HW 02 – REPORT

소속: 정보컴퓨터공학부

학번: 202055501

이름: Dilyana Dimitrova

1. Introduction

1.1 Linear filtering:

Filtering is a technique for modifying or enhancing an image. Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm (filtering function) to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel.

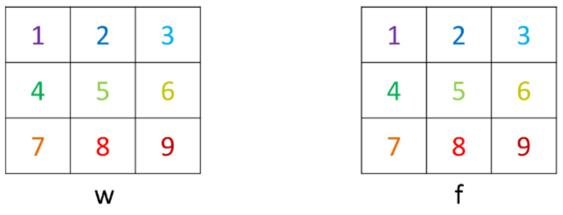
1.2 Cross-correlation and Convolution

- Cross-correlation and convolution are both operations applied to images. Cross-correlation means sliding a kernel (filter) across an image. Convolution means sliding a flipped kernel across an image.
- Convolution is an element-wise multiplication of two matrices (same dimensions) followed by a sum.

Cross-correlation's equation:

$$S[f] = w \otimes f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$



- *w(i, j)* – values of the kernel (filter coefficient)

** multiplication starts from top left corner of the input matrix

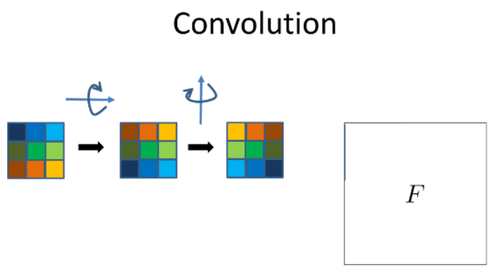
1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 + 8*8 + 9*9

Convolution's equation:

** multiplication starts from bottom right corner of the input matrix and moves to the left and up

$$S[f] = w * f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m-i, n-j)$$



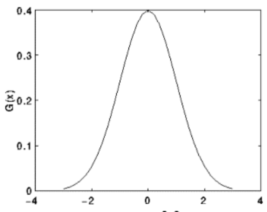
A correlation operation produces a copy of the impulse but rotated by an angle of 180 degrees. Therefore, if we pre-rotate the filter and perform the same sliding sum of products operation, we should be able to obtain the desired result. Rotation is done at 180 degrees about the center element or flipping left-right and up-down.

1.3 **Gaussian filter:**

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses the Gaussian function as a kernel.

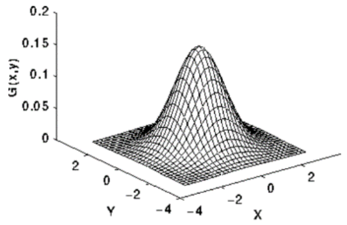
Gauss's function in 1D.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$



Gauss's function in 2D:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



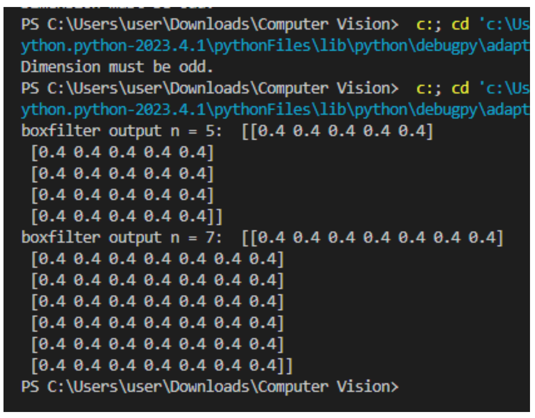
The idea of Gaussian smoothing is to use this 2-D distribution as a `point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point.

