**PLATFORM FOR IMPROVING SEARCHABILITY AND INTERACTIVITY OF RECORDED LECTURES**

**Thesis Draft**

Damsith Karunaratne

(IT16037434)

BSc (Hons) in Information Technology

Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

August 2019

Declaration

“I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: …………………………. Date: …………………

The above candidate has carried out research for the bachelor’s degree Dissertation under my supervision.

Supervisor: Dr. Nuwan Kodagoda: ……………………………………….

Co-Supervisor Ms. Kushnara Suriyawansa: ………………………………

Date: ……………………………

Abstract

E-Learning has become commonplace and many leading Universities provide the facility to view pre-recorded lectures online. This approach gives learners the ability to follow lectures without time or location constraints and consume the lectures at their own pace. Despite their advantages, recorded lectures tend to be lengthy and tedious to watch. They also prove cumbersome when specific information needs to be extracted from them. Another drawback is that the lecture videos fail to show the connection between the lecture and its supporting material such as lecture slides and questionnaires.

Several platforms exist where videos can be edited to make them more interactive, however this is a time-consuming process. V-Tutor will automatically improve the interactivity and accessibility of recorded lectures in a few clicks. The system will take in raw lecture videos along with supporting material such as lecture slides and code samples. It will then carry out noise removal and optimizing on the raw video footage before matching the slides and code samples to occurrences in the video. Some novel features we plan on introducing are automatic generation and suggestion of questions based on the content and automatic video segmentation according to topics. The main objective of the V-Tutor system is to create a web platform which can add interactivity and accessibility to course material thereby improving learner engagement. The focus of this report will be the code matching component which automatically identifies the location within a lecture video which corresponds to a line of code in a particular lecture file.

Acknowledgement

The work described in this research paper was carried out as part of the final year research project for the subject Comprehensive Design Analysis Project. The completed final project is the result of combining all the hard work of the group members and the encouragement, support and guidance given by many others. Therefore, it is my duty to express my gratitude to all who supported us in this endeavor. I would like to thank Dr. Nuwan Kodagooda and Ms. Kushnara Suriyawansa our supervisor and co-supervisor, for the invaluable advice and guidance they provided. I am also extremely grateful to Mr. Jayantha Amararachchi, Senior Lecturer/ Head-SLIIT Centre for Research who gave and confirmed the permission to carry out this research. I also wish to thank all of my colleagues and friends for their support. Finally, I would like to thank all others whose names are not listed particularly but have given their support in many ways and encouraged us to make this project a success.

Table of Contents

[Declaration i](#_Toc19563632)

[Abstract ii](#_Toc19563633)

[Acknowledgement iii](#_Toc19563634)

[Table of Contents iv](#_Toc19563635)

[List of figures vi](#_Toc19563636)

[List of tables vii](#_Toc19563637)

[List of abbreviations viii](#_Toc19563638)

[1 Introduction 1](#_Toc19563639)

[1.1 1.1 Background Literature 2](#_Toc19563640)

[1.2 1.2 Research gap 3](#_Toc19563641)

[1.3 1.3 Research problem 4](#_Toc19563642)

[1.4 Research objectives 5](#_Toc19563643)

[2 2. Methodology 6](#_Toc19563644)

[2.1 2.1 Methodology for code matching component 7](#_Toc19563645)

[2.1.1 2.1.1 Frame extraction 7](#_Toc19563646)

[2.1.2 2.1.2 Building the Image Classifier to filter frames 8](#_Toc19563647)

[2.1.3 2.1.3 Matching Algorithm 12](#_Toc19563648)

[2.2 2.2 Commercialization aspects of the product 14](#_Toc19563649)

[2.3 2.3 Testing and implementation 15](#_Toc19563650)

[3 3. Results and discussion 17](#_Toc19563651)

[3.1 3.1 Results 17](#_Toc19563652)

[3.2 3.3 Discussion 19](#_Toc19563653)

[4 4. Conclusion 20](#_Toc19563654)

[5 References 21](#_Toc19563655)

List of figures

[Figure 1- High Level System Diagram 6](https://mysliit-my.sharepoint.com/personal/it16037434_my_sliit_lk/Documents/Thesis_Draft_Individual.docx#_Toc16584443)

[Figure 2 - High level architecture of code matching system 7](#_Toc16584444)

[Figure 3 - Extracting frames 8](#_Toc16584445)

