Capstone Project

Machine Learning Engineer Nanodegree

Musa Mikail

July 2019

Definition

Project Overview

Malaria is a tropical infectious disease caused by the Plasmodium parasite. The World Health Organization (WHO[1]) reported that in 2017 alone, about 53.7 Million cases of malaria were reported in Nigeria with about 79,800 deaths reported as result of the disease. Diagnosis of the disease is often made by observing and counting the number of infected blood cells under the microscope. However, the accuracy of diagnosis depends largely on the judgement of a human observer to classify and count infected and health cells. This become more challenging in northern Nigeria due to scarcity of competent laboratory personnel. Sivaramakrishnan Rajaraman., et al[2] has developed a Convolution Neural Network (CNN) that facilitates malaria parasite detection from microscopic images.

In this project, a binary classifier was trained and deployed to improve efficiency and quality of malaria diagnosis. The classifier was trained using a dataset published by the United States National Library of Medicine[3] (It includes images of segmented cells taken from thin blood smear slides of both infected and uninfected patients.

[1] World Health Organization. World Malaria Report 2014 (World Health Organization, 2014)

[2] Sivaramakrishnan Rajaraman, Stefan Jaeger, Sameer K. Antani. "Performance evaluation of deep neural ensembles toward malaria parasite detection in thin-blood smear images."

[3] Jaeger, Stefan. "Malaria Datasets." U.S National Library of Medicine, U.S Department of Health and Human Services, July. 18, 2019, https://ceb.nlm.nih.gov/repositories/malaria-datasets/

Problem Statement

The goal of this project is to develop a windows PC application (called *i*-mal) that will simplify and improve the effectiveness of malaria diagnosis especially in rural parts of northern Nigeria.

The steps to follow towards achieving this goal are as follows:

1. Retrieve, explore and pre-process a relevant training dataset.
2. Segment and count individual red blood cells using edge detection.
3. Train a binary classifier to count number of infected red blood cells from step 2.
4. Freeze and Deploy the classifier using a pyQt PC application.
5. Analyse images of sample test slides and make inference.
6. Compare performance of the deployed application with manual diagnostic techniques.

Evaluation Metrics

The Accuracy of *i*-mal will be the combined accuracy of the Canny edge detection estimator (for counting the total number of cells), and the accuracy of the cell binary classifier. Because the available malaria dataset is balanced, accuracy of the binary classifier against the validation data will be used as a metric for evaluating its performance.

Given, Ac = Accuracy of cell counter and A_b = Accuracy of the binary cell classifier

Then,

$$A_c = \frac{Cell\ count\ from\ Canny\ edge\ detector}{Cells\ from\ manual\ count} \times 100$$

$$A_b = \frac{Cells\ correctly\ classified}{Size\ of\ Dataset} \times 100$$

Analysis

Data Exploration

The Malaria dataset comprises of 27,558 PNG images of the segmented red blood cells from thin blood smear slide images captured using a microscope. The images are partitioned into two (2) separate folders labeled “parasitized” and “uninfected”, each representing the two possible labels of either been infected or not. The dataset is balanced with each class containing 13,780 images.

For this project, 75% of the images will rabe randomly selected for model training, while 25% will be reserved for validation. Additional annotated images of thin film smear slides from available from the parasitology.com[4] were used after deployment to test the overall effectiveness of the model.

