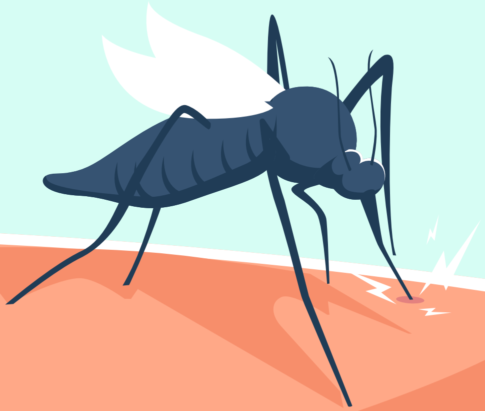
IMAGE PROCESSING
REVIEW III

DETECTING MALARIAL PARASITE USING IMAGE PROCESSING

Team Members
18BCE0924 ABHISHEK KUMAR
18BCE0728 ADITI TARIGOPPULA
18BCI0219 ADYA SHARMA

Under the Guidance of
Prof. Swathi J.N



LINK TO GITHUB REPOSITORY:
https://github.com/mystic-potato/Malarial-Parasite-Detector

DEMO VIDEO:
https://drive.google.com/file/d/16OsHTqE33FKJroR-8bascaMyiOWoUqI9/view?usp=sharing

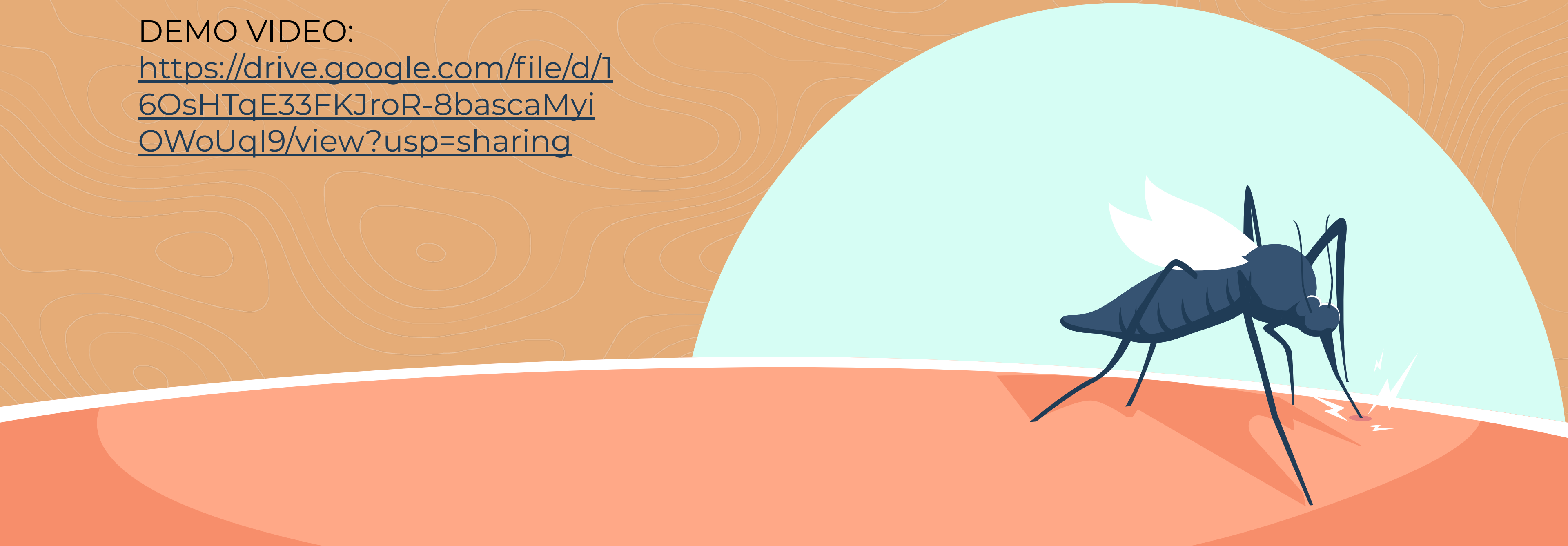


TABLE OF CONTENTS

TABLE OF CONTENTS

ABSTRACT

This paper reviews image analysis studies aiming automated diagnosis or screening of malaria infection in microscope images of thin blood film smears. Malaria is a life-threatening disease caused by Plasmodium parasites that infect the red blood cells (RBCs). Manual identification and counting of parasitized cells in microscopic thick/thin-film blood examination remains the common, but burdensome method for disease diagnosis. Its diagnostic accuracy is adversely impacted by inter/intra-observer variability, particularly in large-scale screening under resource- constrained settings. According to the WHO, this parasite is responsible for passing to more than two million individuals and approximately 300 to 500 million infection cases annually. Although effective ways to manage malaria now exist, the number of malaria cases is still increasing, due to several factors. Hence the purpose of the project is to implement a solution for easy and malaria diagnosis with high accuracy.

INTRODUCTION

Malaria management is a challenging problem all over the globe particularly in Asian and African continents. Presently, even 110 years after the Nobel Prize of Ronald Ross for his work on malaria, people in the European region are also at risk from diseases carried by vectors both within the region and when traveling abroad. While treatment of malaria itself is a challenging problem its quick detection is also a problem with no less significance. There are mainly four species of malaria parasites infecting human beings namely, Plasmodium falciparum, Plasmodium vivax, Plasmodium ovale and Plasmodium malaria. Plasmodium vivax, is found mainly in tropical and subtropical areas and has a severe clinical manifestation. Rapid detection of presence of the parasite in human blood and early institution of antimalarial drugs are the mainstay of management of the disease. WHO recommends that all cases of suspected malaria be confirmed using parasite-based diagnostic testing (either microscopy or rapid diagnostic test) before administering treatment . In the malaria detection test, microscopy based diagnosis has the central importance for species differentiation, parasite quantification, management of severe disease. Additionally, the method may be amenable to a larger section of society because of its scalability and low running cost.

PROBLEM STATEMENT

The primary aim of the project is to develop a fully automated image classification system to positively identify malaria parasites present in thin blood smears, and differentiate the species In this project we proposed a system of using Deep learning with TensorFlow–it is a python dependency used for image processing As using Deep learning with TensorFlow has shown great results with good accuracy.

LITERATURE SURVEY

BASE PAPER IDENTIFICATION

The base paper we have identified for this project is from the same source that we got our database from. The overview of the base paper is given below:

Title:

Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images [1]

Authors:

Sivaramakrishnan Rajaraman, Sameer K. Antani1, Mahdieh Poostch,
Kamolrat Silamut, Md. A. Hossain, Richard J. Maude,Stefan Jaeger and
George R. Thoma

Year of publication:

2018

PROPOSED METHODOLOGY- BASE PAPER

In this study, the authors 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. They 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. In contrast to machine learning techniques which require hand engineered methods, the authors have used Convolutional Neural Networks (CNN), a class of deep learning (DL) models which 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 paper, for the binary task of classifying parasitized and uninfected cells, the variability in
data is several orders of magnitude smaller as compared to previous methods used. As future work they want to release a mobile application that can do this work for them.

In our project we aim to use the same base principles and dataset as the ones used for this paper. We will add extra features as needed and make a GUI for easier usage of the proposed solution.

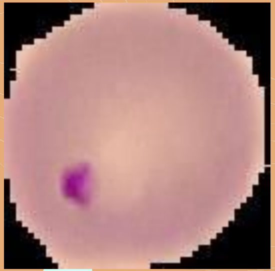
DATASET

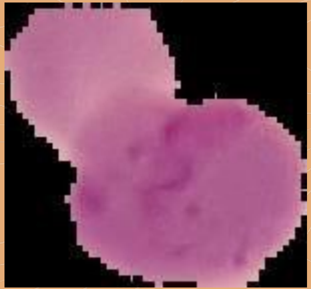
We have taken dataset of this project from the official NIH website- https://ceb.nlm.nih.gov/repositories/malariadatasets/

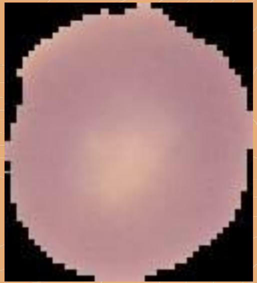
The dataset consists of 27,558 cell images with equal instances of parasitized and uninfected cells.

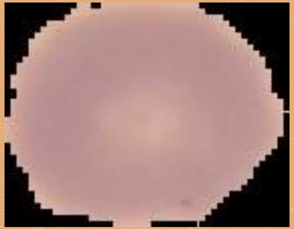
The data was collected using a mobile application which captured Giemsa-stained thin blood smear slides from 150 malaria infected and 50 healthy patients. The smartphone's built-in camera acquired images of slides for each microscopic field of view. The images were manually annotated by an expert slide reader.

Some input cells of the dataset are:





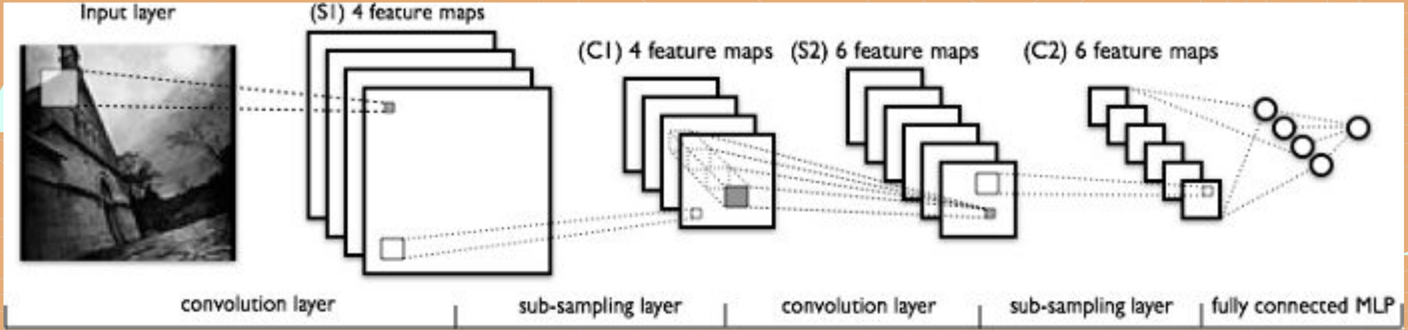




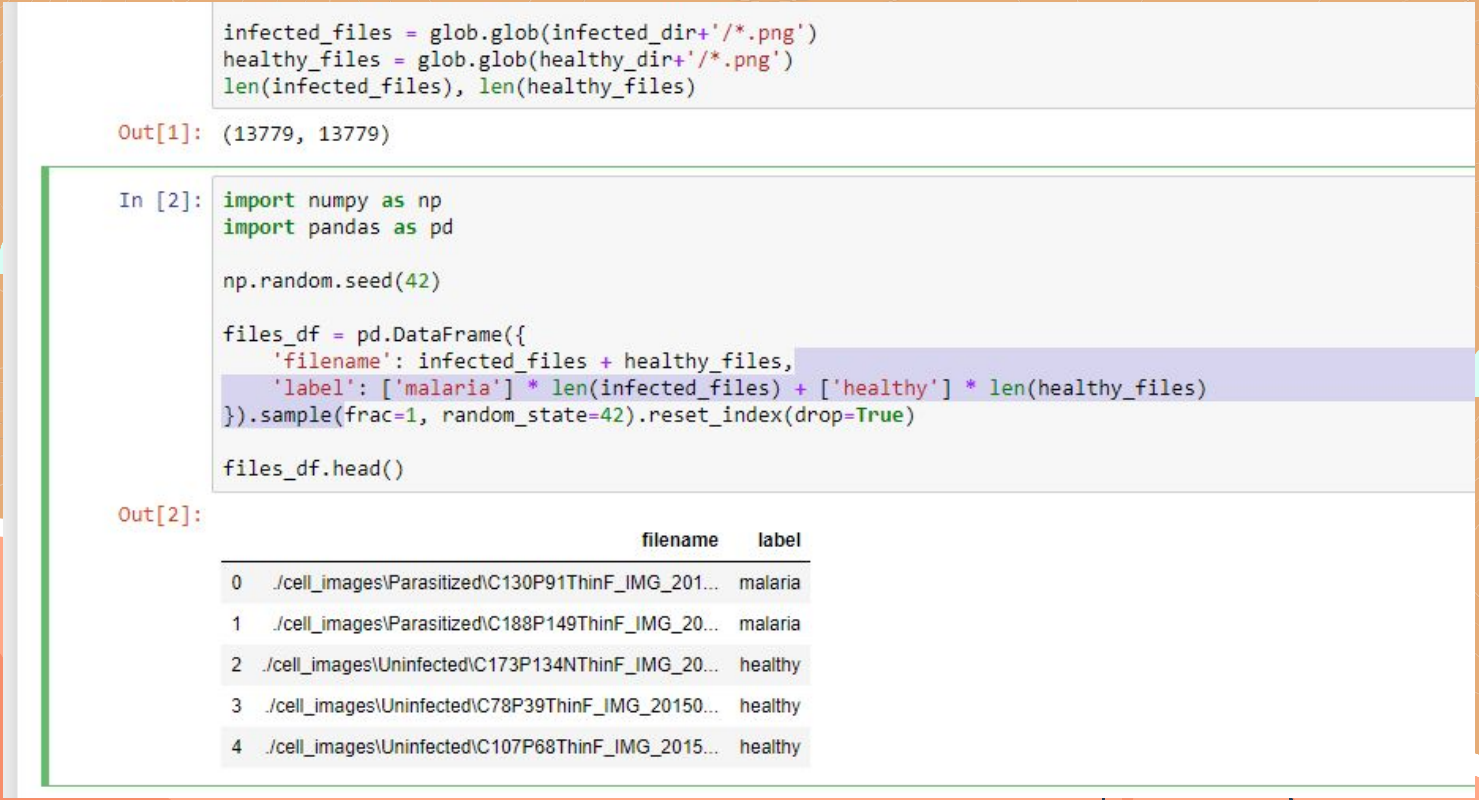
PROPOSED METHODOLOGY

The primary aim of the project is to develop a fully automated image classification system to positively identify malaria parasites present in thin blood smears, and differentiate the species In this project we proposed a system of using Deep learning with Keras and Tensorflow are python dependencies used for image processing. Using Deep learning with Tensorflow has shown great results with good accuracy.
We have trained and saved a model of our machine learning algorithm and used Python library Tkinter to make a GUI in order to use the trained model.
The upcoming sections will explain these in further detail.

CNN -CONCEPT


Input layer
(S1) 4 feature maps
(C1) 4 feature maps
(S2) 6 feature maps
(C2) 6 feature maps
convolution layer
sub-sampling layer
convolution layer
sub-sampling layer
fully connected MLP

VIEW DATASET DETAILS



```
infected_files = glob.glob(infected_dir+'/*.png')
healthy_files = glob.glob(healthy_dir+'/*.png')
len(infected_files), len(healthy_files)
```

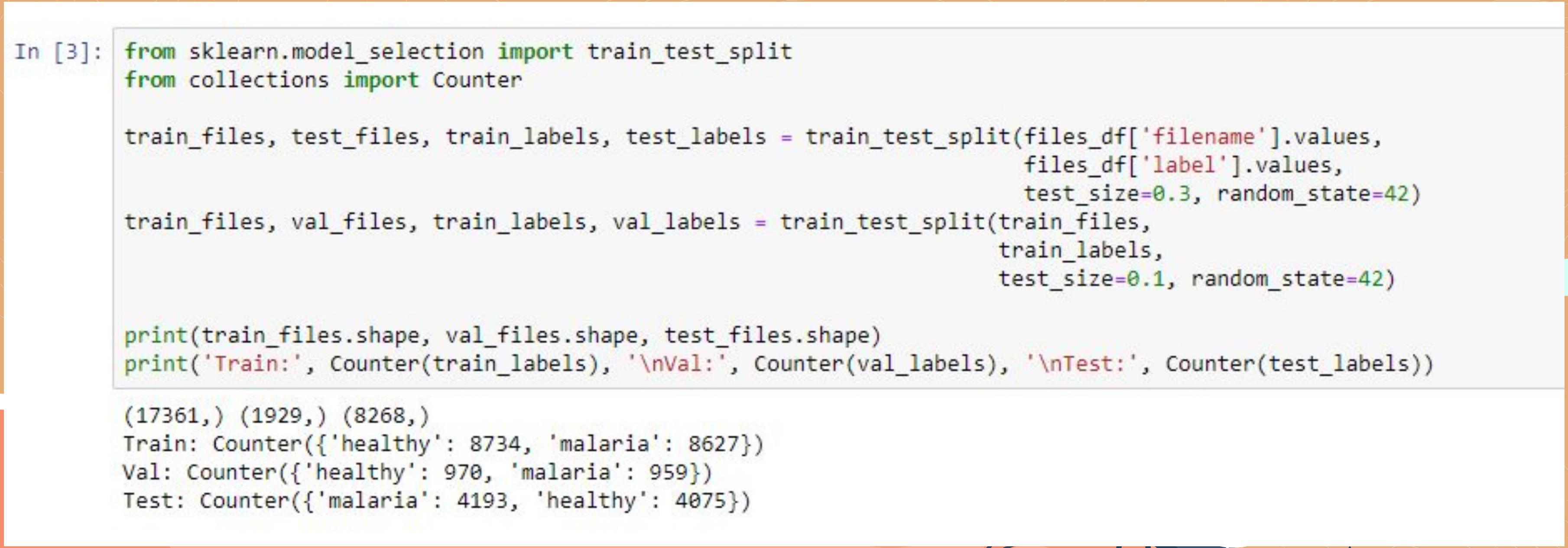
```
Out[1]: (13779, 13779)
```

```
In [2]: import numpy as np
        import pandas as pd

        np.random.seed(42)

        files_df = pd.DataFrame({
            'filename': infected_files + healthy_files,
            'label': ['malaria'] * len(infected_files) + ['healthy'] * len(healthy_files)
        }).sample(frac=1, random_state=42).reset_index(drop=True)

        files_df.head()
```

Out[2]:

	filename	label
0	./cell_images\Parasitized\C130P91ThinF_IMG_201...	malaria
1	./cell_images\Parasitized\C188P149ThinF_IMG_20...	malaria
2	./cell_images\Uninfected\C173P134NThinF_IMG_20...	healthy
3	./cell_images\Uninfected\C78P39ThinF_IMG_20150...	healthy
4	./cell_images\Uninfected\C107P68ThinF_IMG_2015...	healthy

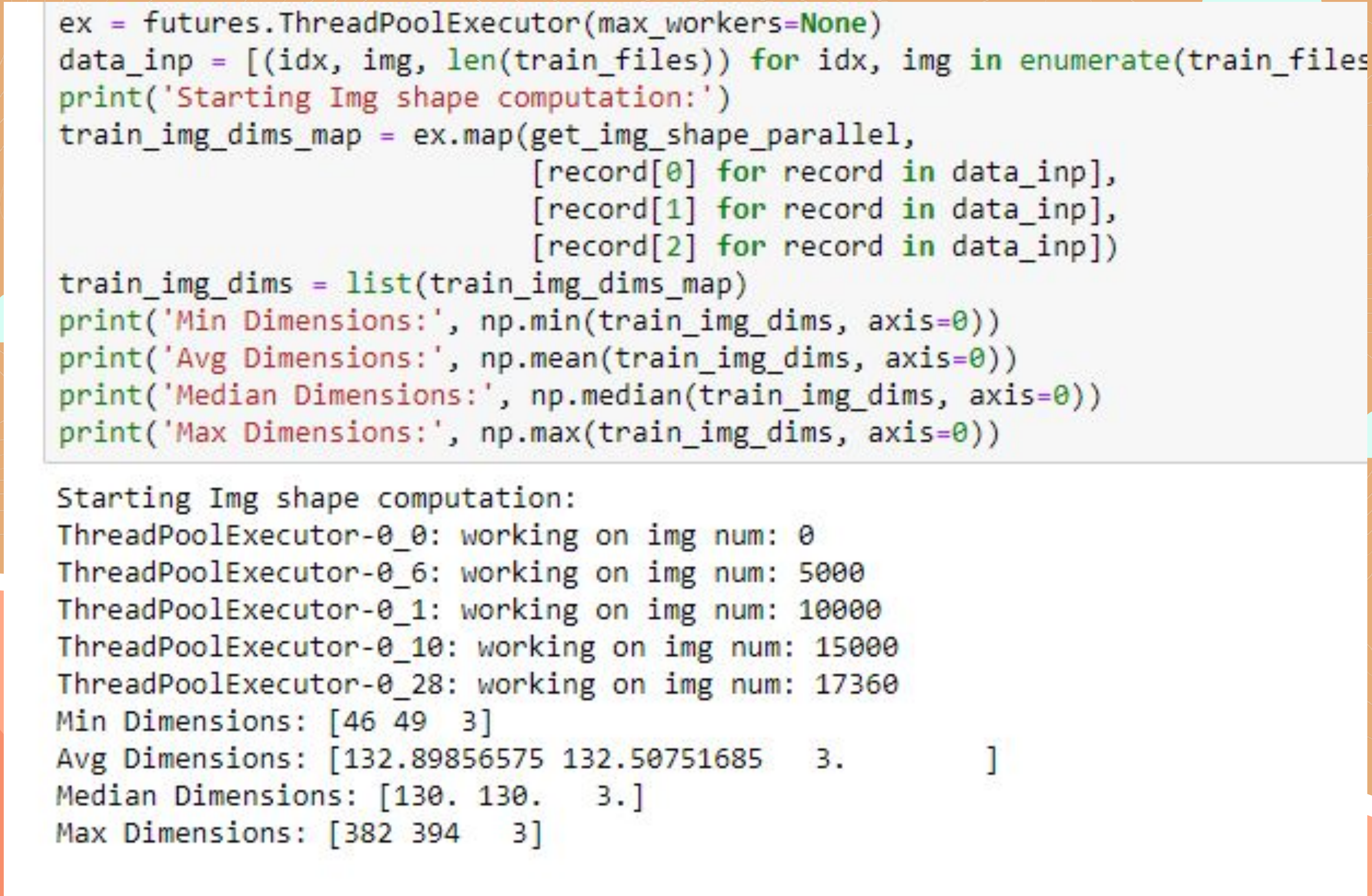
CREATE TRAIN VALIDATE AND TEST DATASET



```
In [3]: from sklearn.model_selection import train_test_split
        from collections import Counter

        train_files, test_files, train_labels, test_labels = train_test_split(files_df['filename'].values,
                                                                              files_df['label'].values,
                                                                              test_size=0.3, random_state=42)
        train_files, val_files, train_labels, val_labels = train_test_split(train_files,
                                                                            train_labels,
                                                                            test_size=0.1, random_state=42)

        print(train_files.shape, val_files.shape, test_files.shape)
        print('Train:', Counter(train_labels), '\nVal:', Counter(val_labels), '\nTest:', Counter(test_labels))
```

```
(17361,) (1929,) (8268,)
Train: Counter({'healthy': 8734, 'malaria': 8627})
Val: Counter({'healthy': 970, 'malaria': 959})
Test: Counter({'malaria': 4193, 'healthy': 4075})
```

