DETECTING MALARIAL CELL WITH CNN

PDS3504 DEEP LEARNING PRACTICAL

HANNAH RUTH.J

19-PDS-005

1.INTRODUCTION

a) MALARIA

Malaria is a deadly, infectious mosquito-borne disease caused by Plasmodium parasites. These parasites are transmitted by the bites of infected female Anopheles mosquitoes. The process of identification includes -Microscopic thick and thin blood smear examinations are the most reliable and commonly used method for disease diagnosis. Thick blood smears assist in detecting the presence of parasites while thin blood smears assist in identifying the species of the parasite causing the infection (Centers for Disease Control and Prevention, 2012). The diagnostic accuracy heavily depends on human expertise and can be adversely impacted by the inter-observer variability and the liability imposed by large-scale diagnoses in disease-endemic/resource-constrained regions (Mitiku, Mengistu & Gelaw, 2003).

b) DATA COLLECTION

The Lister Hill National Center for Biomedical Communications (LHNCBC), part of National Library of Medicine (NLM) who have carefully collected and annotated this dataset of healthy and infected blood smear images. 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 balanced dataset of 13779 malaria and non-malaria cell images.

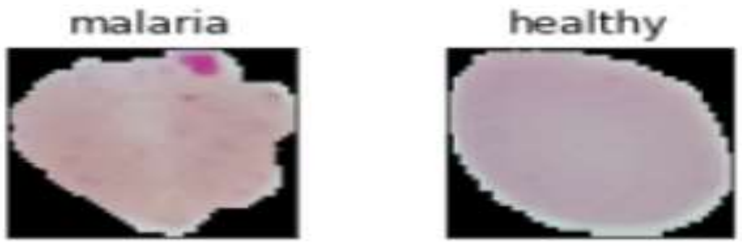


Figure1.1

malaria and non-malaria cell images

c) DEEP LEARNING MODEL PHASE

The dataset is exposed to data pre-processing means of normalization, labelling the images and splitting images into test and train sets in 90:10 ratio.

The CNN model is composed of 5 convolution layers,5 pooling layers,6 dropout layers and a fully connected layer.

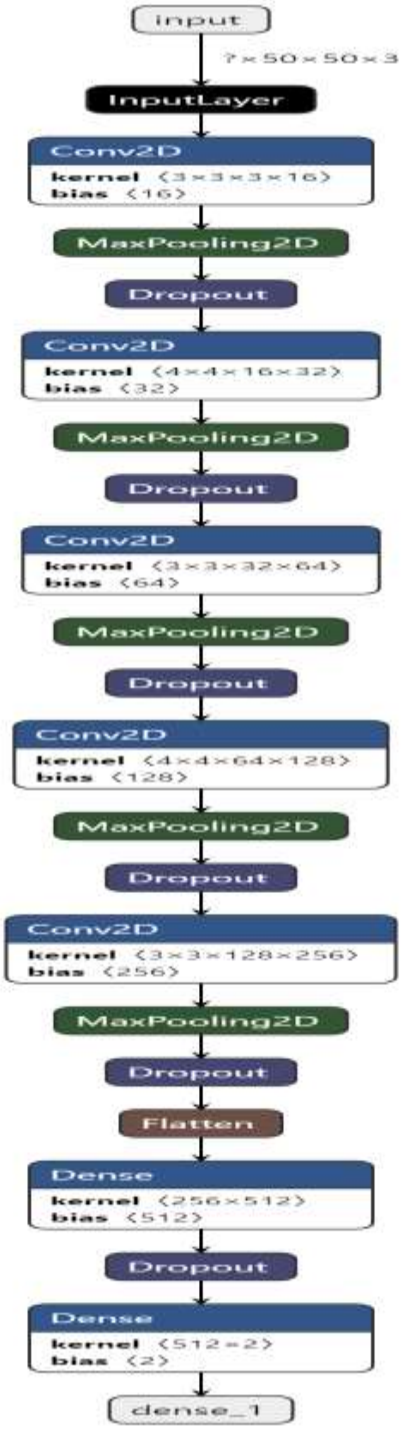


Figure 1.2

CNN Network

CONVOLUTION LAYER

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The activation function relu,the optimizer used is adam,loss function induced is binary cross entropy.

NON LINEARITY (RELU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0,x)$. Other non-linear functions such as tanh or sigmoid that can also be used instead of ReLU.

POOLING LAYER

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information.

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

DROPOUT LAYER

Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

FULLY CONNECTED LAYER

FC layer is a flattened matrix into vector and feed it into a fully connected layer like a neural network.

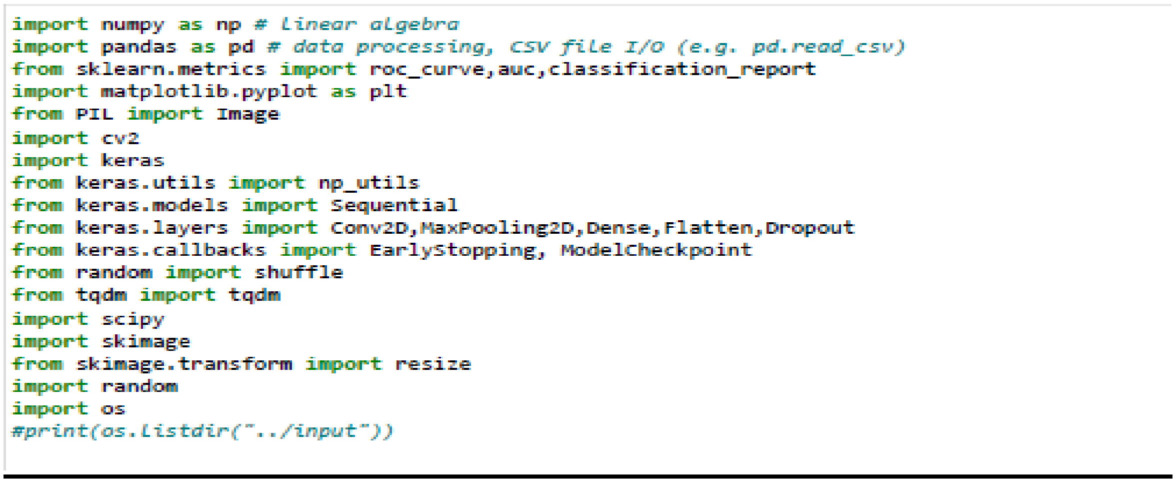
d)DEEP LEARNING MODEL EVALUATION:

Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

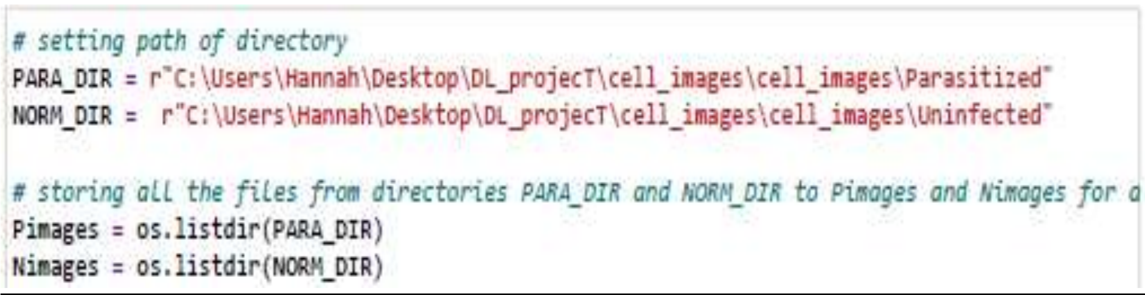
2. IMPLEMENTATION

- Importing Libraries for creating a sequential CNN network.



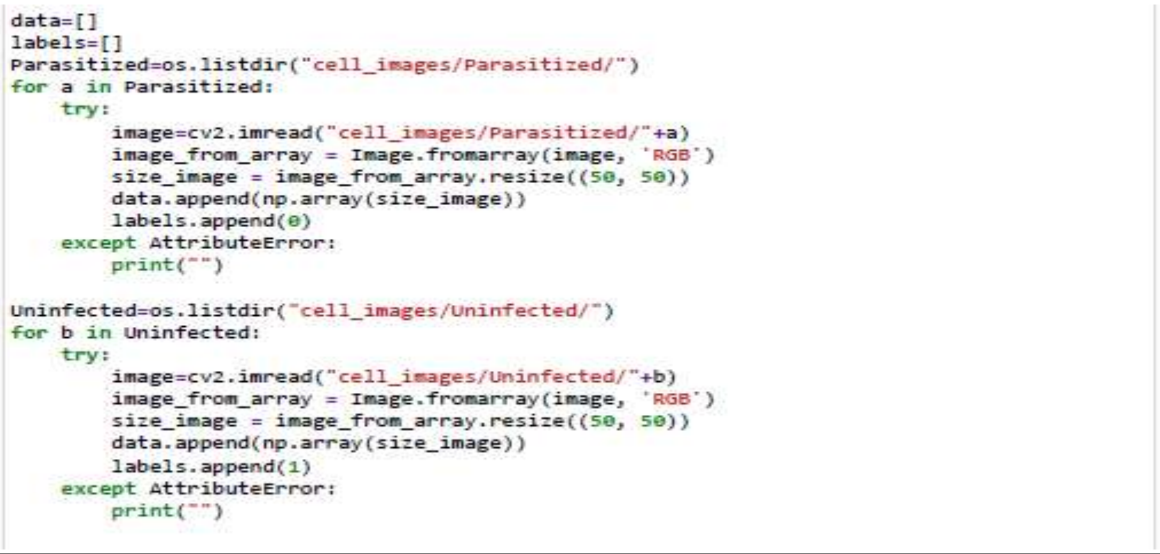
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.metrics import roc_curve,auc,classification_report
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import keras
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
from random import shuffle
from tqdm import tqdm
import scipy
import skimage
from skimage.transform import resize
import random
import os
#print(os.listdir("../input"))
```

- Loading Data and create separate directory to store Parasitized and Normal images.



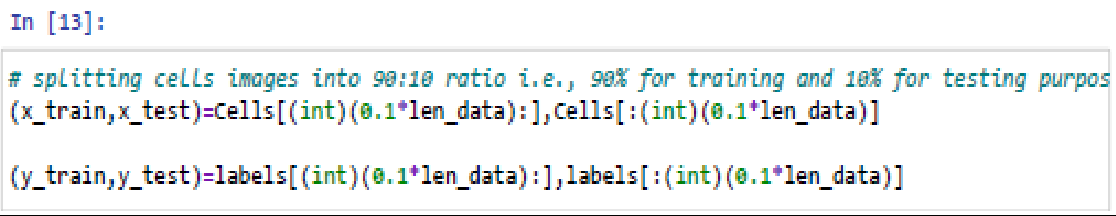
```
# setting path of directory
PARA_DIR = r"C:\Users\Hannah\Desktop\DL_projecT\cell_images\cell_images\Parasitized"
NORM_DIR =  r"C:\Users\Hannah\Desktop\DL_projecT\cell_images\cell_images\Uninfected"

# storing all the files from directories PARA_DIR and NORM_DIR to Pimages and Nimages for a
Pimages = os.listdir(PARA_DIR)
Nimages = os.listdir(NORM_DIR)
```

- Data pre-processing Labelling and resizing the image



```
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("")
```

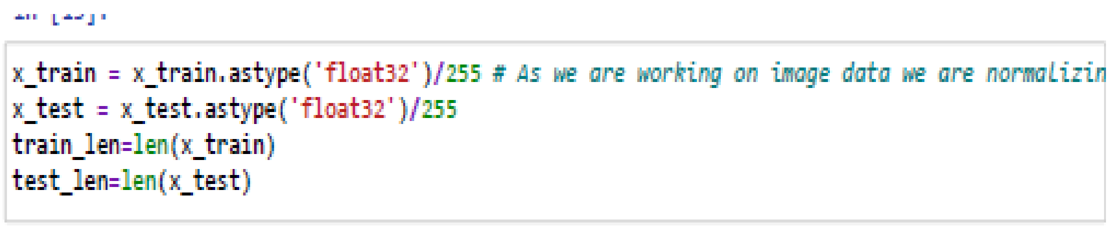
- Splitting data as train and test images



