# Leveraging Image Retrieval for Unsupervised Detection of Recurring Content in TV Shows´ Video Files

Master´s Thesis

Niels ten Boom

**Supervisor:** Prof. dr. ir. A.P. de Vries
**Second reader:** Prof. M.A. Larson
**External supervisor:** D. Odijk

*Special thanks to RTL for enabling me to write this thesis with them.*

## Abstract

This thesis presents research on image retrieval-based approaches for recurring content detection in Video On Demand (VOD) TV-shows. The method utilizes feature vectors constructed from video frames and an image retrieval framework to efficiently compare frame similarity across TV-show episodes, these frame similarities are then used to detect recurring parts. A best precision score of 0.884 at 0.687 recall and a best recall score of 0.892 at 0.748 precision were achieved. An altered experimental setup for bumper detection achieved a best precision score of 0.611 at 0.917 recall and a best recall score of 0.979 at 0.288 precision. With these results we present novel research for unsupervised detection of recurring content in VOD shows.

# Contents

# Chapter 1

# Introduction

Viewing rates for TV are dropping gradually while the number of users of streaming services such as Netflix and Videoland are growing rapidly year over year. This shift from TV to Video on Demand introduces new problems for broadcasters, as they now have to support a hybrid form of traditional broadcasting and Video On Demand (VOD). In this thesis we consider a method that could help to improve the user experience for VOD.

Video content is optimized for just one form, and currently that is broadcast television. This means that the video content may contain recurring content consisting of: recaps, opening credits, bumpers (to ease a viewer into commercials), closing credits and previews. A viewer watching this content back-to-back may find it preferable to be able to skip this recurring content to improve their viewing experience. Providing this functionality to users, requires metadata on where the skippable content occurs in the videos. For a large percentage of content, such metadata has not been retained. Videoland currently hosts over 1000 different titles consisting of multiple seasons and episodes and this selection is always changing. An automated solution that can detect this content in videos would be very useful to transform the video content into a format suited for VOD. In this thesis we will explore methods for such an automated solution.

There is little margin for error with this solution. The accuracy of the end result should be sufficiently high for it to be trusted to label all of the content automatically. If that is not the case, it needs double checking by a human, because if a part of the video gets mislabeled, a user may skip over actual content. It is critical to RTL that this does not happen.

This thesis presents research on unsupervised detection of recaps, opening credits, bumpers, closing credits and previews given the video files from a TV-show. We are going to attempt to tackle this problem by using image retrieval techniques. The motivation behind this is that the overlapping characteristic of the segments is that they reoccur. To be able to detect recurring content we are going to compare similarities of frames across content, this should be done efficiently and accurately, both attributes are important in image retrieval research. Our main research question then is:

> ***How well does an image retrieval approach perform in accurately detecting recurring content across a TV-show?***

With the subquestions:

> ***How can image retrieval be used to detect recurring content?***
>
> ***How accurate is each image retrieval approach?***
>
> ***How can the best method be utilized in practice?***

This thesis aims to answer aforementioned research questions. Our hypothesis is that such a method may achieve a level of effectiveness that allows practical application at RTL. The rest of this chapter will be used to expand on the different type of video classes which this research is focused on. Chapter 2 discusses related work. Chapter 3 elaborates on the used methods for image retrieval. Chapter 4 describes the data and methodology and then the produced results are presented and discussed in Chapter 5. Lastly, a conclusion is drawn in Chapter 6.

## 1.1 Recurring Content Classes

In this section we discuss the different types of content to be detected. What all the content classes have in common is that they (partially) reappear in previous or future episodes or, as is the case with bumpers; that the content reappears within the episode itself. We will use this attribute for the detection.

### Recaps

A television show typically has recaps before the opening credits. In the recaps, content of previous episodes is repeated to refresh the viewer's memory. This is useful for linear TV when an episode is aired every week. Someone watching episodes back-to-back ideally wants to skip the recaps together with the opening credits because they have seen the content recently. Not all material preceding the opening credits is a recap though, it can be original content. See Figure 1.1 for a visual example.

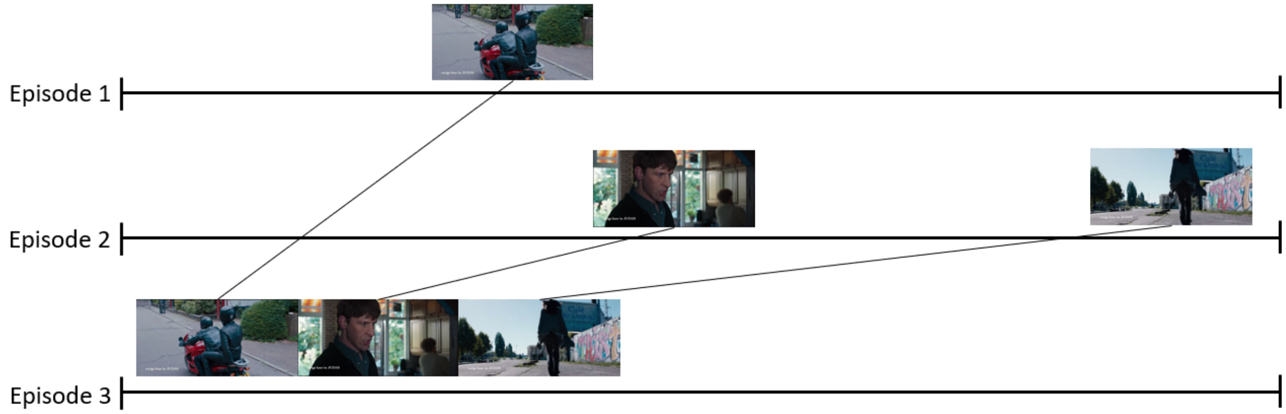


**Figure 1.1:** Visual depiction of recaps. Shots from episode 1 and 2 are shown at the start of episode 3 to give a brief summary of important events.

### Opening credits

The opening credits of a TV-show are generally the same for all of the episodes in the same season. It typically opens the show with a theme song, presenting the most important actors at the start. There is no prior knowledge on how the opening credits of a show look like and they often change every season, therefore the detection should be unsupervised. See Figure 1.2 for a visual example.

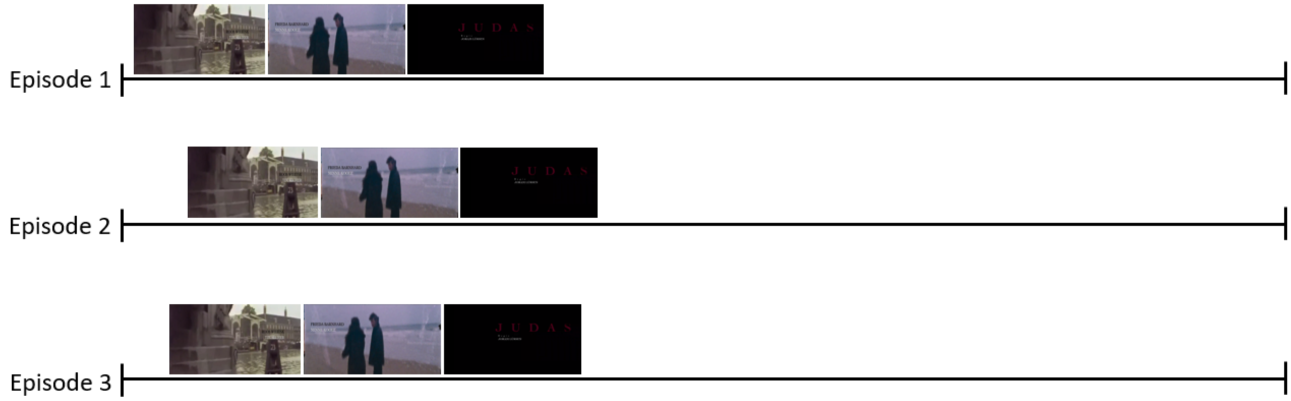


**Figure 1.2:** Visual depiction of opening credits. A very similar sequence of shots that starts at different times at the beginning of an episode.

