MOOC VIDEO: A VIDEO SEGMENT SEARCH ENGINE FOR MOOCS

BY

PARIMAL DEEP SINGH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Chengxiang Zhai

ABSTRACT

For my thesis, under the guidance of Professor Chengxiang Zhai, I have worked on building a search system for MOOC, which will give video results to users for the queries that they make. Whenever the user makes a search from the client, video results will be fetched from the server and displayed to the user. The user will also be pointed to relevant location or segment in the video. We get the text from the video, which is then indexed and used by the search engine. The search can be performed from the web interface, which shows the user videos results from the point of relevance. The results are also evaluated against a test data.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank Professor Chengxiang Zhai for suggesting the project and giving me the opportunity to work on it. I am grateful for his support throughout past two semesters, teaching me various concepts of Text Information Systems and guiding through the milestones. Finally, I would like to thank my parents, all of my peers and friends who have been encouraging and supportive throughout the duration of this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

MOOC stands for Massive Open Online Course. Wikipedia [1] states that a MOOC is an online course aimed at unlimited participation and open access via the web. In addition to traditional course materials such as videos, readings, and problem sets, MOOCs provide interactive user forums that help build a community for students, professors, and teaching assistants. MOOCs are a recent development in distance education, which began to emerge in 2012 [2] [3].

MOOC is a recent phenomenon, which has come up in last few years. There are new course providers that are coming up; existing providers are also increasing the courses that are being offered. Instructors and students are also increasingly realizing the importance of imparting education online and this seems to be where future lies.

With growing popularity of MOOC, there is a need to cater to different kind of audiences. They should be able to find the most relevant courses for them. This creates the need to have a search system that can satisfy the need of varied audience and give appropriate results.

The user should also be able to search for a particular topic in a course too. They should be able to easily access the content, both in text and video form.

1.2 Purpose

The idea is to build a system that can help to search for MOOC videos. Most of the search engines currently work on text data. They crawl websites, scrape content in form of text and index them to generate search results. There is no notable software that deals with indexing the content of the videos. These days almost all online courses have video lectures. Even individual users on YouTube can upload videos with some educational content. The idea is to create a search engine that will give users video results in search. If the user wishes to watch some video tutorial for a particular topic, they can just search for it using text, and they will be returned a link to the relevant video. They can also be pointed to a segment in the video where they will find useful content pertinent to their query.

For now the system works for videos that are present on YouTube, but it can be easily extended to include videos provided by various MOOC providers. This will provide a single system to users to search for content in videos. The system can also be extended to work as a generic Video Search Engine.

1.3 Summary of Work

For the purpose of the thesis, I investigated the presence and emergence of MOOC. I also studied different MOOC providers and features they have. I also compared and analyzed the characteristics and results of existing MOOC Search Engines and came up with a Search Solution that combines great features of the existing MOOC search engines and some novel features that a user might find useful.

There was a gap found in the Video Search Engines for MOOC. There is no satisfactory option for Video Search Engine that will offer users the results with exact location in the video for a query that they are looking for. There exist many video search engines that work on the metadata of the video and return the entire video to the user. The user needs to browse through the entire video to reach relevant part. This new Video Segment Search Engine

tries to solve this problem.

For creating the MOOCVideo, the Video Segment Search Engine, I did following steps. First, I downloaded a set of MOOC videos from YouTube. Then, I convereted the videos to images, which I then convert to text. Finally, Lucene indexes the converted text, which can be used to make search queries. I also build a basic web interface that the client can use to get the results. It is called Video Segment Search Engine because it segments the videos into parts and returns these segments to the users.

Different retrieval methods and parameters were experimented with and these have been discussed in the thesis. A number of useful findings were made and a useable system was built. There are scopes for improvements, which have been discussed in the later part of the thesis.

CHAPTER 2

MOOC

2.1 Benefits

The MOOC Guide [4] lists 12 benefits of MOOCs:

1. You can organize a MOOC in any setting that has connectivity (which can include the Web, but also local connections via Wi-Fi).
2. You can organize it in any language you like (taking into account the main language of your target audience).
3. You can use any online tools that are relevant to your target region or that are already being used by the participants.
4. You can move beyond time zones and physical boundaries.
5. It can be organized as quickly as you can inform the participants (which makes it a powerful format for priority learning in e.g. aid relief).
6. Contextualized content can be shared by all.
7. Learning happens in a more informal setting, at a place of your convenience and often around your own schedule.
8. Learning can also happen incidentally thanks to the unknown knowledge that pops up as the course participants start to exchange notes on the courses study.
9. You can connect across disciplines and corporate/institutional walls.
10. You dont need a degree to follow the course, only the willingness to learn (at high speed).

11. You add to your own personal learning environment and/or network by participating in a MOOC.

12. You will improve your lifelong learning skills, for participating in a MOOC forces you to think about your own learning and knowledge absorption.

