# Android App for Malaria Detection with Microscopic Blood Image Using Deep Learning Model

A Project Progress Report
Submitted in partial fulfillment for the degree of

**BACHELOR OF TECHNOLOGY**
in
**COMPUTER SCIENCE AND ENGINEERING**
Submitted by


| | |
|---|---|
| P C Avinash Kumar Reddy | (N190700) |
| M.Shakunthala | (N190684) |
| P.Sai Bhavani | (N190660) |
| K.Mahesh | (N190686) |
| B.Sirisha | (N190641) |
| G.Sri Vidya | (N190142) |


Under the Esteem Guidance of

Mr.K KRISHNA SINGH(Asst Prof)


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Rajiv Gandhi University of Knowledge Technologies
Nuzvid, Eluru, Andhra Pradesh – 521202

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – Nuzvid

Nuzvid, Eluru, Andhra Pradesh – 521202

## CERTIFICATE OF COMPLETION

This is to certify that the work entitled, "**Android App for Malaria Detection with Microscopic Blood Image Using Deep Learning Model** " is the bonafide work of **P C Avinash Kumar Reddy** (ID No: N190700), **M.Shakunthala** (ID No: N190684), **P.Sai Bhavani** (ID No: N190660), **K.Mahesh** (ID No: N190686), **B.Sirisha** (ID No: N190641),**G.Sri Vidya** (ID No: N190142) carried out under our guidance and supervision for the 3rd year project of Bachelor of Technology in the department of Computer Science and Engineering under RGUKT IIIT Nuzvid. This work was done during the academic session May 2024 – June 2024, under our guidance.

---

Mr.K Krishna Singh
Assistant Professor,
Department of CSE,
RGUKT Nuzvid.

---

Dr. D.V.Nagarjana Devi
Assistant Professor,
Head of the Department,
Department of CSE,
RGUKT Nuzvid.

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies
Nuzvid, Eluru, Andhra Pradesh – 521202

## CERTIFICATE OF EXAMINATION

This is to certify that the work entitled, "**Android App for Malaria Detection with Microscopic Blood Image Using Deep Learning Model** " is the bonafide work of **P C Avinash Kumar Reddy** (ID No: N190700), **M.Shakunthala** (ID No: N190684), **P.Sai Bhavani** (ID No: N190660), **K.Mahesh** (ID No: N190686), **B.Sirisha** (ID No: N190641),**G.Sri Vidya** (ID No: N190142) and hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in final year of Bachelor of Technology for which it has been submitted. This approval does not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn, as a recorded in this thesis. It only signifies the acceptance of this thesis for the purpose for which it has been submitted.

Mr.K Krishna Singh
Assistant Professor,
Department of CSE,
RGUKT-NUZVID

Project Examiner
RGUKT-Nuzvid

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies
Nuzvid, Eluru, Andhra Pradesh – 521202

## DECLARATION

**P C Avinash Kumar Reddy** (ID No: N190700), **M.Shakunthala** (ID No: N190684), **P.Sai Bhavani** (ID No: N190660), **K.Mahesh** (ID No: N190686), **B.Sirisha** (ID No: N190641),**G.Sri Vidya** (ID No: N190142) hereby declare that the project report entitled "**Android App for Malaria Detection with Microscopic Blood Image Using Deep Learning Model** " done by us under the guidance of **Mr.K Krishna Singh**, Assistant Professor is submitted for the partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering during the academic session January 2024 - June 2024 at RGUKT-Nuzvid.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites are mentioned in the references..

Date: 22-06-2024
Place: Nuzvid

P C Avinash Kumar Reddy (N190700)
M.Shakunthala (N190684)
P.Sai Bhavani (N190660)
K.Mahesh (N190686)
B.Sirisha (N190641)
G.Sri Vidya (N190142)

# ACKNOWLEDGEMENT

We would like to express our heartfelt profound gratitude and deep regards to our guide **Mr.K Krishna Singh** for his exemplary guidance, monitoring, and constant encouragement to us throughout the B.Tech course. We shall always cherish the time spent with him during the course of this work due to the invaluable knowledge gained in the field of reliability engineering.

We are extremely grateful for the confidence bestowed in us and entrusting our project entitled "**Android App for Malaria Detection with Microscopic Blood Image Using Deep Learning Model** ".

We express gratitude to **Dr. D.V.Nagarjana Devi** (HOD of CSE) and other faculty members for being a source of inspiration and constant encouragement which helped us in completing the project successfully.

Our sincere thanks to all the batch mates of **2019 CSE**, who have made our stay at **RGUKT-NUZVID**, a memorable one.

Finally, yet importantly, we would like to express our heartfelt thanks to our beloved God and parents for their blessings, our friends for their help and wishes for the successful completion of this project.

# ABSTRACT

Malaria is a deadly syndrome caused by the Plasmodium parasite, transmitted through the bite of infected Anopheles mosquitoes. Despite the availability of several effective drugs, malaria diagnosis remains a challenging, time-consuming, and costly process. Current diagnostic equipment, although highly accurate, tends to be expensive, limiting their accessibility in resource-constrained regions. To address these challenges, this project focuses on the development of a deep learning-based classification model for malaria detection, which can be integrated into an Android application

The proposed solution leverages convolutional neural networks (CNNs) trained on blood smear images to identify the presence of malaria parasites. By utilizing smartphone cameras to capture blood smear images, the application provides a cost-effective and portable diagnostic tool. The deep learning model processes these images to detect and classify malaria, offering a reliable alternative to traditional diagnostic methods.

The initial results of the proposed model demonstrate an accuracy of 82 %, indicating its potential for practical use in real-world scenarios. The Android application is designed to perform real-time image analysis, providing instant classification results. Additionally, it features an intuitive user interface, ensuring ease of use for healthcare providers and field workers. The application also includes functionalities for data sharing and patient management, facilitating integration with existing health information systems.

