Video Classification with Deep Learning

In partial fulfillment of the requirements for the course
ADSP 32023: Advanced Computer Vision with Deep Learning

Sarthak Sunil Dhanke
Ursula Guo
Iris Huang
David Wei

University of Chicago
Masters in Applied Data Science

May 24, 2025

Contents

Abstract

In an era where video content is rapidly proliferating across platforms, understanding and classifying human actions in videos has become a vital task for applications ranging from surveillance to content recommendation. This project addresses the problem of video classification using the UCF101 dataset [2], focusing on a curated 9-class subset that balances diversity of actions with computational feasibility. Our approach investigates two distinct paradigms for spatiotemporal modeling: sequential models that process spatial features per frame using 2D CNN backbones followed by LSTMs [3], and joint models that directly capture spatiotemporal dependencies using (2+1)D convolutional architectures [4].

We implement and compare four architectures: (1) a 2D CNN + LSTM model with a custom VGG-style [5] backbone, (2) the same model with a ResNet-style [6] backbone, (3) a ResNet-based (2+1)D convolutional model, and (4) a channel-separated (2+1)D convolutional model inspired by the Channel-Separated Networks (CSN) framework [7]. The (2+1)D model achieves 79% classification accuracy while being the smallest at only 1.69 MB, making it highly efficient. In contrast, the VGG-style LSTM model achieves 90% accuracy but with a much larger footprint (21 MB). A more balanced comparison with the ResNet-style LSTM model (77% accuracy, 2.7 MB) shows that the (2+1)D model provides competitive performance with comparable parameter count and lower compute requirements.

Beyond classification, we also explore video style transfer as a secondary task. We apply optimization-based neural style transfer [8] by extracting style and content representations from individual frames using our trained models and minimizing style and content loss to generate stylized outputs. In parallel, we evaluate pretrained CycleGAN models [9], though they proved less effective due to temporal inconsistency. Our results show that framewise optimization yields visually compelling results suitable for tasks like stylized video production and film restoration. We conclude with a discussion of deployment architecture and propose avenues for future work, including lightweight real-time models for mobile deployment and extension to tasks such as lip reading and fine-grained motion analysis.

1 Exploratory Data Analysis

1.1 Dataset Overview

We use the UCF101 dataset, a widely adopted benchmark for human action recognition in videos. It contains 13,320 video clips across 101 classes, spanning sports, daily activities, and human-object interactions. Each class consists of 25 groups, with 4–7 clips per group, yielding a broad distribution of actors and scenes. Videos are recorded at 25 fps, with a resolution of 320×240, and a mean clip length of 7.21 seconds.

For this project, we selected a **9-class subset** representing diverse activity types while remaining computationally feasible. The chosen classes are:

Figure 1: Examples from the selected 9-class subset: Apply Eye Makeup, Playing Dhol, Baby Crawling, Haircut, Skydiving, Surfing, Rafting, Cricket Shot, and Shaving Beard.

This subset retains variability across motion types, lighting conditions, and environments (indoor/outdoor), allowing meaningful model evaluation despite the reduced scope.

1.2 Temporal and Structural Characteristics

1.2.1 Duration Distribution

Figure 2 shows the distribution of video durations in the subset. Most videos range from 2 to 6 seconds, with a peak around 4 seconds. A long tail exists, but the majority of clips are short, which justifies using **fixed-length sampling** strategies for training.

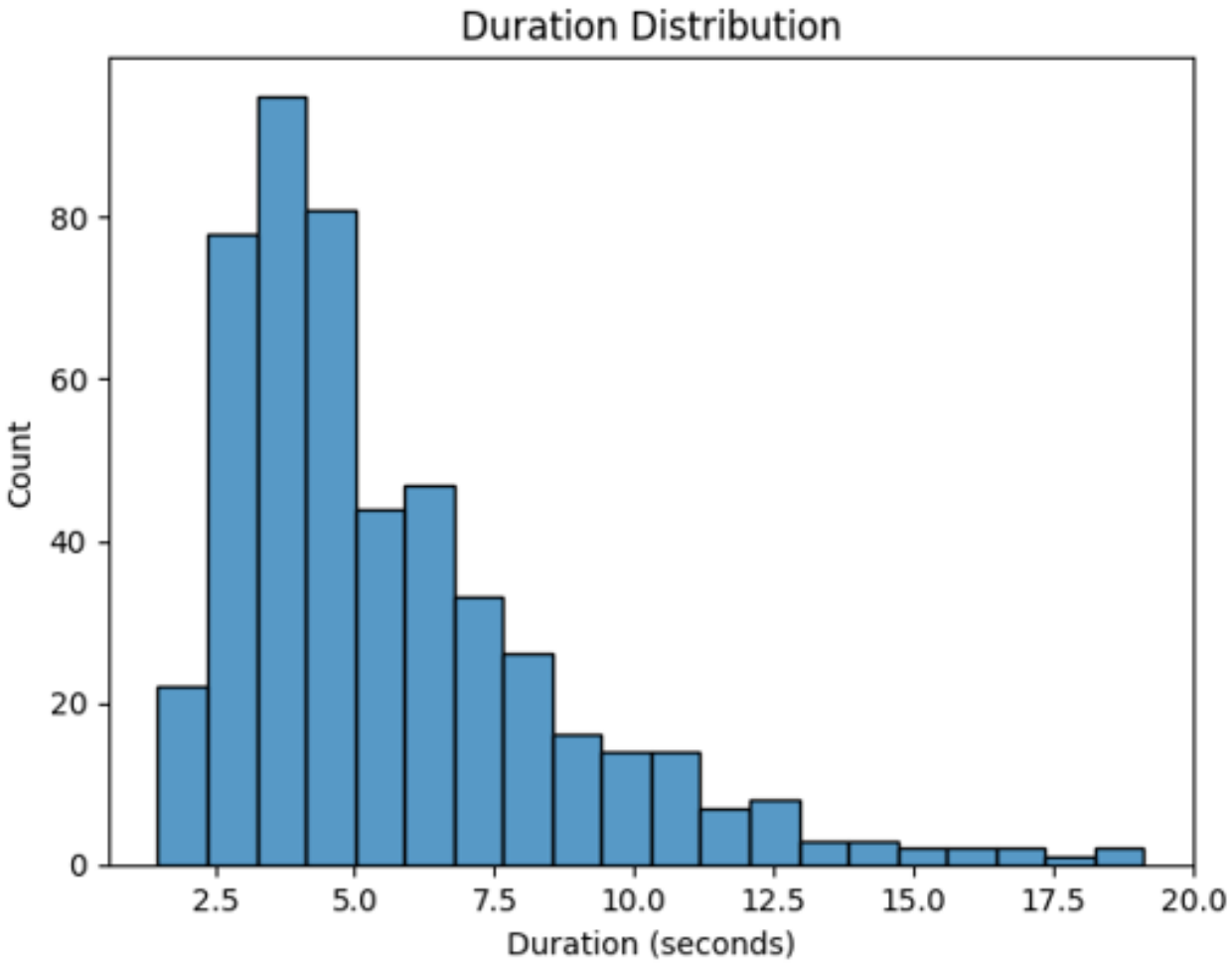


Figure 2: Distribution of video durations in the 9-class subset.

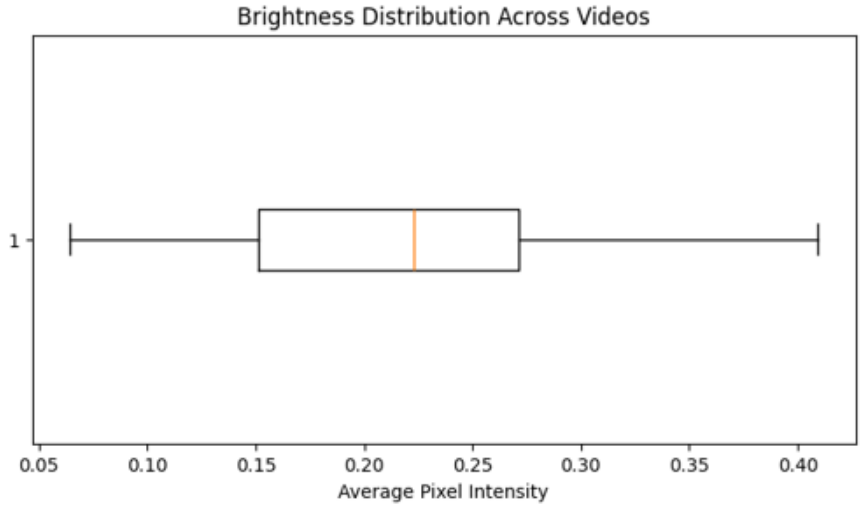


Figure 3: Brightness Distribution Across Videos

1.2.2 Brightness Distribution

Figure 3 shows the average brightness (mean pixel intensity) per video. Most videos fall in the 0.15–0.25 range, with a slight left skew. This reflects the dataset's real-world nature—many clips are user-generated under natural lighting, leading to darker footage.

