Headstone Photo Processing System

Alexander Franco, Minh Pham, Kobee Raveendran, Dax Borde, Steven Schilke

Department of Computer Science,
University of Central Florida,
Orlando, Florida, 32816-2450

***Abstract*—Matching processed headstone photos to official cemetery records is a time intensive task. The headstone photo processing system is an attempt to automate this process through the use of machine learning and optical character recognition to reduce the overall work required to complete this process.**

***Index Terms*—Computer Vision, Object Detection, Application Software**

I. Introduction

The Headstone Photo Processing System is a mesh of multiple software components designed to complete the task of record matching. The primary goal of our project is to automate the process of matching processed images of headstones to official cemetery records. That is, given a set of images from a cemetery, as well as a tabular spreadsheet of the complete records of that cemetery, our system should be able to map each image to a specific record in the table. On top of this, there is the added task of processing each image such that it primarily consists of headstone in focus, and ensuring that the headstone is upright and aligned in a natural fashion. These tasks, especially editing the images via rotation and cropping, become cumbersome for humans to do for a vast set of images, and can quickly overwhelm a single person. This was exactly the case for our sponsor, Dr. Giroux, who previously did all of these tasks manually for thousands of cemetery images. By devising a system capable of efficiently handling, at the very least, the general cases of headstones, the workload on Dr. Giroux, her colleagues, and others working in this field can be reduced significantly.

II. System Overview

A. User Interface

The component the user sees upon startup manages most of the information transfer between the user and software. The user interfaces with this component to provide crucial information such as the file path to the images the user would like processed, the border pixel length, image name formatting, and more. Every other major component of the project is abstracted from the user, making this system a black box with inputs (primarily) being the image set, and outputs being the processed images and spreadsheet with images matched correctly to existing records. Combined with a SQLite database for passing information to each component, the user interface frontend serves as the communication center between the individual components, making calls to the components to process the images.

B. Image Cropping and Alignment Pipeline

Once the input is provided, it is passed to the second major component, image cropping and alignment. In this component, we make use of a machine learning model to identify regions of interest where headstones are likely to be present. From these regions, we narrow down the set of regions to only those most likely to be the headstone in the foreground (as many images contain superfluous artifacts in the background, sometimes even other headstones). Once such a region is identified with a high degree of confidence, the image is cropped to fit only that bounding box, leaving us with an image in which the headstone in focus makes up most or all of the pixels. If a pixel border length was specified, this region would be expanded to accommodate it.

Fig. 1: The desired result of the Cropping and Alignment Pipeline

However, at any stage in this module, it is possible to encounter cases in which the model has difficulty completing any of these sub-tasks with a high degree of confidence. For example, in headstones where a roughly-rectangular shape is not present, or if the angle of the image is such that a rectangular rotation cannot be generated (i.e. headstones that appear steeply trapezoidal), the model may propose such regions with low confidence. These cases are handled by moving such cases to low confidence or ”failure” queue, in which the user will have to manually inspect each failed case. In a perfect world, our system would be able to handle any and all input. Unfortunately, however, minority edge cases have shown up that prove troublesome to handle with even the best methods. We thought it better to let the user handle such edge cases rather than risk matching incorrectly, which would require more exhaustive human inspection through the entirety of the records down the line. Once each image is either correctly cropped or marked as a failure case and handed off to the user, we proceed to the next stage: optical character recognition (OCR).

C. Optical Character Recognition

The OCR module is responsible for converting characters present in image data to plain-text, a form more suitable for indexing and searching in computers. The OCR module takes a single primary input, the path to the directory of cropped images, and has a set of outputs, a set of strings

consisting of the OCR model's predictions of words found in each image provided. A visual representation of the OCR module's input and output are shown in Figure (direct sample from our module). As this module is also composed of predictions, there will be headstones that lead the OCR module to predict somewhat incorrectly. The most common culprits of this occurrence are low-quality headstones that have typically experienced erosion due to their age. In such cases, the OCR will incorrectly predict some of the characters, or in extreme cases, in which even humans may have difficulty discerning characters, predict nothing at all. To handle such cases, we also push these troublesome headstones to the failure queue for the user to handle. For headstones where some characters are predicted with low confidence, we use a threshold function on the average confidence scores of each character (used in conjunction with fuzzy search, explained later) to determine whether to push the sample to the failure queue. This is done because our system does not require perfectly-extracted text from the OCR module in order to match correctly to records. In other words, our OCR module need not be perfect, but only good enough to maintain a certain degree of similarity between the extracted text and the true text found in the database table. With this, we move on to the text matching component of our project.