```
In [13]:
```

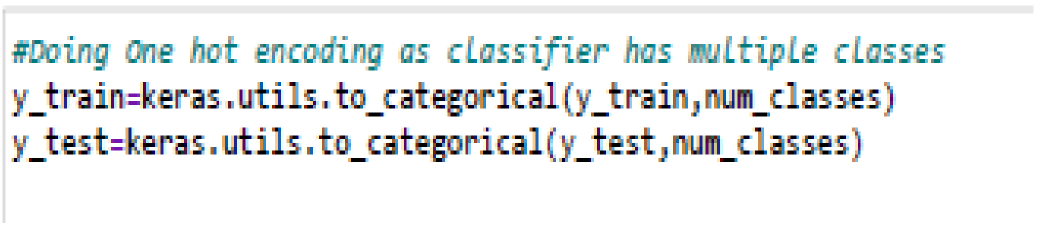
```
# splitting cells images into 90:10 ratio i.e., 90% for training and 10% for testing purpos
(x_train,x_test)=Cells[(int)(0.1*len_data):],Cells[:(int)(0.1*len_data)]

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

- Normalization of the dataset



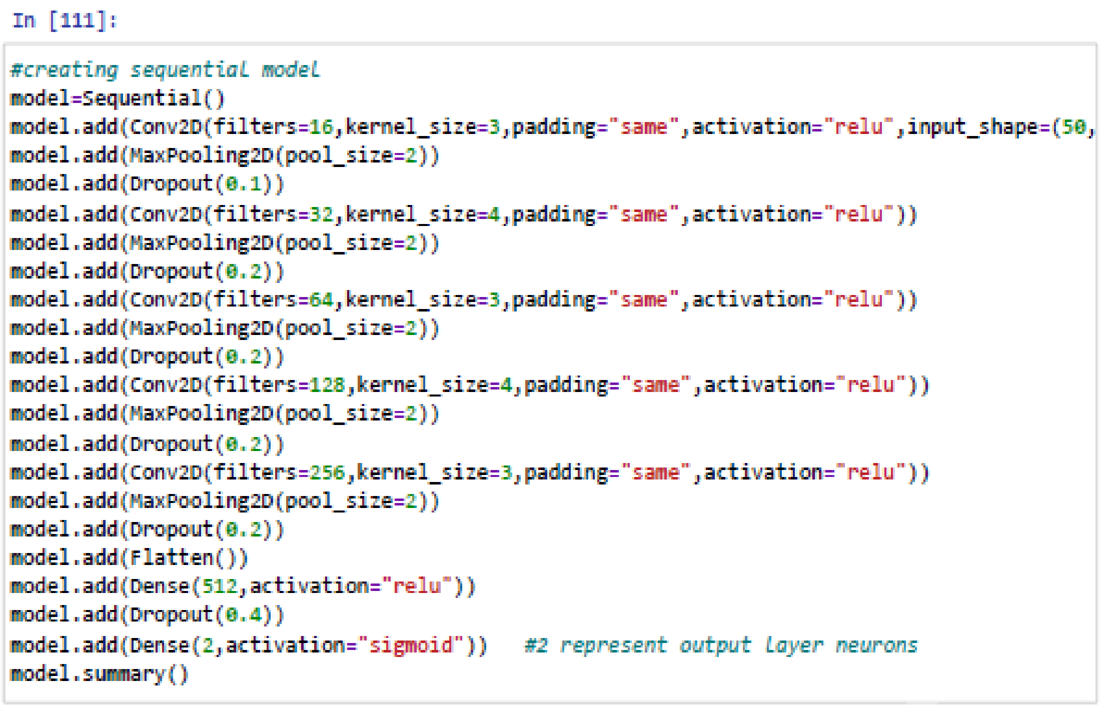
```
x_train = x_train.astype('float32')/255 # As we are working on image data we are normalizin
x_test = x_test.astype('float32')/255
train_len=len(x_train)
test_len=len(x_test)
```

- Label Encoding



```
#Doing One hot encoding as classifier has multiple classes
y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
```

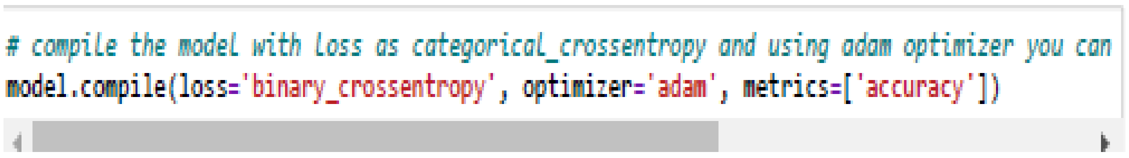
- CNN-Model



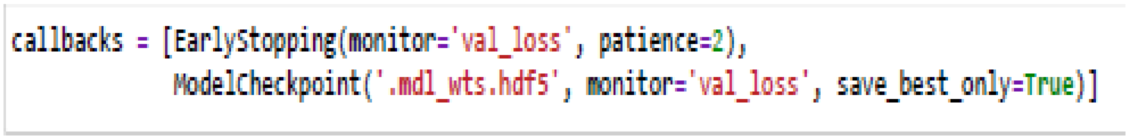
```
In [111]:
```

```
#creating sequential model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=3,padding="same",activation="relu",input_shape=(50,
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.1))
model.add(Conv2D(filters=32,kernel_size=4,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=64,kernel_size=3,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=128,kernel_size=4,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=256,kernel_size=3,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512,activation="relu"))
model.add(Dropout(0.4))
model.add(Dense(2,activation="sigmoid"))    #2 represent output layer neurons
model.summary()
```

