Comp 490 – Final Report
Split-EZ

Pascal Archambault – 40060852

April 2020

1 Abstract

Needing to split everyday expenses with roommates inspired Split-EZ's development throughout this winter semester. This project is composed of two parts: a mobile application interfacing with the user and a receipt analysis API that uses various computer vision techniques to read and extract data off of the picture of a receipt. The project was developed to termination, and an empirical analysis of the API's performance was conducted. Results show that the algorithm has a precision of 34.38% and a recall of 45.83%. These results could be improved and many solutions, such as using a neural network based approach, are proposed to improve the overall project in the short, medium, and long term.

2 Introduction

As fellow students can attest, splitting cost of living expenses between many people can be quite troublesome. Thus, alongside my personal interests in machine learning, computer vision, and mobile applications, I decided to work on a proper solution. Split-EZ is a mobile application that helps organize bills and lets you split them with an indefinite amount of people by simply taking a picture.

3 Background

This project is my first foray into the fields of computer vision and machine learning, as well as modern Android development. As such, a lot of research was required, and some functionalities did not work as well as intended. However, this project allowed me to gain a lot of valuable experience in those fields, as well as guide me towards future endeavours.

4 Problem Statement

Splitting bills between people seems to be a problem that people have all the time, yet there are no readily available solutions for it. There exist certain applications extracting features out of a receipt, but they either do not apply to that exact problem, or they are not free. As such, **there is a need for a free mobile application that splits one or many receipts between different people simply and easily**.

5 Project Details

5.1 Features

5.1.1 Overview

To avoid any confusion, let's define the user as "someone interacting with the application", as there is no account feature. Following this definition, the typical application flow for a user goes as follows:

1. Create a group
2. Add members to the group
3. Take a picture
4. Validate receipt items
5. Save receipt

Other features accessible to the user include: modify/delete any of the created entities, e.g., group members, and visualize the total amount owed by other members of the group, etc.

5.1.2 Detailed Features

For the sake of brevity, I will only detail a few key features of the application as they offer enough intuition about the application's behavior.

Adding a group: A user can add a group by simply giving it a name. There is no limit as to how many groups a user can create.

Adding a member to group: When a group is created, members need to be added for the bill to be split amongst more than just the user. As such, a user can add a member by specifying the member's name. There is no limit as to how many members a group can have.

Taking a picture: A user can take a picture of a receipt, which will then be analysed by the API. The API will then send a response containing every receipt item and their detected price back to the mobile application. Once the

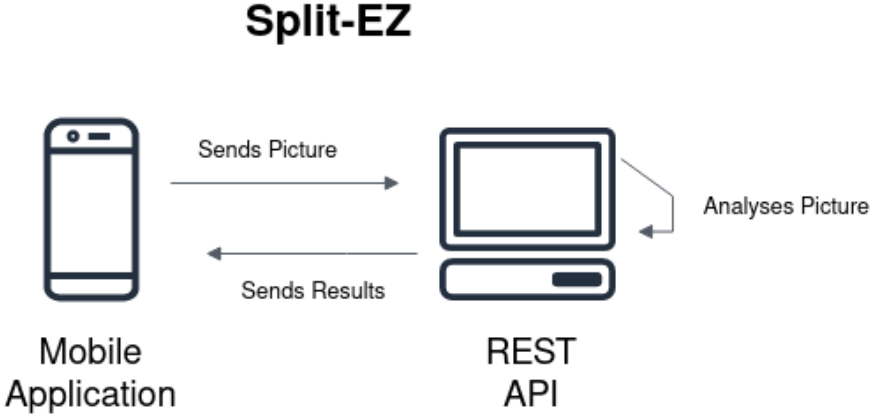
mobile application receives the response, it will display a list of items found and their price to the user, which will need to correct and validate the presented information.

Saving a receipt: Once the user validates every entry returned by the API, they can save all the receipt items under a receipt, which will be split amongst people.

5.2 Architecture

Two major components compose this app: a mobile application, allowing interaction with a user, and a single REST API that reads and analyses a picture of a receipt to return the relevant information to the mobile app.

Figure 1: Application Architecture



5.3 Design

Split-EZ's goal is to be simple and quick to use. As such, the flow of the application is as streamlined as possible. This is reflected both by the limited amount of features available on the app, as well as the overall application flow.

The choice of separating the application into two parts, namely the mobile application and the REST API, also reflects the goal of simplicity. Because the burden of processing and analysing the image is relegated to the API, more phones can use the application. Since most phones have subpar hardware compared to a computer in the same price range, the algorithm's execution is both faster and cheaper.

