OCR Extraction for Pre-Payment Processing

Final Project Report

Deutsche Bank Global Technology

CSC 492 Team #3:

Dustin Hollar

Taha Karimi

Shreyas Kuloor

Tianming Liu

North Carolina State University

Department of Computer Science

December 13, 2019

Table of Contents

1.0.0 Executive Summary[1]

The problem proposed to this team was to create software that utilizes Optical Character Recognition (OCR) and Machine Learning (ML) to locate and extract the fields and text on a bank check uploaded by a user. The software should then validate those fields for accuracy, legitimacy, and formatting, and output the results of the extraction and validation processes to the front end interface, with the option to download a JSON output file or edit and re-validate the extracted text.

Deutsche Bank Global Technology (DBGT) is the sponsor for this project, and they are looking for the development of software through the use of open source software, as opposed to 3rd party and/or commercial products which are costly and do not provide much flexibility. In order to solve this problem, our team decided to use AGILE development methodology practices with an iterative process to ensure continuous progress and allow for review and assessment at weekly sponsor meetings. We were not provided with a pre-existing system and are building the software from a blank slate.

A number of challenges related to scope and scalability have been addressed by the team. For scope, both languages and the country of the check have come up as issues, and we decided to restrict check formats to use English as a language and US formats for checks. For scalability, the task at hand is to determine how best to allow for future expansion of the software, in terms of configurability (ie. supporting different languages on checks) and efficiency (ie. adding multi-threading support for better performance).

After analyzing the requirements, the functionality was organized into a pipeline architecture that contains 5 stages. The first stage is uploading the check image, the second is preprocessing the image, the third stage is extracting data fields from it, the fourth stage is extracting data from the fields and verifying that data, and the last stage is post-processing the check image and drawing the bounding boxes on the final image displayed on the frontend.

Resources needed to implement the project are minimal, with open source libraries for OCR such as PyTesseract and another one for handwriting recognition using Tensorflow. For testing, the program uses Python unit testing framework called PyUnit. OpenCV is used for real time computer vision techniques for preprocessing passes over the image of the check and routing/account number extraction.

The design then outlines the pipeline and the stages mentioned above, and includes a wireframe depicting the user interface and interaction. Implementation of all requirements has been completed, and an input check will be passed through all stages of the pipeline successfully.

The test plan contains front end black box testing that will be performed manually on our complete system, and at this stage all black box testing has been completed. Lastly, unit testing is discussed, outlining the expected input and output of each field and a discussion of test results and coverage.

[1] Primary author: Taha Karimi. Contributing Editors: Tianming Liu, Dustin Hollar, Shreyas Kuloor

2.0.0 Project Description[2]

2.1.0 Sponsor Background

The sponsor for this project is Deutsche Bank Global Technology (DBGT), a subsidiary of Deutsche Bank (DB). Their office is located in Cary, North Carolina. DBGT creates and manages software and technology solutions for DB, with a goal of improving the effectiveness and efficiency of the software DB uses. For this project, DBGT has asked for the development of software utilizing Optical Character Recognition software that can be used to extract and validate the data in a bank check. Although such software is available via 3rd party and commercial products, DBGT reasons that those products tend to be expensive, and limit the flexibility consumers have over the use of that product. By developing their own software, DBGT has far more flexibility and influence over the evolution of the product so that it fits their needs. So by having us develop this software, they can guide and advise the direction of its development so that it provides the most benefit to them, while also providing us a learning experience.

Sam Nitzkowski (919)-463-3512 - Assistant Vice President <sam.nitzkowski@db.com>

Amit Rahalkar - Vice President: LEMG IT Global Head <amit.rahalkar@db.com>

Andrey Tapekha - Director: Chief Technology Officer (Cary) <andrey.tapekha@db.com>

Eugeny Grebenshchikov - Vice President <eugeny.grebenshchikov@db.com>

Arvind Rajagopalan - Vice President <arvind.rajagopalan@db.com>

Douglas Spence - Assistant Vice President <douglas.spence@db.com>

Artem Sokolov - Assistant Vice President <artem.sokolov@db.com>

Ramkumar Saranathan - Assistant Vice President <ramkumar.saranathan@db.com>

Sean Clinton - Technology Analyst <sean.clinton@db.com>

3000 Centre Green Way, 27513 Cary, USA

2.2.0 Problem Statement

DBGT, as a subsidiary software company of Deutsche Bank, is responsible for developing software solutions to meet the complex needs of Deutsche Bank's financial services. As such, they have asked us for software which can perform Optical Character Recognition (OCR) on a bank check, in order to extract the data from it. OCR allows a consumer to simply upload an image of a check to a program in order to

[2] Primary author: Taha Karimi. Contributing Editors: Tianming Liu, Dustin Hollar, Shreyas Kuloor

deposit a check, rather than having to submit the physical check itself. This is a massive increase in convenience for customers, as they can now deposit checks remotely from the comfort of their own homes, and also saves time and money for the bank, as they no longer have to spend money on scanners and machines that are needed to scan physical checks. DBGT contends that although commercial OCR products exist, they are generally expensive and do not allow the customer much control or flexibility regarding it. They are forced to work within the constraints defined by those products, which are designed generically. The development of such technology that is internal to DBGT would serve them very well, as it allows for customization and adaptation as necessary, and curbs the cost of a commercial product at the same time. The primary purpose of this project then is to develop - from scratch - software to extract the data and provide the verification information for further internal use by DBGT in other processes. In addition, the software is expected to be made with modularity and expandability in mind, so that pieces of it can be extracted and reused elsewhere (ex. Just the OCR piece), and it can be scaled as needed (ex. Add more verification conditions, different languages).

2.3.0 Project Goals and Benefits

The major goals of this project are to develop a software system that can:

1. Provide a web interface for users to upload an image of a bank check.
2. Utilize machine learning and OCR techniques to locate and extract the data from that image.
3. Validate the data.
4. Output the results of that data, both to the user on the web interface, and via a JSON file that allows for further processing of the uploaded check image within the company.

The benefit to the sponsor is a software solution for uploading and verifying bank checks that is tailored to their needs and specifications, can be modularized and applied elsewhere, and can be expanded/scaled as they deem necessary. Some potential practical applications of this software could be fraud detection, such as detecting spoofed or maliciously deposited checks or mobile check deposits for DB customers. They will ultimately be able to integrate the software into their own processes to increase efficiency and convenience for both the business and customers, by automating the extraction and validation of that check data, rather than it being done manually.

2.4.0 Development Methodology

Our team will be utilizing Agile methodology practices. This methodology usually includes the use of sprints, but rather than time boxed sprints, we will be using iterations. Each iteration will include the processes of design, implementation, and testing. Then, at the end of that iteration, we review our progress, determine the necessary tasks to move forward, and begin the next iteration to accomplish these new goals. An assumption and core piece of this methodology is that at the end of each iteration, we have a product that is working, and is continuously improving with each iteration. Our team has weekly meetings outside of class meeting times, as well as weekly sponsor meetings at the end of the week,

where we also show new code to them. Our iterations are two weeks long and we aim to have a functional demo of new features at the end of that time.

2.5.0 Challenges and Resolutions

2.5.1 Technical Challenges

One of the technical challenges our team faces is the use of OCR, in order to perform the data extraction. The problem here is primarily choosing the correct library for OCR, and applying it properly to our problem. Specifically, finding and extracting accurately the hand-written characters found on a bank check will not be as simple as the digital text, and so we need to determine the proper method to address that to extract the data correctly. In addition, there are a lot of check formats on the market and there is no common standard. Some checks do not have distinct notifier for text fields, therefore we need to find the format pattern of each field. For instance, dividing the field by line and finding the line pattern to distinguish each field. This is also an issue of technical feasibility. There are not many open source OCR libraries that work well, and even among those, the number of libraries that work for handwriting recognition are even fewer. We are limited by time for training our own models, and limited by technology on finding pre-existing or pre-trained models, so the challenge is finding a good middle ground. We've proceeded with Tesseract for the time being, as this does seem to be the most accurate of our options when handwriting is neat.

A similar challenge related to technical feasibility is the issue of different check formats. This specifically relates to the field extraction stage. Our first approach to field extraction included using various different machine learning approaches, such as EAST field detection, however it became apparent that these methods are very inconsistent and we often had fields that were cut-off, mis-identified, or sometimes were not found altogether. This means that although the field extraction was able to work on a multitude of different check formats, it did not work well on any of them and we could not expect consistent results. We resolved this challenge by deciding with our sponsors to continue with a single check format, and optimizing our field extraction for that format.

Another technical challenge is machine learning and the training of a model using data with regards to signature consistency. If verification of the bank requires validating the check's signature, and ensuring it is correct according to the original signer and consistent with said signers' past signatures, then this would be a task requiring machine learning with personally trained data. Such a task may require far more time and effort than we have available, and poses an issue in terms of how to execute it as well. Due to that, we have decided that as of now, such validation is outside the scope of our project, and may be addressed at a later date, if time and resources are available.

Similar to the above, another challenge we came across was the use of various languages when writing checks. This posed a problem as it would complicate the process of OCR and especially validation of the extracted data, as we would need to first determine the language of the characters, before attempting to verify them. In addition, check formats also vary, depending on the bank and country, which posed an additional challenge. So here as well, the team has determined these to be outside the scope of our project,

at least in the beginning. We will start with the assumption that all checks that will run through this program will be US checks that contain Latin characters only.

Another technical challenge is the need to make the software expandable and scalable. This is a challenge because we must determine where the software can be improved upon in the future (in terms of functionality), and then incorporate or write code such that the software can actually be modified easily in the future to implement the expansion. An example of this is the issue of language, as right now we are only allowing Latin characters. Our solution to address future allowance of multiple languages is to pass the preferred language(s) in through a config file. Many OCR libraries support multiple languages and can be configured with a function call to change the supported languages, so a user provided option can be passed to the system, which will be enough to configure language support. As of now, we will be working with Latin characters, but the idea is that in the future it can easily be changed to another type of language. Another idea to address scalability is to make the program faster, and a method to do that could be multi-threading. The future expansion that includes multi-threading would use those threads to locate, extract, and verify each piece of data in its own thread, so that the processes are finished almost concurrently, thereby reducing the time for processing. Again however, the issue arises that we need to write code that allows for that future improvement. So when we determine how we want to locate and extract data, we are also attempting to determine how that would affect the future expansions, like multi-threading, and making choices with those in mind.

2.5.2 Legal Challenges

There is only one legal challenge, which is that our sponsors have requested us to use only open source libraries that allow for commercial use. In order to perform OCR, we intend to utilize open source software instead of developing our own, since we do not have time and resources for that. However, depending on the open source software's license, some require that any code that uses that library must be made publicly available in full, and this is something our sponsors at DBGT do not want. Therefore, our solution is to avoid any open source software that has such conditions, which may limit us in our choices, but not by much. The licenses of the libraries we are using - such as MIT, BSD, and Apache 2.0 - fall under the correct category, so we are not breaching the sponsors request. In addition, DBGT has given us permission to use the final product in our own portfolios when seeking jobs.

3.0.0 Requirements[3]

3.1.0 Overview

The software shall be able to provide users the ability to upload a check image to a website, then extract and validate the data. The system shall provide output divided by different regions, or fields, on the check image. The system shall also handle any invalid data (as defined below) on the check and provide an

[3] Primary Authors: Dustin Hollar, Tianming Liu, Taha Karimi, Shreyas Kuloor

indication of that. Users shall see results output to the web interface and could obtain the JSON version of the output by clicking a button on the web interface.

3.2.0 Use Cases

3.2.1 UC1 - Upload Check

Description: The user can upload a check image to the system for image processing and verification.
Pre-conditions: On the website landing page, the website shall display an upload button that the user can click. Only files of the format [png, jpeg, jpg, gif, pdf] can be selected and uploaded.
Post-conditions: The user has uploaded a check image to the system with one of the supported formats.
Main Flow:

- The user clicks the upload button.
- The system opens the computer's file explorer and the user can select an image file.
- The user selects an image file [S1][E1].

Sub Flows:

- [S1] A user can upload one check image for processing at a time.

Alternative Flows:

- [E1] The file explorer will only allow the user to select supported file types [png, jpeg, jpg, gif, pdf].

