Using Perceptual Hash Algorithms to Identify Fragmented and Transformed Video Files

Ola Kjelsrud

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2014

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Over the last decades the amount of generated video content has increased exponentially. Easy access to video recording equipment and the Internet has given anyone the ability to create and share video material with the world almost instantaneously. With the enormous amount of content available the problem of managing it become relevant. In situations such as copyright control, media management or digital forensics there is a need to perform automatic video search.

In this master thesis we investigate this problem. Using perceptual hash algorithms we create PYVIDID, a Python based video identification system able to match and search query videos to a large database. PYVIDID can also match both fragmented and transformed video files back to its original source. We also discuss possible application areas for a content based video identification system.

Overall our results clearly shows that perceptual hash algorithms can indeed be used for video identification with high accuracy. We achieve good results regarding both accuracy and speed for both original, fragmented and transformed video files.

Abstract

I løpet av de siste tiårene har mengden generert videomateriale økt kraftig. Enkel tilgang til videoopptak utstyr og Internett har gitt alle muligheten til å lage og dele videomateriale med verden nesten umiddelbart. Med den enorme mengden av innhold som er tilgjengelig er problemet med å håndtere det blitt mer og mer aktuelt. I situasjoner som opphavsrett kontroll, mediehåndtering eller digital etterforskning er det behov for å foreta automatiske video søk.

I denne masteroppgaven undersøker vi dette problemet. Ved å bruke perceptual hash algoritmer utviklet vi PYVIDID, ett Python basert video identifikasjons system som er i stand til å søke etter videoer i en stor database. PYVIDID kan også matche både video fragmenter og transformerte videofiler tilbake til sin opprinnelige kilde. Vi diskuterer også mulige bruksområder for et innholds basert video identifikasjons system.

Samlet sett viser våre resultater tydelig at perceptual hash algoritmer kan brukes til video identifikasjon med høy nøyaktighet. Vi oppnår gode resultater når det gjelder både nøyaktighet og hastighet for både originale, fragmenterte og transformerte video filer.

Contents

Abstract . . . . **ii**
Abstract . . . . **iii**
Contents . . . . **iv**
List of Figures . . . . **vi**
List of Tables . . . . **vii**
1 Introduction **1**
1.1 Topics Covered 2
1.2 Keywords 2
1.3 Problem Description 3
1.4 Research Questions 3
1.5 Justification, Motivation And Benefits 4
1.6 Planned Contributions 4
1.7 Outline 5
2 Previous Work - File Type Classification **6**
2.1 Early Methods 6
2.2 Content Based Classification 6
2.2.1 Byte Frequency Distribution 7
2.2.2 Statistical Approach 9
2.2.3 Metrics 9
2.2.4 Principal Component Analysys 10
3 Previous Work - Video Identification **11**
3.1 Signatures 11
3.1.1 Frame Level Signatures 12
3.1.2 Temporal features 12
3.2 Image Identification 13
3.2.1 Perceptual Hashes 13
3.3 Watermarks 15
3.4 Content Based Methods 15
3.4.1 Near Duplicate Video Clip Detection 15
3.4.2 Atrasoft VideoFinder 16
3.4.3 MPEG-7 17
3.4.4 YouTube Content ID 19
3.4.5 CLIPPED 21
4 Methodology & Experimental Results **23**
4.1 Methodology 23
4.1.1 Lab Environment 27

4.2 PYVIDID . . . 27
4.3 Experiment #1 - Independence . . . 31
4.4 Experiment #2 - Robustness . . . 32
4.5 Experiment #3 - Speed . . . 34
4.5.1 Add video to database . . . 35
4.5.2 Perceptual Hash Algorithms . . . 38
4.5.3 Matching Speed . . . 40
4.6 Experiment #4 - Temporal Location . . . 41
5 Discussion . . . 43
5.1 Results Discussion - Accuracy . . . 43
5.2 Results Discussion - Robustness . . . 44
5.3 Results Discussion - Speed . . . 45
5.3.1 Add Video to Database . . . 45
5.3.2 Perceptual Hash Algorithms . . . 46
5.3.3 Match Video Against Database . . . 47
5.4 Results Discussion - Temporal Localization . . . 47
5.5 Results Discussion - Summary . . . 48
5.6 Applications . . . 48
6 Future Work . . . 51
7 Conclusion . . . 52
Bibliography . . . 53
A PyCLIPPED - Python Code . . . 56
B PyCLIPPED - Java Code . . . 61

List of Figures

1 Shows an example of CLIPPED in action 21
2 Shows the original video and the text transformation. 25
3 Shows the brightness transformations. 25
4 Shows the contrast transformations. 26
5 Shows the zoom and crop transformations. 26
6 Shows the rotation and blur transformations. 27
7 Shows PYVIDID's sliding windows matching algorithm 29
8 Shows an example of PYVIDID in action 30
9 Shows the average time used to hash a single image 30 times. 36
10 Shows the average time used to hash a single image 30 times. 37
11 Shows the average time used to hash a single image 30 times. 38
12 Shows the average time used to hash a single image 30 times. 39
13 Shows the average time used to compare a 30s video clip against each database video. 40
14 Shows the average time used to compare a 25min, 13s video clip against each database video. 41
15 Shows the performance of PYVIDID's temporal location feature with two different length query clips. 42

List of Tables

1 Lab Environment Hardware 27
2 PYVIDID Results - Independence Experiment 31
3 CLIPPED Results - Independence Experiment 32
4 PYVIDID Results - Robustness Experiment 33
5 CLIPPED Results - Robustness Experiment 34

1 Introduction

In 1878 Eadweard Muybridge created the first motion picture every made. His 16 frame footage of a horse galloping answered a popular debated question of this era. Does all 4 hoofs ever leave the ground at the same time when a horse is galloping? By using multiple cameras and assembling the images taken into a moving picture he proved that in fact all 4 hoofs did leave the ground simultaneously. Later his work and others would lead to movies, tv-shows and all other kind of video content. The rest at they say, is history.

In more recent years the amount of video content has exploded. Advances in technology, namely computers and the Internet has given everyone the ability to create and share video material with the world almost instantaneously. Anyone, either a private citizen or a large company can today possess literary thousands if not millions of hours of video content. With all this content the problem of managing it becomes relevant. There is often a need to search for a specific video, or to compare two videos against each other for matching purposes. The most common way of searching for a video file is to use concept based indexing. Using text information such as metadata and keywords which describes video files it is possible to search for particular video content. This obviously requires the information to be both truthful and detailed enough for a search string to find the correct video. Using metadata and keywords has been the de facto standard way of searching for video content for many years and is still heavily in use today. While this is both effective and efficient for locating certain video content, it relies only on metadata, and it cannot perform any detailed comparison of the video content itself.

An alternative to keywords and metadata is content based methods. Content based video identification uses the actual video content itself in order to search and match video files. In theory this should give more accurate and reliable results, since it is not possible to circumvent the system simply by changing the information about a video. The general idea for a content based video identification system is to generate a small video signature from a video file. This video signature can then be compared against a database in order to search for similar or equal content. This also has the added benefit of being able to maintain a database of all content without major storage resources. A video identification system operating in this manner would have many applications areas. Perhaps the most intuitive use would be to match a short video against the database to determine its origin.

The large amount of video material has also lead to many video sharing websites such as YouTube and Break.com. These sites allow their users to upload video content for free with little to non limitations. Running costs for these websites is often paid by commercials shown on the website itself or as a preview for an uploaded video. Video content today is almost exclusively digitally stored. As with all other properly, it has an owner. A lot of video material is therefore copyrighted

in a similar manner as other intellectual property. This gives the owner legal rights to this material, and only they can permit others to use it.

A common problem for video sharing sites is copyright infringements. Users should not be allowed to upload copyrighted material unless they have permission from copyright holders. This needs to be enforced by the uploading website, otherwise legal action from copyright holders might ensue. YouTube reported in 2013 [1] that more than 100 hours of video is uploaded every minute. Clearly this is far too much video content than any human resource can control. The only reliable way of controlling the amount of uploaded video is therefore an automatic video identification system where uploaded videos can be matched against a database containing the copyrighted material. If a match is found the video cant be blocked ensuring copyright compliance.

Easy access to video recording devices is unfortunately not only a good thing. As with most other technology it can be misused and mismanaged. For many years law enforcement has seen an increasing amount child pornography, and media stations frequency reports about police raids related to this. As with most cases within digital forensics the amount of evidence is massive and considerable human resources is required to handle it. Using a video identification system with a database of known material could both increase speed and accuracy for an investigator.

1.1 Topics Covered

The main focus of this master thesis will be content based video identification. We will investigate the possibilities for a video identification system using perceptual hash algorithms to match full length videos, video fragments, and transformed video files against a large database. In order to achieve this we will start by looking into previous research and history within the image and video identification area. We will also cover the possible application areas and performance factors for such a video identification system.

In order to determine if a file is an actual video file we will also be looking into file type classification. The ability to classify data as video content is important in fields like computer forensics where massive amounts of evidence need handling. This will help us determine possible application areas for a video identification system.

1.2 Keywords

File type classification. Video search. Content based video identification. Video fragments. Video transformations. Video identification application areas.

1.3 Problem Description

The technological revolution has lead to a massive increase in video material over the past few decades. Today, anyone can create and share video files with the whole world almost instantaneously. When the amount of content grows too large, it becomes difficult to locate and search for specific video files. Text based search has been the de facto standard way of searching for video content in large databases. Unfortunately this only relies on metadata, not the actual content itself. This leaves this method susceptible to circumvention by forging metadata. It is also not reliable for comparing video files.

An alternative solution to metadata search is to use a content based identification approach. A content based video identification system performs a search using the video content itself. A general problem within the field of video technology is therefore the process of searching and identifying video files based solely on their content. For a video identification system to be effective it should be able to match identical or similar video files back to its original source. In many situations it would also be useful to match fragmented or transformed video files back to its origin. A video identification system functioning as described above could help solve many problem within the movie and video industry. One of these is the issue of copyright control.

In order to avoid copyright infringements, video uploading websites such as YouTube requires an automatic video identification system capable of matching each new uploaded video against a previously generated database. To make matters worse, this can not be done through a simple file comparison technique. This is because it is common for users to upload both fragmented or transformed video files. A fragmented video file is a shorter clip taken from a longer video. This could be a 30s long clip contained within an hour long movie. A transformed video is a video file where the video itself has undergone certain video transformations such as colour or resolution changes. This could have happen either unknowingly or deliberately in an attempt to circumvent the system. If a video identification system could not handle both fragmented and transformed video files it would effectively be useless for this purpose since any change from the original copyrighted video would negate the system as a whole.

This master theses will therefore investigate the possibilities for a content based video identification system using perceptual hashes. The main problem investigated will be the ability of matching fragmented and/or transformed video files against a large database generated from original video content. Additionally we will look into performance factors and possible applications areas for such a system.

1.4 Research Questions

In order for us to contribute to the community related to the problem described above we created the following research questions:

1. What are the current file type classification methods?

2. What are the current video identification methods?
3. How can image identification methods like perceptual hash algorithms be extended to video identification for fragmented and transformed video files?
4. Which factors determines the performance for a video identification system?
5. What applications can a video identification system be used for?

We believe the first two questions should allow us to get an updated view on both file type classification and video identification. This information should then allow us to investigate and develop a content based video identification system for both fragmented and transformed video files. Once, this is done we should be able to test this system and discuss performance factors and possible applications areas.

1.5 Justification, Motivation And Benefits

Content based video identification becomes more and more relevant because of the sheer amount of multimedia content released. Image identification systems such as TinEye[2] and Google's reverse image search[3] is able to do content based image search. We hope to use similar techniques and extend them to a video identification system. Either you are a private citizen or a large company, you will most likely have hours upon hours of video material as your disposal. The application areas for a video identification systems is diverse. It can be used for anti-piracy protection, copyright control, media management, tracking, or monetization. In any case, the ability to automatically match query videos against a large database both quickly and accurately, could save considerable resources. To the authors knowledge there are no current free video identification systems capable to matching both fragmented and transformed video files to a database with high accuracy. Because of this, we believe investigating the possibility for such a system, the performance factors, and applications areas, could be a useful contribution for the community.

In the field of file type classification there has been a lot of work to this date [4] [5] [6] [7]. In order to make progress in this field we feel that a state of the art could be useful to both researchers and other interested parties. This would make it easier to get up to speed, and start performing the actual task at hand. It can often take considerable resources to investigate, evaluate, and understand previous work, our goal would be to speed up this process. While we do not plan to do any new research into this area, we will be using this information as a basis for possible video identification application areas. Therefore it is important for us to get a good overview of previous and current solutions.

1.6 Planned Contributions

This master thesis seeks to mainly investigate the possibility for a video identification system capable of matching both fragmented and transformed query videos against a large database. The first contribution of the thesis is a state of the art analysis of file type classification is done

in order to get an updated view on this area of research. This will enable us to discuss more possibilities for a video identification system such as searching network traffic, or dealing with a large quantity of data where automatically locating video files would be beneficial. It is important to note that when discussing video fragments in relation to video identification, we refer to shorter clips part of a longer video. This is in contrast to disk fragmented files where parts of an actual file such as a header or trailer is missing. This is however an important part of file type classification where either part of a file could be missing, or deliberately changed in order to circumvent the classification method.

The second contribution is a state of the art analysis regarding video identification. In order to investigate and develop a video identification system, we need to know previous research in this area. By first looking into image identification techniques, we can then determine if they can be extended for video identification. The third and main contribution of this master thesis is our investigation into, and development of a video identification system. By building on previous research we aim to develop a video identification system capable of matching both video fragments (different length query videos) and transformed query clips against a database with improved performance compared to previous work. Additionally we investigate and discuss performance factors and applications for a video identification system.

1.7 Outline

In Chapter 2 and 3 we present previous work for file type classification and video identification respectively. In Chapter 4 we present our methodology, PYVIDID and experimental results. We discuss these results in Chapter 5, while we present possible future work in Chapter 6. We conclude the master thesis in Chapter 7.

2 Previous Work - File Type Classification

This chapter discusses previous work in the file type classification field. While this is not directly related to video identification we use this information to discuss possible application areas for a video identification system. Therefore we believe it is important to get an updated view on previous work within this area.

2.1 Early Methods

File formats is used to describe a standard way of how information is formatted and encoded in computer files. This information is crucial for most applications and operating system. If the information is not stored as expected, it will usually generate errors and crashes when used. For the Windows operating system file types is also used for application affiliation. Each file type is affiliated with an applications which will automatically run if a certain file type is opened.

A necessary operation for computers is then the process of determining the file type of a computer file. Early attempts at this classification process was fairly simple manual approaches. Probably the most known method which in fact is still used today is filename extensions. In this method each filename ends with a . followed with a 3-4 letter code. Common examples of this is .jpeg, .png, .gif, .mpeg. This extension based method is simple and effective as long as the standards rules are followed. Unfortunately this is not always the case. The extension based method has no form of security. Anyone can at any time change the file extension of the content in a file.

Another approach called magic numbers is used by most Unix operating system variants. In this method a magic number is stored within the header of each file. This is simply a 2 byte identifier which represents the file type of the file. In order to determine the file type when encountering this magic number the operating systems simply checks the records of a magic number table. Each number will then correspond to a certain file type. Unfortunately this method suffers the same disadvantages as extension based classification. There is no security measures preventing anyone to change the magic number to an arbitrary value.

