Spatiotemporal Action Recognition in Videos Using ConvLSTM with Attention: A Comparative Analysis and Implementation

Mohammed Kaif Kohari

Submitted for the Degree of Master of Science in

MSc Applied Data Science

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

Dec 9th, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 9812

Student Name: Mohammed Kaif Kohari

Date of Submission: Dec 9th, 2024

Signature: Mohammed. K Kohari

Acknowledgments

I would like to express my sincere gratitude to Professor Li Zhang for her invaluable guidance, support, and encouragement throughout the course of my master's thesis. Her expertise, insightful feedback, and constant motivation were instrumental in shaping the direction of this research. I am truly grateful for the opportunity to work under her supervision and for her continuous mentorship, which has greatly contributed to my academic and professional growth.

Abstract

Action recognition is a key task in computer vision, focused on identifying and classifying human activities within video sequences. Recent advancements in deep learning, particularly through convolutional neural networks (CNNs) and recurrent models, have greatly enhanced performance in this domain. The availability of large-scale video datasets, such as UCF-101, has accelerated progress by providing diverse human actions for models to learn from. However, challenges such as variations in human pose, complex backgrounds, occlusion, and motion blur still hinder accurate action classification.

This dissertation explores the application of Convolutional Long Short-Term Memory (ConvLSTM) networks for video action recognition. ConvLSTM integrates the spatial feature extraction capabilities of CNNs with the temporal modeling strengths of LSTM networks, enabling the model to capture both spatial and temporal dependencies within video data. The primary goal of this research is to develop an efficient, scalable model capable of recognizing actions in real-time. The UCF-101 dataset is used to evaluate the effectiveness of the proposed model, with performance compared against traditional CNN and LSTM approaches. Additionally, various preprocessing techniques and hyperparameter configurations are examined to understand their impact on model performance.

The results demonstrate that ConvLSTM is particularly effective at capturing both spatial features, such as textures and object shapes, and temporal dynamics, such as motion patterns and action progressions. The model's ability to integrate these aspects leads to robust performance in classifying actions, even in the presence of challenges like viewpoint variation and occlusion. This research introduces a novel application of ConvLSTM in video action recognition, contributing to advancements in accuracy, scalability, and real-time performance. The findings emphasize the potential of ConvLSTM as a powerful tool for advancing action recognition in domains such as surveillance, human-computer interaction, and autonomous systems.

Contents

1. Introduction

1.1 Background

Action recognition, a branch of computer vision, refers to the task of detecting and classifying actions or events performed by humans in a sequence of video frames. Unlike static image classification, action recognition requires models that can capture the dynamic nature of video data, as human actions unfold over time. The ability to analyze video content and understand the actions being performed is essential in several real-world applications, such as surveillance, healthcare, autonomous vehicles, especially for residents like us who live in megacities like London.

The importance of action recognition has grown significantly with the proliferation of video content on the internet, as well as the increase in the demand for intelligent systems that can analyze this content. In video analysis, challenges arise due to factors such as variations in viewing angles, illumination conditions, motion blur, and occlusion. These challenges have made the task of recognizing human actions from videos particularly difficult. Early methods in action recognition relied on handcrafted features such as optical flow, histograms of oriented gradients (HOG), and interest point descriptors. However, these methods were limited in their ability to scale and adapt to complex, large-scale datasets.

In recent years, deep learning techniques, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated superior performance in various computer vision tasks, including image classification and action recognition. These models are capable of learning hierarchical features from raw data, thus eliminating the need for manual feature extraction. The application of deep learning to video data has opened new doors for action recognition, allowing models to automatically learn both spatial and temporal patterns.

1.2 Motivation

The rapid proliferation of video content across various platforms—ranging from surveillance systems to social media—has created an ever-growing demand for automated video analysis. Action recognition, a critical subset of this domain, enables machines to interpret human activities in videos. This capability has profound implications in areas such as public safety, sports analytics, autonomous systems, and healthcare monitoring.

Despite significant progress in deep learning and computer vision, achieving accurate and real-time action recognition remains challenging due to the complexity of spatiotemporal dynamics, varying video resolutions, and class imbalances. Traditional methods often struggle with scalability and generalization in real-world scenarios.

The integration of advanced architectures such as Convolutional Neural Networks (CNNs) for feature extraction and Long Short-Term Memory networks (LSTMs) for temporal modeling presents a promising approach to tackling these challenges. By leveraging pre-trained ResNet-152 for spatial feature extraction and employing bi-directional LSTMs with attention mechanisms for temporal modeling, this research aims to develop a robust framework for action recognition.

This dissertation is motivated by the pressing need to bridge the gap between academic advancements and practical applications of action recognition, focusing on a scalable and efficient solution to empower systems to understand complex human actions in videos with high accuracy.

1.3 Objectives and Scope

Objectives

1. Investigate ConvLSTM for Video Action Recognition
   To study how ConvLSTM integrates spatial encoding, temporal modeling, and attention mechanisms to effectively capture spatiotemporal patterns in video data and predict results..

2. Application on UCF-101 Dataset
   To evaluate the ConvLSTM model on the UCF-101 dataset, a comprehensive benchmark containing 101 diverse action categories, testing its robustness across varying scenarios.

3. Performance Assessment
   To assess the model using metrics such as accuracy using different performance metric functions.

4. Addressing Action Recognition Challenges
   To address issues like variations in action execution, occlusion, and camera viewpoint shifts that impact real-world video-based models.

Scope

1. Focus on ConvLSTM
   The study exclusively explores ConvLSTM, combining ResNet-based feature extraction with temporal modeling and attention mechanisms, offering a specialized view of its performance.

2. Dataset Restriction
   The research confines itself to the UCF-101 dataset, enabling controlled experimentation and direct comparisons with prior studies.

3. Evaluation Metrics
   The study emphasizes classification metrics but excludes latency and real-time performance evaluations, leaving these aspects for future work.

4. Human Action Recognition Only
   The research strictly focuses on human action recognition and does not cover broader video analysis tasks like object detection or scene understanding.

2. Literature Review

The field of using Computer Vision architectures in videos has witnessed significant advancements, driven largely by the integration of different deep learning methodologies. This section reviews key developments in the domain, discussing foundational frameworks and state-of-the-art models that have shaped the field and helped our study for the research.

1. Historical Foundations of Action Recognition

Action recognition has its origins in traditional computer vision approaches that relied heavily on handcrafted features such as histograms of oriented gradients (HOG) and motion history images (MHI). These techniques, while pioneering, suffered from a lack of robustness in handling complex motion dynamics or variations in lighting, scale, and viewpoint (Laptev, 2005). The invention of Neural Networks marked a huge paradigm shift, introducing automated feature extraction through convolutional neural networks (CNNs) by learning patterns in pixels of image frames, which significantly improved performance by learning patterns and hierarchical feature representations of pixels in images.

The addition of architectures like LSTMs and Transformers which are used for language problems and are popular for capturing dependencies have transformed the field of Video recognition

2. Spatial-Temporal Modeling in Action Recognition

Two-Stream CNNs:
Simonyan and Zisserman (2014) introduced Two-Stream CNNs, which split the task of spatial and temporal modeling into separate networks. The spatial stream captures frame-level information, while the temporal stream processes stacked optical flow to model motion. Despite their success, the reliance on precomputed optical flow increased computational complexity, limiting scalability in real-time applications as they were only suited for optical images

3D Convolutional Neural Networks (C3D):
Tran et al. (2015) proposed extending traditional CNNs to 3D convolutions, allowing simultaneous learning of spatial and temporal features. C3D demonstrated superior performance on video datasets by capturing short-term motion dynamics. However, its reliance on fixed temporal kernels limited its ability to model long-range temporal dependencies effectively.

Inflated 3D Convolutions (I3D):
Carreira and Zisserman (2017) addressed the limitations of C3D by inflating 2D ImageNet-pretrained filters into 3D convolutions, allowing the transfer of knowledge from large-scale image datasets. This method significantly improved performance on datasets like UCF-101 and Kinetics, though its computational cost remained a concern due to its dual-stream architecture.

3. Advances in Temporal Modeling

LSTMs and ConvLSTMs

Long Short-Term Memory (LSTM) networks have been pivotal in sequential data processing. When coupled with convolutional encoders, they excel at capturing temporal dependencies in video data. The introduction of ConvLSTM extended this capability by integrating spatial feature encoding with temporal modeling (Shi et al., 2015). Bidirectional LSTMs further enhanced this framework by modeling both forward and backward dependencies, ensuring a comprehensive understanding of video sequences.

Temporal Shift Module (TSM)

Lin et al. (2019) proposed TSM as an easy alternative to 3D convolutions. By shifting feature maps across the temporal dimension within 2D CNNs, TSM achieved competitive performance while reducing computational overhead, making it suitable for real-time applications. However, its fixed temporal shifts limited its ability to model variable motion dynamics effectively.

4. Attention Mechanisms in Video Understanding

The integration of attention mechanisms has revolutionized action recognition by addressing the problem of irrelevant or redundant information in video frames. Mechanisms like soft attention dynamically assign importance weights to keyframes, enabling models to focus on critical action-relevant information.

