobileNetV3Small trained on ImageNet. This backbone network was originally used as a fixed feature extractor while all of the remaining layers were set to trainable with a learning rate of 0.0001. Once the model converged on the dataset, the last 20 layers of MobileNetV3Small were then fine-tuned with an even lower learning rate of 0.00001 to allow the model to adjust to specific execution patterns in exercise visuals. Training continued until convergence, with early stopping applied to prevent overfitting.

Table 3: Model Performance Metrics

Model	Accuracy	Precision	Recall	F1-score	Best Epoch	Description
Baseline: Frozen ResNet50 + Dense (128+64+2)	82.31%	85.60%	78.40%	81.84%	11	No Temporal Modeling
Frozen: MobileNetV3small + LSTM +Dense (64+2)	87.08%	93.34%	80.32%	86.34%	38	CNN frozen, LSTM trained
Proposed: Fine-Tuned MobileNetV3small + LSTM +Dense (64+2)	92.08%	96.39%	87.70%	91.83%	12	CNN unfrozen (20 layers), LSTM trained

The inclusion of a baseline model further highlights the effectiveness of the proposed approach. While the frozen ResNet50 baseline achieved an accuracy of 82.31% with no temporal modeling, its performance is limited due to the absence of sequence learning and domain-specific feature adaptation. To address this, an intermediate model using MobileNetV3Small with frozen CNN layers and an LSTM was introduced, which improved performance to 87.08% accuracy by incorporating temporal modeling while keeping spatial features fixed.

However, the proposed model goes a step further by unfreezing and fine-tuning the CNN layers along with LSTM training. This allows the network to adapt spatial feature extraction to the specific dataset while simultaneously learning temporal dependencies. As a result, the proposed MobileNetV3Small + LSTM model achieves a significantly higher accuracy of 92.08% and an F1-score of 91.83%. This progression clearly demonstrates that while adding temporal modeling improves performance, jointly fine-tuning both spatial and temporal components leads to the best results by capturing complex domain-specific patterns more effectively.

Initially, the model was trained for 41 epochs. During this time, the training losses and validation losses decreased from an approximate starting value of 0.86, indicating that the model was able to learn useful and meaningful patterns from the exercise data. Loss values during epochs 32–38 were stable, indicating that the model had almost reached convergence. As the training continued past epoch 38, the training loss continued to decrease, but the validation loss began increasing, indicating that the model had begun to memorize the training samples instead of generalizing to new examples, this condition is termed as "overfitting". To further improve the performance of the model, MobileNetV3Small's last 20 layers were unfrozen and fine-tuned at a lower learning rate. During this fine-tuning phase, an additional 14 epochs of training were conducted. During this phase, initially, the model exhibited an increase in training loss due to the new trainable parameters, peaking at approximately 0.43 for training loss and 0.35 for validation loss around epoch 4. As the fine-tuning optimizations progressed, both losses declined steadily, with the model performance improving for both training and validation data. The best performance was recorded between

