Australian
National
University

ENGN6528 Computer Vision
2020 Computer Lab-1 (CLab1)

Prepared by
Lihui Zhang
u6972739

30/03/2020

Australian National University

Task-1: MATLAB (Python) Warm-up. (2 marks):

1. a = np.array([[2,4,5],[5,2,200]])

Create a two-dimension ndarray

```
a: [[  2   4   5]
 [  5   2 200]]
```

2. b = a[0,:]

Get first row of a

```
b:  [2 4 5]
```

3. f = np.random.rand(500,1)

Random value in [0,1) in the (500,1) shape

```
f:  [[9.77325122e-01]
 [1.30044547e-01]
 [1.52295322e-01]
```

4. g = f[f<0]

Create a ndarray g with the same shape of f, but the elements should be less than 0

```
g:  []
```

5. x = np.zeros ((1,100)) + 0.35

Create a zero ndarray in the shape of (1,100) and add 0.35 in each element

```
x:  [[0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35
  0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35
  0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35
```

6. y = 0.6 .* np.ones(1,len(x))

```
y:  [[0.6]]
```

7. z = x – y

x minus y in each element

```
z shape is: (1, 100)
z:  [[-0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25
  -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -0.25
```

8. a = np.linspace(1,50)

Create a one dimension from 1 to 50, and step is 1

```
a: [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36.
 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50.]
```

9. b = a[: :-1]

Take a's elements from end to beginning and step is -1

```
b:  [50. 49. 48. 47. 46. 45. 44. 43. 42. 41. 40. 39. 38. 37. 36. 35. 34. 33.
 32. 31. 30. 29. 28. 27. 26. 25. 24. 23. 22. 21. 20. 19. 18. 17. 16. 15.
 14. 13. 12. 11. 10.  9.  8.  7.  6.  5.  4.  3.  2.  1.]
```

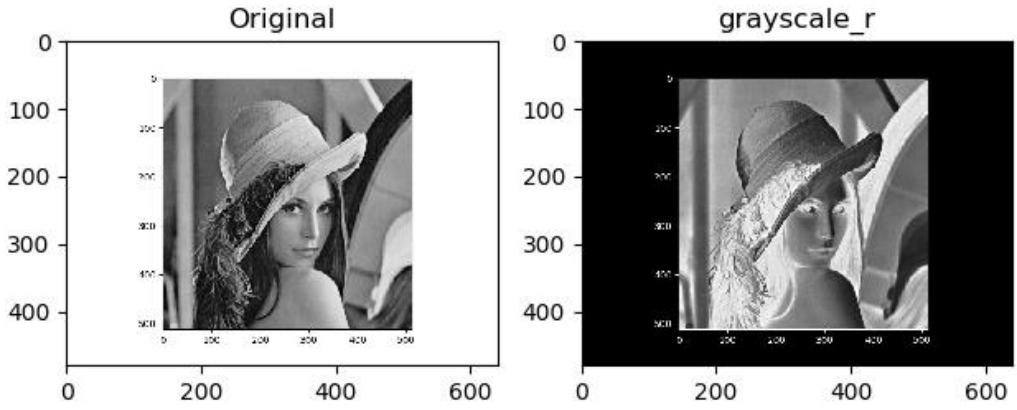
10. b[b <=50]=0

Each elements in b which <= 50 turn to 0

```
b:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
```

Task-2: Basic Coding Practice

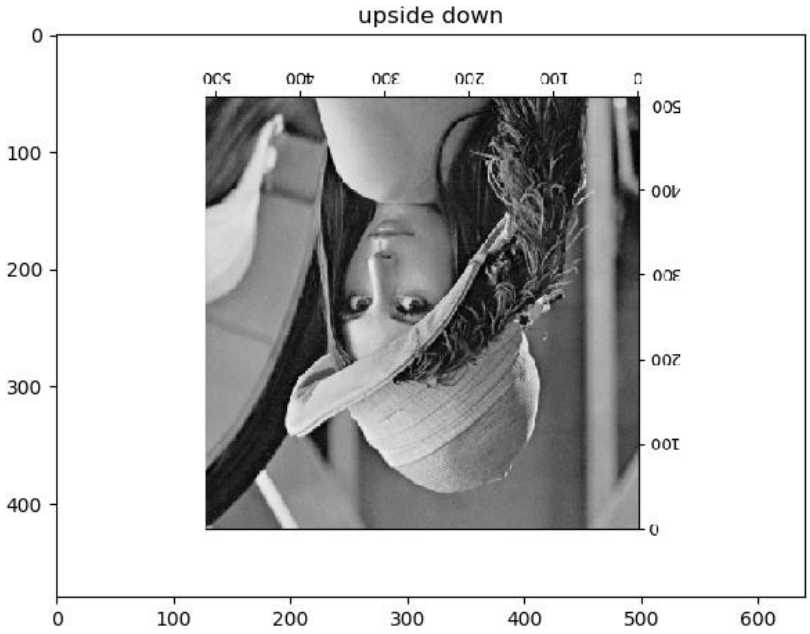
1. Load a grayscale image, and map the image to its negative image, in which the lightest values appear dark and vice versa. Display it side by side with its original version.



