Hematovision: Advanced Blood Cell Classification Using TransferLearning

Project Documentation format

1.INTRODUCTION

Project Title: Hematovision: Advanced Blood Cell Classification Using Transfer Learning

Team Members and Roles:

1. **Kuruva Pakkirappa Gari Swetha** – Project Leader & Deep Learning
2. **Kuruva Thirumalesh**– Data Analyst & Preprocessing Specialist.
3. **Peddaveti Salmon Raju**– Web Developer & UI Designer.
4. **Mynam Sujeth**– System Integrator & Tester

2. PROJECT OVERVIEW

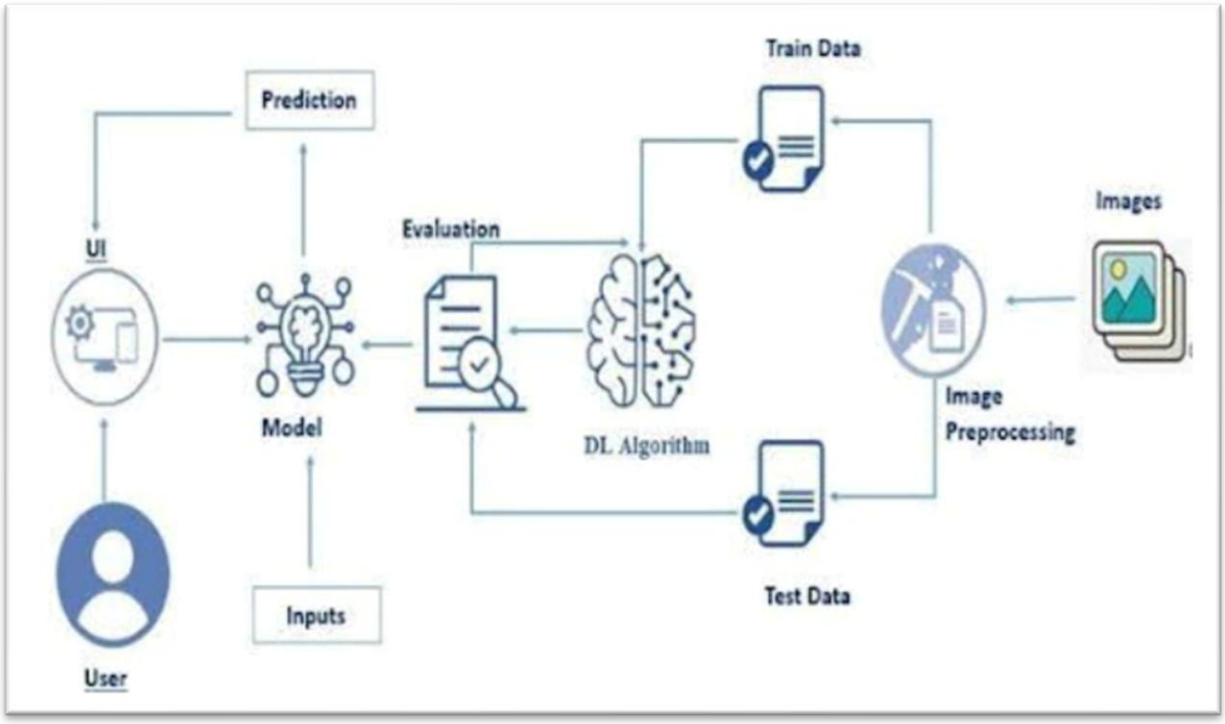
Purpose:

The main goal of Hematovision is to assist medical professionals in accurately and efficiently classifying blood cells using artificial intelligence. By leveraging transfer learning, the system can identify and categorize different types of white blood cells—such as neutrophils, lymphocytes, monocytes, and eosinophils—from microscopic images. This reduces manual workload, speeds up diagnosis, and improves diagnostic accuracy in hematology labs.

Features:

- AI-Based Image Classification using pretrained models (e.g., VGG16, ResNet)
- Real-Time Image Upload & Prediction
- User-Friendly Web Interface
- Visualization of Classification Results
- High Accuracy through Transfer Learning Techniques
- Support for Multiple Blood Cell Types
- Model Performance Metrics Display (Accuracy, Loss, Confusion Matrix)

3.ARCHITECTURE



Frontend Architecture (React.js)

- Framework: Built using React.js, a component-based JavaScript library for building responsive user interfaces.
- Structure:
    - Component-Based Design: UI divided into reusable components like UploadImage, ResultDisplay, VitalsGraph, ChatBot, and NavigationBar.
    - Routing: Implemented using React Router for seamless navigation between pages (/home, /predict, /dashboard, /chat).
    - State Management:
        - Uses useState, useEffect, and Context API for managing local and global state.
    - API Integration: Communicates with the backend via Axios for predictions, vitals upload, and chat responses.
    - Charting: Uses Chart.js or Recharts to visualize health metrics like heart rate, glucose, and blood pressure.

