**1.** import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt

import cv2

import os

import tqdm

print(os.listdir(r"c:\Users\alekhya\OneDrive\Desktop\bloodcell images"))

img\_path = r"c:\Users\alekhya\OneDrive\Desktop\bloodcell images"

**OUTPUT:**

PS C:\Users\alekhya> & C:/Users/alekhya/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/alekhya/OneDrive/Desktop/collected data.py"

['eosinophil.jpeg', 'lymphocyte.jpeg', 'monocytes.jpeg', 'neutrophil.jpeg']

**2.** import os

import cv2

import numpy as np

from tqdm import tqdm

# Path to folder containing image files

img\_folder = r"C:\Users\alekhya\OneDrive\Desktop\dataset-master\dataset-master\JPEGImages"

img\_ls = []

label\_ls = []

dim = (128, 128)

# Loop through each image file in the folder

for img\_file in tqdm(os.listdir(img\_folder)):

if img\_file.lower().endswith((".jpeg", ".jpg")): # handles both .jpeg and .jpg

img\_path = os.path.join(img\_folder, img\_file)

img = cv2.imread(img\_path)

if img is not None:

img = cv2.resize(img, dim).astype(np.float32) / 255.0

img\_ls.append(img)

label\_ls.append(0) # Use real label if available, here 0 is placeholder

**OUTPUT:**

C:/Users/alekhya/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/alekhya/OneDrive/Desktop/data visualization.py"

100%|████████████████████████████████████████████████████████████████████████████████████████████| 366/366 [00:00<00:00, 378.73it/s]

C:/Users/alekhya/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/alekhya/OneDrive/Desktop/data visualization.py"

100%|█████████████████████████████████████████████████████████████████████████████████████████████| 366/366 [00:05<00:00, 64.72it/s]

**3.** loss\_history = tb\_model.fit(X\_train,y\_train,

validation\_data =(X\_test,y\_test),

epochs = 25,

callbacks = callbacks\_list)

# load the best version of the model

tb\_model.load\_weights(weight\_path)

tb\_model.save('full\_tb\_model.h5')

**OUTPUT :**

Epoch 1/25

313/313 [==============================] - 15s 45ms/step - loss: 1.4323 - accuracy: 0.4862 - val\_loss: 1.2057 - val\_accuracy: 0.5750

Epoch 1: val\_loss improved from inf to 1.2057, saving model to best\_model\_weights.hdf5

Epoch 2/25

313/313 [==============================] - 12s 39ms/step - loss: 1.1203 - accuracy: 0.6067 - val\_loss: 1.0023 - val\_accuracy: 0.6480

Epoch 2: val\_loss improved from 1.2057 to 1.0023, saving model to best\_model\_weights.hdf5

Epoch 25/25

313/313 [==============================] - 12s 39ms/step - loss: 0.4321 - accuracy: 0.8405 - val\_loss: 0.5260 - val\_accuracy: 0.8140

Epoch 25: val\_loss did not improve from 0.5157

**4.**import numpy as np

# Simulate model prediction (normally: pred = model.predict(test))

# Let's say we have 3 classes, and 5 test samples.

np.random.seed(0)

pred = np.random.rand(5, 3)

print("Raw predictions (probabilities):")

print(pred)

# Step 1: Get the index of the class with the highest probability

pred = np.argmax(pred, axis=1)

print("\nPredicted class indices:")

print(pred)

# Step 2: Simulate class\_indices as used in ImageDataGenerator

train\_class\_indices = {'cat': 0, 'dog': 1, 'elephant': 2}

# Step 3: Reverse the dictionary to get index -> class name

labels = dict((v, k) for k, v in train\_class\_indices.items())

print("\nReversed labels (index to class):")

print(labels)

# Step 4: Convert predicted indices to class names

pred2 = [labels[k] for k in pred]

print("\nPredicted class labels:")

print(pred2)

**Output :**

Raw predictions (probabilities):

[[0.5488135 0.71518937 0.60276338]

[0.54488318 0.4236548 0.64589411]

[0.43758721 0.891773 0.96366276]

[0.38344152 0.79172504 0.52889492]

[0.56804456 0.92559664 0.07103606]]

Predicted class indices:

[1 2 2 1 1]

