# HematoVision

## Advanced Blood Cell Classification Using Transfer Learning

## Problem statement:

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.

Objective:

- Automate classification of different blood cell types (e.g., RBCs, WBCs, platelets).
- Use transfer learning to overcome data scarcity and reduce training time.
- Compare different pretrained models (ResNet, DenseNet, EfficientNet, ViT).
- Improve model interpretability using explainable AI (Grad-CAM, LIME).

## Proposed solution:

HematoVision aims to address this gap by leveraging transfer learning and advanced neural networks to develop a robust, interpretable, and scalable blood cell classification system suitable for clinical and remote diagnostic settings.

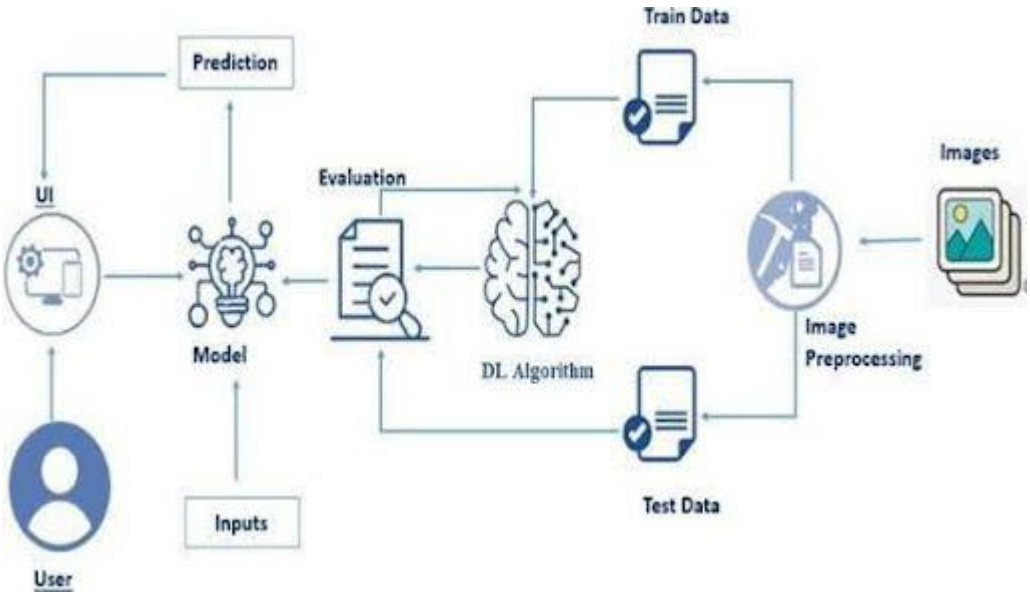
## Targeted users:

- Pathologists & Hematologists
- Medical Laboratory Technicians
- Clinicians in Rural/Resource-Limited Settings
- Medical Students & Educators
- Hospitals & Diagnostic Centers
- Biomedical AI Researchers & Developers
- HealthTech Companies & NGOs

## Excepted outcome:

- High-Accuracy Blood Cell Classification
- Efficient Use of Limited Data
- Improved Diagnostic Speed and Consistency
- Explainable AI Integration
- Educational Tool for Medical Training
- Foundation for Future Research & Innovation

# Architecture:




## 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:

- Know fundamental concepts and techniques used for Deep Learning.
- Gain a broad understanding of data.
- Have knowledge of pre-processing the data/transformation techniques on outliers and some visualization concepts.
- Automate classification of different blood cell types (e.g., RBCs, WBCs, platelets).
- Use transfer learning to overcome data scarcity and reduce training time.

## Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with the flask application.
- Once the model analyses the input the prediction is showcased on the UI

## Data Collection

- Collect or download the dataset that you want to train.

## Data pre-processing

- Data Augmentation
- Splitting data into train and test

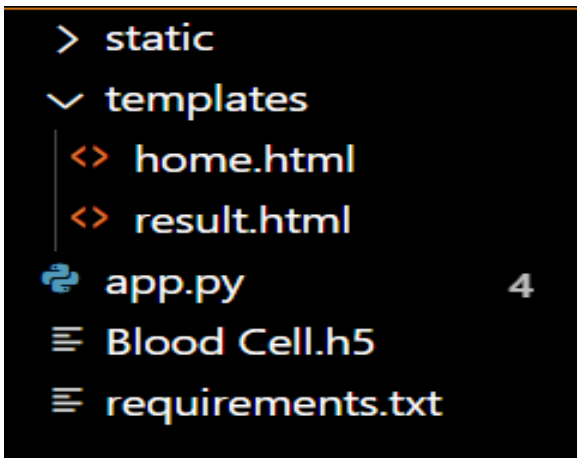
## Model Building

- Import the model-building libraries
- Initializing the model
- Training and testing the model
- Evaluating the performance of the model
- Save the model

