Project Synopsis

on

"Human Action Recognition"

Submitted in complete fulfillment of the requirement for the degree of Bachelors of Engineering by

Gagan Prajapati
Usama Patel
Rajesh Pandit
Kunal Dubey

Under the guidance of
Prof. Priya Deshpande

LOKMANYA TILAK COLLEGE OF ENGINEERING

Affiliated to

UNIVERSITY OF MUMBAI

Department of Electronics & Telecommunication Engineering
Academic Year – 2024-25

CERTIFICATE

This is to certify that the mini project entitled **"Human Action Recognition"** is a bonafide work of **Gagan Prajapati,Usama Patel,Rajesh Pandit,Kunal Dubey** submitted to University Of Mumbai in partial fulfillment of requirement for **"Bachelor Of Engineering"** in **"Computer Science and Engineering (Data Science)"**

Prof.Priya Deshpande	**Dr.Nandini C. Nag**	**Dr.Subash Shinde**
(Project Guide)	(Head of Department)	(Principal)

MINI PROJECT APPROVAL

This mini project entitled **"HUMAN ACTION RECOGNITION"** by **Gagan Prajapati,Usama Patel,Rajesh Pandit,Kunal Dubey** is approved for degree of **Bachelor Of Engineering in Computer Science and Engineering (Data Science)**

Examiner

ACKNOWLEDGEMENT

We would like to acknowledge and extend our heartfelt gratitude to all those people who have been associated with this project and have helped us with it thus making it a worthwhile experience.

Firstly we extend our thanks to various people which include our project guide **Prof.Priya Deshpande** who has shared her opinions and experiences through which we received the required information crucial for our project synopsis. We are also thankful to head of the department **Dr.Nandini C Nag** and all the staff members of Data Science Department for their highly co-operative and encouraging attitudes, which have always boosted us.

We also take this opportunity with great pleasure to thank our Principal **Dr.Subash Shinde** whose timely support and encouragement has helped us succeed in our venture.

Name of Candidate	Signature
1.Gagan Prajapati	
2.Usama Patel	
3.Rajesh Pandit	
4.Kunal Dubey	

ABSTRACT

Human Action Recognition (HAR) is a crucial field in computer vision and artificial intelligence, with applications ranging from surveillance systems to interactive fitness trainers. This project focuses on developing a robust model to recognize human actions from video sequences, using a Convolutional Long Short-Term Memory (ConvLSTM) approach. The model is trained on the UCF50 dataset, which consists of 50 different human actions, achieving an accuracy of 80.33 percent. Although this accuracy may seem moderate, it is sufficient given the challenges associated with processing large video datasets.

The model is capable of predicting a variety of actions, such as JavelinThrow, Swing, WalkingWithDog, TaiChi, and Pullups, among others. Each video consists of an average of 320 frames, and the model effectively learns to recognize temporal dependencies between these frames to accurately classify the actions. To ensure the model performs well during real-time predictions, a specialized video data preprocessing pipeline has been designed to match the model's training data format. This processing pipeline ensures the system can generalize and predict actions from unseen videos efficiently.

The project's results demonstrate that deep learning models, particularly ConvLSTM, can successfully perform human action recognition, opening possibilities for future improvements and applications in various domains such as security, sports analytics, and human-computer interaction.

Contents

List of Figures

List of Abbreviations

CNN . . . . . Convolutional Neural Network

LSTM . . . . Long Short-Term Memory

.pkl . . . . . Pickle

ConvLSTM . Convolutional Long Short-Term Memory

RNN . . . . . Recurrent Neural Network

LRCN . . . . Long-term Recurrent Convolutional Networks

.h5 . . . . . Hierarchical Data Format version 5 (HDF5)

ReLU . . . . . Rectified Linear Unit

GRU . . . . . Gated Recurrent Unit

FC . . . . . Fully Connected (layer)

TPU . . . . . Tensor Processing Unit

GPU . . . . . Graphics Processing Unit

PCA . . . . . Principal Component Analysis

SGD . . . . . Stochastic Gradient Descent

MSE . . . . . Mean Squared Error

IoU . . . . . Intersection over Union

ROC . . . . . Receiver Operating Characteristic

AUC . . . . . Area Under the Curve

IoT . . . . . Internet of Things

Chapter 1

INTRODUCTION

Behavior action recognition using ConvLSTM (Convolutional Long Short-Term Memory) combines the strengths of convolutional neural networks (CNNs) for spatial feature extraction and LSTM networks for capturing temporal dependencies in sequential data, such as videos. ConvLSTM is particularly well-suited for action recognition because it can effectively model both the spatial structure of video frames and the temporal evolution of actions over time.

In this approach, the CNN component extracts relevant spatial features from each frame of the video, capturing details like object appearance, movement, and scene context. These features are then fed into the LSTM layer, which processes the sequence of frames to learn temporal patterns—essential for understanding how actions unfold over time.

This method is especially powerful for tasks like recognizing complex human actions or behaviors, where the temporal relationships between consecutive frames play a key role. ConvLSTM networks are often used in applications like human behavior analysis, sports action recognition, and more.

In summary, using ConvLSTM for behavior action recognition allows for an efficient and accurate understanding of both the spatial and temporal aspects of actions in video data.

1.1 Motivation

The motivation for building a human action recognition system stems from its potential to impact various domains positively:

1. Understanding Human Behavior: These systems facilitate the analysis and interpretation of human actions, enhancing human-machine interaction.