2. Program description:

** detailed comments are included in the file containing the code

2.1 Function *"boxfilter(n)"* returns a box filter of size n by n, where n is odd number.

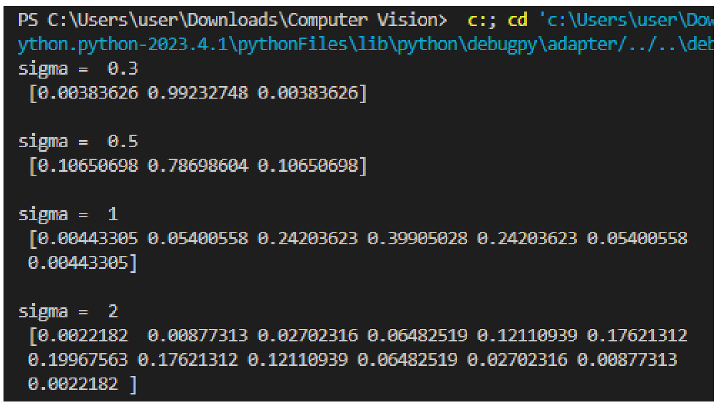
Output for n = 4, 5 and 7:



```
PS C:\Users\user\Downloads\Computer Vision>  c:; cd 'c:\Us
ython.python-2023.4.1\pythonFiles\lib\python\debugpy\adapt
Dimension must be odd.
PS C:\Users\user\Downloads\Computer Vision>  c:; cd 'c:\Us
ython.python-2023.4.1\pythonFiles\lib\python\debugpy\adapt
boxfilter output n = 5:  [[0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4]]
boxfilter output n = 7:  [[0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]
 [0.4 0.4 0.4 0.4 0.4 0.4 0.4]]
PS C:\Users\user\Downloads\Computer Vision>
```

2.2 gauss1d(sigma) - returns a 1D Gaussian filter for a given value of sigma; the filter is a 1D Numpy array with length 6 times sigma rounded up to the next odd integer; x is an array of the distances from the center.

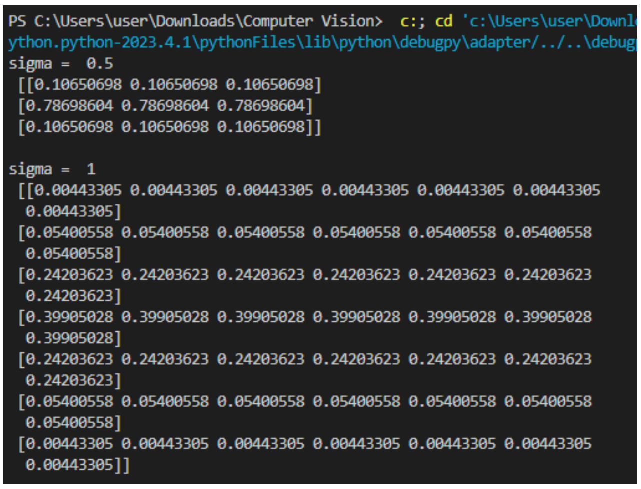
Output:



```
PS C:\Users\user\Downloads\Computer Vision>  c:; cd 'c:\Users\user\Dow
ython.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\deb
sigma =  0.3
 [0.00383626 0.99232748 0.00383626]

sigma =  0.5
 [0.10650698 0.78698604 0.10650698]

sigma =  1
 [0.00443305 0.05400558 0.24203623 0.39905028 0.24203623 0.05400558
 0.00443305]

sigma =  2
 [0.0022182  0.00877313 0.02702316 0.06482519 0.12110939 0.17621312
 0.19967563 0.17621312 0.12110939 0.06482519 0.02702316 0.00877313
 0.0022182 ]
```

2.3 gauss2d(sigma) - returns a 2D Gaussian filter for a given value of sigma.