Furthermore, due to time constraints and other factors such as health issues,

certain planned features either had to be cut or reduced in scope. Such features include user accounts and payment through the mobile application.

5.4 Implementation

5.4.1 Mobile Application

Split-EZ's mobile application is built using Google's Flutter framework. Flutter allows the creation of Dart components, which can then be put together to create new components. Flutter is a recent framework released in May 2017 with a lot of documentation and a strong community, which makes it perfect for the development of a new Android app.

Flutter's novelty is not the only reason it was chosen to build the mobile application. Its "component-based" approach makes designing and building applications simple and straightforward, allowing for more development time on the API. Flutter also facilitates access to the hardware of the phone, allowing us to easily access the camera.

5.4.2 Reading the Receipt

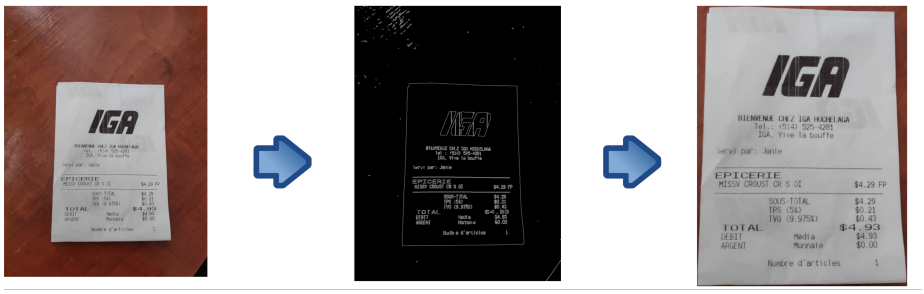
Reading and analysing the receipt is all done through a Python REST API using Flask, OpenCV, and PyTesseract. Flask is a uniquely Python library that provides us with many tools related to building a REST API, like a development server, API endpoint management, request parsing, etc. Flask was chosen because it makes implementing an API much faster and easier than having to manage a web server and having to manually parse incoming requests. It is also the REST API library I am most familiar with. OpenCV is an established open-source computer vision library that allows us to manipulate and process the image. Such manipulations include cropping, changing the color mode, applying different filters, etc. PyTesseract is an optical character recognition library wrapping around the Google Tesseract OCR engine. It allows us to detect text within a given image.

Python was chosen because it is the language I am most familiar with, which lets me focus on the logic of the algorithm rather than waste time on debugging or language-specific implementation. Python also has certain advantages over other languages. There is a strong Python community, which makes finding information easy. It is also a language with an increasing amount of usage, both in academia and industry, due to a plethora of computational and scientific libraries. Thus becoming more proficient in Python is a clear advantage.

Once a request to analyse an image of a receipt has been received by the REST API, feature extraction is done. Extracting the important data, e.g., item, price, from the receipt is done in two parts. First, we use OpenCV to preprocess the image, cleaning, and cropping it to a standard format. Cropping is done in two

parts: Canny edge detection, which identifies lines in the picture, and a custom algorithm, which iterates over every detected line, finding 4 lines making a box and cropping the picture down to our detected box. From the cropped and cleaned receipt image, we apply optical character recognition using PyTesseract to extract the relevant data. The Tesseract OCR engine groups letter together into words, forming clusters as it recognizes close by letters. The output of the Tesseract OCR engine corresponds to a list of such clusters. The output of Tesseract is then parsed using a set of rules and regex to extract the price out of each cluster, since one cluster corresponds to one receipt item. Once the clusters have been parsed, a response to the mobile application sending JSON data containing each receipt item's data.

Figure 2: Edge Detection And Cropping Process



6 Study

6.1 Methodology

The purpose of this study is to assess the performance, in terms of precision and recall, of the receipt analysis algorithm. We are not interested in the execution speed of the algorithm, because receipts have many intrinsic factors that may impact the time it takes to identify every receipt item, such as its condition, the character print, non-alphabetical characters, etc.

I randomly selected a sample of 18 receipts from 4 different businesses out of a population of hundreds of receipts amassed from my day to day life. As such, our sample contains receipts of varying sizes, lengths, and conditions. Every receipt picture has been taken at the same time of day, against the same black background. We define a "correct" identification as identifying either the price of an item or part of its name, while an "incorrect" identification corresponds to noise returned by the algorithm. Figure 3 shows an example of receipt feature identification.

