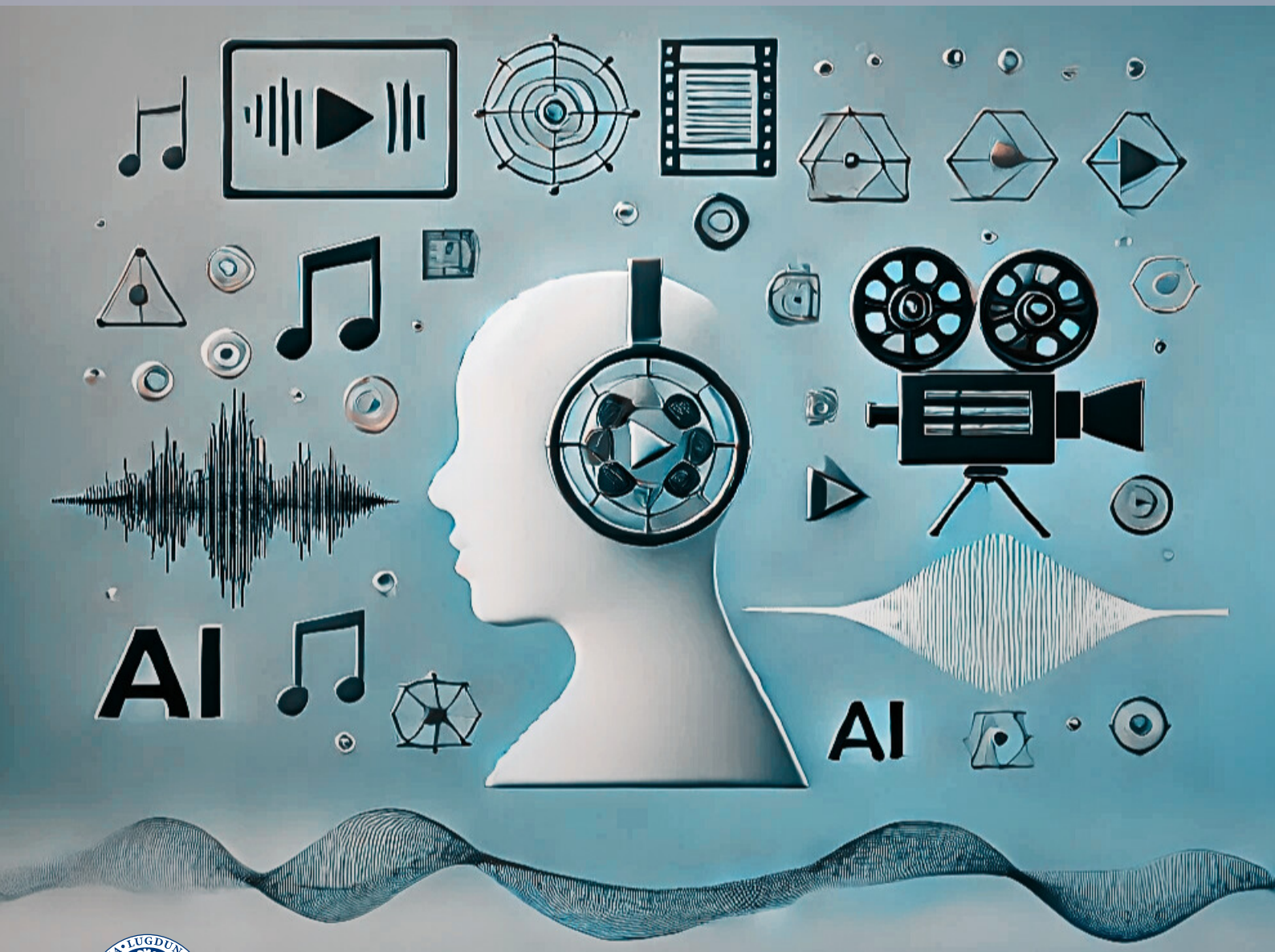
Automating Scene Change Detection in TV and Film

A Deep Multimodal Fusion Approach

MSc Computer Science Thesis

Mark Rademaker



Universiteit Leiden
Observatory

Universiteit Leiden

Computer Science (MSc)

Automating Scene Change Detection in TV and Film: A Deep Multimodal Fusion Approach

Name:	Mark Rademaker
Student ID:	s3382001
Date:	02/01/2025
Specialisation:	Artificial Intelligence
1st supervisor:	Dr. A.J. Knobbe
2nd supervisor:	Dr. M.J. van Duijn

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Artificial intelligence (AI) and machine learning advancements are transforming labor-intensive tasks in TV and film production, such as scene change detection. This study explores the development of a supervised multimodal fusion model to identify scene changes in raw audiovisual content automatically. The proposed approach combines video and audio modalities to detect transitions based on semantic elements, such as changes in location, characters, and narrative context. By leveraging pretraining on a large dataset and fine-tuning for show-specific characteristics, the model adapts to domain-specific challenges.

We evaluate our pipeline through seven research questions addressing audio source separation, shot detection algorithms, feature-based learning, recurrent versus non-recurrent models, multimodal fusion, hyperparameter optimization, and out-of-domain adaptability. Key findings include the effectiveness of using separated audio stems and embedding models and the superior performance of a recurrent architecture with bidirectional layers for capturing temporal dependencies. The multimodal fusion approach consistently outperforms single-modality models, emphasizing the importance of integrating diverse features.

Although our results highlight the promise of this approach, challenges remain due to the inherent subjectivity of scene annotations and the variability of show formats. The proposed framework demonstrates the potential for reducing annotation time and enhancing semantic understanding, particularly in a human-in-the-loop workflow. This research represents a step toward fully autonomous scene change detection while offering practical insights for improving content optimization in TV and film production.

Contents

1 Introduction

Artificial intelligence and machine learning advancements are revolutionizing labor-intensive tasks in TV production, including annotation, which previously required significant manual effort [1]. AI techniques can now quickly analyze large amounts of audiovisual data and potentially identify patterns that human annotators might miss. Additionally, AI-based annotation systems can offer more consistency, relying on predetermined weights to predict outcomes. In contrast, various factors may influence human annotators, making annotation a task requiring multiple people to ensure accuracy and objectivity.
Machine learning algorithms are already used to automate various aspects of TV production. For example, AI systems can automatically tag metadata, identify objects, or create clips from live events—tasks that were once time-consuming and prone to human error [10]. By automating repetitive tasks, AI enables production teams to focus on more creative and high-value work, such as refining content or enhancing creative elements based on the ever-growing volume of data [10] [13]. As predictive analytics and machine learning tools evolve, content creation can become more data-driven. The more features we extract from content accurately and gather user feedback through clicks, the more accessible analytical insights on what appeals to consumers are to creators. Combining AI techniques and insights into consumer preferences opens up new possibilities in TV production. AI not only assists but also challenges traditional notions of creativity by performing tasks previously considered too complex for automation.
AI still faces challenges in automating complex tasks like creating a model capturing *semantics* in audiovisual content. In movies, semantics refers to how meaning is explained through visual and auditory elements, such as dialogue, symbols, camera angles, lighting, and music, and how these elements interact to create more profound interpretations and themes within the narrative. Learning how the semantics contributed to the narrative allows us to build an AI model that can understand more complex tasks, such as *scene change detection*, *sequence detection*, and *segment detection*. Another task within the hierarchical structure of components in television is *shot detection*. Shots represent parts of content captured from a single camera angle. Unlike other tasks in this domain, shot detection stands out because it does not require an understanding of high-level features; instead, it focuses solely on identifying changes in camera angles.
In the media field, a scene can be understood as the smallest unit where a specific event or action occurs, often in a single location and timeframe. Detecting scenes involves identifying transitions between these narrative units based on changes in content like lighting, location, or character interaction. A sequence is a collection of connected scenes that develop a more significant portion of the narrative. Sequence detection involves recognizing how several scenes are related by theme or ongoing action. Finally, a segment is a broader portion of the audiovisual content, such as acts in a film or different sections of a TV program. Segment detection can help break down content into larger thematic or structural units, allowing for better categorization and analysis.
Most scenes, sequences, and segments play a role in developing an enjoyable narrative structure that is thought out from beginning to end. A scene will mostly consist of these five elements [12]:

1. An event, a scene is a pivotal unit of storytelling that typically unfolds within a single setting and timeframe. It should contain at least one story event contributing to the semantics of the storyline.

2. A change, the key characteristic of a scene is its transformative nature. A character's beliefs, feelings, or actions often change significantly from the beginning to the end.

3. One time, most scenes occur within a relatively short timeframe, usually a few minutes or hours.

4. One setting, while films typically follow a single setting per scene, there are exceptions. For instance, cross cuts or cutaways, where the narrative may shift between different settings. However, the focus quickly returns to the main scene's location. In those instances, we need to rely on the reoccurring setting also depicted in Fig. 1

Since segments and sequences are constructed from individual scenes, creating a high-quality scene change detection model is an important first step in creating models with semantic understanding.

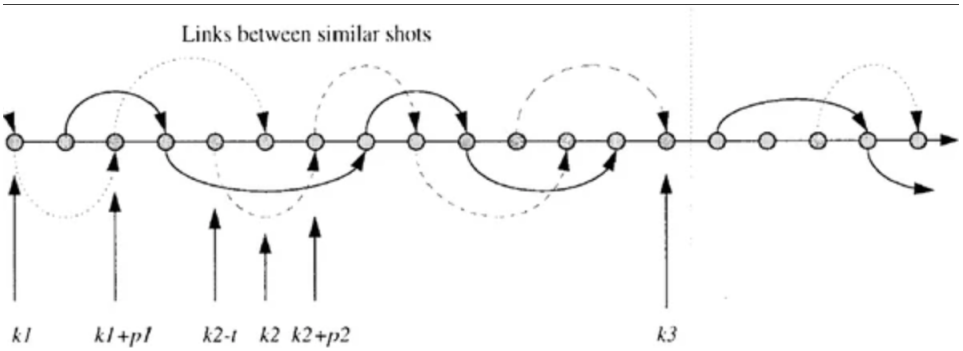


Figure 1: Reoccurring frames within scenes [19], with *k3* being the scene boundary as there is no frame reoccurring after *k3*

In our view, scene change detection involves understanding the film semantics, the study of meaning in film, which remains challenging to automate fully but, if done correctly, can build a far more intelligent AI model. The applied AI technique should be able to understand and extract information from multiple data sources or content modalities. In this domain, a relatively simple model could learn scene change cues and detect some scene transitions. However, we argue that relying solely on such an approach, without considering the essential contributing factors that define a scene, would limit the long-term potential of a detection model. Therefore, we emphasize the importance of adopting a more advanced, high-level feature model. Fig. 2 shows the different modalities (Video, Text, Audio) needed for some challenging film/TV annotation tasks. Large tech companies like Netflix and Amazon are working on training AI models to understand movie semantics for tasks such as scene change detection [15][6]. However, these companies often prioritize keeping their innovations and proprietary data internal for commercial reasons. This makes the development of optimized scene change detection less of a collaborative effort for science and more of a competitive race to dominate the field.

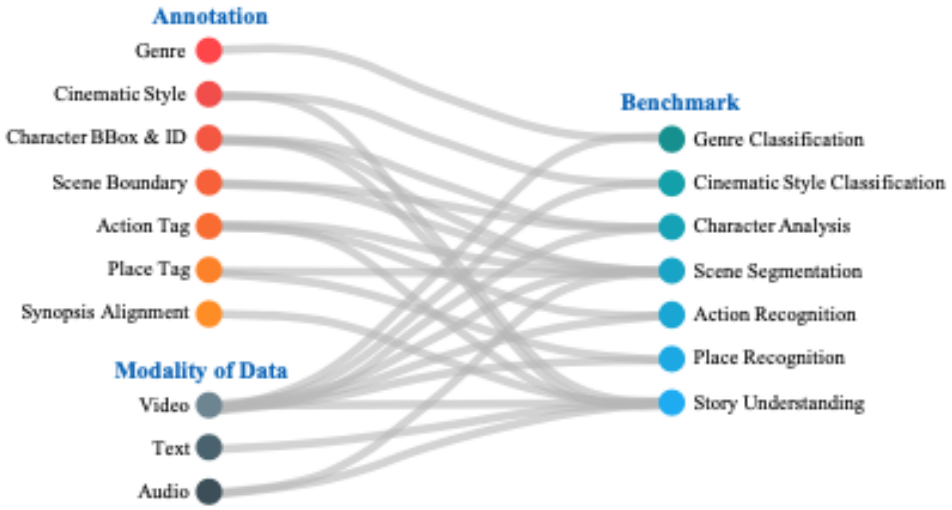


Figure 2: Overview of the combinations of features and modalities that are relevant for Audiovisual analyses tasks [27]

In this paper, we try to contribute to content optimization for TV shows by building a sequence detection model using a multimodal fusion approach. As both audio and video modalities contain information on scene transitions, we fuse both information streams in a pipeline to make a final prediction. Video and audio capture most of the essential features within and between scenes so that we will reserve text analysis for future research. There are two options within classification tasks: supervised and unsupervised. Unsupervised learning does not require labeled data, while supervised learning, though data-dependent, can excel at more complex prediction tasks. To build a model in a supervised way that can effectively learn such narrative structures, we need two things: a lot of data and a model. We first gather the data from a large annotated dataset [8] containing 32K movie clips. The data is then fed into a pipeline, a series of steps transforming the data to make good predictions. This pipeline includes separating an entire sequence into shots, then dividing the audio and video

modalities, splitting the audio into vocals and accompaniment using *Spleeter* [26], embedding both audio and video modalities, and finally feeding these embedded modalities into separate recurrent models. The outputs are fused and passed through the final layers, called the *fusion layers*, which 'fuse' the different modalities into one data stream.
The features that define a scene vary across different types of shows. For example, talk shows are structured differently than reality TV shows. Each program has developed a unique narrative structure, which presents challenges in detecting scene cuts. The Oxford definition of a scene is *"part of a film, play, or book in which the action happens in one place or is of one particular type."* This definition highlights the importance of identifying features such as locations in a detection model. However, creating a universal model for various show types isn't straightforward because the simple rule of identifying location changes does not always apply. For instance, a talk show typically occurs in a single location, but considering an entire show as one scene will not provide helpful information. Additionally, a telephone conversation in TV shows may occur across two locations, contributing to a single scene. However, the back-and-forth transitions will not provide creators with helpful information if treated as two separate scenes. To create a universally applicable scene change detection model, it needs to learn narrative structures of various show types and be adaptive to understand what constitutes a scene in different domains instead of blindly following predefined rules. The model must utilize the semantic elements contributing to the narrative to know where changes and events occur.
We argue that developing a universal supervised scene-detection model is very challenging due to the subjectivity of annotations per type of content and the limited availability of data across different domains. Therefore, when pre-training a model, we focus on capturing key concepts indicative of scene changes. The pre-trained model will learn to recognize how different elements like dialogue, setting, and actions come together to create a distinct narrative unit. This understanding of in-domain (film) scene changes can be transferred to learn out-of-domain scenes without needing an extensive domain-specific dataset. In this research, we will approach the problem in these two phases: initially, we will pre-train a scene change detection model on a large dataset, and subsequently, we will fine-tune the pre-trained model using show-specific data. The model can effectively identify scenes as key building blocks of the overall narrative by understanding the learned elements and their role in advancing the plot. The fine-tuned model is designed to learn the rules of show-specific scene changes, such as theme songs, audience noise, visual text, and camera angles.

We propose the central research question: *"Can scene changes be effectively captured automatically from raw TV content, using a deep multimodal fusion learning approach?"*
To address this, we break it down into the following subquestions:

1. *Q1: Can performance be improved by separating the audio modality into two stems, vocals and accompaniment, using Spleeter?*
2. *Q2: How effectively do shot boundary detection algorithms detect shot transitions within scenes?*
3. *Q3: How can we utilize established embedding models to shift from end-to-end learning models to a feature-based learning framework?*
4. *Q4: Which model, recurrent or non-recurrent, can best be used for feature-based learning to predict scene changes?*
5. *Q5: Does a multimodal fusion approach enhance performance compared to using audio and video separately?*
6. *Q6: What are the optimal hyperparameters for maximizing the performance of this model on the dataset?*
7. *Q7: Can a model learn out-of-domain scene changes (with a different interpretation of a scene) from little data?*

Answering these questions provides valuable insights for developing an intelligent detection model. While our research focuses on scene change detection, some of our findings can also benefit other analytical models in film. For example, genre classification or action recognition models often rely on multiple modalities, suggesting that the steps outlined in this approach may also be applicable to those domains. The structure of this paper is as follows: we begin with a literature review in Sect. 2, followed by a description of the data in Sect. 3. Next, we outline the methods for the model and steps used in our pipeline in Sect. 4. After describing the methods, we cover how the experiment has been set up to test our research questions (Sect. 5). The results are presented in Sect. 6. We conclude with the conclusion in Sect. 8 and a discussion in Sect. 7. Lastly, we present some suggested enhancements that could improve our research in Sect. 9

2 Background and Literature Review

The study of video content analysis, particularly scene and shot detection, has seen advancements since the early 1990s. Early efforts concentrated primarily on detecting shot transitions, which provided a foundation for later, more complex techniques aimed at scene segmentation. Early work was done in this field focused mainly on visual aspects, such as frame differences and the disruption of motion within content [24][34][4][50][2]. Building on these foundations, paper [22] introduced an optical flow-based model enhanced with linear prediction for detecting shot cuts. Flow-based models assess whether one shot naturally follows the next, with a disruption in this flow potentially indicating a shot change. Their work emphasized the need to distinguish between shot cuts, which involve individual shot transitions, and scene cuts, which reflect more significant narrative shifts in the video.

This foundational work is crucial as scene change detection essentially builds upon the techniques and insights gained from shot detection, demonstrating the evolution and interdependence of these methodologies in video processing [5]. The paper [30] introduced one of the earliest algorithms for video segmentation. It identified potential scene boundaries by measuring short-term coherence between consecutive shots, focusing on similarities in color. Shortly followed by [49], who developed the graphical representation of video data by creating a Scene Transition Graph (STG). In the STG, each node symbolizes a shot, while the edges depict the transitions between these shots. The STG is subsequently divided into multiple subgraphs using the complete-link hierarchical clustering method. This division ensures that each subgraph, or scene, adheres to a specific color similarity constraint, effectively grouping shots that share visual characteristics.

[19] works out a similar link-based method. The idea is depicted in Fig. 1 and further used in this paper to assess the quality of our processed data in Sect. 6. These earlier models cannot learn the complex semantics of content and largely focus on single elements, such as the similarity of frames or motion models.

All these papers utilize unsupervised models, which is unsurprising given that the large-scale collection of digital movie data only began to surge in the late 2010s as streaming platforms and digital media replaced DVDs. This shift made vast amounts of video content more accessible for analysis, paving the way for developing more sophisticated data-driven approaches to scene and shot detection. We argue that supervised learning is the most effective way to capture the complexity of scene transitions. This is also shown in paper [28], where they compare the performance of multiple supervised models with those of unsupervised models for scene change detection. The supervised approach outperforms the unsupervised ones.

Additionally, paper [19] outlines the various methodologies commonly employed for classifying scene transitions in the earlier years, categorizing them into three distinct types: rule-based methods, graph-based methods, and stochastic methods.

These methods are built on fundamental concepts for constructing a scene transition model. First, there are established rules that define what constitutes a scene. For instance, many film experts agree that a scene change typically involves changing characters. While these rules do not always need to hold, the rule can be used as an indicative factor. Second, shots with a higher degree of similarity are more likely than those not to be part of the same scene. This core principle underlies the graph-based method, which connects similar shots and identifies scene transitions by analyzing the structure of these relationships. While we argue that pixel similarity does not contribute to developing semantic understanding, it plays a significant role in determining location. Lastly, stochastic methods leverage the uncertainty and variability in scene transitions through probabilistic models. As previously noted, rules don't always need to be followed strictly but serve as change indicators, justifying a probabilistic approach. Rather than relying on predefined rules or direct similarities between shots, these methods learn patterns from the data, estimating the likelihood of a transition based on historical observations. A probabilistic recurrent deep learning model could learn features or rules directly from data while capturing temporal dependencies and incorporating probabilistic elements to handle uncertainty in scene transitions.

We have justified our supervised approach based on some earlier papers; we look into the latest techniques. In the paper [41], the authors explore the efficacy of early and late fusion methods in semantic video analysis. They define early fusion as integrating features from multiple modalities at the initial stages of feature extraction, which are then used to train a model. Late fusion, however, involves training separate models for each modality and then combining the outputs toward the end of the process. Their findings suggest that late fusion performs slightly better than early fusion across most concepts. However, in instances where early fusion outperforms late fusion, the performance gains are more pronounced. In the study [23], the use of *Inception I3D*, a state-of-the-art convolutional network architecture, alongside a large *NTU RGB+D* dataset featuring RGB video, optical flow, and skeleton data, reveals that multimodal fusion, especially through an early fusion strategy leveraging statistical correlations between modalities, significantly enhances human activity recognition compared to uni-

modal approaches. We employ a hybrid fusion approach to combine the advantages of both early and late fusion methods, which will be discussed further in Sect. 4.
Paper [47] creates a model that integrates complex multimodal cues and high-order relationships in a unified structure, enhancing the contextual understanding of video shots for accurate scene boundary detection. High-order relationships are captured through contrastive learning. Learning high-order relations creates semantic understanding, as it needs to identify important aspects of the films that contribute to the narrative. The paper demonstrates the potential of multimodal fusion in audiovisual content analysis.
The mentioned advanced models in this domain typically employ complex network architectures. This complexity is necessary because the models are designed to learn from raw input data without the benefit of pre-existing knowledge. Such architectures enable the models to interpret and understand the raw data automatically. In [3], they leverage scene change detection analysis through audio features such as music, human voice, animal voice, and more. The approach in [3] only uses the audio component to detect scene boundaries. We argue that in some domains, the visual component can be more informative for scene changes than others, such as in BBC's *Planet Earth* [9]. In [48**?**] they have tried to use a multimodal experiment on the *MovieNet*[27] dataset but did not see improvements. They argued that this could be due to incomplete audio data and copyright restrictions on raw audio. Recent studies have also incorporated transformers for using these features from audio and video modalities, as noted by [28]. Paper [28] shows state-of-the art performance with their *TranS4mer* model on the MovieNet [27], BBC [9] and OVSD [37] datasets. However, this paper does not use audio waveforms to input their model. The results in [28] show that the state-of-the-art models are *tranS4rmer*, *ShotCol* [15] and *BaSSL* [32], all supervised, and outperform all unsupervised models by a significant margin. Just recently, [38] presented *Multimodal alignmEnt aGgregation and distillAtion (MEGA)* which showed better performance than both BaSSL, ShotCol on the MovieNet dataset.
In the paper, [8], they evaluate a *Contextual Boost Module (CBM)* for the text-to-video task. This module enhances retrieval performance by integrating contextual information from adjacent video clips, enabling the model to understand and utilize the narrative structure of the video content effectively. The CBM uses additional video clip information to enrich the video embeddings, making them more informative for different tasks by acknowledging movie scenes' sequential and contextual nature. While their approach differs from ours, we can use some of the implications of the results to help improve our approach. Namely, they conclude that: *"Beside improving retrieval, developing richer models to model longer-term temporal context will also allow us to follow the evolution of relationships and higher level semantics in movies, exciting avenues for future work."*. We use the multimodal framework and bidirectional recurrent models discussed in Sect to model longer temporal contexts. 4. To our knowledge, no study has yet fully explored such a framework, including delegating high-level feature extractions to state-of-the-art embedders.

