Natural Language Processing – Mini Project

Team Members :

1. Pushkar Adhyapak – F20111026
2. Anurag Shevale – F20111022
3. Siddharth Sonawane – F20111017
4. Prathamesh Khaire – F20111009

Title : Feature Extraction For Using Zernike Moments

Objectives :

1. To understand the importance of feature extraction in machine learning.
2. To comprehend the mathematical properties of Zernike moments.
3. To implement and compare Zernike moment feature extraction with other techniques in Python.

Appratus:

- Python Programming Knowledge.
- NLP Algorithms.
- Computer with appropriate software for data analysis and programming (e.g., Python)
- We have generated our own images as the dataset.
- A software library for image processing and analysis, such as OpenCV
- Jupyter Notebook/Google Collab

Abstact :

Zernike moments are effective image features for shape classification due to their rotational invariance, ensuring consistent descriptors regardless of shape orientation. Additional techniques can make them scale and translational invariant, enhancing robustness across different scales and positions. Careful application is crucial, requiring consideration of various factors. This paper explores optimization techniques for Zernike moments as shape descriptors, demonstrating exceptional accuracy, up to 100%, in experimental results. These findings underscore the effectiveness and potential of Zernike moments in shape classification.

Introduction :

Zernike moments are a set of orthogonal moments that have proven to be effective in various image processing and pattern recognition tasks, particularly in shape analysis and classification. These moments are computed over a circular region and are invariant to rotation, making them robust descriptors for shapes regardless of their orientation. Originally introduced by Frits Zernike in 1934 for the analysis of optical aberrations in microscope lenses, Zernike moments have found wide applications beyond their original scope.

One of the key advantages of Zernike moments is their ability to provide compact yet discriminative representations of shapes. This is achieved by capturing the shape's radial and angular characteristics, enabling effective shape matching and recognition. Additionally, Zernike moments are computationally efficient to compute, especially when compared to other shape descriptors like Fourier descriptors. This efficiency makes them suitable for real-time applications where computational resources are limited.

Despite their advantages, the application of Zernike moments requires careful consideration of certain factors. For instance, the choice of the circular region over which the moments are computed can impact the descriptor's performance. Moreover, the normalization of Zernike moments is critical to ensure their scale and translation invariance, which are essential for handling shapes of varying sizes and positions. Overall, Zernike moments offer a powerful and versatile tool for shape analysis and classification in image processing.

Implementation :

Input :

```
!pip install numpy --upgrade
!pip install mahotas

# import the necessary packages
from scipy.spatial import distance as dist
import numpy as np
import mahotas
import cv2
from google.colab.patches import cv2_imshow as cvim
import imutils
```

```
def describe_shapes(image):
    # initialize the list of shape features
    shapeFeatures = []

    # convert the image to grayscale, blur it, and threshold it
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (13, 13), 0)
    thresh = cv2.threshold(blurred, 50, 255, cv2.THRESH_BINARY)[1]

    # perform a series of dilations and erosions to close holes
    # in the shapes
    thresh = cv2.dilate(thresh, None, iterations=4)
    thresh = cv2.erode(thresh, None, iterations=2)

    # detect contours in the edge map
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    # loop over the contours
    for c in cnts:
        # create an empty mask for the contour and draw it
        mask = np.zeros(image.shape[:2], dtype="uint8")
        cv2.drawContours(mask, [c], -1, 255, -1)

        # extract the bounding box ROI from the mask
        (x, y, w, h) = cv2.boundingRect(c)
        roi = mask[y:y + h, x:x + w]

        # compute Zernike Moments for the ROI and update the list
        # of shape features
        features = mahotas.features.zernike_moments(roi,
cv2.minEnclosingCircle(c)[1], degree=8)
        shapeFeatures.append(features)

    #Return a tuple of the contours and shapes
    return (cnts, shapeFeatures)

#Load the reference image containing the object we want to detect,
#Then describe the game region
refImage = cv2.imread("/content/iphone.jpeg")
(_, gameFeatures) = describe_shapes(refImage)

#Load the shapes image, then describe each of the images in the image
shapesImage = cv2.imread("/content/allshapes.jpeg")
(cnts, shapeFeatures) = describe_shapes(shapesImage)

# compute the Euclidean distances between the video game features
# and all other shapes in the second image, then find index of the
# smallest distance
D = dist.cdist(gameFeatures, shapeFeatures)
i = np.argmin(D)

# loop over the contours in the shapes image
for (j, c) in enumerate(cnts):
    # if the index of the current contour does not equal the index
    # contour of the contour with the smallest distance, then draw
    # it on the output image
    if i != j:
        box = cv2.minAreaRect(c)
        box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else
cv2.boxPoints(box))
        cv2.drawContours(shapesImage, [box], -1, (0, 0, 255), 2)

    # draw the bounding box around the detected shape
    box = cv2.minAreaRect(cnts[i])
    box = np.int0(cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box))
    cv2.drawContours(shapesImage, [box], -1, (0, 255, 0), 2)
    (x, y, w, h) = cv2.boundingRect(cnts[i])
    cv2.putText(shapesImage, "FOUND!", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 3)

# show the output images
cvim(refImage)
cvim(shapesImage)
cv2.waitKey(0)
```

