

WEB-ENABLED MALARIA DETECTION USING DEEP LEARNING

A PROJECT REPORT

Submitted by

RAJESH R [613521104032]

SHENJIN S A SOLOMON [613521104043]

SANTHOSH KUMAR S [613521104309]

THARUN RAJ R [613521104311]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



GOVERNMENT COLLEGE OF ENGINEERING, DHARMAPURI

ANNA UNIVERSITY :: CHENNAI 600 025

MAY-2025

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “**WEB-ENABLED MALARIA DETECTION USING DEEP LEARNING**” is the bonafide work of the following students, **RAJESH R [613521104032], SHENJIN S A SOLOMON [613521104043], SANTHOSH KUMAR S [613521104309], THARUN RAJ R [613521104311]**, who carried out the project work under my supervision.

SIGNATURE

SIGNATURE

Dr.A.M.KALPANA M.E, Ph.D.,

Mrs.R.NARMATHA M.E.,

HEAD OF THE DEPARTMENT,

SUPERVISOR,

ASSISTANT PROFESSOR,

Department of Computer Science and
Engineering ,

Department of Computer Science and
Engineering ,

Government College of Engineering,
Dharmapuri.

Government College of Engineering,
Dharmapuri.

Submitted for the university examination held at Government College of Engineering, Dharmapuri on

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We are personally indebted to a number of people who gave us their useful insights to aid in our overall progress for this project. A complete acknowledgement would therefore be encyclopaedic. First of all, we would like to give our deepest gratitude to our parents for permitting us to take up this course.

We record our sincere thanks to **Principal Dr.V.SUMATHY, M.E, Ph.D.**, who provided her kind concern for carrying out this project work and providing suitable environment to work with.

We record our sincere thanks to **Dr.S.SENTHILKUMAR, M.E, Ph.D., Vice Principal** for encouraging us to do this project.

We wish to express our sense of gratitude and sincere thanks to our **Head of the Department Dr.A.M.KALPANA, M.E., Ph.D.**, of Computer Science and Engineering for her valuable guidance in the preparation and presentation of this project.

We express our sincere thanks to our **Project Coordinator Mr.K.PRAKASH, M.E., Assistant Professor**, of Computer Science and Engineering for his enlightening thoughts and remarkable guidance that helped us in the successful completion of our project.

We specially thank our project **Supervisor Mrs.R.NARMATHA M.E., Assistant Professor**, of Computer Science and Engineering for her support and valuable guidance to complete this project successfully.

We thank all our department faculties and our well-wishers for their constant support all the time.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTARCT	VI
	LIST OF FIGURES	VII
	LIST OF ABBREVIATIONS	VIII
1	INTRODUCTION	1
	1.1 Overview	1
	1.2 Objective	2
2	LITERATURE SURVEY	3
3	SYSTEM ANALYSIS	7
	3.1 Existing system	7
	3.2 Proposed system	7
4	REQUIREMENT SPECIFICATION	9
	4.1 Introduction	9
	4.2 Software requirements	9
	4.3 Hardware requirements	10
	4.4 Technologies used	10
	4.4.1 Python	10
	4.4.2 Deep Learning	10
	4.4.3 InceptionV3	10
	4.4.4 Tensorflow/Keras	11
	4.4.5 Flask	11
	4.4.6 HTML & CSS	11

	4.4.7 Google Colab	11
	4.4.8 Visual Studio Code	11
5	DIAGRAM	13
	5.1 Architecture Diagram	13
	5.2 Dataflow Diagram	15
	5.3 Sequence Diagram	17
6	SYSTEM DESIGN	19
	6.1 Modules	19
	6.2 Module explanation	19
	6.2.1 Dataset collection	19
	6.2.2 Data preprocessing	19
	6.2.3 Model training and evaluation	20
	6.2.4 Prediction Via web application	20
	6.3 Dataset	21
	6.3.1 Dataset Source	21
	6.3.2 Classes	21
	6.3.3 Dataset Size	21
	6.3.4 Data Split	22
	6.3.5 Preprocessing & Augmentation	22
	6.3.6 Purpose	22
7	TESTING	23
	7.1 Blackbox testing	23
	7.1.1 Types of Blackbox Testing Applied	23
	7.2 Whitebox testing	24

	7.2.1 Types of Whitebox Testing Applied	25
8	SOURCE CODE	25
9	RESULT ANALYSIS	46
10	CONCLUSION & FUTURE ENHANCEMENT	54
	10.1 Conclusion	54
	10.2 Future Enhancement	54
11	REFERENCES	56

ABSTRACT

Malaria is still a major worldwide health concern, especially in subtropical and tropical areas, for which early and proper diagnosis is important. The current project proposes a deep learning-based diagnosis system for malaria from blood cell images utilizing the InceptionV3 model, which has been acclaimed for high accuracy and speed in image classification. The proposed system can detect parasitized and normal blood cell images with good precision. An easy-to-use web interface, which is built on Flask, enables users to upload single or batch images of cells for real-time analysis and diagnosis. The model is trained on publicly available, annotated data and performs well in terms of accuracy, showcasing its viability for practical application. This solution promises to enable medical professionals with early diagnosis and help bring about better healthcare outcomes in malaria-endemic areas through quick prediction times and high reliability.

LIST OF FIGURES

Figure No.	Figure Title	Page No.
5.1	Architecture Diagram	15
5.2	Data Flow Diagram	17
5.3	Sequence Diagram	19
9.1	Training	46
9.2	Prediction	47
9.3	Testing	48
9.4	Training Accuracy vs Validation Accuracy	49
9.5	Home Page	50
9.6	Upload Page	51
9.7	Result Page	52
9.8	Multiple Result Page	53

LIST OF ABBREVIATIONS

Abbreviation	Full Form
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
JPEG	Joint photographic Experts Groups
RAM	Random Access Memory
API	Application Programmig Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
PyTorch	Python-based Deep Learning Library
FLASK	Python Web Framework

CHAPTER-1

INTRODUCTION

1.1 OVERVIEW

Malaria is a life-threatening infection caused by parasites that are spread to individuals via bites from infected mosquitoes. Early diagnosis and proper diagnosis are crucial for proper treatment and containment of its spread, particularly in areas where access to healthcare resources is limited. This project proposes an automated malaria detection solution based on deep learning using the InceptionV3 architecture, which is efficient and accurate in medical image classification.

The system analyzes microscopic images of blood smears to identify whether cells are parasitized or not. Input images are preprocessed and then passed through the InceptionV3 model, which learns rich and discriminative features. The model classifies the input based on learned infection patterns, providing quick and trustworthy results appropriate for supporting medical experts.

A web interface, designed with Flask, enables users to upload single or multiple cell images for on-the-spot analysis. Providing real-time diagnostic feedback, the system offers download options and export of results in CSV format. Optimized for performance and usability, this solution is meant for practical use in healthcare facilities, particularly in remote or under-equipped locations.

1.2 OBJECTIVE

The aim of this project is to develop and deploy an automated malaria detection system based on deep learning for accuracy, efficiency, and user-friendliness. By using the InceptionV3 model for feature extraction and classification, the system will correctly classify images of microscopic red blood cells as parasitized or uninfected.

The goal is to offer a trustworthy, real-time diagnostic tool minimizing human error and speeding up the diagnostic process in clinical settings. The project also entails creating an easy-to-use web interface with Flask so that users can upload images, display results, and export predictions. Finally, the system aims to aid healthcare practitioners in early malaria diagnosis, leading to early treatment and better patient outcomes, especially in malaria-endemic areas.

CHAPTER-2

LITERATURE SURVEY

Title : Advancing Malaria Identification From Microscopic Blood Smears Using Hybrid Deep Learning Frameworks

Author(s) : D.A, Jadhav, C., & Rani

Year : 17 May 2024

Description: This study presents a hybrid deep learning framework for automated malaria detection using microscopic images of Red Blood Cells (RBCs). The system integrates Convolutional Neural Networks (CNNs) such as ResNet50, Bidirectional Long Short-Term Memory (BiLSTM) networks for temporal context modeling, and attention mechanisms to emphasize critical features in the infected cell regions. This combination enables superior feature extraction, spatial understanding, and parasite recognition across varied image conditions.

Advantages

- Hybrid architecture improves detection accuracy and robustness.
- Effective context modeling through BiLSTM and attention mechanisms .

Disadvantages

- The complexity of hybrid models can increase computational costs and training time.
- It may require large, high-quality datasets to achieve optimal performance and avoid overfitting.

Title : “Enhanced Malaria Detection in Microscopic Blood Smears Using a Hybrid Deep Learning Approach”.

