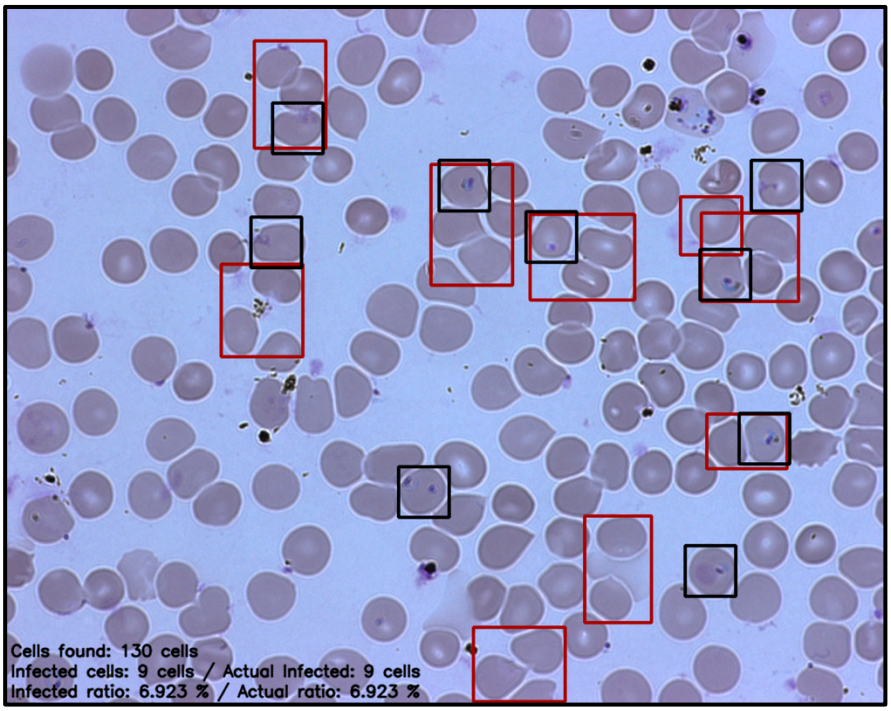
ASSESSING BLOOD SAMPLES WITH MALARIA
CE301 – FINAL PROJECT



NAME: EURICO PINTO

REGISTRATION NUMBER: 1502342

DEGREE COURSE: BENG COMPUTERS WITH ELECTRONICS

SUPERVISOR: DARIUSH MIRSHEKAR-SYAHKAL

SECOND ASSESSOR: DAVID BEBBINGTNON

DEPARTMENT: COMPUTER SCIENCE AND ELETRONIC ENGENEERING

UNIVERSITY: UNIVERSITY OF ESSEX

University of Essex

Acknowledgements

Firstly, I would like to acknowledge the importance of my supervisor Dariush Mirshekar-Syahkal which helped me a lot throughout the project and was always available to meet me on a weekly basis. We would meet to talk about present problems/solutions that I could have been facing during that time. It was also really rewarding because it helped me being constantly showing new work, methods, even solutions every single week. Secondly, I also want to thank Adrian Clark for letting me take part on this project and I would like to thank his help on some other more technical issues that appeared. Most of the help came indirectly from its Computer Vision lectures and Lab scripts that increased my interest in this field and also offered me motivation to want to understand more about it.

Abstract

In places where a lack of medical professionals exists, Malaria treatment is brutally influenced, the necessity of fast actions leads to deficient treatment. A crucial solution, presented in this project, is an automatic system of infected/uninfected cells recognition, to improve treatment efficiency. The recognition system is delivered as a mobile app, where the camera or gallery photos are used and analysed. The medical professionals would use their mobile phones to decipher the ratio of infected cells in an individual. A new way to prescribe a suitable medicine dosage. The photos in question should be taken from a microscope since this type of images were the ones used for training the recognition system. The project started with different training tests in Python using OpenCV library, segmenting the blood sample images, to detect each cell, also, segmenting separated cell images for Haar Cascade Classifier training – a key element to the project functionality. When results were satisfactory, a type of translation to Java in Android Studio was made to function as a mobile app. Throughout time, the number of correct detections increased, in the other hand the number of false detections decreased – mostly related to better image segmentation and a more efficient classifier. Further work is necessary to have results between 90%-100% detection efficiency. A new classifier trained by more images and new segmentation techniques applied to blood sample images will increase the ability for the system to better perceive it.

Table of Contents

Acknowledgements ***2***

Abstract ***3***

List of Symbols ***5***

Introduction ***6***

Computer Vision and Malaria **6**

Literature search ***7***

Malaria **7**

Haar Cascade Classifier **8**

Successful Cases **8**

Computer Vision **9**

Computer Vision history 9

Computer Vision overview 9

Computer Vision techniques 10

OpenCV **15**

Objectives ***16***

Goals with the Cascade Classifier **16**

Goals with Python **16**

Goals with Android App **17**

Design and Implementation ***18***

Cascade Classifier **18**

Python **22**

Android App **24**

Results ***26***

Presentation **26**

Blood samples examples 26

Explanation and Ethics **28**

Project planning ***29***

Overview **29**

Jira **30**

GitLab **31**

Tables, Graphs and equations ***32***

Conclusions ***33***

Discussion **33**

Conclusion & future work **33**

References ***35***

Appendices ***38***

List of Symbols

PI – Positive Images

NI – Negative images

HCC – Haar Cascade Classifier

TP – True positives

TN – True negatives

FP – False positives

FN – False negatives

LBP – Local binary pattern

Introduction

Computer Vision and Malaria

Computer Vision is the scientific field merging the power of computation with the human capability to see and perceive images as matter of data. It focuses on getting information from images or videos from the real world, likewise technological human eyes, while trying to understand the real world, storing relevant information based on those senses that can use to perform certain actions [3]. Computer vision can offer the possibility to improve the way humans use vision, by automatizing some of its methods in the real world and therefore solve world scale issues. This project aims to present exactly this by using Computer Vision latest mechanisms and innovations to present a different solution for Malaria diagnosis and treatment. Malaria is a well-known lethal parasite, most common in countries where medical treatment still faces a lot of challenges. At places where medical staff is short, medical prevention does not present a long-term solution, the lack of time forces health professionals to give incorrect medicine dosages to patients. Medical professionals in these areas do not have the time to accomplish a complete analysis of the blood samples they collect, therefore, its actions are not based in complete information from the state of the patient disease. Computer Vision takes in to present a more automatized solution for the Malaria detection/diagnosis and also a better approach to its treatment. Increasing the automatization of the detection process can mean a great step into the challenge of Malaria treatment. Mobility has to be taken into consideration, for health teams to be able to move around with the prototype and being able to perform an accurate diagnose. The solution demonstrated in this project is a mobile application for an Android device that can automatically receive an image from a medical professional and rapidly output an image with all infected cells detected by its mechanisms, squaring them with red colour. Alongside it will show the number of total cells, infected cells and the ratio of infected cells against total cells. The ratio offers the possibility for medical professionals to have a faster notion of how serious a diagnosis is and what is the best way to deal with it. The output image should be saved into the gallery so that the medical professional can proofread it afterwards to check if everything seems accurate. Apart from the mobile application, most of the tests are run under Python software which also enables the possibility to use a computer to perform infection detection, though the main task focuses on portable devices capable of performing this detection. In the

end, it is expected that this application can increase efficiency and precision of Malaria detection, save medical professionals time and most importantly save lives.

Literature search

Malaria

Malaria Overview

Malaria is a lethal tropical disease caused by Plasmodium parasites being these spread through the bites of contaminated female Anopheles mosquitoes. The two most dangerous species are *P. falciparum* (most common in Africa) and *P. vivax* (most common outside Africa). Although Malaria is preventable and curable it is also responsible for more than 400,000 mortalities each year and this epidemy is also enforced by the World Health Organisation by stating that half of the world population is at risk of Malaria. Likewise, 70% of Malaria related deaths occur in children under 5 years old, having a devastated statistic of one mortal case every 2 minutes. However Malaria has a catastrophic overview, there is still hope in terms of saving lives, since 2000 the number of deaths have been reduced in half, mainly caused by external funding and intervention [21].

Malaria Treatment

The most common symptoms appear after 10 to 15 days, indicators vary from fever, headache to chills, which are too difficult to distinguish as symptoms from Malaria infection. Given that it is too difficult to know if someone is infected based on the patient's symptoms, the World Health Organisation conduct of treatment enforces that a patient with suspected malaria should have a parasitological diagnosis before receiving any treatment [1]. The two main causes for that are to prevent treatment failure but also to not expose parasites to anti-malarial drugs so they are not able to develop drugs resistance [2]. Most of the diagnosis, for lacking of time by the practitioners, are not conducted this way and so as stated by WHO a large number of patients are inaccurately diagnosed with Malaria [2].

