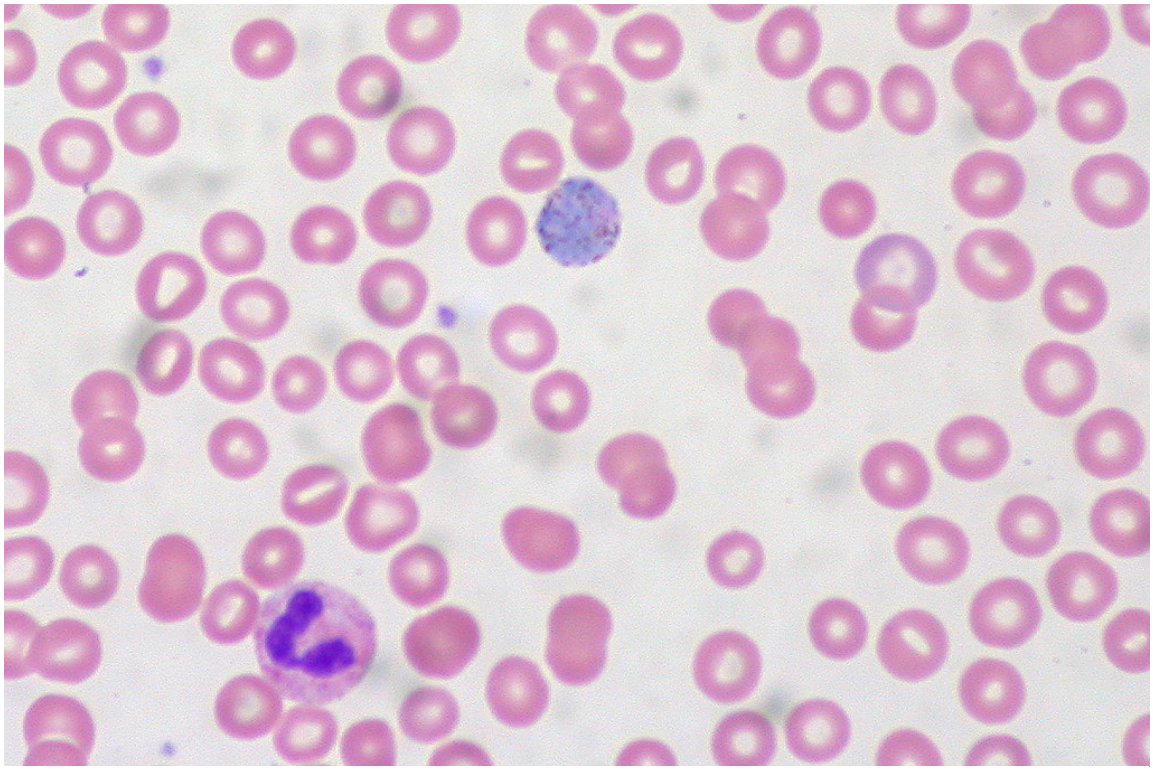
DETECTING MALARIA WITH DEEP LEARNING

Machine Learning combined with open-source tools can improve diagnosis of the fatal disease malaria.

By Nabila Salisu



OVERVIEW

Malaria is one of the most severe public health problems worldwide. It is a leading cause of death and disease in many developing countries, where young children and pregnant women are the groups most affected. What's the point of technology when it can't help the needy and save lives?
Deep learning helps us to build robust, scalable and effective solutions which can be adopted by everyone even in remote corners of the world.
It would therefore be highly beneficial to propose a method that performs malaria detection using Deep Learning Algorithms.

PROBLEM DEFINITION FOR DEEP LEARNING-BASED ON MALARIA DETECTION
The main aim of this malaria detection system is to address the challenges in the existing system by automating the process of malaria detection using machine learning and image processing, and, to build a feed-forward model that can recognize the infected and uninfected malaria cells in the images, using classification, convolutional neural networks, Kera's and TensorFlow.

Key Questions
In this research, we attempted to answer three research questions:

1. *How are we going to automate malaria detection diagnostic process?*
2. *. Compare models to find which performs better on the malaria dataset?*

3. *. Is there any difference in the Convolutional Neural Network performance measured on the malaria dataset before and after applying preprocessing techniques like data augmentation and standardization?*

Dataset to be used for Malaria Detection

The dataset used for this system is a set of 24,958 train and 2,600 test images (colored) that we have taken from microscopic images. These images are of the following categories:

***Parasitized**: The parasitized cells contain the Plasmodium parasite which causes malaria.*

***Uninfected**: The uninfected cells are free of the Plasmodium parasites.*

Objective

Our objective is to build an efficient computer vision model to detect malaria. The model should identify whether the image of a red blood cell is that of one infected with malaria or not, and classify the same as parasitized or uninfected, respectively.

