MALARIA DETECTION USING DEEP LEARNING

Presented by-

Snigda Gedela,
17BCD7071
Computer Science with Specialisation in Data Analytics,
VIT-AP

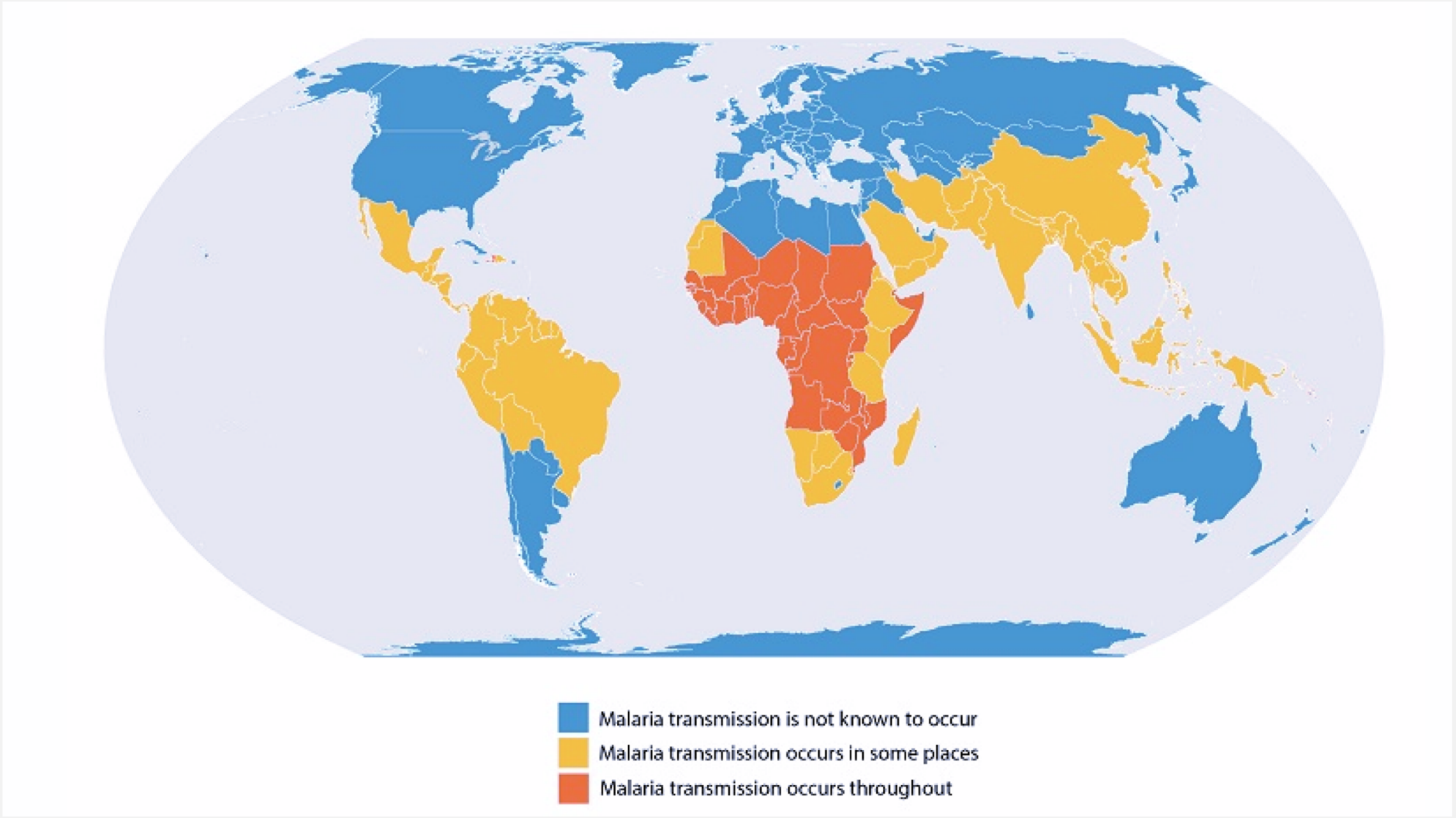
CONTENTS

PROBLEM STATEMENT

- The manual diagnosis of blood smears for detecting malaria is an intensive manual process that requires lot of time and trained people. Trained people manually count how many red blood cells contain parasites out of 5,000 cells at X100 magnifications.
- Techniques like polymerase chain reaction and rapid diagnostic tests are used but then polymerase chain reaction(PCR) is limited in its performance and rapid diagnostic tests(RDTs) are less cost-effective in regions like Hawkes and Katsuva. Moreover, The techniques that are presently being used are limited in their performance.
- India accounted for 6% of global malaria cases and 7% deaths caused by it in 2018, according to a report released by the World Health Organization(WHO) .
- WHO figures also suggest that India is unlikely to reduce its case burden beyond 40% by this year(2020). A key obstruction to eliminating malaria is a weak surveillance system. India, being one of the major contributors to the global burden of malaria, was able to detect only 16% of cases, via the system.

MOTIVATION

- Malaria is a deadly, infectious, mosquito-borne disease caused by Plasmodium parasites that are transmitted by the bites of infected female Anopheles mosquitoes.
- 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.

