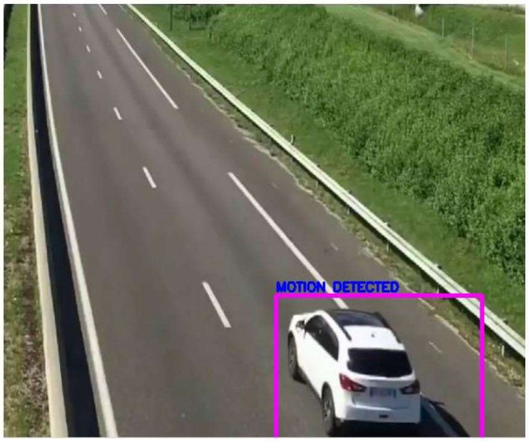
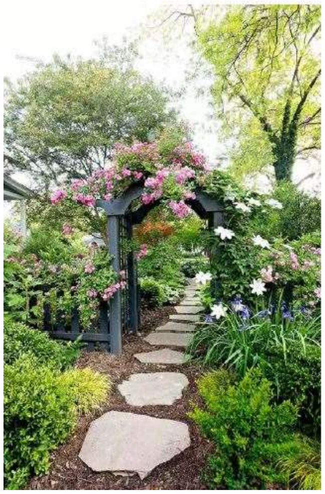
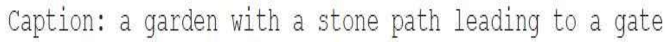
```
import cv2
from google.colab.patches import cv2_imshow
cap = cv2.VideoCapture('Video.mp4')
if not cap.isOpened():
    print("Error: Could not open video.")
    exit()
_, frame1 = cap.read()
prev_frame_gray = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
while True:
    _, frame2 = cap.read()
    if not _:
        break
    curr_frame_gray = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    frame_diff = cv2.absdiff(prev_frame_gray, curr_frame_gray)
    _, thresh = cv2.threshold(frame_diff, 30, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
                               cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        if cv2.contourArea(contour) < 1000:
            continue
        x, y, w, h = cv2.boundingRect(contour)
        cv2.rectangle(frame2, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2_imshow( frame2)
    prev_frame_gray = curr_frame_gray.copy()
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```



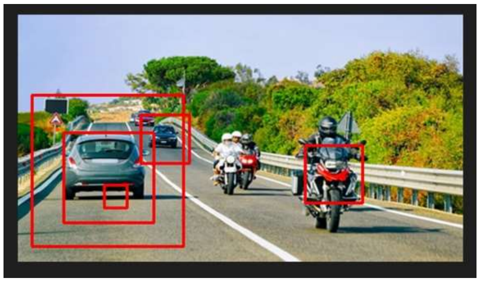
```
!pip install transformers
!pip install torch
!pip install pillow

from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import requests

model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")

url = "https://example.com/your-image.jpg" # Replace with the URL of your image
image = Image.open(requests.get(url, stream=True).raw)

inputs = processor(images=image, return_tensors="pt")

outputs = model.generate(**inputs)
caption = processor.decode(outputs[0], skip_special_tokens=True)
print("Caption:", caption)
```





```
Caption: a garden with a stone path leading to a gate
```

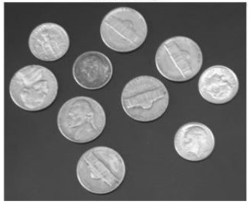
```
from PIL import Image

import cv2

import numpy as np

import requests

image_url = 'https://c.ndtvimg.com/202206/fdp0lr_car_625x300_17_June_22.jpg'

response = requests.get(image_url, stream=True)

image = Image.open(response.raw)

image = image.resize((450, 250))

image_arr = np.array(image)

grey = cv2.cvtColor(image_arr, cv2.COLOR_BGR2GRAY)

blur = cv2.GaussianBlur(grey, (5, 5), 0)

dilated = cv2.dilate(blur, np.ones((3, 3)))