In order to improve the patient diagnosis in areas where Malaria is expanding and medical professionals do not have time to accomplish the most accurate diagnose, a computer vision diagnose seems to be the proper solution. The main goal from a computer vision perspective

would be to predict from a smear blood sample the proportion of infected red blood cells. Nevertheless, the counting of uninfected and infected red blood cells should be achieved too. It is presented by studies that the morphological computer vision practices can effectively be applied to perceive malaria infected parasite in blood smear images [7].

Haar Cascade Classifier

A cascade classifier tells OpenCV library what features to look for in a given image. At the moment, there are thousands of cascade classifiers, commonly focused on recognizing faces or eyes. In this project, the goal is to let OpenCV recognize a specific object relevant for this matter – infected cells, for that a cascade classifier is needed to tell OpenCV how to recognize a cell that is infected with Malaria. It is possible to generate a specific cascade classifier with Haar or Local Binary Pattern features, in which OpenCV is called to train a cascade classifier. In order to train a classifier, samples are needed, a lot of images showing the object that is wanted to be detected (infected cells) – these are called positive samples or positive images. In the other hand, more images without the object - negative samples or negative images – these would be images of single uninfected cells. The number of images depend on the quality of the images, the object to be recognized, the method to generate the samples and the CPU power [9]. Generating a highly accurate cascade classifier takes a lot of time and a huge number of samples, for example, the classifiers used for face recognition: *"It is unclear exactly how many of each kind of image are needed. For Urban Challenge 2008 we used 1000 positive and 1000 negative images whereas the previous project Grippered Bandit used 5000. The result for the Grippered Bandit project was that their classifier was much more accurate than ours."[9].* The number of samples has a great impact in accuracy though it increases time consumption, around weeks to finish a proper training.

Successful Cases

This section aims to demonstrate how other similar projects did their approach on various topics around computer vision and infected cells detection. All of them are examples taken from published papers on an academic basis, but also regarding Malaria automatized detection as the main argument [7][17][18]. This background reading enabled me to get more in touch with different techniques that are available regarding blood sample segmentation

and image processing, also distinguished ways in which an infected cell can be actually classified and detected. New methods around clustering and feature extraction were used to correctly detect malaria parasites inside blood cells [17], they also show important measures to be careful when calculating, specificity and efficiency [18].

Computer Vision

Computer Vision history

Moving forward, Computer Vision had his first start in history around the 1970's, viewed as: "the visual perception component of an ambitious agenda to mimic human intelligence and to endow robots with intelligent behaviours" – described by R. Szeliski [3]. At that time, the difference between this field and Digital Image Processing was that it recovered the 3D structure of the world from images and used it towards full scene understanding [3].

Computer Vision overview

After four decades, Computer Vision is much different and complex, it is part of our daily life, from phone camera's recognizing our faces, to be totally possible for cars to be intelligently autonomous. The main focus in Computer Vision is to interpret images in the way humans do so effortlessly, achieving that by having researchers developing new mathematical techniques to understand the way a 3-dimensional shape and appearance from an object is distinguished [3]. It is also the computerized extraction of information from images, this information being 3D models, camera position, object detection and recognition or grouping and searching image content [5]. Vision is one of the most difficult components in Computer Vision, mainly because it is based on the premise that it is an inverse problem, so it is always trying to specify a specific solution while receiving insufficient information. These specific solutions decipher difficult nature problems that have a significant impact on our society. On the Statistics side of Computer Vision, probability distributions can be used to model the scene but also to select the best algorithm to better perform a certain action. Finally, Computer vision tries to mimic human vision, sometimes using a data and statistical approach, other times using geometry as the recipe to solve problems [5].

Computer Vision techniques

While diving into different techniques on image processing, segmentation, detection, tracking and classifying, the most correct way to choose the right one for this project was to gather information about more than one before jumping right on to the first that appeared. All these stages are used nowadays to detect autonomously infected cells, 1st - Data Pre-processing, 2nd - Candidate Selection and 3rd - Segmentation, Feature extraction and Classification [7]. The next part will focus somehow on that, on the various different topics, from processing an image to matching its important locations.

Image processing

The first stage is called image processing, focused on editing the images that are going to be worked on in the future. There is no right or wrong processing, since there are lots of ways of processing an image and obviously all images differ. The clearest way to process an image is to reverse thinking about it, therefore, if the final goal is known, the way the image needs to be processed is much easier to accomplish, for example: if cell detection is a priority, then the edges of each cell should be well perceived. Image processing can have multiple functions, it can correct exposure, balance colours, reduce image noise or increase sharpness, and it is crucial for computer vision to perform accurate acceptable results. Kalman filter is recommended to eradicate the noise from the data [7].

In image processing there are point operators, in which each output pixel's value depends on the corresponding input pixel value. The different types of point operators are pixel transform – a function that takes one or more input images and produces an output image. Colour transforms – it manipulates colour for a better image visualization. Then, composing and matting that cuts an object from its original background (matting) and then uses it on another image or background (composing). Lastly, histogram equalization brightens some dark values and darkens some light values using an intensity mapping function [3].

There are also linear filtering processes that differ from point operators in the way that it uses a collection of pixel values around the proximity of a given pixel to determine its output value [3]. This process is responsible for tone adjustment, soft blur, sharpen details and accentuate edges (useful for cell detection) and noise removal. The different types of linear filtering are: padding, where new boundaries from boundary effects are removed; separable filtering

which separates convolutions as vertical and horizontal. Next, band-pass and steerable filters – it smooths filters (band-pass), subsequently convolves them with different rules and later steers the filter (steerable). Recursive filtering is other technique that its values depend on previous filter outputs, and there is bilateral filtering – although slow, output pixel's value depends on weighted combination of neighbouring pixel values [3]. Finally, convolution is the process of applying a filter to reduce image noise by replacing each pixel with a weighted average of its neighbours, main effect is smoothing or blurring [4].

Image segmentation

The next process after image processing is image segmentation defined as the collection of pixels or pattern elements into summary representations that show important image properties [4]. Furthermore, is the practice of subdividing an image into significant regions, being foreground against background or individual objects in the image, raised using characteristics like colour, edges, neighbour correspondence [5].

Two different examples of segmentation are: Background subtraction – if anything does not seem as a background it is an important matter, frequently used in malaria detections [7]. On the other hand, shot boundary detection in which changes in video frames are important to take into account. Image segmentation can be done by clustering pixels, a process where a dataset is replaced by clusters (sets of data points that belong together) [4]. Clustering can be used for recognition, data sets of images division, organization, navigation, but also for visualizing sameness between images [5]. Each image pixel is represented by a feature vector that stores all the measurements describing a pixel (intensity, location, colour). Every feature vector belongs exactly to one cluster, and each cluster represents an image segment. An image is obtained by replacing the feature vector at each pixel by the number of that feature vector's cluster in the centre [4]. Clustering Algorithms are: Agglomerative Clustering where each point is a separate cluster, until the clustering is satisfactory, then the two clusters are merged with the smallest inter-cluster distance. On the other hand, the divisive clustering – a single clustering contains all points, until clustering is satisfactory, then split the cluster that yields the two components with the largest inter-cluster distance. To end with, mean shift – there is one mode for each data point, modes are different but very tightly clustered, each kernel estimate should be very close to the actual set of modes and finally, each pixel is replaced by its mode representation [4].

Feature detection and matching

Moving to the next important topic in Computer Vision: Feature detection and Matching, essential component of many computer vision applications. It focuses on keypoint features/ interest points – specific locations in images and edges (object boundaries). In feature detection and matching field, point features are useful to find sets of corresponding locations in different images. There are two main approaches: 1st (more suitable for nearby viewpoints) it finds features in one image that can be accurately tracked using a local search technique (like correlation or least square). The 2nd (more suitable for large amount of motion) independently detects features in all the images, then match features based on their local appearances [3]. There are four important stages in keypoint detection and matching pipeline - 1st. feature detection, 2nd. Feature description, 3rd. feature matching, 4th. Feature tracking.

In Feature Detection, the image is searched for locations that are likely to match well in other images, although it cannot know which other image locations will match. Usually textureless locations are impossible to match but high contrast is highly detected and conclusively matched. Next, Feature Descriptors uses matches of detecting features that determine which features come from corresponding locations in different images, then extracts a local scale, orientation, affine frame estimate and finally uses it to resample the patch, before forming the feature descriptor. Two common examples are MOPS (Multi-Scale Oriented Patches) – that compensates slight accuracies and SIFT (Scale Invariant Feature Transform) – one of the most successful local image descriptors [5] that reduces the influence of gradients far from the centre [3]. SIFT methods can be used using OpenCV library [6].