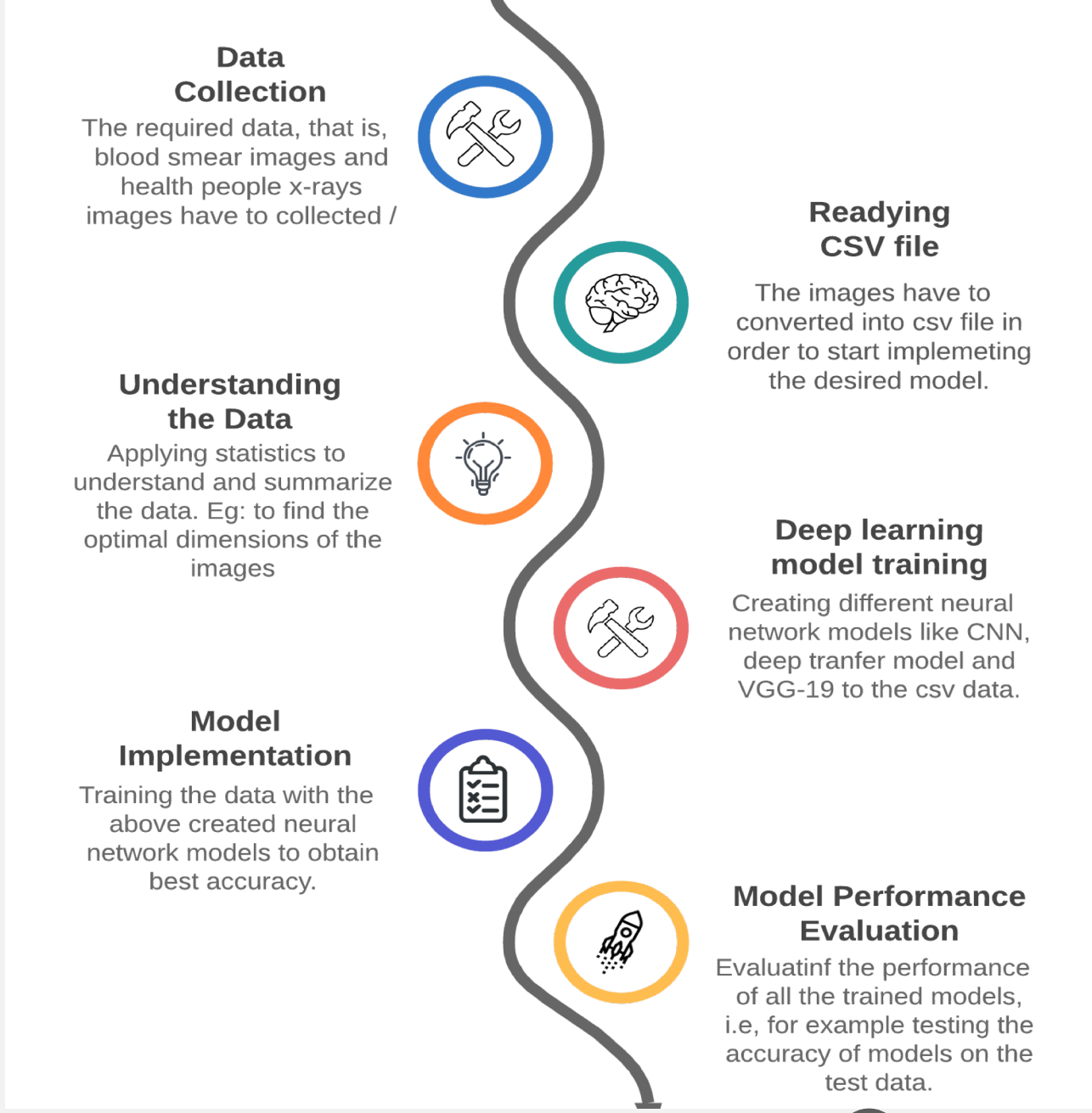


This map shows an approximation of the parts of the world where malaria transmission occurs.

PROJECT OBJECTIVE

- The main objective of this project is to develop an automate malaria detection system using Deep Learning algorithms and thus able to support the health sector by reducing the time taken, cost, the effort put in to detect malaria at present and hence, an easily adoptable solution.

