Logo

Description automatically generated

Final Project – Using OCR to obtain discrete data from images

Craig Perkins and Kwame Nani

Northeastern University

June 2021 – EAI2020

**Final Project – Using OCR to obtain discrete data from images**

Object Character Recognition, or OCR for short, is the recognition of human alphabets, number systems and symbols from images into text on a computer. OCR can be done with handwritten language, pictures of scenes (for example an iPhone image of Times Square) or in many instances on scanned paperwork with the intent of obtaining structure from the form to automate tasks like data-entry. As you can imagine there is a great diversity of inputs to OCR systems and there are a number of strategies for handling different scenarios to improve the accuracy of the transcription1. Most cloud providers have a Vision API2,3,4 to provide OCR Cloud Services and Hewlett-Packard created a popular library called Tesseract for OCR that is now open-source and sponsored by Google5. OCR technologies have spawned a number of companies and services that provide OCR for well-defined problems and the number of use cases has expanded greatly in a short period of time.

OCR systems are being used in production at many large companies. Some noteworthy examples of OCR to point out are:

1. Google Lens - [Techcrunch](https://techcrunch.com/2019/05/07/google-lens-can-translate-foreign-languages-in-photos-and-read-the-text-back-to-you/#:~:text=to%20you%20%7C%20TechCrunch-,Google%20Lens%20can%20translate%20foreign%20language%20text%20in,read%20it%20back%20to%20you&text=Google%20is%20making%20some%20updates,it%20to%20your%20chosen%20language) – This is an example of OCR that I used recently when trying to buy a pancake mix from a Japanese import grocery store in Industry City in Brooklyn, New York. The directions on the packaging were in Japanese and I was able to use Google Lens to read the directions on the back and translate the text into English1.
2. Mobile Check Deposit – See Mitek Systems Mobile Deposit® (<https://www.miteksystems.com/mobile-deposit>) or ABBYY’s article on “Mobile Capture for Mobile Banking” (<https://www.abbyy.com/blog/ocr-sdk/mobile-capture-for-mobile-banking/>)
3. Debit and Credit Card OCR - <https://www.klippa.com/en/ocr/financial-documents/debit-and-credit-cards/>
4. Passport OCR – [Huawei Cloud](https://support.huaweicloud.com/intl/en-us/api-ocr/ocr_03_0106.html#:~:text=Passport%20OCR%20recognizes%20the%20text,structured%20information%20in%20JSON%20format.&text=For%20non%2DChinese%20passports%2C%20two,be%20extracted%20from%20the%20codes.)

# Project Introduction

For this project we were given scanned burial records and asked to devise a method for extracting structured data from the scanned images. The images we obtained are all images of 2 different forms with the majority with typed in text, but a few with handwritten data. All forms include the following data elements in common:

1. Decedent – The name of the deceased person
2. Burial Location Information – Section, Lot and Grave Number
3. Date of Burial
4. Age

Other information contained in some, but not all of the forms, includes the name of the Undertaker, Date of Exhumation (Date of Removal and Date of Reburial), Next of Kin, Veteran Status, Vault and Lot Owner and Further Remarks.

Figure 1.1 – Examples of the scanned burial records

|  |  |
| --- | --- |
| **Form B2** | **Form B3 (Without Lines)** |
| Table  Description automatically generated | Table  Description automatically generated with low confidence |

|  |
| --- |
| **Form B3 (With Lines)** |
| A picture containing table  Description automatically generated |

**Sorting the Images**

Before starting to devise a method to read data from the images we first wanted to get our bearings with the images we were working with. We noticed there were 3 distinct looking forms after combing through many of the images and that led us to trying out unsupervised clustering techniques to see if we could automatically sort the input collection of images into their respective forms. If we were able to create a technique to sort the data, then we could create a more robust pipeline when applying our solution to unseen data. When using Zonal OCR techniques (discussed more in the section on Improving the OCR) it’s important to be able to identify the type of form you are working with to make sure you are applying the zones correctly.

We used a clustering technique using a VGG-16 Neural Network architecture and K-means clustering motivated by this blog post: <https://franky07724-57962.medium.com/using-keras-pre-trained-models-for-feature-extraction-in-image-clustering-a142c6cdf5b1>. The idea with the clustering post is to use a pre-trained model built to identify the image classifications from ImageNet (See <https://keras.io/api/applications/>) and use the model’s predictions as input to a K-Means Clustering. We used a VGG16 model, the model-weights are already available in Keras and then did a K-means clustering with 3 clusters. The results were really good and it segmented the images into 3 distinct clusters based on form type.

