Comparative Study of Classical vs Deep Learning Approaches for Video Classification

Technical Report
Video Analysis – Assignment I: Video Classification
Ashish Vashistha
2024AA05043

1. Introduction

This Assignment develops a comparative video classification system that implements both

- A classical computer vision + classical machine learning pipeline and
- A modern deep learning pipeline for categorizing videos into predefined action classes.

The objective is not only to achieve high classification performance, but also to systematically compare the two paradigms across

  - Accuracy-based metrics,
  - Computational cost,
  - Representation/feature analysis, and
  - Deployment suitability.

Key goals of the study:

- Build a reproducible end-to-end workflow: video loading → sampling → preprocessing → feature/representation extraction → model training → evaluation → comparative analysis.

- Compare multiple models in each paradigm:
  - Classical ML: SVM, Random Forest, k-NN.
  - Deep learning: 2D CNN + temporal pooling (EfficientNet-B0 variants) and 3D CNN (R3D-18).

- Analyze trade-offs between:
  - accuracy vs compute/memory,
  - interpretability vs performance,
  - edge vs cloud feasibility

Dataset

- A subset of the UCF-101 action recognition dataset was used with three action classes: Basketball (Jumping), CricketBowling(Running), and WalkingWithDog(Walking).
- Videos are organized by class folders and split files (train/val/test) were used for reproducible experiments.
- The deep-learning experiments used a larger split (Train/Val/Test = 277/59/60 videos),
- While the classical pipeline was prototyped on a smaller split (Train/Val/Test = 54/12/9 videos) due to very high-dimensional handcrafted features prior to PCA reduction.

2. Background and Related Work

Video classification has historically evolved from hand-crafted features (color/texture/shape/motion descriptors) + classical ML classifiers to representation learning using CNNs and spatiotemporal deep networks. In classical pipelines, performance depends heavily on feature engineering, aggregation strategy, and robustness to intra-class variability. In deep learning pipelines, pretrained backbones and fine-tuning enable strong performance even with limited labeled data by leveraging transfer learning from large-scale datasets.

This report reflects that evolution by implementing:

- A feature-engineering-heavy classical workflow with explicit motion and temporal statistics.
- A learned representation workflow using pretrained EfficientNet-B0 (2D) and pretrained R3D-18 (3D) with fine-tuning.

3. Methodology

A. Classical Video Classification Pipeline

(i) Data Loading, Sampling, and Preprocessing

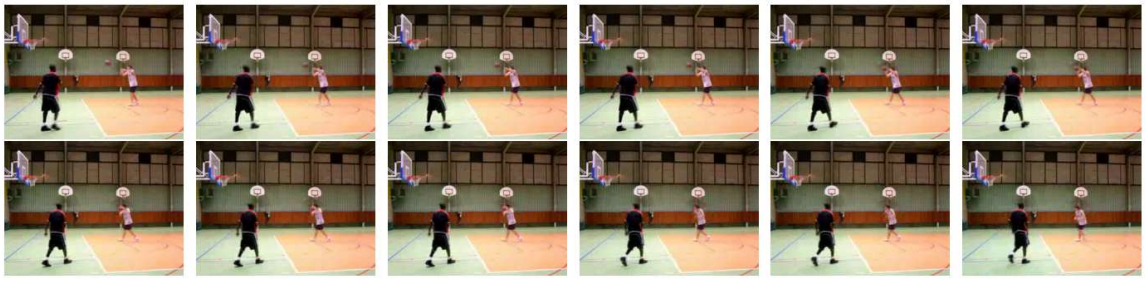
Videos are loaded using OpenCV. The pipeline supports multiple sampling strategies:

- Uniform sampling (e.g., ~1 fps via stride based on video fps),

Uniform sampling (1 fps) - sampled 1fps frames

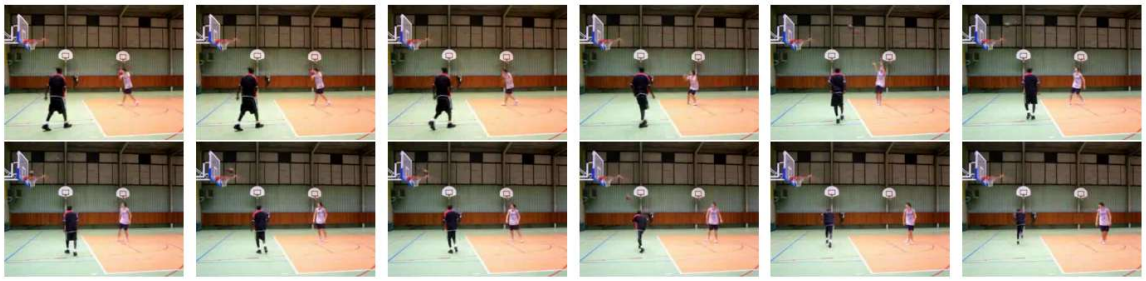
- Dense sampling (every frame, capped for speed),

Dense sampling (every frame) - first 12 frames



- Random sampling (K random frames for augmentation)

Random sampling - 12 frames



Preprocessing includes:

- Resizing frames to a classical-friendly resolution (320×240)

Example resized frame (320x240)

- Quality enhancement using denoising (fast non-local means) and mild sharpening (unsharp mask),

Quality Enhancement: Original vs Enhanced

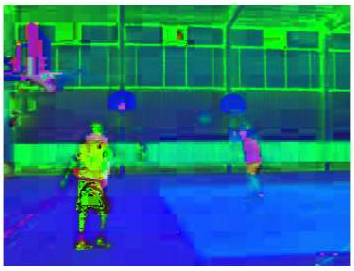
- Normalization to float range [0,1]

Normalized visualization (0..1)

- Optional color-space conversions (RGB/HSV/GRAYSCALE) to support different feature families.

RGB

HSV



Grayscale

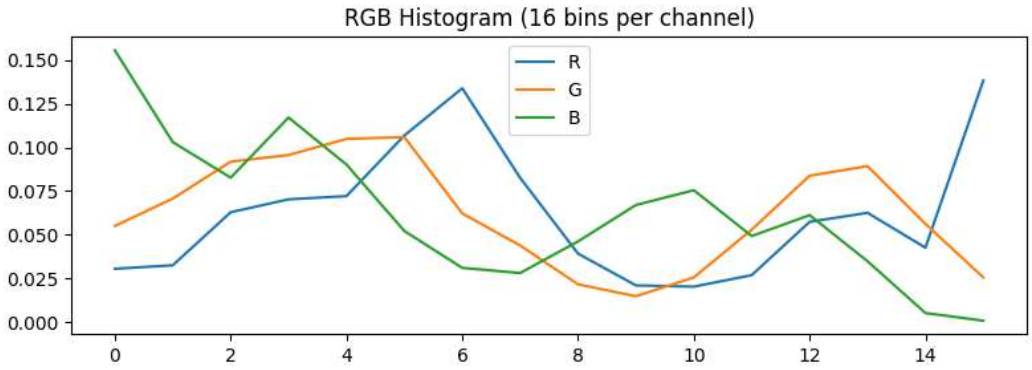
Quality diagnostics are computed (blur score via Laplacian variance, noise estimate, blockiness proxy), and enhancements are applied before feature extraction.

