

HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Category: Artificial Intelligence

Skills Required: Python, Deep Learning, Transfer Learning

Project Description:

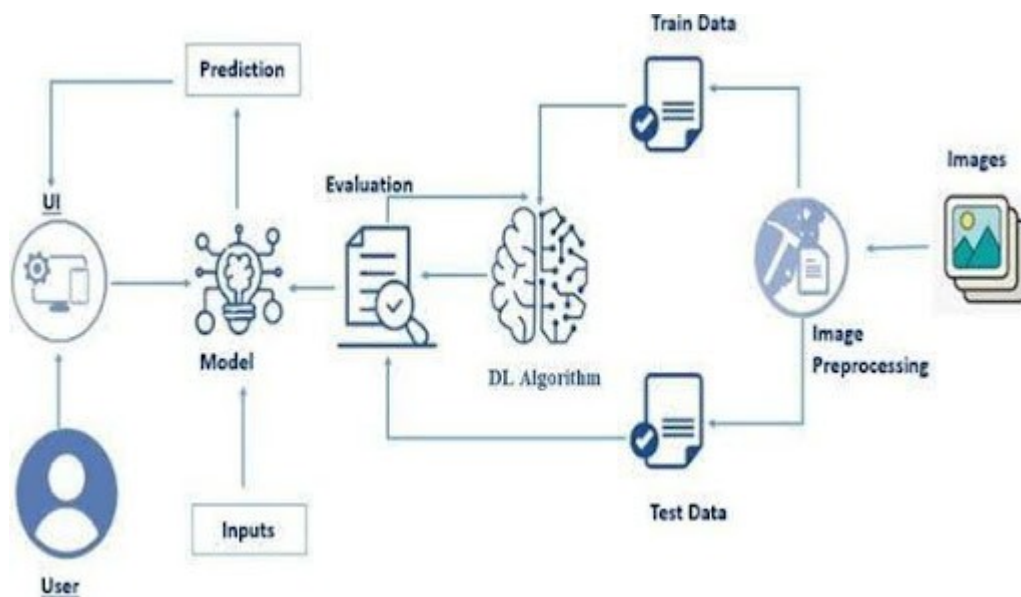
HematoVision aims to develop an accurate and efficient model for classifying blood cells by employing transfer learning techniques. Utilizing a dataset of 12,000 annotated blood cell images, categorized into distinct classes such as eosinophils, lymphocytes, monocytes, and neutrophils, the project leverages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

Technical Architecture:



Prerequisites:

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

Anaconda Navigator:

Refer to the link below to download Anaconda Navigator

link: [Anaconda](#)

Python packages:

Open anaconda prompt as administrator:

1. Type “pip install numpy” and click enter.
2. Type “pip install pandas” and click enter.
3. Type “pip install scikit-learn” and click enter.
4. Type ”pip install matplotlib” and click enter.
5. Type ”pip install scipy” and click enter.
6. Type ”pip install seaborn” and click enter.
7. Type ”pip install tensorflow” and click enter.
8. Type “pip install Flask” and click enter.

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](https://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 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.

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:

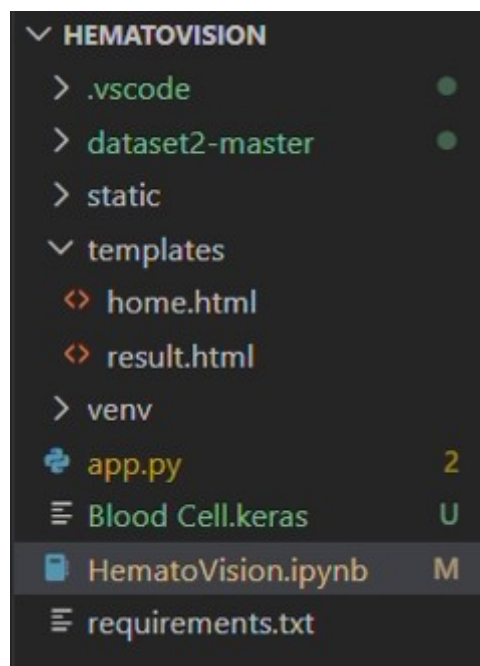
- Import the model-building libraries
- Initializing the model
- Training and testing the model
- Evaluating the performance of the model
- Save the model

Application Building:

- Create an HTML file
- Build python code

Project Structure:

Create the Project folder which contains files as shown below



We are building a Flask application with HTML pages stored in the templates folder and a Python script app.py for scripting.