1.2.3 Pixel Intensity Histogram

Figure 4 reveals a highly polarized intensity distribution: a spike near 0 (dark pixels) and another near 1 (very bright), with sparse midtones. This high-contrast, low-variance trend may challenge models in capturing subtle gradients and textures.

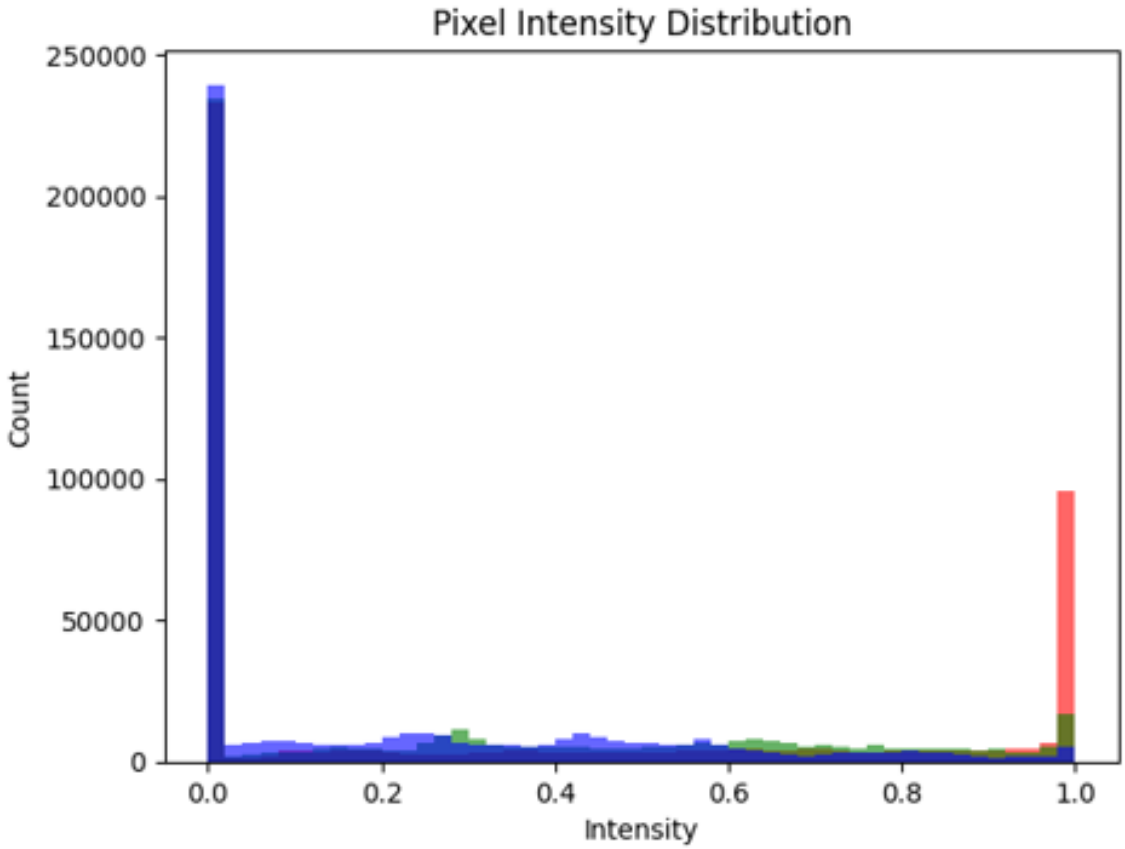


Figure 4: Pixel Intensity Distribution.

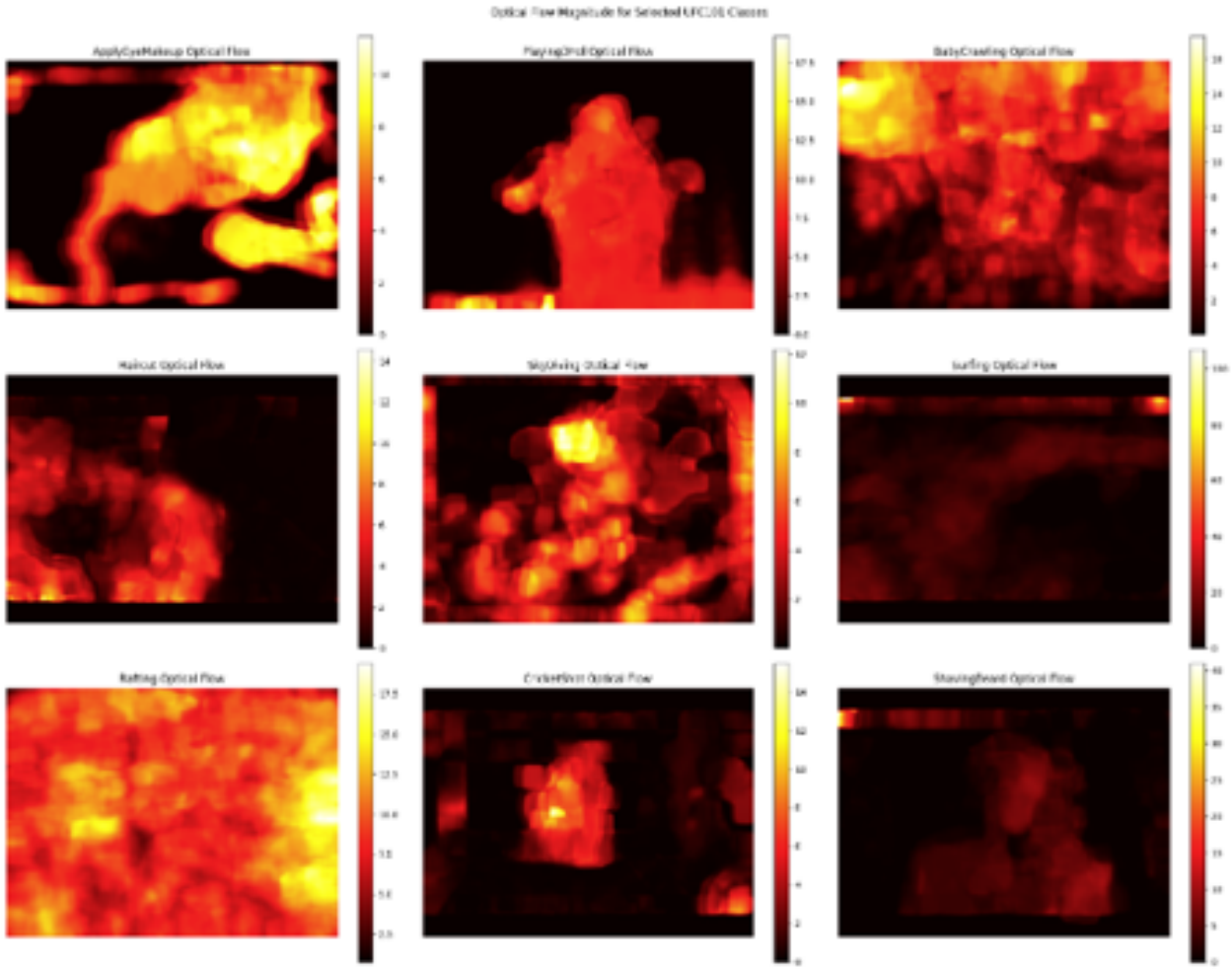


Figure 5: Optical flow magnitude across classes

1.3 Visual Dynamics and Motion

1.3.1 Optical Flow Analysis

Figure 5 presents average flow magnitude heatmaps across the 9 classes. We observe that classes like *Rafting*, *Skydiving*, and *Playing Dhol* exhibit high motion (bright activations), whereas *Apply Eye Makeup* and *Shaving Beard* involve minimal temporal change. This supports our decision to test both **temporal models (LSTM, (2+1)D)** and **static frame baselines**.