3.2.2 UC2 - Extract Check Data

Description: The check image is divided into several regions for the different fields and the values for these fields on the check are extracted from the image (See Figure 1 as an example of the fields extracted).
Pre-conditions: The check has been successfully uploaded to the system.
Post-conditions: The check fields listed below have been extracted.
Main Flow:

- The check is divided into fields, and the following data is extracted [E1]:
    - Date [S1]
    - Pay to the Order of/Payable to [S2]
    - Number Amount [S3]
    - Written Out Amount [S4]
    - Signature [S5]
    - Memo [S6]
    - Routing/Account Number [S7]

Sub Flows:

- [S1] When the check image with the region containing date is passed to extraction, the system shall locate the date within the region, and then extract the data from the image to text.
- [S2] When the check image with the region containing Pay to the Order of is passed to extraction, the system shall locate the Pay to the Order Of within the region, and then extract the data from the image to text.

- [S3] When the check image with the region containing Number Amount is passed to extraction, the system shall locate the Number Amount within the region, and then extract the data from the image to text.
- [S4] When the check image with the region containing Written Out Amount is passed to extraction, the system shall locate the Written Out Amount within the region, and then extract the data from the image to text.
- [S5] When the check image with the region containing Signature is passed to extraction, the system shall locate the Signature within the region, and then extract the data from the image to text.
- [S6] When the check image with the region containing Memo is passed to extraction, the system shall locate the Memo within the region, and then extract the data from the image to text.
- [S7] When the check image with the region containing Routing/Account Number is passed to extraction, the system shall locate the Routing/Account Number within the region, and then extract the data from the image to text.

Alternative Flows:

- [E1] If the uploaded image is not a check image, garbage/random data is extracted.

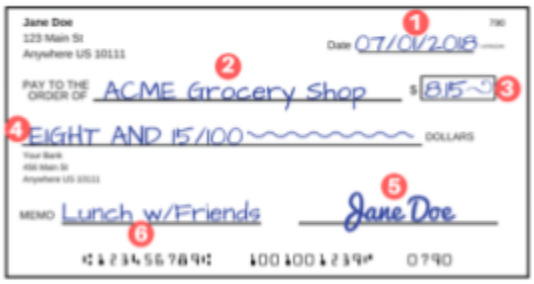


Figure 1 - Example Check Layout

3.2.3 UC3 - Verify Check Data

Description: After the check data is extracted, the fields will be evaluated for validity and reasonability, and the results will be displayed on the page and output to the JSON file (See UC4).

Pre-conditions: The check fields have been extracted.

Post-conditions: The results of the check analysis is displayed on the web interface and output to a JSON file for review.

Main Flow:

- The following extracted data will be evaluated for validity and reasonability:
  - Signature [S1]
  - Routing and Account numbers [S2]
  - Dates [S3][S4]
  - Pay to the Order of [S5]
  - Number Amount and Written Out Amount [S6][S7]
  - Memo [S8]

Sub Flows:

- [S1] The extracted Signature is verified by checking that it exists. Meaning that the extracted text from the Signature field was not blank/empty [E1][E12].
- [S2] The Routing/Account Number shall be extracted as 2 fields: the first field is the Routing Number, and is from the list of valid routing numbers [E2]. The second field has 4-17 characters and is the Account Number [E3][E12].
    - Valid Routing Numbers
- [S3] Program will check Date of Issue/Signing/Submission for valid date formatting, and output a passing status to the JSON file (See UC4) [E4][E12]. Valid formats:
    - ISO (2019-09-04)
    - Short Date (09/04/2019)
    - Long Date (September 9, 2019)
- [S4] Program will check Date of Issue/Signing/Submission for valid dates, and output a passing status to the JSON file (UC4) [E5][E12]. Validation checks:
    - Check that Date of Submission > Date of Signing
    - Check that time between the Date of Signing and Date of Submission does not exceed a configurable amount
- [S5] Program will check the 'Pay to the Order of' extracted data to ensure the characters are from the Latin alphabet, and output a passing status to the JSON file (UC4) [E6][E12].
- [S6] Program will check the extracted Number Amount field to ensure the characters are Latin numbers [E7], then check if it is below a configurable upper threshold, and above a configurable lower threshold (it must be between these two thresholds, which are configurable by the user) [E8][E12].
- [S7] Program will check the extracted Written Out Amount field to ensure the characters are from the Latin alphabet, and convertible to Latin numbers [E9], then check if it is below a configurable upper threshold, and above a configurable lower threshold (it must be between these two thresholds, which are configurable by the user) [E10][E12].
- [S8] Program will check the extracted Memo field to ensure the field is not empty, and output a passing status to the JSON file (UC4) [E11][E12].

Alternative Flows:

- [E1]: If the extracted Signature is empty, this means that the check was not signed, and the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E2]: If the extracted Routing Number does not match one of the valid routing numbers, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E3]: If the extracted Account Number is not 4-17 digits, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E4] If one of the extracted date fields are not in the valid format, then the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E5] If the extracted date (the date of signing) is less than the Date of Submission, or the time period between the Date of Signing and Date of Submission is greater than the configurable value, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E6] If the extracted Pay to the Order of characters are not from the Latin alphabet, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.

- [E7] If the extracted Number Amount is not Latin numbers, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E8] If the extracted Number Amount is not between the upper and lower thresholds, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E9] If the extracted Written Out Amount is not from the Latin alphabet, or cannot be converted to Latin numbers, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E10] If the extracted Written Out Amount is not between the upper and lower thresholds, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E11] If the extracted Memo field is not a string, the program outputs a failing status to the JSON file (UC4), as well as to the web interface.
- [E12] If a field is read incorrectly, the system will allow the user to manually enter the correct value for that field on the web interface, and it can then be re-validated (UC5).

3.2.4 UC4 - Display Output

Description: The website will display the results of the validation on the page and also contain a link to a JSON file that displays the results of reasonability checks after a check has been uploaded. The displayed results are editable by users, so they can correct fields that did not PASS, and a 'Re-Validate' button will also be on the website.
Pre-conditions: The user has uploaded a check image to the system (UC1).
Post-conditions: The user will be shown the results of the reasonability checks on the website page and can view it in JSON format, or change and re-validate the data (UC5).
Main Flow:

- The use case begins when the user wants to view the results of the OCR image extraction on their check image after upload.
- After data processing and validation has been completed, the results will be dynamically displayed on the same page and a "Get JSON" button appears on the page [S1].
- The results are also changeable, and a "Re-Validate" Button also appears on the page (UC5).
- The user clicks the "Get JSON" button on the page.
- A new tab containing the JSON output will be displayed, and users have the ability to download that output as a JSON file via their User Agent [S2].

Sub Flows:

- [S1] The system will show the results for the fields that were read in, along with whether or not they passed/failed validation:
    - Routing Number
    - Account Number
    - Date of Signing
    - Pay to the Order Of
    - Number Amount
    - Written Out Amount
    - Memo
    - Signature

- [S2] The system will show the results for the following fields that were read in, along with whether or not they passed/failed validation and the coordinates, height, and width of their bounding boxes (the region the system believes the text was in):
    - Routing Number
    - Account Number
    - Date of Signing
    - Pay to the Order Of
    - Number Amount
    - Written Out Amount
    - Memo
    - Signature

3.2.5 UC5 - Re-Validate Data

Description: The website will allow users to modify the text the system extracts from the check, and send that data to be validated again by clicking the "Re-Validate" button that is below the "Get JSON" button on the page.
Pre-conditions: The user has uploaded a check image to the system (UC1). The data has been processed, validated, and displayed on the web page (UC2, UC3, UC4).
Post-conditions: The user will be shown the results of the re-validation on the website page and can view it in JSON format, change it, and re-validate again.
Main Flow:

- The user changes the text that was extracted from the check by the system and then clicks the "Re-Validate" button, or clicks the "Re-Validate" button without changing data. The data (whether it was changed or not) is then sent to the backend for validation again [S1].
- When that is completed the results will be dynamically updated and displayed on the same page, where the user can also view the JSON output (UC4).
- The user can then change data and re-validate again [S1].

Sub Flows:

- [S1] When the "Re-Validate" button is clicked, all of the data (whether it was changed or not) is sent to the backend, and each of the following fields is run through the validation checks again. Results are then sent back to the frontend, and the website is updated with the new results.
    - Routing Number
    - Account Number
    - Date of Signing
    - Pay to the Order Of
    - Number Amount
    - Written Out Amount
    - Memo
    - Signature

3.3.0 Non-Functional Requirements

3.3.1 NFR1 - Software Extendibility

1. The software should be extendable so that checks from regions other than the United States can be supported in the future.
2. The software should be extendable so that non-Latin characters on checks can be supported in the future.

3.3.2 NFR2 - Software Modularity

1. The program should be modular so that a discrete stage of the program can be extracted and used for another program.
2. The program should be modular so that new discrete stages of the program can be added to the software.

3.4.0 Constraints

- The software shall be written using Python as the programming language for the backend, and Python Flask, HTML, CSS for the frontend.
- The unittest testing framework shall be used to unit test the software.
- All software/libraries used must be open source and available for commercial use.
- Check fields are filled out using Latin characters.

4.0.0 Resources Needed[4]

Resource	Status	Description/Justification
Python 3.6-3.7	Obtained	A high-level programming language with a simple method to create web applications.
Flask	Obtained	A python library/framework for simple creation of web applications.
PyTesseract	Obtained	An OCR library for Python that we intend to use for our project to simplify the locating and extracting of data.

[4] Primary author: Taha Karimi. Contributing Editors: Tianming Liu, Dustin Hollar, Shreyas Kuloor

Tesseract	Obtained	An optical character recognition engine for various operating systems.
OpenCV	Obtained	Open source library containing functionality for real time computer vision.
PyUnit or unittest	Obtained	Unit testing libraries made for Python, we intend to use for our white box testing.
TensorFlow 1.14	Obtained	An end-to-end open source machine learning platform.

5.0.0 Design[5]

5.1.0 Software Architecture

The high level process of this program is as follows: the user uploads an image of a check, the system then preprocesses the check by removing noise, grayscaling it, and removing shadows from it. The system then determines the regions of interest on the check, extracts the data/text in those regions, and then validates them. The system then sends the extracted data to the frontend (website) for the user to evaluate.

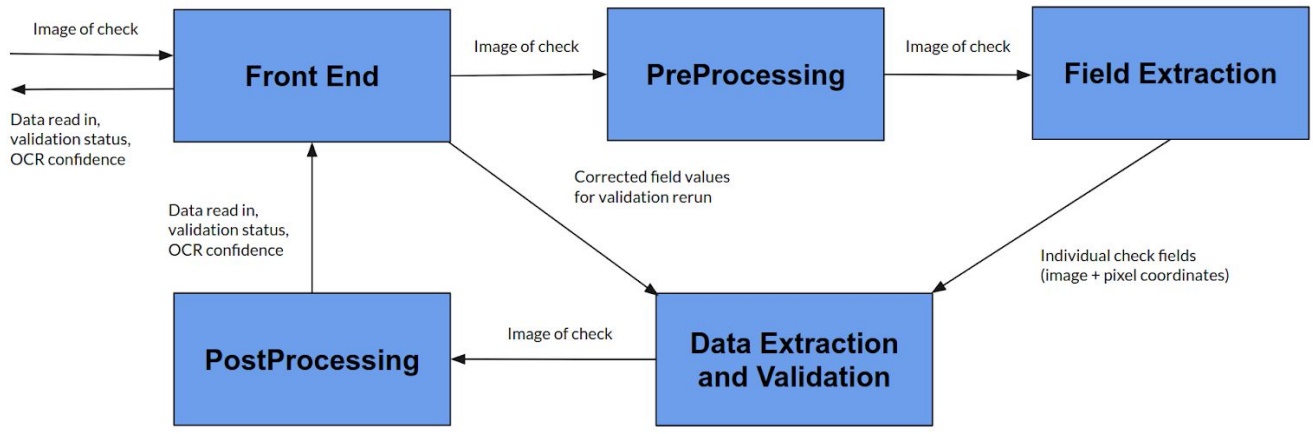


Figure 2: Pipeline Architecture

[5] Primary Author: Dustin Hollar, Taha Karimi. Contributing Editors: Tianming Liu, Shreyas Kuloor

As described above and displayed in **Figure 2**, there are five discrete stages of the system process: Front End, Preprocessing, Field Extraction, Data Extraction, and Postprocessing. Each stage has a designated use with a set of inputs and outputs. The next sections describe the stages of the pipeline in further detail.

5.1.1 Front End Stage

The first stage of the process has two criteria to fulfill. First, it must allow a user to upload an image of a check for processing. Second, a user must be able to view the processed image to evaluate a successful upload or a failure. Red bounding boxes are drawn on the image to show the regions on the check that were evaluated by the program. A user can edit the returned data and re-submit the data for validation.

When the user uploads a check, the image is passed to the PreProcessing stage. Once the program finishes running, a JSON file containing the extracted data and the processed image of the check are passed back to the front end. If the user chooses to re-validate the data, control is passed to the data extraction and validation is performed a second time.