Author : Dev Kumar

Year : 2018

Description: This approach uses a hybrid deep learning model combining VGG16 and InceptionNet to identify malaria parasites in blood smear images. The integration of these networks enhances spatial feature extraction and contextual learning, leading to improved detection accuracy.

Advantages

- The hybrid approach enhances detection accuracy by utilizing complementary strengths from different models, improving overall performance.
- It allows for more robust feature extraction and better generalization, leading to improved identification of malaria in varied image conditions.

Disadvantages

- The complexity of hybrid models requires more computational resources and longer training times.
- It demands large, well-annotated datasets to effectively train the combined models, making it challenging in data-scarce environments.

Title : “Improved Malaria Detection in Microscopic Blood Smears Using Hybrid Deep Learning Models”.

Author(s) : W.David,Yuhang Dong

Year : 2022

Description : This system uses a hybrid model that combines ResNet50 and DenseNet121, enabling efficient extraction of low- and high-level features. The ensemble approach improves the system's robustness and ensures high accuracy even when analyzing diverse or poor-quality blood smear images.

Advantages:

- The hybrid approach leverages the strengths of different models, resulting in higher detection accuracy and better generalization across various blood smear images.
- It improves the system's robustness, making it capable of handling diverse image qualities and variations in malaria presentation.

Disadvantages:

- Hybrid models are computationally intensive and require more time for training and inference, leading to higher resource consumption.
- The system relies on large, high-quality datasets, which may be difficult to obtain in resource-limited settings.

Title : “Hybrid Deep Learning-Based System for Accurate Malaria Detection in Microscopic Blood Smears”.

Author(s) : Pattanaik , Priyadarshini .

Year : 2017

Description : This research utilizes a hybrid deep learning system that integrates a basic CNN for feature extraction with a Support Vector Machine (SVM) classifier for decision-making. The approach automates malaria diagnosis from microscopic images and is designed for use in clinical environments with limited resources.

Advantages:

- Improves detection accuracy by leveraging the strengths of different deep learning models.
- Reduces human error and speeds up diagnosis, making it ideal for use in real-time clinical environments.

Disadvantages:

- Requires high computational power for training and deploying hybrid models.
- Depends on the availability of large, well-labeled datasets for effective performance.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Most existing malaria detection systems rely on manual microscopy, where trained technicians examine blood smear slides under a microscope to identify the presence of malaria parasites. While effective, this method is time-consuming, prone to human error, and heavily dependent on the skill level of the technician. Some recent advancements use classical image processing and traditional machine learning methods for automation, but these approaches often fall short in terms of accuracy and generalizability across different slide qualities and staining techniques. Additionally, many existing automated systems require high computational resources or are not accessible through user-friendly interfaces.

Disadvantages of Existing System:

- Labor-intensive and time-consuming diagnosis process.
- High dependency on human expertise and prone to misdiagnosis.
- Limited accuracy with traditional machine learning methods.
- Lack of accessible and user-friendly platforms for automated detection.

3.2 PROPOSED SYSTEM

The proposed system is a Hybrid Deep Learning-Based Web Application designed to automatically detect malaria from microscopic blood smear images. It utilizes a hybrid deep learning model, primarily based on InceptionV3, enhanced with additional layers for improved feature extraction and classification accuracy.

Advantages of the proposed system:

- The use of a hybrid CNN and InceptionV3 deep learning architecture enables precise and robust feature extraction from blood smear images, thereby improving the accuracy of malaria parasite detection .
- Leveraging transfer learning from a pre-trained model significantly reduces training time and computational requirements, making the system feasible for deployment in resource-constrained environments.
- The system supports both single-image and batch predictions, providing versatility for use in rapid clinical diagnosis or large-scale malaria screening programs.
- The Python-based implementation and modular design facilitate easy integration with web or mobile applications, enabling accessible and immediate malaria diagnosis in healthcare settings.

The combination of these features ensures that the proposed system provides an accurate, efficient, and scalable solution for malaria detection, addressing current challenges and improving diagnostic workflows.

CHAPTER-4

REQUIREMENT SPECIFICATION

4.1 INTRODUCTION

The requirement specification outlines both the software and hardware components essential for the successful development and deployment of the malaria parasite detection system. This includes the tools and frameworks used for model development, training, testing, and deploying a web-based application capable of processing blood smear images to accurately classify the presence of malaria parasites.

4.2 SOFTWARE REQUIREMENTS

- Python (3.7+)
- TensorFlow / Keras
- Flask
- Pillow (PIL)
- NumPy
- Pandas
- HTML & CSS
- Visual Studio Code
- Windows 10
- Google collab
- Ngrok

4.3 HARDWARE REQUIREMENTS

- Intel Core i5 or higher (3.0 GHz, 4 cores)
- GPU
- 16 GB RAM

- 500 GB SSD or higher

4.4 TECHNOLOGIES USED

4.4.1 PYTHON

Python is the primary programming language used in this project. Its simplicity, readability, and vast ecosystem of libraries make it ideal for implementing deep learning models, handling backend development, and integrating different modules.

4.4.2 DEEP LEARNING

Deep learning, a subset of machine learning, leverages neural networks with multiple layers to automatically learn features and patterns from data. In this project, a deep learning model based on InceptionV3 was used to detect malaria parasites in blood smear images. Deep learning enables high accuracy in medical image classification and eliminates the need for manual feature extraction.

4.4.3 INCEPTIONV3

InceptionV3 is a powerful convolutional neural network architecture known for its efficiency and accuracy in image classification tasks. It uses factorized convolutions and auxiliary classifiers to optimize performance. In this project, InceptionV3 was employed as the backbone of the malaria detection model due to its ability to capture complex features from high-resolution medical images.

4.4.4 TENSORFLOW / KERAS

TensorFlow is an open-source deep learning framework developed by Google, and Keras is its high-level API for building and training neural networks. They provide a flexible and user-friendly interface for defining, training, and evaluating models. In this project, TensorFlow/Keras was used to implement and train the InceptionV3 model on malaria-infected and uninfected blood cell images.

4.4.5 FLASK

Flask is a micro web framework for Python used to develop the web interface of the malaria detection system. It acts as the backend server that handles image uploads, processes predictions using the trained deep learning model, and returns results to the user in a simple and efficient manner.

4.4.6 HTML & CSS

HTML and CSS were used to create the front-end of the web application. HTML structures the content, while CSS styles the interface, making it visually appealing and user-friendly. In this project, HTML and CSS were used to design an interactive UI for uploading blood smear images and displaying malaria detection results.

4.4.7 GOOGLE COLAB

Google Colab is a cloud-based platform for training and testing deep learning models using free GPU resources. It was used in this project to train the InceptionV3 model, experiment with different hyperparameters, and monitor performance metrics without needing a local GPU setup.

4.4.8 VISUAL STUDIO CODE

Visual Studio Code is the primary development environment used for coding, testing, and debugging the entire project, including model scripts and Flask web app files. Its extensions and integrated terminal support streamlined the development process.

CHAPTER-5

DIAGRAM

5.1 ARCHITECTURE DIAGRAM

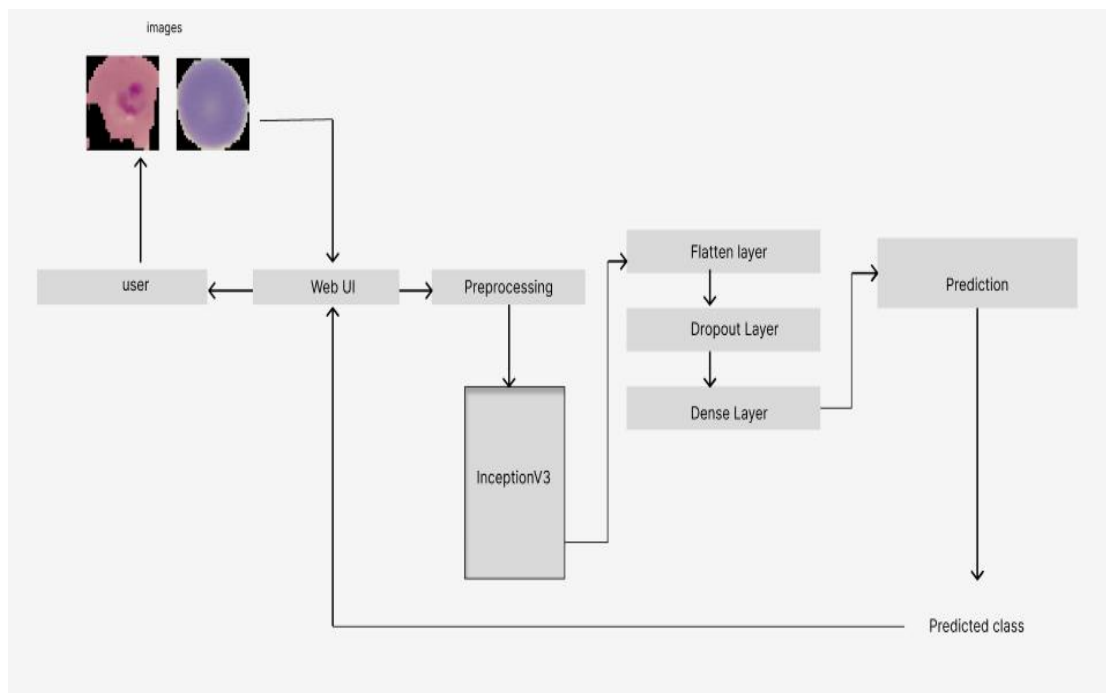


Fig 5.1:Architecture Diagram

Description:

The architecture diagram illustrates the complete workflow of the Automated Malaria Detection System, which integrates a user-friendly web-based interface with a deep learning model for accurate classification of blood slide images. The process begins with the User, who provides input images of blood smears. These images are transmitted to the Web UI, which serves as the interactive frontend, managing the communication between the user and the core prediction engine.

