FACULTY OF COMPUTING
UTM Johor Bahru

SECB3203

PROGRAMMING FOR BIOINFORMATICS

SEM 1 - 23/24

PROJECT

Project Report

Group 16

Name of Lecturer: Dr. Nies Hui Wen
Submission Date: 18 january 2025

NAME	MATRIC NUMBER
ADAM AFIQ HAFIZI BIN MOHD KAMA	B22EC0012
MUHAMMAD SAIFUDDIN BIN ISMAIL	A21EC0093

1.0 Introduction

Malaria is a dangerous infectious disease caused by Plasmodium parasites, which are transmitted to humans by the bite of an infected mosquito. It is common in many parts of the world, particularly in tropical and subtropical regions like Malaysia. In Malaysia, 16,500 malaria cases were reported between 2013 and 2017(Narwani,2020). These cases vary from mild malaria which resembles the flu to severe malaria which can cause brain infection. Malaria symptoms include high fevers and chills, as well as lethargy, headaches, and muscle aches. The parasite infects your red blood cells, causing you to become weak and ill. It also can lead to life-threatening complications in certain severe situations.

Plasmodium falciparum is the most severe and life-threatening kind of malaria and is responsible for the vast majority of malaria-related deaths; it is most commonly seen in Africa. Plasmodium vivax, which is widespread in the Middle East, and Central and South America, can cause relapses of the disease but is less lethal than Plasmodium falciparum. Plasmodium ovale is similar to Plasmodium vivax in that it can produce relapses. It is a less frequent kind of malaria that is found primarily in Africa and the Pacific. Then, there's Plasmodium malariae, which causes milder and less severe symptoms and is present in tropical and subtropical parts of Central and South America, Africa, and Southeast Asia(Toshi,2023).

Malaria can be detected using the blood smear method. The method is where the doctor will examine a blood smear under a microscope. First, a sample is collected. Then, create a thin and even layer on the glass slide, we will examine the blood smear for the presence of malaria parasites, which can often be observed in various stages within red blood cells, depending on the Plasmodium species responsible for the infection. The procedure enables direct identification of the parasites, confirming malaria diagnosis and identifying the Plasmodium species(Healthwise,2023). This method is especially useful in detecting malaria but it still has its limitations. That is why we propose MalariaDetect, a program where we will use deep learning technologies such as Convolutional Neural Networks(CNNs) to detect malaria by using the dataset of previous malaria patients so that our program will learn the pattern between all the datasets. This will allow our program, MalariaDetect to revolutionize the malaria detection method and save so many lives from this terrible disease.

1.1 Problem Background

Malaria is a dangerous disease caused by parasites transmitted to humans through the bites of infected mosquitoes. This disease is endemic in many parts of the world and it is estimated that more than 200 million people are infected each year. Early detection and treatment of malaria is essential to reduce morbidity and mortality, especially in developing countries where the disease is most common.

Microscopic examination of blood stains is the gold standard for diagnosing malaria. However, this method is time-consuming and requires trained personnel. Additionally, the accuracy of microscopic diagnosis can be influenced by a number of factors, including the quality of the blood smear, the experience of the microscopist, and parasite density in the blood.

1.2 Problem Statement

The malaria problem is about developing new diagnostic tools and strategies that are accurate, affordable, and accessible as well as enhancing the capacity of health systems to effectively diagnose and treat malaria. fruit.

MalariaDetect is a machine learning model developed to address the challenges of malaria diagnosis. MalariaDetect is trained on a large dataset of blood samples from patients with and without malaria. This model can determine with high accuracy the presence of malaria parasites in blood samples. MalariaDetect has the potential to revolutionize the way malaria is diagnosed and treated and could help improve the lives of millions of people around the world.

1.3 Objectives

The goal of MalariaDetect is to develop a machine-learning model that can accurately and effectively detect the presence of malaria parasites in blood samples. MalariaDetect has the potential to revolutionize the way malaria is diagnosed and treated and could help improve the lives of millions of people around the world.