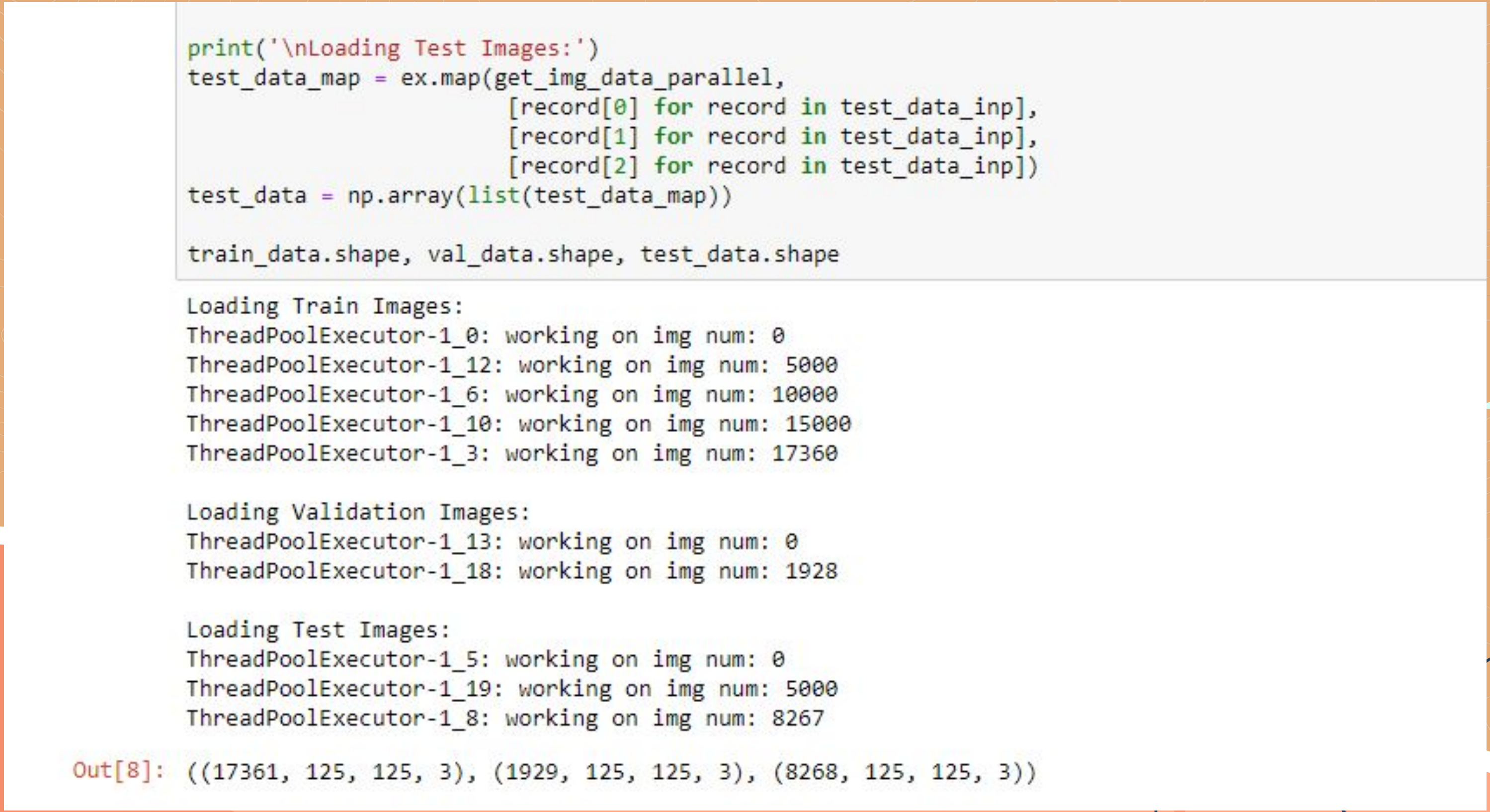
GET IMAGE DIMENSION STATISTICS



```
ex = futures.ThreadPoolExecutor(max_workers=None)
data_inp = [(idx, img, len(train_files)) for idx, img in enumerate(train_files
print('Starting Img shape computation:')
train_img_dims_map = ex.map(get_img_shape_parallel,
                            [record[0] for record in data_inp],
                            [record[1] for record in data_inp],
                            [record[2] for record in data_inp])
train_img_dims = list(train_img_dims_map)
print('Min Dimensions:', np.min(train_img_dims, axis=0))
print('Avg Dimensions:', np.mean(train_img_dims, axis=0))
print('Median Dimensions:', np.median(train_img_dims, axis=0))
print('Max Dimensions:', np.max(train_img_dims, axis=0))
```

```
Starting Img shape computation:
ThreadPoolExecutor-0_0: working on img num: 0
ThreadPoolExecutor-0_6: working on img num: 5000
ThreadPoolExecutor-0_1: working on img num: 10000
ThreadPoolExecutor-0_10: working on img num: 15000
ThreadPoolExecutor-0_28: working on img num: 17360
Min Dimensions: [46 49  3]
Avg Dimensions: [132.89856575 132.50751685   3.        ]
Median Dimensions: [130. 130.   3.]
Max Dimensions: [382 394   3]
```

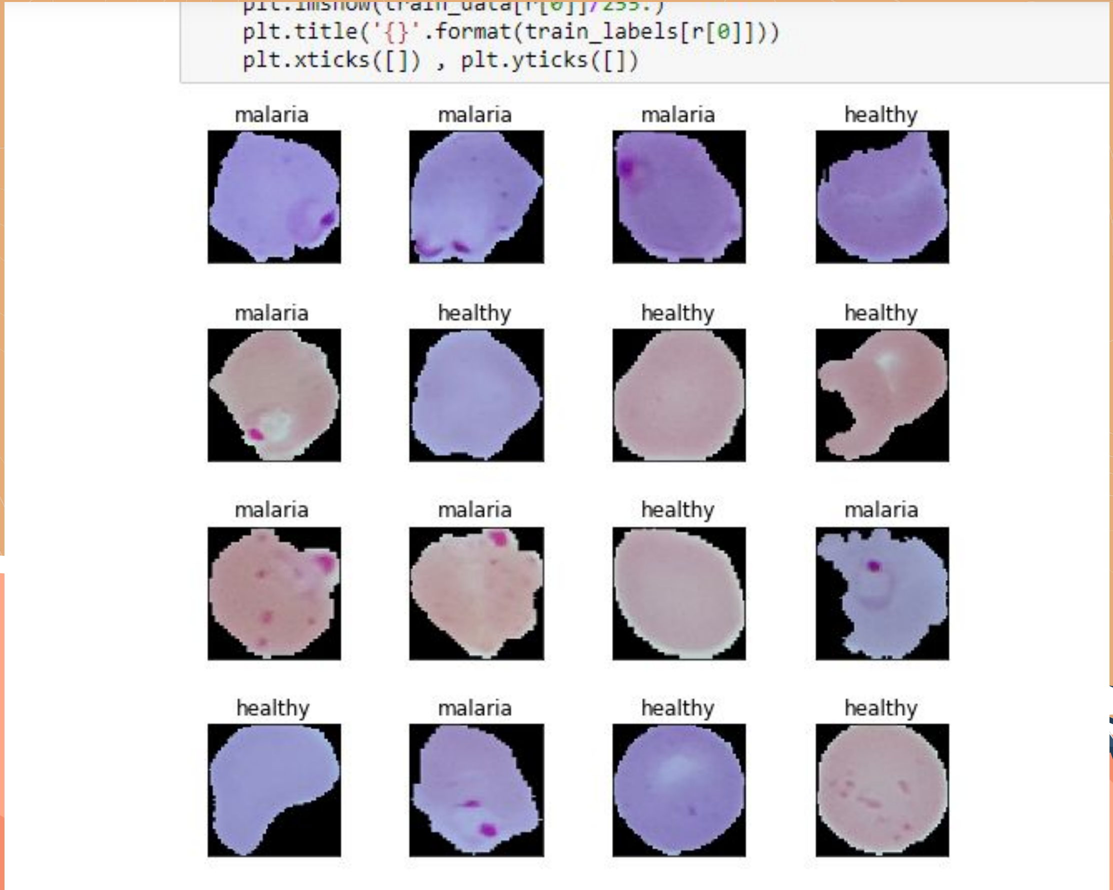
RESIZING IMAGES AND NEW STATISTICS



```
print('\nLoading Test Images:')
test_data_map = ex.map(get_img_data_parallel,
                        [record[0] for record in test_data_inp],
                        [record[1] for record in test_data_inp],
                        [record[2] for record in test_data_inp])
test_data = np.array(list(test_data_map))

train_data.shape, val_data.shape, test_data.shape
```

