Introduction:

I like watching NBA games and playing basketball very much, so I choose the famous star in the NBA as the object of image effects. I have a dream about opening an NBA coffee shop with all the NBA series products, especially the characters latte. I hope it can easily present the image of the stars.

(make by japanese laflorist Geroge)

Method:

A total of five NBA images were used, and five image processing methods were used, namely:

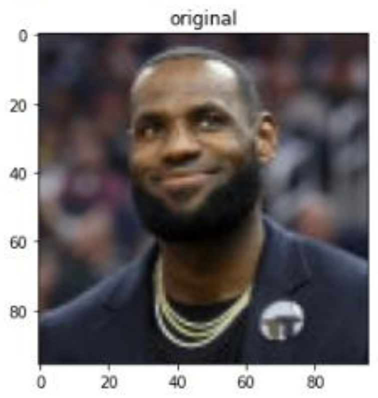
1. Image Interpolation Methods
   A. Nearest Neighbor
2. Intensity Transformations
   A. Gamma correction
   B. Histogram Equalization
3. Contrast Enhancement for color images
   A. Do it separately for the RBG channels by HE
   B. Do it separately for the RGB channels by gamma
4. Smoothing Filters
   A. Box filter
   B. Min\max filter
   C. Median filter
   D. Gaussian filter
   E. Laplacian filter

Preprocessing:

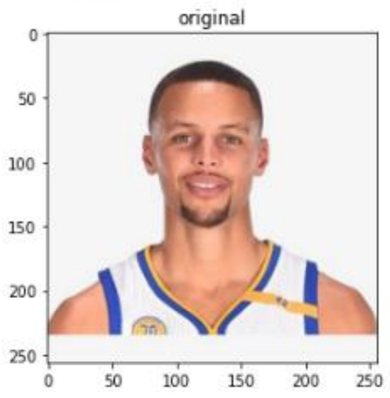
1. Load images

   Use python matplotlib api load images，and change images to 3-dimension array.

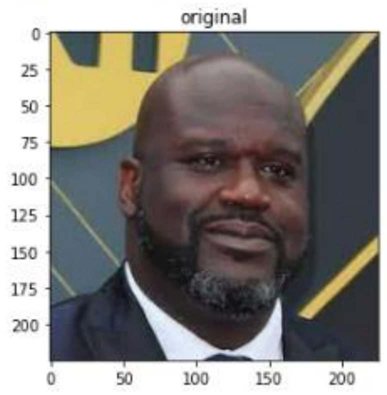
```
image shape :  (96, 96, 3)
```



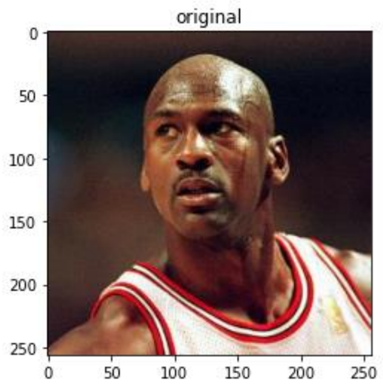
```
image shape :  (256, 256, 3)
```



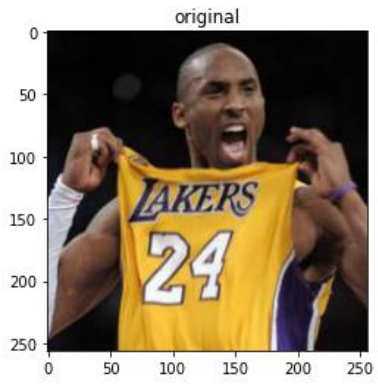
```
image shape :  (225, 225, 3)
```



```
image shape :  (256, 256, 3)
```

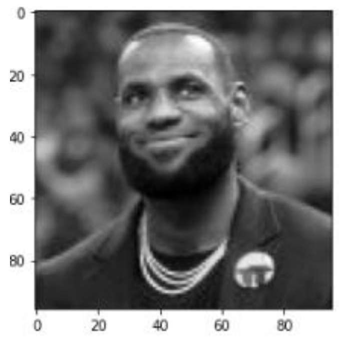


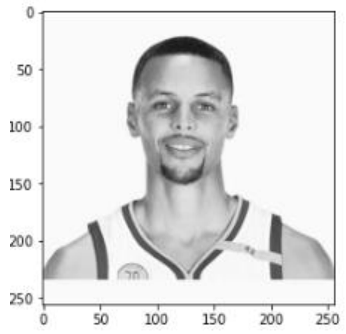
```
image shape :  (256, 256, 3)
```

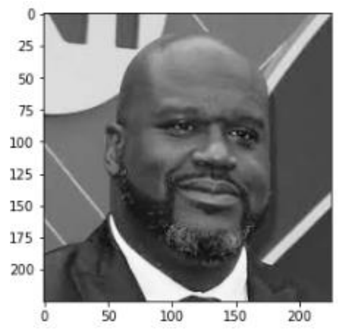


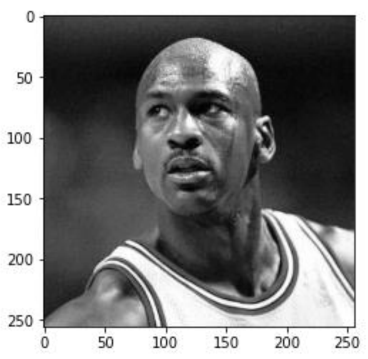
2. Image grayscale

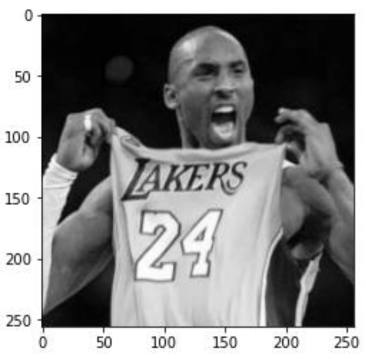
   Use Y' = 0.299 R + 0.587 G + 0.114 B，change images from 3-dimension to 1-dimension。









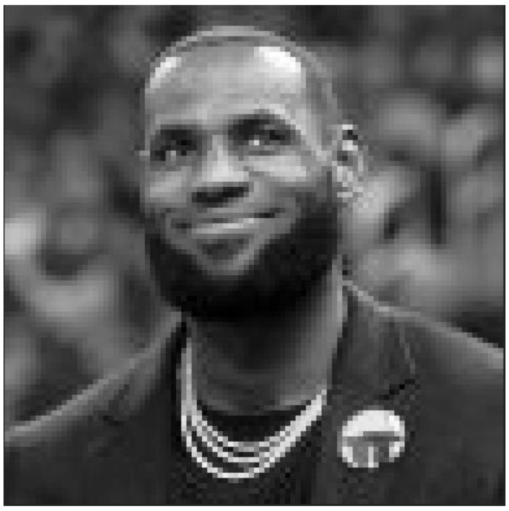


Method detail

1. Image Interpolation Methods

   A. Nearest neighbor

   I copy nearest neighbor matrix to enlarge images。The King LBJ can be the coffee shop sign canvas.

