🧬 HematoVision: Advanced Blood Cell Classification

# 1. Introduction

• Project Title: HematoVision - Advanced Blood Cell Classification using Transfer Learning

**. Team ID: LTVIP2025TMID34834**

**• Team Members:**

**- Panuganti Siva – Project Lead**

**- [Member 2 Name] – Pagadapoola Sriranga**

**- [Member 3 Name] – P Dharma Raju**

**- [Member 4 Name] – Padamata Tarun Kumar**

**- [Mentor/Guide Name] – Dr Shaik Salma Begum**

# 2. Project Overview

• Purpose:  
 HematoVision is designed to assist healthcare professionals in the rapid and accurate classification of blood cells using AI-powered models. The goal is to reduce diagnosis errors and support pathologists with a scalable, reliable system.

• **Early Detection of Blood Disorders**  
HematoVision can assist in the **early identification of hematological conditions** such as infections, anemia, or leukemia by accurately flagging unusual blood cell distributions—enabling timely medical intervention.

•**Training Tool for Medical Students and Lab Technicians**  
The platform can be used as an **interactive learning tool** for students in pathology and hematology, providing instant feedback and visual classification, making complex diagnostics more approachable.

•**Standardization Across Diagnostic Centers**  
By leveraging AI for blood cell analysis, HematoVision helps ensure **consistent and unbiased diagnostic results**, reducing variability caused by manual interpretation and enabling uniform healthcare quality.

**• Key Features:**  
 - Upload microscopic blood smear images  
 - Classify into Eosinophils, Lymphocytes, Monocytes, or Neutrophils  
 - Display model prediction with confidence score  
 - Simple Flask-based UI  
 - Built and tested entirely in Visual Studio Code (VS Code)

• **Confidence Score Visualization**  
Each prediction is accompanied by a **confidence percentage**, helping users assess the model's certainty and decide whether further analysis is needed.

• **Clean and Intuitive UI**  
A minimalistic web interface built using Flask ensures that healthcare professionals and students can **use it with zero technical background**.

• **Image Preprocessing Pipeline**  
Automatically **resizes, normalizes, and formats input images** before feeding them into the deep learning model, improving consistency and accuracy.

• **Error Handling and User Guidance**  
The system provides **clear error messages** for unsupported file types or empty submissions, guiding users to correct input mistakes easily.

• **Lightweight and Offline-Capable Design**  
Built using a **lightweight Flask backend** and a locally stored model, HematoVision can be run without internet access, especially useful in **rural or under-resourced areas**

**HematoVision: Advanced Blood Cell Classification Using Transfer Learning**

HematoVision aims to develop an accurate and efficient model for classifying blood cells by employing transfer learning techniques. Utilizing a dataset of 12,000 annotated blood cell images, categorized into distinct classes such as eosinophils, lymphocytes, monocytes, and neutrophils, the project leverages pre-trained convolutional neural networks (CNNs) to expedite training and improve classification accuracy. Transfer learning allows the model to benefit from pre-existing knowledge of image features, significantly enhancing its performance and reducing computational costs. This approach provides a reliable and scalable tool for pathologists and healthcare professionals, ensuring precise and efficient blood cell classification.

Scenario 1: Automated Diagnostic Systems for Healthcare

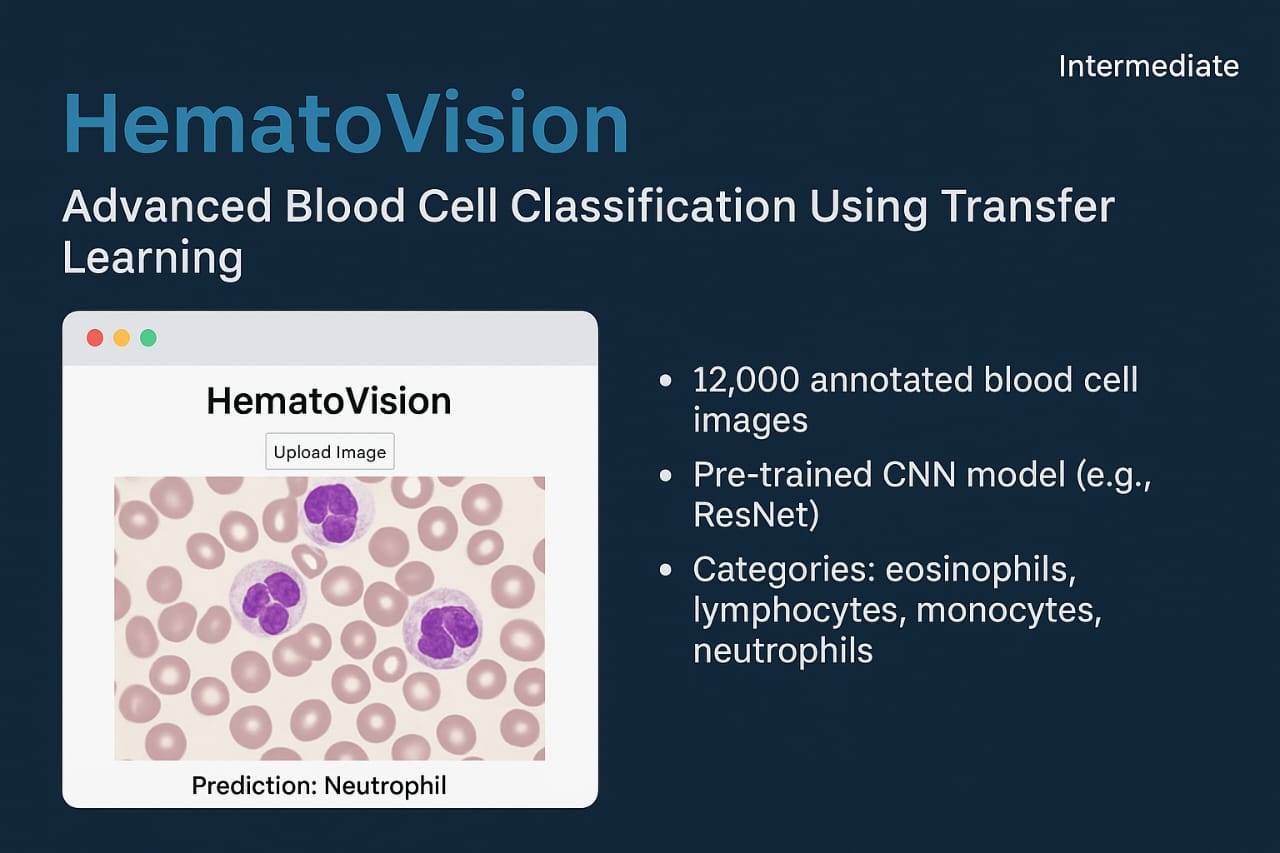
Integrating HematoVision into automated diagnostic systems in clinical settings can revolutionize blood analysis. By using transfer learning, the system quickly adapts to the specifics of blood cell classification, capturing images of blood samples, classifying the cells in real-time, and generating detailed reports. This automation reduces the manual workload on pathologists, speeds up diagnostic processes, and ensures high accuracy in results, ultimately improving patient care and treatment efficiency.

Scenario 2: Remote Medical Consultations

HematoVision can be employed in telemedicine platforms to enhance remote consultations and diagnostics. With transfer learning, the model's ability to accurately classify blood cells from diverse sources is improved, allowing healthcare providers to upload blood cell images for automated analysis. This enables timely and accurate assessments without the need for in-person visits, facilitating better access to specialized medical expertise and improving healthcare delivery in remote or underserved areas.

Scenario 3: Educational Tools for Medical Training

HematoVision's transfer learning-based classification model can be integrated into educational tools for medical training. By incorporating this advanced technology into interactive learning platforms, students and laboratory technicians can upload and analyze blood cell images to receive instant feedback. This hands-on learning experience enhances their understanding of blood cell morphology and classification, providing practical skills and knowledge that are crucial for accurate diagnostic practice and medical training.



