TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

VIỆN TRÍ TUỆ NHÂN TẠO

-----***-----

BÁO CÁO BÀI TẬP CHUẨN BỊ THI GIỮA KÌ

ĐỀ TÀI

Bunny Counting

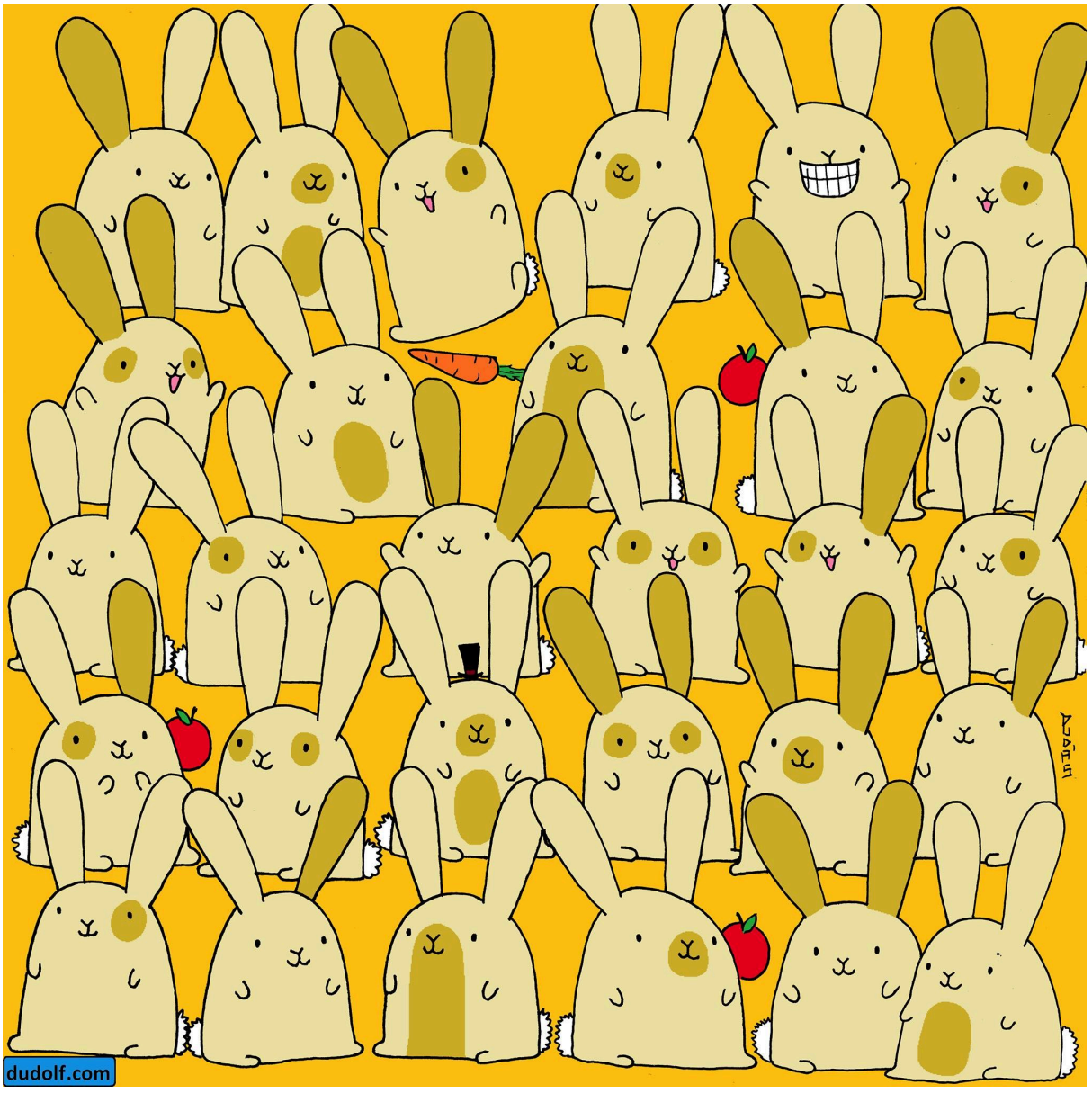
Học phần: Xử lý và phân tích hình ảnh (2425II_AIT3002#_1)
Giảng Viên Hướng Dẫn: TS. Trần Quốc Long
Ngành: Trí tuệ nhân tạo - QH-2022-I/CQ-A-AI
Tên Sinh Viên: Nguyễn Bảo Sơn
Mã Sinh Viên: 22022613

