Improving Access to Engineering Education: Unlocking Text and Table Data in Images and Videos

Uchechukwu Uche-Ike
University of Illinois at Urbana Champaign
uuchei2@illinois.edu

Lawrence Angrave
University of Illinois at Urbana Champaign

Abstract

Accessibility of media, including visual media, is a significant concern in creating engineering education that is inclusive and accessible; without access, students who are blind or have low vision are unable to learn from today's engineering materials. This project presents a new accessibility tool, implemented as a user-friendly browser extension to extract and create accessible structured text — with a focus on tables of information — when embedded within web-based images and video media. The app can extract information to be used in a speech-to-text output of a screen reader; or text-to-braille device; pasted into a programming editing environment (e.g., Matlab, Jupyter notebook, Microsoft Code, or text editor); and further, used as the initial source input for manual or automated construction of audio descriptions to accompany the original media for future dissemination. In addition to improving access to engineering education, this project is a productivity tool for students seeking more access to textual data presented in image form. For example, it serves as a tool for all engineers and student engineers who seek to extract and re-use tabular information embedded inside an image or video that otherwise would require manual entry. The system uses the React.js framework and Tesseract Optical Character Recognition (OCR) engine. The tool preserves privacy because it runs entirely inside the browser: no image data leaves the client. It can extract information from any web page on any website supported by the Chrome browser. Users can screenshot images and videos, extract text and numbers in scientific notation, determine the tabular structure, and meaningfully extract text from tables in tabular form (separated into rows and columns). We describe its design: the user interface features that allow its use by people with low-vision and access specialists. Upon initiation, the user selects a media HTML element to process; the user can choose to extract text from this frame or change the focus to another object on the page. The extraction area is adjustable by keyboard or pointer/touch interaction. Finally, the user chooses to either screen-capture the area and download the image, extract text from the area, or extract tabular information. To make the tool intuitive, interaction is structured as a wizard where users are guided along the stepwise process but can go back to previous steps. We provide examples of the best and worst output of our accessibility tool when applied to engineering education content and evaluate its accuracy and performance to extract tabular information from image samples from engineering disciplines.

Introduction

Engineering content consists of a wide variety of forms - including but not limited to - prose, images and figures, equations, charts and visualizations, programming code, and tabular information. A challenge of the inclusive education approach is to provide accommodations for students with disabilities and use technology to unlock access to these information-rich items. Unfortunately, textual information is frequently embedded in images or video, which is inaccessible to students who are blind or have low vision.

In this research paper, we introduce a new Chrome-based tool that can extract information from an image. Using an Optical Character Recognition (OCR) library, plain text can be machine-recognized from an image area. This work extends this text extraction to the extraction of structure and text of tabular information. This aspect is challenging because table layout and visual presentation can vary significantly. For example, there may be border lines between rows and columns, and column headings may span several lines.

Background

Optical Character Recognition (OCR) technology is a fundamental component of Text extraction tools. OCR operates by identifying text in an image, comparing the identified characters to a model of character features, and finally translating those recognized features into machine-encoded text. Popular OCR tools include Google Drive OCR, Tesseract, Transym, and OmniPage. Regardless of OCR application, they all share six principal steps in text extraction: **Image Acquisition, Pre-Processing**, Segmentation, Feature Extraction, Classification, and **Post Processing**.

Image Acquisition, the first stage, involves acquiring the picture and demarking the light from dark areas of the image. The light areas are treated as the background, while the dark areas are text. Pre-Processing follows this step. It takes the acquired picture and transforms it to ease character recognition in later stages. It includes several subprocesses including: binarization of the image: making it black and white; rotating the picture for improved horizontal alignment; noise removal: removing digital image spots and smoothing out edges; thresholding: cropping out unnecessary parts of the image.

Segmentation crops the pre-processed image to focus on the text portion of the photograph. The first part of Segmentation, Page Segmentation, removes parts of the image not containing text. Character segmentation then isolates individual characters for classification in the later stages, and the Image size normalization substage takes those separated characters and resizes them for the Feature Extraction stage.