PROJECT WORKFLOW

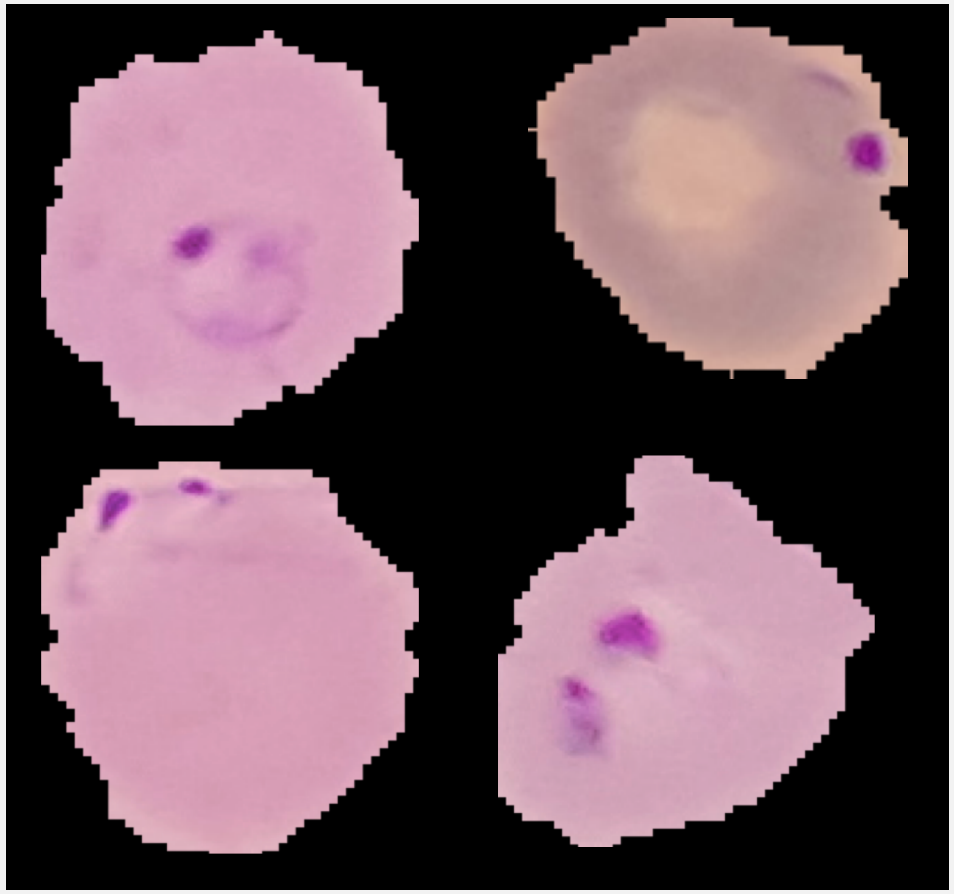


PROJECT PLAN

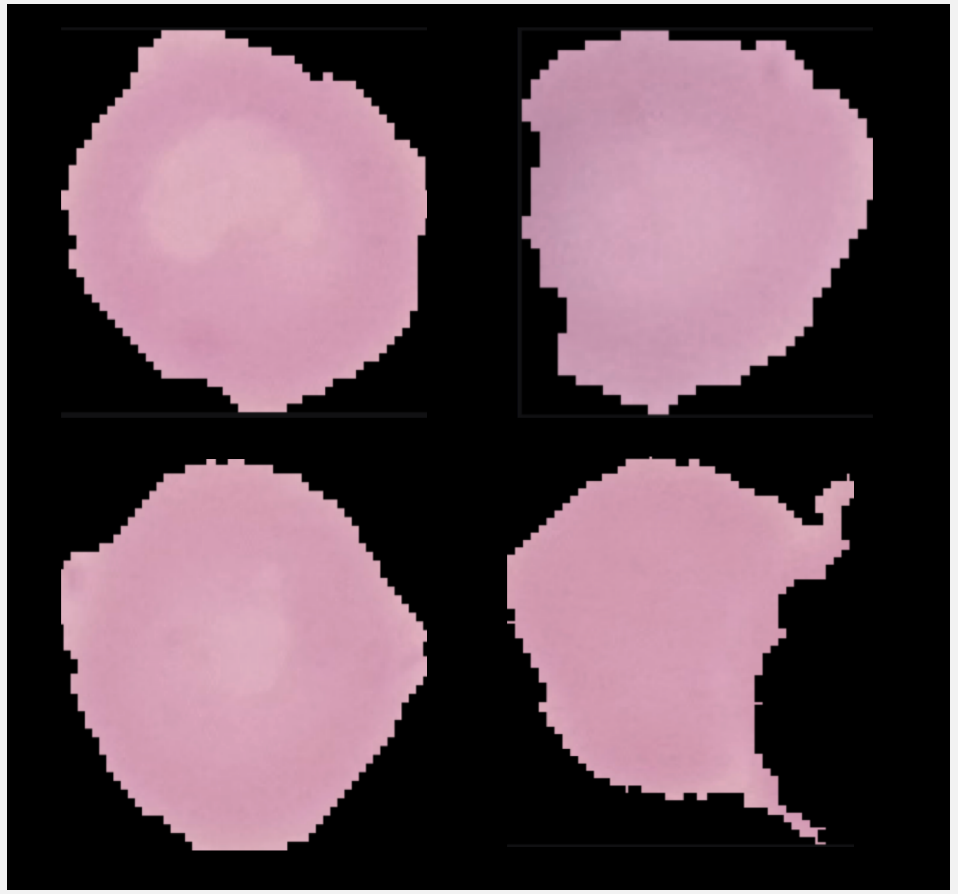
As the malaria detection dataset involves blood smear images, Deep learning algorithms like CNN can be used for effective detection of malaria. It will be implemented on Google Colab Notebook using Python Language.

- **Level 1: Data Collection :** In this level, the required image files for the project will be collected. After the satisfying amount of image collection for the project, they have to be converted to csv files.
- **Level 2: Understanding the data :** The dimensions of the blood smears vary from person to person. Hence, in order to understand the image pattern and to find the optimal dimensions of the images, statistics should be applied.
- **Level 3: Model training and Evaluation :** After fine tuning the dataset, the data has to be divided into train and test data. Then, the optimal Convolutional Neural Networks model layers will be experimentally determined for feature extraction from the underlying data. In order to get the best results, VGG-19 model and Data transfer models are also used to train the data. Finally, improvisations will be made and the model swill be completed with best accuracy.

SOME SAMPLE IMAGES



Parasitized/Infected Cell Images



Uninfected Cell Images

ENGINEERING STANDARDS

The main engineering standards that will be used to build this model are:

- **Convolutional Neural Networks**: It is one of the main neural network category used in image recognition, image classification, objects detection etc. Technically, to train and test, each input image will pass it through a series of convolution layers with filters/kernels, pooling, fully connected layers.
- **Activation Functions**: This is next step after the implementation of the network models, that is, activation functions like SoftMax function is applied on the series of convolution layers to classify an object with probabilistic values between 0 and 1.
- **InceptionV3 model**: Inception V3 by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. It has been shown to attain greater than 78.1% accuracy on the ImageNet dataset.
- **ImageNet Dataset:** ImageNet has over ten million URLs of labelled images. For InceptionV3 model, the ImageNet dataset is composed of 1,331,167 images which are split into training and evaluation datasets containing 1,281,167 and 50,000 images, respectively.
- **Deep Transfer Learning**: It is a model where instead of training the model from the scratch, a network which is trained on a different domain for different source task is taken and it is then adapted for our domain and our target task. In this project, the two most popular strategies, that is, pre-trained model as a feature extractor and pre-trained model with fine-tuning will be used.

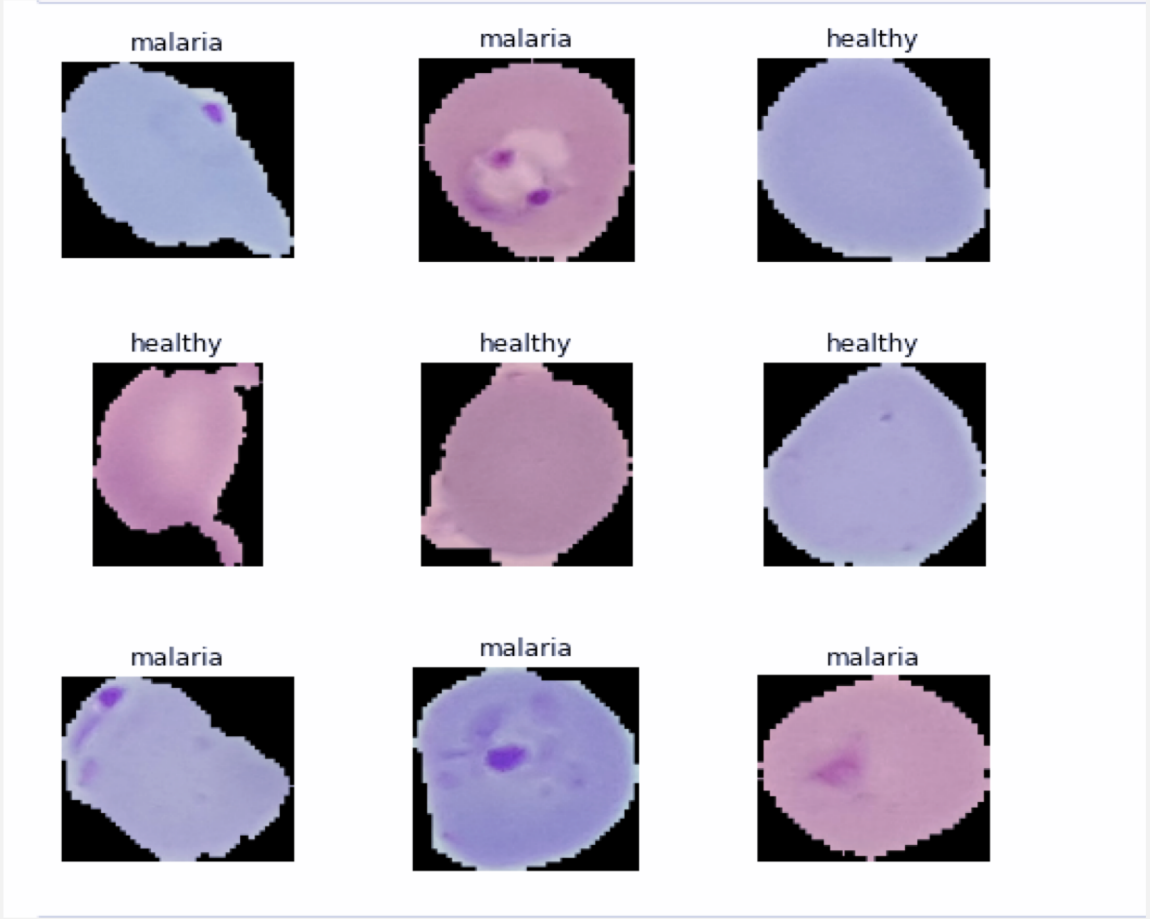
DATASET USED

- Number of images: Infected – 13779 and Healthy – 13779
- This dataset is taken from Kaggle which was initially taken from official NIH website
- https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria

Preprocessing:

INITIAL DATA SET

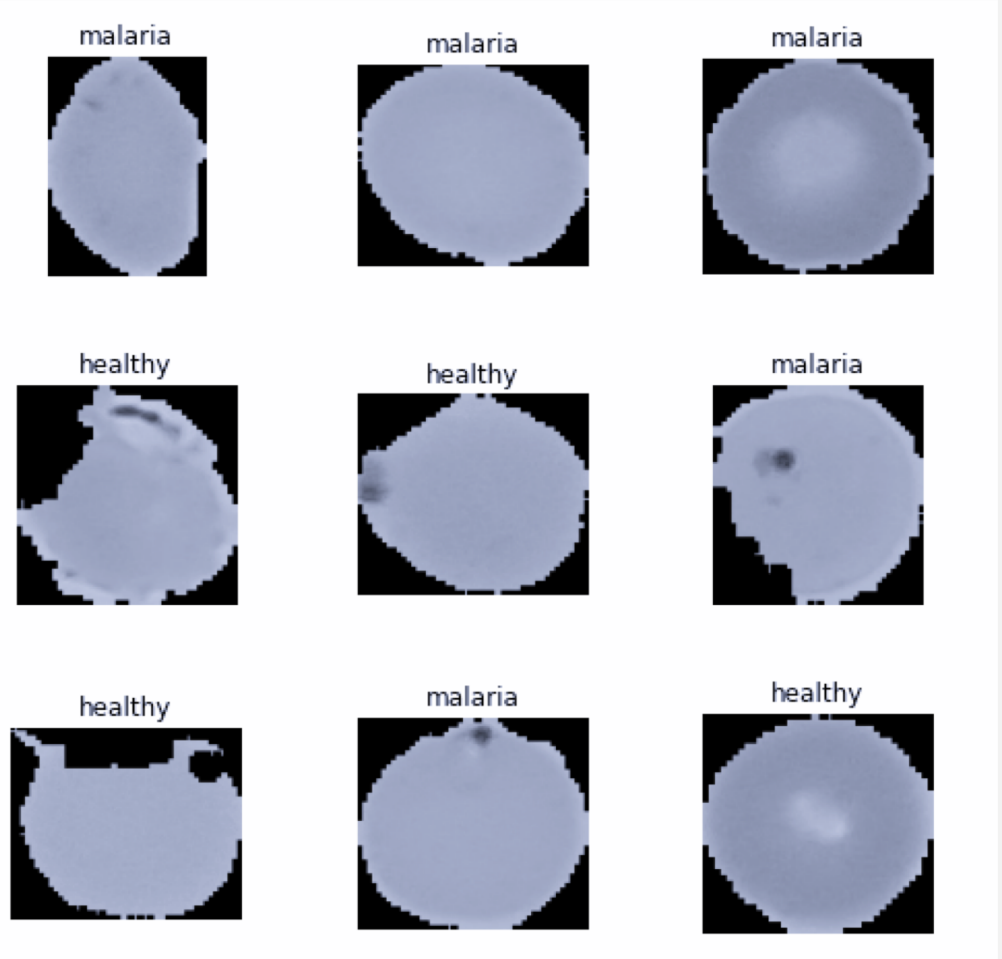
```
from matplotlib.pyplot import figure, imshow, axis,title,xticks,yticks,subplots_adjust,subplot
from matplotlib.image import imread

list_of_files=list(files_df['filename'][0:9])
list_of_titles=list(files_df['label'][0:9])
def showImagesHorizontally(list_of_files):
    fig = figure(figsize=(8,8))
    number_of_files = len(list_of_files)
    for i in range(number_of_files):
        a=fig.add_subplot(3,3,i+1)
        image = imread(list_of_files[i])
        #title(list_of_titles[i])
        subplots_adjust(hspace = 0.5 , wspace = 0.5)
        imshow(image,cmap='Greys_r')
        axis('off')
        title('{}'.format(list_of_titles[i]))
        xticks([]) , yticks([])
showImagesHorizontally(list_of_files)
```



Preprocessing:

GREYSCALE IMAGES

```
list_of_files=list(train_files[10:19])
list_of_titles=list(train_labels[0:9])
def showImagesHorizontally(list_of_files):
    fig = figure(figsize=(8,8))
    number_of_files = len(list_of_files)
    for i in range(number_of_files):
        a=fig.add_subplot(3,3,i+1)
        image = imread(list_of_files[i])
        #title(list_of_titles[i])
        subplots_adjust(hspace = 0.5 , wspace = 0.5)
        imshow(image,cmap='Greys_r')
        axis('off')
        title('{}'.format(list_of_titles[i]))
        xticks([]) , yticks([])
showImagesHorizontally(list_of_files)
```

