

**Project Title: Assessing Blood Samples for Malaria**

**Name: Monika Delekta**

**Registration Number: 1505036**

**Supervisor: Dr Alba Garcia Seco De Herrera**

**Second Supervisor: Dr Adrian Clark**

**Degree: BSc Computer Science**

## **Acknowledgements**

I would like to acknowledge my supervisor Dr Alba Garcia, for her support and feedback during each stage of the project.

## **Abstract**

Malaria is spread by a parasite carried in the female mosquito, which is passed to a human when the mosquito bites them. Areas facing poverty are the most affected by Malaria, this is because there is a lack of proper healthcare. It is too expensive to pay laboratory assistants to diagnose blood samples for Malaria. To save costs, these countries simply provide anti-malarial medication to anyone showing any signs or symptoms of Malaria under clinical diagnosis. This, unfortunately, creates a drug resistance making the anti-malarial drugs ineffective. The focus of this project is to produce software that can automatically diagnose Malaria in images of Giemsa stained erythrocytes under a microscope.

The solution was to produce three different versions, this allowed using different techniques to find the most appropriate solution. The first is programmed to take the image and remove sections of it using histogram analysis, thresholding and morphology in stages, until it is left with only the dark blue sections in the image that are assumed to contain the parasite. The second program uses a machine learning approach that has never been used for this type of implementation before, this is Haar Cascading. Similar to the face recognition applications it is used for in smartphones, the cascade was trained to detect the infected cells within the images. The third program is a CNN, this is another machine learning approach which was trained to determine if cells are infected or uninfected through the use of feature and class classification.

Evaluation shows that the CNN returns the highest accuracy and f-measure score, closely behind is the Haar Cascade. The image processing through segmentation program returns the lowest accuracy, making it not useful for real life implementation. The Haar Cascade, however, proves to be the best method, as although it has a lower accuracy and f-measure score than the CNN it is a much more cost-effective method to implement in areas facing poverty.

# Table of Contents

1. Defining Malaria .....	6
1.1 Diagnosis .....	6
1.2 Geographical Location of Malaria.....	6
1.3 The Link Between Malaria and Poverty .....	6
1.4 The Malaria Drug Resistance .....	7
1.5 Why Malaria is a Problem .....	7
1.6 Previous Work in Automatically Detecting Malaria.....	8
2. Project Aims and Objectives .....	9
2.1 Researching Various Techniques .....	10
2.1.1 Image Pre-Processing Techniques .....	10
2.1.2 Haar Cascading Techniques .....	11
2.1.3 Neural Network Techniques .....	13
2.2 Methodology .....	15
3. System Requirements.....	16
4. Design .....	17
4.1 Understanding the Images Provided .....	17
4.2 Program 1 - Image Processing Through Segmentation.....	18
4.2.1 Thresholding .....	19
4.2.2 Region Labelling.....	21
4.2.3 Morphology .....	21
4.2.4 Counting the Number of Cells .....	22
4.2.5 Centre Points .....	22
4.2.6 Colour Rebalancing and Finding Infected Cells.....	23
4.3 Program 2 - Haar Cascading .....	25
4.3.1 Training the Classifier .....	25
4.3.2 Positive Image Set .....	26
4.3.3 Negative Image Set .....	26
4.3.4 Feeding Images to the Classifier .....	26
4.3.5 The Vector File .....	27
4.3.6 Training the Haar Cascade.....	27
4.3.7 Classifying the Images .....	29
4.3.8 Cropping Tool.....	29
4.3.9 Detecting with the Trained Haar Cascade .....	30
4.4 Program 3 – Convolutional Neural Network .....	30
4.4.1 Training Process .....	31

4.4.2 Detection Phase .....	35
5. Performance Measure Tools .....	35
5.1 Producing the Performance Measure Evaluation Programs .....	36
5.2 Program 1 - Image Processing Through Segmentation.....	36
5.3 Program 2 - Haar Cascade .....	38
5.4 Program 3 - Convolutional Neural Network.....	38
6. Test Results .....	39
7. Evaluating Test Results .....	39
7.1 Program 1 – Image Processing Through Segmentation .....	40
7.1.1 Why Program 1 Return Such Low Results .....	40
7.2 Program 2 – Haar Cascade .....	41
7.3 Program 3 – Convolutional Neural Network .....	42
7.4 Accuracy Vs. F-measure .....	42
8. Limitations and Constraints .....	43
9. Project Planning.....	44
9.1 User stories.....	44
9.2 Backlog .....	45
9.3 Reviewing the Project Management .....	45
9.3.1 Program 1 - Image Processing Through Segmentation .....	45
9.3.2 Program 2 - Haar Cascading.....	46
9.3.3 Program 3 – Convolutional Neural Network .....	46
9.4 Risk Management .....	47
9.5 Achievements .....	47
9.6 Evaluating the Agile Methodology.....	48
10. Code Adaptions and Modifications Used .....	48
11. Legal .....	49
12. Discussion and Concluding Remarks.....	49
12.1 Future Extensions .....	50
13. Definitions and Acronyms .....	51
Bibliography .....	52
Appendix.....	55
Appendix 1 – Morphological Operators Applied to Colour Rebalanced Image .....	55
Appendix 2 – Program 1 Detection Output File .....	55
Appendix 3 – Haar Default Values of Training Process Explained.....	56
Appendix 4 – Training Log for Haar Cascade.....	56
Appendix 5 – Haar Detection Text File Output .....	59

Appendix 6 – Neural Network Image Detection Results .....	59
Appendix 7 – Image Processing Text File Evaluation Results .....	60
Appendix 8 – Haar Evaluation Results .....	62
Appendix 9 – Neural Network Evaluation Results for Background and No Background .....	62
No Background .....	62
With Background .....	63
Appendix 10 – Product Backlog.....	63

## 1. Defining Malaria

Recordings of Malaria date back to as early as 2700 BC in Chinese documents and was described as a major factor in the decline of populations. [1] It was only in 1880, that French surgeon Charles Louis Alphonse Laveran discovered a pattern of asexual parasites named Plasmodium in the blood of his patients which closely linked to the symptoms of Malaria, based on their size and the severity of the patients' symptoms [2]. Since the discovery, it is now known that there are five species of Plasmodium, these are Falciparum, Vivax, Ovale, Malariae, and Knowlesi, all of which have a similar life cycle.

The parasite begins its life-cycle in the salivary glands of the female Anopheles mosquito as a 'sporozoite', i.e. a form of the parasite that is motile and initiates the asexual cycle [3]. The parasite is transferred to a humans' bloodstream when the mosquito bites them, in turn taking blood and leaving behind some of its saliva in the bloodstream. Sporozoites initially enter the bloodstream and find their way to liver hepatocytes where the asexual process begins through splitting and several merozoites are produced. Merozoites then move back into the bloodstream and attach themselves to erythrocytes where they enter the cell and take over by consuming the haemoglobin until eventually the cell is destroyed and more merozoites are released into the bloodstream.

Once the human is infected, the parasite moves quickly, where it causes flu-like symptoms to arise from as early as 7-18 days of becoming infected. The main symptom is fever, but other symptoms also include chills, sweating, nausea/vomiting and headaches [4].

### 1.1 Diagnosis

Malaria can be diagnosed in several ways, but the most common and most effective is with light microscopy. Here a blood sample is prepared as a smear and is stained with the Giemsa stain, making the infection/parasite appear darker in colour in comparison to the cells. Analysing the sample under a light microscope is very time-consuming and requires a lot of expertise in detecting the parasite. Clinical diagnosis is also possible but leads to patients being misdiagnosed as symptoms are not a true indication of Malaria.

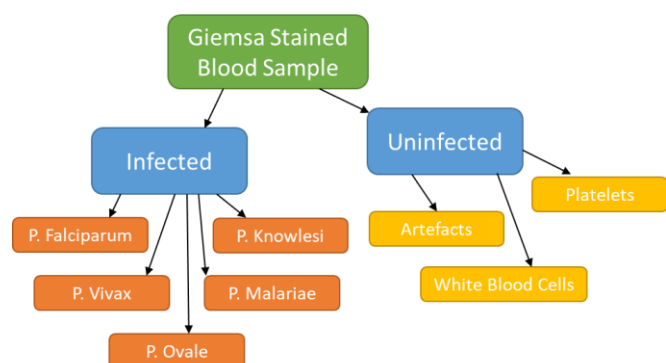


Figure 1: Flow of parasite detection process

### 1.2 Geographical Location of Malaria

Malaria tends to be found in climates between 16-32 degrees Celsius, particularly in tropical and sub-tropical areas, including parts of Africa, South America and Asia [5]. Research by D. E. Loy, et al [6], shows evidence of Plasmodium becoming prominent in these areas where its growth began in chimpanzees and gorillas, which then spread to humans within these areas. The largest amount of reported cases today is in sub-Saharan Africa, more specifically in the central and northern areas of the African continent. WHO identified an estimated 445,000 deaths globally in 2016 from Malaria, with 91% of those from sub-Saharan Africa, with mostly children under the age of 5 affected [7]. This is where Malaria becomes a major problem to society.

### 1.3 The Link Between Malaria and Poverty

Sub-Saharan Africa is not only known for its scorching temperatures and hospitable environment for the Plasmodium parasite but it is also known to struggle with the largest amount of poverty in the world, with around 383 million people living in poverty. It is closely followed by the second highest, Asia, with 327 million [8]. This shows that Malaria and the number of deaths are strongly linked to the

economic welfare of the area it exists within. The main cause of Malaria infections occurring each year is from the lack of education surrounding the topic within these areas facing poverty. A study by S. A. Fana, et al [9] showed less than half of pregnant women in the sample size of 255 women in Nigeria, were educated on prevention measures associated with Malaria. This is a leading factor to deaths in young children in these areas, where the mother is unaware of how to protect her child. Governments also play a significant role in the high number of deaths from Malaria each year. They do not invest enough money in educating such mothers and young children.

The economic crisis within these areas leaves little funding for appropriate healthcare. The costs of paying phlebotomists are simply too high. Light microscopy, is the best method for diagnosis with a 99% accuracy, but it costs around \$21 per hour alongside the costs of equipment and takes around 45 minutes to evaluate each sample [10]. Evaluating the blood samples is a complex task, so it is required to hire educated phlebotomists who can perform the diagnosis correctly, but with the costs so high the government cannot afford to hire and keep them. Doctors are then forced to provide anti-malarial drugs to anyone with symptoms similar to those of Malaria, this could easily be misdiagnosed as the symptoms are so similar to those of the influenza virus. This clinical misdiagnosis issue is leading to a very serious drug resistance.

#### 1.4 The Malaria Drug Resistance

Misdiagnosing the symptoms is not the only factor that creates a drug resistance. Physicians who clinically diagnose patients with Malaria will in most cases not know the severity of the infection, so the incorrect doses are often provided. The lack of healthcare costs also often leads to a poorer quality of the anti-malarial drugs available. These all lead to the failure of incorrect treatment. In the case of misdiagnosis, it leads to the patients' immune system being unable to use the anti-malarial medication in the future because they become immune to the medication [11]. In terms of the incorrect dosages provided to patients who are infected, if the dose is too weak, then the parasite will form a strain strong enough to survive any future higher dosages, making it close to impossible to cure. Alternatively, if the dose is too high, the body will begin to become immune to the anti-malarial medication. The only way to avoid incorrect dosages is to assess a blood sample to view how severe the infection is by determining the number of erythrocyte cells infected. Currently, the speed of drug resistance is developing much faster than the speed of new drugs being produced [12].

#### 1.5 Why Malaria is a Problem

Misdiagnosis on such a scale is causing an endemic of Plasmodium strains to gain a resistance to the anti-malarial drugs created to combat it, which will inevitably end in uncontrollable mobility rates if not controlled. Healthcare costs in areas facing poverty are simply too high which leads to a 'quick fix' in the diagnosis of Malaria set by the government, where only clinical diagnosis is conducted because the costs of microscopic analysis are too high.

The ideal solution would be to produce an algorithm or piece of software that takes images of smeared blood samples, process the images and provides the user with feedback as to whether or not the images are infected with Plasmodium or not. This will benefit any area significantly as the only technical work required would be to create the smear, but from that point, hourly rates for experienced phlebotomists can be avoided, as well as the money saved from anti-malarial drugs that would have been given in many of the misdiagnosed cases.

## 1.6 Previous Work in Automatically Detecting Malaria

Work already conducted to automate this process has already been noted within the initial report [10]. However further research has uncovered more interesting approaches to improve or recreate the current solutions available.

The article by F. Boray Tek, et al [13], defined a method to initially perform pre-processing stages on the image, this consisted of colour correction of the images to consistently represent the colour of all stained objects in the images. To do this area granulometry is used, this estimates the size of the area of each object within the image. The analysis using granulometry can be explained as the evaluation of particles within the image, to reach an estimation of the particle sizes and their specific distributions. For vision systems, this is ideal as it can be automated to search within a specific ROI [14]. The illumination of the images is corrected and the foreground is segmented from the background using an average cell area value (preserving areas of that size).

To identify if cells are infected the K nearest neighbour classifier (KNN) was used, which bases detection on feature similarity. Here if a feature in the training data is similar to a query vector it is assigned to that vector. KNN also defines the distance between two pattern vectors. The vector created in this article was through using the values of a colour histogram of the image, the area granulometry and the measurements of the shapes in the image. In this case the tuning factor for KNN was changed to find various features in the images, in most cases defects left over in the image were being detected as well as the infected cells, this could be tuned but in this case, some infected cells are lost. The article does, however, argue that they required more data to test the results completely.

Also following a granulometry approach to automatically detect infected cells in stained blood smears was the article by C. Di Ruberto, et al [15]. In this case, granulometry was performed on greyscale images, which were then thresholded through the value of the green channel of the image. Morphological operators were then applied to remove further defects. Two different methods of classification were used, the first a morphological approach through a process called skeletonisation. Skeletonisation consists of the removal of most of the foreground of the image by eroding the foreground pixels away from the boundary leaving only the endpoints, essentially leaving the images 'skeleton' [16]. Pattern recognition was then used on the skeleton to find the infected cells. The second approach used was through the analysis of the colour histogram for each cell where if there was a darker shade present this indicated infection within the cell. The article concludes with both methods performing just as well as each other, however, the morphological approach performs better under better lighting and more exposure to the Giemsa stain to make the infection darker.

Taking the classification process a step further was the article by S. Annaldas, et al [17]. This article followed a similar approach to the article by C. Di Ruberto, et al [15], where the green channel was used for thresholding purposes to remove the background from the image. The most interesting idea behind this article was the use of two different classification techniques that both employ a machine learning approach. The first is an ANN, in this case, a pattern neural network was used, which has four layers to identify inputs, patterns, summation, and outputs. The second classification method is SVM, this is a non-linear classifier and is used for pattern recognition. The way in which SVM works is by placing the data on a plane and differentiates it into two or more classes by finding the hyper-plane i.e. the dividing line [18]. An evaluation showed that the two classifiers returned high results. The ANN offered a 78.53% accuracy and the SVM offered a 98.25% accuracy when tested on 70 images. The downside of this article is that it does not state how the images are sized for training which is an important factor.



Interested in the ANN approach, the article by S. Premaratne, et al [19] was evaluated. The difference in this article to the one by S. Annaldas, et al [17], is that they only searched for Plasmodium Falciparum in their ring stages, as it had been identified that these were the main cause of deaths. The cells were cropped from the images into a size of 64x64 and converted to greyscale for training. The neural network architecture used was Feed Forward Backpropagation, this is defined as information only travelling in the forward direction and backpropagation defines any errors by subtracting the trained output from the desired result to be propagated back to the previous layers for re-evaluation. Upon successful training the model was evaluated on 50 images, the article states that it could correctly identify most cases but a complete evaluation had not yet been conducted.

The article by M. Razzak [20], defines clear results from an ANN trained and tested which can be closely compared to the ANN by S. Annaldas returning a 78.53% accuracy. The ANN trained by M. Razzak [20] is trained with pre-processed images where the background is initially removed and the images are segmented using ROI segmentation. Geometrical features such as cell area and roundness were used to crop the cells separately. The main difference between the two articles is that M. Razzak has used a Backpropagation Neural Network architecture, in comparison to the Pattern Neural Network used by S. Annaldas. Although the article does not state the number of images tested, the overall results were still excellent. Results were calculated for each species but ranged from 89.2%-97.9% accuracy. It can be concluded by reviewing the two articles that the choice in the neural network model used will greatly affect the results achieved. Although the number of images used to train and test is not certain it does show quite a large difference between two model architectures.

## 2. Project Aims and Objectives

This section defines the solution to solving the problem based on the research undertaken so far. For the proposed solution, the objectives and goals will be emphasised, as well as the project management methodology to be implemented.

Although there has already been work conducted on the automatic detection of Malaria, further research into the topic to gain improved accuracies and a more efficient method is necessary for such a life critical piece of software. The research already conducted in section 1.6 has shown a vast range of techniques used to approach the problem, with the most commonly used being image segmentation using image processing techniques, SVM and ANN.

In terms of image processing techniques, there are several ways in which the background can be removed to gain only the foreground objects i.e. the erythrocytes within the images. The vast number of techniques that can be used show room for improvement in some of the current techniques already implemented. The research into SVM's has already returned excellent accuracies, so for this project, it has been decided that they will not be required to be tested and improved upon.

In terms of ANN's, there have been several research papers released on the use of them to automatically detect Malaria in erythrocytes, however, the paper by S. Krishnan, et al [21] has shown an interesting use of how changing the number of layers in the network architecture can affect the final accuracy result. The paper by F. Ertam and G. Aydin [22], has also shown that by simply changing the activation function used in the network architecture, the accuracies can vary significantly on the same datasets. This research into neural networks shows that there is room for new research into these, as there are so many ways in which to approach the problem.

For this project, it has been decided that the best way to approach finding the best solution is to produce more than one program to automatically detect Malaria. Through having more than one solution there can be a direct comparison made to find the best solution, as well as any benefits that

could outweigh one or the other. From the research undertaken in section 1 above, there is one main goal in mind, that being to produce three pieces of software that can be improved on or attempted from using a different approach than has already been used in current research. The first is to use image processing techniques where the image will undergo various segmentation techniques to remove the background. Once the background has been removed, new approaches in image processing will be used in order to change the colour space of the erythrocytes to find the infection within them. This is the simplest method which can be directly compared to the second approach produced, which is using machine learning to implement a CNN. The aim of the CNN is to monitor it in terms of how the features appear in various activation layers to ensure the data is being used correctly and is not being overfitted. The aim is also to use fewer layers within the model architecture to improve efficiency and reduce overfitting alongside this.

The third program is another machine learning approach, called Haar Cascading, this shows signs of promise, where it works well on extracting features from faces and is also commonly used for object detection such as number plate recognition. The use of Haar Cascading has not been used for the detection of Malaria as of yet, so the aim for the results returned is that they can be closely compared to the other two programs produced and may therefore show new research possibilities and solutions.

Overall, the production of three different approaches, including one that has never been applied to solving the automatic detection of Malaria before, will allow for an interesting evaluation. The aim of the project is to then evaluate all three programs produced which will show the most compatible, in terms of accuracy and cost-effectiveness to be used in areas facing poverty. This ultimately will meet the goal to prevent the drug resistance from forming further and will save the government money on healthcare costs. In order to meet these goals, further research was undertaken into the techniques to be used for the production of the programs.

## 2.1 Researching Various Techniques

With an understanding of previous work completed and a clear goal to produce the three programs in mind, further research was conducted into understanding some of the techniques that stood out or were found during research. These techniques provide a deeper understanding of how the background removal and detection process could be implemented using the available techniques.

### 2.1.1 Image Pre-Processing Techniques

Thresholding is the simplest type of segmentation, where if a pixels' value is below the threshold it becomes a certain value and if above the threshold it is changed to another. There are various ways in which thresholding can be applied, but the most interesting is Otsu Thresholding. A review of the algorithm was completed by H. Vala, et al [23], here it defined two types of thresholding techniques. The first is global thresholding which depends on the grey-level values of the entire image. The second is local thresholding which segments the image into various sub-regions and uses various thresholding values for each. Otsu thresholding uses global thresholding in a two-dimensional approach which considers the grey-level information from an images histogram as well as the spatial correlation information of neighbouring pixels. Therefore, Otsu provides the most uniform thresholding results, especially on noisy images.

Otsu Thresholding has been used on the segmentation of background from images of flowers by A. Patil and J. Shaikh [24]. Here the approach was followed to segment the image into two parts, the foreground, and the background. This was done using histogram analysis alongside class variance equations to separate the values. The images were first converted into the LAB colour space, which defines the 'L' value as luminance, the 'A' value as a value extending from green to red and the 'B'

value which extending from blue to yellow. Once converted the article explains using only the 'B' channel of the colour space and applying Otsu thresholding to the image, this gave exceptional results which is an interesting approach to adapt.

Another image processing technique is the concept of morphological transformations. This is a technique that applies operations to an image based on shapes within it, the technique also requires a convolution kernel which determines the values for neighbouring pixels. The two basic forms of this technique are erosion and dilation. Erosion 'erodes' away the boundaries of the foreground objects using the values set in the kernel. Dilation does the opposite, where the foreground boundary is increased in size. Dilation and erosion have also been packaged into an opening and closing technique. The opening technique is an iteration of erosion followed by a dilation and the closing technique is an iteration of dilation followed by erosion [25].

Bilateral filtering as explained by C. Tomasi and R. Manduchi [26], is a filtering technique that smooths images to remove noise but also preserves the edges present within the image. The way in which the bilateral filter functions is by replacing pixels with a specific value if they are within the vicinity of each other or show similarity. The bilateral filter will average out weak differences between pixels which results in noise removal. The bilateral filter works especially well with images in the LAB colour space. This is an ideal filter if the edges of the image are necessary. Alternatively, if the edges are not necessary then the median filter should be used. The median filter determines if a pixel is relevant to its surroundings and if it is then it changes the pixel to a value generated by the median of its surrounding pixels [27].

Connected component labelling takes an image as an input and groups objects within the image based on their pixel connectivity [28]. This form of labelling can be useful in terms of finding specific areas within the image to eliminate background objects that are of a different size.

### 2.1.2 Haar Cascading Techniques

The Haar Cascading approach adopts the Viola-Jones algorithm used for face detection, but can also be trained to detect objects. There is no information available on the use of this technique on the detection of Malarial parasites within erythrocytes, which shows that this is a new area of research that could be of use to the research society if successful. Viola and Jones [29] introduced the machine learning approach in 2001, it processes the images very quickly by using features extracted rather than pixels. There are three features extracted from the images, a two-rectangle feature which uses two rectangular areas and takes their pixel count difference, these rectangles are always the same size and can be seen in figure 2 below and can be represented as either i or ii. The second feature extracted, shown in figure 3 below is a three-rectangle feature which is the sum of the two outside rectangles pixel counts subtracted from the centre rectangles pixel count. The final feature extracted, shown in figure 4 below is a four-rectangle feature, which is the sum of two diagonal rectangles subtracted from the opposing diagonal rectangles.