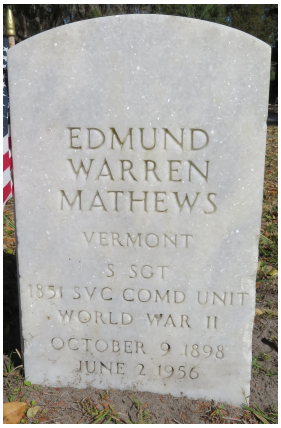
D. Fuzzy Search

String matching is one of the most critical points of our project, as it makes or breaks the functionality of the system. From the OCR component, we use the information extracted to query a SQLite database using approximate string matching, otherwise known as "fuzzy search". This component aims to best match the OCR output to entries with the information, given as a CSV file that has already been logged by Dr. Giroux, via edit distance. Using edit distance allows for some flexibility in the performance of our OCR component. Being within even a character or two would allow for successful matches in our solution. At this stage of headstone processing, the headstone images would have already undergone two thresholds that determine whether or not it is placed in the failure queue. Some more abstract cases that would have gone through the image processing and optical character recognition without issue include headstones with the same name (e.g. father and son) or an unnamed soldier, whose headstone would look identical to all the other unnamed soldiers. These cases are handled by placing them directly into the failure queue, allowing the user full control over how to handle these edge cases. Once the fuzzy search is complete for all images, we should have a complete SQLite database with Dr. Giroux's data matching its corresponding headstone image.

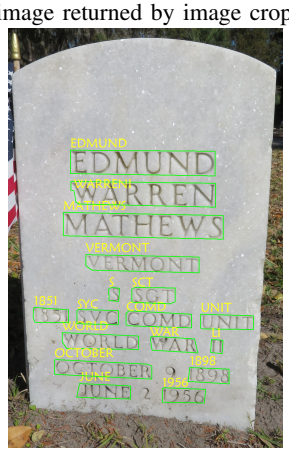
E. Failure Queue

If at any point in the process of our solution the output of each component does not at least meet the threshold we set, it ends up in the failure queue. As mentioned in previous sections, this can be the result of several variables:

- Low confidence for cropping and rotation



(a) Cropped image returned by image cropping module.



(b) Predicted bounding boxes and corresponding text for each box.

Fig. 2: Visual representation of the OCR task on a demo headstone from the Alexandria data set.

- Irregular headstone shape
- Low confidence on OCR output
- No match / multiple matches in fuzzy search

Here, the user is able to view any headstone images that are too irregular for our system to handle. The user also makes the final decision on what to do with the headstone.

III. User Interface

The user interface portion of this project is where all interaction between user inputs and the software components will take place. From the user interface, the user provides inputs such as the photos directory and the official cemetery records and the amount of padding for the images. After providing the inputs, the user interface will pass these parameters to begin the photo processing. It will make calls to the individual components of our project: the image cropping and alignment pipeline, the optical character recognition network, and finally through the fuzzy search algorithm. Since this process will take some time, it's expected that the user will have to wait for this to complete. Once this automated process has completed, the user has the ability to manually review the photos that failed the automated process.

Our group chose to implement our user interface using Github's Electron and Facebook's React JavaScript frameworks. The choice of these two seem to lend themselves to our project. Electron allows for customizability, as well as the ability to use web-tech (such as HTML or CSS) to build cross-platform desktop apps. While our intended user uses Windows, this flexibility allows for our group members to develop on different operating systems. Additionally, React allows for dynamic rendering of user interface elements that keep track of the current state and automatically re-render components as needed. Since our project consists of several different image datasets and the columns of the data gathered by our sponsor may differ from cemetery to cemetery, the ability to dynamically render components for the user based on the different inputs the user interface may receive is invaluable.