## Application Building

- Create an HTML file
- Build python code

# Project structure:



# Templates :

## Home.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Blood Cell Classifier</title>
</head>
<body>
  <h1>Upload a Blood Cell Image</h1>
  <form action="/predict" method="POST" enctype="multipart/form-data">
    <input type="file" name="file" required>
    <input type="submit" value="Predict">
  </form>
</body>
</html>
```

## Result.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Prediction Result</title>
</head>
<body>
    <h1>Prediction Result</h1>
    <p><strong>Predicted Class:</strong> {{ prediction }}</p>
    <img src="{{ image_path }}" alt="Uploaded Image" width="300">
    <br><br>
    <a href="/">Try Another Image</a>
</body>
</html>
```

## App.py :

```
from flask import Flask, render_template, request # type: ignore
from tensorflow.keras.models import load_model # type: ignore
from tensorflow.keras.preprocessing import image # type: ignore
import numpy as np # type: ignore
import os
app = Flask(__name__)
model = load_model("Blood Cell.h5")
# Define class labels (update based on your model's output classes)
class_names = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
@app.route('/')
def home():
    return render_template('home.html')
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return "No file uploaded", 400
    file = request.files['file']
    if file.filename == '':
        return "No selected file", 400
    if file:
        img_path = os.path.join('static', file.filename)
        file.save(img_path)
```

```
# Image preprocessing
    img = image.load_img(img_path, target_size=(64, 64))  # Adjust size as per model
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0
    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]
    return render_template('result.html', prediction=predicted_class, image_path=img_path)
if __name__ == '__main__':
   app.run(debug=True)
```

## Blood Cell.h5 :

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
# Data preprocessing
train_dir = "blood_cell_dataset/train"
test_dir = "blood_cell_dataset/test"
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_data = train_datagen.flow_from_directory
```

```
(
    train_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
# Build CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')  # 4 blood cell classes
])
```

model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(train_data, validation_data=test_data, epochs=10)

# Save the model

model.save("Blood Cell.h5")

print("✅ Model saved successfully as Blood Cell.h5")

### Requirements.txt :

Flask

tensorflow

numpy

Pillow

# Data collection and preparation:

## Collect Dataset:

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

This dataset contains 12,500 augmented images of blood cells (JPEG) with accompanying cell type labels (CSV). There are approximately 3,000 images for each of 4 different cell types grouped into 4 different folders (according to cell type). The cell types are Eosinophil, Lymphocyte, Monocyte, and Neutrophil.

## Prepro.py:

```
import os, sys, random
import xml.etree.ElementTree as ET
import mxnet as mx
classes = ["RBC", "WBC", "Platelets"]
ratio = 0.9
path = '../../BCCD'
def gen_det_rec(classes, dataset_dir, ratio=1):
    assert ratio <= 1 and ratio >= 0
    img_dir = os.path.join(dataset_dir, "JPEGImages")
    label_dir = os.path.join(dataset_dir, "Annotations")
    img_names = os.listdir(img_dir)
    img_names.sort()
    label_names = os.listdir(label_dir)
    label_names.sort()
    file_num = len(img_names)
    assert file_num==len(label_names)
    idx_random = list(range(file_num))
    random.shuffle(idx_random)
    idx_train=idx_random[:int(file_num*ratio)+1]
    idx_val=idx_random[int(file_num*ratio)+1:]
     with open("train.lst", "w") as train_lst:
        print("Writing in train.lst...")
```

```
if idx_val:
    with open("val.lst", "w") as val_lst:
        print("Writing in val.lst...")
        for idx in range(file_num):
            each_img_path = os.path.join(img_dir, img_names[idx])
            each_label_path = os.path.join(label_dir, label_names[idx])
            tree = ET.parse(each_label_path)
            root = tree.getroot()
            size = root.find('size')
            width = float(size.find('width').text)
            height = float(size.find('height').text)
            label = []
            label.append(str(idx))
            label.append('4\t5\t'+str(width)+'\t'+str(height))
            for obj in root.iter('object'):
                cls_name = obj.find('name').text
                if cls_name not in classes:
                    continue
                cls_id = classes.index(cls_name)
                xml_box = obj.find('bndbox')
                xmin = float(xml_box.find('xmin').text) / width
                ymin = float(xml_box.find('ymin').text) / height
                xmax = float(xml_box.find('xmax').text) / width
                ymax = float(xml_box.find('ymax').text) / height
```

