Action Recognition from Sport Videos using Multimodal Transfer Learning

A Report Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology

in

Computer Science & Engineering

by

Shreyansh Verma (20223257),
Sirra Veera Pratap (20223263),
Vinayak Tripathi (20223312),
Tavis Mariageorge James (20223566),
Subham Seth (20223270)

to the

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD PRAYAGRAJ

November, 2025

UNDERTAKING

We declare that the work presented in this report titled "*Action Recognition from Sport Videos using Multimodal Transfer Learning*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, we accept that our degree may be unconditionally withdrawn.

$18th November$, 2025
Allahabad

Shreyansh Verma (20223257),
Sirra Veera Pratap (20223263),
Vinayak Tripathi (20223312),
Tavis Mariageorge James (20223566),
Subham Seth (20223270)

CERTIFICATE

Certified that the work contained in the report titled "*Action Recognition from Sport Videos using Multimodal Transfer Learning*", by

Shreyansh Verma (20223257),
Sirra Veera Pratap (20223263),
Vinayak Tripathi (20223312),
Tavis Mariageorge James (20223566),
Subham Seth (20223270),
has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Abhimanyu Sahu
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

18th November,2025

Abstract

Human Activity Recognition (HAR) is an important domain in computer vision with applications in sports analytics, healthcare monitoring, and surveillance. Standard CNN models extract spatial features well but struggle with the temporal behavior present in videos, motivating multimodal systems that use both appearance and motion cues. This work presents a two-stream framework for efficient large-scale action recognition: a MobileNetV2-based spatial branch using transfer learning for RGB analysis, and a custom CNN-based temporal branch trained on optical flow. Both branches use LSTMs for sequence modeling, and a late-fusion feedforward network combines their outputs for classification. To support large datasets, we built a resumable preprocessing pipeline for 8,232 videos and an improved data-loading system that reduced training time per epoch from over two hours to about seven minutes. Experiments on 63 sports-related UCF-101 classes achieve 89.3% validation accuracy with convergence in around 20 epochs. Confusion-matrix trends show strong overall performance with predictable errors between visually similar actions. Tests on internet videos further demonstrate generalization and indicate scope for improvement through broader data variation and refined temporal modeling. Overall, this framework offers a practical, resource-efficient solution with clear avenues for future enhancement.

Acknowledgements

We take this opportunity to express our heartfelt gratitude to Dr. Abhimanyu Sahu, Department of Computer Science and Engineering, for his valuable guidance, encouragement, and constant support throughout the course of this project. His insights and feedback helped us refine our ideas and overcome several challenges during the development phase.

We are also thankful to the faculty and staff of our department for providing the resources and environment that made this work possible. Lastly, we extend our appreciation to our families and friends for their patience, motivation, and unwavering belief in us.

Contents

Chapter 1

Introduction

Human Activity Recognition (HAR) has emerged as a fundamental research area in computer vision, driven by its transformative potential across diverse application domains. With the exponential growth of video data and advances in deep learning, the ability to automatically interpret human actions from visual sequences has become increasingly valuable. HAR systems enable numerous practical applications including automated sports analytics for performance optimization, continuous patient monitoring in healthcare facilities, intelligent surveillance for security threat detection, assistive technologies for elderly care, and natural human-computer interaction interfaces.

The challenge of recognizing human activities from video data is inherently complex due to several factors. Actions exhibit significant intra-class variation due to differences in execution speed, individual body types, camera viewpoints, and environmental contexts. Simultaneously, inter-class similarity poses difficulties when visually or kinematically similar actions must be distinguished, such as differentiating between various gymnastics movements or distinguishing tennis serve from volleyball spike. Traditional approaches relying on hand-crafted features proved insufficient for capturing the rich spatiotemporal patterns that characterize human activities.

The advent of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized computer vision by enabling automatic learning of hierarchical feature representations. However, standard CNN architectures designed for static image analysis face fundamental limitations when applied to video understanding. These models effectively extract spatial features from individual frames but fail to capture temporal dynamics, the motion patterns and sequential dependencies across frames that are essential for action recognition. A single frame showing a person with a basketball provides no information about whether they are dribbling, shooting, or passing, yet these actions are trivially distinguishable when temporal information is considered.

This recognition gap motivated the development of specialized architectures for video understanding. Two-stream networks, which separately process spatial appearance and temporal motion information, demonstrated significant performance improvements by explicitly modeling both what objects appear in the video and how they move over time. Despite their theoretical elegance, these architectures introduce substantial practical challenges, particularly regarding computational efficiency and data engineering complexity. The preprocessing overhead associated with computing dense optical flow fields for large video datasets, combined with severe input/output bottlenecks during training, often renders these sophisticated models impractical for researchers with limited computational resources.

The primary contributions of this research work are as follows:

1. **Efficient Two-Stream Architecture:** We designed and implemented a lightweight two-stream CNN-LSTM framework that achieves 89.3% validation accuracy on sixty-three sports action categories while remaining trainable on free-tier cloud computing platforms. The strategic selection of MobileNetV2 as the spatial backbone reduces parameters by 97% compared to VGG16 while maintaining competitive performance.

2. **Resumable Preprocessing Pipeline:** We developed a fault-tolerant preprocessing system that enables incremental optical flow computation across multiple interrupted sessions through checkpoint validation. This innovation addresses the critical challenge of completing six to eight hour preprocessing

tasks on platforms with four to five hour runtime limitations, making large-scale video preprocessing feasible without access to uninterrupted compute resources.

3. **Optimized Data Loading Strategy:** We implemented a two-stage training workflow that eliminates the input/output bottleneck by performing one-time bulk transfer of preprocessed data from slow network storage to fast local storage before training. This optimization reduced per-epoch training time by twenty-fold, from approximately 2.5 hours to seven minutes, enabling practical training on cloud platforms with network-attached file systems.

4. **Comprehensive Engineering Documentation:** We systematically documented all practical challenges encountered during implementation, including detailed solutions for preprocessing resumability, storage optimization, and efficient batch loading. This documentation bridges the gap between theoretical model descriptions and practical deployment, providing a reproducible roadmap for researchers working with limited computational resources.

5. **Systematic Performance Analysis:** We conducted thorough evaluation including per-class metrics, confusion matrix analysis, and real-world video testing. Our analysis reveals specific patterns of misclassification between visually similar action categories, providing insights into the learned representations and identifying clear directions for future improvement through data augmentation and architectural refinement.

1.1 Problem Statement

The central problem addressed in this research is the development of an efficient and practical human activity recognition system capable of accurately classifying sports-related actions in video sequences while remaining feasible to implement and train on resource-constrained cloud computing platforms. Specifically, we aim to design a framework that achieves high classification accuracy across sixty-three sports action categories while systematically overcoming the computational and engineering bottlenecks that typically prevent two-stream architectures from being successfully deployed in limited-resource environments.

The problem encompasses several critical challenges. First, the system must effectively capture both spatial appearance information and temporal motion dynamics, requiring a multimodal architecture that can learn complementary representations from RGB frames and optical flow fields. Second, the preprocessing pipeline must handle the computationally intensive task of generating dense optical flow for thousands of videos without exceeding the runtime limitations imposed by free-tier cloud platforms. Third, the training infrastructure must overcome severe input/output bottlenecks to enable efficient GPU utilization despite reading from network-attached storage systems. Finally, the complete system must achieve competitive accuracy while maintaining practical feasibility for researchers without access to high-performance computing clusters.

1.2 Limitations of Existing Approaches

While existing research has established strong theoretical foundations for action recognition, several critical limitations persist in current methodologies that hinder practical deployment and widespread adoption.

Current two-stream architectures as proposed achieve excellent accuracy but require extensive computational resources for both preprocessing and training. The spatial stream typically employs deep networks such as VGG16 or ResNet that contain over one hundred million parameters, demanding substantial GPU memory and training time. More critically, the temporal stream's reliance on pre-computed optical flow creates a severe preprocessing bottleneck that is rarely addressed in detail within the literature.