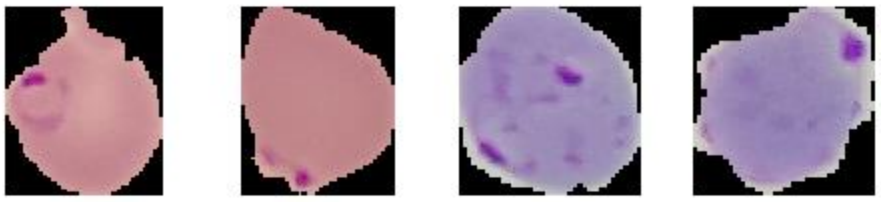


Fig 1a. Images of infected red blood cells. Different coloration due to the type of stain used during smear preparation

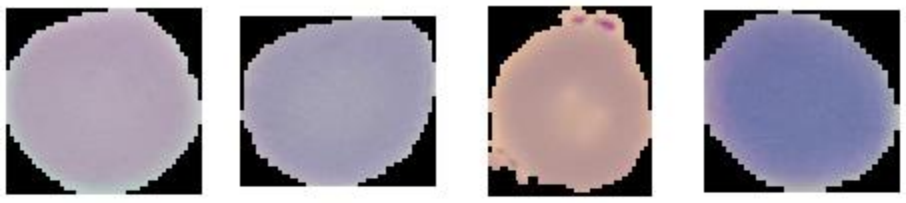


Fig 1b. Images of uninfected red blood cells.

[4] Fadel H. Plasmodium falciparum. PathologyOutlines.com website. http://www.pathologyoutlines.com/topic/parasitologymalariapfalciparum.html. Accessed July 24th, 2019.

Exploratory Visualization

The figure 2 below depicts the distribution of the images in the training set by width and height in pixels.

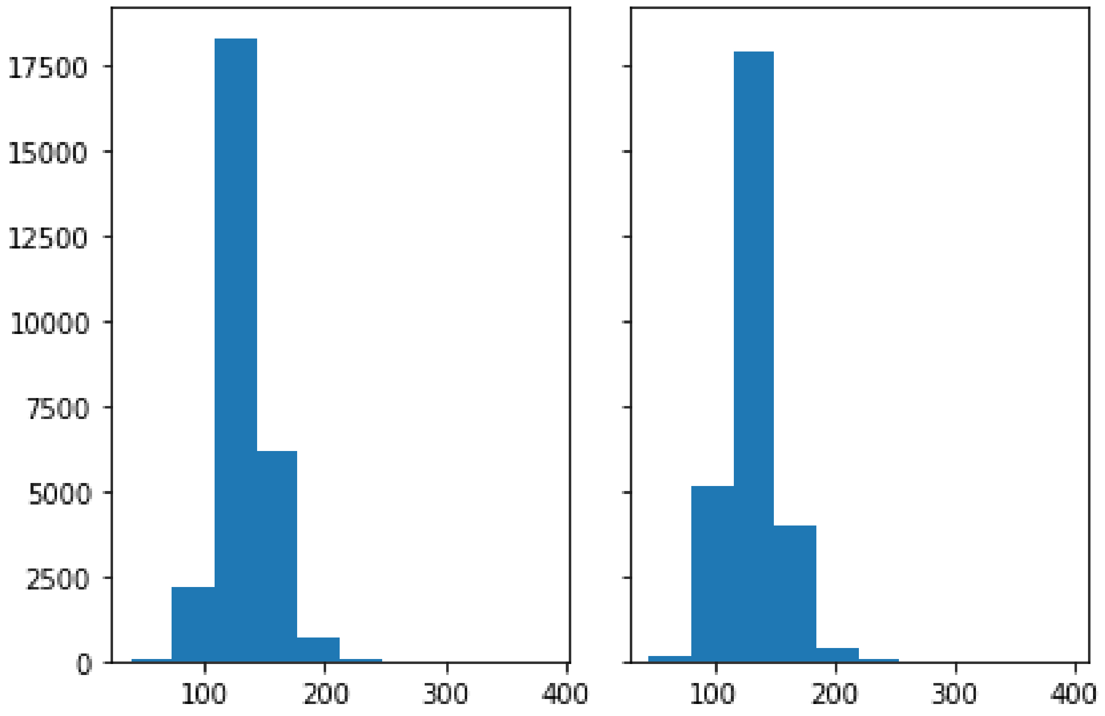


Figure 2. Distribution of the dataset by dimension – (a) height and (b) width

It can be seen that:

- The images in the dataset have sizes ranging from about 50px to about 250px.
- About 80% of the images have dimension distributed around the 130px region
- The modal size is 130X130 px.

