FullContact Advanced OCR

Matthew Baldin
Allee Zarrini
Samuel Reinehr

June 20, 2017

Introduction

FullContact is a contact management platform designed for both professional and enterprise use. FullContact has a suite of applications designed towards reaching their goal of solving the world's contact information problem. Funded by venture capitalists, FullContact has experienced huge growth since its inception. Offices are now held in four locations worldwide, with two locations in the United States, one in India, and one in Latvia. There are more than 250 employees. Of particular note about FullContact is its policy of "paid paid vacation." Paid paid vacation allows FullContact employees to be paid to take paid leave to visit an interesting vacation locale. Interestingly, paid paid vacation's media coverage was estimated by the FullContact team as being worth over eleven million dollars in marketing value. In a sense, the policy has paid for itself several times over. Among their suite of applications and services, FullContact offers the service of transcribing photographs of business cards into contact information. FullContact assigned us to develop an application that takes as input an image of a business card, and outputting formatted data of the card's contents. The vision of the application is to improve upon the current methods FullContact uses to provide this service, including outsourcing the work to humans who look at the image and transcribe the contents by hand. Additionally, the application is required to satisfy a list of requirements.

Application Requirements

Our project's title is Advanced OCR. OCR stands for optical character recognition. Optical character recognition is a computer vision problem involving the extraction of characters from an image. Our project is mainly focused on optimizing the OCR process, as well as interpreting the output that this process returns. For this reason, the team chose to use Google's open source OCR library Tesseract.

Among the specified requirements are the functional requirements of the product. The application shall take an image as input. The output shall consist of structured data relevant to contact information extracted from the image. If no information can be extracted, the solution shall notify the user of this. In detail, the application should accomplish the following: identify the space in the image that holds extractable information, splice the image into parsable pieces, extract text or images from these pieces, and classify the type of information extracted from the pieces. As an example of the entire process, consider an input image of a business card on a table. The solution should split the business card into different sections, parse each section, and classify the information gained from each section, such as phone numbers, addresses, or company names.

Additional requirements are imposed on the development of the product. The application must be developed using the Scrum development pattern. The contractors are to work at the client's office every day from 9 AM to 5 PM. On days when they do meet with the advisor, the contractors are to work remotely from the Colorado School of Mines. Every morning at 9 AM, the team is to meet with their advisor to discuss the project in accordance with the Scrum development process. Gradle will be used to resolve dependencies.

The project is considered done when the application can successfully parse useful information from an input image. The application should output structured information that the FullContact team can integrate into their larger platform. This can be tested with actual

business cards that have verifiable information. The team agreed that a deadline of 6 weeks was very manageable for the scope of this project.

System Architecture

We have designed our system using a pipeline model. At different stages in the pipeline, a specified type of input is received. Each stage processes the input and returns output that meets the requirement for the input specifications of the next stage. An image of a business card is received as the initial input of the system, and it is required that we output structured data concerning the contents of the business card. The client has provided little specification towards the exact implementation of the project, so a pipeline model was chosen as it seems the most appropriate for this type of input to output project, with the additional benefit that we may work on individual portions of the pipeline before requiring the completion of other portions of the entire project.

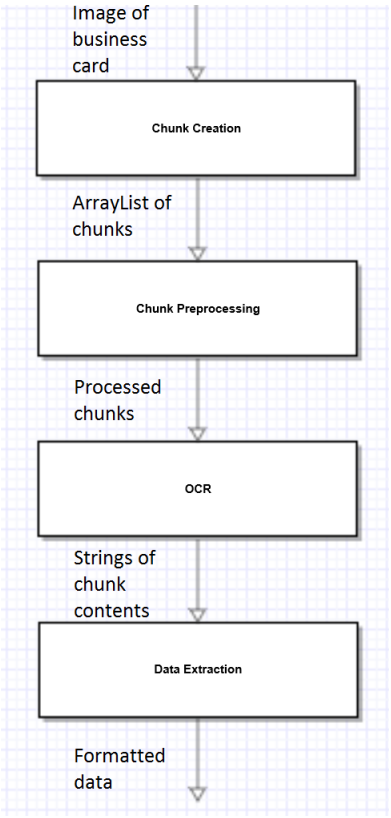

Figure 1: Architecture diagram of the pipeline process.