In the Feature Extraction stage, the OCR application analyzes the isolated characters for distinguishing features: the tail of a capital Q, the stem of an L, and other minute details. The OCR software records these details for the penultimate stage: Classification.
Classification uses several machine learning algorithms: K- Nearest Neighbor (K-NN) classifier, Support Vector Machines (SVM) classifier, and Probabilistic Neural Network (PNN) classifier to classify the segments of the image into characters. The extracted text is then output to the user during the Post-Processing phase. During this phase, the machine-encoded text could be placed in a file and edited for grammatical or spelling errors.

Implementation [Research design]

Our application uses Javascript, the React.js framework and the Tesseract.js OCR engine. The React.js framework allows for a simplistic, dynamic, state-driven user interface without interacting directly with the DOM of a webpage. Tesseract.js is a free, open source OCR tool with customizable features for text extraction, including character allow lists/deny list, Page Segmentation Modes, and adjustable borders for text extraction.

In our text extraction application, the user right-clicks on a webpage and is prompted with a context menu to open the application. Upon clicking "open," the application window becomes visible and selects the first video/image HTML element on the screen. The user is guided through a wizard where they can choose whether or not to change the text extraction focus, adjust the frame of the focus to extract text from, screenshot the frame, and extract tabular information or text from the image. Finally, the wizard prompts the user to repeat the text extraction on the same HTML element or pick a new extraction focus.

Our Text Extraction Application follows this general flow chart (Figure 1): it uses the Javascript Canvas API to screenshot the extraction focus: a video or image. The application then crops the screenshotted image based on the user's input. The cropped image is reduced to two colors, given padding, and borders are placed around the image. These three techniques improve text accuracy. For tabular extraction, this additional preprocessing is omitted for performance sake, and an edge detection algorithm processes the image and returns the dimensions of the table and the dimensions of the individual table cells. The tesseract engine is used on each cell individually. For normal extraction, the app forwards the entire preprocessed image to the Tesseract OCR engine.