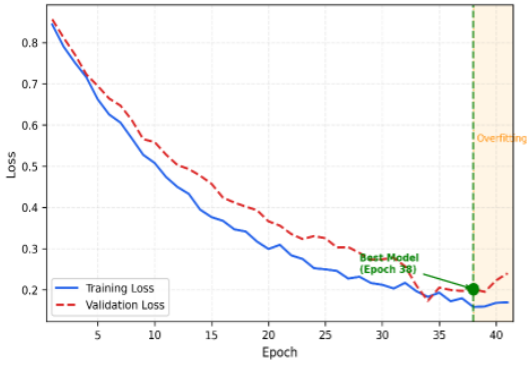

Figure 9. Before Fine-tuning - Loss

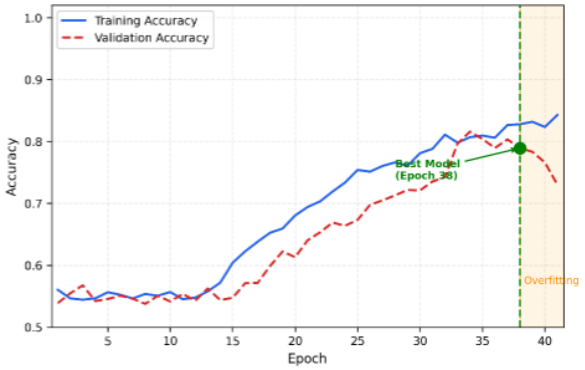

Figure 10. Before Fine-tuning – Accuracy

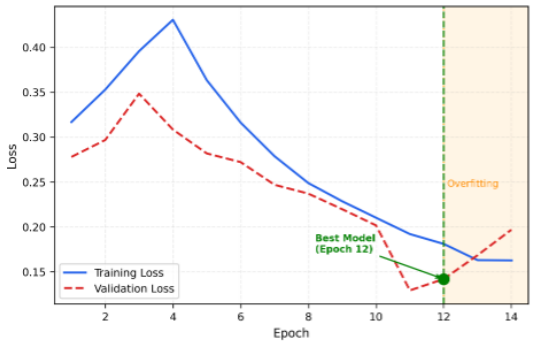

Figure 11. After Fine-tuning -Loss

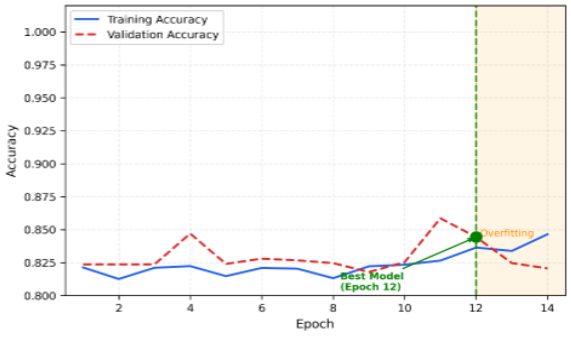

Figure 12. After Fine-tuning – Accuracy

epochs 9 and 12, with the minimum validation loss of approximately 0.148 and the corresponding training loss of approximately 0.179 at epoch 12, indicating that the model had reached a well-generalized state with a negligible train/validation loss gap of only 0.031. Following epoch 12, a clear divergence emerged. The validation loss rose to approximately 0.20 by epoch 14 while the training loss continued to decrease to approximately 0.157, a gap of 0.043, indicating renewed overfitting. Therefore, the training was stopped with the best model checkpoint saved at epoch 12, where the train/validation loss gap was at its minimum.

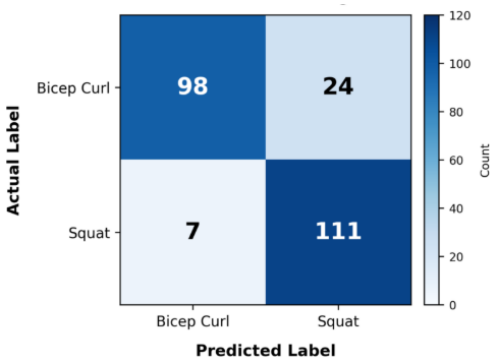

Figure 13. Confusion Matrix before Fine-tuning

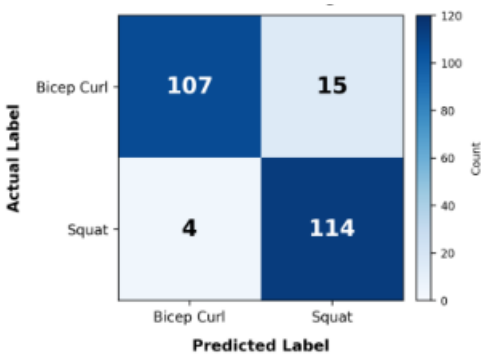

Figure 14. Confusion Matrix after Fine-tuning

The above Confusion Matrices demonstrates that fine-tuning the model resulted in improved classification. Prior to fine-tuning, in Figure 13, the Model misclassified 24 bicep curls into squats, and 7 squats into bicep curls. Following fine-tuning, in Figure 14, the Model had 15 misclassifications of bicep curls into squats and 4 squats into bicep curls; however, Correct classifications increased in both cases. Improvements in class separation were demonstrated, and the model was able to more accurately learn exercise patterns after being fine-tuned. Overall, the fine-tuning of the Model has shown improved performance by accurately identifying similar exercise movements.

