

Research_paper work on Image processing

1. Loading and displaying an image

This program demonstrates loading an image file with OpenCV and displaying it with matplotlib. The image is read using the `cv2.imread()` function and then the color channels are converted from BGR format to RGB format using `cv2.cvtColor()`. The resulting image is displayed using `plt.imshow()`. This feature is useful for visualizing images in various computer vision and image processing tasks.

```
import cv2
import matplotlib.pyplot as plt
file_path = "carrots.png"
image = cv2.imread(file_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.axis('off')
plt.show()
```



2. Grayscale image

An image grayscale program converts a color image to grayscale. Grayscale is a common preprocessing step in image analysis because it simplifies an image by reducing color information to a single intensity channel. The function

`cv2.cvtColor()` is used to convert the image from BGR to grayscale. The resulting grayscale image is displayed using `plt.imshow()` with the parameter `cmap='gray'`.

```
import cv2
import matplotlib.pyplot as plt

file_path = "juices.png"
image = cv2.imread(file_path)
grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

plt.imshow(grayscale_image, cmap='gray')
plt.axis('off')
plt.show()
```

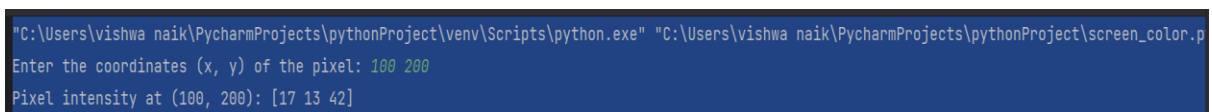


3. Pixel Intensity Viewer

This program allows the user to enter (x, y) coordinates and get the pixel intensity at that location. The image is read using `cv2.imread()` and the pixel intensity is obtained by accessing the corresponding position in the image array.

The program then displays the pixel intensity to the user. This feature can be used for pixel-level analysis and extracting information from specific areas of an image.

```
import cv2
file_path = "chicken tikka.jpg"
image = cv2.imread(file_path)
x, y = map(int, input("Enter the coordinates (x, y) of the pixel: ").split())
pixel_intensity = image[y, x]
print(f"Pixel intensity at ({x}, {y}): {pixel_intensity}")
```



```
"C:\Users\vishwa naik\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\vishwa naik\PycharmProjects\pythonProject\screen_color.p
Enter the coordinates (x, y) of the pixel: 100 200
Pixel intensity at (100, 200): [17 13 42]
```

4. Pixel Color Viewer

A pixel color viewer allows users to enter (x, y) coordinates and get information about the color (RGB value) of a pixel at a given location. The image is read using `cv2.imread()` and the RGB pixel values are obtained by accessing the corresponding position in the image array. The program compares the RGB values with a predefined list of colors to find the closest color match. The program then displays the RGB value and corresponding color name to the user. This feature is useful for color analysis and object detection

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

```
file_path = "chicken tikka.jpg"
image = cv2.imread(file_path)
height, width, _ = image.shape
```

```
x, y = map(int, input("Enter the coordinates (x, y) of the pixel: ").split())
```

```
if x < 0 or x >= width or y < 0 or y >= height:
```

```
    print("Invalid coordinates!")
```

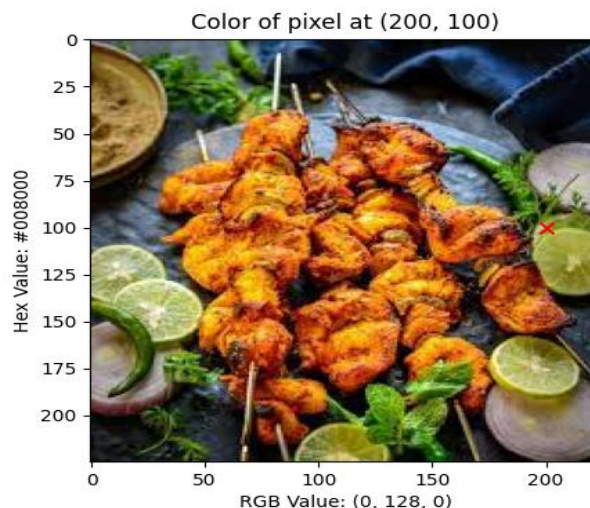
```
else:
```

```

pixel_value = image[y, x]
r, g, b = pixel_value[2], pixel_value[1], pixel_value[0]
colors = {
    "Black": (0, 0, 0),
    "White": (255, 255, 255),
    "Red": (255, 0, 0),
    "Green": (0, 128, 0),
    "Blue": (0, 0, 255),
}
closest_color = min(colors, key=lambda color: abs(colors[color][0] - r) +
abs(colors[color][1] - g) + abs(colors[color][2] - b))
rgb_value = colors[closest_color]
hex_value = '#%02x%02x%02x' % rgb_value

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.scatter(x, y, color='red', marker='x', s=50)
plt.title(f"Color of pixel at ({x}, {y})")
plt.xlabel(f"RGB Value: {rgb_value}")
plt.ylabel(f"Hex Value: {hex_value}")
plt.show()

