GenesisCare

COMP3850 - Group 16

Design Document V1.01 - 20/05/21

Version History

Version Number	Description
1.00 - 29/4/21	Lachlan Preliminary document submitted for D3 including: - Version History - Definitions - Feature Engineering - Solution Architecture - Algorithm/Models/Methods - Detailed Data Descriptions - Scripts/Model Execution (Heading only, not completed) - Assumptions - Prototyping and Feedback - Appendix A-E
1.01 - 20/05/21	Lachlan Updated for D4 - Updated temporal wording in sections 1-4 - Added section 5 as required - Added location note to Assumptions

Definitions, Abbreviations

ID	Phrase	Description
1	API	Application Programming Interface
2	GenesisCare	The client of the project
3	Team 16	The team that is developing the project
4	Python	Python programming language
5	GT, Google Tesseract	Google's open source OCR
6	RGB and Greyscale	Colour (Red Green Blue), and Black and White
7	CV2	OpenCV project
8	OCR	Optical Character Recognition
9	OpenCV	Open source computer vision library
10	MVP	Minimum Viable Product to fulfill basic requirements
11	FP	Final product including all requirements
12	NN	Neural network
13	CNN	Convolutional neural network
14	GUI	Graphical User Interface

1. Feature engineering (Selection, Construction)

The current system of data-processing employed at GenesisCare utilises user submitted photos of their medication boxes, and then a human manually will enter these details into the system. The goal of this project is to automate this end to end, that is take in the pictures that a user has submitted, and have our API process them into a neatly formatted text document that GenesisCare can automatically input into their records. The goals that therefore need to be achieved are:

- Detecting text present in a photo
- Detecting the specific characters/words in this text
- Detecting which of these words are important information (medicine brand, dosage, etc)
- Formatting this into usable text

This broad overview forms the basis of the MVP, detecting, analysing, and processing text within pictures (test cases provided by GenesisCare). However, as users can be unpredictable, it is likely that photos of a low quality will be produced, whether in resolution, lighting, angle, occlusion or otherwise. These challenges need to be accounted for by any model that is developed. Some different methods such as document scanners, image upscaling, and colour, hue, saturation, and brightness manipulation will require additional features within the images to be interpreted by the API to then act as an image pre-processing stage for any system that is developed. Some of these will include:

- Medicine box bounding contour edges
- Medicine box rotation angles, skew angles
- Letter edges
- Underlying detail in blurry and low resolution images

Currently it is planned for the FP that a Document scanner style operation will be performed to detect and apply relevant matrix transformations to normalise the text on the box. This along with colour and edge detection are to form an image preprocessing structure. Lossless denoising and image upscaling are extended plans for FP as necessary.

Once the images have been processed into raw text, this will then need to be processed by finding meaning in the disorganised text produced by the image to text processed, with different processes likely being needed for different pieces of information. Some text features will include:

- Chemical/Medical names
- Titles/Brand names
- Medicine dosage
- Medicine box size

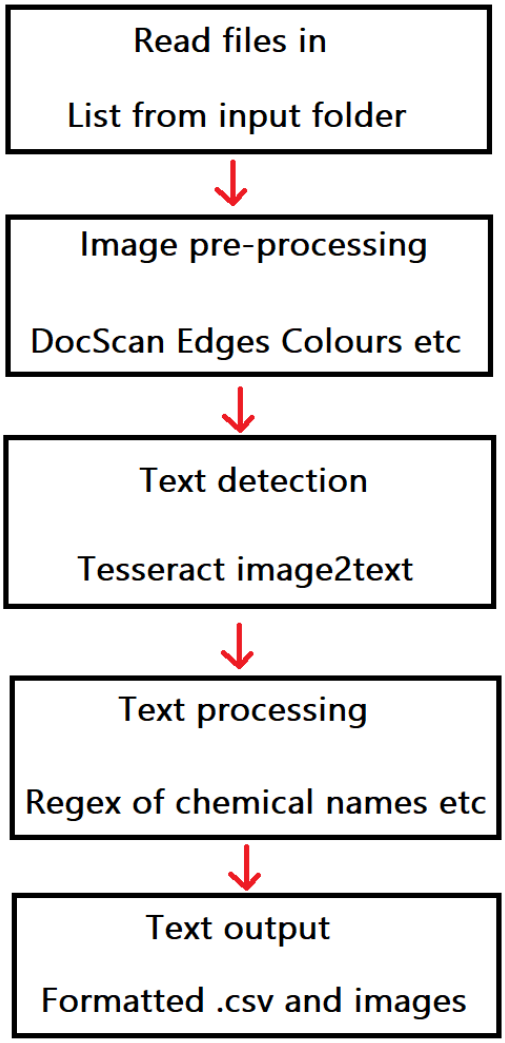
ID	Feature	Description	Stage
1	Detecting text	To be able to read text off a photo, the location of the text must first be determined within the image. This is generally related to convolutional neural networks, where an image feature is detected by convolving the image with a pre-trained mask/filter. Once the locations of various pieces of text within an image are found, they can be classified by character/word Currently using Google Tesseract Expected values to be integers in the range [0, width] and [0,height], where width and height are the dimensions of the image being processed	MVP
2	Reading text	Once text locations within an image are detected, the exact text can be understood. This is typically done by using a hidden layer neural network to provide a probability confidence interval of different characters, and return the most likely character. Currently using Google Tesseract Expected values are to be Characters or Strings of Characters (implementation dependant) representing the text believe to be at the location provided	MVP
3	Analysing text	Broadly, once text has been found, the important piece of this needs to be determined. Depending on the piece of information being found, different techniques may need to be used (discussed later) such as regex, dictionaries, lookup tables or otherwise. Currently using Regex Expected values will be Characters or strings of characters, but of specific topics such as Names, critical numbers, or instructions (implementation dependant)	FP
4	Formatted Text	Any text determined to be important through previous modules will need to be formatted such that the system interacting with the API can interpret the features discovered inside the images. This could be done externally through word processors, or internally with array and String manipulation depending on implementation Currently using array manipulation (first valid result is chosen) Expected values will be csv or txt files with structured ordered lists of results	FP

