Hand Gesture Recognition for Media Control Using a Subset of the Jester Dataset

Ishmanbir Singh
Georgia Institute of Technology
isingh96@gatech.edu

Magnus Hanno Voss
Georgia Institute of Technology
mvoss31@gatech.edu

Richie Jonathan
Georgia Institute of Technology
rjonathan6@gatech.edu

Abstract

Dynamic hand gesture recognition is a critical component for natural, touchless human-computer interaction, yet it remains challenging due to the temporal ambiguity of motion and high computational demands. This project addresses the task of real-time media control by classifying dynamic hand gestures from RGB video sequences. We propose a pipeline that compares varying temporal modeling strategies, specifically 3D-CNNs, CNN-LSTM hybrids, and R(2+1)D architectures, trained on a focused nine-class subset of the Jester dataset. Our core contribution lies in demonstrating that while lightweight recurrent models offer speed, heavy spatiotemporal pretraining is essential for high-fidelity recognition. Experiments reveal that a ResNet-based R(2+1)D model, pre-trained on large-scale video data, achieves an accuracy of 97.85% and a macro F1-score of 0.9785, significantly outperforming a standard 3D-CNN baseline (93.44%) and a CNN-LSTM hybrid (81.89%). We further analyze failure modes, showing that factorized spatiotemporal convolutions effectively resolve direction-sensitive ambiguities (e.g., sliding up vs. down) critical for fluid media control.

1 Introduction

Recent advances in human–computer interaction (HCI) have accelerated the development of touchless interfaces that rely on hand gestures rather than physical controllers or voice input. Dynamic hand gesture recognition enables natural and accessible interaction across domains such as smart devices, AR/VR environments, and assistive technologies. This project develops a deep learning approach for dynamic hand gesture recognition aimed at controlling media applications through natural movements, with the goal of accurately classifying short video clips of hand gestures into specific command categories such as play, pause, skip, or volume adjustment. In doing so, we seek to advance hands-free, camera-based control systems that are both efficient and broadly deployable. Beyond media control, the proposed framework holds potential for applications in sign language recognition, human–robot collaboration, and non-verbal communication within immersive or accessibility-focused settings.

The central research question of this project is: *Which spatiotemporal architecture, a purely volumetric 3D-CNN, a CNN–LSTM hybrid, or an R(2+1)D network, provides the best trade-off between recognition performance and computational efficiency for nine-class media-control gesture recognition on a subset of the Jester dataset?* We define success as achieving at least 93% validation accuracy and a macro F1-score above 0.90 on the nine-class Jester subset, with comparable performance on the held-out test split, while maintaining near real-time inference capability (targeting under $50\,\mathrm{ms}$ per frame on a modern GPU). These criteria provide concrete, measurable targets for both predictive performance and deployability in interactive media-control scenarios.

The proposed method processes short RGB video clips containing 32–40 frames, each frame with an approximate dimension of $128 \times 128 \times 3$ after resizing and padding. The model outputs a single categorical prediction corresponding to one of nine media-control gestures (e.g., *Sliding Left*, *Zooming In*, *Stop Sign*), derived from a filtered subset of the 27-class Jester dataset [4].

Dynamic gestures pose greater challenges than static poses due to motion continuity and temporal ambiguity between similar classes. By comparing purely volumetric (3D-CNN), recurrent (CNN–LSTM), and factorized spatiotemporal (R(2+1)D) models under identical data and training conditions, this work aims to determine effective strategies for robust, real-time gesture recognition suitable for everyday media control applications, as quantified by the success criteria above.

2 Related Work

Gesture recognition lies at the intersection of computer vision and human–computer interaction, encompassing both *gesture classification* (semantic labeling) and *gesture estimation* (pose reconstruction). Dynamic gestures, which evolve over time, remain central to natural interaction tasks such as media control and sign language understanding [2].

Various sensing modalities have been explored. RGB-based vision systems are popular due to their simplicity but are sensitive to lighting and background variation. RGB-D sensors provide additional depth cues and improve spatial understanding [6], while skeleton-based approaches extract joint coordinates and model spatial dependencies using Graph Convolutional Networks (GCNs) [1, 3]. Although efficient, these methods discard fine visual details critical for distinguishing similar hand motions. In contrast, bounding-box-based RGB pipelines retain full spatial information.

