Malaria Blood Detector Based Deep Learning

By

Rija Tonny Christian RAMAROLAHY (rija@aims.edu.gh)

June 2020

AN ESSAY PRESENTED TO AIMS-GHANA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE IN MATHEMATICAL SCIENCES

DECLARATION

This work was carried out at AIMS-Ghana in partial fulfilment of the requirements for a Master of Science Degree.

This work is being considered for publication in a journal.

I hereby declare that except where due acknowledgement is made, this work has never been presented wholly or in part for the award of a degree at AIMS-Ghana or any other University.

Student: Rija Tonny Christian RAMAROLAHY 

Supervisor: Dr. Alessandro CRIMI 

ACKNOWLEDGEMENTS

Firstly, I thank the Almighty God for his blessing and grace; and for giving us health and knowledge to be able to complete this project.

I thank AIMS for this opportunity. Especially, I thank the AIMS-Ghana president and also the academic director for this wonderful academic year. Many thanks to AIMS-Ghana staffs and personnel services for making the learning and the stay possible and enjoyable.

I would like to express my deepest gratitude to my supervisor Dr. Alessandro Crimi. I am very grateful to him for his support and guidance throughout this project. Despite this critical moment of COVID19 pandemic, you were there for me and helped me until the project was completed.

I am also grateful to my tutor Esther Opoku Gyasi for her support from the beginning of the program till the submission of the project. I really appreciated your willingness to help us especially during the project phase. Thank you for you time and your support. Many thanks also to all the tutors at AIMS-Ghana for their support and their help during the program.

I would also like to thank Dr. George Frimpong for his support during the write-up of this project.

Special thanks to my best friends and the entire AIMS-Ghana family 2019-2020; it was an amazing academic year. I really enjoyed all the times spent together.

Last but not least, I would like to thank especially my mothers: Lydia and Bodo, Valerie and her family, my siblings and their spouses: Paty, Andry and Kanto, Veve and Lala, Rina and Tantely, Andry and Pita, my cousins and their spouses: Naina and Diah, Nini and Vola, Patou and Fanja, Elyse and Michou, Sedra and Domoina and all my families. Thank you for your supports, prayers, encouragements and sacrifices for making this happen.

Abstract

Effective and quick detection of malaria is a key factor for reducing its mortality rate. In that case, automated detection (modern method) meets these expectations more than the common methods (light microscopy, rapid diagnostic test) which have some limitations. This modern method is used to detect malaria parasites on stained blood smear images using machine learning techniques and computer vision methods. In contrast to the common methods, this does not require any presence of experts during the detection and showed a better sensitivity on the results. Indeed, this current study is based on the microscopic images of the stained blood smear. And in the case of dealing with large images datasets, convolutional neural network (CNN) which is a class of deep learning model is known to give a better result for feature extraction as well as classification. In this work, we train and evaluate a convolutional neural network models on stained blood smear images datasets for malaria parasites detection. Specifically, we consider both thick and thin stained blood smear. Afterwards, we combine these models with an image processing method to perform an automated detection of malaria parasites.

Contents

1. Introduction

The main aim of this chapter is to introduce the disease 'malaria' and an important concept used in this work called convolution. The chapter also gives a brief introduction to the generative adversial network which is related to this work. Afterwards, we state the objectives and the structure of this work.

1.1 Malaria

Transmitted to humans through the bites of infected female anopheles mosquitoes, a parasite called plasmodium causes a blood disease known as Malaria. For several years now, malaria has become a global disease because almost 228 million cases of malaria were seen across the world in 2018 and 93% of these cases have been reported by the World Health Organization (WHO) in the African region. The countries most affected by this disease are in sub-Sahara Africa. Indeed, it was reported that 18 countries in sub-Sahara Africa and India represent 85% of the global malaria cases in 2018 [13, p. 4].

Malaria is a curable disease, but if it is not treated quickly, it will become a severe illness and might cause death. The WHO report stated that, in 2018, malaria caused 405,000 deaths globally. It is also seen that the most vulnerable group affected are the children under five years and 67% of the global malaria deaths are seen in this group. Again, the WHO African region recorded the highest malaria deaths of 94% in 2018. Moreover, it represents a burden even for the curable cases [13, p. 6].

Common symptoms of malaria comprise fever, fatigue, headache and chills, which make it difficult to recognise. For the severe case of malaria, the person will have additional symptoms like anemia, convulsions and coma which might lead to death. Since malaria is a blood disease, in order to diagnosis it, the person has to perform a blood test quickly to be able to treat and reduce the chance of moving into the severe case. The detection of the presence of the parasites of malaria in the blood is often done by using two methods: microscopy and Rapid Diagnosis Test (RDT) [11, p. 7]. In the case of using the microscopy method, the entire results and the future of the patient depend heavily on the experience and skills of the *microscopists* [12, p. 2-3]. And the RDT which gives a quicker result than the microscopy method also has some limitations and inconveniences which may include; a low sensitivity in the result and more expensive than microscopy in terms of cost (if we do not count the cost of the microscope).

An effective and quick diagnosis is one of the important basis to control malaria [13, p. 52] and several studies have been conducted to find a method which gives these expectations, especially in the last ten years [15]. This work will focus on one of these methods. Precisely it will talk about the automated malaria diagnosis.

1.2 Convolutional Neural Network

For a couple of years now, task automation has become the focus of several researchers in many areas and *Deep Learning* has become the area of interest for many researchers. In this work, we will see a type of Deep Learning which has been considered in achieving high performance in image classification for large dataset. This method is called Convolutional Neural Network (CNN) [2]. In this work, we will use it to automate a task to detect malaria parasites in a blood smear.

CNN is a deep learning class where the hidden layer is made of convolutional layers, pooling layers and fully-connected layers. We will see the details of each of them later but now let us see the most important concept in this method that is the convolution.

In mathematics, a convolution is an operation on two functions which expresses the sliding of one function over the other function. A continuous convolution of two functions f and g written as $f * g$ is the integral of the form

$$f * g(x) = \int_{-\infty}^{\infty} f(\alpha)g(x-\alpha)d\alpha, \tag{1.2.1}$$

and a discrete convolution of two functions f and g is of the form

$$f * g(n) = \sum_{\alpha=-\infty}^{\infty} f(\alpha)g(n-\alpha). \tag{1.2.2}$$

The convolution can be extended in 2D and the formula for continuous and discrete convolution are;

$$f * g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)g(x-\alpha,y-\beta)d\alpha d\beta \tag{1.2.3}$$

and

$$f * g(n,m) = \sum_{\alpha=-\infty}^{\infty}\sum_{\beta=-\infty}^{\infty} f(\alpha,\beta)g(n-\alpha,m-\beta). \tag{1.2.4}$$

This concept of convolution is applied in many areas like signal processing, image processing, digital data processing and electrical engineering. But in this work we will focus on the application of convolution in image processing.

In image processing, the convolution is usually used when we want to do some operation on an image like filtering, blurring, edge detection and so on. We call these operations kernels. A grayscale image can be represented in a 2D matrix where the entries have values between 0 and 255. Then the convolution of a kernel, which will also be a matrix, in a 2D image will give another matrix precisely if f is a $n \times n$ matrix and h is a $k \times k$ matrix (in most case, k is odd), then $w = f * h$ is a $(n - \frac{(k+1)}{2}) \times (n - \frac{(k+1)}{2})$ matrix [22, p. 154]. The operation between the two matrices follows the formula of the 2D discrete convolution.

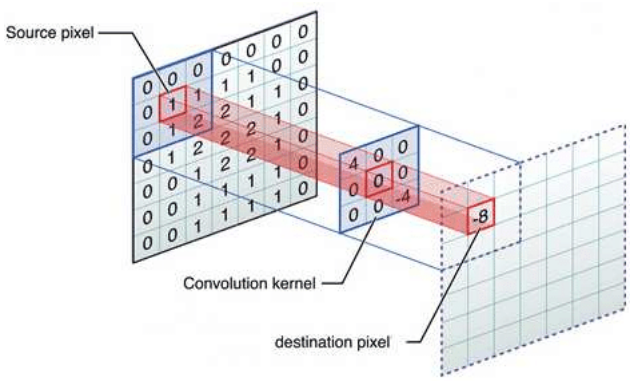


Figure 1.1: Action of the 2D discrete convolution of an input image with a filter [1].

The Figure 1.1 shows the action of the kernel on an image using convolution. Let us consider an example. Consider we have an image with a size 5×5 and a kernel (edge detection) with a size 3×3 such that;

$$f = \begin{pmatrix} 8 & 0 & 2 & 4 & 0 \\ 5 & 7 & 1 & 0 & 3 \\ 2 & 4 & 6 & 4 & 1 \\ 9 & 1 & 3 & 5 & 7 \\ 3 & 1 & 0 & 4 & 2 \end{pmatrix}, \qquad h = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}.$$

Therefore, the convolution of these two matrices will be;

$$w = f * h = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

where the components are given by

$$\begin{aligned}
w_{11} &= 8\times 1+0\times 0+2\times(-1)+5\times 0+7\times 0+1\times 0+2\times(-1)+4\times 0+6\times 1 = 10,\\
w_{12} &= 0\times 1+2\times 0+4\times(-1)+7\times 0+1\times 0+0\times 0+4\times(-1)+6\times 0+4\times 1 = -4,\\
w_{13} &= 2\times 1+4\times 0+0\times(-1)+1\times 0+0\times 0+3\times 0+6\times(-1)+4\times 0+1\times 1 = -3,\\
w_{21} &= 5\times 1+7\times 0+1\times(-1)+2\times 0+4\times 0+6\times 0+9\times(-1)+1\times 0+3\times 1 = -2,\\
w_{22} &= 7\times 1+1\times 0+0\times(-1)+4\times 0+6\times 0+4\times 0+1\times(-1)+3\times 0+5\times 1 = 11,\\
w_{23} &= 1\times 1+0\times 0+3\times(-1)+6\times 0+4\times 0+1\times 0+3\times(-1)+5\times 0+7\times 1 = 2,\\
w_{31} &= 2\times 1+4\times 0+6\times(-1)+9\times 0+1\times 0+3\times 0+3\times(-1)+1\times 0+0\times 1 = -7,\\
w_{32} &= 4\times 1+6\times 0+4\times(-1)+1\times 0+3\times 0+5\times 0+1\times(-1)+0\times 0+4\times 1 = 3,\\
w_{33} &= 6\times 1+4\times 0+1\times(-1)+3\times 0+5\times 0+7\times 0+0\times(-1)+4\times 0+2\times 1 = 7.
\end{aligned}$$

Thus the result is

$$w = \begin{pmatrix} 10 & -4 & -3 \\ -2 & 11 & 2 \\ -7 & 3 & 7 \end{pmatrix}.$$

We can see in Figure 1.2 an example of applying a convolution on an image.

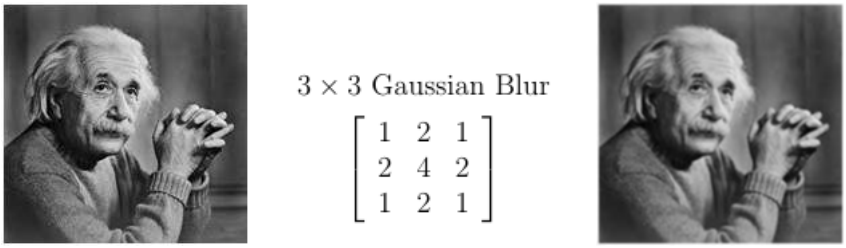


Figure 1.2: Convolution of an image using Gaussian blur as kernel.

We will see in the further chapters the advance concepts about this convolution and the other details.

1.3 Generative Adversial Networks (GANs)

When using convolutional neural network as classifier, the number of training datasets is an important key to get high accuracy. Precisely, if the number of image dataset increases then the model will show a higher accuracy. In medical imaging, getting images of the study of interest may be expensive and getting large datasets may take a long time because we might need to extract the data from a global image or to do some annotations, and these require an expert from the domain or on the data [10]. Therefore, to avoid this problem of getting extra images for the studies, there is a method for data augmentation using an existing data. This method is called Generative Adversial Networks (GANs). GANs is a machine learning technique which can generate synthetic images from existing images. This method is mostly used before training a CNN model to improve its performance withing the datasets obtained by the GANs method.

1.4 Main Objectives

This work will show a method for automated malaria detection using deep learning and image processing methods. Precisely, we will train a convolutional neural network model and use it to detect malaria parasites on blood smears.

1.5 Structure of the Thesis

This work contains five chapters, such that in chapter 2, a review of some literature related to this work is done. In chapter 3, we talk about the data and the methods used for this work. Chapter 4, contains the results and the final chapter is the conclusion of the work.

