ProjectTitle:HematoVision: Advanced Blood Cell Classification Using Transfer Learning
BranchName: ComputerscienceandEngineering

Track: Artificial Intelligence and Machine learning

Team member:Jalla Mounika

MailId:jallamounika01@gmail.com

SubmittedBy:

- Jalla Mounika
- Swarnandhracollegeofengineeringandtechnology
- Artificialintelligenceandmechinelearning
- RollNo:22A21A6132

SubmittedTo:

SmartBridge

Abstract:

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.

Introduction:

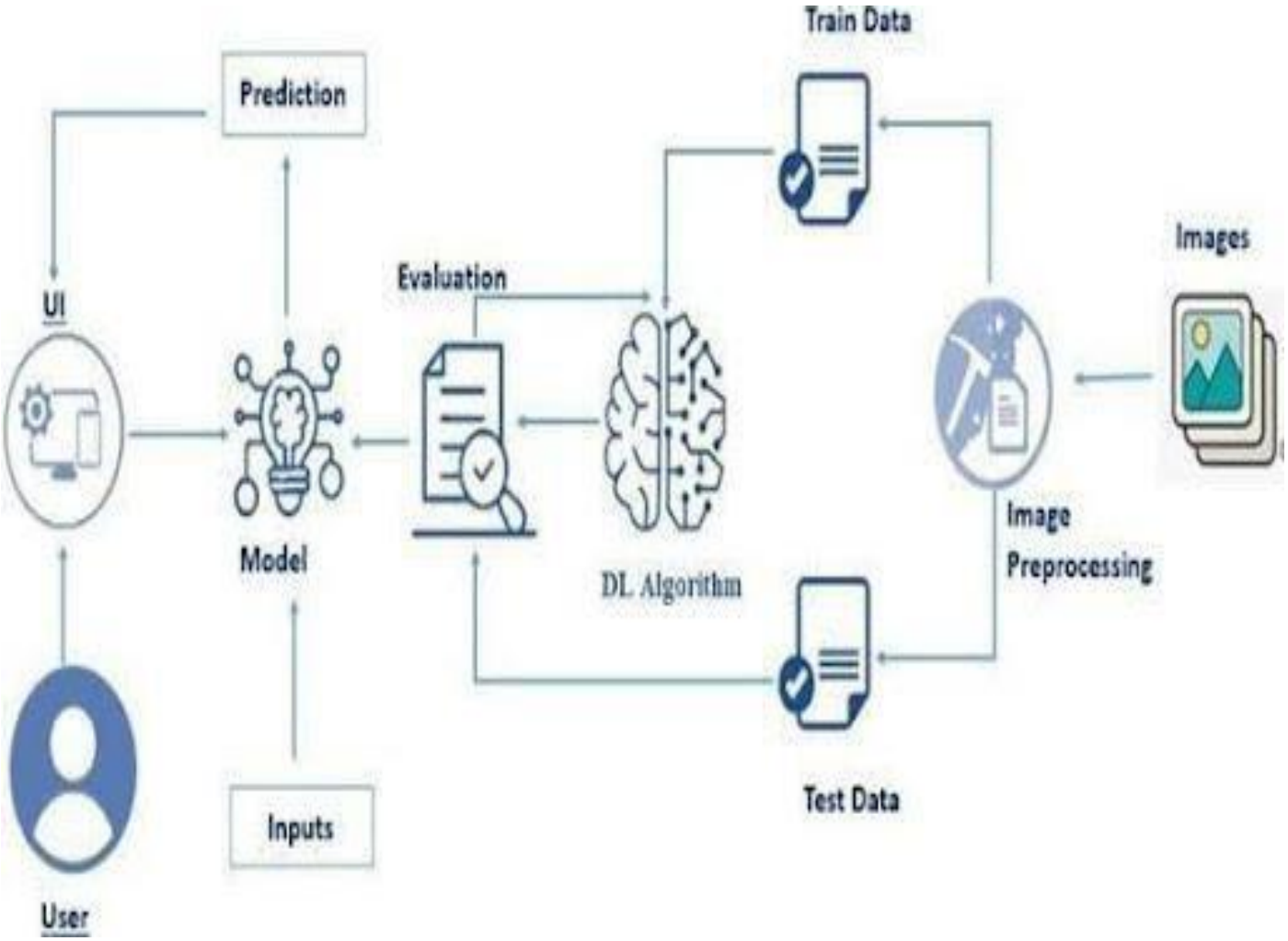
HematoVision aims to develop an accurate and efficient model for classifying blood cells by employing transfer learning techniques. Utilizing a dataset of 12,000 annotated blood cell images, categorized into distinct classes such as eosinophils, lymphocytes, monocytes, and neutrophils, the project leverages 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.