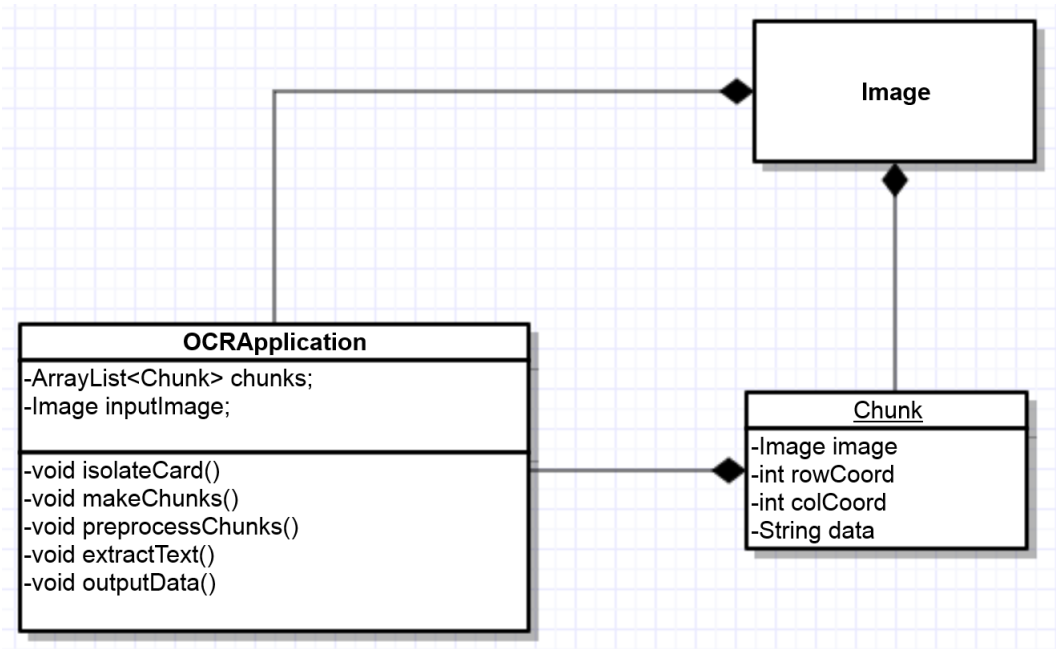

Figure 2: UML Diagram illustrating the framework code structure of the application.

The beginning of the pipeline receives as input a picture of a business card. The first stage processes this image, and outputs a set of images, denoted as chunks. A chunk is an isolated portion of the image believed to hold important information. Each chunk carries with it the location of where they were cropped from the original image. Additionally, the original image of the business card is passed down every stage in the pipeline. The second stage of the pipeline preprocesses each chunk to prepare them for the OCR process. This includes sharpening, contrast adjustments, and converting the image to grayscale. The third stage takes these processed chunks and performs OCR on them. This process returns strings of the characters contained on the chunks. The fourth stage of the pipeline runs various heuristics on the chunks. The purpose of this final stage is to determine the type of information contained on the chunk. For example, for a chunk containing various phone numbers, this stage's desired output is a breakdown of the different phone numbers output into information capable of being integrated into the client's overarching system.

Technical Design

One of the most interesting aspects of our final design is the chunking algorithm used to split a business card into pieces which contain isolated information. We first became interested in the idea of chunking business cards due to the observation that information on a business card appears split up into different segments containing isolated information, like phone numbers or email. An algorithm that splits an image into these isolated chunks could therefore assist us in extracting contact information, as we are granted a powerful heuristic in determining what type of information is contained on each chunk.