[Figure 4- VGG16 model architecture 10](#_Toc16584446)

[Figure 5- Resnet model architecture 11](#_Toc16584447)

[Figure 6 - InceptionV3 architecture 11](#_Toc16584448)

[Figure 7- Pseudocode of code-matching algorithm 12](#_Toc16584449)

[Figure 8 - Example output of code matching algorithm 14](#_Toc16584450)

[Figure 9 -Proposed Business model 15](#_Toc16584451)

[Figure 10- Video review UI 15](#_Toc16584452)

[Figure 11- Homepage 16](#_Toc16584453)

[Figure 12 VTutor video playback page 16](#_Toc16584454)

[Figure 13- InceptionV3 comparison 18](#_Toc16584455)

[Figure 14- Resnet50 comparison 18](#_Toc16584456)

[Figure 15 - VGG16 comparison 19](#_Toc16584457)

List of tables

[Table 1: Comparison of existing products 4](#_Toc16584458)

[Table 2- Comparison of candidate models 17](#_Toc16584459)

List of abbreviations

|  |  |
| --- | --- |
| CNN | Convolutional Neural Network |
| IDE | Integrated Development Environment |
| LMS | Learning Management System |
| OCR | Optical Character Recognition |
| API | Application Programming Interface |
| S3 | Simple Storage Service |
| SNS | Simple Notification Service |
| URI | Unique Resource Identifier |
| ILSVRC | ImageNet Large Scale Visual Recognition Challenge |
| LCS | Longest Common Substring |

# Introduction

Today, e-learning has become an essential component of higher education for both teachers and students. According to a study on the effectiveness of e-learning on education, it was found that students nowadays are more satisfied with web enhanced learning when compared to a traditional classroom environment [1]. Therefore, it is common to see universities and higher education institutes adopting some form of e-learning to assist their students. Many institutes use their own customized version of a Learning Management System (LMS) to provide online course material.

Online education is beneficial for both students and teachers in many ways. For instance, as lecture content is always available online, the possibility of missing a lecture is low and teachers can ensure that students have access to course material irrespective of time and location [1]. Recorded lecture videos also enable students with different styles of learning and different levels of understanding to obtain a better grasp of the subject. For example, those who are familiar with the work can skip ahead to a section of interest while those that need more time to understand the concepts can pause and rewind to digest the lecture at their own pace [2, 3].

Usually, many LMSs enable lecturers to upload course material such as tutorials, lab sheets, lecture slides and recorded lecture videos. Whilst videos are more effective because they address both visual and auditory aspects of teaching [4], many students find it tedious to watch recorded lecture videos because of its duration, which normally lasts around 1 - 3 hours and its lack of interconnectivity and relevance to other course material [5]. Although there are many platforms for creating and editing videos which allow users to create interactive course material, the methods employed by them consume valuable time as they require the lecturer to manually edit the video.

Currently, to the best of my knowledge, there is no system which automatically identifies the relationship between different types of course material and enables the creation of interactive courses in a few steps. Hence, the main focus of our research project is on improving two aspects of course material: accessibility and interactivity. The goal is to develop an intelligent system capable of improving the interactivity and learner engagement of course material in just a few clicks. This thesis will examine the code-matching portion of the research.

## Background Literature

Live coding lectures, often seen in the field of Information Technology, usually feature a screen capture in which the instructor types code into a text editor or Integrated Development Environment (IDE) while narrating. A study conducted by Marc. J. Rubin, on the effectiveness of live-coding lectures, found that students exposed to live-coding lectures performed better when tackling large programming assignments [6].