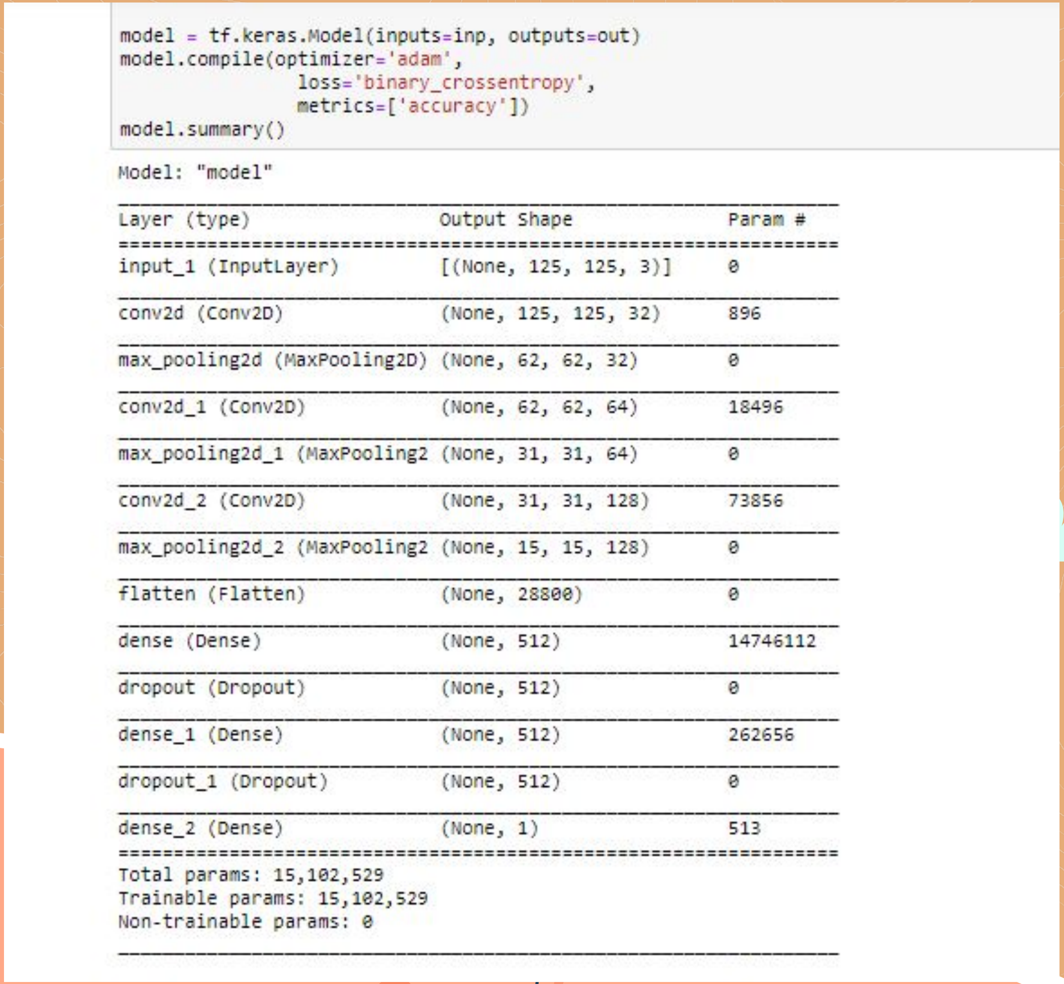
```
Loading Train Images:
ThreadPoolExecutor-1_0: working on img num: 0
ThreadPoolExecutor-1_12: working on img num: 5000
ThreadPoolExecutor-1_6: working on img num: 10000
ThreadPoolExecutor-1_10: working on img num: 15000
ThreadPoolExecutor-1_3: working on img num: 17360

Loading Validation Images:
ThreadPoolExecutor-1_13: working on img num: 0
ThreadPoolExecutor-1_18: working on img num: 1928

Loading Test Images:
ThreadPoolExecutor-1_5: working on img num: 0
ThreadPoolExecutor-1_19: working on img num: 5000
ThreadPoolExecutor-1_8: working on img num: 8267

Out[8]: ((17361, 125, 125, 3), (1929, 125, 125, 3), (8268, 125, 125, 3))
```

SAMPLE CELL IMAGES



```
    plt.imshow(train_data[r[0]]/255.)
    plt.title('{}'.format(train_labels[r[0]]))
    plt.xticks([]) , plt.yticks([])
```

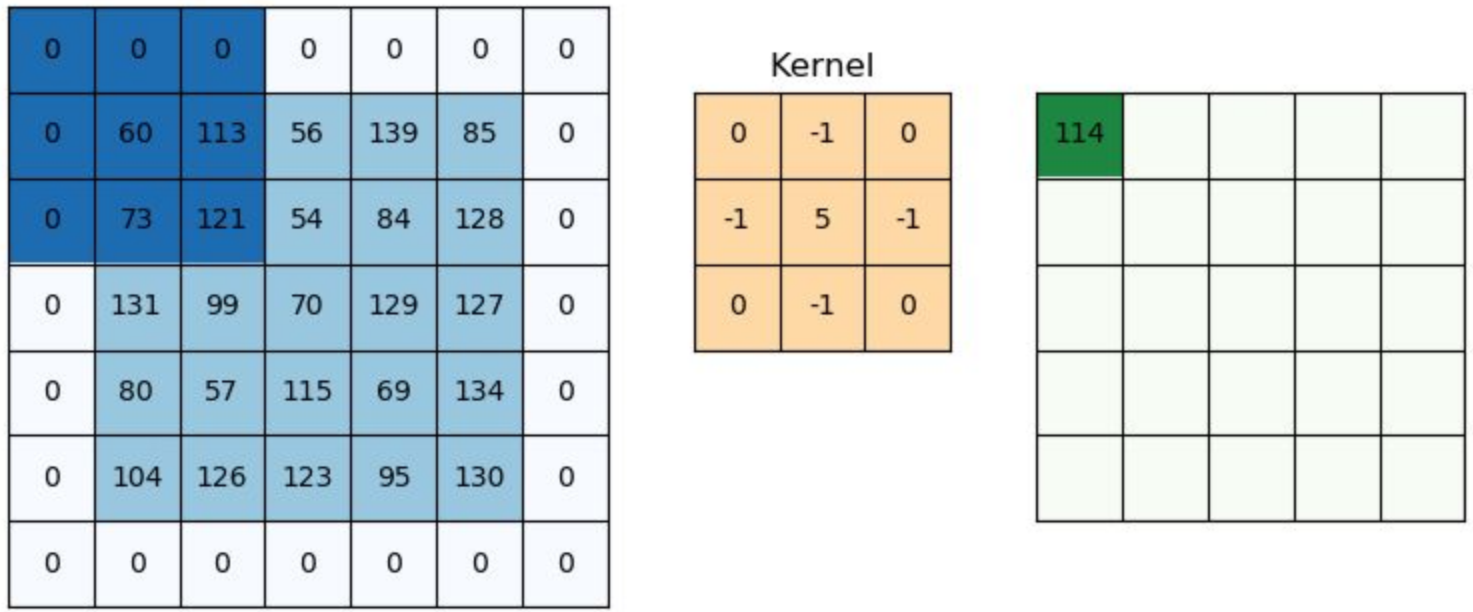
MODEL ARCHITECTURE



```
model = tf.keras.Model(inputs=inp, outputs=out)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 125, 125, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 125, 125, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 62, 62, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 31, 31, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 28800)             0
_________________________________________________________________
dense (Dense)                (None, 512)               14746112
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 15,102,529
Trainable params: 15,102,529
Non-trainable params: 0
_________________________________________________________________
```

CONV2D



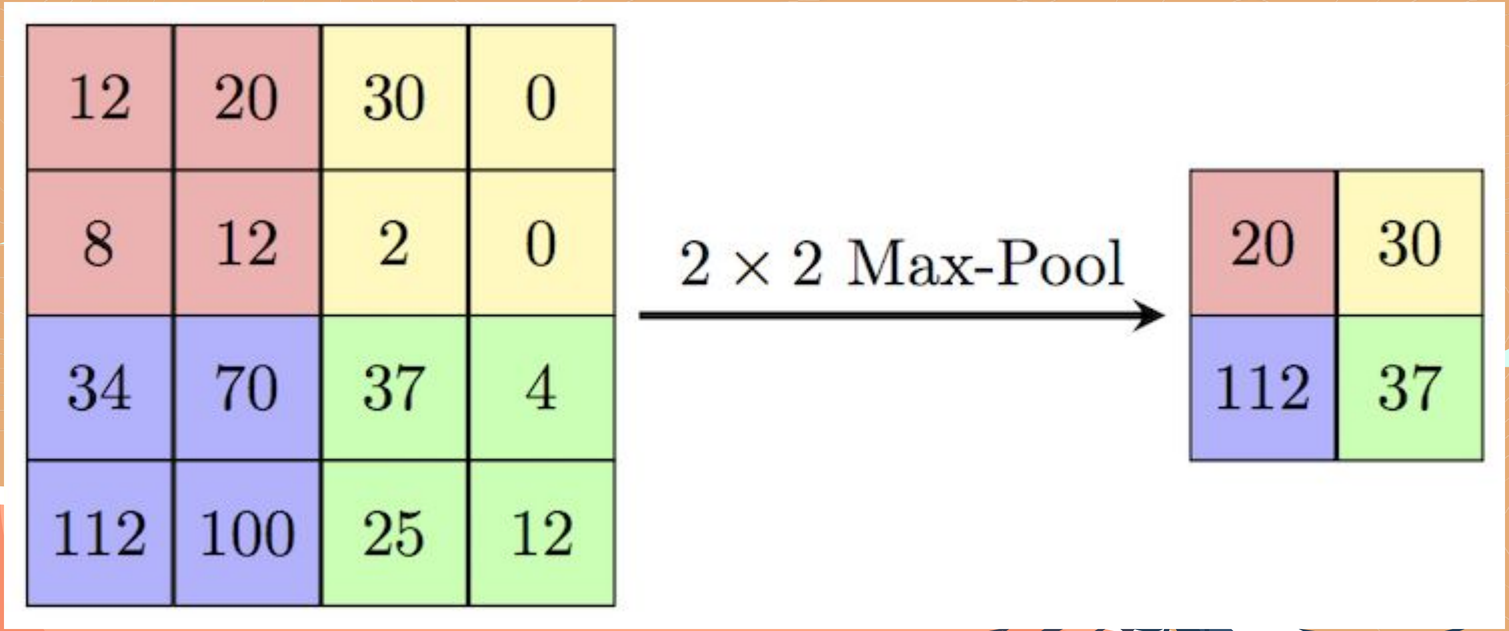
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

