Achieving Real-Time Video Summarization on Commodity Hardware

by

Wesley Taylor

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

in

The Faculty of Science
Computer Science

University of Ontario Institute of Technology

Supervisor: Dr. Faisal Z. Qureshi

April 2018

Abstract

Achieving Real-Time Video
Summarization on Comodity Hardware

Wesley Taylor

Master's Thesis

Faculty of Science (Computer Science)

University of Ontario Institute of Technology

2018

We present a system for automatic video summarization which is able to operate in real-time on commodity hardware. This is achieved by performing segmentation to divide a video into a series of small video clips, which are further reduced or eliminated with the assistance of highly efficient low-level features. A numerical score is then assigned to each segment by our model trained using a set of high-performance hand-crafted features. Finally, segments are selected based on their score to generate a final video summary. On our benchmark dataset, we achieve results competitive to other methods. In cases where our accuracy is lower than competitive methods, we achieve significantly higher performance. We additionally present methods for generating additional summaries almost instantly, and for learning user preferences over time—two processes which are often overlooked in work on video summarization, but essential for real-world use.

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

This chapter serves to provide a general overview of our work. We start by giving a general summary of our research topic, along with describing the motivation behind our work. We describe some possible applications of our work, as well give an overview of an additional project—Cliply—which was developed alongside our work as a practical example application. We conclude by stating our primary contributions, and providing an overview of the general structure of our system.

With video capable mobile devices becoming increasingly ubiquitous, we are seeing an analogous increase in amount of video data that is captured and stored. Additionally, as the difficulty of capturing video and cost of storage decreases, we tend to see a corresponding decrease in the quality of captured videos. As a result of this, it becomes very difficult to locate interesting video clips among the vast sea of data. One solution to this problem lies in the development of a video summarization system which is able to automatically locate these interesting clips and generate a final, curated video summary.

1.1 Motivation

With the appearance of consumer devices such as action cameras, we have reached a point where virtually no effort is required to record large amounts of video data. Although this is a significant accomplishment, it also brings with it a new problem—the videos recorded often require significant editing before they are in a state suitable for viewing. This editing process is non-trivial for an average user, often requiring both expertise with video editing software, and access to specialized hardware. This makes a video summarization system that is able to operate on an average user's machine (commodity hardware) extremely desirable.

Additional problems lie in both the length of videos that are being recorded, and their actual content. In recent years, the cost of digital storage has decreased significantly, while both the resolution and length of videos has increased—a standard action camera video will be recorded at 1080p, and be many hours long in duration. The actual process of hand-editing this video could itself easily require hours of work even for a skilled user. This limitation means that most users will not have time to edit every video they record, resulting in an ever-growing backlog of videos which will never be viewed. A video summarization system able to operate at real-time speeds would be able to process videos in an online manner, effectively meaning that a backlog would never develop.

In terms of content, these modern “raw user” videos are unlike the older “edited” videos which were popular previously in that effectively no effort is made to separately record different events. For example, with an edited video, the operator of the camera would traditionally choose to stop recording if nothing interesting is happening, then resume recording again when they expect something interesting to happen. With raw user videos, the operator of the camera will simply record everything that happens, interesting or not, often resulting in a final video consisting of a few interesting segments, and many more uninteresting ones. A method which is able to efficiently distinguish between these two types of segments would be a significant step towards a video summarization system which is able to generate high-quality summaries for raw user videos.

Finally, different viewers will have different preferences on what they find interesting in a video. With traditional hand-editing, viewers only see the segments deemed interesting by the editor, and the time cost for a human editor to create multiple edits of a single video is significant. The ability to generate multiple possible summaries rather than just a single summary would be a very useful feature for a video summarization system to have, as would be the ability to learn a specific user’s preferences over time.

1.2 Applications

Video summarization has a wide range of applications, including:

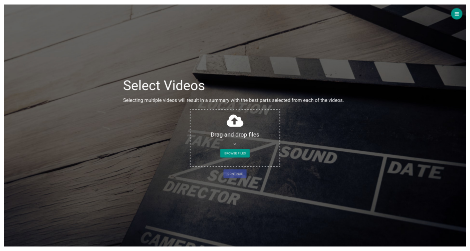
Personal video stories. People often want to share videos with their family and friends for important events such as a vacation or wedding, but showing the entire video may be boring, and editing the video to extract the interesting clips would be time-consuming. Automated video summarization would be able to generate a short summary of the most interesting parts of an event, even taking into account high-level visual information such as which people are present in specific clips.

Sports highlight reels. Currently, sports highlight reels are manually created by domain experts, and only focus on the most popular highlights. Automatic summarization of these videos has the potential to understand domain knowledge for specific sports, and generate multiple highlight reels for different viewers. For example, in hockey, some users may want to see highlights which contain tackles, while other users may wish to see highlights containing goals.

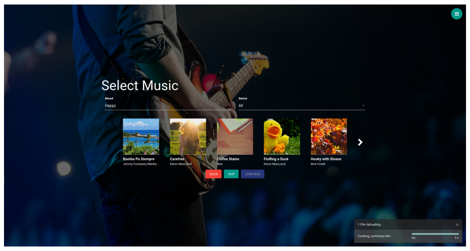
Automated movie and television trailers. Movie trailers are created with the goal of convincing viewers to watch the full movie, but have the disadvantage that they must generally appeal to the largest number of people possible. By using automated video summarization methods to generate multiple trailers, the system would be able to over time learn to personalize trailers for specific groups of people, for example different age demographics, potentially increasing the number of people that end up watching the movie.

1.3 Cliply

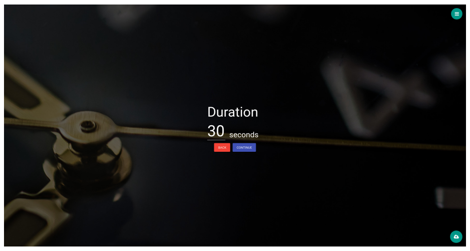
Alongside our work on video summarization, we developed a web-based system which allows users to upload their videos, and generate short "video stories" from them. Although the system incorporates much of our work on video summarization, it also adds a large range of features to enhance raw video summaries and make them more desirable to end users. Some examples include support for adding music and for overlaying a title over the final summary video. Some example screenshots from the upload phase of Cliply can be found in Figure 1.1.



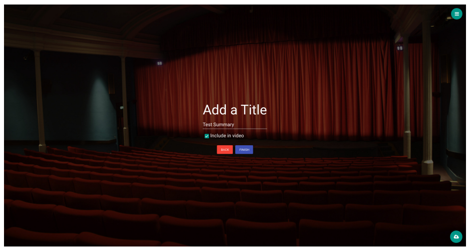
(a) The upload page for Cliply.



(b) The Cliply page used to select music for a summary.



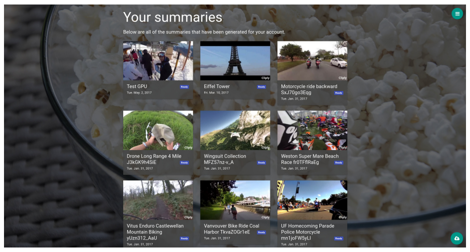
(c) The Cliply page used to pick a duration for the final summary.



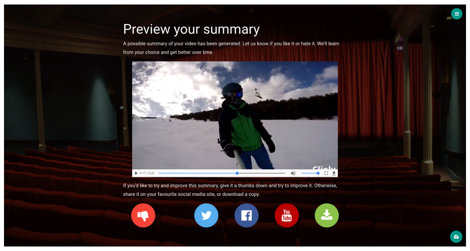
(d) The Cliply page used to select a title, and optionally include it in the final summary video.

Figure 1.1: A visual summary of the four steps a user follows to create a summary video in the Cliply system. Users first upload one or more videos using the page in (a). They are then able to optionally select a music track for the video on the page in (b), a duration for their summary on the page in (c), and finally select a title for their summary on the page in (d), and optionally include it in the final summary video.

Access to this system was extremely helpful during our work on video summarization. By developing our work as extensions to Cliply's automatic summarization system, we were able to avoid a large amount of the boilerplate code traditionally required for re-generating intermediate data and re-testing our work when significant changes occurred. We were able to simply start-up a new local instance of Cliply, add our testing videos, and the underlying system would take care of generating all intermediate and final data. Additionally, we were able to view all the processed videos in Cliply's web-interface, shown in Figure 1.2.


Your summaries
Below are all of the summaries that have been generated for your account.

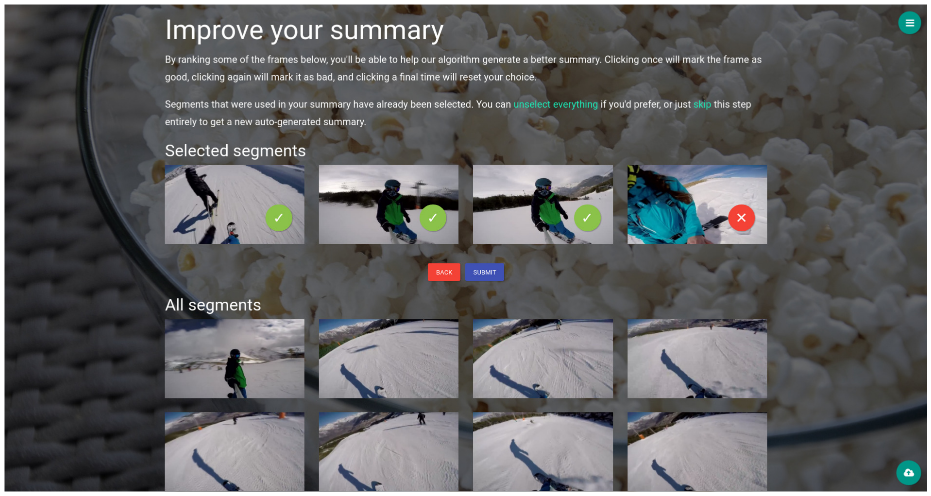
(a) The Cliply page which shows a history of all videos a user has submitted.



(b) The Cliply page for viewing a video summary. From here, users can preview their summary, refine it using the "thumbs down" icon, share it to popular social media sites, or download a copy of it.

Figure 1.2: A demonstration of the interface provided by Cliply for viewing video summaries and their details.

A final benefit of the Cliply system is that it was able to provide us with a user-friendly interface for the "Additional Summaries" step in our summarization system. This step requires optionally marking multiple segments of a video as either "keep" or "discard". Without Cliply, we would need to manually look through each segment, and construct an input file. With Cliply however, we are able to use the simple interface demonstrated in Figure 1.3 to view and select segments, automatically create the input file, and even generate the new summary video.


Improve your summary

By ranking some of the frames below, you'll be able to help our algorithm generate a better summary. Clicking once will mark the frame as good, clicking again will mark it as bad, and clicking a final time will reset your choice.

Segments that were used in your summary have already been selected. You can unselect everything if you'd prefer, or just skip this step entirely to get a new auto-generated summary.

Figure 1.3: The interface provided by Cliply for selecting video segments to specifically keep or discard. Any segments specifically selected are highlighted at the top, while a listing of all segments is provided in the "All Segments" section. A green check mark on a segment indicates it should be kept and a red cross indicates that it should be discarded. If no mark is present, the system will decide to keep or discard it. Clicking on a segment image switches it's state between unselected, keep, or discard.

1.4 Notation

In our work, the lowest level we work at is the video-level, where we have a video V, consisting of a number c of ordered frames f, that is, $V = \{f_0, \ldots, f_c\}$. Each video can also carry with it a number of attributes—in particular, the number of frames in the video $\textit{frames}(V)$, the frame rate of the video $\textit{fps}(V)$, and the dataset a video belongs to $\textit{dataset}(V)$.

Each frame f is represented as a 3-dimensional array of pixel values $p \in [0, 255]$, with the third dimension representing the number of color channels in the image, and hence the color space of the image. The most common cases we deal with are 3 channels, where the image is in the RGB color space, or 1 channel, where the image is in the grayscale color space. In terms of attributes, each frame has a width in pixels $\mathit{width}(f)$, a height $\mathit{height}(f)$, and a channel count $\mathit{channels}(f)$.

Branching off of this base notation for videos, there are a few additional definitions specific to our work in video summarization which are important. The first of these is for that of a segment s which represents an ordered range of frames $\{f_a, \ldots, f_b\}$ from a video V, with $a, b < \mathit{frames}(V)$ and $a < b$. Each segment carries with it a number of attributes, including the number of frames within a segment $\mathit{frames}(s)$, the index of the first frame of the segment $\mathit{start}(s) = a$, and the index of the last frame of the segment $\mathit{end}(s) = b$. We can also define the distance between two segments s_c and s_d with $\mathit{start}(s_d) < \mathit{end}(s_c)$ as $\mathit{distance}(s_c, s_d) = \mathit{start}(s_d) - \mathit{end}(s_c)$.

For a video V, we can have a segmentation S as $S_V = \{s_0, \ldots, s_k\}$, consisting of non-overlapping segments s, that is, $\mathit{end}(s_i) < \mathit{start}(s_{i+1})$. Finally, we can represent our end goal—a summarization U—as a possibly equal subset of this segmentation, that is, $U_V \subseteq S_V$. We often need to reference the set of all the frames within a segmentation or summary, for which we use the notation $\mathit{frames}(S)$ and $\mathit{frames}(U)$ respectively.

We often need to describe n-dimensional feature vectors for an associated object o. For this purpose, we have adapted the notation X_o^n. For example, a 128-dimensional feature vector for frame f_i would be represented by $X_{f_i}^{128}$.

For a segment s_i, we use the notation $Q_{s_i} \in [0, 1]$ to represent the score computed for a segment, where 1 represents an interesting/high-quality segment, and 0 an uninteresting/low-quality segment.

1.5 Contributions

The primary contribution of our work is:

> A high performance video summarization system which is able to perform video summarization at real-time on commodity hardware.

In addition to this primary contribution, parts of our work also serve as relevant contributions in isolation. These include:

1. A video pre-processing process which makes use of very low-level features to efficiently locate undesirable frames, then uses these to compute optimal segments.
2. A method of incorporating user history over time in order to learn to generate personalized video summaries.
3. A method of generating additional summaries almost instantaneously.

1.6 Overview

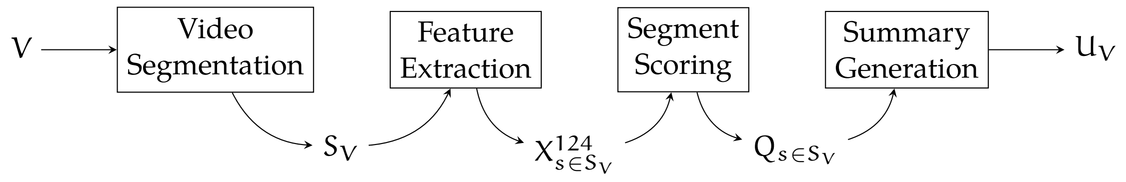


Figure 1.4: Overview of our summarization system. We start with a video V and perform video segmentation, resulting in a segmentation S_V. We perform feature extraction on this segmentation to obtain for each segment $s \in S_V$ a 124-dimensional feature vector X^{124}_s. This feature vector is used by the segment scoring process to generate for each segment $s \in S_V$ a score Q_s. Finally, these scores are used by the summary generation process to generate a final summary U_V.

Our video summarization system follows a tiered approach, where methods with a high performance but lower level of detail are first used as a rough filter for frames and segments of the video. This allows us to quickly eliminate a number of problem frames and segments which commonly appear in raw user videos. By reducing the number of frames that need to be processed, we are able to make use of more computationally expensive methods later on, while maintaining similar performance.

Generally, video summarization consists of four major steps:

1. **Video Segmentation** is described in Chapter 3, where for a target video V, a segmentation S_V is generated. This step includes a pre-processing step to eliminate undesirable frames.

2. **Feature extraction** is described in Chapter 4, where features are first extracted for each frame, then aggregated within each segment s to obtain a

124-dimensional features vector X_s^{124}.

3. **Segment scoring** is described in Chapter 5, where based on the extracted features and a segment scoring model, each segment s is scored with a float value $Qs \in [0, 1]$. A value of 1 represents a high-quality segment, while a value of 0 represents a low-quality one.

4. **Summary generation** is described in Chapter 6, where a selection algorithm is used to construct a final summary U_V. This step also includes the generation of additional summaries.

The majority of our implementation is written in C++ for performance reasons, but there are also parts which are written in Python, such as our machine learning models.

Once we have described our system, we present our evaluation method and results in Chapter 7, finishing up with our conclusion in Chapter 8, where we summarize our contributions, discuss some limitations of our work, and provide some ideas for future work that could be performed to expand on our system.

CHAPTER 2

Background Information

This chapter presents an overview of some background information that may be needed for future chapters. We start by looking at some of the previous research related to our work. We then give an overview of some of the common datasets used for video summarization, including any relevant pre-processing we performed on the data. Finally, we include an overview of some computer vision and machine learning techniques used later in our work.

Although video summarization has been an active research topic in computer vision for decades, it has experienced a renewed interest in recent years. The high computational capabilities of modern hardware allow us to process video in a fraction of the time previously required, which when combined with the evolution of modern vision techniques such as deep neural networks, has resulted in a significant increase in the breadth of techniques which are viable to apply to the topic of video summarization. Combined with the vast quantity of prior work involving video summarization, many interesting research prospects are available to pursue.

2.1 Related Works

Although our primary focus is video summarization, some of the steps we perform during our work are themselves topics with a significant amount of prior research. Among these are video segmentation, which simply deals with taking a video and dividing it up in to a number of segments, and image and video feature extraction, which deals with extracting relevant and useful features from images and videos.

2.1.1 Video Segmentation

Video segmentation has many years of research behind it, resulting in the creation of a variety of algorithms over the years. Early works such as [1, 2, 3] focused on detecting scene transitions in edited videos—for example fades or dissolves—while more modern works such as [4, 5, 6, 7] instead focus on the more difficult problem of segmenting raw user videos.