Each criterion has a different set of input/output. For the first criteria, there are no direct inputs as this is the start of the program. The output is the image of the uploaded check, which is passed to the *PreProcessing* stage. The second criteria has the input of extracted data passed from the *Data Extraction* Stage of the pipeline. Since the user can request to perform further validation on the extracted data, the output of the second criteria is the edited data and is passed back to the *Data Extraction* stage and is marked for validation only.

5.1.2 Preprocessing and Field Extraction Stage

These two stages perform several passes on the check. A pass is where the entire image of the check is processed a single time. These passes can be split into two types: preprocessing and field extraction. Preprocessing passes can have 0...N number of passes over the image of the check. Examples of preprocessing steps can be downsampling a high resolution image, rotating the image, or denoising an image. The purpose of these passes is to make it easier to extract the fields and the data in later passes/stages. The second type of processing pass, field extraction, finds the locations of all of the regions of texts and marks a bounding box's minimum and maximum pixel coordinates around the field. There can be 1...N number of passes over the check for this type. Examples of these passes would be first finding all of the lines of text followed by a second pass over each line to extract the regions of text.

Input to this stage is the image of the check to process and the output of the stage is a structured list containing the extracted fields. Each element in the list, called a *field*, contains the bounding box's minimum and maximum locations on the image and an enumerated type that has not been set yet. The enumerated type is set in a later stage in the pipeline. The output of this stage is passed to the *Data Extraction* stage.

5.1.3 Data Extraction and Validation Stage

This stage has three steps: extract, identify and validate the data. A single field that had been extracted from the previous stage is passed to the *Data Extraction* stage. The data contained in the field is extracted

and its type identified. When identified, the field is validated based on its type. Once the field has been fully processed, the structure of the extracted data and original input is passed as output for this stage.

5.2.0 Front End Design

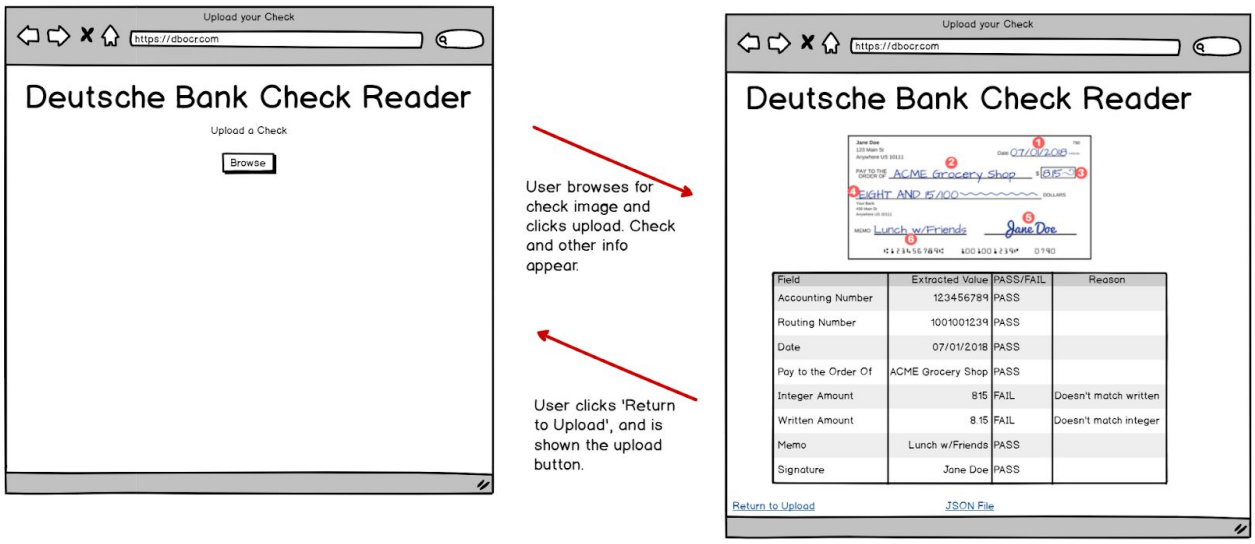


Figure 3: Front End Design

The wireframe in **Figure 3** shows the expected user flow when using our software. Upon running the software and navigating to localhost port 5000 on a User Agent/browser, the user sees the page shown on the left of the wireframe. This gives them the option to browse their computer for an image of a check to process. If they click this button and upload an image, the check is processed, and the current elements disappear, and the elements in the page on the right side appear. The information obtained after processing the check is shown, namely an image of the check with bounded fields, and the individual fields with the results of the extraction and validation. Those results contain the field name, extracted value, and whether the field passed the validation or not. An option to view the data as JSON output is also available, as well as the ability to modify the extracted field values, and re-submit them for validation. The user can also upload another check image for processing.

5.3.0 Software Design

As shown in Section 5.2, this program follows a pipeline architecture, which serves to drive the design of the program. A module is created from each stage of the pipeline, and each module has its own internal design to match its functionality. Section 5.3.1 presents the design for how each module communicates with the other modules. Sections 5.3.2-5 discuss the internal design of each module along with the expected inputs and outputs of the module.

5.3.1 Stateless Design

The program subscribes to a stateless design paradigm, meaning that no state is retained between function calls. Data required by the pipeline is created by the controller and subsequent stages interface with that data. The pipeline passes the data from one stage to the next, allowing for each stage to not require persistent state. This is done to reduce the overall memory footprint of the program. Text extraction techniques used in the Data Extraction stage have a high memory consumption, so limiting memory usage in other portions of the program is imperative for an efficient implementation.

In order to prevent modules from having to maintain information about an image and the fields, the following dataclasses were implemented as Python data classes.

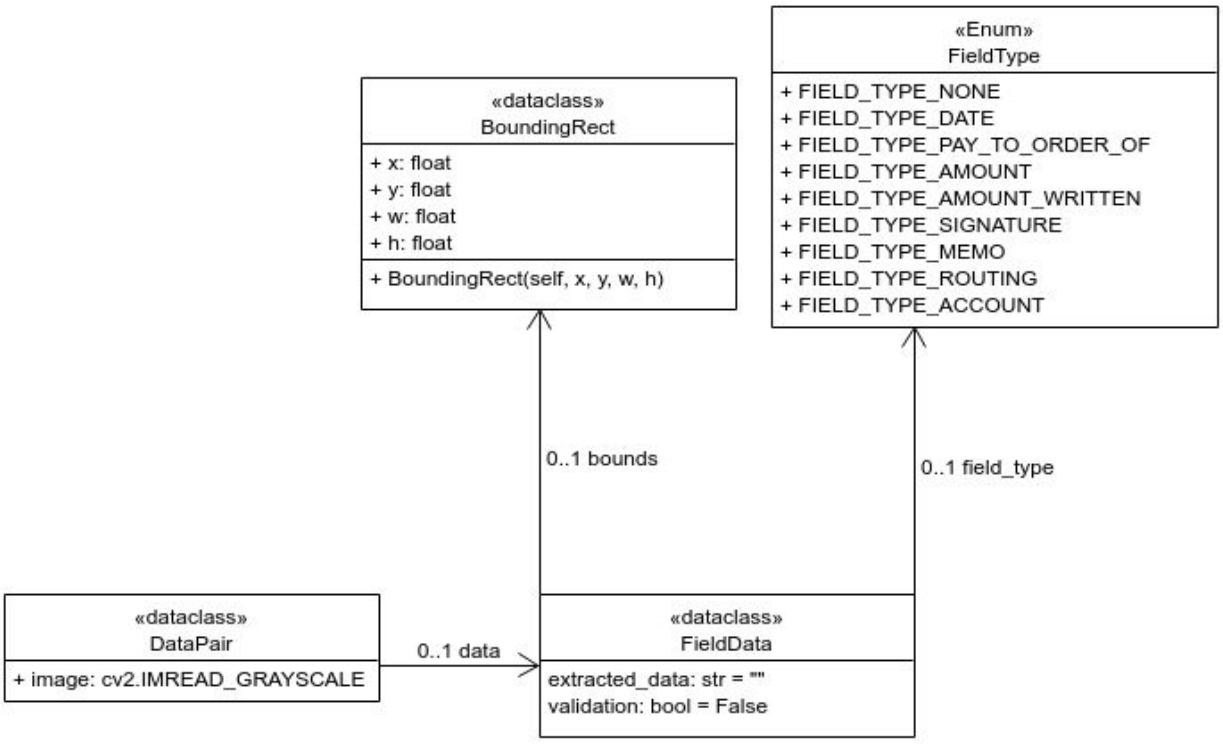


Figure 4: Data UML

The FieldData class represents a single field on a check and will contain bounding rectangle representing the field's location on the check, a type, and the extracted info. A final data class is used, DataPair, is used to pair the field data to an image of the field. Ideally, the image is a cropped version of the overall image.

The data class, DataPair, is what will allow the program to remain stateless. Each Pipeline Stage is passed a reference of a DataPair to fill out a set of designated information. The Field Extraction Stage, a DataPair is constructed for every field extracted from the check. The image of the check is cropped and stored in the image variable. Only the bounding rect is initialized at this point; field_type is set to

FIELD_TYPE_NONE and data_info is set to be an empty string. In the Data Extraction Stage, the extraction pass fills out the extracted data in the data_info variable and the validation pass marks whether or not the field was valid.

At the end of the pipeline, a list of completed DataPairs is converted to a JSON string and are sent to the frontend via Rest API calls. During a single pass through the pipeline, the only state that is constructed is the list of DataPairs, which are maintained by a controller**.**

There is one exception in this design, which is a global dictionary of the various implementations of the Field Interface. This is discussed in further detail in Section 5.3.5. However, this dictionary is global to the program, and is constructed and maintained at project startup and maintains no internal state.

5.3.2 Module Communication

For the pipeline to progress and to have each stage independent from the other, a controller is required. This controller acts as the entry point for the backend. A single function, *controller_entry_point*, can be called by the frontend, who passes the the image to the controller via REST API endpoints. The controller's role in the program is to delegate work to each stage of the pipeline. The following sequence diagram illustrates this process.

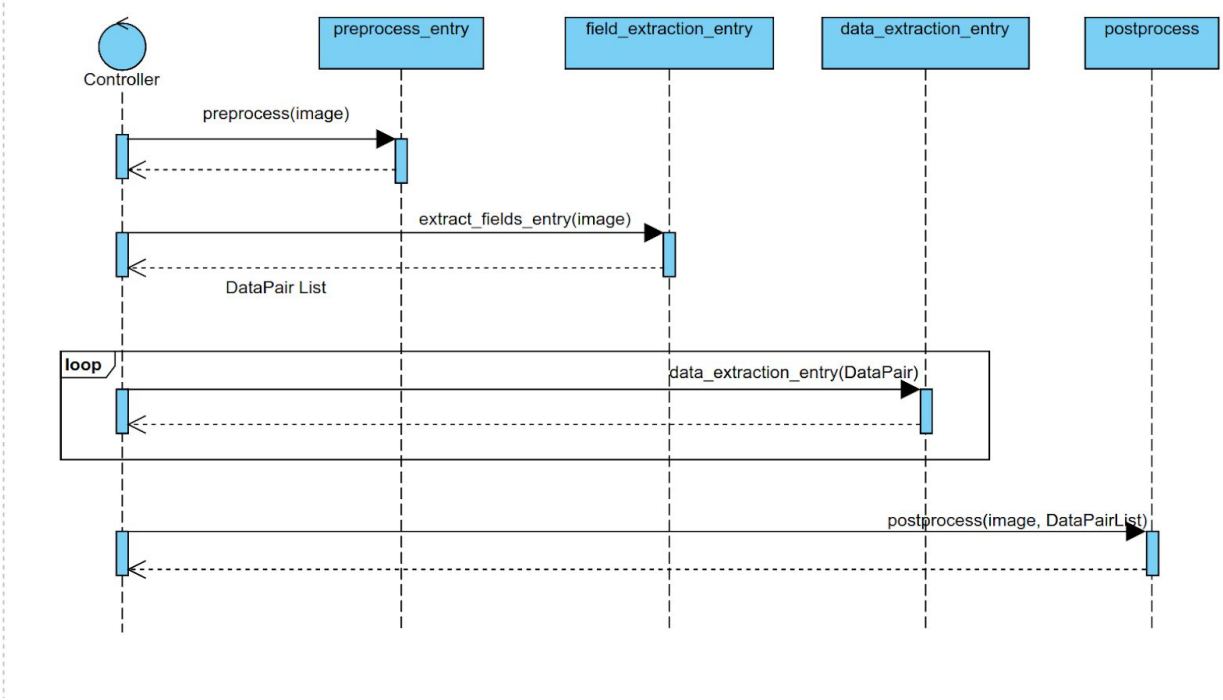


Figure 5: Module Relation Sequence Diagram

As can be seen above, each module has a distinct entry point that is called by the controller. The modules have a unique internal structure that is discussed in the following sections. It should be noted that the call

to *data_extraction_entry_point* is encased within a loop. Rather than send all fields to the Data Extraction stage, only a single field is sent. This is done to allow for the controller to be extended to a multithreaded design in a future iteration of the project. This idea is discussed in Section 5.4.