The third stage is Feature Matching, where after features descriptors are extracted, it establishes preliminary feature matches between images. Uses matching strategy – which sends for further processing and conceives efficient data structures and algorithms to perform this matching the fastest way possible. In feature matching, ROC (Receiver Operating Characteristic) can be used to predict the accuracy of the match but efficient matching has a problem – it is impractical since it compares all features against all other features of potential matched images [3]. The solution would be indexing structures that rapidly search for aspects near a given feature (like multi-dimensional hashing) [3]. At last, Feature Tracking, finds a set of identical feature locations in a first image and then search for their corresponding locations in subsequent images, but is mostly used for video recognition [3].

Edges are also crucial in order to perform feature detection and matching, mainly because edges occur at the boundaries between regions of different colour, intensity and even texture [3]. The best way to classify it, is to express that it is a location that comprises an extreme intensity variation.

The field feature detection and matching is also truly related to object recognition, being one of the most crucial topics in computer vision. Object recognition is the ability of a computer to understand wetter it does or it does not recognize an object. It can be divided into instance recognition – it recognizes 2D and 3D rigid objects, or class recognition - it recognizes any instance of a particular general class [3]. The process of object detection holds onto some different steps, being those: extracting a set of interest points in each database image, storing the associated descriptors in an indexing structure, at recognise time, features are extracted from new image and compared against the stores object features. If most match, a second round of match verification takes place to evaluate its consistency. There is a problem though, using large databases takes time for feature matching, so to overcome this obstacle invert indexing would be a suitable solution to perform quick findings [3].

The sliding window method is used for well-behaved appearance objects that do not deform [4]. The algorithm is based on some actions – it builds a dataset of labelled image windows of fixed size (n*m), this dataset contains positive examples (large and centred instances of the object) and negative examples with none instance. Then, it trains a classifier to discriminate these windows and passes every window in the image to the classifier. The classifier is either positive if it contains the object and negative if it does not. But there is a scale problem, different sizes of the same object can trick the method, to solve that a Gaussian pyramid of the image is prepared. Search windows in each layer of the pyramid takes place. If window is scaled by ‘s’ on x*m window it searches (s*n) * (s*m) windows. Other problem is the overlapping problem – where multiple parts from one object can be positive and sent as multiple objects. The solution for this misconception would be non-maximum suppression – in which a window with a local maximum of the classifier response suppress nearby window [4].

K-nearest Neighbours Classifier is one of the simplest and most used methods for classification, in which the algorithm equates an object to be categorised with all objects in a training set with known class labels and allows the k nearest vote for which class to allocate. Alike k-means clustering algorithm, the number k needs to be selected before-hand. In

addition, the system requires the entire training set to be stored as a consequence, so if it is too large it will be slow doing search. To avoid this some form of binning is usually used to reduce the number of comparisons [5].

OpenCV

Computer Vision can be practiced in a vast majority of programming languages, the one used in this project will be Python, although the final usage of it will be on mobile applications, which require Android running in Java. One of the best libraries that can be used along platforms are 'OpenCV', 'Numpy'[19], 'Scipy'[20] and 'PIL' [5][6]. Most of them are available to Android and iOS and have a wide range of tutorials and documentation on how to process and segment images, execute object recognition or matching, and also some components on machine learning [6]. Crucial methods used on the project are possible thanks to this extraordinary and complete library, from applying correct threshold to find the cell contours.

PIL is the Python Imaging Library which offers image handling and lots of other useful basic image operations: resizing, cropping, rotating, colour conversion [5]. Moreover, NumPy is a package used for scientific computing with Python which has a number of useful concepts: array objects and linear algebra functions. The array object enables to do different operations: matrix multiplication, transposition, solving equation systems, vector multiplication, and normalization. Those operations are useful to align images, model variations, classify, warp and group images [5]. SciPy is a package for mathematics built on NumPy that provides efficient operations like numerical integration, optimization, statistics, signal processing and image processing [5].

OpenCV, Open Source Computer Vision Library [6] is a C++ library with elements that cover many areas of computer vision, comes with functions for reading and writing images as well as matrix operations and math libraries. In OpenCV, images are not saved using the standard RGB colour channels, instead they are reversely stored in BGR order. Colour space conversion are done using the function cvtColor(), for example if converting to grayscale - cv2.cvtColor(image,cv2.COLOR_BGR2GRAY), an important procedure to edge detection [5].

Objectives

Goals with the Cascade Classifier

There are different methods being used on this project regarding the process of detection: image processing, segmentation, cell contours recognition and Cascade Classifier, all of them being a main factor to achieve a parasite detection top-model. The cascade classifier process aims to understand if a single cell is infected, the training of a classifier is crucial to an efficient classification of cells. While the classifier is training, it passes through a lot of stages in which saves complex characteristics present on these images, these similar characteristics are essential to later understand which cells are infected and which ones are not infected. Image segmentation is also important to work on single cell images, to turn it similar to those used to train the classifier. The perfect example is when some cells are retrieved with a light blue background though the background of single cells used to train the classifier is white. This means that for a better detection the image should always have a white background and that is what segmentation will perform on those single cells. Finally, the goal of the cascade classifier in this project is to perform a long-term correct detection of infected cells with Malaria, this goal involves the time to also train this classifier in a complex and complete way that it can perform accurately in the future.

Goals with Python

The work done under Python had the main function to run different tests under blood sample images. The first one, tried to turn the black background from the single cell images, retrieved from the USA dataset, to a lighter one. These single cell images would then be used to train the Haar Cascade Classifier. Since the background colour of actual blood samples is usually a light colour, using a lighter colour on the single cells is crucial for a future efficient detection. The second goal of the process was based on the blood sample images, where these needed to be segmented and then, using the trained Cascade Classifier, acknowledged the position of the actual infected cells. This stage starts with some contours detection to have the coordinates of every single cell present on the image. Basically, turning the image to a grey scale and then applying a threshold [10] to get a black and white image, where there is a clear distinction between cells and background. With this image it is much easier to find the cells'

contours and therefore get its coordinates on the image. When all contours are detected, there is a while loop running through them to check one by one if they stand for infected or non-infected. Before running the cascade Classifier on the single retrieved cells, some segmentation takes place. This is mainly to perform a similar action as told before of changing the colour of the background to white to match the training of the Classifier. When is all set, it is time for the Classifier to intervene and decide wetter the cell is infected or not, in the case that it is, the coordinates are saved to an array that only stores infected cells. In the end these are printed along with actual infected cells retrieved from a .gt file delivered by London, to then be able to understand if the results were positive when compared.

Goals with Android App

The Android App uses Java instead of Python, which means that the code from Python has to be slightly modified in order to achieve the same results. Apart from that there is also the layout component associated with it, a user interface capable of letting the user take a picture or retrieve one from the gallery. The main goal with the Android Application would be to have this user interface that selects the photo to be processed by the program, having similar methods as used by Python program. The app should be quick informing the user the final result with the detected infected cells squared in the same image. It should also state the total number of cells, infected cell and the ratio between infected and total cells. This presents the last step to achieve a viable final product, the application should be capable of getting a photograph from the phone camera or a photograph from the phone gallery. Next, the process should be able to get this picture and work with it, starting with detecting every single cell so that it can evaluate each one with the trained classifier. In the end, the output has to be the original picture with infected cells squared in red, along with total cell count, infected cell count and infected ratio.

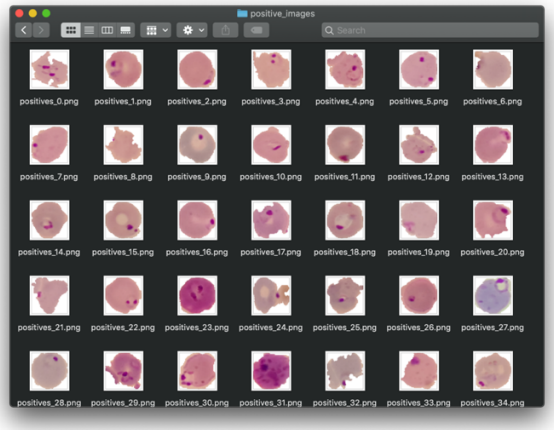
Design and Implementation

Cascade Classifier

For this project, the usage of a specific classifier is the key factor to a successful detection process, but even though there are around 20,000 images for PI and the same for NI, it would be too time consuming to get that to work. For this project, the training uses around: 80 positive samples and 600 negative samples, the feature type of the classifier is HAAR, which is extremely time consuming but acknowledges loads of image features, and for that reason the number of images had to be lowered down. Below there is the process of training the actual cascade classifier under the terminal.