Although lectures of this type have their merits, students often need to revisit a specific point in a lecture and to do this they must search through the video until the relevant section is found. This has been identified as a drawback and many researchers have tried to address this problem with varying degrees of success. In a research conducted in 2011, Kambathula and Iyer suggested a system which would enable automatic tagging of lecture videos to enable easy identification of the different sections. It achieves this by first performing text analysis on the audio extracted from the video and then creating a database of tags from the resulting transcript [7]. Their system can highlight portions in each video in response to user queries, allowing the user to navigate to an exact location in the video. Luca Ponzanelli *et al* in 2016 introduced CodeTube [8], a similar search engine which when given a query, returns self-contained fragments of the corresponding lecture videos. Their system can identify Java code in video frames by applying image processing techniques such as Optical Character Recognition (OCR) and shape detection as well as text analysis methods such as island parsing on subsections of each frame to generate a Heterogenous Abstract Syntax Tree (H-AST) which is used to identify coding constructs [9]. In a research conducted in 2018, a deep-learning approach which leverages Convolutional Neural Networks (CNNs) to classify the presence or absence of Java code in video frames is proposed [10]. Their system is able to achieve an average accuracy of 98% for this binary classification task using a trained VGG16 [11] Neural Network and represents a more scalable solution to identifying code in videos. However it is limited to identifying code in the Java language and cannot be successfully applied in a system which would analyze videos in many programming languages. Research on algorithms such as ResNet50 [12] and InceptionV3 [13] prove they are good candidates for this purpose.

## Research gap

Even though there are several products already available with similar objectives, they mostly focus on the use of manual processes that involve human intervention to make the content more interactive and increase the searchability. Our proposed solution aims to reduce the amount of human interaction needed for this process by introducing a platform which will analyze and augment the content automatically.

As illustrated in Section 1.1, much research has been conducted on identifying source code in video and image files. Although the topic area (matching each line of code in a sample code file to its occurrence in the lecture video) has not been widely researched and would most likely have an algorithmic solution, research that has already been conducted on source code mining and text detection can be used as a basis to create a system which identifies relevant portions of a live-coding video which correspond to the source code file. Table 1.1 is a comparison of the proposed system with existing systems in the market.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Features | LearnWorlds | Echo360 | TechSmith Relay | Our Solution |
| Matching lines in code samples to occurrence in recorded lectures | ✗ | ✗ | ✗ | ✔ |
| Automated segmentation of lecture video into topic units | ✗ | ✗ | ✗ | ✔ |
| Matching slides with the lecture video | ✗ | ✔ | ✗ | ✔ |
| Automated noise removal from the video | ✗ | ✗ | ✗ | ✔ |
| Automatic question generation | ✗ | ✗ | ✗ | ✔ |

Table . - Comparison of existing products

## Research problem

Nowadays, video lectures have become increasingly popular and many education institutes use Learning Management Systems that support video content. Whilst video lectures have benefits such as giving learners remote access to lectures, there are a few drawbacks such as poor searchability through the video and less interaction with the learner.

To overcome these drawbacks many lecture platforms have introduced tools such as web-based video editors that allow lecturers to add captions, divide the video into discussed topics, link lecture slides and embed questions into the video. However, these tools require human intervention which is time-consuming. When considering the domain of computer science, none of the available platforms provide a tool to map programming language code segments from source code files discussed in a lecture with their occurrences in the video.

Because there is a need for video lectures to be more interactive and searchable, and the fact that enhancing them in such a way requires a significant manual workload and time investment, a platform which will analyze and augment the content automatically will prove to be useful.

## Research objectives

The proposed system is a research study to improve the method of delivery for video lectures and increase the engagement of the learner. The main objective of the research is to develop an automated platform which provides a quick and efficient way for lecture creators to deliver video lectures which are more interactive and have increased searchability. However this document examines the code matching component of the above research. Hence the objectives are as follows,

* Create an algorithm to automatically identify the position in the lecture video where each line of code in a given related code sample is discussed so that the learner can reach that position on the video by using the lines of code in the code sample as an index.
* Increase the efficiency of the process of code identification by classifying frames which contain code and those that do not, thereby reducing the resources wasted on unnecessary frames.

# Methodology

A screenshot of a cell phone

Description automatically generated

Figure 2.1 - High Level System Diagram

Figure 2.1 shows a high-level system diagram. The system is comprised of cloud based Node.js[[1]](#footnote-2) microservices which form the backend and an AngularJS[[2]](#footnote-3) web application as the frontend. Upon uploading a lecture video with the appropriate lecture material such as slides and source code files, the system first generates a transcript and extracts frames from the video. The frames and transcript are then fed into the Topic segmentation, Question Generation, Code Matching and Slide Matching modules which generate metadata related to the video. The metadata consists of the timestamps corresponding to the matched slides, matched lines of code, generated questions and topic breaks in the video. The generated metadata is stored as a document in a NoSQL MongoDB database while the uploaded video is stored in an Amazon Simple Storage Service (S3)[[3]](#footnote-4) bucket. Finally the frontend web application will access this data using an Application Programing Interface (API) to provide an enhanced lecture viewing experience to the end users. This chapter will examine the methodology followed to develop the code matching module in detail.