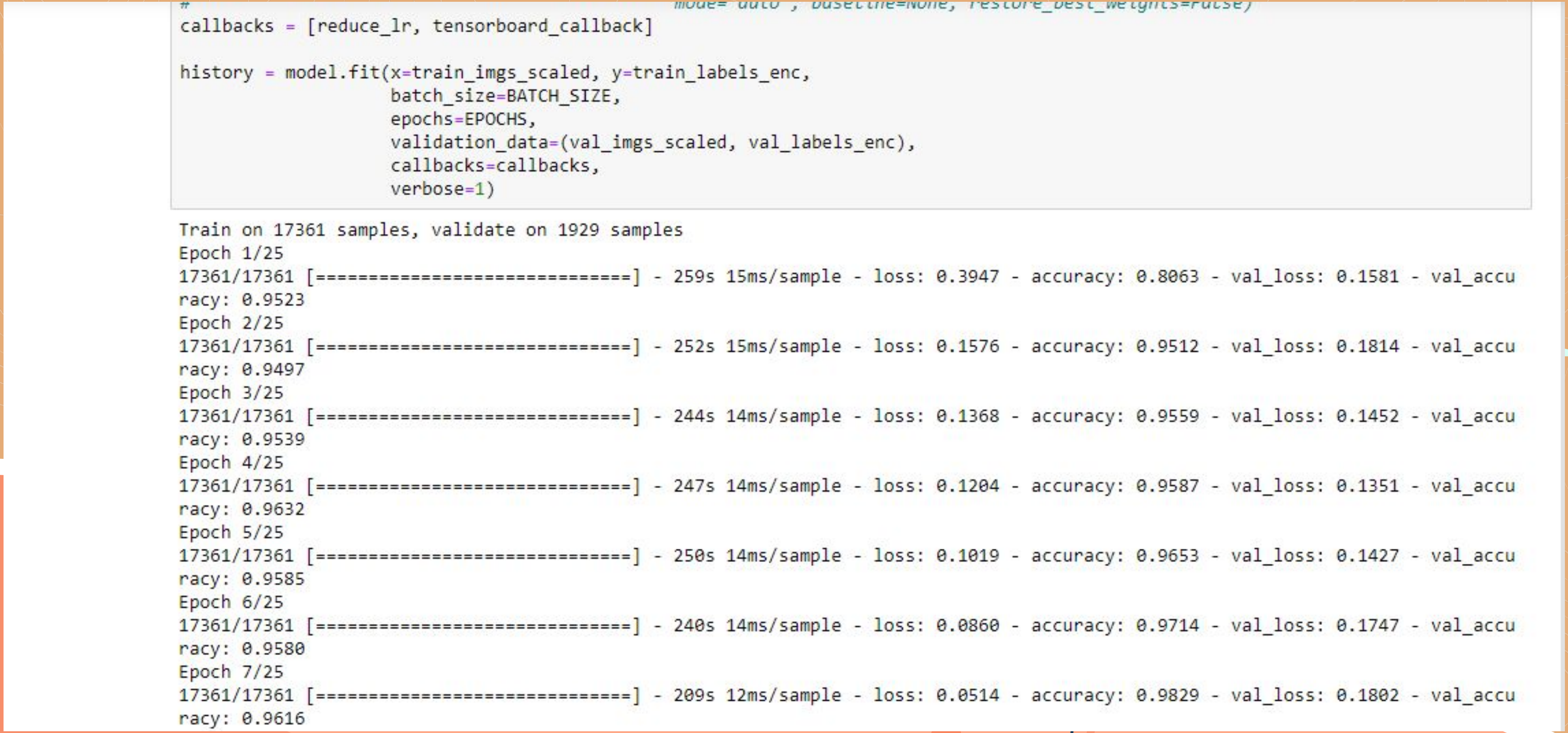
MAX POOLING



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

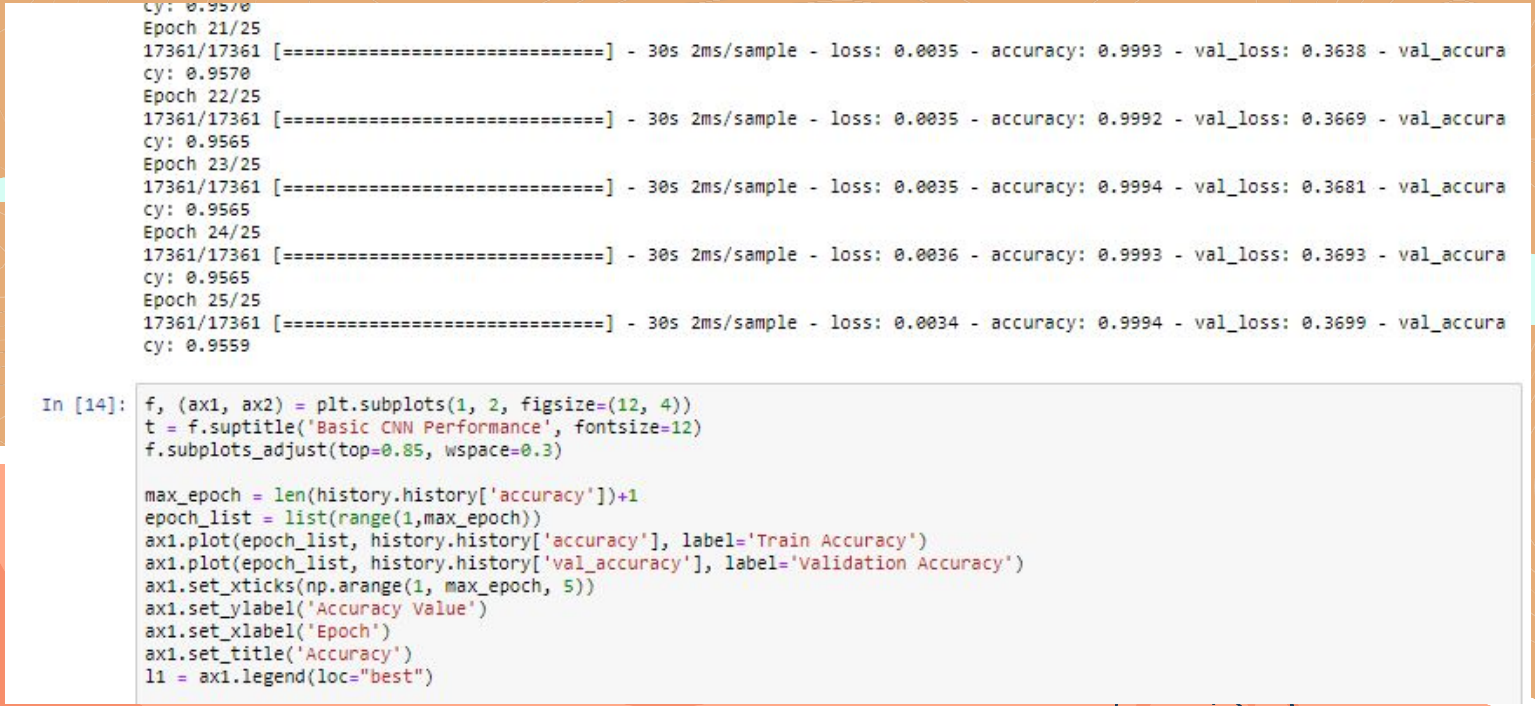
20	30
112	37

TRAIN MODEL



```
#                                               mode='auto', baseline=None, restore_best_weights=False)
callbacks = [reduce_lr, tensorboard_callback]

history = model.fit(x=train_imgs_scaled, y=train_labels_enc,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(val_imgs_scaled, val_labels_enc),
                    callbacks=callbacks,
                    verbose=1)
```

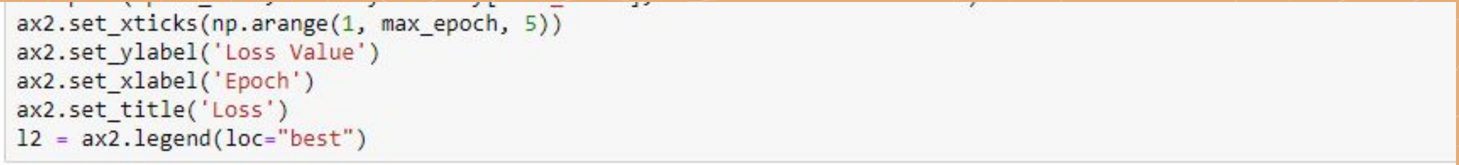
```
Train on 17361 samples, validate on 1929 samples
Epoch 1/25
17361/17361 [==============================] - 259s 15ms/sample - loss: 0.3947 - accuracy: 0.8063 - val_loss: 0.1581 - val_accu
racy: 0.9523
Epoch 2/25
17361/17361 [==============================] - 252s 15ms/sample - loss: 0.1576 - accuracy: 0.9512 - val_loss: 0.1814 - val_accu
racy: 0.9497
Epoch 3/25
17361/17361 [==============================] - 244s 14ms/sample - loss: 0.1368 - accuracy: 0.9559 - val_loss: 0.1452 - val_accu
racy: 0.9539
Epoch 4/25
17361/17361 [==============================] - 247s 14ms/sample - loss: 0.1204 - accuracy: 0.9587 - val_loss: 0.1351 - val_accu
racy: 0.9632
Epoch 5/25
17361/17361 [==============================] - 250s 14ms/sample - loss: 0.1019 - accuracy: 0.9653 - val_loss: 0.1427 - val_accu
racy: 0.9585
Epoch 6/25
17361/17361 [==============================] - 240s 14ms/sample - loss: 0.0860 - accuracy: 0.9714 - val_loss: 0.1747 - val_accu
racy: 0.9580
Epoch 7/25
17361/17361 [==============================] - 209s 12ms/sample - loss: 0.0514 - accuracy: 0.9829 - val_loss: 0.1802 - val_accu
racy: 0.9616
```



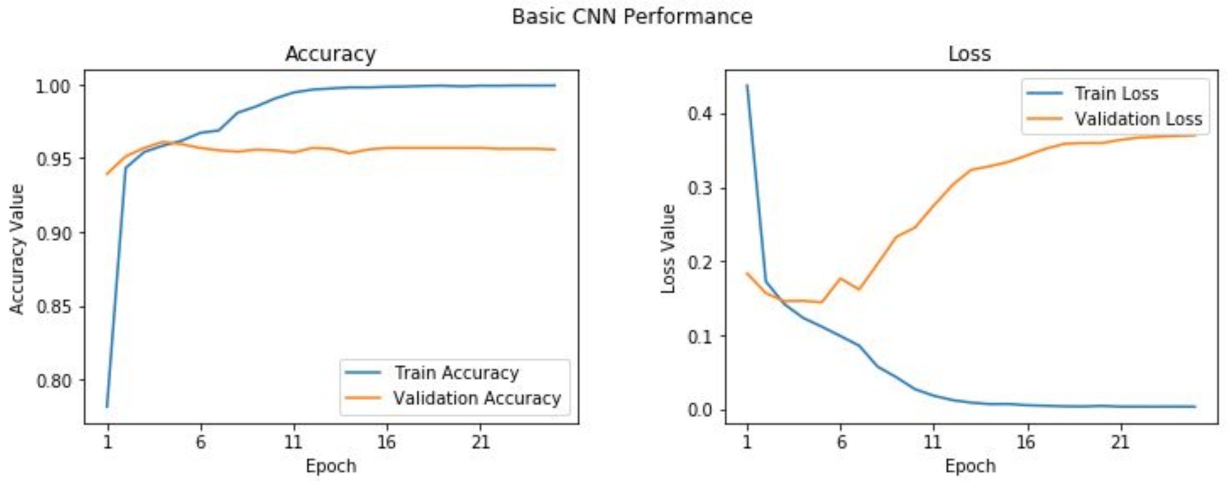
```
cy: 0.9570
Epoch 21/25
17361/17361 [==============================] - 30s 2ms/sample - loss: 0.0035 - accuracy: 0.9993 - val_loss: 0.3638 - val_accura
cy: 0.9570
Epoch 22/25
17361/17361 [==============================] - 30s 2ms/sample - loss: 0.0035 - accuracy: 0.9992 - val_loss: 0.3669 - val_accura
cy: 0.9565
Epoch 23/25
17361/17361 [==============================] - 30s 2ms/sample - loss: 0.0035 - accuracy: 0.9994 - val_loss: 0.3681 - val_accura
cy: 0.9565
Epoch 24/25
17361/17361 [==============================] - 30s 2ms/sample - loss: 0.0036 - accuracy: 0.9993 - val_loss: 0.3693 - val_accura
cy: 0.9565
Epoch 25/25
17361/17361 [==============================] - 30s 2ms/sample - loss: 0.0034 - accuracy: 0.9994 - val_loss: 0.3699 - val_accura
cy: 0.9559
```

```
In [14]: f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
         t = f.suptitle('Basic CNN Performance', fontsize=12)
         f.subplots_adjust(top=0.85, wspace=0.3)

         max_epoch = len(history.history['accuracy'])+1
         epoch_list = list(range(1,max_epoch))
         ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
         ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accuracy')
         ax1.set_xticks(np.arange(1, max_epoch, 5))
         ax1.set_ylabel('Accuracy Value')
         ax1.set_xlabel('Epoch')
         ax1.set_title('Accuracy')
         l1 = ax1.legend(loc="best")
```

