Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/dcan

Image analysis and machine learning based malaria assessment system

Kyle Manning[a], Xiaojun Zhai[*a], Wangyang Yu[*b]

[a]**School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK, CO4 3SQ**
[b]**Key Laboratory of Modern Teaching Technology, Ministry of Education, School of Computer Science Shaanxi Normal University, Xi'an, China**

Abstract

Malaria is an important and worldwide fatal disease that has been widely reported by the World Health Organization (WHO), and it has about 219 million cases worldwide, with 435,000 of those mortal. The common malaria diagnosis approach is heavily reliant on highly trained experts, who use a microscope to examine the samples. Therefore, there is a need to create an automated solution for the diagnosis of malaria. One of the main objectives of this work is to create a design tool that could be used to diagnose malaria from the image of a blood sample. In this paper, we firstly developed a graphical user interface that could be used to help segment red blood cells and infected cells and allow the users to analyze the blood samples. Secondly, a Feed-forward Neural Network (FNN) is designed to classify the cells into two classes. The achieved results show that the proposed techniques has a potential to be used to detect malaria, as it has achieved 92% accuracy with a database that contains 27,560 benchmark images.

KEYWORDS:
Malaria assessment system, Image analysis, Image segmentation, Artificial intelligence

1. Introduction

Malaria is one of the most common diseases that has major side effects on health. Its history can be tracked back to the 16th century BC [1]. Although this disease has been well studied and the progress of the treatment has been investigated significantly, it still kills many people every year. The main cause of this problem is that the examination procedure is very complex, and the cost is not affordable in the developing countries. Due to recent development in the healthcare industry, if a suspected carrier of the disease is diagnosed and treated quickly, the disease can be cured as well as prevented [2]. Microscopic analysis is one of the current gold processes, but it is manual, complex and time consuming. Therefore, in order to improve the efficiency of the process, we develop an efficient Artificial Neural Network (ANN) architecture to assist the diagnosis process for reducing the time and cost of malaria examination. Such a low-cost automated diagnosis system is one of the important steps to enable the Internet-of-Things (IoT) based healthcare systems. The ideal hardware solution for malaria diagnosis in resource-poor settings would be a normal portable slide viewer, but we are still far from those field-usable devices even if modern technology is heading this way. One of the best alternative solutions is to use small camera-equipped computing devices, e.g. a mobile phone, which can be attached to a magnifying device and then compute the para-

*Wangyang Yu and Xiaojun Zhai are (Corresponding authors) (email: ywy191@snnu.edu.cn and xzhai@essex.ac.uk).

[1]Kyle Manning is a student at the School of Computer Science and Electronic Engineering, University of Essex (email: k_manning@live.co.uk).

[2]Xiaojun Zhai is a lecturer at the School of Computer Science and Electronic Engineering, University of Essex (email: xzhai@essex.ac.uk).

[3]Wangyang Yu is an associate professor at the School of Computer Science, Shaanxi Normal University (email: ywy191@snnu.edu.cn).

sitaemia automatically using image analysis and machine learning algorithms [3].

The process of manual malaria diagnosis takes a long time to complete and requires well-trained clinicians and expensive equipment to examine accurately. As a result, the accuracy of diagnosis is closely determined by the state of the person who operates the equipment. Factors including tiredness, nausea and headaches as well as other conditions may affect the ability of the practitioner to correctly diagnose the patient.

Secondly, the usability is another challenge. It is very difficult to allow all clinicians to perform this examination due to the needs of training and practices. Therefore, supplying a Graphical User Interface (GUI) would significantly improve the usability to avoid the need to understand complex coding practices or to be heavily trained in microscopic analysis. On the other hand, using an NN would reduce the average time of diagnosis. Therefore, a simplified NN architecture would also significantly reduce the run-time for cost sensitive IoT-based healthcare systems. In this work, we thus propose a simple Feed-Forward Neural Network (FFNN) called *patternnet* to overcome the above challenges.

There are two common ways to perform automatic malaria diagnosis: 1) Machine Learning (ML) based approaches e.g. using an ANN. 2) Traditional image processing based feature extraction and segmentation approaches. Typical approaches that have been adopted by clinicians include Rapid Diagnostic Tests (RDT), and Serology and Simian Malaria Species Confirmation Service (SMSCS). An ANN-based approach would allow more clinicians to diagnose patients quicker and further enhance the usability of tools. However, malaria detection is an open-ended challenge as there are many ways to deal with it. As a very practical area, there is not a set of algorithms that could work well to resolve the problem of malaria detection, as factors such as blood smear viscosity, image color and quality, the stage of malaria when the smear was taken, and the stain used on the blood smear, can affect the results.

This paper focuses on resolving the problem of malaria detection in challenging environments, e.g. the quality of diagnosis, and the accuracy of samples presented by medical clinicians. In addition, the proposed work also includes an improved segmentation algorithm for Red Blood Cells (RBC) in images of microscopic slides, and an NN-based classification algorithm to classify the cells as parasitized or uninfected. The major contributions of this work are highlighted as follows:

- The results achieved in this work demonstrate that the proposed NN classification and segmentation algorithms drastically improve the performance and success rate of malaria detection, additionally proving that human inaccuracies would be overcome.
- A user-friendly GUI is designed to assist detection of malaria parasites. The proposed system has a potential to assist the trained professionals who conduct a manual examination.

The rest of the paper is organized as follows: The second section provides a brief discussion on the related work. This is followed by a discussion on the system architecture used in this work in Section 3. In Section 4, we present an analysis of experimental results in detail. The article finally concludes in Section 5.

2. Literature review

2.1. Malaria

To this day malaria remains a common disease in tropical regions. Although significant efforts have been made to combat the disease, it continues to take many lives every year.

Malaria cannot survive outside of their host and is caused by the protozoan pathogen known as Plasmodium. Five species of malaria have been found in humans, including P. falciparum, P. ovale, P. vivax, P. malariae and the more recent P. knowlesi, which is usually found in animals but rare cases have been seen in humans [4]. According to the NHS [5], P. falciparum is mainly found in Africa and is thought to be the deadliest species of malaria as it is responsible for the majority of deaths worldwide.

2.2. Computer vision

2.2.1. Image processing

It is common for the first stage to be image processing when analysing digital images. This step performs algorithms on pictures to either extract information such as features and characteristics, or enhance the image to then be passed on to another action, such as image segmentation (see Section 2.2.2). With image processing, it is standard to extract or remove information that allows the important features of an image to be enhanced and makes them more accessible [6].

The aim of image processing for malaria detection is to change the image so that the output does not contain noise and highlights the RBC. However, it is likely that different computers, software and hardware will alter images in ways unexpected by the user, which is an issue not sufficiently discussed. Medical professionals will rely on the image to produce the same output regardless of computer specification or software application. When running a diagnosis program, if the resulting images are sent through an email client or copy and pasted, the image resolution is likely to change, consequently reducing the image quality [6]. High quality images are required to extract an adequate amount of information, therefore, whether to use

JPEG or PNG file types should be considered when providing program input because the compression algorithm is used in both.