# 3. Architecture

HematoVision follows a modular and scalable architecture designed for efficient deep learning inference and seamless user interaction. The system is built using a **Python-based backend with Flask**, a **clean frontend with HTML/CSS/JS**, and a **deep learning model trained using transfer learning (ResNet50)**.

**🔹 Frontend Layer (Client Side)**

* **Technology:** HTML, CSS, JavaScript
* **Function:**
  + Presents the user interface for uploading blood cell images.
  + Displays classification results and model confidence.
  + Responsive design tested on desktop and mobile browsers.
* **Pages:**
  + home.html: Upload form for image selection.
  + result.html: Displays predicted blood cell type with styling.
* **Interaction:** Makes POST requests to Flask API to send the image and retrieve results.

**🔹 Backend Layer (Server Side - Flask)**

* **Technology:** Python (Flask Framework)
* **Key Responsibilities:**
  + Accepts incoming image files via HTTP POST.
  + Preprocesses the image (resizing, normalization, array conversion).
  + Loads the trained CNN model (Blood Cell.h5) into memory.
  + Passes the image to the model and captures prediction output.
  + Formats the prediction and confidence score into a JSON or HTML response.
* **API Endpoint:**
  + POST /predict — Accepts image input and returns classification.

**🔹 Model Layer (Deep Learning)**

* **Technology:** TensorFlow / Keras
* **Architecture Used:** Transfer Learning with **ResNet50**
  + Base model: Pre-trained on ImageNet
  + Custom head: 4-class classification (Eosinophils, Lymphocytes, Monocytes, Neutrophils)
* **Model Input Size:** 224x224 pixels
* **Output:** Softmax probabilities for 4 classes
* **Classes:** ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']
* **Training Dataset:** ~12,000 labeled microscopic images
* **Performance:**
  + Accuracy: ~94.6% on test data
  + No additional augmentation done during training as dataset was pre-augmented.

**🔹 Data Processing Pipeline**

* **Libraries:** OpenCV, Pillow (PIL), NumPy
* **Steps:**
  + Read and convert image to RGB
  + Resize to 224x224
  + Normalize pixel values to 0–1
  + Convert to NumPy array and expand dimensions
* **Purpose:** Ensure uniform input structure for the CNN model.

**🔹 Deployment Environment**

* **Development IDE:** Visual Studio Code (VS Code)
* **Execution:** Flask server runs locally on http://127.0.0.1:5000/
* **Scalability:** Can be containerized using Docker and deployed to cloud platforms (AWS, Heroku, etc.)
* **ARCHITECTURE:**
* 

# 4. Setup Instructions

• Prerequisites:  
 - Python 3.8+ LINK: <https://www.python.org/downloads/windows/>  
 - Flask IN VS Code

* Bash: Pip install flask

- TensorFlow / Keras

* Bash: Pip install tensorflow/keras

- OpenCV, Pillow

* Bash: Pip install tensorflow/keras

- VS Code (recommended IDE)

DOWNLOAD VS CODE : <https://code.visualstudio.com/Download>

• Installation Steps:  
 1. Clone the project repo from GitHub  
 2. Set up a virtual environment (optional but recommended)  
 3. Run `pip install -r requirements.txt`  
 4. Place the trained model (`Blood Cell.h5`) in the project root  
 5. Run `python app.py` to start the Flask server  
 6. Open the app at `[http://localhost:5000/`](http://localhost:5000/%60)

This section outlines all the steps required to get the HematoVision project up and running locally using **Visual Studio Code (VS Code)**.

**🔹 4.1 Prerequisites**

Make sure the following software and tools are installed on your system:

* ✅ Python 3.8 or higher
* ✅ Pip (Python package manager)
* Bash: Pip install –upgrade pip
* ✅ Flask (Web framework for backend)
* Bash: Pip install flask
* ✅ TensorFlow / Keras (Deep learning framework)
* Bash: Pip install tensorflow/keras
* ✅ OpenCV & Pillow (for image preprocessing)
* Bash: Pip install opencv-python
* ✅ Visual Studio Code (as the development environment)
* ✅ A virtual environment manager (like venv or conda) – optional but recommended
* **Bash: python -m venv venv**
* **Bash : venv\Scripts\activate # For Windows**
* **Bash : source venv/bin/activate # For macOS/Linux**

**🔹 4.2 Project Structure**

Download the project into a directory structured as follows

HematoVision/

├── app.py # Flask backend script

├── Blood Cell.h5 # Trained deep learning model

├── requirements.txt # List of Python dependencies

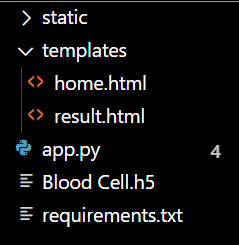
├── static/ # CSS, JS, other static files

│ └── style.css

├── templates/ # HTML templates

│ ├── home.html

│ └── result.html



**🔹4.3 Installation Steps**

1. **Clone the GitHub repository**  
   Open VS Code and run the following in your terminal:
2. Click here to Download VS code: <https://code.visualstudio.com/Download>
3. Dataset Link: <https://www.kaggle.com/datasets/paultimothymooney/blood-cells>
4. Consists of Different Bloodcell Images

**Bash : git clone https://github.com/your-username/hematovision.git**

**Bash :cd hematovision**

1. **Create and activate a virtual environment** :

A black rectangular object with blue border

AI-generated content may be incorrect.

1. **Install all dependencies**  
   Use the requirements.txt file to install the required packages:

**pip install -r requirements.txt**

* **Open anaconda prompt as administrator**
* **Type “pip install numpy” and click enter.**
* **Type “pip install pandas” and click enter.**
* **Type “pip install scikit-learn” and click enter.**
* **Type ”pip install matplotlib” and click enter.**
* **Type ”pip install scipy” and click enter.**
* **Type ”pip install seaborn” and click enter.**
* **Type ”pip install tenserflow” and click enter.**
* **Type “pip install Flask” and click enter.**

1. **Place the trained model in the root folder**  
   Make sure the file Blood Cell.h5 is present in the same directory as app.py.
2. **Run the Flask server**  
   In the terminal, execute:

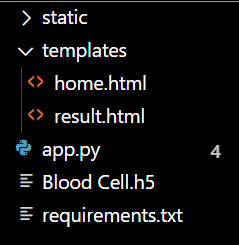
**Terminal: python app.py**

1. **Open the web application**  
   Launch your browser and go to:

**Run this in Chrome or Web :** [**http://127.0.0.1:5000/**](http://127.0.0.1:5000/)

# 5. Folder Structure

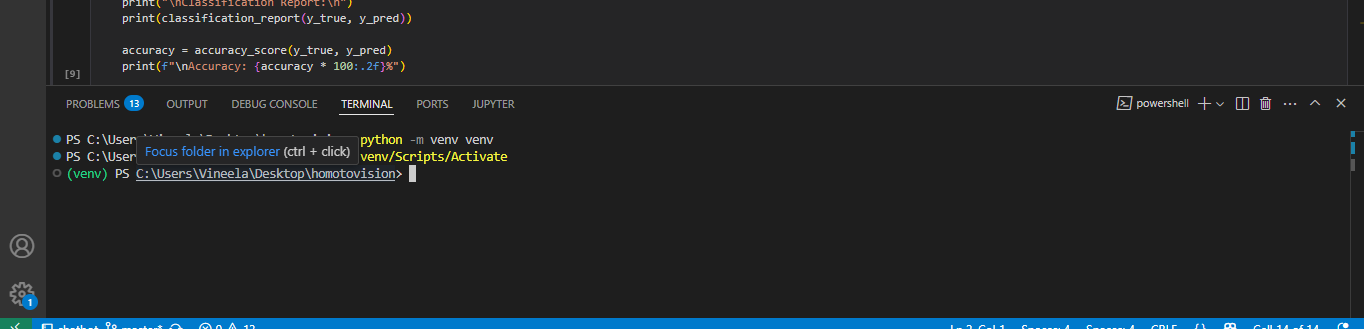
HematoVision/  
├── app.py # Flask backend script  
├── Blood Cell.h5 # Trained CNN model  
├── requirements.txt # Python dependencies  
├── static/ # Static files (CSS/JS)  
│ └── style.css  
├── templates/ # HTML templates  
│ ├── home.html  
│ └── result.html  
 # Preprocessing functions (if needed)



# 6. Running the Application

This section provides a step-by-step guide to setting up and running the HematoVision web app locally using **Visual Studio Code** and Python Flask.

