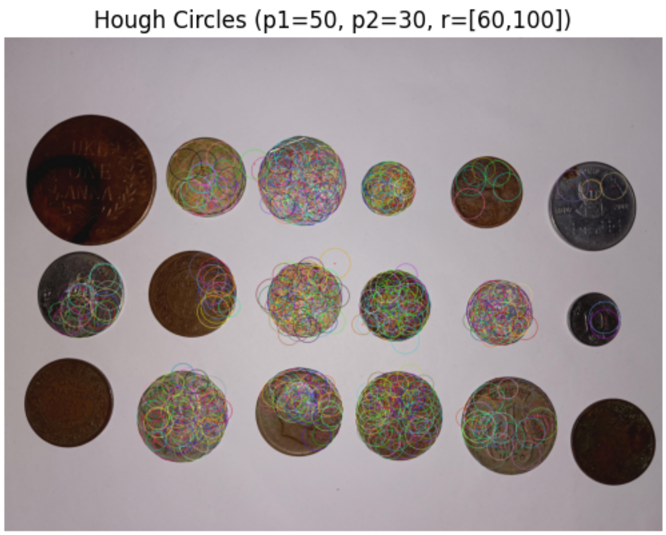
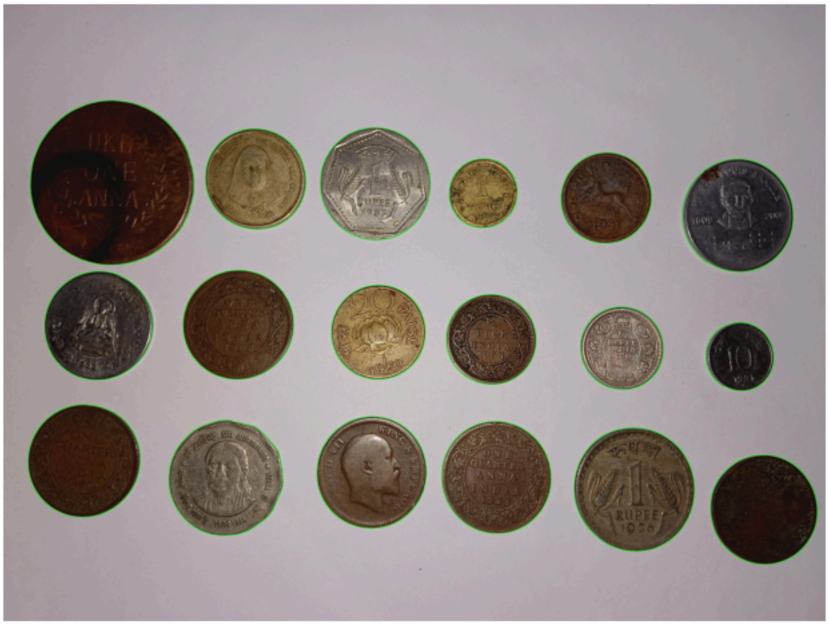
```
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt

def remove_overlapping_circles(circles, distance_threshold=15):
    """
    Removes overlapping circles by ensuring that the distance between
    any two circle centers is greater than `distance_threshold`.
    If two circles overlap, we keep the one with the larger radius.
    """
    if len(circles) == 0:
        return circles

    final_circles = []
    for c in circles:
        x1, y1, r1 = c
        keep = True
        for fc in final_circles:
            x2, y2, r2 = fc
            dist = np.sqrt((x1 - x2)**2 + (y1 - y2)**2)
            if dist < distance_threshold:
                # They overlap; keep the circle with the larger radius
                if r1 > r2:
                    # Replace the old circle
                    final_circles.remove(fc)
                    final_circles.append(c)
                keep = False
                break
        if keep:
            final_circles.append(c)

    return final_circles

def detect_coins(image_path, param1, param2, minRadius, maxRadius,
                 distance_threshold=15, show=False):
    """
    Detects coins using HoughCircles with given hyperparameters,
    removes overlapping circles, and optionally displays the result.
    Returns the number of coins detected.
    """
    # Read image
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Could not read image from:", image_path)
        return 0

    output = image.copy()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Blur to reduce noise
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)

    # Hough Circle detection
    circles = cv2.HoughCircles(
        blurred,
        cv2.HOUGH_GRADIENT,
        dp=1.2,
        minDist=30,      # You can adjust this based on coin spacing
        param1=param1,   # Canny high threshold
        param2=param2,   # Accumulator threshold for center detection
        minRadius=minRadius,
        maxRadius=maxRadius
    )

    # If no circles found, return 0
    if circles is None or len(circles) == 0:
        if show:
            print("No circles detected with these parameters.")
        return 0

    # Convert circles to integer format
    circles = np.round(circles[0, :]).astype("int")

    # Remove overlapping circles
    circles = remove_overlapping_circles(circles, distance_threshold=distance_threshold)
    # Draw circles in random colors if show=True
    if show:
        for (x, y, r) in circles:
            color = (
                random.randint(0, 255),
                random.randint(0, 255),
                random.randint(0, 255)
            )
            cv2.circle(output, (x, y), r, color, 2)  # Draw circle boundary
            cv2.circle(output, (x, y), 2, (0, 0, 255), 3)  # Draw center point

        # Convert the output image from BGR to RGB for correct matplotlib display.
        output_rgb = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)
        plt.imshow(output_rgb)
        plt.title(f"Hough Circles (p1={param1}, p2={param2}, r=[{minRadius},{maxRadius}])")
        plt.axis('off')
        plt.show()

    return len(circles)


if __name__ == "__main__":
    image_path = "1.jpg"  # Replace with your image path
    num_coins = detect_coins(image_path, param1=50, param2=30, minRadius=60, maxRadius=100, distance_threshold=18, show=True)
    print("Number of coins detected:", num_coins)
```



