**CHAPTER FOUR**

**DESIGN AND IMPLEMENTATION OF THE NEW SYSTEM**

**4.1 The New System Design**

A number of new methods have been developed in recent years for the diagnosis of malaria. These include the use of fluorescent microscopy, rapid antigen detection methods and polymerase chain reaction (PCR) - based techniques that detect specific nucleic acid sequences. Despite these advances, malaria diagnosis by means of manual microscopy remains the most widely and commonly used method. Usually, these jobs are conducted by experienced pathologists manually. Microscopic diagnosis entails examining thick and thin blood smears for the presence of Plasmodia. Unfortunately, there are also disadvantages to the method: substantial costs are incurred purchasing and maintaining microscopes and training technicians, the technique is labor intensive and time-consuming and the accuracy of the final diagnosis ultimately depends on the skill and experience of the technician and the time spent studying each slide. Variable smear quality and slide degeneration with time are also problematic. Besides, a recent study on the field shows the agreement rates among the clinical experts for the diagnosis are surprisingly low.

In this research, we proposed an automated system based on supervised machine learning to detect malaria plasmodium which is able to eliminate the most important limits of microscopic method, that is:

1. Time-consuming and tiring job
2. Low accuracy even in experts.

Here in this research we mainly focus on the task of determining the presence of the parasites and highlighting them for ease of identification, because it is the most essential and time-consuming step in the diagnosis of malaria. Also, we propose a motorized microscope which is fully matched with image processing procedure to make the whole diagnosis process automatic. Furthermore, it means that the physician just puts the blood smear under the lens of microscope and runs the system; after a few minutes the report, which includes the number of RBCs and parasites, is issued.

**4.2 Flowchart of the New System**

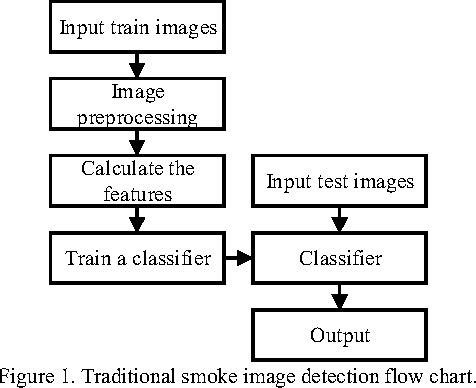
****

Fig 4.1 Flowchart of the new system

*Source: My computer/Photoshop Express*

An automated diagnostic method can be developed by understanding the diagnostic process and representing it by a specifically tailored image processing based algorithm. The image processing based algorithm should perform diagnosis more or less imitating the manual microscopy. The algorithm should be capable of operating in an unsupervised environment and needs to be robust with minimal false negatives (leading to high sensitivity). The unsupervised nature of the proposed procedure should reduce human intervention, and in so doing speed up the diagnosis process. The algorithm should also be sensitive enough to capture parasites at all stages particular at the early stages of their life cycle and do this without missing any parasites irrespective of image variations. In order to perform diagnosis, the method must be capable of differentiating between parasite and artefacts.

**4.3 Requirements for the New System**

The first requirement in this research is blood smears which are Giemsa stained microscopic slides prepared from blood samples that allow microscopically examinations of blood cells. Thin blood smears allow better species identification, because the appearance of the parasites is better preserved in this preparation. Thick blood smears allow screening of a larger volume of blood, therefore, they can give more than a ten-fold increase in sensitivity over thin films. In this research, morphological properties are important for us, hence we used thin films. For malaria diagnosis, blood films should be prepared as soon as possible after blood samples are taken. Such films adhere better to the slides; leave a clearer background after drying, thus, parasite and red cell changes are minimal. After preparing blood films they should be examined by microscope.

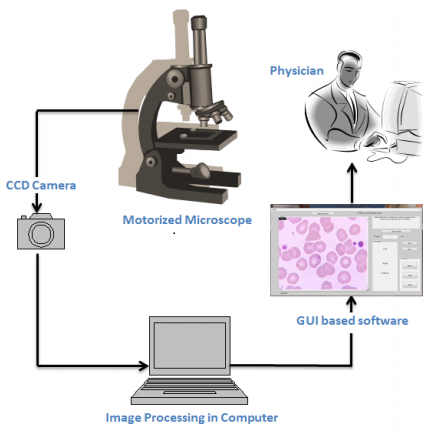


Fig 4.2 Overall scheme of proposed Malaria diagnosis system

*Source: https://www.vinodsblog.com*

A microscope is equipped with two stepper motors which move the blood samples under lens quite smoothly. The amount of movement in each direction is calculated by a microcontroller installed on the microscope board to avoid taking overlapped photos. This also helps avoid calculating each RBC more than once. The photos taken by CCD are transmitted to the image processing program running on a computer. RBCs and infection are detected simultaneously and the final report is issued.

****

Fig 4.4 Thin blood smear slide and acquired digital microscope image at 40X magnification.

*Source: www.semanticscholars.com*

Fig 4.3 Camera attached to Microscope for acquiring images of blood samples

*Source: Faith Mediplex Hospital Laboratory*

The images recorded with the magnification of 1000X are indicates parasite clearly. The Olympus DP25 digital camera of 5 MP attached to the light microscope Olympus BX51, which is connected with the computer, along with the user interface software (DP2 BSW) are shown in Figure 4.3. The acquired the blood image from the focused slide area are collected. The typical malarial thin blood smear image acquired at 40X magnification is shown in the Figure 4.4. Blood images acquired with the various magnifications such as 100X, 200X, 400X and 1000X are shown respectively in the Figures 4.5 A-D. A total of 1,160 images were considered for classification of malarial and non-malarial classes. The acquired thin blood smear image has red blood corpuscles (RBC), malarial parasites, Platelets and other objects. But the proposed technique focus on diagnosis of malaria is based on examination of RBCs, since the malarial parasite infects the RBC.

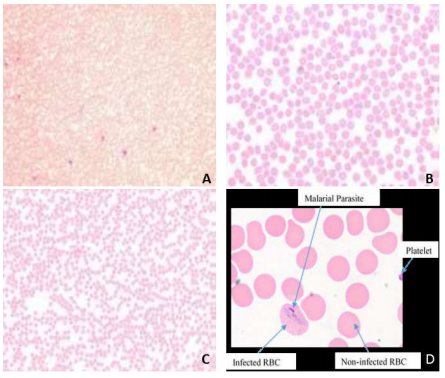


Fig 4.5 The magnified microscopic blood images of the blood sample: (a) at 100X; (b) 200X; (c) 400X; (d) 1000X

*Source: https://www..machinecurve.com/malaria-diagnosis-and-computer-vision*

**4.4 Choice of Programming Language**

The entire project, including all functions, segmentation, image database files, and feature evaluation scripts, have been implemented using MATLAB programming environment version 2019. Although we cannot guarantee that all functions will work properly with any older version of MATLAB, most of the functions, and especially the main script, are simplified where necessary to ensure backward compatibility. MATLAB is a high-performance language for technical computing, which integrates computation, visualization, and programming in an easy-to-use environment. It is an interactive system using an array that does not require dimensioning as the basic data element. Typical uses include machine learning, math and computation, algorithm development, computer vision, data acquisition, analysis and visualization, modeling and simulation, scientific and engineering graphics, and application development including graphical user interface building. It removes the need of programming many routine tasks for numerical computing and allows easy and quick displaying of results both in numerical form as well as in the form of 2D or 3D graphs. The open-architecture of MATLAB allows programmers to incorporate their own area specific set of functions implemented in separate m-files into MATLAB, so that they can be easily used by other functions and scripts written in MATLAB. The following toolboxes are used and required in order to run all functions and scripts in this project:

* Image Processing Toolbox
* Deep Network Designer Toolbox

The Image Processing toolbox is essential as it is used by most of the functions and scripts and it is the only toolbox required for running the segmentation method and the GUI. There are many reasons why it is easier to learn and use MATLAB than other general purpose languages and here are a few:

1. For one, basic MATLAB usage abstracts away almost all of the tricky/annoying/fun details commonly associated with general purpose programming. No memory management, no complicated integration of external libraries, no need to learn about type systems, very simple syntax, etc. Structure is as simple as can be. Calling functions from other files basic and simple.
2. For two, MATLAB is self-contained. It is the language, the runtime, the REPL, the platform, the IDE, and the debugger. Once MATLAB is installed, all you need to do is launch it and everything you need is right there, ready to go. Additional libraries, or “toolboxes”, are extensive (if not expensive) and easy to work with. It also has very detailed help documents for each function.

The default data type in MATLAB is a matrix, so there are umpteen different ways to slice and dice them. This is very handy for computer vision and data processing, which are two very common uses for MATLAB. The same tasks are not always very easy in other languages.

**4.5 The Training Data**

Images of Giemsa stained malaria-infected blood smears were obtained from the Faith Mediplex Hospital Laboratory and they have the following common characteristics:

* Images are available in different magnifications and sizes. The images are available in JPEG format with the resolution of 2 to 3 megapixels.
* Digital images are obtained by scanning and, therefore, contain a part of the noise and artifact from the sample and from the microscope light also noise from the chemical development process or from the scanner.
* Images exhibit high variability in color tone, intensity, contrast, and illumination. The overall color tone varies significantly from grayish, blue, purple, and pink to yellowish and it may even change from the center of the image to its borders. Some images have very low contrast while some images exhibit high contrast between infected and non-infected cells. Many images suffer from irregular illumination.
* The overall shape and appearance of the cells may also vary substantially among the slides. Some cells lack their clear central parts and, in some images, cells may assume shapes that differ from the usual circular shape. Moreover, red blood cells are often overlapping and may form big clusters. Occasionally, blurring and various artifacts may also appear.

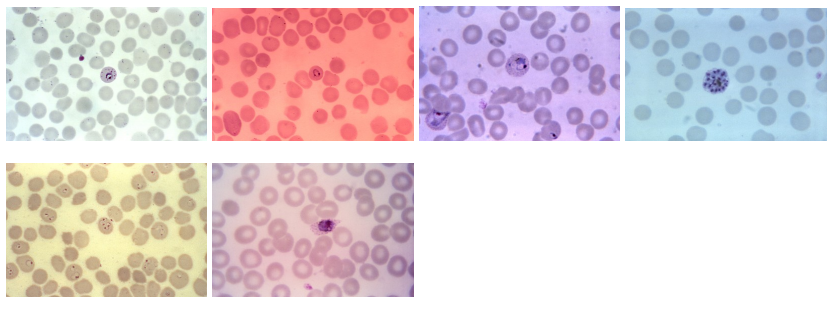
****

Fig 4.6 Samples of available stained blood smear images showing differences in color tone and illumination.

*Source: Faith Mediplex Hospital Laboratory*

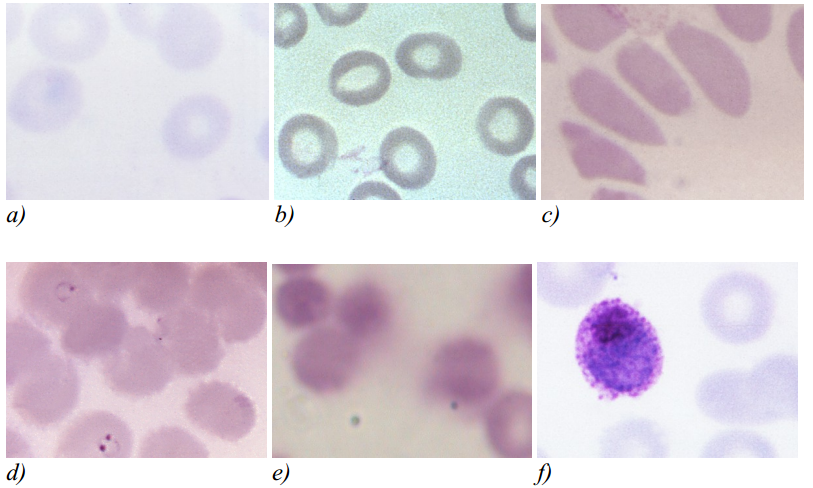
****

Fig 4.7 Cropped samples of available blood smear images showing different qualitative characteristics of the input images.

*Source: Faith Mediplex Hospital Laboratory*

The selection of the features for the further evaluation was based on the visual differences between infected and non-infected red blood cells, the measures of infected red blood cells that are commonly used by technicians for manual microscopic diagnosis, and the feature selection used by other cytological studies. The chosen features can be grouped into three categories: shape features, intensity features, and texture features.

**4.5.1 Shape Features**

These features express the overall size and shape of the cell without taking the density of the cell into account. Although the applicability of the features based only on the shape of the cell is necessarily limited, they can be useful in distinguishing development stages of certain species of plasmodium which are characterized by a specific shape of the infected cell and they may also be useful in distinguishing between red blood cells and other objects, such as white blood cells, platelets of artifacts.

