ADVANCED BLOOD CELL CLASSIFICATION USING TRANSFER LEARNING

SRI VASAVI DEGREE COLLEGE

Mentor Name: SRI L LAKSHMI NARAYANA

Team ID: LTVIP2026TMIDS46479

Team Leader: GANTA BHAVYA SRI

MAIL: - gantabhavya26@gmail.com

REG NO: SBAP0040425

TEAM MEMBERS

1. INTI NAGA VARSHINI

Mail: intivarshini0@gmail.com

Reg no: SBAP0040459

2. KONDAPALLI JYOTHI SIVA PRIYA

Mail: sivapriya37227@gmail.com

Reg no: SBAP0040443

3. SINGULURI VIHARIKA

Mail: viharikas123@gmail.com

Reg no: SBAP0040468

Objective:

The primary objective of the HematoVision project is to design and implement an automated blood cell classification system using deep learning techniques. The specific objectives of this project are as follows:

- To study the fundamentals of deep learning and convolutional neural networks
- To understand the concept and advantages of transfer learning
- To develop a CNN-based model using MobileNetV2 for blood cell

classification:

- To preprocess and organize blood cell image data for effective training
- To train and validate the model using labeled image datasets
- To evaluate model performance using accuracy and loss metrics
- To visualize training and validation performance using graphical plots
- To test the trained model on new, unseen blood cell images
- To predict blood cell type along with confidence percentage

Dataset Details:

The dataset used in this project is the BCCD (Blood Cell Count and Detection) Dataset, which is a publicly available dataset widely used for research in blood cell analysis.

Dataset Characteristics:

- Contains microscopic blood cell images
- Images are labeled according to blood cell type
- Dataset includes four classes:
- Eosinophils
- Lymphocytes
- Monocytes
- Neutrophils

The dataset is organized in a hierarchical folder structure, where each folder represents a specific blood cell class. This organization allows easy loading of images using deep learning frameworks such as TensorFlow and Keras.

- **Dataset Source:**

- https://github.com/Shenggan/BCCD_Dataset
- Before training, the images are resized and normalized to ensure uniformity and compatibility with the MobileNetV2 input requirements.

Technique Used:

- Transfer Learning with pre-trained Convolutional Neural Networks (CNNs)
- Models such as VGG, ResNet, Inception, or MobileNet
- Pre-trained models leverage learned visual features from large datasets
- Fine-tuning applied to adapt the model to blood cell morphology

Core Benefits:

- High accuracy in blood cell type classification
- Faster model development compared to training from scratch
- Efficient handling of complex cell shapes and textures
- Reduces human error in manual microscopic analysis
- Supports scalable deployment in medical environments

Applications:

The HematoVision project has a wide range of applications in the healthcare and research domains:

Medical Diagnostics

- Assists doctors and pathologists in identifying abnormal blood cell distributions.

Pathology Laboratories

- Automates routine blood smear analysis and reduces workload.

Healthcare Automation

- Integrates with laboratory information systems for faster reporting.

- Research and Education
  - Used as a learning tool for medical students and researchers.

- Clinical Decision Support
  - Supports medical professionals by providing AI-based second opinions.

Impact:

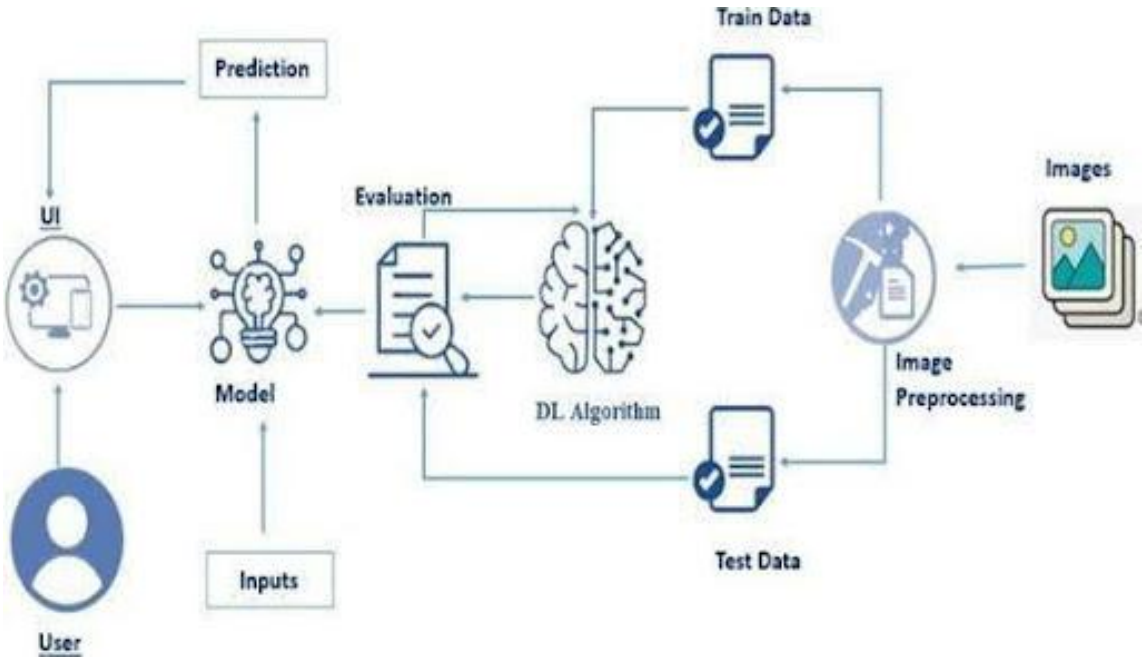
Promotes advancements in **medical diagnostics and healthcare automation** by enabling accurate and efficient blood cell identification through deep learning. The system supports early disease detection, reduces manual workload for pathologists, and enhances clinical decision-making using intelligent image analysis technology.

Architecture:

The architecture diagram illustrates the complete workflow of the **blood cell classification system**. It begins with a user or laboratory technician uploading a microscopic blood cell image through the user interface. The uploaded image undergoes **preprocessing**, including resizing, normalization, and noise reduction to improve image quality.

The processed data is then divided into **training, validation, and testing datasets**. A **transfer learning–based deep learning model** using a pre-trained Convolutional Neural Network (CNN) is employed to train the system on blood cell features. During training, the model learns distinctive patterns such as cell shape, size, texture, and nucleus structure.

After training, the model is **evaluated** using test data to measure performance metrics such as accuracy, precision, recall, and loss. Once validated, the trained model is used to **predict blood cell types** from new input images, providing fast and reliable classification results suitable for real-world medical applications.



- **Prerequisites:**

To complete this project, you must require the following software, concepts, and packages

- **Anaconda Navigator:**
  - Refer to the link below to downloadAnaconda 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.

- **Software Requirements:**
  - Python Programming Language
  - Google Colab (cloud-based development environment)
  - TensorFlow and Keras (deep learning framework)
  - NumPy (numerical computations)
  - Matplotlib (data visualization)
  - GitHub (version control and project hosting)

- **Prior Knowledge:**

  To successfully understand, implement, and deploy the **Advanced Blood Cell Classification Using Transfer Learning** project, the following prior knowledge is essential:

➢ **Deep Learning (DL) Concepts**

- Basics of Machine Learning vs Deep Learning
- Supervised learning
- Training, validation, and testing datasets
- Loss functions and optimization

➢ **Neural Networks**

- Artificial Neural Networks (ANN) fundamentals
- Structure of neurons (input layer, hidden layers, output layer)
- Activation functions (ReLU, Softmax, Sigmoid)
- Backpropagation and gradient descent

➢ **Deep Learning Frameworks**

- Experience with frameworks such as:
- TensorFlow
- Keras
- PyTorch
- Model building, compiling, and training
- Saving and loading models

- **Transfer Learning**

  - Concept of pre-trained models
  - Feature extraction
  - Fine-tuning techniques
  - Popular pre-trained models like:
  - VGG16
  - ResNet
  - MobileNet

- **Convolutional Neural Networks (CNN)**

  - Convolution layers
  - Pooling layers
  - Flattening
  - Fully connected layers
  - Image feature extraction

- **Overfitting and Regularization**

  - What is overfitting?
  - Techniques to prevent overfitting:
  - Dropout
  - Data augmentation
  - Early stopping
  - L2 regularization