(ii) Feature Extraction (Hand-Crafted)

The classical pipeline extracts multiple feature families:

(1) Color features

- RGB histogram (16 bins/channel),



- Color moments (mean, variance, skewness),

```
Color moments: [1.3208119e+02 5.6076382e+03 3.0089197e-01 1.1376966e+02 5.4268594e+03
 3.2916078e-01 8.9726303e+01 4.5973442e+03 3.9678657e-01]
```

- Average RGB color,

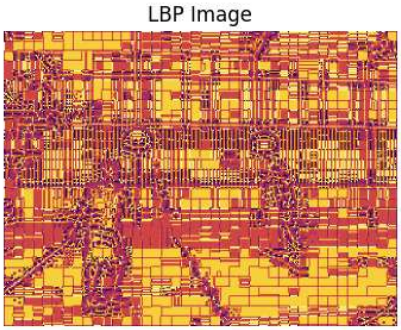
```
[132.08119 113.76966  89.7263 ]
```

(2) Texture features

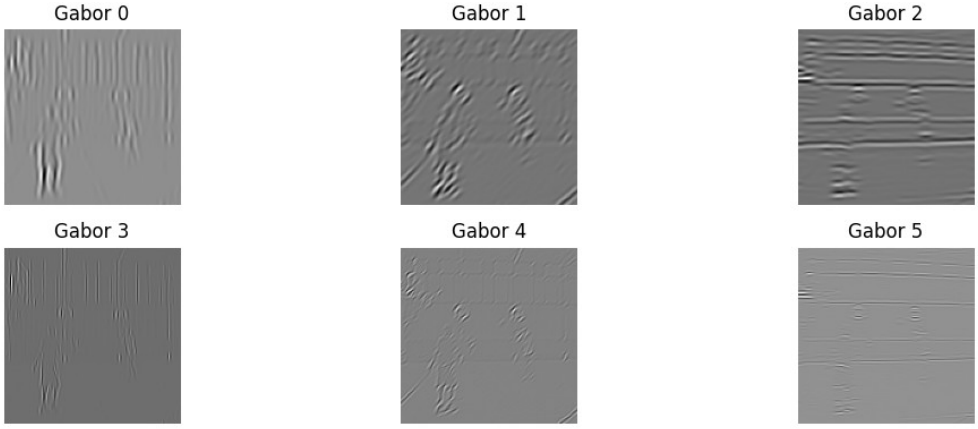
- GLCM statistics (contrast, dissimilarity, homogeneity, energy, correlation, ASM),

Gray (GLCM input)

- LBP histogram (uniform patterns),

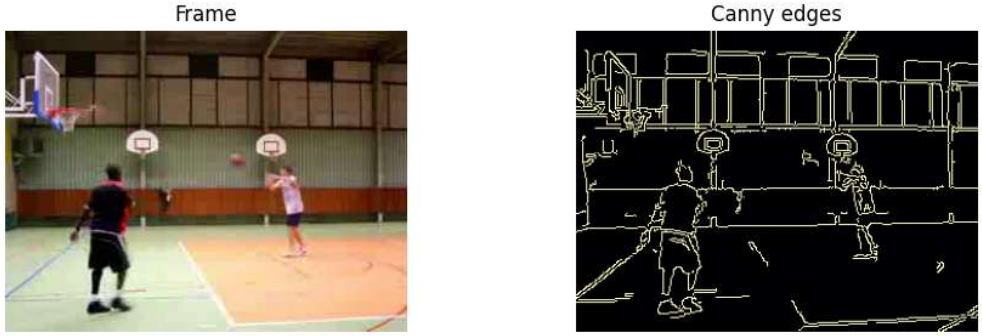


- Gabor filter statistics across frequencies and angles.



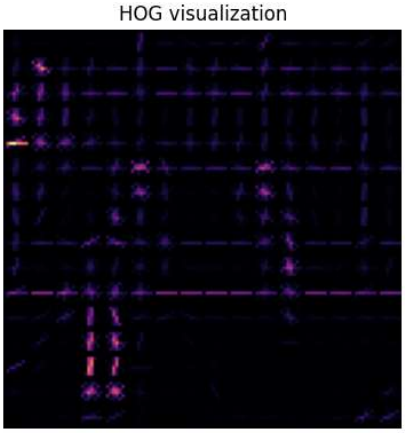
(3) Shape features

- Edge histogram using Canny,



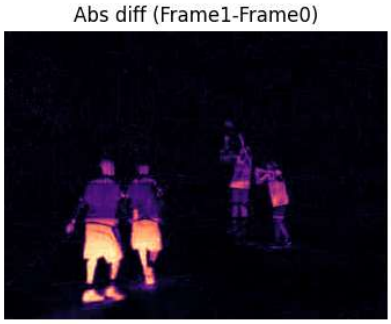
- Contour-based statistics (count, mean/variance of areas and perimeters, etc.),

- HOG descriptor (very high-dimensional).

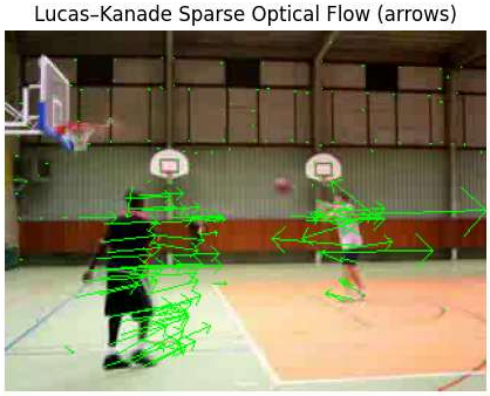


(4) Motion features

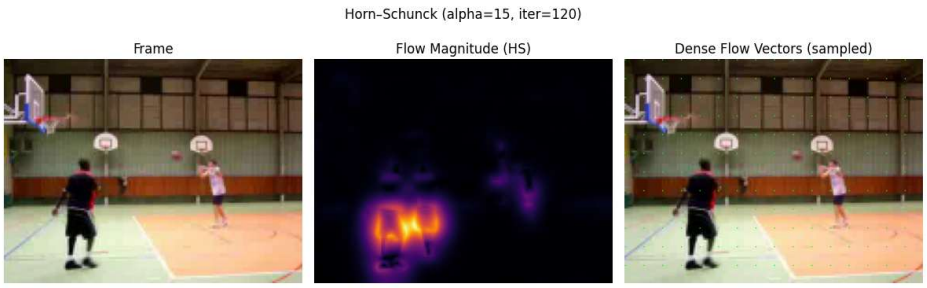
- Frame differencing (absolute diff) with statistical summary + histogram,



