Introduction to Artificial Intelligence

Computer Vision: OpenCV

Janyl Jumadinova
October 12, 2016
Images

How to input or output an image?
Images

How to input or output an image?

**Load Image**
*Read image from disk.*

```cpp
cv::imread(filename, 0/1);
```

*0: read as grayscale image
  1: read as color image*

**Save Image**
*Write image to disk.*

```cpp
cv::imwrite(filename, im);
```

**Visualize Image**
*Show image in a window.*

```cpp
cv::imshow(title, im);
```

*Note: if CV_32FC1, the gray value range is 0 to 1. Everything above 1 is white and everything below 0 is black.*

**Waitkey**
*Waits n milliseconds for user input.*

```cpp
cv::waitkey(n);
```

*If n == -1, it waits forever. Note: There must be a waitkey to show the image.*
Drawing Primitives

`cv::line(im, p1, p2, color, thickness);`

`cv::Point(x, y)`

`cv::circle(im, c, r, color, thickness);`

`CV_RGB(r, g, b)`
rectangle = np.zeros((300, 300), dtype = "uint8")
cv2.rectangle(rectangle, (25, 25), (275, 275), 255, -1)
Examine every pixel in the input images:

- `cv2.bitwise_and` (used in masking example): if both pixels have a value $> 0$, the output pixel is set to 255 in the output image, otherwise it is 0.

- `cv2.bitwise_or`: if either of the pixels have a value $> 0$, the output pixel is set to 255 in the output image, otherwise it is 0.

- `cv2.bitwise_xor`: same as OR, with a restriction: both pixels are not allowed to have values $> 0$.

- `cv2.bitwise_not`: pixels with a value of 255 become 0, pixels with a value of 0 become 255.
Simple Image Operations: Smoothing

- Each pixel in the image is mixed in with its surrounding pixel intensities, becoming a blurred pixel.
- Smoothing increases performance of many image processing and computer vision applications, such as thresholding and edge detection.

```
Smoothing
cv::GaussianBlur(src, dst, ksize, sigma);

cv::Size(width, height)
```
Smoothing

1. **Standard averaging**: takes the average of all pixels in the surrounding area and replaces the central element of the output image with the average.
   - Uses $k \times k$ sliding (left to right, top to bottom) window (kernel), where $k$ is always odd.
   
   ```python
   cv2.blur(image, (5,5))
   ```
Smoothing

1. **Standard averaging**: takes the average of all pixels in the surrounding area and replaces the central element of the output image with the average.
   - Uses $k \times k$ sliding (left to right, top to bottom) window (kernel), where $k$ is always odd.

   ```python
cv2.blur(image, (5,5))
```

2. **Gaussian**: uses a weighted mean, where neighborhood pixels that are closer to the central pixel contribute more weight to the average.
   - Results in a more naturally blurred image than using the average method.
   - Last argument is the standard deviation in the x-axis

   ```python
cv2.GaussianBlur(image, (3, 3), 0)
```
Smoothing

3 **Median**: replaces the central pixel with the median of the neighborhood.
- Effective in removing salt-and-pepper noise.

```python
cv2.medianBlur(image, 3)
```
3 **Median**: replaces the central pixel with the median of the neighborhood.
- Effective in removing salt-and-pepper noise.
  \[ \text{cv2.medianBlur(image, 3)} \]

4 **Bilateral Filter**: introduces two Gaussian distributions: 1) considers spatial neighbors (pixels that appear close together), 2) models the pixel intensity of the neighborhood, ensuring that only pixels with similar intensity are included in the actual computation of the smoothing.
- Effective in reducing noise while still maintaining edges, but slow.
  \[ \text{cv2.bilateralFilter(image, 5, 21, 21)} \] (image, diameter of the pixel neighborhood, color, space)
Simple Image Operations

Convert Color

```cpp
cv::cvtColor(colorIm, grayIm, CV_BGR2GRAY);
```
Simple Image Operations: Thresholding

- Thresholding is the binarization of an image.
- Convert a grayscale image to a binary image, where the pixels are either 0 or 255.
- Useful when want to focus on objects or areas of particular interest in an image.
Simple Image Operations: Thresholding

- Thresholding is the binarization of an image.
- Convert a grayscale image to a binary image, where the pixels are either 0 or 255.
- Useful when want to focus on objects or areas of particular interest in an image.

