EFFICIENTLY NAVIGATING INFORMATION OVERLOAD: DEVELOPING A MACHINE LEARNING BASED APPLICATION FOR SUMMARIZING ACADEMIC VIDEOS AND EXTRACTING KEY TOPICS

Tianyang Wang[1], Aleksandr Smolin[2]

[1]Northwood High School, 4515 Portola Pkwy, Irvine, CA 92620
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

The internet is a vast treasure trove of information on any subject that has allowed students and scientists alike unprecedented access to the world's collective knowledge [1]. Unfortunately, a lot of it is buried in hours long seminars, talks and other videos on video sharing websites like YouTube [2]. When researching a topic for an academic essay or paper, one might rightly be shocked by the massive time investment required to dig up the relevant data or citations from these monolithic recordings [3]. This paper develops an application that aims to apply new machine-learning-based transcription and keyword extraction methods to cut these videos into small, digestible chunks, which are labeled with their most important topics in order to allow us only to have to manually analyze the parts of the video that are relevant to our research without losing valuable context details [4]. We applied our program to instructional videos on YouTube, in order to test how well we can rearrange video documents for a more convenient view of its contents and conducted a qualitative evaluation of the approach. The results show that the application works as expected to provide video clips titled with its central topic for the user to download in a reasonable amount of time via a simple browser-based extension [5].

KEYWORDS

Python Flask, Firebase, Assembly AI, Google Chrome

1. INTRODUCTION

The background of the topic of splitting video based on keywords can be traced back to the rise of online video consumption [6]. With the advent of video-sharing platforms like YouTube, Vimeo, and Facebook, video content has become a powerful medium for communication and entertainment. According to recent statistics, over 500 hours of video content is uploaded to YouTube every minute, and by 2022, online videos will make up more than 82% of all consumer internet traffic.

Splitting video based on keywords refers to the process of segmenting long videos into smaller, more manageable clips that are tagged with relevant keywords [7]. This allows viewers to quickly find and watch the specific content they are interested in without having to sift through lengthy

David C. Wyld et al. (Eds): SIPM, ITCA, CMIT, FCST, CoNeCo, SAIM, ICITE, ACSIT, SNLP -2023
pp. 201-210, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130921

videos. The popularity of this technique can be attributed to its ability to make video content more accessible, engaging, and shareable.

One of the benefits of splitting video based on keywords is that it allows creators to optimize their content for search engines. By using relevant keywords, creators can improve the discoverability of their videos and attract more viewers. Additionally, splitting videos into shorter segments can increase engagement by providing viewers with bite-sized content that is easier to consume and share.

However, there are also potential consequences of splitting video based on keywords. One concern is that it may lead to a fragmented viewing experience, where viewers are unable to follow the narrative or context of the video. This could result in decreased engagement and retention rates. Additionally, the overuse of keywords and tags can be seen as spammy, which could hurt the credibility and reputation of the creator.

In conclusion, splitting video based on keywords is an important topic in the world of online video content. It has gained popularity due to its ability to make video content more accessible and engaging, but also has potential consequences that need to be considered. As video continues to dominate online media consumption, the importance of effectively organizing and presenting video content will only continue to grow.

Splitting a video refers to the process of dividing a single video into smaller, more manageable parts. This can be useful in a variety of contexts, such as editing videos, sharing clips on social media, or creating a video montage. There are several methods and tools available for splitting videos, including:

Online video editors: There are several online video editors, such as WeVideo, Kapwing, and Clipchamp, that allow users to split videos without needing to install any software. These tools typically work by uploading the video to a web-based interface and then selecting the desired start and end points for each clip. Some of these services offer additional features, such as text overlays and sound effects.

Dedicated video splitting tools: There are also several dedicated video splitting tools available, such as MPEG Streamclip, VirtualDub, and Avidemux [9]. These tools are designed specifically for splitting and cutting video files, and often offer additional features such as batch processing and video format conversion.

