HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Introduction

Team ID : **LTVIP2026TMIDS62830**

Team Members:

Team Leader : **Chintalacheruvu L V L Santosh**

Team member : **Kallepalli R P S V Varma**

Team member : **Korrapati Thirumala Narasimha Naidu**

Team member : **Chukka Abhishek Mahin Prabhas**

OBJECTIVE:

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

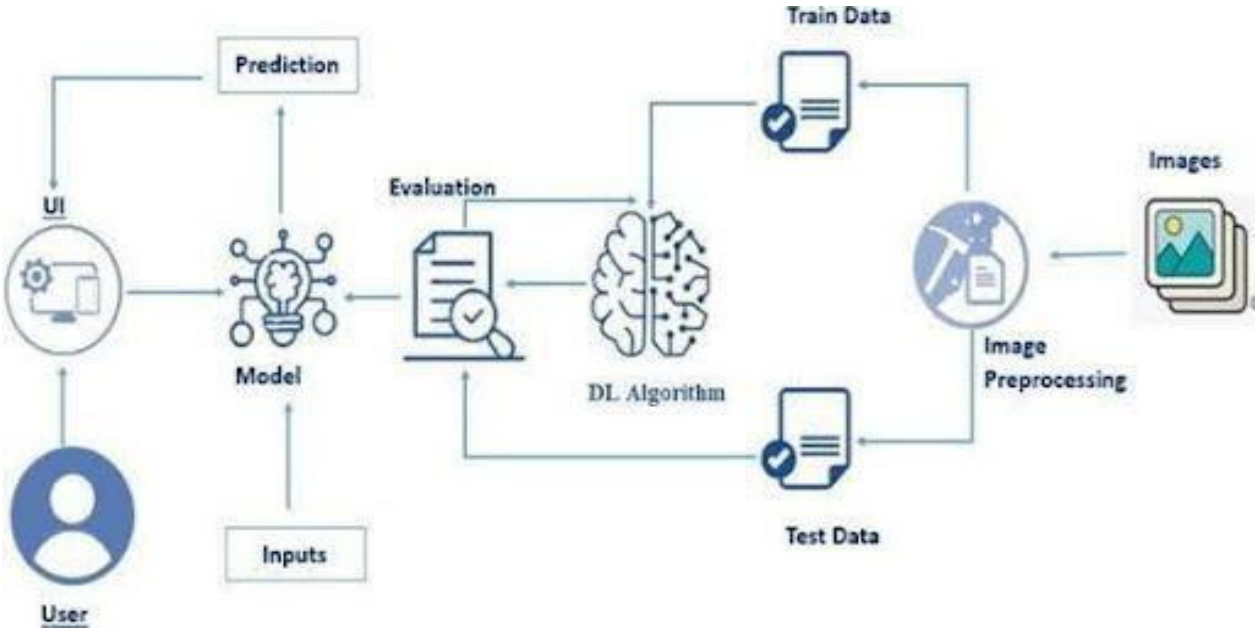
HematoVision can be employed in telemedicine platforms to enhance remote consultations and diagnostics. With transfer learning, the model's ability to accurately classify blood cells from diverse sources is improved, allowing healthcare providers to upload blood cell images for automated analysis. This 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:

Deep Learning Frameworks:

- 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

Project Flow

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

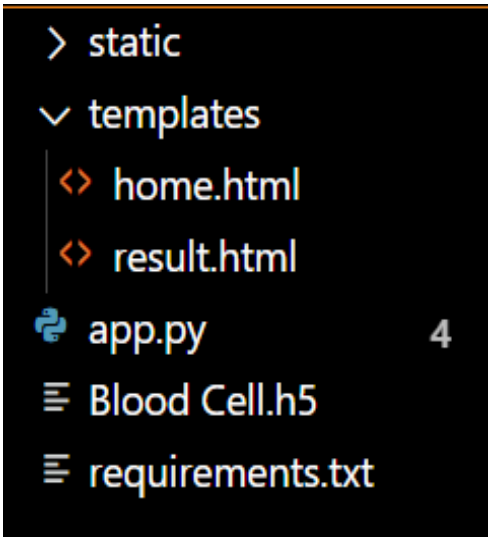
To accomplish this, we have to complete all the activities listed below,