IV. Image Cropping and Alignment

The objective of the Image Cropping and Alignment portion of this project was to automatically crop the input images so that each contains only the most predominant headstone within the image, correctly oriented. Additionally, an optional padding parameter may be included to specify how much of the original image to include around the headstone. This solution was made to fit a set of headstones taken from Alexandria National Cemetery (ANC), St. Augustine National Cemetery (SANC), Florida National (FNC), Ft. Meade National Cemetery(FMNC), and Greenwood Cemetery (GWC).

A. Complications and Compromises

There are a variety of issues with designing a system for finding an accurate bounding box around a headstone. While a majority of the data set given was from military cemeteries, where almost all headstones are uniformly rectangular with a slight curve at the top, unique headstones still exist. Furthermore, even among regularly shaped headstones, if the picture was taken from too high or close an angle, the headstone could become significantly distorted and trapezoidal in shape, complicating rotation solutions. These issues are incredibly difficult to resolve, and therefore compromises must be made on some of the requirements within this section, with careful consideration given to the data set at hand.

See Table I for a breakdown of the data set with respect to the shapes of the headstones. Standard headstones are defined as any rectangular headstone. Embedded headstones are headstones that do not protrude from the ground. Irregular headstones are headstones that do not conform to a predominantly rectangle frame–crosses, obelisks, and headstones with large bases. Irregular headstones also encompasses invalid pictures within the data set, such as wide shots of the cemetery or images of specific parts of certain headstones. These images have no valid cropping, and are thus considered guaranteed failures.

As one can see by Table I, the vast majority (97%) of the headstones in the data set are rectangular in shape. While covering the edge cases of the headstones that are not rectangular would be desirable, they were deemed too scarce to devote time to. Embedded headstones, also a trivial subset of the data, were not considered important enough to warrant a unique solution. However, due to the nature of the embedded headstones, many are cropped correctly.

TABLE I: Types of Headstones

Cemetery	Total Number	Headstone Type		
		Standard	Irregular	Embedded
ANC	4,335	4,279	39	17
SANC	1,307	1,193	27	87
GWC	399	394	2	3
FMNC	193	178	15	0
FNC	135	135	0	0
Total	**6,369**	**6,179**	**83**	**107**
Total Pct.	**100%**	**97.0%**	**1.3%**	**1.7%**

Rotation was another complicated issue, and could be split into two parts:

- *Macro-rotation* - Rotation of the entire image by 90 degrees, oftentimes because the picture was taken in landscape mode instead of portrait mode and vice versa.
- *Micro-rotation* - Rotation of headstone itself by a typically small (less than 15 degrees) amount in order to correct the skew of the headstone.

Macro-rotation was a solvable problem, though appearing in only 17 standard images (Embedded headstones had much higher rates of macro-rotation, but are not being focused on in this solution due to scarcity).

Micro-rotation, on the other hand, was a much more complicated solution, due to the complexity of the problem. Solution attempts to solve micro-rotation ended up being imprecise, with an average error of 2.3 degrees, oftentimes more than the headstone was skewed to begin with, and were incredibly slow, taking an additional ten seconds per photo. Additionally, perspective distortion would cause some images to rotated incorrectly as the micro-rotation solution struggled to find the correct rotation of a trapezoidal shape, see Figure 3 for an example.

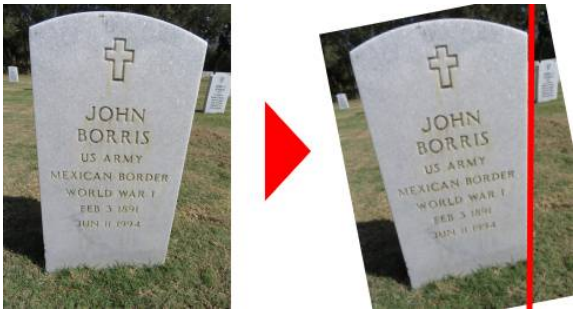


Fig. 3: The result of image rotation on distorted headstones

Due to these issues, it was agreed to fail overly rotated images to allow for the rotation to be handled manually.

