Deep learning project 2025

VIDEO CLASSIFICATION

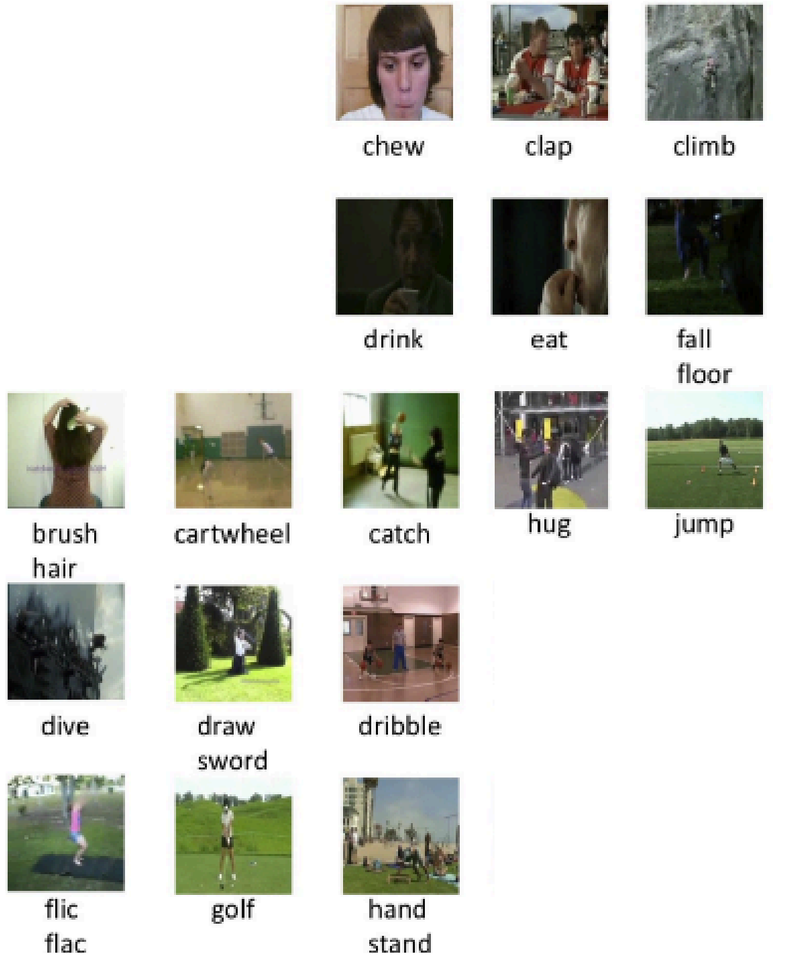
Aurora Castelnovo, Nicole Gemelli

INDEX OF CONTENTS:

ABOUT THE DATASET

The **dataset** in brief:

- designed for human activity recognition and behavior modeling.
- It contains multimodal data (e.g., video, audio, and sensor signals) across 51 diverse activity classes.
- Activities include daily tasks, social interactions, and complex motions, captured in realistic environments, for example: walk,eat,drink,...



OUR GOAL

We aim to classify activities into one of the 51 predefined categories

TO REACH IT:

1. We **standardized** all videos to the same format (frame size, number, and rate) for consistent processing.
2. We **trained** a neural network to classify actions based on video content.
3. We then **evaluated** and **selected** the best-performing model on validation data.
4. We chose a **CNN-based architecture**, as Convolutional Neural Networks are highly effective for image and video tasks due to their ability to capture spatial patterns.