Blood Cell.h5 is our saved model. Further, we will use this model for flask integration.

Data Collection and Preparation:

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.

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.

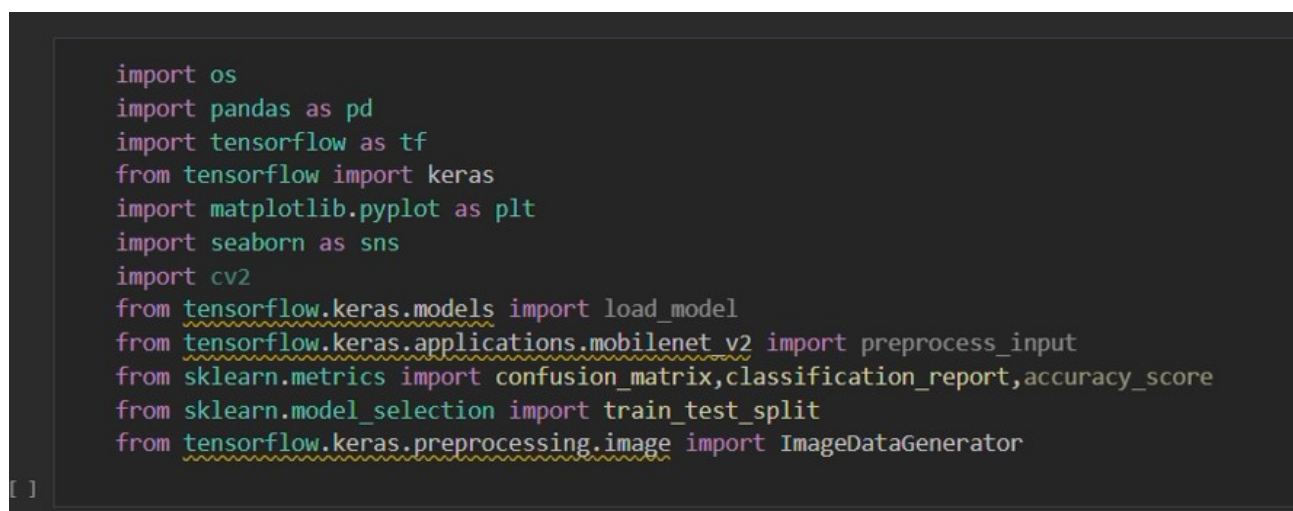
Link: <https://www.kaggle.com/datasets/paultimothymooney/blood-cells/data>

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, classification_report, accuracy_score
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.

```
data_dir='/kaggle/input/blood-cells/dataset2-master/dataset2-master/images/TRAIN'
class_labels=['EOSINOPHIL','LYMPHOCYTE','MONOCYTE','NEUTROPHIL']
```

```
filepaths=[]
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'):
            filepaths.append(os.path.join(class_dir,file))
            labels.append(label)
```

```
bloodcell_df=pd.DataFrame({'filepath':filepaths,'label':labels})
bloodcell_df=bloodcell_df.sample(frac=1).reset_index(drop=True)
bloodcell_df.head()
```

	filepath	label
0	/kaggle/input/blood-cells/dataset2-master/data...	EOSINOPHIL
1	/kaggle/input/blood-cells/dataset2-master/data...	EOSINOPHIL
2	/kaggle/input/blood-cells/dataset2-master/data...	MONOCYTE
3	/kaggle/input/blood-cells/dataset2-master/data...	NEUTROPHIL
4	/kaggle/input/blood-cells/dataset2-master/data...	NEUTROPHIL