### Bumpers

The bumper is a part of the video that eases a viewer into the upcoming commercial. Most of the times it has a voice over and text on screen saying: 'Next' (or the dutch translation of 'next') and footage of what is to be expected after the commercial break is shown. See Figure 1.3 for a visual example.

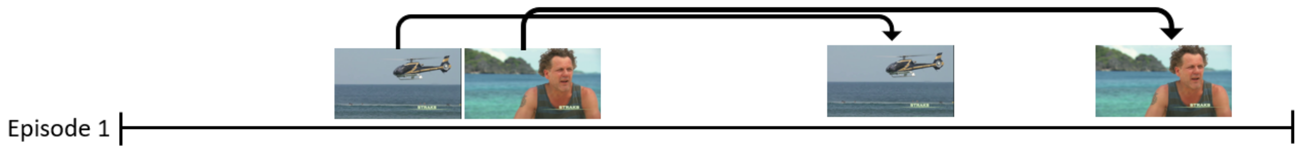


**Figure 1.3:** Visual depiction of bumpers. A summary of important shots are shown together that will occur at a later point of time in the episode.

### Previews

A preview is a segment at the end of an episode playing an advance showing of fragments of the next episode(s) are played. Previews can typically be found in content that was created specially for broadcast TV. It gives the viewer a taste of what to expect in the next episode that will air later. These previews may contain overlayed credits of people who helped in the production of the show. See Figure 1.4 for a visual example.

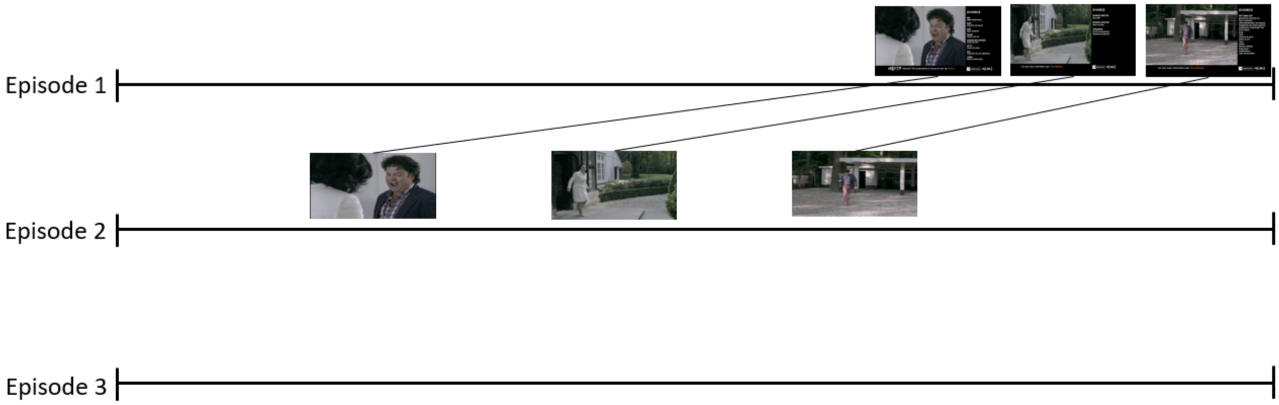


**Figure 1.4:** Visual depiction of previews. It shows shots that will occur in the next episode with a textual overlay.

### Closing Credits

The closing credits at the end of a video mostly consist of scrolling text and there is little variation between the frames. In these texts everybody related to the production of the video is mentioned. In general the frames all have a black background with white text. But backgrounds can also vary, therefore a solution that simply looks for black and white would not suffice. See Figure 1.5 for a visual example.

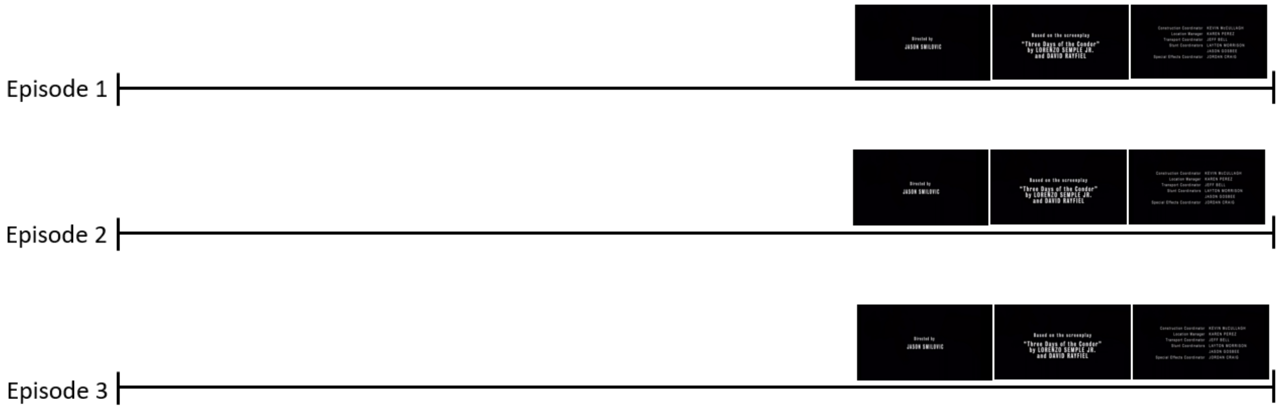


**Figure 1.5:** Visual depiction of closing credits. Frames consisting of text at the end of each episode in which the actors, producers and others related to the show are given credit.

## 1.2 Definitions

This section will be used to introduce definitions used throughout this thesis. We define a season with multiple episodes of a TV-show as T.

$$T = \{E_1, E_2, \ldots, E_x\}$$

An episode within a season is defined as a set of frames.

$$E = \{f_1, f_2, \ldots, f_y\}$$

The videos contains shot boundaries, we refer the set of locations of shotboundaries as B.

$$B = \{b_1, b_2, \ldots, b_z\}$$

# Chapter 2

# Related Work

No prior work exists that explores a solution for the problem previously described (unsupervised detection of these specific recurring segments). Related work exists that focuses on commercial or repeated sequence detection in large broadcast streams. These detection methods can be roughly divided into three groups: fingerprint, feature-based and unsupervised methods.

Fingerprint methods set up a database of fingerprints of known commercials or repeated sequences to detect these in a broadcast stream. Lienhart et al. [1] propose a method based on features to roughly localize advertisements in a stream and then construct a fingerprint database based on color coherence vectors. Gauch et al. [2] propose a method based on constructing feature vectors from color moments in a stream. Covell et al. [3] implement a fingerprinting method based on audio with visual verification after a proposed match. The disadvantage of these fingerprinting methods is that it is not unsupervised and the setting up and maintenance of such a fingerprint database requires much work.

Feature-based methods detect commercials based on extracted video features. Wang et al. [4] fuse the results of audio scene changes and textual content similarity between shots to segment programs including commercials.

With the unsupervised methods the authors typically do a dimensionality reduction operation and then try to find repeated sequences or commercials with clustering methods. Herley et al. [5] convert the stream with a Discrete Cosine Transform (DCT) to reduce dimensionality and then propose an extensive probability framework to detect repeated sequences. Benezeth et al. [6] use the Electronic Programme Guide (EPG) in addition to the dimensionality reduction to detect program boundaries.

Abduraman et al. [7] propose a system to detect repeated sequences in streams by performing a DCT operation on all of the frames in the stream and then use a micro clustering technique to detect repeated sequences. They were also able to link the trailers to their respective programs that occur at a later point in the stream.