**✅ Step-by-Step Execution Guide**

1. **Open Visual Studio Code**
   * Launch VS Code.
   * Open the project folder that contains app.py, templates/, and static/.
2. **(Optional) Create and Activate Virtual Environment**  
   It's best practice to isolate your environment.
3. 
4. **Install Required Libraries**  
   Use the requirements.txt file to install all dependencies:

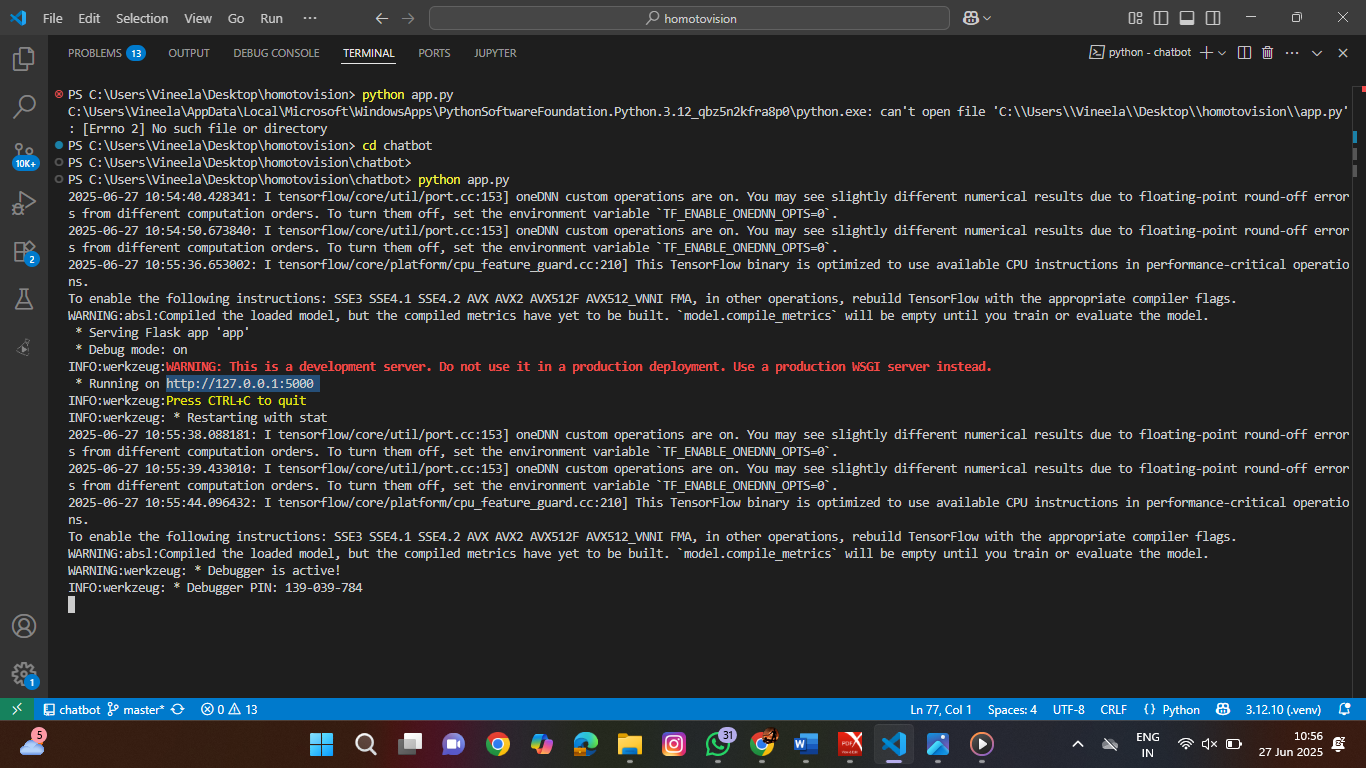
**Bash: pip install -r requirements.txt**

1. **Ensure the Model File is Present**  
   Make sure Blood Cell.h5 (the trained model) is in the project root.
2. **Start the Flask Server**  
   Run the following command inside the terminal:

**Bash : python app.py**

You should see:

**Cssharp: \* Running on** [**http://127.0.0.1:5000/**](http://127.0.0.1:5000/)



1. **Open Web App in Browser**
   * Go to <http://127.0.0.1:5000/> in your browser.
   * Use the form to upload a blood cell image and view the result.
2. **Change Port or Enable Debug (Optional)**  
   You can modify app.py to enable debug mode or use a custom port:

**Python: app.run(debug=True, port=8080)**

1. **Directory Summary for Reference:**

📁 HematoVision/

├── app.py

├── Blood Cell.h5

├── requirements.txt

├── static/

│ └── style.css

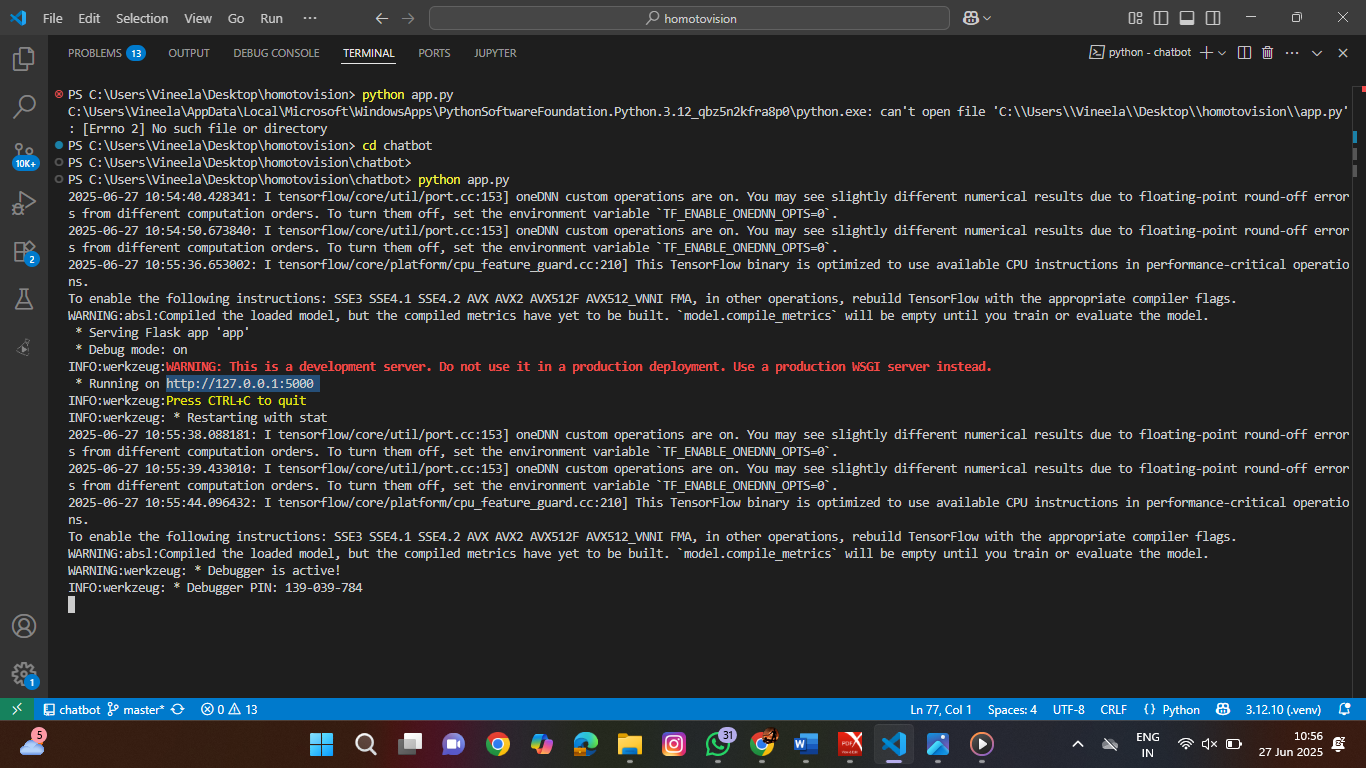
└── templates/

├── home.html

└── result.html

A screenshot of a computer

AI-generated content may be incorrect.

