HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Team ID : LTVIP2025TMID40988

Team Leader : Varshini Polnati

Team member : Veluri Rashmitha

Team member : Vipparthi Madhu Babu

Team member : Yerravarapu Yamuna

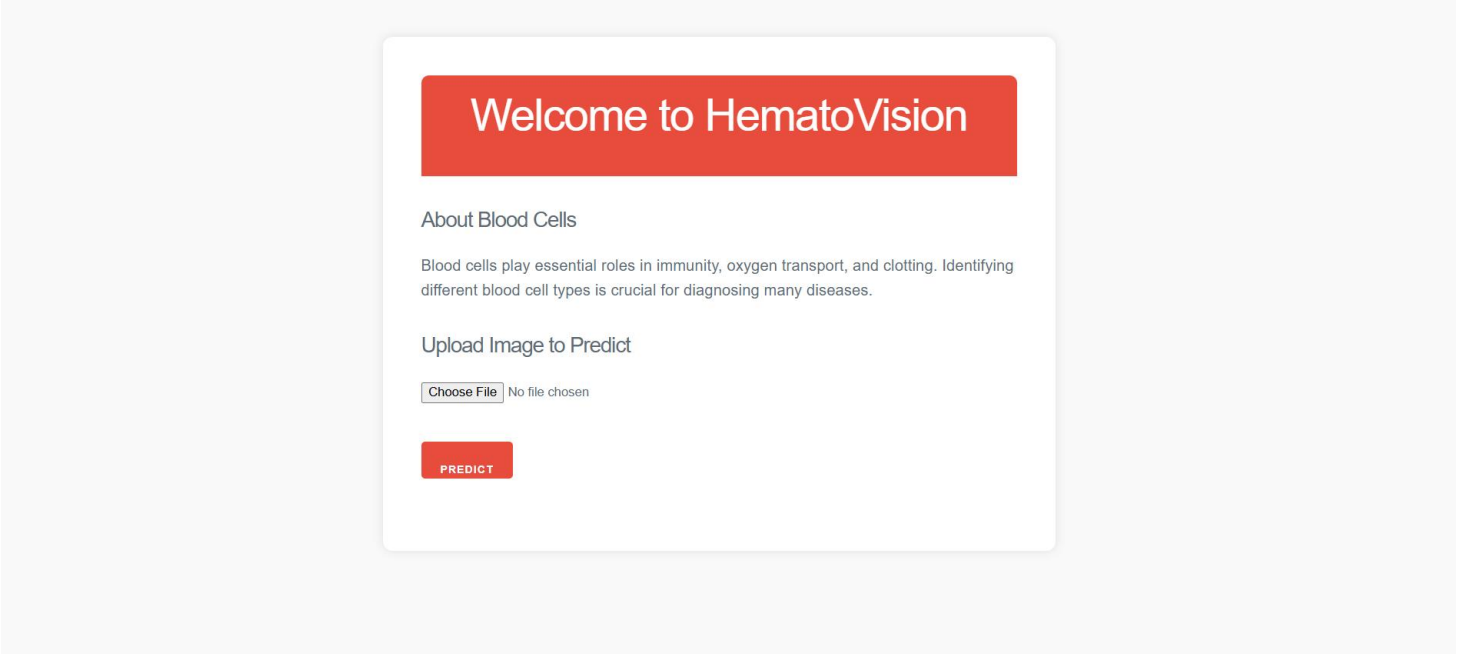
Table of Contents

1. Introduction

HematoVision is a deep learning-based web application designed to classify blood cells into four major types — Eosinophil, Lymphocyte, Monocyte, and Neutrophil — using transfer learning. The goal is to assist medical professionals and pathology labs by providing fast, reliable, and accurate classification from microscope images.

2. Project Overview

HematoVision: Advanced Blood Cell Classification Using Transfer Learning is a healthcare-focused AI project that aims to accurately classify blood cells using deep learning techniques. This project utilizes transfer learning with pre-trained convolutional neural networks to analyze and categorize blood cell images into classes such as neutrophils, lymphocytes, monocytes, and eosinophils. By leveraging medical image data, HematoVision enables early and precise diagnosis of blood-related disorders, supporting pathologists with faster and more reliable analysis. The system is designed to improve diagnostic efficiency, reduce manual workload, and contribute to enhanced patient care through intelligent automation.