Additionally, tests with Tesseract indicated that Tesseract performs better on an image that has been split into different components. Therefore, the team decided that it would be beneficial to design our program around the creation and processing of chunks taken from

the original business card image. However, the implementation of an algorithm designed to do this proved to be more difficult than expected. Regardless, we have successfully developed an algorithm that chunks images in a satisfactory manner.

The algorithm works by determing what color makes up the background of the image. This is referred to as the image's whitespace. After this is determined, all of the points in the image that consist of whitespace within some tolerance are marked. Then, the algorithm identifies locations that contain a sufficient amount of non-whitespace. Based on a tolerance determined by the image's dimensions, the process cuts out the locations which contain a sufficient amount of non-whitespace using OpenCV's image cutting functionality. Some preprocessing of the image can often improve the results of the image. This preprocessing involves blurring the image and increasing the contrast. The algorithm will occasionally generate chunks that contain no useful information. This does not cause problems within the process, and these chunks are ignored.

Figure 3 displays a card that could be used as an input for the algorithm. Figure 4 displays the useful chunks generated from this image.

Figure 3: The input image.

Figure 4: The useful chunks created by the process. Each chunk is a separate image inside the application. "Garbage" chunks containing useless information are ignored by the process.

Another important aspect of our final design is the development of text heuristics. The OCR pipeline stage associates with each image a string of text. As part of our design process, we had to develop several heuristics for the detection of contact information from these strings of text. One heuristic we developed of particular interest is name detection. Name detection is a far more difficult problem than email, address, or phone number, as the assumptions about name structure are not as strong as the other forms of contact information.

To detect names, one approach is to assume that the other information is found. As almost all business cards will contain names, one approach is to assume names are the information that is otherwise not detected. However, if this is the only approach taken, it is extremely prone to errors. Instead of this approach, our team assumes the email address is related to the name. In general, this assumption is very applicable and it only fails a few odd cases. Assuming the email address has some information related to the name, we developed an algorithm to match substrings to the text produced by the OCR process. We begin by partitioning the outputted text into words by splitting across the space and newline delimiters. Then, we attempt to match all possible substrings of the first half of the email address to blocks within the created string array. If successful, we pass this as a name hint, appending the adjacent blocks of strings if there is a match in them as well. With this approach, we are able to accurately handle many cases of name-related emails and we improved our name detection dramatically. Figure 5 illustrates how a substring match from the email allows us to extract the full name of the contact.

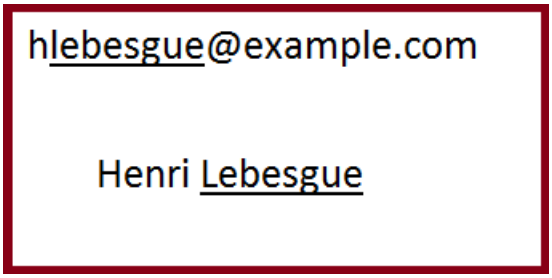

Figure 5: A match of "Lebesgue" in the email allows the full name of "Henri Lebesgue" to be detected.

Design Decisions

Programming language is a very important decision to make during the design of any program. Our group has decided to work in Java for several reasons. The first reason is to be in accoradance with the standard langauge used by our client. For the best ease of integration into FullContact's platform, as well as readability, Java is the ideal choice. Additionally, as a team we have a high level of familiarity with Java due to coursework and additional projects. All of the needed external libraries are available in Java or have Java bindings.