Small rotations, under three degrees, would not be failed, and their bounding box must account for their rotation, to avoid impinging on the headstone.

B. Solutions

Early in the project, the approach to finding the bounding boxes of the headstones was to be done solely through image manipulation; this was intended to decrease the runtime of the program by avoiding expensive neural network predictions, and to avoid having to go through the difficulties of training the neural network on the image set. However, as the pipeline progressed, involving numerous operations such as sobel edge detection, contour manipulation, transforms, and color-space modification, the impracticality of the solution became apparent, although it still got decent results for where it was in development (around 70% of the images were accurately cropped).

Instead, focus was transferred over to constructing a convolutional neural network capable of instance segmentation in order to solve this problem. After thorough research, a flexible framework allowing for image instance segmentation was found, called Mask R-CNN. Although its original purpose was to create perfect masks around the object it detects, it also can find bounding boxes on objects.

The model was trained on 1,150 images of standard, correctly oriented headstones randomly selected from all cemeteries combined and was tested during parameter-tuning on an additional 350 headstones. The training set was down-scaled to 5% of the original image size in the interest of time, and the training was done using Google's Colab. Originally, upon making predictions with the model, the image would be down-scaled to be consistent with the training data. Interestingly, although the images that the model was trained on were very small, the model still performed better on full-sized images (failing slightly less images, but having bounding boxes overlap with the training annotations around 10% tighter) without a significant cost to speed–it ran only about 15% slower.

After training was completed, the model was tested to see how it would react on non-standard headstones, such as irregular headstones and embedded headstones. Since the model was trained only to find one specific class-that of a standard headstone-if any bounding box was made, the model became incredibly certain of its accuracy. However, with standard headstones, this confidence level was high, with over a third of the images having their bounding box found with more than 99.9% confidence. Eventually, a confidence tolerance of .995 was implemented, and any image not meeting this threshold would be sent to the failure queue to be handled by the user manually.

To solve the issue of macro-rotation, several techniques were attempted. The first idea was to train a simpler neural network to classify the orientation of the image, however, the results of this neural network were less than satisfactory, with only a 86% success ratio. Due to the scarcity of rotated images in the data set (only 17), this would incorrectly rotate correct images far more than it would fix incorrect ones.

Eventually, a simple heuristic was used: since the Mask R-CNN model was trained and tuned to find headstones, it would reject sideways headstones. Therefore, rejected headstones could be rotated each direction (90 degrees and 270 degrees counterclockwise–180 degrees was not necessary as no pictures were upside down and it seems unlikely for there ever to be), so the model would attempt to predict the bounding box again. If either of these succeeded, the image would be rotated accordingly. This method never attempted to rotate normal headstones. Additional attempts, such as horizon finding, were attempted but ultimately proved infeasible–more than 500 images of the data set have no horizon.

The entire system for cropping and rotation were created in Python, and were packaged into an executable script for use in this project with PyInstaller. Mask R-CNN extensively utilizes TensorFlow and Keras, and script also uses SciKit-Image for reading and writing images quickly. After predicting the bounding box and rotating the image, the script will crop the image to the result and write it to an output directory, specified by the user, and the script communicates its actions with the Headstone Photo Processing System at large. If the user specifies a padding value, the bounding box's borders will be increased by that value in pixels.

C. Results

As discussed previously, due to the overwhelming presence of rectangular shaped headstones, the model was tuned to perform best on those. When tested on a sample of 1000 headstones randomly sampled from the data set, the model fails to find a cropping for only 5. In all cases that succeeded, the bounding box the model finds has greater than 85% overlap with the hand-made annotation labels. Note that the images used for evaluation were not used during training or parameter tuning of the model.

The model predictably is much worse at finding accurate bounding boxes for embedded and irregular headstones. Cases where the model finds inaccurate bounding boxes (less than 85% overlap with annotations) are also unfortunately far more common with the these cases as well. Rarely, some headstones that have no valid cropping (invalid images) still had a high enough confidence to pass the model. See Table II for a breakdown of the cropping results.