Algorithm for Malaria Detection

Data Description:
The data for our analysis has been carefully collected and annotated of healthy and infected blood smear images. We started by importing necessary deep learning frameworks like TensorFlow, Matplotlib and pandas.

```
import zipfile
import os
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, BatchNormalization, Dropout, Flatten, LeakyReLU, GlobalAvgPool2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers

#to ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Remove the limit from the number of displayed columns and rows. It helps to see the entire dataframe while printing it
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)
```

Build and explore image datasets

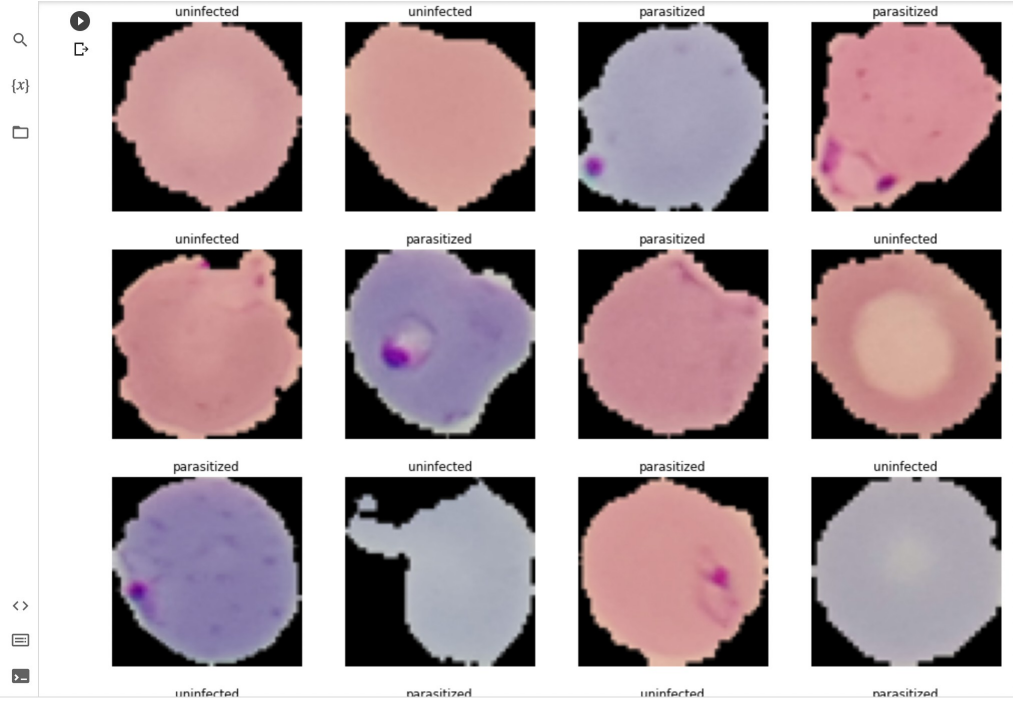
To build deep learning models, we started by

- *splitting the data into the train and the test dataset.*
- *creating the labels for both types of images so, we will be able to train and test the model and do the same for the training data first. we then use the same code for the test data as well*

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. We checked the shape of the train and test data, based on statistics, and tried normalizing each image from 0 to 255 pixels to fix the dimensions.
After plotting the data, it looks like we have a balanced dataset with 12,000 malaria(parasitized) and 12,000 non-malaria (uninfected) cell images.

DATA EXPLORATION
One of the fundamental ways to extract insights from a data set is to visualize the data so that you can look at just a piece of it at a time. Without spending significant time on understanding the data and its patterns one cannot expect to build efficient predictive models.

By visualizing the images from the train data, we can differentiate the parasite cells (pink color) compared to the uninfected ones.



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 We observe a distinct classification between the parasitized cells and the uninfected cells. Based.

Plotting the mean images for parasitized and uninfected, we can see that there isn't a big difference between both images. The images are well-aligned, the same size and shape, possibly with different exposure.