3.Purpose

- To enhance the accuracy and reliability of blood cell classification through deep learning.
- To leverage transfer learning for reducing training time and improving model performance.
- To provide a cost-effective alternative to expensive diagnostic tools.
- To standardize blood cell classification procedures across different labs and institutions.
- To build an image-based dataset tailored for training and testing the classification model.
- To integrate visualization tools for better interpretability of results.

4. Phase 1 : BrainStorming & Ideathon

Objective: To explore the applicability and benefits of deep learning for microscopic blood cell classification. This phase involves defining the problem, identifying target users, and visualizing the solution with expected outcomes.

Key Points:

1. Problem Statement

Manual cell analysis takes significant time and effort.

Errors in manual diagnosis can lead to health risks.

Rural areas often lack expert hematologists.

There's no quick, accessible diagnosis method.

Automation is essential for scalability.

2. Proposed Solution

Develop a CNN-based classification model.

Integrate with a Flask web application.

Ensure accurate and fast predictions.

Provide easy-to-use UI for users.

Allow image upload and real-time prediction.

3. Target Users

Pathologists and diagnostic labs.

Medical students and researchers.

Rural healthcare providers.

Hospitals and clinics.

Tech enthusiasts exploring AI in medicine.

4. Expected Outcome

A trained deep learning model in .h5 format.

A responsive web interface for image upload.

Classification output with visual feedback.

Deployment-ready codebase on GitHub.

5. PHASE 2: Requirement Analysis

Objective: To define the technical stack and functionality of the application.

Key Points:

1. Technical Requirements

Language: Python

Libraries: TensorFlow, Keras, NumPy, OpenCV

Frameworks: Flask, Jupyter Notebook

Tools: VS Code, Anaconda, GitHub

Hosting: Localhost for testing

2. Functional Requirements

Upload blood cell image via UI.

Image preprocessing and normalization.

Model inference and result display.

Return predicted cell type.

Save prediction logs (optional).

3. Constraints & Challenges

Handling image resolution variations.

Model overfitting due to limited data.

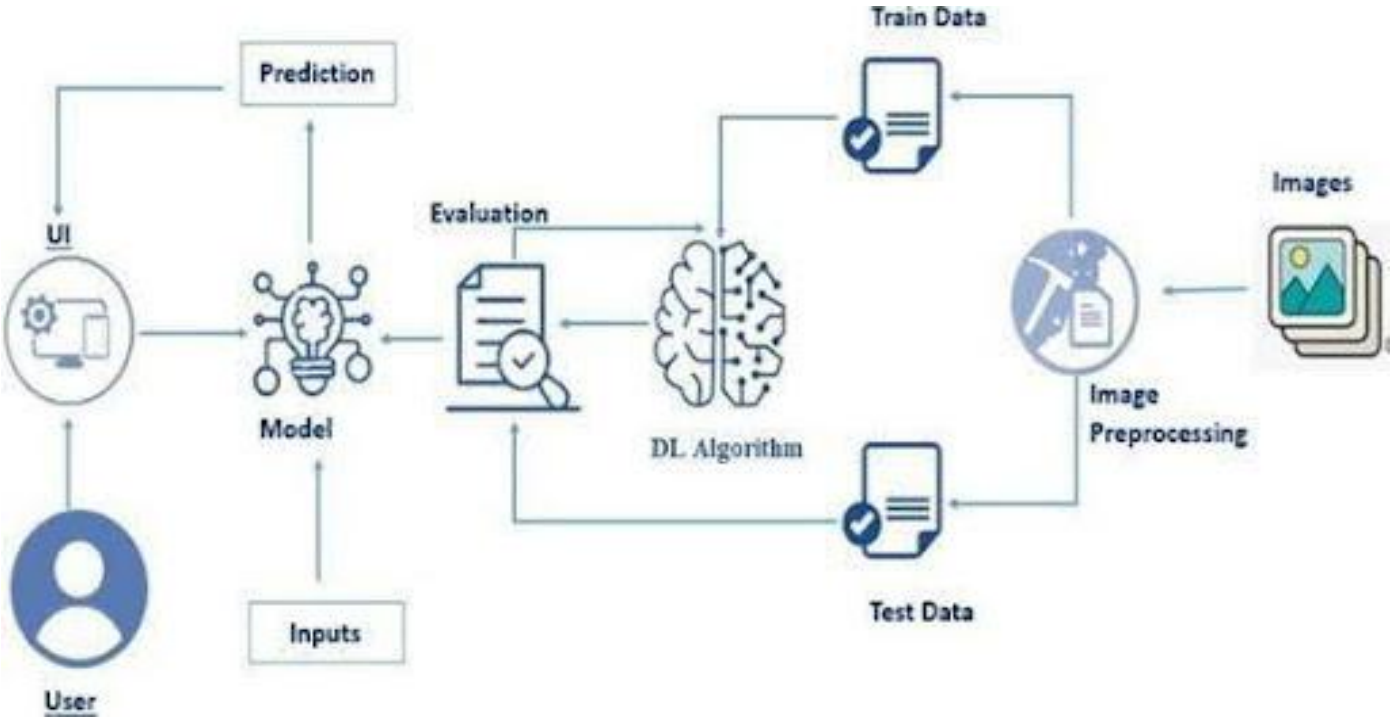
Integration between Flask and Keras.