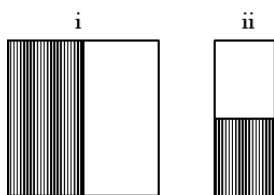


Figure 2: Two-rectangle feature

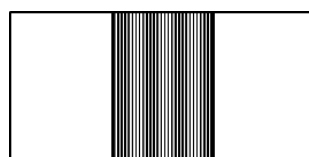


Figure 3: Three-rectangle feature

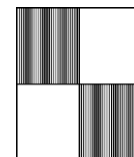


Figure 4: Four-rectangle feature

Although this may appear a time-consuming process with the counting of pixels involved, the speed is increased by using a technique called integral image. This is a store of the sums of pixels to the left of and above each region in the image, this store can then be used as a lookup table for efficiency.

The learning algorithm behind the training process is AdaBoost. This algorithm is used for the boosting of weak classification algorithms that gets an accuracy value of random or higher. The process of using Adaboost is to essentially produce a strong classifier from weak classifiers [30]. For Haar Cascading the main goal of the weak learning algorithm is to select the best feature frame that best separates the positive from the negative images. The following formula is how AdaBoost functions [31]:

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

Where:

- $h_t(x)$  is the weak classifier with outputs limited to -1 or +1
- $H(x) = \text{sign}(f(x))$  is the strong classifier
- $\alpha_t$  is the weight applied to classifier t by AdaBoost, better classifiers are assigned a higher weight. A negative weight is assigned to classifiers with an accuracy below 50%. An accuracy of 50% is given a value of 0.

The initial classifier is trained on an equal weighting which is calculated using [32]:

$$\text{weight}(x) = \frac{1}{\text{number of training occurrences}}$$

After training a weak classifier, the weak classifiers are given a higher weight so they can be used for the training of the next classifier to be improved upon. This is done by applying votes to the classifier based on the error results. Then after further iterations of boosting there is a guarantee of a lower loss for the training samples i.e. through the weak classification rule. The final output of the equation  $H(x)$  is a linear combination of the weak classifiers, where the decision is made based on the sign of the result. The benefits of using AdaBoost in Haar Cascading is that it can be used on most types of training data, it is fast and has no parameters to tune. The main disadvantage of using AdaBoost is that by feeding it classifiers that are too complex it may cause overfitting or vice versa for underfitting. Where overfitting relates to when the noise in the training data is picked up by the classifier during training and it begins to have a negative effect on the performance of the model. The negative effect occurs because in most cases new training data or testing data will not contain the same noise [33].

P. Viola and M. Jones [29] then proposed a cascade of classifiers, this was proposed to reduce classification time and would increase the detection results. Here a positive result returned from a classifier calls the second classifier and so on, in this case, the performance is boosted in each instance. If during this time there is some negative result then the negative sub-window is rejected. Haar Cascading using the cascading of classifiers was first proposed and used for face detection, here it performed very efficiently in comparison to other face detection systems.

The Haar Cascading algorithm has been successfully used by R. Budiman, et al [34] to locate white blood cells in images. To do this, two sets of training data were prepared, the first containing the object for detection i.e. the white blood cells, and the second containing other parts in blood images such as red blood cells, platelets, etc. Both sets of data were also converted to grayscale. Once the model was trained, a threshold was applied to the images to improve detection and the model was applied. Testing showed that overlapped cells caused about a 5% change in detection but overall the

precision value ranged between 74-95%. The results show that Haar classification should be useful for the detection of the Malaria parasite in erythrocytes, however, there are some concerns as the parasite is much smaller so some alterations to the training of such a model are required.

### 2.1.3 Neural Network Techniques

Another machine learning technique used in previous work is the use of an ANN. Neural networks are a supervised learning approach, here the data is labelled and separated into patterns, these patterns are stored as numerical vectors. Neural networks assist in the process of taking unlabelled data and sorting it into clusters of similarities. Most neural networks consist of several interconnected artificial neurons, placed in layers and are all interconnected. These layers are produced by feeding it labelled data to learn to distinguish the data.

During the training process, the neural network stores all of the features of the images and whether or not it predicted it correctly in that particular artificial neuron when incorrectly identifying an image while training it will make adjustments through the use of backpropagation as explained earlier in section 1.6. For image recognition a specific form of neural network is used, this is a CNN. This type of network consists of four layers, these are convolution, activation, pooling and fully connected [35]. These are explained for the use of TensorFlow as it can be easily integrated into Python for the use of this project.

### 3.2.3 i - Convolutional Layer

To understand the first layer the concept of convolution first needs to be understood. Convolution is a form of image processing that always returns a linear result. Convolution is possible with the use of a kernel which is often smaller than the size of the image and different values within it relate to different results on the image such as blurring, edge detection, etc. The process requires an image matrix, a kernel and an output matrix on which to place the calculated results.

The convolution on the image is calculated by flipping the kernel both horizontally and vertically, placing the kernel onto the image and then multiplying every element of the kernel with the overlapping element from the image.

The sum of all of the products is then taken and placed in the same position as the centre of the kernel in the output matrix [36]. A stride can be determined by which the number of pixels the kernel is moved over the image matrix at each time. This can be seen in figure 5 below where an edge detection kernel was used.

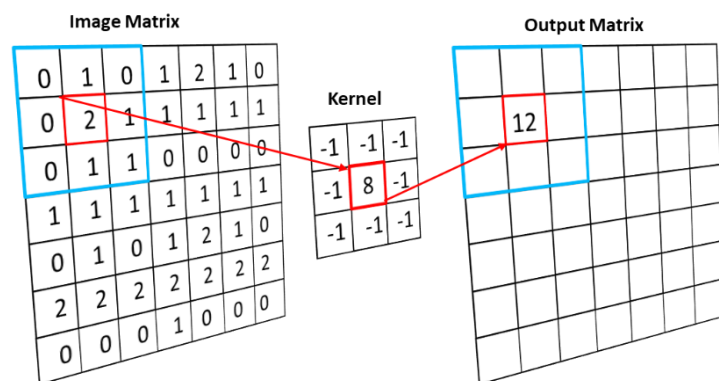


Figure 5: Convolution being calculated with edge detection kernel

In terms of the convolutional layer, in the training process, it can be specified to either use 'valid' or 'same' padding for the convolution of the image. Here 'same' applies padding around the border of the image matrix by adding zeros around the edges, in the case of the kernel overlapping during convolution. Whilst 'valid' is used for the kernel to remain within the image matrix, in what is called a valid position, here the output value will decrease by kernel size subtracted by one. In the convolutional layer, the pixels in each input image are reviewed in search for patterns, when a pattern is found it is stored in an artificial neuron within the layer, these become specific features of the input data.

### 3.1.3 ii - Activation Layer

The activation layer is used next, this layer takes the data and makes a decision whether or not to pass it to the next layer. The use of the activation layer assists in solving complex problems through its non-linearity which also allows backpropagation. There are many types of activation functions but the most commonly used ones are sigmoid, SoftMax and relu.

Sigmoid is the activation function with results returned between 0 and 1 with an S-shaped result which gives a new representation of the original data and hence provides non-linearity [37]. Here large negative numbers will become 0 and in the case of large numbers, they will be 1. The main issue with the sigmoid function is that once the function falls closer to 0 or 1 then the network is not learning anything as the value saturates.

Sigmoid is only able to be used for two classes but the SoftMax function which is similar can handle multiple classes. SoftMax takes the output value from sigmoid and divides it by the total number of outputs making the input probability relate to a particular class [38].

Relu is the most commonly used for neural networks, it can be defined as  $f(x) = \max(0, x)$  and is also non-linear [39]. The formula can be described as if the input value is less than 0 it is 0, but if above zero the value remains the same. The benefit of Relu is that it doesn't activate all artificial neurons at once which means the network becomes sparse and more efficient.

### 3.1.3 iii - Pooling Layer

The pooling layer essentially shrinks the data into a much more processable form. Maxpooling is commonly used which takes a kernel and a stride of the same length and then applies it to the input and outputs the maximum value in every area the kernel covers [40]. This can be seen in figure 6 below. The benefits of using the pooling layer (it is optional) are that it drastically reduces the computation requirements and it will restrict overfitting.

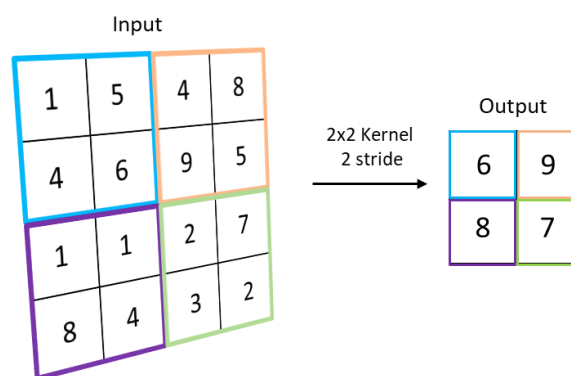


Figure 6: Maxpooling applied with 2x2 kernel and stride of 2

### 3.1.3 iv - Fully Connected Layer

The fully connected layer takes the output from the pooling layer as an input and outputs an N-dimensional vector where N is the number of classes. This layer examines the output from the pooling layer and decides based on the features that belong to each specific class in the network. This means that the artificial neurons in the pooling layer are 'fully connected' to the neurons in this layer. The SoftMax function is used on this layer for the multi-class choices and it ensures that the probabilities of the fully connected layer add up to 1 altogether [41].

Dropout is a function commonly implemented into the architecture of a neural network, its main use is to prevent overfitting. This function drops connections to some neurons in the previous layer to prevent overfitting. The use of dropout also ensures that the overall network structure is reduced. There are also other types of neural networks, these are feed-forward and feedback networks. Feed-forward neural networks are a single

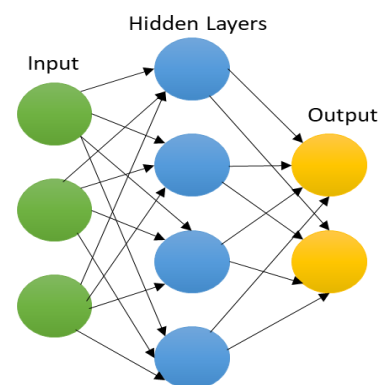


Figure 7: Neural Network Architecture

flow network, this means that it only allows the input to flow to the output and no backward travel is allowed. Feedback neural networks allow signals to travel backward and forwards [42].

The typical architecture of a neural network is shown in figure 7 to the right. Where the input relates to the training data, the hidden layers refer to those explained above i.e. the convolutional, activation, pooling and fully connected layer, and the output is the final trained result.

### *3.1.3 v - Neural Networks in Practise*

The use of neural networks for image processing tasks is becoming increasingly popular. One of which is by S. Krishnan, et al [21], where they used the visualisations of activations to improve the classification process. The interest in this article arose from the number of layers used in the neural network, here 13 layers were used to classify the cells to either infected or uninfected. The article focuses on visualising the features learned by the neural network at each layer, the first convolutional layer returned 20 features that consisted of colours and edges to feed to the deeper layers. When moving onto the second convolutional layer it was found that the network began to understand locations of the parasites and specific shapes within images. These can be classed as high-level features as they are much more complex than the initial features. The third convolutional layer combined the first and second convolutional layer feature set results which give more information on the colour and texture of a shape. The final fully connected layer, therefore, contains the combined abstractions of shapes, textures, and patterns making detection on test images simpler. In the activation layer, it showed that white pixels were more strongly activated at that location within the image.

The final article considered was by F. Ertam and G. Aydin [22], on the use of Tensorflow to classify an MNIST model. In this case, Tensorflow is an open source library created by Google for the use of machine learning. The MNIST model being used is a dataset containing a large number of handwritten digits, where the picture size is 28x28 for each of the digits. The performance of the MNIST dataset was checked by changing the activation functions applied to the model architecture. Relu returned the highest accuracy at 98% this was closely followed by SoftPlus at 97%. This article shows that finding the correct activation function can have a huge impact on the result, in the case of using the Sigmoid function the results were just 78% which is a significant amount lower than Relu and SoftPlus.

Overall, from the research undertaken, the use of neural networks is a very interesting research area especially if the use of various layers and types can alter results heavily. A simple change in a layer or activation function can make a huge impact on the result.

## *2.2 Methodology*

Although it is clear that there will be three proposed solutions produced within this project, the agile methodology will be used instead of the waterfall methodology. With the waterfall methodology, every section of the program would need to be planned out before producing the program but with some consideration into the type of project and the goal of attaining high accuracies some of the more detailed techniques used within the three programs will most likely need to be changed or altered during production. Due to the chance of change and the main goal of the project, this is why the agile methodology has been chosen for the project.

Agile in this case is a more flexible approach to how a project is planned and produced, the methodology consists of sprints which will be 2 weeks long for this project. Sprints are intervals in which work on the project is conducted before it is reviewed again for improvements/adaptions and at the end of each sprint, it is expected that the product can be deployed at that moment so it must always be in a working order. Following the agile approach, it is possible to change techniques used on the solutions if during the production it becomes clear that they are not beneficial to the final solution. There are also scrum meetings held in which ideas and solutions can be improved upon or



altered during the sprints, making this methodology very hands on which is useful for such a life critical piece of software. During the process a product backlog will be kept, this will contain all of the necessary features required in the final solutions and how they will be approached in order. During each sprint items from the backlog will be taken and implemented, at the end of the sprint this is reviewed and if implemented to a satisfactory standard it is removed from the backlog, if not it is returned to the backlog to be revisited at a later stage.

### 3. System Requirements

The software requirements vary for each of the three programs, each with their own manufacturer installing/running requirements. The software required can be seen in table 1 below. The production and testing of all three programs have been conducted using Ubuntu 16.04, all requirements assume that this operating system and specific version are being used. The requirements for each piece of software has been taken from the official website for each.

Software	Requirements
<b>Python 3.5.2</b>	Standard
<b>Cuda 9.0.176</b>	<ul style="list-style-type: none"> <li>GPU supported by Cuda, a list can be found on the GeForce website [43]</li> </ul>
<b>TensorFlow 1.5.0</b>	<ul style="list-style-type: none"> <li>Cuda 9.0.176</li> </ul>
<b>Keras 2.1.4</b>	<ul style="list-style-type: none"> <li>TensorFlow 1.5.0</li> </ul>

Table 1: Software required

There are no specific listed memory requirements but at least 500MB is recommended for the storing of trained models and data. The running and testing of all three programs is created in Python 3.5.2. Below in table 2, is a breakdown of the specific Python modules required.

Python Module	Sub-Module/s	Sub Module of Sub-Module
<b>cv2</b>	*	*
<b>numpy</b>	*	*
<b>os</b>	*	*
<b>prettytable</b>	PrettyTable	*
<b>pandas</b>	*	*
<b>skimage</b>	filters, measure, data, exposure	*
<b>itertools</b>	*	*
<b>sklearn</b>	cross_validation, utils,	<ul style="list-style-type: none"> <li><b>cross_validation</b> - train_test_split</li> <li><b>utils</b> - shuffle</li> </ul>
<b>tensorflow</b>	*	*
<b>keras</b>	backend, utils, models, layers.core, layers.convolutional, optimizers, preprocessing.image	<ul style="list-style-type: none"> <li><b>utils</b> – np_utils</li> <li><b>models</b> - Sequential, model_from_json, load_model</li> <li><b>layers.core</b> - Dense, Dropout, Activation, Flatten</li> <li><b>layers.convolutional</b> - Convolution2D, MaxPooling2D</li> <li><b>optimizers</b> – adam</li> <li><b>preprocessing.image</b> - ImageDataGenerator</li> </ul>

Table 2: Python modules required



## 4. Design

This section contains the information of the design for the three proposed solutions. This includes the way in which they function, reasons behind the specific design decisions and the specific output for each. The section is broken down into three subsections 'Image Processing Through Segmentation', 'Haar Cascading' and 'Neural Network', there is also an initial subsection that explains the main design decision of removing the background from the images.

All code can be found on GitLab at <https://cseegit.essex.ac.uk/MalariaProject> and has also been provided on a USB to ensure the larger files that could not be uploaded are submitted.

### 4.1 Understanding the Images Provided

The data used to produce the three proposed solutions has been split into two. There is a total of 100 images that are available for the entire project. The image set was halved so that 50 images are used for the training and development of the three solutions and the other 50 will be used for the testing of the solutions. There are a further 100 images that are identical to the initial 100 images which contain the ground truth labels. An example of the two images can be seen below in figure 8.

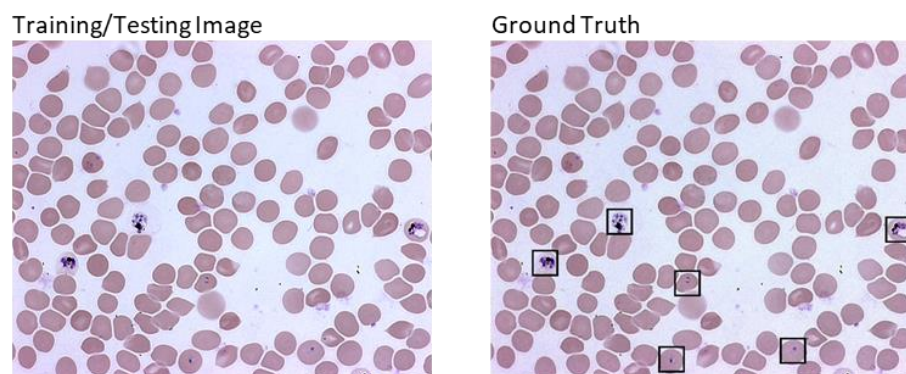


Figure 8: Example images to be used for project solutions

The images used have been taken from a microscopic view of a light microscope where the blood samples were first prepared by staining the blood smear with the Giemsa stain, this stain can be seen working in figure 8 where the infected cells in the ground truth image are a darker colour in comparison to the erythrocytes and background.

Upon inspection of the images, it became clear as to why the automatic detection of Malaria is more difficult than thought. The reason behind this is because of the number of defects in the blood such as platelets, artefacts and white blood cells that also get stained with the Giemsa stain when the blood smear is prepared. The background artefacts can be seen in figure 9 below. The solution for this, as it was in the articles found was to remove the background from the images, however, this needed to be done as carefully as possible to preserve the structure of the erythrocytes within the images.

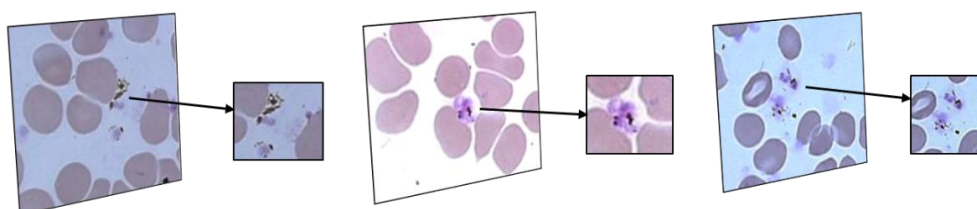


Figure 9: Background defects

There are three programs produced these are "Image Processing Through Segmentation", "Haar Cascading" and "Neural Network".

## 4.2 Program 1 - Image Processing Through Segmentation

The overview of program 1 can be seen in figure 10, each stage will be explained in order of the flowchart.

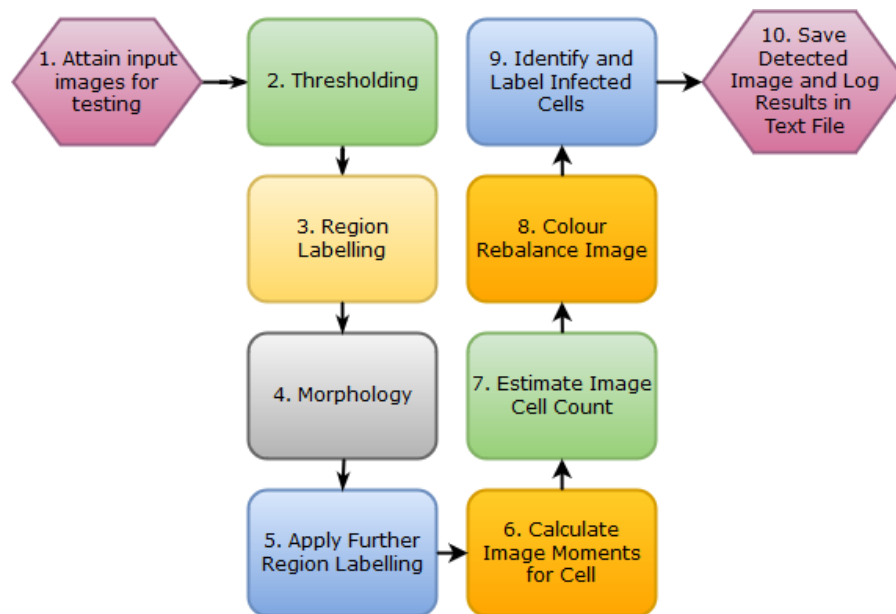


Figure 10: Flowchart showing each stage of program 1 to detect the infected cells in images

To produce an automatic Malaria detection program using image processing, the background needs to be removed. The way in which to initially approach this is to enhance the edges of the image to make the cells and defects in the blood more prominent, this also removed some noise in the image. To do this a bilateral filter is applied to the image which is then followed by applying the Laplacian operator to the bilateral image. The Laplacian result and the original image are then subtracted from each other. The process can be seen in figure 11.

The bilateral filter preserves the edges very well as well as removing some noise from the images. The Laplacian filter, on the other hand, works very well on images with reduced noise and it finds all of the grey level pixels within the images. This pairs well with finding darker sections like cell borders and parasites/defects in the images that have been stained with the Giemsa stain. The centre image of figure 11 is the Laplacian operator applied to the bilateral image, it shows the edges being detected. The final image to the right in figure 11 is the subtracted result of the original and Laplacian images, this result shows the cells and defects edges being more prominent than the original image.

