SONA COLLEGE OF TECHNOLOGY

SMARTBRIDGE- EXTERNSHIP

PROJECT TITLE:

IMPLEMENT DEEP LEARNING TECHNIQUES TO DETECT MALARIA USING IBM CLOUD

Done by ;

Team members :

Puram Chandra sekhar

Kancherla Vengababu

Ganta Vedamouli

INDEX PAGE

1) INTRODUCTION

1.1) Overview

Abstract:Malaria is the deadliest disease in the earth and big hectic work for the health department. The traditional way of diagnosing malaria is by schematic examining blood smears of human beings for parasite-infected red blood cells under the microscope by lab or qualified technicians. This process is inefficient and the diagnosis depends on the experience and well knowledgeable person needed for the examination. Deep Learning algorithms have been applied to malaria blood smears for diagnosis before. However, practical performance has not been sufficient so far. This paper proposes a new and highly robust machine learning model based on a convolutional neural network (CNN) which automatically classifies and predicts infected cells in thin blood smears on standard microscope slides. A ten-fold cross-validation layer of the convolutional neural network on 27,558 single-cell images is used to understand the parameter of the cell. Three types of CNN models are compared based on their accuracy and select the precise accurate - Basic CNN, VGG-19 Frozen CNN, and VGG-19 Fine Tuned CNN. Then by comparing the accuracy of the three models, the model with a higher rate of accuracy is acquired

1.2) Purpose

The purpose is to develop an automated system for enhancement, segmentation and classification of Malaria. The system can be used byneurosurgeons and healthcare specialists to detect malria. The system incorporates image processing, pattern analysis, and computer vision techniques and is expected to improve the sensitivity, specificity, and efficiency of Malria detection. The primary goal of this projects is to extract meaningful and accurate information from these images with the least error possible. The proper combination and parameterization of the phases enables the development of adjunct tools that can help on the early stages of Malaria.

2) LITERATURE SURVEY

2.1) Existing Problem

Malaria being a life threatening disease has caused deep research interests among the scientists all over the world. Earlier, malaria was mostly diagnosed in the laboratory setting requiring a great deal of human expertise. Automatic systems such as those relying on machine learning techniques were initially studied to overcome this problem. Techniques reported in this domain of study mostly considered the hand-crafted features in decision making. For example, [42,43] relied on morphological factors for feature extraction and applied SVM and Principal Component Analysis (PCA) [44] for the classification purpose. However, the accuracy achieved through these kinds of model is low compared to the more recently studied deep learning based techniques.

For automatic detection of malaria pathogens from the microscopic images, Convolutional Neural Network (CNN) [45] received much attention from the researchers in recent times. Dong et al. for example evaluated the performances of three popular Convolutional Neural Network [46], namely LeNet-5 [47], AlexNet [48] and GoogLeNet [49]. In addition, they trained an SVM classifier for comparison purposes and concluded that CNN is advantageous over SVM in terms of the capacity of learning image features automatically. To find the optimal layer of a pre-trained model to extract features from underlying malaria parasite data, Rajaraman et al. evaluated the performances of AlexNet, VGG-16 [50], ResNet-50 [51], Xception [52], DenseNet-121 [53] along with their custom-built model [24]. Liang et al. reported 97.37% accuracy in detection with the help of their 16-layered CNN model and claimed that their model is superior to transfer learning models [54]. Hung et al. pre-trained a model on Imagenet [55] but fine-tuned it on own data for detecting malaria parasite [35].Most of the reported studies in general focused on improving the accuracy using traditional machine learning or deep learning based techniques. A few notable exceptions are the works of Quinn et al. [25] and Rosado et al. [56] who also tackled the problem of computational efficiency of their models. However, both studies reported a notable drop in model accuracy in pursuit of computation efficiency. Rosado et al. also explored the use of smartphones to detect malaria parasites and white blood cells. They reported sensitivity and specificity of 80.5% and 93.8% for trophozoite detection and 98.2% and 72.1% respectively for white blood cells using Support Vector Machine (SVM) [57]. Using the same classifier, they achieved a sensitivity and specificity of 98.2% and 72.1% respectively for white blood cells. The use of deep learning technique,particularly those that leverage transfer learning [31,40,58] yield superior results when considering classification metrics. Unfortunately, the models proposed in those studies are also quite expensive in terms of the required computational resources. Also notable is the fact that while Rosado et al. showed that their model can be deployed on mobile phones, their demonstration did not include low cost mobile phones commonly used in the poorer regions of the world.

In contrast, in this work, we propose several deep learning models which achieve classification performance comparable to the previously reported highly accurate deep learning based solutions. In addition, our models are efficient in terms of required computational resources and have been demonstrated to work efficiently on smart mobile devices, including that are available at very low cost

2.2) Proposed Solution

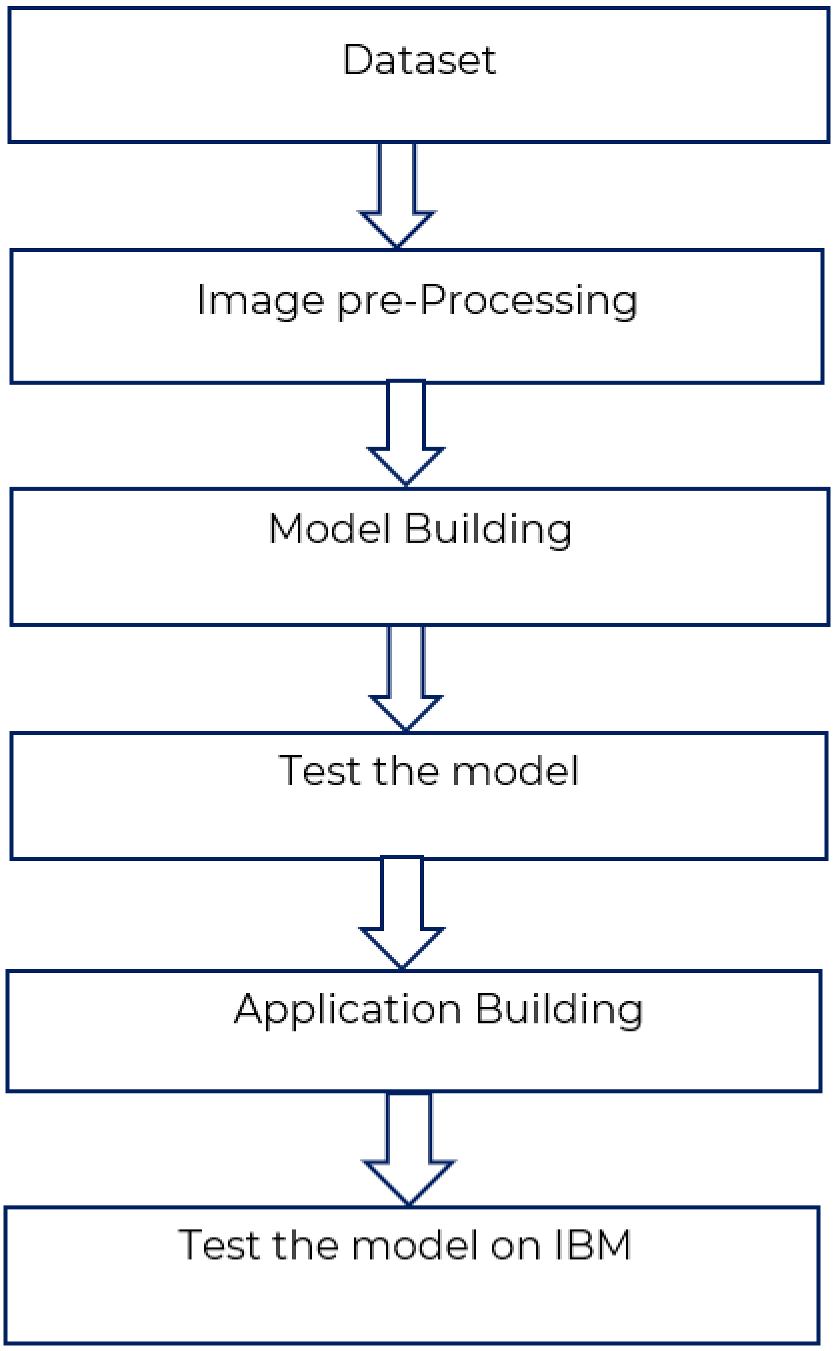
The proposed system has mainly six modules. They are