Large-scale datasets have driven progress in this field. The Jester dataset [4] is a key benchmark with over 148,000 RGB clips across 27 gesture classes, offering strong cross-user diversity and consistency for deep model training. Other datasets, such as NVIDIA Dynamic Hand Gesture [2] and EgoGesture [9], focus on multimodal or egocentric settings but are smaller in scale. Jester's scope makes it particularly suited for our media-control scenario.

Modeling techniques have evolved from handcrafted motion descriptors and SVMs or HMMs to deep spatiotemporal models. Early deep learning approaches utilized 3D-CNNs (e.g., C3D) to learn spatiotemporal features directly from raw video volumes. While effective, standard 3D convolutions are computationally expensive and difficult to optimize due to the vast number of parameters. To mitigate this, hybrid architectures such as CNN–LSTMs were introduced, which decouple the process by using 2D CNNs for spatial feature extraction and recurrent layers (RNN/LSTM) for temporal aggregation [8, 5].

More recently, factorized convolutional architectures have emerged as a superior alternative to full 3D-CNNs. The R(2+1)D network [7] decomposes the computationally heavy 3D convolution ($k \times k \times k$) into two separate operations: a 2D spatial convolution ($1 \times k \times k$) followed by a 1D temporal convolution ($k \times 1 \times 1$). This factorization not only reduces the number of parameters and computational cost but also introduces additional non-linearities (ReLU) between spatial and temporal operations, facilitating the learning of more complex motion functions.

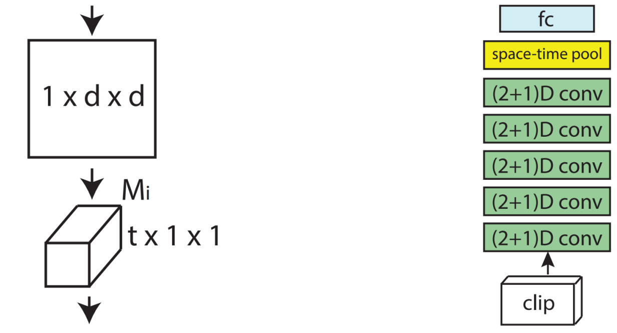
Our project evaluates these three distinct modeling paradigms, which are the direct spatiotemporal learning of 3D-CNNs, the decoupled spatial-recurrent approach of CNN-LSTMs, and the efficient factorized convolutions of R(2+1)D networks. By benchmarking these on a targeted subset of the Jester dataset, we aim to identify the optimal architecture for robust, real-time media control.

3 Method and Approach

Our objective is to build an effective and computationally efficient pipeline for dynamic hand-gesture recognition on the curated nine-class Jester subset described in Section 4. The approach integrates spatiotemporal deep learning architectures, standardized training procedures, and a hypothesis-driven experimental design.

3.1 Hypotheses

We formulate three hypotheses that guide our experimental evaluation.



(a) (2+1)D convolution (b) R(2+1)D

Figure 1: R(2+1)D networks, illustrated in b, use ResNet backbones with factorized (2+1)D convolutions. Each (2+1)D block decomposes a 3D convolution into a spatial 2D convolution followed by a temporal 1D convolution. In a, t denotes the temporal extent and d the spatial width and height [7].

H1: Explicit temporal modeling improves accuracy. Architectures with dedicated temporal structure, such as an R(2+1)D network with factorized spatiotemporal convolutions or a CNN-LSTM hybrid, should outperform a purely convolutional 3D-CNN baseline, particularly for gestures whose class identity depends on motion direction (e.g., up vs. down or zoom in vs. zoom out).

H2: A reduced gesture space improves generalization. Restricting the label space from 27 categories to nine carefully selected, semantically distinct gestures should simplify the task and reduce inter-class ambiguity, resulting in higher accuracy and better results in the confusion matrices.

H3: The CNN-LSTM hybrid offers the best accuracy and efficiency trade-off. Although R(2+1)D is expected to achieve the highest accuracy and F1-score due to deeper temporal modeling capacity, we hypothesize that the CNN-LSTM hybrid will yield competitive performance while using substantially fewer parameters, yielding reduced latency and faster training.

3.2 Model Architectures

We compare three architectures that differ in their treatment of spatial and temporal structure in video: a C3D-style 3D convolutional baseline, a CNN-LSTM hybrid, and an R(2+1)D network based on a torchvision backbone.