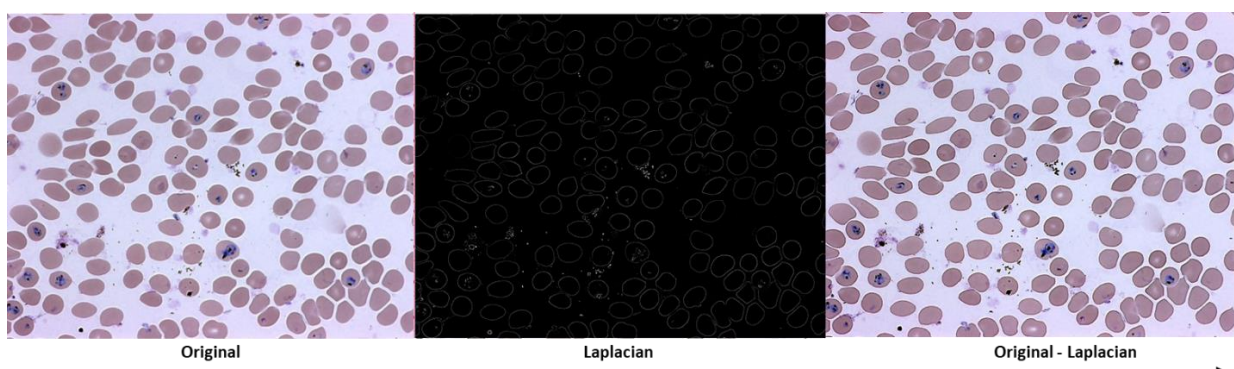


Figure 11: Original image left, Laplacian applied to bilateral image centre, original subtracted from Laplacian image right

The next step was adapted from the article by A. Patil and J. Shaikh [24] where the use of changing the images into the LAB colour space vastly improved the segmentation of the background from the foreground. The adaption was altered to use the LAB image split into its 3 channels and use the 'L' and 'A' channels together instead of only the 'B' channel. The CLAHE technique was applied to the 'L' and 'A' channels and it was then combined with the unaltered 'B' channel to form the complete image.

CLAHE is a form of histogram equalisation which adjusts the contrast and intensity of the image, but instead of simply applying it to the whole image CLAHE takes the image and divides it into 'tiles', the tiles are then independently equalised to get a better result [44]. This is a better approach to use as some pixels with higher/lower intensities can affect the result drastically.

The result can be seen in figure 12, this shows how much more prominent the cells and defects appear, drawing out foreground from the background in images that originally appear to have cells that blend into the background.

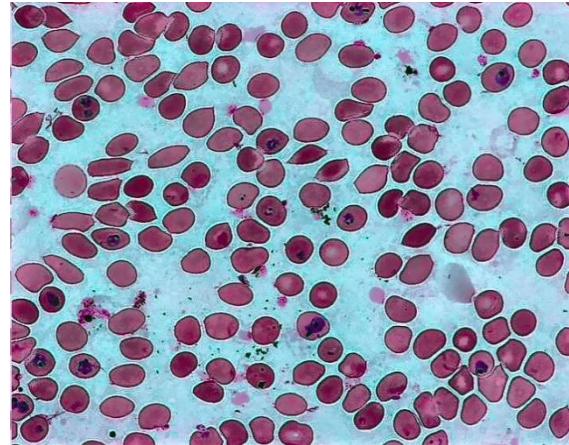


Figure 12: CLAHE applied to L and A channels of LAB image

#### 4.2.1 Thresholding

This section defines part 2 of figure 10. The technical aspects behind the removal of the background began at this stage, the use of a binary threshold was applied to the image based on the centre value of the image. This is similar to the Otsu thresholding technique but using Otsu on the images did not give as effective results, so it is used later on alongside this method. To find the centre value a histogram is created for each image with bins, the bin size is 256 so that the count of pixels for each specific value could be retrieved later on. The input image is then looped over to find the grey value pixels to be added to each bin, adding the values to the bin is completed by:

```
for i in range image shape y:
    for j in range image shape x:
        bin = grayscale value [i, j] / 256
        y [bin] = y [bin] + 1 #to increase peak size of bin value
```

Upon viewing the histograms produced the peaks and values were ragged, this made it difficult to find the two highest peaks within the histogram as each of these are treated as an individual peak. A solution implemented for this is by using a technique called histogram smoothing. Histogram smoothing is the process of taking the average value for each bin and use that value as the height. The result is the identification of the average line over the plotted histogram values, which ultimately removes the rough edges. Using the smoothed histogram, the two highest peaks within the histograms could then be identified. The process can be understood in the pseudo code below:

```
Loop through peak values to find highest peak:
    if peak value > current highest peak:
        highest peak = peak value

Loop through peak values to find second highest peak:
    if (peak value > current second highest value) and (peak value < highest peak value):
        second highest peak value = peak value
```

The centre value of the two highest peaks is then used as the centre value of the image and is marked as a yellow line on the histogram, this can be seen in figure 13 below.

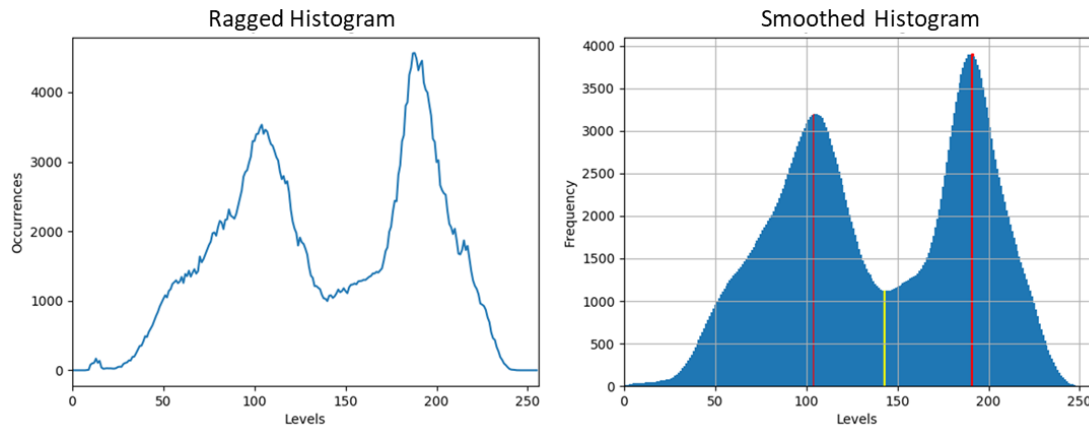


Figure 13: Ragged histogram to the left, to the right is the labelled smoothed histogram where red lines indicate the two highest peaks and the yellow line is the centre value of the histogram found by taking the centre of the two red lines

The retrieved centre value is then used as the thresholding value for the classification of pixels. The thresholding operator used is THRESH\_BINARY\_INV, here the image is inverted and any pixel values above the centre value are converted to 255 and if below the value it is converted to 0. This can be described as:

$$\text{gray value}(x,y) = \begin{cases} 255 & \text{when } > \text{centre value} \\ 0 & \text{when } < \text{centre value} \end{cases}$$

The result can be seen as the centre image in figure 14, where the background is black and has mostly been found except for some minor defects which will be removed later on.

To remove further defects, the centre value of the thresholded image is found and is used as the thresholding value for the classification of pixels using Otsu thresholding this time. The result can be seen in the right-hand side image of figure 14, where most defects have now been removed with only some isolated sub-pixels remaining. The only issue with applying the thresholding techniques to the images is that darker areas within the cells are caught as values below the centre value and are changed to the value 0. This can be seen in the centre and right image of figure 14 where there are small black sections of isolated pixels within some of the cells.

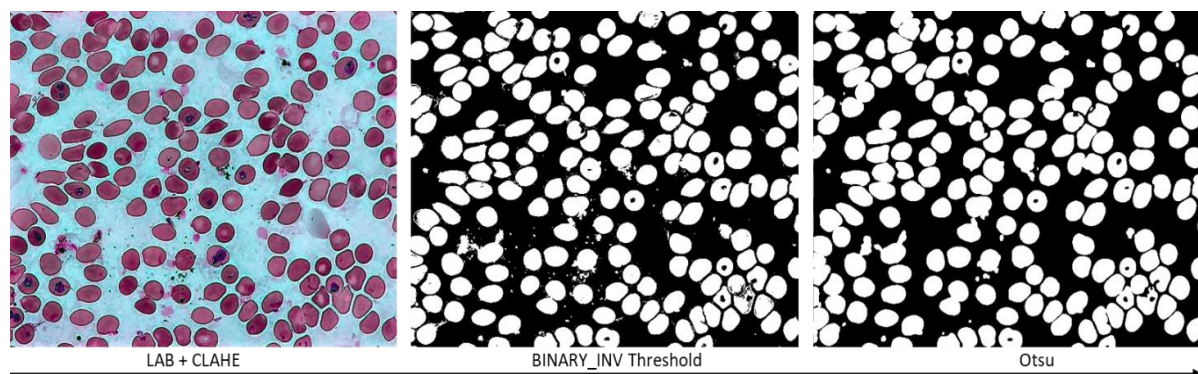


Figure 14: Left is the LAB + CLAHE image, to this the threshold (inverted) is applied, the result is the centre image, to the right is Otsu thresholding applied to the centre image.



#### 4.2.2 Region Labelling

This section defines part 3 of figure 10. To remove the isolated white pixels (defects) in the background, region labelling is used. The code on region labelling to find the isolated white pixels was adapted from the article by A. Rosebrock [45], here measure from the Python module Skimage is used to find the components within the images, the pixels of each object are then discovered and calculated. If a white section of pixels contains less than 350 pixels is it changed to black. The same process was used for the black isolated pixels within the cells, here the thresholded image was first inverted so the black pixels became white. Next, isolated sections of white pixels below the size of 150 are changed to black to match the colour of the cell. The final result can be seen in figure 15 to the right.

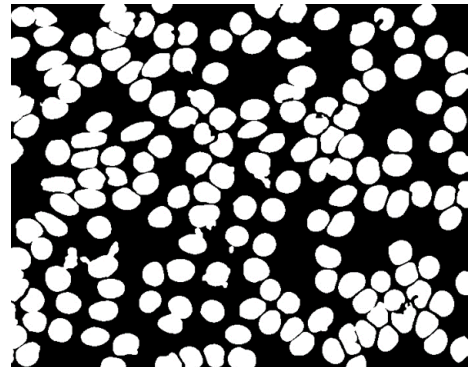


Figure 15: Thresholded image with isolated white and black pixels removed

#### 4.2.3 Morphology

Moving into part 4 of figure 10, the removal of further defects still attached to cells need to be removed, for this, morphological operators are used.

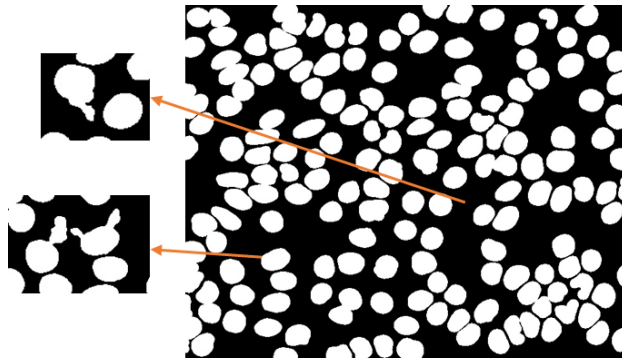


Figure 16: Showing how defects are removed from the cells using MORPH\_ELLIPSE

To preserve the rounded shape of the cells the structuring element used on the morphological operator applied is MORPH\_ELLIPSE, here it preserves elliptic shapes found. The MORPH\_OPEN operator (erosion followed by a dilation) was then applied to the images. Applying this breaks off some of the attached defects from the cells.

Region labelling is then applied to the image again to remove the isolated white pixels broken off from the defects. The result can be seen in figure 16 where within the figure to the left, can be seen how the cell previously appeared and how it appears after the morphology process can be seen in the complete image to the right. The thresholded image is then restored back to the original image, here only the white sections were kept and the black sections were discarded. This can be seen in figure 17 below.

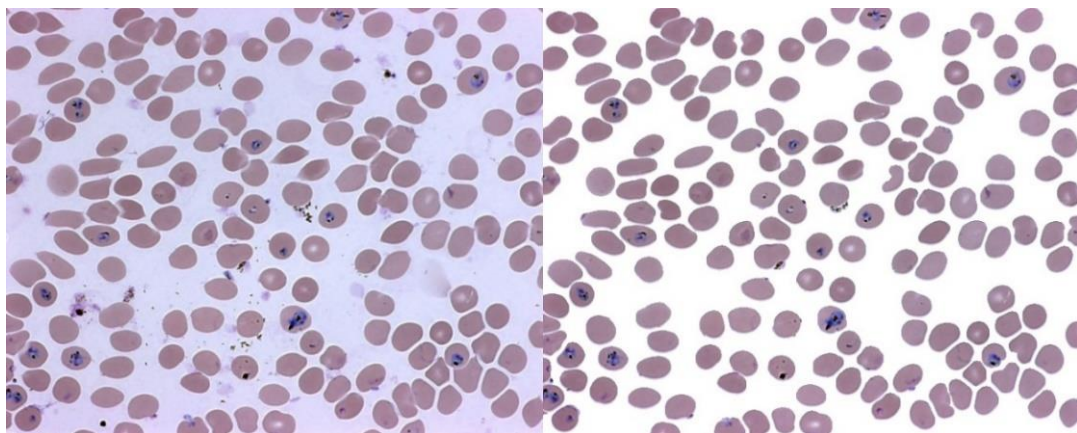


Figure 17: Original to the left, to the right is the background and defect removed image

#### 4.2.4 Counting the Number of Cells

To count the number of cells in the images, the connected components are found within the image i.e. the cells, which are returned as a label. Using all the labels returned the area of each label is found and added to an array. The average of all the areas found is then calculated for each image which is considered as the **average cell size** in each image. The areas are then sorted by frequency and created a histogram based on the results, on the x-axis is the area value and the y-axis contains the frequency count. The value with the highest frequency is found on the histogram to separate lower values from higher values and is marked with a pink line as seen in figure 18. It is assumed that:

**values < segmentation line = single cells**  
**values > segmentation line = overlapped cells**

**if values > segmentation line:**  
    **value = value/average cell size**  
    **if value < 0.5: round down value**  
    **if value > 0.5: round up value**

This returned an accurate cell count for each image with a +/-3 difference in some cases. The cell count will allow the user to understand the severity of the infection overall.

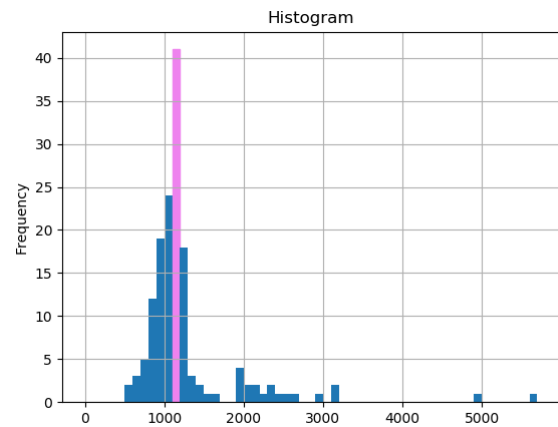


Figure 18: Histogram plotting, x axis – areas, y axis- frequency, the pink line represents the separating value

#### 4.2.5 Centre Points

This section defines part 6 of figure 10. The process begins by applying a technique to all of the cells called flood filling. This is applied to the cells to ensure there are no further isolated pixels that could cause detection issues later on. The initial implementation of the detection process did not work as intended as the process was confused by overlapping cells within the images. To solve the overlapping issue all of the cells needed to be segmented, to do this all areas above the size of 5 (avoids any isolated pixels that may exist) in the image was added to an initial cluster array. The initial cluster array contained all of the cells both overlapped and single, to separate them values at or below the average cell size were kept as single cells and stored in a final cell array. For the overlapped cells, the areas for each are divided by the average cell size. With the value returned:

**value < 2: return to final cell array**  
**value > 2: add to the final cell array**

A height map was created to ensure the correct cells were being retrieved, an example can be seen in figure 19 of two joined cells. The centre x and y positions are found for values above the size of 2. To do this a modification of the code available on the OpenCV website on contour features [46] was used. To find the x and y values:

**Loop over cells in final cell array:**  
    **Apply contouring to cells**  
**Loop over single contours in contouring applied:**  
    **Calculate image moments for each single contour**

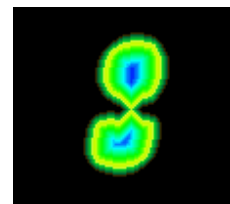


Figure 19: Height map of joined cells

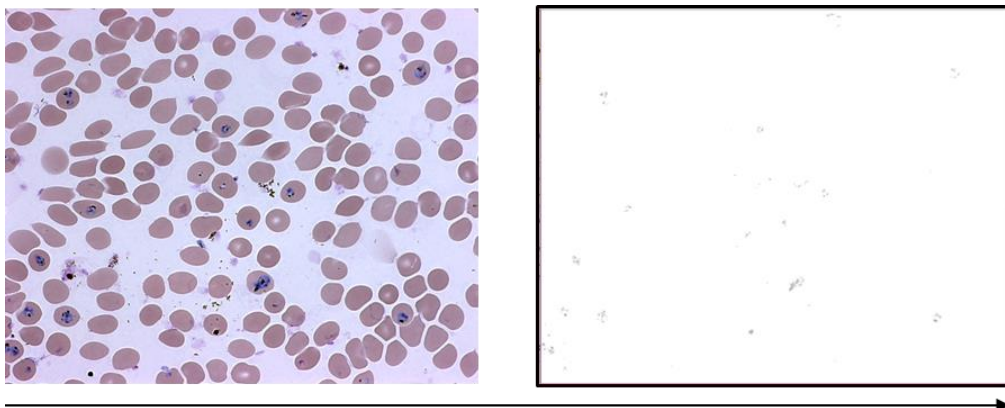
Using the image moments calculated, the x and y centre values were found and stored. This was also computed for the joined cells as each were added separately to the final cell array previously. Using the x and y values retrieved the minimum enclosing circle of the cells is also found so they can be circled later on if detected as infected.

#### 4.2.6 Colour Rebalancing and Finding Infected Cells

This section explains parts 8 and 9 of figure 10. To find the infected sections within cells, the restored image after the removal of background is used. In this process, the images colour values are rebalanced to remove the green and blue values and reduce the red. This will leave the parasites within the cells and defects left behind visible because of the Giemsa stain making them a darker colour. This is shown as:

```
if blue channel: image [blue channel] = 255  
if green channel: image [green channel] = 255  
if red channel value >= 255: image[red] = 255  
else: image[red] = red * (2.3 + 50/255)
```

The red value formula was created through the process of trial and error but works well. The image is then thresholded to retrieve the blue-black parasites and possible defects in the cells, the result can be seen in figure 20 with the original compared to the colour altered image.



*Figure 20: Left is the original image, right is the colour rebalanced image showing how the parasites are found in the process of colour alteration.*

The next step is to find the colour rebalanced blobs shown in figure 20. This is done by identifying the objects in the image and adding them to an array. To ensure the correct objects are being identified morphological operations are applied to the images, this ensures that small sections of isolated pixels are removed. The initial process of detecting infected cells did not go as planned where the brightness of the images is different for each, this affected the detection process heavily. To solve this the images were broken down into 5 ranges. The ranges were determined by the white pixel count in the image (the image was thresholded with a binary invert and the parasites became white and the background black).

The decision behind this can be seen in figure 21, the original images that are darker have some of their cell borders caught in the colour rebalance process. The lighter images, however, work very well with the colour rebalance process as seen in the centre image of figure 21.

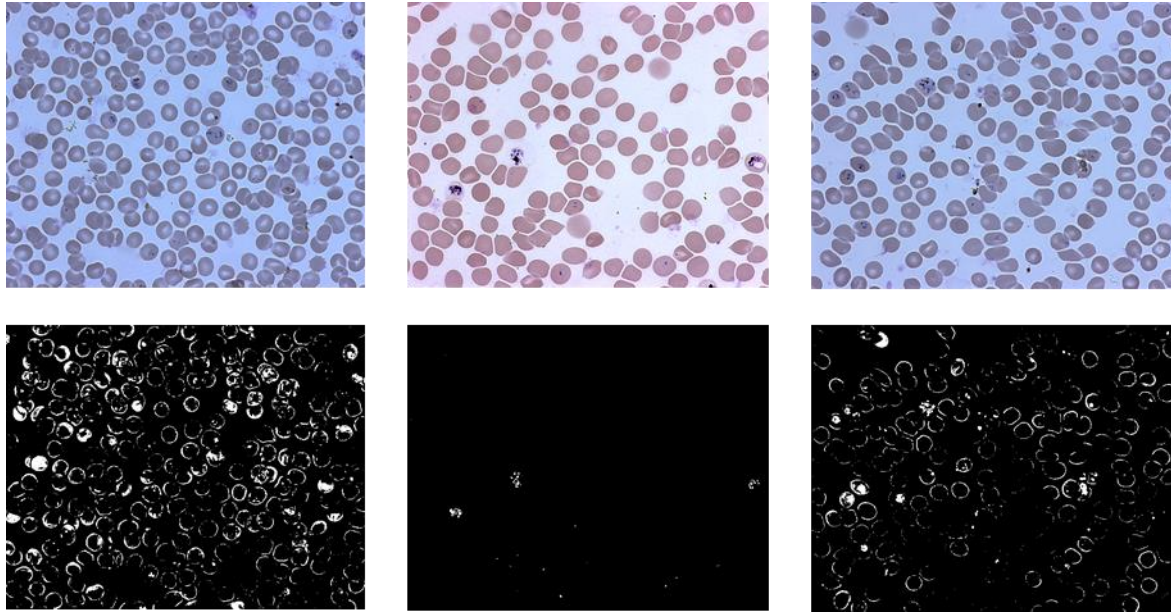


Figure 21: Original images in row 1, their colour rebalanced image can be seen directly below in row 2 showing how the darker the image, the whiter the pixels in the image.

Based on the white pixel values and a visual examination of the range of brightness in the images the five ranges are:

- white pixels > 45,000
- white pixels from 25,000 up to and including 45,000
- white pixels from 10,000 up to and including 25,000
- white pixels from 5,500 up to and including 10,000
- white pixels from 1,000 up to and including 5,500

For each range of white pixel count values, the image is treated differently with morphological operators to achieve the desired result of finding infected cells. This can be seen in appendix 1 for easier visualisation, once the morphological operators are applied to the images, connected component labelling is applied to remove isolated pixels below the value of 5 (this is small defects).

Once the images have been processed as desired, each cluster of objects within the images is iterated through, if the object is found in the initial cluster count (where cells were found and separated into the final cell array) and the morphologically processed image then it is added as an infection count. If the infection count is above the size of 4 (determined through trial and error testing) then the cluster is added to the infected array. This ensures that no small infected sections are added to the array.

The infected array is then iterated through and for each object, a minimum enclosing circle (surrounding the whole cell) is drawn around the cell in red, this can be seen in figure 22. The image returned is the original image where the background removed one is only used in internal processing.



