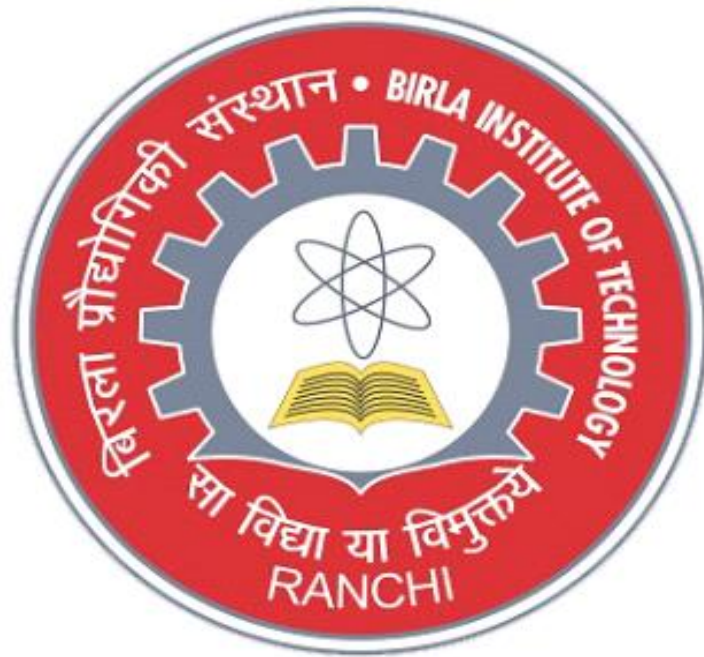


BIRLA INSTITUTE OF TECHNOLOGY,  
MESRA



MALARIA DETECTION  
MOOC-PROJECT

SUBMITTED TO:

RASHMI RATHI UPADHYAY  
(ASST. PROFESSOR)

SUBMITTED BY:

JAYDEEP KUMAR  
SILAWAT  
( MCA/10051/19)

# Abstract

My project is on detection of the Malaria Disease. Malaria is a blood disease caused by the Plasmodium parasites transmitted through the bite of female Anopheles mosquito. Microscopists commonly examine thick and thin blood smears to diagnose disease and compute parasitemia. However, their accuracy depends on smear quality and expertise in classifying and counting parasitized and uninfected cells. Such an examination could be arduous for large-scale diagnoses resulting in poor quality. State-of-the-art image-analysis based on computerised diagnosis methods using machine learning (ML) techniques. Convolutional Neural Networks (CNN), a class of deep learning (DL) models promise highly scalable and superior results with end-to-end feature extraction and classification. Automated malaria screening using DL techniques could, therefore, serve as an effective diagnostic aid. In this study, we evaluate the performance of pre-trained CNN based DL models as feature extractors toward classifying parasitized and uninfected cells to aid in improved disease screening. We experimentally determine the optimal model layers for feature extraction from the underlying data. Statistical validation of the results demonstrates the use of pre-trained CNNs as a promising tool for feature extraction for this purpose.

Our dataset comes from National Institutes of Health.

Downloaded the dataset from : Kaggle Malaria Cell Images Dataset  
<https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria>

Studies on improving the efficiency of malaria infection diagnosis have been started years ago, and those studies have experimented methodologies to generate data from the blood sample. From the review written by M. Poostchi et.al(2018), we read a comprehensive list of different approaches on data processing. From blood sample to image data, there are thick and thin smear, different staining, different microscopy including light microscopy, fluorescent microscopy and quantitative phase imaging. And to process the image data, methods like noise reduction, low image contrast, and uneven illumination have been applied. After that, the one image per blood cell is segmented. In our study, the NIH dataset images are generated from Giemsa-stained thin blood smear slides with level-set based algorithm to detect, and segment the red blood cells.

In this project, we implemented a customized convolutional neural network model and conducted several experiments of the latest CNN models on the problem of malaria cell image recognition. These models have various features, leading to different performance. The very-deep feature indicates the higher probability of the problem of overfitting, which we encountered during our project. By applying advanced machine learning technique, the process of detecting and diagnosing malaria disease could be accelerated and boosted, which could be significant for poor areas.