The classic thresholding-based segmentation algorithm was originally proposed in [1], and operates by first computing the average brightness value of each frame in a video. Segments are then detected by comparing each frame brightness to a pre-selected threshold value, and creating a new segment whenever the brightness falls below the threshold. An improvement is also proposed in [2] which makes the threshold value dynamic and able to adapt to global lighting changes over time.

Overall, this method is very efficient, but only useful for very simple transitions such as fades when scenes are separated by a series of low brightness frames.

The work in [3] is perhaps first major improvement to early threshold-based methods, and instead examines motion and intensity differences between frames to perform segmentation. Rather than examining only values for a single frame, differences over multiple adjacent frames are considered, and segments are created whenever the values match any of a set of previously observed scene transition patterns. This method is able to detect a larger range of transitions, in particular those caused by fast camera movements which may be present in raw user videos.

More recently, the two segmentation methods [4, 5] have been proposed with the express goal of performing segmentation in the case of raw user videos. The method first proposed in [4] extracts a number of features from each frame of a video, then performs agglomerative clustering[8] on these features to generate the final segmentation. The other work proposed in [5] takes a slightly different approach, instead extracting optical flow and blurriness features from each frame, then using a pre-trained classifier to classify frames as either “static”, “in transit”, or “changing attention”. Both of these methods are significant improvements over the previous work involving raw user videos.

Perhaps the most recent work is that which applies multiple change-point detection[6, 7] to the problem of video segmentation. In this work, a number of features are extracted for each frame of a video, and a matrix is formed containing all the features for every video. An optimization is performed using this matrix, and the result is the positions of any relevant segment boundaries. A fundamental difference between this method and most other methods is that change-point detection operates on the entire signal at once, meaning that the resulting segmentation has a higher chance of being globally consistent.

2.1.2 Image and Video Features

Feature extraction is an important part of our work, as the accuracy of our model is dependent on our ability to efficiently extract features from our videos which are relevant to the task of video segment scoring. When efficiency is a concern, there are a large range of low-level hand-crafted features[9, 10, 11, 12, 13, 14, 15] which are relevant to our task. More recent work on video summarization has additionally incorporated higher-level features[16, 17, 18, 19] such as SIFT features and dense motion trajectories, and even very high-level features[20, 21, 22] such as object detection and neural network layer features.

Research in the field of computational aesthetics[9, 10] provides us a large range of low-level features for assessing the "beauty" of images. Some of these, such as [11] are inspired by psychology and art theory, and attempt to compute approximate emotional values, while others such as [12] attempt to compute features which describe the general texture of an image. Some other important features that may be used include image sharpness[14] and the rule-of-thirds based on computing spectral saliency[15] over 9 quadrants of a frame.

As hardware improves, so does the complexity of features which are used for video summarization. Modern works tend to use some combination of general image features such as GIST[17] and SIFT[18], robust motion features such as dense trajectories[19], and even general models for image aesthetics[16]. With the recent popularity of deep learning, we are even starting to see the use of features constructed using computationally complex methods based on neural networks, such as face detection[20], object detection[22], and layer extraction from networks such as GoogLeNet[21].

2.1.3 Video Summarization

Although many methods have been proposed for video summarization over the years, the basic underlying process has generally remained the same. Specifically, the variety of methods[4, 23, 24, 25] first compute a segmentation, then perform scoring of these segments, and finally use 0/1 knapsack[26] to perform segment selection for the final summary. One exception to this is some of the state-of-the-art work which uses LSTM neural networks[27]. Many early works[4, 23, 25] were unsupervised methods, but with the recent appearance of some high-quality video summarization datasets, supervised methods[24, 27] which learn some model based on previous summary data are now the most popular.

Commonly, methods based on clustering[4] and attention[23] are used as baselines when evaluating new methods. Both of these methods are unsupervised. The clustering method involves first extracting a number of features and using clustering to generate a segmentation. For each segment, an additional set of features are extracted, an interestingness score is computed for each segment, and 0/1 knapsack is used to create a final summary. The attention method operates similarly, extracting attention features for each frame which act as interestingness scores, then generating a summary using 0/1 knapsack. These two methods tend to obtain higher accuracy values compared to a randomly generated summary, but are overtaken by most modern methods.

The work performed in [24] is significant, as not only did it develop a state-of-the-art method for video summarization, but it also created a high-quality dataset to be used for benchmarking video summarization methods. In this work, segmentation was performed using change-point detection, segments were scored using a combination of low and high-level features, then the final summary was generated using 0/1 knapsack. The method used in [25] is very similar this method, with the primary differences being that their work performs some pre-processing of the video data in an attempt to improve efficiency, and they use a different set of features for scoring segments.

The current state-of-the-art results on the SumMe dataset were recently obtained using an LSTM neural network[27]. This method is significantly different than previous methods. In particular, the fact that LSTMs operate on sequences of frames rather than either individual frames or individual segments means that there is no need to perform segmentation. Additionally, wheras most methods only generate interestingness scores at the segment-level, the LSTM is actually able to generate per-frame values. This method operates by first extracting the output of the pool 5 layer of the GoogLeNet model as features, then using these as input to the LSTM. The result of this is per-frame interestingness scores, which can then be used in combination with 0/1 knapsack to generate a final summary. The primary benefit of this method is that it actually operates on the sequence of frames in a video, rather than just some aggregation of features over a group of frames.

2.2 Datasets

Below, we describe the primary datasets we use in our work, and discuss any pre-processing that needed to be performed.

2.2.1 A Large-Scale Database for Aesthetic Visual Analysis (AVA)

A significant part of our work is reliant on the ability to accurately compute the "aesthetics" of arbitrary frames of a video. For this purpose, we have found the dataset A Large-Scale Database for Aesthetic Visual Analysis (AVA)[28] to be an essential asset. It contains over $250,000$ images along with aesthetic scores and labels for various semantic and photographic categories. There has previously been a large range of work which makes use of this dataset for the purpose of computing the aesthetics of arbitrary photographs, however, our method deals with a categorically different range of images compared to those seen in traditional (and even modern) computer vision—ones from low-quality, user-recorded videos.

2.2.1.1 Dataset Processing

The base AVA dataset contains multiple files, however, we are primarily interested in the contents of the `AVA.txt` file. This file is a CSV file containing 15 columns, of which only 11 are of interest to us, specifically columns 2 up to and including 12. Column 2 contains the image ID which is needed to download the actual image content from the DPChallenge website. The remaining ten columns contain the number of votes submitted for each level of aesthetic quality, that is, the third column contains the number of votes which rank the image with an aesthetics value of 1, the fourth column the count for a ranking of 2, up to the twelfth column for the 10 ranking count.

Since our primary goal for this dataset is to learn a model for determining the aesthetics ranking of an arbitrary image, there are two important steps we needed to perform before this was possible:

1. Downloading all the images.
2. Computing the aesthetics score of each image.

Images were downloaded using a Python script and saved as JPEGs. Of the 255530 images in the dataset, only 195128 were still available for download. To compute the final score of each image, we simply computed the normalized weighted mean of the votes for each image, giving us a final floating point aesthetics value in the range $[0, 1]$.

2.2.2 SumMe (from "Creating Summaries from User Videos")

The SumMe dataset[24] is the first video summarization dataset focusing specifically on raw user videos that contain a number of interesting events. It contains 25 videos, each of which contains at least 15 different user summaries, for a total of 390 summaries overall. This dataset is currently used as the standard benchmark for state-of-the-art video summarization methods.

2.2.2.1 Dataset Processing

The base dataset contains for each video, a single data file exported from Matlab in `mat` format. Each file has a `user_score` attribute which contains a 2D matrix where each row represents a frame, and each column a user. Each element of this matrix is either 0 if the frame was not selected for inclusion in the summary, or > 0 otherwise. We use Algorithm 1 to convert this raw data matrix into a set of summaries $\{U_0, \ldots, U_c\}$.

2.2.3 TVSum

The TVSum dataset[29] consists of 50 videos from Youtube belonging to various genres. Each video is divided into 2 second long shots which are ranked by 20 users for importance on a scale of 1 to 5. In order to obtain user summaries from this dataset, we use 0/1 knapsack to select for each user's importance scores, a summary with length equal to 15% of the original video.

2.2.4 VSUMM

The VSUMM dataset[30] is an older dataset focusing on static summaries rather than dynamic. This means that each summarization is represented as a number of key frames, rather than a final video clip. This dataset consists of 100 videos from two different sources: 50 from Openvideo, and 50 from Youtube. Each video contains 5 summaries as a list of the frames which represent the static summary. To generate dynamic summaries, we simply create uniform segments centered at each keyframe to obtain a final summary with length equal to 15% of the original video.

2.3 Computer Vision Techniques

In our work, there are a number of techniques specific to computer vision which are prerequisites to some of the more advanced methods examined later in our work. To assist readers, we have created this section to provide a summary of these techniques, along with any assumptions we make.

2.3.1 HOG Features

Histogram of oriented gradients (HOG) features are commonly used in computer vision for the purpose of object detection. They operate by counting occurrences of gradient orientations in local regions of an image. Traditionally, for a given image, these are computed over a dense grid of uniformly spaced cells, usually 8×8 pixels in size. A detailed description of their computation method is beyond the scope of this work, and interested readers are directed to [31] for further information. The result of this method applied to an image is a feature vector consisting of 9 values for each cell, which represent the sum of the magnitude of the gradient vectors for each 20 degree angle bin in the range $[0, 180)$. HOG features tend to be desirable for object detection as they are invariant to changes in lighting and small deformations.

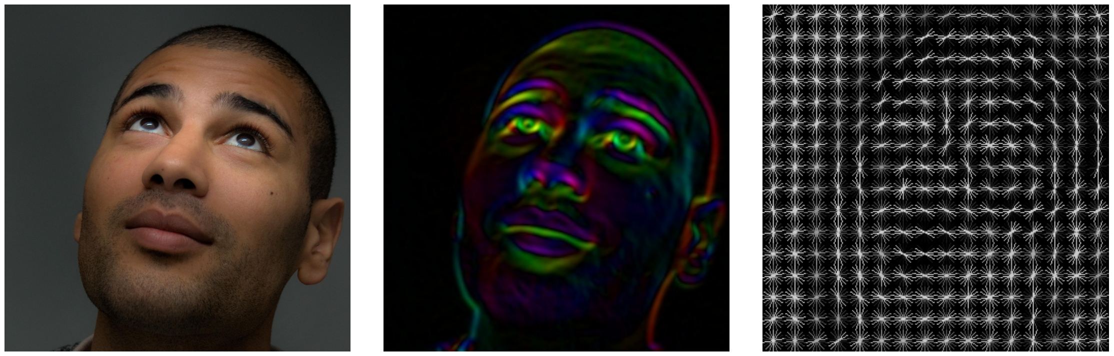


Figure 2.1: A visualization of both the HOG features and gradient image for an example image. The center image was created by computing the gradient of the leftmost image, and assigning a color to each pixel where the hue represents the angle of the gradient, and the intensity of the color represents the magnitude of the gradient. The rightmost image displays the traditional visualization used for HOG features. The image is divided into bins, and 9 white lines are drawn at different angles within each bin, with the intensity of the line representing the strength of the gradient for that specific angle.

A traditional visualization of these features is shown in Figure 2.1. In the right image, each 8×8 cell contains 9 white lines, oriented based on the orientation of the bin they represent, and with a length based on their magnitude in the resulting HOG feature histogram. We additionally display the original image on the left, and the gradient image in the center, where the color of each pixel represents the orientation of the gradient at that point, and the intensity represents the magnitude.

2.3.2 Decision Trees

A decision tree is a model which uses a tree-like graph of decisions and outcomes. Each decision is represented by a node in the graph, and has an associated test on an attribute. Each labeled branch out from a node represents the result of the attribute test. In most cases, there will just be two branches, one for true and one for false.

Finally, all leaf nodes of the graph contain the determined class for a path through the graph, and may optionally include a regression value $r = [0, 1]$ and percentage of observations $o = [0, 100]\%$ that reached the leaf during training.

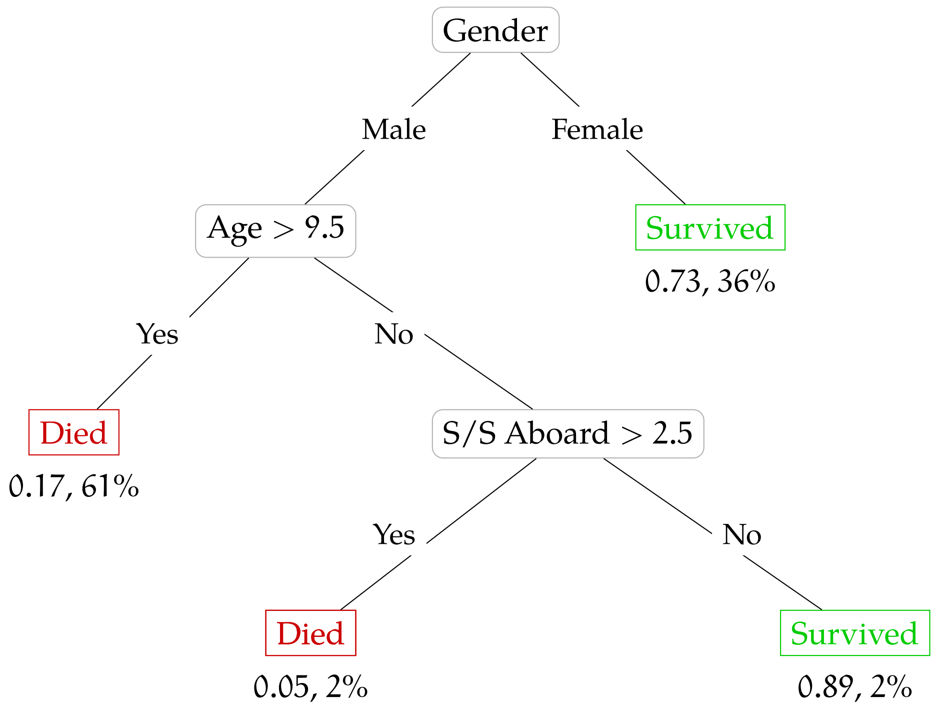


Figure 2.2: An example decision tree for predicting the survival outcome of passengers on the Titanic. In this case, we can see that the dataset is first split based on the value of the `gender` attribute. A value of `Female` results in a `Survived` leaf node with a probability of 0.73, meaning that a female on the Titanic had a 73% chance of survival. A value of `Male`, on the other hand, leads to another decision node. This process continues downwards until no attributes remain with the information gain of a split using them above some threshold value.

An example decision tree is presented in Figure 2.2, which represents the survival of passengers on the Titanic dataset[32]—one of the basic datasets traditionally used when describing the workings of decision trees. Decision nodes are represented by a gray rounded rectangle node and display the attribute test that is used. Leaf nodes are represented as colored rectangles, with one color for each distinct class.

Leaf nodes also include the regression value r and percentage of observations o as labels below them. The series of decisions that lead to a given class can be found by simply working backwards from each leaf node. The regression values, observation percents, and decisions followed to reach each conclusion can be seen in Table 2.1.

Class	**r**	**o**		**Decisions**	
Died	0.05	2%	Gender = Male,	Age $\leqslant 9.5$,	S/S Aboard > 2.5
	0.17	61%	Gender = Male,	Age > 9.5	
Survived	0.73	36%	Gender = Female		
	0.89	2%	Gender = Male,	Age $\leqslant 9.5$,	S/S Aboard $\leqslant 2.5$

Table 2.1: The regression values and observation percents for each class in our example decision tree, along with a list of the decisions which were followed to read each leaf node.

2.3.3 Information Gain in Decision Trees

Information gain is a metric used by many decision tree learning algorithms. It is based on the concept of entropy H from information theory, which was designed to measure the amount of information content a given set of examples belonging to different classes exhibit. Given a set of sample features $T = \{(x_1, x_2, \ldots, x_n, y)\}, y \in Y$ where x_i represents a single feature/attribute value and y represents the class of the feature, the entropy H can be simply defined as

$$H(T) = -\sum_{y \in Y} p_y \log_2 p_y.$$

Each p_y value represents the probability of the class y in the set T, that is, $p_y = |\{t \in T \mid t_y = y\}|/|T|$. We can additionally define conditional entropy of a sample set T

given an attribute value $x_i = a$ as

$$H(T|a) = \sum_{v \in \text{vals}(x_i)} \frac{|\{t \in T \mid x_i = v\}|}{T} \cdot H(\{t \in T \mid x_i = v\}).$$

Finally, we can define the information gain of a sample set T for a single attribute a as

$$IG(T, a) = H(T) - H(T|a).$$

Generally, this value is a good representation of the “relevance” of an attribute. In decision tree algorithms, this value is commonly used to decide which attribute should be used to split up a dataset.

2.3.4 Cross-Validation

Cross-validation is a technique used for the purpose of model validation to attempt to determine the ability of the model to generalize when provided with a set of data independent of the training data. For a typical prediction problem dealing with a set of data X with know results Y, the most basic form of model validation involves splitting the data into two sets—the training set X_t and Y_t, and the testing/evaluation set X_e and Y_e. The model is trained on the training set, and once training has completed, is evaluated on the testing set. This introduces, among other issues, the possibility for either training or testing to be performed on an unrepresentative subset of the data, resulting in either overfitting or an inaccurate computed accuracy metric.

The traditional solution is to perform a procedure known as k-fold cross-validation.

Using this method, the set of data is first split into k random equally-sized subsets X_i and Y_i with $i \in [1, k]$. Training and evaluation is then performed by looping over each $i \in [1, k]$, and

1. training a model m_j using a subset of folds $\bigcup(\{X_j \text{ and } Y_j \mid j \in [1, k], j \neq i\})$, followed by
2. testing the model using the excluded fold X_i and Y_i.

Once metrics have been computed for each i value, the average of each metric is returned as the final value. The value used for k should be adjusted based on various statistics of the input data such as the cardinality and the availability of computational resources.