5. Benchmark Datasets and Evaluation Protocols

UCF101

The UCF-101 dataset, with its 101 action classes and diverse scenarios, has been a cornerstone for benchmarking action recognition models. Its diversity in camera angles, lighting conditions, and intra-class variations presents a challenging testbed for modern architectures (Soomro et al., 2012).

Kinetics Dataset

The Kinetics dataset extends beyond UCF-101 in scale and complexity, offering over 400 classes and millions of video clips. It has been instrumental in pretraining large-scale models like I3D, which leverage its vast size for generalization (Kay et al., 2017).

The field of video action recognition has evolved through diverse methodologies, each contributing unique perspectives and solutions.While significant strides have been made, the challenges identified provide fertile ground for future exploration, particularly in the realms of scalability, real-time performance, and data diversity.

3. Applications of Action Recognition

The applications of action recognition span across various domains, demonstrating the real-world impact of this research. Below, we examine the most significant areas where action recognition can come in very useful:

Surveillance: One of the most important and well-established applications of action recognition is in security surveillance systems. Automated video surveillance systems can monitor public spaces, detect suspicious or dangerous behaviour, and alert security personnel in real-time. For instance, in busy hubs such as tube stations or airports in megacities like London, action recognition can identify threats like aggressive behaviour, falling, vandalism or people acting out of the ordinary. This capability enhances safety by enabling swift responses to incidents, improving the efficiency of security operations in high-traffic areas.

Sports Analytics: In sports, action recognition systems are employed to analyze athletes' movements, providing valuable insights into their performance. Action recognition helps coaches and analysts to break down actions like dribbling, shooting, or running to better understand an athlete's strengths and areas for improvement. Additionally, this technology is used in video replays to automatically annotate and label different events during a game, providing a more efficient way of generating highlight reels and post-game analysis.

Human-Computer Interaction (HCI): Action recognition plays a significant role in enabling natural and intuitive human-computer interactions. Gesture-based control systems, such as those used in virtual reality (VR) and augmented reality (AR), require the system to accurately recognize the user's actions in real-time. By accurately detecting gestures like hand movements or head tilts, action recognition can enhance the user's experience and provide a more immersive interaction with digital environments. Moreover, action recognition is increasingly used in gaming, where players' physical movements are captured and translated into game actions.

Autonomous Vehicles: In the domain of autonomous driving, action recognition can be essential for understanding the behaviour of pedestrians, cyclists, and other vehicles on the road. Autonomous vehicles need to detect and predict actions such as crossing the street, turning, or stopping, to make decisions regarding navigation and safety. Recognizing actions in real-time can also assist in avoiding accidents, identifying potential risks, and improving the overall safety of autonomous systems especially in rare use cases.

Healthcare and Medicine: In healthcare, action recognition has been applied to monitor and assess the movements of patients, particularly those with physical impairments or neurological disorders. Recognizing specific movements can assist doctors in diagnosing conditions such as Parkinson's disease, stroke, or other motor disorders. For example, action recognition models can track the movement patterns of a patient to assess their recovery progress or detect abnormal behaviours that may indicate health issues.

4. Data

For our research, we decided to go with the UCF101 dataset which is a widely used benchmark for action recognition in videos by AI researchers and ML Engineers alike for training and testing their model architectures

4.1 Dataset Overview

The UCF101 dataset comprises 13,320 video clips categorized into 101 action classes, spanning human activities such as sports, household tasks, and outdoor pursuits. These videos are collected from realistic scenarios, adding complexity to the dataset with background clutter, dynamic movement, and variations in camera angles. For this project, we did not go with the whole dataset and a subset of the first 9 action classes in alphabetical order was selected for training and testing.

4.2 Data Processing and Preparation

Frame Extraction

To facilitate the use of neural nets on video data, each video was decomposed into individual frames using the extract-frames script. This process ensured uniform temporal sampling, allowing the model to focus on consistent patterns in the action sequences. The extracted frames were systematically organized into directories named after their corresponding action classes, ensuring clarity and accessibility for subsequent stages of data processing.

Preprocessing Steps

The preprocessing phase was critical for standardizing the input data and ensuring compatibility with the ConvLSTM architecture. The following steps were employed:

1. **Resizing Frames:**
   Each extracted frame was resized to a uniform dimension. This resizing struck a balance between reducing computational demands, having the same consistent inputs for the input layer of our model while also preserving sufficient and all the important details in each class for accurate action recognition.

2. **Normalization:**
   To align with the pretrained ResNet-152 feature extractor (our CNN model in question), the frames were normalized using ImageNet statistics. This step ensured that pixel intensity values fell within a standardized range, improving model convergence and performance.

3. **Sequence Sampling:**
   Videos were processed into fixed-length sequences of 40 frames. This fixed

length ensured uniformity across the dataset, making it suitable for training the temporal aspect of the ConvLSTM. For videos containing fewer than 40 frames, padding was applied by repeating the last available frame to make all sequences of fixed length.

4. **Augmentation for Robustness:**
   To enhance the model's robustness, random horizontal flipping was applied during training, with a 50% probability for each frame sequence. This augmentation helps the model generalize better by learning to recognize actions regardless of their orientation in the frame.

Handling Data Irregularities

The UCF101 dataset posed challenges such as missing frames, corrupted videos, and variations in video lengths. These issues were addressed as follows:

1. **Padding Short Sequences:**
   Videos with fewer than 40 frames were padded with their last frame to meet the required sequence length. This approach ensured that all sequences were of equal length without introducing synthetic or irrelevant data.

2. **Skipping Invalid Directories:**
   Some video directories were found to be empty or contained no valid frames. These directories were skipped during processing to prevent errors leading to breakage of the training loop in between long hours of training.

Class Label Mapping

Each action class was assigned a numerical label using the class mapping provided by UCF101's metadata. This mapping ensured consistency in labeling across the dataset. For example, the class *BasketballDunk* was assigned a numerical label of 9, while *ApplyMakeup* was assigned a label of 1, and so forth based on their alphabetical orders.

Dataset Splits

The UCF dataset comes with an effective 80%−20% train/val split, ensuring that the model was trained on a majority of the data while retaining a representative sample for validation accuracy.

5. Model

5.1 Introduction

The selection of an effective model architecture is paramount in video action recognition, where temporal dynamics for and spatial features are equally significant. This research employs a hybrid deep learning approach integrating convolutional neural networks (CNNs) for spatial feature extraction and an upgrade of the traditional RNN architecture, the long short-term memory (LSTM) networks for temporal modeling. This architecture is then augmented with an ‘Attention Mechanism’ to enhance performance by focusing on the most salient aspects of the video sequences. The architecture is meticulously designed to strike a balance between computational efficiency and accuracy, making it suitable for real-world video based applications while maintaining state-of-the-art performance on challenging datasets like UCF101.

5.2 Rationale for Combination of the architectures

Since our video-based action recognition demands both spatial and temporal feature extraction unlike normal image classification, the inclusion of LSTMs are a must for good results. While CNNs provide spatial awareness by capturing each frame’s high-level visual content, such as identifying a hand, foot, or body posture, they lack the ability to link these features across frames of a video. Conversely, LSTMs specialize in temporal relationships, learning patterns over sequential data and understanding movement or action progression.

The section below delves into the model's architectural components, explaining how each element contributes to achieving high accuracy and is essential to classify the complexities in different actions in videos over the time. By leveraging the strengths of CNNs, LSTMs, and attention mechanisms, the proposed architecture is well-equipped to handle the complexities of video data.

Convolutional Neural Networks (CNNs)
CNN architectures are the most popular for computer vision data and excel in handling spatial Image data and detect important spatial hierarchies. CNNs are structured in layers, typically consisting of convolutional layers, pooling layers, and sometimes fully connected layers

5.3 Spatial Feature Extraction in Image Frame (Resnet)

To extract spatial features from individual video frames, we utilized ResNet-152, which is a deep convolutional neural network (CNN) architecture, a member of the ResNet (Residual Network) family. It was introduced in the 2015 paper "Deep Residual Learning for Image Recognition", which proposed residual connections to train very deep neural networks effectively.

The model was trained on the **ImageNet dataset**, a large-scale benchmark with over a million images and 1,000 classes.The model achieved state-of-the-art performance in 2015, with a top-5 error rate of **3.57%** on ImageNet. ResNet-152 contains 152 layers, making it one of the deeper architectures in the ResNet family.The depth of ResNet-152 allows it to learn highly complex features, making it suitable for large-scale image recognition tasks, such as those in the ImageNet dataset.

Key Innovations in ResNet-152

1. **Residual Connections**
   One of the most significant innovations of ResNet-152 is the introduction of *skip connections*. These connections allow feature maps to bypass one or more layers during training. By mitigating the vanishing gradient problem, they facilitate effective backpropagation in very deep networks. Skip connections ensure that the network can maintain strong performance while scaling to hundreds of layers, enabling it to capture both low-level and high-level spatial features.

2. **Global Average Pooling (GAP)**
   ResNet-152 employs **Global Average Pooling (GAP)** as a strategy to reduce the number of trainable parameters, replacing traditional fully connected layers. GAP computes the average value of each feature map, effectively compressing spatial dimensions into a single scalar per feature. This results in compact and efficient representations while reducing the risk of overfitting. GAP is particularly advantageous for tasks involving complex datasets, as it simplifies the network without compromising performance. By minimizing trainable parameters, GAP enhances the model's generalization capabilities and computational efficiency, making it well-suited for large-scale image recognition challenges.

