ASSESSING BLOOD SAMPLES FOR MALARIA

E. Dines

Acknowledgements

I would like to thank my project supervisor Adrian Clark for his explanations on the ideas surrounding the project. Also for his lectures on computer vision and tying in key elements of machine learning that have benefitted my knowledge and transferable skills that has made this project possible.

I would also like to thank the developers of the OpenCV library, this tool has made advanced image processing and computer vision easily accessible.

And finally, I would like to thank my parents who have supported me whilst at university, who have provided guidance when I needed it most, and always encouraged me to push that little harder.

Abstract

Malaria is a serious disease that is widespread throughout tropical parts of the world.
The World Health Organisation estimated that there were 214 million cases of malaria in 2015 with 438,000 of those being fatal. It is spread via mosquitoes and a single bite is all it takes to become infected, as mosquitoes thrive most in tropical regions it has caused a blight in these parts of the world.
The current method for diagnosis of malaria is as follows, a patient is suspected to have the symptoms related to the disease, a blood sample from the patient is taken, and then analysed by a doctor or trained professional in the area - only then it is confirmed whether or not the patient is infected with malaria. This process is expensive, time consuming and not widely available in the areas the disease affects most.
This has created a need for a computerised system that can work between the blood sample and analysis stage. A system that takes a blood smear image and determines whether any of the red blood cells featured in the image are infected with the parasite that causes malaria. This would allow someone that is trained to take blood samples from patients to carry out the first stage of diagnosing the disease, reducing the need for a doctor at this early stage. It would not serve as a definitive diagnosis tool but as a preliminary test, as misdiagnosis is common with this disease and to tell the patient quickly whether they need to seek further medical attention.

Glossary of Terms

Term	Acronym	Definition
Artificial Neural Network	ANN	See section 3.1.2 of [1].
Command Line Arguments	CLA	Arguments passed to a program via the command line (terminal).
Convolution Neural Network	CNN	See section 2.3.1 of this report.
Computer Vision	CV	(none)
Epoch	(none)	One full training cycle on training dataset.
False Negative	FN	Failing to identify an infected RBC.
False Positive	FP	Falsely identifying a RBC as parasitic.
Ground Truth	(none)	The true data that will be used as validation.
HATE	(none)	Test harness used to evaluate segmentation and classification algorithms, provided by Adrian Clark.
Hue Saturated Value	HSV	A cylindrical colour representation, where the central vertical axis shows hue, the distance from this axis is saturation, and the distance along the axis shows brightness.
Institute of Electrical & Electronic Engineers	IEEE	(none)
Image Processing	IP	(none)
Machine Learning	ML	(none)
OpenCV Library	OCVL	(none)
Operating System	OS	A software system that manages computer hardware and software resources.
Pattern Recognition	PR	(none)
Precision	(none)	Defined by: $tp / (tp + fp)$
Recall	(none)	Defined by: $tp / (tp + fn)$
Rectified Linear Unit	ReLU	Activation function for Neural Networks.
Red Blood Cell	RBC	Common blood cell that carries oxygen around the body.
Red Green Blue Value	RGB	Colour representation.
True Negative	TN	Not identifying an infected RBC when none are present.
True Positive	TP	Correctly identifying an RBC as infected
Virtual Machine	VM	See section 2.5.1 of this report.

Table of Contents

Table of Figures

1 Introduction

This introductory section aims to set the context and provide the motivations of the project. It will discuss how the project idea came to be, the overarching problems behind it, how it will be tackled and a plan that will implement the system. Some of the information discussed in this section has been mentioned in the previous reports, but it is necessary in this document to show how the goals and solutions of the project have changed over the course of the academic year.

1.1 Context

Malaria is one of the most dangerous tropical diseases in the world, spread via female mosquitos it can cause symptoms akin to flu and in some cases, can be fatal. Malaria affects areas located between the Tropic of Capricorn and Cancer, mostly equatorial, these areas are often remote (Saharan) or inaccessible (dense forest). The geographic problems mean that help can't always find its way to the people who need it most.

The desire of this project is to build a system that can automatically diagnose malaria in a patient from a single blood smear.

1.2 Problem Statement

The problem that is faced in the project is building an automated diagnosis system to determine whether a blood smear from a patient is infected with malaria or not. This can be broken down into sub-problems that can be easily managed and written into agile sprints:

1. Accepting an image as input.
2. Pre-processing the images to account for variations and create a 'level playing field'.
3. Segment RBCs from the smear image.
4. Classify these extracted RBCs into two classes, infected and uninfected.
5. Store data into memory.
6. Analyse data and generate a definitive answer to the question "Is the patient whose blood smear was analysed infected with malaria?" and the confidence percentage behind the result.

1.3 Possible Solutions

The possible solutions to each sub-problem faced are discussed below:

1. Parsing an image into a python program will be achieved by using OCVL, it has functions readily available that will read the image from memory and represent it in either a two or three-dimensional array. The two-dimensional array will be for grayscale images, n x n storing values between 0 – 255. The three-dimensional array will be for either RBG or HSV, n x n x 3, the three representing the number channels of colour variation for the two types. RGB stores 0 – 255 for all 3 values, HSV stores: H = 0 – 179, S and V = 0 – 255.
2. Images could be converted to a standard format like grayscale or HSV, in the case of converting the image to grayscale an additional layer could be to perform Histogram Equalisation. This will alter the contrast of the image but could generate uniformity across the dataset. The method will convert the image into a histogram and attempt to stretch the peaks of the graph to the full range of values. This method is useful when the background and foreground of an image are equally as bright or dark, it could make a stronger distinction between the two.
3. Take advantage of object detection functions and algorithms provided by CV libraries. Hand-crafted solutions could be developed using low-level features of the images.
4. There are several options for building a solution to this problem, again examining the low-level features of the cell could be a simple approach, or even the HSV/ RGB values. But more advanced methods could be to use machine learning that specialises in detecting the subtle differences in infected and uninfected cells. These methods are powerful, but computationally heavy and time consuming.
5. This can be done with relative ease, but might not be the case for low cost computing if the memory space is minimal. The program and the data it outputs might have to be compressed.
6. This will be based on the results from the program along with some basic medical knowledge; for example, the percentage of infection that is deemed to be a full case of malaria.

1.4 Demonstrating Accuracy

The output of the segmentation and classification algorithms will be coordinates that relate to bounding boxes of RBCs that have been extracted or deemed to be infected with malaria. Using the coordinates printed by the system these can be passed into the test harness provided by Adrian Clark called HATE. This is a test harness that has been specifically designed to train and evaluate computer vision systems. HATE has many modules, but 2 will be used for this project: 'execute' and 'analyse', the former will run a given program and store the results into a file, the latter will then analyse the output against the ground truth and create a results table that is essentially a condensed confusion matrix.

A high-quality solution to the problem should minimise FNs and maximise TPs, increasing the precision percentage, and aim to have a minimal number of FPs making sure the recall of the system is again of high percentage. With these evaluation metrics, it should indicate what is working well and give a good idea of the direction development needs to move forward in.

1.5 Goals

In addition to the system being able to diagnose malaria from blood samples, which will directly benefit the end user, some more specific goals will be outlined that will help to reach the main goal.

- Portability – the system should be able to run on as many machines as possible, with a future aspiration for the system to be mobile friendly.
- Lightweight – minimal program size with as few dependencies as possible.
- Interface – an intuitive interface that doesn't require training to use.
- Presentable Data – data that is stored in a readable manner and presented as output in a manner that a user can easily interpret it.

1.6 Methodology

The methodology chosen for this project was Agile specifically SCRUM, this has been followed since the start of the development process and was certainly a learning curve for the author. Its flexible and rapid pace meant that for example, the development of a certain algorithm is set to take 2 weeks, but 1 week in, it is found that it is no longer feasible for the product. In a waterfall methodology, the design document would have to be revisited and restarted but agile is built for changes like that and can easily update per your work plan. So the week is not lost, just side-tracked and a new task can be pulled from the product backlog.

This has happened in the development of some algorithm and function for the project, so sprints have had to be pushed back. This lost time has been recovered with some sprints being consolidated into one or finished earlier than expected.

1.7 Sprints and Milestones

The sprints and milestones outlined were those remaining from the previous report. They have now been completed showing the expected and actual finish times.

Sprint 4 (Revised): Improve Segmentation and Organise Multiple Cell Images

ID	*Work Item*	*Start*	*Expected Finish*	*Actual Finish*
1.	Finish *findContours* segmentation.	19/12/2016	23/12/2016	23/12/2016
2.	Work on applying classification to the segments found.	5/1/2017	20/1/2017	20/1/17
3.	Complete flood fill algorithm, find bounding box of uncaptured images. (Possible extension, doesn't need to be completed?)	20/1/2017	18/2/2017	10/2/2017

Sprint 4 was revised to accommodate the new functions that needed to be developed for the segmentation part of the system. Both sub-sprints 1 and 2 were completed on time, as the author had a good grasp on the necessary libraries and techniques learnt from a module studied in CV.

It was anticipated that the flood fill algorithm was take around 3 weeks to implement and develop, it was also an optional sprint as a segmentation algorithm had already been developed. It wasn't as difficult to build and was finished 1 week ahead of time. This time was reallocated to sprint 5.

Sprint 5 (Revised): Build Neural Network and Train

ID	*Work Item*	*Start*	*Expected Finish*	*Actual Finish*
1.	Research into neural networks, can one be built singlehandedly? If not what libraries are most appropriate for development?	10/2/2017	13/2/2017	13/2/2017
2.	Begin to build Neural Network and Train it.	13/2/2017	13/3/2017	15/3/2017
3.	Experiment with transfer learning.	13/3/2017	18/3/2017	18/3/2017
4.	Merge segmentation algorithm with classification. *Milestone	18/3/2017	21/3/2017	21/3/2017

Sprint 5 was updated with transfer learning, as this could end up being a vital part of how the system operates. All sprints were completed in time expect for 2, which was only two days late, as this was a difficult concept and implementation to grasp.

Sprint 6 from the previous report was removed as it was focussed on performing tests, but this was being done throughout development and it was an unnecessary milestone.