2.2 Content Based Classification

The weaknesses for file type classification methods using file extensions or header information has lead to a lot of research into this area. What seems to be the general consensus is that the only way of truly classifying the file type for a computer file is to evaluate the content of the file itself. Using the actual data of a file ensures that tampering with header or trailer information would essentially be a useless attempt at circumventing the system.

2.2.1 Byte Frequency Distribution

In 2003 McDaniel and Heydari[4] presented some of the first work towards automatic file type detection based on file content. Three algorithms were presented: Byte frequency analysis (BFA), byte frequency cross-correlation (BFC) and the file header/trailer (FTH) algorithm. All three were based on and used byte frequency distribution (BFD) in order to determine file types.

The general steps in these algorithms is to use frequency distribution in order to classify file types. Every computer file is stored as a collection of bytes. More specifically each byte consists of 8 bits representing the numbers from 0 to 255. Using this characteristic, the BFA algorithm builds a frequency distribution by counting the number of occurrences of each byte value in test files. Each distribution is then normalized in order to prevent one large file skewing the creation of a fingerprint. A fingerprint is created by averaging all of the distributions made from members of one specific file type. This fingerprint is then a representation of this file type and can be used for comparison against the frequency distribution of unknown files.

The BFC algorithm uses a very similar approach. However, instead of counting the frequency of byte values the BFC algorithm considers so called cross-correlations. These cross-correlations are occurrences of sets such as the "<" and ">" in an HTML document. By counting the number of such sets the BFC algorithm builds a frequency distribution like the BFA algorithm. Then, the same approach is used to create a fingerprint which is used for comparison against the frequency distribution of unknown files.

The value and effectiveness of these fingerprints obviously depends on the files used to generate it. It also depends on which file type it represent. Some file types consists of byte values that are very consistent for all members, while others may differ greatly. A text file will mostly consist of common ASCII characters, while a GIF file will tend to use the full range of byte values. Another important point whith these two algorithms is that they do not consider the order of bytes, only the distribution. Overall the performance of these algorithms was not good enough for any practical use, but spawned a lot more research into this area. The BFA algorithm had an accuracy of 28% while the BFC algorithm had 46%. The FTH algorithm achieved a 96% accuracy, but since this method only considers the header and trailer of a file, it can not be used for content based file type detection.

In 2005 Li[5] introduced a revamp of the BFD[4] methods by McDaniel and Heydari. Still using the byte frequency distribution, Li calculates the mean and standard deviation for each byte value. This information is then stored in a model called a centroid. Like a fingerprint each centroid can consist of data from several members of the same file type. In order to address the issue in the BFD methods of having one fingerprint to represent all members of a single file type, Li proposes to use several centroids for each file type. Li calls the collection of centroids a fileprint. Each centroid is created using 1-gram analysis of test files and consist of their mean and standard deviation byte frequency distribution. By using a clustering methodology unknown files can then be compared against these centroids and the best possible match will be chosen.

One thing to note about this method is that grouping of similar files like Microsoft Office documents such as DOC, PPT and XLS was done. While this only allows these unknown files to be classified as this group, it may be enough for some purposes like virus detection and IDS systems.

Li managed with his centroid method to greatly increase the effectiveness for the BFD approach. When considering hole files, Li achieved on average an 82% accuracy when using a single centroid and an 89% accuracy when using multiple centroids. However as mentioned some files had been grouped together into one type, such as Microsoft Office documents and the two executable formats EXE and DLL. Li also experimented with different fragments of a file (Truncation), using only the first 20, 200, 500 and 1000 bytes for comparison. The results were as expected, the more of the file used, the lower score Li got. From 99% with 20 bytes, to 89% with the hole file using multiple centroids. This is because the header of a file is given much more weight in the frequency distribution if only the first 20 bytes are considered. As discussed before, access to headers and metadata greatly increases the chances for successful classification.

Using this truncation technique has its advantages though. With less data, the computation time for both creating the centroids and comparison is greatly reduced. While Li always started at the beginning of a file, is it reasonable to assume that this can be used for any fragment, although most likely with a reduction in performance.

In 2006 Karresand introduced the Oscar method[6]. Similar to Li's[5] approach, the Oscar method uses centroids with the mean and standard deviation for each byte value. The difference is small adjustments made in the comparison phase. This Oscar method was later extended with a new metric called rate of change(ROC)[7] by the same authors. Instead of using the byte frequency distribution like the previous methods in this field, the ROC metric measures the difference in byte values from one byte to the next. This ensures that byte ordering is taken into consideration. The process of creating the rate of change metric is as follows:

- First the difference between the first and the second byte is computed.
- This process is then repeated until the end of the file.
- After each byte has been compared with each neighbour, the mean and standard deviation of this measurement is used to create a centroid in a similar manner to previous methods using the mean and standard deviation.

Because this centroid is computed from the ROC metric, it generally has better performance, especially for certain file types.

The rate of change metric was mainly created for classifying JPEG files. By counting a few specific markers, namely the existence of the byte values 0xFF and 0x00 next to each other, this method reached an accuracy of 99.2%. The authors also reports slightly better results using the rate of change metric over the standard BFD approach for other file types. With the standard BFD approach the Oscar method yielded an 87% true positive rate and a 22% false positive rate. With the ROC approach the Oscar method yielded a 92% true positive rate and a 20% false positive

rate.

2.2.2 Statistical Approach

Expanding on previous work in the classification field, Erbacher[8] suggested in 2007 that a pure statistical approach could be used to determine file types. By using a total of thirteen different statistical measurements, Erbacher argued that a skilled observer could determine the file type of unknown files of file fragments. The most influential measurements where found to be: Averages, distribution of averages, standard deviation, and distribution of standard deviation. While this work did not suggest a specific implementation for file type classification, Erbacher showed that there were indeed enough differences between many file types in order to either differentiate or determine file types.

In another paper the same year, Erbacher[9] suggested an actual implementation for this pure statistical approach. Statistical analysis data identification (SADI) uses the previous theoretical work[8] in order to implement an actual method for file type classification. SADI works by taking a block of data and performing a statistical analysis on it. Then by using the different statistics mentioned above, it compares the measurements against known file type measurements. If the data matches a known file type above a set threshold, it is considered of that type. When computing the different statistical measurements SADI also utilizes so called sliding windows. The authors found that the best values were between 256 and 1024 bytes. Less than 256 bytes and the graphs would be too obfuscated, more than 1024 bytes and the graphs would be too smooth to be distinguishable from each other. Erbacher uses window sizes of 256 bytes in his research. It is also important to note that if several window sizes are used, each window requires its own known file type measurements for comparison and would therefore need more resources in both computation and storage.

2.2.3 Metrics

Veenman[10] introduced another approach for file type classification in 2007. In his work three different metrics are used in order to determine file types. Byte frequency distribution, entropy and Kolmogorov complexity [11]. The byte frequency distribution is computed and used as mentioned before. The entropy is computed by looking at how much information the data actually contains. A string of zeros will have a low entropy, because it contains very little information. While a random string will have a high entropy because it contains a lot of information. The Kolmogorov complexity is a measurement describing how complex the string of data is. Veenman reports an overall modest 45% accuracy for this method. However, certain file types like HTML and JPEG has as usual very good accuracy.

In 2008 Calhun[12] expanded on Veenman's[10] previous work by adding several new metrics for file type classification. Most noticeably was the use of longest common substring and longest common subsequence. The general idea is that files of the same type will contain common substrings of sequences. While effective, these metrics also require a lot of computation

time. Hence they might be better for offline usage, rather than in environments such as network forensics where speed is crucial. Calhun's results is difficult to compare against others because he only reports accuracy of distinguishing between two file types, such as JPEG vs PDF. However they are promising with an average of up to 88% accuracy using a all metrics presented. What makes this work truly interesting is that his test cases where done on 1024 byte fragments only, where either 128 or 512 bytes where removed so that header information was not considered.

2.2.4 Principal Component Analysys

A new method for file type classification was presented by Amirani[13] in 2008 where the authors use principal component analysis (PCA) and unsupervised neural networks in order to extract features from test files in order to create a fingerprint. Once computed, the fingerprint can be compared against unknown files as with other methods. The authors experimented with six different file types, and reports a 98% accuracy for correctly classifying an unknown test file. While this is a very good results, it is possibly inflated. This is because all six file types is not closely related to each other, and is therefore more easily distinguishable.

3 Previous Work - Video Identification

This chapter discusses previous work in the video identification field.

Originally video identification and search has been done through metadata and keywords. This method is unfortunately only reliant on the metadata describing the video. This information could easily be forged, or simply not exist at all. Therefore content based methods was created which relied upon the video content itself. This chapter is therefore focused on content based methods since this is the basis for our master thesis.

3.1 Signatures

Video identification is currently a hot topic in computer science. Several systems has been published over the recent years with very different approaches. However, they all need to perform two basic tasks:

- Create and store a reference database consisting of information about the video files
- Search through the database by comparing a query video clip to all records in order to determine the best match.

In addition to this, features such as temporal location inside a longer video can be added. The main tasks however, introduces two important challenges for a video identification system:

- How to efficiently generate a database consisting of unique information for each video.
- How to search though the database efficiently.

As with classification this is solved by representing the video content as fingerprints, or signatures. Each video is processed, and sufficient information is extracted in order to create an unique signature for each video. Both the signature generation, and the matching operations needs to be efficient and scalable in order for a video identification system to have any value. Therefore it is important to consider what these signatures can and should contain in order to achieve good results regarding the challenges mentioned above.

Video signatures can be created at three different levels [16]. Frame, shot, and video. At the frame level, features are extracted from individual frames. At the shot level, features are extracted from the shot itself. A video shot is essentially just a sequence of continuous frames captured by a single camera. Shot boundary detection has been thoroughly researched over the years, and efficient methods has been established. At the video level, features are extracted from the entire video.

3.1.1 Frame Level Signatures

Frame level signatures is by far the most common method used for video signature generation. Features at the frame level are extracted by either keypoint based, block based, or global based methods.

Keypoint Based

Keypoint based methods usually first locates sets of key points in a frame, such as an edge or an eye. Second, a region around each keypoint is defined. Lastly the content of this region is computed into a descriptor so that it can be stored and matched against queries later. Because a frame can be treated as a single image, common image techniques such as Harris interest points, and Scale-Invariant Feature Transform (SIFT) can be used for feature extraction. This is the case for all frame level based video signatures. The keypoint based method achieves good robustness against video transformations but also requires more computation than other methods [16].

Block Based

Block based methods computes descriptors based on certain blocks or spatial regions in a frame. This is typically done by dividing each frame into an arbitrary amount of blocks and then compute the mean colour intensity of all blocks. Block based methods usually entails low computational costs, however it is also less robust against video transformations, especially scaling and rotations [16].

Global Based

Global based methods uses the entire frame to compute features used for comparison. One of the most common choices is the use of colour histograms, a representation of the distribution of the different colours in a frame. This is similar to the byte frequency distribution methods used in file fragment classification. Global features is typically the faster than both keypoint and block based methods, however is it also the method most susceptible to global changes such as colour and contrast variations [16].

Dense vs Sparse

If a frame level method has been chosen, the next decision is to choose which frames to use for signature generation. For video signatures there are two main options, dense or sparse. A dense video signature uses every frame in a video, while a sparse video signature uses only selected frames. These frames are called keyframes and is typically chosen at certain intervals, such as once every second or at the start of each shot.

3.1.2 Temporal features

Another decision for video signatures is whether it should contain temporal features or not. Normally video signatures is made up of spatial features where the only temporal information

is related to the order of which frames where analysed. While extra temporal features such as information extracted from a group of frames can help with the localization of video clip embedded in a longer video, it is not necessary. Obviously a video signature can contain both spatial and temporal features.

3.2 Image Identification

As mentioned in Section 3.1, most video signatures consist of features extracted from certain frames. Since a video is just a series of frames, techniques used for image identification, is useful in video identification as well. Identifying images based on content instead of keywords is not a new idea. Existing systems are usually referred to as reverse image search methods. TinEye [2] and Google Images [3] are examples of such systems. Both operates in a similar manner: The user submits an image in order to find similar looking images. Both solutions can handle image transformations such as object removal or resolution changes.

As image identification is a huge research area, we chose to limit this section to image identification technology used in some of the video identification systems explored in this thesis.

3.2.1 Perceptual Hashes

Perceptual hashes is one type of image signatures. Using an image as input, the perceptual hash algorithm produces an X bit hash as output. When searching for similar images, hashes from a query image is compared against a database of image hashes. Unlike a cryptographic hash, where small changes in the input leads to huge changes in the output, perceptual hashes is closely related to each other as long as the images are similar. A common property of perceptual hash algorithms is that image transformations such as scaling, aspect ratio, or colour modification only results in minor hash changes [17]. This makes perceptual hashes ideal for signatures used in image and video identification systems.

Average Hash

Perhaps the simplest implementation of a perceptual hash algorithm is the average hash function. High and low frequencies is terms used in image technology. When an image consists of high frequencies it means that the pixels in the image changes rapidly, this translates to high quality images with lots of details. When an image consist of low frequencies it means that the pixels in the image changes slowly, this translates to low quality and little details. Low frequency images therefore often only shows the structure of the image. The average hash algorithm uses low frequency images in order to create a 64 bit signature hash in the following way [17]:

1. Convert the image down to 8x8 pixels. This will remove a lot of details, and result in a low frequency image with 64 total pixels.
2. Convert the 8x8 image into greyscale. The result is now a black and white picture. This is done to simplify calculation in the next step. Instead of dealing with three colours for each

pixel (RBG), we now one value (Shade of grey, 0-255).

3. Compute the mean value for all pixels.
4. Create a 64 bit hash where each bit is set to 0 or 1 depending on whether the grey scale value is above or below the mean value. This 64 bit hash is now a signature for the original picture.

Because the average hash is computed from a 8x8 pixel image, scaling and aspect ratio will not affect the hash. Other transformations will alter the hash slightly. Comparing two image signatures can be done using normal hamming distance calculation. A distance closer to zero means a higher similarity and vice versa.

pHash

A more complex perceptual hashing algorithm is the pHash implementation. While still relying on averages like average hash, pHash uses Discrete Cosine Transform (DCT) to reduce the image down to a manageable greyscale image. DCT is a common transform used for image compression. By converting the image into the frequency spectrum, it become easier to throw away information (Compression). pHash uses the following steps to create a 64 bit signature hash [17]:

1. Convert the image down to 32x32 pixels. This is to simplify the DCT computation.
2. Compute the DCT of the 32x32 image. This results in a 32x32 collection of frequency scalars.
3. Take the top left 8x8 scalars and throw away the rest. This is done because the top left represents the lowest frequencies in the picture.
4. Compute the mean average of the DCT scalar collection.
5. Create a 64 bit hash where each bit is set to 0 or 1 depending on whether each of the 64 DCT scalars is above or below the mean value.

pHash is more robust than average hash against transformation such as gamma and colour changes. Just like average hashes, pHashes can be compared using hamming distance.

Distance Hash

A third approach to perceptual hashing is the distance function. It uses the following steps to create a 64 bit signature hash [18]:

1. Convert the image down to 9x8 pixels. This removes a lot of details and the result is a low frequency image. Like other methods, this ensures that different scaling and stretching will not affect the hash.
2. Convert the 9x8 image into greyscale to simplify further computation.
3. Compute the gray scale colour difference between each pixel next to each other in each row. Because the picture is 9x8, each row will produce 8 difference values. With 8 rows, this will yield 64 values.

