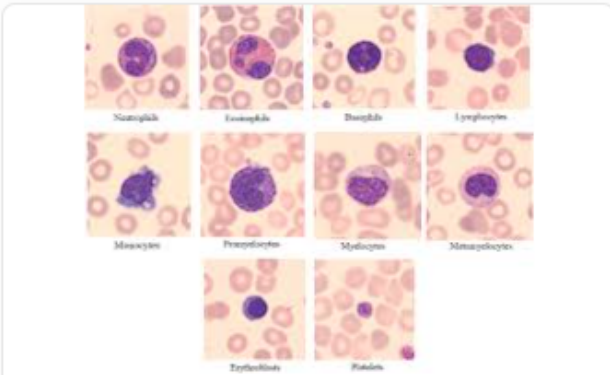
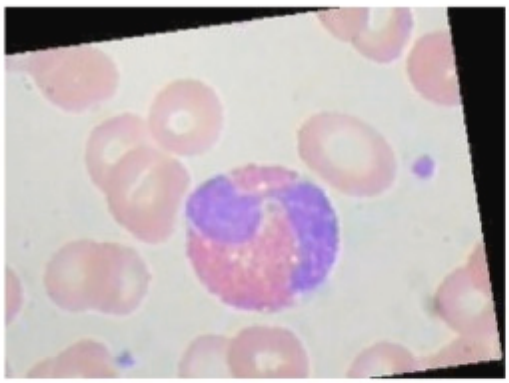
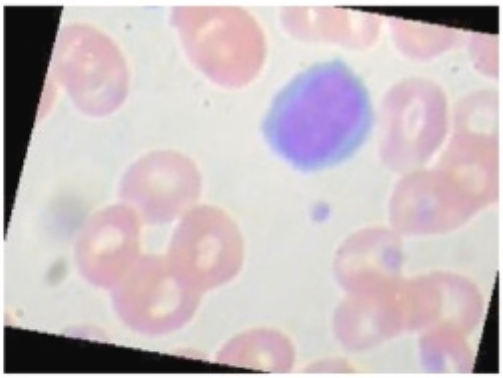
HematoVision: Advanced Blood Cell Classification Using Transfer Learning



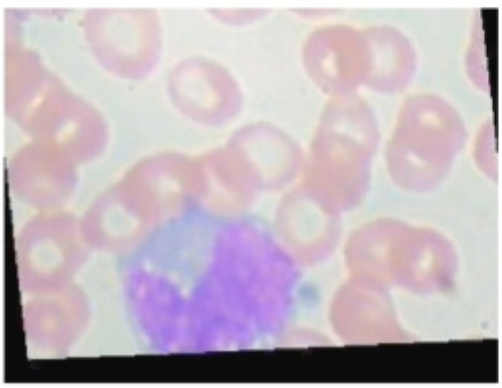
Hematovision: Advanced Blood Cell Classification Using Transfer Learning



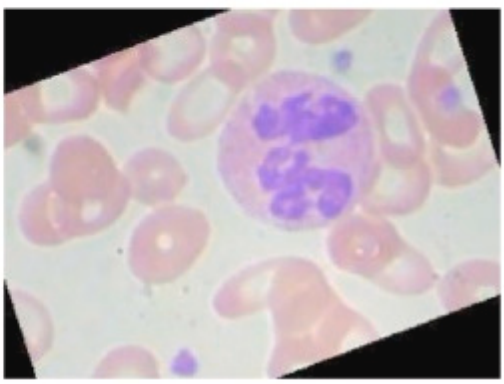
(a)Eosinophil



(b)Lymphocyte



(c)Monocyte



(d)Neutrophil

Introduction:

HematoVision is a deep learning-based project aimed at automating the classification of blood cells using advanced transfer learning techniques. The project leverages the MobileNetV2 architecture, integrated with a Flask web framework, to accurately identify four types of white blood cells: Neutrophils, Lymphocytes, Monocytes, and Eosinophils. By processing medical image data and deploying the solution via a user-friendly web interface, HematoVision supports early diagnosis and efficient healthcare services.

Blood cell classification is a fundamental process in hematological diagnostics, playing a critical role in identifying and managing various diseases such as leukemia, anemia, infections, and other immune-related conditions. Traditionally, this process involves manual examination of stained blood smear slides under a microscope by trained hematologists or pathologists. While effective, manual analysis is time-consuming, prone to human error, and difficult to scale in high-demand or resource-limited settings.

To address these limitations, **HematoVision** presents an advanced, automated solution for blood cell classification using **transfer learning**, a cutting-edge machine learning technique. Transfer learning leverages the knowledge gained by deep neural networks trained on large-scale image datasets and applies it to domain-specific tasks, such as medical image analysis, where annotated data may be limited.

The core of HematoVision lies in using **pre-trained convolutional neural networks (CNNs)**, which are fine-tuned on a specialized dataset of **12,000 annotated blood cell images**. These images are categorized into clinically relevant classes: **eosinophils, lymphocytes, monocytes, and neutrophils**. The transfer learning approach enables the model to understand and recognize intricate features of blood cells without starting from scratch, thus accelerating training and significantly improving classification performance.