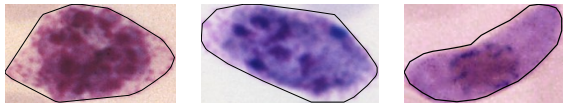


Fig 4.8 Infected red blood cells with distinct shapes

*Source: MATLAB*

Although the number of classes had to be reduced to only two classes – an infected and a non-infected cell – due to the insufficient number of samples, several of these features were evaluated both as guidance for possible future studies, which would include also classification distinguishing between infected and non-infected red blood cells and other objects and classification of parasites according to the development stage and species of the plasmodium, and also to evaluate whether these features could possibly improve the overall discrimination power when combined with other features.

**4.5.2 Intensity Features**

Intensity and color are the most palpable visual differences between red blood cells and parasites. This difference is a result of the staining procedure during which all the object containing DNA, and thus also the plasmodium parasites, are stained in saturated purple color. The intensity features may be advantageous especially when the texture of such a cell is indistinct.

**4.5.3 Textural Features**

Depending on the development stage and the species of the parasite with which a red blood cell is infected, different types of texture can be observed. Parasites of early development stages form rings with distinct speckles of chromatin. Since the rest of the area of the red blood cell is usually quite intact, the texture is given mainly by the chromatin speckles and possibly by the ring lines which are not always visible (Fig 4.9).

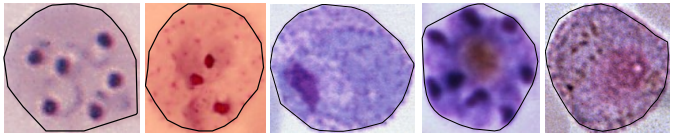


Fig 4.9 Different textural properties of infected red blood cells

*Source: MATLAB*

More distinct texture can be observed in red blood cells infected by parasites in later stages of development. Pigment granules appear early in the growth phase of the parasite as the immature ring-shaped trophozoite becomes mature trophozoite. Depending on the species, the texture in this phase consists mainly of large amoeboid cytoplasm with large chromatin and fine pigment dots called Schüfner's dots.

**4.6 The Convolutional Neural Network**

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes (Junhao 2016). They are algorithms that can identify many aspects of visual data. These make the networks more efficient to implement and vastly reduce the amount of parameters to tune which is why it is ideal for this project. Most recent well performing ConvNets are mainly built from three types of (hidden) layers, convolution layers, pooling layers and fully connected layers although they employ other layers to improve overall accuracy. Convolutional neural networks employ deep learning. Deep learning can be seen as an extension of the well-known multilayer neural network classifiers trained with back-propagation, except that many more layers are used. There are also different kind of layers that are used in typical successions. Deep learning typically requires large training sets. This is the reason why medical applications have been among the last applications to adopt deep learning, as annotated training images are significantly harder to obtain because of expert knowledge requirements and privacy concerns. In this project, we intend to use a convolutional neural network to detect the presence of malaria parasites by differentiating between infected and uninfected cells in thin blood smears.

**4.7 Architecture of the Convolutional Neural Network**

ConvNet is not just a deep neural network that has many hidden layers. It is a deep network that imitates how the visual cortex of the brain processes and recognizes images. That is how much ConvNet differs in concept and operation from the other neural networks. This section briefly introduces the fundamental architecture of ConvNet. ConvNet includes the feature extractor in the training process rather than designing it manually.

The feature extractor of ConvNet is composed of special kinds of neural networks, of which the weights are determined via the training process. The fact that ConvNet turned the manual feature extraction design into the automated process is its primary feature and advantage. ConvNet yields better image recognition when its feature extraction neural network is deeper (contains more layers), at the cost of difficulties in the training process. It consists of a neural network that extracts features of the input image and another neural network that classifies the feature image.

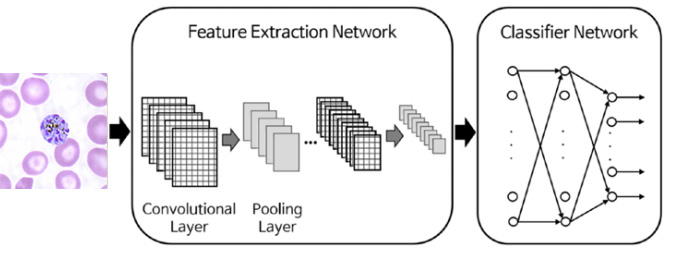


Fig 4.10 Architecture of the convolutional neural network

*Source: My computer/Photoshop Express*

The feature extraction neural network consists of piles of the convolutional layer and pooling layer pairs. The convolution layer, as its name implies, converts the image using the convolution operation. It can be thought of as a collection of digital filters. The pooling layer combines the neighboring pixels into a single pixel. Therefore, the pooling layer reduces the dimension of the image. As the primary concern of ConvNet is the image; the operations of the convolution and pooling layers are conceptually in a two-dimensional plane. This is one of the differences between ConvNet and other neural networks. In summary, ConvNet consists of the serial connection of the feature extraction network and the classification network. Through the training process, the weights of both layers are determined. The feature extraction layer has piled pairs of the convolution and pooling layers. The convolution layer converts the images via the convolution operation, and the pooling layer reduces the dimension of the image. The classification network usually employs the ordinary multiclass classification neural network.

**4.7.1 Image Input Layer**

The input layer is the first layer of our convolutional neural network. The Input layer in the CNN should contain image data. Image data is represented by three dimensional matrices as we saw earlier. You need to reshape all the images it into a single dimension. Suppose you have some images of dimension 150 x 150 and other images of varying dimensions like 200 x 250, 300 x 300, 150 x 200, etc., we need to convert them all into a single uniform dimension before feeding them into the input layer. For this project, I have chosen the dimension of 200 x 200 for all the input images and resized them accordingly. We then specify the color bit after the image dimension where black and white images = 2 bits (B-W) and colored images = 3 bits (R-G-B).

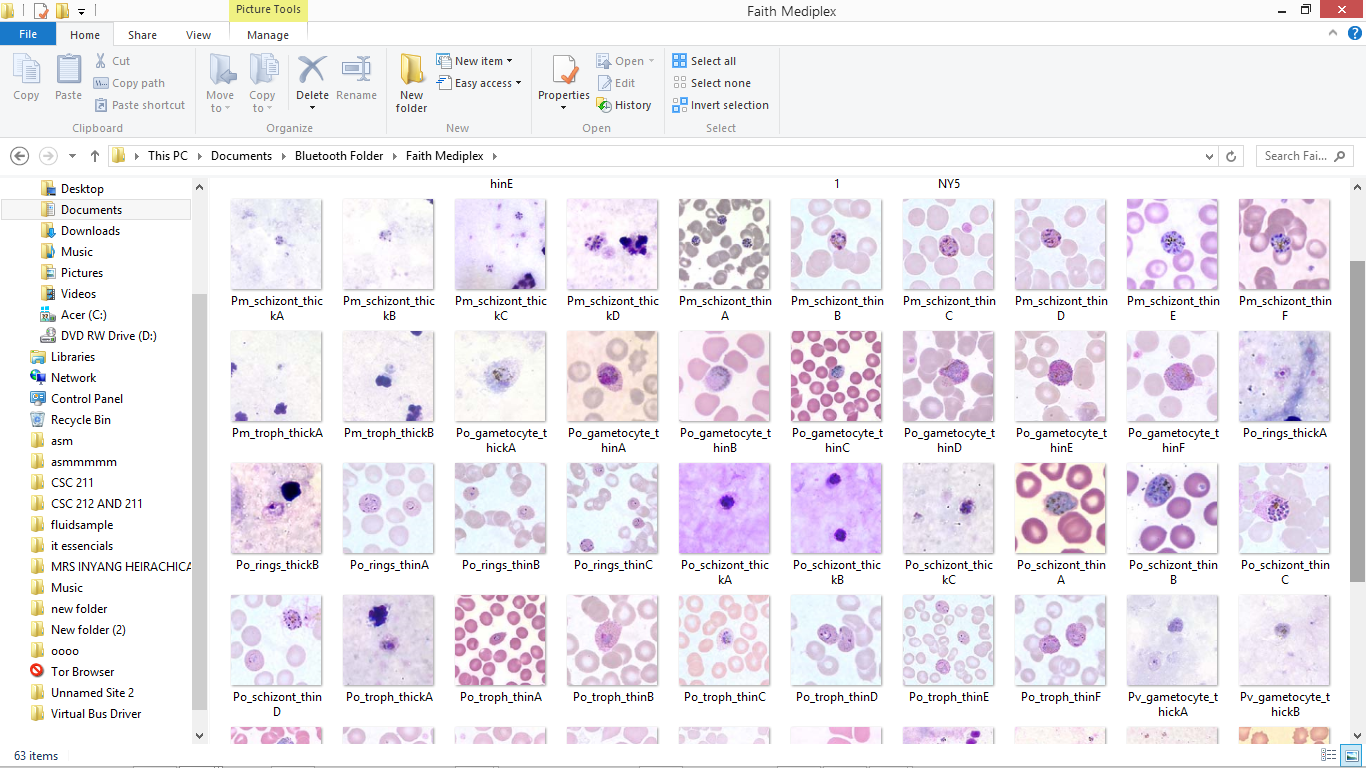


Fig 4.11 Snapshot of some of our training images for the infected class

*Source: My computer*

One of the distinct characteristics of the input layer is that artificial neurons in the input layer have a different role to play – the input layer being constituted of “passive” neurons that do not take in information from previous layers because they are the very first layer of the network.

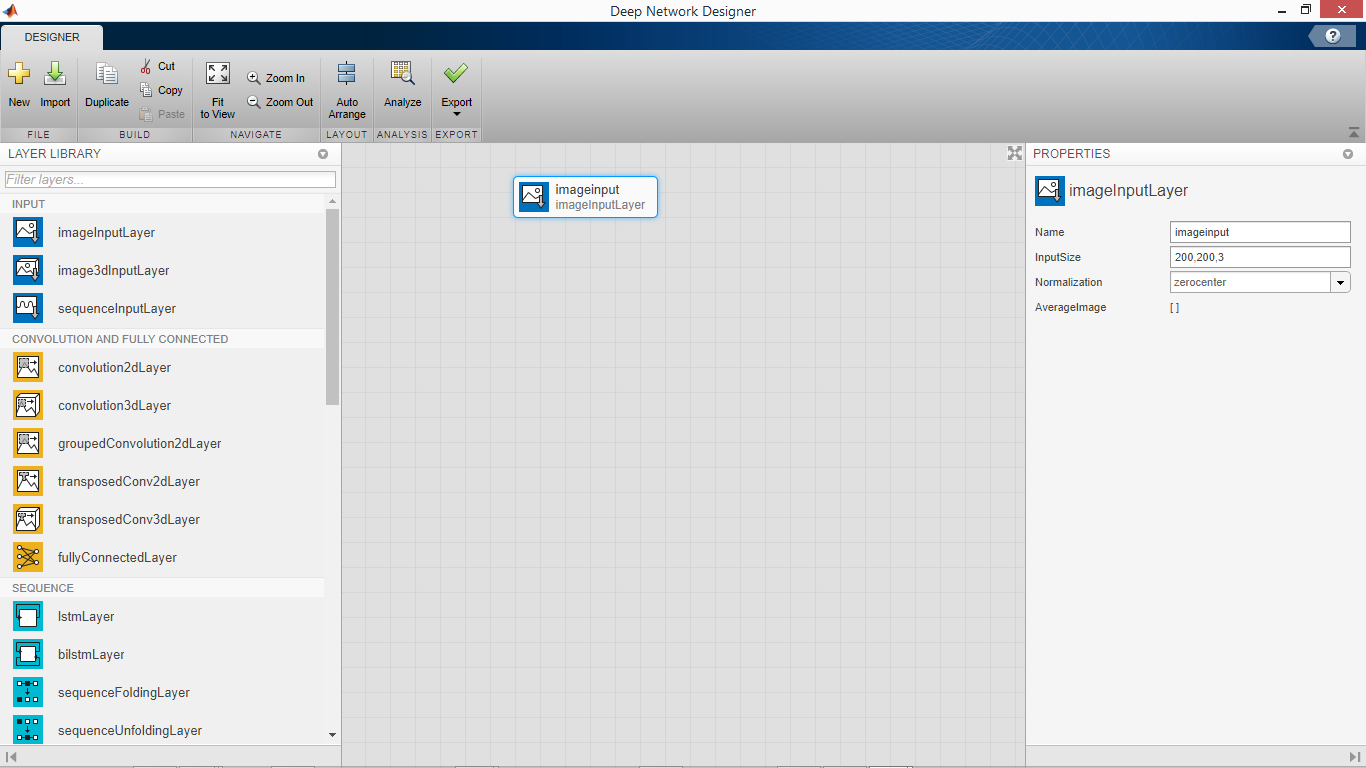


Fig 4.12 The Input Layer

*Source: MATLAB*

**4.7.2 Convolution Layers**

This section explains how the convolution layer, which is one side of the feature extraction neural network, works. The convolution layer generates new images called feature maps. The feature map accentuates the unique features of the original image. The convolution layer operates in a very different way compared to the other neural network layers. This layer does not employ connection weights and a weighted sum instead, it contains filters that convert images. We will call these filters convolution filters. The process of inputting the image through the convolution filters yields the feature map.