Positive Images:

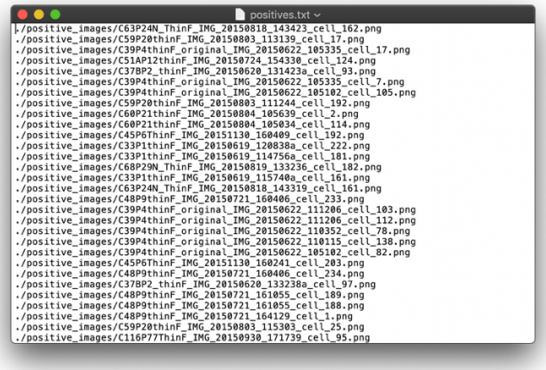
80 Images from USA dataset, all cropped with background (previously black), showing an infected single Cell with Malaria



Terminal command

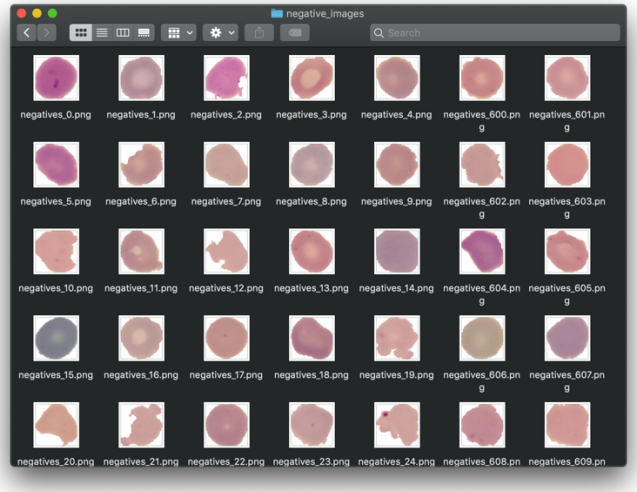
$ find ./positive_images -iname "*.png" > positives.txt

List all positive images in a txt file



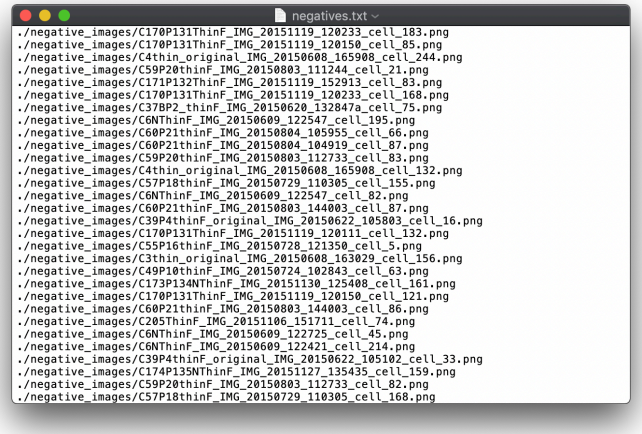
Negative Images:

600 Images from USA, all cropped with white background (previously black), showing non-infected single Cells.



$ find ./negative_images -iname "*.png" > negatives.txt

List all negative images in a txt file



Creating Samples:

With our positive and negative images in place, samples need to be created out of them. Positive and negative samples are used by the classifier, but the negatives are already done. To quote the OpenCV documentation [6] about negative samples: "Negative samples are taken from arbitrary images. These images must not contain detected objects. Negative samples are enumerated in a special file. It is a text file in which each line contains an image filename (relative to the directory of the description file) of negative sample image.". Meaning negatives.txt will serve as a list of negative samples, but positive samples are still needed and

there are a lot of different ways to get them which all lead to different results regarding the accuracy of the trained classifier.

There is a tool from OpenCV called opencv_createsamples, this tool offers several options as to how generate samples out of input images. In the end, the method will output a .vec file useful to train the cascade classifier. The tool opencv_createsamples generates a large number of positive samples from positive images, by applying transformations and distortions with other negative or positive images. This is helpful to increase the number of positive samples without actually having loads of them. Naotoshi Seo [8] wrote some useful scripts to help when generating samples. The first one is createsamples.pl, a small Perl script, to get 600 positive samples, by combining each positive image with a random negative image and then running them through opencv_createsamples. An easier way to accomplish the final positive samples necessary to start the training of the cascade classifier. The code below is run under the terminal and is the final stage to start the training process.

Terminal commands:

```
$       perl bin/createsamples.pl positives.txt negatives.txt samples 600\
    "opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\
    -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 80"
```

Focus on -w and -h, they should have the same ratio as the positive input images.

Final step, to merge all .vec files into one.
Terminal commands:

```
$       python ./tools/mergevec.py -v samples/ -o samples.vec
```

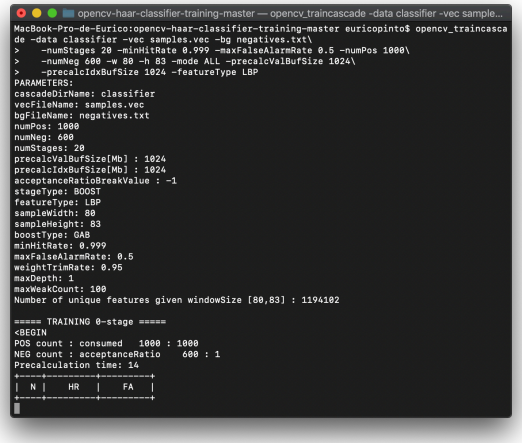
Start Training:

In OpenCV library there are different offers - two different applications for training a Haar classifier: they are called opencv_haartraining and opencv_traincascade. The more suitable for this project was opencv_traincascade since it allows the training process to be multi-threaded, reducing the time it takes to finish and is compatible with the newer OpenCV 2.x API. Cascade files generated by opencv_haartraining and opencv_traincascade differ in format. Many tools and libraries only accept opencv_haartraining to train the classifier but it is not a problem for this project. It is now time to point method opencv_traincascade at the actual positive samples (samples.vec) and negative images, telling it to write its output into

the classifier directory -classifiers directory, present in the repository and finally the sample size, its width and height. There are some parameters that need some specification: numNeg parameter specifies how many negative samples are going to be used on the training, then precalcValBufSize and precalcIdxBufSize specifiy how much memory is going to be used while the cascade classifier is being trained. Finally, numPos states the number of positive samples that are used on this training process. To start training the classifier with opencv_traincascade, which comes with OpenCV library, and save the results to ./classifier directory it just needs to follow the code below. The default type of feature type is HAAR but OpenCV accepts LBP feature type if the user wants the training to be faster.

Terminal commands:
opencv_traincascade -data classifier -vec samples.vec -bg negatives.txt\

-numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 1000\
-numNeg 600 -w 80 -h 83 -mode ALL -precalcValBufSize 1024\
-precalcIdxBufSize 1024 -featureType LBP



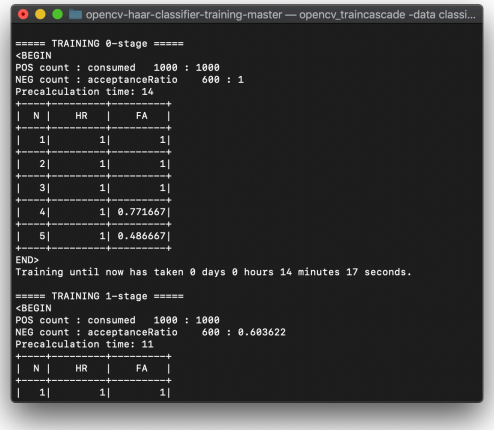


Figure 1- Training the Classifier under terminal

Figure 2 – Different stages of the training process

Python

The Python program responsible for most of the images testing is divided into 2 different files. The first one called delete_background.py with the only goal being to switch the dark background to a lighter one, saving the image result as an image file. On the other side, there is a file called main_program.py that performs every single test that was necessary, from segmentation to find the right number of contours.

The program to delete the background of cells was initially created to help segmenting the data necessary for the cascade classifier training. Since, usually cells have a light background colour the process to turn the background white was an important step to increase the efficiency of the classifier. There are two folders used, one has the unedited single cell images that have to be edited, the other folder receives the edited single cell. This program has to be executed two times, one for PI (infected cells) and another one to NI (uninfected cells). The image is threshold first to get the image divided into main images and background and eventually the idea is to change the pixel values of the background to white. Some bluring, kernel and erosion is used sideways to allow a better background removal. In the end, the images are all segmented and ready to be used for the Cascade classifier.

