DATE : 27 june 2024

DAY : Thursday

TOPICS : OpenCV

OpenCV, short for Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. Originally developed by Intel, it is now maintained by a community of developers under the OpenCV Foundation.

The term Computer vision is used for a subject of performing the analysis of digital images and videos using a computer program

Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos.

let's simplify it. We'll focus on basic image preprocessing steps like converting an image to grayscale, applying Gaussian blur, and detecting edges using the Canny edge detector. Here's a simpler example:

Simple Example: Image Preprocessing

Step-by-Step Code:

Read an image

Convert the image to grayscale

Apply Gaussian blur

Detect edges using the Canny edge detector

Display the results

steps of image preprocessing and edge detection using OpenCV-Python.

```
import cv2
import matplotlib.pyplot as plt


# Step 1: Read the image from file
image = cv2.imread('/content/pexels-photo-1427430.jpeg')
```

Converts the image from BGR (Blue, Green, Red) color space to grayscale.

```
# Step 2: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Applies Gaussian blur to the grayscale image to reduce noise and detail. The (5, 5) tuple specifies the kernel size, and 0 is the standard deviation in the X and Y directions.

```
# Step 3: Apply Gaussian blur to reduce noise
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
```

Applies Gaussian blur to the grayscale image to reduce noise and detail. The (5, 5) tuple specifies the kernel size, and 0 is the standard deviation in the X and Y directions.

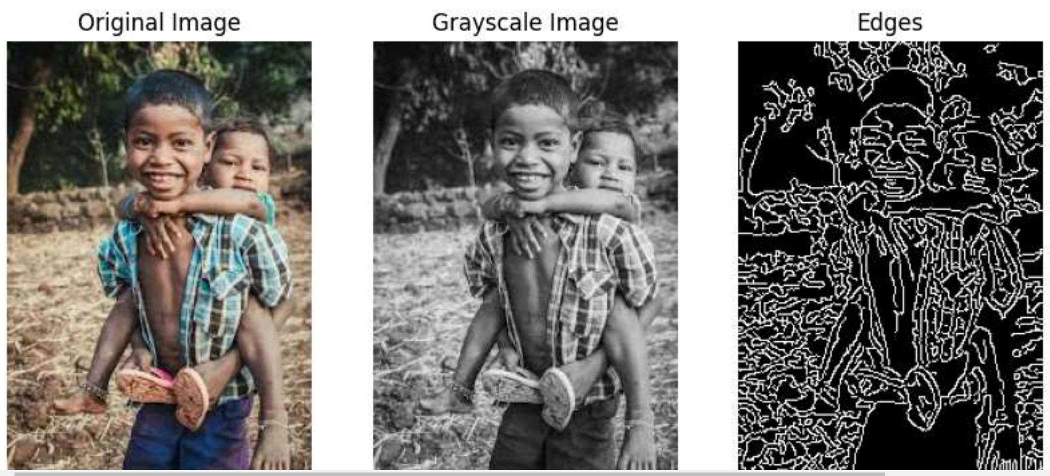
```
# Step 4: Detect edges using the Canny edge detector
edges = cv2.Canny(blurred_image, 50, 150)
```

Detects edges in the blurred image. The lower threshold is 50, and the upper threshold is 150. These thresholds determine how strong an edge needs to be before it is included in the output.

```
# Step 5: Display the results using matplotlib
plt.figure(figsize=(10, 5))
```

```
<Figure size 1000x500 with 0 Axes>
```

```
# Step 5: Display the results using matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Grayscale Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Edges')
plt.imshow(edges, cmap='gray')
plt.axis('off')

plt.show()
```



Explanation:

Reading the Image: The image is read from a file using cv2.imread.

Grayscale Conversion: The image is converted to grayscale using cv2.cvtColor to simplify the processing.

Gaussian Blur: Gaussian blur is applied using cv2. GaussianBlur to smooth the image and reduce noise.

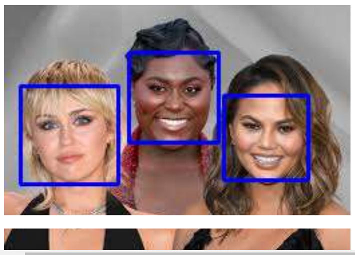
Edge Detection: Edges are detected using the Canny edge detector with cv2.Canny.

Displaying Results: The original image, grayscale image, and edge-detected image are displayed using matplotlib.

Object Detection with Haar Cascades

```
import cv2    # cv2 is the OpenCV library used for image processing.
from google.colab.patches import cv2_imshow  # cv2_imshow is a function from the google.colab.patches module that allows us to display image

# Load the pre-trained Haar Cascade for face detection
#This file (haarcascade_frontalface_default.xml) contains the data needed for detecting faces.
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Read the input image
```

```
img = cv2.imread('/content/Byr_ShortHairRoundFace_LeadRecirc-7e7b338fa0a2464b9c6751e4c7507155.webp')
#cv2.cvtColor converts the image from its original color space (BGR) to grayscale. Face detection works better on grayscale images because i
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# # Detect faces
# 1.1: The scale factor. Specifies how much the image size is reduced at each image scale.
# 4: The minimum number of neighbors. Specifies how many neighbors each candidate rectangle should have to retain it. This parameter affects
faces = face_cascade.detectMultiScale(gray, 1.1, 4)

# Draw rectangles around detected faces
# This loop iterates over the detected faces.
# For each face, x and y are the coordinates of the top-left corner, and w and h are the width and height of the rectangle.
# cv2.rectangle draws a rectangle around each detected face on the original image. The color of the rectangle is blue (specified by (255, 0,
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Display the output
# Display Image:
# cv2_imshow displays the image with the rectangles drawn around detected faces.
# cv2.waitKey(0) waits indefinitely for a key press. This is necessary to keep the window open until the user decides to close it.
# cv2.destroyAllWindows closes all the OpenCV windows. In the context of Colab, this line doesn't have an effect, but it's good practice to
cv2_imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Start coding or generate with AI.