- Lucas–Kanade sparse optical flow statistics,

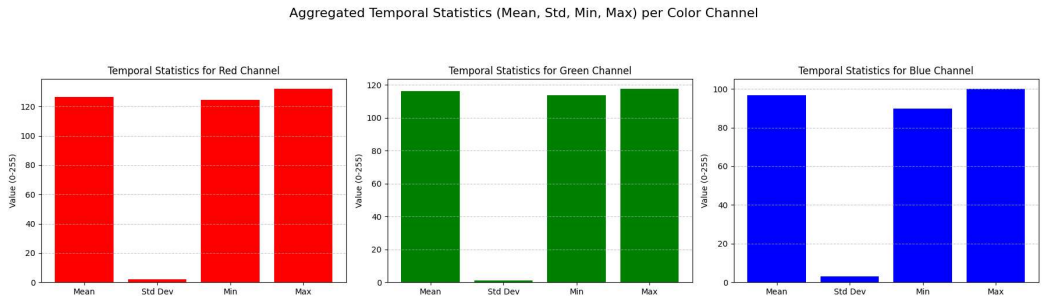


- Horn–Schunck dense optical flow (magnitude/direction histograms + dominant direction).

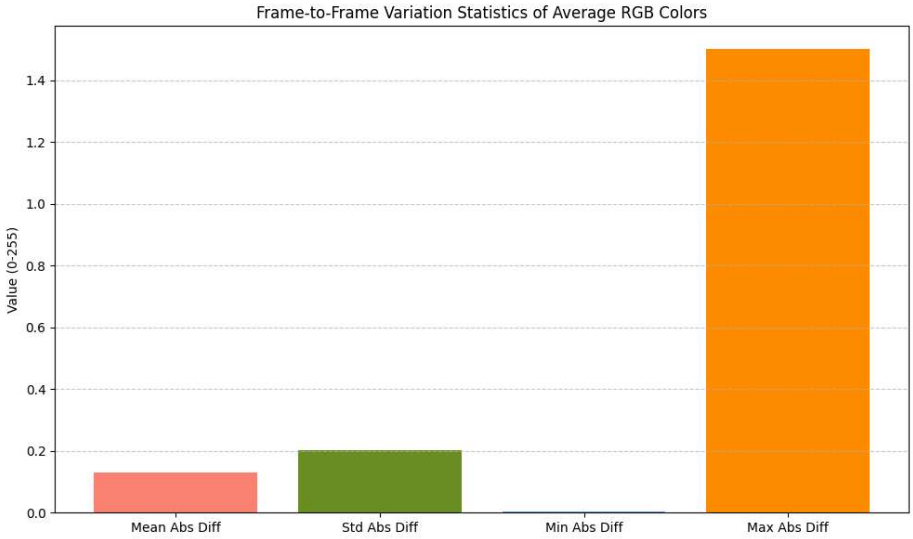


(5) Temporal aggregation Frame-level features are aggregated into video-level descriptors using:

- mean/std/min/max across time (“temporal stats”),



- frame-to-frame variation statistics.

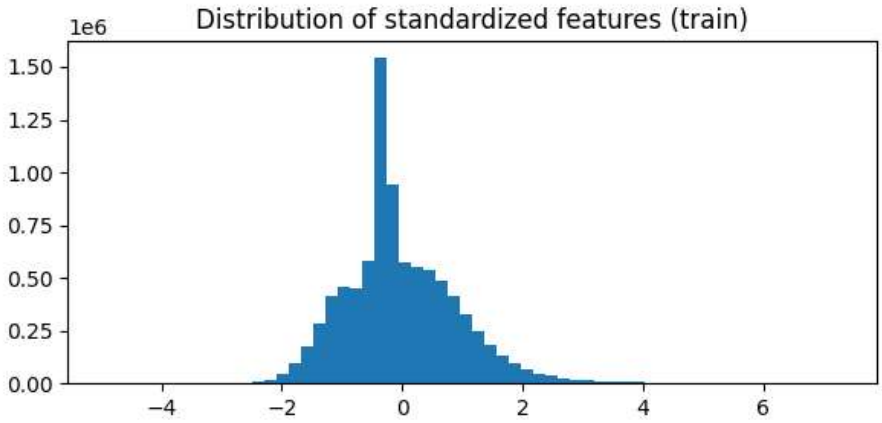


- This produces a **very high-dimensional representation**: the resulting feature matrix for training is reported as **(54, 163828)** (i.e., 163,828 features per video).

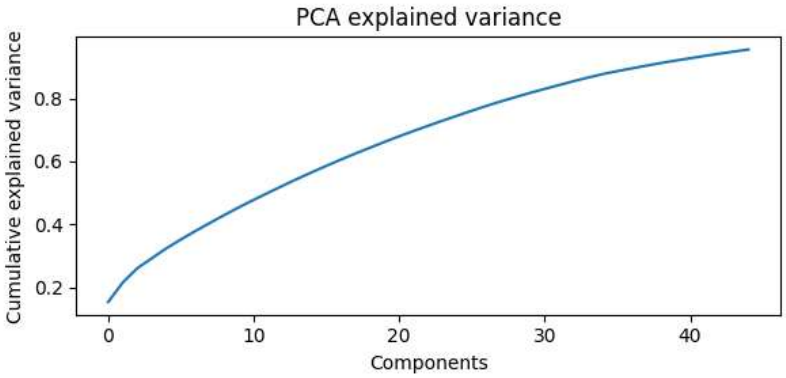
(iii) Classical Model Training

To stabilize and compress the high-dimensional features:

- Missing values are handled via **median imputation**,
- Features are standardized with **StandardScaler**,



- Dimensionality reduction is applied using **PCA retaining 95% variance**, reducing features from **163,828 → 45 components**.