Next, the main program had the responsibility to, in the end, retrieve a final image with all detected cells squared. The process has different stages, the first one is about processing the image, increasing for example the contrast which offers better conditions for threshold and classifier detection. When the contrast is increased, the next step is to turn it to a grey scale image – needed for the threshold operation. The threshold operation is mainly used to get an image just with white and black colours, one representing the cells and the other the background. To find the best threshold value, there is otsu method that retrieves the best threshold value to achieve the perfect balance between black and white. The idea to use otsu is to avoid using actual final numbers, which would be a constraint since images can differ in lots of ways, especially in terms of colour. When otsu is applied, the image is now ready to be analysed and get all the contours founded in that image already with a threshold applied, using the findContours method it is possible to save all the coordinates of every single cell. A for loop is sequentially used to go through every single contour, that will be in the future used to understand if that cell is infected or not. Firstly, the cell area has to be acknowledged to take out those that are too big or too small, meaning basically that some contours could be from anything else but not a cell. After that, the actual single cell in that present moment suffers a background removal, similar to the last program technique, this has to be done to get a similar image as the ones used for the classifier training. When all of that is completed, the single cell is sent to another method – cell recognition; in this method the classifier xml file is loaded and then used as a classifier to analyse each single cell that is sent to the method. If the classifier retrieves something inside the cell, the coordinates of that cell are saved to an array so that they can be used in the end to square cells on the original image. While all of this is happening, there is also the record of the number of cells – being incremented every time a cell with a certain size appears in the for loop. The number of infected cells is retrieved from the size of the array saving all the infected cell coordinates, apart from that the ratio is calculated by dividing the number of infected cells by the number of cells. All of these values are printed on the actual images with the infected cells squared. The main program also offers the possibility to run the mechanism through 30 images available on a folder and save each of them with all the infected cells squared along with the cell records stated before. Some methods are also present in the main program, not to being used but to test the mechanism and check wetter another way of doing something could be better achieved.

Android App

The Android App Design and implementation has some similarities with the main Python program. From the beginning of the project, the work done under Python would be eventually used in the Android App as most of its methods are useful for the final viable product. The Android App has its logic programmed in Java and has a layout in xml associated with it. There is only one page during the all process, this page is the layout defined in the xml file. It contains a title, the application name, and a sub-title asking for the user to perform a task – take a photograph or retrieve one from the mobile phone gallery. The layout also has two button, with 2 different icons, one associated with a camera where the user can take a photograph and the other icon related to the phone's media, where the user can access its mobile phone photographs. In the end, below the two buttons there is a space for an image and text, which in the start of the program is blank, but after the program is run successfully will show the actual selected picture with all the infected cells detected squared and the text will show the results from that picture evaluation.

On the other side, there is the logic behind the layout, where all actions take place, everything happens inside home activity, the main activity for this application. It starts by defining some useful codes – picture request, gallery request and external storage code; those will be needed in the future to ask permission or to perform a certain action. The image and text that appear blank in the beginning are also started to let the future methods set results into them. This is followed by the onCreate method, where the application is first created, getting the main components present in the layout and setting onClick methods on both buttons from the layout. The onClick method symbolises what is going to happen if one of them is clicked by the user, since each one has a different way of performing, the activityResult method appears along with the codes that were saved in the past. The activity request has a request code as a parameter, meaning that if the camera button was clicked, the code would be from a picture request code, in the other hand if the gallery button was clicked the request code would be the one that requests gallery access. Going back to the onClick method, each button as a different method inside onClick, the camera button is sent to dispatchTakePictureIntent method, in this method the application checks if there is permission to access the camera and if true it opens the camera if false asks permission. The same happens to the gallery button, but instead the method is called dispatchGalleryIntent, which verifies permission to access

the storage, letting the user select an image. Jumping back again to the activityResult method, there are two if statements in there that differentiate the request code received as parameter. Both request codes used were defined in the beginning of the activity and are really useful to understand if the user selected the camera or the gallery. However, the actions on the picture retrieved from both methods are the same the process of retrieving them is different and for that reason, on this activity both options are divided into two different if statements. Inside the if statement, the activity has now the responsibility to get the image that the user selected from the gallery or that it took recently if the camera was used. This image is saved as a Mat Class, a type of Class useful to save images so they can be processed afterwards.

The process explained before, states the steps necessary to accomplish a single image chosen by the user, then it is time to focus on the actual image processing performances that have to take place. Most of the mechanisms used tend to be a reasonably copy of what was done under Python with sufficient results, this is possible thanks to OpenCV library that can run under Java as well, though it shows different challenges since the language changes a fair bit. Initially the image had to increase its contrast and had to be converted to a grey scale picture, in order to perform a threshold operation in the future. Threshold operation appears next to get an image that only contains two colours, black and white – truly efficient to differentiate between background and actual cells. The right threshold value is retrieved from otsu method, which is offered by Opencv library to find the right value to apply to a certain image and get the best balance between background and actual objects. After having the image in proper conditions, the program is then sent to analyse its contours and find what are the coordinates of every single cell present in that image. The contours are saved in a List of Mat points, each of them representing one cell in the image, to look over every single of them, a while loop was implemented to inspect every single cell that was saved. The first thing to be careful in this part is to check the contour area, since some of the contours found could just be something but a single cell, the best values encountered to limit the area values were a minimum of 2,000 and maximum of 12,000. When this simple evaluation was completed, the application would focus on analysing each single cell with the already trained classifier, but before applying the classifier the background of the cell should be cleared to a white colour. When the single cell is already segmented and has a white background, it is sent to the method that performs the evaluation of the cell, starting by loading the cascade classifier, an

xml file saved in the application resources. The classifier checks inside the single cell frame to see if it finds any infected cell. If the frame is detected to be infected, its coordinates are saved into an array that will be used in the end to perform the drawing of squares around all detected infected cells. If it is not that case, the application just jumps to the next single cell that is present on the List of Mat points. When all single cells are run over, the program has now an array containing all the infected cell coordinates and can perform the drawing of squares around them in the actual picture, firstly input by the user. Along the picture, 3 different String are set in the main layout of the application, the first stating the total number of cells found, the second presenting the number of infected cells that were detected using this mechanism and the last stating the ratio between the number of infected cells and the number of total cells.

Results

Presentation

The presentation of this project results aims to demonstrate the different outputs throughout the different stages that were taken under the development of the main mechanism, but it is also to show how is the actual mobile application and the main Python program working and what they can perform. There are different results for Python and for the Android App, these results were gathered by analysing 30 different blood samples that contain infected cells with Malaria. The presentation of results aims to present the efficiency and precision of the program in both platforms to have a clear sense of how reliable the application can be in the real world.

Blood samples examples

Throughout time, different results were being obtained from the main program in Python and from the Android mobile application. Loads of factors contributed to that, the way the Cascade Classifier was being trained but also the methods used for segmenting the blood sample images. The results around the 30 blood sample images are divided into two sections: one regarding the performance of the main Python program and the other focusing on the actual final product – the Android mobile application.

Python

Results under Python program helped understanding how different factors affected the right implementation of the cascade Classifier detection process. After several results under the same blood sample images it was possible to recognize what were the actual problems with Cell detection. The efficiency increased when the correct background switch was applied, turning the actual dark background of a single cell to a white one. Another way of improving the results, was the correct implementation of threshold to efficiently detect each single cell present on a blood sample image. This was possible by avoiding final numbers as threshold values, so that images with other tons of colour would also perform the threshold correctly. Each single cell retrieved by the right threshold has to be segmented properly before running through the Classifier. The segmentation process helps the single cell images being similar to the ones used to train the Cascade Classifier. Finally, Python was a helpful tool to work around OpenCV, being easily manipulated to execute different tests on blood sample images. The final results regarding its efficiency and precision are available in the table below.

Number of samples	True Positives	True negatives	False Positives	False Negatives
30	48	3777	107	180

Precision = $\frac{TP}{TP+FP} = \frac{48}{48+107} = 0{,}309 = 31\%$

Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{48+3777}{48+3777+107+180} = 0{,}93 = 93\%$

Android App

The Android app results should be in principle a conversion of what was achieved in Python, the importance of these results is just to make sure that most performances from Python are correctly implemented on the mobile application. In Java, some methods are implemented in different ways because of the way an object-oriented language operates, for that reason some of the results can show differences in efficiency and precision. The android app shows some difference in terms of speed of its mechanisms to deploy an actual image and the camera of the android phone it is not truly reliable to take pictures from a microscope, being that said, the images used are usually retrieved from the gallery to have a proper image that

can be inspected. The results retrieved from the different 30 blood samples are used to calculate the accuracy and precision of the methods in the Android app, important to know how reliable it will be with other future images.