If we had more time for this project, there are many things we could still do to improve the performance and result. First thing we can do is we can adjust the image contrast or convert it to a grayscale image. Because there isn't much information in colour. But we are more interested in patterns. And also we can apply blur to smoothen the image for better result.

The major problem we encountered was the overfitting when training the deep networks, we would like to tackle this problem by doing more experiments. First of all, we would try different optimizer, learning rate, and loss function to see how the overfitting could be possibly affected. Secondly, we would apply more data augmentation techniques to enlarge the dataset because the larger data could reduce the fact of overfitting to some extent. Last but not least, finer regularization, as well as the dropout, should be done considering the current model.

## Introduction

Malaria, a mosquito borne life-threatening disease, causes fever, vomiting, headaches and fatigue; in severe cases it can cause coma or even death. The disease can affect humans or other animals. The disease is commonly transmitted by female anopheles' mosquitoes. The mosquito bite injects the parasite into affected person's blood which then travels to the liver to mature and reproduce. Malaria is caused by a single cell microorganism belonging to the genus Plasmodium where five of their species can infect humans. *P. falciparum* is the deadliest among these species; others include *P. vivax*, *P. ovale*, *P. knowlesi* and *P. malariae*.

Malaria is widespread in tropical and subtropical regions, especially in Latin America, Sub Saharan Africa and Asia. The disease is considered endemic in 13 districts out of 64 in Bangladesh, putting about 14 million people at risk. Among the five species, the most dominant parasite in Bangladesh is *P. falciparum*. In 2016, approximately 731,000 deaths have been reported around the world due to malaria with 90% of them in Africa. Malaria is commonly diagnosed by microscopic examination of blood cells using blood films. Approximately, 167 million blood films had been tested for malaria during 2010 using microscopy, which was less costly and less complex than polymerase chain reaction-based diagnosis. Although it is widely used, microscopic diagnosis has many drawbacks as follows. As malaria is generally associated with poverty and occurs mostly in low economic countries, most laboratories or diagnostic facilities are not equipped with standard testing facilities. Moreover, the diagnosis depends on the skill of the person examining the blood film and level of parasites present thereon. Additionally, the monotonicity of the examination hugely affects the quality of examination, towards the end of a batch especially if the batch has many specimens. The global shortage of pathologist in general has a serious impact on health care system of developing countries and the case of malaria is no exception. Due to the lack of reliable diagnostic facilities, many Bangladeshi citizens opt for treatment overseas which unfortunately is not economically feasible for majority of the people. Since medical images/datasets are usually smaller in size and thus often termed inadequate for learning, the power of transfer learning has also been leveraged in the literature. In the literature Var et al. and Rajaraman et al. proposed methods for computer aided diagnosis based on pre-trained convolutional neural networks as feature extractors to identify malaria parasites. Classical machine learning algorithms have also performed well in this classification task. As shown by Das et al. and Park et al. , Bayesian learning, support vector machines, logistic regression and k-nearest neighbour algorithm perform well in this context. In addition, attempts have been made to remove the stains from the peripheral blood smear images as well as for impulse noise reduction. Mustafa et. al have proposed a pre-processing step, thresholding, which is considered to be one of the most important pre-processing steps in this task. In their work, comparison among Fuzzy C-Mean algorithm, Wolf's method , Bradley's method and Bernsen's method has been shown. Each of these methods is experimented with malaria parasite images.