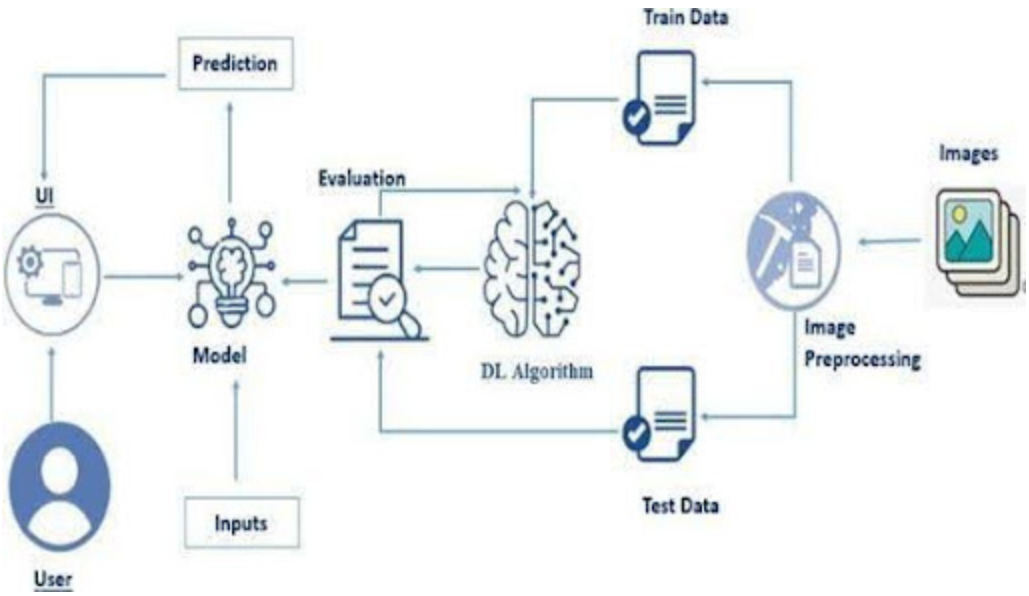
By harnessing the power of artificial intelligence and computer vision, HematoVision aims to:

- **Enhance diagnostic accuracy:** Reduce errors associated with manual classification.
- **Accelerate processing time:** Deliver faster analysis and results for urgent clinical decisions.
- **Ensure scalability:** Deploy across a wide range of clinical settings, from urban hospitals to remote clinics.
- **Support educational initiatives:** Provide interactive platforms for training future healthcare professionals.

Furthermore, the integration of HematoVision into healthcare workflows supports a

new era of **intelligent diagnostic tools** that not only assist clinicians but also democratize access to high-quality diagnostics worldwide. Through real-world applications such as **automated diagnostics**, **telemedicine**, and **medical education**, HematoVision is poised to make a substantial impact on modern healthcare delivery.

Architecture:



System Workflow Overview

This diagram represents a deep learning-based blood cell classification system using **transfer learning**. The process starts with the user uploading blood cell images through a user interface. These images are then preprocessed—resized, normalized, and augmented for consistency. The cleaned data is split into training and testing sets. A pre-trained Convolutional Neural Network (CNN), such as MobileNetV2, is used as the base model. Using transfer learning, the model is fine-tuned with the blood cell images to adapt to the specific classification task. The model learns to distinguish between eosinophils, lymphocytes, monocytes, and neutrophils. After training, the model is evaluated for accuracy and performance. Once validated, it is used to predict new image classes in real time. The final predictions are presented to the user via the UI, enabling quick and accurate blood cell analysis.

Prerequisites

- To complete this project, you must require the following software, concepts, and packages
    - Anaconda Navigator:
        - Refer to the link below to download Anaconda Navigator
    - Python packages:
    - 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.

Project Objectives

By the end of this project, you will:

- Develop an Accurate Classification Model:
  Create a deep learning model capable of accurately classifying blood cells into categories such as eosinophils, lymphocytes, monocytes, and neutrophils.
- Utilize Transfer Learning for Efficiency:
  Implement transfer learning using pre-trained CNNs (e.g., MobileNetV2) to reduce training time and improve performance on limited medical image data.
- Enhance Diagnostic Automation:
  Automate the blood cell identification process to assist pathologists and reduce manual workload in clinical laboratories.
- Ensure Scalability and Real-Time Prediction:
  Build a system that can deliver fast and reliable predictions, scalable for

integration into clinical and remote healthcare environments.

- **Support Remote Diagnostics and Telemedicine:**
  Enable healthcare professionals to analyze blood cell images remotely, improving access to quality diagnostics in rural or underserved areas.
- **Create an Educational Tool:**
  Provide a practical learning platform for medical students and lab technicians to practice and understand blood cell classification.