2. Literature Review

This chapter gives a brief summary of the literature related to this work where we will see the different methods of malaria detection used and the automated malaria detection, some view of image processing and a summary of relevant methods in malaria detection.

2.1 Malaria Diagnosis

Malaria in humans is caused by five different types of parasite: Plasmodium falciparum, Plasmodium vivax, Plasmodium malariae, Plasmodium ovale and Plasmodium knowlesi. The plasmodium falciparum is the one which may lead to death and almost 99% of the malaria cases in sub-Sahara Africa is due to this parasite. Each parasite has a distinct form depending on its development and this occurs at the following stages: ring stage, trophozoide stage, schizont stage and gametocyte stage. Figure 2.1 shows these different stages. These stages help parasitologists to know the types of parasites because they differ from each other [15].

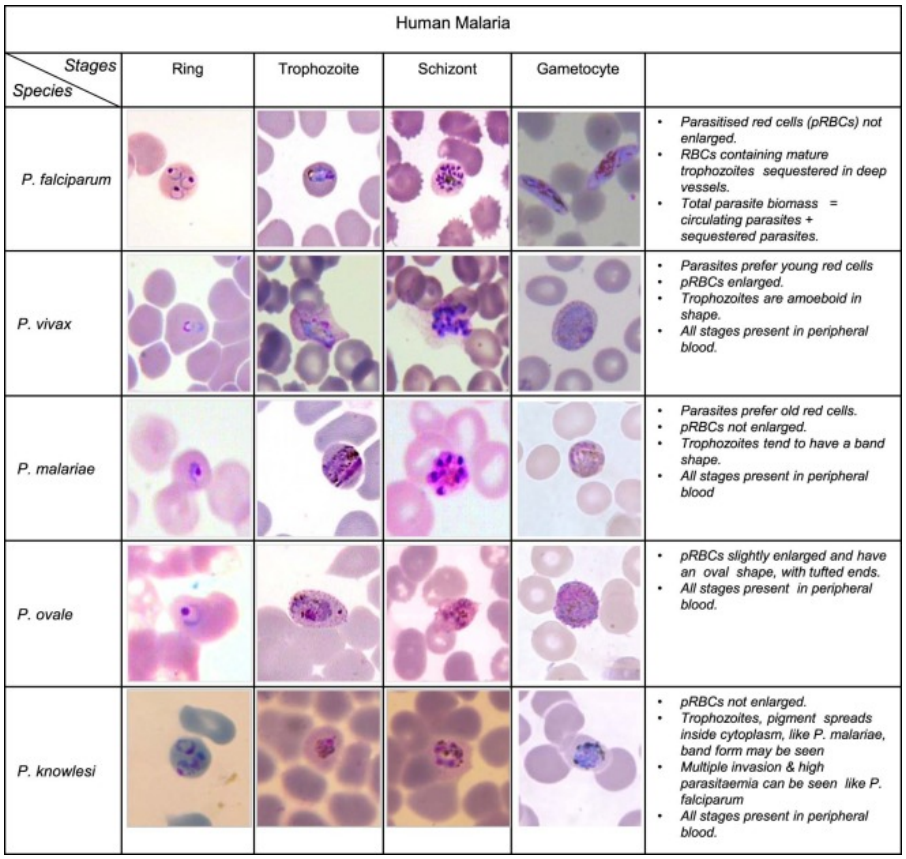


Figure 2.1: The different species of parasites and their stages [15].

Recognising these species of parasites helps in the treatment of the patients. There are several

methods to diagnose malaria and recognise these species of parasites. The most common methods are light microscopy and the Rapid Diagnostic Test (RDT). But there are other methods like Polymerase Chain Reaction (PCR), fluorescent microscopy, flow cytometry and many others.

2.1.1 Light microscopy. This method is the standard method used for malaria diagnosis. It is a very efficient method and gives a good result in terms of sensitivity and specificity because it allows to detect all parasites species. The method needs either thin or thick stained blood smear from the patients and an expert microscopist to detect the malaria parasites from this blood smear using a microscope. A single blood smear takes 15 to 30 minutes for a microscopist to examine it according to their expertise and skills. Then an effective result needs a best parasitologist and a good work environment so that they will be able to maintain their skills to avoid false diagnosis [12].

2.1.2 Rapid Diagnostic Test (RDT). This method takes 10 to 15 minutes to detect the presence of a malaria parasite in a small amount of blood using a special device built for it. It is easier to use but it gives a lower sensitivity and specificity than the microscopy method and it is also known that it costs more expensive for some regions [15].

2.1.3 Polymerase Chain Reaction (PCR). It is a molecular diagnostic model which gives a better result than the two previous methods. This method shows more sensitivity and specificity than the microscopy method. However the method is complex and takes one hour to detect the presence of parasites. Also it is expensive and a presence of expert technicians is needed [19].

2.2 Automated Diagnosis

For the last decade, many researchers have focused on finding an automation diagnosis for malaria. Thus, diverse studies have been written on this area and these methods showed a good accuracy as well as sensitivity and specificity in the detection of malaria parasites. The automated detection of malaria has been done using machine learning techniques, or more specifically deep learning techniques in the most recent models. The first step of the method is common for all models, which is taking an image of the blood smears from a microscope, detecting and segmenting the region of interest (ROI) inside this image using image processing techniques. And the next steps depend on the machine learning methods used.

In the case of using machine learning technique as classifier, an extra step for feature extraction and selection is needed. In this step, some techniques and methods are used in the image to differentiate and recognise the appearance of infected and uninfected cells. This feature selection requires an engineering expertise to analyse feature color, textural and morphological features to distinguish the features of the parasitized and healthy cells [15]. After these steps, a classifier algorithm such as Support Vector Machine (SVM), logistic regression or others can be used to classify and detect the infected and uninfected cells through the feature selection.

On the other hand, deep learning method does not need the steps of hand-engineered feature selection because it could also serve as feature extractor as well as classifier. In the case of malaria detection, Convolutional Neural Network (CNN) model which is a class of deep learning

achieved a high accuracy as well as sensitivity and specificity than the machine learning methods [17]. Evaluating the performance of a CNN model is one of the main focus of this work.

2.3 Image Processing

This step of image processing is important and crucial to automated malaria detection because the classification step depends on this. The main goal of this process is to detect the region of interest from a stained blood smear image to be able to detect malaria parasites. The step contains some internal processes depending on the type of the blood smear image, either thick or thin blood smear.

2.3.1 Thick smear image process. In thick blood smear, only the nuclei of the red blood cells and the white blood cells are seen. Therefore, for this type of blood smear we have to detect the parasites candidate and classify them to be able to know if they are a parasites or not.

- Image acquisition: First, we need an image of the stained thick blood smear that we want to analyse. This process needs a microscope and another device which can take a picture of the image seen from the microscope, for example a camera or smartphone. The quality of the image will depend on the type of microscope chosen and the camera (in case of using it).
- Pre-process: This process is needed to get a better quality of the image of the blood smear, for example reducing the noises. The most approach used in this process is to apply a blurring filter like median or Gaussian filter to reduce the noises in the image.
- Parasite candidate detection: This process is the most important for malaria detection on thick blood smear. There are many techniques that can be used to select the region of interest on a thick blood smear image. One example is to separate the image into two (black and white or foreground and background) according to the grayscale image intensity using threshold method such as Otsu thresholding or histogram threshold method. And consider the region which contains each object detected (foreground) as the region of interest.

  Yang et al. [21] used Otsu thresholding to detect white blood cells and clean them in the threshold image. And based on this threshold image, they used an algorithm called iterative global minimum screening (IMGS) to select parasite candidate.

  Quinn et al. [16] created patches from the image and created an algorithm to avoid the overlap between the patches.

White blood cells detection using Otsu thresholding by Yang et al. [21] is seen in Figure 2.2.

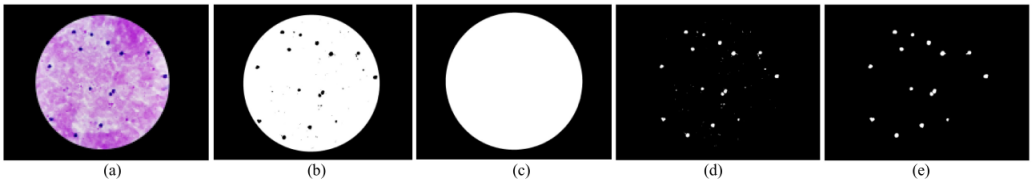


Figure 2.2: White blood detection [21].

In Figure 2.2; (a) represents the input image, (b) shows the result of Otsu thresholding on the image, (c) creating a mask from the threshold, (d) subtracting the threshold by the mask to obtain the white blood cells and (e) the result after removing small noise.

2.3.2 Thin smear image process. The detection of the region of interest in thin smear image also needs the same first two processes used for thick smear image, which are the image acquisition and the pre-process. The processes which differentiate them are the following.

- Cells detection: There exist several methods for cell detection, for example, threshold method such as Otsu thresholding or histogram thresholding, edge detector method such as Laplacian of Gaussian (LoG) or edge detection algorithm and so on. These methods are used to detect the objects of interest inside the image.

  Rajaraman et al. [17] used a multi-scale LoG filter as an edge detector to detect the center of the red blood cells. This method combines the Laplacian and Gaussian filter. The Laplacian filter is used to detect the edge of an image, but the result may appear with some noises and the Gaussian filter is used to smoothen this result.

- Cells segmentation: As for cells detection, there are many techniques for cell segmentation. For example, marker-controlled watershed, active contour model or morphological operation to mark the cells detected.

  Rajaraman et al. [17] used level set activation contour technique as segmentation technique. This method gave a best approximation on the contour because of its topological flexibility [4].

- Post-process: This last process is applied when we want to remove unwanted objects such as staining artefacts in an image. The morphological operation such as opening removes objects and dilates the image after removing them (erosion followed by dilation). This process is also used when we want to take out the white blood cells because we only need red blood cells. A technique based on ground-truth annotation can be used because white blood cells are easy to recognise.

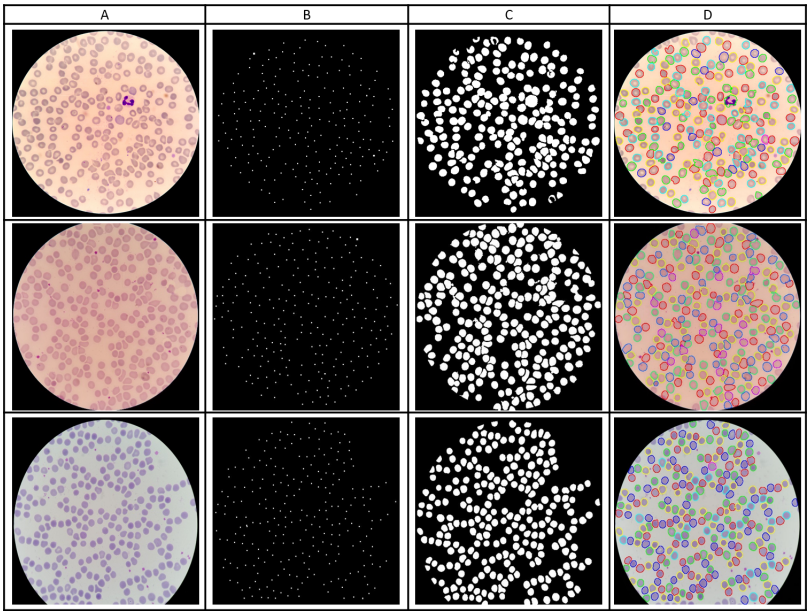


Figure 2.3: Red Blood Cells detection and Segmentation [17].

The Figure 2.3 summarizes the processes used by Rajaraman et al. [17] for cells detection and segmentation; column A represents the picture taken from a microscope, column B shows an edge detection by the LoG filter, column C used the segmentation marker and the last column is the combination of the segmentation and the original image which gives the red blood cells existing in the pictures.

Poostchi et al. [15] summarize some different techniques of image processing used by some related literature.

2.4 Point of Care Diagnostic (POC)

POC is a method to detect diseases using a microscope and a specific device mounted on it. Quinn et al. [16], deployed a point of care diagnostic with a microscope and a smartphone and linked them using adjustable specific hardware.

The smartphone was equipped with a software which can detect a small patch of the image if it contains a pathogen of the disease or not. Precisely, the software was a convolutional neural network model trained on a large image dataset and combined with a computer vision method which creates the patches on the image. They tested this method on a thick blood smear to detect malaria, on a sputum sample to detect tuberculosis and on stool samples to detect intestinal parasites. The Figure 2.4 shows the entire system.