Output:



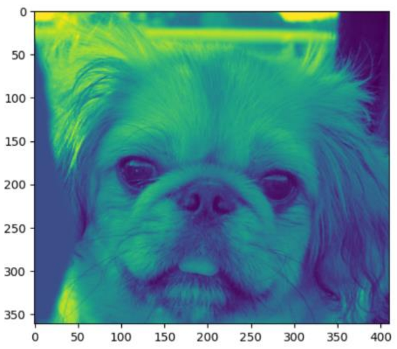
```
PS C:\Users\user\Downloads\Computer Vision>  c:; cd 'c:\Users\user\Downl
ython.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter/../..\debug
sigma =  0.5
 [[0.10650698 0.10650698 0.10650698]
 [0.78698604 0.78698604 0.78698604]
 [0.10650698 0.10650698 0.10650698]]

sigma =  1
 [[0.00443305 0.00443305 0.00443305 0.00443305 0.00443305 0.00443305
  0.00443305]
 [0.05400558 0.05400558 0.05400558 0.05400558 0.05400558 0.05400558
  0.05400558]
 [0.24203623 0.24203623 0.24203623 0.24203623 0.24203623 0.24203623
  0.24203623]
 [0.39905028 0.39905028 0.39905028 0.39905028 0.39905028 0.39905028
  0.39905028]
 [0.24203623 0.24203623 0.24203623 0.24203623 0.24203623 0.24203623
  0.24203623]
 [0.05400558 0.05400558 0.05400558 0.05400558 0.05400558 0.05400558
  0.05400558]
 [0.00443305 0.00443305 0.00443305 0.00443305 0.00443305 0.00443305
  0.00443305]]
```

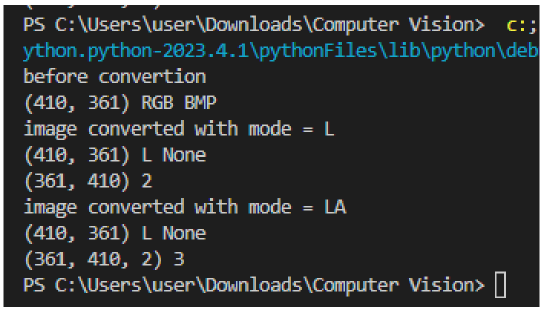
2.4 convolve2d(array, filter) & gaussconvolve2d(array, sigma)

- convolve2d(array, filter) - takes in an image (stored in `array`) and a filter, and performs convolution to the image with zero paddings. The size of the input image should equal the size of the output image.

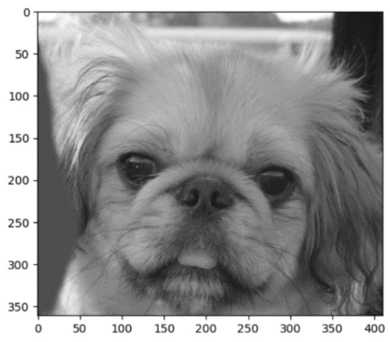
- gaussconvolve2d(array, sigma) - applies Gaussian convolution to a 2D array for the given value of sigma. The result should be a 2D array.

The dog's image is in bmp format (with 3 colour channels). When converting it to greyscale with mode='L', the image's array is 2D. But when I convert the same image to greyscale with mode='LA', I get a black&white image and the image's array is 3D.