2.2.2. Image segmentation

The process of segmentation is to divide an image into smaller subsections (set of pixels) [7], and the level of depth in which the segmentation continues is dependent on the application. In this paper, the full image is the blood smear, and the segmentation is to stop once it reaches the region of interest, thereby making the smaller subsections individual cells. In turn, the classification can work on a focused area of the full image and produce better results.

Attempting to classify cells on a full image could result in misclassifications because every blood smear and cell is different. When using the same stain (Giemsa in this case), it is still possible for the images processed digitally to output slightly different photos (see Figure 7). Solely using a full image would either require more pre-processing or an entirely different approach such as a fine-tuned CNN (Convolutional Neural Network). This is to ensure that what is inside the cell determines the classification.

There are multiple ways of segmenting an image. Because of this, it is the most problematic step in image processing. Segmentation is applied in many situations for different purposes, for example, shape recognition for numbers and letters (identifying number plates), augmented reality, medical industry (disease or human anatomy) and content-based visual information retrieval (objects derived from the image, e.g. colors, textures and shapes). The different techniques can be grouped as follows:

Thresholding is a type of segmentation which is done according to the pixels in the image. To partition the image, this technique works on the intensity levels of the pixels, whereby the background and foreground are the classes which have different intensities that the pixels are assigned. To do so, the algorithm performs a comparison on every pixel to compare the pixels intensity with the threshold set. If the intensity of the pixel is less than the threshold, it is placed in one class; and if it is higher, it is added to the other class. Because of this, it is commonly used to convert images in greyscale to binary. The most basic form can be shown according to [8] as:

$$I_{bw}(x,y) = \begin{cases} 1 & I_{gray}(x,y) \geq T \\ 0 & I_{gray}(x,y) < T \end{cases} \quad (1)$$

where I_{gray} is the grayscale image, I_{bw} is the binarized image, (x, y) represent the coordinates of the pixel in the image and T is the threshold. Watershed is a form of region-based segmentation that separates the image into catchment basins. A catchment basin is a location in a lower region which collects water as a single water body from a higher region as it falls. As mentioned by [9], these are the objects to be identified by the program. In segmentation, the Watershed method is considered a type of topographic tool because of the way it fills the catchment basins. In essence, region-based techniques work in a way opposite to edge-based methods because they group pixels of similar colors, intensity levels, texture or shape. As a result, the sub-regions are formed with thereby creating barriers or edges to stop the sub-regions from merging. Taking the pixel intensity as an example, the intensity determines the altitude.

The challenge of implementing this technique is that it usually suffers from oversegmentation because of the number of sub-regions it creates . Due to noise and intensity irregularities, it creates another “local minima” and is unable to make correct distinctions between regions [7]. It can be improved or entirely avoided using techniques such as MM (Mathematical Morphology) or the marker-controlled Watershed technique which marks each region of interest as a ”local minima” to minimize the creation of insignificant or small areas [10][11].

MM was first introduced by Matheron and expanded by various image processing applications [12][13]. It is not a segmentation technique in itself but is a technique that contains a set of operations used in conjunction with other segmentation methods to obtain a better result based on shapes. The purpose is to remove irrelevant artifacts in an image using a set of transformations. The most common methods are dilation and erosion, which are normally used together.

The operators work on binary or grayscale images, which compare each pixel with its surroundings to produce the pixel in the output image. During dilation, if any surrounding pixel has the value of 1, then the output pixel is set to 1. This is used to fill small areas in objects. Whereas Erosion does the opposite: if any surrounding pixel has the value of 0, then the output pixel is set to 0, this is used to remove small areas which neither aid in object detection nor provide useful information [12].

2.3. Machine learning

ML is a type of AI (Artificial Intelligence) which aims to learn patterns in data to classify them into different categories [24]. Within ML, there are many algorithms for various purposes. There are two main classes of ML: UL (Unsupervised Learning) and SL (Supervised Learning). In UL, an assortment of data is analyzed to find patterns, there is no prior knowledge of the data and it is up to the algorithm to decide how to sort the data. In contrast, SL requires two variables in the algorithm: the unique features of the data as input and the desired targets as the output. The algorithm then attempts to learn the data from the features and maps them to the outputs. This is to create a near accurate mapping function so that when presented with new data, it can classify them

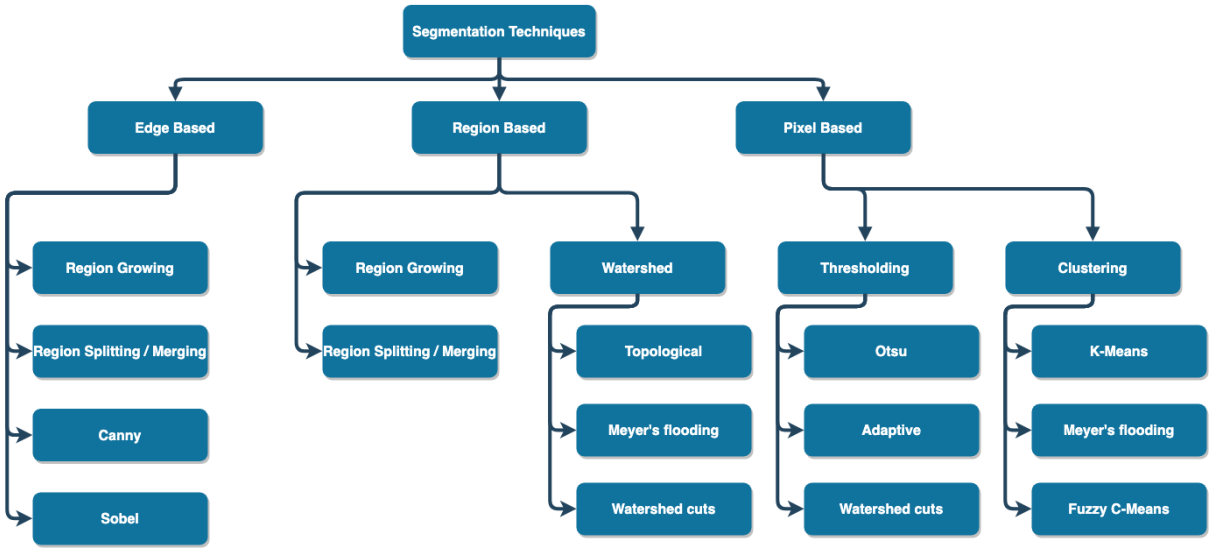

Fig. 1. Segmentation techniques

Ref	Algorithm	Pros	Cons
[14], [15], [16], [17]	Watershed	• Fast • Simple • Ease of use • Better handling of gaps • Better boundary placement with high contrast	• Suffers from oversegmentation • Noise sensitivity • Poor boundary placement of important regions with low contrast • Poor detection of thin objects
[18], [19], [20], [21], [22], [23]	K-Means	• Efficient • Scalable for larger datasets • Unsupervised learning • Low computational costs	• "K" is difficult to estimate • Accuracy is dependent on the initial "K" value • Doesn't guarantee continuous areas
[7], [18], [22], [23]	Thresholding	• Improved region detection with images of high contrast • Simple calculation	• Sensitive to noise because of greyscale overlapping • Intense homogeneity
[7], [20], [21], [23]	Fuzzy C-Means	• Generates good results • Efficient • Always converges • Unsupervised learning	• Takes long to compute • Sensitive to initial cluster center • Sensitive to noise
[20], [23]	Neural Networks	• Automatically learns new patterns and features • Scalable • Can be efficient based on implementation • Achieves the state-of-the-art results • Works well on large datasets • Robust	• Complex implementations take longer to run and require large training data • Can suffer from overtraining/overfitting • Underfitting where it can't learn the training data and therefore can't adapt to new data

Table 1. Comparison of segmentation techniques

correctly. One type of ML commonly used is ANNs, which are connected structures made to resemble the neural structure of the brain in humans. Every ANN consists of multiple interconnected layers in which the data is sent through each layer, usually an input layer, then one or more hidden layers which have multiple neurons and finally an output layer.