Project Flow:

The HematoVision project follows a systematic workflow to develop, train, and deploy a blood cell classification model using transfer learning. The process is divided into key phases, from data handling to application deployment:

1. User Interaction

- The process begins when the user uploads an image through the web-based User Interface (UI).
- This UI is designed using HTML and is connected to a Flask backend.

2. Image Analysis and Prediction

- The uploaded image is passed to a deep learning model integrated into the Flask application.
- The model, built using a pre-trained CNN (e.g., MobileNetV2), analyzes the image and predicts the blood cell type (eosinophil, lymphocyte, monocyte, or neutrophil).
- The prediction result is then displayed on the UI for the user.

3. Development Workflow

To achieve this complete system, the following steps are performed:

a. Data Collection

- Acquire a dataset of annotated blood cell images (e.g., 12,000 images across

multiple classes).

b. Data Pre-processing

- Resize, normalize, and format images to ensure consistency.
- Convert class labels into a suitable format for training.

c. Data Augmentation

- Apply techniques like rotation, flipping, zooming, and brightness changes to increase dataset variability and improve model generalization.

d. Data Splitting

- Split the dataset into training and testing subsets to train and evaluate the model.

e. Model Building

- Import necessary libraries such as TensorFlow, Keras, NumPy, etc.
- Initialize the model using a pre-trained architecture (transfer learning).
- Add custom classification layers for the specific task.

f. Model Training and Testing

- Train the model on the training set and validate it on the test set.
- Monitor performance metrics during training (accuracy, loss).

g. Model Evaluation

- Evaluate the trained model using metrics like accuracy, precision, recall, and confusion matrix.

h. Model Saving

- Save the trained model in .h5 or another format for future use during prediction.

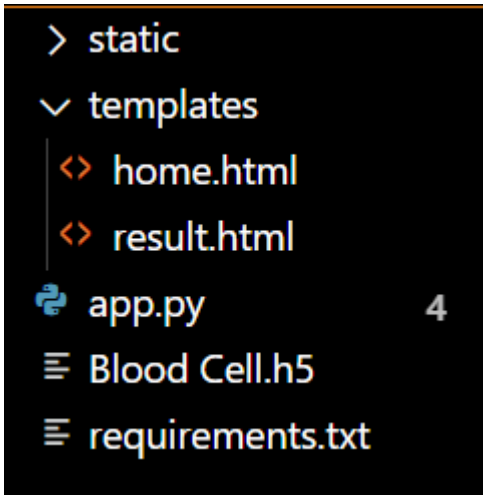
i. Application Building

- Develop a Flask application to handle image uploads and predictions.
- Create a simple HTML UI for users to interact with the system.
- Integrate the saved model with Flask to return real-time predictions.

Project Structure

- The HematoVision project follows a clean and organized folder structure to ensure clarity, maintainability, and smooth integration of all components.
- Create a project folder that includes the following files and directories:

```
HematoVision/
│
├── app.py              # Main Python script to run the Flask application
├── Blood_Cell.h5       # Pre-trained and saved deep learning model
├── static/             # Folder to store static files (CSS, images if any)
│
├── templates/          # Contains HTML templates for the web interface
│   └── index.html      # Main HTML file for the user interface
│
├── requirements.txt    # (Optional) List of required packages
└── README.md           # (Optional) Project overview and setup instructions
```



Data Collection and Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

1. Data Collection

The foundation of any successful deep learning model lies in the quality and quantity of its dataset. For the HematoVision project, we utilize a dataset containing **12,000 annotated blood cell images**, classified into four main categories:

- **Eosinophils**
- **Lymphocytes**
- **Monocytes**
- **Neutrophils**

The dataset can be sourced from public repositories such as:

- Blood Cell Images on Kaggle
- Open-source medical image datasets
- Digitized blood smear samples

Each image in the dataset is labeled and standardized for use in supervised learning tasks.

2. Data Preparation

Before feeding the data into a deep learning model, it must be properly prepared to ensure accuracy and consistency:

a. Image Resizing

All images are resized to a fixed dimension (e.g., 224×224 pixels) to match the input size expected by pre-trained CNNs such as MobileNetV2.

b. Normalization

Pixel values are scaled to a range of 0 to 1 for faster convergence and more stable training.

c. Data Augmentation

To increase dataset variability and reduce overfitting, augmentation techniques are applied, including

