Basics_of_Image_Processing_TutorialDL_final

August 15, 2023

This exercise aims to make you comfortable with the basic image processing tools and libraries. This exercise will serve as a starting point before you dive deep into the course.

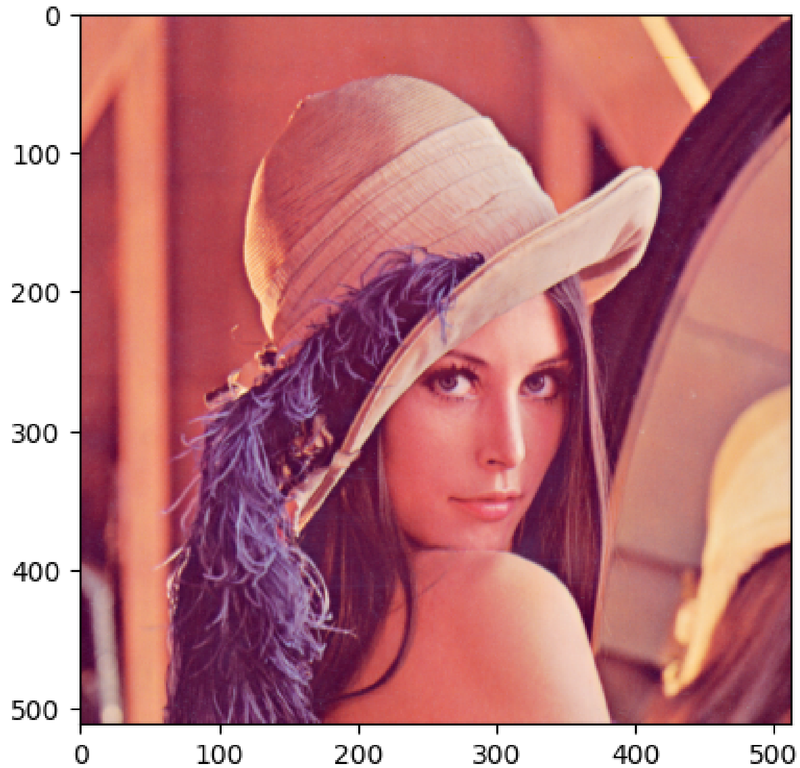
0.0.1 Let's first import basic image processing or related libraries.

```
[ ]: import numpy as np                  # numpy library useful for most of the␣
     ↪mathematical operations
     import matplotlib.pyplot as plt    # useful for data visualization/plotting␣
     ↪purpose. Can also be used for image visualization.

     # For this exercise, we will restrict ourselves to matplotlib only. Please note␣
     ↪that other libraries such as PIL, OpenCV
     # can also be used as image processing libraries.
```

First load an image and visualize it.

```
[ ]: image = plt.imread("lena.png")
     plt.imshow(image)
```

```
[ ]: <matplotlib.image.AxesImage at 0x7d77d44f8910>
```



1 1. Image Information

It is always good to know basic image details, such as its dimensions, before one proceeds for the experiments.

Task1.1 : write code to find image dimension and print it

[ ]:
```
print('The dimensions of the image are: ', image.shape)
print('Image datatype: ', image.dtype)
img_max = image.max()
img_min = image.min()
print(f'The range of values are from {img_min} to {img_max}')
```

```
The dimensions of the image are:  (512, 512, 3)
Image datatype:  float32
The range of values are from 0.0117647061124444 to 1.0
```

Is this image RGB (no of channels?), gray or binary (intensity range?)? What can you say about aspect ratio (defined as width/height) of this image?

1. The third value in the shape tuple is 3, meaning there are 3 channels. Hence, it is an RGB image.

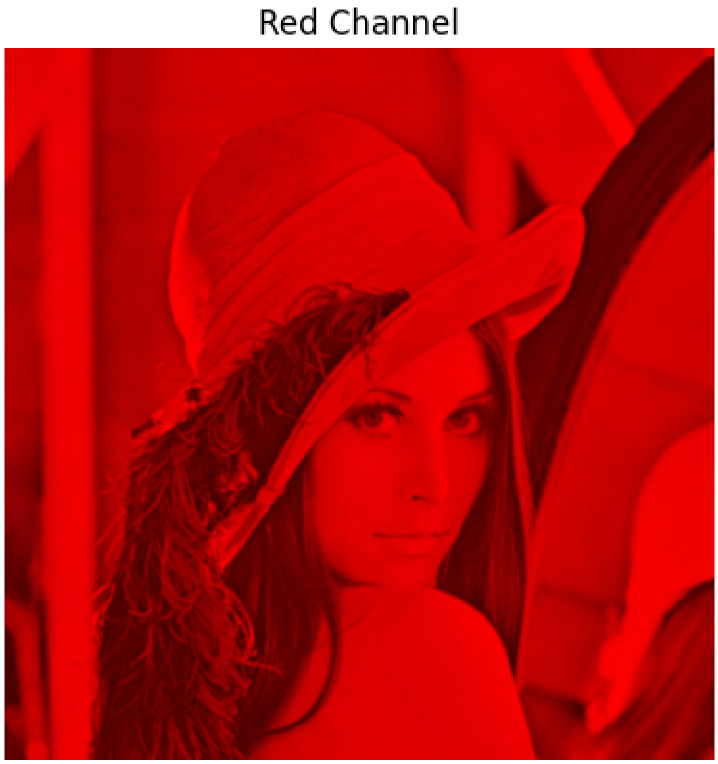
2. As it can be seen, the intensity value is of the datatype float32, hence, we expect to see intensity values in the range of 0 to 1, as it can be confirmed from the above.
3. The aspect ratio is $(512/512) = 1$. So, the image is in the shape of a square.