**First Steps with OCR**

After sorting the input images, we took our first steps with OCR. For this step of the process, no OCR optimization techniques were employed. We just wanted to see out of the box how the different OCR engines performed with the data. We used 4 OCR engines to do the out-of-the-box text to string transcription on the input set of images. The 4 engines we used are:

1. Tesseract – We used a jupyter notebook and the py-tesseract library. You must have tesseract installed on your computer to use pytesseract.
2. Amazon Textract – AWS’ OCR offering
3. Google Cloud Vision API – GCP’s OCR offering
4. Azure Computer Vision – Microsoft’s OCR offering

See the attached zip file with the secrets to authenticate to the cloud services. These are paid services with a free tier so please use the service sparingly, if you run the cells in the notebooks provided once it will just run once over the input set of images. There are 4 csv files starting with `first\_attempt` that contain the text to string for all images in the input set. No optimizations were run yet and no effort was made to parse the output into the discreet fields. We employed Zonal OCR with pytesseract to get structured data from the input forms.

**Improving the OCR**

Since we are using OCR techniques, it’s important to keep track of the confidences of the transcriptions and an audit trail of what techniques were employed to improve the accuracy of the OCR transcription. Each technique used in OCR for this report returns the confidence scores of each word transcribed. We stored the confidences from pytesseract. Kwame and I are both relatively new to OCR and had to do some extensive searching and trying of various techniques to improve the accuracy of the OCR and find techniques on how to extract enough structure for us to at the very least put the fields common to all forms into a searchable database. Docparser has a wonderful blog post on techniques to improve accuracy here - <https://docparser.com/blog/improve-ocr-accuracy/>. Tesseract has a post on the subject as well here - <https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html>.

Our key observation was that our input is already normalized to a great extent and so we can employ a technique in OCR called Zonal OCR (<https://docparser.com/blog/zonal-ocr/>). With Zonal OCR we can segment the input image into boxes corresponding to where we know the field values reside and only perform OCR on that box. In doing this, we do not require any post processing of the output of the text\_to\_string to figure out which values correspond to which field labels since we already know where the field value resides on the document. See figure 1.2 that shows the creation of a Tesseract UZN file using kull (<https://jsoma.github.io/kull/#/>).

|  |
| --- |
| **Figure 1.2 - UZN File Creation Selecting Zones** |
| Table  Description automatically generated |

In order to ensure we are moving in the right direction it’s important to track an *accuracy* metric that we can use to assess the performance with each iteration. Since getting a proper accuracy score would require manually transcribing all of the images and comparing the OCR to the source of truth it’s not possible to get a good algorithmic accuracy score we can work to improve. In OCR you typically focus on character level accuracy and word level accuracy. After each run, we took a look at the results of a few documents in each cluster vs how we would’ve performed the transcription manually to tell if we were moving in the right direction. The Zonal OCR and looking at the individual word confidence scores gave us confidence that we were making progress.

Once we employed the Zonal OCR on each of the labels we decided to store the results both in a CSV file for each Cluster with all of it’s a fields and also store the common fields in aa SQLite database to make the values searchable which is one of the ultimate goals of this project.

**Analysis of Results**

One of the first things we noticed when storing the confidences and structured transcriptions was that we found a number of transcriptions indicating that there was a Date of Reburial. The Date of Reburial Date is the ‘To’ column in Form B3 indicating that a body was reburied after being exhumed. It was surprised because we expected that to come up very little so we checked to see what the values were being transcribed as if there was a high confidence from pytesseract. What that revealed to us is that there was an alignment issue on this form and the same form had the To label places in different spots horizontally, see the figure below. Notice the position of the label **To** which is to the left of **Ter** in the form on the left and to the right of **Ter** in the form on the right. This is an obvious limitation of Zonal OCR which may be improved with more sophisticated clustering techniques.

|  |  |
| --- | --- |
| **Form B3 (Left Alignment)** | **Form B3 (Right Alignment)** |
| Table  Description automatically generated | Text, letter  Description automatically generated |

Other issues the initial analysis revealed was that Tesseract was trying to recognize non-ascii characters. While there may certainly be non-ascii characters in the data we wanted to see if it would improve the recognition more if we set a character whitelist to pytesseract and we did notice modest improvements.