All of the previously mentioned works never cover the specific topic of segmenting the classes mentioned in Section 1.1 and use properties only available to broadcast streams such as the EPG. The methods also focused on recurring content detection are exclusively based on detection of moderate to long sequences, it does not cover short recurring sequences stitched together such as a recap. The methods yielded high accuracies in the range of 90% - 95% precision, also most research is from many years ago and work that focuses on repeated sequence detection in broadcast streams do not vary much in methodology. This work is not focused on broadcast streams and aims for higher accuracies, we will not replicate the previously mentioned methodologies, however from the aforementioned papers we conclude that a large dimensionality reduction step will be necessary to efficiently process a large visual data set.

Dimensionality reduction and efficient matching of large amounts of visual data is important within content based image retrieval, Zheng et al. give a detailed summary of the most significant contributions from past years [8]. Their summary covers three periods in image retrieval: Early methods, SIFT-based methods and Convolutional Neural Network (CNN) based methods. Smeulders et al. present all the contributions of the early methods [9], these methods focused on looking at the color, texture and local geometry of images for retrieval [10, 11]. Not much later the Bag-of-words (BoW) model was proposed as a new method for image retrieval [12]. The advantage of this is that inverted indexes can be used for immediate retrieval of similar images. The BoW model paired with SIFT-descriptors [13] as the feature vectors was used in image retrieval research for years [14, 15, 16, 17, 18]. Since 2012 when the convolutional neural network was introduced [19] research switched to CNN-based methods for image retrieval because they achieved better performance on a variety of image retrieval tasks [20, 21, 22].

## 2.1 Shot Change Detection

Shot change detection is a technique that can be used on videos to determine shot boundaries, see Figure 2.1 for an example of such a shot change.

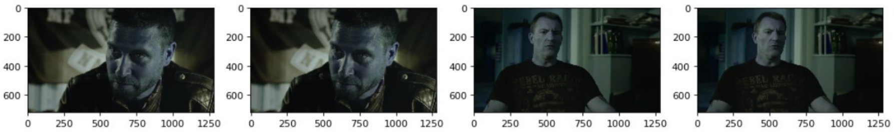


**Figure 2.1: Example of a shot change within 4 frames of a video.**

Shot change detection is going to be used to determine the shot boundaries. A lot of research has been done related to shot detection [23] and many techniques have been proposed.

We use a method that looks at the shift in mean color in HSV space [24]. The shot boundaries are classified by calculating the difference in hue, saturation and value for each frame and then from this the mean is calculated. If the mean is higher than a threshold H, then a shot boundary is marked, refer to algorithm 1 for a pseudo-code representation.

**Algorithm 1: Shot boundary detection**

```
shotBoundaries = List();
previousMeanHSV = getMeanHSV(Episode[0]);
for frame in Episode do
    meanHSV = getMeanHSV(frame);
    if abs(meanHSV - previousMeanHSV) > H then
        shotBoundaries.add(frame);
    end
    previousMeanHSV = meanHSV;
end
```

# Chapter 3

# Image Retrieval

Image retrieval is a subset of the research field Information Retrieval (IR). It investigates the problem where one has a query **q** expressing an information need and wants to find the best possible match for this **q** in a set of documents **D**.

In the case of image retrieval the query consists of an image for which the best matching image should be found in a document set of images. There are two approaches for doing this.

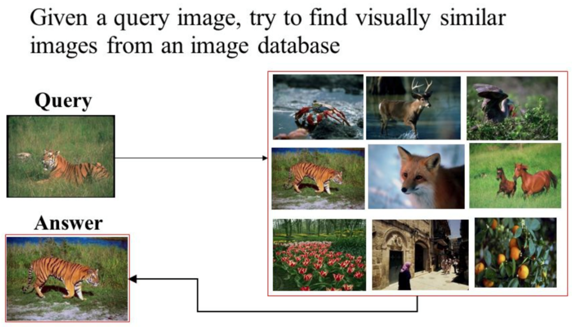

Given a query image, try to find visually similar images from an image database
Query
Answer

**Figure 3.1:** Visual example of image retrieval.

**Text-based image retrieval** refers to retrieval of images based on textual metadata associated with these images. It matches images based on their titles, keywords and more. The downside of this method is that if there is no metadata available then it needs to be added manually. With the growing sizes of image databases this can become quite inefficient, hence the more focus on content-based image retrieval in past years [25].

**Content-based image retrieval** is retrieval based on the content of the image rather than the metadata as is the case with text-based image retrieval. Computer vision is used to evaluate the image similarity, this can be done by looking at colors, shapes, textures and more. The advantage of content based image retrieval is that it does not rely on the metadata of images and thus does not require manual labeling.

## 3.1 Feature Vectors

A feature vector is a vector containing a numeric representation of multiple characteristics of an object. In this case the objects are images, the frames of a video. To illustrate feature vectors for images, a small example is given. Consider the following three images of 10 by 10 pixels consisting only of black or white pixels in Figure 3.2.

If we would like to measure the image similarity between these pictures then we could do this on a pixel-per-pixel basis, while this could work for low dimensional images, comparing HD images consisting of 1920 pixels by 1080 pixels would require massive computation and will require extensive logic to handle small deviations between images. This can be resolved by creating feature vectors to represent the images. A representation for these example images could be the number of black and white pixels in the image, resulting in the feature vector; $\mathbf{x} = [\# \ white \ pixels, \# \ black \ pixels]^T$. This vector can now be represented in 2D-space, and compared to other vectors in 2D space by calculating the distance between other points in that space. The distance could for example be computed by calculating the euclidean distance between each vector.

$$d(\mathbf{p},\mathbf{q}) = \sqrt{(p_1 - q_1)^2 + \ldots + (p_n - q_n)^2}$$

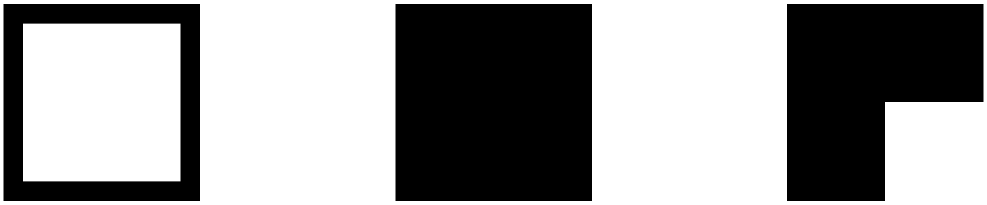


**Figure 3.2:** Three images of 10x10 pixels, consisting of only black and white pixels. The resulting feature vectors are $[64, 36]^T$, $[0, 100]^T$ and $[25, 75]^T$ respectively.

Figure 3.3 plots the feature vectors and the distances between. Example images 2 and 3 are the most alike because the distance between them is the smallest. Which seems in line with a visual inspection.

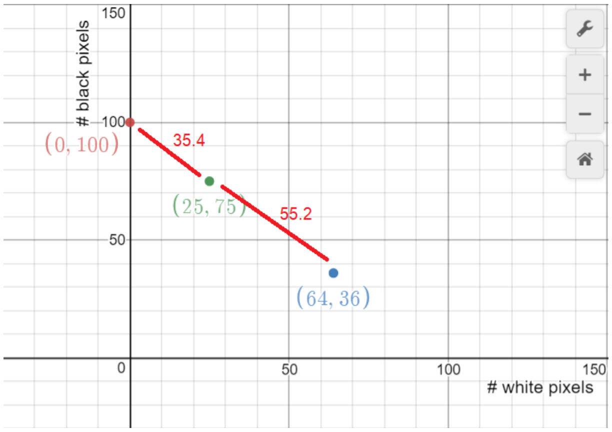


**Figure 3.3:** Example of distance in 2D between feature vectors that represent the example images. According to the chosen features, image 2 and 3 are the most alike.