# Contents

# 1 Introduction

Malaria remains a pervasive and life-threatening disease, primarily affecting tropical and subtropical regions. It is caused by the Plasmodium parasite, which is transmitted to humans through the bites of infected Anopheles mosquitoes. Despite significant advancements in medical treatments and preventive measures, malaria continues to pose a substantial public health challenge, particularly in resource-limited settings. Rapid and accurate diagnosis is essential for effective treatment and control of the disease. However, existing diagnostic methods, such as microscopic examination of blood smears and rapid diagnostic tests (RDTs), can be time-consuming, costly, and often require specialized equipment and trained personnel.

In many endemic regions, the high cost and limited availability of accurate diagnostic equipment hinder effective disease management and timely treatment. To address these challenges, there is a pressing need for innovative solutions that are both cost-effective and accessible. Advances in deep learning and mobile technology offer a promising avenue for improving malaria diagnosis. Deep learning, a subset of artificial intelligence (AI), has demonstrated remarkable success in various image classification tasks, including medical image analysis. By harnessing the power of deep learning algorithms, it is possible to develop an efficient and reliable diagnostic tool that can be deployed on widely available smartphones.

This project aims to develop a deep learning-based malaria classification model that can be integrated into an Android application. The proposed solution involves training a convolutional neural network (CNN) on a large dataset of blood smear images to accurately detect and classify malaria parasites. The application enables healthcare providers and field workers to capture images of blood smears using smartphone cameras, which are then analyzed by the deep learning model to provide instant diagnostic results. Initial evaluations of the model have shown an accuracy rate of 82%, demonstrating its potential for real-world application.

By democratizing access to advanced diagnostic tools, this Android application aims to enhance malaria detection, particularly in low-resource settings. Future enhancements could involve improving the model's accuracy, expanding its capabilities to classify different malaria strains, and integrating telemedicine features for remote consultation and decision support. This innovation has the potential to significantly improve the efficiency and accessibility of malaria diagnosis, ultimately contributing to better disease management and treatment outcomes

# 2 Related Works

| Title | Authors | Publisher(Year) | Key Concept | Cons |
|---|---|---|---|---|
| Detection of Malaria Parasite Infected Blood Cells Using Image Processing | NU RANI | IEEE (2012) | Utilizing image processing for detecting infected blood cells in microscopic images | Performance degrades with image artifacts and poor quality images |
| A Machine Learning Approach for Automated Diagnosis of Malaria | C Chakraborty | MICRON (2013) | Machine learning algorithms for automating malaria diagnosis using image data | Dependency on large labeled datasets for training |
| Malaria Cell Image Dataset Classification Using Convolutional Neural Network | W.David Pan | PeerJ (2018) | Classification of malaria cell images using CNNs | Dataset imbalance affecting model performance |
| Malaria Detection by Deep Learning Using Mobile Microscopy | M Mittal | Journal of King Saud University-Computer and information Scince (2022) | Mobile microscopy integration with deep learning for malaria detection | Performance affected by variability in mobile image quality |
| Convolutional Neural Network for Malaria Diagnosis and Quantitiation | M Jaiswal | Computational and Mathematical Methods in Medicine (2017) | Application of CNNs for malaria parasite detection in blood smear images | Limited evaluation on diverse datasets |
| Deep Learning and Data Augmentation for Malaria Parasite Detection | W Hoyos | BMC Bioinformatics (2024) | Combining deep learning and data augmentation to enhance malaria parasite detection | Model performance highly dependent on quality of augmented data |
| A Hybrid Deep Learning Model for Malaria Detection Using Microscopic Blood Smears | Mostafa M.Fouda | IEEE Transactions on Biomedical Engineering (2024) | Hybrid model combining CNNs and RNNs for robust malaria detection | High resource requirements for training and inference |

Figure 1: Realted works.

# 3 Procedure

## 3.1 Step-1: Data Loading

Mount Google Drive. Load dataset from the specified directory. Split the dataset into training and testing sets.

## 3.2 Step-2: Data Preprocessing

Rescale images to a standard size (e.g., 224x224 pixels). Apply data augmentation techniques such as rotation, zoom, horizontal flip, etc., to increase dataset variability.

## 3.3 Step-3: Train Model

Fit the model using the training data. Trained model with learning rate 0.001 upto 30 epochs with batch size 32. Implement early stopping to monitor validation loss and stop training when the loss stops improving.

## 3.4 Step-4: Evaluate Model

Generate predictions on the test data. Calculate and print the classification report (including precision, recall, F1-score) and confusion matrix.

## 3.5 Step-5: Plot Training And Validation Metrics

Plot the training and validation accuracy and loss over the epochs.

## 3.6 Step-6: Convert Model to TFlite

Load the trained Keras model. Convert the model to TFLite format using the TensorFlow Lite Converter. Save the TFLite model to a specified directory.

## 3.7 Step-7: Inference on New Images

Load and preprocess new images to match the input requirements of the model. Load the TFLite model. Use the TFLite interpreter to make predictions on the new images. Display the predictions.

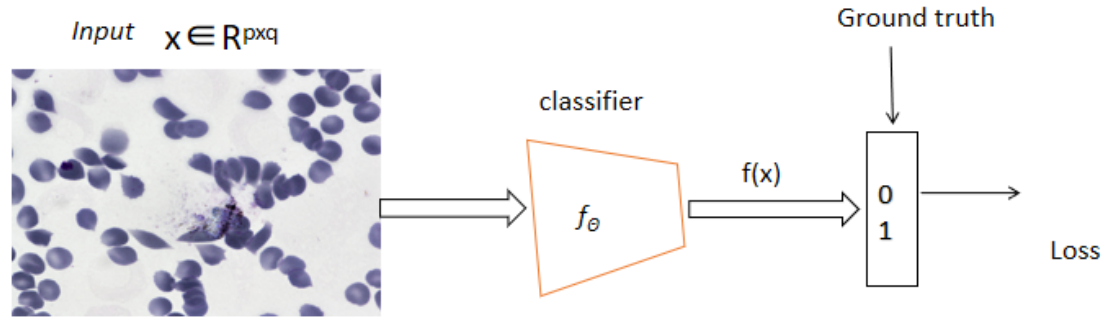
## 3.8 Step-8: Add TFlite Model to Android Studio

Import the TFLite model into an Android Studio project. Integrate the TFLite Interpreter to perform inference.