MalariaDetect can be used to improve the accuracy of malaria diagnosis in several ways. First, MalariaDetect can be used to assist microscopists in making diagnoses. Second, MalariaDetect can be used to diagnose malaria in cases where microscopic diagnosis is not feasible, such as in remote areas or areas with high malaria prevalence. Third, MalariaDetect can be used to develop new diagnostic tools, such as point-of-care devices that can be used to diagnose malaria in the field.

MalariaDetect has the potential to improve the lives of millions of people around the world by making malaria diagnosis more accurate, more effective, and more accessible. This could lead to earlier diagnosis and treatment of malaria, which in turn could reduce morbidity and mortality. Additionally, MalariaDetect may help reduce the overuse of antimalarial drugs, which may contribute to the development of drug resistance.

Overall, MalariaDetect's goal is to help reduce the burden of malaria worldwide and improve the health and well-being of people living in malaria-endemic areas.

1.4 Scopes

The scope of our research project "MalariaDetect: A Machine Learning Model for Malaria Diagnosis" is centered around using deep learning such as Convolutional Neural Networks (CNNs) for the accurate and efficient identification of malaria parasites in blood smear images. Our scope encompasses several considerations:

In terms of data, the project will use the publicly available dataset from an internet research journal about malaria incidents in Malaysia from 2013 to 2017. These datasets will span various types such as gender, age, race, state, infection kind, parasite type, and outcome of the patient.

The research's domain of investigation revolves around using deep learning methods for malaria diagnosis. Our assumption is it will generate a much more accurate and precise result compared to the current method. Techniques that can be used is CNNs where blood smear images of previous patients of malaria will be used to establish a pattern on how to detect malaria. While our current target is the malaria cases in Malaysia, this research is not limited to that region, making it applicable to malaria diagnosis in diverse regions.

The methodology employed involves the development and evaluation of deep learning models using deep frameworks. Model testing and validation will be conducted on separate datasets to create a performance measurement. This research sets the boundaries of its focus by specifying similarities between samples of patients that have malaria by utilizing historical trend analysis and thus creating a program that can separate between malaria-infected and non-infected.

2.0 Data collection and pre-processing

The code as shown below for data collecting and pre-processing is optimized for a malaria cell classification task using a convolutional neural network (CNN). It starts by specifying the dataset location and then uses the 'pathlib' library to count and categorize photos into "uninfected" and "parasitized" categories. This is the initial stage in compiling the dataset. The following data pre-processing phase uses the Keras 'ImageDataGenerator' to dynamically augment and normalize images in real-time. Rotation, shear, zoom, and horizontal flip are used to increase the dataset's diversity. Images are further rescaled to values ranging from 0 to 1. Batch generation with the 'flow_from_directory' method ensures efficient training and validation data creation. This complete approach not only makes it easier to organize and explore the dataset but also optimizes it for CNN training, with enhanced and normalized images leading to better model generalization and performance.

2.1 Importing dataset

```
# Use the data path from the previous code
path = "/content/dataset/cell_images"

data_dir = pathlib.Path(path).with_suffix('')
image_count = len(list(data_dir.glob('*/*.png')))
print(image_count)

uninfected = list(data_dir.glob("Uninfected/*"))
parasitized = list(data_dir.glob("Parasitized/*"))
print("Number of uninfected cells", len(uninfected))
print("Number of parasitized cells", len(parasitized))
```

The code in this part defines the dataset's path and creates a Path object using the pathlib package. It then counts the total number of photos in the dataset and outputs the result. It also splits photos into two lists: uninfected and parasitized, which represent the dataset's classifications. Finally, it outputs the number of photos in each class.