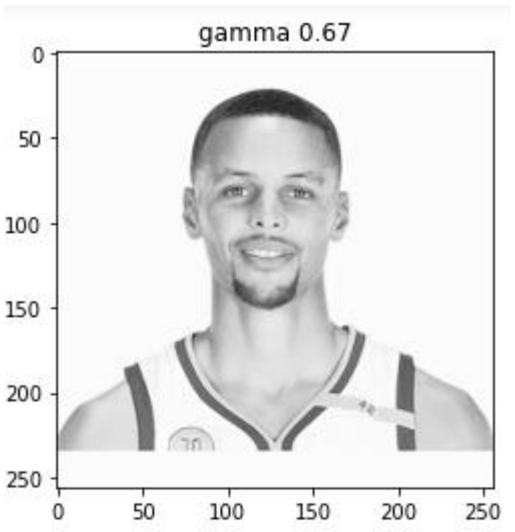


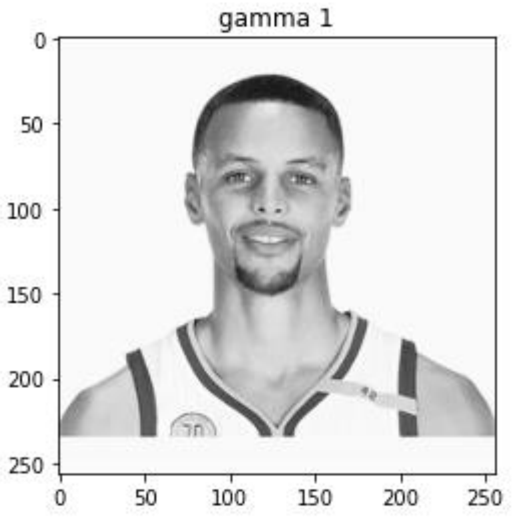
(origin) (n=5)

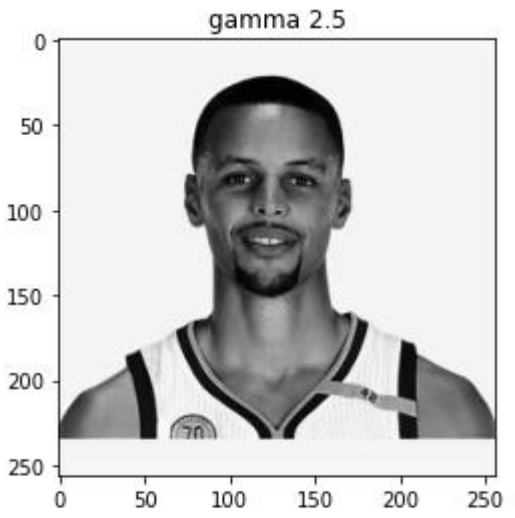
2. Intensity Transformations

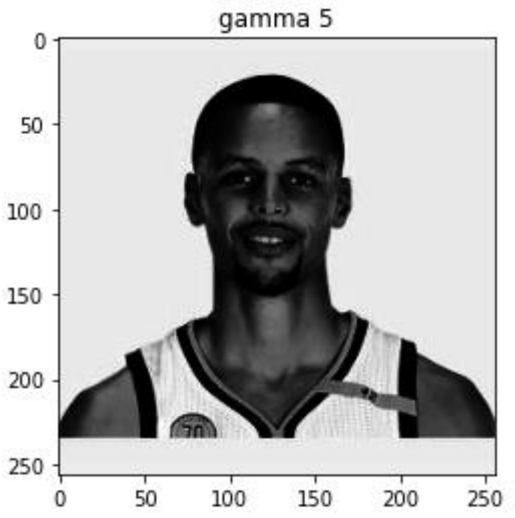
   A. Gamma correction

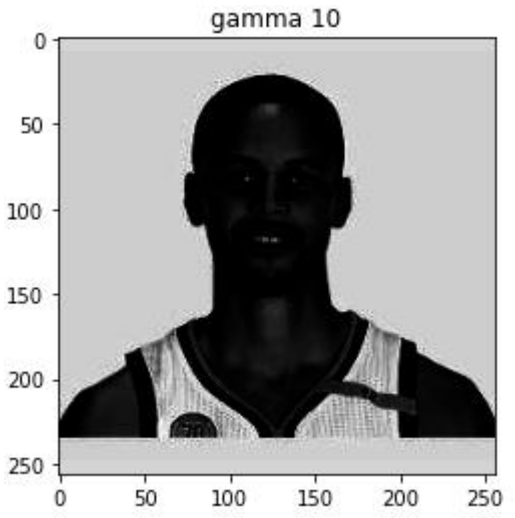
   Use s = c*r*gamma. In order to find evidence that curry is black, enhance contrast(just kidding). The gamma more higher, the people color more black.







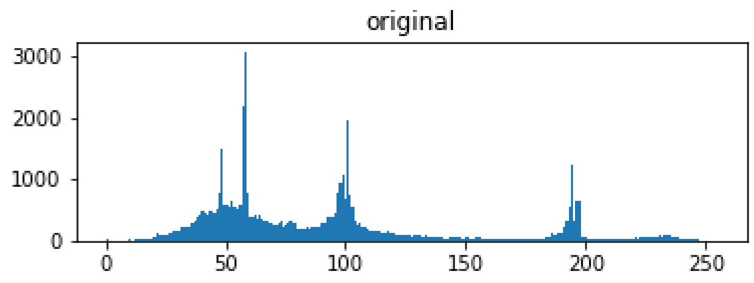




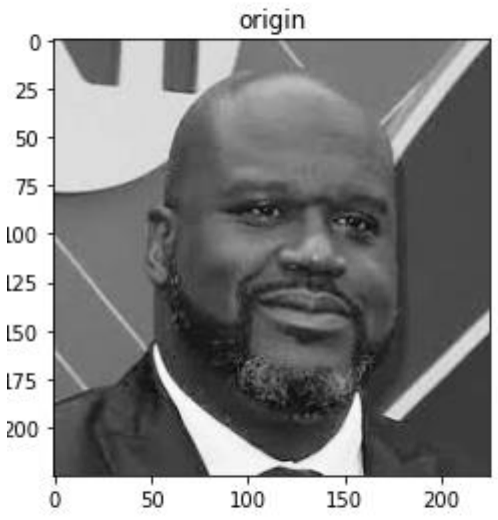
B. Histogram Equalization

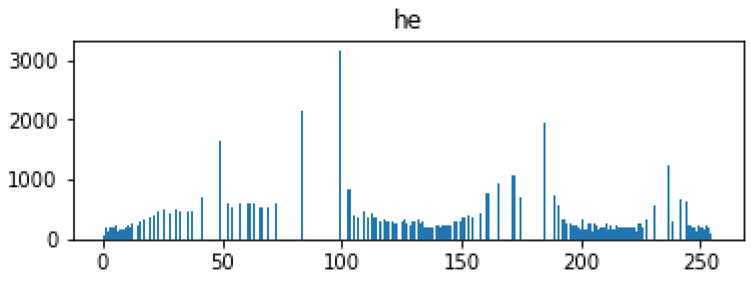
Calculate probability density function(pdf). Then, accumulate probability of every grayscale to get cumulative distribution function(cdf). Finally, every grayscale cdf plus the total grayscale-1(256-1), then you can obtain the new grayscale.

