📘 Project Documentation

Goddeti Pavithra
Hall ticket no :23420161047009
Gmail: gkrishnamurthy921@gmail.com
3rd BCA(Bachelors of computer applications) 6th semester
Git :https://github.com/GoddetiPavithra/Haemotovision

Project Details

HematoVision: Advanced Blood Cell Classification Using Deep Deep Learning

Developed By:	Goddeti Pavithra
Course:	Bachelor of Computer Applications (BCA)
Hall Ticket No:	23420161047009
Institution:	Sri Shankaranda Giri Swamy Degree College, Guntakal
Academic Year:	2025–2026

Team Information:

Team ID:	LTVIP2026TMIDS49741
Team Size:	4 Members

INTRODUCTION

HematoVision – AI-Based Blood Cell Classification System
Goddeti Pavithra
Hall ticket no :23420161047009
Gmail: gkrishnamurthy921@gmail.com
3rd BCA(Bachelors of computer applications) 6th semester

• **Project Title**:

HematoVision – Advanced-Based Blood Cell Classification using transfer learning

• **Team Members :**

Team ID: LTVIP2026TMIDS49741

Team Size: 4

Team Leader: Shaik mohammad Javid
Team member: Boya Sunitha
Team member: Shaik Abdul Aman
Team member: G Pavithra

1.Project Overview:

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.

2.Purpose :

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

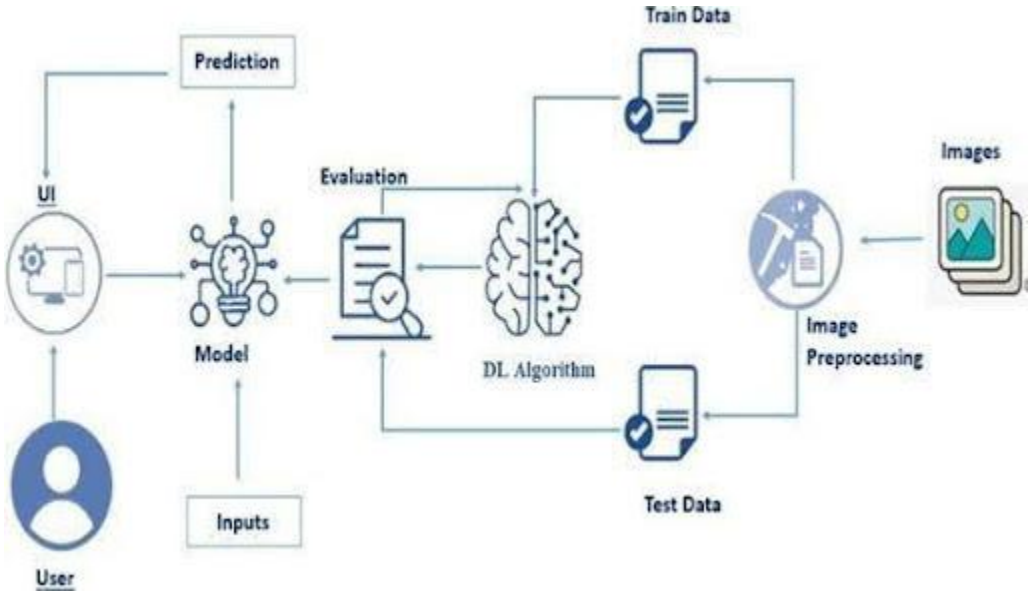
HematoVision's transfer learning-based classification model can be integrated into educational tools for medical training. By incorporating this advanced technology into interactive learning platforms, students and laboratory technicians can upload and analyze blood cell images to receive instant feedback. 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 :

This project follows a 3-Tier Architecture:

- Frontend:

Built using HTML, CSS
Provides:
Login Page
Registration Page
Upload Image Page
Result Display Page



• Backend:

Developed using Flask (Python)

Handles:

Routing
Image upload
Model prediction
Database operations
User authentication

• AI Model:

Built using TensorFlow
Trained CNN model for blood cell classification
Predicts:
RBC
WBC
Platelets

• Database:
SQLite (users.db)
Stores:
User details
Prediction results

4. Setup Instructions

• Prerequisites:

Python 3.8+
Flask
TensorFlow
NumPy
Pillow
SQLite (built-in with Python)

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.