- Dataset Collection
- Image pre-processing
- Model Building
- Test the model
- Application Building
- Test the model on IBM

In dataset we can take multiple MRI images and take one as input image. In pre-processing image to encoded the label and resize the image. In split the data we set the image as 80% Training Data and 20% Testing Data. Then build CNN model train deep neural network for epochs. Then we created a '.h5' for building web application using flask , using that MRL images we predict the tumor in flask web application . After that using the data we do IBM deployment in IBM Watson studio there we get another file ,using that file we build we application using flask and we can predict the brain tumor using dataset ,here we are using deep neural networks in predicting if the tumor is present or not. And then we build web applications using the Flask framework to show that a patient is effected by brain tumor

Malaria being a life threatening disease has caused deep research interests among the scientists all over the world. Earlier, malaria was mostly diagnosed in the laboratory setting requiring a great deal of human expertise. Automatic systems such as those relying on machine learning techniques were initially studied to overcome this problem. Techniques reported in this domain of study mostly considered the hand-crafted features in decision making. For example, [42,43] relied on morphological factors for feature extraction and applied SVM and Principal Component Analysis (PCA) [44] for the classification purpose. However, the accuracy achieved through these kinds of model is low compared to the more recently studied deep learning based techniques.

2) THEORETICAL ANALYSIS



3.1)BlockDiagram

3.2 .1) Software designing

Anaconda Navigator :

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command-line interface (CLI).

Jupyter Notebook :

Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI). A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text mathematics, plots and rich media, usually ending with the “. ipynb" extension.

Tensor flow :

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers can easily build and deploy ML powered applications.

Keras :

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

Flask :

Flask is a microframework written in python. It is classified as a microframework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools

3.2 .2) Hardware designing

· Processor: Intel core i5 or above.

· 64-bit, quad-core, 2.5 GHz minimum per core

· Ram: 8 GB or more

· Hard disk: 10 GB of available space or more.

· Display: Dual XGA (1024 x 768) or higher resolution monitors

· Operating system: Window

4) EXPERMENTAL INVESTIGATIONS

The analysis of the project that would be developed by our hands. It consists of six steps where the execution starts from taking an input image from the data set followed by the image pre-processing, model building, testing the model and then Save the model and its dependencies, Building a Web application using flask that integrates with the model built using IBM Watson studio. Finally, the output is observed after all the above mentioned steps are completed.

4.1) Dataset collection :

The data set used is has been downloaded from Kaggle which consists of nearly 250 images that are used to test and train the system.

4.2) Image pre-processing :

Image Pre-processing includes the following main tasks

➢ Import ImageDataGenerator Library.

The Maleria scaned image dataset has been downloaded from the Kaggle. The dataset consists of around 5000+ images, including infected and uninfected. These scanned images are taken as input to the primary step. The pre-processing is an essential and initial step in improving the quality of the scanned Image. The critical steps in pre-processing are the reduction of impulsive noises and image resizing. In the initial phase,ImageDataGenerator class is used to load the images with different modifications like considering the zoomed image, flipping the image and rescaling the image to a range of 0 and 1

➢ Configure ImageDataGenerator Class.

In image processing, image acquisition is done by retrieving an image from dataset for processing. It is the first step in the workflow sequence because, without an image no processing is possible. The image that is acquired is completely unprocessed. Here we process the image using the file path from the local device

➢ Applying ImageDataGenerator functionality to the trainset and test set.

Specify the path of both the folders in the flow_from_directory method.

4.3) Model Building :

The neural network model is to be built by adding different network layers like convolution, pooling, flattening, dropout and neural layers.

For this step we need to import Keras and other packages that we're going to use in building the CNN. Import the following packages: Sequential is used to initialize the neural network. Adding Convolution layer is used to make the convolutional network that deals with the images. Adding Pooling layer is used to add the pooling layers. Flatten layer is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. Adding fully connected layer which includes hidden layer to the neural network.

SEQUENTIAL:

To initialize the neural network, we create an object of the Sequential class.