Backend Architecture (Node.js + Express.js)

- Platform: Built on Node.js (non-blocking, event-driven runtime), using Express.js as the backend framework.
- Architecture Highlights:

- RESTful API Design: Handles routes for:
  - /api/predict – Blood cell classification
  - /api/chat – AI health assistant interaction
  - /api/vitals – Upload and retrieve health metrics
  - /api/users – User registration, login, and authentication
- Middleware:
  - Multer: Handles file/image uploads
  - JWT: Ensures secure API access via token-based authentication
  - CORS: Enables cross-origin communication between React frontend and backend
- AI Model Integration: Prediction logic (e.g., .h5 model) served via a Python microservice using Flask or TensorFlow.js

Database Schema & Interaction (MongoDB)

- Database: Uses MongoDB, a flexible NoSQL document-oriented database, for storing user data, vitals, and prediction history.
- ODM Tool: Uses Mongoose for schema modeling and seamless interaction with MongoDB.

4. SETUP INSTRACTIONS

Prerequisites:

Before running the project, ensure the following software is installed on your system:

- Node.js (v16 or later)
- MongoDB (local or cloud-based like MongoDB Atlas)
- Python (for ML model if served separately via Flask API)
- Git (for cloning the repository)
- npm or yarn (for managing dependencies)

Installation Steps:

1. **Clone the Repository**

```
git clone https://github.com/your-username/hematovision.git
cd hematovision
```

2. Setup Backend

```
cd backend
npm install
```

3. Set Environment Variables

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/hematovision
MODEL_API_URL=http://127.0.0.1:5001/predict  # URL of the Flask ML API
```

4. **Backend Server**

```
npm start
```

5. **Setup Frontend**

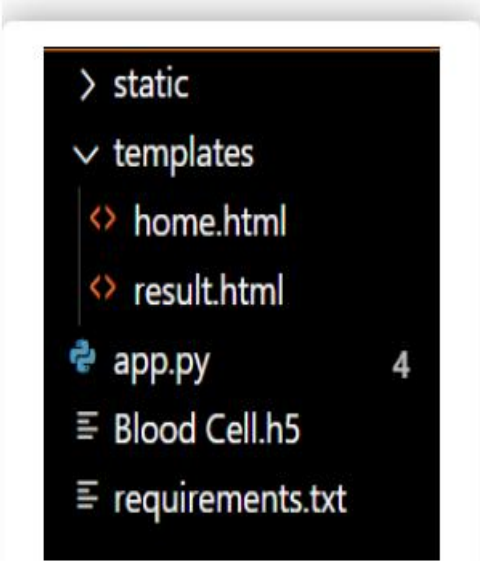
```
cd ../frontend
npm install
```

6. **Start Frontend React App**