However, there are several issues that can arise when splitting videos using these methods and tools. One common issue is loss of video quality. When a video is split, the video codec needs to re-encode the video, which can result in a loss of quality. To minimize this loss of quality, it's important to choose a video splitting method or tool that supports the same video codec as the original file.

Another issue that can arise is syncing problems between the audio and video tracks. When a video is split, the audio track needs to be split as well, which can result in syncing issues if the splitting is not done accurately. Some video editing software, such as Adobe Premiere Pro, offer automatic audio syncing features to help address this issue [10].

Lastly, the time it takes to split videos can be a concern, especially for larger or high-resolution videos. This can be exacerbated by limited processing power or low disk space on the user's computer. To address this, some video splitting tools offer batch processing, allowing users to split multiple videos at once.

In conclusion, splitting videos is a common task that can be accomplished using a variety of methods and tools, including video editing software, online video editors, and dedicated video splitting tools. While these tools offer a range of features and benefits, they also present potential issues such as loss of video quality, syncing problems, and processing time. It's important to choose the right method or tool based on the user's specific needs and technical requirements.

In natural language processing (NLP), a keyword extractor is a tool that automatically identifies and extracts relevant keywords or keyphrases from a given text document. Python, being a popular programming language for NLP, offers several keyword extraction libraries, including Gensim, TextBlob, and NLTK [8].

Keyword extraction is an essential task in NLP that enables machines to understand the underlying meaning of a text document. The extracted keywords can be used to summarize the document, classify it into relevant categories, or retrieve it in a search engine.

The Python keyword extraction libraries use various techniques to extract keywords, including statistical algorithms, rule-based systems, and machine learning models. The features of these libraries may vary, but most of them provide options to extract single or multiple keywords, filter out stopwords, and calculate the importance or relevance of the extracted keywords.

Compared to other keyword extraction methods such as TextBlob, NLTK and Gensim, the Python keyword extraction libraries offer several strengths. They are open-source and easy to use, making them accessible to developers and researchers alike. They also provide options to customize the keyword extraction process, such as filtering out stopwords or using domain-specific dictionaries.

In conclusion, keyword extraction is an important task in NLP that enables machines to understand the underlying meaning of a text document. Python keyword extraction libraries, such as Gensim, TextBlob, and NLTK, provide various techniques for extracting keywords, including statistical algorithms, rule-based systems, and machine learning models. These libraries offer several strengths, including their open-source nature, ease of use, and customizability, making them popular choices for keyword extraction in NLP applications.

A video splitting application based on keywords can provide a more efficient and accurate way of extracting and organizing relevant video content than other existing methods and tools. The use of a video cutter, transcript extractor, and keyword extractor allows for a more targeted approach to identifying and extracting specific segments of video content that are most relevant to a user's needs.

Compared to traditional video editing software, which requires manual scrubbing through hours of footage to find relevant segments, a video splitting application based on keywords can quickly and easily identify relevant sections based on specific search terms. This approach is not only more efficient but also more accurate, as it eliminates the risk of overlooking important information due to human error.

Furthermore, the use of a transcript extractor and keyword extractor enables the application to identify and extract relevant content based on both audio and visual cues. This can be particularly useful for videos that contain important information that may not be visible in the visual content, such as spoken dialogue or text overlays.

The result of using a video splitting application based on keywords can be proven through several methods. One way is to measure the time saved in identifying and extracting relevant content

compared to traditional video editing methods. Another way is to compare the accuracy of the extracted content with manually extracted content to ensure that the application is accurately identifying relevant segments.

Overall, a video splitting application based on keywords has the potential to revolutionize the way we extract and organize video content, providing a more efficient and accurate approach to identifying and extracting relevant segments. The result is a more streamlined and effective way of working with video content that can save time and improve the overall quality of the final product.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the experiment and designing the sample; Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Cloud Server Coding

The most challenging part of my application development is to transfer my local codes to the cloud server, and one of the significant difficulties I faced while writing local code into a format that cloud servers can understand. This is primarily because of the differences in the infrastructure and software used by local computers and cloud servers.