Ex; model=Sequential()

CONVOLUTION:

To add the convolution layer, we call the add function with the classifier object and pass in Convolution2D with parameters. The first argument feature_detectors which is the number of feature detectors that we want to create. The second and third parameters are dimensions of the feature detector matrix. We used 256 feature detectors for CNNs. The next parameter is input shape which is the shape of the input image. The images will be converted into this shape during pre-processing. If the image is black and white it will be converted into a 2D array and if the image is coloured it will be converted into a 3D array. In this case, we'll assume that we are working with coloured images. Input_shape is passed in a tuple with the number of channels, which is 3 for a coloured image, and the dimensions of the 2D array in each channel. If you are not using a GPU it's advisable to use lower dimensions to reduce the computation time. The final parameter is the activation function. Classifying images is a nonlinear problem. So, we use the rectifier function to ensure that we don't have negative pixel values during computation.
Ex; model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))

POOLING:

The Pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Generally, we use max pooling. In this step we reduce the size of the feature map. Generally, we create a pool size of 2x2 for max pooling. This enables us to reduce the size of the feature map while not losing important image information.
Ex; model.add(MaxPooling2D(pool_size=(2,2)))

FLATTENING:

In this step, all the pooled feature maps are taken and put into a single vector for inputting it to the next layer. The Flatten function flattens all the feature maps into a single long column.
Ex; model.add(Flatten())

FULLY CONNECTION:

The next step is to use the vector we obtained above as the input for the neural network by using the Dense function in Keras. The first parameter is output which is the number of nodes in the hidden layer. You can determine the most appropriate number through experimentation. The higher the number of dimensions the more computing resources you will need to fit the model. A common practice is to pick the number of nodes in powers of two. The next layer we have to add is the output layer. In this case, we'll use the sigmoid activation function since we expect a binary outcome. If we expected more than two outcomes, we would use the SoftMax function.

4.4) Test the model :

The model is to be tested with different images to know if it is predicting correctly. . In split the data we set the image as 80% Training Data and 20% Testing Data. Then build CNN model train deep neural network for epochs

4.5) Application Building

After the model is built, we will be integrating it into a web application so that normal users can also use it. The users need to give the scan to know if is infected or not.

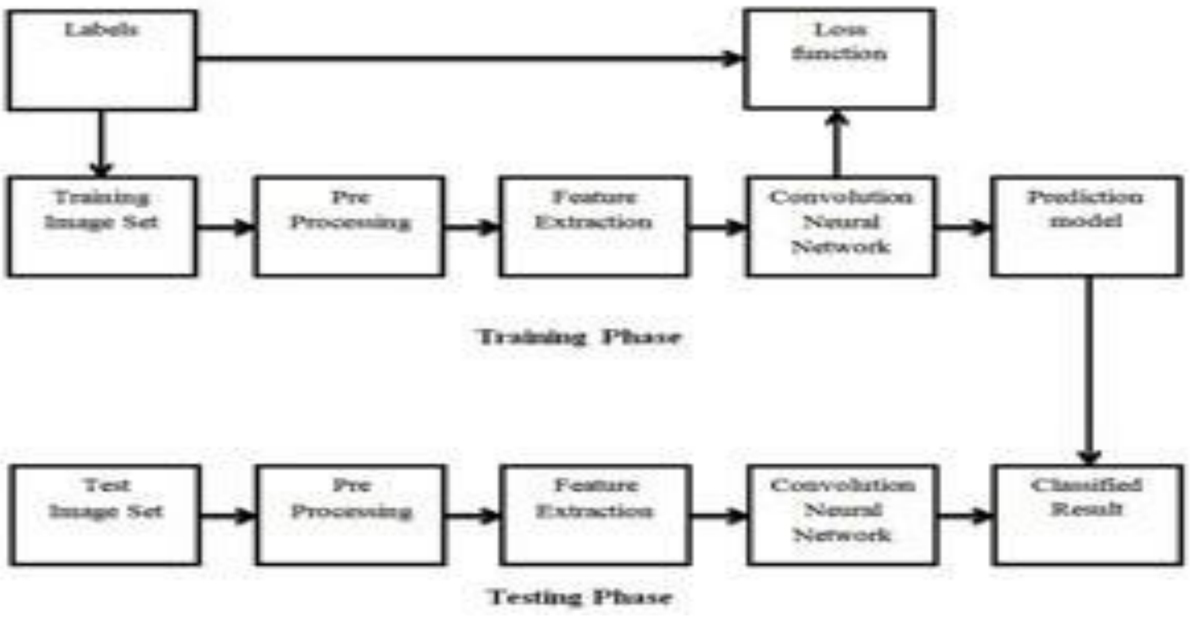
4.6) Test the model in IBM

At finally test the model in IBM using IBM Watson studio and then build a '.h5' using this file we can predict weather a patient is effected infected or not in web application using flask

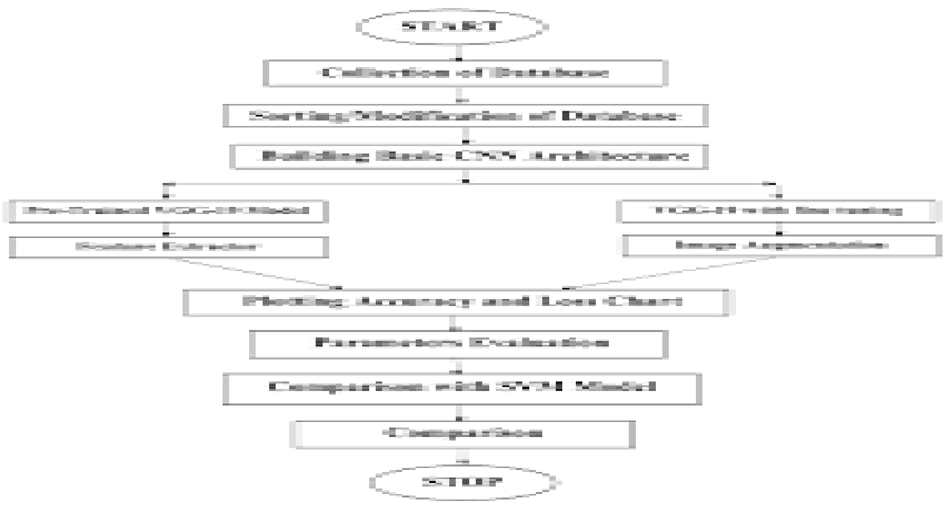
Finally , we detect whether the given Scanned image has infected or not using IBM Watson studio.

