Replay Extraction

Luke Sullivan

lcs9595

1 Introduction

The manual process of identifying and saving sports highlights is inefficient, demanding significant time and resources. This project directly confronts this problem by developing an automated system to extract key highlight moments – specifically, the replay segments that follow live action – directly from sports broadcast video. The proposed solution employs a non-learning-based approach, analyzing inherent visual patterns within the broadcast rather than relying on complex models.

This model is adapted from the framework introduced by Javid et al. (2016), which leverages common visual cues to distinguish replays from live play. Specifically, it identifies Gradual Transitions (GTs) like fades or dissolves that mark the start and end of replays, and confirms the absence of Score Captions (SCs) which are typically only visible during live action. While grounded in this established technique, this project's specific contribution focuses on optimizing and refining this detection framework to improve its computational efficiency and overall performance.

2 Methodology

The project employs a two-stage approach to automatically detect replay segments in sports videos, designed for efficiency by utilizing streaming and parallel processing. The core idea is based on two common characteristics of broadcast replays: they are typically bracketed by gradual visual transitions and lack the score captions usually present during live play.

Gradual Transition Detection: The model analyzes the video frame-by-frame, comparing the luminance histogram of consecutive frames. Significant, sustained differences in these histograms, exceeding certain thresholds, are flagged as GTs (like fades or dissolves).

Candidate Replay Segment (RS) Identification: Time segments located between two consecutive GTs are examined. If the duration of such a segment falls within a plausible range for a replay, it is marked as a candidate RS.

Score Caption Absence Verification: Recognizing that SCs are usually absent during replays, the system checks for their presence within each candidate RS. This involves several steps:

- Each worker reads the frames for its assigned candidate segment(s).

- Frames undergo preprocessing, including grayscale conversion, downsampling, and illumination adjustment via morphological filtering.
- Temporal averaging across frames reduces noise and enhances static elements like SCs.
- Image binarization, based on frame statistics, isolates potential text regions.
- Optical Character Recognition (OCR) is applied to the entire frame.
- If OCR confidently detects a sufficient number of characters in **any** frame of the segment, an SC is considered present, and the segment is discarded as likely live gameplay.

Segments that pass this check (i.e., no SC is detected throughout the candidate RS) are confirmed as final replay segments.

3 Data Overview

The model presented in this project is designed to process standard broadcast sports video footage, specifically in common digital formats like MP4. The primary requirement is that the video contains typical broadcast elements, including potential replay segments indicated by transitional effects and score captions.

For this particular experiment, the input data consisted of video footage from the 2022 Super Bowl. This data was sourced directly from the official NFL YouTube channel to ensure authenticity and representative broadcast quality. The acquisition was streamlined using the yt-dlp command-line tool, which provided the seamless download of the required videos directly into the MP4 format for analysis.

4 Initial Code Base

The foundation for this project draws inspiration from the methodology presented by Javed et al., which combines GT detection and SC analysis for replay detection in sports videos.

Video and Image Handling (OpenCV): Central for video I/O and frame manipulation. The baseline approach reads all video frames sequentially into memory using cv2.VideoCapture, then processes them using functions like cv2.cvtColor, cv2.resize, cv2.morphologyEx, cv2.calcHist, and cv2.compareHist for tasks including preprocessing and GT detection.

Numerical Operations (NumPy): Provides efficient array structures and calculations for pixel-level image data, underpinning OpenCV operations during preprocessing, averaging, and binarization.

Optical Character Recognition (Pytesseract): Used for Score Caption (SC) detection by interfacing with the Tesseract OCR engine. The baseline sequentially passes candidate frames to Pytesseract for text extraction.

5 Optimization

To improve the performance and efficiency of the replay detection pipeline beyond the baseline implementation, several optimization strategies were explored, focusing on reducing memory consumption and execution time.

CUDA: GPU acceleration using NVIDIA's CUDA platform was explored via OpenCV's cv2.cuda module to potentially speed up image processing tasks. Key steps like SC preprocessing, temporal averaging, binarization, and GT histogram calculation were adapted to run on the GPU using cv2.cuda.GpuMat objects. However, this required managing data transfers between the CPU and GPU. Notably, a CPU workaround was necessary for histogram normalization within the GT detection loop due to compatibility issues encountered, requiring GPU-calculated histograms to be downloaded and normalized on the CPU before comparison. Critical components like OCR remained CPU-bound.