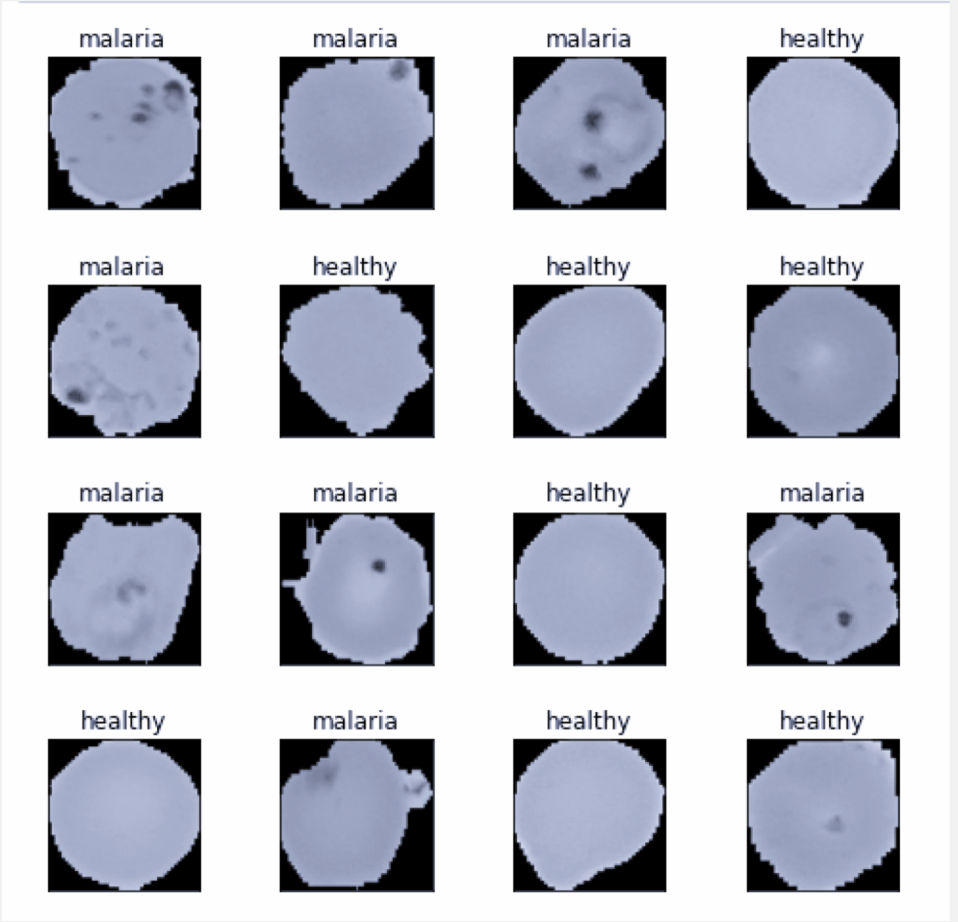


Preprocessing:

RESIZED IMAGES

```
import cv2
from concurrent import futures
import threading

def get_img_shape_parallel(idx, img, total_imgs):
    if idx % 5000 == 0 or idx == (total_imgs - 1):
        print('working on img num: {}'.format(idx))
    return cv2.imread(img).shape

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))
```

```
Min Dimensions: [46 46  3]
Avg Dimensions: [132.9369276  132.32279247   3.        ]
Median Dimensions: [130. 130.   3.]
Max Dimensions: [385 364   3]
```



ADVANTAGES OF PREPROCESSING

- Grey scaling the Data:

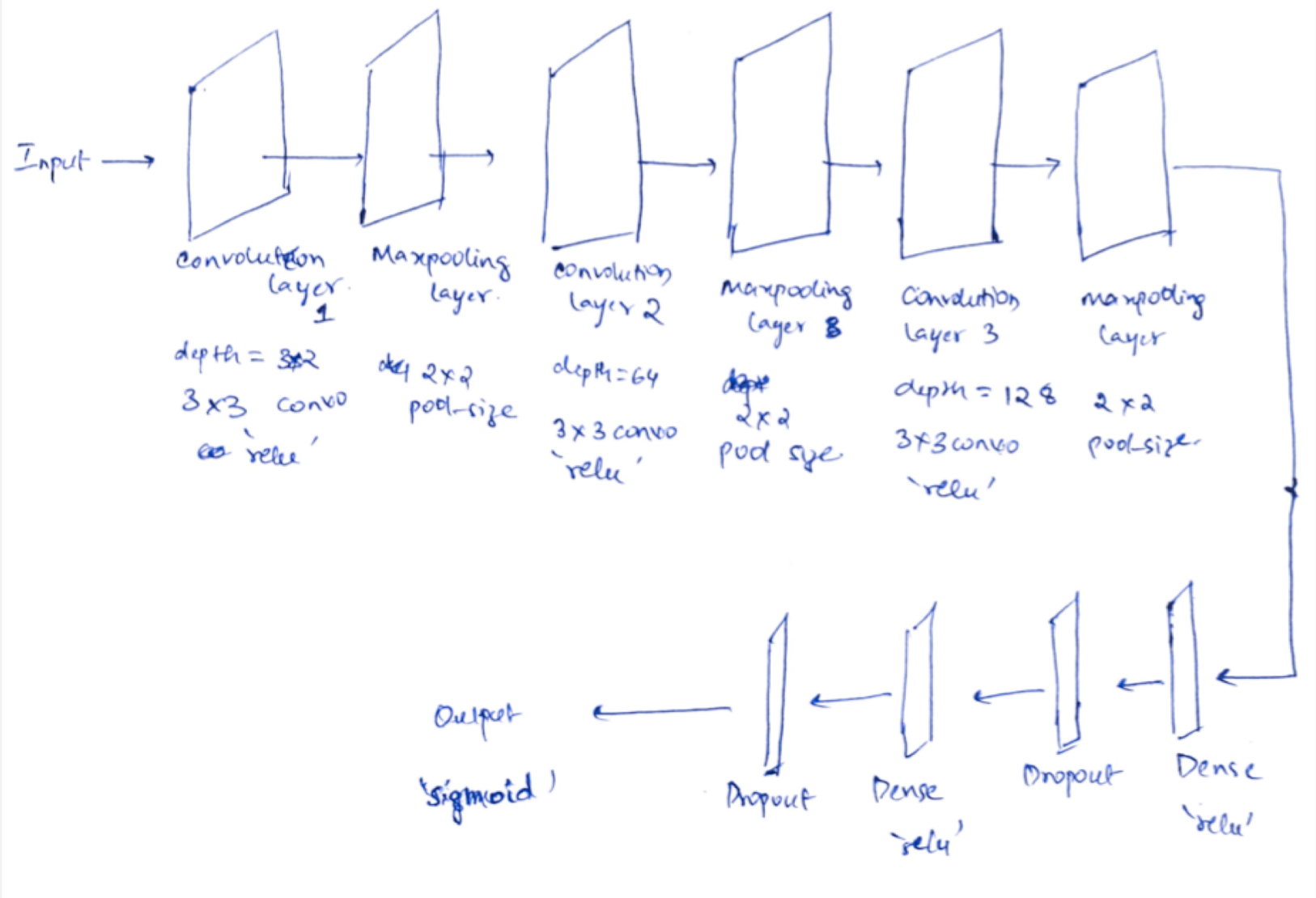
1. Reduce the complexity of the model
2. Color images are harder to process
3. Considered as noise when color wouldn't effect the results

- Rescaling the Data:

1. The images are not treated differently by the CNN model due to the input variation
2. Variability in the output due to unit change in the input would be avoided

Model 1:

CNN ARCHITECTURE



Model 1:

CNN MODEL

```
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
_________________________________________________________________
```

