

# Homework 5. DECONVOLUTION

포항공과대학교 컴퓨터공학과 20190539 원지윤

## 1. Wiener Deconvolution

### 1.1 Implementation

```
def wiener_deconv(file_name, c_value):
    blurred = cv2.imread('../examples/'+file_name+'.jpg', cv2.IMREAD_COLOR).astype(np.float32)
    kernel = cv2.imread('../examples/'+file_name+'_out.jpg.psf.png', 0).astype(np.float32)
    b, g, r = cv2.split(blurred)
    psf = kernel / sum(sum(kernel))
    f_k = psf2otf(psf, (blurred.shape[0], blurred.shape[1]))
    results = []
    for i in [b,g,r]:
        f_i = np.fft.fft2(i)
        l_ = ((f_k ** 2) / (f_k ** 2 + c_value)) * (1 / f_k) * f_i
        results.append(np.real(np.fft.ifft2(l_)) * 255)

    result_image = cv2.merge((results[0],results[1],results[2]))

    cv2.imwrite('../outputs/wiener/'+str(c_value)+'/'+file_name+'.jpg', result_image)
```

- wiener deconvolution을 수행하는 함수 `wiener_deconv` 을 정의했다.
- 먼저, blurred image와 blur kernel 이미지를 각각 불러오고, float로 형변환을 해준 뒤 255.0으로 나누어 normalization을 해준다.
- 채널별로 연산을 수행하기 위해 blurred image는 R, G, B 채널로 분리해준다.
- kernel 이미지는 컨볼루션 연산을 수행한 뒤에 이미지의 밝기가 변하지 않게끔 하기 위해 `kernel / sum(sum(kernel))` 를 통해 normalize를 해줬다.
- Homework 2에서 했던 것과 동일하게, 함수 `psf2otf()` 를 이용했다.

```
def psf2otf(flt, img_shape):
    flt_top_half = flt.shape[0]//2
    flt_bottom_half = flt.shape[0] - flt_top_half
    flt_left_half = flt.shape[1]//2
    flt_right_half = flt.shape[1] - flt_left_half
    # Pad zeros to make the filter size the same as the image size
    flt_padded = np.zeros(img_shape, dtype=np.float32)
    # Shift the center to the top left corner
    flt_padded[:flt_bottom_half, :flt_right_half] = flt[flt_top_half:, flt_left_half:]
    flt_padded[:flt_bottom_half, img_shape[1]-flt_left_half:] = flt[flt_top_half:, :fl
    flt_padded[img_shape[0]-flt_top_half:, :flt_right_half] = flt[:flt_top_half, flt_
    flt_padded[img_shape[0]-flt_top_half:, img_shape[1]-flt_left_half:] = flt[:flt_to
    # 2D FFT
    return np.fft.fft2(flt_padded)
```

- 강의 노트의 코드를 사용하여 구현하였다.
- kernel을 이미지와 동일한 사이즈로 만들고 filter의 중심을 가장자리로 이동(shift)해준 뒤 Fourier Transform을 통해 frequency domain으로 바꿔주는 연산을 해주었다.

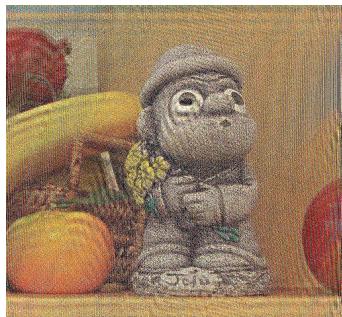
- 이후 채널별로 각각을 푸리에 변환을 해주어 frequency domain으로 바꾸어주고, 아래의 수식에 따른 연산을 수행한 뒤 다시 푸리에 역변환을 통해 spatial domain으로 변경했다.
  - $k$ 는 blur kernel,  $b$ 는 blurred image이며,  $c$ 는 상수이다.  $c$  값은 함수의 매개변수로 입력받아 그 값을 사용했다.