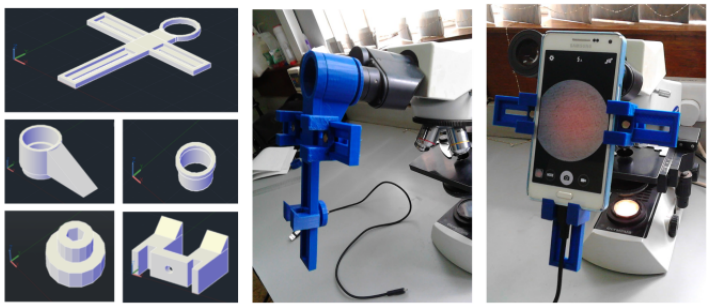


Figure 2.4: Smartphone mounted on a microscope using an adapter [16].

Yang et al. [21] also developed an application for android smartphone for malaria detection on a thick blood smear. The application was based on the iterative global minimum screening method for parasites candidate detection and a trained convolutional neural network for parasites detection.

Figure 2.5 shows the detection of the methods on a thick blood smear.

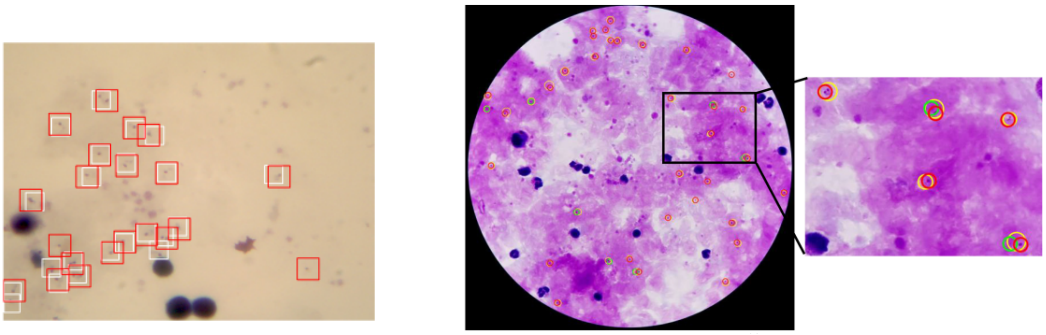


Figure 2.5: Smartphone malaria detection by Quinn et al. [16] and Yang et al. [21].

3. Data Description and Methodology

In this chapter, we are going to see more about convolutional neural network. Again, we will see some image processing algorithms, more about the data and the details of the CNN model used.

3.1 Convolutional Neural Network (CNN)

The CNN model has been known for classifying a large image dataset and many works demonstrated the performance of this model in image classification. For example, AlexNet, which is an architecture of a CNN model won ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012 on classifying 15 million of images belonging to 22,000 categories [9]. Other architectures such as VGGNet, ResNet and GoogleNet also showed best performances on the following ILSRVC competitions [17].

The CNN model comprises of three hidden layers which are convolutional layer, pooling layer and fully-connected layer. Here is an image of the structure of a basic CNN model.

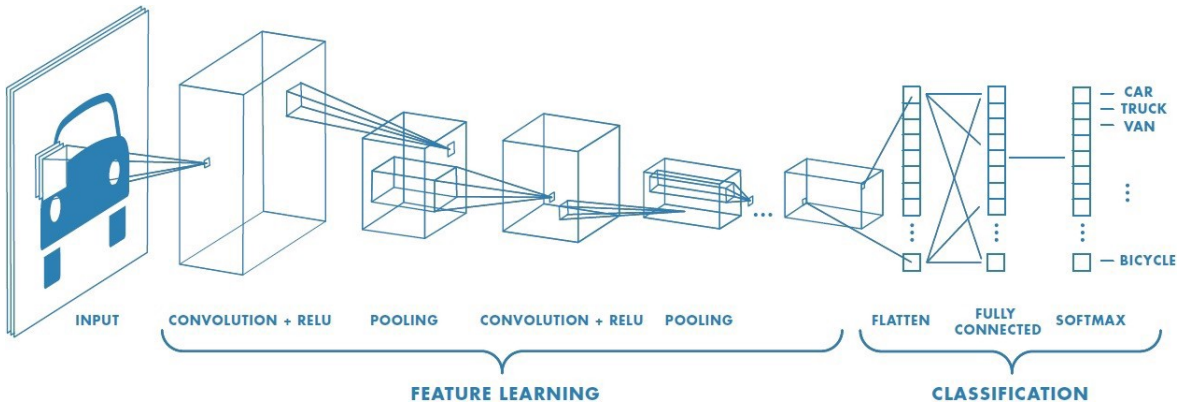


Figure 3.1: Basic Convolutional Neural Network model [18].

3.1.1 Convolutional Layers. As we saw in Figure 3.1, these layers are called feature learning because, in these layers, convolution is applied to an input image using kernel to get its high-level features.

In real-life applications, the input image can be represented by a $h \times w \times d$ matrix where h is the weight of the image, w is its width and d represents the channel of the image. Precisely either the image is in Red Green Blue (RGB) form $d = 3$ or grayscale form $d = 1$. In the case of RGB image, the kernel matrix will also be of the form $k \times k \times 3$ and the convolution is applied between each matrix of the image and the kernel. The concept of convolution follows the formula of 2D discrete convolution in equation (1.2.4) and the computation of the convolution matrix uses the same rule seen in Figure 1.1. After getting the result of the convolution for each matrix, we combine them and add a bias which is equal to 1. The Figure 3.2 shows us an example of the computation of convolution on an image in RGB form.

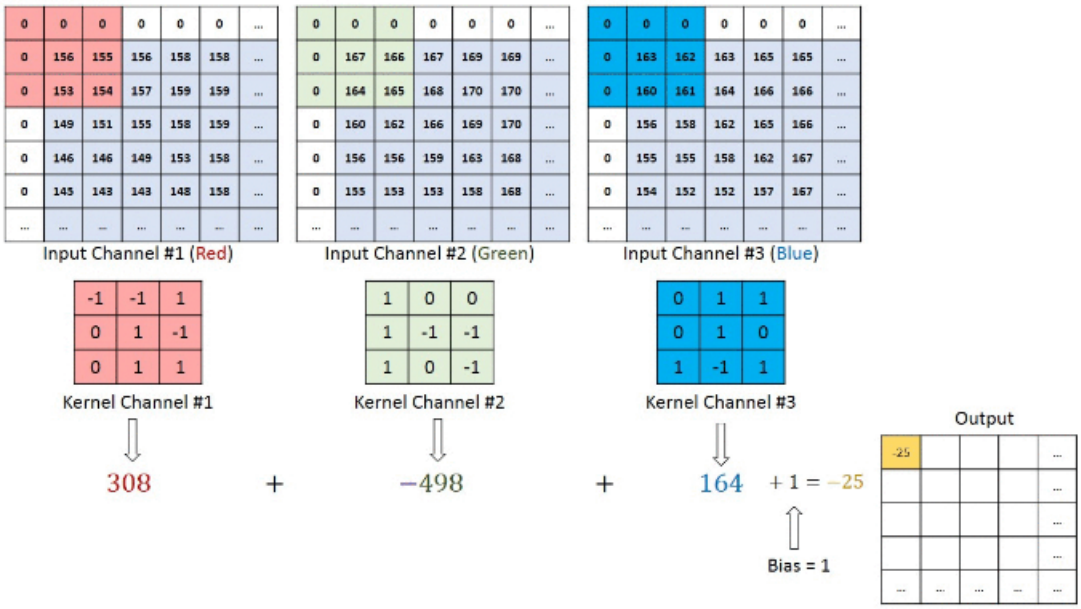


Figure 3.2: Example of the convolution of a RGB image with a kernel [18].

A single convolutional layer will contain many kernels to be able to get the features of the image. And an architecture might use many convolutional layers according to the model. In a model, a convolutional layer is followed by an activation function to rectify the output of the convolution and to have a non linear output. The most used activation functions are:

- Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, this function range the output values between 0 and 1.
- Hyperbolic tangent function: $f(x) = tanh(x)$, this function range the output value between -1 and 1.
- Rectified Linear units (ReLu): $f(x) = \max(0, x)$.

Most of the CNN model use ReLu as activation function because it is seen that a CNN with this activation converges six times faster in the training than with hyperbolic tangent or sigmoid activation function [9].

We can see an example of a convolutional layer used on the infected and uninfected red blood cells in the Figure 3.3.

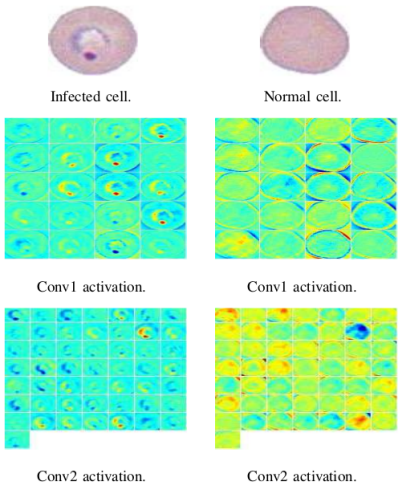


Figure 3.3: Result of two convolutional layers on infected and uninfected red blood cells [7].

3.1.2 Pooling layers. These layers are used to take the dominants features given by the convolutional layers and they are also used to reduce the dimensionality of the parameters of the image. The pooling uses a kernel as for the convolution on an image and the action of the kernel differ according to the type of pooling. There are two types of pooling which are mostly used

- Max pooling which takes the maximum value of the elements from a block on the image according to the size of the kernel.
- Average pooling which takes the average of the elements from a block on the image according to the size of the kernel.

Let us consider an example to illustrate these two types of pooling. Consider a 4×4 image such that the matrix form of the image is

$$\begin{pmatrix} 0 & 4 & 6 & 4 \\ 3 & 9 & 8 & 2 \\ 2 & 0 & 2 & 1 \\ 4 & 6 & 1 & 4 \end{pmatrix}.$$

Therefore applying max pooling and average pooling of size 2×2 gives;

$$2 \times 2 \text{ max pooling} = \begin{pmatrix} 9 & 8 \\ 6 & 4 \end{pmatrix} \text{ and } \qquad 2 \times 2 \text{ average pooling} = \begin{pmatrix} 4 & 5 \\ 3 & 2 \end{pmatrix}.$$

3.1.3 Fully-connected layers. These last layers are the layers for classification. The first step of this layer is to flatten the final output of the pooling layer to have a single column vector of all the possible features. Then use of the concept of neural network, precisely the back propagation process, is to classify the images in the dataset. The final output of this layer is followed by an activation function such as softmax or sigmoid to be able to determine the class of each image.

A neural network contains an input layer, hidden layers and output layer. Consider a neural network with one hidden layer as we see in Figure 3.4.

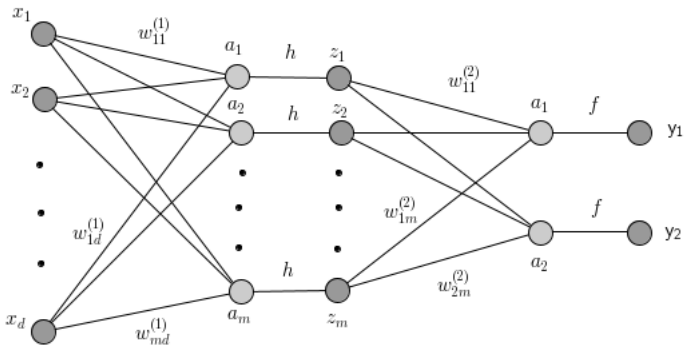


Figure 3.4: Neural network with one hidden layer.

As we can see, the inputs (x_i or z_j) and the outputs (a_j or a_k) of each layer are represented by nodes and each node from consecutive layers is connected to each other by the weight parameters (w_{ji} and w_{kj}) represented as the link between them. Therefore, the outputs of the hidden layer from the input layer have the form

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \tag{3.1.1}$$

where $w_{ji}^{(1)}$ denote the weights in the first layer, $w_{j0}^{(1)}$ the biases, d is the number of inputs and $j = 1, \ldots, m$ the number of node in the hidden layer. We called these quantities activations and each activation is transformed to a non linear output by an activation function;

$$z_j = h(a_j), \tag{3.1.2}$$

where h is the activation function. These non linear outputs become the input of the next layer.

