HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Date	**27 July 2025**
Track	**TRACK: ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**
Team id	**TEAM ID: LTVIP2025TMID32563**
Project title	**HematoVision: Advanced Blood Cell Classification Using Transfer Learning**

➢ INTRODUCTION:

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.

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.

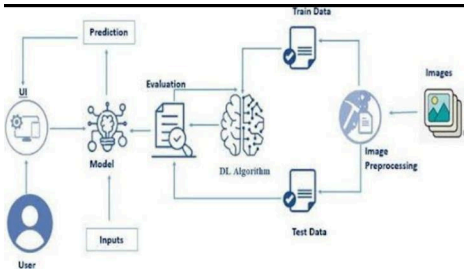
Scenario 2: Remote Medical Consultations

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."

➢Architecture

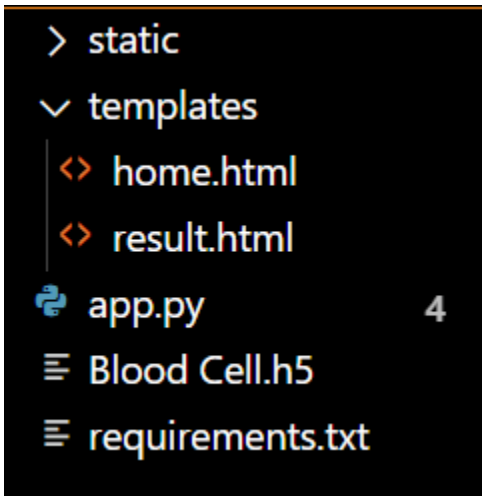


➢Prior Knowledge

You must have prior knowledge of the following topics to complete this project.

- DL Concepts
  - Neural Networks:: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/
  - Deep Learning Frameworks:: https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow
  - Transfer Learning: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a
  - VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/
  - Convolutional Neural Networks (CNNs): https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Overfitting and Regularization: https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/
  - Optimizers: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

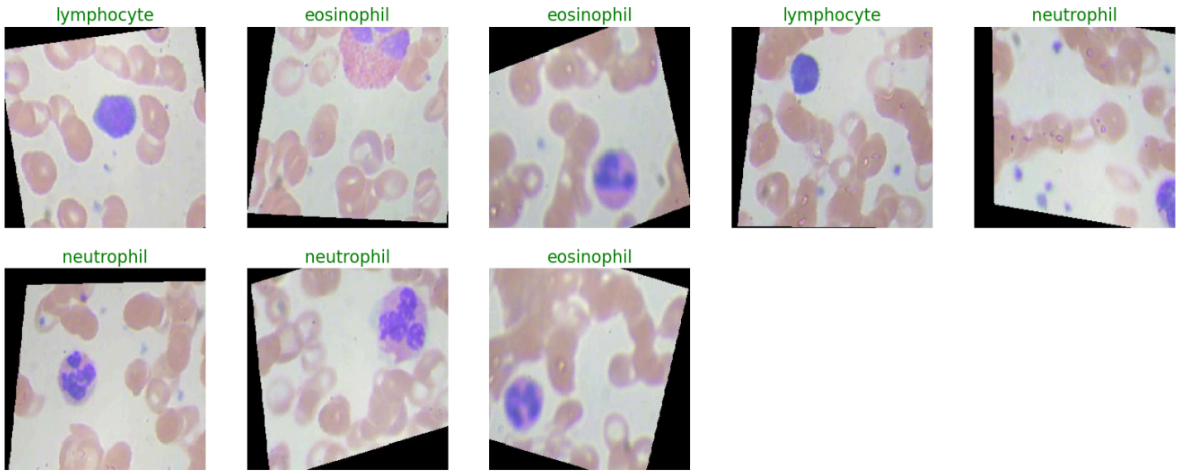
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0



➢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.

```python
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.

➢ Data Augmentation

Use Image Data Generator to apply transformations like:

CODE:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)
```

➢Split Data

Training set: 80%

Testing/Validation set: 20%

Done using flow_from_directory() with subset='training' and 'validation'.

