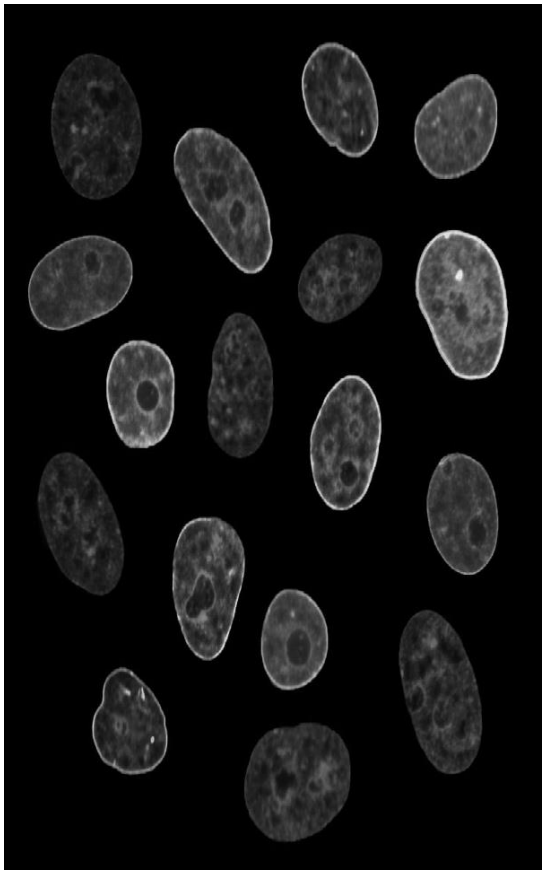
COSC 6380 Digital Image Processing

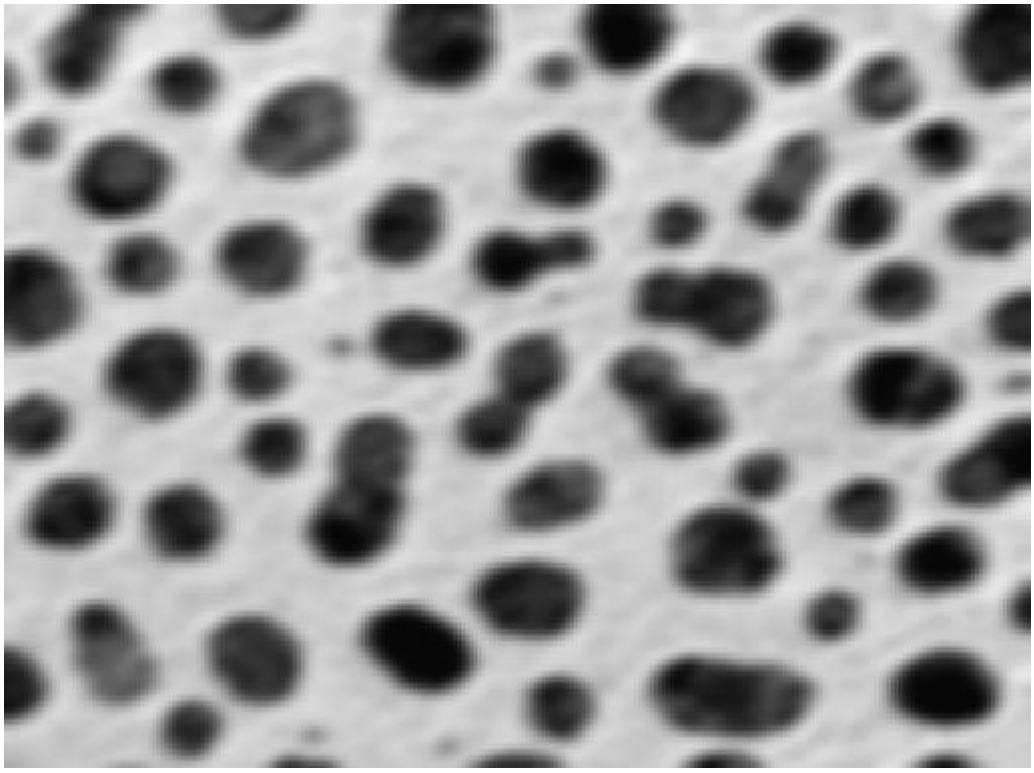
Resampling

1. Interpolation using nearest neighbour

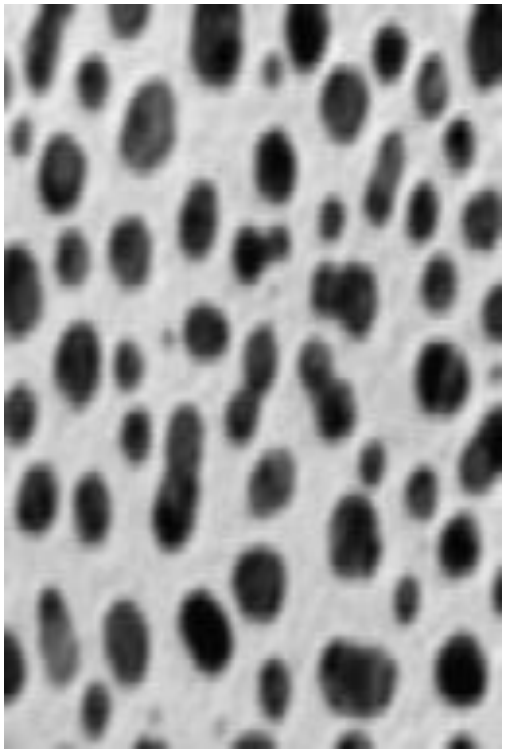
- In this assignment, I used two methods to resample the given images.
- I used PyCharm software to implement and debug the algorithm.
- According to the definition, I took 2 scaling values as an input along x-direction (fx) and y-direction (fy) and any given image to resample using scaling factors.
- Walking through the algorithm,
    i. First, I created an empty image having shape of (w*fy, h*fx), where, w and h is the shape of original image and fx and fy are scaling factors.
    ii. Iterating through rows and columns of newly formed empty image, I found new indices using (i/fy) and (j/fx) and using calculated indices, assigned the pixel value of corresponding original image to newly formed image.
    iii. Returned the newly formed image after going through all the rows and columns.
- Algorithm was written in nearest_neighbor() function of resample() class.
- Problem faced,
    i. Initializing an empty image with proper shape.