2. Surveillance and Security: Action recognition enhances security by detecting suspicious behaviors in public spaces, allowing for quicker responses.

3. Enhancing User Experience: In gaming, virtual reality, and robotics, recognizing actions leads to more immersive and responsive experiences.

4. Healthcare Applications: They are crucial for monitoring patients, especially the elderly, to detect falls or unusual behaviors requiring timely intervention.

5. Sports Performance Analysis: Analyzing player movements provides valuable insights for strategy refinement and skill enhancement.

6. Human-Robot Interaction: Effective action recognition enables robots to assist users by responding to their needs in real-time.

7. Road Safety: The systems improve road safety by analyzing pedestrian behaviors and interactions with vehicles to help prevent accidents.

8. Real-Time Analysis: These systems offer immediate feedback in critical applications like sports broadcasting and autonomous systems.

9. Multimodal Learning: Integrating data from various sources improves robustness and accuracy in recognizing complex actions.

10. Research Advancements: Developing these systems pushes the boundaries of AI and computer vision, fostering innovation in algorithms and technologies.

In summary, building human action recognition systems aims to enhance understanding of behaviors, improve user experiences, promote safety, and drive technological advancements across diverse applications.

1.2 Problem Statement

Human action recognition (HAR) poses significant challenges in accurately identifying and classifying actions from video sequences due to the complexity of processing large datasets and capturing temporal dependencies between frames. Current systems struggle with real-time predictions and maintaining high accuracy across a wide variety of actions. The goal of this project is to develop a robust HAR model that can efficiently recognize human actions in videos, addressing these challenges using a ConvLSTM approach. This system aims to provide reliable predictions for real-time applications like surveillance and sports analytics, while overcoming the difficulties of processing video data.

Chapter 2

LITERATURE SURVEY

2.1 Referred Paper 1

The ConvLSTM model proposed in this paper could potentially be used to build a human action recognition model, as the authors mention that they plan to investigate applying ConvLSTM to video-based action recognition in future work . The key aspects that suggest this are:

1. The ConvLSTM model is designed to handle spatiotemporal data, preserving spatial information by using convolutional structures in the input-to-state and state-to-state transitions. This could be beneficial for modeling the spatial and temporal dynamics of human actions in video data.

2. The authors note that the ConvLSTM model can capture both fast and slow motions by adjusting the size of the convolutional kernels. This flexibility could be useful for recognizing a variety of human actions with different temporal characteristics.

3. The authors propose incorporating the ConvLSTM layer on top of convolutional neural network features, which is a common approach for video-based action recognition . This suggests the ConvLSTM model could be integrated into a broader deep-learning architecture for human action recognition.

However, the paper does not provide any direct evidence or details on how the ConvLSTM model would be applied to human action recognition. Further research and experimentation would be needed to validate the effectiveness of this approach.

2.2 Referred Paper 2

To build a human action recognition system from the paper, we would focus on these sections related to Long Short-Term Memory (LSTM), as it is specifically designed to overcome problems with time-sequence data, which is crucial for action recognition.

Key areas to use:

1. Long Short-Term Memory Architecture: - LSTM architecture, described in Section 4, provides a solution for learning time-sequences by maintaining constant error flow across time steps, which is essential for recognizing actions from a video over time. The memory cells and gate units (input and output gates) in LSTM are designed to store and retrieve relevant information without losing track of long-term dependencies.

2. Memory Cells and Gate Units: - As mentioned in Section 4 (Memory cells and gate units), the LSTM's capability to protect memory cell contents from irrelevant perturbations can be leveraged to process sequential video frames for action recognition. This allows the system to focus on important information across frames while ignoring irrelevant actions.

3. Experiment Results on Time Sequence Tasks: - The experiments in Section 5 show that LSTM outperforms other methods in handling time-sequences with long time lags and noise, which is typical in video-based action recognition tasks where relevant actions occur at different times.

In short, the LSTM's architecture, its ability to handle long-term dependencies, and its efficiency in processing noisy or scattered information in time-sequence tasks make it a good fit for building a robust human action recognition system.

2.3 Referred Paper 3

For this on Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) regularization, here's how we would use the concepts:

1. Regularizing the Model: - This paper introduces **dropout regularization for LSTMs, which can prevent overfitting when training on large datasets for complex tasks such as human action recognition. Dropout would help improve the model's generalization by preventing the LSTM from memorizing training data, ensuring that the network learns meaningful features rather than overfitting to specific sequences.

2. Memory Management with LSTMs: - The LSTM architecture (Section 3.1) discussed in this paper provides a robust way to handle long-term dependencies in time-series data, which is critical for action recognition as human actions span over multiple video frames. The ability of LSTMs to decide whether to forget or retain information in memory cells will be crucial for handling video data that includes both relevant actions and irrelevant movements.

3. Applying Dropout to Non-Recurrent Connections (Section 3.2): - Dropout is applied to non-recurrent connections, which allows the network to retain its capability of remembering important sequences while still benefiting from dropout's regularization effect. This setup would be ideal for your action recognition system, as it ensures that only a subset of connections are "dropped" during training, preventing the corruption of long-term memory retention that is necessary for recognizing human actions across sequences.

4. Multilayer LSTM for Robustness: - The multilayer LSTM (as explained in Figure 2 of the paper) is helpful in creating deep architectures that can learn hierarchical patterns in human actions. This would allow the system to model both low-level movements and high-level actions.