- Data Collection: Collect or download the dataset that you want to train.
- Data pre-processing
    - Data Augmentation
    - Splitting data into train and test
- Model building
    - Import the model-building libraries
    - Initializing the model
    - Training and testing the model
    - Evaluating the performance of the model
    - Save the model
- Application Building
    - Create an HTML file

- Build python code

Project Structure

Create the Project folder which contains files as shown below



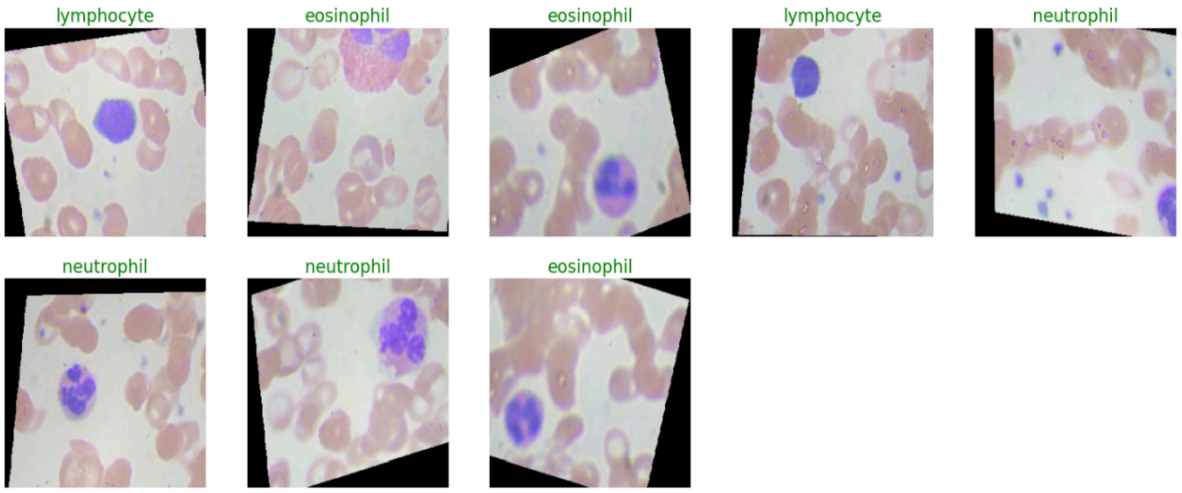
Activity 1.1: Importing the libraries:

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
```

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.



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.

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)
```