Task1.2: Visualization of each channel

An RGB image can be decomposed into three channels, Red(R), Green(G), Blue(B). In this subsection, let's visualize each channel separately.

[ ]:
```
def VisualizeChannel(image,channel):
    '''
    This function is helpful to visualize a specific channel of an RGB image.
    image: RGB image
    channel: channel, one wish to visualize (can take value 0 (for red),␣
 ↪1(green), 2(blue))
    '''
    #write your code here
    output = np.zeros_like(image)
    output[:, :, channel] = image[:, :, channel]
    channel_colour = ['Red', 'Green', 'Blue'][channel]
    plt.imshow(output)
    plt.title(f'{channel_colour} Channel')
    plt.axis('off')
    return output[:, :, channel]    # 'output' is image's particular channel␣
 ↪values
```

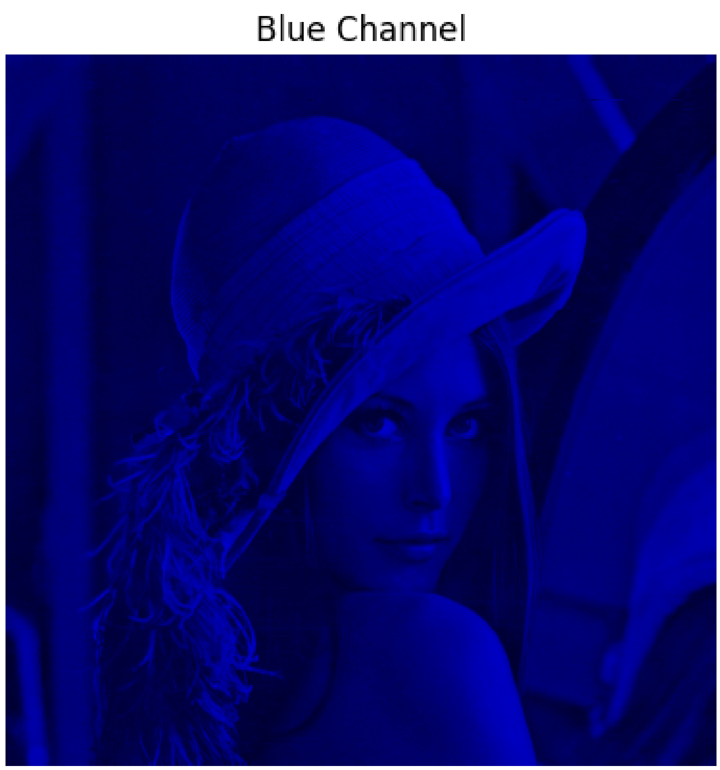
[ ]:
```
red_channel = VisualizeChannel(image, 0)
```



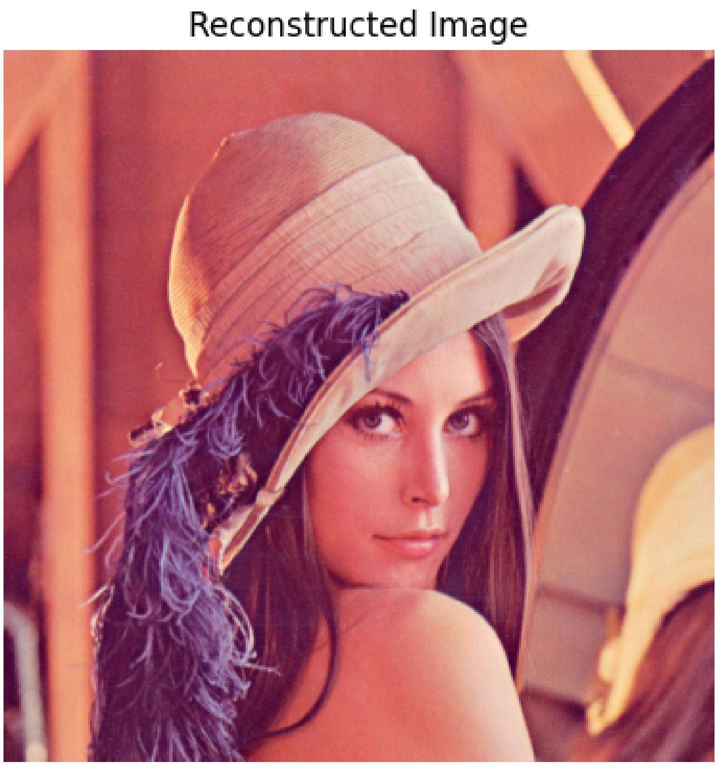

```
[ ]: green_channel = VisualizeChannel(image, 1)
```

Green Channel

```
[ ]: blue_channel = VisualizeChannel(image, 2)
```



[ ]:
```
reconstructed_image = np.stack((red_channel, green_channel, blue_channel),␣
 ↪axis=-1)
plt.imshow(reconstructed_image)
plt.title('Reconstructed Image')
plt.axis('off')
plt.show()
```



[ ]:
```python
assert reconstructed_image.all() == image.all()
```

Can you also comment on the maximum and minimum intensity values of each channel? What can you say about the range of intensity values?