2.3.5 Grid Search

The vast majority of machine learning methods are parameterized based on some "hyper-parameter" values $p \in P$. For most models, sane defaults are provided, however, simple tuning of these parameters to a specific set of training data very often results in an increase in model prediction accuracy. A common method for selecting optimal p values for a given algorithm a and set of data X is to perform "grid search", where a subset of possible values for each parameter $p \in P$ are provided, and for each set of values h_p in the Cartesian product $\{p_0 \times \cdots \times p_n \mid n = |P|\}$, the algorithm a is trained and evaluated using cross-validation.

Algorithm 1 The algorithm used for computing the summarizations for a video from the SumMe dataset.

```
    Inputs:
      $m$: A $f \times c$ user_score matrix
 1: function ExtractSummeSummaries($m$)
 2:     $W \leftarrow$ List( )                              ▷ Construct an empty list
 3:     for $i$ from $0$ below $c$ do
 4:         $U \leftarrow$ List( )                          ▷ Construct an empty list
 5:         prev $\leftarrow 0$
 6:         for $j$ from $0$ below $f$ do
 7:             if $m[j, i] > 0$ then
 8:                 if prev $> 0$ then
 9:                     increment segment end
10:                 else
11:                     append segment
12:                 end if
13:             else
14:                 if prev $> 0$ then
15:                     new segment
16:                 end if
17:             end if
18:         end for
19:     end for
20:     return $S$
21: end function
    Output: A set of summaries $W = \{U_0, \ldots, U_c\}$
```

CHAPTER 3

Video Segmentation

This chapter presents the processes we tested for performing video segmentation on a single video. Overall, we examine three of the primary segmentation methods used by traditional approaches [33, 34]. We additionally describe a more complex method which makes use of multiple change-point detection to dynamically adjust segment boundaries to better fit the underlying content. Since our primary goal is to develop a time-efficient summarization system, we additionally perform segment pre-processing and post-processing in an attempt to eliminate segments that are obviously non-desirable. The computational overhead of this additional processing is minimal, and has the potential for large computational savings later on in our summarization system.

The first step in our summarization system is segmentation, where for an input video V, a segmentation S_V is generated consisting of $k > 0$ non-overlapping segments $\{s_0, \ldots, s_k\}$. Although the segments are not allowed to overlap, it is possible for gaps to exist between segments, that is, we only require that $end(s_i) < start(s_{i+1})$. A visual representation of this process can be found in Figure 3.1.

Multiple approaches to segmentation have been developed over time, each with differing amounts of complexity and types of videos they work best with. For example, uniform sampling is perhaps the least complex, simply choosing segments of equal length with no gaps between them. More advanced methods actually attempt to locate the transitions between different "scenes" in a video, where the visual content has changed significantly. One such example of a method is multiple change-point detection[6], which has shown to be fairly accurate when locating scene transitions.

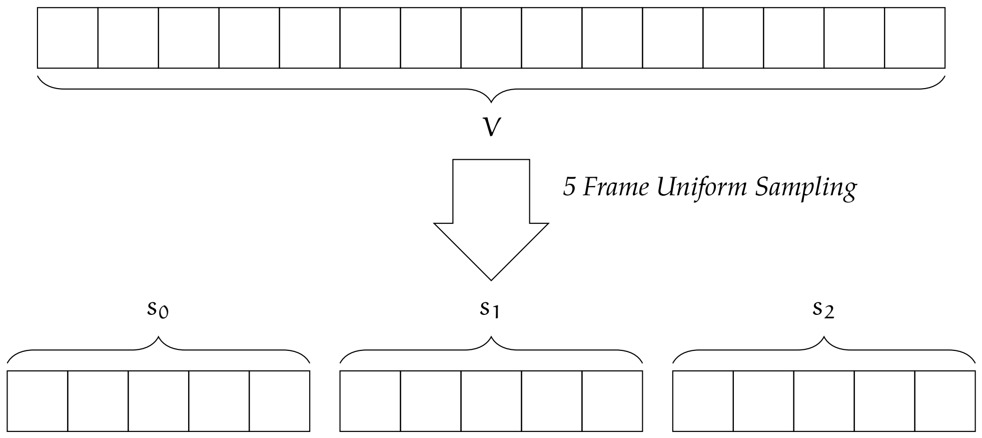


Figure 3.1: An example of the expected results from this step when using uniform sampling. We start with a video V with *frames*$(V) = 15$, and using uniform sampling with a length of 5 frames, we obtain a segmentation S_V consisting of $k = 3$ segments $\{s_0, s_1, s_2\}$.

Our entire segmentation process can be broken down into three distinct steps:

Video Pre-processing Low level frame features are extracted from the video and "labels" are optionally assigned via comparison to empirically derived threshold values.

Finding Initial Segments The video is initially divided into a number of candidate segments.

Segment Post-processing The candidate segments are processed further in an attempt to maximize the quality of the final segments, and eliminate any obviously undesirable segments.

The remainder of this chapter provides details of each step, including any options considered during the development of our pre-processing and post-processing processes.

3.1 Video Pre-processing

The first step we perform is video pre-processing, where low-level features are extracted for each frame of a video and are used to optionally assign initial "labels" to individual frames. In particular, we attempt to locate frames which are too dark, blurry, or have a high degree of uniformity. A visualization of the primary task performed by this step can be found in Figure 3.2.

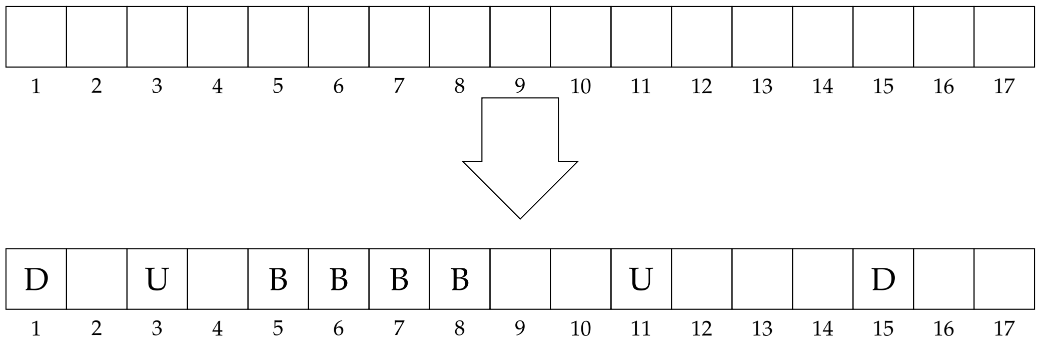
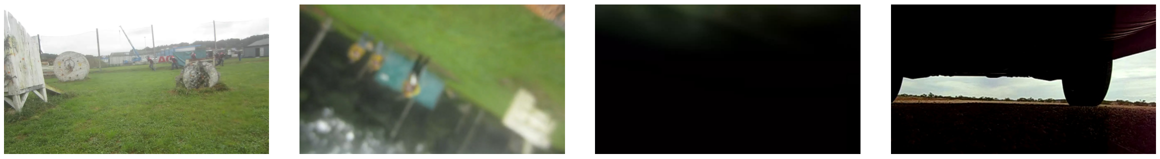


Figure 3.2: The results of applying pre-processing to an input video. The original video is shown at the top of the figure, consisting of 17 frames. At the bottom of the figure, we see the same video, but with some frames having labels applied to them, indicating that they were identified as “undesirable” by one of the pre-processing metrics.

3.1.1 Frame Labeling Features

The features we compute for the purpose of labeling frames have minimal computational requirements, and are thus discarded once they have been used. To perform frame labeling, we simply loop over each frame of the video, compute a number of feature values, then compare each to a previously empirically computed per-feature threshold value. Frames with feature values below this threshold value are considered undesirable, and assigned a label. Some example video frames, along with their computed feature values and corresponding labels can be found in Figure 3.3.



$Y = 0.63$	$Y = 0.45$	$Y = 0.02$	$Y = 0.14$
$S = 1955.33$	$S = 83.19$	$S = 3.42$	$S = 6552.91$
$U = 0.70$	$U = 0.75$	$U = 0.15$	$U = 0.24$
Label: *none*	Label: **blurry**	Label: **dark**	Label: **uniform**

Figure 3.3: Four example frames from a video, along with the values computed for each of the feature values, and the corresponding label assigned to the frame, if any. Any feature values which fall below the threshold value are highlighted in red.

Threshold values for each label were computed by hand-labeling approximately one hundred positive and negative example frames, then extracting the feature values, resulting in a set of feature values for the positive examples x_p and negative examples x_n. To minimize the false positive rate, the final threshold value is selected as $\min(\max(x_n), \min(x_p))$, that is, the largest value from the negative examples which is less than the smallest value from the positive examples.

3.1.1.1 Dark Frames

In order to locate frames with a low amount of illumination, the relative luminance $Y \in [0, 1]$ of each frame $f = (R, G, B)$ consisting of a red channel R, green channel G, and blue channel B is calculated as the average value over all pixels, that is,

$$Y = \text{mean}(0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B).$$

A frame is then labeled as "dark" if $Y \leqslant \alpha$ for some threshold value α. For our work, the optimal value of α was empirically determined to be 0.097. The third image in

Figure 3.3 is an example of a “dark” frame.

3.1.1.2 Blurry Frames

The sharpness S of each frame is computed using the Tenengrad method[35, 36], where frames with $S \leqslant \beta$ for some threshold value β are labeled as “blurry”. We empirically determined the optimal β value to be 502.32. To compute S, first, the horizontal image gradient G_x and vertical image gradient G_y of the grayscale version of each frame is calculated. We then calculate the final sharpness value

$$S = \text{mean}(G_x^2 + G_y^2)$$

as the mean magnitude over all pixels of these image gradients. The second image in Figure 3.3 is an example of a “blurry” frame.

3.1.1.3 Uniform Frames

A low uniformity value $U < \gamma$ for a frame is a general indication that the frame doesn’t contain a significant amount of meaningful information, as the intensity values over all pixels in the frame are very similar. These frames are labeled as “uniform”. A value of $\gamma = 0.2$ was empirically determined to be the optimal value for our purposes. The value for U is obtained by first computing the normalized, 1D, 128-bin grayscale histogram H of the image. We then compute the ratio between the top 5^{th} percentile values of H and the rest of H. In order to preserve the convention that threshold values should be upper bounds, the final value for U is then equal to

1 minus this computed value. The fourth image in Figure 3.3 is an example of a "uniform" frame.

3.2 Finding Initial Segments

In our work, we are mostly concerned with segmenting *raw user* videos—those which are usually captured by amateurs, using consumer devices such as smartphones or action cameras. This contrasts sharply with the vast majority of previous work in the field of video summarization, where *edited* videos with distinct "scenes" (such as TV shows and news programs) have traditionally been used.

Transitions in these raw user videos are significantly more difficult to detect compared to those in edited videos. Even current state-of-the art methods[33, 34] elect to use a basic method such as uniform sampling to select their initial segments, and rely purely on the later steps in their summarization system to refine their segment boundaries. In addition to examining these methods, we also reformulate our video as a multidimensional time-series sequence of features, allowing us to cast video segmentation as a multiple change-point detection problem and use modern signal processing methods.

3.2.1 Uniform Sampling

The most basic method that can be used to find the initial segments of a video involves simply dividing the video into a number of segments of equal length, usually in the range of 3 to 15 seconds each. Although this method is computationally efficient, the quality of segments it generates tend to be low.

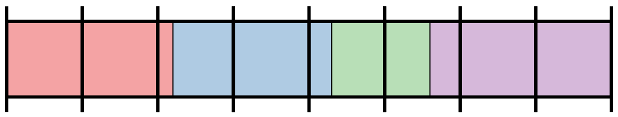


Figure 3.4: An example of uniform sampling applied to a video. The ground-truth scenes of the underlying video are indicated by blocks of different colors.

An example of the results this method produces can be seen in Section 3.2.1. It is important to note that the selected segments rarely correspond to the logical units (scenes) of the video, often only capturing part of an underlying scene, or even parts of multiple different scenes, such as in the third segment in Section 3.2.1. We elected not to use this method, as the generated segments were of very low quality.

3.2.2 Threshold-Based Scene Detection

Threshold-based detectors are among the simplest of scene detection methods which actually process the underlying image data. They operate by simply computing the average brightness value of every frame in the target video, and creating a new segment when this value falls below some threshold value. This method is very efficient and works well for edited videos containing abrupt scene transitions, that is, edited videos. However, for the case of the raw user videos, the method performs poorly, rarely selecting more than a couple of segments for long videos ($>$ 5 minutes in duration). A possible solution to this problem is to enforce a maximum segment duration, such as 5 seconds, although in practice, this would effectively result in uniform sampling.

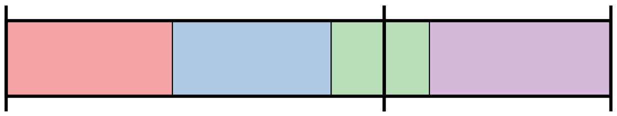


Figure 3.5: An example of threshold-based scene detection applied to a video. The ground-truth scenes of the underlying video are indicated by blocks of different colors. In this case, only two segments were selected in a video consisting of 4 underlying scenes. Furthermore, the boundaries of these segments are a large distance away from any ground-truth scene boundaries.

3.2.3 Content-Aware Scene Detection

Content-aware detection is similar to threshold-based detection, but rather than using the average brightness of each frame, it uses the average pixel value in the HSV color space of the image *difference* between two frames. As before, we loop over each frame and calculate this value, starting a new segment whenever we encounter a frame with a value that falls below some threshold value. The primary benefit of this method over threshold-based detection is that it is actually able to detect when the content of the frame *changes*, rather than just disappears (fades to black). For raw user videos, this method performs moderately well—it is able to capture scene transitions resulting from fast pans of the camera, but does miss slower pans. Any zooming that occurs in the video also tends to result in a new scene, which is only sometimes desirable.

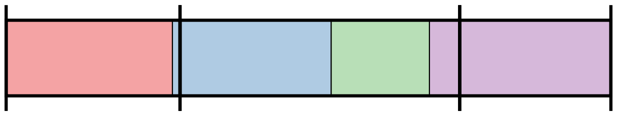


Figure 3.6: An example of content-aware scene detection applied to a video. The ground-truth scenes of the underlying video are indicated by blocks of different colors. Although this method only selects three of the four underlying scenes, the ones that it does select are of high quality, as they have boundaries very close to the ground-truth boundaries.

3.2.4 Change-Point Detection

In some of the most recent works on video segmentation—especially those involving user videos—a method based on multiple change-point detection[6] is used. The purpose of change-point detection when applied to a set of time series data is to locate any times where the probability distribution of the data changes significantly. For the case of videos, we use a feature matrix consisting of color and edge histograms for each frame of the video. Video segmentation is framed as a group fused least absolute shrinkage and selection operator (LASSO) problem, and we make use of the method originally described in the work by Bleakley and Vert [6], and optimized for use with long video sequences in the work by Song et al. [29].

As input, change-point detection requires a time series feature matrix $\mathbf{X}$, where each column represents a frame and each row the features for that frame. There are many options for the frame features that can be used, but keeping with our desire to achieve real-time performance, we elected to use relatively simple 2200-dimensional color and edge histogram features X_f^{2200}. For each frame f, we compute two primary features over a two-level pyramid consisting of five regions: (1) HSV histograms with 128-bins per channel, and (2) edge orientations and magnitudes with 30 bins for each. These are then concatenated to form a final feature vector X_f^{2200}. Once features have been extracted for all $n = \mathit{frames}(V)$ frames of V, we use them as the columns of a matrix, resulting in a final feature matrix $\mathbf{X} \in \mathbb{R}^{2200 \times n}$. It is possible to use more complicated features, such as the outputs from a layer of a neural network, but even for a relatively simple network such as VGG-19, the time required per-frame to extract these features is prohibitive.

Once we have our input feature matrix $\mathbf{X}$, a set of sparse coefficients $\mathbf{A} \in \mathbb{R}^{n \times n}$ are computed by solving the convex optimization problem

$$\arg\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{X}\mathbf{A}\|_F^2 + \frac{\lambda}{2}\|\mathbf{A}\|_{2,1}. \tag{3.1}$$

The term $\|\cdot\|_F^2$ represents the Frobenius norm defined as $\|\mathbf{A}\|_F^2 = \sum_{i,j} |\mathbf{A}_{i,j}|^2$. The term $\|\cdot\|_{2,1}$ is the $\ell_{2,1}$ norm, defined as $\|\mathbf{A}\|_{2,1} = \sum_i (\sum_j |\mathbf{A}_{i,j}|^2)^{1/2}$, which computes the sum of the Euclidean norms of the columns of the matrix. In Equation 3.1, the first term represents the reconstruction error, and the second term the total variation, where $\lambda > 0$ is used to control the relative importance among the two terms. The exact method used to perform the optimization is beyond the scope of our work, so readers are referred to the description by Song et al. [29] if additional details are required.

After performing the optimization and obtaining an optimal $\mathbf{A}$, we can compute a score for each frame $z_{f_i} = \|\mathbf{A}_{i,\cdot}\|_2$. By selecting the top-k highest scoring frames as split points, we are able to obtain a segmentation S_V consisting of $k+1$ segments. We elected to target obtaining segments with an average length of 5 seconds, resulting in $k = \lfloor \mathit{frames}(V)/(5 \cdot \mathit{fps}(V)) \rfloor$.

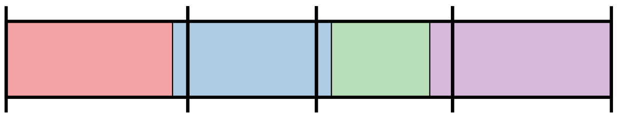


Figure 3.7: An example of change-point detection applied to a video. The ground-truth scenes of the underlying video are indicated by blocks of different colors. This method is able to successfully locate all the underlying scenes of the video, with a high degree of accuracy for the boundary locations compared to the ground-truth data.

Overall, this method can be thought of as a more robust version of threshold-based and content-aware sampling. Rather than relying simply on local color or brightness features, a combination of color and edge histograms are used to locate segment boundaries based on the statistical properties of the entire video.

