

INTRODUCTION:

Artificial intelligence (AI) and open source tools, technologies, and frameworks are a powerful combination for improving society. "*Health is wealth*" is perhaps a cliché, yet it's very accurate! In this article, we will examine how AI can be leveraged for detecting the deadly disease malaria with a low-cost, effective, and accurate open source deep learning solution.

While I am neither a doctor nor a healthcare researcher and I'm nowhere near as qualified as they are, I am interested in applying AI to healthcare research. My intent in this article is to showcase how AI and open source solutions can help malaria detection and reduce manual labor.

MOTIVATION:

Malaria is a deadly, infectious, mosquito-borne disease caused by *Plasmodium* parasites that are transmitted by the bites of infected female *Anopheles* mosquitoes. There are five parasites that cause malaria, but two types—*P. falciparum* and *P. vivax*—cause the majority of the cases.

If an infected mosquito bites you, parasites carried by the mosquito enter your blood and start destroying oxygen-carrying red blood cells (RBC). Typically, the first symptoms of malaria are similar to a virus like the flu and they usually begin within a few days or weeks after the mosquito bite. However, these deadly parasites can live in your body for over a year without causing symptoms, and a delay in treatment can lead to complications and even death. Therefore, early detection can save lives.

The World Health Organization's (WHO) [malaria facts](#) indicate that nearly half the world's population is at risk from malaria, and there are over 200 million malaria cases and approximately 400,000 deaths due to malaria every year. This is a motivation to make malaria detection and diagnosis fast, easy, and effective.

METHODS:

There are several methods that can be used for malaria detection and diagnosis. The paper on which our project is based, "[Pre-trained convolutional neural networks as feature extractors toward improved Malaria parasite detection in thin blood smear images](#)," by Rajaraman, et al., introduces some of the methods, including polymerase chain reaction (PCR) and rapid diagnostic tests (RDT). These two tests are typically used where high-quality microscopy services are not readily available.

The standard malaria diagnosis is typically based on a blood-smear workflow, according to Carlos Ariza's article . "[Malaria Hero: A web app for faster malaria diagnosis](#)," which I learned about in Adrian Rosebrock's "[Deep learning and medical image analysis with Keras](#)." I appreciate the authors of these excellent resources for giving me more perspective on malaria prevalence, diagnosis, and treatment.

Thus, malaria detection could benefit from automation using deep learning.

TOOLS AND TECHNOLOGIES:

PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

APPLICATIONS OF PYTHON:

Web and internet development

Scientific and numeric computing

Data Analysis

Desktop GUIs

Machine Learning

Data visualization

Game Development

Software Development

Business Application

ANACONDA:

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 15 million users and includes more than 1500 popular data-science packages suitable for Windows, Linux, and MacOS.

DATA SCIENCE :

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both structured and unstructured, similar to data mining. Data science is a "concept to unify statistics, data analysis, machine learning and their related methods" in order to "understand and analyze actual phenomena" with data. It employs techniques and theories drawn from many fields within the context of mathematics, statistics, information science, and computer science.

PYTHON PACKAGES:

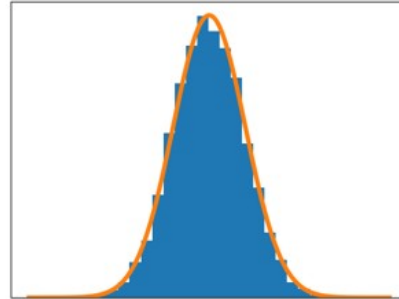
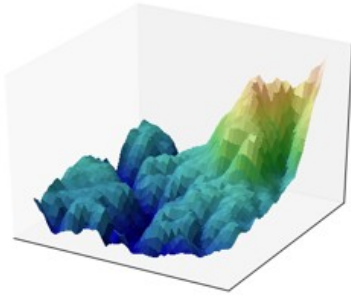
NUMPY:

NumPy (pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee)) is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

MATPLOTLIB:

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.



PANDAS:

Pandas is an open source, BSD-licensed library providing high-performance, easy- to-use data structures and data analysis tools for the *Python* programming language. Pandas library is well suited for data manipulation and analysis using python. In particular, it offers data structures and operations for manipulating numerical tables and time series.

