# HEMATOVISION: ADVANCED BLOOD CELL CLASSIFICATION USING TRANSFER LEARNING

# 1. Abstract

HematoVision is a deep learning-powered web application that classifies blood cells using image input. This document elaborates on its architecture, technical implementation, workflow, and future scope in clinical diagnostics.

# 2. Introduction

Blood cell classification plays a vital role in identifying various blood-related diseases, including leukemia, anemia, and infections. Manual identification is time-consuming and prone to human error. HematoVision bridges this gap by automating the classification process using machine learning and computer vision.

The application provides real-time inference through a user-friendly web interface. The use of Flask for the backend and Keras for deep learning offers a lightweight yet powerful deployment model.

# 3. System Overview

HematoVision allows users to upload an image of a blood cell, processes it using a pre-trained model, and displays the predicted class of the cell. The backend uses Python and Keras to perform inference, while the frontend is built with HTML and CSS.

Typical use case: A laboratory technician uploads a microscope image to get instant predictions, reducing analysis time.

# 4. Technologies Used

• Python 3.11 - General-purpose programming and scripting language.

• Flask - Lightweight web framework for routing and backend logic.

• TensorFlow / Keras - Used to build and load the pre-trained deep learning model.

• HTML + CSS - For building the frontend interface of the application.

• JSON - Stores class index to label mappings.

# 5. Model Architecture and Training

The model used in HematoVision is a Convolutional Neural Network (CNN) trained on a labeled dataset of blood cell images. The images were preprocessed (resized to 224x224, normalized) before being fed into the model. The model was trained using categorical cross-entropy loss and optimized with the Adam optimizer.

During training, an 80-20 train-validation split was used to ensure generalization. Performance was tracked via accuracy and loss metrics, with early stopping implemented to prevent overfitting.

# 6. Application Structure

• app.py: Main Flask app handling routes and prediction logic.

• model/: Contains 'blood\_model.keras' and class mapping JSON file.

• static/uploads/: Stores uploaded images.

• templates/: HTML files (index.html, prediction.html, logout.html).

# 7. Flask Web App Routing

• `/`: Main upload form.

• `/predict`: Handles image upload and prediction logic.

• `/logout`: Shows a thank-you message.

# 8. UI/UX Design

The frontend interface includes a drag-and-drop upload form with clear buttons for submission. Feedback is provided immediately after prediction, and the result page displays the image with the predicted class.

# 9. Prediction Workflow

1. Image is uploaded and stored in static/uploads/.

2. Image is resized to 224x224 pixels and normalized.

3. Converted into a NumPy array and passed to the model.

4. The model outputs probabilities; the highest one is selected.

5. Class index is mapped back to a human-readable label using JSON.

# 10. Deployment Strategy

For local deployment, Python and Flask can be used directly. For production, it is recommended to use a WSGI server like Gunicorn and reverse proxy with Nginx. Docker containers or cloud platforms (AWS EC2, Heroku) may also be used for scalability.

# 11. Error Handling & Debugging

• PermissionError: Fixed by ensuring upload directory permissions.

• Overwrite Error: Timestamped filenames prevent collisions.

• Image Display Issues: Corrected with `url\_for` usage in templates.

# 12. Security Considerations

Input files are validated to ensure only images are accepted. Direct file paths are not exposed. Future improvements include adding user authentication and antivirus scanning of uploads.

# 13. Future Enhancements

• Display prediction confidence score to user.

• Store prediction logs in a database.

• Add an admin dashboard to review image uploads.

• Automatically delete old files to free up space.

# 14. Conclusion

HematoVision showcases how artificial intelligence can aid in medical diagnostics. It is lightweight, easy to deploy, and open to future expansion. This application serves as a prototype for real-world clinical tools using deep learning.

# 15. References

• TensorFlow: https://www.tensorflow.org/

• Flask: https://flask.palletsprojects.com/

• Dataset Source: [To be specified based on original dataset used]