Our baseline is a 3D-CNN model. The C3D model processes inputs of shape $(B, 3, T, H, W)$ with $T = 32$ and $H = W = 128$ through five convolutional groups. The first group applies a $3\times3\times3$ convolution followed by BatchNorm3d, ReLU, and a MaxPool3d layer with stride $(1, 2, 2)$, reducing spatial resolution while preserving the full temporal extent. Subsequent groups increase channels to 128, 256, and 512 while downsampling the spatiotemporal volume via pooled strides of $(2, 2, 2)$. The final group applies two 512-channel 3D convolutions and a final pooling layer with stride $(1, 2, 2)$. The resulting tensor is flattened and passed through two fully connected layers of size 2048 with ReLU activations and dropout ($p = 0.5$), followed by a 9-way classifier. The architecture contains approximately 110M trainable parameters and learns motion implicitly via 3D convolutions.

Our second model is a CNN-LSTM Hybrid. This model factorizes spatial and temporal learning into separate stages. Per-frame features are extracted using a 2D ResNet-18 (without ImageNet pretraining) truncated before the final fully connected layer, yielding a 512-dimensional embedding for each frame. After reshaping the input to process all frames in parallel, the output is reassembled into a sequence of shape $(B, T, 512)$ and passed to a unidirectional two-layer LSTM with hidden dimension 256 and dropout $p = 0.5$. The final time-step output is used as a sequence summary and fed into a linear classifier. LSTM weights use Xavier initialization for input matrices, orthogonal initialization for recurrent matrices, and a forget-gate bias of 1. This hybrid model contains roughly 12.5M trainable parameters, making it substantially smaller than C3D.

Our third model is a R(2+1)D Network. The R(2+1)D architecture which is shown in Figure 1 replaces each 3D convolution with a spatial $(1\times3\times3)$ convolution followed by a temporal $(3\times1\times1)$ convolution, increasing the number of nonlinearities and easing optimization. Concretely, we use the `r2plus1d_18` implementation from `torchvision.models.video`, which follows a ResNet-18-style residual backbone whose 3D convolutions are factorized into 2D spatial and 1D temporal blocks. We evaluate both a randomly initialized variant and a version initialized with Kinetics-400 pretrained weights (`R2Plus1D_18_Weights.DEFAULT`). The original 400-way classifier is replaced with a 9-class linear layer. Across configurations, the model contains approximately 31M trainable parameters, offering a middle ground between CLSTM and C3D while encoding a stronger temporal inductive bias.

3.3 Training Procedure

All models operate on 32-frame RGB clips resized to 128×128, represented as tensors of shape $(B, 3, 32, 128, 128)$. Training is fully supervised with a cross-entropy loss. Training is performed in PyTorch using `DistributedDataParallel` (DDP) on four NVIDIA H200 GPUs (141 GB HBM3e each). Processes are launched with `torchrun`, each bound to a single GPU. We use per-GPU batch sizes of 16 for C3D (global batch size 64) and 32 for CLSTM and R(2+1)D (global batch size 128), enabled by the smaller memory footprint of the latter models.

Data loading uses `DistributedSampler` with seed 42 and epoch-wise reshuffling. Optimization is performed with Adam at a fixed learning rate of 1×10^{-4} and default momentum parameters. No learning-rate scheduling or weight decay is used. Regularization consists of dropout $(p = 0.5)$ in the fully connected and LSTM layers and spatial data augmentation: random cropping, color jitter, and a small random in-plane rotation applied per frame. We do not apply temporal jitter or temporal reordering, to preserve the original gesture dynamics.

Each architecture follows a matching schedule: 20 epochs for non-augmented C3D, 30 epochs for augmented C3D and scratch R(2+1)D, 50 epochs for CLSTM, and 10 epochs for pretrained R(2+1)D fine-tuning. After each epoch, rank 0 logs metrics and saves a checkpoint containing configuration metadata and model states. When resuming training, the latest checkpoint is validated and restored automatically.

3.4 Evaluation Protocol and Reproducibility

All final models are evaluated on the held-out test set of 2,700 clips (300 per class). We report overall accuracy, macro-averaged precision, recall, and F1-score. Per-class F1-scores and both raw and row-normalized confusion matrices are computed to analyze systematic confusions, particularly among the four sliding gestures and the two zooming gestures.

Reproducibility is ensured through fixed random seeds, deterministic data splits, a consistent preprocessing pipeline, and version-controlled training and evaluation scripts. Configuration for each run is stored in the corresponding checkpoint and (when used) Weights & Biases metadata.

