Project_info

RWF-2000 Video Violence Detection System: Project Synthesis

1. Project Overview

The RWF-2000 Violence Detection System project aimed to develop a machine learning pipeline capable of classifying short video segments as either 'Violence' or 'NonViolence' using the RWF-2000 dataset. The project underwent three major versions, with each subsequent iteration addressing a critical flaw in the preceding architecture to ultimately deliver a temporally aware, high-accuracy model.

Final Core Technology (v3.0):

- **Architecture:** X3D-M (eXpanded 3D Convolutional Network).
- **Framework:** PyTorch.
- **Key Feature:** Transfer learning using Kinetics pre-trained weights to capture motion features across 16-frame video clips.

2. Technical Evolution and Flaw Resolution Summary

The development cycle was defined by the transition from static image classification to true temporal sequence analysis.

Version	Core Architecture	Primary Flaw	Corrective Action in Next Version
v1.0	PyTorch R3D-18	**Critical Data Failure:** Placeholder tensor used instead of actual video data. System was non-functional.	Fixed the intention, but introduced a new architectural flaw in V2.0.
v2.0	TensorFlow MobileNetV2	**Critical Temporal Flaw:** Model trained as a 2D image classifier, destroying video temporal context (motion).	Transitioned to a 3D CNN (X3D-M) in V3.0 to restore temporal feature extraction.

v3.0 (Final)	PyTorch X3D-M	**Procedural/Optimization Flaws:** Missing standard channel-wise normalization, no data augmentation, static Learning Rate.	Architectural integrity achieved. Minor optimization necessary for future stability.

3. Final Performance Metrics (v3.0)

The final model, utilizing the X3D-M architecture, demonstrated strong generalization capabilities on the held-out test set, confirming that the temporal features are being effectively learned.

Metric	Result	Context
Test Accuracy	**96.50%**	High reliability for automated detection.
F1-Score (Weighted)	**96.50%**	Excellent balance between Precision and Recall.
Recall (Violence)	**97.00%**	Demonstrates the model's high success rate in identifying true violence clips (minimizing missed detections).

Confusion Matrix

The final results show a high concentration on the diagonal, with a minor bias in favor of **Recall for the 'Violence' class**, which is crucial for a real-time alerting system.

- **False Negatives (8):** Only 8 out of 200 violence clips were missed.
- **False Positives (6):** Only 6 out of 200 non-violence clips were flagged incorrectly.

4. Real-Time Deployment and Application

The finalized X3D-M model weights have been deployed in a standalone Python desktop application for real-time monitoring and prediction.

Component	Technology	Functionality
Application	Tkinter, OpenCV	Provides the graphical user interface (GUI) for a

	(cv2)	"CCTV Monitoring System."
Camera Feed	cv2.VideoCapture(0)	Captures frames from the local webcam in a continuous update_frame loop.
Prediction Logic	PyTorch, X3D-M	Continuously maintains a **16-frame buffer**. When full, the buffer is transformed, stacked into a 3D tensor, and fed to the model for inference.
Output	Softmax Probability	Displays real-time confidence scores for 'Violence' and 'Non Violence' and logs detection events to a history listbox.

5. Current Status and Future Recommendations

Current Status: The system is architecturally sound, achieves production-level performance metrics, and is successfully deployed in a Python application for real-time video analysis.

Future Recommendations (Optimization Phase):

1. **Data Normalization:** Implement standard channel-wise (mean/std) normalization to ensure input data aligns with the Kinetics pre-trained weights.
2. **Robustness via Augmentation:** Introduce spatial and temporal augmentations (e.g., random crop, temporal jitter) on the training set to improve model resilience against real-world variability.
3. **Optimization Refinement:** Introduce a Learning Rate Scheduler to manage the 0.001 static LR, optimizing convergence and potentially enhancing final accuracy marginally.

v1.0

RWF-2000 Video Violence Detection System v1.0 (PyTorch R3D-18)

1. Executive Summary

This document outlines the fourth iteration of the video violence detection system, shifting the core architecture to the standard **R3D-18 (ResNet 3D)** model from torchvision.models.video. This pipeline is designed for robustness and utilizes **Kinetics-400 pre-trained weights** for transfer learning. Crucially, the data pipeline is configured to read **raw video files** on-the-fly using a custom RWF2000VideoDataset, which requires an external decoding library (like Decord, as suggested in the code) for frame extraction and sampling.