2.2 Challenges

The MOOC Guide lists 5 possible challenges for collaborative style MOOCs:

1. It feels chaotic as participants create their own content.

2. It demands digital literacy.

3. It demands time and effort from the participants.

4. It is organic, which means the course will take on its own trajectory (you have got to let go).

5. As a participant you need to be able to self-regulate your learning and possibly give yourself a learning goal to achieve.

2.3 Providers

There are many MOOC providers [5] right now offering online courses. Some of the major ones are:

1. Coursera

2. edX

3. Khan Academy

4. Udemy

5. Udacity

6. Canvas Network

7. Stanford Online

8. UPEx

2.4 Search Engine

Presence of so many MOOC providers leads to necessity of having a search engine, so that users can efficiently find relevant courses in the market. There exists many search engines for MOOC. In November 2012, TechCrunch dubbed one of these new resource sites a Yelp for open online courses [6] and that tagline seems appropriate. Some of MOOC search engines are:

1. Class Central

2. Mooctivity

3. Course Talk

4. Moocse

5. Knollop

6. CourseBuffet

7. Noodle

As the field of open online education grows, there is need for not only a MOOC, but also perhaps a MOOC-aggregator, for every learner. New courses begin each week, on multiple platforms, and the opportunities to learn are dizzying. The proliferation of search engines to make sense of all of the offerings certainly levels the playing field, expands access to all programs and will eventually help learners select courses based on other students reviews.

CHAPTER 3

EXISTING VIDEO SEARCH ENGINES

A video search engine [7] is a web-based search engine that crawls the web for video content. Some video search engines parse externally hosted content while others allow content to be uploaded and hosted on their own servers. Some engines also allow users to search by video format type and by length of the clip. Search results are usually accompanied by a thumbnail view of the video.

The search criterion used by each search engine depends on its nature and purpose of the searches. There are Metadata, Title and Description, Filename, Tags that can be used by the search engine to index the videos. Metadata is information about facts. It could be information about who is the author of the video, creation date, duration, and all the information you would like to extract and include in the same files. It can be internal or external. Tags are words that will be used by search engines as a basis for organizing information.

The usefulness of a search engine depends on the relevance of the result set returned. While there may be millions of videos that include a particular word or phrase, some videos may be more relevant, popular or have more authority than others. This arrangement has a lot to do with search engine optimization. Most search engines use different methods to classify the results and provide the best video in the first few results. However, most programs allow you to sort the results by several criteria like relevance, date of upload, number of views, user rating, etc.

Many existing search engine seems to work on comparing the images together [8]. Until recently, the management of large image databases has relied exclusively on manually entered alphanumeric annotations. Systems

are beginning to emerge in both the research and commercial sectors based on 'content-based' image retrieval, a technique which explicitly manages image assets by directly representing their visual attributes. The Virage image search engine provides an open framework for building such systems. The Virage engine expresses visual features as image 'primitives.' Primitives can be very general (such as color, shape, or texture) or quite domain specific (face recognition, cancer cell detection, etc.) [9]. While commercial video search engines such as Google and Blinkx rely mainly on text in the form of closed captions or transcribed speech [10].

CHAPTER 4

MOOCVIDEO: NEW VIDEO SEARCH ENGINE

The existing popular video search engines for MOOC do not try to leverage the fact that text on the slides can be used to get information about the data. The user is normally shown the link of the video and they need to watch the entire video or search the relevant parts of the video themselves. Some of the results returned to the user may also not be useful.

For the New Video Search Engine, I try to extract the text from the slides on the video and use it to index the data. This is done by segmenting the videos into frames in a regular interval of time and then getting the content from it. The segmentation also helps in finding out where the relevant material appears, so instead of showing only the video link to the user, the user can be pointed to the relevant segment in the video. So, rather than going through the entire video, the user can watch only the relevant part of the video.

This system was created by downloading MOOC videos from YouTube. Right now, only a set of videos was downloaded. But in future, this system can be extended to automatically download videos from YouTube or other MOOC providers. Once, the videos are downloaded, each video is segmented into frames using OpenCV. Once we have the frames, the images are converted to text using OCR (Optical Character Recognition). We use all the generated text at different segments of the video and feed it into Lucene search engine. Various retrieval methods for the search engine were experimented. Once all the information is indexed, we use the search engine to get the search results.

The client will make a search query from their browser, which will hit the server. The server is our Lucene search engine. As the indexer worked on the

segmented frames, the returned results are segment locations in the video. Once we have the set of answers, we display the results to the client. This is briefly how our search engine works. The following chapters will explain in detail how our new Search Engine functions.

So, for our system, input will be the query that the client makes, and the output will be the results returned to the user. These are the steps of real search engine. They need to be performed fast so that the user does not have to wait for a long time to get the results.

