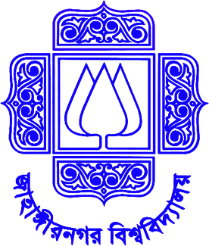
Institute of Information Technology

Jahangirnagar University

Savar, Dhaka-1342



Lab Manual

Course Code: ICT-4202

Course Title: Digital Image Processing Lab

**Lab No.: 5**

**Lab Title: Basic Image Processing Using Scikit-Image Library**

Prepared by

**Mehrin Anannya**

Assistant Professor

Institute of Information Technology

Jahangirnagar University

**Lab Title: Basic Image Processing Using Scikit-Image Library**:

To introduce students with Image processing library named Scikit-image.

**Lab Contents:**

* Image transformation works using Scikit-image library.
* Image blurring using Gaussian filtering techniques.
* Find edges from the images.

**Theory with Hands on Practice:**

# Image processing using scikit-image library:

Let us start by looking at basic image transformation tasks

**Resize, rescale:**

1. Importing Libraries:

- `matplotlib.pyplot`: A plotting library for creating visualizations.

- `skimage.io`: Input/output functions from the scikit-image library.

- `skimage.color`: Functions for color conversion.

- `skimage.transform`: Functions for image transformations.

2. Reading an Image:

- `io.imread()`: Reads an image file. `as\_gray=True` converts the image to grayscale.

3. Rescaling Image:

- `rescale()`: Rescales the input image by a given factor. In this case, it's rescaled by 1/4th (0.25) of the original size.

- `anti\_aliasing=False`: Disables anti-aliasing, which can be used to avoid artifacts.

4. Resizing Image to Specific Dimensions:

- `resize()`: Resizes the input image to the specified dimensions (200x200 in this case).

- `anti\_aliasing=True`: Enables anti-aliasing to reduce aliasing artifacts.

5. Downscaling Image using Local Mean:

- `downscale\_local\_mean()`: Downsamples the image by considering the local mean of elements in each block defined by the given factor (4x3 in this case).

- The result is displayed using `plt.imshow(img\_downscaled)`.

This script demonstrates three different ways to modify the size of an image: rescaling by a factor, resizing to specific dimensions, and downscaling using the local mean. The final downscaled image is then visualized using `matplotlib.pyplot.imshow()`.

import matplotlib.pyplot as plt

from skimage import io, color

from skimage.transform import rescale, resize, downscale\_local\_mean

img = io.imread("images/Osteosarcoma\_01.tif", as\_gray=True)

#Rescale, resize image by a given factor. While rescaling image

#gaussian smoothing can performed to avoid anti aliasing artifacts.

img\_rescaled = rescale(img, 1.0 / 4.0, anti\_aliasing=False) #Check rescales image size in variable explorer

#Resize, resize image to given dimensions (shape)

img\_resized = resize(img, (200, 200), #Check dimensions in variable explorer

anti\_aliasing=True)

#Downscale, downsample using local mean of elements of each block defined by user

img\_downscaled = downscale\_local\_mean(img, (4, 3))

plt.imshow(img\_downscaled)

**A quick look at a few skimage functions:**

1. Importing Libraries:

- `io`: Input/output functions from the scikit-image library.

- `gaussian` and `sobel`: Functions from the `skimage.filters` module for applying Gaussian filtering and Sobel edge detection, respectively.

Gaussian filtering is linear, meaning it replaces each pixel by a linear combination of its neighbors (in this case with weights specified by a Gaussian matrix).

Sobel edge detection: The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically, it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

2. Reading and Displaying an Image:

- `io.imread()`: Reads an image file.

- `plt.imshow()`: Displays the image using matplotlib.

3. Applying Gaussian Filter:

- `gaussian()`: Applies a Gaussian filter to the input image.

- `sigma=1`: Specifies the standard deviation of the Gaussian filter.

- `mode='constant', cval=0.0`: Defines the mode and constant value for handling the border of the image.

- The result is displayed using `plt.imshow()`.

4. \*\*Reading and Applying Sobel Filter to a Grayscale Image:

- `io.imread("images/Osteosarcoma\_01.tif", as\_gray=True)`: Reads the image in grayscale.

- `sobel()`: Applies the Sobel edge detection filter to the grayscale image.