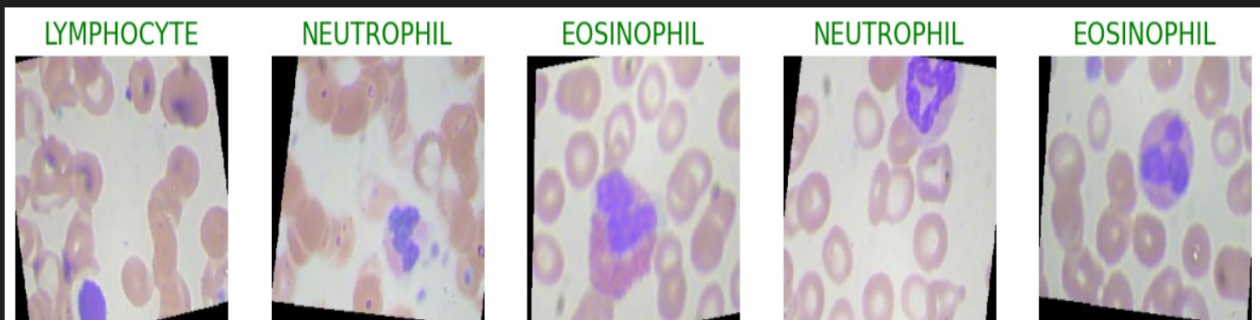
Data Visualization:

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 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=(15,15))
    length=len(images)
    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:

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:

```
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_images.shape)
print(val_set.shape)
print(test_images.shape)
print(train_set.shape)

[ ]

... (6969, 2)
(1992, 2)
(2988, 2)
(7965, 2)

image_gen=ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input)
train=image_gen.flow_from_dataframe(dataframe=train_set,x_col='filepath',y_col='label',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='filepath',y_col='label',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='filepath',y_col='label',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.
```

In this project, we have already separated data for training and testing.

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),activation='relu',input_shape=(244,244,3)),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=256,kernel_size=(5,5),activation='relu',padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(3,3)),

    keras.layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256,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.MaxPooling2D(pool_size=(2,2)),

    keras.layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256,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.MaxPooling2D(pool_size=(2,2)),

    keras.layers.Conv2D(filters=512,kernel_size=(3,3),activation='relu',padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(pool_size=(2,2)),

    keras.layers.Flatten(),
    keras.layers.Dense(1024,activation='relu'),
    keras.layers.Dropout(0.5),
```

```

keras.layers.Dense(512,activation='relu'),
keras.layers.Dropout(0.5),
keras.layers.Dense(4,activation='softmax')
])

model.compile(
    loss='categorical_crossentropy',
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy'])

model.summary()

```

/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass `activity_regularizer` to `super().__init__` (activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 237, 237, 128)	24,704
batch_normalization (BatchNormalization)	(None, 237, 237, 128)	512
conv2d_1 (Conv2D)	(None, 237, 237, 256)	819,456
batch_normalization_1 (BatchNormalization)	(None, 237, 237, 256)	1,024
max_pooling2d (MaxPooling2D)	(None, 79, 79, 256)	0
conv2d_2 (Conv2D)	(None, 79, 79, 256)	590,080

batch_normalization_2 (BatchNormalization)	(None, 79, 79, 256)	1,024
conv2d_3 (Conv2D)	(None, 79, 79, 256)	590,080
batch_normalization_3 (BatchNormalization)	(None, 79, 79, 256)	1,024
conv2d_4 (Conv2D)	(None, 79, 79, 256)	590,080
batch_normalization_4 (BatchNormalization)	(None, 79, 79, 256)	1,024
conv2d_5 (Conv2D)	(None, 79, 79, 512)	1,180,160
batch_normalization_5 (BatchNormalization)	(None, 79, 79, 512)	2,048
max_pooling2d_1 (MaxPooling2D)	(None, 39, 39, 512)	0
conv2d_6 (Conv2D)	(None, 39, 39, 256)	1,179,904
batch_normalization_6 (BatchNormalization)	(None, 39, 39, 256)	1,024
conv2d_7 (Conv2D)	(None, 39, 39, 256)	590,080
batch_normalization_7 (BatchNormalization)	(None, 39, 39, 256)	1,024
conv2d_8 (Conv2D)	(None, 39, 39, 512)	1,180,160
batch_normalization_8 (BatchNormalization)	(None, 39, 39, 512)	2,048
max_pooling2d_2 (MaxPooling2D)	(None, 19, 19, 512)	0
conv2d_9 (Conv2D)	(None, 19, 19, 512)	2,359,808
batch_normalization_9 (BatchNormalization)	(None, 19, 19, 512)	2,048

max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 512)	0
flatten (Flatten)	(None, 41472)	0
dense (Dense)	(None, 1024)	42,468,352
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524,800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 4)	2,052

... Total params: 52,112,516 (198.79 MB)

... Trainable params: 52,106,116 (198.77 MB)

... Non-trainable params: 6,400 (25.00 KB)