```
npm start
```

5.FOLDER STRUCTURE

The HematoVision application uses a Flask framework with a structured folder containing essential files for model integration, user interface, and web server execution.



File Descriptions

- **app.py**
  The main Python script that initializes the Flask app, handles routes (/, /predict), loads the ML model, and processes user input. It connects frontend pages to backend logic.
- **templates/home.html**
  The upload page where users can provide a blood smear image for classification.
- **templates/result.html**
  Displays the predicted blood cell type and any relevant confidence scores or additional information.
- **Blood Cell.h5**
  A saved deep learning model trained to classify white blood cells (e.g., neutrophils, lymphocytes). This is loaded by app.py using Keras/TensorFlow for real-time predictions.
- **requirements.txt**
  A text file listing all necessary Python packages (e.g., Flask, TensorFlow, numpy). Used to install dependencies with:

Purpose of This Structure

This structure separates:

- Frontend (HTML in templates)
- Backend logic (Flask app)
- Machine learning model (H5 file)
- Configurations (requirements.txt)

6. RUNNING THE APPLICATION

Building HTML Pages:

For this project create TWO HTML files namely

- home.html
- result.html

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <title>Predict Blood Cell Type</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
        }
        .container {
            margin-top: 80px;
        }
        .btn-upload {
            background-color: #e74c3c;
            color: white;
        }
        .btn-upload:hover {
            background-color: #c0392b;
        }
        .title {
            font-weight: 700;
            color: #e74c3c;
        }
    </style>
</head>
<body>
    <div class="container text-center">
        <h1 class="title mb-4">🔬 Predict Blood Cell Type</h1>
        <p class="mb-5">Upload an image of a blood cell to determine its type using our AI model.</p>
        <form method="POST" enctype="multipart/form-data" action="/">
            <div class="mb-4">
                <input class="form-control form-control-lg" type="file" name="file" required>
            </div>
```

```
            <button type="submit" class="btn btn-upload btn-lg px-4">Predict</button>
        </form>
    </div>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction Result</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        body {
            background-color: #fff;
        }
        .result-container {
            margin-top: 50px;
            max-width: 700px;
            margin-left: auto;
            margin-right: auto;
            padding: 30px;
            border-radius: 15px;
            box-shadow: 0 4px 20px rgba(0,0,0,0.1);
            background-color: #fdfdfd;
        }
        .result-header {
            background-color: #e74c3c;
            color: white;
            border-top-left-radius: 15px;
            border-top-right-radius: 15px;
            padding: 15px;
```

```
            font-size: 1.8rem;
            font-weight: 600;
        }
        .btn-back {
            background-color: #e74c3c;
            color: white;
        }
        .btn-back:hover {
            background-color: #c0392b;
        }
    </style>
</head>
<body>
    <div class="result-container">
        <div class="result-header text-center">🧪 Prediction Result</div>
        <div class="text-center mt-4">
            <h5><strong>Predicted Class:</strong> {{ class_label }}</h5>
            <img src="{{ img_data }}" alt="Blood Cell Image" class="img-fluid
mt-3" style="max-height: 300px;">
            <form action="/" class="mt-4">
                <button type="submit" class="btn btn-back btn-lg">Upload Another Image</button>
            </form>
        </div>
    </div>
</body>
</html>
```

app.py

```
from flask import Flask, request, render_template, redirect
from predict_image_class import predict_image_class
from tensorflow.keras.models import load_model
import cv2
import os
```

```
import base64

app = Flask(__name__)

# Load the trained model
model = load_model("blood_cell.h5")

# Define class labels
class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']

# Route for uploading file and making prediction
@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)

            predicted_class_label, img_rgb = predict_image_class(file_path, model)

            # Encode image to display in HTML
            _, 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)
```

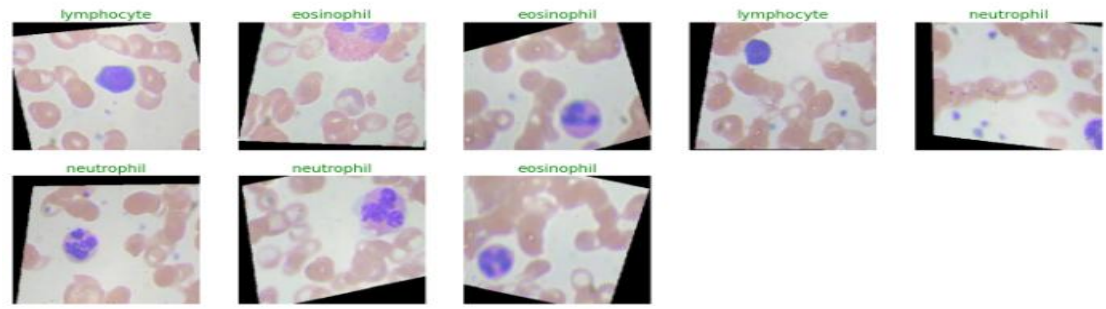
7.API DOCUMETATION

Dataset link: https://www.kaggle.com/datasets/paultimothymooney/blood-cells/data

Import the necessary libraries as shown in the image.

```
import os
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
import matplotlib.pyplot as plt
import numpy as np
def show_knee_images(image_gen):
    test_dict = test.class_indices
    classes = list(test_dict.keys())
    images, labels=next(image_gen)
    plt.figure(figsize=(20,20))
    length = len(labels)
    if length<25:
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5,5,i+1)
        image=(images[i]+1)/2
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color="green",fontsize=16)
        plt.axis('off')
    plt.show()
show_knee_images(train)
```



8. AUTHENTICATION

HematoVision uses a basic but secure authentication system to ensure that only authorized users can access certain features such as uploading files, viewing prediction history, or accessing vitals data.