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: "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
```

Testing Model & Data Prediction

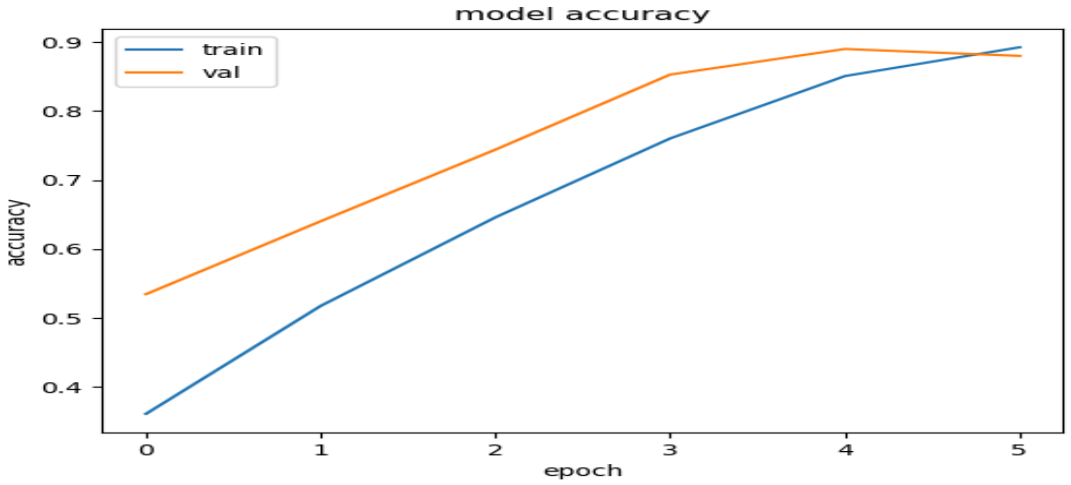
Evaluating the model

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

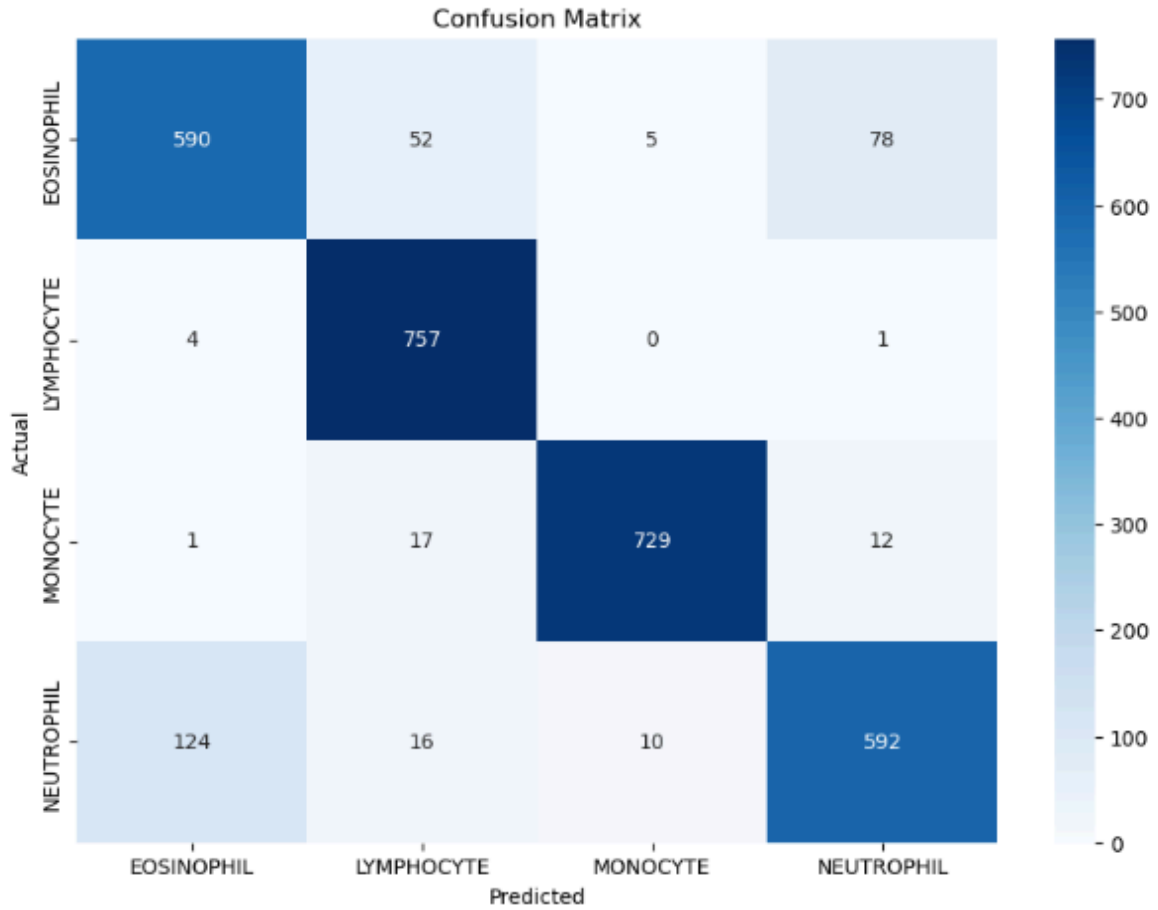
```
pred = model.predict(test)
pred = np.argmax(pred, axis=1) #pick class with highest  probability