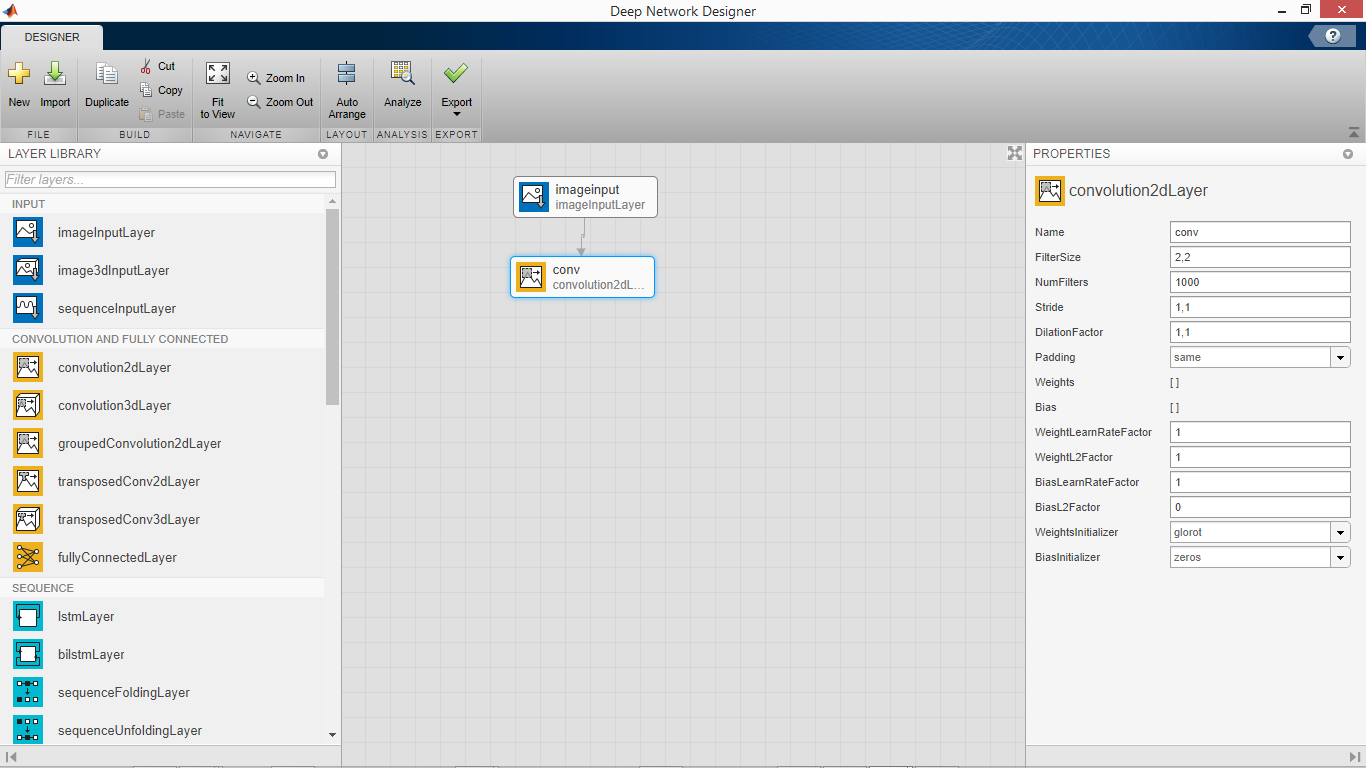


Fig 4.13 The Convolutional Layer

*Source: MATLAB*

We then specify the number of filters for our network which is the number of input images (The training data). We have 1000 images each for the infected samples and uninfected samples which we will feed into the network. We leave the other parameters in the convolution layer as default.

(Fig 6.4) below shows the process of the convolution layer, where the circled \* mark denotes the convolution operation, and the φ mark is the activation function. The square grayscale icons between these operators indicate the convolution filters. The convolution layer generates the same number of feature maps as the convolution filters. Therefore, for instance, if the convolution layer contains four filters, it will generate four feature maps.

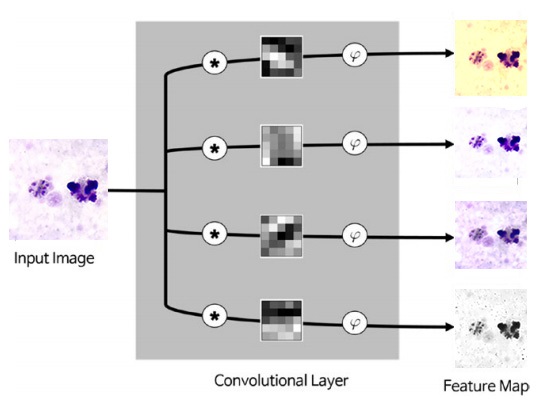


Fig 4.14 In-depth illustration of the convolutional layer

*Source: My computer/Photoshop Express*

**4.7.3 Rectified Linear Unit (ReLU Function)**

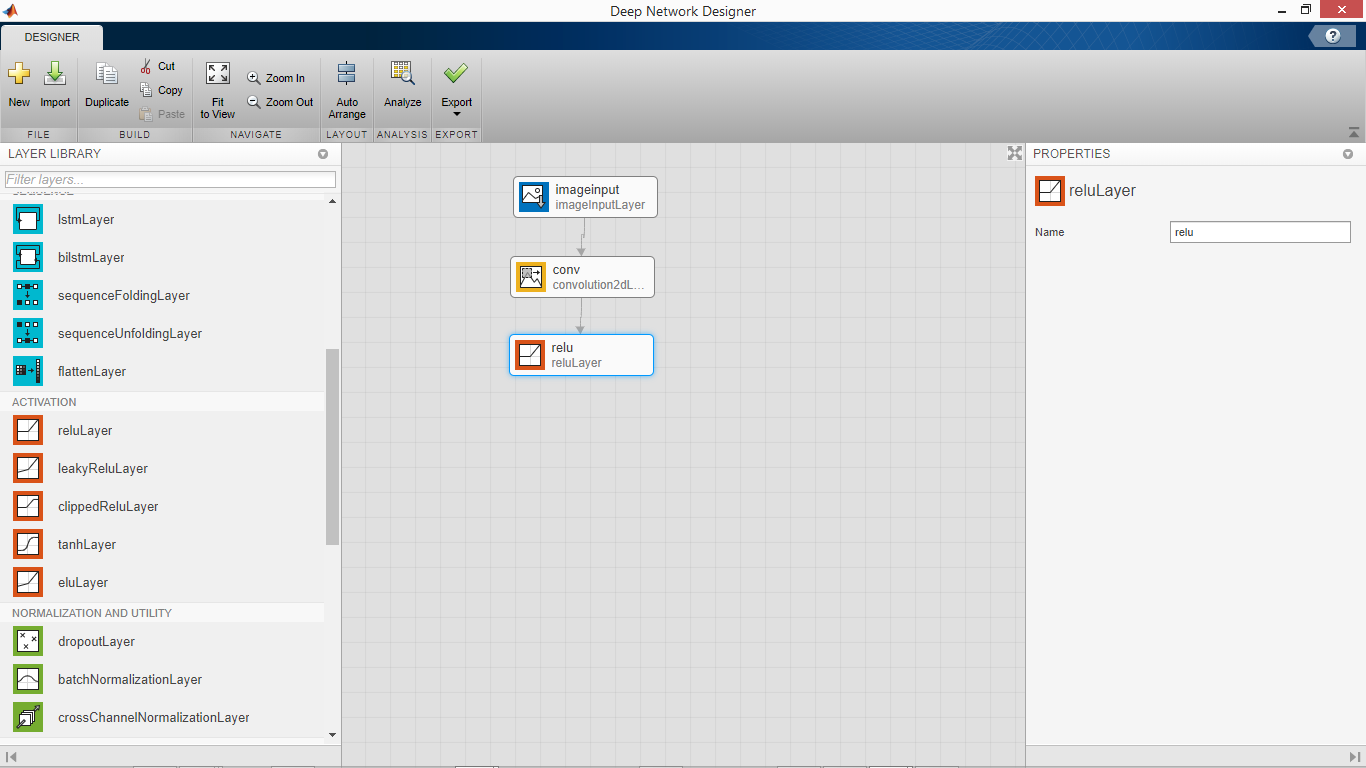
****

Fig 4.15 The ReLU layer

*Source: MATLAB*

The Rectified Linear Unit, or ReLU, is not a separate component of the convolutional neural networks' process. It's a supplementary step to the convolution layer that comes ahead of it.

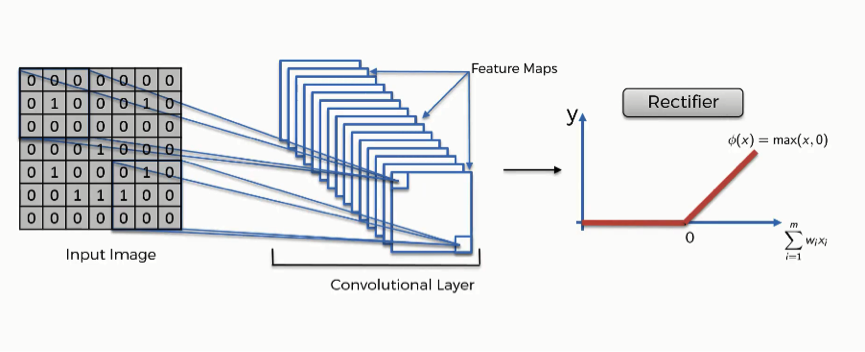


Fig 4.16 In-depth illustration of the ReLU layer

*Source: My computer/Photoshop Express*

The purpose of applying the rectifier function is to increase the non-linearity in our images. The reason we want to do that is that images are naturally non-linear. When you look at any image, you'll find it contains a lot of non-linear features (e.g. the transition between pixels, the borders, the colors, etc.). The rectifier serves to break up the linearity even further in order to make up for the linearity that we might impose an image when we put it through the convolution operation.

The ReLU function also fixes the problem of vanishing gradient. The representative solution to the vanishing gradient is the use of the Rectified Linear Unit (ReLU) function as the activation function. It is known to better transmit the error than the sigmoid function. Fig 4.17 depicts the ReLU function. It produces zero for negative inputs and conveys the input for positive inputs. Its implementation is extremely easy as well.

