Scene Detection Using OpenCV and Python

line 2: *dept. name of organization (of Affiliation)*
line 3: *name of organization (of Affiliation)*
line 4: City, Country
line 5: email address or ORCID

Abstract — **The rapid expansion of digital video content has placed significant demands on how we manage, analyse, and extract meaningful information from large datasets. This is especially true across various industries such as entertainment, security, and academia. As videos grow in size and complexity, traditional manual methods of content analysis are becoming inefficient and unsustainable. To address these challenges, a Flask-based web application has been developed, offering a streamlined platform for automated scene detection and analysis.**

This application uses OpenCV, a leading computer vision library, to examine video frames and determine significant changes in content, signalling scene transitions. The process is facilitated by adaptive algorithms that adapt to the varying characteristics of different videos, ensuring robust and accurate detection of distinct scenes. Once detected, the application provides start and end timestamps for each scene, enabling users to quickly identify and navigate to specific moments within a video.

In addition to identifying scenes, the application offers a range of features designed to make video management more efficient and user-friendly. For instance, it allows users to split videos into individual scenes, effectively creating a video outline that is easier to navigate and manage. This functionality can be particularly useful for tasks like analysing surveillance footage, splitting home videos into manageable segments, or generating engaging thumbnails for streaming platforms.

Furthermore, the application's architecture is designed for scalability and flexibility. By employing Flask, the system can easily integrate with other tools and services, and its deployment through Docker ensures consistent performance across various environments. This makes it suitable for a wide range of applications, from academic research to commercial video production.

Overall, this project demonstrates the potential of combining adaptive algorithms and computer vision to streamline video content management, offering a comprehensive solution for various industries and use cases. By automating scene detection and providing detailed analysis, the application helps users extract meaningful information from large video datasets, facilitating better organization and more efficient content exploration.

Keywords— Scene Detection, Video Analysis, Flask, OpenCV, Adaptive Thresholding, Frame Rate

I. Introduction

The rapid growth of video content has created a significant challenge: how to efficiently organize, analyse, and extract relevant information from vast amounts of footage. Whether it's sorting through hours of surveillance footage, creating engaging content for video streaming, or studying films for academic purposes, finding specific scenes or moments can be time-consuming and inefficient.

To address these challenges, this project introduces a web-based application that uses advanced algorithms to automatically detect scenes in videos. The application provides users with a simple interface to upload video files, analyse their content, and extract detailed information about scene transitions, including timestamps and visual thumbnails. This technology streamlines video management, offering a versatile solution for a wide range of applications.

By leveraging Flask for the backend, OpenCV for video analysis, and Docker for deployment, the application is designed to be robust, scalable, and easy to use. This report provides a comprehensive overview of the project, detailing the motivations behind its development, the methodology used to detect scenes, the features and functionality of the application, and the results achieved through its use.

. By using state-of-the-art technology and robust methodologies, this project aspires to set a new standard for organizing and managing personal photo collections.

II. Motivation

The motivation for this project arises from the growing demand for efficient video content analysis across various domains:

1. **Video Content Management**: As video content becomes more prevalent, managing large datasets requires automated solutions to detect and categorize scenes.
2. **Streaming and Entertainment**: For video streaming platforms, creating engaging thumbnails and allowing users to jump to specific scenes enhances the viewing experience.
3. **Surveillance and Security**: Analysing surveillance footage for specific events or activities can be time-consuming without automated scene detection.
4. **Academic and Film Studies**: Researchers studying films and videos often require tools to analyse and segment content for academic purposes.

These factors drive the need for an application that can detect scenes efficiently, providing users with detailed analysis and allowing for more effective video management.

