GenesisCare

COMP3850 - Group 16

User Manual V1.00 - 20/05/21

Version History **2**

README.md **3**

Comp3850GenesisCare - Medicine Box Reader **3**

Installation **3**

Requirements 4

Format 5

Usage 5

Docker 6

Testing 6

Performance **7**

Features 7

Success 7

Drawbacks 7

Future Plans **8**

Short term 8

Medium term 8

Long term 9

Sponsor meeting feedback **10**

Version History

Version Number	Description
1.00 - 21/05/21	Lachlan Preliminary document submitted for D4 including: - Version History - README.md - Intro/title - Installation - Performance - Future plans - Sponsor feedback

As per consultation with the client beforehand, a README in the documentation/repository is both sufficient and preferred as a method of user training/user manual.

Codebase and User manual/README additionally available at:

https://github.com/LachlanMatt/Comp3850GenesisCare

README.md

Comp3850GenesisCare - Medicine Box Reader

This project is designed to produce a table of results to summarise the important information from a given picture of a medication box including:

1. Brand Name
2. Active Ingredient
3. Concentration/Dosage
4. Quantity In Box

The purpose of this is to automate the process of acquiring user medical records to save time for medical staff and increase the operational efficiency of the business

Installation

Clone the repo to any location

```
git clone https://github.com/LachlanMatt/Comp3850GenesisCare
```

Requirements

Need to install

- github desktop
  - https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop
- A Python editor
  - I use Spyder as it's used in Comp3160, though VSCode and other editors should be useable too
  - https://www.spyder-ide.org/
- Cv2
  - pip install opencv-python
- Numpy
  - pip install numpy
- Pytesseract
  - pip install pytesseract
- Google Tesseract at (configurable):
  - https://tesseract-ocr.github.io/tessdoc/Home.html
  - Guide: https://guides.library.illinois.edu/c.php?g=347520&p=4121425
  - C:\Program Files\Tesseract-OCR\tesseract.exe
  - Win64 download: https://github.com/UB-Mannheim/tesseract/wiki
  - (Mac untested)
  - Mac install (using homebrew): brew install tesseract
- Docker Desktop
  - https://docs.docker.com/docker-for-windows/install/
  - Pull the Container: docker pull jakemay95/comp3850genesiscare
  - Repository Link: https://hub.docker.com/r/jakemay95/comp3850genesiscare

Format

Input:

- Allowed file types (configurable) are .jpg, .jpeg, and .png; Other files will be ignored

Output:

- Formatted .csv file
- Top row is headings, describing the field below
- Left column is the name of the file the data came from
- Undetermined data filled with "-"

Testcases:

- Same format as Output

Usage

Steps to use:

1. Place target images in the input folder (configurable):
   - ./input/
2. (Optional) Enable Debugging=True in ProcessingElements.py to have intermediary data saved to (configurable):
   - ./output/Result_date_time/
3. Run RunDetection.py
4. Output will be Reult_date_time.csv in the output folder (configurable):
   - ./output/

Docker

The application has been containerised utilising Docker. This requires the person in charge of deploying the application to only have Docker installed to download and deploy the latest version of the application. From here, the container image can be downloaded to a local directory and run using the command line using the following command; `docker run -it <whatever-it-was-named-earlier>`.

To build using docker from scratch: The source code must be downloaded to a local directory. From here, the command `docker build -t <whatever-you-want-to-call-it> .` will build the container in the current directory (. being the keyword for current directory in bash). This process can take a while depending on your internet connection.

After the container has been built, it is run using the command above.

Testing

Steps to test:

1. Put testcase images and expected output in their relevant folders (configurable):
   - Input images: ./input/
   - Expected output: ./testcases/
2. Run PerformanceTest.py

Performance

For Version 0.5

Features

Current:

- Can detect the important text from simple images
- Utilise Docker to allow for easy deployment Planned:
- Can use a document scanner function to detect text in difficult images

Success

Currently performs at 26-47% success rate on results 2-4 (excluding brand name) for a given testset of reasonable images (correct orientation, angle, resolution).

Drawbacks

- Can not yet handle images with multiple boxes.
- Can not handle rotated, flipped, or other odd orientations.
- Performs poorly on low resolution images.
- Only tested on Windows, Docker implementation pending.

Future Plans

Short term

The short term plans/goals of the projects are:

- Implement a document scanner to increase the range of images the system can successfully process
    - OnlineDocScanner.py is a (low-efficacy) implementation of this
- Implement Docker integration to allow for easier deployment and development of the API (done)
    - https://www.docker.com/
    - https://hub.docker.com/search?q=&type=edition&offering=community
- Expand medical dictionary for brand and chemical names through the use of the PBS
    - https://www.pbs.gov.au/browse/medicine-listing
- Implement different regex systems for liquid/cream and solid/pill/capsule medicines
    - Quantities have different formatting, such as 30 tablets vs 300mL
- Add a config file to unify the re-used settings/variables such as debugging and file paths
    - ProcessingElements, RunDetector, PerformanceTest all have similar/identical variables at the start

Medium term

Plans that we would like to do but will likely fall outside the scope time fo the project

- Communications protocols to allow for easier integration of the API
- AI upscaling and downscaling of images to increase performance
    - Simple resize() commands are currently used in ProcessingElements.py

Long term

Plans that are well outside the scope of the project but could form the basis of a future revision

- Rewriting the core image functionality of the project to be more modular and accommodating of different text recognition models
- Rewriting the project in C/C++ to increase speed and match the language of many text recognition packages like google OCR

Sponsor meeting feedback

Meeting date and time:

Tuesday 18th May 2021

Feedback received:

“Looks good and is sufficient for my needs, but would be good to add a section about future improvements (if you had say 6 months to work on it, where would you take it). If you do end up adding the API we'll also need connection details etc.”

Team response/action points

“Sweet, I'll add some sections for that”

The team was happy with the feedback and with the user manual that has already been constructed. We will consider the point raise on future development of the system, where we could/would take the project to increase its performance if time was made available.