5) FLOWCHAT :

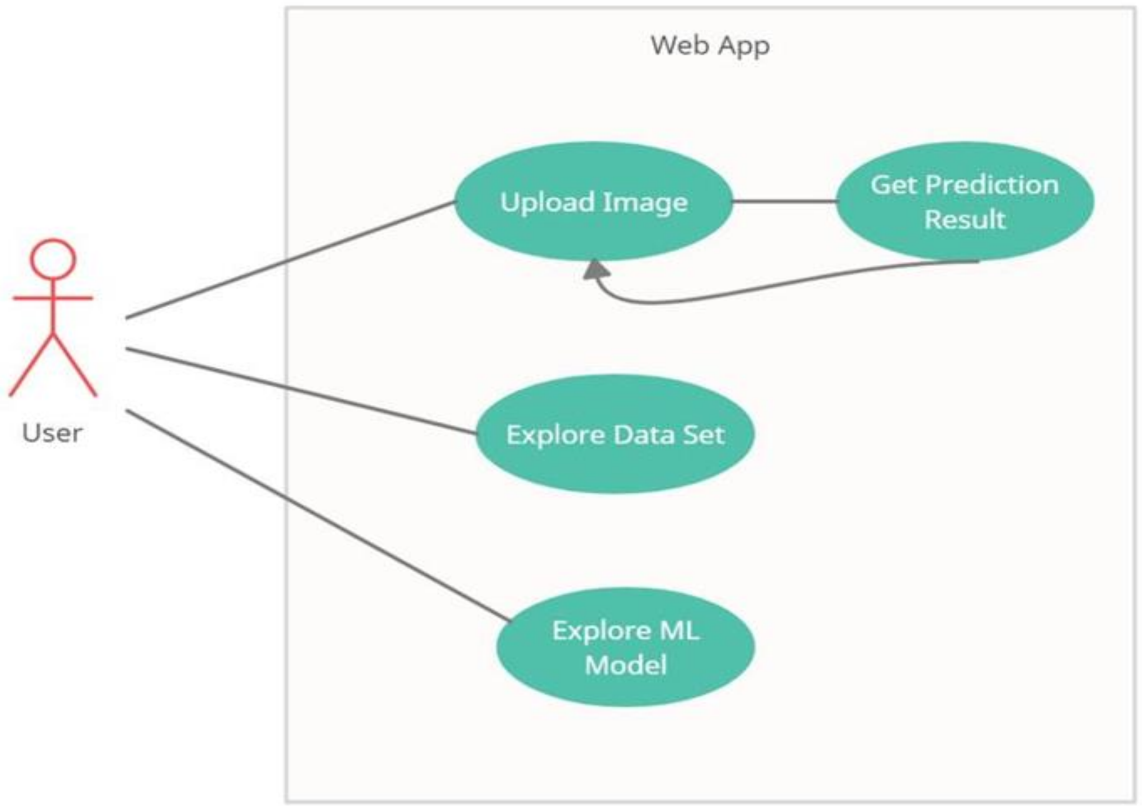
5.1) Block Diagram;

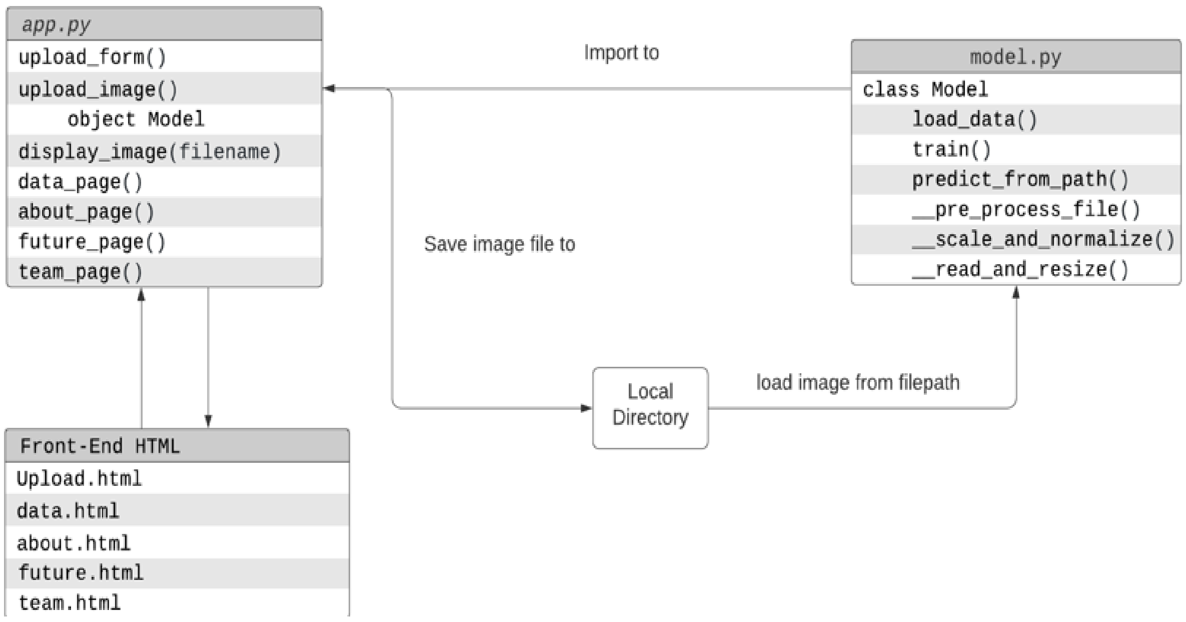


5.2) Flow Diagram:



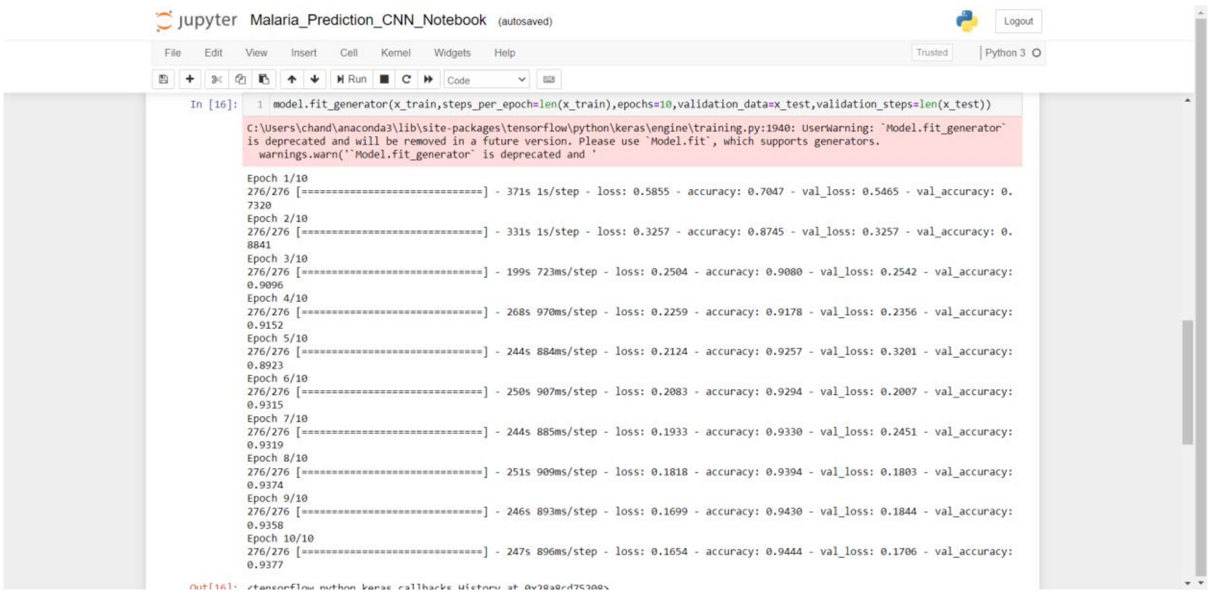
5.3) Use case Diagram :



