University of Genoa

Department of Computer Science, Bioengineering,

Robotics and System Engineering (DIBRIS)

Master of Science Thesis in Computer Engineering

An Efficient Approach to Video Scene Detection

Thesis supervisors:

Prof.Ing. Gianni Viardo Vercelli

Prof.Ing. Massimo Paolucci

Dott. Roberto Ronco

Candidates:

Igor Pio Bianchi

Alessandro Birago

Academic Year

2018-2019

Abstract

An efficient Video Scene Detection (VSD) pipeline is the key passage to face the automatic production of accessible Audio Descriptions (ADs) of movies and TV series. The workflow of the AD process is complex and still manually done, despite the relevance and the current impact for media content producers. The goal of this work is to study the main aspects of this task, starting from the recent approach proposed by Rotman et al. [48]. We explored each phase of the implementation pipeline, finding possible improvements in shot detection and dynamic programming areas.
We propose a reformulation of the dynamic programming algorithm for scene boundary detection that lead to the creation of two alternative methods. The results show that these approaches outperform Rotman et al. in term of cost function and mean accuracy, yielding a reduction in execution time.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AD audio description

AAD automatic audio description

CED color entropy difference method

CEM color entropy max method

CL clustering frame selection method

CNN convolutional neural network

DP dynamic programming

GED gray-scale entropy difference method

GEM maximum gray-scale entropy method

GT ground truth division

HH HSV histogram method

HSV hue, saturation, value

MF middle frame method

R18 Rotman et al. dynamic programming algorithm

RGB red, green, blue

RH RGB histogram method

RS recursive solver

RS_B recursive solver with bounds

VSD video scene detection

Chapter 1

Introduction

1.1 Motivations

Video scene detection (VSD) is commonly defined as the task of temporally dividing a heterogeneous video into its semantic coherent scenes. In Rui et al. [49] and Cour et al. [15], a scene is defined as a sequence of semantically related and temporally adjacent shots depicting a high-level concept or story. The shots that compose the scenes are a set of one or more frames grabbed continually by the same camera at the same time. A frame is an individual picture in a sequence of images. Therefore, a video can be considered as a hierarchical structure composed by scene, shot and frames.

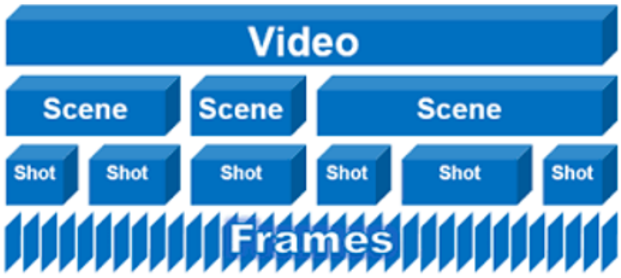


Figure 1.1: Hierarchical structure of a video

Since videos are nowadays considered the largest and most abounding source of information in the multimedia universe, broadcast companies have an increasing need to improve and speed up the process of acquiring metadata. There is a growing interest in developing automatic tools for managing video content which can simplify or substitute long manual operations.

In case of long videos analysis, manually find the part of the video which one is interested in is extremely difficult, so there is a strong necessity to simplify the managing of videos. It would be beneficial to treat videos as pieces of text, allowing significant parts to be easily identified, selected, copy and pasted, and so on. Due to the ability to isolate semantic units from the video, where each unit represents a different topic or story, the scene detection task plays a key role in metadata extraction and it can be considered one of the basic steps for several complex tasks such as video summarization, video browsing and video classification. Moreover, it is a task suitable for different types of video. For example, the sections that correspond to the different subtopics of a movie can be extracted by applying scene detection. In news broadcast videos, scene detection can be used to identify different news stories. In television videos, the commercials can be separated from the regular programs by applying video scene detection.

VSD is also one of key steps for performing an effective Automatic Audio Description (AAD) (1.2.2) which is nowadays one of the most studied and required applications by broadcast companies.

In this work, the problem of scene detection is addressed by analysing the novel approach proposed by Rotman et al.[48]. This method is parameter-free, avoiding the need for fine tuning, and relies on a novel dynamic programming algorithm to detect scene boundaries, that we name *Rotman18* ($R18$). It employs a detailed workflow composed by shot detection, frame selection, feature extraction and scene detection. Each phase of the workflow is studied in order to improve the overall performance.

The core of this work is represented by the reformulation of the dynamic programming algorithm which is responsible for the scene detection task. Two alternative methods are created from the reformulation of the dynamic programming problem: *Recursive Solver* (RS) and *Recursive Solver with Bounds* (RS_B). The first approach is only based on dynamic programming, the other is enhanced with a branch and bound heuristic.

The results show that these methods reach the optimal value of cost function and strongly reduce the execution time.

1.2 Scope and applications

Summary

Video scene detection is a key point for audio description workflow of film and TV shows. It can also be an effective tool in many other tasks of media content management of a variegate range of video genres.

The segmentation in semantic coherent parts of broadcast videos can be considered the main concern of scene detection. Despite that, the focus of this technique can be changed in order to be successfully applied in a range of different types of videos. As an example, in sports videos the task can correspond in finding highlights or scenes where a specific athlete is shown, as proposed in Ariky et al. [4] and Del Fabro et al. [18]. In documentary, news and educational videos the main objective is to automatically generate metadata for each scene in order to browse and re-use the video or part of it, as proposed in Zhai et al. [66]. Therefore, VSD is an essential pre-processing task in a wide range of video manipulation applications, such as video indexing, nonlinear browsing, classification and summarization.

1.2.1 Video summarization, classification and browsing

Video summarization is the process of distilling a video into a more compact form without losing too much information. Although video scene segmentation and video summarization are two closely related video processing tasks, they still differ significantly. Video summarization aims to facilitate large-scale video browsing by producing short and concise summaries that are diverse and representative of the original videos. Video segmentation purpose is to find precise boundaries of the semantic sections, which are typically based on a specific concept or a theme, usually defined by user's intentions. Video summarization and shot-level segmentation methods are inadequate to organise the chapters of movies that correspond to various themes. The first one generates a way too large number of key frames, the second one is inefficient, mainly because movies have long duration and widely varying contents. Scenes performs better in this task because each scene is semantically meaningful and emphasize a specific concept, like setting or action, as described in Panda et al. [40].

Video classification is the process of categorizing the video in genres, like comedy, drama, science fiction, documentary and tutorial. It is a fundamental source of information for video indexing, browsing and audio descriptions. It can take advantage of a foregoing scene segmentation.

Video browsing task consists in enabling the user to reach a specific part of the content in an efficient way: people accessing videos through web or specific apps require to easily find which section is the one they need. Since each scene can be automatically tagged, scene detection enables a finer grained search inside videos, enhancing video accessing and browsing. As an example, scene detection can be used to realize an automatic tool for video content management that can simplify or substitute long manual operations. In Baraldi et al. [5] this technique is employed for scene identification and metadatation whereas in [9] is utilized in enhancing the video browsing,

improving retrieval results presentation with semantically and aesthetically effective thumbnails.

1.2.2 Audio description

Overview

The audio description (AD) is a service that provides to the spectator with visual impairments an objective description of what happens in stage, through short and precise sentences inserted in moment of silence or between the dialogues. A voice describes the actions, the characters, the dialogues, the costumes, the body language, the facial expressions, the settings and anything that can help the spectator to follow what happens on the screen.

An overall definition can be found in the audio description guidelines [45] provided by the ADLAB project (Audio Description: Lifelong Access for the Blind): "AD is a service for the blind and visually impaired that renders Visual Arts and Media accessible to this target group. It offers a verbal description of the relevant (visual) components of a work of art or media product, so that blind and visually impaired patrons can fully grasp its form and content. AD is offered with different types of arts and media content, and, accordingly, has to fulfil different requirements. Descriptions of 'static' visual art, such as paintings and sculptures, are used to make a museum or exhibition accessible to the blind and visually impaired. These descriptions can be offered live, as part of a guided tour for instance, or they can be made available in recorded form, as part of an audio guide. AD of 'dynamic' arts and media services has slightly different requirements. The descriptions of essential visual elements of films, TV series, opera, theatre, musical and dance performances or sports events, have to be inserted into the 'natural pauses' in the original soundtrack of the production. It is only in combination with the original sounds, music and dialogues that the AD constitutes a coherent and meaningful whole, or 'text'. AD for dynamic products can

be recorded and added to the original soundtrack, as is usually the case for film and TV, or it can be performed live, as is the case for live stage performances".
In this work we discuss the AD for movies and TV shows but the descriptions can be also provided for live theatrical performances and for museum exhibitions.

Norms about audio description

There are no universal rules about AD, several guidelines are proposed by various organizations. Despite that, the majority of the aspects are common between the different regulations. From 2015 the situation has improved thanks to the ADLAB guidelines [45], that are becoming a standard in Europe.
In America the main source of information is the ACB (American Council of Blinds) with the Audio Description Project. In the USA the AD is identified as Video Description by the FCC (Federal Communications Commission) whereas in Canada is called Described Video by the CRTC (Canadian Radio-television and Telecommunications Commission).
In Europe, as mentioned above, the main point of reference about regulations is the ADLAB, an European Union programme with the aim to design reliable and consistent guidelines to train the AD specialist with funded Higher Education Institutions courses, that involves academic institutions such as Università degli Studi di Trieste, the Universitat Autonoma de Barcelona, the Insituto Politecnico de Leiria, the University of Antwerp, the University Adama Mickiewicza of Poznań and experts like Elisa Perego, Anna Matamala and Pilar Orero. As it is possible to see in Rai et al. [43], others official European guidelines are the ones from the ITC (Independent Television Commission), the OfCom (UK Office of Communications), the AENOR (Spanish Association for Standardisation and Certification) and the BCI (Broadcasting Commission of Ireland). In Italy, RAI (Radiotelevisione Italiana) has some internal regulation, primarily created to evaluate the AD provided by external

companies. That guidelines are not public so not considerable as standard by the European industry. Another important institution is the no-profit cooperative Senza Barriere Onlus, a reference point in Italy for both public and private companies that wish to make provision for accessible audio-visual material. They drafted a tutorial for would be audio describers [12]. From 2011 it is part of the ADLAB project.

Visually impaired audience

People who listen to audio description are individuals with some degree of vision loss as the result of a wide range of causes. Most of them at one point had lost all or some of their sight and now they may have only peripheral vision: they may see only shapes, light and dark colours, movement, shadows, blurs. Most users of AD are not totally blind. Only a very small part of the legally blind are blind from birth. Others are adventitiously blind or developed total blindness later in life. To establish how to automate the AD process is important to focus on which are the main principles and targets of the traditional AD. In particular it is mandatory to understand what are the needs of the visual impaired people for the story reconstruction.

Story-reconstruction model

In the ADLAB guideline [45] an important description of how audiences reconstruct stories by creating mental models of them is provided: this includes characters, actions, spatio-temporal settings and temporal relations. The actions are central in this representation model. All the aspects are related to them and they drive the story forward. When audiences process and interpret a story, they will look at the actions that are being performed and combine them with other information from the story. On a first level, audiences create frames that serve as a context for every event in the story. In these frames, information on the characters that are presented and the spatial and temporal circumstances in which the event takes place, takes the

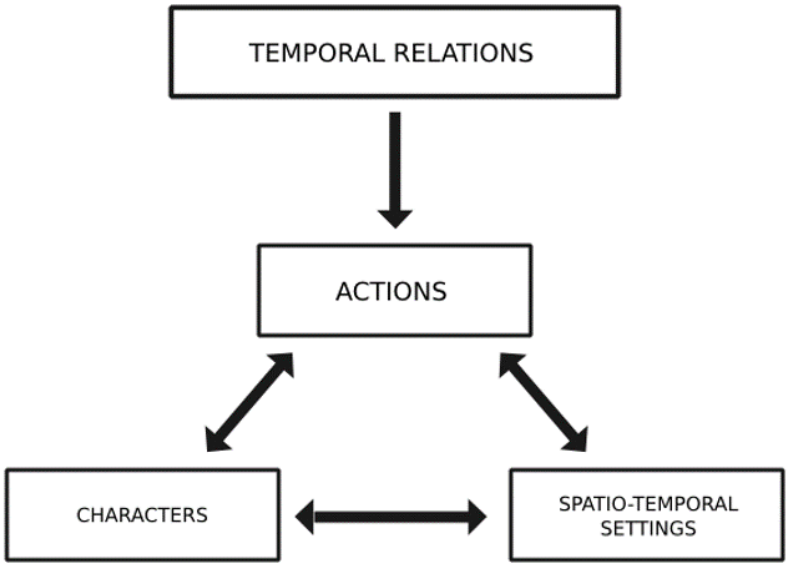


Figure 1.2: [45] Model of story reconstruction

form of general labels. On the second level, more detailed information is added to these labels. When a new story event is presented, the audience will check whether it can be attributed to an already existing frame: in this case the existing frame has to be updated, otherwise a new frame has to be created. The AD, in combination with the audio of movie, first has to contain all the necessary cues to allow the impaired sighted audience to create a context for every event taking place in the story. Only in a second stage detailed information that fit that particular context should be provided.

What to describe

AD must include characters, costumes, spatio-temporal setting, actions, hardly recognizable sounds and text on screen. Characters and their actions and reactions are an essential part of a film narrative. It is important to determine the focal characters

(protagonist, antagonist), the supporting characters and the background ones, to understand the relations between them, if they are new or known, realistic or unrealistic, authentic or fictional, if they change during the narration, if they have a symbolic function, what actions and reactions of a character move the story forward to the greatest extent. The story takes place in spatio-temporal settings intrinsically linked to the characters and their actions, which comprise both a temporal and a spatial dimension. It is primary to determine through what channel the spatio-temporal information is provided, if the setting is new or already known, global or local, real or imagined, if it serves as a background or has a narrative-symbolic function, if it has undergone a complete transformation and the relations between settings and the characters in them or their actions. About the sound effects, it is necessary to determine if they are easy to understand or not and if is important to identify the source of an easily identifiable sound. Text on screen refers to any type of written text that appears on the screen: it is primary to determine if the information provided is already offered by other means or if it is necessary to render it orally in the AD, deciding how to indicate the appearance.

When to describe

There is a general convention that the AD should be produced when there is a gap in the oral channel of the audio-visual text, so it is better not to talk over dialogues, commentaries, significant sounds and, when possible, music and songs. It is important to synchronise the description with actions, settings and sounds, avoiding overlap with the film dialogue and the soundtrack. The describer decides if the description will precede or follow the dialogue line, sound or music event it refers to. He has to keep in mind that a preceding description will set up the event in the audience's mind and a description that follows their referent will contribute to create suspense, surprise and comic effect. In general, the closer a description is provided to an event, the

clearer is the link for the audience.

How to describe

In the descriptions, the use of the present tense and the present continuous for ongoing activities is recommended. Also the use of the third person pronouns is suggested, as they reflect the voice of an omniscient narrator. Clear, precise and concise language, along with simple and concrete vocabulary help audience with information processing and visualisation. Time limitations and the need for an intelligible style promote the use of short sentences. Setting the scene is an essential part of AD and takes precedence over every other aspect. Furthermore, the narrative part should always anticipate the action. Describer watching a video several times may notice mistakes in continuity or in the editing: pointing them out to the viewer is not necessarily helpful because distract him from the video. A golden rule is that the describer, despite having a certain freedom, must be as much objective as possible. He should describe the scene without giving a personal version of what is in it, by not interpreting in the spectator's place. AD requires the same level of attention in delivery and intonation as any commentary or voice-over: good audio describers should be unobtrusive and neutral, but not lifeless or monotonous. The delivery should be in keeping with the nature of the programme. Visually impaired people tend to hold strong opinions about people's voices, if they do not like the voice, they may not listen to the content.

How much to describe

Cohesion is the property that helps the audience to understand a description with reasonable ease and find continuity of sense in it. It is all about striking the right balance. The insertion and the length of description need to be balanced: a pace of 160 words/minute can be a good reference. Unnecessary information needs to be

discharged in order to obtain an equilibrium that allows the viewer to take few breaks. Balance can be also accomplished through informational shifts. The reviewing phase is particularly important to achieve this goal.

Manual audio description process

Even if each AD provider has its own best practice, the manual production process for film and TV series usually includes the 7 following fundamental phases proposed in the ITC guidance [29]:

- *Choosing suitable programmes for description*, even if the AD should be as pervasive as possible, there are some peculiar content that are not appropriate for audio descriptions, as an example the videos that are too fast;
- *Viewing the programme* (also called "Source text", ST), the video have to be watched at least one time in order to understand the plot and other details. If it is a series, the episode should be contextualized: the characters, their names and their relationships to each other need to be known and understood. Most production companies or presentation departments should be able to send a script, that can offer clues to sequences which are not clear at first viewing. Therefore, scripts should only be utilized as a raw source of information because the final edited programme or film often differs substantially from the original screenplay;
- *Preparing a draft Script* (also called "Target text" – TT), the descriptions should be written and timed, avoiding overlaps with the other channels on the soundtrack, especially the dialogues;
- *Reviewing the Script*, the script review could be done while viewing the film, together with a visually impaired collaborator;

- *Adjusting the programme sound level*, when a descriptive commentary is inserted into a video, the background level of programme audio needs to be reduced so that the description can be clearly heard. The narrative voice is fixed at a constant level at the start of the recording;
- *Recording the description*, established background audio levels and checked timings, the script is then recorded paying attention to the delivery;
- *Reviewing the recording*, at the end of the recording phase each description should be listen back to ensure that has been delivered without mistakes, omissions or imperfections.

Professional roles and tools

The creation and distribution of AD is a complex process that requires the collaboration of multiple professionals from different fields:

- audio describers;
- voice talents or voice actors;
- sound technicians;
- copyright experts;
- blind text editors or users.

According to ITC guidance [29] the production of an AD usually employs workstations that normally consists of a number of items:

- a personal computer which acts as a word processor, time-code index, video edit controller and prompting device for recording the description in the gaps between programme dialogue;

- a time-coded DVD-BD player;
- an additional small monitor and associated loudspeakers (even if the PC has a video peripheral);
- a device which stores the descriptive audio.

In addition a sound recording and mastering studio for the voice talents and sound technicians is required.

Realization times of a manual audio description

According to the ITC guidance [29], the average recording time for a one-hour description is approximately two to two and a half hours. The production and review of the script, the sound mixing and editing are not included in the time computation. Globally the complete process to generate one hour of AD can takes a week of work, depending on the type of content: a film is way different from a TV series.

Workflow of an automatic audio description

The creation of an automatic audio description is an open problem, that involves multiple modalities (audio, visual and textual). The solution considered for this challenge is to break down the automation process into individually solvable sub-problems.

In order to generate an AAD, a pipeline that combines the fundamental sub-problems and their outputs is developed. Some of them relies on consolidated technologies, others need a research activity in order to be ready for the process. In this work is discussed only the scene detection phase.

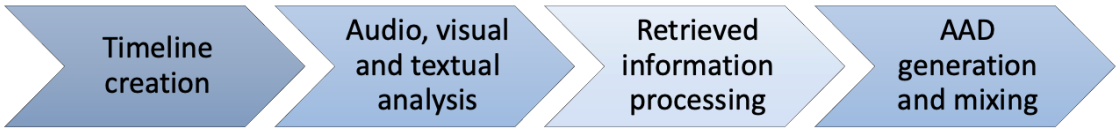


Figure 1.3: Audio description workflow

Creation of the timeline

The first step to obtain an AAD is to divide the video in scenes, in order to generate a timeline of the spatio-temporal units of the video. In each scene the spatio-temporal setting is the same and can be used as a starting point to retrieve coherent information about each single part of the video. The research topics involved in this step are shot and scene detection.

Audio, visual and textual analysis for information retrieval

After the timeline generation, information must be retrieved from each scene. It is possible to obtain information from different modalities:

- *Visual*: video can provide information about the characters, the faces, the settings, the objects and the actions. Technical metadata, if available, can give information about the type of the shot, the brightness, the exposure and many other important video parameters;
- *Audio*: to extend the understanding of the visual part can be useful to analyse the auditory part, despite the fact that is the only one completely perceivable by visually impaired people. Dialogues, voices and sounds can help the algorithm to understand characters, settings and actions by means of information retrieval techniques;

- *Text*: subtitles are usually available for the majority of the video content so they can be used to better transcript the dialogues and the settings information.

The research topics involved in this phase are the facial, sound, settings and action recognition, the object recognition and tracking, the speech and textual recognition and summarization and the dialogues timing verification from the subtitles.

Processing of the retrieved information

After the extraction phase, the information of each scene needs to be correlated, compared and filtered in order to exclude the parts that are already understandable from the auditory channel. The details that let visually impaired people improve the comprehension of the content have to be prioritized. In addition, it is important to identify the number, length and position of the available time-zones where it is possible to place the ADs.
In this case the research topics are information retrieval, prioritization and processing.

Generation and mixing of the audio descriptions

Starting from the information retrieved in the previous phase, the textual parts need to be generated in order to fit the available time-zones computed from the video pause. The research topic involved are natural language processing and voice generation.

1.3 Rotman et al. workflow

In this work, the scene detection task is addressed by reproducing and customizing the innovative approach proposed by Rotman et al. [48], focusing on visual information. Such approach is selected because it is applicable to all possible types of video since it is parameter-free and avoids the need for fine tuning.
The Rotman et al. approach relies on a well defined workflow composed by several

steps: shot detection, frame selection, feature extraction and scene detection.

The shot detection phase is based on the work proposed by Baraldi et al. [8] (3.1.1), aiming at dividing the video in its shots.

The frame selection phase implements the middle frames method (2.3.1) to extract a key frame that represents the shot.

Once the key frames are extracted, the feature extraction step (2.4) is performed on the basis of a deep learning approach that involves the use of the InceptionV3 network (2.4.3) to extract a 2048-dimensional feature vector from each frame. Moreover, it is also possible to extract audio features by using the VGGish [27] model which produces a 128-dimensional vector every 0.96 seconds.

For each modality (visual or audio) the Euclidean distance between each feature vector is computed in order to obtain a symmetric matrix with zero elements on the main diagonal. In case of a multi modality analysis (visual and audio) both distance matrices are normalized and averaged together.

The distance matrix, together with the number of scenes, is provided as input to a novel dynamic programming algorithm, named *Rotman18* ($R18$) (3.4), which performs the scene detection task. The final output is given by a vector where each element is a shot index that identifies the final shot of each scene. This vector must include the starting index 0 and the final index of the shots of the video. For example, in a video with 159 shots and 11 scenes, the scene division can be [0, 4, 5, 14, 29, 49, 110, 114, 142, 152, 157, 159].

Finally, the result is evaluated on several metrics by measuring both the computational cost of the solution and the accuracy of the division. The cost of the solution is computed by using the normalized cost function $\mathcal{H}_{nrm}$ (3.4.3) whereas the accuracy of the division is evaluated by using the *F-score* (4.1.2).

Moreover, the authors also incorporate a method to estimate the number of scenes of the video from a previous work [46] and a non-greedy algorithm for creating a

hierarchical division tree. Such method relies on computing multiple sets of optimal divisions at different granularity levels and determines the hierarchy of divisions according to a consensus score.

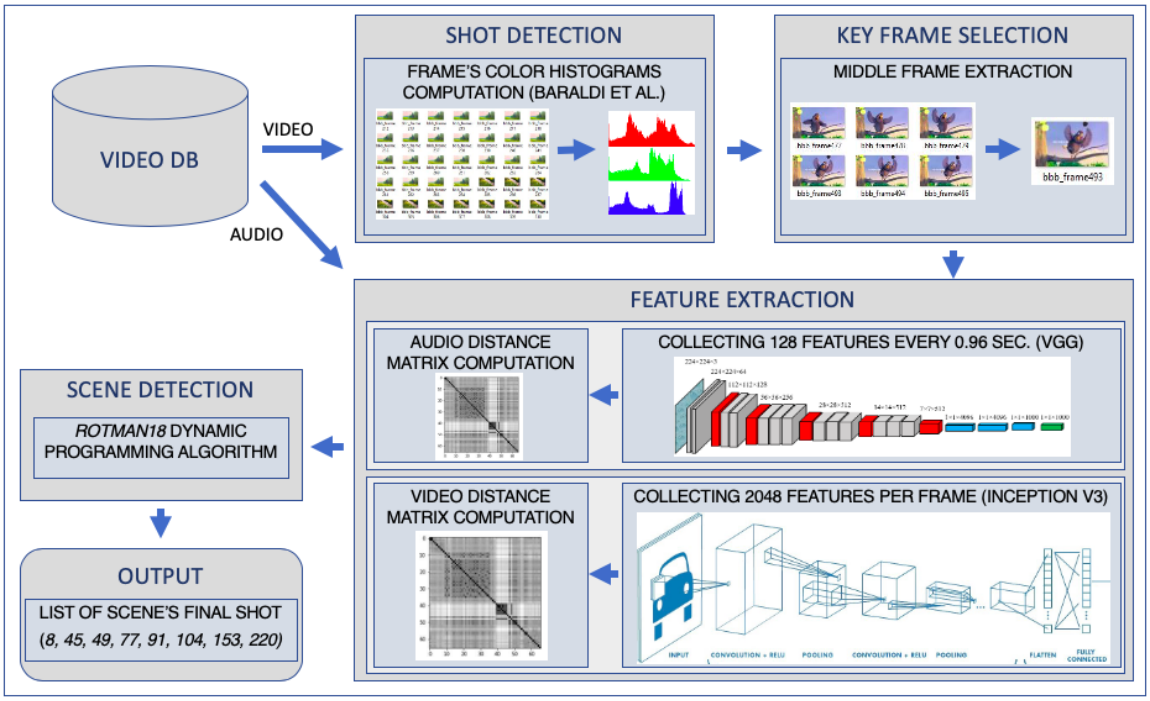


Figure 1.4: Rotman et al. workflow

By making some assumptions that do not strongly alter the Rotman et al. workflow, several methodologies are proposed in Chapter 3 in order to improve the overall performances.

