Hematovision: Advanced Blood Cell Classification Using Transfer Learning

Project Documentation format

1. Introduction

- **ProjectTitle:**[Hematovision: Advanced Blood Cell Classification Using Transfer Learning]
- **TeamMembers:**

1. S. Rucksar (Reg.No.22HM1A05B9)
2. P. Gowthami (Reg.No.22HM1A05A3)
3. S. Moulali Baba (Reg.No.22HM1A05B5)
4. S. Hari Krishna (Reg.No.22HM1A05C6)

2. Project Overview

- **Purpose:**

Hematovision aims to develop an automated, image-based deep learning solution to classify blood cells (e.g., RBC, WBC, Platelets, infected cells) using transfer learning. This model can assist in early diagnosis of diseases such as anemia, leukemia, and infections. The project ensures precision, consistency, and speed in medical diagnosis by leveraging pre-trained CNN architectures.

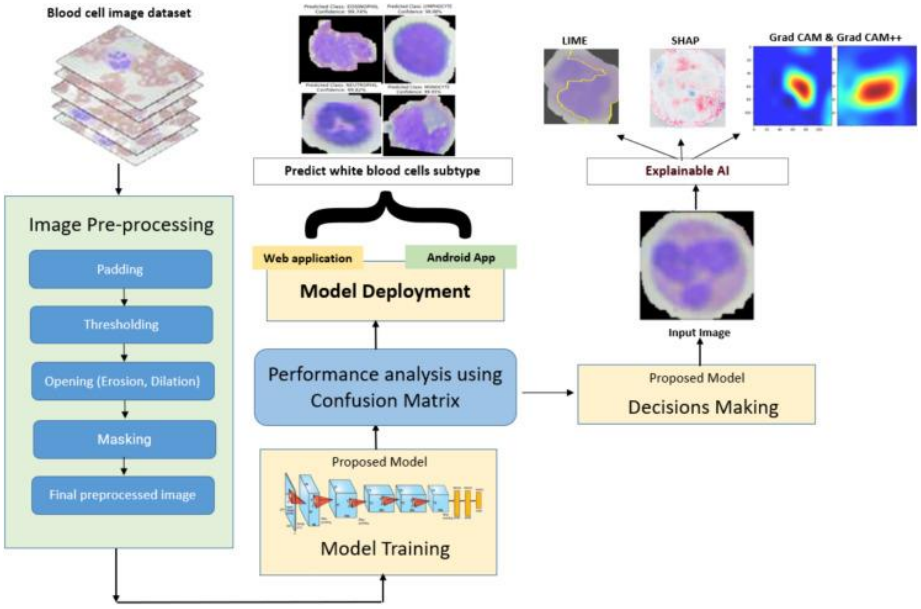
- **Goals:**

  - To automate blood smear image classification using transfer learning.
  - To reduce the workload on hematologists and pathologists.
  - To improve diagnostic accuracy in clinical laboratories.
  - To create a deployable web-based solution for real-time predictions.
  - To support healthcare with AI-powered diagnosis support.

- **Features:**

  - Image-based blood cell classification.
  - Transfer learning using pretrained CNN models
  - Flask-based web interface for image upload and instant results.
  - Display of predicted class and confidence score.
  - Visual display of input and output for user verification.
  - Expandability for new cell classes and disease-specific training.
  - Real-time usage in medical clinics and labs.
  - Compatible with mobile and desktop platforms.

3. Architecture



4. Setup Instructions

Software Requirements:

- Python 3.10+
- Visual Studio Code or Jupyter
- Internet connection (for dataset/model download)

Python Packages:

```
pip install numpy pandas matplotlib seaborn scikit-learn
       pip install tensorflow keras opencv-python
       pip install flask pillow
```