This will present a major challenge during model training and prediction since the first stage convolution layer is expecting a fixed shape input for it to function properly (see the data preprocessing section).

Algorithms and Techniques

The i-Mal have two key components that were cascaded to form a complete diagnostic solution.

A. Red blood cell counter

This was implemented using the Canny edge detection algorithm. The purpose of this component is to identify, count and segment red blood cells into separate images ready for prediction. The works of Vijayarani and Vinupriya [5] showed that Canny edge detection algorithm detects edges with higher accuracy when compared with other techniques.

To simplify implementation, the Open CV implementation of the Canny edge detector was utilized. The tuned parameters for this component are:

- minVal and maxVal - representing the pixel minimum and maximum thresholds.
- apeture_size – which is the size of the sobel filter used in finding image gradients.

B. The Cell Classifier

This is a binary image classifier built on a Convolution Neural Network (CNN) architecture trained using 75% of the dataset. This provides the CNN with enough pixel data to be able to build in internal representation of what both infected and healthy red blood cells. During prediction, the CNN takes in the segmented images of red blood cells from the Red blood cell counter and output the probability of it being either an infected or healthy. A cut-off probability value of 0.5 was set to classify cells as either health or infected.

The following are the hyperparameters for the binary classifier:

- Classification cut-off
- Number of training epochs
- Training steps per epoch (training iterations = number of epochs X training per epoch)
- Optimizer -the solver used during training
- Loss function – the function to minimize during training
- Number of layers
- Type of layers – either convolution, max pooling, fully-connected
- Validation steps – the number of images used during a single validation step

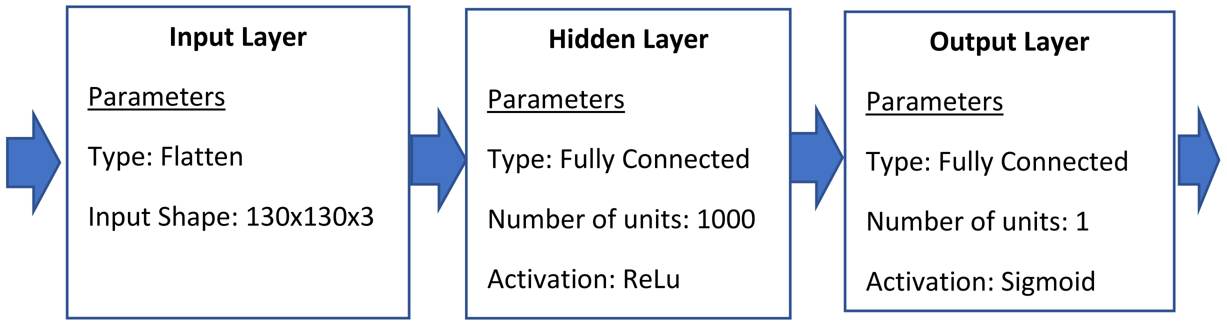
[5] Vijayarani and Vinupriya “Performance Analysis of Canny and Sobel Edge Detection Algorithms in Image Mining”. International Journal of Innovative Research in Computer and Communication Engineering

The Binary classifier was modelled using Tensorflow. Training and validation were done on a GPU in a Jupyter notebook titled "malaria malaria_model" [6]. Inference from the model after deployment is achieved through a Graphical user interface running on the windows PC.

Benchmark

Benchmarking the performance of the model prior to deployment provides insight on the effectiveness of the chosen methodology over others. For this project, a single layer, fully-connected artificial neural network architecture was implemented and trained, hereby referred to as the "Reference model ". The model implementation based on the chosen architecture (hereby referred to as deployment model) was then tuned to provide prediction accuracy higher than reference model.