    j. Finding indices which will be used for assigning to the new image. This problem was solved using int(np.floor()) instruction.
- Algorithm worked perfectly for both the given images with any scaling factor. Here are the outputs,



fx = 0.5, fy = 0.75



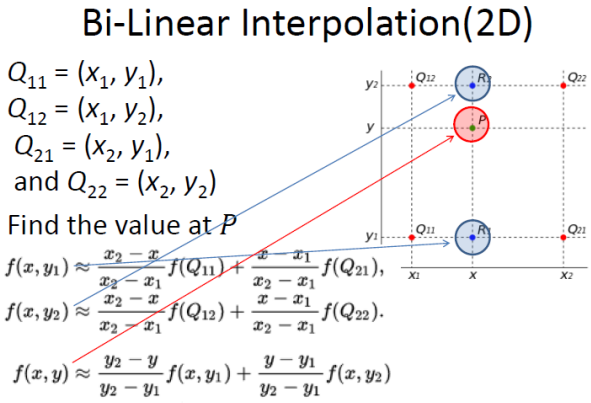
fx = 2, fy = 1.5



fx = 0.5, fy = 0.75

2. Interpolation using bilinear

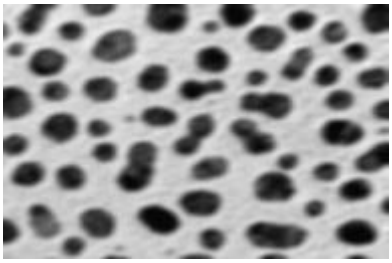
- Second part of the assignment was resampling using bilinear method.
- Algorithm was written in bilinear_interpolation () function of resample() class.
- For generalizing the code, we had given interpolation() class file with linear_interpolation() and bilinear_interpolation() functions.
- linear_interpolation() was having 2 known points along with intensity values and one unknown point to calculated intensity level.
- Whereas, bilinear_interpolation() was having 4 known inputs along with intensity levels and one unknown point where we had to calculate intensity value.
- I used linear_interpolation() function to implement the algorithm and called thrice in bilinear_interpolation() to get intensity level at unknown point.