Input Dataset : Created By Our Team Members.

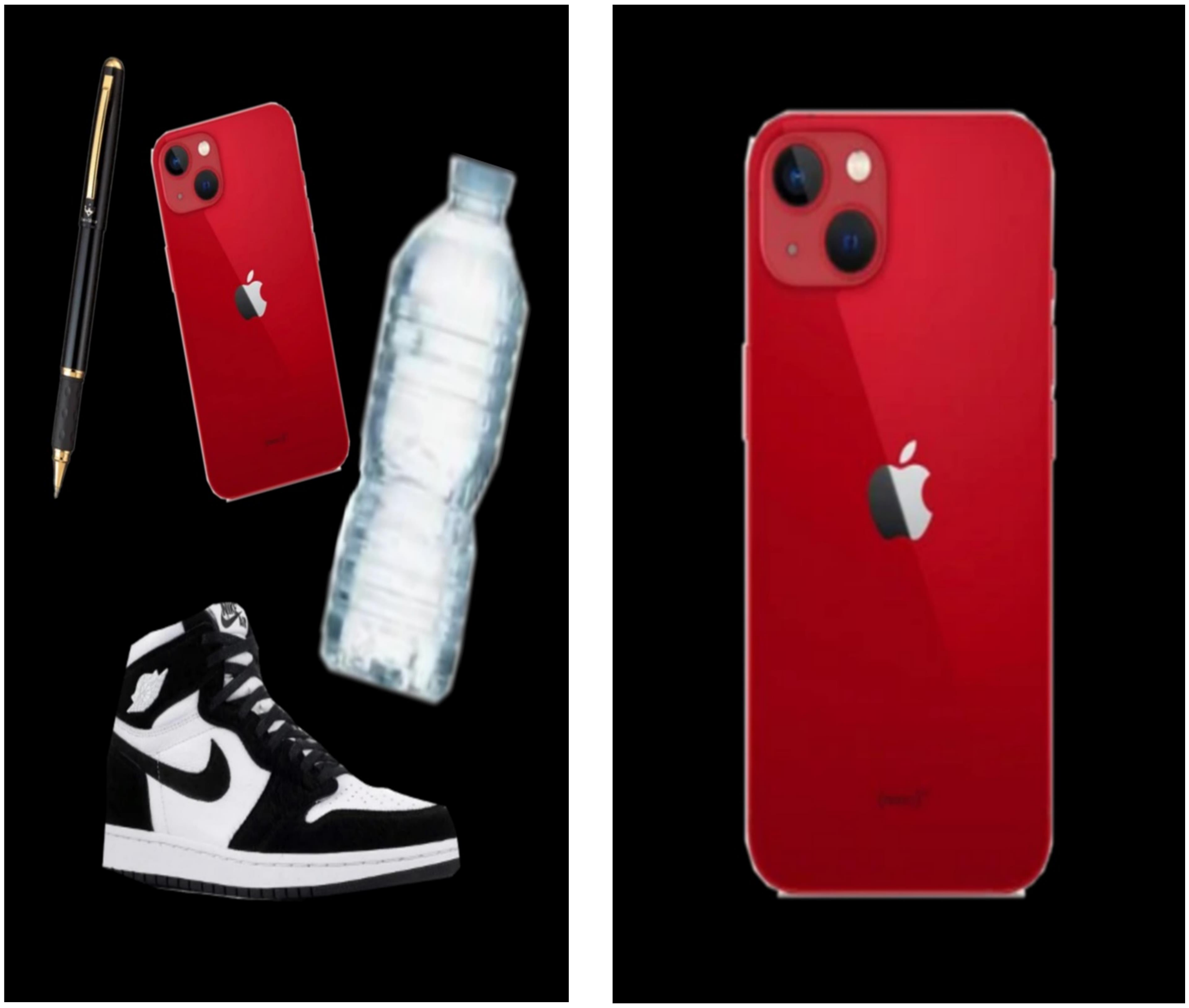

Figure 1 : Dataset Generated By Team Members.

This image displays a variety of everyday objects, including a pen, an iPhone, a water bottle, and a pair of shoes. Each object is outlined with a red bounding box, indicating detection or annotation. The iPhone, positioned prominently in the center, stands out with its vibrant red color, while the water bottle is placed towards the right side and the shoes at the bottom. The pen occupies the top left corner. This collection provides a diverse range of shapes and sizes, presenting a challenging dataset for object detection and recognition tasks. The clear delineation of each item makes it suitable for training and evaluating computer vision algorithms aimed at identifying objects in complex scenes.

Output of the Program :