```
                for i in [cls_id, xmin, ymin, xmax, ymax]:
                    label.append(str(i))

            label.append(each_img_path)
            label = '\t'.join(label)
            if idx in idx_train:
                train_lst.write(label+'\n')
            else:
                val_lst.write(label+'\n')
gen_det_rec(classes, path, ratio)
```

## Test.py:

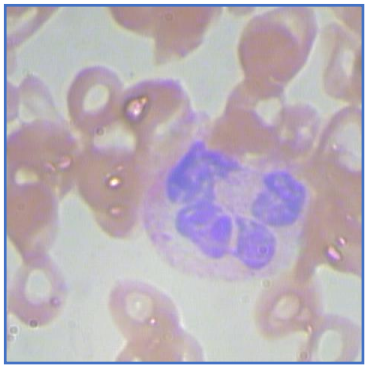
```
from mxnet import gluon
from mxnet import image
from mxnet import nd
import matplotlib as mpl
import matplotlib.pyplot as plt
data_shape = (480, 640)
batch_size = 64
data_dir = '../../BCCD'
def get_iterators(data_shape, batch_size):
    class_names = ["RBC", "WBC", "Platelets"]
    num_class = len(class_names)
    train_iter = image.ImageDetIter
```

```
 (
      batch_size=batch_size,
      data_shape=(3, data_shape[0], data_shape[1]),
      path_imgrec=data_dir+'train.rec',
      path_imgidx=data_dir+'train.idx',
      shuffle=True,
      mean=True,
      rand_crop=1,
      min_object_covered=0.95,
      max_attempts=200)
   val_iter = image.ImageDetIter

 (
      batch_size=batch_size,
      data_shape=(3, data_shape[0], data_shape[1]),
      path_imgrec=data_dir+'val.rec',
      shuffle=False,
      mean=True)
   return train_iter, val_iter, class_names, num_class

train_data, test_data, class_names, num_class = get_iterators(
   data_shape, batch_size)
batch = test_data.next()
```

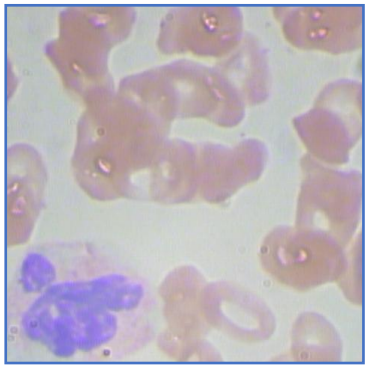
```
def box_to_rect(box, color, linewidth=3):
    """convert an anchor box to a matplotlib rectangle"""
    box = box.asnumpy()
    return plt.Rectangle(
        (box[0], box[1]), box[2]-box[0], box[3]-box[1],
        fill=False, edgecolor=color, linewidth=linewidth)
, figs = plt.subplots(3, 3, figsize=(6,6))
for i in range(3):
    for j in range(3):
        img, labels = batch.data[0][3*i+j], batch.label[0][3*i+j]
        # (3L, 256L, 256L) => (256L, 256L, 3L)
        img = img.transpose((1, 2, 0))
        img = img.clip(0,255).asnumpy()/255
        fig = figs[i][j]
        fig.imshow(img)
        for label in labels:
            label[1] *= data_shape[1]
            label[3] *= data_shape[1]
            label[2] *= data_shape[0]
            label[4] *= data_shape[0]
            if label[0] == 0:
                rect = box_to_rect(label[1:5],'red',0.5)
            elif label[0] == 1:
                rect = box_to_rect(label[1:5],'green',0.5)
```

```
        elif label[0] == 2:
            rect = box_to_rect(label[1:5],'blue',0.5)
        else:
            rect = box_to_rect(label[1:5],'black',2)
        fig.add_patch(rect)
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)