[ ]:
```python
min_red, max_red = np.min(red_channel), np.max(red_channel)
min_green, max_green = np.min(green_channel), np.max(green_channel)
min_blue, max_blue = np.min(blue_channel), np.max(blue_channel)

print("Red Channel:   Min =", min_red, " Max =", max_red)
print("Green Channel: Min =", min_green, " Max =", max_green)
print("Blue Channel:  Min =", min_blue, " Max =", max_blue)
```

```
Red Channel:   Min = 0.21176471  Max = 1.0
Green Channel: Min = 0.011764706  Max = 0.972549
Blue Channel:  Min = 0.03137255  Max = 0.88235295
```

1. Red Channel: The minimum intensity is relatively high, and the maximum intensity reaches the full scale of 1.0. This suggests that the red channel contains bright red regions with a substantial contribution to the overall brightness of the image.
2. Green Channel: The green channel has a broader range, from very low intensity to almost the maximum possible intensity.
3. Blue Channel: Its maximum intensity does not reach the full scale. This could indicate a lack

of extremely bright blue regions in the image.

2 2. Intensity Manipulations

Task2.1: RGB to Gray

We may need a gray image for some of our applications. One can also convert RGB to gray to reduce computational complexity. For this part, we will convert an RGB image to grayscale. Refer this link for explanation: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

[ ]:
```
def RGB2Gray(image, visualize = False):
    '''
    This function converts an RGB image to grayscale
    image: RGB image
    '''
    #write you code here and visualize the result
    gray = 0.3*image[:, :, 0] + 0.59*image[:, :, 1] + 0.11*image[:, :, 2]
    if visualize == True:
        plt.imshow(gray, cmap='gray')
        plt.title('Grayscale Image')
        plt.axis('off')
        plt.show()

    return gray       #'gray' is grayscale image, converted from RGB image
```

[ ]:
```
gray_image = RGB2Gray(image, visualize = True)
```

Grayscale Image

We can also convert a gray image to a binary image. For task2.2, consider a gray image as input (you may take the output from task2.1 as input).

Write code to threshold a gray image such that

I(x,y) = 1 if I(x,y) >= T = 0 if I(x,y) < T where T is threshold

Though there are proper methods(such as the Otsu method) to find a suitable T, we will not go into details of those algorithms and randomly select T values and visualize the result.

Task2.2 : Gray to Binary

Before you proceed to code, Can you comment on the valid range of T? (Hint:

Task1.2)

Red Channel: Min = 0.21176471 Max = 1.0 Green Channel: Min = 0.011764706 Max = 0.972549 Blue Channel: Min = 0.03137255 Max = 0.88235295

The grayscale conversion using the weighted sum method will result in intensity values that fall within a range determined by these values. So I guess the minimum grayscale intensity will correspond to the weighted sum of the minimum RGB values, and the maximum grayscale intensity will correspond to the weighted sum of the maximum RGB values.

Tmin = 0.3 X 0.21176471 + 0.59 X 0.011764706 + 0.11 X 0.03137255 = 0.0739

Similarly, Tmax = 0.9708

SO, the range is 0.0739 to 0.9708

[ ]:
```python
def Gray2Binary(image,T):
    '''
    This function converts a gray image to binary based on the rule stated␣
↪above.
    image: image (can be RGB or gray); if the image is RGB, convert it to gray␣
↪first
    T: Threshold
    '''

    #check if image is RGB if yes, convert it to gray
    flag = len(image.shape)
    if flag == 3:        #i.e. RGB image, hence to be converted to gray
        # write code to convert it to gray or you can call function "RGB2Gray"␣
↪defined in task2.1
        gray = RGB2Gray(image)
    else:
        gray = image

    #Write code to threshold image based on the rule stated above and return␣
↪this binarized image (say it 'bimage')
    bimg = np.where(gray >= T, 1, 0)

    #write code to visualize the resultant image
    plt.imshow(bimg, cmap='gray')
    plt.title(f'Binary Image with Threshold {T}')
    plt.axis('off')
    plt.show()


    return bimg
```

[ ]:
```python
T_mean = np.mean(gray_image)
mid_T = Gray2Binary(image, T_mean)
```

An image is nothing but a matrix. Hence one can perform all kinds of mathematical operations on an image just like a matrix.

To convince ourselves with the above statement, let's crop a section of a gray image, print its value, and perform some mathematical operations. For a better data display, we will cut only 5*5 areas of the gray image.

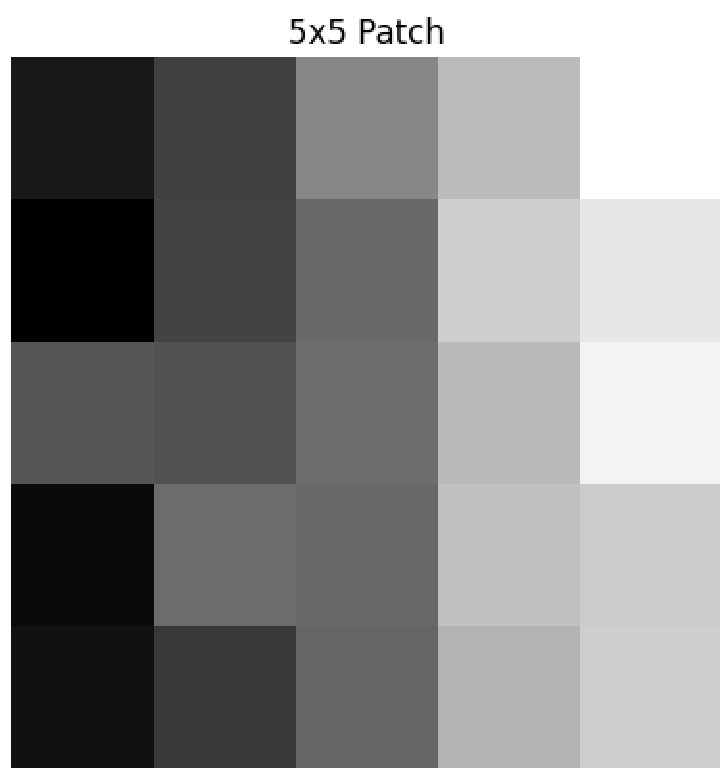
Task2.3: Crop a 5*5 section of a gray image

