GenesisCare

COMP3850 - Group 16

Testing 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 - Model Evaluation - Performance Evaluation - Appendix A-C
1.01 - 20/05/21	Lachlan Update for D4 - Added section to reflect automated testing. - Updated tonal language - Added requirements check - Addes Appendix D-F
1.02 - 21/06/21	Lachlan Updated for handover - Added expanded Testcase results

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	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

1. Model Evaluation

Project available with README.md/user manual at:

https://github.com/LachlanMatt/Comp3850GenesisCare

Previously, individual changes to the model have been evaluated by manually reviewing the debug outputs of the program (labelled photos, raw text output), as applied to a selection of test cases. This has been done on the pre-processing module extensively to get a gauge on the effectiveness of the Google Tesseract library in detecting text in the images with certain types of pre-processing having been applied (colour changes, edge detection, etc). These tests have shown a range of effectiveness for different types of processes. For example colour inversion on colour images does not help, while on greyscale images it can, according to testing performed.

As the model was not yet End-to-end complete (formatted data output, and utilising document scanning), there was not yet formal numerical analysis of the model in detecting the exact information in the test case; To remedy this a plan was developed to work towards a system that can be effectively tested and run, as of 1.00 that was:

1. Finish the draft FP (end-to-end) long before the final submission, preferable D4
2. Create test cases for known images to compare against produced results, including fields for brand name, chemical compound, number of tablets, etc
3. Create a separate testing program (test.py) which will compare the output of the main program with the test-case, and produce numeric results (e.g. percent fields correct, or weighted average with more important values)
4. Optimise the API using the testing program to compare different models and pick the most effective models (highest score metric)

As at 1.01, steps 1,2,3 have been achieved (with 4 in progress), allowing the project to be reliably tested with numeric results provided to allow for comparisons between models (appendix E). This result has slowly increased through the iterations from ~10% to ~50% on the simple test set. However, due to a portion of the project being yet unimplemented (1.01), the harder test set can not yet be tested as the API is not capable of recognising the harder images in that test set (rotated, angled, flipped, or multi photos). This will need to be one of the next points of improvement. From this the next plan of execution is

1. Implement docker compatibility to allow for easier deployment, sharing, and testing (at request of client)
2. Implement document scanning to deal angled and multi images
3. Optimise the image pre-processing and text processing (both image augmentation and regex, among others)

2. Performance Evaluation Results

As stated in Section 1, performance evaluation had previously been done by manually reviewing the debug outputs of the API (Appendix B, C). This has shown some things are more and less effective, like colour inversion. Additionally, using edges (contour map) has not been effective when passing into Google Tesseract for text detection, it will detect all of the writing as present, but will assume it is 'gibberish', a random assortment of characters rather than the actual words in the images (Appendix B, Row 1 and 2).

Qualitative chart of model performance (preliminary trials, prior to end-to-end completion)

ID	Model	Result
1	Colour inversion	Does not help with colour photos, roughly equivalent with greyscale images
2	Edge text detection	Detects location of text correctly, but does not detect the symbols correctly
3	Simple linear upscaling	Has slight benefit but drastically increases processing time due to square increase in data (image size)
4	Simple image rotation	Poor performance, does not fix skew angle, hard to automate, cuts off corners of photos unless resolution is increase a-symmetrically using sin (θ) and cos (θ)
5	Third party document scanner	Not yet tested for text recognition ability, but preliminary results showed poor performance on images that have minimal margin/edge around the box, or those where the image is blurry

With the completion of the plan first mentioned in 1.00, numerical tests have been able to be performed on the system allowing comparisons to be drawn. As most work went into the system becoming functional, less work has gone into testing comparative models. As such the only current comparison is through expansion of the chemical name dictionary, which was an effective increase in performance.

Quantitative chart of model performance (preliminary trials, document scanner not implemented)

ID	Model	Result
1	Baseline	Effectiveness: 33% correctly detected, 10% incorrectly detected, 57% undetected
2	Expanded Chemical list	Effectiveness: 48% correctly detected, 10% incorrectly detected, 43% undetected
3	Expanded TestCase list	Effectiveness: 27% correctly detected, 12% incorrectly detected, 61% undetected

Going forward from here, the difficult choice will be weighing the proportion of those 3 measurements; is it better to misidentify, or not identify at all. Additionally, what if different test sets have different performance. In these cases, the development team will need to choose pragmatically. The process for this is to consider both the times this system is to be most used, and the times the system is to be most useful. The system is to be most used on the most common photo type, which is normal simple photos. The system is to be most useful on hard or “annoying” photos to read, but these will inherently have a lower performance. From these deductions, The development team should prioritise models that are most effective on difficult photos (as these are most beneficial) but only if the performance is reasonable enough that a client could trust the results. Additionally, the development team should prioritise models with high effectiveness, and if a choice has to be made between incorrect and undetected, undetected should be chosen as false information is better than no information due to trust in the system.