2 Background Research

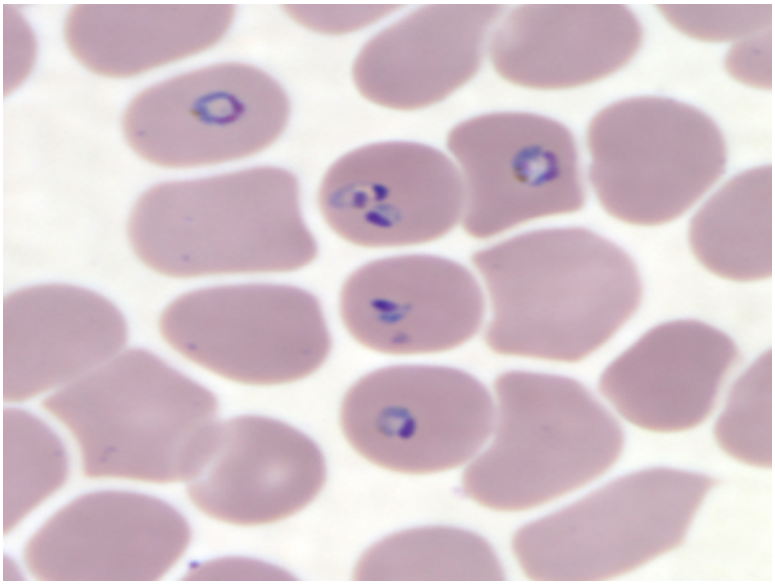
This section will extend upon the topics discussed in section 3 of the interim report [1] and section 2 of the initial report [2]. These two previous reports focussed on the definitions and basic understanding of the topics that are going to be used in the solutions to the problem being faced. The current level of research on the subject will be re-outlined in a more concise manner than in [2] and more advanced aspects will be discussed more in-depth.

2.1 Current Research

The current level of research on the topic of automated diagnosis tools for malaria has not been discussed since the initial report, which focussed mainly on F. B. Teks PhD thesis [3]. His thesis was mainly theoretical and features a lot of mathematical notation (as one would expect) which offered knowledge of how the problems he faced were overcome and how malarias parasites develop. But techniques he used to solve the problem weren't discussed at length as the work he had carried out was pioneering in the field, so understandably algorithms and code excerpts were kept to a minimal. Therefor this resource was helpful as a guide for the flow of data and ordering of processes, but no more than that.

However, during January 2017 FU-Malaria was released, it is a project spearheaded by Mike Nicholls with code and intellectual property from his company Visimagic Computer Vision Labs [4]. It is a platform that takes advantage of functions available in the OCVL for python and is able to mark RBCs with most Malaria species that affect humans. The system they have developed can run on low cost computers like the Raspberry Pi and the Intel Joule, not to mention that it can be used in conjunction with a portable low cost USB microscope to initially capture the blood smears. Along with the launch of their website to the public [5], a GitHub repository was released containing all the source code for the project [6]. This code repository has been analysed and acts as a good measure of functionality for the system that the author has implemented. The system that has been implemented by the team behind FU-Malaria is a programmatic solution. It has a knowledge base of rules that it uses step by step to finally determine whether the current RBC that it is analysing contains parasites associated with malaria.

Figure 1: FU-Malaria Test Input Image [7]



The input test image for the FU-Malaria system, this is a giemsa stained thin blood smear that highlights the RBCs and Artefacts present.

Figure 2: FU-Malaria Processed Output Image [8]

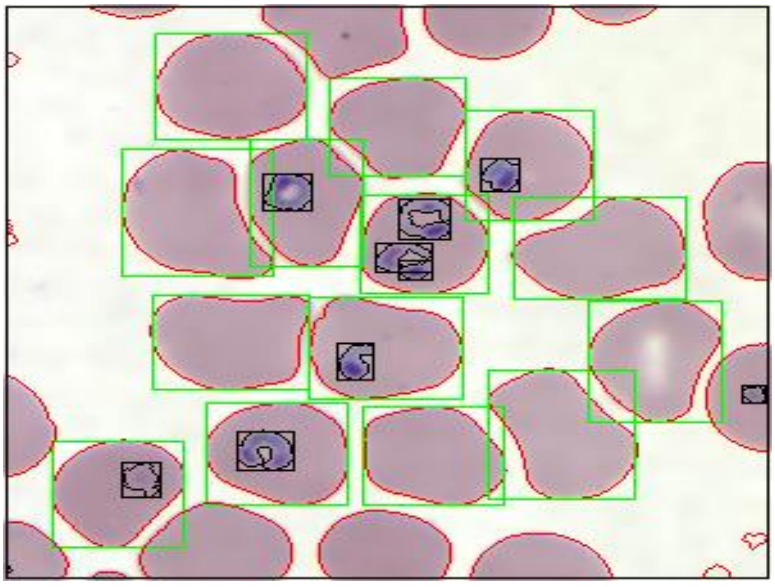


Figure 2 is the output produced by the FU-Malaria system, it has highlighted whole useable RBCs and then coloured black the parasites within the cells to indicate that they have been infected. Demonstrated in their test input and output, Figure 1 and 2, the system has a 100% detection rate of RBCs outlined in red in Figure 2. The green bounding boxes of Figure 2 show whole RBCs that are contiguous circles and have not been cut off by the edges of the image or overlapping with other cells featured. Once the RBCs that are whole have been identified by the system it then uses OCVLs *findContours* function to search for objects or continuous points within the segmented RBC. If objects within the RBC have been found these are highlighted with a black box and determined to be malarial

parasites. This is a general approximation as other outside factors could have influenced incorrect outcome, E.G. a false positive. Issues that arise for this system will be discussed further.
The *findContours* function has been used across the board in this system and can sometimes produce errors, as documented on their GitHub it will falsely identify artefacts in the smear as RBCs and the concave nature of the RBC causes a colour variation in the centre of the cell, which is also misidentified as a parasite. It does this as it relies on the simple idea of searching for objects within objects, to resolve this a more advanced system should be developed that not only considers objects presence in an image but the shape, colour and location of the parasite.
Also, not documented as an issue on their GitHub but a valid concern with how the system operates is if the system was given a blood smear that contained RBCs with the co-infection to Lyme Disease, Babesia [9]. This is a parasite like malaria, which also attacks RBCs in the hosts blood stream. When a sample of blood is taken from a patient infected with Babesia their blood smear will look similar to that of Figure 1. The system would be given a blood smear of a patient that has this disease and would be misdiagnosed as having malaria since objects of a foreign nature are present inside their RBCs.
This general release on information to the public has benefitted the project as (mentioned before) it acts as a good comparison to what is available in the research community and how this project might 'fill a gap in the market'. The source code has shown what a system constructed with a knowledge base of rules and a programmatic flow of data can achieve and the limitations it has.

2.2 Computer Vision

CV techniques have been covered extensively in the interim report [1], however after a period of development some of the techniques researched didn't prove to be as useful as hoped. This means that other techniques have been researched as an alternative to improve the accuracy of modules in the program.

2.2.1 Flood Fill

Is an algorithm that is used to determine whether a node in a multidimensional array is connected to its surrounding nodes. This algorithm can be adapted to many different problems; it is most famously used in the bucket fill function of paint programs to colour subsections of an image [10]. Flood fill is a subcategory of computer graphics algorithms, but can be reapplied to CV tasks, in the case of this project an issue that was encountered that flood fill might be able to resolve. The problem was - fully segmenting RBCs from a smear image without using object detection functions from CV libraries.
The approach that was taken was to apply a morphological gradient to the smear image to produce outlines of the cells (seen in Figure 3 of [1]), then use the outlines to highlight the cells in the original colour image. The algorithm would simply scan the morphological gradient output and any pixel that was of a white shading it would change the colour of the same pixel location in the original image to green. The output of this, Figure 4 of [1], highlights almost all cells in the image to the user, but this simple scan fill algorithm isn't able to gain useful information, such as the number of cells in the image, what's useable and what isn't, and find the bounding coordinates of each cell.
This is where the flood fill algorithm should be able to find this information, achieve the goal of segmenting RBCs from the image, and not having to use prebuilt object detection algorithms that are general purpose. By using the flood fill for each of the outlines in white it will be able to expand itself to encompass the entire circle, which can return the x and y of the top left most pixel and the bottom right most pixel, these coordinates will create a bounding box to segment the RBCs out. Each new outline the algorithm will fill can be counted, so once the full image has been processed the number of RBCs in the image can be returned. Finally, an issue previously discussed underneath Figure 4 of [1] is that the fill could overflow into RBCs that are close or even layered on top of one another, counting multiple cells as one. This can be resolved by implementing a cut off pixel count, so that it will restart the algorithm then continue scanning the subsequent white pixels as a new cell.
This new updated technique used in combination with mathematical morphology could be the answer to an improved segmentation algorithm compared to ones previously built which rely of functions like OCVLs *findContours*.

2.3 Machine Learning

This section will extend upon the interim report section 3.1 [1], that examined the definition of ML, PR and ANNs. For this next level of research, PR has now been specialised into *k*-means clustering which will perform the task of PR by associating features into clusters or by its nearest neighbour in a graph representation of the image. CNNs have been researched as they are a more specialised version of a ANN that are particularly well suited for image recognition, which will hopefully be able to classify infected and uninfected RBCs. And the final topic discussed is transfer learning; which is a conceptual idea relating to CNNs and one of the most recent developments in machine learning.

2.3.1 Convolutional Neural Network

In 1998 Yann LeCun et al. created a 7-level convolutional neural network that can classify hand written digits to a ~0.7% error rate [11]. This sparked large interested in the research community towards CNNs and how they could progress. Up until then features of the images you wished to classify had to be hard coded into tailored detectors, but as there is so much variety in the way the same image looks these tailored detectors didn't perform very well. The advantage of a CNN is that they can build neural connections, whilst being trained, that fire when a certain feature is detected even with variation considered.

To learn how CNNs operate you must first understand what a convolution is. It is composed of a kernel of a specific size, n x n, that can contain another dimension to represent the colour channels of an image, n x n x 3 in the case of RGB. This convolutional layer is the first in the CNN, creating the input into the network. The layer will scan over the image like a flashlight from the top left corner sliding right across the rows then down the columns. Doing this will create a feature map of the input image and each section that the convolution scans over is called the receptive field. This field is passed through a filter of weights that will attempt to identify specific features of the receptive field using element wise multiplication. The weights of the filters will be randomly generated at the start of training and updated using back propagation from each epoch, as these weights are updated they become better at identifying low level features of the image, such as edges, curves, textures, and colour intensities.

Like ANNs and the ideas discussed in Neural Network section of [1] the connections between the neurons and layers will fire when the summed values are greater than the threshold, when enough of the neurons fires in a fully trained network the result will show the class the image belongs to. This model of network can be built to determine the difference between a cell infected with a malarial parasite and not. A training and validation set of data will need to be produced to train then verify that the network is performing as it should, it likely that for the blood smear dataset the training classes should be manually produced from the ground truth collected early on in this academic year. The labour to build a CNN and prepare the data for input into the system will be lengthy but worth the

effort as the accuracy of the system will be much higher than what can be achieved through simple feature extraction and analysis.