The results for macro-rotation were worse than that of the cropping. When tested on 200 rotated standard headstones, the system only managed to right 186 of them, for a failure rate of 7%. The system never rotates upright images. Rotated images are scarce in the data set (only 17 out of 3,369), so this relatively high error rate should only account for a trivial amount of photos being incorrectly rotated–the system rights all but two images. However, in cases with irregular headstones and embedded headstones, the macro-rotation element is much worse; with embedded headstones in particular, it correctly orients them only 34% of the time. Due to the scarcity of embedded headstones, and the logistical difficulty in finding a correct rotation for them with no background features to work off of, this was to be expected.

TABLE II: Cropping Results

Headstone Category	Total Tested	Passed		Failed
		Valid	Invalid	
Standard	1,000	995	0	5
Embedded	107	70	0	37
Irregular	83	28	19	36
Total	**1,190**	**1,093**	**19**	**78**
Total Adj.*	**6,369**	**6,243**	**19**	**107**
Total Adj.* Pct.	**100%**	**98.0%**	**0.3%**	**1.7%**

*Adjusted values account for each category's frequency within the data set

V. Optical Character Recognition

As discussed previously, the primary goal of the OCR module is to extract textual information from images of headstones. Modern OCR techniques often employ a two-stage process for accomplishing this: first narrowing down the image to locations where text is likely to occur, then, from this set of regions, converting the characters to plain-text. These phases are commonly referred to as text detection and recognition. Each stage has yet to be perfected, and our problem brought with it unique challenges that made choosing an OCR solution very difficult.

A. Challenges and Problem Specification

Many solutions for general OCR exist online and in software packages, but our project had unique problems that severely limited the range of solutions we could use. One major issue of performing OCR on headstones is the orientation of the text: an issue our sponsor was already well-acquainted with and specified as a crucial case that our OCR module should be able to handle. Though a fair selection of headstones have strictly horizontal text, a considerable portion have key information, such as the name of the deceased, engraved in a curved fashion. If our OCR was unable to handle curved text, we would be missing out on perhaps the most important identifiable information needed for record matching: names.

Another major challenge was the surface on which the text existed. Most widely OCR systems require near-perfect conditions in the image in order to perform well, which is a luxury absent from some headstones. Since many headstones are old and thus not in pristine condition, the text on them is also imperfect. Especially in headstones affected by heavy erosion, we've noticed that both OCR and human eyes often have trouble discerning characters. Of course, we do not expect the module to handle cases even human eyes have difficulty recognizing, but it should be robust enough to light or moderate erosion, even if only partially recognizing the full text. Thus, our goal with the OCR module was to have a system that could consistently handle curved text, and have somewhat decent performance on eroded headstones.

B. Research

In doing research on modern OCR methods, multiple options were available. The first approach we considered was that of pre-existing, widely-available software packages such as Google's Tesseract Engine, and popular services like the Google Cloud Vision API. Both were easy to set up and use, though they each had their own pitfalls. Using the Google Cloud Vision API would conflict with our attempts to make our solution completely local. And while Tesseract could easily be integrated into a local solution, its performance on curved text did not reach the level we required for this project, in which headstones with curved text are extremely prevalent. This was also the case for the Cloud Vision API; both methods usually failed to pick up any text that was not completely horizontal, or returned nonsensical characters, which would clash with our string matching module. Additionally, both often even struggled to pick up regular, horizontal text *because* it was on a headstone, rather than surfaces where text are more commonly encountered in natural scenes, like paper or street and shop signs. These solutions were simply not robust enough to handle the diversity of surface quality, texture, and text orientation found on headstones.

For this reason, we decided to venture into more advanced topics, specifically from methods proposed in recent computer vision research. This was challenging, because very few methods were end-to-end. In end-to-end methods, the detection and recognition modules of the network are present and often trained in tandem, which ensures compatibility between the detector and recognizer, and often leads to better performance on the overall task. Many papers propose methods that tackle text recognition or detection separately, meaning we would have to first run images through the detector, then pass that output to the recognizer. However, for the most optimal performance, we believed it would be necessary to find models that could be trained together, or at least as part of a single work (as opposed to combining networks from multiple different works). This was done to ensure compatibility between the detector and recognizer, and has also been shown (in end-to-end networks) to have improved performance.