```
history=model.fit(train,epochs=5,validation_data=val,verbose=1)
```

Python

... Epoch 1/5

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its `self.warn_if_super_not_called()`

996/996 ————— 756s 711ms/step - accuracy: 0.2758 - loss: 7.9989 - val_accuracy: 0.3881 - val_loss: 1.2581

Epoch 2/5

996/996 ————— 666s 669ms/step - accuracy: 0.3651 - loss: 1.3743 - val_accuracy: 0.4121 - val_loss: 1.4057

Epoch 3/5

996/996 ————— 666s 669ms/step - accuracy: 0.3651 - loss: 1.3743 - val_accuracy: 0.4121 - val_loss: 1.4057

Epoch 3/5

996/996 ————— 666s 669ms/step - accuracy: 0.4034 - loss: 1.2830 - val_accuracy: 0.3830 - val_loss: 1.6827

Epoch 4/5

996/996 ————— 665s 668ms/step - accuracy: 0.4965 - loss: 1.1568 - val_accuracy: 0.3715 - val_loss: 1.9828

Epoch 5/5

996/996 ————— 664s 667ms/step - accuracy: 0.5612 - loss: 0.9681 - val_accuracy: 0.6340 - val_loss: 0.8402

```
history1=model.fit(train,epochs=1,validation_data=val,verbose=1)
```

996/996 ————— 665s 668ms/step - accuracy: 0.6319 - loss: 0.8060 - val_accuracy: 0.6616 - val_loss: 0.6865

Testing Model & Data Prediction:

Evaluating the model

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

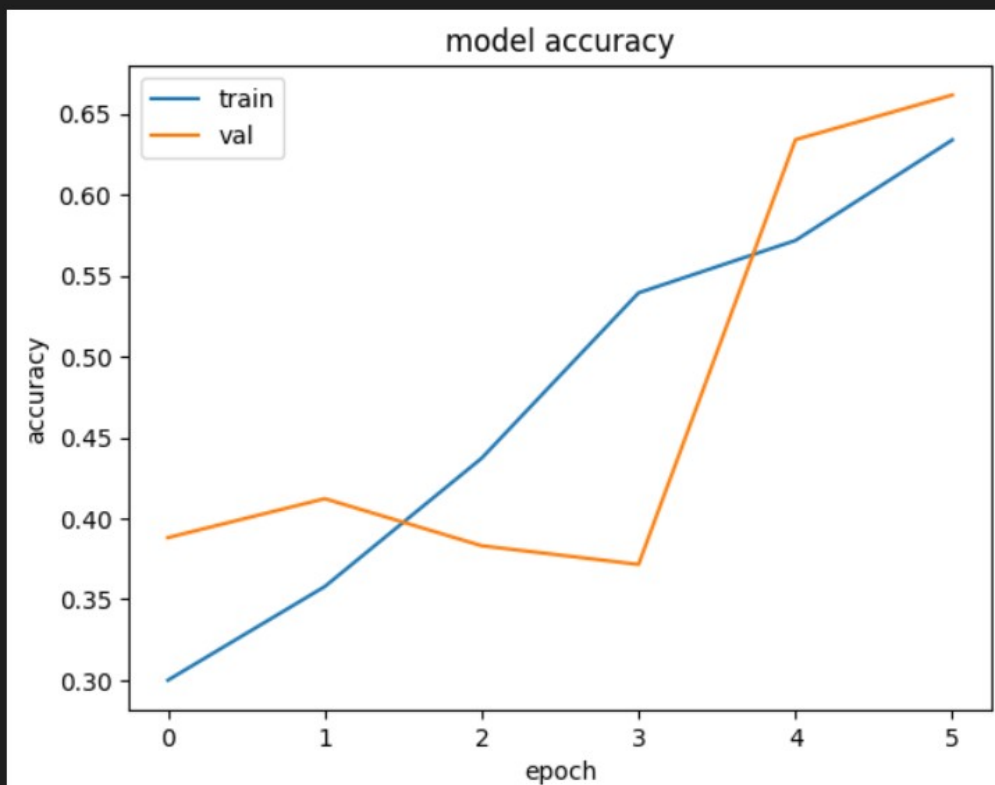
```
pred=model.predict(test)
pred=np.argmax(pred,axis=1)