4. Create a 64 bit hash where each bit is set to 0 or 1 depending on whether the left pixel is brighter than the right pixel.

As usual, hamming distance is used for comparing two images. According to tests performed by Krawetz [18] using a database consisting of 150 000 images with 3 query images: pHash is had best accuracy with zero false positives or negatives. Distance hash produced less than 10 false positives, while average hash is said to generate a huge number of false positives. When it comes to speed pHash was considerably slower than both average and distance hash. pHash used 7 hours, while average hash and distance hash used 3.5 hours.

3.3 Watermarks

Early attempts at video identification, namely to control copyrighted material was achieved using watermarks. A traditional watermark method would simply be to add a static image like a logo throughout the video. This would obviously be visible to all viewers and could possibly serve as a distraction from the actual content. This type of watermarking has been around for years, for example in the form of watermarking paper bills. Detecting illegal use of material could either be done manually or by a fairly simple algorithm looking for the watermark. Obviously manual labour is not effective in the long run, and video editing techniques could be used to remove or change the watermarks. A more modern approach is digital watermarking. This is a concept similar to steganography where a hidden signal is embedded within the actual content. Digital watermarks are often concealed as noise or random data that would otherwise be hidden in the video file in order to avoid tampering. While digital watermarks still has its uses, video identification is not limited to copyright control alone. Law-To et al. [19] also claims that digital watermarks algorithms are not yet robust enough, even for simple copyright control. Because of this there is a need for more advanced methods for video identification.

3.4 Content Based Methods

Content based methods serves as an alternative to adding watermarks to the content. In a content based method, the content itself is the watermark. Features is extracted in order to generate a signature used to uniquely represent each video file.

3.4.1 Near Duplicate Video Clip Detection

Near duplicate video clip detection (NDVC) is a content based video identification method. This is the most common name used in research literature for systems able to determine if a video is equal or similar to another. It operates in the common video identification way by comparing a query clip against a database consisting of video signatures. Near duplicate videos is defined by Shen et al. [20] as similar videos with small differences such as transformations, editing options, or content modifications. In addition to this, a NDVC system can also include temporal features to be able to detect and localize video content inside a longer unrelated video. A NDVC system can be used for many of the applications mentioned for a video identification system such as

copyright enforcement and video usage monitoring.

In 2007 Shen et al. [20] introduced a near duplicate video detection system called UQLIPS. Using two different signature models. Bounded coordinate system (BCS) which ignore all temporal information and frame symbolization (FRAS) which takes temporal information into consideration. BCS creates a single vector to represent each video clip. This is done by extracting RGB colour histograms from the video file, and summarized into a single vector. FRAS also uses RGB colour histograms, but uses this information to create a sequence of symbols representing the most dominant parts of the video. This preserve temporal information. UQLIPS was tested using more than 11000 video clips each with the duration of 60 seconds. BCS and FRAS precision rate is very similar at about 85% with 60& recall rate. However as for speed, BCS is significantly faster. BCS average search time is reported to be 50ms, while FRAS is reported to be in seconds.

3.4.2 Atrasoft VideoFinder

As mentioned above there has been a lot of research in the area related to video identification. In contrast to this fact very few video identification system implementations are available. Klinger and Starkweather developed ClipSeekr [21] which they claim is capable of identifying a video clip within a longer video stream. Unfortunately like almost all other systems availale this is only commercially available and closed source. Without buying the actual software we have no ability to perform any performance tests. OpenCV [22] is an open source library containing several hundreds of computer vision algorithms. Amongst many uses they claim the ability to performing video identification on video streams [23]. While this system actually is open source and available we could not get their example solution to run.

Another video identification system worth mentioning even though it is closed source is Attrasoft VideoFinder. This software claims to be able to perform tasks such as anti piracy, content monetization, tracking, and media management. We choose to mention this system because a demo version with limited functionality exist for testing purposes. While little to non technical information about how the system actually works has been published, we ran the demo version to investigate. Attrasoft VideoFinder seems to operate in the standard way for a video identification system. Generating a signature database, it is possible to match similar query clips against it. Generating a video signature seems to be done in the following manner:

1. Each second of video produces a collection of numbers as seen below

```
11581616247042554406154410292836791411406739585112310111041275666621
42091210288410368410367185255875068743580000161619201620410641411 51
```

```
4807400144985626538114141313461157702197193414827766650061213765161
2447754810116153126613216714649700144674113632058151026272910018108
84781749336219528312973105887738424O-6-6-6-6-6
```

2. All the corresonding collections of numbers from the same video is then used as a video signature.

How these video signatures are match against each other is not published. But it is reasonable to assume that some kind of string matching is used. When it comes to performance, VideoFinder claims the following results from its whitepaper:

- Signature generation speed: It takes 30 minutes to generate a video signature from an 1 hour long video file.
- Matching speed: It takes 1 second per hour of video matched against the database, plus 20 second overhead.

Seeing as we only had access to a demo version our tests was fairly limited. With the included video material from Attrasoft we did achieve results close to this. We used this material because only certain file formats and encodings was possible. However, we suspect that with large video files, especially HD material both signature generation and search times would increase drastically.

3.4.3 MPEG-7

MPEG-7 is a multimedia standard from Moving Picture Experts Group (MPEG). Unlike previous standards like MPEG-4 which deals with multimedia encoding, MPEG-7 is a content description standard. This means that MPEG-7 deals with information (Metadata) about the content, not the content itself. The most relevant part of the MPEG-7 standard to video identification is the video signature tools. This part of the standard describes how to construct a unique video signature used in video identification. The obvious advantage of a standard versus a specific implementation is that it can be used by many different systems. This also allows for signature sharing (Database) between applications. The MPEG-7 video signature tools was developed with 11 important requirements [16]. These requirements had to be met by the video signature in order for the standard to be accepted.

1. Unique. Each video should have its own unique signature.
2. Robust. Video transformations such as colour changes should not result in massive signature changes.
3. Independent. False positive rate should not be above 5 parts per million.
4. Fast signature matching. A query clip should be matched against at least 1000 clips per second on a standard computer.
5. Fast signature generation. There were no set requirement for signature generation. However a MPEG-7 standard implementation is able to extract features from 900 frames each second

on a standard computer.

6. Compact. The video signature should not be larger than 30 720 bits for each second of video.
7. No content modification. The video content used to create a signature should not be modified in any way.
8. Self containment. Each video signature should be enough to represent the video content. No access to the actual content should be necessary.
9. Coding independence. The creation of a video signature should not depend on the encoding of the video content.
10. Partial matching. It should be possible to detect if a query clip is part of a longer video.
11. Accurate temporal location. Localization of query clips inside a longer video should be accurate to a minimum of 2 seconds.

Signature

The MPEG-7 video signature is a dense frame level video signature. A signature for a video consists of two parts [16]: 1) Fine signatures extracted from each video frame, and 2) Coarse signatures extracted from sets of fine signatures. The reason for this extra information in the coarse signature is to speed up the matching step as we will see below.

The first step in creating a video signature is to create the fine signature part. One fine signature is created from each frame and consists of three parts. 1) The frame signature itself, 2) A summary of the frame signature, and 3) A frame confidence level. The frame signature is created for each frame by using average image intensities and differences. The summary is just an approximation used for faster coarse signature computation in the next step. The confidence level is an average of the differences calculated for a frame. Each fine signature uses 656 bit for each second of video.

The second step in creating a video signature is to create the coarse signature part. This is extracted from the videos fine signatures. Each coarse signature is created from 90 fine signatures starting from the first frame. This is done with an overlap of 45, making the first coarse signature from frame 1-90, the second from frame 45-135, and so on. Using the summary part of the fine signatures, each coarse computation takes this information and creates 5 occurrence histograms. Each coarse signature uses 810 bit for each second of video. The last step before storing the video signature is a compression step able to shrink the video signature 27% for better storage optimization. The result is a complete video signature using 5532 bit for each second, thus using 2.5 MB per hour of video.

Signature Matching and Performance

The MPEG-7 signature tools standard only describes how to create a video signature. No mandatory matching scheme is set. A possible implementation of MPEG-7 video signature matching is

presented in [16], this method was also used for evaluating the different standard revisions of MPEG-7. Using 3 steps, the implementation first compares the coarse part of a query signature against the database in order to identify possible candidates. Then for selected candidates the fine signature is used to identify candidate parameters used in the next step. The last step is the actual frame by frame matching where the query signature is compared to every possible temporal location in the database signature. The result is the best possible match in the database with the best temporal position within this video.

The authors of [16] carried out extensive testing on the MPEG-7 video signature. An independent test was used to ensure the standard complied to the false positive requirement of no more than 5 parts per million. Using a database of 1900 3-minute clips (95 hours) and 70 000 30-second query clips made from the database clips, each query clip was compared against every database clip. This ensured that more than 130 million comparisons was done, and a correct false positive estimate could be determined. The MPEG-7 passed this requirement.

The other test was to ensure the robustness requirement. Using a database of 545 3-minute clips (27 hours) and 70 000 30-second query clips made from the database clips, each query clip was compared against every database clip. The query clips where subject to different types of video transformations, and each transformation where done in 3 levels. Light, medium and heavy modification. The test where conducted for both direct, and partial matching. In direct matching the whole query video matches a part of the database video, In partial matching the query video only matches parts of the database video. Overall, MPEG-7 accuracy was 95.49%, where light modifications performed better than heavy modifications. Direct matching also performed better than partial matching with an average of 97.54% vs 93.43%.

3.4.4 YouTube Content ID

While video identification is a popular research area, relatively few systems has been implemented and released. Probably the most used video identification system today is YouTube Content ID. YouTube is currently the most popular video sharing website with users uploading more than 100 hours of video every minute [1]. With the vast amount of video uploaded to the site, YouTube eventually got into legal trouble when users started to uploaded copyrighted material owned by others. Until the introduction of Content ID, the only way to detect such copyright infringements were by manually reporting each case.

With YouTube Content ID, every video uploaded to YouTube is scanned against a database of signatures. In addition to scanning new videos, Content ID also periodically scans older content. Every day, 400 years of video content is scanned [24] by YouTube Content ID. In the YouTube Content ID system, there are three different actors:

1. YouTube Content ID. Creates the database and performs matching.
2. Content owners. Uploads content to be added to the database.

3. Users. Uploads videos to YouTube.

Because YouTube is a free service, everyone is able to become a video uploader. However, to be recognized as a content owner, and have the opportunity to add your content to the database, YouTube has some requirements. A content owner has be to able to claim exclusive rights to the content added to the database. In addition to this, they need to sign an agreement with YouTube officially becoming a Content ID partner.

Once a content owner has been recognized, YouTube Content ID operates like a typical video identification system described above. 1) Content owners uploads a low resolution of their copyrighted material. 2) Features are extracted from uploaded material in order to create a signature for each video. This signature is then added to the database. 3) When users upload new videos, a signature is created the same way, and matched against the database. When an uploaded video matches a video in the database, a decision is made. YouTube Content ID offers three different options for content owners:

- Block. This option either blocks the video completely or partially. Some content owners may wish to allows the video to be seen by certain countries for marketing reasons.
- Moneytize. Moneytize is the most common option. All add revenue goes directly to the content owner, and not the uploader.
- Track. The track option leaves the video unaffected. However detailed statistic will be given to the content owner about when and where the video is most popular. This is an increasingly popular option because it can help content owners to choose where to publish similar content in the future.

The matching step in YouTube Content ID is extremely quick. Only a few minutes is required to search through the database. If someone is familiar to uploading videos on YouTube, this is the time it takes from the video is uploaded until the video is published. While YouTube claims the Content ID system is resistant to video transformations, and other video editing techniques, we have only found test data concerning the audio identification part, not the video functionality.

The idea of a video identification system used to control copyrighted materiel is widely accepted by most of the community. However, YouTube has received criticism for their implementation. In a recent Forbes article [25] two weaknesses were revealed. We mentioned above that there were certain requirements in order to become a content owner. Unfortunately the process of verifying partners was possible to circumvent, leading to false partners claiming the work of others and benefiting from the moneytize option. The other weakness was the ability to separate similar content. This is especially true for video games, where sometimes the only difference is the commentary or the music. This lead to a massive block of gaming videos because Content ID wrongly tagged it for copyrighted material. Because YouTube Content ID operates in a guilty until proven otherwise fashion, the only way to reverse a claim is by manually submitting a dispute. This is time consuming and in the mean time, the actual content owner is loosing potential revenue.

3.4.5 CLIPPED

In 2013 Gardåsen [26] developed a video identification system called CLIPPED. Using a query clip, CLIPPED is able to search a database for a matching video. CLIPPED performs near duplicate video clip detection for both same and different length video clips, hence it is also able to determine if query clip is part of a larger video. After searching through the database, CLIPPED reports the best possible match.

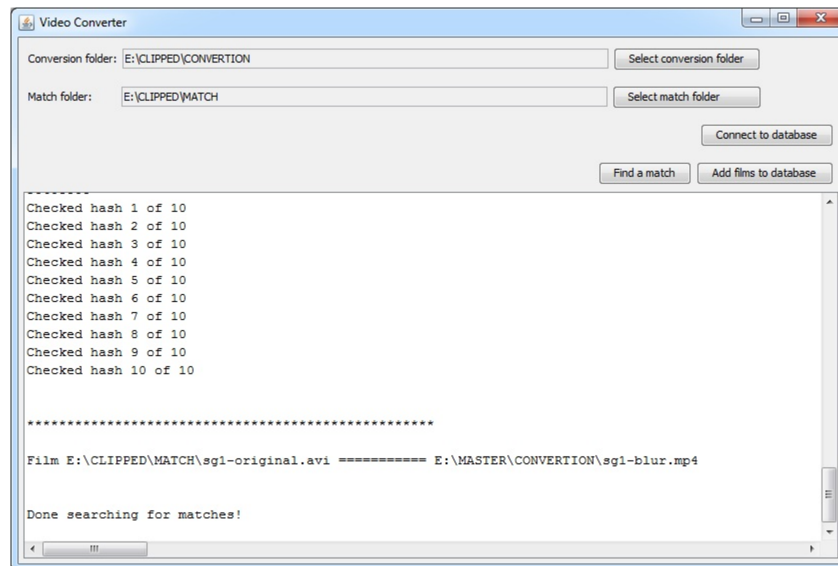

Figure 1: Shows an example of CLIPPED in action

Figure 1 shows CLIPPED performing a search using a query clip named "sg1-origional.avi". CLIPPED reports the best possible match is "sg1-blur.mp4". In this case, the database contained 10 different video files.

CLIPPED is implemented in Java, and uses the HyperSQL database to store video hashes. As other video identification systems it performs two main tasks. 1) Adding video signatures to the database, and 2) Searching for a query clip.

Signatures

In order to create a unique signature for each video, CLIPPED uses a global and sparse signature type. Each video is converted into images, 1 image for each second of video. This is done using ffmpeg [27], a multimedia library. Each image is then hashed using a perceptual hashing (pHash) algorithm, producing a 49 bit hash. CLIPPED uses the java library ImagePHash consisting of a pHash algorithm created by Elliot Shepherd. All the image hashes is then stored in the database as a signature for that particular video. The result is a signature consisting of 49 bits for each second of video. If we discard the small amount of metadata, the video signature uses 0,02 MB per hour of video.

Database

CLIPPED uses the HyperSQL relational database for signature storage. This was chosen because of its simple integration with Java, and because the database is stored in a single text file, making it easy in a testing environment. Gardåsen choose to store signatures in two tables consisting of the following values:

1. NAME_URL:

PK	Name	Url

2. FRAME_HASH:

Frame	Hash	Film

Name_URL contains metadata about the video, and FRAME_HASH contains the signatures itself.