1.3.2 RGB Channel Distributions by Class

Figure 6 displays the normalized histograms for red, green, and blue channels across each class. Color usage varies by context:

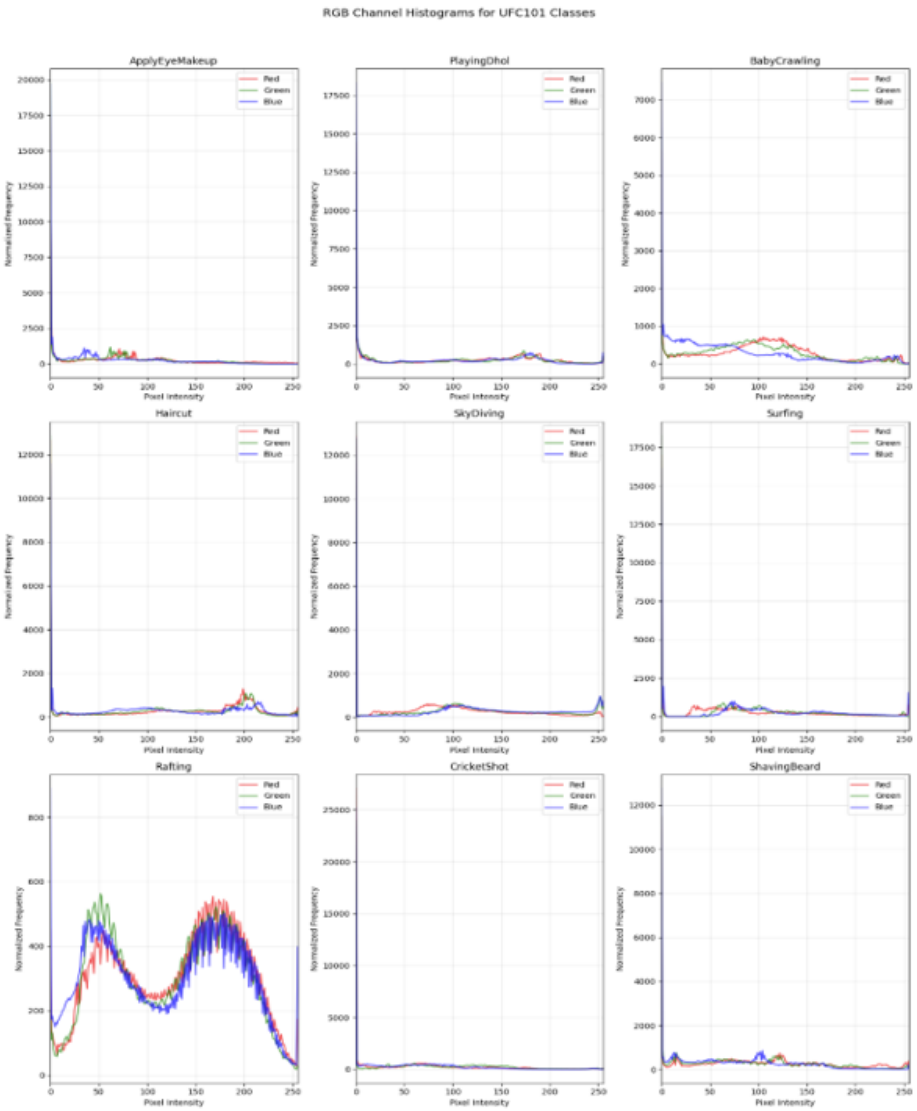


Figure 6: Normalized RGB Channels Histogram across classes

- *Rafting* has strong peaks in all channels—likely due to dynamic outdoor water scenes.
- *Haircut* and *Shaving Beard* skew toward lighter tones, indicative of indoor lighting.
- Other classes remain muted, reinforcing the brightness distribution noted earlier.

1.4 Summary

This analysis suggests the dataset presents a wide spectrum of challenges:

- Short and skewed video lengths
- Low-brightness, high-contrast visual distribution
- Varied motion intensity across classes

- Real-world color and lighting conditions

These factors validate the need for temporal modeling beyond simple frame-level classification and support experimentation with both sequential and spatiotemporal neural architectures. While we did not implement a formal static-frame baseline, it would be valuable in future work to quantify how much performance gain is attributable to temporal modeling. This could involve evaluating a model that classifies videos using only a single representative frame or pooled framewise features from a 2D CNN. Though it was intuitive to assume temporal models would dominate, an explicit comparison could strengthen the case for added complexity.

2 Model Training and Evaluation

2.1 Input Representation and Temporal Sampling Strategy

Video classification differs fundamentally from image classification in that it requires capturing patterns **over time**, not just in single frames. As such, a careful strategy was adopted for sampling, formatting, and resizing video data before feeding it into the models.

2.1.1 Fixed-Length Temporal Sampling [1]

Each video is represented as a sequence of **10 evenly spaced frames**, regardless of original length. This fixed-length approach ensures uniform input shapes, allowing for efficient batch processing and avoiding variable sequence lengths that would complicate model training.

Frames are selected using the following logic:

- A `frame_step` is applied (e.g., every 15th frame), ensuring broad coverage across the video.
- A **random starting point** is chosen to introduce variation across training epochs (like data augmentation).
- If a video is too short, **temporal padding** is applied by repeating edge frames to reach the required length.

This sampling process ensures that:

- Each clip captures the temporal progression of an action,
- Spatial integrity is maintained by working with full frames,
- And augmentation occurs naturally through frame shuffling.

2.1.2 Why Resizing, Not Pooling

To standardize inputs, each frame is resized to a fixed resolution (typically **224×224** or **112×112**, depending on the model). This is done *before* model input, rather than relying solely on internal pooling layers.

This resizing step is **especially important for (2+1)D CNNs**, where frames are processed as part of a 5D tensor `[batch, time, height, width, channels]`. These models operate jointly on spatial and temporal dimensions using 3D convolutions, so input resolution must be consistent across both axes. Resizing ensures proper alignment of spatiotemporal features and compatibility with fixed kernel sizes across depth and width/height.

In contrast, the **2D CNN + LSTM** model treats each frame independently using a `TimeDistributed` wrapper. This wrapper applies the same 2D CNN to each frame separately (in parallel), and the output features (one per frame) are then passed to the LSTM. Since each frame is processed independently in this setup, resizing is not strictly required at the video level—although we still standardize frame dimensions for consistency.

2.1.3 Frame Format and Data Pipeline

Each sampled frame is:

- Converted to `float32` and normalized,
- Resized with padding to maintain aspect ratio (`resize_with_pad`),
- Reassembled into a 5D tensor of shape `[batch_size, time, height, width, channels]`.

For training, a `FrameGenerator` class is used to yield `(video_tensor, label)` pairs dynamically. The data is wrapped in TensorFlow's `tf.data.Dataset` API with caching, shuffling, batching, and prefetching for efficient loading.

2.1.4 Comparison to Static Image Models

Unlike typical image classifiers, which process one input at a time, this pipeline explicitly preserves **sequence ordering**, allowing downstream models (LSTM, (2+1)D CNN) to learn temporal dynamics. No temporal pooling or averaging is performed, preserving action continuity.

Although we did not include a static-frame baseline, future work may compare this pipeline to a simplified model that classifies based on a single frame or pooled frame-level features.

2.2 Model Architectures

This section details the four video classification models implemented and evaluated in this project. Each architecture is designed to process short clips of 10 frames, capturing both spatial and temporal information in different ways. The key distinction across models lies in how they encode temporal relationships: sequentially (via LSTM) or jointly (via spatiotemporal convolutions).

2.2.1 2D CNN + LSTM (VGG-style and ResNet-style)

This architecture (Figure 7) consists of two components:

1. A **2D CNN backbone** (either VGG-style or ResNet-style) that extracts spatial features from each frame,
2. Followed by a **Long Short-Term Memory (LSTM)** network that models temporal dependencies across frames.

Each frame is processed independently by the CNN using a `TimeDistributed` wrapper, which applies the same feature extractor to each frame in parallel. This yields a sequence of feature vectors, one per frame, which are passed to the LSTM. The LSTM captures how features evolve over time and outputs a final embedding, which is passed through a dense classification head.