2. Model Architecture and Configuration

2.1. Base Model and Customization

- **Base Model: R3D-18** (video_models.r3d_18), a 3D Convolutional Network based on the ResNet architecture.
- **Pre-trained Weights:** The model is initialized with **Kinetics-400 V1** weights (video_models.R3D_18_Weights.KINETICS400_V1), ensuring the backbone is well-initialized for motion feature extraction.
- **Customization:** The R3DViolenceDetector class replaces the final classification layer (self.base_model.fc) to map features to the two target classes ('Fight', 'NonFight'):

2.2. Training Configuration

Parameter	Value	Description
Model	R3D-18 (Kinetics-400 Pre-trained)	ResNet 3D model.
Input Shape	**(3, 16, 160, 160)**	(Channels, Frames, Height, Width).
Batch Size	**4**	Small batch size suitable for video processing.
Epochs	2	Set for a brief fine-tuning test; should be increased in production.

Optimizer	Adam	Standard optimization.
Learning Rate (LR)	1e-4	Low rate required for stable transfer learning/fine-tuning.
Loss Function	nn.CrossEntropyLoss	Standard classification loss.
Device	CUDA or CPU	Defaults to the first available GPU (cuda:0).

3. Data Pipeline and Video Handling

3.1. RWF2000VideoDataset Overview

This custom Dataset is responsible for scanning the RWF-2000 directory structure and preparing the necessary parameters for the R3D-18 model.

- **Input Data:** The dataset expects raw video files (.mp4, .avi, .mov, etc.) organized into class folders (TRAIN_ROOT_DIRECTORY/Fight, TRAIN_ROOT_DIRECTORY/NonFight).
- **Input Dimensions:**
  - Temporal Clip Length: **16 frames** (clip_len).
  - Spatial Frame Size: (frame_size).
- **Normalization:** The dataset explicitly defines and applies **Kinetics-400 mean and standard deviation** values, ensuring the input matches the pre-training conditions of the R3D-18 weights.

3.2. Video Decoding and Sampling (User Implementation Required)

The __getitem__ method currently contains a placeholder and requires the user to implement the video decoding logic.

1. **Decoding:** A library like **Decord** is recommended to efficiently read the raw video data from disk.
2. **Sampling:** frames must be sampled (e.g., linearly spaced) across the video's total duration.
3. **Format Conversion:** The output must be a PyTorch Tensor of shape , which is .

NOTE: Without the actual video decoding logic, the training will run on randomized tensor data, yielding non-representative results.

4. Fine-Tuning Execution

The train_and_finetune function executes the training loop. Since this version does not implement a multi-stage fine-tuning schedule or validation checkpointing (unlike v2.0), it performs a direct optimization pass over the training data.

1. **Initialization:** The model is instantiated and moved to the target device. The optimizer uses a low learning rate () to prevent large gradient updates from corrupting the pre-trained weights.
2. **Training:** The model is trained for , calculating the CrossEntropyLoss and updating weights via the Adam optimizer.
3. **Output:** After completion, the final fine-tuned model state dictionary is saved to **r3d18_rwf2000_finetuned.pth**.

5. Key Dependencies

- torch, torchvision, torchaudio (updated versions).
- numpy, pandas.

v2.0

RWF-2000 Video Violence Detection System v2.0 (TensorFlow/Keras)

1. Executive Summary

This document details the implementation of a violence detection system using **TensorFlow 2.x and Keras**. The model employs **Transfer Learning** based on the MobileNetV2 architecture, optimized for lightweight deployment. A key feature of this pipeline is its aggressive use of image augmentation via the imgaug library during the frame extraction process, which is designed to enhance model robustness and generalization.

2. Architecture and Configuration

2.1. Model Architecture

The model is built using the Keras Functional API, adopting a classic transfer learning approach:

- **Base Model: MobileNetV2**, pre-trained on ImageNet.
    - **Configuration:** include_top=False and pooling='avg'.
- **Classification Head:** A single dense layer (Dense(1, activation="sigmoid")) is attached to the averaged output of the MobileNetV2 backbone.
- **Training Strategy (Freezing):** During the initial training phase, the weights of the baseModel (MobileNetV2) are frozen (layer.trainable = False), allowing only the custom classification head to learn.

2.2. Hyperparameters and Optimizer

Parameter	Value	Description
Framework	TensorFlow 2.x / Keras	
Base Model	MobileNetV2 (Pre-trained)	Feature Extractor.
Input Shape		Resized image input dimensions.
Loss Function	binary_crossentropy	Standard for binary classification (Violence/NonViolence).
Initial Optimizer	Adam	Used for compiling the

		model.
Epochs	(Max)	Limited by Early Stopping and custom callbacks.
Batch Size	(Base)	Adjusted based on TPU initialization.
Regularization	l2(0.0001)	Kernel L2 regularization applied to weights.