## 3.9 Step-9: Inference on Mobile App

Load and preprocess new images within the mobile app. Use the TFLite model for predictions. Display predictions on the mobile app.

## 3.10 Model Architecture




$$L(\Theta,x,y)=-\Sigma y_i log\ f(x_i)$$

$$\Theta'=min_\Theta\ E_{(x,y)\in D}\ L(\Theta,x,y)$$

Figure 2: Model Architecture

# 4 Experiments - Datasets Used

## 4.1 Libraries Used

- **TensorFlow and Keras**: Frameworks for building and training neural networks, providing tools for designing both the generator and discriminator models.
- **NumPy, Random, and Math**: Libraries for numerical computations, random number generation, and mathematical functions, essential for data preprocessing and augmentation.
- **Cv2 and Matplotlib**: Libraries for image processing and visualization, used for handling input images and visualizing generated samples and training progress.
- **Sklearn.metrics**:The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance.

# 5 Formulas Used :

## 5.1 LOSS FUNCTION:

$$L(\theta, x, y) = -\sum y \log f(x) \tag{1}$$

# 6 System Hardware

Google Colab TP4 GPU
**RAM Used:** 7.66 GB/12.67 GB
**Disk Used:** 29.10 GB/78.19 GB

# 7 Sample Code

Importing Libraries

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.metrics import classification_report
    import tensorflow as tf
```

Classifing the images into class Label list

```
import os
import csv
import shutil

# Path to the folder
folder_path = "/content/gdrive/MyDrive/malaria1/Label"
source_path= "/content/gdrive/MyDrive/malaria1/images"
dest_folder1="/content/gdrive/MyDrive/img/uninfected"
dest_folder2="/content/gdrive/MyDrive/img/infected"

# Function to read values from the first column of text files
def read_first_column(file_path):
    with open(file_path, 'r') as file:
        first_column_values=[]
        reader = csv.reader(file, delimiter=' ')  # Assuming tab-delimited
        for row in reader:
            cleaned_list = [item.strip() for item in row if isinstance(item, str) and item.str
            first_column_values.append(cleaned_list[0])
    # with open(file_path, 'r') as file:
    #     reader = csv.reader(file, delimiter=' ')  # Assuming tab-delimited
    #     first_column_values = [row[2] for row in reader]
    return first_column_values
```

```python
# Function to categorize files
def categorize_files(folder_path):
    uninfected = []
    infected = {}
    # List all files in the folder
    files = os.listdir(folder_path)
    # Iterate through each file
    for file_name in files:
        if file_name.endswith('.txt'):  # Assuming all files are text files
            file_path = os.path.join(folder_path, file_name)
            first_column_values = read_first_column(file_path)
            if all(value == "0" for value in first_column_values):
                uninfected.append(file_name)
            else:
                count = sum(1 for value in first_column_values if value != "0")
                infected[file_name] = count

    return uninfected, infected

# Call the function to categorize files
uninfected_files, infected_files = categorize_files(folder_path)
```

## Counting Number of cells infected

```python
c=0
infected1=[]
for file_name, count in infected_files.items():
  if(count>3):
    c=c+1
    infected1.append(file_name)
    # print(f"{file_name}: {count}")
print(c)
```

```python
import os
import shutil

def separate_images(txt_list1, txt_list2, source_folder, dest_folder1, dest_folder2):
    # Create destination folders if they don't exist
    os.makedirs(dest_folder1, exist_ok=True)
    os.makedirs(dest_folder2, exist_ok=True)

    # Iterate over each text file in the first list
    for txt_file in txt_list1:
        # Construct the corresponding image file name
        img_file = os.path.splitext(txt_file)[0] + ".png"
        img_path = os.path.join(source_folder, img_file)
        # Check if the image file exists
        if os.path.exists(img_path):
            shutil.copy(img_path, dest_folder1)

    # Iterate over each text file in the second list
    for txt_file in txt_list2:
        # Construct the corresponding image file name
        img_file = os.path.splitext(txt_file)[0] + ".png"
        img_path = os.path.join(source_folder, img_file)
        # Check if the image file exists
        if os.path.exists(img_path):
            shutil.copy(img_path, dest_folder2)

# Example usage:


source_path= "/content/gdrive/MyDrive/malaria1/images"
dest_folder1="/content/gdrive/MyDrive/img3/uninfected"
dest_folder2="/content/gdrive/MyDrive/img3/infected"


separate_images(uninfected_files, infected1, source_path, dest_folder1, dest_folder2)
```

## Train Data Splitting

```python
[ ] import os
    import shutil
    from sklearn.model_selection import train_test_split

    # Define the source and destination directories
    source_dir = '/content/gdrive/MyDrive/img3/'
    train_dir = '/content/gdrive/MyDrive/dataset2/train/'
    test_dir = '/content/gdrive/MyDrive/dataset2/test/'

    # Create train and test directories
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    # Create subdirectories for each class in the train and test directories
    class_names = os.listdir(source_dir)
    for class_name in class_names:
        os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
        os.makedirs(os.path.join(test_dir, class_name), exist_ok=True)

    # Function to split and save images
```

```python
# Function to split and save images
def split_and_save_images(source_dir, train_dir, test_dir, class_names, test_size=0.2, seed=111):
    for class_name in class_names:
        class_path = os.path.join(source_dir, class_name)
        images = os.listdir(class_path)

        train_images, test_images = train_test_split(images, test_size=test_size, random_state=seed)

        # Copy images to respective directories
        for image in train_images:
            shutil.copy(os.path.join(class_path, image), os.path.join(train_dir, class_name, image))

        for image in test_images:
            shutil.copy(os.path.join(class_path, image), os.path.join(test_dir, class_name, image))

