

# CSED551 PA#5

## :Deconvolution

20220848 Minsu Sun

December 20, 2024

## Code

이번 Deconvolution을 구현하는 과정에서 사용된 함수의 설명이다.

### Wiener Deconvolution

```
179 def wiener_deconv(b: np.ndarray, k: np.ndarray, c: float):
180     blurred_img = b.astype(np.float32) / 255
181     kernel = k.astype(np.float32) / np.sum(k)
182
183     result = np.zeros_like(b).astype(np.float32)
184
185     F_k = psf2otf(kernel, blurred_img.shape[:2]) # Degradation Function
186
187     for idx in range(blurred_img.shape[2]):
188         F_b = fft2(blurred_img[:, :, idx])
189         F_l = (
190             (F_k**2 / (F_k**2 + c)) * F_b / F_k
191         ) # Wiener Deconvolution Formula: Minimum MSE Filter
192         result[:, :, idx] = np.real(ifft2(F_l)) * 255
193
194     return np.clip(result, 0, 255)
```

wiener\_deconv

위 코드는 주어진 blurred image와 kernel을 이용하여 Wiener Deconvolution을 진행한다. Wiener Deconvolution의 식에 따라 Frequency 도메인에서 blur된 이미지와 커널 필터 이미지를 이용해 deconvolution을 진행한다.

### TV Deconvolution

다음은 예제로 주어진 `demo_L2.py`에서 가져온 코드 중 수정한 부분이다.

```
90 def deconv_L2(
91     blurred: np.ndarray,
92     latent0: np.ndarray,
93     psf: np.ndarray,
94     reg_strength: float,
95     tau: float,
96 ) -> np.ndarray:
97     """
98     Perform non-blind deconvolution using an L2 norm on the image gradient.
99
100    Parameters:
101    blurred (np.ndarray): The blurred image.
102    latent0 (np.ndarray): Initial solution for the latent image.
103    psf (np.ndarray): Blur kernel (point spread function).
104    reg_strength (float): Regularization strength.
105    weight_x (np.ndarray): Weight map for the x-direction gradient.
106    weight_y (np.ndarray): Weight map for the y-direction gradient.
```

```

107
108     Returns:
109     np.ndarray: The deblurred image.
110     """
111     img_size = blurred.shape
112
113     # Image gradient filters
114     dxf = np.array([[0, -1, 1]])
115     dyf = np.array([[0], [-1], [1]])
116
117     latent = latent0.copy()
118
119     # Compute the conjugate of the PSF by flipping it upside-down and left-and-right
120     psf_conj = np.flip(psf)
121
122     # compute b by convolving the blurred image with the conjugate PSF
123     # K^T \cdot b
124     b = fftconv2(blurred, psf_conj)
125     b = b.ravel()
126
127     # set x as the raveled latent image
128     x = latent.ravel()
129
130     # Parameters for the conjugate gradient method
131     cg_param: Dict[str, np.ndarray] = {
132         "psf": psf,
133         "psf_conj": psf_conj,
134         "img_size": img_size,
135         "reg_strength": reg_strength,
136         "dxf": dxf,
137         "dyf": dyf,
138         "tau": tau,
139     }
140
141     # Run the conjugate gradient method
142     x = conjgrad(x, b, 25, 1e-4, Ax, cg_param)
143
144     # Reshape the solution back to the original image size
145     latent = x.reshape(img_size)
146     return latent
147
148
149 def Ax(x: np.ndarray, p: Dict[str, np.ndarray]) -> np.ndarray:
150     """
151     Matrix-vector multiplication used in the conjugate gradient method.
152
153     Parameters:
154     x (np.ndarray): Input vector.
155     p (Dict[str, np.ndarray]): Parameters containing PSF, weights, and gradients.
156
157     Returns:
158     np.ndarray: Result of the multiplication.
159     """
160     x = x.reshape(p["img_size"])
161     # Convolve with PSF and its conjugate
162     y = fftconv2(fftconv2(x, p["psf"]), p["psf_conj"])
163
164     W_x = correlate(x, p["dxf"], mode="wrap")
165     W_x = 1 / np.vectorize(lambda x: max(x, p["tau"]))(np.abs(W_x))