The similarity of images can be determined by computing the distance between their feature vectors. A low distance indicates a higher similarity. Counting the number of black and white pixels for the construction of feature vectors is not a very robust method, a lot of research in image retrieval focused on the construction of alternative feature vectors to achieve better retrieval results most in line with human judgment. Part of this thesis is determining the best method for the construction of the image feature vectors to apply for our problem. We will expand on the methods that are going to be explored.

As mentioned in Chapter 2, Zheng et al. [8] outline three significant periods in image retrieval, the early methods focused on global descriptors mostly based on color and texture. These global descriptors were not good at handling image changes in illumination, translation, occlusion and truncation. This gave rise to local descriptors based SIFT methods until recently CNN-descriptors became the most popular.

We want to determine whether global or local descriptors are needed. Therefore we take two different global descriptor methods and one recent local descriptor method with one of the best performances. The chosen methods are color histograms as a global descriptor, color texture moments [26] as a method that combines the color and texture of an image and lastly CNN-descriptors were implemented as local descriptors.

This is not an exhaustive list of methods, but because computing feature vectors for a large amount of frames is resource intensive, we limited ourselves to these three. If one of the methods achieves promising results then future research can expand on variations of the method or if none of the methods achieve acceptable results, then future research could try other distinct methods.

## 3.2 Color Histograms

A color histogram is a representation of the distribution of colors in an image. Color histograms are a flexible and low dimensional way of representing images. A color histogram can be computed by counting the number of pixels in a certain color range, the size of this range is variable, called the bin size. A larger bin size results in lower dimensions of the resulting histogram. For example, if one chooses a bin size that is half of the intensity range. The resulting vector would have 6 dimensions, two bins for each color channel. See Figure 3.4 for an example color histogram with such a bin size.

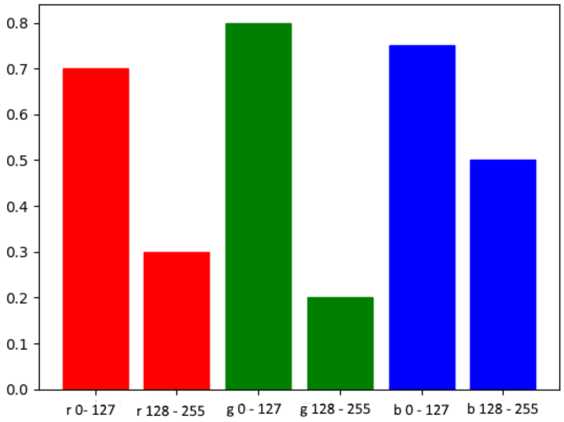
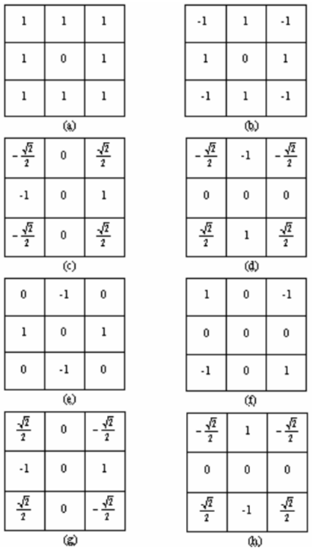


**Figure 3.4:** Color histogram with a large binsize (binsize=128) for illustration.

Color histograms are global descriptors because they describe the image on a global level. Viewing a frame as a color histogram loses a lot of detail, but for the specific problem recurring video content, this may be sufficient.

## 3.3 Color Texture Moments

Color texture moments as proposed by Yu et al. [26] are a low-level feature descriptor that integrates both color and texture characteristics of an image. An image is converted from RGB to HSV color space, and on each color map a 2d convolution is applied for each 3x3 filter shown in Figure 3.5.



| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(a)

| | | |
|---|---|---|
| -1 | 1 | -1 |
| 1 | 0 | 1 |
| -1 | 1 | -1 |

(b)

| | | |
|---|---|---|
| $-\frac{\sqrt{2}}{2}$ | 0 | $\frac{\sqrt{2}}{2}$ |
| -1 | 0 | 1 |
| $-\frac{\sqrt{2}}{2}$ | 0 | $\frac{\sqrt{2}}{2}$ |

(c)

| | | |
|---|---|---|
| $-\frac{\sqrt{2}}{2}$ | -1 | $-\frac{\sqrt{2}}{2}$ |
| 0 | 0 | 0 |
| $\frac{\sqrt{2}}{2}$ | 1 | $\frac{\sqrt{2}}{2}$ |

(d)

| | | |
|---|---|---|
| 0 | -1 | 0 |
| 1 | 0 | 1 |
| 0 | -1 | 0 |

(e)

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 0 | 0 | 0 |
| -1 | 0 | 1 |

(f)

| | | |
|---|---|---|
| $\frac{\sqrt{2}}{2}$ | 0 | $-\frac{\sqrt{2}}{2}$ |
| -1 | 0 | 1 |
| $\frac{\sqrt{2}}{2}$ | 0 | $-\frac{\sqrt{2}}{2}$ |

(g)

| | | |
|---|---|---|
| $-\frac{\sqrt{2}}{2}$ | 1 | $-\frac{\sqrt{2}}{2}$ |
| 0 | 0 | 0 |
| $\frac{\sqrt{2}}{2}$ | -1 | $\frac{\sqrt{2}}{2}$ |

(h)

**Figure 3.5:** Filters used for computing the color texture channels.

Which results in 8 channels for each color map, 24 channels in total. For each of the resulting channels the first and second color moments, which are the mean and standard deviation, are taken. This results in a 48-dimensional feature vector.

## 3.4 CNN Features

Convolutional neural networks (CNN) are a special type of neural networks that are good at handling image tasks. The 2012 paper by Krizhevsky et al. [19] that introduces the convolutional neural network gave rise to a lot of research in this area since then. CNNs are very good at image classification and segmentation tasks. This is because they take advantage of the fact that pixels close to each other are more likely to be related than pixels far apart. A classical neural network would link each pixel to all of the other pixels; this increases computational overhead and reduces accuracy. See Figure 3.6 for a small example of a CNN. The image is convolved with a 3x3 pixel filter which results in a new feature map, then max pooling can be applied to these feature maps that downsamples the maps by only keeping high activations. After repeating these steps a few times, the output can be fed through fully connected layers to for example learn a classification task. CNNs are not only good in image classification and segmentation; it was found that using the resulting channels after the convolution layers as feature vectors also perform well at image retrieval tasks. There are many ways to construct CNN feature vectors that perform well as described in [8].

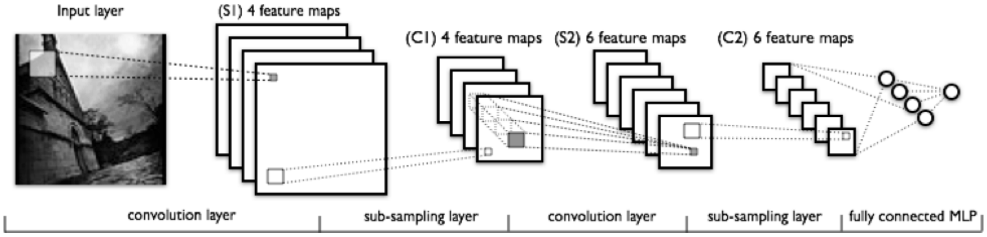


**Figure 3.6:** The working process of a neural network. Image from [27].

We will use the Regional Maximum Activation of Convolutions (R-MAC) method introduced by Tolias et al. [22] as our implementation for the feature vectors. The frames are fed through a pre-trained VGG16 neural network [28], which is a CNN trained for the Imagenet challenge; an image classification problem. A network trained for image classification captures much of the image context with its filters. The last fully connected layers of the network are used for the classification task and are thus discarded, the resulting channels with activations from the convolutional layers are divided up into regions. For each of these regions the Maximum Activations of Convolutions (MAC) feature vector is calculated, and all these feature vectors are then post-processed by applying L2-normalization, PCA-whitening and L2-normalization.