1. Thư viện được sử dụng

- OpenCV(CV2)
- NumPy(numpy)
- Patplotlib(matplotlib.pyplot)

2. Xử lý hình ảnh

a. Hình ảnh được sử dụng:



b. Tách vùng có thỏ:

- Đọc ảnh và chuyển đổi màu: OpenCV đọc ảnh theo chuẩn BGR nên cần chuyển sang RGB để hiển thị đúng màu

- Chuyển đổi ảnh sang HSV để dễ nhận diện màu: HSV giúp tách biệt màu sắc hiệu quả hơn so với RGB

```
# Convert to RGB (OpenCV loads as BGR)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Convert to HSV color space
img_hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
```

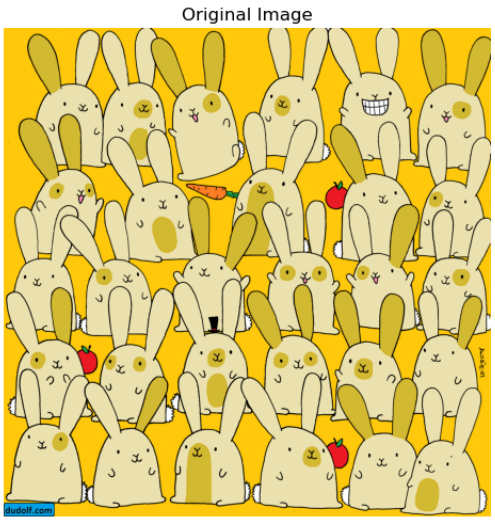
- Xác định phạm vi màu của thỏ (màu be/kem):

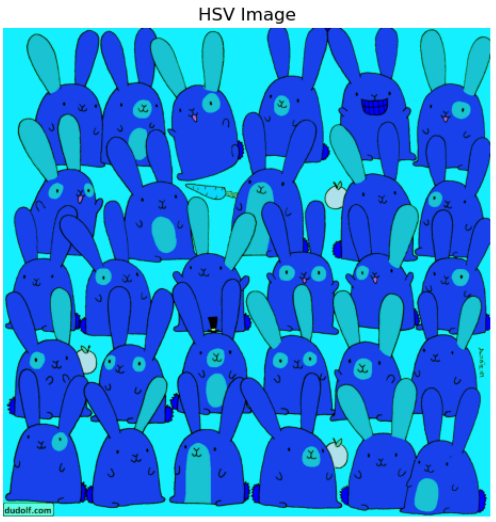
```
# You may need to adjust these values based on the specific image
lower_beige = np.array([15, 30, 150])
upper_beige = np.array([100, 200, 240])
```

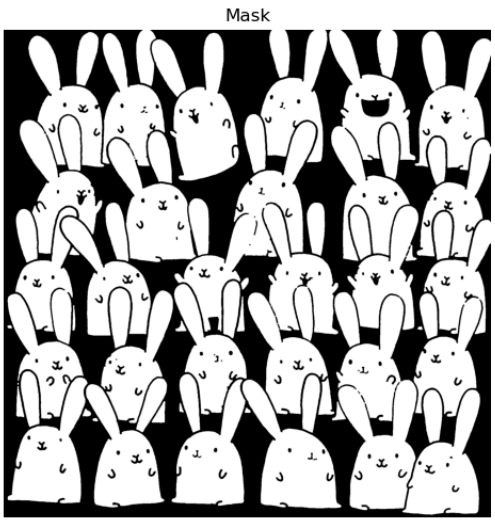
- Tạo mặt nạ (mask) để lọc màu: tạo một ảnh nhị phân, trong đó các pixel có màu nằm trong khoảng màu đã chọn sẽ thành màu trắng (255), các pixel khác thành đen (0)
- Xử lý ảnh để làm sạch mask
- Tách vùng có thỏ ra khỏi ảnh: Giữ lại các pixel thuộc vùng thỏ bằng phép toán bitwise_and() với mặt nạ (mask).

