LectureScribe

"Every Word Matters, Every Lecture Mastered"

Pranav Krishnakumar
Manipal Institute of Technology, Bengaluru
Computer Science and Engineering (Cyber Security)
Bengaluru, India
Pranav11.mitblr2022@learner.manipal.edu

Avi Mehta
Manipal Institute of Technology, Bengaluru
Computer Science and Engineering (Cyber Security)
Bengaluru, India
Avi.mitblr2022@learner.manipal.edu

Shayan Pradhan
Manipal Institute of Technology, Bengaluru
Computer Science and Engineering (Cyber Security)
Bengaluru, India
Shayan.mitblr2022@learner.manipal.edu

Vedansh Dua
Manipal Institute of Technology, Bengaluru
Computer Science and Engineering (Cyber Security)
Bengaluru, India
Vedansh.mitblr2022@learner.manipal.edu

Overview

LectureScribe is an innovative, easy-to-use, tool that streamlines the process of creating lecture notes. It uses OCR, speech recognition, and a top-of-the-line LLM model to create short, succinct, and superb notes.

Optical Character Recognition, OCR, Open-CV, PaddleOCR, Video Pre-processing, Google Speech Recognition, Gemini API

I. Introduction

A lesson that is taught in a classroom as 3 dimensions to it:

- Material presented by the instructor
- Notes written by the students based on their own interpretations.
- Colloquial verbal explanation, jolted down points on the board, which get erased after the duration of the class

Our product aims to solve the loss of the third dimension. A system which understands the presented material, the handwritten notes – and finally combines them to generate a well-put-together PDF of notes.

II. Ease of Use

A. Intuititve User Interface

LectureScibe embraces the term 'minimalism'. The product is aimed at young children, as well as teachers and learning adjacent faculties. Therefore, it is essential for the user-interface to be easy to navigate. There is one button for uploading an mp4 file, and another to convert it. The system automatically downloads the PDF copy of the notes on the user's device.

B. Error Handling

A perfect product with 100% efficiency is also an imaginary product. Keeping this in mind we have inculcated several error-handling and exception-handling mechanisms in place. These are, but not restricted to, error messages when fault in processing, not file is uploaded etc.

III. Transcribing words and senteces shown on the board

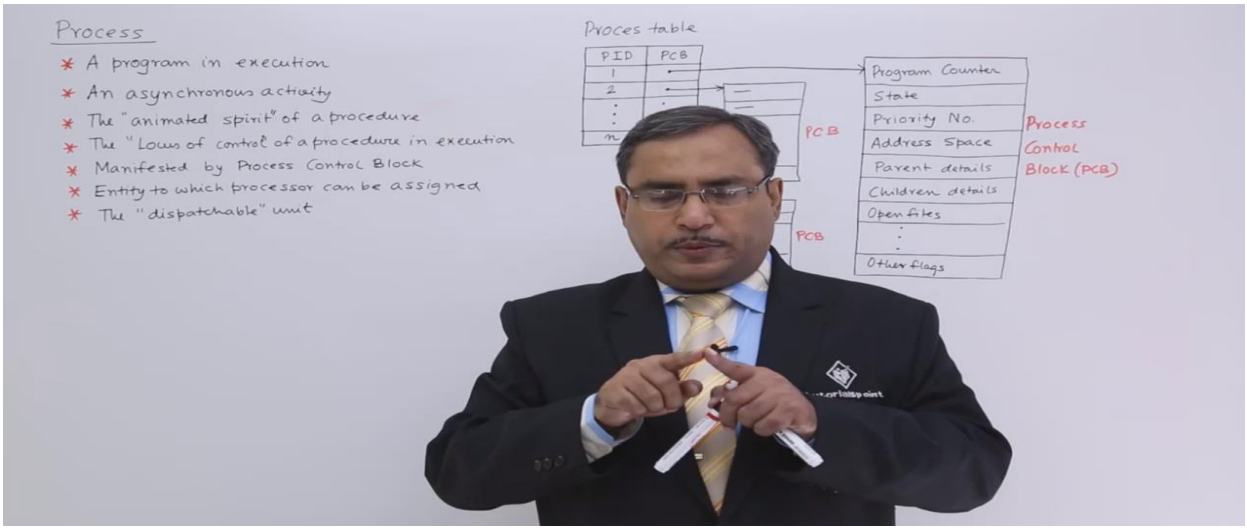
This is the first step that LectureScribe undertakes. The fundamental thing that one needs to understand about carrying out Optical Character Recognition (OCR) on a video is that: a video is simply a collection of frames. So, we can simplify our task by taking out 1 frame every X number of seconds, treating it as an individual picture - independent of any system – and reading any written material within it. Then finally concatenate all the material from the selected frames.