Upon reception by the Web UI, the image proceeds to the Preprocessing module. Here, a series of essential image transformations are applied to prepare

the data for optimal model performance. While specific preprocessing steps are not detailed in the diagram, they typically include resizing, normalization, and other adjustments crucial for standardizing the input for the neural network.

After preprocessing, the image data is fed into the InceptionV3 backbone. InceptionV3 is a powerful pre-trained Convolutional Neural Network (CNN) renowned for its efficiency in extracting rich hierarchical features from images.

The features extracted by InceptionV3 are then passed through a sequence of additional layers designed for classification: a Flatten layer converts the multi-dimensional feature maps into a single vector, followed by a Dropout Layer which helps mitigate overfitting by randomly deactivating neurons during training. Subsequently, a Dense Layer (a fully connected neural network layer) processes these features to make the final classification.

Finally, the system generates the Prediction, which identifies the presence or absence of malaria parasites in the blood slide (e.g., "infected" or "uninfected"). This "Predicted class" is then relayed back to the Web UI to be displayed to the User, providing real-time diagnostic insight. This architecture ensures an efficient, robust, and user-centric approach to automated malaria detection.

5.2 DATAFLOW DIAGRAM

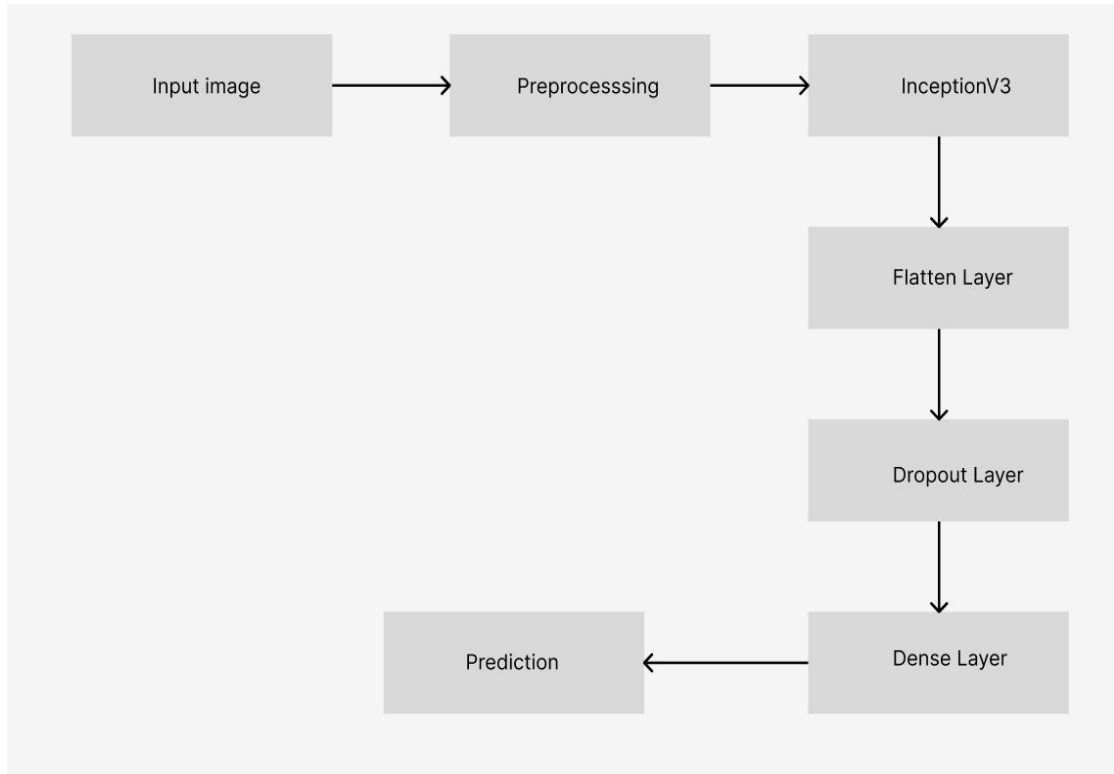


Fig.5.2:Dataflow Diagram

Description:

The process begins when an input image is fed into the system, either uploaded via a user interface or obtained from a dataset. This image undergoes a preprocessing phase, which involves standardization operations such as resizing to the input dimensions required by InceptionV3, normalization of pixel values, and optional augmentations to improve generalization.

Following preprocessing, the refined image is passed into the InceptionV3 model, which serves as the backbone of the feature extraction process. InceptionV3 is a powerful convolutional neural network known for its

ability to extract high-level spatial and semantic features from complex medical images with minimal loss of information.

The output feature maps from InceptionV3 are then passed to a Flatten Layer, which reshapes the multidimensional tensor into a 1D vector, making it compatible for processing by fully connected layers. A Dropout Layer follows, acting as a regularization technique to prevent overfitting by randomly disabling a fraction of neurons during training.

The vector is then passed into a Dense Layer, which maps the extracted features into a decision space. The final node in this pipeline is the Prediction Layer, which outputs a binary result indicating whether the cell in the image is infected with malaria or not.

This modular and hierarchical structure ensures that the system remains scalable, interpretable, and effective for deployment in clinical or mobile diagnostic environments.

5.3 SEQUENCE DIAGRAM

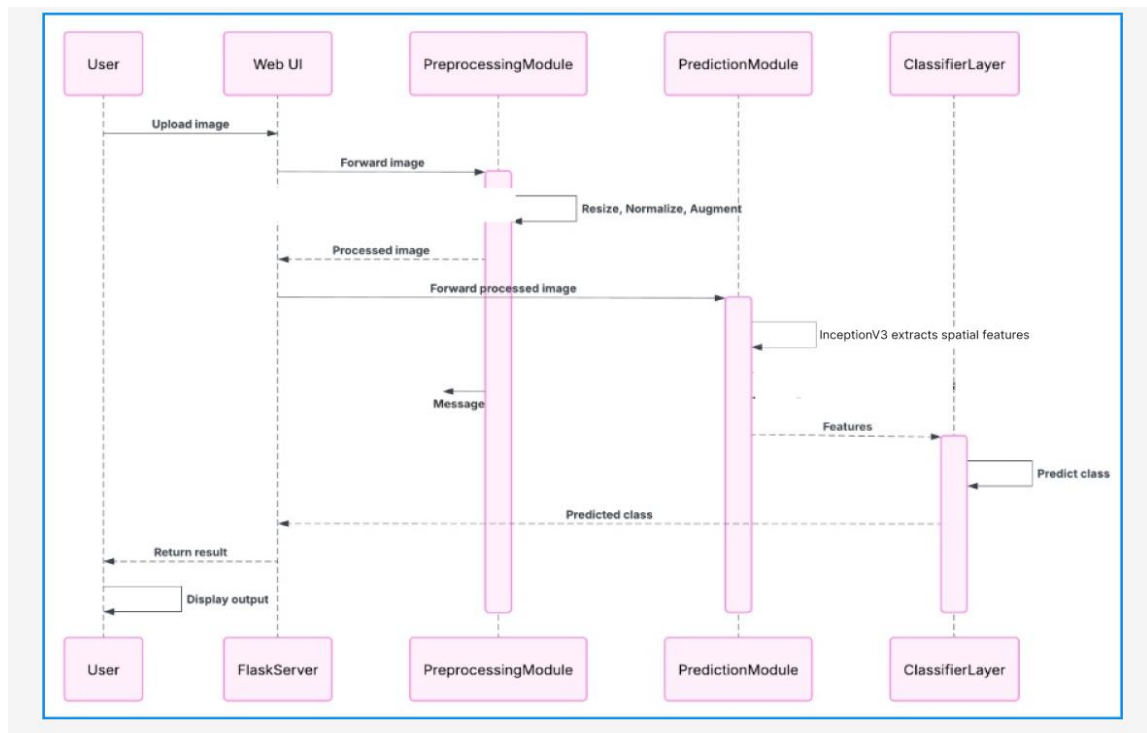


Fig 5.3:Sequence Diagram

Description:

The sequence diagram illustrates the end-to-end interaction between system components in a Malaria Detection System that employs a Flask web interface and an InceptionV3-based deep learning model. The process initiates when the User uploads a blood smear image through the Web UI. This image is transmitted to the Flask Server, which coordinates the backend logic and directs the image to the Preprocessing Module.