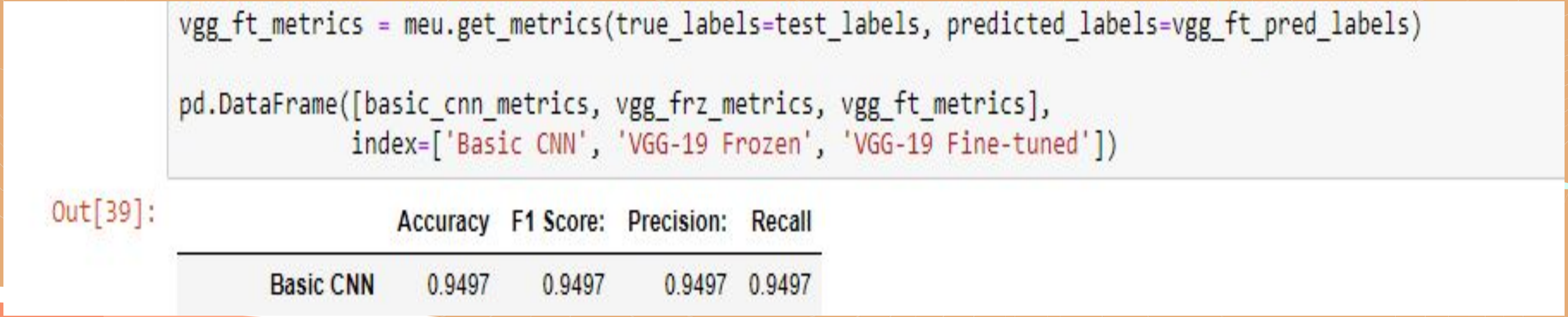
CNN PERFORMANCE



```
ax2.set_xticks(np.arange(1, max_epoch, 5))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")
```



MODEL PERFORMANCE EVALUATION

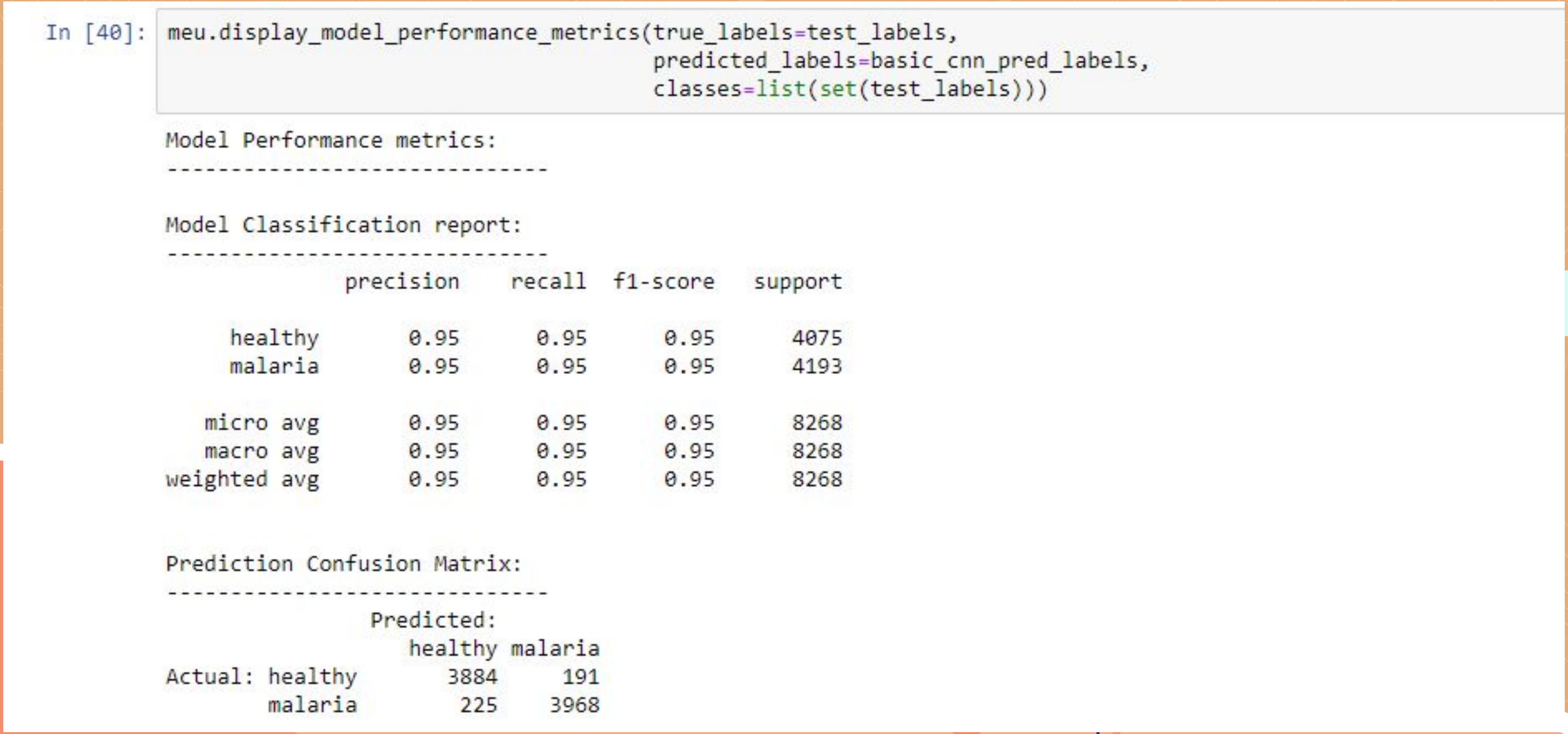


```
vgg_ft_metrics = meu.get_metrics(true_labels=test_labels, predicted_labels=vgg_ft_pred_labels)

pd.DataFrame([basic_cnn_metrics, vgg_frz_metrics, vgg_ft_metrics],
             index=['Basic CNN', 'VGG-19 Frozen', 'VGG-19 Fine-tuned'])
```

Out[39]:

	Accuracy	F1 Score:	Precision:	Recall
Basic CNN	0.9497	0.9497	0.9497	0.9497

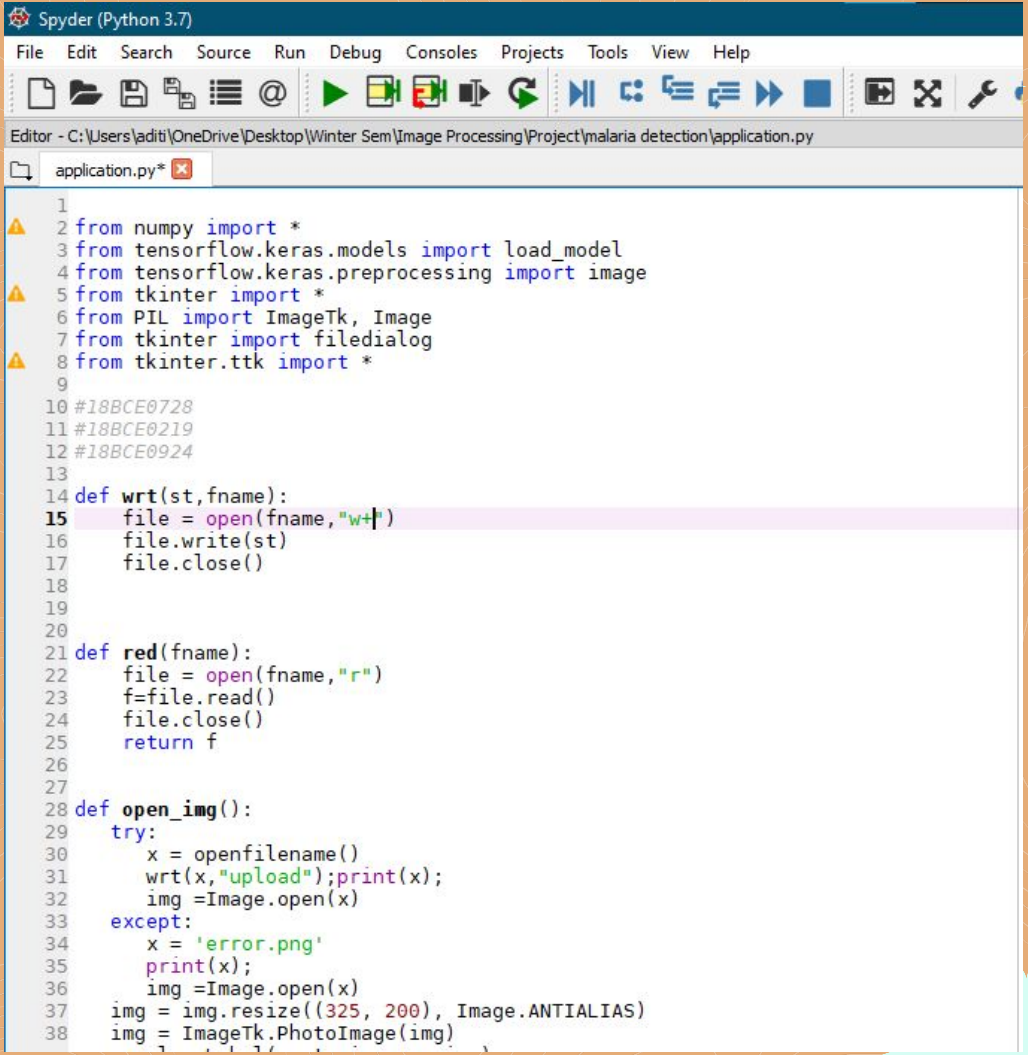
MODEL PERFORMANCE METRICS



```
In [40]: meu.display_model_performance_metrics(true_labels=test_labels,
                                               predicted_labels=basic_cnn_pred_labels,
                                               classes=list(set(test_labels)))
```