The process from downloading the videos, segmenting and converting them to frames, changing the frames to text and finally feeding the text to the Lucene search engine is the pre-processing step. We can do computing intensive steps in pre-processing. These steps like downloading the video, converting the video to frames and then text eventually, and indexing them by Lucene are performed only once. Even if these are slow, they will not affect the system too much.

CHAPTER 5

TECHNOLOGY USED

5.1 OpenCV

OpenCV (Open Source Computer Vision Library) [11] is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

5.2 OCR

OCR (Optical character recognition) [12] is the mechanical or electronic conversion of images of typewritten or printed text into machine-encoded text. It is widely used as a form of data entry from printed paper data records, whether passport documents, invoices, bank statements, computerized re-

ceipts, business cards, mail, printouts of static-data, or any suitable documentation. It is a common method of digitizing printed texts so that it can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as machine translation, text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

OCR is generally an "offline" process, which analyzes a static document. Handwriting movement analysis can be used as input to handwriting recognition. Instead of merely using the shapes of glyphs and words, this technique is able to capture motions, such as the order in which segments are drawn, the direction, and the pattern of putting the pen down and lifting it. This additional information can make the end-to-end process more accurate. This technology is also known as "on-line character recognition", "dynamic character recognition", "real-time character recognition", and "intelligent character recognition".

5.3 Tesseract

Tesseract [13] is an open source OCR engine available. Combined with the Leptonica Image Processing Library it can read a wide variety of image formats and convert them to text in over 60 languages. Tesseract works on Linux, Windows (with VC++ Express or CygWin) and Mac OSX. It can also be compiled for other platforms, including Android and the iPhone.

5.4 Lucene

Apache Lucene [14] is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. It is commonly used in both business and academia. It is Scalable, has High-Performance Indexing. It is Powerful, Accurate and has Efficient Search Algorithms and is a cross-platform solution.

Lucene can index any text-based information you like and then find it later based on various search criteria [15]. Although Lucene only works with text, there are other add-ons to Lucene that allow you to index Word documents, PDF files, XML, or HTML pages. Lucene has a very flexible and powerful search capability that uses fuzzy logic to find indexed items. Lucene is not overly complex. It provides a basic framework that you can use to build full-featured search into your web sites.

5.5 Ruby on Rails

Rails [16] is a web application development framework written in the Ruby language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. It allows you to write less code while accomplishing more than many other languages and frameworks.

5.6 Nodejs

Node.js [17] is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

CHAPTER 6

BUILDING THE SYSTEM

Videos are first downloaded from YouTube and stored with their metadata so that they can be processed. As mentioned earlier, the current system works only on videos downloaded from YouTube, but it can be easily extended to include videos from other content providers. The following steps explain in the detail how our New Video Search Engine was created.

The code for the system was written in Python and Java. It can be broken down into following parts:

1. Converting Video to Frames
2. Converting Frames to Text Files
3. Indexing the Text Files
4. Web Compenent to Display Search Results
5. Evaluating the Search Results

6.1 Converting Video to Frames

First we need to convert videos (mp4 files) into frames or images (jpgs) that can be processed. For this, OpenCV library was used. The code for this was written in Python. A video object was created and frames were obtained from it. The frames were pulled at intervals of 't' sec. Whenever a frame is extracted, it is checked with the previous frame for similarity. There are different template matching algorithms in OpenCV that can be used. Only non-similar frames are retained. This saves us processing too many documents later.

Design choices

The frame extraction was done every t = 10 seconds as it was found to be optimum. This does not lead to creation of too many pages and also does not skip slides of the lecture. Also, for template matching, `CV_TM_CCORR_NORMED` was used. It is one of the standard template matching method.

6.2 Converting Frames to Text Files

Once we get the frames, which are in form of images (jpeg), we need to convert them into text. For this, Tesseract OCR library was used. This takes an input image, processes it and converts the image into text. The frames created from the videos were fed to Tesseract and text was generated. This was ultimately stored in files with some metadata containing information about the Video Link, Video Title and the time when the frame appeared in the video. The code for this was written in Python.

Design choices

Tesseract was chosen as OCR library because it is open source. Other OCR libraries considered and tried were ABBYY OCR [18] and Free Online OCR [19]. ABBYY gave better results in terms of conversion than Tesseract but it was slow as we had to use an API which sends the image to ABBYY's server and gets the results back. Also, ABBYY is not free and not open source.

6.3 Indexing the Text Files

Once we are able to get the data in form of text, we can use Lucene search engine toolkit to index the data. All the videos that are converted to text are indexed. We also run special analyzer on text to make all tokens lowercase, remove stop words based on standard list, have max token length of 35 characters and run porter stemmer on each token. The code for this was written in Java.

Steps 1-3 are preprocessing steps. These are done only once for each new video.