Figure 3 shows a simplified computational graph of the reference model, depicting the network architecture and the model hyperparameters used for the training job.



The figure showed that the reference model is made up of an input layer, 1 hidden layer and 1 output layer. Based on the implemented network architecture, the reference model has a total of 50, 702, 001 trainable parameters. Reference model was able to achieve training accuracy and validation accuracy of 66% and 65% respectively.

[6] https://github.com/datawiz1984/malaria-model/blob/master/malaria_model.ipynb

Methodology

Data Preprocessing

As highlighted in the Exploratory Visualization section, the images in the datasets are of varying sizes. However, the CNN architecture selected for the deployment model require input of a predetermined size. Additionally, typical red blood cells segmented from thin film slides can have spatial orientation, rotation and sizes varying from those of the images in the dataset. To address these challenges, the following data preprocessing activities become essential.

- The images in the dataset were resampled to a uniform dimension of 130X130 px. This dimension was selected around the mode dimension for anti-aliasing.
- The training dataset was split into training (75%) and validation (25%) datasets.
- The available training data was further augmented by applying the following to each image:
    - Random rotations, with 20º as the rotation threshold.
    - A width and height shift limited to 20% of the actual.
    - Flipping along the horizontal axis.

The preprocessing of the training images was implemented using the ImageDataGenerator module of Tensorflow. The image data was read and sampled in 128 batches for effective memory management.

During inference, additional preprocessing was applied on the tested slide images before sent to the model. This was achieved through the red blood cell counter as indicated in figure 3:

- The slide image was converted into a 1-channel grayscale. This highlights the areas of varying intensities that will aid the edge detection algorithm.
- The Open CV Canny edge detection algorithm was applied to the image to identify the edges of the cells.
- The cells were extracted into individual images that were passed to the deployed model for inference.
- Individual cell images are resampled at 130X130px to ensure compatibility with the CNN models input requirements.

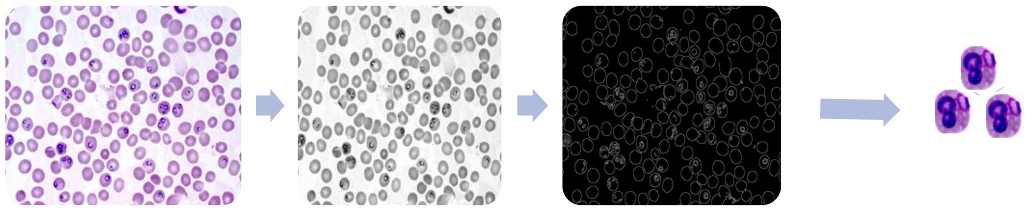


Figure 3: The transition of a slide image in the Red blood counter from original, to grayscale, to cell edges to segmented cells. The performance of the Canny edge detector improved significantly with the conversion of the image to grayscale.

Implementation

The system implementation can be broadly described under two key sub-headings; namely

1. Implementation of Binary classifier
2. Deployment of Binary Classifier

The classifier was modelled as a CNN in a Jupyter notebook titled "malaria model" and can be accessed via the link here[7]. The Binary classifier was implemented by following the following steps:

1. The Malaria Dataset was downloaded and unzipped in the google drive.
2. The data was explored, visualised and pre-processed as outlined earlier in this report.
3. The selected CNN Architecture was implemented using the Sequential method in Tensorflow.
4. The training parameters including number of units, activation functions, loss functions and optimizers were selected.
5. The network was trained, and the values of training and validation accuracies were logged.
6. Analyse the accuracy and if not adequate, adjust model parameters and re-train.
7. Save the trained model in hdF5 format for deployment.

Figure 4 shows the architecture of the CNN used in building the binary classifier. It's a four-stage CNN simplified variant of the VGGNet[8] with only 1 convolution and maxpooling layers in the first three stages, while the fourth layer is composed of fully connected layers.