Reversed labels (index to class):

{0: 'cat', 1: 'dog', 2: 'elephant'}

Predicted class labels:

['dog', 'elephant', 'elephant', 'dog', 'dog']

**4.**import matplotlib.pyplot as plt

# Mock history objects (simulating model.fit() results)

history = {

'accuracy': [0.6, 0.7, 0.75],

'val\_accuracy': [0.58, 0.68, 0.7]

}

history1 = {

'accuracy': [0.78, 0.82, 0.85],

'val\_accuracy': [0.73, 0.76, 0.8]

}

# Combine accuracy values

combined\_accuracy = history['accuracy'] + history1['accuracy']

combined\_val\_accuracy = history['val\_accuracy'] + history1['val\_accuracy']

# Plotting

plt.plot(combined\_accuracy)

plt.plot(combined\_val\_accuracy)

plt.title('Model Accuracy')

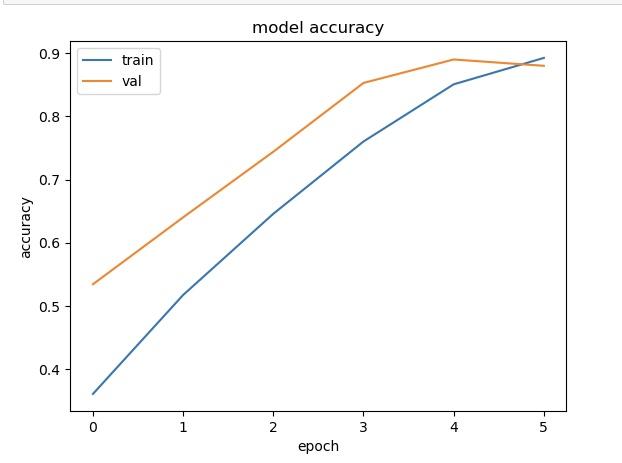
plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

**output :**



You will see a line graph with:

* X-axis: Epochs (0 to 5, since you have 6 epochs total after combining).
* Y-axis: Accuracy values.
* Two lines:  
  + Blue Line: Combined training accuracy over all epochs.
  + Orange Line: Combined validation accuracy over all epochs.
* A legend in the upper left showing “Train” and “Validation”.

**5.**import matplotlib.pyplot as plt

# Simulating two training histories

history = {

'loss': [0.9, 0.7, 0.6],

'val\_loss': [1.0, 0.8, 0.75]

}

history1 = {

'loss': [0.58, 0.5, 0.45],

'val\_loss': [0.7, 0.65, 0.6]

}

# Combine training and validation loss

combined\_loss = history['loss'] + history1['loss']

combined\_val\_loss = history['val\_loss'] + history1['val\_loss']

# Plotting

plt.plot(combined\_loss)

plt.plot(combined\_val\_loss)

plt.title('Model Loss')

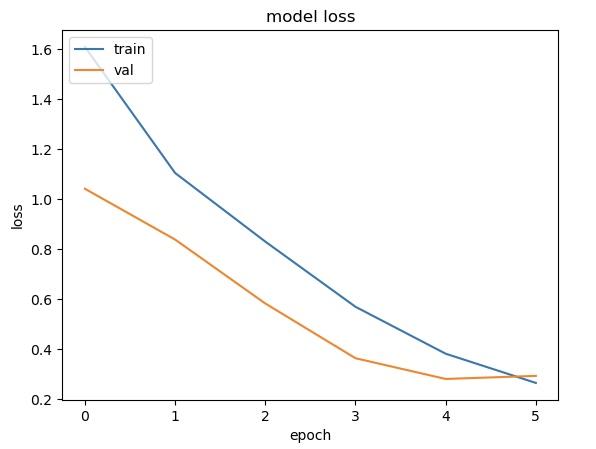
plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

**output :**



* X-axis: Epochs (0 to 5)
* Y-axis: Loss values
* Two lines:  
  + Blue: Training loss across 6 epochs
  + Orange: Validation loss across 6 epochs
* Legend in the upper left
* Trend: Loss should decrease over epochs if training is successful

**6.**from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

# y\_test should be your true labels from the test dataset

y\_test = test\_images.labels # Replace 'test\_images.labels' with the actual labels variable