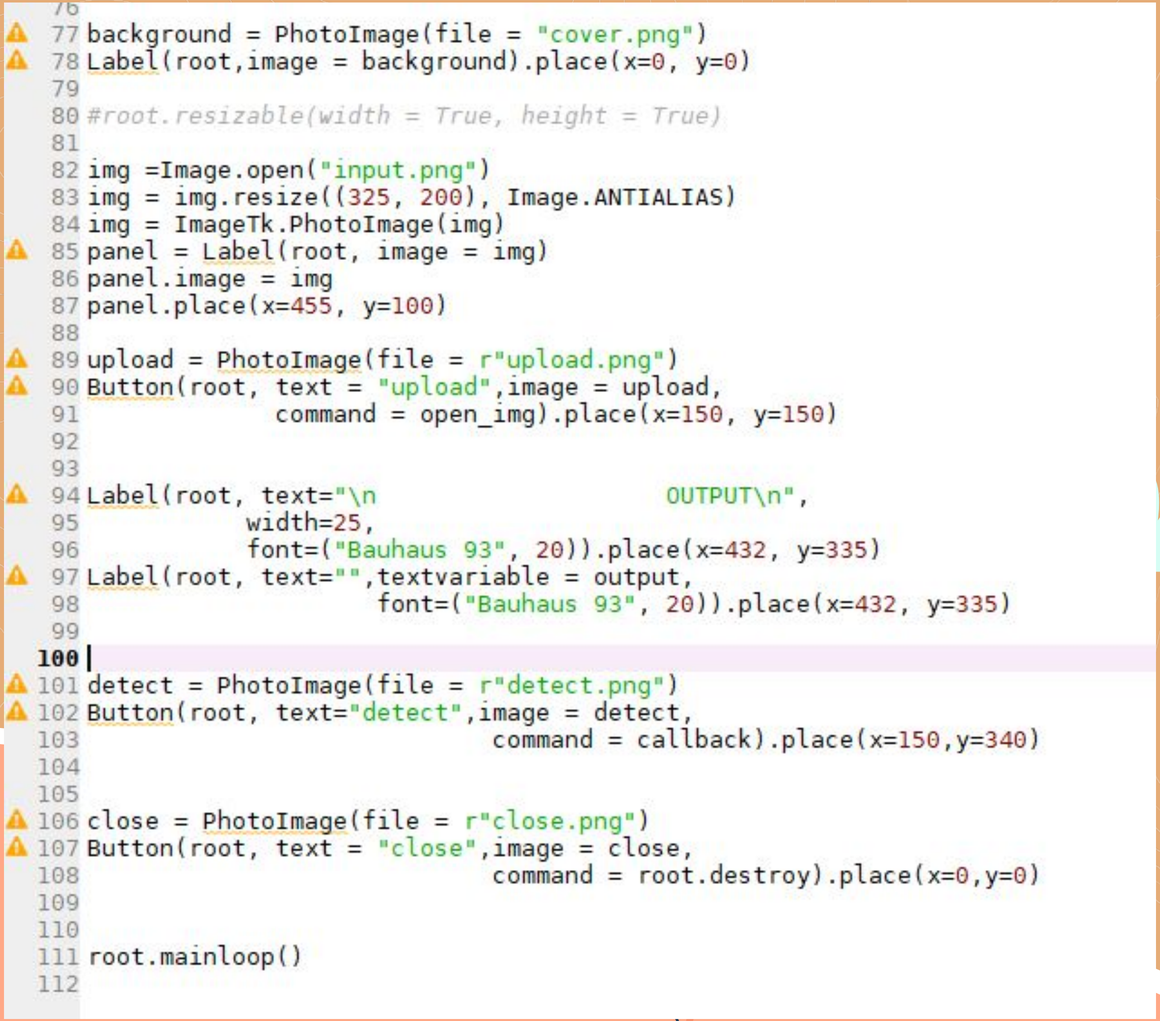
```
Model Performance metrics:
------------------------------

Model Classification report:
------------------------------
              precision    recall  f1-score   support

     healthy       0.95      0.95      0.95      4075
     malaria       0.95      0.95      0.95      4193

   micro avg       0.95      0.95      0.95      8268
   macro avg       0.95      0.95      0.95      8268
weighted avg       0.95      0.95      0.95      8268


Prediction Confusion Matrix:
------------------------------
                Predicted:
                   healthy malaria
Actual: healthy       3884     191
        malaria        225    3968
```

GUI CODE


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\aditi\OneDrive\Desktop\Winter Sem\Image Processing\Project\malaria detection\application.py
application.py*

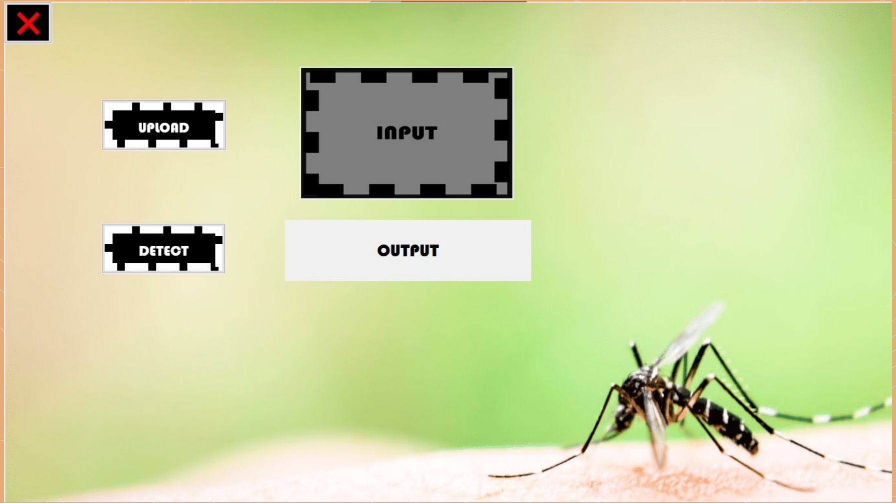
```

from numpy import *
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tkinter import *
from PIL import ImageTk, Image
from tkinter import filedialog
from tkinter.ttk import *

#18BCE0728
#18BCE0219
#18BCE0924

def wrt(st,fname):
    file = open(fname,"w+")
    file.write(st)
    file.close()



def red(fname):
    file = open(fname,"r")
    f=file.read()
    file.close()
    return f


def open_img():
   try:
      x = openfilename()
      wrt(x,"upload");print(x);
      img =Image.open(x)
   except:
      x = 'error.png'
      print(x);
      img =Image.open(x)
   img = img.resize((325, 200), Image.ANTIALIAS)
   img = ImageTk.PhotoImage(img)
```

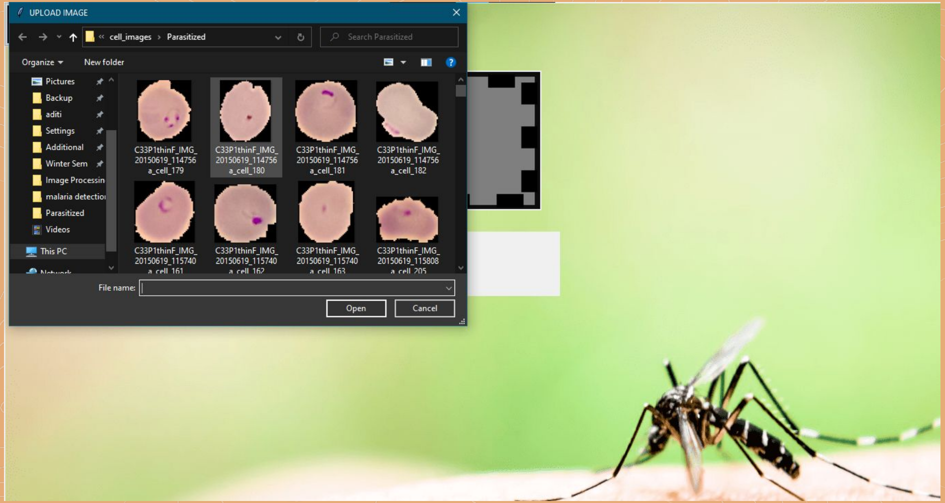
GUI CODE



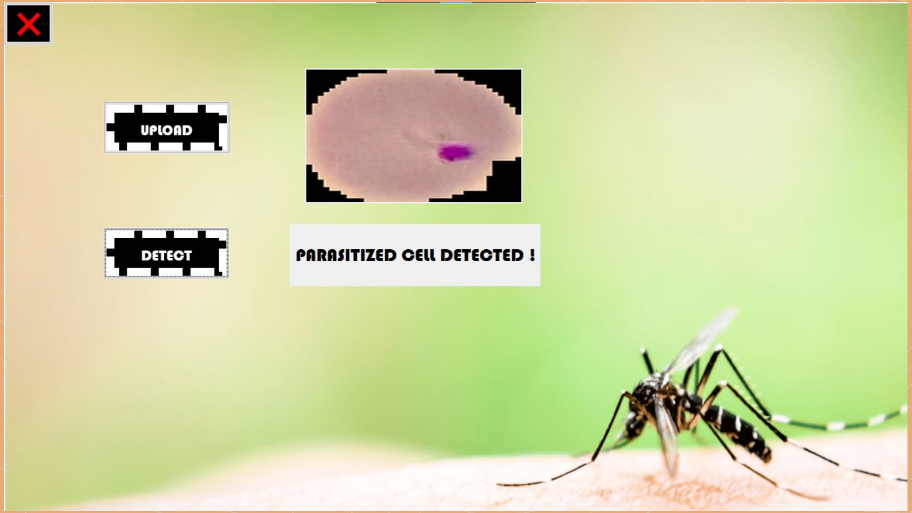
```
background = PhotoImage(file = "cover.png")
Label(root,image = background).place(x=0, y=0)

#root.resizable(width = True, height = True)

img =Image.open("input.png")
img = img.resize((325, 200), Image.ANTIALIAS)
img = ImageTk.PhotoImage(img)
panel = Label(root, image = img)
panel.image = img
panel.place(x=455, y=100)

upload = PhotoImage(file = r"upload.png")
Button(root, text = "upload",image = upload,
             command = open_img).place(x=150, y=150)


Label(root, text="\n                    OUTPUT\n",
           width=25,
           font=("Bauhaus 93", 20)).place(x=432, y=335)
Label(root, text="",textvariable = output,
                    font=("Bauhaus 93", 20)).place(x=432, y=335)


detect = PhotoImage(file = r"detect.png")
Button(root, text="detect",image = detect,
                            command = callback).place(x=150,y=340)


close = PhotoImage(file = r"close.png")
Button(root, text = "close",image = close,
                            command = root.destroy).place(x=0,y=0)


root.mainloop()