[7] https://github.com/datawiz1984/malaria-model/blob/master/malaria_model.ipynb

[8] Simonyan∗ & Andrew Zisserman "Very Deep Convolution Networks for Large Scale Image Recognition.", ICLR. 2015

The training parameters selected for the network are as follows:

- Loss function: Binary crossentropy
- Optimizer: Adaptive moment estimation (Adam)
- Metrics: Accuracy – this is a deem appropriate for performance measurement since the training dataset is balanced.

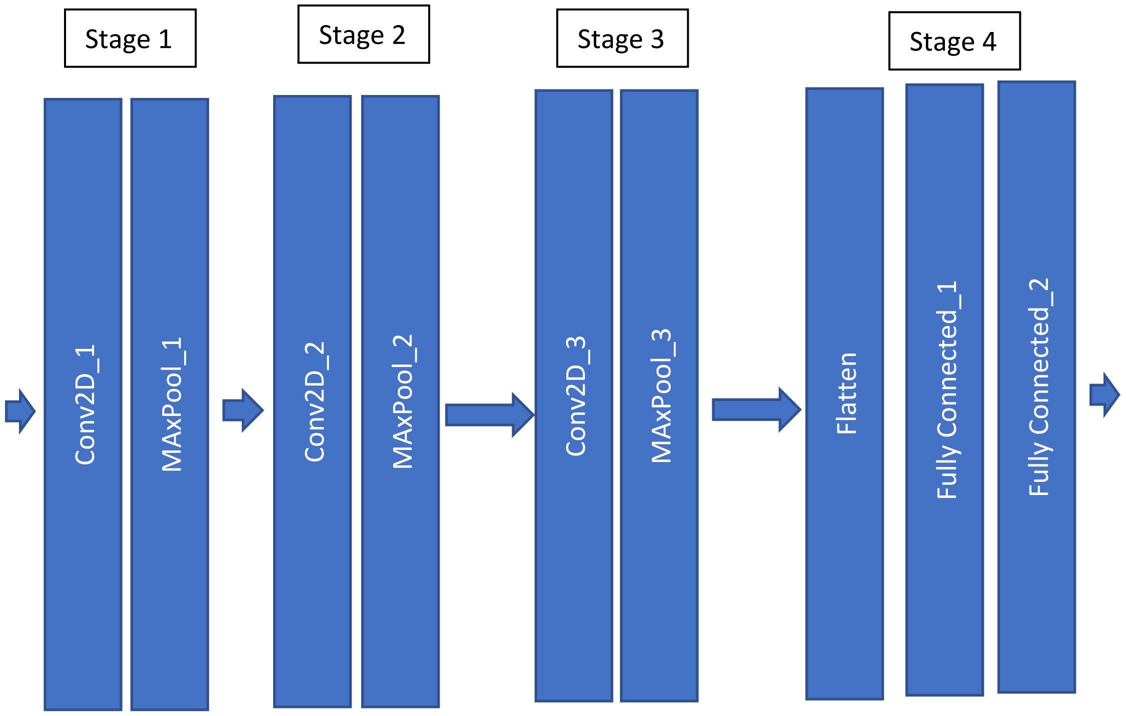


Figure 4. The network architecture of the Binary Classifier.

After training and validation, the binary classifier was deployed. To achive this, the following approach was taken:

- The trained model was saved in a hdf5 format and exported
- A PyQt GUI was created and linked to the python file containing the red blood counter script and the imported model
- The Pyqt application was converted to windows executable.

Refinement

The reference model initially achieved around 60% training accuracy against the training dataset after about 9,500 iterations. Further attempt was made to improve the accuracy of the reference model including increasing the number of units in the connected layer from 1000 to 5000, changing the optimizer from stochastic gradient descent to Adam. However, the accuracy of the reference model was only improved to 66% after 9,500 iteration.