Caption: Load the given image in the Lab package and convert it to grayscale image. Use 255 minus the original image to get the negative ndarray, then plot the negative image

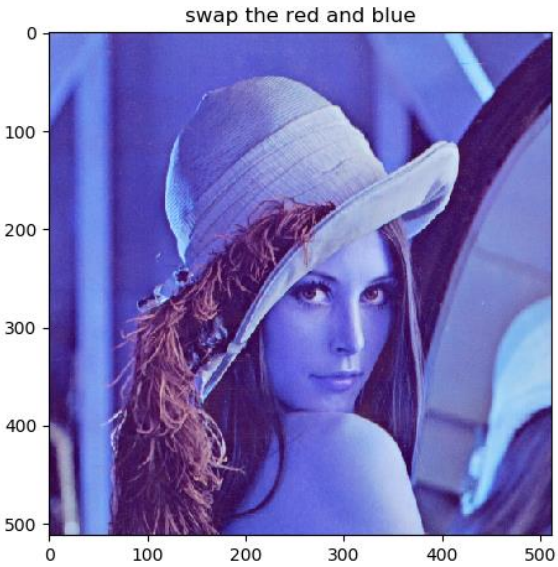
2. Flip the image vertically (i.e, map it to an upside down version).

Caption: Get the transformation matrix first. Use warpAffine function get an upside-down picture img1



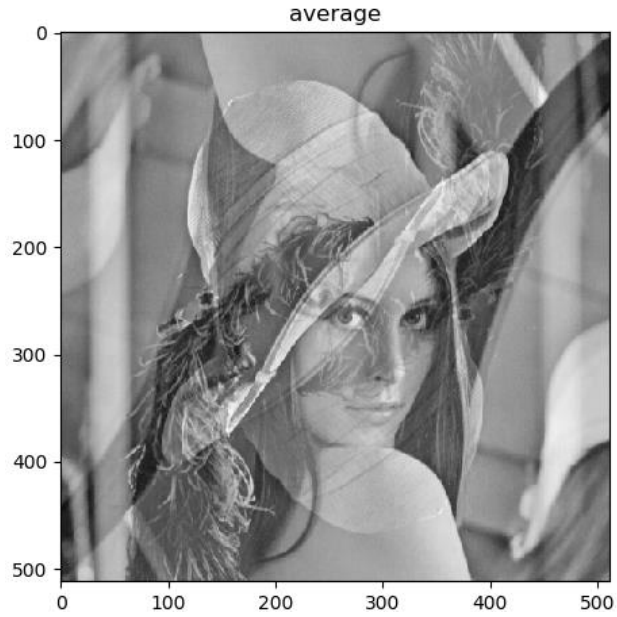
3. Load a colour image, swap the red and blue colour channels of the input

Caption: split the RGB channel and swap the R,B array. Merge the new RGB on a new ndarray and plot it



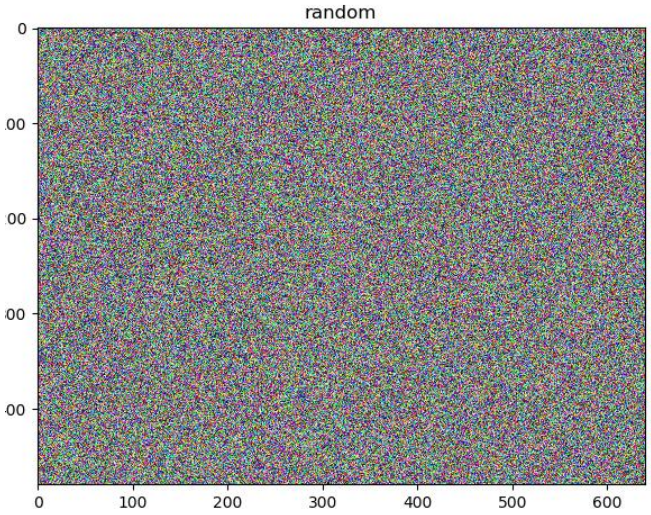
4. Average the input mage with its vertically flipped image (use typecasting).

Caption: variable a is grayscale image, variable b is upside down version. Average them and plot it



5. Add a random value between [0,255] to every pixel in the grayscale image, then clip the new image to have a minimum value of 0 and a maximum value of 255.

Caption: Get the random ndarray with the same shape of grayscale image. Add it with the grayscale ndarray. Then clip it between 0 to 255 to get new ndarray j and plot j