- Optimization and callback of the model

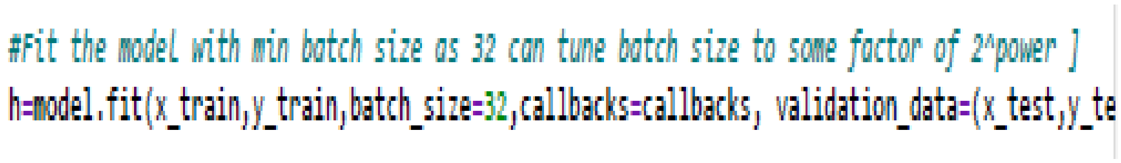


```
# compile the model with loss as categorical_crossentropy and using adam optimizer you can
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```



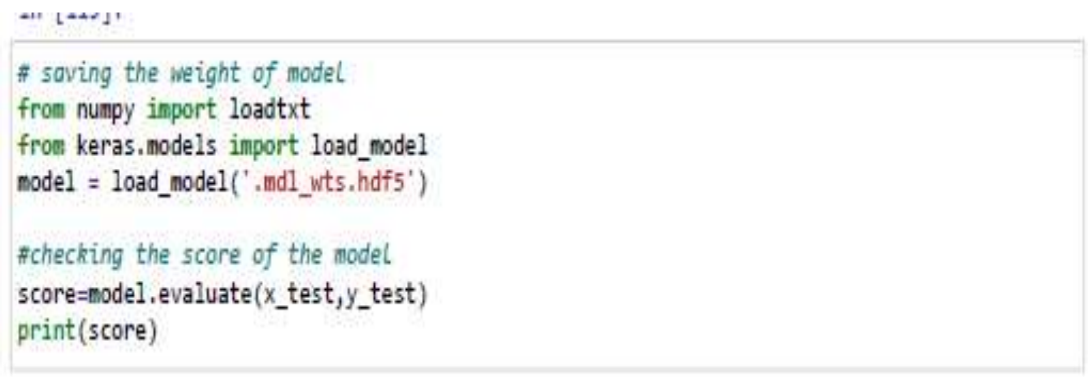
```
callbacks = [EarlyStopping(monitor='val_loss', patience=2),
             ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True)]
```

- Model Fitting



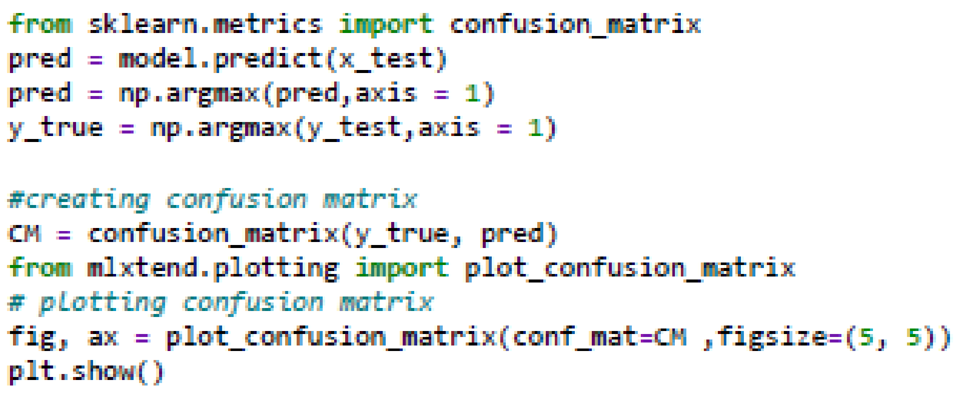
```
#Fit the model with min batch size as 32 can tune batch size to some factor of 2^power ]
h=model.fit(x_train,y_train,batch_size=32,callbacks=callbacks, validation_data=(x_test,y_te
```

- Model Evaluation



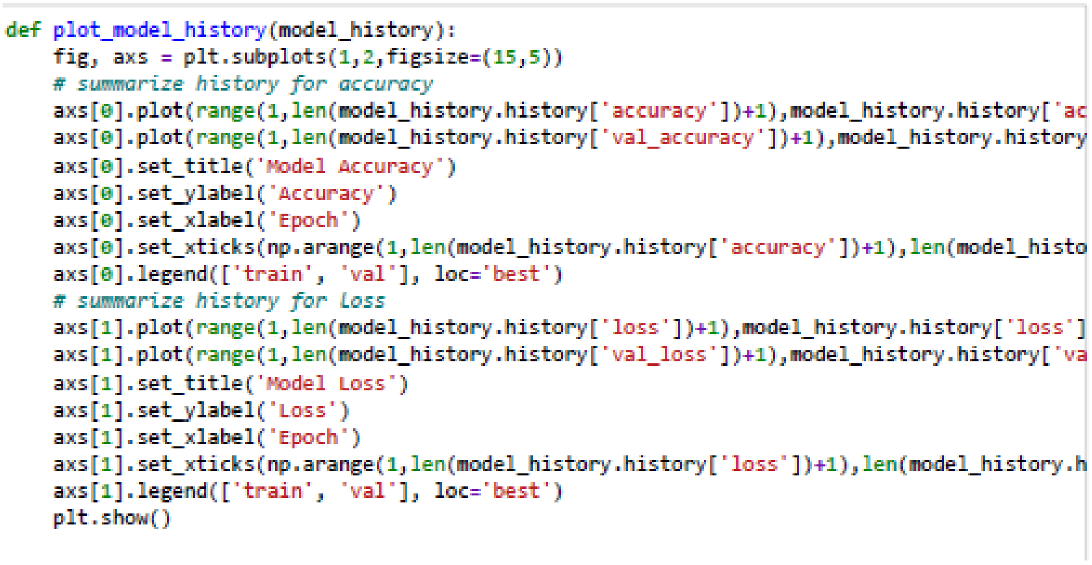
```
# saving the weight of model
from numpy import loadtxt
from keras.models import load_model
model = load_model('.mdl_wts.hdf5')

#checking the score of the model
score=model.evaluate(x_test,y_test)
print(score)
```

- Confusion Matrix



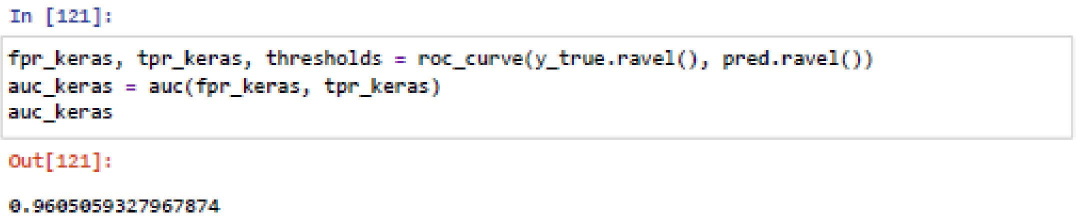
```
from sklearn.metrics import confusion_matrix
pred = model.predict(x_test)
pred = np.argmax(pred,axis = 1)
y_true = np.argmax(y_test,axis = 1)