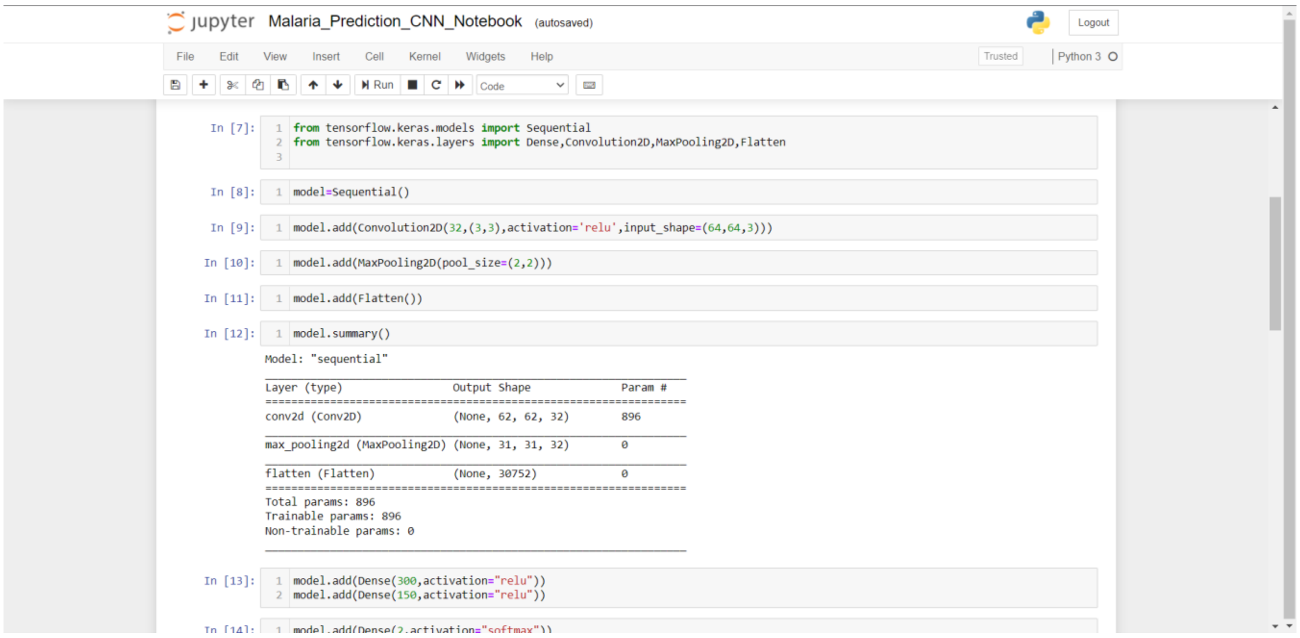


6) RESULT :

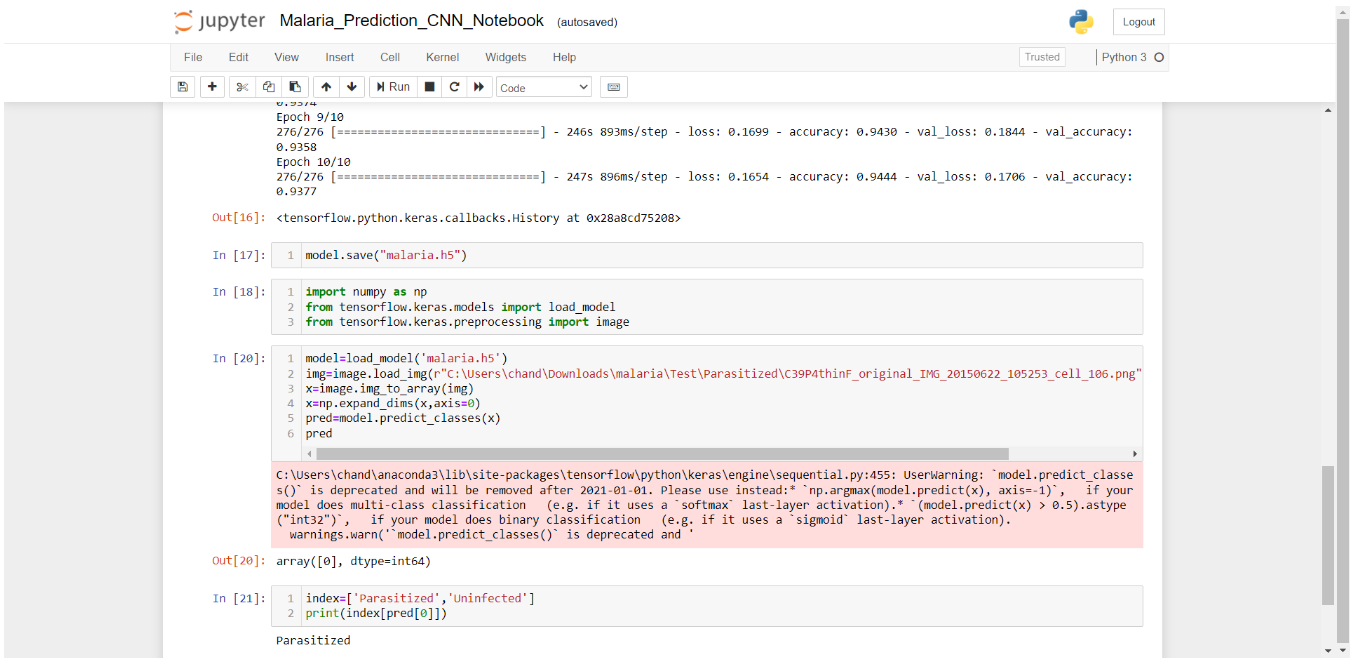
Train &Test the Data



```
In [16]:  model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=10,validation_data=x_test,validation_steps=len(x_test))

C:\Users\chand\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1940: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/10
276/276 [==============================] - 371s 1s/step - loss: 0.5855 - accuracy: 0.7047 - val_loss: 0.5465 - val_accuracy: 0.
7320
Epoch 2/10
276/276 [==============================] - 331s 1s/step - loss: 0.3257 - accuracy: 0.8745 - val_loss: 0.3257 - val_accuracy: 0.
8841
Epoch 3/10
276/276 [==============================] - 199s 723ms/step - loss: 0.2504 - accuracy: 0.9080 - val_loss: 0.2542 - val_accuracy:
0.9096
Epoch 4/10
276/276 [==============================] - 268s 970ms/step - loss: 0.2259 - accuracy: 0.9178 - val_loss: 0.2356 - val_accuracy:
0.9152
Epoch 5/10
276/276 [==============================] - 244s 884ms/step - loss: 0.2124 - accuracy: 0.9257 - val_loss: 0.3201 - val_accuracy:
0.8923
Epoch 6/10
276/276 [==============================] - 250s 907ms/step - loss: 0.2083 - accuracy: 0.9294 - val_loss: 0.2007 - val_accuracy:
0.9315
Epoch 7/10
276/276 [==============================] - 244s 885ms/step - loss: 0.1933 - accuracy: 0.9330 - val_loss: 0.2451 - val_accuracy:
0.9319
Epoch 8/10
276/276 [==============================] - 251s 909ms/step - loss: 0.1818 - accuracy: 0.9394 - val_loss: 0.1803 - val_accuracy:
0.9374
Epoch 9/10
276/276 [==============================] - 246s 893ms/step - loss: 0.1699 - accuracy: 0.9430 - val_loss: 0.1844 - val_accuracy:
0.9358
Epoch 10/10
276/276 [==============================] - 247s 896ms/step - loss: 0.1654 - accuracy: 0.9444 - val_loss: 0.1706 - val_accuracy:
0.9377

Out[16]: <tensorflow.python.keras.callbacks.History at 0x28a8cd75208>
```