III. Contributions and Objectives

The key contributions of this project include:

1. **Automated Scene Detection**: A web application that automatically detects scenes in video content using adaptive algorithms.
2. **User-Friendly Interface**: A simple and intuitive interface for uploading videos and accessing scene analysis results.
3. **Detailed Scene Information**: The application provides start and end timestamps for each detected scene, along with optional scene splitting and thumbnail generation.
4. **Versatile Use Cases**: The application serves a variety of purposes, from organizing home videos to analyzing surveillance footage and creating engaging streaming content.

The primary objectives of the project are:

1. To develop an automated scene detection tool that is easy to use and provides accurate results.
2. To create a user interface that allows users to upload videos and receive detailed scene analysis.
3. To offer additional features, such as scene splitting and thumbnail generation, to enhance video management and content creation.

IV. Methodology

The methodology for detecting scene cuts in a video involves several steps, each designed to capture transitions between scenes accurately. The process begins with uploading a video and ends with a detailed list of detected scenes, along with additional features like scene splitting and thumbnail generation.

A. Video Upload

Users upload video files through the Flask-based web interface. The system checks for allowed file extensions to ensure compatibility with the underlying video processing technology. Flask handles the routing and file management, allowing users to upload video files using a simple form. The uploaded videos are stored in a unique directory, using a secure method to handle filenames and prevent conflicts.

B. Scene Detection Algorithm

The core algorithm for scene detection is implemented in the **AdaptiveDetector** class, which uses adaptive scoring to identify significant changes in video content. Here's a step-by-step explanation of the process:

1. **Video Frame Analysis**: The video is opened using OpenCV, a popular computer vision library. The video is read frame-by-frame, allowing the algorithm to analyse each frame individually.

2. **Frame Components Extraction**: For each frame, the following components are extracted:

    - **Hue**: The colour information, extracted from the HSV colour space.
    - **Saturation**: The intensity of the colour.
    - **Luminance (Lum)**: The brightness or lightness of the frame.
    - **Edges**: Detected using edge detection techniques like Canny Edge Detection. This step involves applying a Gaussian blur to the luma channel and then detecting edges based on intensity

changes.

3. **Frame Scoring**: Once the components are extracted, the algorithm calculates a score representing the relative content change between consecutive frames. This score is derived from the mean pixel distance between the corresponding components of adjacent frames. The scoring considers differences in hue, saturation, luminance, and edges.

4. **Adaptive Thresholding**: The algorithm uses adaptive thresholding to determine whether a significant change has occurred between frames. This involves calculating a rolling average of the frame scores within a specified window width. If a frame's score is significantly higher than the rolling average by a specified adaptive threshold, it is identified as a potential scene cut.

5. **Parameters Affecting Output**:

   - **Adaptive Threshold**: A higher adaptive threshold requires a more significant change between frames to identify a scene cut, reducing the number of detected cuts. A lower threshold increases sensitivity, potentially resulting in more scene cuts.

   - **Window Width**: The window width determines the number of frames considered for the rolling average. A larger window smooths out fluctuations, while a smaller window makes the algorithm more sensitive to rapid changes.

   - **Minimum Scene Length**: This parameter defines the minimum number of frames between scene cuts. It helps prevent detecting short-duration scene cuts, ensuring more stable scene detection.

   - **Minimum Content Value**: This sets a minimum score value for a scene cut to be considered valid. It prevents false positives in cases where the change in content is minimal.

6. **Scene Identification**: As the video is processed, the algorithm records the frame number when a scene cut is detected. The start and end times for each scene are calculated based on the video's frame rate. This information is stored in a list of detected scenes, which is later used to generate the output.

C. Scene Splitting and Thumbnail Generation

The application provides additional features, such as scene splitting and thumbnail generation, to enhance video management and content exploration:

1. **Scene Splitting**: If enabled, the application creates individual video segments for each detected scene. This involves splitting the original video at the identified scene cuts and saving each segment to a separate file. This feature is useful for organizing long videos into manageable segments or for creating specific video clips.

2. **Thumbnail Generation**: The algorithm saves the frame where the scene cut occurs as a thumbnail image. This thumbnail is then displayed in the web interface, providing a visual representation of the detected scenes. It can also be used to create engaging video content, such as preview images for streaming platforms.