2.3. Learning Rate (LR) Schedule

A custom lrfn function is implemented to manage the learning rate dynamically across epochs, typically following a cyclical or decay pattern to maximize convergence speed and stability:

- **Initial LR (start_lr):**
- **Max LR (max_lr):**
- **Ramp-up:** epochs (Linearly increases LR from to).
- **Decay:** Exponential decay with a factor of after the ramp-up phase.

3. Data Pipeline and Preprocessing

The pipeline focuses on efficient frame sampling and rigorous augmentation directly during the extraction phase.

3.1. Frame Extraction and Sampling

The video_to_frames function handles video decoding, frame sampling, and augmentation:

1. **Sampling Rate:** Only every frame is processed (ID % 7 == 0). This reduces temporal redundancy and minimizes dataset size.
2. **Color Space:** Frames are converted from BGR to RGB format (cv2.COLOR_BGR2RGB).
3. **Resizing:** Frames are resized to the target spatial dimension.

3.2. Image Augmentation (via imgaug)

A sequence of aggressive augmentations is applied to every sampled frame, significantly increasing data variability:

Augmentation	imgaug Transform	Effect
Horizontal Flip	iaa.Fliplr(1.0)	Flips the image horizontally (probability).

Brightness	iaa.Multiply((1, 1.3))	Randomly increases brightness by up to .
Zoom	iaa.Affine(scale=1.3)	Scales the image up to times its size.
Rotation	iaa.Affine(rotate=(-25, 25))	Randomly rotates the image between and .

3.3. Data Preparation and Splitting

1. **Data Structure:** All frames are collected into X_original and flattened into a vector before splitting.
2. **Train/Test Split:** StratifiedShuffleSplit is used with and to ensure the class distribution of 'Violence' and 'NonViolence' is maintained across the Training and Test sets.
3. **Normalization:** Input data is normalized to the range before model input: .

4. Training and Evaluation

4.1. Callbacks and Early Stopping

The training uses a comprehensive set of callbacks to manage the process:

- **myCallback (Custom):** Stops training immediately if training accuracy reaches , acting as a hard upper limit.
- **LearningRateScheduler:** Implements the custom LR schedule defined by lrfn.
- **ModelCheckpoint:** Saves weights to ModelWeights.h5 only when validation loss (val_loss) achieves a new minimum (save_best_only=True).
- **EarlyStopping:** Monitors validation loss (val_loss) with a patience of epochs and a minimum delta () of , restoring the best weights found.
- **ReduceLROnPlateau:** Reduces the learning rate if validation loss plateaus (patience).
- **TensorBoard:** Logs training metrics for visualization.

4.2. Final Evaluation

After training, the best weights saved by the ModelCheckpoint are reloaded. The model is evaluated on the dedicated test set (X_test_nn, y_test) using the following metrics:

1. **Loss and Accuracy:** The model_summary function calculates and prints test loss and test accuracy, alongside the metrics from the best saved epoch.
2. **Confusion Matrix:** A heatmap visualization of the confusion_matrix is generated to show correct and incorrect predictions for each class.
3. **Classification Report:** A detailed report provides precision, recall, and F1-score for the 'NonViolence' and 'Violence' classes.

The final model weights are saved to modelnew.h5.

5. Evaluation Results (Best Epoch)

The training was successfully halted by the Early Stopping callback, which restored the weights corresponding to the best validation loss. The results are based on the **Best Epoch (31)**.

5.1. Training Summary

Metric	Training Value	Test/Validation Value	Analysis
Best Epoch	31	31	Model saved weights from epoch 31.
Loss	0.1121	**0.1163**	Very small gap (0.0042) between train and test loss, suggesting minimal overfitting, likely due to the strong regularization and LR schedule.
Accuracy	0.9617	**0.9578**	High accuracy on the test set (4809 samples), which is excellent for an image classifier.