Within the Preprocessing Module, the image undergoes crucial enhancement operations including resizing, normalization, and augmentation techniques such as flipping or brightness adjustment. These preprocessing steps are essential to standardize the input and improve model generalization across varied microscopy conditions. Once processed, the cleaned image is forwarded to the Prediction Module.

In the Prediction Module, the InceptionV3 model is responsible for extracting high-level spatial features from the input image. These features are subsequently transferred to the Classifier Layer, which interprets them to predict the class label—identifying whether the cell is infected or uninfected.

The predicted class is returned to the Flask Server, which then sends the result back to the Web UI. Finally, the User receives the diagnosis as a display output. This clearly defined sequence ensures a streamlined, responsive, and accurate image-based malaria classification workflow, suitable for real-time or clinical deployment scenarios.

CHAPTER-6

SYSTEM DESIGN

6.1 MODULES

- Dataset Collection
- Data Preprocessing
- Model Training and Evaluation
- Real-Time Prediction via Web Application

6.2 MODULE EXPLANATION

6.2.1 DATASET COLLECTION

The foundation of the system is a carefully curated dataset containing microscopic images of red blood cells (RBCs), captured through stained blood smear slides. These images are categorized into two distinct classes: **parasitized** (infected with malaria parasites) and **uninfected** (healthy red blood cells). Each image is annotated and verified by trained medical professionals to serve as ground truth labels for supervised learning. A high quality and diverse dataset is essential to the system's performance, as it enables the model to reliably detect malaria across different populations, laboratories, and imaging environments.

6.2.2 DATA PREPROCESSING

To prepare the microscopic blood smear images for effective training, several preprocessing steps were performed. First, all images were resized to a consistent dimension of 224×224 pixels, ensuring uniformity in input size and compatibility with the model's architecture. Pixel values were normalized to a standard range between 0 and 1, which helped reduce variance and improved the model's ability to extract relevant features.

To enhance the model's robustness and generalization, data augmentation techniques were employed. These included horizontal and vertical flipping, random rotations, zooming, and brightness/contrast adjustments to simulate real-world imaging conditions such as lighting variations and differences in slide preparation. This augmentation effectively increased dataset diversity and reduced the risk of overfitting. Additionally, the dataset is split into training, validation, and test sets to evaluate model performance on unseen data.

6.2.3 MODEL TRAINING AND EVALUATION

The malaria detection system employs InceptionV3, a deep convolutional neural network known for its strong ability to extract detailed and multi-scale features from microscopic blood smear images. InceptionV3 processes the images to capture critical characteristics of parasitized and healthy red blood cells, enabling accurate differentiation between infected and uninfected samples. The model uses a final classification layer with a softmax activation function to predict the probability of each class. It is trained using the cross-entropy loss function and evaluated using metrics such as accuracy, precision, recall, and confusion matrices to ensure robust and reliable performance across varied imaging conditions. This approach leverages powerful feature extraction to provide an effective automated malaria diagnosis solution.

6.2.4 PREDICTION VIA WEB APPLICATION

Once the model is trained and evaluated to ensure accessibility and practical usability, the malaria detection system is deployed through a web application developed using the **Flask** framework. This application serves as the user interface, enabling users such as healthcare professionals or laboratory technicians to upload microscopic blood smear images for analysis. Upon image upload, the backend loads the pre-trained deep learning model and processes the image to predict whether the sample is infected with malaria.

parasites. The prediction results are then communicated back to the frontend and displayed in a clear and user-friendly manner. The web application is designed to be responsive, supporting various devices and screen sizes, which facilitates ease of use in diverse environments. This deployment approach demonstrates the real-world applicability of the model by providing a fast, reliable, and accessible tool for malaria diagnosis.

6.3 DATASET

For this project, a labeled image dataset of microscopic blood smear images is utilized to train and evaluate the deep learning model for malaria parasite detection. The dataset plays a crucial role in enabling the model to accurately identify visual patterns that distinguish infected red blood cells from healthy ones.

6.3.1 DATASET SOURCE

Publicly available datasets such as the **Malaria Cell Images Dataset** from the National Institutes of Health (NIH) are used. The dataset contains microscopic images of blood smears, each labeled as either **parasitized** or **uninfected**. Images are organized into separate folders corresponding to these two classes.

6.3.2 CLASSES

The dataset consists of the following two classes:

- Parasitized – Images showing red blood cells infected with malaria parasites.
- Uninfected – Images of healthy red blood cells without infection.

6.3.3 DATASET SIZE

- Total number of images: Approximately 27,558
- Image dimensions: Varying sizes, resized to 224×224 during preprocessing

- Format: JPEG / PNG

6.3.4 DATA SPLIT

- Training set: 80%
- Testing set: 20%

6.3.5 PREPROCESSING & AUGMENTATION

To enhance model generalization, images undergo preprocessing steps including resizing to 224×224 pixels. Data augmentation techniques such as horizontal flipping, brightness and contrast adjustments, and random rotations are applied to simulate real-world variations in imaging conditions. Additionally, normalization is performed to scale pixel values, improving training stability and performance.

6.3.6 PURPOSE

The dataset is used to:

- Train the hybrid deep learning model to effectively detect malaria parasites from blood smear images
- Evaluate the model's performance on unseen images to ensure reliability and robustness
- Deploy the trained model in a real-time web application for practical malaria diagnosis support.

CHAPTER-7

TESTING

7.1 BLACK BOX TESTING

Definition:

Black box testing involves testing the system purely based on inputs and outputs, without any visibility into the internal workings or source code. The primary goal is to validate that the application behaves as expected from a user's point of view.

In this malaria detection system, the internal model architecture (e.g., InceptionV3 or the chosen deep learning model) is treated as a "black box." Testers provide input images of microscopic blood smears through the web interface and verify whether the predicted output—indicating infected or uninfected samples—is accurate. No access to the model's internal code or processing logic occurs during this testing, ensuring the focus remains on validating the system's external behavior and reliability.

7.1.1 Types of Black Box Testing Applied

Functional Testing:

Ensured that uploading a blood smear image, clicking the “Predict” button, and displaying the diagnosis result all function as intended. Verified that the output label (e.g., “Parasitized” or “Uninfected”) correctly corresponds to the input microscopic image.

System Testing:

Validated the complete application workflow from image upload, file

handling, and server response to rendering the prediction result on the user interface.

User Acceptance Testing (UAT):

Conducted testing with end users, including healthcare professionals, using blood smear images of varied quality, staining, and microscopy conditions. Evaluated the usability of the Flask web interface and the accuracy of model predictions in realistic clinical scenarios.

Regression Testing:

After implementing updates to the model or making changes to the web interface, previous test cases were re-executed to confirm that no new issues were introduced and that previously working functionality remained intact.

7.2 WHITE BOX TESTING

Definition:

White box testing, also known as structural or glass-box testing, involves testing the internal logic and workings of the application. The tester (usually the developer) has full access to the source code and focuses on verifying that all internal paths and functions behave as intended.

White box testing was performed to verify the functionality of key components, including image preprocessing, feature extraction (e.g., InceptionV3), and classification layers. Custom functions for data augmentation, normalization, and prediction were rigorously tested. Unit tests and debug logs helped identify and fix errors efficiently.

7.2.1 TYPES OF WHITE BOX TESTING APPLIED

Unit Testing:

Individual Python functions were rigorously tested. The `preprocess_image()` function was checked to ensure correct resizing, normalization, and data augmentation of blood smear images. The `predict_class()` function was tested to confirm that the model consistently outputs accurate malaria infection classifications. Tests used controlled sample images to produce predictable and verifiable results.

Control Flow Testing:

All critical decision points in the code were verified. For example, when users upload unsupported file types, the system properly displays an error message without crashing. Additionally, if any prediction errors occur during model inference, exceptions are caught and handled gracefully to maintain system stability.