2.3.2 Transfer Learning

Transfer learning, also known as inductive transfer, is a current research problem that is specific to ML, it attempts to take stored knowledge learnt from a problem and reapply this knowledge to a different problem that is somewhat related to the original. An example of this is taking a ANN that has learnt to filter spam emails, but reapplying the knowledge of the network to filter out spam in a search engine indexer. The type of spam may be different between the two domains but by retraining the network with this new data as well as what it learnt in previous sessions it can perform the task as well (if not better) as a network trained from scratch [12].

The advantages of transfer learning include, reducing training times, as the reused network has already spent hours or days being trained on computers much more powerful than those available to an individual developer. A network already trained to recognise images can easily be transferred to recognised a reduced set of images, so by having this trained network as a basis it has already learnt to identify edges, curves, textures, and colour features to name a few. This solid foundation of knowledge will greatly improve the accuracy of a network to identify the new classes it has been transferred to [13].

In November 2015, the Google Brain Team released their open sourced software library for machine learning called TensorFlow. It is used by Google for research and production level software to achieve learning in computers like that of a human and to analyse the vast amounts of data they store [14]. In this software release was Googles own CNN - Inception (currently v.3), this network was trained on the ImageNet dataset for the Large Visual Recognition Challenge from 2012. The network is able capable of differentiating between 1,000 different image classes with a 3.46% error rate [15] which is the lowest score achieved with this dataset.

It is all well and good discussing the prowess of Inception v.3, the interesting part is how the developers of the network implemented a transfer learning function that retrains the top layer of the CNN but still utilises the knowledge of the layers below. This can be done with a simple bash script and yields accuracy of 85% to 99% when tested using 5 classes of different flowers. This network and the functions TensorFlow provides to retrain Inception would allow this to be applied to the blood smear dataset to differentiate between infected and uninfected RBCs. The vast multi-layered Inception will be able to achieve a level of accuracy greater than that of a CNN developed and trained by the author.

2.4 Interface Design

It was decided by the author in the interim report [1] that the goal of having a graphical user interface outlined in the initial report [2] was not feasible for the type of researched based system that was being built. It would be an unnecessary component that would accept input and configurations that, as an alternative, could be easily implemented in a command line parser.

Having options and data being inputted through the command line can be implemented using the library *argparse* that comes as standard with python, all that is needed to be done to access the functions it provides is import the library. The library offers an easy and intuitive way of building command line interfaces. An object of type *ArgumentParser* is created, then using this object optional and positional arguments can be set up each with a description of the usage and the type of input required to satisfy the argument, an example of this can be seen in Figure 3.

Figure 3: CLA Test Using *argparse*

```
tmp-124-218:programs Elliot$ python malarian.py -h
usage: malarian.py [-h] [-sh] [-t TEST] [-g] image

positional arguments:
  image                 file path to image for processing

optional arguments:
  -h, --help            show this help message and exit
  -sh, --show           show image processing pipeline
  -t TEST, --test TEST  select different tests for image
  -g, --graph           output graphs from processing stages
```

Using this library, the goal of having an interface would be achieved but provided by CLA. This could be advantageous in the future, as if a graphical user interface was needed for a more specific user, no underlying changes would need to be made to the python source code. As an interface, could be developed to issue commands to the terminal and accept the output from the program then parse it into a graphical user interface.

2.5 Portability

One of the goals of the system is to enable its use in as many places (geographically and computationally) as possible. Research on portability and virtual environments was undertaken mainly through internet resources, as literature on this subject is mainly theoretical and offers an insight into how these types of systems are built and structured. A simple Google search returns enough information and readily available software to use to quickly solve this goal of the system.

2.5.1 Virtual Machine

The first option is using a virtual machine – which is a complete OS that acts like an autonomous computer but runs on top of the host computers OS. A piece of software that makes this possible is referred to as a *hypervisor*.

So, by using an image of a Linux based OS with all the necessary libraries and compilers installed in conjunction with virtual machine software the malaria identification system will have the ability to run on any machine with an appropriate interpreter for the host. This removes the need to have an install wizard for many different types of computers to enable all the necessary dependencies on the system for the program.

Options for software that would allow this range widely, but the most appropriate would be Oracles VirtualBox [16] which is free and open sourced to the public. It can manage multiple images of OSes with different architectures and its size is a maximum of 118MB [17], this meets the requirement of maintaining a lightweight system.

Having a virtual machine run the system does enable wide portability and the hypervisor that provides the platform is of a small size, but the issues lie in that a complete image of an OS is needed. These images could end up being very large, for instance the standard Ubuntu Desktop image is 895 MB, coupled with the various libraries that are needed to be installed E.G. OCVL and its dependencies; another 500 MB would be added on to the overall size of the OS. This would be a manageable file on a standard desktop or laptop computer, that has >5GB hard drive storage space. But for low cost small computers such as the Raspberry Pi it would be costly as more memory would be needed.

Processing large sections of data is a key part of the program and virtualisation is costly in that respect as the computer is essentially running to OSes on top of one another. This will have an impact on the time it takes for the instructions from the python code to find their way down through the interpreter, the virtualised OS, the hypervisor, and finally to the host OSes hardware. Again, the example of a Raspberry Pi having a 900Mhz ARM processor [18] does provide a good level of processing power if the program was installed directly on the computer. But in the case of virtualising say Ubuntu on top of this which as a minimum requires a 700MHz processor [19] plus the hypervisor software running in the background – processing speed will be greatly reduced.

Overall a virtual machine for the program would achieve the portability goal, but at a compromise of memory and processing speed. These factors will be considered when designing a system.

2.5.2 Containerisation

Containerisation is based on intermodal containers used in the shipping industry, these allow efficient transportation across multiple different means throughout the world. These ideas have been taken and applied to virtualisation, a container has everything an application needs to run but not a full OS. The engine these containers run on top of consists of the necessary libraries and settings that require the software to work. These containers allow efficient, lightweight, contained systems that can be guaranteed to run the same wherever it may be deployed.

In general terms, it virtualises the user-space segment of an OS removing the rest of the OS that isn't needed for the application to run, as this is already present on the computer. This leads to improved performance, with the commands from the application not having to pass through as many stages as a VM would and the start up speed of a small container is seconds compared to minutes with a VM.

Containerisation is a relatively new concept in the virtualisation community, with the main software that provides it, Docker, only being released in 2013 [20]. Docker has fully implemented the topic discussed above and can create, manage and deploy lightweight containers to servers, cloud computers and mobile systems. This removes the problem of "it works on my machine" as each container is isolated and operates completely within its own domain, no outside factors can influence the application that is being 'shipped' [21].

Docker can run on Linux and Windows, meaning it is highly portable, the containers are lightweight and Docker can automate repetitive task of setup and configuring development environments. Docker would be a good option to deploy the automated diagnosis system into as, for the components mentioned above, it would meet several goals of the project and it is free to use making it the perfect tool for the project.

3 Segmentation Algorithms

Segmentation is a key part of this project; it is integral to extracting the vital RBCs from the smear image in a clean and efficient manner. Many different approaches to this problem have been undertaken, these will be outlined below, and a single solution will be selected by its best performance which is measured by the test harness HATE.

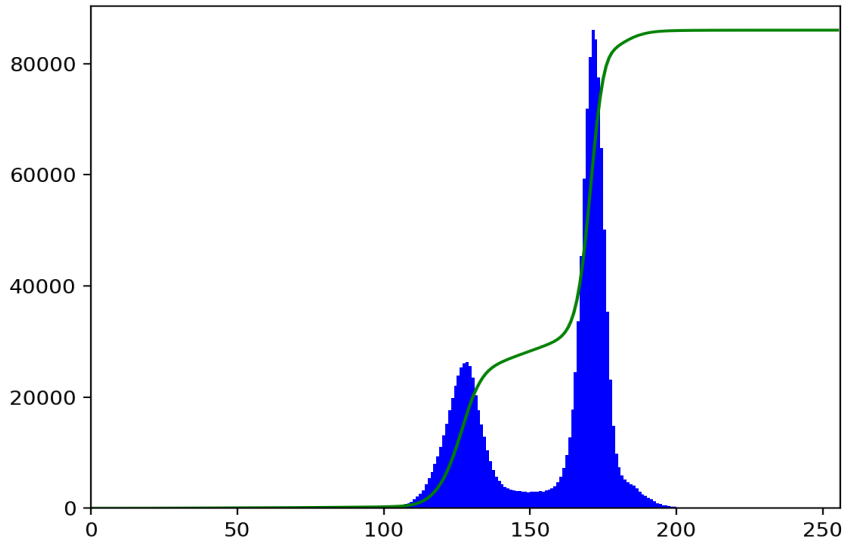
Some of the methods have previously been discussed in section 5.3.3 of [1], however since then additions to the algorithms have been made meaning that the information is now no longer valid in describing how the algorithms function.

3.1 Data Normalisation with Histogram Equalisation

Using histogram equalisation to generate uniformity throughout the dataset was discussed briefly in section 1.3 of this report. The method was not investigated further in the background research section, since it has been fully implemented in a single line of python code with the OCVL. The effects of this normalisation will be examined below.

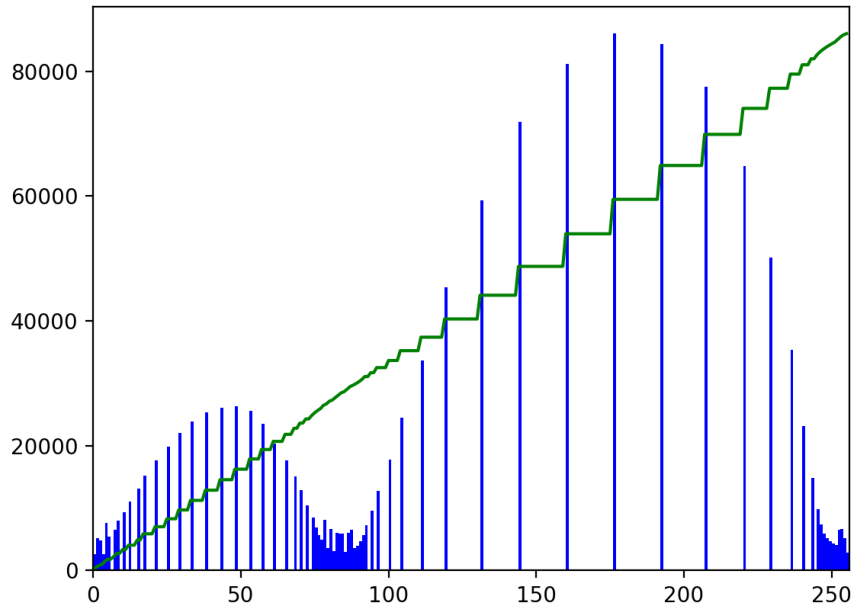
A fresh blood smear image was converted into grayscale and its histogram was generated and displayed graphically using MatPlotLib, a python library.