Three classical ML algorithms are trained and tuned:

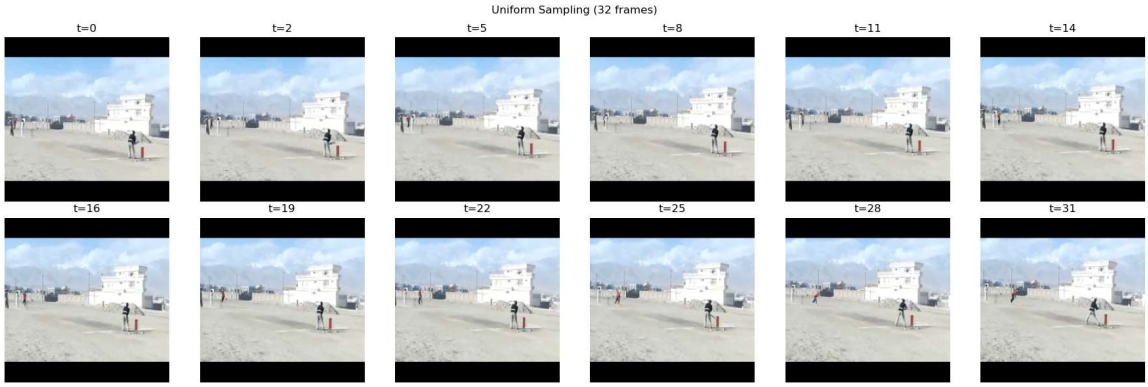
1. **SVM** (linear + RBF; grid search over C, kernel, gamma),
2. **Random Forest** (n_estimators, max_depth, min_samples_split via grid search),
3. **k-NN** (k, distance metric, weighting via grid search)

B) Deep Learning Video Classification Pipeline

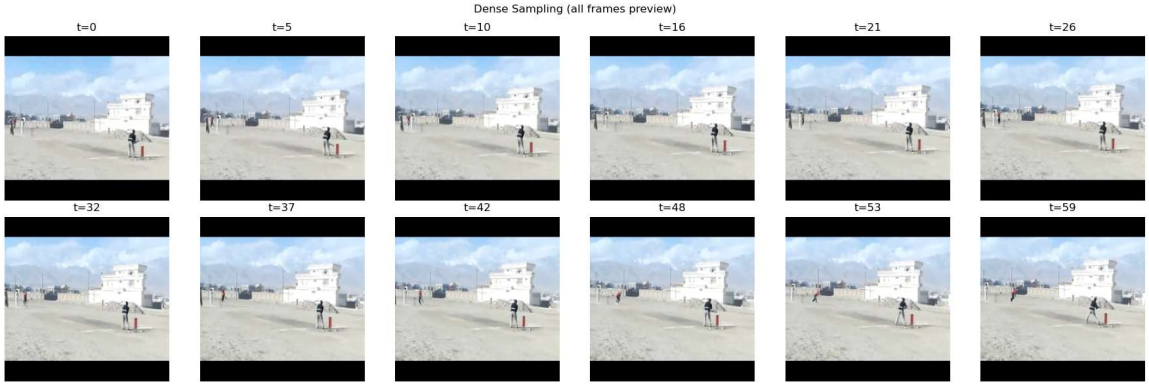
(i) Dataset Handling, Sampling, and Normalization

- The deep learning pipeline follows the same dataset organization (class folders + split txt files). It reads split files for train/val/test and infers the number of classes (**3 classes**) with splits **277 / 59 / 60**.
- Frames are loaded via OpenCV, resized to **224×224** and sampled using:

- Uniform sampling (fixed number of frames, e.g., **32**),

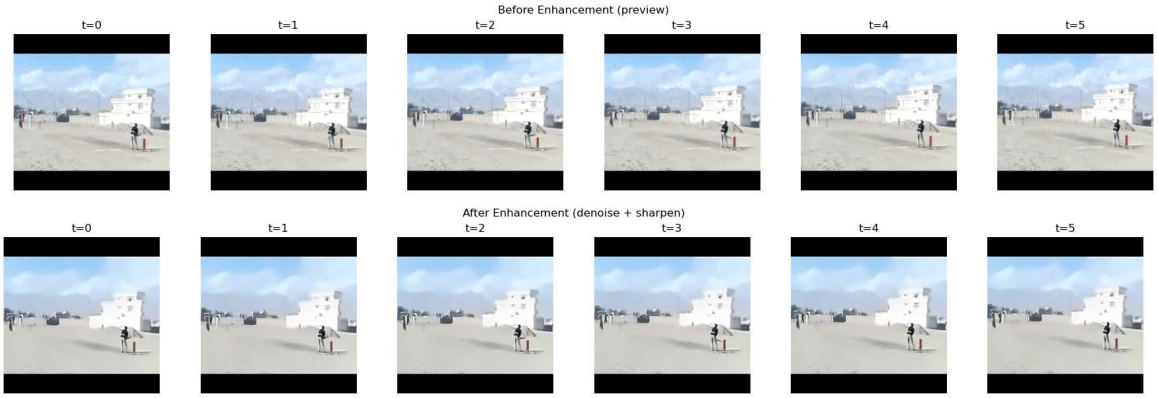


- Dense sampling (all frames),



- Random sampling (data augmentation effect).

- Preprocessing and enhancement include denoising + sharpening (optional), and tensor normalization using **ImageNet mean/std**; dataset-specific mean/std computation is also demonstrated.



(ii) Model 1 — 2D CNN + Temporal Pooling

A pretrained **EfficientNet-B0** backbone is used as a frame feature extractor. The model processes T- frames per video, extracts per-frame features, and applies temporal pooling to get a video-level vector. Three pooling strategies are implemented:

- **Average pooling** across time,
- **Max pooling** across time,
- **Attention pooling**, where a lightweight attention module learns frame weights.

- A fine-tuning strategy freezes most layers and unfreezes the last few blocks (fine_tune_last_n=2), with dropout and LayerNorm before the final classifier.
- Training uses:

- Adam optimizer with weight decay,
- StepLR learning rate scheduling,
- early stopping based on validation accuracy

(iii) Model 2 — 3D CNN Clip Model (R3D-18)

A pretrained R3D-18 (3D ResNet-18) from torchvision is used to classify short clips directly as (C,T,H,W)volumes (T=16 frames). The final FC layer is replaced with a LayerNorm + Dropout + Linear classifier head. Fine-tuning unfreezes layer4 and the classifier head.

4. Experimental Setup

Data splits were defined using text files to ensure reproducibility.

- Classical experiments used a smaller split during prototyping due to feature extraction cost.
- Deep learning experiments used the full split and were run on a CUDA-enabled GPU.