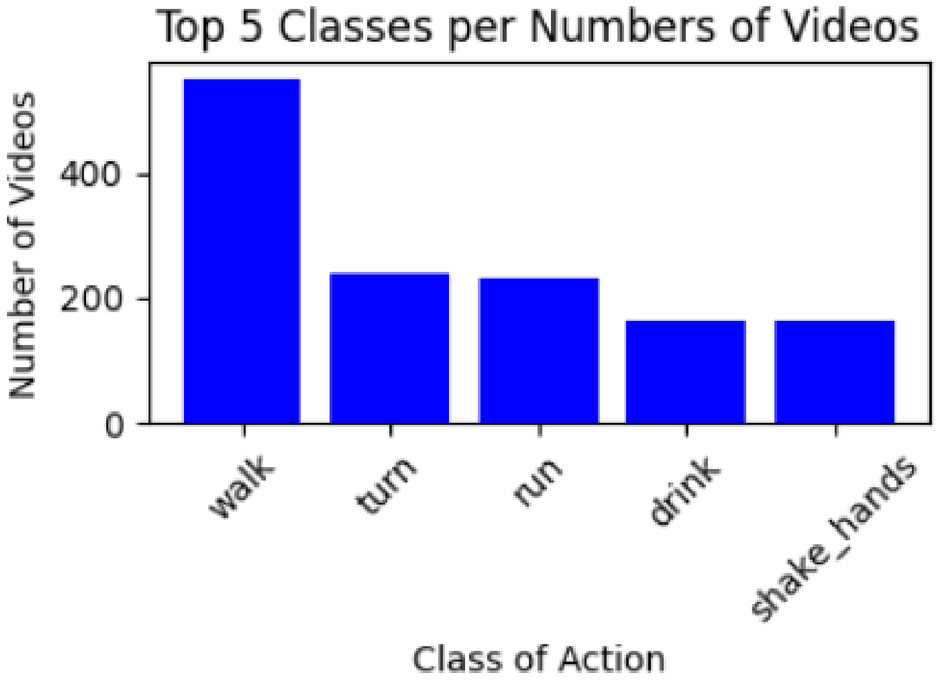
SOME EXPLORATORY ANALYSES

Class Distribution in HMDB51

The dataset includes **51** action classes with **imbalanced** video sample distribution.

- The class "**walk**" is overrepresented with 548 videos, over 2 times more than the second class, "**turn**" (240).
- The majority of other classes has about **140–160** videos, with the **smallest** class at **101** videos.

This **imbalance** may bias model performance, favoring frequent classes.



OTHERS EXPLORATORY ANALYSES

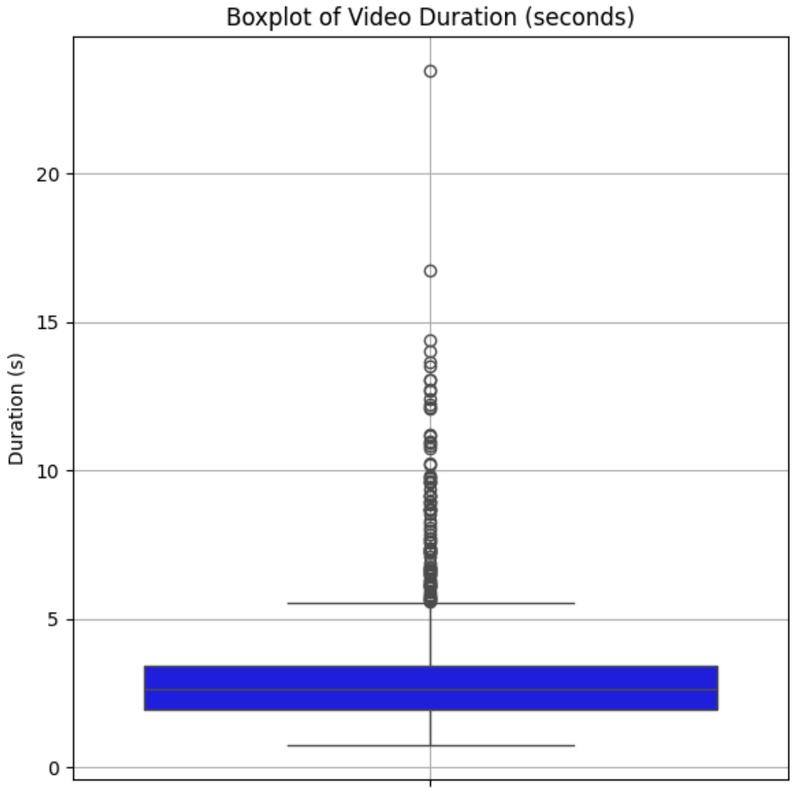
Variability in video length and resolution

Duration (in seconds):

- Mirrors the frame count distribution due to constant 30 FPS.
- Left-skewed: Median closer to the upper quartile.
- Shorter lower whisker, longer upper whisker.
- Outliers indicate unusually long or short videos.

Resolution / Width (in pixels):

- Varies from 176 to 592 pixels, unlike constant height.
- Boxplot shows spread due to varying video formats.
- Median and box shape highlight common resolutions.
- Variability reflects original video source, not duration.