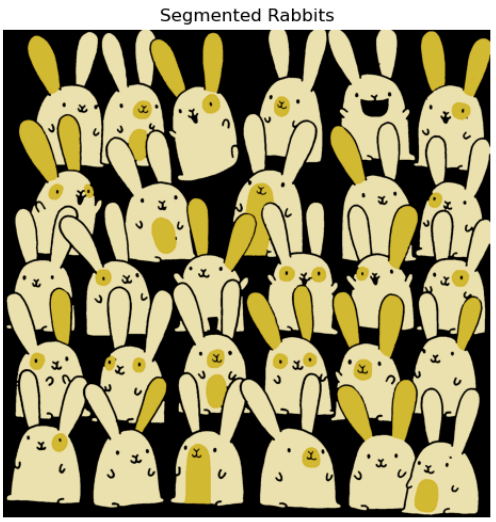
```
# Create a mask for beige/cream colors
mask = cv2.inRange(img_hsv, lower_beige, upper_beige)

# Apply morphological operations to clean up the mask
kernel = np.ones((5, 5), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

# Apply the mask to get only the rabbits
rabbits_only = cv2.bitwise_and(img_rgb, img_rgb, mask=mask)
```

- Hiển thị kết quả:
  - Ảnh gốc
  - Ảnh HSV
  - Mask- vùng màu trắng là thỏ được nhận diện
  - Ảnh có thỏ đã được tách





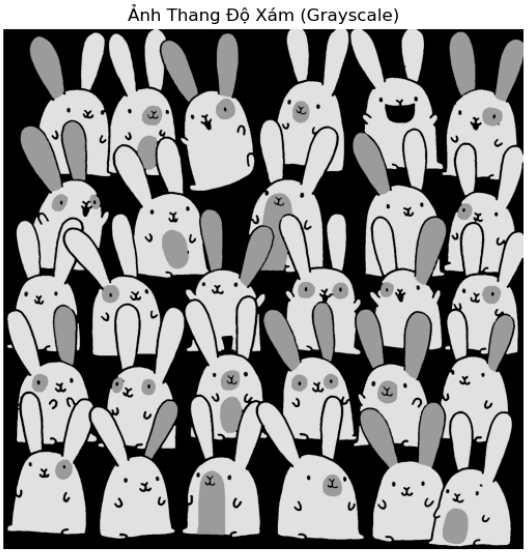


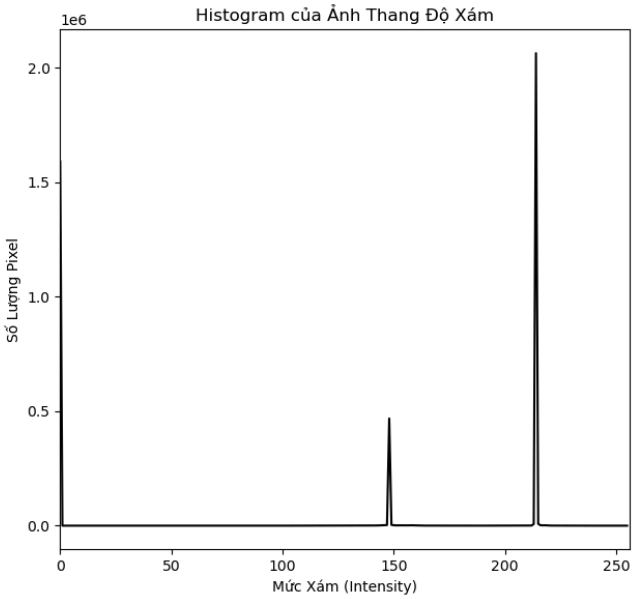


c. Chuyển đổi ảnh thỏ đã được tách ra sang ảnh thang độ xám
   - Chuyển ảnh sang thang độ xám: Chuyển ảnh màu rabbits_only (đang ở định dạng RGB/BGR) sang ảnh xám (grayscale).
   - Tính histogram: cv2.calcHist dùng để tính histogram, cho biết phân bố cường độ sáng của ảnh.

```
gray_image = cv2.cvtColor(rabbits_only, cv2.COLOR_BGR2GRAY)

histogram = cv2.calcHist([gray_image], [0], None, [256], [0, 256])
```