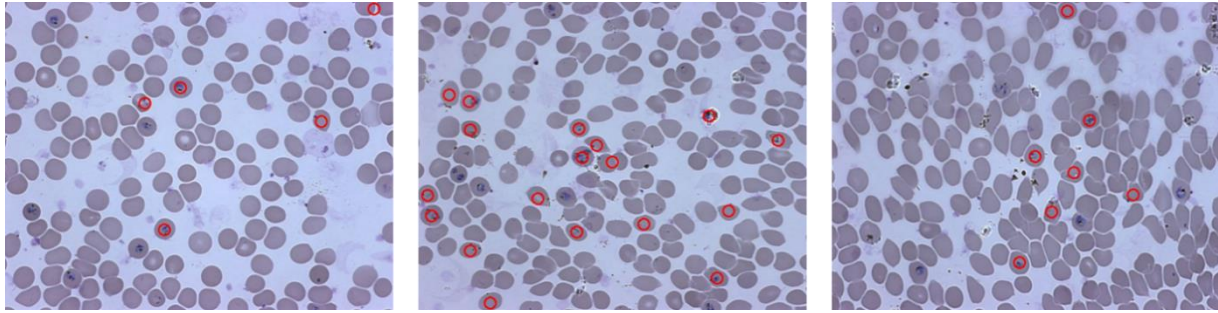


Figure 22: Final detected images returned, shown on the original input image where infected cells detected are circled in red.

Upon the initial running of the program, the user will be asked for the folder containing the input images and the folder in which they wish to save the detected and labelled images. There is error handling added to the code, here if the folder is incorrect it will inform the user to try again.

Once the images have been processed the information is stored in a table for each image, where the file name, number of infected cells, average cell count and whether or not the image sample is infected are added. The table is added to the text file "Image\_Processing\_Detection\_Result.txt", an example can be seen in Appendix 2. The detected image is also saved at each stage.

#### 4.3 Program 2 - Haar Cascading

This section explains how the Haar classification method was implemented and how it has been programmed to detect whether or not a cell is infected or uninfected. Haar Cascading is a form of machine learning, the model used for the classification of new images needs to first be created. The first section will explain how the model used was created. The overall architecture can be seen in figure 23.

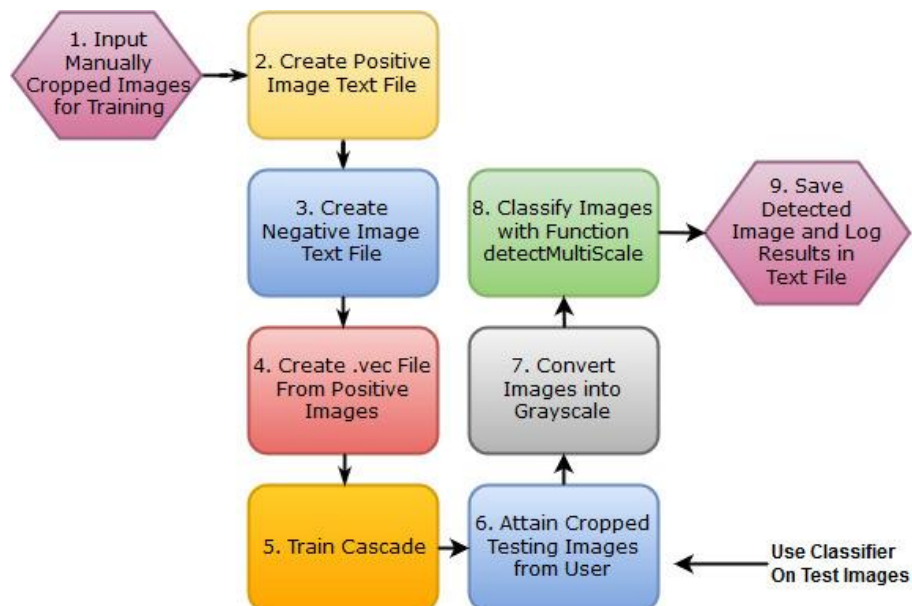


Figure 23: Flowchart showing overall architecture of the Haar Cascade training and use

##### 4.3.1 Training the Classifier

Starting at pat 1 of figure 23, the training of the classifier requires input images. The images required are the two sets for it to be able to determine whether or not the input image, later on, is infected or uninfected. During the training and detection stage, the images were taken and manually cropped

into their individual cells/cell subsections. This fits the requirements of the Haar Classifier process to ensure that correct object is being used in all images for training.

#### 4.3.2 Positive Image Set

The positive image set contains the infected images, for this process, the image must only be of the object to be detected by the classifier. If there are other objects within the image to be detected the classifier will become confused during the training process and it will not detect as it should. To attain only the parasite, the infected cells were manually cropped out of 40 images (the remaining 10 were used for testing later on before the final test). By manually cropping the infected cells from the images, it ensured that there were no defects and a minimal amount of background was caught in the corner of the image. A small sample of the cropped cell images added to the positive image set can be seen in figure 24. The images are converted to grayscale to remove noise. The width and height of each image must also be the same for all images, this ensures that the parasites are trained to be found for their exact size when comparing features. The height and width used are 30x30 for all of the positive images. Once collected the images are stored in a folder for the training stage, the total count of positive images collected is 317.

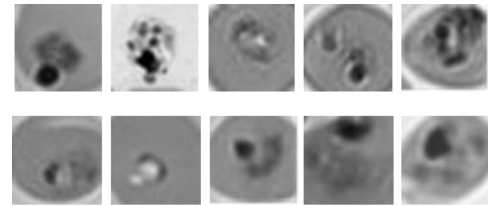


Figure 24: Positive image sample set, manually cropped to attain minimal background and only the parasite (object of interest).

#### 4.3.3 Negative Image Set

The negative image set (uninfected cell images) needs to be prepared with images that vary in size and contain most of the background. Within the images provided, there are a lot more uninfected cells than infected cells. This was beneficial for training purposes, as with more uninfected cells to show the classifier the more likely it would learn how to separate the cells. The uninfected images were also manually cropped to ensure that different sizes could be taken from the images, overlapped cells could be chosen to train the classifier that the different edges are not an infection and single cells to show that not all single cells are infected. A small set of the uninfected cell images used for training can be seen in figure 25, the images are once again converted to grayscale to remove excess noise. The total count of negative images collected and used is 1931.

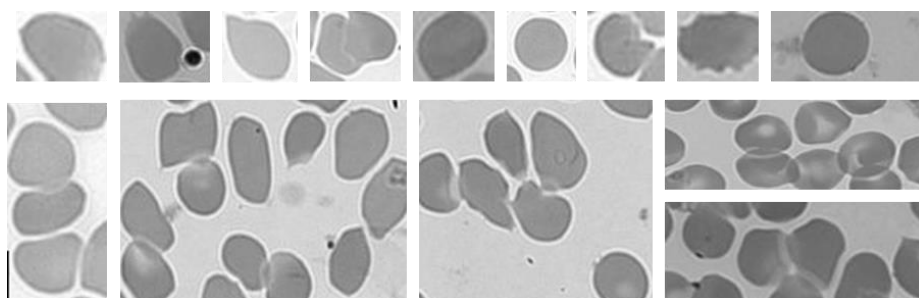


Figure 25: Set of manually cropped uninfected cells used for training

#### 4.3.4 Feeding Images to the Classifier

With both image sets prepared, parts 2 and 3 of figure 23 need to be completed. For the classifier to find and use the negative images it needs to be informed of their locations and names. To do this a text file was produced containing the filenames, the directory structure is:

```
/training
  1grey.png
  2grey.png
```

3grey.png  
...  
negCells.txt

The file negCells.txt is the text file containing all of the image names and it contains:

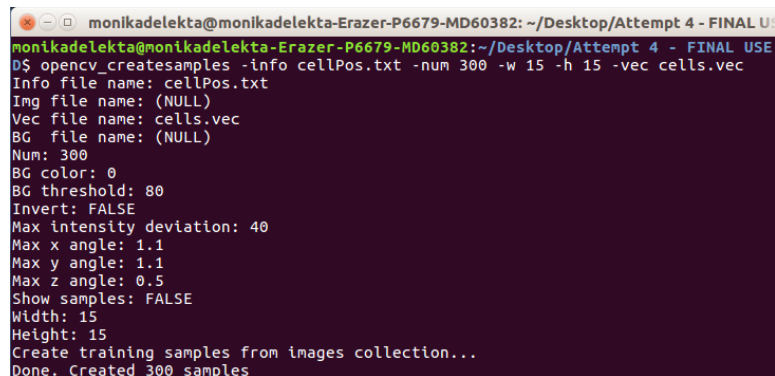
1grey.png  
2grey.png  
3grey.png  
...

The positive images are stored in the same directory as the negative images, however, the text file for these images is different. The structure for each file named within the text file is required to contain: **file name, number of objects in the image, location of the object in the image (x and y coordinates), image width and image height**. As the positive images contain only the single infected cell cropped out, the object location is the entire image, the value (0, 0) is provided for the x and y coordinates so the classifier begins processing the image from the top left corner. The text file cellPos.txt was created to contain this information, some example information is:

1grayPos.png 1 0 0 30 30  
2grayPos.png 1 0 0 30 30  
3grayPos.png 1 0 0 30 30  
...

#### 4.3.5 The Vector File

This section defines part 4 of the flowchart in figure 23. Using the positive training images and the text file cellPos.txt, a vector file is created. The vector file is created so that it is compatible with the training stage and it contains all of the images required for training. To create the .vec file the command “**opencv\_createsamples -info cellPos.txt -num 300 -w 15 -h 15 -vec cells.vec**” is entered into the Ubuntu terminal. Where the **-info** is the text file containing the positive image information, **num** is the number of positive images to add to the vec file (300 is provided as in previous training approaches the exact value got stuck in an endless loop). The width (**-w**) and height (**-h**) value were halved too for this process, this is because previous training results did not return a good detection result if it remained 30x30, as the process enlarges the images which makes them blurry and noisy. The final **-vec** command provides the name for the output vec file. The result of the command entered into the command line can be seen in figure 26.



```
monikadelekta@monikadelekta-Erazer-P6679-MD60382: ~/Desktop/Attempt 4 - FINAL USE
monikadelekta@monikadelekta-Erazer-P6679-MD60382:~/Desktop/Attempt 4 - FINAL USE
$ opencv_createsamples -info cellPos.txt -num 300 -w 15 -h 15 -vec cells.vec
Info file name: cellPos.txt
Img file name: (NULL)
Vec file name: cells.vec
BG file name: (NULL)
Num: 300
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 15
Height: 15
Create training samples from images collection...
Done. Created 300 samples
```

Figure 26: Entering the opencv\_createsamples command and the output of the process checking the 300 images

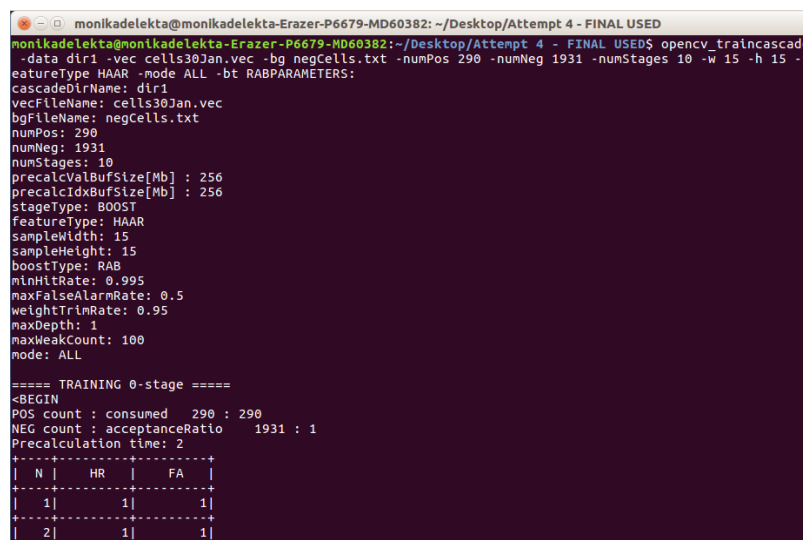
#### 4.3.6 Training the Haar Cascade

Next is part 5 of the flowchart in figure 23, where the files will be put together to train the classifier. To do this the command opencv\_traincascade was used. This command takes the negative and positive images and trains the boosted cascade of weak classifiers. The command issued was, “**opencv\_traincascade -data dir1\_30Jan -vec cells30Jan.vec -bg negCells.txt -numPos 290 -numNeg 1931 -numStages 10 -w 15 -h 15 -featureType HAAR -mode ALL -bt RAB**”. The argument being entered into the command line and some of its output can be seen in figure 26 below. Here -vec

accepts the positive .vec file, -bg the negative text file, -numPos the number of positive images to be used (which is less again as it remains in an endless loop otherwise) and -numNeg the number of negative images. Some of the more complex arguments can be explained as:

- **numStages** – the number of cascade stages to be trained, the training process automatically stopped at 10 in all instances of training as there is not a lot of data for it to use. The cascade stops at stage 9 (starting from 0) where it states the process is terminated as the required leaf false alarm rate is achieved, this means the cascade has reached the desired potential based on the arguments provided
- **featureType** – this can either be HAAR or LBP (local binary pattern). LBP is used as a texture operator which calculates the texture in the image by comparing each pixel to its neighbouring pixels. LBP is generally faster but less accurate than the HAAR feature type. The HAAR process uses the original integral image and rectangular feature representation. The decision to use HAAR is because it is more accurate and because there are not so many images to process this doesn't noticeably slow down the process.
- **mode** – BASIC or ALL can be used here. BASIC doesn't alter the input images and uses them as they are. ALL will use the input images as they are and create a new set of each image where they are rotated by 45 degrees. Using ALL the image sets are doubled, as there are not many images this is why ALL is used, it utilises the images available to get the best result.
- **bt** – this is the type of boosting classifier to be used, commonly used are Discrete AdaBoost (DAB), Real AdaBoost (RAB) and LogitBoost (LB)[47]. RAB uses predicted class probabilities and works well on categorical data. RAB is, therefore, useful to distinguish between infected and uninfected cells. RAB was also tested against the other boosting classifiers and it returned the best result.

When the command is executed within the command line the process will show the chosen attributes for each value and it will also show other attributes that are used as their default value, this can be seen in figure 27. The default values can be further understood in Appendix 3.



```

monikadelekta@monikadelekta-Eraser-P6679-MD60382: ~/Desktop/Attempt 4 - FINAL USED
monikadelekta@monikadelekta-Eraser-P6679-MD60382:~/Desktop/Attempt 4 - FINAL USED$ opencv_traincascade
-data dir1 -vec cells30Jan.vec -bg negCells.txt -numPos 290 -numNeg 1931 -numStages 10 -w 15 -h 15 -f
eatureType HAAR -mode ALL -bt RABPARAMETERS:
cascadeDirName: dir1
vecFileName: cells30Jan.vec
bgFileName: negCells.txt
numPos: 290
numNeg: 1931
numStages: 10
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: HAAR
sampleWidth: 15
sampleHeight: 15
boostType: RAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed 290 : 290
NEG count : acceptanceRatio 1931 : 1
Precalculation time: 2
+-----+
| N | HR | FA |
+-----+
| 1 | 1 | 1 |
+-----+
| 2 | 1 | 1 |
+-----+

```

Figure 27: Entering the opencv\_traincascade command into the command line and the output it returns

The training then follows this process, the training log can be found in Appendix 4. Once the training has been completed, there are 10 XML files for each classification stage found in the directory, these are named stage0.xml to stage9.xml. There is also a file named params.xml created, this contains all the information on the parameters used in the executing command. The final file is the classifier and

is named cascade.xml, this is the combination of all the stage0.xml to stage9.xml files and is used to classify test images.

#### 4.3.7 Classifying the Images

To apply the classifier to new input images it must be called in a python script to apply it to images. As the classifier was trained using cropped images of cells the input images also need to be cropped. The manual approach taken to crop the images for training cannot be adapted in this approach as it would be too time-consuming and would not assist in solving the issue of saving money to pay employees. To solve this issue a cell cropping tool was created.

#### 4.3.8 Cropping Tool

The images to be used for classification have their background removed to remove any risk of false positives. The process of removing the background from input images is identical to the process used in the image processing through segmentation program as explained in sections 5.2.1-5.2.3.

Canny Edge Detection is then applied to the image to find the edges of the image. The process involves:

1. Pre-process image to remove noise with Gaussian Blur.
2. Calculate gradients magnitudes. If high then there is an edge to be detected.
3. Calculate maximum suppression, if a pixel is not a maximum value it is suppressed based on the gradient direction.
4. Apply threshold to make the found edges visible.

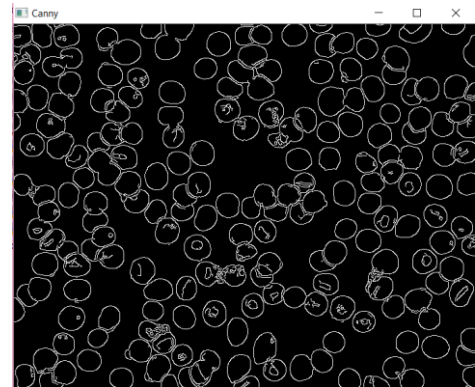


Figure 28: Canny Edge Detection applied to the image in the cropping tool

An example image of the result of Canny Edge Detection can be seen in figure 28. Contouring is then applied to the image, to find only external contours so any parasites or defects found within the cells do not have a contour applied to it. The remaining processes completed by the cropping tool is then:

1. Find a bounding rectangle for each contour, this returns x and y coordinates used to crop the cells.
2. Crop the cells individually from the images using the bounding rectangles if the bounding rectangle is between the range of 26 and 65. Values below 26 were found to be defects caught and above 65 were repeated sections of bounding rectangles enclosing other bounding rectangles.

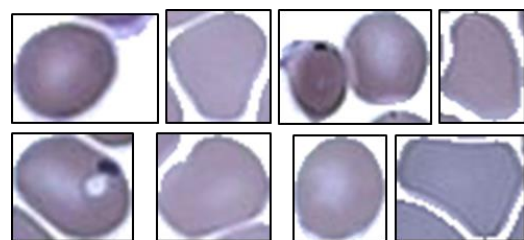


Figure 29: Example output of cropped images from the cropping tool.

The final results of some cropped cells can be seen in figure 29, not all cropped images are the same size but this is still acceptable for the classification process.



#### 4.3.9 Detecting with the Trained Haar Cascade

Using the cropped images, they are first converted into grayscale to remove excess noise (part 7 of figure 23) and then the detection process can begin as seen in part 8 in figure 23. The classifier is applied to the images through the function 'detectMultiScale'. This function takes three arguments, the first is the cropped image, the second is a scale factor value and the third is the minimum neighbours value. The scale factor value specifies how much each image is decreased in size at each image scale. The scale factor produces a pyramid of the image in the various sizes until it reaches 0 from its original size. The scale factor provided is 3.60, this means the image is reduced by 60% each time. The minimum neighbours value follows a sliding window approach, the windows are created into an image pyramid from the scale factor value. The minimum neighbours value used is 4, which means that each candidate rectangle must have 4 neighbours to keep it as a detected object otherwise it will be removed. The image pyramid can be seen in figure 30.

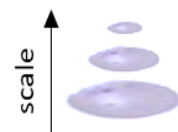


Figure 30: Image pyramid created from scale factor

Once the rectangles have been determined within the image a red rectangle is drawn around the object. The detected results can be seen in figure 31 below. When the classifier is tested on full images that have not been cropped there are many more false positives which means the difference in a complete image to a single cropped cell causes too much noise and confuses the classifier.



Figure 31: Haar Cascade classification results, red square indicates an infected cell has been found.

The Haar Cascade program takes an initial input of a folder containing all the images to be detected. Once the classifier has processed and detected all the images in the directory provided, the text file "Haar\_Results\_Detection.txt" is created. The text file contains a table with all of the image names and "True" to indicate if the image is infected or "False" if uninfected. An example file used on the test data can be seen in Appendix 5. The detected images are also saved to a folder.

#### 4.4 Program 3 – Convolutional Neural Network

This section explains how the CNN was as well as how the model can be used to test images. The first section explains the architecture and training process and the second section explains how it detects images. The entire process can be seen in the flowchart in figure 32, it includes a breakdown of each layer in the model too.

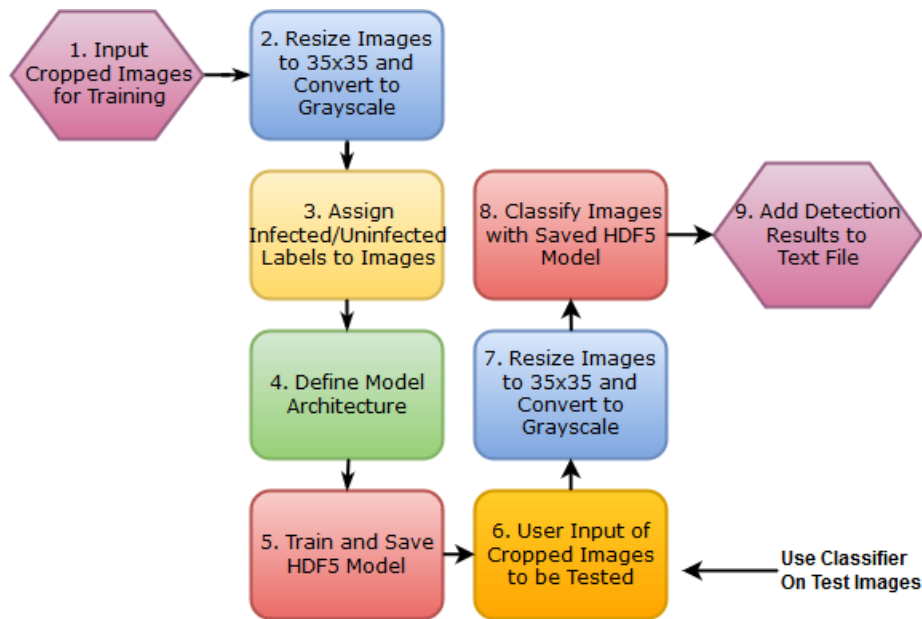


Figure 32: The training process of the CNN explained, this shows the layers used in the model architecture and is then followed by the classification of the saved model on new images.

#### 4.4.1 Training Process

The training of the neural network was completed using Tensorflow and Keras in Python. Initially, there were difficulties with getting the model created and detecting any images of the cells. The fourth attempt of creating a model is the one that worked as intended, the reason the others failed is because the images fed into the network were resized to the incorrect size and too many convolutional layers were added to the architecture. This was only really understood when looking at the CNN for MNIST. MNIST uses a very similar sized image as an input and it also has few but distinguishable features. The decision was therefore made to adapt the layers used in the network architecture from the MNIST code [48]. Using the adapted code, the CNN started returning much better results.