A simple segmentation of the image by converting RGB to HSV helped us detect the presence of parasites at every stage of its lifecycle, each species changes in its size, color, shape, and morphology.

To reduce the amount of noise and remove speckles within the image, we Applied a low pass filter, which removes detail occurring at high spatial frequencies, it is perceived as a blurring effect. Gaussian blur is a filter that makes use of a Gaussian kernel. By blurring the image, we can identify parasite within affected cells.

Proposed Approach

The deep learning architecture with the above configuration successfully achieved the classification of malaria-infected blood cells.

However, this system can only be run by a Machine Learning Engineer who implements the calculations to perform classification into a program. Of course, the system created is not only for use by Machine Learning Engineers. Remember, the goal is to help health workers make it easier to diagnose. This means that this system must be run or operated by the user (in this case, the user in question is a health worker) to detect or classify which cell images are infected with malaria and which are not infected.

For this reason, we need to create a Deep learning model, or more specifically a convolutional neural network (CNNs). This model has proven very effective in a wide variety of computer vision tasks, t will be beneficial and easier for health workers or users to operate.

The key layers in a CNN model include convolution and pooling layers. Convolution layers learn spatial hierarchical patterns from data, which are also translation-invariant, so they can 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 will help us with down sampling 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.

Measures of success 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.

To evaluate the success of our different models we will cover different types of evaluation metrics:

- *Classification Accuracy*
- *Logarithmic Loss*
- *Confusion Matrix*
- *Area under Curve*
- *F1 Score*

Convolutional Neural Network (CNN)

Deep learning is a very significant subset of machine learning because of its high performance

across various domains. Convolutional Neural Network (CNN) is a powerful image processing

deep learning type often using in computer vision that comprises an image and video recognition along with a recommender system and natural language processing (NLP). CNN uses a multilayer system consists of the input layer, output layer, and a hidden layer that comprises multiple convolutional layers, pooling layers, fully connected layers.

Deep learning 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.

Build a baseline model for Malaria Detection

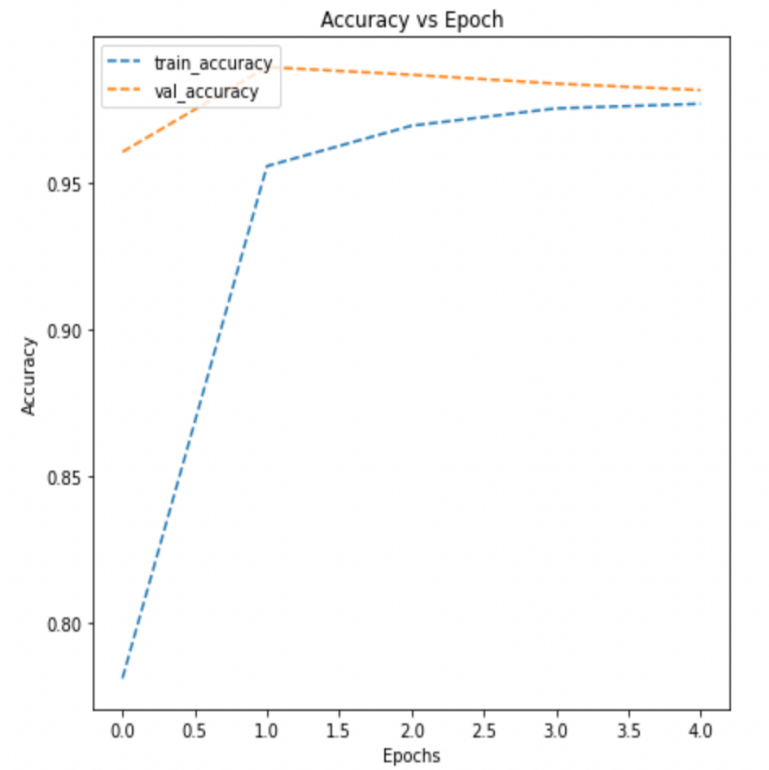
Our first malaria detection model will build and train a basic CNN from scratch. First, let's define our model architecture.

```
#creating sequential model

model=Sequential()
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu",input_shape=(64,64,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(Conv2D(filters=32,kernel_size=2,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="softmax"))#2 represent output layer neurons
model.summary()
```

Based on the architecture in this code, our CNN model has three convolution and pooling layers, followed by two dense layers, and dropouts for regularization. Let's train our model.

