

HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Team ID: LTVIP2026TMIDS55747

Team Size : 4

Team Leader : Mirla Durga Bhavani

Team member : Degala Lokesh Babu

Team member : Bondili Sri Bhavani Meghanath Singh

Team member : Panditi Akhil Kumar

Project Description :

Blood cell analysis is a critical diagnostic procedure used to detect infections, anemia, leukemia, and other hematological disorders. Traditionally, blood smear examination is performed manually under a microscope by a pathologist. This process is time-consuming, requires high expertise, and is prone to human error.

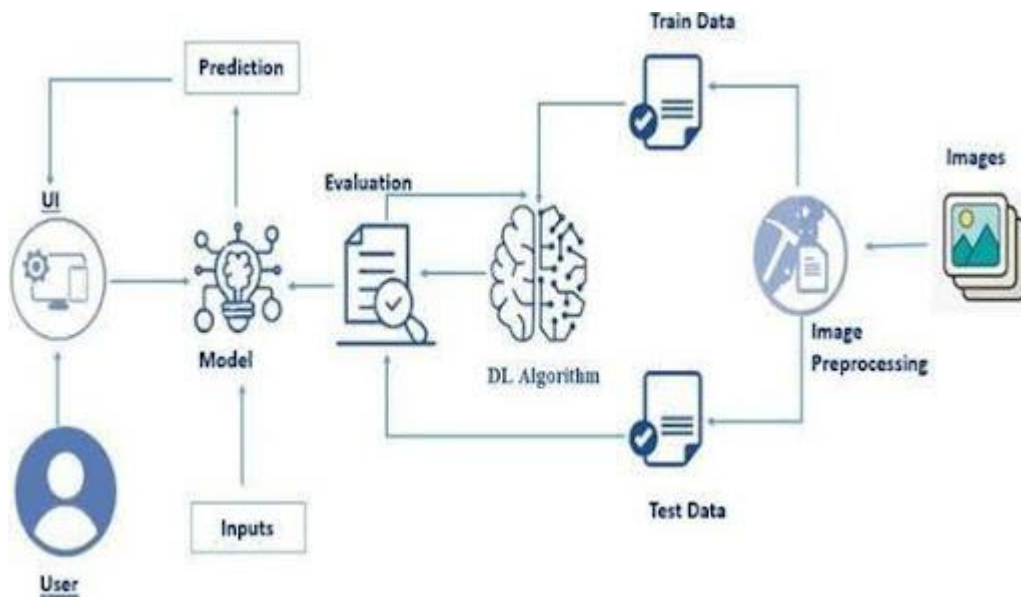
Hematovision is a deep learning-based system designed to automatically classify blood cells from microscopic images. The model analyzes blood smear images and predicts the type of cell (e.g., Eosinophil, Lymphocyte, Monocyte, Neutrophil).

By automating classification:

- Diagnosis becomes faster.
- Human error is reduced.
- Large volumes of samples can be processed efficiently.

The system uses Convolutional Neural Networks (CNN) for image classification and integrates the trained model into a Flask-based web application for real-time predictions.

Technical Architecture :



Pre-Requisites :

- Anaconda Navigator
- Python 3.x
- VS Code / PyCharm
- Jupyter Notebook

Python Packages:

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

Prior Knowledge Required :

- Machine Learning fundamentals
- Deep Learning basics
- CNN architecture

- Image preprocessing
- Overfitting and regularization
- Model evaluation metrics
- Flask basics

Project Objectives :

By completing this project, you will:

- Understand medical image classification.
- Learn CNN model building from scratch.
- Perform image preprocessing and augmentation.
- Evaluate deep learning models.
- Deploy a trained model using Flask.

Project Flow:

User uploads blood cell image.

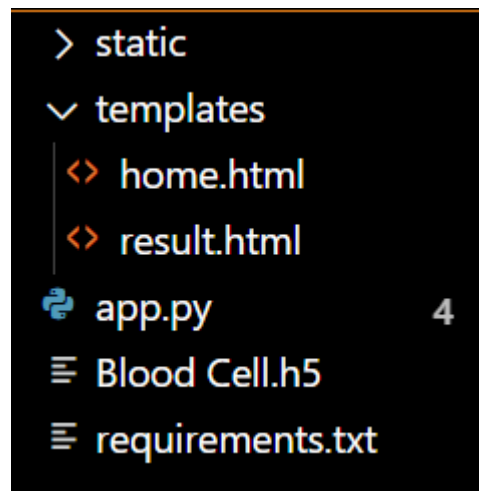
Image is preprocessed

Image is passed to trained CNN model

Model predicts cell type.