Dataset Splits

We followed the splits as per the original splits defined in UCF-101 Dataset

Pipeline	Train	Validation	Test
Classical (prototype split)	54	12	9
Deep learning (full split)	277	59	60

Hyperparameter Tuning Approach

Classical (Part A):

- **Preprocessing + Dimensionality Reduction tuned for feasibility:** Because the handcrafted feature vector is extremely high-dimensional (**163,828 features/video**), the pipeline applies **median imputation + StandardScaler**, followed by **PCA retaining 95% variance**, reducing the feature space to **45 components** before model fitting.
- **Grid Search with Cross-Validation (CV=3):** Hyperparameters are tuned using GridSearchCV with **3-fold cross-validation** for all classical models, ensuring consistent and fair selection.
    - **SVM tuning:** Grid over C ∈ {0.1, 1, 10}, kernel ∈ {linear, rbf}, and gamma ∈ {scale, 0.01, 0.001} (gamma applies to RBF). Best found: **RBF, C=1, gamma=scale**.
    - **Random Forest tuning:** Grid over n_estimators ∈ {200, 400}, max_depth ∈ {None, 20, 40}, min_samples_split ∈ {2, 5}. Best found: **200 trees, max_depth=None, min_samples_split=2**.
    - **k-NN tuning:** Grid over n_neighbors ∈ {3,5,7,9,11,15}, weights ∈ {uniform, distance}, and metric ∈ {euclidean, manhattan}. Best found: **k=3, weights=distance, metric=euclidean**.

Deep Learning (Part B):

- **Model variants as tuning strategy:** Instead of exhaustive hyperparameter search, the notebook compares multiple architecture/pooling variants under identical data splits:
  - EfficientNet-B0 with **avg**, **max**, and **attention** temporal pooling
  - 3D CNN baseline: **R3D-18** clip model
- **Training stabilizers (training hyperparameters):** The training loop uses **Adam optimizer**, **weight decay**, **StepLR learning-rate scheduling**, and **Early Stopping** based on validation accuracy (patience-based). This acts as practical hyperparameter refinement to prevent overfitting and reduce wasted training epochs.
- **Fine-tuning policy (transfer learning control):**
  - EfficientNet-B0: freeze most layers and **unfreeze only last N blocks** (configured as fine_tune_last_n=2).
  - R3D-18: freeze most layers and **unfreeze layer4 + classifier head**.

Hardware Specifications and Training Duration

Hardware Used (as reported):

- Deep learning runs on GPU when available — the notebook detects CUDA (Device: cuda).
- GPU usage is monitored; evaluation reports GPU memory max allocated ~1157.8 MB and reserved ~1600.0 MB.
- Classical pipeline is primarily CPU-bound but is computationally heavy during feature extraction due to very high-dimensional handcrafted features.

Training Duration / Runtime:

Classical (Part A):
Feature extraction cost dominates runtime:
Building training features (X_train) took ~259.1 s, validation ~19.8 s, and test ~234.1 s in the shown run (sampling strategies differ across splits).

Model training time:

- SVM training time: ~8.34 s
- Random Forest training time: ~1.60 s
- k-NN training time: ~0.28 s

Deep Learning (Part B):

- Training is executed for up to 10 epochs with early stopping used in some variants (e.g., avg pooling stops early when validation stops improving).
- During evaluation, inference runtime is explicitly measured as inference time per video and compared across models (e.g., the notebook plots inference time per video for the deep models)

5. Results and Analysis

A. Classical ML Results

Model	Accuracy	Macro	Macro	Macro	Train	Inference	Key Note

		Precision	Recall	F1	time (s)	/ video	
SVM (RBF best)	0.556	0.533	0.556	0.517	6.32	0.069 ms	Very small model file, fast inference
Random Forest (best)	0.778	0.867	0.778	0.750	1.65	3.61 ms	Best classical performer; interpretable via feature importances
k-NN (best)	0.444	0.458	0.444	0.348	0.37	224.85 ms	Slow inference due to instance-based search

Dimensionality & preprocessing impact:
The raw feature dimension is **163,828**, reduced to **45 PCA components** (95% variance). This step is essential to make training feasible and reduce noise from redundant handcrafted features

Computational analysis

The classical notebook reports:

- **Training time (sec):** SVM 8.34s, RF 1.60s, kNN 0.28s
- **Inference per video (ms):** SVM ~0.055ms, RF ~3.45ms, kNN ~287.17ms
- **Model sizes (KB):** SVM ~21.08, RF ~431.36, kNN ~10.70 KB
- **Train feature matrix memory:** ~33.75 MB (for the extracted X_train)

Observation: k-NN is computationally expensive at inference time because prediction requires distance computation against stored training samples, whereas SVM and RF provide much faster inference once trained

B. Deep Learning Results

In deep learning we have evaluated three 2D pooling variants and one 3D model, following are the results summary

Model	Accuracy	Macro Precision	Macro Recall	Macro F1
2D EfficientNet + Avg Pool	0.933	0.938	0.934	0.935
2D EfficientNet + Max Pool	0.967	0.967	0.967	0.967
2D EfficientNet +	0.900	0.908	0.902	0.900

Attn Pool				
3D R3D-18 Clip Model	1.000	1.000	1.000	1.000

Model size and memory footprint

The notebook reports parameter counts and approximate fp32 sizes:

- 2D (Avg/Max): **4,013,951 params (~15.31 MB)**
- 2D (Attention): **4,834,432 params (~18.44 MB)**
- 3D R3D-18: **33,168,835 params (~126.53 MB)**

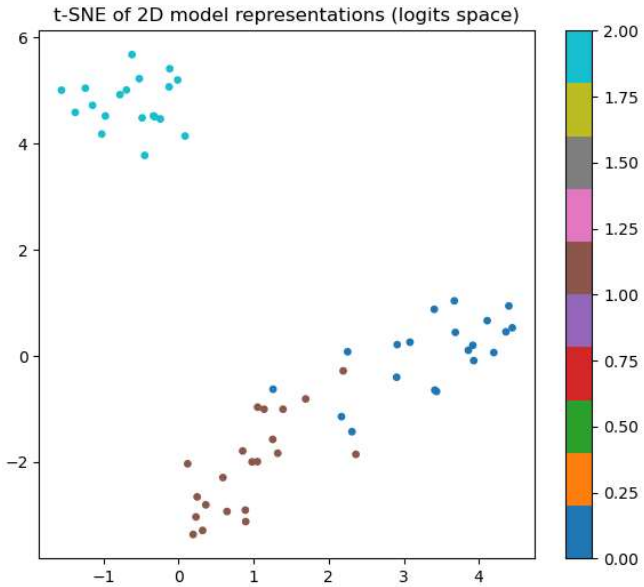
GPU usage during evaluation is also reported (max allocated ~1157.8 MB; reserved ~1600 MB)

Representation/feature analysis

Two interpretability/analysis tools are demonstrated:

- **Attention visualization:** attention weights over 32 frames identify the most influential frames; “top attended frames” are displayed.