## Methodology for code matching component

The code matching component performs the task of identifying the timestamp in the video where a particular line of code is discussed. This task can be broken down into the following steps,

1. Splitting the video into a set of frames which preserves the ordering of each frame in sequence
2. Filtering frames which contain code.
3. Obtaining a textual representation of the content in each frame
4. Iterating through each line in the source code file to find the earliest frame in the frame sequence which contains the line of code.

Figure 2.2 shows a high-level architecture of the code matching component.

A picture containing device

Description automatically generated

Figure . - High level architecture of code matching system

### Frame extraction

As a prerequisite to step 1, the video is uploaded to Amazon S3. Once the video has been uploaded, we utilize the Amazon Simple Notification Service (SNS)[[4]](#footnote-5) which is a highly available and fully managed pub/sub messaging service to notify the backend server that the video has been uploaded. The notification contains a Unique resource identifier (URI) of the uploaded video which is then used as an input for the ffmpeg[[5]](#footnote-6) library to extract frames. Figure 2.3 shows the javascript code to spawn a child process which calls the ffmpeg library to perform frame extraction. The video is sampled at 0.5 frames per second and each frame is named in sequence. This naming order is required to determine the timestamp of each frame.

A picture containing object

Description automatically generated

Figure . - Extracting frames

### Building the Image Classifier to filter frames

To match a code sample to its occurrence in the lecture video, text detection must be carried out on each video frame to extract a textual representation of it. Existing research in the fields of OCR and Text detection suggests that it is more efficient to use a Machine Learning model to detect candidate frames before running the OCR algorithm [8, 10] instead of wasting CPU cycles on frames which do not contain code. Based on this research we trained a machine learning algorithm to detect such frames. Considering the time constraints of the project we applied Transfer Learning techniques [14, 15] to repurpose an established machine learning model for this classification. The following sub sections describe the steps carried out to create this image classifier.

1. **Creating the Dataset**

The initial step of any supervised classification problem in machine learning is to collect a suitable dataset. Since at the time of writing this report there was no publicly available dataset with images that fall into the classes of *code* and *not-code,* the data had to be generated manually. Generating images was trivial given the abundance of lecture material available to us. However manually labeling each image proved to be difficult given the time constraints of the project. Therefore a python script was created to scrape the results of a google image search for each relevant class, download them to a file and ensure that they are valid. The next step was to remove visually similar images by resampling using a Lanczos filter [16] and extracting an average pixel level for each band in the image to create a hash representation of each image. The hashes were then compared to discard similar images. The resulting images were inspected manually to remove any images that were irrelevant to each class. As a result a dataset of 450 images of code and 450 images of other material such as slides which did not contain code was created in a relatively short time period[[6]](#footnote-7). The value of this dataset was further increased by utilizing the Keras pre-processing package [17] which performs random transformations and normalization on an image to generate multiple new images, thus increasing the amount of effective training and test data available.

1. **Reusing a pre-trained model with transfer learning**

Most deep learning models which solve complex problems need a vast amount of labeled data which, considering the time and effort required, can prove difficult to collect. Furthermore, training a deep learning model on a large dataset can take days or even weeks. However the application of knowledge gained from pre-trained models to solve related problems also known as *transfer learning*, can drastically reduce the effort needed to train a new model.

Supervised Deep learning models extract different features at different layers. Yosinski et al. In their paper [18], discuss how the lower layers of a neural network act as abstract feature extractors to detect features like edges and curves, while the dense layers towards the end identify features which are more specific to the task it was trained for. Therefore by freezing the weights of a robust pre-trained network and removing the last, fully connected layer, it can act as a feature extractor for similar classification tasks.

Many high-performing models have been developed for image classification for the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Most of the models developed for this challenge are available for free through various deep learning APIs including Keras. Transfer learning was carried out on the VGG16 [11], ResNet50 [12] and InceptionV3 [13], models which are typically used for most image classification experiments. Each model was compared based on its accuracy and validation loss to determine which model is suitable to use as a classifier.

1. **Comparison of architectures**

Several candidate deep learning architectures were analyzed to determine the best fit to classify images as containing code and not containing code. Each proposed model is introduced in this section.