$$l = \mathcal{F}^{-1}\left(\frac{|\mathcal{F}(k)|^2}{|\mathcal{F}(k)|^2 + c} \cdot \frac{\mathcal{F}(b)}{\mathcal{F}(k)}\right)$$

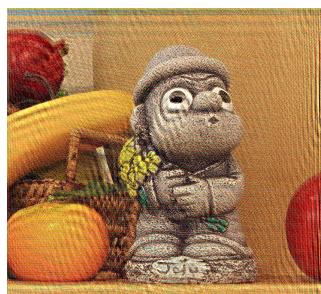
- 채널별 연산 결과에 255를 곱하고 다시 합쳐 최종 결과 이미지를 생성했다.

## 1.2. Result

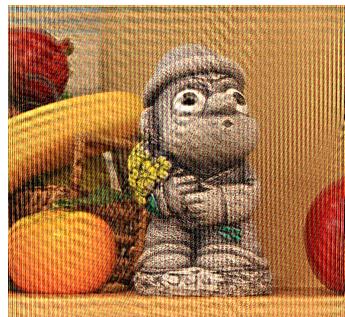
•  $c = 0$

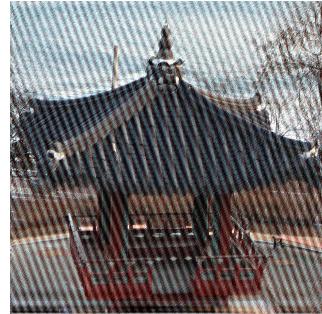
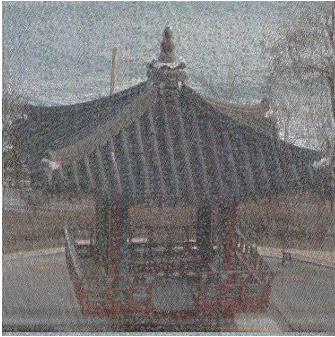


•  $c = 0.001$



•  $c=0.01$





- c 값에 따른 결과물이다.

## 1.2. Discussion

### • Parameter c

- wiener deconvolution에서는 least expected squared error를 이용해 해를 구한다.
  - 따라서  $\min E[\|L - HB\|^2]$ 인 H(deconvolution filter)를 찾는 것이 목적이다.
  - H: deconvolution filter
  - B: blurred image
  - L: latent sharp image
- $B = KL + N$ 이므로  $\min E[\|L - H(KL + N)\|^2] = \min E[\|(1 - HK)L - HN\|^2]$  이 된다. 최적 해를 구하기 위해 위의 식을 H에 대해 미분한 값이 0이 될 때를 찾으면

$$H = \frac{K^2}{K^2 + \frac{E[\|N\|^2]}{E[\|L\|^2]}} \cdot \frac{1}{K} \text{ 가 된다.}$$

H와 B를 곱하면 최적 L을 구할 수 있다. 따라서  $L = HB = \frac{K^2}{K^2 + \frac{E[\|N\|^2]}{E[\|L\|^2]}} \cdot \frac{B}{K}$ ,

$l = F^{-1}(L) = F^{-1}(HB) = F^{-1}\left(\frac{K^2}{K^2 + \frac{E[\|N\|^2]}{E[\|L\|^2]}} \cdot \frac{B}{K}\right)$  가 된다. (여기서 F는 푸리에 변환)

▪  $\frac{E[\|N\|^2]}{E[\|L\|^2]}$ 는 SNR(signal-to-noise ratio at each frequency  $\omega$ )을 이용해 정의할 수 있으며  $\frac{1}{SNR(\omega)} = \frac{\text{noise variance at } \omega}{\text{signal variance at } \omega}$  가 된다. 대부분의 이미지는 signal variance가 1로 스케일링되는 경향이 있고, noise variance는 일정하기 때문에 모든  $\omega$ 에 대해  $\sigma(\omega) = \omega^2 N$ 로 가정할 수 있다. 그러면  $SNR(\omega) = 1/c\omega^2$ 로 가정할 수 있고, 이를 활용해 식을 단순화하면 아래와 같은 식으로 나타낼 수 있다.