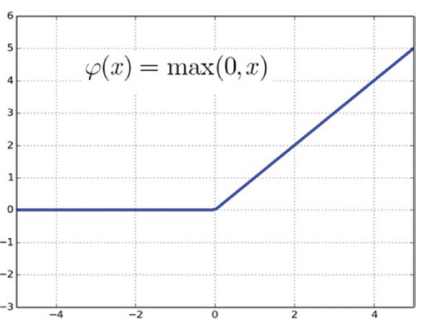


Fig 4.17 The ReLU function

*Source: www.towardsdatascience/ReLU\_activation/notes.html*

**4.7.4 Batch Normalization**

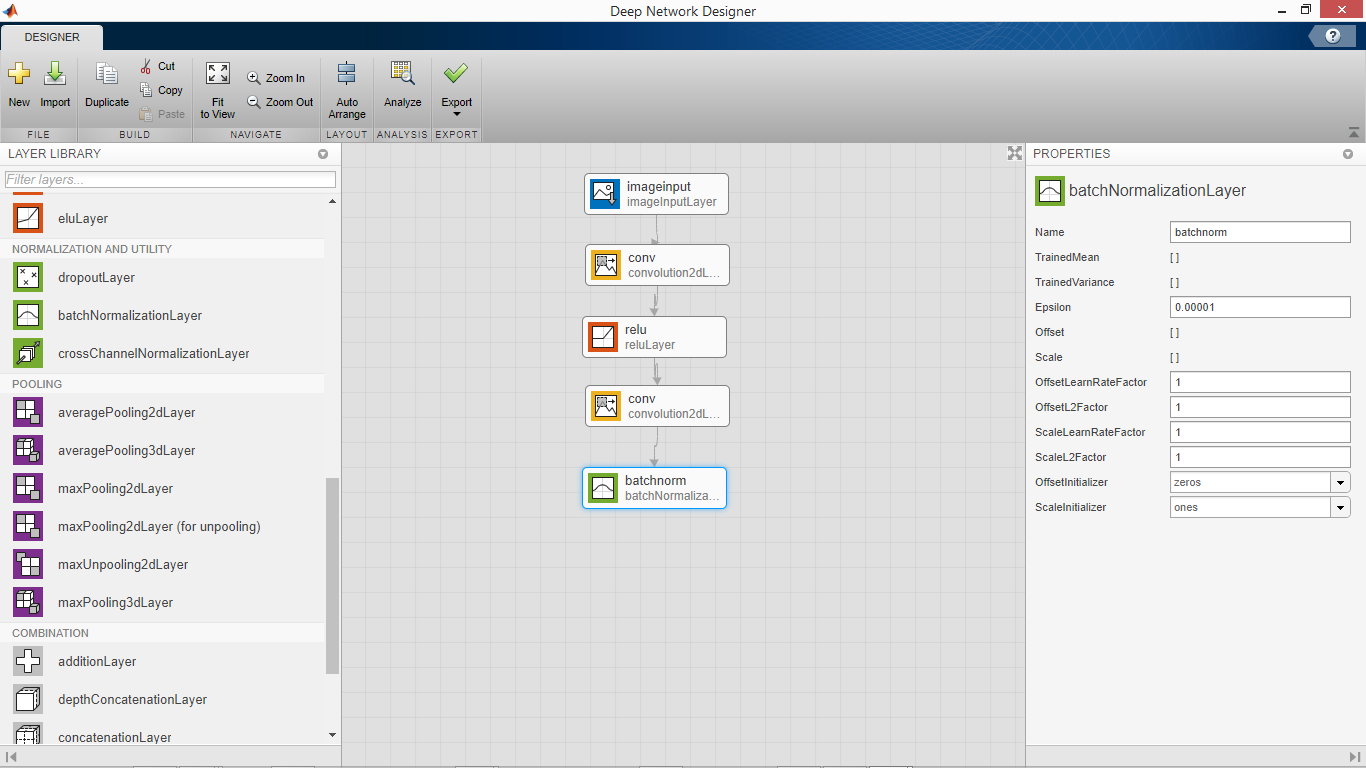


Fig 4.18 Batch Normalization Layer

*Source: MATLAB*

We include another convolution layer and then a batch normalization layer. Batch normalization is a technique for improving the speed, performance, and stability of artificial neural networks. Batch normalization was introduced in the year 2015. It is used to normalize the input layer by adjusting and scaling the activations. Batch normalization was initially proposed to solve internal covariate shift. During the training stage of networks, as the parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly, such that the current layer needs to constantly readjust to new distributions. This problem is especially severe for deep networks, because small changes in shallower hidden layers will be amplified as they propagate within the network, resulting in significant shift in deeper hidden layers.

Therefore, the method of batch normalization is proposed to reduce these unwanted shifts to speed up training and to produce more reliable models. Besides reducing internal covariate shift, batch normalization introduces many other benefits. With this additional layer, the network can use higher learning rate without vanishing or exploding gradients. Furthermore, batch normalization regularizes the network such that it is easier to generalize, and it is thus unnecessary to use dropout to mitigate overfitting. The network also becomes more robust to different initialization schemes and learning rates.

**4.7.5 Pooling Layer**

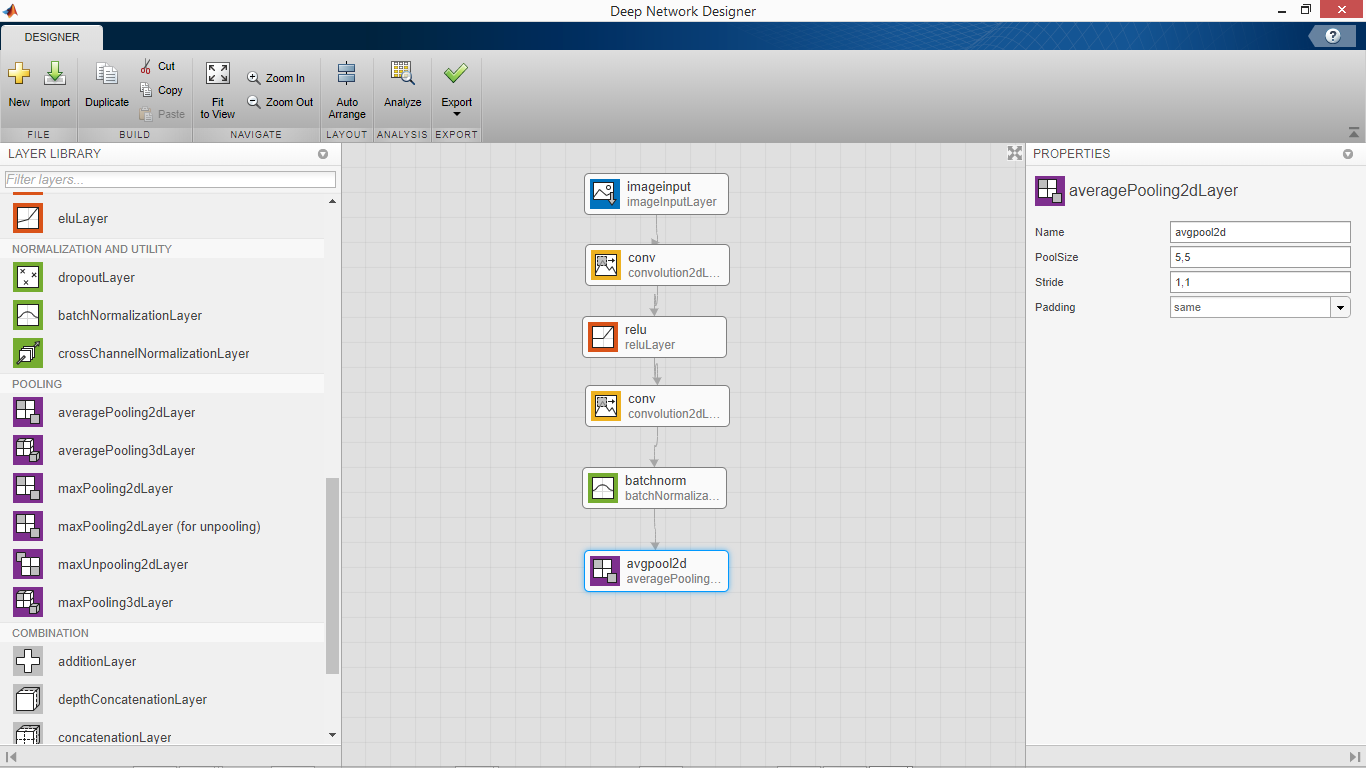


Fig 4.19 The Pooling Layer

*Source: MATLAB*

The pooling layer reduces the size of the image, as it combines neighboring pixels of a certain area of the image into a single representative value. Pooling is a typical technique that many other image processing schemes have already been employing. In order to conduct the operations in the pooling layer, we should determine how to select the pooling pixels from the image and how to set the representative value. The neighboring pixels are usually selected from the square matrix, and the number of pixels that are combined differs from problem to problem. The representative value is usually set as the mean or maximum of the selected pixels. Pooling layer is used to reduce the spatial volume of the input image after convolution. It is used between two convolution layers. If we apply the fully connected layer after the Convolution layer without applying pooling, then it will be computationally expensive and take a longer amount of time to train.

There are several ways to condense the information, but a usual one, which I will be using for this project, is known as average-pooling. In average-pooling, each group of entry points is transformed into the average value of the group of points instead of its maximum value. The operation of the pooling layer is surprisingly simple. As it is a two-dimensional operation, and an explanation in text may lead to more confusion, let’s go through an example**.**

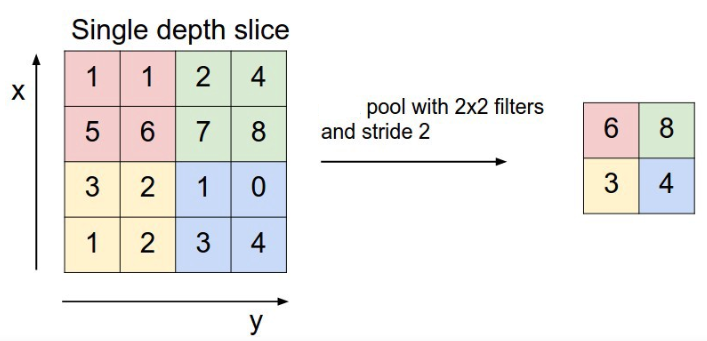
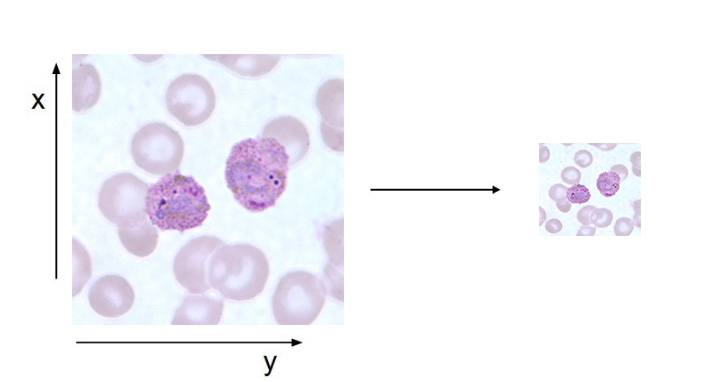


Fig 4.20 In-depth illustration of the pooling layer

*Source: My computer/Photoshop Express*

We combine the pixels of the input image into a 2 x 2 matrix without overlapping the elements. Once the input image passes through the pooling layer, it shrinks into a 2 x 2-pixel image. The pooling layer compensates for eccentric and tilted objects to some extent. For example, the pooling layer can improve the recognition of a malaria parasite, which may be distorted or off-center in the input image. In addition, as the pooling process reduces the image size, it is highly beneficial for relieving the computational load and preventing overfitting.

**4.7.6 The Fully-connected Layer**

Fully connected layer involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training. It comes before the Softmax layer which is used for multi-classification and the Output layer which contains the labels. The Output layer is at the end of the network. The fully connected (FC) layer in the CNN represents the feature vector for the input. This feature vector/tensor/layer holds information that is vital to the input. When the network gets trained, this feature vector is then further use for classification. During training, this feature vector is being used to determine the loss, and help the network to train.

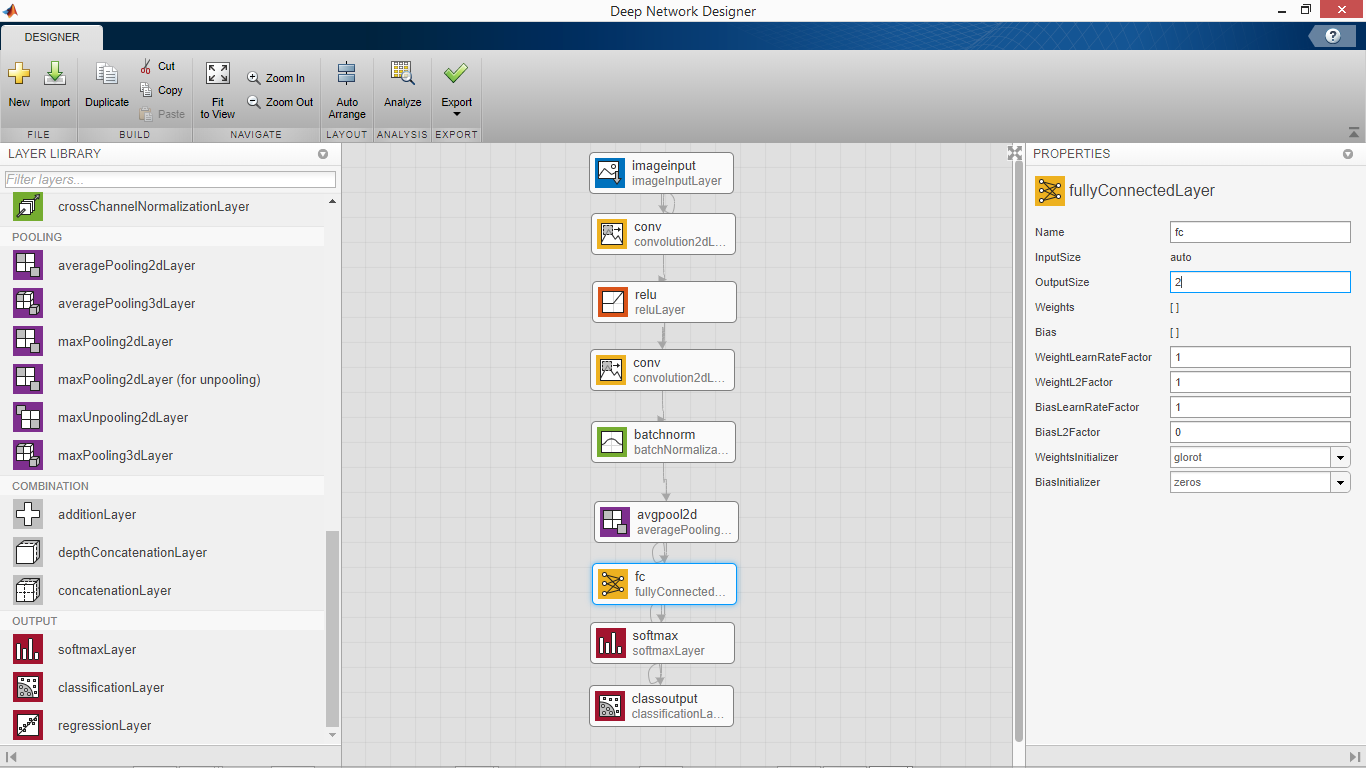


Fig 4.21 The Fully-connected Layer

*Source: MATLAB*

The convolution layers before the fully-connected layer hold information regarding local features in the input image such as edges, blobs, shapes, etc. Each convolution layer holds several filters that represent one of the local features. The fully-connected layer holds composite and aggregated information from all the convolution layers that matters the most.