VGG16 is a network which is characterized by its simplicity. The *16* stands for the number of weight layers in the network. Figure 2.4 taken from Simonyan and Zisserman’s paper [11] , depicts the model architecture.

A screenshot of a cell phone

Description automatically generated

Figure .- VGG16 model architecture

There are two major drawbacks when considering this model which are, the time taken to train the network and the size of the model. VGG16 is over 574MB large.

Resnet50, introduced in the paper “Deep Residual Learning for image recognition” by He et al [12], has a deeper architecture than VGG16. However at 102MB the model is substantially smaller in size due to the usage of global average pooling as opposed to fully-connected layers. Figure 2.5 depicts the layer arrangement of the Resnet architecture.

A screenshot of a cell phone

Description automatically generated

Figure .- Resnet model architecture

Szegedy et al. introduced the Inception architecture in their 2014 paper [13]. Figure 2.6 depicts a summary of the architecture. Out of all three models discussed, this model has the smallest size at just 96MB.

A screenshot of a cell phone

Description automatically generated

Figure . - InceptionV3 architecture

### Matching Algorithm

The deep learning model will identify the frames which contain code, thereby forming a set of images ***I****.* For each image ***i* ∈ *I*** the text contained within is extracted using the Optical Character Recognition capabilities of TESSERACT-OCR[[7]](#footnote-8) and stored in a set of text files ***T*** which preserves the file naming scheme of the extracted frames such that the chronological ordering of the frames is not disturbed. For example; *frame-0000.txt* refers to the first frame in the sequence.

The main intuition behind the algorithm is that the code will be discussed at the earliest time it occurs within the video. And therefore each frame should be analyzed from the end of the sequence to the beginning while looking for the best match. Figure 2.7 describes the proposed algorithm which determines the frame which is most suitable to use as the index. In the diagram, *T* refers to the set of text files extracted from the frames and *C* is the source code file which is to be matched.

A screenshot of text

Description automatically generated

Figure .- Pseudocode of code-matching algorithm

In the algorithm mentioned in Figure 2.7 the *gestalt()* functionrefers to a function based on Ratcliff and Obershelp’s “gestalt pattern matching” algorithm [19]. The algorithm works by first finding the longest common substring (LCS) from the two strings. Then it splits the strings into two parts, one to the left and another to the right of the common substring. Next the process is repeated, first for the left parts of both strings, and then the right parts. The process of finding the LCS is repeated recursively until the size of any split is less than a predefined value. Finally the similarity score is calculated using the following formula. takes a value between 0 and 1 where 1 means that the two strings match completely and 0 means that not even one common letter was found.

The execution time of the gestalt algorithm in the best case is andin the worst case. A variation of the algorithm which returns an upper on the value is implemented in the python *difflib* library [20]. It performs this by using the intersect of all common symbols in each string instead of recursively matching each sequence.

This reduces the execution time to in the worst case andin the best case at the cost of accuracy.

Figure 2.8 shows an example output of the code matching algorithm. Here, each line is provided with its corresponding timestamp location in the video.

**A close up of a map

Description automatically generated**

Figure . - Example output of code matching algorithm

## Commercialization aspects of the product

As shown in the literature survey and research gap, although some of the available commercial products provide similar features to those introduced in our platform, none of them provided the facility of automating the video enhancement process. The V-Tutor platform has promising business potential in the field of eLearning. A Few of the business models available for the platform are mentioned below.

* Standalone platform for video lectures provided as a SaaS product, where customers can obtain a license and use the features.
* A Free platform for video lectures with premium content that viewers must pay purchase or subscribe to view.
* Integration with existing MOOCs and LMSs as a plugin or API.

Figure 2.9 Shows the proposed business model for our product.

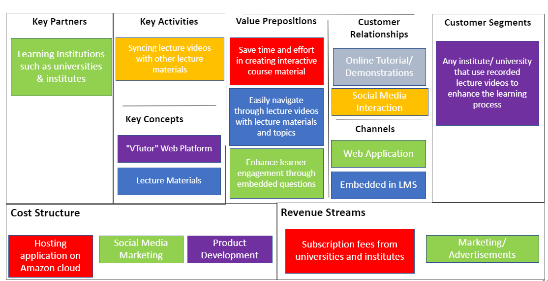


Figure . -Proposed Business model

## Testing and implementation

Figures 2.10 to 2.12 depict the user interface of the frontend web application.