$$l = \mathcal{F}^{-1}\left(\frac{|\mathcal{F}(k)|^2}{|\mathcal{F}(k)|^2 + c} \cdot \frac{\mathcal{F}(b)}{\mathcal{F}(k)}\right)$$

- 따라서 c 값은 SNR을 조절하는 parameter이며, signal과 noise 간의 균형을 조절하는 데 영향을 미친다고 할 수 있다.
- 위의 결과 이미지들을 통해 c에 따른 변화를 살펴보자.
  - c=0일 때는 inverse filtering과 동일하며, c 값에 따라 노이즈의 정도가 달라지는 것을 확인할 수 있다. inverse filtering인 c=0일 때에 비해서는 모든 이미지가 더 적은 노이즈를 갖는다는 점은 동일하다.
  - 다만, 이미지의 종류에 따라 같은 c 값이라도 퀄리티가 다르게 나타나는 경우가 있다. 예를 들어, boy\_statue 이미지의 경우 c=0.01일 때가 가장 노이즈가 적고 높은 퀄리티인 것처럼 보이지만 harubang 이미지의 경우 c=0.01일 때와 c=0.1일 때의 퀄리티 차이가 크지 않으며 오히려 c=0.01일 때 물결 패턴이 더 강하게 보여 퀄리티가 낮은 것처럼 보인다.

### • Limitation

- 높은 퀄리티의 결과물을 얻기 위해서는 c 값을 적절히 택하는 것이 중요한데, 이를 미리 알 수 있는 방법이 없어 수동적으로 확인 후 결정해야한다는 한계가 있다. 또한, 이미지의 종류에 따라 최적값이 다르기 때문에 매번 새로운 최적값을 찾아야 한다는 문제도 있다.

## 2. TV deconvolution

### 2.1. Implementation

```
def tv_deconv(file_name):
    blurred = cv2.imread('../images/examples/' + file_name + '.jpg', cv2.IMREAD_COLOR).astype(np.float32)
    kernel = cv2.imread('../images/examples/' + file_name + '_out.jpg.psf.png', 0).astype(np.uint8)
    results = []
    psf = kernel / sum(sum(kernel))
    b, g, r = cv2.split(blurred)
    for x in [b, g, r]:
        results.append(255 * deconv_L2(x, x, psf, 0.0001))
    result_image = cv2.merge((results[0], results[1], results[2]))
    cv2.imwrite('../images/outputs/tv/' + file_name + '.jpg', result_image)
```

- blurred image와 blur kernel 이미지를 각각 불러오고, float로 형변환을 해준 뒤 255.0으로 나누어 normalization을 해준다.
- 채널별로 연산을 수행하기 위해 blurred image는 R, G, B 채널로 분리해준다.
- kernel 이미지는 컨볼루션 연산을 수행한 뒤에 이미지의 밝기가 변하지 않게끔 하기 위해 `kernel / sum(sum(kernel))`를 통해 normalize를 해줬다.
- 함수 `deconv_L2()`를 정의해 deconvolution을 수행한 뒤 다시 R, G, B 채널을 합쳐 최종 이미지를 생성했다.
- deconv\_L2()**

```
def deconv_L2(blurred, latent0, psf, reg_strength):
    img_shape = blurred.shape
    latent = latent0.copy()
    # compute b
    b = fftconv2(blurred, np.rot90(psf, 2)) # (K^T)b
    b = b.flatten()

    # set x
    x = latent.flatten()

    # run conjugate gradient
    result = conjgrad(x, b, 25, 1e-4, Ax, psf, img_shape, reg_strength)
    latent = result.reshape(img_shape)
    return latent
```