PCA is a technique that can be used to reduce the dimensionality of vectors while roughly keeping the same information it encapsulates. PCA-whitening is a closely related step that removes the correlation between features and makes the vector have unit variance.

After these steps, all the feature vectors are summed into one resulting feature vector and L2-normalization is applied one last time as the final step, resulting in a 512-dimensional vector.

# Chapter 4

# Methodology

We want to explore whether we can detect recurring content unsupervised and if so, which method of feature vector construction scores the best in terms of precision and recall. Initially a non-generalized approach was tried, where a unique method for each content class was used (For example: OCR for textual closing credits, similarity matrices for opening credits and feature based classification for recaps and previews). However, this was swiftly found to be ineffective because of a high variety of edge cases in creative non-structured video content.

The characteristic of all the segment classes is that they reoccur either in previous or future episodes. To match frames from one episode with other episodes, feature vectors were constructed from the frames and the distances between these vectors were computed and saved. If a part of the video consecutively has matching feature vectors from previous or future episodes, then that part is labeled as a recurring segment. It is therefore important that the resulting feature vectors do not mismatch between actual recurring content and content that looks alike. Methods chosen for construction of the feature vectors should therefore not result in feature vectors of very high dimensionality, because this has proven to reduce the distance between the farthest and closest points [29], likely diminishing the distinctive property of the feature vectors that is needed.

We will do separate experiments for the bumper detection as they require a different approach for detection because they only contain recurring content from within an episode itself.

All of the experiments were coded in Python and the code of the experiments can be found on Github [30].

## 4.1 Data

The data set contains 80 video files originating from 16 different seasons of tv-shows, amounting to around 50 hours of content. Only for three of these seasons do we use the full array of episodes, from the rest of the seasons, only the first three episodes will be looked at to have a more varied data set at a reasonable amount of human judgement. Originally these files were in *.mxf* full broadcast format, this meant each file being 25GB on average. All the files were converted to MP4 files having a width of 320 pixels width and the length to maintain the aspect ratio. The resizing step was done to drastically decrease the computing time needed. The conversion was done with FFmpeg [31].

For each file the start and end timestamps for the recaps, opening credits, bumpers, closing credits and previews were annotated in HH:MM:SS format in a CSV file, so that it could be loaded and compared effectively.

**Table 4.1:** Details of the data that was used to perform the experiments with.

| | Total seconds | % of recurring | % of total |
|---|---|---|---|
| Recaps | 3275 | 34.8% | 1.7% |
| Opening credits | 4669 | 49.5% | 2.5% |
| Closing credits | 690 | 7.3% | 0.4% |
| Previews | 592 | 6.3% | 0.3% |
| Recurring | 9423 | - | 4.9% |
| **Total** | **188981** | - | - |

Table 4.1 shows the distribution of the different recurring content classes. It can be noted that the classes are not in balance; the recurring content is weighted towards recaps and opening credits. Recurring content consists of 4.9% of the total content, thus with a perfect method, that is the percentage of time we could save of people watching the content back-to-back.

## 4.2 Feature Vector Construction

Video plays at 25 frames per second, and neighboring frames usually vary slightly. We will take advantage of this property to reduce computational complexity, by either sub sampling every 3 frames or only considering shot boundary frames. We take all the video files of a season $E_x \in T$ and convert each file to a set of feature vectors $S_x = \{v_1, v_2, \ldots, v_l\}$ with the function $f$.

$$f(E_x) = S_x = \{v_1, v_2, \ldots, v_l\}$$

The rest of the sections will expand on different implementations of the function $f$, how the different types of feature vectors are constructed. We use three different methods for the feature vector construction. These methods were chosen because of their efficient computation and accuracy in image retrieval tasks and to see if there is a difference between global and local descriptors applied to this problem. The methods chosen were color histograms, color texture moments and CNN features.

### 4.2.1 Color Histograms

For the construction of the color histograms the pixel intensities in each channel (Red, Green, Blue (RGB)) are counted and binned according to a specified bin size. All these bins will be concatenated to result in a feature vector, the size of the feature vector is a result of the chosen bin size.

Each single channel is represented by 60 bins. These are then concatenated into a 1D vector, resulting in a feature vector $\mathbf{v}$ where $|\mathbf{v}| = 180$. This vector is then L1 normalized, such that:

$$\mathbf{v} = \frac{v_i}{\sum_{i=1}^{n} |v_i|}$$

A histogram is computed for every 3 frames or shotboundary frame in the video and then stored and saved in a list that preserves the order of the video. This method will be named CH in the results chapter.

### 4.2.2 Color Texture Moments

Each filter A with dimensions M x M from Figure 3.5 is convolved with each HSV color channel B per frame of interest:

$$C[i,j] = \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}} A[m,n]B[i-m, j-n]$$

This results in 3 * 8 = 24 channels, of which the mean $\mu$ and standard deviation $\sigma$ are and concatenated into a single vector $\mathbf{v}$, meaning that $|\mathbf{v}| = 48$. This method will be called CTM in the results chapter.

### 4.2.3 CNN Features

For the computation of the CNN feature vectors we use a re-implementation written in Python from Github [32]. This re-implementation was used opposed to the original implementation by the authors, because it was written in Python and could thus be integrated well into our experimental framework. This re-implementation could only handle frames sized 224x224 however. So for this step the frames were further resized from 320xHEIGHT to 224x244. These frames were then fed through a VGG16 neural network pre-trained on Imagenet and on the channels before the fully connected layers the steps described in section 3.4 are applied. Resulting in the single vector $\mathbf{v}$ where $|\mathbf{v}| = 512$.

This method will be called CNN in the results chapter.

## 4.3 Matching and Detection

All the resulting lists of feature vectors will be added to the index of an empty Faiss instance. Faiss is a library for efficient similarity search of dense vectors [33, 34] and thus perfectly suited for our task. This library efficiently handles building the inverted index and nearest neighbor matching. The library supports implementations of several techniques. We use their ”Exact search for L2” implementation. Which does an exact brute force search given a set of vectors $x_i$ in dimension $d$ as the documents, when given a new vector $x$ in dimension d as the query, it efficiently performs the operation:

$$i = argmin_i ||x - x_i||$$

Where $||x - x_i||$ is the euclidean distance (L2). We use Faiss as opposed to implementing our own brute force search because it is open soure and it implemented multi-threading and utilization of BLAS libraries in an optimal way [35].