We have decided to make extensive usage of external libraries to facilitate the implemenatation of our project. The external libraries being used are outlined here, along with our rationale for using them. We are using OpenCV to process images. While many image processing algorithms could be reimplemented by our team, OpenCV is an extensively tested and optimized implementation that we are happy to take advantage of. For this reason it is better to use OpenCV rather than reproduce its functionality. The second external library we are using is Tesseract. Tesseract is an OCR library. Implementing our own OCR library would take well over 6 weeks of work. With Tesseract, we can abstract this part of the process away with a library call. This allows us to focus our project into two main halves: preparing an image for Tesseract and interpreting Tesseract's results. Also, Tesseract can output a confidence value for the characters it has detected. This allows us to have a fairly reliable metric to determine whether or not it will be possible for our application to accurately extract contact information. A library developed at FullContact is used for extracting contact data. This library was originally designed to grab contact data from email signatures, however, we have successfully adapted it to the retrieval of contact information from business cards. This adapation comes from the application of various heuristics developed by the team. The heuristics develop what the libary denotes as hints, which are passed to a contact information extractor object to retrieve the contact information.

We have decided to make use of Gradle to handle dependencies and library installation. Our decision to do this was urged by the client, however it has helped us learn about modern dependency management. Additionally, Gradle allows the user to stay up to date on the libaries being used to ensure full combatability and ease of use. This helps any developer using our software avoid the hassle of manually installing several libraries with various version releases.

We decided to design our implementation as a pipeline so that, should the client decide to incorporate our work into their system, they may simply feed the process an image of a business card and receive structured data as an output. This had the additional benefit of allowing us to work on different segments of the program due to a lack of dependencies between pipeline stages.

Results

We successfully implemented the client's wishes and fulfilled the requirements outlined in our definition of done. The process takes an image of a business card, and outputs structured data in the form of a Contact object type in accordance with the rest of FullContact's software suite. However, we did not have time to implement any of the extensions to the program's functionality that we discussed at the beginning of the project.

The application has been qualitatively tested on one hundred business cards. A sample of 30 business cards was tested manually for correctly extracted information. Of these 30 business cards, we detected $\frac{21}{30}$ names, $\frac{13}{15}$ addresses, $\frac{26}{30}$ emails, and $\frac{30}{30}$ phone numbers. In addition, the development process closely followed the developmental principle of test-driven development, meaning each feature of the program has been tested extensively, with periodic assurance that new features have not affected previous functionality. While not every business card results in a success, our application produces a confidence value associated with the output that allows a decision to be made whether or not to utilize the application's output for any specific business card. In the common case, the application is observed to perform well. The team worked mostly on pictures of business cards online. However, all pictures that the team took themselves of a business card extracted all information present on the card.

At the beginning of the project we discussed many possible features that could be included to extend the application's functionality. Some of these features are language detection, searching for online social media accounts of contacts, and an improvement of the OCR process through the use of machine learning. Specifically, the team was especially interested in the usage of Google's TensorFlow machine learning library towards tackling the problem of OCR. Additionally, there is still work to be done to in incorporating the application into the larger framework of FullContact's infrastructure, however; this process will be facilitated by the pipeline design of the program.

The most valuable lesson we learned from the development of this application is that minor issues should be quickly addressed, before they snowball into larger issues. We encountered this in the context of utilizing Gradle to manage dependencies for our application. As we had trouble implementing this correctly, our client told us to worry about it later. However, ignoring the problem resulted in much hassle that could have been avoided, and in hindsight we should have pushed harder to eliminate such a minor issue immediately at the start, before it became a time consuming issue that impeded our progress.

We also learned about the importance of a focused work environment. As we worked on site, it became clear that coming into work every day surrounded by coworkers who are also focused on similar work creates an environment of productivity. In the same way, we learned how to work in such an environment and be helpful to our fellow coworkers. Consistently interacting with the client has improved the entire team's communication skills.

Appendix

To use the application, simply follow the instructions included in the README file from the github repository. The program was created in Java, so it should be able to be ran by all common operating systems. The code follows the Google Java coding style guide. We developed the application in the IDE IntelliJ. Other Java IDE's with support for Gradle projects should also be able to open the project.