**4.8 The Training Phase**

We begin the training phase by analyzing the convolutional neural network for errors and then exporting it into the MATLAB workspace for training. Using the network analysis tool, we get a full, detailed and concise view of the neural network and the component layers as well any alerts for warnings and errors.

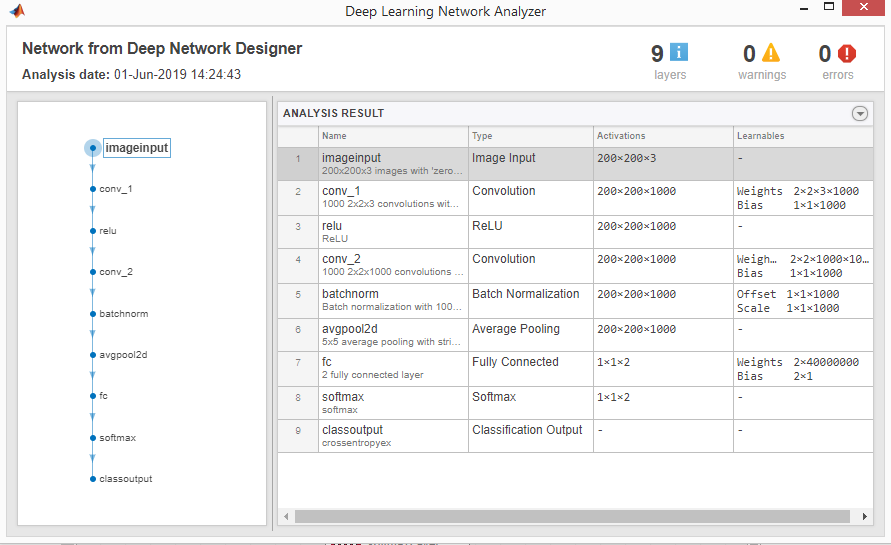


Fig 4.22 The Deep Learning Network Analyzer

*Source: MATLAB*

We can then export the network and proceed with the training phase using the MATLAB code shown in the figure below:

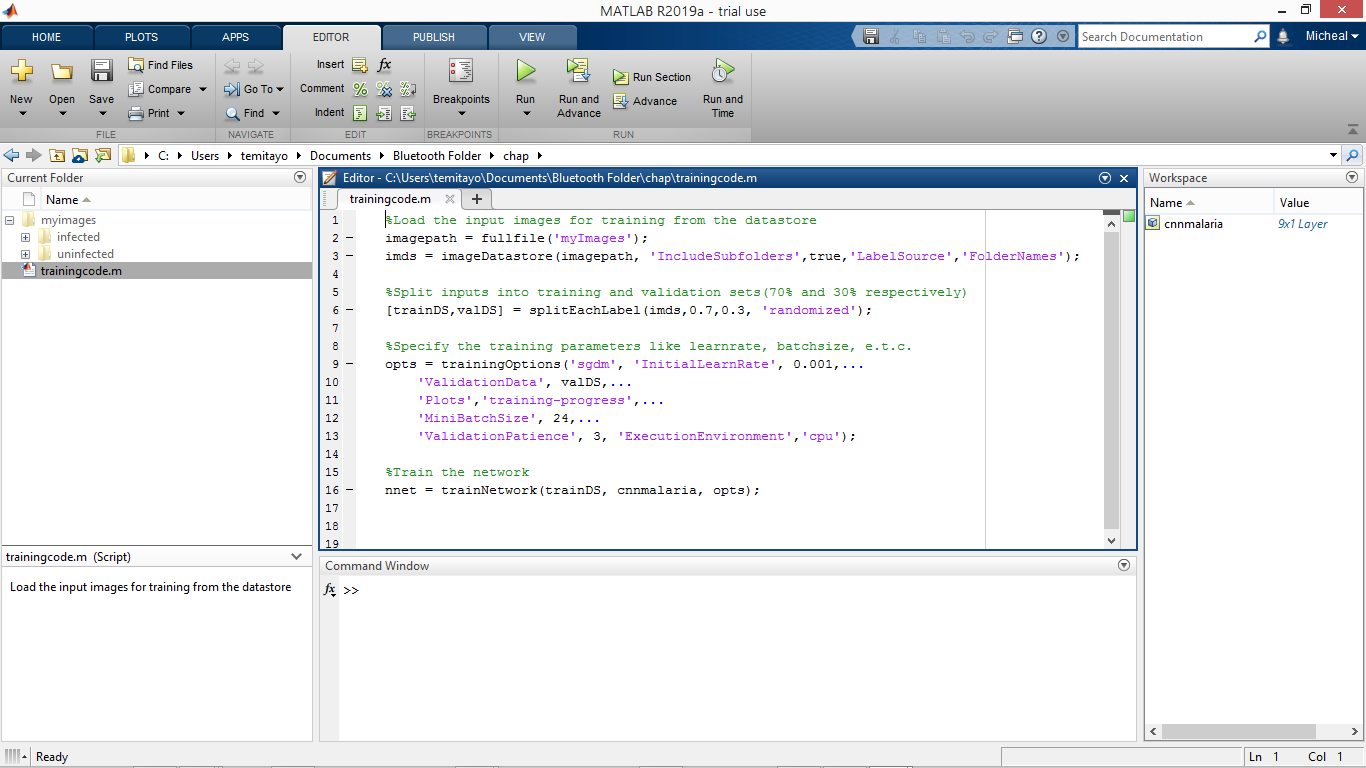


Fig 4.23 The MATLAB script for training the convolutional neural network

*Source: MATLAB*

The code on Line 11 (*'Plots','training-progress',...)* calls the figure plot table to show the training process which is shown below:

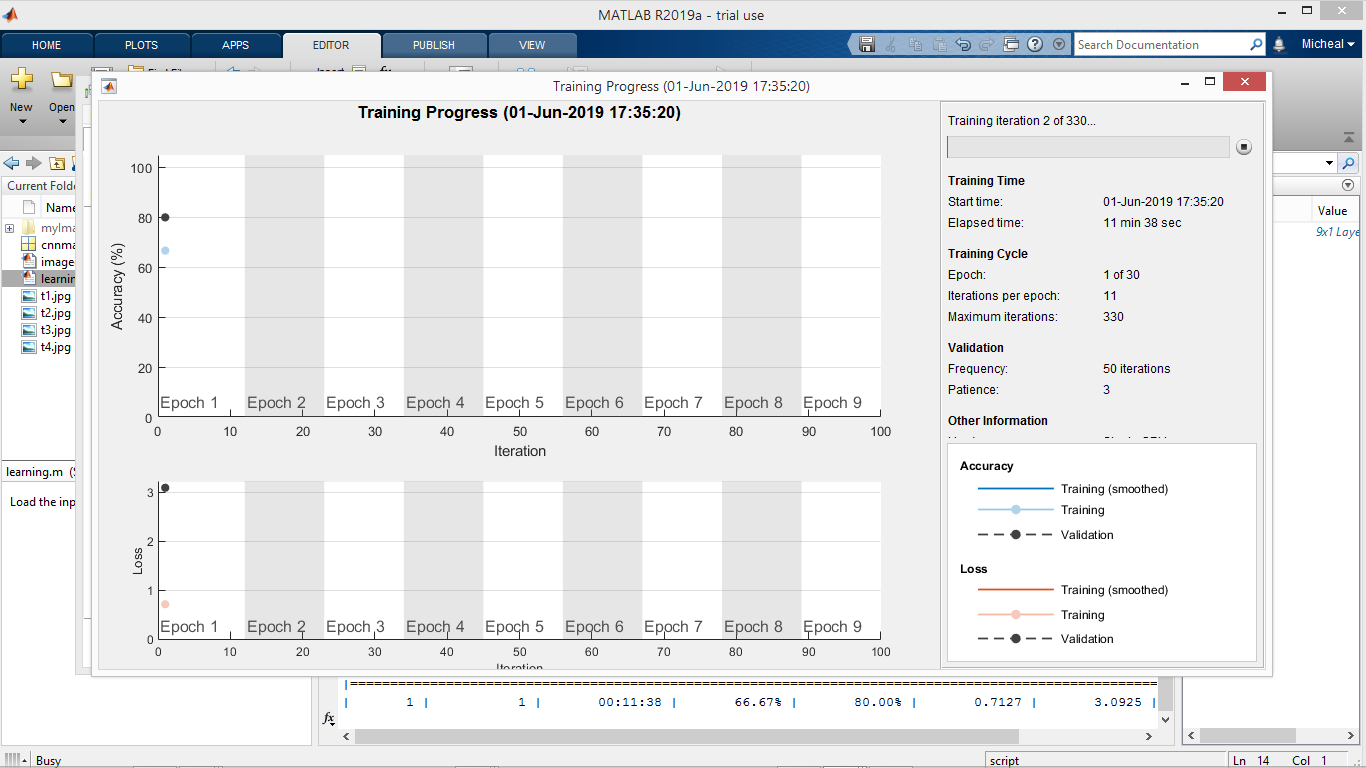


Fig 4.24 The Training Progress at Epoch 1

*Source: MATLAB*

As seen in Fig 4.24 above, the training and validation start at 66.67% and 80.00% respectively for the first epoch. In this network, there are 11 iterations per epoch. The training process continuously progresses and improves as shown in Fig 4.25 below. As the training continues in Fig 4.25, the loss increases temporarily for epoch 1.