[ ]:
```python
def ImageCrop(image,r0,c0):
    '''
    This function crops 5*5 rectangular patch defined by image coordinates(r0,c0),(r0,c0+5),(r0+5,c0) and (r0+5,c0+5)
    of an image.
    image: Image can be RGB or gray
    r0: starting row index
    c0: starting column index
    '''
    # write code to check if input is RGB , if its RGB convert it to gray
    if len(image.shape) == 3:
        gray = RGB2Gray(image)
    else:
        gray = image
```

```
    #print(gray.shape)

    # write code to select 5*5 rectangular patch defined as above (say it␣
↪'patch')
    patch = gray[r0:r0 + 5, c0:c0 + 5]

    # visualize patch and print its value
    plt.imshow(patch, cmap='gray')
    plt.title('5x5 Patch')
    plt.axis('off')
    plt.show()

    return patch
```

[ ]:

```
r0, c0 = 96, 24
patch = ImageCrop(image,r0,c0)
print(patch)
#plt.imshow(patch)
```




```
[[0.37031376 0.39443138 0.43745098 0.468902   0.5105882 ]
 [0.3551765  0.3964314  0.41909808 0.48058823 0.49580395]
 [0.40705884 0.40525493 0.42160785 0.46850982 0.5032157 ]
```

```
 [0.36125493 0.42113727 0.41870588 0.47258824 0.4789412 ]
 [0.3657255  0.3894902  0.41682354 0.4649412  0.48054904]]
```

Now you have 5*5 patch and you know its values too. Can you try

1. multiplying patch by 0.5
2. multiplying patch by 2
3. create another random 5*5 patch (numpy array) and add/subtract it to the patch

Does it follow matrix addition/subtraction and multiplication rules? You can also play around with other matrix operations.

[ ]:
```python
patch_down = patch*0.5
print(patch_down)
```

```
[[0.18515688 0.19721569 0.21872549 0.234451   0.2552941 ]
 [0.17758825 0.1982157  0.20954904 0.24029411 0.24790198]
 [0.20352942 0.20262747 0.21080393 0.23425491 0.25160784]
 [0.18062747 0.21056864 0.20935294 0.23629412 0.2394706 ]
 [0.18286274 0.1947451  0.20841177 0.2324706  0.24027452]]
```

[ ]:
```python
patch_up = patch*2
print(patch_up)
```

```
[[0.7406275  0.78886276 0.87490195 0.937804   1.0211765 ]
 [0.710353   0.7928628  0.83819616 0.96117646 0.9916079 ]
 [0.81411767 0.81050986 0.8432157  0.93701965 1.0064313 ]
 [0.72250986 0.84227455 0.83741176 0.9451765  0.9578824 ]
 [0.731451   0.7789804  0.8336471  0.9298824  0.9610981 ]]
```

[ ]:
```python
rand_patch = np.random.rand(5,5)
patch_sub = patch - rand_patch
patch_add = patch + rand_patch
print(patch_sub)
print(patch_add)
```

```
[[ 0.00210415 -0.03195791 -0.49216814  0.32726342  0.26973723]
 [-0.37337913  0.22124331 -0.09804299  0.02592856  0.18880235]
 [ 0.20620015 -0.27454179 -0.0239253   0.4588828   0.02857142]
 [ 0.09987752  0.06261381  0.32135561 -0.03775434 -0.33751155]
 [ 0.00485287 -0.09431438 -0.48611443  0.0631175  -0.41739595]]
[[0.73852338 0.82082068 1.36707009 0.61054057 0.75143923]
 [1.08373215 0.57161947 0.93623915 0.93524789 0.80280556]
 [0.60791752 1.08505165 0.867141   0.47813685 0.97785993]
 [0.62263235 0.77966073 0.51605616 0.98293082 1.29539396]
 [0.7265981  0.87329475 1.31976151 0.86676491 1.37849402]]
```

[ ]:
```python
patch_mult = patch@rand_patch
print(patch_mult)
```

```
[[0.81840183 0.93951252 1.24980357 0.68046121 1.25923157]
 [0.8083182  0.91796938 1.21637752 0.67907314 1.24375444]
 [0.83387198 0.9425988  1.27580601 0.68727163 1.25694431]
 [0.82030085 0.91359626 1.21862501 0.68030449 1.2309436 ]
 [0.79709366 0.90671489 1.2062836  0.66327437 1.21661149]]
```

Seems to follow matrix operations!