3.5 Inference Efficiency Across Hardware

To assess deployability, we benchmark inference latency across three hardware backends: (i) CPU inference on an Apple M3 Pro, (ii) GPU inference on the same device via Apple's Metal Performance Shaders (MPS), and (iii) CUDA inference on a single NVIDIA H200 GPU. For each model, we run 100 forward passes on a single $(1, 3, 32, 128, 128)$ clip and report the mean latency and corresponding frames per second (FPS) on each device.

The C3D model, dominated by large 3D convolutions, achieves extremely high throughput on CUDA ($\sim$2.3 ms per clip) but is substantially slower on CPU and MPS. The CNN-LSTM hybrid, built primarily from 2D convolutions and LSTM layers, shows the most consistent performance across devices and is therefore well-suited for non-CUDA platforms. The R(2+1)D model lies between these extremes: it benefits strongly from CUDA acceleration but incurs overhead on CPU/MPS due to sequential spatial and temporal convolution kernels.

These measurements complement our accuracy evaluations by showing that the most accurate model (pretrained R(2+1)D) is not necessarily the most deployable, highlighting important trade-offs for real-time gesture recognition.

4 Data

All experiments use a curated subset of the Jester dataset [4], a large-scale collection of 148,092 RGB hand-gesture clips recorded at 12 FPS from a fixed first-person viewpoint. Each video contains 25–50 frames (height 100 px) and belongs to one of 27 gesture classes covering directional motions, finger gestures, zooming actions, and several ambiguous "other" categories.

Each sample consists of a folder of sequential JPEG frames. For this project we restrict the label space to nine gestures relevant for media-control tasks: *Doing Other Things*, *No Gesture*, *Sliding Two Fingers Down/Left/Right/Up*, *Stop Sign*, *Zooming In With Full Hand*, and *Zooming Out With Two Fingers*. This removes several noisy or visually overlapping categories and yields a clearer, more focused classification setting.

To build a balanced dataset, we sample 1000 training, 30 validation, and 300 test clips per class using a fixed seed. After filtering and preprocessing, the subset contains 441,855 frames (3.18 GB) with perfectly uniform class frequencies across splits. Preprocessing enforces temporal and spatial consistency. We discard clips with fewer than 32 frames, and for all remaining clips we uniformly sample exactly 32 frames from the original sequence, so that every model sees fixed-length inputs. Each frame is resized to a height of 128 px while preserving aspect ratio, then center-cropped or padded to 128×128 and normalized to $[0, 1]$. Validation and test examples use deterministic preprocessing with no augmentation. Training uses data augmentation, consisting of random cropping, color jitter, and slight random rotation, to maintain gesture semantics while improving robustness, without modifying the temporal ordering of frames.

Although the dataset contains many subjects, it is filmed entirely in controlled indoor environments with consistent lighting and camera placement, and some gestures (e.g. *Doing Other Things*) are inherently ambiguous. These characteristics contribute to several of the failure modes discussed in Section 5.

5 Experiment and Results

Our experiments compare five model configurations trained on the curated nine-class Jester subset introduced in Section 4. All models use identical data splits, preprocessing, and supervised learning objectives. The evaluated configurations are: (1) a 3D-CNN baseline (20 epochs, no augmentation), (2) the same 3D-CNN with mild augmentation (30 epochs), (3) a CNN-LSTM hybrid (50 epochs), (4) an R(2+1)D model trained from scratch (30 epochs), and (5) a pretrained R(2+1)D model fine-tuned for 10 epochs.

5.1 Quantitative Performance

Table 1 summarizes accuracy, macro F1, parameter count, and CUDA latency. The 3D-CNN baseline performs surprisingly well, reaching 93.44% accuracy. Using data augmentation increases performance to 94.11%, confirming that even mild augmentation benefits spatiotemporal models.

Model	Accuracy (%)	Macro F1
3D-CNN (no aug, 20 ep.)	93.44	0.9343
3D-CNN (+aug, 30 ep.)	94.11	0.9411
CNN-LSTM (50 ep.)	81.89	0.8203
R(2+1)D (scratch, 30 ep.)	92.19	0.9222
R(2+1)D (pretrained, 10 ep.)	**97.85**	**0.9785**

Table 1: Performance of all models on the nine-class Jester subset.

The CNN-LSTM hybrid is the weakest model, achieving only 81.89% accuracy. Its confusion matrix, shown in Figure 2 shows widespread mixing between the four sliding directions and frequent misclassification of *Doing Other Things*.