1. ProblemStatement:

- 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.

ARCHITECTURE:



Public

PREREQUISITES:

- Tocompletethisproject,youmustrequirethefollowingsoftware,concepts,andpackages
  - AnacondaNavigator:
    - RefertothelinkbelowtodownloadAnacondaNavigator
  - Pythonpackages:
  - Openanacondapromptasadministrator
  - Type“pipinstallnumpy”andclickenter.
  - Type“pipinstallpandas”andclickenter.
  - Type“pipinstallscikit-learn”andclickenter.
  - Type”pipinstallmatplotlib”andclickenter.
  - Type”pipinstallscipy”andclickenter.
  - Type”pipinstallseaborn”andclickenter.
  - Type”pipinstalltenserflow”andclickenter.
  - Type“pipinstallFlask”andclickenter.

A) PRIORKNOWLEDGE

- DLConcepts
  - NeuralNetworks::https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/
  - DeepLearningFrameworks::https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow
  - TransferLearning: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/
  - ConvolutionalNeuralNetworks (CNNs):https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - OverfittingandRegularization: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/
- FlaskBasics:https://www.youtube.com/watch?v=lj4I_CvBnt0

B) PROJECTOBJECTIVES

Bytheendofthisproject,youwill:

- KnowfundamentalconceptsandtechniquesusedforDeepLearning.
- Gainabroadunderstandingofdata.
- Haveknowledgeofpre-processingthedata/transformationtechniquesonoutliersandsomevisualization concepts.

Public

C) PROJECTFLOW

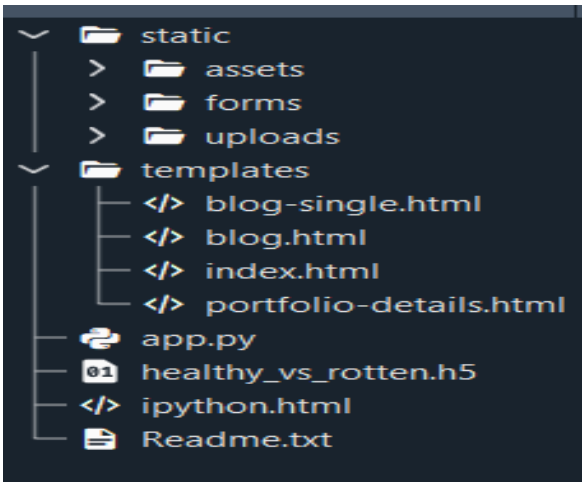
- TheuserinteractswiththeUI(UserInterface)tochoosetheimage.
- Thechosenimageisanalysedbythemodelwhichisintegratedwiththeflaskapplication.
- Oncethemodelanalysestheinputthepredictionisshowcasedontheui

Toaccomplishthis,wehavetocompleteallttheactivitieslistedbelow,

- DataCollection:Collectordownloadthedatasetthatyouwanttotrain.
- Datapre-processing
  - DataAugmentation
  - Splittingdataintotrainandtest
- Modelbuilding
  - Importthemodel-buildinglibraries
  - Initializingthemodel
  - Trainingandtestingthemodel
  - Evaluatingtheperformanceofthemodel
  - Savethemodel
- ApplicationBuilding
  - CreateanHTMLfile
  - Buildpythoncode

ProjectStructure

CreatetheProjectfolderwhichcontainsfilesasshownbelow



- WearebuildingaFlaskapplicationwithHTMLpagesstoredinthetemplatesfolderandaPython script app.py for scripting.
- Healthy_vs_rotten.h5isoursavedmodel.Further,wewillusethismodelforflaskintegration.