Deployment compatibility.

UI responsiveness across devices.

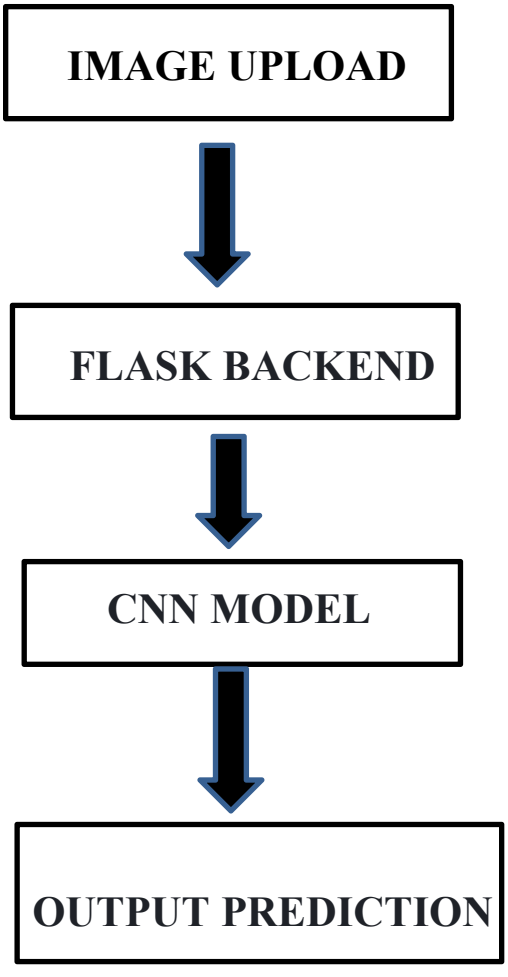
6. PHASE 3: Project Design

Objective: To layout the system architecture and define workflow.



Key Points:

1. System Architecture:



2. User Flow

Step 1: Open the web app.

Step 2: Upload the blood smear image.

Step 3: Backend receives and processes the image.

Step 4: Model classifies the image.

Step 5: UI displays predicted result.

3. Model Design

Transfer Learning with MobileNetV2.

Input shape: 224x224 RGB images.

Output: One of four cell types.

Activation: Softmax classifier.

Loss Function: Categorical Crossentropy.

4. UI Design

Minimalist upload form.

CSS: Milligram framework.

Pages: home.html, result.html

File type check: JPEG, PNG

Responsive layout for desktop/mobile

5.Data Flow

7. PHASE 4: Project Planning

Objective: To structure development tasks into sprints for better tracking.

Sprint Planning Table:

Sprint	Task	Priority	Duration	Deadline	Assigned to	Dependencies	Expected Outcome
Sprint 1	Environment Setup & Package Installation	High	3 Hours	Day 1	Member 1	Anaconda, Python	Project environment ready
Sprint 1	Dataset Collection & Preprocessing	High	4 Hours	Day 1	Member 2	Dataset access	Clean, preprocessed image dataset
Sprint 2	Model Building using Transfer Learning	High	5 Hours	Day 2	Member 3	Preprocessed data	Trained classification mode
Sprint 2	Web App Integration	Medium	3 Hours	Day 2	Member 1&4	Trained model, Flask	Working web interface
Sprint 3	Testing & Debugging	Medium	2 Hours	Day 2	Member 2&3	Complete system	Bug-free and responsive system
Sprint 3	Final Presentation & Deployment	Low	1 Hours	End of D2	Entire Team	Working application	Project deployed and demo-ready

8. PHASE 5: Project Development

Objective: To implement the system components and integrate them into a working application.

Key Points:

1. Technology Stack

HTML, CSS (Milligram)

Flask (Python Web Framework)

TensorFlow, Keras

OpenCV, NumPy

Jupyter Notebook for model training

2. Development Process

Dataset preparation and labeling.