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()
```



```
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

Saving the model

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

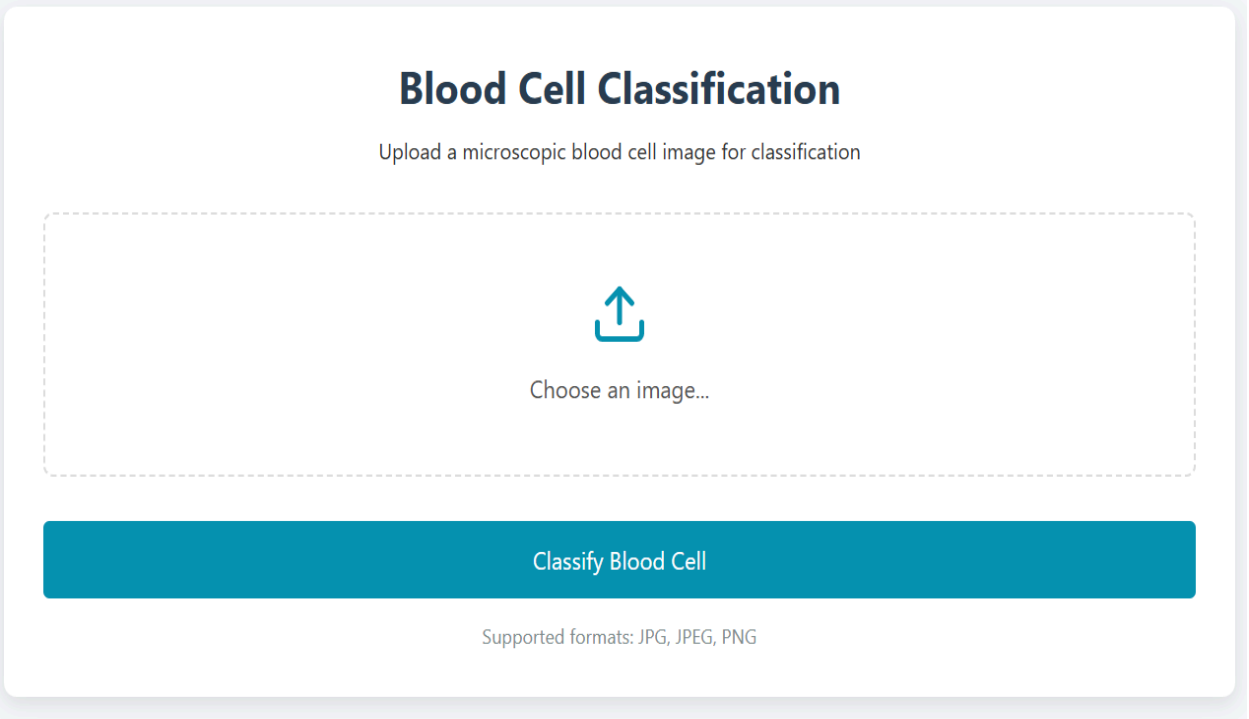
Building HTML Pages:

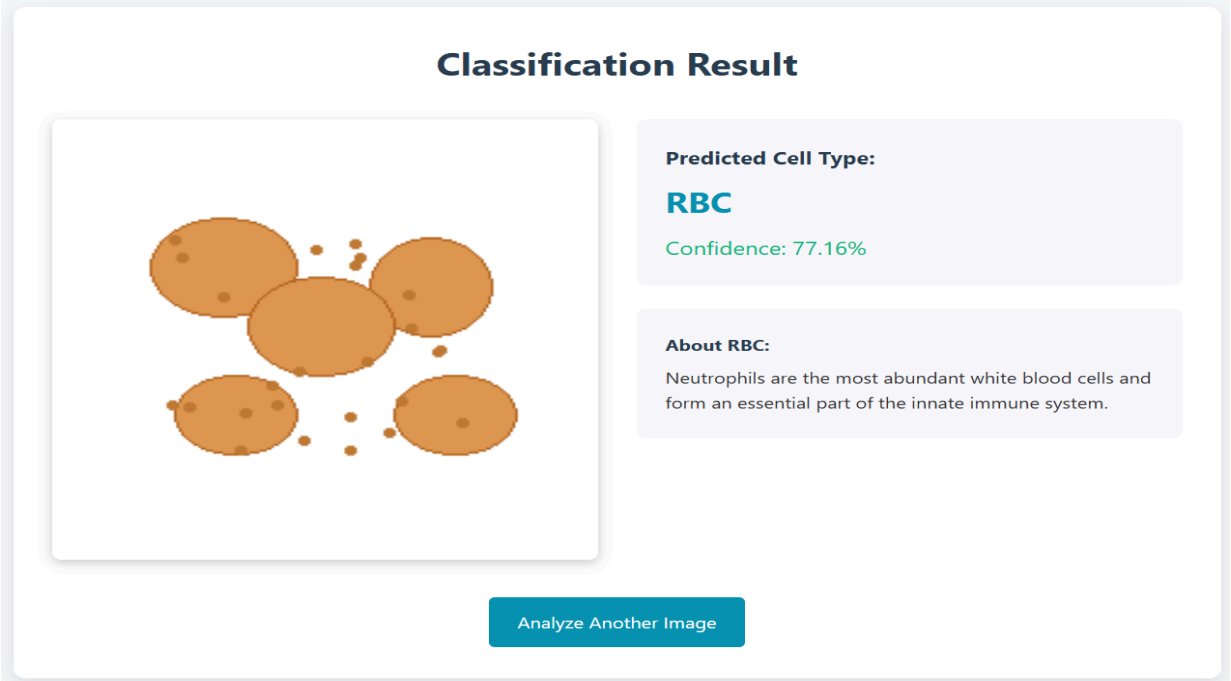
For this project create three HTML files namely