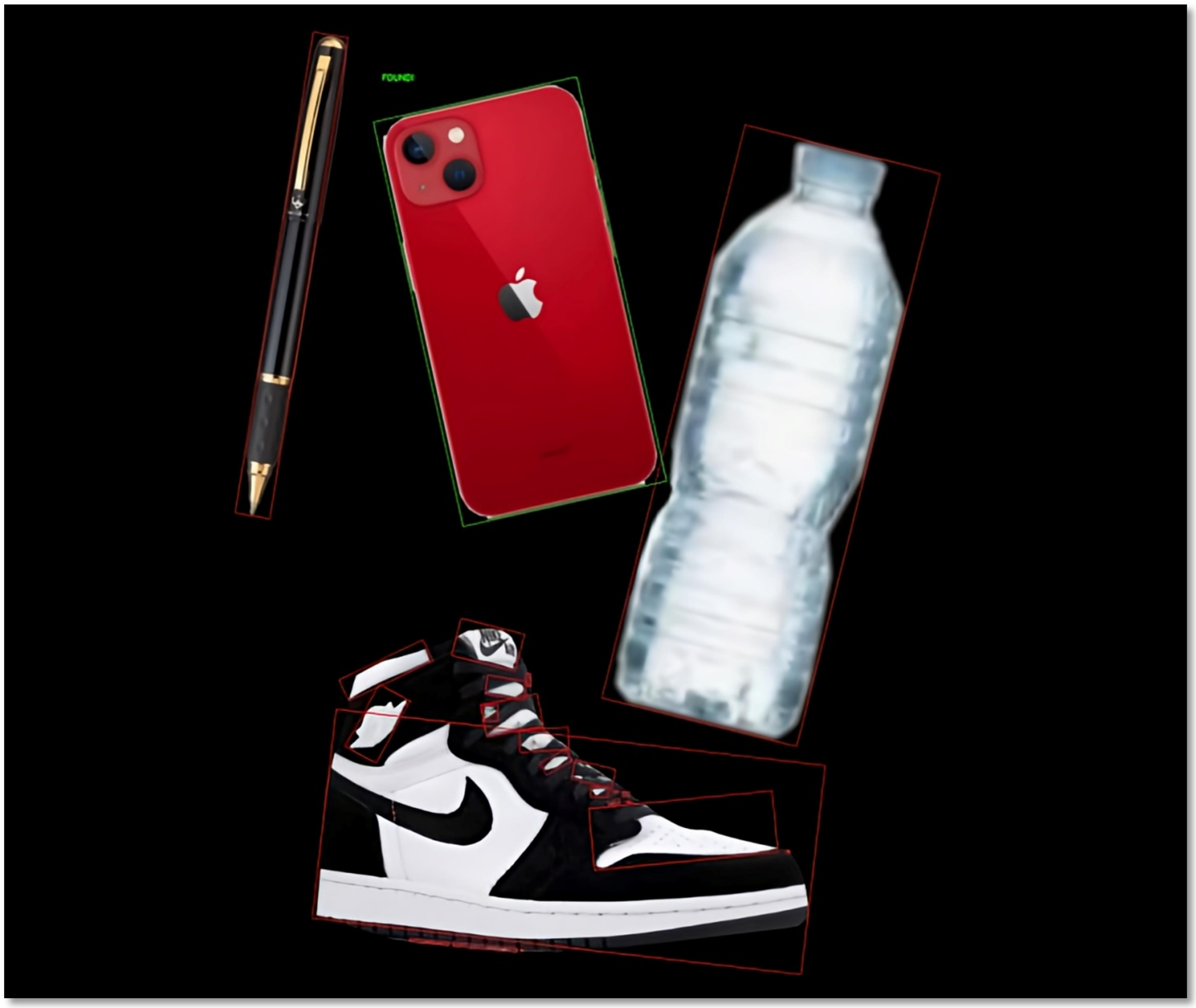

Figure 2 : From Bunch of Products The Input Product Was Found

In this intriguing composition, a diverse selection of everyday items is artfully arranged against a stark black backdrop. Amidst the assortment, a vivid red iPhone commands attention, its glossy surface radiating elegance and sophistication. Adjacent to the iPhone, a sleek pen exudes professionalism, while a transparent water bottle offers a refreshing contrast with its cool, translucent appearance. Completing the ensemble, a pair of stylish sneakers adds a touch of urban flair, showcasing intricate details and bold design elements. The use of Zernike moments for object detection has intricately outlined each item, underscoring the precision and effectiveness of this technique in accurately identifying shapes amidst a complex visual landscape.

Conclusion :

In conclusion, when describing multiple shapes in an image using Zernike Moments, it is important to follow a few key steps. First, extract the region of interest (ROI) for each object in the image. Then, apply Zernike Moments to each ROI to capture the shape characteristics.

When utilizing Zernike Moments, it is crucial to consider the radius and degree parameters. The radius determines the size of the disc onto which the shape is mapped. It is essential to choose a radius that is sufficiently large to include all relevant pixels of the shape. If any pixels fall outside the radius, they will be disregarded. Therefore, careful attention must be paid to ensure an appropriate radius is selected before extracting Zernike Moments.

Additionally, the degree parameter directly affects the dimensionality of the resulting feature vector. Higher degree values lead to larger and potentially more distinctive feature vectors. However, it is important to consider the computational cost as the number of degrees increases.

When setting the radius and degree parameters, it is recommended to prioritize the radius selection. Reflecting on the size of the objects in the dataset will help determine an adequate radius that captures their shapes effectively. Once the radius is established, the degree parameter can be fine-tuned accordingly. Starting with a value of degree=8 is often a reasonable initial choice, with adjustments made as needed based on the desired dimensionality of the feature vector.

By following these guidelines, one can effectively describe shapes in an image using Zernike Moments, taking into account the appropriate radius and degree parameters to capture the desired shape characteristics.