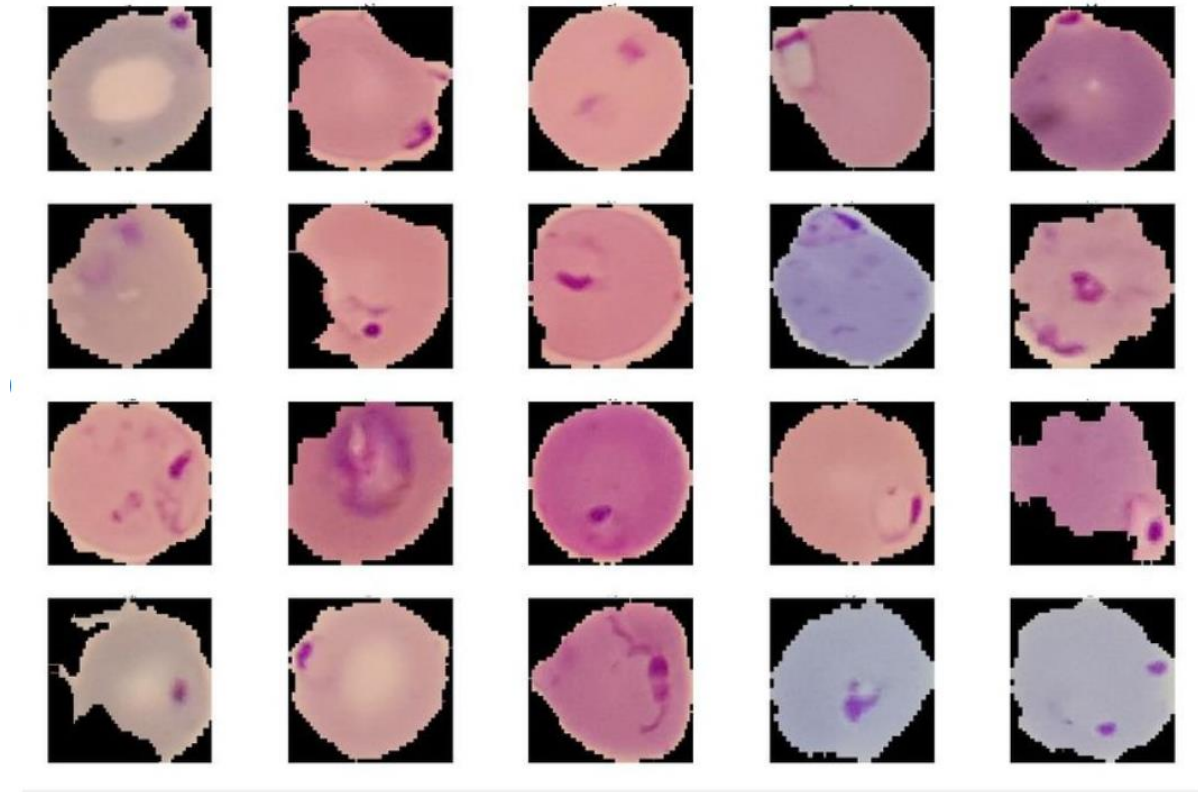
## NIH Malaria Dataset

The dataset used in this work is taken from National Institute of Health. It consists of segmented cells from the thin blood smear slide images from the Malaria Screener research activity. Giemsa-stained thin blood smear slides from 150 *P. falciparum*-infected and 50 healthy patients were collected and photographed at Chittagong Medical College Hospital, Bangladesh. The images were manually annotated by an expert slide reader at the Mahidol-Oxford Tropical Medicine Research Unit. The dataset contains 27,558 segmented cell images, with equal instances of 13,779 parasitized and 13,779 uninfected segmented red blood cell images. Positive samples contained plasmodium and negative samples contained no plasmodium but could contain other types of objects including staining artifacts/impurities. The patches of segmented red blood cells are of 3-channels (RGB) with size variation of 110-150 pixels which have been later re-sampled to 200 x 200 output dimension, a channel depth of 3 and 32-bit floating point precision (FP32) to suit the input requirements of different classification algorithms used in this work. Different pre-processing techniques are also applied to achieve faster convergence which will be discussed later in detail. Below figures show some samples from the dataset containing uninfected and parasitized segmented red blood cells respectively.