3.3 Segment Post-Processing

After the initial segment selection has finished, we are left with a list of non-overlapping segments with no gaps between them, that is,

$$S_V = \{s_0, \ldots, s_k\} \qquad \text{with} \qquad \mathit{end}(s_i) + 1 = \mathit{start}(s_{i+1}). \tag{3.2}$$

The next step makes use of the frame labels computed in Section 3.1.1 to post-process each segment and eliminate undesirable frames and segments, effectively converting segments in the form of Equation 3.2 into their final form. Our goal with this step is to make use of the simple visual features currently available to us to eliminate segments which are obviously undesirable. Any frames or segments that can be eliminated now will result in large computational savings in later steps of our system, as we will no longer have to extract the higher-level, more computationally complex features.

Specifically, this post-processing consists of four steps:

Segment splitting The labels of the frames within each segment are examined, and based on a metric, segments are optionally split into multiple new segments.

Segment trimming For the case of segments which contain labeled frames on either end, we remove these frames from the segment.

Low-Quality segment elimination The fraction of labeled frames is computed for each segment, and any segments with a percent exceeding some threshold value are removed.

Short segment merging and elimination For any segment less than some threshold duration, we examine its neighbor segments, and either merge it with its neighbors if the distance between them is below some threshold, or eliminate it otherwise.

In the remainder of this section, we provide details of each step in this process.

3.3.1 Segment Splitting

In this step, we use a sliding window approach to split segments at any point right before a window in which all frames are labeled. This means that we use a window starting at the frame after the current one. The size of the window is selected to be half the number of frames in an optimal segment. We stop once our window extends past the frame range for the current segment. We additionally take into account the label of the current frame, only spitting if we are not currently on a labeled frame.

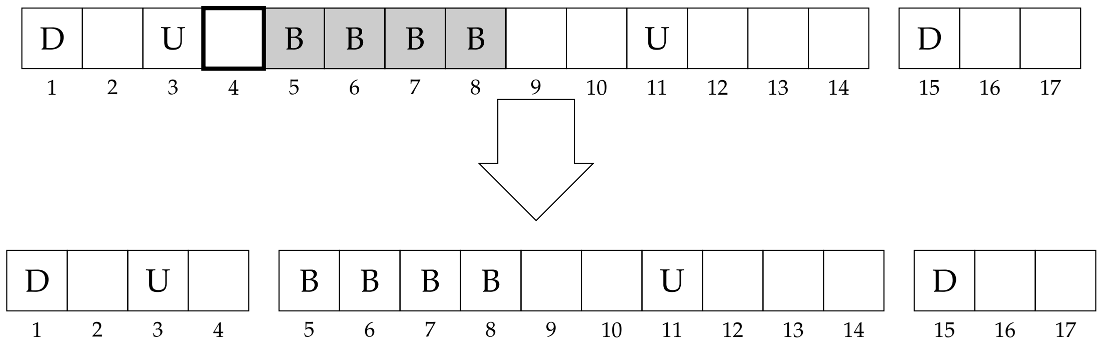


Figure 3.8: A visualization of the segment splitting step with a window size of 4 frames. The top half of the figure shows our video before splitting is applied. Frames highlighted in gray indicate ones which are covered by our current window, and the frame with a thick border represents the current frame we are examining. This frame and window have been specifically selected as they represent an instance where splitting should actually be performed. The resulting state after splitting can be found at the bottom of the figure. Specifically, we can see that the first video segment has been split into two segments just before the segment with four blurry frames in a row.

An example of a segment before and after splitting can be found in Figure 3.8. It is important to note that this step only splits segments and makes no attempt to remove individual frames—they will be eliminated by subsequent steps of post-processing.

3.3.2 Segment Trimming

Since segment splitting never actually removes any frames, we are often still left with a number of segments containing labeled frames at the beginning/end. The process of segment trimming operates by simply looping over every segment, and removing any frames at the beginning or end of that segment which are labeled. Once this step has finished, the first and last frame of every segment will *not* have a label. A visual summary of the operations and end result performed by this method on the results of the previous step can be found in Figure 3.9.

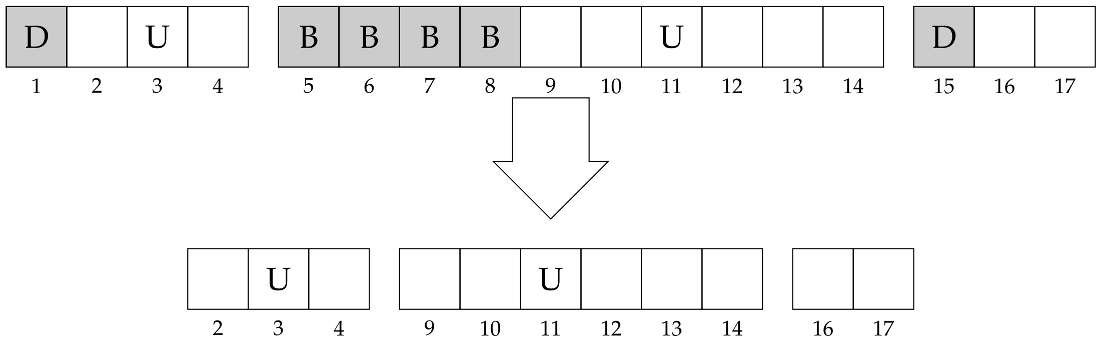


Figure 3.9: A visualization of the segment trimming step applied to the segments which resulted from the segment splitting step. In the top of the figure, labeled frames at the beginning or end of each segment have been highlighted. The bottom of the figure shows the result after these highlighted frames have been removed from their respective segments. For any non-labelled frames, frame numbers have been added to assist with future steps.

3.3.3 Low-Quality Segment Elimination

Once segments have been split, we eliminate any segments which consist mostly of labeled frames. To do this, we simply loop over every segment, calculate the percentage of frames which are labeled, and compare it to some threshold value. If the percentage exceeds this threshold value, we remove the segment.

The primary benefit of this step over previous steps is that it operates globally rather than locally, examining the segment as a whole rather than a small subset. A visual summary of this method applied to the results of the previous step can be found in Figure 3.10.

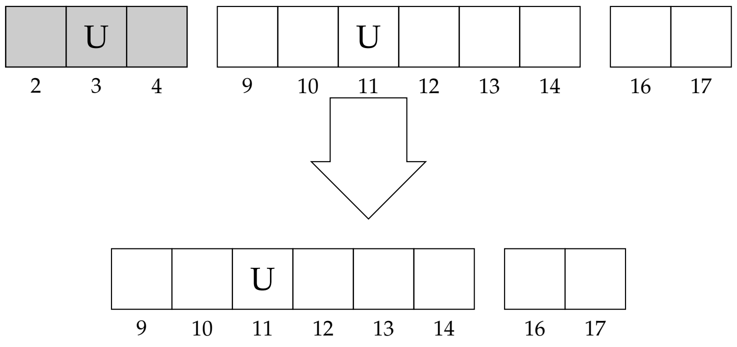


Figure 3.10: A visualization of the segment elimination step can be found above. In this case, we have used 30% as the threshold value for the percentage of labeled frames needed before a segment is eliminated. There is one segment for which this is true, highlighted in the top of the diagram. The bottom of the figure shows the resulting segments after this segment has been eliminated.

3.3.4 Short Segment Merging and Elimination

After the previous steps have been performed, we may still be left with segments that have a small duration with a majority of frames which are of high quality (not labeled). These segments can not be included in the final segmentation, as their later inclusion in a summary will result in abrupt “jumps” between scenes, which are often undesirable to end-users. However, the fact that they contain mostly high-quality frames means that they should only be removed as a last resort. To maximize our use of these segments, we attempt to merge them with neighboring segments, only removing them if no merge is possible.

In this step, we examine each segment with a duration below some threshold duration d_m, then examine each of its neighbor segments. If the distance between two segments is below some threshold value d_b, we merge the two segments. Note that a segment can be merged with both its previous and next neighbors if the distance between both falls below the threshold value. If the segment can not be merged with either of its neighbors, it is instead removed.

Overall, for a segment s with a previous neighbor s_p and next neighbor s_n, there are four possible outcomes:

1. If both the distance between s_p and s falls below the threshold value, and the distance between s_n and s falls below the threshold value as well, both s_p and s_n are removed, and the range of s is extended to range from the beginning of s_p and end of s_n.
2. If just the distance between s_p and s is below the threshold value, s_p is removed and the range of s is extended to start at the first frame of s_p.
3. If just the distance between s_n and s is below the threshold value, s_n is removed and the range of s is extended to end at the last frame of s_n.
4. Otherwise, s is removed.

Pseudocode of this can be found in Algorithm 2, and the result of this process applied to the results of the previous step can be seen in Figure 3.11.

3.4 Final Result

We started this chapter with an example video consisting of 17 frames, and worked through each step of our video segmentation system to arrive at a final result consisting of a single 9 frame segment, as seen in Figure 3.12. It is important to note the significant reduction of almost 50% in the number of frames before and after

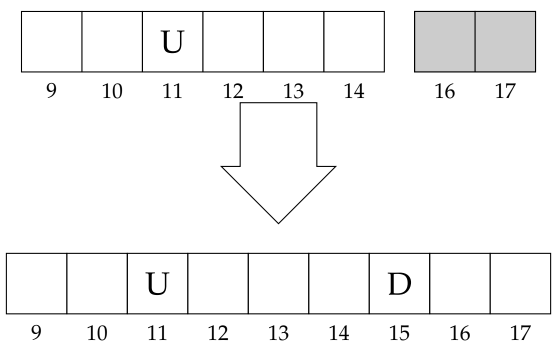


Figure 3.11: A visualization showing how segment merging is performed. In this case, there is a single segment with a duration below the threshold duration, highlighted in the top of diagram. The distance between this segment and its previous neighbor is less than the threshold so the two segments have been merged. The final segmentation state is shown in the bottom of the figure.

performing segmentation. Although the reduction amount is most likely less in practice, we can easily see the benefit of our post-processing. This, combined with our initial segmentation using multiple change-point detection gives us a distinct performance advantage compared to other methods at this early step in our video summarization system.

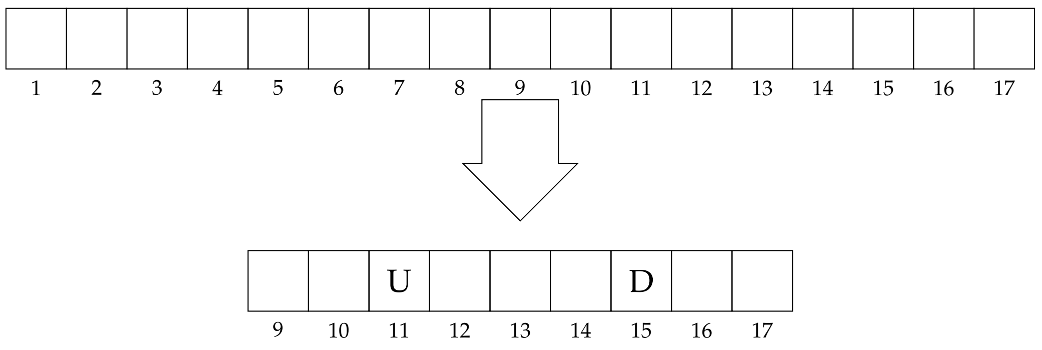


Figure 3.12: A visualization of the result of our entire video segmentation process applied to an example video. Performing an initial segmentation along with post-processing using very low-level features allows us to go from a 17 frame long video segment, to a single 9 frame long segment.

Algorithm 2 The algorithm used for performing segment merging and elimination.

```
Inputs:
   $S$: A segmentation consisting of $n$ segments $\{s_0, \ldots, s_{n-1}\}$
   $d_m$: The minimum segment frame duration threshold
   $d_b$: The between segment frame duration threshold
1: function PostProcessShortSegments($S$, $d_m$, $d_b$)
2:     for $s_p, s, s_n$ in Zip($S, S[1:], S[2:]$) do
3:         if $frames(s) > d_m$ then
4:             continue
5:         end if
6:         merged $\leftarrow$ False
7:         if $distance(s_p, s) \leqslant d_b$ then
8:             $S \leftarrow$ Remove($S, s_p$)
9:             $start(s) \leftarrow start(s_p)$
10:            merged $\leftarrow$ True
11:        end if
12:        if $distance(s, s_n) \leqslant d_b$ then
13:            $S \leftarrow$ Remove($S, s_n$)
14:            $end(s) \leftarrow end(s_n)$
15:            merged $\leftarrow$ True
16:        end if
17:        if merged = False then
18:            $S \leftarrow$ Remove($S, s$)
19:        end if
20:    end for
21:    return $S$
22: end function
Output: A new version of $S$ with segment merging and elimination applied
```

CHAPTER 4

Feature Extraction

This chapter provides an overview of the methods we use to extract features from our videos, which are later used for scoring segments. At the frame-level, we extract both low-level features, which are time-efficient to compute, and higher-level features, which are more expensive to compute, but have the potential to provide a deeper degree of scene understanding. Once these frame-level features are computed, we additionally compute per-segment features as aggregations of these features.

Once all candidate segments S_V for our video V have been located, the next step is to extract a number of features for these segments. In our work, we extract features at varying levels of detail. In particular, we extract a large number of low-level visual features which have previously been used in computational aesthetics to score the attractiveness or "beauty" of an image. These features are efficient to compute, but are limited in the amount of scene understanding they are able to achieve. Additionally, we compute features using more complicated computer vision methods—in particular face detection and face recognition. These higher-level features are extracted with the intention of gaining a deep understanding of the content of a video. They are expensive to compute, but we hypothesize that their benefits as additional features will outweigh this expense when applied to the task of segment scoring. We start by extracting frame-level features, and conclude by performing an aggregation over these features for each segment in order to obtain a single feature vector X_s for each segment $s \in S_V$.

4.1 Frame Features

Initially, for each frame f in each segment s of our segmentation S_V, we extract a feature vector X_f. In particular, we extract a combination of low-level features related to computational aesthetics, and higher-level features related to face detection and recognition.

4.1.1 Low-level Features

Low-level features have the benefit of being computationally efficient to compute, with their primary drawback being that they are unable to capture higher-level concepts, such as the presence of a specific person in an image. We currently extract a number of hand-crafted features related to computational aesthetics, which over the years, have shown to be useful for the task of scoring the attractiveness of an image[9, 10, 11, 37]. Overall, we compute 59 different feature values for each frame, leaving us with a set of low-level feature vectors X_f^{59}. A summary of the features we extract, along with their dimensionality and a short description of each can be found in Table 4.1.

Feature	Dim.	Description
Contrast	1	The ratio between the luminance range and average luminance.
Image Mean HSV	3	The average H, S, and V values over the entire image.
Center Mean HSV	3	The average H, S, and V values for the image center quadrant.
Itten Histograms[11]	20	Histograms of H values over 12 bins, S values over 5 bins, and V values over 3 bins.
Itten Contrasts[11]	3	Standard deviation of each Itten Histogram.
Pleasure, Arousal, Dominance[11]	3	Approximate emotional values computed as linear combinations of the mean V and S values.
Haralick Texture Features[12]	13	Average Haralick texture features over all four directions.
Contrast Balance	1	Distance between the original and contrast-normalized grayscale image.
Exposure Quality	1	Negative absolute value of luminance histogram skew.
JPEG Quality[13]	1	No-reference quality estimation algorithm for JPEG images.
Tenengrad[14]	1	Sharpness according to the Tenengrad method.
Spectral Residual	9	Rule of thirds using spectral saliency[15] in 9 quadrants.

Table 4.1: A summary of the low-level features extracted from each frame. The dimensionality of each feature is provided, as well as a short description of what it computes. References have been provided whenever possible, in the case that more information is required for a specific feature.

4.1.2 High-level Features

The high-level features that we compute were selected with the intention of gaining a deeper understanding of the actual content of a video. Where low-level features are only able to compute general concepts such as the sharpness or relative brightness of single frames of a video, high-level features have the potential to provide us with information more akin to what a human may observe when viewing a video. For example, face detection and recognition features allow us to determine specifically *which* people are present in a frame—something that humans instinctively do when viewing a video.

For the vast majority of videos, knowledge of the people that are featured in the video and statistics related to their appearance are extremely important for proper understanding of the video content. For our case of raw user videos, this knowledge would allows us to determine which people appear the most and least in a video, and score the segments accordingly. We first perform face detection, which gives us the number of faces present in a frame, as well as the position of each face. Using this information, we then perform face recognition, where we perform clustering on every detected face, resulting in a list of specific people that appear in the video, and the frames they appear in.

The fact that we already extract a number of low-level image aesthetics features, combined with our access to the AVA dataset containing a large number of user image aesthetic rankings, additionally led us to the idea of extracting a general, high-level aesthetics feature for each frame. Since our primary concern is distinguishing between aesthetically desirable and undesirable frames, and since our per-frame features will be aggregated into per-segment features for our final summarization, we elected to train an XGBoost classification model. Frames classified as 0 represent aesthetically undesirable frames, while frames classified as 1 represent desirable ones.

4.1.2.1 Face Detection

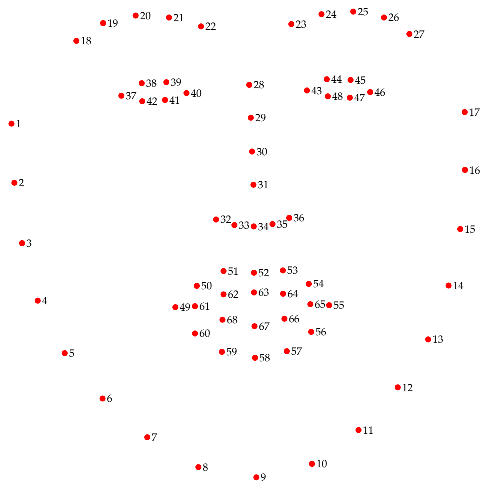
Face detection is vision process which takes as input a frame f, and returns a possibly empty list of bounding boxes which for the current frame, contain face boundaries. Many face detection methods have been developed over the years, but for our work, we make use of the one provided by the `dlib` library—a modern, high-performance C++ library which facilitates the use of high-quality deep-learning models. Specifically, `dlib` uses a Felzenszwalb's HOG (FHOG)[20] object detector pre-trained to detect faces. This FHOG detector uses a combination of image pyramid downsampling with a $5/6$ ratio, a 80×80 sliding window, and HOG features (previously discussed in Section 2.3.1) to perform object detection. The basic idea behind the FHOG detector is to detect objects by examining each sub-window of an image at a number of different scales, which effectively reduces object detection to a problem of binary classification. For each sub-window, we extract an arbitrary feature vector X^{31} as the concatenation of the HOG features for each cell. This is performed for a number of decreasing scales of the image, forming a HOG pyramid H. For each scale of H, we have a learned filter F which represents the concatenation of optimal HOG features to represent the target object at the given scale. Given X^{31} and F, we are able to compute the score of X^{31} simply as $F \cdot X^{31}$. This score is then summed over each scale in the pyramid H to obtain a final score. Finally, any sub-windows which have a score exceeding some threshold score γ are selected as probable locations for the target object.