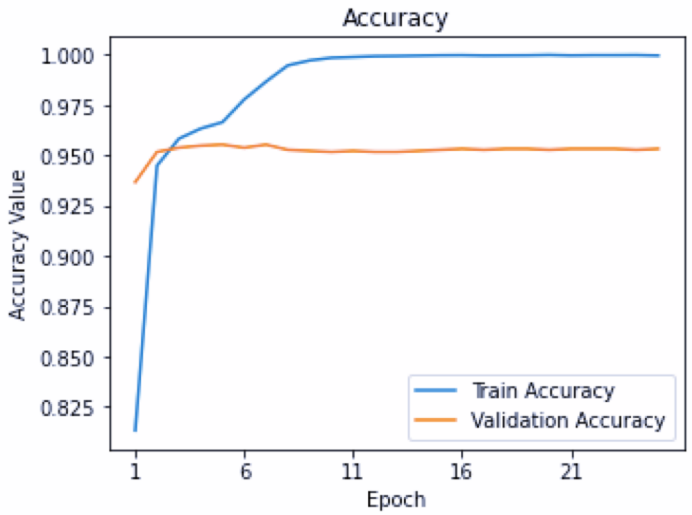
Model 1:

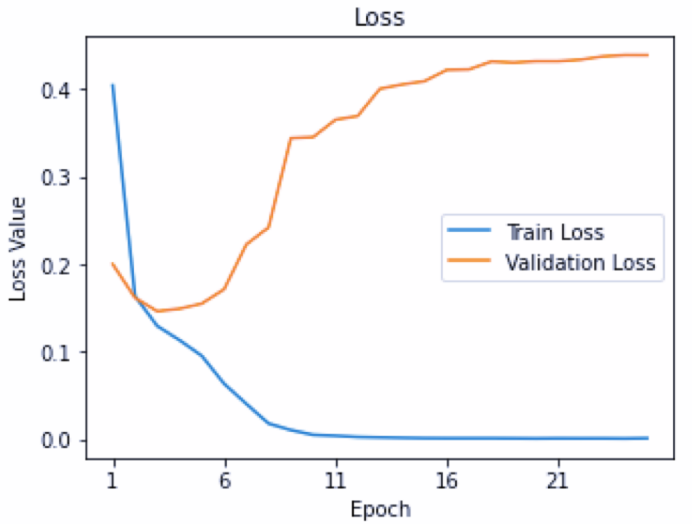
TRAINING THE MODEL

```
Epoch 16/25
272/272 [==============================] - 347s 1s/step - loss: 0.0014 - accuracy: 0.9999 - val_loss: 0.4220 - val_ac
curacy: 0.9533 - lr: 1.5625e-05
Epoch 17/25
272/272 [==============================] - 362s 1s/step - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.4225 - val_ac
curacy: 0.9528 - lr: 1.5625e-05
Epoch 18/25
272/272 [==============================] - 364s 1s/step - loss: 0.0014 - accuracy: 0.9998 - val_loss: 0.4315 - val_ac
curacy: 0.9533 - lr: 7.8125e-06
Epoch 19/25
272/272 [==============================] - 382s 1s/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.4304 - val_ac
curacy: 0.9533 - lr: 7.8125e-06
Epoch 20/25
272/272 [==============================] - 418s 2s/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.4318 - val_ac
curacy: 0.9528 - lr: 3.9063e-06
Epoch 21/25
272/272 [==============================] - 425s 2s/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 0.4318 - val_ac
curacy: 0.9533 - lr: 3.9063e-06
Epoch 22/25
272/272 [==============================] - 408s 2s/step - loss: 0.0012 - accuracy: 0.9999 - val_loss: 0.4336 - val_ac
curacy: 0.9533 - lr: 1.9531e-06
Epoch 23/25
272/272 [==============================] - 412s 2s/step - loss: 0.0012 - accuracy: 0.9999 - val_loss: 0.4374 - val_ac
curacy: 0.9533 - lr: 1.9531e-06
Epoch 24/25
272/272 [==============================] - 403s 1s/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.4389 - val_ac
curacy: 0.9528 - lr: 1.0000e-06
Epoch 25/25
272/272 [==============================] - 418s 2s/step - loss: 0.0014 - accuracy: 0.9997 - val_loss: 0.4388 - val_ac
curacy: 0.9533 - lr: 1.0000e-06
```

Model 1:

VISUALIZATION






```
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")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
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 Training:

THE FINAL MODEL

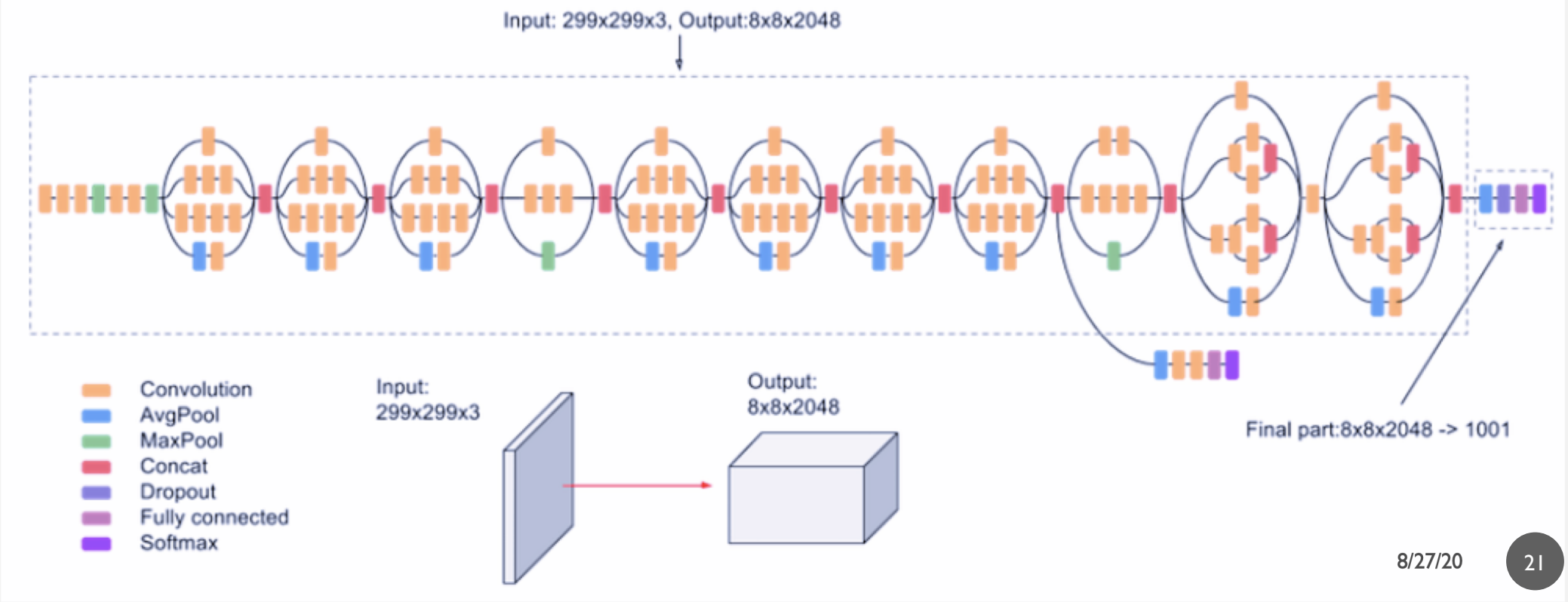
```
import tensorflow as tf

from keras.models import Model
from keras.optimizers import Adam
from keras.layers import GlobalAveragePooling2D
from keras.layers import Dense
from keras.applications.inception_v3 import InceptionV3
from keras.utils.np_utils import to_categorical