Result displayed on UI.

Project Structure:



Milestone 1: Data Collection

Activity 1: Download Dataset

Dataset used: Blood Cells Images

Source:

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

The dataset contains images of:

- Eosinophil
- Lymphocyte
- Monocyte
- Neutrophil

Images are organized into training and testing folders.

Milestone 2: Data Visualization and Analysis

Activity 1: Import Libraries

Import necessary libraries like numpy, pandas, matplotlib, seaborn.

```

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)

```

Activity 2: Read and Explore Dataset

- Check number of images per class.
- Verify image dimensions.
- Display sample images.

```
# 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)
```

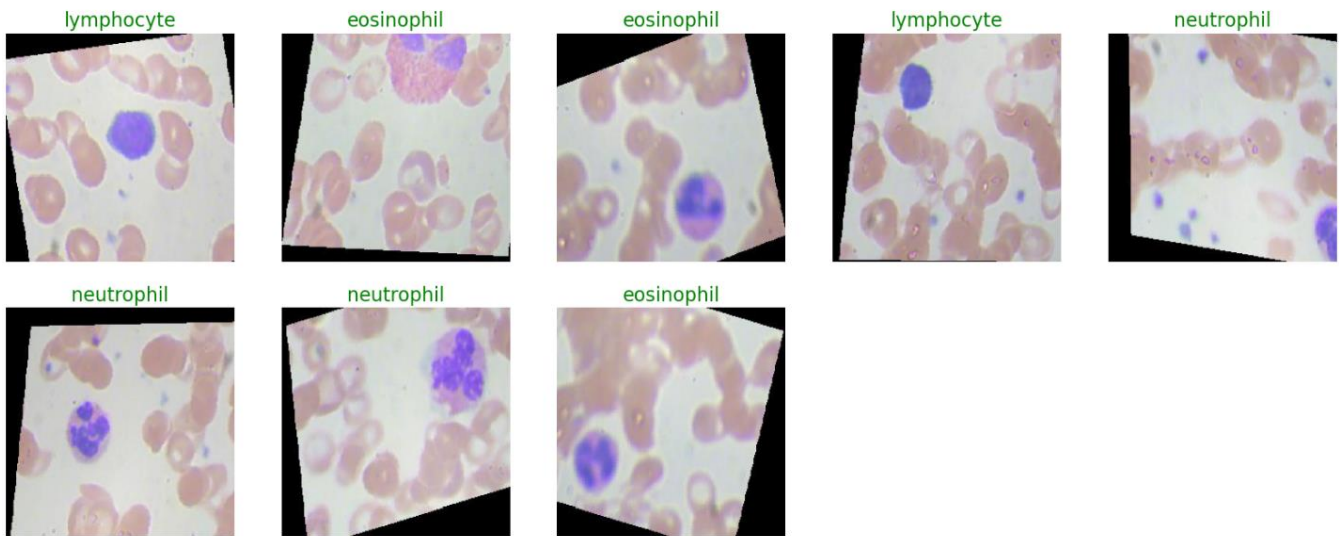
```
bloodCell_df.head()
```

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

Activity 3: Univariate Analysis

Analyze:

- Distribution of images per class.
- Pixel intensity distribution.



Activity 4: Bivariate Analysis

Example 1:

- Compare average pixel intensity between classes.

Example 2:

- Compare image size distribution across categories

Activity 5: Descriptive Analysis

- Count total images.
- Check class balance.
- Check resolution consistency.

Milestone 3: Data Preprocessing

Activity 1: Resize Images

All images resized to (64x64) or (128x128).

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data

- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Activity 2: Normalization

Pixel values scaled from 0–255 to 0–1

Activity 3: Label Encoding

Convert class names into numeric labels:

- Eosinophil → 0
- Lymphocyte → 1
- Monocyte → 2
- Neutrophil → 3

Activity 4: Data Augmentation

Use ImageDataGenerator:

- Rotation
- Zoom
- Horizontal flie

This prevents overfitting.