Explanation and Ethics

This section serves to present an explanation of the results of the Python main program and the Android mobile application presented above. The whole process till it finally achieves the set of all infected cells has complex stages. This complexity makes it able to have distinguished performances between those mechanisms. The results presented in terms of efficiency and precision are crucial to understand if an application has the necessary ability that can be used on an actual industry. This touches the ethics topic, where the application needs to offer confidence and safety for those who use it – medical teams, but not forgetting the key people that can be benefitted from this application - the actual patients. A wrong result signifies a wrong diagnosis, a diagnosis is the most important step to block a parasite from growing or spreading. The application needs to have high standards of results to avoid precarious diagnosis. The usage of this application should always be careful and as results can vary, medical teams should not use it as a final answer, but as a tool to helped them increase the speed of a correct diagnosis.

Project planning

Overview

Project planning was the best structured routine solution to accomplish the final result while being tremendously accurate on how processes should run throughout time. There were different stages during the conclusion of this project, they all concluded to be crucial for the successful achievement of the whole project. In the beginning, there was an abstraction about what the plan should actually be, the reason being the lack of full clarity about what needed to be done till the completion of the same. At this time, it was clear that most of the knowledge would come from articles touching topics about Malaria disease and its treatment, similar academic projects with successful outcomes and Computer Vision. The background reading in the beginning helped a lot to fulfil the empty notions about the whole project. After that it was much easier to draw a plan I could keep with, though some changes and risks were always known to happen in the future. These proved to be relevant for the health of the project since new errors were always appearing, it was strongly advisable that those were usually waited for, or that, a prior solution was being already established for that problem. Another hard matter was identifying and preventing future risks, the ones thought to happen were related to a deficient use of necessary technology or finding difficulties to convert methods from one programming language to another. Another risk was the time consuming of training a complete Haar Cascade Classifier, which the longer the list of images the longer the time to be completed. A good way of solving this time challenge was to be very careful when starting that process, being extremely preoccupied with all the prior processes before the train starts.

The plan consisted in firstly achieving some results under Python that would perform some detection although it could not be perfect. The main purpose was to at least achieve some detection performance, so the project would have some start point, where it could be used to further investigation and development. It was also established that after achieving some interesting results on Python, a type of conversion should be applied to the Android App so that on an early stage the project could have its first prototype, even if it were too simple.

Next, after both being completed, the plan was then to be continuously working on the Python program to improve its results throughout time, an important set of tasks to help with this was also planned to be constantly improving the necessary methods for the project.

Another task that was being done sideways was the development and efficiency increment of the Cascade Classifier for better detection results. Every time there was significant improvements on the Python programming detecting new cells and avoiding new errors, those new methods should be added to the Android application so that the workflow could be transferred to the end point. Moreover, sometimes project planning can get off the rails and if there is no backup solution ready to use some changes can be chaotic to the safeness of the project. To avoid this, most of the changes thought to happen were always thought to happen and theoretically designed to avoid chaos from happening.

The project planning proved to be an essential key for the well-being of itself, since it was well structured since the beginning it was much easier to implement new challenges but most importantly the capability to overcome difficulties and risks. When prevention is taken seriously the probability of the project flow correctly are much bigger. Additionally, it proved to be a great step to better acknowledge the way project planning needs to be conducted and how dependent the project can be from the planning. All of this was accomplished alone, personally an excellent experience to intensify the rewarding activity this project has been.

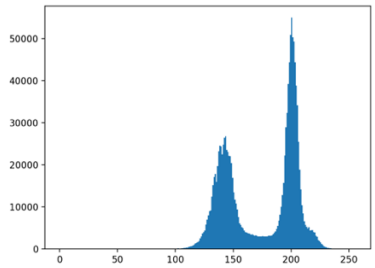
Jira

Alongside the project planning, Jira showed to be an extremely powerful tool for long projects, commonly used in large companies that have considerable multiple teams working on the same project. I firstly divided the big parts of topics that the project could have into epics, an epic to Python with Opencv, other to Android with Opencv, and an all section about training a classifier. The idea was to easily organise different parts of the project and then create sub-tasks, stories and bugs inside each one; usually tasks were used all the time to keep in records of what had to be done at a certain time. Bug reports also started appearing when the program started giving the same error a lot of times, so the bug report would be created to pay more attention to this in the future. Stories were the ones less used, usually because throughout the project it seemed that stories would not be that useful for the completion of the main tasks.

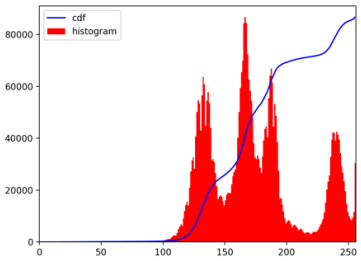
GitLab

Another tool used to keep track of the actual code being made was GitLab, a powerful management tool for code and to keep record of what the last steps and prototype of a projects was. GitLab was usually useful during the process of coding to avoid losing the code, it would be always saved on the GitLab 'cloud' with just a few commands on the command line. GitLab also helped when the process of coding got out of the rails and a back up was needed to avoid losing all the functions that have already been created, a way to avoid modification of being catastrophic.

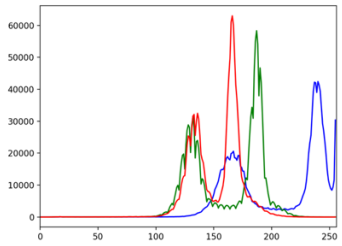
Tables, Graphs and equations



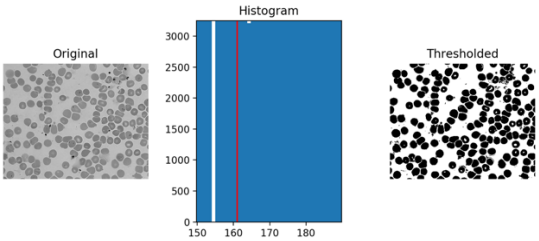
Graph 1. Histogram of blood sample image



Graph 2. Histogram along with brightness



Graph 3. RGB channels of blood sample image



Graph 4. Threshold value shown on an histograma

Histograms were abundantly used throughout the project, a good way to first interpret an image that is unknown, also important to improve the threshold on the images, getting the right threshold value based on the histogram graphs, this is key to achieve correctly each cell on the image with a proper crop. Histograms shows the number of pixels per each value of colour, for example on a grey scaled image, with just one channel, it would check how many pixels are in each values, from 0 to 255. This presents the way the colour of the image is dispersed in terms of colour.

.

Conclusions

Discussion

This section aims to discuss the conclusions of the project and overall the final project experiment and experience. The results shown to be improving through time most of times related to similar factors. Usually a better classifier, using more images at training stage, would perform better detections and would detect less false ones. Another example, when there is some contrast increment usually the number of cells detected as false would also decrease. Turning the dark background every time to a lighter one on every single cell that is detected showed to increase the number of accurate detections that were not possible before. During the course of this project, it was learnt that most of the crucial knowledge came from learn by doing, by trying, testing different methods to perform the same actions, evaluating its results and taking into consideration if some methods should be switched to more efficient ones. Overall, the project received interesting results from both Python and mobile application, though not perfect, they show that it is possible to use computation to understand infected cells in a blood sample, wetter using a computer or a mobile phone.

Conclusion & future work

To conclude, I believe this project was an extraordinary experience that enabled me to see the wide range of matters that computer vision can touch and eventually solve. On this specific example, the action of automatically detect infected cells on a certain blood sample image in a fast way to improve the medical assistance not available at some remote places. It is also interesting to conclude how without knowing which image is going to be taken from the mobile camera phone, the program is right now ready to perform its learned mechanisms and almost perfectly understand where the infected cells in that previously unknown image are. The number of images used on this classifier could be increased, since a more complete classifier will always perform more efficiently. At this point the classifier sometimes detects false images as infected and misses some of them, mainly related to poor segmentation, meaning the edition on the images should focus on increasing the similar characteristics that

are present on infected cells avoiding characteristics present on non-infected cells. For the future, the work needed would focus in the training of the cascade classifier, increasing the number of PI and NI but also using HAAR features, though it can take weeks to complete, which means more features are saved and that is important for the infected cell detection process. Another work could be done on implementing an iOS application for iPhone users, so It reaches more devices, meaning more people, but always keeping the same efficiency and precision in results.