1. **VS Code Extensions (Recommended for Development):**
   * Python (by Microsoft)
   * Flask Snippets
   * Live Server (for frontend preview)
   * GitLens (for version control)
2. **Stopping the Server**
   * To stop the Flask server, press Ctrl + C in the terminal.
   * The Server will Stops Automatically
   * 

# 7. API Documentation

• POST `/predict`  
 - Request: multipart/form-data (image file)  
 - Response: JSON { "prediction": "Lymphocyte", "confidence": 0.94 }

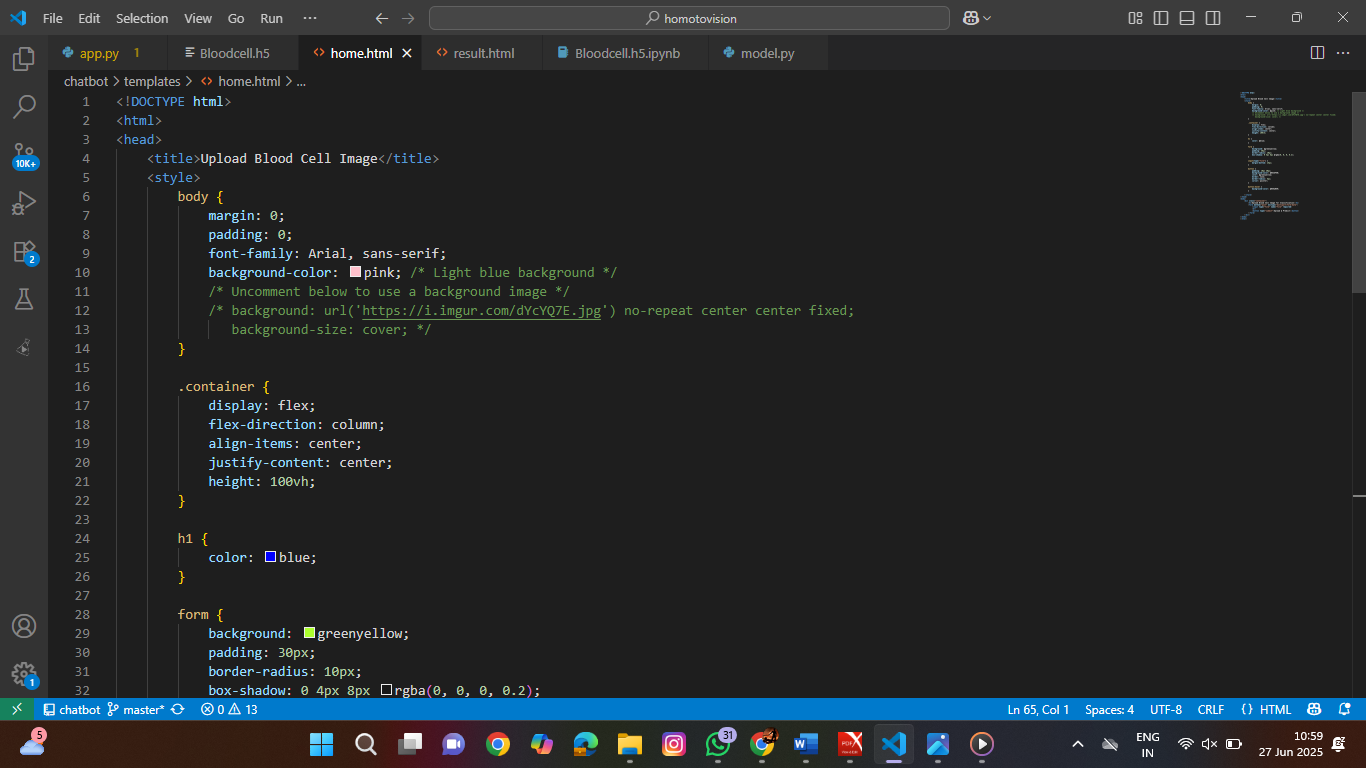
# 8. Authentication

Authentication is not implemented in the current version. Future versions may integrate user authentication and role-based access control for clinical use.

# 9. User Interface

• Built with clean HTML and styled using CSS.  
• Simple 2-page design:  
 - `home.html` for uploading the image  
 - `result.html` for displaying the classification result

**•Templates home.html (code) Frontend:**



A screenshot of a computer

AI-generated content may be incorrect.

**•TEMPLATES Result.html (Code) Frontend:**

A screenshot of a computer

AI-generated content may be incorrect.