Cloud servers are designed to work with a vast network of interconnected systems, and their operating systems and hardware are optimized for high-speed computing and data transfer. This is different from local computers that are often designed for individual use and may not have the same level of infrastructure optimization.

As a result, I must understand the specific requirements of the cloud servers and optimize their local code accordingly. I deal with this through research on the internet and the continuous trial and error and improvement of the code.

2.2. The Difference in Programming Languages

Another challenge I faced is the difference in programming languages and tools used by local computers and cloud servers. Cloud servers often use specialized tools and programming languages that may not be compatible with those used on local computers. Therefore, I must have a thorough understanding of the cloud server's programming languages and tools to ensure that their local code is compatible with the cloud servers.

Moreover, cloud servers are designed to be scalable, which means that they can handle a large amount of data and traffic. As a result, I also need to ensure that the local code on my computer can handle the scale of the cloud servers.

2.3. Shortage of Online Storage

The shortage of cloud server storage also poses a significant challenge for me. Cloud server storage is critical because it enables individuals and organizations to store and access data from remote locations, allowing them to access the data anywhere and anytime. However, the demand for cloud server storage has increased exponentially due to the rise of big data and the need for cloud-based services. This high demand has led to a shortage of storage capacity, making it difficult for computer scientists to manage and store large amounts of data.

One of the major difficulties in dealing with the shortage of cloud server storage is the cost. Building and maintaining data centers requires a significant amount of capital investment. Since the cost I could afford is limited, I must organize my codes to make them efficient and effective.

3. SOLUTION

Video splitting based on keyword extraction is a process of automatically identifying specific parts of a video that are relevant to specific keywords or topics. This process involves breaking down a video into smaller, more manageable parts, which can then be analyzed using natural language processing techniques to identify keywords and their corresponding timestamps. This process can be useful in a variety of applications, such as video search, summarization, and editing.

The solution to video splitting based on keyword extraction involves several steps and components, including a video parser, a keyword extractor, and a video splitter. These components work together to analyze the video content, identify relevant keywords, and split the video into smaller parts based on the identified keywords.

The video parser component uses API from assembly ai to extract the transcript of the video into paragraphs.

The keyword extractor component uses natural language processing techniques to identify relevant keywords or topics in the video content. This component can use pre-trained models or can be trained on a specific domain of interest. The keyword extractor used in this application is Keyword Extractor from the python library.

Once the relevant keywords have been identified, the video splitter component splits the video into smaller parts based on the identified keywords and their corresponding timestamps. The resulting parts can be used for further analysis or editing.

The overall system architecture of the application Video Editing Downloader takes a video file as input and outputs a set of smaller video clips that correspond to specific keywords or topics. The video parser reads the video content and provides it to the keyword extractor, which identifies the relevant keywords. The keyword extractor then provides the keywords and their timestamps to the video splitter, which splits the video into smaller parts based on the identified keywords.

In conclusion, video splitting based on keyword extraction is a useful technique for automatically identifying relevant parts of a video. This technique involves several steps and components, including a video parser, a keyword extractor, and a video splitter. By using natural language processing techniques, this system can identify keywords and topics in the video content and split the video into smaller parts based on the identified keywords.

The video splitting application based on keywords is a useful tool that can automatically divide a long video into smaller clips based on predefined keywords. This application consists of three components: "video cutter", "transcript extractor", and "keyword extractor". In this response, I will discuss each component and how it is implemented in the application.