The purest form of NNs is the FFNN because information only flows in one direction, in contrast to RNN (Recurrent Neural Networks) [25] in which information is cycled and retains data from previous loops. A CNN is also a form of FFNN [26]. The most basic

FFNN consists of only three: input, hidden (using the sigmoid transfer function) and output (using the softmax transfer function), whereas more complex CNNs consist of more specific layers to allow for more customization. The input layer in a CNN uses raw images which are fed into the network. The next layer, namely, the convolutional layer, extracts features from regions of the image. In a simple CNN, the next layer is ReLU (Rectified Linear Unit) which uses an activation or transfer function of which the simplest being a threshold-based function (see Section 2.2.2). After ReLU is pooling, this layer reduces the size of the input image (the dimensions, height and width) through down-sampling using the neurons from the previous layer. Pooling also reduces the overall network size. The final layer usually is FC (Fully Connected), which connects all the neurons in one layer to all the neurons in another layer and works the same as in MLP (Multi-Layer Perception).

In general, the larger the dataset used for training, the better the algorithm should perform, because it is using more data to construct a better understanding of how to classify the images. More training data allows the algorithm to generate a more complex function for mapping, which should then lead to higher accuracy when testing.

2.4. Discussion

Image segmentation is not limited to one or two methods, and all of the methods have their own benefits and drawbacks, so choice comes down to the situation. Table 1 discusses the most commonly used techniques in image segmentation.

It's clear from Table 1 that if the application is based on a large dataset then an NN might prove to be a viable option as it can produce better results using complex patterns. If the intention is focused on improving segmentation alone, then adapting the threshold technique in conjunction with another method such as KM or Watershed could prove beneficial in smaller applications. Other changes can accommodate NNs and segmentation when used together. Therefore, the proposed work focuses on developing a unified solution to obtain a low-cost segmentation and classification in resource constraint environments.

3. System architecture

The proposed system architecture consists of two major parts: 1) segmentation; and 2) classification. In the segmentation part, the blood smears have firstly been separated using the segmentation algorithm to filter out the infected cells. The algorithm used in this paper is called the Watershed technique. Since the original Watershed technique is not accurate enough to segment the infected cells for the reasons we discussed previously, we propose an improved segmentation algorithm combined with an FFNN to assist the identification of benign and infected cells. In the classification part, we also propose a lightweight feature extraction algorithm to enhance the classification process to improve the accuracy of the NN. In the following subsections, we firstly investigate the initial segmentation and classification algorithms in Section 3.1 and 3.2, and then we introduce the proposed segmentation and classification algorithms respectively in Section 3.4 and 3.5. The overall system flowchart is detailed in Figure 2.

3.1. Initial segmentation

Oversegmentation is one of the main problems when using the original Watershed technique (i.e. Algorithm 1) if the image is not preprocessed. The image will be firstly converted into greyscale before Watershed is run on it.

Algorithm 1 Original Watershed Segmentation
Input: *image*
Output: *segmentedImage*

```
1: function SEGMENTATION_WATERSHED(im)
2:     imageRGB ← im  ▷ Read image into program
3:     convert image from RGB to grayscale
4:     segmentedImage ← WATERSHED(im)
5:     return segmentedImage
6: end function
```

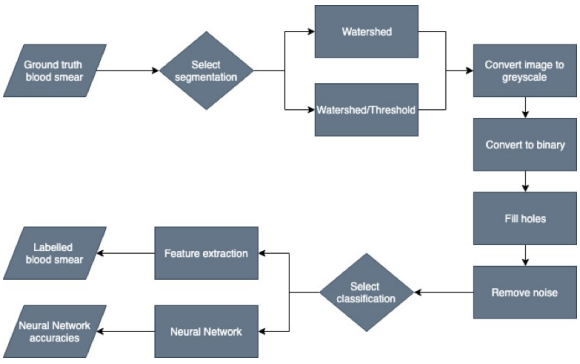

Fig. 2. Program flowchart

Algorithm 2 was proposed by MathWorks [27] to improve the original Watershed algorithm, however, it cannot fully identify lighter cells and highlighted cells at the edge of the image. The regions of interest are highlighted across the image in bright RGB colors and the background is blue. In Algorithm 2, lines 4 - 8 perform several operations on the image to further enhance the objects from the background. The gradient magnitude shows how the grey levels change in the image. The Sobel algorithm, namely, used for edge detection. In addition, a set of morphological operations, namely, imopen, imerode and imreconstructe

are used to remove objects smaller than a set number of pixels and small holes and identify the areas of the image with high intensity respectively. It also converts the image to binary using a threshold and then calculates the distance between each "0" and non-zero pixel on lines 6 and 7. Lines 8 – 16 run the Watershed method [27], and then displays the results in the GUI.

3.2. Initial classification

The Deep Learning Toolbox(DLT) provides a tool named *nprtool* for pattern recognition, which creates a basic FFNN using patternnet which requires an array of the image elements. However, to retrieve better results from the algorithm, the input should be features of the image. The *nprtool* creates a GUI, but at the end of the program it is possible to automatically save the default file which is created to run the NN. This file can then be modified from the default settings. Once the file is saved, the user can then run the NN through the command line without repeating the process in *nprtool*. A basic and advanced script file is created by the tools named *NN_Basic_Script.m* and *NN_Advanced_Script.m* respectively, which have been altered to update the ratios between the fields: training, testing and validation with the hidden layer size.

Previously, the NN received a 2D image that was reshaped to 1D as input and resized to $64 \times 64 = 4096$ due to the large original images. The 1D image input was replaced in the pre-processing stage with features of the images, and this leads to a smaller input but more accurate results. The new process is shown in Section 3.5.

3.3. Image pre-processing

Images need to be pre-processed in order to allow the algorithms to produce a successful output when run against them. Images in a dataset after the pre-processing need to be uniformed regardless of the original appearance. Nonetheless, in every case the images should be converted to grayscale because then they can be adjusted based on the grey level intensity.

3.4. Improved segmentation

To improve the previous Watershed techniques (see Algorithm 2 and Algorithm 1), we propose to use a combination of thresholding and the Watershed technique. The pseudo code for the improved segmentation is presented in Algorithm 4. To solve the problem of not selecting all the cells, thresholding the image creates a better extraction of foreground and background (see Figure 11) whilst removing noise using MM.

