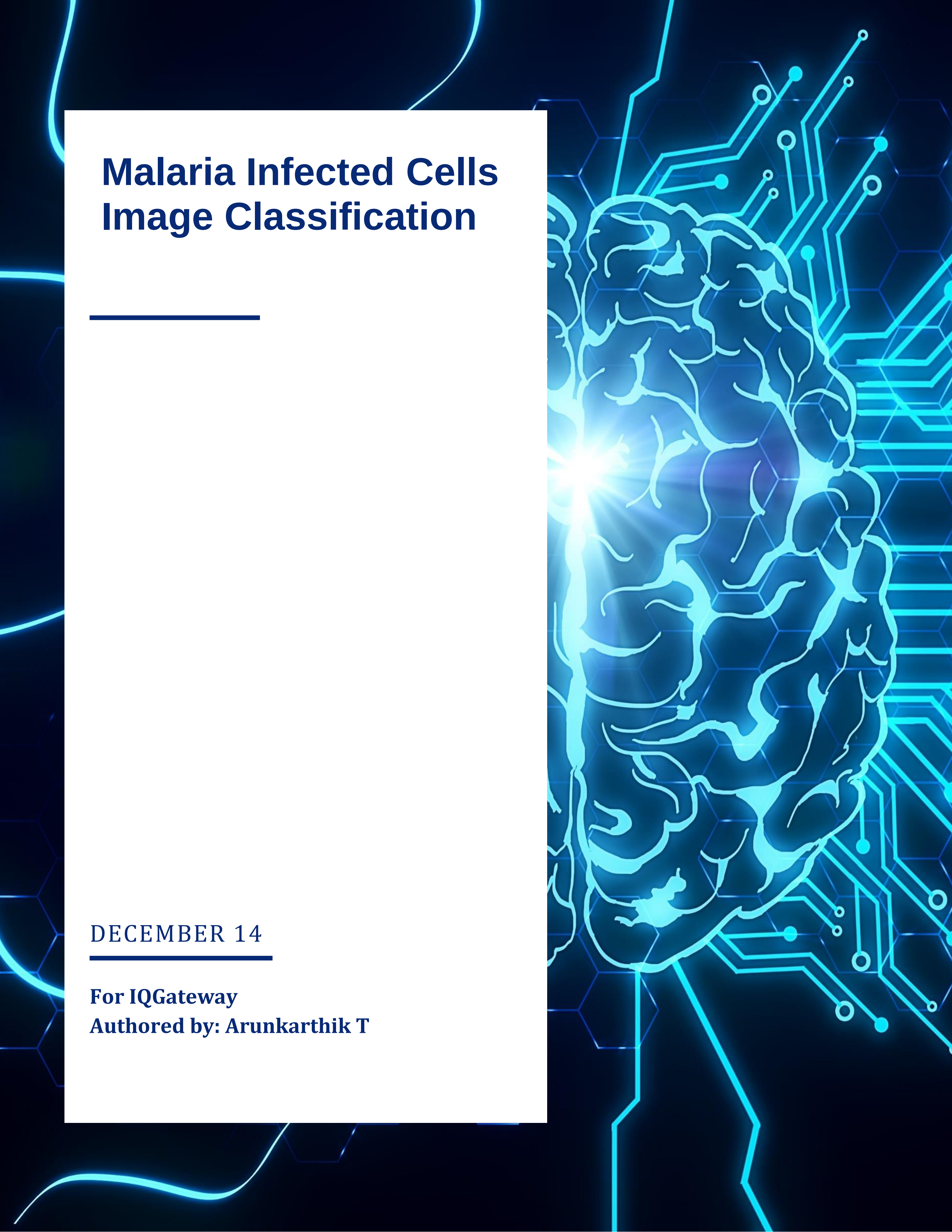


Malaria Infected Cells Image Classification

DECEMBER 14

For IQGateway
Authored by: Arunkarthik T

Classifying the parasitized and uninfected malaria blood sample images

Classification using Convolutional Neural Network a Machine Learning Model

Table of contents:

1. Objective of the study

As per World Health Organization, Malaria is one of the one of the life-threatening diseases and a big threat to humans. P. falciparum malaria is potential to cause death within 24 hours to the patients. So, it is critical to identify the trace of malaria at the early stages. Not only the early detection is critical but the prediction accuracy of the detection even more significant. Wrong medications might lead to serious side effects and cause damages on the patient's body. Conventionally, blood samples were tested manually by the lab technicians and classify the same with tedious counting process. It was time consuming and ineffective. So, researchers suggested to automate this process using machine learning algorithms. Convolutional neural networks (CNN) are the widely used effective deep learning algorithm used for efficient image classification. CNN has a tunable hyper parameter which allows the neural network to analyses the patterns in a much deeper level. Careful selection of hyper parameters will result in high accuracy classification models.