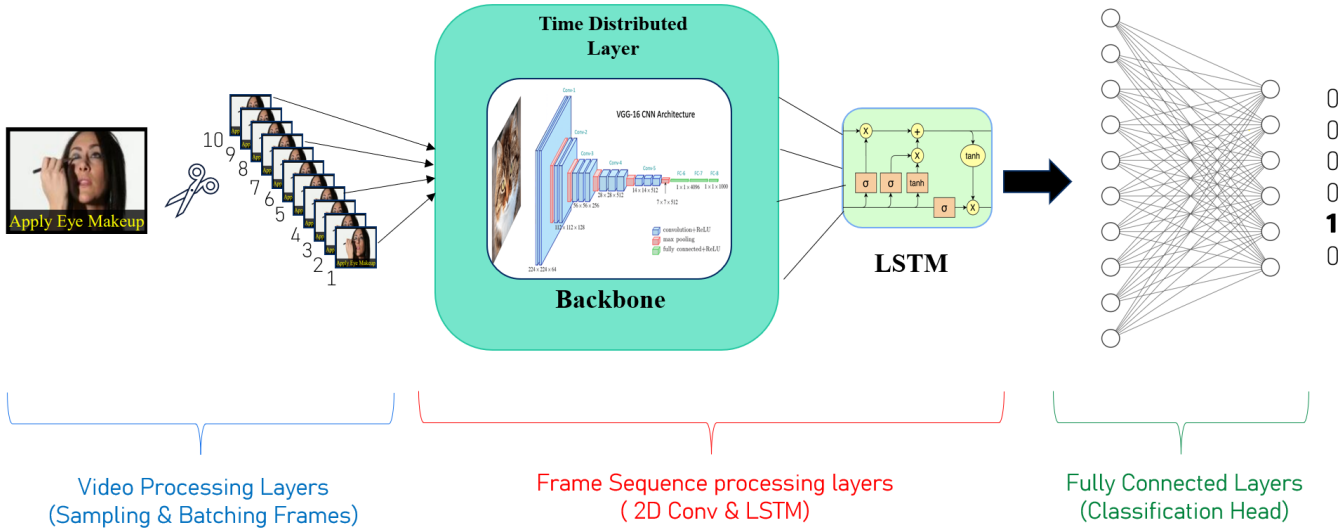


Figure 7: TimeDistributed CNN + LSTM pipeline.

- **VGG-style CNN** (Figure 8) uses 10 layers of Conv → ReLU → BatchNorm blocks with intermittent max-pooling, ending in global average pooling. It outputs a 512-D feature vector per frame.

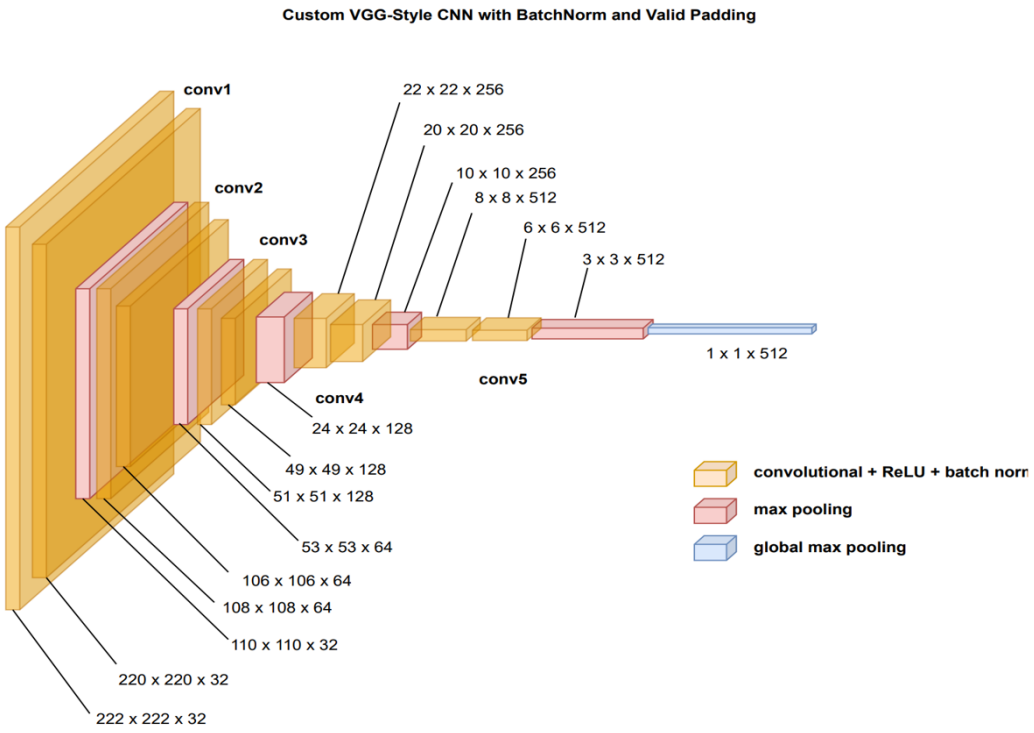


Figure 8: VGG style backbone architecture

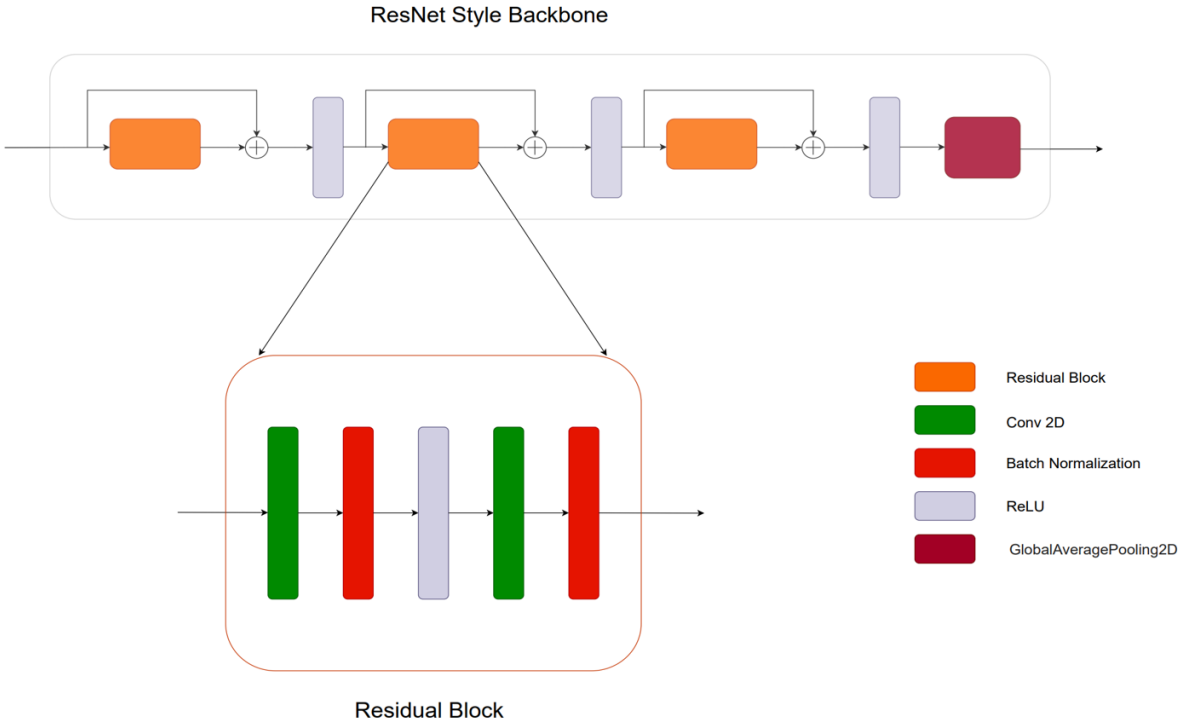


Figure 9: ResNet style backbone architecture

- **ResNet-style CNN** (Figure 9) uses 3 stacked residual blocks with identity connections, making it lighter and more generalizable.