Research papers including Abdulazeem et al. [1] and Gupta and Mahapatra [2] demonstrate effective use of transfer learning for the spatial stream and validate the two-stream paradigm for action recognition. However, these works provide limited discussion of the practical engineering challenges encountered during implementation. The optical flow computation process is typically described briefly without addressing the substantial time requirements, storage overhead, or strategies for handling computation interruptions on platforms with runtime limitations. This gap between theoretical description and practical implementation leaves researchers struggling to reproduce results when working with limited computational budgets.

Furthermore, existing approaches often overlook the critical input/output bottleneck that emerges during training. When preprocessed optical flow data exceeds fifty gigabytes and resides on slow network-attached storage, the data loading pipeline becomes the dominant bottleneck rather than model computation. Published works rarely discuss strategies for optimizing data access patterns or implementing efficient caching mechanisms, leading to severe GPU underutilization where expensive hardware remains idle while waiting for data transfers.

The temporal aggregation strategies employed in early two-stream models also present limitations. Simple averaging of predictions from sparsely sampled frames discards valuable sequential information about the temporal ordering of sub-actions within a complete activity. While more recent work has incorporated recurrent neural networks to address this limitation, comprehensive guidance on architectural design choices, hyperparameter selection, and fusion strategies remains sparse in the literature.

Chapter 2

Related Work

The field of Human Action Recognition has evolved significantly over the past two decades. This chapter provides an overview of the foundational research that leads to and informs our proposed model, tracing the progression from hand-crafted features through modern deep learning architectures.

2.1 Early Approaches in Action Recognition

Before the rise of deep learning, HAR relied heavily on hand-crafted features. These methods involved manually engineering algorithms to extract specific visual patterns such as Scale-Invariant Feature Transform (SIFT) or Histogram of Oriented Gradients (HOG). Researchers would design feature detectors based on domain knowledge and intuition about what visual cues might be important for action recognition. While these approaches were foundational and achieved reasonable performance on controlled datasets, they proved to be brittle and fragile when deployed in real-world scenarios. The manually designed features were highly sensitive to changes in lighting conditions, camera viewpoint, occlusions, and background clutter. More critically, these methods were unable to capture the high-level semantic meaning of an action, instead relying on low-level visual statistics that often failed to generalize across different contexts.

2.2 The Rise of Deep Learning

The breakthrough success of Convolutional Neural Networks in image classification, fundamentally revolutionized the field of computer vision. This watershed moment proved that neural networks could automatically learn hierarchical feature representations directly from raw pixel data, eliminating the need for manual feature engineering. Researchers immediately sought to extend these powerful techniques to the domain of video understanding.

An early attempt to apply deep learning to action recognition was the 3D-CNN architecture. This approach extended the traditional 2D convolutional filter into a 3D spatiotemporal cube that could simultaneously process spatial dimensions and the temporal dimension. The intuition was elegant: if 2D convolutions can detect spatial patterns like edges and textures, then 3D convolutions should be able to detect spatiotemporal patterns like motion trajectories. While this innovation was conceptually promising, 3D-CNNs proved to be computationally massive and notoriously difficult to train. The number of parameters exploded compared to 2D counterparts, requiring enormous amounts of training data and computational resources that were impractical for most research groups.

2.3 The Two-Stream Architecture

A major breakthrough came from Simonyan and Zisserman [3] with their introduction of the Two-Stream Convolutional Network. This architecture forms the conceptual foundation upon which our project is built. Their key insight was that forcing a single network to simultaneously learn both appearance information and motion patterns was inherently inefficient and potentially counterproductive. Instead, they proposed building two separate, specialized networks that could each focus on a distinct aspect of the video understanding problem.

The first stream, termed the spatial stream, consisted of a standard 2D-CNN trained on individual RGB video frames. This network's purpose was to learn the appearance and context of the scene, answering questions such as what objects are present, what is the setting, and what spatial relationships exist between elements.

The second stream, called the temporal stream, was a separate 2D-CNN trained on stacks of pre-computed optical flow fields. Optical flow provides a dense pixel-wise representation of motion between consecutive frames, encoding both the magnitude and direction of movement at every location in the image. This stream's sole purpose was to learn patterns of movement without being distracted by appearance details.

The innovation of this two-stream approach was validated by its exceptional performance. By fusing the predictions from these two expert streams through simple averaging or concatenation, the model achieved state-of-the-art results on standard benchmarks. This empirical success proved that explicitly separating and independently learning spatial and temporal features was a superior strategy compared to attempting joint spatiotemporal learning in a single network.

2.4 The Role of Transfer Learning in HAR

Despite the architectural advantages of two-stream networks, training them remained computationally expensive, particularly for the spatial stream which required learning robust visual representations from video data. Researchers including Abdulazeem et al. [1] and Gupta and Mahapatra [2] quickly adopted transfer learning as a practical solution to this challenge.

The underlying logic is both simple and powerful. A deep CNN pre-trained on the massive ImageNet dataset, which contains fourteen million labeled images across thousands of object categories, has already learned to recognize fundamental visual patterns including edges, corners, textures, object parts, and complete objects. Models such as VGG16, ResNet, and MobileNetV2 represent expert visual feature extractors that have been trained on orders of magnitude more data than most action recognition datasets provide. By freezing the weights of these pre-trained networks and using them as fixed feature extractors, the action recognition model only needs to learn the final high-level mapping from visual features to action categories. This dramatically reduces the number of parameters that must be learned from scratch, decreasing both training time and the amount of labeled video data required while simultaneously improving accuracy through the use of more robust learned features.

2.5 The Rise of Sequential Models

The original two-stream architecture by Simonyan and Zisserman had an important limitation in how it aggregated temporal information. Their approach involved sampling a small number of frames or flow stacks from each video and averaging the individual predictions. While this captured some notion of temporal variation, it fundamentally discarded the sequential ordering of the action. Activities are inherently temporal sequences where the order of events matters critically: shooting an arrow requires pulling back the bow before releasing it, and these sub-actions must occur in the correct temporal order.

To address this limitation, researchers began incorporating Recurrent Neural Networks, specifically Long Short-Term Memory units, into the action recognition pipeline. LSTMs are specifically designed to process sequential data while maintaining an internal memory state that can capture long-range temporal dependencies. By feeding the CNN-extracted features from consecutive frames into an LSTM, the model gains the ability to learn complex temporal relationships and patterns. For instance, the LSTM can learn that the action of archery typically involves a characteristic sequence: drawing the bow backward, holding steady, and then releasing. This sequential modeling capability represents a significant advancement over simple temporal pooling strategies.

2.6 Identified Research Gaps and Engineering Challenges

While the theory of a two-stream CNN-LSTM architecture enhanced with transfer learning is well established in the literature, a significant gap exists between theoretical design and practical real-world implementation. Research papers such as Abdulazeem et al. [1] often describe the optical flow computation process with brief statements as if it were a trivial preprocessing step. However, in practice, this single step represents the largest computational bottleneck of the entire project lifecycle.

Calculating dense optical flow for a large-scale dataset such as UCF-101, which contains over thirteen thousand videos, is extraordinarily resource-intensive. Each video requires processing every consecutive frame pair through computationally expensive algorithms like Farnebäck's method. Even on modern hardware, this process often requires several days of continuous computation. Moreover, the resulting preprocessed dataset frequently exceeds fifty gigabytes of stored motion fields, introducing a severe input/output bottleneck during the training phase.

This bottleneck manifests as GPU underutilization during training. Modern deep learning relies on keeping the GPU fully saturated with computation, but when the data loading pipeline cannot supply batches quickly enough, the expensive GPU hardware sits idle waiting for data transfers from storage. This problem is particularly acute when training on cloud platforms with network-attached storage or slow file systems. Reading thousands of small preprocessed files sequentially from disk becomes the rate-limiting step, and training that should take minutes per epoch instead takes hours.

Addressing these engineering challenges forms a core motivation behind the design choices in our proposed two-stream framework. Our work focuses not only on achieving high accuracy through sophisticated model architecture, but also on developing practical solutions for computational efficiency, resumable preprocessing pipelines, and optimized data access strategies that enable successful deployment on resource-constrained platforms.