5	Detecting box edges	To re-align the front-face of the medicine box with the camera's perspective, the edges of the box's face will need to be identified and measured to determine how to align them. While determining edges in an image is simple with a kernel filter, determining which 4 straight line edges constitute a box, and the start and end position of these is challenging. This could be done with a machine learning CNN though this would require training sets of thousands of photos, so instead a mathematical approach can be taken through approximations of polygons Plan to use CV2 built in functions Expected outputs are the edge lines of the box, measured as a list of ordered line endpoints (corner coordinates)	FP
6	Detecting box angle	To be able to unwarp a skew-angled box, the angle of the box with reference to the camera needs to be directly or indirectly calculated to determine the linear transform that will undo this skew Plan to use cv2 built in functions Expected values will depend on implementation, though direct angle numbers, or transformation matrices could be possible values for further computation	FP
7	Detecting Letter edges	To increase the effectiveness of image processing, image pre-processing can be used. Detecting the edges of letters allows OCR processes to focus on the shape rather than texture and colour of letters, which could increase accuracy. This can be achieved with single edge detection kernels Using CV2 built in functions Expected values would be a new image displaying the pixels that contain edges, which can then be used as a replacement image	MVP
8	Finding details in low-quality images	ML image upscaling is an emerging field, which attempts to find details inside low quality photos by learning the "essence" of photos to learn how to increase the detail of other images. This is done by investigating underlying image structure, though is outside the scope of this project, so only pretrained models would be implemented Not planned to implement at time of writing (V1.01) Expected values would be an image similar to the original, but more detailed	FP, if at all

9	Detecting medical name	Text processing has many alternatives, however as chemical names follow certain naming conventions, regex or other can be used to find the naming structures in words to determine if they are a chemical (medicine) name or not. E.g benzene, iodine, sulphate, though this will be imperfect as medicine has a similar ending Currently using regex and an external list of suffixes Expected values will include Strings that are names of medicines/chemicals	FP
10	Detecting brand name	Text processing has many alternatives, however detecting unknown proper nouns (brand names) is challenging, though could be found by checking for words that do not exist in a standard dictionary No approach currently finalised Expected values will be Strings that are medical brand names	FP
11	Detecting medicine dosage	Text processing has many alternatives, however as dosages follow certain formatting, regex or other can be used to find the naming structures in unit measurements to determine if they are dosage values or not. E.g 5mg, searching for number of units (mg is uncommon in English) Currently using regex to detect number and keywords Expected values will be a number or string for the size of the dose in relevant units	FP
12	Detecting medicine quantity	Text processing has many alternatives, however as dosages follow certain formatting, regex or other can be used to find the naming structures in unit measurements to determine if they are pill number values or not. E.g 30 tablets, searching for keywords like tablets or capsules Currently using regex to detect number and keywords Expected values will be a number or string for the size of the box in relevant units	FP

2. Solution Architecture

Following through the features discussed in part 1, the three basic ideas of Read Analyse Process for the images into text leads to some logical modules to accomplish this aim. When testing this initial idea though, these alone are not enough to have good performance, so an additional model of image preprocessing is needed at the beginning of the pipeline. This will allow the incoming images to be cleaned and normalised so that later stages of text detection will have a higher chance of being successful, and in turn increasing the overall performance of the developed system.



Visual representation of planned system

ID	Module	Description	Stage
1	Read files in	As part of configuration settings, a folder is designated as input to the API. Default: ./input/ This module uses Python's OS functions to get the list of files in the input folder, and then import the listed files into the API using CV2, which formats them as 2D or 3D arrays (greyscale vs colour depending). The list of arrays is then passed to the Image pre-processing module. This supports file types that can be processed by CV2 (jpeg and png tested). For FP an additional check is being performed to ignore files of an invalid type in the input folder (.docx).	MVP, FP
2	Image pre processing	The list of image arrays from Mod-1 is taken as input, and will be looped over in order for each image in the list. Each image will have a set of transformations applied to it which will produce multiple processed images per raw image, in an attempt to make the following Mod-3 more accurate when applied to the group of processed image arrays instead of just the raw image alone. Current transformations (from CV2) that are included in the preprocessing stage are Colour2Greyscale, EdgeDetection, and ColourInvertion. This list of Processed images is then sent to Mod-3 for each of the original images received from Mod-1. For FP, additional pre-processing transformations such as a DocumentScanner and AiUpscaler and planned and considered to be implemented respectively, and possibly other pre-processing transformations as testing and optimisation continues. These could be done using CV2 functions or open source pre-trained Neural Network models.	MVP, FP
3	Text detection	The lists of processed image arrays from Mod-2 are processed using Google Tesseract, with each individual image having the tesseract text generation functions applied to them. This generates raw text, which is turned into a list of strings for each processed image. These processed image lists are then combined for each raw image and sent to Mod-4 as a 2D array of strings. The raw text is prone to errors (Appendix B) so will need to be cleaned in Mod-4. For FP there are no planned improvements to this module.	MVP
4	Text processing	For FP, the 2D arrays of raw text strings from Mod-3 are received, and have different text processing steps applied to them to find specific details, such as medicine	FP

