Brian Cornet
DSC 498
Dr. Donghui Yan
12/8/2020

Capstone Project I: Final Report

Working project title: Raw Image Scanner (RIS)

Introduction

This project aims to create a data collection method for camera images and videos of digital displays. Specifically, it is intended for a user to be able to provide an input (an image, video, etc.) of a source device's display (a television, monitor, phone screen, etc.) that contains some kind of quantitative data collection (a list or table of text strings, a collection of images, etc.) to collect and organize that data. The intent is to provide a quick means of copying structured information from one device to another for instances where accessing the raw data directly is inconvenient or prohibited. Typically, this restriction is enforced by design for closed platforms such as video game consoles (Nintendo Switch, Sony PlayStation 4, etc.) and iOS devices from Apple Inc., file formats that may be encrypted, or data streams that may not be recorded locally. Furthermore, many users lack the technical skills to access and manipulate arbitrary data structures directly and instead rely on existing software appropriate to the file format to do so. In all instances, the data itself is not private to the user, but any collection will have to be done by hand.

For example, a user may be playing a video game with the goal of collecting over 10,000 distinct items. The game provides a text list of every item obtained thus far (displaying 10 items at a time) but offers no hints towards which items are missing or how many items remain (i.e. a list of distinct items $N =$ {"Apple", "Ball", "Car"} with no indications as to whether "Apricot" or "Xylophone" are in $\overline{N}$ or how large $|N \cup \overline{N}|$ may be). Suppose the user wants to make a checklist of which items they currently had and cross-reference it with a list of known items obtained online to determine which items they're missing. While the data exists in the internal memory as a sequence of binary values (0 = missing, 1 = obtained), the internal memory itself is inaccessible to the user. The user also has thousands of items, many of which have similar names ("Wooden End Table", "Wooden Low Table"), are difficult to spell or type ("Myllokunmingia", "Übel/Equité"), or are simply long ("Archive Tower Giant Door Key", "Imperial Army Identification Tag"), making compilation by hand tedious and prone to errors. The user can easily record a video of themselves scrolling through the entire list with their personal smartphone, creating a digital basis for their data. By using optical character recognition (OCR) methods, a program could examine the frames of the video for text strings representative of every item the user has obtained, transforming their original digital basis into a more concise and appropriate data structure for the cross-reference operation.

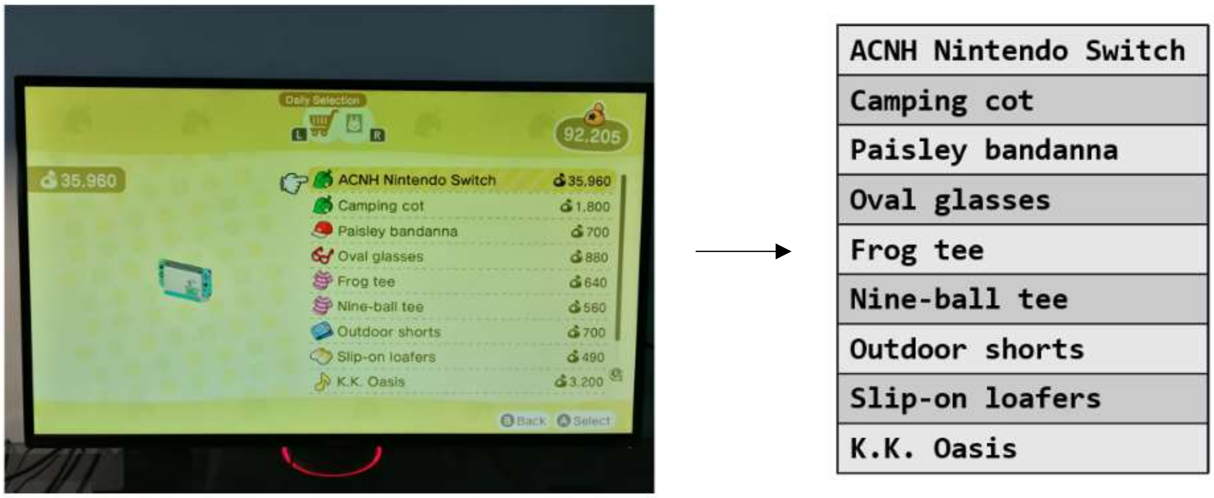

Figure 1. Converting an image of an item list into a set of strings. Image from Animal Crossing: New Horizons, Nintendo 2020.