ONeal is one of my favorite stars, rolling the NBA court with a tall and sturdy body.

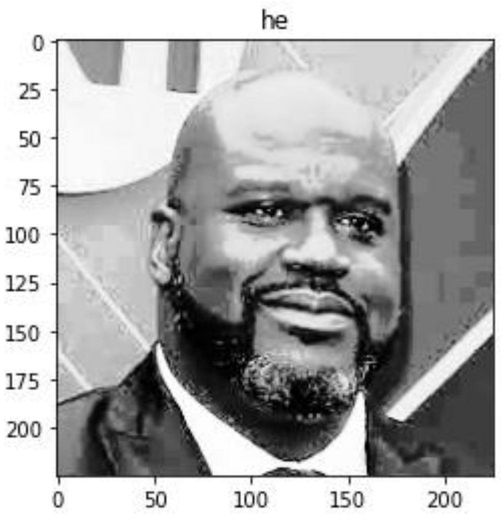


(origin grayscale histogram)



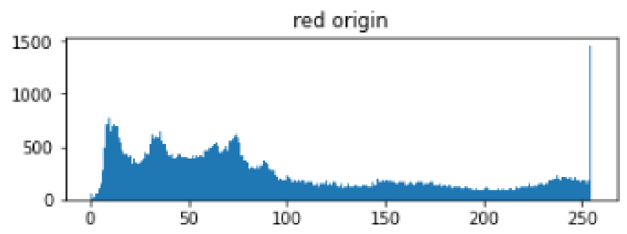


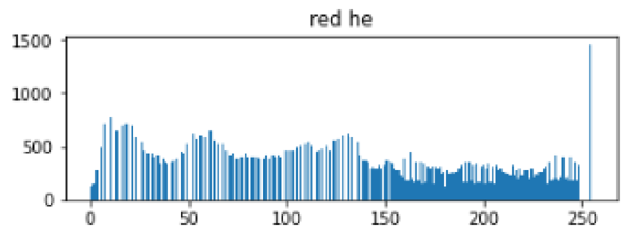
(after he grayscale histogram)



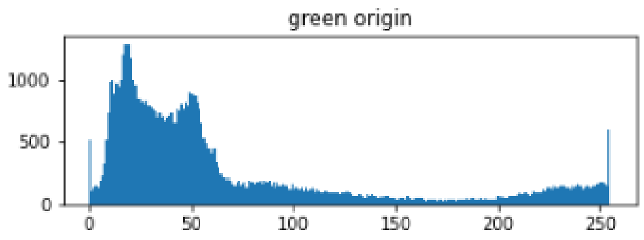
3. Contrast Enhancement for color images

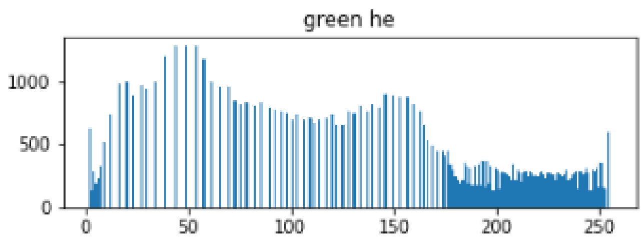
   A. Do it separately for the RBG channels by HE



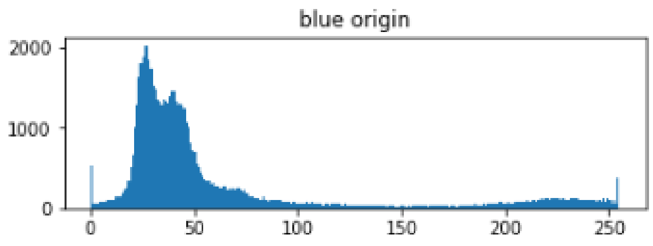


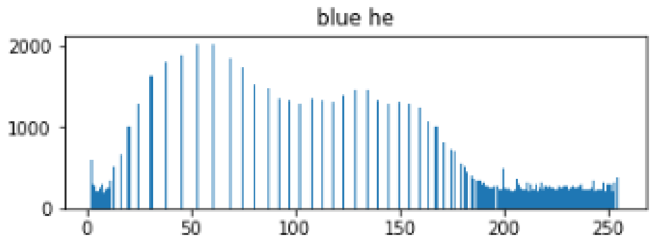
(origin red v.s. after he red histogram))



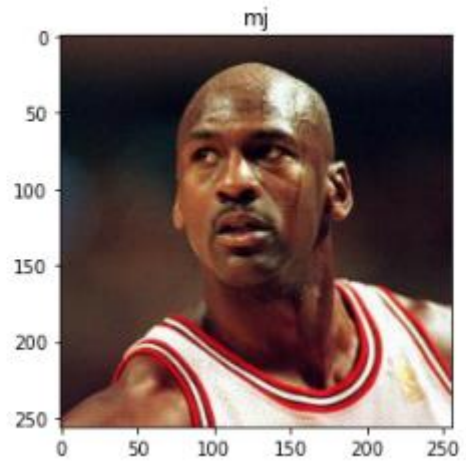


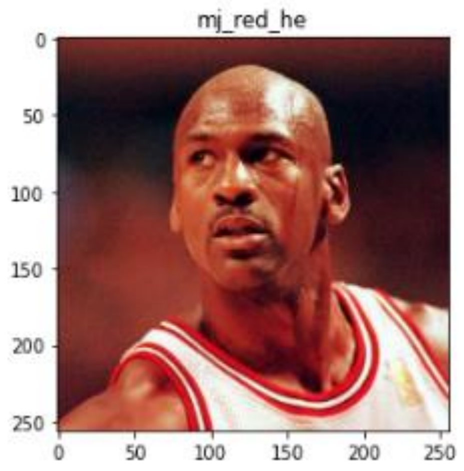
(origin green v.s. after he green histogram))