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

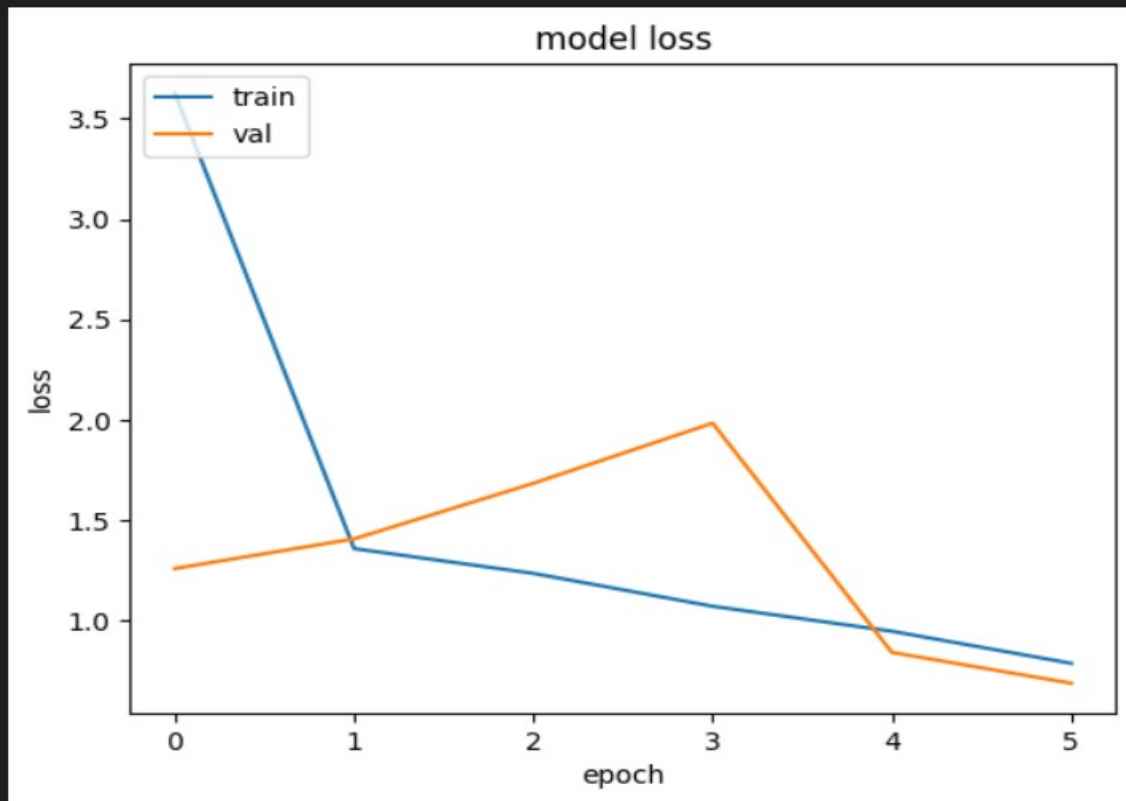
Python

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its
self._warn_if_super_not_called()
374/374 115s 306ms/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()