d. Train-Test Split

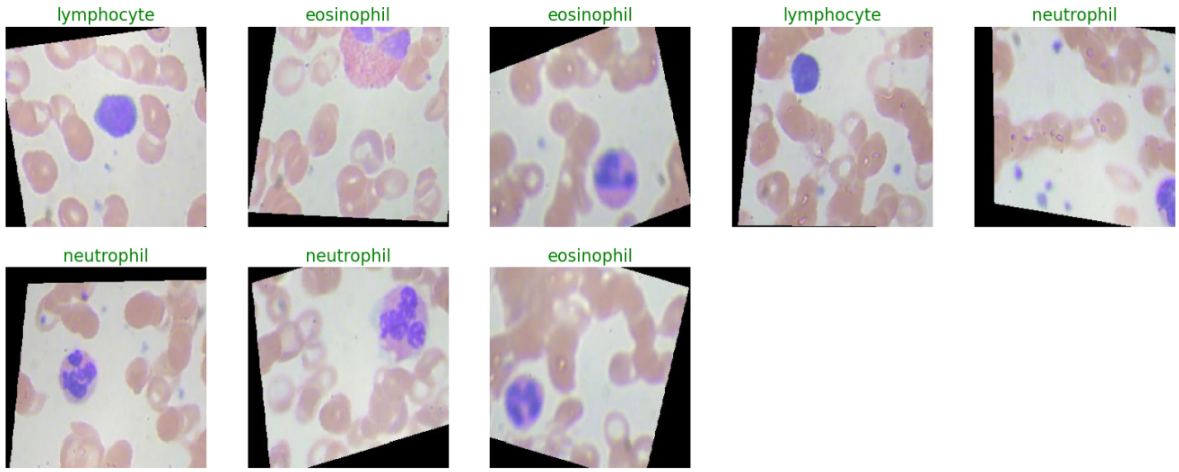
The dataset is split into:

- **Training Set** (typically 80%): Used to train the model.
- **Testing Set** (20%): Used to evaluate model performance on unseen data.

Data Visualization

The provided Python code imports necessary libraries and modules for image manipulation. It selects a random image file from a specified folder path. Then, it displays the randomly selected image using IPython's Image module. This code is useful for showcasing random images from a directory for various purposes like data exploration or testing image processing algorithms.

```
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)
```



In the above code, I used class ace of diamond for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as ace of diamond.

Split Data and Model Building

Train-Test-Split:

In this project, we have already separated data for training and testing.

```
train_images, test_images = train_test_split(bloodCell_df, test_size=0.3, random_state=42)
train_set, val_set = train_test_split(bloodCell_df, test_size=0.2, random_state=42)
```

```
print(train_set.shape)
print(test_images.shape)
print(val_set.shape)
print(train_images.shape)
```

```
(7965, 2)
(2988, 2)
(1992, 2)
(6969, 2)
```

```
image_gen = ImageDataGenerator(preprocessing_function= tf.keras.applications.mobilenet_v2.preprocess_input)
train = image_gen.flow_from_dataframe(dataframe= train_set,x_col="filepaths",y_col="labels",
                                      target_size=(244,244),
                                      color_mode='rgb',
                                      class_mode="categorical",
                                      batch_size=8,
                                      shuffle=False
                                     )
test = image_gen.flow_from_dataframe(dataframe= test_images,x_col="filepaths", y_col="labels",
                                     target_size=(244,244),
                                     color_mode='rgb',
                                     class_mode="categorical",
                                     batch_size=8,
                                     shuffle= False
                                    )
val = image_gen.flow_from_dataframe(dataframe= val_set,x_col="filepaths", y_col="labels",
                                    target_size=(244,244),
                                    color_mode= 'rgb',
                                    class_mode="categorical",
                                    batch_size=8,
                                    shuffle=False
                                   )
```

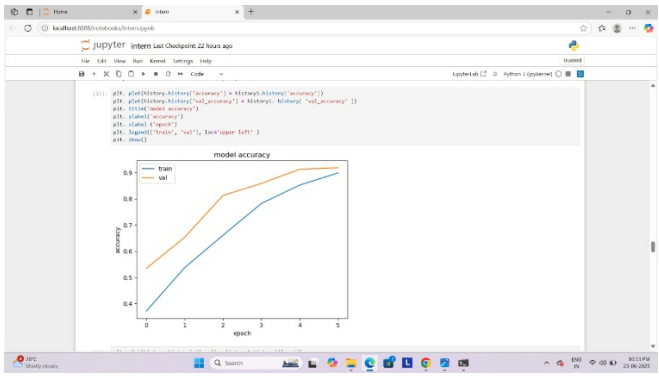
```
Found 7965 validated image filenames belonging to 4 classes.
Found 2988 validated image filenames belonging to 4 classes.
Found 1992 validated image filenames belonging to 4 classes.
```

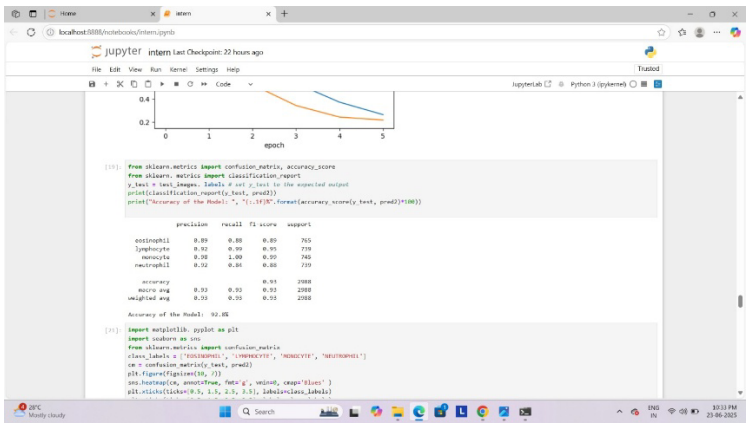
The preprocessed dataset is split into **training and testing sets**, typically in an 80:20 ratio.
This ensures the model is trained on one portion and evaluated on unseen data to check its performance.
A **pre-trained CNN model** like MobileNetV2 is used as the base for transfer learning. Custom classification layers are added on top to suit the blood cell classification task.
The model is then compiled, trained, and optimized using the training data.