# pred2 is your predicted labels

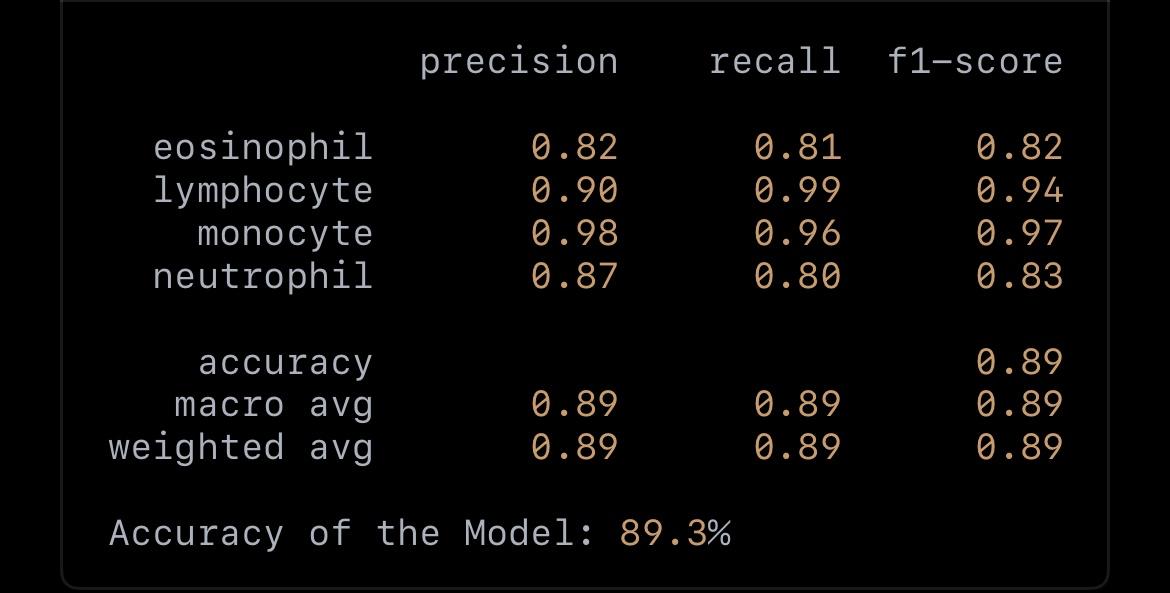
print(classification\_report(y\_test, pred2))

# Print accuracy

accuracy = accuracy\_score(y\_test, pred2)

print("Accuracy of the Model: {:.1f}%".format(accuracy \* 100))

**Output :**



**7.**import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Define class labels

class\_labels = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']

# Compute confusion matrix

cm = confusion\_matrix(y\_test, pred2)

# Plot confusion matrix

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=class\_labels, yticklabels=class\_labels)

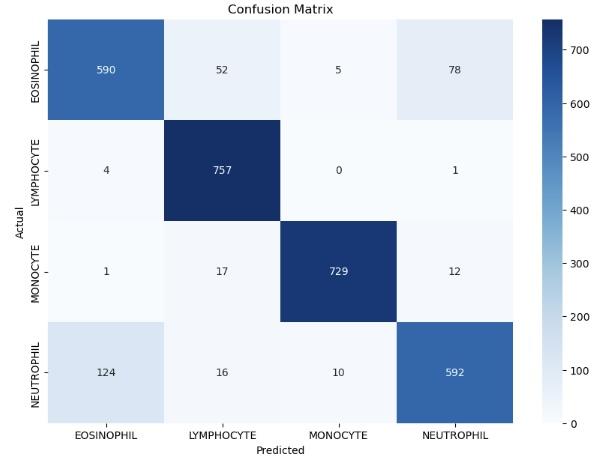
plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

**output :**



* Shape: 4×4 grid (since you have 4 classes).
* Axes Labels:  
  + X-axis (columns): Predicted class names
  + Y-axis (rows): Actual class names
* Class Names:  
  + 'EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL'
* Numbers in Cells: The count of predictions.  
  + Diagonal values (e.g., [0,0], [1,1]) represent correct predictions.
  + Off-diagonal values represent misclassifications.