# Split and save the images
split_and_save_images(source_dir, train_dir, test_dir, class_names)

print("Images split and saved successfully.")
```

## Model

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, accuracy_score  # Import accuracy_score here

# Data augmentation for training set
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Simple rescaling for validation/test set
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

```python
# Directory paths
train_dir = '/content/gdrive/MyDrive/dataset2/train'
test_dir = '/content/gdrive/MyDrive/dataset2/test'

# Training and test generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```python
# Model architecture (example)
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
```

```python
# Train the model
history = model.fit(
    train_generator,
    epochs=30,
    validation_data=test_generator,
    callbacks=[early_stopping]
)

# Save the model
model.save('/content/malaria_class_augmented.h5')
# Evaluate on test data
predictions = model.predict(test_generator, steps=np.ceil(test_generator.n / test_generator.batch_size))
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)

accuracy = accuracy_score(true_classes, predicted_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
model.summary()
```

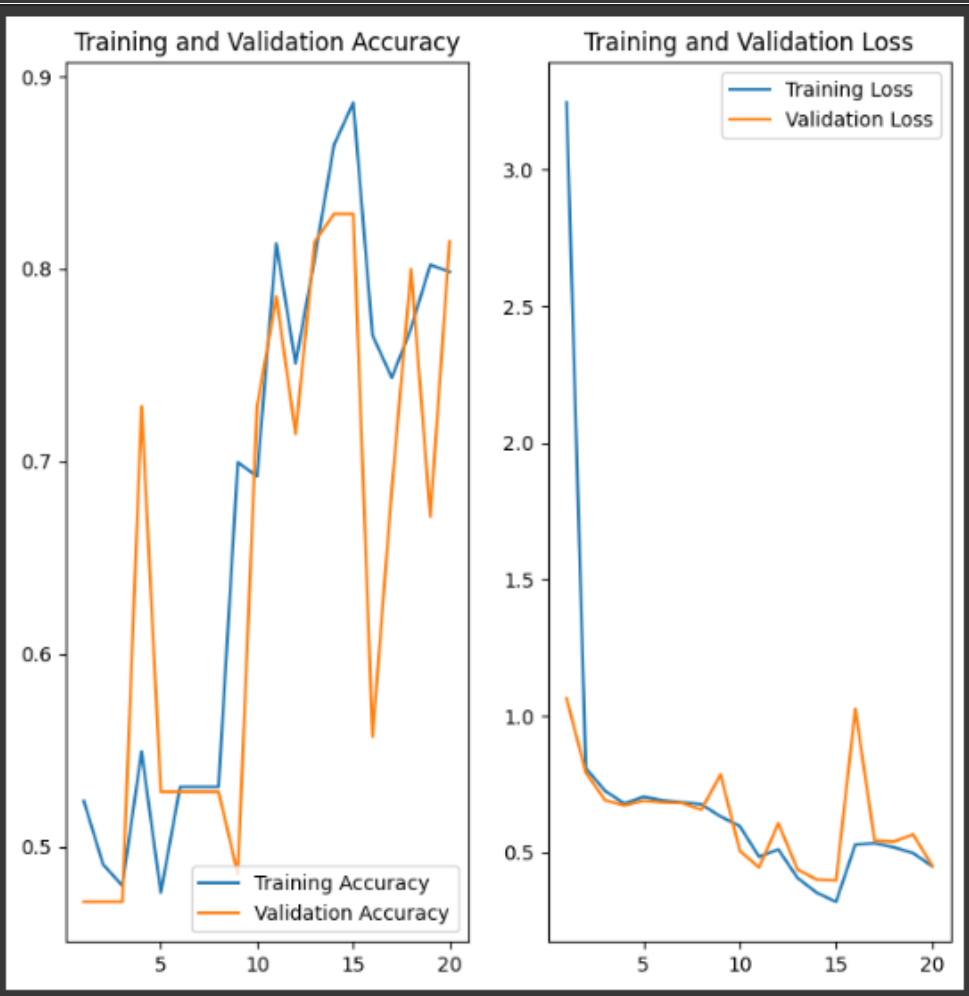
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_3 (MaxPoolin  (None, 111, 111, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_4 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 86528)             0

 dense_2 (Dense)             (None, 512)               44302848

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 2)                 1026

=================================================================
Total params: 44397122 (169.36 MB)
Trainable params: 44397122 (169.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## Plotting

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(1, len(history.history['accuracy']) + 1)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

## Plotting

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(1, len(history.history['accuracy']) + 1)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



## Converting to tflite

```
[ ]  import tensorflow as tf

     # Load the Keras model from .h5 file
     model = tf.keras.models.load_model("malaria_class_augmented.h5")

     # Convert the model to TensorFlow Lite format
     converter = tf.lite.TFLiteConverter.from_keras_model(model)
     tflite_model = converter.convert()

     # Save the TensorFlow Lite model to a .tflite file
     with open("malaria_classification82.tflite", "wb") as f:
         f.write(tflite_model)
```

## Label prediction on test data

```
[ ]  import tensorflow as tf
     import numpy as np
     from PIL import Image
     import os
     import matplotlib.pyplot as plt

     # Load the TFLite model
     interpreter = tf.lite.Interpreter(model_path="/content/malaria_classification82.tflite")
     interpreter.allocate_tensors()

     # Get input and output details
     input_details = interpreter.get_input_details()
     output_details = interpreter.get_output_details()

     # Get input shape
     input_shape = input_details[0]['shape']
     IMG_HEIGHT, IMG_WIDTH = input_shape[1], input_shape[2]

     def preprocess_image(image_path):
         # Load image
         img = Image.open(image_path).convert('RGB')
         # Resize image
         img = img.resize((IMG_WIDTH, IMG_HEIGHT))
         # Convert image to numpy array
         img = np.array(img).astype(np.float32)  # Ensure the data type is float32
         # Normalize the image to [0, 1]
         img = img / 255.0
         # Add batch dimension
         img = np.expand_dims(img, axis=0)
         return img
```