2. Methodology

Malaria dataset has two classes(folders) namely Parasitized and Uninfected as sown in figure 2.1 (Sample image shown for reference)

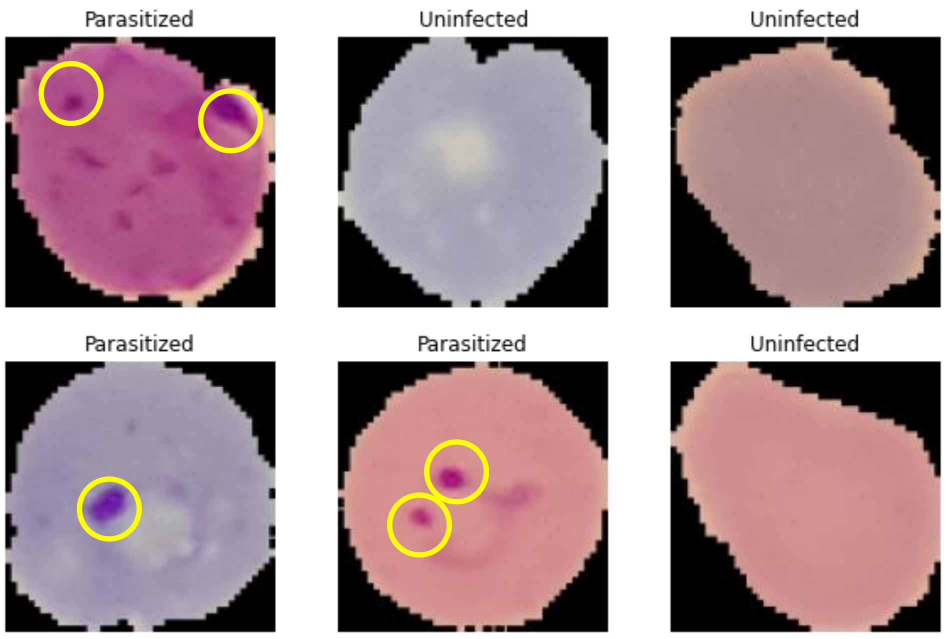


Fig. 2. 1 Parasitized and Uninfected images

Highlighted spots in Fig. 2.1 are the parasite traces CNN needs to learn and understand. CNN has to be modelled and trained carefully using proper parameters. This modelling and training process along with other steps are called image classification. Python programing language is used for this image classification, TensorFlow, Keras, Pandas, NumPy, OS, PIL, CV2 and Matplotlib libraries are used. Streamlit web framework is used to deploy the trained model in web-based platform. Below approach is used for the whole image classification process.

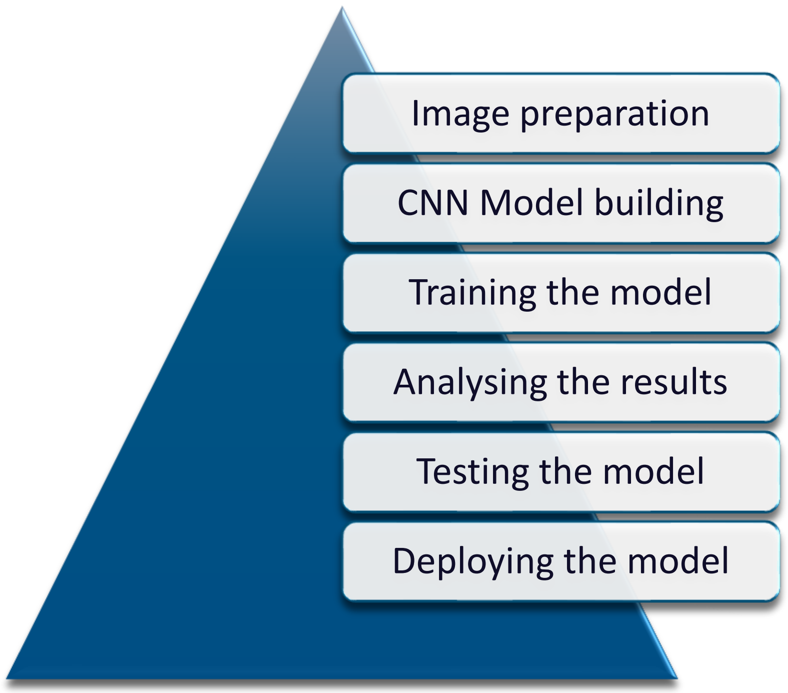


Fig. 2. 2 Methodology of image classification

2.1 Image Preparation

Images needed to be prepared with normalized scaling, similar image sizes under training, testing and validation categories. Below table shows the minimum, maximum and mean of the image sizes. Fig. 2.3 illustrates the minimum, maximum and mean of the images in parasitized and uninfected classes. Though the size of parasitized images are 1.5 times higher than the uninfected images, the mean seems to be closer. This figure used for finding the optimum resizing value. Since the value of the width is closer to 130, 128 seems to be the right choice to have the power of 2 values. All the images were resized in to 128 x 128 pixels.

