✅ circle\_classification.py

Import numpy as np

Import cv2

Import matplotlib.pyplot as plt

From sklearn.neighbors import KNeighborsClassifier

# Create Circle Image

imgSize = 100

circleImg = np.zeros((imgSize, imgSize), dtype=np.uint8)

center = (imgSize // 2, imgSize // 2)

radius = 30

cv2.circle(circleImg, center, radius, 1, thickness=-1)

# Define Feature Extraction Functions

Def calculate\_area(img):

Return np.sum(img)

Def calculate\_perimeter(img):

Return np.sum(cv2.Canny(img, 100, 200) > 0)

Def calculate\_circularity(area, perimeter):

Return (4 \* np.pi \* area) / (perimeter \*\* 2) if perimeter != 0 else 0

# Extract Features

Area = calculate\_area(circleImg)

Perimeter = calculate\_perimeter(circleImg)

Circularity = calculate\_circularity(area, perimeter)

# Train k-NN

trainingData = np.array([[area, perimeter, circularity]])

trainingLabels = np.array([1]) # 1 = Circle

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(trainingData, trainingLabels)

# Test and Predict

testFeatures = np.array([[area, perimeter, circularity]])

predictedLabel = knn.predict(testFeatures)

print(“The image contains a circle.”) if predictedLabel == 1 else print(“Unknown shape.”)

✅ square\_classification.py

Import numpy as np

Import cv2

Import matplotlib.pyplot as plt

From sklearn.neighbors import KNeighborsClassifier

# Create Square Image

imgSize = 100

squareImg = np.zeros((imgSize, imgSize), dtype=np.uint8)

squareImg[30:70, 30:70] = 1

# Define Feature Extraction Functions

Def calculate\_area(img):

Return np.sum(img)

Def calculate\_perimeter(img):

Return np.sum(cv2.Canny(img, 100, 200) > 0)

Def calculate\_circularity(area, perimeter):

Return (4 \* np.pi \* area) / (perimeter \*\* 2) if perimeter != 0 else 0

# Extract Features

Area = calculate\_area(squareImg)

Perimeter = calculate\_perimeter(squareImg)

Circularity = calculate\_circularity(area, perimeter)

# Train k-NN

trainingData = np.array([[area, perimeter, circularity]])

trainingLabels = np.array([2]) # 2 = Square

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(trainingData, trainingLabels)

# Test and Predict

testFeatures = np.array([[area, perimeter, circularity]])

predictedLabel = knn.predict(testFeatures)

print(“The image contains a square.”) if predictedLabel == 2 else print(“Unknown shape.”)

✅ triangle\_classification.py

Import numpy as np

Import cv2

Import matplotlib.pyplot as plt

From sklearn.neighbors import KNeighborsClassifier

# Create Triangle Image

imgSize = 100

triangleImg = np.zeros((imgSize, imgSize), dtype=np.uint8)

vertices = np.array([[50, 30], [30, 70], [70, 70]], dtype=np.int32)

cv2.fillPoly(triangleImg, [vertices], 1)

# Define Feature Extraction Functions

Def calculate\_area(img):

Return np.sum(img)

Def calculate\_perimeter(img):

Return np.sum(cv2.Canny(img, 100, 200) > 0)

Def calculate\_circularity(area, perimeter):

Return (4 \* np.pi \* area) / (perimeter \*\* 2) if perimeter != 0 else 0

# Extract Features

Area = calculate\_area(triangleImg)

Perimeter = calculate\_perimeter(triangleImg)

Circularity = calculate\_circularity(area, perimeter)

# Train k-NN

trainingData = np.array([[area, perimeter, circularity]])

trainingLabels = np.array([3]) # 3 = Triangle

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(trainingData, trainingLabels)

# Test and Predict

testFeatures = np.array([[area, perimeter, circularity]])

predictedLabel = knn.predict(testFeatures)

print(“The image contains a triangle.”) if predictedLabel == 3 else print(“Unknown shape.”)