5.3.3 Pre/Post Processing

The pre- and post-processing stages of the pipeline have a straightforward design. Preprocessing will have three major functionalities. The first is to downscale the image should it be too high a resolution. This is done to decrease the number of pixels that have to be processed in later stages of the pipeline. The second is to grayscale the image, as many of the techniques used in the field and data extraction stages require the image to be grayscale. A developer has the option of utilizing several other features like removing shadow/lighting effect by normalizing the luminance channel of an image.

Preprocessing one last approach, which is detecting lines in an image. While this approach was a success, we found that it tended to remove parts of text and was unreliable. A future developer has the option to incorporate this approach should the situation allow it. However, the preprocessing stage does not actively utilize this functionality.

The post-processing stage has two jobs to perform. For each detected field on the check, a bounding region is drawn over the area on the check. This is done so that an end user can visualize the output of the program. The second job serializes the extracted data into a JSON format. Both the JSON data and processed image are sent to the frontend.

5.3.4 Field Extraction

The field extraction stage has a single routine that extracts the fields on the check. The controller passes the processed image to the entry point of this stage, and each field is detected and stored in a list of DataPairs. See Section 6.6.1 for further detail on the implementation of this stage.

5.3.5 Data Extraction

Data Extraction contains the most complex functionality of the stages. To illustrate the control flow of the stage, the following control flow diagram is provided.

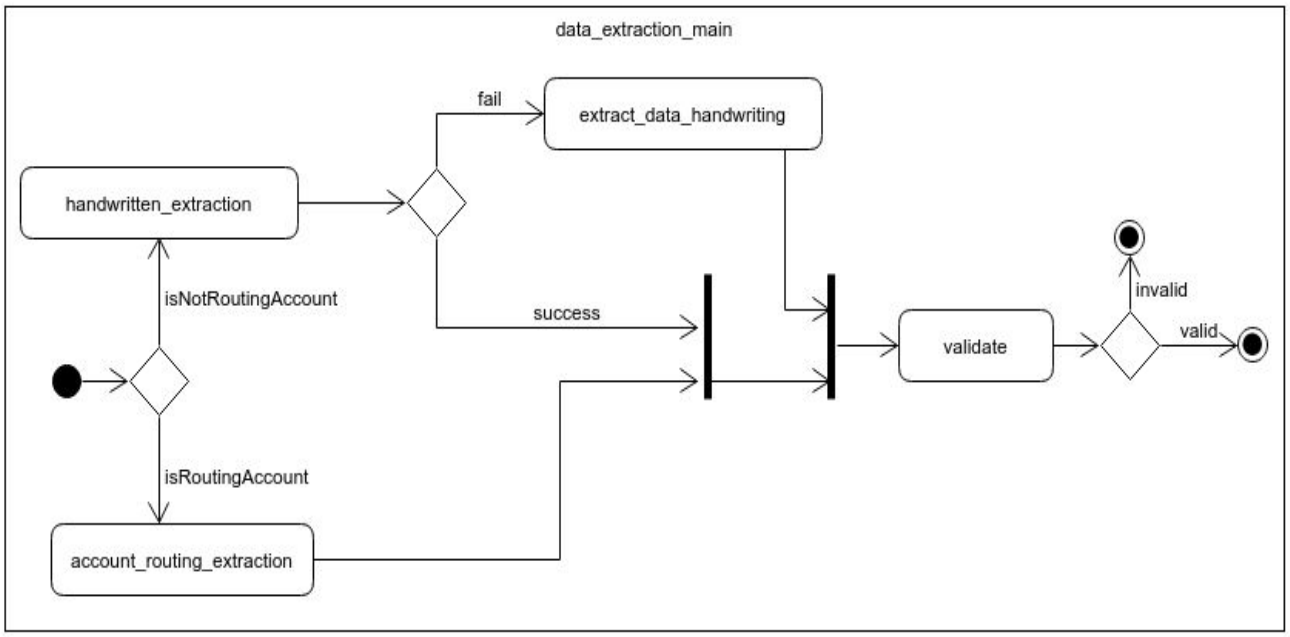


Figure 6: Data Extraction Control Flow Diagram

Upon entering the data extraction stage, the *handwritten_extraction* function is called for each individual field identified by field extraction. If possible the extracted data is returned and set in the field data class. The data is then validated using an interface hierarchy. The following UML shows this relationship.

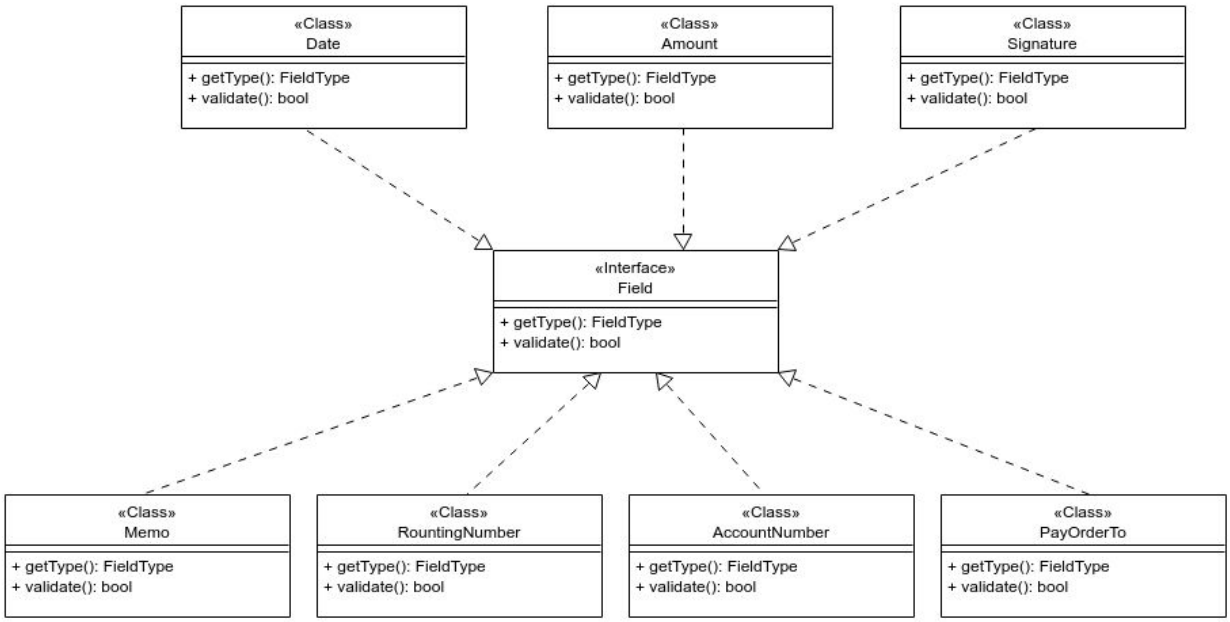


Figure 7: Field Interface UML

A single interface, *Field*, contains a validation function. Each unique field on a check is given a class that inherits from *Field*. The *Amount* field is slightly different than the others, as it has a written out section and a number section on a check. So it's validation is run a little differently, as the written out amount is first converted to an integer amount. Once the field extraction stage completes the controller does a final processing pass on the *Amount* fields to compare them to make sure they match.

5.3.6 JSON Formatting

Use Case 4 requires that data output from the extraction stages be output to a readable format, so the data classes shown in the Field Hierarchy UML are set up to provide quick conversion to a JSON string. A sample JSON object of a single *Field* is shown below.

```
"element0": {
    "field_type": Enum,
    "bounds": {
        "x": integer,
        "y": integer,
        "w": integer,
        "h": integer
    },
    "extracted_data": String,
    "validation": Bool
}
```

Once the pipeline completes the data extraction stage, all fields are serialized into a single JSON string.

5.3.7 Rest API Endpoints

In order to decouple the frontend and the backend to improve modularity, Rest API endpoints are used for communication. The following API calls are listed below.

1. POST: */api/upload/*
    a. Used to send an image to the backend for processing. Returns the output JSON object containing extracted data, bounding boxes, validation status, and field type.
2. POST: */api/revalidate/*
    a. Used to send data back to the backend for revalidation. Returns an updated JSON object containing the resubmitted JSON object with the updated validation statuses.
3. GET: /
    a. Returns the main frontend html page.

5.4.0 Architectural Implications on Future Design

When discussing the design, a major concern was performance. Image recognition can be an expensive operation, so it was important to choose a design that can promote performance as much as possible. One area that was considered was a single threaded versus a multithreaded application. While creating a multithreaded program is currently beyond the scope of this project, it is important to design the program for multithreading if there is a possibility of the program being adapted for a multithreaded implementation. The pipeline architecture is designed to support multithreading should a later iteration add this functionality. The *Field Extraction* stage outputs a list of fields for the *Data Extraction* stage to process, but this stage only processes a single field at a time. As each field is independent of the other, the *Data Extraction* stage can be multithreaded to process multiple fields at the same time!

Another consideration was a system that queues a series of checks to process. In a single stage program, the queue would process a check one by one, which is a slow process. However, with a pipeline architecture, the queue would be able to start processing the next check after the previous check finishes the first stage of the pipeline. To illustrate this concept, Check 1 enters the first stage of the pipeline, and all other checks are in queue. Check 1 completes the first stage and then moves to the second stage, then Check 2 enters the first stage of the pipeline. This continues until all checks have been processed. Now the throughput is the length of the longest stage to complete rather than the combinations of all stages. A pipeline architecture is a good fit for a system with discrete steps and for when it is needed to process many similar items.

6.0.0 Implementation[6]

As a general overview of the project implementation, there are two main pieces: a frontend and a backend. The frontend will handle the uploading of an image, display of the processed data, and provide the ability to download a JSON file. To do this, it will use the Flask web framework. The backend will handle all of the data processing, so the preprocessing, field extraction, data extraction, data validation, and any post-processing, and then return that data to the frontend. This will be implemented in Python.

6.1.0 Iteration Definition

In order to accomplish the tasks mentioned above, we have decided to split our work into four iterations, which are centered around different functional requirements, and are two weeks each. For the first iteration, we are to implement a simple but working frontend which allows: navigation to a site, the ability for users to upload an image, and functionality for the smaller pieces such as displaying the data that was processed and allowing for the download of a JSON file. As for the backend, we are to finish field extraction, which means finding and bounding all of the locations/regions of text on the check. We are also to finish data extraction for a handwritten amount, specifically the Number Amount on a check. Lastly, we are also to implement data validation for that extracted Number Amount.

For the second iteration, on the backend of the system, we are to implement the non-handwriting data extraction of the Account and Routing Number field and test and improve the handwriting data extraction that was implemented in the previous iteration. On the frontend, we are to finish the REST API for communication between the frontend and the backend. And lastly, for this iteration, we are to collect data such as check images and other images for future testing of the software.

In the third iteration, we are to implement field identification and expand on field extraction for every field on the check. That is, once a ‘field’ (a region of text) has been extracted from a check, it is then identified. We must then identify what type of field it is, e.g. Memo or Date. Data validation of each field will then be implemented, and that is all for iteration three.

In the last iteration, iteration four, we are to focus on extensive testing of the program, in order to flesh out any bugs or reveal any flaws. Further polishing of the code, such as conciseness, formatting, reducing redundancy or time taken for execution, as well as any extra or bonus features would be added in this iteration. No bonus features have been discussed yet except for multi-threading, so as of now that is the only option on the table.

[6] Primary Authors: Tianming Liu, Dustin Hollar, Taha Karimi, Shreyas Kuloor

6.2.0 Current Status

Regarding our current status, Iterations 1 through 4 have passed and we have completed a functional end to end system that meets all requirements stated in Section 3.0.0. A dynamic front-end website has been completed through the use of HTML, CSS, and Javascript. This front-end website communicates with our backend through a REST API to send check images and retrieve results. Preprocessing to remove noise, watermarks, shadows, and other image distortions has been completed. Field Extraction has been completed, and the primary eight check fields (Date, Pay to the Order Of, Numeric Amount, Written Amount, Memo, Signature, Routing Number, and Account Number) are being identified and bounded, although currently is only functional for one specific type of check format due to time constraints (see Figure 8). Data extraction is functional, but has some limitations. Currently, due to a reliance on Tesseract as the primary method of data extraction, checks must be written neatly for data to be extracted accurately. The number of pre-trained models for handwriting OCR is very limited, and those that do exist are trained on very few handwriting styles. Given more time, we could have trained a model ourselves with a much more extensive dataset for handwriting, and this may have increased the accuracy of the data extraction stage. Postprocessing has also been completed. Bounding boxes are being drawn on the original input check image and this updated image is matched with output data serialized to a JSON format. This packet is sent to the front-end for display.