# Get the InceptionV3 model so we can do transfer learning
base_inception = InceptionV3(weights='imagenet', include_top=False,
                             input_shape=INPUT_SHAPE)

out = base_inception.output
out = GlobalAveragePooling2D()(out)
out = Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.2))(out)
out = Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.2))(out)
out=  Dropout(0.5)
predictions = Dense(1, activation='sigmoid')(out)


# only if we want to freeze layers
for layer in base_inception.layers:
    layer.trainable = False

model = Model(inputs=base_inception.input, outputs=predictions)

model.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics=['accuracy'])
model.summary()
```

- Used InceptionV3 Model as the Base Inception of my CNN model
- Then added a Global Average Pooling Layer to reduce the size of the input neurons
- Added two more Dense layers with 512 neurons and leaky_relu activation function.
- Added a dropout layer to avoid overfitting(Dropout Regularization)
- Finally, a prediction dense layer with sigmoid activation function.
- Used Adam Optimizer and binary_crossentropy loss function.

Inception V3 Model:

THE INCEPTIONV3 MODEL

InceptionV3 model: Inception V3 by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. It has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. It is the culmination of many ideas developed by multiple researchers over the years.



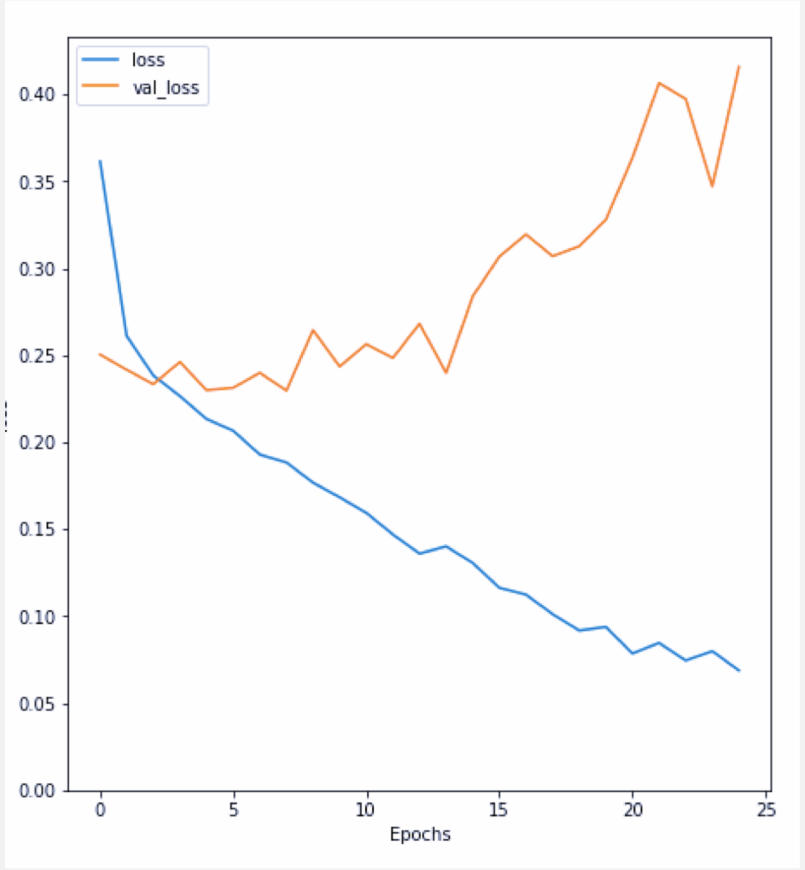
Testing:

25 EPOCHS

```
batch_size=BATCH_SIZE
train_steps_per_epoch = train_files.shape[0] // batch_size
val_steps_per_epoch = val_files.shape[0] // batch_size

#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),verbose=1)

history = model.fit(train_imgs_scaled,train_labels_enc,
                            steps_per_epoch=train_steps_per_epoch,
                            validation_data=(val_imgs_scaled, val_labels_enc),
                            validation_steps=val_steps_per_epoch,batch_size=BATCH_SIZE,
                            epochs=25, verbose=1)
```

```
curacy: 0.8964
Epoch 20/25
271/271 [==============================] - 295s 1s/step - loss: 0.0938 - accuracy: 0.9640 - val_loss: 0.3280 - val_ac
curacy: 0.9089
Epoch 21/25
271/271 [==============================] - 289s 1s/step - loss: 0.0785 - accuracy: 0.9712 - val_loss: 0.3635 - val_ac
curacy: 0.9000
Epoch 22/25
271/271 [==============================] - 280s 1s/step - loss: 0.0846 - accuracy: 0.9669 - val_loss: 0.4065 - val_ac
curacy: 0.8969
Epoch 23/25
271/271 [==============================] - 284s 1s/step - loss: 0.0745 - accuracy: 0.9720 - val_loss: 0.3974 - val_ac
curacy: 0.8995
Epoch 24/25
271/271 [==============================] - 280s 1s/step - loss: 0.0799 - accuracy: 0.9689 - val_loss: 0.3472 - val_ac
curacy: 0.9042
Epoch 25/25
271/271 [==============================] - 285s 1s/step - loss: 0.0688 - accuracy: 0.9750 - val_loss: 0.4158 - val_ac
curacy: 0.9062
```

Screenshot of the testing with 25 epochs



Loss Graph against the epochs

Testing:

EARLY STOPPING

15 EPOCHS(RELU)

```
batch_size=BATCH_SIZE
train_steps_per_epoch = train_files.shape[0] // batch_size
val_steps_per_epoch = val_files.shape[0] // batch_size

#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),verbose=1)

history = model.fit(train_imgs_scaled,train_labels_enc,
                                steps_per_epoch=train_steps_per_epoch,
                                validation_data=(val_imgs_scaled, val_labels_enc),
                                validation_steps=val_steps_per_epoch,batch_size=BATCH_SIZE,
                                epochs=15, verbose=1)
```