Video Cutter

The video cutter is responsible for splitting the input video into smaller clips based on the keywords. To implement this component, Moviepy from the python library is used in this application. To cut a video into clips using Moviepy, the library should be imported and the video file should be loaded into the program. The start and end times of the video segments should be specified, and Moviepy would cut the video based on the time segment input.

```
def cut_video(in_filepath, start, end, out_filepath):
    my_clip = mp.VideoFileClip(in_filepath)
    clip2 = my_clip.subclip(start, end)
    clip2.write_videofile(out_filepath)
    return out_filepath
```

Figure 1. Screenshot of code 1

To split the video, the video cutter component first needs to locate the timestamps where each keyword appears. One approach to do this is by using the transcript extractor component, which will be discussed next. After obtaining the timestamps, the video cutter component can then split the video into smaller clips based on the start and end timestamps of each keyword.

Transcript Extractor

The transcript extractor component is responsible for extracting the transcript from the input video [15]. This component uses API from Assembly AI to transcribe the audio from the video into text. This service provides an API to extract the transcript from a video file. The API is useful in situations where you have a large number of video files that need to be transcribed. It can be used for a variety of purposes, including creating captions for videos, generating subtitles, and extracting text from videos for analysis. It can also be used to create transcripts for video interviews or conference calls. The API is fast, accurate, and can handle a variety of file formats, including MP4, AVI, and MOV. Overall, Assembly AI's API is an efficient and effective way to extract transcripts from video files.

```
transcript_request = {'audio_url': upload_response.json()['upload_url'], }
transcript_response = requests.post(transcript_endpoint, json=transcript_request, headers=headers)
print('Transcription Requested')

polling_response = requests.get(f"{transcript_endpoint}/{transcript_response.json()['id']}",
                                headers=headers)

while polling_response.json()['status'] != 'completed':

    sleep(20)
    polling_response = requests.get(transcript_endpoint + "/" + transcript_response.json()['id'],
                                    headers=headers)
    print("File is", polling_response.json()['status'])

# self.__transcript = polling_response.json()
paragraph_response = requests.get(f"{transcript_endpoint}/{transcript_response.json()['id']}/paragraphs",
                                  headers=headers)
paragraph_response = paragraph_response.json()
self.__transcript = paragraph_response
print(self.__transcript)
```

Figure 2. Screenshot of code 2

Keyword Extractor

The keyword extractor component is responsible for extracting the relevant keywords from the input video [14]. This component uses Keyword Extractor to identify the keywords that best represent the content of the video. A keyword extractor is a Python library that automatically identifies and extracts important words or phrases from a given text or document. These extracted keywords can then be used for various purposes such as text summarization, information retrieval, and keyword-based search.

To implement the keyword extractor component, the “Keyword Extractor” provided by python library is used in this application. This library provides pre-trained models and functions for this specific task. The keyword extractor component takes the cleaned transcript from the transcript extractor component as input, and then uses this library to extract the keywords.

```
def extract_keyword(self, text):
    keywordextracter = yake.KeywordExtractor(top=6, stopwords=None)
    keyword = keywordextracter.extract_keywords(text['text'])
    keywords = list(keyword)
    if len(keywords) == 0:
        return None
    return keywords[-1][0]
```

Figure 3. Screenshot of code 2

Connecting the components

To connect the three components, we need to define an interface or API that allows them to communicate with each other. The video cutter component takes the input video and a list of keywords as input, and then uses the transcript extractor component to extract the cleaned transcript and locate the timestamps where each keyword appears. The video cutter component then uses these timestamps to split the video into smaller clips. Finally, the keyword extractor component takes the cleaned transcript from the transcript extractor component as input, and then uses it to extract the relevant keywords.

In conclusion, the video splitting application based on keywords is implemented using three components: "video cutter", "transcript extractor", and "keyword extractor". These components use video editing libraries and python libraries to split the video, extract the transcript, and extract the keywords. The components are connected through an interface or API that allows them to communicate with each other. This application can be useful for video summarization, video indexing, and video search.

4. EXPERIMENT

4.1. Experiment 1

The first experiment is to test out if the application can function as designed. It is important to test the application on different devices and different internet to make sure that first, it can be run on all devices and won’t be affected if the user switches operation systems like windows and macOS, and second, the difference in details such as processing time and the fluency of the process must be recorded and analyzed to see how different devices affect the application. The experiment tests three devices. A Macbook and two windows laptops under different internet. The application would run on all three devices and would process the same video.