The deployed model however, showed impressive performance of around 85% accuracy on first run. Further refinement we made to achieve higher accuracy as follows:

- Changing the optimizer from stochastic gradient descent to Adam optimizer. This significantly improves the speed of convergence.
- Changing the number of kernels in the convolution layers also improves rate of convergence. It was observed that the increasing the number of convolutions as we move deeper into the network increases the speed of convergence.
- Attempts were made to increase the number of units in the fully connected layers. However, a proportional improvement in accuracy was not observed.

After following the procedure highlighted above, the deployment model was trained over 9,500 iterations to achieve training and validation accuracy of about 95% and 96% respectively.

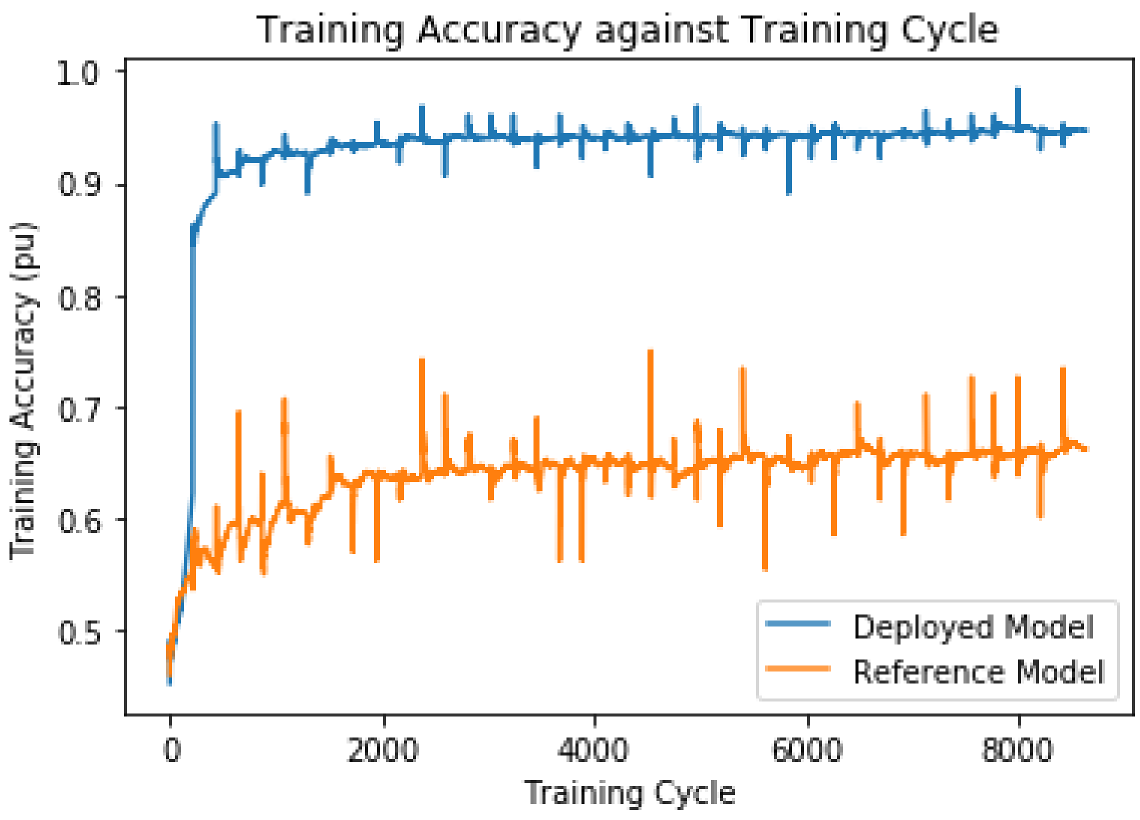
Model Evaluation and Validation

The trained network was validated using 25% of the training data reserved for this purpose. The final network was re-trained based on the optimal network parameters and architecture before it was frozen and exported for deployment.

Table 1 summarizes the various components of the final network with their respective parameters. The classifier has 1,629,473 trainable parameters.