```

GUI

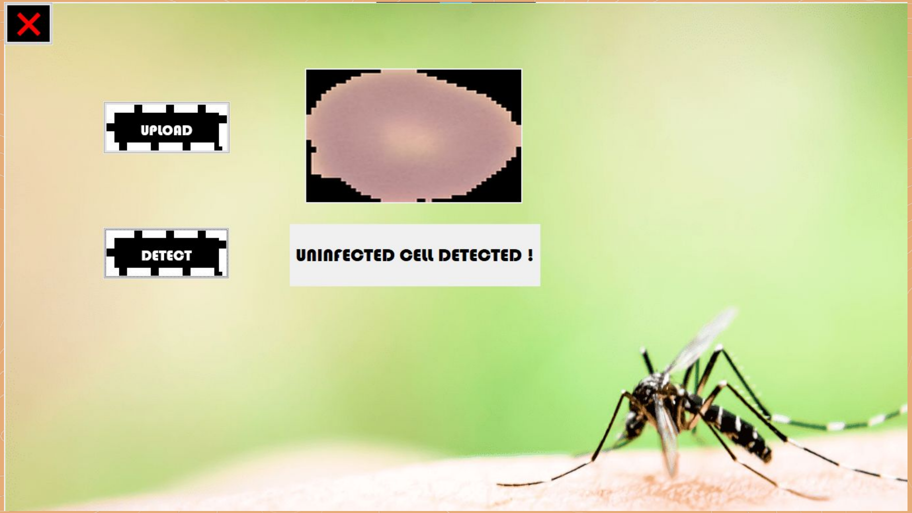

UPLOAD
INPUT
DETECT
OUTPUT


UPLOAD IMAGE
« cell_images › Parasitized
Search Parasitized
Organize
New folder
Pictures
Backup
aditi
Settings
Additional
Winter Sem
Image Processin
malaria detectio
Parasitized
Videos
This PC
Network
C33P1thinF_IMG_20150619_114756a_cell_179
C33P1thinF_IMG_20150619_114756a_cell_180
C33P1thinF_IMG_20150619_114756a_cell_181
C33P1thinF_IMG_20150619_114756a_cell_182
C33P1thinF_IMG_20150619_115740a_cell_161
C33P1thinF_IMG_20150619_115740a_cell_162
C33P1thinF_IMG_20150619_115740a_cell_163
C33P1thinF_IMG_20150619_115808a_cell_205
File name:
Open
Cancel

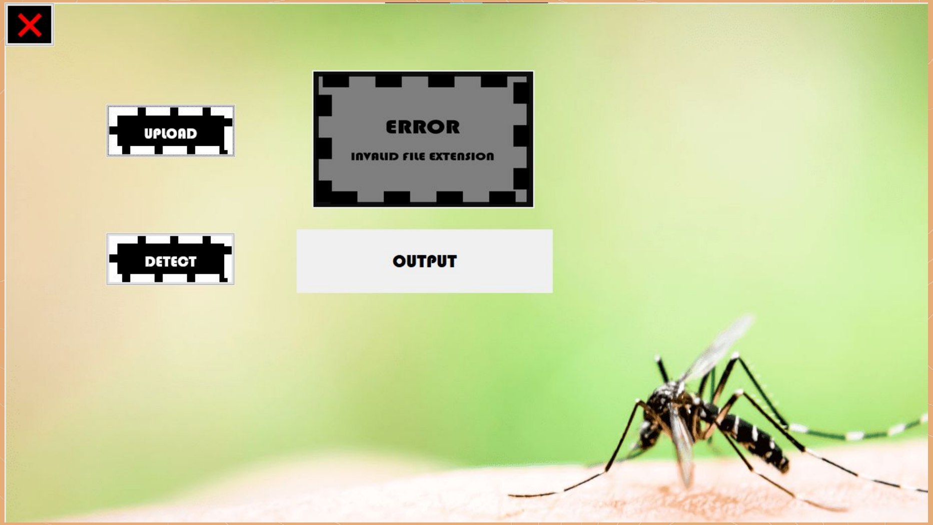
GUI

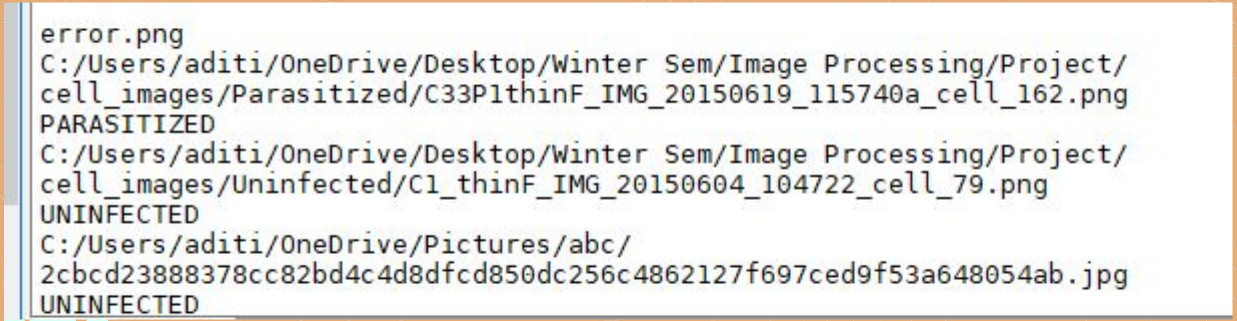


PARASITIZED CELL DETECTED !

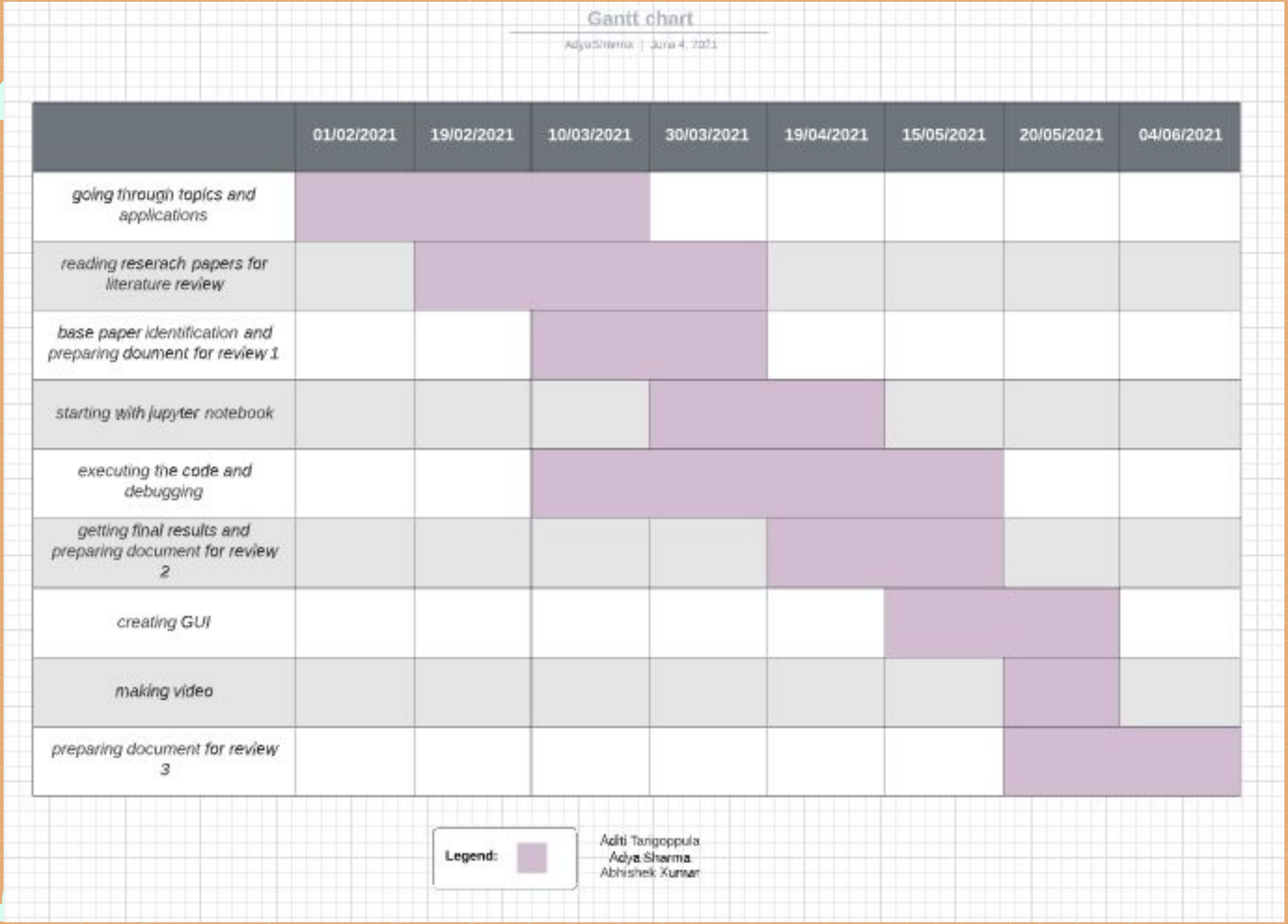


UNINFECTED CELL DETECTED !





```
error.png
C:/Users/aditi/OneDrive/Desktop/Winter Sem/Image Processing/Project/
cell_images/Parasitized/C33P1thinF_IMG_20150619_115740a_cell_162.png
PARASITIZED
C:/Users/aditi/OneDrive/Desktop/Winter Sem/Image Processing/Project/
cell_images/Uninfected/C1_thinF_IMG_20150604_104722_cell_79.png
UNINFECTED
C:/Users/aditi/OneDrive/Pictures/abc/
2cbcd23888378cc82bd4c4d8dfcd850dc256c4862127f697ced9f53a648054ab.jpg
UNINFECTED
```



Gantt chart

AdyaSharma | June 4, 2021

	01/02/2021	19/02/2021	10/03/2021	30/03/2021	19/04/2021	15/05/2021	20/05/2021	04/06/2021
going through topics and applications	▨	▨	▨					
reading reserach papers for literature review		▨	▨	▨				
base paper identification and preparing doument for review 1			▨	▨				
starting with jupyter notebook				▨	▨			
executing the code and debugging			▨	▨	▨	▨		
getting final results and preparing document for review 2					▨	▨		
creating GUI						▨	▨	
making video							▨	
preparing document for review 3							▨	▨

Legend: ▨

Aditi Tangoppula
Adya Sharma
Abhishek Kumar

CONCLUSION AND FUTURE WORK

Malarial parasite detection in patients nowadays is very expensive. Our model surpasses most previously developed models in a range of the accuracy metrics. The model has an advantage of being constructed from a relatively small number of layers. This reduces the computer resources and computational time. The reduction in computer resources and computational time makes it a cost effective methodology. Moreover, we test our model on two types of datasets and argue that the currently developed deep-learning-based methods cannot efficiently distinguish between infected and contaminated cells. A more precise study of suspicious regions is required.

In the future we plan to make an actual web Application and further improve accuracy Of our model.

REFERENCES

[1]Rajaraman, S., Antani, S. K., Poostchi, M., Silamut, K., Hossain, M. A., Maude, R. J., ... & Thoma, G. R. (2018). Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. PeerJ, 6, e4568.

[2]https://lhncbc.nlm.nih.gov/LHC-downloads/dataset.html

THANK YOU 😊