Activity 5: Train-Test Split

Dataset divided into:

- Training set
- Validation set
- Testing set

```
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.

Milestone 4: Model Building

Activity 1: Build CNN Model

Layers used:

- Conv2D
- MaxPooling2D
- Flatten
- Dense
- Dropout

```

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 Normalization)	(None, 73, 73, 128)	512
conv2d_1 (Conv2D)	(None, 73, 73, 256)	819456
batch_normalization_1 (Batch Normalization)	(None, 73, 73, 256)	1024
max_pooling2d (MaxPooling2D)	(None, 24, 24, 256)	0
conv2d_2 (Conv2D)	(None, 24, 24, 256)	590080
batch_normalization_2 (Batch Normalization)	(None, 24, 24, 256)	1024
conv2d_3 (Conv2D)	(None, 24, 24, 256)	65792
batch_normalization_3 (Batch Normalization)	(None, 24, 24, 256)	1024
conv2d_4 (Conv2D)	(None, 24, 24, 256)	65792
batch_normalization_4 (Batch Normalization)	(None, 24, 24, 256)	1024
conv2d_5 (Conv2D)	(None, 24, 24, 512)	1180160
batch_normalization_5 (Batch Normalization)	(None, 24, 24, 512)	2048
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 512)	0
conv2d_6 (Conv2D)	(None, 12, 12, 512)	2359808
batch_normalization_6 (Batch Normalization)	(None, 12, 12, 512)	2048
conv2d_7 (Conv2D)	(None, 12, 12, 512)	2359808
batch_normalization_7 (Batch Normalization)	(None, 12, 12, 512)	2048

```
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 name 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:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

996/996 [=====] - 4064s 4s/step - loss: 1.6087 - accuracy: 0.3605 - val_loss: 1.0419 - val_accuracy: 0.5341

Epoch 2/5

996/996 [=====] - 3582s 4s/step - loss: 1.1049 - accuracy: 0.5171 - val_loss: 0.8389 - val_accuracy: 0.6401

Epoch 3/5

996/996 [=====] - 3307s 3s/step - loss: 0.8307 - accuracy: 0.6457 - val_loss: 0.5835 - val_accuracy: 0.7440

Epoch 4/5

996/996 [=====] - 6200s 6s/step - loss: 0.5703 - accuracy: 0.7602 - val_loss: 0.3648 - val_accuracy: 0.8529

Epoch 5/5

996/996 [=====] - 9178s 9s/step - loss: 0.3828 - accuracy: 0.8507 - val_loss: 0.2819 - val_accuracy: 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_accuracy: 0.8800

Compile Model

Loss Function: Categorical Crossentropy

Optimizer: Adam

Metrics: Accuracy

Activity 3: Train Model

Train using:

Epochs

Batch size

Monitor training and validation accuracy.

Activity 4: Evaluate Model

Evaluation metrics:

- Accuracy
- Confusion Matrix
- Classification Report

Example 1:

- Training accuracy: 95%

Example 2:

- Testing accuracy: 90%