Matching

When searching for a match, CLIPPED uses the same method as described above in order to create a signature for the query clip. This signature is then compared against all the signatures in the database. For each query clip, every video in the database is checked. If the query clip is of same length as the current database video, this comparison is a trivial one to one mapping. However, if the query clip is shorter, the matching algorithm has to check the query clip against all possible positions in the database video. Because each signature is a collection of hashes, and each hash is a 49 bit string, Hamming distance is used as a comparator. The sequence of hashes with the lowest average Hamming distance to the query clip is reported as best the possible match.

Gardåsen tested CLIPPED with a database consisting more than 140 hours of video. The content of the database was a mix of full length movies, and shorter tv show. Downloading 9 query clips from YouTube from said movies and tv shows, Gardåsen reports a 100% true positive rate. Query clip length varied upwards from 3 minutes to full video length.

4 Methodology & Experimental Results

The first part of this chapter describes our methodology used for testing content based video identification systems. We describe which experiments were done, how they were done and what dataset we used to conduct them. Then we describe why and how we implemented PYVIDID, our video identification system based on perceptual hash algorithms able to match both fragmented and transformed video content against a large database.

In the second part of this chapter we show the results of our experiments.

4.1 Methodology

While video identification is a popular research area in computer science, few actual implementations are available. Most like YouTube Content ID [28] are either closed source, only commercially available or not published at all. While this is the general status there are exceptions such as a demo version of Attrasoft Video Finder [29]. Another problem when evaluation video identification systems is they where all designed with different purposes in mind. Some were created only to perform near duplicate video clip detection for similar length video clip, while others perform operations such as temporal localization of video clips embedded in a longer video. Having different applications areas will determine which functions is allocated more resources under development and consequently performed better in a finished product.

The performance tests in this section is mainly designed for CLIPPED and PYVIDID since they are the only two video identification systems we had full access to. There were primarily 4 main criteria we wanted to conduct experiments to determine.

1. Independence - How unique is each video signature.
2. Robustness - How is the video identification system able to handle common video transformations either knowingly or unknowingly used to circumvent the system.
3. Speed - How fast is signature generation and signature matching.
4. Temporal localization accuracy - How accurate can the video identification system determine a query clips temporal position in a longer video.

Dataset

The dataset used in the experiments is a large collection of various video files. This ranges from full length movies, tv-shows, and soccer games to shorter 30 and 5 second clips extracted from various video sources. Video encoding and bitrate is also highly variable. This variance was chosen in order to best simulate a real world scenario, where users most likely would use video

identification on various video files.

The video signature database used in our experiments contained ∼500 hours of video. This is definitely on the higher end of the scale compared to previous research in video identification. In order to conduct the experiments we also needed query clips. In addition to using the full length videos as query clips we extracted ∼100 shorter query clips from videos in the database. We choose to use the following three length query clips for our experiments:

- Full length clips
- 30 second query clips
- 5 second query clips

Using these three different length query clips allowed our experiments to reveal how accuracy for the video identification would vary depending on query clip length.

Transformations

For the robustness test we modified the query clips with various video transformations. We choose a combination of transformation based on previous work and added some of our own. Transformations was added because they can either be a result of normal video editing, or deliberate attempts to circumvent the video identification process. In any case, the robustness experiment should indicate if the video signature can handle such adjustments. Using the transformations below we aim to best simulate a real world scenario, although in most cases to the extreme, as shown in the brightness example. Just as a curiosity we also created a reverse version of the 30s query clips. We did this to see if the video identification could somehow identify correct source video even for such a massive change.

Text

The original video and the text transformation is shown in Figure 2. The left picture shows the original video clip with no transformation performed. The right picture shows the text transformation created by adding a static text throughout the whole video clip.

Figure 2: Shows the original video and the text transformation.

Brightness

The brightness transformations are shown in Figure 3. The left picture shows the brightness turned down by 25%. The right picture shows brightness turned up by 25%.

Figure 3: Shows the brightness transformations.

Contrast

The contrast tranformations are shown in Figure 4. The left picture shows the contrast turned down 25%. The right picture shows contrast turned up by 25%.

Figure 4: Shows the contrast transformations.

Zoom and Crop

The zoom and crop transformations are shown in Figure 5. The left picture shows a 25% zoom. The right picture shows a 25% crop by adding a black border.

Figure 5: Shows the zoom and crop transformations.

Rotation and Blur

The rotation and blur transformations are shown in Figure 6. The left picture shows a rotation of 10 degrees to the right. The right picture shows a 1% blur effect.

Figure 6: Shows the rotation and blur transformations.

4.1.1 Lab Environment

All experiments conducted in this master thesis was performed on the hardware shown in Table 1. By installing a fresh version Windows 7 with only the necessary software we believe our results are both correct and repeatable with a similar setup.

CPU	Intel Quad Core i5-3570 3.5Ghz
Motherboard	Gigabyte GA-Z77X-UD3H Z77
HD	Samsung 830 128GB SSD
RAM	Corsair 16GB DDR3 1600Mhz
GPU	Nvidia GeForce GTX 670

Table 1: Lab Environment Hardware

Developing PYVIDID was done using Spyder [30] an open source IDE. Running the Python code was done with WinPython [31], a free portable Python interpreter. WinPython allowed us easy management of Python libraries such as the ImageHash moodule containg the Perceptual hash algorithms. For development and modification of Java code we use the Eclipse [32] IDE.

4.2 PYVIDID

We performed extensive tests on Gardåsen's video identification system CLIPPED with impressive results. Especially considering the fairly simple approach to video identification compared to previous work in the area. CLIPPED achieved considerably better results with substantial less resources regarding computation, storage, and technology. While the overall idea behind CLIPPED was great, the software itself was a proof of concept implementation. This meant the software contained some bugs, but more importantly left us wanting for more functionality like temporal localization and more options regarding signature generation. In an attempt to speed up video signature generation, Gardåsen had developed a function using threads to hash images. This led to garbage collection errors when trying to delete images after hashing which caused the system

to get stuck in a loop, sometimes indefinitely. Using the non threaded option while slower, solved this problem. More importantly than minor bugs, were the potential for improvements, especially regarding speed and results reporting. We also wanted other options for testing purposes. Because of this we decided to develop our own video identification solution called PYVIDID.

PYthon VIDeo IDentification (PYVIDID) is our Python implementation of a video identification system. PYVIDID is heavily based on Gardåsen's CLIPPED [26], and basic operations such as video signature generation and matching is very similar. In addition to the experimentation benefits of having two systems to compare, we developed PYVIDID instead of adding functionality to the existing CLIPPED for the following main reasons:

1. We wanted to add additional functionality and flexibility to video signature generation by adding several different perceptual hash algorithms and hash sizes. Python offered us these options using the ImageHash [33] library.
2. We wanted to add temporal localization of a query clip embedded in a longer video. Meaning if the query clip is part of a larger video in the database, this should be detected and the best possible location within the video should be reported.
3. We wanted to determine if using another programming language and libraries would affect the performance, both regarding accuracy and speed.
4. Lastly we wanted to add additional modularity to the application for future development. We use this flexibility to develop matching algorithms in both Python and Java.

As PYVIDID was only developed for testing purposes we did not add any GUI to the application. This led to a relatively small amount of code, convenient for experimentation. All PYVIDID source code is added in appendix A and B.

Signatures

PYVIDID operates in a similar manner to Gardåsen's CLIPPED regarding video signatures. First we use ffmpeg [27] to convert a video into a collection of images. We chose the rate of 1 image per second of video. However, PYVIDID would easily be adaptable to different rates. Then, each image is hashed with a perceptual hash algorithm producing a 64 bit hash. The result is a video signature consisting of 8 bytes for each second of video. Discarding the small amount of overlay for signature storage, PYVIDID produces on default settings a video signature consisting of 0.027 MB per hour of video.

While basic operations is similar to CLIPPED, PYVIDID offers extensive options regarding video signature generation. In order to achieve this we use the Python ImageHash [33] library written by Johannes Buchner. This library offers the use of the three different hash functions described in Section 3.2.1:

- Average hash.
- pHash.

- Distance hash.

The library also offers the option of changing the hash size from the standard 64 bit to 128 and 256 bit. This is in contrast to the Java ImagePHash library which only contained the pHash function producing a 49 bit hash. As each algorithm has its own advantages and weaknesses, PYVIDID offers a more complete video identification system usable in different scenarios.

The options for video signature generation is simply stored in global variables for easy configuration under testing. This would also give a possible GUI easy access to change values depending on user input. For video signature storage PYVIDID uses the SQLite database. Table structure is similar to CLIPPED except the removal of the URL field for better performance, as this is never used. SQLite stores the whole database in a single text file, making it ideal for testing purposes. While the single database file is a beneficial feature for testing, we realize that a more advanced database like MySQL likely would achieve better performance in a larger setting. PYVIDID is developed with this in mind, as each function is to an extent modular. Changes to for example database storage will not affect signature generation or signature matching, as long as the SQL language is supported.

Matching

In order to match a query clip against the video database, PYVIDID generates a video signature for the query clip in the same manner mentioned above. The resulting video signature consisting of a 64 bit hash per second of video is then matched against every video signature in the database. This is done using a sliding window technique where every possible query clip position is checked against every possible position for all videos in the database.

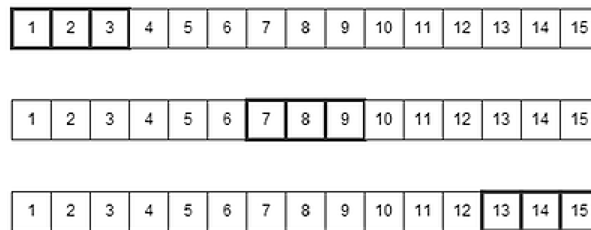

Figure 7: Shows PYVIDID's sliding windows matching algorithm

Figure 7 shows this sliding window technique. Each number represents a 64 bit binary hash. In this example a 3 second query clip is compared against a 15 seconds long database video. For each window the hamming distance is calculated between each hash, and averaged. The lowest average is then the best result for this video and stored. Generally for a query clip of length Q, and a database video of length D, the amount of positions P compared for each video is:

$$P = D - Q$$

This sliding window ensures that PYVIDID will not only report the best possible match for same length videos, but also for different length video clips. In addition we also store the best possible temporal location computed in every video comparison, allowing us to report this information to the user.

```
0            1 -> 34    = sg1-reverse.mp4
843          1 -> 34    = sg1-rotation10.mp4
827          1 -> 34    = sg1-text.mp4
884          1 -> 34    = sg1-zoom-1024.mp4

Best Match Average: 0
sg1-reverse ==  sg1-reverse.mp4
Temporal: 1-34
```

Figure 8: Shows an example of PYVIDID in action

Figure 8 shows an example of a PYVIDID video clip search. In this case the database consist of 4 different video files. The query clip used is called "sg1-reverse". The first 4 rows represent matching the query clip to every video in the database. The first column reports the average hamming distance between the query clip and the current database video. The second and third column represent the best temporal location within each video. In this case every video in the database as well as the query clip was 34 seconds long, and the results shows that. The last column show which video the query clip is currently being matched against. At the end, PYVIDID reports the best possible match with best average hamming distance, video name, and best temporal position within that video.

After performance testing on PYVIDID we found that our Python matching algorithm was a lot slower than Gardåsen's CLIPPED. While we did add additional features, both systems makes use of a sliding window technique using hamming distance. These results and possible causes are respectively shown and discussed in Section 4.5 and 5. Because of this, we decided to develop a Java version of the matching algorithm for PYVIDID. Again PYVIDID's modularity allowed us to add/replace this functionality with ease.

The Java version of PYVIDID's matching algorithm operates in an identical manner to the Python version. In order to compare a query clip against the database using the Java matching version,

PYVIDID generates the query video signature and stores is in a text file. This signature file is then read by the Java version, and matched against the database. Just like Python, Java can access the database using SQLite. Although the two versions are more or less identical, the Java version performs a lot better.

4.3 Experiment #1 - Independence

Arguably the most important task for a video identification system is to correctly match a query clip to its origin with high accuracy. This would allow users to identify the most likely source video the clip was taken from. Most video identification systems generates video signatures for this purpose. An important property for video signature is therefore their uniqueness. If the source of a query video was mistakenly identified, it would effectively make the video identification system useless. This could happen if the video signatures used want not unique. Because of this it is important for both similar and different video content to produce a unique signature.

In order to determine the uniqueness for the video signatures used by PYVIDID and CLIPPED we performed the independence experiment. We wanted to measure the true positive and false negative rates for both systems. In order to make this measurement we used 3 different length query clips:

1. Video 1: 20x full length videos already located in the database. Length varied from 25 to 107 minutes.
2. Video 2: 20x 30s video clips taken from a random video in the database.
3. Video 3: 20x 5s video clips taken from a random video in the database.

These query videos were simply taken from a random video in the database using Sony Video Vegas 11. Every query video was taken from its original source and stored as a MPEG-4 (mp4) file. This should not affect the experimental results as no video transformation was applied. Every query video was then matched against our database containing ∼500 hour of video. There were two possible outcomes for this experiment: Match or No-Match.

As we had 20 query videos for each length, this experiment was repeated 60 times for each video identification system, 120 in total. The results from the independence experiments for PYVIDID and CLIPPED is shown in Tables 2 and 3.

Query Length	Match	No-Match
Full Length	100%	0%
30 second	100%	0%
5 second	95%	5%

Table 2: PYVIDID Results - Independence Experiment

Query Length	Match	No-Match
Full Length	100%	0%
30 second	100%	0%
5 second	95%	5%

Table 3: CLIPPED Results - Independence Experiment

As shown in Tables 2 and 3, both PYVIDID and CLIPPED achieved a 100% true positive rate for both full and 30s length query clips. Using a 5s query clip resulted in a 95% true positive rate for both systems. It is important to note that all query clips were present in our database, either as a full length video, or part of a larger video.

While the query clips were created at random, we made sure none of them contained common parts such as intros or tv commercials. In order to get a complete picture, we also repeated the experiment with query clips common in several videos. When experimenting with this case, both PYVIDID and CLIPPED will always report the first best hit as the best result, even if several database videos produces the same hamming distance measurement. This scenario was one of the reasons we improved the result reporting in PYVIDID. While CLIPPED only report the best match, PYVIDID will by default report all match results. This can easily be modified to only show the top 5 results or simply all better results than a certain threshold.

4.4 Experiment #2 - Robustness

The independence experiment indicated the true and false positive rates for the video identification systems. It is important to note that none of the query clips used in that experiment contained any kind of video transformations. While a video identification system has many application areas, one particularly important use is copyright control. The overall goal for these systems is to ensure that copyrighted material is not being misused by anyone without the proper rights. In an effort to circumvent these kind of systems users may make minor changes to the material. With video files, this is often done with certain video transformations such as contrast, brightness, or scaling changes. Armed with this trick, users can circumvent the copyright control systems and make them ineffective, sometimes even useless.

The robustness experiment was therefore conducted in order to determine how robust video identification systems is to certain video transformations. This, in addition to the independence experiment was arguably the most important part of our evaluation because together they indicate how well a video identification system can detect both simple video copies, and elaborate attempts at circumvention. One example for such a dual purpose system is YouTube Content ID [28], were the ability to detect both cases is crucial. If a copyright infringement is detected, a decision such as blocking or monetization of content can be applied.