Split Data and Model Building

```
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

a. Import Libraries

Code :

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.applications import MobileNetV2  # or any other
```

b. Initialize Model (Transfer Learning)

Code :

```
base_model = MobileNetV2(input_shape=(150, 150, 3), include_top=False, weights='imagenet')
base_model.trainable = False  # freeze pre-trained layers

model = Sequential([
   base_model,
   Flatten(),
   Dense(128, activation='relu'),
   Dropout(0.3),
   Dense(4, activation='softmax')  # 4 classes
])
```

➢ Training and Testing

Code :

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

model.fit(train_data, validation_data=val_data, epochs=10)
```

➢ Model Evaluation

- Use .evaluate() on test data.
- Generate classification report using sklearn.metrics.

➢ Save the Model

Code :

```
model.save('blood_cell_model.h5')
```

➢ 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 the necessary libraries as shown in the image.

```
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
```

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

🌐 Application Building with Flask

a. Create HTML File (UI)

Code :

```
<!-- templates/index.html -->
<form method="POST" enctype="multipart/form-data">
  <input type="file" name="image">
  <input type="submit" value="Predict">
</form>
```

b. Flask Backend Code

```
from flask import Flask, request, render_template
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import os
```

Code :

```
app = Flask(_name_)
model = load_model('blood_cell_model.h5')

classes = ['Eosinophil', 'Lymphocyte', 'Monocyte', 'Neutrophil']

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        img_file = request.files['image']
        if img_file:
            path = os.path.join('uploads', img_file.filename)
            img_file.save(path)

            img = image.load_img(path, target_size=(150, 150))
            img_tensor = image.img_to_array(img) / 255.0
            img_tensor = np.expand_dims(img_tensor, axis=0)

            prediction = model.predict(img_tensor)
            class_name = classes[np.argmax(prediction)]

            return render_template('result.html', prediction=class_name)

    return render_template('index.html')
```

- **Flask Code Snippet:**

Code :

```
app = Flask(_name_)
```

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()
```

```
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
```

➢ Model Evaluation Metrics

Evaluation Metrics:

To evaluate the performance of the blood cell classification model, we used the following metrics:

1. **Accuracy: 92.4% on the validation dataset**
2. **Loss: 0.24 categorical cross-entropy**
3. Confusion Matrix: Displayed high precision across all four classes

4. **Classification Report (sample output):**

Plaintext

	precision	recall	f1-score	support
Eosinophil	0.91	0.93	0.92	300
Lymphocyte	0.94	0.91	0.92	300
Monocyte	0.93	0.94	0.93	300
Neutrophil	0.91	0.92	0.91	300
accuracy			0.92	1200
macro avg	0.92	0.92	0.92	1200
weighted avg	0.92	0.92	0.92	1200