Figure 4: Grayscale Blood Smear Image Histogram



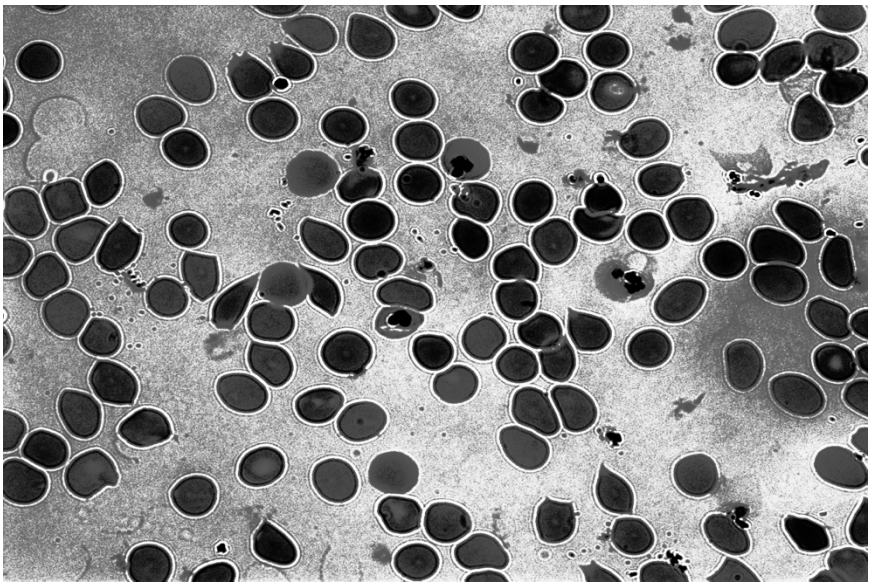
The histogram shows the 8-bit grayscale range along the *x* axis and the frequency of each value on the *y* axis. The green line represents the cumulative distribution function (CDF), calculated by cumulative sum of the flattened histogram, multiplied by the maximum value in the histogram and the divided by the maximum value in the cumulative sum. This gives a good indication to how the image varies.

Figure 5: Equalised Histogram



When the *equalizeHist* function is applied to the image, Figure 5 is the resulting histogram. Comparing this to Figure 4 the *x* axis values no longer lie between 100 and 200 but now stretch the full spectrum from 0 through to 255. The CDF now shows more linear properties compared to that in Figure 4 showing it has successfully equalised the histogram.

Figure 6: Equalised Blood Smear Image:



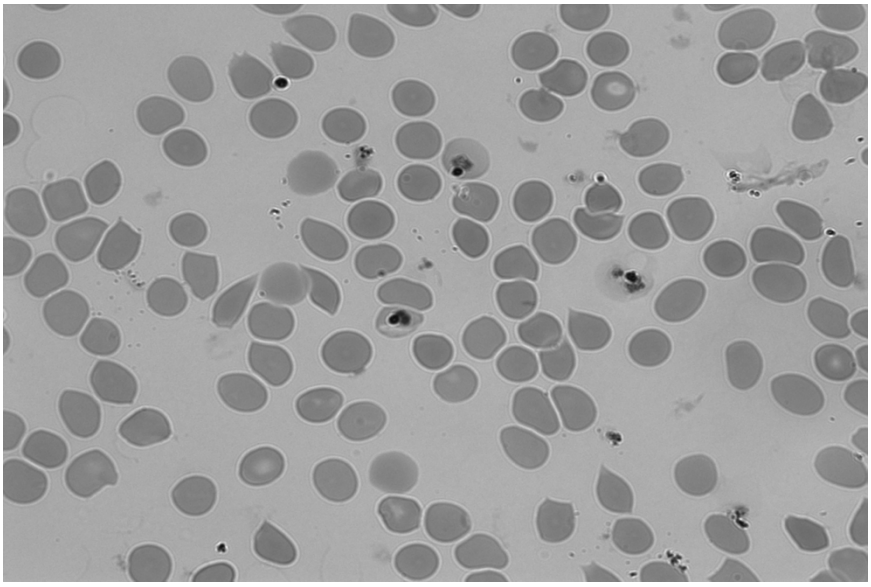
This is the image that Figure 5 represents, this is much darker than the standard grayscaled image (Figure 7) showing a greater range in contrast. Unfortunately, the opposite affect has occurred to what was expected. The hope was that histogram normalisation would remove any large anomalies in the data and create a strong distinction between the back and foreground. Instead the natural contrast of the image has been exacerbated to the point where it would impair the object detection algorithms. Therefore, it was decided that simply converting the image to grayscale was the best option.

3.2 Otsu and *findContours* Method

The *findContours* function that the OCVL provides has been used extensively in the first two algorithms that were developed. As discussed in the current research section of this report; this function is also used in the research community to achieve the same type of object detection in blood smear images. However, the function can be temperamental when applied to specific tasks - this aspect of the function was eventually discovered in the development of the algorithms that used it. But could have been avoided had the information in the current research section been found earlier on in development.

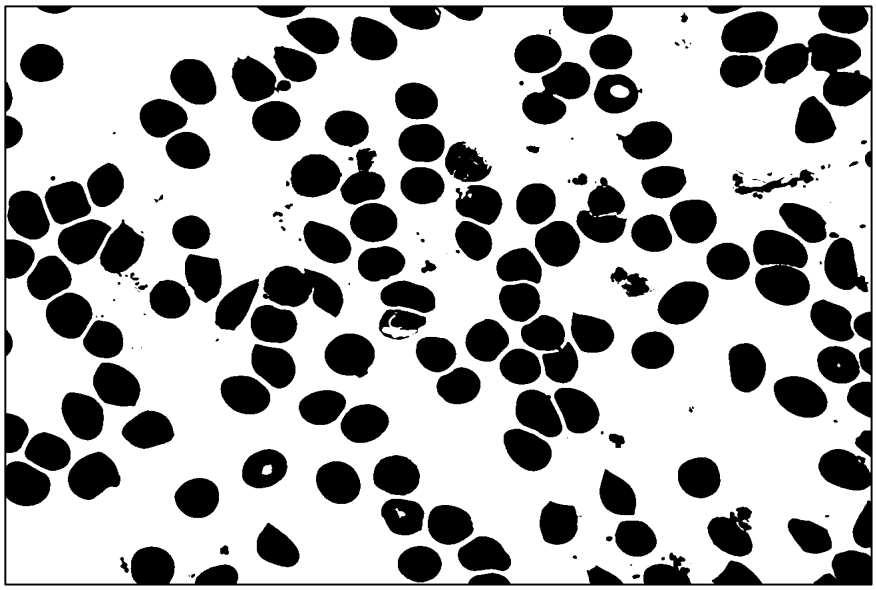
Moving on to the specifics of how this algorithm works, we must first examine Otsu's Method. It is a form of binary thresholding that will attempt to find the middle point between a bimodal histogram (an example of this can be seen in Figure 4). Once this point has been found the pixel values will be separated into two classes, a foreground and background region. OCVL provides a function called *threshold* that takes as argument a grayscale image, a range of values E.G. 0 – 255 to represent grayscale values, and a threshold type in this case Otsu. When a grayscaled image, like that seen in Figure 7, of a blood smear is inputted into the threshold hold function the resulting binary image to differentiate between objects in the foreground than the background is produced seen in Figure 8.

Figure 7: Grayscale Blood Smear Image



By converting all images inputted into the system to grayscale it removes variations in the colouring of purple from the giemsa stain and the change in lighting from one side of the image to another which is common with backlight microscopes. This image has also had a Gaussian blur to remove small amounts of noise in the image and any large variations in the colour channels that might cause anomalies in the threshold stage.

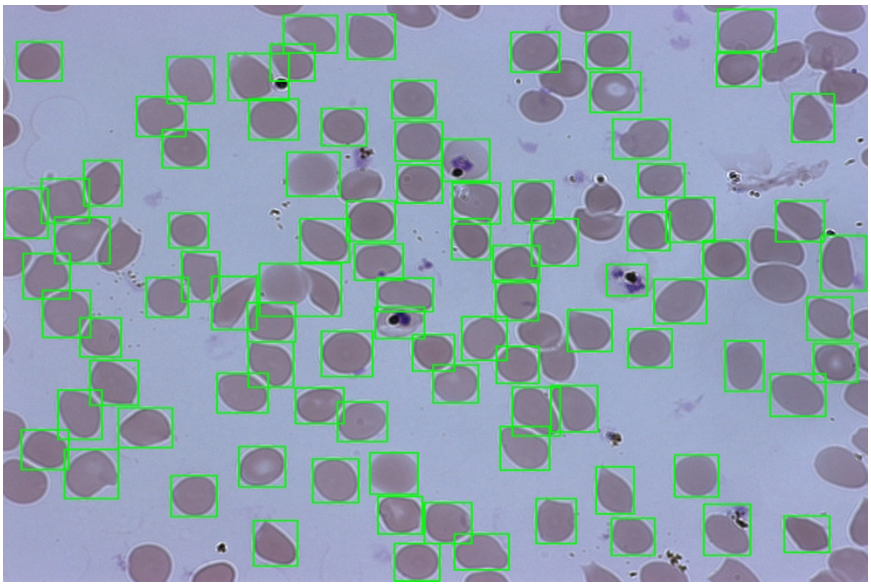
Figure 8: Otsu Threshold of Grayscale Blood Smear Image



The figure shows the output of performing an Otsu binary threshold to the image, it has successfully distinguished between the foreground and background, with them being coloured black and white respectively. A black border has been added to the image to denote where the white background of the image ends. Some RBCs in the image have been fully deemed as foreground objects (coloured completely black) but some have a light colour variation in the centre as the cells are concave platelets. This colour variation has been matched to the same colour as the background, so it appears as if some on the cells have hole through the middle, this is not the case and is an unfortunate side effect of using this thresholding method.

However, now that the edges of the RBCs in the image are much more defined an object detection function should perform well on the processed image. When Figure 8 is passed into the *findContours* function Figure 9 is the given output.