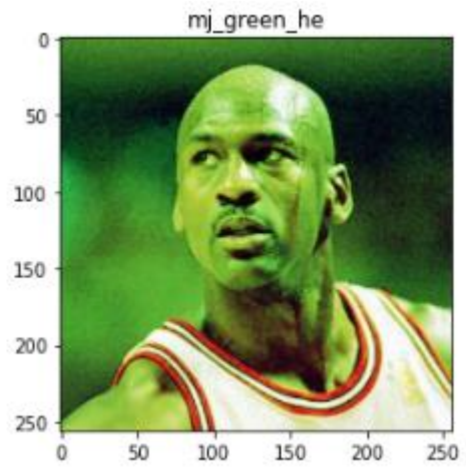


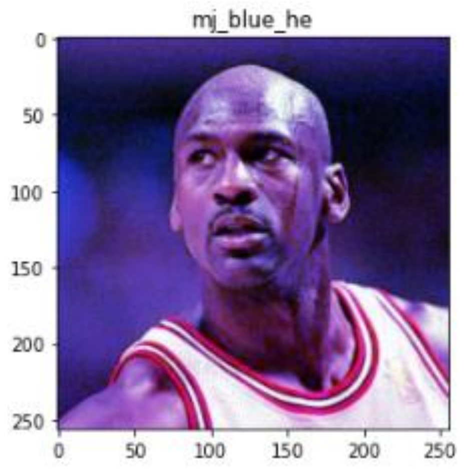


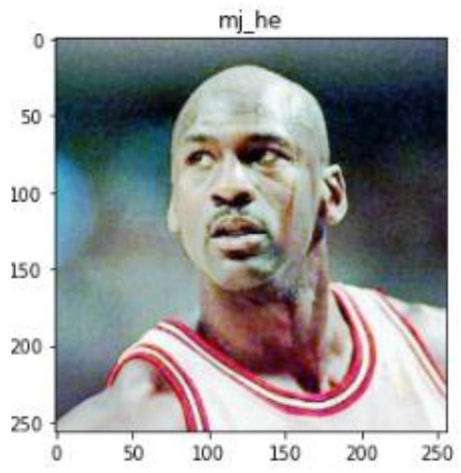
(origin blue v.s. after he blue histogram))



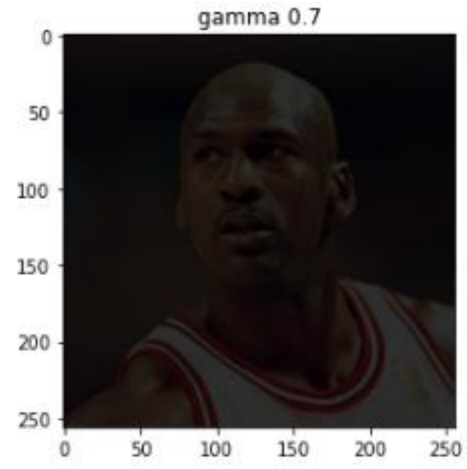


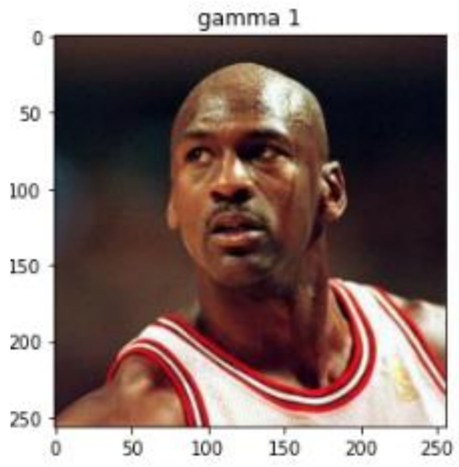


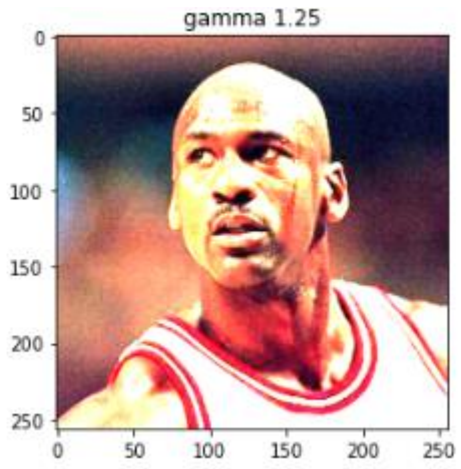


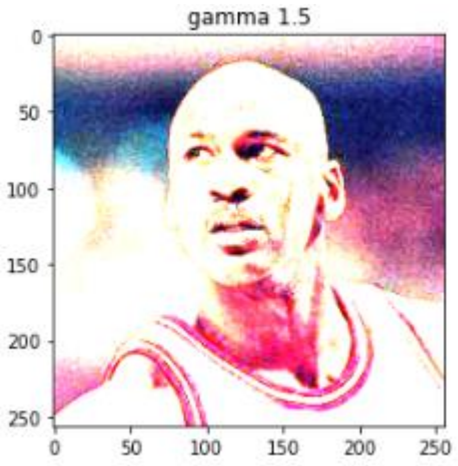


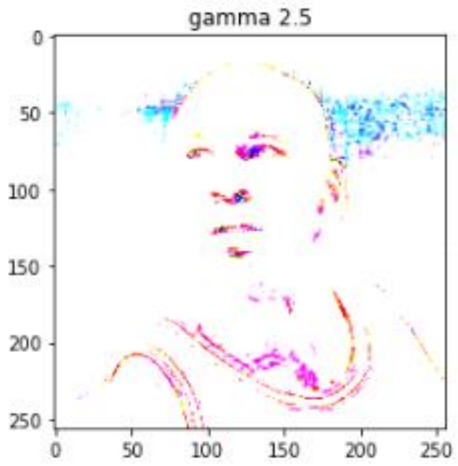
B. Do it separately for the RGB channels by gamma