- **Optimizers**

  - Gradient Descent
  - Adam Optimizer
  - RMSprop
  - Learning rate concept

Project Objectives:

To design and develop an accurate and efficient deep learning model for automatic classification of blood cell images using transfer learning techniques**.**

Specific Objectives:

- **Develop an Automated Classification System**
  - Build a model that can automatically classify blood cells into:

- Eosinophil
- Lymphocyte
- Monocyte
- Neutrophil

- **Apply Transfer Learning**
  - Use pre-trained CNN models such as:
  - ResNet50
  - VGG16
  - MobileNetV2
  - Reduce training time and improve accuracy.

- **Improve Classification Accuracy**
  - Implement
  - Data augmentation
  - Regularization techniques (Dropout)
  - Hyperparameter tuning

- **Feature Extraction**
  - Extract important microscopic features such as:
  - Cell shape
  - Nucleus structure
  - Texture patterns

- **Model Evaluation**
  - Evaluate model performance using:
  - Accuracy
  - Confusion Matrix
  - Precision
  - Recall
  - F1-Score

- **Reduce Manual Effort in Diagnosis**
  - Assist medical professionals by:
  - Providing faster predictions
  - Reducing human error
  - Supporting early disease detection

- **Develop a Reusable and Scalable System**
  - Save trained model (.h5 file)
  - Make the model deployable for:
  - Web applications
  - Clinical support systems

- **Project Flow:**

- **Problem Definition**
  - Identify the need for automatic blood cell classification.
  - Define the classes:
  - Eosinophil
  - Lymphocyte
  - Monocyte
  - Neutrophil

- **Dataset Collection:**
  - Download BCCD or similar dataset.
  - Organize images into class-wise folders.
  - dataset/
  - train/
  - validation/
  - test/

- **Data Preprocessing:**
  - Resize images (224 × 224).
  - Normalize pixel values (0–1).
  - Remove corrupted images if any.

- **Data Augmentation:**
  - Apply transformations:
  - Rotation
  - Horizontal Flip
  - Zoom
  - Shear

- **Model Selection (Transfer Learning):**
  - Choose pre-trained model:
  - ResNet50
  - VGG16
  - MobileNetV2
  - Remove top layer (include_top=False)
  - Freeze base layers initially.

- **Model Building:**
  - Add custom layers:
  - Global Average Pooling
  - Dense Layer (ReLU)
  - Dropout Layer
  - Output Layer (Softmax – 4 classes)

- **Model Compilation:**
  - Loss Function: Categorical Crossentropy
  - Optimizer: Adam
  - Metrics: Accuracy

- **Model Training:**
  - Train using training dataset.
  - Validate using validation dataset.
  - Monitor:
  - Training Accuracy
  - Validation Accuracy
  - Loss

- **Model Evaluation:**
  - Test model on unseen test data.
  - Generate:
  - Confusion Matrix
  - Classification Report
  - Accuracy Score

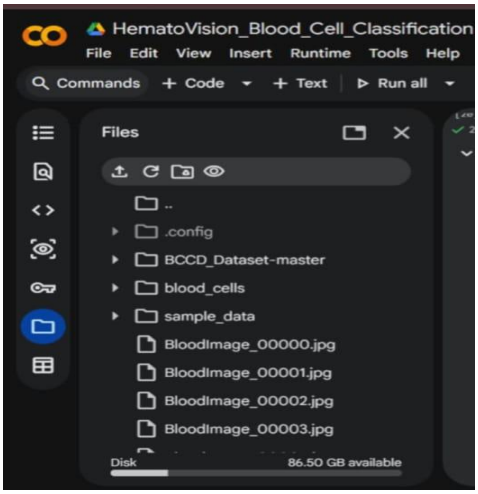
- **Model Prediction:**
  - Load trained model (.h5 file)
  - Predict new blood cell image
  - Display predicted class

- **Save & Deploy**
  - Save model file
  - Upload project to GitHub
  - Share results (PDF / Report / Screenshots)

- **Project Structure:**

Create the Project folder which contains files as shown below:

The project is implemented as a web-based deep learning application using **Python, TensorFlow/Keras, and Flask**.