By combining the concepts of LSTM memory cells, dropout regularization, and a multilayer architecture, you can create a robust action recognition system that generalizes well to new video sequences and efficiently handles both short- and long-term dependencies.

2.4 Research paper4

The UCF50 dataset is a widely-used benchmark for action recognition tasks, consisting of 50 action categories derived from realistic videos sourced from YouTube. This dataset was developed as an extension of the UCF11 dataset, offering a broader range of activities such as sports, daily actions, and performances. Unlike many other datasets that feature staged scenarios, UCF50 emphasizes real-world complexity by including variations in camera motion, object scale, background clutter, lighting conditions, and object pose.

Each of the 50 action categories contains videos grouped into 25 groups, with each group consisting of at least four clips. Clips within a group often share certain similarities, such as the same person or background, which presents a more nuanced challenge for recognizing actions consistently across different but related scenes.

Given its realistic nature and diversity, UCF50 has been instrumental in testing the generalizability and robustness of deep learning models in action recognition. For instance, our model has been trained on this dataset, and despite the challenges associated with handling large and diverse video sequences, we achieved an accuracy of 80.33%.

The dataset has provided a robust platform for evaluating the temporal learning capabilities of models like ConvLSTM, particularly in capturing the temporal dependencies across video frames to perform accurate action recognition.

2.5 Existing System & its Drawbacks

1. Google's Video Intelligence API Drawbacks: Primarily focuses on general object detection and video content analysis. It lacks specialized features for real-time action recognition and struggles with complex human movements.

2. Microsoft Azure Video Analyzer Drawbacks: While it provides video processing capabilities, the system may have latency issues in real-time applications and is not designed specifically for fine-grained action recognition in health or rehabilitation contexts.

3. Facebook's Action Recognition for Content Moderation Drawbacks: Mainly focused on content moderation and detecting harmful behaviors in social media content, making it unsuitable for real-time surveillance or rehabilitation needs.

4. IBM Watson Media Drawbacks: This platform focuses more on content analysis and personalized media experiences rather than real-time human action recognition, especially for applications in security, health, and therapy.

5. NVIDIA DeepStream SDK Drawbacks: While highly optimized for real-time video analytics, the complexity of setting up and using the SDK can be a barrier. It may not provide built-in solutions for healthcare or rehabilitation-specific tasks.

6. OpenPose for Health and Fitness Apps Drawbacks: Although it's excellent at pose estimation, it requires significant computational resources and customization to be applied in real-time surveillance or assistive technologies.

7. Real-Time Surveillance Systems Drawbacks: Most existing real-time surveillance systems focus on detecting predefined security threats rather than human action recognition. They often miss nuanced actions that can indicate potential health or security concerns.

8. Sports Analytics Platforms Drawbacks: These platforms are optimized for analyzing performance in athletic environments but may not translate well into general human action recognition for accessibility or rehabilitation purposes.

9. Assistive Technologies Drawbacks: Many assistive technologies are focused on specific disabilities and lack the versatility to detect a wide range of human actions in real-time, limiting their effectiveness.

Chapter 3

PROPOSED METHODOLOGY

3.1 Introduction

To develop the Human Action Recognition model, we will utilize convolutional layers, followed by Long Short-Term Memory (LSTM) layers to capture temporal patterns and dependencies in the video sequences. Before building and training the model, we will first preprocess the video data, which involves extracting frames and preparing the dataset for optimal input to the neural network. The detailed steps of data processing and frame extraction are outlined below

3.2 Flowchart

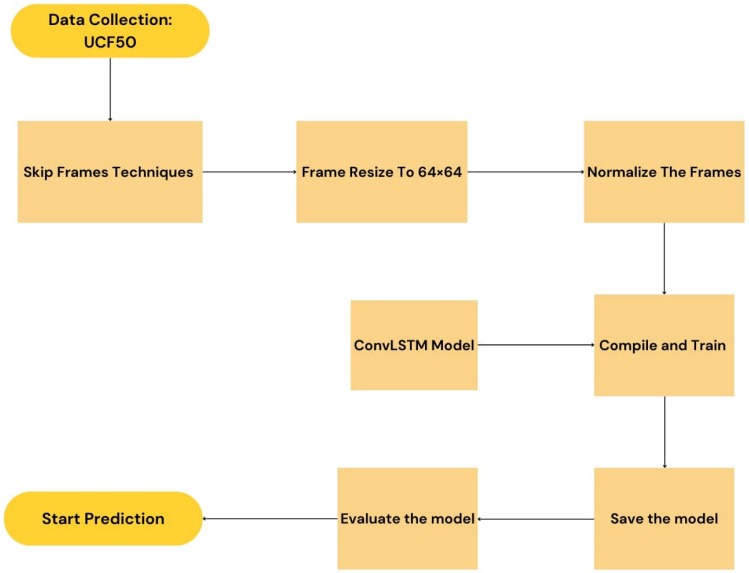

Figure 3.1: Flowchart

3.3 Dataset Processing

First, we will read the video files from the UCF50 dataset and resize each frame to a fixed dimension of 64x64 pixels. This resizing helps reduce computational complexity while maintaining essential features of the actions. Next, we will normalize the pixel values of each frame by scaling them to the range [0-1], achieved by dividing the pixel values by 255. This normalization ensures faster convergence during model training. The model will be trained on all 50 action from the dataset, which include:

CLASSES_LIST = ['BaseballPitch', 'Basketball', 'BenchPress', 'Biking', 'Billiards', 'BreastStroke', 'CleanAndJerk', 'Diving', 'Drumming', 'Fencing', 'GolfSwing', 'HighJump', 'HorseRace', 'HorseRiding', 'HulaHoop', 'JavelinThrow', 'JugglingBalls', 'JumpingJack', 'JumpRope', 'Kayaking', 'Lunges', 'MilitaryParade', 'Mixing', 'Nunchucks', 'PizzaTossing', 'PlayingGuitar', 'PlayingPiano', 'PlayingTabla', 'PlayingViolin', 'PoleVault', 'PommelHorse', 'PullUps', 'Punch', 'PushUps', 'RockClimbingIndoor', 'RopeClimbing', 'Rowing', 'SalsaSpin', 'SkateBoarding', 'Skiing', 'Skijet', 'SoccerJuggling', 'Swing', 'TaiChi', 'TennisSwing', 'ThrowDiscus', 'TrampolineJumping', 'VolleyballSpiking', 'WalkingWithDog', 'YoYo']

3.3.1 Frames Extraction

The frame extraction function is designed to extract the required frames from a video after resizing and normalizing them. First, we declare an empty list called frames_list to store the extracted video frames. Next, we utilize the cv2.VideoCapture object with video_reader = cv2.VideoCapture(video_path) to read the video files. After this, we count the total number of frames in the video using video_frames_count =int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

To optimize performance and reduce computational costs, we employ a skip frame technique that allows us to extract frames at specified intervals instead of processing every single frame.

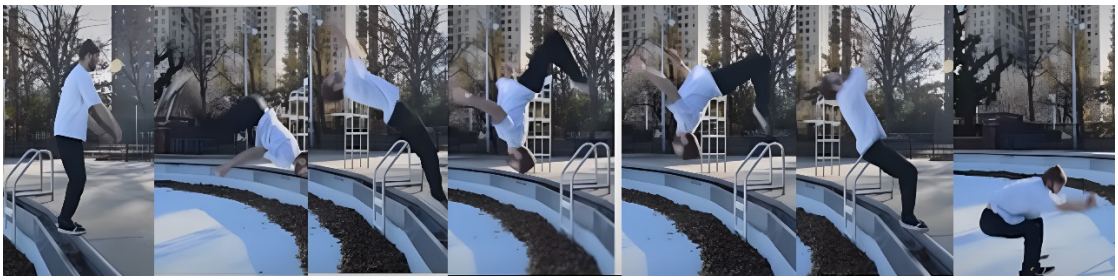

Figure 3.2: BackFlipping

Example of the Skip Frame Technique: The image above depicts a backflipping activity consisting of 7 frames. By analyzing all 7 frames, we can accurately predict the activity.

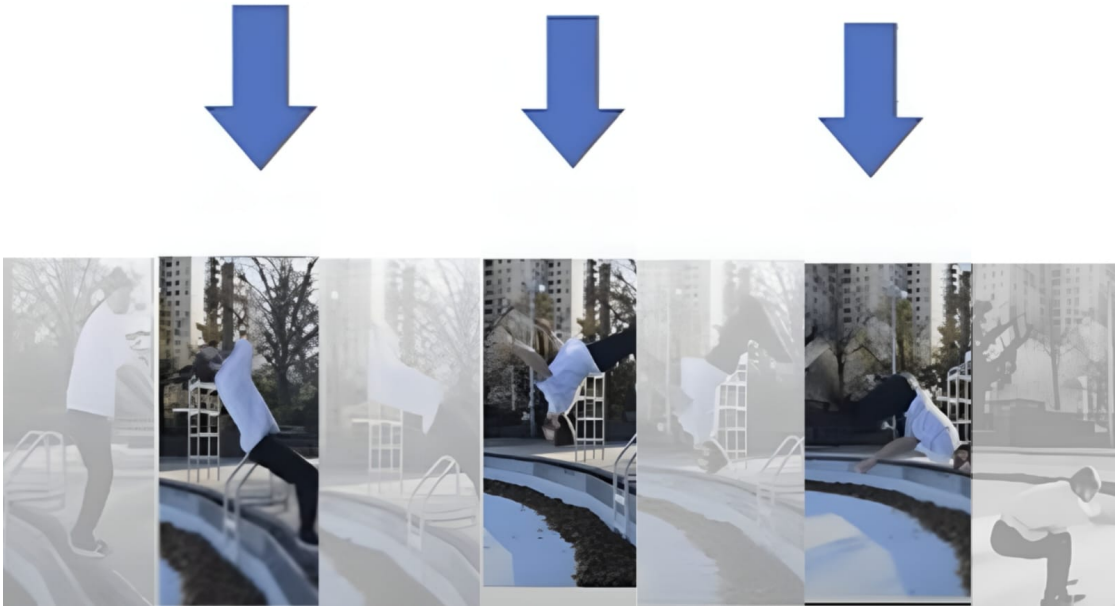

Figure 3.3: BackFlipping After Applying Skip Frames Technique

However, after applying the skip frame technique, as shown in the image below, we can make the same prediction using only 3 frames. This results in approximately a 50% reduction in computational cost while still effectively predicting the activity. Essentially, the skip frame technique involves skipping a fixed number of frames from the total and selecting the subsequent frame for analysis.