The scratch-trained R(2+1)D model performs comparably to the 3D-CNN but does not surpass it. In stark contrast, the pretrained R(2+1)D achieves 97.85% accuracy and a macro F1 of 0.9785, by far the best results in our experiments.

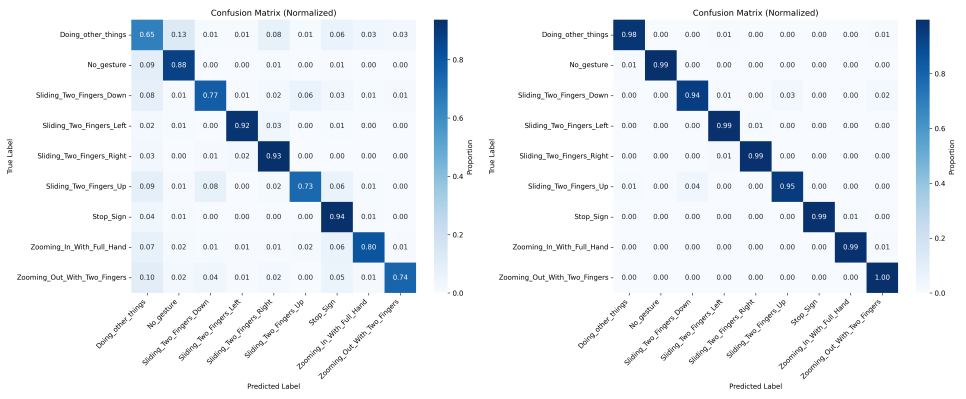


Figure 2: Row-normalized confusion matrix for the CNN-LSTM hybrid model (left) and pretrained R(2+1)D model (right).

Figure 2 shows the confusion matrix of the pretrained model; nearly all predictions lie on the diagonal.

Training dynamics also differ sharply across architectures. Figure 3 shows the loss and accuracy curves for the augmented 3D-CNN, which improves gradually over 30 epochs. Figure 4 shows that the pretrained R(2+1)D converges rapidly: high accuracy is reached within the first few epochs.

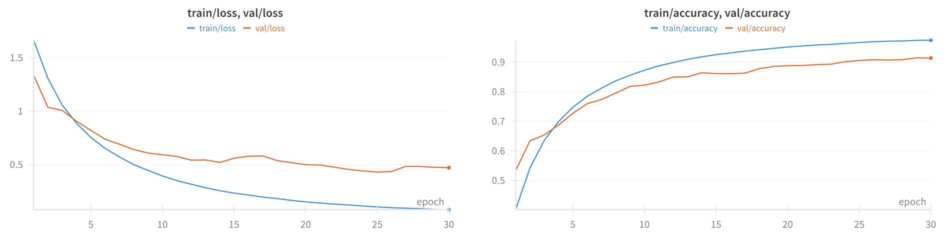


Figure 3: Training curves for the 3D-CNN with augmentation.

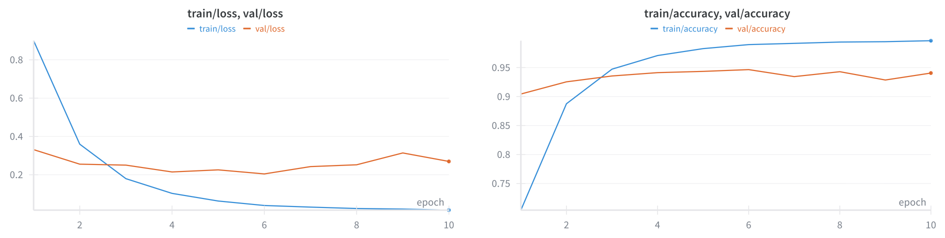


Figure 4: Training curves for the pretrained R(2+1)D model. Strong initialization yields rapid convergence.

5.2 Inference Efficiency

Inference performance varies dramatically across hardware backends (Table 2). On CUDA, the C3D models achieve very high throughput (≈2.34 ms per clip), benefiting from optimized cuDNN kernels for 3D convolution. However, the same architecture incurs substantial slowdowns on CPU (≈296 ms) and M3-MPS (≈198 ms), reflecting the limited optimization of large 3D kernels on non-CUDA platforms.