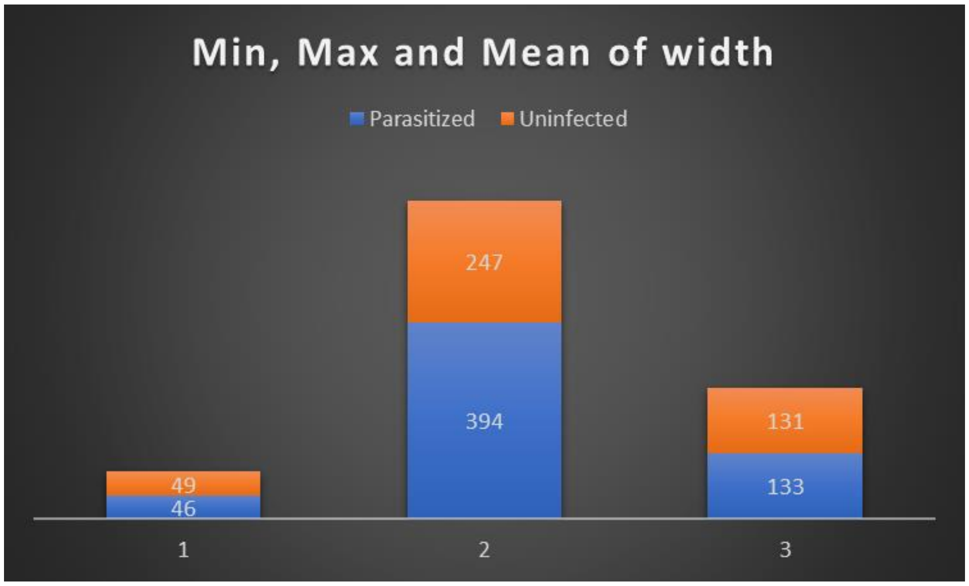


Fig. 2. 3 Min, Max and Mean of width

Training, testing and validation data were split as shown in figure 2.4. Np.random was used to randomize the images and Shutil library was used for copying the random images into train, test and validation folder.

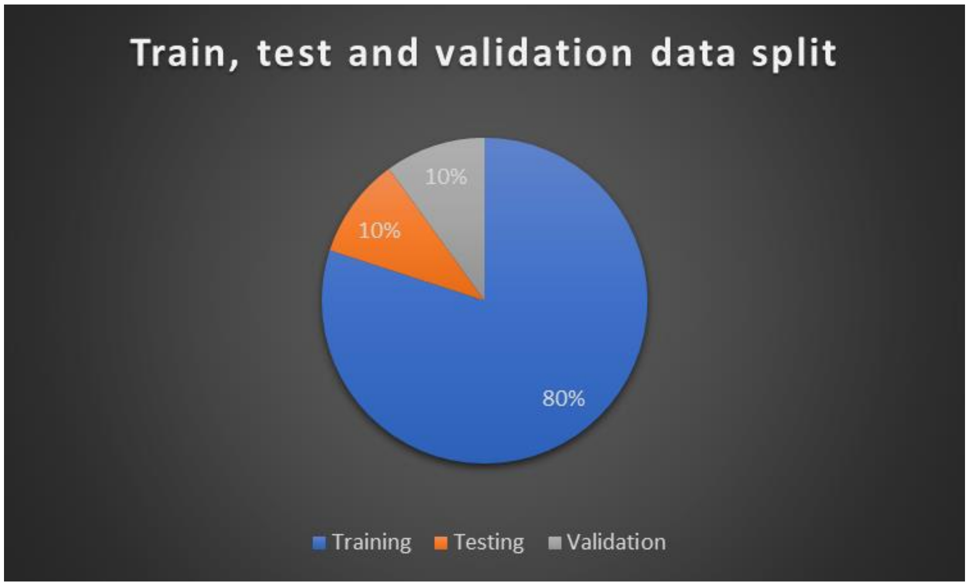


Fig. 2. 4 Train, test and validation split

Images were normalized from 1 to 255 for helping the deep learning model to compare the images with values within the boundary using ImageGenerator function of Keras.

