PROJECT REPORT

Deep Learning Lab (AIL-401)

PROJECT TITLE: MALARIA DETECTION USING DEEP CONVOLUTIONAL NETWORKS

BS(AI)-06

Group Members

Name	Enrollment
1. SYED MUHAMMAD YAMANN	02-136212-001
2. MUNEEZA IFTIKHAR	02-136212-012
3. HAFSA HAFEEZ SIDDIQUI	02-136212-026

Submitted to:

Ms Munaza Azhar

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

Abstract

Malaria continues to be a significant global health issue, particularly in low-resource settings where timely and accurate diagnostics are often inaccessible. This report presents a deep learning-driven approach to malaria detection, leveraging deep convolutional neural networks (CNNs) to enhance diagnostic precision and scalability. The proposed framework integrates two models: a binary classifier to detect malaria infections and a multiclass classifier to identify specific Plasmodium species subtypes. Using public datasets and advanced techniques such as data augmentation and transfer learning, the system achieves 96% accuracy in binary classification and 89% accuracy in multiclass classification. Moreover, an intuitive graphical user interface (GUI) implemented using Gradio ensures seamless usability. This innovative solution has the potential to transform malaria diagnostics, making them more efficient and accessible worldwide.

Contents

1.INTRODUCTION

Malaria remains a pervasive global health challenge, disproportionately affecting populations in low-resource settings and claiming hundreds of thousands of lives each year. Central to the battle against this parasitic disease is the need for prompt, accurate, and widely accessible diagnostic methods. Although microscopy-based examination of stained blood smears has long served as the diagnostic gold standard, this approach is labor-intensive, time-consuming, and subject to human error. In regions with limited healthcare infrastructure and constrained availability of trained personnel, these factors frequently impede timely, reliable diagnoses and, in turn, the delivery of effective treatment.

Advances in deep learning and computer vision, particularly deep convolutional neural networks (CNNs), have begun to reshape the landscape of disease diagnostics. By analyzing microscopic blood smear images, CNNs can identify morphological patterns indicative of Plasmodium infections—signatures that may be too subtle or variable for human observers to consistently detect. Leveraging large, annotated datasets, deep learning models can streamline diagnostic workflows, potentially improving accuracy, reducing human workload, and increasing access to timely care. This report introduces a deep CNN-based malaria detection system that seeks to enhance the speed, consistency, and scalability of malaria diagnosis.

2. PROBLEM STATEMENT

Current malaria diagnosis methods rely heavily on manual microscopy, which is difficult to scale and often prone to interpretative errors, especially in low-resource settings. These limitations can delay the initiation of treatment, increase the risk of transmission, and ultimately worsen patient outcomes. To address this issue, we propose a deep convolutional neural network-based framework that not only detects malaria-infected cells from microscopic images but also classifies the specific Plasmodium species subtype. By integrating binary and multiclass classification models into a single system, we aim to improve diagnostic accuracy, expedite patient care, and support more effective malaria control efforts.

3. METHODOLOGY

The project is divided into two modules: the first detects whether malaria is present through binary classification, and the second classifies the specific type of malaria among the 14 subtypes in the dataset through multiclass classification.

3.1 Binary Classification

The primary goal is to train a model that classifies whether a given input image belongs to the positive or negative class. This section utilizes the publicly available "malaria" dataset from TensorFlow Datasets (TFDS), which contains labeled images of parasitized and uninfected blood cells

3.1.1 Data Preprocessing

- Images are resized to (128, 128) pixels.
- Pixel values are normalized by dividing by 255.0, scaling them to the range [0, 1].
- Labels are cast to float32 for compatibility with binary classification.
- Dataset is split into 80-20 ratio for training and validation
- Training data is shuffled with buffer size of 1000 to randomize the order and are batched for efficient processing.