Chapter 3

Proposed Method

This research proposes a practical and efficient two-stream deep learning framework specifically engineered to overcome the identified limitations while maintaining high classification accuracy. Our architecture integrates spatial and temporal streams with Long Short-Term Memory networks for sequential modeling, enhanced through strategic use of transfer learning and optimized data engineering.

The spatial stream employs MobileNetV2[4] , a lightweight pre-trained CNN with only 3.5 million parameters, as a frozen feature extractor. This design choice balances recognition accuracy with computational efficiency, making the model suitable for training on free-tier cloud platforms with limited GPU memory. Twenty frames are extracted through sparse temporal sampling across each video's duration, ensuring representation of all action phases while maintaining computational tractability. These frames are processed through the time-distributed MobileNetV2 backbone, and the resulting feature sequence is fed into an LSTM layer that learns to model temporal dependencies in the spatial feature evolution.

The temporal stream processes dense optical flow computed using the Farnebäck algorithm, capturing pure motion information independent of appearance. Ten consecutive optical flow fields extracted from each video's temporal midpoint are processed through a custom two-layer CNN designed specifically for the two-channel flow representation. This lightweight architecture learns motion-specific features that are then aggregated through a second LSTM layer, producing a compact summary of the action's temporal dynamics.

Both streams converge at a late-fusion module where their 256-dimensional LSTM outputs are concatenated and processed through a feedforward network for final classification. This architecture enables each stream to develop specialized expertise before integration, while the LSTM layers ensure that sequential information is preserved rather than discarded through simple averaging.

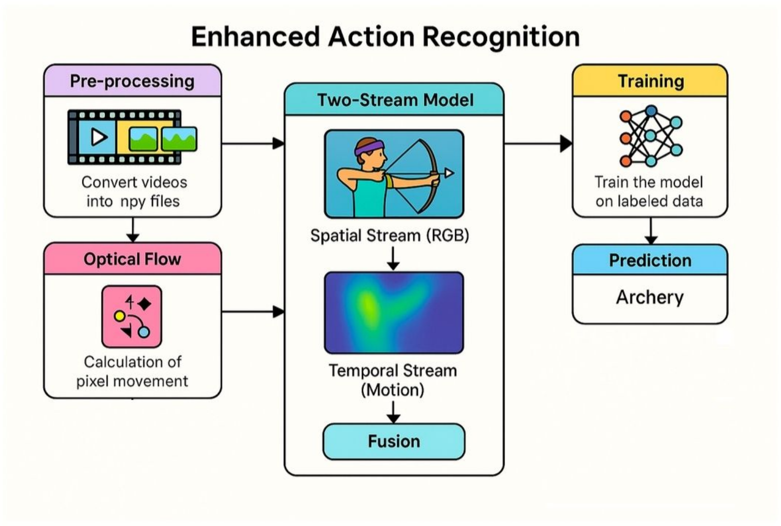


Figure 1: Enhanced Action Recognition using Two-Stream Model

Critically, our proposed system incorporates novel engineering solutions to address practical implementation challenges. A resumable preprocessing pipeline with checkpoint validation enables incremental dataset construction across multiple interrupted cloud computing sessions. A two-stage training workflow separates one-time bulk data transfer from iterative model training, eliminating the input/output bottleneck by copying preprocessed data from network storage to local high-speed solid-state storage before training commences.

This chapter details the architecture of our proposed solution, which implements a fused, two-stream multimodal network specifically designed to overcome the computational bottlenecks of traditional Human Activity Recognition models.

The system combines an efficient preprocessing pipeline with a lightweight yet high-performance network architecture that can be trained successfully on limited computational resources.

3.1 Overall System Architecture

Our model consists of two independent processing streams that extract complementary information from video data, followed by a late-fusion stage that combines their learned representations for final classification. The spatial stream processes sparse RGB frames to understand scene context and object appearance, answering the question of what is present in the video. The temporal stream processes dense optical flow fields to capture motion patterns, addressing how objects and people are moving through space. Finally, a fusion head implemented as a feedforward neural network receives the high-level summaries from both LSTM modules and performs the final action classification across sixty-three sports categories.

This architecture represents a careful balance between theoretical sophistication and practical implementability. Each component has been selected not only for its contribution to recognition accuracy but also for its computational efficiency and compatibility with the constraints of cloud-based training environments.

3.2 Stream 1: The Spatial Stream

The spatial stream analyzes the visual appearance of video frames to extract semantic information about objects, scenes, and spatial relationships. This stream answers questions such as: What objects are visible? What is the environmental context? How are elements spatially arranged within the frame?

3.2.1 Data Preprocessing: Sparse Temporal Sampling

To create a fixed-length input suitable for batch processing, we implement sparse temporal sampling where twenty frames are uniformly extracted from each video. The preprocessing function reads each video's total frame count and generates twenty evenly spaced frame indices spanning the entire duration. This sampling strategy is superior to simply extracting the first twenty consecutive frames because it provides a compact yet comprehensive summary of the complete action sequence. Actions often have distinct phases such as preparation, execution, and follow-through, and our sparse sampling ensures representation of all temporal phases.

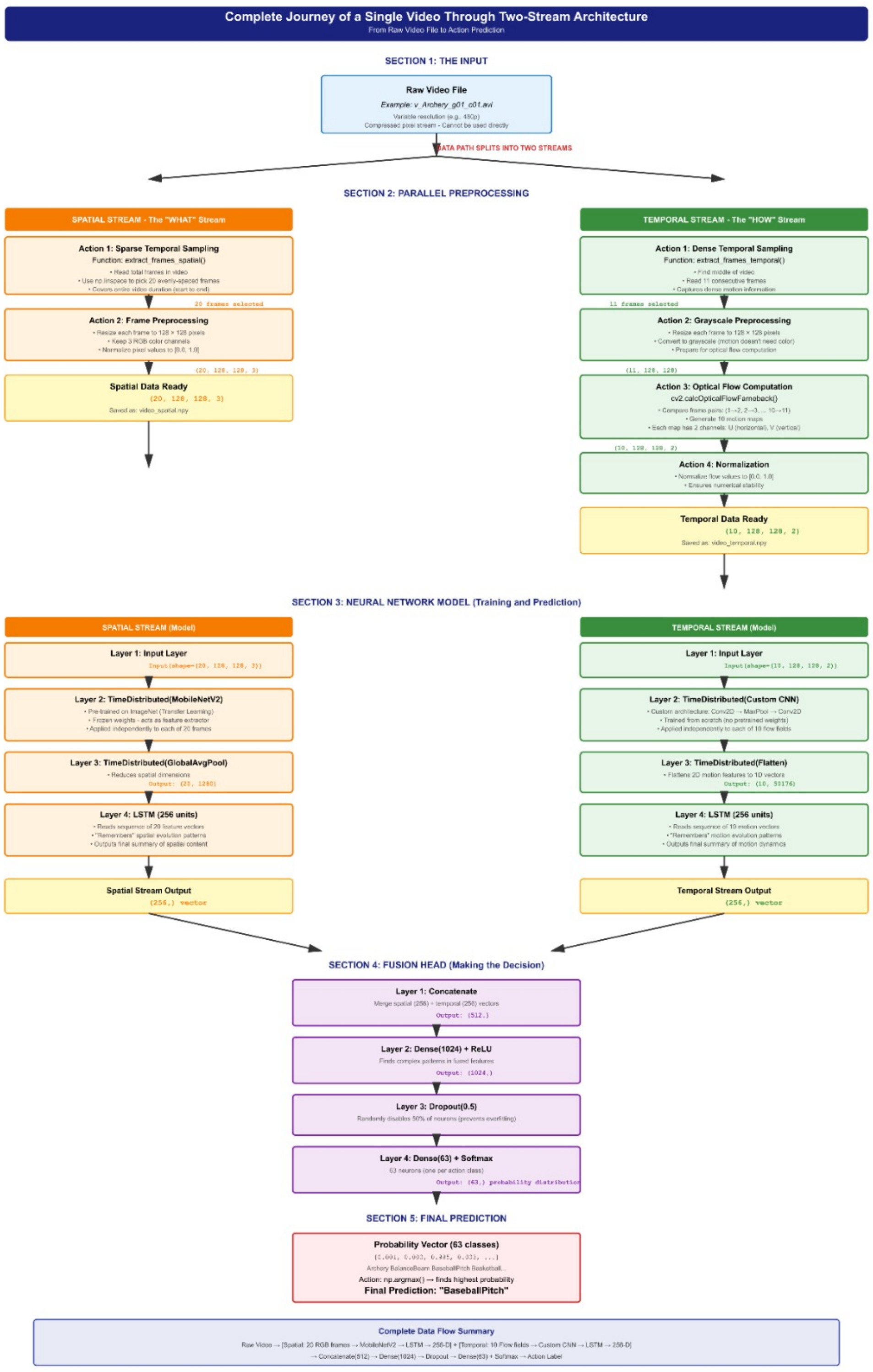
Each sampled frame undergoes standardized preprocessing. Frames are resized to a uniform dimension of 128 by 128 pixels using bilinear interpolation, ensuring consistent input dimensions regardless of source video resolution. Pixel intensities are normalized from the original range to floating-point values between zero and one, which improves gradient flow during backpropagation and accelerates convergence. The final output represents twenty RGB frames ready for input to the neural network.