Design choices

Title and the text both are indexed with title given twice the weight of text. This was done because title most of the times has the crux of the content and as in most information retrieval tool considered more important.

6.4 Web Component to Display Search Results

The code for getting the search results from the indexed data was written in Java. The results can be seen from command line or the web interface. To manage the Java part, Maven, a software product management tool is used. Ruby on Rails and nodejs is used to setup the Web Interface. nodejs is used as the front end with Rails server.

Design choices

BM25Similarity similarity function was used currently. We can use LMDirichletSimilarity or TFIDFSimilarity as well but BM25Similarity gave best results. The experimental outcomes of these are mentioned in the later part of the document.

6.5 Evaluating the Search Results

A test data was created to evaluate the performance of the search engine. The code for it was written in Java. Evaluation was done by calculating the Mean Average Precision (MAP) score. Design choices for evaluation are mentioned in detail in a separate chapter.

CHAPTER 7

EVALUATION AND RESULTS

The evaluation of the search engine was done by using Mean Average Precision (MAP) scores. Mean Average Precision is the standard single number measure for comparing search algorithms. Average precision (AP) is the average of precision values at all ranks where relevant documents are found. Precision was calculated by using the formula: Precision = (Relevant results Retrieved)/ ((Relevant results Retrieved) + (Irrelevant results Retrieved)).

A test data was created which had tuples in the form (Q, D). Q is the query and D is the relevant result for the query. For our search engine, Q is a query text for a topic and D is the segment where the result occurs in the video. This data was created by manually going through the videos and finding out relevant segments for various topics. All these (Q, D) tuples were put in a file judgements.txt. Then, queries were run for all the Q that we collected against our search engine. The results that we got (Q', D'), were checked for the precision. D' was considered to be relevant if for Q = Q', D' appears in D within a certain time frame. For example, if segment D starts at time t and D' starts at segment t' in the same video, it was considered relevant if $|t - t'| \leq$ k. It was considered to be irrelevant if for Q = Q', D' does not appear in D within the above time frame. Average precision was calculated using the above formula and which in turn was used to calculate Mean Average Precision.

I created a data set of 50 Queries and relevant video segments for them. Then, the Search Engine was run on these 50 queries to find out the Mean Average Precision. The following table shows MAP for LMDirichletSimilarity, BM25Similarity and TFIDFSimilarity. k is the time difference used in the evaluation as explained above. t is the interval in sec we extract the frame from the video. Time taken to index videos for different t is also tabulated.

Table 7.1: MAP Values for LMDirichletSimilarity (k, t in sec)

	t = 1	**t = 5**	**t = 10**	**t = 30**	**t = 60**
k = 60	0.3467	0.3635	0.4082	0.3703	0.1920
k = 120	0.3832	0.4121	0.4351	0.4198	0.4135
k = 300	0.5111	0.5119	0.5236	0.5131	0.4869
k = 600	0.5886	0.5811	0.5966	0.5662	0.5443
k = 900	0.6136	0.6021	0.6267	0.6052	0.5965

Table 7.2: MAP Values for BM25Similarity (k, t in sec)

	t = 1	**t = 5**	**t = 10**	**t = 30**	**t = 60**
k = 60	0.3489	0.3694	0.4053	0.3653	0.1903
k = 120	0.3847	0.4207	0.4335	0.4160	0.4143
k = 300	0.5139	0.5250	0.5270	0.5137	0.4869
k = 600	0.5914	0.5900	0.6007	0.5668	0.5455
k = 900	0.6178	0.6121	0.6326	0.6075	0.5972

Table 7.3: MAP Values for TFIDFSimilarity (k, t in sec)

	t = 1	**t = 5**	**t = 10**	**t = 30**	**t = 60**
k = 60	0.3466	0.3653	0.40391	0.3706	0.1978
k = 120	0.3816	0.4171	0.4345	0.4104	0.4114
k = 300	0.5074	0.5150	0.5225	0.5089	0.4876
k = 600	0.5866	0.5859	0.5999	0.5600	0.5431
k = 900	0.6145	0.6104	0.6327	0.6024	0.5937

Table 7.4: Time to Index Videos (t in sec)

t = 1	**t = 5**	**t = 10**	**t = 30**	**t = 60**
2697.5	1161.7	847.1	706.8	548.7

Few things that we can observe from the above tables are:

1. The results from TFIDFSimilarity, BM25Similarity and LMDirichletSimilarity are comparable, with BM25Similarity performing slightly better.

2. We can also see that as the value of t decreases from 60 to 10 sec, the MAP score increases. This can be argued to happen because t is decreasing, we are creating more frames, so the possibility of skipping any frame of useful information is also decreasing. With a low t value we make sure that we will not miss any frame by mistake.