With this in mind, we initially settled on Mask TextSpotter [1], a very efficient neural network architecture for detecting and recognizing text of arbitrary shapes, submitted to CVPR 2019. Compared to most previous scene text recognition methods, which adhered to the convention of using bounding boxes, Mask TextSpotter was one of the few to follow representations like that of TextSnake [2], which fit bounding shapes more tightly to the text, allowing for more flexibility than a simple rectangle could provide. This practice leads to better performance in the text recognition stage, as there is little background in the proposed text regions. Mask TextSpotter, as its name suggests, drew influence from Mask R-CNN to generate masks by segmenting instance text regions. By treating the OCR problem as more of a semantic segmentation problem (in which individual pixels are classified as belonging to certain objects; in this case, text), the orientation of the

text becomes less of an issue. This idea is taken even further by being applied to the text *recognition* stage as well. The authors' method uses character-level semantic segmentation with a spatial attention module (to keep track of words as character sequences via proximity of the constituent characters) to identify characters and group them into words. Despite how effective Mask TextSpotter was, however, we ultimately could not use it in our solution for a significant reason.

During testing, while Mask TextSpotter performs well enough, we encountered issues when attempting to package it into the final application. Mask TextSpotter is implemented using the PyTorch framework, which is more research-oriented than alternatives like TensorFlow. In practice, all training and even most testing is expected to be done on GPUs, which is troublesome in our case because we cannot assume that the end user has a CUDA-compatible GPU to run predictions on. Because of this, PyTorch is ill-suited for CPU-only computation. The most prevalent method of doing CPU inference in PyTorch typically requires the use of the PyTorch CPU package, but using it brought dependency issues upon us. Attempting to modify the existing implementation using workaround methods to enable CPU-only inference also led to problems with other dependencies requiring the CUDA-enabled version. Since having a CPU-compatible version of the OCR was a high priority, we were forced to switch to another method.

AttentionOCR [3] is another very effective OCR neural network framework, also submitted in late 2019. Like Mask TextSpotter, AttentionOCR is flexible enough to handle text of various orientations, including curved text, and has the added benefit of being able to recognize text containing non-Latin characters. Though the latter ability was not required for this project, since all headstones contained only Latin characters, it is nice to know that the OCR module can be flexible for future use cases. AttentionOCR was trained on various iterations of the ICDAR data set, which part of a competition called the Robust Reading Challenge. The images encompass examples of scene text, or text that appears "in the wild": anything from street signs to logos to shop signs.

In the initial use of AttentionOCR, we attempted to train on the original data set (ICDAR 2017 - using multiple sections of the data set). It was trained for 1000 epochs, with 500 training iterations per epoch, all done on UCF's Newton GPU cluster. WHen testing this trained model, however, we found that the performance we got was slightly under the authors' competition model, for which the training code is closed source. The authors likely used training tricks or slight alterations to their network architecture to achieve this. Fortunately, however, they did provide demos of their pretrained models in Docker containers, which we opted to use instead. While still somewhat susceptible to eroded headstones, this model performed well enough on curved text to serve as our OCR backbone, often obtaining near-perfect performance on clear headstones.

One extreme advantage of AttentionOCR was that it was implemented in the TensorFlow framework, which is much more popular in the industry sphere and thus has much more support for CPU-only deployment. Using this TensorFlow implementation of AttentionOCR, combined with the Intel MKL-DNN library (which supports a data format used in the GPU version of this code that TensorFlow did not natively support), we were finally able to achieve a working CPU-only OCR module. Similar to the image cropping module, we then packaged the OCR module using PyInstaller to yield a simple executable file that required little to no extra setup from the end user to use. The bulk of our OCR module requires TensorFlow for running inference using the AttentionOCR trained detection and recognition models, and makes use of OpenCV (cv2) for image pre-processing and annotations.

Cemetery	Sample Size	Performance (%)
FNC	118	99.15
FMNC	100 (54%)	59%

TABLE III: OCR results on subsets of various cemetery data sets.