DATACOLLECTIONANDPREPARATION

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

A) COLLECTINGTHEDATASET

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.

Public

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

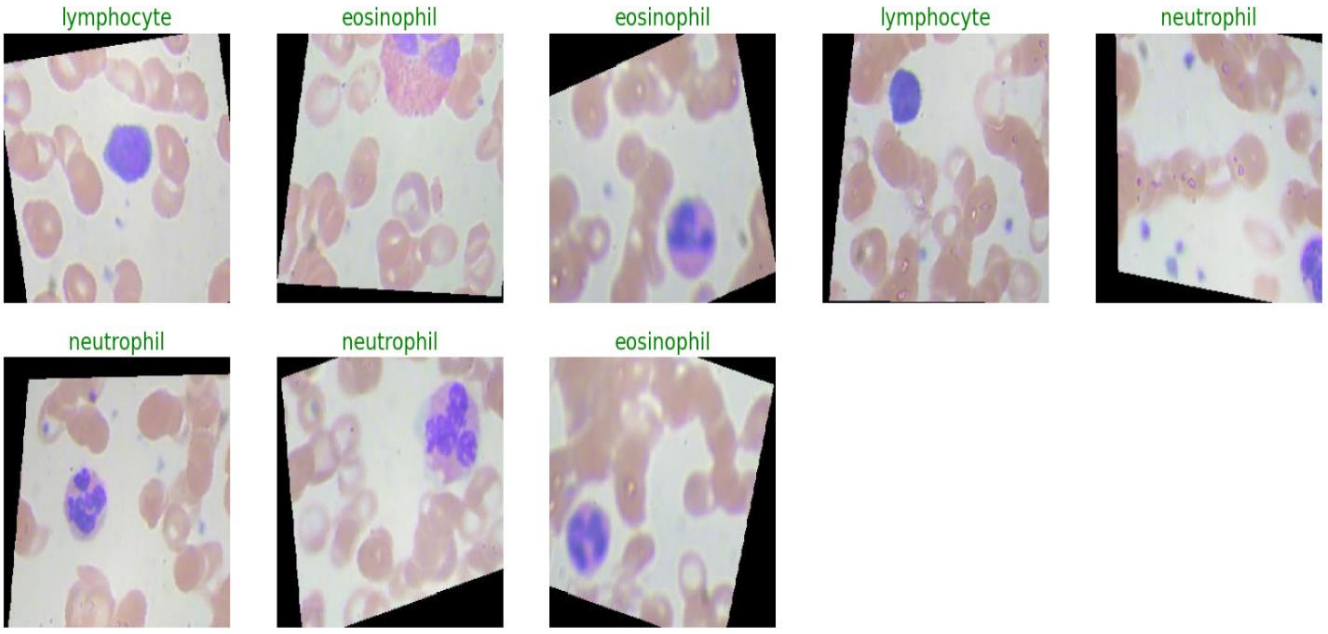
As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

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.

DataVisualization

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)
```



DATAAUGMENTATION

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.

SPLITDATAANDMODELBUILDING

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.
```

A) **ModelBuilding:**

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.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model

vgg = VGG16(include_top = False,input_shape = (224,224,3))

for layer in vgg.layers:
  print(layer)
```

```
<keras.src.engine.input_layer.InputLayer object at 0x79c096fde230>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fde4d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0081b7a90>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff7ef2f80>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0944058l0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c00834be30>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff6dad540>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fd2c20>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0944053601
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0944054b0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fc490>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae7d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dad4b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae020>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fff10>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe0b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe270>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fd300>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fe650>
```