The project directory contains the following main components:

- **app.py** – Flask backend application
- **templates/** – HTML pages for user interface
- **static/** – CSS, uploaded images, and assets
- **model.h5** – Trained deep learning model
- **dataset/** – Blood cell image dataset

The HTML files provide the user interface where the user uploads a microscopic blood cell image.
The Python Flask script loads the trained model and predicts the blood cell type.

- **Data Collection and Preparation:**

Machine Learning and Deep Learning models depend heavily on the quality and quantity of data. Data is the most crucial component that enables the model to learn meaningful patterns. In this project, blood cell image data is used to train and evaluate the classification model.

- **Collect 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.

```
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
```

- **Read the Dataset:**

➢ The images are loaded using Python libraries such as:

- TensorFlow/Keras
- NumPy
- OpenCV
- Matplotlib

The dataset is unzipped and image paths are read into the program. Each image is resized and normalized before feeding it into the neural network

```
# 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)
```

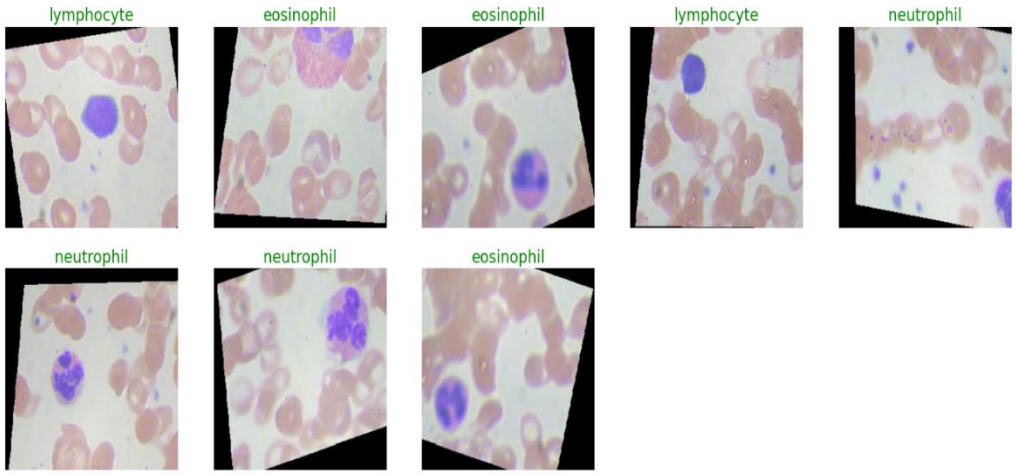
```
bloodCell_df.head()
```

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

- **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 diamo

- **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.
```

- **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.