```python
def predict_image(image_path):
    # Preprocess the image
    input_data = preprocess_image(image_path)

    # Set the tensor to point to the input data to be inferred
    interpreter.set_tensor(input_details[0]['index'], input_data)

    # Run the inference
    interpreter.invoke()

    # Get the results
    output_data = interpreter.get_tensor(output_details[0]['index'])
    return output_data
```

```python
def predict_images_in_folder(folder_path):
    image_paths = [os.path.join(folder_path, filename) for filename in os.listdir(folder_path)
                   if filename.endswith(".jpg") or filename.endswith(".png")]

    # Define the number of images per row
    images_per_row = 3

    # Create subplots
    fig, axes = plt.subplots(nrows=len(image_paths) // images_per_row + 1,
                             ncols=images_per_row, figsize=(15, 15))

    for idx, image_path in enumerate(image_paths):
        predictions = predict_image(image_path)
        predicted_class = class_labels[np.argmax(predictions)]

        # Load the original image for visualization
        original_img = Image.open(image_path).convert('RGB')

        # Get the appropriate subplot
        ax = axes[idx // images_per_row, idx % images_per_row]

        # Display the image with the predicted label
        ax.imshow(original_img)
        ax.set_title(f"Predicted: {predicted_class}")
        ax.axis('off')  # Hide axes

    # Hide any remaining empty subplots
    for j in range(len(image_paths), len(axes.flat)):
        axes.flat[j].axis('off')

    plt.tight_layout()
    plt.show()

# Example usage
predict_images_in_folder('/content/gdrive/MyDrive/td')
```

# 8 Output

### Prediction Output

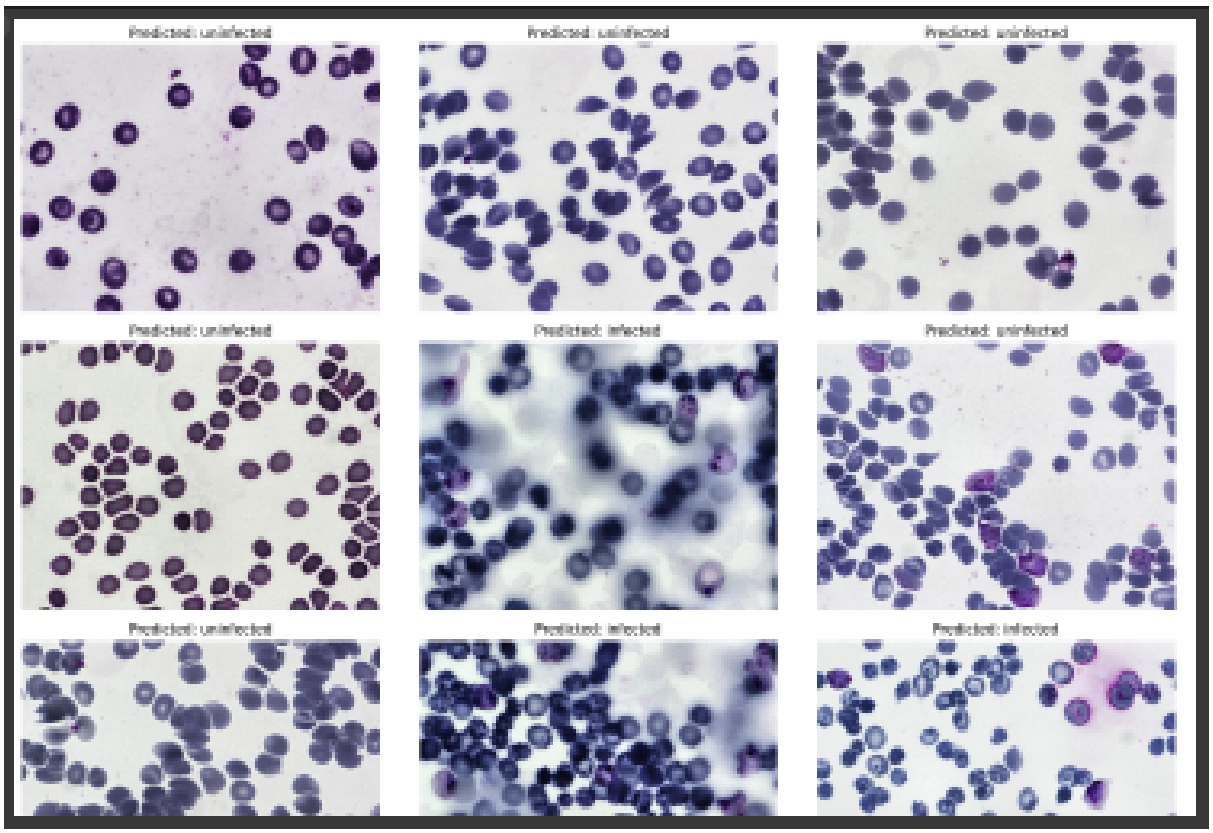



Figure 3: Prediction output.

# 9 Android app

### coding

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.cv = findViewById(R.id.CV);
    this.tv = findViewById(R.id.tv);
    this.cv.setLifecycleOwner(this);
    imv = findViewById(R.id.imageCaptured);
    this.cv.addCameraListener(new CameraListener() {
        @Override
        public void onPictureTaken(PictureResult result) {
            byte[] data = result.getData();
            resized = Bitmap.createScaledBitmap(BitmapFactory.decodeByteArray(data, offset: 0, data.length), inputSize, inputSize, filter: true);
            imv.setImageBitmap(resized);
        }
        @Override
        public void onVideoTaken(VideoResult result) { }
        @Override
        public void onVideoRecordingStart() { }
        @Override
        public void onVideoRecordingEnd() { }
        @Override
        public void onZoomChanged(float newValue, float[] bounds, PointF[] fingers) { }
        @Override
        public void onOrientationChanged(int orientation) { }
    });
```