*Samples drawn from NIH Malaria dataset which are uninfected red blood cells.*

Image Ref.- [https://www.researchgate.net/figure/Samples-drawn-from-NIH-Malaria-dataset-which-are-uninfected-red-blood-cells-It-is-seen\\_fig1\\_334669002](https://www.researchgate.net/figure/Samples-drawn-from-NIH-Malaria-dataset-which-are-uninfected-red-blood-cells-It-is-seen_fig1_334669002).



*Samples drawn from NIH Malaria dataset which are malaria infected parasite red blood cells.*

Image Ref.- [https://www.researchgate.net/figure/Samples-drawn-from-NIH-Malaria-dataset-which-are-malaria-infected-parasite-red-blood\\_fig2\\_334669002](https://www.researchgate.net/figure/Samples-drawn-from-NIH-Malaria-dataset-which-are-malaria-infected-parasite-red-blood_fig2_334669002)

## Experimental Setup

All of these experiments were performed on a machine with:

Windows system Microsoft Windows 10 Home Single Language with Intel® Core(TM) i5-8265U CPU @ 1.60GHz processor, 1 TB HDD, 8 GB RAM, Python® 3.9, Keras® 2.4.0 with TensorFlow® 2.4.1, Jupyter notebook (anocanda3), Numpy 1.19.2 .

## Related Work:

Studies on improving the efficiency of malaria infection diagnosis have been started years ago, and those studies have experimented methodologies to generate data from the blood sample. From the review written by M. Poostchi et.al(2018), we read a comprehensive list of different approaches on data processing. From blood sample to image data, there are thick and thin smear, different staining, different microscopy including light microscopy, fluorescent microscopy and quantitative phase imaging. And to process the image data, methods like noise reduction, low image contrast, and uneven illumination have been applied. After that, the one image per blood cell is segmented. In our study, the NIH dataset images are generated from Giemsa-stained thin blood smear slides with level-set based algorithm to detect, and segment the red blood cells. Previous automated diagnosis has focused on traditional classification methods. In a study conducted by Y. Purwar et. al in 2011, they used k-means clustering and reached between 50 and 88 percent recall on over 150 samples; and from another study by D. Anggraini et. al in the same year, a classification model based on Bayes Decision Theory has reached a 92.59 recall and F-1 score of 0.78. The difference of results between these two studies suggests that supervised learning is the more robust one.

We also learned the paper Transfer Learning with ResNet50 for that Malaria Cell-Image Classification by S.B Reddy and D.S Juliet, which we later decided the convolution neural networks that we are going to compare. In this paper, the authors mentioned the gradient vanishing problem caused when choosing Sigmoid Function as the activation function in Neural Networks with deep depth. This problem is alleviated by the introduction of ResNet construction, which used the residual in studying, which we later examined its advantages in our study.

In the same paper, it also proposed their method of using pre-trained weights before the fully connected layers. S.B Reddy and D.S Juliet proposed that they used a ResNet 50 layer with pre-trained weights for the ResNet50 layer to perform feature extraction. Using pre-trained neural network before the fully connected layers instead is a common method applied in building Convolution Neural Network models, in this project, we adopted this idea and build three models by applying a fully connected layer in Keras, ResNet and VGG19 in the front, respectively

Model:

## **1. CNN :**

Convolutional Neural Network, known as CNN, is the most typical method of deep learning. In 1980, the introduction of the neural sensor neocognitron marked the birth of the first initial convolutional neural network. It was also the first application of the receptive field concept in the field of artificial neural networks. The neural cognitive machine decomposed a visual pattern. A number of sub-patterns (features) are then entered into the layered hierarchically connected feature plane for processing. Then in 1988, the Shift-invariant neural network proposed to improve the function of the CNN, enabling it to be recognized even when the object is displaced or slightly deformed. Feed-forward convolutional neural network architecture is extended laterally and is connected in a feedback neural abstract pyramid (Neural abstraction pyramid) in. The resulting recurring convolution network allows for flexible incorporation of context information to iteratively resolve local ambiguities. In contrast to previous models, the highest resolution image output is produced. Finally, in 2005, a paper with GPU implementation of CNN appeared, which marked a more effective way to implement CNN. After the 2012 ImageNet contest, CNN stood out because of its high precision.