3 Datasets

In this section, we discuss relevant datasets for detecting scene changes. We start by describing some available datasets and their characteristics.
As paper [11] already shows, the majority of the analysis in scene change detection is done on relatively small sets, such as *BBC Planet Earth* [9] and *Open Video Scene Detection (OVSD)* [37]. In our approach, we need an extensive dataset to train a model to learn the complex task of scene change detection, for which we can consider *MovieNet* [27] or *Condensed Movies Dataset (CMD)* [8]. MovieNet and CMD are large open-source datasets containing thousands of movies, facilitating learning various types of audiovisual content. For MovieNet, only the extracted embeddings are open-sourced, and the raw video and audio segments are unavailable. MovieNet consists of extracted audio features using *NaverNet* [18], which limits the potential of using the MovieNet dataset in a multimodal framework, as we are limited to the NaverNet embedder. CMD provides other challenges; more on this we discuss in Sect. 3.1.
We describe the extensive training dataset as *in-domain data*. We use this term as the training covers this domain, in this case, movies. After training, we can fine-tune a model on a domain-specific dataset, or *out-of-domain dataset*. We refer to this as out-of-domain data because the content type may differ from traditional movies, such as the BBC's Planet Earth documentary. This distinction shows the difference in genre and style from the typical cinematic formats usually used in movies. This dataset does not have to be as large as the in-domain dataset, as the model can transfer some of the generic patterns learned for scene changes from the in-domain dataset and then adjust to domain-specific patterns using relatively little data. Sect. 3.2 describes some domain-specific datasets that one can use for fine-tuning a model on a smaller set.

3.1 In-domain Data

The dataset introduced in paper [8], CMD, offers the largest amount of free, open-source audiovisual content of the clips with labeled scenes. The dataset consists of movies from 1930 to 2019, covering a broad historical range from early black-and-white films to the latest animated movies. CMD contains over 33,000 clips from 3,605 movies, each about two minutes long and representing key scenes, reducing each film to around 20 minutes.
The clips maintain the semantic complexity of full-length films while being significantly shorter. The dataset contains the semantic descriptions of high-level semantic concepts for each video clip, covering intent/motivation, relationships, emotions, attributes, and contextual information from surrounding clips in the storyline. This metadata could make this a valuable dataset for various semantic analysis tasks. To ensure the clips contain enough information about the storyline, clips have been manually aligned to Wikipedia plot summaries in [8]. Although the clips accounted for only 15% of the movie's total duration, they covered 44% of the plot, indicating that the clips represent key scenes contributing to the storyline.
Tab. 1 describes the dataset. Since our dataset consists of clips rather than full movies, the original continuous flow between clips is broken. A potential solution to this problem is discussed in Sect. 5.1.

Table 1: In-domain dataset (CMD) description

Statistic	**Description**	**Value**
#Movies	Total movies included	3,605
#Scenes	Total video scenes extracted from movies	33,976
#Hours	Total hours of content	1,270
Average Scene Length	Average duration of one scene	00:02:15
Median Scene Length	Median duration of video scenes	00:02:12
Average Shot Length	Average duration of one shot	3.56 s

This table illustrates the depth and breadth of CMD [8], making the dataset capable of supporting complex scene recognition tasks within a diverse cinematic context. The data's extensive coverage ensures a robust foundation for training models to interpret and analyze visual scenes across various movies accurately.

3.2 Out-of-domain Data

As annotating scenes is a labor-intensive task, most of the datasets are relatively small. Therefore, fine-tuning a model on out-of-domain data instead of fully training a model on this smaller dataset could improve performance and make the model more robust.
OVSD [37] and BBC Planet Earth [9] are commonly used smaller benchmark datasets. Tab. 2 presents descriptive statistics from the BBC *Planet Earth* dataset [9], which includes metadata on the exact number of shots and scene boundaries. This makes the dataset an ideal benchmark for evaluating our shot detection models, as further detailed in Sect. 5.2.2.

Table 2: BBC Planet's Earth dataset description

	# Shot	**# Scene**	**# Video**	**Time (h)**	**Source**
BBC [9]	4,900	670	11	9	Documentary

The BBC dataset [9] is notable for its high-quality video and precise annotations of scene transitions. Five annotators have independently marked the starting shot number for every scene across the entire dataset, which makes the dataset less susceptible to subjective interpretation.
It provides a real-world challenge to scene change detection models due to the professional level of video editing that includes a mix of rapid cuts, gradual transitions, and various audiovisual effects designed to enhance storytelling. This makes it particularly useful for evaluating how well scene and shot-detection algorithms can adapt to different content and editing styles commonly used in professional broadcasting. Tab. 3 contains the statistics per episode, relative to Tab. 1, BBC Planet Earth [9] has shorter scene lengths than CMD [8].

Table 3: BBC Planet Earth Dataset statistics per episode

Video Name	**Time**	**#Shots**	**#Scenes**	**Time/Scene**
From Pole to Pole (01)	49:15	445	46	1:04
Mountains (02)	48:05	383	44	1:05
Ice Worlds (03)	49:17	421	48	1:02
Great Plains (04)	49:03	472	57	0:52
Jungles (05)	49:14	460	54	0:55
Seasonal Forests (06)	49:19	526	52	0:57
Fresh Water (07)	49:17	531	57	0:52
Ocean Deep (08)	49:14	410	46	1:04
Shallow Seas (09)	49:14	366	58	0:51
Caves (10)	48:55	374	53	0:55
Deserts (11)	49:00	467	50	0:59
Total	9:48:53	5322	618	0:57

To test on different domains, we have the available reality-TV dataset of *Survivor* and *Hunted*. The TV shows are popular and televised globally, making their analysis widely applicable. As Tab. 4 shows, episodes, on average, last around one hour with around 20 scenes, which, on average, take two to three minutes.
Being a reality show set in various dynamic locations, Survivor frequently transitions between different environments, each shift typically signaling a new scene. This change in setting is often used to differentiate between different show segments, such as challenges, camp life, and Tribal Council, aligning with scene changes.
Furthermore, Survivor features a diverse cast of contestants who engage in various actions and interactions. This diversity in people and activities contributes to frequent scene changes as the show moves from one event or group interaction to another. While locations change regularly, the settings remain very similar in appearance, typically a deserted island. This similarity in appearance could make scene change detection more challenging than just recognizing location changes. Instead, actions, sounds, and characters become crucial indicators of semantic shifts in the narrative, requiring a more nuanced approach to identify scene transitions effectively.
Hunted provides another perspective within the reality TV genre, focusing on a chase concept that requires diverse locations. The structure of the series differs from that of *Survivor*. While *Survivor* features elements such as camps, games, and voting to eliminate contestants, *Hunted* constantly switches between characters, the hunters and the hunted, resulting in a different structure for scene changes. In Hunted, participants navigate various landscapes—from city streets to rural areas—making the visual context of each scene highly variable. Scene change detection in such a dataset would extend beyond simple location recognition.
In contrast to Survivor, which operates in a more controlled and staged environment, Hunted unfolds in a much more unpredictable setting. The urban sounds and spontaneous actions in its real-world settings influence the narrative, contrasting with Survivor's relatively structured environment. This uncontrolled environment introduces new complexities to analyzing scene changes and narrative flow.
Tab. 4 describes the datasets. The annotation for both datasets was conducted by multiple annotators who collaboratively determined the transition periods.

Table 4: Out-of-domain datasets description

Show	Domain	Location Type	#Episodes	Time/Episode	Scenes/Episode	Time/Scene
Survivor	Reality-Tv	Dynamic	16	01:05:30	22.25	00:02:57
Hunted	Reality-Tv	Dynamic	8	00:54:20	19.50	00:02:47

4 Methodology

This section outlines the research methodology, detailing the steps to prepare and process the data for optimal predictive performance. The preprocessing phase involves transforming raw audio and video content into a suitable format for input into our model. This includes separating audio content into distinct components, referred to as stems, as described in Sect. 4.1. The synchronization and discretization of data, which rely on a shot detection model, are explained in Sect. 4.2.
Sects. 4.3 and 4.4 dive into feature engineering, using embedders, and the design of fusion models using the embeddings as input. These models utilize multimodal layers integrating audio and video features, enabling information fusion from different modalities.

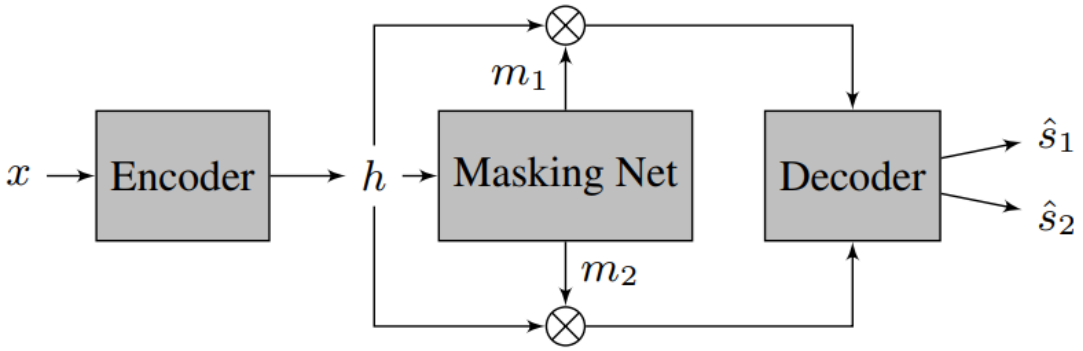


Figure 3: Simple encoder-decoder structure, with masking, for source separation

4.1 Audio Source Separation

To use information from audio effectively, we want to decompose the audio signals into meaningful parts. To do so, we use *Spleeter*. The model, developed by *Deezer* [26], is used for audio source separation. It employs deep learning models to separate a given audio track into two stems: vocals and accompaniment.
The vocals refer to the voices isolated from the rest of the audio, whereas accompaniment includes all other audio elements except the vocals. Typically, this is background music, but it may also be urban sounds. This feature is particularly useful for various tasks, as it isolates mixed components or stems, allowing for a more detailed analysis of each stem.
The models it uses are *U-nets*, as described in paper [29]. A U-net is a Convolutional Neural Network (CNN) type that features an encoder/decoder structure with skip connections. Spleeter utilizes 12-layer U-nets, consisting of six encoder and six decoder layers. An encoder separates a data mixture into latent sources. Subsequently, a decoder independently reconstructs a signal from each latent source. These reconstructed signals are then combined to estimate the original data mixture. Fig. 3 illustrates a simple encoder/decoder framework for audio source separation with masking. These networks are exceptionally trained to soft masks for each audio source (stem). Soft masks are real values typically between 0 and 1. Soft masks allocate varying degrees of signal intensity to different sources.
Spleeter was trained on Deezer's internal datasets. Based on the estimated source spectrograms, the audio sources are separated by soft masking. The goal of source separation is to estimate each source's spectrogram from the mixed spectrogram.
It is initially trained for music information retrieval, allowing for the separation of vocals from background sound in music. However, this method could also be effectively applied to various other tasks. Separating those components into stems could enhance sound interpretation for TV shows and movies that frequently feature character voices and instrumental background music.

4.2 Detection of Shot Boundaries

In this section, we explore the process of detecting shot boundaries, which fulfills two crucial functions.
Firstly, detecting shot boundaries allows scenes to be decomposed into meaningful segments. Multiple shots constitute a scene, making shots the fundamental building blocks upon which a scene is constructed. Understanding where these blocks begin and end is crucial, as the start and end of a scene typically coincide with changes in shots.
Secondly, detecting shot boundaries facilitates the synchronization of different modalities, such as audio and video, referred to as *synchronization*. Additionally, the inherent continuous nature of some input modalities often necessitates discretization, not only for synchronization purposes but also to manage data processing efficiently. Computers cannot handle continuous data without limits; thus, specifications like *frames per second (fps)* for video and *kilohertz (kHz)* for audio are used to describe what are, in essence, always discrete data types. Discretization, or *downsampling*, of the data is beneficial for ensuring that computational resources and

memory are not unnecessarily exhausted.
Audio signals are continuous waveforms, often sampled at rates like 16kHz, while video content consists of sequential frames determined by the fps, each representing a discrete data type. Downsampling audio and video data can be done by selecting the size of these intervals and creating an embedded representation of that time interval. Standard methods include equal-width binning, which divides the data range into intervals of equal length, and equal-frequency binning, which ensures each bin contains roughly the same number of data points. Using shots as bins is neither of those two as a shot may have varying lengths and varying numbers of data points. A pre-trained model must be introduced since it's impossible to bin sequences into shots based on predefined rules like equal-width or equal-frequency binning.
A popular method to detect shot boundaries is *PySceneDetect* [14]. PySceneDetect analyzes differences between adjacent video frames using various methods, such as content changes, histogram differences, or perceptual hashing and triggers a scene cut when the difference exceeds a set threshold. PySceneDetect has three options to detect shot boundaries.

1. *Threshold*, this method relies on a fixed threshold to identify significant changes in pixel intensity or color between frames. A shot boundary is detected when the intensity or color difference exceeds the threshold. It can detect slow transitions.

2. *content*, This method measures changes in content between consecutive frames by analyzing variations in the HSV color space and comparing them to a predefined threshold. When this threshold is exceeded, a potential scene cut is identified.

3. *Adaptive*, instead of using a fixed threshold to detect changes, employs a rolling average of changes between adjacent frames. This adaptive approach helps reduce false detections, particularly in rapid camera movement scenarios, by dynamically adjusting the threshold based on recent frame-to-frame variations.

The methods employed by PySceneDetect for detecting scene changes primarily utilize pixel intensity as their metric for identifying transitions between shots. This approach can pose specific challenges, especially when analyzing black-and-white content. In such content, the pixel intensity range can be limited primarily to shades of gray, reducing the contrast differences these methods rely on to detect scene changes effectively. Given this limitation, its performance can be significantly hindered when PySceneDetect is applied to black-and-white films.

4.3 Feature Extraction

The first step in processing the output data from the previous processing steps into more informative input data for our model involves transformation into useful embeddings. Embeddings are specialized representations that map the input data into an embedding space where similar data points are positioned closely. We will look into embedders for both the audio and video modalities.
In our framework, once the content is split into shots, we can input the data into a pre-trained model and extract embeddings from intermediate layers. These layers capture complex patterns and relationships within the data, providing richer representations than the original input. Models like Wav2Vec for audio and VideoMAE for video excel at capturing these local dependencies, enabling the extraction of meaningful features for downstream tasks.

4.3.1 Video Embedder

For video embedding extraction, we use *Video Masked Autoencoder (VideoMAE)* [43]: A recent adaptation of the masked autoencoder concept for videos. VideoMAE uses a transformer-based architecture that learns efficient video representations by reconstructing masked frames. In VideoMAE, a frame is split into blocks of 16X16, and each block gets a token embedding, then a significant portion of these blocks is masked, using *tube masking*. In tube masking, the same mask is applied across the temporal dimension of the video. This method ensures that spatial and temporal information is hidden, forcing the model to learn features that account for both dimensions. This typically involves a high masking rate ranging from around 70%-90% of the pixels in each frame, which forces the model to focus on reconstructing the missing information. This makes the model able to learn patterns using relatively little available information. Fig. 4 shows how frames in multiple dimensions are masked using tube masking. The power behind these two models can be attributed

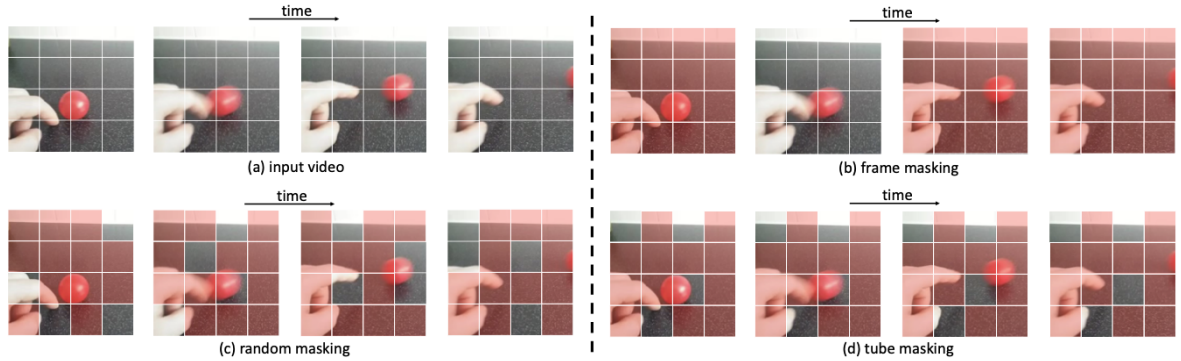


Figure 4: Tube masking hides the same blocks for different frames

to their attention mechanism [45]. The attention mechanisms in these models help to focus on different parts of the input sequence, which are essential for capturing the relevant features needed for effective embedding. After the masking process, only the unmasked tokens are fed into the transformer encoder. A shallow decoder is utilized, layering over the encoder's output and learnable mask tokens to reconstruct the video best. The encoder produces a latent representation that captures the essential information, allowing the decoder to regenerate the missing content from this compressed form. This efficient representation effectively acts as a robust embedding or vector representation of the video. Fig. 5 shows the entire process, including tube masking, encoding, and decoding the video.
By learning to reconstruct masked video frames, VideoMAE develops a nuanced understanding of the video content, including object appearances, interactions, and temporal changes. These learned features can include crucial information about different content within a video. This is essential for detecting when a scene changes, as these changes often involve shifts in the dynamics of objects, people, or settings. The rich embeddings produced by VideoMAE can be used to identify elements within shots. Similarly to acoustic features, visual features combined with elements from different time frames can contribute to a deeper semantic understanding of a shot.

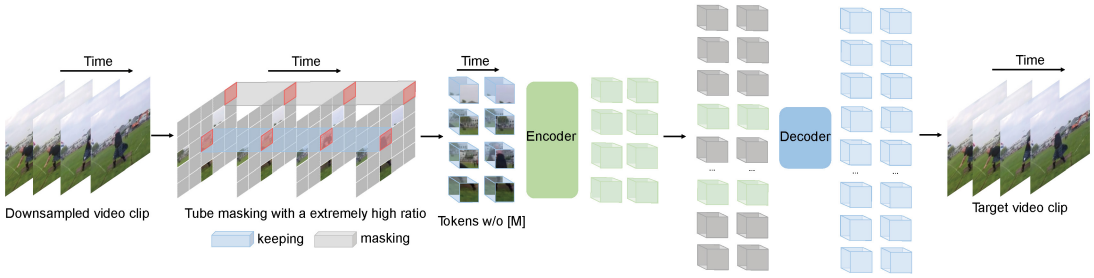


Figure 5: VideoMAE model design

4.3.2 Audio Embedder

For the audio input data case, we aim to derive embeddings from raw audio data, or waveforms, to leverage the recognition capabilities of the established model for our task. *Wav2Vec 2.0* [7], a more advanced version of the original Wav2Vec model [39], is one commonly used model for audio recognition tasks. This model is pre-trained on extensive amounts of unlabeled audio data, such as the LibriSpeech dataset, including English audiobook readings.
Wav2Vec 2.0 employs a self-supervised learning methodology. It processes input through CNN layers, masks parts of the latent audio representations, similarly to VideoMAE, passes the data to a transformer encoder, and learns to predict the masked parts. By predicting masked segments, it trains to reconstruct audio from incomplete sounds, thereby learning patterns in sound.
Fig. 6 shows how raw waveforms X are transformed into context representations C. The model processes raw audio input X and generates latent speech representations $z_1, z_2, z_3, \ldots, z_T$ for T time-steps using a CNN. The output from the feature encoder is quantized into q_t using a quantization module $Z \rightarrow Q$. Quantization reduces

the complexity of audio data by mapping continuous features to smaller, discrete values. These representations are then masked and passed to Transformer blocks, $g : Z \rightarrow C$, which construct contextual representations $c_1, c_2, c_3, \ldots, c_t$, using the attention mechanism [45] to capture information from the entire sequence. The contrastive loss ensures the model can learn embeddings for self-supervised learning by minimizing this loss. The model is trained to correctly identify the true quantized representation of a masked segment from a set of candidates (including distractors, which are incorrect representations). Through learning quality representations of the waveforms, Wav2Vec 2.0 effectively allows for meaningful embeddings.