Model Building:

In this project, model building is performed using **transfer learning**, where a pre-trained Convolutional Neural Network (CNN) such as **MobileNetV2** is used as the base model. The top layers of the pre-trained model are removed, and new custom layers are added to perform blood cell classification. These layers include a **Global Average Pooling layer**, one or more **Dense (fully connected) layers**, and a **Softmax output layer** to classify the input image into one of the four categories: eosinophil, lymphocyte, monocyte, or neutrophil. The model is then compiled using an appropriate loss function (e.g., categorical crossentropy), optimizer (e.g., Adam), and evaluation metrics (e.g., accuracy). This approach speeds up training and improves performance, especially when working with a limited medical dataset.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=128, kernel_size=(8, 8), strides=(3, 3), activation='relu', input_shape=(224, 224, 3)),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3, 3)),

    keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Flatten(),
    keras.layers.Dense(1024, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1024, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4, activation='softmax')
])

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.optimizers.SGD(learning_rate=0.001),
    metrics=['accuracy']
)

model.summary()
```

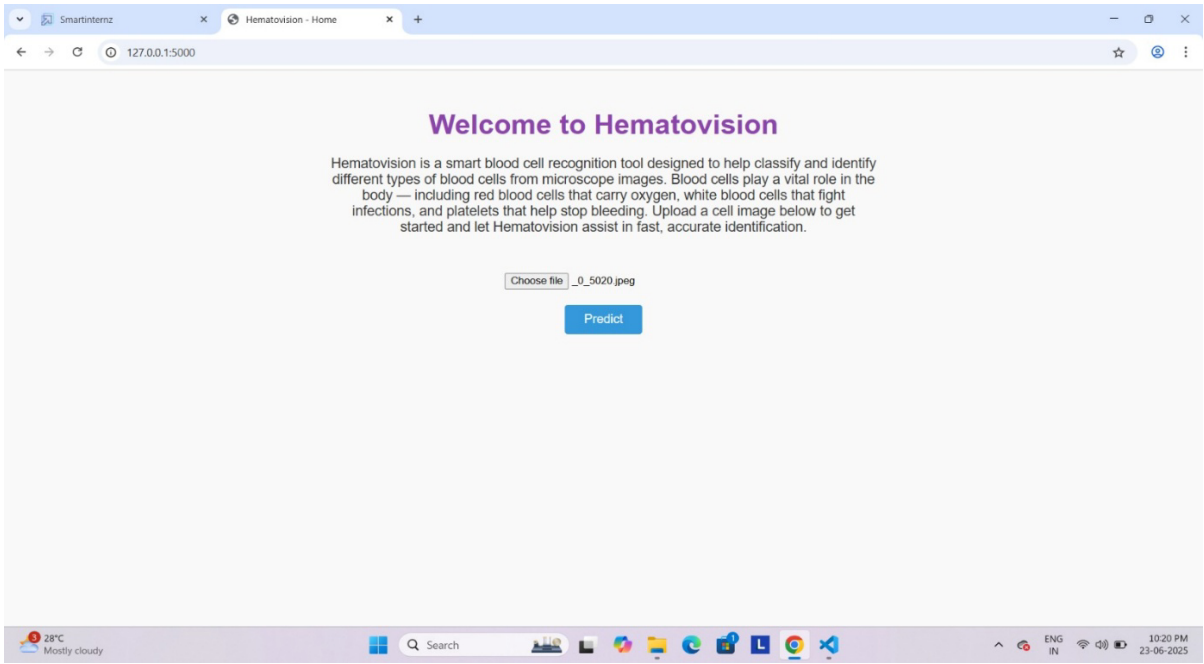


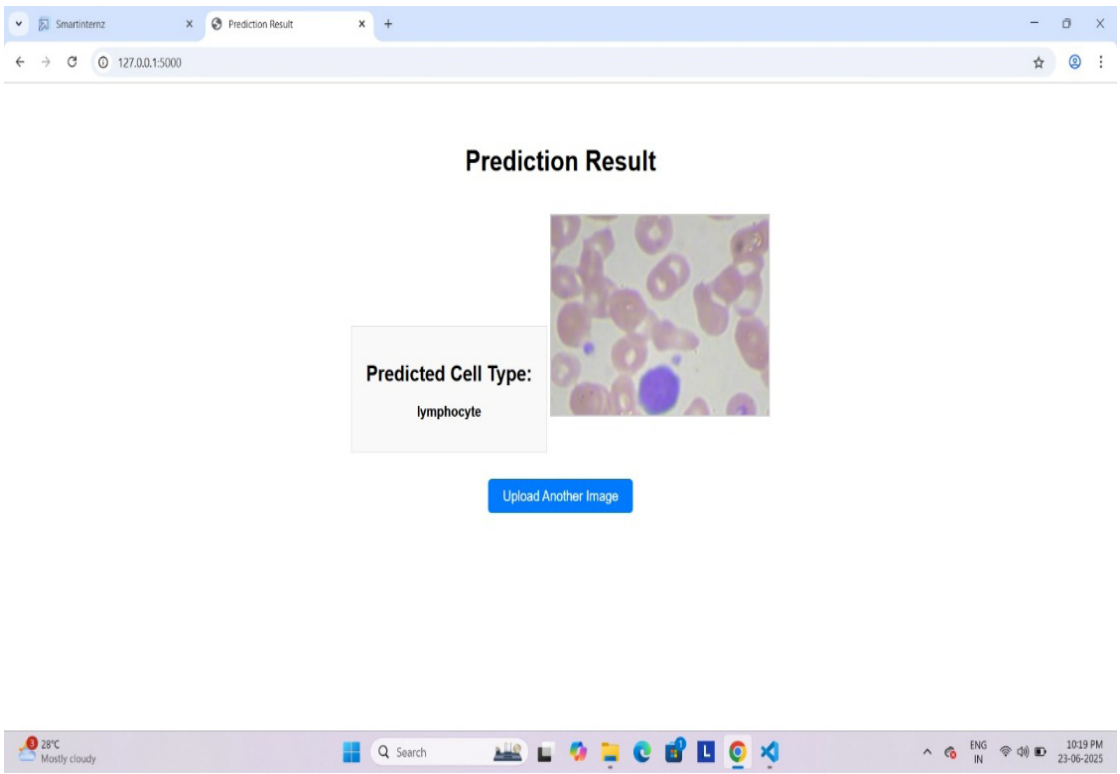


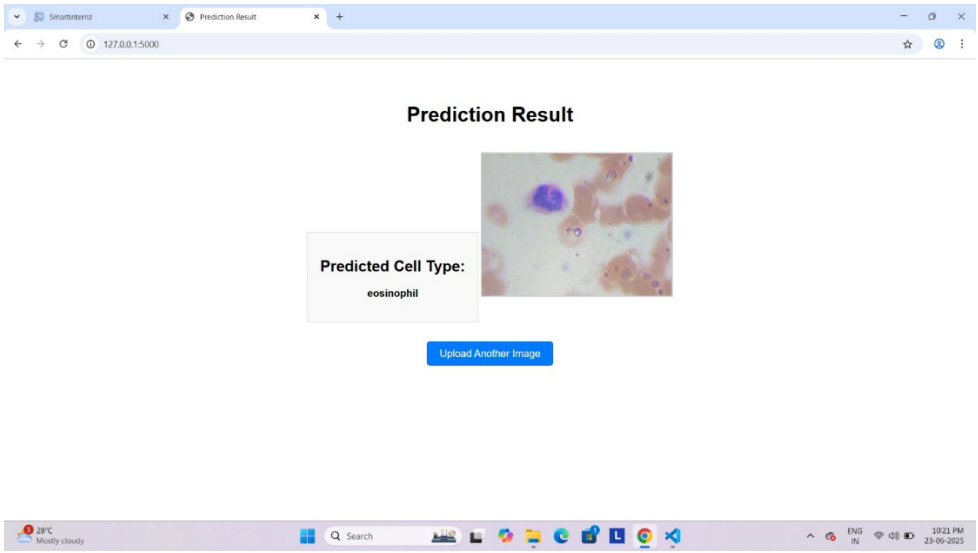
Testing Model & Data Prediction

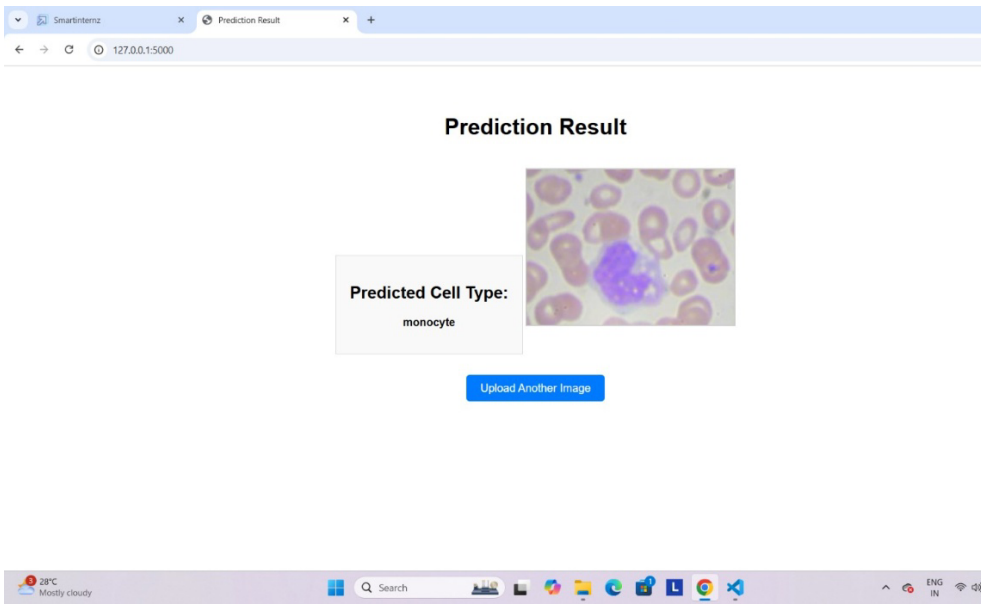
Evaluating the model