Evaluation of the baseline model

We got a validation accuracy of 98%, which is pretty good, although our model looks to be overfitting. We can get a clear perspective on this by plotting the confusion matrix, training, validation accuracy and loss curves.



we can clearly observe that the training and validation accuracy are increasing, also we notice that validation accuracy is slightly higher than the train accuracy. So now let's try to build another model with few more add on layers and try to check if we can try to improve the model. Therefore, let's try to build a model by adding few layers if required and altering the activation functions.

Model 1: improving the performance of our model by adding new layers

In order to add two more convolutional layers, we need to apply the pooling operation after initializing CNN. With slight modification in the number of filters.

In our improved model, we used the same base model architecture with the exception that we added two layers.

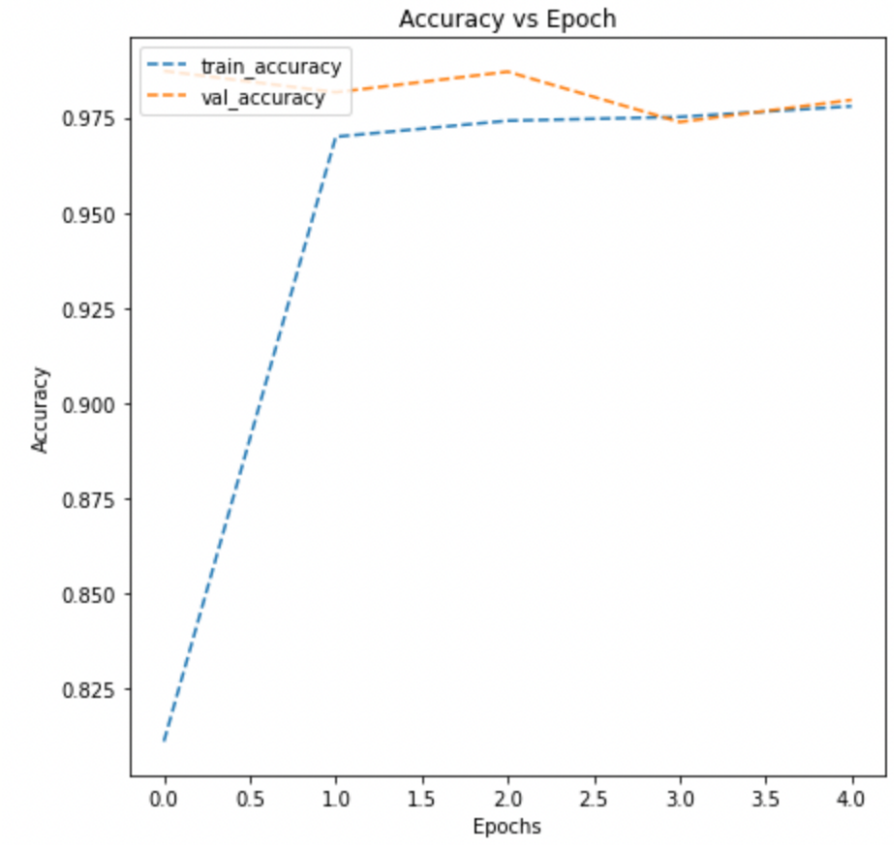
it gives us a validation accuracy of almost 98% and, based on the training accuracy, it doesn't look like our model is overfitting as much as our first model. This can be verified with the following learning curve above.

```
[46] #creating sequential model
     model1=Sequential()

     model1.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu",input_shape=(64,64,3)))
     model1.add(MaxPooling2D(pool_size=2))
     model1.add(Dropout(0.2))

     model1.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
     model1.add(MaxPooling2D(pool_size=2))
     model1.add(Dropout(0.2))

     model1.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
     model1.add(MaxPooling2D(pool_size=2))
     model1.add(Dropout(0.2))

     model1.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
     model1.add(MaxPooling2D(pool_size=2))
     model1.add(Dropout(0.2))

     model1.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
     model1.add(MaxPooling2D(pool_size=2))
     model1.add(Dropout(0.2))

     model1.add(Flatten())

     model1.add(Dense(500,activation="relu"))
     model1.add(Dropout(0.4))
     model1.add(Dense(2,activation="softmax"))#2 represent output layer neurons
     model1.summary()
```

Evaluation of the improved model(model1)

At the same time, FP count also decreased (to 27). A lower value FP is also expected from our model since this will prevent the patient from further undergoing unnecessary laboratory tests and treatment and will reduce financial burden on the health provider.

```
plot_accuracy(history1)
```