VISUAL FEATURE ANALYSIS

Limitations of Static Visual Features

Limitations:

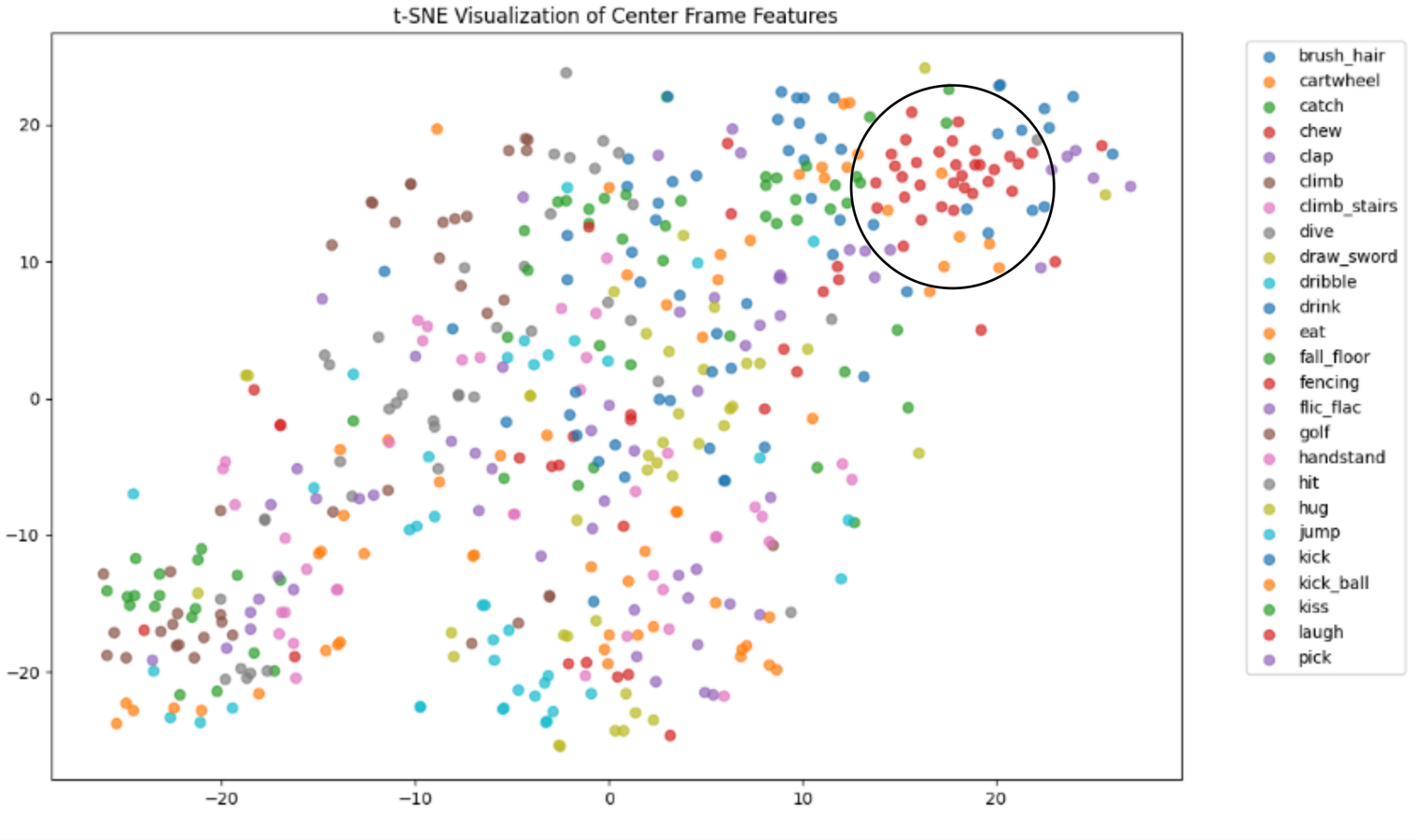
- High intra-class variability observed in actions like dive and kick_ball, where samples are dispersed in the feature space.
- Overlap among visually similar actions — e.g., hug, kiss, hit — makes classification harder when using only static frames.

Conclusion:

This suggests that static frame features are insufficient to fully capture action semantics. Temporal models such as 3D CNNs or LSTMs are necessary to leverage motion and context across time.

VISUAL FEATURE ANALYSIS

t-SNE Visualization of Center Frame Features (ResNet18)



- **Partial class** separability is observed in the 2D feature space.
- Actions like kick, dribble, and chew form tight clusters, indicating visual similarity.
- Some actions such as brush_hair and cartwheel are well separated, suggesting distinctive visual features.