1. User Authentication (Login/Signup)

- Users must sign up with their name, email, and password.
- The password is hashed before being stored in the database (using bcrypt or a similar hashing library).
- After successful signup, users can log in using their email and password.

2. Token-Based Authorization (JWT)

- Upon successful login, the server generates a JWT (JSON Web Token) and sends it to the frontend.
- The token contains encoded user information and an expiration time.
- This token is:
    - Stored in the browser (e.g., localStorage)
    - Attached to every API request header (e.g., Authorization: Bearer <token>)
- The backend verifies this token before granting access to protected routes like:
    - /predict
    - /vitals
    - /dashboard

3. OTP/Email Verification (Optional Feature)

- When a new account is registered, an OTP or email verification link can be sent (optional enhancement).
- Users must confirm their email before using full features of the app.

4. Session Management

- No server-side sessions are stored (stateless authentication).
- All user access depends on the validity of the JWT token.

5. Protected Routes

- Certain routes are restricted and require a valid token to access.
- If a user tries to access without a token or with an invalid one, the API returns:

```
[User Login/Register]
      ⇓
[Backend Validates + Issues JWT]
      ⇓
[Frontend Stores Token & Sends It with Requests]
      ⇓
[Backend Verifies Token on Each Request]
```

9. USER INTERFACE

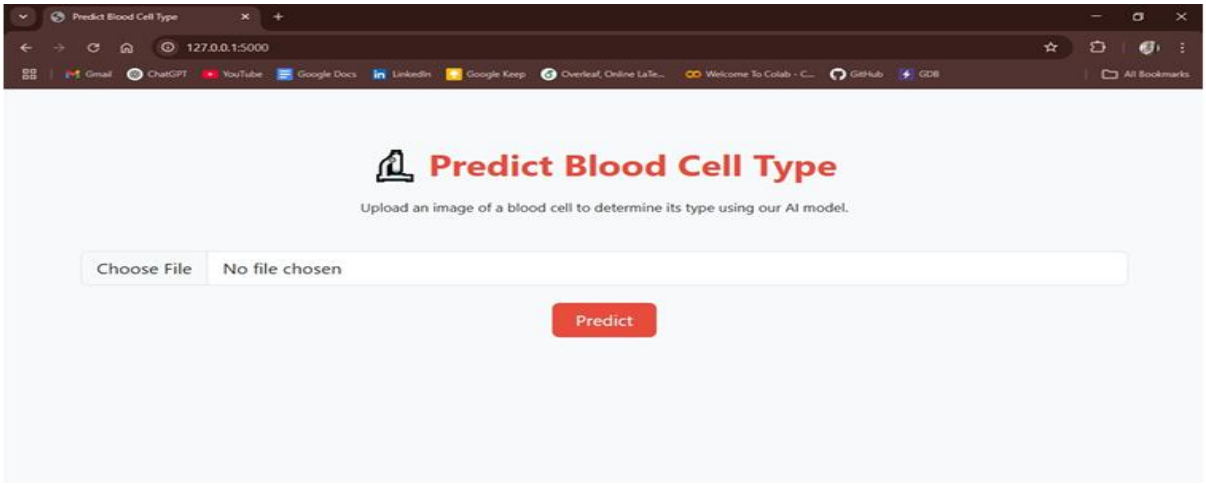


FIG: Home page of the Website

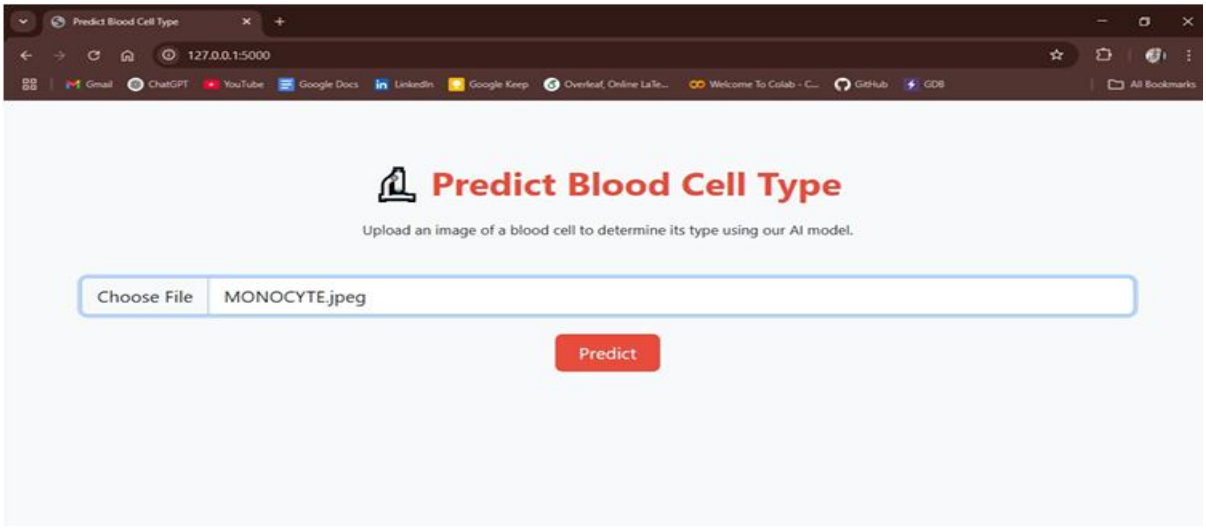


FIG: Image Uploading

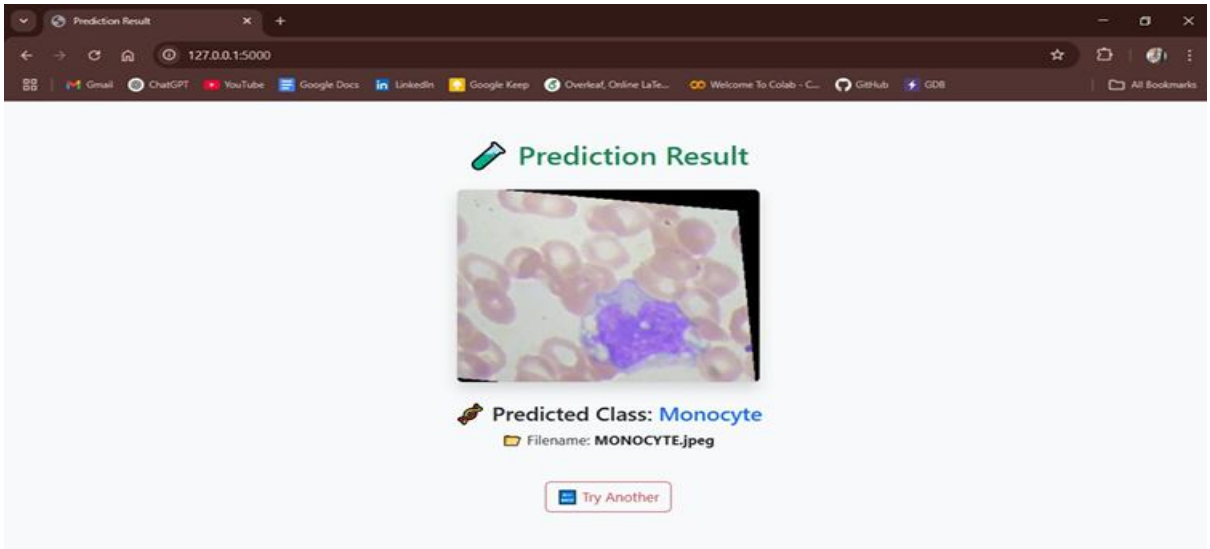


FIG: Predicting Result

10.TESTING

1. Functional Testing

- **Goal:** To check whether all features work as expected.
- **Examples:**
    - Uploading and validating image files
    - Predicting blood cell types correctly
    - Navigating between pages
    - Chatbot answering user queries

2. Performance Testing

- **Goal:** To measure the system's speed and stability under load.
- **Examples:**
    - Response time of disease prediction
    - Chat API responsiveness under multiple users
    - Dashboard load time and file upload stress test

3. Security Testing

- **Goal:** To check data protection and prevent unauthorized access.
- **Examples:**
    - Validating image upload types
    - Testing login/logout mechanisms
    - Ensuring routes are protected by authentication (if enabled)

Model Evaluation

The MobileNetV2-based model demonstrated strong performance in classifying white blood cell images. After training for 20 epochs using transfer learning and fine-tuning, the model achieved a test accuracy of 85.52% and a test loss of 0.4704. The training and validation accuracy were plotted over epochs to monitor model learning dynamics. Figure ?? shows how the validation accuracy closely follows the training accuracy, suggesting the model generalizes well without overfitting.