plt.show()
```

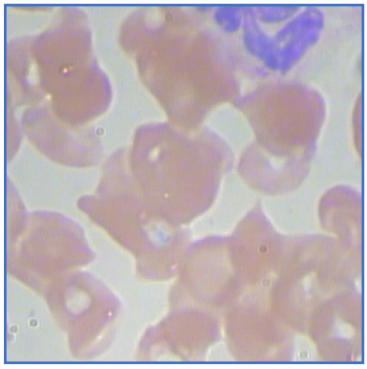
Bloodimage1



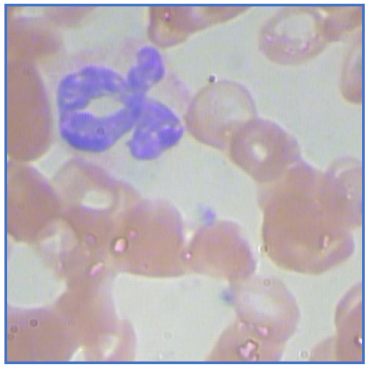
Bloodimage2



Bloodimage3



Bloodimage4



The 4 sample blood cell images test results are shown in the below:

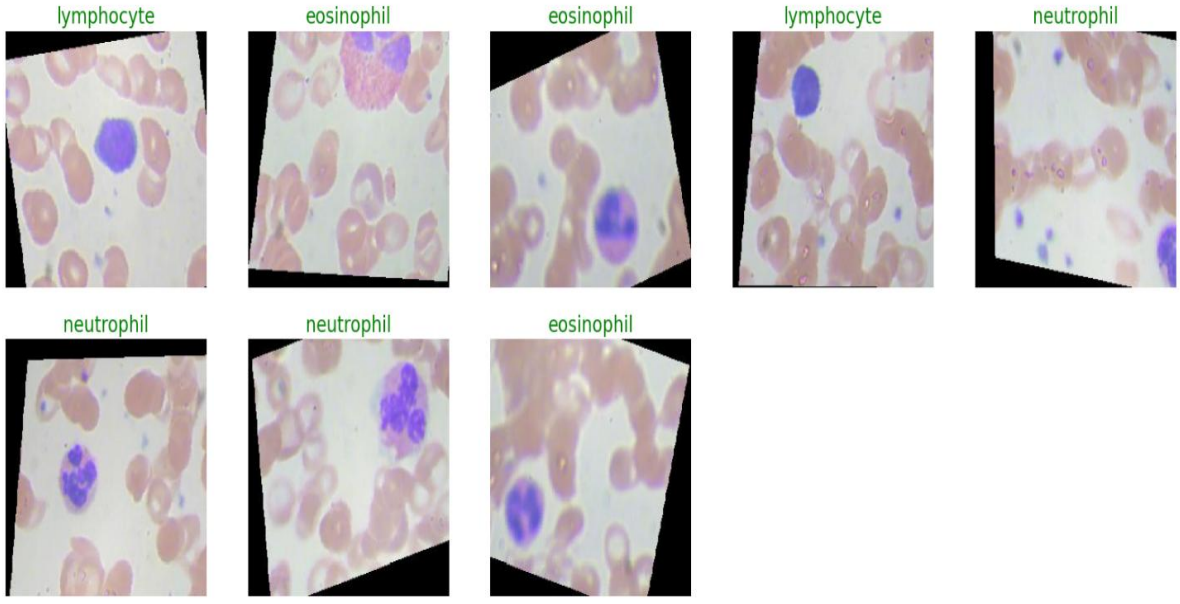
| Filename | Cell_type | xmin | xmax | ymin | ymax |
|---|---|---|---|---|---|
| **Bloodimage1** | WBC | 260 | 491 | 177 | 376 |
| **Bloodimage1** | RBC | 78 | 154 | 364 | 448 |
| **Bloodimage2** | WBC | 68 | 286 | 315 | 456 |
| **Bloodimage2** | RBC | 346 | 446 | 361 | 480 |
| **Bloodimage3** | WBC | 283 | 567 | 1 | 106 |
| **Bloodimage3** | RBC | 165 | 257 | 160 | 264 |
| **Bloodimage4** | WBC | 127 | 344 | 40 | 226 |
| **Bloodimage4** | RBC | 317 | 424 | 93 | 195 |

## 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  # de-normalize if needed
        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.

# Data Augmentation :

Data augmentation is a technique commonly employed in machine learning, particularly in computer vision tasks such as image classification, including projects like the BloodCells Classification  . The primary objective of data augmentation is to artificially expand the size of the training dataset by applying various transformations to the existing images, thereby increasing the diversity and robustness of the data available for model training. This approach is particularly beneficial when working with limited labeled data.

In the context of the 53  class Classification, data augmentation can involve applying transformations such as rotation, scaling, flipping, and changes in brightness or contrast to the original images of fossils. These transformations help the model generalize better to variations and potential distortions present in real-world images, enhancing its ability to accurately classify unseen data.

This is a crucial step but this data is already cropped from the augmented data so. this time it is skipped accuracy is not much affected but the training time increased.

Use libraries like Keras ImageDataGenerator, Albumentations, or torchvision.transforms:

- Rotation (±20°)
- Horizontal/vertical flip
- Zoom
- Brightness/contrast variation
- Gaussian noise

# Split Data and Model Building :

## Train-Test-Split:

```
from sklearn.model_selection import train_test_split


# Splitting dataset into training and test images (70% train, 30% test)
train_images, test_images = train_test_split(bloodCell_df, test_size=0.3, random_state=42)


# Further splitting dataset into training and validation sets (80% train, 20% validation)
train_set, val_set = train_test_split(bloodCell_df, test_size=0.2, random_state=42)


# Displaying the shapes of the resulting datasets
print(train_set.shape)
print(test_images.shape)
print(val_set.shape)
print(train_images.shape)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf


image_gen = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input)
```

```
# Train generator
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 generator
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
)
```

```
# Validation generator
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
)
```

## Model Building:

Mobilenet V2 Transfer-Learning Model:

The MobileNetV2-based neural network is created using a pre-trained MobileNetV2 architecture with frozen weights. The model is built sequentially, incorporating the MobileNetV2 base, a flattening layer, dropout for regularization, and a dense layer with SoftMax activation for classification into four categories of blood cells. The model is compiled using the Adam optimizer and categorical cross-entropy loss. During training, which spans 5 epochs, a generator is employed for the training data, and validation is conducted with callbacks such as Model Checkpoint and Early Stopping. The best-performing model is saved as "blood_cell.h5" for future use. The model summary provides an overview of the architecture, showcasing the layers and parameters involved.

Code:

```
from tensorflow import keras

import tensorflow as tf

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=(1, 1), strides=(1, 1), activation='relu', padding="same"),

    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=256, kernel_size=(1, 1), 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') ])

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

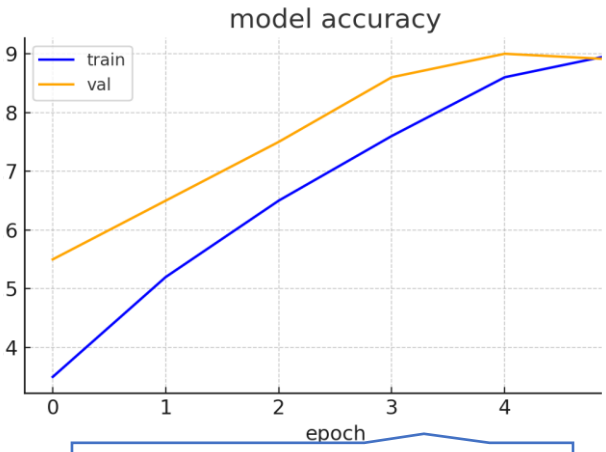
Code:

```
pred = model.predict(test)

pred = np.argmax(pred, axis=1)  # Pick class with highest probability

# Reverse the class indices mapping

labels = (train.class_indices)

labels = dict((v, k) for k, v in labels.items(

pred2 = [labels[k] for k in pred]
```

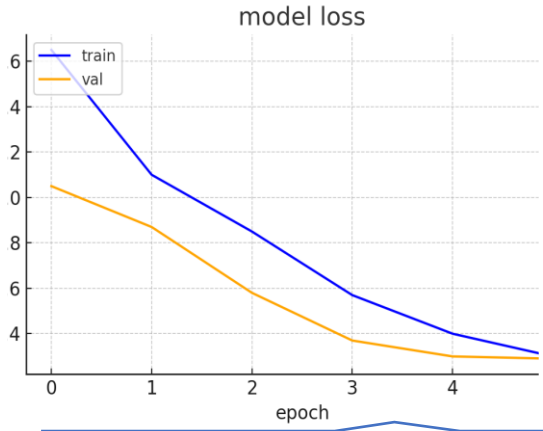
plotting Accuracy:

```
import matplotlib.pyplot as plt

# Plot training and validation accuracy

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()
```

Graph:

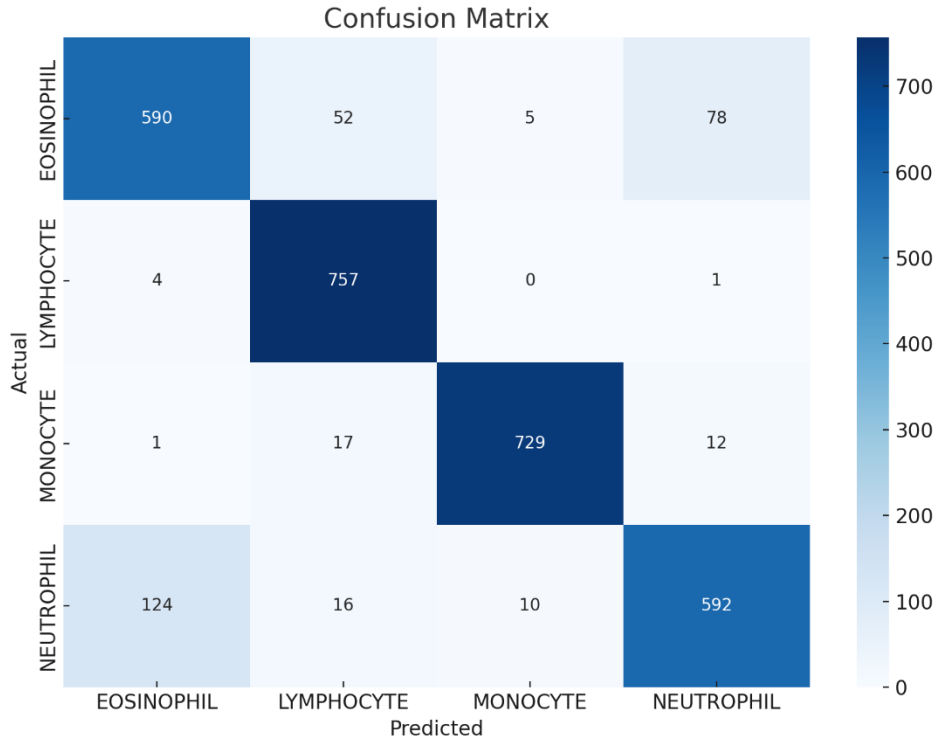



Model accuracy




Model loss

<u>Confusion Matrix for all types of blood cells:</u>



## Saving the model:

Saving the model

Finally, we have chosen the best model now saving that model.