The full implementation of FHOG used in our work goes even further by learning a parts-based model as a combination of the previously described model, initially limited to images in smaller scales of the pyramid. To achieve this, it takes advantage of the knowledge that a more detailed version of the current image exists in order to compute sub-filters over parts of the current sub-window at the same scale. The aggregate score of these parts then results in a significantly more informative scoring of each window.

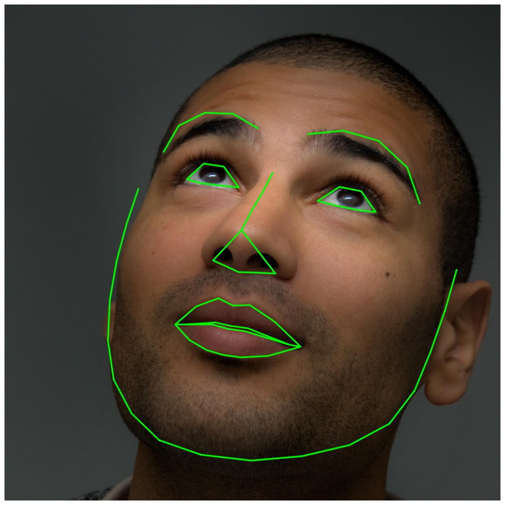
The final result of this step is that for each frame f, we have a possibly empty list of m face bounding boxes $B_f = \{b_0, \ldots, b_m\}$.

4.1.2.2 Face Recognition

For each bounding box b, we extract a feature vector X^{128} using the default face recognition model included with `dlib`. Then, we perform face pose estimation using the default face shape predictor[38] from `dlib`. This predictor was trained using the iBUG 300-W[39] face landmark dataset. This predictor takes as input a bounding box containing a face, and computes the position of 68 important "facial landmarks", shown in Figure 4.1.a. Furthermore, these facial landmarks are aligned with the underlying face image, giving a final result similar to the one presented in Figure 4.1.b.



(a) A visual display of the base landmark positions for a predictor using 68 landmark points.



(b) An example of a face template aligned to a face based on predicted facial landmark positions.

Figure 4.1: A visual demonstration of important concepts related to face detection and alignment. The first example in (a) shows the general positioning of important landmarks for an arbitrary face, while the second example in (b) provides an example of how these landmark positions can be aligned to a face with an arbitrary orientation.

Once these landmarks have been computed, we are able to perform the first half of face recognition, where for a bounding box and set of landmark features, we extract a feature vector X^{128} representing the embedding of the face in a pre-learned metric vector space. In this space, faces which are very similar (from the same person) will have a very small distance between them, and faces which are different (representing different people) will have a large distance between them. These features are computed using the pre-trained deep metric model for faces included in `dlib`.

Specifically, the model used is a version of the deep neural network ResNet-34[40], modified to contain only 29 layers, and half the number of filters per layer. The network is also modified to use a metric loss function in order to learn a metric space for faces. Training is performed using over 3 million faces from the FaceScrub dataset[41] and the VGG-Face dataset[42], and when tested on the Labeled Faces in the Wild (LFW)[43] dataset, it is able to predict with 99.38% accuracy if two images are of the same person.

To perform actual face recognition, we perform clustering using every face feature vector X^{128} over all frames for every segment of the video. Specifically, we use the Chinese whispers graph clustering algorithm[44] to compute an optimal clustering of faces into "people", with no prior knowledge about the number of people being clustered. Chinese whispers is a linear-time hard partitioning, randomized, flat clustering method. This means that although the result can change between iterations and there is no hierarchical information between clusters, we get a final clustering which (1) assigns each face to only a single segment, and (2) can be efficiently computed for all *feasible* durations of input video. A video with a duration of 1 hour can easily contain > 50000 nodes, so a linear-time algorithm is extremely desirable.

The initial "graph" used as input to Chinese whispers is constructed by simply looping over every pair of features $\left\{ \left(X^{128}_{f_a}, X^{128}_{f_b}\right) \mid f_a, f_b \in \mathit{frames}(s), f_a \neq f_b \right\}$ across all segments and frames computed in the previous step, and creating an "edge" between two nodes when their distance is below some threshold value τ. A value of $\tau = 0.6$ was selected, as it matches the value that was used for the metric loss layer of the deep neural network used in the previous step. Chinese whispers is then run using this graph, resulting in for every frame f, a set of recognized faces $\{\ell \mid \ell \in \mathbb{Z}\}$.

4.1.2.3 Aesthetics Model

As mentioned previously, in addition to using low-level aesthetics features as input features themselves, we leverage the AVA dataset to train a general model for computing a single score for a given image. Every image in the AVA dataset has an average user score in the range $[0.0, 1.0]$. Similar to previous models for image aesthetics[16][28], we classify an image as 0 if the average score is below 0.5, and 1 otherwise. We train an XGBoost classification model using 10-fold cross validation and a train-test split of 70%/30%. Our model obtains an accuracy of 73.66%. This is significantly higher than the reference model[28], which obtains an accuracy of 53.85%, and slightly lower than the modern ILGnet[16] deep learning model, which obtains an accuracy of 82.66%. For the significant speed increase our model has over ILGnet, this accuracy is acceptable for our purposes.

4.1.3 Final Feature Vector

For our final feature vector, we concatenate all our low-level and high-level features. We start with our low-level feature vector X_f^{59}. We concatenate the number of faces present in the frame to get a new feature vector X_f^{60}. For the face recognition feature, we first filter out any recognized faces which only appear in a single frame, as these represent unimportant faces. We then concatenate the number of recognized faces present in the frame, to get a new feature vector X_f^{61}. Finally, we append our aesthetics model value, giving us a final frame feature vector X_f^{62}.

4.2 Segment Features

Now that we have for each frame f a feature vector X_f^{62}, we perform an aggregation over all frame features in each segment to obtain for each segment s, a feature vector X_s. For each feature value $\{x^0, \dots, x^{61}\} \in X^{62}$, we compute the mean and standard deviation across all frames in each segment. We then have as a final feature vector

$$X_s^{124} = \bigcup_{i=0}^{61} \left\{ \text{mean}\left(\left\{x_f^i \mid f \in s\right\}\right), \text{std}\left(\left\{x_f^i \mid f \in s\right\}\right)\right\} \tag{4.1}$$

for each segment $s \in S_V$.

CHAPTER 5

Video Segment Scoring

This chapter presents the methods which we developed for scoring a segment based on it's quality using the previously computed segment features. We start by describing the candidate models we selected for computing scores, discussing the advantages and limitations of each. We go on to describe our methods for training and testing possible models, and provide a short analysis of the initial training results. We conclude by performing an analysis of the importance of each feature in our selected model.

In this step, we operate on the segment features $\left\{X_s^{124} \mid s \in S_V\right\}$ computed in the previous step, and compute a set of segment scores $\{Q_s \mid s \in S_V\}$. In our work, we evaluate models using three tree-based machine learning methods: (1) decision trees, (2) random forests, and (3) XGBoost. We start by giving an overview of these models, describing their primary method of operation, their advantages against other methods, as well as their limitations.

We move on to describe our method for training and evaluating each model. We then discuss the results, and select one, or possibly two models to use for segment scoring, providing justification for our choice. Using out best model, we extract features importances and analyze them in order to possibly eliminate features which do not contribute significantly to the final accuracy of our models, thereby increasing the performance of our system without sacrificing accuracy.

5.1 Candidate Machine Learning Models

For scoring segments, we examine a number of candidate models. Due the structure of our input feature vector—a concatenation of various feature values covering different topics—we mostly focus on methods which require minimal knowledge and pre-processing of the input data. In particular, we focus on three tree-based models:

1. Decision Trees
2. Random Forests
3. XGBoost

5.1.1 Decision Tree Learning

Decision tree learning makes use of a decision tree, previously described in Section 2.3.2, to perform classification or regression using our input features as the attributes during training. In our work, we use SKlearn for training decision trees and performing classification or regression, which uses an optimized version of the Classification And Regression Trees (CART) algorithm[45]. This is a greedy recursive method which at each step, finds the combination of attribute and threshold value which maximizes the information gain (previously described in Section 2.3.3). The general operation of the algorithm is fairly simple—a rule involving one of the attributes which maximizes the information gain is selected, the current node is split into two new nodes, and the same process is applied to each of these new nodes. The algorithm terminates when either: (1) it is determined that no additional information gain is possible, or (2) when some pre-set stopping condition such as a maximum depth is met. The result is a decision tree where each branch ends in a leaf node consisting of a single class, which can be traced backwards to the root to obtain the unique set of rules that define it.

Advantages

Decision trees are among the simplest of tree-based models, and hence have an easy-to-understand internal structure. Their primary advantages over non-tree models are as follows:

- They are easy to interpret and visualize.
- Their inner workings can be easily observed and recorded, facilitating reproducible research.
- They can handle numerical and categorical data without pre-processing.

- They tend to have moderately high performance on large datasets, both in terms of speed and accuracy.

Limitations

The fact that decision trees are among the simplest of tree-based models, however, also results in a few significant limitations:

- As a result of the greedy model commonly used to decide the optimal decision at each node, they are only able to perform local rather than global optimizations.
- They are prone to overfitting in the case of deep trees, and suffer from decreased accuracy for shallower trees when compared to more complex tree-based models.

5.1.2 Random Forest

Random forests are an ensemble learning method which combines bootstrap aggregating[46] (“bagging”) and decision trees. They operates by first generating multiple decision trees by continually resampling the training data with replacement, then performing either voting or averaging over the output of each tree to obtain the final classification or regression value respectively. One of their major benefits is that they offer significantly less overfitting when compared to a single decision tree. They effectively combine many low-bias, high-variance models to achieve a low final error.

The algorithm used for training is fairly simple—basically general bootstrap aggregating combined with a decision tree learning algorithm modified to use attribute bagging[47]. For a training set $X = x_1, \ldots, x_n$ with outputs $Y = y_1, \ldots, y_n$, bagging operates by constructing a forest F of decision trees f_b for $b \in [1, B]$, where B is a hyper-parameter. In our work, we perform grid search with cross-validation to select an optimal value of B. For each value $b \in B$, we randomly sample, with replacement, B examples from our data X and Y, resulting in a new set of data X_b and Y_b. This new set of data is used to train a decision tree f_b. Note that B determines both the number of trees that are constructed, and the size of the set of examples used to train them. Once we have our trained forest F, we can compute the output y' for an unseen input feature vector x' as either

$$y' = \text{mean}(\{f_b(x') \mid b \in B\}) \qquad \text{or} \qquad y' = \text{mode}(\{f_b(x') \mid b \in B\})$$

for regression or classification respectively.

As mentioned previously, random forests make use of a modified decision tree learning algorithm which incorporates feature bagging. At each decision node, rather than evaluating all features, a random subset of p features is instead selected. This is done in an attempt to avoid the cross-tree correlation that may occur during ordinary bootstrap aggregation if a small number of features are strong predictors for the output value. If this was the case, these features would be selected by the vast majority of the B trees, resulting in a strong correlation among the trees—an extremely undesirable property. For classification, $p = \left\lfloor \sqrt{|x|} \right\rfloor$ is typically used, while for regression, $p = \max(\min(|x|, 5), |x|/3)$ is recommended.

Advantages

The major advantages of random forests over decision trees lies in the fact that they make use of multiple trees, while decision trees use only a single tree. In particular:

- By averaging over several trees, the risk of overfitting is significantly reduced.
- They are able to achieve reduced variance and bias compared to decision trees, therefore generally resulting in an increase in accuracy.

Limitations

The use of multiple trees, however, also comes with some costs:

- They are harder to visualize and understand.
- They are more computationally expensive.

5.1.3 XGBoost

XGBoost[48] is a machine learning gradient boosting method developed in the last few years which has recently enjoyed a large amount of attention due to its use in the winning models of many machine learning competitions. The model structure consists of a number of weak prediction models, such as decision trees, which are built in a stage-wise fashion. In this respect, XGBoost is similar to random forests, the primary difference lying in the types of decision trees they build.

Where random forests generate full decision trees with a low bias and high variance, XGBoost instead builds a number of very shallow decision trees (called "weak learners"), each with a high bias and low variance. The very first tree is simply a very shallow tree, which by itself, has bad performance. It then builds another shallow tree which is trained to predict what the first tree missed. This process continues, iteratively creating additional weak learners, until some stopping condition is reached—traditionally the number of trees to build. The final model is then formed as an ensemble of these weak learners.

Advantages

The primary advantages of XGBoost over other tree-based models are a result of its use of many weak learners. We have that:

- They generally outperform random forests with optimal parameters.
- They achieve reduced overfitting compared to random forests and decision trees.

Limitations

The use of weak learners, combined with the many possible hyper-parameters available for tuning, lead to a few limitations. In particular:

- They often require fine-tuning to achieve optimal performance.
- They are very difficult to visualize and understand.

5.2 Model Training

Initially, we trained a segment-level interestingness model for each of the three base models—decision trees, random forests, and XGBoost—using the default parameters. The models are evaluated using features extracted from uniform 5 second segments across all videos in the SumMe and TVSUM50 datasets. Train-test splits are generated using 10-fold cross-validation on shuffled data, and the mean-squared-error is used as the error metric for evaluating each model. The results for each model are presented in Table 5.1.

Model	**Min**	**Max**	**Mean**	**Std. Dev.**
Decision Tree	0.04005	0.05145	0.04559	0.00380
Random Forest	0.02302	0.03025	0.02673	0.00238
XGBoost	0.02244	0.02907	0.02537	0.00214

Table 5.1: Metrics for the mean-square-error of each of our three base models evaluated using 10-fold cross validation. We can see that of the three models, XGBoost has the best performance, with the random forest model performing slightly worse, and the decision tree significantly worse.

As we can see from Table 5.1, both the XGBoost and random forest models obtain very similar error rates, with XGBoost slightly out-performing the random forest model, and both significantly out-performing the decision tree model. For this reason, we will use both XGBoost and random forest models for evaluating our system. It is important to note that at this point in the system, we elected not to perform model fine-tuning, instead waiting until Chapter 7, where we discuss an error metric designed specifically for use with video summaries.

5.3 Feature Importance

An additional benefit of using tree-based models such as decision trees and XGBoost is that the underlying models they make use of effectively compute the "importance" of each feature as a side effect of their learning process. Averaging over all folds for a given model, we can obtain for each feature an "importance" value, which we normalize across all features to be in the range $[0.0, 1.0]$. For a given feature, a value of 1 means that the feature is very important to the model, while a value of 0 means that the feature is effectively useless to the model.

Since we previously saw that the XGBoost model obtained the best performance among the models we tested, we compute the feature importances by averaging over all the folds of our XGBoost model, yielding the results presented in Figure 5.1.

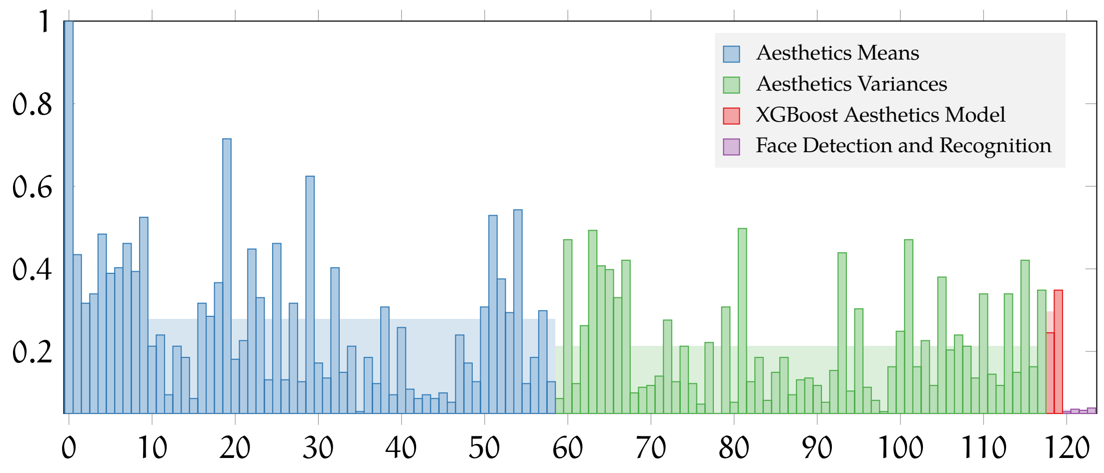


Figure 5.1: A plot of feature importances for each feature included in our final feature vector. For the purpose of visualization, we have grouped the features into four major groups, each represented by its own color: blue represents the mean values of each aesthetic feature, green the variances of each aesthetic feature, red the mean and variance of our XGBoost aesthetics model values, and finally purple the mean and variance values for our face detection and face recognition features. The background of each group additionally contains an aggregate bar which shows the average importance across the entire group.