```
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()
```

```
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)
```

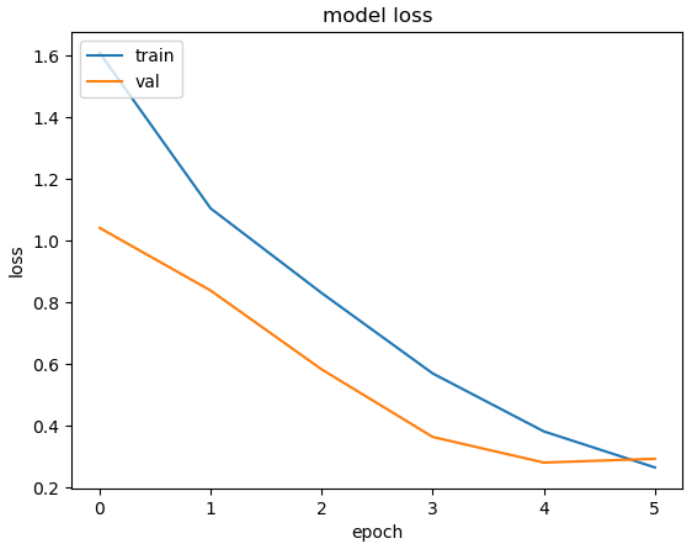
```
996/996 [==============================] - 3392s 3s/step - loss: 0.2661 - accuracy: 0.8925 - val_loss: 0.2942 - val_accurac
y: 0.8800
```

- **Testing Model & Data Prediction:**

Evaluating the model

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

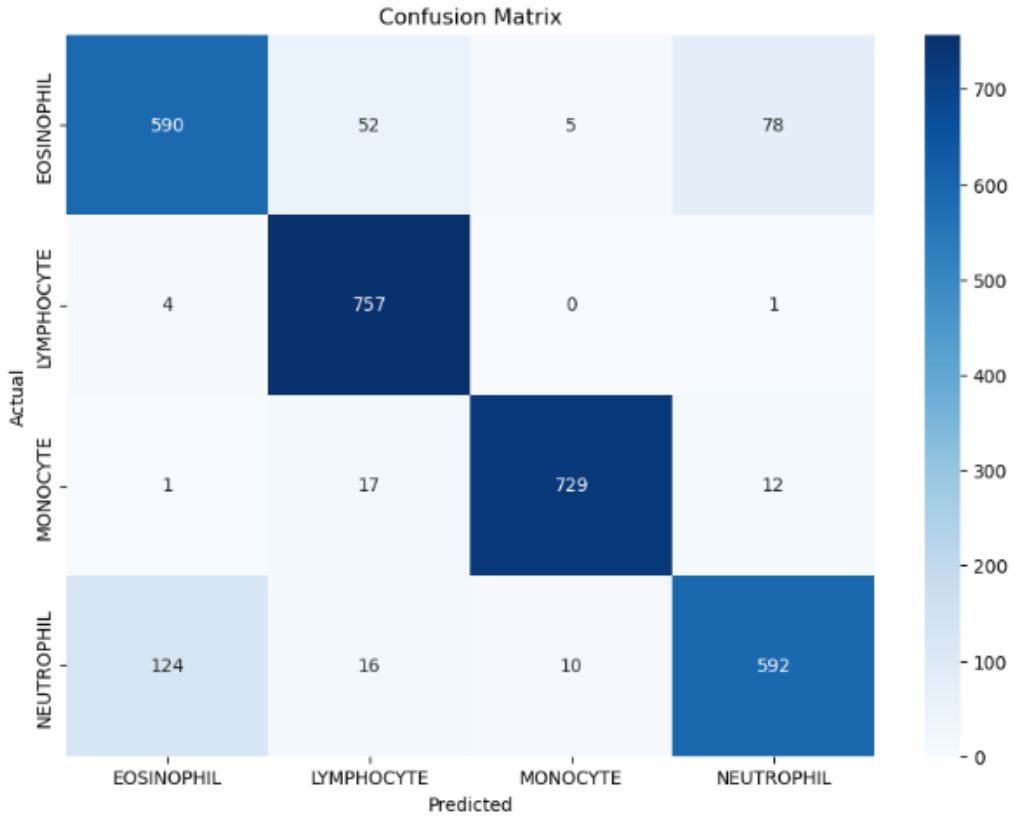
```
plt.plot(history.history['loss'] + history1.history['loss'])
plt.plot(history.history['val_loss'] + history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
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%
```

```
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()
```



- **Saving the model:**

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

```
model.save("Blood Cell.h5")
```

- **Application Building:**

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script

Building HTML Pages:

- For this project create three 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

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")
```

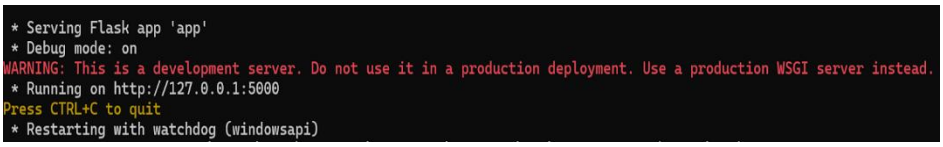
Here we are routing our app to the output() function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. Predict () function. This function returns the prediction. This prediction value will rendered to the text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

Main Function:

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

- **Run the web application:**

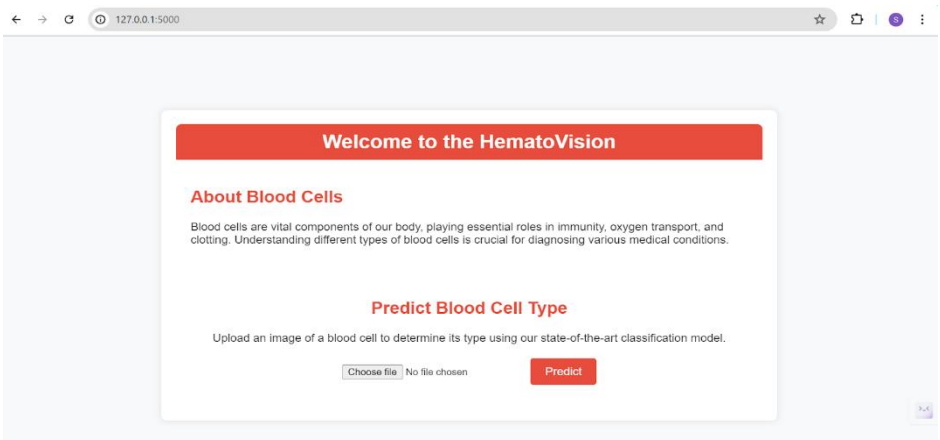
- Open Anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “app.py” command
- Navigate to the local host where you can view your web page.
- Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see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



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

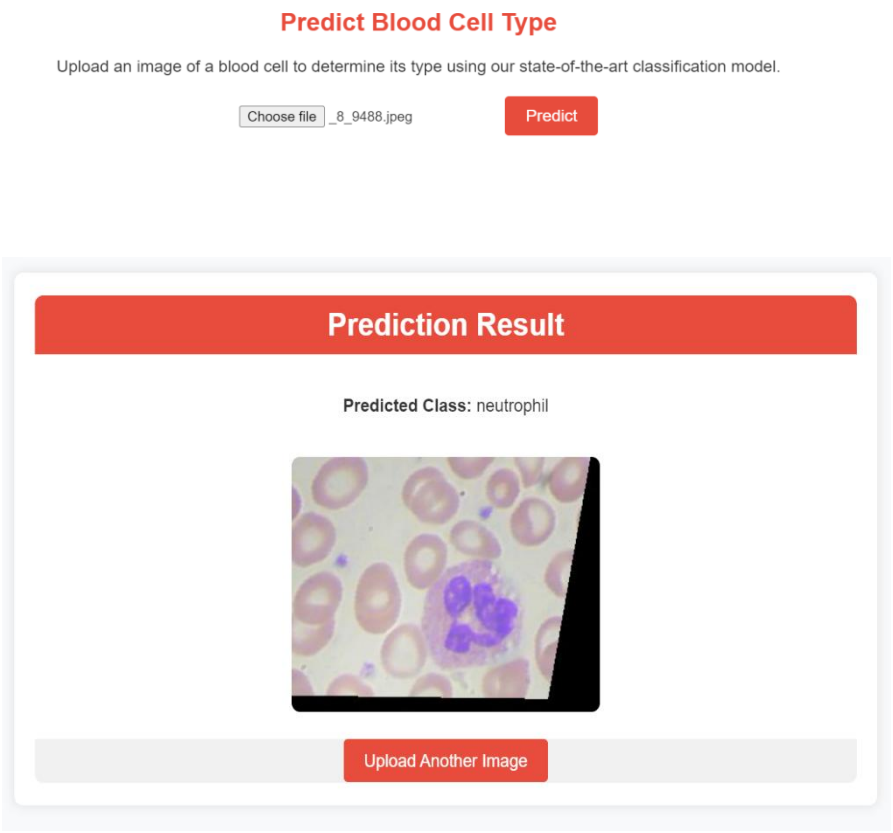
- UI Image preview:

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

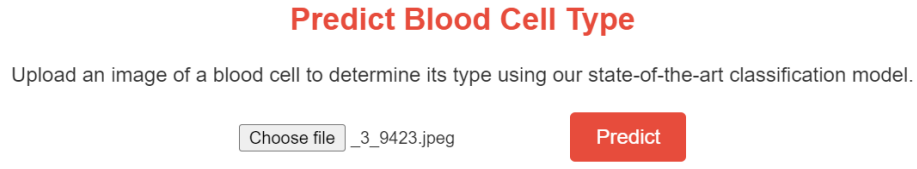


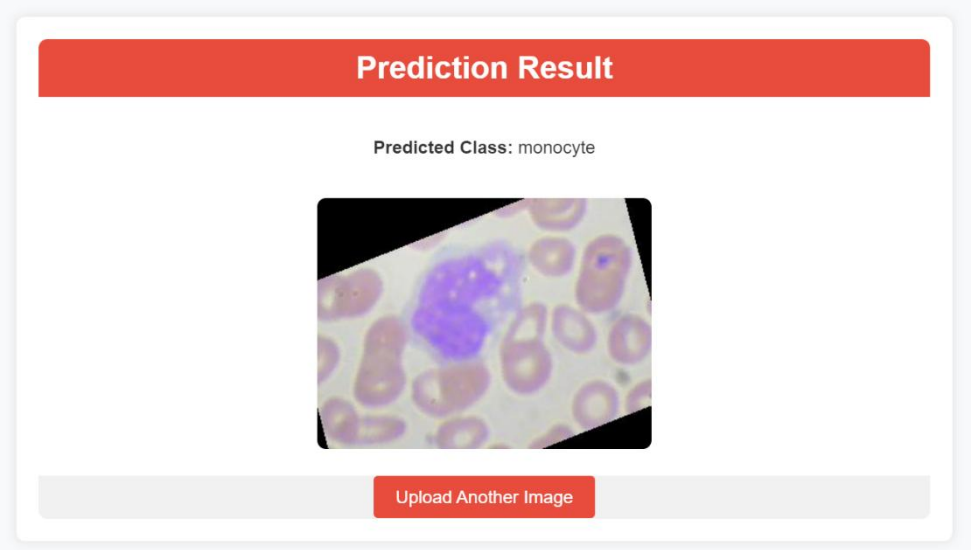
By clicking on choose file it will ask us to upload the image , then by clicking on the predict button , it will take us to the result.html