Figure 9: Object Detection on Otsu Threshold

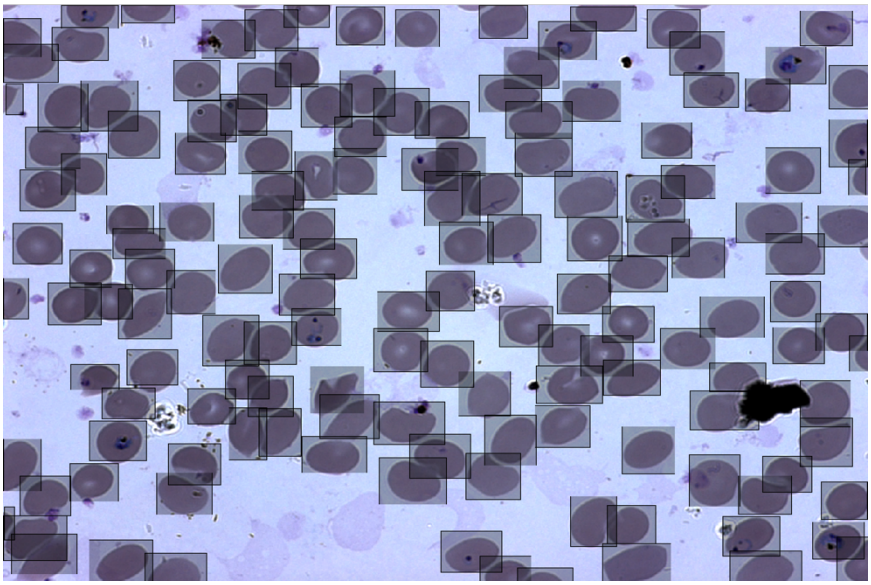


The figure shows that a high level of RBCs have been found using the object detection function. The function returns a list of contours found in the image, these can be iterated over using a for loop. Each element of the list can be used as argument to the *boundingBox* function which will create a box around the object the perfectly to perfectly encompass it. The box is then drawn onto a full colour copy of the image; these are the green boxes around the RBCs.

Although the algorithm has performed well at this early stage of development there are several issues that need to be discussed. The first being that the function can only seem to detect fully connected circular objects that haven't been cut off or overlaid in anyway. Also, the fact that it fails to detect artefact as valid objects in the image, although artefacts are not the target object, it would be helpful later in the classification stage to have these segmented from the image. The fact that it is unable to detect non-circular object is shown by how many RBCs around the perimeter of the image it has missed, these are still valid RBCs that need to be classified, so it is important for an algorithm to detect these missing cells.

To evaluate the performance of the algorithm the test harness HATE will be used. Two images from the dataset have had their ground truth found and collected, this data will serve as the actual results that will be compared with the test results from the algorithm. To give a visual representation to the ground truth, HATE also provides a *showtruth* task that will draw outlines over the ground truth coordinates that were collected from the image. This can be seen in Figure 10.

Figure 10: Ground Truth Plotted on Smear Image



This image shows just how many usable RBCs there are in the one of the smear images, although Figure 9 and 10 don't show the same exact blood smear they have the same characteristics of cells that overlap and have been cut off by the edges of the image. This shows that this is common amongst the smear images and should be considered, further development is needed to resolve the issue of an algorithm missing those cells.

Moving back to evaluation, this first segmentation algorithm was run by HATE against the ground truth images, the results can be seen in Table 1.

Table 1: Results Table of HATE Analysis

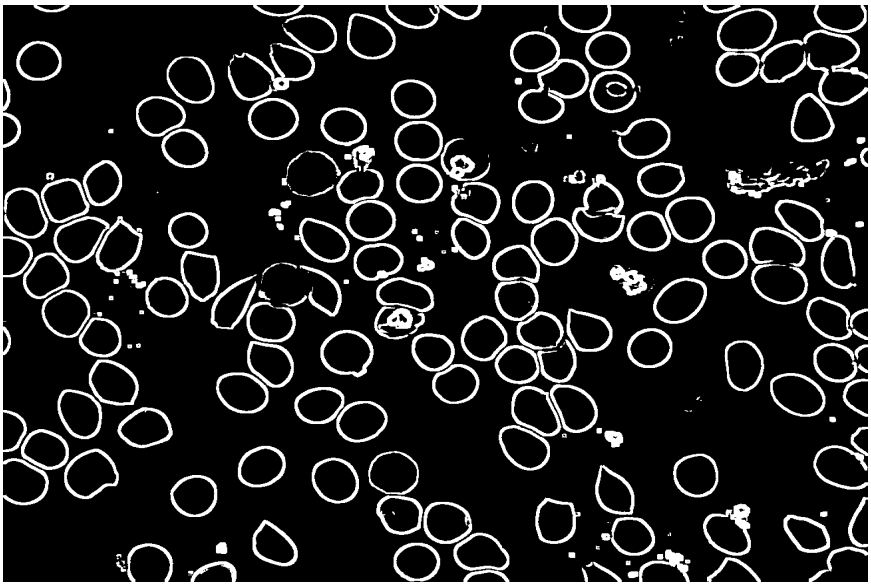
Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
304	183	0	0	121	0.602	0.602	1.000

This table of results shows that there were 304 RBCs ground truth coordinates collected, of the 304 the segmentation algorithm produced 183 coordinates that matched (found a bounding box the same as ones in Figure 10). Capturing 183/304 RBCs is an accuracy of 60.2%, this isn't particularly good for a diagnosis system that needs to ensure a high level of data is collected from the sample.
If, in the almost 40% of cells missed by the segmentation algorithm, there was a cell that was infected with malaria but none in the 60% captured correctly, the system would return information that indicates that the patient whose blood smear it is doesn't show signs of being infected with malaria. This would be a major flaw in the system, falsely diagnosing patients would add to the world-wide problem of malaria not improve it. So, it is imperative that the segmentation algorithm is capable of capturing as many RBCs in the smear as possible, even at a compromise of increasing the number of FPs (identifying artefacts and other bodily blood objects as RBCs).

3.3 Morphological Gradient and *findContours* Method

In this method, a mathematical morphology gradient is applied to the grayscale image to create outlines of the objects within the image. These types of image have been presented before in section 5.3.3 of [1], however this time an Otsu threshold was applied to the image which is in grayscale format. The expand and shrink will create halos around the RBCs and artefacts in the smear, these will not be solid objects like those in Figure 8. This is a different approach and the results that are returned should be analysed against the previous algorithm to see if improvement were made.

Figure 11: Otsu of Morphological Gradient



The objects in the figure are as close to the true definition of contours as possible, as in the contour lines of a topographical map. This is what the *findContours* algorithm is designed to find, so the test on this image should perform much better than that of the algorithm discussed in 3.2.

Figure 12: Object Detection on Morphological Gradient

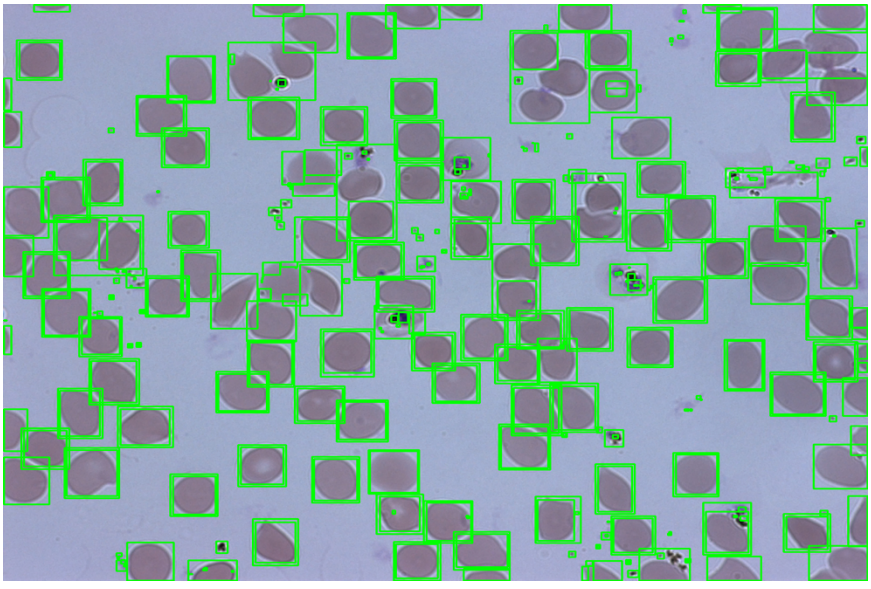


Figure 12 is the result of applying the object detection methods (used previously) to the morphological gradient image. Figure 12 is very different to that of Figure 9, multiple RBCs have been boxed several times. Small artefacts in the image have also been detected by the algorithm, these are shown by the much smaller boxes compared to the RBC boxes. The final issue is that multiple RBCs have been segmented as if they were one, these boxes will be referred to as clusters.
Although there are several issues with the above image, almost all RBCs have been found in some way or form. Even though the development of this algorithm has yet to be completed it would be interesting to see how it performs when run with HATE.

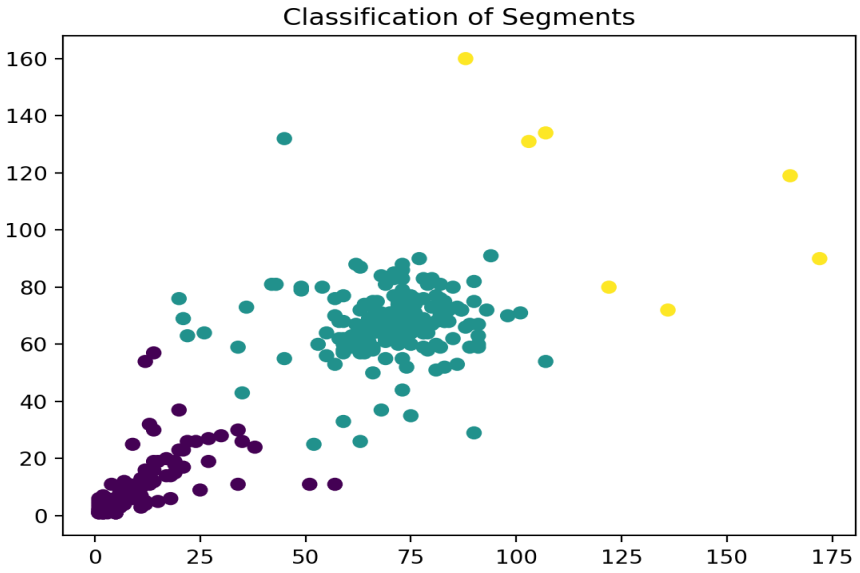
Table 2: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
304	183	0	670	0	1.000	1.000	0.312

The data shows that the algorithm is performing incorrectly, but it has shown that it is possible to fully segment all usable RBCs in the smear images, evidenced by the 100% accuracy achieved. The number of FPs has reaffirmed the issues discussed prior, the number of FPs is being increased by the double or sometimes triple segmentation of RBCs, artefact detection and clusters of cells. The reason for cluster segmentation is the fact that the outlines merge together meaning the algorithm detects them as one. As the number of FPs is so high, the ability of the system to collect the correct data (recall) is low. Therefore, the unwanted data, artefacts and clusters, need to be identified by the system and discarded. By reducing the amount of objects the algorithm detects, the number of FPs will decrease and the percentage recall will begin to improve.
A novel approach to solving this problem is to set a width (w) and height (h) threshold of the bounding box, only allowing say w and h in range 70 – 80 pixels. This would be a quick and easy solution; however, some usable RBCs can be elongated and exceed this threshold range.
The next option was to use machine learning that would be able to classify between the differences of Artefacts, RBCs and Clusters. There are some relatively advanced machine learning techniques that could be used but time restrictions remain and using that level engineering for a simple problem seems excessive.
k-means clustering seems like the most appropriate method for this issue, it is a form of vector quantisation taken from signal processing and reapplied to cluster analysis. Implementing this method on a dataset, it will attempt to partition the data into k clusters (k representing the number of clusters requested). Each observation in the dataset will be classified to a cluster based on the nearest mean. The best way to fully understand this method is to see it represented in a graph, fortunately SciKitLearn [22] a machine learning library for python provides a *k*-means class to build the clusters, then used in partnership with MatPlotLib a plotting library for python, the data can be shown in a graph and colourised to show partitions.