- `cmap='gray'`: Specifies the colormap for displaying the image using `plt.imshow()`.

gaussian(img, sigma=1, mode='constant', cval=0.0)

In the above function:

img: This parameter represents the input image on which Gaussian blurring will be applied. It could be a grayscale image or a multi-channel image.

sigma: This parameter specifies the standard deviation of the Gaussian kernel used for blurring. A larger value of sigma will result in a blurrier image. If not specified, it defaults to 1.

mode: This parameter defines how boundaries are handled during convolution. It can take several values including:

'constant': Pads the image with a constant value specified by the cval parameter.

'reflect': Reflects the image at the boundaries.

'nearest': Replicates the edge values of the image to extend it.

'wrap': Wraps the image around.

'mirror': Reflects the image such that it appears tiled.

cval: This parameter is only relevant when mode='constant'. It specifies the constant value to use for padding the image boundaries.

sigma- Sigma defines the amount of blurring

These code snippets demonstrate loading, displaying, and applying filters (Gaussian and Sobel) to images using the scikit-image library. The results are then visualized using matplotlib.

from skimage import io

from skimage.filters import gaussian, sobel

img = io.imread("images/Osteosarcoma\_01\_25Sigma\_noise.tif")

plt.imshow(img)

gaussian\_using\_skimage = gaussian(img, sigma=1, mode='constant', cval=0.0)

plt.imshow(gaussian\_using\_skimage)

img\_gray = io.imread("images/Osteosarcoma\_01.tif", as\_gray=True)

sobel\_img = sobel(img\_gray) #Works only on 2D (gray) images

plt.imshow(sobel\_img, cmap='gray')

# Basic image processing using opencv:

openCV is a library of programming functions mainly aimed at computer vision. Very good for images and videos, especially real time videos. It is used extensively for facial recognition, object recognition, motion tracking, optical character recognition, segmentation, and even for artificial neural netwroks.

Useful preprocessing steps for image processing, for example segmentation.

1. SPlit & Merge channels

2. Scaling / resizing

4. Edge detection

### Scaling:

Scaling is just resizing of the image. OpenCV comes with a function [**cv2.resize()**](https://docs.opencv.org/3.3.1/d5/df1/group__imgproc__hal__functions.html#ga2fe39d2201b12e1b961ca56b2aff9ff2) for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are **cv2.INTER\_AREA** for shrinking and **cv2.INTER\_CUBIC** (slow) & **cv2.INTER\_LINEAR** for zooming. By default, interpolation method used is **cv2.INTER\_LINEAR** for all resizing purposes. You can resize an input image either of following methods:

import cv2

import numpy as np

img = cv2.imread('messi5.jpg')

res = cv2.resize(img,None,fx=2, fy=2, interpolation = cv2.INTER\_CUBIC)

#OR

height, width = img.shape[:2]

res = cv2.resize(img,(2\*width, 2\*height), interpolation = cv2.INTER\_CUBIC)

**Resize images:**

1. use cv2.resize. Can specify size or scaling factor.
2. Inter\_cubic or Inter\_linear for zooming.
3. Use INTER\_AREA for shrinking
4. Following xample zooms by 2 times.

import cv2

img = cv2.imread("images/RGBY.jpg", 1) #Color is BGR not RGB

resized = cv2.resize(img, None, fx=2, fy=2, interpolation = cv2.INTER\_CUBIC)

cv2.imshow("original pic", img)

cv2.imshow("resized pic", resized)

cv2.waitKey(0)

cv2.destroyAllWindows()

**To merge each image into bgr:**

import cv2

# Read the two images

image1 = cv2.imread('Images/Flower1.jpeg')

image2 = cv2.imread('Images/Flower2.jpeg')

# Resize image2 to match the dimensions of image1

image2=cv2.resize(image2, (image1.shape[1], image1.shape[0]))

# Merge the two images (simple addition)

merged\_image = cv2.addWeighted(image1, 0.5, image2, 0.5, 0)

# Display or save the merged image

cv2.imshow('Merged Image', merged\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Edge detection:**

* cv2.Canny(img, 100, 200): Applies the Canny edge detector to the grayscale image (img). The second and third arguments are the minimum and maximum intensity gradient values to consider for an edge.

import cv2

img = cv2.imread("images/Osteosarcoma\_01.tif", 0)

edges = cv2.Canny(img,100,200) #Image, min and max values

cv2.imshow("Original Image", img)

cv2.imshow("Canny", edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Image filtering in python - Unsharp mask:

Unsharp mask enhances edges by subtracting an unsharp (smoothed) version of the image from the original. Effectively making the filter a high-pass filter.