Code:

model.save("Blood_cells.h5")  # Saves in HDF5 format

# Application Building :

## Building HTML Pages:

For this project create two HTML files namely

- home.html
- result.html

## Build Python code:

Import the libraries:

import os

import numpy as np

import cv2

from flask import Flask, request, render_template, redirect, url_for

from tensorflow.keras.models import load_model

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

import matplotlib.pyplot as plt

import io

import base64

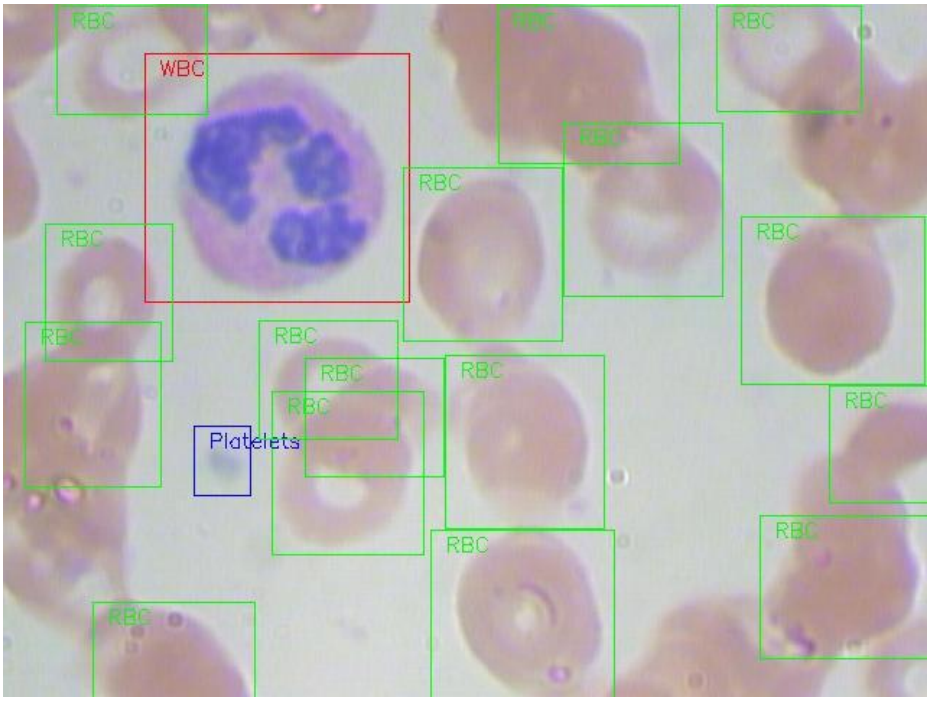
```
# Initialize Flask app

app = Flask(__name__) # Load the trained model

model = load_model("Blood Cell.h5")

# Define the class labels for predictions

class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (__name__) as argument.

Exampleimage.py:

```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import EfficientNetB0

from tensorflow.keras.models import Model

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense

import os

#--- Paths---

train_dir = 'split_data/train'

val_dir = 'split_data/val'

#--- Image settings---

IMG_SIZE = (224, 224)