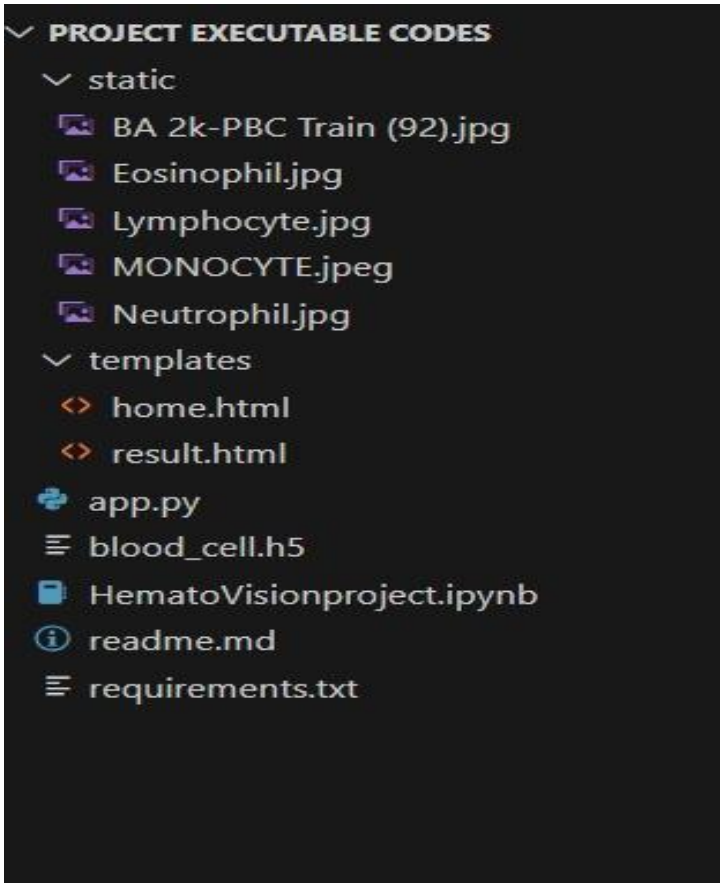
Installation Steps:

1. Create virtual environment (optional but recommended):

```
python -m venv venv
       source venv/Scripts/activate  # For Windows
       source venv/bin/activate      # For Linux/Mac
```

2. Download dataset (e.g., Blood Cell Dataset from Kaggle).
3. Place images in /data/train, /data/val, /data/test.
4. Run data_preparation.py to preprocess images.
5. Train model using train_model.py.
6. Save the model to /model/best_model.h5.
7. Run app.py to launch the web application.
8. Open browser at http://127.0.0.1:5000.

5. Folder Structure



6. Running the Application

Backend:

Activate your environment and run:

```
python app.py
```

Server will start at: `http://127.0.0.1:5000`

Frontend:

Access the frontend through browser. Features include:

- Home page
- Upload & Predict page
- Real-time display of prediction and confidence

7. API Documentation

Home Page:

- **Method:** GET
- **Returns:** index.html

Predict :

- **Method:** POST
- **Input:** Blood cell image (JPG, PNG)
- **Returns:** *Predicted label, confidence, image preview*

Sample Output:

```
{
"predicted_label":"Neutrophil",
"confidence":98.45,
"image_path":"static/uploads/image1.png"
}
```

8. Authentication

Current Status:

The current version of the Hematovision application is designed as a **public, demonstration-focused tool**. It does not implement any authentication or authorization mechanisms.

All users accessing the system:

- Can open the homepage and prediction page
- Can upload images of blood smears.

Why Authentication Matters:

While Hematovision works well for demo or prototype purposes, deploying it in **real medical or clinical settings** requires proper security and user management features. These ensure:

- Controlled access to patient-sensitive data
- Secure retraining and model update permissions
- Activity logs and usage auditability
- Multi-user functionality for labs and hospitals

Proposed Authentication Enhancements

To improve the security and scalability of Hematovision, the following authentication and authorization features can be added:

1. **Session-Based Authentication**

- Users must log in with a username and password
- Flask sessions and cookies track the user's login state

- Suitable for web browser access

2. Token-Based Authentication (JWT)

- Users receive a token after login
- Token is attached to API requests for secure access
- Preferred for mobile apps or REST API clients

3. Role-Based Access Control (RBAC)

- Define roles like:
  - **Admin**: Can upload new data, retrain model
  - **Lab Technician**: Can upload images for prediction
  - **Doctor/Viewer**: Can view history and results
- Each role has specific permissions

Future Features Based on Authentication

Feature	Description
Admin Login	Allow only verified personnel to update models/datasets
User Dashboard	Display history of uploads and predictions
Upload Restrictions	Prevent unauthorized users from uploading files
Audit Logs	Track who accessed what and when
Secure API Access	Use API tokens to control prediction access remotely