Task2.4: Uniform Brightness Scaling

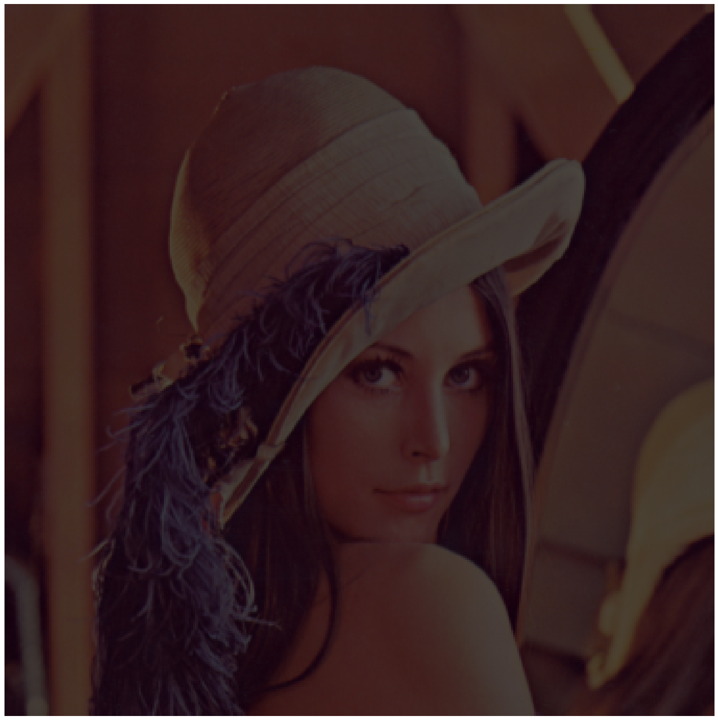
2.0.1 Hopefully, you are convinced that an image is a matrix. Hence we can perform multiplication/division or addition/subtraction operations. These operations will change the brightness value of the image; can make an image brighter or darker depending on the multiplying/scaling factor. For this task, let's change the image brightness uniformly.

Consider scale to be 0.3,0.5,1,2 for four different cases. What is your observation?

[ ]:
```python
def UniformBrightScaling(image,scale):
    '''
    This function uniformly increases or decreases the pixel values (of all␣
 ↪image locations) by a factor 'scale'.
    image: image (can be RGB or gray image)
    scale: A scalar by which pixels'svalues need to be multiplied
    '''
    #write your code here
    output = image * scale
    output = np.clip(output, 0, 1) #maybe commented out?


    #display the resultant image
    plt.imshow(output, cmap='gray' if len(output.shape) == 2 else None)
    plt.title(f'Image with Brightness Scale {scale}')
    plt.axis('off')
    plt.show()

    return output        #replace output with the variable name you used for␣
 ↪final result
```

[ ]:
```python
UniformBrightScaling(image, 0.3)
```

Image with Brightness Scale 0.3



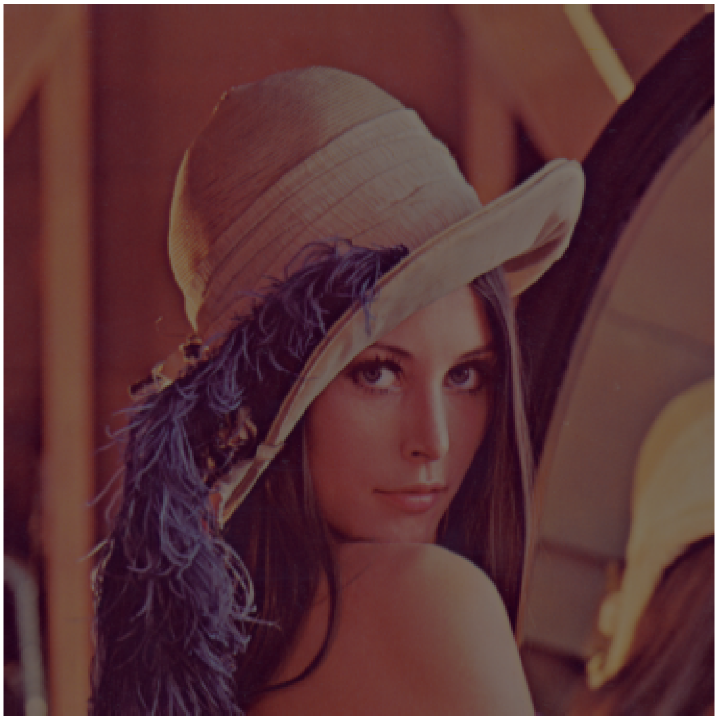
```
[ ]: array([[[0.26588237, 0.16117649, 0.14705883],
            [0.26588237, 0.16117649, 0.14705883],
            [0.26235294, 0.16117649, 0.1564706 ],
            …,
            [0.27058825, 0.17411765, 0.14352942],
            [0.26000002, 0.1529412 , 0.12941177],
            [0.23529413, 0.1164706 , 0.10588236]],

           [[0.26588237, 0.16117649, 0.14705883],
            [0.26588237, 0.16117649, 0.14705883],
            [0.26235294, 0.16117649, 0.1564706 ],
            …,
            [0.27058825, 0.17411765, 0.14352942],
            [0.26000002, 0.1529412 , 0.12941177],
            [0.23529413, 0.1164706 , 0.10588236]],

           [[0.26588237, 0.16117649, 0.14705883],
            [0.26588237, 0.16117649, 0.14705883],
            [0.26235294, 0.16117649, 0.1564706 ],
            …,
            [0.27058825, 0.17411765, 0.14352942],
            [0.26000002, 0.1529412 , 0.12941177],
```

```
 [0.23529413, 0.1164706 , 0.10588236]],

...,

[[0.09882354, 0.02117647, 0.07058824],
 [0.09882354, 0.02117647, 0.07058824],
 [0.1082353 , 0.03176471, 0.0682353 ],
 ...,
 [0.20352943, 0.08588236, 0.09882354],
 [0.20235296, 0.08000001, 0.08941177],
 [0.20823531, 0.07294118, 0.09294118]],

[[0.09647059, 0.02588235, 0.06705882],
 [0.09647059, 0.02588235, 0.06705882],
 [0.11294118, 0.03764706, 0.07294118],
 ...,
 [0.21058825, 0.08235295, 0.09294118],
 [0.21294118, 0.08352942, 0.09529413],
 [0.21764708, 0.08705883, 0.09529413]],

[[0.09647059, 0.02588235, 0.06705882],
 [0.09647059, 0.02588235, 0.06705882],
 [0.11294118, 0.03764706, 0.07294118],
 ...,
 [0.21058825, 0.08235295, 0.09294118],
 [0.21294118, 0.08352942, 0.09529413],
 [0.21764708, 0.08705883, 0.09529413]]], dtype=float32)
```