2.2.2 (2+1)D Convolutional Model

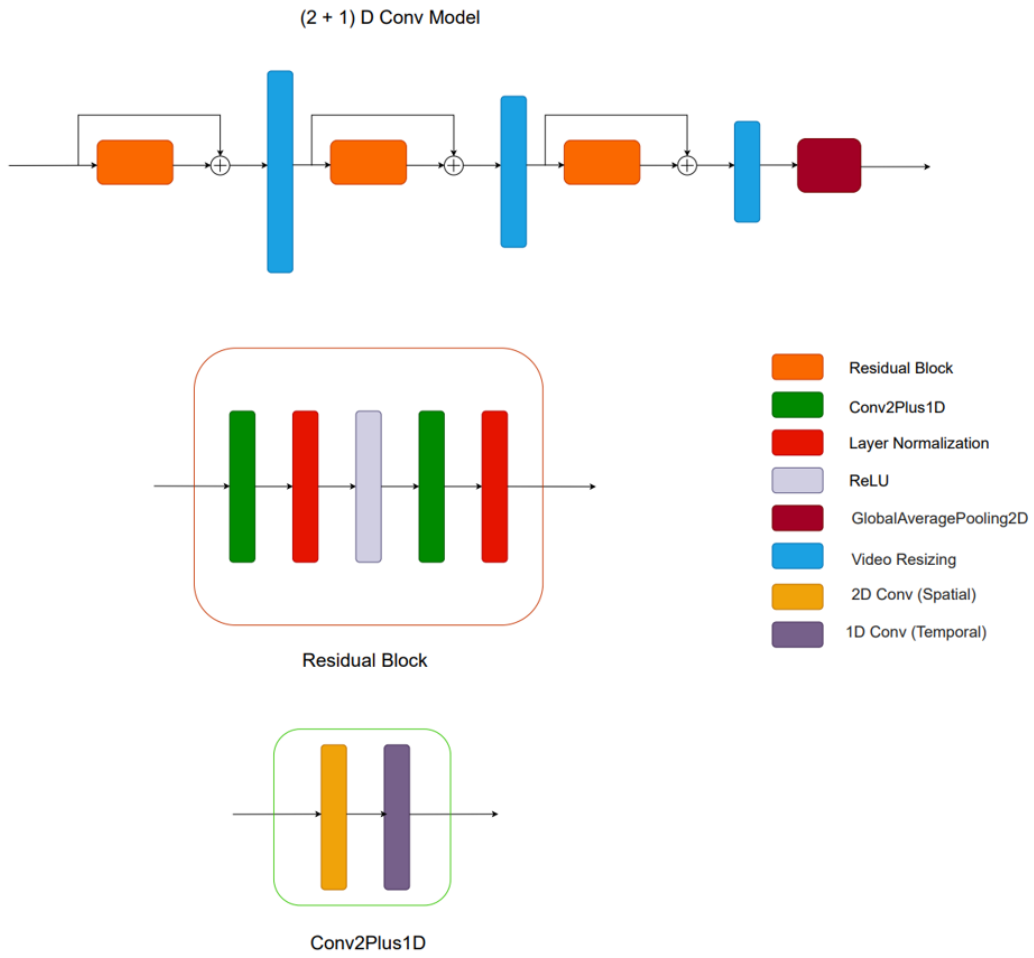


Figure 10: (2+1)D Conv pipeline.

The second family of models uses a **spatiotemporal convolutional architecture** based on (2+1)D decomposition — where a 3D convolution is factorized into:

- A **2D spatial convolution** followed by a
- **1D temporal convolution**.

This decomposition reduces parameter count and improves training stability while retaining the ability to model temporal structure directly within the convolutional layers.

The model consists of:

- An initial Conv2Plus1D block,

- Several stacked **residual blocks**, each using Conv2Plus1D,
- Periodic **video resizing layers** to downsample spatial resolution while keeping temporal resolution intact,
- A final global average pooling and dense classification layer.

This model processes inputs of shape `[batch, time, height, width, channels]`, and benefits from the uniform temporal sampling and pre-resizing discussed earlier.

2.2.3 Channel-Separated (2+1)D Convolution

parameter sharing.

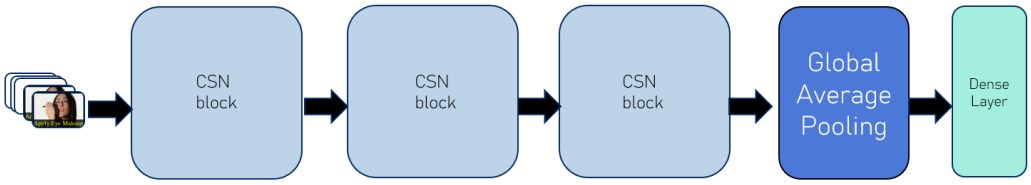


Figure 11: Channel-Separated (2+1)D convolutional model.

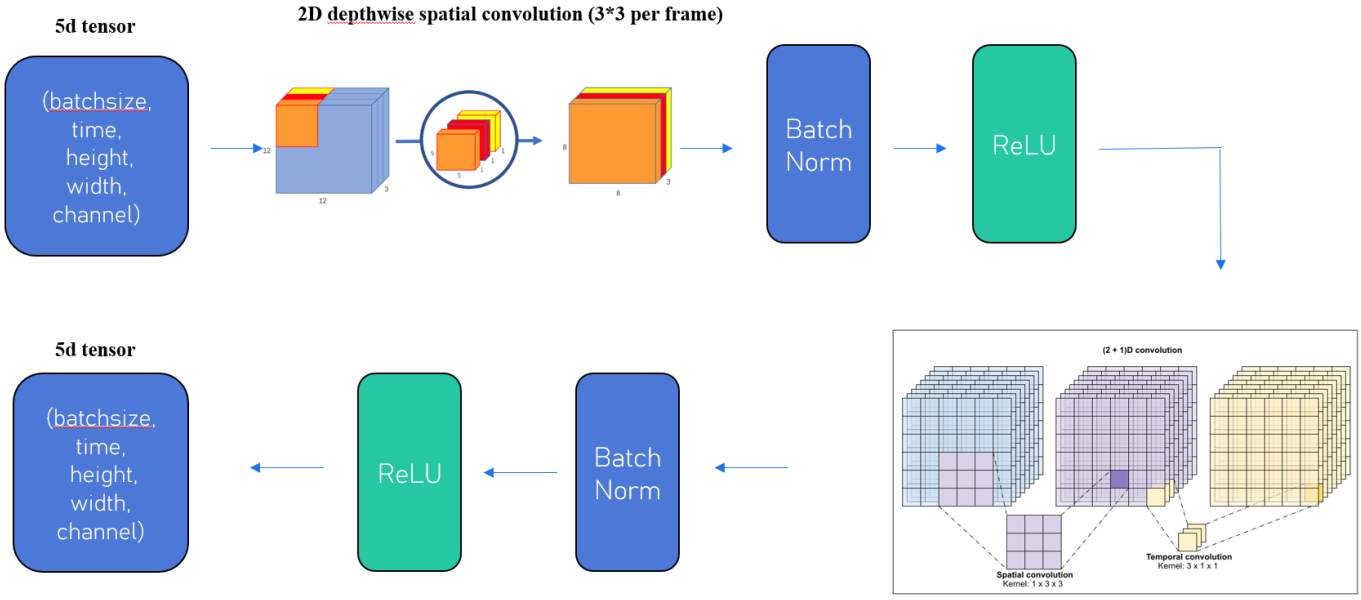


Figure 12: CSN Block.

We also experimented with a variant of the (2+1)D model using **Channel-Separated Networks (CSN)**. This approach further reduces redundancy by decoupling spatial and temporal convolutions across input channels, promoting efficient

Although lighter in compute, this model struggled with convergence on our 9-class subset and did not significantly outperform the base (2+1)D model.

2.2.4 Summary of Design Choices

Table 1: Comparison of model architectures and performance.

Model Type	**Temporal Modeling**	**Spatial Encoder**	**Params**	**Accuracy (Test)**
2D CNN + LSTM (VGG)	Sequential (LSTM)	VGG-style CNN	˜21M	90%
2D CNN + LSTM (ResNet)	Sequential (LSTM)	ResNet-style CNN	˜2.7M	77%
(2+1)D CNN	Joint spatiotemporal	ResNet + Conv2+1D	˜1.69M	79%
CSN	Channel-separated Conv	Conv2+1D variant	˜1.7M est.	Lower, unstable

2.3 Training Setup and Evaluation

2.3.1 Training Configuration

All models were trained using standardized 10-frame clips resized to 224×224, using a curated 9-class subset from the UCF101 dataset. Temporal padding was used for short videos, and sampling was done consistently across models.

- **Loss**: `SparseCategoricalCrossentropy(from_logits=True)`
- **Optimizer**: Adam (LR = 1e-4)
- **Batch Size**: 2
- **Epochs**: ˜30–50 (with early stopping)
- **Split**: 70% train / 15% validation / 15% test

2.3.2 Model Evaluation

To evaluate the performance of each model, we computed **per-class accuracy, precision, recall, and F1-score** using test set predictions. This approach helps us understand where each model excels or struggles, especially across classes with varying visual and motion complexity.