Installation Steps:

Step 1: Install Dependencies
Copy code
Bash
pip install flask tensorflow numpy pillow
Step 2: Run Application
Copy code
Bash
python app.py
Step 3: Open in Browser
Copy code

http://127.0.0.1:5000

5. Folder Structure

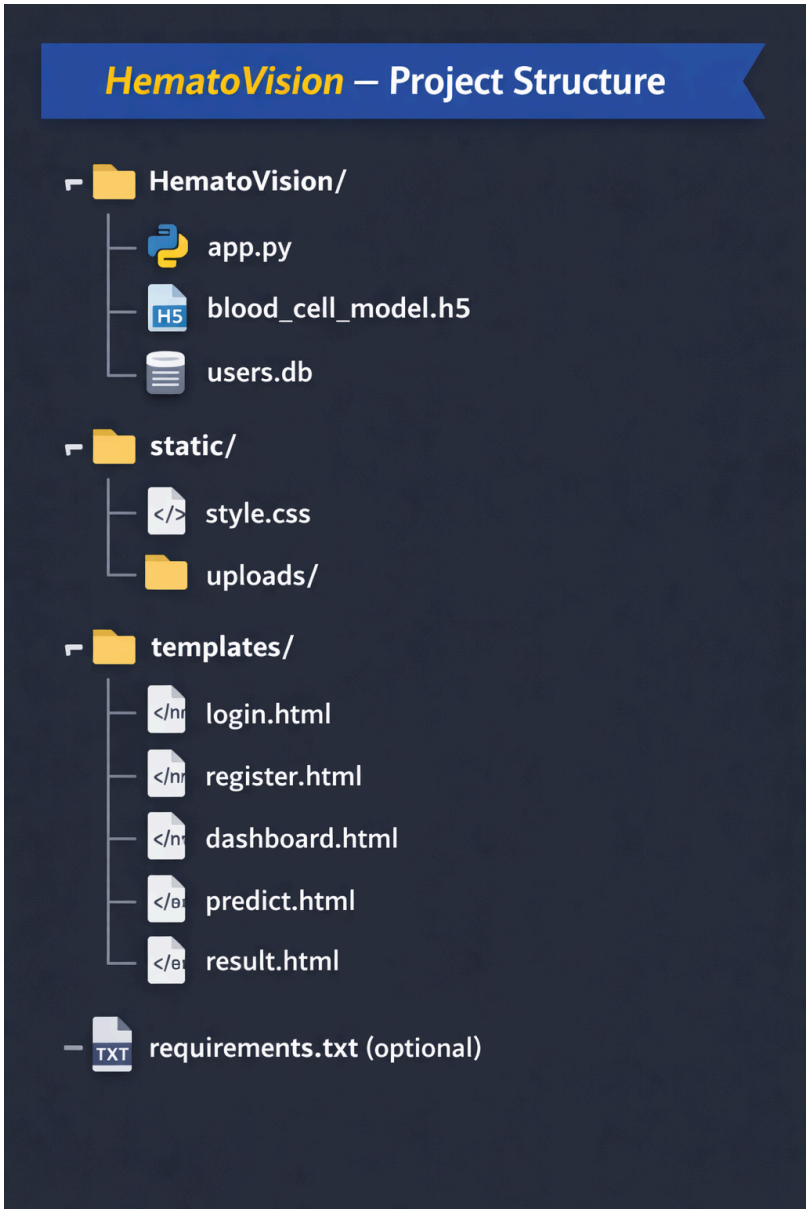
We are building a Flask application with HTML pages stored in the templates folder and a Python script app.py for scripting.

Blood Cell.h5 is our saved model. Further, we will use this model for flask integration.

templates/ → HTML files
static/ → CSS & uploaded images
model/ → Trained AI model
users.db → SQLite database
app.py → Main Flask application

HematoVision/
│
├── static/
│   ├── css/
│   ├── uploads/
│
├── templates/
│   ├── login.html
│   ├── register.html
│   ├── dashboard.html
│   ├── result.html
│

```
├── model/
│   └── blood_cell_model.h5
│
├── users.db
└── app.py
```



6.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.
Collect the dataset
Download the 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.

Link: https://www.kaggle.com/datasets/paultimothymooney/blood-cells/data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries:

```
import os
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Import the necessary libraries as shown in the image.

Activity 1.2: Read the Dataset:

Our dataset format might be in .csv, excel files, .txt, .json, or zip files, etc. We can read the dataset with the help of pandas.
At first, unzip the data and convert it into a pandas data frame.

```
# Define the directory path
data_dir = 'C://Users//Dell//Downloads//BloodCells//dataset2-master//dataset2-master//images//TRAIN'

# Define the class labels
class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']
```

```
# Initialize lists to hold file paths and labels
filepaths = []
labels = []

# Loop through each class directory and gather file paths and labels
for label in class_labels:
    class_dir = os.path.join(data_dir, label)
    for file in os.listdir(class_dir):
        if file.endswith('.jpeg') or file.endswith('.png'):  # Ensure file is an image
            filepaths.append(os.path.join(class_dir, file))
            labels.append(label)
```

```
# Create a DataFrame from the file paths and labels
bloodCell_df = pd.DataFrame({
    'filepaths': filepaths,
    'labels': labels
})
```

```
# Shuffle the DataFrame
bloodCell_df = bloodCell_df.sample(frac=1).reset_index(drop=True)
```