D. Results Display

After processing, the application displays the detected scenes on the web interface. The results include the following information for each scene:

- Scene number
- Start time and end time (formatted in HH:MM:SS.ms)
- Scene duration
- Thumbnail image for the scene cut start frame

The results can be downloaded or accessed through a unique URL, allowing users to retrieve specific scenes or download thumbnails for further use.

V. Functionality

The The video scene detection application is designed with a user-centric approach, offering an intuitive interface and a streamlined workflow. It guides users through each step, from video upload to scene analysis and result retrieval, ensuring a seamless experience. Here's a detailed breakdown of the application's functionality:

1. **Upload Page**: This is the entry point for users to upload their video files for analysis. The page includes an HTML form with a file input field, allowing users to select and upload a video file. The form is designed to accept various video formats, ensuring compatibility with the application. Users can also specify additional parameters to customize the scene detection process:

    - **Adaptive Threshold**: This parameter defines the sensitivity to scene changes. A higher value makes the algorithm more selective, while a lower value increases its sensitivity.

    - **Minimum Scene Length**: This determines the minimum number of frames between scene cuts. It helps prevent the detection of overly short scenes.

    - **Window Width**: This parameter affects the rolling average used for adaptive thresholding. A wider window smooths fluctuations, while a narrower window increases sensitivity to changes.

    - **Minimum Content Value**: This defines a minimum score required to identify a scene cut, helping to filter out insignificant transitions.

2. **Scene Detection**: Once a video is uploaded, the application begins processing. It uses the **AdaptiveDetector** class to analyze the frames, applying adaptive algorithms to detect scene transitions. This process involves reading the video frame-by-frame, calculating scores based on the difference in hue, saturation, luminance, and edges, and then applying adaptive thresholding to identify significant changes in content. The scene detection algorithm also considers user-defined parameters to ensure flexibility in the detection process.

3. **Scene Results**: After processing, the application generates a detailed list of detected scenes. Each scene is identified by a unique number, and the list includes start and end times (in HH:MM:SS.ms format), scene durations, and optional thumbnails for scene cuts. The results are displayed in a tabular format on the web interface, allowing users to quickly identify and navigate to specific scenes.

4. **Home Page**: The home page serves as the main landing page for the application. It provides an overview of the application's purpose and encourages users to start the analysis process by uploading a video. The page contains a brief description of the application's functionality and a button to navigate to the upload page.

VI. Technologies Used

The development and deployment of this application rely on a combination of powerful technologies, each contributing to the application's functionality and performance. Here's a detailed overview of the technologies used:

- **Flask**: Flask is a lightweight and flexible web framework used to build the application's backend. It handles routing, form submissions, and overall web application structure. Flask's simplicity and modularity make it ideal for building web-based applications with complex functionalities.

- **Python**: Python is the primary programming language for the backend and the scene detection algorithm. It offers extensive libraries and frameworks, allowing developers to build complex applications efficiently. Python's readability and ease of use make it a popular choice for backend development.

- **OpenCV**: OpenCV is a comprehensive computer vision library used for video frame analysis and scene detection. It provides various functions for image and video processing, allowing developers to implement complex algorithms for scene detection. OpenCV's edge detection, colour space conversions, and other image analysis features are critical to the application's functionality.

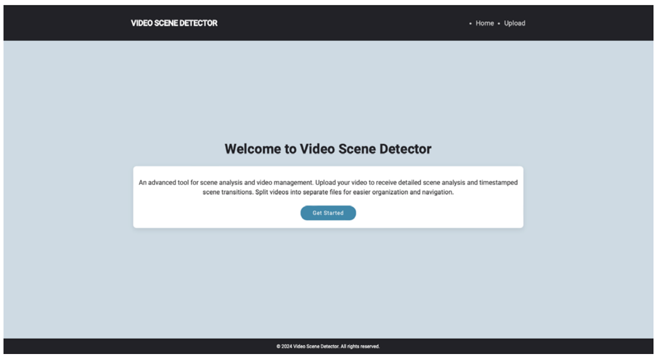
- **Docker**: Docker is used for deployment and environment management. It ensures that the application runs consistently across different platforms, making deployment easier and more reliable. Docker allows developers to create a consistent development environment, reducing compatibility issues and ensuring smooth deployment.