Here we have tested with the Mobilenet V2 Model With the help of the predict () function.


Smartinternz
Hematovision - Home
127.0.0.1:5000
Welcome to Hematovision
Hematovision is a smart blood cell recognition tool designed to help classify and identify different types of blood cells from microscope images. Blood cells play a vital role in the body — including red blood cells that carry oxygen, white blood cells that fight infections, and platelets that help stop bleeding. Upload a cell image below to get started and let Hematovision assist in fast, accurate identification.
Choose file
_0_5020.jpeg
Predict
28°C
Mostly cloudy
Search
ENG
IN
10:20 PM
23-06-2025


Smartinternz
Prediction Result
127.0.0.1:5000
Prediction Result
Predicted Cell Type:
lymphocyte
Upload Another Image
28°C
Mostly cloudy
Search
ENG
IN
10:19 PM
23-06-2025

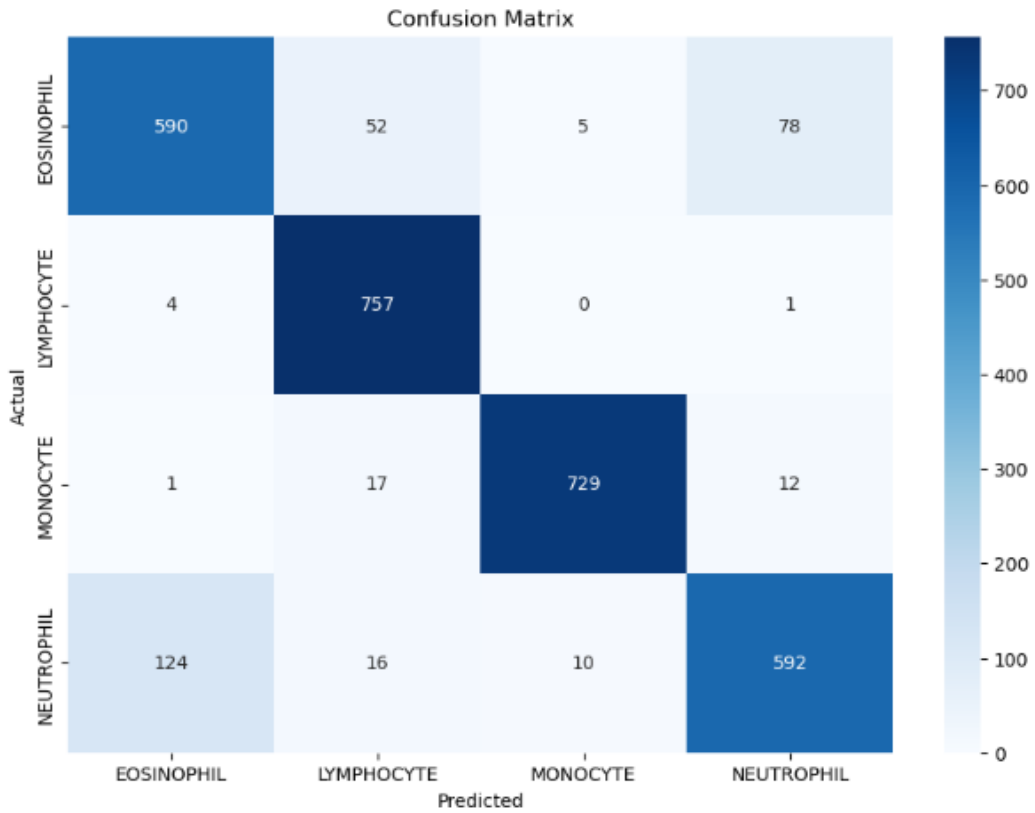




After training, the model is evaluated using the test dataset to measure its performance on unseen data.
The testing process helps identify overfitting and ensures the model generalizes well.
Key metrics like accuracy, precision, recall, and F1-score are used to assess the results.
Once validated, the saved model is used for real-time predictions.
During prediction, a new image is input through the Flask interface or directly in code.
The image is preprocessed (resized and normalized) to match the model's input format.
The model processes the image and predicts the probability scores for each class.
The class with the highest probability is selected as the final prediction.

The result is then displayed on the web interface for user interpretation.