   - Hiển thị ảnh thang độ xám và vẽ histogram



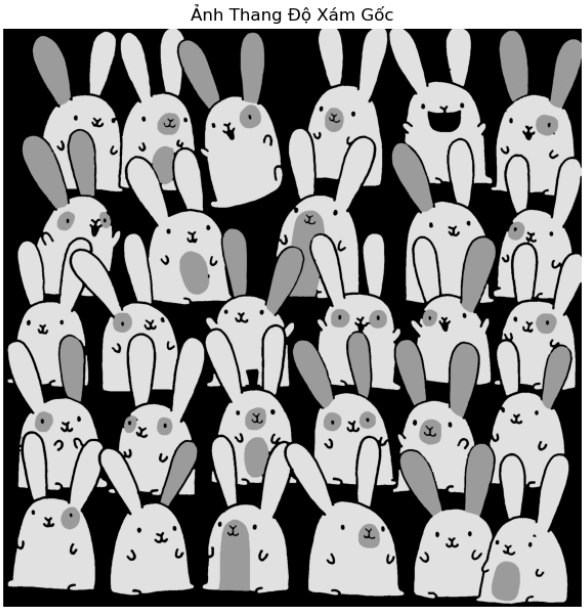


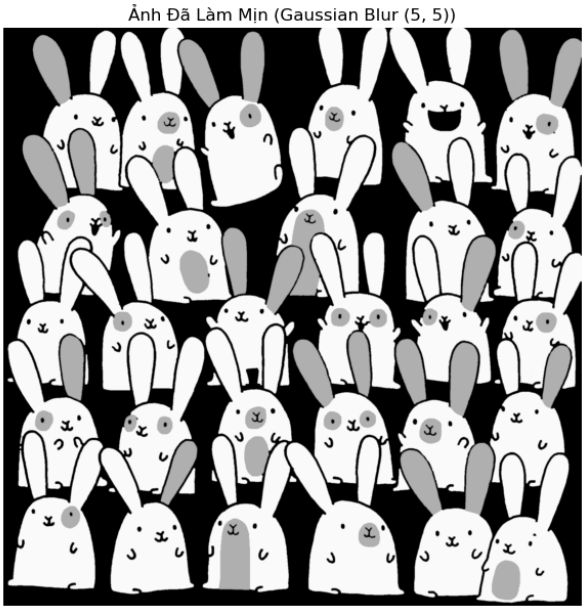
d. Làm mịn ảnh bằng Gaussian Blur.
Giúp giảm nhiễu và làm mượt đường biên trong ảnh.

- **Gaussian Blur**: là ma trận 5x5 dùng để làm mịn ảnh. Kernel càng lớn -> ảnh càng bị làm mờ mạnh.

```
GAUSSIAN_KERNEL_SIZE = (5, 5)

blurred_gray_image = cv2.GaussianBlur(gray_image, GAUSSIAN_KERNEL_SIZE, 0)
```

- Hiển thị ảnh trước và sau khi làm mờ:



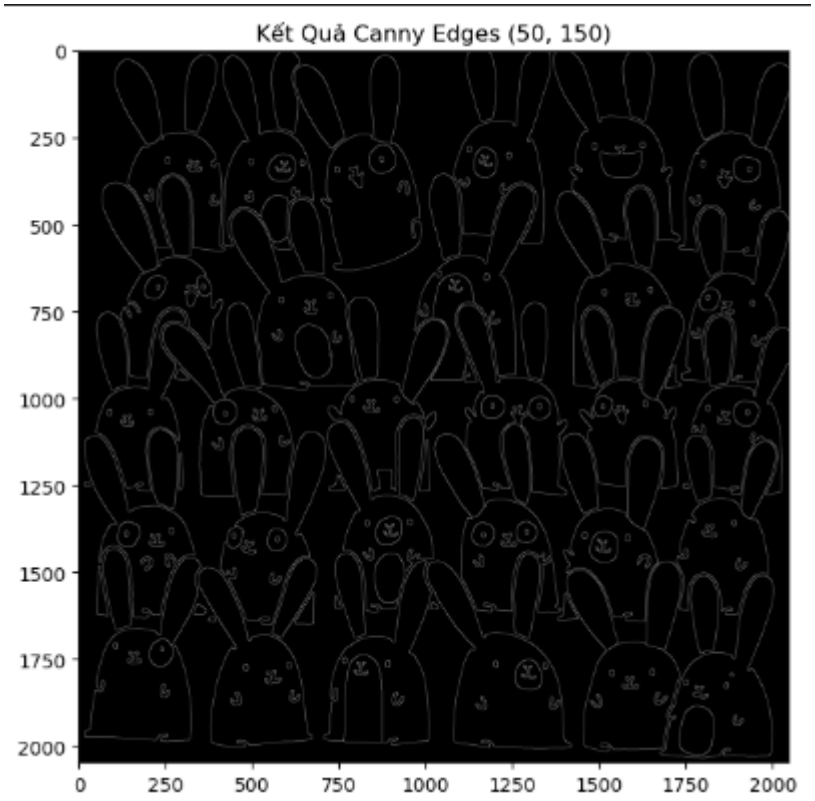


e. Phát hiện cạnh bằng phương pháp Canny Edge Detection và Dialtion để làm nổi bật biên cạnh.