Using the formula seen in equation (3.1.1), we obtain the final outputs by applying an activation function to the outputs activations;

$$a_k = \sum_{j=1}^{m} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{3.1.3}$$

$$y_k = f(a_k), \tag{3.1.4}$$

where $k = 1, \ldots K$ is the number of outputs, m is the number of inputs from the previous layer, $w_{kj}^{(2)}$ denotes the weights in the second layer, $w_{k0}^{(2)}$ the biases and f the activation function.

Using equations (3.1.1), (3.1.2), (3.1.3) and (3.1.4), we obtain the formula for the final outputs of a neural network with one hidden layer;

$$y_k(x, w) = f\left(\sum_{j=1}^{m} w_{kj}^{(2)} h\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right), \tag{3.1.5}$$

where x is the set of input and w the set of weights and biases. And often, we add the biases inside the weights set by adding a variable $x_0 = 1$ into the inputs set. Therefore, equation (3.1.5) becomes

$$y_k(x, w) = f\left(\sum_{j=0}^{m} w_{kj}^{(2)} h\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right)\right). \tag{3.1.6}$$

Using this equation, we can generalise the formula for the outputs on a neural network with n hidden layer as

$$y_k(x, w) = f_n\left(\sum_{r=0}^{S} w_{kr}^{(n)} f_{n-1}\left(\dots f_2\left(\sum_{j=0}^{M} w_{lj}^{(2)} f_1\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)\right)\right), \tag{3.1.7}$$

where x is the set of inputs, w the set of weights, $f_n, \dots, f_2, f_1$ are the activation functions, D the number of inputs, S the number of final outputs and M the number of nodes in the first hidden layer.

Therefore, during the training of this network, the objective is to find the weight parameters which minimize the error function chosen. The back propagation algorithm is seen to be more efficient in this procedure. This algorithm contains two stages such that the first stage evaluates the gradient of the error with respect to the weights and the second stage use this gradient to adjust the weights using a conjugate gradient optimisation method.

Consider the sum of squares error function

$$E(w) = \frac{1}{2}\sum_{k=1}^{K}\Big(y_k(x, w) - t_k\Big)^2, \tag{3.1.8}$$

where x is the input set, w the weights set, y the outputs from the network and t the true outputs or the target set. Then we evaluate the gradient of this error with respect to a particular weight w_{ji} i.e

$$\frac{\partial E(w)}{\partial w_{ji}}. \tag{3.1.9}$$

Let us apply chain rule on this derivative since we know that the weight is related to the activations output a_j;

$$\frac{\partial E(w)}{\partial w_{ji}} = \frac{\partial E(w)}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}, \tag{3.1.10}$$

where a_j is the activations computed for a layer as we seen in equation (3.1.1). Then, replacing a_j by its form (putting the bias inside the weight set), we obtain;

$$\begin{aligned}\frac{\partial a_j}{\partial w_{ji}} &= \frac{\partial \sum_i w_{ji} z_i}{\partial w_{ji}},\\ &= \frac{\partial(w_{j0}z_0 + w_{j1}z_1 + \dots + w_{ji}z_i + \dots)}{\partial w_{ji}},\\ &= z_i.\end{aligned}$$

Assume that $\delta_j = \dfrac{\partial E(w)}{\partial a_j}$, then equation (3.1.10) becomes

$$\frac{\partial E(w)}{\partial w_{ji}} = \delta_j z_i. \tag{3.1.11}$$

To compute the derivative of the error with respect to a particular weight, simply evaluate the value of δ_j in this layer and multiply it by the input of this layer.

Let us find out a general form of δ_j. Using equation (3.1.8) we can get δ_k for the outputs y_k

$$\delta_k = \frac{\partial E}{\partial y_k} = \frac{\partial\left(\frac{1}{2}\sum_{k=1}^{K}\left(y_k(x,w) - t_k\right)^2\right)}{\partial y_k} = y_k - t_k. \tag{3.1.12}$$

This equation means that the value of δ_k is known since y_k is computed by (3.1.7) and t_k is also known. And using this information, the name back propagation is showing up because we will use it to compute δ_j. Let us apply chain rule on the form of δ_j

$$\delta_j = \frac{\partial E}{\partial a_j} = \sum_l \frac{\partial E}{\partial a_l}\frac{\partial a_l}{\partial a_j}, \tag{3.1.13}$$

where we sum over the outputs activations for which the input j has a connection. Thus, using the definition of δ and a_l we get

$$\begin{aligned}
\delta_j &= \sum_l \delta_l \frac{\partial(w_{lj} z_j)}{\partial a_j}, \\
&= \sum_l \delta_l \frac{\partial(w_{lj} h(a_j))}{\partial a_j}, \\
&= \sum_l \delta_l h'(a_j) w_{lj}, \\
&= h'(a_j) \sum_l w_{lj}\delta_l,
\end{aligned}$$

where h is the activation function used in layer of j. Hence, the value of δ_j will depend on the value of δ of the next consecutive layer which is δ_l for which j has a connection

$$\delta_j = h'(a_j) \sum_l w_{lj}\delta_l. \tag{3.1.14}$$

Therefore, this means that we can compute all the δ's for each layer using this backward method because we always know the δ of the outputs of each layer starting from the final outputs y_k. Using equation (3.1.14), the derivative in equation (3.1.9) can be evaluated.

Consequently, the weights now can be adjusted using a conjugate gradient optimisation method and this is the second stage of the algorithm. One simple form of conjugate gradient optimisation

$$w_{\tau+1} = w_\tau - \eta \nabla E(w_\tau), \tag{3.1.15}$$

where $w_{\tau+1}$ is the updated weights, w_τ is the old weights, η the learning rate and $\nabla E(w_\tau)$ is the gradient of the error evaluated by (3.1.9). This method is called gradient descent.

In case of training large scale dataset, using stochastic gradient descent (SGD) optimisation or its variants is more appropriate instead of the simple gradient descent method. Since these methods update the weights and optimize the error function for a randomly selected batch; and this make the training faster. For example, for the SGD optimiser, the equation (3.1.15) becomes;

$$w_{\tau+1} = w_\tau - \eta \nabla E_n(w_\tau),$$

for $n = 1, 2, \ldots, N$ which is the number of batches and the global error is given by

$$\nabla E(w) = \frac{1}{n} \sum_{n=1}^{N} \nabla E_n(w).$$

In this work, an extension of the SGD called Adaptive Moment Estimation or simply Adam were used as the optimisation method for the training. The formula for the weights in this method is given by;

$$w_{\tau+1} = w_\tau - \eta \frac{\hat{m}_\tau}{\sqrt{\hat{v}_\tau} + \epsilon},$$

where $w_{\tau+1}$ is the updated weights, w_τ is the old weights, η is the learning rate, ϵ is a positive constant to avoid division by zero in the formula, $\hat{m}_\tau$ and $\hat{v}_\tau$ are called the estimators of the mean and the uncentered variance relatively and given by;

$$\hat{m}_\tau = \frac{m_\tau}{1 - \beta_1^\tau},$$

and

$$\hat{v}_\tau = \frac{v_\tau}{1 - \beta_2^\tau}.$$

And m and v are called the moving averages which are evaluated by;

$$m_\tau = \beta_1 m_{\tau-1} + (1 - \beta_1) \nabla E_\tau,$$

and

$$v_\tau = \beta_2 v_{\tau-1} + (1 - \beta_2)(\nabla E_\tau)^2,$$

where β_1 and β_2 are the hyper-parameters and ∇E_τ is the gradient of the error at a random batch in τ.

Adam optimiser was tested on different machine learning models and datasets; and it has shown a good result with respect to different optimisation methods including SGD [8].

3.1.4 Dropout. To reduce overfitting on the outputs of the fully-connected, a dropout is often used. In case of having a large number of neuron in the fully-connected layers, dropping out some neurons which have less probability will give the best result and make the training faster since the neurons dropped-out will not participate in the back propagation algorithm [9].

3.2 Algorithms for Image Processing

3.2.1 Otsu thresholding algorithm [14]. The aim of this algorithm is to find the threshold that minimises the intra-class variance also called within class variance or maximises the inter-class variance also called between class variance which separates an image into two classes in the sense of foreground and background (black and white). Consider an image expressed to a grayscale form where the pixels level are represented in the range of $1, \ldots, L$. Denote by n_j the number of pixel level j, then the total number of pixels is $T = n_1 + \cdots + n_L$. Thus each pixel level has a probability

$$p(j) = n_j/T, \tag{3.2.1}$$

where $p_j \geq 0$ and $\sum_{j=1}^{L} p(j) = 1$. Assuming that we want to separate the image in two classes C_0 and C_1 using a threshold t, then the probability of each class is represented by

$$w_0(t) = \sum_{j=1}^{t} p(j), \tag{3.2.2}$$

for the first class, and

$$w_1(t) = \sum_{j=t+1}^{L} p(j), \tag{3.2.3}$$

for the second class. The mean levels for each of them are

$$\mu_0(t) = \sum_{j=1}^{t} jp(j|C_0) = \frac{\sum_{j=1}^{t} jp(j)}{w_0}, \tag{3.2.4}$$

and

$$\mu_1(t) = \sum_{j=t+1}^{L} jp(j|C_1) = \frac{\sum_{j=t+1}^{L} jp(j)}{w_1}, \tag{3.2.5}$$

since $p(j|C_0)$ and $p(j|C_1)$ are the probabilities of j being in class C_0 and C_1 respectively.

Therefore, the within class variance for two classes separated by a threshold t is given by

$$\sigma_W^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t), \tag{3.2.6}$$

where

$$\sigma_0^2 = \sum_{j=1}^{t} (j - \mu_0)^2 p(j|C_0) = \frac{\sum_{j=1}^{t} (j - \mu_0)^2 p(j)}{w_0},$$

and

$$\sigma_1^2 = \sum_{j=t+1}^{L} (j - \mu_1)^2 p(j|C_1) = \frac{\sum_{j=t+1}^{L} (j - \mu_1)^2 p(j)}{w_1},$$

are the variances for each class.

The between class variance which measure the variance of the distance between the two classes is given by

$$\sigma_B^2(t) = w_0(t)(\mu_0 - \mu_T)^2 + w_1(t)(\mu_1 - \mu_T)^2, \tag{3.2.7}$$

where

$$\mu_T = w_0\mu_0 + w_1\mu_1 = \sum_{j=1}^{L} jp(j)$$

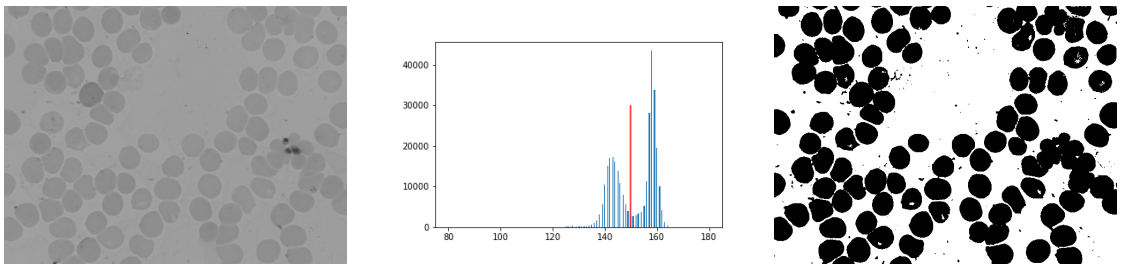
is the total mean of the image.

For reasons of simplicity and a quicker computation, maximisation of the between class variation is often used to find the threshold.

Here is the python code implementation of this Otsu thresholding using opencv package;

```
import cv2
cv2.threshold(image,0,255,cv2.THRESH_OTSU)
```

where the image is in grayscale form. This code returns two values which are the threshold that separate the two classes and the image in binary form precisely in black and white or background and foreground. The Figure 3.5 shows an example of the result of Otsu's method on an image.



(a) Thin blood smear image (b) Histogram of the image (c) Result of Otsu threshold

Figure 3.5: Example of Otshu threshold

Figure 3.5a is the input image in grayscale form, figure 3.5b shows the histogram of the image with the threshold obtain from Otsu method colored in red and figure 3.5c shows the binary form of the image by Otsu method.

3.2.2 Cells segmentation using marker based watershed transformation. In this section, we are going to see a method used for cells segmentation and detection. This method combines some morphological operations with watershed transformation to detect and segment the cells inside the blood smear image.

Morphological operations are operations applied to an input image, precisely a binary image, using a kernel to obtain an output image with the same shape. Since the image is in matrix form, therefore the kernel will be in matrix form and the kernel slides on the matrix of the image as the convolution but the type of morphological operation will determine the action of the kernel on the matrix.

Furthermore, there are some steps that we have to do to achieve our final goal.

