**A yellow and black logo

Description automatically generated****A logo of a university

Description automatically generatedDEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Subject: Digital Image Processing**

**Assignment # 02**

**SUBMITTED TO:**

**Lab Engineer**

**SUBMITTED BY:**

**Student Name: Eshmal Rumman**

**Reg # 413285**

**DE- 44 Dept CSE Syndicate B**

# **White Blood Cell Analysis and Classification**

**Flow Diagram**

A diagram of a process

AI-generated content may be incorrect.

A diagram of a process

AI-generated content may be incorrect.

A diagram of a data analysis process

AI-generated content may be incorrect.

**Feature Visualization**

A diagram of a company

AI-generated content may be incorrect.

A diagram of a company

AI-generated content may be incorrect.A diagram of a company

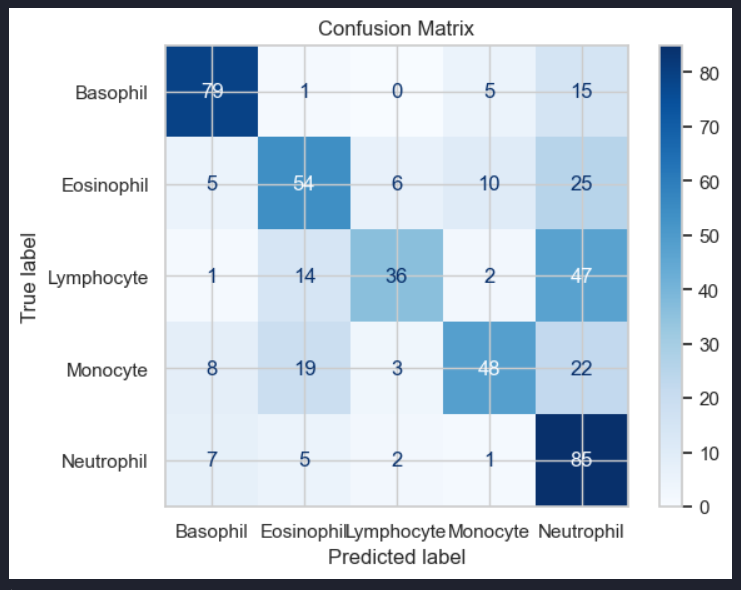
AI-generated content may be incorrect.A diagram of a company

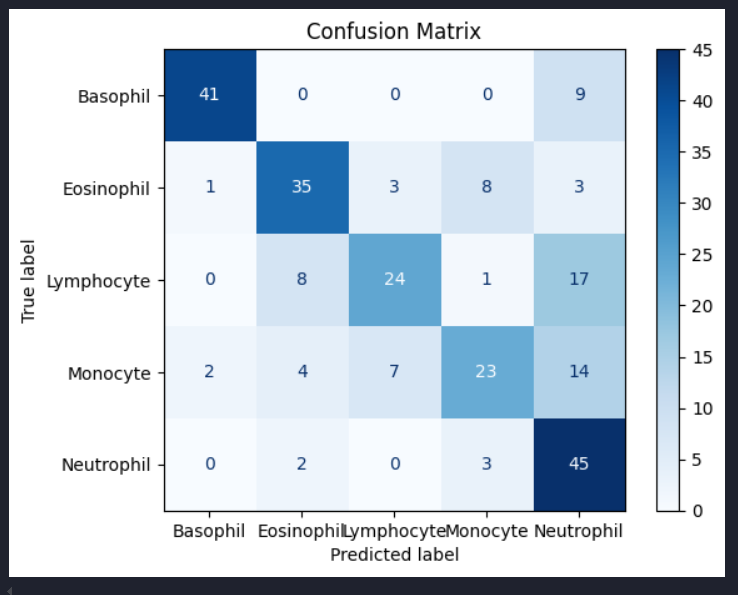
AI-generated content may be incorrect.A diagram of a company

AI-generated content may be incorrect.A diagram of a company

AI-generated content may be incorrect.

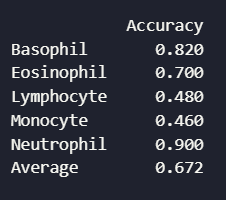
**Confusion Matrix**

****

****

**Tabular Result**

****

****

**Setbacks**

Initially used LBP to extract features and then visualize them. Got 255 features but all of them were overlapping a great deal due to which those features couldn’t be used for classification. Then shifted to K-Means clustering, the extracted features were overlapping as well but not as much as in the LBP. So in the end used K-Means Clustering and used the features to classify the WBCs.

