REPORT

국민대학교
KOOKMIN UNIVERSITY

컴퓨터 비전 과제1 보고서

Image Filtering

20172099 김의찬

010-2426-7395
uichan8@naver.com
전자공학부 전자시스템공학전공

1. 이미지에 필터 적용하기


```python
def my_imfilter(image, kernel):
    """
    이미지에 필터를 적용해주는 함수

    Inputs
    - image: numpy nd-array of dim (m,n) or (m, n, c)
    - kernel: numpy nd-array of dim (k, l)
    - Returns : filtered_image: numpy nd-array of dim of equal 2D size (m,n) or 3D size (m, n, c)
    - Errors if: filter/kernel has any even dimension -> raise an Exception with a suitable error message.
    """

    #케널에 대한 검증 : 짝수이면 오류
    assert kernel.shape[0] % 2 != 0, "error: kernel[0] size must be odd"
    assert kernel.shape[1] % 2 != 0, "error: kernel[1] size must be odd"

    #이미지 정보 추출
    is_color = image.ndim > 2
    if is_color:
        third_shape = image.shape[2]
    else:
        third_shape = 0

    #출력이미지 초기화
    filtered_image = np.zeros(image.shape)

    #제로패딩
    row = kernel.shape[0]//2
    col = kernel.shape[1]//2
    if third_shape == 0:
        image = np.pad(image,[(row,row),(col,col)],'constant', constant_values = 0)
    else:
        image = np.pad(image,[(row,row),(col,col),(0,0)],'constant', constant_values = 0)

    #브로드케스팅 하기위한 전처리
    kernel = kernel.reshape(kernel.shape[0],kernel.shape[1],1)

    #컨볼루션(코레이션) : 이미지에서 슬라이싱 한 후에, 내적한걸 filtered_image에 저장
    for i in range(filtered_image.shape[0]):
        for j in range(filtered_image.shape[1]):
            tem = image[i:i+kernel.shape[0],j:j+kernel.shape[1]] * kernel
            if third_shape == 0:
                filtered_image[i,j] = np.sum(np.sum(tem))
            else:
                filtered_image[i,j] = np.sum(np.sum(tem,axis = 0),axis = 0)


    return filtered_image
```

1) 필터가 짝수이면 오류 출력

2) 이미지 정보 추출은 차원이 2인 이미지와, 차원이 3인 이미지를 구분하기 위해 미리 정보를 저장하는 부분입니다.

3) zero padding은 np.pad 함수를 사용하여 구현하였습니다.

4) Broadcasting 전처리는 규칙에 맞춰주기 위해 필터의 shape 를 (k,n) -> (k,n,1)로 설정하는 부분입니다

5) 컨볼루션은 이미지를 슬라이싱해서 필터 크기에 맞게 자른 후 단순 곱해서 합을 구해서

3. 중간값 필터

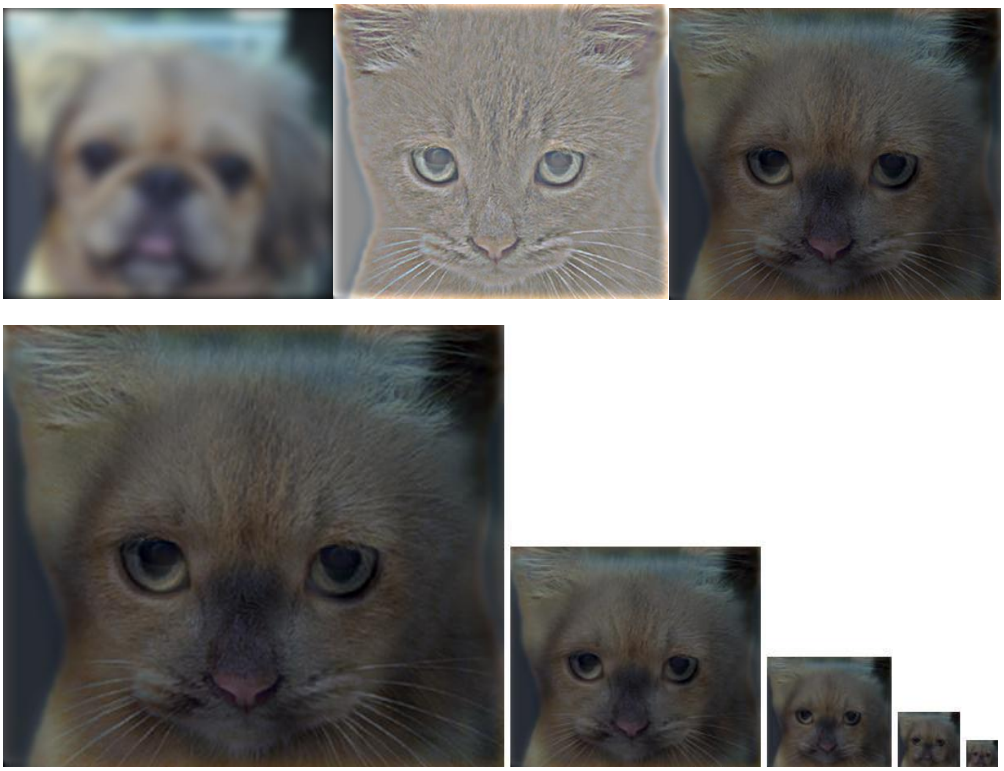
```python
def mid(nplist):
    location = np.size(nplist)//2
    nplist = nplist.reshape(np.size(nplist))

    for i in range(location+1):
        for j in range(i+1,np.size(nplist)):
            if nplist[i] > nplist[j]:
                tem = nplist[i]
                nplist[i] = nplist[j]
                nplist[j] = tem
    return nplist[location]
```