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))

closing = cv2.morphologyEx(dilated, cv2.MORPH_CLOSE, kernel)

car_cascade_src = 'cars.xml'

car_cascade = cv2.CascadeClassifier(car_cascade_src)

cars = car_cascade.detectMultiScale(closing, 1.1, 1)

cnt = 0

for (x, y, w, h) in cars:

   cv2.rectangle(image_arr, (x, y), (x + w, y + h), (255, 0, 0), 2)

   cnt += 1

print(cnt, " cars found")

annotated_image = Image.fromarray(image_arr) annotated_image.show()

cv2.waitKey(0)

cv2.destroyAllWindows()
```



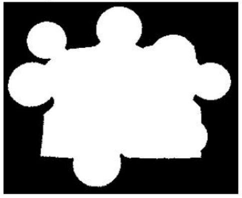
```
5  cars found
```

```
I = imread('coins.png');
imshow(I)
title('Original Image')
mask = zeros(size(I));
mask(25:end-25,25:end-25) = 1;
imshow(mask)
title('Initial Contour Location')
bw = activecontour(I,mask);
imshow(bw)
title('Segmented Image, 100 Iterations')
bw = activecontour(I,mask,300);
imshow(bw)
title('Segmented Image, 300 Iterations')
```

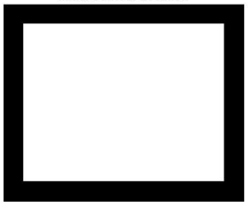
Original Image



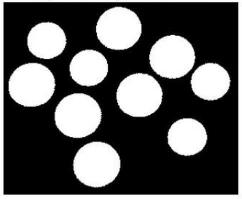
Segmented Image, 100 Iterations



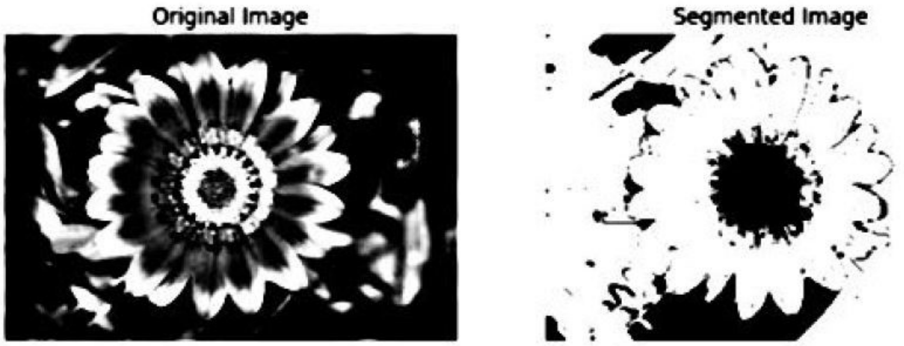
Initial Contour Location



Segmented Image, 300 Iterations



```
import cv2

import numpy as np

import matplotlib.pyplot as plt

class RegionGrow:

    def _init_(self, image_path, seed_point, threshold):

        self.img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

        self.seed = seed_point

        self.threshold = threshold

        self.segmented_img = np.zeros_like(self.img)

    def grow(self):

        rows, cols = self.img.shape

        to_process = [self.seed]

        self.segmented_img[self.seed] = 255

        while to_process:

            x, y = to_process.pop(0)

            for dx in [-1, 0, 1]:

                for dy in [-1, 0, 1]:

                    nx, ny = x + dx, y + dy

                    if 0 <= nx < rows and 0 <= ny <cols and self.segmented_img[nx, ny] == 0:

                        if abs(int(self.img[nx, ny]) - int(self.img(x, y)) ) <= self.threshold:

                            self.segmented_img[nx, ny] = 255

                            to_process.append((nx, ny))

    def save_results(self, original_img_path='original_image.png',segmented_img_path='segmented_image.png'):

                     plt.figure(figsize=(10,5))

                     plt.subplot(3, 2, 1)

                     plt.title('Original Image')

                     plt.imshow(self.img, cmap='gray')

                     plt.axis('off')

                     plt.subplot(2, 2, 2)

                     plt.title('Segmented Image')

                     plt.imshow(self.segmented_img, cmap='gray')

                     plt.axis('off')

                     plt.savefig('combined_image.png')

                     plt.close()

image_path = 'images.jpeg'

seed_point = (100, 100)

threshold = 10

rg = RegionGrow(image_path, seed_point, threshold)

rg.grow()

rg.save_results()
```



Original Image

Segmented Image