1.4 Structure of the thesis

In Chapter 2, an overview of the most common approaches in the literature related to the Rotman et al. workflow is reported. It also contains a brief review of the most commonly used datasets for scene detection task (2.5).

Chapter 3 explores the implemented methodologies related to the steps of the Rotman et al. workflow: shot detection (3.1), frame selection (3.2), feature extraction (3.3) and scene detection (3.4)(3.5).

In Chapter 4 are reported the evaluation metrics (4.1) and the results of the implemented shot segmentation (4.2), frame selection (4.4) and scene detection algorithms (4.5).

The conclusions are addressed in Chapter 5.

Chapter 2

State of the art and background

Summary

In this chapter, a state of art related to the methodologies of the scene detection workflow proposed by Rotman et al. [48] is explored. In the first section, the main scene detection approaches proposed in the literature are reported. The following sections focus on operating principles and possible alternatives related to the steps that precede the scene detection algorithm: shot detection approaches (2.2), frame selection approaches (2.3) and feature extraction approaches (2.4). Although some of them are nowadays standardized procedures, we decided to explore them, since the purpose of this work is to possibly improve the Rotman et al. approach. In Section 2.5, the most commonly used dataset in literature related to the task of scene detection are reported.

2.1 Scene detection approaches

In this section, a brief overview on the main scene detection methods is explored by focusing on methods which involves the use of visual features. In literature,

four classes of approaches are distinguished: rule-based approaches, graph-based approaches, stochastic based approaches and clustering based approaches. Finally, we summarized the most interesting scene detection methodologies introduced in latter days.

2.1.1 Rule-based approaches

Typically, film producers follow some basic rules, related to film editing or film grammar, when creating a scene. Consequently, the structure of a scene could present recurring patterns. Rule-based approaches exploit those common patterns to identify scene boundaries. The most common rules are:

- *180-degree rule*, it regards the spatial relationship between characters in the scene. An imaginary axis between the characters and all cameras that are located on one side of this axis is drawn. The scene is taken only from one side such that participants preserving the same left-right relationship to each other from shot to shot.

- *Action matching rule*, filmmakers creates the impression of a continuity in action by using a cut which connects two different view of the same action at the same moment. In two consecutive shots, the motion direction shall be the same.

- *Film tempo rule*, the rhythm of a scene is strongly influenced by shot motion, shot length and regularity of sounds. Typically, tempo is different in adjacent scenes and scenes with high tempo are never aligned together. In Adams et al. [2] such behavior is exploited by marking large rhythm transitions as scene boundary whereas small rhythm transitions are marked as event within the scene.

- *Shot reverse rule*, it is a rule usually applied in dialog scenes between two characters. The camera moves between the two characters while they are talking.

Film historian David Bordwell defines such technique as: "wherein one character is shown looking (often off-screen) at another character and then the other character is shown looking 'back' at the first character. Since the characters are shown facing in opposite directions, the viewer unconsciously assumes that they are looking at each other".

- *Establishment/breakdown rule*, in a scene an overview shot that shows all involved participants and objects is performed. By exploiting the shot reverse rule, the overview shot is typically followed by a series of breakdown shots which show close-ups.

In Liu e al. [35], by studying the narrative patterns of shot and scenes in movies and TV shows, a visual based probabilistic framework is proposed to imitate the authoring process. This method addresses the scene detection task by incorporating contextual dynamics and learning a scene model.
Since each video is characterized by a genre, the application of such rules for the scene detection task have different effects depending on the video content. Sometimes there could be an improvement of the performance, sometimes it could not. Moreover, it is not sure that directors will always follow those rules.

2.1.2 Graph-based approaches

In graph-based approaches, the scene detection task is faced as a graph partitioning problem. Typically, as basic step, the video is divided in shots that are grouped together based on visual similarity or temporal closeness. A graph structure is built by considering cluster of shots or single shots as nodes which are connected by edges based on similarity or temporal closeness. Graph methods aims at dividing a graph in multiple sub-graphs where each sub-graph is a scene.
In Rasheed and Shah [44], a weighted undirected similarity graph is built where the

nodes represent each shot and the edges are based on a weighted shot similarity function which measures the similarity of Hue, Saturation and Value (HSV) color histograms and the motion content of two shots. The individual scenes are obtained by splitting the graph into sub-graph using the normalized cut technique for graph partitioning. If a clustering of shots is performed, the global similarities of shots is considered rather than the individual shot pairs.

In Sakarya and Telatar [50], a step-wise partitioning of the graph is performed. At each iteration, this algorithm isolates a group of shots which have a high visual similarity with each other and a high visual dissimilarity from the rest of the shots. This set of shots is called dominant and represents one scene. Therefore, at each iteration two sets are created: the dominant set and the set composed by the remaining shots. Based on the dominant set, a tree-based peeling strategy is performed to determine the remaining scenes.

Graph-based approaches perform very well for videos with always repeating types of scenes, like news broadcasts or talk shows. They struggle in segmenting dynamic scenes where a motion is performed. Another drawback of such methods is the lack of a unified agreement on how to represent the video. Some of them represent shots as nodes while others represent shot clusters as nodes. Therefore, experimental results can not be shared between methods.

2.1.3 Stochastic-based approaches

In stochastic based approaches, the task of scene detection is solved by developing a probabilistic model which involves a mathematical representation of the video content. An optimal segmentation is approximated by maximizing the a posteriori probability of the estimated scene boundaries to be correct.

In Zhai and Shah [65], the boundaries between scenes are identified thanks to the Markov chain Monte Carlo (MCMC) technique. Monte Carlo (MC) methods are a

subset of computational algorithms that use the process of repeated random sampling to make numerical estimations of unknown parameters. As first step of their works, a fixed number of boundaries are initialized at random locations in the video. Then, those boundaries are updated by performing diffusion and jumps. The diffusion updates the boundaries between adjacent scenes whereas jumps consists on merging two scenes or splitting an existing scene. Relying on the a priori model and data likelihood, the algorithm aims at maximizing the a posteriori probability of the correctness of the scene segmentation proposed.

Another probabilistic method is performed by Han and Wu [24]. They use dynamic programming with a heuristic search schema to investigate in a certain order all the possible boundary combination. As an example, a video of 5 shots has 16 possible boundary sequences to be explored. At each iteration, a sequence of boundaries is investigated and evaluated against random boundary thresholds. After each iteration, each boundary gets a vote. Then, a Monte Carlo method is applied to the vote in order to stabilize the searching process. At the end, the real scene segmentation is composed by the boundaries with enough votes.

In general, the performance of stochastic-based approaches is strongly influenced by the selection of the training dataset. They require a huge amount of data for training the model and create a good dataset. If this is accomplished, such approaches can achieve high accuracy.

2.1.4 Clustering-based approaches

Clustering-based approaches aims at grouping frames into meaningful clusters depending on shot similarity.

Baraldi et al. [5] proposed an approach for scene detection task based on spectral clustering that aims at grouping adjacent shots. Each shot is represented by a $l1$-normalized histogram obtained from the sum of Red, Green, Blue (RGB) color

histograms of the frames belonging to the same shot. Then, they create a similarity matrix that jointly describes appearance similarity and temporal proximity between shots by using the Bhattacharyya distance and the normalized temporal distance. The Bhattacharyya distance is widely used in feature extraction and selection research and measures the similarity of two probability distributions. Finally, they apply the spectral clustering to the similarity matrix, using the normalized Laplacian matrix and the maximum eigengap criterion.

The same authors proposed another work [6] where they extract features and compute distances between shots by training a Siamese deep network. The structure of such network is composed by multiple branches containing Convolutional Neural Networks (CNNs). The distances between shots are arranged in a similarity matrix which is used with spectral clustering to group adjacent shots. The scene borders are identified between shots that are members of distinct clusters.

Another method which involves spectral clustering is described by Panda et al. [40]. They propose a Nyström approximated multi-similarity spectral clustering approach with a temporal integrity constraint. As first step, they perform shot detection using an information theory-based shot method [13] and represent each shot by extracting its middle frame. Next, they created a series of shot similarity matrices based on color, texture, motion and semantics extracted from each middle frame. Then, spectral clustering [28] which incorporates temporal integrity constraints is performed. It aims at dividing the video shots into clusters depending on multiple similarity measures based on the similarity matrices, avoiding the use of a temporal distance. A Nyström approximation is applied to find the approximated eigenvectors in spectral grouping in order to reduce the high computational cost of constructing multiple similarity measures. Such technique uses a subset of its columns for obtaining a low-rank approximation of the large kernel matrix [20]. Finally, a label is assigned to each shot according to the cluster it belongs to and a scene boundary is identified when two

adjacent shot labels are different.

Since typically the clustering-based approaches do not consider the temporal position and order of video frames, their efficiency can be affected by outlier shots and ping-pong shot sequences.

2.1.5 Recent approaches

In Protasov et al. [42], a pipeline which involves feature extraction and filtering, shot clustering and labelling is proposed. As in Rotman et al. approach, a CNN (2.4.2) is used for feature extraction: Protasov et al. chose the Places205-AlexNet image classification network which is the AlexNet CNN trained on 205 scene categories of the Places database, with 2,5 million labeled pictures of scenes. The difference is that each feature vector is filtered to reduce impulse noise before applying the shot detection. The shots are identified by using feature values deviation in adjacent frames. Once shots are obtained, the chain method [60] is applied to extract key frames, that are then clustered in order to find the scene edges.

Haroon et al. [26] pursue the common steps of the scene detection workflow: shot detection, key frame selection and scene boundary detection. The novelty is represented by the use of a Bag of Visual Word model (BoVW) for feature quantization. Such technique comes from the Bag of Words model (BoW) which is typically applied in document classification to count the number of each word, use the frequency of each word to know the keywords and make a frequency histogram from it. Therefore, the document is treated as a BoW and the video is handled as a BoVW. In image classification, this model uses key points (points insensitive to rotation and expansion) and descriptors (description of a key point) extracted from images to build vocabularies and represent each image as a frequency histogram of features that are in the image. Scale-invariant feature transform (SIFT) [36] algorithm is used to transform an image into a large collection of key points and descriptors. Since by applying SIFT, each

image is represented by 2-3 thousand feature vectors, it is computationally expensive to compare two images. So, they present a BoVW model variant called Vector of Linearly Aggregated Descriptors (VLAD) to reduce the feature space. For a given frame, it is computed a vector which contains the occurrence of a word (feature) that appears in the frame. K-means clustering is applied to this vector to extract the centroids which represent the visual words. Instead of computing the histogram of visual words, VLAD computes the sum of the differences between residual descriptors and visual words and concatenates them into single vector, resulting faster than BoVW. BoVW or VLAD vectors are used to determine the similarity between two key frames and determine the scene boundaries.

In Kishi et al. [32] a method that exploits multi-modal information from the video is designed. Bag of features processing is used to refine the low features obtained from key frames by applying CISFT [11] (a variant of SIFT which uses color descriptors instead of gray scale ones) and MFCC [16] (an algorithm to extract audio features). Again, k-means clustering is used to extract each cluster centroids (feature word) from each modality which compose a set called feature dictionary. Each shot is represented by a feature word occurrence vector, namely feature word histogram. Both aural and visual words are fused for each shot while maintaining the single modal semantic patterns. A shot clustering scene transition graph (STG) [37] method is performed on these features in order to obtain the scene boundaries.

A new approach based on dimension reduction and temporal clustering is explored by Peng et al. [41]. Such innovative methodology considers the user's intention when applying the scene detection: it is possible to obtain a segmentation more or less detailed depending on user's choice. The features extraction from frames is performed by using the InceptionV3 model pretrained on ImageNet followed by a dimensionality reduction which extracts the features in a coarse granularity. Next, the distance similarity between frames is computed by measuring the similarity between sub-blocks

of two frames with different partitions. Depending on user's choice, a user-guided temporal clustering is used to segment videos on a time domain. Finally, a hierarchical clustering is explored in order to allow the division into semantic sections at different abstraction levels.

Ji et el. [30] proposed a different scene detection approach which exploits image captioning. As usual the first step is the shot detection: shot boundaries are identified by comparing colour histograms of each frame. Next, the key frame extraction by selects the maximum-entropy frame within each shot. Each key frame is given as input to a long short-term memory (LSTM) [62] network which is responsible for the generating a semantic text which describes the key frame. A LSTM is an artificial recurrent neural network typically used in speech recognition and handwriting recognition which is characterized by feedback connections. The scene segmentation is identified by comparing both the colour histograms and the generated text of each key frame.

2.2 Shot boundary detection approaches

The task of shot boundary detection is an operation typically used in software for post-production of videos that aims at the division of a film into basic temporal unit called shot. A shot is defined as an unbroken sequences of pictures (frames) taken by the same camera, representing a continuous action in time and space. Since scenes are typically defined as a sequence of semantically related and temporally adjacent shots depicting a high-level concept or story, shot boundary detection is considered a required preliminary step in scene detection and more generally in automated indexing, content-based video retrieval and summarization applications. Different shots are divided by a shot boundary, also known as transition, that can be classified in two main type, namely, abrupt transition and gradual transition. Abrupt transitions or hard

cuts are sudden transitions from one shot to another that occurs in one frame. Thus, an abrupt transition occurs between the last frame of a shot and the first frame of the following shot. By contrast, gradual transitions or soft cuts are transitions in which two shots are combined using chromatic, spatial or spatial-chromatic effects which gradually substitute one shot by another. Soft cuts may include two or more frames that are visually interdependent and contain fragmented information. There are lot of different type of gradual transition as for example dissolves, wipes and fades. The shot boundary detection performance is strongly affected by the extracted features and it can be measured by its ability in detecting correct transition compared to the computational cost of the algorithm. A transition that is detected correctly is called a hit. A shot boundary that occurs in the video, but it was not detected is called a missed hit (false negative). By contrast a false hit (false positive) is a transition that is detected by the algorithm but actually it is not present in the video. For example, a false positive occurs when the algorithm identifies high level differences between the shot due to motion in the video (motion blur). The ideal case is represented by an algorithm where the video content of frame is optimally expressed by the features extracted and its behavior leads to minimizes false positives and false negatives while maintaining both low computational cost and high speed. Owing to some effects that appear in a video shot such as flash lights, light variations, object/camera motion and camera operation (such as zooming, panning, and tilting), the ideal situation is almost impossible to fulfill. Moreover, it can be observed that the accuracy and the computational cost of an algorithm are strongly related: as the accuracy increase, the computational cost is increased and vice-versa. Despite those factors, there are a plenty of available methods for shot boundary detection that can be used without a specific customization with impressive performance.

2.2.1 Shot boundary detection steps

Generally, shot boundary detection methods are based on detecting video shot transition by measuring the visual discontinuities between successive frames. As proposed in [1], the shot boundary detection is performed thanks to the following steps: feature extraction of visual content, construction of a similarity/dissimilarity measure and classification of the measure.

Feature extraction of visual content

In order to perform the feature extraction, it is necessary to apply a certain function Ψ to each frame i of the video, by obtaining the extracted feature $F(i)$. A suitable extraction function leads to the extraction of invariant and sensitive features. An invariant feature is a visual information that remains stable within shots because it is insensitive against the temporal variations of the frame such as object and camera motions. Conversely, a sensitive feature contains considerable changes within shot transitions. To obtain a high level of accuracy in detecting shot transitions, a shot boundary detection method needs to be able to combine invariant and sensitive features as pixels, histograms, edges, motions and statics.

Construction of a similarity/dissimilarity measure

In this step, the visual content is represented as a series of measures with one or multiple dimensions by computing the similarity/dissimilarity between two consecutive frames features $F(i)$ and $F(i+1)$. This computation can be expressed with the Minkowski distance also known as l_p-norm which can be considered as a generalization of both Euclidean distance and Manhattan distance. The formula is expressed as:

$$D(i, i+1) = \left(\sum_{k=1}^{K} |F_k(i) - F_k(i+1)|^p \right)^{\frac{1}{p}} \tag{2.1}$$

where K is the number of features and $p > 0$. By putting $p = 1$ the Manhattan distance is given whereas by putting $p = 2$ the Euclidean distance is given. Ideally, the dissimilarity distance assumes high values in correspondence of shot transitions and low values within the same shots whereas it is the opposite in case of similarity measure. The performance of similarity/dissimilarity distance can be affected by vast amounts of disturbance such as object or camera motion and flash light occurrence presented in video.

Classification of the measures

The results obtained by computing the distance between two consecutive frames are evaluated against a threshold. If the score is higher than the threshold a transition is detected. There are three way to select a threshold, fixed, adaptive and machine learning. In fixed threshold approaches, the value of the threshold is selected based on video type at the beginning of the algorithm and it never changes during the execution. By contrast, in adaptive threshold approach, the similarity/dissimilarity measures are compared to a threshold which adapts itself to the properties of the current video during the execution by considering various parameters in the video. These approaches based on thresholding struggles in distinguishing between transitions and disturbance factors in similarity/dissimilarity measures.

Machine learning-based approaches can be used to overcome this issue by assuming transitions detection as a classification problem. This method eliminates the need for thresholds and embed multiple features although it can be tough to find a suitable feature combination for shot boundary detection.

2.2.2 Shot boundary detection methods

In this section we reported some of the basic shot boundary detection approaches based on visual features and one innovative approach that uses machine learning.

Pixel-based method

The pixel-based approach is both the most obvious and most simple algorithm of all which evaluates the differences in the intensity values between corresponding pixels in two consecutive frames. The easiest way to detect if two frames are different is to calculate the sum of absolute pixel differences and compare it against a threshold. A transition is declared if the sum exceeds the selected threshold. The sum of absolute pixel differences is equivalent to the Manhattan distance and can be expressed as:

$$D(i, i+1) = \frac{1}{N_x N_y} \sum_{x=1}^{N_x} \sum_{y=1}^{N_y} \sum_{z=1}^{N_z} |I(i, x, y, c_k) - I(i+1, x, y, c_k)| \tag{2.2}$$

where x and y are the location of the pixels of two consecutive frames $(i, i+1)$, $I(\cdot)$ indicates the pixel intensity, N_x and N_y are width and height of the frame and c is the number of color channels. Beside this formula, several other similarity measures such as the Euclidean distance, Euclidean norm, the mean absolute error and the Camberra distance can be applied [10]. Early examples of pixel-based shot boundary detection are the one from Kikukawa T. et al. [31] and the one from Nagasaka et al. [38], that extended the first method for the purpose of reducing disturbance in dissimilarity signals. Zhang H. et al.[67] added a preprocessing step to the Kikukawa method in order to detect hard and soft transitions to automatically select the threshold. Shahraray B. [51] evolved this transition detection method dividing the frame in 12 regions and finding the best match between regions in the current frame and the corresponding neighborhood regions in the next frame. Despite several variants of pixel-based methods have been proposed, the performance of such approaches is still affected by flash lights, light variations, object/camera motion and camera operation.

As result, intra-shots with camera motion can be incorrectly classified as gradual transitions. Pixel-based methods are used often to produce a basic set of possible correct detected transitions as they performed very well in detecting all visible abrupt transitions.

Histogram-based method

A histogram is a table that contains for each color within a frame, the number of pixels that are shaded in that color. Histogram-based approaches detects shot boundaries by computing the grayscale or color histograms of each frame and compare them among each other. A shot boundary is detected when the difference among the histograms becomes larger than the preset threshold. Thanks to the tolerable trade-off between accuracy and computation cost, these approaches are widely used in practice. One of the simplest approaches is the one proposed by Swanberg et al. [55], that consist in computing histogram for each space in the RGB color space after partitioning each frame into blocks. It is computed a measure called histogram distance measure HDM as follows:

$$HDM(i, i+1) = \sum_{k=1}^{3} \sum_{b=1}^{N_b} \sum_{h=1}^{N_k} \frac{(H(i,h,c_k,b) - H(i+1,h,c_k,b)^2}{(H(i,h,c_k,b) - H(i+1,h,c_k,b)} \tag{2.3}$$

where $N_H$is the total number of possible gray levels, N_B is the total number of blocks and $H(i,h,c_k,b)$ is the histogram value for the h^{th} level in channel k at the b^{th} block in the frame i. Several approaches which involves various color spaces for histogram and distance measure have been proposed over time. One of the earliest approaches was the one proposed by Nagasaka and Tanaka [38] that utilizes gray level for hard transition detection: it resulted ineffective against temporary noise, flash lights and camera motion. Shortly after, Zhang H. et al. [67] introduced the twin comparison technique, in which successive frames' gray histograms are computed and compared using the HDM equation for the purpose of detecting hard and soft transitions thanks to low and high thresholds. Lienhart et al. [33] computed HDM by means

of a color histogram, obtained discretizing the RGB color component while Ahmed et al. [3] for the same purpose utilized the hue component only. Finally, Gargi et al. [21] implemented an HDM with different color spaces for histogram (including RGB, HSV, YIQ, L*a*b, L*u*v* and Munsell [52][58]) and distance measures (bin to bin, chi-squared χ^2 and histogram intersection)[10]. Generally, these approaches assume that there is minimal diversity between the histograms of two consecutive frames within a shot which represent firm objects and backgrounds. Due to this assumption, the histograms of two frames that belong to different consecutive shots are comparable only if their contents are completely or partially different. However, two frames that represent completely different content as for example a picture of the sea and a beach compared to a picture of a corn field and the sky, may have the same histogram. As result, there is no guarantee that histogram-based methods identify all hard transition without incurring in false positives and misdetection, but they are less affected by motion respect to pixel-based approaches because they ignore the changes in the spatial distribution of frames.

Edge-based methods

The core of such approaches is the idea that the edges of the objects in the first frame after a hard transition cannot be found in the last frame before the transition and vice versa. So, edge-based approaches focus on computing the difference between positions of the edges of the current frame and the edges of the previous frame. If the difference is large, a transition is detected. This method, that was introduced for the first time in hard and shot transition detection by Zabih et al [64], involves several steps as applying motion compensation, finding the boundaries of objects within images (edge detection), counting the edge pixels in two consecutive frames, defining entering and exiting pixels and computing the maximum between pixels out and pixels in (edge change ratio). Nam et al. [39] later proposed a similar approach based on wavelet

transform for the spatial subsampling of the frames and edge extraction. An edge-based method was used to detect dissolve transitions in [33] and [34], and to discover fade-in and fade-out transitions in [68]. Since these approaches may include such operations, the overall computational cost is higher than histogram and pixel-based approach. In term of performance, they are relatively robust against camera motion and can detect both hard cuts and gradual transitions, but they are still affected by false positives resulting from zoom camera operations.

Deep learning-based methods

In addition to the traditional methods based on visual characteristics, in recent years, novel approaches that takes advantage of the CNN abilities to classify and to extract high level features from images and video frame are created. They introduce a completely different workflow respect to the previous methods, conceiving the shot detection as a binary classification problem. To reach the objective, the CNN is trained to correctly predict if a frame is part (or not) of the same shot of the previous frame. In order to train the network, a dataset with many frames and automatically generated labels about hard cuts, soft cuts, flashes and none-transition examples is preliminary composed. That approach is the one presented in Gygli [23], and its implementation is explored in Section 3.1.2.

A partially different procedure is the one shown by Xu et al. [63], where a CNN is trained on the ImageNet dataset in order to extract a feature set from the sixth layer of the network. The purpose is to construct the similarity signal between the feature sets of two consecutive frames, trough the Cosine similarity measure. These methodologies seem to produce good segmentation results compared to visual characteristics approach and can be further improved by adding samples of specific film genres during the training phase.

2.3 Frame selection approaches

As shown in Rotman's workflow and in most of the scene detection approaches proposed in literature, the shot detection phase is usually followed by the application of a key frame selection method. This technique, typically employed in video summarization task, aims at finding a subset of the total frames that properly represents the video. In this way, users can quickly browse over the video by viewing only a few highlighted frames. In this work, this method is applied to the frames that belongs to a shot obtaining one or more frames that best represent each shot. Moreover, instead of considering all the images in the video, the scene detection algorithm relies only on those key frames to identify the scene boundaries. Since usually a scene detection algorithm works on the feature extracted from images, by reducing the number of images to be processed it is possible to speed up the process in terms of time.
In the following sections, we reported the most interesting key frame selection method found in literature.

2.3.1 Frame selection methods

The easiest approach is to extract a precise frame from each shot. In [38], the first frame of each shot is selected as key frame. Although being simple, usually the first frame is not stable and does not capture the major visual content. Another deterministic method is to extract the middle frame of each shot as key frame [48][40][9]: it is considered a good choice as it can capture the general view of the overall content. Other approaches rely on the colour features of each image since are not greatly affected by the spins or movement of the input image. In [32][65][44], key frames are identified by computing and comparing the colour histograms extracted from frames. Typically, such approaches extract more than one key frame from each shot. As first step the middle frames are extracted. Next, the histogram of each frame is compared

against the one extracted from the middle frame within the same shot. If the difference exceeds a certain threshold, the frame is selected as key frame and added to a set of key frames which represents the shot. Several colour spaces (HSV, RGB) could be employed for computing colour histograms (3.2.3). These methods assure that every key-frame is distinct preventing redundancy.
In [30][26], colour entropy is used as indicator. Entropy is a statistical measure of randomness that can be used to characterize the texture of the input image. Typically, the frame that scores the max value of entropy is selected as key frame. Grey-scale and color frames are employed in such methods (3.2.4).
In other works [69][25][22][14], clustering is used to extract key frames. The frames within the shot are clustered into several clusters which number is regulated by a parameter, depending on their visual similarity. This similarity can be computed by considering colour, texture, shape of the salient object of the frame, or the combination of the above. Once the clusters are formed, some key frame are extracted from the clusters: as example the frame closest to the centroid can be selected as key frame. Avila et al. [17] presented an improved version of the k-means algorithm where they grouped the frames in sequential order instead of being randomly distributed between the clusters before applying the traditional k-means algorithm. One frame is then selected for each cluster.