gamma 0.7
0
50
100
150
200
250


gamma 1
0
50
100
150
200
250

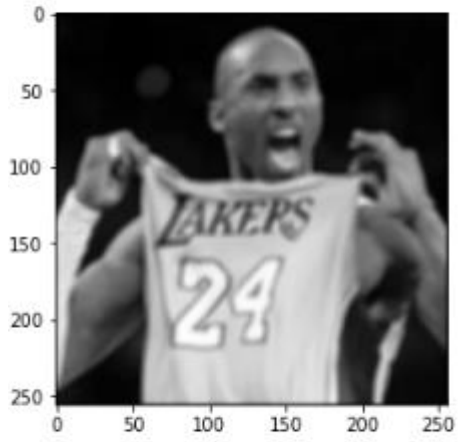

gamma 1.25
0
50
100
150
200
250


gamma 1.5
0
50
100
150
200
250


gamma 2.5
0
50
100
150
200
250

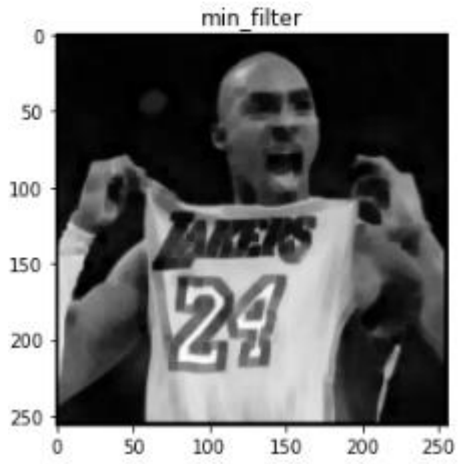
5. Smoothing Filters

A. Box filter

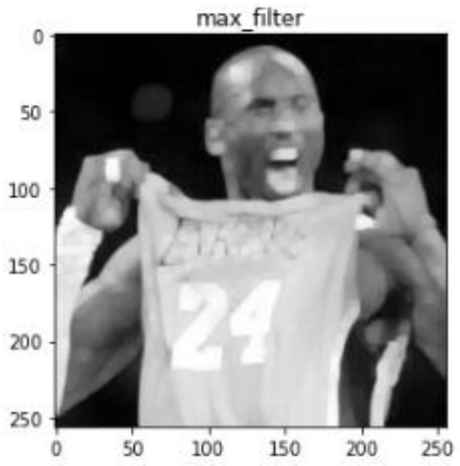


(kernel=5)

B. Min\max filter

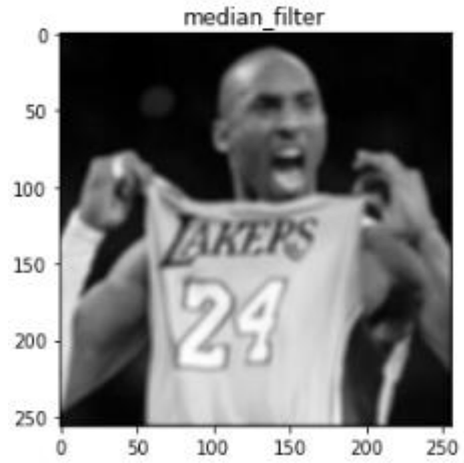


(kernel=5)



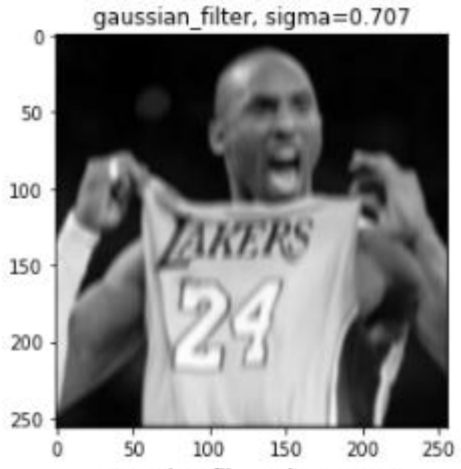
(kernel=5)

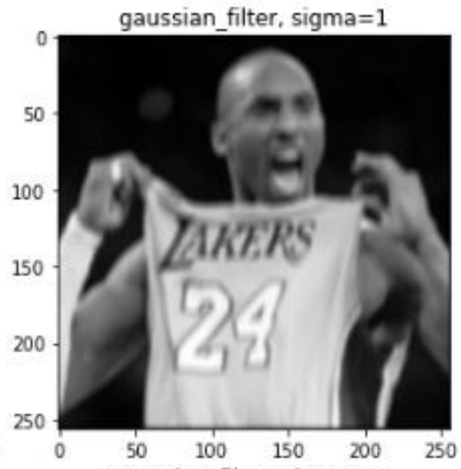
C. Median filter

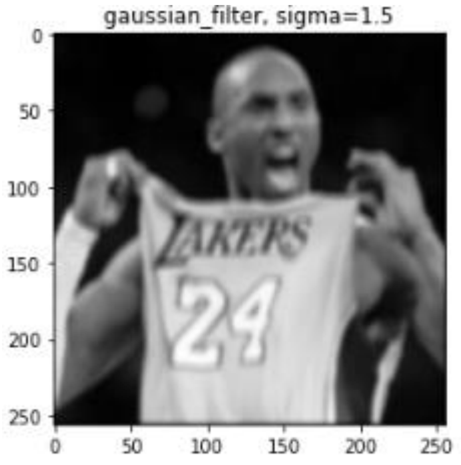


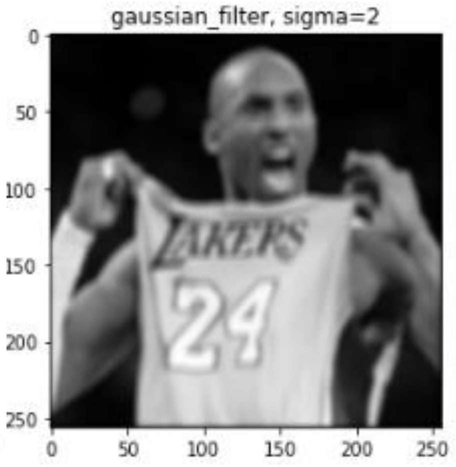
(kernel=5)

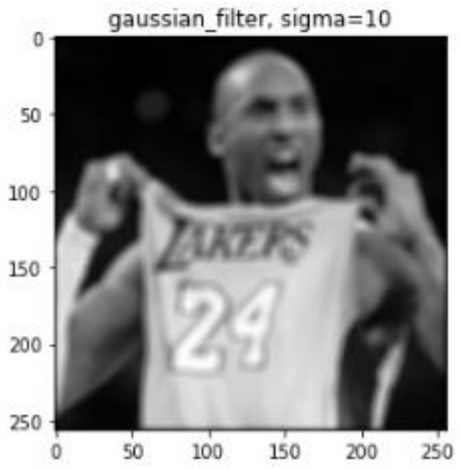
D. Gaussian filter

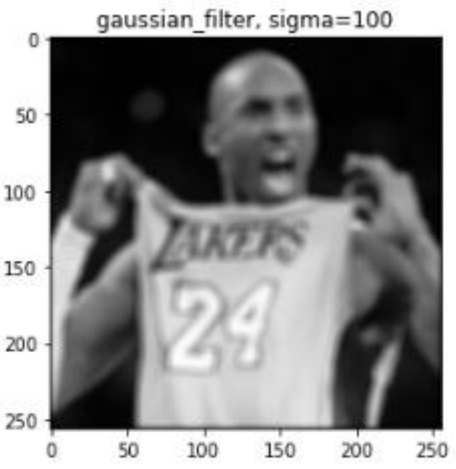






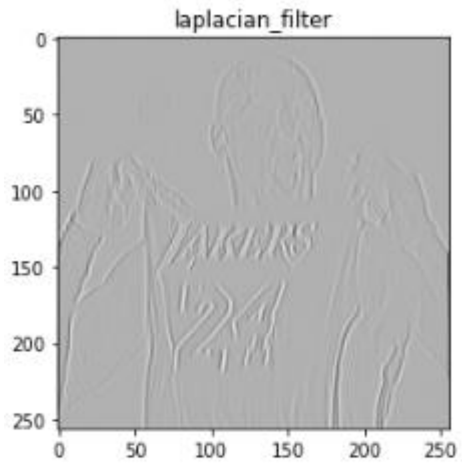






E. Laplacian filter

We can obtain contour obviously.