##### 4.4.1 i Image Preparation with The Use of a Cropping Tool

The preparation of the images was the same as for the Haar Cascade program, however, for the CNN the images of single cells were not manually cropped. The cropping tool (explained in section 4.3.8 of the Haar Cascade program) was altered to use the ground truth images provided and find infected and uninfected cells at the same time. The way in which this was programmed was:

1. Ask for user input of a directory containing the ground truth images
2. Apply a mask to the ground truth images to search within a range of [0,0,0] and [50,50,50] to find only the black squares labelling infected cells
3. Apply contouring to the mask created from the ranges and find the bounding rectangle for each contour (it was found all single squares were either 41 or 42 in width and height, for overlapping squares the sizes varied but the ranges found were broken down into those shown in Table 3 where each range for the number of squares covers various orientations in which the squares can be positioned

Range	Number of Squares
(width >= 41) & (width <= 47)) & (height >= 41) & (height <= 55)	1
(width >= 44) & (width <= 47)) & (height >= 70) & (height <= 90)	2
(width >= 48) & (width < 75)) & (height >= 40) & (height <= 90)	2
(width >= 75) & (width < 95)) & (height >= 40) & (height < 80)	2

(width >= 70) & (width <= 140)) & (height >= 80) & (height <= 140)	3
(width >= 100) & (width <= 140)) & (height >= 50) & (height < 85)	3

**Table 3:** Ranges showing how the cells in the cropping tool were found from ground truth squares where height is the height of the single or overlapped multiple squares and the width is the width of the single or overlapped multiple squares. The number of squares is the number of squares that fit within the width and height range, 1 is a single square, 2 is two squares next to or on top of each other and 3 is three squares next to or on top of each other.

4. Remove background from unlabelled images (directory input asked for from the user) as it was in sections 5.2.1-5.2.3
5. Based on the bounding rectangle ranges found, crop out the infected cells from the unlabelled image using the x and y coordinates from the masks bounding rectangle, e.g. a range matching 3 squares will be split into 3 separate cells. There were 470 images cropped.
6. Save cropped infected cells into an 'infected' directory for the neural network to find
7. Crop the uninfected cells, with an exception to not crop out the infected cells x and y coordinates found and cropped earlier
8. Store cropped uninfected cells in an 'uninfected' directory (2,000 images cropped)

Previous training iterations showed that using all of the uninfected images made the model lean more towards the uninfected class as it unbalanced the model. To resolve this instead of using all the 2000 images of uninfected cells the ratio was made 1:1 for both sets. The uninfected directory had images deleted from it randomly until 470 remained. Having 1:1 ratio of images allowed the model to compare much better and didn't cause overfitting as it did with the large number of images in one set previously. Once prepared the images are then converted into grayscale to remove excess noise and resized to 35x35. For the images to be accepted by Tensorflow the images need to be expanded in their image shape to also include their channel value, for these images it is 1 as grayscale only has one channel. For this numpy is used to expand the dimensions of the shape from (Height, Width, Length) to (Height, Width, Length, Channels).

#### 4.4.1 ii Assigning Labels

Moving into part 3 of figure 32, the two sets of images have labels assigned to them. The label 0 was assigned to uninfected cells and the label 1 was assigned to infected cells. This was input into the neural network using numpy categorical which converts a class vector to a matrix. For the two channels, it can be viewed as:

- [1, 0] – uninfected
- [0, 1] - infected

The images are then shuffled randomly to feed to the neural network. As there is not a lot of data to be used a small sample of the random set is split off as a validation set for the training process. The value of 10% was given, so 10% of the sample is retained for testing, this is chosen randomly.

#### 4.4.1 iii Model Architecture

Part 4 of figure 32 is to define the model architecture. The model architecture adapted from MNIST does not have as many layers as S. Krishnan, et al [21] did because with too many layers the data began to overfit and results became less accurate. A sequential model is then defined which is a linear stack of layers, this means that layers can be added iteratively making it useful to define your own architecture. The layers added can be seen in order in table 4:

Layer	Arguments
<b>Convolution2D</b>	Applies 32 3x3 layers, border_name same is used indicating to add zeros around the borders of the images and the input shape of the images is provided too (number of samples and number of channels, rows, and columns)



<b>Activation</b>	Relu activation function
<b>Convolution2D</b>	Applies 64 3x3 filters to find further features
<b>Activation</b>	Relu activation function
<b>MaxPooling2D</b>	Applies max pooling with a 2x2 filter
<b>Dropout</b>	0.25 provided means that there is a 0.25 probability any element can be dropped from the training process if it doesn't meet requirements
<b>Flatten</b>	No argument, this stage converts the 2D arrays into a single linear vector, this allows the use of fully connected layers and combines features found from the previous two convolution2D layers.
<b>Dense</b>	128 provided, connects 128 neurons to each neuron in the next layer to preserve and build on features
<b>Activation</b>	Relu activation function
<b>Dropout</b>	0.5 provided, this will randomly remove 0.5 neurons in a layer to remove useless features and speed up the process
<b>Dense</b>	Classes provided (value of 2), this connects to the features to determine between 2 classes in the end result
<b>Activation</b>	SoftMax activation function, which handles the two input classes from the dense layer and output is between 0 and 1 meaning it can be used for predictions

**Table 4: CNN architecture implemented**

With the model architecture defined the learning process is configured with a loss function, an optimiser and a list of metrics. The loss function used was categorical cross entropy, the decision behind this was because categorical cross entropy is used for measuring the performance of a model which gives an output in the range of 0 and 1. Categorical cross entropy is also ideal for the use on multiple class classifications from which each class such as infected and uninfected used are their own class. The optimiser chosen is one commonly used with categorical cross entropy, this optimiser is adam. Adam is a requires little memory to run and is useful noisy images, it functions by updating the weights of the neurons based on the training data. This makes it ideal for the model being trained as it will adapt to the various types of infected cells to be found. Finally, the metric used is accuracy, this is used to evaluate the accuracy of the model, the results aren't used for the training but contains the result of the model based on training data. Accuracy relates to how accurate the model is at predicting.

#### 4.4.1 iv Training

With the model defined part 5 of figure 32 is entered. For this program, the training stage was completed in 2 different ways with 2 sub-sections for each. This allowed for a much better evaluation later on as to which way is the best to train a neural network. Tow using augmentation and two not using augmentation. Where augmentation is the process of processing the images in multiple ways which include rotations, flipping, zooming, and shearing. For each of these various epoch values were used to find the optimal classifier, the epoch values used on each were 15, 20 and 30. Epoch can be explained as one forward and one backward pass of the entire training set to update the training weights and resolve errors/misclassifications. The 4 methods can be described as:

1. Augmented with images containing background/defects
2. Augmented with images containing no background/defects
3. No augmentation applied to images containing background/defects
4. No augmentation applied to images containing no background/defects

For both sets of augmented and non-augmented models, there are two types of images used, one with background remaining in images and one without. This is to test the difference between exposing the CNN to the true images compared to a filtered version. To attain the images with background the cropping tool was simply altered not to use the filtering process prior to the cropping stage.

#### 4.4.1 iv.1 Augmented Applied

During training the images are set to be rotated by 40 degrees randomly, the width and height is altered by 0.1 randomly, the image is randomly applied shear transformations which changes the direction of the image according to parallel lines by 0.2, the image is randomly zoomed in by 0.2 and finally the image is randomly horizontally flipped. By doing this it allows the CNN to generalise the data well as it is fed different images each time and hence increases the size of the image set used. This is completed using the images with background and those without.

#### 4.4.1 iv.2 No Augmentation Applied

To create the background and no background models, each of the image sets (infected and uninfected containing background and infected and uninfected containing no background), are taken and used as they are for training, there is no further alteration to the images. Once the model has been trained for both they are saved as an HDF5 file containing the weights and arguments used to use on new test images at a later stage.

#### 4.4.1 v Resulting Models

There were 3 models for each by the end, as each model for each epoch was saved as well for evaluation later on. Once trained the models returned a test lost and accuracy to review how well the training process went. The results for each can be seen below:

1. Augmented with images containing background/defects
  - Epoch 15 - **Test Loss: 0.21293318969138125, Test accuracy: 0.8936170225447797**
  - Epoch 20 - **Test Loss: 0.221515159023569, Test accuracy: 0.9042553204171201**
  - Epoch 30 - **Test Loss: 0.3003111247052538, Test accuracy: 0.8510638221781305**
2. Augmented with images containing no background/defects
  - Epoch 15 - **Test Loss: 0.1063092699710359, Test accuracy: 0.9787233953780317**
  - Epoch 20 - **Test Loss: 0.041285176067910295, Test accuracy: 0.9893617021276596**
  - Epoch 30 - **Test Loss: 0.07555973498111075, Test accuracy: 0.9787233953780317**
3. Not augmented with images containing background/defects
  - Epoch 15 - **Test Loss: 0.3971500488671851, Test accuracy: 0.8936170149356761**
  - Epoch 20 - **Test Loss: 0.245126743900015, Test accuracy: 0.9468085119064819**
  - Epoch 30 - **Test Loss: 0.2899247794709307, Test accuracy: 0.9680850975056912**
4. Not augmented with images containing no background/defects
  - Epoch 15 - **Test Loss: 0.02149725657085234, Test accuracy: 0.9787234042553191**
  - Epoch 20 - **Test Loss: 0.06371551149703086, Test accuracy: 0.9787233953780317**
  - Epoch 30 - **Test Loss: 0.10862144653467422, Test accuracy: 0.9574468097788222**

As seen above there are various accuracies returned that are very similar but they will all return different results on images. An example training image can be seen in figure 33. The learning process can be visualised in the activation layer, this is the 2<sup>nd</sup> convolutional layer with an activation of Relu and can be seen in figure 34. The images all show in stages the features learned, here it is clear that the features found are the cell borders and the infected parasite. This means the model will only look for parasites within cell borders.



Figure 33: Test Image

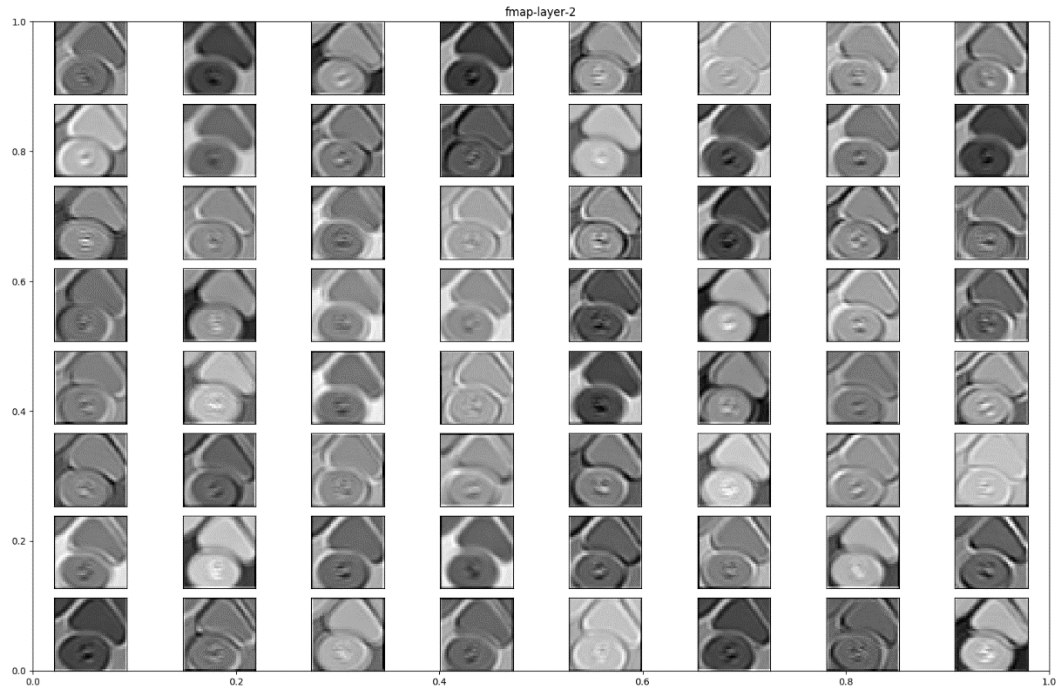


Figure 34: Map showing the features learned in layer 2

#### 4.4.2 Detection Phase

Applying the classifier to test images begins at part 6 of figure 32. The following process is used for all of the HDF5 models trained and if the HDF5 model trained is for images with no background the test images have their background removed beforehand.

Using the saved HDF5 model on new training images was completed with code written in Python that also implements Tensorflow and Keras. The detection process can be described as:

1. Ask user for a directory containing the input images (randomly mixed with infected and uninfected images that have been cropped with the cropping tool), an error message is returned if the directory cannot be found
2. Resize images to 35x35 to fit the model
3. Convert the images to grayscale
4. Expand image array to add the channel value as explained in section 4.4.2
5. Import HDF5 model and use it to predict input images

The HDF5 model will return an array containing two values (one for infected and the other uninfected) between the values of 0 and 1. The array can be visualised as **[uninfected, infected]**. The closer to 1 the more the model assumes it belongs to that class. All of the values are calculated for each image using (where model [0][0] == uninfected and model [0][1] == infected):

- if model [0][0] > model [0][1] then uninfected
- if model [0][0] < model [0][1] then infected

Once all of the values have been calculated they are added to a text file, an example of an output text file can be seen in Appendix 6.

## 5. Performance Measure Tools

This section explains the process taken to extract and evaluate the performance variables of each of the three programs explained above. The first section explains the production of the code to extract

the data and is then followed by the results and a discussion of them. The values measured (accuracy, precision, recall, and f-measure) on each program are to understand how many of the images it correctly identifies as infected or uninfected, to provide the most accurate diagnosis possible.

### 5.1 Producing the Performance Measure Evaluation Programs

For each of the three programs an evaluation tool was created, the tools all use the same measures to find a specific result but all have a different meaning. The evaluation process involves taking the image detected by the program and comparing the result to the ground truth to find the precision, recall, accuracy and f-measure values. The evaluation tools produced are all different as each program returns a different value to mark an infection on an image. For instance, program 1 takes a complete image and detects it, so it will also return a complete image which means the tool will need to find the detections within the entire image. Programs 2 and 3 are similar to each other as they both take images of cropped cells created by the cropping tool and compute their detections on these. The Haar Cascade returns a value of 0 if uninfected or 1 or more if infected and the neural network returns an array with two values and the higher value will mark the image as infected or uninfected (explained in section 4.4.2).

Precision is a value relating to the number of relevant images detected out of the total relevant images. It is calculated using:

$$precision = \frac{TP}{TP + FP}$$

Recall is a value relating to the number of relevant images detected out of all the images retrieved. This can be calculated using:

$$recall = \frac{TP}{TP + FN}$$

Accuracy is a value that describes how well the program is at detecting the Malarial parasite within images, it accounts for all errors which lower the value. Accuracy can be calculated as:

$$accuracy = \frac{TP + TN}{TN + TP + FN + FP}$$

Precision, recall, and accuracy all have a value between 0 and 1, the close to 1 the stronger the result. F-measure is used to find the mean of both precision and recall into one result. To calculate this the formula below is used:

$$f - measure = 2 \frac{Precision * Recall}{Precision + Recall}$$

F-measure shows how robust and how precise the program is, if only looking at precision and recall some values that are more difficult to detect that make a classifier stronger may be missed if only looking at precision, f-measure accounts for this to determine if the classifier detects as many difficult images as possible as well as the easier images.

### 5.2 Program 1 - Image Processing Through Segmentation

The evaluation tool created for the image processing through segmentation method was more complex than the others as it had to detect all of the detections on a full image instead of on a cropped single cell image. The evaluation process is as follows:

1. Apply a range to the image to find the red detection circles returned from program 1

2. Labelling squares on the ground truth images are found by applying a range to find the black squares as they were in section 4.4.1, any small defects of the same colour caught are removed with region labelling
3. Contouring is applied to the ground truth mask to find the bounding rectangles
4. The ranges in Table 5 are applied to find the single, double and triple overlapped labelled squares in the ground truth images. The double and triple ranges have more than one range as the squares could be placed vertically or horizontally making their width and height values change. As rectangle sizes fall within ranges the number of squares they relate to are counted to find the total number of ground truth squares in each image

Range	Number of Squares
(width >= 41) & (width <= 47) & (height >= 41) & (height <= 55)	1
(width >= 44) & (width <= 47) & (height >= 70) & (height <= 90)	2
(width >= 48) & (width < 75) & (height >= 40) & (height <= 90)	2
(width >= 75) & (width < 95) & (height >= 40) & (height < 80)	2
(width >= 80) & (width <= 140) & (height >= 80) & (height <= 140)	3
(width >= 70) & (width <= 75) & (height >= 90) & (height <= 100)	3
(width >= 100) & (width <= 140) & (height >= 50) & (height < 85)	3

Table 5: Ranges used to find ground truth squares where height is the height of the single or overlapped multiple squares and the width is the width of the single or overlapped multiple squares. The number of squares is the number of squares that fit within the width and height range, 1 is a single square, 2 is two squares next to or on top of each other and 3 is three squares next to or on top of each other.

5. Bitwise OR the detected image from program 1 and the ground truth image masks to created one complete image, this can be seen in figure 35
6. Apply the contouring technique RETR\_TREE, this retrieves all contours from the image and creates a hierarchy tree so internal and external contours can be found. By using RETR\_TREE it returns an array for each contour, the array consists of [Next, Previous, First Child, Parent].
7. Find and store the bounding rectangles of all the contours

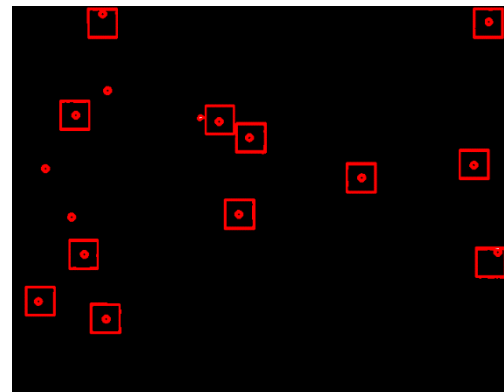


Figure 35: Mask of detected image and mask of ground truth image combined with binary OR.

Finding the TP, FP, TN and FN values:

1. Find all FP values by searching for bounding rectangles with a width and height of either 4 or 5 for some or 7 or 8 for others, this depends on some rectangles overlapping during detection. If found the 'Parent' value of the hierarchy is searched. Here it would be hierarchy[3], FP values can be determined as: **If hierarchy[3] == -1: false positive += 1**
2. To find the TP values the parent and first child values are evaluated. The bounding rectangle range is between a height of 85 and width of 34, this was chosen to remove small contours that are not the ground truth squares and 85 was chosen randomly as no contours from testing have exceeded the value of 70, this is simply a safety barrier for future cases. To find the true positives the value of the first child and parent are found for each contour through iteration. True positives can be determined by:  
**if first child and parent == -1: not a true positive #no contour within ground truth**  
**else: true positive += 1**
3. FN values are found using: **total number of ground truth squares – total true positives found**

4. The TN value was left as zero as an estimation of the cell count is not as accurate as desired and the same result is returned when using zero compared to another.

For each image in the test directory the TP, FP, TN and FN values are added to a table along with its filename. The table is output into a text file called 'ImProc\_Results.txt'. The accuracy, precision, recall and f-measure values are calculated using the TP, FP, TN and FN values for each image. These are also added to the text file in a new table, alongside their specific image filename. Once all the images have been iterated over and their values calculated, an average of the precision, recall, accuracy, and f-measure is calculated and added to the bottom of the table for an understanding of how the program performs on average. This table is then added to the text file, the file can be seen in Appendix 7.

### 5.3 Program 2 - Haar Cascade

The evaluation tool for Haar Cascading is different to the one for program 1 as it must be evaluated with cropped images of cells instead of complete images. The cropped images are extracted from the images using the same cropping tool used to train the CNN in section 4.4.1 i. It is used because it uses the ground truth images. This is essential in the evaluation process to ensure the program can be tested correctly against the correct values. The evaluation tool outputs the cropped images into separate infected and uninfected directories to ensure the images are separated. The Haar Cascade evaluation tool works by:

1. Asking the user for a directory input, the directory must contain two subdirectories, these are 'infected' and 'uninfected'. These should contain the cropped images with their background removed
2. Convert images into grayscale
3. Scale factor value of 3.60 and minimum neighbours value of 4 is applied to the cascade classifier to detect infected images
4. If the classifier returns 0 the cell is detected as uninfected and if the value is greater than 0 it is infected. The values are compared to the folder the image came from i.e. either infected or uninfected. The rules used to determine TP, FP, TN or FN are:
  - If the classifier does not detect an infection and the ground truth is **uninfected** then it is a **TN**
  - If the classifier detects the image as infected and the ground truth is **infected** then it is a **TP**
  - If the classifier detects the image as infected and the ground truth is **uninfected** then it is an **FP**
  - If the classifier does not detect an infection and the ground truth is **infected** then it is an **FN**
5. Using the TN, FN, TP and FP values the precision, recall, f-measure, and accuracy values are calculated
6. Add the TN, FN, TP and FP values to a table
7. Add the precision, recall, f-measure and accuracy values to a table

The two tables are then added to the text file 'Haar\_Results.txt'. The text file created can be found in Appendix 8.

### 5.4 Program 3 - Convolutional Neural Network

The evaluation tool for the CNN performs in a similar way to the Haar Cascade evaluation tool. This evaluation tool also requires a directory with two subdirectories containing 'infected' and 'uninfected' images of cropped cells generated from the cropping tool. The cropping tool was altered in this case to crop images with their background included too (the filtering process was simply removed) so that



this could be tested against those with their background removed. This left two pairs of infected and uninfected cropped images, one with background and one without.

The calculation of TP, FP, TN and FN values is completed using the values that the CNN classifier returns within an array as mentioned in section 4.4.2 of the CNN detection phase. To rules below are used:

- If ground truth is **uninfected** and the model array value is larger for uninfected then it is a **TN**
- If ground truth is **infected** and the model array value is larger for infected then it is a **TP**
- If ground truth is **infected** and the model array value is larger for uninfected then it is an **FN**
- If ground truth is **uninfected** and the model array value is larger for infected then it is an **FP**

Once calculated the values are used to calculate the precision, recall, accuracy, and f-measure. The process above is completed twice, the first is with cropped cells containing their background and the second is with cropped cells containing no background.

The TP, FP, TN, and FN values are added to a table and added to the text file 'NN\_Results\_With\_Background.txt' for the cropped cells with background and 'NN\_Results\_No\_Background.txt' for the cropped cells without background. The accuracy, precision, recall and f-measure values are then added to another table and appended to the text file. Both text files can be seen in Appendix 9.

## 6. Test Results

Below in Table 6 are the results for each program, only the highest background and no-background classifiers for augmented and non-augmented have been added as the rest are irrelevant. For the CNN classifier trained with images containing their background, there were two classifiers returning the same accuracy so they were both added to the table.

Program	Background	Classifier	Augmented	Precision	Recall	F-Measure	Accuracy
Image Processing	N	Segmentation	N	0.33	0.37	28%	20.47%
Haar Cascading	N	cascade.xml	Y	0.83	0.75	79%	91.13%
CNN	N	CNN_No_Background_EPOCH20.hdf5	N	0.84	0.94	89%	95%
CNN	N	CNN_No_Background_EPOCH20.hdf5	Y	0.78	0.94	85%	93%
CNN	Y	CNN_with_background_EPOCH20.hdf5	N	0.78	0.91	84%	92%
CNN	Y	CNN_with_background_EPOCH30.hdf5	N	0.77	0.92	84%	92%
CNN	Y	CNN_with_background_EPOCH20.hdf5	Y	0.68	0.95	80%	88%

Table 6: Results from testing each classification program, the background is N if there is no background and Y if the background was not removed. Augmented is the process of manipulating the images to attain a larger image set through flipping, rotation, zoom, and shearing it is Y if it was applied and N if not applied.