Integration Testing:

Testing ensured that all modules interact seamlessly. The Flask backend correctly receives and forwards uploaded blood smear images to the preprocessing module. The processed images are passed to the malaria detection model for inference, and the classification results are returned to and properly displayed by the user interface.

Code Coverage Testing:

Comprehensive testing verified that all vital parts of the codebase—such as model loading, image transformations, prediction logic, and error handling routines—were executed during tests to confirm that no critical functionality was left untested.

CHAPTER-8

SOURCE CODE

```
#import necessary libraries

import os

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import InceptionV3

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.models import load_model


#to mount the google drive for dataset from drive

from google.colab import drive

drive.mount('/content/drive')


from tensorflow.keras.applications import InceptionV3

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam
```

```

# Load InceptionV3 base model (without top layers)

base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))

# Make base model trainable

base_model.trainable = True

# Build the full model

model = Sequential([

    base_model,

    Flatten(),          # Output: (None, 8192)

    Dense(1024, activation='relu'), # Output: (None, 1024)

    Dropout(0.5),

    Dense(1, activation='sigmoid') # Binary classification

])

# Compile the model

model.compile(

    optimizer=Adam(learning_rate=1e-4),

    loss='binary_crossentropy',

    metrics=['accuracy']

)

```

```

# Print model summary

model.summary()

import os

import glob

import requests

import zipfile

import io


# Step 1: Download the dataset ZIP from GitHub

url = "https://github.com/xmartlabs/malaria-
detector/archive/refs/heads/master.zip"

print("Downloading dataset...")

response = requests.get(url)

if response.status_code == 200:

    print("Download successful, extracting files...")

    with zipfile.ZipFile(io.BytesIO(response.content)) as zip_ref:

        zip_ref.extractall(".")

    print("Extraction completed.")

else:

    print("Failed to download dataset.")

    exit(1)

```

```

# Step 2: Define dataset directories after extraction

ROOT_DATA_DIR = './malaria-detector-master/cell_images'

INFECTED_DIR = os.path.join(ROOT_DATA_DIR, 'Parasitized')

UNINFECTED_DIR = os.path.join(ROOT_DATA_DIR, 'Uninfected')


# Step 3: Get list of image files

infected_files = glob.glob(os.path.join(INFECTED_DIR, '*.png'))

uninfected_files = glob.glob(os.path.join(UNINFECTED_DIR, '*.png'))


# Step 4: Print the counts

print(f'Amount of parasitized images: {len(infected_files)}')

print(f'Amount of uninfected images: {len(uninfected_files)}')

print(f'Total Images: {len(infected_files) + len(uninfected_files)}')


# Data preparation and augmentation

# Data Augmentation & Normalization Setup

train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)


#Training Data Preparation

train_generator = train_datagen.flow_from_directory(

    './malaria-detector-master/cell_images',

    target_size=(128, 128), # match model input

    batch_size=32,

```

```
class_mode='binary',  
subset='training'  
)
```

#Validation Data Preparation

```
val_generator = train_datagen.flow_from_directory(  
    './malaria-detector-master/cell_images',  
    target_size=(128, 128),  
    batch_size=32,  
    class_mode='binary',  
    subset='validation'  
)
```

#Model Training

```
history = model.fit(  
    train_generator,  
    validation_data=val_generator,  
    epochs=5,  
    verbose=1  
)
```

Model Performance Visualization

```
plt.figure(figsize=(8,6))  
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
#To save the model
```

```
model.save('/content/drive/MyDrive/inceptionv3.h5')
```

```
#Full working model at collab
```

```
import os
```

```
import numpy as np
```

```
import pandas as pd
```

```
from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.preprocessing import image
```

```
# Load your trained model
```

```
model = load_model('/content/drive/MyDrive/InceptionV3.h5')
```

```
def preprocess_and_predict(img_path):
```

```
    try:
```

```
        img = image.load_img(img_path, target_size=(128, 128))
```



```

img_array = image.img_to_array(img)

img_array = np.expand_dims(img_array, axis=0)

img_array = img_array / 255.0

prediction = model.predict(img_array)[0][0]

label = 'Infected' if prediction >= 0.5 else 'Uninfected'

return float(prediction), label

except Exception as e:

    print(f'Error processing {img_path}: {e}')

    return None, None

```

choice = input("Choose prediction type (enter number):\n1. Single Image\n2. Folder of Images\nYour choice: ")

```

if choice == '1':

    img_path = input("Enter the full path of the image: ").strip()

    if not os.path.isfile(img_path):

        print("File does not exist:", img_path)

    else:

        pred_score, pred_label = preprocess_and_predict(img_path)

        if pred_score is not None:

            print(f'Prediction score: {pred_score}')

            print(f'Result: {pred_label}')

elif choice == '2':

```

```

folder_path = input("Enter the full path of the image folder: ").strip()

if not os.path.isdir(folder_path):

    print(" Folder does not exist:", folder_path)

else:

    image_files = [f for f in os.listdir(folder_path) if f.lower().endswith(('.jpg',
'.jpeg', '.png'))]

    if not image_files:

        print(" No images found in folder:", folder_path)

    else:

        print(f" Found {len(image_files)} images.")

        results = []

        for filename in image_files:

            img_path = os.path.join(folder_path, filename)

            pred_score, pred_label = preprocess_and_predict(img_path)

            if pred_score is not None:

                results.append({

                    'filename': filename,

                    'prediction_score': pred_score,

                    'label': pred_label

                })

        if results:

            df = pd.DataFrame(results)

            output_csv = '/content/drive/MyDrive/cell_predictions.csv'

            df.to_csv(output_csv, index=False)

```

```

        print(f" Predictions saved to: {output_csv}")

    else:

        print(" No predictions made. Check for errors above.")

else:

    print(" Invalid choice. Please run again and choose 1 or 2.")

# to host online

#Setup for Running Flask App in a Colab

# Install pyngrok and flask

!pip install flask pyngrok tensorflow pillow pandas


# Set your ngrok authtoken (run this once per session)

!ngrok
2xRZPc8KhsrfRGT2R84fbFuVqn9_6MdCDB2W7b8e31P5mocUh          authtoken

#to load the model from drive

from google.colab import drive

drive.mount('/content/drive')

# Install dependencies (run once)

!pip install flask pyngrok tensorflow pillow pandas


# --- Flask app code starts here ---

import os

import io

import numpy as np

import pandas as pd

```

```
from flask import Flask, request, render_template_string, redirect, url_for, flash,
send_file
```

```
from tensorflow.keras.models import load_model
```

```
from PIL import Image
```

```
from pyngrok import ngrok
```

```
app = Flask(__name__)
```

```
app.secret_key = "supersecretkey"
```

```
MODEL_PATH = '/content/drive/MyDrive/InceptionV3.h5'
```

```
model = load_model(MODEL_PATH)
```

```
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
```

```
def allowed_file(filename):
```

```
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS
```

```
def preprocess_image(img):
```

```
    img = img.resize((128, 128))
```

```
    img_array = np.array(img)
```

```
    if img_array.shape[-1] == 4:
```

```
        img_array = img_array[..., :3]
```

```
    img_array = img_array / 255.0
```

```
img_array = np.expand_dims(img_array, axis=0)

return img_array
```

```
def predict(img):

    img_array = preprocess_image(img)

    pred = model.predict(img_array)[0][0]

    label = "Infected" if pred >= 0.5 else "Uninfected"

    return pred, label
```

```
INDEX_HTML = """

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <title>Malaria Cell Detection</title>

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet" />

</head>

<body class="bg-light">

<div class="container py-5">

    <h1 class="mb-4 text-center">Malaria Blood Smear Cell Detection</h1>
```

```
{% with messages = get_flashed_messages() %}

{% if messages %}

<div class="alert alert-danger" role="alert">

  <ul class="mb-0">

    {% for message in messages %}

      <li>{{ message }}</li>

    {% endfor %}

  </ul>

</div>

{% endif %}

{% endwith %}
```

```
<form method="POST" enctype="multipart/form-data" class="border p-4
bg-white rounded shadow-sm">
```

```
<div class="mb-3">

  <label class="form-label me-3">Choose upload type:</label>

  <div class="form-check form-check-inline">

    <input class="form-check-input" type="radio" name="upload_type"
id="singleRadio" value="single" checked>

    <label class="form-check-label" for="singleRadio">Single
Image</label>

  </div>

  <div class="form-check form-check-inline">

    <input class="form-check-input" type="radio" name="upload_type"
id="multipleRadio" value="multiple">
```

```
        <label    class="form-check-label"    for="multipleRadio">Multiple
Images</label>
```

```
    </div>
```

```
</div>
```

```
<div class="mb-3" id="single_upload">
```

```
    <label    for="single_file"    class="form-label">Upload    Single
Image</label>
```

```
    <input    class="form-control"    type="file"    id="single_file"
name="single_file" accept=".jpg,.jpeg,.png" />
```

```
</div>
```

```
<div class="mb-3 d-none" id="multiple_upload">
```

```
    <label    for="multiple_files"    class="form-label">Upload    Multiple
Images</label>
```

```
    <input    class="form-control"    type="file"    id="multiple_files"
name="multiple_files" accept=".jpg,.jpeg,.png" multiple />
```

```
</div>
```

```
<button type="submit" class="btn btn-primary w-100">Predict</button>
```

```
</form>
```

```
</div>
```

```
<script>
```

```
    const singleRadio = document.getElementById('singleRadio');
```

```

const multipleRadio = document.getElementById('multipleRadio');
const singleUpload = document.getElementById('single_upload');
const multipleUpload = document.getElementById('multiple_upload');

function toggleUploadFields() {
    if(singleRadio.checked) {
        singleUpload.classList.remove('d-none');
        multipleUpload.classList.add('d-none');
    } else {
        singleUpload.classList.add('d-none');
        multipleUpload.classList.remove('d-none');
    }
}

singleRadio.addEventListener('change', toggleUploadFields);
multipleRadio.addEventListener('change', toggleUploadFields);
</script>

</body>

</html>

""""

RESULT_HTML = """"