Each model's performance is summarized in the tables below:

Table 2: VGG + LSTM: Per-Class Evaluation

Class	Accuracy	Precision	Recall	F1 Score
ApplyEyeMakeup	1.000	1.000	1.000	1.000
BabyCrawling	0.990	0.913	1.000	0.955
CricketShot	0.995	1.000	0.962	0.980
Haircut	0.969	1.000	0.700	0.824
PlayingDhol	0.990	0.929	1.000	0.963
Rafting	1.000	1.000	1.000	1.000
ShavingBeard	0.990	0.926	1.000	0.962
SkyDiving	0.990	0.941	0.941	0.941
Surfing	0.985	0.905	0.950	0.927

Table 3: ResNet + LSTM: Per-Class Evaluation

Class	Accuracy	Precision	Recall	F1 Score
ApplyEyeMakeup	0.913	0.594	0.826	0.691
BabyCrawling	0.954	0.800	0.762	0.780
CricketShot	0.990	1.000	0.923	0.960
Haircut	0.923	0.632	0.600	0.615
PlayingDhol	0.913	0.655	0.731	0.691
Rafting	0.974	0.842	0.889	0.865
ShavingBeard	0.949	1.000	0.600	0.750
SkyDiving	0.980	0.810	1.000	0.895
Surfing	0.985	1.000	0.850	0.919

Table 4: (2+1)D Conv: Per-Class Evaluation

Class	Accuracy	Precision	Recall	F1 Score
ApplyEyeMakeup	0.934	0.708	0.739	0.723
BabyCrawling	0.939	0.667	0.857	0.750
CricketShot	0.980	0.893	0.962	0.926
Haircut	0.918	0.600	0.600	0.600
PlayingDhol	0.964	0.852	0.885	0.868
Rafting	0.990	1.000	0.889	0.941
ShavingBeard	0.918	0.765	0.520	0.618
SkyDiving	0.954	0.786	0.647	0.710
Surfing	0.990	0.909	1.000	0.952

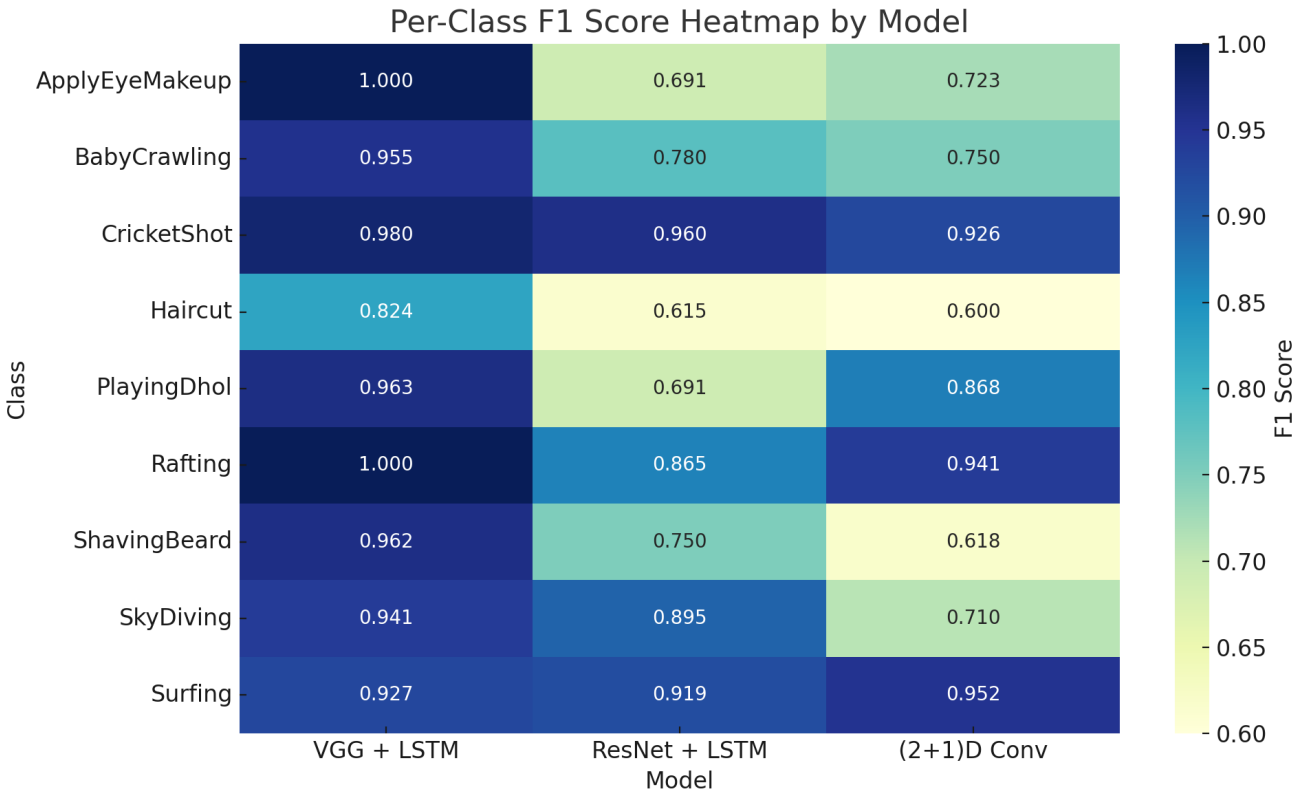


Figure 13: F1 Score Heatmap Across Models

3 Model Operations

3.1 Deployment Architecture

To enable real-world use of our video classification system, we propose a modular deployment pipeline. The architecture supports real-time or batch inference and is designed to accommodate both edge (mobile) and cloud-based environments.

3.1.1 Proposed Inference Pipeline

1. **Input Interface**
   - Accepts short video clips (5–10 seconds), uploaded or captured live.
   - Automatically resizes frames and samples 10 frames per video.
2. **Preprocessing Module**
   - Uses the same temporal sampling and resizing logic as during training.
   - Normalizes frames and prepares inputs in the correct tensor shape:
     - For (2+1)D: `[batch, time, height, width, channels]`
     - For LSTM: frame features in `[batch, time, features]`
3. **Model Server (via TensorFlow SavedModel or ONNX)**
   - The trained model is exposed via an API (REST or gRPC).
   - Supports swapping models (e.g., VGG-LSTM, ResNet-LSTM, or (2+1)D) without client-side change.
4. **Output Interface**
   - Returns top-k predicted action labels with probabilities.
   - Optionally renders predictions overlaid on video or sampled keyframes.

3.2 Deployment Targets

Table 5: Deployment targets and feasibility for different platforms.

Platform	**Feasibility**	**Model**	**Notes**
Web / Desktop	Easy	Any	Requires Flask, FastAPI, or Streamlit
Mobile (Edge)	**Feasible**	(2+1)D or ResNet-LSTM	Must optimize/export via TFLite
Cloud (Batch)	Ideal	All models	Good for processing large video sets

3.3 Model Maintenance and Updating

To ensure long-term utility and robustness, we propose a lightweight maintenance plan:

- **Monitoring**: Track model confidence distribution over time. Sudden shifts may indicate drift.
- **Feedback loop**: In production settings, allow users to flag incorrect predictions for periodic re-labeling and retraining.
- **Model versioning**: Store model versions with metadata (training config, data stats) for reproducibility.
- **Re-training triggers**:
  - Drop in performance on validation set
  - New action classes added
  - Change in input resolution or quality

4 Neural Style Transfer for Video Stylization

In addition to our core focus on video classification, we explored neural style transfer as a secondary task to extend our video processing pipeline into the domain of creative visual transformations. Our goal was to generate stylized video frames that retain the semantic structure of the original content while adopting the color palette, brushstroke patterns, and textures of a target artwork. This application aligns with growing interest in the intersection of deep learning and digital art.

4.1 Methodology

We implemented and compared two complementary approaches:

(1) Optimization-based style transfer [10]:

This classical technique follows the framework introduced by Gatys et al. [10]. The method extracts high-level *content features* and low-level *style features* using a pretrained convolutional neural network, specifically VGG19, whose intermediate activations effectively encode structural and texture information.

Given a content image C, a style image S, and a generated image G, we define a composite loss function:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}}(C, G) + \beta \mathcal{L}_{\text{style}}(S, G)$$

where:

- $\mathcal{L}_{\text{content}}$ measures the Euclidean distance between high-level activations of C and G from a deep VGG19 layer.

- $\mathcal{L}_{\text{style}}$ compares Gram matrices of multiple convolutional layers between S and G, capturing correlations that represent texture and style.

In our application, we sampled frames from a user-uploaded video using `ffmpeg`, applied the optimization procedure to each frame individually, at a user defined frame sampling frequency. Although this method lacks temporal awareness, it preserved spatial content faithfully and reduced perceptual jitter between frames.

Streamlit App Integration: We developed a user-friendly interface via `Streamlit` that guides the user through uploading a content video and a style image. The backend performs the following pipeline:

1. Extract frames from the video using `ffmpeg`.
2. Load and cache a pretrained fine-tuned VGG19 model in TensorFlow.
3. Extract style features from the user-provided artwork.
4. Iteratively optimize each frame to minimize the total loss defined above.
5. Reconstruct the stylized video by stitching processed frames using `ffmpeg`.

This pipeline enabled efficient experimentation with different artistic styles and content videos while offering full control over the stylization parameters.