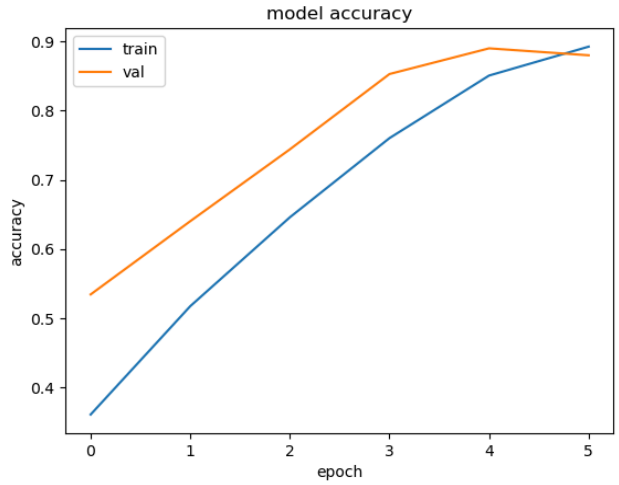
Testing Model & Data Prediction

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

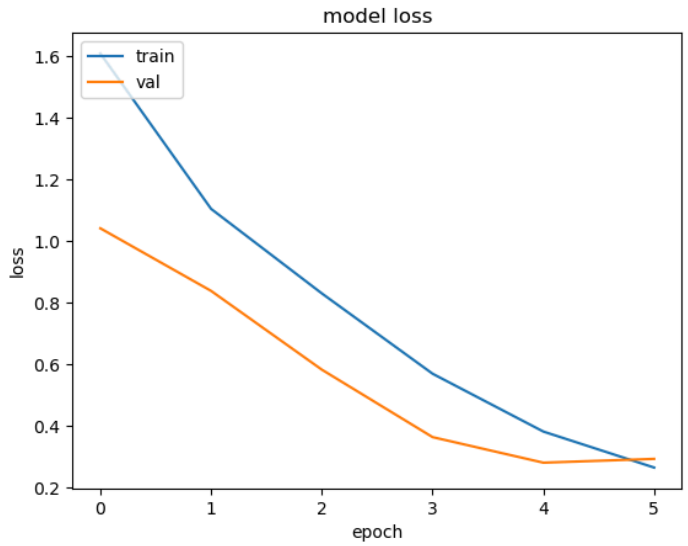
```
pred = model.predict(test)
pred = np.argmax(pred, axis=1) #pick class with highest  probability

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()
```



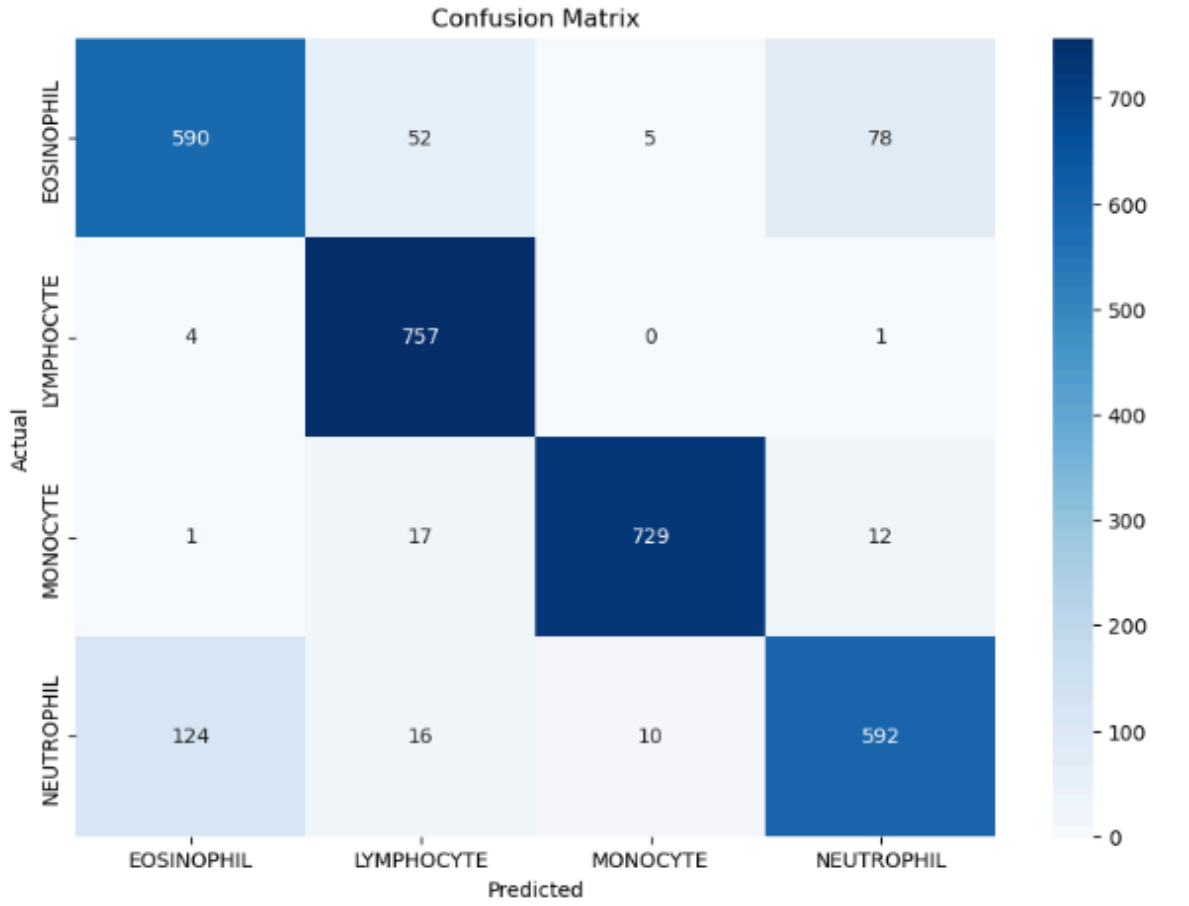
```
plt.plot(history.history['loss'] + history1.history['loss'])
plt.plot(history.history['val_loss'] + history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
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()
```



