Zisan Ahmed

Undergraduate Student

American International University Bangladesh

Video Content Matching Report

The objective of this report is to document the process and outcomes of a Python program designed to analyze the visual and audio content similarity between two videos. The videos under consideration are named main.mp4 and cut-part.mp4.

The primary aim of this analysis is to develop an understanding of the degree of similarity between the videos, considering both their visual and audio aspects. The program utilizes fundamental techniques of video processing and audio analysis to calculate similarity scores, which are then used to quantify the resemblance between the videos.

The Methodology section outlines the steps undertaken to achieve the task, including video loading, frame and audio extraction, and similarity score calculation. In the Results section, the obtained similarity percentages are presented along with a discussion of the implications of these findings.

Now proceed to the Methodology section to gain an understanding of the technical aspects that drove this analysis.

Methodology

To achieve the task of assessing content similarity between the "main.mp4" and "cut-part.mp4" videos, the following steps were carried out:

(Anaconda and Jupyter notebook have been used)

1. Video Loading: The videos were loaded into the program using the `moviepy` library, which facilitated easy access to their visual and audio components.

2. Frame and Audio Extraction: The frames of both videos were extracted and converted to grayscale to facilitate the Structural Similarity Index (SSIM) calculation. The audio signals were also extracted for audio similarity analysis.

3. Visual Similarity Calculation: The SSIM metric from the `skimage.metrics` library was employed to calculate the structural similarity between corresponding frames of the videos.

4. Audio Similarity Calculation: Root Mean Square Error (RMSE) was used to quantify the differences between the audio signals of the two videos.

5. Overall Similarity Percentage: The visual and audio similarity scores were combined to compute an overall similarity percentage.

Results

The results of the analysis are as follows:

Visual Similarity: 9.75%

Audio Similarity: 85.31%

Overall Similarity: 47.53%

The observed similarity percentages, especially in the context of "cut-part.mp4" being a portion of "main.mp4

Conclusion

Through this analysis, i have gained valuable insights into the content similarity between the provided videos. The developed Python program successfully extracted relevant features and provided a quantitative measure of similarity. However, i tried tensorflow, pytoch, pydub, keras, and other libraries for advanced analysis, but I failed since I couldn't deal with the errors.

Code

Video Content Matching

Import Required Libraries

In [17]:
```
import moviepy.editor as mp
import imageio
import numpy as np
from skimage.metrics import structural_similarity as compare_ssim
```

This section imports the required Python libraries that will be used for video and image processing. The moviepy.editor library is used to work with video files, imageio for image manipulation, numpy for numerical operations, and structural_similarity from skimage.metrics for calculating SSIM.

Load and Process Videos

In [285]:
```
# Load videos
main_video = mp.VideoFileClip("main.mp4")
cut_video = mp.VideoFileClip("cut-part.mp4")

# Convert videos to numpy arrays (frames)
main_frames = np.array(list(main_video.iter_frames()))
cut_frames = np.array(list(cut_video.iter_frames()))

# Convert frames to grayscale for SSIM comparison
main_gray_frames = [imageio.core.util.Array(main_frame[:, :, 0]) for main_frame
cut_gray_frames = [imageio.core.util.Array(cut_frame[:, :, 0]) for cut_frame in
```

These lines load the videos "main.mp4" and "cut-part.mp4" using the VideoFileClip class from the moviepy.editor library. This step prepares the videos for subsequent analysis.

the loaded videos are converted into arrays of frames. The iter_frames() function is used to iterate through each frame of the videos and store them as NumPy arrays.

Then the frames converted to grayscale, which is a requirement for using the SSIM metric.The frames are iterated, and each color frame is converted to grayscale by selecting the first channel (red channel) of the frame.

Calculate Visual Similarity (SSIM)

In [286]:
```
ssim_scores = []
for main_gray_frame, cut_gray_frame in zip(main_gray_frames, cut_gray_frames):
    ssim_score = compare_ssim(main_gray_frame, cut_gray_frame)
    ssim_scores.append(ssim_score)

# Calculate average SSIM score
average_ssim_score = np.mean(ssim_scores)
visual_similarity_percentage = average_ssim_score * 100
print(f"Visual Similarity: {visual_similarity_percentage:.2f}%")
```

```
Visual Similarity: 9.75%
```

In this loop, SSIM scores are calculated for each pair of corresponding frames from the main and cut videos. The SSIM metric, obtained from the skimage.metrics library, measures structural similarity between two images. The calculated scores are stored in the ssim_scores list.The average SSIM score is calculated by taking the mean of all the individual SSIM scores. This score represents the visual similarity between the frames of the videos. It's converted to a percentage for easier interpretation.

Calculate Audio Similarity

In [287]:
```
from scipy.signal import resample

# Resample audio to match the shorter length
if main_audio.shape[0] > cut_audio.shape[0]:
    main_audio_resampled = resample(main_audio, cut_audio.shape[0], axis=0)
    cut_audio_resampled = cut_audio
else:
    main_audio_resampled = main_audio
    cut_audio_resampled = resample(cut_audio, main_audio.shape[0], axis=0)

# Calculate RMSE between resampled audio signals
rmse = np.sqrt(np.mean((main_audio_resampled - cut_audio_resampled)**2))
audio_similarity_percentage = 100 * (1 - rmse / np.max(main_audio_resampled))
print(f"Audio Similarity: {audio_similarity_percentage:.2f}%")
```

```
Audio Similarity: 85.31%
```

The scipy.signal.resample function is imported to assist in resampling the audio signals. This step ensures that both audio signals have the same length for accurate comparison. The code checks the lengths of the main and cut audio signals. If the main audio is longer, it is resampled to match the length of the cut audio, and vice versa.

Root Mean Square Error (RMSE) is calculated between the resampled audio signals of the main and cut videos. RMSE quantifies the differences between the two audio signals. A lower RMSE indicates higher similarity. The RMSE value is converted to a percentage to provide an intuitive measure of audio similarity.

Calculate Overall Similarity Percentage

```
In [288]: overall_similarity_percentage = (visual_similarity_percentage + audio_similarity
          print(f"Overall Similarity: {overall_similarity_percentage:.2f}%")
```

```
Overall Similarity: 47.53%
```

The overall similarity percentage is calculated as the average of the visual similarity percentage and the audio similarity percentage. This provides a comprehensive measure of how similar the two videos are in both visual and audio aspects.

```
In [289]: print("Results:")
          print(f"Visual Similarity: {visual_similarity_percentage:.2f}%")
          print(f"Audio Similarity: {audio_similarity_percentage:.2f}%")
          print(f"Overall Similarity: {overall_similarity_percentage:.2f}%")
```

```
Results:
Visual Similarity: 9.75%
Audio Similarity: 85.31%
Overall Similarity: 47.53%
```

Finally, the calculated similarity percentages are printed out in a formatted manner. This provides a clear summary of the visual similarity, audio similarity, and overall similarity between the videos.

This code implements the entire content similarity analysis process, including loading videos, analyzing frames, calculating visual and audio similarity, and presenting the results. It provides a comprehensive view of how the Python program assesses the content similarity between "main.mp4" and "cut-part.mp4".