**Code & Outputs**

|  |
| --- |
| import cv2 as cv  import numpy as np  import matplotlib.pyplot as plt  import os  from skimage.measure import label  import pandas as pd  import seaborn as sns  import imageio  from sklearn.metrics import confusion\_matrix, accuracy\_score, ConfusionMatrixDisplay |

|  |
| --- |
| Code |
| def load\_images(input\_path, output\_path\_g, output\_path\_r):      input\_folder = input\_path      output\_folder\_g = output\_path\_g      output\_folder\_r = output\_path\_r      for filename in os.listdir(input\_folder):          if filename.lower().endswith('.jpg'):              img\_path = os.path.join(input\_folder, filename)              img = cv.imread(img\_path)       # loads in BGR format              img = cv.resize(img, (256, 256))                # convert to RGB format              img\_rgb = cv.cvtColor(img, cv.COLOR\_BGR2RGB)                # convert to grayscale              img\_gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY)                # using median blur (great for salt-n-pepper noise) with kernel size 5              img\_blur\_g = cv.medianBlur(img\_gray, 5)              img\_blur\_r = cv.medianBlur(img\_rgb, 5)                # doing equalised histogram for contrast enhancement              img\_eq\_g = cv.equalizeHist(img\_blur\_g)                output\_path = os.path.join(output\_folder\_g, filename)              cv.imwrite(output\_path, img\_eq\_g)              output\_path = os.path.join(output\_folder\_r, filename)              cv.imwrite(output\_path, img\_blur\_r)              print(f"Processed and saved: {filename}")  load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Train\\Neutrophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray\\Neutrophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB\\Neutrophil")  load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Train\\Monocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray\\Monocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB\\Monocyte")  load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Train\\Lymphocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray\\Lymphocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB\\Lymphocyte")  load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Train\\Eosinophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray\\Eosinophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB\\Eosinophil")  load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Train\\Basophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray\\Basophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB\\Basophil") |
| Output |
|  |

|  |
| --- |
| Code |
| def kmeans\_segmentation(image, k=8):      # Reshape image to a 2D array of pixels      pixel\_values = image.reshape((-1, 3))      pixel\_values = np.float32(pixel\_values)      # Apply K-means clustering      \_, labels, centers = cv.kmeans(pixel\_values, k, None,                                      criteria=(cv.TERM\_CRITERIA\_EPS + cv.TERM\_CRITERIA\_MAX\_ITER, 100, 0.2),                                      attempts=10, flags=cv.KMEANS\_RANDOM\_CENTERS)      labels = labels.flatten()      segmented\_image = centers[labels].reshape(image.shape).astype(np.uint8)      return segmented\_image, labels.reshape(image.shape[:2])  def extract\_nucleus(image, label\_map):      unique\_labels = np.unique(label\_map)      mean\_intensity = [np.mean(image[label\_map == label]) for label in unique\_labels]      nucleus\_label = unique\_labels[np.argmin(mean\_intensity)]      nucleus\_mask = (label\_map == nucleus\_label).astype(np.uint8)      return nucleus\_mask  def clean\_nucleus(nucleus\_mask):      labeled, num = label(nucleus\_mask, return\_num=True)      if num == 0:          return np.zeros\_like(nucleus\_mask)      sizes = [(labeled == i).sum() for i in range(1, num + 1)]      largest\_label = np.argmax(sizes) + 1      return (labeled == largest\_label).astype(np.uint8)  def extract\_features(image, nucleus\_mask):      # Compute area ratio      nucleus\_area = np.sum(nucleus\_mask)      total\_area = image.shape[0] \* image.shape[1]      area\_ratio = nucleus\_area / total\_area      # Compute circularity      contours, \_ = cv.findContours(nucleus\_mask, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)      if contours:          largest\_contour = max(contours, key=cv.contourArea)          perimeter = cv.arcLength(largest\_contour, True)          circularity = (4 \* np.pi \* cv.contourArea(largest\_contour)) / (perimeter \*\* 2 + 1e-5)      else:          circularity = 0      # Compute mean and standard deviation of RGB values      mask = nucleus\_mask.astype(*bool*)      pixels = image[mask]      mean\_rgb = np.mean(pixels, axis=0) if pixels.size > 0 else [0, 0, 0]      std\_rgb = np.std(pixels, axis=0) if pixels.size > 0 else [0, 0, 0]      granularity = np.var(image[mask]) if pixels.size > 0 else 0      red\_green\_ratio = mean\_rgb[0] / (mean\_rgb[1] + 1e-5)      return {          "area\_ratio": area\_ratio,          "circularity": circularity,          "mean\_r": mean\_rgb[0],          "mean\_g": mean\_rgb[1],          "mean\_b": mean\_rgb[2],          "granularity": granularity,          "red\_green\_ratio": red\_green\_ratio      }    def process\_images\_and\_save\_csv(image\_dir, output\_csv, k=8):      data = []      for cell\_type in os.listdir(image\_dir):          cell\_dir = os.path.join(image\_dir, cell\_type)          if not os.path.isdir(cell\_dir):              continue          for file in os.listdir(cell\_dir):              image\_path = os.path.join(cell\_dir, file)              image = cv.imread(image\_path)              if image is None:                  print(f"Failed to load image: {file}")                  continue              # Step 1: K-means segmentation              segmented\_image, label\_map = kmeans\_segmentation(image, k=8)              # Step 2: Extract nucleus              nucleus\_mask = extract\_nucleus(segmented\_image, label\_map)              # Step 3: Clean nucleus              cleaned\_mask = clean\_nucleus(nucleus\_mask)              # Step 4: Extract features              features = extract\_features(image, cleaned\_mask)              features["label"] = cell\_type              features["filename"] = file  # Add filename to keep track              data.append(features)      # Step 5: Save to CSV      df = pd.DataFrame(data)      df.to\_csv(output\_csv, index=False)      print(f"Features saved to: {output\_csv}")  process\_images\_and\_save\_csv("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputRGB", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\output\_features.csv") |
| Output |
|  |