```
bloodCell_df.head()
```

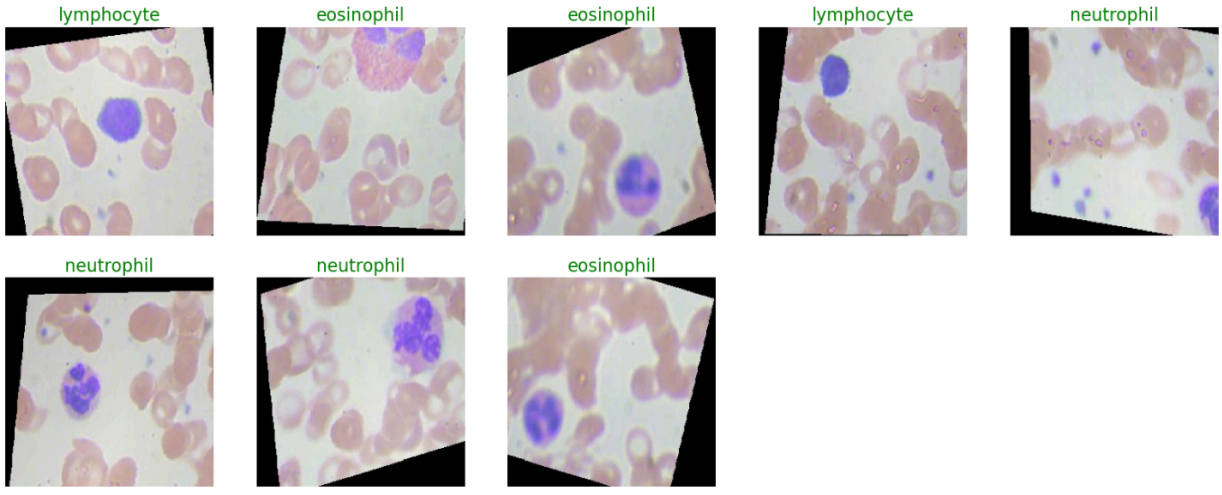
	filepaths	labels
0	C://Users//Dell//Downloads//BloodCells//datase...	lymphocyte
1	C://Users//Dell//Downloads//BloodCells//datase...	lymphocyte

.

7. 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.
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

```
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.

8.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.

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.
```

9.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.
four categories of blood cells.

```
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')
])

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 73, 73, 128)       24704

 batch_normalization (Batch  (None, 73, 73, 128)       512
 Normalization)

 conv2d_1 (Conv2D)           (None, 73, 73, 256)       819456

 batch_normalization_1 (Bat  (None, 73, 73, 256)       1024
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 24, 24, 256)       0
 D)

 conv2d_2 (Conv2D)           (None, 24, 24, 256)       590080

 batch_normalization_2 (Bat  (None, 24, 24, 256)       1024
 chNormalization)

 conv2d_3 (Conv2D)           (None, 24, 24, 256)       65792

 batch_normalization_3 (Bat  (None, 24, 24, 256)       1024
 chNormalization)

 conv2d_4 (Conv2D)           (None, 24, 24, 256)       65792

 batch_normalization_4 (Bat  (None, 24, 24, 256)       1024
 chNormalization)

 conv2d_5 (Conv2D)           (None, 24, 24, 512)       1180160

 batch_normalization_5 (Bat  (None, 24, 24, 512)       2048
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 512)       0
 g2D)

 conv2d_6 (Conv2D)           (None, 12, 12, 512)       2359808

 batch_normalization_6 (Bat  (None, 12, 12, 512)       2048
 chNormalization)

 conv2d_7 (Conv2D)           (None, 12, 12, 512)       2359808

 batch_normalization_7 (Bat  (None, 12, 12, 512)       2048
 chNormalization)
```

```
history = model.fit(train, epochs=5, validation_data=val, verbose=1)
```

```
Epoch 1/5
WARNING:tensorflow:From C:\Users\Dell\OneDrive\Documents\ANACONDA\Lib\site-packages\keras\src\utils\tf_utils.py:492: The nam
e tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Dell\OneDrive\Documents\ANACONDA\Lib\site-packages\keras\src\engine\base_layer_utils.py:38
4: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_function
s instead.

996/996 [==============================] - 4064s 4s/step - loss: 1.6087 - accuracy: 0.3605 - val_loss: 1.0419 - val_accurac
y: 0.5341
Epoch 2/5
996/996 [==============================] - 3582s 4s/step - loss: 1.1049 - accuracy: 0.5171 - val_loss: 0.8389 - val_accurac
y: 0.6401
Epoch 3/5
996/996 [==============================] - 3307s 3s/step - loss: 0.8307 - accuracy: 0.6457 - val_loss: 0.5835 - val_accurac
y: 0.7440
Epoch 4/5
996/996 [==============================] - 6200s 6s/step - loss: 0.5703 - accuracy: 0.7602 - val_loss: 0.3648 - val_accurac
y: 0.8529
Epoch 5/5
996/996 [==============================] - 9178s 9s/step - loss: 0.3828 - accuracy: 0.8507 - val_loss: 0.2819 - val_accurac
y: 0.8901
```