3. Lower value of t also leads to increase in the pre-processing time. Lower the value of t, higher is the number of frames that we need to process, higher is the computation involved, in terms of memory, disk and time.

4. Lower value of t might also slow down search, as indexer created will be of larger size and it will take more time to retrieve the results. These values though were not calculated.

5. Therefore, there is a tradeoff between the computation and accuracy. If we wish to have a very accurate system, and computation time and memory is not an issue, we should keep t as small as possible. For a real world practical system, we need to take an intermediate value of t.

6. If the value of t is decreased below 10 sec, the MAP scores start to fall. This can be argued to happen because we are creating too many unnecessary frames. These start to affect the accuracy of the system.

7. The optimal value of t is found to be 10 sec.

8. As the value of k increases, the value of MAP also increases. This happens because as k increases, we are having more allowance for a match between expected output and actual output.

9. As BM25Similarity gives best results, it is used for the final system.

CHAPTER 8

WEB SEARCH ENGINE

For experiments, I had a small data set of 10 videos from different domains like Computer Science, Physics, Psychology and Economics. Within Computer Science there were videos on Operating Systems, Bayes Network, etc.

Figure 8.1: Basic web interface

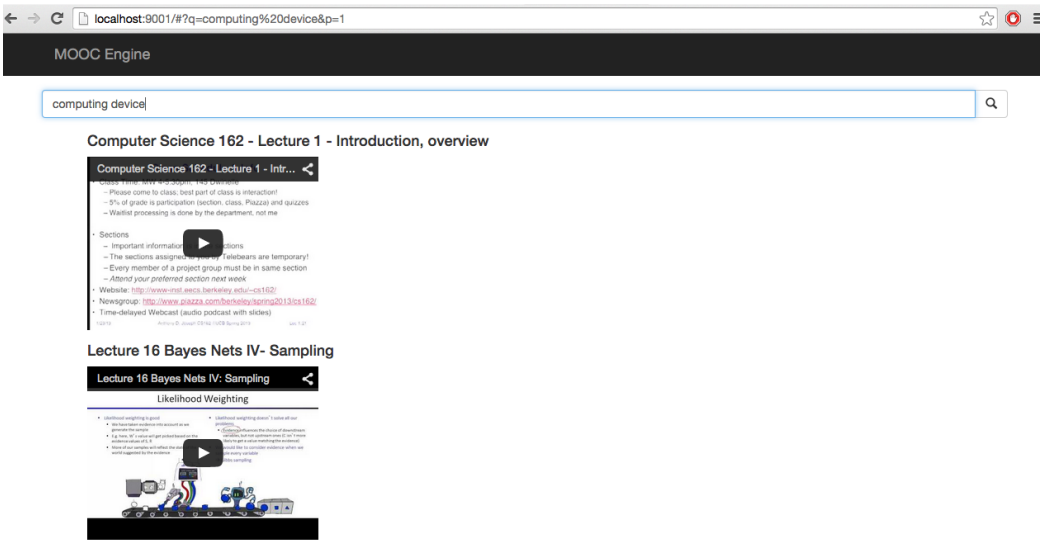

Figure 8.2: Results obtained for a query "computing devices"