Test For Class-1 : Neutrophil

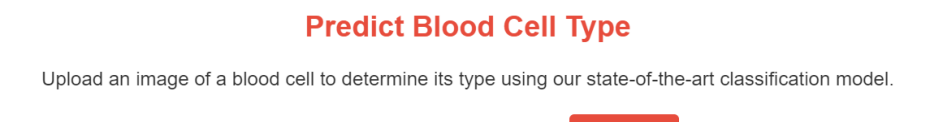


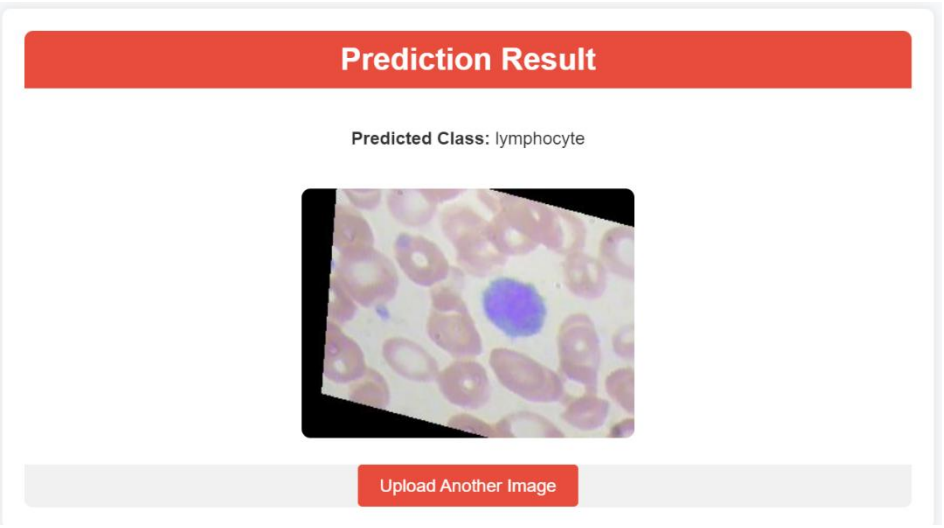
Test For Class-2 : Monocyte



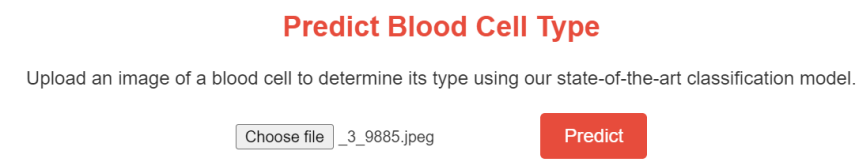


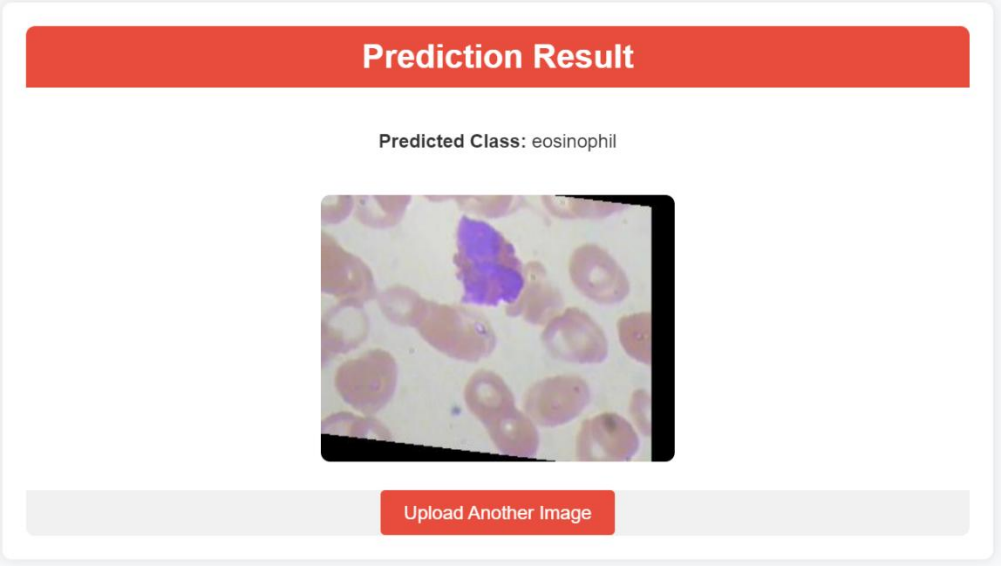
Test For Class-3 : Lymphocyte





Test For Class-4 : Eosinophil





- **Conclusion:**

The project “Advanced Blood Cell Classification Using Transfer Learning” successfully demonstrates the application of deep learning techniques in the medical field for automated image classification. The main objective of this project was to develop an efficient model capable of accurately classifying different types of blood cells using convolutional neural networks and transfer learning.

By utilizing pre-trained models, the system was able to achieve high accuracy while reducing training time and computational complexity. Transfer learning played a crucial role in improving performance, especially when working with a limited dataset. Data preprocessing, augmentation, model training, and performance evaluation were carefully implemented to ensure reliable and consistent results.

The developed model can assist medical professionals by automating the blood cell identification process, reducing manual effort, and minimizing human error. This project highlights the importance of artificial intelligence in healthcare and shows how deep learning can be used to solve real-world medical problems effectively.

In conclusion, this project not only achieved its technical objectives but also provides a strong foundation for future improvements, such as deploying the model as a web or mobile application for practical use in laboratories and hospitals.

Acknowledgement:

I would like to express my sincere gratitude to my project guide and faculty members for their continuous support, valuable guidance, and encouragement throughout the completion of this project, “Advanced Blood Cell Classification Using Transfer Learning.” Their suggestions and technical insights greatly helped in improving the quality of this work.

I also thank my institution for providing the necessary resources and learning environment to successfully carry out this project. The knowledge and skills gained during the course played an important role in understanding and implementing deep learning concepts effectively.

I am grateful to my friends and classmates for their support, discussions, and motivation during the development of this project. Finally, I would like to thank my family for their constant encouragement and support, which helped me complete this work successfully.

THANK YOU,

SMART BRIDGE

GANTA BHAVYA SRI

TEAM LEADER

SIGNATURE OF THE HOD **SIGNATURE OF THE PRINCIPAL**