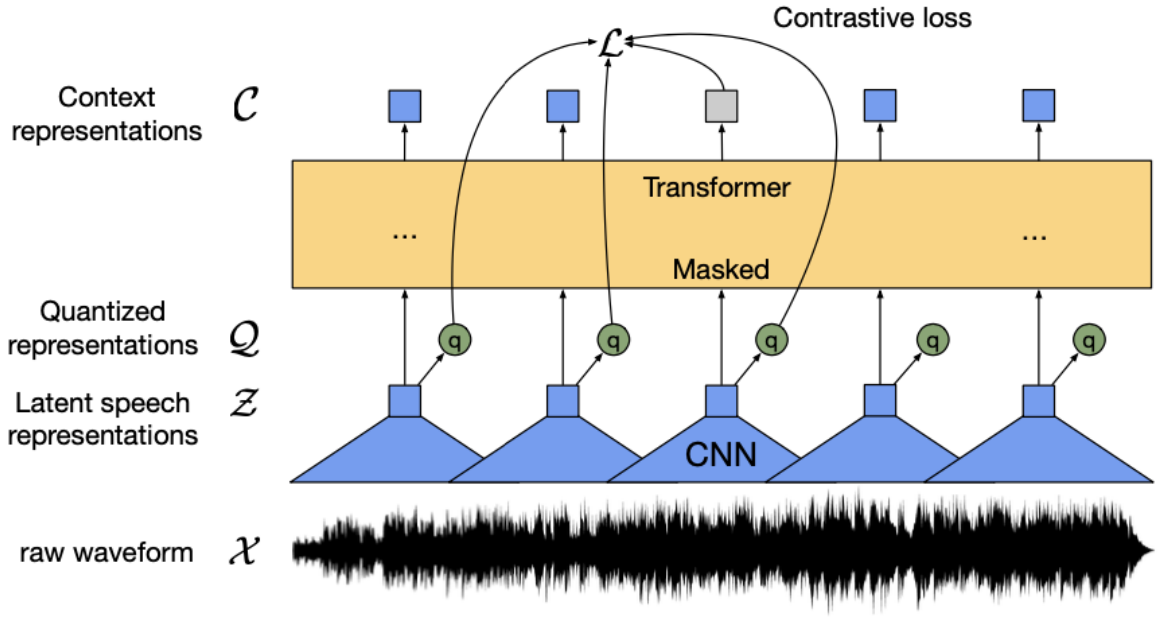


Figure 6: Wav2Vec embedder features CNN layers and Transformer layer encoder

4.3.3 Embedding Visualization Technique

Sects. 4.3.1 and 4.3.2 cover capturing data in features. However, evaluating the quality of the embeddings is often limited to the task they are tested on. For instance, using VideoMAE on object detection tasks can serve to assess and interpret its performance in object detection. However, this does not give quality insights into its location or face recognition performance. For testing embeddings containing multiple features, a more general approach is to use a dimensionality reduction model and visualize the embeddings in a two-dimensional space. Embeddings can then be evaluated based on their relative positions with other embeddings and see why they are positioned closely and if that makes sense to your overall task. As embeddings are often high-dimensional and hard to interpret, it is essential to visualize the relative vectors. *t-Distributed Stochastic Neighbor Embedding (t-SNE)* [44] is a robust machine learning algorithm for dimensionality reduction, particularly suited for the visualization of high-dimensional datasets. t-SNE transforms data from a high-dimensional space to a low-dimensional space by converting similarities between data points to joint probabilities and then minimizing the Kullback-Leibler divergence between these probabilities in the high-dimensional and low-dimensional spaces.

4.3.4 Embedding Post-Processing

Lastly, output embeddings of embedders such as Wav2Vec and VideoMAE have a vector size dependent on the time interval of the data they cover. This means that every embedded shot can have a different size. This can cause further problems in processing these embeddings, as neural network layers typically have a fixed input size. Therefore, post-processing steps must ensure that such embedders' output can be used in subsequent layers.

Given an input sequence $X = \{x_1, x_2, \ldots, x_n\}$, where each x_i represents a frame or shot of the data (e.g., audio waveform or video frame), we use a pre-trained model f (such as Wav2Vec or VideoMAE) to transform each x_i into an embedding e_i in an embedding space:

$$e_i = f(x_i) \tag{1}$$

The embeddings, output from equation 1 $E = \{e_1, e_2, \ldots, e_n\}$ form a sequence where each $e_i \in \mathbb{R}^d$, with d being the dimensionality of the embedding space. These embeddings' dimensionality is determined by the shot length the embedding represents. After embedding the data, one can reduce the size to decrease memory usage and computational load using pooling layers. Additionally, this ensures a fixed size for the input across different embedded sequences or shots. To achieve this, adaptive pooling layers can downsample the embeddings to a consistent, fixed size, eliminating the need for padding to the longest shot. Adaptive pooling maps the varying input of size (H_{in}, W_{in}) to an output size (H_{out}, W_{out}). The stride and kernel size are automatically determined by the pooling layer as follows:

$$stride_h = \left\lfloor \frac{H_{in}}{H_{out}} \right\rfloor, \quad kernel_h = H_{in} - (H_{out} - 1) \times stride_h \tag{2}$$

$$stride_w = \left\lfloor \frac{W_{in}}{W_{out}} \right\rfloor, \quad kernel_w = W_{in} - (W_{out} - 1) \times stride_w \tag{3}$$

H stands for the 'height' or the row size, and W stands for *weight*, or the column size. The stride determines how far the pooling window moves across the input at each step, while the kernel size defines the size of the pooling window. This ensures the input is downsampled to the desired output dimensions while retaining the most important features.

4.4 Multimodal Feature Processing

This section presents four different model architectures that can be used on the embeddings from Sect. 4.3. These include a *non-recurrent layer*, a simple feedforward network described in Sect. 4.4.1, along with *recurrent layers* (*Bidirectional Gated Recurrent Unit (BiGRU) model*, *Bidirectional Long Short-Term Memory (BiLSTM) model*, and *Transformer model*) tested in Sect. 4.4.2. These architectures are designed to harness the strengths of learned local dependencies while capturing temporal dependencies globally. Additionally, as a benchmark, we assess the performance of a non-recurrent model, which processes each shot independently without utilizing information from previous or future shots.

4.4.1 Non-recurrent model

Once the embeddings e_t^{audio}, e_t^{video} are passed into separate feedforward networks, the model learns the audio and video features through its layers. This process is achieved through a series of weights and non-linear activation functions within the fully connected layers, which allows the network to capture interactions and patterns that contribute to the task for the audio and video modalities within that specific shot.
Thus, the output at time step t, computed as:

$$y_t = g(e_t)$$

.
It reflects the combination of learned features from the audio and video embeddings, resulting in a more comprehensive understanding of the multimodal data.
While such a model can effectively use the local dependencies between audio and video modalities if appropriately trained, it can not use global dependencies. It can determine the current prediction relative to the features in other timesteps. We hypothesize that this is a crucial feature because a significant component of a detection model is its flow of action, people, and location. Therefore, we test with recurrent models as well.

4.4.2 Recurrent model

In contrast to the feedforward model, using a GRU, LSTM, or transformer model also allows the model to learn global dependencies. Sequential models are pivotal in processing data where order and context are essential, such as in time-series analysis. These models process input data sequentially, ensuring that the output at each step depends on the current input and the previous outputs, effectively capturing the temporal dynamics.
One state-of-the-art recurrent model is the BiGRU introduced in [40], which extends the traditional GRU architecture [17] by processing data in both forward and backward directions, enhancing the model's ability to capture context from both past and future frames. A BiGRU is a stacked GRU consisting of two GRU layers, where one is reversed, meaning it uses future states to predict the current state. Where an ordinary GRU uses the previous hidden states to predict the current state, this bidirectional approach efficiently models sequential

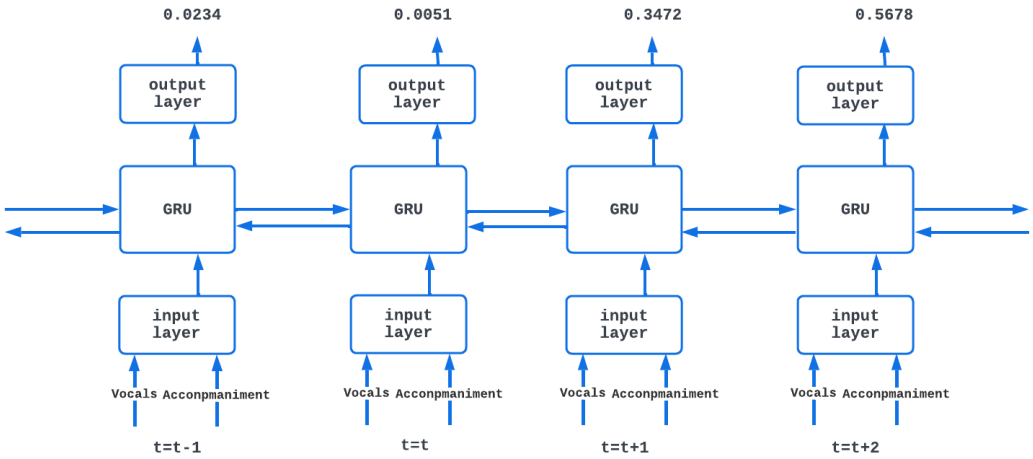


Figure 7: BiGRU model design

data like audio and video, allowing the network to understand temporal dependencies better. Since our model will not be used in real-time, future data will be available.

The GRU's efficiency in handling high-dimensional data makes it suitable for our case, where we process large-scale audio and video features. The GRU architecture includes the update and reset gates, which regulate the flow of information. These gates help determine what information should be retained or discarded, allowing the model to learn long-term dependencies without excessive computational costs.

While LSTMs are the predecessor of GRUs, they are slightly more complex in structure. LSTMs have three gates and two internal states (hidden and cell states). This allows for more control over what information is retained or discarded but can lead to slower training and more computational demand as the architecture consists of more parameters. In general, testing is the best way to conclude the choice of the model for the task. Like a BiGRU, a BiLSTM model also consists of two stacked layers, one of which is reversed.

In our framework, we use the embeddings from Equation 1 and process E^{audio} and E^{video} embeddings at each time step in an audio and video model. These embeddings are passed as input to a bidirectional recurrent model. In a bidirectional recurrent model, the sequence is processed simultaneously in both forward and backward directions, with the forward and backward hidden states computed independently at each time step.

At each time step t, the forward hidden state h_t^{fw} is computed as a function of the previous forward hidden state h_{t-1}^{fw} and the current embedding e_t:

$$h_t^{fw} = g_{fw}(h_{t-1}^{fw}, e_t) \tag{4}$$

Simultaneously, the backward hidden state h_t^{bw} is computed as a function of the future backward hidden state h_{t+1}^{bw} and the same embedding e_t:

$$h_t^{bw} = g_{bw}(h_{t+1}^{bw}, e_t) \tag{5}$$

The hidden states capture information from the past (via the forward layer) and the future (via the backward layer), allowing the model to learn richer temporal relationships between the audio and video features. The output at each time step t is based on this combined hidden state h_t, which includes contextual information from both directions. The output hidden state is then fused and processed by final layers that are a function of the hidden states and output a prediction y_t, explained in Sect. 4.4.3. The entire process is shown in the diagram of Fig. 7.

Transformers consist of an encoder-decoder structure, where each layer comprises multi-head attention and feedforward networks. Unlike GRUs and LSTMs, which rely on sequential processing and gated mechanisms to handle long-term dependencies, Transformers use self-attention mechanisms to capture parallel relationships across all sequence elements. These relationships, or attentions, are bidirectional, allowing the model to learn dependencies from past and future data. This enables them to model long-range dependencies more effectively without requiring sequential computation.

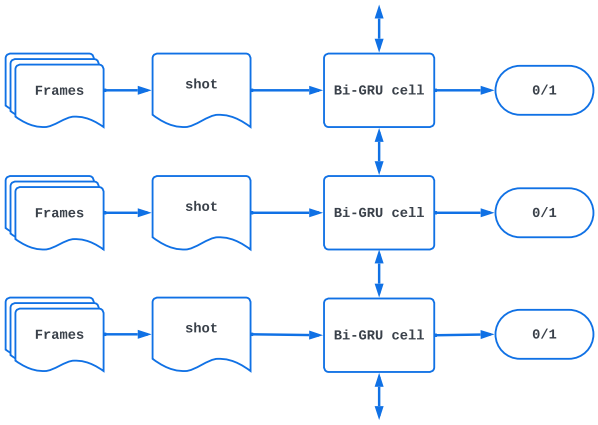


Figure 8: BiGRU model with local and global dependencies

The main advantage of Transformers is their ability to scale efficiently with larger datasets, making them the dominant architecture for models like *BERT* [20] and *GPT* [35]. However, they require significant computational power and memory, particularly during training. Testing and fine-tuning with a specific task is crucial to determine whether Transformers outperform other models, such as LSTM, in various contexts.
For a sequence of length T, the self-attention mechanism needs to compute the attention weights between every pair of tokens. Specifically, the computational cost scales as $O(T^2)$. This is because the model must calculate the attention score for each token in the sequence with all other $T-1$ tokens.

$$AttentionComplexity = O(T^2)$$

This quadratic complexity means that as you increase the sequence length, the memory and computation required grow rapidly. In our model, at each layer of the Transformer, the embedding $e_t \in [e_t^{audio}, e_t^{video}]$ at time step t is updated based on its relationship with embeddings from all other time steps. This is done through the scaled dot-product attention mechanism:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

.
Where Q, K, and V represent the queries, keys, and values derived from the concatenated embeddings, and d_k is the dimension of the keys.
Thus, each time step can attend to all others, at most, allowing the model to capture both local and global dependencies across the sequence of audio and video features. The output at each time step is computed as a weighted sum of the values, where the weights reflect the relevance of other time steps in the sequence.
We can stack these blocks on top of each other for all these models. Each block or layer can build upon the knowledge acquired by the previous layer, enabling the model to capture increasingly intricate relationships within the sequences. The input to the subsequential recurrent layer is the hidden state of the last layer, which is determined by the number of hidden nodes in that layer. A more significant number of hidden nodes allows the layer to represent more complex features and capture intricate relationships within the data.

$$h_2 = f(W_2 \cdot h_1 + U_2 \cdot y + b_2) \tag{6}$$

This equation shows how the second layer takes the hidden state from the previous layer (h_1), combines it with the current input (y) from the pooling layer, and transforms it through the activation function (f) using learned weights (W_2, U_2) and biases (b_2) to generate its hidden state (h_2). Stacking these layers effectively expands the model's capacity to learn long-term dependencies and temporal patterns across the audiovisual data. The number of stacked layers can be tuned during hyperparameter optimization to achieve the optimal balance between model complexity and performance. However, it's important to note that increasing the number of layers significantly impacts computational costs. To test the appropriate number of layers to capture the complexity of the data, we do hyperparameter tuning, as described in Sect. 5.2.5.

4.4.3 Modality Fusion and Prediction

In multimodal learning, *early fusion*,*late fusion*, and *hybrid fusion* are commonly used strategies for combining inputs from different modalities like audio and video:

- *Early fusion* involves combining the input features from both modalities (e.g., audio and video) at the very beginning of the model before any further processing. This approach allows the model to learn shared representations across the modalities from the start but can sometimes result in losing modality-specific features.
- *Late fusion* combines the features extracted from each modality after they have been processed independently by the model. This allows each modality to be modeled separately, capturing modality-specific information before combining them to make a final prediction.
- *Hybrid fusion* combines early and late fusion aspects. In this approach, each modality is initially processed independently through its dedicated network or processing layers, allowing the model to capture modality-specific features. After some independent processing, the intermediate representations of the modalities are fused, and the model jointly processes the combined features. This fusion can occur at different points in the network, allowing the model to retain modality-specific features while also learning interactions between the modalities. Hybrid fusion provides the best of both worlds, leveraging the strengths of early fusion by capturing joint representations while preserving the modality-specific information handled through separate early layers.

The fusion layer is pivotal in integrating the features extracted from audio and video modalities. By combining these multimodal inputs, the network can leverage complementary information, enhancing the robustness and accuracy of the scene change detection. The final output of the recurrent model is a sequence of hidden states $H = \{h_1, h_2, \ldots, h_n\}$ for both forward and backward layers.
Given two sequences of hidden states from a bidirectional recurrent model in a late-fusion or hybrid-fusion approach:

$$H^{fw} = \{h_1^{fw}, h_2^{fw}, \ldots, h_n^{fw}\} \quad and \quad H^{bw} = \{h_1^{bw}, h_2^{bw}, \ldots, h_n^{bw}\}$$

The forward hidden states H^{fw} and the backward hidden states H^{bw} capture context from the past and future, respectively. To combine these representations at each time step t, we apply concatenation for fusion. The forward and backward hidden states are concatenated to form the fused hidden state:

$$H_t^{bidirectional} = [h_t^{fw}; h_t^{bw}]$$

Next, these bidirectional hidden states must be concatenated for the audio and video embeddings to ensure that the last layers receive information from both modalities at each time step.

$$H_{fused_t}^{bidirectional} = [H_{audio_t}^{bidirectional} : H_{video_t}^{bidirectional}] \tag{7}$$

During training, the network's parameters are adjusted to optimize the model's performance on the classification task. The final feedforward layers can learn:

- **Modality-specific patterns:** The layers can capture patterns specific to each modality (audio or video), allowing the network to process the features of each embedding for scene change detection independently.
- **Cross-modality interactions:** The model can learn complex interactions and relationships between the two modalities by jointly processing the audio and video embeddings in the fusion layers.
- **Temporal dependencies:** The network can learn how past and future states are interrelated by the different modalities by processing forward and backward hidden states, allowing the model to capture temporal dependencies better, which is crucial for understanding the flow of scenes over time. For example, a sequence of tense music often precedes a fight scene, illustrating the temporal dependency between audio and video data.

The learning process occurs as follows: at each layer, the weights of the fully connected network are updated based on the backpropagation of errors, allowing the model to improve its ability to combine audio and video information iteratively. A non-linear activation function, ReLU, further enables the network to model complex relationships between the features from both modalities.

Once the hidden states are fused, the prediction at each time step t can be computed by applying a softmax function on the transformed fused hidden state:

$$\hat{y}_t = softmax(W_{out} H_{fused_t}^{bidirectional} + b_{out}),$$

The weight matrix W_{out} learns how to weigh the importance of the task's forward and backward contexts and audio and video modalities, with the bias b_{out} added to the softmax function. Moreover, the network learns cross-dependencies between the values of the hidden states passed to the final layers. For example, the final fully connected layer can identify patterns where audio from three shots ago interacts with visual frames from future shots, allowing the model to capture complex temporal relationships across modalities.
For scene change detection, we need the final layer to predict whether a given input frame or sequence of frames signifies a transition. The output layer consists of one node. If this node is active, which means the softmax function will be larger than 0.5, it will predict a scene transition and will predict 0 if it is smaller than 0.5.

5 Experimental Setup

In this section, we build upon the theory from Sect. 4 by detailing the application of the theoretical framework to evaluate our research questions. Previously, in Sect. 1, we introduced our central research question and several sub-questions. The question captures the central focus of our research:
"Can scenes be effectively captured automatically from raw TV content, using a supervised multimodal fusion learning approach?" .
First, we describe how we use data described from Sect. 3 and use it effectively to test our research questions in Sect. 5.1. Then, we discuss our setup to test our sub-questions in Sect. 5.2.1, 5.2.2, 5.2.3, 5.2.4, 5.2.5 and 5.2.6.

5.1 Data

This section describes the data approach used in this research and the challenges associated with accessing high-quality labeled datasets for effective use in our scene change detection model. Our research consists of two phases: pre-training and fine-tuning. We want to create a model that can adapt to different types of content, from talk shows to movies, without needing much labeled data to train a model from scratch. We must pre-train the model on a large dataset to learn some essential features associated with scene changes, for which we describe the data approach in Sect. 5.1.1. After this, we can use the pre-trained model and fine-tune it to the data type we want, covered in Sect. 5.1.2.

5.1.1 In-Domain

The availability of a free and large dataset has proven to be one of the bottlenecks in this field. Much of the work, including the work by streaming service companies [6] and [16], is kept in-house for larger datasets. Streaming services typically use proprietary databases from films to train models, complicating benchmarking analysis.
Previous research has predominantly used the *MovieNet* dataset [27] for pre-training models. MovieNet offers key content frames and audio and place features, as raw audiovisual files are restricted due to copyright issues. Consequently, this limitation means that the raw data is not free, and models trained on this dataset must rely on the features provided rather than raw data. For audio features, they use the *Short-Time Fourier Transform (STFT)* [31], a widely-used method that converts an audio signal from its original time domain into a two-dimensional representation in both time and frequency domains—however, state-of-the-art feature extraction models, like those discussed in Sect. 4.3 could significantly improve the performance over the one discussed in paper [27]. Therefore, we opt for the largest freely available dataset *Condensed Movies Dataset (CMD)* [8], which we already described in Sect. 3.
CMD, described in Sect. 3 requires some preprocessing steps, as only key scenes are available. CMD only consists of these key scenes, so they do not necessarily follow one another as in the original content. We attempt to address this issue by concatenating scenes as if they follow one another sequentially, which unfortunately disrupts the natural narrative flow. To navigate this challenge, we need to adjust our interpretation of training. Rather than focusing on learning transitions, we propose that the model might better learn to identify distinct events.