FIRST ATTEMPT:

PRETRAINED MODELS ON THE WHOLE DATASET

ARCHITECTURE AND PROCESSING

1. Frame Extraction

- Extract 16 evenly spaced frames per video
- Done via cv2.VideoCapture

2. Feature Extraction

- Model: Pre-trained GoogLeNet
- Removed final classification layer → used as feature extractor
- Output: 1024-dimensional vector per frame

3. Sequence Structuring

- Each video: (16 frames × 1024 features)
- Invalid (None) sequences are removed
- Final valid dataset: 7091 videos

4. Label Encoding & Splitting

- Labels encoded using LabelEncoder
- Train/validation split: 90% / 10% (stratified)

MODELING APPROACHES AND TRAINING (1)

CNN + LSTM (abandoned)

- **Custom** model CNN_LSTM_VideoClassifier used for sequence learning
- Input: full (16 × 1024) feature matrix
- Training interrupted at epoch 5 due to GPU memory crash
- CUDA debugging (CUDA_LAUNCH_BLOCKING=1) and exception handling include

→ our hardware environment proved insufficient → **GPU** memory was not enough to load and train deep pretrained architectures on full-resolution HMDB51 videos.
Even after adjusting batch size and frame sampling, we encountered runtime memory crashes.

MODELING APPROACHES AND TRAINING (2)

Fully Connected Model (final):

- Input: temporal average of 16 frame features → (1 × 1024) vector
- Architecture: *Linear(1024 → 256) → ReLU → Dropout(0.3) → Linear(256 → 51)*

Training Details:

- Optimizer: Adam (lr=1e-4)
- Loss: CrossEntropyLoss
- Epochs: 10
- Batch size: 8
- Device: GPU (if available), fallback to CPU

→ 7091 total samples: ~6381 training, ~710 validation

EVALUATION AND RESULTS

Validation metrics

- Accuracy: 55.63%
- Validation Loss: 1.6278
- Samples Evaluated: 710

Classification performance

- Macro avg F1-score: 0.5522
- Weighted avg F1-score: 0.5437

Observations from confusion matrix

- Good performance: drink, shoot_bow, wave
- Confusion in similar actions: chew, laugh, smile
- Balanced detection for most classes, some overfitting or underperformance on less frequent actions

OURS ATTEMPT (NON-PRETRAINED): LIGHTWEIGHT CUSTOM MODEL ON REDUCED DATA

Objective:

Test a custom, lightweight architecture built entirely from scratch

Approach:

- Selected a reduced subset of HMDB51 (fewer classes and videos)
- Built a minimal 2D CNN + LSTM pipeline to assess baseline performance
- Trained on center-extracted frames (16 per video)

Model Architecture:

- 3 Convolutional Layers:
  Channels: 32 → 64 → 128
  Each followed by ReLU and MaxPooling
- Spatial Reduction:
  AdaptiveAvgPool2d to (1 × 1)
- Temporal Modeling:
  1-layer LSTM processing per-frame 128-dim features

Final validation accuracy: ~7%

Performance limited by:

- Very small dataset
- Shallow architecture
- Insufficient training time

Now we need to:

- Evaluate more robust temporal-spatial models (e.g., 3D CNNs, TimeSformer, VideoMAE)
- Consider cloud-based training or extreme frame reduction
- Investigate architectural improvements to the baseline CNN

This led to the development of a series of progressively stronger CNN-based architectures

VARIOUS MODELS TRAINED

1 Baseline CNN + LSTM

Model built entirely from scratch, using a minimal CNN followed by an LSTM for temporal modeling

- 3 convolutional layers (ReLU activations)
- MaxPooling after each conv layer
- LSTM layer (hidden size 256)
- Fully connected output layer (51 classes)

2 CNN with BatchNorm + LeakyReLU

Introducing Batch Normalization and LeakyReLU to improve convergence and avoid vanishing gradient problems

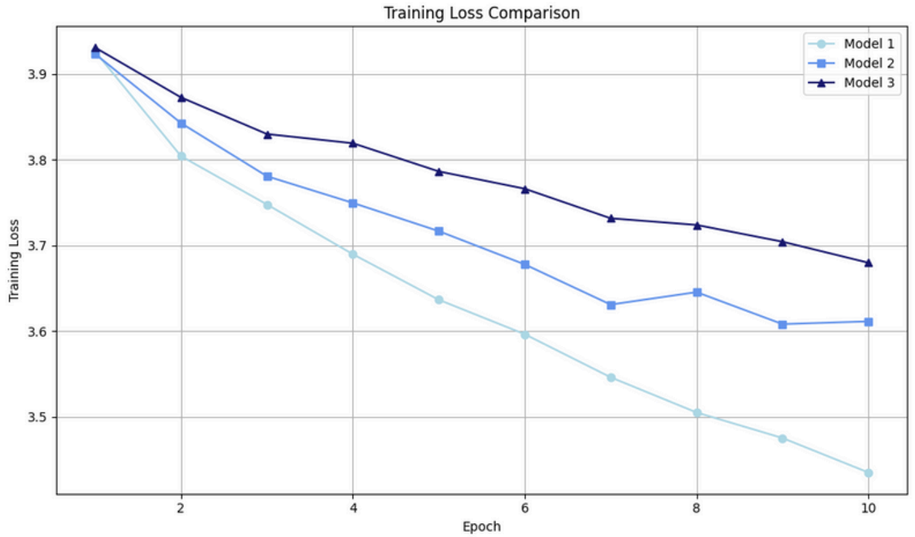
- Same structure as Model 1
- + Batch Normalization after each conv
- LeakyReLU instead of ReLU
- Balanced class weights during training