- sample code를 참고하여 작성했다.
- Iterative Re-weighted Least Squares (IRLS) method를 사용하며, 아래와 같은 least squares problem을 conjugate gradient method를 활용해 풀며 최적해를 구했다.
$$\begin{aligned} l^* &= \underset{l}{\operatorname{argmin}}(||b - Kl||^2 + \alpha(l^T D_x^T W_x D_x l + l^T D_y^T W_y D_y l)) \\ &\Rightarrow (K^T K + 2\alpha(D_x^T W_x D_x + D_y^T W_y D_y))l = K^T b \end{aligned}$$
- conjgrad 함수는  $Ax = b$  꼴의 least squares problem을 해결하는 함수로, 위의 식을 풀도록 하기 위해 x에는 latent를, b에는  $K^T b$ 를 `fftconv2(blurred, np.rot90(psf, 2))`와 같이 계산해 넣어주었다.
  - fftconv2()**는 이미지와 커널의 convolution 연산을 frequency domain에서 수행한 뒤 다시 푸리에 역변환을 통해 spatial domain으로 바꾸어 반환하는 함수이다. 아래와 같이 정의하여 사용했다.

```
def fftconv2(img, psf):
    return np.real(np.fft.ifft2(np.fft.fft2(img) * psf2otf(psf, (img.shape[0], im
```

- conjgrad()**

```

def Ax(x, psf, img_shape, reg_strength):
    psf_conj = np.rot90(psf, 2)
    weight_x = np.ones(img_shape)
    weight_y = np.ones(img_shape)
    dxf = np.array([0, -1, 1])