3.3.2 Create Dataset

Create an empty list `features=[ ]`, `labels = [ ]` to store features and corresponding labels, Iterate through all the classes specified in the class list, apply the skip frame technique, and append the resulting frames to the features list along with their corresponding labels.

After populating the labels list, use Keras's `to_categorical` method to convert the labels into `one-hot-encoded vectors`. Applying one-hot encoding to the labels in a dataset with 50 different classes enhances usability for classification tasks. Each label, ranging from 0 to 49, is converted into a binary vector of length 50, where only one position is set to 1 to indicate the class, while the rest are 0. For instance,

class 0 is represented as [1, 0, 0, ..., 0], class 1 as [0, 1, 0, ..., 0], and so on. This method treats each class as distinct, preventing the model from misinterpreting the labels as having any order. It also allows the model to output a probability distribution across all classes, making it compatible with categorical cross-entropy loss during training. Overall, one-hot encoding improves model performance and interpretability, enabling better learning of each class's unique features.

3.3.3 Split the Data into Train and Test Set

So now, we will split our data to create training and testing sets. We will also shuffle the dataset before the split to avoid any bias and get splits representing the overall distribution of the data.

3.4 Implement the ConvLSTM Approach

3.4.1 Convolutional Neural Network (CNN)

.

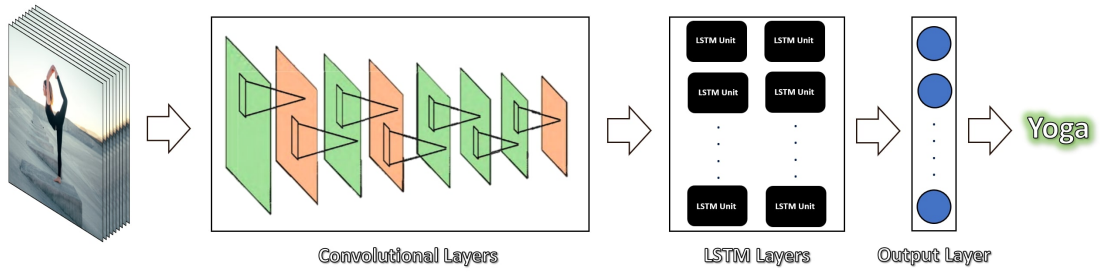

Figure 3.4: Convulational Layer

Convolutional layers are designed to capture spatial patterns in data, typically images or frames of a video. They apply filters to the input data, enabling the model to extract features like edges, textures, and other visual elements. As more filters are added, more complex and abstract features can be identified. This approach is highly effective for tasks that require pattern recognition in images, such as video classification or object detection.

3.4.2 Long Short-Term Memory (LSTM)

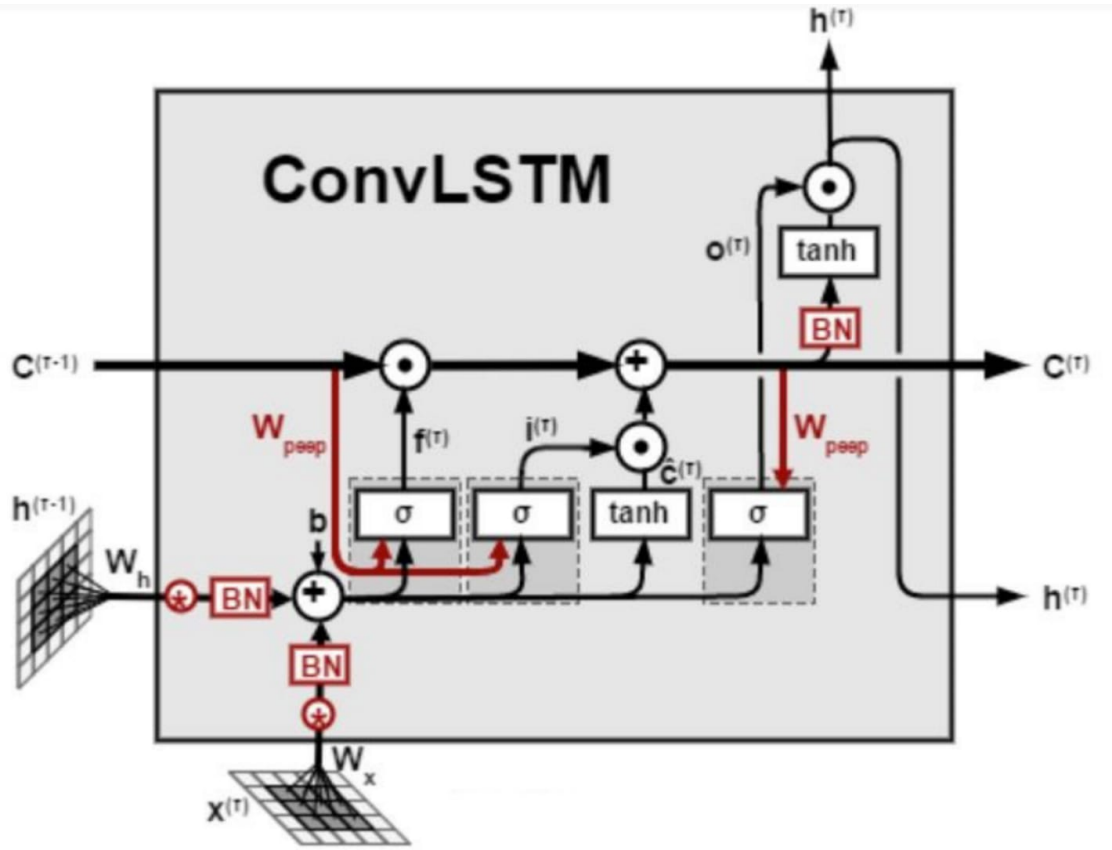

Figure 3.5: Long Short-Term Memomry(LSTM)

LSTM is a type of recurrent neural network (RNN) designed to handle sequence data, especially when there are long-term dependencies. Unlike traditional RNNs, which struggle to remember information over long sequences due to the problem of vanishing gradients, LSTMs are equipped with special gates that control the flow of information, allowing them to retain important data over time.