```

```

166 W_y = correlate(x, p["dyf"], mode="wrap")
167 W_y = 1 / np.vectorize(lambda x: max(x, p["tau"]))(np.abs(W_y))
168
169 y += p["reg_strength"] * convolve(
170     W_x * correlate(x, p["dxf"], mode="wrap"), p["dxf"], mode="wrap"
171 )
172 y += p["reg_strength"] * convolve(
173     W_y * correlate(x, p["dyf"], mode="wrap"), p["dyf"], mode="wrap"
174 )
175
176 return y.ravel()

```

Modified Code

Iterative Reweighted Least Squares 방법을 적용하기 위해 Weight 계산을 진행하도록 수정하였다.  $Ax = b$ 의 형태를 가지도록 하여 Sample Code에서 제공한 Gradient Conjugate Solver를 사용하도록 하였다. `cg_param`으로 넘겨주던 고정된 Weight가 아닌 매번 iteration마다 이를 계산하여 사용한다. Weight를 설정할 때 필요한  $\tau$  또한 추가하여 이를 기반으로  $W_{x,y}$  행렬을 구성하도록 하였다. (원래의 IRLS 방법은 HW x HW Matrix를 사용하여  $W_{x,y}$ 를 나타내지만, 메모리 제한으로 인해 H x W Matrix로 나타내어 Sparse한 Matrix를 Dense하게 사용하였다.)

다음은 위의 수정된 함수들을 이용하여 TV Deconvolution을 진행하는 함수이다.

```

197 def TV_deconv(b, k, tol=0.001, tau=1e-4):
198     blurred_img = b.astype(np.float32) / 255
199     kernel = k.astype(np.float32) / np.sum(k)
200
201     result = np.zeros_like(b)
202
203     for idx in range(blurred_img.shape[2]):
204         img = blurred_img[:, :, idx]
205         result[:, :, idx] = (
206             deconv_L2(img, img, kernel, tol, tau) * 255
207         ) # fix tau for typical 1e-4
208     return np.clip(result, 0, 255)

```

TV\_deconv

주어진 RGB 채널을 각 채널로 분리하여 Deconvolution을 적용한 이후 다시 합쳐 이미지를 구성하였다. IRLS 방법을 위한  $\tau$  또한 같이 전달할 수 있도록 수정하였다.

## Discussion - Wiener Deconvolution

### Parameter $c$

Wiener Deconvolution의 목표는 아래와 같다. (Frequency Domain)

$$\min_H E[\|L - HB\|^2]$$

이는 아래와 같이 풀어 Error가 Minimum이 되는 Deconvolution Filter  $H$ 를 얻는 것을 목표로 한다. (B: Blurred Image, L: Latent Sharp Image)

$$\begin{aligned}
B &= KL + N, \\
\min_H E[\|L - H(KL + N)\|] &= \min_H E[\|(1 - HK)L - HN\|] \\
\min_H E[\|(1 - HK)L - HN\|] &= \min_H \{\|1 - HK\|^2 E[\|L\|^2] - 2(1 - HK)E[LN] + \|H\|^2 E[\|N\|^2]\}
\end{aligned}$$

우리는 Noise  $N$ 을 평균이 0인 Normal Distribution을 따를 것으로 예상함으로,  $L$ 과는 독립적이기에  $E[LN] = E[L]E[N]$ 이다. 평균을 0으로 가정하므로  $E[L]E[N]$ 은 결국 0이 된다. 이를 정리하면 아래와 같다.

$$\min_H \{\|1 - HK\|^2 E[\|L\|^2] + \|H\|^2 E[\|N\|^2]\}$$

위의 식을  $H$ 를 기준으로 미분하여 기울기가 0이 되는 지점을 찾으면 이곳이 최적의  $H$ 라는 것을 알 수 있다. 이를 다시  $H$ 로 정리하면 아래와 같다.

$$\begin{aligned}
&\Rightarrow -2(1 - HK)E[\|L\|^2] + 2HE[\|N\|^2] = 0 \\
\Rightarrow H &= \frac{KE[\|L\|^2]}{K^2 E[\|L\|^2] + E[\|N\|^2]} = \frac{K^2}{K^2 + \frac{E[\|N\|^2]}{E[\|L\|^2]}} \times \frac{1}{K}
\end{aligned}$$

$B$ 에  $H$ 를 곱해 아래와  $l$ 을 공식화할 수 있다. ( $F$ 는 Fourier Transform,  $F^{-1}$ 는 Inverse Fourier Transform을 의미한다.)

$$l = F^{-1}\left(\frac{K^2}{K^2 + \frac{E[\|N\|^2]}{E[\|L\|^2]}} \times \frac{B}{K}\right)$$