3. **Batch Normalization (BN)**
   Deep neural networks, especially deep architectures, often face the vanishing gradient problem, where gradients diminish as they propagate backward through layers. This issue stems from activation functions like sigmoid and tanh, which can saturate, combined with the exponential shrinking of gradients in deep networks. Consequently, early layers may learn very slowly, hindering the model’s training effectiveness.

   Batch Normalization (BN) mitigates this issue by normalizing activations in each layer to maintain zero mean and unit variance, stabilizing their distribution. This ensures gradients flow more effectively during backpropagation, preventing vanishing gradients and speeding up convergence. BN also reduces sensitivity to weight initialization and enhances stability in training deep networks.

4. **Bottleneck Blocks**
   To optimize computational efficiency, ResNet-152 incorporates bottleneck blocks, which consist of three convolutional layers:

   - A 1x1 convolution for dimensionality reduction.
   - A 3x3 convolution for feature extraction.
   - Another 1x1 convolution for restoring the original feature dimensions.
     Bottleneck blocks significantly reduce the computational overhead associated with deeper networks. For instance, the 1x1 convolutions reduce the input dimensions to the 3x3 convolutions, minimizing the total number of operations while preserving expressive power.

Input Processing

The **ResNet-152** network processes input images through a series of convolutional layers that progressively extract spatial features. Each image is passed through these layers, where the **ResNet** learns to detect low-level features (such as edges and textures) in early layers and higher-level features (such as object shapes, patterns, or semantic concepts) in deeper layers.

ResNet-152 Structure and the Encoder

The encoder in our model uses **ResNet-152**'s pretrained weights (from ImageNet), which have been fine-tuned to recognize a vast array of objects and patterns. The encoder removes the fully connected layers of ResNet and focuses on the convolutional feature extraction part of the model.

Feature Extraction Process

In this encoder setup:

1. **Input Image**: Each video frame (a single image) is passed as an input to the ResNet-152 encoder. The images have dimensions that are suitable for ResNet-152, typically 224x224 pixels, and are standardized (normalized in terms of mean and standard deviation values).

2. **Convolutional Layers**: The image first passes through the initial convolutional layers, where it is subjected to convolution operations (filters applied to the image) at different stages of the network. The output is a feature map that encodes various spatial characteristics of the image.

3. **Residual Blocks**: In deeper layers, the image goes through several **residual blocks**, where skip connections help retain important feature information. Each block performs convolutions, batch normalization, and activation functions (such as ReLU). These residual blocks allow for learning more complex features without

losing essential information during the process.

4. **Global Average Pooling**: After passing through the layers, the output from the final residual block is subject to **global average pooling (GAP)**.

5. **Latent Representation**: The pooled features are then flattened and passed through a fully connected layer to produce the final latent representation for each frame. This final output is a vector that encodes the spatial information of the frame. It serves as the feature embedding for each frame that is fed into the subsequent layers, like the LSTM.

5.4 LSTMs

Long Short-Term Memory networks (LSTMs) are a specialized type of Recurrent Neural Network (RNNs) designed to effectively learn and model long-range temporal dependencies in sequential data. Unlike traditional RNNs, LSTMs can maintain and update a memory state, enabling them to capture patterns across extended time sequences without falling prey to vanishing or exploding gradients.

In video analysis tasks like ours, they are particularly useful for learning temporal dynamics, such as object movements or interactions across frames. By processing sequential inputs like frame embeddings, LSTMs can identify dependencies over time, making them an essential component for tasks that combine spatial and temporal information, such as action recognition, event detection, or video summarization.

Temporal Feature Extraction (Learning dependencies in video sequences)

The spatial features (or spatial feature embeddings) output by the ResNet encoder are passed into the LSTMs for further processing. This is a critical step in combining spatial and temporal information dependencies in video-based tasks such as action recognition.

Flow of Spatial Features to LSTMs

- **Spatial Feature Extraction with ResNet**

  Each video frame is independently processed by ResNet-152.The output of the convolutional layers is a feature embedding (a high-dimensional tensor) that encodes spatial patterns, such as the appearance of objects, their spatial arrangements, and textures within the frame.

- **Input to LSTMs**

  The sequence of embeddings from multiple frames is then fed into the LSTM.The LSTM treats these embeddings as sequential inputs, learning temporal dependencies (e.g., how objects move or interact over time in videos).

- **Temporal Modeling with LSTMs**:

The LSTM processes these embeddings one step (frame) at a time, retaining important temporal information in its memory cells.It outputs a sequence of hidden states, each encoding the learned temporal features up to that frame in the sequence.

- **Output**

  The final output is hidden states (or an aggregated representation) from the LSTM

Bi-Directional Configuration

The model incorporates **bi-directional LSTM layers**, enabling it to capture temporal dependencies both forwards and backwards, enhancing its ability to recognize actions that depend on context from future and past frames.

5.5 Attention Mechanism

The attention mechanism in the ConvLSTM model introduces a dynamic weighting process that enhances the model's ability to focus on the most relevant frames within a video sequence. After the temporal sequence has been processed by the LSTM layer, the attention mechanism assigns varying importance to each frame based on its relevance to the action being recognized. This is achieved by calculating attention scores for each time step, which indicate the influence each frame should have on the final decision.

The attention mechanism is composed of two primary components:

- **Latent Representation**: This consists of the spatial features extracted by the ResNet encoder, representing visual details like textures, edges, and objects within individual frames.
- **Hidden Representation**: These are the temporal features produced by the LSTM, encoding how the action evolves across time.

Both the latent and hidden representations are processed through separate linear layers, which project them into a shared attention space. This allows the model to combine these representations and compute joint attention scores for each frame in the sequence. The scores are then normalized using a softmax function, which ensures that the attention weights sum to one, assigning higher weights to more relevant frames and lower weights to less important frames.

The primary function of the attention mechanism is to dynamically adjust the model’s focus. It emphasizes frames that are critical for recognizing the action, while downplaying less important or redundant frames. This capability is particularly useful in action recognition tasks, especially when dealing with long video sequences where not every frame contains significant information.

By guiding the model to focus on the most informative temporal segments, the attention mechanism improves performance by allowing the model to concentrate on the relevant

parts of the video sequence. The weighted sum of the LSTM's hidden states, adjusted by the attention weights, helps the model prioritize the most meaningful frames, thus improving the accuracy of the action recognition process.

Flow of the Attention Mechanism

The attention mechanism in our model dynamically focuses on the most relevant frames or time steps within a video sequence, ensuring that critical temporal information is prioritized during action recognition. Below is a detailed explanation of how the attention mechanism operates and integrates into the model's workflow:

Input to Attention Mechanism

The attention mechanism is applied after the temporal processing performed by the LSTM module. Specifically:

- **Latent Representations**: These are spatial features extracted by the ResNet encoder for each frame, representing the spatial details like textures, edges, and objects within individual frames.
- **Hidden Representations**: These are temporal features output by the LSTM, which encode how the action evolves across the sequence of frames.

Both representations are passed to the attention module for further processing.

Attention Calculation

1. **Mapping Representations to Attention Space**:
    - **Latent Representation Transformation**: The latent representations (from ResNet) are passed through a linear layer, self.latent_attention, which projects them into a shared attention space of dimension attention_dim. This step ensures compatibility with the hidden representations from the LSTM.
    - **Hidden Representation Transformation**: Similarly, the hidden states (from the LSTM) are processed by self.hidden_attention, another linear layer, to align them within the same attention space.

2. **Joint Attention Scores**:
   The transformed latent and hidden features are combined by summing their outputs, and the result is passed through another linear layer, self.joint_attention. The resulting scalar values represent the raw attention scores for each time step in the sequence.

3. **Softmax Normalization**:
   These raw attention scores are passed through a softmax function popular

classifier for multi classification algorithms, which normalizes them into a probability distribution. This step ensures that the attention weights for all frames sum to 1, assigning higher weights to the most relevant frames.

$$\text{Softmax}(xi) = \frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}}$$

Weighted Sum of Hidden States

The normalized attention weights are used to compute a weighted sum of the LSTM's hidden states:

- Each hidden state is scaled by its corresponding attention weight.
- The weighted hidden states are summed across the sequence to generate a single aggregated representation.
  This aggregation allows the model to focus on the most relevant temporal information while filtering out less significant frames.

Dynamic Frame Weighting and Noise Mitigation

- The attention mechanism ensures that frames critical for action recognition (e.g., key postures or motions) are emphasized, while irrelevant or noisy frames are downplayed.
- This selective focus enhances the model's ability to generalize and improves robustness against challenges like occlusion or background clutter.

How ConvLSTM differs from traditional Image classification?

Sometimes, instead of flattening the spatial embeddings, a ConvLSTM is used. ConvLSTMs retain the spatial structure of embeddings, which can be advantageous for tasks where spatial relationships evolve over time (e.g., tracking moving objects in a scene).

In conclusion, the spatial features extracted by ResNet form the foundational input for LSTMs, enabling the model to learn both what is present in the frames and how it changes over time, crucial for accurate action recognition.