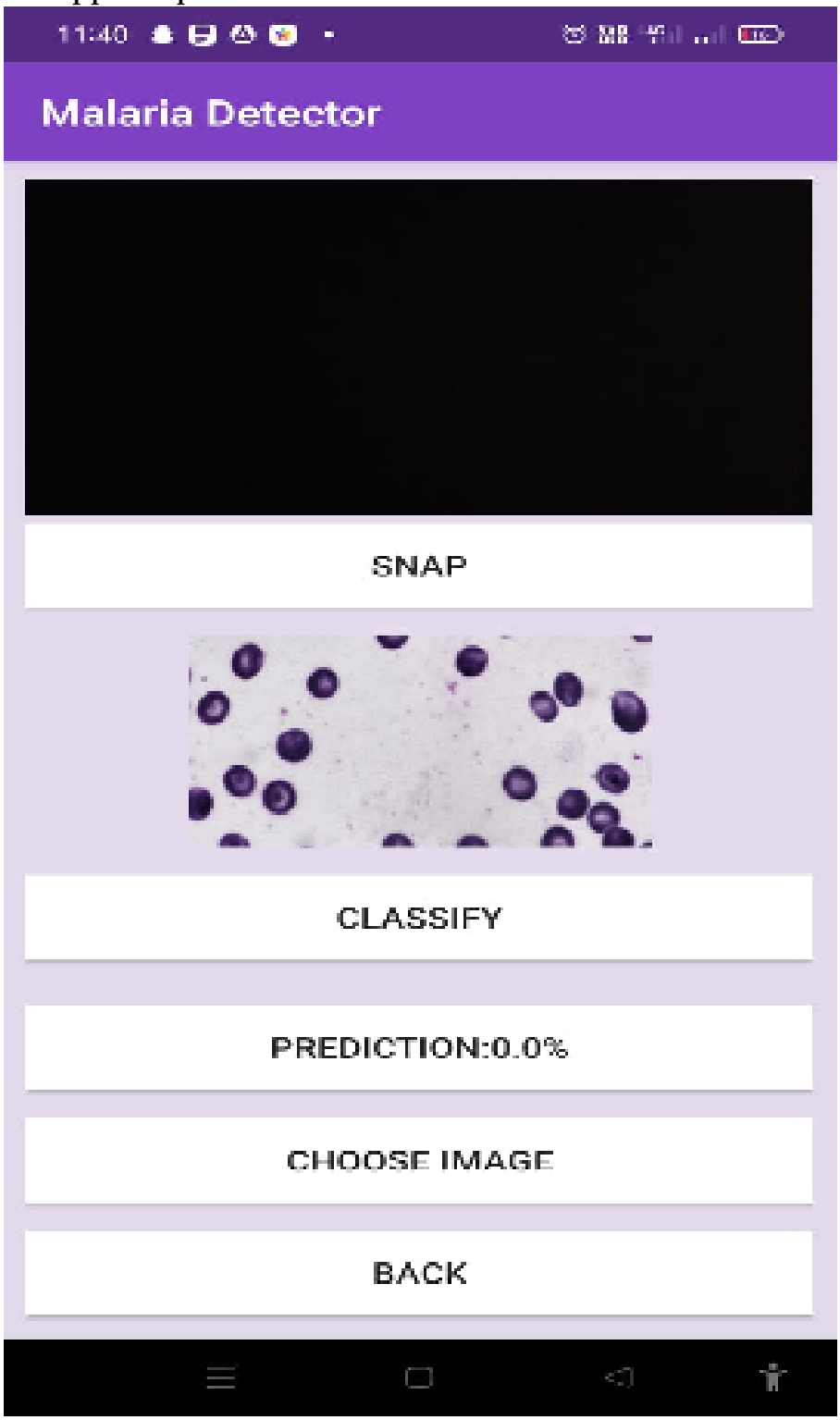
```
    try {
        MappedByteBuffer tfliteModel = FileUtil.loadMappedFile( context: this,  filePath: "malaria_classification82.tflite");
        tfLite = new InterpreterFactory().create(tfliteModel, new InterpreterApi.Options());
    } catch (IOException e) {
        Log.e( tag: "tfliteSupport",  msg: "Error reading model", e);
    }
    buttonChoose = findViewById(R.id.buttonChoose);
    buttonChoose.setOnClickListener((v) -> { openGallery(); });
    buttonBack = findViewById(R.id.buttonBack);
    buttonBack.setOnClickListener((v) -> { resetState(); });
}
public void take_photo(View v) {
    toast = Toast.makeText( context: this,  text: "snapping...", Toast.LENGTH_SHORT);
    toast.setGravity( gravity: 17,  xOffset: 0,  yOffset: 0);
    toast.show();
    this.cv.takePicture();
}
private void openGallery() {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent,  title: "Select Picture"), PICK_IMAGE_REQUEST);
}
```