Figure 8.3: Video when clicked, starts from the relevant point

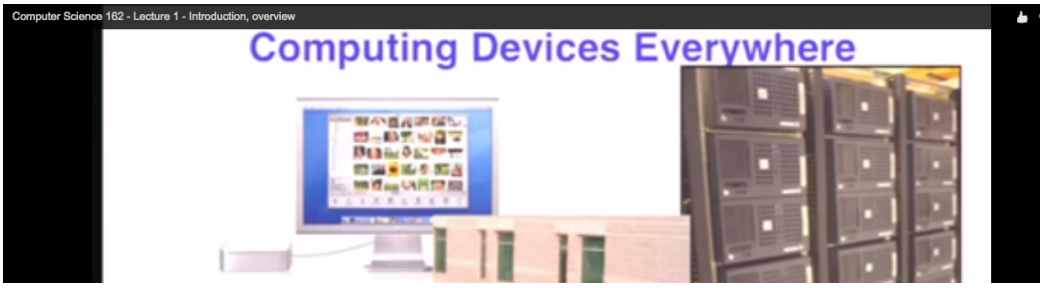

Figure 8.4: Video starting from relevant slide

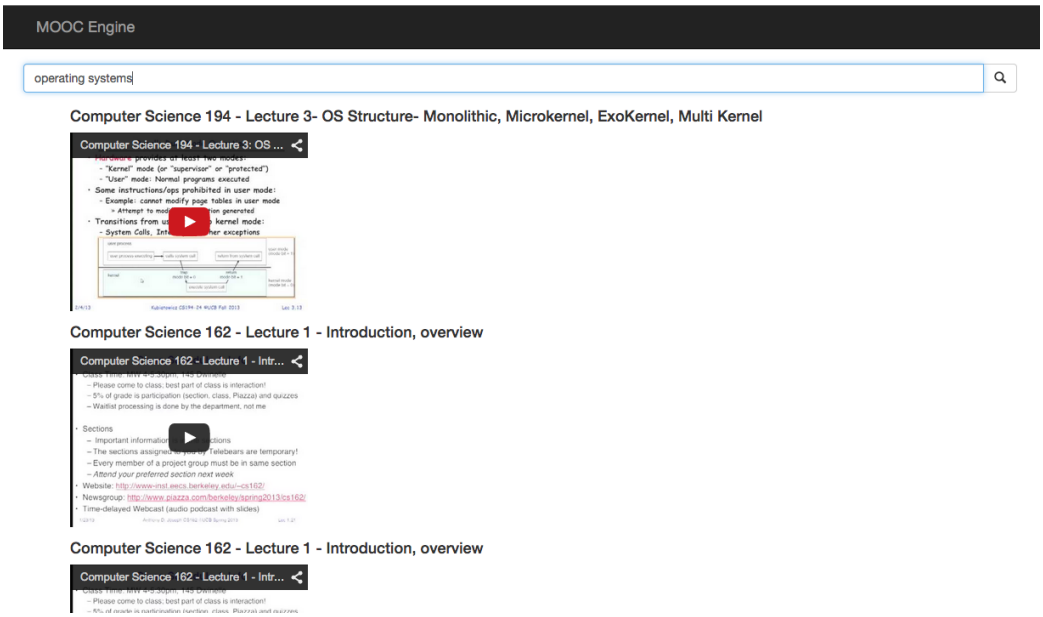

Figure 8.5: Results obtained for a query "operating systems"

CHAPTER 9

CONCLUSION AND FUTURE WORK

A search engine was created that can search for MOOC videos. The user or the client can search for videos of topics that they are interested in and they are also taken to relevant segment in the videos. The search engine works by converting videos into frames, which are then converted to text. They are eventually indexed and can be retrieved as search results. Different similarity functions gave comparable results on evaluation. Various template-matching algorithms were tried and different parameters for extraction of frames at various time intervals were experimented.

This system also helps in indexing video lectures so that the search engine can easily find their results. This system is highly useful for people who are able to search for the video by looking at the title or getting the result in a regular search engine, but are not sure of where to find the relevant content. The segmentation of videos solved this problem.

Some of the limitations and possible future work opportunities are:

1. The current system needs the videos to be downloaded manually. Future work could involve finding out MOOC videos automatically on YouTube from relevant MOOC channels and downloading them.

2. Also, the system works on YouTube videos, but it can be extended to include videos from various other MOOC providers. This will be useful for a large MOOC audience, which is growing exponentially.

3. Improvements can be made to the Web interface so that it can be used as a full fledged system. We can give option to filter the results or sort the results.

4. The search results are right now sorted based on the relevance found

by the system. It will be useful if more options can be provided to the user so that they can sort the results as required. Different options to sort can be: Date of the video, duration of the lecture, title, channel, etc.

5. The system can be integrated with MOOC search engines that give text results so that we can make a complete system to give text and video results to the user.

6. We can easily extend the system to search for generic videos, not only MOOC videos, depending on use cases. This will help us to make a general Video Search Engine.

REFERENCES

[1] "Massive open online course," 2015. [Online]. Available: http://en.wikipedia.org/wiki/Massive_open_online_course

[2] Laura Pappano, "The year of the mooc," Apr. 2014, The New York Times. [Online]. Available: http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html?pagewanted=all&_r=0

[3] Tamar Lewin, "Universities abroad join partnerships on the web," Mar. 2013, The New York Times. [Online]. Available: http://www.nytimes.com/2013/02/21/education/universities-abroad-join-mooc-course-projects.html

[4] "Benefits and challenges of a mooc," Feb. 2013, MoocGuide. [Online]. Available: http://moocguide.wikispaces.com/2.+Benefits+and+challenges+of+a+MOOC

[5] "Swimming in a sea of moocs." [Online]. Available: http://www.evolllution.com/distance_online_learning/swimming-in-a-sea-of-moocs/

[6] Rip Empson, "Coursetalk launches a yelp for open online courses and what this means for higher education," Nov. 2012, Tech Crunch. [Online]. Available: http://techcrunch.com/2012/11/29/coursetalk-launches-a-yelp-for-open-online-courses-and-what-this-means-for-higher-education/

[7] "Video search engine," 2015. [Online]. Available: http://en.wikipedia.org/wiki/Video_search_engine

[8] John R. Smith and Shih-Fu Chang, "Image and video search engine for the world wide web," *Proc. SPIE 3022, Storage and Retrieval for Image and Video Databases V*, vol. 84, Jan. 1997. [Online]. Available: http://dx.doi.org/10.1117/12.263446