## 7. Evaluating Test Results

As stated earlier the reason behind the evaluation process is to evaluate how well each of the programs are at correctly determining if an image contains infected or uninfected cells. The number

of images it correctly identifies adds to how accurate the program is at correctly diagnosing a case of Malaria in a blood sample. The results in Table 6 are presented in four different values, these are precision, recall, f-measure, and accuracy. The reason for this is because using just one metric may not evaluate a program as well as it should where it may miss important characteristics necessary that could be captured by another metric.

Accuracy accounts for the total number of correct images found out of total number of images in the set. This means that accuracy accounts for the number of correct predictions made out of the data provided. Precision relates to the number of relevant images detected out of the total relevant images. When the precision is high (closer to 1) then there is an indication of a low FP rate. Recall is a value relating to the number of relevant images detected out of all the images retrieved. If the value of recall is high (closer to 1) then it means that out of all the positive images the majority of them have been found by the algorithm, there is, therefore, a low FN rate. F-measure is the combination of precision and recall into a harmonic mean, taking FP and FN values into consideration.

### 7.1 Program 1 – Image Processing Through Segmentation

For program 1, the accuracy returned was 20.47%, this accuracy was calculated using the average accuracy for all the 50 images within the test set, this was because it was the only program able to detect on the full-sized image. The accuracy of 20.47% is very low which means that there is less than one-fourth of a chance that the program will detect the image correctly for infected cells. The F-measure value has also been calculated as an average f-measure value of the 50 images in the test set, the result is higher than the value of accuracy at 28%, pushing it just above a one in four chance of correctly determining if a cell is infected or uninfected in the image on average.

The low percentages may be surprising as some of the images returned quite a high accuracy and f-measure score individually. For example, as seen in the 'Results Calculated' section within Appendix 7, image 061.jpg returned an accuracy of 77.78% and an f-measure score of 88%. On the opposite scale image 0.82.jpg returned an accuracy percentage of 2.78% and an f-measure score of 5%. This means that the results varied so much, extremely low values simply brought down the final average percentage results for accuracy and f-measure. There is also a lot more low results in comparison to a small number of high results, this means that extreme lows cannot be balanced by extreme highs as they are overpowered by the number of low values.

#### 7.1.1 Why Program 1 Return Such Low Results

Program 1 has by far returned the lowest results for both accuracy and f-measure. A further evaluation of this shows why. Looking at figure 36 it shows image 070.jpg which within the results table of Appendix 7 it has an accuracy of 60% and an f-measure of 75%. This is quite high for this program and is one of the images contributing to the higher values of the final result. Taking a closer look at the image it is clear that it is quite a bright image and the cells aren't dark either making the parasites clearly darker than the rest of the image. This

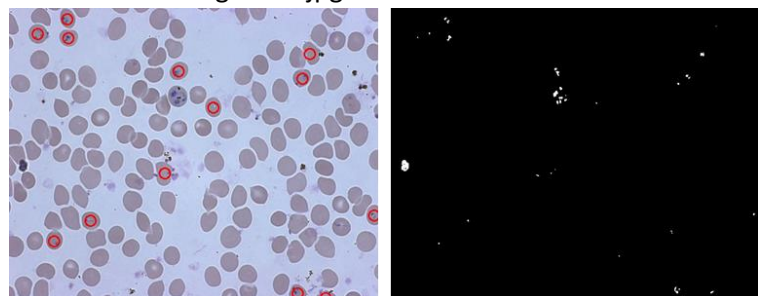


Figure 36: Image 070.jpg, very light, detected image to the left and to the right its colour rebalance to find the parasites within it

shows in the black and white image to the right of figure 36 which shows only the possible parasites in white. Some of the additional cells marked as infected do in fact contain small amounts of defects but the FP rate of cells containing no defect or infection is very low in comparison to other images.



Figure 37 below shows the opposite spectrum, the colour balance of the images seen as the black and white images show a lot more white sections than in figure 36, where the cells within the images are being caught too. This is because the cells are much darker than those in figure 36 which means they blend in a lot more with the stained parasites and defects. It can be seen here that in all the images the dark blue-black parasites/defects are missed and other cells are marked as infected instead, this is because these images confuse the program. Image 1 of figure 37 contains image 084.jpg which returned an accuracy of 7.41% and f-measure of 14%. Image 2 shows image 082.jpg which returned an accuracy of 2.78% and f-measure of 5%. Finally, image 3 contains image 075.jpg which returned an accuracy of 5% and an f-measure of 10%. This explains the reason as to why the first program does not perform as well as it should, during the time of producing the program there were no contrast alterations applied to the image nor was there any differing of the colour rebalancing values for darker images. It was discovered as an issue during the production of the code, however, the only change made to accommodate this was to alter the morphological operations applied to the black and white image to remove as many white sections as possible before the detection process.

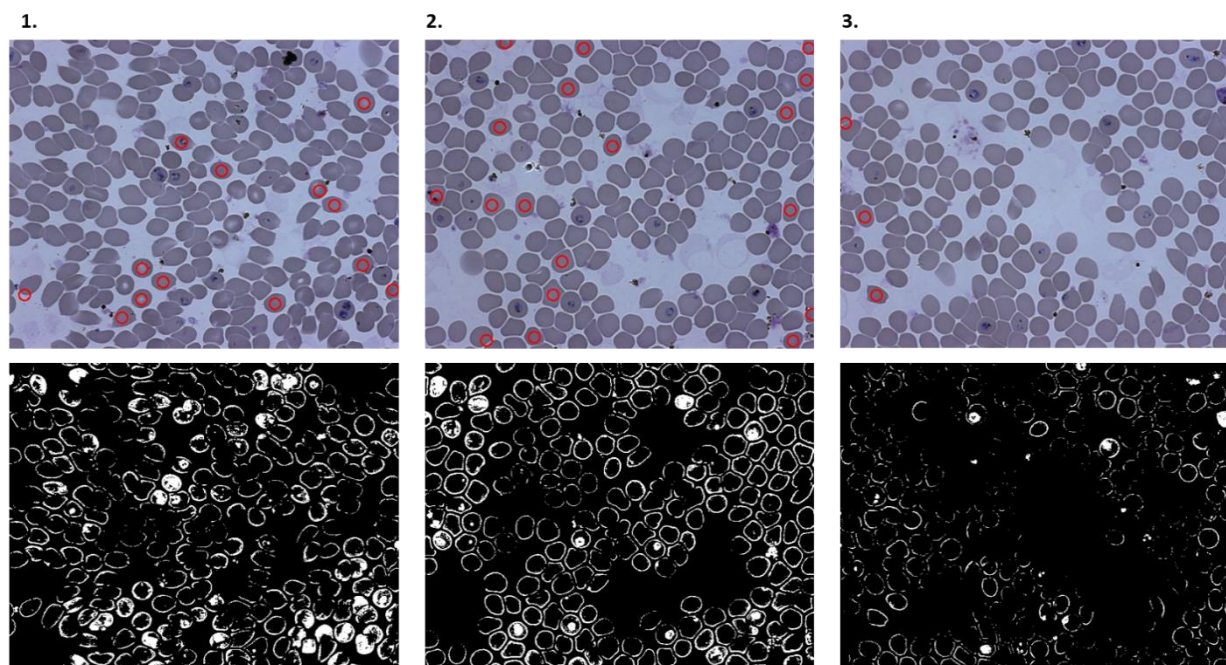


Figure 37: 1 is image 084.jpg, 2 is image 082.jpg and 3 is image 075.jpg. Showing how the darkness of the image affects white pixel count in colour rebalance.

Due to the low result, program 1 will no longer be competing as an option for a possible future distribution as it will require several alterations and is much slower taking about 40 minutes to detect 50 images in comparison to the Haar Cascade and CNN.

## 7.2 Program 2 – Haar Cascade

For the Haar Cascade, the testing was completed using the cropped images of cells with their backgrounds removed. The Haar Cascade program returned the second highest accuracy just after the CNN, the accuracy returned was 91.13%. This is a high accuracy meaning that out of all cases the program would correctly identify a cropped image of a cell fed to it 91.13% of the time. Surprisingly the f-measure result is 12.13% lower than the accuracy result at 79%. This means that on average the precision and recall values accounted for the FP and FN values more so than the accuracy value did.

### 7.3 Program 3 – Convolutional Neural Network

For all the classifiers trained for the CNN shown within Table 6, applying augmentation to the images has a small effect of about 1% on the recall values between them, but in the case of the precision results applying augmentation lowers the precision value by just under 10% in all cases. Applying augmentation to the images, therefore, had a negative effect on the f-measure result because of the precision used to calculate this, it also showed in the accuracy result where with augmentation the result was  $\pm 3\%$  lower than images that were not augmented. The classifiers with augmentation applied are therefore discounted from further evaluation.

This leaves comparing the classifiers tested on images containing their background, to those containing no background. The result is as expected and the classifier (epoch 20) tested on images containing no background returned the higher f-measure and accuracy. The f-measure score returned is 89%, the highest so far out of all three programs and it is also 5% higher than the classifier tested on images with their background. In comparison to the Haar Cascade classifier, the f-measure result is 10% higher for the neural network.

The accuracy returned for the classifier tested on images with no background is 4% higher than that of the classifier tested on images with their background at 95%, which is also the highest result of the three programs created. Although the results are close for background and no background image classifiers, there is still a difference pushing the removed background classifier test result to the top. This is most likely because some defects are caught in the images that still contain their background, whereas if the background is removed they cannot be detected. The difference between the f-measure and accuracy is 5% where the accuracy is higher.

### 7.4 Accuracy Vs. F-measure

From the evaluation of the CNN, it has shown a 5% increased difference in accuracy in comparison to the f-measure. For the Haar Cascade, there is also a 12.13% difference where accuracy outperforms the f-measure result. The decision between whether accuracy or f-measure is better to use is based on their use, accuracy is an ideal performance measure to use if FP and FN values have the same value i.e. the same number or close, such as 50 infected and 50 uninfected cells, to the data. F-measure is an ideal performance measure if there is an imbalance in sets, here it would be relevant if there are 50 infected and 1000 uninfected cells in the image sets.

F-measure is useful when there is an imbalance in data because it considers FN values from the recall value and FP values from the precision value and averages out the values. Accuracy, on the other hand, treats them as the same value and creates an unfair prediction if the sets are unbalanced one set with less to predict will most likely return a higher value than a set with more to predict which is why f-measure is an ideal solution.

To understand this further Table 7 below shows the TP, FP, TN and FN values returned for the Haar Cascade and CNN classifiers (CNN classifier with the highest accuracy and F-measure results is used).

Program	TP	TN	FP	FN
Haar Cascade	409	1,883	86	137
Neural Network (Epoch20 No Background No Augmentation)	511	1,874	95	35

Table 7: TP, TN, FP and FN values for Haar Cascade and Neural Network

To understand how the difference has occurred for the accuracy value it can be seen in the TP and TN sections of the table for both the Haar Cascade and CNN. The formula for accuracy is the sum of the TN and TP divided by the sum of the TN, TP, FN, and FP. The sum of the TN count and TP count for the Haar Cascade is 2,292 and for the CNN it is 2,385. These values are both divided by 2,515 which is the

total amount of cropped cell images tested (some of the negative cell images contain 2-5 cells within one image which is why the number is so small). Although both have many TN and TP values, they also both have large numbers of FP and FN values.

This, therefore, shows that accuracy does not account for these values, which in most cases is acceptable but, in a life critical program, it is important to account for the positive and negative images incorrectly labelled or missed. Any cells that are infected and are not detected by the classifier cause risk of under diagnosis and the incorrect amount of medication will be given. Or vice versa the marking of uninfected cells as infected causes over diagnosis which will result in too much medication given as the numbers will become much higher. This would not be shown if using accuracy, in terms of accuracy it will simply state that the program is strong at detecting infected cells.

This is where f-measure is more useful, by accounting for the number of FP and FN values through gaining the harmonic mean of the precision and recall values, it is much easier to understand how many positive cells the classifier misses and how many negative cells the cell classes as infected. The aim is to minimise these values, but by providing information on the more difficult areas of the classification process any future adaptations can be based on how strong the classifier is in terms of the more difficult areas in reducing FP and FN values through the f-measure score.

Through the evaluation process, it has become clear that using f-measure is a much better approach, as it accounts for unbalanced classes which in this case for both the Haar Cascade and CNN have more images of uninfected cells than infected as the majority of images are mostly populated by uninfected cells. Using f-measure will create an average value allowing it to be much clearer how accurate it is on both classes overall and will show how it handles some more difficult images. By following this it means that the CNN is the much better program to implement due to its f-measure score.

## 8. Limitations and Constraints

During the time of production of all three programs, there have been several constraints and limitations which may have affected the final results of the programs. The main constraint is the ground truth provided, there were certain cells which the programs did detect as infected but weren't marked as infected, some of these seem as though they should be labelled as infected in the ground truth. This can be proved by evaluating figure 38, this contains two images which were provided in the image training set. These are two separate images but on closer inspection, they are the exact same image but are labelled differently in the ground truth. This can be seen in the red square drawn onto the image to the left in figure 38 where in the image to the right the cell within the red square is labelled.

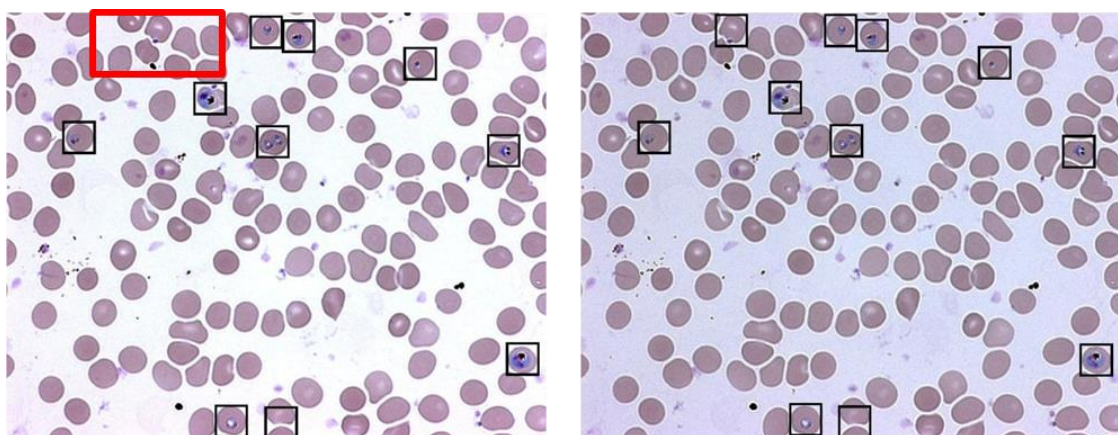


Figure 38: Two of the same images in the training set labelled differently in the ground truth

The only difference between the two images is the illumination, otherwise, the cells are positioned in the same place. This shows that some of the ground truth labelling cannot be trusted if the CNN is to be distributed it will need to be retrained with the images checked once again by another professional to validate some of the labelling.

The cropping tool and program 1 also take a while to process, this means that the user will need to wait a bit longer for these to complete. A way to resolve this is to not use histogram analysis in the thresholding process to remove the background, however, this removes most of the background defects so a suitable alternative would need to be chosen or several different types of thresholding would need to be applied.

With only 50 images for training, it was difficult to know exactly if all images were the same as these in the test set. The small number of images provided did affect the performance of the Haar Cascade and the CNN, with more images they would both have higher f-measure scores and higher accuracies; however, it is an ideal future development for these to improve their performance.

The cropping tool works well, sometimes it does not crop joined cells very well which may affect the performance of the classifiers at times. To resolve this a possible approach would be to separate the cells using Hough transform or watershed segmentation so the cropping tool would know where the cells need to be cropped exactly. Alternatively, the approach used to find all the cells in program 1 could be used where the centre is found using contouring and image moments, from there the further centres of the surrounding cells can be found and the separating values could be estimated.

The final limitation is the need for a large amount of processing power for the CNN, this comes at an expense making the strongest scoring classifier the most expensive of them all. Although this is still cheaper than the cost of employing a professional phlebotomist to evaluate the blood samples.

## 9. Project Planning

This section will explain the usefulness and management process of the project undertaken during the production of the three programs. As the agile methodology was used the workload was logged on a management program called Trello and can be found here <https://trello.com/malariaproject17>. Initially, the project was intended to be completed by the end of week 18, this left about a month to complete the documentation and fix any bugs. The extra month planned in also allowed for any overruns within the project planning process to relieve stress and provide the best outcome possible.

### 9.1 User stories

The initial planning began with user stories, the user stories were produced in the perspective of the user and beneficiaries of the final program, these were then used as priority features in the programs. The medical assistant, in this case, is the laboratory assistant who is not a phlebotomist and does not understand the diagnosis of Malaria. The government is the party controlling funding and making decisions based on the benefit of the population and economy. Finally, the patient is the person with malarial symptoms to be tested. The main user stories can be seen in Table 8 below.

User	Story
<b>Government</b>	As the government, we wish to protect our population with an advanced method of detecting Malaria that is also economically possible.
<b>Government</b>	As the government, we wish to drastically reduce the percentage of resistance to anti-malarial drugs forming through misdiagnosis.



<b>Medical Assistant</b>	As a medical assistant, I want a piece of equipment/program that is easy to use to detect Malaria automatically and quickly as I have no prior knowledge on Malaria diagnosis and how it should look under the microscope.
<b>Patient</b>	As a patient, I want to be diagnosed correctly so that I can be 100% certain I have Malaria or not, this will prevent my body forming a resistance to the anti-malarial drugs I would require if I am infected.

**Table 8: User stories for extraction of main product features**

## 9.2 Backlog

The project was controlled and planned with a product backlog, in the agile methodology this is essentially a list of the requirements that need to be completed by the due date that forms the objectives of the project. The way in which the product backlog is produced is through user stories, in the case of this project the user is either one of two the developer or the student. The user will be a developer if it is with regards to the production of code relating to the product and a student when it relates to a deadline for the project. Each addition to the backlog has a due date and a difficulty applied to it. The complete product backlog can be seen in Appendix 10. The backlog begins after the submission of the initial report, earlier backlog entries can be found on pages 11-13 of the initial report [10].

For each item within the backlog sub-sections were created to complete it section by section. The sub-sections were produced every two weeks during the sprint meetings, these can be seen on Trello.

## 9.3 Reviewing the Project Management

The project was initially planned to be completed by week 18, but when reviewing the sprints, the evaluation was completed by week 26. This, therefore, meant the project overran by 8 weeks in total before the final documentation was started. It was initially expected for the project to overrun as the deadline of week 18 was set to ensure that the majority of the programs and their evaluations were completed. However, when taking a closer look at why the project overran it is clear that it did not go over the estimated deadline at all. This is because, in the initial report [10], it was stated that only two programs would be produced, these being the image processing through segmentation and the Haar Cascade. During the time of production, my supervisor and I spoke about adding a CNN as a third program if I had time remaining after creating the first two programs. In addition to the CNN, I also had to produce three different evaluation tools for each program.

An initial 15 hours was planned for each week, but in some weeks, there were fewer deadlines from other modules so more time could be spent on this project allowing the sprint deadlines to be met faster. The backlog also estimated at least 2 weeks extra for each program to ensure that it would be completed on time and there wasn't any rushing to complete the relevant sections.

### 9.3.1 Program 1 - Image Processing Through Segmentation

This was the first program produced, production on this began in week 4 on the 29<sup>th</sup> of October 2017. The due date for the completion of the first program in the product backlog is 14/01/2018 which is week 15, this gave 11 weeks to grasp the concept of computer vision, the programming concepts it uses and use it to attempt to remove the background from the images and detect the infected cells within the images. The last sprint entry due date for this program is on the 7<sup>th</sup> of January 2018 which is week 14, this meant that the first program was completed a week before it was planned to. However, a closer review into the sprints shows that all the sprints were completed within the time initially planned, with only a couple of cases where it overran. This means that the initial estimate set was too large but it gave an extra week to work on program 2.

There was a stage where an extra two weeks was used, this was during the period where experimentation between using a manual or automatic approach to extract the centre value from the histogram of the image for thresholding was completed. This was completed from the 30<sup>th</sup> of October until the 19<sup>th</sup> of November. During the sprints, the manual method was completed by the 5<sup>th</sup> of November. This method was not very technical but it was time-consuming. The reason this section was therefore extended by two weeks was because the automatic method was very technical, the initial week was used to find the peaks within the histograms using automation code. In the next sprint, the values were then used and added to the relevant sections for the automatic thresholding to take place. Due to the extra time estimation for the first program to be completed within and the completion of other sprints earlier than estimated, this did not affect the overall project management.

A time where the production of the project became difficult and it seemed like the project management would overrun was during the stage of attempting to segment the cells from each other so they could be counted or a close estimate could be made. A week was planned to complete this but the process proved to be difficult and required a lot of research and practise so 2 weeks were used instead of 1. This occurred between sprints 3 and 4 so the item had to be added back to the backlog to be worked on in the following sprint. Overall, meeting the deadline for the first program according to the backlog went very well even though some sections required some more time.

### 9.3.2 Program 2 - Haar Cascading

The Haar Cascading program was initially planned within the backlog to being on the 14<sup>th</sup> of January 2018 in week 15. However, due to program one being completed by the 7<sup>th</sup> of January 2018, the work on the Haar Cascade program began a week early. The planned completion date of the Haar Cascade was the 25<sup>th</sup> of February 2018 in week 21, this left 6 weeks to complete it initially but with the extra week available 7 weeks.

The extra week was used to conduct sufficient research into how the Haar Cascade process works and how it could be implemented to produce a custom XML file to be used to classify images after training. As with all machine learning techniques, the classifier will not usually work the first time so alterations need to be made and the classifier retrained from this. The retraining was planned into the time set for the second program to be produced within, during this time there were two unsuccessful runs where the classifier did not detect the images as intended. On the third attempt, the classifier worked well and the Haar Cascade program was completed on the 4<sup>th</sup> of February 2018. This was three weeks earlier than planned within the backlog. This was mostly because only three attempts were necessary and the time of production was during a period where other module deadlines were completed so more time could be spent on this each week.