```



```

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <title>Prediction Results</title>

  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet" />

</head>

<body class="bg-light">

<div class="container py-5">

  <h1 class="mb-4 text-center">Prediction Results</h1>

  <div class="table-responsive shadow-sm rounded bg-white p-3">

    <table class="table table-bordered table-hover mb-0">

      <thead class="table-primary">

        <tr>

          <th>Filename</th>

          <th>Prediction Score</th>

          <th>Label</th>

        </tr>

      </thead>

      <tbody>

```

```

{% for pred in predictions %}

<tr>

    <td>{{ pred.filename }}</td>

    <td>{{ "%.4f"|format(pred.score) }}</td>

    <td>{{ pred.label }}</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

<div class="mt-4 text-center">

    {% if not single %}

    <form action="{{ url_for('download_csv') }}" method="post" class="d-
inline">

        <input                                type="hidden"                                name="csv_data"
value="{{ csv_data|tojson|safe }}">

        <button type="submit" class="btn btn-success me-3">Download
CSV</button>

    </form>

    {% endif %}

    <a href="{{ url_for('index') }}" class="btn btn-secondary">Back to
Upload</a>

</div>

</div>

```

```
</body>
```

```
</html>
```

```
"""
```

```
@app.route('/', methods=['GET', 'POST'])
```

```
def index():
```

```
    if request.method == 'POST':
```

```
        upload_type = request.form.get('upload_type')
```

```
        if upload_type == 'single':
```

```
            if 'single_file' not in request.files:
```

```
                flash('No file part')
```

```
                return redirect(request.url)
```

```
            file = request.files['single_file']
```

```
            if file.filename == ":
```

```
                flash('No selected file')
```

```
                return redirect(request.url)
```

```
            if file and allowed_file(file.filename):
```

```
                try:
```

```
                    img = Image.open(file.stream)
```

```
                    pred_score, pred_label = predict(img)
```

```
                    predictions = [{'filename': file.filename, 'score': pred_score, 'label':  
pred_label}]
```

```

        return render_template_string(RESULT_HTML,
predictions=predictions, single=True)

    except Exception as e:

        flash(f'Error processing image: {e}')

        return redirect(request.url)

    else:

        flash('Allowed file types: png, jpg, jpeg')

        return redirect(request.url)

elif upload_type == 'multiple':

    files = request.files.getlist('multiple_files')

    if not files or files[0].filename == "":

        flash('No files selected')

        return redirect(request.url)

    results = []

    for file in files:

        if file and allowed_file(file.filename):

            try:

                img = Image.open(file.stream)

                pred_score, pred_label = predict(img)

                results.append({'filename': file.filename, 'score': pred_score,
'label': pred_label})

            except Exception as e:

```

```

        flash(f'Error processing {file.filename}: {e}')
    else:
        flash(f'Skipped invalid file: {file.filename}')

    if results:
        df = pd.DataFrame(results)
        csv_buffer = io.StringIO()
        df.to_csv(csv_buffer, index=False)
        csv_buffer.seek(0)
        csv_content = csv_buffer.getvalue()
        return render_template_string(RESULT_HTML, predictions=results,
single=False, csv_data=csv_content)
    else:
        flash('No valid images uploaded')
        return redirect(request.url)

return render_template_string(INDEX_HTML)

@app.route('/download_csv', methods=['POST'])
def download_csv():
    csv_data = request.form.get('csv_data')
    if not csv_data:
        flash("No CSV data available for download.")
        return redirect(url_for('index'))

```

```
return send_file(io.BytesIO(csv_data.encode()),
                 mimetype='text/csv',
                 as_attachment=True,
                 download_name='malaria_predictions.csv')

# Start ngrok tunnel (make sure you added your auth token beforehand)
public_url = ngrok.connect(5000)
print(f" * ngrok tunnel URL: {public_url}")

# Run Flask app
app.run(port=5000)
```

CHAPTER-9

RESULT ANALYSIS

Training

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
WARNING:absl>Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
Model loaded successfully.
Found 22048 images belonging to 2 classes.
Found 5510 images belonging to 2 classes.
Epoch 1/10
689/689 ----- 144s 111ms/step - accuracy: 0.7561 - loss: 1.3380 - val_accuracy: 0.9708 - val_loss: 0.0830
Epoch 2/10
689/689 ----- 68s 98ms/step - accuracy: 0.9721 - loss: 0.0857 - val_accuracy: 0.9751 - val_loss: 0.0771
Epoch 3/10
689/689 ----- 68s 98ms/step - accuracy: 0.9738 - loss: 0.0801 - val_accuracy: 0.9735 - val_loss: 0.0795
Epoch 4/10
689/689 ----- 68s 99ms/step - accuracy: 0.9755 - loss: 0.0696 - val_accuracy: 0.9708 - val_loss: 0.0848
Epoch 5/10
689/689 ----- 69s 100ms/step - accuracy: 0.9777 - loss: 0.0652 - val_accuracy: 0.9701 - val_loss: 0.0899
Epoch 6/10
689/689 ----- 69s 100ms/step - accuracy: 0.9801 - loss: 0.0614 - val_accuracy: 0.9715 - val_loss: 0.0904
Epoch 7/10
689/689 ----- 82s 100ms/step - accuracy: 0.9835 - loss: 0.0506 - val_accuracy: 0.9724 - val_loss: 0.0847
Epoch 8/10
689/689 ----- 69s 99ms/step - accuracy: 0.9830 - loss: 0.0516 - val_accuracy: 0.9710 - val_loss: 0.1015
Epoch 9/10
689/689 ----- 69s 100ms/step - accuracy: 0.9867 - loss: 0.0421 - val_accuracy: 0.9701 - val_loss: 0.1143
Epoch 10/10
689/689 ----- 69s 100ms/step - accuracy: 0.9891 - loss: 0.0335 - val_accuracy: 0.9721 - val_loss: 0.1083
```

Fig 9.1: Training

Description:

This image shows the terminal output during the training phase of a deep learning model across 10 epochs. The model begins with a training accuracy of 75.61% and quickly improves to 98.91% by the final epoch, indicating a steep learning curve. Simultaneously, the training loss decreases significantly from 1.3380 to 0.0335, reflecting effective optimization. Validation accuracy remains consistently high (around 97%), with a slight increase in validation loss in the later epochs, possibly suggesting mild overfitting. Overall, the model demonstrates strong training performance with stable generalization.

Prediction