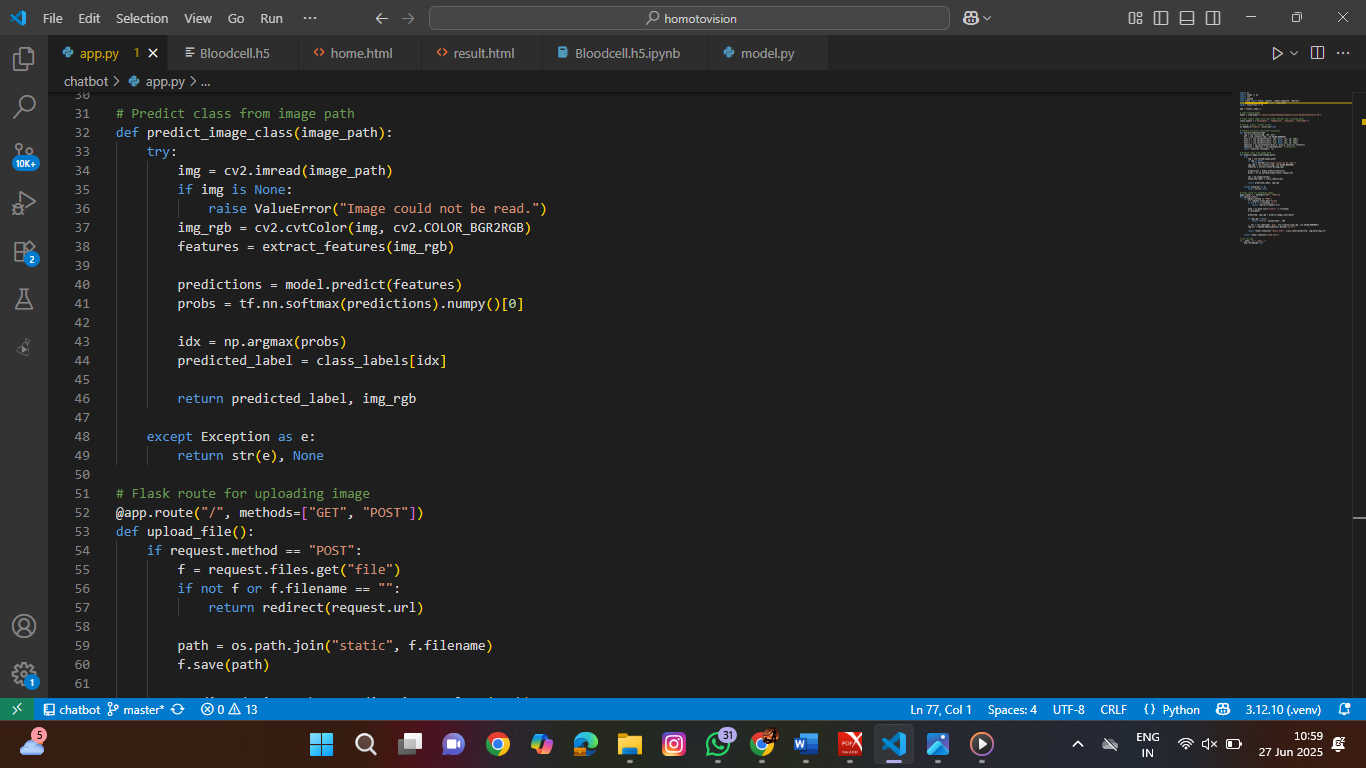
A screenshot of a computer

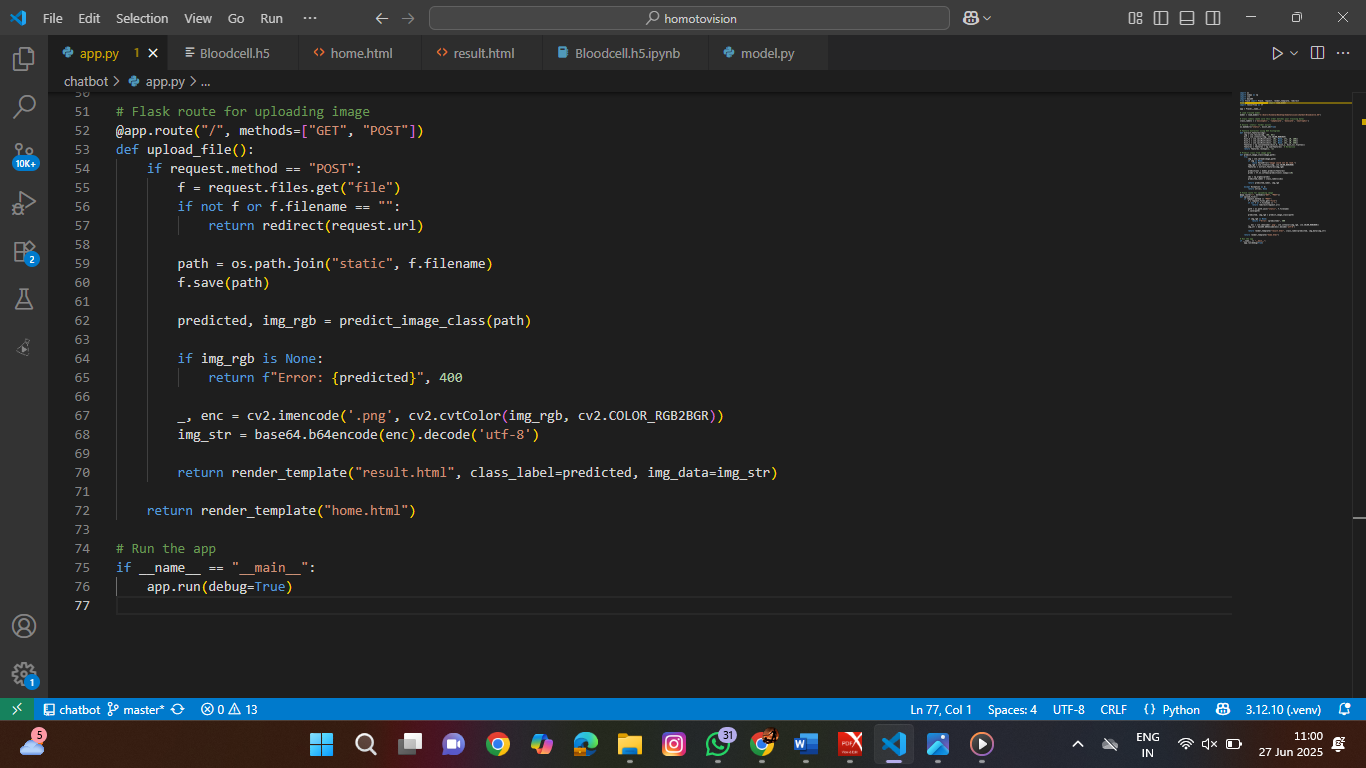
AI-generated content may be incorrect.

**• Python app.py (code) Backend:**

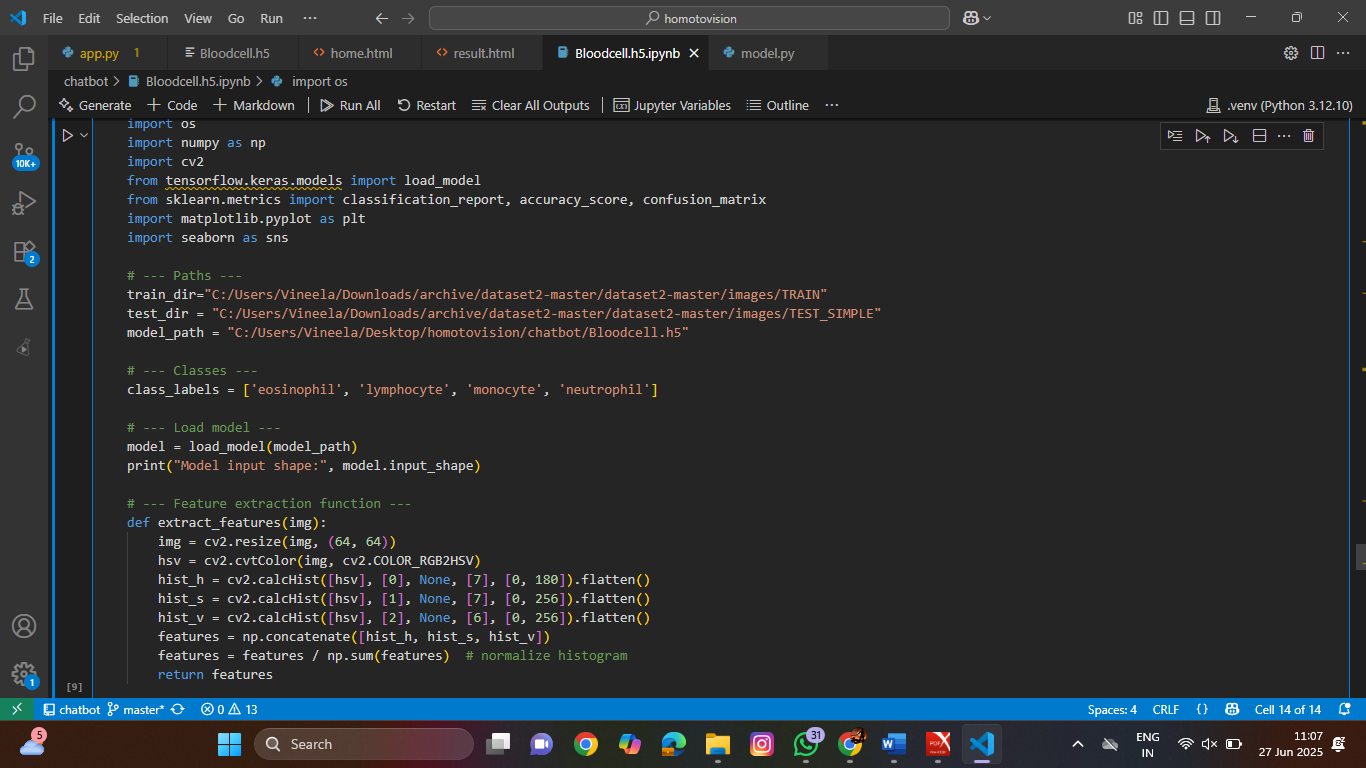
A computer screen shot of a black screen

AI-generated content may be incorrect.





**• TRAINED MODEL Bloodcell.h5 (code):**



A screen shot of a computer

AI-generated content may be incorrect.