```

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

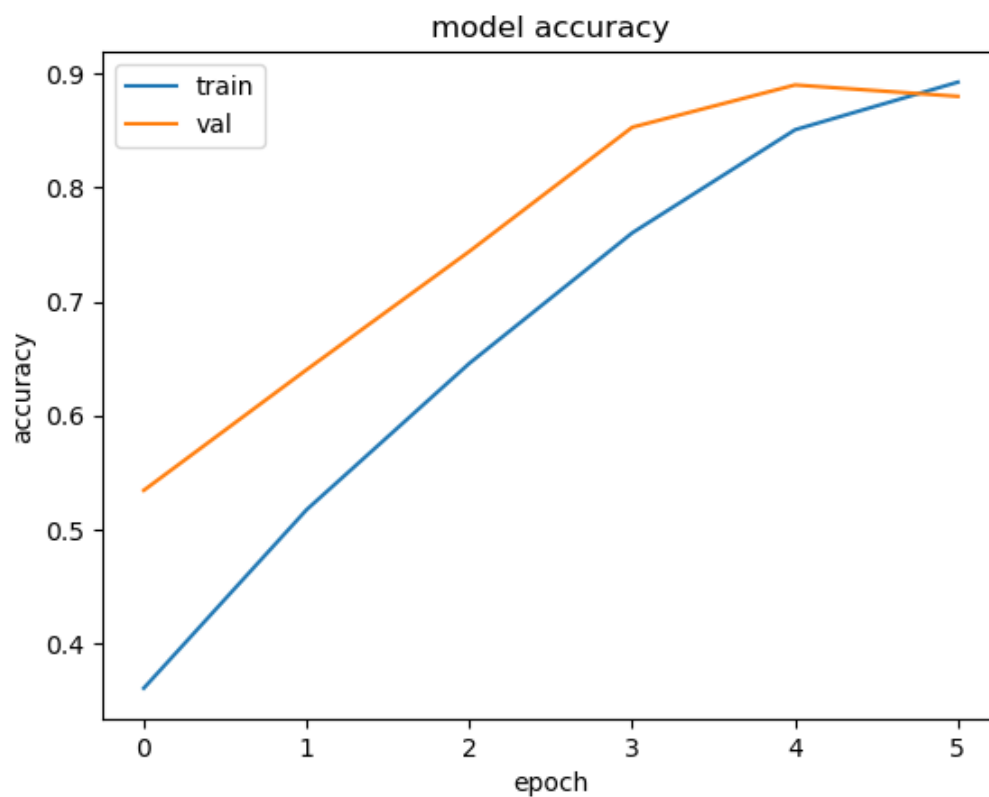
Accuracy of the Model: 89.3%

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

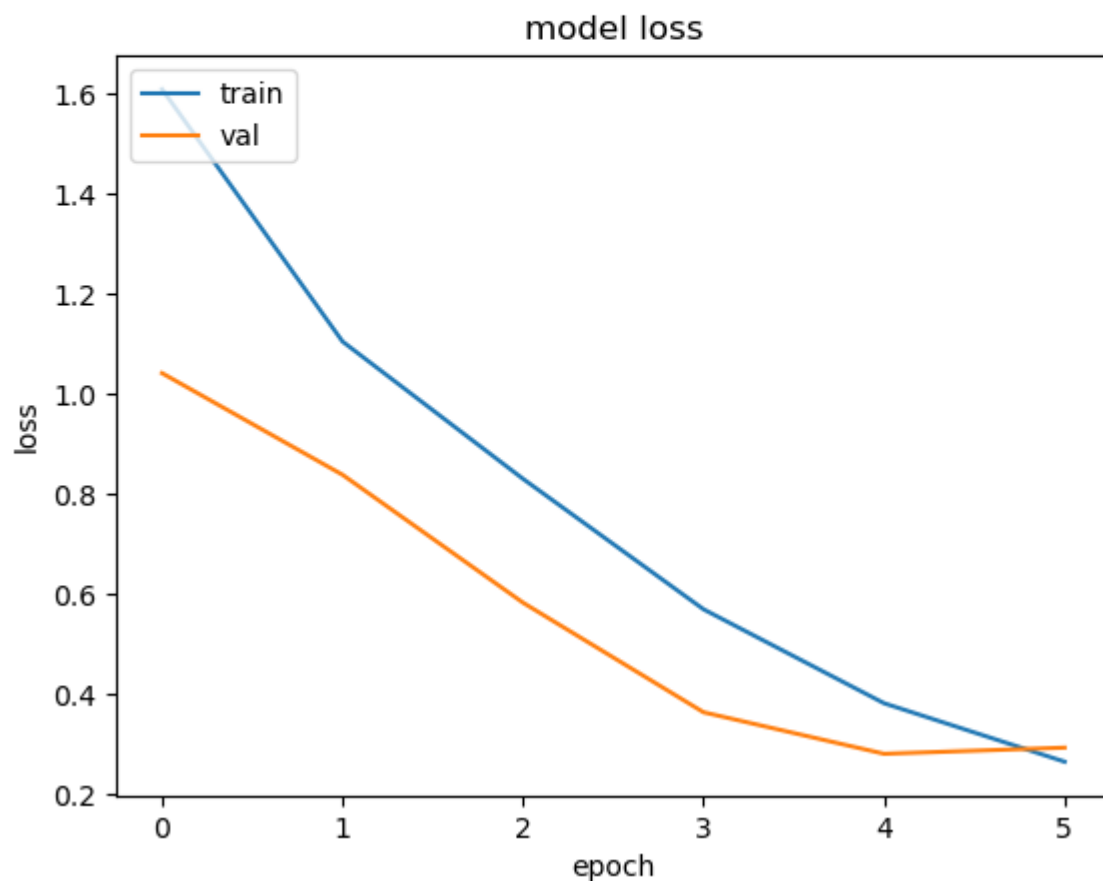
labels = (train.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred2 = [labels[k] for k in pred]
```

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()
```



```

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)

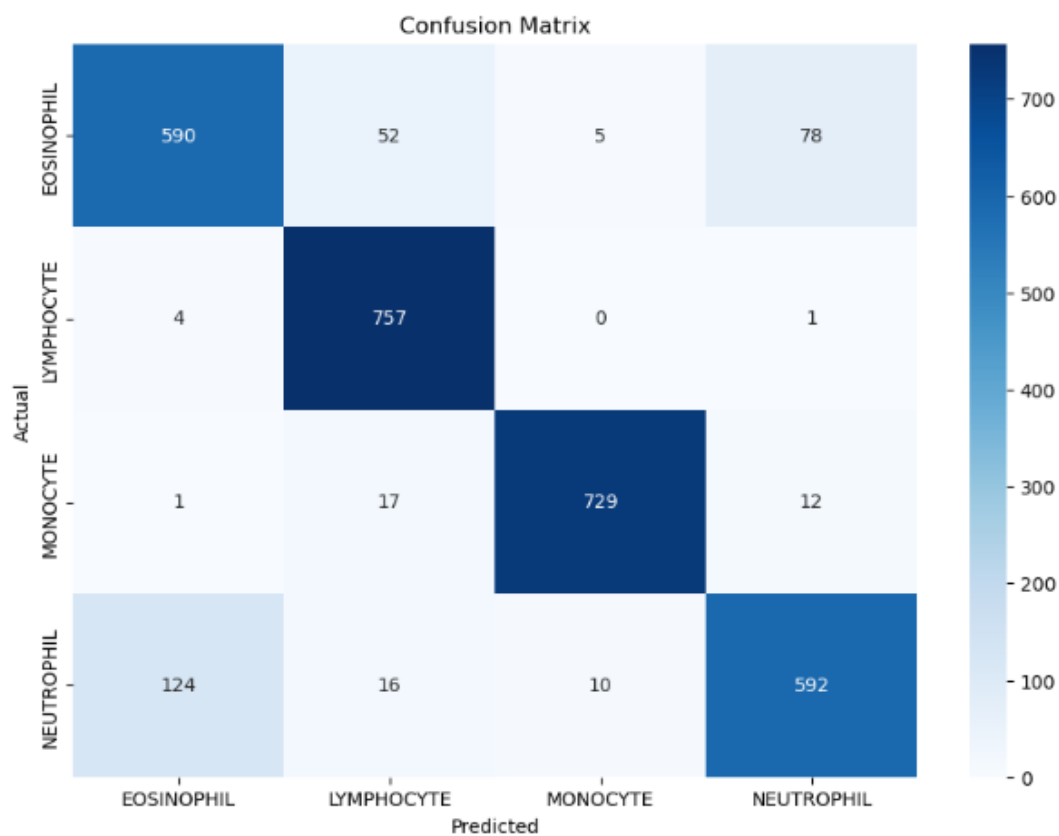
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues')

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()

```



Activity 5: Save Model

Model saved as:

hematovision_model.h5


Milestone 5: Application Building

Activity 1: Build HTML Pages

Create inside templates folder:

- home.html
- upload.html
- Result.html

The screenshot displays the HematoVision web application. The header is a dark blue bar with a red blood drop icon and the text 'HematoVision' and 'Advanced Blood Cell Classification using Transfer Learning'. The main content area is light gray. In the center, a white card titled 'Start Analysis' contains an upload instruction, a file selection button labeled 'Choose File' (which shows 'No file chosen'), and a blue 'Analyze Sample' button. Below this card are three white boxes: 'Objective' (describing WBC identification), 'The Technology' (mentioning MobileNetV2), and 'Performance' (showing 94.8% validation and 85% overall accuracy).



HematoVision

Advanced Blood Cell Classification using Transfer Learning

Start Analysis

Upload a microscopic blood smear image (JPG/PNG)

Choose FileNo file chosen

Analyze Sample

Objective

Automating the identification of White Blood Cells (WBC) to assist hematologists in faster diagnosis of infections and blood disorders.