One important conclusion we can draw from Figure 5.1 is that among all the features used by our model, those involving face detection and face recognition have minimal importance, meaning they do not contribute a relevant amount of information to the model. This is of particular importance to us, as these features are computationally expensive to compute compared to the other features we use, and our initial hypothesis was that the computational cost of these features would be offset by their actual importance when computing a segment score. Figure 5.1 shows that this is obviously not the case, and thus resulted in us deciding to exclude face detection and recognition features from our final feature vector. Therefore, we remove the last four features from our segment features, giving us the new set of segment features $\{X_s^{120} \mid s \in S_V\}$.

Another relevant observation is related to the features obtained from our XGBoost aesthetics model. Looking at Figure 5.1, we can see that these features have a relatively high importance. In particular, their combined average importance is the highest among the major groups of features. This is important, as it means that we were able to train a supervised model for individual image aesthetics using frame-level data from the AVA dataset, and successfully apply it to the task of segment scoring. In particular, this is testimony in favour of the possibility of combining aggregation and low-level data to obtain relevant future predictions of higher-level values.

CHAPTER 6

Video Summarization

In this chapter, we describe our method for generating a summary of a video using the list of scored segments computed previously. We start by describing the basic method we use for performing an initial segment selection. We go on to describe the method we developed for incorporating user-preference data into our summarizations, as well as our method for generating additional summaries in an unsupervised manner. We finish up by describing possible future work that could be incorporated into our summarization methods.

Once all segments have been scored, we are able to perform the final video summarization step. In this step, we operate on the segment scores $\{Q_s \mid s \in S_V\}$ computed previously, and generate a final summary U_V for our video. We start by describing how segments are initially selected to obtain a final summarization. When describing this, we also discuss our method of incorporating user preference information into the segment selection process.

In the case that a user is unsatisfied with a generated summary, we also support generating additional summaries, either in a supervised or unsupervised manner. In the supervised case, a user provides either a 0 or 1 score for one or more segments, and these scores are taken into account when generating the new summary. Specifically, we highlight the intuitive and easy-to-use interface provided by the Cliply system to facilitate this. We also support an unsupervised variant, where additional summaries can be generated with no user input. In the case that a user is or isn't satisfied with a summary, this provides us with insight into their individual preferences towards different segments. We therefore also describe how we make use of *this* information to over time, learn a user-specific model for segment interestingness.

Finally, we discuss the possible future work that could be incorporated into our summarization methods. Keeping in mind our goal of creating summaries at real-time on commodity hardware (an average user's machine), we focus on methods that still have relatively small computational costs.

6.1 Initial Segment Selection

Our initial segment selection method is based on formulating summary generation as a knapsack problem, specifically a $0/1$ knapsack problem. Given a set of items (segments) $s \in S_V$, each with a weight (duration) $\textit{frames}(s)$ and a value (score) Q_s, we determine which segments to include in our final summary such that the final length is less than or equal to our target summary duration, and the sum of segment scores is maximized. We initially present the unsupervised variant of our algorithm, used when no user-preference data is available, that is, we are dealing with a new user. We go on to describe how we can modify our algorithm to be supervised in order to use it with a user which has previously used our system, and therefore has individual user-preference data available.

6.1.1 0/1 Knapsack

Our primary goal for video summarization is simply that given a list of segments $s \in S_V$ with scores Q_s, select a set of segments $U_V \subseteq S_V$ which maximize the sum of scores across all segments and have a total duration less than or equal to some target duration W. This effectively formulates our summarization as an 0/1 knapsack problem, which for our specific case, can be expressed as

$$\operatorname*{arg\,max}_{U \subseteq S_V} \sum_{s \in U} Q_s \qquad \text{subject to:} \qquad \sum_{s \in U} \mathit{frames}(s) \leqslant W.$$

In our work, we use of a simple dynamic programming solution[26]. If we define T as an $n \times W$ array, and $T(i, w)$ as the maximum score that can be obtained with a duration up to or less than w using the first i items of $S_V = \{s_0, \ldots, s_{n-1}\}$, we have the recursive definition:

$$T(0, w) = 0$$

$$T(i, w) = \begin{cases} T(i-1, w) & \text{if } \textit{frames}(s_i) > w \\ \max(T(i-1, w), T(i-1, w - \textit{frames}(s_i)) + Q_{s_i}) & \text{if } \textit{frames}(s_i) \leqslant w. \end{cases}$$

The solution can be found by computing the value of $T(n, W)$, a process which we describe in Algorithm 3.

This algorithm serves as the baseline for our unsupervised method, where we simply run the algorithm using our segments and scores as inputs, and as the output, get a final summary.

6.2 Additional Summaries

Although it would be optimal if all users were satisfied with our initial summary, this is often not the case. In the case that a user is unsatisfied with a summary U, we support generating an additional summary U' almost instantly, as we only need to adjust segment scores and generate a new summary using our 0/1 knapsack algorithm. We additionally support a supervised variant of generating additional summaries, where we allow users to provide either a 0 or 1 score of one or more segments, and incorporate these preferences when generating the new summary.

A user generating an additional summary is also an opportunity for us to learn their preferences over time. Specifically, for a user u, we maintain a history of any per-user scores we obtain for specific segments Q^u.

6.2.1 Unsupervised Additional Summary Generation

When a user is unsatisfied with a generated summary, and does not provide any input as to their preference toward the inclusion of any segments in the next summary, we consider this to be the unsupervised case of additional summary generation. In this case, we simply scale the score of all segments from the previous summary down by some bias value $\gamma \in [0.0, 1.0]$, and generate a new summary. In our work, we used a γ value of 0.1. This scaling process is repeated until a summary is generated containing different segments than the original, or the score of all segments reaches a maximum. This algorithm is presented in Algorithm 4.

Although we do not specifically know what the user disliked about the original summary U, we are still able to partially learn user preference data from them with the help of U'. In essence, we assume that the user disliked all segments which were removed from U to create U'. That is, for a specific user u and segments $s \in U - U'$, we remember the new, reduced segment score $Q_s^u = Q_s$ in their history.

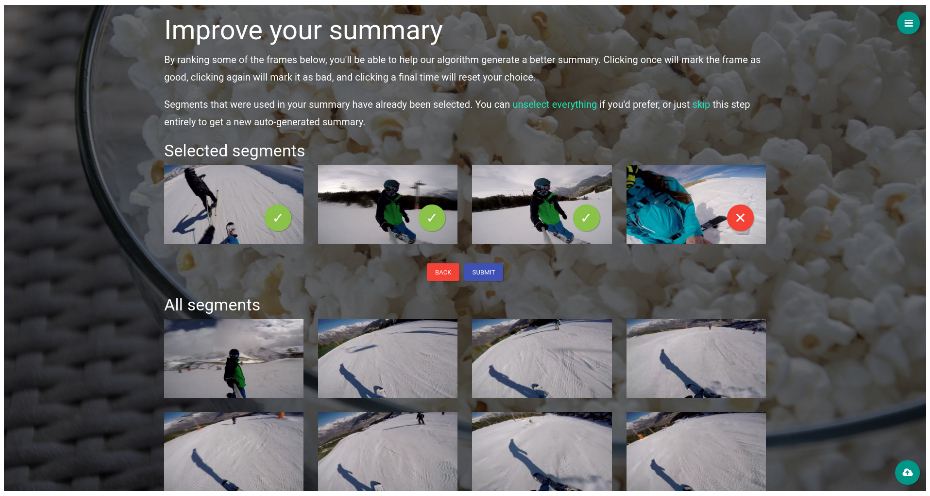
6.2.2 Supervised Additional Summary Generation

In the case that a user provides segment preference input, we consider this to be the supervised case of additional summary generation. An initial summary is first generated using all segments that the user indicated they want to remain in the final summary. Any segments which the user does not want included have their score set to 0, ensuring that they will not get selected for future summaries until all other choices have been exhausted. Finally, the unsupervised generation method is used to select segments to make up the remaining time needed for the final summary. This algorithm is presented in Algorithm 5.

Algorithm 5 Algorithm for generating an additional summary in the supervised case.

```
   Inputs:
     $S$: A segmentation.
     $Q$: A set of scores for each segment.
     $W$: A target frame count for the final summary.
     $U$: The previously generated summary for the segmentation $S$.
     $S^0$: A set of segments $S^0 \subseteq S$ the user wants to discard.
     $S^1$: A set of segments $S^1 \subseteq S$ the user wants to keep.
1: function AdditionalSummarySupervised($S, Q, W, U, S^0, S^1$)
2:     $U' \leftarrow S^1$
3:     for $s \in S^0$ do
4:         $Q_s \leftarrow 0$
5:     end for
6:     $U' \leftarrow U' \cup$ AdditionalSummary($S, Q, W, U - U'$)
7:     return $U'$
8: end function
   Output: A new summary $U' \subseteq S, U' \neq U$
```

This is perhaps the process that benefits the most from the Cliply system, as it provides an intuitive user interface to allow users to categorize segments, as shown in Figure 6.1. Using this interface, users are provided with a representative frame for each segment, and are able to alter the score they attribute to segments simply by clicking on each representative frame. The current categorization of segments is displayed at the top of the page, and when users are satisfied with their selection, they can simply submit it and have a new supervised summary generated.


Improve your summary

By ranking some of the frames below, you'll be able to help our algorithm generate a better summary. Clicking once will mark the frame as good, clicking again will mark it as bad, and clicking a final time will reset your choice.

Segments that were used in your summary have already been selected. You can unselect everything if you'd prefer, or just skip this step entirely to get a new auto-generated summary.

Selected segments

All segments

Figure 6.1: An example of the user interface provided by Cliply to assist users in scoring segments. In this case, we can see that the user has categorized three segments as 1, indicated by the green check mark button in the bottom right of the first three segment images. We also see that they have categorized the fourth segment as 0, indicated by the red cross button. At this point, the user has the option of scrolling down to assign a category to more segments, or to submit their current selection and generate a new supervised summary.

Unlike the unsupervised case, a user u disliking a summary U in this case provides us with significant insight into their personal preferences, as we know exactly which segments they liked and disliked. For any segments $s \in S^0$ that the user disliked, we can add this to their scoring history as $Q_s^u = 0$. Additionally, for any segments $s \in S^1$ that the user explicitly wanted to keep, we add this to their scoring history as $Q_s^u = 1$. The final use of unsupervised summary generation takes care of updating their history with any remaining derived preference information.

6.3 User Preferences

As mentioned previously, for each user u of our system, we maintain a history of their segment scores as Q^u. Using these historical segment scores, we are actually able to over time, learn a model for each user which takes into account their specific segment preferences. For each user, we maintain a model similar to the one we use for segment scoring, but trained using their score history Q^u rather than the scores provided by our reference datasets. XGBoost supports continuing training of a previous model, so we are able to implement this by starting with an untrained model for each user, and fine-tuning the model whenever new data is added to their segment score history Q^u.

It is trivial to modify our system to incorporate a per-user model when available. Rather than using Q as the segment score input to any of our summarization functions (SelectSegments, AdditionalSummary, and AdditionalSummarySupervised), we simply modify Q to be an linear combination of the scores from Q and Q^u, where Q_s is used in place of Q_s^u if no history exists for a specific segment s. That is, we set

$$Q_s = \alpha Q_s + \beta Q_s^u,$$

where both α and β default to 0.5, resulting in an average of the two scores. We are then able to use any of the previously discussed summary generation methods, with the benefit that they now incorporate per-user preference data.

Algorithm 3 Dynamic programming algorithm for segment selection.

```
**Inputs:**
  $S$: A segmentation consisting of $n$ segments $\{s_0, \dots, s_{n-1}\}$.
  $Q$: A set of scores for each segment.
  $W$: A target frame count for the final summary.
1: **function** SelectSegments($S$, $Q$, $W$)
2:     $T = \textsc{Zeros}(n, W)$                    ▷ Construct a $n \times W$ array of zeroes
3:     **for** $j = 1, \dots, n$ **do**
4:         **for** $w = 0, \dots, W$ **do**
5:             **if** $\textit{frames}(s_j) > w$ **then**
6:                 $T[j, w] \leftarrow T[j-1, w]$
7:             **else**
8:                 $T[j, w] \leftarrow \max(T[j-1, w], T[j-1, w - \textit{frames}(s_j)] + Q_{s_j})$
9:             **end if**
10:        **end for**
11:    **end for**
12:    $U \leftarrow \textsc{List}(\ )$                    ▷ Construct an empty list
13:    $w \leftarrow W$
14:    **for** $j = n, \dots, 1$ **do**
15:        **if** $T[j, w] \neq T[j-1, w]$ **then**
16:            $U \leftarrow \textsc{Append}(U, s_{j-1})$        ▷ Append segment $s_{j-1}$ to $U$
17:            $w \leftarrow w - \textit{frames}(s_{j-1})$
18:        **end if**
19:    **end for**
20:    **return** $U$
21: **end function**
**Output:** A summary $U \subseteq S$
```

Algorithm 4 Algorithm for generating an additional summary in the unsupervised case.

```
    Inputs:
      $S$: A segmentation.
      $Q$: A set of scores for each segment.
      $W$: A target frame count for the final summary.
      $U$: The previously generated summary for the segmentation $S$.
 1: function AdditionalSummary($S$, $Q$, $W$, $U$)
 2:     $U' \leftarrow U$
 3:     while $U' = U$ do
 4:         for $s \in U$ do
 5:             $Q_s \leftarrow Q_s - \gamma \cdot Q_s$
 6:         end for
 7:         $U' \leftarrow$ SelectSegments($S, Q, W$)
 8:     end while
 9:     return $U'$
10: end function
    Output: A new summary $U' \subseteq S, U' \neq U$
```

CHAPTER 7

Evaluation and Results

In this chapter, we present our approach to evaluating and optimizing our video summarization method. Specifically, we provide error metrics for summaries generated using our system, compared to metrics over the reference summaries for each relevant dataset. We additionally perform fine-tuning on our XGBoost model to maximize the accuracy of our model. In conclusion, we present the final model we used, along with any options that were used for that model. Overall, this chapter presents our final model, along with the path we followed to arrive at this final model.

At this point in our system, we have from Chapter 5 a method for scoring the segments in a video, and from Chapter 6 a method for selecting a subset of these scored segments to form a final summary of the target video. In order to assess the quality of our generated segments, we introduce the pairwise F_1-measure, which is used to compare a generated summary to multiple user summaries. For the random forest and XGBoost models from Chapter 5, we perform grid search over various model parameters, and continue with the optimal parameters for each variable. In the end, we compare the final pairwise F_1-measure values for both the random forest and XGBoost models, and select the model which obtains the highest value.

7.1 Pairwise F_1-measure

Since our datasets contain multiple summaries from different users, we need a method to evaluate the performance of a summary against all of the user summaries. The standard method used for this is the pairwise F_1-measure, originally proposed in [24]. For a given generated summary U and set of user summaries $J = \{U^0, \dots, U^n\}$, we first compute for each user summary U^i in J the precision p_i and recall r_i as

$$p_i = \frac{|\,\mathit{frames}(U) \cap \mathit{frames}(U^i)|}{|\,\mathit{frames}(U^i)|} \qquad \text{and} \qquad r_i = \frac{|\,\mathit{frames}(U) \cap \mathit{frames}(U^i)|}{|\,\mathit{frames}(U)|}.$$

We are then able to compute the pairwise F_1-measure F_U of our summary as

$$F_U = \frac{1}{n+1} \sum_{i=0}^{n} 2 \cdot \frac{p_i r_i}{p_i + r_i}.$$

Better methods are represented by higher F_1-measure values.

7.2 Sampling Methods

When training our models, there are various ways we can divide our data into a train and test split. Since for the SumMe dataset we would like to test our model on every video to compare to current methods, we need to use a sampling method better than the traditional split of a random shuffle of all the segments. Overall, we use four different sampling methods, each using a different subset of all the available data for training the models. Along with a short description of each, we provide a figure showing an example of the train and test splits that would be used for an example video.

It is important to note that as a result of our method being computationally efficient and able to operate in real-time, and the minimal training time required, we are actually able to make use of the leave-one-out variant of exhaustive cross-validation. For the case of more modern summarization methods, especially those that make use of deep networks, this is not generally an option as both the training time and segment scoring time would be too long.

7.2.1 Other Datasets

Our first method involves for a given dataset, performing testing using every video in the dataset, and using every video from the other datasets as the training set. This allows us to train a single model for each dataset, then perform summarization for every video in a given dataset. Since no videos from the testing dataset are used for training, this sampling method allows us to assess the ability of our model when applied to an arbitrary video in-the-wild. An example visualization of this sampling method can be found in Figure 7.1.

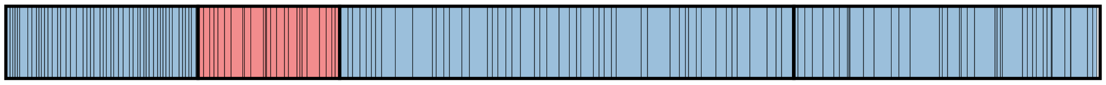


Figure 7.1: An example of the train and test set used for computing the summarization performance of any video in the SumMe dataset using the other datasets sampling method. Videos colored blue represent the training set, while ones in red represent the test set. In this case, every video in the SumMe dataset is used for testing, and all other videos are used for training.

7.2.2 All Datasets Leave-one-video-out

For this sampling method, we train a model for each video which uses the segments from every other video across all datasets as the training set, and the segments for the current video as the testing set. By comparing the results of this sampling to the one described in Section 7.2.1, we are able determine the importance of training using videos similar to our target testing videos. An example visualization of this sampling method can be found in Figure 7.2.

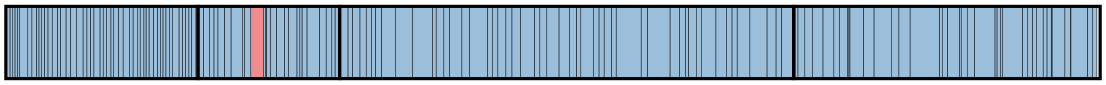


Figure 7.2: An example of the train and test set used for computing the summarization performance of a single video using the all datasets leave-one-out sampling method. Videos colored blue represent the training set, while ones in red represent the test set. In this case, all but one video is used to train a model for scoring the segments of a single video.