## **2. ResNet**

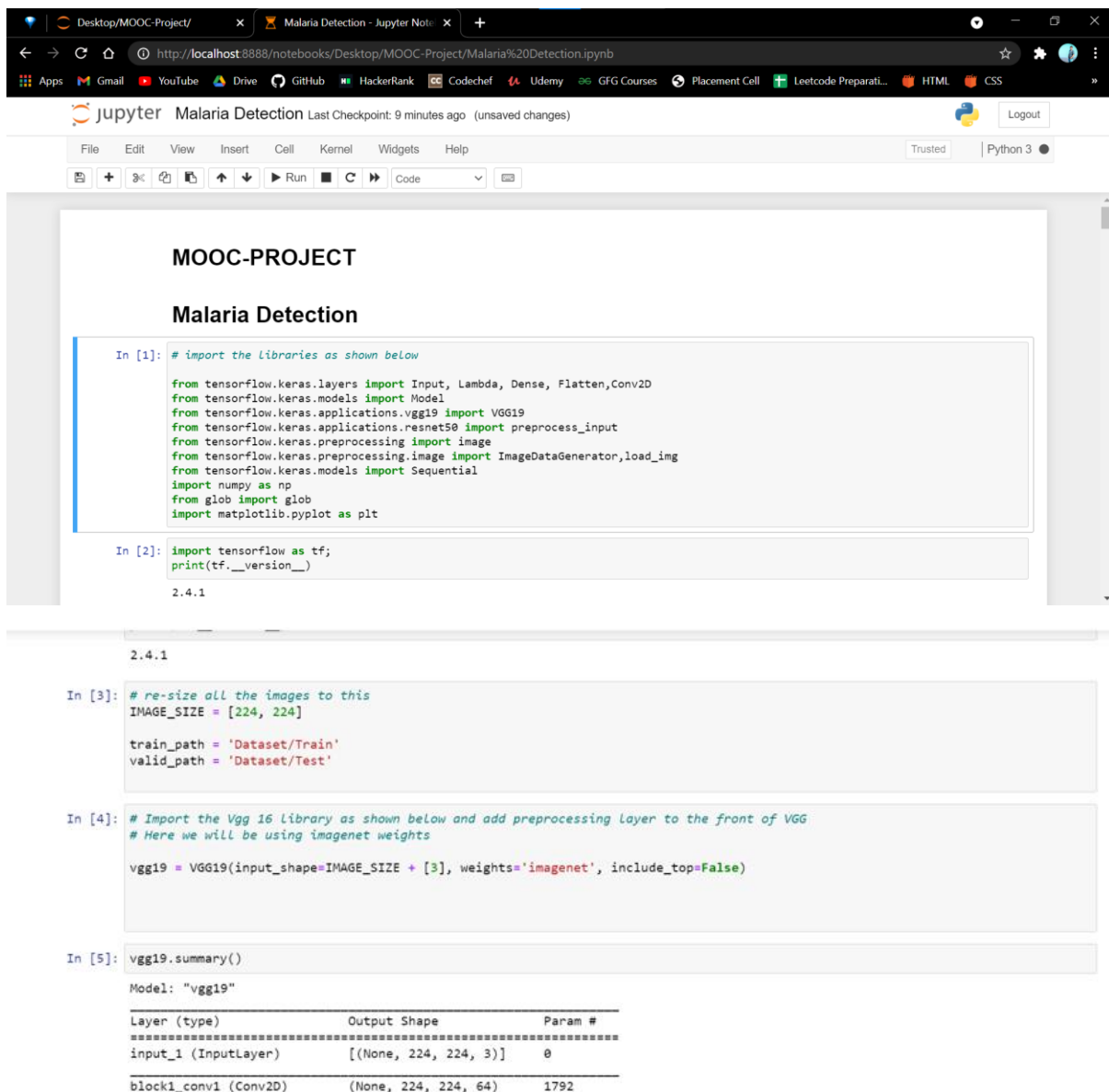
ResNet (Residual Neural Network) was proposed by four Chinese scientists leading by Kaiming He from Microsoft Research Institute. Through the use of ResNet Unit, they successfully trained the 152-layer neural network and won the championship in ILSVRC2015. The error rate on top5 was 3.57%. At the same time, the parameter mation transmission, which also causes gradients to disappear or gradient explosions. These make it difficult to train deep network. ResNet solves this problem to a certain extent. By directly bypassing the input information to the output and protecting the integrity of the information, the entire network only needs to learn the part of the input and output differences, simplifying the learning objectives and difficulty.