2.2 CNN Model Building

CNN model has the architecture as shown in fig 2.5.

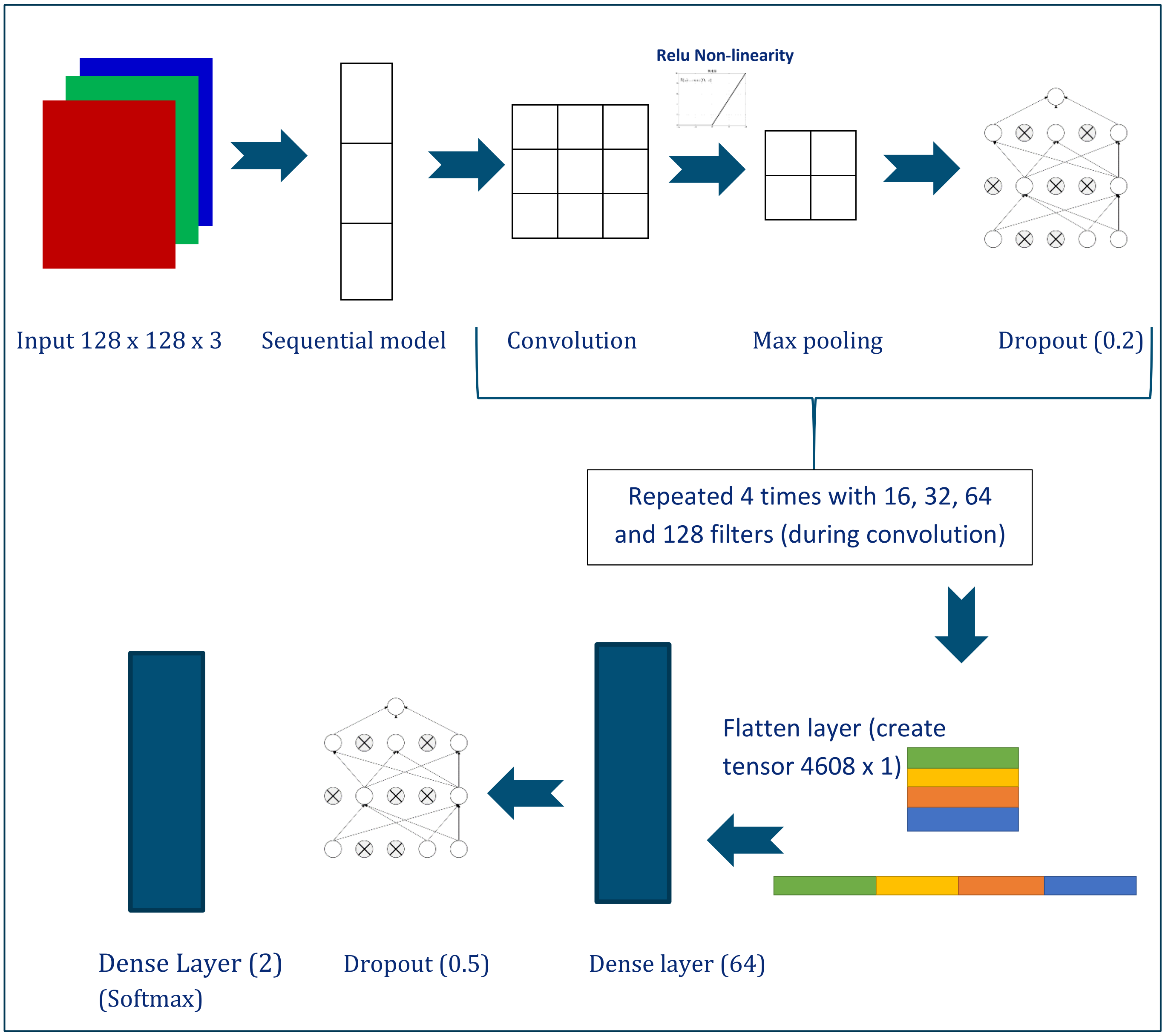

Relu Non-linearity
Input 128 x 128 x 3
Sequential model
Convolution
Max pooling
Dropout (0.2)
Repeated 4 times with 16, 32, 64 and 128 filters (during convolution)
Flatten layer (create tensor 4608 x 1)
Dense Layer (2) (Softmax)
Dropout (0.5)
Dense layer (64)