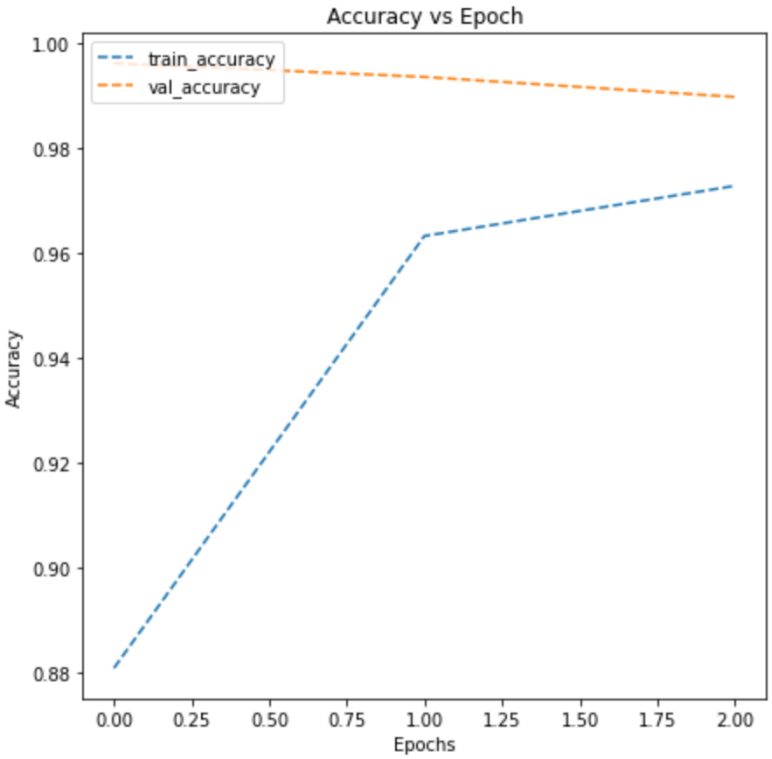
The validation accuracy is also increasing with the increase in epochs. Same as training accuracy, validation accuracy also has a rapid increase up to around 2 epochs and then has lower increase, almost like a constant.
We have received a base accuracy of 98% with high precision and recall towards classifying the infected and normal cells which is reasonable. By investigating the confusion matrix, we can see that the count for False Negatives (FN) is reduced to 16.
FN indicates that the model declares a malaria patient to be healthy whereas the patient is parasitized. This will severely hamper the patient treatment and may result in death. Our goal is to reduce this number with the proposed improved model. A reduced number of FN will ensure that our model is effective in identifying parasitized cell images.
The overall accuracy of this model is better than the baseline model.

Model 2: with Batch Normalization
To improve the performance of our previous model we added some Batch normalization and used LeakyRelu as our activation function.

```
#Building the Model2

model2 = Sequential()
model2.add(Conv2D(32, (3,3), input_shape=(64, 64, 3), padding='same'))
model2.add(LeakyReLU(0.1))

model2.add(Conv2D(32, (3, 3),input_shape = (32, 32, 1),  padding = "same"))
model2.add(LeakyReLU(0.1))
model2.add(MaxPool2D(pool_size = (2, 2)))

model2.add(BatchNormalization())

model2.add(Conv2D(32, (3,3), input_shape=(64, 64, 3), padding='same'))
model2.add(LeakyReLU(0.1))



model2.add(Conv2D(filters = 64, kernel_size = (3, 3), padding = "same"))
model2.add(LeakyReLU(0.1))
model2.add(MaxPool2D(pool_size = (2, 2)))

model2.add(BatchNormalization())

model2.add(Flatten())

model2.add(Dense(512))
model2.add(LeakyReLU(0.1))
model2.add(Dropout(0.4))

model2.add(Dense(2,activation="softmax"))

adam = optimizers.Adam(learning_rate=0.001)
```

We have received a base accuracy of 98% with high precision (99) and recall (98) towards classifying the infected and normal cells which is reasonable.
By investigating the confusion matrix as shown in the notebook, we can see that the count for False Negatives (FN) is high (27) ...