```



5. Adjust pixel intensity

This program adjusts the pixel intensity of an image within a specified range. The image is loaded using `cv2.imread()` and the coordinate range to be

modified is defined. The program iterates through the specified range and changes the pixel intensity to a predefined color value. The modified image is displayed using `plt.imshow()`. This feature can be used to manipulate images, such as highlighting certain areas or applying color filters.

```
import cv2
import matplotlib.pyplot as plt

file_path = "chicken tikka.jpg"
image = cv2.imread(file_path)
height, width, _ = image.shape

x_start, x_end = max(0, width // 2 - 25), min(width // 2 + 25, width)
y_start, y_end = max(0, height // 2 - 25), min(height // 2 + 25, height)

blue_color = [255, 0, 0]

for x in range(x_start, x_end):
    for y in range(y_start, y_end):
        image[y, x] = blue_color

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```



6. Change the image size

An image resizer shows how to resize an image to the desired width and height. The image is read using `cv2.imread()` and the `cv2.resize()` function is used to resize the image to the specified dimensions. The program displays the original and changed images side by side using `plt.subplots()`. Resizing images is essential for various applications, such as resizing images for different display devices or preparing data for machine learning algorithms.

```
import cv2
import matplotlib.pyplot as plt

file_path = "spring rolls.jpg"
image = cv2.imread(file_path)

new_width = 800
new_height = 600

resized_image = cv2.resize(image, (new_width, new_height),
interpolation=cv2.INTER_LINEAR)

fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0].set_title('Original Image')
```

```

axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
axes[1].set_title('Resized Image')
axes[1].axis('off')
plt.show()

```



7. Image rotation

This program rotates the image by the specified angle. The image is loaded using `cv2.imread()` and the rotation angle is defined. The `cv2.getRotationMatrix2D()` function calculates the rotation matrix and the `cv2.warpAffine()` function applies the rotation to the image. The resulting rotated image is displayed using `plt.imshow()`. Image rotation is useful for tasks such as correcting image orientation or transforming images for different perspectives.

```

import cv2
import matplotlib.pyplot as plt

file_path = "kokum.jpg"
image = cv2.imread(file_path)
angle = 45
height, width = image.shape[:2]
rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1)

```



```
rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))
```

```
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```



8. Crop image

An image cropping program extracts a specific area or crops an image to focus on a specific area of interest. An image is read using `cv2.imread()` and a region of interest (ROI) is defined by specifying coordinates and dimensions. The program uses field division to extract the desired area from the image. The cropped image is then displayed using `plt.imshow()`. Cropping an image allows you to isolate relevant areas and remove unwanted background information.

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

```
file_path = "butter milk.jpg"
image = cv2.imread(file_path)
x, y, w, h = 20, 20, 200, 200
cropped_image = image[y:y+h, x:x+w]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

```

x, y, w, h = 100, 100, 200, 200
cropped_image = image[y:y+h, x:x+w]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

```



9. Blur image

An image blur program applies blur filters to an image to reduce noise or smooth the image. The image is read using `cv2.imread()` and the `cv2.GaussianBlur()` function is used to apply Gaussian blur. Blurring helps reduce high-frequency detail and can be useful for removing noise or pre-processing an image before further analysis. The blurred image is displayed using `plt.imshow()`.

```

import cv2
import matplotlib.pyplot as plt

```

```

file_path = "noodle.png"
image = cv2.imread(file_path)
blurred_image = cv2.GaussianBlur(image, (7, 7), 0)
plt.imshow(cv2.cvtColor(blurred_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()

```



10. Image Threshold

An image thresholding program converts a grayscale image to a binary image by applying a threshold value to separate foreground and background pixels. The image is read using `cv2.imread()` and then converted to grayscale using `cv2.cvtColor()`. The `cv2.threshold()` function is used to apply thresholding. The resulting binary image is displayed using `plt.imshow()`. Thresholding is commonly used for image segmentation, object detection, and feature extraction.