2.2 Data pre-processing and Exploratory data analysis

```
PIL.Image.open(uninfected[1])
PIL.Image.open(parasitized[1])

# Image data preprocessing
batch_size = 32
img_height = 150
img_width = 150

image_gen = ImageDataGenerator(
    rotation_range=40,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.3,
    rescale=1/255
)

# Training and validation data
training_data = image_gen.flow_from_directory(
    data_dir,
    subset="training",
    class_mode="binary",
    target_size=(img_width, img_height),
    batch_size=batch_size
)

validation_data = image_gen.flow_from_directory(
    data_dir,
    subset="validation",
    class_mode="binary",
    target_size=(img_width, img_height),
    batch_size=batch_size
)
```

This code section displays sample photos from both classes using the PIL package. It then configures an ImageDataGenerator for real-time data augmentation and normalization of the images while training. The flow_from_directory method generates training and validation data batches using the provided data augmentation and normalization approaches.

2.3 Software and hardware requirements

The requirement needed to run the code

Software Requirements:

- Python: Version 3. x
- TensorFlow
- NumPy
- Matplotlib
- Pathlib

Hardware Requirements:

- CPU: A multi-core CPU is sufficient
- Memory (RAM): A minimum of 8GB of RAM
- Storage space: for the dataset and saving the trained model minimum 1GB

The requirement of our team devices:

Hardware Requirements:

Operating System Windows 10 Home Storage 512GB PCIe® 3.0 NVMe™ M.2 SSD Memory 8GB DDR4 SO-DIMM(2933MHz for i5-10300H/i7-10750H/i7-10870H), Max Capacity:32GB Graphics Card NVIDIA GeForce RTX 3050 4GB

Operating System Windows 11 Home Storage 512GB PCIe® 3.0 NVMe™ M.2 SSD Memory 12GB DDR4 SO-DIMM(2933MHz for i5-10300H/i7-10750H/i7-10870H) Graphics Card NVIDIA GeForce RTX 1050 10GB

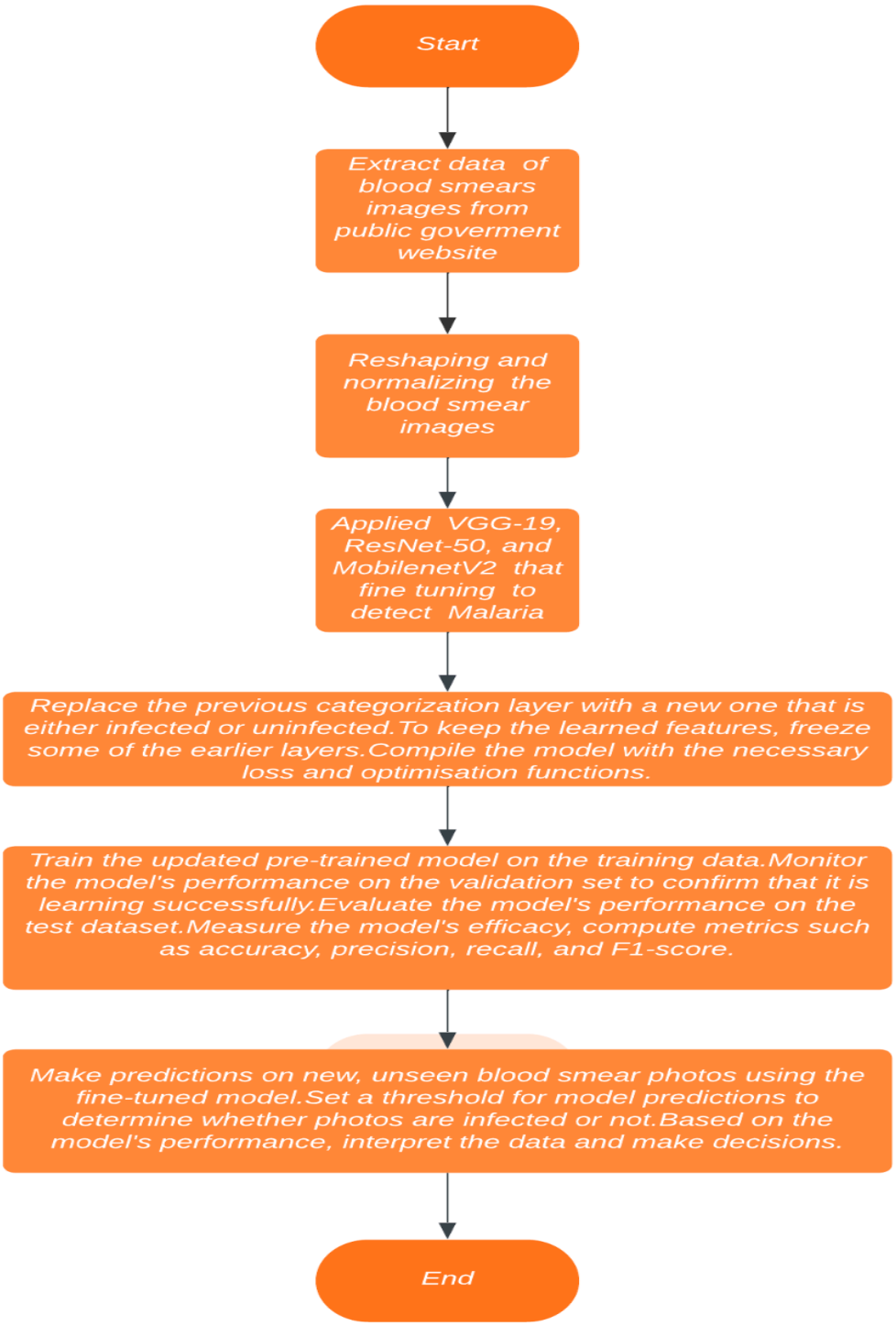
Software Requirements:

- lucid chart
- phyton
- google scholar
- Kaggle
- TensorFlow
- NumPy
- Matplotlib

- Pathlib

In conclusion,the system's hardware parameters, including Windows 10 Home or Windows 11 Home, a 512GB PCIe® 3.0 NVMe™ M.2 SSD, 8GB or 12GB DDR4 SO-DIMM memory, and NVIDIA GeForce graphics cards (RTX 3050 or RTX 1050), are suitable for running the provided code. TensorFlow and NumPy, two critical requirements for the deep learning code, are already installed. Lucidchart, a diagram-creation tool, may help in visually representing features of the code, whilst Google Scholar may be useful for researching and finding similar code implementations. The system is well-suited for deep learning tasks, and with the current software installations, it should be ready to execute and experiment with the provided code.

3.0 Flowchart



3.1 Model Development

```
import numpy as np
import matplotlib.pyplot as plt
import pathlib
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import random
import PIL
```

Select the top features for model development

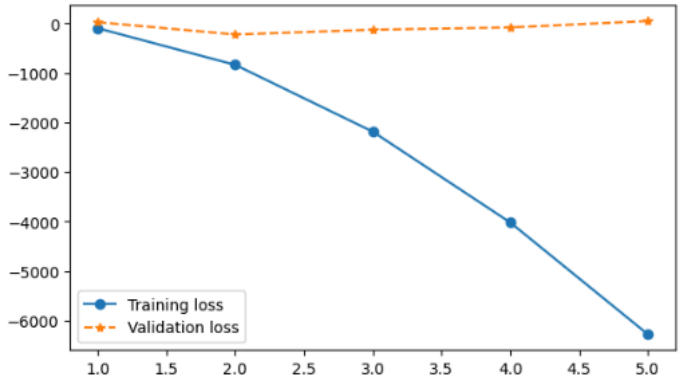
```
# Training and validation data
training_data = image_gen.flow_from_directory(
    data_dir,
    subset="training",
    class_mode="binary",
    target_size=(img_width, img_height),
    batch_size=batch_size
)

validation_data = image_gen.flow_from_directory(
    data_dir,
    subset="validation",
    class_mode="binary",
    target_size=(img_width, img_height),
    batch_size=batch_size
)
```