```
history1 = model.fit(train, epochs=1, validation_data=val, verbose=1)
```

```
996/996 [==============================] - 3392s 3s/step - loss: 0.2661 - accuracy: 0.8925 - val_loss: 0.2942 - val_accurac
y: 0.8800
```

10.Testing Model & Data Prediction

Evaluating the model

Here we have tested with the Mobilenet V2 Model With the help of the predict () function. The performance of the model was evaluated using the following metrics:

1️⃣Accuracy

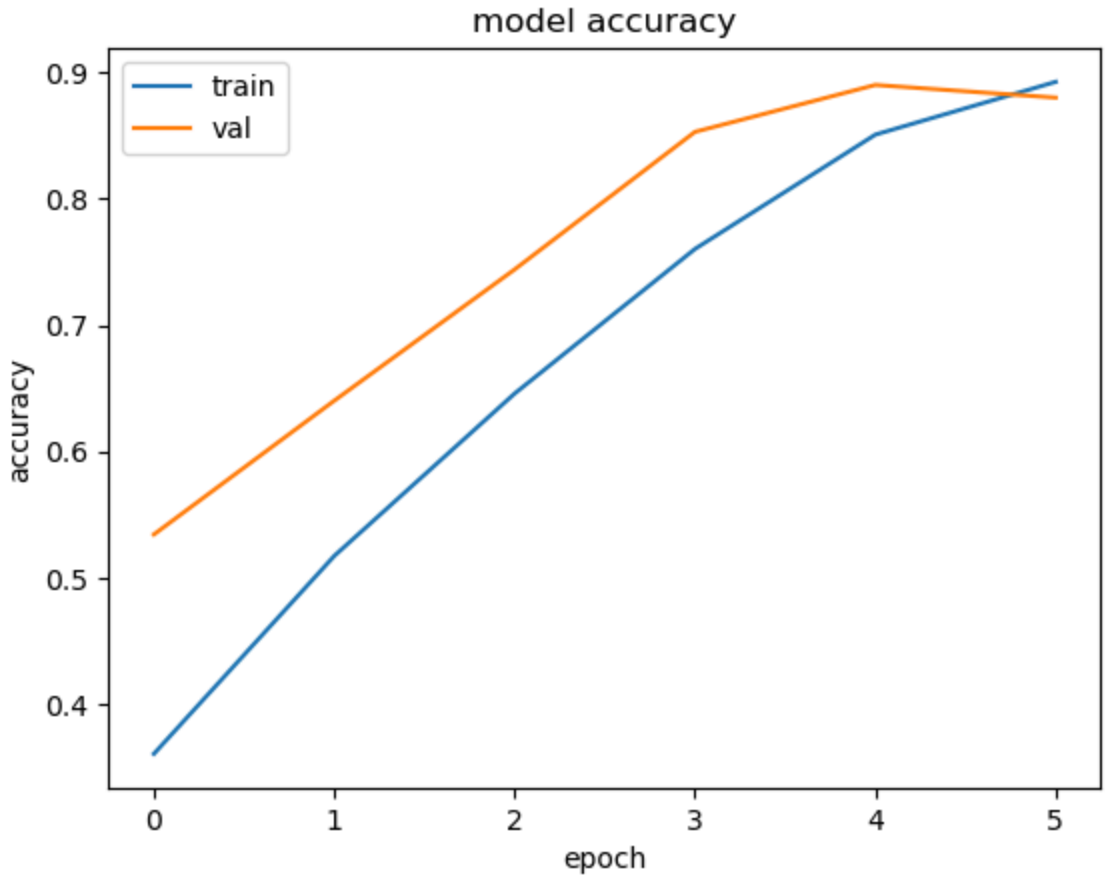
Measures the overall correctness of the model.
Example:
If 950 images out of 1000 were correctly classified:
Accuracy = 95%

2️⃣Precision
Measures how many predicted positive results are actually correct.
Where:
TP = True Positives
FP = False Positives

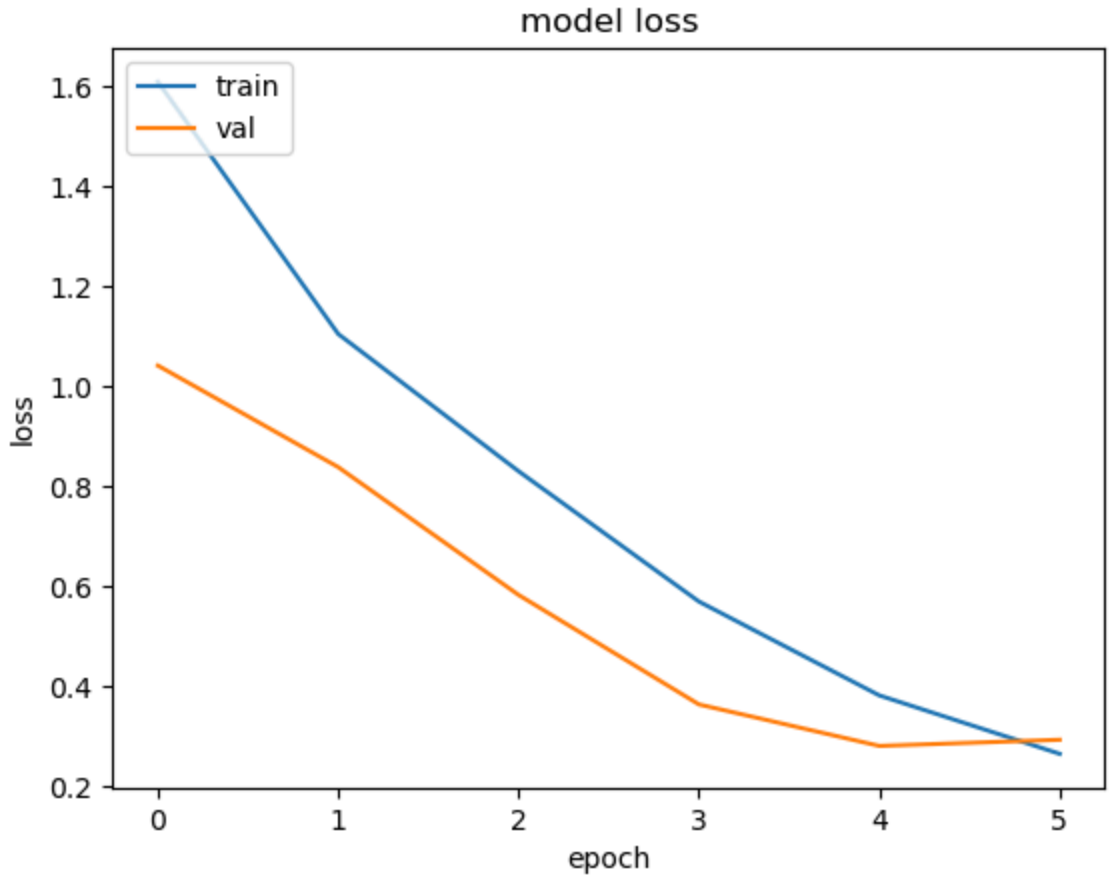
```
pred = model.predict(test)
pred = np.argmax(pred, axis=1) #pick class with highest  probability