3.1.2 Model Architecture

The model employs a deep CNN structure for progressive feature extraction, employing the following layers and operations:

- **Conv2D Layers:** Multiple convolutional layers progressively extract spatial features from the input images, beginning with filters of size 32 and increasing to 64, 128, and 256. ReLU activation functions are used throughout, introducing non-linearity and enhancing representational power.
- **Batch Normalization:** Batch normalization layers stabilize and accelerate the training process by normalizing intermediate activations, thereby improving model convergence.
- **MaxPooling2D:** Max pooling layers down sample the feature maps, reducing their spatial dimensions and computational complexity while retaining critical information.
- **GlobalAveragePooling2D:** A global average pooling layer compresses the extracted features into a compact, fixed-size vector, serving as the final feature representation before classification.

3.1.3 Saving The model

The trained model is saved as **deep_binary_classification_model.h5** which will be later utilized for the GUI implementation.

3.2 Multiclass Classification

It utilizes transfer learning for multiclass classification using the **MobileNetV2** architecture as a base model and the **MalariaSD dataset**, which encompasses multiple categories of malaria infections (including various Plasmodium species such as P. falciparum, P. vivax, and others), enabling the model to accurately differentiate between these distinct subtypes.

3.2.1 Data Augmentation

Image augmentation was performed using **ImageDataGenerator** with transformations including pixel value rescaling, rotations (up to 20°), shifts (up to 20% in width and height), shearing, zooming (up to 20%), and horizontal flipping. This approach increases dataset variability and improves the model's generalization capability.

3.2.2 Model Architecture

- **Base Model (MobileNetV2):**
  - Pre-trained on ImageNet.
  - include_top=False to remove the default classification layers.
  - trainable=False to keep pre-trained weights fixed.
  - Provides high-quality feature maps for subsequent layers.
- **Custom Head:**
  - Flatten**:** Converts feature maps into a 1D vector.
  - Dense (256 neurons, ReLU): Adds learnable parameters for higher-level feature abstraction.
  - Dropout (rate=0.5**):** Reduces overfitting by randomly dropping units.
  - Output Layer (Softmax): Outputs class probabilities, with the number of neurons equal to the number of subtypes.

3.2.3 Saving the model

The trained model is saved as **transfer_learning_model.h5** which will be later utilized for the GUI implementation.

3.3 GUI Implementation

- The GUI was implemented using **Gradio**, a Python library that simplifies the creation of interactive web interfaces for machine learning models.
- Two pre-trained deep learning models are integrated:
  - A binary classifier to determine if the image is infected or uninfected.
  - A multiclass classifier to identify the specific malaria subtype if infection is present.
- Users can upload an image, which is then preprocessed and passed through the binary model.
- If classified as parasitized, the subtype is predicted using the multiclass model.

4. CODE

4.1 Binary Classification Code

4.1.1 Data Preprocessing:

```
def preprocess_binary(image, label):
    image = tf.image.resize(image, IMG_SIZE) / 255.0
    label = tf.cast(label, tf.float32)
    return image, label
```

```
[ ] binary_train_ds = train_ds.take(int(len(train_ds) * 0.8)).map(preprocess_binary).shuffle(1000).batch(BATCH_SIZE)
    binary_val_ds = train_ds.skip(int(len(train_ds) * 0.8)).map(preprocess_binary).batch(BATCH_SIZE)

    binary_labels = [label.numpy() for _, label in train_ds.map(preprocess_binary)]
    binary_class_weights = calculate_class_weights(binary_labels)
```

4.1.2 Model Architecture Code

```
binary_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    BatchNormalization(),
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

Show hidden output

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

4.1.3 Training Stage

```
binary_model.fit(
    binary_train_ds,
    validation_data=binary_val_ds,
    epochs=10,
    class_weight=binary_class_weights
)
```

```
[ ] binary_model.save('deep_binary_classification_model.h5')
    print("Deep CNN models trained and saved successfully!")
```

4.2 Multiclass Classification Code

4.2.1 Data Augmentation