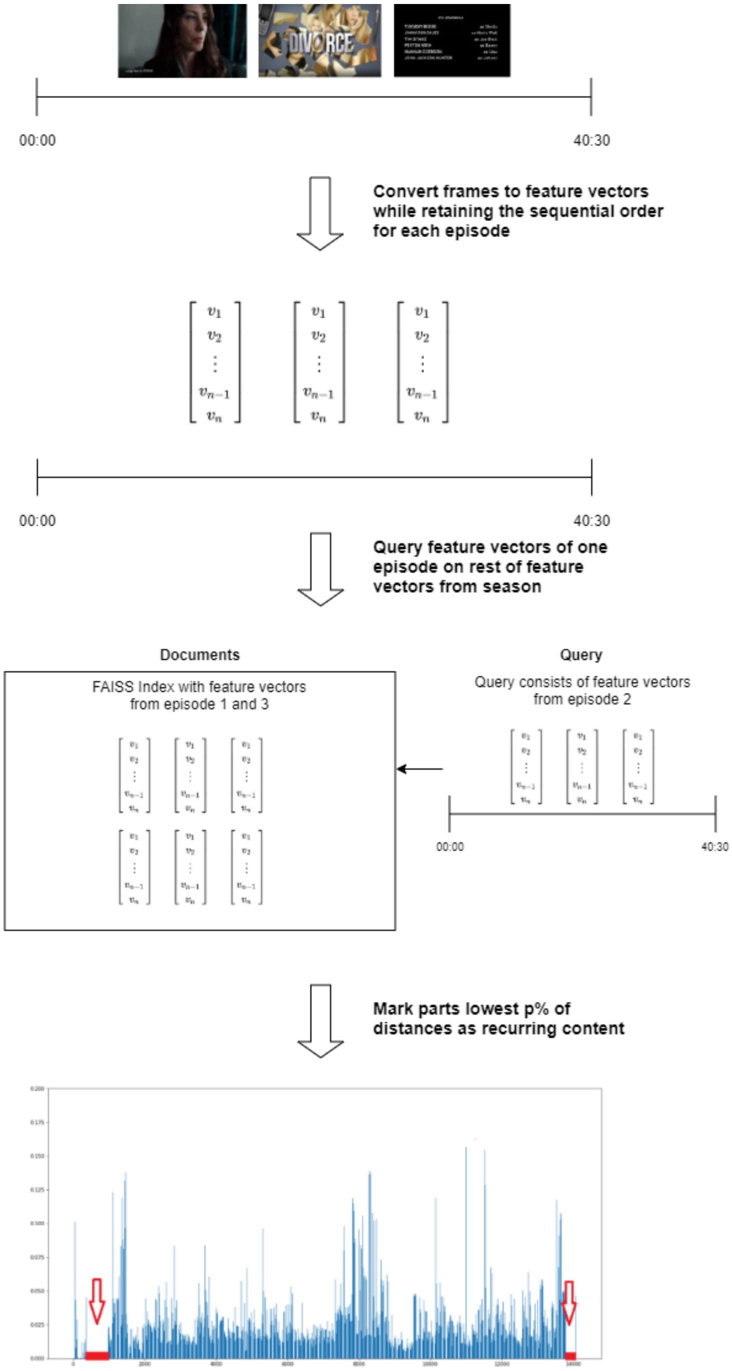

00:00
40:30
Convert frames to feature vectors while retaining the sequential order for each episode
$v_1$
$v_2$
$\vdots$
$v_{n-1}$
$v_n$
00:00
40:30
Query feature vectors of one episode on rest of feature vectors from season
Documents
FAISS Index with feature vectors from episode 1 and 3
Query
Query consists of feature vectors from episode 2
00:00
40:30
Mark parts lowest p% of distances as recurring content
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2000
4000
6000
8000
10000
12000
14000

**Figure 4.1:** Diagram of the described methodology. The recurring content detection process for episode 2 is portrayed.

### 4.3.1 Recurring Content

To mark a part of the video as recurring content, we do the following steps: The frames of interest are converted to feature vectors while retaining the sequential order of episodes. Then the episode is used as an image query against all other episodes, resulting in a vector with distances for the best matches. We mark the locations of the indexes where these distances are lower than the percentile $p \in \{5\%, 7.5\%, 10\%\}$ of the lowest-distances vector. These are our initial detections. Because these detections can still contain noise, we do the following post-processing steps: Detections are merged into one if they are less than 10 seconds apart; and, we only keep the two largest detections within the first 20% of the video or we keep the detection if it ends in the last 15 seconds of the video. The rationale behind this is that the recurring content that we want to detect is either in the first or last part of the video. See Figure 4.1 for a visual depiction of the described methodology and refer to Algorithm 2 for a pseudo-code implementation.

### 4.3.2 Bumpers

The methodology for detecting the bumpers within video content is different, because a bumper only contains recurring content from within the video itself. It shows shots that will be shown after the break, these shots may not reoccur in other episodes within a season. Therefore we cannot use the same methodology as described before for detection.

For the detection of bumpers we take a sliding window of 20 seconds of video frames, this will be the query. The index will consist of the remaining frames in the video, for every frame of interest the resulting distances of the two nearest neighbors will be saved and then this process is repeated for the next 20 seconds in the video all the way to the end.

After all the closest in-episode nearest neighbors are computed for each frame then from these results bumpers can be detected. We found that a global threshold for determining matches does not work well, probably because of the fact that bumpers are very short. Therefore we will use the ratio test described by Lowe [13] for deciding on whether there is a match. With the ratio test the distance between the closest neighbor and second-closest neighbor is taken into account. The ratio of the distance of the closest neighbor to the distance of the second-closest neighbor determines whether there is a correct match, if the ratio is low then there is a higher chance for the match to be correct. In [13] they classify a match to be correct if the ratio of the distances is equal to 0.8 or lower, we experimented with a ratio $r \in \{0.5, 0.6, 0.7, 0.8\}$ to find the optimal ratio for this problem. Refer to Algorithm 3 for a pseudo-code implementation.

**Algorithm 2: Recurring content detection**

```
import faiss
percentile = 5 //or 7.5 or 10

//Code for determining recurring content in episode with index i
season = T;
episode = season.get(i);
season.remove(episode);
episodeVectors = convertToVectors(episode);
seasonVectors = convertToVectors(season);

index = faiss.buildIndex(seasonVectors);
distances = index.search(episodeVectors);

//Gets the value where x% of the values will be lower than that value
threshold = getPercentileDistance(distances, percentile);

recurringIndexes = new List();
for i=0; i < distances.length; i++ do
    distance = distances[i];
    if distance < threshold then
        recurringIndexes.add(i);
    end
end

//Merge indexes as recurring content when they are less than 10 seconds
 apart
detections = MergeIndexes(recurringIndexes, 10);

//Now only return the two largest detections in the beginning of the video
//and detection that is at the end of a video
detectionsAtStart = new List();
detectionsAtEnd = new List();
for i=0; i < detections.length; i++ do
    detection = detections[i];
    if occursAtBeginOfVideo(detection) then
        detectionsAtStart.add(detection);
    end
    if occursAtEndOfVideo(detection) then
        detectionsAtEnd.add(detection);
    end
end
return getTwoLongestIntervals(detectionsAtStart) + detectionsAtEnd
```

**Algorithm 3: Bumper detection**

```
import faiss;
RATIO = 0.5 // or 0.6 or 0.7 or 0.8

//Code for determining bumpers in episode with index i
season = T;
episode = season.get(i);
episodeVectors = convertToVectors(episode);

//To get a stepsize of 20 seconds of video we need to jump (20*25)/3 indexes
//because frames are subsampled every 3 frames
stepSize = (20 * 25) / 3 ;
currentStep = 0;
inEpisodeDistances = new List();
while currentStep < episodeVectors.size() do
    query = episodeVectors[currentStep:currentStep + stepSize] ;
    index = faiss.buildIndex(EpisodeVectors[0:currentStep]);
    index.add(EpisodeVectors[currentStep+stepsize:episodeVectors.size()-1]);

    //Compute 2 nearest neighbors for all frames in the current step
    resultDistances = index.search(query, neighbors = 2);
    inEpisodeDistances.concatenate(resultDistances);
end
ratios = inEpisodeDistances[:,0] / inEpisodeDistances[:,1];
ratioIndexes = new List();
for i=0; i < ratios.length; i++ do
    ratios = ratios[i];
    if ratio < RATIO then
        ratioIndexes.add(i);
    end
end
//Return the consecutive ratio indexes
//that together are longer than 2.5 seconds
return getConsecutiveRatioIndexes(ratioIndexes, 2.5);
```

# Chapter 5

# Results

This chapter contains the results for the experiments of detecting the recurring content (recaps, opening/closing credits and previews) and the experiments of the bumper detection.