The data that will be used for the graph will be the *w* and *h* of the segmented image, this is the most obvious feature that determines the difference between the classes.

Figure 13: *k*-means of Segment Size

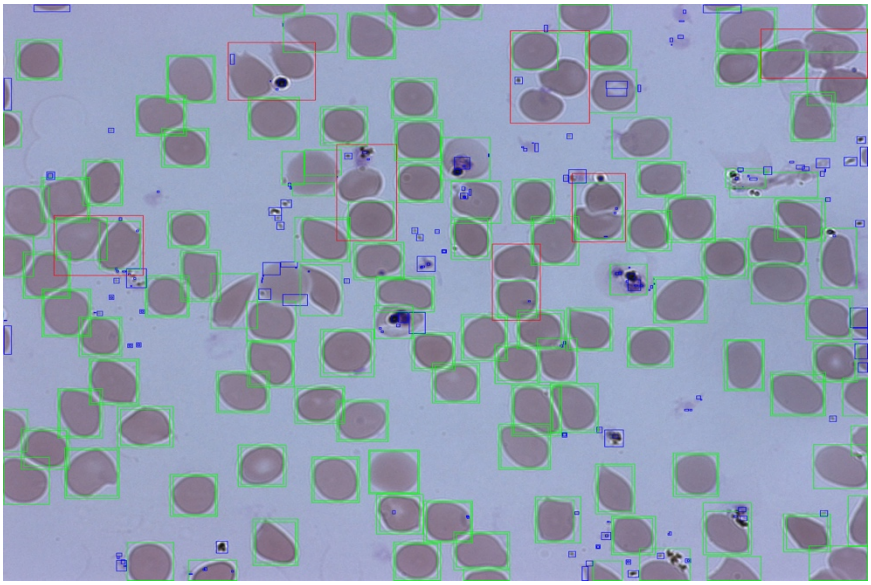


The *x* axis of the figure is the width of the segment and the *y* axis is the height. This ranges from almost 1 by 1 pixel to 170 by 120. The purple shows what the function believes to be Artefacts, the green denoting RBCs and yellow, the clusters of RBCs that are large in *w* and *h*.

Unfortunately, *k*-means does generalise the partitions and data that would be best in a different class can be misclassified. A case of this is a green data point that is 50 by 130, seen in the figure, this is likely to be a cluster of RBCs but it lies outside of the positive linear correlation of the graph, so it has been incorrectly classified as green when is should be yellow.

Even with this small amount of data misclassification the observations have been applied to the segmentation algorithm that will now colour the bounding box depending on the partition the data has been allocated to.

Figure 14: *k*-means Partitioned Bounding Boxes



The blue and red boxes are the pictorial representation of the purple and yellow partitions of Figure 13. From the image, it is clear to see that the *k*-means function has identified how the sizes vary of the segments and successfully classified a healthy number of artefacts and clusters from the objects found.

This updated algorithm needs to be tested with HATE to gain some proof that it has removed the unwanted data. Two conditionals were added to the algorithm, one to ensure that the double segmentation doesn't occur (a simple *hasItBeenFound* function) and the other to only print coordinates of segments classified into the green partition (class = 1).

Table 3: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
304	304	0	64	0	1.000	1.000	0.826

This new size classification on top of the original morphological gradient algorithm has vastly improved the recall of the system compared to that of Table 2. Recall has increased from 31.2% to 82.6%, a 50% overall increase, the number of FPs has been decreased by a factor of 10. Having 64 FPs across the two images means we can safely assume there were 32 per image, this is a reasonable margin of error for the algorithm when a standard smear of this nature will probably feature in the range of 100 to 150 RBCs.

So far in development of segmentation algorithm, this has been the only one that has correctly reached an accuracy of 100% for the test images, and a high level of recall that is reasonable for a medical based system like this one.

3.4 Flood Fill Method

The final method was to apply a type of flood fill algorithm, discussed in 2.2.1, to the Otsu of a Morphological Gradient seen in Figure 11. This algorithm should be able to locate each individual RBCs or other objects in the smear and find the bounding box for each. It will operate in a similar way to the algorithms based around the *findContours* function but will instead use a hand-crafted object detection algorithm specifically for RBC extraction.

The flood fill algorithm will operate recursively and take advantage of the array representation of an image, thanks to OCVL. The algorithm is of importance to how the segmentation is going to operate, its structure will be outlined below in brief pseudocode.

Algorithm 1: Flood Fill for RBC Extraction

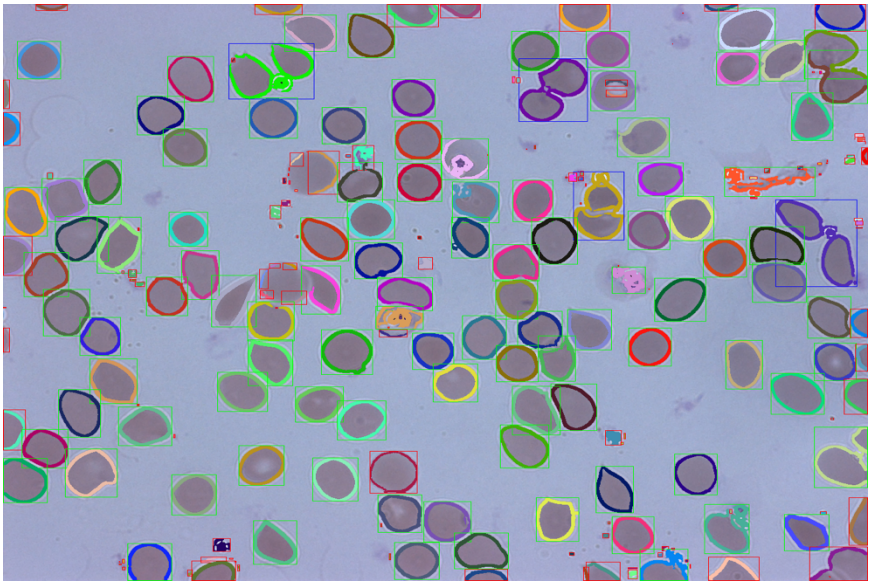
Flood-Fill (y, x, oldColour, newColour):

1. If y or x is greater than the width or height of the image, return.
2. If y or x is less than 0, return.
3. If pixel at (y, x) is equal to newColour, return.
4. Set pixel at (y, x) to newColour.
5. Call Flood-Fill function with:
    a. (y+1, x, oldColour, newColour)
    b. (y-1, x, oldColour, newColour)
    c. (y, x+1, oldColour, newColour)
    d. (y, x-1, oldColour, newColour)
6. Return.

This is a general-purpose flood fill algorithm that will form the basis of the code implementation for the segmentation algorithm. Some additional lines will be added in the implementation to count the number of pixels that have been converted per RBC outline and a minimum-maximum function for the coordinates converted, to find the bounding box.

The algorithm was written into python code along with a scanner that will only do the initial call to the method when a white pixel is encountered in the image array. The function returns when one of the base cases are met, either lines 1, 2, 3, or 6.

Figure 15: Flood Fill Test on Smear Image



This figure is the result of running the Otsu Morphological Gradient through the flood fill algorithm. The colour of the outline is randomly generated to show the distinction between the RBCs, some RBCs that are close together have the same colour, as their outlines overlap so the expansive search of the algorithm has filled both. This was an issue discussed in the interim report [1], but has been resolved by using a pixel count threshold where if the number of pixels (*np*) converted exceeds a certain amount this will trigger a return base case in the method, restarting the process.

The bounding boxes have been found using the minimum and maximum of the coordinates from each outline found, and the colourisation of the bounding boxes is the same as the *k*-means algorithm but is purely based on *np* in that region. For example, Artefact is $np < 900$, a cluster is $np > 2000$ and RBCs are everything in-between.

Table 4: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
304	304	0	7	0	1.000	1.000	0.977

The results from HATE show that the number of FPs for the algorithm has reached the lowest score yet of 7, making the new recall rate 97.7%. This algorithm has achieved the best results so far, but is has yet to be tested on a dataset no larger than 2 smear images (see section 6.4). With these results this algorithm will be used as the default segmentation function. The number of RBCs found in the image has also been found using this method, there were 117 in Figure 15.

It has taken a lot of time and effort to get these results, this is because information on RBC segmentation from blood smear images is still a new subject in the research community. Also, there are many ways of gaining the same results, demonstrated by the 4 chapters of this section, and it was important that these methods were explored and tested.

4 Cell Classification

This section will focus on the different methods of CV and ML that were used to classify parasitic and non-parasitic RBCs. This chapter will not feature as many figures as the previous as the methods discussed favour numerical data over visual representations.

4.1 RGB Examination

The first method that was tested was examining the segmented RBCs are searching for variation in RGB values that would create a clear partition between infected and uninfected cells. This is a trivial solution to the problem but often simple programs can produce the best results.

Using XV – image viewing and modifying software, some thresholds were found that represent the colours of infected RBCs. One set of colours found is a dark purple, this comes from the giemsa stain that parasites absorb more than cells do. When the parasite has begun to fully mature in the RBCs it forms its own nucleus that is coloured almost black, this defined colour is useful as black doesn't occur naturally in RBCs or artefacts in a smear sample.