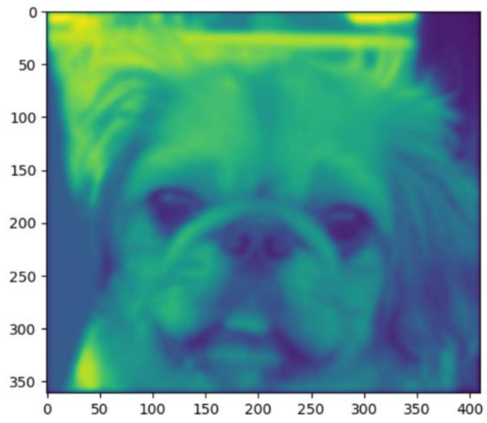


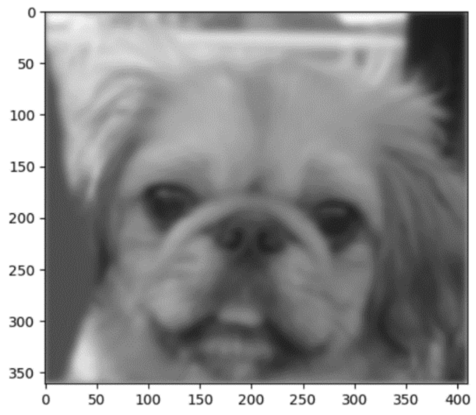


```
PS C:\Users\user\Downloads\Computer Vision>  c:;
ython.python-2023.4.1\pythonFiles\lib\python\deb
before convertion
(410, 361) RGB BMP
image converted with mode = L
(410, 361) L None
(361, 410) 2
image converted with mode = LA
(410, 361) L None
(361, 410, 2) 3
PS C:\Users\user\Downloads\Computer Vision> 
```



I tried to convolve both images with the gaussconvolve2d function and the output is as follows:



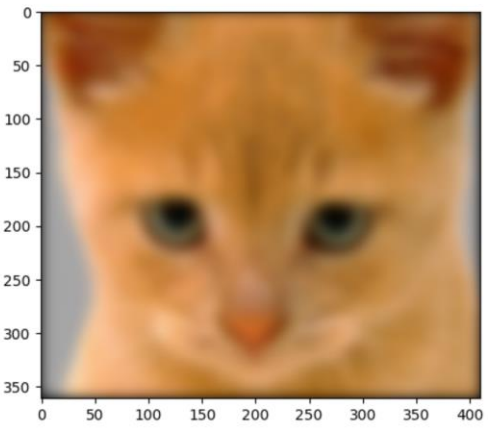


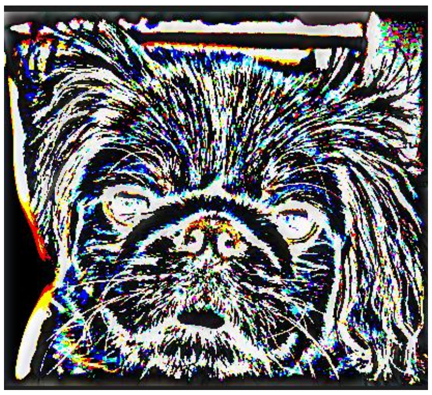
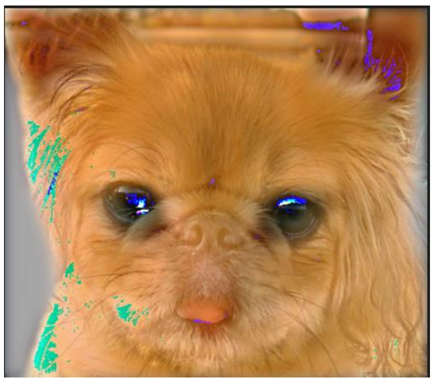
For the image converted with mode='LA', the array is 3 dimensional, the additional attribute is for colour. Therefore, I needed to modify little bit the convolve2d function with an additional (3rd) loop iterating through each colour channel. In this case the returned array will be 3 dimensional.

Hybrid images

Gaussian filtering produces a low pass (blurred) filtered version of an image. Consequently, the difference between the original and the blurred results in a high pass filtered version of the image. A hybrid image is the sum of a low-pass filtered version of one image and a high-pass filtered version of another image. There is a free parameter, which can be tuned for each image pair, which controls how much high frequency to remove from the first image and how much low frequency to leave in the second one. This is called the "cut-off-frequency". The cut-off frequency is controlled by changing the sigma of the Gaussian filter used in constructing the hybrid images.

Outputs for the low frequency image of the cat, high frequency image of the dog and hybrid image:



3. Conclusion

Through this assignment we have learned about convolution and image smoothing using Gaussian filter. We have created low and high frequency images and a hybrid one from them.