The improved segmentation (see Table 3.4) operates by converting the image to greyscale, generating an adaptive threshold, and then applies the threshold to convert the image to binary after clearing the border on lines 3 - 5. A set of operations (MM, distance transform and Gaussian filter) are then run from lines 6 - 9, which removes noise.

Algorithm 2 Marker based Watershed Segmentation

Input: *image*
Output: *stats* ▷ Bounding area of the extracted cells

1: **function** SEGMENTATION_WATERSHED_MARKER(*im*)
2: *imageRGB* ← *im* ▷ Read image into program
3: convert image from RGB to grayscale
4: invert image
5: *gmag* ← IMGRADIENT(*invertedImage*)
6: MATHEMATICAL MORPHOLOGICAL OPERATIONS(*invertedImage*)
7: convert image to binary
8: compute distance transform
9: *segmentedImage* ← WATERSHED
10: convert matrix of labels to RGB image
11: *stats* ← LABEL2RGB(*segmentedImage*) ▷ calculate coordinates of cells in image
12: show RGB image with labels layered over
13: **for** each bounding region *stats*, *kk* ∈ {1, . . . , HEIGHT(*stats*)} **do**
14: draw bounding region as rectangle
15: add cell count next to rectangle
16: **end for**
17: **return** *stats*
18: **end function**

3.5. Improved classification

Cell classification heavily relies on the segmentation method and the NN (see Algorithm 3), where the NN is trained to work on a variety of images such as both smears and single cells. As previously mentioned, all images need to be pre-processed, which entails processing the original image and storing them in a location to be used. A typical FFNN consists of multiple layers of neurons in the feed forward architecture, where each perception layer feeds a non-linear mapping of its inputs to the next layer. The non-linearity of this mapping is contributed by an appropriately chosen activation function.

Given that the FFNN is a fully connected network, all neurons in a layer are weight-combined and connected to neurons in the next layer. During the training phase of the neural network, the weight function is defined. Figure 3 is an example of an MLP network, where a hidden layer consists of S neurons and each neuron contains R weights corresponding to the number of inputs.

The input weights $\mathbf{W}$ can be written as an $S \times R$ matrix shown in Equation 2.

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix} \tag{2}$$

For an input vector $\vec{p}$ consisting of R elements $(p_1, p_2, p_3, \dots, p_R)$, the response q_s of the sth neuron

Algorithm 3 Improved NN Classification

Input: *imgCount*

Output: *malariaInputs*, *malariaOutputs* ▷ *malariaInputs*: $n \times 5$ matrix (5 is the features of the images and n is the total number of images), *malariaOutputs*: classification targets represented as $n \times 2$ matrix (2 is the amount of classifications possible and n is the number of images presented)

1: **function** NN(*imgCountInput*)
2: create destination location
3: create datastore for healthy/unhealthy cells
4: shuffle both datastores
5: **if** *imgCountInput* == *null* || *imgCountInput* ≠ *digit* **then**
6: *imgCount* = 1000
7: **else**
8: *imgCount* = imgCountInput
9: **end if**
10: *initialMalariaInputs* = matrix of all zeros (image count x total features)
11: **for** $i \in \{1, \ldots, imgCount\}$ **do**
12: generate output filename
13: **if** $i \leq imgCount \div 2$ **then**
14: generate input filename from unhealthy datastore
15: **else**
16: generate input filename from healthy datastore
17: **end if**
18: copy input file to destination with output filename
19: read output image file into program
20: resize and save output file to 64×64
21: convert output file from RGB to grayscale
22: adaptively threshold image with high sensitivity
23: convert image to binary using threshold
24: MATHEMATICAL MORPHOLOGICAL OPERATIONS(*binaryImage*)
25: convert image from uint8 to double
26: *features* = generate array of image features
27: append *features* to *initialMalariaInputs*
28: **end for**
29: *randPerm* = generate random permutation of image count
30: $fHalf \leftarrow$ GENERATEMALARIATARGETS(*malariaTargets*) ▷ generate targets in the form 0 1
31: $sHalf \leftarrow$ GENERATEMALARIATARGETS(*malariaTargets*) ▷ generate targets in the form 1 0
32: *initialMalariaInputs* ← CONCAT($fHalf$, $sHalf$) ▷ vertically concatenate the two matrices
33: shuffle *initialMalariaInputs* and *initialMalariaTargets* using *randPerm*
34: save *initialMalariaInputs* into file
35: save *initialMalariaTargets* into file
36: **end function**

can be described by Equation 3.

$$q_s = F((\vec{w}_s)^T \cdot \vec{p} + b_s) \tag{3}$$

where $\vec{w_s}$ corresponds to a row in $\mathbf{W}$, b_s is the bias associated with the input, and F is the activation function. In a feed-forward MLP architecture with more than one hidden layers, similar operations as Equation 3 are performed between two consecutive layers.

In the FFNN, three layers are used: an input layer (1 input with 5 elements), a hidden layer (consisting of 10 neurons) and an output layer (2 outputs representing the classes).

For the NN to work fairly and in an unbiased manner, the cells need to be processed and stored so they are readily available. An initial location and an initial amount of images (determined from the user input) are set. After that, it creates datasets using the paths for the healthy and unhealthy cells and all images in the path are shuffled to ensure random selection. Line 9 creates an empty matrix where the features are stored. Lines 10 – 27 form the main pre-processing loop, in which half of the images are healthy, the other half are infected and saved to the destination folder. The images are opened and resized so that they all have equal dimensions and are converted to binary using a threshold to make them clearer. MM operations are run on the images and converted to a class of double type to generate the features. The remaining lines generate

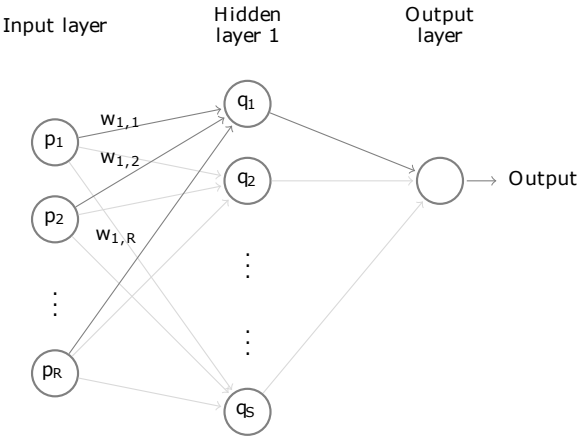

Fig. 3. A generic MLP architecture consists of an input layer of size R and a hidden layer with S neurons.

the targets based on the inputs, shuffle them again and finally save the inputs and targets to a file.