These colour ranges are applied to the segmented RBC and a binary mask is created of pixels that fall into one or more of the categories. Using Numpy arrays and OCVLs *countNonZero* function we can find how many pixels have been found, when this number exceeds 10 the cell is classified as being infected.

Table 5: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
1001	769	0	652	232	0.7682	0.7682	0.5412

HATE was now run with the ground truth of the infected RBCs; this dataset is of all 100 images from the sample. There is a total of 1001 cells infected with malaria, and this simple system correctly identified 769 of them. This has given the system an accuracy rating of 76.82%.

However, the idea of using pixel count is too basic for this problem. The colours that have been associated with parasitic cells can occur in nonparasitic cells due to unsmooth staining of the sample and caused by other foreign objects in the smear. This has caused a sharp spike in the number of FPs, 652, greater than half of the number of tests, meaning ~46% of the total number of cells selected were incorrect. This is misidentifying far too many uninfected cells as infected.

232 infected cells were completely missed by the algorithm, showing that not all infected cells will contain the colours that have been associated with them. A more advanced approach should not only consider the colour, but the shape, size, and other features the RBCs hold. There is a lot of data available and it should be taken advantage of.

4.2 Convolution Neural Network Application

CNNs will compute lots of features of the image, this will not be hand programmed but learnt through training and validation. The intricacies of CNNs have been discussed in section 2.3.1, this will now be implemented in python and tested with HATE to get an accuracy rating.

4.2.1 Generating Learning Classes

For a CNN to learn the differences between image classes, classes of training data must be prepared. It was expected at first that two classes of training data, infected and uninfected, would have to be collected by hand. This would be the most sensible and accurate approach but time is of the essence, so the classes were generated using some basic python scripts.

The first was a script that scanned the ground truth of the infected cells from images 50 through to 100. The code cut the true infected cells out of the images and stored them in the infected class file, in total there were around 500 images. To select all uninfected cells the flood fill segmentation algorithm from chapter 3 was reapplied to images that had had the infected cells removed, meaning that all cells remaining in the smear were uninfected. This created a class of around 4,500 images, the data was now ready to train a neural network.

4.2.2 Keras

To build a CNN for cell classification Googles Tensorflow library was used but this library doesn't offer a direct approach for designing your own network. Luckily Keras is a wrapper class for Tensorflow and Theano, the idea behind the API was to enable fast experimentation for machine learning, and this is how it will be used for this project.

Keras is very intuitive to use, each layer of a model can be created in a single line of code. The first stage of developing the CNN was creating a sequential model, that is a linear stack of layers, that will be treated as objects and feed data downwards to the next.

The first layer implemented is a convolutional one that is 32 x 32 x 3 array of pixel values from the images, this is the topic discussed in 2.3.1, each element in the array will represent the intensities and our CNN should be able to learn to classify these intensities.

The feature map generated from the first layer will be passed through an activation layer (like the threshold of a neuron) that will use ReLU. ReLU is a non-linear operation that will replace all the negative pixel values in the feature map with 0, there are other activation function like sigmoid but ReLU tends to perform better on image classification tasks. Using ReLU increases the non-linear relationship in the data, so the neural network will learn more complex functions than standard linear regression [23].

This structure has been replicated three more times, two for hidden layers of the network and one for the output. The hidden layers of the network will hopefully learn more abstract features associated with infected and uninfected cells.

Overfitting might occur in this network; this is where the network has become too dependent on the training data and when it is given data it has not seen before it will fail to correctly classify it. To resolve this, training epochs should be kept minimal for a small training set like this one and a dropout layer can be added to the model. This will nullify random activations in the hidden layer by setting them to 0, and by doing this overfitting should be minimised.

A loss function will measure the difference between the target output and the expected output, this will be minimised to improve the accuracy of the network and used to update the weights of the hidden layers.

All in all, this model can be implemented in less than 100 lines of code using Keras and Tensorflow, the source code for this can be seen in appendix A.1. under 'classifiers'.

The CNN was trained with a rebalanced class set, it was previously 500/4500, but is now 500/500. The data was rebalanced as the network will have a heavy biased to always classifying any input as the larger dataset, in this case uninfected [24]. When a CNN is trained on an unbalanced dataset the validation will return highly accurate results but this isn't the case. The accuracy paradox [25] demonstrates that a CNN will gain > 90% accuracy but when tested on unseen data, will perform poorly. This is because the metric of accuracy is not returning and testing enough data to show that the CNN has a biased towards the larger dataset. These problems can be simply avoided by having minimal epochs and balanced data.

For this test epochs was set to 5 and steps set to 70, the results of the trained CNN with HATE, which took around 20 minutes to train and another 20 minutes to test, can be seen below.

Table 6: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
1001	865	0	278	136	0.8641	0.8641	0.7568

When the network was training, validation accuracy was around 97%, but the accuracy of the 100 images was 86.41%. This difference was expected as the CNN had not seen this data before and features might have been seen that it might not have learnt in training. 86.41% is a high accuracy and what a medical system like this one should be achieving. The number of FPs compared to Table 5 has been decreased by almost 2/3, this gives the recall a rating of 75.68%.

For further development of the CNN, other activation functions could be tried and tested, along with added hidden layers. This could decrease the number of FPs, but might overcomplicate a network that only needs to be small to achieve a good accuracy.

4.3 Transfer Learning on Ground Truth Classes

With the infected and uninfected data classes being prepared for the previous CNN, this is all that is needed to use TensorFlows Inception a pre-trained CNN that can transfer its knowledge onto a new dataset. This was discussed extensively in 2.3.2.

Inception is trained by calling its retrain python program with a pointer to a file containing the classes, this took around 30 minutes to train on the 1000 image class dataset. Inception can now be interfaced in the same way as section 4.2.2, a segmented image is passed into the model and a prediction will be returned.

This was run with HATE, but since Inception has 48 layers it would take around 2-3 minutes to classify all the cells in a single image. This meant it would take around 200-300 minutes to get the results from HATE. Unfortunately, Docker will time out before training has completed but the results file can still be recovered. This results table only features 747 tests for the reasons mentioned.

Table 7: Results Table of HATE Analysis

Tests	TP	TN	FP	FN	Accuracy	Precision	Recall
747	685	0	485	62	0.917	0.917	0.585

Of the tests completed by Inception the accuracy was 91.7% which is an improvement on the Keras CNN, but it should be kept in mind that the full ability of Inception is unknown as the test wasn't fully completed. With this in mind, the large number of FPs shows that it isn't preforming very well compared to the previous, its recall is poor, identifying too many other objects as infected.

It was expected that Inception would perform much better than the previous method, but Inception has been trained on images that occur in a real-life context not on blood smears and RBCs.

4.4 Classification Options

Of the 3 different classification methods tested one needs to be chosen as the default for the system. In the authors opinion, the hand built Keras CNN is the best option, it performs faster than Inception and only requires a model file of size 800 kB and a python interface file of 3kB.

Inception does offer a high accuracy rating, but its recall is too low for a system were a 1% difference of infection rate could result in a positive diagnosis of malaria (see section 5.2).

The RBG examination of 4.1 is too rudimentary and returns poor results in a test environment, if it was sent out into the wild where there are lots of variations in smears and blood it wouldn't perform as well.

The Keras CNN is the best options as it has both a high accuracy and a good recall. Using this method also allows users to change the design of the model and add new data into the classes, so that every time a new smear is entered into the system it can learn from it and improve its overall ability. This couldn't be done with Inception as retraining is very CPU intensive and Keras is not, making it perfect for low cost computing.

5 Complete System

In this chapter, we will explore how the algorithm from chapter 3 and classification method from chapter 4 can be merged together to build one system that will meet the goals of section 1.5.

5.1 Full Python System

As discussed in the background research section 2.5.2, Docker a tool for containerisation, would be used to make the system highly portable. It is easy to install taking just a few minutes and creating new containers for application in single lines on the terminal. To take advantage of the full power of Docker some basic set up bash scripts were written that can run Docker, create the relevant container and install the libraries needed for the system to run if an internet connection is available. These bash scripts are specific to Unix like OSes but run inside Docker so these are platform independent, the only part that would have to target individual platforms is the installer for Docker and the scripts. But instead of building an install wizard this could be done by the user following a tutorial.

Once the system has been fully installed and all dependencies have been properly configured, the system can be run using '. /malarian' with one compulsory argument - an image location, and several optional ones. This command line interface has been implemented using pythons *argparse*, the final amount of optional arguments has yet to be decided, as the user could have many different options. They could choose between the 3 different segmentation algorithms for their needs, or the 3 different classification methods if they want to carry out test of their own for research say. A pre-set default will be implemented to use the most effective of each stage so that the best results are found for the user.

All the different aspects of the system have been aggregate into one using python *import*, the other files have been consolidated into files of function with no main methods. So, these are easy to import and hand pick what functions the user wants to use via the optional arguments. These libraries that have been created are reusable as they are modular in nature, for example the segmentation library could be reused for another disease.

5.2 Infection Calculation

One of the problems outlined for the system was giving a definitive answer to whether a patient is infected with malaria or not. With the data that has been collected an answer can be found to the best of the system's ability, using some basic formulas.

The number of cells in each smear needs to be known and the amount of cell that are believed to be parasitic.

Formula 1: Infection Calculation [26]

$$\textit{\% of infection} = (\text{Parasitic RBCs} / \text{Total RBCs}) * 100$$

This calculation will return the percentage of parasitemia, which is a measure of infection in the blood stream. Per [27], a website that specialises in malaria information exchange, a result of less than 1% is in a margin of error and the test should be taken again straight away or at a later date. However, if the percentage is in the range of 2-3% or greater, this would be defined as a full case of malaria and medical assistance is required for the patient.

This formula and conditions have been programmed into the system, and produce information to the user like, "infected with malaria, seek medical attention", "low count, retake test" or "no parasites found, not infected with malaria". This results are shown with the percentage as well.

6 Evaluation

This final chapter marks the end of the implementation, the previous served to demonstrate how the system was created and tested. This section will evaluate the system that has been created and comment on achievements that have been made.

6.1 Project Planning

The planning of the project flowed well with the work load, however several of the sprints from previous reports were either updated or removed. This was permissible as agile was being used, so when stages of development changed the sprints were flexible. Most sprints in the second half of the academic year were met on or ahead of time. This allow the reallocation of resources to new ideas and methods that have improved the system.

In the first half of the year a large amount of time was spent on developing segmentation algorithms; this was because there was little information about how this could be done theoretically and implementing it into code. This wasn't wasted time as there were some good results from the algorithms that have been developed, and they rival that of systems released for commercial use.