The experiment results show that the difference in devices like macbook and windows does not make a difference. The processed results are the same and the time it takes is nearly the same as well. However, different internet does make a difference in the processing time. A faster internet can process the video in a significant shorter time, a decrease of about 30%.

4.2. Experiment 2

The second experiment is to test out how much can processing a video over and over again improve its processing time. As I build the application, the process of videos is all on the online cloud storage, and the data of the video's process would be left in the storage. My idea is that as long as a video has been processed, when it is processed again, the remaining information can be used to greatly increase the speed of processing it this time. As far as I am concerned, this is worth trying a lot since it could be a way to lower the process time of videos. The experiment is set on one computer, and all it needs to do is to process the same video through the application again and again, recording the time it takes everything and see the difference.

The experiment tested on processing a two minute video shows that it takes 20 seconds to process the first time, and 15 seconds to process the second time. After that, the rest of the experiments which are 4 times all take about 16 to 15 seconds to process the video.

The results of the first experiment show that the application does work on all devices despite the difference of the process time, which exceeds my expectation. The processing time part would be something to work on in the future, but overall, it proves that the application is completed and correctly functioned.

The second experiment's results show that there would be a decrease in time of about 25% after the video is processed once, and after the video is processed twice, the rest of the process time remains the same. It is probably because of the way data is stored in the cloud. Although the time did not further shorten as the process of the video increases, it has reached my expectation that an increase in processing speed exists.

5. RELATED WORK

This paper is a survey on video summarization techniques that use semantic features, such as object detection and recognition, to summarize videos [11]. The main contribution is a comprehensive review of the state-of-the-art methods for semantic-based video summarization, including their advantages and limitations.

Compared to our work, this paper focuses on summarizing the entire video rather than splitting it based on keywords. However, some of the techniques discussed in the paper, such as object detection and recognition, can be used to extract keywords that can be used in our keyword extractor component.

The strength of this paper lies in its comprehensive review of semantic-based video summarization techniques, which can provide valuable insights for our work in terms of feature extraction.

This paper proposes an automatic video summarization technique based on graph modeling, which involves identifying key frames and connecting them using a graph-based representation

[12]. The main contribution is a framework for generating a summary video that preserves important information while minimizing redundancy.

Compared to our work, this paper also focuses on summarizing the entire video rather than splitting it based on keywords. However, the graph-based representation technique used in this paper can be used to visualize the relationships between keywords extracted by our keyword extractor component.

The strength of this paper lies in its graph-based representation technique, which can provide a more intuitive summary of the video compared to other techniques that simply extract key frames. This paper presents a technique for extracting keywords from lecture videos using speech recognition [13]. The main contribution is a method for identifying important terms based on their frequency and relevance to the lecture topic.

Compared to our work, this paper focuses on extracting keywords from lecture videos rather than summarizing them. However, the speech recognition technique used in this paper can be used to generate transcripts that can be used by our transcript extractor component.

The strength of this paper lies in its focus on extracting relevant keywords from lecture videos, which can provide valuable insights for our work in terms of selecting keywords that are important for the user.

6. CONCLUSIONS

The video splitting application is designed to automatically split a long video into shorter clips based on relevant keywords extracted from its transcript. The application is composed of three main components: a video cutter, a transcript extractor, and a keyword extractor.

The first component, the video cutter, takes the original video and splits it into smaller clips based on timecodes provided by the other two components. The transcript extractor generates a transcript of the video using API technology. The keyword extractor then scans the transcript to identify relevant keywords that indicate where the video should be split.

One of the main challenges of this application is to lower the time used to accurately identify keywords and cut the video into clips. In the tests, it is proven that the speed of processing videos will increase as the trains of the AI increase by identifying more and more videos.