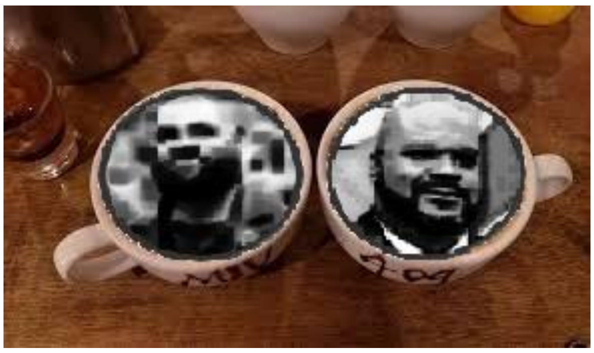


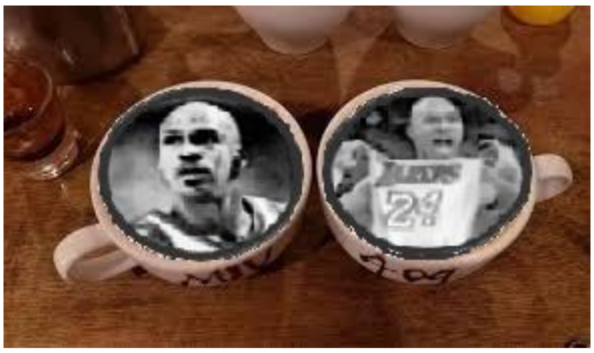
Result

First I use histogram equalization to enhance constrast.

Second I use min filter let image more roughly.

Then can get the coffee latte, like below images.





Code:

Image preprocessing

A. load images

```
import numpy as np
import matplotlib.pyplot as plt

mj = plt.imread("person0.jpg")
kobe = plt.imread("person1.jpg")
lbj  = plt.imread("person2.jpg")
curry  = plt.imread("person3.jpg")
shaq  = plt.imread("person4.jpg")

print("image shape : ", mj.shape)
plt.imshow(mj)
plt.title('original')
plt.show()

print("image shape : ", kobe.shape)
plt.imshow(kobe)
plt.title('original')
plt.show()
```

B. to gray

```
#  Y' = 0.299 R + 0.587 G + 0.114 B
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

mj_gray = rgb2gray(mj)
plt.imshow(mj_gray, cmap='Greys_r')
plt.show()

kobe_gray = rgb2gray(kobe)
plt.imshow(kobe_gray, cmap='Greys_r')
plt.show()
```

1. Image Interpolation Methods

   A. Nearest Neighbor

```
def NN_interpolation(a, n):
    '''
    n: enlarge times
    '''
    temp = np.zeros((a.shape[0] * n, a.shape[1] * n))
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i*n:i*n+n,j*n:j*n+n] = a[i][j]
    return temp
```

```
lbj_large = NN_interpolation(lbj_gray, 5)
plt.imshow(lbj_large, cmap='Greys_r')
plt.show()
# save images
plt.imsave('lbj_large.png', lbj_large, cmap='Greys_r')
plt.imsave('lbj_original.png', lbj_gray, cmap='Greys_r')
```

2. Intensity Transformations

A. Gamma correction

```
def gamma_correction(a, gamma):
    '''
    gamma:contrast enhancement
    '''
    c = 1
    s = c * a ** gamma
    return s
```

```
gammalist = [0.67, 1, 2.5, 5, 10]
for gamma in gammalist:
    s = gamma_correction(curry_gray, gamma)
    plt.title('gamma {}'.format(gamma))
    plt.imshow(s, cmap='Greys_r')
    plt.show()
```

B. Histogram Equalization

```
def he(a):
    # plot origin histogram
    original = a.reshape(-1)
    plt.subplot(2,1,1)
    plt.title('original')
    plt.hist(original, bins=256)
    plt.show()


    # reference:https://codeinfo.space/imageprocessing/histogram-equalization/
    hist, bins = np.histogram(a.reshape(-1), bins=256, range=[0,255])
    pdf = hist/a.size                                 # pdf
    cdf = pdf.cumsum()                                # cdf
    equ_value = np.around(cdf * 255).astype('uint8')  # round(cdf*255)

    img2int = np.around(a.reshape(-1)).astype(int)    # float to int

    temp = np.zeros(a.reshape(-1).shape[0])
    for i, pixel in enumerate(img2int):
        temp[i] = equ_value[pixel]

    he = temp.reshape((a.shape[0], a.shape[1]))

    # plot histogram equalization
    plt.subplot(2,1,2)
    plt.hist(temp, bins=256)
    plt.title('he')
    plt.show()

    return he
```

```
shaq_he = he(shaq_gray)

# plot origin image
plt.imshow(shaq_gray, cmap='Greys_r')
plt.title("origin")
plt.show()

plt.imshow(shaq_he, cmap='Greys_r')
plt.title("he")
plt.show()
```