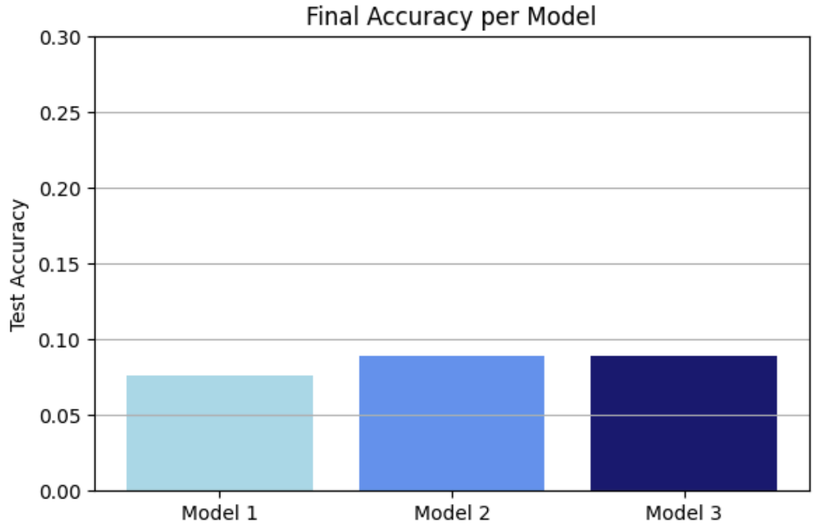
3 CNN with Dropout + Deeper CNN blocks

Adds Dropout and additional convolutional layers to improve generalization and reduce overfitting. It’s the most regularized version

- Increased conv layers and filters
- Introduced Dropout (0.3)
- LSTM and FC layers unchanged

LOSS AND ACCURAY COMPARISON





Model 1 (Baseline) shows the fastest loss reduction but lowest accuracy, likely overfitting. Models 2 and 3 converge more stably, with Model 3 achieving the best (though still limited) test accuracy, thanks to stronger regularization.

Low performance across all models may also stem from class imbalance in the dataset.

HOW TO HANDLE CLASS IMBALANCE?

WEIGHTED CLASSES

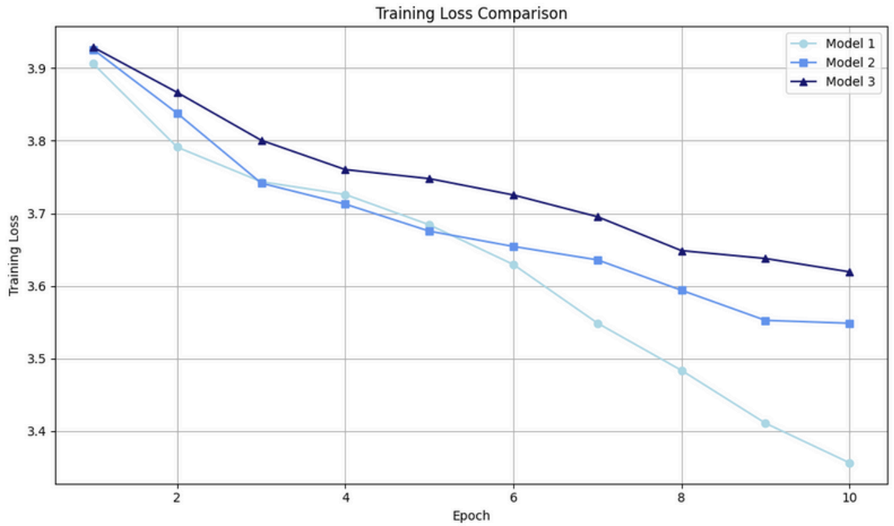
Highly imbalanced distribution of video across classes → e.g. walk is overrepresented