## 5.1 Recurring Content

The results were evaluated as a retrieval problem. By doing so, the measures precision and recall could be used to describe the quality of recurring content detection for the methods proposed. The methods return time ranges in the form of [t, t']. The annotations are also in this format. The true positive part is the intersection between the detected time intervals and the annotation time intervals. The false positive parts are parts from the detections that have no overlap with the annotations and false negative parts are parts from the annotations that have no intersection with the detections. To simplify the explanation of the evaluation process; it is framed as a retrieval problem. Each second of recurring content time intervals is marked as a relevant instance and if recurring content gets detected, which are true positives, then those are relevant detected seconds, under this model we can calculate precision and recall as following:

$$\text{detected relevant seconds} = \text{true positives}$$

$$\text{total detected seconds} = \text{true positives} + \text{false positives}$$

$$\text{total relevant seconds} = \text{true positives} + \text{false negatives}$$

$$\text{precision} = \frac{\text{detected relevant seconds}}{\text{total detected seconds}}$$

$$\text{recall} = \frac{\text{detected relevant seconds}}{\text{total relevant seconds}}$$

Because it is possible for the manual annotation to also be slightly inaccurate, we mark detections that are within a 2 second margin of the relevant section, to also be relevant. Figure 5.1 depicts how the aforementioned precision and recall formulas apply to the detections and ground truth of a single episode. For the calculation of the total results on all episodes, the true positives, false positives and false negatives of all episodes are summed together and then precision and recall are calculated over the whole data set. The results are shown in Table 5.1 plus the Precision-Recall curves for each recurring content class separately, in Figure 5.2.

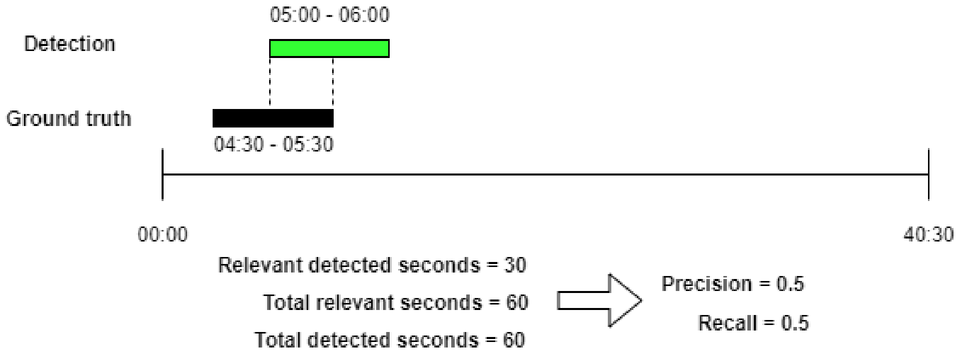

05:00 - 06:00
Detection
Ground truth
04:30 - 05:30
00:00
40:30
Relevant detected seconds = 30
Total relevant seconds = 60
Total detected seconds = 60
Precision = 0.5
Recall = 0.5

**Figure 5.1:** Visual example of the described evaluation methodology.

**Table 5.1:** Results of the recurring content detection at varying values for the percentile threshold. If a distance is below the threshold, it is classified as recurring content.

| **Method** | **P** | **R** | **P** | **R** | **P** | **R** |
|---|---|---|---|---|---|---|
| | Uniform Sampling | | | | | |
| | 5% lowest | | 7.5% lowest | | 10% lowest | |
| CH | **0.884** | 0.687 | 0.830 | 0.812 | 0.786 | 0.855 |
| CTM | 0.883 | 0.681 | 0.841 | 0.796 | 0.796 | 0.853 |
| CNN | 0.854 | 0.733 | 0.808 | 0.838 | 0.748 | **0.892** |
| | Shotboundaries | | | | | |
| CH | **0.904** | 0.529 | 0.852 | 0.638 | 0.763 | 0.707 |
| CTM | 0.856 | 0.515 | 0.830 | 0.652 | 0.771 | **0.724** |
| CNN | 0.837 | 0.522 | 0.800 | 0.635 | 0.725 | 0.713 |

## 5.2 Bumpers

Because bumpers are short segments, the bumpers were evaluated in a different manner. We do not calculate precision and recall based on numbers of retrieved relevant

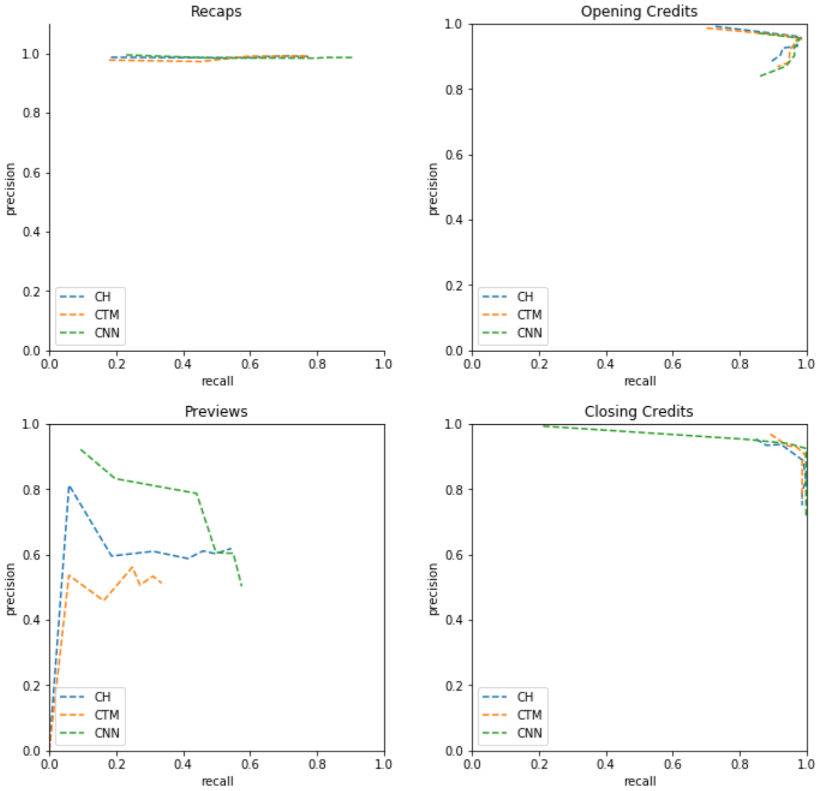


**Figure 5.2:** Precision-Recall curves for each recurring content class for the percentile values in the range [2.5, 20]. Data points that are in the upper right indicate good performance.

seconds but consider only whether a detection has overlap with a bumper or not. If a detection has overlap with a bumper, then this detection is marked as relevant. Precision and recall will thus be calculated according to the following formulas:

$$\text{precision} = \frac{\text{detections with overlap}}{\text{total detections}}$$

$$\text{recall} = \frac{\text{detections with overlap}}{\text{total bumpers in episode}}$$

Figure 5.3 depicts how the precision and recall is calculated for the bumper detection. All the results of these experiments can be viewed in Table 5.2

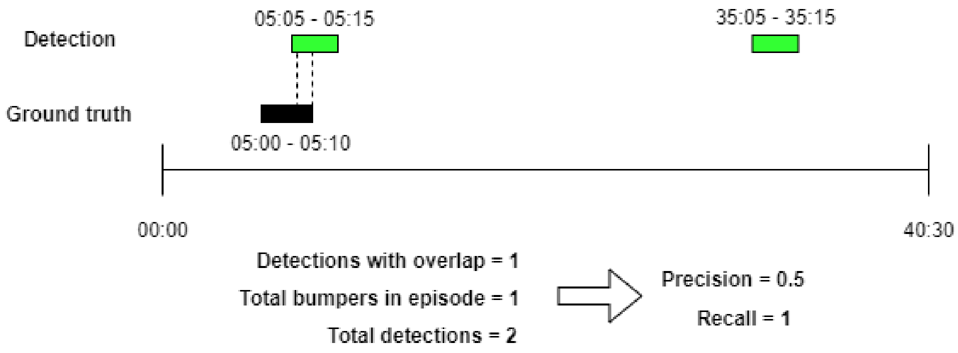


**Figure 5.3:** Visual example of the described evaluation methodology for bumper detection.

**Table 5.2:** Results of the experiments on the bumper detection for varying values of the ratio test.

| **Method** | **P** | **R** | **P** | **R** | **P** | **R** | **P** | **R** |
|---|---|---|---|---|---|---|---|---|
| | **0.5 ratio** | | **0.6 ratio** | | **0.7 ratio** | | **0.8 ratio** | |
| CH180 | 0.195 | 0.312 | 0.203 | 0.562 | 0.129 | 0.646 | 0.085 | 0.792 |
| CTM | 0.145 | 0.542 | 0.089 | 0.667 | 0.072 | 0.833 | 0.072 | 0.958 |
| CNN | **0.611** | 0.917 | 0.484 | 0.938 | 0.288 | **0.979** | 0.081 | 0.979 |