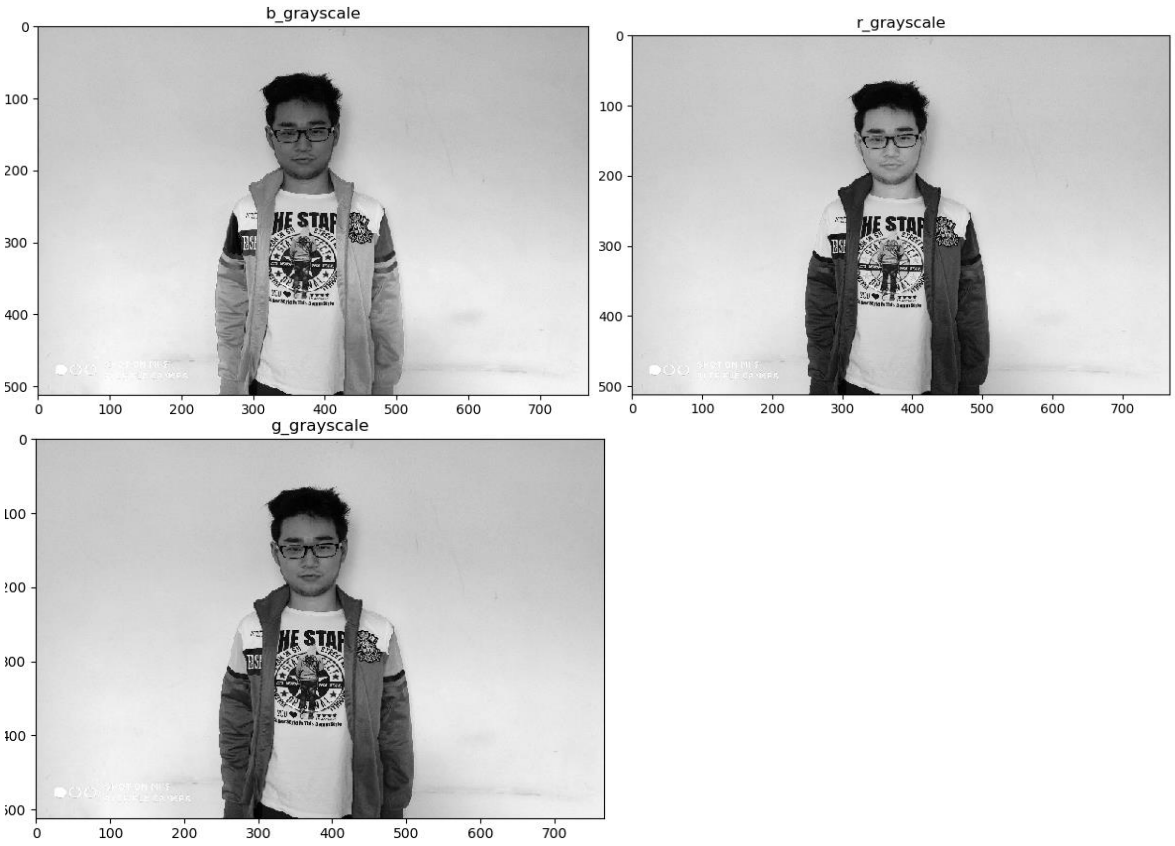
Task-3: Basic Image I/O

1. Read this face image from its JPG file, and resize the image to 768 x 512 in columns x rows

Use INTERCUBIC method to resize the original image

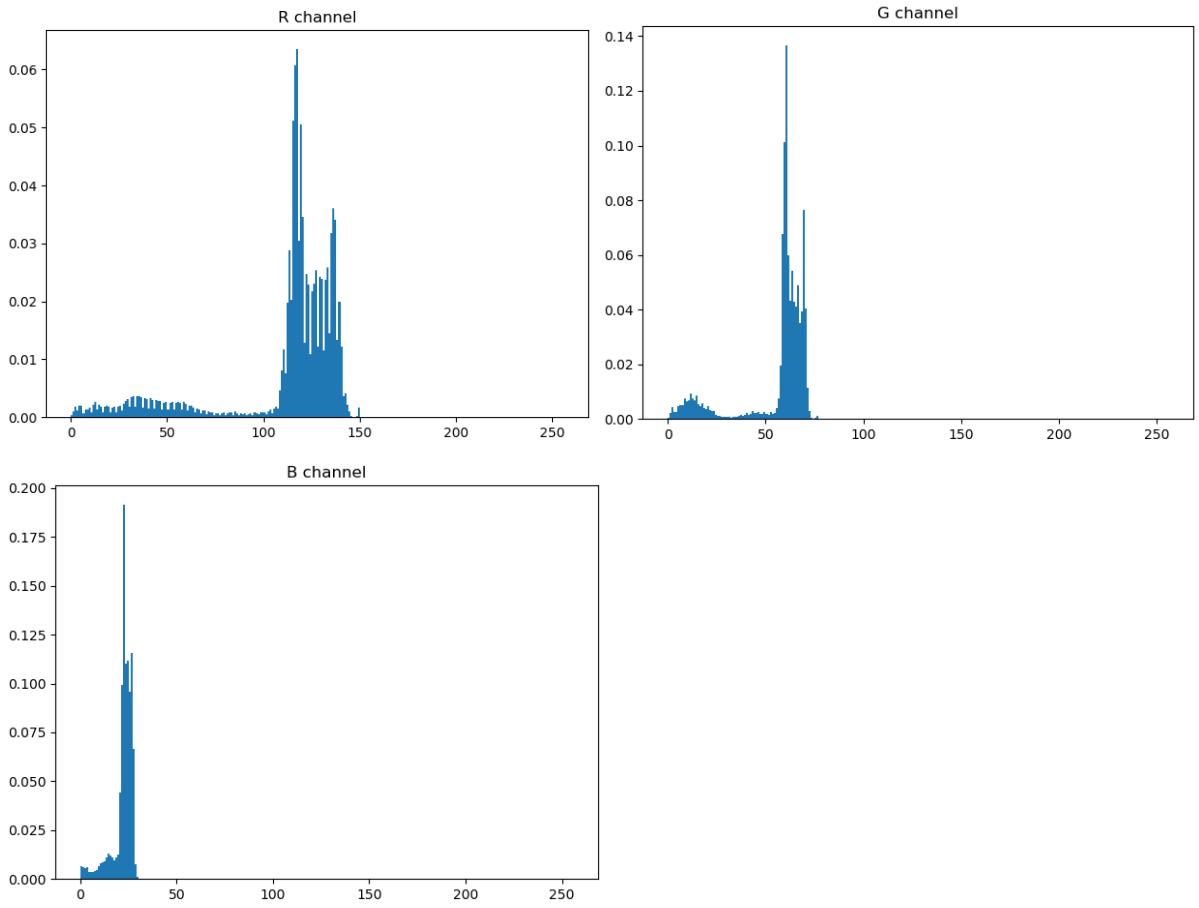
2. Convert the colour image into three grayscale channels, i.e., R, G, B images, and display each of the three channel grayscale images separately

Caption: Split the three channels from original picture, and plot them in grayscale