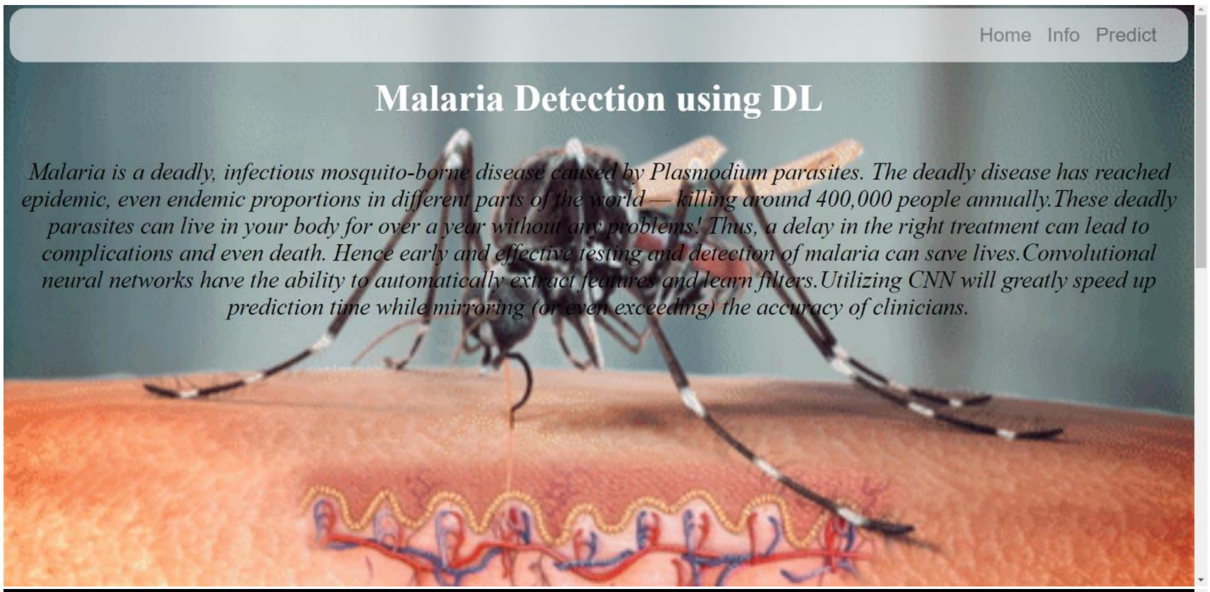

jupyter Malaria_Prediction_CNN_Notebook (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [7]:  from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense,Convolution2D,MaxPooling2D,Flatten

In [8]:  model=Sequential()

In [9]:  model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))

In [10]: model.add(MaxPooling2D(pool_size=(2,2)))

In [11]: model.add(Flatten())

In [12]: model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         conv2d (Conv2D)              (None, 62, 62, 32)        896
         _________________________________________________________________
         max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
         _________________________________________________________________
         flatten (Flatten)            (None, 30752)             0
         =================================================================
         Total params: 896
         Trainable params: 896
         Non-trainable params: 0
         _________________________________________________________________

In [13]: model.add(Dense(300,activation="relu"))
         model.add(Dense(150,activation="relu"))

In [14]: model.add(Dense(2,activation="softmax"))
```


jupyter Malaria_Prediction_CNN_Notebook (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

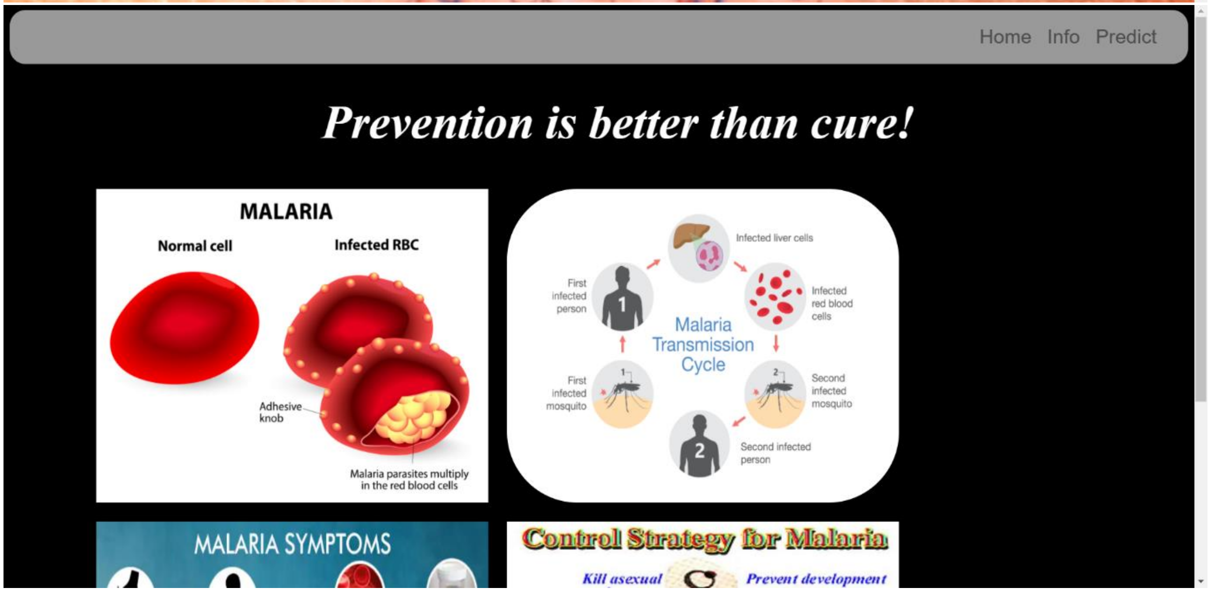
```
         0.9374
         Epoch 9/10
         276/276 [==============================] - 246s 893ms/step - loss: 0.1699 - accuracy: 0.9430 - val_loss: 0.1844 - val_accuracy:
         0.9358
         Epoch 10/10
         276/276 [==============================] - 247s 896ms/step - loss: 0.1654 - accuracy: 0.9444 - val_loss: 0.1706 - val_accuracy:
         0.9377