(2) CycleGAN [9]:

CycleGAN is a generative framework introduced by Zhu et al. [9] for unpaired image-to-image translation. Unlike optimization-based approaches, it does not require paired examples of source and target domain images. Instead, it simultaneously trains two generators and two discriminators in a cycle-consistent structure that enforces both realism and content preservation.

Given two image domains X and Y, the model learns mappings $G : X \to Y$ and $F : Y \to X$, alongside discriminators D_Y and D_X that evaluate the realism of generated images.

In our implementation, we utilized a pretrained generator $G : X \to Y$ trained on unpaired image sets. For inference, video frames were extracted using `ffmpeg`, resized and normalized, and passed through the generator to produce stylized outputs. Each frame was processed independently to simulate artistic transformation (e.g., photo $\to$ painting). The model exhibits very fast inference time and effectively modifies the color and texture of input frames. However, because the pretrained generator was trained on a target domain that is semantically distant from our input video content (e.g., artistic images versus real-world human action footage), the visual transformation is limited in perceptual impact. The results are not consistently successful, particularly in preserving stylistic coherence, due to the domain gap between training data and test frames.

4.2 Results

Figure 14 shows example frames processed using both methods. The optimization-based method yields sharper, semantically accurate results at the cost of speed. CycleGAN offers stylization speed, but lacks frame-to-frame coherence.

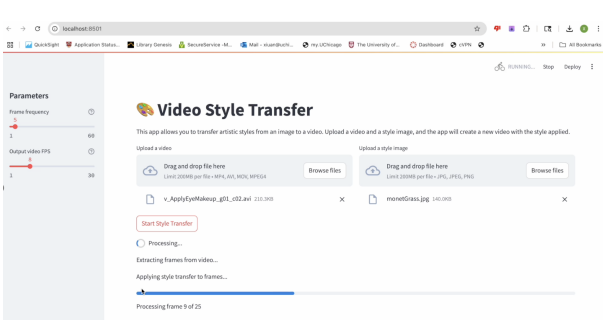
(a) Original Frame

(b) Optimized Style Transfer

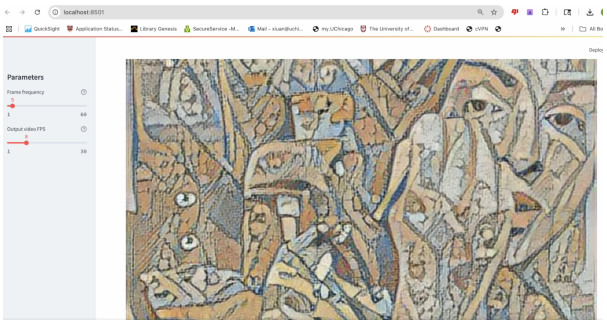
(c) CycleGAN

Figure 14: Comparison of frame-wise style transfer.

Figure 15 shows the user interface of our Streamlit-based style transfer application. The app allows users to upload a content video and a target artwork image, and then apply neural style transfer using optimization-based method.



(a) Upload Interface



(b) Final Video Interface

(c) Input: Style Picture

(d) Input: Original Video

Figure 15: Streamlit app interface, with input and output for video style transfer.

5 Conclusion

In this project, we explored the challenging task of human action classification in video using a curated subset of the UCF101 dataset. Our aim was to understand how different neural architectures handle temporal and spatial information in short video clips, and to evaluate their relative strengths and weaknesses.

We implemented and compared four models:

- Two **2D CNN + LSTM** pipelines with VGG-style and ResNet-style backbones,
- A **(2+1)D convolutional network** for joint spatiotemporal modeling,
- And a lightweight **Channel-Separated (2+1)D Conv** variant.

The results show that the **VGG + LSTM** model achieved the highest accuracy (90%) and macro F1 score (˜95%), but at the cost of model size and computational load. In contrast, the **(2+1)D Conv** model provided a competitive 79% accuracy with a much smaller footprint (1.69 MB), highlighting the effectiveness of spatiotemporal decomposition for compact video modeling. The **ResNet + LSTM** model, while more efficient than VGG,

showed slightly lower performance, revealing tradeoffs between representational capacity and parameter count.

Beyond classification, we experimented with **neural style transfer on video frames**, using both optimization-based methods and pretrained CycleGANs. The framewise approach yielded visually coherent results, suggesting promising applications in stylized video generation, while CycleGAN models struggled with temporal consistency.

Our project emphasizes the importance of carefully engineered input sampling strategies and frame preprocessing, particularly for models relying on temporal coherence. We also proposed a feasible deployment pipeline and outlined a plan for long-term model maintenance.

5.1 Future Work

Potential directions include:

- Training with the full UCF101 dataset to improve generalization,
- Testing on fine-grained or ambiguous action categories,
- Real-time inference on mobile devices,
- And extending the classification framework to more creative applications such as video segmentation or lip reading.

This work demonstrates not only technical implementation, but also the thought process and design decisions that make deep video models function effectively in real-world settings.

6 Appendix

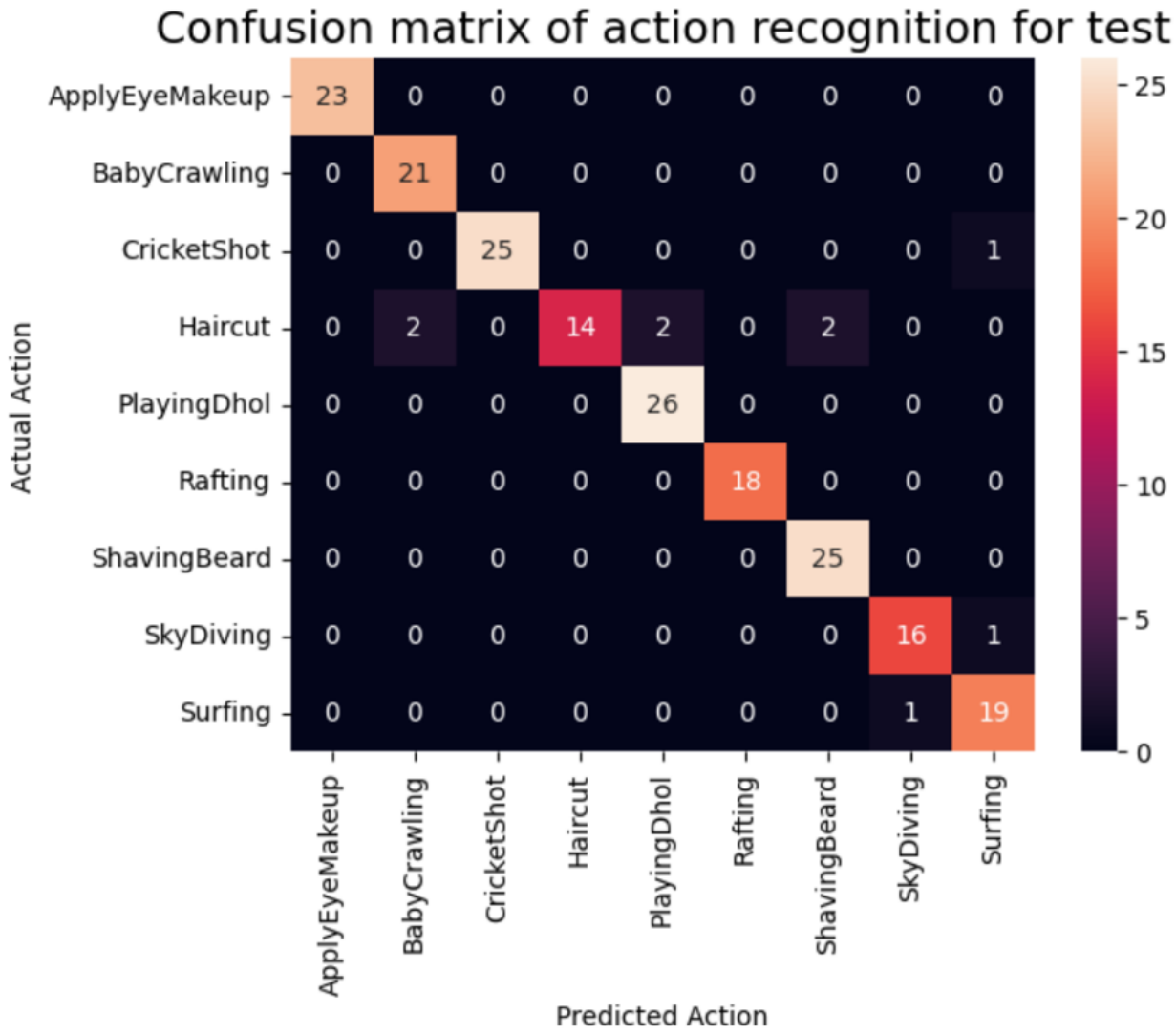


Figure 16: Confusion Matrix 2D Conv + LSTM (VGG Style)