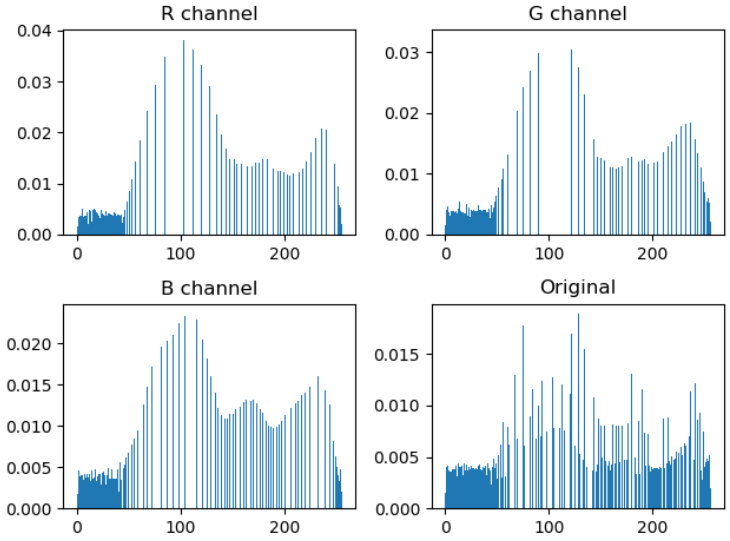
3. Compute the histograms for each of the grayscale images, and display the 3 histograms

Caption: Use hist function to draw hologram of each channel



4. Apply histogram equalisation to the resized image and its three grayscale channels, and then display the 4 histogram equalization image

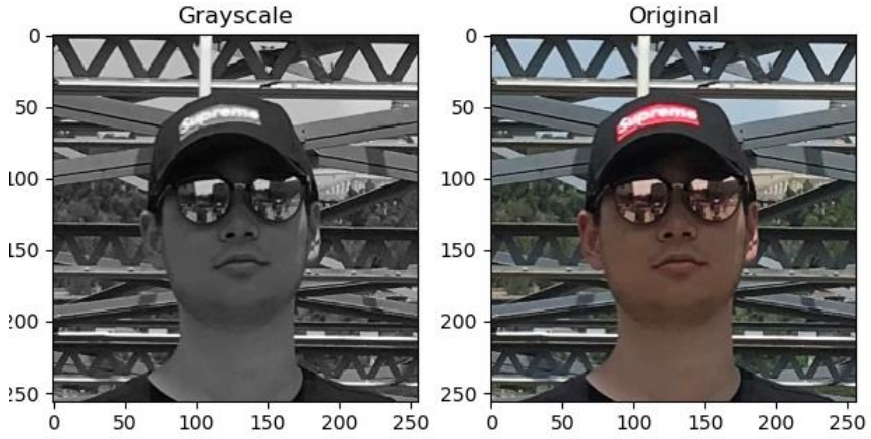
Caption: Equalize R,G,B channels and merge them to the equalization original image



Australian National University

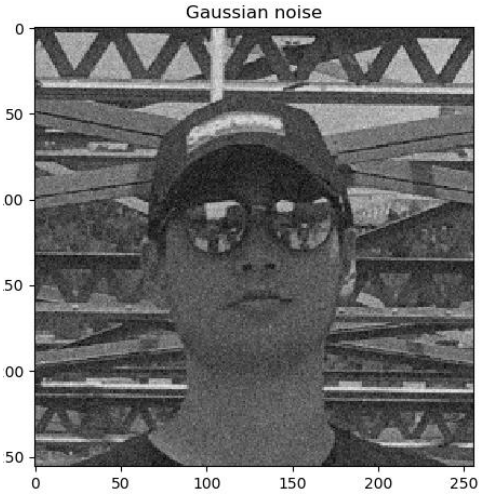
Task-4: Image Denoising via a Gaussian Filter

1. resize it to 256x256, and save this square region to a new grayscale image

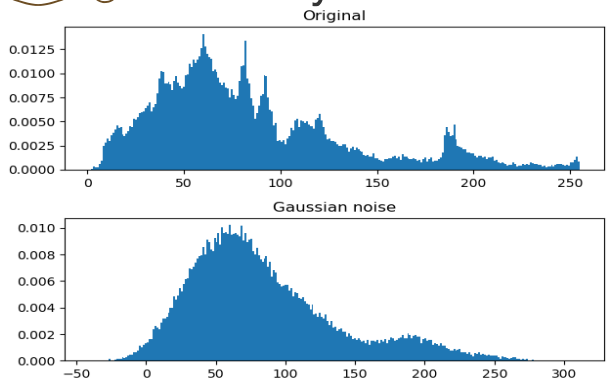


2. Add Gaussian noise to this new 256x256 image

Caption: Use randn function to create a random array of zero mean, and standard deviation of 1, then multiply by 15 to get a new random array of zero mean, and standard deviation of 15. Add this array on grayscale image.