For the robustness experiment we extracted 10 30s video clips from videos already contained

within the database. Then we performed the video transformations shown in Section 4.1 on each 30s video clip. The results was 10 sets with 10x 30s video clips with various transformations. Each set contained the following 30s video clips:

1. Video 1: Original video clip. No transformation.
2. Video 2: Text transformation. Text added throughout the while video clip.
3. Video 3: Brightness transformation. Brightness -25%
4. Video 4: Brightness transformation. Brightness +25%
5. Video 5: Contrast transformation. Contrast -25%
6. Video 6: Contrast transformation. Contrast +25%
7. Video 7: Zoom transformation. 25% zoom.
8. Video 8: Crop transformation. 25% crop.
9. Video 9: Rotation tranformation. Picture rotated 10 degrees to the right.
10. Video 10: Blur transformation. Picture 1% blurred.

This meant that each transformation had 10 different video clips matched against the database. In total 100 query clips was matched against the database in this experiment. The results for PYVIDID and CLIPPED respectively is shown in Tables 4 and 5.

Video Transformation	Match	No-Match
No transformation	100%	0%
Text transformation	100%	0%
Brightness -25%	100%	0%
Brightness +25%	100%	0%
Contrast -25%	100%	0%
Contrast +25%	100%	0%
Zoom 25%	100%	0%
Crop 25%	30%	70%
Rotation 10 degrees	90%	10%
Blur 1%	100%	0%

Table 4: PYVIDID Results - Robustness Experiment

Video Transformation	Match	No-Match
No transformation	100%	0%
Text transformation	100%	0%
Brightness -25%	100%	0%
Brightness +25%	100%	0%
Contrast -25%	100%	0%
Contrast +25%	100%	0%
Zoom 25%	100%	0%
Crop 25%	10%	90%
Rotation 10 degrees	70%	30%
Blur 1%	100%	0%

Table 5: CLIPPED Results - Robustness Experiment

As mentioned above we created a reversed version of the 30s original query clips as well. This was purely done by curiosity, we wanted to match these clips against the database to determine if it somehow could handle such a enormous transformation. This was not the case. From the 10 reversed query videos, only 1 was able to match correctly back to the source. And from looking at the hamming distances this seemed more random than anything else. Match and no match levels were almost equal with little to non distinction.

4.5 Experiment #3 - Speed

Speed is an important performance factor for a video identification system. In production environments, databases will likely contain thousands if not millions of hours of video. Therefore, both adding and matching a video to the database must be an effective process. A prime example of this is YouTube Content ID [28], where over a 100 hours of video is matched against the database every minute. In order to measure the speed performance of PYVIDID and CLIPPED we performed the following three experiments:

1. Time used to add a video to the database.
2. Time used to hash a single image by each perceptual hash algorithm.
3. Time used to match a query video against the database.

In order to take these measurements we used Python and Java time libraries. This gave us access to time related functions allowing us to measure time elapsed in a simple way. Both programming language libraries claims to be accurate even for multi core processors. Since we had full access to the source code for both video identification systems, we easily modified it for these experiments. While each experiment required a different approach concerning code placement, all three were pretty much the same. In the following example we measure the time it takes for PYVIDID (Python) to hash a single image using the pHash function:

```
import time                                                                    #Import
    time library
start = time.time()                                                      #Records start time
```

```
imagehash.phash(Image.open(image), hashSize) #Hashes image
print(time.time() - start)                                          #Prints hash time
```

First we import the time library. Then we save the current time, run the perceptual hash algorithm and finally report the time elapsed. In Java, we used the System.nanoTime() function in a similar manner.

4.5.1 Add video to database

Both CLIPPED and PYVIDID uses 3 general steps to add a video to the database. First the video is converted into images. Then, a perceptual hash algorithm is used to hash all images producing a collection of hashes. Finally, this collection of hashes is stored in a database as a video signature. We wanted to measure the time it took to add different sized and length video files to the database. For this experiment we chose 4 different sized and length video files:

1. Video 1: Length: 30s. Size 40MB.
2. Video 2: Length: 42min 11s. Size 355MB.
3. Video 3: Length: 82min 1s. Size 714MB.
4. Video 4: Length: 107min 40s. Size 3658MB.

These video files was chosen to give us an overall indication on which video identification is faster. Additionally, it should indicate which factors affects adding time and in which way. This experiment was conducted by adding each video to each video identification system 3 times. These measurements was then averaged. The measurements for PYVIDID was plotted into Figure 9, while the measurements for CLIPPED was plotted into Figure 10.

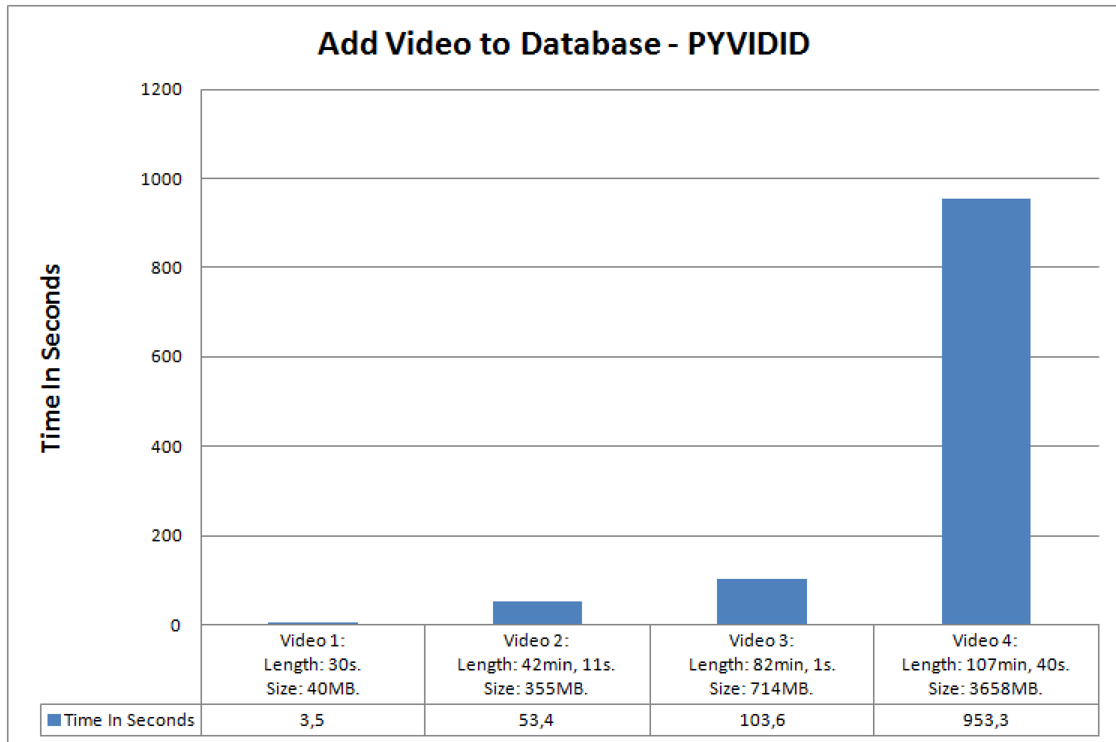

Figure 9: Shows the average time used to hash a single image 30 times.

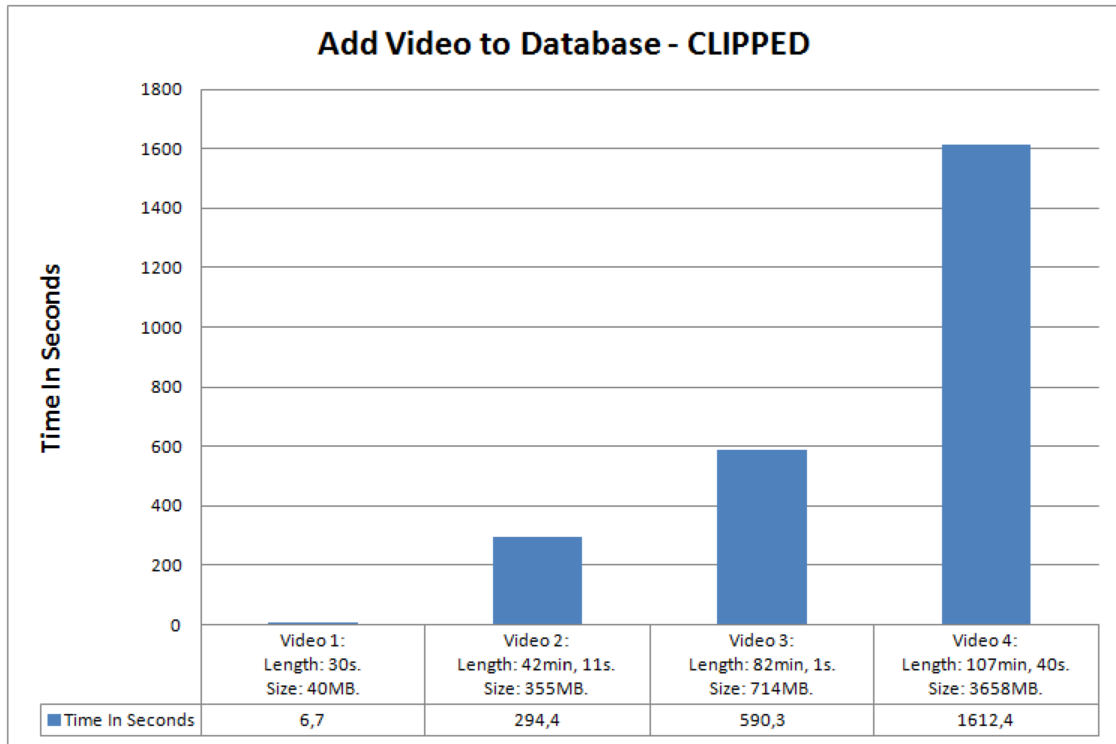

Figure 10: Shows the average time used to hash a single image 30 times.

For easy comparison we also added all results into on graph in FIGURE 11.

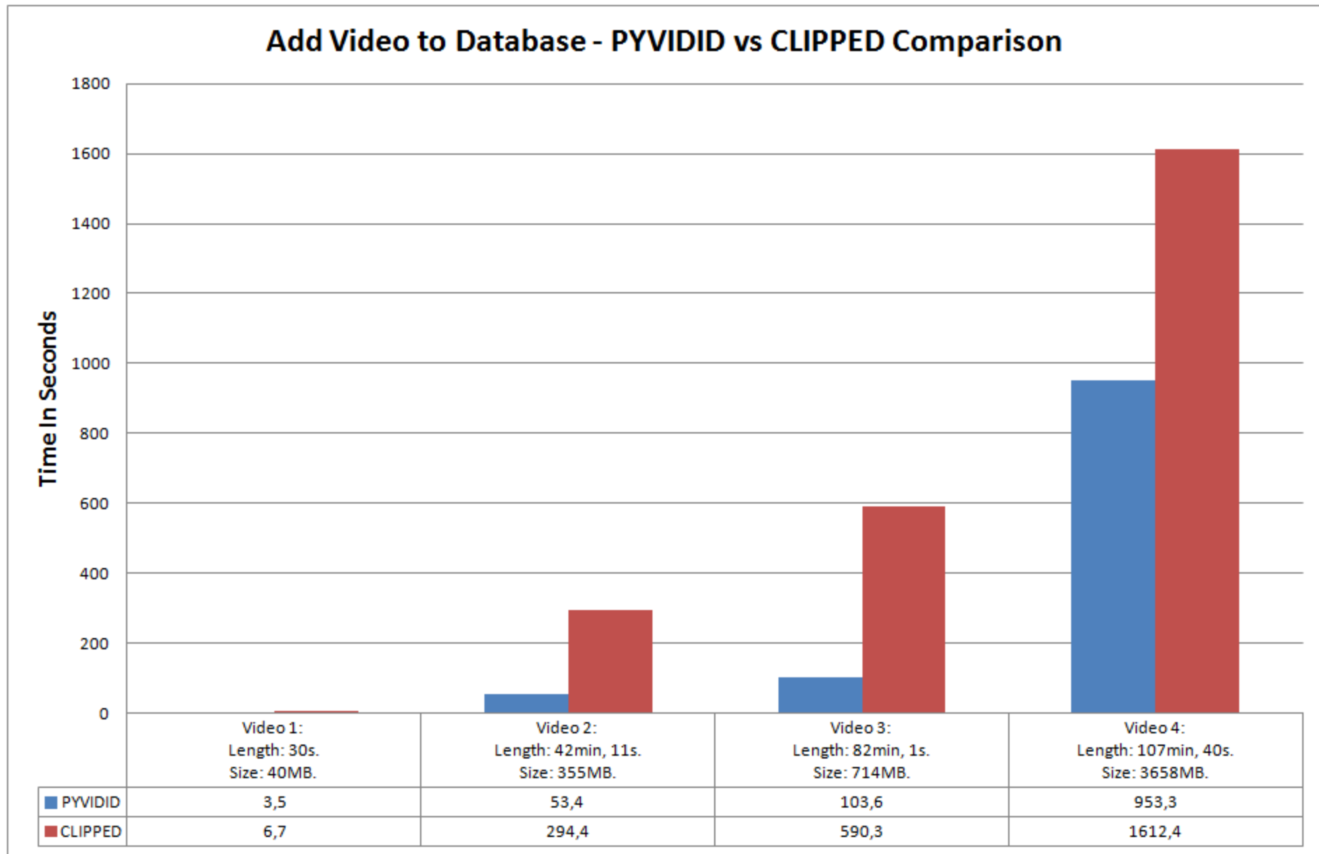

Figure 11: Shows the average time used to hash a single image 30 times.

4.5.2 Perceptual Hash Algorithms

CLIPPED and PYVIDID uses perceptual hash algorithms for video signature generation by hashing images taken at one second intervals. CLIPPED utilizes the pHash function in the Java ImagePHash library to produce a 49 bit hash, while PYVIDID offers three different perceptual hash algorithms producing a 64 bit hash by default. This is achieved using the Python ImageHash library. In addition to the three perceptual hash algorithms: Average hash, Phash and Distance hash, PYVIDID also offers different hash sizes from the standard 64 bit to 128 and 256 bit.

In this experiment we wanted to measure the differences between the video identification systems image hashing algorithms. We chose to only measure the Python Phash algorithm with different hash sizes because a version of this is used by both video identification systems. Average hash, and Distance hash is only measured with the standard 64 bit hash size. Therefore we performed this experiment 6 times with the following hashing algorithms:

1. Java pHash 49 bit.
2. Python Phash 64 bit.
3. Python Phash 128 bit.
4. Python Phash 256 bit.
5. Python Average hash 64 bit.
6. Python Distance hash 64 bit.