Component	**Identifier**	**Parameters**
Stage 1	Convolution Layer_1	Convolutions: 16 Input Shape: 130X130X3 Kernel Size: 3X3 Activation: Rectified Linear Unit (ReLU) Stride: 1X1
	Max Pooling Layer_1	Kernel Size: 2X2
Stage 2	Convolution Layer_2	Convolutions: 32 Kernel Size: 3X3 Activation: Rectified Linear Unit (ReLU) Stride: 1X1
	Max Pooling Layer_2	Kernel Size: 2X2
Stage 3	Convolution Layer_3	Convolutions: 64 Kernel Size: 3X3 Activation: Rectified Linear Unit (ReLU) Stride: 1X1
	Max Pooling Layer_3	Kernel Size: 2X2
Stage 4	Fully Connected Layer_1	Layer Type: Flatten Output Size: 12, 544
	Fully Connected Layer_2	Layer Type: Fully connected Number of units: 128 Activation: Rectified Linear Unit (ReLU) Output features: 1,605,760
	Fully Connected Layer_3	Layer Type: Fully connected Number of units: 1 Activation: Sigmoid Output features: 1,605,760

Figure 5 shows the performance of the deployment model against the reference model.



The deployed application was able to count the number of segmented cells and pass them to the model to get the estimated % parasitemia. The deployed binary classifier can count the total number of cells in an uploaded slide image and capable of calculating the percentage parasitemia based on the inference from segmented images.

Justification

To verify the accuracy of the deployed model, PyQt application was used to test its effectiveness. The Application calculated the number of segmented cells from the sample slide with a 92% accuracy benchmarked with manual counting techniques.

- The binary classifier can classify the segmented cells with about 89% accuracy benchmarked with manual counting techniques.
- Few false positives and false negatives were observed from the inferences.

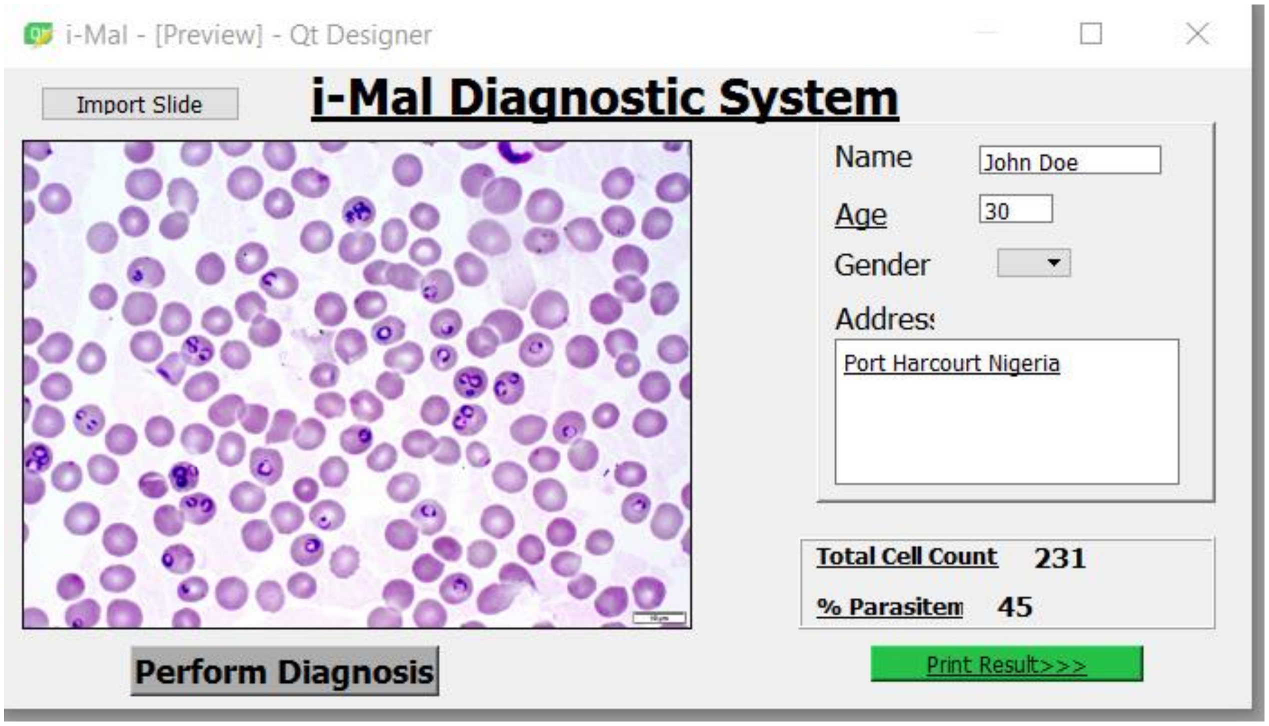
This evidently showed that the application is effectively segmenting and predicting the status of each individual red blood cell. However, for the application to be fit for real-world deployment, additional refinement is required.