References

[1] "Malaria", *World Health Organization*, 2018. [Online]. Available: http://www.who.int/news-room/fact-sheets/detail/malaria. [Accessed: 12- Oct- 2018].

[2] "Malaria", *Who.int*, 2018. [Online]. Available: http://www.who.int/ith/ITH_chapter_7.pdf. [Accessed: 12- Oct- 2018].

[3] R. Szeliski, *Computer vision - Algorithms and Applications*, 1st ed. Springer, 2011.

[4] D. Forsyth and J. Ponce, *Computer Vision - A modern approach*, 2nd ed. Pearson Education UK, 2012.

[5] J. Solem, *Programming computer vision with Python*, 1st ed. Creative Commons, 2012.

[6] "OpenCV library", *Opencv.org*, 2018. [Online]. Available: https://opencv.org/. [Accessed: 17- Oct- 2018].

[7] P. Pattanaik, T. Swarnkar and D. Sheet, "Object Detection Technique For Malaria Parasite In Thin Blood Smear Images", *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2017.

[8] University College London, *Dataset of Micrographs blood samples*. Available: http://vase.essex.ac.uk/malaria.tar.gz

[9] "Train you own OpenCV Haar Classifier", *Coding-Robin*, 2013. [Online]. Available: https://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html. [Accessed: 02- Feb- 2019].

[10] "Basic thresholding operations", *OpenCV*, 2011-2014. [Online]. Available: https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/threshold/threshold.html. [Accessed: 30- Sep- 2018].

[11] "Contours", *OpenCV*, 2018. [Online]. Available: https://docs.opencv.org/3.4.2/d4/d73/tutorial_py_contours_begin.html [Accessed: 05- Oct- 2018].

[12] "Contours Properties", *OpenCV*, 2018. [Online]. Available: https://docs.opencv.org/3.4.3/d1/d32/tutorial_py_contour_properties.html [Accessed: 10- Oct- 2018].

[13] "Cascade Classifier Training", *OpenCV*, 2017. [Online]. Available: https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html [Accessed: 23- Nov- 2018].

[14] "Cascade Classifier Training", *OpenCV*, 2019. [Online]. Available:https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html [Accessed: 23- Nov- 2018].

[15] "Photo Basics", *Android Studio*.[Online]. Available: https://developer.android.com/training/camera/photobasics [Accessed: 2- Dec- 2018].

[16] NIH, USA, *Dataset of single infected and uninfected cells, 2018*. Avaiable: https://ceb.nlm.nih.gov/repositories/malaria-datasets/

[17] N. Khan, H. Pervaz, A. Latif and A. Musharraf, "Unsupervised Identification of Malaria Parasites using Computer Vision", International Conference on Computer Science and Software Engineering (JCSSE), 2014.

[18] B. Srivastava, A. Anvikar, S. Ghosh, N. Mishra, N. Kumar, A. Houri-Yafin, J. Joel Pollak, S. J. Salpeter and N. Valecha, "Computer-vision-based technology for fast, accurate and cost effective diagnosis of malaria", Malaria Journal 2015.

[19] "Numpy library", *Numpy*, 2018. [Online]. Available: https://www.numpy.org/ [Accessed: 27- Oct- 2018].

[20] "Scipy library", *Scipy*, 2018. [Online]. Available: https://scipy.org/. [Accessed: 30- Oct- 2018].

[21] "Malaria – Strategy overview", *Gates Foundation*. [Online]. Available: https://www.gatesfoundation.org/What-We-Do/Global-Health/Malaria. [Accessed: 15- Nov- 2018].