|  |
| --- |
| Code |
| def plot\_feature\_vs\_label(df, features):      sns.set(style="whitegrid")      num\_features = len(features)        plt.figure(figsize=(6 \* num\_features, 5))        for i, feature in enumerate(features, 1):          plt.subplot(1, num\_features, i)          sns.boxplot(x="label", y=feature, data=df, palette="Set2")          plt.title(f"{feature} vs WBC Type")          plt.xticks(rotation=45)        plt.tight\_layout()      plt.show()  df = pd.read\_csv("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\output\_features.csv")  plot\_feature\_vs\_label(df, ["area\_ratio", "circularity", "mean\_r", "granularity", "red\_green\_ratio"]) |
| Output |
|  |

|  |
| --- |
| Code |
| def classify\_wbc(circularity, area\_ratio, mean\_r, granularity, red\_green\_ratio):      # Reference ranges based on visual boxplot analysis      wbc\_profiles = {          "Lymphocyte": {              "circularity": (0.6, 0.9),              "area\_ratio": (0.04, 0.07),              "mean\_r": (110, 130),              "granularity": (1600, 2200),              "red\_green\_ratio": (3.5, 6.0)          },          "Eosinophil": {              "circularity": (0.45, 0.75),              "area\_ratio": (0.03, 0.06),              "mean\_r": (120, 150),              "granularity": (2000, 3000),              "red\_green\_ratio": (4.0, 6.5)          },          "Monocyte": {              "circularity": (0.3, 0.6),              "area\_ratio": (0.06, 0.09),              "mean\_r": (130, 160),              "granularity": (1600, 2500),              "red\_green\_ratio": (3.0, 6.0)          },          "Basophil": {              "circularity": (0.1, 0.35),              "area\_ratio": (0.08, 0.12),              "mean\_r": (85, 115),              "granularity": (1000, 1900),              "red\_green\_ratio": (1.5, 3.5)          },          "Neutrophil": {              "circularity": (0.35, 0.6),              "area\_ratio": (0.03, 0.06),              "mean\_r": (95, 125),              "granularity": (1400, 2200),              "red\_green\_ratio": (4.5, 7.0)          }      }      # Normalised range-based distance      def feature\_diff(value, range\_tuple):          low, high = range\_tuple          if low <= value <= high:              return 0          elif value < low:              return (low - value) / (high - low)          else:              return (value - high) / (high - low)      features = {          "circularity": circularity,          "area\_ratio": area\_ratio,          "mean\_r": mean\_r,          "granularity": granularity,          "red\_green\_ratio": red\_green\_ratio      }      min\_score = *float*('inf')      predicted\_label = None      for label, profile in wbc\_profiles.items():          score = sum(              feature\_diff(features[feature], profile[feature])              for feature in features          )          if score < min\_score:              min\_score = score              predicted\_label = label      return predicted\_label      def classify\_from\_csv(csv\_path, output\_path=None):      df = pd.read\_csv(csv\_path)      # Check if the necessary features are present      required\_cols = ["filename", "area\_ratio", "circularity", "mean\_r", "granularity", "red\_green\_ratio"]      if not all(col in df.columns for col in required\_cols):          raise *ValueError*(f"CSV file must contain the following columns: {', '.join(required\_cols)}")      # Apply classification row by row      df["predicted\_label"] = df.apply(lambda row: classify\_wbc(          row["circularity"],          row["area\_ratio"],          row["mean\_r"],          row["granularity"],          row["red\_green\_ratio"]      ), axis=1)      # Save the results if an output path is provided      if output\_path:          df.to\_csv(output\_path, index=False)          print(f"Classification results saved to: {output\_path}")      return df  csv\_input = "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\output\_features.csv"  csv\_output = "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results.csv"  classified\_df = classify\_from\_csv(csv\_input, csv\_output) |
| Output |
|  |