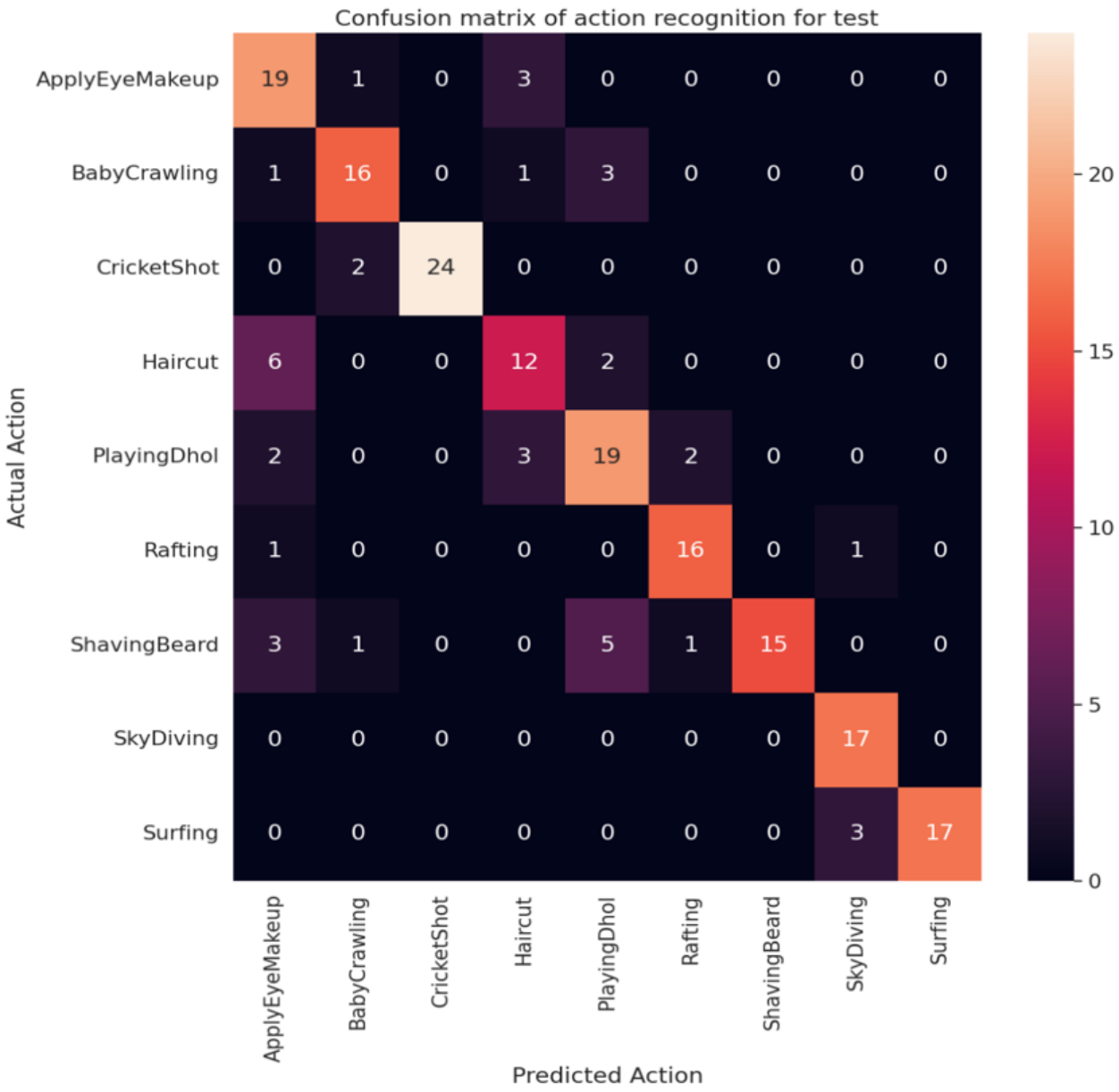


Figure 17: Confusion Matrix 2D Conv + LSTM (ResNet Style)

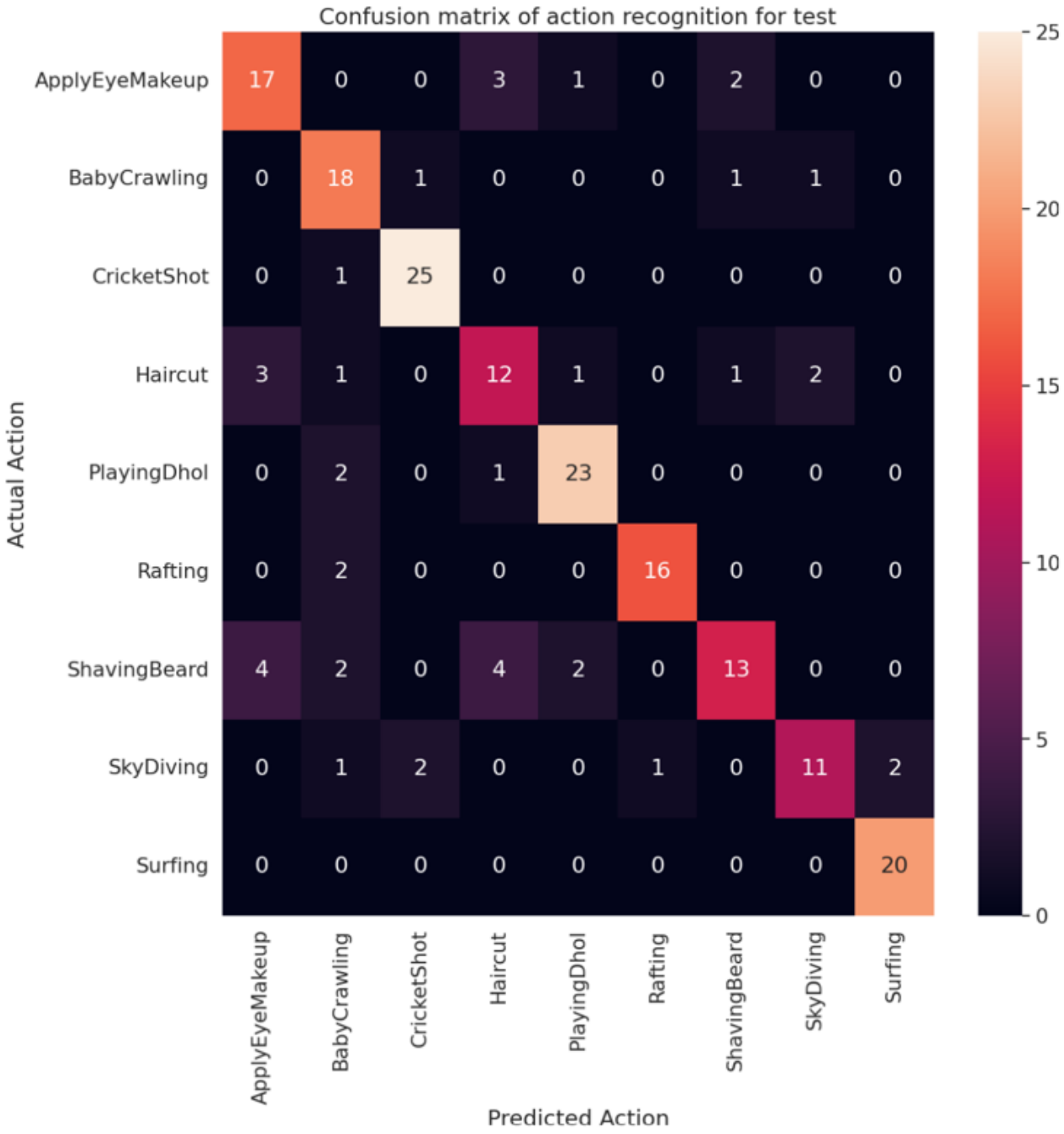


Figure 18: Confusion Matrix (2 + 1)D Conv

References

[1] TensorFlow. Action recognition with 3d cnns. `https://www.tensorflow.org/tutorials/video/video_classification`, 2021. Accessed: 2025-05-22.

[2] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[3] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[4] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6450–6459, 2018.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[7] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. *arXiv preprint arXiv:1904.02811*, 2019.

[8] Nicholas Kolkin, Jason Salavon, and Greg Shakhnarovich. Style transfer by relaxed optimal transport and self-similarity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[10] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.