[9] Jeffrey R. Bach,Charles Fuller ; Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh C. Jain, Chiao-Fe Shu, "Virage image search engine: an open framework for image management," *Proc. SPIE 2670, Storage and Retrieval for Still Image and Video Databases IV*, vol. 76, Mar. 1996. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid =1015128

[10] Shih-Fu Chang, William Chen, Horace J. Meng, Hari Sundaram and Di Zhong, "A fully automated content-based video search engine supporting spatiotemporal queries," *IEEE Transactions on circuits ans systems for video technology*, vol. 8, Sep. 1998. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=718507

[11] "Massive open online course," 2015. [Online]. Available: http://opencv.org/about.html

[12] "Massive open online course," 2015. [Online]. Available: http://en.wikipedia.org/wiki/Optical_character_recognition

[13] "tesseract-ocr," 2015. [Online]. Available: https://code.google.com/p/tesseract-ocr/

[14] "Apache lucene core," 2015. [Online]. Available: https://lucene.apache.org/core/index.html

[15] "The lucene search engine," 2015. [Online]. Available: http://www.javaranch.com/journal/2004/04/Lucene.html

[16] "Getting started with rails," 2015. [Online]. Available: http://guides.rubyonrails.org/getting_started.html

[17] "Node.js," 2015. [Online]. Available: https://nodejs.org/

[18] "Abbyy cloud ocr sdk," 2015. [Online]. Available: http://ocrsdk.com/

[19] "Free online ocr," 2015. [Online]. Available: https://www.newocr.com/api/

APPENDIX A: SETUP

1. We can go to `https://github.com/parimaldeep/mooc-se` and download zip or clone the project on our server or desktop.

2. We need to install OpenCV. Homebrew can be used to install it on Mac.

```
$ brew install cmake
$ brew install ffmpeg
$ brew install opencv --python27 --ffmpeg
```

3. Then we need to install libpng too. MacPorts can be used to install it:

```
$ port install xorg-server
$ port install xorg-libXmu
$ port install libxml2
$ port install libpng
```

4. Next we need to setup Tesseract. Homebrew can be used to install it on Mac.

```
$ brew install tesseract
```

5. Videos that we want to index should be placed in `{root}/ocr/data/`. `{root}/ocr/data/videoNames.txt` should contain the video URL in one line and video title in the next line.

6. Next we need to start the script to convert video to text form. This is done by running:

```
$ python {root}/ocr/videosToText.py
```

7. Then we need to create search engine using Lucene.

```
$ cd {root}
$ module load apache-maven
$ mvn install --file server-pom.xml
$ mvn --file command-line-pom.xml package
$ java -jar target/GOOSE-CL-0.1-SNAPSHOT.jar --index
```

8. We need to install required gems:

```
$ cd {root}/web/
$ bundle install
```

9. We need to set up nodejs:

```
$ cd {root}/web/node/
$ npm install
```

10. Start the rails server:

```
$ cd {root}/web/
$ rails s
```

11. Start the jetty server:

```
$ mvn --file server-pom.xml package
$ java -jar target/GOOSE-WEB-0.1-SNAPSHOT.jar
```

12. Start the proxy:

```
$ cd web/node
$ node proxy.js
```

13. We can navigate to `http://localhost:9001/` and use the search engine.

14. Create a file at `{root}/judgements.txt` to write test data. The format is: First line: query text. Second line: space delimited list of relevant URLs. Repeat the same with other queries.

15. Once we get the judgements.txt file , we can run the evaluation script. To compile it and execute, run:

```
$ mvn --file eval-pom.xml package
$ java -jar target/GOOSE-EVAL-0.1-SNAPSHOT.jar
```

APPENDIX B: SAMPLE CODE

Listing 1: Converting videos to frames

```
def videoToFrameConverter(fileName):
    cap = cv2.VideoCapture('data/' + fileName)
    prevFrame = None
    msec = 0
    times = []
    while(cap.isOpened()):
        cap.set(0, msec)
        msec = msec + 10000
        ret, frame = cap.read()
        if ret == False:
            break
        if prevFrame is not None:
            res = cv2.matchTemplate(frame,
                prevFrame, cv2.TM_CCORR_NORMED)
            if res >= 0.99:
                continue;
        time = int(math.floor(msec/1000))
        cv2.imwrite('process/frame' +str(time)+ '.jpg',
                    frame)
        times.append(time)
        prevFrame = frame;
    cap.release()
    cv2.destroyAllWindows()
    return times
```