3.2.2 Architecture: Transfer Learning with MobileNetV2

The spatial stream architecture implements a time-distributed CNN followed by an LSTM for temporal aggregation, as inspired by the work of Abdulazeem et al. [1]. The model accepts input tensors representing twenty sequential frames of specified dimensions. For the convolutional backbone, we selected MobileNetV2 as our transfer learning base model. While previous work such as Abdulazeem et al. utilized VGG16, we chose MobileNetV2 for its superior efficiency characteristics. VGG16 contains approximately 138 million parameters and requires substantial computational resources, whereas MobileNetV2 employs depthwise separable convolutions to achieve comparable accuracy with only 3.5 million parameters, making it significantly more suitable for our resource-constrained cloud environment.

The pre-trained MobileNetV2 model, with its top classification layers removed, is wrapped within a time-distributed layer. This wrapper applies the identical CNN architecture independently to each of the twenty input frames, generating a sequence of twenty feature vectors where each vector represents the learned spatial features of one frame. Critically, the convolutional weights remain frozen during training, meaning we treat the pre-trained network purely as a fixed feature extractor. This design choice dramatically reduces the number of trainable parameters and leverages the rich visual representations learned from ImageNet's fourteen million images.

The resulting sequence of twenty spatial feature vectors is then fed into an LSTM layer containing 256 hidden units. This recurrent layer processes the temporal sequence and outputs a single 256-dimensional vector that encodes the LSTM's final hidden state. This output vector represents the model's learned summary of the entire spatial stream, capturing what objects and scenes appear throughout the video and how their spatial arrangements evolve over time.



Complete Journey of a Single Video Through Two-Stream Architecture
From Raw Video File to Action Prediction
SECTION 1: THE INPUT
Raw Video File
Example: v_Archery_g01_c01.avi
Variable resolution (e.g. 480p)
Compressed pixel stream - Cannot be used directly
DATA PATH SPLITS INTO TWO STREAMS
SECTION 2: PARALLEL PREPROCESSING
SPATIAL STREAM - The "WHAT" Stream
Action 1: Sparse Temporal Sampling
Function: extract_frames_spatial()
• Read total frames in video
• Use np.linspace to pick 20 evenly-spaced frames
• Covers entire video duration (start to end)
20 frames selected
Action 2: Frame Preprocessing
• Resize each frame to 128 × 128 pixels
• Keep 3 RGB color channels
• Normalize pixel values to [0.0, 1.0]
(20, 128, 128, 3)
Spatial Data Ready
(20, 128, 128, 3)
Saved as: video_spatial.npy
TEMPORAL STREAM - The "HOW" Stream
Action 1: Dense Temporal Sampling
Function: extract_frames_temporal()
• Find middle of video
• Read 11 consecutive frames
• Captures dense motion information
11 frames selected
Action 2: Grayscale Preprocessing
• Resize each frame to 128 × 128 pixels
• Convert to grayscale (motion doesn't need color)
• Prepare for optical flow computation
(11, 128, 128)
Action 3: Optical Flow Computation
cv2.calcOpticalFlowFarneback()
• Compare frame pairs: (1→2, 2→3, ... 10→11)
• Generate 10 motion maps
• Each map has 2 channels: U (horizontal), V (vertical)
(10, 128, 128, 2)
Action 4: Normalization
• Normalize flow values to [0.0, 1.0]
• Ensures numerical stability
Temporal Data Ready
(10, 128, 128, 2)
Saved as: video_temporal.npy
SECTION 3: NEURAL NETWORK MODEL (Training and Prediction)
SPATIAL STREAM (Model)
Layer 1: Input Layer
Input(shape=(20, 128, 128, 3))
Layer 2: TimeDistributed(MobileNetV2)
• Pre-trained on ImageNet (Transfer Learning)
• Frozen weights - acts as feature extractor
• Applied independently to each of 20 frames
Layer 3: TimeDistributed(GlobalAvgPool)
• Reduces spatial dimensions
Output: (20, 1280)
Layer 4: LSTM (256 units)
• Reads sequence of 20 feature vectors
• "Remembers" spatial evolution patterns
• Outputs final summary of spatial content
Spatial Stream Output
(256,) vector
TEMPORAL STREAM (Model)
Layer 1: Input Layer
Input(shape=(10, 128, 128, 2))
Layer 2: TimeDistributed(Custom CNN)
• Custom architecture: Conv2D → MaxPool → Conv2D
• Trained from scratch (no pretrained weights)
• Applied independently to each of 10 flow fields
Layer 3: TimeDistributed(Flatten)
• Flattens 2D motion features to 1D vectors
Output: (10, 50176)
Layer 4: LSTM (256 units)
• Reads sequence of 10 motion vectors
• "Remembers" motion evolution patterns
• Outputs final summary of motion dynamics
Temporal Stream Output
(256,) vector
SECTION 4: FUSION HEAD (Making the Decision)
Layer 1: Concatenate
Merge spatial (256) + temporal (256) vectors
Output: (512,)
Layer 2: Dense(1024) + ReLU
Finds complex patterns in fused features
Output: (1024,)
Layer 3: Dropout(0.5)
Randomly disables 50% of neurons (prevents overfitting)
Layer 4: Dense(63) + Softmax
63 neurons (one per action class)
Output: (63,) probability distribution
SECTION 5: FINAL PREDICTION
Probability Vector (63 classes)
[0.001, 0.003, 0.985, 0.003, ...]
Archery BalanceBeam BaseballPitch Basketball...
Action: np.argmax() → finds highest probability
Final Prediction: "BaseballPitch"
Complete Data Flow Summary
Raw Video → [Spatial: 20 RGB frames → MobileNetV2 → LSTM → 256-D] + [Temporal: 10 Flow fields → Custom CNN → LSTM → 256-D]
→ Concatenate(512) → Dense(1024) → Dropout → Dense(63) + Softmax → Action Label

Figure 2: Complete Journey of a Single Video Through Two-Stream Architecture

3.3 Stream 2: The Temporal Stream

The temporal stream focuses exclusively on motion patterns, analyzing how pixels move between consecutive frames without being influenced by appearance details, following the principles established by Simonyan and Zisserman [3]. This stream addresses questions such as: What is the dominant direction of motion? What is the speed and magnitude of movement? What spatial patterns characterize the motion field?

3.3.1 Data Preprocessing: Dense Optical Flow

This preprocessing component represents the most computationally intensive phase of the entire project. To capture pure motion information, we employ OpenCV's implementation of the Farnebäck optical flow algorithm. This algorithm estimates dense pixel-wise displacement fields between consecutive frames, producing two-channel flow representations where the first channel encodes horizontal motion and the second channel encodes vertical motion at every pixel location. The magnitude and direction of these flow vectors provide a rich representation of movement that is largely invariant to lighting changes and object appearance.