    dyf = np.array([[0], [-1], [1]])

    x = x.reshape(img_shape)
    y = fftconv2(fftconv2(x, psf), psf_conj) # ( $K^T K$ )
    y += reg_strength*cv2.filter2D(weight_x*x, -1, dxf) # ( $\{D_x\}^T W_x D_x$ ) $l$ 
    y += reg_strength*cv2.filter2D(weight_y*x, -1, dyf) # ( $\{D_y\}^T W_y D_y$ ) $l$ 
    # Compute Ax
    return y.flatten()

def conjgrad(x, b, maxIt, tol, Ax_func, psf, img_shape, reg_strength):
    r = b - Ax_func(x, psf, img_shape, reg_strength)
    #  $K^T b - (K^T K + (\{D_x\}^T W_x D_x) + (\{D_y\}^T W_y D_y))l$ 
    p = r.copy()
    rsold = np.sum(r * r) # 0이 되어야 최적해

    for _ in range(1, maxIt + 1):
        Ap = Ax_func(p, psf, img_shape, reg_strength)
        alpha = rsold / np.sum(p * Ap)
        r = r - alpha * Ap
        rsnew = np.sum(r * r)
        x = x + alpha * p

        if np.sqrt(rsnew) < tol:
            break

        p = r + rsnew / rsold * p
        rsold = rsnew

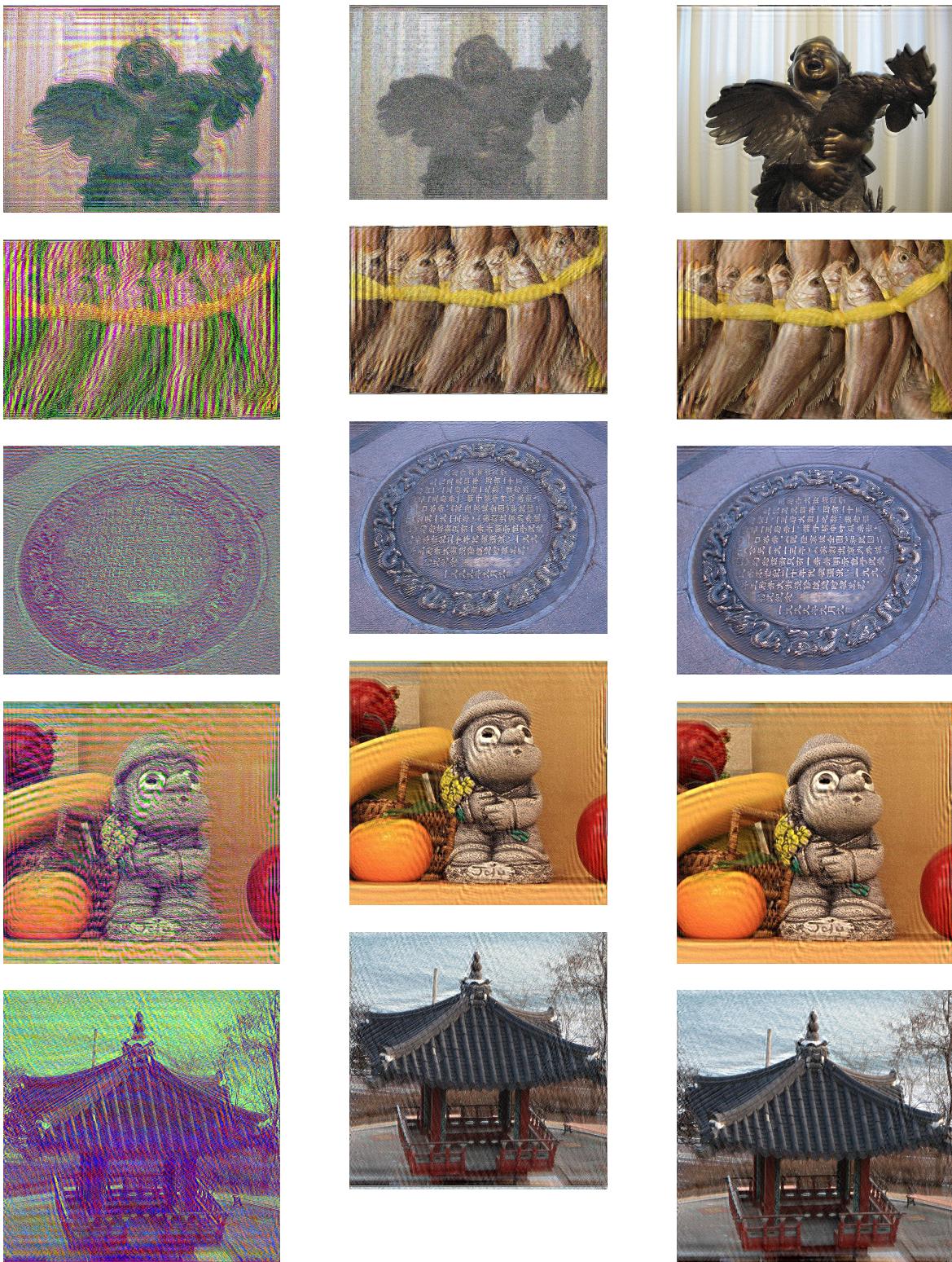
    return x