Each experiment consisted of hashing the same image 30 times. The resulting measurements was then averaged and plotted into Figure 12

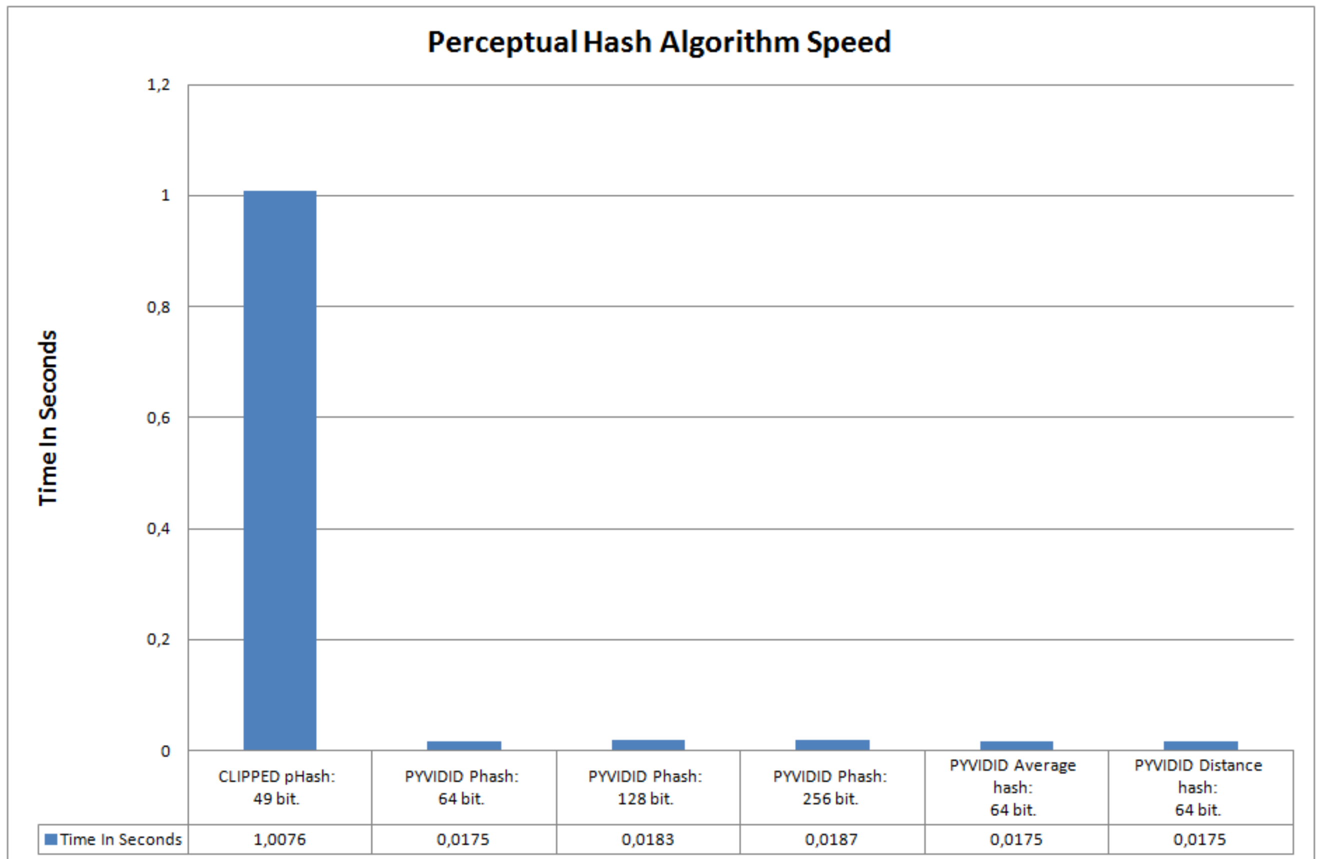

Figure 12: Shows the average time used to hash a single image 30 times.

Figure 12 shows the average time in seconds it took for each perceptual hash algorithm to hash the same image 30 times. As we will discuss in Chapter 5, this is an important measurement for performance because it only considers algorithm speed. This is in contrast to the previous experiment of adding a video to the database, were converting a video into images is included.

This clearly shows a major performance improvement in PYVIDID.

4.5.3 Matching Speed

Both CLIPPED and PYVIDID uses hamming distance to compare a query signature against the database. This is done using a sliding window technique, were the major difference between the two systems is the added temporal localization feature added in PYVIDID. We wanted to measure the matching speed of both video identification systems. In this experiment we measured the time it took to match a query video against the database. For this experiment we chose two different query videos:

1. Video 1: A 30 second clip taken from a random video in the database.
2. Video 2: A full tv-show episode. Length: 25min, 13s

These two videos were chosen because they make use of all matching functions in the video identification systems. A short clip for the sliding window technique and temporal localization, and a full length video already contained within the database for direct matching.

Because our PYVIDID implementation contains two different matching methods (A Python and Java version) this experiment was conducted three times for each video. One with CLIPPED, and one with PYVIDID's Python and Java versions respectively. Each video was matched against the full database consisting of ~500 hours of video. We measured the time it took for the query video to compare against every video in the database. These measurements was then averaged and plotted into Figure 13 and 14.

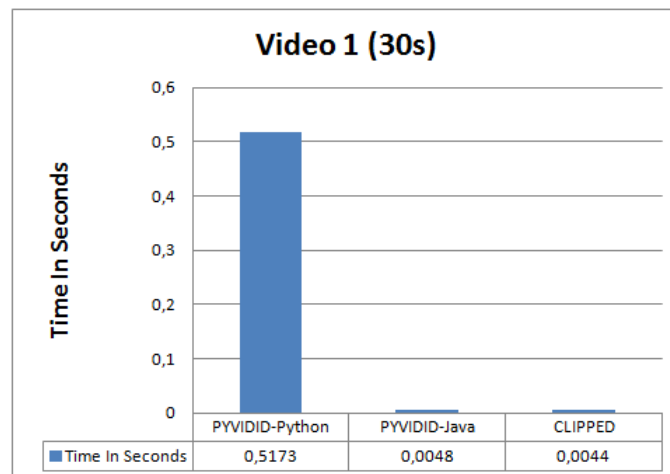

Figure 13: Shows the average time used to compare a 30s video clip against each database video.

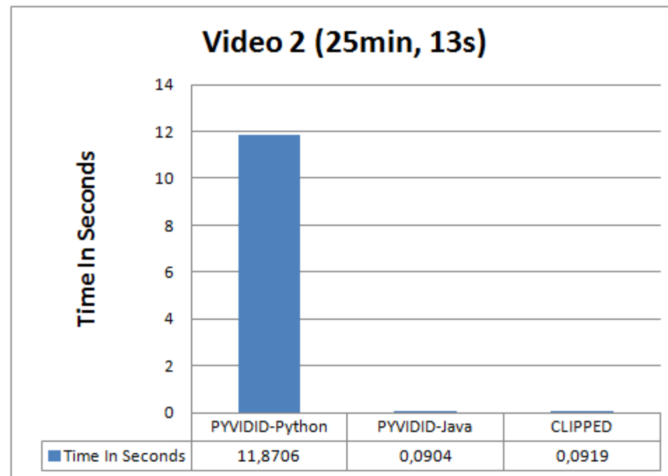

Figure 14: Shows the average time used to compare a 25min, 13s video clip against each database video.

4.6 Experiment #4 - Temporal Location

The temporal location experiment was conducted in order to determine the accuracy of locating a query clip in a longer video. This experiment was only performed on PYVIDID since this was the only video identification system we had access to with this feature. We chose the following query video types to match against the database of ∼500 hours of video.

1. Video type 1: A 30 second clip taken from a random video in the database.
2. Video type 2: A 5 second clip taken from a random video in the database.

We created 10 different query clips for each video type. These clips were taken from random videos in the database. This meant we performed this experiment 10 times for each video type, 20 in total. There were three possible outcomes for this experiment:

- The query clip matched the correct video, and reported the correct temporal location.
- The query clip matched the correct video, but reported the wrong temporal location.
- The query clip matched the wrong video, therefore we disregard the temporal location feature.

While the query clips were chosen at random, we made sure it did not contain common parts such as intros in tv-show, and commercials in sport events. In order to determine if the temporal location was correct we allowed a +-2 second margin. This meant that if the temporal location reported was less than 2 seconds off in either direction, we judged this results as a correct match. After performing the experiment we plotted the results for each video type into Figure 15.

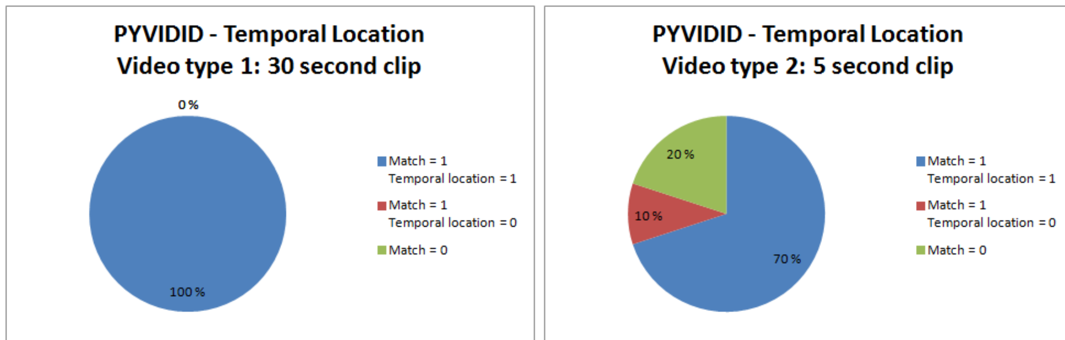

Figure 15: Shows the performance of PYVIDID's temporal location feature with two different length query clips.

For the 30s clips PYVIDID achieved a 100% accuracy for locating a query clip within a longer video. For the 5s clips we achieved a 70% accuracy. 20% matched the correct video, but failed to located the correct temporal location. 10% simply matched the wrong video altogether.

5 Discussion

In this chapter we discuss the results of our experiments. We also discuss possible application areas for a content based video identification systems.

5.1 Results Discussion - Accuracy

The most used measurement when evaluating and comparing video identification systems is accuracy and speed. In order to determine the accuracy for PYVIDID and CLIPPED, we conducted the independence experiment matching different length query clips against the database. Using three different length video clips(Full, 30s, 5s), we matched 20 query clips of each length to our database containing ~500 hours of video. These query clips contained no video transformation whatsoever and was taken from videos contained within the database. The results from this experiment is shown in Section 4.3 Tables 2 and 3.

The independence experiment gave us true and false positive rates for the video identification systems. Since the query videos did not contain any video transformations, these experiments indicates the uniqueness of the video signatures used in both systems. This is obviously an important measurements, because if similar video files produced undistinguishable signatures, the system would be effectively useless in most scenarios. Using a full length query video resulted in a 100% true positive rate for both PYVIDID and CLIPPED. This is somewhat expected, since the correct matching would not require any use of the sliding windows technique. Only a simple one-to-one matching step. In terms of looking closer at the results, the hamming distance reported on the best match was clearly distinguishable from other non matches. As expected the hamming distance between the same file is very close to 0. We observed a variance from 0-10 when experimenting with full length files. This is a large contrast to matching with a 30s clip, where the hamming distance on a correct match usually was around 100-200, with the second best match at ~700 hamming distance. Clearly also significantly distinguishable, but interesting nonetheless.

When matching 30s long query clips against the database we also achieved a 100% true positive rate. This experiment used the sliding window technique and still achieved excellent results, proving the technique is viable. When matching 5s long query clips we achieved a 95% true positive rate. We believe the reason for the 5% false positive rate is that 5s long query clips is simply too short to always achieve correct matching. As mentioned before, we made sure the query clips did not contain common parts such as intros or commercials. However, our database contained a lot of tv show episodes with similar scenes. One example is scenes captured on the same set, were characters rarely moves. This leads to very similar video signatures. A possible solution for this problem with short query clips, is to not only generate a video signature, but combine it

with some kind of audio information as well. Unfortunately, this would result in a much more advanced video identification system, where signature generation, storage, and matching would require a lot more resources. In the case of our experiment we could simply lay the blame on the test data used in the databases. However, we believe these results is applicable to a real life scenario, were similar video files most likely will be a reality. Therefore, video identification systems needs to take this problem into consideration.

When comparing PYVIDID and CLIPPED against other video identification systems presented in previous literature, we definitely agree in what Neal Krawetz states in [18]: *Perceptual hashes is pretty simple to implement and is far more accurate than it has any right to be.* One thing to note however, is that very few production systems has been published. But as far as publications and open/available video identification systems go, we believe PYVIDID and CLIPPED is an important contribution within the field of video identification.

5.2 Results Discussion - Robustness

A common use for a video identification system is to enforce copyright control. While the independence experiment indicated video identification accuracy, it did not take video transformations into consideration. For a video uploading site such as YouTube, it is common for users to publish copyrighted material. This can and most likely will lead to lawsuits against the publishing site. A solution to this problem is to use a video identification system to identify copyright infringements. If the published video is identical to the copyrighted material, the results for the independence experiment applies. However, as often is the case, users either deliberately or unknowingly apply video transformations to the content, making the identification process more difficult.

Another important function for a video identification systems is therefore the ability to match query videos with common video transformations. A system without this capability would effectively be useless against this circumvention tactic, which is very easy for a user to perform. In order to determine the robustness for PYVIDID and CLIPPED against video transformations, we performed the robustness experiments. Using the same database as before, we took 10 of the 30s query clips from the independence experiment to use as our query clips. From each of these clips, we created 9 clips with various transformations. Counting the original clip, we then had 10 sets containing 10x30s query clips, 100 in total. The video transformations used in this experiment was shown in Section 4.1.

Each query clip was then matched against the database, and the results of the experiment is shown in Section 4.4 Tables 4 and 5. As expected, both video identification systems achieved a 100% accuracy for the original query clip. Although we had already matched this clip against the database in the independence experiment, we chose to include it as a control sample. Somewhat surprisingly again, perceptual hashes performs very well when matching most of the video transformation. Only the crop and rotation transformations achieved a less than 100% true positive

rate.

Crop was the worst transformation for both systems. PYVIDID achieved a 30% true positive rate, while CLIPPED clocked in at 10%. This result is unexpected, since the crop transformation in reality only adds a black border for the whole video. As explained in Section 3.2.1, the pHash algorithm reduces the image down to a 32x32 pixel image before making the hash. We assume this removes to much information to make the image identifiable. We believe the difference between the systems is because of the hash size used. PYVIDID uses a 64 bit hashes, making it contain more information in it's video signatures compared to CLIPPED's 49 bit hashes. In any case, this transformation proves that while perceptual hashes is effective is most cases, it is definitely not invincible. The other transformation causing problems is the rotation variant. Here the picture is rotated 10 degrees to the right. This also causes some black areas on certain edges of the video. As with the crop transformation, we believe the same problem occurs. PYVIDID achieved a 90% true positive rate, compared to CLIPPED's 70%. Again, we believe the difference is in the amount of information stored in the video signatures.

5.3 Results Discussion - Speed

Speed performance for a video identification system depends on a variety of factors. For PYVIDID and CLIPPED we performed 3 different experiments to determine some of them. We will also discuss other potential factors we believe is important for a video identification system in this section.

5.3.1 Add Video to Database

Adding and matching videos is the two main processes of a video identification system. Unlike matching, adding a video to the database can often be done in an offline environment with considerable time and computational resources. Because of this we argue that adding a video to the database while important, is not the most crucial performance factor for a video identification system. However, for a system to be effective in use, we believe that is should at least perform a at one-to-one ratio against the video length, meaning that adding a 1 hour video to the database should not take longer than 1 hour.

We performed an experiment on PYVIDID and CLIPPED measuring the time if took to add different size and length video files to our databases. Figure 11 shows our results as a comparison between the two systems. From these results we can draw two main conclusions:

1. PYVIDID performs considerably better at all sizes and video lengths.
2. Adding speed for both systems depends both on size and video length.