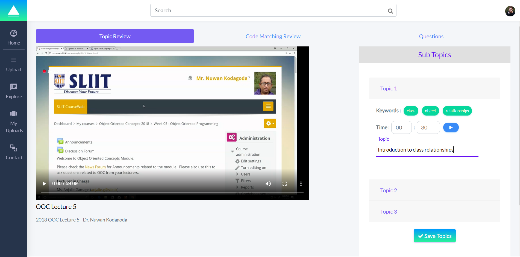


Figure .- Video review UI

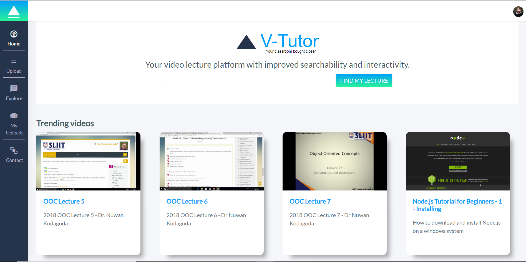


Figure .- Homepage

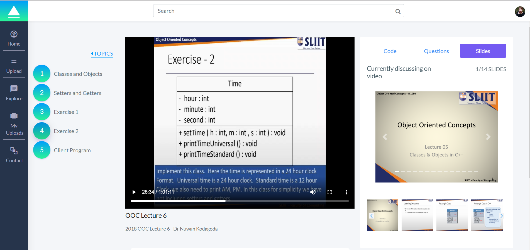


Figure 2.12 VTutor video playback page

# Results and discussion

## Results

For training the frame classification model, the same dataset of 900 images divided into the classes, “code” and “not code” was used. All training was performed without GPU acceleration on a laptop with 8GB of RAM and an intel core i5 8250u processor. Each model was trained for 20 epochs with 80 training steps per epoch and a batch size of 8. Table 3.1 compares the results obtained in training the 3 models on the dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Training time | Top 1 Validation accuracy | Top 1 Validation loss |
| VGG16 | 104s per epoch | 0.9387 | 0.1598 |
| Resnet50 | 26s per epoch | 0.8924 | 0.3351 |
| InceptionV3 | 96s per epoch | 0.8096 | 0.4187 |

Table .- Comparison of candidate models

Figure 3.1 compares the accuracy and loss curve for the InceptionV3 model. According to the graph, the model starts overfitting from the 3rd epoch onwards leading to a higher accuracy but also a high loss. Figure 3.2 compares the accuracy and loss curve for the Resnet50 model during training. It performs marginally better in terms of generalization and does not seem to overfit the data. Finally Figure 3.3 shows the accuracy and loss graphs for the VGG16 model. This model has proved to be the best out of the three models for this particular task as it has the highest accuracy as well as the lowest validation loss while not overfitting the data.

A close up of a map

Description automatically generated

Figure .- InceptionV3 comparison

A close up of a map

Description automatically generated

Figure .- Resnet50 comparison

A close up of a map

Description automatically generated

Figure . - VGG16 comparison

## Discussion

When considering the base models used to create the binary image classifier, VGG16 outperformed Resnet50 and InceptionV3. This could be attributed to the fact that VGG16 is a relatively simple model which requires very little tuning to work properly. The fact that the InceptionV3 based model started overfitting leads to the conclusion that more finetuning is required to get a viable result. For example, instead of freezing all convoluted layers, some of the deeper layers could remain trainable. Furthermore, regularization is another course of action which can reduce overfitting. The sub-optimal results may also be due to the fact that the dataset was generated manually in a short period of time.

# Conclusion

The current trend in online learning has led to many universities and higher educational institutes offering Learning Management Systems to deliver online course materials for their students. One of the commonly used techniques for online delivery of lectures is the use of recorded lecture videos. This comes with a host of benefits such as the ability to learn at one’s own pace and from the convenience of one’s home.

Given the benefits of recorded lectures they do have certain drawbacks when it comes to the efficiency of transferring knowledge. This is partly due to its lack of interactivity and searchability. Finding specific information within the video requires viewers to scroll through from the beginning. This problem has been addressed to a certain extent by certain lecture platforms by allowing the lecturer to create well organized interactive videos. However, most of these require either recording lectures using proprietary software or editing the videos manually which is a time-consuming process.