2.4 Feature extraction using convolutional neural network

In recent years, one of the most common approaches to extract information from images and videos is to use deep convolutional neural networks. Deep CNN architectures can be trained on large datasets and used for tasks such as images recognition, images classifications and objects detection. To perform these tasks, the neural net-

work is used first as a powerful feature extractor of both low-level and high-level features and then as a classifier which uses these features to associate an image to a category of the dataset. Today, there are available many pretrained CNN models ready to fulfill several tasks. In this section it is described a typical CNN architecture and furthermore the InceptionV3 model used in the Rotman et al. approach [48].

2.4.1 Artificial neural network

An Artificial Neural Network (ANN) is an information processing model structured as a collection of connected units or nodes, called artificial neurons, which reminds the structure of the biological brain. Each connection, like the synapses in the brain, can send a signal to other nodes. Typically, an ANN is represented as a forward fully connected network where each node in a layer is connected to all the nodes in the following layer. The starting layer is called input layer, the last layer is called output layer and between them there are several layers, depending on the complexity of the network, called hidden layers. Each layer is responsible for identifying a specific behavior of the input data. The nodes take as input a product between the input data and the weights which characterize each connection. These input-weight products are then summed and filtered with a non-linear function which determines what data should pass to the next layer. Neural networks learn things in the same way as the brain, typically by a feedback process called back-propagation. Usually a part of the dataset is used for the learning process where the weights of the network are continuously adjusted to reduce the difference between the output of the network and the output it was meant to produce.

2.4.2 Convolutional neural network architecture

A CNN can be defined as a deep learning algorithm which can distinguish the various aspect in the input image, associating learning weights and biases to them. Since

RGB images are 3 bi-dimensional matrices of pixel values stacked together, there are many parameters to consider when an image is processed. The role of the CNN is to reduce the image complexity by applying relevant filters, without losing features which are critical for obtaining a good result. The main advantage of using a CNN is the ability to learn these filters without manual intervention. It is also important to use an architecture which is not only good at learning features but also is scalable to massive datasets in order to speed up the process.

As it can be seen in Figure 2.1, the network can be divided in two parts: the feature extractor and the classifier. The main steps involved in the network are convolution, activation, pooling and classification.

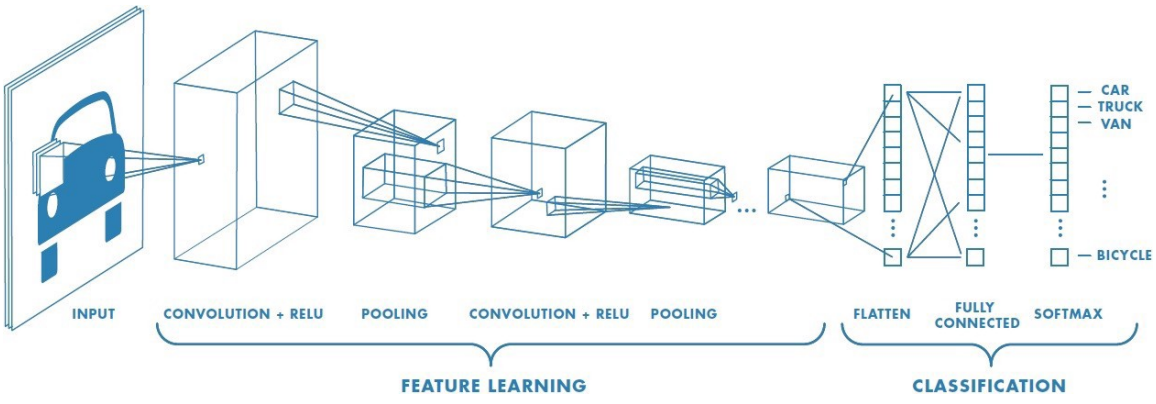


Figure 2.1: Structure of a CNN

Convolution

As it said before in this chapter, the input image can be considered as a matrix of pixel values. The input matrix is convolved with another matrix called filter or kernel. For simplicity, to explain the convolution it is assumed that the input image and the filter are bidimensional matrices. The convolution consists of sliding the kernel matrix over the input matrix by, as example, one pixel and for every position the element wise multiplication is computed. The multiplication outputs are summed

to get the final value which forms a single element of the output matrix called feature map or activation map.

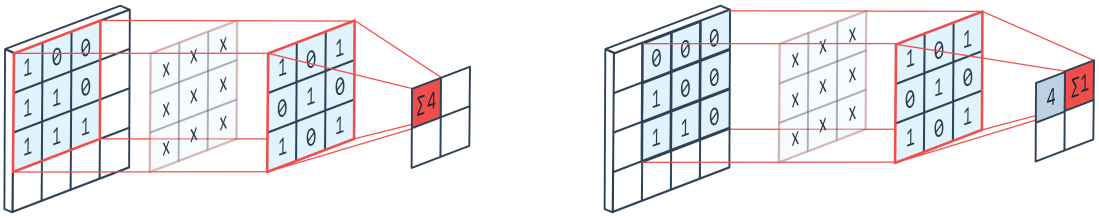


Figure 2.2: Convolution operation

The size of the feature map depends on three parameters called stride, padding and depth. Stride indicates how much the kernel should slide over the input matrix. When the stride is 1, then the kernel moves by one pixel at a time. When the stride is 2, the filter jumps 2 pixels at a time and so on. Consequently, as the stride increases, the dimension of the feature map reduces. Padding consists of increasing the size of the input matrix by adding zeros around the borders. This operation is useful when the reduction of the dimension of the feature map can cause a loss of information along the borders of the input space. As result, the dimensions of the feature map are increased. In general, given an input of size W and a kernel of size K, the size of the output O produced by the convolution is given by the following formula

$$O = \frac{W - K + 2P}{S} + 1 \tag{2.4}$$

where P is the padding and S is the stride. The depth of the feature map is controlled by the number of filters applied. For example, if three filters are applied at the same time, they produce three different feature maps. The result is given by a 2D-matrix of depth three because the different feature maps are stacked together. In the case of RGB images (depth three), the kernel has the same depth as that of the input image.

Generally, the purpose of the convolution operation is to extract the high-level features from the input image. Since different values on the kernel will produce different feature maps for the same input image, it is possible to perform operations such as edge detection, sharpen and blur. Usually a CNN is composed by several convolution operations, so the first ones extract low-level features while at the end the network can adapt to recognize high-level features. For example, the network may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to detect higher-level features, such as facial shapes in higher layers. In principle, by increasing the number of convolution steps, the network can learn more complicated features.

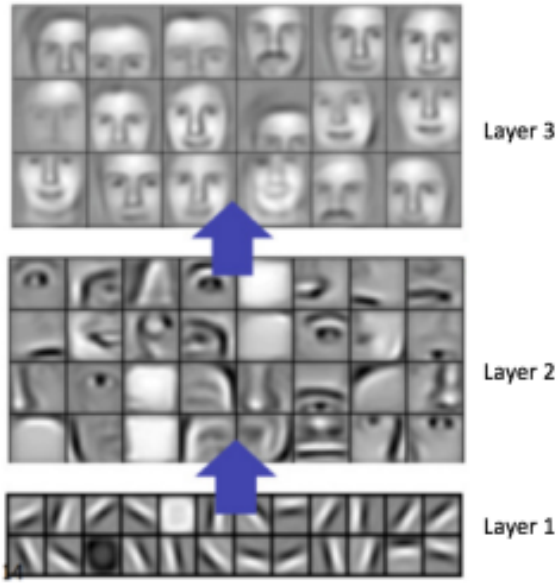


Figure 2.3: Feature extracted from each layer

Activation

The convolution operation is always followed by the application of a non-linear, monotonic increasing and bounded function. The most commonly used function is the Rec-

tified Linear Unit (ReLU). ReLU is an operation applied per pixel which replaces all negative pixel values in the feature map with a zero. Since the convolution is a linear operation, the ReLu introduces non-linearity in the network. This is done because most of the real-world data processed by a CNN are non-linear. Other non-linear functions are tanh and sigmoid.

Pooling

Pooling, also called down sampling, is an operation that aims to reduce the size of the feature map by keeping the most important visual information. The pooling function summarizes a neighborhood of the dimension $n \times n$ into one value. This operation reduces the number of parameters and computations by making the network invariant to small transformation in the input image. There can be two possible type of pooling: max pooling and average pooling. Max pooling returns the maximum value of the pixels from the portion of the image covered by the kernel. It also discards altogether the noisy activations and performs de-noising, along with dimensionality reduction. Average pooling returns the average of all the values from the portion of the image covered by the kernel.

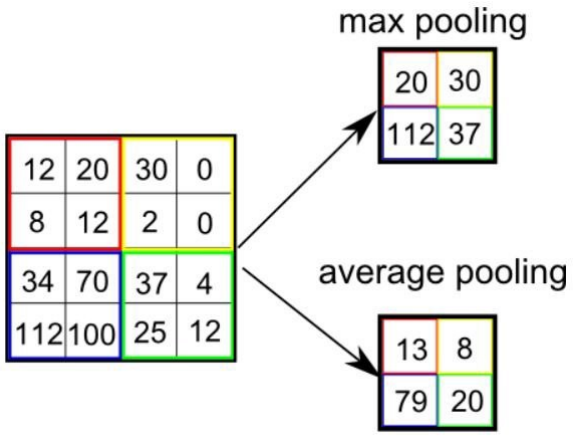


Figure 2.4: Max and average pooling

Classification

In this part of the network the features obtained from the previous operations (convolution, padding and pooling) are managed to classify the input image into various classes based on the training dataset. It is important to underline that this part of network will be removed in our work because the CNN will be used as a feature extractor.

The classification is performed thanks to a Fully Connected layer which is a traditional Multi-Layer Perceptron that uses a SoftMax activation function in the output layer. The role of the SoftMax function is to take a vector of arbitrary real-valued scores and transform it to a vector of values between zero and one.

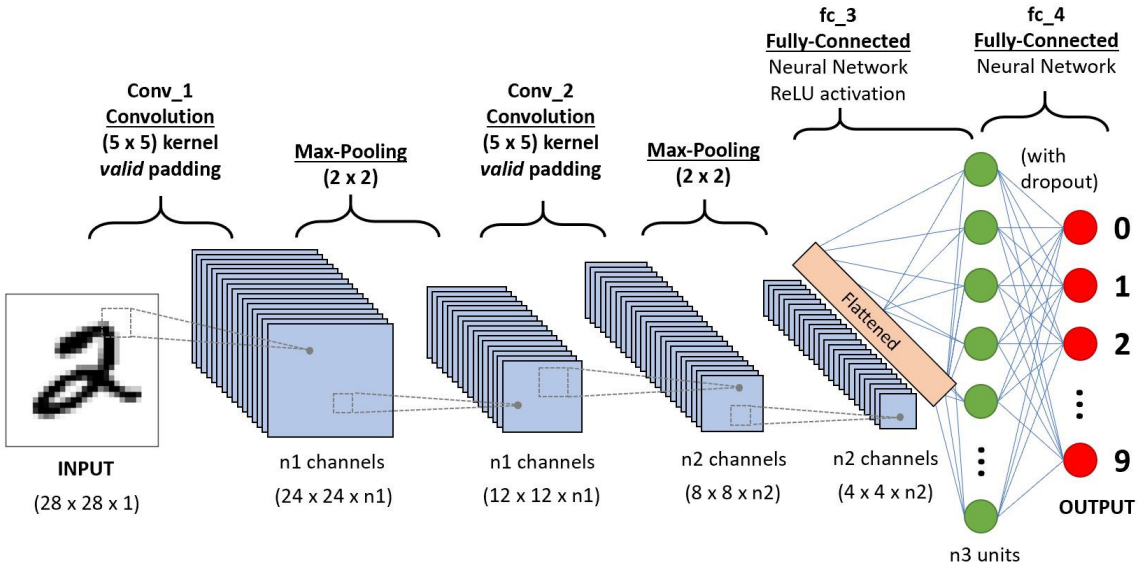


Figure 2.5: Complete CNN architecture

2.4.3 InceptionV3

The CNN selected for this work is the InceptionV3 model [57]. This model is the 3rd evolution of the architecture GoogLeNet (InceptionV1)[56] developed by Google. This network is 42 layers deep, with a computational cost only about 2.5 times higher than GoogLeNet and much more efficient of the VggNet [54]. It became the 1st Runner Up for image classification in ImageNet Large Scale Visual Recognition Competition (ILSVRC)[1] 2015. This kind of network performs very well in big data scenarios where huge amount of data need to be processed at reasonable cost or in scenarios where the memory or computational capacity is limited.

Inception Modules

In a traditional CNN each type of layer (convolutional, pooling) extracts a different kind of information. For example, the information gained from a 5×5 convolutional kernel is different form the one obtained from a 3×3 kernel which in turn is different from the one obtained by the pooling. As it is said before in this chapter, increasing the number of convolution layers lead the network to learn more complex features, so one way to improve the performance of the model is to add more convolutional layers of different size. For this purpose, the inventors of Inception model introduced the inception module. The idea behind the inception module is to compute multiple transformations in parallel and concatenate their results afterwards. In each module, it is performed a 5×5 convolutional transformation in parallel with a 3×3 convolutional transformation and a pooling (Figure 2.6).

[1] http://www.image-net.org/challenges/LSVRC/

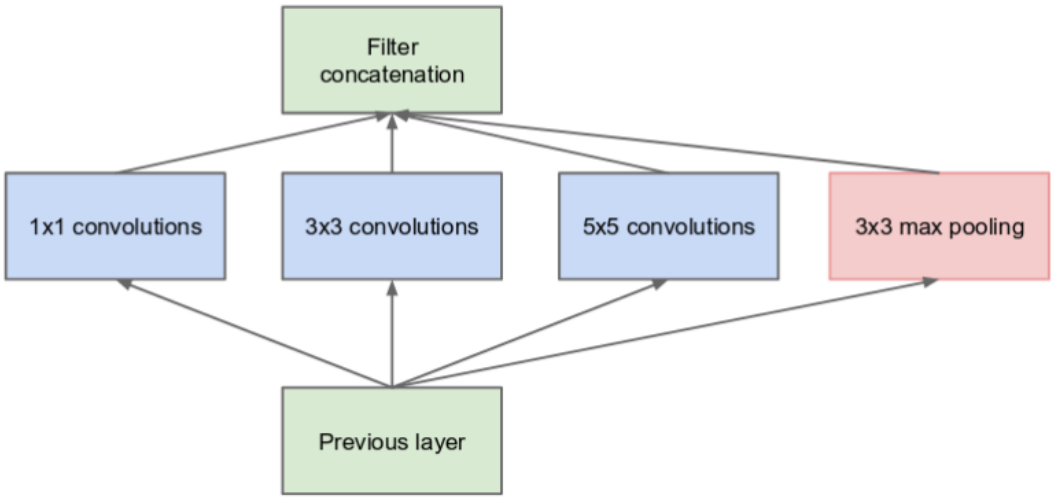


Figure 2.6: Inception module type 1

By increasing the depth of the model, the main drawback is that the computational cost become too expensive. "Any uniform increase in the number of filters in a convolutional layer results in a quadratic increase of computation" [56]. To limit the computational cost, the inception module is modified by adding an extra 1×1 convolution before the 3×3 and 5×5 convolution (Figure 2.7). Although the number of operations is increased respect to the previous version of the module, the 1×1 convolutions are cheaper than the 5×5 and reduce the number of input channels. For example, using twenty units of 1×1 filters, an input of size $64 \times 64 \times 100$ can be compressed down to $64 \times 64 \times 20$. In this way, the inception module results simultaneously "deep" (many layers) and "wide" (many parallel operation).

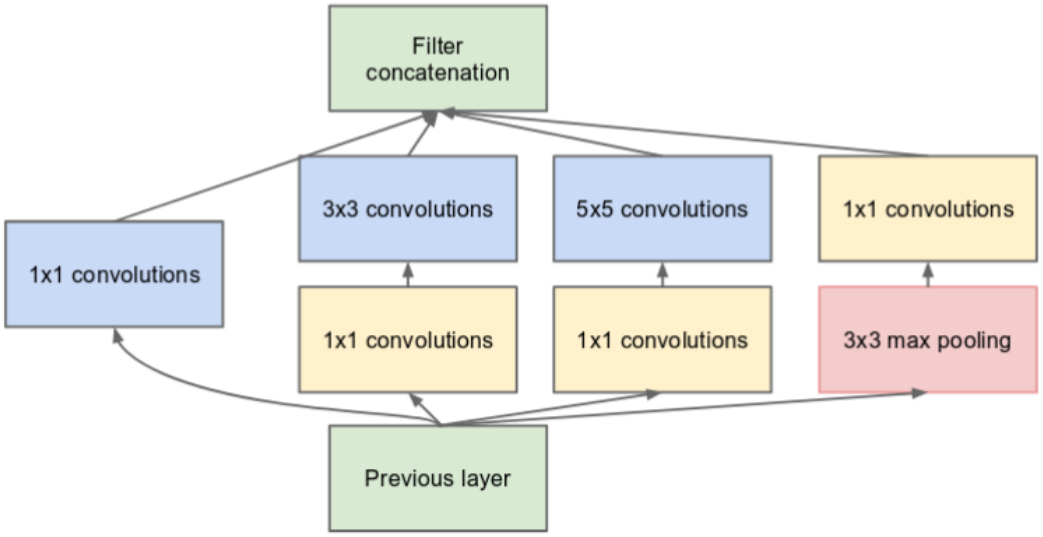


Figure 2.7: Inception module type 2

The inception layer is further revised by applying convolution factorization (Figure 2.8). The 5×5 convolution layers are substituted by two 3×3 convolution layers. In this way the number of parameters is decreased from $5 \cdot 5 = 25$ to $3 \cdot 3 + 3 \cdot 3 = 18$ reducing the computational complexity. Moreover, it softens the reduction of the input dimensions avoiding any loss of information.

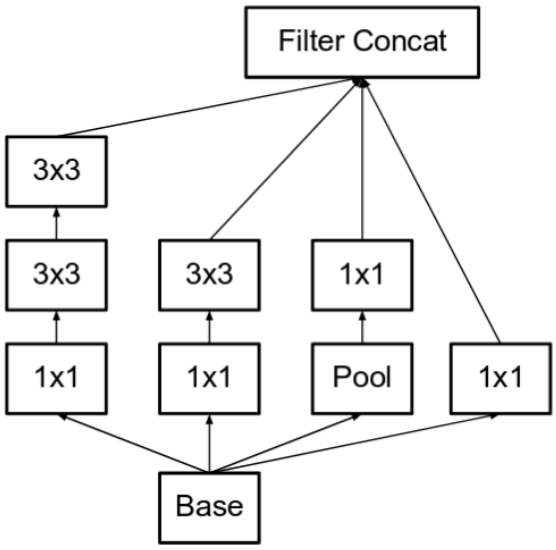


Figure 2.8: [57] Inception module type 3

The convolution factorization is then extremized by decomposing the $n \times n$ convolutions into asymmetric $1 \times n$ and $n \times 1$ convolutions (Figure 2.9). As example, 3×3 convolutions, which involve $3 \cdot 3 = 9$ parameters, are transformed into a 3×1 convolution followed by a 1×3 convolution, which involves $3 \cdot 1 + 1 \cdot 3 = 6$ parameters. As it is shown, the number of parameters reduces by 33%.

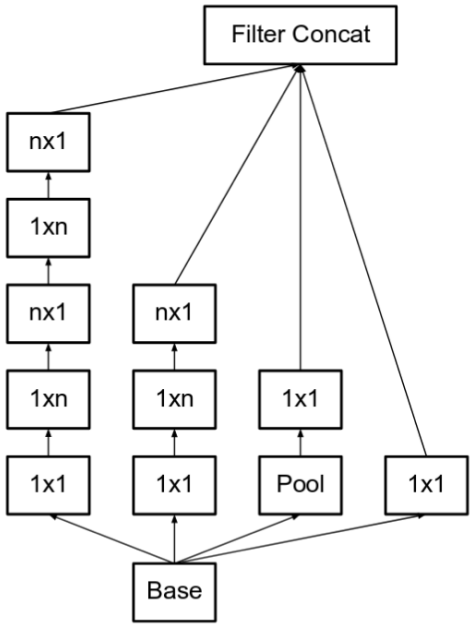


Figure 2.9: [57] Inception module type 4

As marked in Figure 2.10, the $1 \times n$ and $n \times 1$ filters are then executed in parallel instead of in sequence.

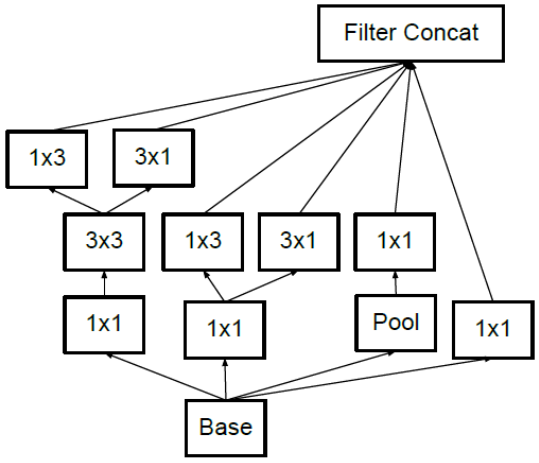


Figure 2.10: [57] Inception module type 5

All the types of inception layers described above are the key elements of the InceptionV3 architecture.

Efficient grid size reduction

The dimension of the feature map produced by the convolutional layers is usually downsized by applying a pooling operation, as it is explained in Section 2.4.2. This operation involves the transformation in dimension of the input matrix from a $n \times n$ matrix with k channels to a $\frac{n}{2} \times \frac{n}{2}$ matrix with $2k$ channels. The overall computation is $2d^2k^2$ which is too expensive. Moreover, by switching the position of the convolutional layer with the pooling one (first applying pooling and then the convolution layer), even if the computational cost is reduced, the expressiveness of the network is drastically diminished. To solve this issue, a convolutional layer and a pooling layer are used in parallel, both with stride 2. This improvement is called efficient gird size reduction and leads to a less expensive but still efficient network.

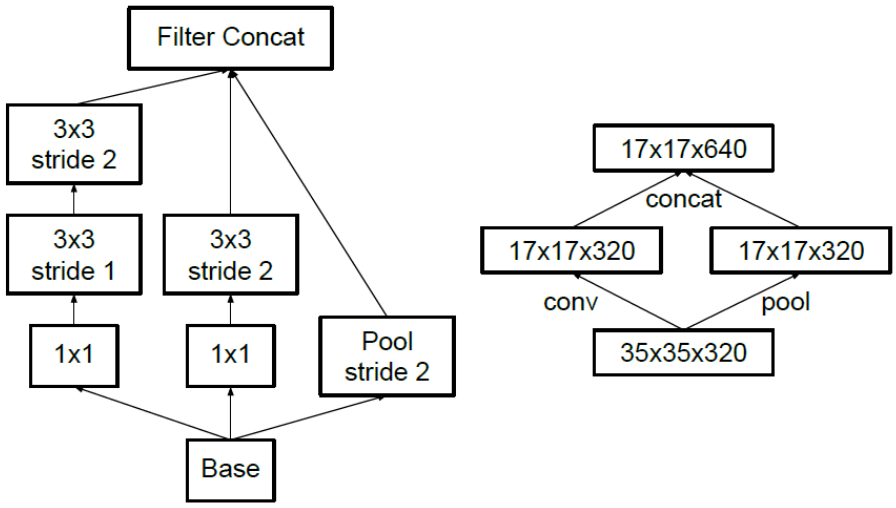


Figure 2.11: [57] Efficient grid size reduction

InceptionV3 architecture

All the improvements described above are employed in the InceptionV3 model. The layout of the model is shown in Figure 2.12. There are 3 inception modules type 3 which takes as input a $35 \times 35 \times 288$ matrix. These are reduced to a $17 \times 17 \times 768$ matrix by involving the effective grid size reduction. Then are applied 4 instances of modules type 4, followed by a grid size reduction which gives a $8 \times 8 \times 1280$ output. This output is managed by two inception modules of type 5. The layer before the classifier produces as output a vector of 2048 features. This network is trained on ImageNet database and it can classify images into 1000 object categories, by taking 299×299 images as input.

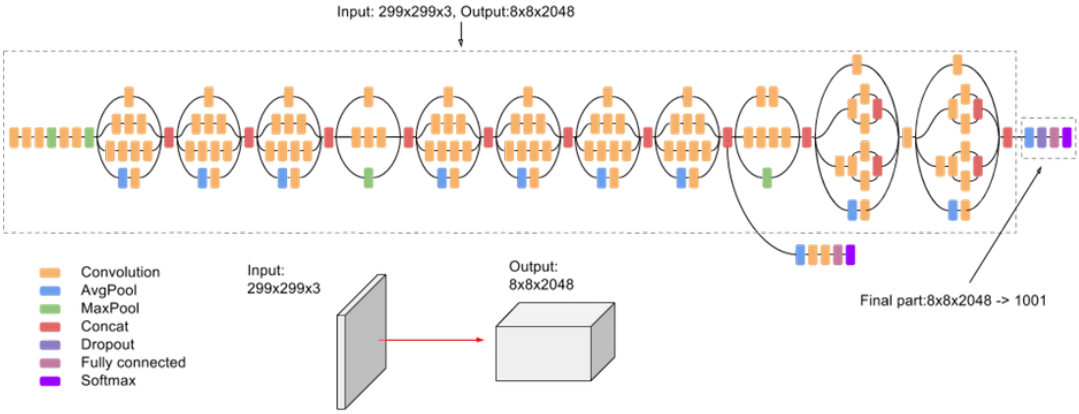


Figure 2.12: InceptionV3 architecture

2.5 Datasets

One of the main drawbacks in scene detection task is the lack of large open source datasets. Since long heterogeneous videos that consist of multiple scenes are typically copyrighted material, most of the methods proposed in literature use personal test sets which makes it hard to reproduce or compare the performances of different methods.

It must be noted that this is a questionable practice from a science theory point of view [19]. It belongs to the basic principles of modern science that experiments must be reproducible by others.
The following are some of the datasets used in literature.