### 3. VGG19

VGG19 is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). There are other variants of VGG like VGG11, VGG16 and others. VGG19 has 19.6 billion FLOPs.

Project Code Snapshots:--



The screenshot shows a Jupyter Notebook interface with the following content:

## MOOC-PROJECT

### Malaria Detection

```
In [1]: # import the libraries as shown below

from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

In [2]: import tensorflow as tf;
print(tf.__version__)

2.4.1

2.4.1

In [3]: # re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'Dataset/Train'
valid_path = 'Dataset/Test'

In [4]: # Import the Vgg 16 Library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

vgg19 = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

In [5]: vgg19.summary()

Model: "vgg19"

Layer (type)           Output Shape          Param #
=====
input_1 (InputLayer)    [(None, 224, 224, 3)] 0
block1_conv1 (Conv2D)   (None, 224, 224, 64)  1792
```

block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080

block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
=====		
Total params: 20,024,384		
Trainable params: 20,024,384		
Non-trainable params: 0		

```
In [6]: # don't train existing weights
for layer in vgg19.layers:
    layer.trainable = False
```

```
In [7]: # useful for getting number of output classes
folders = glob('Dataset/Train/*')
```

```
In [8]: folders
```

```
Out[8]: ['Dataset/Train\\Parasite', 'Dataset/Train\\Uninfected']
```

```
In [9]: # our layers - you can add more if you want
x = Flatten()(vgg19.output)
```

```
In [10]: prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vgg19.input, outputs=prediction)
```

```
In [11]: # view the structure of the model
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080

block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 2)	50178
Total params: 20,074,562		
Total params: 20,074,562		
Trainable params: 50,178		
Non-trainable params: 20,024,384		

```
In [12]: from tensorflow.keras.layers import MaxPooling2D
```

```
In [13]: ### Create Model from scratch using CNN
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
model.add(Dense(2,activation="softmax"))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 224, 224, 16)	208
=====		
max_pooling2d (MaxPooling2D)	(None, 112, 112, 16)	0
=====		
conv2d_1 (Conv2D)	(None, 112, 112, 32)	2080
=====		
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 32)	0
=====		
conv2d_2 (Conv2D)	(None, 56, 56, 64)	8256
=====		
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 64)	0
=====		
flatten_1 (Flatten)	(None, 50176)	0
=====		
dense_1 (Dense)	(None, 500)	25088500
=====		
dense_2 (Dense)	(None, 2)	1002
=====		
Total params: 25,100,046		
Trainable params: 25,100,046		
Non-trainable params: 0		

```
In [14]: # tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

```
In [15]: # Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
In [16]: training_set = train_datagen.flow_from_directory('Dataset/Train',
                                                         target_size = (224, 224),
                                                         batch_size = 32,
                                                         class_mode = 'categorical')
```

Found 416 images belonging to 2 classes.