An LSTM unit has three primary gates:

1. Forget Gate: Decides which information from the previous time step should be discarded.

2. Input Gate: Determines what new information should be added to the cell state.

3. Output Gate: Controls what part of the cell state should be output as the hidden state for the next time step.

In your ConvLSTM model, the LSTM component processes the sequence of frames extracted from the video, learning how the spatial features change over time. This temporal understanding is crucial for accurately recognizing actions, enabling the model to distinguish between similar activities like walking and running, or sitting and standing.

3.4.3 Model Architecture

We start with a ConvLSTM2D layer that applies convolution over both spatial and temporal dimensions of the input frames. The number of filters is set to 4 in the first layer, meaning it will learn 4 feature maps, and the kernel size of (3, 3) defines the window for these convolutions. More filters added in subsequent layers enable the model to learn more detailed and abstract features. Recurrent dropout (set to 0.2) helps prevent overfitting, and return_sequences=True ensures that the LSTM passes the processed sequence data to the next layer.

MaxPooling3D layers downsample the feature maps, reducing spatial dimensions while keeping important features intact. The TimeDistributed Dropout layers apply dropout across time steps, where 20% of neurons are dropped to regularize the model and reduce overfitting, ensuring that the remaining neurons are more robust.

Finally, a Dense layer with the Softmax activation function produces the output. Softmax converts the final output into probabilities, distributing the likelihood across all class labels. For example, if the model is predicting a video showing a swimming activity, the "swimming" class might have a 90% probability, while other classes share the remaining 10%. This helps in making confident predictions about which class the video belongs to.

Below is the complete structure of the model:

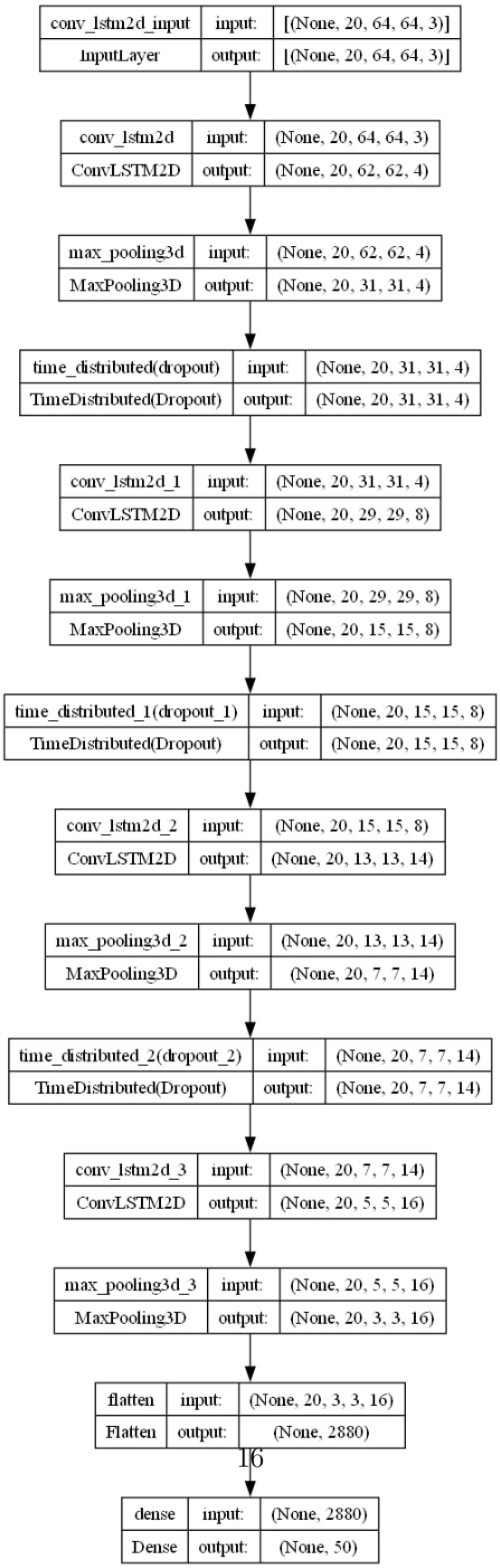

Figure 3.6: ConvLSTM Model Structure

Chapter 4

Training And Evaluation

4.1 Compile & Train the Model

Here's an easy-to-understand description of the code:

1. Early Stopping Callback:
The `EarlyStopping` callback monitors the model's performance on the validation set `(val_loss)` during training.
If the validation loss doesn't improve for 10 consecutive epochs (patience = 10), training stops early to prevent overfitting.
The parameter `mode='min'` ensures that the goal is to minimize the validation loss.
`restore_best_weights` =True means that once training stops, the model's weights will revert to the bestperforming version (the one with the lowest validation loss).