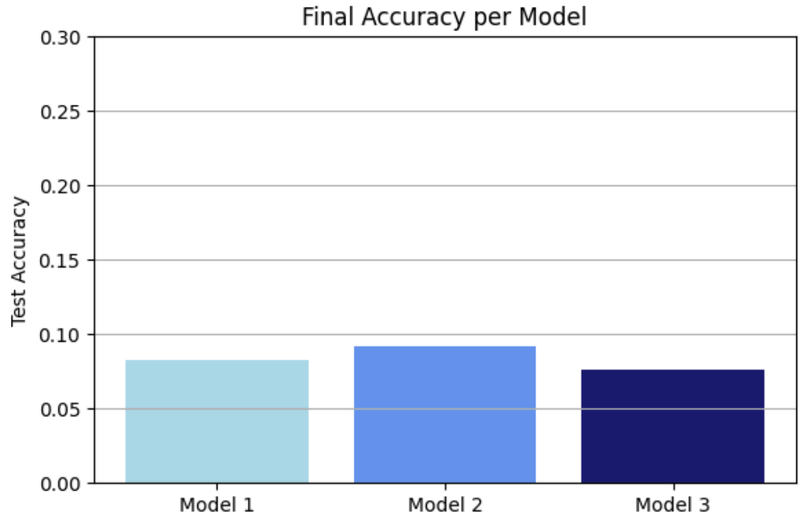
Solution:

- Use Weighted Cross-Entropy Loss, which assigns greater importance to underrepresented classes.
- Class weights are automatically calculated using

  sklearn.utils.class_weight.compute_class_weight
- The model combines a CNN to extract spatial features from video frames and an LSTM to capture temporal patterns across frame sequences.
- Training is performed using the Adam optimizer (learning rate: 1e-4), with parameters updated via backpropagation on each batch.
- Evaluation: performance tracked using accuracy and validation loss

LOSS AND ACCURAY COMPARISON (CLASS WEIGHTS)





- Model 1 shows a sharp increase in accuracy, but this is likely unreliable → Its simplicity and lack of regularization may have caused it to overfit to minority classes with high weights.
- Model 2 shows a clear and stable improvement → Thanks to BatchNorm and LeakyReLU, it handles the reweighted loss more effectively and generalizes better.
- Model 3 experiences a drop in performance → Its deeper and already regularized architecture may have become too constrained when combined with class weights.

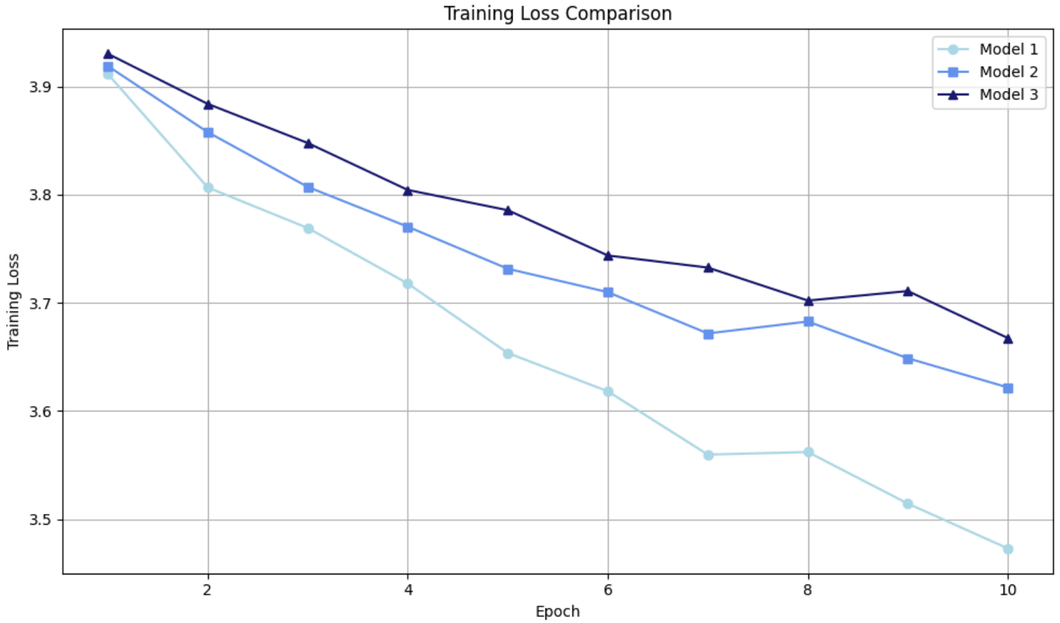
HOW TO PREVENT OVERFITTING?