```
datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)
```

```
train_gen = datagen.flow_from_directory(
    MULTICLASS_DATASET_PATH,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)
```

```
Found 640 images belonging to 16 classes.
```

```
val_gen = datagen.flow_from_directory(
    MULTICLASS_DATASET_PATH,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)
```

4.2.2 Transfer Learning Model (MobileV2Net)

```
base_model = MobileNetV2(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3), include_top=False, weights='imagenet')
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# Custom Model Head
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(len(train_gen.class_indices), activation='softmax')
])
```

4.3 Testing Code

```
# Define paths
BINARY_MODEL_PATH = '/content/drive/MyDrive/deep_binary_classification_model.h5'
MULTICLASS_MODEL_PATH = '/content/drive/MyDrive/transfer_learning_model.h5'

# Load models
IMG_SIZE = (128, 128)
binary_model = tf.keras.models.load_model(BINARY_MODEL_PATH)
multiclass_model = tf.keras.models.load_model(MULTICLASS_MODEL_PATH)

multiclass_labels = [
    "falciparum_gametocyte", "falciparum_ring", "falciparum_schizont", "falciparum_trophozoite",
    "malariae_gametocyte", "malariae_ring", "malariae_schizont", "malariae_trophozoite",
    "ovale_gametocyte", "ovale_ring", "ovale_schizont", "ovale_trophozoite",
    "vivax_gametocyte", "vivax_ring", "vivax_schizont", "vivax_trophozoite"
]
```

```
def preprocess_image(image):
    img = load_img(image, target_size=IMG_SIZE)
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

def predict_image(image):
    img_array = preprocess_image(image)
    binary_prediction = binary_model.predict(img_array)
    if binary_prediction[0][0] > 0.5:
        binary_result = "Uninfected"
        binary_confidence = binary_prediction[0][0]
        return binary_result, None, binary_confidence
    else:
        binary_result = "Parasitized"
        binary_confidence = 1 - binary_prediction[0][0]
        multiclass_prediction = multiclass_model.predict(img_array)
        predicted_class = np.argmax(multiclass_prediction, axis=1)[0]
        confidence = multiclass_prediction[0][predicted_class]
        subclass = multiclass_labels[predicted_class]
        return binary_result, subclass, confidence
```

4.4 GUI through Gradio

```
def interface_predict(image):
    binary_result, subclass, confidence = predict_image(image)

    if binary_result == "Uninfected":
        result = f"Prediction: {binary_result}"
        subclass_info = "No subtype applicable."
    else:
        result = f"Prediction: {binary_result}"
        subclass_info = f"Subtype: {subclass}"

    return result, subclass_info
```

```
# Define Gradio Blocks
with gr.Blocks() as demo:
    gr.Markdown("# Malaria Detection System")
    gr.Markdown("Upload an image to determine if it shows a malaria infection and identify the subtype if applicable.")

    with gr.Row():
        with gr.Column():
            image_input = gr.Image(type="filepath", label="Upload Image")
            predict_button = gr.Button("Predict")
        with gr.Column():
            prediction_output = gr.Textbox(label="Prediction", lines=2)
            subclass_output = gr.Textbox(label="Subtype (if infected)", lines=2)

    predict_button.click(
        fn=interface_predict,
        inputs=image_input,
        outputs=[prediction_output, subclass_output]
    )


    gr.Markdown("### Note: Please ensure the image is clear for accurate predictions.")