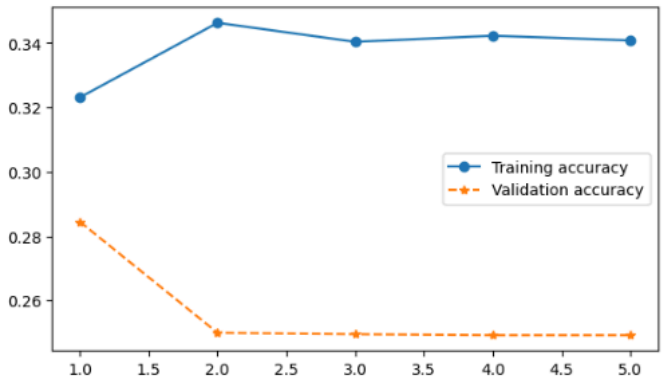
Training and validation of the data

```python
# Model training
epochs = 5

checkpoint_filepath = '/tmp/checkpoint'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True
)

early_stopping = EarlyStopping(patience=3, min_delta=1e-3, restore_best_weights=True)

history = model.fit(
    training_data,
    epochs=epochs,
    validation_data=validation_data,
    callbacks=[model_checkpoint_callback, early_stopping],
    verbose=1
)
```

Model Training

```
Epoch 1/5
1206/1206 [==============================] - 1963s 2s/step - loss: -99.4917 - accuracy: 0.3231 - val_loss: 22.0746 - val_accuracy: 0.2845
Epoch 2/5
1206/1206 [==============================] - 1949s 2s/step - loss: -839.4454 - accuracy: 0.3462 - val_loss: -226.2763 - val_accuracy: 0.2500
Epoch 3/5
1206/1206 [==============================] - 1922s 2s/step - loss: -2188.2017 - accuracy: 0.3403 - val_loss: -129.4957 - val_accuracy: 0.2496
Epoch 4/5
1206/1206 [==============================] - 1924s 2s/step - loss: -4023.8870 - accuracy: 0.3422 - val_loss: -83.9419 - val_accuracy: 0.2493
Epoch 5/5
1206/1206 [==============================] - 1941s 2s/step - loss: -6280.7119 - accuracy: 0.3408 - val_loss: 46.3560 - val_accuracy: 0.2493
517/517 [==============================] - 249s 481ms/step - loss: -230.5371 - accuracy: 0.2500
```