The fact that adding speed depends on both video size and length can clearly be seen when comparing the results for video 2, 3, and 4. If our hypothesis was that only one factor decided adding speed, we disprove this with the following reasoning: Video 3 is about double the size

and length of video 2. If our hypothesis was correct, we would expect the results for video 4 to be proportional to this. However, video 4 resulted in considerably worse adding speed for both systems, proving that both factors decide the adding time.

With knowledge on how both systems operate we can therefore say that there are two major processes that determines the speed of adding a video to the database:

- The conversion step where ffmpeg is used to convert a video file into images.
- The hashing step where a perceptual hashing algorithm is used to hash every image into a binary hash.

Since we also performed an experiment on perceptual hashing algorithms speed, we can with that in mind conclude that this experiment showed us that the ffmpeg conversion step is a major bottleneck in both systems. We believe the reason for this bottleneck is the way ffmpeg converts a video into images. First, the video is read into memory from disk, then each second of video produces an images which is then stored on the disk again. After this, each image is then again read from disk and fed into the perceptual hash algorithm used to create a binary hash. This obviously requires a lot of disk reads and writes, making it a slow process. While we cannot prevent ffmpeg reading the video file, an alternative solution would be to keep the images in memory and perform the hashing step there. This should significantly reduce the signature generation process for both PYVIDID and CLIPPED.

5.3.2 Perceptual Hash Algorithms

The core idea behind PYVIDID and CLIPPED is to use perceptual hash algorithms for video signature generation. CLIPPED uses the Java ImagePHash library, offering a PHash algorithms for this purpose. We developed PYVIDID in Python allowing us access to the ImageHash library containing several different perceptual hash algorithms. In order to determine the performance for each algorithm we performed an experiment measuring the time it took for each algorithm to hash a single image. The results from this experiment in shown in Section 4.5.2 Figure 12.

CLIPPED uses on average 1.0076 seconds to hash a single image into a perceptual hash using the PHash algorithm. In Section 5.3.1 we claimed that for a video identification system to be effective it should at least be able to add videos to the database at a one-to-one ratio compared against the video length. Since adding a video to the database is a 3 step process; 1)Convert video into images. 2)Hash images. 3)Create a video signature and add it to the database, we believe this to be too slow for an effective video identification system. On the other hand, PYVIDID with the Python library delivers very good results. Using Phash to produce a 64 bit hash, PYVIDID uses on average 0,0175 seconds to hash a single image. This is 57 times quicker that CLIPPED, making this a major performance upgrade.

5.3.3 Match Video Against Database

Matching a video against the database is perhaps the most important performance factor regarding speed. This process is in almost all cases performed live, where matching time is crucial to the video identification systems effectiveness. Again, looking at YouTube as an example; All uploaded video is checked against the database containing copyrighted material before published. If the matching process is too slow, users will be unhappy, and possibly switch to alternate services.

In order to determine the performance of PYVIDID and CLIPPED we performed an experiment measuring matching time with different video lengths. We choose to not include the video signature generation step for the query video, as this speed is already well documented in previous experiments. Our measurements therefore consisted of the time used to compare the query signature against all video signatures in the database. The results of these experiments is shown in Section 4.5.3, Figures 13 and 14.

PYVIDID was first developed fully in Python. After performing these experiments however, we saw that the Python version of the matching algorithm was very slow compared to CLIPPED. Instead of spending too much time optimizing the Python version, we tried to implement the exact same algorithm in Java. As the results show, this greatly reduced the matching time down to the same level as CLIPPED. Both PYVIDID algorithms uses a simple sliding window technique with hamming distance to match binary video hashes against each other.

When comparing PYVIDID to CLIPPED the results is as expected. Since CLIPPED uses a 49bit hash compared to PYVIDID's 64bit, we expected to see a minor speed difference, which is the case. Our matching results does not include any measurements on larger hash sizes because of the time required to perform the experiment. However, in a quick test, we did confirm that larger hash sizes only slightly increases matching speed, lower than the linear increase between 49bit and 64bit already tested.

ABIT MORE DISCUSSION HERE Speed summary. ffmpeg only really thing that takes time in pyvidid ..clearly size of file matters when adding to database sincce ffmpeg is bottleneck. ...java vs python matching. hamming funksjonen som faktisk skiller ???? matching: as expected, lenght is the only factor since signatures are same size no mather what the size of original video.

5.4 Results Discussion - Temporal Localization

Locating the position of a query clip within a larger video is a PYVIDID only feature. To our knowledge, no free video identification system has this capability either. In order to determine the effectiveness of PYVIDID's temporal localization feature, we conducted an experiment using two different length query videos. The results of this experiment is shown in Section 4.6 Figure 15.

Our experiment allowed 3 possible outcomes:

- The query video resulted in a correct video and temporal position.
- The query video was matched to the correct video, but gave the wrong temporal localization.
- Wrong video match entirely, in this case temporal location was discarded.

Using a 30s long query video, we achieved a 100% accuracy for temporal localization. This is somewhat expected, since the matching accuracy for this length query clips is also 100% as shown in previous experiments. This is because the temporal localization is simply an extra bit of information stored during the matching process. Therefore, if the matching is accurate, we expected the temporal localization to be accurate as well, especially for longer query clips. Using a 5s query clip gave us a 70% accuracy for temporal localization. The remaining 30% was split into 10% correct match, but wrong temporal position, and 20% wrong video match entirely. As discussed with the independence experiment, we believe the reason for the non video match is the fact that we used several tv show seasons in the database containing similar scenes. For the wrong temporal position results, there were 2 scenarios. One of the results only missed the correct position with ~5s, while the other one was completely off base. Again, these results came from query clips taken from tv show episodes. As with the non matching results, we believe that similar scenes in the same video is the cause. This is because 5s is sometimes not enough to completely distinguish similar scenes taken at the same set.

5.5 Results Discussion - Summary

After comparing our results to previous work in the video identification field it is clear that PYVIDID has a significant performance increase. PYVIDID can perform video identification for video fragments as well as fully length files. It also has promising results regarding video transformations for all file lengths as well as temporal localization. With perceptual hash algorithms we achieve high speed and accuracy for signature generation and matching. While similar to Gardåsens's CLIPPED, we have developed PYVIDID from scratch in Python, and made significant improvements.

It is also important to note that PYVIDID is by no means perfect in its current condition. Using ffmpeg for image generation is a major bottleneck. As discussed a possible solution to this would be to use in memory management of images when generating video signatures. We believe this is one of the major weaknesses of PYVIDID at this time.

5.6 Applications

With advances in technology the use of image and video content has sky rocketed over the past decades. As mentioned in Section 3 this has lead to an increased amount of research within the identification field. A common property for video identification systems is that they can be used for several different purposes. In this section we discuss some of the applications we believe is most relevant as of today. Some of them are maybe a bit far fetched, but like many areas within computer science, solutions that seems to futuristic is often closer than we think. It is important

to note that while most systems can be used for different purposes, they often require different weighed performance factors.

Video Database

Having a database of all video files requires a lot storage space. This is relevant for both private users and businesses that deals with a large amount of video content. An alternative for storing all the video content is to only store video signatures. As shown in Section 4.2, PYVIDID only requires 0.027 MB per hour of video stored in its database. In cases storage capacity is an issue, this would be a convenient alternative. This option also has another advantage, especially if copyrighted/sensitive content is added to the database. Since the database only contains video signatures, any unauthorized access would not mean that the content is compromised. There is no reliable way of reversing the video signatures back to the original content. This is because perceptual hashes as seen in Section 3.2.1 removes most of the information before hashing each image.

Copyright Control

Perhaps the most intuitive use for a video identification system is copyright control. A common problem for video uploading sites is the threat of legal action from copyright holders who finds their content uploaded by other users. Therefore, these sites need a solution who can stop copyright infringements. A prime example for this type of system is the YouTube Content ID [28]. Each uploaded video is matched against the a database containing copyrighted material gathered from copyright owners. If a match is found, the video can either be blocked, monetized, or tracked.

Monitoring and Security

Using a video identification system for distribution monitoring is similar, but bot equal to copyright control. Many organizations and companies produces or owns a lot of sensitive multimedia content. It is therefore in their interest to keep track of all this sensitive material. This could be done using a video identification system in conjunction with network monitoring. If any video material is detected leaving the network, a video identification system can be used to match it against a database containing the sensitive material. If a match is found, this network traffic can be blocked so that any deliberate or accidental transmission is avoided. This type of security mechanism would require considerable resources in both development and operational cost. However, if the content is valued enough, it might be a possible use for a video identification system.

In a more moderate monitoring scheme, a video identification system can be used for simple monitoring and tracking of material. One example of this is for a advertising company to keep track of their commercials used in broadcasts. By matching the broadcast to their database, they can make detailed statistics about how and when their commercials are used. This information can then checked against agreements to make sure the broadcaster adheres to the contract.

Forensics

Video identification can also be used within the field of computer forensics. A major challenge for law enforcement is child pornography. The anonymity and accessibility the Internet provides only adds to this challenge. Both images and videos is easily exchanged with little to no forensic evidence left behind. In the media, we frequently hear about police raids and evidence gathering related to child pornography. Unfortunately with the large amount of computers and other storage media confiscated, it takes a lot of manpower to go sift through it. A possible automatic solution to this is a video identification system.

Unlike most other issues, most countries and agencies has a similar view on child pornography. By working together, a database containing all known material can be generated. This cooperation is crucial, because the Internet has no borders, or jurisdictions. The more material added to the database, the more effective it will be. A video identification system can then be used to match all video files on a system gathered as evidence. For this kind of video identification to be effective it also need the ability to automatically locate video files. This can be done using simple methods such as file endings, or more advanced methods such as content based file classification discussed in Section 2. Taking PYVIDID as an example, it can already perform the identification process for such a system. While a possible classification features remains to be explored, it is definitely possible as shown in Section 2.

6 Future Work

In this master thesis we investigated how perceptual hash algorithms could be used for video identification. We developed PYVIDID, a video identification system capable of searching for both fragmented and transformed video files in a large database. While PYVIDID is operational at this date, it is still a proof of concept implementation and a lot of work is required before it can be used in any kind of production environment. Most noticeable is error handling, input validation and result reporting. These all are fairly common tasks to complete and should not be very challenging. In fact, we plan to do this in cooperation with Kjetil Gardåsen, author of CLIPPED [26] in order to develop a fully functioning video identification system ready for use.

When it comes to possible future work within the field of video identification the major bottleneck in PYVIDID remains to be explored. We concluded in our speed performance experiments that using ffmpeg to convert a video into images which is then saved on the hard drive is a major bottleneck. Future work in this area would be to skip this step, and have the perceptual hash algorithms take the images directly from memory and only save the perceptual hash on disk. This would effectively speed up the video signature generation step a lot.

While the accuracy for most query videos used in our experiments where high, we did have certain performance issue related to short video clips. Using a 5 second long query clip we saw the accuracy drop drastically, especially for certain video transformations. We believe this is because too little information is contains within the video signature as the video signatures used in PYVIDID only contains picture information. A possible solution to this problem would be to add additional information such as an audio signature. After all a video file is both video and audio. While this would also require a much more advanced identification system, we believe this option could be worth exploring in the future.

We also speculated in using file type classification techniques in areas such as digital forensics in order to automatically locate video content in large data collections. This could then be tied together with a video identification system to perform video search. As our work within this area is pure discussion and speculation this remains to be explored. Creating such a system would therefore be an interesting task for another day.

7 Conclusion

In this master thesis we investigated the possibilities for a content based video identification system using perceptual hash algorithms. By developing PYVIDID, a Python based video identification system, we have shown that perceptual hash algorithms can in fact be used to identify video content with high accuracy. We have also shown that PYVIDID can effectively identify both video fragments and transformed video files. For a 30 second or longer video clip, PYVIDID is capable of searching through a database containing ∼500 hours of video with a 100% accuracy according to our test results. While the results slightly diminish for shorter video fragments, a 5 second long clip still produced a respectable 95% accuracy. When it comes to video transformations PYVIDID is also able to handle quite drastic changes to a query clip while matching it back to its original source.

The video signatures generated by perceptual hash algorithms is also very small compared to previous work in this area. PYVIDID generates on default settings a video signature consisting of 0.027 MB per hour of video. The small video signatures also contributes to effective signature creation and matching steps.

Overall, our results in this master thesis clearly shows that image identification techniques such as perceptual hash algorithms can be extended to video identification with high performance. PYVIDID is able to match both video fragments and transformed video files with an almost surprisingly high accuracy. While some speed optimization related to signature generation and matching is possible in the future, PYVIDID already has the performance required for most application areas discussed in thesis. We believe the most relevant areas for a content based video identification system today is copyright control, content management and possibly digital forensics.

Bibliography

[1] YouTube. May 2013. Here's to eight great years. http://youtube-global.blogspot.no/2013/05/heres-to-eight-great-years.html [Online; accessed 15-May-2014].

[2] TinEye. 2014. Tineye - reverse image search. http://www.tineye.com [Online; accessed 15-May-2014].

[3] Google. 2014. Google images. http://images.google.com/imghp?hl=en [Online; accessed 15-May-2014].

[4] McDaniel, M. & Heydari, M. 2003. Content based file type detection algorithms. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 10 pp.–.

[5] Li, W.-J., Wang, K., Stolfo, S., & Herzog, B. 2005. Fileprints: identifying file types by n-gram analysis. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, 64–71.

[6] Karresand, M. & Shahmehri, N. 2006. Oscar — file type identification of binary data in disk clusters and ram pages. In *Security and Privacy in Dynamic Environments*, Fischer-Hübner, S., Rannenberg, K., Yngström, L., & Lindskog, S., eds, volume 201 of *IFIP International Federation for Information Processing*, 413–424. Springer US.

[7] Karresand, M. & Shahmehri, N. 2006. File type identification of data fragments by their binary structure. In *Information Assurance Workshop, 2006 IEEE*, 140–147.

[8] Erbacher, R. & Mulholland, J. 2007. Identification and localization of data types within large-scale file systems. In *Systematic Approaches to Digital Forensic Engineering, 2007. SADFE 2007. Second International Workshop on*, 55–70.

[9] Moody, S. & Erbacher, R. 2008. Sadi - statistical analysis for data type identification. In *Systematic Approaches to Digital Forensic Engineering, 2008. SADFE '08. Third International Workshop on*, 41–54.

[10] Veenman, C. 2007. Statistical disk cluster classification for file carving. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, 393–398.

[11] Lempel, A. & Ziv, J. Jan 1976. On the complexity of finite sequences. *Information Theory, IEEE Transactions on*, 22(1), 75–81.

[12] Calhoun, W. C. & Coles, D. September 2008. Predicting the types of file fragments. *Digit. Investig.*, 5, S14–S20.

[13] Amirani, M., Toorani, M., & Beheshti, A. July 2008. A new approach to content-based file type detection. In *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, 1103–1108.

[14] Roussev, V. & Garfinkel, S. 2009. File fragment classification-the case for specialized approaches. In *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE '09. Fourth International IEEE Workshop on*, 3–14.

[15] Gopal, S., Yang, Y., Salomatin, K., & Carbonell, J. Dec 2011. Statistical learning for file-type identification. In *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, volume 1, 68–73.

[16] Paschalakis, S., Iwamoto, K., Brasnett, P., Sprljan, N., Oami, R., Nomura, T., Yamada, A., & Bober, M. July 2012. The mpeg-7 video signature tools for content identification. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(7), 1050–1063.

[17] Krawetz, N. May 2011. Looks like it. `http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html` [Online; accessed 15-May-2014].