5.2. Classification Report (Test Set)

Class	Precision	Recall	F1-Score	Support
NonViolence	0.96	0.95	0.95	2243
Violence	0.96	0.96	0.96	2566

Weighted Average	0.96	0.96	0.96	4809

5.3. Final Assessment of Results

1. **High Metrics:** The reported $\approx 96\%$ accuracy and F1-score are exceptionally high for a complex classification task.
2. **Stability:** The low difference between training and test loss/accuracy confirms the optimizer, LR schedule, and regularization effectively prevented overt overfitting *to the static images*.
3. **The Temporal Caveat:** These high metrics **must be interpreted with caution**. Since the model only sees static images (frames) and ignores the flow of time, the model is likely succeeding by learning **spatial cues** (e.g., specific postures, blood, or high-contrast scenes common in the 'Violence' class). **The system will fail at distinguishing two identical frames where one leads to a fight and the other leads to a handshake.**

v3.0

RWF-2000 Video Violence Detection System v3.0 (PyTorch X3D-M)

Final Model and Deployment Report

1. Executive Summary

This document reports on the final version (v3.0) of the violence detection system. The iteration successfully transitioned to the **X3D-M (eXpanded 3D) Convolutional Network** using PyTorch, resolving the critical temporal context flaw present in the V2.0 image-based approach. The model's reliance on 3D convolutions ensures that motion and temporal features are correctly extracted and analyzed.

The system was fully trained for 100 epochs with best-model checkpointing. The final evaluation on the dedicated test set yielded high and stable results, confirming the viability of the X3D-M architecture for this task.

Final Metric	Result	Basis
Test Accuracy	**96.50%**	Unbiased performance on held-out videos.
F1-Score (Weighted)	**96.50%**	Indicates high reliability in both classes.
Temporal Basis	**3D CNN (X3D-M)**	Correctly analyzes 16-frame motion clips.

2. Model Architecture and Configuration

2.1. Base Model and Customization

Parameter	Value	Description

Core Model	X3D-M	State-of-the-art 3D CNN, optimized for video.
Weights	Kinetics Pre-trained	Loaded via `torch.hub.load` for transfer learning.
Customization	Final Projection Layer	Modified to map features to 2 output classes (`Violence`, `NonViolence`).

2.2. Training Configuration

Parameter	Value	Description
Framework	PyTorch	
Epochs	100	Total training cycles.
Batch Size	8 (Per GPU)	Configured for `DataParallel` if available.
Input Shape	(3, 16, **112, 112**)	(C, T, H, W). **Correction:** The preprocessing script used 112×112 and not the 256×256 placeholder.
Optimizer	Adam	Standard optimization. **Static Learning Rate of 0.001.**
Loss Function	`nn.CrossEntropyLoss`	

3. Data Pipeline Analysis and Flaws

3.1. Preprocessing and Data Structure

The pipeline correctly preprocesses the RWF-2000 raw video files into NumPy array (`.npy`) format before training. Each sample is a tensor of shape: $(\text{Frames}, H, W, \text{Channels}) \rightarrow (16, 112, 112, 3)$. The custom `VideoDataset` handles the transposition to the required PyTorch format: $(\text{Channels}, \text{Frames}, H, W)

\rightarrow (3, 16, 112, 112)$.

3.2. Procedural Flaw: Missing Channel-Wise Normalization

The preprocessing step includes data scaling (`frames / 255.0`) but critically **omits the channel-wise normalization** (subtracting mean and dividing by standard deviation) required for optimal performance when using Kinetics pre-trained weights.

Impact: Training convergence may be slower and final performance potentially reduced due to the input data not matching the distribution on which the model was pre-trained.

4. Final Evaluation Results (Best Model Checkpoint)

The model was evaluated using the weights saved at the point of the **highest validation accuracy** (the `best_model`). The total test set size was 400 samples.

4.1. Classification Report

Class	Precision	Recall	F1-Score	Support
NonViolence	0.9697	0.9600	0.9648	200
Violence	0.9604	0.9700	0.9652	200
Weighted Average	0.9650	0.9650	0.9650	400

4.2. Confusion Matrix

The confusion matrix confirms a balanced performance, with a slight bias towards minimizing False Negatives (missed violence), which is desirable for a detection system.

True / Predicted	NonViolence	Violence
True	**194** (Correct)	6 (False Positive)

NonViolence		
True Violence	8 (False Negative)	**192** (Correct)

5. Conclusion and Recommendations for Future Optimization

Version 3.0 provides a high-performing, temporally aware solution. However, future work should focus on integrating standard optimization techniques that were omitted in this version.

1. **Integrate Channel-Wise Normalization: IMMEDIATE PRIORITY.** Modify the `ToTensor` transform to apply the Kinetics-400 mean and standard deviation to the input clips.
2. **Implement Data Augmentation:** Introduce spatial augmentations (e.g., Random Crop, Horizontal Flip) and temporal augmentations (e.g., time jitter) within the `VideoDataset` to further improve the model's robustness and generalization.
3. **Add Learning Rate Scheduler:** Replace the static 0.001 LR with a dynamic scheduler (e.g., step decay or cosine annealing) to fine-tune the optimization process and potentially achieve faster, better final convergence.