```
curacy: 0.9036
Epoch 10/15
271/271 [==============================] - 299s 1s/step - loss: 0.1520 - accuracy: 0.9381 - val_loss: 0.2942 - val_ac
curacy: 0.8984
Epoch 11/15
271/271 [==============================] - 299s 1s/step - loss: 0.1535 - accuracy: 0.9386 - val_loss: 0.2900 - val_ac
curacy: 0.8969
Epoch 12/15
271/271 [==============================] - 835s 3s/step - loss: 0.1395 - accuracy: 0.9440 - val_loss: 0.2951 - val_ac
curacy: 0.8948
Epoch 13/15
271/271 [==============================] - 325s 1s/step - loss: 0.1336 - accuracy: 0.9478 - val_loss: 0.2905 - val_ac
curacy: 0.9042
Epoch 14/15
271/271 [==============================] - 317s 1s/step - loss: 0.1261 - accuracy: 0.9513 - val_loss: 0.3274 - val_ac
curacy: 0.8818
Epoch 15/15
271/271 [==============================] - 326s 1s/step - loss: 0.1226 - accuracy: 0.9511 - val_loss: 0.2985 - val_ac
curacy: 0.9109
```

RELU ACTIVATION

- Partial Saturation
- Not Zero Centered
- Computationally least expensive

Testing:

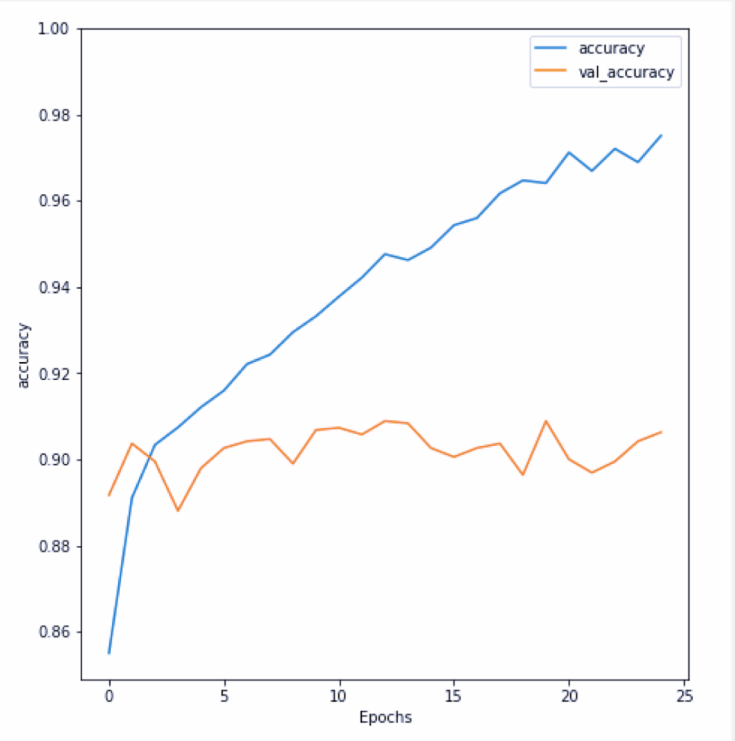
15 EPOCHS(LEAKY_RELU)

```
batch_size=BATCH_SIZE
train_steps_per_epoch = train_files.shape[0] // batch_size
val_steps_per_epoch = val_files.shape[0] // batch_size

#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),verbose=1)

history = model.fit(train_imgs_scaled,train_labels_enc,
                            steps_per_epoch=train_steps_per_epoch,
                            validation_data=(val_imgs_scaled, val_labels_enc),
                            validation_steps=val_steps_per_epoch,batch_size=BATCH_SIZE,
                            epochs=15, verbose=1)
```

```
curacy: 0.9297
Epoch 10/15
271/271 [==============================] - 275s 1s/step - loss: 0.1162 - accuracy: 0.9546 - val_loss: 0.2129 - val_ac
curacy: 0.9344
Epoch 11/15
271/271 [==============================] - 281s 1s/step - loss: 0.1056 - accuracy: 0.9590 - val_loss: 0.2120 - val_ac
curacy: 0.9318
Epoch 12/15
271/271 [==============================] - 298s 1s/step - loss: 0.0979 - accuracy: 0.9618 - val_loss: 0.2309 - val_ac
curacy: 0.9281
Epoch 13/15
271/271 [==============================] - 317s 1s/step - loss: 0.0934 - accuracy: 0.9650 - val_loss: 0.2298 - val_ac
curacy: 0.9292
Epoch 14/15
271/271 [==============================] - 283s 1s/step - loss: 0.0890 - accuracy: 0.9644 - val_loss: 0.2310 - val_ac
curacy: 0.9281
Epoch 15/15
271/271 [==============================] - 280s 1s/step - loss: 0.0719 - accuracy: 0.9742 - val_loss: 0.2816 - val_ac
curacy: 0.9271
```

Screenshot of the testing with 15 epochs

- No Saturation
- Not Zero Centered
- Computationally least expensive



COMPARISON

	The other Paper	My model
Author	Dipanjan Sarkar	
Dataset	Kaggle Dataset	Kaggle Dataset
Size of the Dataset	Infected, Healthy images: (13779, 13779)	Infected, Healthy images: (13779, 13779)
Preprocessing Techniques Used:	Resizing the data	Grayscale the data Resizing the data
The model used:	VGG-19 model	The Basic CNN architecture Transfer Learning model using InceptionV3 model
Result(Accuracy):	96.3%	97.1%

Reference	Task	Summary	Results	Dataset
Dong et al.[3]	Classification	evaluated the performances of three popular Convolutional Neural Network, namely LeNet-5, AlexNet and GoogLeNet	Accuracy : about 95% for all the three CNNs.	Kaggle Dataset

STANDARD CONCEPTS AND TOOLS USED

- Used Python Programming Language
- Jupyter Notebook for the code implementation
- Deep Learning Concepts:

1. Convolutional Neural Networks
2. Activation Functions
3. Greyscale Images
4. InceptionV3 model
5. Early Stopping Method
6. Dropout Regularization

REFERENCES

1. Fact Sheet about Malaria: https://www.who.int/news-room/fact-sheets/detail/malaria
2. How malaria RDTs work: https://www.who.int/malaria/areas/diagnosis/rapid-diagnostic-tests/about-rdt/en/
3. Dong Y., Jiang Z., Shen H., Pan W.D., Williams L.A., Reddy V.V., Benjamin W.H., Bryan A.W. Proceedings of the 2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI); Orlando, FL, USA. 16–19 February 2017: https://ieeexplore.ieee.org/abstract/document/7897215
4. Department of Electrical & Computer Engineering, North South University, Dhaka 1229, Bangladesh; (K.M.F.F.): https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7277980/
5. Understanding of Convolutional Neural Networks(CNN) – Deep Learning: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

THANK YOU