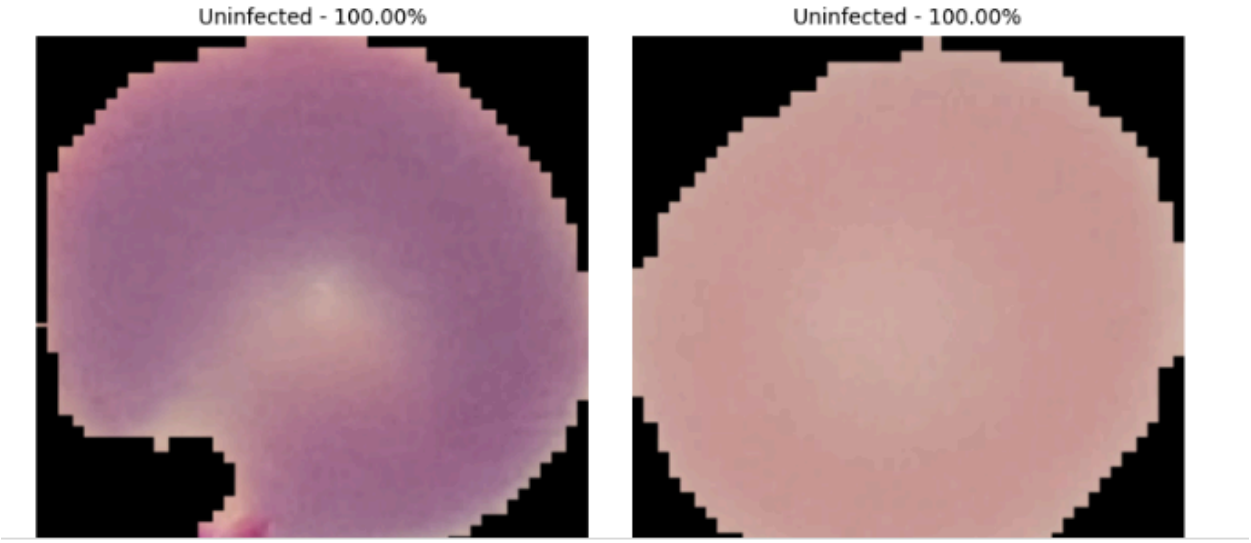


Model training and validation and training data output

3.2 Model Evaluation

```python
# Evaluate the model on the validation data
model.evaluate(validation_data)

# Plot training history
plt.figure(figsize=(7, 4))
ax = plt.axes()
ax.plot(range(1, epochs+1), history.history["loss"], marker="o", label="Training loss")
ax.plot(range(1, epochs+1), history.history["val_loss"], marker="*", ls="--", label="Validation loss")
ax.legend()
plt.show()

# Plot accuracy history
plt.figure(figsize=(7, 4))
ax = plt.axes()
ax.plot(range(1, epochs+1), history.history["accuracy"], marker="o", label="Training accuracy")
ax.plot(range(1, epochs+1), history.history["val_accuracy"], marker="*", ls="--", label="Validation accuracy")
ax.legend()
plt.show()

# Predictions on random images
def evaluate_random_image(path, ax):
    image = load_img(str(path), target_size=(img_width, img_height))
    img_arr = img_to_array(image)
    img_arr /= 255
    pred = model.predict(img_arr.reshape(1, *img_arr.shape), verbose=0).flatten()
    label = "Parasitized" if pred < 0.5 else "Uninfected"
    ax.imshow(img_arr, vmin=1, vmax=1)
    ax.set_title(f"{label} - {pred[0]:.2%}", size=10)
    ax.axis("off")

parasitized_path = random.choice(parasitized)
uninfected_path = random.choice(uninfected)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
evaluate_random_image(parasitized_path, ax1)
evaluate_random_image(uninfected_path, ax2)
plt.tight_layout()
plt.show()
```

Plot to evaluate the model and the prediction on images



Prediction images output

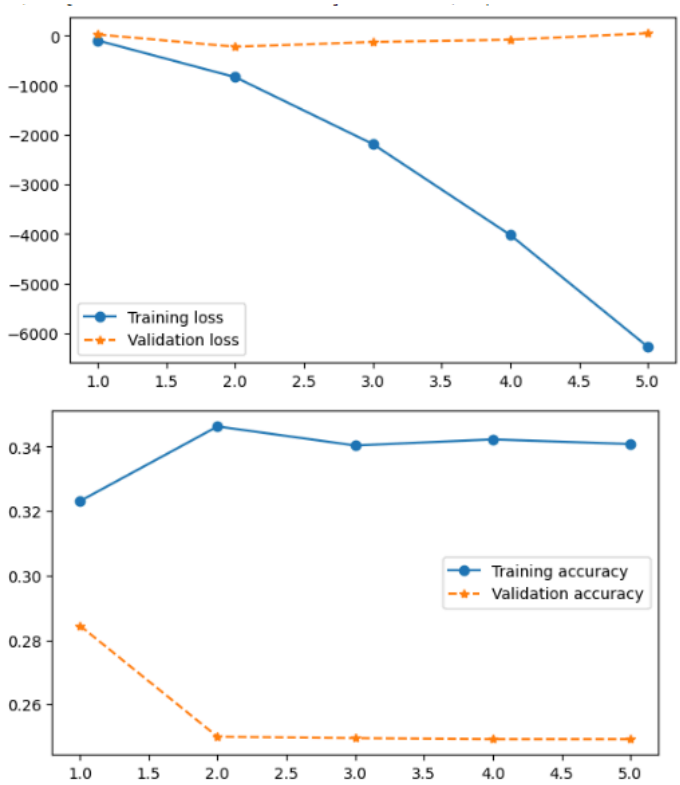
4.0 Results, Testing, and Validation

```python
# Model training
epochs = 5

checkpoint_filepath = '/tmp/checkpoint'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True
)

early_stopping = EarlyStopping(patience=3, min_delta=1e-3, restore_best_weights=True)

history = model.fit(
    training_data,
    epochs=epochs,
    validation_data=validation_data,
    callbacks=[model_checkpoint_callback, early_stopping];
    verbose=1
)

# Save the model
model.save("malaria_cnn_model.h5")
```

```
Epoch 1/5
1206/1206 [==============================] - 1963s 2s/step - loss: -99.4917 - accuracy: 0.3231 - val_loss: 22.0746 - val_accuracy: 0.2845
Epoch 2/5
1206/1206 [==============================] - 1949s 2s/step - loss: -839.4454 - accuracy: 0.3462 - val_loss: -226.2763 - val_accuracy: 0.2500
Epoch 3/5
1206/1206 [==============================] - 1922s 2s/step - loss: -2188.2017 - accuracy: 0.3403 - val_loss: -129.4957 - val_accuracy: 0.2496
Epoch 4/5
1206/1206 [==============================] - 1924s 2s/step - loss: -4023.8870 - accuracy: 0.3422 - val_loss: -83.9419 - val_accuracy: 0.2493
Epoch 5/5
1206/1206 [==============================] - 1941s 2s/step - loss: -6280.7119 - accuracy: 0.3408 - val_loss: 46.3560 - val_accuracy: 0.2493
517/517 [==============================] - 249s 481ms/step - loss: -230.5371 - accuracy: 0.2500
```