Fig. 2. 5 Architecture of CNN model

Training data input was programmed as below:

```
train_generator = train_datagen.flow_from_directory(directory= trainFolder,
                                                     target_size=(128, 128),
                                                     class_mode='binary',
                                                     subset='training',
                                                    shuffle=True,
                                                     batch_size=32
                               )

valid_generator = train_datagen.flow_from_directory(directory= valFolder,
                                                      target_size=(128, 128),
                                                     class_mode='binary',
                                                           shuffle = True,
                                                     subset='validation',
                                                     batch_size=32,

                                                     )

classes = ['Parasitized', 'Uninfected']
```

3 Training the model

History is the traditional keyword used for training the model. 20 epochs were used and 462 steps were considered in each epoch. Verbose was enabled to check the status of the model while training. Training accuracy observed was 95%. Validation accuracy also seems to be closer to the training accuracy. This shows the model was designed with the correct hyper parameters. The code for training the model is shown below.

```
history = ds_model.fit(train_generator,
                    epochs=20,
                    steps_per_epoch= len(train_generator),
                    validation_data = (valid_generator),
                    callbacks = [early_stop],
                    verbose=1
```

```
Epoch 1/20
462/462 [==============================] - 74s 159ms/step - loss: 0.6461 - acc
uracy: 0.6155 - val_loss: 0.4466 - val_accuracy: 0.8502
Epoch 2/20
462/462 [==============================] - 76s 163ms/step - loss: 0.3703 - acc
uracy: 0.8823 - val_loss: 0.1827 - val_accuracy: 0.9427
Epoch 3/20
462/462 [==============================] - 79s 170ms/step - loss: 0.2345 - acc
uracy: 0.9282 - val_loss: 0.1448 - val_accuracy: 0.9438
Epoch 4/20
462/462 [==============================] - 76s 165ms/step - loss: 0.2171 - acc
uracy: 0.9337 - val_loss: 0.1351 - val_accuracy: 0.9570
```

```
Epoch 5/20
462/462 [==============================] - 76s 164ms/step - loss: 0.2027 - acc
uracy: 0.9372 - val_loss: 0.1305 - val_accuracy: 0.9537
Epoch 6/20
462/462 [==============================] - 76s 165ms/step - loss: 0.2008 - acc
uracy: 0.9427 - val_loss: 0.1220 - val_accuracy: 0.9581
Epoch 7/20
462/462 [==============================] - 77s 167ms/step - loss: 0.1882 - acc
uracy: 0.9460 - val_loss: 0.1194 - val_accuracy: 0.9648
Epoch 8/20
462/462 [==============================] - 80s 174ms/step - loss: 0.1856 - acc
uracy: 0.9478 - val_loss: 0.1085 - val_accuracy: 0.9659
Epoch 9/20
462/462 [==============================] - 81s 176ms/step - loss: 0.1741 - acc
uracy: 0.9512 - val_loss: 0.1250 - val_accuracy: 0.9648
Epoch 10/20
462/462 [==============================] - 84s 181ms/step - loss: 0.1694 - acc
uracy: 0.9527 - val_loss: 0.1231 - val_accuracy: 0.9615
```

4 Analyzing the results

CNN model was checked with the training and validation accuracy. It is evident that the validation accuracy starts at 84% and converging closer to the training accuracy. The trend seems to be promising as shown in fig. 4.1.