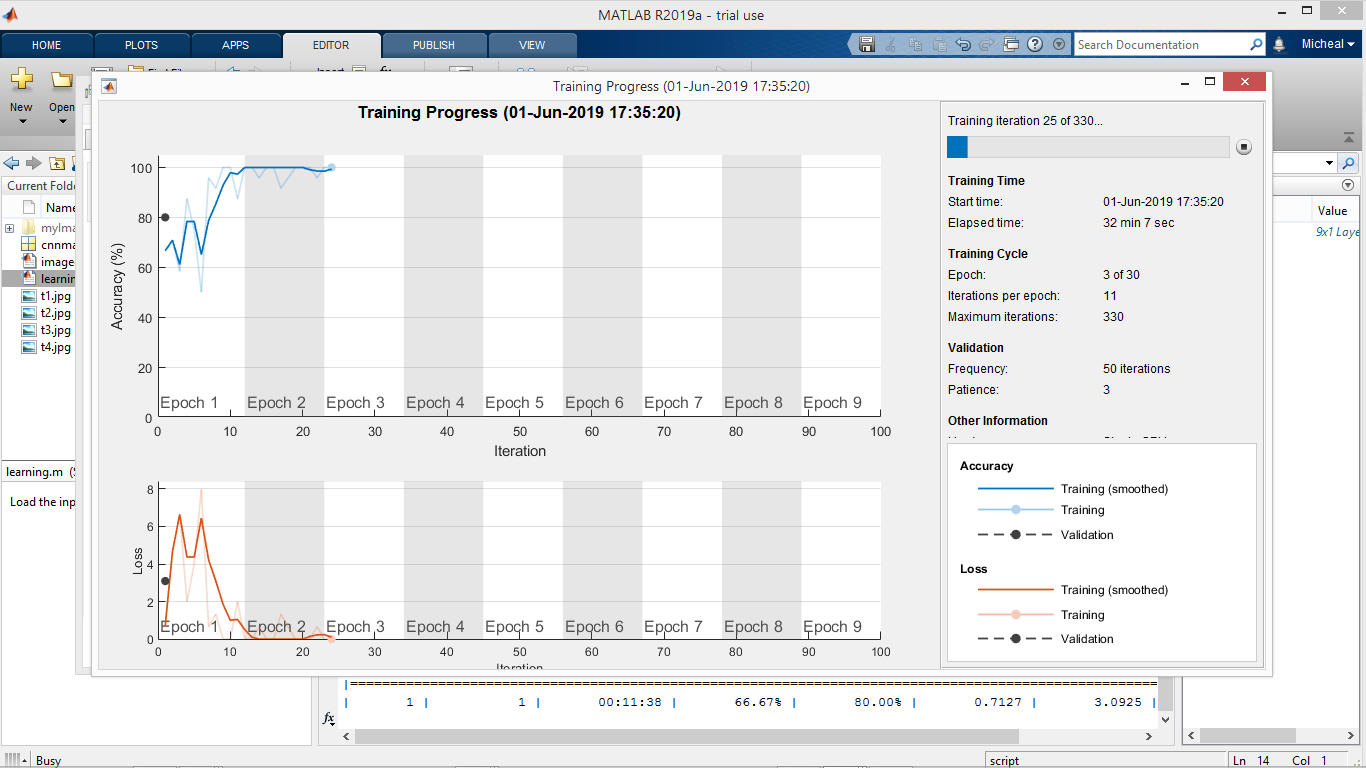


Fig 4.25 The Training Progress at Epoch 3

*Source: MATLAB*

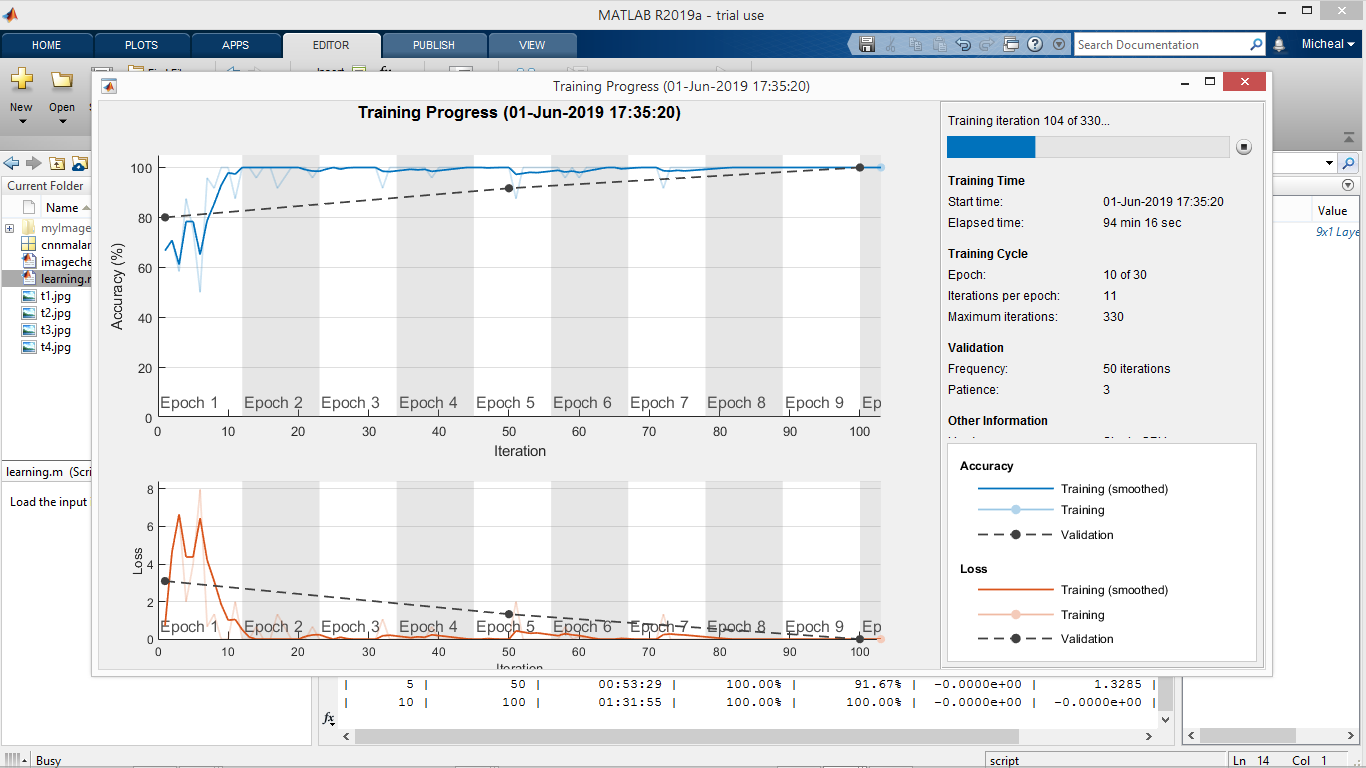


Fig 4.26 The Training Progress at Epoch 10

*Source: MATLAB*

In Fig 4.26 above, the network has completed 10 training epochs out of 30 with a training and validation accuracy of 100.00% for both. And as we can see in Fig 4.26 and Fig 4.27, the loss is gradually reducing. Loss is the penalty for a bad prediction i.e. loss is a number indicating how bad the model's prediction was on a single example.

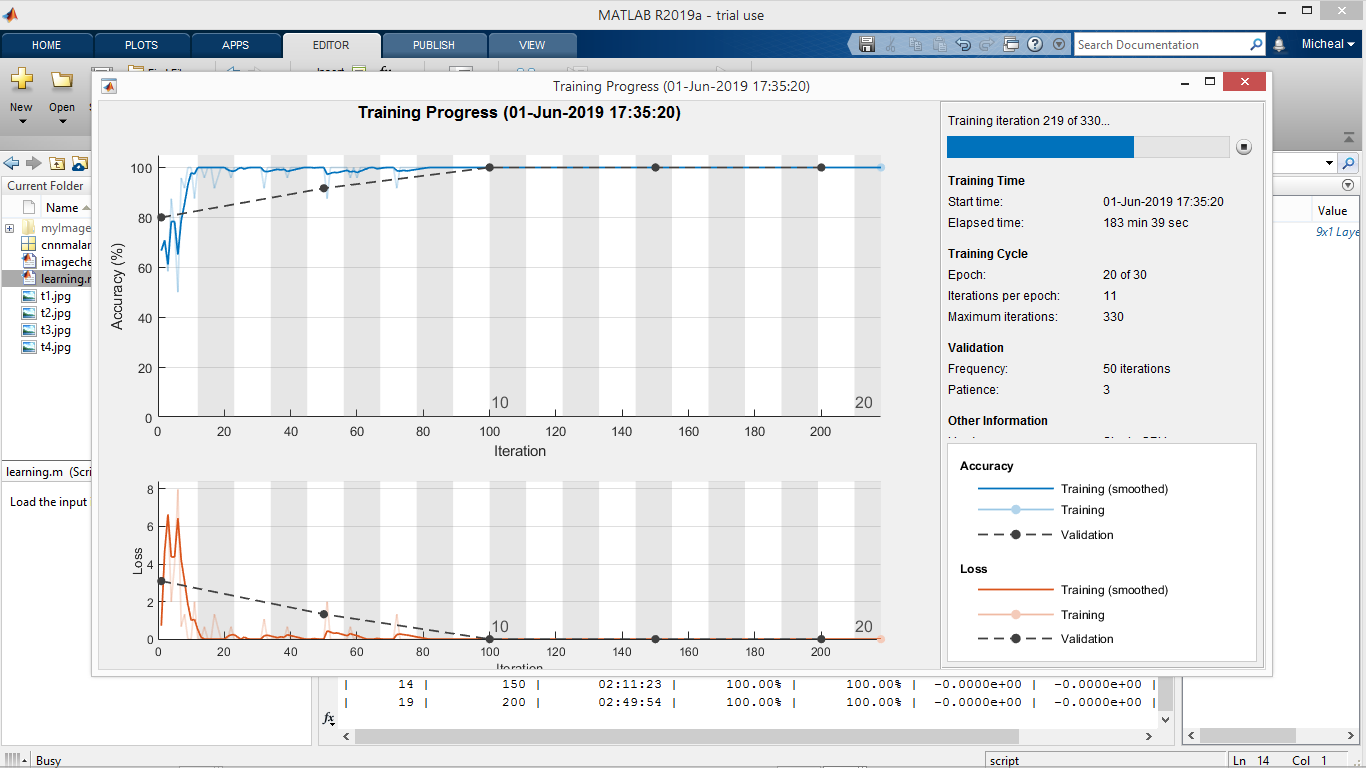


Fig 4.27 The Training Progress at Epoch 20

*Source: MATLAB*

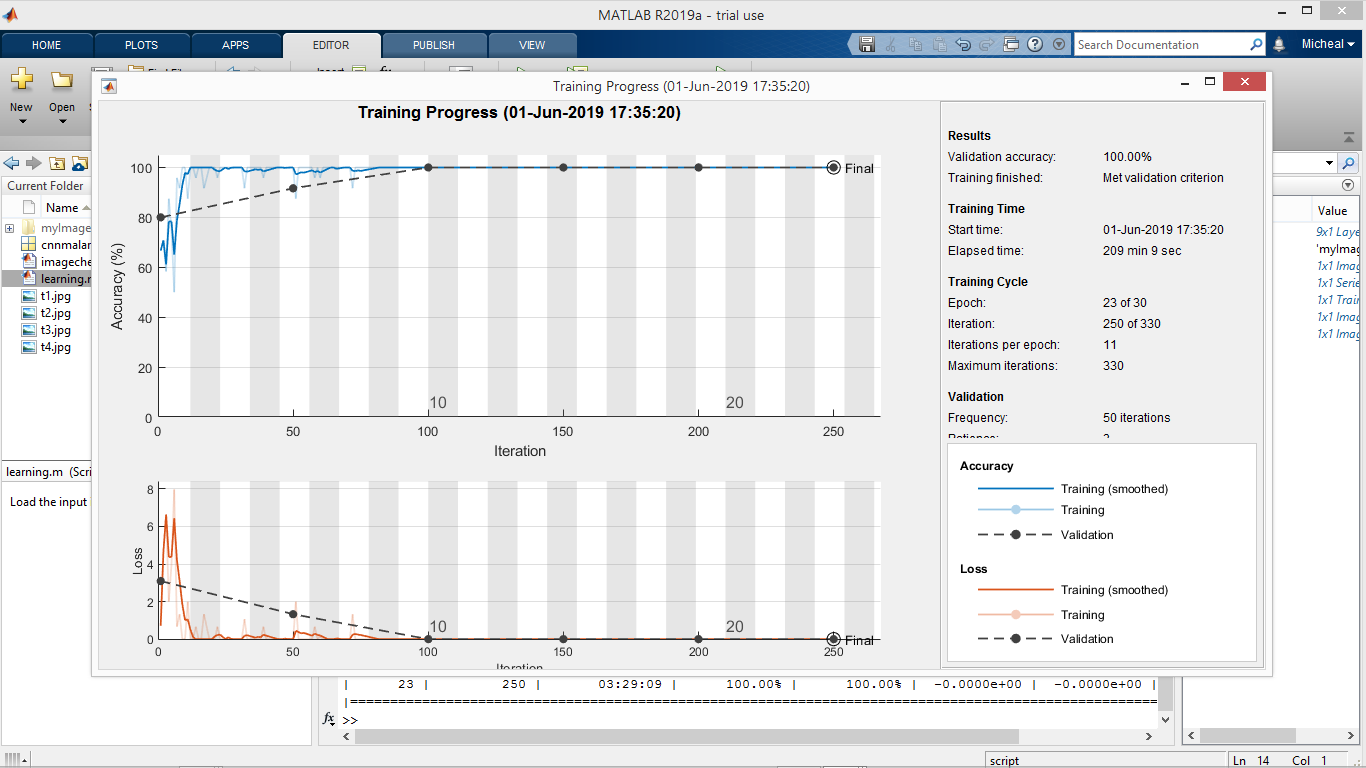
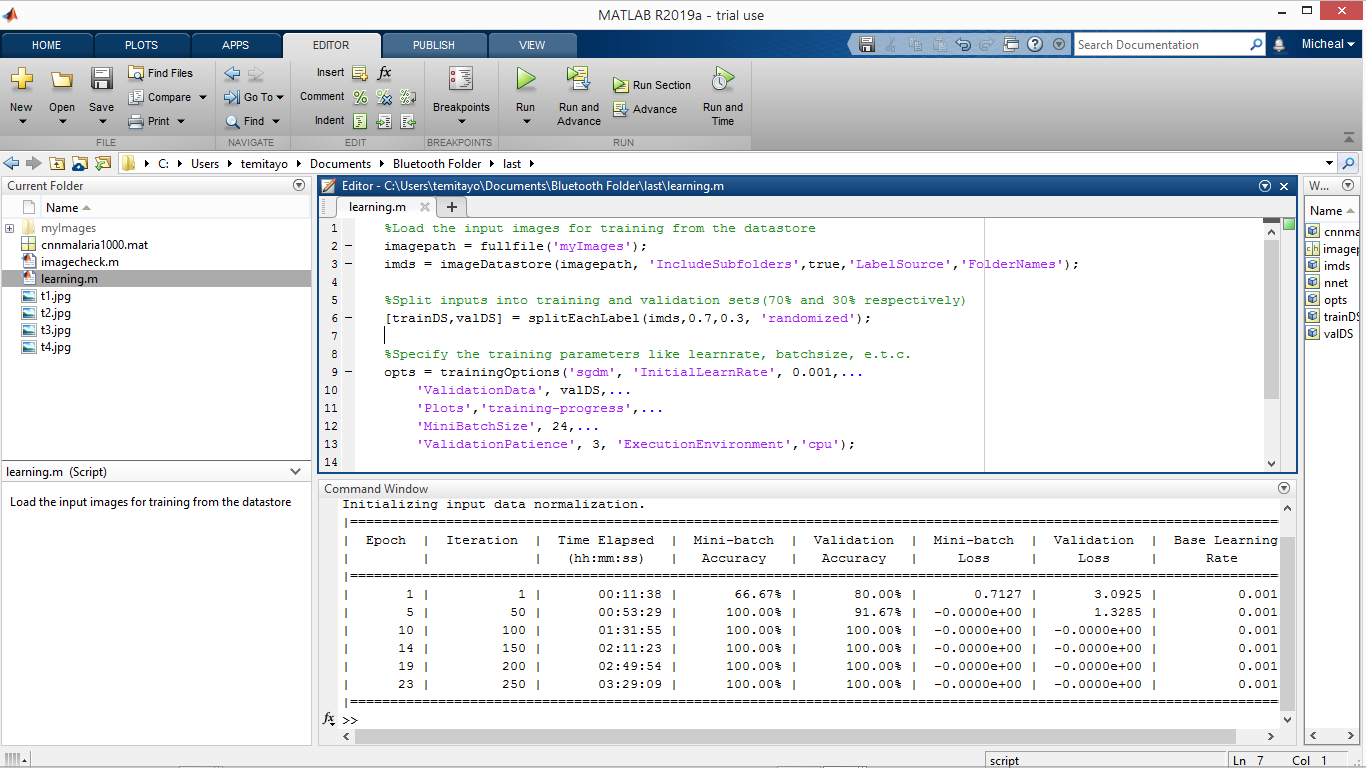