The effectiveness of the application is demonstrated by comparing its performance to a manual video splitting process. In tests, the application was able to accurately split a two minute video into shorter clips in an average time of ten seconds, while it takes a human an average time of one minute to categorize the video's content. While the application is not perfect, it provides a significant time savings over manual video splitting, and can be used to quickly create shorter clips for use in social media, marketing materials, or educational content.

One major limitation of cloud storage in a video splitting application is the potential for slow upload and download speeds. Since videos are typically large files, it can take a significant amount of time to upload or download them from the cloud server. Additionally, users may experience interruptions in their internet connection, which can disrupt the uploading or downloading process and potentially corrupt the data being stored. Another limitation is the security of the data. While cloud storage providers typically have robust security measures in place, there is always a risk that sensitive data could be compromised if a breach occurs. Finally,

cloud storage solutions may not always be cost-effective for users who need to store large amounts of data for long periods of time.

There are multiple ways of solving the lack of efficiency of the application. One of them is to simplify the codes so the server will take less time to process them. This is a task I will be working on in the future. Another way to increase the efficiency of video processing is to get a stronger server. Different servers have different capabilities of running codes. With the help of a stronger server, the time that is needed to process the video will definitely be lowered.

REFERENCES

[1] Malone, Ruth E., and Edith D. Balbach. "Tobacco industry documents: treasure trove or quagmire?." Tobacco control 9.3 (2000): 334-338.
[2] Freeman, Becky, and Simon Chapman. "Is "YouTube" telling or selling you something? Tobacco content on the YouTube video-sharing website." Tobacco control 16.3 (2007): 207-210.
[3] Carlin, D. B., et al. "Multichannel optical recording using monolithic arrays of diode lasers." Applied optics 23.22 (1984): 3994-4000.
[4] Feng, Peiyuan, et al. "A machine learning-based framework for modeling transcription elongation." Proceedings of the National Academy of Sciences 118.6 (2021): e2007450118.
[5] Mathur, Arunesh, et al. "Characterizing the Use of Browser-Based Blocking Extensions To Prevent Online Tracking." SOUPS@ USENIX Security Symposium. 2018.
[6] Intaglietta, M., and W. R. Tompkins. "Microvascular measurements by video image shearing and splitting." Microvascular research 5.3 (1973): 309-312.
[7] Grundmann, Matthias, et al. "Efficient hierarchical graph-based video segmentation." 2010 ieee computer society conference on computer vision and pattern recognition. IEEE, 2010.
[8] Nadkarni, Prakash M., Lucila Ohno-Machado, and Wendy W. Chapman. "Natural language processing: an introduction." Journal of the American Medical Informatics Association 18.5 (2011): 544-551.
[9] Kim, Jin-Gyu, et al. "VirtualDub as a Useful Program for Video Recording in Real-time TEM Analysis." Applied Microscopy 40.1 (2010): 47-51.
[10] Bertolino, Pascal. "Sensarea, a general public video editing application." 2014 IEEE International Conference on Image Processing (ICIP). IEEE, 2014.
[11] Apostolidis, Evlampios, et al. "Video summarization using deep neural networks: A survey." Proceedings of the IEEE 109.11 (2021): 1838-1863.
[12] Ngo, Chong-Wah, Yu-Fei Ma, and Hong-Jiang Zhang. "Automatic video summarization by graph modeling." Proceedings Ninth IEEE International Conference on Computer Vision. IEEE, 2003.
[13] Kanadje, Manish, et al. "Assisted keyword indexing for lecture videos using unsupervised keyword spotting." Pattern Recognition Letters 71 (2016): 8-15.
[14] Firoozeh, Nazanin, et al. "Keyword extraction: Issues and methods." Natural Language Engineering 26.3 (2020): 259-291.
[15] Liu, Fei, Feifan Liu, and Yang Liu. "A supervised framework for keyword extraction from meeting transcripts." IEEE Transactions on Audio, Speech, and Language Processing 19.3 (2010): 538-548.