[ ]:

```
UniformBrightScaling(image, 0.5)
```

Image with Brightness Scale 0.5



```
[ ]: array([[[0.44313726, 0.26862746, 0.24509804],
         [0.44313726, 0.26862746, 0.24509804],
         [0.4372549 , 0.26862746, 0.26078433],
         …,
         [0.4509804 , 0.2901961 , 0.23921569],
         [0.43333334, 0.25490198, 0.21568628],
         [0.39215687, 0.19411765, 0.1764706 ]],

        [[0.44313726, 0.26862746, 0.24509804],
         [0.44313726, 0.26862746, 0.24509804],
         [0.4372549 , 0.26862746, 0.26078433],
         …,
         [0.4509804 , 0.2901961 , 0.23921569],
         [0.43333334, 0.25490198, 0.21568628],
         [0.39215687, 0.19411765, 0.1764706 ]],

        [[0.44313726, 0.26862746, 0.24509804],
         [0.44313726, 0.26862746, 0.24509804],
         [0.4372549 , 0.26862746, 0.26078433],
         …,
         [0.4509804 , 0.2901961 , 0.23921569],
         [0.43333334, 0.25490198, 0.21568628],
```

```
 [0.39215687, 0.19411765, 0.1764706 ]],

...,

[[0.16470589, 0.03529412, 0.11764706],
 [0.16470589, 0.03529412, 0.11764706],
 [0.18039216, 0.05294118, 0.11372549],
 ...,
 [0.3392157 , 0.14313726, 0.16470589],
 [0.3372549 , 0.13333334, 0.14901961],
 [0.34705883, 0.12156863, 0.15490197]],

[[0.16078432, 0.04313726, 0.11176471],
 [0.16078432, 0.04313726, 0.11176471],
 [0.1882353 , 0.0627451 , 0.12156863],
 ...,
 [0.3509804 , 0.13725491, 0.15490197],
 [0.35490197, 0.1392157 , 0.15882353],
 [0.3627451 , 0.14509805, 0.15882353]],

[[0.16078432, 0.04313726, 0.11176471],
 [0.16078432, 0.04313726, 0.11176471],
 [0.1882353 , 0.0627451 , 0.12156863],
 ...,
 [0.3509804 , 0.13725491, 0.15490197],
 [0.35490197, 0.1392157 , 0.15882353],
 [0.3627451 , 0.14509805, 0.15882353]]], dtype=float32)
```

[ ]:
```
UniformBrightScaling(image, 1)
```

Image with Brightness Scale 1

```
[ ]: array([[[0.8862745 , 0.5372549 , 0.49019608],
            [0.8862745 , 0.5372549 , 0.49019608],
            [0.8745098 , 0.5372549 , 0.52156866],
            …,
            [0.9019608 , 0.5803922 , 0.47843137],
            [0.8666667 , 0.50980395, 0.43137255],
            [0.78431374, 0.3882353 , 0.3529412 ]],

           [[0.8862745 , 0.5372549 , 0.49019608],
            [0.8862745 , 0.5372549 , 0.49019608],
            [0.8745098 , 0.5372549 , 0.52156866],
            …,
            [0.9019608 , 0.5803922 , 0.47843137],
            [0.8666667 , 0.50980395, 0.43137255],
            [0.78431374, 0.3882353 , 0.3529412 ]],

           [[0.8862745 , 0.5372549 , 0.49019608],
            [0.8862745 , 0.5372549 , 0.49019608],
            [0.8745098 , 0.5372549 , 0.52156866],
            …,
            [0.9019608 , 0.5803922 , 0.47843137],
            [0.8666667 , 0.50980395, 0.43137255],
```

```
 [0.78431374, 0.3882353 , 0.3529412 ]],

...,

[[0.32941177, 0.07058824, 0.23529412],
 [0.32941177, 0.07058824, 0.23529412],
 [0.36078432, 0.10588235, 0.22745098],
 ...,
 [0.6784314 , 0.28627452, 0.32941177],
 [0.6745098 , 0.26666668, 0.29803923],
 [0.69411767, 0.24313726, 0.30980393]],

[[0.32156864, 0.08627451, 0.22352941],
 [0.32156864, 0.08627451, 0.22352941],
 [0.3764706 , 0.1254902 , 0.24313726],
 ...,
 [0.7019608 , 0.27450982, 0.30980393],
 [0.70980394, 0.2784314 , 0.31764707],
 [0.7254902 , 0.2901961 , 0.31764707]],

[[0.32156864, 0.08627451, 0.22352941],
 [0.32156864, 0.08627451, 0.22352941],
 [0.3764706 , 0.1254902 , 0.24313726],
 ...,
 [0.7019608 , 0.27450982, 0.30980393],
 [0.70980394, 0.2784314 , 0.31764707],
 [0.7254902 , 0.2901961 , 0.31764707]]], dtype=float32)
```