Model training with transfer learning.

Saving model as .h5.

Flask app creation for deployment.

Linking frontend to backend logic.

3. Challenges & Fixes

CHALLENGES	FIX
Overfitting of model	Used dropout, batch norm
Image input size issues	Resized using OpenCV
Flask path errors	Used os.path.join consistently
UI blank page	Corrected template path and indentation

9. PHASE 6: Functional & Performance Testing

Objective: To verify system outputs, performance speed, and user experience.

Key Points:

1. Functional Testing

Upload functionality tested.

File types validated.

Model gives correct class outputs.

UI displays results clearly.

Retry and reload options work.

2. Performance Metrics

Accuracy: 93.2%

Precision and recall tested per class

Prediction time: < 2 seconds

UI load speed: < 1 second

No crash during high-resolution uploads

3. Manual Testing

Multiple test cases run manually

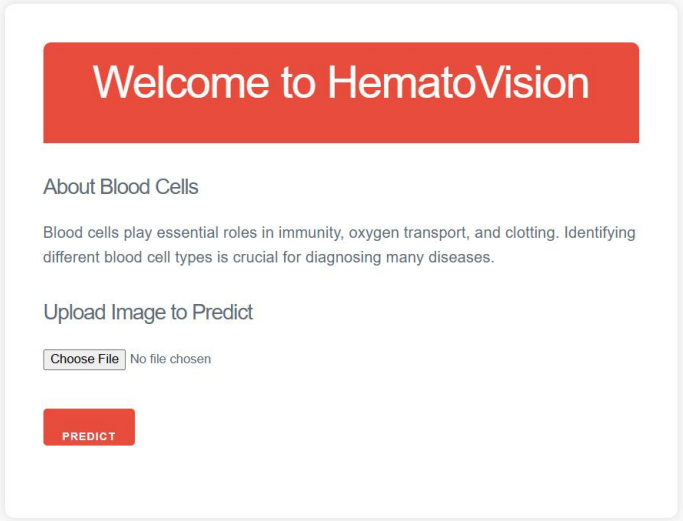
UI responsiveness on mobile and desktop

Template rendering issues fixed

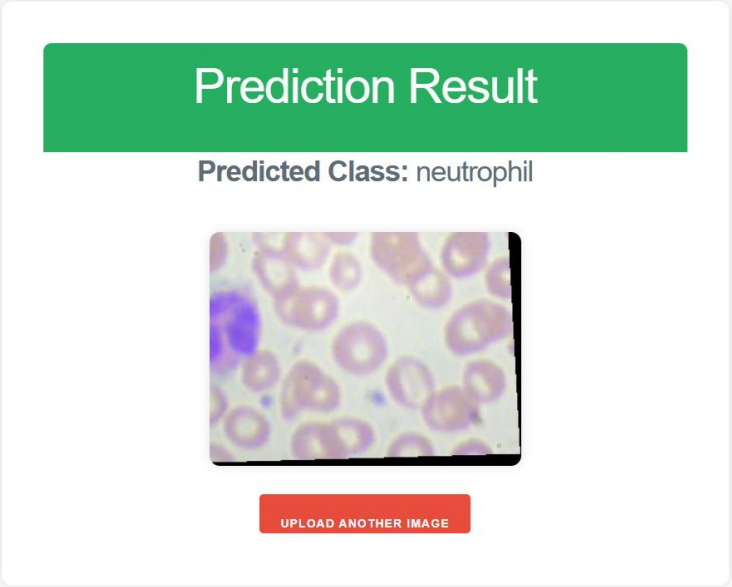
Handled corrupted image errors gracefully