4.2 Computational Efficiency Analysis

Table 4: Models Size Comparison

Model	Size	Temporal Modeling	Input	Edge Deployable
VGG16+LSTM	~553 MB	Yes	RGB	No
ResNet50	~96 MB	No	RGB (frames)	Partial
Proposed: ExerLiteNet	~178 MB	Yes (LSTM)	RGB	Yes

Table 4 compares ExerLiteNet against established video classification architectures in terms of model size. VGG16 [5] occupies approximately 553 MB while ResNet50 [10], used as the accuracy baseline in this work, has a backbone size of approximately 96 MB but provides no temporal modeling capability. ExerLiteNet, at approximately 178 MB, includes the complete spatiotemporal pipeline (MobileNetV3Small backbone followed by stacked LSTM layers) and classification head. It remains substantially smaller than all video-capable architectures, confirming its suitability for deployment on resource-constrained edge devices.

The proposed model processes input at approximately 15 frames per second from a standard webcam feed, with an end-to-end inference latency of ~2.3 seconds per 15-frame sequence. This includes frame capture, preprocessing (resize + normalize), sequence construction, and forward pass. The proposed model requires ~1.8 GFLOPs per inference sequence, compared to ~15.43 GFLOPs for VGG16 and ~3.8 GFLOPs for ResNet50, confirming its suitability for resource-constrained deployment.

5. Discussion

The proposed ExerLiteNet model achieved a test accuracy of 92.08% and an F1-score of 91.83% after fine-tuning, which is a clear improvement over the frozen ResNet50 baseline (82.31%) and the frozen MobileNetV3Small + LSTM model (87.08%). This difference highlights the significant contribution of temporal modeling to classification performance. Even though ResNet50 is a much larger model, it struggled to distinguish between exercises that look similar in individual frames, such as the standing position that appears in both squats and bicep curls. Adding LSTM layers helped the model understand the motion pattern across frames rather than relying on a single frame at a time. fine-tuning the last 20 layers of MobileNetV3Small further improved performance because the pretrained ImageNet weights were originally trained for general object recognition and not for detecting body movements specific to exercises. After

fine-tuning, the model was better at picking up on features like arm position and leg movement, which also resulted in a smaller gap between training and validation loss at the best checkpoint.

Comparing the results with related works, ExerLiteNet performs competitively considering its simple hardware requirements. YogNet by Yadav et al. [2] achieved 96.31% accuracy but it uses OpenPose for skeleton extraction and heavy 3D CNNs, which are not suitable for low-end devices. Bang et al. [3] achieved 92.1% on a 10-class workout dataset but their model depends on MediaPipe to extract joint coordinates before classification. Sensor-based methods like Wu et al. [1] at 95.39% and Chen et al. [6] at 92.5% also perform well but require wearable sensors and a Kinect camera respectively, which most people do not have at home. ExerLiteNet achieves accuracy comparable to Chen et al.'s Kinect-based system, while relying solely on a standard webcam., with a model size of approximately 178 MB and around 1.8 GFLOPs per inference, which is much lower than VGG16+LSTM at around 553 MB and 15.43 GFLOPs.

Looking at the remaining errors, the model still misclassified 19 samples in the test set. 15 bicep curls were predicted as squats and 4 squats were predicted as bicep curls. The higher error rate for bicep curls is likely because bicep curl movements are smaller and less obvious compared to squats, which involve a clear up and down movement of the entire body across frames. Some sequences may also have been captured during the rest positions between repetitions, where both exercises look almost the same, making it harder for the model to give the correct prediction. The fact that the model started over-fitting just two epochs after the best checkpoint also suggests that the main limitation is the size of the dataset rather than the model architecture itself, which is further discussed in Section 6.1.