Fig 4.28 The Training Progress at the Final Epoch

*Source: MATLAB*

In fig 4.28 above, training has been completed and reached the final state with an overall validation accuracy of 100.00% in 30 epochs. As we can see in Fig 4.28, the training loss is zero. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples. In fig 4.29 below, the total result of the completed training process is shown in the command window.

Fig 4.29 The Total Result of the Completed Training Process

*Source: MATLAB*

We can take this simplified illustration below to explain and explore how the training is performed, as well as the validation and how the training data is processed from the moment it is inserted into the convolutional neural network until it develops its classes (**infected**, **uninfected**).

First we have our untrained network which we just designed and specified the outputs as two classes:

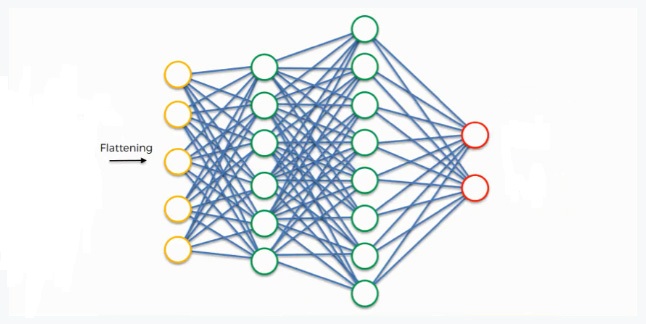


Fig 4.30 Simplified Illustration of the Training Process (1)

*Source: My computer/Photoshop Express*

At the very beginning, we have an input image which we convolve, pool, flatten, and then pass through the convolutional neural network. By the end of this channel, the neural network issues its predictions. Say, for instance, the network predicts the image to be infected by a probability of 80%, yet the image actually turns out to be uninfected. An error has to be calculated in this case with convolutional neural networks, it is more commonly referred to as a “loss function.” We use the cross-entropy function in order to achieve that. The loss function informs us of how accurate our network is which is shown on the figure table during the training process, which we then use in optimizing our network in order to increase its effectiveness. That requires certain things to be altered in our network. These include the weights (the blue lines connecting the neurons, which are basically the synapses), and the feature detector since the network often turns out to be looking for the wrong features and has to be reviewed multiple times for the sake of optimization. As we work to optimize the network, the information keeps flowing back and forth over and over (back propagation) until the network reaches the desired state.

During the training process, the fully-connected layer practically works as follows:

* The neuron in the fully-connected layer detects a certain feature; a purple stain (malaria parasite).
* It preserves its value.
* It communicates this value to both the “infected” and the “uninfected” classes.
* Both classes check out the feature and decide whether it's relevant to them.

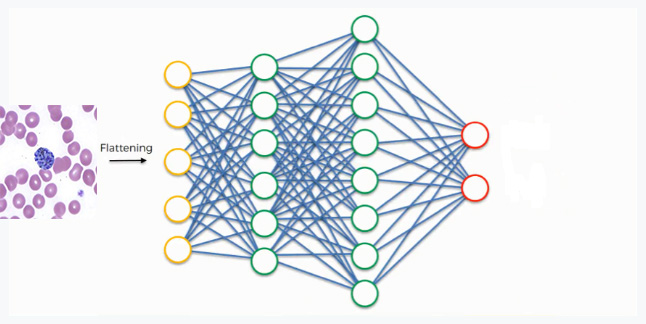


Fig 4.31 Simplified Illustration of the Training Process (2)

*Source: My computer/Photoshop Express*

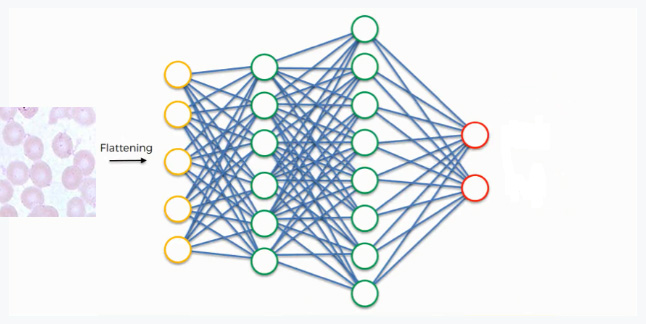


Fig 4.32 Simplified Illustration of the Training Process (3)

*Source: My computer/Photoshop Express*

In this network, the priority placed on the purple stain synapse is high because it’s a common feature for infected images, which means that the network is confident that it is an infected image. Since the information is constantly flowing in both directions, the “uninfected” class takes note of this and understands that the purple stain is a feature of an infected image, then it simply can't belong to the uninfected class. Even if at first it would have considered the images with that feature, now it dismisses them. This explains the reason why the validation and training accuracy is very poor in the first few epochs of the training process as the network has is still considering the feature of the two classes (infected, uninfected). This happens gradually as it receives the same reading multiple times. The infected class on its part will start focusing more on the features carrying the highest priorities (the three thick purple lines in the figure below), and it will ignore the rest. The same process simultaneously occurs with the uninfected class, enabling it to pick out its own priority features. What we end up with is what you see in the image below. As this process goes on repeat for hundreds or thousands of times, you find yourself with a well-trained and optimized convolutional neural network.

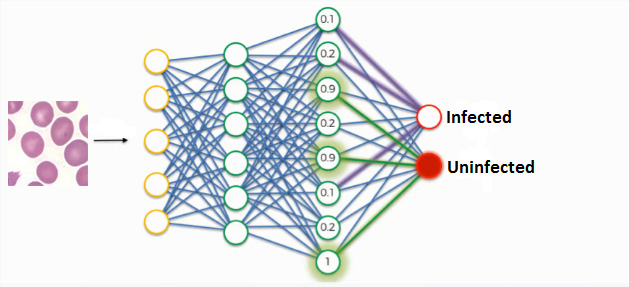


Fig 4.33 Simplified Illustration of the Training Process (1)

*Source: My computer/Photoshop Express*

**4.9 The Implementation of the New System**

The convolutional neural network was applied to four test images (Test Image 1, 2, 3 and 4). These images were chosen due to their lack of visual noise (red blood cells are evenly colored, no background artifacts) in order to carry out a baseline performance evaluation and identify strengths and weaknesses in the neural network. The MATLAB code to test the convolutional neural network on new images is given below:

---------------------------------------------------------------------------------------------------------------------

*Clear*

*load cnnmalaria1000;* %Load the Neural network

*im = imread('imagename.jpg');* %Read the image from the workspace

*im = imresize(im,[200,200]);* %Resize the image to meet the specification of the nnet

*label = classify(nnet, im);* %Classify the picture using the neural network

*im = imresize(im,[600,600]);* %Increase the size of the image after classification to 600px

*imshow(im);* %Show the image

*title("CNN Classification = " + char(label));* %Show the nnet label as a string of characters

*hold on;* %Keep displaying the image

------------------------------------------------------------------------------------------------------------------

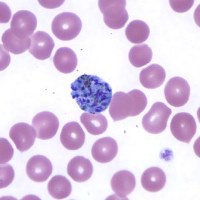


Fig 4.34 Test Image 1

*Source: Faith Mediplex Hospital Laboratory*

The convolutional neural network exhibits a good performance on test image 1. It demonstrates the identification of the infected cell in the image of the blood sample by detecting the malaria parasite which is clearly visible right in the middle of image and the convolutional neural network classifies it accurately as infected.

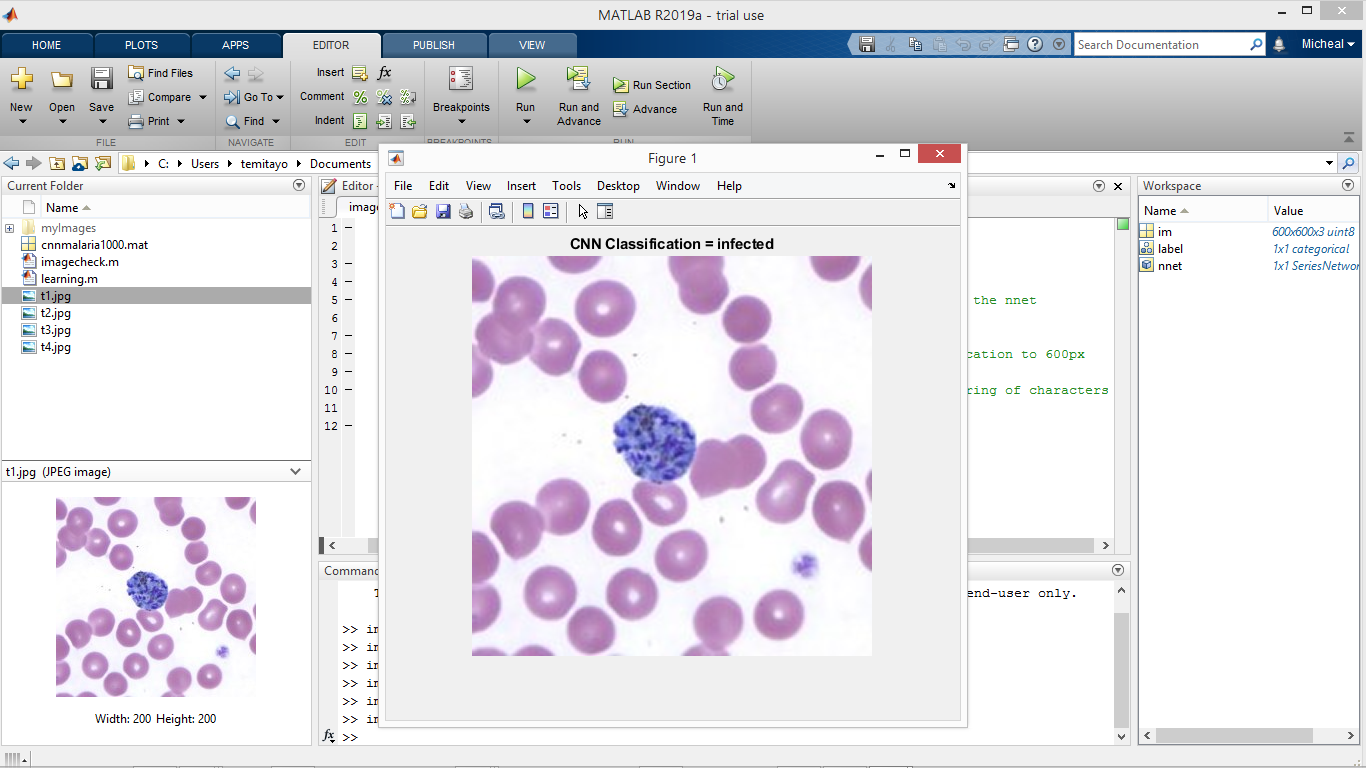


Fig 4.35 The Result of Test Image 1 using the Convolutional Neural Network

*Source: MATLAB*

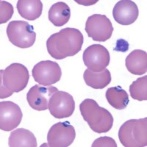


Fig 4.36 Test Image 2

*Source: Faith Mediplex Hospital Laboratory*

Test image 2 is dominated by parasites in the schizont stage, which appear as round purple clusters containing many dark purple spots. The convolutional neural network also exhibits a good performance on test image 2 by identifying the infected cells in the image of the blood and classifying the image accurately as infected.