- The effectiveness of the red blood cell counter can be improved by improving the edge detection thresholds and exploring means of reducing false counts due to artefacts from dye stains.
- Additional sample slides will be required to further validate and evaluate the effectiveness of the application.
- Further engage with professional parasitologist to gain additional insight on other means through which the quality of the test slides could be made for suitable for the red cell counter to effectively segment the cells.

Conclusion

Free-Form Visualization

Figure 5 shows the i-Mal deployed application performing inference on a sample slide. The application output 231 as the number of cells on the slide while the percentage parasitemia was calculated as 45%.



Name John Doe
Age 30
Gender
Addres: Port Harcourt Nigeria
Total Cell Count 231
% Parasitem 45

Table 2 depicts the result of benchmarking the i-Mal system against the manual counting technique popularly used in calculating percentage parasitemia.

Observation	**i-Mal Result**	**Manual Method**	**%variance**
Total cell count	231	211	9.4%
Total infected cells	103	89	16%
% parasitemia	45	42	7%
Time to diagnosis	< 2 secs	8	240%

Based on the observation, the following conclusions were arrived at:

- The Red blood cell counter has achieved more than 90% accuracy in segmenting and counting the number of red blood cells.
- There were about 14 false positive cases predicted by the model.
- The application was able to achieve diagnosis with 7% variance compared to manual counting technique.
- Speed of diagnosis by i-Mal is about 240% faster than the manual counting technique.

Reflection

A structured methodology has been employed to train and deploy the binary classifier to support malaria diagnosis. The following steps summarized this methodology:

1. The malaria dataset from the United States National Library of Medicine was download and split into training and validation datasets.
2. The training data was explored and pre-processed to prepare it for use by binary classifier.
3. A reference model based on a simple ANN was implemented and tuned to enable benchmarking of the deployment model.
4. The deployment model based on a Convolution Neural Network (CNN) was implemented, trained and tuned based on some key hyperparameters.
5. The trained binary classifier was deployed through a windows GUI application

6. Additionally, a red blood cells counter script was written to enable the segmentation and counting of red blood cells counter during inference making.

The task of implementing the application was quite challenging. The step that presented the most challenge was step 5 that require that I spend time to research how the canny edge detection algorithm works in Open CV. It was however a worthwhile experience because of the experience I got while doing that.

Additionally, I am delighted to have access to the malaria training dataset that enable me make impact in the way malaria is been diagnosed in Nigeria. I will explore further research and collaboration opportunities to deepen my research in this field.

Improvement

To further improve the effectiveness and access to the system in supporting real-life malaria diagnosis, the following additional actions will be taken:

- Explore means of exporting the classifier model as an android application. This will improve access to the diagnosis tool by harnessing the fast penetration of mobile technology in rural.
- Secure additional sample slides to further evaluate the effectiveness of the red blood counter and the binary classifier, and where required perform additional tuning on the model.