Figure 8: Check Image Format

6.3.0 Security Considerations

In order to address security concerns about interacting with personal data, the program does not maintain a list of client data. The program is designed to take an image of a check and pass the extracted data to a different system. This way no client data is stored within the system. To further this approach, the

program does not contain any persistent state related to the check data. The program passes the data along when it finishes processing. There were no other security considerations taken throughout the project. Our focus was developing the key functionalities in our requirements, and as requested by our sponsor, minimal priority was put on API and communication security.

6.4.0 Project Folder Structure

- ➔ **src/**
  - ◆ **main/**
    - **templates/**
      - home.html
    - **config/**
      - config.yml
    - **resources/**
      - **east/**
        - ◆ frozen_east_text_detection.pb
    - **backend/**
      - **data_extraction/**
        - ◆ **field/**
          - **data/**
            - field_data.py
          - account_number_field.py
          - amount_field.py
          - date_field.py
          - field.py
          - memo_field.py
          - pay_to_order_of_field_py
          - routing_number_field.py
          - signature_field.py
        - ◆ **handwriting_extract/**
          - main.py
        - ◆ config_helper.py
        - ◆ data_extraction_main.py
        - ◆ extract_methods.py
        - ◆ field_list.py
      - **field_extraction/**
        - ◆ field_extractor_main.py
      - **postprocess/**
        - ◆ postprocess_main.py
      - **preprocess/**
        - ◆ preprocess_main.py
      - **utils/**

                - cv_utils.py
                - json_utils.py
            - controller.py
        - main.py
- **test/**
    - **field_extraction/**
        - test_field_extraction.py
    - **input/**
        - Input images used in black box testing
    - **test-files/**
        - Input images used in white box testing
    - **unit/**
        - **data_extraction/**
            - extract_methods_test.py
        - **field/**
            - account_number_field_test.py
            - amount_field_test.py
            - date_field_test.py
            - memo_field_test.py
            - pay_to_order_field_test.py
            - routing_number_field_tes.tpy
            - Signature_field_test.py
- setup.py
- requirements.txt
- compile_libs.sh

We chose this folder structure firstly in order to separate our main source code from our test code. Secondly, it allows us to separate the frontend (which is just main.py and the /templates folder) from the backend. Lastly, we are also able to isolate each stage of the project into sub-directories of the backend folder, so that the code for each stage remains separate.

6.5.0 Project Configuration/Settings

There is a config file named *config.yml* located in the *src/main/config* directory of the project. This file is the central location for all configurable values in our system. Figure 9 is an image of the default values contained in *config.yml*.

```yaml
thresholds:
  # Maximum allowed check amount in dollars.
  amount_max: 100000

  # Minimum allowed check amount in dollars.
  amount_min: 0

  #The maximum age of a check in months before it is considered invalid.
  check_age_limit_months: 6
```

Figure 9: Default config.yml

As of now, the only configurable values are the maximum amount (amount_max) and minimum amount (amount_min) for the amount field used during check validation, and the maximum age of a check (check_age_limit_months). The amount_max value specifies the maximum allowed amount on the check (default: 100000). Any value higher than this configurable value on a check will result in the field failing validation. Similarly, the amount_min field specifies the minimum allowed amount field on the check (default: 0). Any value lower than this configurable value will result in the field failing validation. The check_age_limit_months value specifies the maximum age in months of a check that will be allowed through validation. The check date is compared to the current date, and if the difference exceeds this value, the check will be rejected.

6.6.0 Algorithm Implementation

6.6.1 Field Extraction

The overall goal for the field extraction stage is to compute the bounding regions of each field and to crop the region of the image that these fields represent. There are three approaches implemented: Computer Vision Techniques, EAST Text Detector, and a Custom Implementation. Each technique has its own benefits and shortcomings, but the implementation that is currently used is the custom approach. The first two are also provided as they can potentially assist in future techniques.

The first approach to detecting fields uses several algorithms provided by the OpenCV library. To start with, thresholding is applied to a grayscale image. An example of a simple thresholding process is when a threshold is provided and all pixel values above the threshold are set to 1 and all values below are set to 0. This process can be seen in **Figure 10**[7]. This is done to isolate all text on the image from the background. A resulting effect could be all text is white and the background is black.

[7] Image Source: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

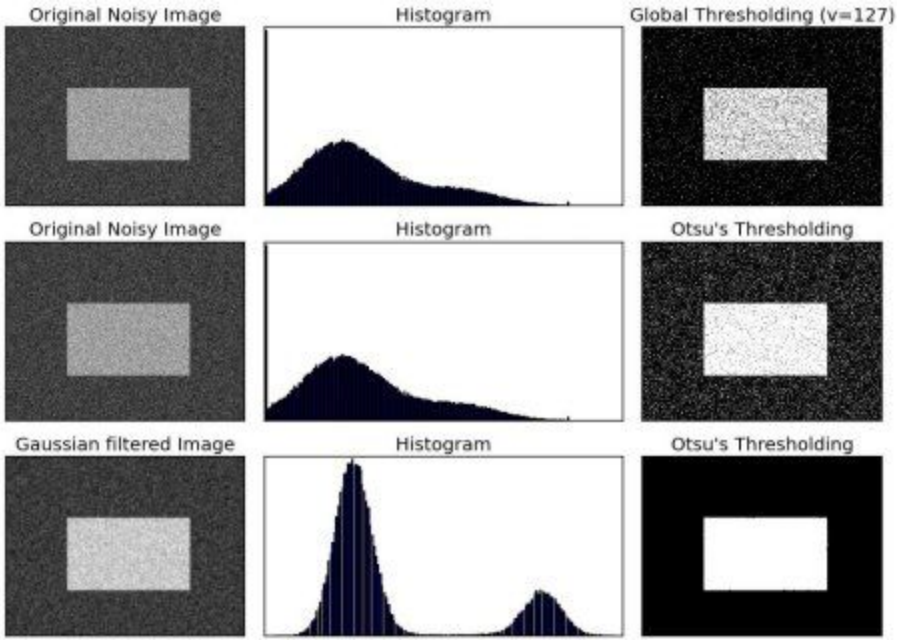


Figure 10: OpenCV Image Dilation

Next, all white pixels in the image are dilated. The dilation is shown in **Figure 11**[8], where the white pixels are expanded into the black pixels. This is done to merge text together for the final step in the algorithm

Figure 11: OpenCV Thresholding

Using the dilated image, the bounding region can be computed using a contouring algorithm from the OpenCV library. This algorithm joins curves together into a single region. The minimum and maximum bounds along with the extent of the region can be computed from the generated curve. Once the bounding region has been computed, a DataPair is created with the image being a cropped image of the bounding region and a FieldData with only the bounding region filled in.

[8] Image Source:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

The second approach to implementing field extraction utilizes the EAST Text Detector, which is a pre-trained model that detects text in a natural image. An example of a "natural image" would be an image casually taken from a phone. Alone the EAST Text Detector was not much use, as it calculated bounding region rather inaccurately, so two heuristics were added to algorithm. First, all overlapping bounding regions are merged together. Second, all nearby regions were merged together based on a provided threshold. While this algorithm could reliable find all text in an image, it struggled to consistently find bounding boxes and the aforementioned heuristics did not do much to solve the error. However, this approach works fairly well when the written text is close together and not spread out.

This final approach provided the most consistent approach to detecting fields, so field extraction currently utilizes this algorithm. There are two steps: an offline and online step. The offline step scans over a series of images searching for a set of potential bounding boxes. A developer can take a look at the set of output images to mark the successful and unsuccessful scans. The parameters gained from this offline step are read from a config file by the field extraction module and are used to isolate the fields. Unlike the previous two approaches, this one estimates the field type during the extraction and removes any need for identification during the data extraction stage.

The major weakness of this approach is that we have to have parameters to a specific type of check in order to obtain consistent output. This approach could benefit greatly from more machine learning approach that would allow it to handle multiple types of checks. Another benefit the offline step could use is a system that provides marked images of a check. A marked image would be an image that a developer has marked where the maximum bounding region for each field should be. This would allow for the offline step to look for these marks rather than the current brute force approach.

6.6.2 Data Extraction

As of now, we are not personally creating any algorithms for the data extraction stage of the project. We are, however, implementing libraries that utilize algorithms to perform OCR. The key OCR libraries we are using for this project are Tesseract, an open source Github project Handwritten Line Text Recognition using Deep Learning with Tensorflow (https://github.com/sushant097/Handwritten-Line-Text-Recognition-using-Deep-Learning-with-Tensorflow), and the use of template matching for routing/account number extraction. Tesseract uses an implementation of the Long short-term memory (LSTM) architecture, which is a type of artificial recurrent neural network. The LSTM architecture utilizes cells, input and output gates, and forget gates to store values over time and control the flow of information between nodes. A collection of many of these nodes forms the recognition engine that Tesseract uses to perform OCR. Tesseract comes pre-trained with many different fonts and is optimized through a combination of gradient descent and backpropagation. sushant097's project utilizes deep learning (a convolutional neural network and CTC decoding) to extract information from a line of handwritten text. Template matching for account/routing extraction compares images of an account or routing number to a provided template image and attempts to find whether they are a match.

7.0.0 Black Box Test Plan[9]

7.1.0 Overview

Our project pipeline is composed of four main stages, those being Front-end, Field Extraction, Data Extraction/Validation, and Data Output. This streamlined design allows us to treat each stage as a module and effectively test each stage individually by creating appropriate test input for each stage and evaluating the output of each stage. This ultimately allows for thorough testing of the entire system. This test plan will outline how we plan to test each stage/module of the program, as well as a more general black box test plan for system testing of the overall system. The black box testing will be performed manually. The white box unit testing tool we will be using for this project is unittest. Our expected overall and individual test coverage for this project is 70%.

7.2.0 Front End Testing & Overall System Testing (Black Box)

Front end system testing will be done via the following black box test plan. Tests will be extensive, covering error cases, boundary values, and a variety of correct/accurate inputs to ensure that the website/GUI is displaying consistent messages and elements at each step of the check image analysis process.

This section of the test plan covers the system testing we will perform on the overall program to ensure it meets requirements and is functional. The program is not necessarily expected to return perfect output (unreasonable to expect the software to read what was written perfectly, due to limitations of open source OCR software), so the following test plan is used as a reference for regression testing, ensuring consistent output (same input image should always return same output) and ensuring that some output is being returned by the software and that it is being correctly validated. Failure of any of the following tests indicates that there has been some change in the internal data extraction process of the system. The test files for the following black box tests will be located in the Appendix (Section 10.0.0) and in the *test/test-files/bbtp_files* directory of the project folder.

Test ID	Description	Expected Results	Actual Results
testValidTextGoodBackground	**Preconditions:** The user has access to the test image files, the web application has been	After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.	Pass

[9] Primary Author: Shreyas Kuloor. Contributing Editors: Dustin Hollar, Taha Karimi, Tianming Liu

<table>
<tr>
<td></td>
<td>loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_good_text_good_bg.png”
3. Click the “Upload” button.</td>
<td>A table containing each check field appears on the page with the following information:

<table>
<tr><th colspan="3">Extracted Data</th></tr>
<tr><th>Field</th><th>Value</th><th>Pass?</th></tr>
<tr><td>Account Number</td><td>000000000</td><td>true</td></tr>
<tr><td>Routing Number</td><td>10000000001</td><td>false</td></tr>
<tr><td>Date</td><td>10/30/2019</td><td>true</td></tr>
<tr><td>Pay to the Order Of</td><td>/ Person</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>300. 00</td><td>false</td></tr>
<tr><td>Written Amount</td><td>Three Hundred and 0/100</td><td>true</td></tr>
<tr><td>Memo</td><td>Test Memo</td><td>true</td></tr>
<tr><td>Signature</td><td>Signals,</td><td>true</td></tr>
</table>
</td>
<td></td>
</tr>
<tr>
<td>testValidTextBadBackground</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_good_text_bad_bg.png”
3. Click the “Upload” button.</td>
<td>After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.