```
len(vgg.layers)
```

```
19
```

```
for layer in vgg.layers:
  layer.trainable = False

x= Flatten()(vgg.output)

output = Dense(28, activation ='softmax')(x)

vgg16 = Model(vgg.input,output)

vgg16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
```

TESTINGMODEL&DATAPREDICTION

Evaluating the model

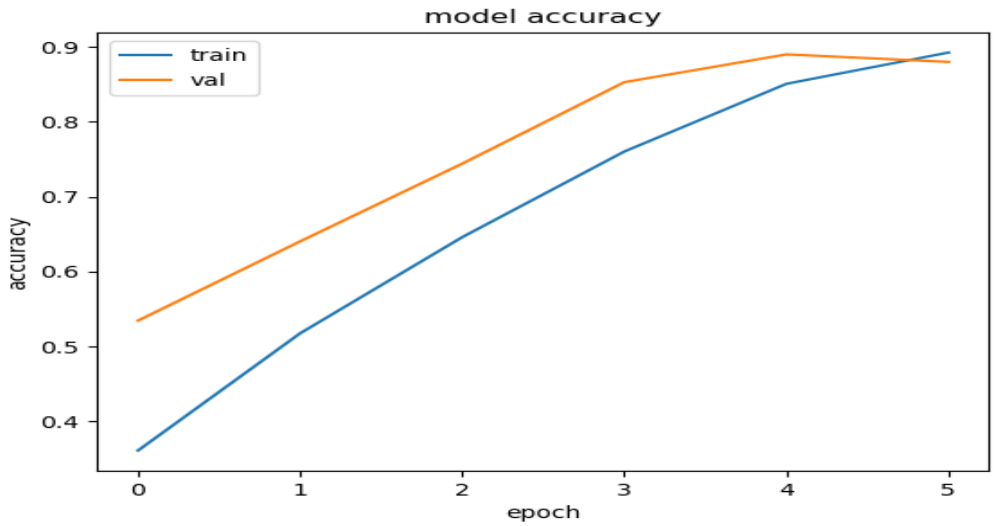
Here we have tested with the Mobilenet V2 Model With the help of the predict ()