```
def preprocess_and_predict(img_path):
    try:
        img = image.load_img(img_path, target_size=(128, 128))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0
        prediction = model.predict(img_array)[0][0]
        label = 'Infected' if prediction > 0.5 else 'Uninfected'
        return float(prediction), label
    except Exception as e:
        print(f'Error processing {img_path}: {e}')
        return None, None

choice = input("Choose prediction type (enter number): 1. Single Image 2. Folder of Images\nYour choice: ")

if choice == "1":
    img_path = input("Enter the full path of the image: ").strip()
    if not os.path.exists(img_path):
        print("X File does not exist.", img_path)
    else:
        pred_score, pred_label = preprocess_and_predict(img_path)
        if pred_score is not None:
            print(f"Prediction score: {pred_score}")
            print(f"Result: {pred_label}")
else:
    folder_path = input("Enter the full path of the image folder: ").strip()
    if not os.path.exists(folder_path):
        print("X Folder does not exist.", folder_path)
    else:
        image_files = [f for f in os.listdir(folder_path) if f.lower().endswith('.jpg' or '.png')]
        if not image_files:
            print("X No images found in folder.", folder_path)
        else:
            print(f"Found {len(image_files)} images.")
            results = []
            for filename in image_files:
                img_path = os.path.join(folder_path, filename)
                pred_score, pred_label = preprocess_and_predict(img_path)
                if pred_score is not None:
                    results.append(
                        {
                            'filename': filename,
                            'prediction_score': pred_score,
                            'label': pred_label
                        }
                    )
            if results:
                df = pd.DataFrame(results)
                output_csv = os.path.join(os.path.dirname(__file__), 'output_predictions.csv')
                df.to_csv(output_csv, index=False)
                print(f"Predictions saved to: {output_csv}")
            else:
                print("X No predictions made. Check for errors above.")
else:
    print("X Invalid choice. Please run again and choose 1 or 2.")
```

Fig 9.2: Prediction

Description:

This image shows Python code for **Malaria Detection**, focusing on the process of making predictions on new blood slide images. The main part of the script allows the user to choose between predicting on a single image or a folder of images, performing necessary file path validation and image processing. It concludes with a prediction function that takes an input image and returns the predicted class and demonstrates the core inference logic for a malaria detection system.

Testing

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
Choose prediction type (enter number):
1. Single Image
2. Folder of Images
Your choice: 1
Enter the full path of the image: /content/drive/MyDrive/malaria_dataset/test/abilash.png
1/1 8s 8s/step
Prediction score: 0.14754100143909454
Result: Uninfected
```

Fig 9.3:Testing

Description:

Here, the model is evaluated on a single image selected from the test dataset. The code prompts the user to choose between single or batch image prediction, and then processes the specified image using the trained model, the model successfully loads and performs the prediction. The output shows a prediction score of 0.1475, which is below the classification threshold (commonly 0.5), leading to a final result of "Uninfected." This demonstrates that the model is functioning correctly in inference mode and is capable of classifying unseen test images based on learned features.

Training Vs Testing Accuracy

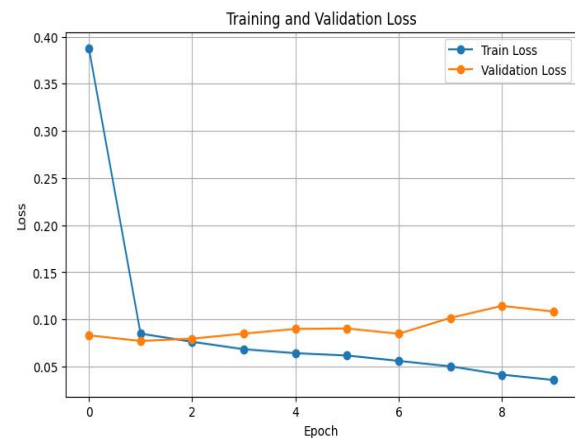
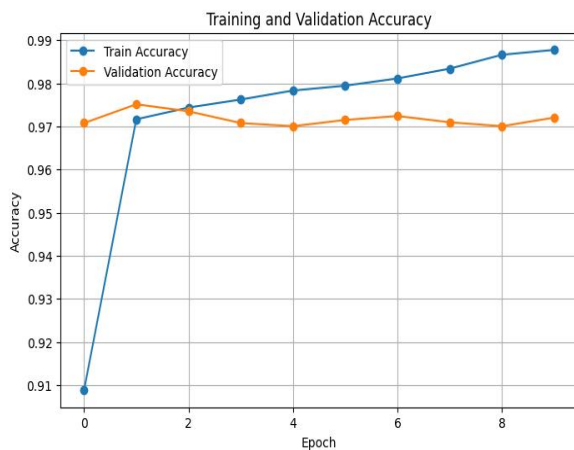


Fig 9.4: Training Vs Testing Accuracy

Description:

This image presents two key plots that effectively summarize the model's training performance. The left plot illustrates the comparison between training and validation accuracy over 10 epochs. The training accuracy steadily increases, reaching close to 99%, while the validation accuracy remains consistently high, indicating good generalization. The right plot shows the training and validation loss curves. The training loss rapidly decreases in the early epochs and continues to decline steadily, while the validation loss remains relatively stable with slight fluctuations. This suggests the model is learning efficiently without significant signs of overfitting or instability.

Home Page

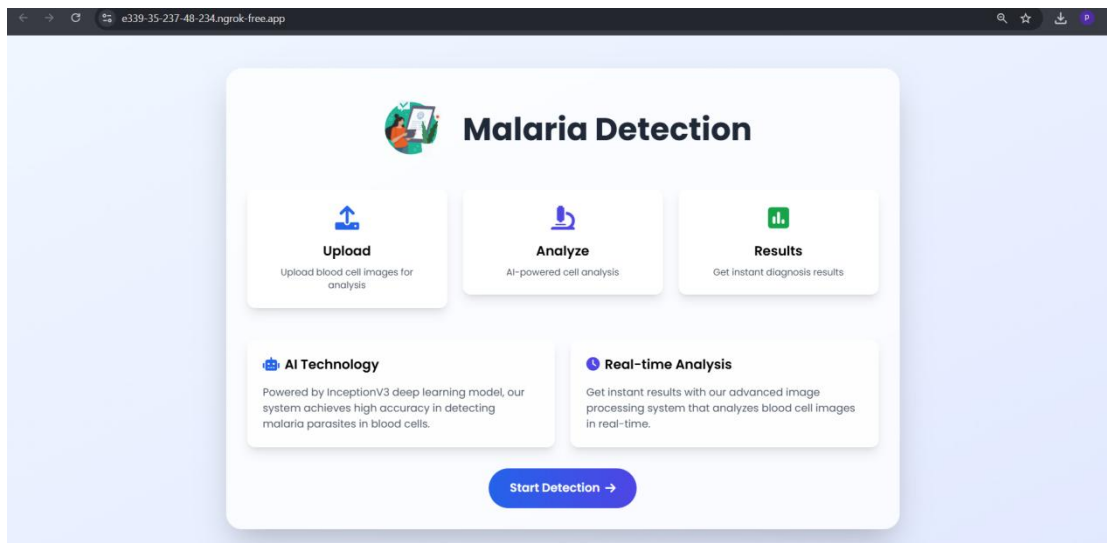
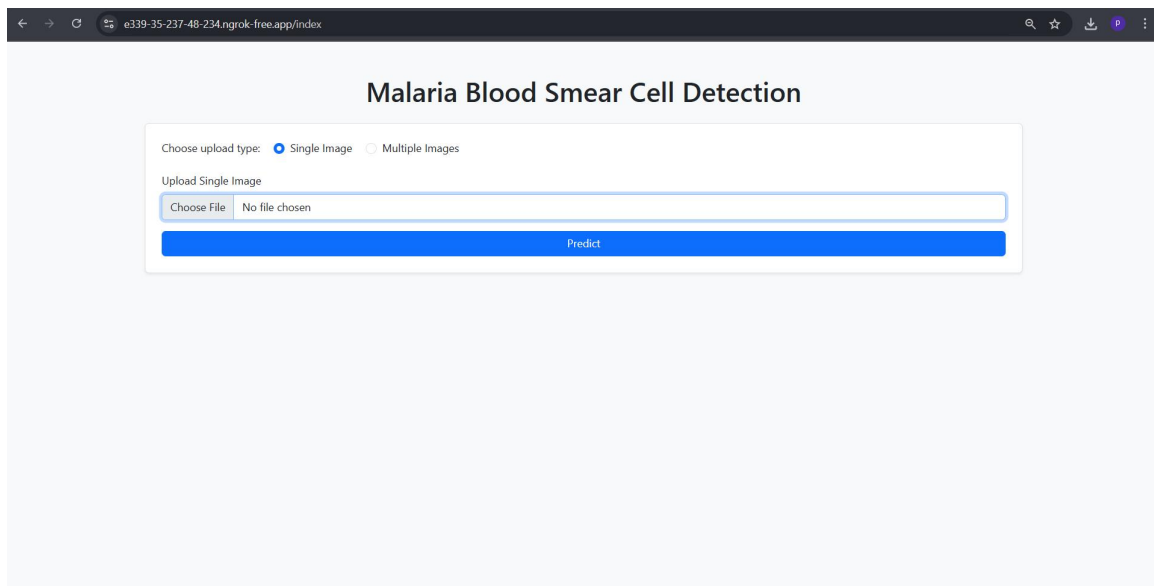


Fig 9.5:Home Page

Description:

The screenshot displays the homepage of a Malaria Detection Web Application designed for quick and user-friendly diagnosis. Users can upload blood smear images, which are analyzed using an AI-powered InceptionV3 model to detect malaria parasites. The interface guides users through three steps: Upload, Analyze, and Results, ensuring a smooth workflow. Key features include high detection accuracy through deep learning and real-time image analysis. With a clean design and a “Start Detection” button, the application offers instant diagnostic results for efficient and accessible malaria screening.