A table containing each check field appears on the page with the following information:

<table>
<tr><th colspan="3">Extracted Data</th></tr>
<tr><th>Field</th><th>Value</th><th>Pass?</th></tr>
<tr><td>Account Number</td><td>1000000000</td><td>true</td></tr>
<tr><td>Routing Number</td><td>0000000001</td><td>false</td></tr>
<tr><td>Date</td><td>5-5-2015</td><td>false</td></tr>
<tr><td>Pay to</td><td>Test Check ,</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td>the Order Of</td><td></td><td></td></tr>
<tr><td>Numeric Amount</td><td>400.00</td><td>true</td></tr>
<tr><td>Written Amount</td><td>Four Hundred and 0/100</td><td>true</td></tr>
<tr><td>Memo</td><td>Test</td><td>true</td></tr>
<tr><td>Signature</td><td>Mn, —</td><td>true</td></tr>
</table>
</td>
<td></td>
</tr>
<tr>
<td>testInvalidAmountNum</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_invalid_num_amt.png”
3. Click the “Upload” button.</td>
<td>After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.

A table containing each check field appears on the page with the following information:

<table>
<tr><th colspan="3">Extracted Data</th></tr>
<tr><th>Field</th><th>Value</th><th>Pass?</th></tr>
<tr><td>Account Number</td><td>10000000001</td><td>true</td></tr>
<tr><td>Routing Number</td><td>10000000001</td><td>false</td></tr>
<tr><td>Date</td><td>1/15/2019</td><td>true</td></tr>
<tr><td>Pay to the Order Of</td><td>Test Account</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>“500-00</td><td>false</td></tr>
<tr><td>Written Amount</td><td>Five Hundred and 6/100</td><td>true</td></tr>
<tr><td>Memo</td><td>Test</td><td>true</td></tr>
<tr><td>Signature</td><td>Magli</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
</table>

<table>
<tr>
<td>testInvalidAmountWritten</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_invalid_written_amt.png”
3. Click the “Upload” button.</td>
<td>After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.

A table containing each check field appears on the page with the following information:

<table>
<tr><th colspan="3">Extracted Data</th></tr>
<tr><th>Field</th><th>Value</th><th>Pass?</th></tr>
<tr><td>Account Number</td><td>00000000</td><td>true</td></tr>
<tr><td>Routing Number</td><td>000000000</td><td>false</td></tr>
<tr><td>Date</td><td>{1/13/2017</td><td>false</td></tr>
<tr><td>Pay to the Order Of</td><td>yest Atcount</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>200.00</td><td>true</td></tr>
<tr><td>Written Amount</td><td>TWo Hundred</td><td>false</td></tr>
<tr><td>Memo</td><td>Test</td><td>true</td></tr>
<tr><td>Signature</td><td>T owe td fae</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
<tr>
<td>testAllCaps</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_all_caps.png”
3. Click the “Upload” button.</td>
<td>After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.

A table containing each check field appears on the page with the following information:

<table>
<tr><th colspan="3">Extracted Data</th></tr>
<tr><th>Field</th><th>Value</th><th>Pass?</th></tr>
<tr><td>Account Number</td><td>000000000</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td>Routing Number</td><td>000000000</td><td>false</td></tr>
<tr><td>Date</td><td>10/20/2014</td><td>false</td></tr>
<tr><td>Pay to the Order Of</td><td>‘ TEST. A@OVNT</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>200.00</td><td>true</td></tr>
<tr><td>Written Amount</td><td>TWO HONNRED AND _ Yiog</td><td>false</td></tr>
<tr><td>Memo</td><td>TEST Avie CARS</td><td>true</td></tr>
<tr><td>Signature</td><td>YD ayt tite...</td><td>true</td></tr>
</table>
</td>
<td></td>
</tr>
<tr>
<td>testMessyText</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_messy.png”
3. Click the “Upload” button.</td>
<td>After a short wait, the image of the check you just uploaded appears on the page with each identified field outlined.

A table containing each check field appears on the page with the following information:

<table>
<tr><td colspan="3">Extracted Data</td></tr>
<tr><td>Field</td><td>Value</td><td>Pass?</td></tr>
<tr><td>Account Number</td><td>1025</td><td>true</td></tr>
<tr><td>Routing Number</td><td>00000000</td><td>false</td></tr>
<tr><td>Date</td><td>Oct. 20, 2016</td><td>false</td></tr>
<tr><td>Pay to the Order Of</td><td>Ye tent . mun</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>BO, 30</td><td>false</td></tr>
<tr><td>Written Amount</td><td>7wo Hundred find. 80 .</td><td>false</td></tr>
<tr><td>Memo</td><td>MossV</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td>Signature</td><td>WEE</td><td>true</td></tr>
</table>
</td>
<td></td>
</tr>
<tr>
<td>testGetJSON</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
1. Click the “Browse” button.
2. Select the check image “test_all_caps.png”
3. Click the “Upload” button.
4. Once the output table appears. click the “Get JSON” button.</td>
<td>A new tab opens containing the data output in JSON format. The data should be identical to the “testGetJSON.json” file.</td>
<td>Pass</td>
</tr>
<tr>
<td>testRevalidate</td>
<td>Preconditions: The user has access to the test image files, the web application has been loaded and the user is on the landing page.

Steps:
5. Click the “Browse” button.
6. Select the check image “test_all_caps.png”
7. Click the “Upload” button.
8. Once the output table appears, correct the “Value” column for each field to the following values:

Extracted Data</td>
<td>The new values are revalidated and the “Pass?” column of the output table is updated accordingly. The output table should now contain the following information:

<table>
<tr><td colspan="3">Extracted Data</td></tr>
<tr><td>Field</td><td>Value</td><td>Pass?</td></tr>
<tr><td>Account Number</td><td>000000000</td><td>true</td></tr>
<tr><td>Routing Number</td><td>000000000</td><td>false</td></tr>
<tr><td>Date</td><td>10/20/2019</td><td>true</td></tr>
<tr><td>Pay to the Order Of</td><td>TEST ACCOUNT</td><td>true</td></tr>
<tr><td>Numeric Amount</td><td>200.00</td><td>true</td></tr>
<tr><td>Written Amount</td><td>TWO HUNDRED AND 0/100</td><td>true</td></tr>
</table>
</td>
<td>Pass</td>
</tr>
</table>

	<table><tr><th>Field</th><th>Value</th><th>Pass?</th></tr><tr><td>Account Number</td><td>000000000</td><td>true</td></tr><tr><td>Routing Number</td><td>000000000</td><td>false</td></tr><tr><td>Date</td><td>10/20/2019</td><td>false</td></tr><tr><td>Pay to the Order Of</td><td>TEST ACCOUNT</td><td>true</td></tr><tr><td>Numeric Amount</td><td>200.00</td><td>true</td></tr><tr><td>Written Amount</td><td>TWO HUNDRED AND 0/100</td><td>false</td></tr><tr><td>Memo</td><td>TEST ALL CAPS</td><td>true</td></tr><tr><td>Signature</td><td>YD ayt tite...</td><td>true</td></tr></table> 9. Click the “Re-validate” button. 10. Click the “Get JSON” button.	<table><tr><td>Memo</td><td>TEST ALL CAPS</td><td>true</td></tr><tr><td>Signature</td><td>YD ayt tite...</td><td>true</td></tr></table> A new tab opens containing the data output in JSON format. The data should be identical to the “testGetJSON_updated.json” file.	

7.3.0 Results

All black box tests in the test plan are currently passing, which is expected. The results of this black box test plan are intended to show that the output of the system is consistent and to be used in regression testing. As seen in the expected and actual results, the data extraction is not necessarily perfect for different styles or formats of handwriting, however, we can test whether data extraction is still returning output and whether that output is being returned consistently. If any of these tests are to fail, it should signal that there has been some significant change in one of the pipeline stages and they should be examined more closely, which makes the black box test plan useful for regression testing.

8.0.0 Unit Tests and Results[10]

8.1.0 Data Extraction and Validation

8.1.1 Overview

This section seeks to provide an overview of the unit test procedures for the data extraction stage and validation of extracted data. The *Data Extraction* stage consists of 8 sub-modules that are dedicated to validating a specific field. The validation routines are tested for routing numbers, account numbers, signature, date, "Pay to the Order of", numeric amount, written amount and the memo.

The validation for each field is unique and contained in its own class, therefore a set of unit tests is needed for each field's validation. The three main approaches to data extraction (Tesseract, Handwritten Extraction, and Routing/Account Extraction) are also unit tested. Unit tests for validation assume that data has already been extracted, and unit tests for data extraction assume that the input is a cropped image of some text or part of the check.

8.1.2 Account Number Validation Module

Account numbers are identifying numbers associated with the account of the check holder. Account numbers do not have rigid rules, and can vary from 4 to 17 digits long. The system will check if the account number is composed of only digits, whether it is of the appropriate length and whether it exists on the check in the first place.

8.1.3 Routing Number Validation Module

Routing numbers are a way of determining where an account was opened from. This field can be used to determine a false check if the number evaluated does not match a known pattern or region code. There are 5 widespread Routing Number patterns, but for now only the US ABA Routing Number is tested.

Valid ABA Routing Numbers must have the following format: XXXX YYYY C. The full list of valid Routing Numbers can be found in the Appendix, Section 10.0.0 of this report.

8.1.4 Signature Module

Signatures are a verification of identity. Due to the scope of this project, the only validation we will be performing on the signature module is whether it exists. Given further time, validation for signatures may include a consistency check (the signature on the check should be similar to the signature on file for that person or their signature on previously signed checks). However, this would require a significant

[10] Primary Authors: Dustin Hollar, Shreyas Kuloor. Contributing Editors: Taha Karimi, Tianming Liu

expansion to the system, such as some way to store the signatures associated with a person. Unit tests will simply verify that the system is correctly validating empty and filled out signatures.

8.1.5 Date of Issue/Submission Module

Valid dates are formatted according to the following standards:

- ISO (2019-09-04)
- Short Date (09/04/2019 OR 09/04/19)
- Long Date (September 9, 2019 OR Sept. 9, 2019)

The system should be able to correctly identify whether the input date matches one of the formats listed above, whether the date is in the past (dates on checks should not be later than the date of upload if the check is valid), and whether the date on the check is too old (checks are typically accepted up to 6 months after the date of signing, but this value will be configurable in our system).

8.1.6 “Pay to the Order of” Module

For the scope of this project, the validation on Pay to the Order Of field is simply a verification of whether the field was filled out. With a larger scope or in practice, validation may include whether the subject of the field is actually real (does the name match the account number). Tests on this module will test whether the system can correctly identify whether this field was either filled out or not filled out.

8.1.7 Numeric Amount Module

The validation for this module is a number check. Amounts should be greater than a defined minimum value and lower than a defined maximum value. These limits are configurable via config file in the system. Amounts that are outside the limits will be rejected. Tests on this section will ensure that the system correctly determines when an amount is either within or outside of the boundaries.

8.1.8 Written Amount Module

This module will verify whether a written amount is written in the correct format and whether it is a valid number. This does not have to match the numeric amount. If the numeric and written amount are different values, it is standard for the written amount to be used for the check amount, so matching is unnecessary. After verifying the validity of the written amount, it is treated as a numeric amount and compared to the configured bounds (see 8.1.7).

8.1.9 Memo Module

Due to the nature of the memo field, there will be no additional validation for this field, as the memo field does not need to be filed out for a check to be valid. Testing for this field will simply involve verifying that the system is correctly reading in the memo text.

8.2.0 Unit Test Results

8.2.1 Overview

Unit Test Tool: unittest
Coverage Tool: pytest-cov

Validation for all check fields has been unit tested, with 70% or higher statement coverage achieved on all field classes. All unit tests for these fields are passing. The three main approaches of data extraction (Tesseract, TensorFlow Handwriting Line Text Recognition, and Routing/Account Template Matching) have been unit tested with 70% or higher statement coverage. Due to the natural limitations of OCR, we are not expecting the system to return the perfect output for every test image. We are more concerned with whether the system is detecting and extracting data from each test image, and whether it is extracting the same data from the same image consistently. As a result, our current tests for data extraction show that the OCR successfully reads in something from each test image when it is supposed to, and it will continue to extract the same text each time.