1. Convert to grayscale
2. Apply smoothing (blurring): remove some of the high frequency edges in the image that are not of interest
3. Apply thresholding
1. **Basic**: pixel values $> T$ are set to the maximum value (the third argument).
   - Returns two values: 1) $T$, the value we manually specified for thresholding (second argument), 2) actual thresholded image.

\[
\text{cv2.threshold(blurred, 155, 255, cv2.THRESH_BINARY)}
\]
1. **Basic**: pixel values $> T$ are set to the maximum value (the third argument).
   - Returns two values: 1) $T$, the value we manually specified for thresholding (second argument), 2) actual thresholded image.
   ```python
   cv2.threshold(blurred, 155, 255, cv2.THRESH_BINARY)
   ```

2. **Adaptive**: considers small neighbors of pixels and then finds an optimal threshold value $T$ for each neighbor.
Thresholding

2 **Adaptive**: considers small neighbors of pixels and then finds an optimal threshold value $T$ for each neighbor.

2.1 **Mean**: the mean of the neighborhood of pixels $\rightarrow T$.

```python
cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 11, 4)
```
Thresholding

2 **Adaptive**: considers small neighbors of pixels and then finds an optimal threshold value $T$ for each neighbor.

2.1 **Mean**: the mean of the neighborhood of pixels → $T$.
   
   ```python
   cv2.adaptiveThreshold(blurred, 255,
                        cv2.ADAPTIVE_THRESH_MEAN_C,
                        cv2.THRESH_BINARY_INV, 11, 4)
   ```

2.2 **Adaptive Gaussian**: weighted mean of the neighborhood of pixels → $T$.
   
   ```python
   cv2.adaptiveThreshold(blurred, 255,
                         cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                         cv2.THRESH_BINARY_INV, 15, 3)
   ```
First, we find the gradient of the grayscale image, allowing us to find edge-like regions in the x and y direction.
1. **Laplacian** method: computes the gradient magnitude image, with the first argument - grayscale image, the second argument is the data type for the output image.

```python
cv2.Laplacian(image, cv2.CV_64F)
```
- To catch all edges, we use a floating point data type, then take the absolute value of the gradient image and convert it back to an 8-bit unsigned integer.
```
lap = np.uint8(np.absolute(lap))
```
2 **Sobel** function: computes gradient magnitude representations along the x and y axis to find both horizontal and vertical edge-like regions.
- The last two arguments: 1, 0 - to find vertical edge-like regions; 0, 1 - to find horizontal edge-like.
  
  ```
  sobelX = cv2.Sobel(image, cv2.CV_64F, 1, 0)
  sobelY = cv2.Sobel(image, cv2.CV_64F, 0, 1)
  ```
- Combine the gradient images in both the x and y direction with a bitwise OR.
Edge Detection

Canny edge detector

Reveal the outlines of the objects in the image.

1. Smoothen the image to remove noise.
2. Compute Sobel gradient images in the x and y direction, suppressing edges.
3. Suppress the edges.
4. Determine if a pixel is edge-like or not.
Canny Edge Detection

cv2.Canny(image, 30, 150)
Last two arguments: threshold values.
- gradient value < threshold1 is a non-edge, - gradient value > threshold2 is an edge,
- values between threshold1 and threshold2 are either edges or non-edges based on the connection of their intensities.
Class Exercise: Count Objects

1. Parse an image as an argument
2. Convert to grayscale
3. Apply smoothing (blurring)
4. Apply edge detection (Canny)
5. Find contours in the edged image
   
   ```python
   cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE),
   ```
   where edged.copy() is the copy of the output from Step 4. [http://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html](http://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html)
6. Output how many contours did it find
7. Draw a circle around each object in the original image
   
   ```python
   cv2.drawContours(image, cnts, -1, (0, 255, 0), 2)
   ```