```
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()
```



SAVINGTHEMODEL
Finally,wehavechosenthebestmodelnowsavingthatmodel

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

Toolsand TechnologiesUsed:

Anaconda,Python,Flask

Libraries:NumPy,Pandas,Scikit-learn,TensorFlow,Matplotlib

ConceptsandPrerequisites:

- DeepLearning&CNN
- TransferLearningusingVGG16
- Flaskforwebdeployment

Methodology:

- Dataset:28-classfruit/vegetableimagesfromKaggle
- Preprocessing:Normalization,Augmentation
- Model:VGG16withSoftMax,trainedover10epochs
- Accuracyvalidatedviapredictionsamples

SystemArchitecture:

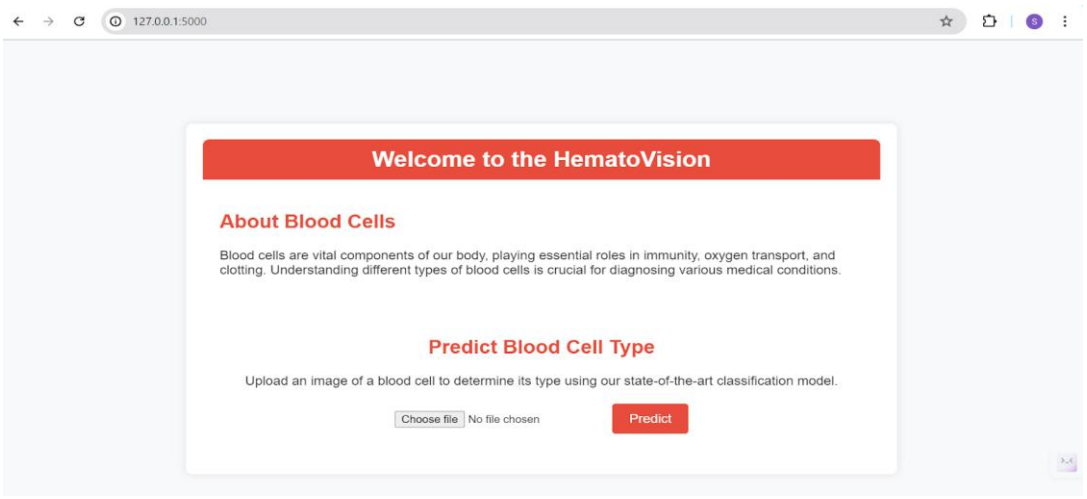
1. UI Input
2. FlaskAPI
3. ModelPrediction
4. OutputDisplayed

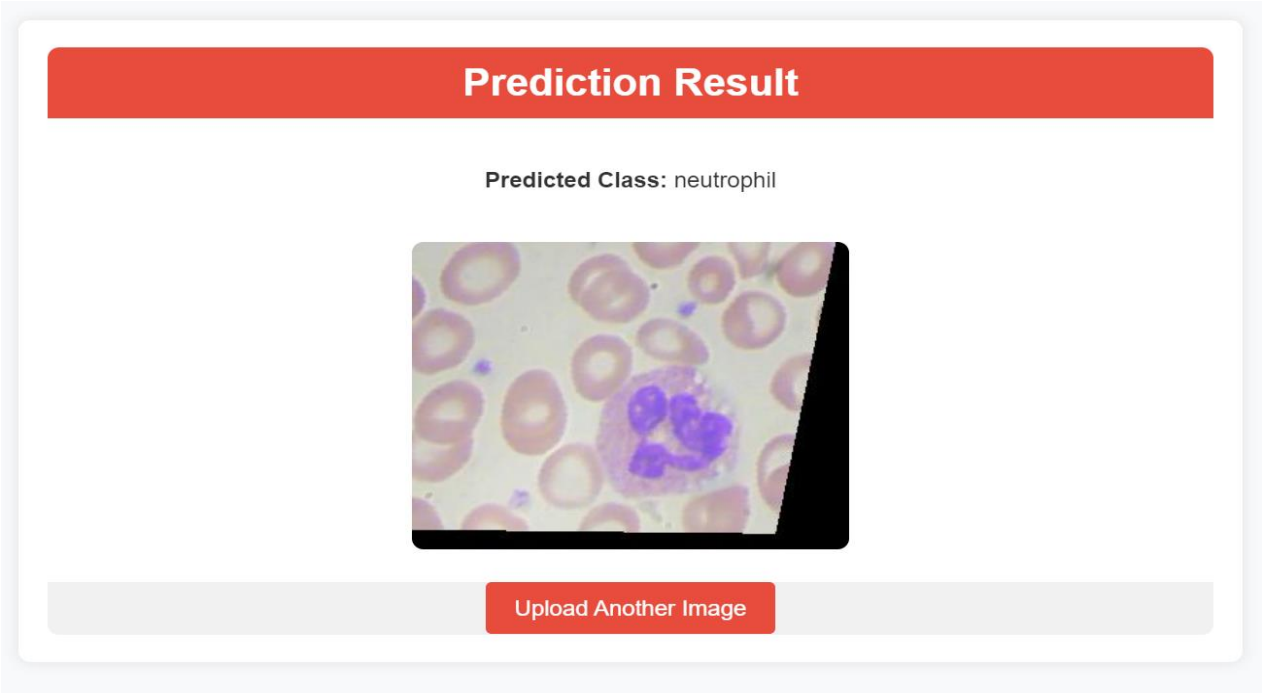
ApplicationBuilding

In thissection,wewillbebuildinga webapplicationthatisintegratedintothemodelwebuilt.A UIis providedfortheuseswherehe hastoenterthevaluesforpredictions.Theentervaluesaregivento the saved model and prediction is showcased on the UI.

- BuildingHTMLPages
- Buildingserver-sidescript

Results andObservations:





Conclusion:

Thisprojectshowshowtransferlearningenablesefficientclassificationandautomationinblood cells.

Future Scope:

- Broaderspoilagedetection
- IoT integration
- Mobileappsupport

References:

Kaggle Dataset

GeeksforGeeksVGG16

Analytics Vidhya Bl