We tried to determine if we should remove words with a confidence level below a given threshold, but ended up keeping all of the transcribed words in. If we had time for more analysis and development we were thinking of including a threshold configuration on the ingestion pipeline into SQLite to only include tokens above a user defined threshold.

**Cleaning the data for SQLite**

One of the questions we sought to answer from the data given is: what was the average age of the deceased buried in the cemetery? In order to answer that question we wanted to ensure that we stored the Age as a number in the database so that we could perform SQL aggregation functions when asking questions on the dataset. Instead of loading the CSVs as is into the SQLite database we wrote a python script to process the csv and insert data into sqllite so that we could attempt to get better structured dates and other data points.

**Other methods tried**

Since Kwame and I are relatively new to OCR we tried many different techniques for improving the recognition. You can see the attached Python notebook called pytesseract-scratch.ipynb that shows some of the different strategies. Without a Zonal approach we would have needed to match the field labels with the field values in order to obtain structure from the OCR Result. Some of our lines of thinking in this regard was to see if we could identify different font weights (i.e. bold vs normal) to distinguish between the field labels and field values. Pytesseract does support getting the word font attributes, but we were not able to get this working well. It was this avenue of trying to distinguish the labels from the values that led us to discovering Zonal OCR. Other techniques we tried was specifically looking for things like date tokens or phone tokens based on regexes or using python’s dateutil.parser to see if the forms were filled out correctly.

**Conclusion**

Artificial Intelligence and Deep Learning will have a profound impact on our societies and it’s important to understand where we are in the Digital Transformation of the Global Economy. Deep Learning has permeated our lives in ways we may not be even be aware of and continues to play a larger role in the modern home and modern economy. Our current methods for Artificial Intelligence are powerful but are also quite narrow in scope and some prominent researchers have indicated that we need new techniques to solve different sets of problems6. In 2005, Ray Kurzweil published the book “The Singularity is Near” in which he predicts at some point in the future that machines will surpass human intelligence. I do not wish to speculate in this regard, but we are witnessing a transformation in the global economy in which computers are assisting with decision making and in other aspects are the decision makers themselves.

Futurists have shown cars without steering wheels and envisioned other big changes to the way we design products and cities with the emergence of these new technologies like autonomous vehicles. In preceding sections I talked about the impact of AI in my own domain of Pathology. It’s not hard to imagine a Pathologist’s office of the future without a microscope, but I think that is still quite some time off. That being said I think we are going to make leaps this decade towards that vision. I will commit to using deep learning techniques in my own career and at the moment I am interested in pursuing a higher degree in Computational Pathology and use my skillset to advance Cancer Care using machine learning.

References

1. Improve OCR Accuracy With Advanced Image Preprocessing - <https://docparser.com/blog/improve-ocr-accuracy/>
2. Amazon Textract - <https://aws.amazon.com/textract/>
3. Google Cloud Vision API - <https://cloud.google.com/vision/docs/ocr>
4. Azure Computer Vision - <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>
5. Tesseract - <https://github.com/tesseract-ocr/tesseract>
6. Google Lens can translate foreign language text in photos and read it back to you - [Techcrunch](https://techcrunch.com/2019/05/07/google-lens-can-translate-foreign-languages-in-photos-and-read-the-text-back-to-you/#:~:text=to%20you%20%7C%20TechCrunch-,Google%20Lens%20can%20translate%20foreign%20language%20text%20in,read%20it%20back%20to%20you&text=Google%20is%20making%20some%20updates,it%20to%20your%20chosen%20language)
7. Mitek Systems – Mobile Deposit - <https://www.miteksystems.com/mobile-deposit>
8. ABBYY – Mobile Capture for Banking - <https://www.abbyy.com/blog/ocr-sdk/mobile-capture-for-mobile-banking/>
9. Huawei – Passport OCR - [Huawei Cloud](https://support.huaweicloud.com/intl/en-us/api-ocr/ocr_03_0106.html#:~:text=Passport%20OCR%20recognizes%20the%20text,structured%20information%20in%20JSON%20format.&text=For%20non%2DChinese%20passports%2C%20two,be%20extracted%20from%20the%20codes.)