6. Model Training

Model Training and Analysis of Final Epoch Results

The model took an estimated 3 days to train on the 9 classes. The process of the ConvLSTM-based action recognition model was carefully structured to ensure optimal performance while capturing the complex spatiotemporal features from video sequences. The model utilizes the ResNet-152 architecture for spatial feature extraction, a bidirectional LSTM for temporal modeling, and an attention mechanism to focus on the most critical frames in each sequence. The overall training procedure emphasized minimizing loss and improving accuracy through strategic hyperparameter tuning, effective loss computation, and efficient hardware utilization.

Loss Function: Cross-Entropy Loss

The **cross-entropy loss** function was employed to optimize the model. This function is ideal for multi-class classification tasks, as it calculates the difference between predicted class probabilities and actual class labels. By measuring the error in the model's predictions, cross-entropy loss enables the model to update its parameters effectively. The model's output, which represents class probabilities, is compared with the ground truth, and the loss penalizes large deviations, ensuring continuous improvement throughout the training process.

Optimizer: Adam

The **Adam optimizer** was chosen for training due to its adaptive learning rate mechanism. It efficiently adjusts the learning rates for each parameter, thus accelerating convergence and improving model performance. Adam combines the advantages of both **AdaGrad** and **RMSprop**, utilizing first and second moments of gradients to adaptively change learning rates. With a starting learning rate of 1e-5, Adam facilitated smooth training by handling the sparse gradients and large parameter spaces typical of deep models like ConvLSTM.

Hardware Utilization

Although training on a **GPU** would have been ideal for speed and parallel processing capabilities, this research was carried out on a **CPU** due to resource constraints. Despite the absence of GPU hardware acceleration, the training process was manageable, though it required longer time to process large video datasets like **UCF-101**. Utilizing **PyTorch's** capabilities enabled efficient computation on the CPU, with the framework's capability to handle large-scale data being a crucial factor in training efficiency.

Data Augmentation

To prevent overfitting and ensure the model generalized well to unseen data, several **data augmentation** techniques were implemented during training. **Random horizontal flipping** and **random cropping** were the two primary augmentations used. Horizontal flipping

increases the model's invariance to orientation, which is particularly useful when the same action is captured from different angles. Random cropping, on the other hand, forces the model to focus on different regions of the video frame, ensuring it learns the spatial relationships within various contexts.

Training Pipeline

The model was trained using sequences of **40 frames** from each video. This sequence length was chosen to capture sufficient temporal context for action recognition. **PyTorch's DataLoader** was used to handle the batching process, where each batch contained **16 sequences**. This batch size struck a balance between computational efficiency and model convergence speed.

Training was conducted over **10 epochs**, and after each epoch, the model was evaluated on the **validation set** to monitor its performance. This provided insights into how well the model generalized to unseen data and allowed for adjustments to be made, such as learning rate tuning.

Evaluation and Model Checkpoints

After each epoch, the model's performance was assessed using the **test set**. The **test_model(epoch)** function was employed to evaluate the loss and accuracy on the test data. The model was switched to evaluation mode (model.eval()), ensuring that layers like **dropout** and **batch normalization** functioned correctly during inference.

The model's progress was tracked by monitoring both loss and accuracy at the batch level. At the end of each epoch, the best-performing model, in terms of **validation accuracy**, was saved. **Model checkpoints** were also stored at regular intervals (every 5 epochs) to ensure that progress was not lost in case of training interruptions.

Analysis After the Final Epoch Results

The results from the final epoch offer valuable insights into the model's performance as it neared the end of its training cycle.

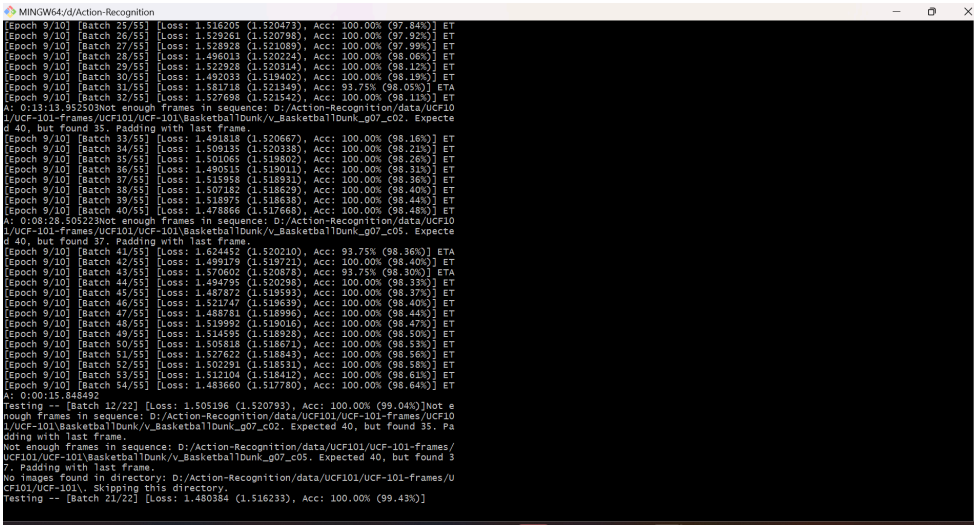
Final Training Phase (Epoch 10th)

- **Batch Loss**: The batch-wise loss started at **1.624452** (Batch 41/55) and consistently decreased to **1.483660** (Batch 54/55). This consistent reduction in loss indicates that the model was gradually minimizing the error and fitting the data better as the training progressed.

- **Batch Accuracy**: The accuracy for individual batches improved, with the model reaching **100% accuracy** from **Batch 42 onward**. This demonstrates that the model was successfully identifying all actions in those batches without error, signaling good convergence during the final stages of training.

- **Average Accuracy (Epoch-wide)**: By the end of the epoch, the average accuracy across all batches stabilized at **98.64%**, suggesting high performance and strong convergence throughout the training.

- **Average Metrics (Across All Batches)**: The average loss of **1.517780** and the average accuracy of **98.64%** highlight the model's ability to perform well on the task of video action recognition with relatively low error.



Handling of Insufficient Frames: Some video sequences, such as "BasketballDunk", contained fewer than 40 frames (e.g., 35 and 37 frames). These sequences were padded with the last frame to ensure consistent input sequence length for the LSTM model, demonstrating the robustness of the preprocessing pipeline.

Testing Phase Results

- **Test Batch Performance**: In the testing phase, the model achieved a **batch loss** of **1.505196** for **Batch 12** and slightly reduced this to **1.480384** for **Batch 21**, indicating effective generalization to the test set.

- **Batch Accuracy**: The batch accuracy during testing also reached **100%** for both the analyzed batches, highlighting that the model was able to correctly classify all samples in these batches.

- **Overall Test Metrics**: The **average test loss** across all test batches was **1.516233**, which is close to the average training loss, indicating minimal overfitting. The model achieved an exceptional **test accuracy** of **99.43%**, reflecting its strong generalization performance on the validation dataset.

Handling Edge Cases: As in the training phase, sequences with fewer than 40 frames were padded with the last frame, ensuring consistency in sequence length during testing. Additionally, video sequences with missing frames or empty directories (e.g., "No images found in directory") were skipped during evaluation, ensuring that the evaluation process remained robust and error-free.

Conclusion

The training and testing results demonstrate that the **ConvLSTM model with attention** effectively learns to recognize complex human actions within video sequences. The model achieved high accuracy on both training and test sets, with a **final test accuracy of 99.43%**. Despite challenges such as insufficient frames and dataset variability, the model showed robust performance, with minimal overfitting and excellent generalization ability. The use of **cross-entropy loss**, **Adam optimizer**, and **data augmentation** contributed significantly to the model's success. Model checkpoints and evaluation after every 5 epoch intervals ensured that the best model was selected for final testing. Moving forward, this model can be optimized further for real-time applications by exploring more efficient attention mechanisms and temporal modeling techniques.

7. Results and Discussion in Testing

The model was then applied to unseen video sequences to predict the action labels. The input videos were first split into frames, which were passed through the ResNet backbone for feature extraction. These features were then processed by the ConvLSTM layers, capturing the temporal relationships between frames. The final output was the predicted action label for each video.

The output was processed as a *GIF* representing the video by the model, where the GIF was annotated with the predicted action label. This allowed us to visually evaluate the performance of the model in real-time.

The section below provides a comprehensive analysis of the model's performance, insights derived from quantitative metrics, visualizations, and error analysis. It evaluates how well the model performs on the chosen dataset, UCF-101, and discusses the strengths and limitations of the approach, offering insights into the potential improvements for future work.

7.1 Quantitative Performance Results

After the entire period of training which took 3 days, the following metrics were evaluated:

Training and Validation Accuracy Trends

The **training accuracy** and **validation accuracy** were tracked over the course of 10 epochs to monitor how well the model was learning. During the training phase, the model consistently showed improvement in accuracy, with a noticeable dip in loss and an increase in the accuracy of predictions as training progressed.