```



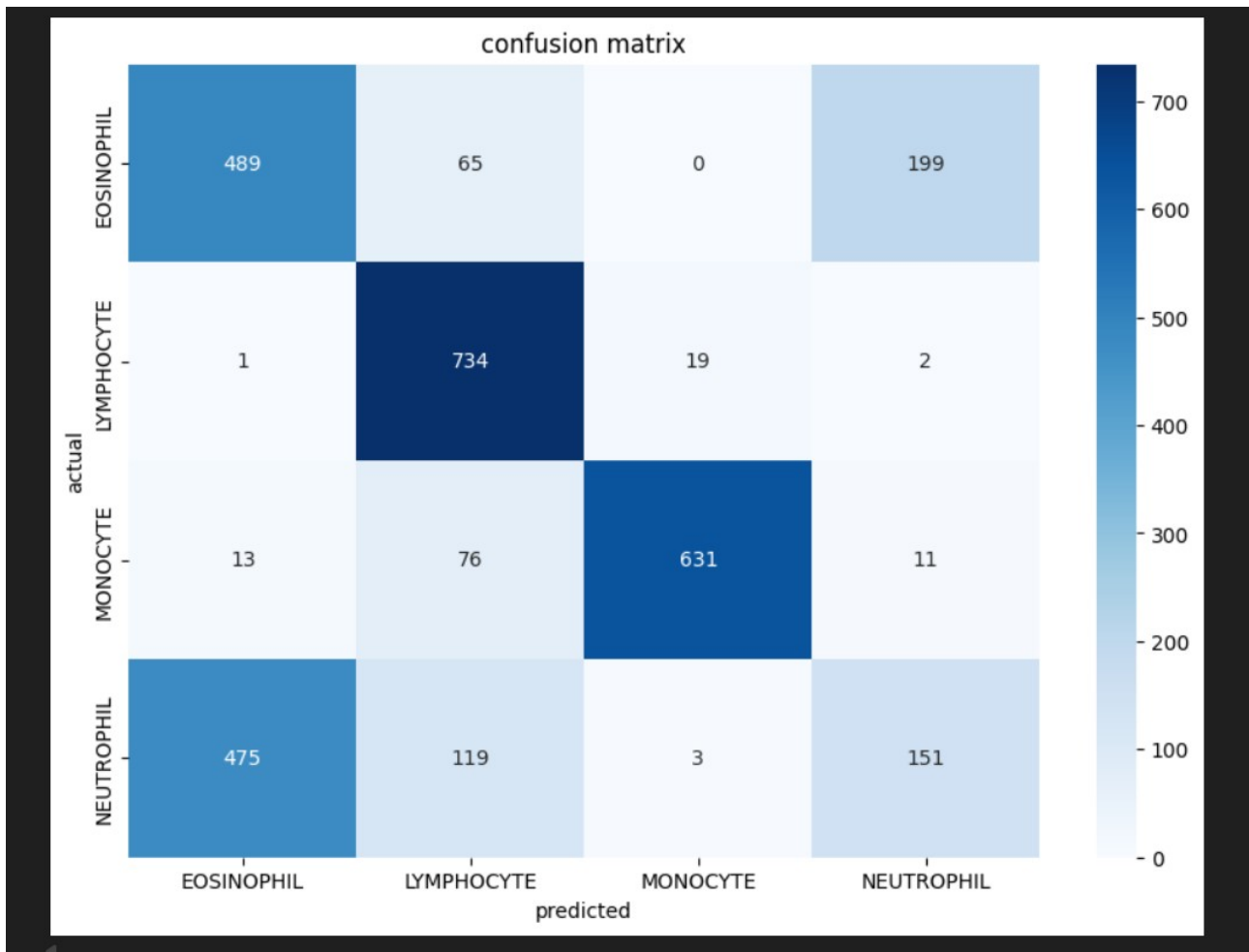
```
plt.plot(history.history['loss']+ history1.history['loss'])
plt.plot(history.history['val_loss']+ history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','val'],loc='upper left')
plt.show()
```