Now arises the question of: "What is the perfect value of X". A high value might risk omitting essential portions of the lecture, and lower values add load of the backend, there taking more compute-units and time.

After an analysis of various classroom lectures, we settled on a value of 15 seconds. Written content in a classroom generally doesn't change within this time as the instructor delves into the topic and needs to explain each line. So every 15 seconds a frame is extracted for the given lecture and analysed.

Now using PaddleOCR, which is an open-source library of Python, we will extract the hand-written content out of an image.

Demonstration of pre-processing and analysis will take place on this sample image



Pre-processing

Any person with experience in Optical Character Recognition will vouch for how haphazard our systems are without pre-processing. We undertake several steps to help the automated system understand what is written. These steps become even more paramount when we are reminded of the fact that most use cases will be handwritten. Each person's handwriting differs. Hence, stripping it down to its barest form - devoid of any color or variables - will greatly improve our output.

There are a variety of pre-processing steps but after a lot of testing we settled on the following: gray scaling, inverted binary thresholds, and applying gaussian blur.

A grayscale image is one that only contains shades of white i.e. in the range of black and white.

Binary Thresholding is the process of setting a pixel to 0 if it's value is below a certain threshold, and setting it to maximum value if above. Then finally, we invert the colors to obtain an inverted binary threshold.

Gaussian Blur is a mathematical function that smooths out pixel that it deems necessary.

After applying all these steps, we create contours around the detected text – making it easier for the software to identify where to start and end reading from. Within the contour we apply further smoothening to improve the quality of our output.

Text detection

The area within the contours are called the Region of Interest (ROI), we specifically instruct he OCR software just focus on it. Coordinates of all the contours are saved in arrays. On enumerating through them we get each of their positions.

Within each ROI we applied a sharpening function as a final processing-measure.

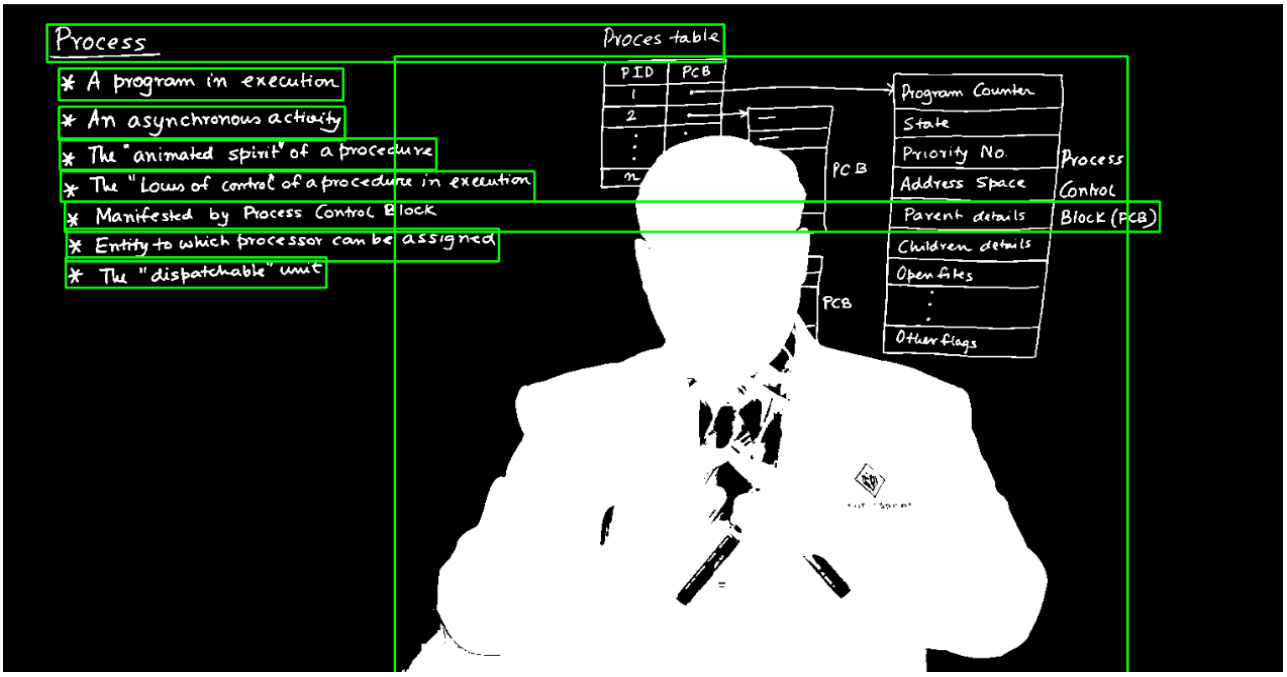
Each contour is on a different X-Y axis and the software haphazardly reads through them so we involved a function that reads it left-to-right (along the X axis). The project is primarily aimed at the English language and L-to-R matched the reading direction, making it easier for interpretation.