#creating confusion matrix
CM = confusion_matrix(y_true, pred)
from mlxtend.plotting import plot_confusion_matrix
# plotting confusion matrix
fig, ax = plot_confusion_matrix(conf_mat=CM ,figsize=(5, 5))
plt.show()
```

- Plotting History of Model's Accuracy

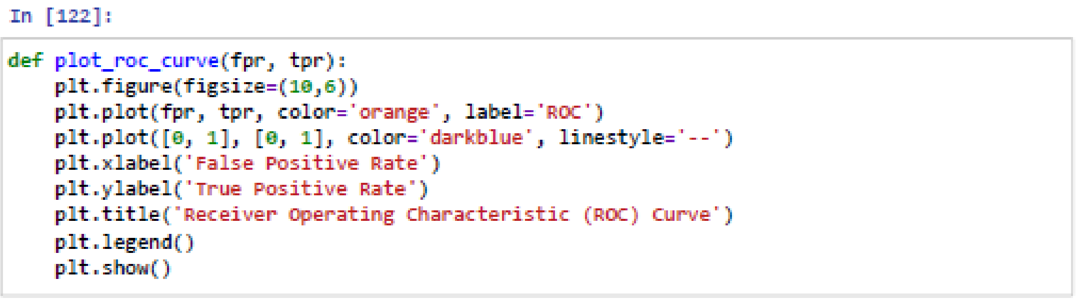


```
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['ac
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.history
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_histo
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss']
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['va
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.h
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()
```

- Plotting ROC AUC Curve

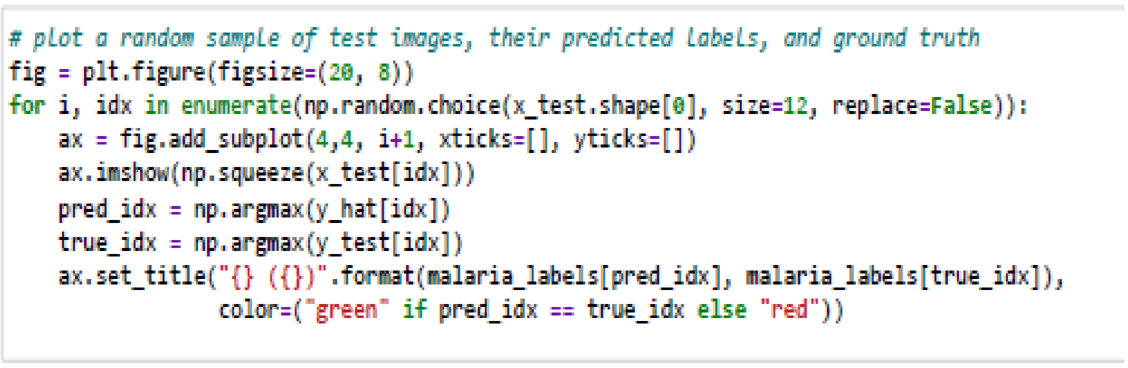


```
In [121]:

fpr_keras, tpr_keras, thresholds = roc_curve(y_true.ravel(), pred.ravel())
auc_keras = auc(fpr_keras, tpr_keras)
auc_keras

Out[121]:

0.9605059327967874
```



```
In [122]:

def plot_roc_curve(fpr, tpr):
    plt.figure(figsize=(10,6))
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

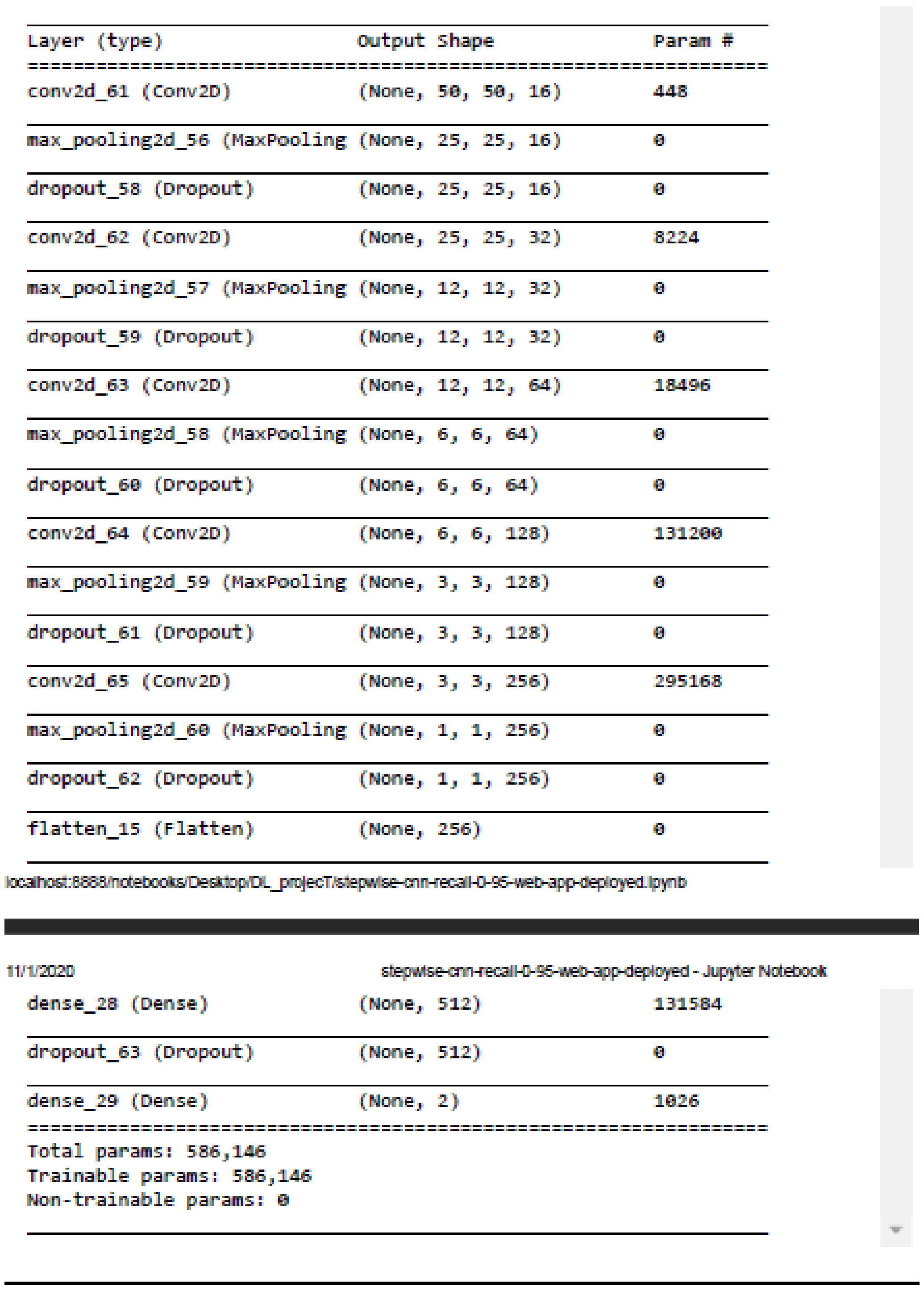
- Plotting Sample Prediction



```
# plot a random sample of test images, their predicted labels, and ground truth
fig = plt.figure(figsize=(20, 8))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=12, replace=False)):
    ax = fig.add_subplot(4,4, i+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_hat[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(malaria_labels[pred_idx], malaria_labels[true_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

3.OUTPUT

- Sequential model -5 convolution layers,5 pooling layers,6 dropout layers and a fully connected layer.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_61 (Conv2D)           (None, 50, 50, 16)        448
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 25, 25, 16)        0
_________________________________________________________________
dropout_58 (Dropout)         (None, 25, 25, 16)        0
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 25, 25, 32)        8224
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 12, 12, 32)        0
_________________________________________________________________
dropout_59 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_58 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_60 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 6, 6, 128)         131200
_________________________________________________________________
max_pooling2d_59 (MaxPooling (None, 3, 3, 128)         0
_________________________________________________________________
dropout_61 (Dropout)         (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 3, 3, 256)         295168
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 1, 1, 256)         0
_________________________________________________________________
dropout_62 (Dropout)         (None, 1, 1, 256)         0
_________________________________________________________________
flatten_15 (Flatten)         (None, 256)               0
_________________________________________________________________
```