3. Contrast Enhancement for color images

A. Do it separately for the RBG channels by HE

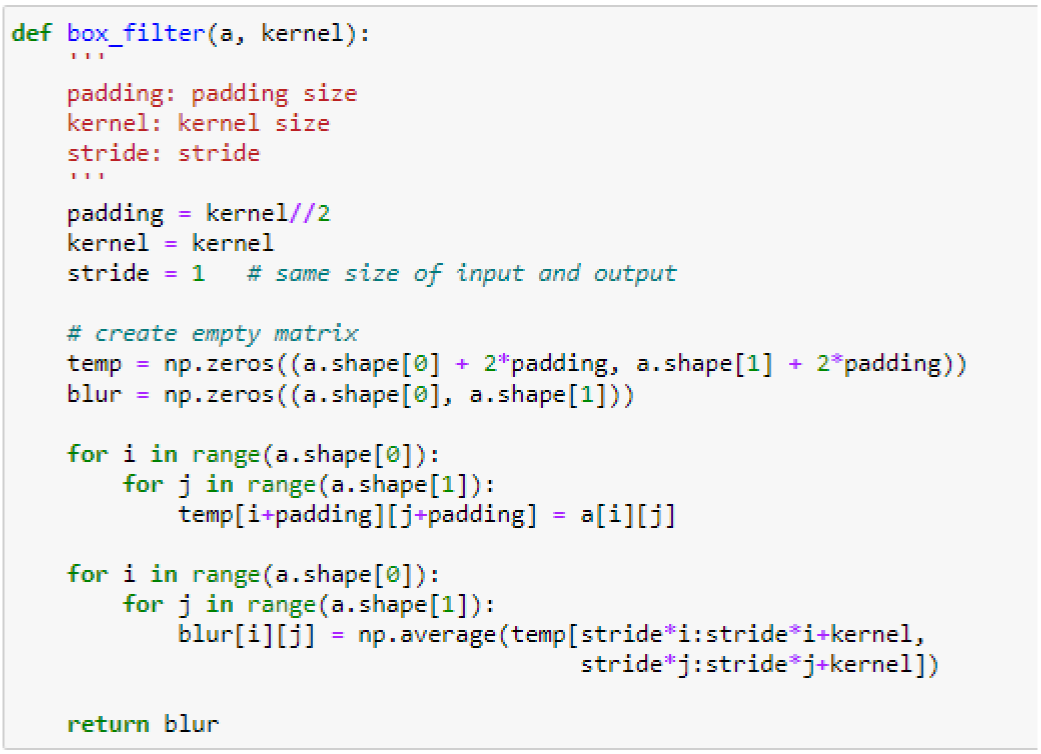
```
# enhance red
mj_red = mj[:,:,0]
mj_red_he_t = he(mj_red, "red origin", "red he")
mj_red_he = np.stack((mj_red_he_t, mj[:,:,1], mj[:,:,2]), axis=2).astype(int)

# enhance green
mj_green = mj[:,:,1]
mj_green_he_t = he(mj_green, "green origin", "green he")
mj_green_he = np.stack((mj[:,:,0], mj_green_he_t, mj[:,:,2]), axis=2).astype(int)

# enhance blue
mj_blue = mj[:,:,2]
mj_blue_he_t = he(mj_blue, "blue origin", "blue he")
mj_blue_he = np.stack((mj[:,:,0], mj[:,:,1], mj_blue_he_t), axis=2).astype(int)

mj_he = np.stack((mj_red_he_t, mj_green_he_t, mj_blue_he_t), axis=2).astype(int)

# origin
plt.imshow(mj)
plt.title("mj")
plt.show()
# enhance red
plt.imshow(mj_red_he)
plt.title("mj_red_he")
plt.show()
# enhance green
plt.imshow(mj_green_he)
plt.title("mj_green_he")
plt.show()
# enhance blue
plt.imshow(mj_blue_he)
plt.title("mj_blue_he")
plt.show()

plt.imshow(mj_he)
plt.title("mj_he")
plt.show()
```

B. Do it separately for the RGB channels by gamma

```
gammalist = [0.7, 1, 1.25, 1.5, 2.5]
for gamma in gammalist:
    mj_red_gamma   = gamma_correction(mj_red, gamma)
    mj_green_gamma = gamma_correction(mj_green, gamma)
    mj_blue_gamma  = gamma_correction(mj_blue, gamma)
    mj_gamma = np.stack((mj_red_gamma, mj_green_gamma, mj_blue_gamma), axis=2).astype(int)

    plt.imshow(mj_gamma)
    plt.title("gamma {}".format(gamma))
    plt.show()
```