A significant problem with concatenating two separate events without their natural transitions and natural order in the narrative is the introduction of a bias. When concatenating non-subsequent scenes, the transition might be abrupt as they are not aligned. This bias suggests that abrupt changes in setting, sound, and characters indicate a scene change, whereas, in reality, these transitions run much more smoothly. We theorize that there are abrupt cuts even within a single scene, but recurring elements such as characters, settings, voices, and sounds should indicate continuity within the same scene. When a new scene begins—which in our concatenated sequence is not the actual subsequent scene—it represents a departure from the previous context.
Given these challenges and the limitations of the only large and freely available dataset, we slightly need to modify our research question and adopt the following assumption: *Assumption: Scenes are independent of one another, indicating that the features (e.g., sound, characters, action) of each scene do not rely on those in subsequent scenes. Furthermore, the sequential order of scenes is not critical for distinguishing between them.* This assumption allows us to proceed without expecting to uncover global across entire movie dependencies or require a complete narrative understanding for semantic interpretation. We can still build a model that understands when the context shifts from the current scene or event and hopefully leverage that to find global dependencies for the out-of-domain dataset discussed in Sect. 5.1.2.
Additionally, we need annotations indicating the scene transition points at the boundaries of each video. However, scene transitions can be subjective to the annotator, meaning their interpretation of a scene influences the ground truth. As a result, building a generic model to predict all scenes is constrained by the interpretation used in the dataset.
We hypothesize that this constraint can be partially mitigated by fine-tuning the model on a specific type of content, using labels annotated according to a different interpretation. Furthermore, with sufficient data, one can tailor the interpretation of a scene to one's preferences as long as the data aligns with that interpretation. This is also why [6] chose the off-by-n F1 score as an evaluation criterion, further explained in Sect. 5.3.2, to mitigate some of these interpretation biases on the exact timestep a scene starts or ends. It also accounts for slight timing errors and enables us to adjust the loss function to favor predictions that closely match the labeled scene transitions.
We aim to develop an end-to-end model that leverages raw video and audio data. Previous studies, such as [47] and [48], demonstrated poor performance of the audio modality when using the *NaverNet* embedder [18]. As a result, we opted to test state-of-the-art audio and video embeddings and decided against using the MovieNet dataset. This decision aligns with recent studies supporting alternative approaches [6], which we elaborate on in Sect. 5.2.3. Therefore, we utilize the CMD dataset, which is both sufficiently large and freely accessible. As outlined earlier in this section, we address the dataset's limitations by relaxing our research question and assuming that scenes are independent events, enabling us to work within these constraints.

5.1.2 Out-of-Domain

In Sect. 3 we introduced the *Hunted*, *Survivor* and *BBC Planet Earth's* datasets. These smaller datasets are interesting to our research to see how a pre-trained model can adjust to another domain with limited data (Q7).
Because annotations are not absolute ground truth, performance metrics based on them may not fully capture the model's capabilities. For example, a model might correctly detect a scene change based on a significant shift in visual or auditory elements but be penalized because the annotation prioritizes narrative continuity. Conversely, a model could match annotations perfectly without necessarily understanding the underlying semantics of the content. Therefore, the results of our out-of-domain dataset may require more intensive analysis to assess the performance than simply interpreting its performance using the metrics in Sect. 5.3.2.
Due to the limited data, it is crucial to assign content to the fine-tuning and testing sets carefully. To address this, we evaluated the performance of these datasets using cross-validation. This approach allows us to utilize more data for fine-tuning while avoiding conclusions based on only a few episodes.

5.2 Experiments

This section outlines our experimental approach to addressing the (sub)questions introduced in Sect. 1. The foundational methods supporting our experiments have been detailed in Sect. 4. Here, we apply these methods to evaluate and analyze the research questions systematically.

5.2.1 Audio Source Separation

Q1: Can performance be improved by separating the audio modality into two stems, vocals and accompaniment, using Spleeter?
We hypothesize that our method could benefit from such a separation model as audio in scenes consists of multiple components. For instance, background sounds might indicate the setting is in the middle of a bar, or the scene might contain a loud discussion between two characters. Such elements are essential to consider as a scene transition will not occur in the middle of a dialogue, monologue, or conversation in movies. The separation method Spleeter, explained in Sect. 4.1, enables isolated musical analysis and voice analyses. Musical analyses would allow us to recognize the meaning of music within a larger frame, such as the intro or outro of events using music. Voice analyses would allow for identifying characters, dialogues, and more, which holds valuable information in determining whether we are currently in the same scene as prior shots.
We evaluate the performance of the Spleeter model by comparing its results with a (*two-stem approach*), against a model that does not incorporate Spleeter (*single-stem approach*) on three tasks: urban sound classification, language classification, and music genre classification. These audio elements can contribute to the overall semantics of a scene, so we test whether splitting the audio into two stems improves task performance. After splitting or not splitting the data, we pass it on to a feedforward neural network (fully connected) with four layers. This arbitrarily chosen model should be able to get a reasonable estimate of the actual performance with and without Spleeter. Tab. 5 describes the datasets used in this evaluation.
These datasets consist of relatively 'clean' data, meaning there is little perturbation from non-relevant sounds to our task. To simulate realistic audiovisual content in our domain, we test the performance of the tasks with perturbing sounds. We add random background noise from the MUSAN dataset to both the music and speech datasets, as well as music to the speech and speech to the music classification tasks. These tests will show whether the two-stem approach can better isolate different acoustic elements—such as music, environment, speech, or action—and allow for better task performance. To validate this, we evaluate the models using metrics like accuracy, AUC, and F1-score, also described in Sect. 5.3.2.
In Sect. 6.1, we present the results for music classification with noise and speech and language classification with music and noise as perturbations.

Dataset	Task	Categories	Samples
UrbanSound8k	Urban Sound classification	10	8732
Librivox	Language classification audio books	16	172
MUSAL	Music genre classification	5	200

Table 5: Audio recognition dataset descriptions

5.2.2 Detection of Shot Boundaries

Q2: How effectively do shot boundary detection algorithms detect shot transitions within scenes?
As outlined in Sect. 4.2, we identified two primary reasons for treating each shot as a discrete unit in film analysis. First, this approach preserves the hierarchical structure of cinematic content, with shots functioning as the fundamental components of scenes. Second, it ensures synchronization across different modalities by employing a shot detection model to define each unit's start and end points precisely.
In identifying shot boundaries within video sequences, we evaluated three distinct methods—adaptive, content, and threshold-based—using PySceneDetect, as described in Sect. 4.2. Initially, these methods were tested on human-annotated episodes of Expeditie Robinson, with the results presented in Sect. 6.1.
To further assess the effectiveness of each method, we expanded our evaluation to the BBC Planet Earth dataset [9], which includes detailed annotations of shot boundaries for every episode. This dataset provides a diverse range of shot transitions, such as fades, dissolves, and intricate cuts—the most common transitions used between shots. Analyzing performance across these datasets aims to determine the most accurate and reliable approach to shot boundary detection. The methods were adjusted to include:

- Exclusion of black-and-white films, as mentioned in Sect. 4.2 transitions are primarily based on pixel intensities in the HSV colorspace. Content with exclusively black-and-white pixels could show bad performance for the methods, which is not indicative of the task. Therefore, we remove those from the analysis as well as the CMD [8] training set.
- shots exceeding 2 minutes or shorter than 0.1 seconds were deemed outliers or irrelevant for analysis.

Given the typical frame rate of 16 fps and an average shot length of approximately 3.6 seconds for CMD, as reported in paper [11], we can sample an average of 57.6 frames per shot. We argue that 16 frames per second is excessive for understanding the action in a shot, as many of the frames will be very similar. We, therefore, use a sampling rate of 16 frames per shot to represent a shot. This will lower the computational demands for the video modality. With 16 frames per shot, we fix the length of every video shot. VideoMAE can handle different frame rates and predict what happens with limited frames available. As VideoMAE [43] is trained using tube-masking, the embedders are trained to reconstruct missing data and can still make quality embeddings with 16 frames per shot.
We evaluate the *off-by-5* and *off-by-10* F1 scores, which are explained in Sect. 5.3.2, meaning that with a frame rate of 16 fps, a shot can be misaligned by up to 0.3125 and 0.625 seconds and still be counted as a true positive for that transition. This approach allows some flexibility in annotating a specific transition frame, particularly in fade transition cases.

5.2.3 Feature Extraction

Q3: How can we utilize established embedding models to shift from end-to-end learning models to a feature-based learning framework?
We will evaluate the performance of VideoMAE [43] and Wav2Vec 2.0 [7], as described in Sects. 4.3.1,4.3.2, in detecting various features within the shots. We use the second to last layer, the penultimate layer, for generating the embeddings, as most modern applications commonly use activations from the penultimate layer of deep learning models as embeddings [31].
Given the complex and time-intensive process of collecting and analyzing the models for tasks spanning various feature types, we do not conduct exhaustive tests of the embedders for face recognition, location recognition, action recognition, music classification, etc. Instead, we focus on evaluating the embeddings based on similarity, and we use t-SNE [44], discussed in Sect. 4.3.3, to identify shared features across pairs. However, it's important to acknowledge the limitations of this approach. The two-dimensional t-SNE visualization will not fully capture the complexity of the 768-dimensional and 1053-dimensional vectors produced by VideoMAE and Wav2Vec, respectively. While we cannot draw any definitive conclusions from this preliminary analysis, the results may indicate whether these models show promise in capturing the desired features.
In our analysis of VideoMAE embeddings, we assess the relative positions of embedded video shots from the films *"Home Alone"* and *"Forrest Gump."* We use these embeddings to explore the similarity between consecutive shots. Initially, we assess whether the embeddings can effectively cluster shots that share similarities in low-level features such as pixels, evaluating the model's ability to group these visually similar frames closely together. This approach determines if frames that appear alike are positioned near each other based on the embeddings. The analysis then extends to shots with high-level features such as the same location, action, or characters, exploring how the embeddings reflect these elements in their positioning. This qualitative assessment helps determine if the VideoMAE model effectively integrates such features into its vector representation.
To evaluate the audio embedder, we analyze its performance on the two distinct stems mentioned in Sect. 4.1: vocals and accompaniment. In Sect. 5.2.1, we validate the separation of audio data into these two stems using Spleeter. Subsequently, we assess how the Wav2Vec 2 embedder processes these stems. We use the Wav2Vec 2 LARGE model, which contains 24 transformer blocks with model dimension 1,024, inner dimension 4,096, and 16 attention heads. Its depth makes the model applicable to understanding the complex nature of acoustic features.
For the vocal embeddings, we provide audio transcripts of 18 shots from a scene in the movie "The Hunger Games." Shots 1 through 12 feature a dialogue between characters *Primrose* and *Katniss*. We expect that if the embeddings effectively capture the nuances of human voice recognition, distinct clusters or regions should emerge corresponding to each character's voice. Additionally, we aim to determine whether the model can capture the content of the spoken text in its embeddings, independent of the voice characteristics.
To evaluate the accompaniment embeddings, we focus on identifying the types of background sounds in "The Hunger Games." Each shot in one scene is manually annotated and categorized into one of two musical categories: Music or No Music. Furthermore, we categorize each shot in the same scene of the movie into one of three types: background voice, front voice, and no voice. The reason why we test these three voice types using the accompaniment stem is that we hypothesize that it is crucial to determine if there is a difference between the primary voice, no voice, and a voice in the background, which may more closely resemble the background sounds rather than being recognized as distinct voices. We consider the accompaniment and voice stems as containers where various sound components are segregated; we do not care which component is

placed in which bucket as long as it's placed consistently and can improve the single-stem approach. These tests provide deeper insights into how the models utilize these two categories to classify and cluster sounds, similar to the perturbation tests described in Sect. 5.2.1.
We assess whether the Wav2Vec model can effectively cluster shots that are similar in terms of music presence and the presence of vocal disturbances. This evaluation is crucial because music and voice are key elements that contribute to film narrative structure, playing a fundamental role in the semantics. Understanding how these auditory components interact and are integrated by the model helps to interpret their correlation and significance in the storytelling process. Therefore, our audio embedders with two stems must effectively capture these parts in one of its vectors.

5.2.4 Multimodal Feature Processing

Q4: Which model, recurrent or non-recurrent, can best be used for feature-based learning to predict scene changes?
In Sect. 4.4 We discussed the different approaches in multimodal fusion. To answer Q4 and Q5, we detail our pipeline's pre-training process and results, focusing on determining and validating the optimal model architecture. As part of the evaluation process, we assume that a lower validation loss indicates better practical performance.
To address question Q4, we test these three types of models using GRU, LSTM, and Transformer architectures and compare their performances against our baseline, a simple feedforward network. All recurrent models process batches of 100 shots each, configuring the models to learn the dependencies between these 100 shots. To determine if this sequence length is not too large, we will also evaluate their performance with a sequence of 25 shots. We do not test for larger sequences than 100 to avoid making the task more computationally demanding.
The baseline model is only passed one shot at a time and comprises two layers: a hidden layer with 64 nodes and an output layer that predicts '1' for a scene change and '0' otherwise for each shot. By introducing the baseline model configuration, we test for the best recurrent model and validate whether recurrent models yield better results than non-recurrent models. The model must predict a scene change based on one shot's visual and acoustic elements. It can, therefore, only use the local dependencies within that short timeframe (of, on average, 3.6 for our data), but not the global dependencies.
The GRU and LSTM models are structured with three layers: a bidirectional LSTM (biLSTM) or GRU (biGRU) block containing both forward and backward layers, each with 64 hidden nodes. These hidden states are then processed by a final linear layer with a single node to predict scene changes. The transformer model is slightly more complex; this Transformer architecture transforms the input data through linear expansions, positional encoding, multi-headed attention within the Transformer layers, and normalization steps, resulting in a classification output.
Q5: Does a multimodal fusion approach enhance performance compared to using audio and video separately?.
Additionally, to determine the most suitable (multi-)modal approach for our research, we evaluate the prediction performance of three model types: audio, video, and fusion, on CMD.
In Sect. 4.4, we discuss the concept of hybrid fusion. Through the implementation of hybrid fusion, the model taps into the strengths of each modality, thereby improving its capability to deliver more precise predictions or analyses by using features from both audio and video sources. In this hybrid approach, we treat the information from each modality independently, initially processing them through the embedder and subsequently through the (non-)recurrent models we evaluate for Q4. These processed outputs are then directed to a fusion layer.
The Fusion layer utilizes audio and video data to derive predictions and includes an additional layer compared to the separate audio and video models. This model channels the outputs from the penultimate layer into the linear fusion layer, facilitating the final prediction process. The fusion layer uses a non-linear activation function (*ReLU*), and finally, the second layer outputs the predictions, mapping the fused features to the target class. Since we have a binary prediction task, this output layer consists of one node, which is activated if the softmax function is larger than 0.5. Fig. 9 illustrates our setup for the Fusion model.
We eliminate the late-fusion layer for the audio and video models and instead directly use a final layer to make predictions, which processes the outputs from the (non-)recurrent models we evaluate in Q4.
We examine the learning curves to assess a GRU model's audio, video, and fusion effectiveness in learning from the data. These curves will illustrate how quickly each model approaches its minimum loss and indicate instances of overfitting on the training data. After answering Q4 and Q5, we determine which model to continue with, Sect. 5.2.5 covers the hyperparameter tuning phase of this model.

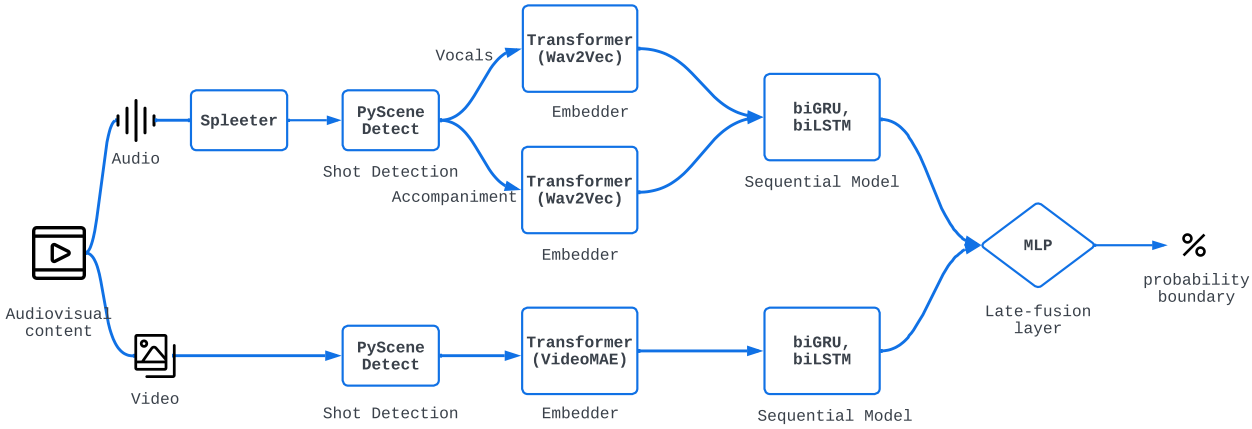


Figure 9: Late-fusion model pipeline

5.2.5 Hyperparameter Tuning

Q6: What are the optimal hyperparameters for maximizing the performance of this model on the dataset?
This subsection details the specific hyper-parameters utilized within our models, configurations, and the reason behind testing these hyper-parameters. The results of Q6 are provided in the results Sect. 6.5.
Our architecture tests use single and multiple stacked BiGRU or BiLSTM layers to extract complex temporal features from the preprocessed data progressively. We must test some crucial hyper-parameters for quality performance to optimize our model. We test hyperparameters using a grid search. A grid search is a hyperparameter optimization technique that tests a predefined set of hyperparameter values to find the best combination for model performance. We test for the following parameters and values:

- **Number of Layers**: This refers to the depth of the network. As we use bidirectional blocks, and every block consists of a forward and backward layer, we only test for multiples of two as the number of layers. More layers allow the model to learn more complex patterns but can also increase the risk of overfitting and require more computational resources for training. We test for the number of hidden layers $L \in \{2, 4, 6\}$.
- **Number of Nodes**: The number of nodes (or the hidden size) in each layer defines the capacity of that layer. More nodes enable the model to capture more intricate details in the data but can also make the model more prone to overfitting and computationally expensive. We test for size $h \in \{64, 128, 256, 512\}$
- **Learning Rate**: This is the step size at each iteration of the optimization process. It controls how much the model's parameters are updated in response to the computed gradient. A smaller learning rate leads to slower, more precise updates, while a larger learning rate speeds up training but risks overshooting the optimal solution. We test for learning rates $\varepsilon \in \{0.01, 1e^{-3}, 1e^{-05}\}$.

After identifying the optimal model as described in Sect. 5.2.4, we proceed to determine the appropriate width (number of nodes) and depth (number of layers) for each model. The configurations for depth and width are tested within the (non-)recurrent blocks outlined in Sect. 5.2.4. This involves experimenting with varying numbers of nodes and layers for the GRU, LSTM, Transformer, or feedforward blocks based on the best-performing model identified from Q4. Additionally, we conduct a grid search for the audio, video, and fusion models, regardless of the findings from Q5, to explore whether different parameter configurations might influence the outcomes.
Based on the initial test for determining the optimal epoch count, we start at a significantly higher number of epochs than the point at which we expect to achieve the minimum loss. After each epoch, we assess whether the validation loss has decreased below the previous minimum. We save the weights and loss as the new best model with minimal loss if it has. This approach guarantees we consistently capture and retain the best-performing model throughout the training session.
Additional enhancements made to our model to optimize performance are detailed in Sect. 5.3. These additional components are included in the model based on previous positive outcomes in the field, we do not directly test their impact on performance enhancement in our current study.

5.2.6 Learning Out-of-Domain Scene Changes

Q7: Can a model learn out-of-domain scene changes (with a different interpretation of a scene) from little data?
In this section, we test the pipeline for fine-tuning on another domain. The primary objective is to enhance the model's performance in these domains by using a pre-trained model and adjusting it to different types of content (or domains) that could include news, cartoons, talk shows, reality TV, and game shows. This approach involves using the large pre-trained model as the foundation from Sect. 5.2.4 is refined using smaller, domain-specific datasets. The fine-tuning process ensures that the model retains its general knowledge while gaining expertise in particular content areas, thus improving its accuracy and relevance for domain-specific tasks with a limited amount of data necessary. By applying this methodology, we aim to demonstrate the effectiveness of targeted fine-tuning in achieving good results for domains where data is limited. In our analysis, we include TV shows: *Hunted*, *BBC Planet Earth*, and *Survivor*.
A significant challenge comes from varying interpretations of what defines a scene. For instance, scenes in BBC's *Planet Earth* typically last around 1 minute as can be seen in Tab. 3, whereas our training data consists of scenes averaging around 2 minutes, from Tab. 1, similar to shows like *Hunted* and *Survivor*, Tab. 4. This difference can lead the model to predict too few or too many scene boundaries because it might have a different interpretation of scenes.
As a result, we need to reevaluate how we interpret the model's predictions, moving beyond a fixed threshold of 0.5. Instead, we can assess scene boundaries based on the relative scores of different shots. For example, if most shots are predicted with a score of 0.01 and one stands out with a score of 0.3, this likely signals a meaningful distinction. Similarly, when the model frequently overshoots the boundary and some predictions hover around 0.5, these might not always indicate an actual scene change. To compensate for this different interpretation, we decide to resolve this problem using two enhancements:

1. We adjust the weight of the scene boundary class towards the inverse ratio of shots that are scene boundaries and not. This ensures that we accommodate for the class imbalance.

2. We determine the optimal threshold for predicting a scene in the out-of-domain dataset by calculating the optimal Youden J statistic. After establishing this threshold, we reassess the model's performance using the newly set threshold to verify its effectiveness. More on the Youden J statistic and other metrics used for analysis in Sect. 5.3.2.

Fine-tuning the model on the domain-specific data helps it adapt to the annotator's unique interpretation of what defines a scene. Adjusting the model using examples marked by the annotators helps it learn to recognize and replicate its specific criteria for scene changes, improving its alignment with its judgments in determining a scene's start and end point.
With the limited data, we must carefully select the right size for training, validation, and testing. The larger the training dataset, the more the model can learn. However, this means we have a smaller validation and test set, which will lead to a less confident estimation of the quality of the model. Because of the smaller dataset, we use cross-validation with two validation episodes and 1 test episode for all TV shows. We fix the number of validation and test episodes; as a result, the number of folds is limited to the number of episodes the dataset contains, as we do not test the model twice on the same episode. This approach allows us to evaluate the model's performance more robustly by ensuring that the results are not dependent on a single data split. Each fold of the cross-validation process provides an opportunity to train the model on different subsets of the data, thereby enabling us to assess its consistency and reliability across various scenarios. The variance of the cross-validation will also provide insights into the variability of the TV shows. If the TV shows exhibit consistent structural patterns, the impact of the chosen split on the results will be minimal, leading to lower variance in the output. Conversely, if there is significant variability in the structure of the TV shows, we might observe a greater variance in the results.
We do not need an extensive search for optimal parameters for fine-tuning since the model architecture has already been defined during the pre-training phase. We set the number of epochs to 20 and chose the lowest validation loss. After ten epochs, we observe that the model will overfit on the relatively small datasets. We stick to fine-tuning to a low learning rate of 0.0001. In cases where the dataset and number of training steps are smaller, a slightly higher learning rate can be advantageous, helping the model converge faster to improved weights.