Figure 3: Receipt Analysis Example

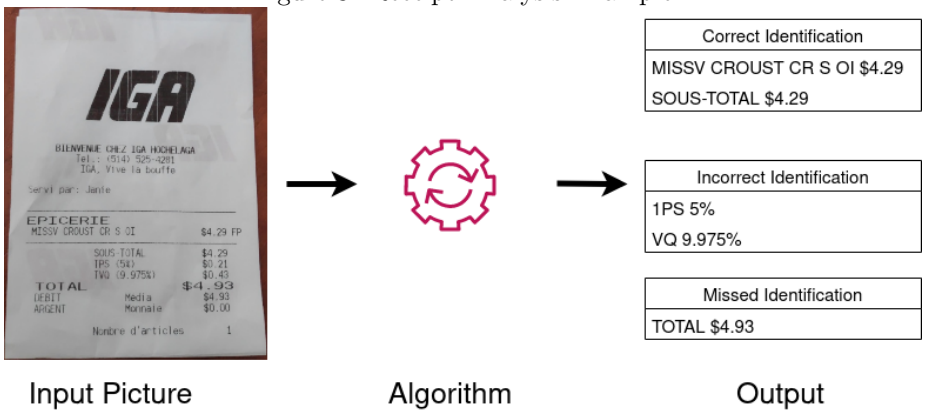


Table 1: Receipt Analysis Results

Business	Correct Identification	Missed Identification	Incorrect Identification	Sample Size
IGA	2	4	7	4
Marché Newon	6	11	15	8
Maxi	12	9	16	3
Tim Hortons	2	2	4	3
Total	22	26	42	18

6.2 Results

From the data in table 1, we conclude that the receipt analysis algorithm has a precision of 34.38% and a recall of 45.83%. In other words, 34.38% of our identifications were correct, while we could only identify 45.38% valid items out of receipts. These metrics show that a computer vision approach can extract and recognize certain receipt features out of an image. However, due to the unstructured nature of image data, there is a lot of noise introduced when taking a picture of a receipt. Preprocessing the image allows for the reduction of certain sources of noise like angle, distance, and frame. Receipts themselves introduce a lot of variance in our results when trying to extract certain features. The receipt's overall condition, as well as the ink quality, the typeface, and the language, are examples of such intrinsic features.

Therefore, we cannot conclude whether this algorithm fails to identify relevant data with higher precision because of its inherent logic or intrinsic limitations.

7 Discussion

This project came from a place of passion, where I set out to complete a project based on an idea that would add value to my daily life. However, in retrospect, I may have bitten more than I could chew. While I knew the REST API part of the project would not be perfect, I still wanted to make the most out of it. However, work on the mobile application took more time than expected, thus slowed the development of the project, leaving less time to work on the more complex features.

I also encountered a stall in the development of the API, as a lot of work resulted in no tangible benefits since a lot of time was spent learning and toying with machine learning and neural networks. In fact, nothing was conclusive, as it would either require thousands of images of receipts in good condition that I do not possess. There are certain ways to acquire this much data, but it mostly relies on paid services generating receipts based on a template. This approach might work if I were to always analyse receipts from the same store, but would fail when we have more than one type of receipt. I believe that training a neural network to extract features from many receipts is impossible with my means, as I would require access to thousands of pictures of receipts from different businesses, akin to the type of data accounting departments at large companies have access to.

The ambitious scope of the project led me to delve into multiple fields and gain a better understanding of how to approach certain problems. Thus, even though the project's overall practicality may not be what it could have been, the learning aspect of the project was a resounding success.

8 Future Improvements

8.1 Short Term

The only part which could be improved in the short term would be the cropping process. As the custom algorithm's current implementation is pretty naive, it could benefit from some refactoring in order to be executed in parallel, making its run time much faster. However, no small changes could be made to improve the precision of the algorithm.

8.2 Medium Term

With a few additional months, it may be possible to increase the precision of the algorithm by implementing a neural network able to extract the relevant portions of text and price data that interest us, instead of having to do OCR over the whole image. This would decrease both the overall amount of false negatives and false positives.

The concept of user accounts could also be introduced to the application. This would let users manage all of their receipt operations in the cloud, as well as better, manage their groups and group members.

8.3 Long Term

The most obvious long term improvement that could be made would be to develop a custom neural network able to extract a label and a price out of the receipt, no matter their relative position.

The mobile application could also receive a few additional core features. Users could pay their debt to other users directly through the application, or they could see an outline of what the algorithm sees in real-time through the camera before snapping a picture, etc.

9 Conclusion

Even though the project does not function at a production level, and ended being much smaller in scope than first designed, it still managed to see the end of development into something concrete. Working on Split-EZ has rewarded me with plenty of experience in mobile development and computer vision, but has also helped me step foot in the bigger fields of machine learning and neural networks.