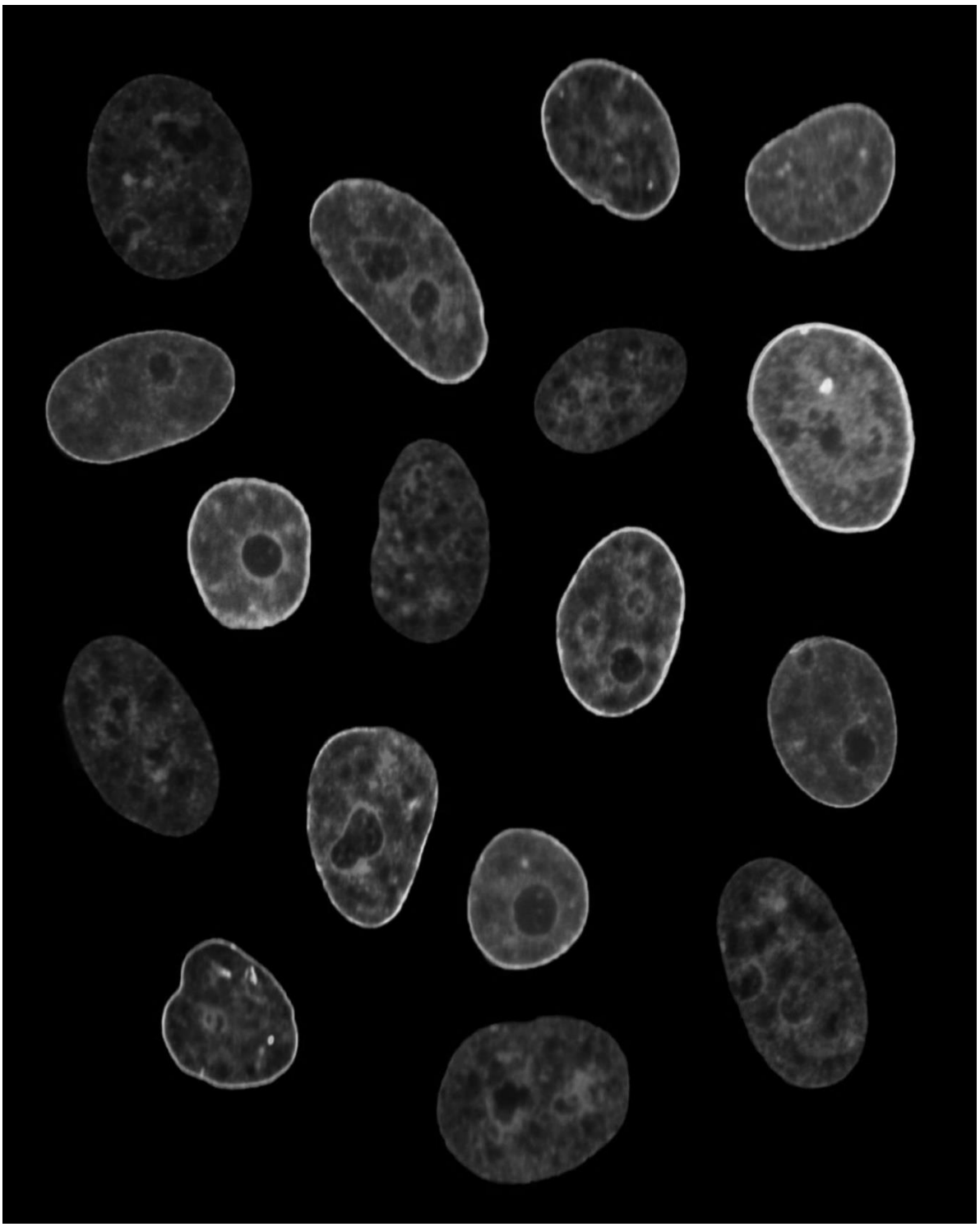
- Walking through the algorithm,
  (a) First, I created an empty image having shape of (w*fy, h*fx), where, w and h is the shape of original image and fx and fy are scaling factors.
  (b) Iterating through rows and columns of newly formed empty image, I found new indices (nx, ny) using (i/fy) and (j/fx) and using calculated indices, I found 4 new points (x1,y1), (x1,y2), (x2,y1) and (x2,y2) using np.floor() and np.ceil(). Along with that, I stored intensity values of all 4 calculated points in 4 different variables.
  (c) Now, to find the intensity value at (nx, ny), I called bilinear_interpolation() function from interpolation() class.
  (d) In bilinear_interpolation() function, I called linear_interpolation() function in which algorithm was written.
  (e) In linear_interpolation(), 2 known points (x1,y1) ad (x2,y2) along with intensity at those points and one unknown point (x,y) were the inputs and intensity at unknown point (i) was the output. Intensity level was calculated using linear interpolation formula.