5.3 Model Optimization

This section outlines the specific arrangements and settings of the models used in this study, specifically those of Sect. 4.4. It details the architecture, including layer configurations, activation functions, and connectivity between components. This section also describes the operational parameterers, such as learning rates, batch sizes, and the number of epochs, providing a comprehensive view of the model's setup for replicability and further research. By defining these configurations, we ensure that the model's architecture is optimized for handling the particular challenges posed by the audiovisual scene change detection task.
Sects. 4.1, 4.2 and 4.3 discuss pre-trained models, meaning they already have optimized configurations and do not require additional training. In contrast, Sect. 4.4 presents a model that must be trained and optimized. Therefore, we must determine the optimal model configuration and training setup to achieve minimal loss. In this section, we will focus on these aspects.

5.3.1 Training Configuration

In this section, we describe some settings crucial for optimized training, in contrast to Sect. 5.2.5 we do not exhaustively test these settings but instead rely on the proven performance of these techniques. The learning rate is set at 0.001, and the data is processed in sequences of 100.
A higher batch size can increase the efficiency of the model's training. Torch has optimized batch computing by leveraging the parallel computing capabilities of *GPUs*. However, we face limitations in using a large batch size due to large sequences of embeddings with many dimensions and memory load on the GPU. The main reason for using a large batch size is to make the loss calculation more stable by avoiding weight updates based on a very small sample size or outliers. However, this is less of a concern because we calculate the loss over an entire sequence, averaging it across all shots. For example, a sequence with 100 shots results in 100 individual loss calculations, meaning the average loss is already more robust.
To train our model, we must get feedback from the current predictions. This is typically done using a loss function. Our model employs the *Binary Cross-Entropy (BCE)* loss function integrated with a *sigmoid activation function* to handle the output logits effectively. This combination is crucial for addressing class imbalances found commonly in datasets with scene transitions compared to non-scene transitions. According to [42], a standard method for implementing a weighted loss function involves assigning higher weights to the minority class and lower weights to the majority class.
In Sect. 3.1, Tab. 1 indicates that the median clip length is 132 seconds, while each shot lasts approximately 3.6 seconds, resulting in a ratio of 1:36.7. However, Tab. 2 shows 4,900 shots across 670 scenes, leading to a ratio of 1:7.3 between shots and scenes. Based on this analysis, we assigned a weight ratio of 20:1 for training, balancing the class distribution to reflect the relative importance of minority classes in the model's loss function.
The adjusted BCE loss function, which applies differential weights to the classes, is given by:

$$L(x, y, w) = -\frac{1}{N}\sum_{i=1}^{N} w_i \left[y_i \cdot \log(\sigma(x_i)) + (1 - y_i) \cdot \log(1 - \sigma(x_i))\right] \tag{8}$$

Here, x denotes the logits - the model outputs before the sigmoid transformation. The target labels y are binary, where 1 represents a scene transition and 0 represents a non-transition. Weights w are assigned such that $w_1 > w_0$, where w_1 is the weight for the scene transition class and w_0 for the non-transition class. This weighting strategy ensures the model adequately focuses on the underrepresented scene transition class, thereby correcting for the imbalance and promoting a more balanced learning outcome.
We calculate the loss over all predictions in one sequence to ensure stability in the learning process. The loss calculated before an update step is the sum of the losses in a sequence, where the sequence size is tested for different values.

$$L_{Seq} = -\frac{1}{Seq}\sum_{i=1}^{Seq} L(x_i, y_i, w_i) \tag{9}$$

Following the computation of the sequence loss, gradients must be calculated for each model parameter to update the weights and biases effectively. The gradient of each parameter is determined by differentiating the batch loss for that parameter. This can be expressed as:

$$g = \frac{\partial L_{Seq}}{\partial \theta} = \frac{1}{Seq} \sum_{i=1}^{Seq} \frac{\partial L(x_i, y_i, w_i)}{\partial \theta} \tag{10}$$

Where θ represents any parameter of the model (e.g., weights or biases). $\frac{\partial L(x_i, y_i, w_i)}{\partial \theta}$ is the partial derivative of the loss function for the parameter θ for the i-th example in the sequence.
This gradient tells us the direction and magnitude by which the parameter θ should be adjusted to minimize the loss, providing a pathway toward more accurate model predictions.
For updating the parameters based on the gradient in Equation 10, we employ the *Adam optimizer* [21], which is widely used in the machine learning community. According to [21], the optimizer is *"computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters"*. The parameters are updated using the following updating rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{11}$$

where θ_t are the parameters at time step t, η is the learning rate, and ϵ is a small scalar added to improve numerical stability.
We incorporate a learning rate scheduler with warmup steps to improve our neural network's training stability and efficiency. The scheduler gradually increases the learning rate from an initially small value to the predetermined maximum learning rate over a set number of warmup steps. This method is particularly beneficial during the initial phase of training, where starting with a learning rate that is too high could lead to undesirable divergences.
The learning rate adjustment can be mathematically represented as follows:

$$\eta_t = \begin{cases} \frac{t}{T_{warm-up}} \eta_{max} & \text{if t} < T_{warm-up} \\ \eta_{max} & \text{otherwise} \end{cases} \tag{12}$$

where η_t is the learning rate at step t, $T_{warm-up}$ is the total number of warm-up steps, and η_{max} is the maximum learning rate.
This warmup process allows the optimizer to start with smaller, more cautious steps, making it less likely to overshoot minima in the loss landscape at the beginning of the training. As the training progresses past the warmup phase, the learning rate stabilizes at η_{max}, allowing for more robust convergence in subsequent training iterations.
To improve the model's generalization and prevent overfitting, we incorporate *weight decay*, or *L2 regularization*, techniques within the network. Weight decay is a regularization used primarily to prevent overfitting in training neural networks. It works by adding a penalty term to the loss function, which is proportional to the sum of the squared values of the model parameters (weights). This penalty term effectively discourages large weights by shrinking them during training.
The main idea behind weight decay is to keep the model weights small, which can help make the model less sensitive to noise in the training data. This can improve the model's generalization capabilities to new, unseen data, preventing it from fitting too closely to the training set, a phenomenon known as overfitting. Weight decay is sometimes referred to as L2 regularization when the penalty is the L2 norm of the weights.
The formula for weight decay, which is a form of L2 regularization, is incorporated into the loss function during the training of a neural network. The updated loss function with weight decay is given by:

$$L' = L + \lambda \sum_{\theta} \theta^2 \tag{13}$$

The term $\lambda \sum_{\theta} \theta^2$ acts to penalize larger weights, encouraging the model to maintain smaller weight values. This penalty is added directly to the loss function. During training, the optimization algorithm (like *Stochastic Gradient Descent* (SGD)) will minimize this adjusted loss function, balancing the fit to the data with the magnitude of the weights. The Adam update steps, with regularization, now look like:

$$\theta_{t+1} = \theta_t - \left(\frac{\eta}{\sqrt{\hat{v}_{\theta,t}} + \epsilon} \hat{m}_{\theta,t} + \eta \lambda \theta_t \right) \tag{14}$$

5.3.2 Evaluation Metrics

In addition to outlining the neural network architecture and its parameters, it is crucial to discuss the choice of the loss function used during the training of our models. To enhance the accuracy of scene change detection, especially in handling the ambiguity and inconsistency of human-labeled data, we employ the off-by-n F1 score as our primary loss function. This loss function is particularly suited for the complexities involved in detecting scene cuts in movies, where transitions may not always be abrupt or clearly defined. The off-by-n approach allows predictions within an n-shot or n-frame range, depending on the context, of the actual transition points to be considered correct. It provides a balanced measure that emphasizes precision and recall and ensures that the model does not overly penalize near-miss predictions. The off-by-n F1 score is designed to accommodate minor differences between the predicted scene transitions and the ground truth labels.

The off-by-n F1 score modifies the traditional F1 score by expanding the criteria for a true positive. A prediction is a true positive if it falls within a window of n frames or shots from a labeled transition point. For scene change detection, n refers to the range of shots; for shot detection, n refers to the range of shots. This modification is implemented by adjusting the precision and recall calculations to include predictions within this range. The formula for off-by-n F1 score is given by:

$$F1_{off-by-n} = 2 \times \frac{Precision_{off-by-n} \times Recall_{off-by-n}}{Precision_{off-by-n} + Recall_{off-by-n}}$$

Where $Precision_{off-by-n}$ and $Recall_{off-by-n}$ are calculated by considering detections within the n-frame margin around each ground truth transition. By incorporating the off-by-n F1 score into our training process, we can better train models that are robust to minor inaccuracies in data labeling and more aligned with the practical requirements of scene transition detection in movies. This approach not only improves model tolerance to annotation errors but also potentially increases the usability of the model in real-world applications where exact frame accuracy is less critical than detecting the proximity of scene changes. In [36], they use a time-interval acceptance range of 10 seconds. In [25], they use an acceptance range of three preceding and following shots. This tolerance acknowledges the practical challenges of pinpointing exact transition frames and mimics the real-world leniency in scene-cut editing. However, it complicates comparing different algorithms and makes benchmarking studies difficult. In [46], they propose using measures coverage and overflow. In this paper, they refer to a scene as a *Logical Story Unit (LSU)*, the measures, overflow, and coverage consider the number of shots guessed correctly inside such an LSU figure.

Following the discussion on the off-by-n F1 score, another crucial evaluation metric that merits attention in the context of model performance is *Average Precision (AP)*. AP evaluates how well a model ranks instances, focusing on the placement of true identifications throughout the result set. It is calculated by integrating the area under the precision-recall curve, reflecting both the precision and recall at every threshold where a relevant item is retrieved:

$$AP = \sum_{k=1}^{n} (R_k - R_{k-1}) \times P_k$$

Represent the precision and recall at the k-th threshold. This formulation ensures that more weight is given to thresholds where an increase in recall occurs, thereby emphasizing the importance of retrieving all relevant items while maintaining high precision.

The significance of AP in practical applications lies in its ability to penalize incorrect rankings more heavily when they involve higher-ranked items, thus rewarding systems that not only identify relevant items but also rank them higher. This makes AP an invaluable metric for systems where the order of results significantly impacts user experience or effectiveness, such as in search engines or multimedia retrieval.

In addition to Average Precision and the off-by-n F1 score, two other essential metrics used for evaluating model performance are the *AUC (Area Under the ROC Curve), Youden's J* statistic and the *Euclidean distance*. The AUC is a widely used performance metric that evaluates the overall ability of a binary classifier to differentiate between positive and negative instances across all possible classification thresholds. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) for varying thresholds, and the AUC summarizes the area under this curve. AUC represents the model's overall ability, whereas Youden's J is focused on finding the single best threshold for decision-making. Youden's J statistic is a metric derived from the ROC curve that measures the performance of binary classification models. It is defined as the point on the ROC curve that maximizes the difference between the true positive rate (sensitivity) and the false positive rate (1-specificity), represented mathematically as:

$$J = Sensitivity + Specificity - 1$$

We use the Youden J statistic to determine the optimal threshold, also known as the optimal cut-point estimate. An optimal cut-point estimate refers to finding the threshold value for a classification model that best separates the classes (usually binary) based on a performance criterion.
To find the optimal threshold, we compute Youden's J statistic for each threshold and select the threshold that maximizes the value, corresponding to the minimal Euclidean distance metric. Lastly, the Euclidean distance metric can be used on the ROC curve to assess how close a classifier's performance is to the ideal point $(0, 1)$, which represents perfect sensitivity and no false positives. The Euclidean distance from a point (FPR, TPR) on the ROC curve to this ideal point is given by:

$$d = \sqrt{(1 - TPR)}^2 + (FPR)^2$$

where TPR is the true positive rate and FPR is the false positive rate. A smaller Euclidean distance indicates better classifier performance, as the point is closer to the optimal ROC curve.
We evaluate performance using several metrics, not only because each metric may provide a different perspective on the model's true performance but also because many studies employ different evaluation metrics. Consequently, reporting multiple metrics ensures that future research can more easily compare results on these datasets across various methodologies.

5.4 Technical Specifications

Here, we comprehensively describe the hardware and software setups utilized to train our model. We outline the hardware configuration, which includes the *ALICE High-Performance Computing (HPC)* facility and the various software libraries and tools integrated into our framework.
We train our model on *ALICE* [33], which is the *High-Performance Computing (HPC)* facility of Leiden University. The HPC has multiple CPUs and GPUs and contains all the requirements for efficiently training our model. The model training specifically utilizes the GPUs due to their parallel processing capabilities, which are crucial for handling batch operations where computations can be performed simultaneously. This significantly reduces the overall computation time.
The GPUs in use are NVIDIA GPUs, which are paired with Intel CPUs, optimizing both computation speed and efficiency. Our code is developed in Python 3.7, which is well-supported by the NVIDIA *CUDA framework* for accelerating computations using GPU resources.
The system now consists of a total of 48 CPU and GPU nodes. The system is equipped with *NVIDIA Tesla T4, GeForce RTX 2080TI* and *A100 GPUs*. We specifically use A100 GPUs.
Some tasks that do not require exhaustive GPU training run on a *2.3 GHz Quad-Core Intel Core i5* processor. We developed our framework using Python 3.7 for training and testing our models. We primarily utilized the *Keras* and *TensorFlow* libraries for model training, with TensorFlow handling the model architecture development. In the case of Wav2Vec, which was initially built using Torch, we integrated it with the Keras framework to maintain consistency in our approach. Keras allows us to convert Wav2Vec's output into tensors, enabling smooth integration with TensorFlow. Similarly, for VideoMAE, the original implementation is based on TensorFlow. Moreover, we have used libraries Spleeter Because of its ability to split the audio data into vocals and accompaniment, as mentioned in sect. 4.1. As discussed in Sect. 4.2 We have used the PySceneDetect library for shot detection. Additionally, we utilized several other essential libraries for our project: *matplotlib* for generating and visualizing graphs, *cv2 (OpenCV)* for displaying output videos after scene change detection, *sklearn* for its statistical performance metrics and pandas for its data manipulation and storage capabilities. These libraries helped streamline our workflow.
Memory management has been a persistent challenge, mainly due to the high dimensionality of the video and audio embeddings, which are processed sequentially. Handling large sequences of data can lead to memory problems. To mitigate these issues, it is crucial to implement regular memory clearing throughout the training process. This involves freeing up any unused memory resources to prevent memory overload and ensure smooth execution, which improves training efficiency and reduces the risk of crashes or slowdowns.

6 Results

In this section, we test different facets of our entire pipeline. We follow the structure of Sect. 5 and our pipeline for our test results, focusing on the critical stages of data preparation, pre-training, and fine-tuning. We employ different configurations and techniques to optimize the performance of our models on various

Figure 10: Home alone subsequent shots, similar shots reappear

modalities, including audio, video, and fusion. The subsequent sections provide detailed insights into the methodologies and results obtained at each stage.

6.1 Audio Source Separation

First, we present the results of our experiments designed to address the research question outlined in Sect. 5.2.1:
Q1: Can performance be improved by separating the audio modality into two stems, vocals and accompaniment, using Spleeter?
Tabs. 6 7 and 8 present the results of the experiment described in Sect. 5.2.1.

Table 6: UrbanSound8k Dataset, environment sound classification task

Data Type	Method	Accuracy	F1-Score
Environment	Single-stem	0.634	0.584
Environment	Two-stem	**0.646**	**0.642**

Tab. 6 shows that for classifying environment sounds a two-stem approach ($F1_{score} = 0.642$) is outperforming a single-stem approach ($F1_{score} = 0.584$).

Table 7: MUSAN Dataset, music recognition task scores with different types of pertubation

Data Type	Method	Accuracy	F1-Score
Music	Single-stem	0.512	0.500
Music	Two-stem	0.512	**0.535**
Music + Noise	Single-stem	**0.463**	**0.446**
Music + Noise	Two-stem	0.439	0.432
Music + Speech	Single-stem	0.415	0.430
Music + Speech	Two-stem	**0.512**	**0.486**

Table 8: Librivox Dataset, speech recognition task scores with different types of perturbation

Data Type	Method	Accuracy	F1-Score
Speech	Single-stem	**0.637**	0.624
Speech	Two-stem	0.571	**0.679**
Speech + Noise	Single-stem	0.457	0.576
Speech + Noise	Two-stem	**0.514**	**0.582**
Speech + Music	Single-stem	0.543	0.552
Speech + Music	Two-stem	**0.571**	**0.564**

We observe in Tab. 7 that noise interferes with music because both are treated as accompaniment, resulting in a lower F1 score for the two-stem approach ($F1_{score} = 0.432$, compared to $F1_{score} = 0.446$). In contrast, speech and music are effectively separated into vocals and accompaniment, respectively, and as a result, the two-stem approach ($F1_{score} = 0.486$) outperforms the single-stem approach ($F1_{score} = 0.432$). The same effects can be found in Tab. 8 for the speech task with music perturbation. This evidence suggests that a simple model can more effectively recognize the two sounds when separated using Spleeter when combining music and speech. This demonstrates Spleeter's valuable contribution to the scene change detection task, as it

improves the model's ability to identify both music and speech components accurately. In Tab. 8, we see that for the speech task with noise perturbation, the two-stem approach outperforms the single-stem approach. This aligns with our initial assumption that noise will not qualify as vocals, according to Spleeter. As a result, separating the data using Spleeter simplifies the speech + noise task, making it easier to isolate speech from background noise. This distinction is evident in the performance results when using Spleeter, which shows improved outcomes over the non-Spleeter approach, particularly when combining music and speech for both music and speech classification tasks. Moreover, we do not observe a decrease in performance when classifying the clean music and speech datasets. This outcome validates our approach of splitting audio into accompaniment and vocals, as in our data, music, and speech are often combined and serve as critical indicators of the content's semantics.

6.2 Detection of Shot Boundaries

In this section, we present the results of our shot boundary detection experiments using the PySceneDetect model. We try to answer:
Q2: How effectively do shot boundary detection algorithms detect shot transitions within scenes?
The experiment is described in Sect. 5.2.2, and the results of that experiment are shown in Tabs 9 and 10.

Table 9: Expeditie Robinson Shot detection results, with off-by-n frames f1

Method	off-by-5 f1	off-by-10	f1
Threshold	0.00	0.00	0.00
Content	0.88	0.88	0.78
Adaptive	0.89	0.89	0.80

We observe an off-by-10 f1 score of 0.89 for the adaptive method, compared to a score of 0.87 for the content-aware detection model, as seen in Tab. 9. In this research, we continue with an adaptive method. We tested the content and adaptive methods on a BBC Planet Earth dataset.

Table 10: PySceneDetect on BBC *Planet Earth* Dataset for adaptive and content methods

Video Name	**#Shots**	**off-by-n F1**			
		Adaptive		*Content*	
		n=5	*n=10*	*n=5*	*n=10*
From Pole to Pole (01)	445	0.899	0.902	0.856	0.861
Mountains (02)	383	0.908	0.908	0.857	0.860
Ice Worlds (03)	421	0.951	0.953	0.848	0.854
Great Plains (04)	472	0.910	0.914	0.876	0.887
Jungles (05)	460	0.949	0.949	0.875	0.885
Seasonal Forests (06)	526	0.926	0.928	0.885	0.903
Fresh Water (07)	531	0.909	0.911	0.872	0.881
Ocean Deep (08)	410	0.880	0.880	0.740	0.751
Shallow Seas (09)	366	0.943	0.943	0.866	0.874
Caves (10)	374	0.853	0.855	0.661	0.682
Deserts (11)	467	0.933	0.933	0.901	0.909
Total	5322	0.915	0.916	0.840	0.850

Tab. 10 shows that the Adaptive method achieves higher F1 scores for $n = 5$ and $n = 10$ than the content method. Additionally, the minimal score difference between $n = 5$ and $n = 10$ for the Adaptive method suggests higher precision in identifying the exact shot within transitions. The adaptive method gets the best $F1 - score$ on the BBC dataset, which features a mix of fading transitions and fast cuts, giving us some confidence in the robustness of accurately extracting shots within scenes from varying types of content. The poor performance of the Caves episode, as shown in Tab. 10, could be attributed to the fact that caves are mostly dark environments, resulting in many black pixels in the shots. Fig. 11 further shows why the shots in this episode are not detected as accurately as in other episodes.

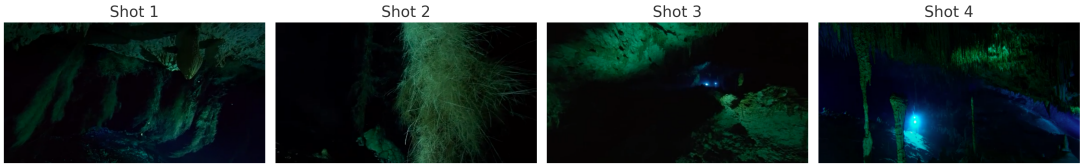


Figure 11: Relative poor performance of Caves episode could be due to shot transition that contains mostly black pixels

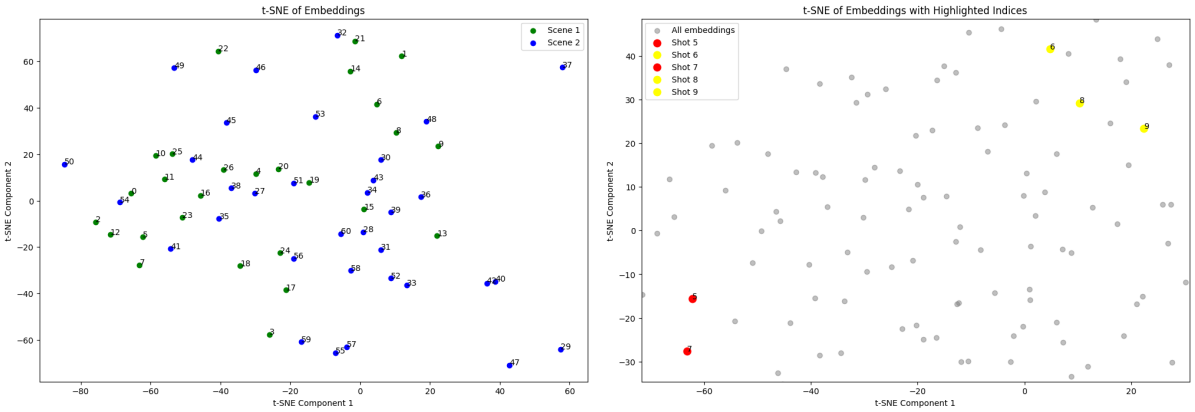
6.3 Feature Extraction

This section presents the results of our experiments designed to address the research question outlined in Sect. 5.2.4:
Q3: How can we utilize established embedding models to shift from end-to-end learning models to a feature-based learning framework?