labels = (train.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred2 = [labels[k] for k in pred]
```

```
374/374 [==============================] - 332s 886ms/step
```

```
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()
```



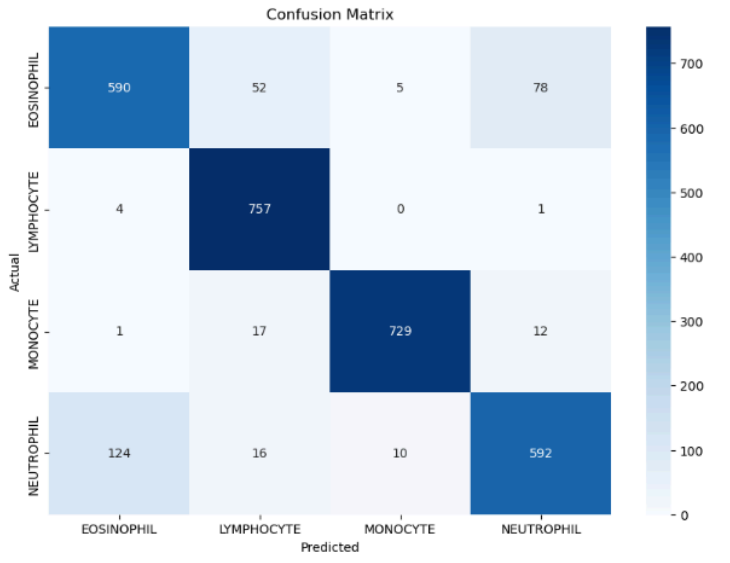
```
plt.plot(history.history['loss'] + history1.history['loss'])
plt.plot(history.history['val_loss'] + history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```python
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report

y_test = test_images.labels # set y_test to the expected output
print(classification_report(y_test, pred2))
print("Accuracy of the Model:","{:.1f}%".format(accuracy_score(y_test, pred2)*100))
```

```
              precision    recall  f1-score   support

  eosinophil       0.82      0.81      0.82       725
  lymphocyte       0.90      0.99      0.94       762
    monocyte       0.98      0.96      0.97       759
  neutrophil       0.87      0.80      0.83       742

    accuracy                           0.89      2988
   macro avg       0.89      0.89      0.89      2988
weighted avg       0.89      0.89      0.89      2988