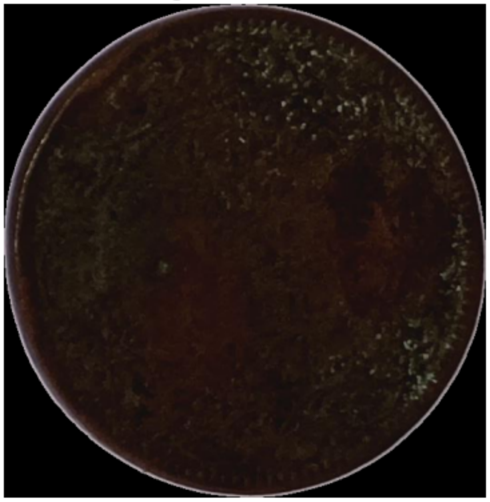
```
Number of coins detected: 629
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def detect_and_count_coins(image_path):
    # Load the image
    image = cv2.imread(image_path)
    if image is None:
        print("Error: Image not found.")
        return
    original = image.copy()

    # Preprocessing: Convert to grayscale and blur
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (9, 9), 2)

    # Edge detection using Canny, then dilate edges
    edges = cv2.Canny(gray, 50, 150)
    edges = cv2.dilate(edges, None, iterations=2)

    # Find contours from the edges
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Filter contours by circularity and area to find coin-like shapes
    min_area = 500  # Adjust based on your image
    coins = []
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area < min_area:
            continue
        perimeter = cv2.arcLength(cnt, True)
        if perimeter == 0:
            continue
        # Calculate circularity (ideal circle ~1)
        circularity = (4 * np.pi * area) / (perimeter ** 2)
        if circularity > 0.7:
            coins.append(cnt)

    # Part a: Visualize detected coins by drawing contours on the original image
    detected_image = original.copy()
    cv2.drawContours(detected_image, coins, -1, (0, 255, 0), 2)

    plt.figure(figsize=(8, 6))
    plt.imshow(cv2.cvtColor(detected_image, cv2.COLOR_BGR2RGB))
    plt.title('Detected Coins')
    plt.axis('off')
    plt.show()

    # Part b: Segment each coin and display them individually
    for idx, cnt in enumerate(coins):
        mask = np.zeros_like(gray)
        cv2.drawContours(mask, [cnt], -1, 255, -1)
        segmented = cv2.bitwise_and(original, original, mask=mask)
        x, y, w, h = cv2.boundingRect(cnt)
        cropped = segmented[y:y+h, x:x+w]

        plt.figure()
        plt.imshow(cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB))
        plt.title(f'Segmented Coin {idx + 1}')
        plt.axis('off')
        plt.show()

    cv2.destroyAllWindows()

    # Part c: Count coins and print the total count
    count = len(coins)
    print(f'Total number of coins: {count}')
    return count

# Example usage
detect_and_count_coins('1.jpg')
```