7.2.3 Same Dataset Leave-one-video-out

For this sampling method, we train a model for each video which uses the segments from every other video in the dataset as the training set, and the segments for the current video as the testing set. This allows us to assess how our model performs when dealing with a limited amount of training data. It is also useful to compare to the sampling described in Section 7.2.2 to get an idea of the expected change in performance possible when additional data is available. An example visualization of this sampling method can be found in Figure 7.3.

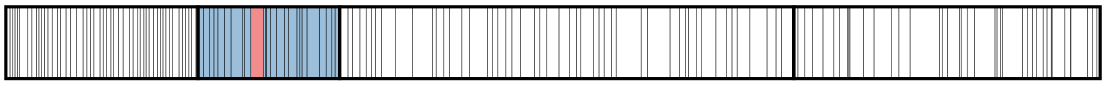


Figure 7.3: An example of the train and test set used for computing the summarization performance of a single video using the same dataset leave-one-out sampling method. Videos colored blue represent the training set, while ones in red represent the test set. In this case, one video from the SumMe dataset is used as a test, while the remaining videos in the dataset are used for training. Videos in the other datasets are not used at all.

7.2.4 Leave-one-segment-out

This sampling method operates at a lower level than the previous methods, and for each *segment*, training a model which uses every other segment as the training set, and just the current segment as the test set. By comparing to the previous methods, especially the one from Section 7.2.2, this method allows us to determine how the performance of our model changes when our training data contains segments which are highly correlated to our target test segment. An example visualization of this sampling method can be found in Figure 7.4.

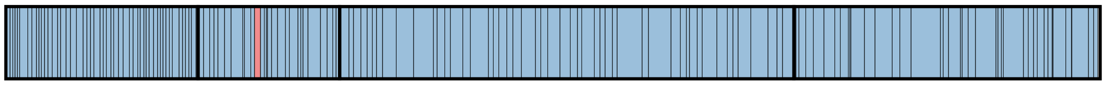


Figure 7.4: An example of the train and test set used for computing the summarization performance of a single video using the leave-one-segment-out sampling method. In this case, we divide the data at the segment level, rather than the video level. Segments colored blue represent the training set, while ones in red represent the test set. In this case, a single segment of a video in the SumMe dataset is used for testing, while all other segments across all datasets are used for training the model.

7.3 Model Optimization

Our final goal is to obtain an optimized model for segment scoring which maximizes the mean F_1-measure of all videos in the SumMe dataset. We start by first performing summarization using each of our two base models and four sampling methods. Based on these results, we select the best model, and proceed to perform grid search over the parameters of this model.

7.3.1 Base Models

As base models, we use an XGBoost model and a random forest model, each using the default parameters, which can be found in Figure 7.5.

XGBoost Parameters	
Parameter	**Value**
`learning_rate`	0.1
`gamma`	0
`max_depth`	3
`min_child_weight`	1
`n_estimators`	100
`subsample`	1
`colsample_bytree`	1
`reg_alpha`	0

(a) Default parameters used for our XGBoost regression model.

Random Forest Parameters	
Parameter	**Value**
`max_features`	auto
`n_estimators`	10

(b) Default parameters used for our random forest regression model.

Figure 7.5: The default parameters used for each of our base models.

For testing, we use the SumMe dataset, and for each model, perform training and testing using each of the four sampling methods previously discussed in Section 7.2. The primary sampling method of interest to us is the "Other Datasets" method, as it maximizes the training data available, while also minimizing the chances of overfitting, since the dataset used for evaluation is completely independent from the one used for training. The results of our initial testing can be found in Table 7.1.

	XGBoost				Random Forest			
Video Name	**Other**	**All One-out**	**Same One-out**	**Segment One-out**	**Other**	**All One-out**	**Same One-out**	**Segment One-out**
Air Force One	0.144	0.195	0.110	0.531	0.200	0.189	0.212	0.504
Base jumping	0.160	0.131	0.154	0.216	0.180	0.123	0.149	0.205
Bearpark climbing	0.261	0.178	0.181	0.301	0.165	0.218	0.210	0.286
Bike Polo	0.310	0.155	0.219	0.413	0.356	0.233	0.239	0.392
Bus in Rock Tunnel	0.120	0.126	0.144	0.286	0.093	0.131	0.187	0.272
Car railcrossing	0.141	0.165	0.104	0.366	0.091	0.150	0.304	0.348
Cockpit Landing	0.233	0.120	0.108	0.423	0.135	0.146	0.052	0.402
Cooking	0.348	0.287	0.154	0.408	0.183	0.341	0.284	0.388
Eiffel Tower	0.090	0.145	0.084	0.413	0.208	0.165	0.241	0.392
Excavators river crossing	0.176	0.229	0.152	0.287	0.188	0.088	0.231	0.273
Fire Domino	0.192	0.173	0.126	0.337	0.158	0.112	0.158	0.320
Jumps	0.109	0.130	0.423	0.423	0.433	0.402	0.121	0.402
Kids playing in leaves	0.145	0.103	0.065	0.439	0.089	0.136	0.079	0.417
Notre Dame	0.110	0.096	0.107	0.362	0.104	0.210	0.159	0.344
Paintball	0.304	0.178	0.186	0.471	0.272	0.102	0.251	0.447
Playing on water slide	0.218	0.251	0.103	0.256	0.052	0.250	0.149	0.243
Saving dolphines	0.183	0.098	0.064	0.294	0.187	0.080	0.098	0.279
Scuba	0.187	0.276	0.080	0.203	0.141	0.089	0.098	0.193
St Maarten Landing	0.557	0.580	0.051	0.491	0.460	0.209	0.206	0.466
Statue of Liberty	0.130	0.144	0.106	0.339	0.153	0.047	0.125	0.322
Uncut Evening Flight	0.074	0.093	0.137	0.444	0.058	0.192	0.085	0.422
Valparaiso Downhill	0.225	0.335	0.188	0.342	0.129	0.199	0.158	0.325
car over camera	0.295	0.305	0.287	0.357	0.120	0.156	0.203	0.339
paluma jump	0.143	0.120	0.111	0.464	0.046	0.115	0.145	0.441
playing ball	0.092	0.262	0.244	0.368	0.094	0.223	0.224	0.350
Average	0.198	0.195	0.147	0.369	0.172	0.172	0.175	0.351
Variance	0.011	0.011	0.006	0.007	0.011	0.006	0.004	0.006

Table 7.1: The resulting F_1-measure values computed for each model and sampling method combination for each video in the SumMe dataset. Higher values represent better performance for a given video and sampling method.

The results presented in Table 7.1 provide us with a large range of information related to our models and datasets. In particular, we see that for identical sampling methods, the XGBoost model outperforms the random forest model for all sampling methods except for the "Same One Out" method. The fact that the random forest model performs better only when trained using data from the same dataset could indicate that it is overfitting on dataset-specific trends. This, combined with the generally higher accuracy obtained using XGBoost suggests that we should use it as our only model moving forward.

Another important thing to note is the significantly higher F_1-measure values attained when using the "Segment One-out" method, where only the single testing segment is excluded from training, as opposed to the "All One-out", where all segments from the testing video are excluded. If the difference was minor, this could perhaps be explained simply by the fact that using segments with a strong correlation to our target segment yields a model with a deeper understanding of the category of segment is was trained to score. However, the significant difference between the F_1-measure suggests otherwise—specifically that the default parameters used for each model encourage—or at least facilitate—overfitting.

Since we will only be using an XGBoost model moving forward, we feel it is also important to examine the differences between the F_1-measure values obtained for each sampling method, and provide possible explanations for the difference we observed. Generally, we see that for the "Segment One-out" method, the F_1-measure value is almost double that of the next closest sampling method—"All One-out". As mentioned previously, this suggests that the current model parameters encourage overfitting in the final model. Future parameter tuning may be able to reduce or even mitigate this adverse effect. We also see that the "Other" method slightly out-performs the "All One-out" method, while still maintaining the same variance among all videos. We can also see that when just using the same dataset for training and testing ("Same One-out"), we end up with the worst results among all sampling methods. Taking into account these observations among all sampling methods for XGBoost, we can infer two major conclusions, namely that:

1. videos in the same dataset do not generally exhibit higher content correlations than videos from different datasets; and
2. additional data can be helpful, but the addition of data with too many similarities to the test set can also be detrimental to the final trained model.

The first conclusion is based on the fact that using video from the same dataset during training actually resulted in a *decrease* in the quality of the final summary, while the second conclusion is based on the fact that the addition of out-of-dataset data for the "Other" and "All One-out" models resulted in a quality increase compared to the "Same One-out" sampling method.

Using the default parameters for our XGBoost model, we are able to obtain an average F_1-measure value slightly below, but still quite similar to the score obtained by the reference dataset[24] (0.234), and even similar to modern methods[29] (0.2655). Our next step is therefore to perform grid search over the relevant parameters of the default XGBoost model in order to maximize the F_1-measure obtained for the SumMe dataset.

7.3.2 XGBoost Grid Search

Grid search is a common method used in machine learning, where for multiple model parameters and possible values, we iterate over Cartesian products of different combinations of these possible parameter values, and at each step retain the parameter set which results in the maximum final F_1-measure averaged over every video. If needed, a more technical description is provided in Section 2.3.5.

For the case of an XGBoost model, we perform grid search in a step-wise manner, where at each step, we iterate over the Cartesian product of the parameters and respective values for each step, and moving forward, retain the parameter combination which maximizes the F_1-measure over all videos in the SumMe dataset.

7.3.2.1 Step 1: Max Depth and Minimum Child Weight

The two parameters which have the most significant effect on the outcome of an XGBoost model are the "Max Depth", and the "Minimum Child Weight". For the "Max Depth", the typical range of values to test is $[3, 5, 7, 9]$, and for "Minimum Child Weight" the range is $[1, 3, 5]$. The resulting F_1-measure values for each parameter set can be found in Table 7.2. For this initial set of values, we found the optimal result occurred with a "Max Depth" of 3 and a "Minimum Child Weight" of 5. We performed a second grid search over values similar to this, yielding the results in Table 7.3. Overall, we are able to achieve an increase from an F_1-measure of 0.198 to 0.237 using a "Max Depth" of 3 and a "Minimum Child Weight" of 5.

Max Depth	**Minimum Child Weight**	**F_1-measure**
3	1	0.207
3	3	0.213
3	5	0.237
5	1	0.199
5	3	0.234
5	5	0.201
7	1	0.203
7	3	0.213
7	5	0.192
9	1	0.219
9	3	0.209
9	5	0.188

Table 7.2: The resulting mean F_1-measure values obtained for each of our initial grid search parameter values.

Max Depth	Minimum Child Weight	F_1-measure
2	4	0.219
2	5	0.214
2	6	0.212
3	4	0.209
3	5	0.237
3	6	0.216
4	4	0.222
4	5	0.214
4	6	0.217

Table 7.3: The resulting mean F_1-measure values obtained for our second set of grid search parameter values.

7.3.2.2 Step 2: Gamma

The next parameter we tune is "Gamma", testing values $[0.0, 0.1, 0.2, 0.3, 0.4]$, and the default is 0.0. The results of the grid search can be found in Table 7.4. We can see that in this case, the default value of 0.0 results in the maximum F_1-measure.

Gamma	F_1-measure
0.0	0.237
0.1	0.230
0.2	0.200
0.3	0.198
0.4	0.209

Table 7.4: The resulting mean F_1-measure values obtained for fine-tuning of "Gamma".

7.3.2.3 Step 3: Subsample and Col-sample By-tree

The final parameters we tune are “Subsample” using the values $[0.6, 0.7, 0.8, 0.9, 1.0]$, with 1.0 being the default value, and “Col-sample By-tree” using the values $[0.6, 0.7, 0.8, 0.9, 1.0]$, with 1.0 being the default. A subset of the results of the grid search can be found in Table 7.5. We can see that in this case, the default values of 1.0 and 1.0 result in the maximum F_1-measure.

Subsample	**Col-sample By-tree**	**F_1-measure**
0.7	0.6	0.201
0.7	0.7	0.221
0.7	0.8	0.196
0.7	0.9	0.209
0.7	1.0	0.187
0.8	0.6	0.203
0.8	0.7	0.216
0.8	0.8	0.211
0.8	0.9	0.199
0.8	1.0	0.205
0.9	0.6	0.223
0.9	0.7	0.217
0.9	0.8	0.213
0.9	0.9	0.198
0.9	1.0	0.204
1.0	0.6	0.223
1.0	0.7	0.216
1.0	0.8	0.224
1.0	0.9	0.218
1.0	1.0	0.237

Table 7.5: The resulting mean F_1-measure values obtained for fine-tuning of “Subsample” and “Col-sample By-Tree”.

7.4 Results

In the end, we were able to develop and train a model using modern video summarization datasets which is able to obtain close-to state-of-the-art results on the SumMe dataset. Perhaps our most significant contribution lies in the speed at which we are able to compute these summaries compared to other state-of-the-art methods. Even on 4-year old commodity hardware—an `i5-3380M` CPU and with `16GB` of RAM and no dedicated GPU—we are able to generate an initial summary of a video with an arbitrary duration at faster than real-time speeds. That is, for a video with a duration of an hour, we are able to generate a summary in less than an hour. Afterwards, additional summaries can be generated almost instantly, and can even incorporate user input if desired.

In the remainder of this section, we take a look at the accuracy values obtained using previous work for the SumMe dataset, and compare them to our method. We go on to analyze the performance of our method, specifically computing the average speed over all videos in our test dataset. We additionally compute a linear fit to our raw data, and demonstrate that our method appears to have linear complexity in relation to video duration. We finish up by providing a performance versus accuracy plot for our method and other modern summarization methods, along with a small discussion on what it demonstrates.

7.4.1 SumMe Accuracy

Primarily, we use the SumMe dataset for evaluating our summarization system. We first train a segment scoring method using data from our other datasets, then for each video in the SumMe dataset, use our system to generate a summary. For each of these generated summaries, we compute the pairwise F_1-measure value against each user summary, recording the average value over all user summaries in Table 7.6.

	Dataset		Humans			Computational Methods				
Videoname	**Random**	**Upper Bound**	**Worst**	**Mean**	**Best**	**Uniform**	**Cluster**	**Attn.**	**Summe**	**Ours**
Air Force One	0.144	0.490	0.185	0.332	0.457	0.161	0.143	**0.215**	**0.318**	**0.362**
Base jumping	0.144	0.398	0.113	0.257	0.396	**0.168**	0.109	**0.194**	**0.121**	0.106
Bearpark climbing	0.147	0.330	0.129	0.208	0.267	0.152	**0.158**	**0.227**	0.118	**0.261**
Bike Polo	0.134	0.503	0.190	0.322	0.436	0.058	**0.130**	0.076	**0.356**	**0.301**
Bus in Rock Tunnel	0.135	0.359	0.126	0.198	0.270	**0.124**	0.102	0.112	**0.135**	**0.147**
Car railcrossing	0.140	0.515	0.245	0.357	0.454	**0.146**	0.146	0.064	**0.362**	**0.192**
Cockpit Landing	0.136	0.443	0.110	0.279	0.366	0.129	**0.156**	0.116	**0.172**	**0.201**
Cooking	0.145	0.528	0.273	0.379	0.496	**0.171**	0.139	0.118	**0.321**	**0.348**
Eiffel Tower	0.130	0.467	0.233	0.312	0.426	**0.166**	**0.179**	0.136	**0.295**	0.088
Excavators river crossing	0.144	0.411	0.108	0.303	0.397	0.131	**0.163**	0.041	**0.189**	**0.231**
Fire Domino	0.145	0.514	0.170	0.394	0.517	**0.233**	**0.349**	**0.252**	0.130	0.169
Jumps	0.149	0.611	0.214	0.483	0.569	0.052	**0.298**	0.243	**0.427**	**0.542**
Kids playing in leaves	0.139	0.394	0.141	0.289	0.416	**0.209**	**0.165**	0.084	0.089	**0.093**
Notre Dame	0.137	0.360	0.179	0.231	0.287	0.124	**0.141**	**0.138**	**0.235**	0.107
Paintball	0.127	0.550	0.145	0.399	0.503	0.109	0.198	**0.281**	**0.320**	**0.213**
Playing on water slide	0.134	0.340	0.139	0.195	0.284	**0.186**	0.141	0.124	**0.200**	**0.218**
Saving dolphines	0.144	0.313	0.095	0.188	0.242	**0.165**	**0.214**	**0.154**	0.145	0.128
Scuba	0.138	0.387	0.109	0.217	0.302	**0.162**	0.135	**0.200**	**0.184**	0.140
St Maarten Landing	0.143	0.624	0.365	0.496	0.606	0.092	0.096	**0.419**	**0.313**	**0.557**
Statue of Liberty	0.122	0.332	0.096	0.184	0.280	**0.143**	0.125	0.083	**0.192**	**0.259**
Uncut Evening Flight	0.131	0.506	0.206	0.350	0.421	**0.122**	0.098	**0.299**	**0.271**	0.081
Valparaiso Downhill	0.142	0.427	0.148	0.272	0.400	0.154	0.154	**0.231**	**0.242**	**0.288**
car over camera	0.134	0.490	0.214	0.346	0.418	0.099	**0.296**	0.201	**0.372**	**0.408**
paluma jump	0.139	0.662	0.346	0.509	0.642	**0.132**	0.072	0.028	**0.181**	**0.334**
playing ball	0.145	0.403	0.190	0.271	0.364	**0.179**	**0.176**	0.140	**0.174**	0.151
Average	0.139	0.454	0.179	0.311	0.409	0.143	0.163	**0.167**	**0.234**	**0.237**

Table 7.6: F_1-measure values resulting from testing various summarization methods on videos from the SumMe dataset. For each video, among the computational methods, the three highest results are highlighted using different shades of green. Darker shades are used for higher F_1-measure values, and hence better results.

Although the data presented in Table 7.6 provides us with a number of important observations, the primary one of interest to us is the fact that among all the computational methods, ours achieves the highest accuracy (F_1-measure). We achieve the highest accuracy on over 50% of the tested videos, and in cases where we don't, we often observe similar results to the SumMe method, where the less computationally complex methods such as uniform and cluster-based selection achieve the highest accuracy. A possible improvement to our work which makes use of these methods is later discussed in Section 8.2.