Evaluation of the model training and validation accuracy

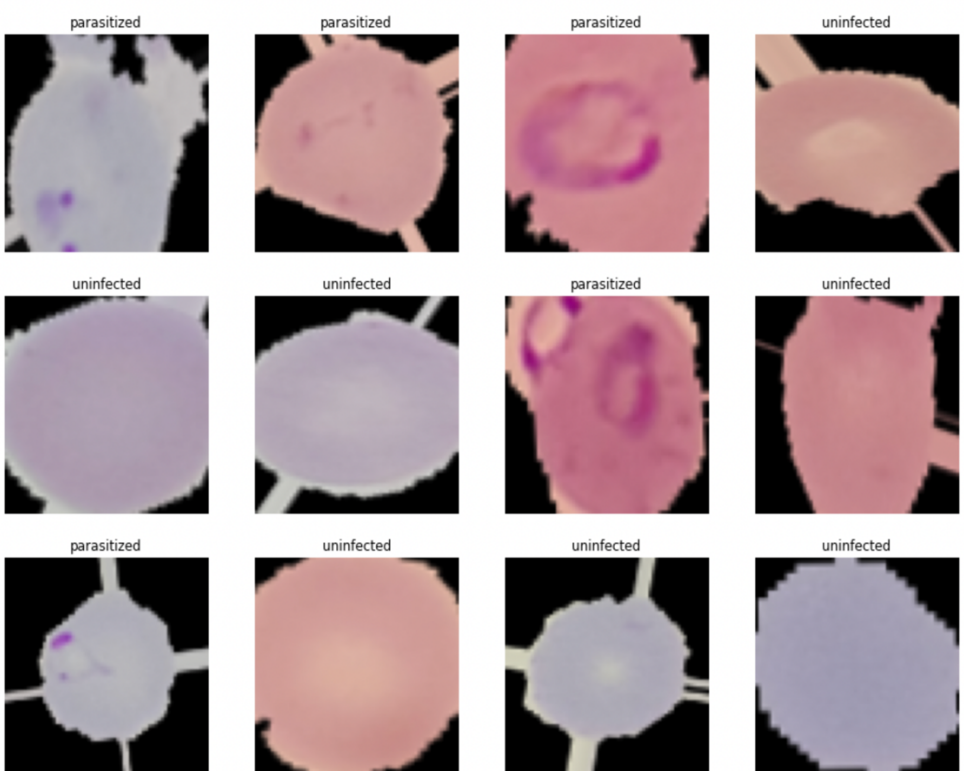


Learning curve of this underfit model has a high validation loss at the beginning which gradually lowers upon adding training examples and suddenly falls to an arbitrary minimum at the end. Training loss and validation loss are close to each other at the end also.

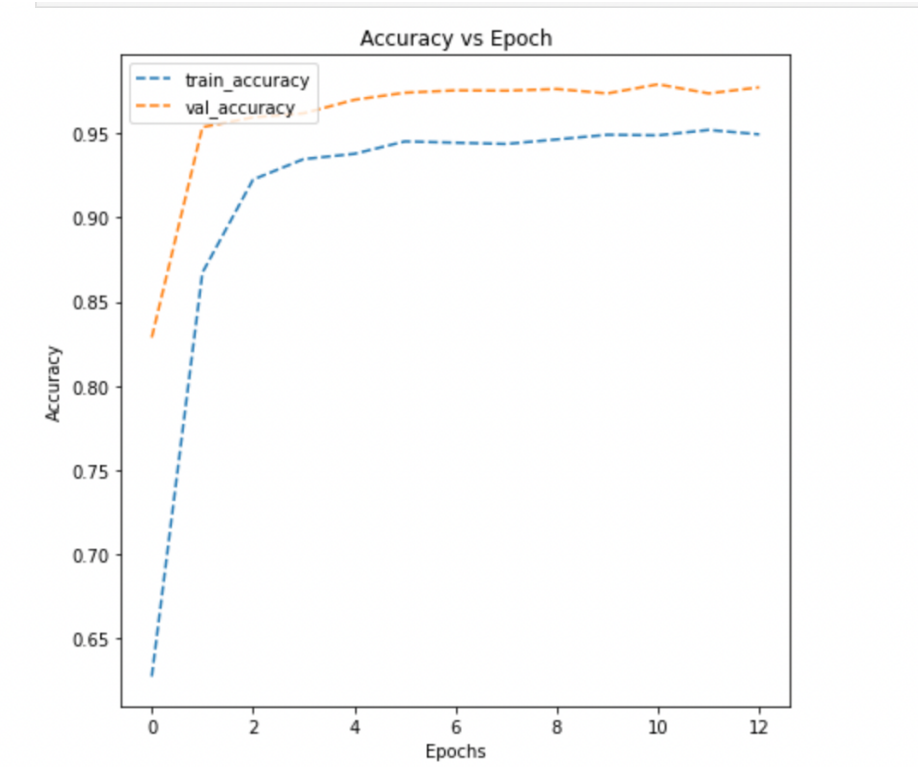
The validation accuracy of model 2 (97%) is lower in comparison to previous model, unfortunately the model has done poorly on the validation data and is overfitting the training data also.