3. Display the two histograms side by side



4. Implement your own Matlab function that performs a 5x5 Gaussian filtering

Class my_gauss_filter: input (source image, kernel size, standard deviation)

Output (destinated image)

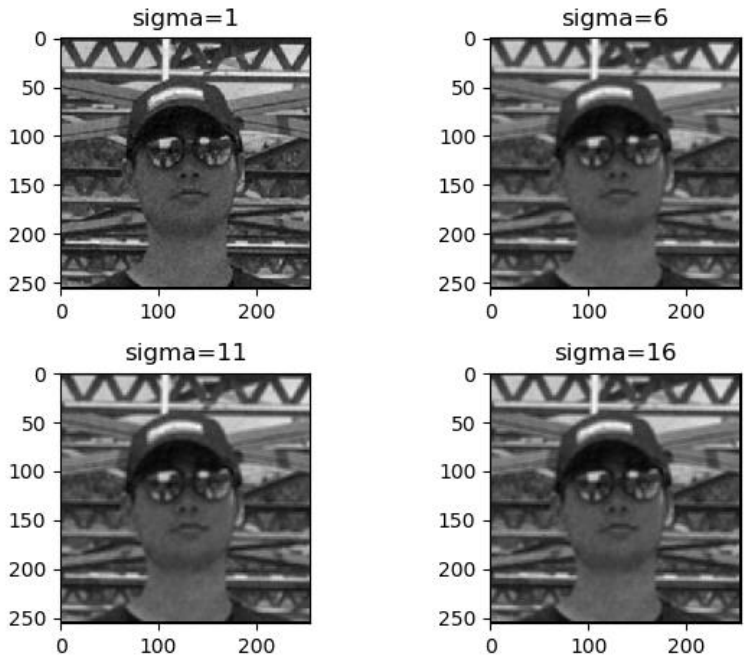
Function gauss_kernel: Apply Gaussian function to the given size kernel

Function get_size: get the size of source image and create a blank ndarray to fill in the target image

Function my_gauss_filter: convolute the filter to each pixel of source image, and put convoluted pixels to the blank ndarray

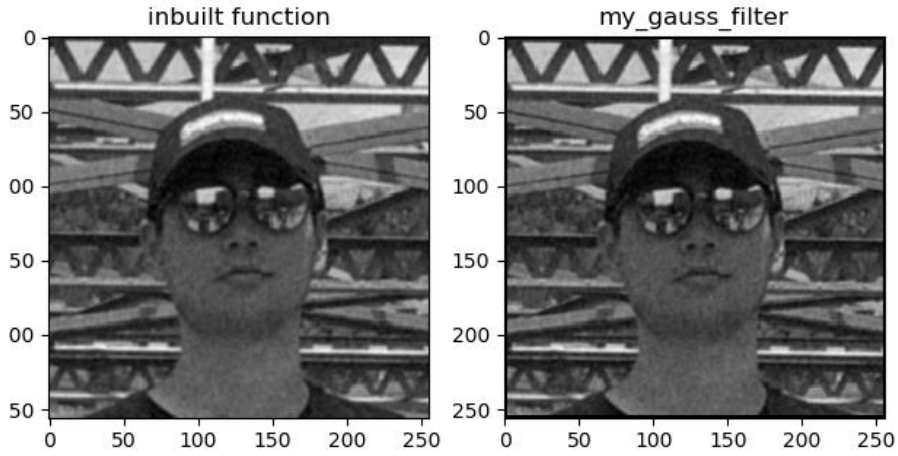
Code append on the end of report

5. Apply your Gaussian filter to the above noisy image, and display the smoothed images and visually check their noise-removal effects



Caption: As the standard deviation increase, the image becomes more blur. When standard deviation is one, the de-noise image has the best noise-removal effects.

6. Compare your result with that by Matlab's inbuilt 5x5 Gaussian filter



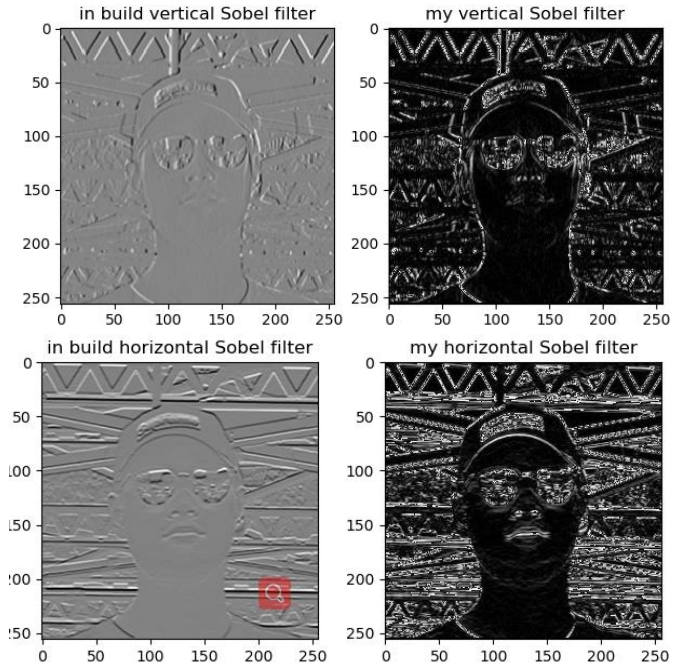
Caption: two results are nearly identical

Task -5: Implement your own 3x3 Sobel filter in Matlab

Function my_Sobel_filter: input (source image, detect vertical or horizontal edge)

Output (destinated image)

Convolute each pixel with the Sobel kernel



Caption: Compard to the build-in function Sobel, the background of sobel function image is different. The black parts of my sobel image means that the gradient on this point is nearly zero. The function of Sobel filter is to detect the edge of the image. The ensence of convolution between sobel kernal and pixcel is to calculate the gradient of light intensity change (edge). The detected Sobel image shows the gradient of light intensity change on each pixel.