[ ]:
```
UniformBrightScaling(image, 2)
```

Image with Brightness Scale 2

```
[ ]: array([[[1.        , 1.        , 0.98039216],
            [1.        , 1.        , 0.98039216],
            [1.        , 1.        , 1.        ],
            …,
            [1.        , 1.        , 0.95686275],
            [1.        , 1.        , 0.8627451 ],
            [1.        , 0.7764706 , 0.7058824 ]],

           [[1.        , 1.        , 0.98039216],
            [1.        , 1.        , 0.98039216],
            [1.        , 1.        , 1.        ],
            …,
            [1.        , 1.        , 0.95686275],
            [1.        , 1.        , 0.8627451 ],
            [1.        , 0.7764706 , 0.7058824 ]],

           [[1.        , 1.        , 0.98039216],
            [1.        , 1.        , 0.98039216],
            [1.        , 1.        , 1.        ],
            …,
            [1.        , 1.        , 0.95686275],
            [1.        , 1.        , 0.8627451 ],
```

```
 [1.        , 0.7764706 , 0.7058824 ]],

…,

[[0.65882355, 0.14117648, 0.47058824],
 [0.65882355, 0.14117648, 0.47058824],
 [0.72156864, 0.21176471, 0.45490196],
 …,
 [1.        , 0.57254905, 0.65882355],
 [1.        , 0.53333336, 0.59607846],
 [1.        , 0.4862745 , 0.61960787]],

[[0.6431373 , 0.17254902, 0.44705883],
 [0.6431373 , 0.17254902, 0.44705883],
 [0.7529412 , 0.2509804 , 0.4862745 ],
 …,
 [1.        , 0.54901963, 0.61960787],
 [1.        , 0.5568628 , 0.63529414],
 [1.        , 0.5803922 , 0.63529414]],

[[0.6431373 , 0.17254902, 0.44705883],
 [0.6431373 , 0.17254902, 0.44705883],
 [0.7529412 , 0.2509804 , 0.4862745 ],
 …,
 [1.        , 0.54901963, 0.61960787],
 [1.        , 0.5568628 , 0.63529414],
 [1.        , 0.5803922 , 0.63529414]]], dtype=float32)
```

2.0.2 Observation: Brightness increases with increasing scaling factor

[ ]:
```
## Image normalization
```

3 3. Image Filtering

In this section, you will perform some of the image filtering techniques.

Convolution is one of the most widely used operations for images. Convolution can be used as a feature extractor; different kernel results in various types of features. Refer https://en.wikipedia.org/wiki/Kernel_(image_processing) to see few examples of kernel.

[ ]:
```
def feature_extractor(image,kernel, visualize = False):
    '''
    This function performs convolution operation to a gray image. We will␣
↪consider 3*3 kernel here.
    In general kernel can have shape (2n+1) * (2n+1)  where n>= 0
```

```python
    image: image (can be RGB or gray); if RGB convert it to gray
    kernel: 3*3 convolution kernel
    '''
    # kernel shape verification
    k_rows, k_cols = kernel.shape
    if k_rows % 2 == 0 or k_cols % 2 == 0:
        raise ValueError("Kernel must have odd dimensions.")
    # first convert RGB to gray if input is RGB image

    l = len(image.shape)

    if l == 3:
        #write code to convert it to gray scale
        if len(image.shape) == 3:
            image = RGB2Gray(image)
    # write code to create a zero array of size (r,c) which will store the resultant value at specific pixel locations (say it output)
    r, c = image.shape
    output = np.zeros((r, c))

    #write code to create a zero array with size (r+2,c+2) if (r,c) is the gray image size.  (say it pad_img)
    pad_size_r = k_rows // 2
    pad_size_c = k_cols // 2
    pad_img = np.zeros((r + 2 * pad_size_r, c + 2 * pad_size_c))
    #now copy gray image to above created array at location starting from (1,1)
    pad_img[pad_size_r:pad_size_r + r, pad_size_c:pad_size_c + c] = image


    #write code to convolve the image
    for row in range(pad_size_r, pad_size_r + r):      # use appropriate range values for row and col
        for col in range(pad_size_c, pad_size_c + c):
            # select 3*3 patch with center at (row,col), flatten it. flatten the kernel and take dot product between both (or directly take element wise multiplication and sum it)
            patch = pad_img[row - pad_size_r:row + pad_size_r + 1, col - pad_size_c:col + pad_size_c + 1]
            conv_result = np.sum(patch * kernel)
            # store this scalar value to output matrix with starting location (0,0)          (alternatively one could also create a list and reshape it to output size)
            output[row - pad_size_r, col - pad_size_c] = conv_result

    if visualize:
        plt.imshow(output, cmap='gray')
```

```
        plt.title('Convolution Result')
        plt.axis('off')
        plt.show()

    return output
```

[ ]:
```
## Note that the steps described above are to help you get started. You can
↪follow other valid steps too. Result from all
 #of the method should be the same. Pseudocode is available at: https://en.
↪wikipedia.org/wiki/Kernel_(image_processing)
```

for the above case, consider all 3 * 3 kernels from https://en.wikipedia.org/

---

wiki/Kernel_(image_processing). What was your observation with different kernels? You can also play with other kernels, take any 3*3 matrix of your choice, convolve it with a gray image and see if it extracts some image features. (You should be able to correlate your learning from this experiment during CNN lectures)