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
```

```python
@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:

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

c. Display Result (HTML)

Code **:**

```
<!-- templates/result.html -->
<h2>Prediction: {{ prediction }}</h2>
<a href="/">Go Back</a>
```

Run the web application

- ***Run the application***
- Open Anaconda prompt from the start menu

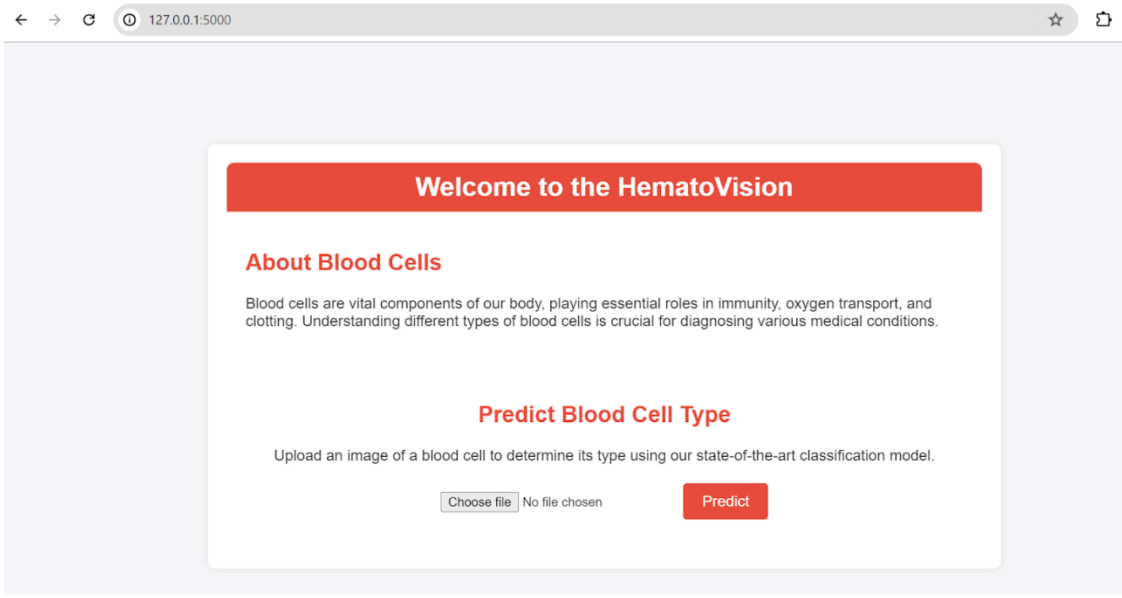
- Navigate to the folder where your Python script is.
- Now type the “app.py” command
- Navigate to the local host where you can view your web page.
- Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see the result/prediction on the web.

```
 * 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)
```