3.6. Implementation

The software framework is divided into the following 6 main blocks:

1. *Upload* calls *uploadbutton_Callback*, which displays a file brower to let the user select a blood smear.
2. *Select technique*, which lets the user select the segmentation method. It is called *popupmenu1_Callback*, which sets the variable *handles.first_menu* to the value from the drop-down.
3. *Run* calls *techrunbutton_Callback*, which executes the technique, stores the segmented image and table of regions and then enables the run button for extension.
4. *Select* extension which contains the classification techniques. It is called *popupmenu2_Callback*, which sets the variable *handles.secon_menu* to the value from the drop-down.
5. *Run* calls *extrarunbutton_Callback*, which runs the extraction unless an NN is chosen, which then validates the user input and generates the required files using the first dataset.
6. *Reset* calls *resetbutton_Callback*, which then calls *Reset_Overlay(handles)*.

Segmentation_Watershed_Threshold(handles)

The proposed improved Watershed algorithm adds a number of MM operations in the process of the original algorithm, including imopen, imerode, etc. In addition, the images are also pre-processed to enhance the features for NNs. Finally, the SaMLT (Statistics and Machine Learning Toolbox) is used to analyze the collected data.

NN_Advanced(conf,num,layers,genPlot)

The following parameters have been used in the proposed NN configuration, which includes the training method, transfer function, performance function, etc.

- Training function - *trainscg*: how the network is trained.
- Processing function - *removeconstantrows*: it removes rows if the same value is in every column as it doesn't provide any extra information to the NN. *mapminmax* maps the matrix values in the range -1 to 1.
- Dividing function - *dividerand*: how the input data is split in the network.
- Performance function - *crossentropy*: how the performance of the network is calculated.
- Goal - *0*: how well the NN needs to perform.
- Gradient - *1e-6*: the minimum performance gradient which shows how well the performance needs to stop improving before it's stopped.
- Epochs - *1000*: the number of times the data is shown to the network to be learned.
- Max Fail - *6*: how many times the validation needs to fail before the NN stops training.

Utilizing an FFNN to test the possibility of achieving a high accuracy using a basic ANN is another reason why it is tested in such a way. Why a flattened image and the image features are compared is because a standard CNN uses 2D images and computes the image features whilst training the NN.

Additional functions

Reusable functions are added to prevent code duplication and aid code generation. The following functions are created for resetting and debugging the program.

Reset_Overlay(handles)

Targeting the second image panel, the function removes all objects, including the image, without calling the function. Segmentation and extraction techniques would continue to overlap on the panel. Image zoom is reset, the run button is disabled for *Select extension* because the image required no longer exists and the command line is cleared using *Reset_Command_Window()*.

Reset_Command_Window()

This function is called at the start of each main method to remove any generated command line outputs.

Find_Rectangles(im)

For debugging purposes, it provides an easy comparison of the infected cells found in the labeled images from the second dataset (see Section 4.3) and the extracted image. By taking the unlabeled image, the function matches it with the labeled counterpart using the filename and performs operations to obtain the bounding borders of the infected cells.

Graphical user interface

The Graphical User Interface Development Environment (GUIDE) was used to create the GUI as it features the ability to drag and drop components. GUIDE sets the layout for the UI in an *.fig* file with the buttons and selections located at the bottom and two image panels (left and right respectively) located above. The functionality is coded in the respective automatically generated *.m* file with the same filename. The *.m* file is populated with a set of callback functions which run on button clicks and other user input events.

The program needs a set of requisites to follow in order to be able to aid its users:

- Ability to upload images
- Select a segmentation technique
  - Watershed
  - Watershed & Thresholding
- Select a classification technique
  - Run the area extraction algorithm
  - Generate the images for the neural network
- Before and after image panels
- Resetting the program back to initial startup

Upon launching the program, Figure 4 is displayed to the user without showing any images. When a user uploads an image, it is placed in the left image panel, i.e., *030.jpg*, running a segmentation technique which presents the mid-process and final segmented image(s) in the right panel, i.e., *Segmented Image*.

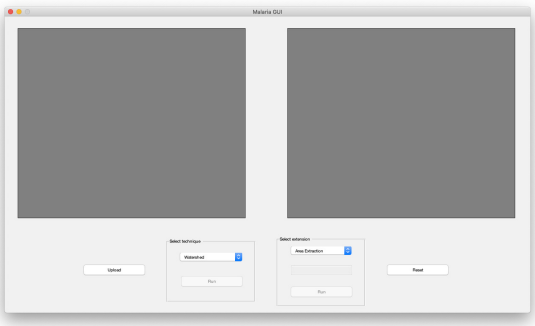

Fig. 4. Program GUI on start-up

Located at the bottom of the GUI is the command panel. It contains the main program functionality with which the user interacts directly. If the user wants to extend the program, the code would need to be edited as this functionality is not in the GUI. Although methods can be run multiple times (on the image originally selected and not the images in current state), the program flows from left to right when using the command panel.

A means to prevent run-time errors operates through logic checking, which prevents the user from the running methods before completing preceding steps. An image is required first before any segmentation method can be run, and an image needs to be segmented before the extraction algorithm can be run. In both cases, the run button is enabled if the requirement is fulfilled. The GUI aids the automated solution by keeping the interface simple and therefore allows anyone to run it.

4. Experimental results

4.1. Database

Two datasets of images are used to evaluate the program. A reliable database is key to testing the programs performance, as it accounts for the different types of images that potentially could be presented. In general, at the top level there are two main image types: A) individual cells; and B) blood smears (a cluster of RBCs, infected and non-infected). The images are stored in separate folders instead of sharing the same folder to make them more identifiable when selecting the blood smears to be segmented. The folders are arranged as follows in Table 2:

Category	Sample Size
Cells (Parasitized)	13,780
Cells (Uninfected)	13,780
Blood Smears (Unlabeled)	30
Blood Smears (Labeled)	30

Table 2. Dataset sample sizes

Single RBC images located in the first dataset were provided by the National Institute of Health (NIH) [28], and blood smears in the second dataset were provided by University College London (UCL) [29]. Similar characteristics such as color can be found in both sets, but not all cells have the same shape though the majority are circular in appearance as shown in Figure 5 and Figure 6. Common features include:

- Color - cells vary slightly between bright pink and pastel purple due to the staining, the uploading method, the camera used to take the image and cell shading.
- Texture - some appear rougher around the edges and in the center.

- Luminosity - the cells vary in brightness as some are darker than others.
- The database contains only ground-truthed images to focus on regions of interest.

4.2. Cells

There are 27,560 images in the first dataset in which there are only single RBCs. They are kept consistent with a resolution of 72 Pixels Per Inch (PPI), and a width-and-height of 121 by 133 pixels, where $121 \times 133 = 16,093$ pixels. Ground truth involves separating the background and the foreground for clearer distinctions. In our case the background is black and foreground is colored to showcase the cell, but this may not always be the case and images would need further processing.