Text-less images can also be used for comparison in a "spot the differences" problem. For example, a user is playing a video game with the goal of visiting every room in a large castle at least once. The game provides a representation of the castle's map that shows a visited room's position and hides unvisited rooms, as well as a percentage of the total rooms visited. Again, the digital data would represent each room with a binary value (0 = unvisited, 1 = visited) but cannot be accessed by the user. A clean image of the completed map is available online. The user knows they have a small number of unvisited rooms remaining but is unsure of which ones and exactly how many. Rather than try to detect which rooms are unvisited by carefully examining the differences between the image, the user could take a picture of their current map. This picture could then be rotated and skewed to make the orientation of the completed map, resized to fit its dimensions, and cropped to match its positions before both maps are converted into binary matrices representing each room. Assuming the images are both accurate, a difference between any two elements at a given coordinate would indicate the locations of a missing room, which could then be highlighted on either map.

A screen capture from any digital display would be preferable for an accurate interpretation over a photograph of a screen. Generally, most captured text and graphics will be static, smooth, properly aligned, correctly sized, framed, and free of environmental noise or errors. Camera images

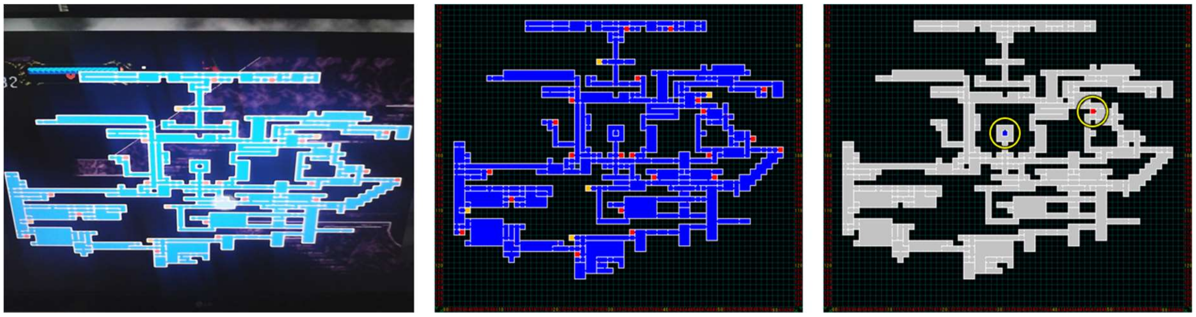

Figure 2. Detecting the differences between two images. Image from Castlevania: Symphony of the Night, Konami 1997.

meanwhile are subject to any number of errors caused by the angle of the lens relative to the screen, the lighting of the environment, objects outside the display's boundaries (including the display's frame), unsteady positioning when held by hand, detection of interlacing between frames and display pixels (moiré), or filters and corrections applied by the camera's software. However, one of the primary objectives of this software is to improve the convenience of collecting data, and not all users will have access to screen capturing and transfer methods. Comparatively, mobile phones equipped with cameras are extremely popular and convenient. Ignoring compatibility with camera images should thus be considered insufficient.

The bulk of programming work in this project will be conducted in Python due to the abundance of machine learning libraries associated with image recognition and correction. OCR for text-based images is achieved through Tesseract 4 and implemented in Python using the PyTesseract library. Per Tesseract's documentation, retraining methods are not explored as they are considered unlikely to help improve output accuracy. Instead, the image itself is transformed in various ways to improve quality and allow Tesseract to better "see" the appropriate text. Image correction methods are mandatory in the correction of camera images regardless and will be a primary focus of this project.

Initial OCR Experimentation

Though this software is intended to be able to compare images, its first goal is to successfully implement OCR methods for accurate text collection. Direct image comparisons should be relatively simple to implement following the completion of the OCR process given the overlap in image correction methods needed. As a result, the development of direct image comparison methods has yet to begin.

Initial OCR experiments were conducted on several images of different font. First, a control image "test.bmp" which contains black text on a white background generated in Microsoft Paint. The Consolas (bold), Calibri (bold), and Times New Roman (normal) fonts were printed in several locations with clear lines and spacing between each line. Figure 3 shows the original image next to the recognized text. Note that the result was slightly imperfect: an additional period or "." was erroneously detected after the word "yes", which was not detected until after adding the "yes" to the image. The "calibri?" line was also considered to be part of the Consolas block above it despite being closer to the Times New Roman block below it. Both errors suggest that Tesseract prefers text to be consistently aligned horizontally and vertically when presented with text spanning multiple lines.

Figure 3. "test.bmp" (with border added for visibility) and results.