		name, active compound/chemical, number of tablets etc. Tools such as regex for detecting chemical name formats, and searching for units (mg, mL), numbers, and keywords (tablets, capsules) are being used to achieve this. This module builds the formatted csv as it searches for information. When it finds (or doesn't find) the information for a specific property from an image, it inserts this into a neatly formatted 2D processed text array. This array includes a header row with titles and labels so that it is both computer and human readable. All images are stored in the 1 table as it is a summary of all the medicines found for a single user. This summary array is then sent to Mod-5.	
5	Text output	The 2D Array from Mod-4 is saved to a .csv file using CV2 save text functions. For FP, it is possible to also save the processed images, raw text, and processed text produced in Mod-2,3,4 for debugging or other purposes. This is not strictly required for general running of the API, and would also require significant storage space over time.	MVP

Resource Requirements and restrictions:

Currently there are no reasonable technical limits to performance or resource use as the system is not real-time (no <16.6ms for a 60Hz video for example, only 1 or a few still images need to be processed). However, there are practical limitations on resources as to not be wasteful of GenesisCare's server resources, and to allow quick turnaround of results if needed urgently (a 1 hour processing time per image would be an example that fails both). Additionally, longer processing times make it more difficult to develop for the system as engineers must wait long periods of time while tuning the system. Due to these reasons, it is optimal to reduce processing times where it does not impact the performance of the system, or a tradeoff between performance and speed can be justified to the client.

3. Algorithm/Models/Methods

Currently data processing algorithms and models are used in Mod-2,3,4 as shown in section 2. Each model has different types of algorithms and models, for processing images, detecting text, and processing text respectively.

ID	Name	Description	Stage
Mod-2_1	Ai upscaler	Not implemented (might not be necessary) Using a pre-trained open source CNN model or an algorithm to increase the resolution of the images, and remove blur. Will depend on what is available and how effective it is at increasing accuracy.	FP (if at all)
Mod-2_2	Document scanner	Not yet implemented. A pre-trained document scanner algorithm is planned to be implemented using inbuilt CV2 functions. This uses kernels, estimating geometry, and linear transformations to find edges, estimate valid shapes, find their corners, and calculate the linear transformation to apply that will create an orthographic view of the image instead of a skew view. This will be capable of fixing rotation, off angle, flipped or otherwise non-aligned images	FP
Mod-2_3	Colour transforms	Converting images to greyscale or inverting colours using inbuilt CV2 functions.	MVP
Mod-2_4	Edge detection	Detecting the edges of text using inbuilt CV2 functions to remove colour, texture, and noise from the image.	MVP
Mod-3_1	Ocr Google Tesseract	Google tesseract is accessed through py-tesseract (a compatibility layer as GT is written in C++). This is a pre-trained CNN that detects text within an image and returns what it says and its location.	MVP
Mod-4_1	Regex text processing	1.1 Regex is being used to detect the structure of chemical compounds. Chemical compounds use standard naming conventions (such as sugars ending in -ose), so regex is being used to detect these word endings, or other common structures that appear in chemicals more often than standard English. 1.2 Regex is also being used to detect numbers as these will likely be technical information. 1.3 Regex is also being used to detect keywords such as tablet or mg (milligram). It is possible to perform 2 and 3 without the use of regex, however regex makes this simpler and easier.	FP

Mod-4_2	Dictionary	Partially implemented (extension might not be necessary) For Chemical naming this is currently being used to build a repository of chemical name components that are searched through with regex. An extension is being considered using a (medical or otherwise) dictionary to check if words are proper nouns, and checking grammatical errors to help process text (such as brand name), though it's not clear how this could be effectively used. The Pharmaceutical Benefits Scheme has public databases but it is not clear if these can be accessed in full, in such a way to be beneficial to the project	FP

Previously the model components have been tested by visually/manually inspecting the results of individual steps to determine if they are effective.

Currently for FP, the whole model is being tested by using test cases (explained in the testing document, test cases manually created) to automatically run the whole project and determine the accuracy rate over a range of input files by comparing its generated output to the expected value. To test the effect of individual models within each module, debugging abilities to disable certain sections and models will be implemented so that a side by side comparison can be done showing the impact of each model, and if the cost penalty is worth it.

4. Detailed Data Descriptions

Currently images are imported into the API, are processed into new images, which are then processed into text, which is then formatted and saved. It is possible (for debugging reasons) that the intermediary stages of making new images and text could also be saved, but is not planned for standard operation.

ID	Name	Description
1	Data imported	1.1 Images that are jpeg and png are imported into the program to be processed. Once imported they are stored as a set of arrays corresponding to their pixels (grid position and colour channel). Colour images are 3D arrays (x, y, c) and greyscale images are 2D arrays (x, y)
2	Data created	2.1 The images in 1.1 are processed into more images which should be more easily processed in Mod-3. These images are not saved during standard operation, but could be saved for debugging purposes. Like 1.1, these are 3D or 2D arrays depending on if they are colour or greyscale 2.2 Text is read from the photos in 2.1, and is then turned into a 1D array for each photo in 2.1, and then a 2D array for each photo in 1.1 (combined list of the 1D arrays)
3	Data stored	3.1 Formatted text is generated by processing the text from 2.2. This is used as a 2D array (image, property) With column and row headers (property name, image name) to hold the important information extracted from the images. It is then stored as a .csv file so it can be opened in and interface easily with other programs of GenesisCare's future decision 3.2 It is possible for the 2.x data to be stored for debugging purposes, however this is not done during standard operation as it costs computing resources and storage space