When considering the Software Engineering domain, one approach to increasing the searchability of lecture videos is to use the relevant source code files as an index where the viewer can locate the position in the lecture video where a line of code is being discussed just by clicking on the line. This paper explores the feasibility of such a system and the methodology which can be followed to create it.

The system, although restricted to matching source code can also be modified to match any other textual material with their occurrence in the video.

# References

|  |  |
| --- | --- |
| [1] | "Importance and Effectiveness of E-learning," 1 December 2015. [Online]. Available: https://higheredrevolution.com/importance-and-effectiveness-of-e-learning-9513046ed46c. [Accessed 6 March 2019]. |
| [2] | M. Brandsteidl, T. Mayerhofer, M. Seidl and C. Huemer, "Replacing traditional classroom lectures with lecture videos: an experience report," in *Proceedings of the 8th edition of the Educators' Symposium*, 2012. |
| [3] | M. Maher, H. Lipford and V. Singh, "Flipped classroom strategies using online videos," UNC Charlotte, North Carolina, 2013. |
| [4] | M. Chau, M. Lin, J. F. Nunamaker and H. Chen, "Segmentation of Lecture Videos Based on Text: A Method Combining Multiple Linguistic Features," in *Proceedings of the 37th Hawaii International Conference on System Sciences*, Hawaii, 2004. |
| [5] | Z. Woolfitt, "The effective use of video in higher education," 2015. [Online]. Available: https://www.inholland.nl/media/10230/the-effective-use-of-video-in-higher-education-woolfitt-october-2015.pdf. [Accessed 6 March 2019]. |
| [6] | M. J. Rubin, "The effectiveness of live-coding to teach introductory programming," in *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE ’13*, Denver, Colorado, USA, 2013. |
| [7] | V. K. Kamabathula and S. Iyer, "Automated Tagging to Enable Fine-Grained Browsing of Lecture Videos," in *2011 IEEE International Conference on Technology for Education*, 2011. |
| [8] | Luca Ponzanelli et al, "CodeTube: Extracting Relevant Fragments from Software Development Tutorials," in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE ’16*, 2016. |
| [9] | A. M. a. M. L. L. Ponzanelli, "StORMeD: Stack Overflow ready made data," in *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*, 2015. |
| [10] | J. Ott, A. Atchison, P. Harnack, A. Bergh and E. Linstead, "A deep learning approach to identifying source code in images and video," in *Proceedings of the 15th International Conference on Mining Software Repositories - MSR ’18*, 2018. |
| [11] | K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR,* vol. abs/1409.1556, 2014. |
| [12] | K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. |
| [13] | C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. |
| [14] | J. Brownlee, "A Gentle Introduction to Transfer Learning for Deep Learning," Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. [Accessed 1 March 2019]. |
| [15] | W. Nowak, "How to Train Your Model (Dramatically Faster)," Towards Data Science, 2019. [Online]. [Accessed 1 March 2019]. |
| [16] | J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 22, no. 8, p. 888–905, 2000. |
| [17] | "Image Preprocessing - Keras Documentation," Keras.io, 2019. [Online]. Available: https://keras.io/preprocessing/image/. [Accessed 05 Aug 2019]. |
| [18] | J. Yosinski, J. Clune, Y. Bengio and H. Lipson , "How transferable are features in deep neural networks?," in *27th International Conference on Neural Information Processing Systems*, Montreal, Canada, 2014. |
| [19] | Xlinux.nist.gov, "Ratcliff/Obershelp pattern recognition," 2019. [Online]. Available: https://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html. [Accessed 02 Aug 2019]. |
| [20] | "7.4. difflib — Helpers for computing deltas — Python 2.7.16 documentation," 2019. [Online]. Available: https://docs.python.org/2.7/library/difflib.html#module-difflib. [Accessed 12 Aug 2019]. |

1. https://nodejs.org/en/ [↑](#footnote-ref-2)
2. https://angular.io/ [↑](#footnote-ref-3)
3. https://aws.amazon.com/s3/ [↑](#footnote-ref-4)
4. https://aws.amazon.com/sns/ [↑](#footnote-ref-5)
5. https://ffmpeg.org/ [↑](#footnote-ref-6)
6. The dataset can be found at: https://www.kaggle.com/dammakaru/code-images [↑](#footnote-ref-7)
7. https://github.com/tesseract-ocr/ [↑](#footnote-ref-8)