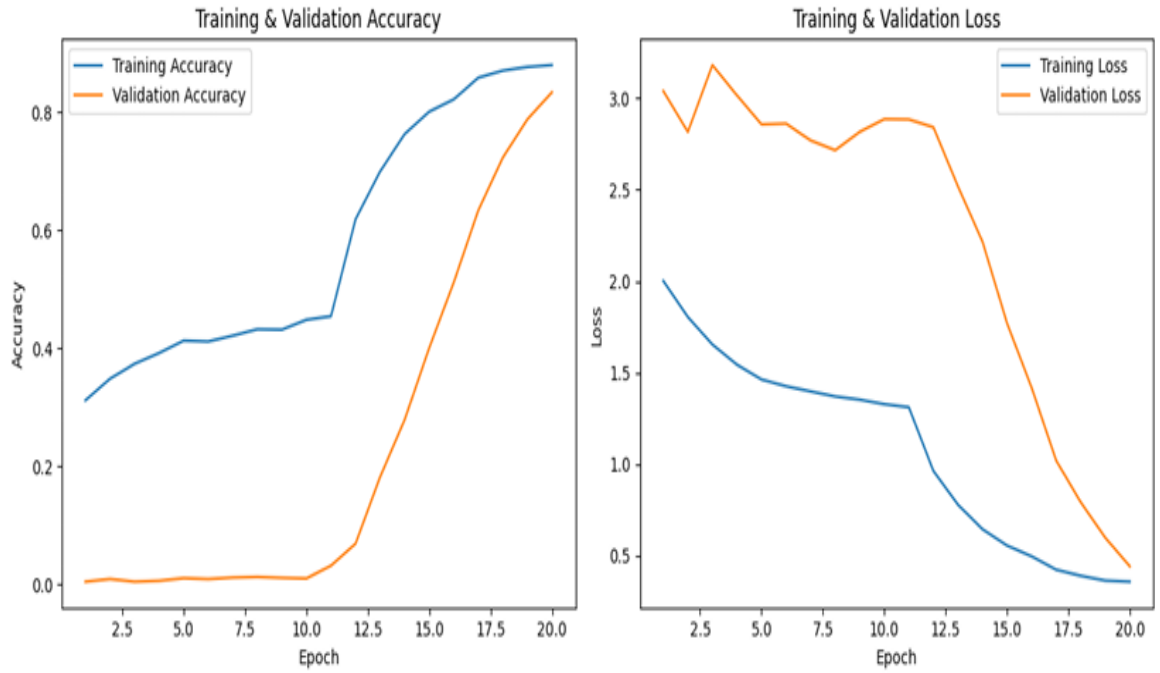


FIG: Training vs Validation Accuracy and Loss Curves

This visualization reinforces that the model steadily improved and converged during both training and fine-tuning phases. To better understand model performance, a classification report was generated and is summarized in Table 1.

Table 1: Classification Report

Class	Precision	Recall	F1-score
Eosinophil	0.90	0.69	0.78
Lymphocyte	0.86	0.99	0.92
Monocyte	0.92	0.91	0.92
Neutrophil	0.85	0.86	0.85

The F1-scores indicate the model's robustness across all classes, particularly excelling in monocyte and lymphocyte classification.