Task-6: Image Rotation

1. Implement your own function my_rotation() for image rotation by any given angle between [-90, 90].

Class Rotation: Rotate image 90, 45, -15, -45, -90 degree

Input: Source image, the rotation degree

Output: destinated image

def my_rotate: convolute the rotation filter on the image, and put new coordinate position to blank image.

def demo: Execute the function

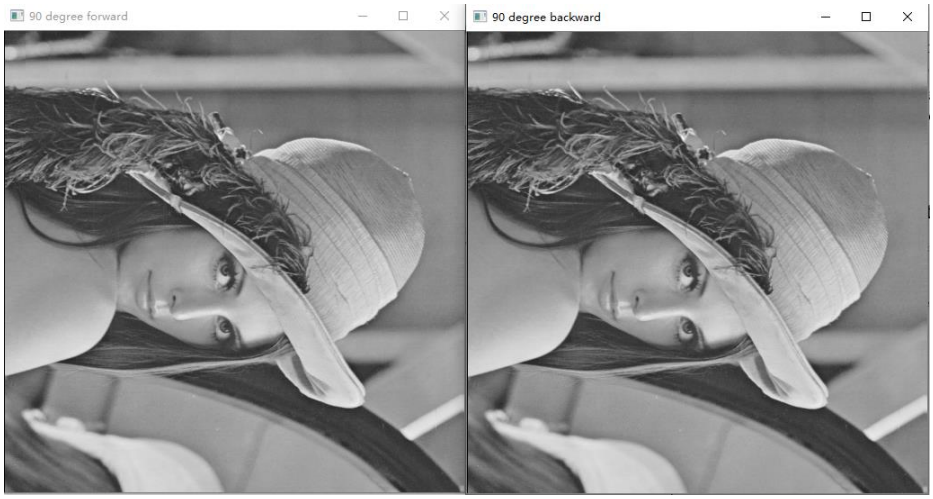


Fig. 90 degree forward mapping and inverse mapping

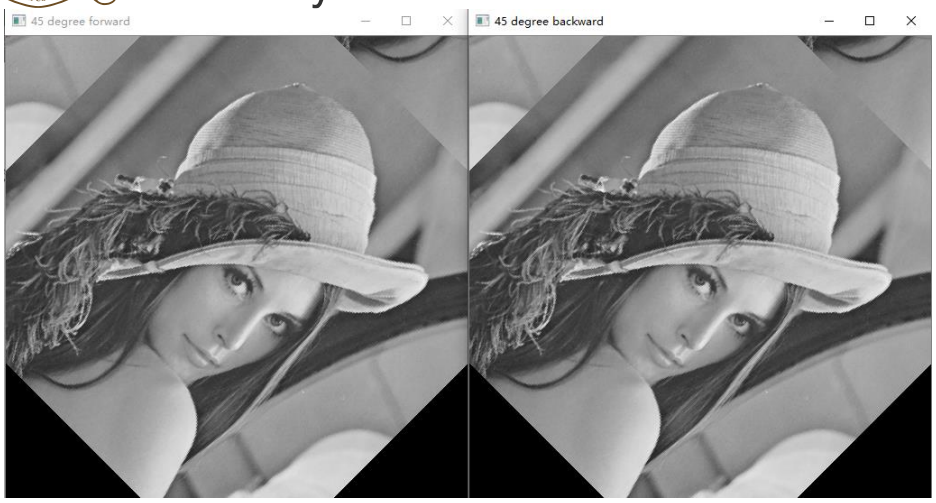


Fig. 45 degree forward mapping and inverse mapping

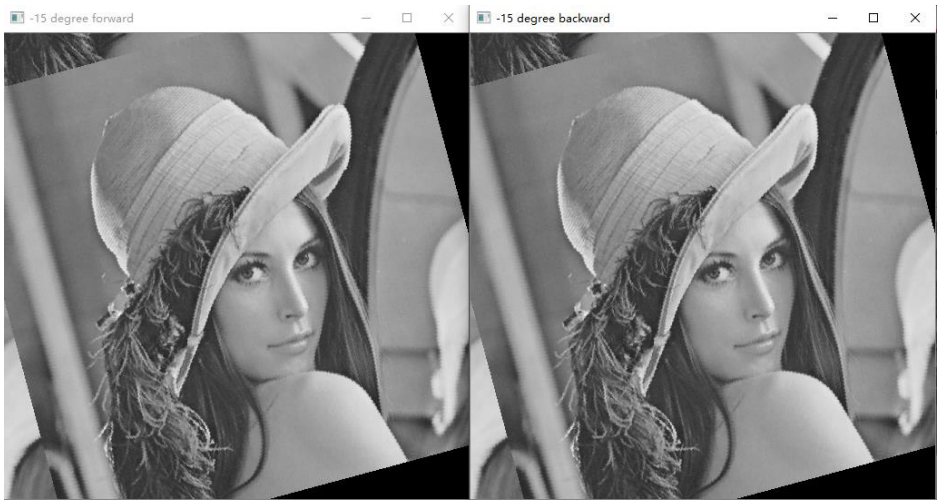


Fig. -15 degree forward mapping and inverse mapping

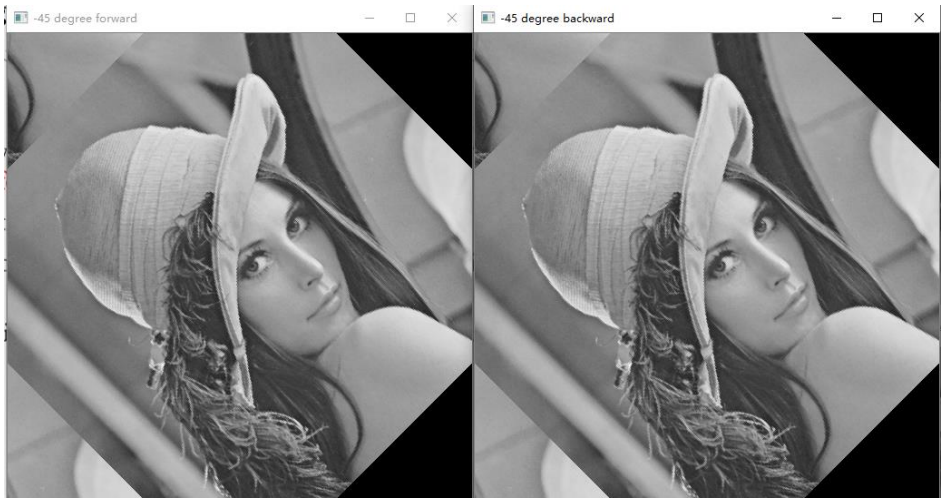


Fig. -45 degree forward mapping and inverse mapping

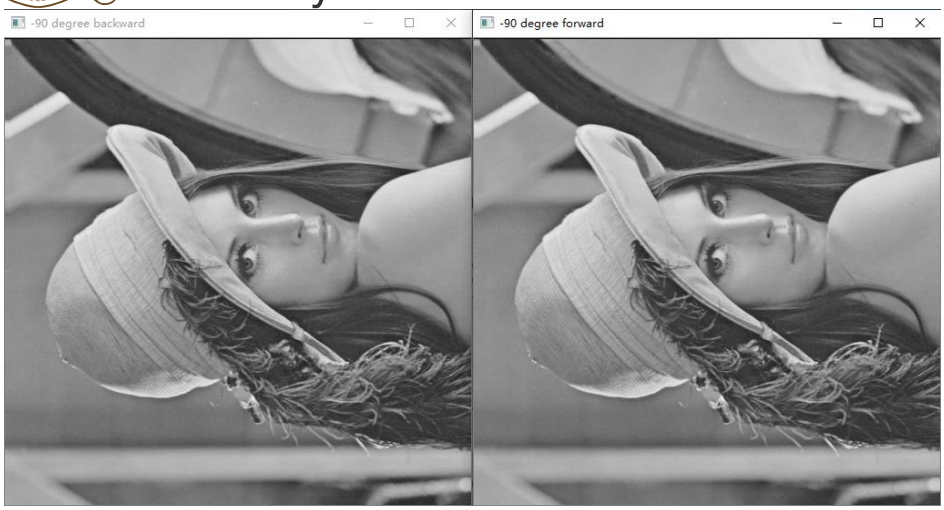


Fig. -90 degree forward mapping and inverse mapping

Code append on the end of report

2. Compare forward and backward mapping and analyze their difference

Both of mapping methods perform well on the rotation procedure.

To achieve the rotation function, the first step is to obtain the rotation matrix. For the forward mapping, the 2D rotation matrix is [[cosθ ,sinθ],[-sinθ, cosθ]]. The new u and v position are the dot product of rotation matrix and original x, y position. Finally, fill the new u and v position to the blank image. For the inverse mapping, the 2D rotation matrix is the inverse matrix of forward mapping. The destined image coordinate position u and v multiply the rotation matrix, we can get the corresponding original x, y position. Then copy the pixel from original coordinate to u and v to get destined image. In conclusion, forward warping iterates over source and send pixels to destination, while inverse warping iterates over destination and copy the pixels from source.