```
In [17]: training_set
Out[17]: <tensorflow.python.keras.preprocessing.image.DirectoryIterator at 0x2a80003700>
```

```
In [18]: test_set = test_datagen.flow_from_directory('Dataset/Test',
                                                    target_size = (224, 224),
                                                    batch_size = 32,
                                                    class_mode = 'categorical')
```

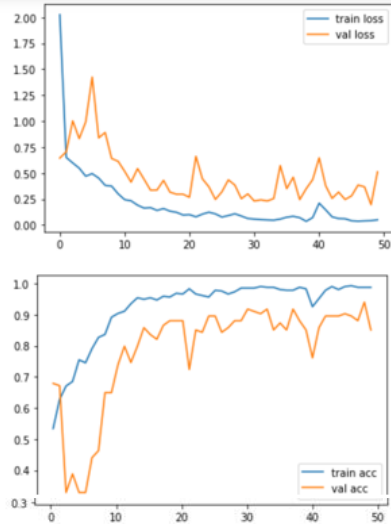
Found 134 images belonging to 2 classes.

```
In [*]: # fit the model
# Run the cell. It will take some time to execute
r = model.fit(
    training_set,
    validation_data=test_set,
    epochs=50,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)
```

```
Epoch 1/50
13/13 [=====] - 12s 856ms/step - loss: 3.2716 - accuracy: 0.5306 - val_loss: 0.6447 - val_accuracy:
0.6791
Epoch 2/50
13/13 [=====] - 11s 871ms/step - loss: 0.6640 - accuracy: 0.5910 - val_loss: 0.7050 - val_accuracy:
0.6716
Epoch 3/50
13/13 [=====] - 12s 879ms/step - loss: 0.5996 - accuracy: 0.6670 - val_loss: 1.0028 - val_accuracy:
0.3284
Epoch 4/50
13/13 [=====] - 12s 936ms/step - loss: 0.5451 - accuracy: 0.6819 - val_loss: 0.8319 - val_accuracy:
0.3881
Epoch 5/50
13/13 [=====] - 13s 986ms/step - loss: 0.4891 - accuracy: 0.7431 - val_loss: 0.9921 - val_accuracy:
0.3284
Epoch 6/50
13/13 [=====] - 12s 934ms/step - loss: 0.4585 - accuracy: 0.7604 - val_loss: 1.4238 - val_accuracy:
0.3284
Epoch 7/50
13/13 [=====] - 11s 878ms/step - loss: 0.4950 - accuracy: 0.7559 - val_loss: 0.8386 - val_accuracy:
0.8955
Epoch 45/50
13/13 [=====] - 12s 925ms/step - loss: 0.0596 - accuracy: 0.9807 - val_loss: 0.2460 - val_accuracy:
0.8955
Epoch 46/50
13/13 [=====] - 12s 908ms/step - loss: 0.0551 - accuracy: 0.9876 - val_loss: 0.2808 - val_accuracy:
0.9030
Epoch 47/50
13/13 [=====] - 12s 890ms/step - loss: 0.0188 - accuracy: 0.9975 - val_loss: 0.3852 - val_accuracy:
0.8955
Epoch 48/50
13/13 [=====] - 12s 880ms/step - loss: 0.0407 - accuracy: 0.9857 - val_loss: 0.3676 - val_accuracy:
0.8806
Epoch 49/50
13/13 [=====] - 11s 857ms/step - loss: 0.0340 - accuracy: 0.9921 - val_loss: 0.1960 - val_accuracy:
0.9403
Epoch 50/50
13/13 [=====] - 11s 831ms/step - loss: 0.0395 - accuracy: 0.9928 - val_loss: 0.5124 - val_accuracy:
0.8507
```

```
In [20]: # plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```



<Figure size 432x288 with 0 Axes>

In [21]: # save it as a h5 file

```
from tensorflow.keras.models import load_model
model.save('model_vgg19.h5')
```

In [ ]:

In [22]: y\_pred = model.predict(test\_set)