6.3.1 Video Features

When we see subsequent shots from the *Home Alone* movie, shown in Fig. 12. Fig. 13b depicts shots within the same area, where consecutive shots share the same location and character between distinct shots. This is the same phenomenon shown in Fig. 1 earlier. Now Fig. 12 illustrates how similar embeddings can suggest that the shots belong to the same scene if they are within a reasonable time and have a high similarity in their embedding vector. The model can infer whether the scene is still going on by examining the similarities between shots within a reasonable timeframe. Fig. 13a presents the VideoMAE embeddings from a scene in the movie Home Alone, with Fig. 13b showing that the positions of the embeddings effectively capture the similarities between shots. However, it is essential to note that since t-SNE only preserves the relative positioning of embeddings in two dimensions, we cannot draw definitive conclusions about the exact distances between the embeddings. While the example in Fig. 10 is straightforward and doesn't necessarily showcase advanced feature extraction, it still illustrates the potential of embeddings in capturing scenes and the pattern shown in Fig. 1.

Figure 12: Home Alone subsequent shots



(a) All shot embeddings plotted using t-SNE for Home Alone (1990)

(b) Shots 5, 6, 7, 8 and 9 embeddings positions using t-SNE for Home Alone (1990)

Figure 13: Various visualizations for the movie Home Alone (1990). (a) t-SNE plot of all shot embeddings. (b) t-SNE plot of shots 5, 6, 7, 8, and 9 embeddings.

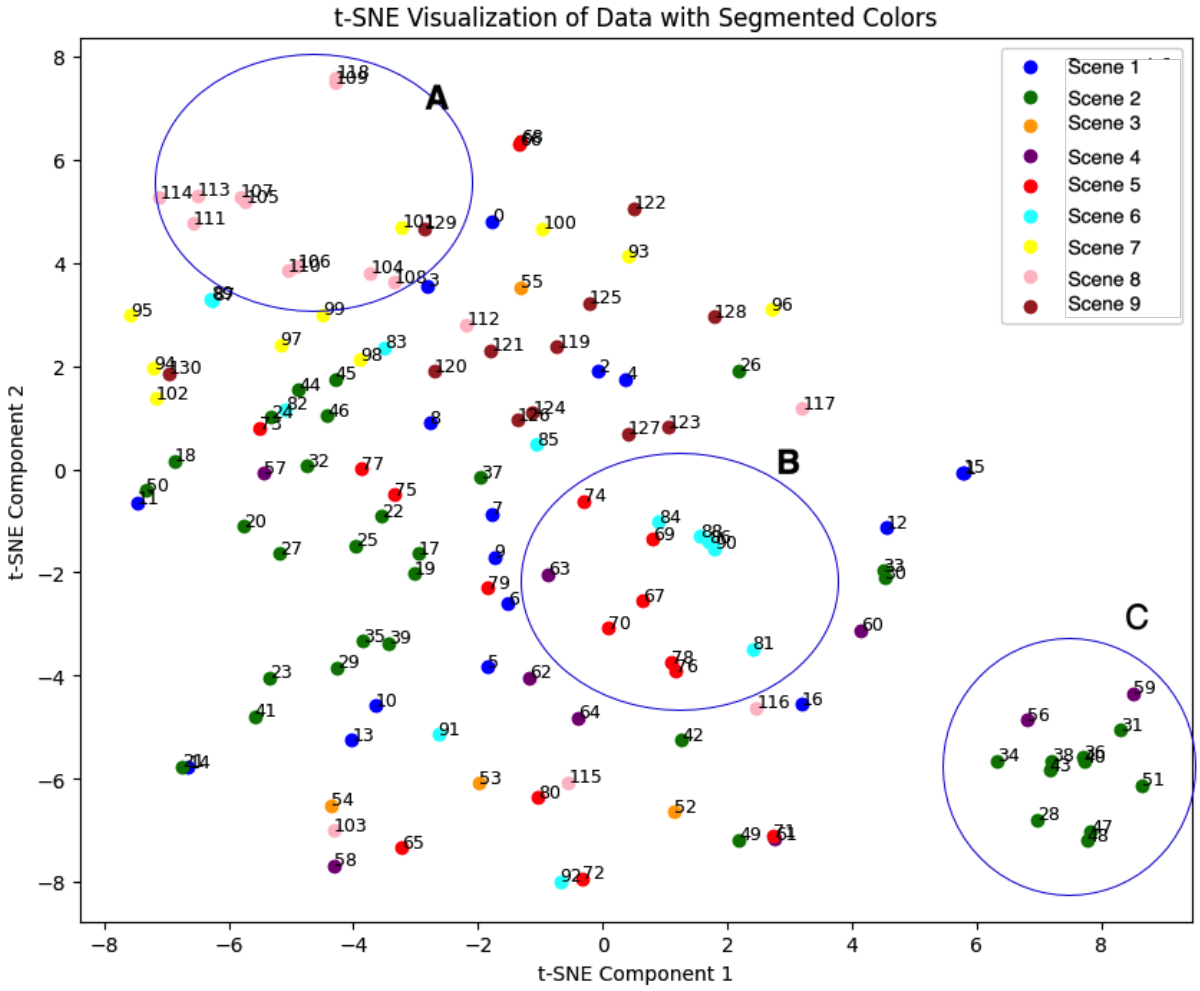


Figure 14: A: Conversation Jenni and Forrest, B: Conversations Lt Dan and Forrest in two separate scenes, C: Forrest Running

To further analyze the ability of VideoMAE to group video shots based on features, we present the embeddings of 9 scenes from Forrest Gump used in our analysis. Cluster A consists of shots from a conversation between Jenny and Forrest. The embeddings for Jenny are found in shots 104, 106, 108, and 110, while the embeddings

for Forrest appear in shots 105, 107, 109, and 111. This demonstrates not only VideoMAE's capability to identify and cluster recurring shots of Jenny and Forest but also its ability to position these embeddings next to each other, showing its strength in detecting similar spatial contexts and shot patterns. A sample of the shots of the conversation can be found in Fig. 15.

Figure 15: VideMAE recognizes shots in cluster A covering a conversation between Jenni and Forrest

The same applies to cluster B, where we have two scenes in which the same character (Lt Dan) sits near water. We see that the conversations can be separated but are closely located. This shows that the embedder can find the same characters and can still find details such as clothes and other aspects that make these conversations different.

Figure 16: VideMAE shows with cluster B the ability to identify character Lt. Dan in separate scenes

Having observed that the embedder can recognize characters and settings, our next focus is identifying actions. Cluster C predominantly consists of clips from Scene 2, precisely the moment in Forrest Gump where Forrest discovers his ability to run and continues running. This is illustrated in Fig. 17. We observe that certain shots are positioned closely in the 2-dimensional space despite not having a high pixel-level similarity. This suggests that the embedding space captures higher-level semantic actions, in this case, 'running,' rather than just low-level visual similarities like color or texture.

Figure 17: VideoMAE recognizes Forest running in multiple shots for cluster C

6.3.2 Audio Features

While we see some clustering in Fig. 18 for *Katniss* (shots 2,3,4,8,9), it seems that it is much more plausible to assume that these shots are closely located because the pitch level and volume of the conversation are closely located.
We see an indication that between shots 1 and 10, the embeddings frequently return to similar regions, suggesting some level of similarity between these shots. After shot 10, however, the embeddings no longer return to the same areas, indicating a shift in the audio content.
In Fig. 18, shots 1-12 are marked in red, representing the conversation between Primrose and Katniss, while the other shots, mainly consisting of silence, are marked in green. While there is some indication that our hypothesis about the embeddings capturing vocal characteristics might be true, the loss of information during the dimensionality reduction process makes it difficult to draw definitive conclusions for this case.

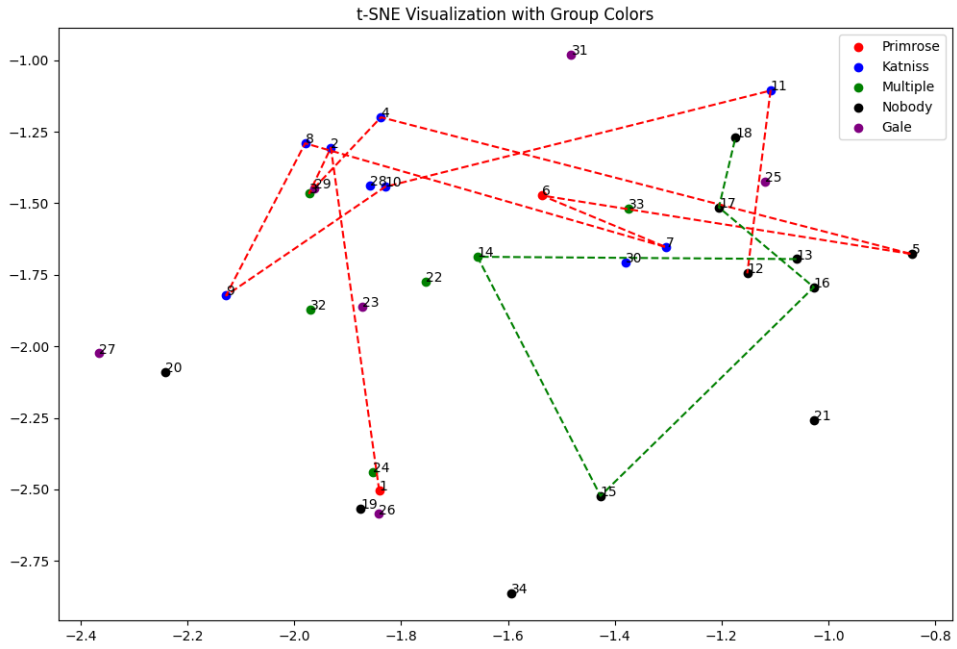
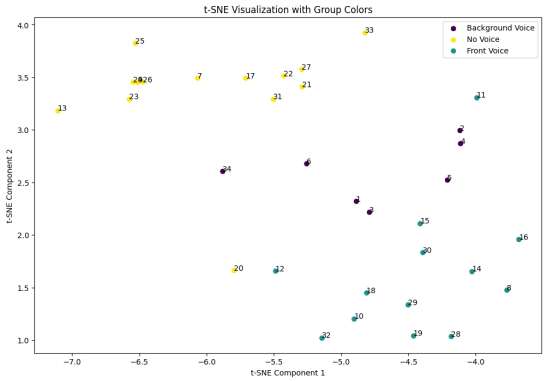


Figure 18: Scene Conversation Hunger Games (2012) Scene 2/10 embeddings on vocals

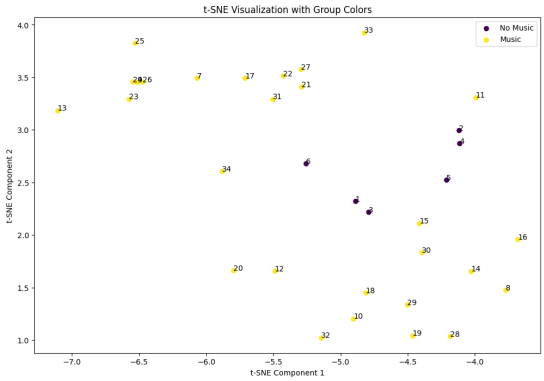
1. *Primrose: Just try to win, maybe you can*
2. *Katniss: "Of course .... and maybe I can"*

3. *Katniss: "I'm smart, you know ...." Primrose: "You can hunt"*
4. *Katniss: "Exactly....."*
5. **Silence**
6. *Primrose: "To protect you"*
7. *Katniss: "Thank you .... *Silence* Katniss: "You can't tune out again"*
8. *Katniss: "You can't not like when dad died"*
9. *Katniss: "I won't be there anymore. You're all she has, no matter what you feel!*
10. *Katniss: "You have to be there for her, do you understand"*
11. **Silence*, Katniss:"Don't Cry, *silence**
12. *Katniss: " .. Don't cry don't ...*
    —— END OF CONVERSATION ———
13. **Silence**
14. **Door Opens* Other man : "It's time" *Panic* Katniss: "Prim it's okay, prim it's okay" Primrose: *Screaming* "No, no, nooo!"*
15. - 18.**Silence**

From Figs. 19a and 19, we can infer that the embeddings produced by Wav2Vec 2 carry meaningful distinctions between different types of audio content. The clustering observed in both figures indicates that the model can encode various auditory features, such as vocals, the distinction between foreground and background voices, and the presence of music—into its embeddings. We can cluster clearly into distinct groups by splitting the sound into accompaniment and vocals. For example, the "No Voice" group from the vocals and the "Music" group from the accompaniment could suggest that these segments represent the introduction or outro of a scene, often accompanied by music, potentially indicating that a scene change is imminent or has just occurred.



(a) accompaniment can identify the location of the voices, for scene 12/12 of Hunger Games



(b) Accompaniment Music positional for music in audio, for scene 12/12 hunger games

Figure 19: Accompaniment analysis for scene 12/12 of Hunger Games

The above plots using t-SNE of VideoMAE and Wav2Vec indicate that the embeddings carry some information about every shot, justifying the current approach of using pre-trained embeddings efficiently. Moreover, separating audio components into vocals and accompaniment using Spleeter, followed by their embedding via Wav2Vec, allows for richer, more distinct representations that capture essential auditory nuances critical for tasks that require similar capabilities as scene transition detection in media.

6.4 Multimodal Fusion for In-Domain Scene Change Detection

The following section presents the results of our experiments designed to address the research questions outlined in Sect. 5.2.4:
Q4: Which model, recurrent or non-recurrent, is more effective for feature-based learning to predict scene changes?
Q5: Does a multimodal fusion approach enhance performance compared to using audio and video separately?

Before we test for different types of recurrent and non-recurrent models, we start by showing the training process for a baseline bidirectional Gated Recurrent Unit (Bi-GRU) model in Tab. 20. For the three models, using video, audio, and fusion, the training loss in Fig. 20 shows a general downward trend, indicating that the models based on the different modalities are learning the patterns in training data over time. The validation losses also initially show a general downward trend, which suggests that the models' learned patterns from the training data can also be applied to unseen data, showing the ability to generalize.
Focusing on the audio model, this model records the lowest validation loss of 0.8625 after 96 steps. The video model's validation loss reaches its lowest value at step 101 with a value of 0.7706. After this, the validation loss increases significantly, indicating that the model is prone to overfitting. The fusion model reaches its lowest loss after 84 steps with a loss of 0.6412, followed by similar overfitting behavior. As mentioned before, these scores do not guarantee good performance in the context of our test dataset due to the bias caused by concatenating scenes that do not follow each other consecutively. Nonetheless, these learning curves show the ability of the model to learn from the complex data, as evidenced by the decreasing validation losses in the first 100 steps. In evaluating different types of recurrent model architectures for capturing time dependencies

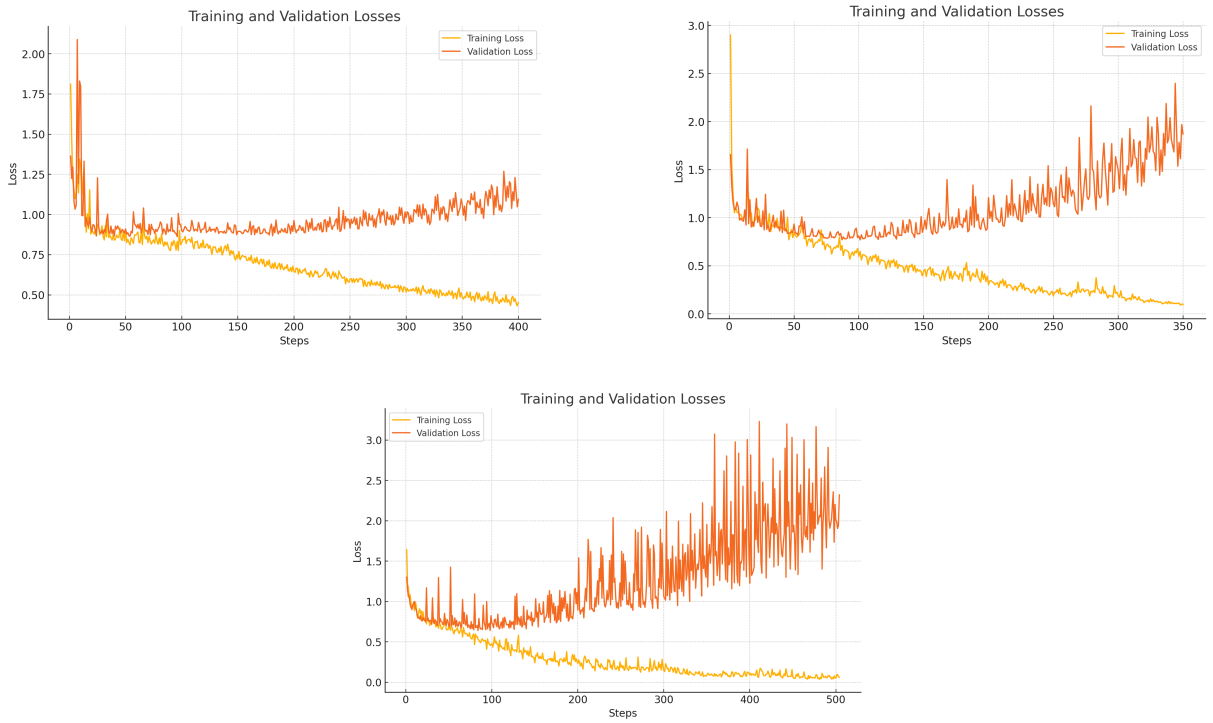


Figure 20: Learning Curves lr 0.001, hidden size 64 and 1 layer for Different Modalities: (a) Audio, (b) Video, (c) Fusion

against the baseline model, the recurrent models outperform the baseline, non-recurrent model, highlighting their superiority in handling sequential data, as shown in Tab. 11. The GRU model, in particular, shows the lowest validation losses across all tested modalities: audio, video, and fusion, emphasizing its effectiveness in integrating temporal information. With a validation loss of 0.6418 in the fusion modality, the GRU model demonstrates a robust capability to synergize audio and video data efficiently. In contrast, the LSTM model, while outperforming the non-recurrent approach, registers slightly higher validation losses than the GRU model. This might suggest that the complexities inherent to LSTMs do not offer additional benefits over GRUs in this experimental setup. The Transformer model, employing attention mechanisms, also outperforms the non-recurrent model but does not reach the performance levels of its recurrent counterparts, possibly due to the model configuration and the nature of the datasets not fully leveraging the strengths of the Transformer architecture. This comparative analysis outlines the importance of choosing recurrent models for tasks that rely heavily on understanding temporal dynamics, with GRUs being the most effective in this context.
To explore the impact of sequence length on model performance, we tested a shorter sequence of 25 shots. The results are shown in Tab. 12. The results indicate that all models, including the GRU, LSTM, and Transformer,

Table 11: The validation loss for all models run with 1 model layer, 64 hidden nodes, and a learning rate of $1e-3$, with a sequence length of 100 shots.

Model	Audio	Video	Fusion
	Non-Recurrent		
Feedforward	1.0960	1.5081	1.0171
	Recurrent		
GRU	0.8625	0.7706	0.6418
LSTM	0.9336	0.8806	0.7721
Transformer	1.0079	0.9081	0.8847

experienced higher validation losses with the shorter sequence compared to a more extended sequence of 100 shots. This suggests that while shorter sequences reduce computational complexity, they may compromise the models' ability to capture and utilize temporal dynamics within the data effectively. Moreover, it shows that GRU and LSTM models especially benefit from longer sequences to predict scene changes.

Table 12: The validation loss for different models with a sequence length of 25 shots.