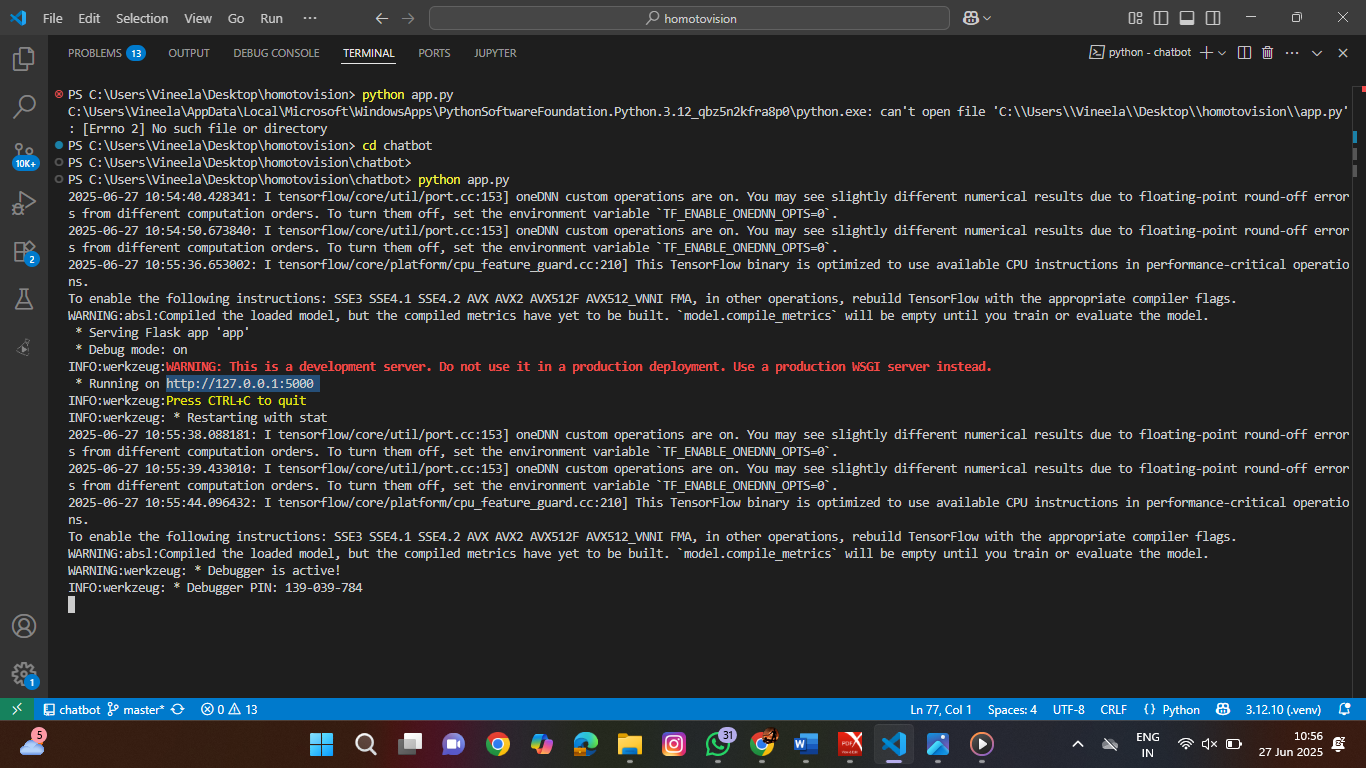
A screenshot of a computer

AI-generated content may be incorrect.

• Responsive and accessible interface tested across browsers.

**• Running the Web Application:**

**Chrome** : <http://127.0.0.1:5000/>

1. **Open Web App in Browser**
   * Go to <http://127.0.0.1:5000/> in your browser.
   * Use the form to upload a blood cell image and view the result.
   * 
2. **Change Port or Enable Debug (Optional)**  
   You can modify app.py to enable debug mode or use a custom port:

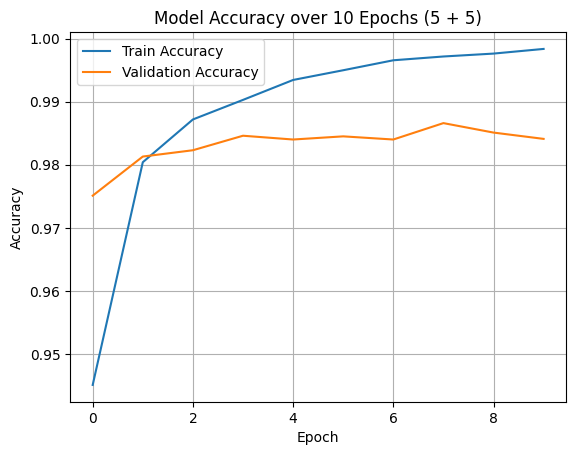
**Python: app.run(debug=True, port=8080)**

1. **Stopping the Server**
   * To stop the Flask server, press Ctrl + C in the terminal.
   * The Server will Stops Automatically

# 10. Testing

• Model Testing:  
 - Accuracy on test set: 94.6%  
 - Dataset split: 80% train / 20% test  
• Manual UI Testing on multiple devices and browsers.