Next was a test using a list from *Animal Crossing: New Horizons* (Nintendo 2020). The font in this image is very simple, perfectly aligned, and features a green font on a yellow background (except for a highlighted row which has brown text against a yellow and green striped background). The initial attempt produced little useful text: "@YPacanthostesa" for an icon of an animal skull and bone next to the word "Acanthostega" for the intended result area. By cropping the image to just the relevant text section, the results were able to correctly read the first and last entries "Acanthostega" and "Archelon tail", but nothing else. Selecting a single item in the middle of this list correctly produced the intended result "Anomalocaris", indicating that the issue was likely caused by the highlight bar. This shows the potential of cropping blocks of text to improve accuracy directly.

The final test used a list from *Monster Hunter Generations Ultimate* (Capcom 2018). The relevant text in this image is white on a dark background with some transparency showing background details. The font is fairly small with some overlap between characters with some artifacts around characters due to lossy compression. The first and third columns contain left-aligned word strings, the second and fourth columns contain right-aligned number strings with a preceding plus or minus sign, and the fifth column contains three symbols that can be circles or hyphens. The first two columns are also highlighted with a light brown background, the first row is highlighted with a green background, and the intersection is a slightly brighter green. Unlike the previous test, the accuracy of this test failed to reach 100% for a single row due to the difficulty in perceiving certain words ("Snowbaron" was interpreted as "Srowboron"), the mathematical symbols next to the numbers ("+1" interpreted as either "S" or "4"), and the inability to comprehend the circles and dashes in the last column as special symbols. This test demonstrates the importance of image correction even for captured images, as well as the need for uncommon symbol comprehension or interpretations of specific shapes or patterns.

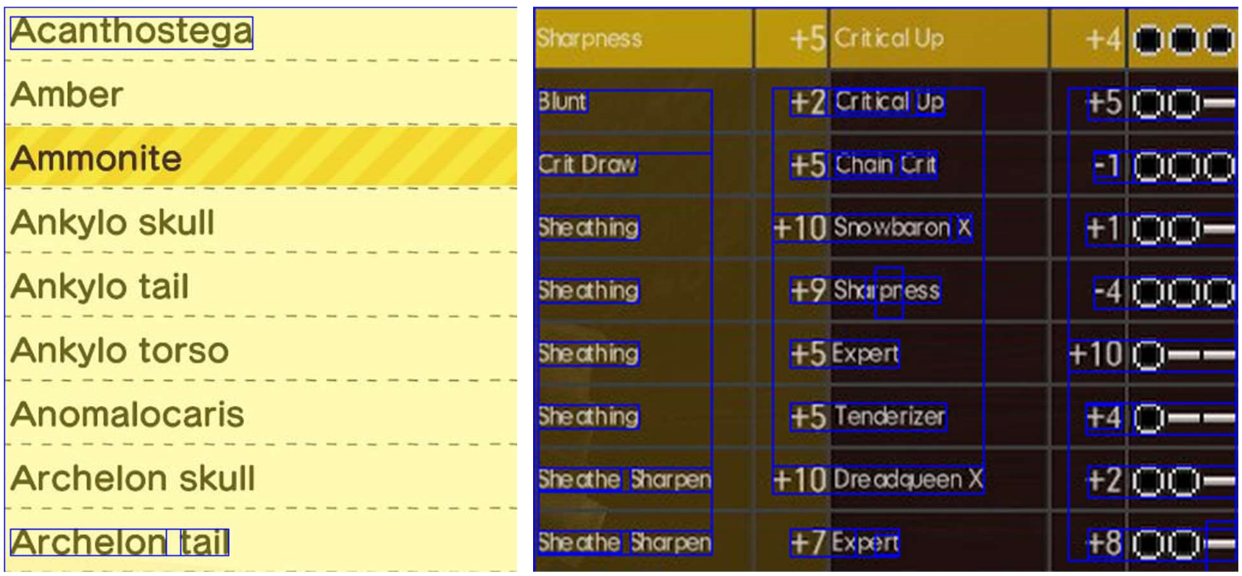

Figure 4. "acnh.png" (left) and "mhgu.png" (right) cropped with highlighting over text areas recognized.

Image Correction Methods

In addition to cropping, the accuracy of Tesseract's output can be improved through other methods of image correction. Several of the methods have been explored and implemented in the project's environment, though some remain to be fully explored. Combining several methods in varying orders has shown to produce different outputs, so an analysis of permutations may be necessary to improve output accuracy and time complexity.