Model 3 with Data Augmentation

Amongst the popular deep learning applications, computer vision tasks such as image classification, object detection, and segmentation have been highly successful. Data augmentation can be effectively used to train the DL models in such applications. After Using ImageDataGenerator to generate images, we were able to see that The infected cells seem to contain some red globular structures whereas healthy cells do not seem to contain such structures in them.
The proposed deep learning model with data augmentation will be used to identify these patterns in cell images to effectively detect malaria parasites in a patient.
We were able to visualize infected cells seem to contain some red globular structures whereas healthy cells do not seem to contain such structures in them. The proposed deep learning model will be used to identify these patterns in cell images to effectively detect malaria parasites in a patient.



Evaluation of the model training and validation accuracy



We can see that our model 3 does not converge well and a significant difference between the training and validation results both for loss and accuracy. This indicates that our base model is not trained well and might be overfitting to training data. Consequently, the model might not generalize well on unseen test data.

Deep transfer learning

Just like humans have an inherent capability to transfer knowledge across tasks, transfer learning enables us to utilize knowledge from previously learned tasks and apply it to newer, related ones, even in the context of machine learning or deep learning.

The idea we want to explore in this project is:

Can we leverage a pre-trained deep learning model (which was trained on a large dataset, to solve the problem of malaria detection by applying and transferring its knowledge in the context of our problem?

We will be using the pre-trained VGG-16 deep learning model, developed by the Visual Geometry Group (VGG) at the University of Oxford, for our experiments. A pre-trained model like VGG-16 is trained on a huge dataset (ImageNet) with a lot of diverse image categories. Therefore, the model should have learned a robust hierarchy of features, which are spatial-, rotational-, and translation-invariant regarding features learned by CNN models. Hence, the model, having learned a good representation of features for over a million images, can act as a good feature extractor for new images suitable for computer vision problems like malaria detection. Let's discuss the VGG-16model architecture before unleashing the power of transfer learning on our problem.

Model 4 Pre-trained model (VGG16)

For building this model, we will leverage TensorFlow to load up the VGG-16 model and freeze the convolution blocks so we can use them as an image feature extractor. We will plug in our own dense layers at the end to perform the classification task.

```
transfer_layer = vgg.get_layer('block5_pool')
vgg.trainable=False

# Add classification layers on top of it

x = Flatten()(transfer_layer.output) #Flatten the output from the 3rd block of the VGG16 model
x = Dense(256, activation='relu')(x)

# Similarly add a dense layer with 128 neurons
x = Dropout(0.3)(x)

# Add a dense layer with 64 neurons

x = BatchNormalization()(x)
pred = Dense(64, activation='softmax')(x)

model4 = Model(vgg.input, pred) #Initializing the model
```