6. Conclusion

This study presents a lightweight deep learning architecture for exercise recognition from RGB video sequences using spatiotemporal learning. The proposed architecture incorporates a MobileNetV3Small network to extract spatial features from each video frame sequence while using a long short-term memory (LSTM) network to factor in temporal models of video frame sequencing. The model was trained on a custom dataset of exercise videos and evaluated using standard performance metrics. The experimental results showed that the proposed model achieved an accuracy rate of 92.08% once fine-tuned, with improved classification efficiency and fewer misclassified instances. Class separation was better with fine-tuning than with normal frozen CNN backbone, as evidenced by the confusion matrix analysis. This allows for faster calculations and still provides excellent performance from the lightweight CNN backbone; thus, the model can be deployed on embedded systems and edge platforms where there is limited computing power.

Overall, the findings of this study validate that combining lightweight convolutional architectures with sequence modeling techniques provides an effective solution for video-based exercise recognition. The proposed approach supports scalable and accessible fitness monitoring systems using standard RGB input without the need for specialized hardware.

6.1 Limitations

The current model struggles with low-resolution input or poor lighting conditions, variations in image quality and illumination affect the extraction of features, resulting in misclassifications in some instances. The current model is also based on the assumption that only one person is being displayed within the frame; therefore, it cannot classify an exercise correctly for multiple individuals who are exercising at the same time. Additionally, the dataset used for training is relatively small, consisting of only 299 videos across two exercise classes, which limits the model's ability to generalize. This is also reflected in the training behavior, where the model began over-fitting just two epochs after the best checkpoint, suggesting that the performance is constrained more by the amount of available training data than by the model architecture itself.

6.2 Future Improvements

Future work will first address the limitations related to input quality by improving robustness to low resolution and varying lighting conditions through data augmentation and enhanced preprocessing techniques. To support multi-person scenarios, a bounding-box-based detection method will be introduced so that each individual can be processed independently. The number of classes that can be used in this system will increase, which will increase the real-world applicability of the system to fitness environments. Integration with IoT devices will also allow for more standalone systems for real-time monitoring of individuals in a gym or at home. Future work will also include advanced methods of posture analysis using expert knowledge for more accurate feedback after detecting the type of exercise being performed.

Acknowledgment

The authors express gratitude to the Department of Computer and Electronics Engineering at Kantipur Engineering College for research facilities and technical support. Grateful appreciation for guidance in this study goes to Er. Sahaj Shakya.

References

[1] Z. Wu, J. Zhang, K. Chen, and C. Fu, "Yoga posture recognition and quantitative evaluation with wearable sensors based on two-stage classifier and prior bayesian network," *Sensors*, vol. 19, pp. 5129-5140, 2019.

[2] S.K. Yadav, A. Agarwal, and A. Kumar, "Yognet: A two-stream network for real-time multi-person yoga action recognition and posture correction," *Knowledge-Based Systems*, vol. 1, pp. 10-20, 2022.

[3] G. S. Bang and S. B. Park, "Workout Classification Using a Convolutional Neural Network in Ensemble Learning," *Sensors (Basel)*, vol. 24, p. 3133, 2024.

[4] F. B. Ashraf, M. U. Islam, M. R. Kabir, and J. Uddin, "YoNet: A Neural Network for Yoga Pose Classification," *SN Computer Science*, vol. 4, p. 198, 2023.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, vol. arXiv:1511.08458, Warsaw, 2014.

[6] H. T. Chen et al., "Yoga posture recognition for self-training," in *International Conference on Multimedia Modeling*, 2014.

[7] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint*, vol. arXiv:1511.08458, 2015.

[8] A. Howard et al., "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[9] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.