[ ]:
```
identity_kernel = np.array([[0, 0, 0],
                           [0, 1, 0],
                           [0, 0, 0]])
identity_image = feature_extractor(gray_image, identity_kernel, visualize=True)
```

Convolution Result

[ ]:
```
ridge_kernel = np.array([[-1, -1, -1],
                        [-1, 8, -1],
                        [-1, -1, -1]])
ridge_image = feature_extractor(gray_image, ridge_kernel, visualize=True)
```

Convolution Result

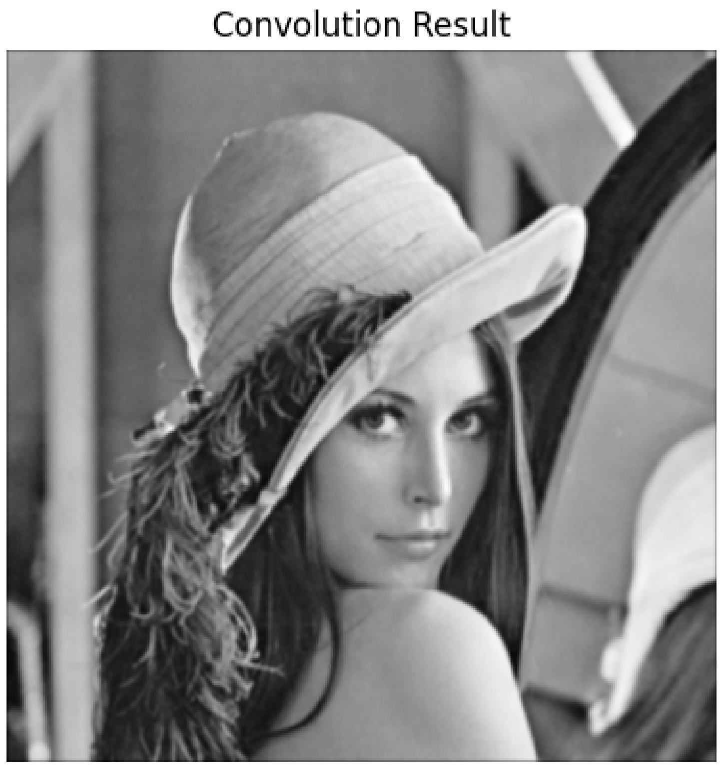
[ ]:
```
sharpen_kernel = np.array([[0, -1, 0],
                          [-1, 5, -1],
                          [0, -1, 0]])
sharpen_image = feature_extractor(gray_image, sharpen_kernel, visualize=True)
```

Convolution Result

[ ]:
```
box_blur_kernel = np.array([[1, 1, 1],
                            [1, 1, 1],
                            [1, 1, 1]]) / 9
box_blur_image = feature_extractor(gray_image, box_blur_kernel, visualize=True)
```


```
[ ]: gaussian_blur_3_kernel = np.array([[1, 2, 1],
                                        [2, 4, 2],
                                        [1, 2, 1]]) / 16
     gaussian_blur_3_image = feature_extractor(gray_image, gaussian_blur_3_kernel,␣
      ↪visualize=True)
```


```
[ ]: gaussian_blur_5_kernel = np.array([[1, 4, 6, 4, 1],
                                        [4, 16, 24, 16, 4],
                                        [6, 24, 36, 24, 6],
                                        [4, 16, 24, 16, 4],
                                        [1, 4, 6, 4, 1]]) / 256
     gaussian_blur_5_image = feature_extractor(gray_image, gaussian_blur_5_kernel, visualize=True)
```



[ ]:
```
unsharp_masking_kernel = np.array([[1, 4, 6, 4, 1],
                                   [4, 16, 24, 16, 4],
                                   [6, 24, -476, 24, 6],
                                   [4, 16, 24, 16, 4],
                                   [1, 4, 6, 4, 1]]) / (-256)
unshaarp_masking_image = feature_extractor(gray_image, unsharp_masking_kernel,␣
 ↪visualize=True)
```

3.0.1 Observations seem to be in accordance with what was seen on the wikipedia page

4 4.Geometric Transformation

Task4.1: Image Rotation (In-plane) Write a function which rotates an image by 10 degrees in anticlockwise direction. (You can use inbuit functions for this, however it is encouraged to write code from scratch)

The following code has been written assuming the the origin being at top left corner of an image, as it is usally the convention in image processing and computer vision, to my knowledge.

[ ]:
```
def rotate_image(image, angle = 10):
    theta = np.radians(angle)
    rotation_matrix = np.array([
        [np.cos(theta), -np.sin(theta)],
        [np.sin(theta), np.cos(theta)]
    ])
    Y, X = image.shape[0], image.shape[1]
    center_x = X/2-0.5
    center_y = Y/2-0.5
    rotated_image = np.zeros_like(image)
    for y in range(Y):
```

```
        for x in range(X):
            # translating the coordinates
            del_x = x-center_x
            del_y = y-center_y

            new_coordinates = rotation_matrix.dot([del_x, del_y])

            #back-translate
            new_x = int(np.round(new_coordinates[0] + center_x))
            new_y = int(np.round(new_coordinates[1] + center_y))

            #boundary condition
            if (0 <= new_x < X) and (0 <= new_y < Y):
                rotated_image[y, x] = image[new_y, new_x]

    return rotated_image
```

[ ]:

```
rotated_image1 = rotate_image(image)
plt.imshow(rotated_image1)
plt.axis('off')
plt.show()
```