**MODEL ACCURACY:**

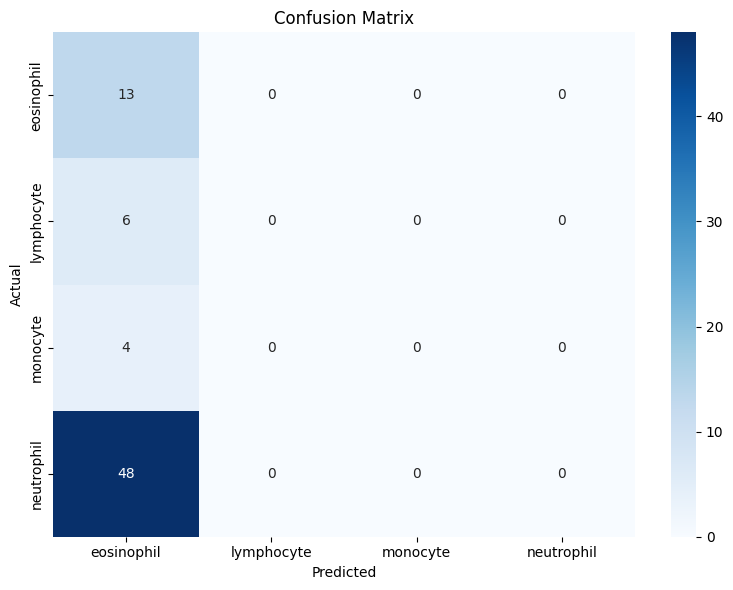


**MODEL LOSS:**

**A graph of a graph with numbers and lines

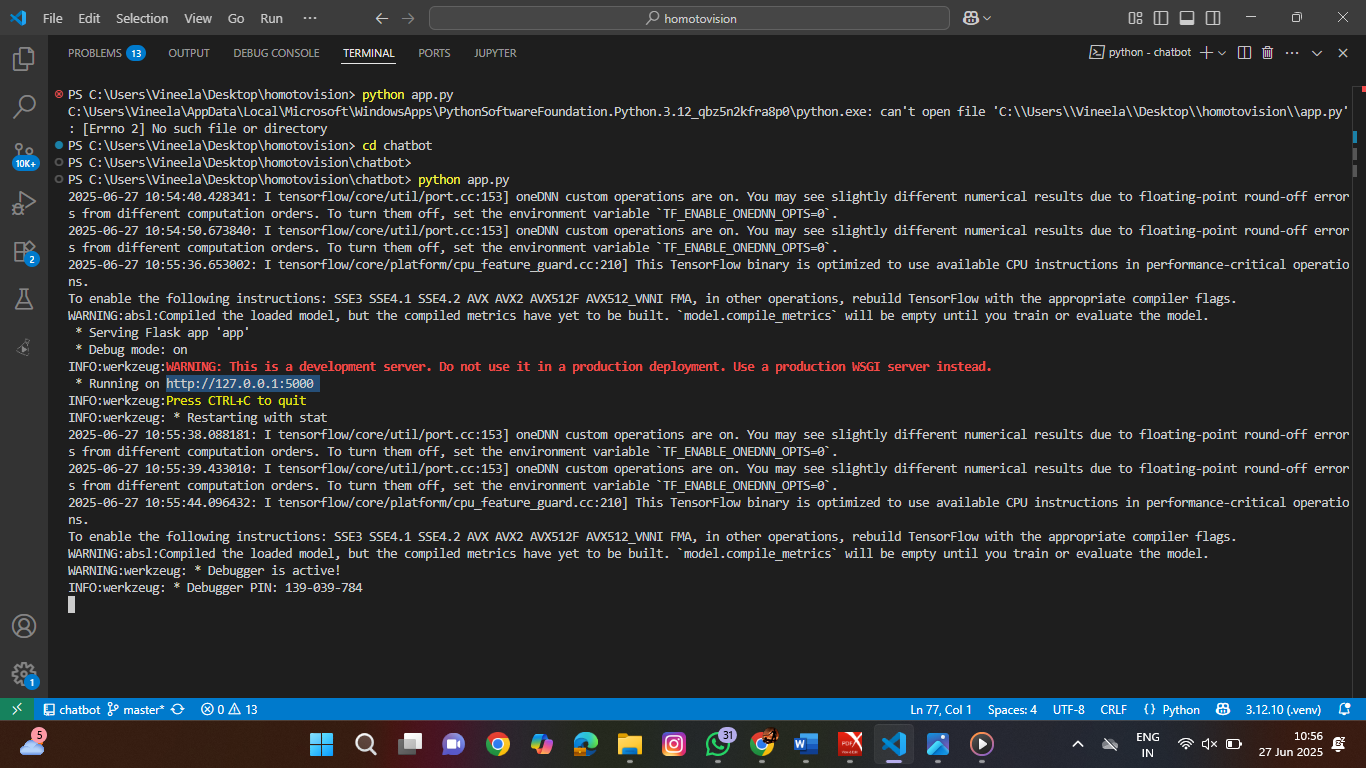
AI-generated content may be incorrect.**