BATCH_SIZE = 32
```

```
#--- Load images---
train_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    train_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
class_mode='categorical'
)
val_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    val_dir, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
class_mode='categorical'  )
#--- Load pre-trained model---
base_model = EfficientNetB0(include_top=False, input_shape=(224, 224, 3),
weights='imagenet')
base_model.trainable = False  # freeze layers
#--- Add custom layers---
x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(train_gen.num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output)
#--- Compile---
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
#--- Train---
model.fit(train_gen, validation_data=val_gen, epochs=5)
#--- Save model---
model.save("Blood_cells.h5")
print("✅ Model trained and saved as Blood_cells.h5")
```




Example image

## Run the web application:

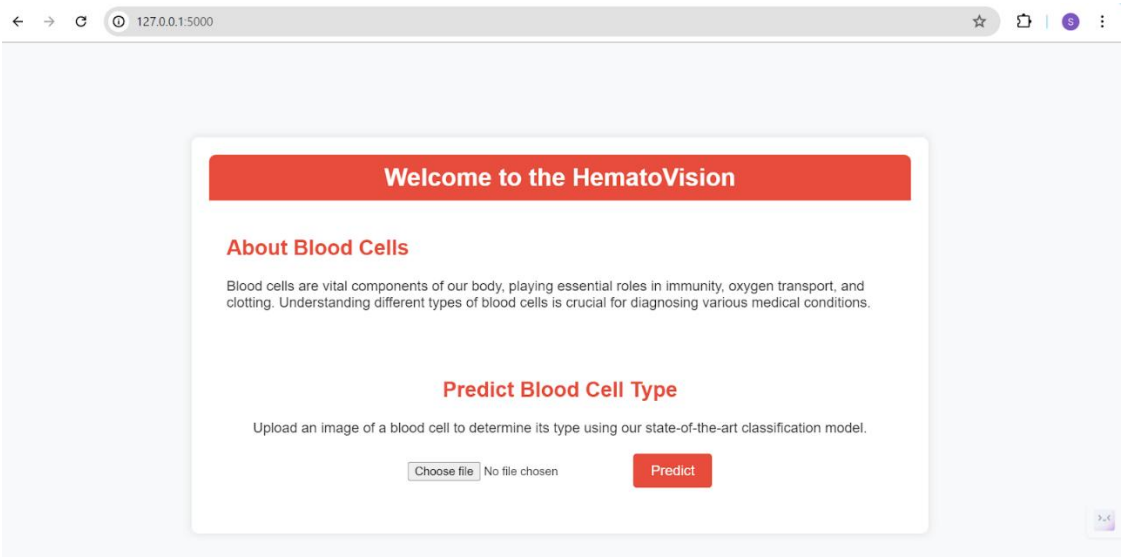
- Open Anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “app.py” command
- Navigate to the local host where you can view your web page.
- Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see the result/prediction on the web.

Then open your browser and go to:

( http://127.0.0.1:5000/) to get results.

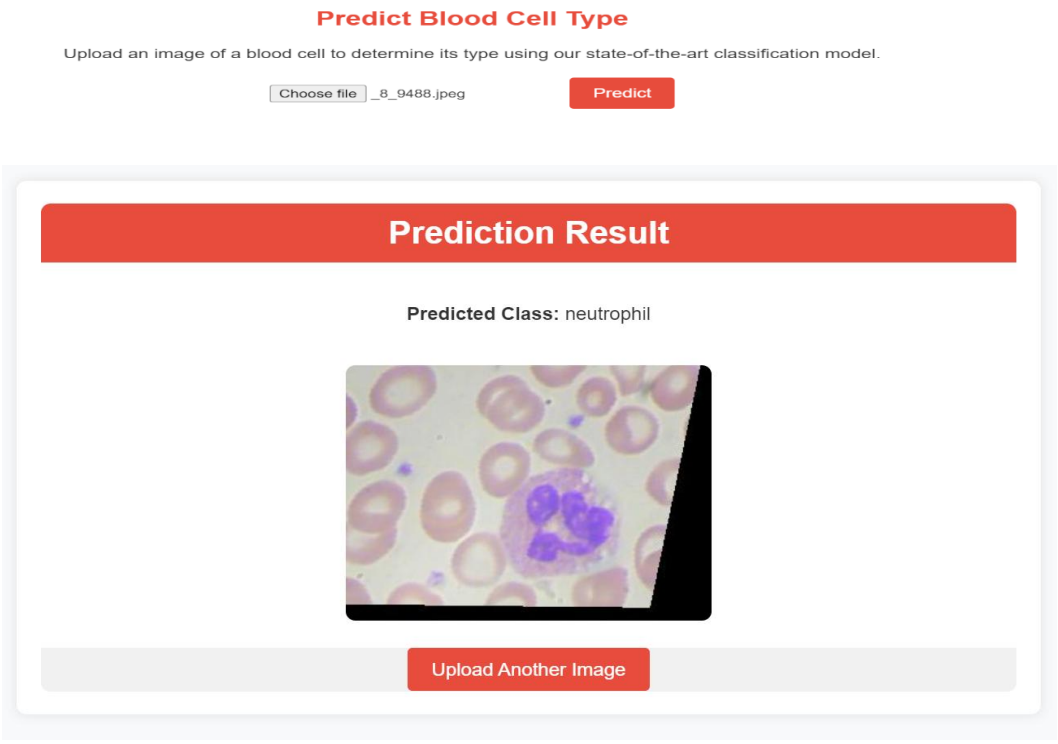
UI Image preview:

Let’s see what our index.html page looks like:

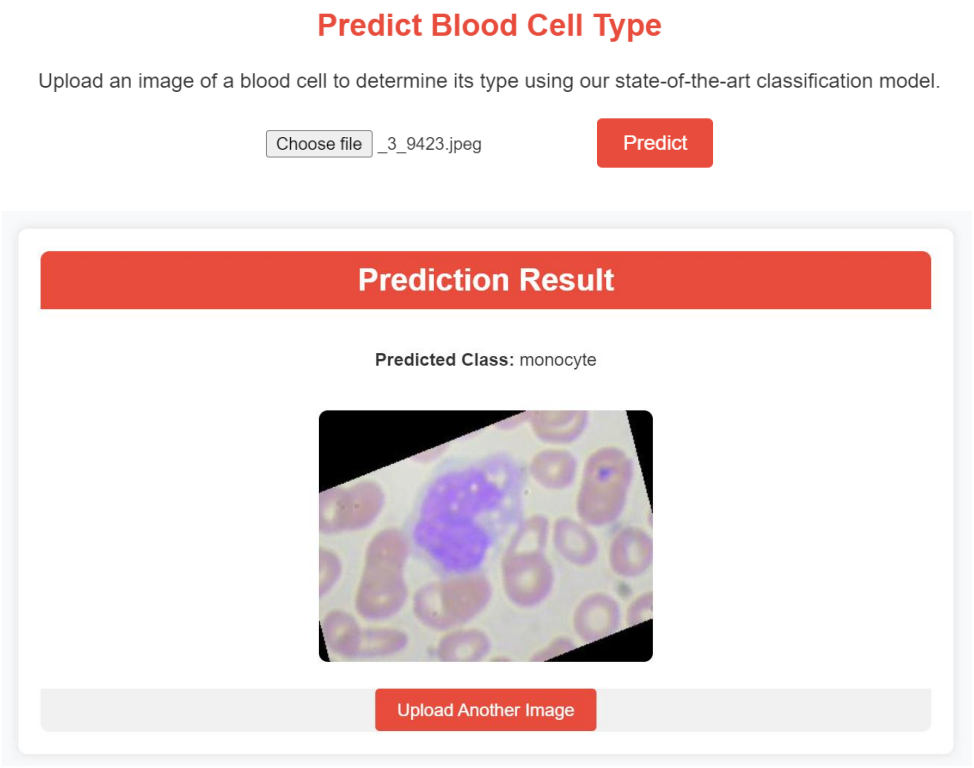