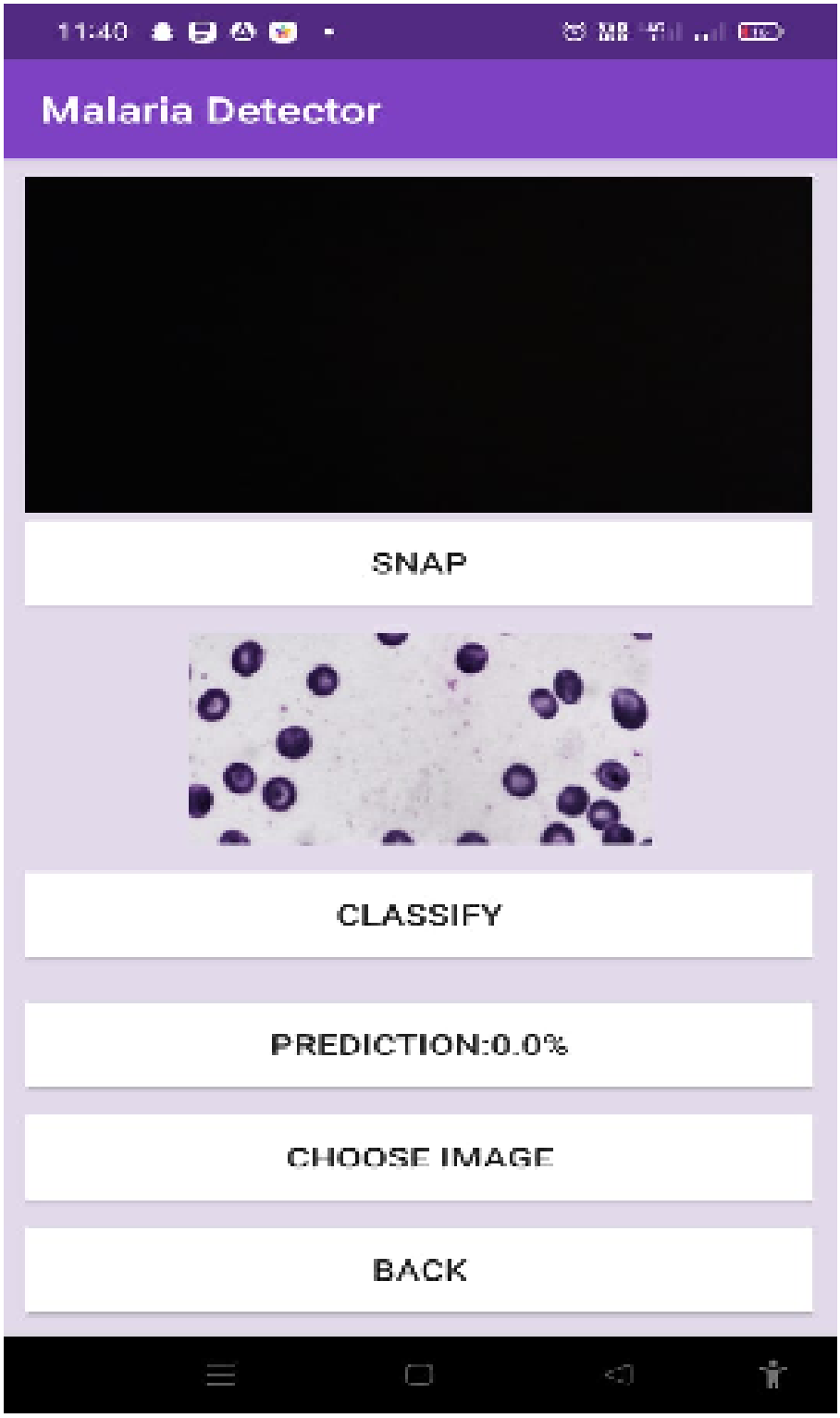
```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == PICK_IMAGE_REQUEST && resultCode == RESULT_OK && data != null && data.getData() != null) {
        Uri uri = data.getData();
        try {
            Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
            resizedBitmap = Bitmap.createScaledBitmap(bitmap, inputSize, inputSize,  filter: true);
            imv.setImageBitmap(resizedBitmap);
        } catch (Exception e) {
            e.printStackTrace();
            Toast.makeText( context: this,  text: "Failed to load image", Toast.LENGTH_SHORT).show();
        }
    }
}
```

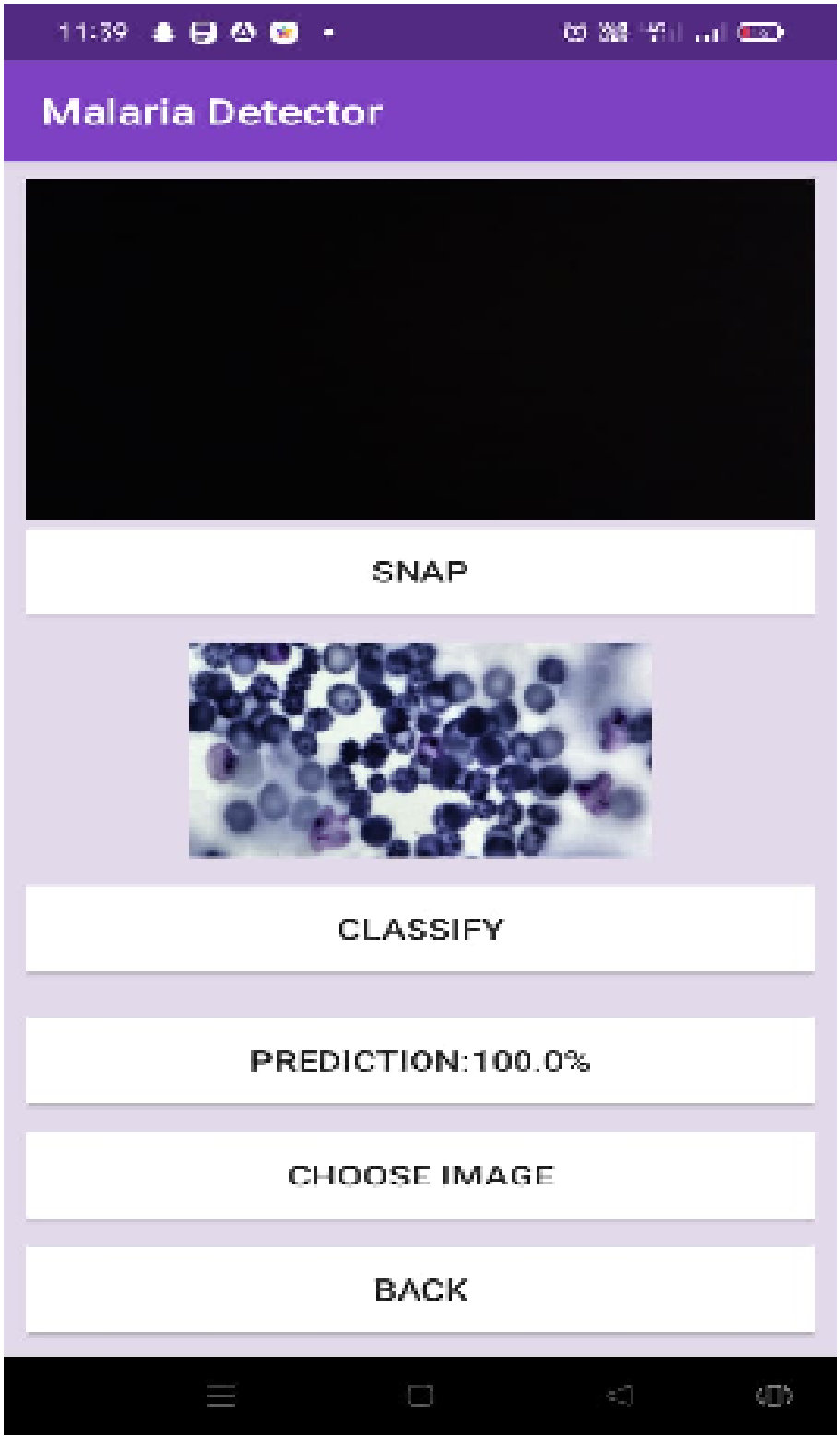
```
public void classify(View V) {
    if (resizedBitmap == null && resized == null) {
        Toast.makeText( context: this,  text: "No image to classify", Toast.LENGTH_SHORT).show();
        return;
    }
    Bitmap bitmapToClassify = resizedBitmap != null ? resizedBitmap : resized;

    TensorImage inTensorImage = new TensorImage(DataType.FLOAT32);
    inTensorImage.load(bitmapToClassify);

    ImageProcessor imageProcessor = new ImageProcessor.Builder()
            .add(new ResizeOp(inputSize, inputSize, ResizeOp.ResizeMethod.BILINEAR))
            .add(new NormalizeOp( mean: 0.0f,  stddev: 1.0f))  // Normalize according to your model's requirement
            .build();
    inTensorImage = imageProcessor.process(inTensorImage);
    int[] outShape = tfLite.getOutputTensor( i: 0).shape();
    DataType outDataType = tfLite.getOutputTensor( i: 0).dataType();
    TensorBuffer outTensorBuffer = TensorBuffer.createFixedSize(outShape, outDataType);
    tfLite.run(inTensorImage.getBuffer(), outTensorBuffer.getBuffer());
    float[] outputValues = outTensorBuffer.getFloatArray();
    String st = "";
    if (outputValues[0] < 0.5) {
        st = "Not infected";
    } else {
        st = "Infected";
    }
    tv.setText("Prediction is "+st+"-"+ outputValues[0] * 100 + "%");
}
```

```java
private void resetState() {
    // Reset image views and any other UI elements
    imv.setImageBitmap(null);
    tv.setText("Result");
    // Reset the bitmap variables
    resizedBitmap = null;
    resized = null;
    Toast.makeText( context: this,  text: "Ready for new prediction", Toast.LENGTH_SHORT).show();
}
```