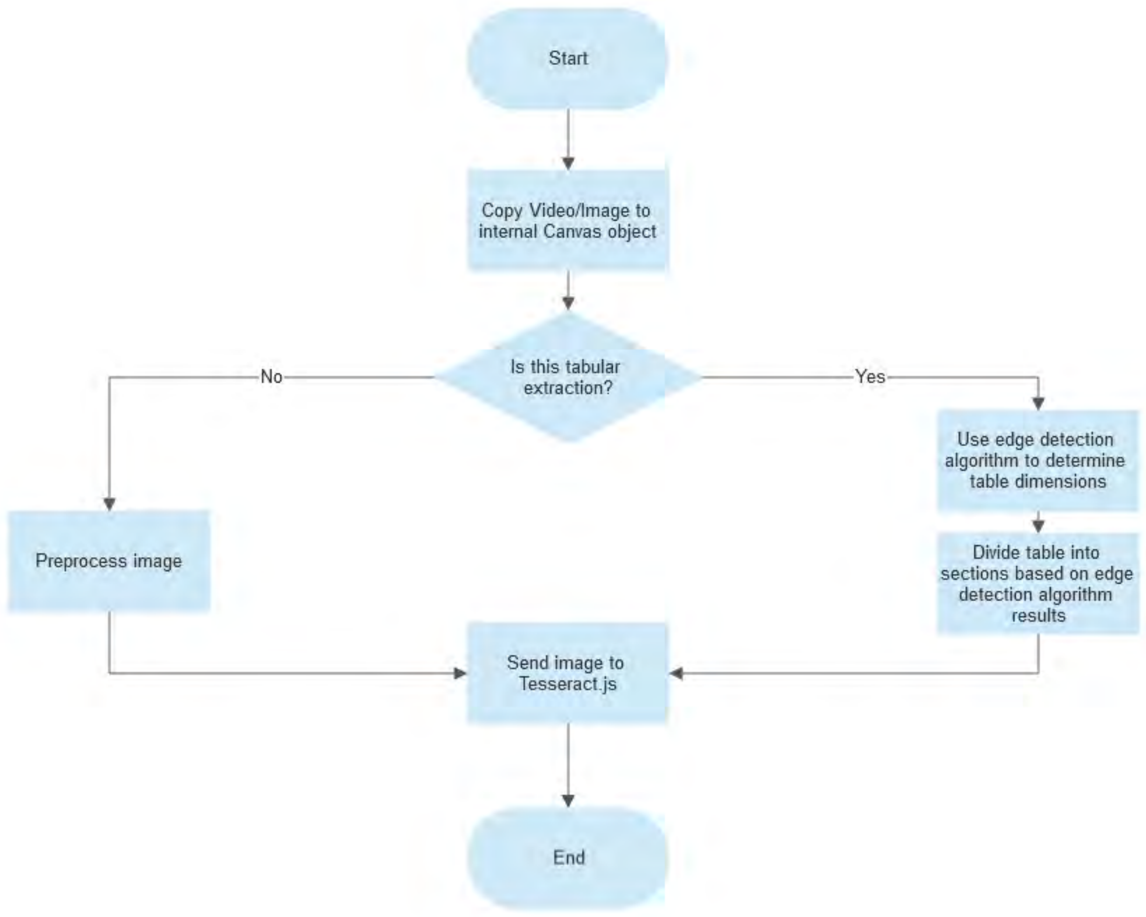

Start
Copy Video/Image to internal Canvas object
Is this tabular extraction?
No
Yes
Preprocess image
Use edge detection algorithm to determine table dimensions
Divide table into sections based on edge detection algorithm results
Send image to Tesseract.js
End

Figure 1

In the case of numerical text extraction, an allow list orders Tesseract to recognize exclusively characters standard numerical notation: "0123456789e*^." For normal text extraction, an allowlist is not used. The app then displays the extracted text. Tabular text can be displayed as an HTML table, CSV file, or JSON table.

Development began with linking Tesseract.js functionality with React.js. Tesseract allows users to insert several object types as parameters for OCR text extraction: Image URLs, filenames, and HTML canvas objects, among others. HTML's Canvas object is a convenient medium for text extraction as React.js can take a "screenshot" of an image or video in the website DOM. React.js can directly send that screenshot to Tesseract.js.

After linking React.js' functionality to Tesseract's functionality, the next concern became the tabular extraction feature. The principal issue was table formatting. Table formats vary wildly: cell boundaries, cell arrangements, and color schemes can all affect how an algorithm can interpret a table from an image. For the application's early stage, we assumed that tables have a uniform layout with a black-and-white color scheme.

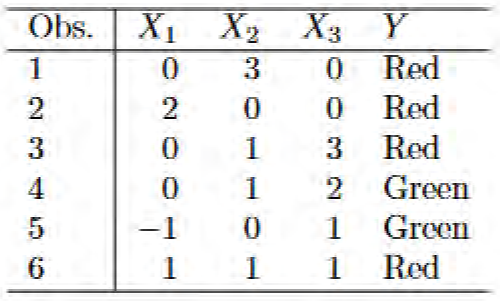
Our initial implementation of a tabular extraction used a K-means clustering algorithm to find areas in the image with the most amount of black pixels. The intuition was black clusters on the image would be where cell entries reside. Cell dimensions could be interpreted from the relative position of those clusters along the X and Y axes. We abandoned this approach after experiments found that the clustering algorithm required a significant amount of time and often misidentified the table edges as cell entries.

The improved methodology uses an edge detection algorithm that records into an array the number of times the pixel color changes significantly while scanning horizontally and then vertically across an image. The logic of this approach is that text in an image would cause a row of pixels to have more "edges" detected than space or cell borders on the table. The mode number of edges is found. Any rows/columns in the image that have more edges than the mode are treated as text subsections; those with a less or equal number of edges are non-text subsections. The app interprets the horizontal and vertical subsections as the rows and columns of the table; these subsections determine the position of the table cells in the image. The application sends the table cells' coordinates to Tesseract.js.