SNR는  $Var(\text{Noise})$ 와  $Var(\text{Signal})$ 의 비율로 나타나므로 위의 식을 아래와 같이 SNR의 표현으로 나타낼 수 있다.

$$l = F^{-1}\left(\frac{K^2}{K^2 + \frac{1}{SNR(\omega)}} \times \frac{B}{K}\right)$$

SNR은 기존에 예측이 되어야 하는 값이지만, Natural Image의 경우 Signal의 Variance가  $\frac{1}{\omega^2}$ 로 Scaling되며, Noise의 경우에는 보통 White Noise로 Variance가 상수인 경향을 가진다. 이를 통해 SNR을 다음과 같이 예상할 수 있다.

$$\begin{aligned}
\sigma_S &= \frac{1}{\omega^2} \\
\sigma_N(\omega) &= \text{constant} \\
SNR(\omega) &= \frac{1}{c\omega^2}
\end{aligned}$$

이에 따라 Latent Image는 다음과 같이 나타낼 수 있다.

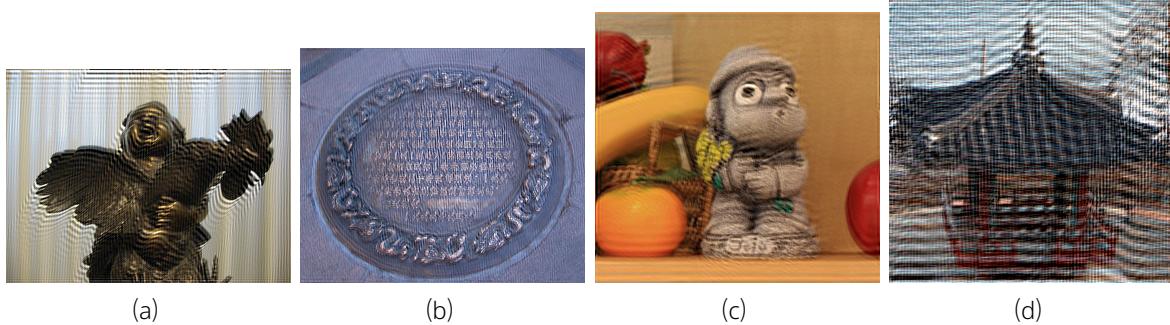
$$l = F^{-1}\left(\frac{K^2}{K^2 + c\omega^2} \times \frac{B}{K}\right)$$

이를 Spatial Domain에서의 Fourier Transform과 함께 나타내면 다음과 같이 나타낼 수 있다.

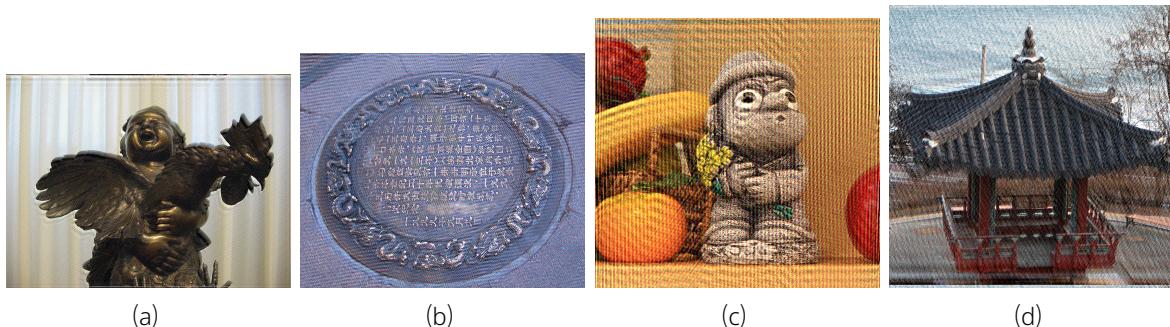
$$l = F^{-1}\left(\frac{|F(k)|^2}{|F(k)|^2 + c} \times \frac{F(b)}{F(k)}\right)$$