7.4.2 Performance

In order to support our claim of faster than real-time performance, we performed video summarization for each video in the SumMe dataset using our method, and recorded the required processing time in Table 7.7. From this raw data, we can see that on average, we are able to perform summarization at 1.82 times real-time. It is also important to notice that across all videos, our speed never falls below real-time. We additionally plot this data in Figure 7.6.

Video Name	**Duration (s)**	**Time (s)**	**Speed**
Jumps	38.00	19.12	1.99x
Cooking	85.80	22.16	3.87x
Fire Domino	53.73	27.99	1.92x
St Maarten Landing	70.04	36.72	1.91x
Scuba	74.03	48.45	1.53x
paluma jump	85.89	46.89	1.83x
Bike Polo	102.13	69.50	1.47x
Playing on water slide	102.27	54.76	1.87x
playing ball	103.97	54.52	1.91x
Kids playing in leaves	106.34	71.29	1.49x
Bearpark climbing	133.64	78.31	1.71x
Statue of Liberty	154.52	69.89	2.21x
car over camera	146.21	71.04	2.06x
Air Force One	179.76	103.59	1.74x
Notre Dame	192.00	106.87	1.80x
Base jumping	157.79	105.27	1.50x
Eiffel Tower	198.84	118.90	1.67x
Car railcrossing	169.34	115.14	1.47x
Bus in Rock Tunnel	171.10	109.00	1.57x
Valparaiso Downhill	172.77	115.51	1.50x
Paintball	254.25	137.37	1.85x
Saving dolphines	222.99	120.15	1.86x
Cockpit Landing	301.83	200.50	1.51x
Uncut Evening Flight	322.72	215.42	1.50x
Excavators river crossing	388.84	210.87	1.84x
Average			1.82x

Table 7.7: Raw performance data for our method applied to each video in the SumMe dataset. The duration of each video is provided, along with the time required for our method to complete, and corresponding speed as a multiplier of real-time.

When plotted, the raw point data in Figure 7.6 appeared to follow a linear pattern, so we elected to compute a linear line-of-best-fit to the data. We use least squares to fit a first-degree polynomial to our data, obtaining a fit $y = 0.615x - 4.94$, which is plotted alongside our data in Figure 7.6. For this line, we compute the coefficient of determination to be $R^2 = 0.943$. This is a relatively high value, indicating that the line is a good estimator for our data, and therefore that our system appears to have linear complexity in terms of the video duration.

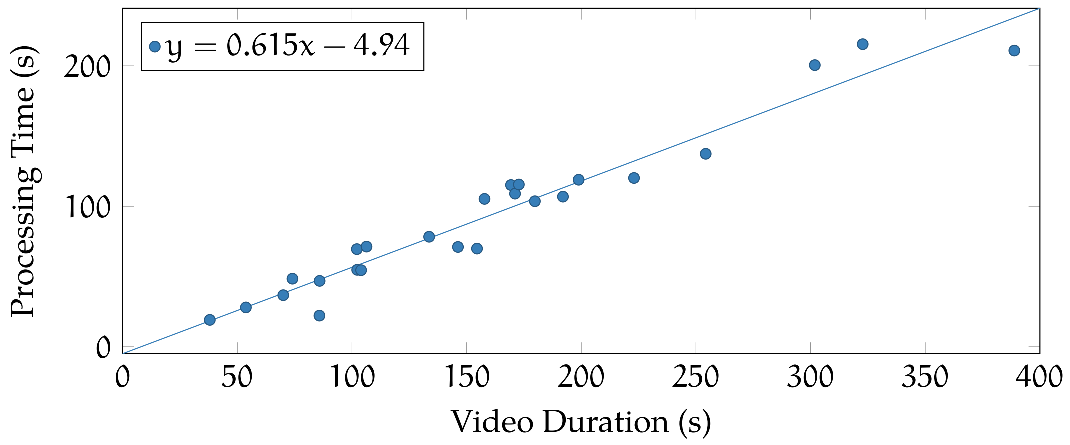


Figure 7.6: A plot of the video duration versus computation time data from Table 7.7. We additionally plot a line of best fit to our data, demonstrating the fact that the complexity of our method appears to be linear in terms of the duration of a video.

In order to compare the performance of our method to other methods, we compute the average performance of each method as a multiplier of real-time, and plot the performance value versus the accuracy of each method in Figure 7.7. When looking at the plot, we can see that there are two major groups; the more computationally complex methods of SumMe and LSTM appear in the top left corner, indicating that they achieve a high accuracy at the cost of low performance. The other group we see is in the bottom right corner, where the less complex methods are able to achieve very high performance, but suffer from significantly reduced accuracy. Finally, when we look at our method, we can see that we achieve a performance comparable to the computationally simple methods, and accuracy comparable to the more complex methods. In particular, only the LSTM method is able to achieve a higher accuracy than our method, at the cost of significantly decreased performance.

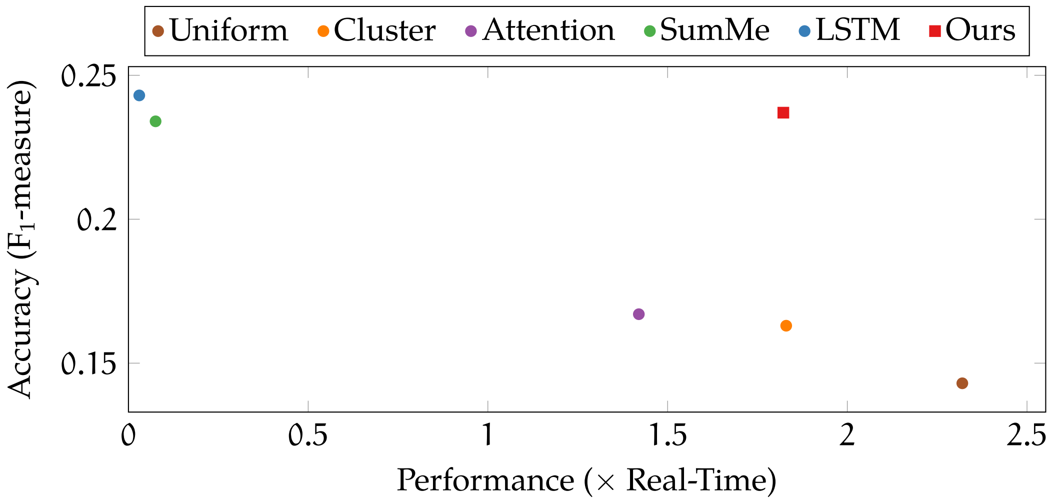


Figure 7.7: A plot of performance as a multiplier of real-time versus accuracy as an F_1-measure value for various summarization methods, including ours. Data for all methods previously examined in Section 7.4.1 is included, along with data for a state-of-the-art LSTM based method[27].

CHAPTER 8

Conclusions

In this chapter, we start by discussing the major contributions of our work. In addition to these contributions, we discuss the major limitations of our work compared to other similar works on video summarization, and some possible future work that could be performed to combat these limitations.

Overall, in our work, we were able to develop a high performance system for video summarization which obtains competitive results on the SumMe dataset compared to other modern methods. The fact that our system was designed to be efficient does however mean that with it comes some limitations, which we discuss, along with possible future work that could be used to combat these limitations.

8.1 Contributions

The primary contribution of our work is:

> A high performance video summarization system which is able to perform video summarization at real-time on commodity hardware.

Additionally, we developed a segmentation method which makes use of very low-level features to efficiently locate undesirable frames, then uses this information to compute optimal segments. This method could possibly be used by other summarization systems to improve their performance. In order to support generating personalized video summaries, our summarization method is able to accumulate user history over time, thereby learning each user's personal preferences. Finally, the design of our system supports generating additional summaries at virtually no cost, and a method for obtaining user preference information whenever an additional summary is requested.

8.2 Limitations and Future Work

While our lack of use of very high-level features gives us a significant speed benefit, it also leads to some limitations. Chief among these is that our method is unable to understand high-level concepts such as the presence of people or objects in a scene, or the actions a person may be performing. Without this higher level of understanding, we may be limited in the maximum accuracy we are able to obtain. In the future, it may be beneficial to investigate the result of incorporating additional features such as detected objects[22], general features such as GIST[17]/SIFT[18], motion features such as dense trajectories[19], or neural network layer features such as the pool 5 layer of GoogLeNet[21].

The results presented in Table 7.6 show us that for some specific videos, even very simple summarization methods such as uniform and cluster-based ones are able to achieve accuracy measures which exceed those of more complex methods such as ours. Since these simple methods generally have a very low computational complexity, reformulating our segment scoring model as an ensemble of our current method and some of these simpler methods could potentially be an easy way to gain the accuracy benefits of each individual method.

Our use of 0/1 knapsack as a method for selecting segments is efficient, but only operates locally on segments. This means that our generated summaries are unable to incorporate global factors across all segments. If we consider a video with multiple large scenes, one of which contains the majority of high-scoring segments, we can see why this may be an issue; the final summary will most likely only contain segments from a single scene, whereas a representative summary would contain segments from each scene. A possible improvement would be to incorporate some of the ideas from [33], and additionally take into account summary-level features such as interestingness, representativeness, and uniformity to optimize our final segment selection.

Our method of generating additional summaries is fairly rudimentary, simply reducing the scores of previously selected segments by some fixed bias value, and performing segment selection again. In some cases, this may result in an additional summary containing none of the segments that were in the original summary, which is often undesirable. A possible improvement would be to generate a summary which only reduces the scoring of some subset of the selected segments. The subset could be selected randomly, or perhaps based on the number of prior summaries each segment was already included in. Changing to a method such as this could potentially allow us to pinpoint exactly what segments the user is unsatisfied with.

Our use of fixed parameter values at various points in our system limits the ability of our system to improve and learn over time. One example in particular is the model weights α and β which are used for determining the amount of influence that the global scoring model and per-user scoring model have for determining a final segment scoring. We currently used fixed values of 0.5 for each of these. Different users may have different preferences in relation to the values used for these weights, and therefore incorporating them into our learning process could be an easy way to increase the accuracy of our system.

Bibliography

[1] HongJiang Zhang, Atreyi Kankanhalli, and Stephen W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Syst.*, 1(1):10–28, 1993.

[2] Huo Yi, Zhang Pengzhou, and Wang Yanfeng. Adaptive threshold based video shot boundary detection framework. In *Image Analysis and Signal Processing (IASP), 2012 International Conference on*, pages 1–5. IEEE, 2012.

[3] Chung-Lin Huang and Bing-Yao Liao. A robust scene-change detection method for video segmentation. *IEEE Trans. Circuits Syst. Video Techn.*, 11(12):1281–1288, 2001.

[4] Yong Jae Lee, Joydeep Ghosh, and Kristen Grauman. Discovering important people and objects for egocentric video summarization. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 1346–1353. IEEE Computer Society, 2012.

[5] Zheng Lu and Kristen Grauman. Story-driven summarization for egocentric video. In *2013 IEEE Conference on Computer Vision and Pattern Recognition,*

Portland, OR, USA, June 23-28, 2013, pages 2714–2721. IEEE Computer Society, 2013.

[6] Kevin Bleakley and Jean-Philippe Vert. The group fused lasso for multiple change-point detection. *arXiv preprint arXiv:1106.4199*, 2011.

[7] Zaid Harchaoui and Olivier Cappé. Retrospective mutiple change-point estimation with kernels. In *Statistical Signal Processing, 2007. SSP'07. IEEE/SP 14th Workshop on*, pages 768–772. IEEE, 2007.

[8] D. Defays. An efficient algorithm for a complete link method. *Comput. J.*, 20(4): 364–366, 1977.

[9] Rossano Schifanella, Miriam Redi, and Luca Maria Aiello. An image is worth more than a thousand favorites: Surfacing the hidden beauty of flickr pictures. In Meeyoung Cha, Cecilia Mascolo, and Christian Sandvig, editors, *Proceedings of the Ninth International Conference on Web and Social Media, ICWSM 2015, University of Oxford, Oxford, UK, May 26-29, 2015*, pages 397–406. AAAI Press, 2015.

[10] Miriam Redi, Nikhil Rasiwasia, Gaurav Aggarwal, and Alejandro Jaimes. The beauty of capturing faces: Rating the quality of digital portraits. In *11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition, FG 2015, Ljubljana, Slovenia, May 4-8, 2015*, pages 1–8. IEEE Computer Society, 2015.

[11] Jana Machajdik and Allan Hanbury. Affective image classification using features inspired by psychology and art theory. In Alberto Del Bimbo, Shih-Fu

Chang, and Arnold W. M. Smeulders, editors, *Proceedings of the 18th International Conference on Multimedia 2010, Firenze, Italy, October 25-29, 2010*, pages 83–92. ACM, 2010.

[12] Robert M. Haralick, K. Sam Shanmugam, and Its'hak Dinstein. Textural features for image classification. *IEEE Trans. Systems, Man, and Cybernetics*, 3(6):610–621, 1973.

[13] Hamid R. Sheikh, Zhou Wang, and Alan C. Bovik. No-reference perceptual quality assessment of JPEG compressed images. In *Proceedings of the 2002 International Conference on Image Processing, ICIP 2002, Rochester, New York, USA, September 22-25, 2002*, pages 477–480. IEEE, 2002.

[14] Kuang-Chern Ng, Aun Neow Poo, and Marcelo H. Ang. Practical issues in pixel-based autofocusing for machine vision. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, ICRA 2001, May 21-26, 2001, Seoul, Korea*, pages 2791–2796. IEEE, 2001.

[15] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007.

[16] Xin Jin, Jingying Chi, Siwei Peng, Yulu Tian, Chaochen Ye, and Xiaodong Li. Deep image aesthetics classification using inception modules and fine-tuning connected layer. In *8th International Conference on Wireless Communications & Signal Processing, WCSP 2016, Yangzhou, China, October 13-15, 2016*, pages 1–6. IEEE, 2016.

[17] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[18] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[19] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, pages 3169–3176. IEEE Computer Society, 2011.

[20] David A. Forsyth. Object detection with discriminatively trained part-based models. *IEEE Computer*, 47(2):6–7, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[22] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788. IEEE Computer Society, 2016.

[23] Naveed Ejaz, Irfan Mehmood, and Sung Wook Baik. Efficient visual attention based framework for extracting key frames from videos. *Sig. Proc.: Image Comm.*, 28(1):34–44, 2013.

[24] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc J. Van Gool.

Creating summaries from user videos. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII*, volume 8695 of *Lecture Notes in Computer Science*, pages 505–520. Springer, 2014.

[25] Yale Song, Miriam Redi, Jordi Vallmitjana, and Alejandro Jaimes. To click or not to click: Automatic selection of beautiful thumbnails from videos. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 659–668. ACM, 2016.

[26] Pamela H. Vance. Knapsack problems: Algorithms and computer implementations (S. martello and p. toth). *SIAM Review*, 35(4):684–685, 1993.

[27] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, volume 9911 of *Lecture Notes in Computer Science*, pages 766–782. Springer, 2016.

[28] Naila Murray, Luca Marchesotti, and Florent Perronnin. AVA: A large-scale database for aesthetic visual analysis. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2408–2415. IEEE Computer Society, 2012.

[29] Yale Song, Jordi Vallmitjana, Amanda Stent, and Alejandro Jaimes. Tvsum: Summarizing web videos using titles. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 5179–5187. IEEE Computer Society, 2015.

[30] Sandra Eliza Fontes de Avila, Ana Paula Brand
textasciitilde ao Lopes, Antonio da Luz Jr., and Arnaldo 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.

[31] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 886–893. IEEE Computer Society, 2005.

[32] Thomas E. Cason. The titanic3 data frame.

[33] Michael Gygli, Helmut Grabner, and Luc J. Van Gool. Video summarization by learning submodular mixtures of objectives. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3090–3098. IEEE Computer Society, 2015.

[34] Mayu Otani, Yuta Nakashima, Esa Rahtu, Janne Heikkilä, and Naokazu Yokoya. Video summarization using deep semantic features. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision - ACCV 2016 - 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part V*, volume 10115 of *Lecture Notes in Computer Science*, pages 361–377. Springer, 2016.

[35] John F. Schlag, Arthur C. Sanderson, Charles P. Neuman, and Frank C. Wimberly. Implementation of automatic focusing algorithms for a computer vision system with camera control. Technical Report CMU-RI-TR-83-14, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1983.

[36] J.M. Tenenbaum and Stanford Artificial Intelligence Laboratory. Accommodation in computer vision. Technical report, Department of Electrical Engineering, Stanford University, 1970.

[37] Ke Zhou, Miriam Redi, Andrew Haines, and Mounia Lalmas. Predicting pre-click quality for native advertisements. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 299–310. ACM, 2016.

[38] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1867–1874. IEEE Computer Society, 2014.

[39] Christos Sagonas, Epameinondas Antonakos, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: database and results. *Image Vision Comput.*, 47:3–18, 2016.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[41] Hongwei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE International Conference on Image Processing, ICIP 2014, Paris, France, October 27-30, 2014*, pages 343–347. IEEE, 2014.

[42] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In Xianghua Xie, Mark W. Jones, and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, pages 41.1–41.12. BMVA Press, 2015.

[43] Erik Learned-Miller, Gary B. Huang, Aruni RoyChowdhury, Haoxiang Li, and Gang Hua. Labeled faces in the wild: A survey. *Advances in Face Detection and Facial Image Analysis*, pages 189–248, 2016.

[44] Chris Biemann. Chinese whispers. In *Proceedings of TextGraphs: the First Workshop on Graph Based Methods for Natural Language Processing on the First Workshop on Graph Based Methods for Natural Language Processing - TextGraphs '06*. Association for Computational Linguistics, 2006.

[45] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[46] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[47] Robert K. Bryll, Ricardo Gutierrez-Osuna, and Francis K. H. Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302, 2003.

[48] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu Aggarwal,

Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.