5. Scripts/Model Execution

The project is available on github at:

https://github.com/LachlanMatt/Comp3850GenesisCare

For brevity, convenience, and being up to date, full project code will be provided in Appendix D, additionally a structure and explanation will be provided. The repository contains a README.md (UserManual) for additional reference, as well as all other files relating to the project API. There are a is a few key files within the repository for different purposes:

The model - ProcessingElements.py

This is the functions that run the program; loading, processing, detecting, formatting, and saving the information in the API. This file does not execute when run to allow for file segmentation and importation. It has the following structure, as explained in detail in earlier sections of this report, and full presented in Appendix D:

```
33  # structure:
34  # main
35  #   genRegex
36  #   importPics
37  #   processImages
38  #       detectText
39  #           labelImage
40  #       processText
41  #           bestAnswer
```

The Run file - RunDetector.py

Imports the functions from ProcessingElements.py and runs the main function (runProgram()), to allow for other run or test files to exist that can treat these function differently (allowing for future expansion)

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Tue May 18 12:00:07 2021
4
5   @author: Lachlan
6   """
7
8   ####################################
9
10  import ProcessingElements as PE
11
12  ####################################
13
14  #run the main program
15  PE.runProgram()
```

The Test file - PerformanceTest.py

Imports relevant dependencies (e.g numpy) and the ProcessingElements.py functions to be able to run the main program similarly to how RunDetetor does, but with the additional ability to test the program at the same time. The Test file works by importing the most recent output of the program run on a test set, and importing the known test set output and comparing the 2 outputs. This is done through field-wise comparisons with regex to see if either field is a subset of the other (to account for formatting differences that are otherwise similar). Each time a match is found, a counter is incremented, and this is returned as a percentage to the tester at the end.

```
1   ##################################
2
3   import numpy as np
4   import os
5   import time
6   import re
7   import ProcessingElements as PE
8
9   ##################################
10
11  timestr = time.strftime("%Y-%m-%d_%H-%M-%S")
12  testPath='./testcases/'
13  outputPath='./output/'
14  outputType='.csv'           # .txt or .csv supported
15  debuggingPath=outputPath+'Result_'+timestr+"/"
16
17  ##################################
18
19  start = time.time()
20
21  #retrieves the name of the newest .csv in the output folder
22  def importResult():
23      arr = os.listdir(outputPath)
24      # print(arr)
25      fileList = []
26      for a in arr:
27          if (re.match(".+("+outputType+")$",a)):
28              fileList.append(a)
29      return fileList[len(fileList)-1]
30
31  #retrieves the list of test cases in the testcase folder
32  #currently only returns the first file it finds
33  def importTestCase():
34      arr = os.listdir(testPath)
35      # print(arr)
36      fileList = []
37      for a in arr:
38          if (re.match("(TestCase).+("+outputType+")$",a)):
39              fileList.append(a)
40      return fileList[0]
41
```

```
    #loop over the output result and the testcase arrays
    #element by element compare them using regex (if either is a subset of the other)
    #use regex to remove single brackets and other grammar that can break regex statements
    #return the percentage of times a match was found
    def calculateAccuracy(Result, Testcase):
        count = 0
        total = 0
        print("i, j, Result, r, t:  i j Result Processed Testcase")
        for i in range(len(Result)):
            #ignore row/col 0 as they are headers
            if (i==0):
                continue
            for j in range(len(Result[i])):
                if (j==0):
                    continue
                #have only 0 brackets or a bracket pair
                process = re.search(r"(\w+|/|\-| )*(\((\w+|/|\-| )*\))*(\w+|/|\-| )*", Result[i][j])
                r = process[0]
                t = Testcase[i][j]
                #remove spaces and other annoyign characters
                R = re.sub("[^a-zA-Z0-9\-\/\(\)]","", r)
                T = re.sub("[^a-zA-Z0-9\-\/\(\)]","", t)
                print("i, j, Result, r, t: ", i, j, Result[i][j], R, T)
                total = total + 1
                if (re.match(".*"+R+".*",T) or re.match(".*"+T+".*",R)):
                    count = count + 1
                    print("count, total: ", count, total)
        if (count==0):
            return 0
        return round(count/total*100, 2)

    #run the main program
    PE.runProgram()

    #import the data and calculate the accuracy
    ResultAdd = importResult()
    Result =     np.loadtxt(outputPath+ResultAdd, dtype = 'str', delimiter = ",")
    TestcaseAdd = importTestCase()
    Testcase =   np.loadtxt(testPath+TestcaseAdd, dtype = 'str', delimiter = ",")
    accuracy = calculateAccuracy(Result, Testcase)
    print("accuracy: "+str(accuracy)+"%")