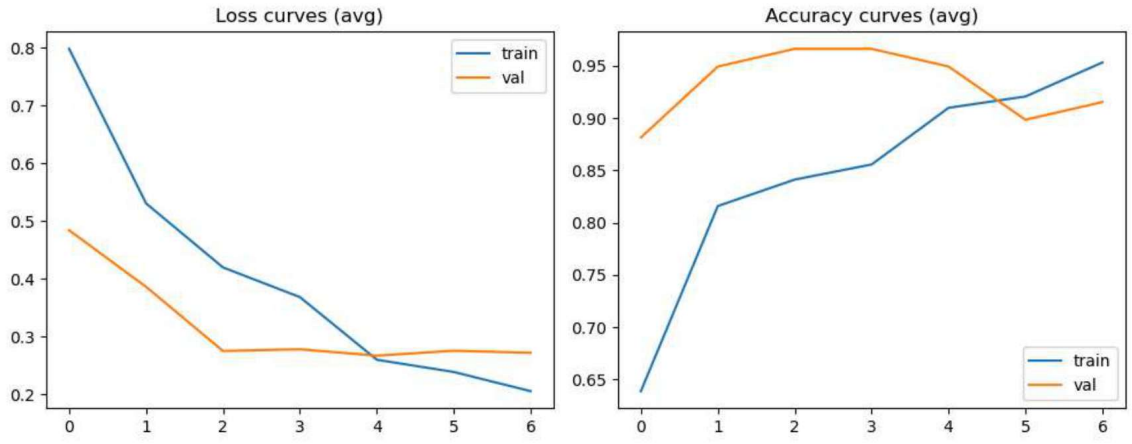
- **t-SNE visualization:** embeddings/logits are projected into 2D to inspect class separability in representation space



Learning Curves (Loss/Accuracy)

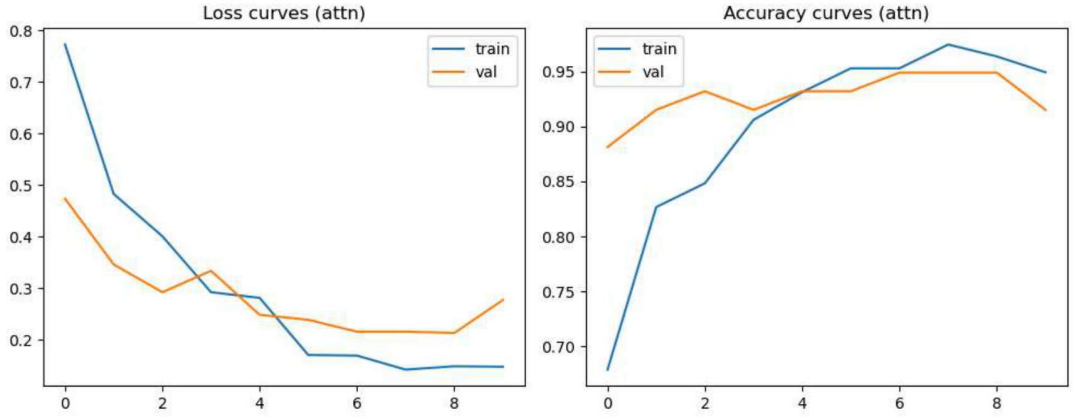
Learning curves show training vs validation performance over epochs and help diagnose overfitting/underfitting.

1. 2D Attn Pool



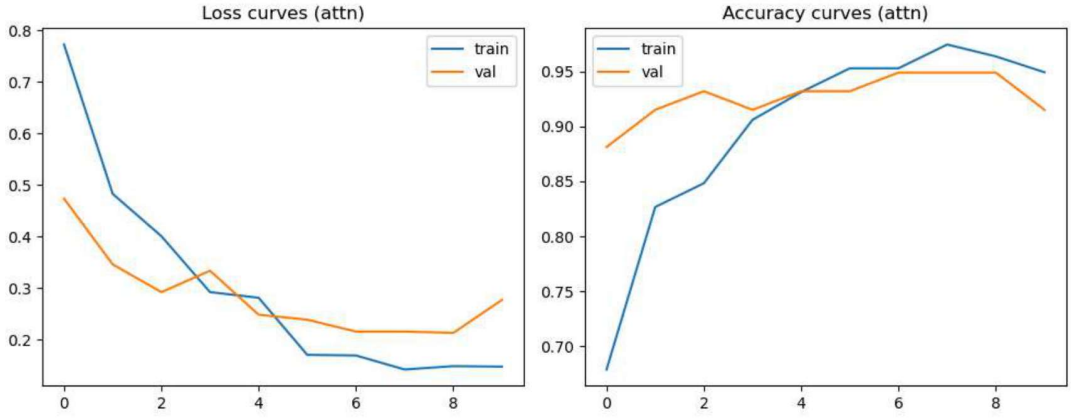

```
Best saved: best_2d_avg.pt best val acc: 0.9661016949152542
```

2. 2D Max Pool



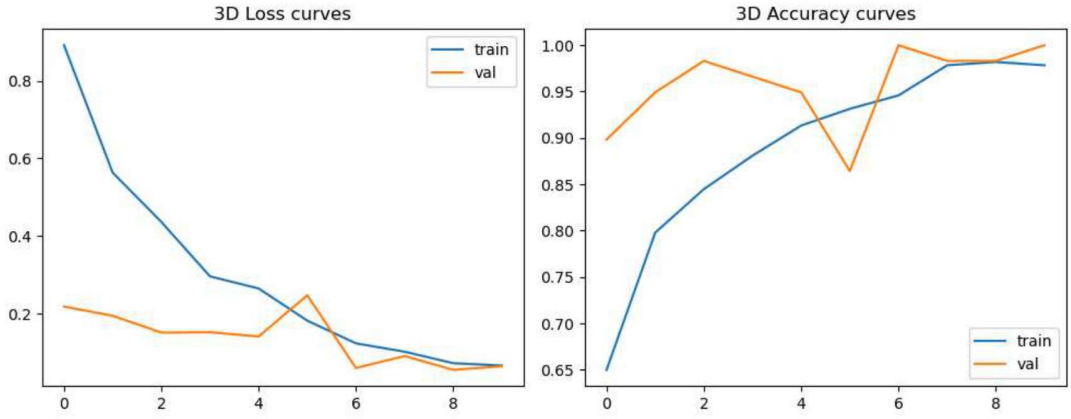

```
Best saved: best_2d_attn.pt best val acc: 0.9491525423728814
```