### 9.3.3 Program 3 – Convolutional Neural Network

As the Haar Cascade was completed 3 weeks early, the decision was made to produce the additional CNN program. Having the additional program allowed for a much more thorough evaluation to be made so even if this caused an overrun in the project management it makes the project as a whole more complete. Initially, the Haar Cascade training images were created through the manual cropping of the cells as when the classifier was tested it was completed on the complete image initially. When it came to producing the CNN, research showed that the testing images were also required to be cropped into individual/small sections of cells. The cropping tool was completed on time with the backlog on the 12<sup>th</sup> of February. The CNN production began within the same week of the cropping tool being completed in week 20 and the backlog predicted it to be completed by the 18<sup>th</sup> of March 2018 which is week 24. This gave the CNN 4 weeks to be produced within.

This shows that the CNN and the cropping tool is the reason as to why the project overran. The production of the CNN was highly technical and took four attempts. However, it was still completed on time with what was planned within the initial backlog. The further two weeks taking the project to be completed in week 26, as the evaluation programs for each of the three programs were produced.

#### 9.4 Risk Management

The initial risk taken for the project was that I had little knowledge of computer vision. This was quite a big risk considering new programming techniques needed to be learned to produce a program. The risk was handled with a lot of initial research and practise, I also took the computer vision module this year, which during the time of program 1's production taught useful techniques that have been implemented. If I did not understand sections that were needed for any of the programs a sufficient amount of research was conducted to understand it or my supervisor assisted in trying to explain.

The main risk taken during this project was to produce two programs initially, this was already a risk, as most other students were only producing one program within the same amount of time. By adding a third program as well as the production of the evaluation tools for each of the three programs there was a large amount of risk added. The risk could have led to the project not being completed on time or with three programs that did not match the required standards. However, with careful planning and additional hours put into each program for the highest quality possible the risk was managed well and has led to completing the three programs with plenty of time to spare for the documentation.

There was also a risk of having insufficient images to train the two machine learning programs, this was a huge problem initially as with only 50 images it was not possible to train a good classifier. This issue was resolved by creating the image cropping tool which crops the 50 large images into their individual cells and small subsections of it if overlapped. By doing so the image sets counts went into the thousands which is the ideal amount to train a classifier (more is better but this is better than 50 images). The CNN and Haar Cascade classifiers return high accuracy and f-measure scores which shows that by using an extra 2 weeks to produce the cropping tool, it has benefitted their performance.

There were also risks of not completing certain sections within the sprint and having to complete it within the next sprint. This took away the focus from the main idea of that specific sprint as the focus was then placed on completing that task, this only happened twice as mentioned in the 'Project Management' section and it was handled by setting a specific number of hours to that section and if still not complete then a review would need to be made to completely remove it or alter it. The main idea was to not focus on a single subtask for too long as it would put the entire project behind.

#### 9.5 Achievements

The main achievement of this project has been to create three programs from scratch, within the same amount of time that other students will only have produced one program and alongside other module deadlines. This is an achievement as although program 1 doesn't give very high accuracies or f-measure scores it set up the process for the next two programs and taught me a lot about computer vision and its techniques. The CNN and Haar Cascade return very high accuracies and f-measure scores, considering they were only retrained three times in total each the results returned can be regarded as a strong achievement for me.

With the Haar Cascade, I had doubts, after being told this method may not work very well. I have been able to produce the classifier to a very high standard and with a limited number of images. The result of 91.13% is very high considering the worst was assumed and that it wasn't retrained many times because of the lack of time. With extra training images and time, this could be a strong contender in the future of Malaria detection because of how cheap it is to distribute and apply to smartphones.



From this project, as a student, I have learned a lot about computer vision techniques which has benefited my interest in retrieval where I can now work with text and images. This will benefit my future career immensely. The machine learning techniques were difficult to understand but now that I have learned and used them I am able to apply them to any problem. Knowing machine learning techniques is useful as most projects today require the use of these to either compare to an alternative or use it on its own, as they return such high results.

## 9.6 Evaluating the Agile Methodology

Throughout the duration of the project, the agile methodology proved to be the most suitable. One of the main advantages of using agile over the waterfall methodology is the risk mitigation, this being because through constant feedback at every sprint risks can be recognised and resolved before moving forward. On the other hand, with using the waterfall methodology there is a constant risk involved as the development follows a set routine of requirements to be met that cannot be changed easily. Using the agile methodology has enabled the three programs to be created, as, with careful planning and evaluation of what needed to be completed in the time available, I was able to determine that it was possible. If I followed the waterfall methodology instead, it would have been set in stone from the beginning that I would only do the image processing through segmentation and Haar Cascade programs as the rest of my time would be used upon planning the programs using documentation before production actually began.

The agile methodology also proved suitable as at each sprint there was a product that could be used at any point (perhaps not for the complete purpose but it did function and I was aware of more technical areas that needed more time), whereas with the waterfall methodology there would not be any form of product until the initial documentation was complete. This means that with the agile methodology the progress could be measured much more accurately than with the estimations used for the waterfall methodology.

As the project overran the initial estimate, I feel that if I did produce the three programs under the waterfall methodology I would not have had the time to complete them, or if I did they would not be to a high standard. This being due to the amount of documentation required alongside the waterfall methodology. At each sprint, I was also able to attain some feedback from my supervisor which was helpful if there were changes that needed to be made, these could easily be implemented and would not throw off the entire project management like it would with the waterfall methodology.

The agile methodology, therefore, proved to be the most suitable methodology for this project, allowing me to make slight alterations to the features or code when required to save time and to get the best possible programs produced.

## 10. Code Adaptions and Modifications Used

The code used in the three programs produced did require some modifications and adaptions of code. The following adaptions/modifications were made:

1. The use of separated LAB channels with Otsu thresholding applied to images of flowers as explained in the article by A. Patil and J. Shaikh [24] was modified for the background removal of the images. A. Patil and J. Shaikh only used the 'B' channel, however, the code was modified and the 'L' and 'A' values were used with CLAHE applied to them. A modification of this code was used as the results of using the 'L' and 'A' channels with CLAHE instead of just the 'B' channel proved to work a lot better for my images. The way in which it was used is explained in section 4.2.

2. The use of region labelling was adapted from an online article on Pyimagesearch by A. Rosebrock [45]. A similar method to the one used was used to complete the region labelling tasks within the background removal of the images.
3. Calculating the image moments can only be completed in one way in terms of contouring, for this the OpenCV documentation on contour features[46] was used. This assisted in finding the centre x and y values of the cells.
4. The CNN code for training and evaluation was modified from a CNN model found on GitLab [49]. This code went into much more detail than my version however it was useful to understand the process of producing a CNN.
5. The CNN architecture shown in section 4.4.1 was adapted from the MNIST training code as explained earlier, this was found online on GitLab [48]. The model used is slightly different but very similar.

## 11. Legal

The main concern of using the Haar Cascade was the licensing behind it. However, after some research on the topic, it was discovered that OpenCV has released the use of the Haar Cascade classifier under a BSD license. This means that the Haar Cascade can be used for academic and commercial use. The BSD license requires distributed code to contain the original copyright notice, two restrictions and a disclaimer of liability [50], otherwise this is a free open source piece of software that can be used in a cost-effective way to detect and diagnose Malaria.

## 12. Discussion and Concluding Remarks

Overall within the period of time provided, I have successfully been able to produce three programs as intended, each of the three programs are sufficiently different. The first program showed little promise with a very low accuracy and F-measure. As mentioned earlier this was mainly because of the method used to detect the dark sections within the image. This program is therefore discarded from being an option to be used in the real world, as it only offers a 20.47% accuracy and 28% f-measure which in both cases is extremely low. There are several options in which this program could be improved, the first is to create a separate colour rebalancing algorithm for the different ranges of the darkness of the images as done for the morphology stage. By doing so the images passed with their background removed could have a different amount of colour alterations applied to them to ensure each specific range only extracts the Giemsa stained artefacts within the images. A second alteration would be to completely remove the colour rebalance algorithm and apply histogram analysis to each cell or section of an image, if the section/cell contains the blue-black colours there will be a higher peak within the histogram and this can be marked as an infection. This program was however proved useful in the determination of removing the background from images and was used for the cropping tools that I also produced for the training and testing of the Haar Cascade and CNN.

Excluding program 1 has left the final program to be used to be chosen between the Haar Cascade or the CNN. Both programs use supervised machine learning to classify the images and both use cropped cells to be trained and to be tested. For the accuracy value, there is only a 3.87% difference between the Haar Cascade and CNN. If accuracy is being considered, which as mentioned before is not the ideal performance measure to use, there is not a lot of difference to determine correctly which is better. This is because possibly one or two variables could have been tweaked during the Haar Cascade training to possibly reach the same value as the CNN if not even higher. In terms of accuracy, the CNN does return the highest value, but either could be considered as an option.

In terms of f-measure the CNN is once again higher than the Haar Cascade program, however this time it is 10% different. The difference is much larger and shows that the CNN allows for more complex situations within images and is able to make better classification decisions. In terms of only evaluating the F-measure result then the CNN by far is the better option to choose. The f-measure score is most likely the better performance measure to be used as it ensures that unbalanced classes are accounted for. In the case of this project during testing there are a lot more cropped single uninfected cells than there are cropped single infected cells, this makes the accuracy value slightly biased as it does not account for this. For this reason, I would recommend using the CNN. With a higher f-measure it means that the CNN can make stronger decisions based on the number of true detections made.

However, to meet the initial goal of my research it was mentioned that poverty was the main issue. The CNN will be much more expensive to implement than the Haar Cascade, as the Haar Cascade is already implemented in a cost-effective way on smartphone cameras on the industrial scale, it also does not require a lot of processing power. This means that the Haar Cascade could be easily implemented into a smartphone. Whereas with the CNN a lot of research would be required to create a program that will support the model on a smartphone, including the requirement to have processing power. The need for less processing power also means that the Haar Cascade is significantly faster than the CNN, which has shown during testing where it took seconds compared to a few minutes for the CNN. As the CNN requires more processing power, it will also be more expensive to implement. For these reasons, the Haar Cascade would be recommended, it may have slightly more FP values than the CNN but it will still detect very well and it will be much cheaper to implement. The only disadvantage of using the Haar Cascade is that eventually if given too many images it will stop learning as the Haar-like features will become the same and will begin to block further learning, so this would need to be monitored.

Compared to the initial plan I have been able to meet all the intended requirements, with the addition of the CNN which was not initially intended to be produced. Alongside this, the production of the cropping tool and evaluation tools for each program were created and within the time allocated to me. I am satisfied with the results returned which could only be improved with more images that could be provided to the training process.

### 12.1 Future Extensions

The extensions for either the Haar Cascade or CNN all lead to one result. The classifiers could be implemented into an application for a smartphone or similar small hardware device with a screen and camera. The smartphone or device would require a microscope to be clipped onto the lens with a prepared blood smear slide attached to it, this would allow for images similar to those used for testing and training to be created from new samples of blood. The application would then use the classifiers to determine if the images of the blood smears contain any infected cells. the results should then be calculated and returned to the lab assistant to provide information as to whether or not the patient has Malaria or not.

This would be the ideal method to distribute either of the classifiers into the areas facing poverty. By having a lab assistant produce the blood smears and provide them to the program, it not only speeds up the process but it also cuts the costs drastically as these workers are much less skilled than a phlebotomist who requires a much higher hourly rate.

Alternatively, if more images become available then a future extension would be to retrain both the Haar Cascade and CNN with the extra images and then evaluate them once again. It may show stronger signs of the better classifier to use and implement.

### 13. Definitions and Acronyms

Word/Acronym	Definition
<b>ROI</b>	Region of Interest
<b>CLAHE</b>	Contrast Limited Adaptive Histogram Equalisation
<b>False Positive (FP)</b>	when an infection is not present and it is identified as infected
<b>True Positive (TP)</b>	when an infection is present and it is identified as infected
<b>False Negative (FN)</b>	when an infection is present and it is not identified
<b>True Negative (TN)</b>	when an infection is not present and it is not identified
<b>HSV</b>	Hue Saturation Value
<b>WHO</b>	World Health Organisation
<b>ANN</b>	Artificial Neural Network
<b>SVM</b>	Support Vector Machine
<b>CNN</b>	Convolutional Neural Network

## Bibliography

- [1] C.-C. for D. C. and Prevention, "CDC - Malaria - About Malaria - History," 2017.
- [2] C.-C. for D. C. and Prevention, "CDC - Malaria - About Malaria - History - Laveran and the Discovery of the Malaria Parasite," 2017.
- [3] Merriam-Webster Incorporated, "Sporozoite | Definition of Sporozoite by Merriam-Webster," 2018. [Online]. Available: <https://www.merriam-webster.com/dictionary/sporozoite>. [Accessed: 29-Mar-2018].
- [4] NHS, "Malaria - Symptoms - NHS.UK." [Online]. Available: <https://www.nhs.uk/conditions/malaria/symptoms/>. [Accessed: 29-Mar-2018].
- [5] Geographical Association, "HEALTH ISSUES IN GEOGRAPHY," pp. 15–17.
- [6] D. E. Loy *et al.*, "Out of Africa: origins and evolution of the human malaria parasites *Plasmodium falciparum* and *Plasmodium vivax*," *Int. J. Parasitol.*, vol. 47, pp. 87–97, 2017.
- [7] WHO, "WHO | Key points: World malaria report 2017," WHO, 2017.
- [8] O.-O. E. Roser Max, *Global Extreme Poverty*. OurWorldInData.org, 2017.
- [9] S. A. Fana, M. Danladi, A. Bunza, S. A. Anka, A. U. Imam, and S. U. Nataala, "Prevalence and risk factors associated with malaria infection among pregnant women in a semi-urban community of north-western Nigeria."
- [10] M. Delekta, "Initial Report: Assessing Blood Samples for Malaria," pp. 1–16.
- [11] P. B. Bloland, "Drug resistance in malaria," p. 12, 2001.
- [12] L. Stratton, M. S. O'Neill, M. E. Kruk, and M. L. Bell, "The persistent problem of malaria: Addressing the fundamental causes of a global killer," 2008.
- [13] F. Boray Tek, A. G. Dempster, and I. Kale, "Parasite detection and identification for automated thin blood film malaria diagnosis," *Comput. Vis. Image Underst.*, vol. 114, pp. 21–32, 2009.
- [14] S. Ferrari, V. Piuri, and F. Scotti, "Virtual environment for granulometry analysis," *VECIMS 2008 - IEEE Conf. Virtual Environ. Human-Computer Interfaces Meas. Syst. Proc.*, no. July, pp. 156–157, 2008.
- [15] C. Di Ruberto, A. Dempster, S. Khan, and B. Jarra, "Analysis of infected blood cell images using morphological operators," 2001.
- [16] E. W. R. Fisher, S. Perkins, A. Walker, "Morphology - Skeletonization/Medial Axis Transform," 2003. [Online]. Available: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm>. [Accessed: 30-Mar-2018].
- [17] A. S. S. Shirgan, and V. R. Marathe, "Automatic Identification of Malaria Parasites using Image Processing Miss. Shruti Annaldas," *Int. J. Emerg. Eng. Res. Technol.*, vol. 2, no. 4, pp. 107–112, 2014.
- [18] Sunil Ray, "Understanding Support Vector Machine algorithm from examples (along with code)," 2017. [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>. [Accessed: 30-Mar-2018].
- [19] S. P. Premaratne, N. D. Karunaweera, and S. Fernando, "A Neural Network Architecture for Automated Recognition of Intracellular Malaria Parasites in Stained Blood Films," pp. 4–7, 2006.
- [20] M. I. Razzak, "Automatic Detection and Classification of Malarial Parasite," *Muhammad Imran Razzak Int. J. Biometrics Bioinforma.*, no. 91, pp. 2015–1.
- [21] S. Krishnan, S. Antani, and S. Jaeger, "Visualizing Deep Learning Activations for Improved Malaria Cell Classification."
- [22] F. Ertam and G. Aydin, "Data classification with deep learning using Tensorflow," *2017 Int. Conf. Comput. Sci. Eng.*, pp. 755–758, 2017.

- [23] M. H. J. Vala and A. Baxi, "A review on Otsu image segmentation algorithm," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 2, pp. 387–389, 2013.
- [24] A. B. Patil and J. Shaikh, "OTSU Thresholding Method for Flower Image Segmentation," pp. 1–6, 2016.
- [25] OpenCV - Doxygen, "OpenCV: Morphological Transformations," 2018. [Online]. Available: [https://docs.opencv.org/trunk/d9/d61/tutorial\\_py\\_morphological\\_ops.html](https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html). [Accessed: 30-Mar-2018].
- [26] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *Sixth Int. Conf. Comput. Vis. (IEEE Cat. No.98CH36271)*, pp. 839–846.
- [27] W. E. Fisher R, Perkins S, Walker A, "Spatial Filters - Median Filter," 2003. [Online]. Available: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>. [Accessed: 30-Mar-2018].
- [28] W. E. Fisher R, Perkins S, Walker A, "Image Analysis - Connected Components Labeling," 2003. [Online]. Available: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm>. [Accessed: 30-Mar-2018].
- [29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. 2001 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition. CVPR 2001*, vol. 1, p. I-511-I-518, 2001.
- [30] Chris McCormick, "AdaBoost Tutorial," 2014. [Online]. Available: <http://mccormickml.com/2013/12/13/adaboost-tutorial/>. [Accessed: 31-Mar-2018].
- [31] R. E. Schapire, "Explaining AdaBoost," in *Empirical Inference*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 37–52.
- [32] J. Brownlee, "Boosting and AdaBoost for Machine Learning," 2016. [Online]. Available: <https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>. [Accessed: 31-Mar-2018].
- [33] Brownlee Jason, "Overfitting and Underfitting With Machine Learning Algorithms - Machine Learning Mastery," 2016. [Online]. Available: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>. [Accessed: 01-Apr-2018].
- [34] R. Aditya, M. Budiman, B. Achmad, A. Arif, and L. Zharif, "Localization of White Blood Cell Images using Haar Cascade Classifiers," pp. 1–5, 2016.
- [35] C. C. Tanz Ophir, "Neural networks made easy | TechCrunch," 2017. [Online]. Available: <https://techcrunch.com/2017/04/13/neural-networks-made-easy/>. [Accessed: 02-Apr-2018].
- [36] Machine Learning Guru, "Image Convolution." [Online]. Available: [http://machinelearningguru.com/computer\\_vision/basics/convolution/image\\_convolution\\_1.html](http://machinelearningguru.com/computer_vision/basics/convolution/image_convolution_1.html). [Accessed: 02-Apr-2018].
- [37] Changhu Isaac, "Activation Functions in Artificial Neural Networks | Isaac Changhau," 2017. [Online]. Available: <https://isaacchanghau.github.io/2017/05/22/Activation-Functions-in-Artificial-Neural-Networks/>. [Accessed: 02-Apr-2018].
- [38] Gupta Dishashree, "Fundamentals of Deep Learning - Activation Functions and their use," 2017. [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/>. [Accessed: 02-Apr-2018].
- [39] Waila Singh Anish, "Activation functions and it's types-Which is better?," 2017. [Online]. Available: <https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f>. [Accessed: 02-Apr-2018].
- [40] Deshpande Adit, "A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 – Adit Deshpande – CS Undergrad at UCLA ('19)," 2016. [Online]. Available: <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>. [Accessed: 02-Apr-2018].
- [41] Ujjwalkarn, "An Intuitive Explanation of Convolutional Neural Networks," 2016. [Online]. Available:



- <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>. [Accessed: 02-Apr-2018].
- [42] S. D. Stergiou Christos, "Neural Networks." [Online]. Available: [https://www.doc.ic.ac.uk/~nd/surprise\\_96/journal/vol4/cs11/report.html](https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html). [Accessed: 02-Apr-2018].
  - [43] NVIDIA Corporation, "CUDA | Supported GPUs | GeForce," 2018. [Online]. Available: <https://www.geforce.com/hardware/technology/cuda/supported-gpus>. [Accessed: 03-Apr-2018].
  - [44] OpenCV, "OpenCV: Histograms - 2: Histogram Equalization," 2015. [Online]. Available: [https://docs.opencv.org/3.1.0/d5/daf/tutorial\\_py\\_histogram\\_equalization.html](https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html). [Accessed: 03-Apr-2018].
  - [45] Adrian Rosebrock, "Detecting multiple bright spots in an image with Python and OpenCV - PyImageSearch," 2016. [Online]. Available: <https://www.pyimagesearch.com/2016/10/31/detecting-multiple-bright-spots-in-an-image-with-python-and-opencv/>. [Accessed: 04-Apr-2018].
  - [46] OpenCV, "OpenCV: Contour Features," 2017. [Online]. Available: [https://docs.opencv.org/3.4.0/dd/d49/tutorial\\_py\\_contour\\_features.html](https://docs.opencv.org/3.4.0/dd/d49/tutorial_py_contour_features.html). [Accessed: 04-Apr-2018].
  - [47] OpenCV, "OpenCV: Cascade Classifier Training." [Online]. Available: [https://docs.opencv.org/3.3.0/dc/d88/tutorial\\_traincascade.html](https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html). [Accessed: 06-Apr-2018].
  - [48] treszkai fchollet, matsuyamax, Smerity, kemaswill, "keras/mnist\_cnn.py at master · keras-team/keras · GitHub," 2018. [Online]. Available: [https://github.com/keras-team/keras/blob/master/examples/mnist\\_cnn.py](https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py). [Accessed: 06-Apr-2018].
  - [49] anujshah1003, "own\_data\_cnn\_implementation\_keras/custom\_data\_cnn.py at master · anujshah1003/own\_data\_cnn\_implementation\_keras · GitHub," 2018. [Online]. Available: [https://github.com/anujshah1003/own\\_data\\_cnn\\_implementation\\_keras/blob/master/custom\\_data\\_cnn.py](https://github.com/anujshah1003/own_data_cnn_implementation_keras/blob/master/custom_data_cnn.py). [Accessed: 19-Apr-2018].
  - [50] The Linux Information Project, "BSD license definition," 2005. [Online]. Available: <http://www.lininfo.org/bsdlicense.html>. [Accessed: 19-Apr-2018].

## Appendix

### Appendix 1 – Morphological Operators Applied to Colour Rebalanced Image

Showing the morphological operations applied to each of the white pixel counts in the colour rebalanced images. Range is the range of white pixel count the image can fall between and iterations is the number of iterations for the specific morphological operator it is in line with in column 2, applied.