Then it's time to extract the text from the sharpened image using the built-in OCR function in PaddleOCR.

```
result = ocr.ocr(sharpened, cls=True)
```

At the beginning of the function we maintained a 'detected_text' string. After the aforementioned 'result' variable is found we append it to 'detected_text'. By doing so we create a large string which contains the extracted text from the video every 15 seconds. The function then returns this string which is the OCR extracted text from the entire video.

The pre-processed and contoured image, on which OCR is performed



IV. Audio Transcription

Now we move on to the 2nd facet of the product which – speech-to-text and creating a transcript of the entire lecture. We will be use the Google SpeechRecognition library.

First the MP4 needs to be converted into WAV format, it's a lossless and uncompressed format making it easier for the machine to understand. Even after manipulation the quality remains unchanged which is an important feature as we'll need to carry out some edits in the upcoming steps.

We go through the entire audio and wherever there's a silence longer than 700 milliseconds, we split it. Thus, creating "chunks". Processing these chunks for words makes it easier for the software, since there's no need for it to try and identify the end of a word/sentence.

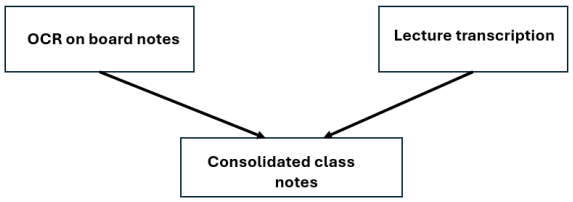
We enumerate through the array of chunks and use the 'recognize_google' function.

```
text = r.recognize_google(audio_listened)
```

Once the text from the chunk is extracted, it's appended onto a 'cumulative_text" string.

Finally, after all the chunks are enumerated through and the string is returned. This is the final transcript of the video.

V. Creating Final Notes



We have gone through the steps of extracting the text on the board or presentation and transcribing the entire lecture. It's time to combine them both and create lecture notes.

This task is quite a tricky one because it involves two different sources of information. One needs to understand the meaning and essence of the lecture. Take the useful parts of the transcript and extracted text, maintaining the lectures natural flow, context, level-of-teaching, and keywords.

So we employ the use of a Large Language Model (LLM). By writing a perfect prompt and feeding it the available information it will take no time to create the required output.

For LectureScribe, we have used Google's Gemini model. Specifically, the "Gemini-1.5-Flash". This model gave us the perfect combination of speed and accuracy. Within 5 minutes one can obtain an API key for the model and use it in their project.

In the prompt we include a detailed guide on what we need, the speech transcript, and the extracted text. Then we call the 'generate_content' function, which returns an output in a matter of seconds.

```
response = model.generate_content(prompt)
```

VI. Returning a PDF of Notes

Even the wisdom of God is no good if a person can't read it. This is the philosophy we used and convert the response returned by the model into neat, well-written notes. Then convert it into the PDF for easy writing.

The response we obtain is in a markdown format. We convert it to HTML using the markdown2 library. Once in HTML we implement CSS to convert it into a required font, add the required spacing, margin size, and page size – among many other things.
Then this is converted into a PDF and returned to the user.

An excerpt of the final notes delivered LectureScribe

The Process: A Deeper Dive

A process is a fundamental concept in operating systems, representing a program in execution. Here's a breakdown of its key aspects:

What is a Process?

- **A Program in Execution:** Unlike a program stored on a hard drive or pen drive, a process is a program that has been loaded into the computer's memory and is actively being run.
- **An Asynchronous Activity:** Processes don't execute on a fixed schedule. Instead, they execute when the system determines they are ready to run.
- **The "Animated Spirit" of a Procedure:** Processes aren't static entities. They have a dynamic flow of control, with instructions being executed in a specific sequence, often involving conditional branching and looping.

- **Manifested by the Process Control Block (PCB):** The PCB is a data structure that stores critical information about a process, including:
  - **PID (Process Identification Number):** A unique identifier assigned to each process.
  - **Program Counter:** Indicates the memory location of the next instruction to be executed.
  - **State:** Represents the current status of the process (e.g., running, blocked, ready).
  - **Priority:** Determines the process's priority in accessing resources.
  - **Parent Process ID (PPID):** Identifies the process that created the current process.
  - **Children Details:** Information about any processes created by the current process.
  - **Open Files:** A record of files currently open by the process.
  - **Other Flags:** Additional process-specific details.