Detected Coins

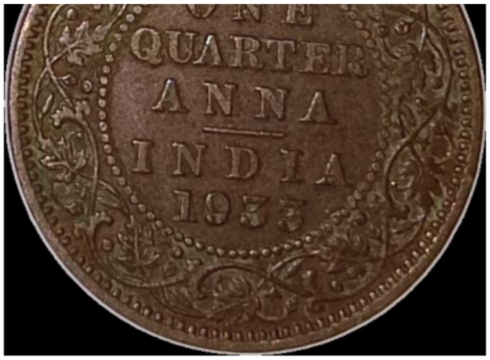


Segmented Coin 1

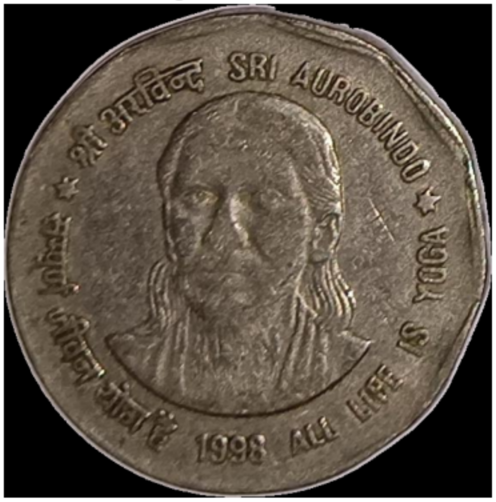


Segmented Coin 2

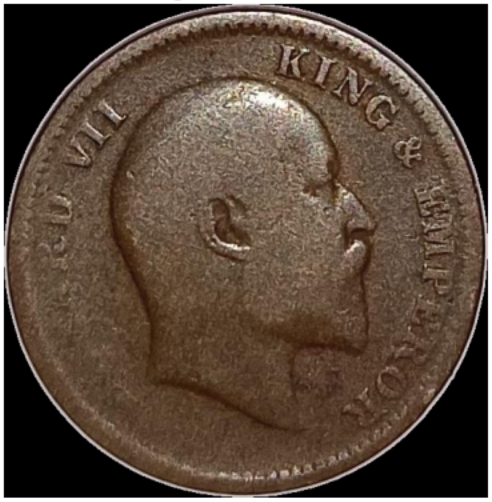
Segmented Coin 3



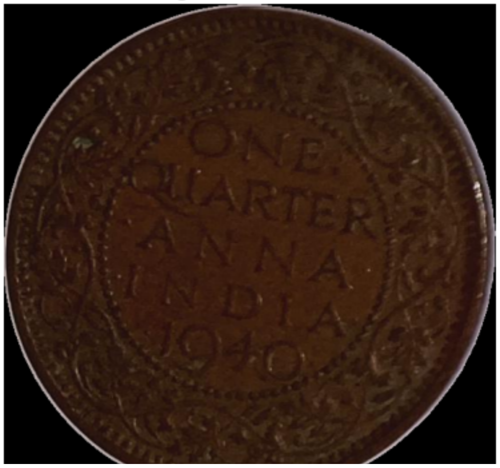
Segmented Coin 4



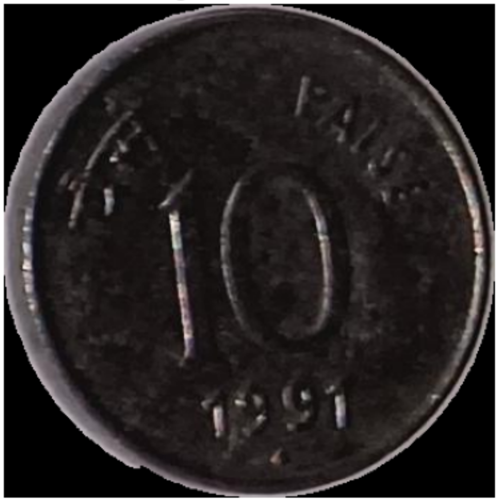
Segmented Coin 5



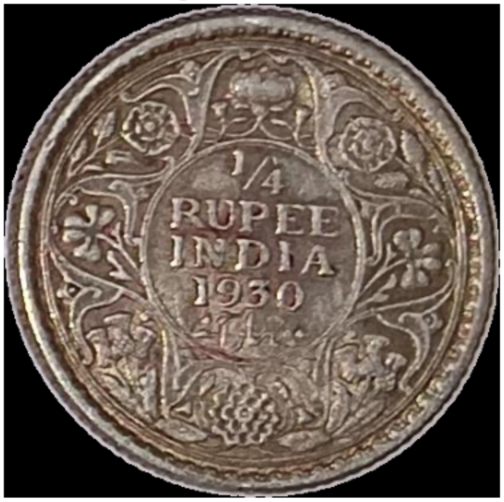
Segmented Coin 6



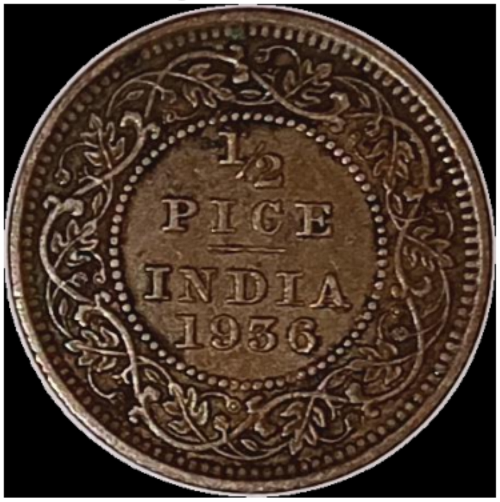
Segmented Coin 7



Segmented Coin 8



Segmented Coin 9



Segmented Coin 10

Segmented Coin 11

Segmented Coin 12

Segmented Coin 13

Segmented Coin 14