    #print the time the system took to run
    end = time.time()
    print ("Testtime: "+ str(round(end-start)) + " seconds")
```

6. Assumptions

Assumptions are split between Project Plan, Design Document, and Testing Document.

We are making an API for the client which they will be able to utilise, reducing the need for human involvement of non-critical medical activities by having a computer recognise medication lists instead of through time intensive human discussion/recording.

We assume that this project will require collaboration with GenesisCare to verify that our solution meets the criteria set out at various stages of completion, through the use of weekly meetings and reviews of work. The project produced must fulfill their needs to make it a viable business decision for them to invest in.

It is assumed that

- Pictures will be provided that are of a correct file format
- Pictures provided will be of a minimum reasonable standard to be readable by a human
- The python modules used (listed in README) will be installed on the server GenesisCare runs the program on, or through a docker container
- Google tesseract will be installed on the server GenesisCare runs the program on, or through a docker container
- Complex information like full sentences will not be required
- Enough storage and processing power will be available on to server to allow for the output to be calculated and storage in a timely manner
- The program developed and documentation will be delivered in full to GenesisCare
- An MVP will be completely by the required date, and built off over the coming weeks, a draft FP will be available before final submission so that it can be optimised
- Designing new NN and CNN's will not be done as that would require large training sets of labelled photos which is outside the scope of the project
- Open source and commercially allowed libraries need to be used if external libraries are used
- Users' privacy needs to be maintained as these are medical records
- Users are capable of taking and uploading images to the web portal on a device

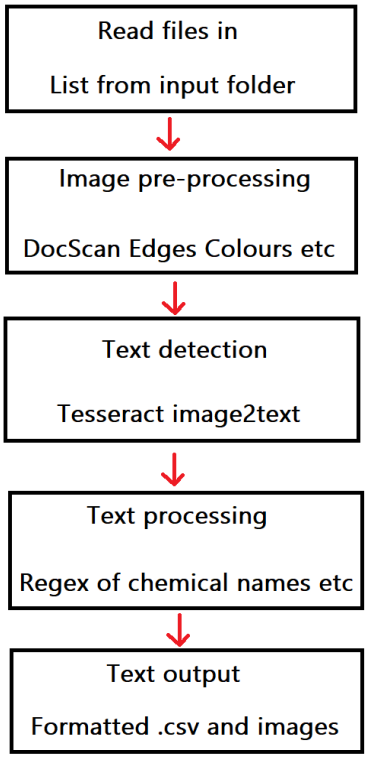
7. Prototyping and Feedback

Prototype

The project is an API, so does not have a GUI. Currently the intermediary photos that are generated are presented to the user/developer (Appendix C), and the Raw generated text is saved to a .csv file (Appendix B). There are configuration options to change where the project reads from (input) and outputs to, and the output file type (both .csv and .txt are supported) (Appendix E).