After solving the tabular extraction issue, the last major challenge with the "Extract Text as Numerical Value" functionality. Ideally, the feature would extract text in Scientific notation from images and return the value of the number in standard form as text. For example, the text "3.02e2" on an image would translate to "302" in the application output. However, initial versions of the application had text extractions for mathematical expressions such as 3.02^5 be recognized as 3.02M5 or 3.0275. The problem was Tesseract's default OCR model did not recognize the caret ("^") character. To rectify this issue, a new training model was created based on the existing Tesseract model for English with additional training data to help recognize the caret symbol.

Preliminary Testing

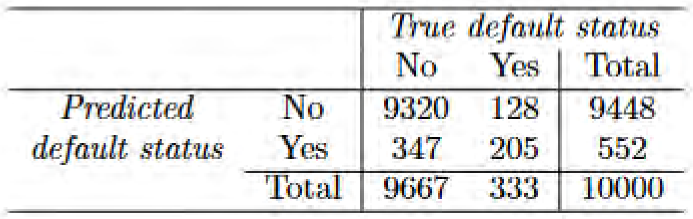
For testing, we looked into the accuracy of the tabular extraction feature of our proposed out. We ran our proposed text extraction app on several STEM-related course books. Screenshots were taken of the tables in these course books with generous amounts of whitespace around the tables. We then classified the screenshots into two main groups: Regular and Irregular. Regular tables contain a fixed number of rows and columns and do not have missing table entries. An example of a regular table is shown below in Figure 2.



Obs.	X_1	X_2	X_3	Y
1	0	3	0	Red
2	2	0	0	Red
3	0	1	3	Red
4	0	1	2	Green
5	−1	0	1	Green
6	1	1	1	Red

Figure 2

Irregular tables are tables that do not conform to the specifications of Regular tables: asymmetrical table dimensions, empty table entries, multiple header rows/columns, etc. An example of an irregular table can be shown in Figure 3.



		True default status		
		No	Yes	Total
Predicted default status	No	9320	128	9448
	Yes	347	205	552
	Total	9667	333	10000

Figure 3

The Regular table group further divides into two subgroups: Alphanumeric and Numeric. Alphanumeric tables contain both alphabetic and numeric characters. Likewise, alphabetic tables only have alphabetical characters in the table contents - alphabetical characters include greek letters.

To measure our app's word accuracy, we recorded four metrics. The word accuracy, the header accuracy, the data accuracy, and the extracted dimensions of the table. We calculate the word accuracy as the difference between the number of correct characters extracted by the number of incorrect characters extracted divided by the total number of characters in the original image. For numerical figures, a single-digit error results in the entire number being incorrect. Other artifacts from extraction, such as newline characters ("\n"), are ignored for accuracy measurement. Header accuracy and data accuracy are similarly defined but only consider text in the header and data sections of a table.

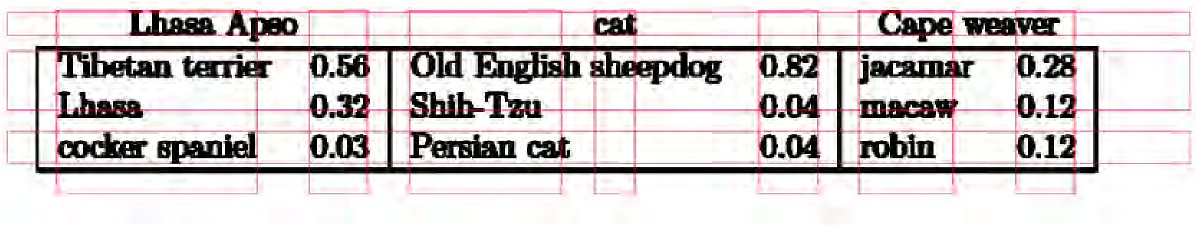
Results

Table Type	Word Accuracy	Header Accuracy	Data Accuracy	Expected Dimensions	Extracted Dimensions
Alphabetical					
Table 5	89.66%	0%	91.57%	5x2	5x2
Table 8	52.41%	42.55%	52.96%	10x6	2x11
Table 4	0%	0%	0%	4x4	N/A
Alphanumerical					
Table 2	86.66%	84.61%	100%	3x5	1x1
Table 3	18.64%	12.82%	75%	3x5	2x2
Table 4	73.13%	69.49%	100%	3x5	2x2
Irregular					
Table 2	83.09%	90.32%	37.5%	N/A	7x1
Table 1	13.73%	4.65%	37.5%	N/A	7x1
Table 8	0%	0%	0%	N/A	1x1

These are some highlights of the results of running our app on tables of each type of table from the college coursework.