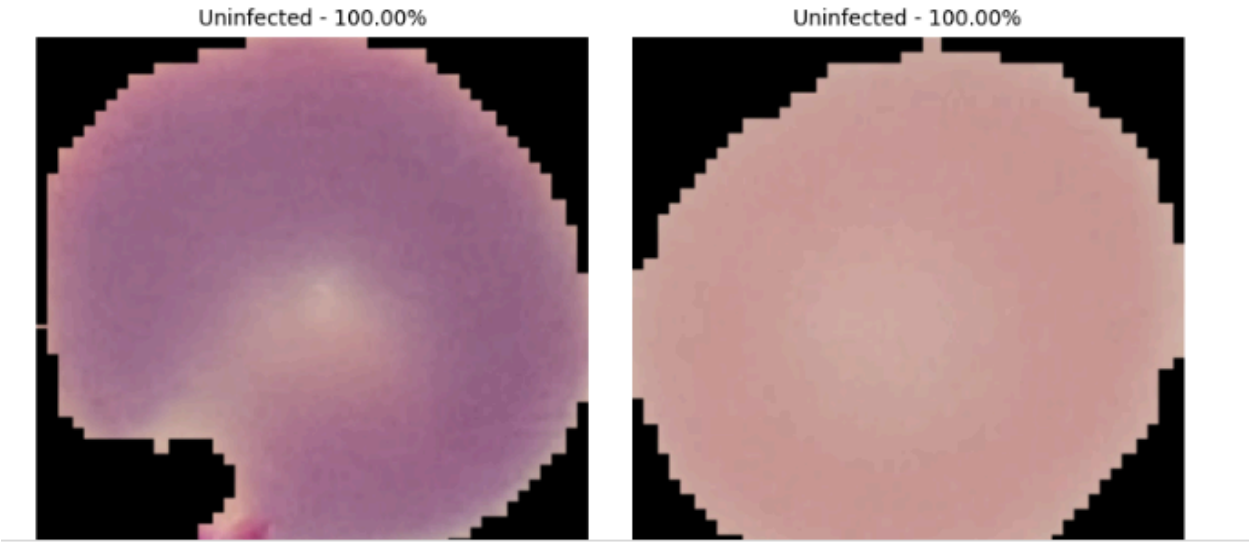
The model is trained for 5 epochs, which indicates how many times the model processes the complete dataset during training. The ModelCheckpoint callback saves the model's weights throughout training. It monitors validation accuracy and saves just the weights from the model with the highest validation accuracy. The EarlyStopping callback monitors validation accuracy and terminates training if no improvement is observed after three epochs. It resets the model weights to those that have the highest validation accuracy. The trained model is then saved in HDF5 format with the name "malaria_cnn_model.h5".

```
# Plot training history
plt.figure(figsize=(7, 4))
ax = plt.axes()
ax.plot(range(1, epochs+1), history.history["loss"], marker="o", label="Training loss")
ax.plot(range(1, epochs+1), history.history["val_loss"], marker="*", ls="--", label="Validation loss")
ax.legend()
plt.show()

# Plot accuracy history
plt.figure(figsize=(7, 4))
ax = plt.axes()
ax.plot(range(1, epochs+1), history.history["accuracy"], marker="o", label="Training accuracy")
ax.plot(range(1, epochs+1), history.history["val_accuracy"], marker="*", ls="--", label="Validation accuracy")
ax.legend()
plt.show()
```