2. Compiling the Model:
The model is compiled by specifying the loss function, optimizer, and evaluation metrics.
The loss function is **categorical cross-entropy**, which is suitable for multi-class classification.
The **Adam optimizer** is used to adjust the learning rate during training.
The model will also track **accuracy** as a performance metric.

3. Training the Model:
The model is trained on the `features_train` and `labels_train` datasets for up to 50 epochs, using a batch size of 4.
The data is shuffled at each epoch `(shuffle=True)` to prevent the model from learning any unintended order dependencies.
20% of the training data is reserved for validation `(validation_split=0.2)`, allowing the model to test its performance on unseen data during training.
The `EarlyStopping` callback is applied to monitor and potentially stop training early if needed.

4. Label encoder is utilized to convert categorical class labels into numerical values, which the model requires for training. This transformation not only ensures consistency between training and validation datasets but also facilitates the loss calculation during training. The label encoder thus plays a critical role in preparing the data for effective model training and evaluation.

4.2 Evaluating the Trained Model

4.2.1 Analysis of Loss Curves

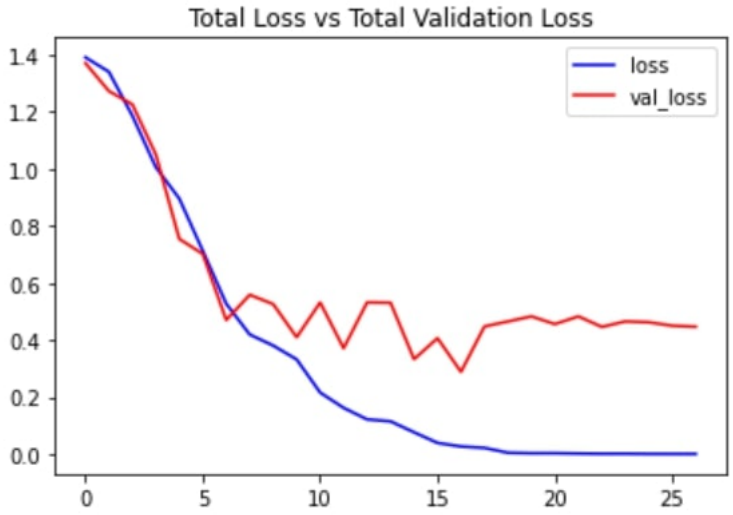

Figure 4.1: Total Loss vs Total Validation Loss

Analysis of loss curves:

General Observations: Decreasing Loss: Both the training loss (blue line) and validation loss (red line) are generally decreasing over the epochs, indicating that the model is learning.

Overfitting Potential: The validation loss starts to increase after a certain point, while the training loss continues to decrease. This suggests a potential for overfitting, where the model is becoming too specialized to the training data and may not generalize well to unseen data.

Specific Insights:

Initial Training: The model seems to be learning rapidly in the initial epochs, with both training and validation loss dropping significantly.

Overfitting Region: The point where the validation loss starts to increase indicates the onset of overfitting. This might be due to the model learning noise or irrelevant patterns in the training data.

Gap Between Training and Validation Loss: The increasing gap between the training and validation loss further emphasizes the overfitting issue. The model is performing better on the training data than on the validation data.

4.2.2 Analysis of Accuracy Curves

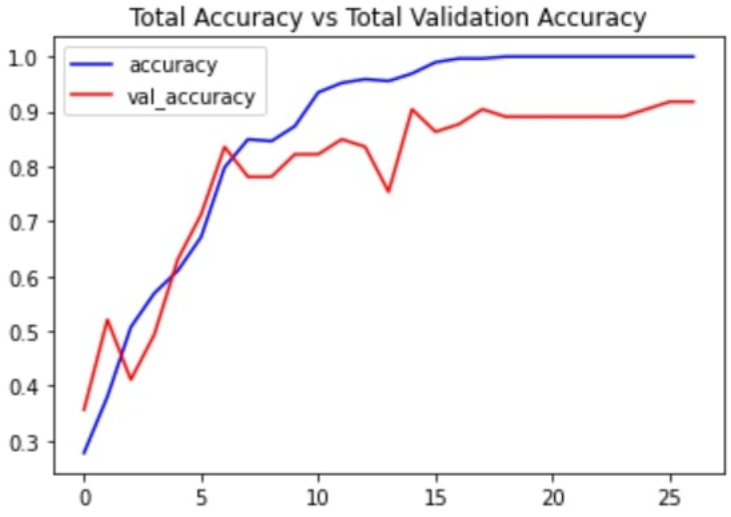

Figure 4.2: Total Accuracy vs Total Validation Accuracy

Analyzing the Accuracy Curves

General Observations:

Increasing Accuracy: Both the training accuracy (blue line) and validation accuracy (red line) are generally increasing over the epochs, indicating that the model is learning.

Potential Overfitting: While the training accuracy continues to increase, the validation accuracy plateaus or starts to decrease slightly. This suggests a potential for overfitting, where the model is becoming too specialized to the training data and may not generalize well to unseen data.

Specific Insights:

Initial Training: The model seems to be learning rapidly in the initial epochs, with both training and validation accuracy increasing significantly.

Overfitting Region: The point where the validation accuracy starts to plateau or decrease indicates the onset of overfitting. This might be due to the model learning noise or irrelevant patterns in the training data.