**enhanced image = original + amount \* (original - blurred)**

The amount of sharpening can be controlled via a scaling factor, a multiplication factor for the sharpened signal. skimage uses Gaussian smoothing for image blurring therefore the radius parameter in the unsharp masking filter refers to the sigma parameter of the gaussian filter.

#This code shows that unsharp is nothing but original + amount \*(original-blurred)

from skimage import io, img\_as\_float

from skimage.filters import unsharp\_mask

from skimage.filters import gaussian

img=img\_as\_float(io.imread("images/sandstone\_blur\_2sigma.tif", as\_gray=True))

gaussian\_img = gaussian(img, sigma=2, mode='constant', cval=0.0)

img2 = (img - gaussian\_img)\*2.

img3 = img + img2

from matplotlib import pyplot as plt

plt.imshow(img3, cmap="gray")

**skimage.filters.unsharp\_mask used directly to apply the unsharp mask:**

from skimage import io

from skimage.filters import unsharp\_mask

img = io.imread("images/sandstone\_blur\_2sigma.tif")

#Radius defines the degree of blurring

#Amount defines the multiplication factor for original - blurred image

unsharped\_img = unsharp\_mask(img, radius=3, amount=2)

import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12, 12))

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(img, cmap='gray')

ax1.title.set\_text('Input Image')

ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(unsharped\_img, cmap='gray')

ax2.title.set\_text('Unsharped Image')

plt.show()

# Image filtering- Gaussian denoising for noise reduction:

**cv2.filter2D:** https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=filter2d#filter2d

**cv2.GaussianBlur** - https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/

**skimage.filters.gaussian** - https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.gaussian

**Application of Gaussian blur to an image using both OpenCV (`cv2.GaussianBlur`) and scikit-image (`skimage.filters.gaussian`).**

img\_gaussian\_noise = img\_as\_float(io.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', as\_gray=True))

img\_salt\_pepper\_noise =img\_as\_float(io.imread('images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif', as\_gray=True))

Reading two different images: one with Gaussian noise (`img\_gaussian\_noise`) and one with salt and pepper noise (`img\_salt\_pepper\_noise`).

gaussian\_using\_cv2 =cv2.GaussianBlur(img, (3,3), 0, borderType=cv2.BORDER\_CONSTANT)

Applying Gaussian blur using the `cv2.GaussianBlur` function from OpenCV. The kernel size is (3,3), and the standard deviation (`sigma`) is set to 0.

gaussian\_using\_skimage = gaussian(img, sigma=1, mode='constant', cval=0.0)

Applying Gaussian blur using the `skimage.filters.gaussian` function from scikit-image. The standard deviation (`sigma`) is set to 1.

cv2.imshow("Original", img)

cv2.imshow("Using cv2 gaussian", gaussian\_using\_cv2)

cv2.imshow("Using skimage", gaussian\_using\_skimage)

Displaying the original image and the images after applying Gaussian blur using both OpenCV and scikit-image.

This code allows us to compare the results of Gaussian blur applied using OpenCV and scikit-image on an image with Gaussian noise. The differences in the parameters used (such as the kernel size and standard deviation) may result in slightly different visual outputs.

import cv2

from skimage import io, img\_as\_float

from skimage.filters import gaussian

img\_gaussian\_noise = img\_as\_float(io.imread('images/Osteosarcoma\_01\_25Sigma\_noise.tif', as\_gray=True))

img\_salt\_pepper\_noise= img\_as\_float(io.imread('images/Osteosarcoma\_01\_8bit\_salt\_pepper.tif', as\_gray=True))

img = img\_gaussian\_noise

gaussian\_using\_cv2 = cv2.GaussianBlur(img, (3,3), 0, borderType=cv2.BORDER\_CONSTANT)

gaussian\_using\_skimage = gaussian(img, sigma=1, mode='constant', cval=0.0)

#sigma defines the std dev of the gaussian kernel. Slightly different than

#how we define in cv2

cv2.imshow("Original", img)

cv2.imshow("Using cv2 gaussian", gaussian\_using\_cv2)

cv2.imshow("Using skimage", gaussian\_using\_skimage)

cv2.imshow("Using Salt Pepper Image", img\_salt\_pepper\_noise)

cv2.waitKey(0)

cv2.destroyAllWindows()

Task:

* Take a screenshot of an image and apply gaussian filtering on the background of the image and find edged from it.