- Dữ liệu đầu vào là ảnh đã làm mờ blurred_gray_image
- Canny Edge Detection: sử dụng 2 ngưỡng
  - CANNY_THRESHOLD_1 = 50: Ngưỡng dưới – các pixel có giá trị nhỏ hơn sẽ bị loại bỏ.
  - CANNY_THRESHOLD_2 = 150: Ngưỡng trên – các pixel có giá trị cao hơn sẽ được giữ lại.
  -
- Phát hiện cạnh bằng thuật toán Canny: cv2.Canny()

```
CANNY_THRESHOLD_1 = 50
CANNY_THRESHOLD_2 = 150

# Using Canny Edge Detection
canny_edges = cv2.Canny(blurred_gray_image, CANNY_THRESHOLD_1, CANNY_THRESHOLD_2)
```

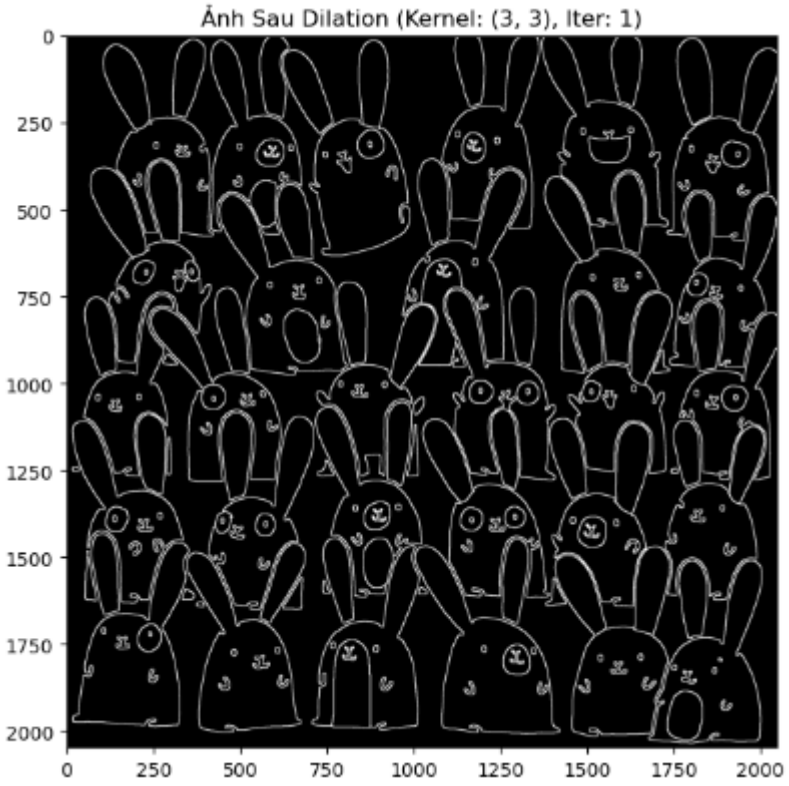
- **Kết quả:** Ảnh đầu ra là ảnh nhị phân (đen/trắng), trong đó các cạnh có màu trắng (255), phần còn lại màu đen (0).



- Giãn ảnh (Dilation) để làm nổi bật biên cạnh: Dilation giúp làm dày các đường biên cạnh, giúp dễ nhìn hơn.

```python
# Perform dilation to highlight the border
DILATION_KERNEL_SIZE = (3, 3)
DILATION_ITERATIONS = 1

kernel = np.ones(DILATION_KERNEL_SIZE, np.uint8)
dilated_edges = cv2.dilate(canny_edges, kernel, iterations=DILATION_ITERATIONS)
```

- **Kết quả:** Biên cạnh trở nên dày hơn, có thể dùng để nhận diện vật thể dễ dàng hơn.



=> Từ 2 kết quả ta có thể nhận thấy rằng dilation làm đường biên cạnh nổi bật hơn nhiều so với đường biên cạnh của Canny Edge Detection.

3. Phát hiện số lượng thỏ trong ảnh:

a. Phân tích contour:

- Tiền xử lý ảnh
  - **cv2.threshold()**: Áp dụng ngưỡng nhị phân để biến ảnh xám thành ảnh trắng-đen (0 hoặc 255).