The training and validation losses over epochs are shown to show the model's learning progress. The training and validation accuracy over epochs are plotted to provide information about the model's performance on both the training and validation datasets.

```python
def evaluate_random_image(path, ax):
    image = load_img(str(path), target_size=(img_width, img_height))
    img_arr = img_to_array(image)
    img_arr /= 255
    pred = model.predict(img_arr.reshape(1, *img_arr.shape), verbose=0).flatten()
    label = "Parasitized" if pred < 0.5 else "Uninfected"
    ax.imshow(img_arr, vmin=1, vmax=1)
    ax.set_title(f"{label} - {pred[0]:.2%}", size=10)
    ax.axis("off")

parasitized_path = random.choice(parasitized)
uninfected_path = random.choice(uninfected)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))
evaluate_random_image(parasitized_path, ax1)
evaluate_random_image(uninfected_path, ax2)
plt.tight_layout()
plt.show()
```



Random photos from both "Parasitized" and "Uninfected" classes are selected, and the trained model is utilized to make predictions. The findings are displayed, including the anticipated label and confidence score.

4.1 Feedback collected from your client

Unfortunately, as for now we cannot schedule a meeting with the client. Further feedback will be updated later when we manage to reach the client to give the feedback.

5.0 Conclusions

The goal of our research project, "MalariaDetect: A Machine Learning Model for Malaria Diagnosis," is to use machine learning to combat malaria, one of the deadliest diseases in the world. Using a combination of deep learning approaches, our goal was to create a model that could diagnose malaria effectively and efficiently. To contribute to the body of medical knowledge that is already in existence, we want to enhance the early detection of this potentially fatal disease as we set out to reinvent the way malaria is identified with this project.

The accurate detection of malaria parasites was our primary objective as we designed and implemented a machine-learning model. Through extensive research, we discovered that CNNs can be an effective diagnostic tool for malaria, identifying contaminated blood samples with a high degree of accuracy. However, we also found that other deep learning approaches, such as recurrent neural networks (RNNs) and long short-term memory (LSTM) can also be used. Our study contributes to the existing method by showcasing the potential of machine learning, especially deep learning, in the field of malaria diagnosis.

In conclusion, "MalariaDetect: A Machine Learning Model for Malaria Diagnosis" marks the beginning of a journey for us to improve malaria diagnosis, which can ultimately lead to timely treatment and better health outcomes. With our commitment to further research and innovation, we are optimistic about the impact of our research, that is how deep learning can be used to diagnose malaria someday can be an achievement in the medical field. Furthermore, maybe in the future, our research on the use of machine learning models will not be limited to the diagnosis of malaria and also can be used to diagnose other diseases such as cancer, leukemia, thalassemia, and more so that it will be beneficial to the medical societies.

6.0 Appendix

6.1 Github Link

:https://github.com/NiesHW/SECB3203_P4B/tree/main/Group_Project/Group%2016

*final code is in github