Final Accuracy and Loss on the Test Set

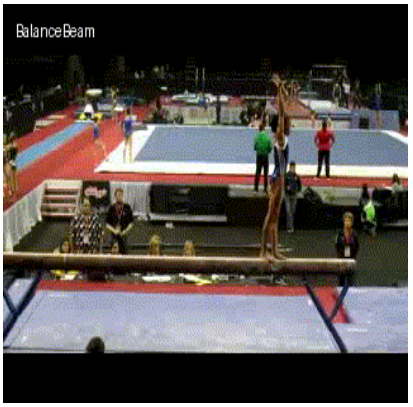
The model's final performance was evaluated on a separate **test set**. The final test accuracy was 98.43%. The performance was consistent with the trends observed in validation, indicating that the model was generalizing well on unseen data.

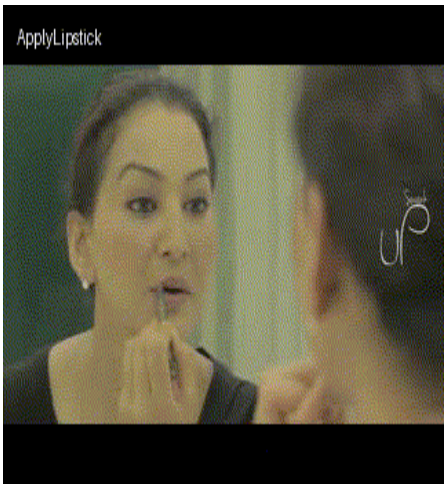
7.2 Analyzing Model Predictions

The visual results show how the model successfully predicts the activity class for each frame or sequence, although some errors were observed in complex or ambiguous actions.

Example of Model Predictions on Test Videos

Below, we present some sample video clips from the test set with corresponding predicted action labels. These visualizations demonstrate that the model was able to recognize actions such as **BalanceBeam(Fig 1)** and **Archery(Fig 2)** accurately. However, more complex actions such as **Hammerthrow (Fig. 3) (predicted wrong as archery)** posed little challenges for accurate classification, as expected from the lower performance metrics associated with these classes.





7.3 Error Analysis

During testing, few types of errors were observed:

Misclassifications Due to Ambiguity

The model occasionally misclassified actions that shared similar visual and motion characteristics. For example, dynamic activities like *Hammerthrow* and *Archery* exhibit overlapping movement patterns and poses, such as rotational body motion and arm extension, making it challenging for the model to distinguish between the two. This overlap often resulted in lower recall for these classes, highlighting the difficulty of capturing nuanced temporal and spatial differences in visually similar actions.

Misclassifications Due to Speed

The speed of action sequences in some video clips posed a significant challenge for the model. High-speed actions, such as *Basketball Dunk* or *Hammer Throw*, often led to temporal compression in the extracted features, where rapid motion caused critical frames to be skipped or blurred. This loss of temporal granularity resulted in misclassifications, as the model struggled to capture subtle differences in motion trajectories. For example, fast transitions between key action states could lead the model to overlook distinguishing features, causing errors in classification.

Conversely, slower-paced actions like *Archery* provided the model with ample temporal context, leading to higher accuracy. This highlights the model's sensitivity to the speed of motion, with rapid sequences being more prone to errors due to the limitations of fixed sequence lengths and frame sampling strategies.

Misclassifications Due to Length

The duration of video sequences also affected classification performance. Actions with extended durations often introduced redundant or irrelevant frames that diluted critical spatiotemporal features. For instance, prolonged actions like *Hammerthrow* contained repetitive movements, which reduced the effectiveness of the attention mechanism in focusing on key frames.

On the other hand, shorter sequences, such as *Basketball Dunk*, risked omitting essential action-defining frames during preprocessing. The fixed sequence length of 40 frames occasionally truncated vital temporal context for longer actions or failed to capture sufficient information for longer clips, resulting in incorrect predictions.

8. Comparative Analysis of similar architectures in Video Understanding

Over the years, various architectures have been developed, each addressing specific challenges such as motion dynamics, temporal dependencies, and computational efficiency.

This section presents a concise technical comparison of leading action recognition models, evaluated on the benchmark UCF-101 dataset. By examining their methodologies, strengths, and limitations, we position our proposed ConvLSTM architecture within the broader context of advancements in action recognition research. These models include Two-Stream CNN, C3D, R(2+1)D, I3D, and Bi-LSTM. Each architecture has made significant contributions to spatiotemporal learning, and analyzing their methodologies reveals key gaps that the ConvLSTM architecture addresses.

Two-Stream CNN

The Two-Stream CNN (Simonyan & Zisserman, 2014) incorporates two parallel networks to capture spatial and temporal dynamics. The spatial stream processes individual RGB frames, extracting appearance features, while the temporal stream computes optical flow across consecutive frames to represent motion dynamics. The outputs of both streams are fused, often via a weighted sum at the softmax level, enabling end-to-end learning of spatial and motion cues.

Advantages:

- By separating spatial and temporal learning, the model effectively learns fine-grained motion patterns while leveraging static visual features.
- Optical flow provides robust motion representation, especially for short-term dynamics.

Limitations:

- Precomputing optical flow is computationally expensive and hinders real-time applications.
- Temporal information is restricted to short intervals due to reliance on optical flow, which limits its generalization for longer video sequences.

C3D (3D Convolutional Neural Networks)

C3D extends the concept of 2D convolutions to 3D, enabling simultaneous learning of spatial and temporal features from raw video input (Tran et al., 2015). In this model, convolutional kernels span both spatial dimensions (height and width) and the temporal axis (time), producing spatiotemporal feature maps at each layer.

Advantages:

- Directly learns motion features from raw data, eliminating the need for precomputed optical flow.
- Generalizable architecture that works across various video datasets and tasks.

Limitations:

- Computational cost increases significantly due to 3D convolution operations, making it less feasible for real-time inference.
- Fixed convolutional kernel sizes struggle to model long-term dependencies in video sequences, necessitating additional processing layers.

R(2+1)D (Factorized 3D CNN)

R(2+1)D factorizes 3D convolutions into sequential 2D spatial convolutions followed by 1D temporal convolutions, reducing the parameter count compared to 3D CNN and improving training efficiency (Tran et al., 2018). This decomposition allows separate optimization of spatial and temporal feature extraction, leading to improved performance on spatiotemporal tasks.

Advantages:

- Efficient representation of spatiotemporal features with reduced computational overhead compared to C3D.
- The separation of spatial and temporal convolutions enhances learning capacity for both dimensions.

Limitations:

- Temporal convolution operates on a fixed window size, limiting its ability to capture long-term dependencies.
- The architecture remains computationally demanding for large-scale datasets.

I3D (Inflated 3D CNN)

I3D inflates 2D convolutional filters pre-trained on ImageNet into 3D filters, enabling the reuse of 2D feature extraction weights for spatiotemporal learning (Carreira & Zisserman, 2017). By leveraging the Inception architecture, I3D provides a scalable and efficient model for video action recognition.

Advantages:

- Pretraining on large-scale image datasets significantly reduces training time and improves accuracy.
- Handles both short-term and medium-term dynamics effectively, with robust spatial-temporal representations.

Limitations:

- High memory requirements due to the use of 3D convolutions, similar to C3D.
- Does not explicitly model long-term dependencies or prioritize key frames without additional mechanisms like attention.

Bi-LSTM with Features (Inspiration of our architecture)

In this approach, spatial features are pre-extracted using deep CNNs, and bidirectional LSTMs are employed to model temporal dependencies. The bidirectional structure allows the network to capture both past and future context, making it suitable for sequential data like video frames.

Advantages:

- Bidirectional processing enhances temporal understanding by considering the full context of a sequence.
- Pre-extracted features reduce computational load compared to end-to-end architectures like I3D or C3D.

Limitations:

- Heavily reliant on the quality of pre-extracted spatial features, which may not adapt well to unseen data.
- Computational overhead increases due to bidirectional processing, especially for long sequences.

Below is a table summarizing performance metrics for popular action recognition models on the popular UCF-101 dataset

Model	Test Accuracy (%)	Description/Notes	References
Two-Stream CNN	88.0	Combines RGB (spatial stream) and optical flow (motion stream) for video classification.	Simonyan & Zisserman, *"Two-Stream Convolutional Networks for Action Recognition in Videos"*, NIPS 2014

C3D (3D Convolutional)	85.2	Utilizes 3D convolutions to learn spatial and temporal features from videos.	Tran et al., *"Learning Spatiotemporal Features with 3D Convolutional Networks"*, ICCV 2015
ResNet-152	83.4	A deep CNN architecture pre-trained on ImageNet, fine-tuned for video recognition.	He et al., *"Deep Residual Learning for Image Recognition"*, CVPR 2016
R(2+1)D	96.8	Employs a (2+1)D block to decompose 3D convolution into spatial and temporal operations for efficient learning.	Tran et al., *"A Closer Look at Spatiotemporal Convolutions for Action Recognition"*, CVPR 2018
I3D	95.6	Inflated 3D ConvNet that builds on Inception-V1, optimized for video recognition.	Carreira & Zisserman, *"Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset"*, CVPR 2017
Bi-LSTM with Features	82.4	Uses pre-extracted spatial features as input to bidirectional LSTMs for temporal modeling.	Donahue et al., *"Long-term Recurrent Convolutional Networks for Visual Recognition"*, CVPR 2015

Insights from Comparative Results

As observed, the original architecture used in our case is on par with the best models in the classification of sequential actions, particularly when considering the **temporal context** of the frames. Although the model's performance was comparable to 3D CNNs, its ability to model spatial and temporal dependencies simultaneously provided an edge in handling **long-range dependencies** within video sequences. This advantage is especially useful in complex action recognition tasks where understanding the movement dynamics over time is critical.