localhost:8888/notebooks/Desktop/DL_projecT/stepwise-cnn-recall-0-95-web-app-deployed.ipynb

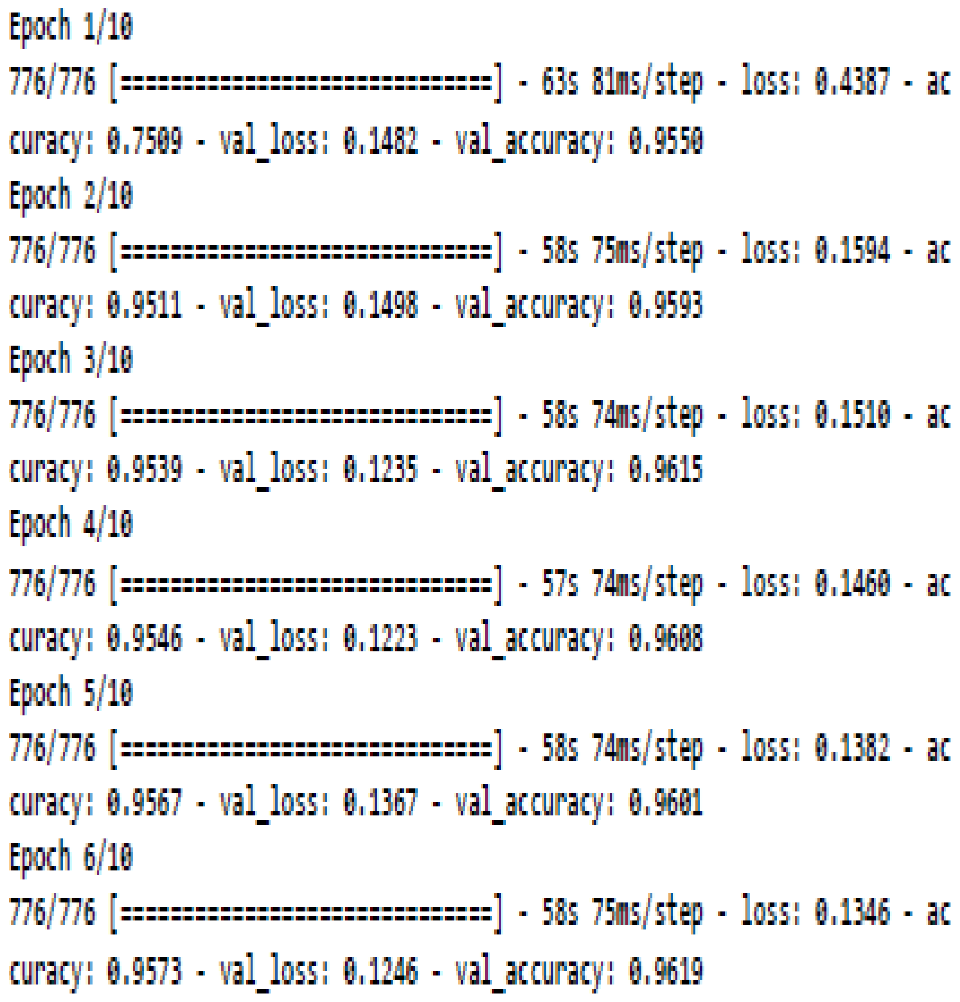
11/1/2020 stepwise-cnn-recall-0-95-web-app-deployed - Jupyter Notebook

```
dense_28 (Dense)             (None, 512)               131584
_________________________________________________________________
dropout_63 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_29 (Dense)             (None, 2)                 1026
=================================================================
Total params: 586,146
Trainable params: 586,146
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.1

CNN model parameters

- Fitting the model with min batch size as 32 and epoch 10

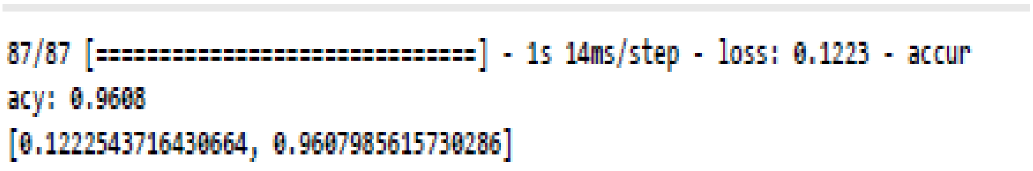