2.5.1 Open Video Scene Detection dataset

Rotman et al. [46][47] proposed a dataset created from Creative Commons licensed videos freely available for download and use. The dataset is composed by 21 freely available videos from a variety of genres. There are either short or full-length movies including but not limited to animation, documentary, drama, crime, comedy, and sci-fi. Each video is provided with the ground truth scene division gained from the director script or from the work of several independent human annotators. The different divisions were aggregated and validated one against the other to eliminate biases.

2.5.2 Rai dataset

Baraldi et al. [5] proposed several scene detection methods that use a collection of ten videos taken from the Rai Scuola video archive as dataset. The videos are mainly documentaries and talk shows. This dataset together with the related scene annotations of each videos are freely downloadable.

2.5.3 BBC Planet Earth dataset

The same authors proposed in the work [6] a dataset that contains eleven episodes from the BBC documentary series Planet Earth. Each episode is approximately 50 minutes long, and the whole dataset contains around 4900 shots and 670 scenes. Each video has been labelled by five different annotators.

2.5.4 TRECVID dataset

The TRECVID[2] initiative is a series of workshops that aims to promote research in content-based video retrieval and analysis. To accomplish such purpose large test collections, realistic system tasks, uniform scoring procedures and a forum for organizations interested in comparing their results are provided.

2.5.5 YouTube-8M dataset

YouTube-8M[3] is a large-scale labeled video dataset that consists of millions of YouTube video IDs, with high-quality machine-generated annotations from a diverse vocabulary of about 3,800 visual entities. The (multiple) labels per video are Knowledge Graph entities, organized into 24 top-level verticals. Each entity represents a semantic topic that is visually recognizable in video, and the video labels reflect the main topics of each video.

[2]https://trecvid.nist.gov/

[3]https://research.google.com/youtube8m/

Chapter 3

Methodological approach

Summary

In this chapter all the implemented methodologies are reported. Starting from the workflow proposed by Rotman et al. (1.3), several methods are built, tested and evaluated in order to improve the performances. During the process of recreating the Rotman et al. approach, some assumptions are taken:

- *Estimation of scenes*, we decide to not estimate the number of scenes of the videos. Since the scene detection task is typically a post production process, the correct number of scenes is assumed to be known. Such value is given by the length of the ground truth division provided by the employed datasets.

- *Features*, our scene detection algorithm only relies on the visual features for computing the scene division. We decide to focus on the visual part rather than the auditory one because it is the prevalent and more promising in literature. The feature extraction phase is performed by considering only the frames extracted from the shots, therefore we created a single distance matrix which encodes the visual similarity between the shots.

- *Hierarchy creation*, since this task is optional in the Rotman et al. approach and does not affect the performances of the scene detection workflow, we choose to not implement the non-greedy algorithm for creating a hierarchical division tree.

Following the order of the steps proposed in Rotman et al. workflow, first the shot detection step is studied: we compared the performance of the method involved in Rotman workflow (3.1.1) against a more recent method proposed in literature (3.1.2). Next, an exhaustive research is proposed for the frame selection phase (3.2), followed by the description of the feature extraction process (3.3) taken from Rotman. The core of the proposed methodologies is represented by the reformulation of the dynamic programming algorithm responsible for the scene detection. This leads to the implementation of *RS* (3.5.2) and *RS_B* (3.5.3) which underlines a performances improvement. A graphical representation of our workflow is reported in Figure 3.1.

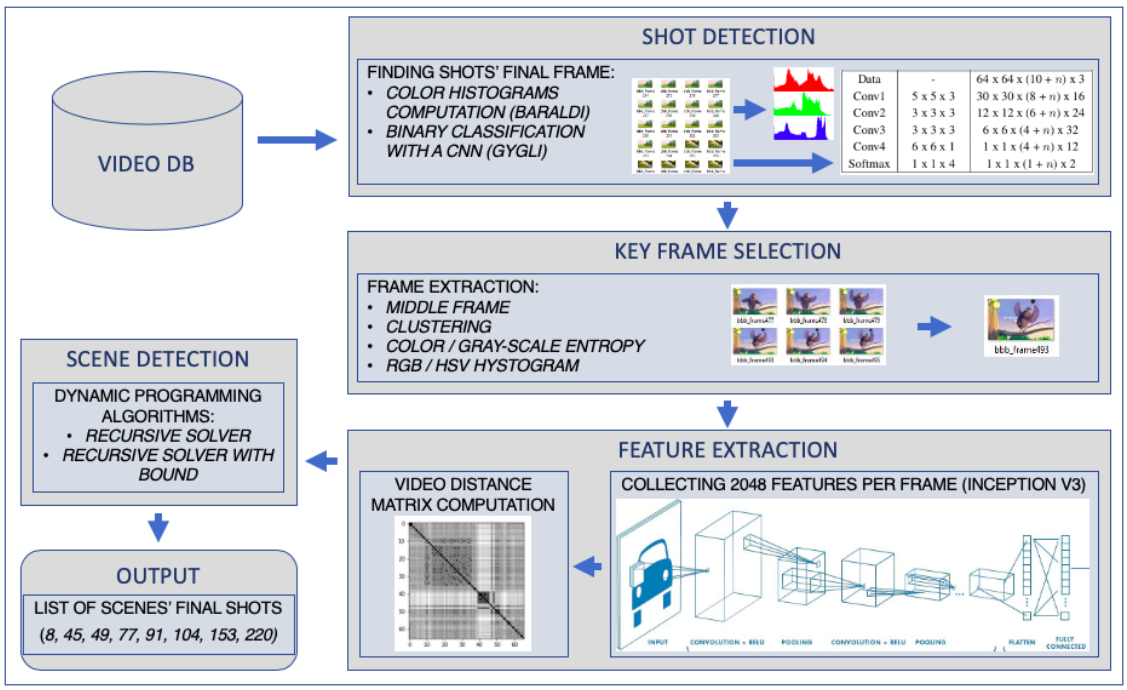


Figure 3.1: Proposed RS *and* RS_B *workflow*

3.1 Shot detection

Since shot boundary detection is a fundamental part of the scene segmentation workflow but not the main subject of our work, we decide to not implement a shot detection algorithm from scratch. Despite the fact that nowadays it is considered a solved problem, shot detection is a challenging task for an automatic algorithm, due to several reasons. Video editors often try to make shot transitions subtle, so that they are not distracted from the video content. In some cases the transitions are completely hidden. Then, videos show strong variations in content and motion speed. In particular fast motion, leading to motion blur, is often falsely considered a shot change. Finally, transitions such as dissolves have small frame to frame changes, making it difficult to locate them, especially with traditional methods. To overcome these problems, we tested two methodologies that are very different in their implementation.

3.1.1 Baraldi et al. algorithm

This algorithm, presented in [8], is based on an extended difference measure that represents the change in the content of two different frames or half-frames in the video. To discover the occurrence of hard cuts and gradual transitions, the extended difference measure is filtered with specified thresholds and parameters. This approach assures high accuracy levels, while yielding low execution times.

Parameters

Given two successive shots in a video sequence, the transition length L is defined as $s - e - 1$, where e is the ending frame of the first shot and s is the starting frame of the second shot. Therefore, the center of a transition is given by $n = \frac{e+s}{2}$ that may be an inter-frame position in case of a non integer value. An abrupt transition is identified by $L = 0$ so its center n is always a non-integer value. Then the extended

difference measure M_w^n is defined, centered on frame or half-frame n with $2n \in N$ and with frame-step $2w \in N$ as:

$$M_w^n = \begin{cases} d[F(n-w), F(n+w)], & \text{if } n+w \in \mathbb{N} \\ \frac{1}{2}\left(M_w^{n-\frac{1}{2}} + \frac{1}{2}M_w^{n+\frac{1}{2}}\right), & \text{otherwise} \end{cases} \tag{3.1}$$

The first term of the expression is composed by $d[F(i), F(j)]$ which is the distance in feature between frame i and frame j computed as a linear combination of the sum of squared differences of frames i and j and of the χ^2 distance of color histograms extracted from frames i and j, both normalized by the number of pixels in a frame. Since $F(i)$ express a feature that describes the single frame i, it cannot be directly computed at half-frames. Therefore, in case of inter-frames, a linear interpolation is used as it shown by the second term of the expression.

Implementation

The first step of the algorithm aims to find a set of candidate positions for transitions by thresholding the extended difference measure M_w^n at all frames and half frames positions with a window $w = 0.5$. In practice the threshold is called T and it is set to 80. The list of possible transition $C = t_i = (f_i, l_i)$ is computed by the aggregation of adjacent candidate position, where f_i is the first portion of the transition and l_i the second one. The main purpose of this operation is to identify hard cuts, but some false positives may be produced.. The list of possible transitions is validated by computing the variation in difference values between the transition and the adjacent shots. For this scope, a new index called $Peak$ is defined as:

$$Peak_w(t) = \max_{f \leqslant n \leqslant l}(M_w^n) - \min(M_w^{f-2w} + M_w^{l+2w}) \tag{3.2}$$

Whenever a remarkable variation is registered on at least one side of the candidate transition, the candidate is validated. In practice the threshold for $Peak_w(t)$ called T_p is set to $\frac{T}{2}$. In order to identify gradual transitions, it is necessary to repeat the

previous step with increasing values of w until a max value W. This procedure could invalidates previously found transitions because other positions could overcome the threshold values. To avoid this behavior, a limit of frames T_s is defined where before and after this value, the validated transitions are not further analyzed.

3.1.2 Gygli algorithm

Gygli [23] propose a shot detection method based on a CNN that conceives such task as binary classification problem. It is an end to end model, from pixel to final shot detection.

It integrates a training part that consist in creating a dataset with automatically generated transitions such as cuts, dissolves and fades from a set of videos. To efficiently process videos, a CNN, that is fully convolutional in time and that allows to use a large temporal context without the need of repeatedly processing frames, is involved. This architecture is stated to obtain state of the art results, running at unprecedented speed (+120 time faster real-time).

Training

To tackle the most difficult and unsolved shot boundary detection challenges, such as dissolves, subtle transitions, motion blur and flashes, a novel and efficient fully convolutional neural network architecture, inspired from the one used for image segmentation, is involved.

A new large-scale dataset composed by one million frames and automatically generated labels is created in order to train the network: hard cuts, crop cuts, dissolves, wipes, fade transitions are generated over the chosen videos along with none-transition examples and frames with added artificial flash, in order to make the model invariant to flashes.

This configuration allows to train the network without manually annotate the shot

boundaries of the video.

To generate training and validation data, 2 YouTube[1] playlists (LongTakes1[2] and LongTakes2[3]) of approximately 188 video are used. These playlists contain long single-shot scenes taken from film, TV-shows, documentary, commercials, surveillance and generic YouTube videos. Each video is sampled in 10 snippets of 64×64 RGB frames: in order to automatically produce shot changes some of these snippets are combined with artificial transitions (cut, dissolve) and flashes.

At the end of this process two types of training examples are generated: snippets consisting of frames from a single shot (non-transitions) and transition snippets, which have a transition from one shot to another. The CNN takes as input groups of 10 frames at time and classifies if the two center frames of the group are part of the same shot or if there is an ongoing transition.

Network characteristics

In this approach, shot boundary detection is posed as a binary classification problem: the goal is to correctly predict if a frame is part of the same shot of the previous frame or not.

[1] https://www.youtube.com/

[2] https://www.youtube.com/playlist?list=PLxf1dxhJ3H9orru0qzPy1j5VDa41c4x7Z

[3] https://www.youtube.com/playlist?list=PLxf1dxhJ3H9pzLItmYdDeBQa0RE8zmsC3

Layer	Kernel size (w,h,t)	Feature Map (w,h,t,channels)
Data	-	$64 \times 64 \times (10+n) \times 3$
Conv1	$5 \times 5 \times 3$	$30 \times 30 \times (8+n) \times 16$
Conv2	$3 \times 3 \times 3$	$12 \times 12 \times (6+n) \times 24$
Conv3	$3 \times 3 \times 3$	$6 \times 6 \times (4+n) \times 32$
Conv4	$6 \times 6 \times 1$	$1 \times 1 \times (4+n) \times 12$
Softmax	$1 \times 1 \times 4$	$1 \times 1 \times (1+n) \times 2$

Table 3.1: Gygli CNN layers

The net architecture is a compact version of Convolutional 3D(C3D)[59], implemented in TensorFlow[4], that consists of 3D convolutions only (each followed by a ReLU non-linearity) and avoids fully connected layers. This makes the network fully convolutional in time.

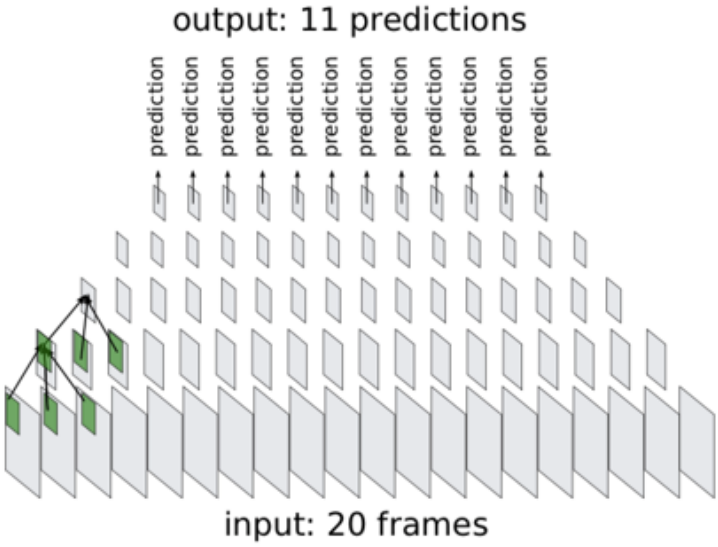


Figure 3.2: [23] Gygli CNN architecture

[4]https://www.tensorflow.org/

The network takes in input 10 frames and it is trained to predict if frame 6 is part of the same shot as frame 5. It is possible to increase the input size due to its fully convolutional architecture: providing 20 frames, the network would predict labels for frames 6 to 16 (making 11 predictions), minimizing redundant computation and allowing to obtain large speedups at inference. In the same way, by analysing 100 frames, 91 predictions are produced: at inference, the videos are processed in snippets of 100 frames, with an overlap of 91 frames.

3.2 Frame selection

The frame selection step consists on finding a suitable subset of the total frame images extracted from the video.
The feature extraction step will take each image of the selected subset as input and it will extract a visual feature vector in order to compute the dissimilarity between each frame. Therefore, both the quantity and quality of the selected frames could affect the performance of the scene detection algorithm.
We implemented several method of frame selection by focusing on both the quantity and quality of frames to be chosen. We tried both stochastic and deterministic methods of selection in order to discover if exists a method which improves the performance of the scene detection task respect to the one proposed in Rotman et al. [48].

3.2.1 Middle frame method

Rotman et al. proposed a method that consists on extracting the central frame of each shot and then computing the corresponding feature vectors. The floor operator is used in case of shots with an even number of frames. This method is used as a benchmark for the evaluation of the following methodologies.

3.2.2 Clustering method

We created a method based on the k-means algorithm, that is used to identify the most significant frames for each shot. As first step, the feature vectors of each frame of the video are extracted, standardized and opportunely separated based on the shot they belong to. Then the k-means algorithm is performed. It identifies the optimal number of clusters by trying to group data in 1 to 6 clusters of frames for each shot. The optimal number of clusters is chosen by validating the performance of each division. For each cluster, the centroid is found.

The choice of the key frames is proposed by exploring several solutions. One method is to select as key frames the feature vectors closest to each centroids. Therefore, if the number of optimal clusters for the analysed shot is n, n-key frames are extracted from this shot. This results to be the most promising option between the alternatives proposed below, so it is the one reported in the results related to the frame selection phase (4.4). A variation of such approach is to extract the feature vectors that correspond to each centroid. In this way, it is like n dummy frames are used for the feature extraction. A further variation consists on averaging the feature vectors corresponding to each centroids in the cluster.

Finally, another variation is based on grouping the frames of each shot in a single cluster and identifying the centroid. The frame nearest to the centroid is selected as key frame for the feature extraction.

3.2.3 Histogram methods

Such method relies on studying the colour information in the shots by computing and comparing the colour histogram of each frame. Since colour is not greatly affected by the spins or movement of the input image, it is one of the most relevant features in image processing. As proposed in [30] and [26], each shot S_z is represented by a set of key frames K_z. As first step, the middle frame method is applied to the

shot, therefore the middle frame is chosen as key frame and added to K_z. Next, each frame of the shot is compared against the frames in the set K_z: if the result of the comparison is lower than a certain threshold and if such behaviour is repeated respect to all the frames in K_z, such frame is added to the set K_z. The comparison function is the histogram intersection and it is defined as:

$$d(H_x, H_y) = \sum_{h \in Allbins} \min(H_x(h), H_y(h)) \tag{3.3}$$

where H_x and H_y are respectively the histograms of the frame x and frame y and h is the individual bin of a histogram. Two methods which exploits the formula described above are explored. The first one uses the RGB space for the colour histograms H_x and H_y of the frames. The second method uses the HSV space for the same colour histograms of the frames. Both methods use eight bins in each dimension.

3.2.4 Entropy methods

This key frame extraction method is based on computing the entropy of an image. Entropy is a is defined as:

$$E_y = -\sum_y H_y \log(H_y) \tag{3.4}$$

where H is the is the normalized histogram of image's pixels intensity. Two methods which exploit entropy are explored. In the first one the entropy is computed by using the formula above where H is the is the normalized histogram of grey-scale image's pixels intensity. The frame which scores the max entropy value is selected as key frame. In the second one, the entropy is computed by using the normalized histogram of RGB image's pixels intensity. Then, two types of selection are experimented on those methods. The first consists on selecting as key frame the one that scores the max entropy value. The second one is similar the one described in the Section 3.2.3, where each shot is represented by a set of key frames. The first frame of each shot is selected as key frame and added to the key frame set. Then, the entropy difference

between each two consecutive frames is computed within the shot. If the difference exceeds a certain threshold, the second frame in the difference is added to the key frame set.

3.2.5 Other methods

The following are the other deterministic and stochastic methods of key frame selection that we implemented from scratch. They result less effective then the above ones and are not part of the final test phase:

- *Three frames*: this method is an evolution of the middle frames (3.2.1), it consists on selecting 3 frames for each shot: one in the first half (1/4 of the total frames), one in the middle and one in the second half (3/4 of the total frames). The aim is to obtain features that could represent the top part, the middle part and the lower part of the shot in order to give an estimation of the possible visual changes all over the shot.

- *Fixed number of frames*: this method is a further evolution of the three frames extraction in a semi deterministic way. The number of frames to be extracted from the whole video are selected based on a percentage computed over the total number of frames of the video. The number of frames to be extracted from each shot is given by computing the division between the length of each shot and the total number of frames of the video. For each shot with at least two frames, the first frame is taken and the rest of frames of the shot are taken with a fixed interval. This interval is defined by the division between the shot length and the number of frames of each shot. For each shot with only one frame to be extracted, the middle frame is selected. The result is to extract a fixed number of frames where each frame has a fixed distance from each other in the same shot.

- *Random percentage of frames*: this method is based on a random extraction of frames. The number of frames to be extracted is determined by computing a percentage of the total number of frames. The single images are randomly extracted all over the whole video without repetitions meeting the constraint to have at least one frame for each shot. In this way larger shot are more likely to have a larger number of frames extracted. Increasing percentage of extracted frames are employed. The goal is to see how much randomicity affects the result in order by verifying if there is a trend related to the percentage of frames or if the score saturates after a certain percentage.

- *Frame dynamics*: this method is implemented in two versions and it can be performed immediately after the application of one of the frame selection methods explained before in this section. The first version consists on creating a fixed window of frames. For each frame, the window is composed by the 11 images before and after the selected frame and by the selected frames itself. The pixel values of each image of the window are normalized between values $[-1, 1]$ and then averaged together.The second version considers a variable window of frames. The half-length of the window is chosen for each shot by computing the minimum between the distance between the selected frames dived by a fixed number and the frame per second value of the video. In both versions, after the length of the window is decided, the pixel values of each image of the window are normalized between values [-1,1] and then averaged together in order to create a single frame. Therefore, the frame extraction step receives as input a series of dummy images composed by the average of the pictures considered in the window. The main difference between the two versions is that in the second case the windows never overlap. The aim of both methods is to track the differences caused by the motion in the shot and see how such behaviour could affect the results.

3.3 Feature extraction using InceptionV3

3.3.1 ImageNet

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22.000 categories. At least one million of the images are also provided with bounding boxes that surrounds the subject of the photo. The images were collected from the web and labeled by human annotators. Each category of images consists of several hundred images. The project to develop and maintain the dataset was organized and executed by a collaboration between academics at Princeton, Stanford, and other American universities. Between 2010 and 2017 a subset of ImageNet dataset was employed in an annual competition called ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This challenge focuses on multiple tasks such as image classification, single-object localization and object detection. Typically, in this challenge the training dataset was composed of 1 million images, with 50.000 for a validation dataset and 150.000 for a test set. All the most common convolutional neural networks such as AlexNet, VGG and Inception are presented in those challenges and evaluated on ImageNet dataset. Thanks to such challenges, the pretrained weights on ImageNet dataset of many CNN models are available and can be easily loaded and used for prediction, feature extraction, and fine-tuning.

3.3.2 Feature extraction

For the purpose of this work, we used the inceptionV3 model as a feature extractor. In practice, we built the extractor by loading the pretrained inceptionV3 model on the ImageNet dataset, removing the final classication layer of the network and extracting the next-to-last layer of the CNN. As result, the network extracts a 2048-dimension vector for every image taken as input. The feature extraction is applied to the key frames obtained from the feature selection phase. This results in having, for each

key frame, a 2048-dimensional vector that represent it, allowing to measure distances between shots in a semantic and more meaningful way.

3.3.3 Feature aggregation

The extracted 2048-dimension vectors are used to compute the similarity between the frames selected by the frame selection algorithm. If the frame selection produces one key frame for each shot, N vectors are extracted from the video (one for each key frame) where N identifies the number of shots of the video. Those vectors are simply stacked together in chronological order of extraction to create a $N \times 2048$ matrix. The first feature vector represents the first shot and it becomes the first row of the matrix, the second feature vector represents the second shot and it becomes the second row of the matrix and so on. The Euclidean distance is performed between each line of such matrix: the first line is measured against all the other lines to create a $N \times N$ symmetric distance matrix called D with 0 values on the main diagonal. This matrix is required as input by the scene detection algorithm (3.4.2). If the frame selection produces $m > 1$ key frames for each shot, we apply an aggregation process since the scene detection algorithm requires as input an a $N \times N$ matrix. Supposing $M > N$ key frames are selected in the frame selection phase, an $M \times 2048$ matrix is obtained after applying the feature extraction procedure. Again the Euclidean distance is performed obtaining a $M \times M$ matrix. Next, by considering the number of key frames of each shot, several sub-matrices taken from the $M \times M$ matrix are aggregated to produce a single value that will be placed in the $N \times N$ matrix. The aggregation is performed by selecting the max value of each sub-matrix. The single value obtained from each sub-matrix is placed in a specific position of the $N \times N$ in order to maintain the symmetry and the meaning of the distances computed before. For example, given a $M \times M$ matrix obtained from a video composed by 4 shot such

as:

$$\begin{bmatrix}
\mathbf{0} & \mathbf{95} & 81 & 57 & 77 & 85 & 49 & 44 & 47 & 46 & 69 & 52 \\
\mathbf{95} & \mathbf{0} & 44 & 95 & 88 & 82 & 90 & 93 & 48 & 73 & 81 & 73 \\
\mathbf{81} & \mathbf{44} & 0 & 40 & 65 & 67 & 70 & 94 & 72 & 76 & 78 & 94 \\
\mathbf{57} & \mathbf{95} & 40 & 0 & 72 & 72 & 59 & 55 & 48 & 94 & 63 & 92 \\
\mathbf{77} & \mathbf{88} & 65 & 72 & 0 & 62 & 91 & 87 & 45 & 63 & 66 & 90 \\
\mathbf{85} & \mathbf{82} & 67 & 72 & 62 & 0 & 91 & 41 & 40 & 43 & 66 & 94 \\
\mathbf{49} & \mathbf{90} & \mathbf{70} & \mathbf{59} & \mathbf{91} & \mathbf{91} & 0 & 83 & 83 & 82 & 64 & 67 \\
\mathbf{44} & \mathbf{93} & \mathbf{94} & \mathbf{55} & \mathbf{87} & \mathbf{41} & 83 & 0 & 42 & 83 & 84 & 80 \\
\mathbf{47} & \mathbf{48} & \mathbf{72} & \mathbf{48} & \mathbf{45} & \mathbf{40} & 83 & 42 & 0 & 57 & 85 & 52 \\
\mathbf{46} & \mathbf{73} & \mathbf{76} & \mathbf{94} & \mathbf{63} & \mathbf{43} & \mathbf{82} & \mathbf{83} & \mathbf{57} & 0 & 78 & 88 \\
\mathbf{69} & \mathbf{81} & \mathbf{78} & \mathbf{63} & \mathbf{66} & \mathbf{66} & \mathbf{64} & \mathbf{84} & \mathbf{85} & 78 & 0 & 80 \\
\mathbf{52} & \mathbf{73} & \mathbf{94} & \mathbf{92} & \mathbf{90} & \mathbf{94} & \mathbf{67} & \mathbf{80} & \mathbf{52} & 88 & 80 & 0
\end{bmatrix}$$

where the amount of key frames related to each shot are respectively $[2, 4, 3, 3]$ (2 key frames for the firs shot, 4 key frames for the second shot and so on).