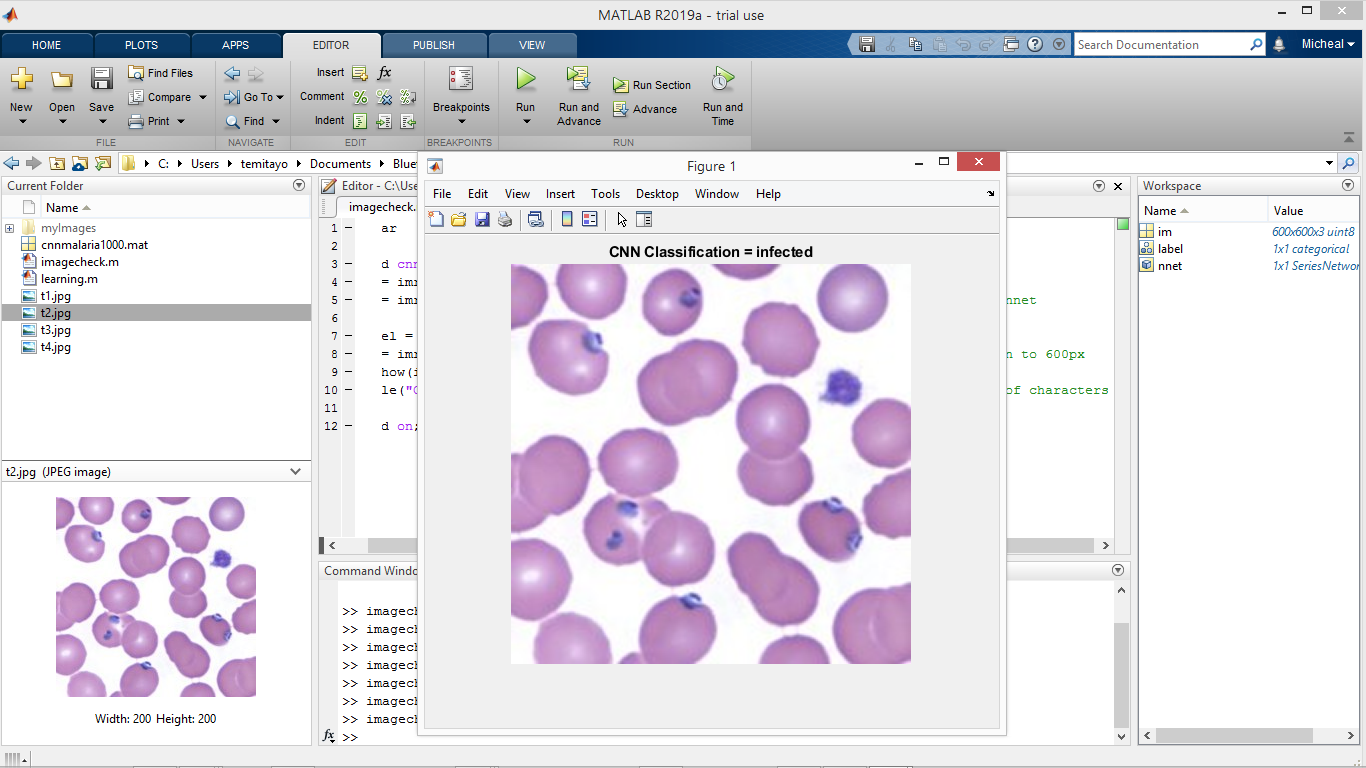


Fig 4.37 The Result of Test Image 2 using the Convolutional Neural Network

*Source: MATLAB*

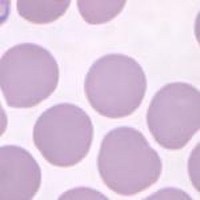


Fig 4.38 Test Image 3

*Source: Faith Mediplex Hospital Laboratory*

In Test image 3, the red blood cells are evenly colored, no background artifacts and most importantly no malaria parasites. The convolutional neural network exhibits a good performance on test image 3 by classifying it accurately as uninfected.

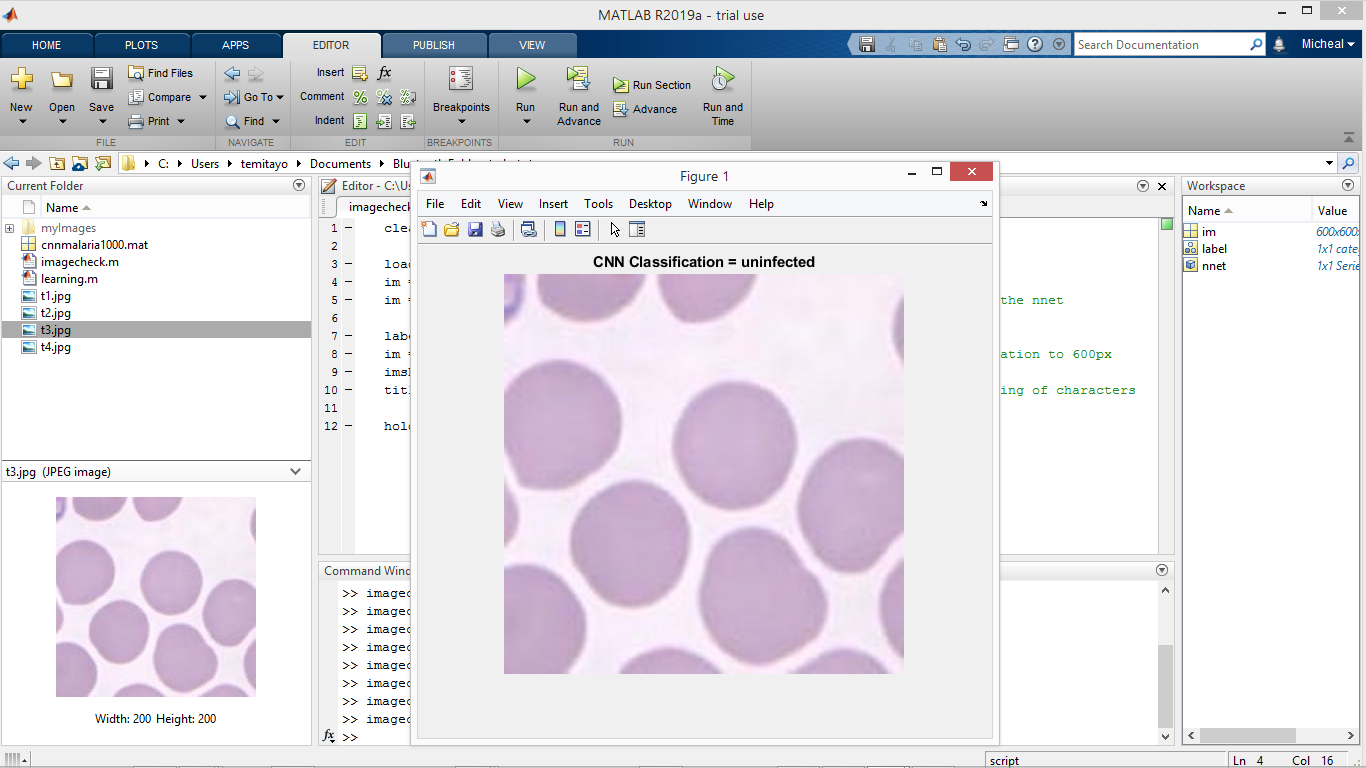


Fig 4.39 The Result of Test Image 3 using the Convolutional Neural Network

*Source: MATLAB*



Fig 4.40 Test Image 4

*Source: Faith Mediplex Hospital Laboratory*

In Test image 4, the red blood cells are also evenly colored, no background artifacts and most importantly no malaria parasites. The convolutional neural network exhibits a good performance on test image 4 by classifying it accurately as uninfected.

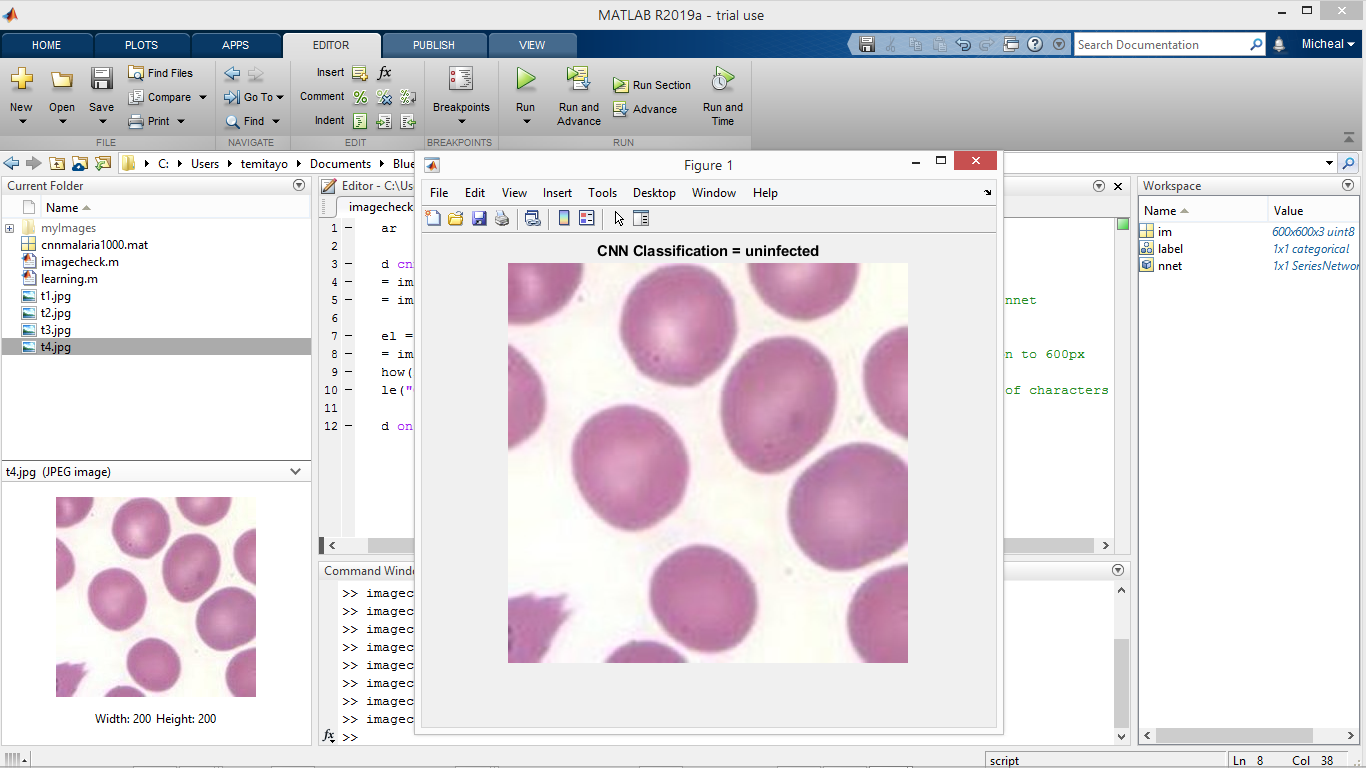


Fig 4.41 The Result of Test Image 4 using the Convolutional Neural Network

*Source: MATLAB*