```
Epoch 1/10
776/776 [==============================] - 63s 81ms/step - loss: 0.4387 - ac
curacy: 0.7509 - val_loss: 0.1482 - val_accuracy: 0.9550
Epoch 2/10
776/776 [==============================] - 58s 75ms/step - loss: 0.1594 - ac
curacy: 0.9511 - val_loss: 0.1498 - val_accuracy: 0.9593
Epoch 3/10
776/776 [==============================] - 58s 74ms/step - loss: 0.1510 - ac
curacy: 0.9539 - val_loss: 0.1235 - val_accuracy: 0.9615
Epoch 4/10
776/776 [==============================] - 57s 74ms/step - loss: 0.1460 - ac
curacy: 0.9546 - val_loss: 0.1223 - val_accuracy: 0.9608
Epoch 5/10
776/776 [==============================] - 58s 74ms/step - loss: 0.1382 - ac
curacy: 0.9567 - val_loss: 0.1367 - val_accuracy: 0.9601
Epoch 6/10
776/776 [==============================] - 58s 75ms/step - loss: 0.1346 - ac
curacy: 0.9573 - val_loss: 0.1246 - val_accuracy: 0.9619
```

Figure 4.2

Sequential model output

- Model Evaluation-Model Performance with Test and Train dataset



```
87/87 [==============================] - 1s 14ms/step - loss: 0.1223 - accur
acy: 0.9608
[0.1222543716430664, 0.9607985615730286]
```

Figure 4.3

Accuracy Score of model

- Confusion Matrix

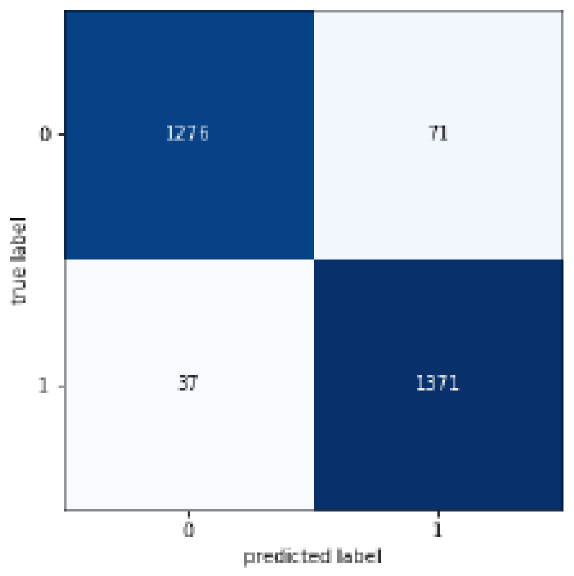


Figure 4.4

- Plotting History of Model's Accuracy

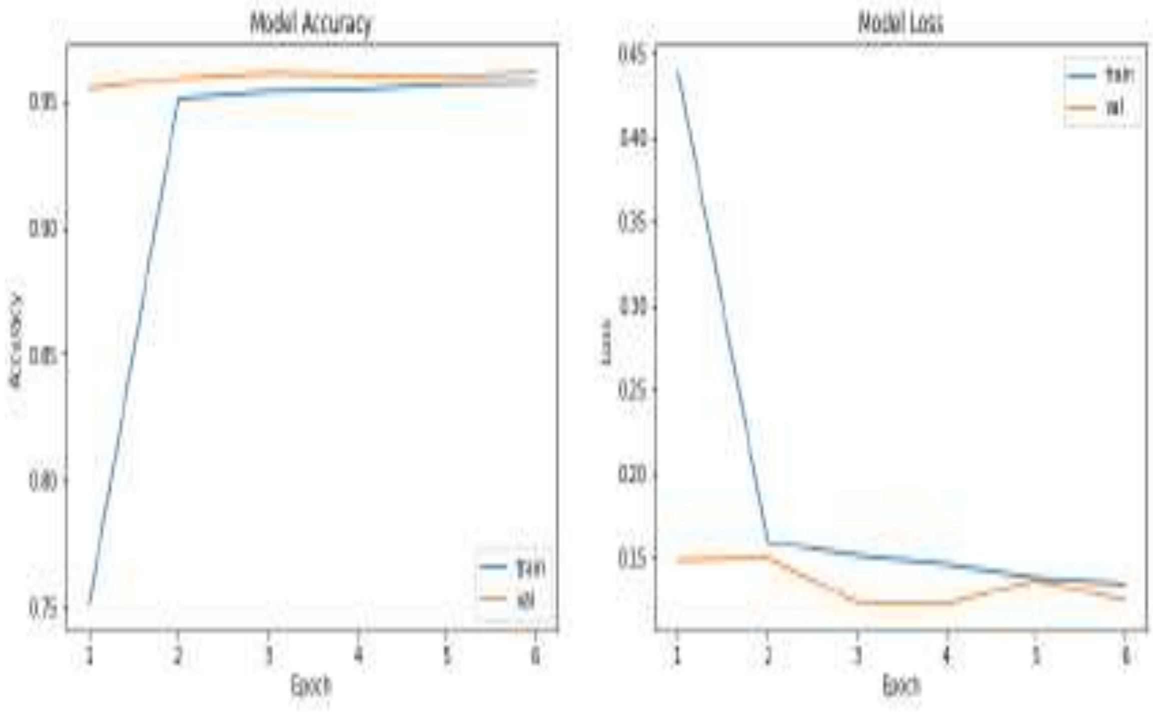
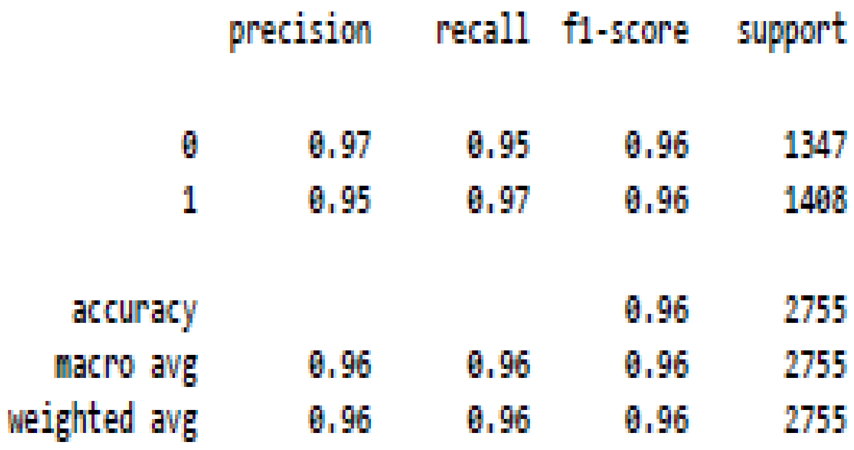


Figure 4.5

Accuracy ,Loss graph

- Classification report