C. Results and Observations

See table III below for an overview of the OCR module on various cemeteries. Unless a percentage is present, the full data set (that passes through the crop phase) is used in the evaluation.

As the table shows, the performance of the OCR module varies depending on the data set. This is due to inherent differences in headstones between the cemeteries. For example, there are several null or somewhat incorrect predictions made by the OCR module on the Fort Meade National Cemetery (FMNC) because a significant portion of headstones in FMNC are old and heavily eroded; a similar situation is present in the Alexandria cemetery. Unlike these two, however, the Florida National Cemetery (FNC) has relatively newer headstones and thus sees much better performance. Both cemeteries have curved text, and the OCR performs well on them, though cemeteries with lower-quality headstone will have hampered performance. See figure 4 for an example on curved text from the Fort Meade Cemetery.

VI. Fuzzy Search

At the onset of the project, we thought that the OCR output would be sufficient to look within the database files. However, while our OCR solution performed well, occasionally there would be a few characters that would be incorrect, whether it was due to the lighting or composition of the headstone. Simply searching for exact string matches will give decent results, but a more robust search method was required. Thus, we chose to adopt fuzzy search, which uses approximate string matching via Levenshtein distance.

A. Solution

As was previously addressed, the use of fuzzy search with edit distance was a compromise to allow for flexibility in the OCR output. Our original method consisted of creating a string

Fig. 4: Performance of our OCR module on curved text with decent clarity

consisting of several fields from the CSV data that Dr. Giroux (e.g. first name, last name, etc.) and querying the OCR output in the SQLite database. However, we ran into an issue: we were getting false positives for entries that were close enough to the strings we queried. So we had to look to other solutions to improve our string matching performance.

On the basis that our OCR output performs well with only a few tokens having 1 or 2 incorrect characters we adopted a new search heuristic: search for exact matches, then fuzzy search. We iterate through the OCR output, querying the database for individual tokens to see if we can find an exact match. After we have iterated through all of the tokens and still have not found a single match, we run a similar procedure to our original fuzzy search method. Fuzzy search will be performed on the subset of the database that contains any exact matches, which we found to improve the performance of our string searching. If both the exact string matching and fuzzy search do not work, we put the image into the failure queue to allow the user to determine what to do with image.

VII. Conclusion

In the preliminary tests that we ran, the individual components of our solution performed well, achieving high accuracy on image cropping and rotation, optical character recognition, and string matching. We were also able to get the individual components working together to create an end-to-end image editing and optical character recognition system for our sponsor, Dr. Giroux to automate the work she does for the National Cemetery Administration.

Despite our success, there are many aspects of the project that could have been improved upon, as well as many challenges we faced. This project proved to be much more challenging than we first believed. Leveraging two computer vision models for different components of our solution introduced a lot of unknown variables due to the "black box" nature of neural networks. Our limited experience with current computer vision models and the lack of intuition gained from working extensively with neural networks was a big factor in the decision to use pretrained models with finetuning. A more specialized solution could perform better, but would cost more in terms of resources and time. Additionally, we had some difficulties working with fuzzy search, getting imperfect matches with our SQLite database and thus more images that go into the failure queue. A more robust solution for matching OCR output to the database is definitely an area that could see some improvement with further work.

Overall, we think our group did well given the time frame and scope of our project. It was a learning experience for all of the group members and allowed us to gain insight on what it is like working on an application that borrows from multiple disciplines of computer science.

References

[1] M. Liao, P. Lyu, M. He, C. Yao, W. Wu, and X. Bai, "Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes," *arXiv e-prints*, p. arXiv:1908.08207, Aug. 2019.

[2] S. Long, J. Ruan, W. Zhang, X. He, W. Wu, and C. Yao, "Textsnake: A flexible representation for detecting text of arbitrary shapes," *CoRR*, vol. abs/1807.01544, 2018. [Online]. Available: http://arxiv.org/abs/1807.01544

[3] J. Zhang, W. Wang, D. Huang, Q. Liu, and Y. Wang, "A Feasible Framework for Arbitrary-Shaped Scene Text Recognition," *arXiv e-prints*, p. arXiv:1912.04561, Dec. 2019.