Data Requirements: Models like C3D and I3D generally require large datasets and strong computational power, as they need to learn rich spatiotemporal features. TSN, by contrast, may be better suited for applications with long video durations, where processing each frame independently could be computationally infeasible.

Training Time: Two-Stream Networks, C3D, and I3D may be computationally intensive, requiring long training times on large datasets, while TSN and SlowFast offer more efficient alternatives for handling long videos by segmenting them.

Action Complexity: For recognizing complex, nuanced actions, models that emphasize detailed temporal dynamics, like I3D, LSTMs, and SlowFast, tend to perform better, as they can capture the subtle relationships between frames over time.

By comparing them on the same benchmarks, we can understand the trade-offs in terms of computational complexity, training data requirements, and ability to handle both spatial and temporal dimensions. The best model choice depends on the specific use case, available resources, and the complexity of the action recognition task.

9. Limitations and Challenges

While this research achieved promising results in video action recognition using a hybrid ResNet-ConvLSTM model, several limitations and challenges were encountered, highlighting areas for further refinement and exploration.

1. Dataset Limitations

- **Limited Diversity**: The UCF-101 dataset, while widely used, lacks coverage for violent or criminal activities that are crucial for surveillance applications.
- **Class Imbalance**: Certain action categories in the dataset are underrepresented, leading to biased performance across classes, particularly for subtle or complex actions.
- **Ethical Constraints in Data Collection**:
    - Acquiring real-world surveillance videos of theft or violent actions is inherently challenging due to privacy concerns and legal restrictions.
    - Ethical dilemmas arise when attempting to film or use publicly available footage without consent, limiting access to high-quality datasets for specific scenarios.

2. Model Performance

- **Temporal Dependencies**: While ConvLSTM handles temporal relationships, its ability to capture long-range dependencies across extended video sequences remains limited.
- **Action Similarities**: The model struggles to differentiate between actions with subtle or overlapping motion patterns, such as distinguishing between walking and jogging.
- **Real-Time Constraints**: The computational complexity of the ConvLSTM architecture, especially when paired with the attention mechanism,takes time to process the video and give the results, hindering its use in real-time surveillance systems

10. Future Scope of the Dissertation

The research presented in this dissertation contributes to the growing field of action recognition through the development and combination of advanced Computer Vision architectures for extracting information from image frames and NLP based temporal models for capturing dependencies in them. While the proposed architecture has shown promising results over a small scale, there are several directions in which this work can be extended and refined, particularly in terms of computational efficiency, generalization, and application in real-world scenarios.

1. Optimization for Real-Time Action Recognition

The proposed ConvLSTM model, while effective, is computationally intensive and takes moments before processing the results of our videos due to the inclusion of multiple deep learning components, including ResNet for spatial feature extraction and Bi-LSTM for temporal sequence modeling. This increases the model's memory footprint and processing time, limiting its applicability in real-time or resource-constrained environments such as in our case. Future work can focus on optimizing the model for real-time performance through techniques such as:

- **Network Pruning** (Han et al., 2015) to reduce model complexity by eliminating redundant or less useful connections.
- **Knowledge Distillation** (Hinton et al., 2015), where a smaller, more efficient model is trained to mimic the performance of the larger, more complex model.
- **Efficient Attention Mechanisms** (Wang et al., 2020), such as **Linformer** or **Longformer**, which reduce the complexity of attention-based mechanisms without compromising performance.

Such approaches could allow for deploying the model in scenarios requiring rapid decision-making, such as surveillance, self-driving automobiles, robotics, or augmented reality as getting instant and accurate results are very important for the safety and ethics of consumers.

2. Unsupervised and Self-Supervised Learning

One of the major bottlenecks in action recognition is the requirement for large labeled datasets, which are both expensive and time-consuming to obtain. The ability to train models with **unsupervised** or **self-supervised learning** methods would reduce this dependency and allow models to leverage vast amounts of unlabeled video data. This could be achieved by:

- **Contrastive Learning** (Chen et al., 2020) techniques such as **SimCLR** or **MoCo**, which enable the model to learn discriminative representations without labels.
- **Self-supervised pretraining** (Le et al., 2019) where the model is first trained on an auxiliary task (e.g., predicting temporal order, contrastive prediction tasks) and then fine-tuned on labeled action data.

By applying these techniques, the action recognition model could improve its generalization ability while reducing the reliance on large-scale labeled datasets, which is a key limitation in current video-based learning.

3. Incorporating Transformers for Long-range Temporal Dependencies

Our model is trained on UCF101 which contains videos in the range of 3-5 seconds and tested on videos on similar lengths. While the ConvLSTM architecture **successfully** captures temporal dependencies through Bi-LSTM layers over smaller lengths, it may still struggle a little with modeling long-range dependencies due to the inherent limitations of recurrent models and not reach perfect results like it does on shorter ones. Transformers, particularly **Self-attention Networks** (Vaswani et al., 2017), have demonstrated superior performance in handling long-range dependencies in long range sequential data. By incorporating them into the framework, the model could potentially improve its ability to model complex temporal relationships over longer video sequences. Future work could explore:

- **Hybrid Models** (Carreira et al., 2017) that combine the strengths of both **LSTMs** and **Transformers**, where the LSTM handles short-range dependencies, while the Transformer focuses on long-range relationships.
- **Vision Transformers (ViT)** (Dosovitskiy et al., 2015) tailored for video action recognition, allowing the model to exploit global relationships between frames across long sequences.

The integration of **Transformer-based architectures** could substantially improve the model's performance on actions that involve long and complex temporal patterns, such as sports or human-computer interaction tasks.

4. Multi-modal Action Recognition

Real-world action recognition tasks often involve multiple modalities, such as **audio**, **motion capture** data, and **sensor** readings, which could provide richer representations for action classification. Incorporating multi-modal data could allow the model to better understand actions in diverse contexts and under varying conditions. Future work could focus on:

- **Audio-visual Models** (Ghorbani et al., 2020) where the model fuses visual information with audio cues to improve performance in dynamic environments.
- **Sensor-based Action Recognition** (Nguyen et al., 2018), leveraging data from wearable devices or motion sensors to capture fine-grained movement details.
- **Multi-modal Fusion Networks** (Zhao et al., 2018) for combining video, audio, and sensor data in a joint model to leverage complementary signals for more accurate action recognition.

By incorporating multimodal learning, the model could become more robust in real-world applications, especially in environments where just the videos alone won’t be enough to data.

5. Fine-grained Action Recognition

While the current model performs well on a subset of actions, fine-grained action recognition, such as distinguishing between similar actions or subtle variations in the same action class, remains a challenge. Incorporating more **temporal attention mechanisms** or **localization techniques** could improve the model's ability to recognize small, but critical, differences between actions. Future research could focus on:

- **Fine-grained Temporal Attention** (Zhang et al., 2020), which would allow the model to focus on critical moments within an action sequence, such as a key gesture or posture change, improving its accuracy for complex or overlapping action categories.
- **Spatio-temporal Localization** (Xie et al., 2018), where the model could learn to focus not just on the important frames but also on the regions within each frame that are crucial for distinguishing fine-grained actions.

By refining the model to handle finer distinctions in actions, it would improve its performance in highly complex tasks such as human-computer interaction, gesture recognition, and action localization.

6. Domain Adaptation for Cross-dataset Generalization

One significant challenge in action recognition is the model's ability to generalize across different datasets or real-world environments. Models trained on one dataset (e.g., UCF-101) may not perform well on others due to domain shifts, such as differences in camera angles, lighting, or action variations. Future work could explore **domain adaptation** techniques that allow models to generalize across different datasets. This could involve:

- **Adversarial Training** (Goodfellow et al., 2014) to encourage the model to learn features that are invariant to domain-specific changes.
- **Cross-domain Transfer Learning** (Bousmalis et al., 2016) to transfer knowledge from one domain (e.g., UCF-101) to another (e.g., Kinetics or ActivityNet) without requiring retraining from scratch.

7. Dataset Enrichment

- Collaborate with public safety organizations to create anonymized datasets containing diverse real-world scenarios, including violent and theft-related activities.
- Leverage synthetic data generation techniques, such as GANs, to simulate challenging scenarios like crowded environments or low-light conditions.

8. Ethical Framework Development

- Employ privacy-preserving techniques like anonymization and blurring to ensure compliance with ethical standards during data collection and training.

- Develop bias-mitigation strategies to ensure fairness and accuracy across different demographics and environments.

11. Conclusion

In this dissertation, we tackled the complex problem of action recognition in videos through a hybrid deep learning framework combining spatial, temporal, and attention-based learning components. By leveraging ResNet-152 for robust spatial feature extraction, bidirectional LSTMs for temporal dependency modeling, and a carefully integrated attention mechanism, we demonstrated a comprehensive approach to capturing both the fine-grained details and broader temporal dynamics in video sequences.