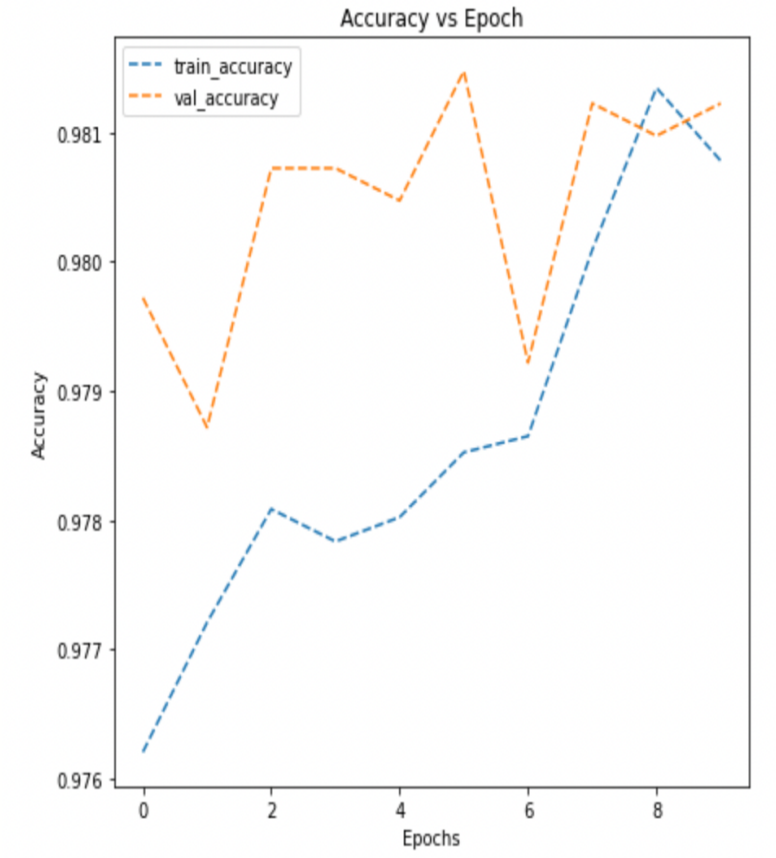
```
[94] from tensorflow.keras.applications.vgg16 import VGG16
     from tensorflow.keras import Model

     vgg = VGG16(include_top=False, weights='imagenet', input_shape=(64,64,3))
     vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792
```

0s completed at 11:52 PM

After training our model using similar configurations and callbacks to the ones we used in our previous model.

we then plotted the train and validation accuracy for model 4, we notice a validation accuracy of almost 98% and, based on the training accuracy, it looks like the model is overfitting, but not as much as the previous model (model 3).



Different network architecture results on each of the metrics

Models	***Test Accuracy***	***Precision***	***recall***	***F1 score***
Baseline model	*98*	*98*	*98*	*98*
Model 1	*98*	*98*	*99*	*98*
Model 2	*98*	*99*	*98*	*99*

Model 3	*97*	*98*	*97*	*98*
Model 4	*98*	*98*	*99*	*98*

We have used accuracy, precision, recall, F1-score, to evaluate the performance of our models. Since in our dataset the number of samples from each target class is equal, we consider accuracy as our primary metric. Accuracy refers to the proportion of correct predictions over all predictions made by the model. In addition, we calculated precision, recall or sensitivity, specificity, and F1-score from the confusion matrix which contains False Positives (FP), True Positives (TP), False Negatives (FN), and True Negatives (TN).

Furthermore, precision and recall for each target class are calculated from a classification report. Precision measures the proportion of patients that are identified as infected really carry malaria parasites. Recall or sensitivity measures of the proportion of patients that are infected are diagnosed by the model as having malaria parasites. Specificity is the opposite of recall which measures the proportion of patients that are not infected and diagnosed by the model as not carrying any malaria parasites. F1-score is calculated as a single metric from the harmonic mean of precision and recall

Conclusion:

We have evaluated our network architectures on different metrics, namely, Accuracy, Loss, Precision, Recall, F1 score, conclusion
Our best model(model1) achieves an accuracy of 98%, in addition, a high recall value of 99% indicates model sensitivity in predicting infected cells with malaria. Furthermore, the number of false negative (FN) cases significantly (almost half) decreased in our best improved model compared to the base model which indicates the success of our model in reducing the risk of identifying a malaria patient as healthy which is detrimental to a patient's line of treatment.
It is worth mentioning here that our model did not consider using data augmentation to artificially increase the size of the training dataset largely since our dataset contains a decent number (12,000) of segmented cell images with the same number of normal and parasitized instances well enough to train a deep learning model without significantly running into overfitting problem. Hence, we trained our model without artificially augmenting our dataset and yet obtained better or comparable performance to the techniques in identifying a malaria patient.

.
The model also yields a high value of F1 score (98%) compared to all other existing models under study, indicating a strong correlation between predicted and true labels. We also observed that the proposed improved model showed better performance compared to the customized and other CNN models (pretrained such as VGG-16)
Thus, we believe that the results obtained from this work will benefit towards developing valuable solutions so that reliability of the treatment and lack of medical expertise can be solved. As an immediate extension of this work, we will consider using image augmentation on the training data with the hope to further alleviate overfitting problem and different adaptive variants of the SGD optimizer to observe their impact on the performance results. In the future, we also plan to achieve better prediction by using ensemble methods through model stacking.