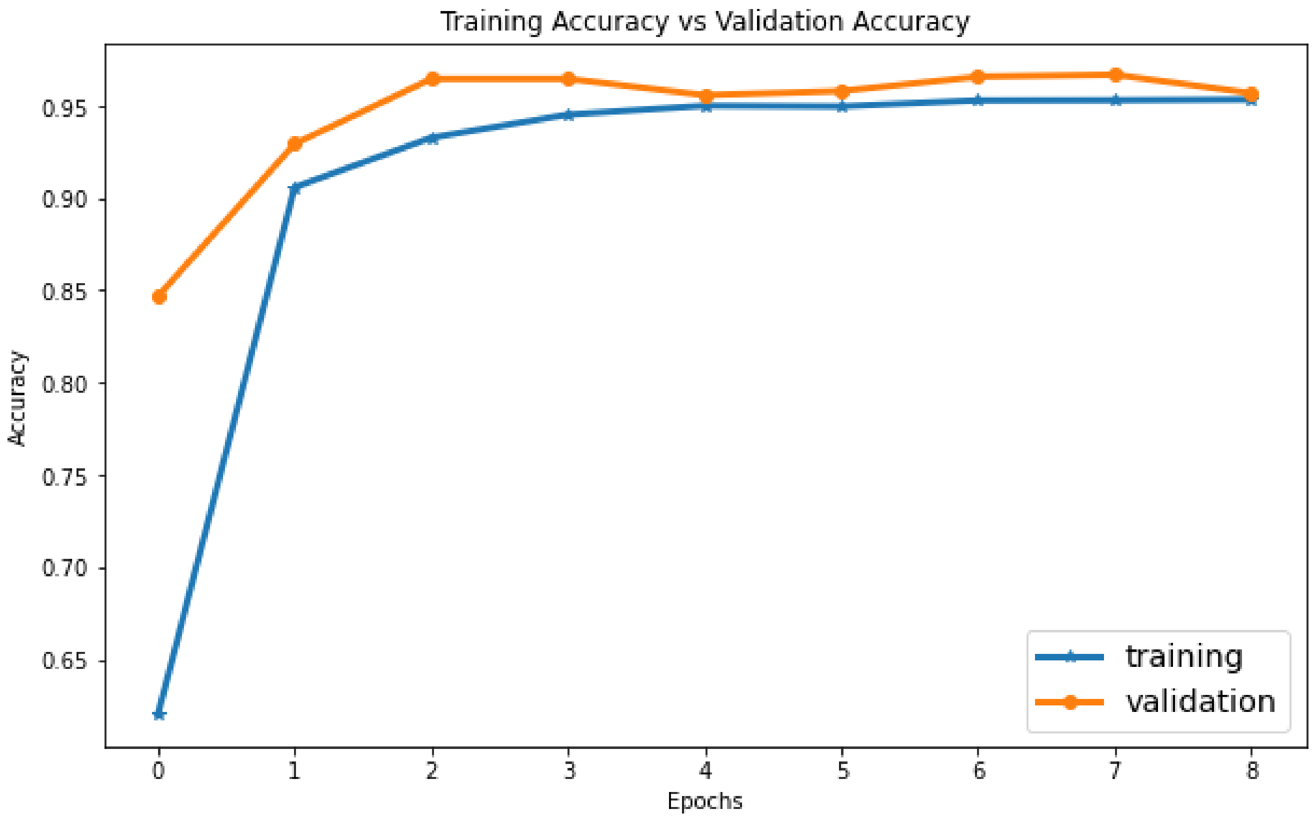


Fig. 4. 1 Training accuracy vs validation accuracy

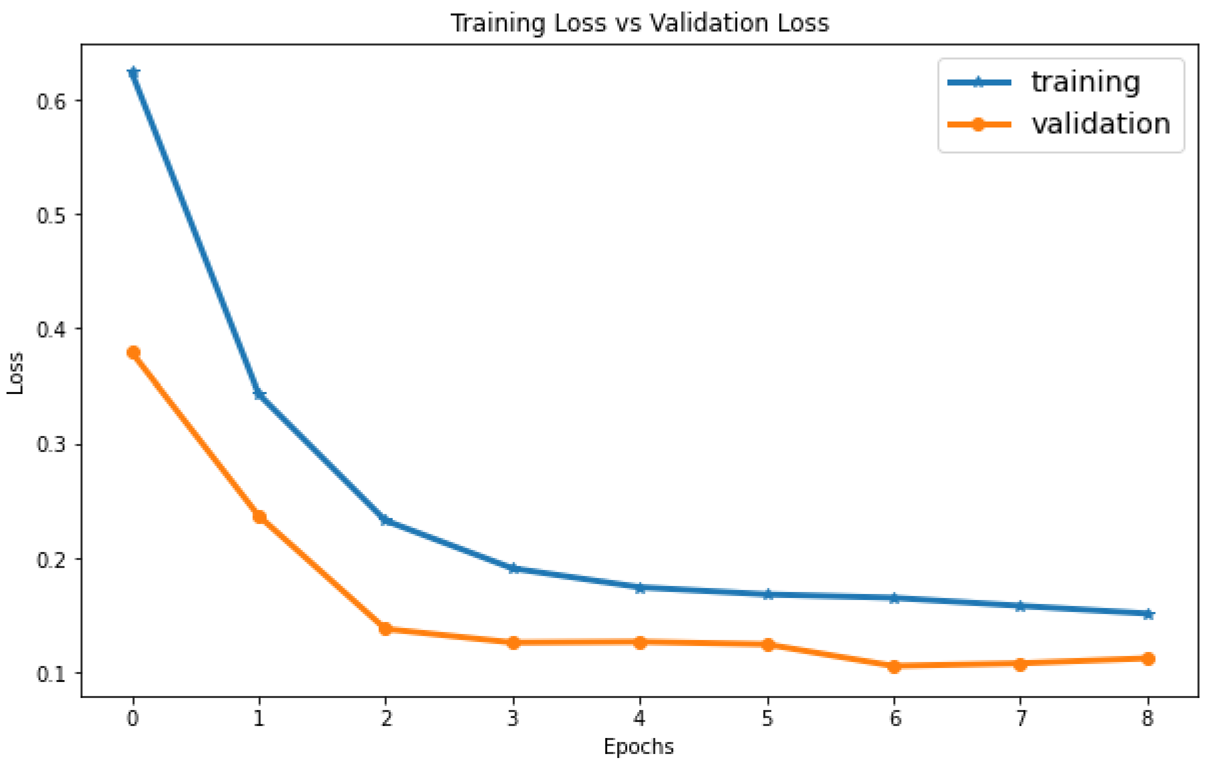
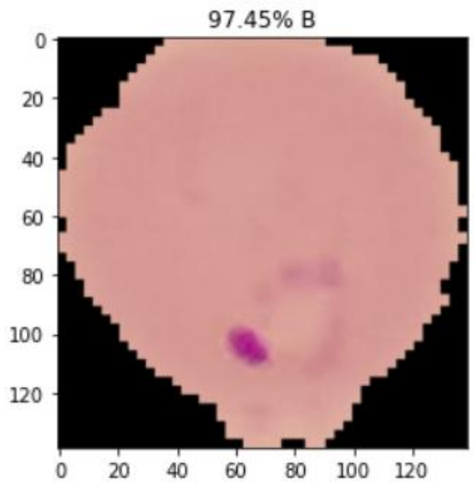