More work will need to be dedicated to all sections of the processing, image preprocessing, text detection, and text processing, as all of these can have substantial impacts on the final outcome.

3. Requirements Check

The project was divided into separate requirements with different completion levels. The first is the MVP which serves as a proof of concept that the system can work as intended, by implementing the important base functions of the API (limited image recognition and text formatting). The FP then includes all facets of the project including both functionality and useability by GenesisCare. As testing is now underway, these performance requirements of the project can now be tested to determine the current success of the project.

ID	Requirement	Stage	Success Rating
1	API must be able to read the text from a simple set of test cases	MVP	Moderate, system has ~50% success on simple test cases
2	API must be able to read the text from a hard set of test cases including rotated, off angle, and multi-box photos	FP	Fail, document scanner function is not implemented so system can not be evaluated on hard images
3	API must be able to format some key text E.G. Brand, Dose	MVP	Low-Moderate, system has ~25-50% success on simple test cases
4	API must be able to format the read text from requirement 1 and 2	FP	High, system can process text moderately, and can format very well
5	API must be able to communicate with GenesisCare's Servers	FP	Fail, Communications protocols not implemented
6	API must be fully self-contained (using docker or otherwise)	FP	Fail, Docker being investigated but not yet implemented
7	API must Process a single photo in < 5 seconds	FP	Success, system takes 16 seconds for 7 simple photos, ~2sec/photo
8	API must keep medical files secure during processing and communication with the server	FP	Low, intermediary files and not saved by default, but can be saved, and are saved unencrypted, like the output
9	User manuals and Installation manuals must be produced	FP	Moderate, User manual created, feedback received

Fail	Low	Moderate	High	Success
3	1.5	2.5	1	1

4. Assumptions

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

We are testing the API for the client so they will be able to utilise it, 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 deviceAssumptions are split between Project Plan, Design Document, and Testing Document.

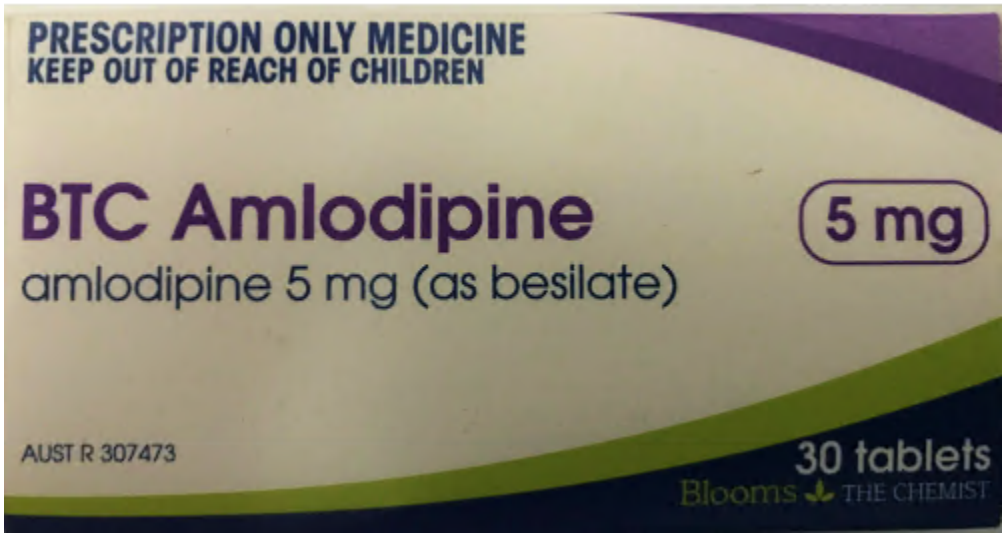
We are making an API for the client which they will be able to utilise, reducing the need for human involvement of non-critical medical activities by having a computer recognise medication lists instead of through time intensive human discussion/recording.

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

- The output of the program will follow a known template to be readable automatically
- The test cases of the program will follow the same known template as outputs
- The images in the test cases will match in order with the image files the program will read in, including file order
- Fields of test cases will be simple enough to have a reasonable chance to match from the API. Phrases should preferably be limited to 2 words (1 space separator)
- Input, output, testa cases, and other external values will be in the expected location
- Test cases will not be empty sets