$$I = \frac{I_1(x_2 - x)}{(x_2 - x_1)} + \frac{I_2(x - x_1)}{(x_2 - x_1)}$$

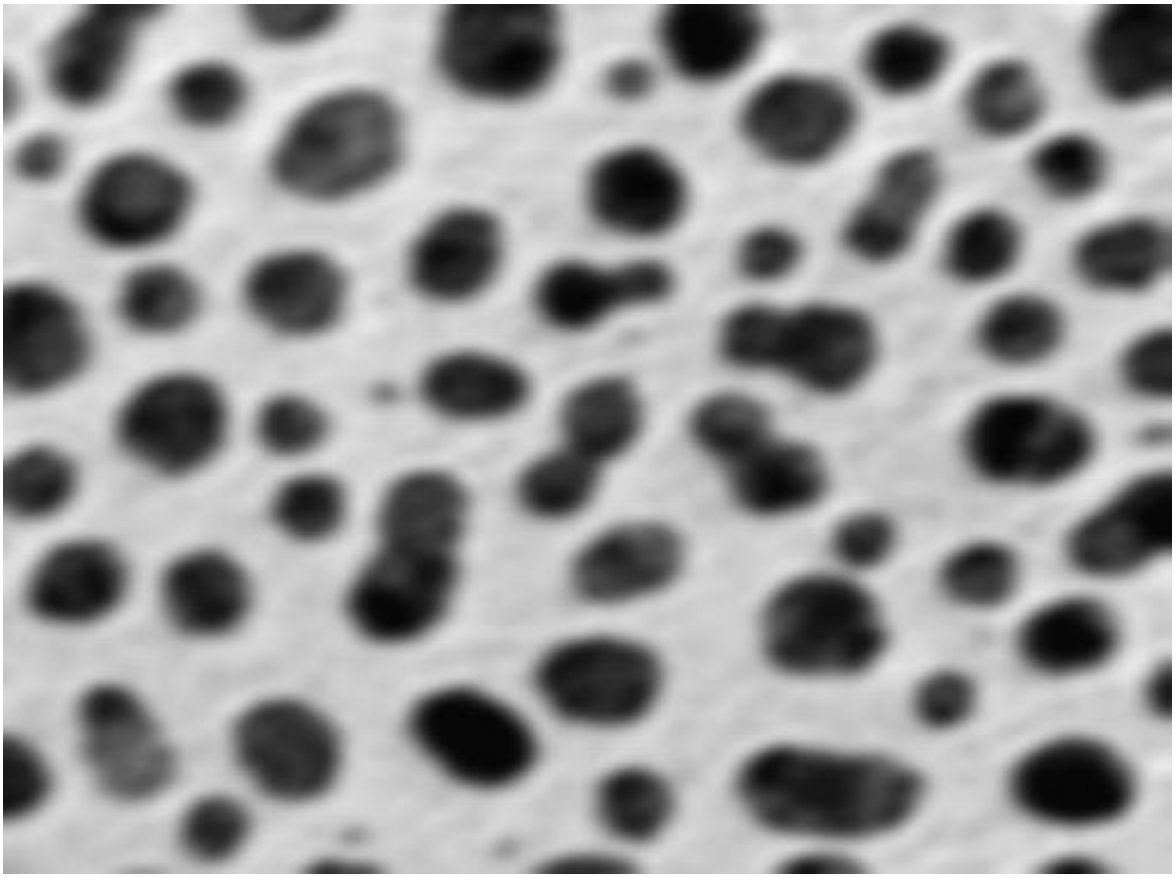
  (f) Returned the newly formed image after going through all the rows and columns.
- Problem faced,
  - Finding the 4 new points (x1,y1), (x1,y2), (x2,y1) and (x2,y2). Solved using np.floor() and np.ceil()
  - Keeping the x1, x2, y1, y2 in the range.
- Algorithm worked perfectly for both the given images with any scaling factor. Here are the outputs,



fx = 0.75, fy = 0.5



fx = 1.5, fy = 1.75



fx = 2, fy = 1.5