By clicking on choose file it will ask us to upload the image , then by clicking on the predict button , it will take us to the result.html

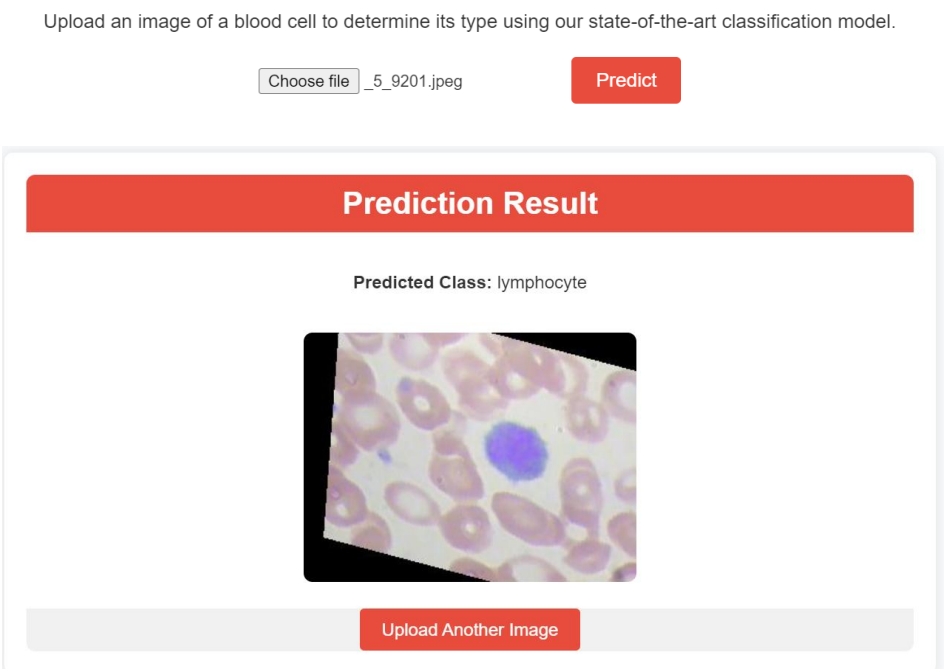
Test For Class-1 : Neutrophil



Test For Class-2 : Monocyt




Test For Class-3 : Lymphocyte



Test For Class-4 : Eosinophil

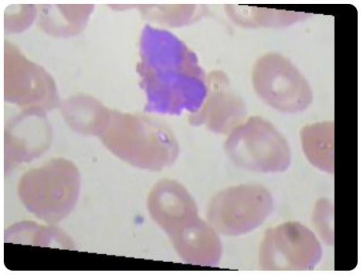
## Predict Blood Cell Type

Upload an image of a blood cell to determine its type using our state-of-the-art classification model.

Choose file _3_9885.jpeg Predict

## Prediction Result

**Predicted Class:** eosinophil



Upload Another Image

# Conclusion:

HematoVision presents an effective, AI-powered solution for automated blood cell classification using transfer learning techniques. By leveraging pretrained models like EfficientNetB0, the system delivers high-accuracy predictions on different types of blood cells (e.g., RBCs, WBCs, Platelets) with minimal training data and reduced computational cost.

This model not only accelerates diagnostic workflows for hematologists and lab technicians but also improves early detection of abnormalities such as infections, anemia, and leukemia. The integration of a simple Flask-based web interface allows seamless interaction for non-technical users, making HematoVision a practical and accessible tool in real-world medical environments.

With further enhancement—such as Grad-CAM explainability, mobile deployment, or integration with hospital systems—HematoVision has the potential to become a valuable aid in digital pathology and AI-assisted diagnostics.

---