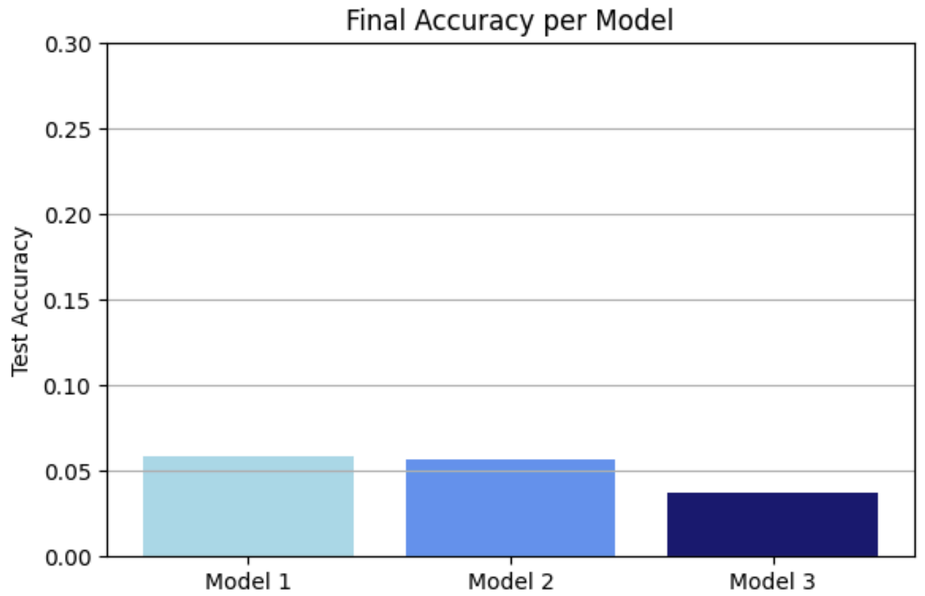
WHY APPLY AUGMENTATION AFTER USING CLASS WEIGHTS

After applying class weights, we address class imbalance, but that alone doesn't prevent overfitting — especially when the dataset is small or the selected model is complex.

Augmentation:

- artificially increases dataset diversity.
- helps the model generalize better by seeing slightly different versions of the same class (e.g., flips, crops, brightness changes).
- especially helpful after class weighting, which can push the model to focus more on rare classes → makes sure it doesn't overfit to specific samples from those classes.





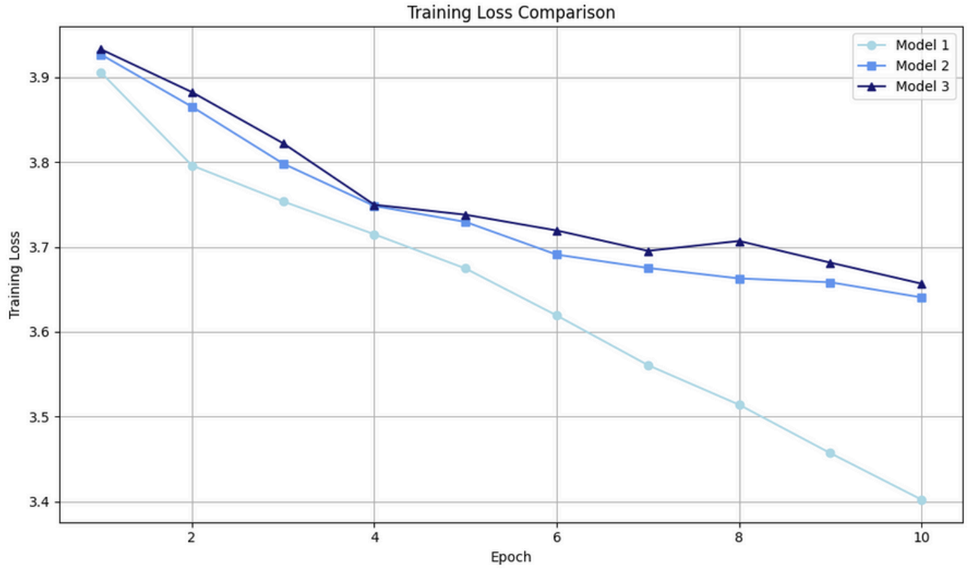
- Data augmentation did not improve the training process
- Notable drop in accuracy across all models
- The overall performance was lower than in previous models trained without augmentation
- Loss curves showed higher variability, particularly for Model 1, which had the most unstable training