## 5.3 Discussion

The results show that the performance of the different methods for recurring content detection, varying the feature vector construction approach, do not differ much. The way the feature vectors are constructed does not show large differences when looking at the overall results. However, when looking at the scores for each class separately in Figure 5.2, it can be noticed that the CNN-method performs better on the recaps and previews retrieval tasks. Possible explanations for this could be that the CNN-method is a local descriptor as opposed to a global one, therefore being able to handle the more tough cases of recaps and previews. Another explanation could be that higher dimensionality vectors are more suited, because the CNN vectors had the highest dimensionality.

Table 5.1 presents that using shotboundaries results in lower recall scores compared to the uniform sampling of taking a frame every 3 frames. A possible explanation is that the shotboundary frames sometimes mismatch because the content shown does not consist of full shots.

Overall, the detection results are decent but not extraordinary, meaning that there is room for improvement in the methodology.

The results of the different methods for the bumper detection show different performance results. The CNN-based method shows a large difference with the other methods. An explanation for this could be that again the CNN feature vectors are local descriptors as opposed to the other two methods that are global descriptors.

This thesis also does have its limitations. Most of the experiments are performed on the first three episodes of a single season. However, an end-solution would take a whole season into account. Also, sometimes the recurring content spans across different seasons (recapping the previous season in the first episode for example), our experiments did not take this attribute into account. We only took the visual data into account by looking at video frames, not utilizing the audio data. Future studies could incorporate audio data into the methodology to see if that improves the results.

The bumper detection is based on how RTL inserts its bumpers into content. With our experiments, we did not test if our methodology also applies to the bumpers of other publishers of commercial content.

# Chapter 6

# Conclusion and Future Work

In this thesis we explored how well an image retrieval based approach could detect recurring content in a TV-show in an unsupervised manner. Experiments were performed using three different kind of feature vectors. Based upon these experiments we conclude that an image retrieval based approach definitely has potential for the unsupervised detection of recurring content in TV-Shows. The experiments showed that the three different methods all get reasonable results. Overall, it is found that the CNN-based method using uniform frame sampling has the most potential. Using this method for a metadata labeling task will still require manual human supervision. Future research is needed to improve upon the methodology or to experiment with other implementations of CNN-based feature vectors. This thesis presented novel research for the unexplored research area of recurring content detection that is focused on VOD content.

Future studies could be performed on a larger dataset consisting of full seasons of several shows to further prove the validity of this method and these future studies could try variations of the methodology and different ways of constructing the CNN feature vectors. Also the experiment could be repeated with data where the recurring content classes are more in balance to verify the results.

# Chapter 7

# Bibliography

[1] R. Lienhart, C. Kuhmünch, and W. Effelsberg, "On the detection and recognition of television commercials," 1997.

[2] J. M. Gauch and A. Shivadas, "Finding and identifying unknown commercials using repeated video sequence detection," *Computer Vision and Image Understanding*, vol. 103, no. 1, pp. 80–88, 2006.

[3] M. Covell, S. Baluja, and M. Fink, "Advertisement detection and replacement using acoustic and visual repetition," in *Multimedia Signal Processing, 2006 IEEE 8th workshop on*, pp. 461–466, IEEE, 2006.

[4] J. Wang, L. Duan, Q. Liu, H. Lu, and J. S. Jin, "A multimodal scheme for program segmentation and representation in broadcast video streams," *IEEE Transactions on Multimedia*, vol. 10, no. 3, pp. 393–408, 2008.

[5] C. Herley, "Argos: Automatically extracting repeating objects from multimedia streams," *IEEE Transactions on multimedia*, vol. 8, no. 1, pp. 115–129, 2006.

[6] Y. Benezeth and S.-A. Berrani, "Unsupervised credit detection in tv broadcast streams," in *Multimedia (ISM), 2010 IEEE International Symposium on*, pp. 175–182, IEEE, 2010.

[7] A. E. Abduraman, S.-A. Berrani, and B. Merialdo, "An unsupervised approach for recurrent tv program structuring," in *Proceedings of the 9th European Conference on Interactive TV and Video*, pp. 123–126, ACM, 2011.

[8] L. Zheng, Y. Yang, and Q. Tian, "SIFT meets CNN: A decade survey of instance retrieval," *IEEE transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 5, pp. 1224–1244, 2018.

[9] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 12, pp. 1349–1380, 2000.

[10] H. Yu, M. Li, H.-J. Zhang, and J. Feng, "Color texture moments for content-based image retrieval," in *Proceedings. International Conference on Image Processing*, vol. 3, pp. 929–932, IEEE, 2002.

[11] B. S. Manjunath and W.-Y. Ma, "Texture features for browsing and retrieval of image data," *IEEE transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 837–842, 1996.

[12] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *IEEE International Conference on Computer Vision*, p. 1470, IEEE, 2003.

[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[14] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2161–2168, Ieee, 2006.

[15] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2007.

[16] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European Conference on Computer Vision*, pp. 304–317, Springer, 2008.

[17] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *CVPR 2010-23rd IEEE Conference on Computer Vision & Pattern Recognition*, pp. 3304–3311, IEEE Computer Society, 2010.

[18] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1704–1716, 2012.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[20] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *European Conference on Computer Vision*, pp. 584–599, Springer, 2014.

[21] J. Yue-Hei Ng, F. Yang, and L. S. Davis, "Exploiting local features from deep networks for image retrieval," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition workshops*, pp. 53–61, 2015.

[22] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of CNN activations," *arXiv preprint arXiv:1511.05879*, 2015.

[23] R. W. Lienhart, "Comparison of automatic shot boundary detection algorithms," in *Storage and Retrieval for Image and Video Databases VII*, vol. 3656, pp. 290–302, International Society for Optics and Photonics, 1998.

[24] H. Shao, Y. Qu, and W. Cui, "Shot boundary detection algorithm based on hsv histogram and hog feature," in *5th International Conference on Advanced Engineering Materials and Technology*, pp. 951–957, 2015.

[25] F. Rajam and S. Valli, "A survey on content based image retrieval," *Life Science Journal*, vol. 10, no. 2, pp. 2475–2487, 2013.

[26] H. Yu, M. Li, H.-J. Zhang, and J. Feng, "Color texture moments for content-based image retrieval," in *Proceedings. International Conference on Image Processing*, vol. 3, pp. 929–932, IEEE, 2002.

[27] "Deep Learning for Java." https://deeplearning4j.org/.

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

[29] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *International Conference on Database Theory*, pp. 217–235, Springer, 1999.

[30] "Recurring content detector, implementation used for the experiments." https://github.com/nielstenboom/recurring-content-detector.

[31] "FFmpeg." http://ffmpeg.org.

[32] "Re-implementation of regional maximum activations of convolutions (RMAC) feature extractor for Keras." https://github.com/noagarcia/keras_rmac.

[33] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *arXiv preprint arXiv:1702.08734*, 2017.

[34] "Faiss, a library for efficient similarity search and clustering of dense vectors - Facebook Research." https://github.com/facebookresearch/faiss.

[35] "Blog: Faiss, A library for efficient similarity search." https://code.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/.