The $N \times N$ matrix (4×4) obtained by the max aggregation is:

$$\begin{bmatrix}
0 & 95 & 93 & 81 \\
95 & 0 & 94 & 94 \\
93 & 94 & 0 & 85 \\
81 & 94 & 85 & 0
\end{bmatrix}$$

Since this procedure needs to produce a diagonal symmetric matrix where all the elements on the diagonal are 0, the aggregation of the first 2 shot (represented by first and second line of the $M \times M$ matrix and marked in red) must generate the value 0. Therefore, $N(0, 0) = 0$. Next, the aggregation of the second shot (represented by 3^{th} to 6^{th} line of the $M \times M$ matrix and marked in orange) is performed by identifying the submatrix which number of columns is given by the number of key frames of the previous shot and the number of rows is given by the amount of key frames of the

current shot.

$$\begin{bmatrix} 81 & 44 \\ 57 & 95 \\ 77 & 88 \\ 85 & 82 \end{bmatrix}$$

The value obtained from this submatrix is placed in the 2^{nd} row, 1^{st} column of the $N \times N$ matrix and, since it is symmetric, in the 1^{st} row, 2^{nd} column. Then a 0 value is placed on the diagonal, at the intersection between 2^{nd} row and column.

The aggregation of the third shot requires the computation of 2 submatrices. The first one (marked in green) is identified in the $M \times M$ by the columns of the key frames of the first shot and the rows of the key frames of current shot. The second one (marked in blue) is located by the columns of the second shot and the rows of the current shot.

$$\begin{bmatrix} 49 & 90 \\ 44 & 93 \\ 47 & 48 \end{bmatrix}; \begin{bmatrix} 70 & 59 & 91 & 91 \\ 94 & 55 & 87 & 41 \\ 72 & 48 & 45 & 50 \end{bmatrix}$$

The computation results of the 2 submatrices are placed as follow: the first one is located in the 3^{rd} row, 1^{st} column of the $N \times N$ matrix and, since it is symmetric, in the 1^{st} row, 3^{rd} column. The second one is placed in the 3^{rd} row, 2^{nd} column of the same matrix and, since it is symmetric, in the 2^{nd} row, 3^{rd} column. Then a 0 value is located on the diagonal of the matrix, at the intersection between 3^{rd} row and column.

The last shot's 3 submatrices aggregation follows the same pattern.

$$\begin{bmatrix} 46 & 73 \\ 69 & 81 \\ 52 & 73 \end{bmatrix}; \begin{bmatrix} 76 & 94 & 63 & 43 \\ 78 & 63 & 66 & 66 \\ 94 & 92 & 90 & 94 \end{bmatrix}; \begin{bmatrix} 82 & 83 & 57 \\ 64 & 84 & 85 \\ 67 & 80 & 52 \end{bmatrix}$$

3.4 *Rotman18* scene segmentation problem

In this section, we introduce a video scene detection problem formulation and its dynamic programming based solution both proposed by Rotman et al. [48]. Their method involves a normalized cost function to optimal group consecutive shots into scenes. This algorithm provides a temporally consistent division of the video into scenes and it does not require fine tuning of any parameters for different types of content.

3.4.1 Generalities on dynamic programming

Dynamic programming (DP) is a mathematical optimization and computer programming method for solving a complex problem by partitioning it into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions using a memory-based data structure. The two key attributes required to solve a problem with DP are an optimal substructure and overlapping subproblems. Optimal substructure implies that the solution to a given optimization problem can be achieved by the combination of optimal solutions to its subproblems. Such optimal substructures are usually described by means of recursion.

The Shortest Path problem shows an example of optimal substructure property: if a node x lies in the shortest path from a source node u to destination node v then the shortest path from u to v is combination of the shortest path from u to x and the shortest path from x to v.

The overlapping subproblems property implies that any recursive algorithm solving the problem solves the same subproblems over and over, rather than only generating new subproblems.

Computed solutions to subproblems are stored in a table so they are queried as needed without being recomputed. Generally, since DP is heavily based on memory usage,

the space of subproblems must be small in order to achieve acceptable performance. As an example of DP, the problem of computing the n^{th} Fibonacci number $F(n)$, can be broken down into the subproblems of computing $F(n-1)$ and $F(n-2)$, and then adding the two. The subproblem of computing $F(n-1)$ can itself be broken down into a subproblem that involves computing $F(n-2)$. Therefore, the computation of $F(n-2)$ is reused, and the Fibonacci sequence thus exhibits overlapping subproblems. Given a problem where DP applies there could be two different ways of exploring the subproblems structure:

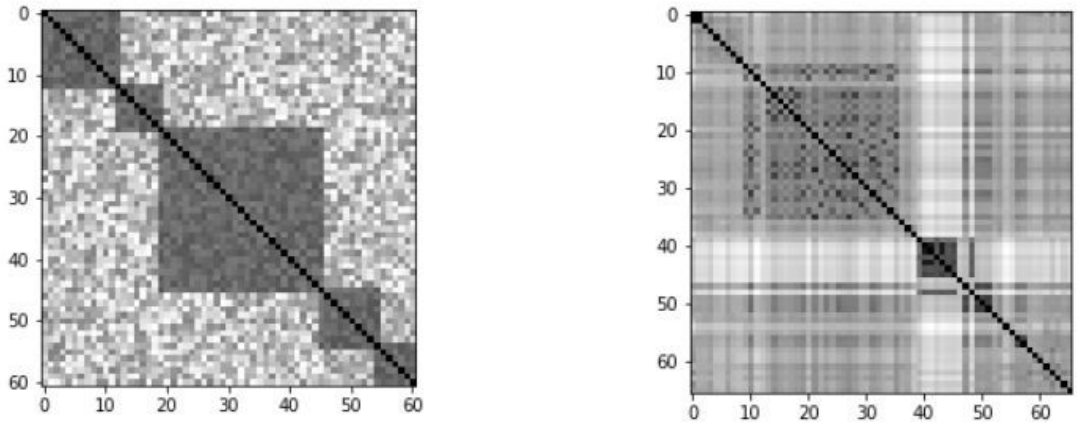
- *Top-down*: this approach breaks the large problem into multiple subproblems and the solutions are stored along the way: whenever a new subproblem needs to be solved, first the table is checked to see if it is already solved. If a solution is already recorded, it can be used directly, otherwise the subproblem is solved and the solution added to the table (this procedure is called *memoization*).

- *Bottom-up approach*: this approach first starts by computing results for the "smaller" subproblems; it then solves a bigger subproblems using the already computed smaller subproblems results, finally solving the whole problem. This process is referred to as *tabulation* since it requires filling a table from the start.

3.4.2 Notation

We report the original notation presented in [48]. Given N shots, and N related feature vectors x_i, $i = 1, ..., N$, with each $x_i \in \mathbb{R}^m$ (i.e., each of them consisting of m features), the list of the N feature vectors X_1^N is defined as $(x_1, \ldots, x_N)$. A metric $\mathfrak{D} : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ is defined to measure the distance between the feature vectors. $\mathfrak{D}$ is chosen to return a low value for the similar feature vectors and has to satisfy the mathematical conditions of a distance metric (non-negativity, identity of indiscernibles, symmetry, subadditivity). The matrix D is created by entering

the values of $\mathfrak{D}$ in a two-dimensional array: $\mathfrak{D}(x_i, x_j)$ provides the values of i^{th} row and j^{th} column of the matrix D. Due to the symmetry property of the distance metric, the matrix D is symmetrical, with all the elements on the main diagonal equal to zero. The ideal matrix D should have a block-diagonal structure. Larger values of distance are portrayed by brighter pixels, smaller values by darker ones. As represented in the Figures 3.3a and 3.3b, groups of features taken from adjacent shots return low distance values between them, resulting in darker (lower valued) blocks on the diagonal.



(a) Synthetic data

(b) Real data

Figure 3.3: D matrices: synthetic data (a) and real data (b). Brighter pixels represent larger distance values.

For a video scene detection algorithm to be effective, it is necessary to select a metric and to pick out features that return a small distance between feature vectors of the same scene, and large distances for features from different scenes. The purpose of this algorithm is to find a vector of indices $\tilde{t} = (t_0, \ldots, t_K)$, $t_0 := 0$, that denotes the partitioning into K scenes, so that $x_{t_i} \in X_1^N$ for $i = 1, ..., N$, and $t_i < t_j$ for any i, j such that $i < j$. In this way, it is ensured that scenes are sequential, and that every scene has at least one feature. For each $i = 1, .., K$, t_i denotes the last feature vector

index of the i^{th} scene. Therefore, $t_K = N$ always holds.

3.4.3 Cost function

We denote $\Omega^K \subset \mathbb{N}^K$ as the set of n-tuples that satisfy the requirements described at the end of the last paragraph. In general, a cost function $\mathcal{H} : \Omega^K \rightarrow \mathbb{R}$ maps a possible $\tilde{t}$ to a value indicating the quality of the scenes division. It is important to choose a cost function that has a neutral behaviour, with no inclination towards a specific partitioning, and that the global minimum of such functions reflects the ideally optimal scene partitioning. There are several possibilities for defining the cost function. Rotman et al. have proposed three of them [46], [47], [48].

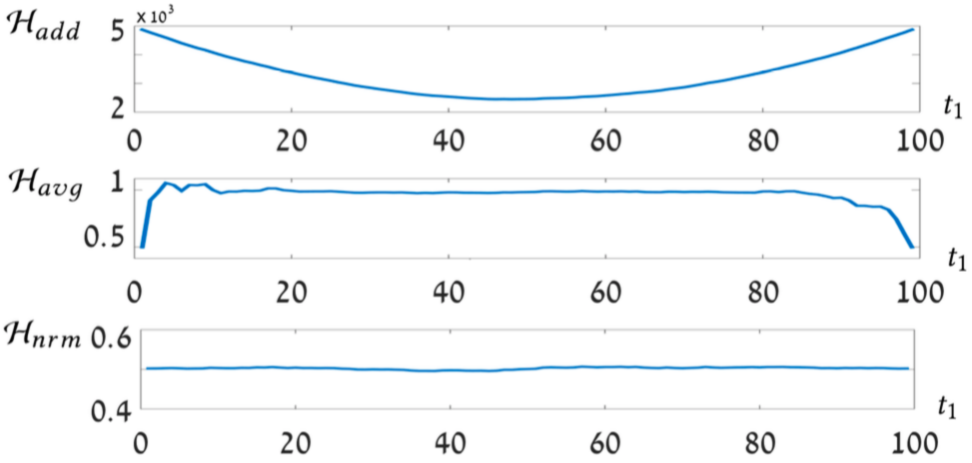


Figure 3.4: [48] Comparison between the different cost functions. The graph shows the behaviour of the 3 functions with respect to the division point t_1. In the example the D matrix is uniformly distributed with $N = 100$ and $K = 2$.

Addictive cost function

The additive cost function is defined as:

$$\mathcal{H}_{add}(\tilde{t}) = \sum_{i=1}^{K} \sum_{j_1=t_{i-1}+1}^{t_i} \sum_{j_2=t_{i-1}+1}^{t_i} D(x_{j_1}, x_{j_2}) \tag{3.5}$$

$\mathcal{H}_{add}$ sums all the values in the blocks on the diagonal. This cost function penalizes divisions containing big blocks. In other words, it has a strong inclination to favour equal-size partitions. As an example, in the case of a matrix with uniformly distributed elements to be partitioned into 2 scenes, a division towards the middle is highly foster. In fact, in this way the number of values being summed is the smallest ($\frac{N^2}{2}$ compared to roughly N^2 at the borders). There are 2 typical scenarios in which this cost function fails: when the blocks vary greatly in size and when the block structure is not very pronounced.

Average cost function

The average cost function is defined as:

$$\mathcal{H}_{avg}(\tilde{t}) = \sum_{i=1}^{K} \frac{\sum_{j_1=t_{i-1}+1}^{t_i} \sum_{j_2=t_{i-1}+1}^{t_i} D(x_{j_1}, x_{j_2})}{(t_i - t_{i-1})^2} \tag{3.6}$$

$\mathcal{H}_{avg}$ sums all the values in the blocks on the diagonal and divides them by the area of the blocks. Unlike the additive one, the average cost function prefers partitioning near the borders because a small block has a much higher probability of giving an extremely low average value compared to bigger blocks, since less values are accumulated. The numerator of $\mathcal{H}_{avg}$ increases faster than the denominator as blocks grow: in case of large blocks, the numerator of $\mathcal{H}_{avg}$ is more influential than its denominator. Instead, in case of small blocks, the situation is reversed since fewer values are added to the numerator. This behaviour leads to smaller blocks (even consisting of a single shot in extreme cases) partitioning.

Normalized cost function

The normalized cost function is defined as

$$\mathcal{H}_{nrm}(\tilde{t}) = \frac{\sum_{i=1}^{K} \sum_{j_1=t_{i-1}+1}^{t_i} \sum_{j_2=t_{i-1}+1}^{t_i} D(x_{j_1}, x_{j_2})}{\sum_{i=1}^{K} (t_i - t_{i-1})^2} \tag{3.7}$$

$\mathcal{H}_{nrm}$ sums all the values in the blocks on the diagonal and divides them by the sum of the areas of the blocks, excluding the values on the diagonal. In [48], an equivalent alternative definition of $\mathcal{H}_{nrm}$ is provided. In particular, its denominator is the result of the difference between the denominator of Equation 3.7 and N. In the intentions of the authors, this alternative definition allows to achieve a better normalization, since the values on the diagonal neither affect the numerator nor the denominator. As experimentally noted in [48], both definitions of normalized cost functions allow unbiased and correct partitioning, while overcoming the difficulties mentioned in additive and average cost functions. From now on, we consider the normalized cost function in Equation 3.7 for further analysis in this chapter.

The main issue in the usage of $\mathcal{H}_{nrm}$ is that the denominator is unknown at the beginning because the parameter K, that represents the number of scenes in which the video will be partitioned, is unknown a priori. The estimation of the parameter K is out of the scope of our work. We assume that the value of K is known because we conceived this work with a particular focus on the algorithm used for optimal grouping of scenes.

3.4.4 *Rotman18* solution approach

DP is used to globally minimize $\mathcal{H}_{nrm}$ and retrieving the optimal partitioning $\tilde{t}$. Since the solution to any given subproblem depends on the solution of the complete problem, the minimization of $\mathcal{H}_{nrm}$ is particularly difficult. The purpose of the DP method is to solve the subproblem $X_n^N = x_n, ..., x_N$ by dividing it in a subset of features $X_{i>n}^N$. However, finding the optimal t_i depends on $(t_1, ..., t_n, ..., t_{i-1})$, since

they affect the denominator. The denominator $\sum_1^K (t_i - t_{i-1})^2$ is the only external element that affects the sub-sequence optimizations. It can be treated as an unknown parameter $p \in \mathbb{N}$ and it can be called area since it is the area of the blocks on the diagonal. Although the evaluation over all possible values of the parameter might seem costly, in practice the number of possibilities can be greatly limited. The optimization process can be applied to every possible value of the p parameter by introducing a novel DP scheme. $\mathcal{H}^{n,k}$ is a superscript that indicates a subproblem where a sub-sequence of feature vectors $x_n, ..., x_N$ is divided into k segments. If not indicated, it is assumed to be $n = 1$ and $k = K$. The following three tables are defined:

- $C(n, k, p)$ is the table that express the optimal value of $\mathcal{H}^{n,k}_{nrm}$ when the left area corresponds to p, which is the area added to the denominator from dividing the rest of the sequence X_1^{n-1}.

- $I(n, k, p)$ is the table that contains the indices of the optimal partitioning of X_n^N in k scenes. It holds the optimal t_1 of the solution to $\mathcal{H}^{n,k}_{nrm}$ in order to enable recreation of the partitioning.

- $P(n, k, p)$ is the table that expresses the area used in the optimal solution contributed by X_n^N.

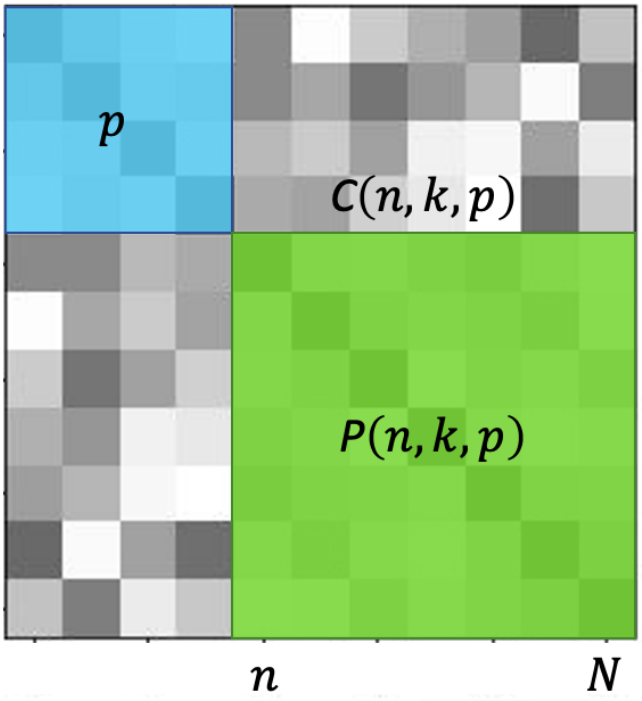


Figure 3.5: Graphical representation of C, I, P. As explained in this section, p is the area given by the sum of the areas of the blocks in which the shots $[1, n-1]$ can be partitioned. The same argument applies for P.

Initialization

The state of the DP algorithm is (n, k, p). The original intervals proposed in [48] for p are $[n, n^2]$ and $[\lceil \frac{n^2}{k} \rceil, (n-k+1)^2+k-1]$. However, in this section a new formulation for the bounds of p is presented. In fact, the initialization iterates over $n \in [1, N]$ and for $p \in [p_{min}(n-1, 1), p_{max}(n-1, 1)]$, where:

$$p_{min}(n,k) = \begin{cases} \frac{(n-1)^2}{K-k}, & \text{if } ((n-1) \bmod (K-k)) = 0 \\ ((n-1) \bmod (K-k)), & \text{otherwise} \end{cases} \tag{3.8}$$

and

$$p_{max}(n,k) = k - 1 + (n-k)^2 \tag{3.9}$$

are defined for $n = 1, ..., N$, $k = 1, ...K$, and $n \leqslant k$. We also set $p_{min}(n,k) = p_{max}(n,k) = 0$ for $n \leqslant 0 \vee k \leqslant 0 \vee n < k$. In the innermost iteration, the entries $C(n,1,p), I(n,1,p)$, and $P(n,1,p)$ are filled. These are the base cases for the DP algorithm. In fact, if $k = 1$, i.e., there is only one scene, all the shots in $[n,N]$ are grouped together in one single block, yielding an immediate cost function value computation. Such value is given by:

$$C(n,1,p) = \frac{\sum_{j_1=n}^{N}\sum_{j_2=n}^{N} D(x_{j_1}, x_{j_2})}{p + (N-n+1)^2} \tag{3.10}$$

The numerator is the sum of the elements of the single block formed by shots in $[n,N]$ and the denominator is given by the sum of the areas of the blocks. In particular, the denominator is the sum of p, which is the area of the blocks X_1^{n-1}, and $(N-n+1)^2$, which is the area of the single block that yields the optimal X_n^N division.
Since the shots in $[n,N]$ are grouped into a single scene, the ending shot of such scene is necessarily N. Therefore:

$$I(n,1,p) = N \tag{3.11}$$

Again, since there is only a single scene:

$$P(n,1,p) = (N-n+1)^2 \tag{3.12}$$

because there is only a single block with dimension $N-n+1$ in the optimal grouping.

Main procedure

The main procedure iterates over $k \in [2,K]$, $n \in [1,N]$, and $p \in [p_{min}(n-1, K-k), p_{max}(n-1, K-k)]$. Some combinations of k, n, p are excluded a priori, since they do not represent an actual subproblem. In particular, the condition $(k = K) \wedge (n > 1)$

must hold because it is not possible to divide $n-1>0$ shots in 0 scenes. Moreover, in a similar way, the condition $(n-1) \leqslant (K-k)$ holds, to prevent the division of less than $K-k$ shots in $K-k$ groups. Finally, the condition $N-(n+1)<k$ must hold because it is not possible to divide less than $N-(n+1)$ shots in k groups.
In the innermost iteration, the sequence of features X_n^N is divided in two parts by iterating on index i in range $[n, N-k+1]$. The first one is composed by the aggregation of the features X_n^i into a single group. The second one is given by the allocation of the remaining X_{i+1}^N features in $k-1$ groups. A value $\hat{i}$ must be chosen in the interval $[n, N-k+1]$ so that the cost function associated to the subproblem identified by n, k, and p is optimal. In particular, such cost C is given along I and P by the following equations:

$$C(n,k,p) = \min_{i \in [n, N-k+1]} \left\{ G_{n,k,p}(i) + C(i+1, k-1, p+(i-n+1)^2) \right\} \tag{3.13}$$

$$I(n,k,p) = \operatorname*{argmin}_{i \in [n, N-k+1]} \left\{ G_{n,k,p}(i) + C(i+1, k-1, p+(i-n+1)^2) \right\} \tag{3.14}$$

$$P(n,k,p) = (I(n,k,p)-n+1)^2 + P(I(n,k,p)+1, k-1, p+(I(n,k,p)-n+1)^2) \tag{3.15}$$

where:

$$G(n,k,p)(i) = \frac{\sum_{j_1=n}^{N} \sum_{j_2=n}^{N} D(x_{j_1}, x_{j_2})}{(p+(i-n+1)^2 + P(i+1, k-1, p+(i-n+1)^2)} \tag{3.16}$$

and $G_{n,k,p}(i) := G(n,k,p)(i)$. G is the contribution of the block X_n^i to the cost function. The denominator is composed by three elements that represent the area of the blocks on the diagonal found so far during the procedure: p is the area of solving for X_1^{n-1}, $(i-n+1)^2$ is the area for X_n^i and $P(i+1, k-1, p+(i-n+1)^2)$ is the area for X_{i+1}^N. $P(n,k,p)$ represents the area of the new block created by choosing $I(n,k,p)$ as the optimal point for division, and his value is composed by the contributions to the denominator (of the cost function) of X_n^i and X_{i+1}^N. At the end of the procedure, once the tables C, I and P are filled, the optimal cost function value clearly is $\tilde{\mathcal{H}}_{nrm}^{n,k} = C(1, K, 0)$.

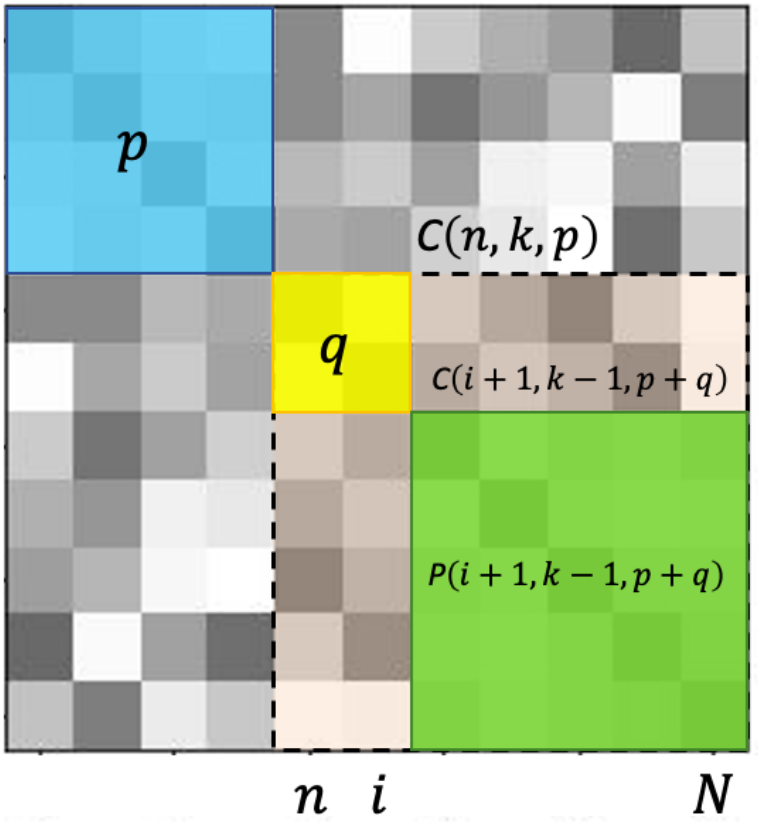


Figure 3.6: Another graphical representation of the solving procedure, analogous to Figure 3.5, which highlights how the main solving procedure actually works.

In Figure 3.6, a graphical representation of the main solving procedure is proposed. The entire block represents D. Suppose to solve the subproblem of grouping the shots in $[n, N]$ into k scenes, assuming that the sum of the areas of the blocks in which the shots in $[1, n-1]$ are partitioned is p. The algorithm iterates over each possible subdivision index i. Let q denote the area $(i-n+1)^2$ of the block (marked in yellow) formed by shots in $[n, i]$. Then, the value $C(n, k, p)$ is computed using the solution $C(i+1, k-1, p+q)$, stored after a previous computation. This solution is obtained by combining the solution to the trivial problem of grouping the shots from n to i

into a single scene, with the solution to the problem of subdividing shots from $i+1$ to N into $k-1$ blocks (clearly by considering the area outside of such subproblem as $p+q$).