- **Otsu thresholding**: The first step is to separate the image into two classes, i.e. foreground in white and background in black. We have previously seen the implementation of this algorithm, and we need to make the cells the foreground. Here is the python implementation;

```
import cv2
cv2.threshold(image,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

  We add the term cv2.THRESH_BINARY_INV to make the cells the foreground.

- **Erosion [20]**: This is the first morphological operation applied to the image. This algorithm is often used for noise removal or to separate connected object in an image. Let F be the matrix of the image and H the kernel. The erosion of F by H is denoted by $F \ominus H$ and H act on F according to its center. The pixel of the output with respect to the center of H maintains the value of the pixel on F if H fits F and 0 otherwise. Precisely, if (x, y) is the position of the center of H on F then

$$(F \ominus H)(x, y) = \begin{cases} F(x, y) & \text{if } H \text{ fits } F \\ 0 & \text{otherwise.} \end{cases} \tag{3.2.8}$$

  Consider an example with a binary 5×5 matrix F as the image and a 3×3 matrix H for the kernel such that;

$$F = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

  Then the erosion of F by H is

$$F \ominus H = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

  Some pixels of the object in F are removed therefore it becomes thin. Here is the python implementation of this erosion using opencv package;

```
import cv2
kernel =  np.ones((3,3), np.uint8)
cv2.morphologyEx(image,cv2.MORPH_ERODE,kernel,iterations=1) or
cv2.erode(image,kernel,iterations=1)
```

To use a morphological operation, we use the function cv2.morphologyEx() and for the erosion operation, we add cv2.MORPH_ERODE or simply use the function cv2.erode(). Here the image is a binary image, we use the same kernel as in the example and *iterations* is the number of times that the operation will be applied to the image. This code return the eroded image.

- **Dilation [20]**: The dilation is used to increase the area of the foreground of an image and decrease the area of background. The dilation of a matrix F with a kernel H is denoted by $F \oplus H$ and the action is the same as for the erosion but the output takes the value 1 if H hits F and 0 otherwise. Precisely, if (x, y) is the position of the center of H on F then

$$(F \oplus H)(x, y) = \begin{cases} 1 & \text{if } H \text{ hits } F \\ 0 & \text{otherwise.} \end{cases} \tag{3.2.9}$$

Let us compute the dilation of a 5×5 binary matrix F by a kernel 3×3 matrix H such that

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad H = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

The dilation of F by H is

$$F \oplus H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Pixels are added to the object in F and then its area in the image increases. Here is the python code implementation of the dilation using opencv package;

```
import cv2
kernel =  np.ones((3,3), np.uint8)
cv2.morphologyEx(image,cv2.MORPH_DILATE,kernel,iterations=1) or
cv2.dilate(image,kernel,iterations=1)
```

For dilation operation, we use cv2.MORPH_DILATE and the parameters are the same as for the erosion. This code returns the dilated image.

We call opening the operation which combine erosion and dilation or precisely an erosion followed by dilation. The opening of an image F by a kernel H is denoted by $F \circ H$ and it is given by

$$F \circ H = (F \ominus H) \oplus H. \tag{3.2.10}$$

The code implementation for opening in python using opencv is

```
import cv2
kernel =  np.ones((3,3), np.uint8)
cv2.morphologyEx(image,cv2.MORPH_OPEN,kernel,iterations=1)
```

For opening operation, we add the parameter cv2.MORPH_OPEN.

- **Distance transform [5, p. 13-15]**: This is an operator that finds the closest background for each object of an image. An image is made of a foreground (the object A) and a background (B), thus the distance transformation D of the input image makes an image which show the distance of the pixel of A with respect to the pixel of the closest background B, i.e

$$D(p) = \min\{dist_M(p,q), q \in B\},$$

where p is the pixel of the object A and $dist_M(p,q)$ is the distance metric between pixel p and q. Here are some distance metrics mostly used. Consider the coordinates of the pixel p as (p_x, p_y) and for q as (q_x, q_y), thus we have;

$$dist_{L_1}(p,q) = |p_x - q_x| + |p_y - q_y|, \tag{3.2.11}$$
$$dist_C(p,q) = \max\{|p_x - q_x|, |p_y - q_y|\}, \tag{3.2.12}$$
$$dist_{L_2}(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}, \tag{3.2.13}$$
$$dist_{L_2^2}(p,q) = (p_x - q_x)^2 + (p_y - q_y)^2, \tag{3.2.14}$$

where equation (3.2.11) is known as city block, (3.2.12) as chess board, (3.2.13) as euclidean distance and (3.2.14) the square of the euclidean distance. Consider an example with a simple 3×3 binary matrix A such that;

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

Using the square of the euclidean distance in equation (3.2.14), the distance transformation matrix of A is;

$$D = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

Since the coordinates of the background are $(2,2)$ and $(3,2)$, then the first component of D i.e. at the position $(1,1)$ is obtained by;

$$\min\{(1-2)^2 + (1-2)^2, (1-3)^2 + (1-2)^2\} = \min\{2,5\} = 2,$$

and the other components are computed using the same approach. The python implementation of the distance transformation using opencv package is;

```
import cv2
cv2.distanceTransformation(image,cv2.DIST_L2,maskSize=5)
```

The image (first parameter) should be in binary form, the second parameter (cv2.DIST_L2) is the type of the distance used. Here, we use the euclidean distance and the last parameter is the size of the distance transformation mask on the image (it is either 3 or 5).

- **Connected components [3, p. 65-71]**: This step is used to label all the objects in the image. The algorithm labels each connected components in the image with the same integer such that the components of the background are labelled with 0, and the other connected components of the objects are labelled from 1 to the maximum numbers acquired by the algorithm. The labelling is done pixel by pixel to get the connected components and follow the following rules:

  1. Start from the first pixel on the image, if the pixel is 0 then label it 0, else label it 1. And go to the next pixel.
  2. If the current pixel is not 0, then look at its neighbours, i.e. left, right, up, down, and the diagonals (if using $8-$connected).
     - If all neighbours have a common label, then label the current pixel with this common label.
     - If all neighbours have a pixel 0, then label the pixel with the last label plus 1.
     - If the neighbours have different labels, then label the pixel with the smallest label of its neighbours.
  3. Repeat the above steps if all pixels are not labelled.

  The code implementation of this algorithm is;

  ```
  import cv2
  cv2.connectedComponents(image)
  ```

  This code returns the number of labels used and an array with the same shape as the image but its components are the labels (integers).

- **Watershed**: This algorithm is used to detect the boundary of each regions inside an image based on the labels of the desired regions. The python implementation of this algorithm is;