This step ensures the model is functional and ready for deployment in real-world applications.

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

class_labels = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']


cm = confusion_matrix(y_test, pred2)

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues')

plt.xticks(ticks=[0.5, 1.5, 2.5, 3.5], labels=class_labels)
plt.yticks(ticks=[0.5, 1.5, 2.5, 3.5], labels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```



The confusion matrix visualizes the model's classification performance across four blood cell types.
It shows a high accuracy in predicting **Lymphocytes** and **Monocytes**, with 757 and 729 correct predictions respectively.

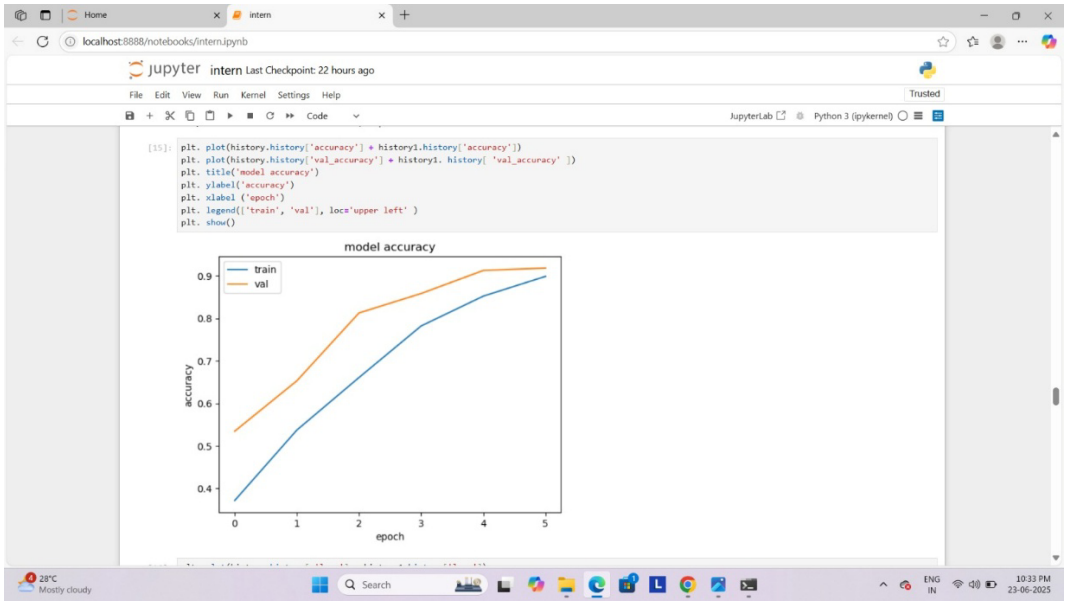
Some misclassifications occurred, such as Neutrophils predicted as Eosinophils (124 instances).
This helps identify which classes the model confuses most, guiding further tuning.
Overall, the matrix indicates strong performance with room for improvement in distinguishing similar cell types.

Conclusion:

The HematoVision project successfully demonstrates the application of deep learning and transfer learning in the domain of medical image classification. By leveraging a dataset of 12,000 labeled blood cell images and employing a pre-trained CNN model like MobileNetV2, the system achieves high accuracy in classifying blood cells into four categories: eosinophils, lymphocytes, monocytes, and neutrophils. The use of transfer learning significantly reduces training time and computational cost while maintaining strong performance, making it ideal for practical deployment in real-world healthcare settings.