Figure 5 and Figure 6 show uninfected and infected cells respectively. In general, the uninfected cells normally have a clear background, and the infected cells have various shadowed backgrounds, which are related to the Plasmodium species of malaria [3]. One location for the NN images allows for an easier setup and adaptability if the dataset needs to be changed. The sub-folders are set up equally as:

- 13,780 are parasitized cells
- 13,780 are uninfected cells

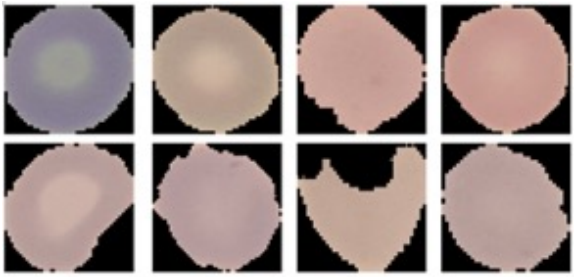

Fig. 5. Uninfected cells

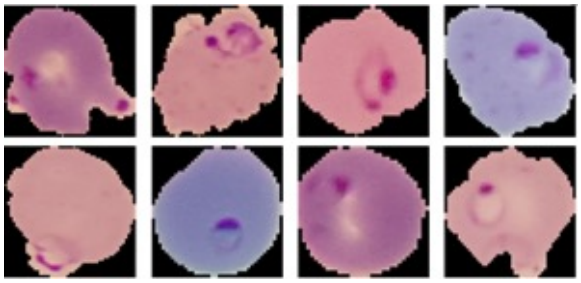

Fig. 6. Infected cells

4.3. Blood smears

The second dataset contains 60 images of blood smears and as before they have been ground-truthed. Each image has a resolution of 150 PPI and a width-and-height of 1300 by 1030 pixels, where $1300 \times 1030 = 1,339,000$ pixels or 1.3 megapixels.

The images cover both classes of cells without overly populating the image but enough for the segmentation to be handled. The folders are organized as:

- 30 are unlabeled (see Figure 7)
- 30 are labeled (see Figure 8)

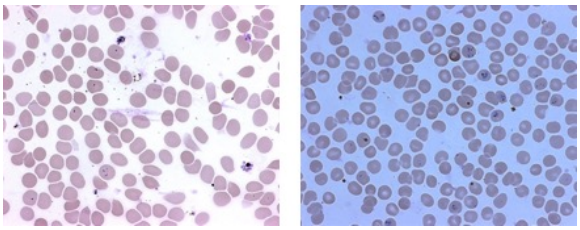

Fig. 7. Unlabeled blood smears

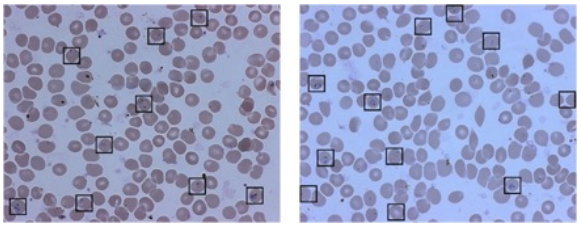

Fig. 8. Labeled blood smears

Figure 7 shows examples of unlabeled blood smears which have been stained to show all the RBCs. Figure 8 shows the same blood smears as in Figure 7, but highlights the infected cells with a black outlined square around them.

4.4. Performance evaluation

4.4.1. Segmentation

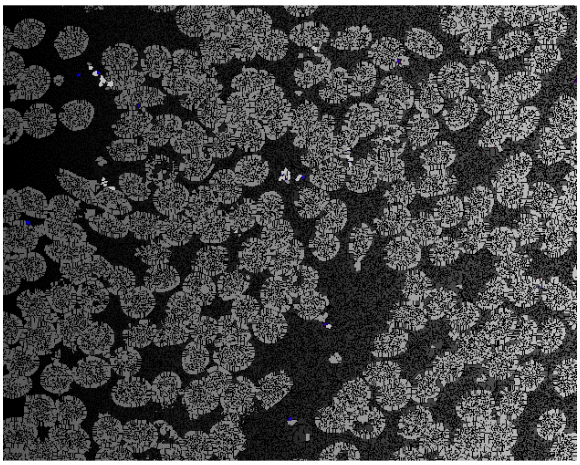

Fig. 9. Result of the Watershed segmentation

Figure 9 shows the output of the original Watershed technique (see Algorithm 1) when it is used without any additional preprocessing of the image. The Watershed technique alone does not perform well as it results in oversegmentation with thousands of tiny regions. Attempting to run the extraction algorithm on this image would prove useless because it would be slow and inaccurate as the algorithm would check each region.

The segmented image is so granular that no new information can be gained. And this is the biggest problem because too many regions are found. In this case, the image cannot be used for further analysis without merging the white separated regions which are the cells. Although the cells can clearly be seen in white, they do not correspond to what has been segmented by the algorithm.

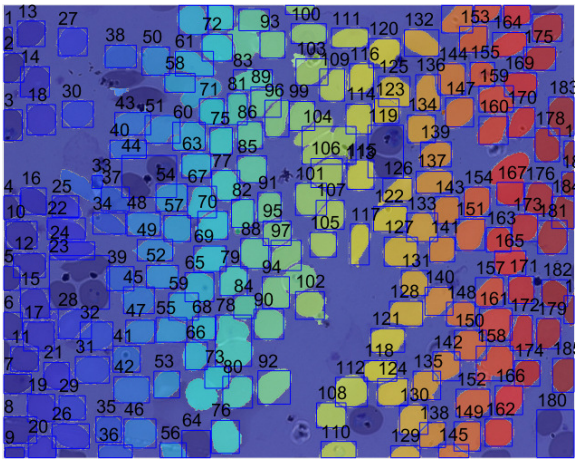

Fig. 10. Result of the Marker-based Watershed segmentation

Figure 10 shows the result of the second algorithm (see Algorithm 2), although it is better when not all cells were detected separately. Most cells were found but the segmentation didn't correctly segment cells close together. Instead, they are treated as a single cell. This was solved by the improved segmentation (see Algorithm 4).

Important cells can be grouped together with non important one. And this can be seen by taking a closer look at the bounding boxes in Figure 10. What's worse with this algorithm is that some cells are completely missed, which could also lead to missing important cells. The extraction works cell-by-cell, so clear distinctions are needed as grouped cells lead to misleading results.

After testing the previous versions of the Watershed technique by comparing Figure 10 and Figure 11 the improved method is tested and displays a more accurate result at the expense of removing cells around the border. The solution combats the problems of oversegmentation and cell grouping by using a combination of thresholding with the Watershed technique. Images used in all three algorithms are Giemsa stained. It is likely that using a different stain would result in different outputs. The output of this algorithm means that cells are ready to be analyzed further without the risks.

Algorithm 4 Improved Watershed Segmentation

Input: *image*
Output: *stats* ▷ Bounding area of the extracted cells

function SEGMENTATION_WATERSHED_MODIFIED(*im*)
  imageRGB ← *im* ▷ Read image into program
  convert image from RGB to grayscale
  compute adaptive threshold
  convert image to binary using threshold
  clear all cells around border
  MATHEMATICAL MORPHOLOGICAL OPERATIONS(*binaryImage*)
  remove additional noise
  compute distance transform
  compute gaussian filter
  segmentedImage ← WATERSHED
  convert matrix of labels to RGB image
  stats ← LABEL2RGB(*segmentedImage*) ▷ Calculate coordinates of cells in image
  show RGB image with labels layered over
  for each bounding region *stats*, *kk* ∈ {1, . . . , HEIGHT(*stats*)} **do**
    draw bounding region as rectangle
    add cell count next to rectangle
  end for
  return *stats*
end function

The improved segmentation performs significantly better by capturing all the cells. Cells around the edge have been removed to prevent and reduce the likelihood of errors due to incomplete cells. Multiple cells in Figure 11 that were removed from the border are cut off. It is possible to capture these, but this could result in incorrect detections as the parasite could be off-screen.