Out[16]: <tensorflow.python.keras.callbacks.History at 0x28a8cd75208>

In [17]: model.save("malaria.h5")

In [18]: import numpy as np
         from tensorflow.keras.models import load_model
         from tensorflow.keras.preprocessing import image

In [20]: model=load_model('malaria.h5')
         img=image.load_img(r"C:\Users\chand\Downloads\malaria\Test\Parasitized\C39P4thinF_original_IMG_20150622_105253_cell_106.png"
         x=image.img_to_array(img)
         x=np.expand_dims(x,axis=0)
         pred=model.predict_classes(x)
         pred

         C:\Users\chand\anaconda3\lib\site-packages\tensorflow\python\keras\engine\sequential.py:455: UserWarning: `model.predict_classe
         s()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your
         model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype
         ("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
           warnings.warn('`model.predict_classes()` is deprecated and '

Out[20]: array([0], dtype=int64)

In [21]: index=['Parasitized','Uninfected']
         print(index[pred[0]])

         Parasitized
```

Flask Implementation :

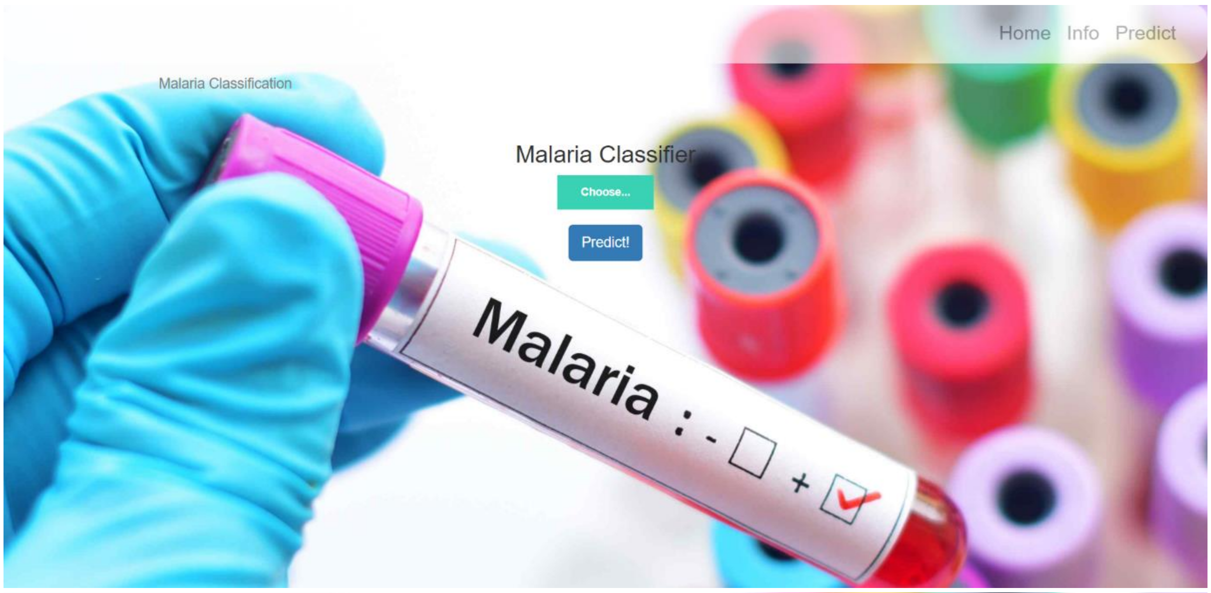


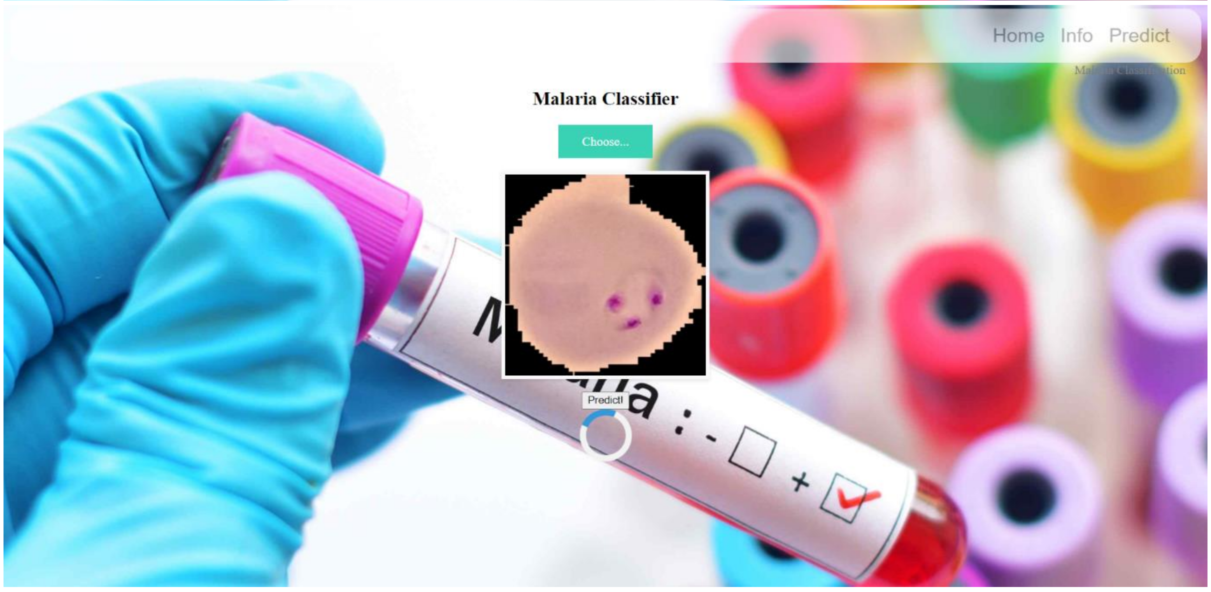
Malaria Detection using DL

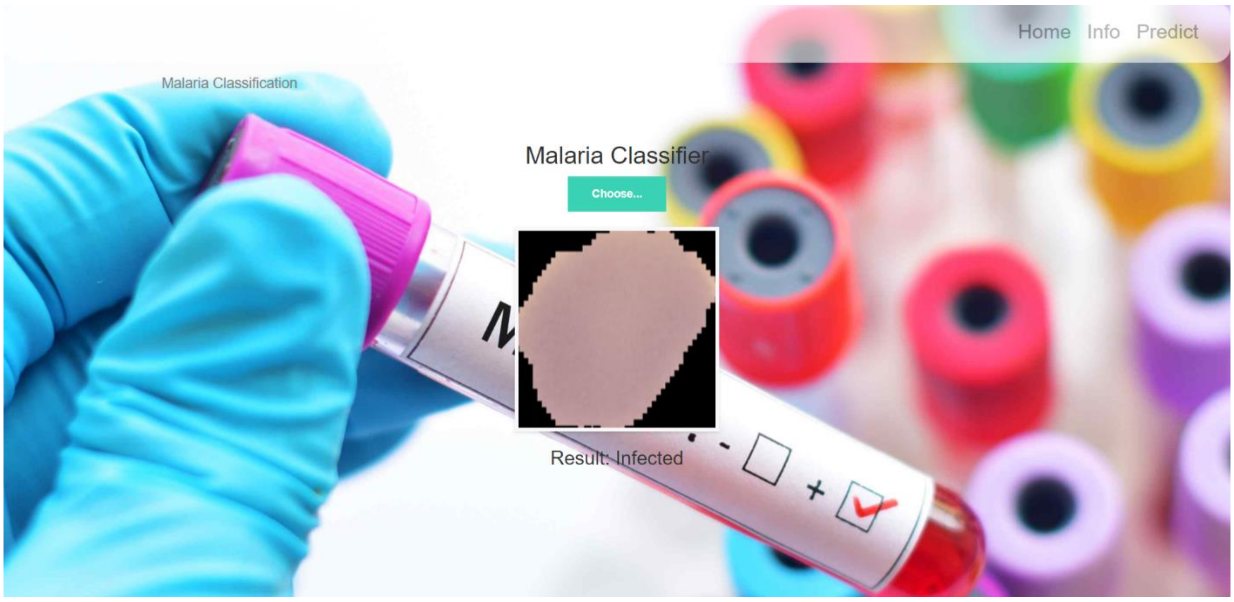
Malaria is a deadly, infectious mosquito-borne disease caused by Plasmodium parasites. The deadly disease has reached epidemic, even endemic proportions in different parts of the world — killing around 400,000 people annually.These deadly parasites can live in your body for over a year without any problems! Thus, a delay in the right treatment can lead to complications and even death. Hence early and effective testing and detection of malaria can save lives.Convolutional neural networks have the ability to automatically extract features and learn filters.Utilizing CNN will greatly speed up prediction time while mirroring (or even exceeding) the accuracy of clinicians.



Prevention is better than cure!







Result: Infected

7) ADVANTAGES & DISADVANTAGES

Advantages :

- Simple and computationally fast
- Unsupervised always converge the boundarious of Maleria infected or not
- Topological changes are naturally possible

Disadvantages :

- Limited applicability to enhancing Maleria
- Long computational time, sensitive to noice
- Topological changes are computationally expensive

8) APPLICATIONS :

- The main aim of the applications is Maleria identification.
- The main reason behind the development of this application is to provide proper treatment as soon as possible and protect the human life which is in danger.
- This application is helpful to doctors as well as patient.
- The manual identification is not so fast, more accurate and efficient for user. To overcome those problem this application is design.
- It is user friendly application.

9) CONCLUSION :