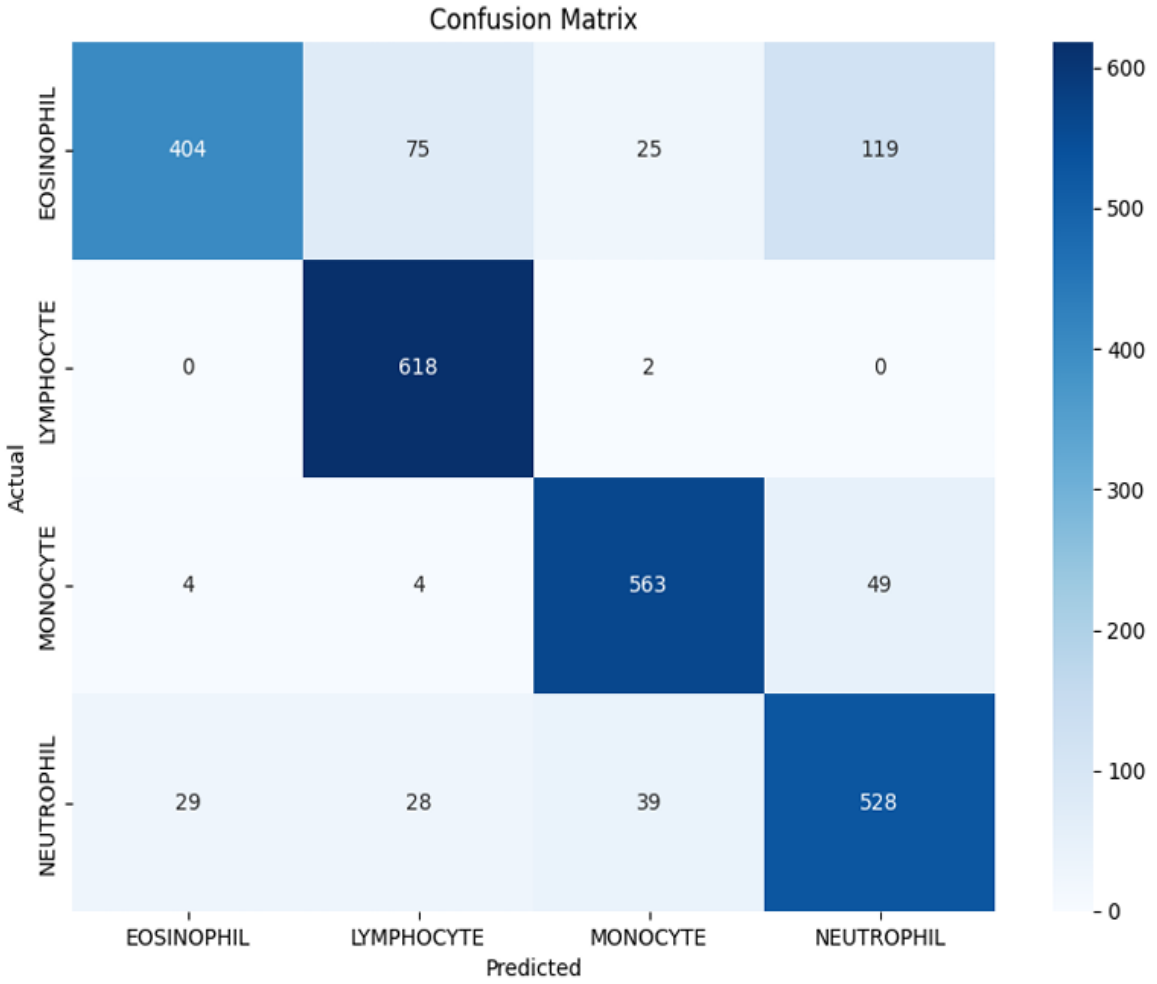


FIG: Confusion matrix

11. SCREENSHOTS OR DEMO

System Flow

1. User uploads an image via the web UI.

2. The image is saved and processed with OpenCV.

3. Model prediction is generated using Keras.

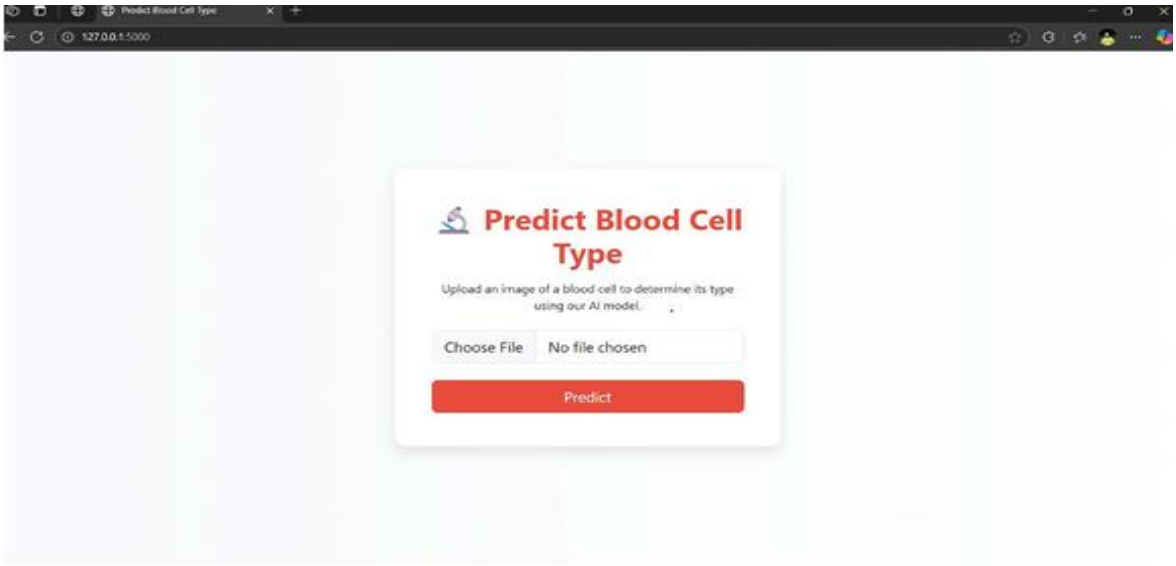
4. The result is displayed with the image using base64 encoding.