Gap Between Training and Validation Accuracy: The increasing gap between the training and validation accuracy further emphasizes the overfitting issue. The model is performing better on the training data than on the validation data.

4.3 Acurracy Score

During the training process, the model achieved a loss of 0.8976 and an accuracy of 80.33% after completing all four batches. The loss value indicates how well the model's predictions align with the actual labels, with lower values representing better performance. An accuracy of over 80% suggests that the model correctly classified a significant majority of the training samples, demonstrating its effectiveness in recognizing human actions within the dataset. This performance reflects the model's ability to learn meaningful patterns from the data, which is crucial for successful action recognition tasks.

4.4 Saving The Model

The lines `convlstm_model.save(model_file_name)` and `joblib.dump(label_encoder, encoder_file_name)` save the trained ConvLSTM model and the label encoder, respectively. The model is saved in the **.h5** format, which preserves its architecture, weights, and training configuration. This allows for future use without the need for retraining, making deployment and further fine-tuning on new data easier. Meanwhile, saving the label encoder as a **.pkl** file ensures that the mapping of original class labels to their corresponding numerical values is retained. This enables consistent decoding of predictions back to their original labels during inference, ensuring seamless integration between the training and prediction phases.

Chapter 5

Model Prediction

This model prediction code is designed for predicting human action recognition from video files using a pre-trained ConvLSTM model. The key components of the code include model loading, video frame preprocessing, prediction, and display of results.

5.1 Model and Label Encoder Loading

The paths for the saved ConvLSTM model (`conv_lstm_model.h5`) and the label encoder (`label_encoder.pkl`) are defined. The model is loaded into memory using `load_model` , while the label encoder is loaded using `joblib.load` . These components are crucial for performing predictions on new video data.

5.2 Video Frame Preprocessing

The `preprocess_video` function captures frames from a specified video file until the required sequence length (set to 10 frames) is reached. Each frame is resized to match the input dimensions expected by the model (`64x64` pixels) before being stored in a list. If there are not enough frames, the function gracefully handles the situation by returning an empty array.

5.3 Video Display with Prediction

The `display_video_with_prediction` function utilizes OpenCV to display the video along with the predicted action label. It first preprocesses the video frames and checks if there are enough frames for prediction. If so, it continuously reads and displays each frame, resizing it for broader visibility. The predicted class label is overlayed on each frame using OpenCV's putText function. The video can be exited by pressing the 'q' key.

5.4 Start Prediction Process

The `start_prediction` function iterates through a dataset containing multiple action classes (specifically from the UCF50 dataset). It attempts to process a single video from each class directory. If the user signals to stop (by pressing 'q'), the video display and processing will cease.

5.5 Output Images

Figure 5.1: Prediction 1

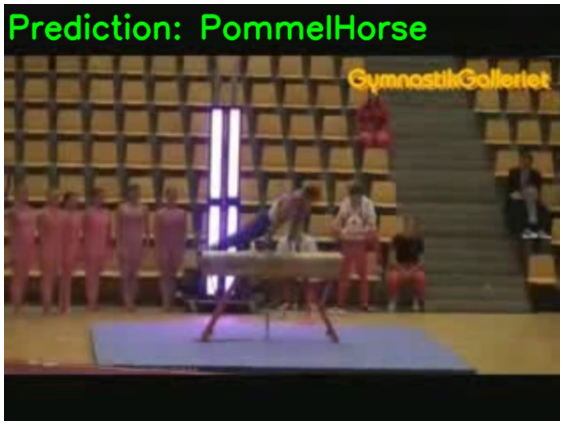

Figure 5.2: Prediction 2

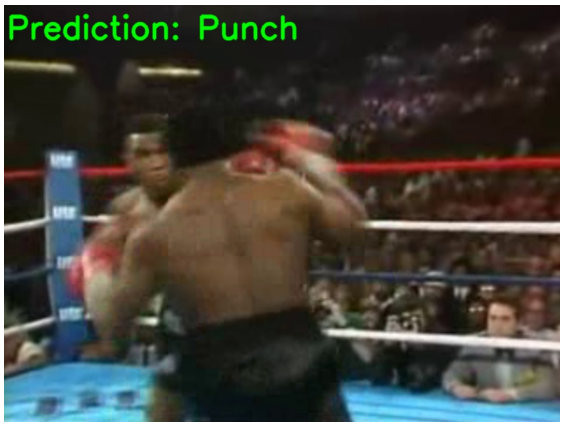

Figure 5.3: Prediction 3

Figure 5.4: Prediction 4

Chapter 6

CONCLUSIONS

In conclusion, this human action recognition project successfully demonstrates how to identify and classify human activities in videos using a ConvLSTM model. By capturing the temporal dynamics of video sequences, the system achieved high accuracy in recognizing various actions.

Key steps included data preprocessing, model training, and evaluation, with techniques like early stopping used to prevent overfitting. The integration of a label encoder ensured accurate predictions by mapping class labels to numerical values.

Overall, this project showcases the effectiveness of deep learning in action recognition and lays the groundwork for future improvements, such as using advanced architectures or larger datasets. The potential applications in areas like surveillance and sports analysis highlight the real-world relevance of this work.

Chapter 7

References

[1] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural Computation*, MIT Press, 1997.

[3] W. Zaremba, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[4] K. K. Reddy and M. Shah, "Recognizing 50 human action categories of web videos," *Machine Vision and Applications*, vol. 24, no. 5, pp. 971-981, 2013.