8.2.2 Exempt Units

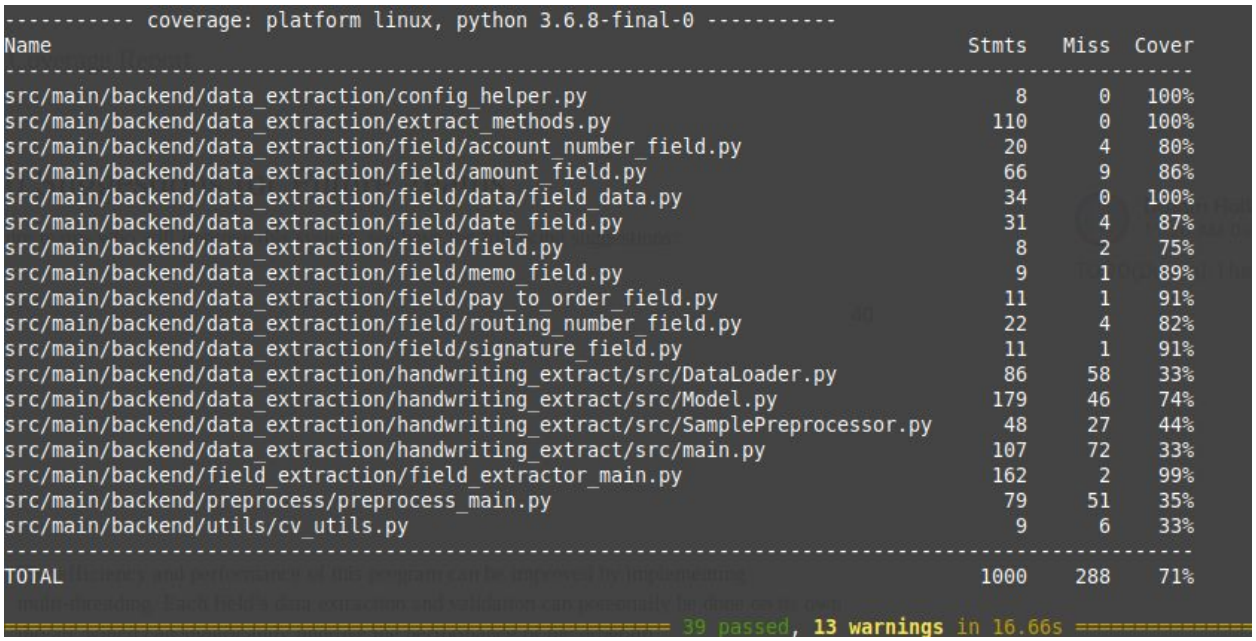
Classes in the *src/main/backend/handwriting_extract* directory were not well covered as these are third-party classes from an open source framework.

Preprocessing and postprocessing files (*src/main/backend/preprocess* and *src/main/backend/postprocess*) were also not covered in unit testing, as there is no way to verify whether an image has been properly preprocessed or post processed without manually looking at the output image of these stages. Instead, these two stages are tested through the black box test plan (if processing is not being performed correctly, the output of black box tests will not match the test plan).

The controller files *src/main/backend/data_extraction/data_extraction_main.py* and *src/main/backend/controller.py* have not been covered in unit testing as they are utility files that drive the flow of the system and do not perform any functions on their own.

The dataclasses *src/main/backend/data_extraction/field/data/field_data.py* and *src/main/backend/data_extraction/field_list.py* have not been covered as they do not contain any functions and are used to hold data and define structures.

8.2.3 Coverage Report



```
---------- coverage: platform linux, python 3.6.8-final-0 ----------
Name                                                                                                Stmts   Miss  Cover
-----------------------------------------------------------------------------------------------------------------------
src/main/backend/data_extraction/config_helper.py                                                       8      0   100%
src/main/backend/data_extraction/extract_methods.py                                                   110      0   100%
src/main/backend/data_extraction/field/account_number_field.py                                         20      4    80%
src/main/backend/data_extraction/field/amount_field.py                                                 66      9    86%
src/main/backend/data_extraction/field/data/field_data.py                                              34      0   100%
src/main/backend/data_extraction/field/date_field.py                                                   31      4    87%
src/main/backend/data_extraction/field/field.py                                                         8      2    75%
src/main/backend/data_extraction/field/memo_field.py                                                    9      1    89%
src/main/backend/data_extraction/field/pay_to_order_field.py                                           11      1    91%
src/main/backend/data_extraction/field/routing_number_field.py                                         22      4    82%
src/main/backend/data_extraction/field/signature_field.py                                              11      1    91%
src/main/backend/data_extraction/handwriting_extract/src/DataLoader.py                                 86     58    33%
src/main/backend/data_extraction/handwriting_extract/src/Model.py                                     179     46    74%
src/main/backend/data_extraction/handwriting_extract/src/SamplePreprocessor.py                         48     27    44%
src/main/backend/data_extraction/handwriting_extract/src/main.py                                      107     72    33%
src/main/backend/field_extraction/field_extractor_main.py                                             162      2    99%
src/main/backend/preprocess/preprocess_main.py                                                         79     51    35%
src/main/backend/utils/cv_utils.py                                                                      9      6    33%
-----------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                                1000    288    71%
========================================================== 39 passed, 13 warnings in 16.66s ==========
```

9.0.0 Suggestions for Future Teams[11]

For future teams who will improve this system, we have the following suggestions:

The efficiency and performance of this program can be improved by implementing multi-threading. Each field's data extraction and validation can potentially be done on its own thread, which can significantly improve the performance of the program.

To achieve more accurate results, a team could look into training their own algorithm or implementing their own algorithm for OCR. There are libraries that can be trained further, and the training set for those libraries may need to be increased and different handwriting style datasets are needed for training. Due to time constraints, we were unfortunately not able to explore approaches that required the actual training and testing of the OCR system. This would be interesting for a future team to look into once the infrastructure of the system has already been created.

Another option for a future team is to try and train the OCR library integrated into our system, Handwritten Line Text Recognition using Deep Learning with Tensorflow. This deep learning algorithm is currently trained on the IAM handwriting dataset, but as it turns out, this is not nearly enough training for the algorithm to reliably read handwriting correctly for styles that don't match the training set. A future team could try and train this with a larger dataset of handwriting.

[11] Primary Authors: Dustin Hollar, Shreyas Kuloor, Tianming Liu. Contributing Editors: Taha Karimi

Due to the fact that the front-end was a low priority aspect of this project, the website is a very simple html, css and javascript page. If needed, a future team could expand this frontend to be more intuitive, nicer to look at, and even add more visualization features (why a field failed validation, toggling the bounding box visualization, etc.).

There is little to no error handling present in the system. For example, if the user passes in a random image, the system will still try and run the full process on the image and treat it as if it was a check, which is unnecessary and inefficient. There are also no errors display through the front-end if an error occurs somewhere in the backend. A future team could add error handling to the system and display error messages to the user when error conditions are met. For example, the system could impose file size limits, ensure that the image is actually a check, and display errors to the user when something goes wrong in the backend.

10.0.0 Team Contact Information

Name	**Email**
Dustin Hollar	dustinrhollar@gmail.com
Taha Karimi	mtkarimi97@gmail.com
Shreyas Kuloor	shreyas.kuloor@gmail.com
Tianming Liu	tianlium@gmail.com

11.0.0 Appendix

11.1.0 Valid Routing Numbers - US ABA Routing Numbers

State	**Routing Number**
South Dakota	91408501
Minnesota Moorhead	91300023

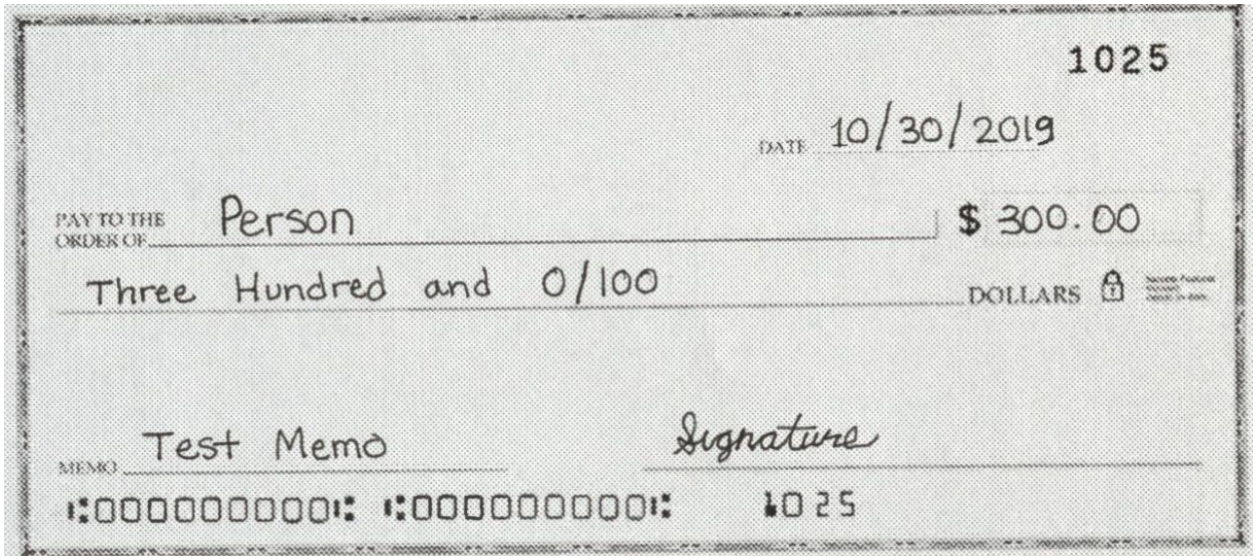
Montana	92900383
Arizona	122105155
Kentucky – Southern	83900363
Colorado	102000021
Wyoming	307070115
Ohio	42000013
Minnesota	91000022
Missouri – Western	101200453
Nevada	121201694
California – Southern	122235821
Oregon	123000220
New Mexico	107002312
Wisconsin	75000022
Iowa	73000545
Illinois - Northern	71904779
Kentucky – Northern	42100175

Kansas	101000187
Illinois - Southern	81202759
Utah	124302150
Arkansas	82000549
California – Northern	121122676
North Dakota	91300023
Washington	125000105
Indiana	74900783
Ohio Cleveland	41202582
Nebraska	104000029
Colorado, Aspen	102101645
Minnesota, East Grand Forks	91215927
Missouri	81000210
Tennessee	64000059
Iowa, Council Bluffs	104000029
Idaho	123103729