**App Output:**


11:40
Malaria Detector
SNAP
CLASSIFY
PREDICTION:0.0%
CHOOSE IMAGE
BACK


11:40
Malaria Detector
SNAP
CLASSIFY
PREDICTION:0.0%
CHOOSE IMAGE
BACK


11:39
Malaria Detector
SNAP
CLASSIFY
PREDICTION:100.0%
CHOOSE IMAGE
BACK

# 10 Conclusion

- The development of a deep learning-based classification model for malaria detection represents a significant advancement in addressing the limitations of traditional diagnostic methods. The integration of this model into an Android application offers a cost-effective, portable, and accessible solution for malaria diagnosis, particularly in resource-constrained regions. The initial accuracy of 82% highlights the model's potential for practical application, while the real-time image analysis and intuitive user interface ensure usability for healthcare providers and field workers.

- This innovative approach not only democratizes access to advanced diagnostic tools but also enhances the efficiency of malaria detection. Future enhancements, including improving model accuracy, classifying different malaria strains, and incorporating telemedicine features, could further elevate the application's impact on disease management and treatment outcomes. Ultimately, this project aims to contribute to better healthcare delivery and control of malaria, particularly in underserved areas, by leveraging the power of deep learning and mobile technology.

# 11 Future Enhancements

## Real-world Deployment and Integration:

- **Mobile App Development:** Develop mobile applications that integrate the TFLite model, providing healthcare workers with a user-friendly tool for on-site malaria diagnosis.
- **Integration with Healthcare Systems:** Collaborate with healthcare providers to integrate the diagnostic tool into existing medical systems, ensuring seamless and widespread adoption.
- **Model Accuracy Improvement:** Enhance the deep learning model to achieve higher accuracy through advanced techniques such as transfer learning, and hyper-parameter tuning
- **Classification of Different Malaria Strains:** Expand the model's capabilities to classify different strains of malaria parasites, providing more detailed diagnostic information

# References


[1] Ghate, D. A., Jadhav, C., Rani, N. U. (2012). Automatic detection of malaria parasite from blood images. Int J Comput Sci Appl, 1

[2] Das, Dev Kumar, et al. "Machine learning approach for automated screening of malaria parasite using light microscopic images." Micron 45 (2013): 97-106.

[3] Pan, W. David, Yuhang Dong, and Dongsheng Wu. "Classification of malaria-infected cells using deep convolutional neural networks." Machine learning: advanced techniques and emerging applications 159 (2018).

[4] Pattanaik, Priyadarshini Adyasha, et al. "Malaria detection using deep residual networks with mobile microscopy." Journal of King Saud University-Computer and Information Sciences 34.5 (2022): 1700-1705.

[5] Mehanian, Courosh, et al. "Computer-automated malaria diagnosis and quantitation using convolutional neural networks." Proceedings of the IEEE international conference on computer vision workshops. 2017.

[6] Hoyos, Kenia, and William Hoyos. "Supporting Malaria Diagnosis Using Deep Learning and Data Augmentation." Diagnostics 14.7 (2024): 690.

[7] Dev, Antora, et al. "Advancing Malaria Identification from Microscopic Blood Smears Using Hybrid Deep Learning Frameworks." IEEE Access (2024).