# Launch the interface
demo.launch()
```

5.OUTPUT

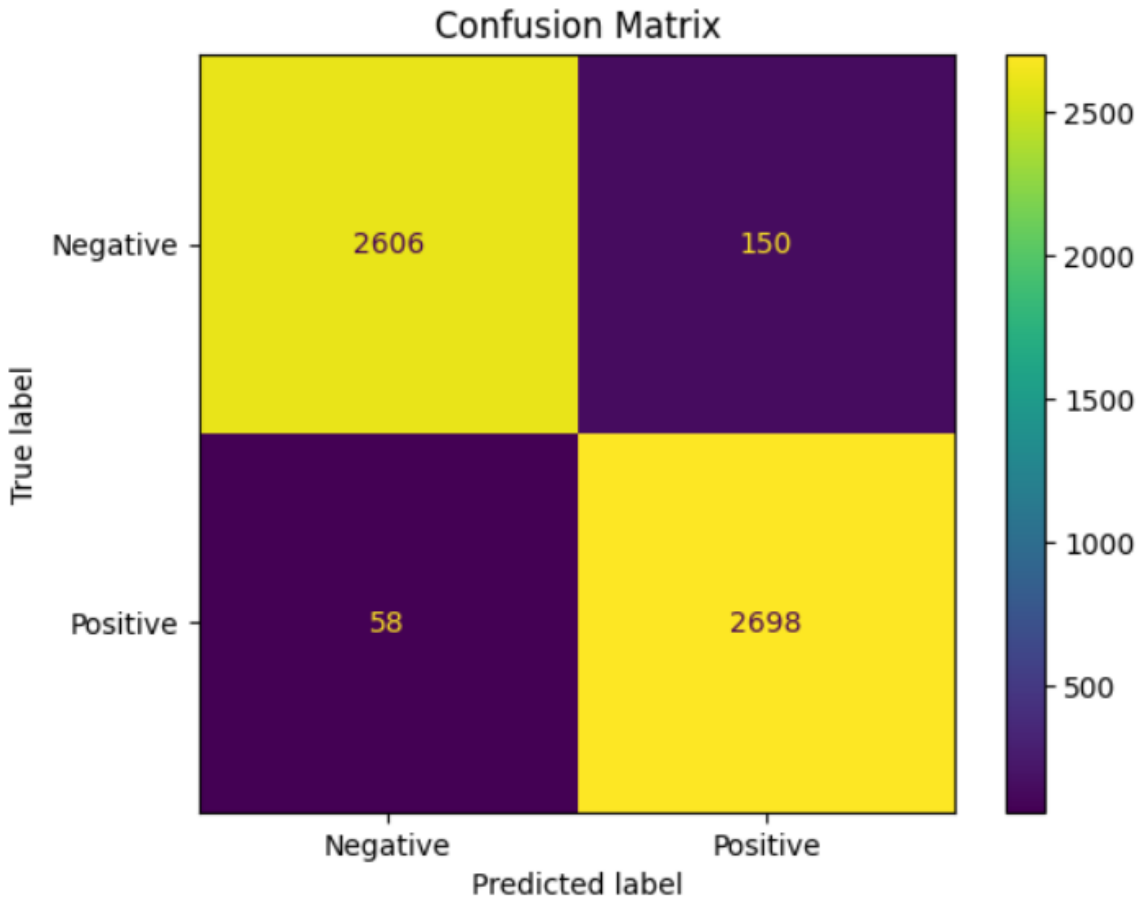
5.1 Accuracy of Binary Classification Model

The binary classification model shows strong performance, achieving a high overall accuracy of 96% and balanced precision and recall for both classes.

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.95	0.96	2756
1	0.95	0.98	0.96	2756
accuracy			0.96	5512
macro avg	0.96	0.96	0.96	5512
weighted avg	0.96	0.96	0.96	5512

The confusion matrix indicates that most samples are correctly identified, with only a small number of misclassifications. This suggests that the model is both sensitive and specific, effectively distinguishing infected cells from uninfected ones.



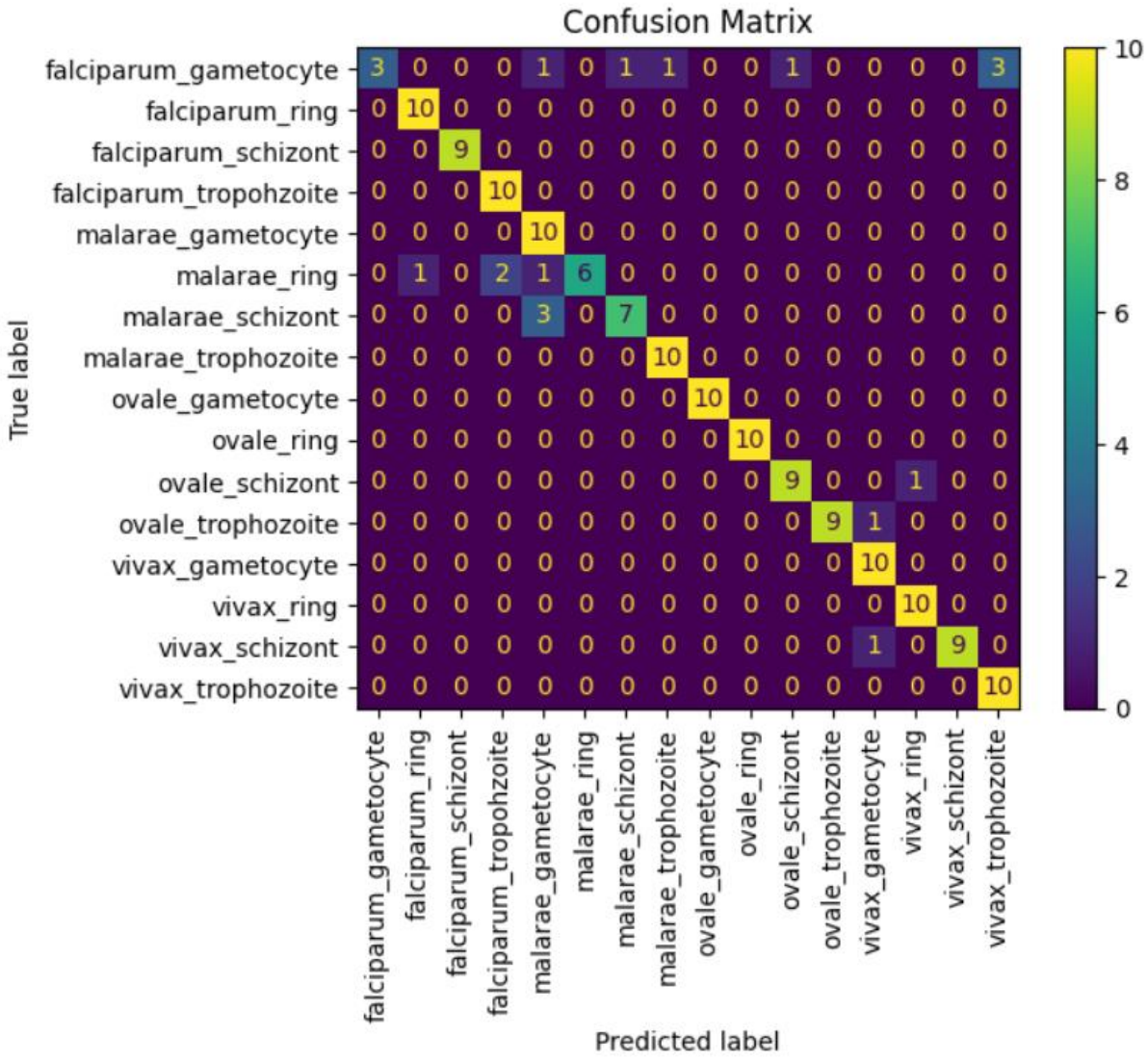
5.2 Accuracy of Multiclass Classification Model

The multiclass model shows generally strong performance, with an overall accuracy of 89% and macro/weighted averages around 91% precision and recall.

Classification Report:

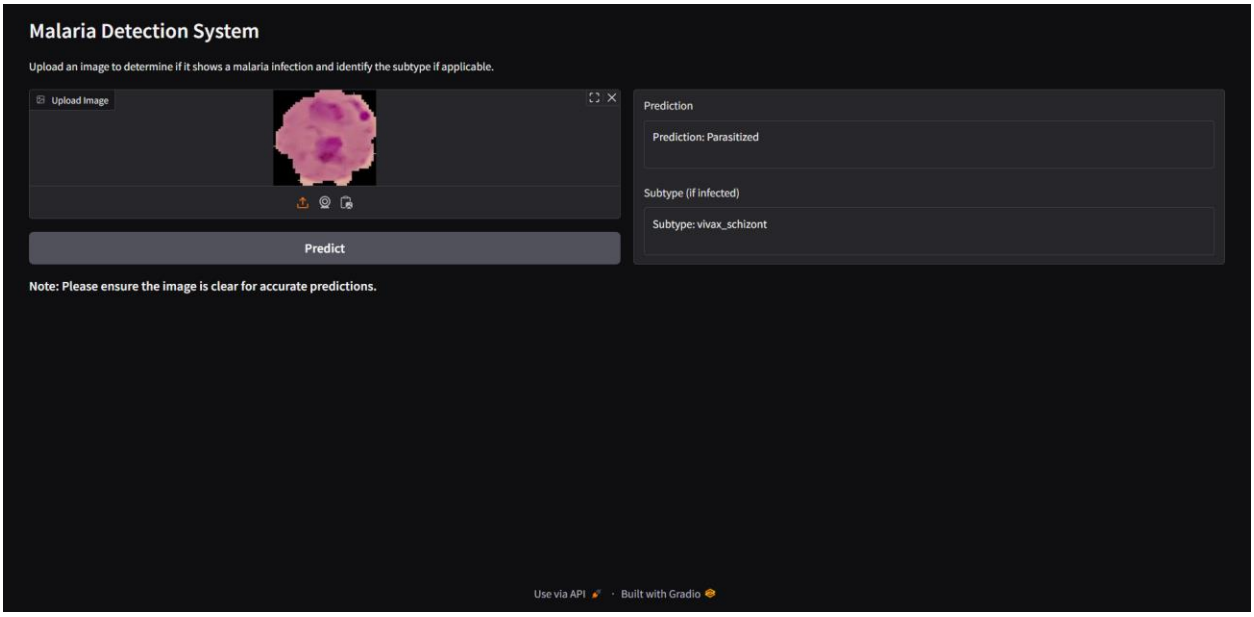
	precision	recall	f1-score	support
falciparum_gametocyte	1.00	0.30	0.46	10
falciparum_ring	0.91	1.00	0.95	10
falciparum_schizont	1.00	1.00	1.00	9
falciparum_tropohzoite	0.83	1.00	0.91	10
malarae_gametocyte	0.67	1.00	0.80	10
malarae_ring	1.00	0.60	0.75	10
malarae_schizont	0.88	0.70	0.78	10
malarae_trophozoite	0.91	1.00	0.95	10
ovale_gametocyte	1.00	1.00	1.00	10
ovale_ring	1.00	1.00	1.00	10
ovale_schizont	0.90	0.90	0.90	10
ovale_trophozoite	1.00	0.90	0.95	10
vivax_gametocyte	0.83	1.00	0.91	10
vivax_ring	0.91	1.00	0.95	10
vivax_schizont	1.00	0.90	0.95	10
vivax_trophozoite	0.77	1.00	0.87	10
accuracy			0.89	159
macro avg	0.91	0.89	0.88	159
weighted avg	0.91	0.89	0.88	159

The confusion matrix reveals that most categories are well-distinguished, though some classes display imbalanced precision and recall values, indicating potential variability in the model's ability to consistently identify certain malaria subtypes.