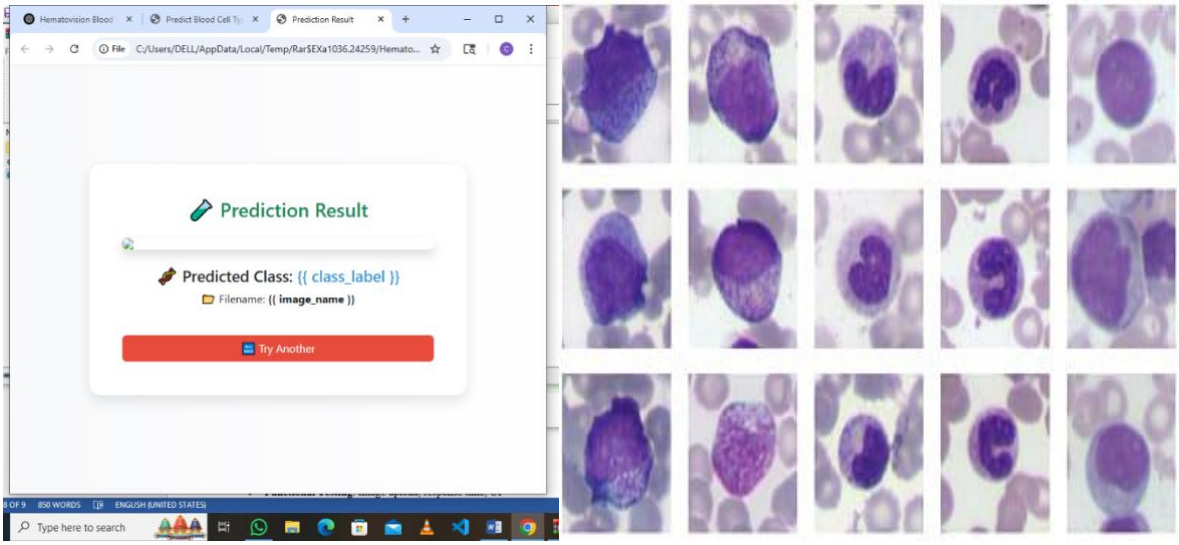
Technologies to Use for Implementation:

Technology	Purpose
Flask-Login	User session management
Flask-JWT-Extended	Secure token generation and access
SQLite/PostgreSQL	Store user credentials and roles
HTTPS with SSL	Encrypt all communication
OAuth 2.0 (optional)	Third-party login like Google, Microsoft

Example Workflow (Future Version)

1. **Admin logs in** using secure credentials
2. **Dashboard opens** showing usage, predictions, and model stats
3. **Technician logs in** to upload test samples
4. Unauthorized users see **"Access Denied"** message

9. User Interface



Description	Image 1	Image 2	Image 3	Image 4	Image 5
Original images					
Pre-processed images					
A-FCM-based cell segmented images					
WBC outcomes					
RBC outcomes					
Platelets outcomes					

10. Testing:

Testing Strategy:

To ensure the **robustness, generalization**, and **accuracy** of the Hematovision blood cell classification system, a combination of **manual** and **automated testing techniques** was employed throughout the development process.

Model Evaluation Testing:

- Evaluated the trained deep learning model using separate validation and test datasets to avoid data leakage.
- Performance metrics such as:
  - Accuracy
  - Confusion matrix
  - Precision, recall, and F1-score
  - Classificationreport
    were generated using scikit-learn and TensorFlow.

External Image Testing:

- The model was manually tested using real-world blood smear images that were not part of the training dataset**.**
- This helped assess the model's generalization ability across:
  - Varying lighting conditions
  - Different resolutions
  - Diverse microscope hardware
- It also verified predictions on rare or less frequent blood cell types**.**

Web Application Functional Testing:

- Verified that all web pages (Home, About, Predict) were:
  - Properly linked
  - Fully rendered
  - Displayed accurate textual content
- Tested image upload and prediction functionality via the /predict endpoint in the browser.
- Ensured:
  - The uploaded image previewed correctly
  - The predicted blood cell class was clearly shown
  - The confidence score was displayed with each result
- UI responsiveness and button functionality were also checked on both desktop and mobile browsers.

Tools Used:

Tool / Library	**Purpose**
TensorFlow / Keras	Model training, validation, and inference
scikit-learn	Confusion matrix, classification report, accuracy metrics
OpenCV & Pillow	Image preprocessing and conversion
Matplotlib / Seaborn	Visualization of evaluation results (e.g., heatmaps)
Flask (Debug Mode)	Web application testing and route debugging
Browser Console (Chrome DevTools)	Manual frontend inspection and testing
Manual Testing	Image uploads and interaction testing via UI

11. Screenshots or Demo

These steps guide you from setting up the environment to running the application locally using the Flask web server. Follow them in sequence to successfully launch the project.

Step 1: Clone or Download the Project

- Download the project files from GitHub or unzip the provided package.
- Navigate to the project directory using the terminal:

```
cd hematovision/
```

Step 2: Create a Virtual Environment (Optional but Recommended)

To keep dependencies isolated:

```
python -m venv venv
```

- **Activate Environment:**
  - On Windows:

    ```
    venv\Scripts\activate
    ```

  - On Mac/Linux:

    ```
    sourcevenv/bin/activate
    ```

Step 3: Install Required Packages

Install the Python dependencies required to run the model and web app:

```
pip install tensorflow flask numpy pandas scikit-learn opencv-python
pillow matplotlib seaborn
```

Step 4: Download and Organize Dataset

- Place the dataset inside the `/data/` folder.
- Organize the data into:

  ```
  /data/train/
  /data/validation/
  /data/test/
  ```

- Each class (e.g., Lymphocyte, Neutrophil) should be a subfolder containing images.