|  |
| --- |
| Code |
| def evaluate\_predictions(csv\_path, actual\_label\_col="label", predicted\_label\_col="predicted\_label"):      df = pd.read\_csv(csv\_path)      if actual\_label\_col not in df.columns or predicted\_label\_col not in df.columns:          print(f"[ERROR] '{actual\_label\_col}' or '{predicted\_label\_col}' column not found in the CSV.")          return      y\_true = df[actual\_label\_col]      y\_pred = df[predicted\_label\_col]      # Confusion Matrix      cm = confusion\_matrix(y\_true, y\_pred, labels=np.unique(y\_true))      acc = accuracy\_score(y\_true, y\_pred)      print(f"[INFO] Accuracy: {acc \* 100:.2f}%")      disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_true))      disp.plot(cmap=plt.cm.Blues)      plt.title("Confusion Matrix")      plt.show()  evaluate\_predictions("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results.csv") |
| Output |
|  |

|  |
| --- |
| Code |
| def apply\_padding(image, kernel\_size, pad\_value):      padding\_size = kernel\_size // 2      return cv.copyMakeBorder(image, padding\_size, padding\_size, padding\_size, padding\_size, cv.BORDER\_CONSTANT, value=pad\_value)  def rotated\_lbp(img):      weights = np.array([1, 2, 4, 8, 16, 32, 64, 128])        lbpFinal = np.zeros\_like(img, dtype=np.uint8)        kernel\_size = 3      rep = np.zeros((kernel\_size, kernel\_size), dtype=np.uint8)        padded\_img = apply\_padding(img, kernel\_size, 0)      height, width = img.shape        for i in range(height):          for j in range(width):              region = padded\_img[i:i+kernel\_size, j:j+kernel\_size]              center\_pixel = region[1, 1]                binary\_values = (region >= center\_pixel).astype(np.uint8)                neighbor\_pixels = np.zeros(8, dtype=np.uint8)              ind1 = 0              for m in range(3):                  for n in range(3):                      if m == 1 and n == 1:                          continue                      neighbor\_pixels[ind1] = binary\_values[m, n]                      ind1 += 1                dif = np.abs(neighbor\_pixels - center\_pixel)              D = np.argmax(dif)              rotated = np.roll(binary\_values, -D)                ind = 0              for k in range(3):                  for l in range(3):                      if k == 1 and l == 1:                          continue                      rep[k, l] = rotated[k, l] \* weights[ind]                      ind += 1                lbpFinal[i, j] = np.sum(rep)        return lbpFinal    def extract\_features\_lbp(input\_path):      data = []      num\_bins = 256  # Define number of bins beforehand      for cell\_type in os.listdir(input\_path):          class\_path = os.path.join(input\_path, cell\_type)          if not os.path.isdir(class\_path):              print("??", class\_path)              continue          for img\_file in os.listdir(class\_path):              img\_path = os.path.join(class\_path, img\_file)              img = cv.imread(img\_path, cv.IMREAD\_GRAYSCALE)              if img is None:                  print("????", img\_path)                  continue              # Apply LBP              lbp = rotated\_lbp(img)              # Compute histogram (feature vector)              hist, \_ = np.histogram(lbp.ravel(), bins=num\_bins, range=(0, num\_bins))                # Normalize histogram              hist = hist.astype("float")              hist /= (hist.sum() + 1e-6)              # Append to data              feature\_row = [img\_file] + *list*(hist) + [cell\_type]              data.append(feature\_row)              print(f"{img\_file} – Histogram sum: {hist.sum()}")      if not data:          print("No data extracted. Check your folder paths or image read logic.")          return      # Construct column names based on num\_bins      columns = ['filename'] + [f'lbp\_{i}' for i in range(num\_bins)] + ['label']        df = pd.DataFrame(data, columns=columns)      df.to\_csv('E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\lbp\_features.csv', index=False)    extract\_features\_lbp("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputGray")  df = pd.read\_csv('E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\lbp\_features.csv')  # Plot one feature across all cell types  for i in range(2):  # First 10 LBP bins      plt.figure(figsize=(8, 4))      sns.boxplot(x='label', y=f'lbp\_{i}', data=df)      plt.title(f'Boxplot of lbp\_{i} by WBC Type')      plt.xticks(rotation=45)      plt.tight\_layout()      plt.show()  # Example: Scatter plot for two features  sns.scatterplot(data=df, x='lbp\_159', y='lbp\_215', hue='label')  plt.title("Feature Distribution (lbp\_20 vs lbp\_45)")  plt.show() |
| Output |
|  |