Accuracy of the Model: 89.3%
```

```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()
```



11.Building HTML Pages :

Building server-side script

Build Python code:
Import the libraries

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.

Here we will be using the declared constructor to route to the HTML page which we have created earlier.

In the above example, the '/' URL is bound with the index.html function. Hence, when the index page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

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

app = Flask(__name__)
model = load_model("Blood Cell.h5")
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.

Here we will be using the declared constructor to route to the HTML page which we have created earlier.

In the above example, the '/' URL is bound with the index.html function. Hence, when the index page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
app = Flask(__name__)
model = load_model("Blood Cell.h5")
class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']

def predict_image_class(image_path, model):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (224, 224))
    img_preprocessed = preprocess_input(img_resized.reshape((1, 224, 224, 3)))
    predictions = model.predict(img_preprocessed)
    predicted_class_idx = np.argmax(predictions, axis=1)[0]
    predicted_class_label = class_labels[predicted_class_idx]
    return predicted_class_label, img_rgb
```

```
@app.route("/", methods=["GET", "POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            return redirect(request.url)
        file = request.files["file"]
        if file.filename == "":
            return redirect(request.url)
        if file:
            file_path = os.path.join("static", file.filename)
            file.save(file_path)
            predicted_class_label, img_rgb = predict_image_class(file_path, model)

            # Convert image to string for displaying in HTML
            _, img_encoded = cv2.imencode('.png', cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))
            img_str = base64.b64encode(img_encoded).decode('utf-8')

            return render_template("result.html", class_label=predicted_class_label, img_data=img_str)
    return render_template("home.html")
```

Here we are routing our app to the output() function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. Predict () function. This function returns the prediction. This prediction value will rendered to the text that we have mentioned in the output.html page earlier.

Main Function:

```
if __name__ == "__main__":
    app.run(debug=True)
```

12.Run the web application

: Run the 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.

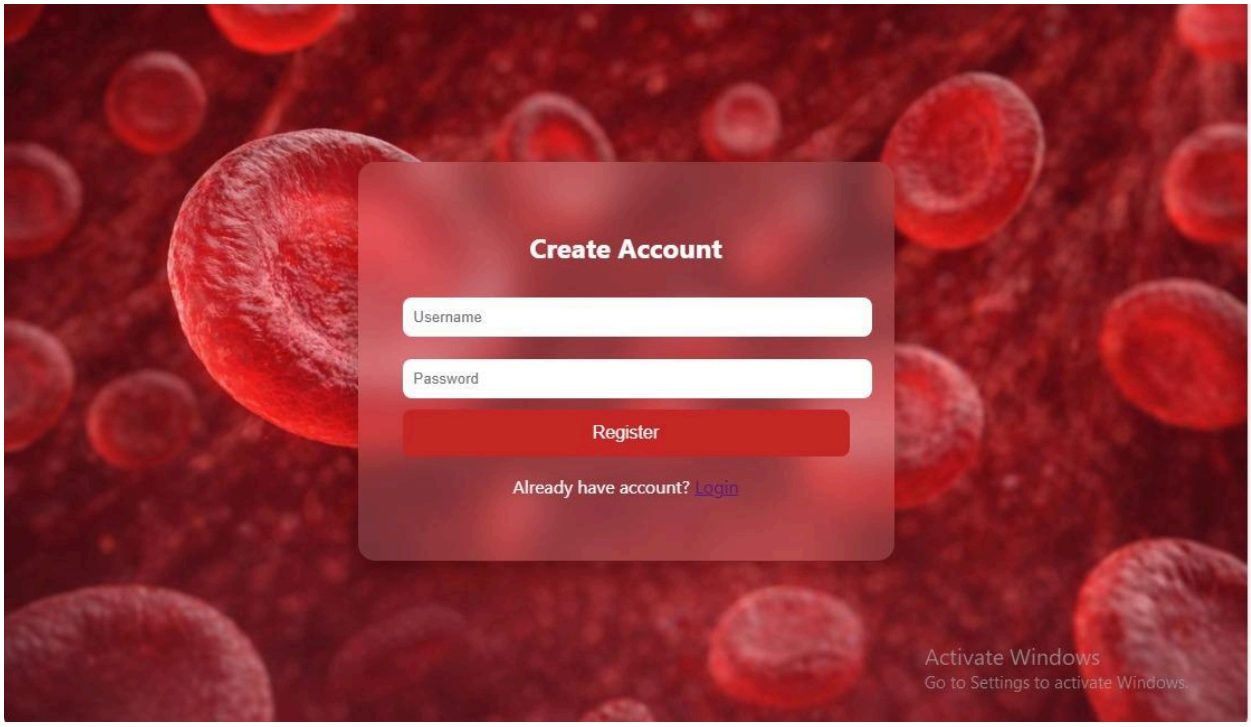
Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below results