SCIKIT LEARN:

Scikit-learn provides machine learning libraries for python some of the features of Scikit- learn includes:

- i.Simple and efficient tools for data mining and data analysis
- ii.Accessible to everybody, and reusable in various contexts
- iii.Built on NumPy, SciPy, and matplotlib
- iv.Open source, commercially usable - BSD license

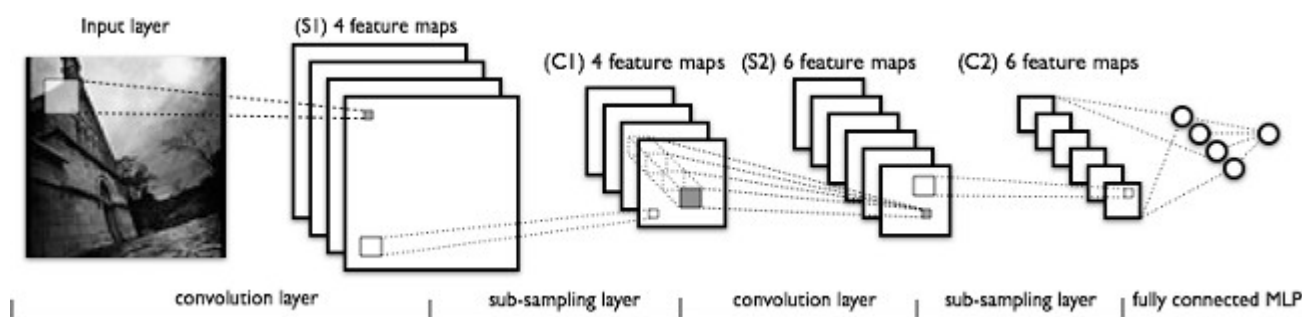
KERAS:

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System),and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the Xception deep neural network mode.

MALARIA DETECTION FOR DEEP LEARNING:

Manual diagnosis of blood smears is an intensive manual process that requires expertise in classifying and counting parasitized and uninfected cells. This process may not scale well, especially in regions where the right expertise is hard to find. Some advancements have been made in leveraging state-of-the-art image processing and analysis techniques to extract hand-engineered features and build machine learning-based classification models. However, these models are not scalable with more data being available for training and given the fact that hand-engineered features take a lot of time.

Deep learning models, or more specifically convolutional neural networks (CNNs), have proven very effective in a wide variety of computer vision tasks. (If you would like additional background knowledge on CNNs, I recommend reading [CS231n Convolutional Neural Networks for Visual Recognition](#).) Briefly, the key layers in a CNN model include convolution and pooling layers, as shown in the following figure.



A typical CNN architecture

Convolution layers learn spatial hierarchical patterns from data, which are also translation-invariant, so they are able to learn different aspects of images. For example, the first convolution layer will learn small and local patterns, such as edges and corners, a second convolution layer will learn larger patterns based on the features from the first layers, and so on. This allows CNNs to automate feature engineering and learn effective features that generalize well on new data points. Pooling layers helps with downsampling and dimension reduction.

Thus, CNNs help with automated and scalable feature engineering. Also, plugging in dense layers at the end of the model enables us to perform tasks like image classification. Automated malaria detection using deep learning models like CNNs could be very effective, cheap, and scalable, especially with the advent of transfer learning and pre-trained models that work quite well, even with constraints like less data.

The Rajaraman, et al., paper leverages six pre-trained models on a dataset to obtain an impressive accuracy of 95.9% in detecting malaria vs. non-infected samples. Our focus is to try some simple CNN models from scratch and a couple of pre-trained models using transfer learning to see the results we can get on the same dataset. We will use open source tools and frameworks, including Python and TensorFlow, to build our models.

COLLECTING DATA :

Kaggle is a platform for predictive modelling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the datasets uploaded by companies and users. This crowd sourcing approach relies on the fact that there are countless strategies that can be applied to any predictive modelling task and it is impossible to know beforehand which technique or analyst will be most effective.

On 8 March 2017, Google announced that they were acquiring Kaggle. They will join the Google Cloud team and continue to be a distinct brand. In January 2018, Booz Allen and Kaggle launched Data Science Bowl, a machine learning competition to analyze cell images and identify nuclei.