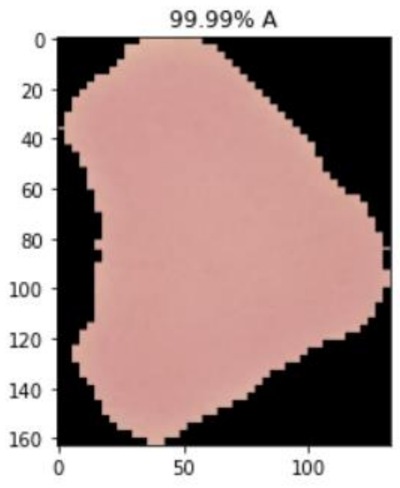
Fig. 4. 2 Training loss vs validation loss

Since there is only two classes to be predicted, binarycrossentropy loss function wile compiling the model. This seems to be promising as shown in fig. 4.2.

5. Testing the model

Model prediction was performed after rescaling the input images. Sample prediction results are shown below. And the prediction accuracy of the trained model is varying from 76% to 99%.





6. Deploying the model

Model was successfully deployed in Streamlit web app, perhaps the file of uploaded image was not available as a path string file. And PIL image resizing was not working properly in the model. So this attempt became the failure.

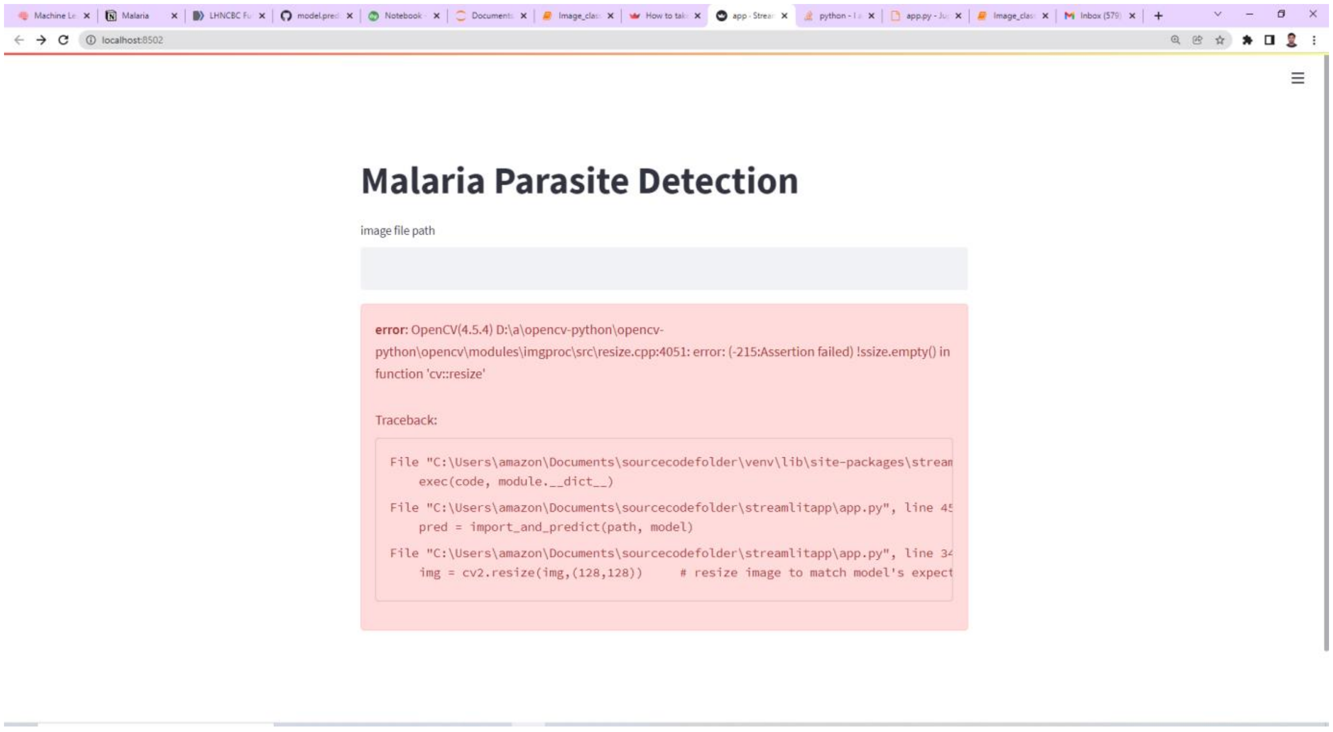
Code and the snippets for reference.

```
import streamlit as st
import tensorflow as tf
import streamlit as st
import cv2
import numpy as np
import os
import pandas

from PIL import Image, ImageOps
@st.cache(allow_output_mutation=True)

def load_model():
    model=tf.keras.models.load_model('myMalaria_Model.hdf5')
    return model
with st.spinner('Model is being loaded..'):
    model=load_model()

st.write("""
        # Malaria Parasite Detection
        """
        )

st.set_option('deprecation.showfileUploaderEncoding', False)
print(file)

def import_and_predict(image_data, model):
    img = cv2.imread(image_data)
    img = cv2.resize(img,(128,128))     # resize image to match model's expected sizing
    img = img.reshape(1,128,128,3) # return the image with shaping that TF wants.
    image = np.array(img)
    image = image *(1./255)
    pred = ds_model.predict(image)
```