Listing 2: Converting frames to text files

```
def videoToTextConverter(videoLink, filename, num):
```

```
    youtubeLink = 'http://www.youtube.com/embed/'
    times = videoToFrameConverter(filename);
    for time in times:
        os.system('tesseract -l eng process/frame' +
            str(time) + '.jpg process/output'+
            str(time));
        with open('videos/videoText' + str(num)+ '-' +
            str(time) + '.txt', 'w') as outfile:
            videoID = videoLink.split('v=')[1];
            embedURL = youtubeLink + videoID +
                '?start=' + str(time);
            outfile.write(embedURL + '\n');
            outfile.write(filename.split('.mp4')[0] +
                '\n')
            fname = 'process/output' + str(time) +
                '.txt'
            with open(fname) as infile:
                for line in infile:
                    outfile.write(line)
        with open('videos/videos.txt', 'a') as vRfile:
            vRfile.write('videoText' + str(num) +
                '-' + str(time) + '.txt\n')
```

Listing 3: Indexing files

```
public static void index(String indexPath,
    String prefix, String fileList)
        throws IOException {
        FieldType _urlFieldType = new FieldType();
        FieldType _titleFieldType = new FieldType();
        FieldType _contentFieldType = new FieldType();
        IndexWriter writer = setupIndex(indexPath);
        BufferedReader br = new BufferedReader(
            new FileReader(prefix + fileList));
        String line = null;
        int indexed = 0;
```

```
        while ((line = br.readLine()) != null) {
                ArrayList<String> lines =
                    new ArrayList<String>();
                BufferedReader docReader =
                    new BufferedReader(
                    new FileReader(prefix + line));
                String docLine;
                while ((docLine = docReader.readLine())
                    != null) {
                   lines.add(docLine);
                }
                docReader.close();
                if (lines.size() < 3)
                        continue;
                String content = new String();
                for(int i = 2; i < lines.size(); ++i)
                        content += lines.get(i) + " ";
                Document doc = new Document();
                doc.add(new Field("url",
                    lines.get(0), _urlFieldType));
                doc.add(new Field("title",
                    lines.get(1), _titleFieldType));
                doc.add(new Field("content",
                    content, _contentFieldType));
                writer.addDocument(doc);

                ++indexed;
        }
        br.close();
        writer.close();
}
```

Listing 4: Searching files

```
private SearchResult runSearch(Query luceneQuery,
    SearchQuery searchQuery)
```

```
{
    try
    {
        TopDocs docs = indexSearcher.search(
            luceneQuery, searchQuery.fromDoc() +
            searchQuery.numResults());
        ScoreDoc[] hits = docs.scoreDocs;
        String field = searchQuery.fields().get(0);
        SearchResult searchResult = new
            SearchResult(searchQuery, docs.totalHits);
        for(ScoreDoc hit : hits)
        {
            Document doc = indexSearcher.doc(hit.doc);
            ResultDoc rdoc = new ResultDoc(hit.doc);
            String highlighted = null;
            try
            {
                Highlighter highlighter = new
                    Highlighter(formatter, new
                        QueryScorer(luceneQuery));
                String title =
                    doc.getField("title").
                        stringValue();
                rdoc.title(title);
                String contents =
                    doc.getField(field).stringValue();
                rdoc.content(contents);
                String url =
                    doc.getField("url").stringValue();
                rdoc.url(url);
            catch(InvalidTokenOffsetsException e)
            {
                e.printStackTrace();
            }
            searchResult.addResult(rdoc);
        }
```

```
            searchResult.trimResults(
                searchQuery.fromDoc());
            return searchResult;
        }
        catch(IOException exception)
        {
            exception.printStackTrace();
        }
        return new SearchResult(searchQuery);
    }
```

Listing 5: Evaluating results

```
    private static double eval(String query,
        String docString) {
            ArrayList<ResultDoc> results =
                _searcher.search(query).getDocs();
            if (results.size() == 0) return 0;
            HashSet<String> relDocs = new
                HashSet<String>(
                    Arrays.asList(docString.split(" ")));
            int i = 1;
            double avgp = 0.0;
            double numRel = 1;
            System.out.println("\nQuery: " + query);
            for (ResultDoc rdoc : results) {
                    String urlRes = rdoc.url();
                    boolean found = false;
                for (String urlRel : relDocs) {
                        String[] urlResA =
                        urlRes.split("start=");
                        String[] urlRelA =
                            urlRel.split("start=");
                        if (urlResA[0].equals(urlRelA[0])
                            && Math.abs(
                            Integer.parseInt(urlResA[1]) -
```

```
                        Integer.parseInt(urlRelA[1]))
                        <= 300) {
                        found = true;
                        break;
                    }
                }
                if (found) {
                    avgp += numRel/ i;
                    numRel++;
                    System.out.print("  ");
                } else {
                    System.out.print("X ");
                }
                System.out.println(i + ". " + urlRes);
                ++i;
            }
            System.out.println
                ("Average Precision: " + (avgp / (i-1)));
            return avgp / (i - 1);
        }
```