  ```
  import cv2
  cv2.watershed(image,marker)
  ```

  Where the parameter marker corresponds to the labels of the desired regions in the image. Here we use the result from the connected components algorithm as the marker. This code returns a matrix where all the boundaries of each object according to the marker have a value equal to -1 and the other components have different values.

3.3 Procedure for Automated Detection

Here are the steps for automated detection used in this work.

- For thick smear image: we first separate the image into two, i.e. background and foreground in the binary form using a simple threshold technique. The value of the threshold is chosen by observation on some samples of images. Afterwards, we take the objects on the foreground as parasites candidates. We take off the white blood cells based on the size of the objects and classify the rest.
- For thin smear image: we first convert the image into grayscale level and increase its contrast. Afterwards, we use Otsu threshold to separate the image into background and foreground. Based on this result, we apply the algorithm for segmentation using watershed transformation. Firstly, we remove the noise using the morphological operation opening and increase the area of foreground using dilation. Next we use distance transform algorithm to find the distance between the closest background and each object to be able to separate them. Afterwards, we threshold the result of this algorithm to separate each cells and then apply connected components algorithm to label each object. Finally, we apply the watershed algorithm to mark each border of each object using the label from the connected components. We take off the bad segmentation, i.e. the regions containing more than one cells, based on the area of each object segmented and we classify the rest.

3.4 Data

3.4.1 Data collections. In this work, we used two different datasets.

- The first dataset contains $27,558$ individual cells images. The images comprise of infected and uninfected cells, and their numbers are equal. The cells were taken from a stained thin blood smear of 150 infected and 50 healthy patients, and their pictures were taken by a smartphone mounted on a microscope at Chittagong Medical College Hospital, Bangladesh [17].
- The second dataset consists of 1182 pictures of stained thick blood smear taken from a smartphone attached to a microscope at Makerere University, Uganda. The images come out with annotations of bounding boxes containing plasmodium made by experts where they found 7245 objects [16].

3.4.2 Data preparation. Before training our data, we did some preparations.

- For the first dataset, we combined the infected and uninfected images and split it into train, test and validation data. The images were resized to 100×100 pixel resolution and we normalised the data before using them.

- For the second dataset, we constructed data by taking positive (with plasmodium) and negative (without plasmodium) patches from the images. We first took patches of the images based on the annotations in order to take the objects (plasmodium) and put them as the positive patches. After that, we took random patches based on a fixed size from each image, and we made sure that no object will be present on these patches, we put them as the negative patches. Afterwards, we combined the positive and negative patches and split it into train, test and validation data. The patches were resized to 50×50 pixel resolution and normalised before using them.

3.5 Convolution Neural Network Model

We train the datasets with a simple CNN which have three convolutional layers and two fully connected layers.

Table 3.1: Simple CNN model

	Kernels/Neurons	Shape (size)
Input		100×100 / 50×50
Conv1 + ReLu	32	3×3
Maxpooling1		2×2
Conv2 + ReLu	64	3×3
Maxpooling2		2×2
Conv3 + ReLu	128	3×3
Maxpooling3		2×2
Flatten		
FC1 + ReLu	512	
Droupout (0.3)		
FC2 + ReLu	512	
Droupout (0.3) + Sigmoid		

From Table 3.1, the input image for the first dataset has a pixel resolution $100 \times 100 \times 3$ and $50 \times 50 \times 3$ for the second dataset. The first convolutional layer is made of 32 kernels, the second is made of 64 kernels and the third has 128 kernels; all with the same size 3×3. ReLu is applied to the output of each convolutional layer as the activation function and each convolutional layer is followed by a max pooling of size 2×2. The output of the last max pooling is flattened into a vector and becomes the input of the first fully-connected layer. The fully-connected layers have 512 neurons each and ReLu is also applied to their outputs as the activation function. Each fully-connected layer is followed by a dropout with probability of 0.3 and the output of the last dropout feeds into a sigmoid classifier.

3.6 Evaluation Metrics

We train our model on the train and validation data for each dataset. Then we evaluate the Area Under the Curve - Receiver Operating Characteristics (AUC-ROC) curve and the performance metrics with the test data. We also evaluate the model trained by the individual cells on the test data of the thick images datasets using AUC and the performance metrics. We evaluate the performance using accuracy, F_1 score, precision score, recall, sensitivity and specificity.

The outcomes of the classification are distributed into four values. These are: true positive (TP); which is the number of correct positive prediction, true negative (TN); as the number of correct negative prediction, false positive (FP); as the number of incorrect positive prediction and false negative (FN); as the number of incorrect negative prediction. These values construct the confusion matrix illustrated by the following table:

Table 3.2: Confusion matrix

	Predicted as positive	Predicted as negative
Labeled positive	TP	FN
Labeled negative	FP	TN

Using Table 3.2, let us see the formula of these evaluations;

- Accuracy is given by
$$Acc = \frac{TP + TN}{TP + TN + FN + FP}.$$
- The formula for precision or positive predicted value is
$$Prec = \frac{TP}{TP + FP}.$$
- Recall is given by
$$Rec = \frac{TP}{TP + FN}.$$
- F_1 score is
$$F_1 = \frac{2 \times Prec \times Rec}{Prec + Rec}.$$
- Sensitivity or the true positive rate is
$$SN = \frac{TP}{TP + FN}.$$
- Specificity or true negative rate is given by
$$SP = \frac{TN}{TN + FP}.$$

All these performances have values between 0 and 1. If they are close or equal to 1, then the model does a best classification otherwise, it is a bad classification.

The accuracy tells us how best the model correctly classifies the datasets. The performance of the model is given by the value of precision, recall and F_1 score. Thus the higher these values, the better the performance.

We train and test our first model on Google Colab for the need of GPU and the second model on a simple laptop without GPU.

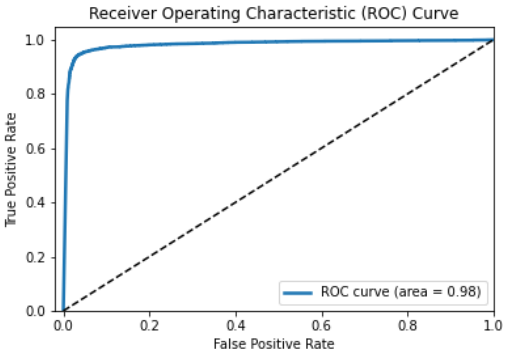
4. Results

In this chapter, we will see the results from the training and evaluations of the models by their AUC-ROC curve and performance metrics. We will also see results of the cells segmentation and parasites detection for thick and thin smear image.

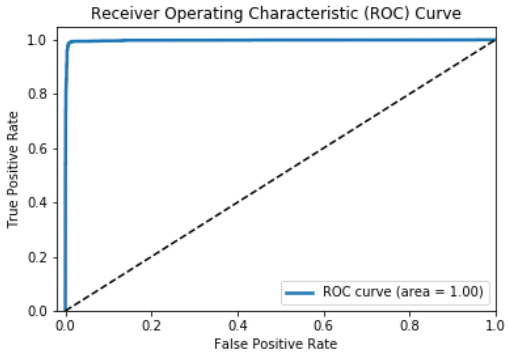
4.1 AUC-ROC Curve

The curve of ROC shows the ability of the model to classify the two classes using the true positive and the false positive rate. The AUC shows the measure of separability between the classes. AUC belong to a range between 0 and 1; best classification means the AUC is equal or close to 1 and bad classification means AUC equal or close to 0.

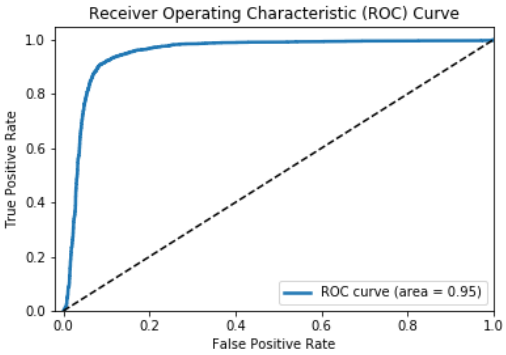
After training and testing the models, we obtain the following plots for the receiver operating characteristic (ROCs) curve with their corresponding AUCs



(a) ROC from the first model.



(b) ROC from the second model.



(c) ROC from the mixed test.

Figure 4.1: ROC curves

The ROC in Figure 4.1a was gotten from our model trained and tested on the individual cells

dataset. We can see a good result since the value of AUC is equal to 0.98. The second ROC in Figure 4.1b was gotten from our model trained and tested on the patches from the thick smear images dataset with annotations. The result is very good since the AUC is equal to 1. The third ROC in Figure 4.1c was gotten from our model trained on the individual cells tested on the patches from the thick smear images. The result is also good because AUC is 0.95 and this mean that our first model achieved a good classification on the thick images even through there was a difference on the size of the patches and the input image of the model. Precisely the patches have a size 40×40 pixel resolution and the size of the input of the model is 100×100 pixel resolution.

4.2 Performance Metrics

The performance of the models trained and tested on the two datasets are shown in the following table.

Table 4.1: Performance of the models.

	Accuracy	Precision	Recall	F_1 score	Sensitivity	Specificity
Individual cells	0.953	0.953	0.953	0.953	0.947	0.959
Thick patches	0.989	0.989	0.989	0.989	0.993	0.982

The performance of the mixed test is shown in the following table;

Table 4.2: Performance of the first model on the patches of the thick smear images.

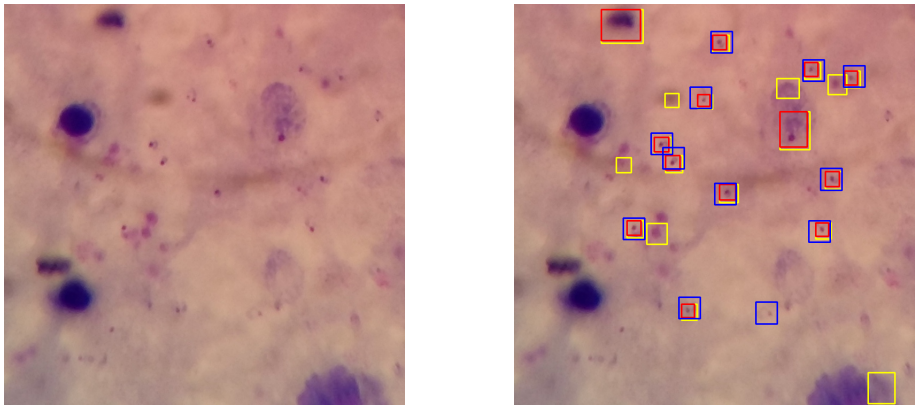
	Accuracy	Precision	Recall	F_1 score	Sensitivity	Specificity
Model tested on patches	0.895	0.893	0.895	0.895	0.894	0.896

According to the results in Table 4.1, our models performed a good classification for the two datasets. For the first model, the accuracy value shows that 95% of the first dataset were correctly classified and for the second model, 98% of the second dataset were correctly classified. The high values of the precision, recall and F_1 score for the two models give information about their good performances.

Despite the difference in size between the patches and the input of the model, Table 4.2 also shows us the good performance of the mixed test because from the accuracy value, 89% of the second dataset were correctly classified by the first model. And also, the value of precision, recall and F_1 score are quite high.

4.3 Parasites Detection

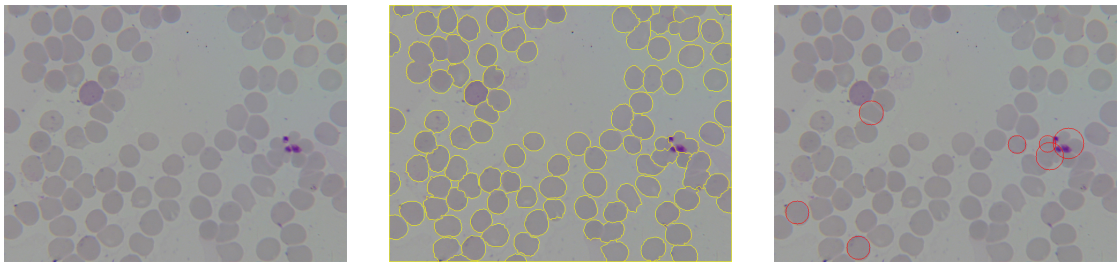
Let us now see the results of the cells detection and segmentation (from a single thick smear image and a single thin smear image) and the detection of the parasites on these images from the models.



(a) Thick blood smear image (b) Parasites detection

Figure 4.2: Parasites detection on a thick smear image

The Figure 4.2 shows the detection of parasites in a thick blood smear image. Figure 4.2a is the original image and Figure 4.2b is the result of our algorithm. The yellow rectangles show the candidate parasites, the red boxes show the detected parasites from the model and the blue boxes show the parasites from the annotations data.



(a) Thin blood smear image (b) Cells segmentation (c) Infected cells detection

Figure 4.3: Cells segmentation and infected cells detection on a thin smear image

Figure 4.3 shows the result of the segmentation using watershed transform and detection of the infected cells by our model. Figure 4.3a is the original thin smear image, Figure 4.3b shows the result of our segmentation which gives good result and Figure 4.3c gives the infected cells predicted by our model. The cells circled in red are those detected as infected by the model.

5. Conclusion

In conclusion, we can say that, our deep learning models trained on two different datasets have given good results for malaria parasites classification. In contrast to some machine learning models, which need some techniques for feature extractions, deep learning model with less work has shown a better efficiency in this type of problem. For example, Dong et al. [7] obtained an accuracy of 0.92 using support vector machine (SVM) and Das et al. [6] obtained an accuracy of 0.84 and 0.83 with Bayesian learning and SVM respectively for malaria parasites classification on a stained thin blood smear. However, it is preferable to have different models for thick and thin smear image.

We have also seen that combining these models with image processing methods gave us automated malaria detection. This automated detection for malaria can help the technician to reduce their work and do not need any presence of experts. To be able to use this in real life, we need to have a microscope and a device to put the necessary software, precisely the model and the image processing algorithms. For example, we can use a laptop and connect it to the microscope, or use a smartphone and mount it on the microscope or use a mini-computer with display, attach an objective camera to the microscope and connect it with the mini-computer.

References

[1] A. Agapitos, M. O'Neill, M. Nicolau, D. Fagan, A. Kattan, A. Brabazon, and K. Curran. Deep evolution of image representations for handwritten digit recognition. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2452–2459. IEEE, 2015.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

[3] K. H. Berthold and P. Horn. Robot vision, 1986.

[4] D. Bibin, M. S. Nair, and P. Punitha. Malaria parasite detection from peripheral blood smear images using deep belief networks. *IEEE Access*, 5:9099–9108, 2017.

[5] O. Cuisenaire. Distance transformations: fast algorithms and applications to medical image processing. Technical report, 1999.

[6] D. K. Das, M. Ghosh, M. Pal, A. K. Maiti, and C. Chakraborty. Machine learning approach for automated screening of malaria parasite using light microscopic images. *Micron*, 45: 97–106, 2013.

[7] Y. Dong, Z. Jiang, H. Shen, W. D. Pan, L. A. Williams, V. V. Reddy, W. H. Benjamin, and A. W. Bryan. Evaluations of deep convolutional neural networks for automatic identification of malaria infected cells. In *2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 101–104. IEEE, 2017.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] J. P. Martinez Garcia. Applying deep learning/gans to histology image for data augmentation: a general study. Retrieved from http://hdl.handle.net/10609/91330, Accessed, January 2018.

[11] World Health Organization. *Guidelines for the treatment of malaria*. World Health Organization, 2015.

[12] World Health Organization. *Malaria microscopy quality assurance manual-version 2*. World Health Organization, 2016.

[13] World Health Organization. *World malaria report 2019*. World Health Organization, 2019.

[14] N. Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[15] M. Poostchi, K. Silamut, R. J. Maude, S. Jaeger, and G. Thoma. Image analysis and machine learning for detecting malaria. *Translational Research*, 194:36–55, 2018.

[16] J. A. Quinn, R. Nakasi, P. K. Mugagga, P. Byanyima, W. Lubega, and A. Andama. Deep convolutional neural networks for microscopy-based point of care diagnostics. In *Machine Learning for Healthcare Conference*, pages 271–281, 2016.

[17] S. Rajaraman, S. K. Antani, M. Poostchi, K. Silamut, M. A. Hossain, R. J. Maude, S. Jaeger, and G. R. Thoma. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568, 2018.

[18] S. Saha. A comprehensive guide to convolutional neural networks: the eli5 way. *Towards Data Science*, 15, 2018.

[19] N. Tangpukdee, C. Duangdee, P. Wilairatana, and S. Krudsood. Malaria diagnosis: a brief review. *The Korean journal of parasitology*, 47(2):93, 2009.

[20] H. Tulsani, S. Saxena, and N. Yadav. Segmentation using morphological watershed transformation for counting blood cells. *IJCAIT*, 2(3):28–36, 2013.

[21] F. Yang, M. Poostchi, H. Yu, Z. Zhou, K. Silamut, J. Yu, R. J. Maude, S. Jaeger, and S. Antani. Deep learning for smartphone-based malaria parasite detection in thick blood smears. *IEEE journal of biomedical and health informatics*, 2019.

[22] X. Yang. *Introduction to Algorithms for Data Mining and Machine Learning*. Academic Press, 2019.

APPENDICES

A Codes Implementation for Training and Testing the Models

```
import os
import glob
import pandas as pd
import numpy as np
#for splitting data
from sklearn.model_selection import train_test_split
from collections import Counter
import cv2
from concurrent import futures
from sklearn.preprocessing import LabelEncoder
import threading


np.random.seed(42)

def resize_img(idx, img, IMG_DIMS, total_img):
    if idx % 5000 == 0 or idx == (total_img - 1):
            print('{}: working on img num: {}'.format(threading.
    current_thread().name, idx))
    img = cv2.imread(img)
    img = cv2.resize(img, dsize = IMG_DIMS,
                         interpolation = cv2.INTER_CUBIC)
    img = np.array(img, dtype=np.float32)
    return img

def data_preparation(base_dir, IMG_DIMS):
    try:
        infected_dir = os.path.join(base_dir,'Parasitized')
        healthy_dir = os.path.join(base_dir,'Uninfected')
    except:
        print('Directory not found')

    #counting the number of the images in each files
    infected_files = glob.glob(infected_dir+'/*.png')
    healthy_files = glob.glob(healthy_dir+'/*.png')
    #print(len(infected_files), len(healthy_files))


    ############ PREPARING DATA ##############

    #build a dataframe for the data
    files_df = pd.DataFrame({'filename': infected_files + healthy_files,
                'label': ['malaria'] * len(infected_files) + ['healthy'] *
    len(healthy_files)}).sample(frac=1, random_state=42).reset_index(drop=
    True)
```