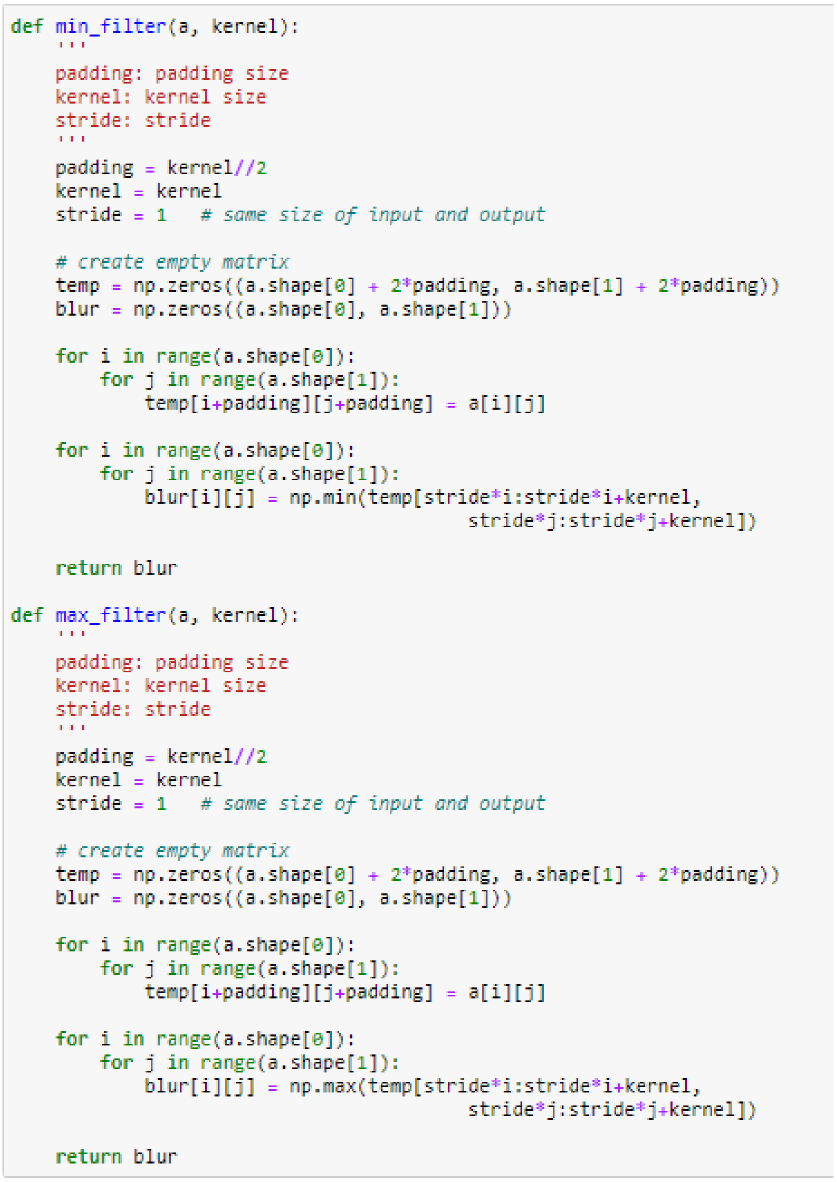
4. Smoothing Filters

A. Box filter



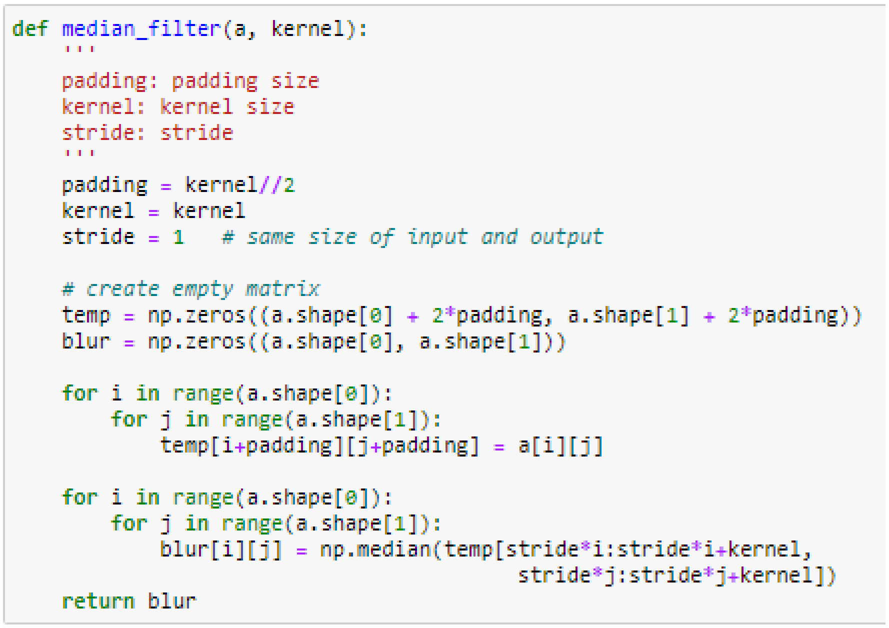
```python
def box_filter(a, kernel):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = np.average(temp[stride*i:stride*i+kernel,
                                         stride*j:stride*j+kernel])

    return blur
```

B. Min\max filter



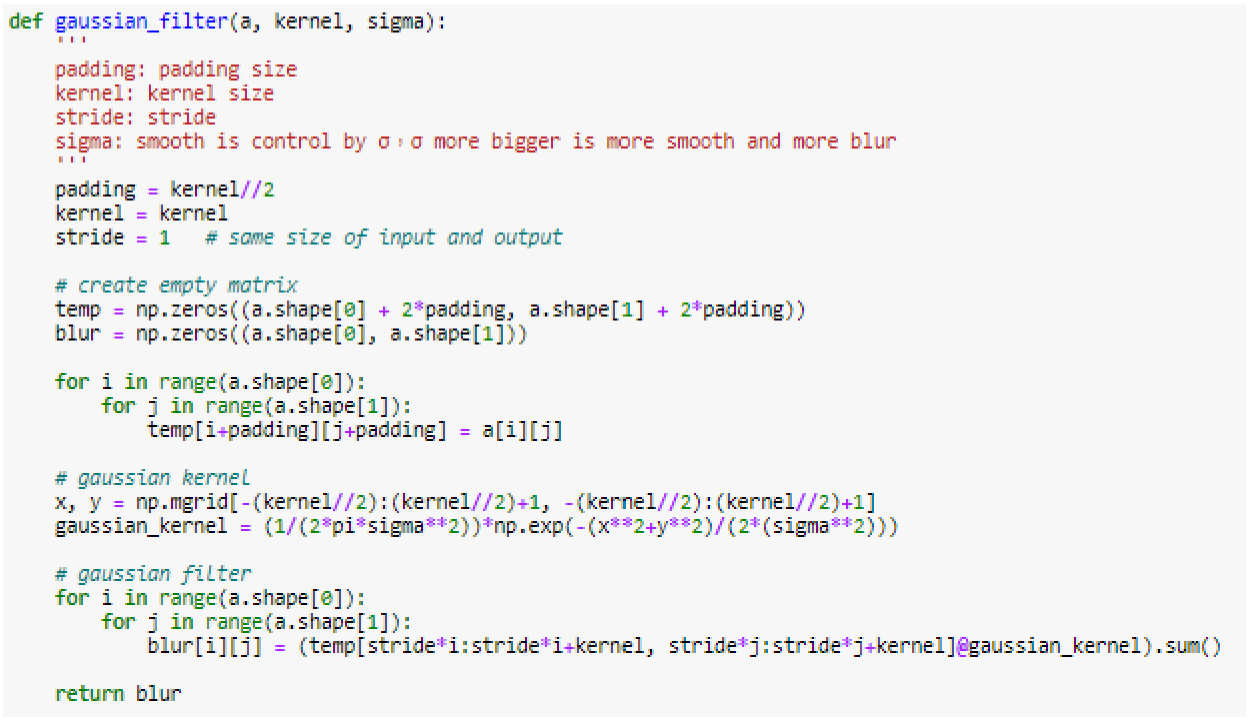
```python
def min_filter(a, kernel):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = np.min(temp[stride*i:stride*i+kernel,
                                     stride*j:stride*j+kernel])

    return blur

def max_filter(a, kernel):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = np.max(temp[stride*i:stride*i+kernel,
                                     stride*j:stride*j+kernel])

    return blur
```

C. Median filter



```python
def median_filter(a, kernel):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = np.median(temp[stride*i:stride*i+kernel,
                                         stride*j:stride*j+kernel])
    return blur
```

D. Gaussian filter



```python
def gaussian_filter(a, kernel, sigma):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    sigma: smooth is control by σ，σ more bigger is more smooth and more blur
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    # gaussian kernel
    x, y = np.mgrid[-(kernel//2):(kernel//2)+1, -(kernel//2):(kernel//2)+1]
    gaussian_kernel = (1/(2*pi*sigma**2))*np.exp(-(x**2+y**2)/(2*(sigma**2)))

    # gaussian filter
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = (temp[stride*i:stride*i+kernel, stride*j:stride*j+kernel]@gaussian_kernel).sum()

    return blur
```

E. Laplacian filter

```python
def laplacian_filter(a, kernel=3):
    '''
    padding: padding size
    kernel: kernel size
    stride: stride
    '''
    padding = kernel//2
    kernel = kernel
    stride = 1   # same size of input and output

    # create empty matrix
    temp = np.zeros((a.shape[0] + 2*padding, a.shape[1] + 2*padding))
    blur = np.zeros((a.shape[0], a.shape[1]))

    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            temp[i+padding][j+padding] = a[i][j]

    # laplacian kernel
    laplacian_kernel = np.array([[0,1,0],
                                 [1,-4,1],
                                 [0,1,0]])

    # laplacian filter
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            blur[i][j] = (temp[stride*i:stride*i+kernel, stride*j:stride*j+kernel]@laplacian_kernel).sum()


    return blur
```