```
inputPath = "./input/"
outputPath='./output/'
outputType='.csv' # .txt or .csv supported
```

Currently the program is in an MVP state, as it can detect text and save it for the user, which is the most important quality required. Going forward, the missing functionality of the regex text processing and document scanning (section 3) will need to be implemented to finalise the project. These will need to be done well in advance of the due date as any solution will require heavy optimisation due to the image pre-processing. Different preprocessing models will need to be searched, tried, tested and compared to achieve the most accurate results for GenesisCare. As shown in section 1, the general form of the API follows the following structure:



The full code (Appendix D) currently does not implement the Text processing module as shown. It currently produces photos, presents them to the user, detects the text from those, and saves text.

Sponsor meeting feedback

Meeting date and time:

Tuesday 27th April 2021

Feedback received:

"Group 16 demonstrated their prototype to myself over our weekly Zoom session (27/04).

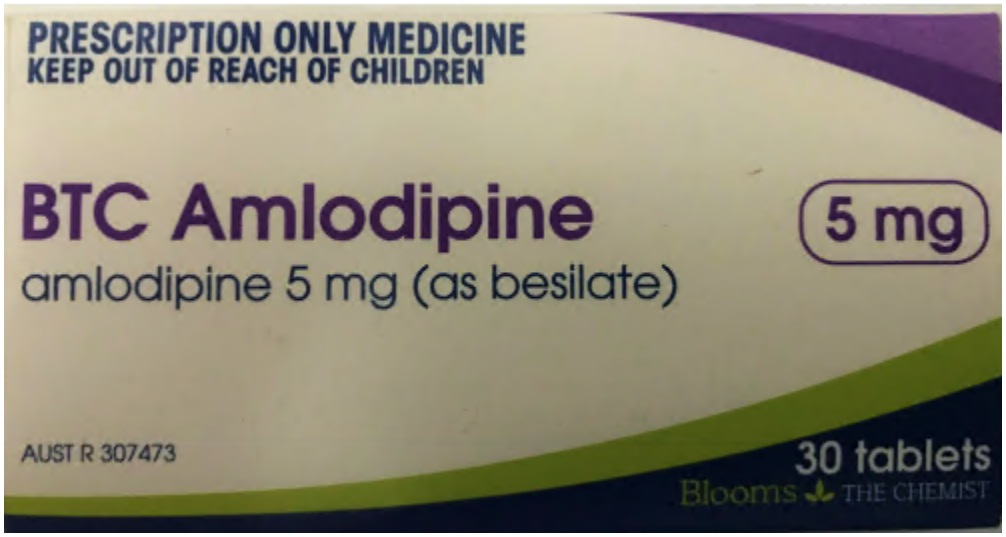
The prototype is fairly basic in nature at this stage, however the direction is correct and align with our end goal in mind. Currently the application is able to read, manipulate and process medication boxes in images (provided) - as well as text detection from within the images. We are looking forward to seeing the progress made in the coming weeks."

Team response/action points

"Sweet, I've made some more progress since yesterday, have implemented input and output folders, output csv files, put our files onto github so we can better share them, and started testing a 3rd party document scanner on some test images but not yet implemented one into the main program. Still need to do regex and docker, then optimisation steps can happen"

Additionally, we will continue to make git commits to keep a tracking history of the program being developed. Separation of the main file (Appendix D) should be done to create configs and other file dependencies to allow for parallelization of development, and future ease of modification. A GitHub link was also sent to GenesisCare

Appendix A - Simple Test Case Image

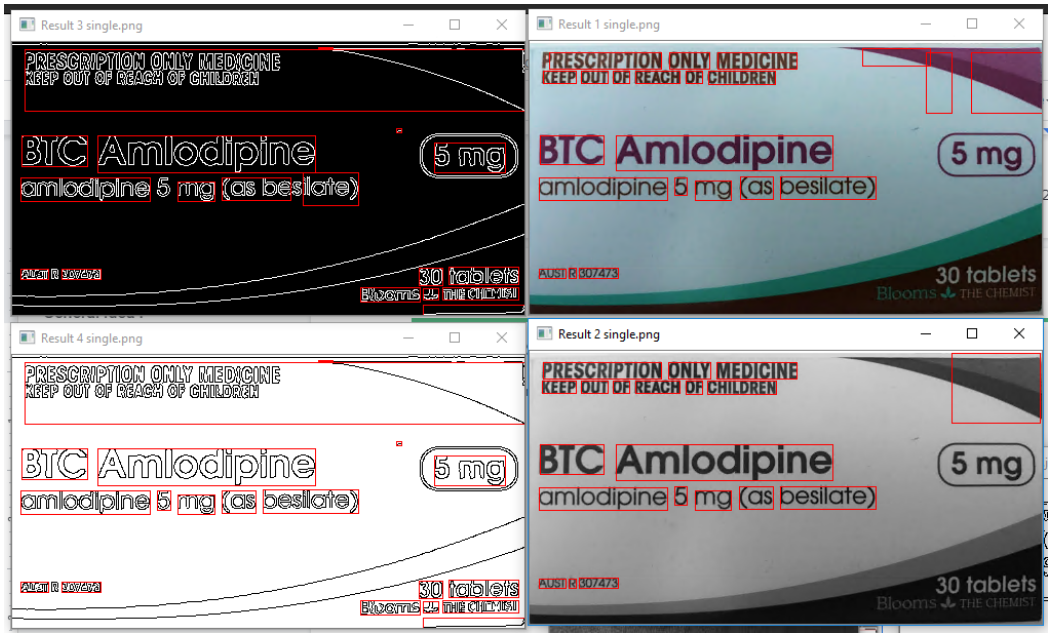


Appendix B - Simple Test Case Raw Text Output

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	['='	'ae'	'BIC'	'Amlodipi	'—]'	'me'	'amlodipi	'5'	'mg'	'(as'	'besilate)	'ANE'	'ORE'	'30'	'tailets'	'Blvems'	'2'	'TECHIE!'	'eg']			
2	['='	'ae'	'BIC'	'Amlodipi	'—'	'Gime'	'olileliel	'Mine'	'nCepec'	'sole)'	'ANE'	'OTE'	'30'	'tablets'	'Bivens'	'2'	'TECHIES!	'ee	']			
3	['PRESCRI	'ONLY'	'MEDICIN	'a'	'KEEP'	'OUT'	'OF'	'REACH'	'OF'	'CHILDRE	'_'	'^<_'	'BIC'	'Amlodipi	'amlodipi	'5'	'mg'	'(as'	'besilate)	'AUST'	'R'	'307473']
4	['PRESCRI	'ONLY'	'MEDICIN	'='	'KEEP'	'OUT'	'OF'	'REACH'	'OF'	'CHILDRE	'BIC'	'Amlodipi	'amlodipi	'5'	'mg'	'(as'	'besilate)	'AUST'	'R'	'307473']		

Appendix C - Simple Test Case Labelled Produced Images

The project does not have a GUI as it is an API, these are debugging results



Appendix D - Current Program

ProcessingElements.py (program functions). Run and test code in Section 5

Also provided in full at:

https://github.com/LachlanMatt/Comp3850GenesisCare

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 22 23:20:56 2021

@author: Lachlan Matthews
"""

###################################

import cv2
import numpy as np
import pytesseract
import os
import time
import re


###################################

pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
timestr = time.strftime("%Y-%m-%d_%H-%M-%S")
inputPath = "./input/"
inputType = ["png", "jpg", "jpeg"]
testPath='./testcases/'
outputPath='./output/'
outputType='.csv'           # .txt or .csv supported
debuggingPath=outputPath+'Result_'+timestr+"/"
imageWidthLim = [400, 800]  # Max should be 2xMin+
debugging = False           #enable view and save of intermediary images and text

###################################

# structure:
# main
#   genRegex
#   importPics
#   processImages
#       detectText
#           labelImage
#       processText
#           bestAnswer

# ChemicalRegex = genChemicalRegex()
# QuantityRegex = genQuantityRegex()
# UnitsRegex = genUnitsRegex()

#main function
def runProgram():
    if (debugging):
        start = time.time()
        os.mkdir(debuggingPath)
    RegexStrings = genRegex()
    names = importPics()
    answerList = processImages(names, RegexStrings)
    print(answerList)
    np.savetxt(outputPath+"Result_"+timestr+outputType,answerList, fmt = '%-1s', delimiter=",")
    #np.savetxt(outputPath+"Name.csv",wordLists, fmt = '%-1s', delimiter=",")
    if (debugging):
        end = time.time()
        print ("Runtime: "+ str(round(end-start)) + " seconds")
        cv2.waitKey(0)
        cv2.destroyAllWindows()