```
y_test=test_images.label
print(classification_report(y_test,pred2))
```

	precision	recall	f1-score	support
EOSINOPHIL	0.50	0.65	0.56	753
LYMPHOCYTE	0.74	0.97	0.84	756
MONOCYTE	0.97	0.86	0.91	731
NEUTROPHIL	0.42	0.20	0.27	748
accuracy			0.67	2988
macro avg	0.66	0.67	0.65	2988
weighted avg	0.65	0.67	0.65	2988

```
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.

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

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 werkzeug.utils import secure_filename
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
```

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.keras")
class_labels = ['eosinophil', 'lymphocyte', 'monocyte', 'neutrophil']
UPLOAD_DIR = "static"

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:
            os.makedirs(UPLOAD_DIR, exist_ok=True)
            safe_name = secure_filename(file.filename)
            file_path = os.path.join(UPLOAD_DIR, safe_name)
            file.save(file_path)
            predicted_class_label, img_rgb = predict_image_class(file_path, model)

            _, 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 :

1. 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.

```

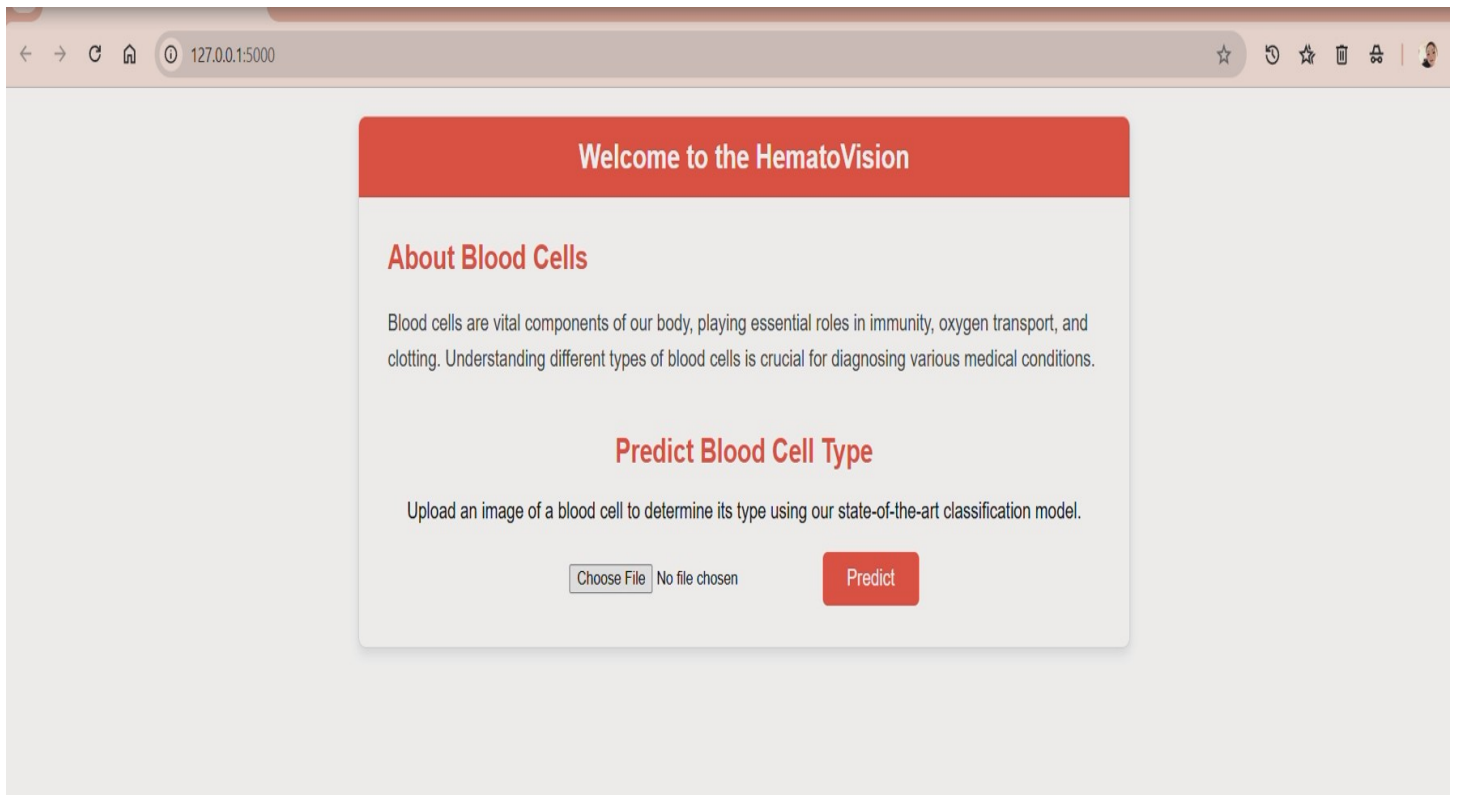
