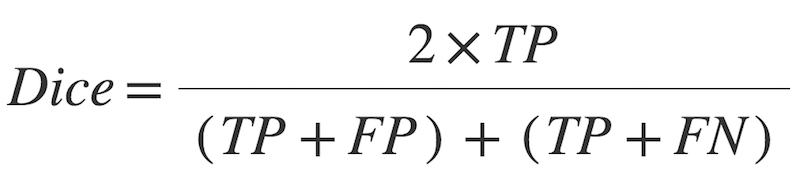
import cv2  
import numpy as np  
#from sklearn.cluster  
#import KMeans  
import zipfile  
  
# Load ZIP file containing images  
with zipfile.ZipFile('C://Users//Amina Qadeer//PycharmProjects//pythonProject//Original Images-20230404T044216Z-001.zip', 'r') as zip\_ref:  
 zip\_ref.extractall('./images')  
  
# Preprocess images  
def preprocess\_image('C://Users//Amina Qadeer//PycharmProjects//pythonProject//IMD049.bmp'):  
 # Load image  
 img = cv2.imread(image\_path)  
  
 # Convert to grayscale  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 # Apply median blur  
 blur = cv2.medianBlur(gray, 5)  
  
 # Apply adaptive thresholding  
 thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY\_INV, 11, 2)  
  
 return thresh  
  
# Perform segmentation and K-means clustering to extract features  
def extract\_features(image\_path):  
 # Preprocess image  
 img = preprocess\_image(image\_path)  
  
 # Perform segmentation  
 contours, \_ = cv2.findContours(img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 # Extract features using K-means clustering  
 features = []  
 for contour in contours:  
 # Compute bounding box  
 x, y, w, h = cv2.boundingRect(contour)  
  
 # Crop contour  
 crop = img[y:y+h, x:x+w]  
  
 # Resize to fixed size  
 crop = cv2.resize(crop, (64, 64))  
  
 # Flatten image  
 crop\_flat = crop.flatten()  
  
 # Add to features list  
 features.append(crop\_flat)  
  
 # Perform K-means clustering  
 kmeans = KMeans(n\_clusters=2)  
 kmeans.fit(features)  
 centroids = kmeans.cluster\_centers\_  
  
 # Compute distance between centroids  
 dist = np.linalg.norm(centroids[0] - centroids[1])  
  
 # Return features  
 return dist  
  
image\_path = ''C://Users//Amina Qadeer//PycharmProjects//pythonProject//IMD049\_lesion.bmp''  
features = extract\_features(image\_path)  
print(features)

**Dice coefficient**

The Dice score is not only a measure of how many positives you find, but it also penalizes for the false positives that the method finds, similar to precision. so it is more similar to precision than accuracy.



In [9]:

**def** dice\_coe(y\_true, y\_pred, smooth **=** 100):

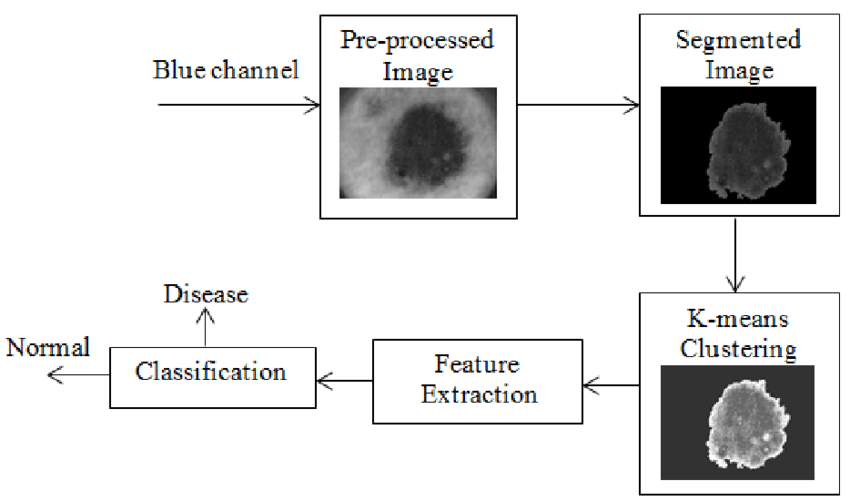
y\_true\_f **=** K**.**flatten(y\_true)

y\_pred\_f **=** K**.**flatten(y\_pred)

intersection **=** K**.**sum(y\_true\_f **\*** y\_pred\_f)

**return** (2. **\*** intersection **+** smooth) **/** (K**.**sum(y\_true\_f) **+** K**.**sum(y\_pred\_f) **+** smooth)

BLOCK DIAGRAM:



We could map the given images to foreground to there respective masks and in a loop check if it completely fits.