```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      1347
           1       0.95      0.97      0.96      1408

    accuracy                           0.96      2755
   macro avg       0.96      0.96      0.96      2755
weighted avg       0.96      0.96      0.96      2755
```

Figure 4.6

- Computing Area Under Curve (AUC)

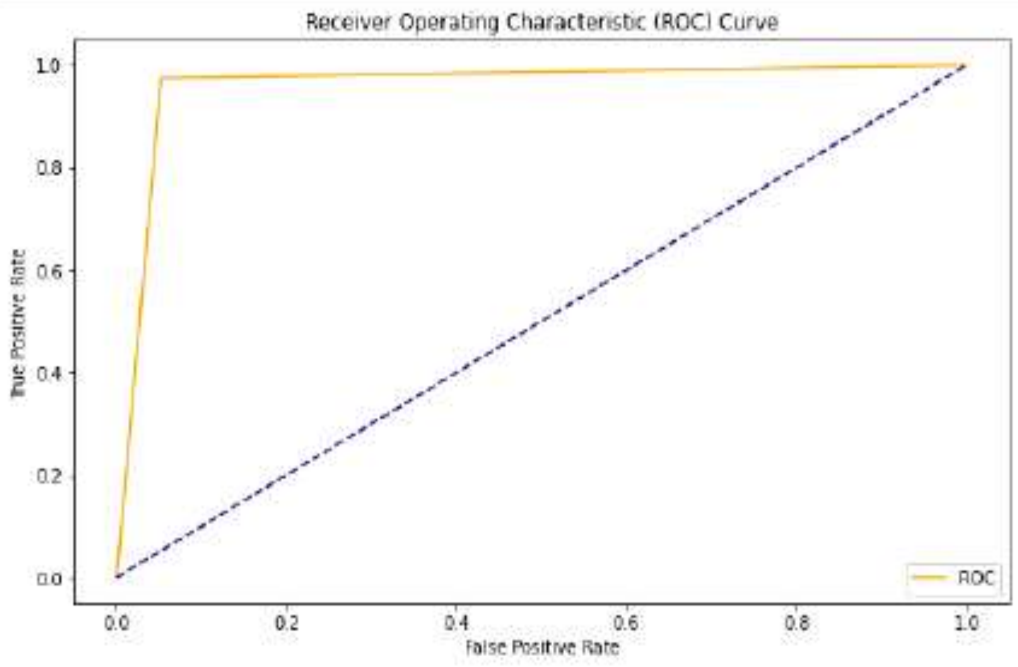


Figure 4.7

ROC curve

- Ground truth vs Predicted

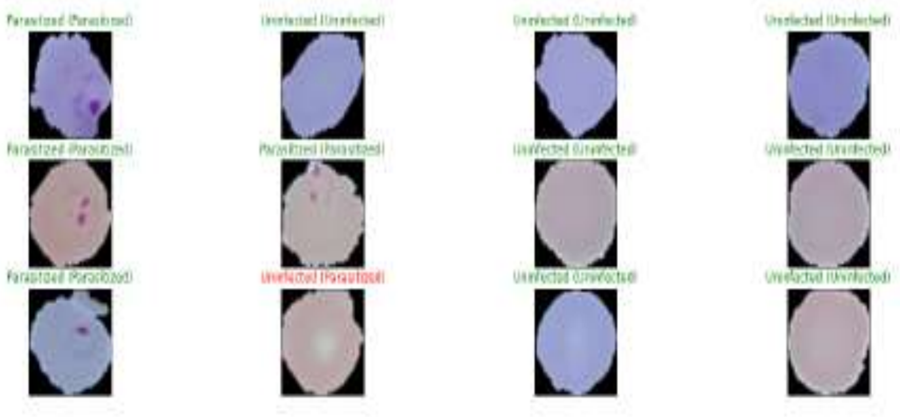


Figure 4.8

4.INFERENCE

- The model had train accuracy (Fig 4.2) over 95.73% and loss over 13.46%, test accuracy of 96.19% and loss of 12.46%. This shows that the model has performed well for the test data higher than the training period. Due to combination of the convolution layers, the max pool layer and the drop out layer with the optimizer adam backpropagating the learning process and the callback method halts for optimal loss rate the execution of the epoch.
- The confusion matrix(Fig 4.4) shows affected cell(1371),unaffected cell(1276),the fault in prediction counts upto 108.
- The classification report(Fig 4.6) show the precision of model predicting unaffected cell(97%) and affected cell(95%).
- The challenges faced is computation.

5. REFERENCES

1. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images- Sivaramakrishnan Rajaraman1, Sameer K. Antani1, Mahdieh Poostchi1, Kamolrat Silamut2, Md. A. Hossain3, Richard J. Maude2,4,5, Stefan Jaeger1, George R. Thoma1- April 16, 2018-PubMed -29682411

2. Improving Malaria Parasite Detection from Red Blood
Cell using Deep Convolutional Neural Networks- Aimon Rahman1 , Hasib Zunair1 , M Sohel Rahman2* , Jesia Quader Yuki1 , Sabyasachi Biswas1 , Md Ashraful Alam3 , Nabila Binte Alam4 , M.R.C. Mahdy1-March 18,2019- Dhaka Central International Medical College.

3. Classification of Malaria-Infected Cells Using Deep Convolutional Neural Networks- W. David Pan, Yuhang Dong and Dongsheng Wu- June 13th 2017- intechopen.72426

4. Deep learning approach to detect malaria from microscopic images- Vijayalakshmi A1&Rajesh Kanna B- 11 January 2019- Springer Science+Business Media, LLC, part of Springer Nature 2019

5. Deep CNN frameworks comparison for malaria diagnosis- Priyadarshini Adyasha Pattanaik, Zelong Wang and Patrick Horain - September 2019- researchgate

6. Image analysis and machine learning for detecting malaria- MAHDIEH POOSTCHI, KAMOLRAT SILAMUT, RICHARD J. MAUDE, STEFAN JAEGER, and GEORGE THOMA- December 19, 2017- Elsevier Inc.

7. Deep Learning for Smartphone-based Malaria Parasite Detection in Thick Blood Smears- Feng Yang*, Mahdieh Poostchi, Hang Yu, Zhou Zhou, Kamolrat Silamut, Jian Yu, Richard J Maude, Stefan Jaeger*, Sameer Antani- Dec. 19, 2018-IEEE journal.

8. Malaria Parasite Detection in Thick Blood Smear Microscopic Images Using Modified YOLOV3 and YOLOV4 Models- Fetulhak Abdurahman1*, Kinde Anlay2and Mohammed Aliy-Research square- September 2020.