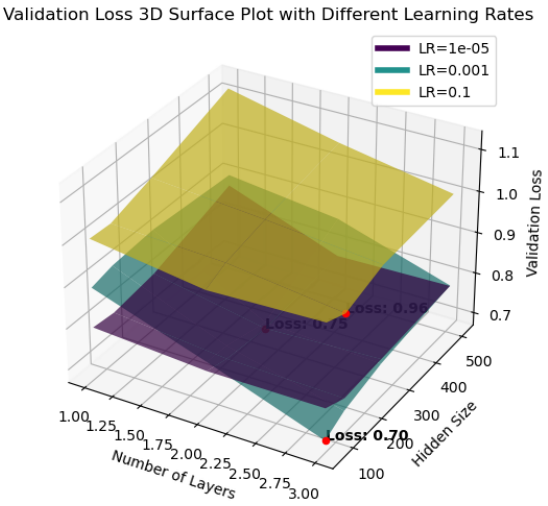
	Audio	Video	Fusion
GRU	0.8902	0.9165	0.9197
LSTM	0.9438	0.9064	0.9040
Transformer	1.1288	0.9687	0.9539

6.5 Hyperparameter Tuning

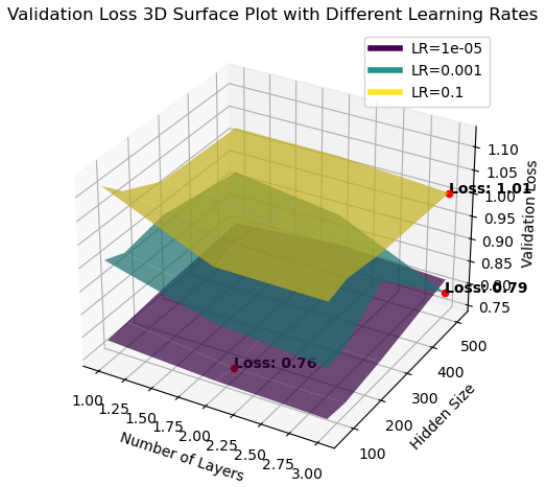
This section provides our results to:
Q6: *What are the optimal hyperparameters for maximizing the performance of this model on the dataset?*.
Sect. 5.2.5 describes the experimental setup of this research question and the parameters we test. The figures below show the validation loss surface plots for the number of layers, hidden size, and learning rate per type of model. The figures show the number of blocks instead of layers, which means that in terms of layers, the x-label must be multiplied by two. In the figures provided (21a for video and 21b for audio), the minimum validation losses achieved are 0.70 and 0.76, respectively, both achieved when trained using multiple layers, indicating that more recurrent layers can enhance model performance. From Fig. 20, we have already seen that it takes more steps for the baseline model to optimize it for the video model. Additionally, the video modality performs better when trained with more layers, indicating that heavier training is necessary to learn the complexity of the video modality than the audio modality. The optimal number of hidden nodes is 128 for the audio model and 64 for the video model. The $1 \cdot 10^{-5}$ and $1 \cdot 10^{-3}$ learning rates dominate the 0.1 learning rate for both modalities. Fig. 21a shows that the video model is optimal for $lr = 1 \cdot 10^{-3}$, and Fig. 21b shows that the audio model is best when trained with $lr = 1 \cdot 10^{-5}$.
Turning to the fusion model, as depicted in Fig. 21c, we refine our parameter search to learning rates of 0.01, $1 \cdot 10^{-3}$, and $1 \cdot 10^{-5}$, excluding 0.1 due to its inferior performance. The fusion model achieves a notably lower minimum validation loss of 0.58, benefiting from a configuration of three layers and 64 nodes at a learning rate of $1 \cdot 10^{-4}$. This configuration underlines the advantage of more layers and fewer nodes in effectively reducing validation loss. This was also the case for the audio and video models. Moreover, the fusion model's performance on the validation set showcases its superior capability to integrate audio and video data as it outperforms the model trained on the modalities without fusion.

6.6 Out-of-Domain Scene Changes

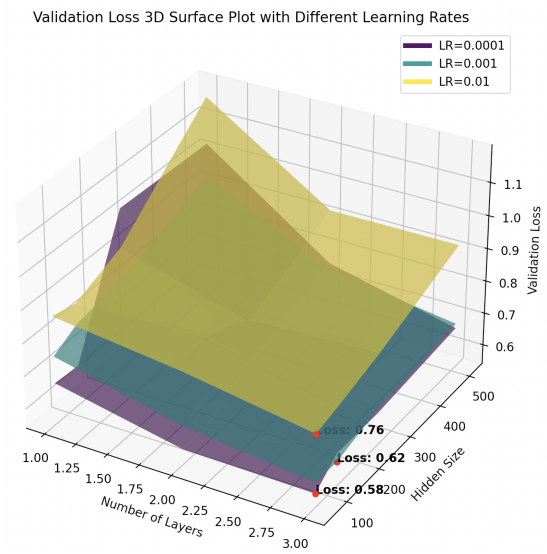
This section presents the results of our experiments designed to address the research question outlined in Sect. 5.2.6:
Q7: Can a model learn out-of-domain scene changes (with a different interpretation of a scene) from little data?



(a) Video Model, the lowest validation score is 0.70 with a learning rate of $1 \cdot 10^{-3}$, three blocks, and 64 nodes



(b) Audio Model, the lowest validation score is 0.76 with a learning rate of $1 \cdot 10^{-5}$, two blocks, and 128 nodes



(c) Fusion model, the lowest validation score is 0.58 with $1 \cdot 10^{-3}$ lr, three blocks, and 64 nodes.

After determining the optimal configurations for every modality-type model, we can test them on our out-of-domain test set to see their ability to adapt to new types of content. We evaluate the model for *Planet Earth* as described in Sect. 5.2.6 using a cross-validation method. The results clearly show that the fusion of audio and video modalities yields the most accurate predictions across *Off-by-n* and *threshold* $= t$ configurations ($t = 0.5, n = 0$ and $t = 0.5, n = 1$). Specifically, for off-by-one scoring (i.e., allowing a one-frame error margin in scene change detection), the fusion model achieves a notable F1-score of 0.432, precision of 0.486, and recall of 0.424. These score differences over the standard F1-score, where no leniency is allowed, imply that the exact moment marking the beginning of a new scene can be subjective to the annotators. Therefore, implementing off-by-one scoring, which introduces a margin to accommodate this subjectivity, may reflect the model's performance more accurately. This adjustment acknowledges the variability in human interpretation of scene transitions, making the evaluations more realistic and relevant.

Table 13: *BBC planet Earth* Cross-validation scores using eight training episodes and two validation episodes

Show	Type	F1-score	Precision	Recall
			t=0.5 n=0	
Planet Earth	Audio	0.153 (0.083)	0.545 (0.272)	0.108 (0.072)
	Video	0.198 (0.032)	0.416 (0.149)	0.130 (0.023)
	Fusion	0.330 (0.090)	0.332 (0.131)	0.304 (0.087)
			t=0.5 n=1	
Planet Earth	Audio	0.188 (0.110)	0.587 (0.240)	0.138 (0.096)
	Video	0.256 (0.055)	0.494 (0.115)	0.178 (0.052)
	Fusion	0.432 (0.097)	0.486 (0.138)	0.424 (0.124)

As anticipated from pre-training data, the fusion model demonstrates superior performance over audio and video models. With precision and recall scores of 0.486 and 0.424, respectively, the model successfully identifies approximately 42.4% of the scene changes out of 48.6% of the changes it predicts.

To enhance accuracy, we suggest adopting a Human-in-the-loop approach and focusing less on penalizing false positives and more on minimizing false negatives. This approach is beneficial as the model learns to detect annotated scene changes but may also identify semantic shifts that an annotator could miss. Based on the model's output, the annotator will have a narrowed search field for all the scene changes. Implementing a prediction threshold allows us to efficiently identify more scene changes, optimizing the annotator's workload. In the context of a Human-in-the-loop approach, this translates to inspecting about 20 scenes to accurately identify ten true scene changes, highlighting the model's cost-effectiveness. We could consider lowering the threshold to enhance the detection rate, which would likely increase the number of scene changes detected at the expense of inspecting more scenes. This adjustment could lead to a more comprehensive but labor-intensive review process. The results of an adjusted threshold are shown in Tab. 14.

The table evaluates scene change detection across different modalities in *Planet Earth*, demonstrating the fusion model's superior performance. We use the Youden J statistic to determine the threshold value for scene classification. This measure is ideal as it balances sensitivity and specificity, adjusting slightly for the difference in the model's strictness based on the results of the validation set. Using the formula:

Table 14: Scene-changes performance metrics BBC Planet Earth using different modality models

Show	Type	Youden-J	AUC	EUCL	AP
Planet Earth	Audio	0.393 (0.056)	0.730 (0.052)	0.445 (0.043)	0.265 (0.011)
	Video	0.396 (0.150)	0.720 (0.090)	0.438 (0.116)	0.256 (0.088)
	Fusion	0.478 (0.091)	0.782 (0.054)	0.372 (0.053)	0.278 (0.090)
			Optimal Youden J Threshold		
Planet Earth	Audio	0.107 (0.007)			
	Video	0.132 (0.014)			
	Fusion	0.156 (0.053)			

The training set had fewer positive samples than the test set. Therefore, relying on a threshold of 0.5 would likely replicate the balance of the samples in the training set.

In Tab. 15, we calculate the F1-scores, incorporating precision and recall measures, using the threshold Youden's J statistic. The variable t represents the threshold used, corresponding to the values shown in Tab. 15. For variable n, we determine the permissible number of shots by which a prediction may deviate, referred to as the off-by-n F1-score. We observe a significant difference in performance between the off-by-0 and off-by-1 scores. While a variance is expected, the substantial difference shows the challenge in precisely identifying the exact shot that marks the scene change.

This Youden J threshold, or the optimal cut point estimate, decreases the F1-score and Precision scores; however, it significantly improves the recall. With the fusion model, an annotator can now find 78 % of the scene changes, where 31 % of the inspected scene changes are true. Using our Human-in-the-loop approach, we prefer false positives over false negatives, as we want to recover the scene instead of missing it entirely.

Table 15: Scene-changes performance metrics BBC planet Earth using different modality models

Show	Type	F1-score	Precision	Recall
			t=Opt n=0	
Planet Earth	Audio	0.235 (0.045)	0.150 (0.042)	0.593 (0.083)
	Video	0.258 (0.081)	0.162 (0.057)	0.644 (0.103)
	Fusion	0.288 (0.074)	0.194 (0.084)	0.630 (0.153)
			t=Opt n=1	
Planet Earth	Audio	0.315 (0.070)	0.265 (0.011)	0.795 (0.042)
	Video	0.318 (0.075)	0.256 (0.088)	0.772 (0.072)
	Fusion	0.360 (0.108)	0.318 (0.103)	0.778 (0.153)

The predictions on the test set, including episode 4 from the cross-validation, are displayed in Fig. 22. The red lines represent the scene changes identified by one annotator, while the blue dots indicate the predictions made by the fine-tuned models. Although some peaks correspond to actual scene changes, the model occasionally misses specific scene transitions and falsely predicts some shots as the scene changes. The type of content is vastly different from the training set, and we may have trouble identifying the changes.

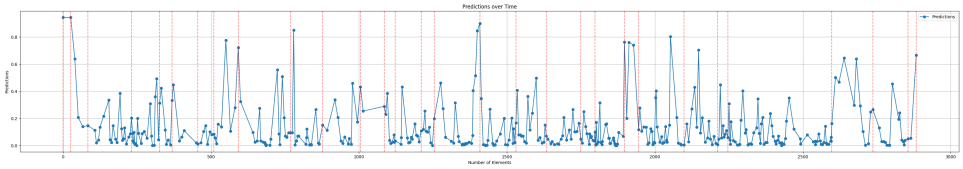


Figure 22: Plotted predictions on BBC Planet Earth test episode 4 from cross-validation, with red lines actual annotated changes and blue the predictions

For *Survivor* and *Hunted*, we only test for the fusion model. The model scores on the respective shows are 0.232 and 0.280 off-by-1 F1-scores, respectively, when taking a threshold value of 0.5 for predictions, as seen in Tab. 16. We see, from Tab. 17 that for the shows, the model scores 0.838 and 0.874 AUC on average, respectively, indicating that the model is significantly better at predicting a scene change than simply predicting the majority class, in this case, the non-scene change class.

Table 16: Cross-validation statistics, F1, precision and recall values for threshold value 0.5 (t) and off-by-n frames (n) for TV shows *Survivor* and *Hunted*

Show	F1 Score (Std. Dev.)	Recall (Std. Dev.)	Precision (Std. Dev.)
		t=0.5, n=0	
Survivor	0.130 (0.059)	0.082 (0.040)	0.316 (0.140)
Hunted	0.208 (0.039)	0.146 (0.045)	0.442 (0.133)
		t=0.5, n=1	
Survivor	0.232 (0.070)	0.154 (0.050)	0.482 (0.136)
Hunted	0.280 (0.035)	0.198 (0.043)	0.526 (0.144)

Table 17: Cross-validation statistics, AUC, Youden's J, threshold value for TV shows *Survivor* and *Hunted*

Show	Youden J	AUC	Eucl.	AP
Survivor	0.586 (0.092)	0.838 (0.035)	0.310 (0.064)	0.176 (0.011)
Hunted	0.676 (0.090)	0.874 (0.050)	0.248 (0.056)	0.268 (0.102)
	Optimal Youden J Threshold			
Survivor	0.014 (0.004)			
Hunted	0.254 (0.003)			

Again, we want to consider changing the threshold to adapt to a different notion of semantic change to make predictions. We see that especially the recall is relatively poor; when looking at Figs. 23 and 24, we see that while we observe some peaks, the predictions rarely pass the 0.5 thresholds, which is why we have

poor recall performance. We observe that we can increase the recall but at the cost of precision when lowering the threshold to the Youden J optimal threshold. The off-by-1 F1 scores show a significant improvement for the *Hunted* dataset, rising to 0.398. However, the *Survivor* dataset performs notably worse, scoring 0.130. Youden's J statistic only considers the TPR and FPR, which aligns with our goal. We aim to overestimate the scene change class, allowing a Human-in-the-loop approach to analyze the positively predicted samples further. However, this goes at the expense of lower off-by-1 F1 scores. Fig. 30 also shows that the ROC curves steeply increase for lower FPR and flatten out for higher FPR, resulting in a low optimal cut point estimate.

Table 18: Cross-validation F1 scores at Youden's J point for TV shows *Survivor* and *Hunted*

Show	**F1 Score**	**Recall**	**Precision**
		t=Opt, n=0	
Survivor	0.112 (0.019)	0.752 (0.113)	0.060 (0.010)
Hunted	0.268 (0.085)	0.238 (0.071)	0.318 (0.126)
		t=Opt, n=1	
Survivor	0.130 (0.025)	0.854 (0.059)	0.072 (0.013)
Hunted	0.398 (0.108)	0.372 (0.090)	0.442 (0.142)

As seen in Figs. 23 and 24, there are noticeable peaks around the scene boundaries, suggesting that the model has successfully learned certain aspects of scene transitions in the TV shows. To qualitatively assess the results, we refer to Sect. 7.

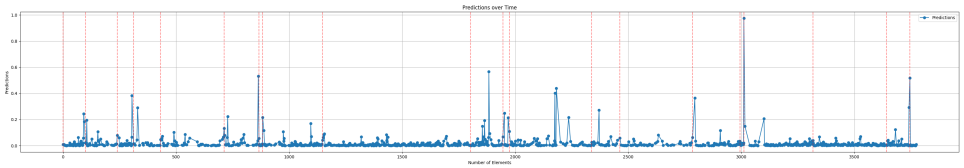


Figure 23: Plotted predictions on *Hunted* test episode 7 from cross-validation, with red lines actual annotated changes and blue the prediction

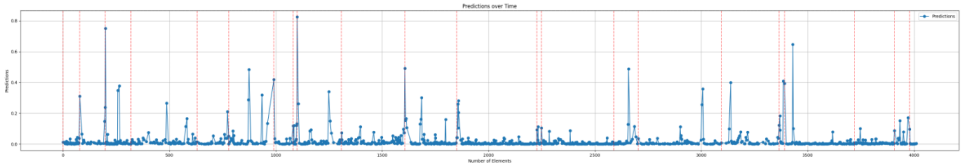


Figure 24: Plotted predictions on *Survivor* test episode 1 from cross-validation, with red lines actual annotated changes and blue the predictions

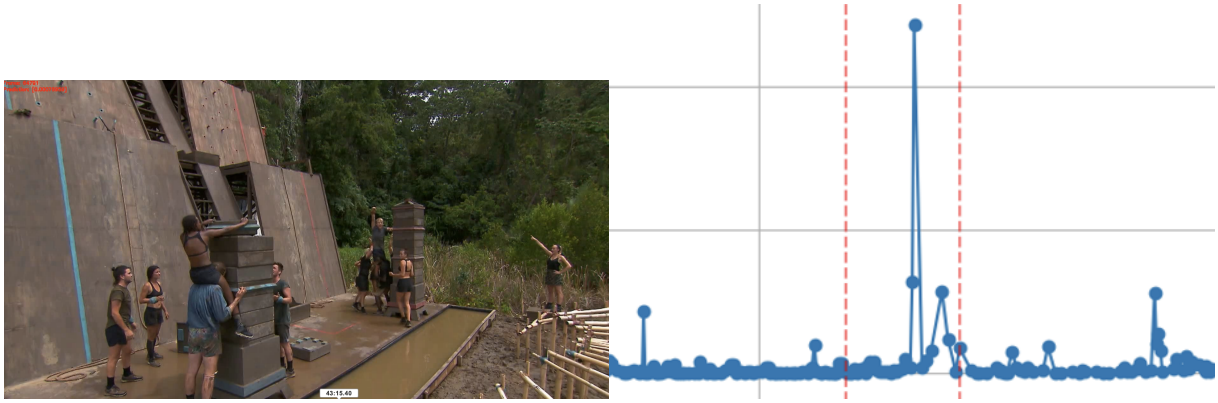
7 Discussion and Limitations

Using large datasets is crucial for pre-training scene segmentation models, yet limitations exist that cause problems in finding a sufficient dataset. In our case, CMD [8] contains a typical constraint caused by legal issues: only key scenes are released instead of full-length movies.

We face significant limitations that could impact its effectiveness for training models in scene segmentation. Although the scenes in the dataset are complete segments from the movies, they are not presented in their sequential order. As a result, we need to relax the assumption that scenes are blocks of events dependent on their context and instead consider them as independent events. This non-sequential arrangement eliminates the opportunity to learn about scene transition indicators, a crucial aspect of understanding continuous narrative flows in movies. The absence of sequential data can introduce biases and potentially lead to overestimating the training performance.

Comparatively, other datasets like BBC [9], though free from some of the legal restrictions affecting MovieNet and CMD, suffer from their critical limitation: size. These datasets are too small for effective pre-training of

classification models, which calls for large and varied datasets to develop robust and generalizable AI systems. This problem in dataset applicability highlights the ongoing challenge in the field: balancing the availability of legally usable data with the need for large-scale, diverse datasets for advanced machine learning tasks.
The fine-tuning dataset contains semantic changes in the TV show that the annotator fails to capture, but the model does. These semantic changes might include shifts in location, characters, or visual context that the model recognizes as significant, even though the annotator does not explicitly label them as scene changes. Whether or not these are actual scene changes depends on the annotator's interpretation of what constitutes a scene. This discrepancy introduces a layer of uncertainty, making the prediction results less reliable since the labels do not represent absolute ground truth. Regardless, recognizing these changes can still be valuable for practical use in identifying how behavior changes due to this change.
As a result, the performance metrics do not always capture the model's performance, as reflected by the scores in Table 16. In *Survivor*, scenes are manually annotated, and the transitions are based more on narrative logic than on changes in visual or auditory elements. For example, Figure 25a shows an annotated scene transition shot, with the corresponding model predictions plotted in Figure 25b. The shots depict the end of a game, which is a significant moment in the storyline. However, when the game ends, there are no changes in the characters, location, or audio (music or vocals).
The model does predict a scene change when the location changes, which is shortly after the ending of a game. This indicates that the model is focused on detecting changes in elements, which, as discussed in Sect. 4.3, is a fundamental aspect of how, according to generic rules, scene transitions are commonly identified. Arguably, the short discussion after the game, before switching locations, can be considered part of the game; therefore, the model's misprediction might not tell the whole story.This example may highlight the ambiguity in the definition of scenes. However, we still see some good performance when further assessing the predictions.



(a) Scene change as annotated

(b) Predictions in blue and scene changes in red

Figure 25: Comparison between annotated scene changes and model predictions.

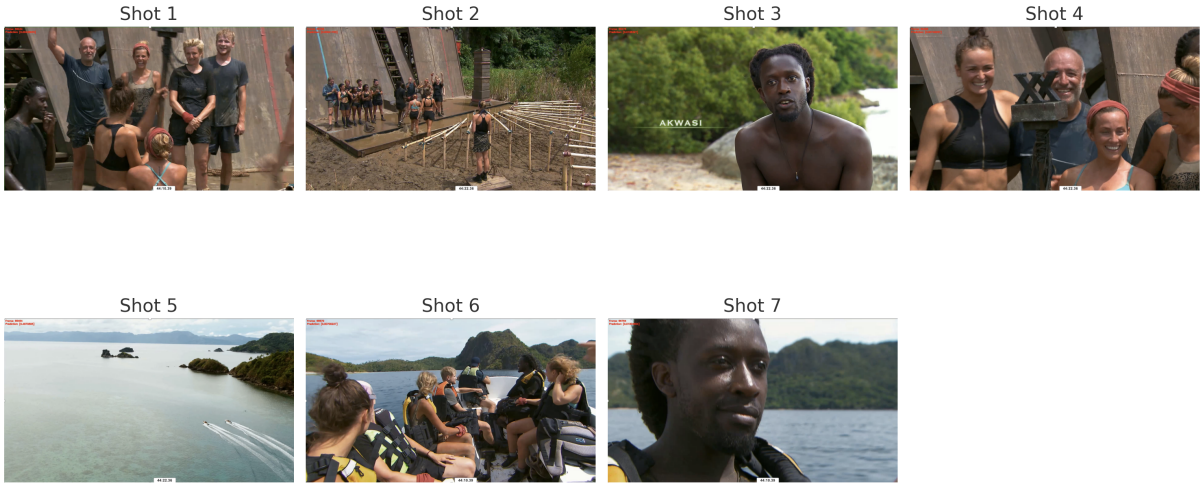


Figure 26: Shot 5 is predicted as a scene change by the model, whereas the annotator does not label it as such.

We refer to Figure 27 to further evaluate the model's behavior. In the middle of the graph, we see two scene changes (red lines) followed by a blue peak. The first scene change marks a preview of the upcoming content before cutting to a break, after which the show resumes. The model predicts both as the scene changes, as indicated by the relative change in score compared to the surrounding shots. However, the high peak following these two annotated scene changes is predicted as a scene change but is not annotated as such.
When we examine the show more closely, we see that after returning from the break, there is a brief segment where the characters are transported by boat to a new location, where a critical event, *De Eilandraad*, begins. Figure 28 depicts this arrival and the segment's start. While the model's prediction isn't entirely incorrect, it again identifies a shift of elements, interpreting the boat ride to the location as a separate scene. Whether these two parts are considered one or two scenes can be subjective and altered according to one's preference. This demonstrates how a human-in-the-loop approach could help fine-tune such preferences and improve scene change predictions. Moreover, it shows how the results of Sect. 6 might be misleading for humanly annotated shows *Hunted* and *Survivor*.