4.4.2. Neural network

Using the correct inputs and methods leads to a better NN performance. The NN was run 5 times. For each run, the parameters were changed to test whether the parameters affected performance. Initially the NN used default parameters with different layer sizes and inputs (see Table 3) and the second set of configurations (config) use the best config from Table 3 with modified parameters (see Table 4).

There are 9 configs in total, where the first 3 use an 1D image reshaped from 2D, 4 - 6 use 4 features, and 7 - 9 use 5 features of the cells. The calculated features are:

- White count - In a binary image, the white area is the region of interest, which makes sense to use because non-infected cells should have a white count of 0.

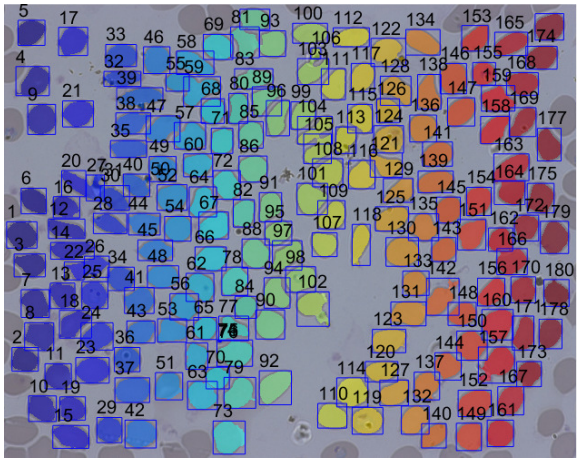

Fig. 11. Result of the improved Watershed segmentation

Configs	Input layers	Hidden layers	Output layer
1	4096	10	2
2	4096	50 20	2
3	4096	5 20 10	2
4	4	10	2
5	4	50 20	2
6	4	5 20 10	2
7	5	10	2
8	5	50 20	2
9	5	5 20 10	2

Table 3. NN configuration with default parameters

- Skewness - It determines the surface of a cell, because it can detect darker areas in images. In this case, it would be the parasites in grey. The image histogram tells how balanced as well as symmetrical the grey levels are.
- Entropy - It determines how much information is in the image so that an image filled with many black areas will have a higher entropy than an image with one color.
- Area - It calculates the area of objects in the image. An infected cell will have an area because the parasite is of a different color. In non-infected cells, the value is next to none because no objects will appear.
- Variance - It shows how each pixel differs from the next. In a cell the parasite is of a darker purple and therefore will show an increased variance around the cell because neighboring pixels will be lighter.

From Table 5 and Figure 12, it can be seen that configs 4-6 performed marginally better than 1 - 3 (a high of 69.9% accuracy) and slightly better than 7 - 9 (high of 91.4% accuracy). Changing the input type from an 1D image to features showed most promising results . Patterns were identified easier when using features as the NN struggled to find a common ground with the 1D image. The FFNN could not handle an input of that size, whereas a complex CNN would be able to form a better understanding. The convolutional layer in a CNN detects the image features rather than having them passed.

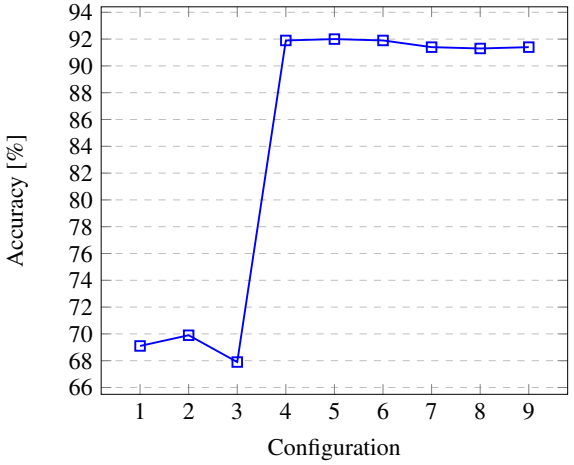

Fig. 12. Default configuration performance visualization

Information is lost when the image is reshaped, which is why configs 1 - 3 didn't perform well because features are harder to define. A filter, also known as kernel, is used in the convolutional layer to prevent it from losing important information when capturing features. This is why this layer is used multiple times when a CNN is used at different stages. The more times the layer is used, the more complex features found.

The following parameters were changed to further verify the system:

- Max Fail - To see whether increasing this value would allow the NN to perform better. It would give the NN more of a chance to improve its performance before failing completely.
- Training function - Whether the program being trained differently would affect the performance to determine the best way to train the NN based on the features.
- Performance function - As the training function changes, the required performance function also needs to change with it.

Comparing the results from Table 5 and Table 6, it can be known that the parameter changes are negligible. Allowing the network to fail later and changing the training functions make little difference between 0.1% and 0.3%, which in most cases is not an enough significant increase in performance to warrant the increase in time taken to perform. It might be beneficial to test the NN with more than 4 features if that has an impact on performance, although the additional feature in Configs 7 - 9 does not change the output.

Configs	Input layers	Hidden layers	Output layers	Additional parameters
4(a)	4	10	2	Max fail = 100; Training = trainbr; Performance = mse;
4(b)	4	10	2	Max fail = 50; Training = trainlm; Performance = sse;
4(c)	4	10	2	Max fail = 30; Training = traingd; Performance = mse;

Table 4. NN configuration with modified parameters

Configuration	Overall Avg Accuracy
1	69.1%
2	69.9%
3	67.9%
4	91.9%
5	92.0%
6	91.9%
7	91.4%
8	91.3%
9	91.4%

Table 5. NN performance with default parameters

Config 4(c) from Table 6 does not perform well compared with the other tests, including the original tests without the updated parameters. By default the training function is *trainscg*, which is an improvement from *traingd*. Both functions are gradient descents where the weights (connections between neurons) are updated. *trainlm* works based on the Levenberg-Marquardt algorithm, which is fast on small datasets but requires a lot of memory for larger datasets. *trainbr* works similarly to *trainlm*, but combines errors with weights. *trainscg* is able to converge quicker, which results in a faster run time.

Configuration	Overall Avg Accuracy
4(a)	92.1%
4(b)	92.2%
4(c)	87.5%

Table 6. NN performance with modified parameters

Comparing the performance of the stat-of-the-art malaria classification systems is very hard, because the systems are evaluated on blood samples from entirely different origins. This means that parameters for image acquisition or slide preparation are completely different. In addition, the sample sizes and conditions of samples are varied from case to case. Currently, no publicly available image benchmark set could be used for fair comparisons of systems [3]. Therefore, we report our performance matrix in terms of accuracy, sensitivity, specificity, precision and F1 score in Table 7.