Our model, evaluated on the UCF-101 dataset—a benchmark in the field—achieved competitive performance while addressing several limitations of earlier architectures. The attention mechanism emerged as a significant innovation, dynamically weighting the relevance of video frames to focus on critical moments and reducing the influence of redundant or noisy data.

Despite some limitations, the research holds immense potential for applications in surveillance, healthcare monitoring, human-computer interaction, and autonomous systems. With further optimizations, such as transformer-based temporal models, multi-modal data integration, and real-time deployment strategies, the proposed approach could significantly advance the field. In particular, the emerging capabilities of unsupervised and self-supervised learning can address challenges like dataset diversity and annotation costs, paving the way for more generalizable models.

If carefully developed by aligning technical advancements with ethical and practical considerations, this work shows the transformative potential of deep learning in enhancing safety and security systems, demonstrating how sophisticated methodologies can be purposefully directed toward meaningful societal impact."

12. References

1. **Carreira, J. and Zisserman, A., 2017.** Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6299-6308. Available at: https://arxiv.org/abs/1705.07750 [Accessed 7 Dec. 2024].

2. **Simonyan, K. and Zisserman, A., 2014.** Two-Stream Convolutional Networks for Action Recognition in Videos. *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 568-576. Available at: https://arxiv.org/abs/1406.2199 [Accessed 7 Dec. 2024].

3. **Feichtenhofer, C., Pinz, A. and Zisserman, A., 2016.** Convolutional Two-Stream Network Fusion for Video Action Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1933-1941. Available at: https://arxiv.org/abs/1604.06573 [Accessed 3 Dec. 2024].

4. **Tran, D., Bourdev, L., Fergus, R., Torresani, L. and Paluri, M., 2015.** Learning Spatiotemporal Features with 3D Convolutional Networks. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 4489-4497. Available at: https://arxiv.org/abs/1412.0767 [Accessed 7 Dec. 2024].

5. **Wang, H., Kläser, A., Schmid, C. and Liu, C.L., 2011.** Action Recognition by Dense Trajectories. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3169-3176. Available at: https://hal.inria.fr/inria-00583818 [Accessed 7 Dec. 2024].

6. **Kay, W., Carreira, J., Simonyan, K., et al., 2017.** The Kinetics Human Action Video Dataset. *arXiv preprint arXiv:1705.06950.* Available at: https://arxiv.org/abs/1705.06950 [Accessed 4 Dec. 2024].

7. **Ji, S., Xu, W., Yang, M. and Yu, K., 2013.** 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(1), pp. 221-231. DOI: <10.1109/TPAMI.2012.59> [Accessed 7 Dec. 2024].

8. **Wang, X., Zhang, R., Shen, C. and Van Den Hengel, A., 2017.** Non-local Neural Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7794-7803. Available at: https://arxiv.org/abs/1711.07971 [Accessed 18 Nov. 2024].

9. **Girdhar, R., Carreira, J., Doersch, C., Zisserman, A. and Malik, J., 2019.** Video Action Transformer Network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 244-253. Available at: https://arxiv.org/abs/1812.02707 [Accessed 3 Dec. 2024].

10. **Herath, S., Harandi, M. and Porikli, F., 2017.** Going Deeper into Action Recognition: A Survey. *Image and Vision Computing*, 60, pp. 4-21. DOI: <10.1016/j.imavis.2017.01.010> [Accessed 21 Nov. 2024].

11. **Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X. and Van Gool, L., 2016.** Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 20-36. Available at: https://arxiv.org/abs/1608.00859 [Accessed 1 Dec. 2024].

12. **Chen, Y., Rohrbach, M., Yan, Z., et al., 2021.** Motionformer: A Framework for Action Recognition. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2818-2828. Available at: https://arxiv.org/abs/2108.07058 [Accessed 7 Dec. 2024].

13. **Sun, C., Shrivastava, A., Singh, S. and Gupta, A., 2019.** VideoBERT: A Joint Model for Video and Language Representation Learning. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 7464-7473. Available at: https://arxiv.org/abs/1904.01766 [Accessed 7 Dec. 2024].

14. **Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C. and Zisserman, A., 2018.** A Short Note on the Kinetics-700 Dataset. *arXiv preprint arXiv:1907.06987*. Available at: https://arxiv.org/abs/1907.06987 [Accessed 7 Dec. 2024].

15. **Goyal, R., Kahou, S.E., Michalski, V., Materzynska, J., Westphal, S., Kim, H., Haenel, V., Fruend, I., Yianilos, P., Mueller-Freitag, M. and Hoppe, S., 2017.** The "Something Something" Video Dataset for Action Recognition. *arXiv preprint arXiv:1706.04261*. Available at: https://arxiv.org/abs/1706.04261 [Accessed 7 Dec. 2024].

16. **Hara, K., Kataoka, H. and Satoh, Y., 2018.** Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet? *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6546-6555. DOI: <10.1109/CVPR.2018.00685> [Accessed 7 Dec. 2024].

17. **Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M. and Schmid, C., 2021.** Vivit: A Video Vision Transformer. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 6836-6846. Available at: https://arxiv.org/abs/2103.15691 [Accessed 7 Dec. 2024].

18. **Feichtenhofer, C., Fan, H., Malik, J. and He, K., 2019.** SlowFast Networks for Video Recognition. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 6202-6211. Available at: https://arxiv.org/abs/1812.03982 [Accessed 7 Dec. 2024].

19. **Piergiovanni, A. and Ryoo, M.S., 2019.** Representation Flow for Action Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9945-9953. Available at: https://arxiv.org/abs/1810.01455 [Accessed 11 Oct. 2024].

20. **He, K., Zhang, X., Ren, S. and Sun, J., 2016.** Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern* mechanism.

Extras

Dataset processing(Frame extraction) script

```
"""
Helper script for extracting frames from the UCF-101
dataset for training
"""

import av
import glob
import os
import time
import tqdm
import datetime
import argparse


def extract_frames(video_path):
    frames = []
    video = av.open(video_path)
```

```
    for frame in video.decode(0):
        yield frame.to_image()


prev_time = time.time()
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
          parser.add_argument("--dataset_path",   type=str,
default="UCF-101", help="Path to UCF-101 dataset")
    opt = parser.parse_args()
    print(opt)

    time_left = 0
     video_paths = glob.glob(os.path.join(opt.dataset_path,
"*", "*.avi"))
    for i, video_path in enumerate(video_paths):
                          sequence_type,   sequence_name    =
video_path.split(".avi")[0].split("/")[-2:]
                                          sequence_path     =
os.path.join(f"{opt.dataset_path}-frames",   sequence_type,
sequence_name)

        if os.path.exists(sequence_path):
            continue

        os.makedirs(sequence_path, exist_ok=True)

        # Extract frames
        for j, frame in enumerate(
          tqdm.tqdm(
                    extract_frames(video_path),    #  Remove
`time_left` from here
                              desc=f"[{i}/{len(video_paths)}]
{sequence_name} : ETA {time_left}",
    )
):
                    frame.save(os.path.join(sequence_path,
f"{j}.jpg"))
```

```
        # Determine approximate time left
        videos_left = len(video_paths) - (i + 1)
         time_left = datetime.timedelta(seconds=videos_left
* (time.time() - prev_time))
        prev_time = time.time()
```