Model	**CUDA (ms)**	**MPS (ms)**	**CPU (ms)**	**Params (M)**
C3D (no aug, 20 ep.)	2.344	198.88	298.81	110.5
C3D (aug, 30 ep.)	2.344	197.43	295.57	110.5
CNN–LSTM (50 ep.)	**1.757**	**13.95**	**174.46**	12.49
R(2+1)D (scratch, 30 ep.)	5.303	216.98	348.32	31.30
R(2+1)D (pretrain, 10 ep.)	5.266	217.13	388.73	31.30

Table 2: Mean per-clip inference latency (100 runs) for each model across CUDA (H200), Apple MPS (M3 Pro), and CPU backends.

The CNN–LSTM hybrid is by far the most portable model. It achieves the fastest latency on both CPU (174.46 ms) and MPS (13.95 ms), and even slightly outperforms the C3D models on CUDA (1.76 ms). This stability across hardware is expected given its use of 2D convolutions and LSTM layers, which have efficient kernels on all backends. The relative penalty for moving from GPU to CPU is also significantly smaller than for the 3D-CNN and R(2+1)D architectures.

The R(2+1)D models show a different pattern. Although moderately fast on CUDA ($\approx$5.3 ms), they scale poorly on CPU (348–389 ms) and MPS ($\approx$217 ms), even slower than C3D. Factorized 2D+1D convolutions introduce additional kernel launches on CPU/MPS, producing noticeable overhead. The pretrained variant is slightly slower than the scratch model, likely due to deeper or more optimized internal layers.

Overall, these results demonstrate that accuracy and deployability do not align: the pretrained R(2+1)D is the most accurate model (97.85%), whereas the CNN–LSTM is the only architecture that remains fast across all hardware platforms and is therefore the strongest candidate for CPU-only or mobile deployment scenarios.

5.3 Ablations for the Hypotheses

H1: Temporal modeling improves accuracy. Our results partially support H1. While both temporal architectures trained from scratch (CNN–LSTM and R(2+1)D) fail to surpass the purely convolutional 3D-CNN baseline, the pretrained R(2+1)D model achieves the highest accuracy and macro F1 by a substantial margin. This indicates that temporal modeling *can* improve accuracy, but only when paired with strong spatiotemporal priors obtained through large-scale pretraining (e.g., Kinetics-400). Thus, H1 is confirmed only in the presence of pretraining, and not when models are trained from scratch.

H2: Reduced gesture space improves generalization. H2 is supported for the 3D convolutional architectures. All 3D-based models exceed 92% accuracy on our nine-class subset, and the pretrained R(2+1)D reaches 97.85%, which is comparable to the top-performing methods on the full 27-class Jester leaderboard despite the reduced label space. Confusion matrices show fewer systematic confusions, indicating that restricting the gesture set reduces ambiguity and improves generalization.

However, H2 does *not* hold for the CNN–LSTM hybrid. On the official Jester leaderboard, CNN–LSTM variants typically achieve 86–89% accuracy on the full 27-class task, whereas our nine-class CNN–LSTM reaches only 81.89%. The reduced label space does not improve its generalization, likely due to the model's smaller capacity and weaker temporal modeling.

Therefore, H2 is partially supported, i.e. it holds for high-capacity spatiotemporal models (C3D, R(2+1)D) but is refuted for the lower-capacity CNN–LSTM.

H3: CNN–LSTM offers the best accuracy and efficiency trade-off. H3 is also partially supported. The CNN–LSTM hybrid achieves the lowest inference latency across all hardware backends and shows the smallest GPU to CPU slowdown, confirming its computational efficiency and portability. However, its classification accuracy is markedly below all 3D-based models. Given its parameter count (12.5M against 31M for R(2+1)D and 110M for C3D), the model struggles to learn fine-grained temporal distinctions such as sliding directions and zoom gestures.

Thus, **H3 is confirmed in terms of efficiency, but refuted in terms of accuracy**. CNN–LSTM offers the best speed, but not the best accuracy–efficiency trade-off when accuracy is weighted heavily.

5.4 Failure Mode Analysis

Across non-pretrained models, the most common errors occur among the four sliding gestures, as shown in Fig. 2, which differ only by motion direction, and between the two zoom gestures, whose distinctions rely on subtle finger configurations. These failure modes reflect the fine-grained nature of the gesture set.

The pretrained R(2+1)D model greatly reduces these misclassifications—its confusion matrix is nearly diagonal—but small residual errors suggest that incorporating explicit motion cues (e.g., optical flow, hand keypoints, bounding boxes, etc.) or multi-view information could yield further gains.