These technologies work together to create a robust and scalable application for automated scene detection. Flask provides the web framework, Python and OpenCV enable the backend functionality and scene analysis, and Docker ensures reliable deployment across various environments. The combination of these technologies results in a flexible and powerful application that can be used in various scenarios, from managing surveillance footage to analyzing films for academic purposes.

VII. RESULTS

The results section demonstrates the effectiveness of the Adaptive Detection Algorithm in identifying scenes of a video.

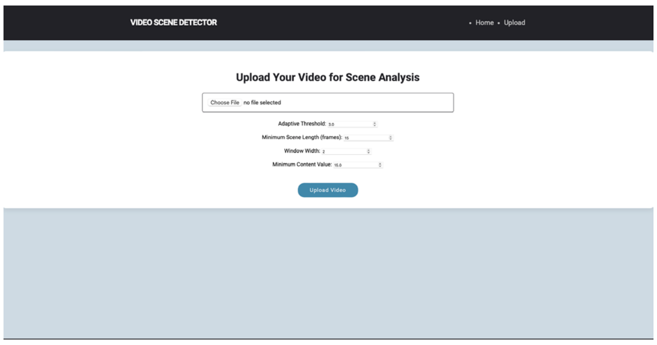
A. Home Page



Welcome to Video Scene Detector

An advanced tool for scene analysis and video management. Upload your video to receive detailed scene analysis and timestamped scene transitions. Split videos into separate files for easier organization and navigation.