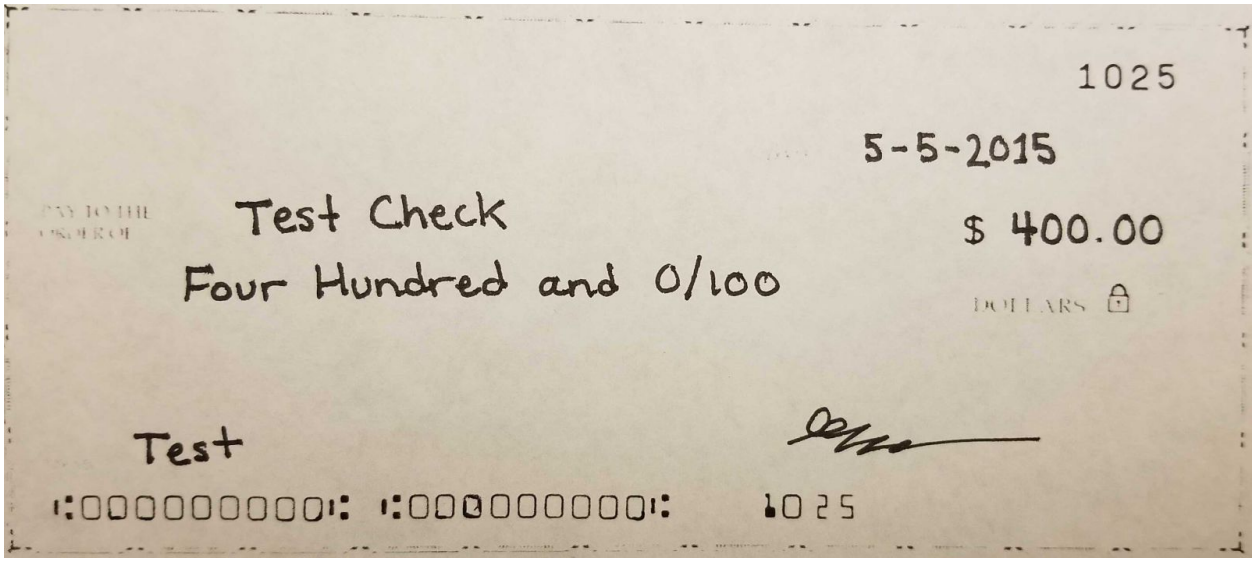
All other states	91000022

11.2.0 Test Files

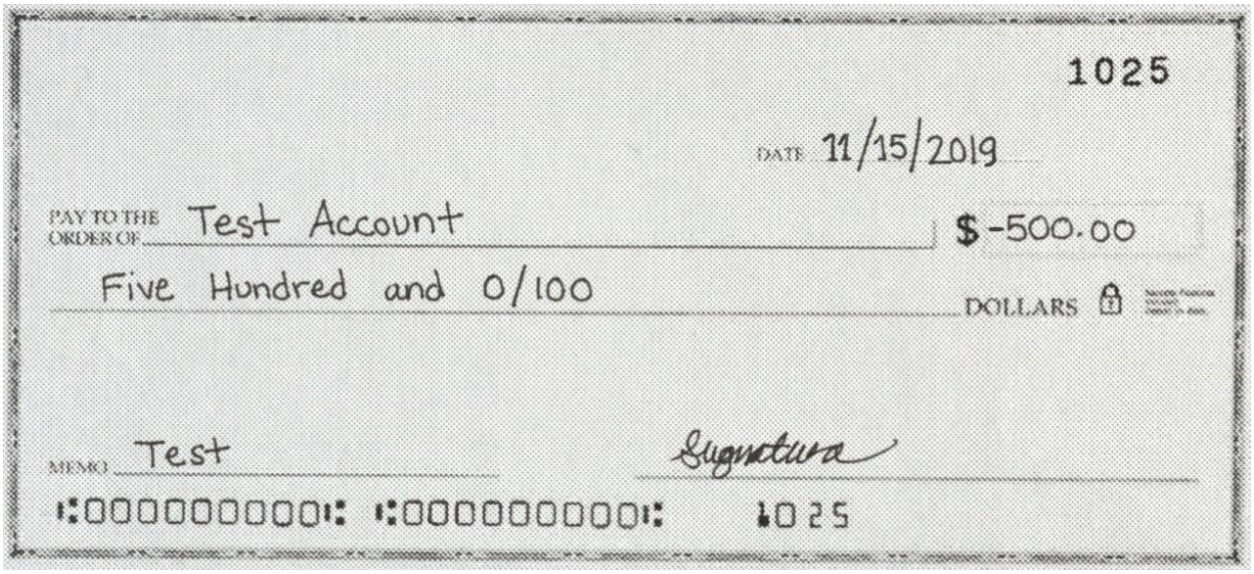
test_good_text_good_bg.png



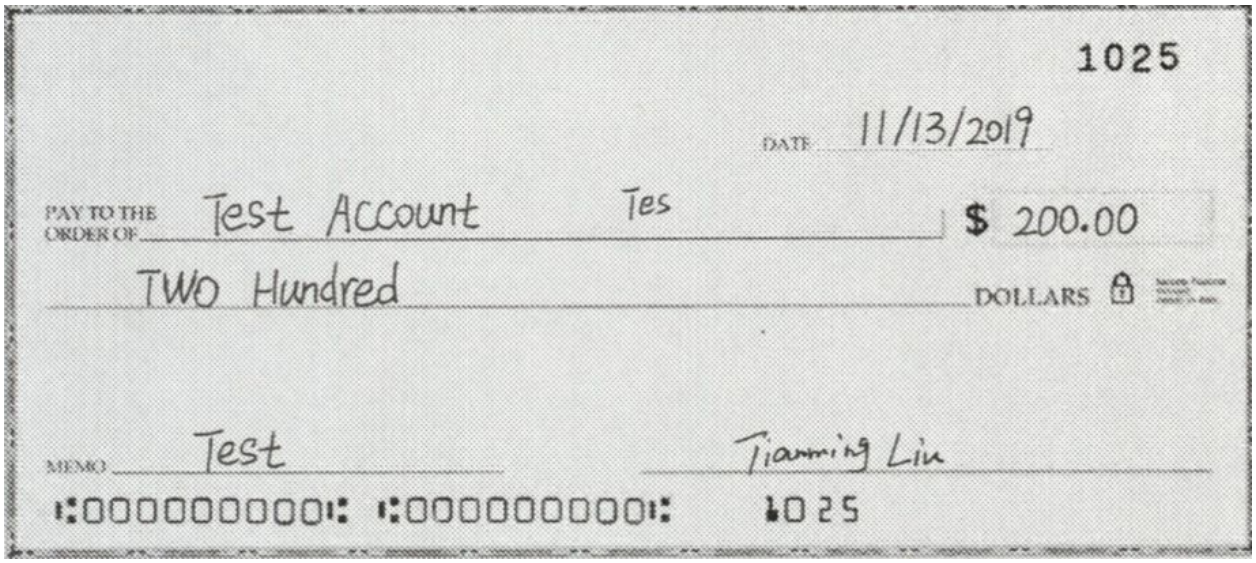
test_good_text_bad_bg.png



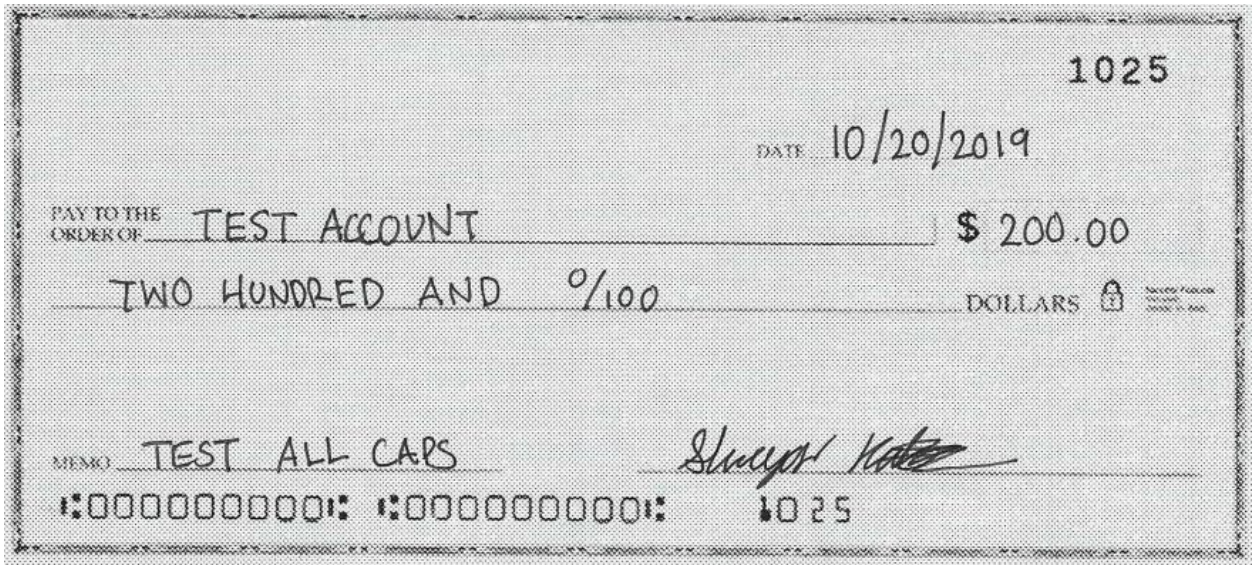
test_invalid_num_amt.png



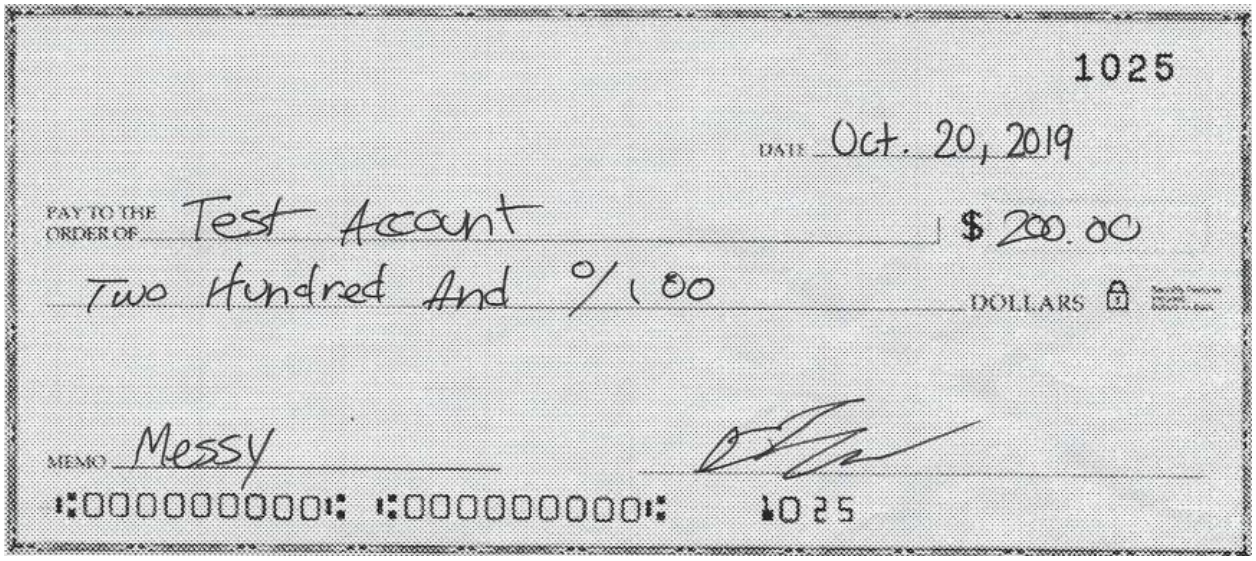
test_invalid_written_amt.png



test_all_caps.png



test_messy.png



testGetJSON.json


```
{
  "element0": {
    "bounds": {
      "h": 46,
      "w": 175,
      "x": 570,
      "y": 70
    },
    "extracted_data": "10/20/2014",
    "field_type": 1,
    "validation": false
  },
  "element1": {
    "bounds": {
      "h": 63,
      "w": 535,
      "x": 105,
      "y": 117
    },
    "extracted_data": "‘ TEST. A@OVNT",
    "field_type": 2,
    "validation": true
  },
  "element2": {
    "bounds": {
      "h": 63,
      "w": 136,
      "x": 697,
      "y": 117
    },
    "extracted_data": "200.00",
    "field_type": 3,
    "validation": true
  },
  "element3": {
    "bounds": {
      "h": 47,
      "w": 614,
      "x": 43,
      "y": 180
    },
    "extracted_data": "TWO HONNRED AND _ Yiog",
    "field_type": 4,
    "validation": false
  },
  "element4": {
    "bounds": {
      "h": 44,
      "w": 316,
      "x": 78,
      "y": 280
    },
    "extracted_data": "TEST Avie CARS",
    "field_type": 6,
    "validation": true
  },
  "element5": {
    "bounds": {
      "h": 41,
      "w": 368,
      "x": 438,
      "y": 280
    },
    "extracted_data": "YD ayt tite...",
    "field_type": 5,
    "validation": true
  },
  "element6": {
    "bounds": {
      "h": 54,
      "w": 877,
      "x": 0,
      "y": 339
    },
    "extracted_data": "000000000",
    "field_type": 7,
    "validation": false
  },
  "element7": {
    "bounds": {
      "h": 54,
      "w": 877,
      "x": 0,
      "y": 339
    },
    "extracted_data": "000000000",
    "field_type": 8,
    "validation": true
  }
}
```

testGetJSON_updated.json


```
{
  "element0": {
    "bounds": {
      "h": 46,
      "w": 175,
      "x": 570,
      "y": 70
    },
    "extracted_data": "10/20/2019",
    "field_type": 1,
    "validation": true
  },
  "element1": {
    "bounds": {
      "h": 63,
      "w": 535,
      "x": 105,
      "y": 117
    },
    "extracted_data": "TEST ACCOUNT",
    "field_type": 2,
    "validation": true
  },
  "element2": {
    "bounds": {
      "h": 63,
      "w": 136,
      "x": 697,
      "y": 117
    },
    "extracted_data": "200.00",
    "field_type": 3,
    "validation": true
  },
  "element3": {
    "bounds": {
      "h": 47,
      "w": 614,
      "x": 43,
      "y": 180
    },
    "extracted_data": "TWO HUNDRED AND 0/100",
    "field_type": 4,
    "validation": true
  },
  "element4": {
    "bounds": {
      "h": 44,
      "w": 316,
      "x": 78,
      "y": 280
    },
    "extracted_data": "TEST ALL CAPS",
    "field_type": 6,
    "validation": true
  },
  "element5": {
    "bounds": {
      "h": 41,
      "w": 368,
      "x": 438,
      "y": 280
    },
    "extracted_data": "YD ayt tite...",
    "field_type": 5,
    "validation": true
  },
  "element6": {
    "bounds": {
      "h": 54,
      "w": 877,
      "x": 0,
      "y": 339
    },
    "extracted_data": "000000000",
    "field_type": 7,
    "validation": false
  },
  "element7": {
    "bounds": {
      "h": 54,
      "w": 877,
      "x": 0,
      "y": 339
    },
    "extracted_data": "000000000",
    "field_type": 8,
    "validation": true
  }
}
```