```
43    #print(files_df.head())
44
45    #split the data
46    train_files, test_files, train_labels, test_labels = train_test_split(
      files_df['filename'].values,
47
         files_df['label'].values,
48
         test_size = 0.3, random_state = 42)
49    train_files, val_files, train_labels, val_labels = train_test_split(
      train_files,
50
      train_labels,
51
      test_size = 0.1, random_state = 42)
52    print("training data: ", train_files.shape, " \n ", Counter(
      train_labels))
53    print("validation data: ", val_files.shape, '\n ', Counter(val_labels))
54    print("test data: ", test_files.shape, '\n ', Counter(test_labels))
55
56
57    ex = futures.ThreadPoolExecutor(max_workers=None)
58    train_data_inp = [(idx, img, IMG_DIMS, len(train_files)) for idx, img
      in enumerate(train_files)]
59    val_data_inp = [(idx, img, IMG_DIMS, len(val_files)) for idx, img in
      enumerate(val_files)]
60    test_data_inp = [(idx, img, IMG_DIMS, len(test_files)) for idx, img in
      enumerate(test_files)]
61
62    #resize images in each data
63    print('Loading Train Images:')
64    train_data_map = ex.map(resize_img,
65                            [record[0] for record in train_data_inp],
66                            [record[1] for record in train_data_inp],
67                            [record[2] for record in train_data_inp],
68             [record[3] for record in train_data_inp])
69    train_data = np.array(list(train_data_map))
70
71    print('\n Loading Validation Images:')
72    val_data_map = ex.map(resize_img,
73                            [record[0] for record in val_data_inp],
74                            [record[1] for record in val_data_inp],
75                            [record[2] for record in val_data_inp],
76             [record[3] for record in val_data_inp])
77    val_data = np.array(list(val_data_map))
78
79    print('\n Loading Test Images:')
80    test_data_map = ex.map(resize_img,
81                            [record[0] for record in test_data_inp],
82                            [record[1] for record in test_data_inp],
83                            [record[2] for record in test_data_inp],
84             [record[3] for record in test_data_inp])
85    test_data = np.array(list(test_data_map))
86
```

```
    #train_data = [resize_img(train_files[i], IMG_DIMS) for i in range(len(
    train_files))]


    ### normalize data and save it
    train_imgs = train_data/255.
    val_imgs = val_data/255.
    test_imgs = test_data/255.


    le = LabelEncoder()
    le.fit(train_labels)
    train_labels = le.transform(train_labels)
    val_labels = le.transform(val_labels)
    test_labels = le.transform(test_labels)

    #to save the data
    #np.savez('train.npz', img=train_imgs, label =train_labels)
    #np.savez('val.npz', img=val_imgs, label=val_labels)
    #np.savez('test.npz', img= test_imgs, label=test_labels)

    return train_imgs, train_labels, val_imgs, val_labels, test_imgs,
    test_labels
```

Listing 5.1: Preparation individual cells data

```
import cv2
import numpy as np
import xml.etree.ElementTree as ET
import os
import glob
from sklearn.model_selection import train_test_split


#taking the position of the object(plasmodium) in an image if there exist
def get_position_plasmodium(image_name):
    xml_file = image_name[:-3] + 'xml'

    tree = ET.parse(xml_file)
    root = tree.getroot()

    #if there is no object
    if len(root.findall('object')) == 0:
        return np.array([])
    else:
        pos_plasm = []
        for item in root.findall('object'):
            box = item.find('bndbox')
            xmin = round(float(box.find('xmin').text))
            xmax = round(float(box.find('xmax').text))
            ymin = round(float(box.find('ymin').text))
            ymax = round(float(box.find('ymax').text))

            xmin, xmax, ymin, ymax = int(xmin), int(xmax), int(ymin), int(
```

```
    ymax)
29          pos_plasm.append((xmin, xmax, ymin, ymax))
30       return np.vstack(pos_plasm)
31
32
33 ####### Create a data from a pacthes of an image using the annotated data
    ##################
34
35 # w, h are the width and height of the patches
36 # num_neg is the number maximal of the negative patches(without plasmodium
    )
37 def create_patches(dir_, w, h, num_neg):
38    try:
39       image_dir = glob.glob(dir_ + '/*.jpg')
40    except:
41       print('Directory not found')
42    print('Total images: ',len(image_dir))
43    count_no_object = 0
44    count_object = 0
45
46    #the output patches and label
47    patches = []
48    labels = []
49    num_image = 0
50
51    print('########## Data creation ##########')
52    #getting the patches of the plasmodium and create a random negative
    patches
53    for image in image_dir:
54       num_image += 1
55       if num_image % 500 == 0 or num_image == len(image_dir)-1:
56          print('{} images done'.format(num_image))
57
58          #taking the object present in an image
59       bbox = get_position_plasmodium(image_name=image)
60       im = cv2.imread(image)
61       width, height, c = im.shape
62
63          #taking the positive patchs
64
65       if len(bbox) == 0:      #if no object
66          count_no_object += 1
67       else:
68          for box in bbox:     #for each plasmodium create a patch from the
     image
69             x = np.where(box[0] < 0, 0, box[0])
70             y = np.where(box[2] < 0, 0, box[2])
71             patches.append(im[y:y+h, x:x+w])
72             count_object += 1
73             labels.append(1)   #label corresponding to the presence of
    plasmodium
74
75
76          #taking the random negative patches
```

```
77
78          for j in range(num_neg):
79              x = np.random.randint(low=0, high=width)
80              y = np.random.randint(low=0, high=height)
81              xmax = w + x
82              ymax = h + y
83
84                  #looking for overlap between positive and negative patch
85                  #not consider the patche if there is a plasmodium inside
86              overlap = False
87              if len(bbox) == 0:
88                  pass
89              else:
90                  for box in bbox:
91                              #avoid overlap on the patches of positive and
    negative
92                      xmin, xmax, ymin, ymax = box
93                      xmin = np.where(xmin<0,0,xmin)
94                      ymin = np.where(ymin<0,0,ymin)
95                      cx = xmin + int((xmax - xmin)/2)
96                      cy = ymin + int((ymax - ymin)/2)
97                      if x<= cx <= xmax and y <= cy <= ymax:
98                          overlap = True
99
100             if not overlap:
101                 patches.append(im[y:y+h, x:x+w])
102                 labels.append(0) #label corresponding to a negative patch
103
104     print('Number of object {}'.format(count_object))
105     print('Number of images without object {}'.format(count_no_object))
106     return patches, labels
107
108
109 ######## create an array from the patches, labels and split the data
        ################
110 def data_creation(dir_, IMG_DIMS, w, h, num_neg):
111     patches, labels = create_patches(dir_, w, h, num_neg)
112     labels = np.asarray(labels)
113     #resize the image data and split in to train, validation and test
114     train = np.zeros((len(patches),IMG_DIMS[0],IMG_DIMS[1],3))
115     for i in range((len(patches))):
116         train[i] = cv2.resize(patches[i],IMG_DIMS)
117
118     print('The number of data {}'.format(len(train)))
119
120
121     train_imgs, test_imgs, train_labels, test_labels = train_test_split(
        train,
122
            labels,
123
            test_size = 0.3, random_state = 42)
124     train_imgs, val_imgs, train_labels, val_labels = train_test_split(
        train_imgs,
```

```
    train_labels,
    test_size = 0.1, random_state = 42)
    print('Number of train images {}'.format(len(train_imgs)))
    print('Number of test images {}'.format(len(test_imgs)))
    print('Number of validation images {}'.format(len(val_imgs)))

    train_imgs = train_imgs/255.
    test_imgs = test_imgs/255.
    val_imgs = val_imgs/255.

    #to save the data
    #np.savez('train.npz', img=train_imgs, label =train_labels)
    #np.savez('val.npz', img=val_imgs, label=val_labels)
    #np.savez('test.npz', img= test_imgs, label=test_labels)

    return train_imgs, train_labels, test_imgs, test_labels, val_imgs,
    val_labels
```

Listing 5.2: Preparation thick smear data

```
import tensorflow as tf
import numpy as np

tf.random.set_seed(42)

################################### CNN MODEL ##########################
def CNN_model(INPUT_SHAPE):
    input = tf.keras.layers.Input(shape=INPUT_SHAPE)

    conv1 = tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu
    ', padding='same')(input)
    pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu
    ', padding='same')(pool1)
    pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='
    relu', padding='same')(pool2)
    pool3 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv3)

    flatten = tf.keras.layers.Flatten()(pool3)

    hidden1 = tf.keras.layers.Dense(512, activation='relu')(flatten)
    drop1 = tf.keras.layers.Dropout(rate=0.3)(hidden1)
    hidden2 = tf.keras.layers.Dense(512, activation='relu')(drop1)
    drop2 = tf.keras.layers.Dropout(rate=0.3)(hidden2)

    output = tf.keras.layers.Dense(1, activation='sigmoid')(drop2)

    model = tf.keras.Model(inputs=input, outputs=output)
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
    model.summary()

    return model
```

Listing 5.3: Model creation

```
import tensorflow as tf
import datetime
import time
import os
import numpy as np

import matplotlib.pyplot as plt

############### PLOT ACCURACY #########
def plot_accuracy(history, model_name):
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
    t = f.suptitle('Basic CNN Performance', fontsize=12)
    f.subplots_adjust(top=0.85, wspace=0.3)

    max_epoch = len(history.history['accuracy'])+1
    epoch_list = list(range(1,max_epoch))
    ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
    ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accuracy')
    ax1.set_xticks(np.arange(1, max_epoch, 5))
    ax1.set_ylabel('Accuracy Value')
    ax1.set_xlabel('Epoch')
    ax1.set_title('Accuracy')
    l1 = ax1.legend(loc="best")

    ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
    ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
    ax2.set_xticks(np.arange(1, max_epoch, 5))
    ax2.set_ylabel('Loss Value')
    ax2.set_xlabel('Epoch')
    ax2.set_title('Loss')
    l2 = ax2.legend(loc="best")
    plt.savefig('figures/'+model_name+'_Accuracy.png')
    plt.show()


############# TRAINING THE MODEL ##################
def training(model, train_imgs, train_labels, val_imgs, val_labels, BATCH_SIZE, EPOCHS, model_name):
    start_time = time.time()
    logdir = os.path.join('./callbacks/', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

    tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
```

```
    factor=0.5,
                                                patience=2, min_lr=0.000001)


    callbacks = [reduce_lr, tensorboard_callback]

    history = model.fit(x=train_imgs, y=train_labels,
                        batch_size=BATCH_SIZE,
                        epochs=EPOCHS,
                        validation_data=(val_imgs, val_labels),
                        callbacks=callbacks,
                        verbose=1)

    elapsed_time = time.time() - start_time
    print("Time of the training")
    print(time.strftime("%H:%M:%S", time.gmtime(elapsed_time)))

    plot_accuracy(history, model_name)
    model.save('models/'+model_name+'.h5')
    return model
```

Listing 5.4: Training

```
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import LeaveOneOut
from sklearn import metrics
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

def predict(model, test_imgs, batch_size):
    test_label_predict = model.predict(test_imgs, batch_size=batch_size)
    return test_label_predict

def plot_AUC_ROC(model, test_imgs, test_labels, model_name):
    loo = LeaveOneOut()
    print('### Cross validation ####')
    i = 0
    ##cross validation
    for test_index, drop_index in loo.split(test_imgs, test_labels):
        if i%250 == 0 or i == len(test_imgs)-1:
            print('{} images done'.format(i))
        test_im, drop_test = test_imgs[test_index], test_imgs[drop_index]
        test_lab, drop_label = test_labels[test_index], test_labels[
    drop_index]
        i += 1


    test_label_predict = predict(model, test_im, batch_size=512)
    fpr, tpr, _ = roc_curve(test_lab, test_label_predict[:,0])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='ROC curve (area = {0:0.2f})'.format(roc_auc),
    linewidth=2.5)
    plt.plot([0, 1], [0, 1], 'k--')
```

```
    plt.xlim([-0.02, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.savefig('figures/'+model_name+'_AUC.png')
    plt.show()