B. Upload Page



Upload Your Video for Scene Analysis

Choose File no file selected

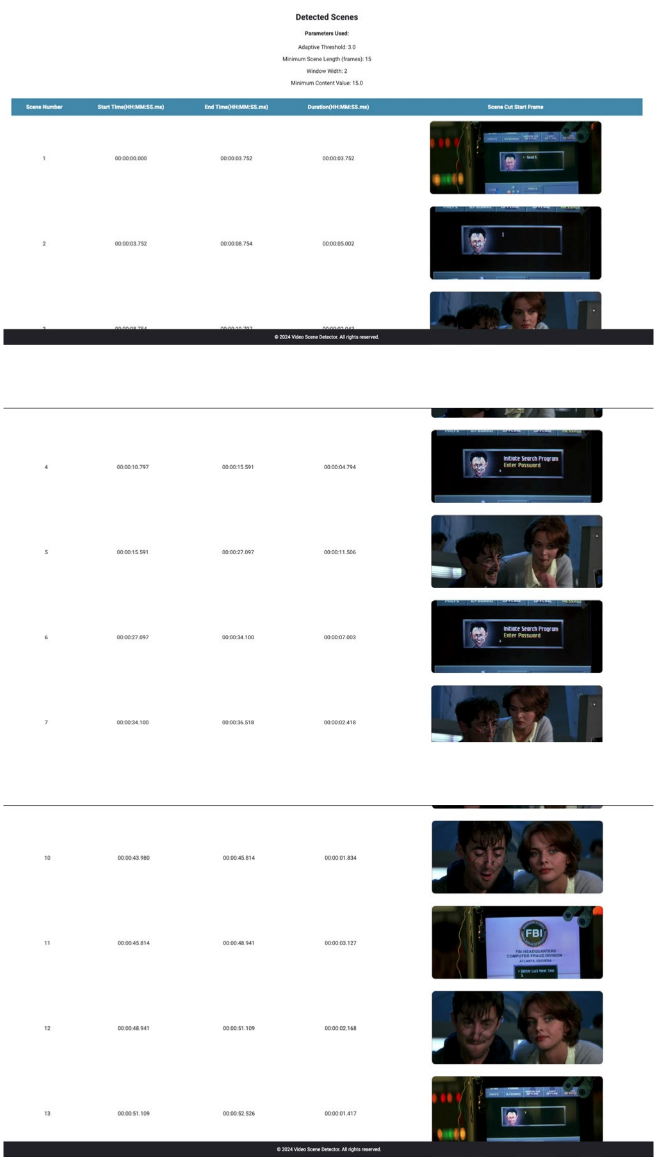
Adaptive Threshold: 3.0

Minimum Scene Length (frames): 15

Window Width: 2

Minimum Content Value: 15.0

C. Results



Detected Scenes

Parameters Used:

Adaptive Threshold: 3.0

Minimum Scene Length (frames): 15

Window Width: 2

Minimum Content Value: 15.0

Scene Number	Start Time(HH:MM:SS.ms)	End Time(HH:MM:SS.ms)	Duration(HH:MM:SS.ms)	Scene Cut Start Frame
1	00:00:00.000	00:00:03.752	00:00:03.752	
2	00:00:03.752	00:00:08.754	00:00:05.002	
3	00:00:08.754	00:00:10.797	00:00:02.043	
4	00:00:10.797	00:00:15.591	00:00:04.794	
5	00:00:15.591	00:00:27.097	00:00:11.506	
6	00:00:27.097	00:00:34.100	00:00:07.003	
7	00:00:34.100	00:00:36.518	00:00:02.418	
10	00:00:43.980	00:00:45.814	00:00:01.834	
11	00:00:45.814	00:00:48.941	00:00:03.127	
12	00:00:48.941	00:00:51.109	00:00:02.168	
13	00:00:51.109	00:00:52.526	00:00:01.417	

VIII. DISCUSSION

The project successfully addresses the need for automated scene detection and analysis in video content. The adaptive algorithm, combined with OpenCV's capabilities, allows for accurate detection of scene transitions, even in complex videos with varying lighting and content. The user-friendly interface and additional features, such as scene splitting and thumbnail generation, enhance the overall experience and utility of the application.

However, there are some areas for improvement and further development:

1. **Performance Optimization**: Processing large video files can be time-consuming. Future

iterations could focus on optimizing the algorithm for speed and efficiency.

2. **Additional Features**: The application could be expanded to include more advanced scene analysis features, such as detecting specific events or activities within scenes.

3. **Integration with Other Platforms**: Integrating the application with popular video streaming platforms could enhance its utility for content creators and streamers.

IX. Conclusion

The Video Scene Detector project provides a comprehensive solution for automated scene detection and analysis in video content. By combining Flask, OpenCV, and Docker, the application offers a user-friendly interface and accurate scene detection capabilities. The project addresses a wide range of use cases, from organizing home videos to analysing surveillance footage and creating engaging streaming content.

The application successfully meets its objectives, offering a versatile tool for video content management and analysis. Future improvements could focus on performance optimization and additional features, further enhancing the application's utility across various industries and applications.

.

References

[1] Flask Documentation. (n.d.). [Online]. Available: https://flask.palletsprojects.com/

[2] OpenCV Documentation. (n.d.). [Online]. Available: https://docs.opencv.org/4.x/index.html

[3] Gruzman, I. S., & Kostenkova, A. S. (2014, October). Algorithm of scene change detection in a video sequence based on the threedimensional histogram of color images. In *2014 12th International Conference on Actual Problems of Electronics Instrument Engineering (APEIE)* (pp. 1-1). IEEE.

[4] Reddy, B., & Jadhav, A. (2015, February). Comparison of scene change detection algorithms for videos. In *2015 Fifth International Conference on Advanced Computing & Communication Technologies* (pp. 84-89). IEEE.