In addition to model accuracy, the project emphasizes usability and accessibility. The trained model is integrated into a Flask-based web application with a user-friendly HTML interface, allowing users to upload images and receive instant predictions. This makes HematoVision not only useful for clinical diagnostics but also highly relevant for telemedicine and remote healthcare, where specialists may not be physically present. The system also has great potential in medical education by offering students and lab technicians a practical tool for learning blood cell classification.

Overall, HematoVision highlights the transformative potential of AI in healthcare diagnostics. The project combines effective data processing, model training, evaluation, and deployment into a single, cohesive pipeline. While the current model performs well, future enhancements—such as expanding the dataset, improving class differentiation, and enabling mobile integration—can further strengthen its impact. HematoVision stands as a robust and scalable solution that bridges the gap between advanced machine learning and real-world medical applications.


Home
intern
localhost:8888/notebooks/intern.ipynb
jupyter intern Last Checkpoint: 22 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[15]:
plt. plot(history.history['accuracy'] + history1.history['accuracy'])
plt. plot(history.history['val_accuracy'] + history1. history['val_accuracy'])
plt. title('model accuracy')
plt. ylabel('accuracy')
plt. xlabel ('epoch')
plt. legend(['train', 'val'], loc='upper left')
plt. show()
model accuracy
train
val
accuracy
epoch
0.9
0.8
0.7
0.6
0.5
0.4
0
1
2
3
4
5
28°C
Mostly cloudy
Search
ENG
IN
10:33 PM
23-06-2025