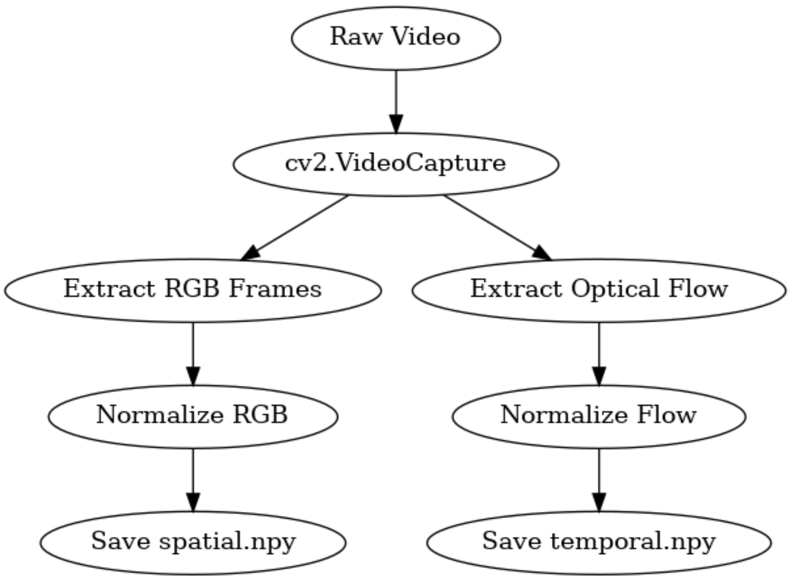


Figure 3: Extraction of RGB Frames and Optical Flow Data from Videos

3.3.2 Data Preprocessing: Dense Sampling

For temporal processing, we extract ten optical flow fields from each video. Since computing optical flow requires frame pairs, we need eleven consecutive frames to generate ten flow fields. The preprocessing function locates the temporal midpoint of each video and reads eleven consecutive frames from that region, ensuring we capture motion from the most action-dense portion rather than potentially static opening or closing segments.

Each of the eleven frames undergoes preprocessing before flow calculation. Frames are resized to 128 by 128 pixels and converted to grayscale, as optical flow algorithms operate on intensity gradients rather than color information. We then iteratively apply the optical flow algorithm to each consecutive frame pair, generating ten motion maps. Each flow field represents horizontal and vertical displacement at every pixel location. These raw flow values are normalized to ensure numerical stability during training. The final output represents ten sequential motion maps that encode the temporal dynamics of the action.

3.3.3 Architecture: Custom CNN

Unlike the spatial stream where we could leverage pre-trained models, the temporal stream requires a custom architecture because standard pre-trained networks expect three-channel RGB input whereas our optical flow fields contain only two channels for motion components. We therefore designed a lightweight custom CNN trained from scratch, specifically optimized to extract motion features.

The model accepts input tensors representing ten sequential flow fields. The convolutional architecture consists of two convolutional blocks, each containing a convolutional layer followed by max pooling for spatial downsampling. The first convolutional layer uses thirty-two filters with three-by-three kernels and ReLU activation, learning to detect basic motion patterns such as directional edges in the flow field. After two-by-two max pooling, a second convolutional layer with sixty-four filters further abstracts these patterns into higher-level motion features. This CNN is wrapped within a time-distributed layer, applying the same convolutional operations independently to each of the ten motion maps and producing a sequence of ten abstract motion feature vectors.

The temporal sequence of motion features is then processed by an LSTM layer with 256 hidden units. This recurrent layer learns to recognize temporal patterns in the motion sequence, such as the characteristic acceleration and deceleration phases of different sports actions. The LSTM outputs a single 256-dimensional vector representing its learned summary of the motion stream, encoding information about the overall motion pattern such as whether the action involves upward jumping motions followed by downward landing, or cyclical repetitive motions characteristic of running.

3.4 Fusing the Streams

At this stage of the architecture, we have generated two complementary 256-dimensional summary vectors. The first comes from the spatial stream and encodes what objects, people, and environmental context appear in the video. The second comes from the temporal stream and encodes how movement unfolds across the temporal sequence. The final architectural component must effectively combine these learned representations to produce the action classification.

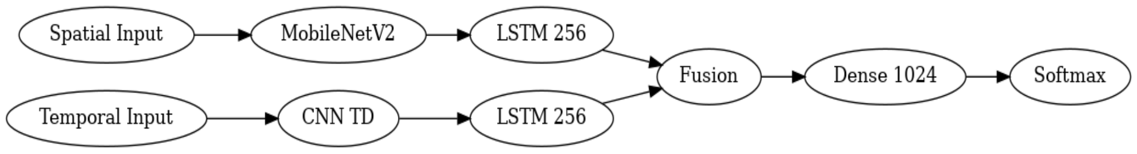


Figure 4: Fusing of 256 dimensional LSTM vectors

3.4.1 Sequential Processing with LSTMs

Our fusion approach represents a significant departure from the original two-stream architecture proposed by Simonyan and Zisserman [3]. Rather than simply averaging the raw predictions from both streams, we employ LSTMs as the final processing stage within each stream before fusion. This means we are not fusing individual frame-level convolutional features, but rather holistic sequence-level summaries that each LSTM has learned to extract from its respective input modality.

The fusion mechanism concatenates the two 256-dimensional embeddings to create a single 512-dimensional combined feature vector. This concatenated representation flows into a fully connected feedforward network consisting of a dense layer with 1024 units and ReLU activation, followed by dropout with probability 0.5 for regularization. The final output layer is a dense layer with sixty-three units corresponding to our action classes, activated by softmax to produce a probability distribution over all possible actions. The class with the highest probability represents the model's predicted action label.

This late-fusion architecture allows each stream to develop specialized expertise in its respective domain before combination, and the fusion network learns to weigh and integrate their complementary information for optimal classification performance.

Chapter 4

Experimental Results

This chapter details the practical implementation of our proposed framework, covering the dataset selection, computational environment, and the critical engineering challenges that required innovative solutions to enable successful training on resource-constrained hardware.

4.1 Dataset

We conducted our experiments using the UCF-101 dataset [5] , which has become a standard benchmark for evaluating human action recognition systems. This dataset consists of 13,320 real-world video clips sourced from YouTube, organized into 101 distinct action categories spanning a wide range of human activities. For our research, we focused on a specific subset of this data to maintain computational feasibility while ensuring sufficient complexity for meaningful evaluation. We programmatically filtered the dataset to include only the sixty-three sports-related action categories, such as archery, basketball dunk, fencing, volleyball, and others. This filtering resulted in a final working dataset of 8,232 videos that formed the basis for all preprocessing, training, and evaluation experiments.

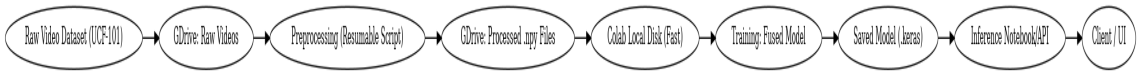


Figure 5: Training Process

4.2 Computational Environment

All experiments were conducted on Google Colab's free tier platform, which provided access to cloud-based GPU acceleration without requiring local hardware investment. The platform allocated an Intel Xeon CPU with variable core count, 12.7 gigabytes of system RAM, and an NVIDIA T4 GPU with sixteen gigabytes of video memory. Storage was split between permanent cloud storage on Google Drive for persistent data and model checkpoints, and temporary local SSD storage for high-speed data access during training sessions.

The software environment utilized TensorFlow version 2.x with the Keras high-level API for model construction and training. OpenCV provided computer vision functionality including video reading and optical flow computation. NumPy handled numerical array operations and data serialization. Scikit-learn provided utilities for dataset splitting and evaluation metrics. Matplotlib and Seaborn enabled visualization of training curves and confusion matrices.

4.3 The Engineering Bottlenecks

The primary challenge of this project was not theoretical model design, but rather a series of significant engineering bottlenecks that made the standard implementation approach completely impractical on our chosen platform. These challenges required innovative solutions that became core contributions of this work.

4.3.1 Bottleneck 1: The Preprocessing Problem

The most computationally expensive task in our entire pipeline was the calculation of dense optical flow fields for the temporal stream. This preprocessing phase required processing all 8,232 videos in the filtered dataset, extracting and computing ten optical flow fields from each video. When executed on the allocated CPU, this represented a six to eight hour compute-bound workload. However, Google Colab's free tier enforces strict runtime limits, automatically terminating sessions after approximately four to five hours of continuous execution or periods of detected inactivity. This constraint created an impossible situation where our preprocessing script could never complete in a single session, and each premature termination would result in complete loss of all progress made during that session.