We proposed a computerized method for the segmentation and identification of a brain tumor using the IBM Watson studio. The input scanned images are tested and trained by image pre-processing .These images are pre-processed using an adaptive modelling technique such as sequential, convolution, pooling, flattening for the elimination of noises that are present inside the original image and then we obtained a '.h5' file which we used in flask to create a web application inorder to detect malaria is present or not. The proposed model had obtained an accuracy of 84% and yields promising results without any errors and much less computational time.

10) FUTURE SCOPE :

It is observed on extermination that the proposed approach needs a vast training set for better accurate results; in the field of image processing, the gathering of dataset is a tedious job, and, in few cases, the datasets might not be available. In all such cases, the proposed algorithm must be robust enough for accurate recognition of scanned area of the Images. The proposed approach can be further improvised through in cooperating weakly trained algorithms that can identify the abnormalities with a minimum training data and also self-learning algorithms would aid in enhancing the accuracy of the algorithm and reduce the computational time

11) Source code :

```
import os

import numpy as np #used for numerical analysis

from flask import Flask,request,render_template # Flask-It is our framework which we are going to use to run/serve our application.

#request-for accessing file which was uploaded by the user on our application.

#render_template- used for rendering the html pages

from tensorflow.keras.models import load_model #to load our trained model

from tensorflow.keras.preprocessing import image




app=Flask(__name__)#our flask app

model=load_model(r'C:\Users\chand\Downloads\malaria.h5')#loading the model


@app.route("/") #default route

def about():

    return render_template("about.html")#rendering html page


@app.route("/about") #route about page

def home():

    return render_template("about.html")#rendering html page


@app.route("/info") # route for info page

def information():

    return render_template("info.html")#rendering html page


@app.route("/upload") # route for uploads

def test():

    return render_template("index6.html")#rendering html page
```

```
@app.route("/predict",methods=["GET","POST"]) #route for our prediction
def upload():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        print(f)
        basepath=os.path.dirname('__file__')#storing the file directory
        print(basepath)
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file
        print(filepath)

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)
        x=np.expand_dims(x,axis=0)
        pred=model.predict_classes(x)
        print("prediction",pred)#printing the prediction
        if(pred==1):#checking the results
            result="Uninfected"
        else:
            result="Infected"
        return result#resturing the result
    return None

#port = int(os.getenv("PORT"))
if __name__=="__main__":
    app.run(debug=False)#running our app
    #app.run(host='0.0.0.0', port=8000,debug=False)
```