LAB # 02: Basic Image Processing using Python Packages

Lab Objective:

To introduce students with python programming language and its packages.

Lab Description:

Open Source **C**omputer **V**ision Library (**OpenCV**) is an open source software library written in C++ for machine learning and computer vision applications.

To install packages for Python, the following command can be used: pip install package_name. Install the following packages for this lab:

- opencv-python
- numpy // Used for numerical operations

NumPy:

NumPy is a Python package that is used for numeric calculations relying on mathematical formulae and matrices. Many packages in Python rely heavily on numpy including OpenCV.

NumPy allows us to create 2-D arrays (images) that can be manipulated using different commands. Some of those commands are given here.

1. To create a 2D array of zeros using NumPy:

   my_array = **numpy.zeros ((row, columns), dtype=numpy.uint8)**
2. To create a 2D array of ones using NumPy:

   my_array = **numpy.ones ((row, columns), dtype=numpy.uint8)**
3. To check the size of a 2D array: size = **numpy.shape(my_array)**
4. To slice a 2D array (Note the difference from indexing that we were using for Python lists):

   x = y [**row_start: row_ end**, **col_start: col_end**]
5. To horizontally, vertically and depth-wise concatenate images together using Numpy:

   h_concatenated = **np.hstack ((a, b))**
   v_concatenated = **np.vstack ((a, b))**
   d_concatenated = **np.dstack ((a, b, c))**

Why OpenCv?

OpenCV is comprehensive collection of more than 2500 machine learning and computer vision algorithms that can be used from something as simple detecting faces in images to project augmented reality overlaid with scenery.

Another area in which OpenCv excels is its superior support for multiple interfaces and all the major operating systems as compared to other solutions. OpenCv supports Windows, Linux, OS X and Android and provides interfaces for C, C++, Python, Java and MATLAB.

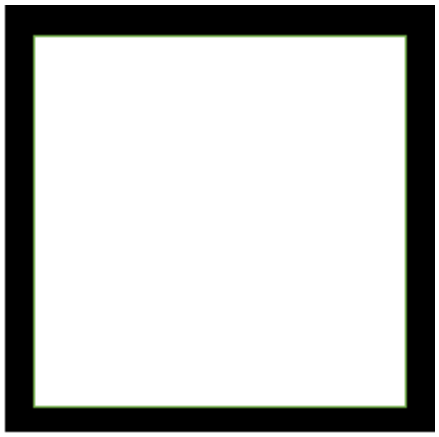
Some of the applications that can be accomplished easily with OpenCv are: identifying objects, tracking camera movements, stitching images together, finding similar images in a database using an image, face detection and tracking moving objects in a video feed etc.

Some useful OpenCV commands are given below:

1. Reading an image using OpenCv**: my_image = cv2.imread("test_image.jpg",0)**
   The second argument determines whether the image is read as a grayscale image or a colored image. **0** is used for reading an image as **grayscale** and while **1** is used for reading in **color**. If **-1** is passed then the image is read **as is**.
2. Displaying an image using OpenCv: **cv2.imshow ("Title of the window", my_image)**.
   Two more commands that need to be used while displaying an image are: **cv2.waitKey(x) cv2.destroyAllWindows ()**. The **waitKey ()** function waits for a key being pressed for x number of milliseconds. If **0** is passed to **waitKey ()** as an argument, it will wait indefinitely for a key press. cv2.destroyAllWindows () closes are the open image windows.
3. Writing an image to disk: **cv2.imwrite("image_name.jpg", my_image)**
4. Resizing an image can be achieved by: **my_image_resized = cv2.resize(my_image, (780, 540), interpolation = cv2.INTER_LINEAR)**
   One of the following options can be used for interpolation type: **cv2.INTER_AREA, cv2.INTER_CUBIC, cv2.INTER_LINEAR.**

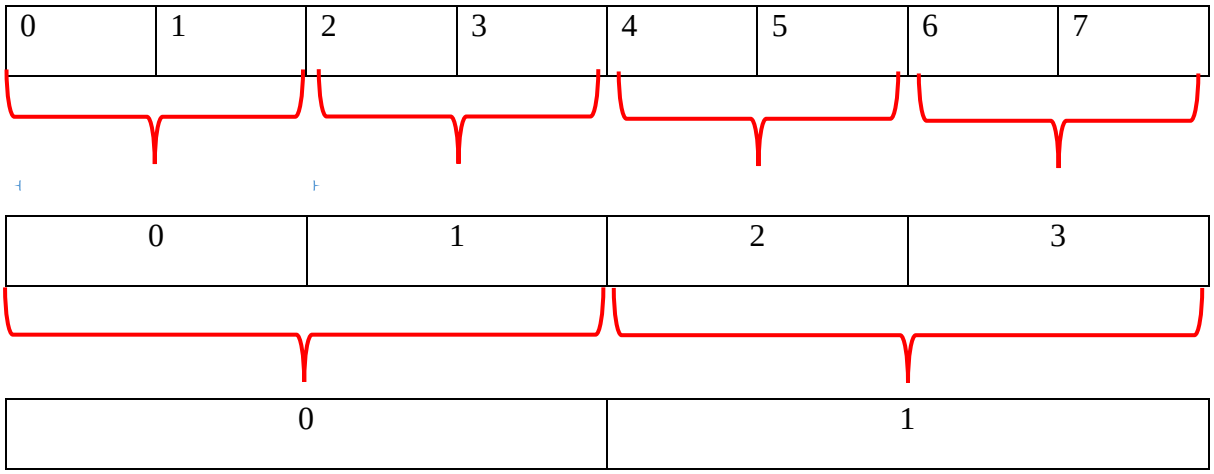
Lab Tasks:

1. Create an *r x c* matrix of ones and pad 10 pixels wide border of zeros across each side of it, such that its order will become (4+500+4) x (4+500+4) = 508x508 as shown below:



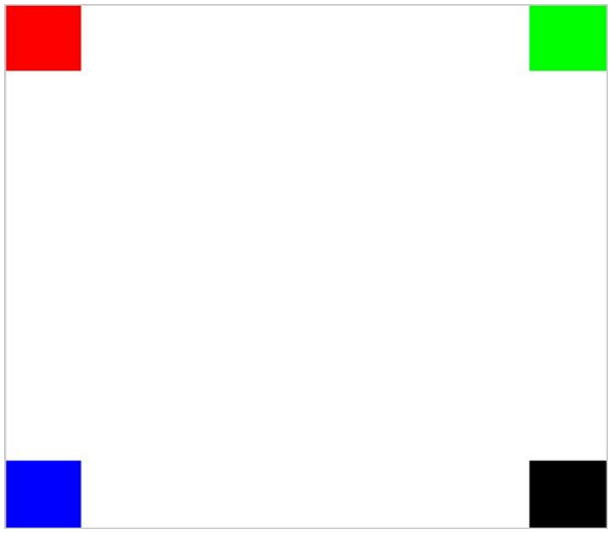
Create a generic function so that the values (500 and 4) can be passed by the user.

2. **Intensity level resolution** defines the resolution at bit level i.e. how many bits are used to represent a pixel value. The more bits we have per pixel, the more levels there are. For example, a grayscale image has 256 different levels because for each pixel value we use 8 bits to store the value. If the bits per pixel are decreased to 4, then the maximum levels that we can have is 16. Similarly, if only 1 bit is used for it then we can have only two levels (or a binary image).
   To change the bit level resolution, different levels (or intensities) can be grouped together to form new levels. For example, for a 3 bits/pixel image:



Read a grayscale image (given below) and convert the image to 16 levels, then to 4 levels and finally to 1. Display all four images.

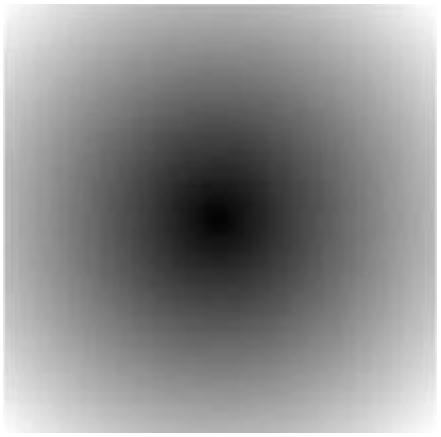
3. Write a function to create a white image of 500x500 (or any other size entered by the user) and then create 4 boxes of Red, Green, Blue and Black respectively on each corner of the image as shown below. The size of the colored boxes should be $1/8^{th}$ the size of the image. (**HINT:** the arrays of ones and zeros can be in more than 2 dimensions)



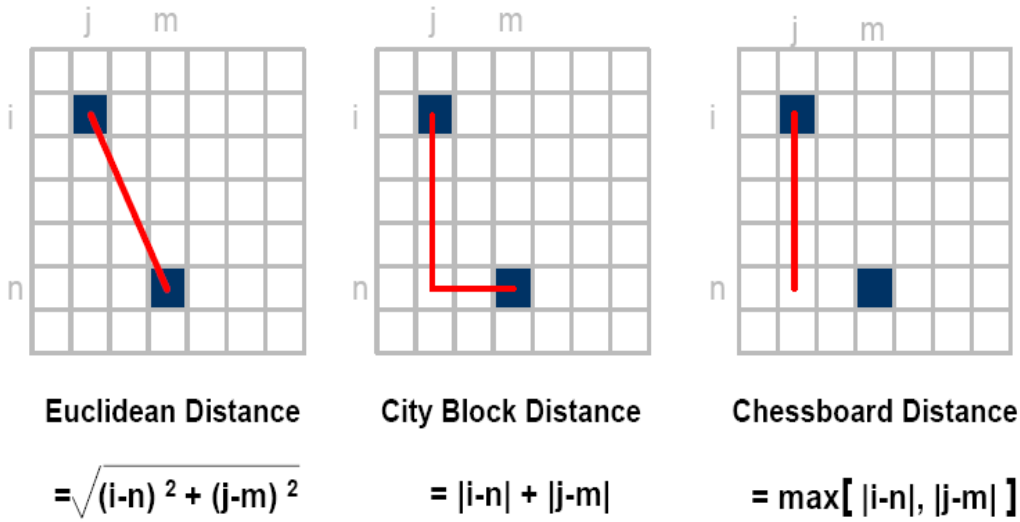
4. Read an image and flip it vertically, horizontally and vertically and horizontally as shown below:

5. Using the same image that you have used before, modify the image at center i.e. the lower half of the image should be the copy of the upper half as shown below (**HINT:** You can use nested loops). Write the image to the disk using the right command.

6. Create an empty image of zeros of size 501x501 and then create a distance map by calculating the Euclidian Distance of every pixel from the center. Display the distance map. It should look something like given here. Enclose you code in the form of a function and let the user decide whether Euclidian distance should be used for distance or any other distance formula e.g. City block distance. (Don't forget to declare the data type of the array of zeros as **uint8**):



The different distances can be calculated as follows:



7. Read an image from disk and decimate it by 4 i.e. reduce the size of the image by 4. (A 1024x1024 image will become 256x256). Now, take the decimated image and interpolate it by 4 times returning it to original size. Use pixel replication interpolation for this purpose. Display all 3 images side by side. This task is to be accomplished using loops.

Home Task

1. Write 3 different Python functions that can create the images given below. Code them in such so that the size of the image itself and the boxes and lines can be changed.