Measure	Training	Testing
Accuracy	92.5%	91.0%
Sensitivity	98.0%	98.5%
Specificity	87.1%	83.9%
Precision	88.3%	85.4%
F1 Score	92.3%	91.4%

Table 7. NN performance matrix

5. Conclusion

The proposed malaria segmentation and classification system achieves an overall 92% accuracy by updating the parameters in the NN and shows great promise by using an FFNN. The intention of this work is to create a single process from start to finish to allow a user to diagnose a patient from a blood smear. Such a light-weight and cost-effective automated diagnosis system is one of the important steps to enable the IoT/mobile phone based connected healthcare solutions. It is clear from the work undertaken that NNs and segmentation techniques have come a long way, but there are still many more improvements to be made. The limitations of these techniques are that they do not work in all cases and require additional tuning to work as expected. Although this is a problem, it has highlighted the potential for the future. Segmentation and classification could in the future be combined to form a single supervised process to aid malaria diagnosis, take the segmented cells, generate the targets from the labeled images and then pass those to the NN. Moreover, the segmentation does not work well on cells which are not clearly defined, such as faint cells which blend with the background. Additional thresholding and morphological operations could help to separate the foreground from the background at the risk of including irrelevant artifacts. Overall, due to the increase in popularity of NNs and image processing techniques, we move closer to creating a fully automated process which will aid the malaria diagnosis.

From a supplementary aid to the state-of-the-art tool, it will become a gold standard.

Acknowledgments

This work is partly supported by the Fundamental Research Funds for the Central Universities of China under grants GK202003080, by the Natural Science Foundation of Shaanxi Province under Grants 2021JM-205, and the UK Engineering and Physical Sciences Research Council through grants EP/V034111/1.

References

[1] F. E. Cox, History of the discovery of the malaria parasites and their vectors (2010). doi:10.1186/1756-3305-3-5.

[2] Centers for Disease Control, CDC - Malaria (2019).

[3] M. Poostchi, K. Silamut, R. J. Maude, S. Jaeger, G. Thoma, Image analysis and machine learning for detecting malaria, Translational Research 194 (2018) 36 – 55, in-Depth Review: Diagnostic Medical Imaging. doi:https://doi.org/10.1016/j.trsl.2017.12.004.

[4] Organization, World Health, WHO — Malaria (2011).
URL https://www.who.int/ith/diseases/malaria/

[5] NHS, Malaria - Causes - NHS (2018).
URL https://www.nhs.uk/

[6] J. Russ, The Image Processing Handbook, Sixth Edition, Taylor & Francis Group, LLC, 2011. doi:10.1201/b10720.

[7] N. M. Zaitoun, M. J. Aqel, Survey on Image Segmentation Techniques, in: Procedia Computer Science, Vol. 65, 2015, pp. 797–806. doi:10.1016/j.procs.2015.09.027.
URL https://linkinghub.elsevier.com/retrieve/pii/S1877050915028574

[8] J. Chen, H. Shao, C. Hu, Image Segmentation Based on Mathematical Morphological Operator, in: Colorimetry and Image Processing, IntechOpen, 2018, Ch. Image Segm. doi:10.5772/intechopen.72603.

[9] J. Kiprop, What Is a Catchment Area of a River or Lake? - WorldAtlas.com (2018).
URL https://www.worldatlas.com

[10] H. Huang, X. Li, C. Chen, Individual tree crown detection and delineation from very-high-resolution uav images based on bias field and marker-controlled watershed segmentation algorithms, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 11 (7) (2018) 2253–2262.

[11] X. Zhai, F. Bensaali, Improved number plate character segmentation algorithm and its efficient fpga implementation, Journal of Real-Time Image Processing 10 (1) (2015) 91–103.

[12] X. Zhai, F. Bensaali, R. Sotudeh, Real-time optical character recognition on field programmable gate array for automatic number plate recognition system, IET Circuits, Devices & Systems 7 (6) (2013) 337–344.

[13] E. Dougherty, Mathematical morphology in image processing, CRC press, 2018.

[14] H. P. Ng, S. H. Ong, K. W. C. Foong, P. S. Goh, W. L. Nowinski, Medical Image Segmentation Using K-Means Clustering and Improved Watershed Algorithm, IEEE, 2006. doi:10.1109/SSIAI.2006.1633722.

[15] H. T. Nguyen, M. Worring, R. Van Den Boomgaard, Watersnakes: Energy-Driven Watershed Segmentation, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 25 (3). doi:10.1109/TPAMI.2003.1182096.

[16] V. Grau, A. U. J. Mewes, M. Alcañiz, R. Kikinis, S. K. Warfield, Improved Watershed Transform for Medical Image Segmentation Using Prior Information, IEEE TRANSACTIONS ON MEDICAL IMAGING 23 (4) (2004) 447. doi:10.1109/TMI.2004.824224.

[17] A. S. Areeckal, M. Sam, S. S. David, Computerized radiogrammetry of third metacarpal using watershed and active appearance model, in: Proceedings of the IEEE International Conference on Industrial Technology, Vol. 2018-Febru, IEEE, 2018, pp. 1490–1495. doi:10.1109/ICIT.2018.8352401.

[18] S. Yuheng, Y. Hao, Image Segmentation Algorithms Overview (2017).

[19] N. Dhanachandra, Y. J. Chanu, Image Segmentation Method using K-means Clustering Algorithm for Color Image, Vol. 2, EDCAECT, 2015.

[20] N. Kumari, S. Saxena, Review of Brain Tumor Segmentation and Classification, in: Proceedings of the 2018 International Conference on Current Trends towards Converging Technologies, ICCTCT 2018, IEEE, 2018. doi:10.1109/ICCTCT.2018.8551004.

[21] A. Salihah, A. Nasir, H. Jaafar, W. Azani, W. Mustafa, Z. Mohamed, The Cascaded Enhanced k-Means and Fuzzy c-Means Clustering Algorithms for Automated Segmentation of Malaria Parasites, Malaysia Technical Universities Conference on Engineering and Technology (MUCET 2017) 150 (MATEC Web Conf., 150 (2018) 06037). doi:10.1051/matecconf/201815006037.

[22] N. Sharma, M. Mishra, M. Shrivastava, COLOUR IMAGE SEGMENTATION TECHNIQUES AND ISSUES: AN APPROACH, International Journal of Scientific & Technology Research 1 (4).
URL www.ijstr.org

[23] S. D. Rajeshwar Dass, Priyanka, Image Segmentation Techniques, IJECT 3 (1). doi:10.1.1.227.6638.

[24] N. Castle, An Introduction to Machine Learning Algorithms (2017).

[25] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, T. S. Huang, Dilated recurrent neural networks, in: Advances in Neural Information Processing Systems, 2017, pp. 77–87.

[26] M. Tan, Q. V. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, arXiv preprint arXiv:1905.11946.

[27] C. Moler, MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink (2020).
URL https://uk.mathworks.com/

[28] NIAID, Malaria — NIH: National Institute of Allergy and Infectious Diseases (2016).
URL https://www.niaid.nih.gov

[29] UCL, UCL - London's Global University.
URL https://www.ucl.ac.uk/