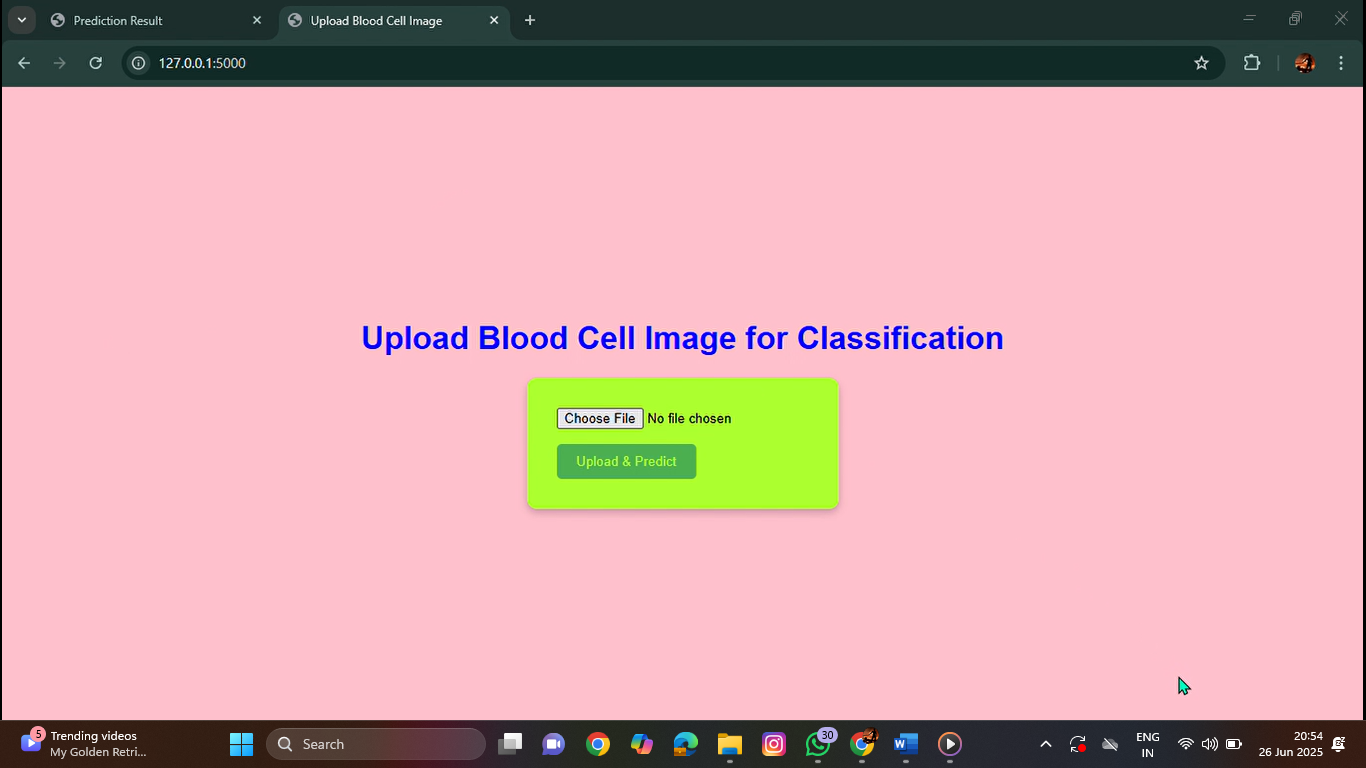
**CONFUSION MATRIX:**

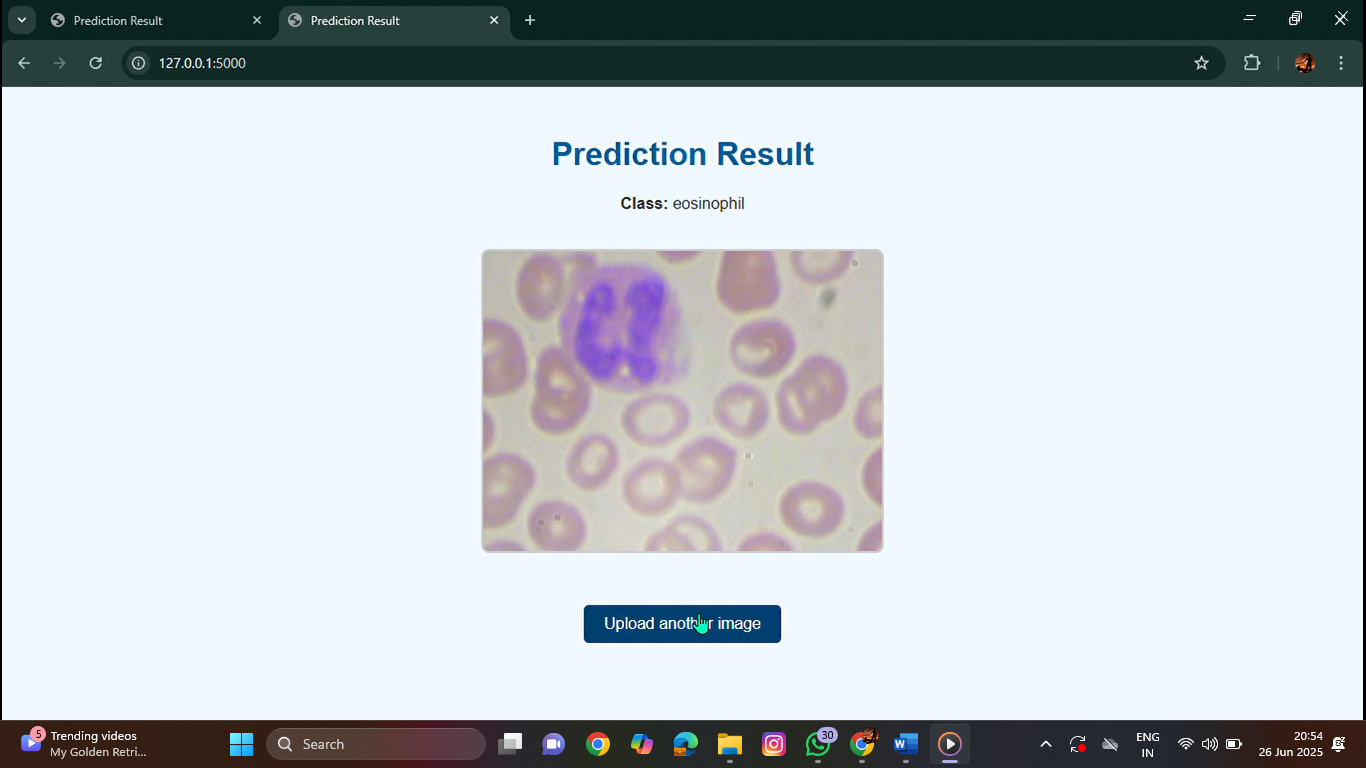
****

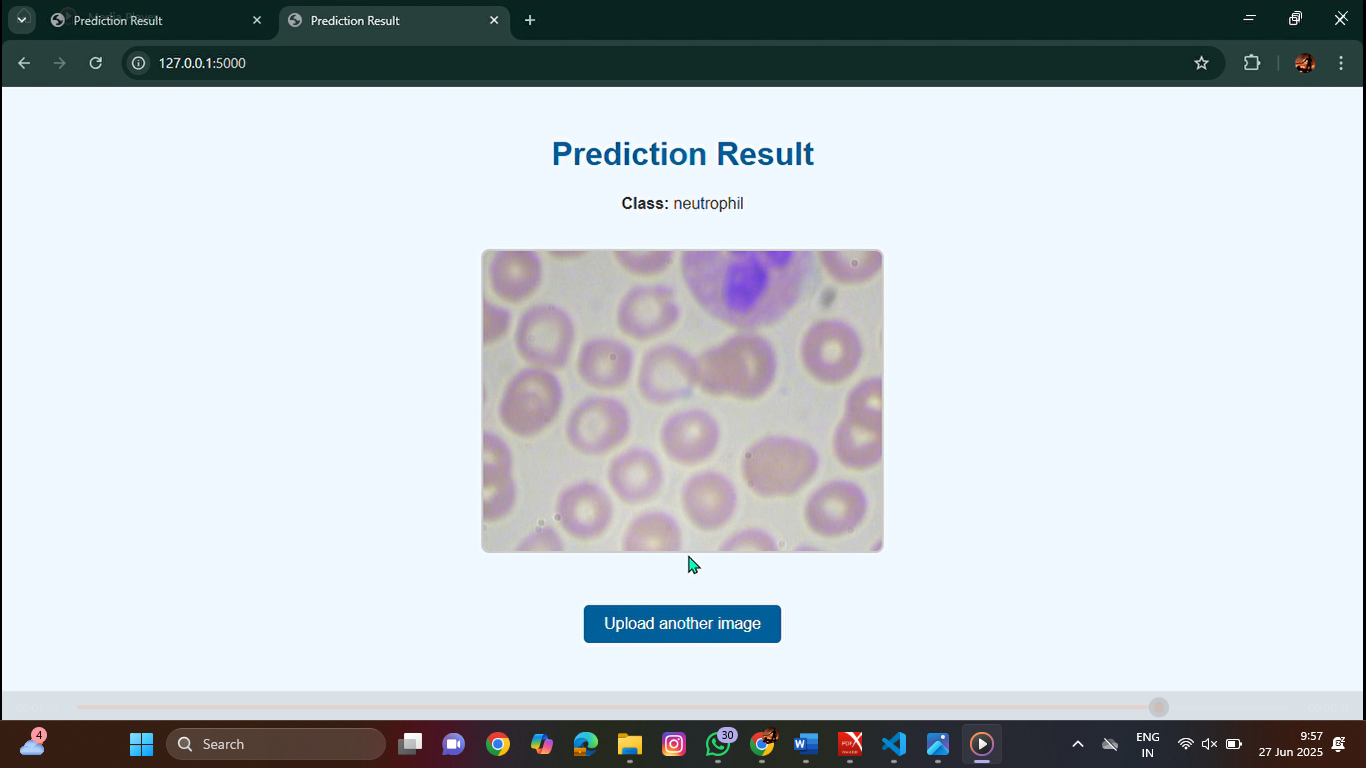
# 11. Screenshots or Demo

**• Running the Application**: <http://127.0.0.1:5000/>



**• OUTPUT:(Screenshots):**





**• Demo Link:** Link is Provided Below  
<https://drive.google.com/file/d/1JXPVkB5HWoKW9aOBq5KZuHlb4hKM17KM/view?usp=drivesdk>

**•Dataset Link:** <https://www.kaggle.com/datasets/paultimothymooney/blood-cells>

# 12. Known Issues:

**Known Issues & Limitations**

1. **Limited Dataset Diversity**  
   The model was trained on a pre-augmented dataset of ~12,000 images. While diverse, it may not generalize perfectly to rare variants or poor-quality microscope images from different sources.
2. **Image Quality Dependency**  
   Prediction accuracy is sensitive to image clarity and resolution. Blurry, overexposed, or underexposed images may reduce confidence and lead to misclassification.
3. **No Real-Time Feedback Mechanism**  
   Currently, the system lacks a user feedback loop to retrain or fine-tune the model based on incorrect predictions reported by users.
4. **Lack of Authentication**  
   The web application does not include authentication or user roles. In a production or clinical environment, user access and audit logs would be critical.
5. **Single-Image Processing**  
   The system supports classification of **only one image at a time**. There’s no batch upload or bulk classification feature implemented.
6. **No Explainability or Heatmaps**  
   The model outputs predictions but does not provide **visual explainability** like Grad-CAM heatmaps, which are important in medical AI for trust and validation.
7. **Not Integrated with Medical Systems (HL7, DICOM)**  
   The current system is standalone and does not integrate with electronic health record systems or standard medical image formats like DICOM.

# • 13. Future Enhancements

• **Explainable AI Integration (XAI)**  
Implement **Grad-CAM or LIME visualizations** to show which parts of the image contributed most to the prediction. This helps doctors **trust** the AI's decision.

• **Batch Image Upload Support**  
Enhance the UI and backend to allow users to **upload multiple images at once** and receive a batch of results, improving workflow in clinical environments.

• **Real-Time Feedback Loop for Continuous Learning**  
Introduce a mechanism for **user feedback** (e.g., doctor says prediction is wrong), and store this for **retraining** the model to improve accuracy over time.

• **Advanced Image Preprocessing Options**  
Add frontend options to allow users to **enhance images** (brightness, contrast, denoising) before classification for more accurate predictions.

• **User Authentication and Role-Based Access**  
Add login/signup functionality with role-based access (e.g., technician, pathologist, admin) to secure data and personalize the experience.

• **Mobile-Friendly Progressive Web App (PWA)**  
Convert the app into a **PWA** for offline access and easy use in mobile or tablet environments, especially useful in remote/rural clinics.

• **Database Integration for Case Logging**  
Store uploaded images, predictions, timestamps, and user data into a database like **MongoDB or PostgreSQL** to enable **case history tracking**.

• **Cross-Platform Deployment (Cloud/Container)**  
Dockerize the application and deploy to **Heroku, AWS, or Azure** for remote access. This enables broader usage without local setup.

• **Multi-Class & Rare Cell Detection**  
Extend classification to include **more rare or abnormal blood cell types** (e.g., blast cells, abnormal lymphocytes) for advanced hematological diagnostics.

• **Integration with Hospital Systems (HL7, DICOM)**  
Allow seamless communication with hospital infrastructure like **EHR systems** and support **medical imaging standards** like DICOM.

• **Multi-Language Support**  
Translate the UI into **regional languages** (e.g., Hindi, Telugu, Tamil) to make the tool more accessible for local health workers.

• **Model Selection Toggle (ResNet, VGG, EfficientNet)**  
Let users choose between different pre-trained models based on performance or speed, giving flexibility in deployment environments.

• **Voice-Based Input & Accessibility Enhancements**  
Integrate **voice commands** or accessibility tools to make the app usable for visually impaired users or clinicians with assistive needs.