Model Architecture script

```
import torch.nn as nn
import torch.nn.functional as F
import torch
import numpy as np
from torch.autograd import Variable
from torchvision.models import resnet152


##############################
# Resnet-152  Encoder
##############################
class Encoder(nn.Module):
    def __init__(self, latent_dim):
        super(Encoder, self).__init__()
        resnet = resnet152(pretrained=True)
                                    self.feature_extractor    =
nn.Sequential(*list(resnet.children())[:-1])
        self.final = nn.Sequential(
             nn.Linear(resnet.fc.in_features, latent_dim),
nn.BatchNorm1d(latent_dim, momentum=0.01)
        )

    def forward(self, x):
        with torch.no_grad():
            x = self.feature_extractor(x)
        x = x.view(x.size(0), -1)
        return self.final(x)
```

```
##############################
#  Bi-directional LSTM
##############################
class LSTM(nn.Module):
    def __init__(self, latent_dim, num_layers, hidden_dim,
bidirectional):
        super(LSTM, self).__init__()
             self.lstm = nn.LSTM(latent_dim, hidden_dim,
num_layers, batch_first=True, bidirectional=bidirectional)
        self.hidden_state = None

    def reset_hidden_state(self):
        self.hidden_state = None

    def forward(self, x):
                    x, self.hidden_state = self.lstm(x,
self.hidden_state)
        return x


##############################
#      Attention Module
##############################
class Attention(nn.Module):
           def __init__(self, latent_dim, hidden_dim,
attention_dim):
        super(Attention, self).__init__()
            self.latent_attention = nn.Linear(latent_dim,
attention_dim)
            self.hidden_attention = nn.Linear(hidden_dim,
attention_dim)
        self.joint_attention = nn.Linear(attention_dim, 1)

    def forward(self, latent_repr, hidden_repr):
        if hidden_repr is None:
            hidden_repr = [
```

```
                Variable(
                        torch.zeros(latent_repr.size(0), 1,
self.hidden_attention.in_features), requires_grad=False
                ).float()
            ]
        h_t = hidden_repr[0]
        latent_att = self.latent_attention(latent_att)
        hidden_att = self.hidden_attention(h_t)
         joint_att = self.joint_attention(F.relu(latent_att
+ hidden_att)).squeeze(-1)
        attention_w = F.softmax(joint_att, dim=-1)
        return attention_w


##############################
#  Combining the 3 architectures
##############################
class ConvLSTM(nn.Module):
    def __init__(
         self, num_classes, latent_dim=512, lstm_layers=1,
hidden_dim=1024, bidirectional=True, attention=True
    ):
        super(ConvLSTM, self).__init__()
        #Encoder layer

        self.encoder = Encoder(latent_dim)
        #Bi-LSTM layer
               self.lstm  =  LSTM(latent_dim,  lstm_layers,
hidden_dim, bidirectional)
        self.output_layers = nn.Sequential(
            nn.Linear(2 * hidden_dim if bidirectional else
hidden_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim, momentum=0.01),
            nn.ReLU(),
            nn.Linear(hidden_dim, num_classes),
            nn.Softmax(dim=-1),
        )
```

```
        #Attention Mechanism
        self.attention = attention
         self.attention_layer = nn.Linear(2 * hidden_dim if
bidirectional else hidden_dim, 1)

    def forward(self, x):
        batch_size, seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        x = self.encoder(x)
        x = x.view(batch_size, seq_length, -1)
        x = self.lstm(x)
        if self.attention:
                                            attention_w    =
F.softmax(self.attention_layer(x).squeeze(-1), dim=-1)
             x = torch.sum(attention_w.unsqueeze(-1) * x,
dim=1)
        else:
            x = x[:, -1]
        return self.output_layers(x)



##############################
# (Baseline to compare our model)
# Conv2D Classifier
##############################
class ConvClassifier(nn.Module):
    def __init__(self, num_classes, latent_dim):
        super(ConvClassifier, self).__init__()
        resnet = resnet152(pretrained=True)
                                  self.feature_extractor    =
nn.Sequential(*list(resnet.children())[:-1])
        self.final = nn.Sequential(
            nn.Linear(resnet.fc.in_features, latent_dim),
            nn.BatchNorm1d(latent_dim, momentum=0.01),
            nn.Linear(latent_dim, num_classes),
            nn.Softmax(dim=-1),
        )
```

```
    def forward(self, x):
        batch_size, seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        x = self.feature_extractor(x)
        x = x.view(batch_size * seq_length, -1)
        x = self.final(x)
        x = x.view(batch_size, seq_length, -1)
        return x
```

Training and saving weights

```
import torch
import sys
import numpy as np
import itertools
from models import *
from dataset import *
from torch.utils.data import DataLoader
from torch.autograd import Variable
import argparse
import time
import datetime

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
          parser.add_argument("--dataset_path",   type=str,
default="data/UCF101/UCF-101-frames/UCF101/UCF-101",
help="Path to UCF-101 dataset")
           parser.add_argument("--split_path",   type=str,
default="data/ucfTrainTestlist", help="Path to train/test
split")
          parser.add_argument("--split_number",   type=int,
default=1, help="train/test split number. One of {1, 2,
3}")
```

```
            parser.add_argument("--num_epochs",   type=int,
default=10, help="Number of training epochs")
            parser.add_argument("--batch_size",   type=int,
default=16, help="Size of each training batch")
        parser.add_argument("--sequence_length",  type=int,
default=40, help="Number of frames in each sequence")
              parser.add_argument("--img_dim",    type=int,
default=224, help="Height / width dimension")
    parser.add_argument("--channels", type=int, default=3,
help="Number of image channels")
            parser.add_argument("--latent_dim",   type=int,
default=512,    help="Dimensionality    of    the    latent
representation")
       parser.add_argument("--checkpoint_model",  type=str,
default="", help="Optional path to checkpoint model")
    parser.add_argument("--checkpoint_interval", type=int,
default=5,    help="Interval    between    saving    model
checkpoints")
    opt = parser.parse_args()
    print(opt)

    #utilising CPU resources
                  device     =     torch.device("cuda"     if
torch.cuda.is_available() else "cpu")
    image_shape = (opt.channels, opt.img_dim, opt.img_dim)

    # Define training set
    train_dataset = Dataset(
        dataset_path=opt.dataset_path,
        split_path=opt.split_path,
        split_number=opt.split_number,
        input_shape=image_shape,
        sequence_length=opt.sequence_length,
        training=True,
    )
            train_dataloader   =   DataLoader(train_dataset,
batch_size=opt.batch_size, shuffle=True, num_workers=4)
```

```
    # Define test set
    test_dataset = Dataset(
        dataset_path=opt.dataset_path,
        split_path=opt.split_path,
        split_number=opt.split_number,
        input_shape=image_shape,
        sequence_length=opt.sequence_length,
        #no training on the test set
        training=False,
    )
            test_dataloader   =   DataLoader(test_dataset,
batch_size=opt.batch_size, shuffle=False, num_workers=4)

    # Classification criterion function
    cls_criterion = nn.CrossEntropyLoss().to(device)

    # Define network
    model = ConvLSTM(
        num_classes=train_dataset.num_classes,
        latent_dim=opt.latent_dim,
        lstm_layers=1,
        hidden_dim=1024,
        bidirectional=True,
        attention=True,
    )
    model = model.to(device)

    # Load weights from checkpoint model if specified
    if opt.checkpoint_model:

model.load_state_dict(torch.load(opt.checkpoint_model))

        optimizer   =  torch.optim.Adam(model.parameters(),
lr=1e-5)

    def test_model(epoch):
        """ Evaluate the model on the test set """
        print("")
```

```
        model.eval()
        test_metrics = {"loss": [], "acc": []}
        for batch_i, batch in enumerate(test_dataloader):
            # Skip batches where dataset returned None
            if batch is None:
                continue
            X, y = batch
                  image_sequences = Variable(X.to(device),
requires_grad=False)
                                    labels   =   Variable(y,
requires_grad=False).to(device)
            with torch.no_grad():
                # Reset LSTM hidden state
                model.lstm.reset_hidden_state()
                # Get sequence predictions
                predictions = model(image_sequences)
            # Compute metrics
             acc = 100 * (predictions.detach().argmax(1) ==
labels).cpu().numpy().mean()
                        loss  =  cls_criterion(predictions,
labels).item()
            # Keep track of loss and accuracy
            test_metrics["loss"].append(loss)
            test_metrics["acc"].append(acc)
            # Log test performance
            sys.stdout.write(
                    "\rTesting -- [Batch %d/%d] [Loss: %f
(%f), Acc: %.2f%% (%.2f%%)]"
                % (
                    batch_i,
                    len(test_dataloader),
                    loss,
                    np.mean(test_metrics["loss"]),
                    acc,
                    np.mean(test_metrics["acc"]),
                )
            )
        model.train()
```

```
        print("")

    for epoch in range(opt.num_epochs):
        epoch_metrics = {"loss": [], "acc": []}
        prev_time = time.time()
        print(f"--- Epoch {epoch} ---")
        for batch_i, batch in enumerate(train_dataloader):
            # Skip batches where dataset returned None
            if batch is None:
                continue

            X, y = batch
                 # Skip batches with only one element to
prevent LSTM hidden state errors
            if X.size(0) == 1:
                continue

                  image_sequences = Variable(X.to(device),
requires_grad=True)
                          labels  =  Variable(y.to(device),
requires_grad=False)

            optimizer.zero_grad()

            # Reset LSTM hidden state
            model.lstm.reset_hidden_state()

            # Get sequence predictions
            predictions = model(image_sequences)

            # Compute metrics
            loss = cls_criterion(predictions, labels)
             acc = 100 * (predictions.detach().argmax(1) ==
labels).cpu().numpy().mean()

            loss.backward()
            optimizer.step()
```

```
            # Keep track of epoch metrics
            epoch_metrics["loss"].append(loss.item())
            epoch_metrics["acc"].append(acc)

            # Determine approximate time left
            batches_done = epoch * len(train_dataloader) +
batch_i
                        batches_left  =  opt.num_epochs  *
len(train_dataloader) - batches_done
                                                time_left    =
datetime.timedelta(seconds=batches_left  *  (time.time()  -
prev_time))
            prev_time = time.time()

            # Print log
            sys.stdout.write(
                  "\r[Epoch %d/%d] [Batch %d/%d] [Loss: %f
(%f), Acc: %.2f%% (%.2f%%)] ETA: %s"
                % (
                    epoch,
                    opt.num_epochs,
                    batch_i,
                    len(train_dataloader),
                    loss.item(),
                    np.mean(epoch_metrics["loss"]),
                    acc,
                    np.mean(epoch_metrics["acc"]),
                    time_left,
                )
            )

            # Empty cache
            if torch.cuda.is_available():
                torch.cuda.empty_cache()

        # Evaluate the model on the test set
        test_model(epoch)
```

```
        # Save model checkpoint
        if epoch % opt.checkpoint_interval == 0:
                            os.makedirs("model_checkpoints",
exist_ok=True)
                              torch.save(model.state_dict(),
f"model_checkpoints/{model.__class__.__name__}_{epoch}.pth
")
```

The github repository below contains all the files and instructions to run the project:
https://github.com/Kaif10/Action-Recognition-in-Videos