The backward of forward mapping is some destined pixels do not have corresponding pixels in source image, so we need to design interpolation methods to splat the holes. There are no holes happened in inverse warping, which is an advantage. But the inverse warping needs to calculate the inverse matrix.

The sample image is small (512 * 512) and the calculation of the inverse matrix is simple, so both of mapping methods are good for it.

3. Compare different interpolation methods and analyze their difference

Matlab provides three types of interpolation methods.

Nearest-neighbor interpolation: The output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered (Mathworks help center R2020a). This method requires less resource and may casue some inaccuracies.

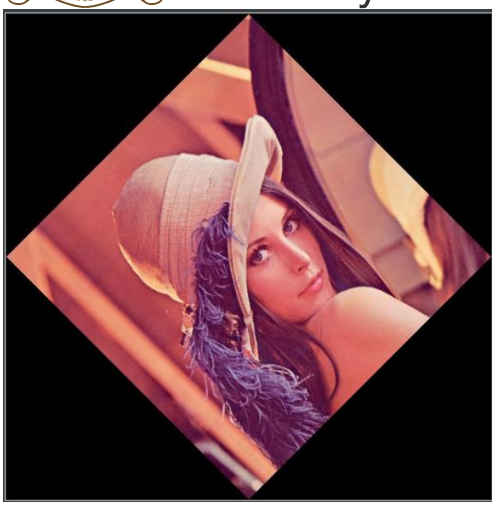


fig: Nearest-neighbor interpolation image

Bilinear interpolation: The output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood. (Mathworks help center R2020a). It covers more pixels into the calculation of rotation. Compared to Nearest-neighbor interpolation, it requires more resource but get more accurate result.

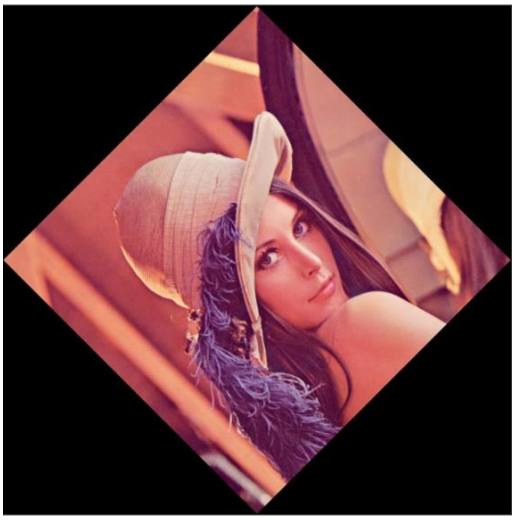


fig: Bilinear interpolation image

Bicubic interpolation: The output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood. (Mathworks help center R2020a). This method involves more pixels in calculation and more accurate than Bilinear interpolation.