Appendix A - Simple Test Case Image

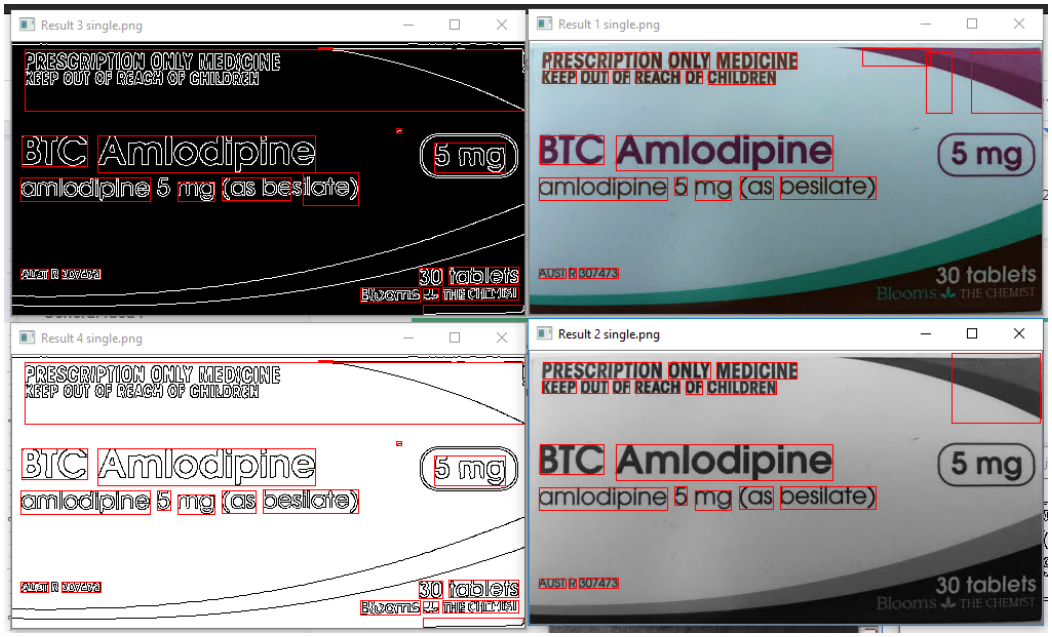


Appendix B - Simple Test Case Raw Text Output

	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'	'TECHIEI'	'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 - Combined Test Case Formatted Text Output

The final output .csv of the program for a given input testset

	A	B	C	D
1	Filename	Chemical	Units	Quanity
2	normal	desvenlafaxine	50 mg	28 x
3	normal_2	armodafinil	150 mg	-
4	normal_3	mesalazine	4 g	-
5	poor_quality	-	valproate 200mg	-
6	scan	-	peroxide (100mg/g)	-
7	scan_2	-	-	-
8	single	amlodipine	5 mg)	30 tablets

Appendix E - Test File output comparing Result, Processed, and Test text

```
In [38]: runfile('C:/Users/Lachlan/Documents/GitHub/Comp3850GenesisCare/
PerformanceTest.py', wdir='C:/Users/Lachlan/Documents/GitHub/
Comp3850GenesisCare')
Reloaded modules: ProcessingElements
i, j, Result, r, t:  i j Result Processed Testcase
i, j, Result, r, t:  1 1 desvenlafaxine desvenlafaxine desvenlafaxine
count, total:  1 1
i, j, Result, r, t:  1 2 50 mg 50mg 50mg
count, total:  2 2
i, j, Result, r, t:  1 3 28 x 28x 28tablets
i, j, Result, r, t:  2 1 armodafinil armodafinil armodafinil
count, total:  3 4
i, j, Result, r, t:  2 2 150 mg 150mg 150mg
count, total:  4 5
i, j, Result, r, t:  2 3 - - 30tablets
i, j, Result, r, t:  3 1 mesalazine mesalazine mesalazine
count, total:  5 7
i, j, Result, r, t:  3 2 4 g 4g 4g
count, total:  6 8
i, j, Result, r, t:  3 3 - - 30sachets
i, j, Result, r, t:  4 1 - - sodiumvalproate
i, j, Result, r, t:  4 2 valproate 200mg valproate200mg 200mg
i, j, Result, r, t:  4 3 - - 100tablets
i, j, Result, r, t:  5 1 - - benzoylperoxide
i, j, Result, r, t:  5 2 peroxide (100mg/g) peroxide(100mg/g) 100mg/g
count, total:  7 14
i, j, Result, r, t:  5 3 - - 50g
i, j, Result, r, t:  6 1 - - probiotic
i, j, Result, r, t:  6 2 - - 5g
i, j, Result, r, t:  6 3 - - 7sachets
i, j, Result, r, t:  7 1 amlodipine amlodipine amlodipine
count, total:  8 19
i, j, Result, r, t:  7 2 5 mg) 5mg 5mg
count, total:  9 20
i, j, Result, r, t:  7 3 30 tablets 30tablets 30tablets
count, total:  10 21
accuracy: 47.62%
Testtime: 0 seconds
```

Appendix F - Test File Code

```
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")
```