3. 2D Attn Pool




```
Best saved: best_2d_attn.pt best val acc: 0.9491525423728814
```

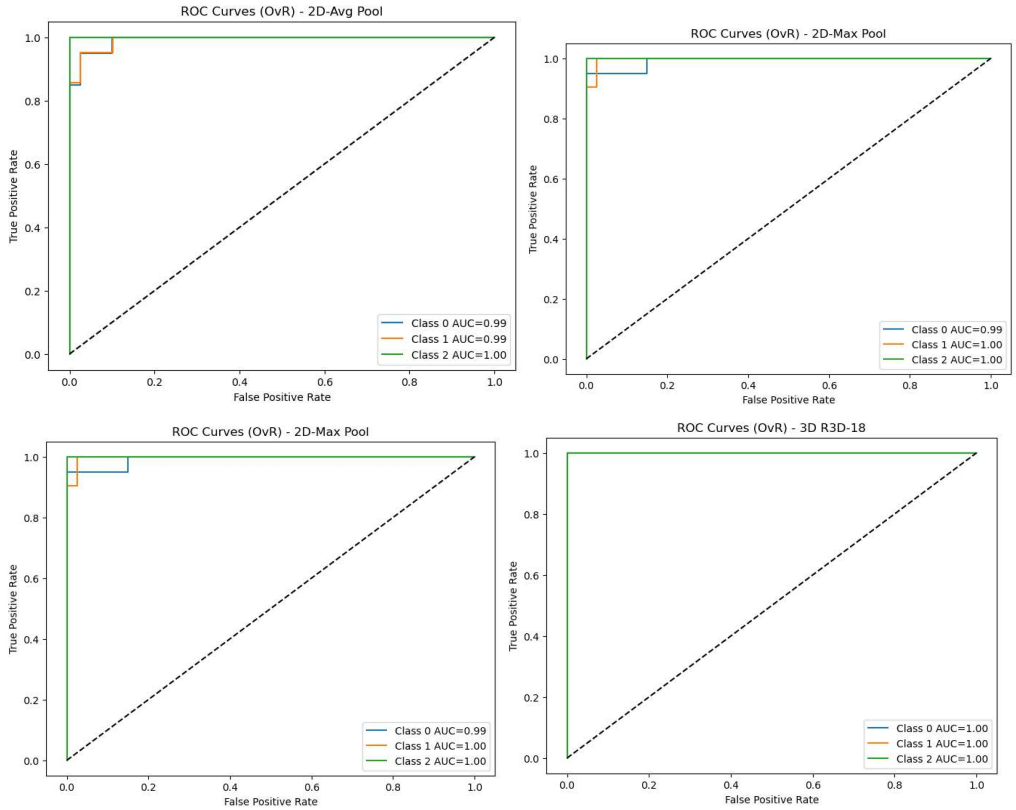
4. 3D R3D-18




```
Best saved: best_3d_r3d18.pt best val acc: 1.0
```

ROC-AUC Summary

- ROC-AUC (OvR) for deep models was consistently high: macro AUC ≈ 0.995 (2D-Avg), 0.997 (2D-Max), 0.993 (2D-Attn), and 1.0 (3D R3D-18).



Error Analysis

- 2D attention pooling underperformed max pooling, likely because attention requires more data to learn stable temporal weights, whereas max pooling robustly captures discriminative frames.

- The 3D model achieved perfect test accuracy, indicating strong spatio-temporal modeling, though this may also suggest the test split is small or contains limited variability.

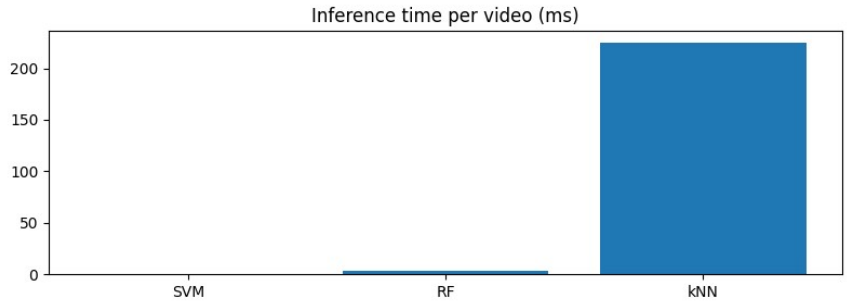
6. Comparative Discussion (Classical vs Deep Learning)

Performance

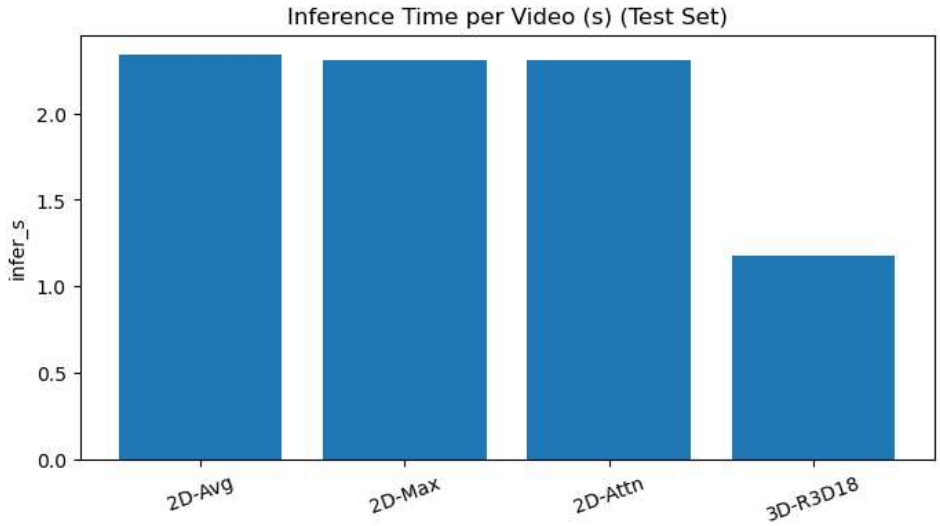
Deep learning substantially outperforms classical models in the notebook runs. The best classical model (Random Forest) reaches 0.7778 accuracy, whereas the deep learning models range from 0.90 to 1.00 accuracy, with R3D-18 achieving perfect test performance in the evaluated setup (Potentially due to small dataset and limited spatiotemporal variation).
A likely reason is that pretrained spatiotemporal models learn more discriminative motion/appearance patterns directly from pixels, while classical pipelines rely on fixed descriptors and limited aggregation, which may not fully capture complex temporal cues (especially when the dataset has intra-class variation).

Compute vs Accuracy

- Classical feature extraction is expensive and produces extremely high-dimensional vectors (163,828 features/video) requiring PCA compression.