Optimal solution reconstruction

Thanks to an iterative procedure, it is possible to reconstruct the optimal subdivision $\tilde{t}$. The initialization of the reconstruction is given by setting $\tilde{t}_0 = 0$ and $P_{tot} = 0$ which, at the beginning of each iteration i, represents the actual value of the parameter p. The following formulas are computed for $i = 1, ..., K$:

$$\tilde{t}_i \leftarrow I(\tilde{t}_{i-1} + 1, K - i + 1, P_{tot}) \tag{3.17}$$

$$P_{tot} \leftarrow P_{tot} + (\tilde{t}_i - \tilde{t}_{i-1})^2 \tag{3.18}$$

As an example, the first three computations are:

- $\tilde{t}_1 \leftarrow I(\tilde{t}_0 + 1, K, 0)$, and the area starting from $\tilde{t}_0 + 1$ must be divided in K scenes. At the beginning of this iteration, P_{tot} is equal to 0 because the entire block (from n to N) is considered for the division;
- $\tilde{t}_2 \leftarrow I(\tilde{t}_1+1, K-1, (\tilde{t}_1-\tilde{t}_0)^2)$, and the area starting from $\tilde{t}_1+1$ must be divided in $K-1$ scenes. At the beginning of this iteration, $P_{tot} = (\tilde{t}_1 - \tilde{t}_0)^2$, which is the remaining area on the left of $[\tilde{t}_1 + 1, N]$;
- $\tilde{t}_3 = I(\tilde{t}_2 + 1, K - 2, (\tilde{t}_2 - \tilde{t}_1)^2 + (\tilde{t}_1 - \tilde{t}_0)^2)$, and the area starting from $\tilde{t}_2 + 1$ must be divided in $K - 2$ scenes. At the beginning of this iteration, $P_{tot} = (\tilde{t}_2 - \tilde{t}_1)^2 + (\tilde{t}_1 - \tilde{t}_0)^2$, which is the new remaining area on the left until $\tilde{t}_2$;

and so on. The procedure ends when $\tilde{t}_k$ is computed.
As a sanity check for the procedure, the final value of P_{tot}, which is the sum of all the blocks provided by the optimal division $\tilde{t}$, should be equal to $P(1, K, 0)$. In fact,

this value represents the sum of the areas of the blocks on the diagonal which form the solution.

$$P_{tot} = P(1, K, 0) \tag{3.19}$$

Binary lookup table for pruning impossible subproblems

The additional limitation of p proposed in this thesis does not exclude all the impossible values of p. A further improvement of the optimization process might be possible by precomputing which exact values of p are possible, and which are not, as done by Rotman et al. [48]. A static binary lookup table $B(n, k, p)$ is computed completely offline and it is independent from the choice of distance metric or features. $B(n, k, p)$ is $true$ when p is a possible area of the division of the n features in k groups, and $false$ otherwise. B is initialized iterating for $n \in [1, N]$ and for $p \in [0, N^2]$, enforcing the following condition on B:

$$B(n, 1, p) = true \quad \iff \quad n^2 = p \tag{3.20}$$

The rest of the table is filled up iterating over $k \in [2, K]$, $n \in [1, N]$, and $p \in [p_{min}(n, k), p_{max}(n, k)]$. The conditions $(k = K) \wedge (n > 1)$ and $k > n$ must hold in order for the triplet n, k, p to identify a subproblem of the whole problem. By considering the ranges defined above, the rest of the table is filled up using the Formula 3.21 given as:

$$B(n, k, p) = \bigvee_{i=1}^{\lceil\sqrt{p}-1\rceil} B(n - i, k - 1, p - i^2) \tag{3.21}$$

3.5 Exact methods for scene segmentation

It is necessary to introduce an alternative notation in order to properly describe the solution approach to the VSD problem in [48], described in detail in the previous paragraph.

3.5.1 Notation

For the scene segmentation boundaries, we adopt the same notation used by Rotman et al. [48], i.e., we denote the distance metric between shots as $\mathcal{D} : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$. In particular, for shots i, j and their respective feature vectors x_i, x_j, we define $\mathcal{D}_{i,j} := \mathcal{D}(x_i, x_j)$. We also denote the ending shot of the k^{th} scene as t_k. According to this choice, the complete scene segmentation of N shots in K scenes can be expressed as $\mathbf{t}$, with $\mathbf{t} = (t_0, t_1, ..., t_K)$, where $t_0 := 0$. However, we also introduce a reformulation of the cost function $\mathcal{H}_{nrm}$, specializing it for any segment of $\mathbf{t}$.

In particular, given shots i, j, $0 \leqslant i \leqslant j \leqslant N-1$, the ending shots of the scenes $\hat{K}$ into which the shots in $[i, j]$ are grouped, are expressed as an ordered list $\mathbf{t}^{i,j} := (t_i, t_{i+1}, ..., t_j)$. Therefore, we define a cost metric for the scene segmentation of shots in $[i, j]$ which is analogous to the metric defined for the whole segmentation:

$$\mathcal{H}_{nrm,\hat{K}}(\mathbf{t}^{i,j}) := \frac{\Phi_{\hat{K}}(\mathbf{t}^{i,j})}{\Gamma_{\hat{K}}(\mathbf{t}^{i,j})} \tag{3.22}$$

Where:

$$\Phi_{\hat{K}}(\mathbf{t}^{i,j}) = \sum_{n=i}^{t_i}\sum_{p=i}^{t_i}\mathcal{D}_{n,p} + \sum_{m=i}^{j-1}\sum_{n=t_m+1}^{t_{m+1}}\sum_{p=t_m+1}^{t_{m+1}}\mathcal{D}_{n,p} \tag{3.23}$$

$$\Gamma_{\hat{K}}(\mathbf{t}^{i,j}) = \sum_{m=i+1}^{j}(t_m - t_{m-1})^2 \tag{3.24}$$

We consider this notation more appropriate for evaluating the subproblems which naturally arise in the DP approach of [48] to the problem proposed by the authors.

3.5.2 An alternative dynamic programming method

For the *Recursive Solver* (*RS*) formalization, let us consider the problem of grouping N consecutive shots into K scenes, and identify the subproblem of dividing the shots from i to j into k scenes with the triplet (i, j, k). We assume that the shots are

numbered from 0 to $N-1$.

The cost function in equation 3.22 allows us to properly evaluate a subproblem (i,j,k). Note that this triplet, in order to identify a subproblem, must satisfy the condition:

$$k^l_{i,j} \leqslant k \leqslant k^u_{i,j} \tag{3.25}$$

Where $k^l_{i,j} = max\{K-i-(N-1-j),1\}$, and $k^u_{i,j} = min\{K-min\{i,1\}-min\{N-1-j,1\},j-i+1\}$. In fact, k is lower bounded by the minimum number of scenes needed to qualify (i,j,k) as a subproblem of $(0,N-1,K)$, i.e., the number of scenes k^e in which the shots outside $[i,j]$ can be grouped into, added to k, must be equal to K. Since the maximum value $k^{e,max}$ of k^e is $K-i-(N-1-j)$, and $k \geqslant K-k^{e,max}$, the expression for $k^l_{i,j}$ naturally follows. Similarly, $k^u_{i,j}$ limits k to be consistent with the number of scenes $k^{e,min} = min\{i,1\}-min\{N-1-j,1\}$. Since k cannot exceed the number of shots $j-i+1$, Equation 3.25 follows.

First, for the sake of compactness, given $\mathbf{t}^{i,j} := (t_i, t_{i+1}, ..., t_j)$, and k scenes into which we group shots from $[t_i,t_j]$, we define $\Phi_{i,j,k} := \Phi_k(\mathbf{t}^{i,j})$, $\Gamma_{i,j,k} := \Gamma_k(\mathbf{t}^{i,j})$, and $\mathcal{H}_{i,j,k} := \mathcal{H}_{nrm,k}(\mathbf{t}^{i,j})$.

The solution of the subproblem (i,j,k) can be expressed as the combination of the solution of smaller subproblems, in particular:

$$\mathcal{H}_{i,j,k} = \min_{t=i,...,j-1} \min_{\hat{k}=k^l_{i,j},...,k^u_{i,j}} \frac{\Phi_{i,t,\hat{k}} + \Phi_{t+1,j,k-\hat{k}}}{\Gamma_{i,t,\hat{k}} + \Gamma_{t+1,j,k-\hat{k}}} \tag{3.26}$$

having initialized $\mathcal{H}_{i,j,k} = 0$. After the computation of $\mathcal{H}_{i,j,k}$, we can easily express the value of $\Phi_{i,j,k}$ and $\Gamma_{i,j,k}$ that allow $\mathcal{H}_{i,j,k}$ to attain the optimum.

First, we express the optimal value of t and $\hat{k}$ of 3.26 as:

$$(t^{min}, \hat{k}^{min}) = \underset{t=i,...,j-1,\hat{k}=k^l_{i,j},...,k^u_{i,j}}{\operatorname{argmin}} \frac{\Phi_{i,t,\hat{k}} + \Phi_{t+1,j,k-\hat{k}}}{\Gamma_{i,t,\hat{k}} + \Gamma_{t+1,j,k-\hat{k}}} \tag{3.27}$$

The following two equations immediately follow:

$$\Phi_{i,j,k} = \Phi_{i,t^{min},\hat{k}^{min}} + \Phi_{t^{min}+1,j,k-\hat{k}^{min}} \tag{3.28}$$

$$\Gamma_{i,j,k} = \Gamma_{i,t^{min},\hat{k}^{min}} + \Gamma_{t^{min}+1,j,k-\hat{k}^{min}} \tag{3.29}$$

The DP procedure used to obtain $\mathcal{H}_{0,N-1,K}$ through the exploitation of the recursive relations 3.26 - 3.29 is shown in Algorithm 2. We define a table I where $I_{i,j,k}$ is the optimal division $\mathbf{t} := (t_0, t_1, ..., t_k)$, given by dividing the shots from i to j in k scenes. Moreover, in Algorithm 1, we define a table S where each element $S_{i,j}$ contains the sum of the elements of the diagonal block from i to j, computed by exploiting the DP procedures described in [46].

Algorithm 1: Compute Matrix Blocks

input : a distance matrix $\mathcal{D} \in \mathbb{R}_+^{N \times N}$

output: $S \in \mathbb{R}_+^{N \times N}$ such that $S_{i,j} = \sum_{t=i}^{j} \sum_{t'=i}^{j} \mathcal{D}_{t,t'}$

```
1 begin
2     S ← a N × N zero matrix
3     for i ← N − 1 downto 1 do
4         l ← N − i + 1
5         for j ← 0 to i − 1 do
6             S_{j,j+l−1} = S_{j,j+l−2} + S_{j+1,j+1+l−2} + 2 · D_{j+l−1,j} − S_{j+1,j+l−2}
```

As regards the computational complexity of Algorithm 1, lines $4-6$ are executed $N-1$ times. At the i^{th} iteration, lines $4-5$ are executed i times. Therefore, the computational complexity of Algorithm 1 is $\theta(\sum_{i=1}^{N-1} i) = \theta(N^2)$.

Line 6 is the core of the algorithm, which is graphically summarized in Figure 3.7. Starting from the block of dimension 1, it is possible to compute the sum of the elements of all the blocks on the diagonal until a maximum dimension of N. In particular, the sum of the elements of any block B of dimension 2 is carried out by summing the values of the two blocks B' and B'' of dimension 1 on the diagonal belonging to B, and then subtracting the values of the blocks generated by the

intersection of B' and B'', that in this case results to be empty, and finally adding the values of the two remaining elements of B. Since D is symmetric, these two elements are equivalent. For a generic l block B_l, with $l = 3, ..., N$ starting from shot j and ending at shot $j + l - 1$, the sum of the elements $S_{j,j+l-1}$ is obtained by adding the sum of the elements of the two block $S_{j,j+l-2}$ and $S_{j+1,j+l-1}$, each of dimension $l - 1$, then subtracting the elements of the block $S_{j+1,j+l-2}$ of dimension $l - 2$ and adding the values of the two remaining elements $D_{j+l-1,j}$ and $D_{j,j+l-1}$ belonging to B_l.

As an example, when $l = 4$, $S_{j,j+l-2}$ and $S_{j+1,j+l-1}$ are marked in orange, $S_{j+1,j+l-2}$ is marked in yellow, and the two elements $D_{j+l-1,j}$ and $D_{j,j+l-1}$ are marked in green.

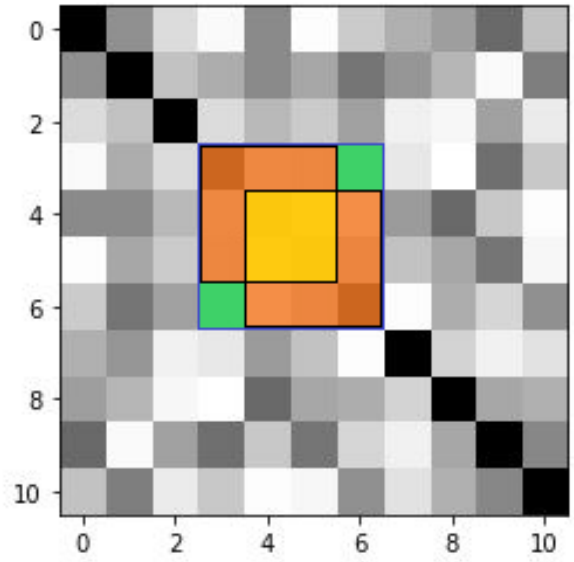


Figure 3.7: Table S initialization example for a block of dimension 4

The algorithm that employs the recursive relations 3.26 - 3.29 is shown in Algorithm 2. For the sake of compactness, we denote the non-decreasingly ordered list of the ending shots of the k scenes into which the shots in $[i, j]$ are grouped as $\mathbf{t}_k^{i,j}$.

Algorithm 2: Recursive Solver

input : shot indices i, j; k scenes; $S \in \mathbb{R}_+^{N\times N}$ filled with Algorithm 1; $\Phi \in \mathbb{R}^{N\times N\times K}$; $\Gamma \in \mathbb{R}^{N\times N\times K}$; $\mathbf{t}_k^{i,j} \in {\mathbb{Z}_+}^{k\times N\times N\times K}$

output: $\Phi_{i,j,k}, \Gamma_{i,j,k}, \mathbf{t}_k^{i,j}$

1 **begin**
2 **if** $\Phi_{i,j,k} >= 0$ **then**
3 **return** $\Phi_{i,j,k}, \Gamma_{i,j,k}, \mathbf{t}_k^{i,j}$
4 **if** $k == 1$ **then**
5 $\Gamma_{i,j,k} \leftarrow (j-i+1)^2$
6 $\Phi_{i,j,k} \leftarrow S_{i,j}$
7 $\mathbf{t}_k^{i,j} \leftarrow [j]$
8 **else**
9 $\mathcal{H}_{i,j,k} =\leftarrow \infty$
10 **for** $t \leftarrow i$ *to* $j-1$ **do**
11 **for** $\hat{k} \leftarrow \max(t+k-j, 1)$ *to* $\min(k-1, t-i+1)$ **do**
12 $\Gamma_{i,t,\hat{k}}, \Phi_{i,t,\hat{k}}, \mathbf{t}_{\hat{k}}^{i,t} \leftarrow RecursiveSolver(i,t,\hat{k})$
13 $\Gamma_{t+1,j,k-\hat{k}}, \Phi_{t+1,j,k-\hat{k}}, \mathbf{t}_{k-\hat{k}}^{t+1,j} \leftarrow RecursiveSolver(t+1,j,k-\hat{k})$
14 $\mathcal{H} \leftarrow \frac{\Phi_{i,t,\hat{k}}+\Phi_{t+1,j,k-\hat{k}}}{\Gamma_{i,t,\hat{k}}+\Gamma_{t+1,j,k-\hat{k}}}$
15 **if** $\mathcal{H} < \mathcal{H}_{i,j,k}$ **then**
16 $\mathcal{H}_{i,j,k} \leftarrow \mathcal{H}$
17 $\Phi_{i,j,k} \leftarrow \Phi_{i,t,\hat{k}} + \Phi_{t+1,j,k-\hat{k}}$
18 $\Gamma_{i,j,k} = \Gamma_{i,t,\hat{k}} + \Gamma_{t+1,j,k-\hat{k}}$
19 $\mathbf{t}_l \leftarrow \mathbf{t}_{\hat{k}}^{i,t}$
20 $\mathbf{t}_r \leftarrow \mathbf{t}_{k-\hat{k}}^{t+1,j}$
21 $\mathbf{t}_k^{i,j} \leftarrow$ concatenation of $\mathbf{t}_l$ with $\mathbf{t}_r$
22 **return** $\Phi_{i,j,k}, \Gamma_{i,j,k}, \mathbf{t}_k^{i,j}$

Lines $2-3$ verify if the subproblem of dividing shots from i to j in k scenes is already solved because, thanks to its recursive structure marked in line $12-13$, it is possible that for a given i, j, k, a subproblem could be already solved. If this is the case, it

is returned the numerator of the cost function $\Phi_{i,j,k}$, the denominator of the cost function $\Gamma_{i,j,k}$ and the optimal division list of shot index $\mathbf{t}_k^{i,j}$.

Lines $4-7$ cover the case of dividing the shots from i to j in one scene. Therefore, the numerator of the cost function is given by the sum of the values of the block $S_{i,j}$, the denominator of the cost function results equal to $(j-i+1)^2$ which is the area of the block and the optimal division list of indexes is equal to a list of a single element which contains the last shot index j.

Lines $9-21$ face the case of dividing the shots from i to j in more than one scene when the subproblem is not already solved. In particular, for each allowed value of t and $\hat{k}$, in lines $12-13$ it is reported a recursive call to this algorithm in order to find the solution of two smaller subproblems. The first consists on dividing the shots from i to t in $\hat{k}$ scenes whereas the second consists on dividing shots from $t+1$ to j in $k-\hat{k}$ scenes. At the end of the procedure there are returned the values of $\Phi_{i,j,k}, \Gamma_{i,j,k}, \mathbf{t}_k^{i,j}$ that provides the optimal value of $\mathcal{H}_{i,j,k}$ according to the Formula 3.26.

3.5.3 Method improvement through Branch and Bound

Starting from the notation defined in (3.5.1) and the algorithm defined in (3.5.2), we propose a new method, called *Recursive Solver with Bounds* (RS_B), that exploits bounding to improve the performance. This methodology is obtained by exploiting pruning that is a technique used to reduce the size of the problem by removing sections that provide no possibilities to improve the optimal solution. This method is created from the one proposed in (3.5.2) by adding the initialization proposed in Algorithm 5 and the bounding conditions proposed in Algorithm 6 and 7. In order to obtain such result, five new tables are created to store the computation values. The tables $\Phi^u \in \mathbb{R}^{N\times N\times K}$ and $\Phi^l \in \mathbb{R}^{N\times N\times K}$ are defined. Such tables are respectively the tables that contain the upper bound numerator $\Phi^u_{i,j,k}$ and the lower bound numerator

$\Phi^l_{i,j,k}$ of the optimal cost function $\mathcal{H}_{i,j,k}$. Similarly, the tables $\Gamma^u \in \mathbb{R}^{N\times N\times K}$ and $\Gamma^l \in \mathbb{R}^{N\times N\times K}$ are defined. They consist of the upper bound $\Gamma^u_{i,j,k}$ denominator and the lower bound denominator $\Gamma^l_{i,j,k}$ of the optimal cost function $\mathcal{H}_{i,j,k}$, respectively. It is important to note that $\Phi^u_{i,j,k}$ does not necessarily represent the minimum value of numerator, it generates the lowest cost function value when combined with the denominator $\Gamma^u_{i,j,k}$. This argument applies in the same way for the lower bound.
Then, it is defined $Sml[i][j]$ as a list of each possible dimension of the blocks on the main diagonal of the matrix D. Sml is initialized after $S_{i,j}$ and contains the matrices from the first shot in ascending order. The algorithm requires some auxiliary functions to perform the pruning. In order to fill up the tables Φ^u and Γ^u , the Algorithm 3 is defined. The lower bound of the numerator is identified by uniforming the dimension of the blocks on the diagonal that compose the solution of the subproblem of dividing shots from i to j in k scenes. As an example, if $j - i = 5$ and $k = 3$, the uniforming action is performed by creating two block of side 2 and one block of side 1. Moreover, the minimum value of the sum of the elements of these blocks is identified.

Algorithm 3: Compute Lower Bound

input : shot indices i, j; k scenes;

output: $\Phi^l_{i,j,k}, \Gamma^l_{i,j,k}$

1 **begin**

2 **if** $j - i + 1 = k$ **then**

3 **return** $0, k$

4 $d^u = \lceil \frac{j-i+1}{k} \rceil$

5 $d^l = \lfloor \frac{j-i+1}{k} \rfloor$

6 **if** $(j - i + 1) \bmod k == 0$ **then**

7 $n_{d^u} = k$

8 **return** $n_{d^u} \cdot \min(Sml[d^u][i : (j + 1 - d^u + 1)], (((j - i + 1 - (k - 1))^2) + (k - 1))$

9 **else**

10 $n_{d^u} = (j - i + 1) \bmod k$

11 $n_{d^l} = k - (j - i + 1) \bmod k$

12 **return** $n_{d^u} \cdot \min(Sml[d^u][i : (j + 1 - d^u + 1)] +$
$n_{d^l} \cdot \min(Sml[d^l][i : (j + 1 - d^l + 1)]), (((j - i + 1 - (k - 1))^2) + (k - 1))$

Lines 2-3 face the case of having a number of shots equal to the number of scenes k, therefore the value of $\Phi^l_{i,j,k}$ is 0.

Lines 4-5 compute the maximum and the minimum dimension of the blocks that can compose the solution as d^u and d^l by trying to compute blocks of uniform dimension.

Lines 6-8 face the case of having a number of shots multiple of the number of scenes. In this case $\Phi^l_{i,j,k}$ is computed as the number of blocks of side d^u, which in this case is equal to k, multiplied by the minimum value of the sum of the elements of all possible blocks of side d^u that starts in the interval $(i, j + 1 - d^u + 1)$, obtained by applying Sml. Otherwise, the number of blocks of side d^u is equal to the rest of the division between the number of shots and k whereas the number of blocks of side d^l is given by k minus the rest of the division between the number of shots and k.

In lines 12, $\Phi^l_{i,j,k}$ is computed as the quantity defined in line 8 plus the number of blocks of side d^l, multiplied by the minimum value of the sum of the elements of all

possible blocks of side d^l that starts in the interval $(i, j + 1 - d^u + 1)$. This latter quantity can be 0 if there is no left side of the block. Moreover, the value of the denominator $\Gamma^l_{i,j,k}$ is returned.

The upper bound tables Φ^u and Γ^u are filled up thanks to the Algorithm 4. This algorithm is similar to the one described in 2 but instead of trying all the possible values of $\hat{k}$ and t, it relies on a precise value of $\hat{k}$ and t. This is shown in lines 8-9.

Algorithm 4: Compute Upper Bound

input : shot indices i, j; k scenes; $S \in \mathbb{R}_+^{N \times N}$ filled with Algorithm 1; $\Phi^u_{i,j,k}, \Gamma^u_{i,j,k} \in {\mathbb{R}_+}^{N \times N \times K}$

output: Φ^u, Γ^u

1 **begin**
2 **if** $\Phi^u_{i,j,k} < \infty$ **then**
3 **return** $\Phi^u_{i,j,k}$, $\Gamma^u_{i,j,k}$
4 **if** $k == 1$ **then**
5 $\Gamma^u_{i,j,k} \leftarrow (j - i + 1)^2$
6 $\Phi^u_{i,j,k} \leftarrow S_{i,j}$
7 **else**
8 $t = \lfloor \frac{i+j}{2} \rfloor$
9 $\hat{k} = \lfloor \frac{max(t+k-j,1)+min(k-1,t-i+1)}{2} \rfloor$
10 $\Phi^u_{i,t,\hat{k}}, \Gamma^u_{i,t,\hat{k}}, \leftarrow ComputeUpperBound(i, t, \hat{k}, S, \Phi^u, \Gamma^u)$
11 $\Phi^u_{t+1,j,k-\hat{k}}, \Gamma^u_{t+1,j,k-\hat{k}} \leftarrow ComputeUpperBound(t + 1, j, k - \hat{k}, S, \Phi^u, \Gamma^u)$
12 $\Phi^u_{i,j,k} \leftarrow \Phi^u_{i,t,\hat{k}} + \Phi^u_{t+1,j,k-\hat{k}}$
13 $\Gamma^u_{i,j,k} \leftarrow \Gamma^u_{i,t,\hat{k}} + \Gamma^u_{t+1,j,k-\hat{k}}$
14 **return** $\Phi^u_{i,j,k}$, $\Gamma^u_{i,j,k}$

Therefore, by exploiting Algorithms 3 and 4, the initialization of the bounding process is given as follows:

Algorithm 5: Bound Initialization

input : number of shots N, number of scenes K

Sml

$S \in \mathbb{R}_{+}^{N \times N}$ already filled with Algorithm 1;

output: Φ^l,Γ^l,Φ^u,Γ^u

```
1 begin
2     Φ^u, Γ^u, Φ^l, Γ^l ∈ R^{N×N×K}
3     for i ← 0 to N − 1 do
4         for j ← i to N − 1 do
5             for k ← k^l_{i,j} to k^u_{i,j} do
6                 Φ^l_{i,j,k}, Γ^l_{i,j,k} ← ComputeLowerBound(i, j, k, S, Φ^l, Γ^l)
7     ComputeUpperBound(i, j, k, S, Φ^u, Γ^u)
```

After the initialization of the matrices Φ^l, Γ^l, Φ^u, Γ^u, the RS algorithm needs to be modified by adding two bounding conditions (Algorithms 6 and 7) in order to be more efficient.

Algorithm 6: First Bound Condition

input : shot indices i, j; k scenes; Φ^u, Γ^u, Φ^l, $\Gamma^l \in \mathbb{R}^{N \times N \times K}$; $\mathcal{H}^u_{0,N-1,K}$

output: true if the current subproblem can be pruned, **false** otherwise

```
1  begin
2      î ← max(i − 1, 0); ĵ ← min(j + 1, N − 1)
3      for k̂ ← max(1, K − k − N + 1 + j) to min(K − k, i) do
4          Φ̂^u_{0,N−1,K} ← Φ^u_{0,î,k̂} + Φ^u_{i,j,k} + Φ^u_{ĵ,N−1,K−k−k̂}
5          Γ̂^u_{0,N−1,K} ← Γ^u_{0,î,k̂} H(i) + Γ^u_{i,j,k} + Γ^u_{ĵ,N−1,K−k−k̂} H(N − 1 − j)
6          Φ̂^l_{0,N−1,K} ← Φ^l_{0,î,k̂} + Φ^l_{i,j,k} + Φ^l_{ĵ,N−1,K−k−k̂}
7          Γ̂^l_{0,N−1,K} ← Γ^l_{0,î,k̂} H(i) + Γ^l_{i,j,k} + Γ^l_{ĵ,N−1,K−k−k̂} H(N − 1 − j)
8          if Φ̂^u_{0,N−1,K} / Γ̂^u_{0,N−1,K} < H^u_{0,N−1,K} ∨ Φ̂^l_{0,N−1,K} / Γ̂^l_{0,N−1,K} < H^u_{0,N−1,K} then
9              return false
10     return true
```

where H is the Heaviside function:

$$H(i) = \begin{cases} 1, & \text{if } i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.30}$$

In line 4, it is computed the numerator upper bound $\hat{\Phi}^u_{0,N-1,K}$ of the cost function of the problem of dividing shots from 0 to $N-1$ in K scene. This done by summing the upper bounds of three subproblems. The upper bound numerator of the "left" subproblem $\Phi^u_{0,\hat{i},\hat{k}}$ is given by dividing shots from 0 to $\hat{i}$ in $\hat{k}$ scenes. The upper bound numerator of the "middle" subproblem $\Phi^u_{i,j,k}$ is given by dividing shots from i to j in k scenes and the upper bound numerator of the "right" subproblem $\Phi^u_{\hat{j},N-1,K-k-\hat{k}}$ is given by dividing shots from $j+1$ to $N-1$ in $K-k-\hat{k}$ scenes.