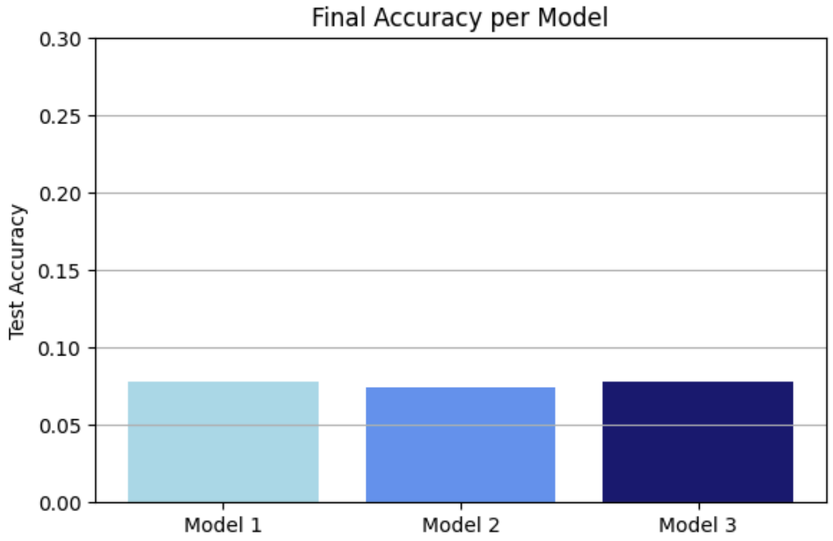
Given these results, we decided not to proceed with data augmentation in the final configuration, as it introduced noise without improving generalization, likely due to the limited dataset size

WHY APPLY DROPOUT AFTER USING CLASS WEIGHTS

Dropout:

- reduces overfitting in deeper or more complex models.
- randomly deactivates neurons during training, encouraging the model to not rely too heavily on specific paths in the network.
- After applying class weights, models can overemphasize minority classes, especially if they have limited examples. Dropout adds regularization to help prevent that.





Models with Dropout performed similarly to other regularization strategies and clearly outperformed those using data augmentation.

- model 2, though still accurate, became the weakest, while models 1 and 3 perform similarly → the accuracy differences are smaller compared to earlier configurations.
- models 2 and 3 maintain stable loss trajectories, while model 1 shows a sharp loss drop, suggesting a more confident fit during training.

→ Dropout helped stabilize training but didn’t lead to major performance gains

OUR FINAL MODEL CHOICE

Layer	Shape	# Parameters
cnn.0.weight	(32, 3, 3, 3)	864
cnn.0.bias	(32,)	32
cnn.1.weight	(32,)	32
cnn.1.bias	(32,)	32
cnn.4.weight	(64, 32, 3, 3)	18432
cnn.4.bias	(64,)	64
cnn.5.weight	(64,)	64
cnn.5.bias	(64,)	64
cnn.8.weight	(128, 64, 3, 3)	73728
cnn.8.bias	(128,)	128
cnn.9.weight	(128,)	128
cnn.9.bias	(128,)	128
lstm.weight_ih_l0	(1024, 128)	131072
lstm.weight_hh_l0	(1024, 256)	262144
lstm.bias_ih_l0	(1024,)	1024
lstm.bias_hh_l0	(1024,)	1024
fc.weight	(51, 256)	13056
fc.bias	(51,)	51
Total		502067

WE have chosen as best model the second one (CNN with BatchNorm + LeakyReLU), with balnced weights

WHY CNN AND NOT RNN?

While Recurrent Neural Networks (RNNs) have traditionally been used for video classification due to their ability to model temporal sequences, we deliberately opted for a CNN-based approach in this project. The decision was guided by both practical constraints and recent research trends.

Motivations Behind the Choice:

- We experimented with a hybrid model, using 2D CNNs for spatial feature extraction and a lightweight LSTM for temporal modeling.
- CNNs are generally more efficient to train and easier to optimize, especially under limited computational resources.
- Modern 3D CNN architectures inherently capture both spatial and temporal dynamics, reducing the need for explicit sequence modeling via RNNs.
- RNNs can be computationally expensive and sensitive to class imbalance and limited data, which were key challenges in our setup.

PROBLEMS AND FURTHER IMPROVEMNETS

Problems Encountered:

- GPU limitations restricted batch size and training speed → rorced us to use a reduced dataset (e.g., ~30 videos/class), lowering data diversity
- Overfitting risk due to limited training samples
- Long training times with deeper architectures (e.g., CNN + LSTM)
- Overall low accuracy constrained by these factors

Potential Improvements:

- Leverage more powerful GPUs or cloud computing for faster and larger-scale training
- Adopt pretrained video models (e.g., TimeSformer, VideoMAE) for better feature extraction
- Apply data augmentation at the video level to increase variability. Experiment with smarter frame sampling strategies to retain key temporal info
- Fine-tune on the full HMDB-51 dataset for improved generalization and performance

THANKS FOR THE ATTENTION