```

- sample code를 참고해 작성했다.
- Ax\_func는  $x$ (latent, I)와 psf(kernel, K)를 입력으로 받아  $(K^T K + \alpha(D_x^T W_x D_x) + (D_y^T W_y D_y))l$ 를 계산하는 함수이다.
- $((D_x^T W_x D_x) + (D_y^T W_y D_y))l$ 를 계산할 때 속도를 빠르게 하기 위해 opencv에서 제공하는 함수 cv2.filter2D를 이용해 컨볼루션 계산을 수행해주었다.
- $r$ 은  $(K^T K + 2\alpha(D_x^T W_x D_x + D_y^T W_y D_y))l - K^T b$ 를 계산한 것과 동일하므로, `rsold = np.sum(r * r)` 이 0이 될 때가 최적해가 된다. 따라서 threshold(tol)보다 작아지지 않으면 반복하여 연산을 수행하며  $Ax = b$  꼴에서의 최적해  $x$ 를 찾는다.

## 2.2. Results

- reg\_strength = 0.01, maxIt = 25
- reg\_strength = 0.001, maxIt = 25
- reg\_strength = 0.0001, maxIt = 25



### 2.3. Discussion

- Energy function & MAP
  - MAP to Energy function

▪ Non-blind deconvolution에서는  $b$ 와  $k$ 가 주어졌을 때, latent sharp image  $l$ 을 찾는 것이 목적이다. 따라서 MAP를 사용해 수식으로 나타내면  $\max_l P(l|b, k) = \min_l -\log P(l|b, k)$ 가 되고 이를 만족하는  $l$ 을 찾는 것이 목적이다. 수

식을 변형하면,  $P(\mathbf{l}|\mathbf{b}, \mathbf{k}) \propto P(\mathbf{b}|\mathbf{l}, \mathbf{k})P(\mathbf{l})$  이므로  $\min_l (-\log P(\mathbf{l}|\mathbf{b}, \mathbf{k})) = \min_l (-\log P(\mathbf{b}|\mathbf{l}, \mathbf{k}) - \log P(\mathbf{l}))$ 가 된다.

- 에너지 함수는 목적함수이므로  $E(\mathbf{l}) = -\log P(\mathbf{b}|\mathbf{l}, \mathbf{k}) - \log P(\mathbf{l})$ 와 같이 나타낼 수 있다.
  - 여기서  $P(\mathbf{b}|\mathbf{l}, \mathbf{k})$ 는  $\mathbf{l}, \mathbf{k}$ 가 주어졌을 때  $\mathbf{b}$ 에 대한 확률이다. 이는  $\|b - k * l\|^2$ 가 최소가 되면 최대화 된다. 따라서  $-\log P(\mathbf{b}|\mathbf{l}, \mathbf{k}) \propto \|b - k * l\|^2$  이므로  $-\log P(\mathbf{b}|\mathbf{l}, \mathbf{k})$ 를  $c * \|b - k * l\|^2$ 로 나타낼 수 있고 이때  $c$ 는 임의의 상수( $c > 0$ )이다.
  - $P(\mathbf{l})$ 은 latent 이미지에 대한 prior이자 regularization term으로, TV deconvolution에서는 total variation을 이용한다. total variation은 아래와 같이 정의된다. total variation이 최소가 되면 필요한 경계선은 유지하면서 필요 없는 noise들을 제거하게 되어  $P(\mathbf{l})$ 은 최대가 된다. 따라서  $-\log P(\mathbf{l}) \propto E_{TV}(l)$  이므로  $-\log P(\mathbf{l})$ 를  $d * E_{TV}(l)$ 로 나타낼 수 있고 이때  $d$ 는 임의의 상수( $d > 0$ )이다.

$$E_{TV}(l) = \|\partial x * l\|_1 + \|\partial y * l\|_1 \\ (\partial x = [0, -1, 1], \partial y = [0, -1, 1]^T)$$

- 최종적으로 대입하면 에너지 함수는 아래와 같이 정의할 수 있다.

$$\begin{aligned} E(l) &= c * \|b - k * l\|^2 + d * E_{TV}(l) \\ &= \|b - k * l\|^2 + d/c * E_{TV}(l) \\ &= \|b - k * l\|^2 + \alpha * E_{TV}(l) \end{aligned}$$

- 상수를 줄이기 위해 식을  $c$ 로 나누어주면  $d/c$ 가  $E_{TV}$  앞에 곱해지게 되며 이를  $\alpha$ 로 치환할 수 있다. 이 과정을 통해  $\alpha$ 가 유도되는 것이다. 그리고 이  $\alpha$ 는 regularization의 강도를 조정하는 parameter가 된다.
- **Limitation**
  - 계산 과정에서 non-linear optimization 문제를 해결해야 하며, 연산을 반복적으로 수행하며 최적해를 찾아야 한다. 따라서 계산 효율성 측면에서 속도가 느려질 수 있다는 단점이 있다. 특히 사이즈가 큰 이미지에 대해서는 속도 저하가 더욱 클 것이다. 또한 regularization term에 곱해지는 매개변수 alpha를 적절히 설정해주어야 높은 퀄리티의 결과물을 얻을 수 있는데, 이를 찾기 어렵다는 문제도 있다.