Similarly, in lines 5-7, $\hat{\Gamma}^u_{0,N-1,K}$, $\hat{\Phi}^l_{0,N-1,K}$, $\hat{\Gamma}^l_{0,N-1,K}$ are computed. For the computation of $\hat{\Gamma}^u_{0,N-1,K}$ and $\hat{\Gamma}^l_{0,N-1,K}$, an Heaviside function $H(i)$ is employed that drops to zero the value of the left and right subproblems when the middle subproblems incorporate them.

Line 8 of Algorithm 6 checks if it necessary to prune this solution by comparing the values of upper and lower bound against the current optimal solution $\mathcal{H}^u_{0,N-1,K}$.

The following condition can be added after line 11 of Algorithm 2 in order to enhance its pruning capabilities:

Algorithm 7: Second Bound Condition

```
1 if (Φ^l_{i,t,k̂} + Φ^l_{t+1,j,k−k̂}) / (Γ^l_{i,t,k̂} + Γ^l_{t+1,j,k−k̂}) > H_{i,j,k} then
2     pruning
```

In this way, all the solutions that give a lower bound grater than optimal solution are discarded.

Chapter 4

Experimental results

Summary

All the results obtained by the application of the methodologies proposed in (3) are reported in this chapter.

The main test phase consists in a sequential run of our entire VSD pipeline composed by shot detection, frame selection, feature extraction and dynamic programming algorithms. In addition, specific tests has been previously done to evaluate and select the shot detection and frame selection methods.

All the algorithms are coded in Python 3.7, with the only exception of the Baraldi et al. shot detection implementation (3.1.1), provided in C++. The complete test environment is presented in Appendix A.

All the test are performed on the same Windows 10 virtual machine in order to obtain comparable results. These are shown in a tabular format.

The computational time CPU is measured in seconds, with a precision of 16 milliseconds. The N and K are respectively the number of shots and scenes of the video which refers to, and can be used to understand the dimensionality of the problem. $\mathcal{H}_{nrm}$ is the cost function value that has to be minimized by the algorithm.

Starting from the standard $F1 - measure$ (4.1.1), used in shot detection algorithm tests, we evaluated and implemented several measures in order to find the one suitable to describe the accuracy of the scene division task. We analyzed DED (4.1.5), $F-score$ (4.1.2), $F-score^*$ (4.1.3) and we implemented the $F-score^*_{shot}$ (4.1.4). The choice fell on DED as scene division metric because is a symmetric uni-dimensional measure defined in range $[0, 1]$ and it does not show the negative values problem of $F - score$ and the clipping issue of the $F - score^*$ and $F - score^*_{shot}$.

This chapter is structured as follows. In Section 4.1, the evaluation measures related to scene and shot detection are presented and analyzed.

In Section 4.2, the results related to the shot detection methodologies proposed in Sections 3.1.1 and 3.1.2 are reported.

In Section 4.4 are displayed the results related to the application of the key frames selection methods proposed in Sections 3.2.1-3.2.5.

Finally, in Section 4.5 are reported the results related to the application of the dynamic programming methodologies proposed in Sections 3.4.4, 3.5.2 and 3.5.3, which are the most relevant results of this work.

4.1 Evaluation metrics

As reported in [19], several evaluation measures are proposed in literature to quantify the scene segmentation performances. We selected and implemented the most useful and common in VSD research field in order to test and evaluate the shot and scene detection algorithms. Each one has flaws and advantages as it can be seen in sections below. As we explain in the Summary of this chapter, DED is the chosen metric for our tests.

4.1.1 Recall, Precision, F1-measure

In most of the early-days works about VSD, *recall* and *precision* are the chosen metrics for the performance evaluation of the scene segmentation algorithms. These measures are not specifically designed for managing video scene detection problem: *recall* and *precision* are a standard in information retrieval field and express the accuracy of retrieved results.

Given the set of shots of automatically retrieved scenes A and the set of shots of ground truth scenes R (called relevant scenes), *recall* and *precision* are computed as follows:

$$recall = \frac{|R \cap A|}{|R|} \tag{4.1}$$

$$precision = \frac{|R \cap A|}{|A|} \tag{4.2}$$

This formulation can be used to evaluate shot detection performances as well, replacing A with the set of frames of the automatically generated shots and R with the frames of the ground truth shots. As it is possible to see from the formulation, these measures are affected by each other, so the $F1 - measure$ is introduced as the harmonic mean of *recall* and *precision*:

$$F1 - measure = \frac{2 \cdot (precision \cdot recall)}{(precision + recall)} \tag{4.3}$$

Since *recall*, *precision* and $F1 - measure$ rely on how many relevant elements are found and how many elements are found altogether, the application of such measures to the evaluation of video scene detection task is not immediate. The main drawback of such evaluation measures is that they can not distinguish the magnitude of a misdetection. If a scene boundary is detected one shot before or after its ground truth position, an error is counted in *recall* as if the boundary was not detected at all, and in *precision* as if the boundary was put far away. In literature, there is not a unified method to manage this issue [7].

4.1.2 Coverage, Overflow, F-score

Vendrig and Worring [61] introduced two measures called *Coverage* and *Overflow* that are used by Rotman et al. [48] as a reference for measuring the accuracy of the division produced by their scene detection algorithm.

Coverage measures the quantity of shots, belonging to the same scene, that are correctly grouped together. *Overflow* evaluates to what extent shots, not belonging to the same scene, are erroneously grouped together. For a video, the *Coverage* and *Overflow* are the average *coverage* and *overflow* ratios of its ground truth scenes.

The combination of *Coverage* and *Overflow* is called $F - score$ and although it was the metric chosen in Rotman et al. paper, in our work has not been used as the main index of performance due to his limitations.

These measures do not express whether scenes are correctly detected but how accurately they are detected. Given the set of automatically detected scenes $\breve{V}_\lambda = [\breve{\lambda}_1, \breve{\lambda}_2, ..., \breve{\lambda}_n]$ and the set of the ground truth scenes $V_\lambda = [\lambda_1, \lambda_2, ..., \lambda_n]$, where $\breve{\lambda}_j$ identifies a detected scene and λ_t identifies a ground truth scene, *Coverage* and *Overflow* are defined as follows.

Firstly, the *coverage* is measured for each scene λ_t and then the global *Coverage* C of the video is computed as a weighted sum of the $C(\lambda_t)$ computed for each scene. The *coverage* $C(\lambda_t)$ is computed as a fraction where the numerator is the longest overlap between the j^{th} generated scene $\breve{\lambda}_j$ and the ground truth scene λ_t, and the denominator is the length of the ground truth scene λ_t.

The # operator expresses the cardinality in shots of the variable which is applied to (that can be a scene or the video). The *coverage* $C(\lambda_t)$ is then multiplied by the ratio between the number of shots that compose the scene λ_t and the total number of shot of the video V_σ. By repeating such operation for each scene and consequently summing the results, the *Coverage* C for the entire video is obtained. As it can be seen in the Figure 4.1, the best case $C = 1$ is reached when $\breve{\lambda}_j = \lambda_t$ and the detected

scene is equal to the corresponding scene ground truth.

For each scene:

$$C(\lambda_t) = \frac{\max\limits_{j=0...n} \#(\breve{\lambda}_j \cap \lambda_t)}{\#(\lambda_t)} \tag{4.4}$$

For the entire video:

$$C = \sum_{t=0}^{\#(V_\lambda)-1} C(\lambda_t) \cdot \frac{\#(\lambda_t)}{\#(V_\sigma)} \tag{4.5}$$

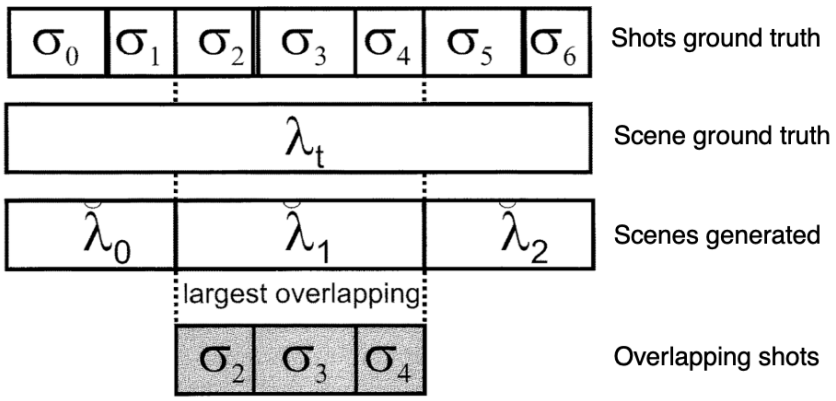


Figure 4.1: [61] Visual representation of Coverage *computation*

The *overflow* $O(\lambda_t)$ quantifies the amount of overlap of every $\breve{\lambda}_j$ corresponding to λ_t, with its two surrounding scenes λ_{t-1} and λ_{t+1}. In the numerator, each value of the summation is not null only if an overlapping between $\breve{\lambda}_j$ and λ_t exists. The denominator is expressed by the sum of the length in terms of shot of the two surrounding scenes respect to λ_t. Only $\#(\lambda_1)$ is considered as denominator for the computation of $O(\lambda_0)$ whereas only $\#(\lambda_{n-1})$ is considered for $O(\lambda_n)$. The overall *Overflow* O of the video is computed with the same formulation as the overall *Coverage*: each $O(\lambda_t)$ is multiplied by a proper ratio and summed together. The best case is achieved when $O = 0$ and each detected scene correspond to the ground truth one, hence $\breve{\lambda}_j = \lambda_t$.

For each scene:

$$O(\lambda_t) = \frac{\sum_{j=0}^{n} \#(\frac{\breve{\lambda}_j}{\lambda_t}) \cdot \min(1, \#(\breve{\lambda}_j \cap \lambda_t))}{\#(\lambda_{t-1}) + \#(\lambda_{t+1})} \tag{4.6}$$

For the entire video:

$$O = \sum_{t=0}^{\#(V_\lambda)-1} O(\lambda_t) \cdot \frac{\#(\lambda_t)}{\#(V_\sigma)} \tag{4.7}$$

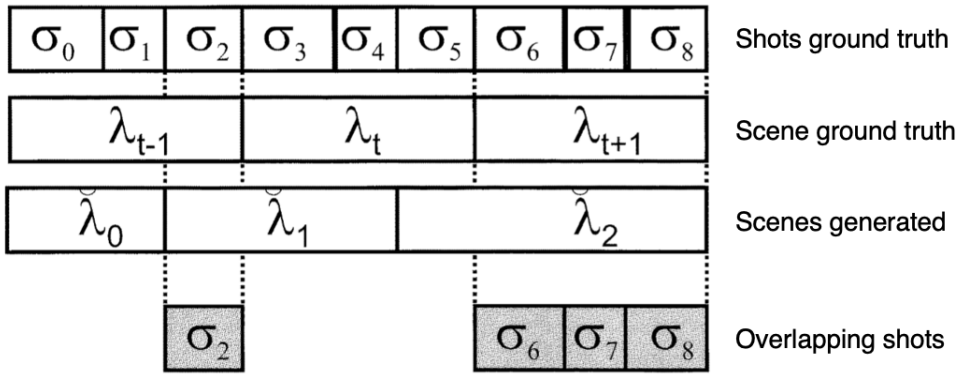


Figure 4.2: [61] Visual representation of Overflow *computation*

The $F - score$ is defined as the harmonic mean of C and $1 - O$:

$$F = 2\frac{(1-O) \cdot C}{(1-O) + C} \tag{4.8}$$

These metrics show drawbacks which may affect the evaluation. $F - score$ is not symmetric as noted by Sidiropulous et al. [53] so an early or late positioning of the scene boundary of the same amount of shots can lead to very different results. In addition, an unwarranted dependency between an error and the length of scene observed many shots before the error is caused by the relation of the *Overflow* with the previous and next scenes. The *Coverage* does not penalize other overlapping scene out of the maximum one: the measure value is not modified by any other over-segmentation in the overlapping scenes [6]. Sometimes the total *Overflow* O results

greater than 1 and consequently the *F-score* results negative. To solve this issue an improvement of the *Overflow* measure is proposed in (4.1.3).

4.1.3 Coverage*, Overflow*, F-score*

Baraldi et al. [8] proposed a modification of the previous formulation to solve the drawbacks mentioned in Section 4.1.2. Being computed at the shot level, an error on a small shot is given the same importance of an error on a very long shot. In order to compute *Coverage** and *Overflow** at frame level, the cardinality operator # is substituted with the number of frames of a scene $l(\lambda_t)$.

Since the amount of the $Overflow$ error should be related to the current scene length instead of that of its two neighbors, then $O(\lambda_t)$ is normalized with respect to the length of the current scene λ_t instead of that of the previous λ_{t-1} and the latter ones λ_{t+1}.

The amount of *Overflow** is then limited to one: this assures that our *Coverage** and *Overflow** belong to [0, 1]. Therefore, the *Coverage** is defined as follows.

For each scene:

$$C^*(\lambda_t) = \frac{\max\limits_{j=0\ldots n} l(\breve{\lambda}_j \cap \lambda_t)}{l(\lambda_t)} \tag{4.9}$$

For the entire video:

$$C^* = \sum_{t=0}^{l(V_\lambda)-1} C(\lambda_t) \cdot \frac{l(\lambda_t)}{l(V_\sigma)} \tag{4.10}$$

The *Overflow** is defined as follows.

For each scene:

$$O^*(\lambda_t) = \min\left(1, \frac{\sum_{j=0}^{n} l(\frac{\breve{\lambda}_j}{\lambda_t}) \cdot \min(1, l(\breve{\lambda}_j \cap \lambda_t))}{l(\lambda_{t-1}) + l(\lambda_{t+1})}\right) \tag{4.11}$$

For the entire video:

$$O^* = \sum_{t=0}^{l(V_\lambda)-1} O(\lambda_t) \cdot \frac{l(\lambda_t)}{l(V_\sigma)} \tag{4.12}$$

Finally, it is defined $F - score^*$ as the harmonic mean between C^* and O^*. The F^* measure is limited in [0,1] range:

$$F^* = 2\frac{(1 - O^*) \cdot C^*}{(1 - O^*) + C^*} \tag{4.13}$$

The drawback of these measures comes from a characteristic that should have been a strength: the *Overflow* limitation in the range [0, 1] causes an information loss because each greater value is indistinctly clipped to 1.

4.1.4 Shot level Coverage*, Overflow*, F-score*

Based on the method proposed in Section 4.1.3, we made a change in order to apply the measures at shot level, by maintaining the range of the O between 0 and 1.
Since the original formulation of the *Coverage* (4.4)(4.5) gives results in [0,1] range, it is not modified.
For each scene:

$$O^*_{shot}(\lambda_t) = \min\left(1, \frac{\sum_{j=0}^{n} \#(\frac{\breve{\lambda}_j}{\lambda_t}) \cdot \min(1, \#(\breve{\lambda}_j \cap \lambda_t))}{\#(\lambda_{t-1}) + \#(\lambda_{t+1})}\right) \tag{4.14}$$

For the entire video:

$$O^*_{shot} = \sum_{t=0}^{\#(V_\lambda)-1} O^*_{shot}(\lambda_t) \cdot \frac{\#(\lambda_t)}{\#(V_\sigma)} \tag{4.15}$$

As seen before, the $F - score^*_{shot}$ is defined as the harmonic mean of the C^*_{shot} and $1 - O^*_{shot}$:

$$F^*_{shot} = 2\frac{(1 - O^*_{shot}) \cdot C^*_{shot}}{(1 - O^*_{shot}) + C^*_{shot}} \tag{4.16}$$

The main drawback of these measures compared to the Baraldi formulation is that, being taken at shot level, it is impossible to discern between an error computed on a very long or a short shot.

4.1.5 Differential Edit Distance

The final scene segmentation evaluation measure is the *Differential Edit Distance* (DED) presented by Sidiropoulos [53]. The authors noted that *Coverage* and *Overflow* are not symmetric: when a boundary is detected ϵ shots before the ground truth border, it produces a different score from the one delivered by a boundary detected ϵ shots after the ground truth border. Therefore, they proposed a uni-dimensional measure which models the scene segmentation as a label assignment problem, without the drawbacks of combining with the harmonic mean two different metrics like *Coverage* and *Overflow* or *precision* and *recall*.

Thanks to these characteristics we chose DED as the performance index for scene segmentation.

Given a set B_E of automatically detected scenes and a set B_G of ground truth scenes, a set $B = 0, B_E \cap B_G, N$ is computed, where N is the total number of shots of the video. The video is decomposed in $|B| - 1$ sub-videos, where $|\cdot|$ is the cardinality operator. Each sub-video, called SV_b, where $b = 1, 2 \ldots |B| - 1$, starts at the $B(b) + 1$ shot and ends at the $B(b + 1)$ shot included. For each sub-video, a co-occurrence matrix CM_b is computed, where each element $CM_b(i, j)$ is determined by the number of shots that belongs to both the i^{th} ground truth scene v_i^g and the j^{th} automatically detected scene v_j^e. Then, it is defined a cost matrix CC_b where each element is given by $\widehat{CM_b} - \widehat{CM_b(i, j)}$, where $\widehat{CM_b}$ is the maximum element of the matrix $CM_b(i, j)$. The cost matrix is given as an input to the Hungarian algorithm in order to obtain the element combination that determines the minimum sum when it is chosen exactly one element from each row and each column.

By applying the element combination obtained from the Hungarian algorithm, it is possible to compute W_{bas}, the optimal matching between the ground truth and automatically detected scenes of SV_b. Therefore, the number of shots that will not change

scene labels is computed as follows:

$$N_{w_b} = \sum_{(v_i^g, v_j^e) \in W_b} CM_b(i,j) \tag{4.17}$$

Finally, the *Differential Edit Distance* is obtained. DED measures the distance between the ground truth set of scenes and the automatically detected scenes as the minimum number of shots that need to be changed to transform the generated set in the ground truth set:

$$DED = \frac{N - N_w}{N} \tag{4.18}$$

where $N_W = \sum_{b=1}^{|B-1|} N_{W_B}$.

It should be noted that since DED is a dissimilarity measure in range $[0, 1]$, lower values of the metric corresponds to better scene detection performances and 0 is the value of the exact segmentation.

4.2 Shot detection results

In the Tables 4.1 and 4.2, the comparison between the Baraldi et al. algorithm (3.1.1) and Gygli algorithm (3.1.2) is reported. The performance is measured with the $F1 - measure$ (4.1.1), that combines *recall* and *precision*, on both *Rai* (2.5.2) and BBC datasets (2.5.3). These datasets are chosen for the evaluation because they contain the list of ground truth shots of each video.

The results show that, on average, the Gygli method reaches the best performances on both datasets. The greater performance gain is obtained in the BBC dataset and also the majority of the videos in the *Rai* datatset achieve an improvement with respect to Baraldi et al. implementation. Therefore, Gygli method is selected for the following scene detection tests.

Video	F1-measure	
	Baraldi et al.	**Gygli**
rai01	0,8601	0,8746
rai02	0,9661	0,6609
rai03	0,8581	0,7318
rai04	0,7830	0,8131
rai05	0,8006	0,6755
rai06	0,2669	0,7297
rai07	0,4566	0,6386
rai08	0,6732	0,7160
rai09	0,7722	0,8148
rai10	0,8278	0,8188
Mean	**0,7265**	**0,7474**

Table 4.1: Shot detection on the Rai dataset

Video	F1-measure	
	Baraldi et al.	**Gygli**
bbc01	0,3414	0,3530
bbc02	0,1692	0,7471
bbc03	0,5753	0,7633
bbc04	0,2254	0,6338
bbc05	0,3659	0,6329
bbc06	0,3903	0,6417
bbc07	0,3667	0,5733
bbc08	0,5307	0,5954
bbc09	0,5866	0,7168
bbc10	0,4796	0,3415
bbc11	0,2836	0,6845
Mean	**0,3922**	**0,6076**

Table 4.2: Shot detection on the BBC dataset

4.3 Relationship between $\mathcal{H}_{nrm}$ and evaluation metrics

It is important to note that smaller values of $\mathcal{H}_{nrm}$ does not always imply better values of DED. In the Table 4.3, the values of $\mathcal{H}_{nrm}$ and DED (better as closer to 0) are computed by using $R18$ (3.4.4) and RS (3.5.2) methods with the middle frame selection algorithm (3.2.1). The reported results are related to the video Tears of Steel from the $OVSD$ dataset. Since it is employed the same frame selection method, the two algorithms operates on the same distance matrix D, so the $\mathcal{H}_{nrm}$ values are comparable.

In the last column of the table is computed, as a reference, the cost function and

DED value of the Ground Truth division (GT).

tos	**RS**	**R18**	**GT**
$\boldsymbol{\mathcal{H}_{nrm}}$	19,3638	19,4858	20,1879
DED	0,4271	0,4010	0,0000

Table 4.3: Correlation between H_{nrm} and DED in the video tos

The value of $\mathcal{H}_{nrm}$ related to the ground truth division results higher than the ones computed with RS and $R18$, even though it obviously scores the best possible value of DED, since it is the exact division. Moreover, RS outperform $R18$ in terms of cost function but its DED is worse than the one of $R18$. This behaviour could affect the results shown in Sections 4.4 and 4.5.

4.4 Frame selection results

The methods proposed in Section 3.2 are tested in order to see if a type of key frames selection that systematically improves the quality of the scene division exists. For this test, the RS_B optimal algorithm is performed on the videos of the datasets $OVSD$ (2.5.1), Rai (2.5.2) and BBC (2.5.3) that have a smaller amount of shots and scenes. The cost function value $\mathcal{H}_{nrm}$ is not reported in the table since depends from the matrix D. In the same video, the different frame selection methods can generate diverse values of D, so the resulting costs $\mathcal{H}_{nrm}$ may not be comparable.
The results are reported in Table 4.4, where MF stands for the middle frame method (3.2.1), CL stands for our clustering method (3.2.2), RH and HH (3.2.3) stand for respectively RGB histogram method, HSV histogram method and CEM, GEM, CED, GED (3.2.4) stand for respectively maximum color entropy method, maximum grey-scale entropy method, color entropy difference method and grey-scale entropy difference method.

Video	DED							
	MF	**CL**	**RH**	**HH**	**CEM**	**GEM**	**CED**	**GED**
1000d	0,3087	0,3087	0,3087	0,3087	0,3153	0,3268	0,3120	0,3218
bbb	0,2458	0,3184	0,2291	0,2291	0,2905	0,2235	0,2626	0,2626
bwnsP1	0,2966	0,3169	0,3124	0,2966	0,3213	0,3236	0,3146	0,3213
cl	0,2895	0,2939	0,2895	0,2851	0,2895	0,2895	0,2895	0,2982
ed	0,3602	0,3093	0,3475	0,3347	0,3305	0,3602	0,3347	0,3347
fbwP2	0,3419	0,3162	0,3419	0,3419	0,3547	0,3034	0,3077	0,3077
honey	0,3773	0,4862	0,4126	0,3712	0,3988	0,4202	0,4433	0,4310
jw	0,3059	0,2993	0,3092	0,2961	0,3059	0,3059	0,3059	0,3454
lcdp	0,4671	0,4740	0,4671	0,4671	0,4602	0,4671	0,4637	0,4637
lmP1	0,4101	0,4185	0,4101	0,4213	0,4466	0,4326	0,4213	0,4213
meridian	0,1770	0,1770	0,1770	0,1770	0,1681	0,1593	0,1504	0,1770
oceaniaP1	0,4170	0,4332	0,4211	0,4291	0,4251	0,4211	0,4413	0,4372
pentagon	0,3255	0,3962	0,3255	0,3255	0,3632	0,3585	0,3962	0,3962
route66P2	0,6120	0,6215	0,6120	0,6120	0,6230	0,6088	0,6199	0,6073
sdm	0,2204	0,2204	0,2204	0,2163	0,2163	0,2122	0,2204	0,2204
sintel	0,4085	0,4366	0,2254	0,2254	0,3979	0,3134	0,2817	0,2817
ssP2	0,3819	0,3643	0,3769	0,3116	0,3920	0,3819	0,4070	0,4070
swP1	0,4290	0,4079	0,4260	0,4139	0,4169	0,4199	0,4079	0,4290
tos	0,4271	0,4271	0,4271	0,4271	0,3906	0,4740	0,4219	0,4219
valkaamaP1	0,1694	0,1564	0,1889	0,1857	0,1661	0,1694	0,1726	0,1694
bbc06P1	0,3754	0,4164	0,4006	0,3849	0,3533	0,3628	0,3438	0,3438
bbc07P1	0,3086	0,3765	0,3457	0,3457	0,3333	0,3302	0,3364	0,3333
bbc10P1	0,4018	0,3720	0,4018	0,3988	0,3542	0,3750	0,4018	0,4018
rai01	0,4961	0,4961	0,4961	0,4961	0,4567	0,4724	0,4961	0,4961
rai02	0,2778	0,2222	0,2361	0,2778	0,2639	0,3194	0,2639	0,2639
rai03	0,1765	0,1765	0,1008	0,1681	0,1765	0,1933	0,1849	0,1849
rai04	0,5232	0,4834	0,4702	0,4702	0,4636	0,4305	0,4702	0,4437
rai05	0,4275	0,4420	0,4493	0,4420	0,4130	0,4130	0,4058	0,4130
rai06	0,0615	0,0615	0,0615	0,0615	0,0769	0,0615	0,0615	0,0615
rai07	0,3433	0,3358	0,3134	0,3881	0,3358	0,3134	0,3806	0,3806
rai08	0,3411	0,3411	0,3131	0,3271	0,3178	0,3505	0,3505	0,3505
rai09	0,3604	0,5225	0,3333	0,3333	0,3243	0,3423	0,3423	0,3423
rai10	0,2477	0,3028	0,2477	0,2477	0,2477	0,2385	0,2294	0,2294
Mean	**0,3428**	**0,3555**	**0,3333**	**0,3338**	**0,3391**	**0,3386**	**0,3407**	**0,3424**

Table 4.4: Comparison between the frame selection methods

The results show that do not exist a method that significantly outperform the others. Although on average RH slightly obtain the best DED score, we chose MF for the following scene detection tests since it is the faster and less computationally expensive alternative yielding a good overall performance. It is also the method chosen by Rotman et al. [48].