중간 값을 구하는 함수는 교환 정렬을 통해 중간 값이 있는 곳 까지만 정렬해서 추출했습니다.

```python
def my_medfilter(image, size):
    """
    Inputs
    - image: numpy nd-array of dim (m,n) or (m, n, c)
    - size: kernel size
    Returns
    - filtered_image: numpy nd-array of dim of equal 2D size (m,n) or 3D size (m, n, c)
    """
    #케널에 대한 검증 : 짝수이면 오류
    assert size % 2 != 0, "error: kernel size must be odd"

    #이미지 정보 추출
    is_color = image.ndim > 2
    if is_color:
        third_shape = image.shape[2]
    else:
        third_shape = 0

    #출력이미지 초기화
    filtered_image = np.zeros(image.shape)

    #제로패딩 : np.concat으로 붙이기 (제로패딩 사실 필요없음)
    padding_size = size//2

    if third_shape == 0:
        image = np.pad(image,[(padding_size,padding_size),(padding_size,padding_size)],'constant', constant_values = 0)
    else:
        image = np.pad(image,[(padding_size,padding_size),(padding_size,padding_size),(0,0)],'constant', constant_values = 0)

    #이미지에서 슬라이싱 한 후에, 중간값 산출 및 반환
    for i in range(filtered_image.shape[0]):
        for j in range(filtered_image.shape[1]):
            tem = image[i:i+size,j:j+size]
            if third_shape != 0:
                tem = np.array([mid(tem[:,:,0]),mid(tem[:,:,1]),mid(tem[:,:,2])])
            else:
                tem = mid(tem)
            filtered_image[i,j] = tem

    return filtered_image
```

이미지를 필터 크기만큼 슬라이싱해서 그 배열에서 중간값을 구해서 결과 이미지 배열에 반환하였습니다. 여기에서 제로 패딩 부분을 지워도 프로그램이 돌아가도록 만들었는데, 패딩을 하게 되면 이미지 네 코너에서 검은색 부분이 나타납니다. (한글 문서에 하라고 나와 있었습니다.)

4. 하이브리드 이미지



https://stanford.edu/class/ee367/reading/OlivaTorralb_Hybrid_Siggraph06.pdf

논문을 참고하라고 문서에 나와 있는데, 들어가지지 않아서 타 논문을 참고하였습니다.

$$H = I_1 \cdot G_1 + I_2 \cdot (1 - G_2)$$

G1은 high-pass (1- G2) 는 low-pass filter 입니다. Low-pass이미지는 논문처럼 구현하지 않고, 코드에 나온 지시문대로 원본 이미지를 블러된 이미지를 빼서 구현했습니다. '**이미지를 서로 더한다**' 라는 부분을 참고했습니다. 이미지 픽셀 값의 범위는 0~1 이기 때문에 np.clip 함수로 넘어가는 값들을 0,1로 고정했습니다.

```python
def gen_hybrid_image(image1, image2, cutoff_frequency):
    """
    Inputs:
    - image1 -> The image from which to take the low frequencies.
    - image2 -> The image from which to take the high frequencies.
    - cutoff_frequency -> The standard deviation, in pixels, of the Gaussian
                          blur that will remove high frequencies.

    Task:
    - Use my_imfilter to create 'low_frequencies' and 'high_frequencies'.
    - Combine them to create 'hybrid_image'.
    """

    assert image1.shape[0] == image2.shape[0]
    assert image1.shape[1] == image2.shape[1]
    assert image1.shape[2] == image2.shape[2]

    # Steps:
    # (1) Remove the high frequencies from image1 by blurring it. The amount of
    #     blur that works best will vary with different image pairs
    # generate a 1x(2k+1) gaussian kernel with mean=0 and sigma = s, see https://stackoverflow.com/questions/17190649/how-to-obtain-a-gaussian-filter-in-python
    s, k = cutoff_frequency, cutoff_frequency*2
    probs = np.asarray([exp(-z*z/(2*s*s))/sqrt(2*pi*s*s) for z in range(-k,k+1)], dtype=np.float32)
    kernel = np.outer(probs, probs)

    low_frequencies = np.zeros(image1.shape)
    low_frequencies = my_imfilter(image1,kernel) # 이미지1 필터링

    # (2) Remove the low frequencies from image2. The easiest way to do this is to
    #     subtract a blurred version of image2 from the original version of image2.
    #     This will give you an image centered at zero with negative values.

    high_frequencies = np.zeros(image1.shape)
    high_frequencies = image2 - my_imfilter(image2,kernel) #이미지2 필터링해서 원본에서 빼줌


    # (3) Combine the high frequencies and low frequencies

    hybrid_image = np.zeros(image1.shape)
    hybrid_image = (low_frequencies + high_frequencies)/2 #합치기
    hybrid_image = np.clip(hybrid_image,0,1) #클리핑


    # (4) At this point, you need to be aware that values larger than 1.0
    # or less than 0.0 may cause issues in the functions in Python for saving
    # images to disk. These are called in proj1_part2 after the call to
    # gen_hybrid_image().
    # One option is to clip (also called clamp) all values below 0.0 to 0.0,
    # and all values larger than 1.0 to 1.0.

    return low_frequencies, high_frequencies, hybrid_image
```