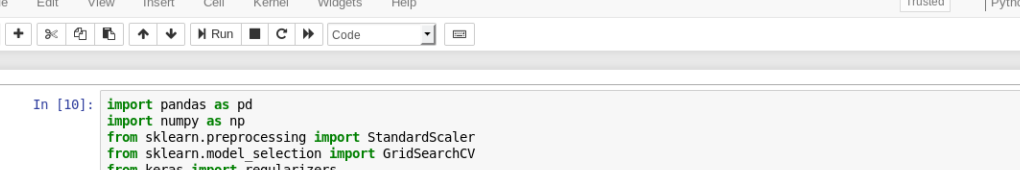
DATASET:

We have two folders that contain images of cells, infected and healthy. We can get further details about the total number of images by entering. It looks like we have a balanced dataset with 13,779 malaria and 13,779 non-malaria (uninfected) cell images. Let's build a data frame from this, which we will use when we start building our datasets. To build deep learning models, we need training data, but we also need to test the model's performance on unseen data. We will use a 60:10:30 split for train, validation, and test datasets, respectively. We will leverage the train and validation datasets during training and check the performance of the model on the test dataset.

The images will not be of equal dimensions because blood smears and cell images vary based on the human, the test method, and the orientation of the photo. Let's get some summary statistics of our training dataset to determine the optimal image dimensions (remember, we don't touch the test dataset at all!).

MODULES IMPORTED:

MODULES IMPORTED:



Jupyter MALARIA_DETECTION_BY_DEEP_LEARNING (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [10]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from keras import regularizers
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import numpy as np
import os
import cv2
import keras
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

from random import shuffle
from tqdm import tqdm
import scipy
import skimage
from skimage.transform import resize
import random
```

In [11]:

```
print(os.listdir('./cell_images'))