Upload Page



The screenshot displays a web application interface for "Malaria Blood Smear Cell Detection". The browser's address bar shows the URL "e339-35-237-48-234.ngrok-free.app/index". The page features a central form with the following elements:

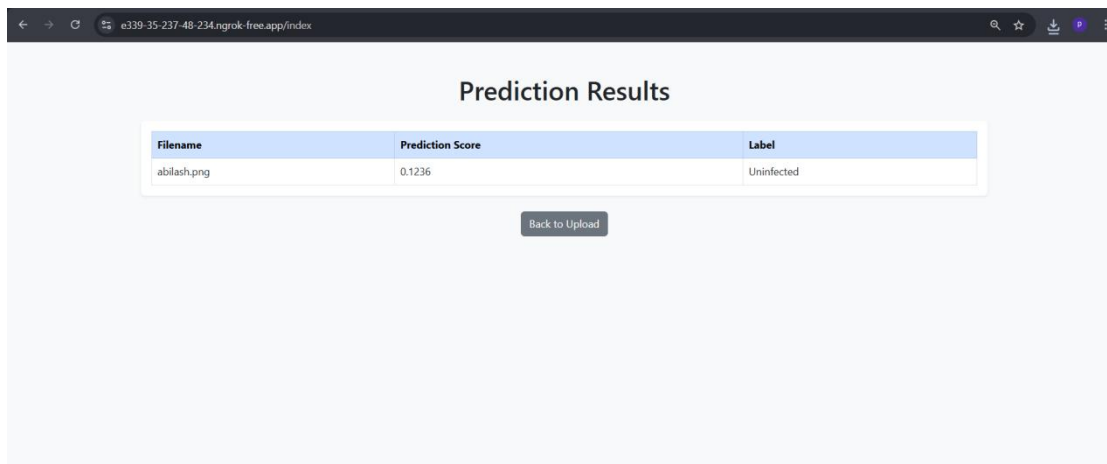
- Title:** "Malaria Blood Smear Cell Detection"
- Upload Type Selection:** "Choose upload type:" with two radio buttons: "Single Image" (selected) and "Multiple Images".
- Single Image Upload Section:**
 - Label:** "Upload Single Image"
 - File Selection:** A button labeled "Choose File" next to a text field displaying "No file chosen".
 - Action:** A prominent blue button labeled "Predict" located below the file selection area.

Fig 9.6: Upload Page

Description:

The screenshot showcases the image upload interface of a Malaria Blood Smear Cell Detection web application. Users can select either a single image or multiple images for upload. The clean layout features a file selection field and a prominent “Predict” button, allowing users to quickly submit blood smear images for AI-based analysis. This intuitive interface ensures a streamlined and accessible diagnostic process for detecting malaria.

Result Page



The screenshot shows a web browser window with the address bar displaying 'e339-35-237-48-234.ngrok-free.app/index'. The main content area is titled 'Prediction Results' and contains a table with the following data:

Filename	Prediction Score	Label
abilash.png	0.1236	Uninfected

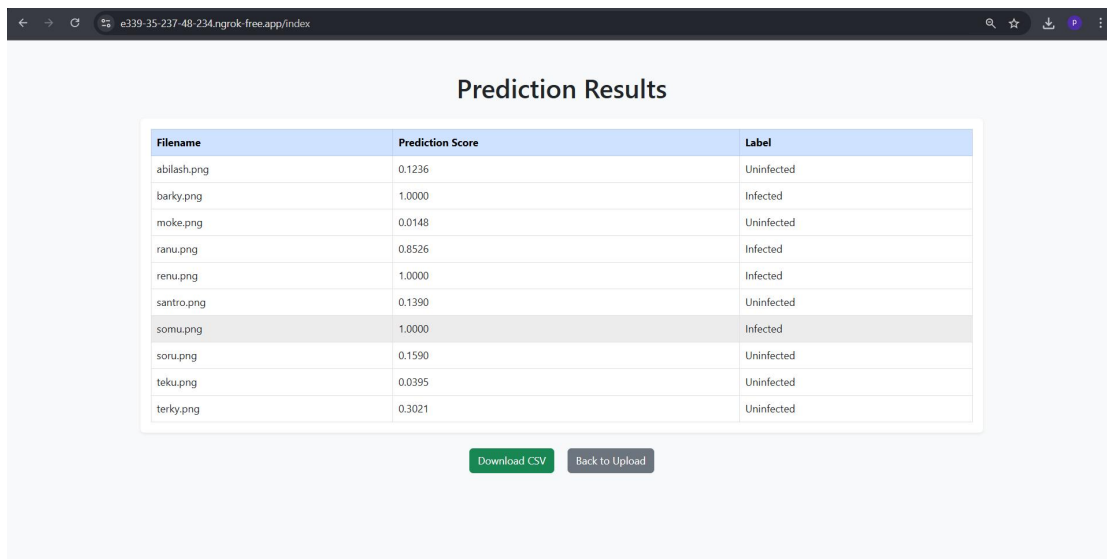
Below the table is a button labeled 'Back to Upload'.

Fig 9.7:Result Page

Description:

The displayed screen presents the prediction results of the malaria detection system. It lists the filename of the uploaded image, the corresponding prediction score generated by the model, and the final classification label. In this example, the file "abilash.png" received a prediction score of 0.1236 and was classified as "Uninfected." The interface also provides a convenient "Back to Upload" button, allowing users to analyze additional images efficiently.

Multiple Result Page



Prediction Results

Filename	Prediction Score	Label
abilash.png	0.1236	Uninfected
barky.png	1.0000	Infected
moke.png	0.0148	Uninfected
ranu.png	0.8526	Infected
renu.png	1.0000	Infected
santro.png	0.1390	Uninfected
somu.png	1.0000	Infected
sonu.png	0.1590	Uninfected
teku.png	0.0395	Uninfected
terky.png	0.3021	Uninfected

[Download CSV](#) [Back to Upload](#)

Fig 9.8:Multiple Result Page

Description:

The results page displays a table summarizing malaria detection predictions for multiple uploaded blood smear images. Each row includes the filename, the model's prediction score, and the final label indicating whether the image is "Infected" or "Uninfected." Users can conveniently view detailed outcomes and download the results in CSV format using the "Download CSV" button. A "Back to Upload" option is also provided for re-analysis or uploading new images.

CHAPTER 10

CONCLUSION & FUTURE ENHANCEMENT

10.1 CONCLUSION

This project presents an effective and accessible deep learning-based system for the automatic detection of malaria from blood smear images. Using a Convolutional Neural Network (CNN) architecture such as InceptionV3, the model is trained to differentiate between parasitized and uninfected cells with high accuracy. Comprehensive preprocessing steps including normalization, resizing, and data augmentation ensured that the model could generalize well to varying imaging conditions and sample quality.

A user-friendly web application was developed using the Flask framework, allowing users to upload blood smear images and receive real-time predictions directly in the browser. The system demonstrates reliable performance with strong evaluation metrics such as accuracy, precision, and recall, making it suitable for use in resource-constrained environments, such as remote clinics or field laboratories. The lightweight model, combined with automated prediction, offers a practical and scalable tool for aiding malaria diagnosis and reducing diagnostic workload.

10.2 FUTURE ENHANCEMENT

To further improve the effectiveness and scalability of the proposed malaria detection system, several future enhancements are envisioned. Firstly, the integration of a larger and more diverse dataset, including samples from various geographic regions and imaging devices, can improve the model's robustness and generalizability. Secondly, expanding the classification capability to distinguish between different *Plasmodium* species (e.g., *P. falciparum*, *P. vivax*) can provide more detailed diagnostic insights.

Additionally, implementing real-time image capture using mobile phone cameras or portable digital microscopes can enhance accessibility in field settings. The system can also be upgraded with active learning techniques, allowing the model to improve continuously with user feedback. Moreover, deploying the application on cloud platforms with secure user access can facilitate remote diagnosis and centralized data storage for large-scale epidemiological monitoring.

Finally, integrating the system into a broader healthcare framework—such as electronic medical records or mobile health platforms—can support comprehensive patient management and decision-making in malaria-endemic regions.

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).