```

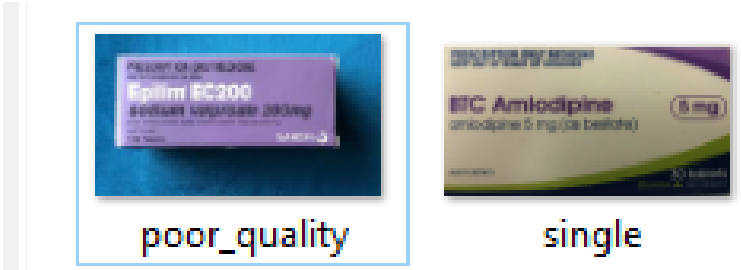
```
63
64      #generats a dictionary contain strings or string arrays
65      #they are used to process the raw text into formatted text
66      def genRegex():
67          RegexStrings = {
68              "ChemicalRegex": genChemicalRegex(),
69              "QuantityRegex": genQuantityRegex(),
70              "UnitsRegex": genUnitsRegex()
71          }
72          return RegexStrings
73
74      #the chemical name (active ingredient) is from a csv file
75      #that is able to be edited/added to independantly from this file
76      def genChemicalRegex():
77          MedicalNames = np.loadtxt("./MedicalWords.csv", dtype = 'str', delimiter = ",")
78          ChemicalBlock ="Amlodipine"
79          for n in MedicalNames:
80              ChemicalBlock = ChemicalBlock+"|"+n
81          ChemicalRegex = ".*("+ChemicalBlock+").*"
82          return ChemicalRegex
83
84      #[0] detect keyword associated with value
85      #[1] guarantee it's a number with no capital Oh's instead of zero's
86      def genQuantityRegex():
87          QuantityRegex = ["(.*(tablet|capsule|sachet).*)|(x)" , ".*[0-9]+[^O].*"]
88          return QuantityRegex
89
90      #[0] detect keyword associated with value
91      #[1] guarantee it's a number with no capital Oh's instead of zero's
92      def genUnitsRegex():
93          UnitsRegex =[ "(.*(mg).*)|(g)", ".*[0-9]+[^O].*"]
94          return UnitsRegex
95
96      #get a list of file names of the correct
97      #file type from the configured input folder
98      def importPics():
99          arr = os.listdir(inputPath)
100         # print(arr)
101         fileList = []
102         fileType = inputType[0]
103         for t in inputType:
104             fileType = fileType+"|"+t
105         for a in arr:
106             if (re.match(".+\.("+fileType+")$",a)):
107                 fileList.append(a)
108         return fileList
109
```

```
#resize the images to be roughly similar in size
#send to image pre-processing to generate augmented images and raw text
#raw text is sent to text processor
#return formatted text array
def processImages(names, RegexStrings):
    wordLists = []
    answerList = [["Filename", "Chemical", "Units", "Quanity"]]
    imageWidthLim[1] = max(imageWidthLim[1], imageWidthLim[0]*2)
    for file in names:
        img = cv2.imread(inputPath+file)
        while (img.shape[0]>imageWidthLim[1]):
            img = cv2.resize(img,(int(img.shape[1]/2),int(img.shape[0]/2)))
        while (img.shape[0]<imageWidthLim[0]):
            img = cv2.resize(img,(int(img.shape[1]*2),int(img.shape[0]*2)))

        # img = cv2.GaussianBlur(img,(7,7),0)
        edges = cv2.Canny(img,100,200)
        #edges = cv2.GaussianBlur(edges,(5,5),0)
        edges = cv2.cvtColor(edges,cv2.COLOR_GRAY2RGB)
        invedges = cv2.bitwise_not(edges)
        print("\n"+file)

        name = file.split('.')[0]
        localWordList = []
        #whiteback edges
        wordList = detectText(name, invedges, False, False, 1, 1)
        wordLists.append(wordList)
        localWordList.append(wordList)

        #blackback edges
        wordList = detectText(name, edges, False, False, 1, 2)
        wordLists.append(wordList)
        localWordList.append(wordList)

        #colour
        wordList = detectText(name, img, True, False, 1, 3)
        wordLists.append(wordList)
        localWordList.append(wordList)

        #greyscale
        wordList = detectText(name, img, False, False, 1, 4)
        wordLists.append(wordList)
        localWordList.append(wordList)

        #inverted, not running :
        # wordList = detectText(name, img, True, True, 1, 5)
        # wordLists.append(wordList)
        # localWordList.append(wordList)

        bestAnswer = processText(localWordList,name, RegexStrings)
        answerList.append(bestAnswer)

    return answerList
```

```
163
164     #augmented images have pre-processing stages applied
165     #text in images is detected
166     #these images are sent to have text lebelled
167     #return raw text
168     def detectText(name, img, colour, invert, scale, idNumber):
169         if (colour):
170             img1 = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
171             # print(name + " colour")
172             #print(img1)
173         else:
174             img1 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
175             # print(name + " grey")
176             #print(img1)
177         if (invert):
178             img1 = cv2.bitwise_not(img1)
179         if (scale!=1):
180             w,h,c = img.shape
181             img1 = cv2.resize(img1,(int(h*scale),int(w*scale)))
182             boxes = pytesseract.image_to_data(img1)
183             if (colour==False):
184                 img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2RGB)
185             wordList = labelImage(img1, colour, False, boxes, scale)
186             img1 = cv2.resize(img1,(h,w))
187             if (debugging):
188                 cv2.imshow('Result ' +str(idNumber)+'_'+str(scale)+'xS '+name,img1)
189                 cv2.imwrite(debuggingPath + name + " " + str(idNumber)+".png", img1)
190             return wordList