['Uninfected', 'Parasitized']
```

DATA LOADING:

JupyterMARIADB_DETECTION_BY_DEEP_LEARNING (autosaved)

FileEditViewInsertCellKernelWidgetsHelp

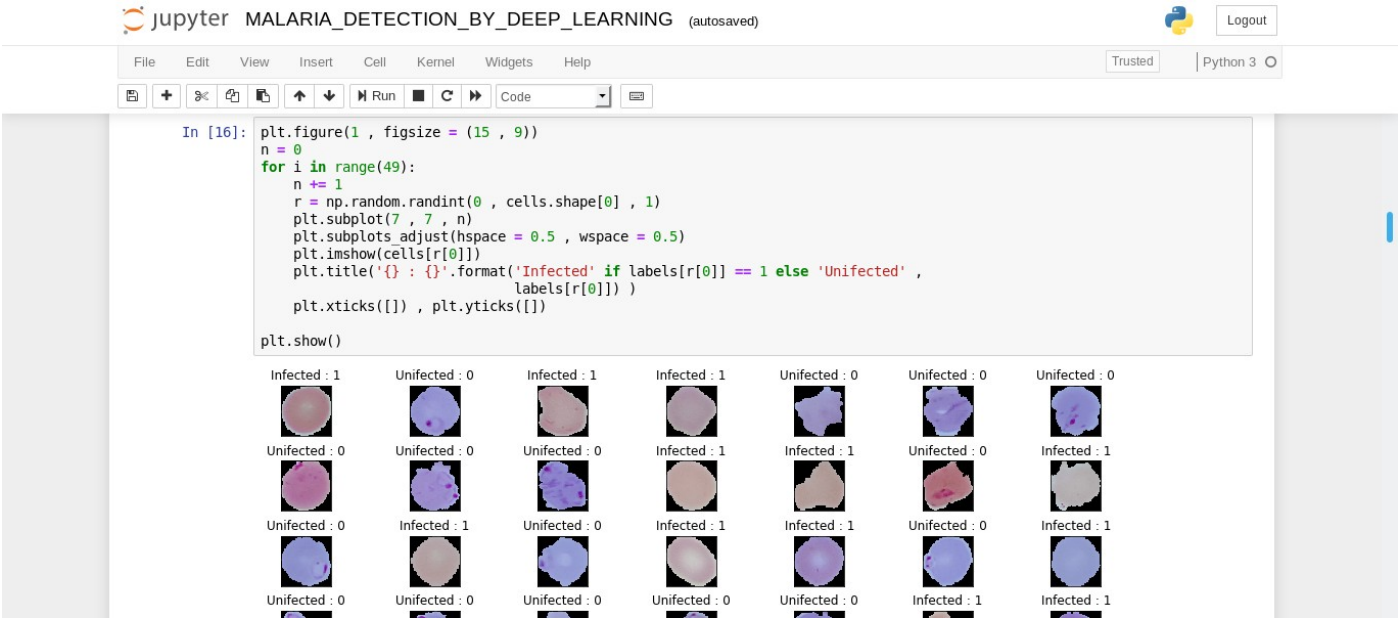
Trust Python 3

uninfected = os.listdir('./cell_images/Uninfected/')

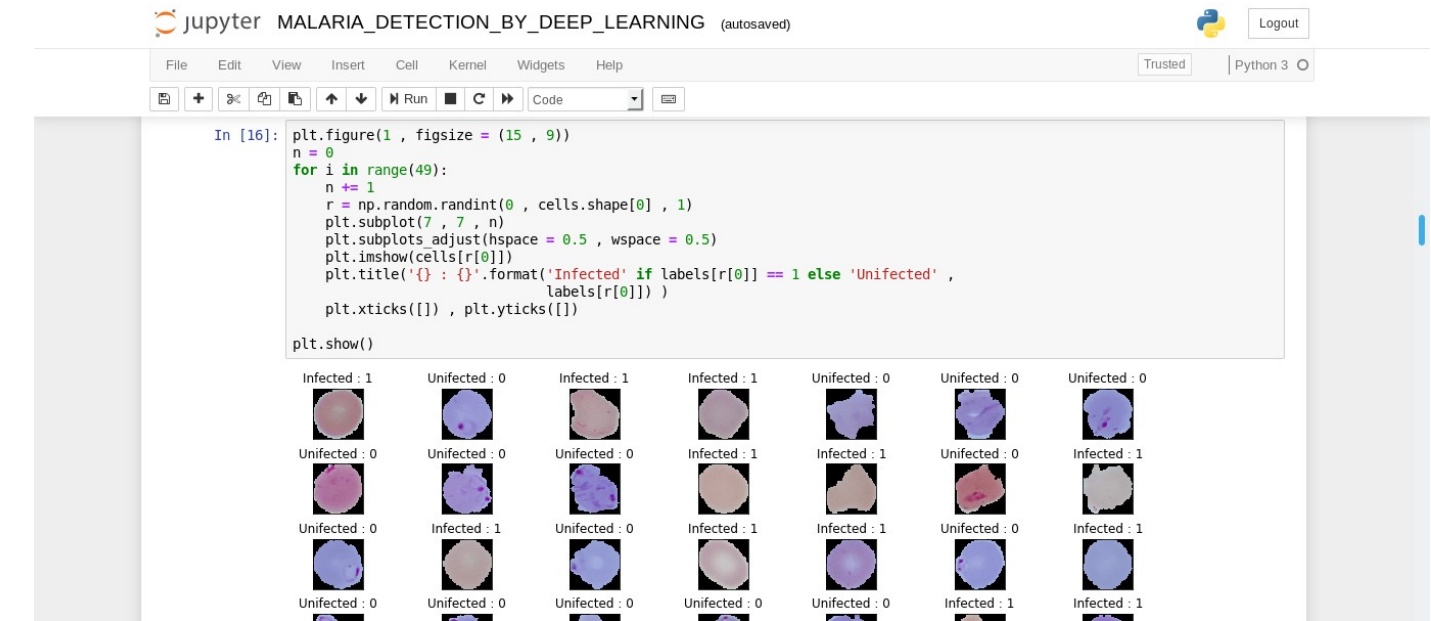
In [13]: data=[]
labels=[]
Parasitized=os.listdir("./cell_images/Parasitized/")
for a in Parasitized:
 try:
 image=cv2.imread("./cell_images/Parasitized/"+a)
 image_from_array = Image.fromarray(image, 'RGB')
 size_image = image_from_array.resize((50, 50))
 data.append(np.array(size_image))
 labels.append(0)
 except AttributeError:
 print("")

Uninfected=os.listdir("./cell_images/Uninfected/")
for b in Uninfected:
 try:
 image=cv2.imread("./cell_images/Uninfected/"+b)
 image_from_array = Image.fromarray(image, 'RGB')
 size_image = image_from_array.resize((50, 50))
 data.append(np.array(size_image))
 labels.append(1)
 except AttributeError:
 print("")

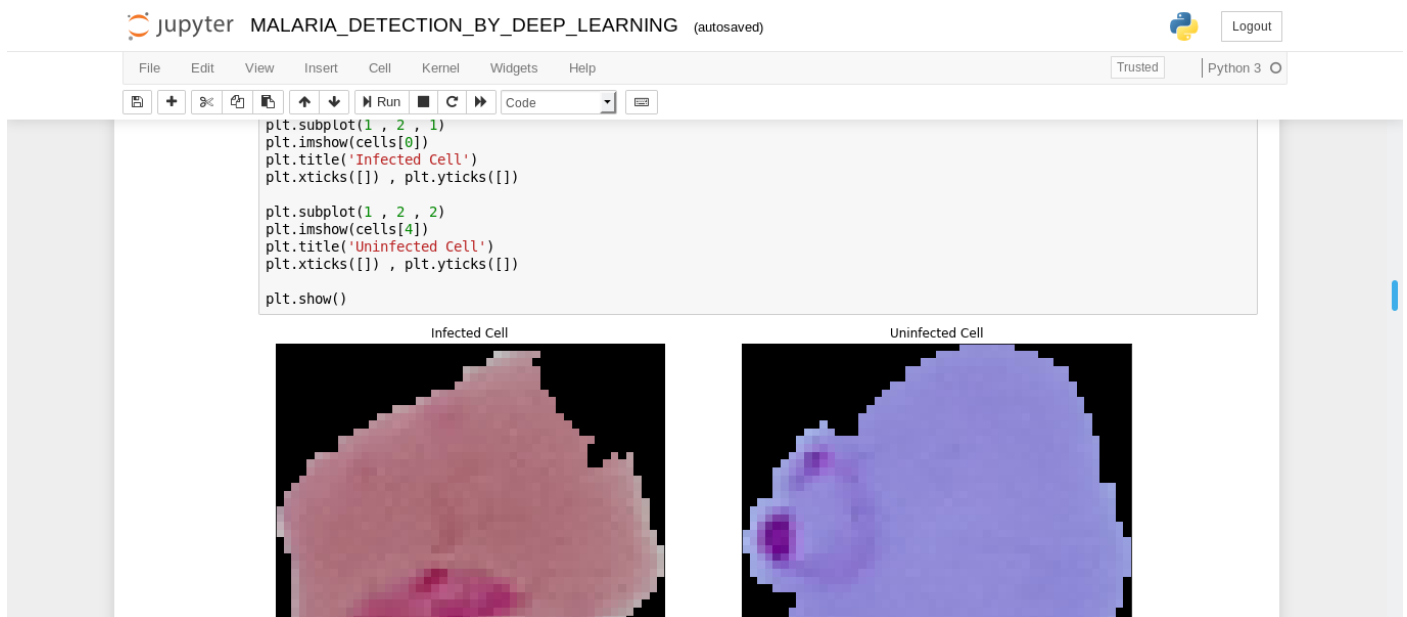
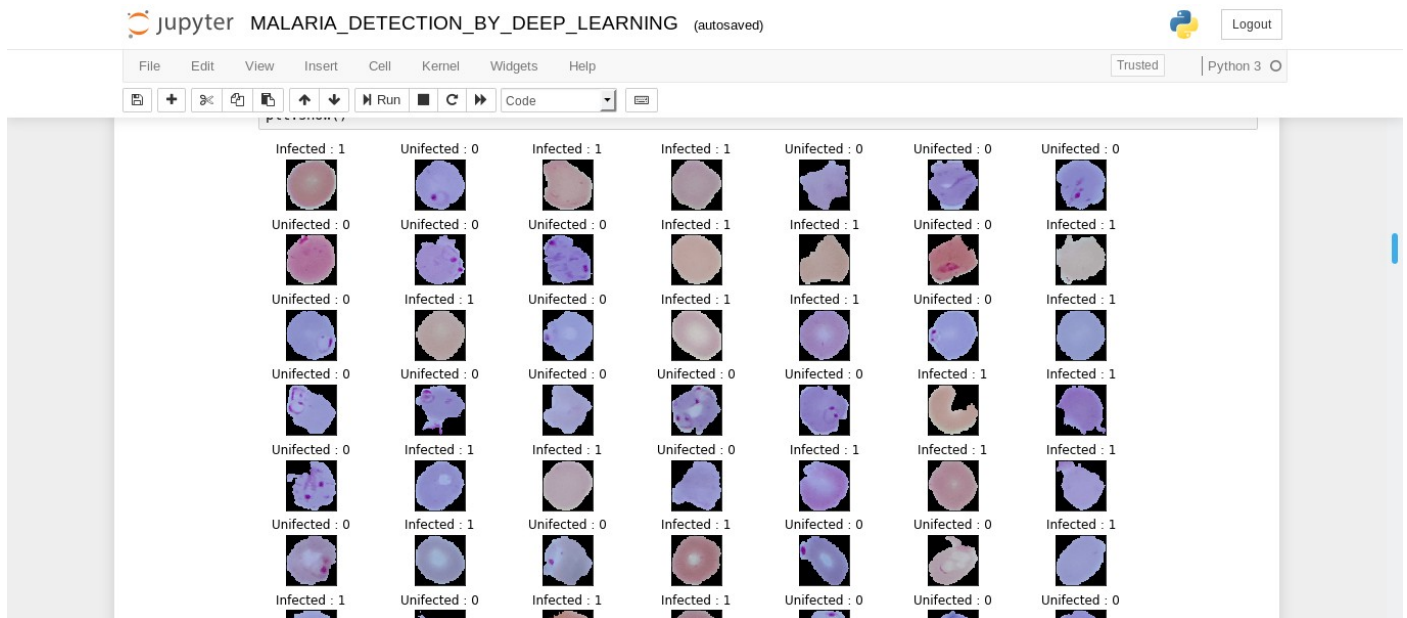
DATA VISUALIZATION:



We apply parallel processing to speed up the image-read operations and, based on the summary statistics, we



will resize each image to 125x125 pixels. Let's load up all of our images and resize them to these fixed dimensions. We leverage parallel processing again to speed up computations pertaining to image load and resizing. Finally, we get our image tensors of the desired dimensions, as depicted in the preceding output. We can now view some sample cell images to get an idea of how our data looks.



Based on these sample images, we can see some subtle differences between malaria and healthy cell images. We will make our deep learning models try to learn these patterns during model training.



MODEL TRAINING:

In the model training phase, we will build three deep learning models, train them with our training data, and compare their performance using the validation data. We will then save these models and use them later in the model evaluation phase.

SPLITTING DATASET INTO TRAIN AND TEST:

The whole dataset is divided into two parts (train and test) for training and testing the model.