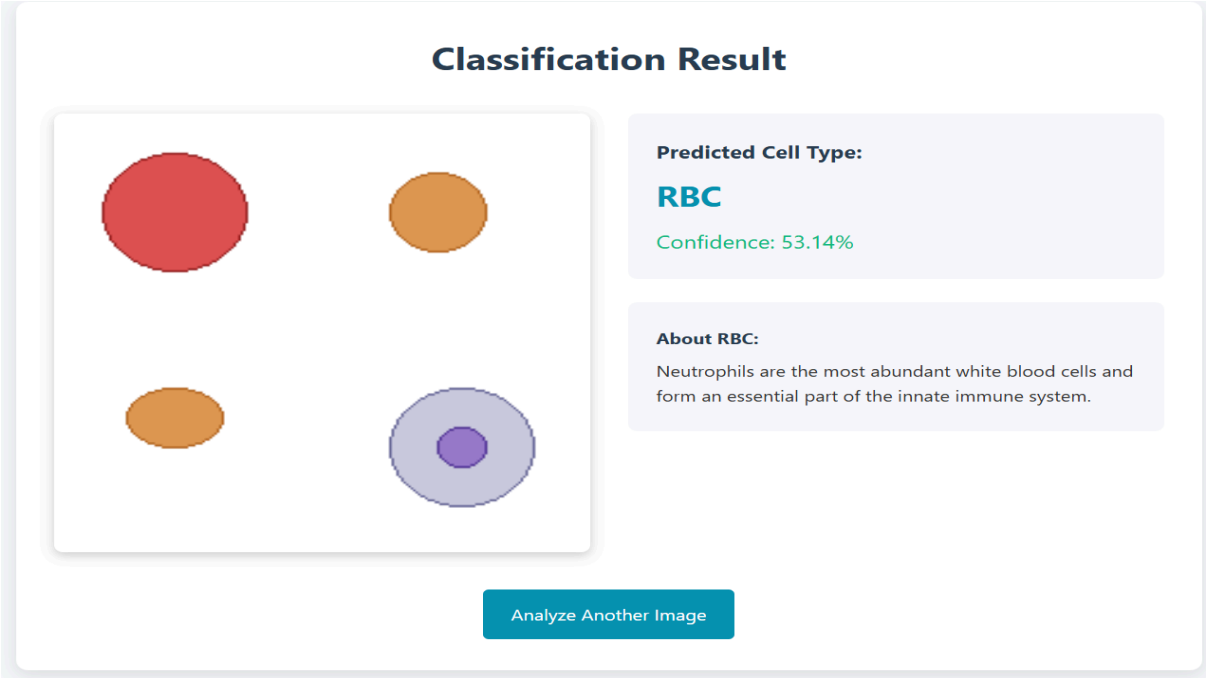
- home.html
- result.html

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.







Project Objectives

By the end of this project, you will:

- Know fundamental concepts and techniques used for Deep Learning.
- Gain a broad understanding of data.

- Have knowledge of pre-processing the data/transformation techniques on outliers and some visualization concepts.