Multiprocessing: To accelerate the computationally intensive Score Caption (SC) verification stage, this optimization leverages CPU parallelism using Python's multiprocessing library. Candidate replay segments, identified after GT detection, are distributed across a pool of worker processes. Each worker independently handles its assigned segments by reading the necessary video frames directly and performing the complete SC detection sequence, including preprocessing and OCR. By executing these checks concurrently across multiple CPU cores, this approach significantly reduces the total time required to filter candidate segments compared to sequential processing.

Streaming: To handle large video files efficiently without exceeding memory limits, the project incorporated streaming optimizations. Instead of loading the entire video into RAM, key processes read data directly from the file as needed. Specifically, the GT detection stage analyzes frames sequentially from the input stream, requiring only the current and previous frame's histogram data in memory. Similarly, the final export process writes replay segments by reading the required frames directly from the source video file just before writing them to the output file. Furthermore, within the multiprocessing stage, each worker reads only the frames for its assigned segment, minimizing the memory footprint per process. This overall streaming approach ensures scalability to process lengthy, high-resolution videos.

8 Results

The model was tested on 5 minutes of the 2022 NFL Super Bowl, comprising 35,965 frames. Within this sequence, 5,858 frames were identified as ground truth replay segments. The optimized model detected 6,158 frames as belonging to replays. The key accuracy metrics achieved were:

- Overall Accuracy: **92.97%**
- F1-Score (Replay Class): **78.96%**

These results indicate a strong capability of the model to distinguish between replay and live segments, with a high overall accuracy and a reasonable F1-score, which balances precision and recall for the replay class.

Model Type	**GT Detection Time**	**SC Detection Time**	**Overall Time**
Python + numPy	74.66 seconds	248.13 seconds	403.42 seconds
CUDA	132.79 seconds	312.92 seconds	530.22 seconds
Multiprocessing	98.08 seconds	137.15 seconds	336.04 seconds
Multiprocessing + Streaming **(best)**	28.29 seconds	99.31 seconds	152.74 seconds

Table 1: Time results based on optimization strategy

The sequential Python baseline provided an initial performance benchmark. The CUDA implementation unexpectedly performed slower, likely hindered by data transfer overheads and necessary CPU workarounds. Multiprocessing significantly improved performance by parallelizing the SC detection. However, the combined Multiprocessing + Streaming approach proved most effective; streaming optimized I/O for GT detection, and coupled with multiprocessing, it delivered the fastest overall execution time, approximately 2.6 times faster than the baseline. This highlights the superior impact of CPU parallelism and efficient data handling over the attempted GPU acceleration for this specific application.

9 Conclusion

This project successfully developed and optimized a non-learning-based approach for automatically extracting replay segments from sports broadcast videos, addressing the limitations of manual highlight identification. By leveraging Gradual Transition detection to identify potential segment boundaries and Score Caption absence verification using OCR to filter live play, the system demonstrated strong performance on NFL Super Bowl footage, achieving 92.97% overall accuracy and a 78.96% F1-score for the replay class.

Crucially, the optimization efforts revealed significant performance differences. While an attempt to utilize GPU acceleration via CUDA resulted in slower execution times due to data transfer overheads and necessary CPU workarounds, CPU-bound optimizations proved highly effective. Implementing multiprocessing drastically reduced the time required for the computationally intensive SC verification stage. The most substantial improvement came from combining multiprocessing with data streaming, which minimized memory usage and optimized I/O for large video files. This configuration delivered the fastest performance, executing approximately 2.6 times faster than the sequential baseline. This indicated that optimizing CPU parallelism and data handling strategies offered superior performance gains compared to the attempted GPU acceleration, enabling efficient and scalable replay extraction from sports broadcasts.

10 Citations

[1] A. Javed, K. B. Bajwa, H. Malik, and A. Irtaza, "An Efficient Framework for Automatic Highlights Generation from Sports Videos," in *2019 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, Swat, Pakistan, 2019, pp. 1-6, doi: 10.1109/ICECCE47252.2019.8940770.
[2] L. Sullivan, “Replay Extraction Model”, GitHub Repository, 2025, github.com/TheeLuke/Replay-Extraction.git