4.3.2 Our Solution 1: The Resumable Preprocessing Script

We engineered a fault-tolerant resumable preprocessing pipeline that could incrementally build the complete preprocessed dataset across multiple interrupted sessions. The implementation logic is straightforward but highly effective. Before processing each video, the script constructs the expected output file path on Google Drive for both spatial frames and optical flow fields. The script then checks whether these output files already exist on disk. If the files are found, indicating this video was processed in a previous session, the script immediately skips to the next video with minimal overhead. If the files do not exist, the script performs the complete preprocessing operation including frame extraction, optical flow computation, normalization, and saves the resulting arrays to persistent storage.

This approach enabled successful dataset construction through an iterative process spanning multiple days. On the first day, we initiated the preprocessing script and allowed it to run for the maximum session duration of approximately four hours, during which it processed roughly five thousand videos before the session automatically terminated. On the second day, we executed the identical preprocessing script again. The script rapidly skipped the five thousand previously processed videos in about five minutes by checking for existing output files, then resumed processing from the next unprocessed video. This second session ran for approximately three additional hours and successfully completed the remaining videos. The end result was a complete preprocessed dataset exceeding fifty gigabytes, constructed entirely on a free-tier cloud platform that would have made traditional approaches impossible.

4.3.3 Bottleneck 2: The I/O Problem

After completing the preprocessing phase, we began the model training process and immediately encountered a second critical bottleneck that prevented effective training. Initial training runs exhibited extremely poor performance, with each training epoch estimated to require over two hours to complete. Monitoring tools revealed that the GPU utilization remained at or near zero percent for the vast majority of training time, indicating the expensive computational hardware was sitting idle rather than performing useful work.

Investigation revealed the root cause was severe input/output throughput limitations. Our initial implementation of the data generator was configured to read preprocessed files directly from Google Drive's mounted filesystem. However, Google Drive is fundamentally a cloud file synchronization service, not a high-performance disk system. It is extremely slow at handling thousands of small random file access requests. During each training step, the data generator would request sixteen batch samples, requiring the system to locate and transfer thirty-two separate files from Google Drive's cloud servers to the Colab runtime. This network-based file access became the dominant bottleneck, with the model spending approximately twenty-three seconds per training step, of which over twenty-two seconds was simply waiting for file transfers to complete while the GPU remained completely idle.

4.3.4 Our Solution 2: The Local Disk Copy

We resolved this bottleneck by implementing a two-stage workflow that separated one-time data transfer from iterative training. At the beginning of each training session, before any model construction or compilation, we execute a single shell command to perform a complete copy of the preprocessed dataset from slow Google Drive storage to fast local SSD storage. This bulk transfer operation completes in approximately fifteen to twenty minutes, leveraging optimized bulk transfer protocols rather than thousands of individual file requests.

Following this initial copy operation, we reconfigured the data generator to read preprocessed files from the local SSD path rather than the Google Drive path. This simple change completely eliminated the input/output bottleneck. Local SSD access latencies are measured in microseconds rather than seconds, and the system can easily sustain the required throughput of reading thirty-two files per training step. The impact was dramatic and immediate: training step time decreased from twenty-three seconds to less than one second, and complete epoch duration dropped from approximately two and a half hours to just seven minutes. This twenty-fold speedup was absolutely critical in making the project practically feasible within our available computational budget.

4.4 Implementation Details

4.4.1 Training Data Generation

The complete preprocessed dataset of 8,232 samples was split into training and validation subsets using an 80/20 ratio with a fixed random seed for reproducibility. This resulted in 6,585 samples allocated to the training set and 1,647 samples reserved for validation. During training, a Keras-compatible data generator handled batch loading. This generator accepted file path lists for each subset and dynamically loaded batches of preprocessed arrays directly from the local SSD filesystem, enabling efficient memory usage by loading only the data required for each training step rather than attempting to hold the entire fifty gigabyte dataset in RAM.

4.4.2 Model Hyperparameters

Table 1 presents the complete set of hyperparameters and configuration choices used throughout our experiments. For data preprocessing, all frames were resized to 128 by 128 pixels to balance spatial resolution with computational efficiency. The spatial stream employed sparse temporal sampling with twenty frames extracted per video, while the temporal stream used dense sampling with ten optical flow fields computed per video. The dataset split allocated eighty percent of samples to training and twenty percent to validation.

Table 1: Model Hyperparameters and Configuration

Parameter	**Value**
Data Parameters	
Target Image Size	128×128 pixels
Spatial Sequence Length	20 frames (Sparse Sampling)
Temporal Sequence Length	10 frames (Dense Sampling)
Dataset Split	80% Train / 20% Validation
Training Parameters	
Epochs	20
Batch Size	16
Optimizer	Adam
Learning Rate	1×10^{-4}
Loss Function	Categorical Crossentropy
Model Architecture Parameters	
Spatial Backbone	MobileNetV2 (Pre-trained)
Spatial Backbone Trainable	False
Temporal Backbone	Custom 2-Layer CNN
LSTM Units (per stream)	256
LSTM Dropout	0.5
Fused Hidden Layer	1024 (Dense)
Fused Dropout	0.5
Output Layer (Classes)	63
Output Activation	Softmax

Training was conducted for twenty epochs with a batch size of sixteen samples. The Adam optimizer was selected for its adaptive learning rate properties, configured with an initial learning rate of 0.0001. The model was trained to minimize categorical crossentropy loss, which is the standard choice for multi-class classification problems.

Regarding architectural parameters, the spatial stream utilized MobileNetV2 pre-trained on ImageNet as its convolutional backbone with all layers frozen. The temporal stream employed a custom two-layer CNN trained from scratch. Both streams incorporated LSTM layers with 256 hidden units and dropout probability of 0.5 applied to the LSTM outputs for regularization. The fusion network consisted of a dense layer with 1024 units followed by dropout with probability 0.5, culminating in a final output layer with sixty-three units corresponding to the action classes and softmax activation for probability distribution generation.

4.4.3 ModelCheckpoint

To protect our multi-hour training runs from unexpected session disconnections or runtime terminations, we implemented a model checkpoint callback provided by Keras. This callback was configured to monitor validation accuracy throughout the training process. The callback was instructed to save the complete model weights and architecture to Google Drive only when the current epoch achieved higher validation accuracy than all previous epochs. This strategy ensured that even if training was interrupted prematurely, we would retain the best-performing version of the model discovered during training. The checkpoint filepath was set to persistent Google Drive storage rather than temporary local storage, guaranteeing that saved models would survive session terminations and remain accessible for future evaluation and deployment.

4.5 Results and Analysis

After completing the full twenty-epoch training process, our proposed two-stream framework achieved excellent performance metrics that demonstrate strong generalization capability and effective learning of spatiotemporal action patterns across sixty-three distinct sports categories.

4.5.1 Training Performance

The model exhibited expected and healthy training dynamics throughout the twenty-epoch training process. Training loss decreased steadily from its initial value, eventually converging to a low value indicating successful optimization. Training accuracy climbed progressively, reaching approximately 95.5 percent by the final epoch. This high training accuracy confirms that the model successfully learned to classify the training samples with very high confidence.

Simultaneously, validation accuracy tracked training accuracy reasonably closely for the first fifteen to eighteen epochs, peaking at approximately 89.3 percent around epoch eighteen. The validation loss curve initially decreased in parallel with training loss, indicating genuine learning rather than memorization. However, beyond epoch eighteen, validation loss began to flatten and exhibit slight increases while training loss continued to decrease. This divergence between training and validation metrics represents the classic signature of overfitting, where the model begins to memorize specific training examples rather than learning generalizable patterns.

The approximately six percentage point gap between peak training accuracy of 95.5 percent and peak validation accuracy of 89.3 percent is reasonable and expected for a dataset of this complexity. Critically, our implementation of the model checkpoint callback automatically preserved the model weights from epoch eighteen where validation performance peaked, rather than retaining the slightly overfitted version from epoch twenty. This ensures that our final saved model represents the optimal balance between training performance and generalization capability.