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)

```


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

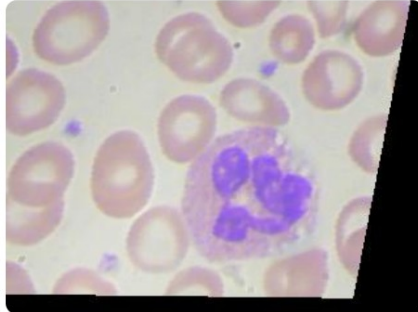
Predict Blood Cell Type

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



Prediction Result

Predicted Class: neutrophil



Upload Another Image

Test For Class-2 : Monocyte

Predict Blood Cell Type

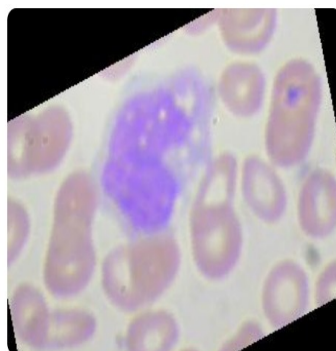
Upload an image of a blood cell to determine its type using our state-of-the-art classification model.

Choose file _3_9423.jpeg

Predict

Prediction Result

Predicted Class: monocyte



Upload Another Image

Test For Class-3 : Lymphocyte

Predict Blood Cell Type

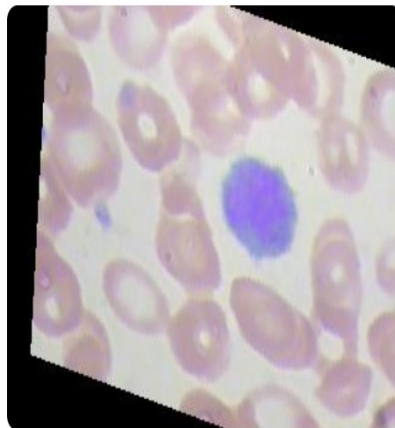
Upload an image of a blood cell to determine its type using our state-of-the-art classification model.

Choose file _5_9201.jpeg

Predict

Prediction Result

Predicted Class: lymphocyte



Upload Another Image

Test For Class-4 : Eosinophil

Predict Blood Cell Type

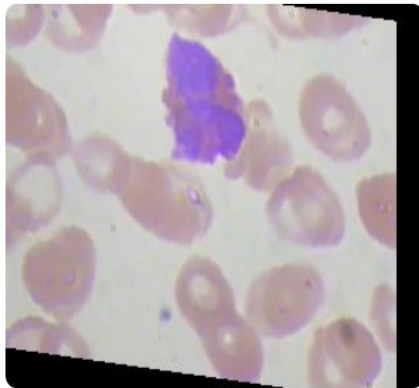
Upload an image of a blood cell to determine its type using our state-of-the-art classification model.

Choose file _3_9885.jpeg

Predict

Prediction Result

Predicted Class: eosinophil



[Upload Another Image](#)