Discussion

Overall, while our proposed app was sometimes accurate at deducing table dimensions, the accuracy of the extracted text was poor. We believe this is because the size of the tables from the engineering coursework was significantly smaller than those we used for our preliminary testing. Our edge detection algorithm described previously relies on differences in pixel color while scanning through the image. For tables with a dense amount of text or minute padding between table cells and their entries, our edge detection algorithm often interpreted multiple rows/columns as a single row/column (Figure 4). We furthermore believe this problem was exacerbated by our image binarization algorithm; while effective in improving the quality of larger images, for smaller images, we imagine binarization made deducing table dimensions harder for our edge detection algorithm. In scenarios where our edge detection algorithm accurately deduced table dimensions, the meager padding between the cell borders and contents hindered the Tesseract OCR software from correctly transcribing the image as text based on empirical data. Tesseract OCR would forgo some of the text altogether or misinterpret characters .



Lhasa Apso		cat		Cape weaver	
Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

Figure 4

The alphabetic tables had the highest accuracy overall. We assume this is because most tables in this category were larger and had ample space between their cells. Predictably, the irregular tables had the worst accuracy overall. Since these tables had merged rows/columns headers, our edge detection algorithm would erroneously split the headers into two among the aforementioned issues. The alphanumeric tables were a peculiar case because the Tesseract OCR technology would moderately accurately transcribe the text; however, because these tables were the smallest out of all groups, our edge detection algorithm had its worst performance deducing the table dimensions.

Conclusion

A means of extracting text embedded in media is crucial in creating a more accessible world for people in all educational disciplines. In this research paper, we proposed our application for extracting text from videos and images with several features such as screenshotting, text extraction, and tabular extraction. We furthermore tested the application on several real life instances of text and found some insightful results.

Further Work

Although our application's accuracy was disappointing, the results of our testing revealed opportunities for future potential improvements to our application. The critical step in our application's table extraction process was determining the table dimensions; any errors in this step propagate to the other stages in table extraction. To improve the issues with table dimension detection, we will evaluate a new algorithm that combines edge detection and cell border detection to deduce the table dimensions. The general strategy of this algorithm is to use the edge detection algorithm for larger tables where ample padding exists within cell dimensions and utilize cell border detection for smaller images. This proposed cell border detection algorithm will seek elongated horizontal and vertical lines in an image that is identical in the pixel color and assume those are the borders between cells.

Beyond automated dimension detection, our application could employ user input in finding table dimensions. The app would first run the previously described algorithms on an image and prompt the user to adjust the rows and columns using an interface similar to Microsoft Word (Adding Rows/Columns, Splitting Cells, etc). The application would then forward these dimensions to Tesseract.

To improve pre-processing of cells for Tesseract, we propose the idea of using each cell's contents and adding upscaling, padding, and other pre-pro-cessing effects to the cells similarly to how we described the general pre-processing . We had forgone these for table extraction as a compromise for performance; however, these tests have shown that for smaller images this pre-processing may prove indispensable for Tesseract to transcribe the text correctly.

References

Smith, Ray. "An overview of the Tesseract OCR engine." *Ninth international conference on document analysis and recognition (ICDAR 2007)*. Vol. 2. IEEE, 2007.

Tupaj, Scott, et al. "Extracting tabular information from text files." *EECS Department, Tufts University, Medford, USA* 1 (1996).

J. Memon, M. Sami, R. A. Khan and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)," in *IEEE Access*, vol. 8, pp. 142642-142668, 2020, doi: 10.1109/ACCESS.2020.3012542.

“What Is ...” *Amazon*, The University, 1978, https://aws.amazon.com/what-is/ocr/.