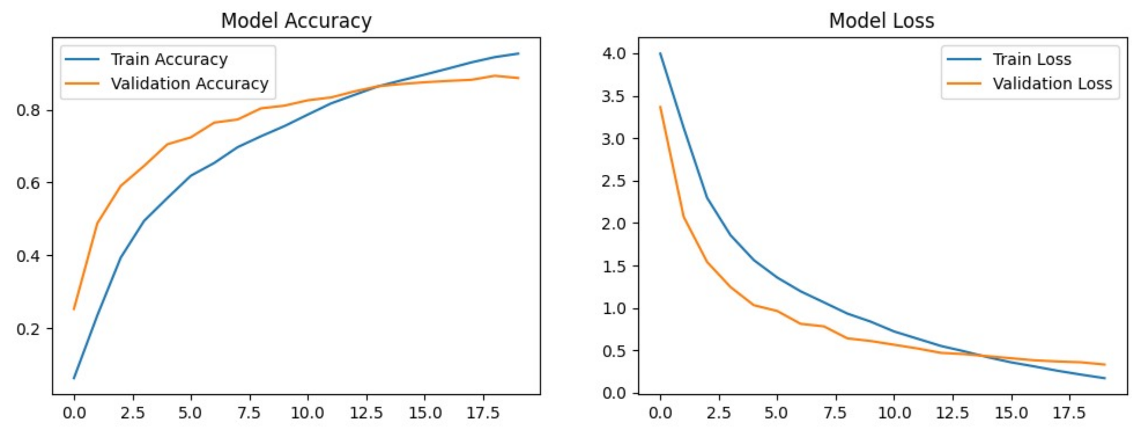


Figure 6: Training and validation accuracy and loss curves across twenty epochs, showing convergence behavior and identification of optimal checkpoint at epoch eighteen.

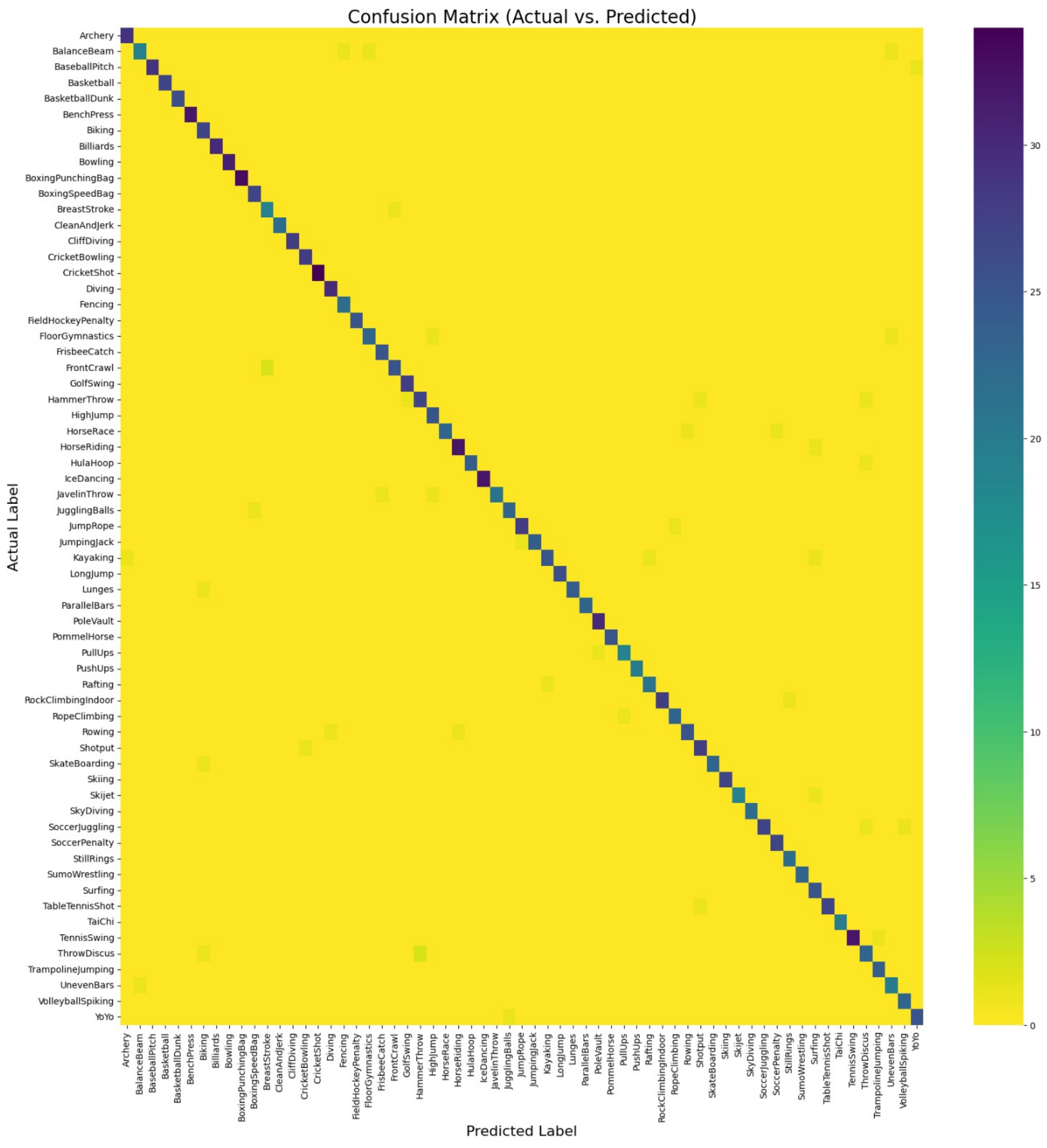


Figure 7: Confusion matrix showing classification performance across sixty-three sports action categories. Bright diagonal indicates high accuracy, with off-diagonal hotspots revealing systematic confusions between visually similar actions.

4.5.2 Per-Class Performance

We computed detailed per-class metrics including true positives, true negatives, false positives, and false negatives for every action category. These metrics provide a complete picture of model performance beyond aggregate accuracy. True positives represent samples where the model correctly predicts the action class when that action is actually present. False positives occur when the model incorrectly predicts an action class for a sample that belongs to a different category. False negatives represent samples of a particular action that the model fails to recognize, instead predicting an incorrect class. True negatives count all samples that do not belong to a class and are correctly identified as not belonging to that class.

Tables 1 through 4 present confusion matrix metrics for selected representative action classes. These tables demonstrate that most classes achieve very high true positive and true negative counts with minimal false positive and false negative errors. For instance, balance beam achieved nineteen true positives against only three false negatives, indicating the model correctly identified approximately eighty-six percent of balance beam samples while maintaining only one false positive misclassification.

Table 1: Confusion Matrix Metrics for Class *Balance Beam*

Metric	**Value**
True Positives (TP)	19
True Negatives (TN)	1624
False Positives (FP)	1
False Negatives (FN)	3

Table 2: Confusion Matrix Metrics for Class *Breast Stroke*

Metric	**Value**
True Positives (TP)	19
True Negatives (TN)	1625
False Positives (FP)	2
False Negatives (FN)	1

Table 3: Confusion Matrix Metrics for Class *Floor Gymnastics*

Metric	**Value**
True Positives (TP)	23
True Negatives (TN)	1621
False Positives (FP)	1
False Negatives (FN)	2

Table 4: Confusion Matrix Metrics for Class *Hammer Throw*

Metric	**Value**
True Positives (TP)	27
True Negatives (TN)	1615
False Positives (FP)	2
False Negatives (FN)	3

Analysis of the complete classification report revealed interesting patterns in per-class performance. Action classes with distinctive motion patterns and unique objects, such as archery, yo-yo, and fencing, achieved very high precision and recall values often exceeding ninety-five percent. These actions have characteristic visual and temporal signatures that make them easily distinguishable. Conversely, action classes belonging to more crowded semantic groups, such as various gymnastics events or track and field activities, exhibited lower precision due to higher inter-class similarity in both spatial appearance and motion patterns. Multiple gymnastics events involve similar environments such as indoor gyms with mats, similar body poses and movements, and comparable camera angles, making discrimination challenging even for sophisticated models.