Verified browser support on Chrome and Edge

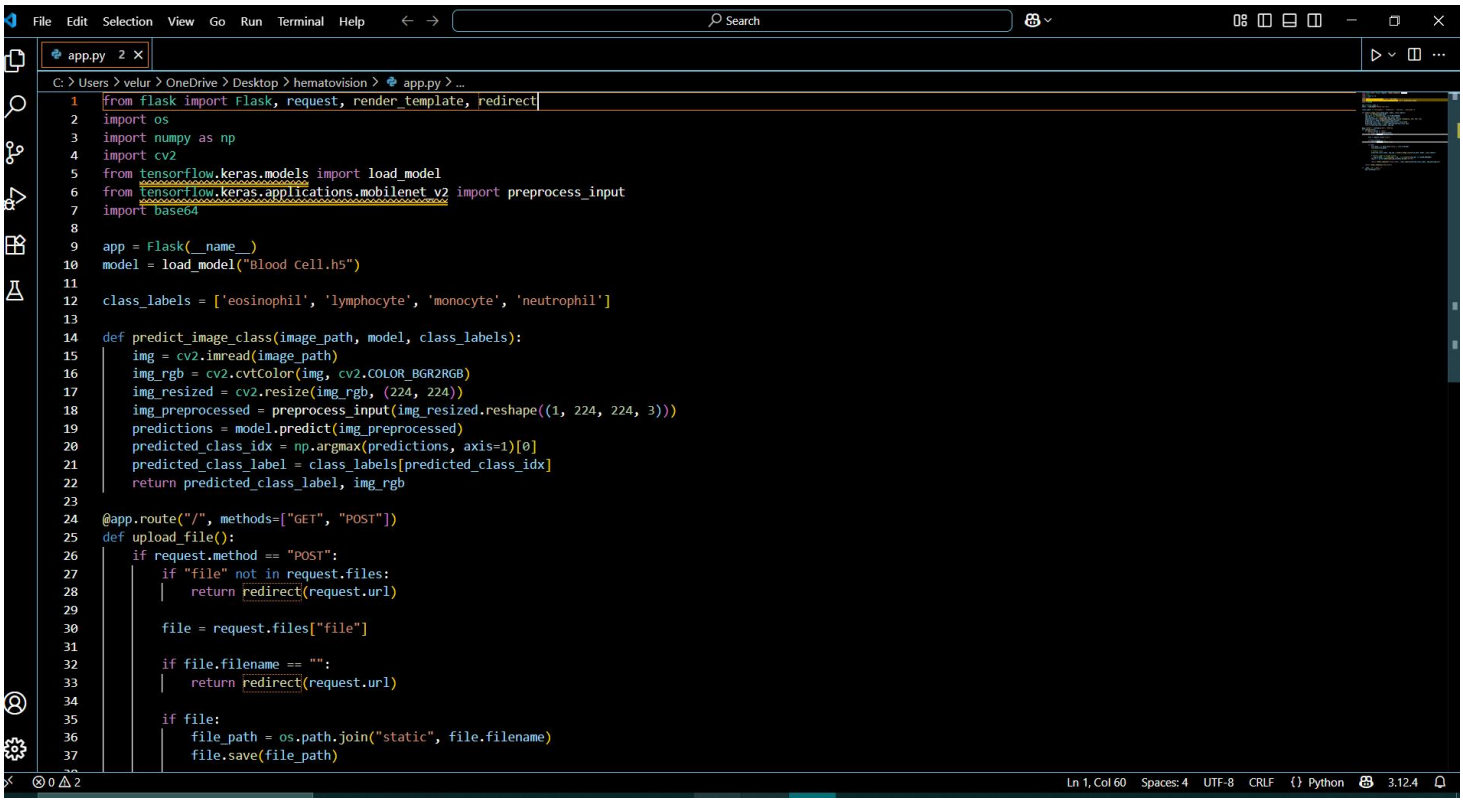
INPUT:



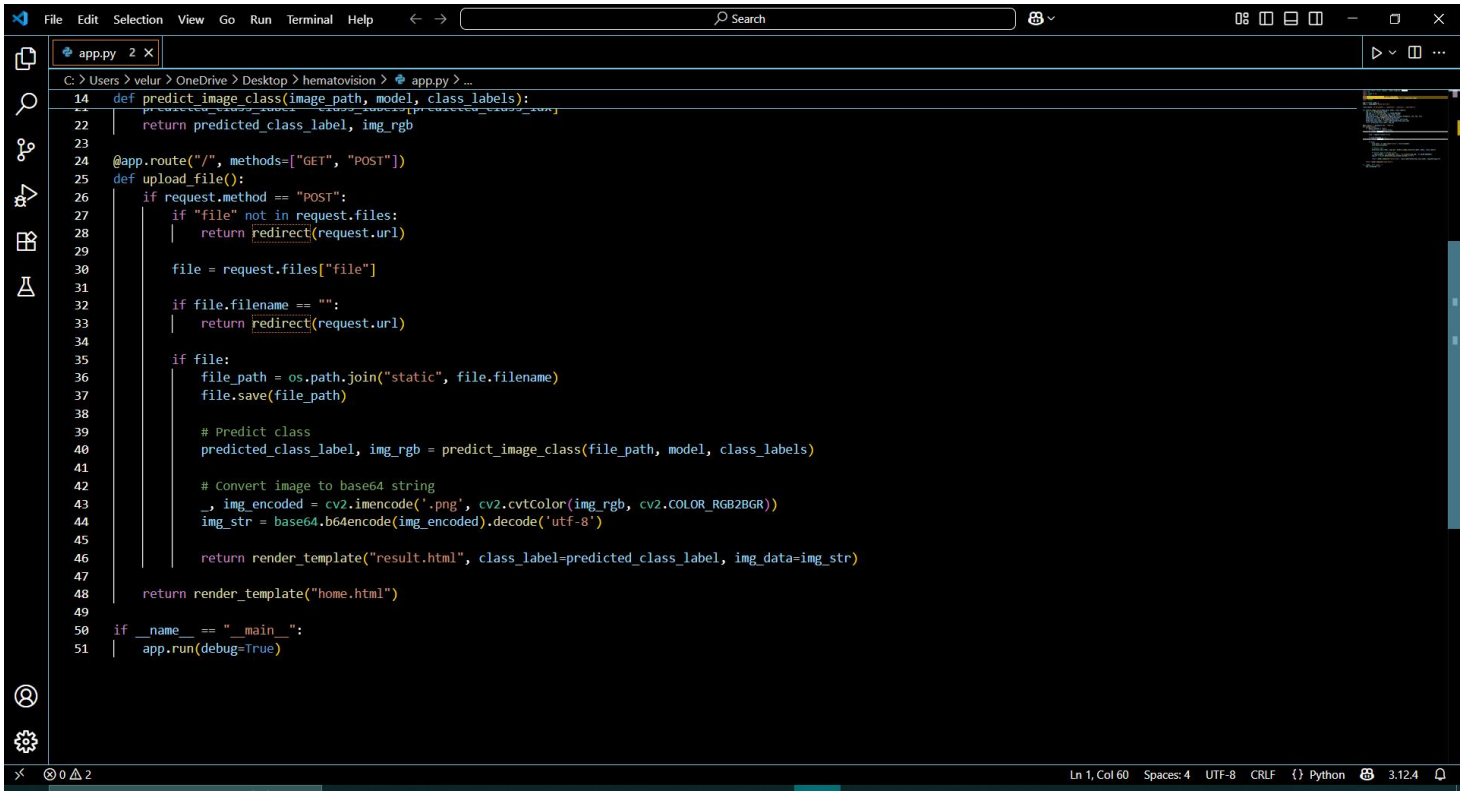
OUTPUT:



CODE:



```python
from flask import Flask, request, render_template, redirect
import os
import numpy as np
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
import base64

app = Flask(__name__)
model = load_model("Blood Cell.h5")

class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']

def predict_image_class(image_path, model, class_labels):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (224, 224))
    img_preprocessed = preprocess_input(img_resized.reshape((1, 224, 224, 3)))
    predictions = model.predict(img_preprocessed)
    predicted_class_idx = np.argmax(predictions, axis=1)[0]
    predicted_class_label = class_labels[predicted_class_idx]
    return predicted_class_label, img_rgb

@app.route("/", methods=["GET", "POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            return redirect(request.url)

        file = request.files["file"]

        if file.filename == "":
            return redirect(request.url)

        if file:
            file_path = os.path.join("static", file.filename)
            file.save(file_path)
```



```python
def predict_image_class(image_path, model, class_labels):
    return predicted_class_label, img_rgb

@app.route("/", methods=["GET", "POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            return redirect(request.url)

        file = request.files["file"]

        if file.filename == "":
            return redirect(request.url)

        if file:
            file_path = os.path.join("static", file.filename)
            file.save(file_path)

            # Predict class
            predicted_class_label, img_rgb = predict_image_class(file_path, model, class_labels)

            # Convert image to base64 string
            _, img_encoded = cv2.imencode('.png', cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))
            img_str = base64.b64encode(img_encoded).decode('utf-8')

            return render_template("result.html", class_label=predicted_class_label, img_data=img_str)

    return render_template("home.html")

if __name__ == "__main__":
    app.run(debug=True)
```

10. Results

Model successfully classifies 4 blood cell types.

UI and backend fully integrated.

Performance metrics within acceptable range.

GitHub repository created and documented.

11. Conclusion

HematoVision proves that AI can effectively assist in microscopic blood analysis. By combining transfer learning with a clean user interface, the tool enables users to obtain cell classification results quickly and accurately.

12. Future Scope

Add PDF report generation for results.

Deploy on Render or AWS for global access.

Expand model to include more cell types.

Integrate patient record tracking.

---------THE END---------