More time could've been spent on the classification stage of the system, and this is reflected in the amount of information in chapter 3 than 4. But a lot of the subjects in chapter 3 were best represented in figures, which made the chapter long than the rest and 'quality over quantity'. Even if more time was available for the classification it might not have been needed, as a lot of the system that was trying to be built had extensive documentation and tutorials on how to use the machine learning libraries needed. Also, the fact that the data was prepared meant that setting up neural networks to learn from it wasn't as time consuming as one would expect.

However, using sprints and Microsoft project to plan was useful, but a tool that is specifically designed for this type of development might have proved useful for maintaining a backlog of tasks that needed to be completed. This is something that could be used in a future project.

6.2 Conclusion

The overall aim of this project was to develop a system that could automatically diagnose malaria in a patient from a single blood smear image. The various components and knowledge needed to be built into the system for this to be achieved, like computer vision methods to extract the cells in the image, machine learning to learn the differences between parasitic and nonparasitic cells, and specific parasite blood content calculations. This has all been achieved with the dataset that was provided and implemented with a user interface in python.

Although the system has been proven to work, with an 86% accuracy, it has been tailored to the data that was provided by Adrian Clark. This means that yes it operates to a high standard in the test environment but sending it out into the wild, the accuracy might not be as high as expected. Since not all microscopes will take images like that in the dataset, the images might be taken at a shorter distance (magnification) or a new camera might have a different focal length as the one that was used for the test set. For the images to work properly in the system they should be taken so they produce images like that of the test dataset. This could be resolved by adding in some weights that the user will input if a different camera set up is being used.

There are many possible variations that could affect the system and its ability to classify cells, all the possible outcomes will not be discussed but acknowledged as a future problem of the system. These issues of variation in the target market could be addressed by having feedback methods to the developer so that new issues that arise could be dealt with. Or the code could be made opened sourced so that if an error is encountered, anyone with knowledge of CV and ML could readjust the code to overcome their specific problem.

All the additional goals outlined in 1.5 have been achieve in one way or another. The portability of the system has been reached by using Docker to manage application containers that hold and run the necessary libraries and dataset for the system. The system size is in total around 1 GB, with 500 – 800 MB of libraries for OCVL and Tensorflow, then 100 MB for the smear image data set, then the remaining size leftover for the algorithms and models generated for the systems algorithms. This could be more lightweight as the smear images don't need to be shipped with the program and Docker doesn't virtualise memory so 1 GB in a container is 1 GB in memory, this could be reduced by using a VM to have virtual memory.

The interface of the system implemented in argparse isn't very advanced and will only take basic string input, but expanding the functionality too much could present new challenges for the end user. So even though it is basic, it is to the level that a layman could understand and use effectively.

The final output of the system is in a presentable manner; this can be stored to a file that includes the ground truth results, the percentage of infection in the blood and a recommendation by the system for the patient.

The system was completed in good time, which is surprising as development sections were revisited and side tracked from time to time. The author believes that implementation was made much easier and quicker by carrying out extensive background research. With this knowledge, how the system needed to be implemented was clear from the start and coding it seemed like second nature.
The system still has a high number of FP recordings in the classification stage, 278 to be exact. This is a manageable amount at this current time, if more time was available for development some new techniques could be applied to reduce that number. If this development stage was to be done again more time would've been reclaimed from development of segmentation algorithms and used to improve the machine learning part of the system.

6.3 Personal Reflection

Looking back on this project, it has been a hard task. One of the hardest parts of developing the system was getting a segmentation algorithm to work to a reasonable level of accuracy, without producing large amounts of false positive readings. I think this is one of the reasons why segmentation took longer that was expected, as I became captivated by reducing the number of FPs. This side-track in the end was useful as the final flood fill algorithm lived up to all my expectations. It was also nice to build my own new unique algorithm to segment the RBCs and not having to rely too heavily on a CV library.
The machine learning approach was difficult at first, it was quite a learning curve. This topic has never been studied in full and is always mentioned in passing in other subjects. So, it was interesting to learn an entire new subject that was fairly advanced on my own back. Although Keras and Inception do abstract away some of the more technical mathematical properties of neural networks, like gradient decent and derivatives of loss functions for back propagation. I still feel as a programmer that I could solve a wide range of problems with my new knowledge of machine learning and more over neural networks.
But writing this final report has made me realise that yes there were times I struggled with the concepts and implementations but I am proud of myself for achieving everything I set out to do. And it feels good to know that the system I have designed could make a difference is someone else's life.

6.4 Further Implementation

There were some ideas and goals that couldn't been reached in this development process, however these will be outlined below to show where the system could go to in the future.

The system could be more accurate and account for more variations if the data set was larger, for both segmentation algorithms and classification methods. The code for each stage could be made open sourced to allow experts from around the world to make contributions to improve the software or medical researchers to submit more thin blood smears to the database to help show the system how wide ranging the data can be.

If the system is developed beyond a small low cost computer that can be shipped to the remote parts of the world, a mobile application for smart phone could be designed to collect blood smear images and return an accurate diagnosis. Some of the areas the disease affects like tropical Africa have undergone a technology jump, where no telephone lines exist but satellite communications do. So, access to internet connected smart phones is more common than one would expect.

The CNN that was used was trained with 5 epochs, this could be increased to guarantee a better accuracy and recall rate, but increasing the epoch could compromise the system by risking a high level of overfitting. An alternative to reduce FPs and not increase epochs is to try different activation functions like *sigmoid*, *tanh* or *linear*. These different activations might have a better affect than ReLU for this specific problem.

And the final extension would be to develop the code into a RESTful API that would remove the need for a dedicated system run in Docker. This would allow all manner of applications to access the API, web browsers, mobile applications, laptops, anything with an internet connection. The code wouldn't have to be changed a great deal as Flask provides a web framework for python.

7 References

[1] E. Dines, “Interim Report,” 2016.

[2] E.Dines, “Initial Report,” 2016.

[3] F. B. Tek, “Computerised Diagnosis of Malaria,” 2007.

[4] M. Nicholls, 2017. [Online]. Available: http://visimagic.com/.

[5] M. Nicholls, “About,” January 2017. [Online]. Available: http://fumalaria.com/.

[6] FU-Malaria, “FU-Malaria Code Repository,” 2017. [Online]. Available: https://github.com/fu-malaria/fu-malaria.

[7] M. Nicholls, “Input Image,” [Online]. Available: https://github.com/fu-malaria/fu-malaria/blob/master/Pf_rings_thinC.jpg.

[8] M. Nicholls, “Output Image,” [Online]. Available: https://github.com/fu-malaria/fu-malaria/blob/master/Pf_rings_thinB.jpg.

[9] “Babesia,” [Online]. Available: https://www.lymedisease.org/lyme-basics/co-infections/babesia/.

[10] “Paint Bucket Tool,” [Online]. Available: https://www.getpaint.net/doc/latest/PaintBucket.html.

[11] Y. L. e. al., “Gradient-Based Learning Applied to Document Recognition,” 1998. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[12] L. Y. Pratt, “Discriminability-Based Transfer between Neural Networks,” in *Neural Information Processing Systems Conference*, 1993.

[13] W. L. ,. Y. J. ,. P. S. ,. S. R. ,. D. A. ,. D. E. ,. V. V. ,. A. R. Christian Szegedy, “Going Deeper with Convolutions,” [Online]. Available: http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf.

[14] B. Team, “About,” [Online]. Available: https://www.tensorflow.org/about/.

[15] V. V. I. S. W. Christian Szegedy, “Rethinking the Inception Architecture for Computer Vision,” [Online]. Available: https://arxiv.org/pdf/1512.00567.pdf.

[16] “Overview of Oracle VM VirtualBox,” Oracle, [Online]. Available: http://www.oracle.com/technetwork/server-storage/virtualbox/overview/index.html.

[17] “Downloads of Oracle VM VirtualBox,” Oracle, [Online]. Available: http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html.

[18] “Raspberry Pi 2 Model B Overview,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[19] “Installation/System Requiements,” [Online]. Available: https://help.ubuntu.com/community/Installation/SystemRequirements.

[20] “Company History,” [Online]. Available: https://www.docker.com/company.

[21] “About,” [Online]. Available: https://www.docker.com/what-docker.

[22] S. Developers, “Documentation,” [Online]. Available: http://scikit-learn.org/stable/documentation.html. [Accessed 2016].

[23] A. Karpathy, “Convolutional Neural Networks for Visual Recognition,” [Online]. Available: http://cs231n.github.io/convolutional-networks/.

[24] D. J. Brownlee, “Tactics to combat imbalanced classes,” [Online]. Available: http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[25] D. J. Brownlee, “Accuarcy Paradox,” [Online]. Available: http://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/.

[26] “Blood Specimens - Microscopic Examination,” [Online]. Available: https://www.cdc.gov/dpdx/diagnosticprocedures/blood/microexam.html.

[27] “Microscopic Tests,” [Online]. Available: http://www.malariasite.com/microscopic-tests/.

[28] E. Dines, “Entry,” *Log Book of Project,* 2016-2017.

[29] O. D. Team, “Morphological Transformations,” 10 November 2014. [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. [Accessed 2016].

[30] O. D. Team, “Image Thresholding,” 2016. [Online]. Available: http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html. [Accessed 2016].

[31] R. B. &. T. Jackson, Neural Computing, York: Institute of Physics Publishing, 1990.

[32] F. a. Ponse, Computer Vision - A Modern Approach.

[33] A. Clark, “Feature detection and description”.

[34] G. T. Developers, “Tensorflow Python API,” [Online]. Available: https://www.tensorflow.org/api_docs/python/. [Accessed 2016].

[35] O. D. Team, “Changing Colourspaces,” [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html. [Accessed 2016].

[36] World Health Organisation, “Number of malaria deaths,” 2015. [Online]. Available: http://www.who.int/gho/malaria/epidemic/deaths/en/. [Accessed 2016].

[37] NIH Fogarty International Center, “Scientists Report that Misdiagnosis of Malaria is Common 349,” 2008. [Online]. Available:

https://www.fic.nih.gov/news/globalhealthmatters/pages/malaria-misdiagnosis.aspx. [Accessed 2016].

[38] T. M. Mitchell, Machine Learning, Maidenhead: McGraw-Hill, 1997.

[39] E.Dines, *Product Backlog From Initial Report,* 2016.

[40] Elliot Dines, "Initial Report," 2016.

8 Appendices

A.1 Automated Diagnosis System Source Code

The code for this project has been stored in a GitHub Repository, found at: https://github.com/elldi/malarian.