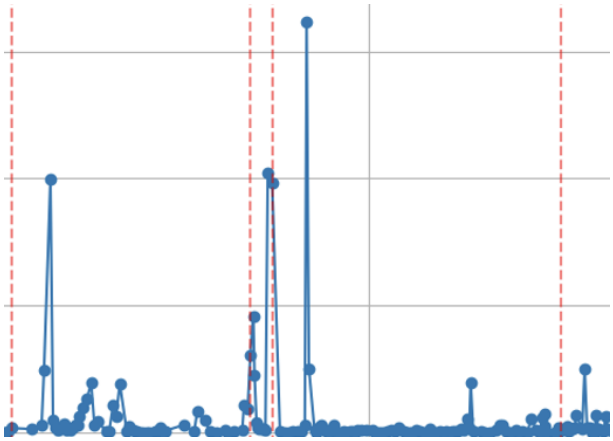


Figure 27: Predictions in blue and scene changes in red

Figure 28: Row 1 is predicted to be scene 1 and row 2 is predicted to be scene 2, scene 2 is known as *De Eilandraad* which is a crucial element of the show

8 Conclusion

In conclusion, while our framework demonstrates potential by capturing changes in the narrative and showing semantic understanding, building a fully autonomous scene change detection model remains challenging due to the inherent ambiguity in scene interpretation. However, our current model can significantly reduce time by interpreting visualizations, such as Figures 23 and 24, making it effective in a human-in-the-loop approach. We considered shots to be fundamental scene components and applied the PySceneDetect model to our data. This model, despite its confusing name, detects shot transitions. The results demonstrated that most of the shot changes could be successfully identified. The results from audio source separation (Sect. 6.1) and feature extraction (Sect. 6.3) suggest that these steps are promising for the development of a fully autonomous scene change detection model. We demonstrate that utilizing a recurrent model allows us to effectively capture the temporal dependencies between these shots, allowing the use of contextual information to interpret the current shot more accurately. In our test, the GRU architecture provided the best validation results. Using multiple bidirectional layers and more nodes improves the model's performance on our validation set (Sect. 6.5). The fusion model outperforms audio and video models, emphasizing the value of integrating features from multiple modalities, as the results in Sect. 6.4. This multimodal approach could be further improved by incorporating a text modality, leveraging state-of-the-art transcription and *NLP (Natural Language Processing)* models to capture the narrative structure more effectively.
While our results on the out-of-domain test set do not indicate a flawless model, we discuss in Sect. 7 that evaluating the performance metrics might not accurately be able to assess its true capabilities to understand film/TV semantics and its predictions on scene changes. Manually reviewing the predictions indicates that the model has some understanding of narrative shifts, showing the promise of the current approach in a step to automate the detection of scenes fully.

9 Future Work

The research provides a robust framework for detecting scene transitions in audiovisual content using advanced neural network architectures. Despite achieving some good results, several areas remain where future research could further enhance model performance and applicability. Answering our research (sub-)questions can provide us with important information to build an intelligent detection model. While this study focuses on scene change detection, its findings can also contribute to other analytical models in films. Other tasks models also involve multiple modalities, which suggests that the steps in this approach could also be relevant to other tasks. Exploring such applications could expand the use of this framework and further validate the approach.
As shown in Figure 2, the text modality was not included in our analysis. However, with recent advancements in transcription models, future research could benefit from incorporating transcriptions as a third modality. This addition would enhance narrative comprehension by allowing advanced NLP models to detect shifts in the storyline, providing more essential features beyond just audio and video features.
Future studies could explore experimenting with different embedding layer outputs for wav2vec 2 and video-MAE. Due to the challenges in accurately quantifying the true performance of embeddings, we have utilized

the output from the penultimate layer. However, this may not be the optimal choice. Further research should consider evaluating various layers to determine which provides the most effective embeddings for their specific applications despite the difficulty in measuring true quality.

Another area of exploration could involv integrating convolutional neural networks (CNNs) with our current GRU-based models to enhance feature extraction capabilities, particularly for complex scene transitions involving significant visual changes. Moreover, if the computational capacity allows for it, the model can be trained from end to end, removing the need for a pre-trained embedder. The pre-trained embedders in this research are not specifically trained for this task, which leaves room for optimization by training them on our specific dataset and task as well.

With the rapid advancement of machine learning hardware and software, future studies should consider adapting the current frameworks to take full advantage of real-time processing capabilities, potentially enabling live scene transition detection in streaming media.

References

[1] AI in TV Production 2023: Beyond the Hype, the Quiet Revolution Continues – K7 Media, 10 2023.

[2] P. Aigrain and P. Joly. The automatic real-time analysis of film editing and transition effects and its applications. *Computers & Graphics*, 18(1):93–103, 1994.

[3] M. Akewar. A scene boundary detection approach using audio features. *Authorea Preprints*, 2023.

[4] A. Akutsu, Y. Tonomura, H. Hashimoto, and Y. Ohba. Video indexing using motion vectors. In *Visual Communications and Image Processing'92*, volume 1818, pages 1522–1530. SPIE, 1992.

[5] E. Ardizzone, G. A. Gioiello, M. La Cascia, and D. Molinelli. A real-time neural approach to scene cut detection. In *SPIE Storage and Retrieval for Image and Video Databases IV*, 1996.

[6] H. T. Avneesh Saluja, Andy Yao. Detecting scene changes in audiovisual content - Netflix TechBlog. 7 2023.

[7] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[8] M. Bain, A. Nagrani, A. Brown, and A. Zisserman. Condensed movies: Story based retrieval with contextual embeddings. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[9] L. Baraldi, C. Grana, and R. Cucchiara. A deep siamese network for scene detection in broadcast videos. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1199–1202, 2015.

[10] B. J. Billingsley. Industry Insights: How AI is impacting broadcast production workflows, 7 2024.

[11] D. Bose, R. Hebbar, K. Somandepalli, H. Zhang, Y. Cui, K. Cole-McLaughlin, H. Wang, and S. Narayanan. Movieclip: Visual scene recognition in movies. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2083–2092, 2023.

[12] J. Bunting. How to Write a Scene: The Definitive Guide to Scene Structure, 12 2023.

[13] J. Careless. The Impact of AI/ML on TV Production and Playout, 4 2020.

[14] B. Castellano. Home - PySceneDetect.

[15] S. Chen, X. Nie, D. Fan, D. Zhang, V. Bhat, and R. Hamid. Shot contrastive self-supervised learning for scene boundary detection. In *CVPR 2021*, 2021.

[16] S. Chen, X. Nie, D. Fan, D. Zhang, V. Bhat, and R. Hamid. Automatically identifying scene boundaries in movies and TV shows - Amazon Science, 3 2024.

[17] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[18] J. S. Chung. Naver at activitynet challenge 2019–task b active speaker detection (ava). *arXiv preprint arXiv:1906.10555*, 2019.

[19] M. Del Fabro and L. Böszörmenyi. State-of-the-art and future challenges in video scene detection: a survey. *Multimedia systems*, 19:427–454, 2013.

[20] J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[21] P. K. Diederik. Adam: A method for stochastic optimization. *(No Title)*, 2014.

[22] O. Fatemi, S. Zhang, and S. Panchanathan. Optical flow based model for scene cut detection. In *Proceedings of 1996 Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 470–473. IEEE, 1996.

[23] K. Gadzicki, R. Khamsehashari, and C. Zetzsche. Early vs late fusion in multimodal convolutional neural networks. In *2020 IEEE 23rd international conference on information fusion (FUSION)*, pages 1–6. IEEE, 2020.

[24] A. Hampapur, T. Weymouth, and R. Jain. Digital video segmentation. In *Proceedings of the second ACM international conference on Multimedia*, pages 357–364, 1994.

[25] A. Hanjalic, R. L. Lagendijk, and J. Biemond. Automated high-level movie segmentation for advanced video-retrieval systems. *IEEE transactions on circuits and systems for video technology*, 9(4):580–588, 1999.

[26] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam. Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, 5(50):2154, 2020.

[27] Q. Huang, Y. Xiong, A. Rao, J. Wang, and D. Lin. Movienet: A holistic dataset for movie understanding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 709–727. Springer, 2020.

[28] M. M. Islam, M. Hasan, K. S. Athrey, T. Braskich, and G. Bertasius. Efficient movie scene detection using state-space transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18749–18758, 2023.

[29] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde. Singing voice separation with deep u-net convolutional networks, 2017.

[30] J. R. Kender and B.-L. Yeo. Video scene segmentation via continuous video coherence. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 367–373. IEEE, 1998.

[31] K. R. Liu. 2007 ieee international conference on acoustics, speech, and signal processing (icassp). *Signal Processing (ICASSP)*, 2007.

[32] J. Mun, M. Shin, G. Han, S. Lee, S. Ha, J. Lee, and E.-S. Kim. Boundary-aware self-supervised learning for video scene segmentation. *arXiv preprint arXiv:2201.05277*, 2022.

[33] N. Name). ALICE High Performance Computing facility.

[34] K. Otsuji, Y. Tonomura, and Y. Ohba. Video browsing using brightness data. In *Visual Communications and Image Processing'91: Image Processing*, volume 1606, pages 980–989. SPIE, 1991.

[35] A. Radford. Improving language understanding by generative pre-training. 2018.

[36] Z. Rasheed and M. Shah. Scene detection in hollywood movies and tv shows. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–343. IEEE, 2003.

[37] D. Rotman, D. Porat, and G. Ashour. Robust and efficient video scene detection using optimal sequential grouping. In *2016 IEEE international symposium on multimedia (ISM)*, pages 275–280. IEEE, 2016.

[38] N. Sadoughi, X. Li, A. Vajpayee, D. Fan, B. Shuai, H. Santos-Villalobos, V. Bhat, and R. Mv. Mega: Multimodal alignment aggregation and distillation for cinematic video segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23331–23340, 2023.

[39] S. Schneider, A. Baevski, R. Collobert, and M. Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

[40] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[41] C. G. Snoek, M. Worring, and A. W. Smeulders. Early versus late fusion in semantic video analysis. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 399–402, 2005.

[42] H. Tantai. Use weighted loss function to solve imbalanced data classification problems. 2 2023.

[43] Z. Tong, Y. Song, J. Wang, and L. Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.

[44] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[46] J. Vendrig and M. Worring. Systematic evaluation of logical story unit segmentation. *IEEE Transactions on Multimedia*, 4(4):492–499, 2002.

[47] X. Wei, Z. Shi, T. Zhang, X. Yu, and L. Xiao. Multimodal high-order relation transformer for scene boundary detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22081–22090, 2023.

[48] H. Wu, K. Chen, Y. Luo, R. Qiao, B. Ren, H. Liu, W. Xie, and L. Shen. Scene consistency representation learning for video scene segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14021–14030, June 2022.

[49] M. Yeung, B.-L. Yeo, and B. Liu. Segmentation of video by clustering and graph analysis. *Computer vision and image understanding*, 71(1):94–109, 1998.

[50] H. Zhang, C. Y. Low, and S. W. Smoliar. Video parsing and browsing using compressed data. *Multimedia tools and applications*, 1:89–111, 1995.

A Appendix: Additional Results

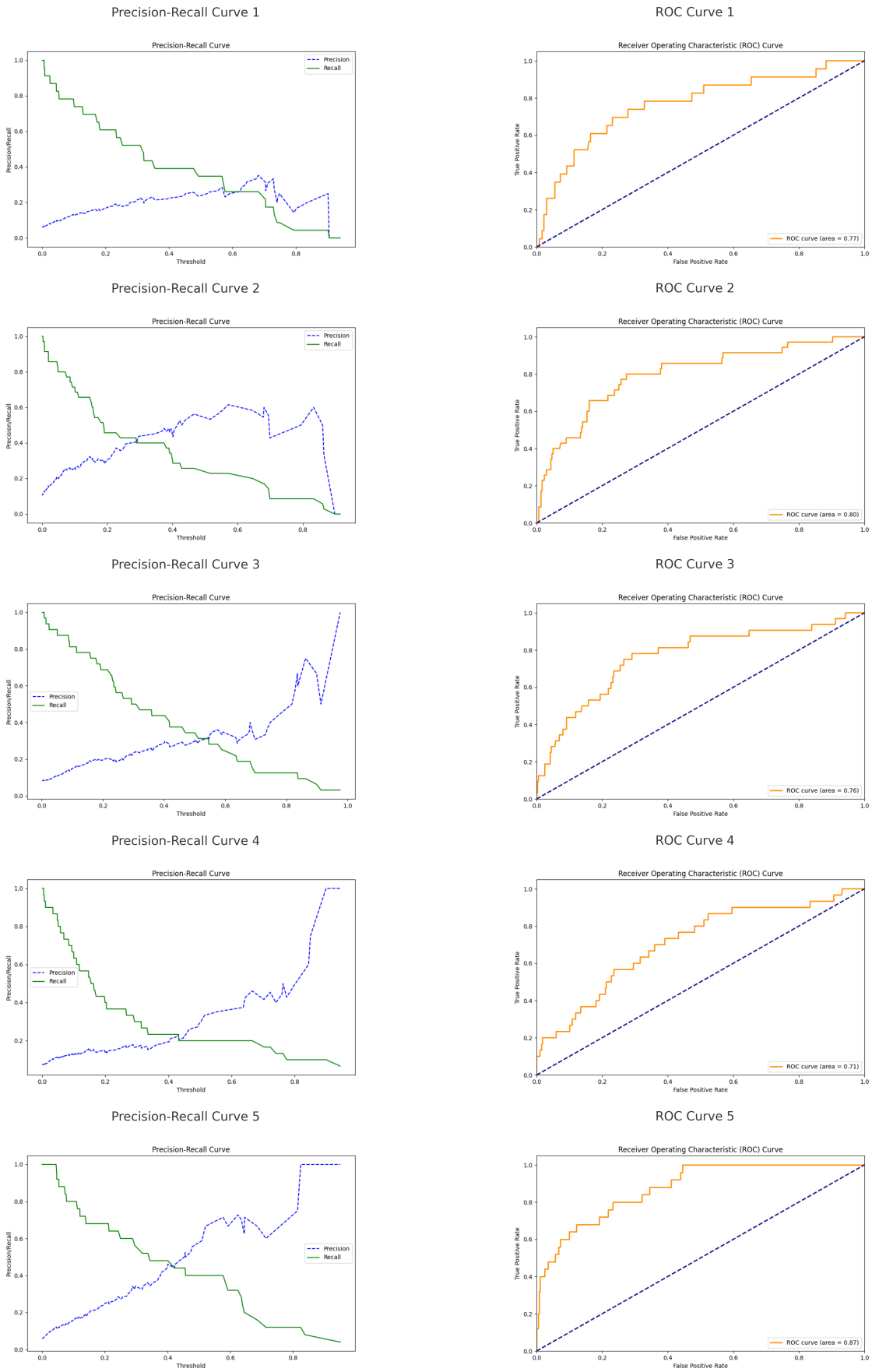


Figure 29: Cross validation results on test episodes of BBC Planet Earth episodes 1-5

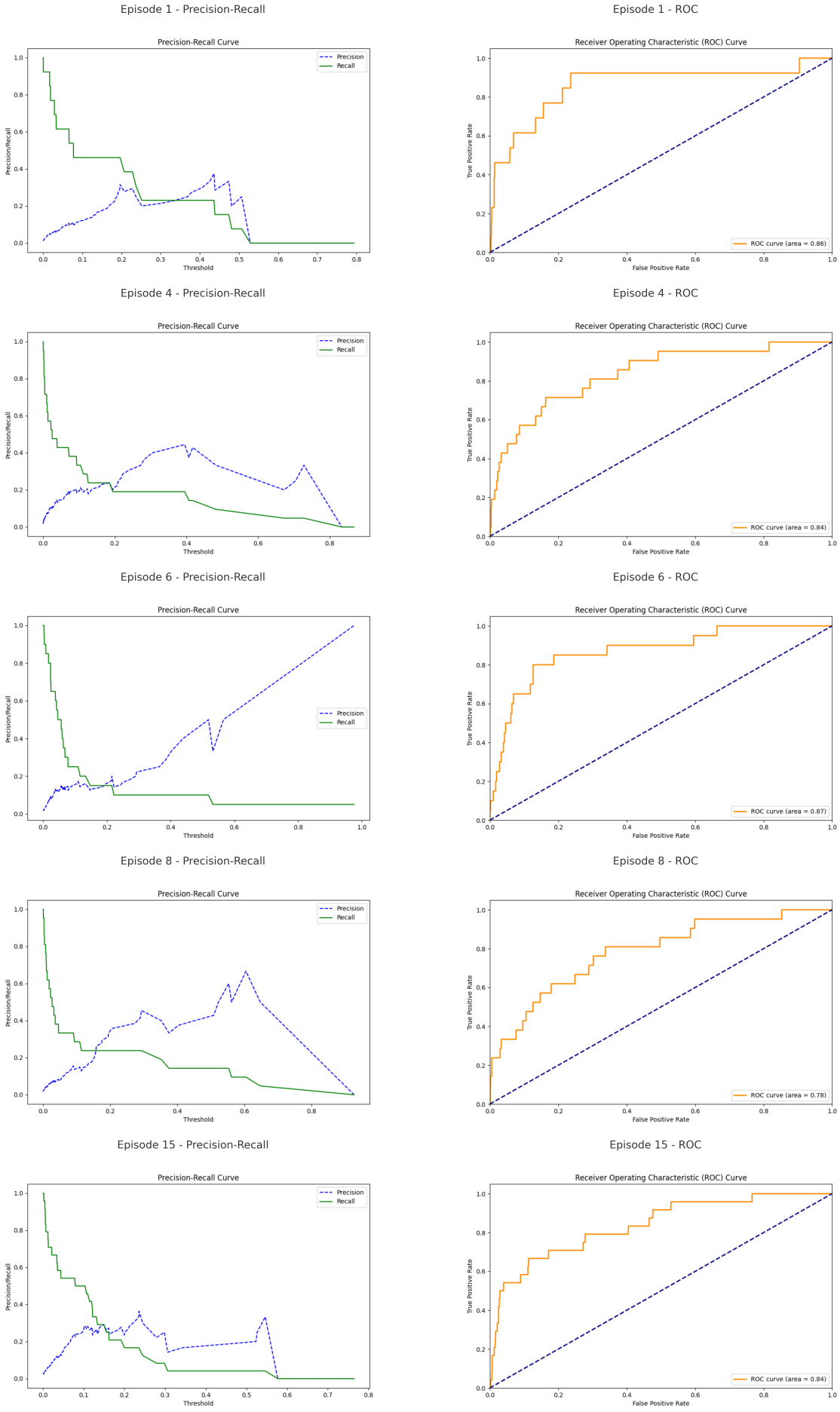


Figure 30: Cross-Validation curves on test episodes of Survivor

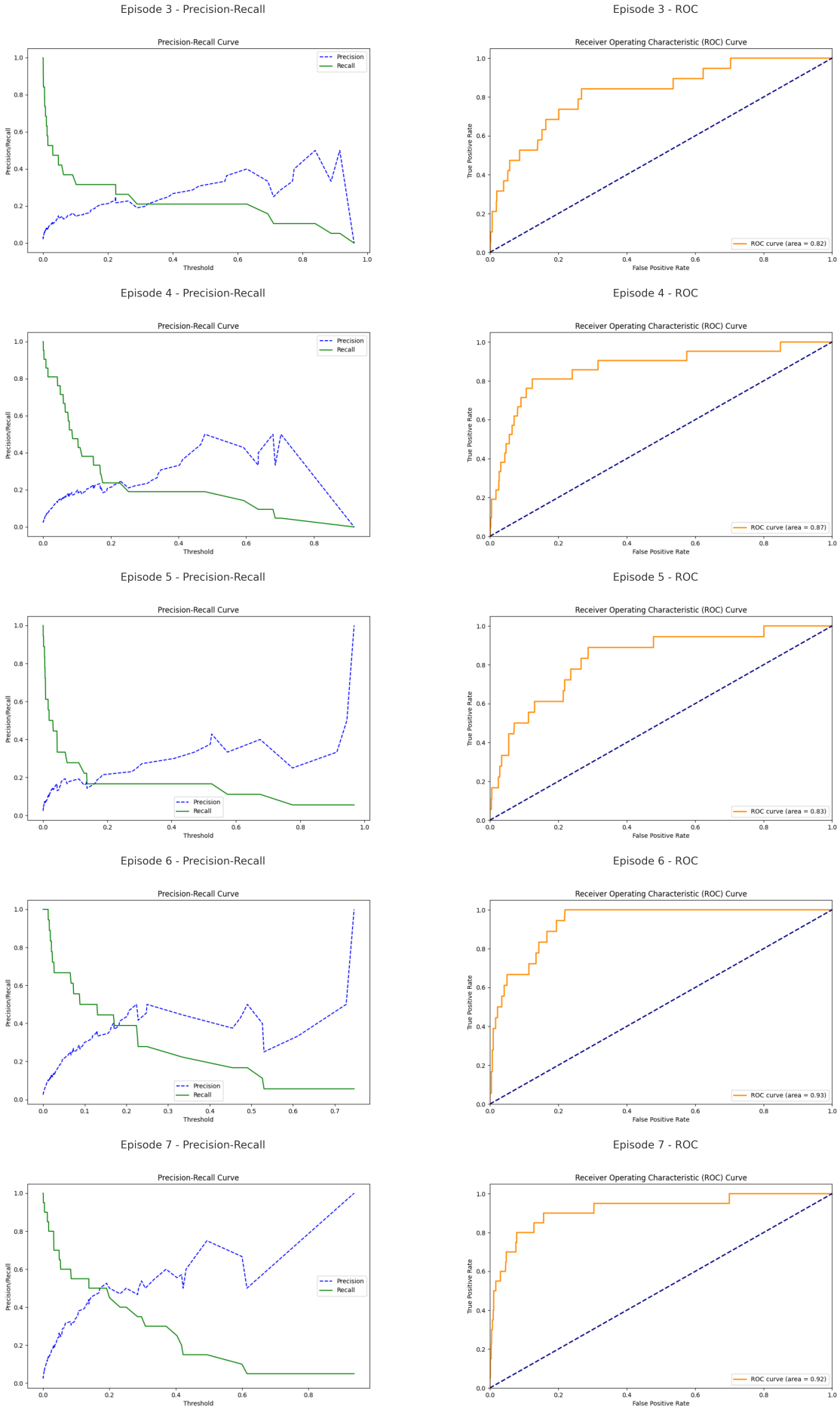


Figure 31: Cross-Validation curves on test episodes of Hunted