Appendices

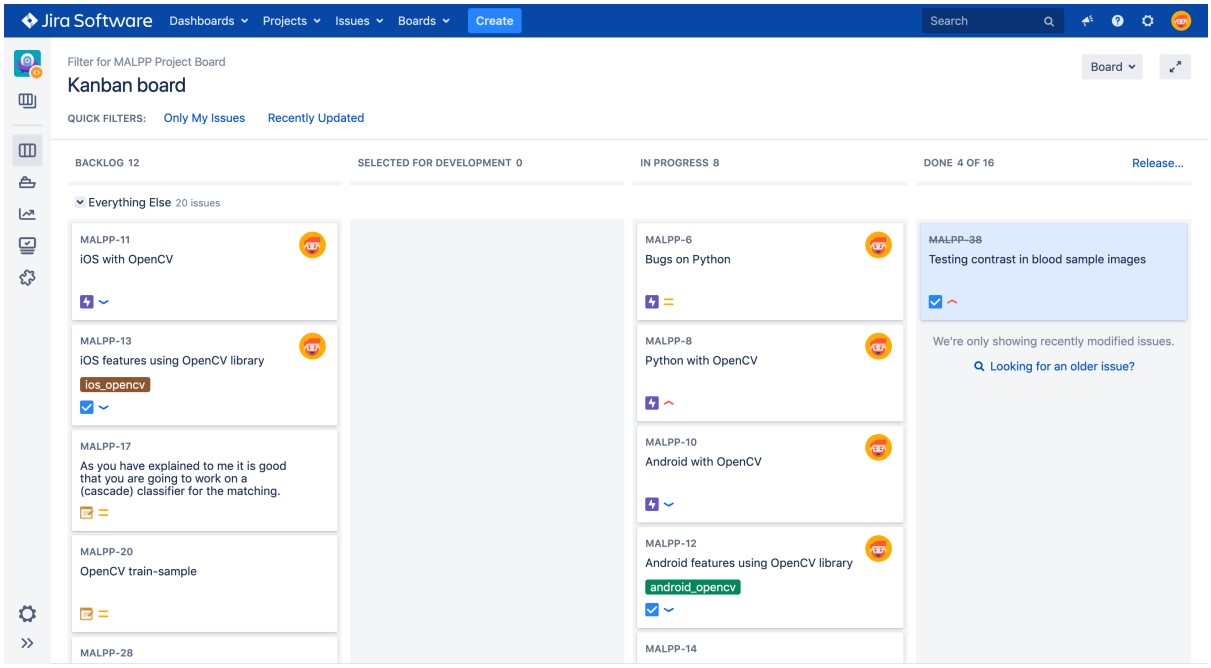


Figure 2 - Kanban board on Jira Software

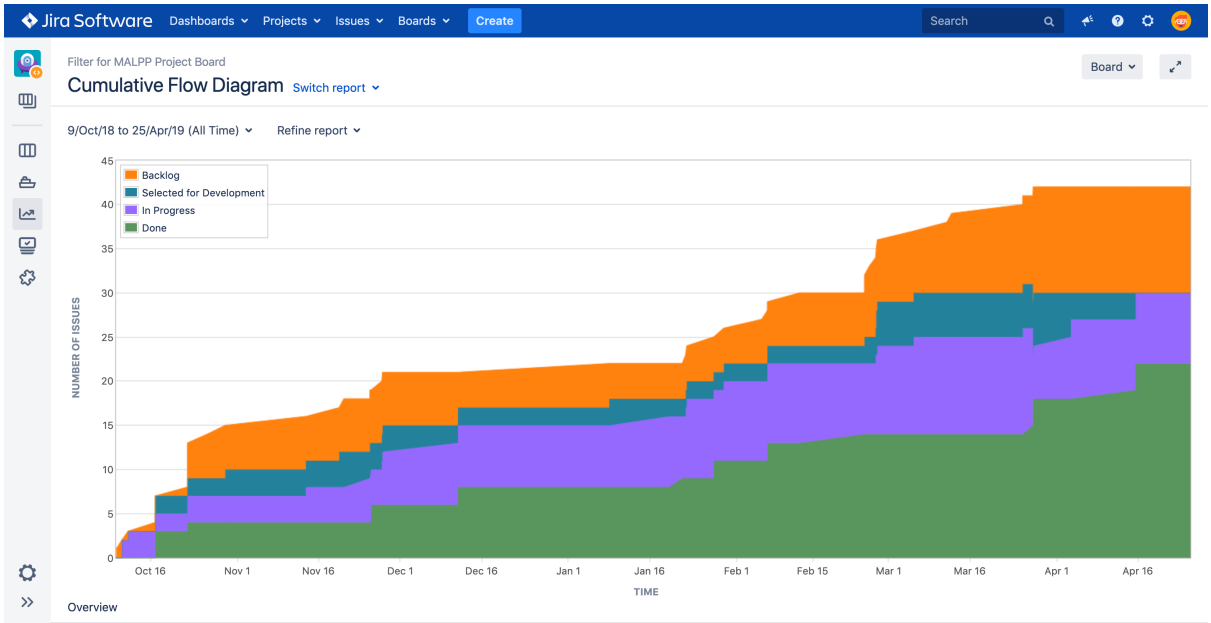


Figure 2 – Cumulative Flow Diagram on Jira Software

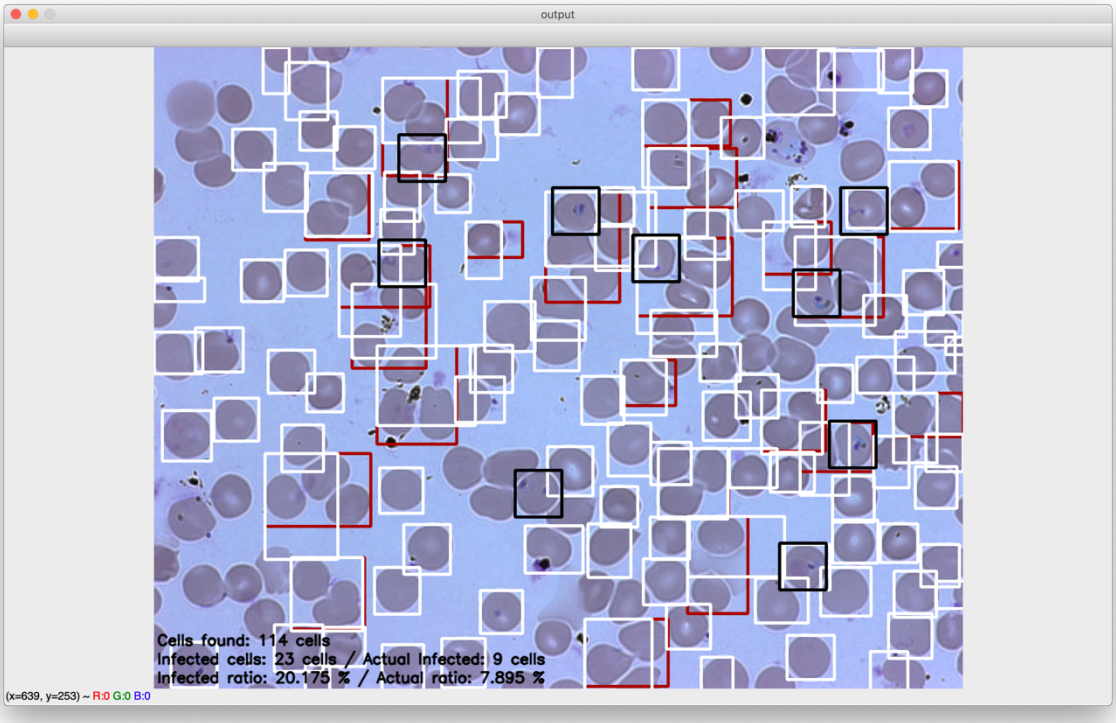


Figure 3. Detection process under Python

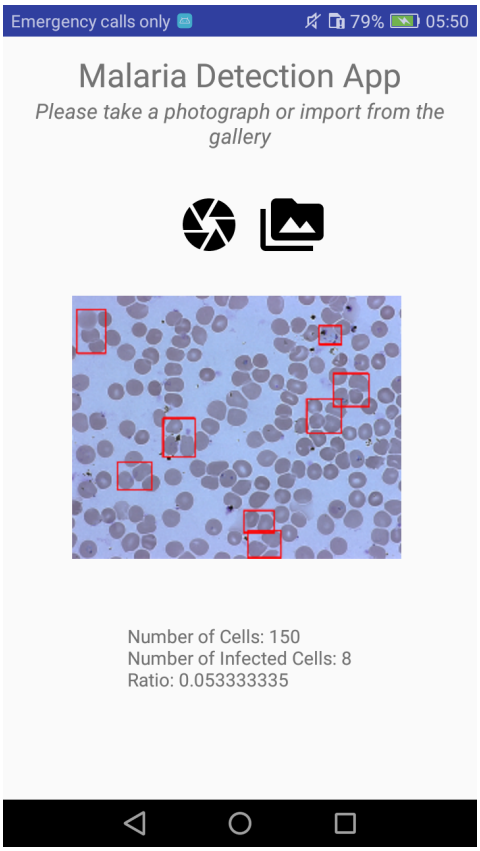


Figure 4. Android Application prototype

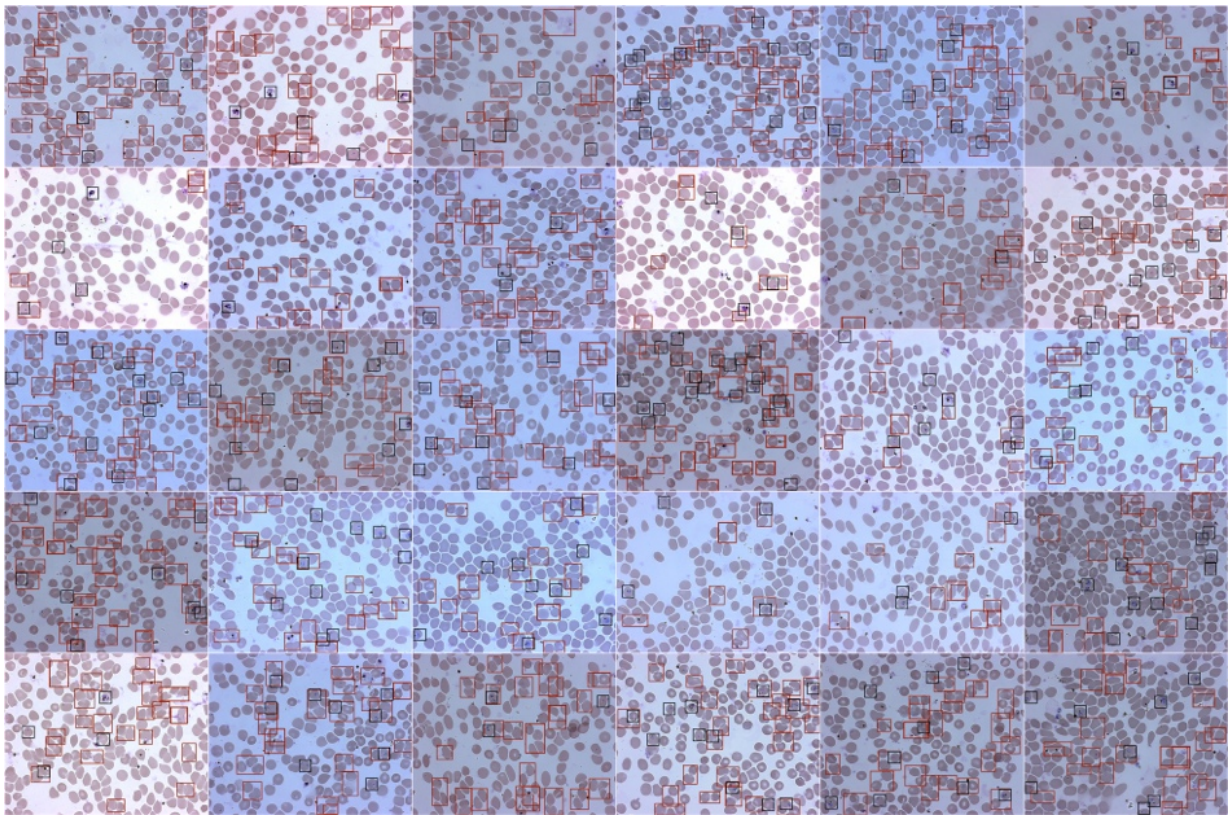


Figure 6. All 30 images after detection process

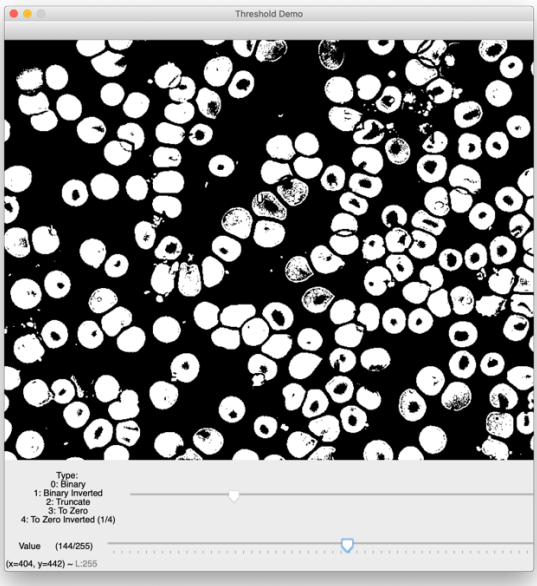


Figure 5. Threshold value dynamic calculator