The Technology

Utilizing **MobileNetV2** architecture with Transfer Learning and Fine-Tuning on over 10,000 images.

Performance

Achieved a **94.8%** validation accuracy and **85%** overall test accuracy across 4 distinct cell classes.



Activity 2: Flask Backend (app.py)

Steps:

- Import Flask
- Load saved model
- Create route for home
- Create route for prediction
- Process uploaded image
- Pass image to model
- Display prediction

```

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']

```

```

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")

```

```

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

```

Run Application:

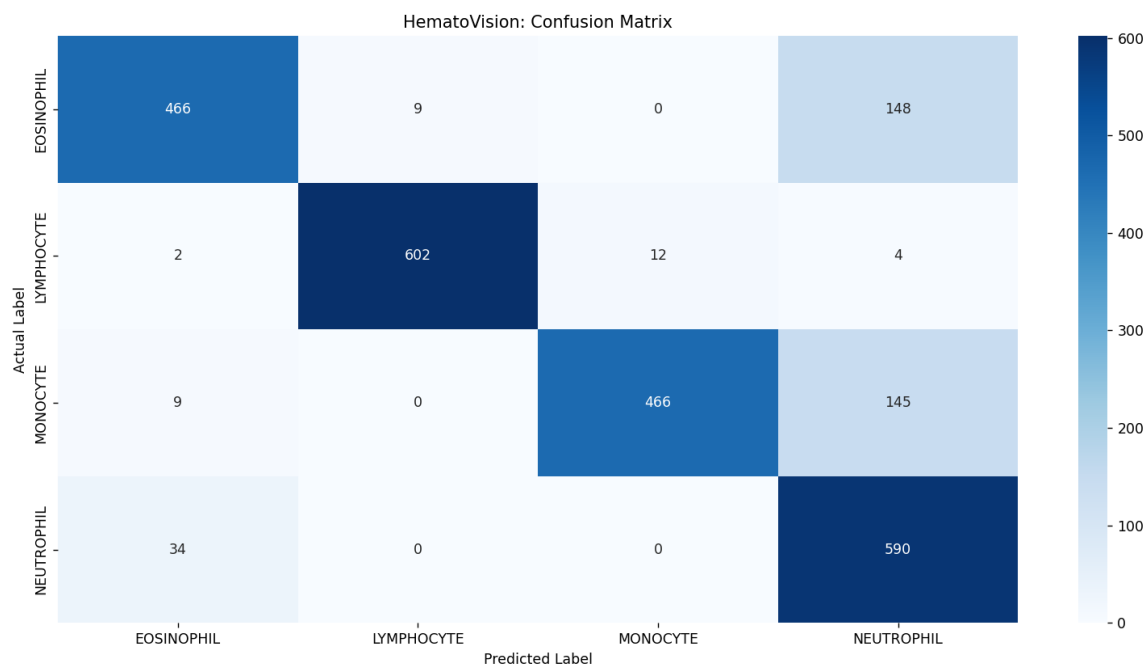
python app.py

```

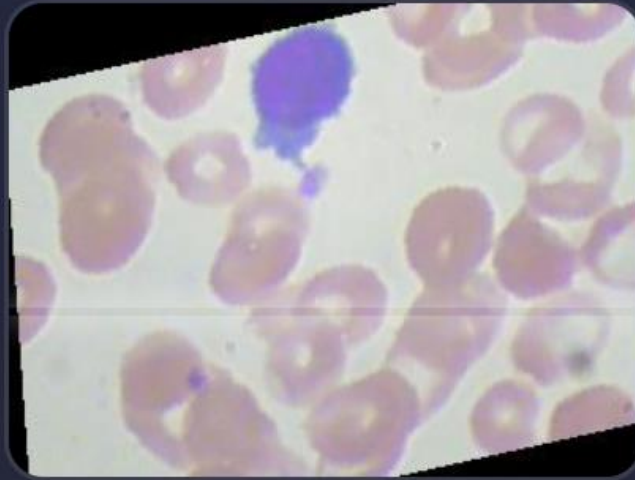
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)

```

<http://127.0.0.1:5000/>



DIAGNOSTIC ANALYSIS



LYMPHOCYTE

PROCESSED: HIGH ACCURACY

100.0% Match

AI INSIGHT: This sample shows morphological characteristics typical of lymphocytes. Analysis performed using MobileNetV2 Deep Learning architecture.

New Analysis