|  |
| --- |
| Code |
| import pandas as pd  # Load your classification results  df = pd.read\_csv("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results.csv")  # Create a confusion matrix grouped by true label  grouped = df.groupby('label')  # Calculate per class accuracy  per\_class\_accuracy = {}  for label, group in grouped:      correct = (group['label'] == group['predicted\_label']).sum()      total = len(group)      accuracy = correct / total      per\_class\_accuracy[label] = round(accuracy, 4)  # Convert to DataFrame  accuracy\_df = pd.DataFrame.from\_dict(per\_class\_accuracy, orient='index', columns=['Accuracy'])  # Calculate average per-class accuracy (macro avg)  average\_accuracy = accuracy\_df['Accuracy'].mean()  accuracy\_df.loc['Average'] = average\_accuracy  # Save or print  print(accuracy\_df) |
| Output |
|  |
| Code |
| def test\_csv():      #load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Test\\Basophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTg\\Basophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr\\Basophil")      #load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Test\\Neutrophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTg\\Neutrophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr\\Neutrophil")      #load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Test\\Eosinophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTg\\Eosinophil", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr\\Eosinophil")      #load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Test\\Lymphocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTg\\Lymphocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr\\Lymphocyte")      #load\_images("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\Test\\Monocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTg\\Monocyte", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr\\Monocyte")        #process\_images\_and\_save\_csv("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\oTr", "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\output\_features\_test.csv")        #csv\_input = "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\output\_features\_test.csv"      #csv\_output = "E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results\_test.csv"      #df = classify\_from\_csv(csv\_input, csv\_output)        evaluate\_predictions("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results\_test.csv")        # Create a confusion matrix grouped by true label      df = pd.read\_csv("E:\\6th Semester\\DIP\\Assignment 2\\A2\_wbc\_data\\wbc\_data\\outputCSV\\classification\_results\_test.csv")      grouped = df.groupby('label')      # Calculate per class accuracy      per\_class\_accuracy = {}      for label, group in grouped:          correct = (group['label'] == group['predicted\_label']).sum()          total = len(group)          accuracy = correct / total          per\_class\_accuracy[label] = round(accuracy, 4)      # Convert to DataFrame      accuracy\_df = pd.DataFrame.from\_dict(per\_class\_accuracy, orient='index', columns=['Accuracy'])      # Calculate average per-class accuracy (macro avg)      average\_accuracy = accuracy\_df['Accuracy'].mean()      accuracy\_df.loc['Average'] = average\_accuracy      # Save or print      print(accuracy\_df)    test\_csv() |
| Output |
|  |