Step 5: Train the Model (Optional if Pretrained Model is Provided)

If a trained model is not included, run:

```
python train_model.py
```

- The script will load the dataset, apply preprocessing, and train the model.
- A trained model (`best_model.h5`) will be saved inside the `/model/` folder.

Step 6: Verify Model File

Ensure the file `model/best_model.h5` exists. If not, you must train the model using Step 5.

Step 7: Run the Flask Application

Start the backend server:

```
python app.py
```

If successful, you will see:

```
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Step 8: Open the Web Interface

1. Open your web browser.
2. Enter:

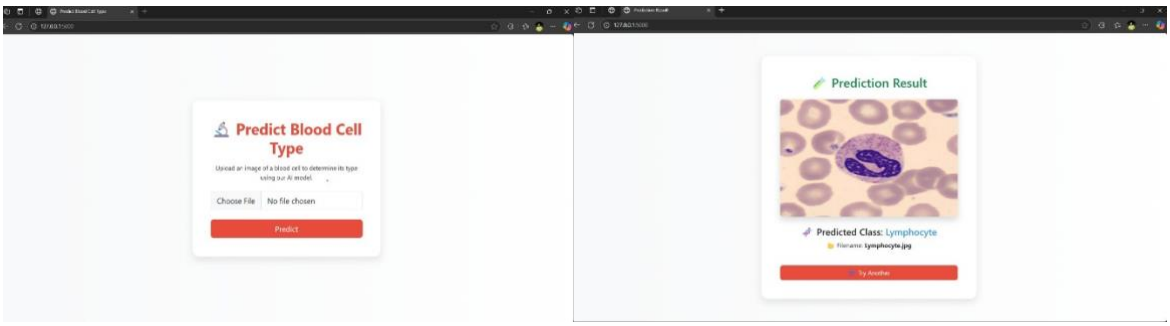
   ```
   http://127.0.0.1:5000/
   ```

3. Use the UI to:
   - Upload a blood smear image
   - Get prediction and confidence score
   - View the image with the result

Step 9: Test with Sample Images

Use sample test images to validate model behavior. Check if the predicted label matches the expected blood cell class.

Screenshots :



Project Demo Link:

https://drive.google.com/file/d/1fls94YIWcvp0eJuR7GeZtaWMAlghDRpx/view?usp=drivesdk

12. Known Issues

While the Hematovision system performs well under standard conditions, several known limitations and issues were identified during testing and deployment. These are summarized below to guide future improvements and enhance awareness of current constraints.

Model-Related Issues

Issue	Description
Misclassification in Edge Cases	The model sometimes misclassifies blood cells when images are blurry, low-resolution, or taken under non-standard lighting conditions.
Low Confidence Scores	For certain borderline cases or rare cell types, the confidence score returned by the model may be low, indicating uncertainty.
Imbalanced Dataset Bias	If the dataset used for training is not balanced across all blood cell classes, the model may show bias toward more common types.

Image Handling Issues

Issue	Description
Large Image Delay	Uploading large image files (>5MB) may cause a delay in prediction or slow page rendering.
No Image Preview Before Upload	The user currently cannot see a preview of the selected image before submitting it.
Unsupported File Types	The application does not strictly validate image types, allowing potential upload of unsupported or invalid files (e.g., `.txt`, `.exe`).

User Interface & UX Issues

Issue	Description
Lack of Error Feedback	If something goes wrong the user might not receive a clear error msg.
Limited Mobile Responsiveness	The web interface is not fully optimized for use on smartphones or small-screen tablets.
No Drag-and-Drop Support	Users can only use the traditional file upload button and cannot drag-and-drop images into the form.

Security and Access Issues

Issue	Description
No Authentication	Anyone accessing the local web address can use the tool without logging in.
File Overwrite Risk	If a user uploads a file with the same name as an existing one, it may overwrite the old file without warning.

Testing-Related Issues

Issue	Description
Inconsistent Results on External Images	When using real-world microscope images not part of the training data, results may vary based on quality and magnification.
Manual Reset Required	After uploading an image, there's no automatic reset of the interface — the user must manually refresh to submit another image.

Recommendations for Resolution

- Implement stricter file type and size validation
- Add image preview and drag-and-drop support
- Introduce mobile responsiveness
- Develop authentication and user role management
- Use data augmentation and oversampling to balance training data
- Apply real-time error handling with user feedback messages

13. Future Enhancements

Model Improvements: Larger dataset, additional disease labels

Advanced UI: Image preview, drag & drop

Batch Processing: Upload multiple images at once

Security: File type validation, role-based access

Reporting: Downloadable prediction reports

Cloud Deployment: Deploy on Heroku or AWS

Mobile Support: Mobile-first responsive design