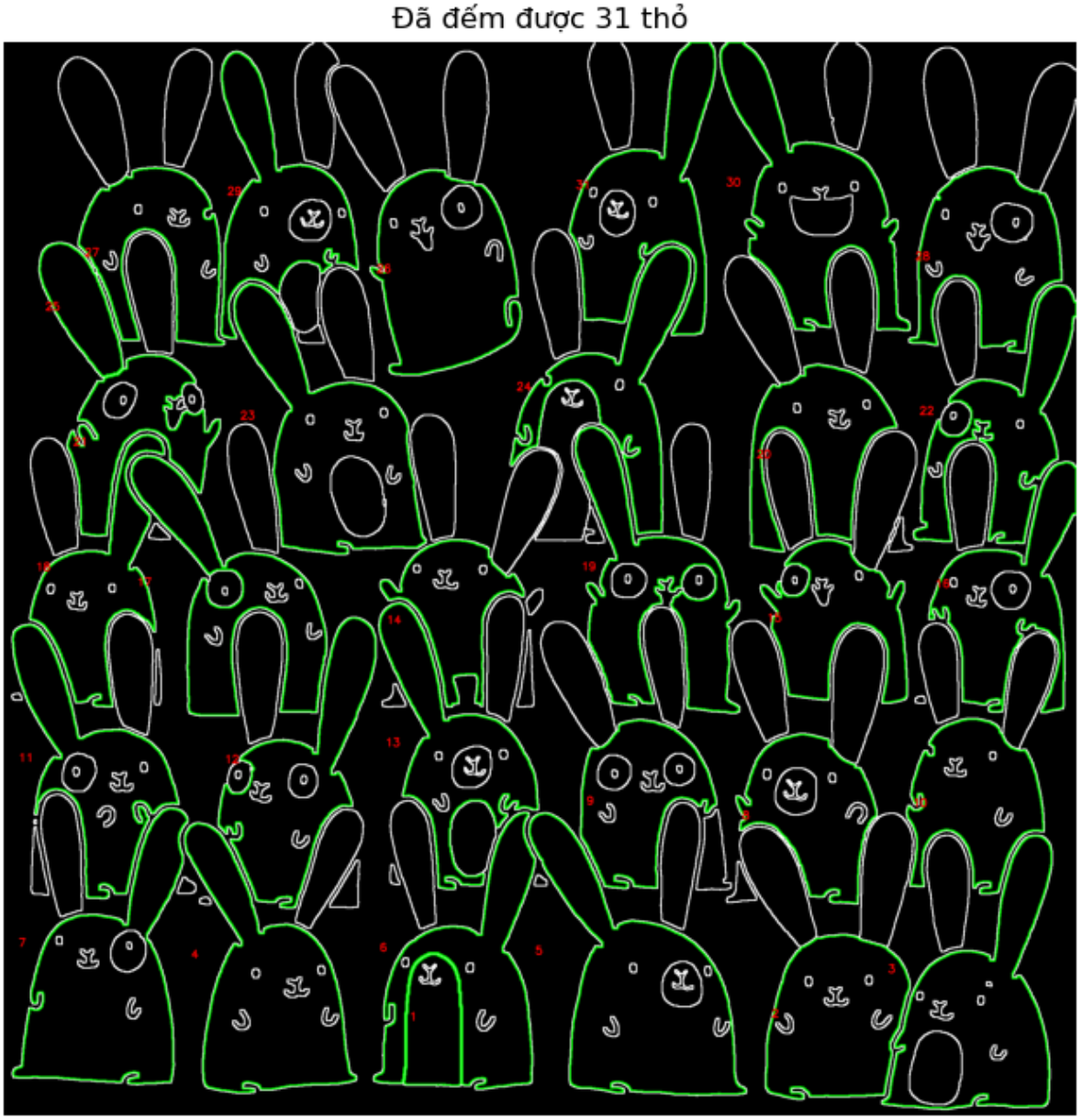
    - **cv2.bitwise_not()**: Đảo ngược ảnh nhị phân → giúp phát hiện contour chính xác hơn.
- Tìm Contour trong ảnh
    - **cv2.findContours()**: Tìm các **contour** (đường viền của các vật thể) trong ảnh.
    - **cv2.RETR_CCOMP**: Trả về cả **contour ngoài** và **contour trong** (có cấu trúc phân cấp).
    - **cv2.CHAIN_APPROX_SIMPLE**: Giảm số điểm lưu trữ của contour.
- Tạo ảnh để vẽ kết quả
    - Chuyển ảnh từ **grayscale** sang **BGR (màu)** để dễ vẽ lên ảnh.
- Duyệt qua các contour và lọc các đối tượng có hình dạng giống con thỏ

```
# Calculate the area and circumference of Contour
area = cv2.contourArea(contour)
perimeter = cv2.arcLength(contour, True)

# Calculate the frame ratio
x, y, w, h = cv2.boundingRect(contour)
aspect_ratio = h / w if w > 0 else 0
```