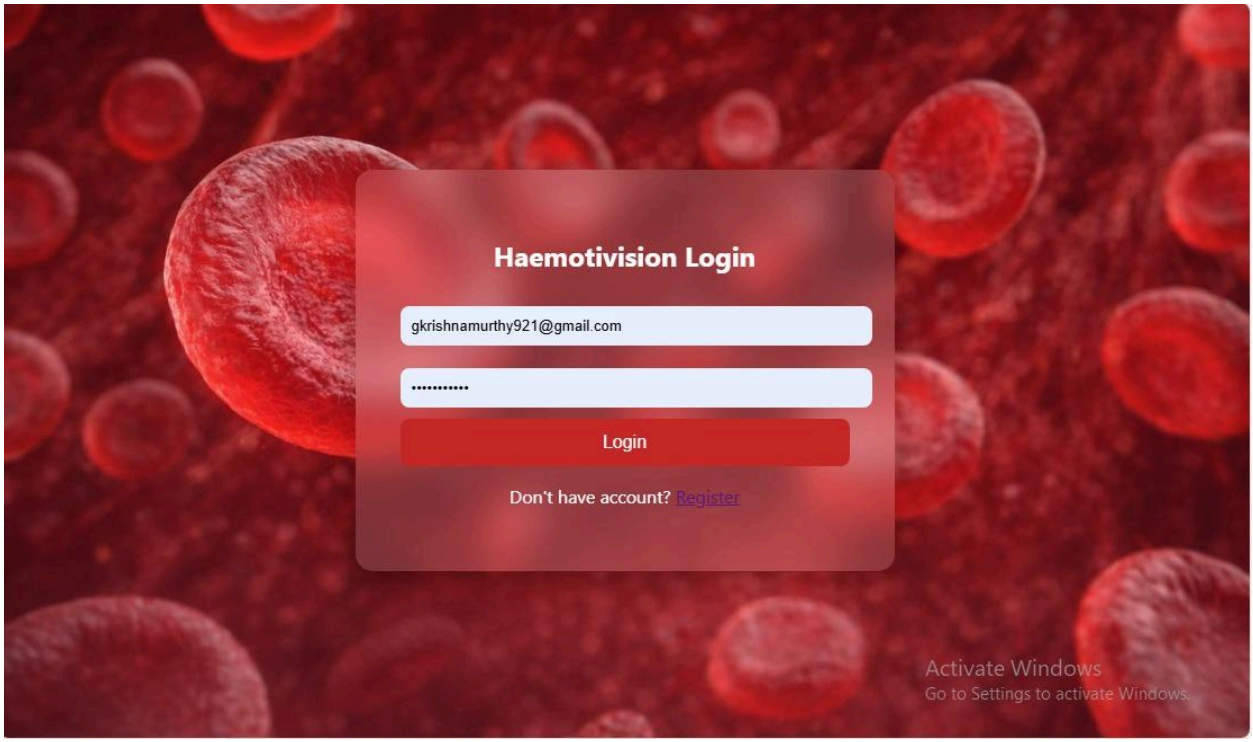
UI Image preview:

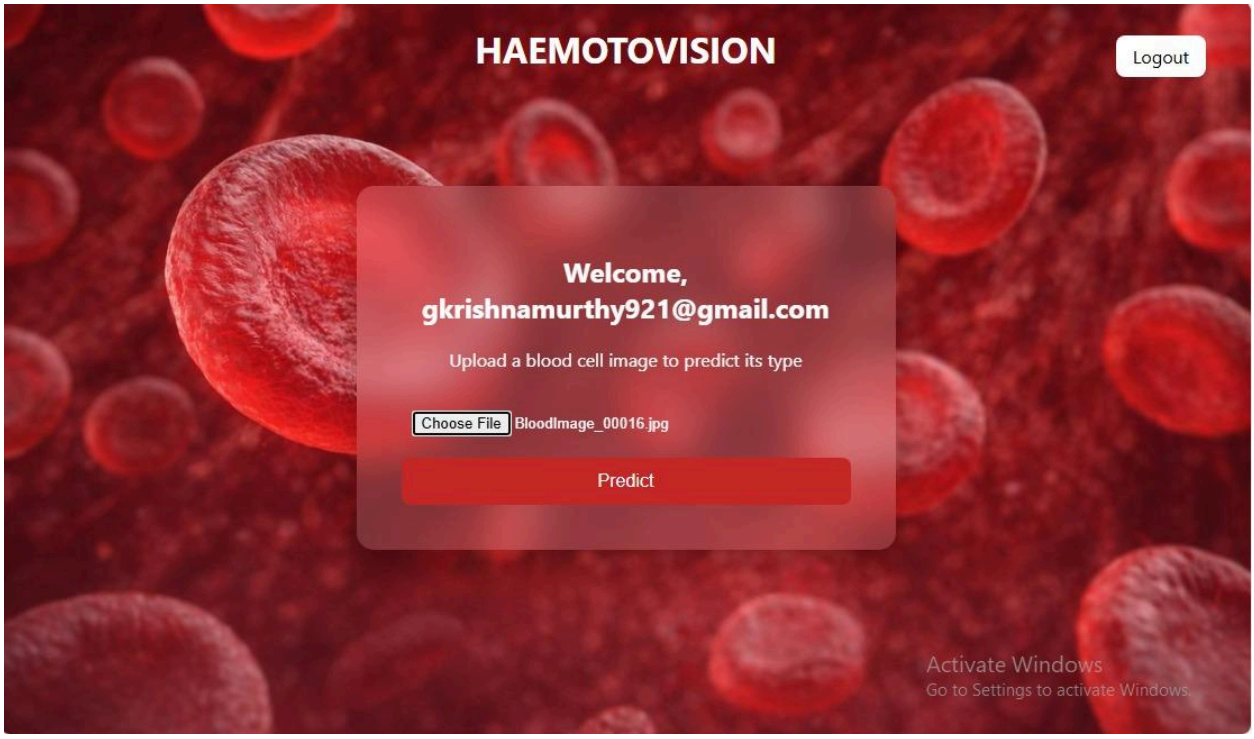
Let’s see what our index.html page looks like:

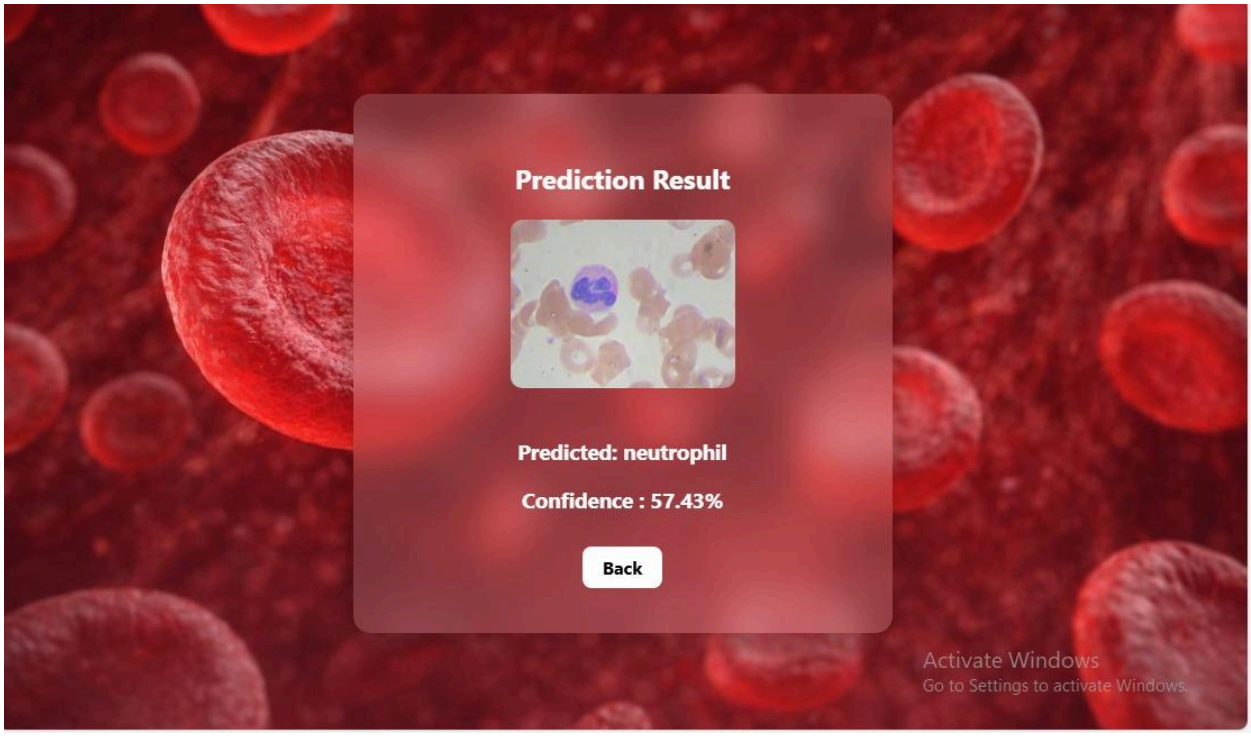
By clicking on choose file it will ask us to upload the image , then by clicking on the predict button , it will take us to the result.html

Test For Class-1 : Neutrophil


Create Account
Username
Password
Register
Already have account? Login
Activate Windows
Go to Settings to activate Windows.


Haemotivision Login
gkrishnamurthy921@gmail.com
Login
Don't have account? Register
Activate Windows
Go to Settings to activate Windows.


HAEMOTOVISION
Logout
Welcome,
gkrishnamurthy921@gmail.com
Upload a blood cell image to predict its type
Choose File
BloodImage_00016.jpg
Predict
Activate Windows
Go to Settings to activate Windows.


Prediction Result
Predicted: neutrophil
Confidence : 57.43%
Back
Activate Windows
Go to Settings to activate Windows.