Basic Functions

*import cv2 as cv*

# Grayscale

To convert a bgr(blue green red) image to grayscale, we use the following function:

# Reading an image

*img = cv.imread(‘Path to the image’)*

# Conversion

*gray = cv.cvtColor(img,cv.COLOR\_BGR2GRAY)*

# Display the grayscale image

*cv.imshow(“Gray”,gray)*

# Blurring

To blur an image, we use the following function:

*blur = cv.GaussianBlur(img, (3,3), cv.BORDER\_DEFAULT)*

*cv.imshow(‘Blur’, blur)*

There are multiple ways to blur an image, for now we are using gaussian blur.

In the cv.GaussianBlur() function, the tuple containing (3,3) is actually the kernel or a window that is moved across the image. The kernel has to have odd values to have a definitive center. An odd-sized kernel has a single central element, allowing the filter to align with the pixel the is currently being processed. If you try an even matrix in this function in python, you will get the following error:

*cv.error: OpenCV(4.10.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\smooth.dispatch.cpp:294: error: (-215:Assertion failed) ksize.width > 0 && ksize.width % 2 == 1 && ksize.height > 0 && ksize.height % 2 == 1 in function 'cv::createGaussianKernels'*

OpenCV provides several modes for border handling, including:

* cv.BORDER\_CONSTANT: Pads the border with a constant value (default is 0).
* cv.BORDER\_REPLICATE: Repeats the last border pixel (essentially extending the edges).
* cv.BORDER\_REFLECT: Reflects the border pixels symmetrically.
* cv.BORDER\_WRAP: Wraps around the image (pixels on one side are copied from the opposite side).

If you don’t specify, then the default cv.BORDER\_DEFAULT is applied:

cv.BORDER\_DEFAULT uses the cv.BORDER\_REFLECT\_101 mode, which reflects pixels at the image borders with a one-pixel shift. It uses reflective padding. Padding is used for pixels near the edge of the image, part of the kernel extends beyond the image boundaries, meaning there are no neighboring pixels outside the image to apply the filter. There are different kinds of padding methods.

# Edge

For this part you should check OpenCV’s official documentation:  
<https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html>

*canny = cv.Canny(img,thresh1,thresh2) # example usage: 100 for thresh1 and 200 for 2*

The thresh1 and thresh2 are integer values, if you enter floating values, they will be automatically **rounded down** to nearest integer values by opencv. The values of thresh2 must be greater than the value of thresh1. The pixels below thresh1 are treated as “non-edges”, while the pixels above thresh2 are treated as “sure-edges”. The pixels in between the thresh1 and thresh2 are determined edges or not edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

There are some optional parameters in this functional as well:

**edges** (Optional):

* An output array to store the detected edges. If not provided, the function will return the edges.

**apertureSize** (Optional):

* The size of the Sobel kernel used for finding image gradients. It must be odd and can be 3, 5, or 7. The default value is 3.

**L2gradient** (Optional):

* A boolean flag indicating whether to use a more accurate norm for edge detection.
  + If False (default), the L1 norm (|dx| + |dy|) is used.
  + If True, the L2 norm (sqrt(dx^2 + dy^2)) is used for calculating image gradient magnitude.

Again I emphasize you to read opencv’s documentation for this function for better understanding, the link is provided above.

*cv.imshow(“Canny”,canny) # use it to display the image with edges*

We can reduce the edges, by passing a blurred image as:

*canny = cv.Canny(blur,thresh1,thresh2)*

# Erosion and Dilation

“**Morphological operations** are a set of operations that process images based on shapes. They apply a structuring element to an input image and generate an output image.   
The most basic morphological operations are two:**Erosion and Dilation**   
**Basics of Erosion:**

* Erodes away the boundaries of the foreground object
* Used to diminish the features of an image.

**Basics of dilation:**

* Increases the object area
* Used to accentuate features

” GeeksforGeeks

*# Dilating an image*

*dilated = cv.dilate(canny, (3,3), iterations=3)*

*# Displaying*

*cv.imshow(“Dilated”, dilated)*

*# Eroding an image*

*eroded = cv.erode(dilated, (3,3), iterations=3)*

*# Display*

*cv.imshow(“Eroded”, eroded)*

By setting the same parameter values, you can get almost get the original canny image back after applying erosion to the dilated image.

Parameters:

**Kernel**: A matrix (typically 3x3, 5x5, etc.) that defines the region for erosion/dilation.

**Iterations**: Specifies how many times the erosion/dilation is applied (default is 1).

# Resizing and Cropping

We have created a custom rescale function as well in the previous module.

*# Resizing*

*resized = cv.resize(img,(500,500)) # using default interpolation, you can use other*

*# try cv.INTER\_CUBIC*

*cv.imshow("Resized", resized)*

cv.INTER\_CUBIC is slow but gives the highest detailed image.

# Cropping

Since, the image is actually an array, we can crop it by using array slicing:

*cropped = img[20:200, 200:400]*

*cv.imshow("Cropped", cropped)*