- Phân tích hình dạng để xác định thỏ
- Kiểm tra Convexity Defects để lọc thỏ chính xác hơn

```
if 20000 < area < 120000 and 0.8 < aspect_ratio < 3.0:

    # Convexity defects check
    hull = cv2.convexHull(contour, returnPoints=False)
    if hull is not None and len(hull) > 3:
        defects = cv2.convexityDefects(contour, hull)
        significant_defects = sum(1 for defect in defects if defect[0][3] > 500) if defects is not None else 0

        if significant_defects <= 9:
            cv2.drawContours(output_image, [contour], -1, (0, 255, 0), 2)
            cv2.putText(output_image, str(rabbit_count+1), (x+10, y+h//2),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            rabbit_count += 1
```

- Vẽ kết quả lên ảnh và hiển thị kết quả:
    - Vẽ đường viền xanh lá xung quanh thỏ.
    - Đánh số thứ tự của thỏ trên ảnh.
    - Hiển thị ảnh từ BGR sang RGB và số lượng thỏ đếm được



- Kết quả đưa ra chưa chính xác bởi vì có những đường viền ở giữa chú thỏ vẫn được tính (thỏ thứ 6) nên cần thay đổi bước xử lý ảnh ban đầu để có thể đưa ra kết quả chính xác hơn.

b. Template Matching:

Thực hiện phát hiện số lượng thỏ trong ảnh bằng cách sử dụng phương pháp Template Matching dựa trên đặc điểm đôi tai của thỏ.

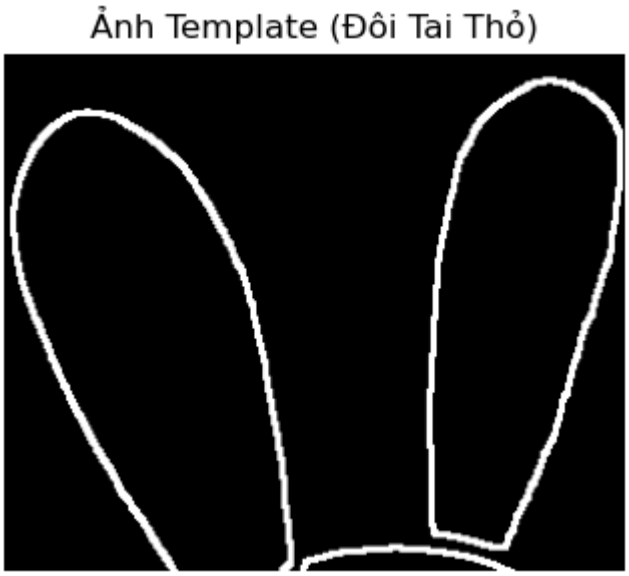
- Tạo ảnh nhị phân từ ảnh đã giãn (dilated edges): **cv2.threshold()**: Chuyển ảnh **dilated_edges** sang ảnh nhị phân.

```
_, thresh = cv2.threshold(dilated_edges, 128, 255, cv2.THRESH_BINARY)
```

- Cắt một phần ảnh làm mẫu (template):
  - Chọn **một vùng từ ảnh nhị phân thresh** làm mẫu (template).
  - Vùng này đại diện cho **đôi tai thỏ**, giả sử đôi tai là đặc điểm dễ nhận diện nhất.

```
template = thresh[0:250, 100:400]

plt.figure(figsize=(4, 4))
plt.imshow(template, cmap="gray")
plt.title("Ảnh Template (Đôi Tai Thỏ)")
plt.axis("off")
plt.show()
```