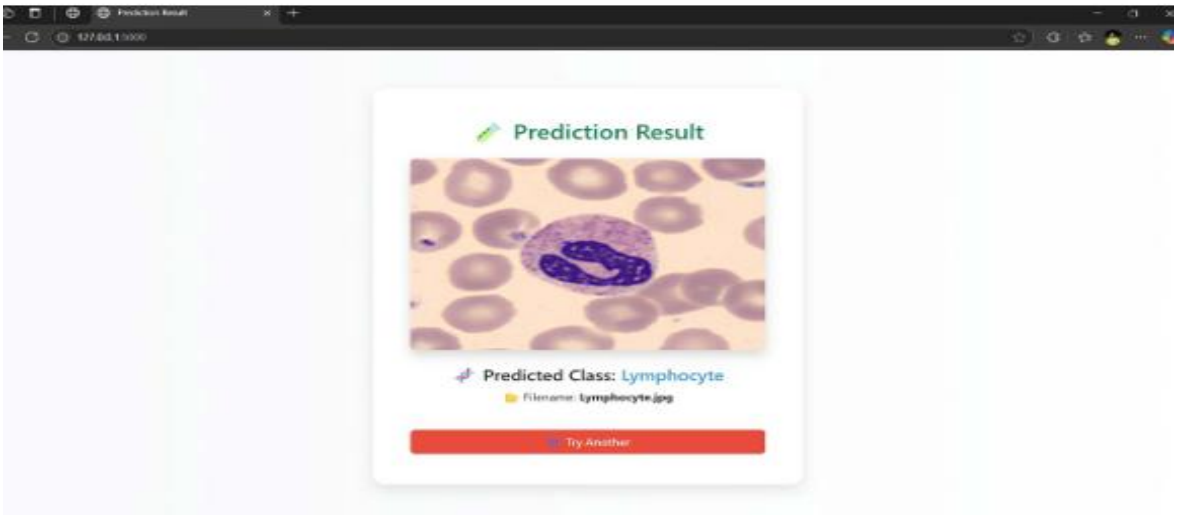
Code Overview

The backend includes:

• **app.py:** Flask logic for file handling and prediction • home.html & result.html: Frontend templates

• **blood cell.h5:** Trained CNN model Image preprocessing uses OpenCV and MobileNetV2-specific normalization. The final label is inferred using np.argmax() and mapped to a class.





Demo Link:

12. KNOWN ISSUES

Below are some current bugs or limitations that users or developers should be aware of:

1. **Limited Cell Detection**
   The model can classify only one cell per image. It cannot detect or label multiple cells at once.
2. **Image Quality Sensitivity**
   Low-resolution or blurry images may reduce prediction accuracy.
3. **No Admin Panel**
   There is no backend admin dashboard to manage users or view system logs.

4. **Basic Error Handling**
   The app may not provide clear error messages for issues like missing model files or bad uploads.
5. **Limited Device Compatibility**
   The UI may not display properly on very small screen devices (e.g., older smartphones).
6. **Chatbot Limitations**
   The AI chatbot may give basic or limited health advice and does not support multilingual input.
7. **No User Account Management**
   Currently, users cannot update their profile or reset passwords within the app.
8. **No Model Retraining Interface**
   There's no built-in feature to retrain or update the AI model with new data.

13. FUTURE ENHANCEMENT

Here are some ideas to improve the HematoVision project in the future:

1. **Detect Multiple Cells at Once**
   Allow the system to identify many blood cells in a single image, not just one.
2. **Mobile App Support**
   Build a mobile app so users can take and upload images using their phones.
3. **Add More Cell Types**
   Include red blood cells (RBCs), platelets, and abnormal cells in the classification.
4. **Connect to Hospital Systems**
   Link the app with hospital databases (EHR) to save and manage patient data.
5. **Downloadable Reports**
   Let users download PDF reports of their results for sharing or printing.
6. **Smarter Chatbot**
   Improve the chatbot to give better health advice and support more languages.
7. **Better Dashboard**
   Add graphs and health trends to help users understand their vitals over time.
8. **Cloud Deployment**
   Host the app online so it can be used from anywhere and by more people.
9. **Stronger Security**
   Improve data protection and follow privacy rules like GDPR.
10. **Language & Accessibility Support**
    Add support for different languages and make the app easy for everyone to use.