##### CONFUSION METRICS #######

def get_metrics(true_labels, predicted_labels):
    cm = metrics.confusion_matrix(true_labels, predicted_labels)
    tp = cm[0,0]
    tn = cm[1,1]
    fp = cm[0,1]
    fn = cm[1,0]

    sensitivity = tp/(tp+fn)
    specificity = tn/(tn+fp)
    return {
          'Accuracy': np.round(metrics.accuracy_score(true_labels,
    predicted_labels), 4),
          'Precision': np.round(metrics.precision_score(true_labels,
    predicted_labels, average='weighted'),4),
          'Recall': np.round(metrics.recall_score(true_labels,
    predicted_labels, average='weighted'), 4),
          'F1 Score': np.round(metrics.f1_score(true_labels,
    predicted_labels, average='weighted'), 4),
          'Sensivity': np.round(sensitivity,4),
          'Specificity': np.round(specificity,4)
          }

def evaluation_metrics(model, test_imgs, test_labels):
    test_label_predict = predict(model, test_imgs , batch_size=512)
    test_label_predict[:] = [np.where(test_label_predict[i] > 0.5, 1, 0)
    for i in range(len(test_label_predict))]

    model_metrics = get_metrics(true_labels= test_labels, predicted_labels=
    test_label_predict)
    result = pd.DataFrame([model_metrics], index=['Basic CNN'])
    print(result)
```

Listing 5.5: Test

```
from Data_Ind_cells import data_preparation
from Data_thick import data_creation
from Train import training
from Test import plot_AUC_ROC, evaluation_metrics
from model import CNN_model
import os
import glob
import sys
```

```

BATCH_SIZE = 64
EPOCHS = 25

##### train model with the individual cells datasets
def ind_cells_cls(base_dir, IMG_DIMS, INPUT_SHAPE ):
    train_imgs, train_labels, val_imgs, val_labels, test_imgs, test_labels
    = data_preparation(base_dir, IMG_DIMS)
    model = CNN_model(INPUT_SHAPE)
    model_trained = training(model, train_imgs, train_labels, val_imgs,
    val_labels, BATCH_SIZE, EPOCHS, model_name='CNN_ind_cells')
    return model_trained, test_imgs, test_labels


##### train model with the thick smear images datasets
def thick_img_cls(dir_, IMG_DIMS, INPUT_SHAPE):
    train_imgs, train_labels, test_imgs, test_labels, val_imgs, val_labels
    = data_creation(dir_, IMG_DIMS, w=40, h=40, num_neg=20)
    model = CNN_model(INPUT_SHAPE)
    model_trained = training(model, train_imgs, train_labels, val_imgs,
    val_labels, BATCH_SIZE, EPOCHS, model_name='CNN_thick_imgs')
    return model_trained, test_imgs, test_labels

if __name__ == '__main__':
    directory = sys.argv[1]
    if directory == './cell_images' or directory == 'cell_images':
        print('you choose to train individual cells')
        base_dir = os.path.join(directory)
        CNN_ind_cells, test_imgs, test_labels = ind_cells_cls(base_dir,
    IMG_DIMS=(100,100), INPUT_SHAPE=(100, 100, 3))
        print('######## Test #######')
        plot_AUC_ROC(CNN_ind_cells, test_imgs, test_labels, model_name='
    CNN_ind_cells')
        evaluation_metrics(CNN_ind_cells, test_imgs, test_labels)

    elif directory == './plasmodium-phonecamera' or directory == '
    plasmodium-phonecamera':
        print('you choose to train thick smear images')
        dir_ = os.path.join(directory)
        CNN_thick_imgs, test_imgs, test_labels = thick_img_cls(dir_,
    IMG_DIMS=(50,50), INPUT_SHAPE=(50, 50, 3))
        print('######## Test #######')
        plot_AUC_ROC(CNN_thick_imgs, test_imgs, test_labels,model_name='
    CNN_thick_imgs')
        evaluation_metrics(CNN_thick_imgs, test_imgs, test_labels)
```

Listing 5.6: Main

B Codes Implementation for Automated Malaria Detection

```
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import sys
from Data_thick import get_position_plasmodium
import time


### function to look for overlapping
### intersection between two rectangles
def calculateIntersection(a0, a1, b0, b1):
    if a0 >= b0 and a1 <= b1: # Contained
        intersection = a1 - a0
    elif a0 < b0 and a1 > b1: # Contains
        intersection = b1 - b0
    elif a0 < b0 and a1 > b0: # Intersects right
        intersection = a1 - b0
    elif a1 > b1 and a0 < b1: # Intersects left
        intersection = b1 - a0
    else: # No intersection (either side)
        intersection = 0

    return intersection


def parasite_detection(image_name, model):
    t = time.time()
    try:
        image = cv2.imread(image_name)
    except:
        print('Image not found')
    image_copy = image.copy()
    gray = cv2.cvtColor(image_copy, cv2.COLOR_BGR2GRAY)

    #find of the plasmodium candidate in the image
    blur = cv2.GaussianBlur(gray, (3,3), 0)
    ret, thresh = cv2.threshold(blur, 110, 255, cv2.THRESH_BINARY_INV)

    cnt, hier = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
    count_candidate = 0
    count_plasmodium = 0

    #predicting the plasmodium canditates by our model
    listold=[(0,0,0,0)]
    for c in cnt:
        overlap = False
        x, y, w, h = cv2.boundingRect(c)
        if w < 50 or h < 50:  #taking out the white blood cells
            x = np.where(x > 10, x - 10, x)
            y = np.where(y > 10, y - 10, y)
            w += 20
            h += 20
            xmax = x+w
```

```
55          ymax = y+h
56          AREA = (xmax-x)*(ymax-y)
57
58          # look the overlapping
59          for xold, yold, xmaxold, ymaxold in listold:
60              width = calculateIntersection(xold, xmaxold, x, xmax)
61              height = calculateIntersection(yold, ymaxold, y, ymax)
62              area = width * height
63              ratio = area / AREA
64              if ratio > 0.1:   # if area of intersection is > 0.1 don't
    take the new rectangle
65                  overlap = True
66          if not overlap:
67              count_candidate += 1
68              Roi = image[y:y+h, x:x+w]
69              Roi_image = cv2.resize(Roi,(50,50)) / 255.
70              Roi_tf = tf.expand_dims(Roi_image, 0)
71
72              label = model.predict(Roi_tf)
73              label = np.where(label[0] > 0.5, 1, 0 )
74              cv2.rectangle(image_copy, (x,y), (x+w+5, y+h+5),(0, 255, 255),
     2)  ### yellow rectangle for the candidate plasmodium
75              if label == 1:
76                  cv2.rectangle(image_copy, (x,y), (x+w, y+h),(0, 0, 255), 2)
       ### red rectangle for the plasmodium by the model
77                  count_plasmodium += 1
78
79          xold = x
80          yold = y
81          xmaxold = x+w
82          ymaxold = y+h
83          listold.append((xold,yold,xmaxold,ymaxold))
84
85      #showing the plamodium from the annotated if there is
86      bbox = get_position_plasmodium(image_name=image_name)
87      if len(bbox) == 0:
88          pass
89      else:
90          for box in bbox:
91              x = box[0]
92              y = box[2]
93              w = box[1] - box[0]
94              h = box[3] - box[2]
95              cv2.rectangle(image_copy, (x,y), (x+w, y+h),(255, 0, 0), 2)  ####
     blue rectangle for the plasmodium by the annotation
96
97      elapsed_time = time.time() - t
98
99      cv2.imwrite(image_name[:23]+'detection.png', image_copy)
100     print('Time for detection: ', time.strftime("%H:%M:%S", time.gmtime(
    elapsed_time)))
101     print('Number of candidate plasmodium: ', count_candidate)
102     print('Number of plasmodium from the model: ', count_plasmodium)
103
```

```
    plt.figure(figsize=(10,10))
    plt.imshow(cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()


if __name__ == '__main__':
    model_dir = sys.argv[1]
    image_name = sys.argv[2]
    try:
        model = tf.keras.models.load_model(model_dir)
        model.compile(optimizer='adam',loss='binary_crossentropy',
                      metrics=['accuracy'])
        print('Model loaded successfully')
    except:
        print('Model not found')
    parasite_detection(image_name=image_name, model=model)
```

Listing 5.7: Thick image parasites detection

```
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
from skimage import measure
import sys
import time
import numpy as np

def fillholes(img):
    im = img.copy()
    h,w = im.shape[:2]

    ## the mask of the fill should be greater than the image
    mask = np.zeros((h+2, w+2), np.uint8)
    seedpoint = (0, 0) # starting point
    ## if the first pixel is not a background, check the closest background
     to be the starting point
    if im[0,0] != 0:
        for i in range(h):
            for j in range(w):
                if im[i,j] == 0:
                    seedpoint = (i,j)
                    break
    ## filling the holes
    _,im_fill,_,_ = cv2.floodFill(im, mask, seedpoint,255)
    im_fill_inv = cv2.bitwise_not(im_fill)
    ## combine the image filled with the original
    img_fill = img | im_fill_inv
    return img_fill

def segmentation_cells(image_name):
    try:
        image = cv2.imread(image_name)
    except:
```

```
        print('Image not found')
    im_copy = image.copy()

    ##### cells segmentation #########
    gray_im = cv2.cvtColor(im_copy, cv2.COLOR_BGR2GRAY)

    ## increase the contrast of the image
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(10,10))
    cl = clahe.apply(gray_im)

    #dividing the region in black and white
    ret, thresh = cv2.threshold(cl, 0, 255, cv2.THRESH_BINARY_INV | cv2.
    THRESH_OTSU)

    #removing the noise
    #morphology consists of dilating or eroding the white region
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations =
     3)

    open_new = fillholes(opening) #filling the holes

    # #find the sure bagckround
    sure_bg = cv2.dilate(open_new, kernel, iterations = 5)


    # #find the sure foreground
    dist_transform = cv2.distanceTransform(open_new, cv2.DIST_L2, maskSize
    =5)
    dist_transform =  np.uint8(dist_transform)
    ## the separation of the cells depend on this threshold, it will depend
     on the image,
    ##testing on different images, I found that 9 gave the best separation
    ret, sure_fg = cv2.threshold(dist_transform, 9, 255, cv2.THRESH_BINARY)

    sure_fg = cv2.erode(sure_fg, kernel)

    # #find the unkown region
    unknown = cv2.subtract(sure_bg, sure_fg)

    # #label the foreground objects
    ret, label = cv2.connectedComponents(sure_fg)


    # #add to all labels so that the sure_bg = 1 not 0
    label += 1

    # #label the unkown region as 0
    label[unknown == 255] = 0

    #find the contour of each cells
    markers = cv2.watershed(im_copy , label)

    ######## taking the regions with respect to the markers ########
```

```
    regions = measure.regionprops(markers, intensity_image=cl)

    #put into red these contours
    im_copy[markers == -1] = [0, 255, 255]
    cv2.imwrite(image_name+'seg.png', im_copy)
    plt.figure(figsize=(10,10))
    plt.imshow(cv2.cvtColor(im_copy, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
    return regions,image



     ####### cells classifications #########
def classification_cells(image_name, model):
    t = time.time()
    count_parasites = 0
    count_cells = 0
    regions, image = segmentation_cells(image_name)
    image_cp = image.copy()

    for prop in regions[1:]: #we took off the first region because it seems it's for the border
        count_cells += 1
        #take only the region containing one cell, accodring to their area
        if prop.area < 2000:
            y_min, x_min, y_max, x_max = prop.bbox ##### taking a box for each regions
            #take a patche of the image according to the bounding box
            cells = image[y_min:y_max, x_min:x_max]
            cell = cv2.resize(cells,(100,100))/255.

            cell = tf.expand_dims(cell, 0)

            predict = model.predict(cell)     #### predict the cell with our model
            predict = np.where(predict[0][0]>0.5,1,0)

            if int(predict) == 1:
                cy,cx = prop.centroid
                cv2.circle(image_cp, (int(cx), int(cy)), int((x_max-x_min)/2), (0,0,255), 1) ### put a circle in the infected cells
                count_parasites += 1

    elapsed_time = time.time() - t
    print('Time for segmentation and detection: ', time.strftime("%H:%M:%S", time.gmtime(elapsed_time)))
    cv2.imwrite(image_name+'cls.png', image_cp)
    print('Number of cells segmented {}'.format(count_cells))
    print('Number of infected cells {}'.format(count_parasites))

    plt.figure(figsize=(10,10))
    plt.imshow(cv2.cvtColor(image_cp, cv2.COLOR_BGR2RGB))
    plt.axis('off')
```

```
133     plt.show()
134 
135 
136 if __name__ == '__main__':
137     model_dir = sys.argv[1]
138     image_name = sys.argv[2]
139     try:
140         model = tf.keras.models.load_model(model_dir)
141         model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['
    accuracy'])
142         print('Model loaded successfully')
143     except:
144         print('Model not found')
145     classification_cells(image_name=image_name, model=model)
```

Listing 5.8: Thin image infected cells detection