The CNN–LSTM's weak performance on the nine-class subset is consistent with its low model capacity (12.5M parameters), which limits its ability to exploit the reduced label space, unlike the higher-capacity 3D convolutional architectures and both R(2+1)D variants.

6 Conclusion

In this work, we developed and evaluated a deep learning pipeline for dynamic hand gesture recognition tailored for media control. By comparing 3D-CNN, CNN-LSTM, and R(2+1)D architectures, we demonstrated that while simpler 3D convolutions provide a strong baseline, leveraging factorized spatiotemporal convolutions with pretraining yields superior results, achieving 97.85% accuracy.

We observed that some limitations are that our current approach assumes a static camera position and processed frames of 128×128, which may degrade in wild, varying-resolution environments. Furthermore, while the R(2+1)D model is highly accurate, its inference throughput ($\approx$ 190 clips/s) is lower than the CNN-LSTM ($\approx$ 569 clips/s), which indicates a trade-off when targeting extremely resource-constrained edge devices.

Future research could focus on: (1) Integrating multimodal data (e.g., depth or optical flow) to resolve remaining directional ambiguities; (2) Deploying the model in a real-time webcam interface to test latency in real-world scenarios; and (3) Exploring model quantization to deploy the heavy R(2+1)D architecture on mobile processors without significant accuracy loss.

References

[1] Quentin De Smedt, Hazem Wannous, and Jean-Philippe Vandeborre. Skeleton-Based Dynamic Hand Gesture Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1206–1214, Las Vegas, NV, USA, June 2016. IEEE.

[2] Manousos Linardakis, Iraklis Varlamis, and Georgios Th. Papadopoulos. Survey on Hand Gesture Recognition From Visual Input. *IEEE Access*, 13:135373–135406, 2025.

[3] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. MediaPipe: A Framework for Building Perception Pipelines, June 2019.

[4] Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The Jester Dataset: A Large-Scale Video Dataset of Human Gestures. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 2874–2882, October 2019.

[5] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree, and Jan Kautz. Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4207–4215, Las Vegas, NV, USA, June 2016. IEEE.

[6] Dinh-Son Tran, Ngoc-Huynh Ho, Hyung-Jeong Yang, Eu-Tteum Baek, Soo-Hyung Kim, and Gueesang Lee. Real-Time Hand Gesture Spotting and Recognition Using RGB-D Camera and 3D Convolutional Neural Network. *Applied Sciences*, 10(2):722, January 2020.

[7] 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.

[8] Muneeb Ur Rehman, Fawad Ahmed, M. Khan, Usman Tariq, Faisal Alfouzan, Nouf Alzahrani, and Jawad Ahmad. Dynamic Hand Gesture Recognition Using 3D-CNN and LSTM Networks. *Computers, Materials & Continua*, 70:4675–4690, January 2022.

[9] Yifan Zhang, Congqi Cao, Jian Cheng, and Hanqing Lu. EgoGesture: A New Dataset and Benchmark for Egocentric Hand Gesture Recognition. *IEEE Transactions on Multimedia*, 20(5):1038–1050, May 2018.

7 Team Contribution

Member	**Technical Contributions (Implementation, Experiments, Analysis, Writing)**
Ishmanbir Singh	Contributed to research of related literature and prior work. Implemented the majority of the codebase, including the full data preprocessing pipeline, dataset filtering scripts, and integration of all three model architectures (C3D, CLSTM, R(2+1)D). Set up and executed large-scale distributed training runs on multi-GPU H200 nodes. Handled model checkpointing, logging, and infrastructure. Contributed to debugging, pipeline validation, and reviewing experimental results. Reviewed and edited the final report.
Richie Jonathan	Coordinated project planning, milestones, and experimental design. Verified the correctness of the training and evaluation pipeline, ran runs on the 3D-CNN models for the midterm report, and conducted evaluation and analysis of results (metrics, inference benchmarking, confusion matrices). Wrote the technical sections including the Method/Approach, Data documentation, Evaluation protocol, Experiments & Results, Ablations, and Failure Analysis.
Magnus Voss	Researched related literature and prior work on gesture recognition, temporal modeling, and the Jester dataset. Wrote the Abstract, Introduction, Background, Motivation and also Conclusion. Supported and review writing of the technical sections too. Reviewed and edited the final report.