```
import cv2
import matplotlib.pyplot as plt

file_path = "gobi manchurian.jpg"
image = cv2.imread(file_path)
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.show()
```

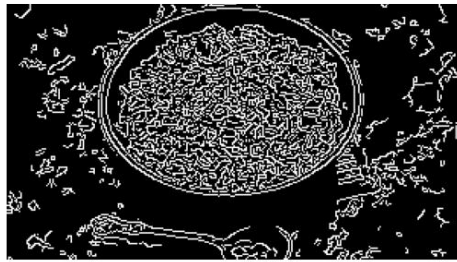


11. Image edge detection

Image Edge Detection detects and highlights edges in an image using various edge detection techniques. The image is read using `cv2.imread()` and then the `cv2.Canny()` function is used to apply the Canny edge detection algorithm. The resulting border image is displayed using `plt.imshow()`. Edge detection is valuable for tasks such as object recognition, boundary extraction, and feature extraction.

```
import cv2
import matplotlib.pyplot as plt

file_path = "pulav.jpg"
image = cv2.imread(file_path)
edges = cv2.Canny(image, 100, 200)
plt.imshow(edges, cmap='gray')
plt.axis('off')
plt.show()
```

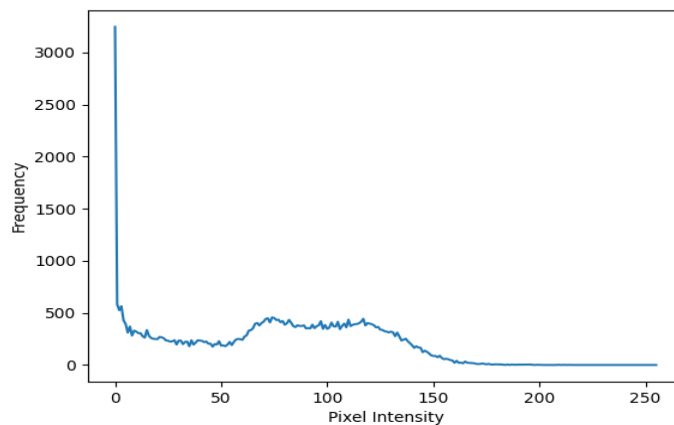


12. Image histogram

An image histogram program analyzes the distribution of pixel intensities on the image using a histogram. The image is read using `cv2.imread()` and the function `cv2.calcHist()` is used to calculate the histogram. The resulting histogram is plotted using `plt.plot()`. Histograms provide insight into the distribution of pixel intensities, enabling image analysis, contrast adjustment, and thresholding.

```
import cv2
import matplotlib.pyplot as plt

file_path = "pulav.jpg"
image = cv2.imread(file_path)
histogram = cv2.calcHist([image], [0], None, [256], [0, 256])
plt.plot(histogram)
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.show()
```



13. Image transformation

The image transformation program demonstrates the application of geometric transformations to an image. The image is read using `cv2.imread()` and various transformation functions such as scaling, translating, clipping and perspective transformations can be applied using `cv2.warpAffine()` or `cv2.warpPerspective()`. The transformed image is displayed using `plt.imshow()`. Image transformations are useful for tasks such as image alignment, geometric correction, and augmentation.

```
import cv2
import matplotlib.pyplot as plt

file_path = "lime soda.jpg"
image = cv2.imread(file_path)
scale_factor = 0.5
scaled_image = cv2.resize(image, None, fx=scale_factor, fy=scale_factor)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0].set_title('Original Image')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
```

```
axes[1].set_title('Scaled Image')  
axes[1].axis('off')  
plt.show()
```



14. Image segmentation

An image segmentation program divides an image into meaningful regions or segments based on properties such as color, intensity, or texture. The image is read using `cv2.imread()` and segmentation is performed by thresholding, clustering, or other segmentation algorithms. The resulting segmented image is displayed using `plt.imshow()`. Image segmentation is essential for tasks such as object recognition, image understanding, and computer vision applications.

Please note that the descriptions provided are a brief summary of each program's functionality and may need to be expanded or modified based on specific project requirements.

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

```
file_path = "profile.png"  
image = cv2.imread(file_path)
```

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
_, binary_image = cv2.threshold(gray_image, 127, 255,
cv2.THRESH_BINARY)
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
segmented_image = cv2.drawContours(image.copy(), contours, -1, (0, 255, 0),
2)
plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