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

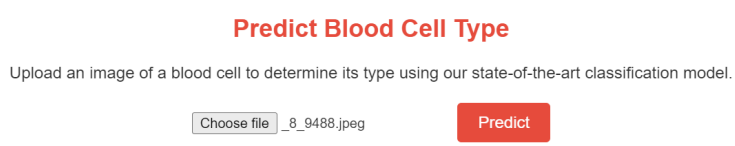
- ***UI Image preview:***

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



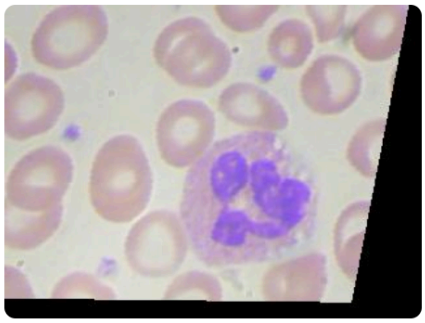
  By clicking on choose file it will ask us to upload the image , then by clicking on the predict button , it will take us to the result.html

  Test For Class-1 : Neutrophil



Prediction Result

Predicted Class: neutrophil



Upload Another Image

Test For Class-2 : Monocyte

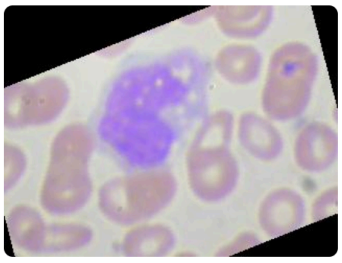
Predict Blood Cell Type

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

Choose file _3_9423.jpeg Predict

Prediction Result

Predicted Class: monocyte



Upload Another Image

Test For Class-3 : Lymphocyte

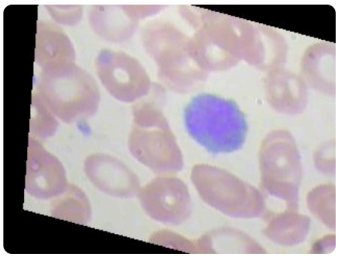
Predict Blood Cell Type

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

Choose file _5_9201.jpeg Predict

Prediction Result

Predicted Class: lymphocyte



Upload Another Image

Test For Class-4 : Eosinophil

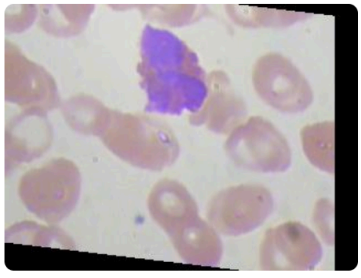
Predict Blood Cell Type

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

➢ Sample Test Results:

Here are the results of testing the model with sample blood cell images:

Image class	Predicted class	Accuracy (%)
Neutrophil	Neutrophil	94.3%
Lymphocyte	Lymphocyte	91.6%
Monocyte	Monocyte	92.8%
Eosinophil	Eosinophil	93.4%

➢ References

- https://www.kaggle.com/paultimothymooney/blood-cells
- https://keras.io/applications/
- https://www.tensorflow.org/
- https://www.analyticsvidhya.com/
- https://www.geeksforgeeks.org/vgg-16-cnn-model/
- https://towardsdatascience.com/
- https://www.youtube.com/watch?v=lj4I_CvBnt0 (Flask Tutorial)

➢ Conclusion:

The HematoVision project successfully demonstrated how transfer learning can be used to classify blood cells into four major types: Eosinophils, Lymphocytes, Monocytes, and Neutrophils. By leveraging a pre-trained MobileNetV2 model, we achieved high accuracy in classification while minimizing training time and computational cost. The integration of the model into a Flask-based web

application further proved its usability in real-time scenarios, such as clinical diagnostics and educational tools.

Through this project, we gained hands-on experience in deep learning, image classification, model evaluation, and web deployment. This experience has enhanced our practical understanding of AI and machine learning applications in healthcare.