4.5 Dynamic Programming Results

4.5.1 Comparison between *RecursiveSolver* and *Rotman18*

In the Table 4.5 is reported a comparison between the performances of the $Rotman18$ method ($R18$) (3.4.4) and our $RecursiveSolver$ algorithm (RS) (3.5.2) on 10 videos taken from the $OVSD$ dataset. The performance is evaluated by taking into account the value of the cost function $\mathcal{H}_{nrm}$, the amount of the execution time expressed in seconds CPU and the quality of the division DED (4.1.5)(better as closer to 0).

Video	N	K	$\mathcal{H}_{nrm}$		DED		CPU(s)	
			RS	R18	RS	R18	RS	R18
bbb	176	15	17,3672	17,4521	0,2458	0,3128	125,27	1527,13
cl	228	7	18,0569	18,0868	0,2895	0,3114	44,89	1583,36
ed	235	9	19,7270	19,8936	0,3602	0,4068	96,74	2717,89
jw	298	15	13,6265	13,6723	0,3059	0,3092	670,59	14027,06
lcdp	285	11	16,7798	16,8429	0,4671	0,4740	291,38	7900,20
meridian	111	9	14,8752	14,9025	0,1770	0,2035	8,92	115,97
pentagon	207	31	14,1204	14,2635	0,3255	0,4481	865,05	5659,74
sdm	245	34	12,3332	12,6282	0,2204	0,1959	1794,62	12614,44
sintel	282	8	20,2604	20,3293	0,4085	0,4507	126,57	4694,83
tos	192	11	19,3638	19,4858	0,4271	0,4010	82,73	1490,64

Table 4.5: Comparison between RS and R18

As it can be seen in Table 4.5, RS reaches lower values of the cost function while strongly reduces the amount of execution time. Moreover, even though there is not always a complete correlation between $\mathcal{H}_{nrm}$ and DED, as stated in (4.3), in most of the cases RS outperform $R18$ in terms of DED.

The time gain between the two algorithms, indicated as $\Delta CPU(\%)$, is shown in the Table 4.6. RS takes, on average, about 93,15% less computational time compared to $R18$.

Video	N	K	CPU (s)		$\Delta CPU(\%)$
			RS	R18	
bbb	176	15	125,2694	1527,1261	-91,80%
cl	228	7	44,8867	1583,3614	-97,17%
ed	235	9	96,7416	2717,8852	-96,44%
jw	298	15	670,5914	14027,0596	-95,22%
lcdp	285	11	291,3755	7900,1963	-96,31%
meridian	111	9	8,9214	115,9713	-92,31%
pentagon	207	31	865,0492	5659,7427	-84,72%
sdm	245	34	1794,6153	12614,4441	-85,77%
sintel	282	8	126,5727	4694,8322	-97,30%
tos	192	11	82,7311	1490,6404	-94,45%
Mean					**-93,15%**

Table 4.6: Execution time of RS and R18

4.5.2 Comparison between *RecursiveSolver with Bounds* and *RecursiveSolver*

Next, in Tables 4.7, 4.8, 4.9 a comparison between the performance of our *RS* (3.4.4) and *RecursiveSolver with Bounds* (*RS_B*) (3.5.3) algorithms is proposed. The idea is to verify if the bounding procedure proposed in *RS_B* can further improve the performances.

All the videos of the datasets *OVSD*, *Rai* and *BBC* are employed for the analysis. Some of the videos are splitted in two parts since the whole video tables initialization would have exceeded the amount of available RAM.

Video	N	K	$\mathcal{H}_{nrm}$	DED	CPU (s)		$\Delta CPU(\%)$
					RS_B	RS	
1000d	591	22	16,8422	0,3087	8722,1666	12519,9514	-30,3339%
bbb	176	15	17,3672	0,2458	89,6579	125,2694	-28,4279%
bwnsP1	443	19	17,1507	0,2966	2641,1335	3765,2189	-29,8545%
bwnsP2	460	17	13,2266	0,3796	2314,2985	3397,8519	-31,8894%
ch7P1	446	17	17,7943	0,1951	2159,8891	3033,7819	-28,8054%
ch7P2	516	15	17,5962	0,2201	2623,7736	3671,3264	-28,5334%
ch7P3	643	12	17,5177	0,3406	3173,3005	4360,3101	-27,2231%
cl	228	7	18,0569	0,2895	35,2197	44,8867	-21,5364%
ed	235	9	19,7270	0,3602	74,1574	96,7416	-23,3449%
fbwP1	407	20	18,7462	0,5157	2250,4223	3146,4107	-28,4765%
fbwP2	231	21	17,2635	0,3419	409,9304	581,4186	-29,4948%
fbwP3	157	21	16,0733	0,4337	115,6430	162,8316	-28,9800%
honey	640	20	16,5827	0,3773	9083,8766	13110,5561	-30,7133%
jw	298	15	13,6265	0,3059	458,4474	670,5914	-31,6354%
lcdp	285	11	16,7798	0,4671	204,4261	291,3755	-29,8410%
lmP1	349	17	17,1855	0,4101	978,5202	1410,1076	-30,6067%
lmP2	422	10	14,2486	0,3052	551,9802	778,8913	-29,1326%
meridian	111	9	14,8752	0,1770	6,7501	8,9214	-24,3387%
oceaniaP1	242	20	15,1056	0,4170	428,1183	598,2940	-28,4435%
oceaniaP2	495	11	17,2491	0,2430	1151,5709	1537,5954	-25,1057%
pentagon	207	31	14,1204	0,3255	593,2154	865,0492	-31,4241%
route66P1	494	20	18,3717	0,3765	4071,5850	5796,0563	-29,7525%
route66P2	633	12	17,1039	0,6120	2982,9217	4045,4641	-26,2650%
route66P3	351	14	15,6728	0,4375	677,8579	912,6446	-25,7260%
route66P4	461	9	18,5728	0,3707	589,7311	754,5787	-21,8463%
sdm	245	34	12,3332	0,2204	1217,0567	1794,6153	-32,1829%
sintel	282	8	20,2604	0,4085	95,0020	126,5727	-24,9428%
ssP1	473	12	17,5115	0,2055	1218,7129	1720,7831	-29,1768%
ssP2	391	14	16,2504	0,3819	935,6445	1343,5325	-30,3594%
ssP3	507	11	17,6659	0,5828	1247,7604	1720,7584	-27,4878%
ssP4	253	14	16,9861	0,3152	239,3487	338,9287	-29,3808%
swP1	329	15	17,8280	0,4290	635,6854	901,1133	-29,4556%
swP2	553	13	18,8032	0,4521	2330,5954	3333,3930	-30,0834%
swP3	665	12	18,9655	0,4138	3450,0879	4888,5282	-29,4248%
swP4	338	15	17,5309	0,4647	688,9519	955,8350	-27,9215%
tos	192	11	19,3638	0,4271	60,0637	82,7311	-27,3989%
valkaamaP1	307	20	16,0319	0,1694	921,6756	1326,7000	-30,5287%
valkaamaP2	225	15	15,7012	0,3004	189,0821	263,5681	-28,2606%
valkaamaP3	233	16	15,8855	0,3278	241,6613	339,8978	-28,9018%
Mean							-28,3907%

Table 4.7: OVSD dataset: comparison between RS_B and RS

Video	**N**	**K**	$\mathcal{H}_{nrm}$	**DED**	**CPU (s)**		$\Delta CPU(\%)$
					RS_B	**RS**	
rai01	127	7	18,3423	0,4961	5,8126	7,0927	-18,0483%
rai02	71	12	15,6418	0,2778	3,2188	4,0933	-21,3645%
rai03	119	16	15,6968	0,1765	28,3600	39,9170	-28,9526%
rai04	142	22	15,9035	0,5232	90,6581	131,4844	-31,0503%
rai05	134	13	12,5960	0,4275	26,8756	37,7929	-28,8872%
rai06	65	5	18,2996	0,0615	0,3438	0,3281	4,7847%
rai07	134	9	18,6254	0,3433	12,5002	16,4045	-23,7997%
rai08	214	12	19,5602	0,3411	102,0803	142,5317	-28,3807%
rai09	110	14	13,6815	0,3604	16,6722	23,2633	-28,3326%
rai10	106	16	11,7488	0,2477	19,2660	26,9036	-28,3887%
Mean							**-23,2420%**

Table 4.8: Rai dataset: comparison between RS_B and RS

Video	N	K	$\mathcal{H}_{nrm}$	DED	CPU (s)		$\Delta CPU(\%)$
					RS_B	RS	
bbc01	519	23	17,3694	0,2933	6330,6632	9320,6644	-32,0793%
bbc02	428	36	15,2789	0,3341	8371,3294	12920,7706	-35,2103%
bbc03	461	33	16,1106	0,3548	8961,5147	13421,2663	-33,2290%
bbc04	562	30	16,3501	0,3606	14109,1078	20404,7844	-30,8539%
bbc05	584	25	16,5185	0,3618	11035,4178	16168,1980	-31,7461%
bbc06P1	314	19	17,6417	0,3754	901,1440	1284,2457	-29,8309%
bbc06P2	361	14	18,0266	0,3453	742,1718	1031,8809	-28,0758%
bbc07P1	323	17	15,3655	0,3086	788,5322	1111,5389	-29,0594%
bbc07P2	395	20	17,0702	0,3924	2075,4500	2924,5770	-29,0342%
bbc08	509	29	15,8192	0,4325	9693,2646	13997,1077	-30,7481%
bbc09	404	33	15,1882	0,3603	6017,7513	8621,3043	-30,1991%
bbc10P1	333	14	17,2225	0,4018	561,1520	823,6734	-31,8720%
bbc10P2	508	8	17,4661	0,4392	606,4813	813,2669	-25,4265%
bbc11	529	26	17,4038	0,3590	8560,1473	13208,9808	-35,1945%
Mean							**-30,8971%**

Table 4.9: BBC dataset: comparison between RS_B and RS

Both methods reach the same value of cost function $\mathcal{H}_{nrm}$ and DED, so these values are reported only once.

The results above show a sensible improvement in execution time of RS_B respect to RS while maintaining the optimal solution. As the amount of shots N and scenes K increase, the difference in execution time between RS and RS_B is larger. In case of a small amount of shots (< 70) and scenes (< 5), RS could be slightly faster than RS_B as can be seen for the video *rai06*. Typically, such amount of N and K does not reflect a real case scenario. Moreover, the results show that on the videos of the 3 datasets RS_B takes, on average, about 30% less computational time than RS.

Chapter 5

Conclusions

In this work, the VSD problem has been faced starting from the Rotman et al. [48] innovative approach. We reproduced their workflow and proposed a reformulation of the dynamic programming method that brought us to develop two alternative algorithms: the first only based on dynamic programming, and the other enhanced with a branch and bound heuristic. The results show that these methodologies lead to an improvement in term of cost function optimality, execution time and accuracy of scene division.

This is the main result obtained from a wider study on the scene detection task and its related steps that incorporated the development and testing of methodologies of shot detection and frame selection. In particular we analyzed several algorithms for shot boundary detection and we experimentally observed that the Gygli method [23] is more effective than the one employed in the Rotman et al. paper (Baraldi et al. [8]).

As regards the frame selection, the tested methods do not significantly outperform the one proposed in the Rotman et al. workflow.

Moreover, we performed an in-depth study on the scene division measures in order to find the suitable index that best represents the scene division accuracy.

As result, we created a promising workflow that audio describers can immediately apply on different genres of video in order to obtain a good scene division in a low amount of time. It could be used as a starting point for solving the scene detection task proposed in the AAD pipeline.

Bibliography

[1] S. H. Abdulhussain, A. R. Ramli, M. I. Saripan, B. M. Mahmmod, S. A. R. Al-Haddad, and W. A. Jassim. Methods and challenges in shot boundary detection: a review. *Entropy*, 20(4):214, 2018.

[2] B. Adams, C. Dorai, and S. Venkatesh. Toward automatic extraction of expressive elements from motion pictures: Tempo. *IEEE Transactions on Multimedia*, 4(4):472–481, 2002.

[3] M. Ahmed, A. Karmouch, and S. Abu-Hakima. Key frame extraction and indexing for multimedia databases. In *Vision Interface*, volume 99, pages 1–1, 1999.

[4] Y. Ariki, M. Kumano, and K. Tsukada. Highlight scene extraction in real time from baseball live video. In *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 209–214, 2003.

[5] L. Baraldi, C. Grana, and R. Cucchiara. Analysis and re-use of videos in educational digital libraries with automatic scene detection. In *Italian Research Conference on Digital Libraries*, pages 155–164. Springer, 2015.

[6] 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.

[7] L. Baraldi, C. Grana, and R. Cucchiara. Measuring scene detection performance. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 395–403. Springer, 2015.

[8] L. Baraldi, C. Grana, and R. Cucchiara. Shot and scene detection via hierarchical clustering for re-using broadcast video. In *International Conference on Computer Analysis of Images and Patterns*, pages 801–811. Springer, 2015.

[9] L. Baraldi, C. Grana, and R. Cucchiara. Recognizing and presenting the storytelling video structure with deep multimodal networks. *IEEE Transactions on Multimedia*, 19(5):955–968, 2016.

[10] Y. Bendraou. *Video shot boundary detection and key-frame extraction using mathematical models*. PhD thesis, 2017.

[11] G. J. Burghouts and J.-M. Geusebroek. Performance evaluation of local colour invariants. *Computer Vision and Image Understanding*, 113(1):48–62, 2009.

[12] E. Busarello and F. Sordo. Manuale per aspiranti audio descrittori di audiofilm per non vedenti. *Scurelle (TN): Cooperativa Sociale Senza Barriere ONLUS*, 2011.

[13] Z. Cernekova, I. Pitas, and C. Nikou. Information theory-based shot cut/fade detection and video summarization. *IEEE Transactions on circuits and systems for video technology*, 16(1):82–91, 2005.

[14] V. T. Chasanis, A. C. Likas, and N. P. Galatsanos. Scene detection in videos using shot clustering and sequence alignment. *IEEE transactions on multimedia*, 11(1):89–100, 2008.

[15] T. Cour, C. Jordan, E. Miltsakaki, and B. Taskar. Movie/script: Alignment and parsing of video and text transcription. In *European Conference on Computer Vision*, pages 158–171. Springer, 2008.

[16] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

[17] S. E. F. De Avila, A. P. B. Lopes, A. da Luz Jr, and A. de Albuquerque Araújo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 32(1):56–68, 2011.

[18] M. del Fabro and L. Böszörmenyi. Video scene detection based on recurring motion patterns. In *2010 Second International Conferences on Advances in Multimedia*, pages 113–118. IEEE, 2010.

[19] M. Del Fabro and L. Böszörmenyi. State-of-the-art and future challenges in video scene detection: a survey. *Multimedia systems*, 19(5):427–454, 2013.

[20] A. Farahat, A. Ghodsi, and M. Kamel. A novel greedy algorithm for nyström approximation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 269–277, 2011.

[21] U. Gargi, R. Kasturi, and S. H. Strayer. Performance characterization of video-shot-change detection methods. *IEEE transactions on circuits and systems for video technology*, 10(1):1–13, 2000.

[22] Y. Gong and X. Liu. Video summarization using singular value decomposition. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 2, pages 174–180. IEEE, 2000.

[23] M. Gygli. Ridiculously fast shot boundary detection with fully convolutional neural networks. *arXiv preprint arXiv:1705.08214*, 2017.

[24] B. Han and W. Wu. Video scene segmentation using a novel boundary evaluation criterion and dynamic programming. In *2011 IEEE International conference on multimedia and expo*, pages 1–6. IEEE, 2011.

[25] A. Hanjalic and H. Zhang. An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis. *IEEE Transactions on circuits and systems for video technology*, 9(8):1280–1289, 1999.

[26] M. Haroon, J. Baber, I. Ullah, S. M. Daudpota, M. Bakhtyar, and V. Devi. Video scene detection using compact bag of visual word models. *Advances in Multimedia*, 2018, 2018.

[27] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

[28] H.-C. Huang, Y.-Y. Chuang, and C.-S. Chen. Multi-affinity spectral clustering. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2089–2092. IEEE, 2012.

[29] U. ITC. Itc guidance on standards for audio description. Technical report, Technical Report: Independent Television Commission, 2000.

[30] H. Ji, D. Hooshyar, K. Kim, and H. Lim. A semantic-based video scene segmentation using a deep neural network. *Journal of Information Science*, 45(6):833–844, 2019.

[31] T. Kikukawa and S. Kawafuchi. Development of an automatic summary editing system for the audio-visual resources. *Transactions on Electronics and Information J75-A*, pages 204–212, 1992.

[32] R. M. Kishi, T. H. Trojahn, and R. Goularte. Temporal video scene segmentation by fused bags-of-features. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, pages 173–180, 2018.

[33] R. W. Lienhart. Comparison of automatic shot boundary detection algorithms. In *Storage and Retrieval for Image and Video Databases VII*, volume 3656, pages 290–301. International Society for Optics and Photonics, 1998.

[34] R. W. Lienhart. Reliable dissolve detection. In *Storage and Retrieval for Media Databases 2001*, volume 4315, pages 219–230. International Society for Optics and Photonics, 2001.

[35] C. Liu, D. Wang, J. Zhu, and B. Zhang. Learning a contextual multi-thread model for movie/tv scene segmentation. *IEEE transactions on multimedia*, 15(4):884–897, 2013.

[36] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[37] D. Moreira, S. Avila, M. Perez, D. Moraes, V. Testoni, E. Valle, S. Goldenstein, and A. Rocha. Multimodal data fusion for sensitive scene localization. *Information Fusion*, 45:307–323, 2019.

[38] A. Nagasaka and Y. Tanaka. Automatic video indexing and full-video search for object appearances. *Journal of Information Processing*, 15(2):316, 1992.

[39] J. Nam and A. H. Tewfik. Combined audio and visual streams analysis for video sequence segmentation. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2665–2668. IEEE, 1997.

[40] R. Panda, S. K. Kuanar, and A. S. Chowdhury. Nyström approximated temporally constrained multisimilarity spectral clustering approach for movie scene detection. *IEEE transactions on cybernetics*, 48(3):836–847, 2017.

[41] X. Peng, R. Li, J. Wang, and H. Shang. User-guided clustering for video segmentation on coarse-grained feature extraction. *IEEE Access*, 7:149820–149832, 2019.

[42] S. Protasov, A. M. Khan, K. Sozykin, and M. Ahmad. Using deep features for video scene detection and annotation. *Signal, Image and Video Processing*, 12(5):991–999, 2018.

[43] S. Rai, J. Greening, and L. Petré. A comparative study of audio description guidelines prevalent in different countries. *Londra: Royal National Institute of Blind People*, 2010.

[44] Z. Rasheed and M. Shah. Detection and representation of scenes in videos. *IEEE transactions on Multimedia*, 7(6):1097–1105, 2005.

[45] A. Remael, N. Reviers, and G. Vercauteren. *Pictures painted in words: ADLAB Audio Description guidelines*. 2015.

[46] 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.

[47] D. Rotman, D. Porat, and G. Ashour. Robust video scene detection using multimodal fusion of optimally grouped features. In *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE, 2017.

[48] D. Rotman, D. Porat, G. Ashour, and U. Barzelay. Optimally grouped deep features using normalized cost for video scene detection. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, pages 187–195, 2018.

[49] Y. Rui, T. S. Huang, and S. Mehrotra. Constructing table-of-content for videos. *Multimedia systems*, 7(5):359–368, 1999.

[50] U. Sakarya and Z. Telatar. Video scene detection using dominant sets. In *2008 15th IEEE International Conference on Image Processing*, pages 73–76. IEEE, 2008.

[51] B. Shahraray. Scene change detection and content-based sampling of video sequences. In *Digital Video Compression: Algorithms and Technologies 1995*, volume 2419, pages 2–13. International Society for Optics and Photonics, 1995.

[52] T.-Y. Shih. The reversibility of six geometric color spaces. *Photogrammetric Engineering and Remote Sensing*, 61(10):1223–1232, 1995.

[53] P. Sidiropoulos, V. Mezaris, I. Kompatsiaris, and J. Kittler. Differential edit distance: A metric for scene segmentation evaluation. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(6):904–914, 2011.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[55] D. Swanberg, C.-F. Shu, and R. C. Jain. Knowledge-guided parsing in video databases. In *Storage and retrieval for Image and Video Databases*, volume 1908, pages 13–24. International Society for Optics and Photonics, 1993.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[57] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[58] M. Tkalcic and J. F. Tasic. *Colour spaces: perceptual, historical and applicational background*, volume 1. IEEE, 2003.

[59] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[60] V. TRUONGBT. Video abstraction: Asystematicreviewand classification. *ACM-Transon Multimedia Computing*.

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

[62] C. Wang, H. Yang, C. Bartz, and C. Meinel. Image captioning with deep bidirectional lstms. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 988–997, 2016.

[63] J. Xu, L. Song, and R. Xie. Shot boundary detection using convolutional neural networks. *2016 Visual Communications and Image Processing (VCIP)*, pages 1–4, 2016.

[64] R. Zabih, J. Miller, and K. Mai. A feature-based algorithm for detecting and classifying scene breaks. In *Proceedings of the third ACM international conference on Multimedia*, pages 189–200, 1995.

[65] Y. Zhai and M. Shah. Video scene segmentation using markov chain monte carlo. *IEEE transactions on Multimedia*, 8(4):686–697, 2006.

[66] Y. Zhai, A. Yilmaz, and M. Shah. Story segmentation in news videos using visual and text cues. In *International Conference on Image and Video Retrieval*, pages 92–102. Springer, 2005.

[67] H. Zhang, A. Kankanhalli, and S. W. Smoliar. Automatic partitioning of full-motion video. *Multimedia systems*, 1(1):10–28, 1993.

[68] J. Zheng, F. Zou, and M. Shi. An efficient algorithm for video shot boundary detection. In *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, pages 266–269. IEEE, 2004.

[69] Y. Zhuang, Y. Rui, T. S. Huang, and S. Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269)*, volume 1, pages 866–870. IEEE, 1998.

Appendix A

Test environment and framework libraries

The project is developed in Python 3.7 on a Desktop PC with a 3.6 GHz Intel ®Core^{TM}i7 processor, 24GB RAM and 64-bit Windows 10 operating system.
The result reported in this work are obtained by executing the tests on a Microsoft Azure Virtual Machine with 8 physical core (64 virtual cores) of an Intel Xeon Platinum 8168 (SkyLake) CPU with a frequency 2.7GHz (3.7GHz in Turbo Boost), 64GB RAM and 64-bit Windows 10 operating system.

A.1 Anaconda environment

The Python environment is set up thanks to Anaconda[1] which is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics).
Anaconda comes with more than 1,500 packages easy to install via Anaconda Naviga-

[1] https://www.anaconda.com/

tor(GUI) or via command line interface (CLI). Also, Anaconda offers the possibility to set up a separate environment to run a different version of Python and install the libraries of interest. All the possible available packages are on Anaconda Cloud.
We chose Jupyter Notebook [2] as the Python editor and Anaconda Prompt to run the test from command line. The versions selected for the Anaconda environment are:

Name	Version
Anaconda	2019.03
Conda	4.6.11
Python	3.7.3
Jupyter Notebook	6.0.0

A.2 Libraries

The default Anaconda (base) environment does not include all the libraries required for the implementation of this project, so we created a dedicated environment in which we installed all the required libraries.

Keras

In order to deal with machine learning part of the work, Keras is employed. It is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. The modules required are Keras Preprocessing and Keras Applications. Keras Preprocessing provides utilities for working with image data, text data, and sequence data.
Keras Applications provides model definitions and pre-trained weights for a number of popular architectures, such as InceptionV3, VGG16, ResNet50, Xception, MobileNet, and more. If the PC supports GPU, the use of keras-gpu library is recommended to

[2] https://jupyter.org/

speed up the process of the feature extraction.
Version: 2.2.4

Tensorflow

TensorFlow[3] is an open source software library for numerical computation using data-flow graphs developed by the Google. Tensorflow is used as a back-end for Keras. The package is included in the Keras. It also available tensorflow-gpu suitable for computer equipped with GPU.
Version: 1.13.1

OpenCV

OpenCV[4] (Open Source Computer Vision Library) is an open source library that has more than 2500 optimized algorithms for computer vision and machine learning tasks. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.
Version: 3.4.2

Scikit-learn

Scikit-learn[5] offers several efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. It can be used to build models. This library is based on Numpy, Scipy and Matplotlib.
Version: 0.21.3

[3]https://www.tensorflow.org/
[4]https://opencv.org/
[5]https://scikit-learn.org/

Pandas

Pandas[6] is an open source library for data manipulation and management written for the Python programming language.
Version: 0.24.2

Other Libraries

Name	Version
Matplotlib	3.1.0
Numpy	1.16.4
Pillow	6.1.0
Scipy	1.5.0
Scikit-image	0.15.0

[6]https://pandas.pydata.org/