```
In [25]: (x_train,x_test)=Cells[(int)(0.1*len_data):],Cells[::(int)(0.1*len_data)]
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
train_len=len(x_train)
test_len=len(x_test)
```

```
In [26]: (y_train,y_test)=labels[(int)(0.1*len_data):],labels[::(int)(0.1*len_data)]
```

```
In [27]: y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
```

MODEL:

Jupyter

MALARIA DETECTION_BY DEEP LEARNING

(autosaved)

Logout

FileEditViewInsertCellKernelWidgetsHelp

TrustedPython 3

+

↶

↷

↺

↻

↵

↶

↷

↺

↻

↵

Code

```
In [30]: from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=10,
    zoom_range = 0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False)
datagen.fit(x_train)


In [31]: from keras.callbacks import EarlyStopping
earllystop= EarlyStopping(monitor='val_acc', patience=3)
epochs = 20
batch_size = 256


In [32]: model=Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding="same", activation="relu", input_shape=(50,50,3)))
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"))
```

ACCURACY SCORE:

The accuracy of the model is 95.5% as shown on the below figure. And the confusion matrix shows the validation accuracy of the model (i.e. among the whole dataset how many of that the model has predicted correctly). It is shown that the model has predicted 2632 images correctly and for the remaining images it is showing incorrect result.

The figure displays two screenshots of a Jupyter Notebook interface, likely from a Kaggle competition, showing the process of training a deep learning model for malaria detection.

Top Screenshot: Model Architecture Summary

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 50, 50, 16)	208
max_pooling2d_7 (MaxPooling2)	(None, 25, 25, 16)	0
conv2d_8 (Conv2D)	(None, 25, 25, 32)	2080
max_pooling2d_8 (MaxPooling2)	(None, 12, 12, 32)	0

Bottom Screenshot: Training Process

The code cell shows the training of a model using `model.fit_generator` with the following parameters:

```
history = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                             epochs = epochs, validation_data = (x_test,y_test),
                             verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size
                             , callbacks=[earlystop])
```

The output shows a warning about deprecated TensorFlow ops and the training progress for 7 epochs:

```
WARNING:tensorflow:From /root/anaconda3/lib/python3.7/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/20
96/96 [=====] - 102s 1s/step - loss: 1.8058 - acc: 0.6282 - val_loss: 0.5768 - val_acc: 0.7358
Epoch 2/20
96/96 [=====] - 58s 603ms/step - loss: 0.5186 - acc: 0.7860 - val_loss: 0.3809 - val_acc: 0.874
0
Epoch 3/20
96/96 [=====] - 59s 614ms/step - loss: 0.3431 - acc: 0.8889 - val_loss: 0.2778 - val_acc: 0.904
5
Epoch 4/20
96/96 [=====] - 58s 604ms/step - loss: 0.2907 - acc: 0.9139 - val_loss: 0.2834 - val_acc: 0.915
4
Epoch 5/20
96/96 [=====] - 58s 609ms/step - loss: 0.2736 - acc: 0.9214 - val_loss: 0.2395 - val_acc: 0.932
8
Epoch 6/20
96/96 [=====] - 58s 607ms/step - loss: 0.2519 - acc: 0.9316 - val_loss: 0.2386 - val_acc: 0.934
3
Epoch 7/20
96/96 [=====] - 58s 608ms/step - loss: 0.2503 - acc: 0.9349 - val_loss: 0.2166 - val_acc: 0.951
```

jupyterMALARIA_DETECTION_BY_DEEP_LEARNING (autosaved)

Python 3

Logout

FileEditViewInsertCellKernelWidgetsHelp

Run

Code

```
Epoch 8/20
96/96 [=====] - 58s 608ms/step - loss: 0.2372 - acc: 0.9371 - val_loss: 0.2244 - val_acc: 0.9539
Epoch 9/20
96/96 [=====] - 58s 609ms/step - loss: 0.2258 - acc: 0.9427 - val_loss: 0.2237 - val_acc: 0.9390
Epoch 10/20
96/96 [=====] - 59s 617ms/step - loss: 0.2219 - acc: 0.9423 - val_loss: 0.1987 - val_acc: 0.9543
Epoch 11/20
96/96 [=====] - 59s 613ms/step - loss: 0.2086 - acc: 0.9464 - val_loss: 0.2024 - val_acc: 0.9543
Epoch 12/20
96/96 [=====] - 59s 614ms/step - loss: 0.2127 - acc: 0.9445 - val_loss: 0.1932 - val_acc: 0.9572
Epoch 13/20
96/96 [=====] - 59s 613ms/step - loss: 0.2012 - acc: 0.9471 - val_loss: 0.1690 - val_acc: 0.9561
Epoch 14/20
96/96 [=====] - 59s 617ms/step - loss: 0.2042 - acc: 0.9466 - val_loss: 0.1831 - val_acc: 0.9564
Epoch 15/20
96/96 [=====] - 61s 633ms/step - loss: 0.1954 - acc: 0.9488 - val_loss: 0.1819 - val_acc: 0.9554

In [35]: from sklearn.metrics import confusion_matrix
pred = model.predict(x_test)
pred = np.argmax(pred,axis = 1)
```

CONCLUSION:

jupyterMALARIA_DETECTION_BY_DEEP_LEARNING (autosaved)

Python 3

Logout

FileEditViewInsertCellKernelWidgetsHelp

Run

Code

```
In [35]: from sklearn.metrics import confusion_matrix
pred = model.predict(x_test)
pred = np.argmax(pred,axis = 1)
y_true = np.argmax(y_test,axis = 1)

In [36]: CM = confusion_matrix(y_true, pred)
from mlxtend.plotting import plot_confusion_matrix
fig, ax = plot_confusion_matrix(conf_mat=CM , figsize=(5, 5))
plt.show()
```

Malaria detection is not an easy procedure, and the availability of qualified personnel around the globe is a serious concern in the diagnosis and treatment of cases. We looked at an interesting real-world medical imaging case

study of malaria detection. Easy-to-build, open source techniques leveraging AI can give us state-of-the-art accuracy in detecting malaria, thus enabling AI for social good.

I encourage you to check out the articles and research papers mentioned in this article, without which it would have been impossible for me to conceptualize and write it. If you are interested in running or adopting these techniques, all the code used in this article is available on

https://github.com/subhrockzz/DEEP_LEARNING/blob/master/MALARIA_DETECTION_BY_DEEP_LEARNING.ipynb.

Let's hope for more adoption of open source AI capabilities in healthcare to make it less expensive and more accessible for everyone around the world!

REFERENCES:

<https://www.python.org/>

<https://anaconda.org/anaconda/python>

<http://www.numpy.org/>

<https://matplotlib.org/>

<http://scikit-learn.org/>

<https://pandas.pydata.org/>

<https://pandas.pydata.org/>

<https://ipython.org/>