Australian National University

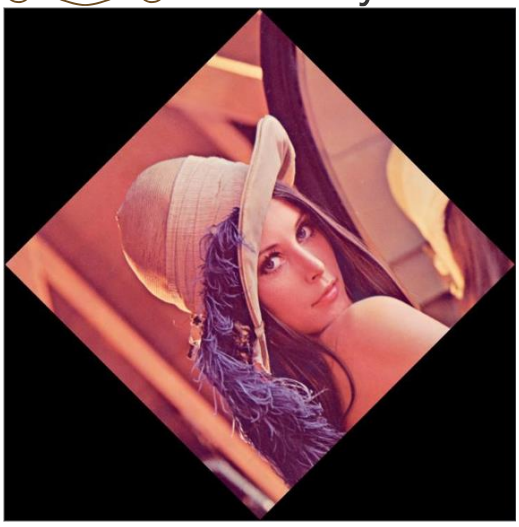


fig: Bicubic interpolation image

Code:

Task 4 4.

```
class my_gauss_filter:
    def __init__(self, src, size, sigma=1.0):
        self.size = size  # create attributes and methods
        self.src = src
        self.sigma = sigma
        self.h, self.w, self.sum = 0, 0, 0
        self.kernel = np.zeros((size, size), np.float32)
        self.dst = np.array([0])
        self.gauss_kernel()
        self.get_size()
        self.my_gauss_filter()

    def gauss_kernel(self):
        for i in range(self.size):
            for j in range(self.size):
                norm = math.pow(i - 1, 2) + pow(j - 1, 2)
                self.kernel[i, j] = math.exp(-norm / (2 * math.pow(self.sigma, 2)))  # Gaussian function
        sum = np.sum(self.kernel)  # sum the applied pixels
        self.kernel = self.kernel / sum  # divided by sum to get Gaussian filter
        return

    def get_size(self):
        self.h, self.w = self.src.shape[0], self.src.shape[1]  # get the width and height of source img
        self.dst = np.zeros((self.h, self.w))  # create a blank ndarray
        return
```

```
    def my_gauss_filter(self):
        for i in range(self.h - 4):
            for j in range(self.w - 4):
                self.sum = 0
                for k in range(5):
                    for l in range(5):
                        self.sum += self.src[i + k, j + l] * self.kernel[k, l]  # Convolution
                self.dst[i + 2, j + 2] = self.sum  # put convoluted pixels to the blank ndarray
        return plt.imshow(self.dst, cmap='gray')  # plot de-noised img
```

Task 5

```
def my_Sobel_filter(src, op):
    Mx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  # sobel filter in x axis
    My = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])  # sobel filter in y axis
    dst = np.zeros_like(src)
    for i in range(254):
        for j in range(254):
            my_sum = 0
            for k in range(3):
                for l in range(3):
                    if op == 'vertical':
                        my_sum += (src[i + k, j + l] * Mx[k, l])  # Convolution
                    else:
                        my_sum += (src[i + k, j + l] * My[k, l])
            dst[i + 1, j + 1] = abs(my_sum)  # put convoluted pixels to the blank ndarray
    return plt.imshow(dst, cmap='gray')  # plot
```

Task 6.1

```
class Rotation:
    def __init__(self, src, theta):
        self.src = src
        self.theta = theta
        self.x0 = 256
        self.y0 = 256  # center of the rotation
        self.row, self.col = self.src.shape  # the height and width of img
        self.For = np.zeros((self.row, self.col), dtype="uint8")  # blank img for forward mapping
        self.Back = np.zeros((self.row, self.col), dtype="uint8")  # blank img for inverse mapping
        self.demo()

    def my_rotate(self):
        for i in range(self.row):
            for j in range(self.col):
```

```
            # the original position multiply the rotation matrix, and add the position of  center to adjust position on new img
            u = (i - self.x0) * math.cos(self.theta * math.pi / 180) + (j - self.y0) * -math.sin(
                self.theta * math.pi / 180) + self.x0
            v = (i - self.x0) * math.sin(self.theta * math.pi / 180) + (j - self.y0) * math.cos(
                self.theta * math.pi / 180) + self.y0
            # if there are no corresponding position on new coordinate, copy the nearliest pixels
            u, v = int(u), int(v)
            # fill in the blank img
            if u < self.row and v < self.col:
                self.For[i, j] = self.src[u, v]
            # because the inverse mapping change the destined position to original position, use negative angle to instead angle
            self.theta_r = -self.theta
            # the original position multiply the rotation matrix, and add the position of  center to adjust position on new img
            x = (i - self.x0) * math.cos(self.theta_r * (math.pi / 180)) + (j - self.y0) * math.sin(
                self.theta_r * (math.pi / 180)) + self.x0
            y = (i - self.x0) * -math.sin(self.theta_r * (math.pi / 180)) + (j - self.y0) * math.cos(
                self.theta_r * (math.pi / 180)) + self.y0
            x, y = int(x), int(y)
            if x < self.row and y < self.col:  # delete the new position which is out of the blank img
                self.Back[i, j] = self.src[x, y]
                continue

    def demo(self):
        self.my_rotate()  # execute the above function
        cv2.imshow(str(self.theta) + ' degree forward', self.For)
        cv2.imshow(str(self.theta) + ' degree backward', self.Back)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

Reference:

"Imrotate." Rotate Image - MATLAB Imrotate - MathWorks , 2020, ww2.mathworks.cn/help/images/ref/imrotate.html.