Range	Morphological Operators and Kernel	Iterations
<b>Above 45,000</b>	<ul style="list-style-type: none"> <li>Erode – (5, 5)</li> <li>Dilate – (7, 7)</li> <li>Erode – (3, 3)</li> <li>Erode (8, 8)</li> <li>Erode (5, 5)</li> </ul>	3 10 4 4 2
<b>from 25,000 up to and including 45,000</b>	<ul style="list-style-type: none"> <li>Erode – (10, 10)</li> <li>Morphology Close – (10, 10)</li> <li>Erode – (5, 5)</li> <li>Dilate – (3, 3)</li> <li>Erode - (10, 10)</li> <li>Erode – (5, 5)</li> </ul>	1 1 2 1 8 5
<b>from 10,000 up to and including 25,000</b>	<ul style="list-style-type: none"> <li>Erode – (5, 5)</li> <li>Dilate – (7, 7)</li> <li>Erode – (3, 3)</li> <li>Erode (8, 8)</li> <li>Erode (5, 5)</li> </ul>	3 10 4 4 2
<b>from 5,500 up to and including 10,000</b>	<ul style="list-style-type: none"> <li>Erode – (3, 3)</li> <li>Erode – (5, 5)</li> </ul>	1 1
<b>from 1,000 up to and including 5,500</b>	<ul style="list-style-type: none"> <li>Erode – (3, 3)</li> <li>Dilate – (3, 3)</li> </ul>	1 2

Table i: Morphology operations applied to each range of white pixel counts for specific images falling within that range

### Appendix 2 – Program 1 Detection Output File

The following is the output from the test file “Image\_Processing\_Detection\_Results.txt”. Where number of infected is the number of labelled cells by the detection program, the average cell count is the average count of cells in each image and infected is ‘Y’ if infected or ‘N’ if not infected.

Results for Image Processing Detection Results By Image Name:

File	Number of Infected	Average Cell Count	Infected
051.jpg	8	219	Y
052.jpg	10	165	Y
053.jpg	13	200	Y
054.jpg	8	189	Y
055.jpg	4	144	Y
056.jpg	24	173	Y
057.jpg	19	151	Y
058.jpg	20	155	Y
059.jpg	14	135	Y
060.jpg	29	143	Y
061.jpg	8	200	Y
062.jpg	8	181	Y
063.jpg	17	178	Y

064.jpg	12		207		Y	
065.jpg	14		177		Y	
066.jpg	9		135		Y	
067.jpg	9		123		Y	
068.jpg	10		142		Y	
069.jpg	12		165		Y	
070.jpg	13		151		Y	
071.jpg	5		220		Y	
072.jpg	13		215		Y	
073.jpg	17		194		Y	
074.jpg	5		186		Y	
075.jpg	3		221		Y	
076.jpg	18		180		Y	
077.jpg	7		178		Y	
078.jpg	16		201		Y	
079.jpg	21		196		Y	
080.jpg	13		179		Y	
081.jpg	12		119		Y	
082.jpg	18		293		Y	
083.jpg	25		107		Y	
084.jpg	13		380		Y	
085.jpg	7		209		Y	
086.jpg	1		295		Y	
087.jpg	4		187		Y	
088.jpg	12		221		Y	
089.jpg	4		211		Y	
090.jpg	5		221		Y	
091.jpg	13		131		Y	
092.jpg	7		139		Y	
093.jpg	23		156		Y	
094.jpg	8		140		Y	
095.jpg	5		139		Y	
096.jpg	20		143		Y	
097.jpg	27		153		Y	
098.jpg	27		145		Y	
099.jpg	27		179		Y	
100.jpg	23		147		Y	
+-----+-----+-----+-----+						

### Appendix 3 – Haar Default Values of Training Process Explained

These are the default values used for the training of the Haar Cascade classifier:

- **precalcValBufSize** – amount of memory assigned to the buffer, 256Mb is the default.
- **precalcIdxBufSize** – memory assigned for the buffer for pre-calculated feature values, 256Mb is assigned to this as well.
- **minHitRate** – minimal hit rate to be obtained, the default is 0.995 which means 0.005% of positive images can be misclassified during training.
- **maxFalseAlarmRate** – maximum false alarm rate, it defines how many features need to be added during each training stage, 0.5 is the default.
- **weightTrimRate** – indicates if trimming is to be used, this is a value between 0 and 1 and is used to preserve computational time. If the value is below the weight trim rate it is not used in the next training iteration.
- **maxDepth** – maximum depth of a weak tree
- **maxWeakCount** – maximum number of weak trees for each cascade stage.

### Appendix 4 – Training Log for Haar Cascade

The following contains the training log from the working Haar Cascade classifier. This was output into the command line and saved. This contains a section of it as the text file was too large.

```
opencv_traincascade -data dir1_30Jan -vec cells30Jan.vec -bg negCells.txt -numPos 290 -numNeg 1931 -
numStages 10 -w 15 -h 15 -featureType HAAR -mode ALL -bt RAB
```

PARAMETERS:

```
cascadeDirName: dir1_30Jan
vecFileName: cells30Jan.vec
bgFileName: negCells.txt
numPos: 290
numNeg: 1931
numStages: 10
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: HAAR
sampleWidth: 15
sampleHeight: 15
boostType: RAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL
```

===== TRAINING 0-stage =====

<BEGIN

POS count : consumed 290 : 290

NEG count : acceptanceRatio 1931 : 1

Precalculation time: 3

	N	HR	FA
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	0.58985	
7	1	0.672191	
8	0.996552	0.515277	
9	0.996552	0.417918	

END>

Training until now has taken 0 days 0 hours 0 minutes 38 seconds.

===== TRAINING 1-stage =====

<BEGIN

POS count : consumed 290 : 291

NEG count : acceptanceRatio 1931 : 0.419965

Precalculation time: 3

```
+-----+
| N |  HR |  FA |
+-----+
| 1 |   1 |   1 |
+-----+
| 2 |   1 |   1 |
+-----+
| 3 |   1 |   1 |
+-----+
| 4 |   1 |   1 |
+-----+
| 5 |   1 |   1 |
+-----+
| 6 |   1 |   1 |
+-----+
| 7 |   1 | 0.760746 |
+-----+
| 8 | 0.996552 | 0.6261 |
+-----+
| 9 | 0.996552 | 0.655619 |
+-----+
|10 | 0.996552 | 0.469705 |
+-----+
```

END>

Training until now has taken 0 days 0 hours 1 minutes 19 seconds.

===== TRAINING 2-stage =====

<BEGIN

POS count : consumed 290 : 292

NEG count : acceptanceRatio 1931 : 0.200457

Precalculation time: 2

```
+-----+
| N |  HR |  FA |
+-----+
| 1 |   1 |   1 |
+-----+
| 2 |   1 |   1 |
+-----+
| 3 |   1 |   1 |
+-----+
| 4 |   1 |   1 |
+-----+
| 5 |   1 | 0.862765 |
+-----+
| 6 |   1 | 0.870533 |
+-----+
| 7 |   1 | 0.774728 |
+-----+
| 8 | 0.996552 | 0.613672 |
+-----+
| 9 | 0.996552 | 0.638011 |
+-----+
```

| 10| 0.996552| 0.493009|

+-----+-----+

END>

Training until now has taken 0 days 0 hours 2 minutes 0 seconds.

...

===== TRAINING 9-stage =====

<BEGIN

POS count : consumed 290 : 299

NEG count : acceptanceRatio 1931 : 0.000592952

Required leaf false alarm rate achieved. Branch training terminated.

## Appendix 5 – Haar Detection Text File Output

This contains a subsection of the text file “Haar\_Results\_Detection.txt”, as the entire text file was too large. File is the image file name and infection detected is ‘False’ if there is no infection detected or ‘True’ if there is an infection detected.

Results for Haar Cascading on Cropped Images With Background Removed

Counts of TP, TN, FP and FN:

File	Infection Detected
051.jpgCropped1.png	False
051.jpgCropped10.png	False
051.jpgCropped11.png	False
051.jpgCropped12.png	False
051.jpgCropped13.png	False
051.jpgCropped14.png	False
051.jpgCropped15.png	False
051.jpgCropped16.png	True
051.jpgCropped17.png	False
051.jpgCropped18.png	False
051.jpgCropped19.png	True
051.jpgCropped2.png	False
051.jpgCropped20.png	False
051.jpgCropped21.png	False
051.jpgCropped22.png	False
051.jpgCropped23.png	False
051.jpgCropped24.png	False
051.jpgCropped25.png	False
051.jpgCropped26.png	False
051.jpgCropped27.png	False
051.jpgCropped28.png	False
051.jpgCropped29.png	False
051.jpgCropped3.png	False
051.jpgCropped30.png	False
051.jpgCropped31.png	False
051.jpgCropped32.png	False
051.jpgCropped33.png	True
051.jpgCropped34.png	False
051.jpgCropped35.png	False
051.jpgCropped36.png	False
051.jpgCropped37.png	False

## Appendix 6 – Neural Network Image Detection Results

This contains a small section of the results from the neural network detecting input images, the file shown if on images without their background. Where in the infected column N is uninfected and Y is infected.



## Results for Neural Network Epochs 15, 20 and 30 - Without Background

Results Calculated:

File	Infected
085.jpgCropped39.png	N
068.jpgCropped72.png	N
065.jpgCropped96.png	N
088.jpgCropped65.png	N
073.jpgCropped128.png	N
073.jpgCropped58.png	N
076.jpgCropped111.png	N
079.jpgCropped84.png	N
089.jpgCropped2.png	N
078.jpgCropped76.png	N

...

## Appendix 7 – Image Processing Text File Evaluation Results

Showing the results for the image processing through segmentation program calculated using the evaluation tool. Total infected is the number of infected cells found in the entire image, TP, TN, FP and FN are the counts found within the images.

Results for Image Processing Program Used On Complete Images With Background Removed

Counts of TP, TN, FP and FN:

File	Total Infected	TP	TN	FP	FN
078.jpg	18	8	0	49	10
057.jpg	1	0	0	20	1
055.jpg	5	1	0	3	4
063.jpg	13	6	0	13	7
070.jpg	18	12	0	2	6
089.jpg	8	2	0	3	6
065.jpg	14	6	0	15	8
100.jpg	17	13	0	29	4
060.jpg	6	4	0	25	2
076.jpg	19	11	0	6	8
058.jpg	6	3	0	19	3
073.jpg	9	5	0	13	4
097.jpg	17	6	0	20	11
062.jpg	10	3	0	5	7
066.jpg	8	4	0	5	4
092.jpg	13	0	0	5	13
053.jpg	3	2	0	13	1
074.jpg	5	0	0	22	5
080.jpg	22	8	0	12	14
088.jpg	14	6	0	7	8
081.jpg	9	5	0	37	4
071.jpg	13	4	0	2	9
068.jpg	7	5	0	6	2
093.jpg	9	2	0	21	7
096.jpg	6	2	0	27	4
087.jpg	21	0	0	4	21
082.jpg	16	1	0	20	15
052.jpg	2	0	0	13	2

095.jpg	10	3	0	2	7	
085.jpg	16	2	0	5	14	
054.jpg	2	1	0	8	1	
077.jpg	21	5	0	2	16	
069.jpg	10	6	0	8	4	
056.jpg	7	2	0	21	5	
051.jpg	3	2	0	9	1	
072.jpg	11	4	0	19	7	
079.jpg	14	9	0	45	5	
084.jpg	13	2	0	14	11	
098.jpg	19	11	0	16	8	
067.jpg	6	2	0	7	4	
099.jpg	8	4	0	27	4	
090.jpg	15	3	0	2	12	
064.jpg	11	3	0	6	8	
094.jpg	10	4	0	7	6	
061.jpg	8	7	0	1	1	
075.jpg	11	1	0	9	10	
059.jpg	2	1	0	14	1	
091.jpg	12	4	0	9	8	
086.jpg	11	0	0	2	11	
083.jpg	6	3	0	26	3	
+-----+-----+-----+-----+-----+-----+						

Results Calculated:

File	Precision	Recall	F-measure	Accuracy
078.jpg	0.14	0.44	0.21	11.94
057.jpg	0.00	0.00	0.00	0.00
055.jpg	0.25	0.20	0.22	12.50
063.jpg	0.32	0.46	0.37	23.08
070.jpg	0.86	0.67	0.75	60.00
089.jpg	0.40	0.25	0.31	18.18
065.jpg	0.29	0.43	0.34	20.69
100.jpg	0.31	0.76	0.44	28.26
060.jpg	0.14	0.67	0.23	12.90
076.jpg	0.65	0.58	0.61	44.00
058.jpg	0.14	0.50	0.21	12.00
073.jpg	0.28	0.56	0.37	22.73
097.jpg	0.23	0.35	0.28	16.22
062.jpg	0.38	0.30	0.33	20.00
066.jpg	0.44	0.50	0.47	30.77
092.jpg	0.00	0.00	0.00	0.00
053.jpg	0.13	0.67	0.22	12.50
074.jpg	0.00	0.00	0.00	0.00
080.jpg	0.40	0.36	0.38	23.53
088.jpg	0.46	0.43	0.44	28.57
081.jpg	0.12	0.56	0.20	10.87
071.jpg	0.67	0.31	0.42	26.67
068.jpg	0.45	0.71	0.56	38.46
093.jpg	0.09	0.22	0.12	6.67
096.jpg	0.07	0.33	0.11	6.06
087.jpg	0.00	0.00	0.00	0.00
082.jpg	0.05	0.06	0.05	2.78
052.jpg	0.00	0.00	0.00	0.00
095.jpg	0.60	0.30	0.40	25.00
085.jpg	0.29	0.12	0.17	9.52
054.jpg	0.11	0.50	0.18	10.00
077.jpg	0.71	0.24	0.36	21.74
069.jpg	0.43	0.60	0.50	33.33
056.jpg	0.09	0.29	0.13	7.14
051.jpg	0.18	0.67	0.29	16.67
072.jpg	0.17	0.36	0.24	13.33
079.jpg	0.17	0.64	0.26	15.25
084.jpg	0.12	0.15	0.14	7.41
098.jpg	0.41	0.58	0.48	31.43
067.jpg	0.22	0.33	0.27	15.38

099.jpg	0.13	0.50	0.21	11.43
090.jpg	0.60	0.20	0.30	17.65
064.jpg	0.33	0.27	0.30	17.65
094.jpg	0.36	0.40	0.38	23.53
061.jpg	0.88	0.88	0.88	77.78
075.jpg	0.10	0.09	0.10	5.00
059.jpg	0.07	0.50	0.12	6.25
091.jpg	0.31	0.33	0.32	19.05
086.jpg	0.00	0.00	0.00	0.00
083.jpg	0.10	0.50	0.17	9.38
Averages:	0.27	0.38	0.28	17.67

## Appendix 8 – Haar Evaluation Results

Showing the file created by the Haar Cascade evaluation tool. The counts of TP, FP, TN, and FN are first shown in a table where below them are their relevant counts. The second table shows the value calculated for precision, recall, f-measure, and accuracy with their results found below their title.

Results for Haar Cascading on Cropped Images With Background Removed

Counts of TP, TN, FP and FN:

TP	TN	FP	FN
409	2341	279	137

Results Calculated:

Precision	Recall	F-measure	Accuracy
0.59	0.75	0.66	86.86%

## Appendix 9 – Neural Network Evaluation Results for Background and No Background

Shows two files, the first for epoch models 15, 20, and 30 trained on images without their background and the second for epoch models 15, 20, and 30 trained with their background. They both contain two tables, the first showing each model in line with their TN, FN, TP, and FP counts on the training set and the second showing each models value calculated for precision, recall, f-measure, and accuracy. Alongside these, there are also columns 'augmented' which is 'Y' if the model applied augmentation to the images or 'N' if not, and 'background' which is 'Y' if the images used for training did not have their background removed or 'N' if they did have their background removed.

### No Background

Results for Neural Network Epochs 15, 20 and 30 - Without Background

Counts of TP, TN, FP and FN:

File	TP	TN	FP	FN
CNN_No_Background_EPOCH15.hdf5	519	1830	139	27
CNN_No_Background_EPOCH20.hdf5	511	1874	95	35
CNN_No_Background_EPOCH30.hdf5	514	1852	117	32
CNN_No_Background_EPOCH15.hdf5	531	1465	504	15
CNN_No_Background_EPOCH20.hdf5	511	1828	141	35
CNN_No_Background_EPOCH30.hdf5	528	1783	186	18

Results Calculated:

File	Augmented	Background	Precision	Recall	F-measure	Accuracy
CNN_No_Background_EPOCH15.hdf5	Not_Augmented	N	0.79	0.95	0.86	0.93
CNN_No_Background_EPOCH20.hdf5	Not_Augmented	N	0.84	0.94	0.89	0.95
CNN_No_Background_EPOCH30.hdf5	Not_Augmented	N	0.81	0.94	0.87	0.94
CNN_No_Background_EPOCH15.hdf5	Augmented	N	0.51	0.97	0.67	0.79
CNN_No_Background_EPOCH20.hdf5	Augmented	N	0.78	0.94	0.85	0.93
CNN_No_Background_EPOCH30.hdf5	Augmented	N	0.74	0.97	0.84	0.92

## With Background

Results for Neural Network Epochs 15, 20 and 30 - With Background

Counts of TP, TN, FP and FN:

File	TP	TN	FP	FN
CNN_with_background_EPOCH15.hdf5	513	1543	210	33
CNN_with_background_EPOCH20.hdf5	497	1614	139	49
CNN_with_background_EPOCH30.hdf5	505	1606	147	41
CNN_with_background_EPOCH15.hdf5	524	1470	283	22
CNN_with_background_EPOCH20.hdf5	521	1512	241	25
CNN_with_background_EPOCH30.hdf5	531	1375	378	15

Results Calculated:

File	Augmented	Background	Precision	Recall	F-measure	Accuracy
CNN_with_background_EPOCH15.hdf5	Not_Augmented	Y	0.71	0.94	0.81	0.89
CNN_with_background_EPOCH20.hdf5	Not_Augmented	Y	0.78	0.91	0.84	0.92
CNN_with_background_EPOCH30.hdf5	Not_Augmented	Y	0.77	0.92	0.84	0.92
CNN_with_background_EPOCH15.hdf5	Augmented	Y	0.65	0.96	0.77	0.87
CNN_with_background_EPOCH20.hdf5	Augmented	Y	0.68	0.95	0.80	0.88
CNN_with_background_EPOCH30.hdf5	Augmented	Y	0.58	0.97	0.73	0.83

## Appendix 10 – Product Backlog

Showing the product backlog used during the production of the three programs and the time estimates provided for each so that the project would be completed on time.

User Story	Difficulty	Due Date
As a developer, I need to look at the labelled images of blood samples provided to understand what I need to segment out of the image.	*	29/10/2017
As a developer, I need to sort the images from those containing the least amount of parasites to the most, this will make testing easier whilst producing the algorithm.	*	29/10/2017
As a developer I need to know the correct values for thresholding, to do so I will need to create and review the histogram of each grayscale image.	**	19/11/2017

As a developer, I need to use the retrieved values for thresholding to implement the thresholding process. It must be ensured that all red blood cells are kept and not segmented out of the image.	**	19/11/2017
As a developer, I need to ensure this algorithm works on all of the images before moving on.	**	19/11/2017
As a developer I need to research ways to segment and count overlapping/joined cells both from the original image and from the segmentation process.	***	19/11/2017
As a developer, I need to research into the best approach for determining if the red blood cells are clean or infected.	**	17/12/2017
As a developer I need to implement from my research the techniques learned to segment and count the overlapping/joined cells. EXTENDED	****	24/12/2017
As a student I need to complete and submit the interim oral examination form.	**	6/12/2017
As a student, I need to prepare for my interim oral examination.	*	10/12/2017
As a student, I need to attend the interim oral examination, here I must speak about the work completed so far on my project and any technical difficulties surrounding it.	*	24/12/2017
As a developer, I need to implement the ideas from my research on how to determine if cells are infected or clean.	****	07/01/2018
As a developer I need to find a way to segment the cells that are being over detected because of overlapping of the cells in the images.	****	07/01/2018
As a developer, I need to create an algorithm so when an image is provided to the program, the number of infected cells in the image are counted and the total is returned.	***	07/01/2018
As a developer I need to add a circle around the cells that have been determined as infected by the algorithm produced.	***	07/01/2018
As a developer, I need to test the algorithm on all of the images in my test set.	**	07/01/2018
As a developer I need to comment and refactor my code, this is to ensure it is easy to understand if somebody else needs to understand it in the future.	**	14/01/2018
As a developer I need to research ways of training my own Haar Classifier to detect the infected cells.	**	14/01/2018
As a developer, I need to create two image sets, one with positive images containing infected red blood cells and one with negative images containing anything but infected red blood cells.	**	21/01/2018
As a developer, I need to find all of the coordinates of each infected red blood cell on every image. This must then be added to the positive image set file next to the specific images location.	***	21/01/2018
As a developer, I need to create a file containing all of the negative images locations.	*	28/01/2018
As a developer, I need to use the positive and negative files to create a .vec file which can be used to train the classifier.	**	04/02/2018
As a developer, I need to train the classifier using all of the files on images previously created.	****	25/02/2018
As a developer I need to ensure the classifier detects the infected cells as accurately as possible. To do so I may need to add certain filters to the images before detection to make the detection process easier for the classifier.	****	25/02/2018
As a developer I need to alter my code to ensure images are processed according to their brightness. I must also output the number of infected cells in each image.	****	25/02/2018
As a developer, I need to test the classifier on a new set of images to ensure it detects all infected cells.	***	25/02/2018
As a developer, I need to create a cell cropping tool for both positive and negative cells. This will allow me to train and test the neural network automatically.	****	04/03/2018
As a developer I need to understand what a neural network is and how it works.	***	04/03/2018

As a developer, I need to install all of the necessary programs needed for TensorFlow to work.	****	04/03/2018
As a developer, I need to learn how to use TensorFlow and its wrappers so that I can train my own dataset later on.	****	04/03/2018
As a developer I need to prepare all of my data for the training of my CNN.	***	04/03/2018
As a developer, I need to create the code which contains the model architecture to train my neural network.	****	05/03/2018
As a developer, I need to train my neural network.	****	05/03/2018
As a developer, I need to test my neural network on my test set.	****	05/03/2018
As a developer, I need to retrain my models with augmentation applied to my images.	****	04/03/2018
As a developer, I need to test the models trained with augmented data.	***	04/03/2018
As a student, I need to create a poster describing my methods and why they are necessary. The poster should be understood by those without a technical background. To go alongside this, I also need to complete the abstract to draw people to my poster.	**	07/03/2018
As a developer, I need to retrain my neural network by balancing the number of training samples and altering the model architecture.	****	18/03/2018
As a developer, I need to test my trained models to ensure they work as intended.	***	18/03/2018
As a developer, I need to evaluate the accuracy of my neural network.	****	25/03/2018
As a developer, I need to evaluate the accuracy of my Haar Cascading code.	****	01/04/2018
As a developer, I need to evaluate the accuracy of the Image Processing code.	***	01/04/2018
Before evaluation on the test images, as a developer I need to check and refine any code to ensure I attain the best accuracies.	****	01/04/2018
As a student, I need to plan out the layout for my final document to ensure it is correct before producing it.	***	08/04/2018
As a student, I need to write my final report containing all technical details about all methods implemented.	***	27/04/2018

**Table ii: Product backlog used for the production of the three programs, difficulty can be read as \* - easy, \*\* - moderate, \*\*\* - difficult and \*\*\*\* - highly technical**