191         else:
192             boxes = pytesseract.image_to_data(img1)
193             if (colour==False):
194                 img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2RGB)
195             wordList = labelImage(img1, colour, False, boxes, scale)
196             if (debugging):
197                 cv2.imshow('Result ' +str(idNumber)+' '+name,img1)
198                 cv2.imwrite(debuggingPath + name + " " + str(idNumber)+".png", img1)
199             return wordList
200
201     #images are labelled and raw text array is generated
202     #return raw text
203     def labelImage(img, colour, text, boxes, scale):
204         wordList = []
205         for x,b in enumerate(boxes.splitlines()):
206             if x!=0:
207                 b = b.split()
208                 if len(b)==12:
209                     if (debugging):
210                         x,y,w,h = int(b[6]),int(b[7]),int(b[8]),int(b[9])
211                         cv2.rectangle(img,(x,y),(w+x,h+y),(0,0,255),scale)
212                         if (text):
213                             cv2.putText(img,b[11],(x,y+40),cv2.FONT_HERSHEY_COMPLEX,0.5,(50,50,255),scale)
214                     wordList.append(b[11])
215         #print(wordList)
216         return wordList
217
```

```
217
218     #raw text is inputted, and using regex a processed array is produced
219     #return processed text
220     def processText(localWordList,name, RegexStrings):
221
222         ChemicalRegex = RegexStrings["ChemicalRegex"]
223         QuantityRegex = RegexStrings["QuantityRegex"]
224         UnitsRegex = RegexStrings["UnitsRegex"]
225
226         ChemicalResults = []
227         UnitsResults = []
228         QuantityResults = []
229
230         for List in localWordList:
231             i = 0
232             for word in List:
233                 if (re.match(ChemicalRegex,word) and word not in ChemicalResults):
234                     ChemicalResults.append(List[i])
235                 if (re.match(UnitsRegex[0],word) and word not in UnitsResults):
236                     phrase = List[i-1] +" "+ List[i]
237                     if (re.match(UnitsRegex[1],phrase) and phrase not in UnitsResults):
238                         UnitsResults.append(phrase)
239                 if (re.match(QuantityRegex[0],word) and word not in QuantityResults):
240                     phrase = List[i-1] +" "+ List[i]
241                     if (re.match(QuantityRegex[1],phrase) and phrase not in QuantityResults):
242                         QuantityResults.append(List[i-1] +" "+ List[i])
243                 i = i+1
244
245         print("ChemicalResults", ChemicalResults)
246         print("UnitsResults", UnitsResults)
247         print("QuantityResults", QuantityResults)
248
249         if (debugging):
250             np.savetxt(debuggingPath+name+" RawText"+outputType,localWordList, fmt = '%-1s', delimiter=",")
251             np.savetxt(debuggingPath+name+" ChemicalResults"+outputType,ChemicalResults, fmt = '%-1s', delimiter=",")
252             np.savetxt(debuggingPath+name+" UnitsResults"+outputType,UnitsResults, fmt = '%-1s', delimiter=",")
253             np.savetxt(debuggingPath+name+" QuantityResults"+outputType,QuantityResults, fmt = '%-1s', delimiter=",")
254
255         ChemicalAnswer  = bestChemicalAnswer(ChemicalResults)
256         UnitsAnswer     = bestUnitsAnswer(UnitsResults)
257         QuantityAnswer  = bestQuantityAnswer(QuantityResults)
258         bestAnswer = [name, ChemicalAnswer, UnitsAnswer, QuantityAnswer]
259         return bestAnswer
260
261     #given a list of possible answers, the best is picked
262     #currently picks first, advanced functionality is unimplemented
263     #if list is empty, return - (empty lists have no [0])
264     def bestChemicalAnswer(ChemicalResults):
265         if (len(ChemicalResults) >0):
266             return ChemicalResults[0]
267         else:
268             return "-"
269
270     def bestUnitsAnswer(UnitsResults):
271         if (len(UnitsResults) >0):
272             return UnitsResults[0]
273         else:
274             return "-"
275
276     def bestQuantityAnswer(QuantityResults):
277         if (len(QuantityResults) >0):
278             return QuantityResults[0]
279         else:
280             return "-"
281
282     #run program by using the test file or the run file,
283     #not the elements file
284     # runProgram()
285
```

Appendix E - Input and Output Folders



Name	Date modified	Type	Size
Name_2021-04-29_11-16-10	29/04/2021 11:16 ...	CSV File	1 KB
Name_2021-04-29_11-15-16	29/04/2021 11:15 ...	CSV File	1 KB
Name_2021-04-28_16-10-23	28/04/2021 4:10 PM	CSV File	1 KB
Name_2021-04-28_00-30-40	28/04/2021 12:30 ...	CSV File	1 KB
Name_2021-04-28_00-30-08	28/04/2021 12:30 ...	CSV File	1 KB
Name_2021-04-28_00-30-01	28/04/2021 12:30 ...	CSV File	1 KB
Name_2021-04-28_00-27-05	28/04/2021 12:27 ...	CSV File	1 KB
Name_2021-04-28_00-26-50	28/04/2021 12:26 ...	CSV File	1 KB
Name_2021-04-28_00-26-32	28/04/2021 12:26 ...	CSV File	1 KB
Name_2021-04-28_00-23-07	28/04/2021 12:23 ...	CSV File	1 KB
Name	28/04/2021 12:20 ...	CSV File	1 KB
Name	27/04/2021 11:43 ...	Text Document	1 KB