This function needs the path as a string file, but Streamlit could not produce the

```
    return pred

if file is None:
    st.text("Please upload an image file")
else:
    image = Image.open(file)
    st.image(image, use_column_width=True)
    pred = import_and_predict(dest, model)
    if pred>0.5:
        print("The blood cell is uninfected")
    else:
        print("The blood cell is Prasitized")
```

I tried using the path as text, the error pops up as error**: OpenCV(4.5.4) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\resize.cpp:4051: error: (-215:Assertion failed) !ssize.empty() in function 'cv::resize'.**



Malaria Parasite Detection

image file path

```
error: OpenCV(4.5.4) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\resize.cpp:4051: error: (-215:Assertion failed) !ssize.empty() in function 'cv::resize'

Traceback:

File "C:\Users\amazon\Documents\sourcecodefolder\venv\lib\site-packages\strean
    exec(code, module.__dict__)
File "C:\Users\amazon\Documents\sourcecodefolder\streamlitapp\app.py", line 45
    pred = import_and_predict(path, model)
File "C:\Users\amazon\Documents\sourcecodefolder\streamlitapp\app.py", line 34
    img = cv2.resize(img,(128,128))     # resize image to match model's expect
```

Streamlit code for text input:.

```
import streamlit as st
import tensorflow as tf
import streamlit as st
import cv2
import numpy as np
import os
import pandas

from PIL import Image, ImageOps
@st.cache(allow_output_mutation=True)

def load_model():
    model=tf.keras.models.load_model('myMalaria_Model.hdf5')
    return model
with st.spinner('Model is being loaded..'):
    model=load_model()

st.write("""
        # Malaria Parasite Detection
        """
        )



#file = st.text_input("label goes here", default_value_goes_here)

path = st.text_input('image file path')

st.set_option('deprecation.showfileUploaderEncoding', False)

def import_and_predict(image_data, model):
    model1 = model
    img = cv2.imread(image_data)
    img = cv2.resize(img,(128,128))     # resize image to match model's expected sizing
    img = img.reshape(1,128,128,3) # return the image with shaping that TF wants.
    image = np.array(img)
    image = image *(1./255)
    pred = ds_model.predict(image)
    st.image(image, use_column_width=True)
    return pred

if path is None:
    st.text("Please enter the path for image file")
else:
    pred = import_and_predict(path, model)
```

```
if pred>0.5:
    print("The blood cell is uninfected")
else:
    print("The blood cell is Prasitized")
```

Summary:
Classification model was successfully trained, predicted and tested. Deployment of the model attempt went to failure because of the image uploading issue.