[18] Krawetz, N. 2013. Kind of like that. `http://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html` [Online; accessed 15-May-2014].

[19] Law-To, J., Chen, L., Joly, A., Laptev, I., Buisson, O., Gouet-Brunet, V., Boujemaa, N., & Stentiford, F. 2007. Video copy detection: A comparative study. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, 371–378, New York, NY, USA. ACM.

[20] Shen, H. T., Zhou, X., Huang, Z., Shao, J., & Zhou, X. 2007. Uqlips: A real-time near-duplicate video clip detection system. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, 1374–1377. VLDB Endowment.

[21] Starkweather, K. . Clipseekr. `http://www.phash.org/apps/#clipseeker` [Online; accessed 15-May-2014].

[22] OpenCV. Opencv. `http://opencv.org/` [Online; accessed 15-May-2014].

[23] OpenCV. Video input with opencv and similarity measurement. `http://docs.opencv.org/doc/tutorials/highgui/video-input-psnr-ssim/video-input-psnr-ssim.html` [Online; accessed 15-May-2014].

[24] YouTube. May 2014. Statistics. `http://www.youtube.com/yt/press/statistics.html` [Online; accessed 15-May-2014].

[25] Tassi, P. 2013. The injustice of the youtube content id crackdown reveals google's dark side. `http://www.forbes.com/sites/insertcoin/2013/12/19/the-injustice-of-the-youtube-content-id-crackdown-reveals-googles-dark-side` [Online; accessed 15-May-2014].

[26] Gardåsen, K. T. 2013. Clipped: A solution for

finding the source footage from a video clip. https://github.com/Data-Kjetil/CLIPPED [Online; accessed 15-May-2014].

[27] FFMPEG. Ffmpeg. http://www.ffmpeg.org/ [Online; accessed 15-May-2014].

[28] Google. How content id works. https://support.google.com/youtube/answer/2797370?hl=en [Online; accessed 15-May-2014].

[29] Attrasoft. Video finder. http://www.attrasoft.com/products_videofinder.asp [Online; accessed 15-May-2014].

[30] Spyder. Spyder. http://spyder-ide.blogspot.no/ [Online; accessed 15-May-2014].

[31] WinPython. Winpython. http://winpython.sourceforge.net/ [Online; accessed 15-May-2014].

[32] Eclipse. Eclipse. http://www.eclipse.org/ [Online; accessed 15-May-2014].

[33] Buchner, J. Imagehash. https://pypi.python.org/pypi/ImageHash/0.3 [Online; accessed 15-May-2014].

A PyCLIPPED - Python Code

```
#Video Identification system implemented in python
#using the ImageHash library.
#Author: Ola Kjelsrud
#
#Functionality based on Kjetil Gardaasen's CLIPPED.
#https://github.com/Data-Kjetil/CLIPPED



import subprocess
import os
import imagehash
from PIL import Image
import sqlite3


ADD = "ADD"                #Add videos to database path
MATCH="MATCH"              #Matctch videos against database path
TMP = "TMP"                #Tmp picture storage
MATCHHASHES = "MATCHHASHES" #Folder to store query hashes

#Section 1
##################################################################
#This section adds all videos in ADD folder to the database.

#Adds all videos in ADD folder to database.
#Each video is only added once.
def AddVideos():
    SetUpDatabase()
    global ADD
    db = GetVideosInDatabase()
    for video in os.listdir(ADD):
        if video not in db:
            print("Adding Video: " + video)
            DeleteImages()
            ConvertVideoToImages(ADD, video)
            hashes = HashImages(video)
            AddHashesToDatabase(video, hashes)
    DeleteImages()



#Sets up the database with tables and keys.
def SetUpDatabase():
    con = sqlite3.connect("db\\VideoIdentification.db")
    with con:
```

```
        cur = con.cursor()
        cur.execute("""PRAGMA foreign_keys=ON""")
        cur.execute("""CREATE TABLE IF NOT EXISTS VIDEO_NAMES(VIDEO
            INTEGER PRIMARY KEY AUTOINCREMENT, NAME TEXT NOT NULL)""")
        cur.execute("""CREATE TABLE IF NOT EXISTS VIDEO_HASHES(VIDEO
            INTEGER NOT NULL, FRAME INTEGER NOT NULL,
            HASH VARCHAR(64) NOT NULL, PRIMARY KEY(VIDEO, FRAME),
            FOREIGN KEY(VIDEO) REFERENCES VIDEO_NAMES(VIDEO))""")


#Retrieves a list of videos in the database.
#Format: a[1] = NameOfVideo1...
def GetVideosInDatabase():
    db = []
    db.append("NULL")
    con = sqlite3.connect("db\\VideoIdentification.db")
    with con:
        cur = con.cursor()
        cur.execute("""PRAGMA foreign_keys=ON""")
        cur.execute("""SELECT NAME FROM VIDEO_NAMES""")
        for i in cur:
            db.append(i[0])
    return db


#Deletes all images in TMP folder
def DeleteImages():
    global TMP
    for image in os.listdir(TMP):
        os.remove(TMP + "\\" + image)


#Convert a video to images. 1 frame/sec.
def ConvertVideoToImages(convert, video):
    global TMP
    ffmpeg = "ffmpeg-20131208-git-ae33007-win64-static\\bin\\ffmpeg.exe"
    arg1 = " -v 0 -i "
    arg2 = convert + "\\" + video
    arg3 = " -r 1 "
    arg4 = TMP + "\\" + os.path.splitext(video)[0] + "-%d.jpg"
    subprocess.Popen(ffmpeg + arg1 + arg2 + arg3 + arg4).wait()




#Hash all converted images in TMP directory related to video
def HashImages(video):
    global TMP
    hashSize = 8
    hashes = []
    imh =0
    hashes.append(video)
    for i in range(1, len(os.listdir(TMP))+1):
```

```
        image = TMP + "\\" + os.path.splitext(video)[0] +
            "-" + str(i) + ".jpg"
        #print("Hashing Image: " + image)
        imh = str(imagehash.phash(Image.open(image), hashSize))
        hashes.append(bin( int(imh, 16))[2:].zfill(64))
    return hashes


#Add all hashes from a video to the database
def AddHashesToDatabase(video, hashes):
    con = sqlite3.connect("db\\VideoIdentification.db")
    with con:
        cur = con.cursor()
        cur.execute("""PRAGMA foreign_keys=ON""")
        cur.execute("""INSERT INTO VIDEO_NAMES(VIDEO, NAME)
            VALUES(?, ?)""", (None, video))
        videonumber = cur.lastrowid
        for i in range(1, len(hashes)):
            cur.execute("""INSERT INTO VIDEO_HASHES(VIDEO, FRAME, HASH)
                VALUES(?, ?, ?)""", (videonumber, i, str(hashes[i])))



#Section 2
###################################################################
#This section searches the database for a match
#against each video in the MATCH folder


#Search for a match in the database for all videos in MATCH folder
def MatchVideos():
    global MATCH
    db = GetVideosInDatabase()
    for match in os.listdir(MATCH): #For every video in MATCH folder
        best = [10000, 10000, "None"]
        average = 0
        lowestAverage = 10000
        DeleteImages()
        ConvertVideoToImages(MATCH, match)
        matchhashes = HashImages(match)
        for i in range(1, len(db)): #For every video in DATABASE
            dbhashes = GetHashesInDatabase(db[i], i)
            average = 0
            lowestAverage = 10000
            temporal= 0
            dbl = len(dbhashes)
            matchl = len(matchhashes)
            start = time.time()
            for k in range(1, dbl): #For every frame in DATABASE video
                if matchl <= (dbl-k)+1:
                    average = 0
                    #Move match window 1 step forward
                    for j in range(1, matchl):
```

```
                    diffs = 0
                    for ch1, ch2 in
                            zip(dbhashes[k+j-1], matchhashes[j]):
                        if ch1 != ch2:
                            diffs += 1
                    average += diffs
                if average < lowestAverage:
                    lowestAverage = average
                    temporal=k
            print("\nComparing: " + str(dbhashes[0]) + " vs " +
                str(matchhashes[0]) + "\n" , (start - time.time()))
            print("Average: " + str(lowestAverage) + " - Temporal: " +
                str(temporal) + " sek - Video: " + db[i])
            if lowestAverage < best[0]:
                best[0] = lowestAverage
                best[1] = temporal
                best[2] = db[i]
        print("\n\nBest match for video: " + match + "\nAverage: " +
                str(best[0]) + " - Temporal: " + str(best[1]) +
                    " sek - Video: " + best[2])
    DeleteImages()



#Retrieves all hashes related to a video in the database.
#Format: a[1]= Hash1...
def GetHashesInDatabase(name, number):
    hashes = []
    hashes.append(name)
    con = sqlite3.connect("db\\VideoIdentification.db")
    with con:
        cur = con.cursor()
        cur.execute("""PRAGMA foreign_keys=ON""")
        cur.execute("""SELECT FRAME, HASH FROM VIDEO_HASHES
            WHERE VIDEO = ?""", str(number))
        for i in cur:
            hashes.append(i[1])
    return hashes

#Retrieve hashes for videos in MATCH folder from database.
#Store in MATCHHASHES as .txt
def MatchHashToFile():
    global MATCH
    global MATCHHASHES
    DeleteHashes()
    for match in os.listdir(MATCH): #For every video in MATCH folder
        DeleteImages()
        ConvertVideoToImages(MATCH, match)
        matchhashes = HashImages(match)
        f = open(MATCHHASHES + "\\" + os.path.splitext(match)[0] +
                ".txt", "w")
        for i in range(1, len(matchhashes)):
```

```
            f.write(matchhashes[i] + "\n")
    DeleteImages()

#Deletes all query hashes in MATCHHASHES folder.
def DeleteHashes():
    global MATCHHASHES
    for file in os.listdir(MATCHHASHES):
        os.remove(MATCHHASHES + "\\" + file)

#Reports size of database in hours.
def GetHoursOfVideoInDatabase():
    con = sqlite3.connect("db\\VideoIdentification.db")
    with con:
        cur = con.cursor()
        cur.execute("""PRAGMA foreign_keys=ON""")
        cur.execute("""SELECT COUNT(FRAME) FROM VIDEO_HASHES""")
        for i in cur:
            print("Hours of video in database: " + str(i[0]/60/60))




#Deletes the whole database
def DeleteDatabase():
   con = sqlite3.connect("db\\VideoIdentification.db")
   with con:
       cur = con.cursor()
       cur.execute("""DROP TABLE IF EXISTS VIDEO_HASHES""")
       cur.execute("""DROP TABLE IF EXISTS VIDEO_NAMES""")


#Main - Presents a main menu for operation
def Main():
    close = True
    while close:
        print("\n\n\nPCLIPPED Video Identification")
        print("1: Add to database")
        print("2: Match query video")
        print("3: Get hours of video in database")
        print("4: Exit")
        ans = input("Enter meny choice:\n")
        if ans == "1":
            AddVideos()       #Add videos to database
        elif ans == "2":
            MatchHashToFile() #Get hashes from database
            subprocess.Popen("java -jar MatchVideos.jar").wait()
        elif ans == "3":
            GetHoursOfVideoInDatabase() #Prints database size
        else:
            close = False
            exit

Main()                  #Run Main()
```

B PyCLIPPED - Java Code

```
package VideoIdentification;

import java.io.*;
import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

public class MatchVideos {
       //Searches the database for a match against matchhashes
       public static void SearchDatabaseForMatch(ArrayList<String> matchhashes ) throws
           Exception{
              // Average Hamming distance of the compared frames.
              int average = 0;
              // The current lowest average Hamming distance for compared frames.
              int lowestAverage = 1000000;
              // The best overall lowest average Hamming distance.
              int bestAverage = 1000000;
              // Best temporal location - Start
              int bestStart = 0;
              // Best temporal location - Stop
              int bestStop = 0;
              String bestVideo = "";
              // A list of films in the database.
              ArrayList<String> videos = new ArrayList<String>();
              // A list of the hashes for a film.
              ArrayList<String> dbhashes = new ArrayList<String>();
              int i = 0;
              int j = 0;
              int k = 0;
              int start = 0;
              int stop = 0;
              //Set up connection to SQL database and get all films in the database
              Connection c = null;
              PreparedStatement ps = null;
              ResultSet rs = null;
       Class.forName("org.sqlite.JDBC");
       c = DriverManager.getConnection("jdbc:sqlite:DB\\VideoIdentification.db");
              ps = c.prepareStatement("SELECT NAME FROM VIDEO_NAMES");
              rs = ps.executeQuery();
              videos.add(null);
       while ( rs.next() ) {
              videos.add(rs.getString("NAME"));
       }
              //Go through every film in the database and
              //compare it againt the query video(matchhashes)
```

```
        for(i = 1; i < videos.size(); i++){
                ps = c.prepareStatement("SELECT FRAME, HASH FROM VIDEO_HASHES
                     WHERE VIDEO = ?");
                ps.setInt(1, i);
                rs = ps.executeQuery();
                dbhashes.clear();
                lowestAverage = 1000000;
                dbhashes.add(videos.get(i));
                while ( rs.next() ) {
                dbhashes.add(rs.getString("HASH"));
        }
                //Use the sliding windows techinique to go through
                //every possible query clip position
                for( j = 1; j < dbhashes.size(); j++){
                        if(matchhashes.size() <= dbhashes.size()-j+1){
                                average = 0;
                                for(k = 1; k < matchhashes.size(); k++){
                                        int dist = distance(dbhashes.get(j+k-1),
                                            matchhashes.get(k));
                                        average += dist;
                                }
                                if (average < lowestAverage){
                                        lowestAverage = average;
                                        start = j;
                                        stop = j+k-1;
                                }
                        }
                }
                //Print results for each video
                System.out.println(lowestAverage + "\t " + start + " -> " + stop
                     + "\t= " + videos.get(i));
                if(lowestAverage < bestAverage){
                        bestAverage = lowestAverage;
                        bestStart = start;
                        bestStop = stop;
                        bestVideo = videos.get(i);

                }
        }
        //Print the best overall results
        System.out.println("\nBest Match Average: " + bestAverage + "\n" +
            matchhashes.get(0).split("\\.")[0] + " == " + " " + bestVideo +
            "\nTemporal: " + bestStart/3600 + ":" + (bestStart % 3600) / 60 +
            ":" + bestStart%60 + "---->" + bestStop/3600 + ":" + (bestStop %
            3600) / 60 + ":" + bestStop%60 + "\n\n");
}

//Function calculating hammint distance between two strings

public static int distance(String s1, String s2) {
        int counter = 0;
        for (int k = 0; k < s1.length();k++) {
                if(s1.charAt(k) != s2.charAt(k)) {
```

```
                        counter++;
                    }
                }
                return counter;
        }

        //Main function. Reads all match hashes from file and sends them to the
        //SearchDatabaseForMatch function for database search.

        public static void main(String[] args) throws Exception{
        ArrayList<String> matchhashes = new ArrayList<String>();
        String file;
        File folder = new File("MATCHHASHES");
        File[] listOfFiles = folder.listFiles();
                BufferedReader br = null;

                //For every query video search all video in the database.
        for (int i = 0; i < listOfFiles.length; i++){
                file = listOfFiles[i].getName();
                String line;
                        br = new BufferedReader(new FileReader("MATCHHASHES\\" + file));
                        matchhashes.clear();
                        matchhashes.add(file);

                        while ((line = br.readLine()) != null) {
                                matchhashes.add(line);
                        }
                        SearchDatabaseForMatch(matchhashes);
        }
        }
}
```