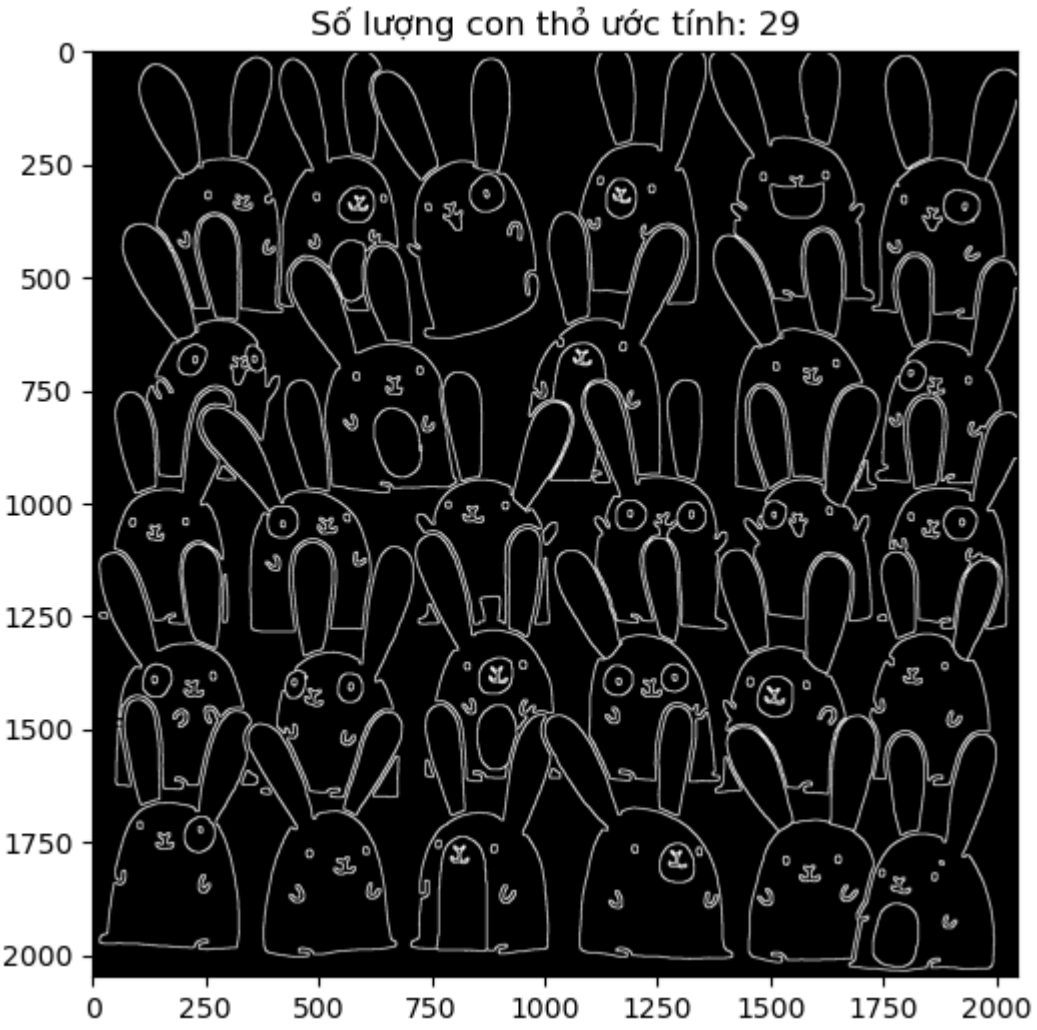
- Kết quả nhận được:



- So khớp mẫu với toàn bộ ảnh bằng Template Matching:
  - **cv2.matchTemplate()**: So khớp mẫu template với toàn bộ ảnh thresh.
  - **cv2.TM_CCOEFF_NORMED**: Phương pháp so khớp **tương quan hệ số chuẩn hóa** (Normalized Cross-Correlation).

- Lọc ra các vị trí khớp mẫu theo ngưỡng:
    - **Ngưỡng 0.333**: Chỉ giữ lại các điểm có **mức độ tương đồng ≥ 0.333**.
    - **np.where()**: Tìm các vị trí trong result thỏa mãn điều kiện.
    - **zip(*locations[::-1])**: Chuyển đổi tọa độ từ (hàng, cột) sang (x, y)

- Ước tính số lượng thỏ:
    - **Số lượng thỏ ≈ số lần tìm thấy đôi tai**.
    - Nếu mỗi con thỏ có **một đôi tai được nhận diện**, ta có thể suy ra số lượng thỏ.
- Hiển thị kết quả với các khung nhận diện:
    - **Vẽ hình chữ nhật** tại các vị trí khớp với template.
    - **Hiển thị ảnh với số lượng thỏ được phát hiện**.



=> Kết quả cuối cùng là số lượng thỏ được đếm. Kết quả này phụ thuộc rất nhiều vào threshold=0,333 nếu tăng threshold thì kết quả sẽ thấp hơn và ngược lại

Source code: https://github.com/nguyenbaoson/Bunny_Counting.git