4.5.3 Testing on Real-World Videos

To evaluate the model's ability to generalize beyond the UCF-101 distribution, we performed qualitative testing on new videos sourced directly from the internet that were not part of any training or validation set. These real-world tests provide valuable insight into how the model behaves when confronted with different video quality, camera angles, and production styles compared to the YouTube clips in UCF-101.

The model successfully and correctly identified a low-resolution YouTube clip of a bowling action, demonstrating effective generalization to real-world video data with quality characteristics similar to the UCF-101 training distribution. This result confirms that the learned representations transfer appropriately to unseen samples from similar data distributions.

However, the model misclassified a high-resolution, professionally-shot basketball dunk video as still rings. This misclassification aligns precisely with the confusion patterns we observed in the confusion matrix analysis, and provides additional evidence for a mild degree of overfitting to the specific visual textures and quality characteristics present in UCF-101. The professionally-shot video likely contained different lighting, camera motion, resolution, and post-processing compared to typical YouTube uploads, and these distribution shifts appear to have degraded classification performance.

These mixed results on real-world videos validate that while the model performs reliably within the UCF-101 distribution and achieves good generalization to similar low-to-medium quality internet videos, further robustness could be achieved by incorporating higher-quality and more diverse training data that better spans the full range of possible video characteristics encountered in practical deployment scenarios.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This project implemented a multimodal two-stream deep learning system for Human Action Recognition that addresses both theoretical modeling and practical engineering challenges in video analysis. By combining a spatial stream built through transfer learning with MobileNetV2 and a temporal stream trained on optical-flow images using custom CNN layers, the model learned the complex spatiotemporal cues required to classify sixty-three sports actions.

Alongside the model design, the work resolved several real-world constraints rarely discussed in literature but essential for reproducible deployment. A resumable preprocessing pipeline was created to complete multi-hour optical-flow generation even when cloud sessions were interrupted. A local caching mechanism removed severe I/O delays and reduced the per-epoch training time from more than two hours to roughly seven minutes, making GPU training feasible on free cloud services.

The final system reached a validation accuracy of 89.3 percent. Confusion matrix analysis showed reliable performance with expected errors between visually similar actions, and testing on external online videos demonstrated reasonable generalization with room for improvement through broader training diversity. Overall, this work shows that advanced multimodal action recognition can be executed efficiently under resource limitations, and the documented engineering methods form a practical guide for similar efforts.

5.2 Future Work

The model's performance suggests several directions for further improvement. The mild overfitting observed during training indicates that stronger augmentation inside the data generator would be beneficial. Incorporating random flips, small rotations, and modest zoom changes would increase input diversity and improve robustness to viewpoint and camera variation, which should enhance validation and real-world performance.

Another extension involves comparing temporal modeling approaches. Optical flow using Farnebäck delivers strong motion cues but remains the most time-intensive stage of the pipeline. Exploring much faster alternatives, such as frame differencing, could clarify whether the added cost of optical flow is justified or whether a cheaper method can achieve most of the accuracy at a fraction of the preprocessing time.

With the data pipeline now efficient, more expressive backbones can be evaluated for the spatial stream. MobileNetV2 was chosen for efficiency, but architectures like VGG16, ResNet50, or EfficientNet may yield higher accuracy given the now-manageable training times. Similar architectural scaling can be applied to the temporal stream's CNN to capture richer motion patterns.

Finally, hyperparameter optimization remains largely unexplored. The current configuration used fixed values for learning rate, batch size, and fusion-layer structure. Automated tuning using tools like Optuna or KerasTuner can search across learning rates, dropout levels, sequence lengths, and fusion-network size to identify combinations that improve accuracy and provide clearer insight into the model's sensitivity.

Together, these directions create a practical path for extending this work toward more accurate, scalable, and deployment-ready action recognition systems.

References

[1] Y. Abdulazeem, H. M. Balaha, W. M. Bahgat, and M. Badawy, "Human action recognition based on transfer learning approach," *IEEE Access*, vol. 9, pp. 82 058–82 069, 2021. [Online]. Available: https://www.researchgate.net/publication/352142405_Human_Action_Recognition_Based_on_Transfer_Learning_Approach

[2] S. Gupta and R. P. Mahapatra, "Transfer learning based human activity recognition in video data," *Grenze International Journal of Engineering and Technology*, vol. 10, no. 2, p. 12, 2024. [Online]. Available: https://www.researchgate.net/publication/371923151_Deep_Custom_Transfer_Learning_Models_for_Recognizing_Human_Activities_via_Video_Surveillance

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. [Online]. Available: https://arxiv.org/abs/1801.04381

[5] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human action classes from videos in the wild," in *Proceedings of the IEEE International Conference on Computer Vision*, 2012. [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.php

Action Recognition from Sport Videos using Multimodal Transfer Learning

ORIGINALITY REPORT

6%	5%	4%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	2%
2	Submitted to University of Bradford Student Paper	1%
3	Shankar Babu, Mahesh Babu Kota. "Synergies in Smart and Virtual Systems using computational intelligence", CRC Press, 2025 Publication	<1%
4	Tanya Buddi, Rohit Kandakatla, Rameshrao Nitin Kotkunde, Upadrasta Ramamurty, Asma Perveen. "Multi-Disciplinary Research and Sustainable Development - Proceedings of 2nd International conference on Multi-Disciplinary Research and Sustainable Development (ICMED-2025), 7th and 9th March 2025", CRC Press, 2025 Publication	<1%
5	Chen, Siyuan. "Towards Prescriptive Maintenance Using Digital Twins and Artificial Intelligence", Chalmers Tekniska Hogskola (Sweden), 2025 Publication	<1%

6	Shubham Rai, Anshika Singh, Prince Yadav. "Performance Optimization of Eco-Engineered Waterproof Concrete Blocks Using Machine Learning and Industrial By-Products", Springer Science and Business Media LLC, 2025 Publication	<1 %
7	acikbilim.yok.gov.tr Internet Source	<1 %
8	Submitted to La Trobe University Student Paper	<1 %
9	publikationen.uni-tuebingen.de Internet Source	<1 %
10	Xu, Chenfeng. "Efficient Machine Learning for Intelligent Machines", University of California, Berkeley, 2025 Publication	<1 %
11	export.arxiv.org Internet Source	<1 %
12	www.ijtsrd.com Internet Source	<1 %
13	Daniel Maitethia Memeu. "Automated malaria diagnosis and parasitemia estimation using a customized OpenFlexure microscope", Springer Science and Business Media LLC, 2025 Publication	<1 %
14	Hakilo Sabit. "Artifical Intelligence-Based Smart Security System Using Internet of	<1 %

	Things for Smart Home Applications", Electronics, 2025 Publication	
15	Wen-Hsi Lan, Chia-Ling Tsai, Wei-Chi Wu. "Optimizing Field-of-View for Type-1 Retinopathy of Prematurity via Multiple Instance Learning", Springer Science and Business Media LLC, 2025 Publication	<1 %
16	Submitted to JIS University Student Paper	<1 %
17	Submitted to JURNAL JPPKK Student Paper	<1 %
18	ijritcc.org Internet Source	<1 %
19	Hodes, Scott Gudger. "Adversarial Machine Learning Across the Digital-Physical Divide: Attack Generalization, Optical Projection, and Robustness Strategies", The Pennsylvania State University, 2025 Publication	<1 %
20	kuscholarworks.ku.edu Internet Source	<1 %
21	lib.dr.iastate.edu Internet Source	<1 %
22	www.frontiersin.org Internet Source	<1 %
23	annals-csis.org Internet Source	<1 %

24	qmro.qmul.ac.uk Internet Source	<1 %
25	V. Sharmila, S. Kannadhasan, A. Rajiv Kannan, P. Sivakumar, V. Vennila. "Challenges in Information, Communication and Computing Technology", CRC Press, 2024 Publication	<1 %
26	dspace.iiuc.ac.bd:8080 Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		