Inversion

This is a simple process where the entire image is inverted -- black becomes white, white becomes black, and so on. If every pixel is represented by a string of bytes, then this can be achieved through a binary XOR operation on each individual byte with the constant bit string 11111111 (255 in decimal, 0xFF in hex). Tesseract's documentation specifies a preference for dark text on light backgrounds as seen in "test.bmp" and "acnh.png" in figures 3 and 4 respectively. This should improve the results of an image with light text on a dark background such as "mhgu.png" in figure 4.

Experimentations with “test.bmp” and “acnh.png” showed that their inverted images produced results with 0% total accuracy, whereas an inverted “mhgu.png” showed a significant improvement in accuracy for most words with a few instances of reduced accuracy for single characters at specific positions.

Rescaling

Based on experiments conducted by “Willus Dotkom” accessible through Tesseract’s Improve Quality page, Tesseract 4 works best when analyzing text where the height of capital letters (in pixels) is within a range of approximately 20 to 36 pixels depending on the font. Some fonts such as Helvetica-Narrow were observed to have errors within this range. To account for the possibility of unusual fonts, the program is designed to rescale the image to multiple pixel sizes and evaluate each font size independently. It may be worth investigating retraining methods for particularly unusual fonts as recommended by the Tesseract documentation. PyTesseract’s image_to_boxes() function provides the coordinates for each individual character’s start and end positions for each dimension. Finding the height for capital letters should be trivial except in cases where no capital letters are detected. Inclusivity for other tall characters such as “d” or “8”, verification of the first characters in words, and verification of characters that look similar in upper- and lower-case (e.g. “X” and “x”) may be warranted.

Binarization

This process assigns each pixel to be either black or white, providing a clear contrast between edges and making shape recognition easier. Tesseract natively performs this through Otsu’s method but recommends other methods for uneven background colors from the OpenCV and scikit-image modules.

One method explored was to separate the image into bitplanes by splitting the color values of each pixel into its own binary image. Input images are assumed to have a 24-bit color map, with 8 bits (1 byte) assigned to red, green, and blue color brightness respectively. Describing the image in an environment such as MATLAB or Python’s Numpy module typically results in a $M \times N \times B$ tensor of unsigned 8-bit integers representative of $M \times N$ pixels and B bytes. Reshaping these image tensors into a shape of $M \times N \times 8B$ binary values can be done with AND operations over each $M \times N$ matrix in B. The resulting images are bitplanes and can be displayed or examined individually. Note that the bit length of an image is irrelevant; this method is applicable to other formats such as 8-bit grayscale or 32-bit color images with transparency. Any n-bit color values can be split into n separate bitplanes. In most cases, the most significant bit (0x80) will provide most if not all of the relevant text while less significant bits instead contribute to noise. However, there may be cases where the detail provided from less significant bits may be relevant.

An experiment using bitplane decomposition on “acnh.png” demonstrates the effectiveness of this method in instances where certain text/background color combinations interfere with output accuracy. This image was originally unable to detect any words between the first and last lines due to the highlights included on the third line. By separating the image into bitplanes, the color changes applied to the highlighted row are conveniently ignored for the most significant red and green bits. Tesseract is then able to interpret the text with 100% accuracy from either bitplane. The results shown in figure 5 indicate the importance of the green color byte in this image as it had the highest average accuracy across its own subset of bitplanes. Meanwhile, the blue color byte is clearly identified as the cause of the original error seen in figure 4.

Figure 5. “acnh.png” separated into 24 bitplanes. Ordered least to most significant bits left to right.

Other methods

Several other methods are suggested by Tesseract’s documentation. Some of these methods have been implemented through manual processes such as cropping but were not formally included through sophisticated module methods. These include the following:

- Noise removal, where random variations in brightness are removed.
- Dilation and erosion, where edges of shapes are made thicker or thinner to increase visibility.
- Rotation, where a skewed image is aligned vertically and horizontally.
- Border removal, where noisy backgrounds are cropped to prevent being interpreted as characters.
- Border addition, where a solid border is added to help Tesseract designate a text area.

Proper use of these techniques will be essential to the success of OCR in camera images and are thus considered mandatory for this project’s future.

Conclusion

Converting images to simpler discrete data structures for comparison or storage should be possible with the methods described in this report. Tesseract 4 serves as a reliable platform for collecting text from images and will remain as the core platform for OCR implementation. The overall potential of this project relies on various image correction methods to produce reliable interpretations of source data, particularly for camera images. Several such methods have been investigated in limited contexts; other remaining methods will be explored in the future. Other considerations for future developments include the direct image comparison method and an implementation of machine learning algorithms to improve the accuracy of OCR with unusual fonts. Additional concepts not mentioned here may also be introduced as needed.