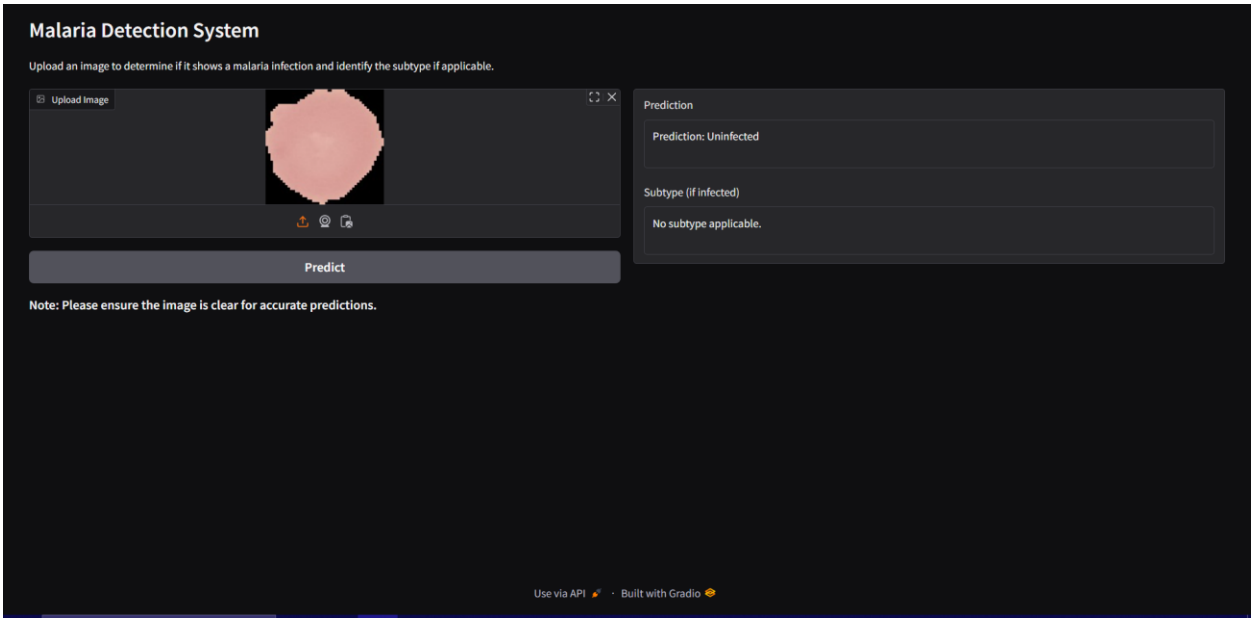


5.3 GUI Interface

It shows the infected image and accurately detects the subtype which in this case is vivax_schizont.



In this case, it accurately predicts uninfected and no subtype is applicable.



6. FUTURE DEVELOPMENT

6.1 Integration with Real-Time Clinical Systems:

- Expand the system for direct integration with hospital and laboratory workflows, ensuring real-time diagnostics.
- Implement automated sample handling to minimize human intervention.

6.2 Dataset Expansion:

- Incorporate more diverse datasets, including images from various geographical regions, to improve generalization across different populations and malaria strains.
- Focus on adding more underrepresented subtypes to balance classification performance.

6.3 Mobile Application Development:

- Develop a mobile application version of the GUI to make the tool accessible in remote and underserved areas.
- Enable offline functionality to cater to regions with limited internet access.

6.4 Hardware Optimization:

- Explore hardware accelerators like edge devices or microcontrollers for deploying the model in resource-constrained environments.
- Optimize model architectures for faster inference on low-power devices.

7. CONCLUSION

This project demonstrates the potential of deep learning in revolutionizing malaria diagnostics. By employing advanced CNN architectures and transfer learning, the system achieves high diagnostic accuracy for both binary and multiclass classifications. The integration of a user-friendly GUI ensures the practical applicability of this solution in diverse settings. While promising results have been achieved, challenges such as dataset diversity and hardware optimization must be addressed to maximize the system's impact. Looking ahead, this framework serves as a robust foundation for not only enhancing malaria detection but also paving the way for AI-powered solutions to address other global health challenges.