In [23]: y\_pred

```
Out[23]: array([[3.50142643e-03, 9.96498585e-01],
 [2.88912188e-03, 9.97110844e-01],
 [9.99988080e-01, 1.17332445e-06],
 [9.99080777e-01, 9.19266371e-04],
 [3.27414786e-03, 9.96725798e-01],
 [9.99822676e-01, 1.77376729e-04],
 [1.00000000e+00, 3.05488729e-12],
 [9.44138885e-01, 5.58611564e-02],
 [9.99558508e-01, 4.41466633e-04],
 [7.72920903e-03, 9.92270827e-01],
 [1.00000000e+00, 3.36962347e-09],
 [4.60742936e-02, 9.53925729e-01],
 [1.00000000e+00, 6.05053829e-09],
 [1.00000000e+00, 3.31007574e-24],
 [1.00000000e+00, 2.84664310e-08],
 [5.41792251e-03, 9.94582117e-01],
 [1.00000000e+00, 3.21605241e-17],
 [1.79749951e-01, 8.20250094e-01],
 [1.67329970e-03, 9.98326719e-01],
 [7.73992315e-02, 9.22600746e-01],
```

In [24]: import numpy as np  
y\_pred = np.argmax(y\_pred, axis=1)

In [25]: y\_pred

```
Out[25]: array([1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
 1, 1], dtype=int64)
```

In [ ]:

In [26]: from tensorflow.keras.models import load\_model  
from tensorflow.keras.preprocessing import image

In [27]: model=load\_model('model\_vgg19.h5')

In [ ]:

In [28]: img=image.load\_img('Dataset/Test/Uninfected/2.png',target\_size=(224,224))

```
In [29]: x=image.img_to_array(img)
x
```

```
Out[29]: array([[0., 0., 0.],
               [0., 0., 0.],
               [0., 0., 0.],
               ...,
               [0., 0., 0.],
               [0., 0., 0.],
               [0., 0., 0.]],

          [[0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.],
           ...,
           [0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.]],

          [[0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.],
           ...,
           [0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.]],

          [[0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.],
           ...,
           [0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.]],

          [[0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.],
           ...,
           [0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.]],

          [[0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.],
           ...,
           [0., 0., 0.],
           [0., 0., 0.],
           [0., 0., 0.]]], dtype=float32)
```

```
In [30]: x.shape
```

```
Out[30]: (224, 224, 3)
```

```
In [31]: x=x/255
```

```
In [32]: x=np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
img_data.shape
```

```
Out[32]: (1, 224, 224, 3)
```

```
In [33]: model.predict(img_data)
```

```
Out[33]: array([[0., 1.]], dtype=float32)
```

```
In [34]: a=np.argmax(model.predict(img_data), axis=1)
```

```
In [35]: if(a==1):
           print("Uninfected")
           else:
           print("Infected")
```

```
Uninfected
```

## References

1. World Health Organization. *Malaria: fact sheet*. No. WHO-EM/MAC/035/E. World Health Organization. Regional Office for the Eastern Mediterranean, 2014.  
<https://apps.who.int/iris/handle/10665/204183>
2. Wilson, Michael L. "Malaria rapid diagnostic tests." *Clinical Infectious Diseases*, Volume 54, Issue 11, 1 June 2012, Pages 1637-1641, <https://doi.org/10.1093/cid/cis228>
3. Kaggle: Malaria Cell Images Dataset <https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria>
4. Centers for Disease Control and Prevention. 2012. CDC-Malaria. Available at <http://www.cdc.gov/malaria/about/biology/>.
5. Chollet F. Deep learning models. Available at <https://github.com/fchollet/deep-learning-models>.
6. Poostchi M, Silamut K, Maude RJ, Jaeger S, Thoma GR. 2018. Image analysis and machine learning for detecting malaria. <https://doi.org/10.1016/j.trsl.2017.12.004>.
7. NPTEL - Machine Learning, ML by Prof. Carl Gustaf Jansson KTH.  
[https://onlinecourses.nptel.ac.in/noc21\\_cs51/preview](https://onlinecourses.nptel.ac.in/noc21_cs51/preview).
8. WHO. 2016. World malaria report. Available at <http://apps.who.int/iris/bitstream/10665/252038/1/9789241511711-eng.pdf?ua=1>