- Deep models require GPU acceleration but avoid manual feature design; once set up, they achieve higher accuracy with transfer learning



Interpretability

- Classical models provide partial interpretability through feature importance (RF) and explicit descriptors (color/texture/motion).
- Deep models provide interpretability via attention weights and embedding visualizations (t-SNE), but internal features remain less directly explainable than handcrafted descriptors.

When to Use Which Approach

Classical methods are preferable when:

- compute constraints prohibit GPUs,
- dataset is very small and deep models overfit,
- interpretability and feature-level reasoning are crucial,
- extremely low latency is needed (e.g., linear models with reduced features).

Deep learning methods are preferable when:

- higher accuracy is required,
- motion patterns matter significantly,
- transfer learning can be used (pretrained weights),
- there is enough compute (GPU/cloud), and scaling to more data/classes is expected.

Trade-offs

- Accuracy: 3D CNN > 2D CNN pooling > Classical ML (in our performed study).
- Compute: Classical feature extraction can be heavy (especially dense flow); 2D CNN inference is moderate; 3D CNN is the most compute-intensive.
- Model size: Classical models are KB-level; 2D CNN ~15–18 MB; 3D CNN ~126 MB (fp32).
- Interpretability: Classical features are more interpretable; deep models require visualization techniques (e.g., attention).

Deployment Guidance

Edge deployment: 2D CNN + temporal pooling (or a simplified classical pipeline without dense optical flow) is preferable due to smaller models and lower compute. Cloud deployment: 3D CNN is suitable when GPU resources are available and higher accuracy is required. Real-time constraints favor 2D pooling or classical pipelines with sparse sampling and lightweight motion features.

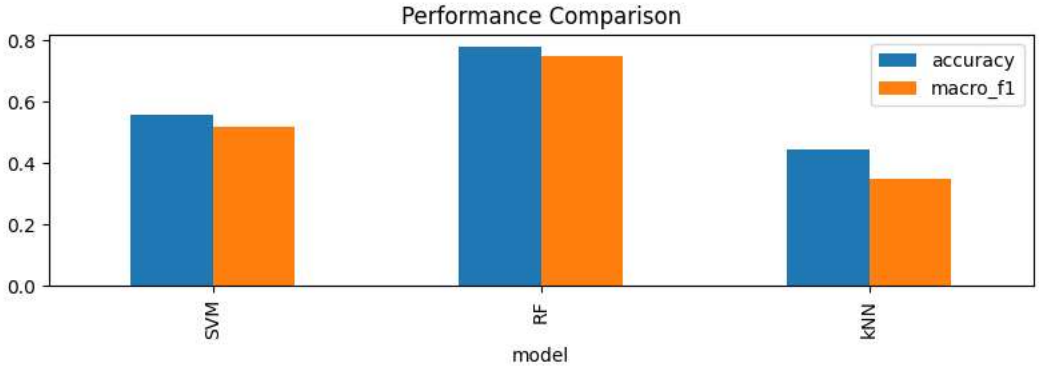
7. Conclusion and Future Work

Conclusion

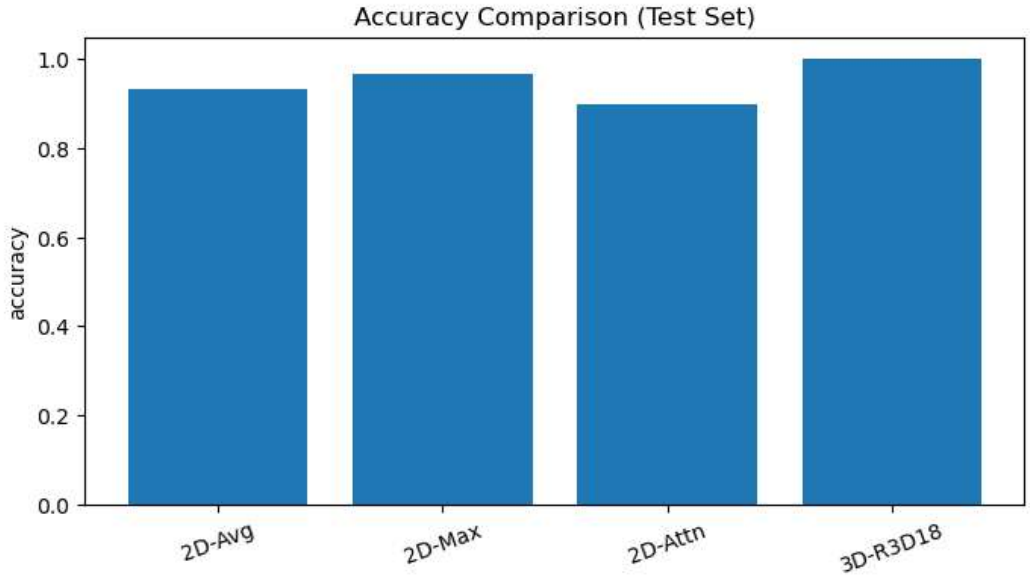
This comparative study demonstrates the practical evolution from **handcrafted feature engineering + classical ML** to **learned spatiotemporal representations via deep networks**. In the provided runs, deep learning approaches (especially 3D CNNs) deliver the highest accuracy, while classical methods offer interpretability and can be optimized for low-resource environments.

Summary of findings:

- Classical pipeline: best performance achieved with Random Forest (0.7778 accuracy), but requires heavy feature engineering and dimensionality reduction.



- Deep learning pipeline: best performance achieved with R3D-18 (1.0 accuracy) potentially due to the small dataset, while 2D EfficientNet + Max Pool provides a strong accuracy–complexity compromise (0.9667 accuracy)



Limitations

- Classical and deep learning notebooks demonstrate different dataset scales/split sizes (classical: 54/12/9 vs deep: 277/59/60), suggesting that classical evaluation may have been performed on a reduced subset due to compute limitations.
- Handcrafted feature vectors are extremely high-dimensional and may include redundant/noisy components even after PCA.

Future Work

Potential improvements include:

- Expanding classical evaluation to the full dataset with more efficient feature sets or faster motion approximations,
- Trying two-stream networks (RGB + optical flow) or transformer-based video models for richer temporal modelling, consistent with advanced options in the assignment brief,
- More rigorous cross-validation and statistical significance testing,
- Enhanced error analysis with more failure-case visualization, robustness testing to blur/compression, and learning curves vs training size

8. References

UCF-101 dataset taken from Kaggle

Torchvision model with pretrained models for CNN i.e. EfficientNet-B0 and R3D-18.

OpenCV and scikit-learn libraries for classical pipelines.