**결론적으로, Wiener Deconvolution에서의 상수  $c$ 는 SNR을 나타내는 상수로 사용됨을 확인할 수 있다.**

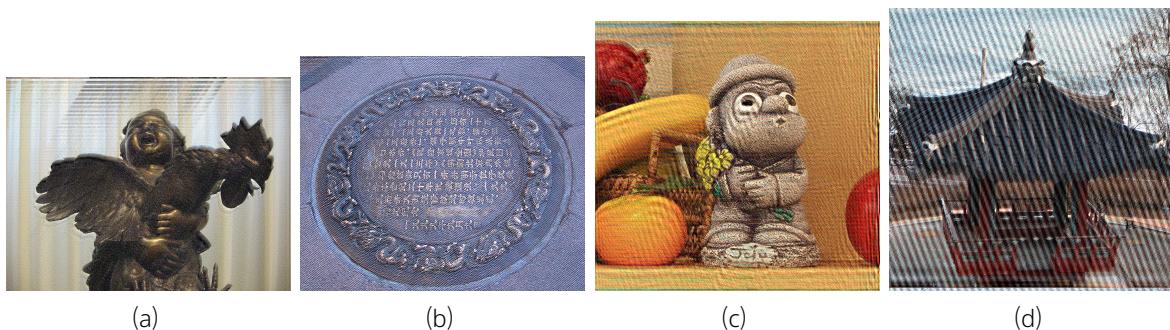
아래는 Wiener Deconvolution의 결과이다.



Result of Wiener Deconvolution( $c=0.1$ )



Result of Wiener Deconvolution( $c=0.01$ )



Result of Wiener Deconvolution( $c=0.001$ )

Parameter  $c$ 의 값에 따라서 결과가 달라지는 것을 볼 수 있다.

## Limitation

위의 정리에서 확인한 바와 같이, parameter  $c$ 는 이미지의 SNR을 추정하는 값이다. 그렇기 때문에, 이미지마다 Noise의 정도를 생각하여 이를 결정해야 한다. 다만, 이에 대한 정보가 없을 경우 여러가지의  $c$  값을 확인하여 결정해야 된다는 점이 한계점으로 생각된다. 결과에서도 확인할 수 있듯이, 특정 이미지에 특정  $c$  값이 어울리는 것을 볼 수 있으며, 다른

이미지의 경우에는 다른  $c$  값을 따르거나 혹은 아예 맞지 않을 수도 있다. 이미지의 Gradient 값을 이용해 SNR을 추정하여 이를  $c$ 의 값으로 결정하여 사용할 경우 더 나은 추정을 보일 것으로 생각된다.

## Discussion - TV Deconvolution

### MAP Problem and Energy Function $E(l)$

MAP 문제에 의하면 다음과 같은 목표를 가지고 문제를 풀도록 한다.

$$l = \max_l p(l|b, k)$$

이는 posterior probability로 Bayes Theorem에 의하여 다음과 같이 나타낼 수 있다.

$$l = \max_l p(l|b, k) = \max_l \frac{p(b|l, k)p(l)p(k)}{p(b)p(k)}$$

이에  $-\log$ 를 써워 정리하면 다음과 같다.

$$\begin{aligned} l &= \max_l \frac{p(b|l, k)p(l)p(k)}{p(b)p(k)} \\ &= \min_l [-\log(\frac{p(b|l, k)p(l)}{p(b)})] \\ &= \min_l [-\log(p(b|l, k)) - \log(p(l)) + \log(p(b))] \end{aligned}$$

이 때,  $p(b)$ 는  $|l$ 에 따라 변화하지 않으므로 상수로 처리가능하며,  $b = kl + n$ 이라는 Noise Model에 의해 다음과 같이 정리될 수 있다.

$$\begin{aligned} l &= \min_l [-\log(p(b|l, k)) - \log(p(l)) + \log(p(b))] \\ &= \min_l [-\log(p(b - kl|l, k)) - \log(p(l)) + C] \\ &\quad (\because (b - kl) \sim \mathcal{N}(0, \sigma^2)) \\ &= \min_l [-\log(N(b - kl|0, \sigma^2)) - \log(p(l)) + C] \\ &= \min_l [C_1 \|b - kl\|^2 - \log(p(l)) + C] \\ &= \min_l [\|b - kl\|^2 - \log(p(l))] \end{aligned}$$

MAP의 목적은 에너지 함수를 최소화 시키는 것이므로, 위의 수식은 이에 따라 Latent  $l$ 의 에너지 함수  $E(l)$ 을 다음과 같이 정의할 수 있다.

$$E(l) = \|b - kl\|^2 - \log(p(l))$$

이 때,  $l$ 의 prior  $p(l)$ 을 다음과 같이 formulate 한다.

$$p(l) = \frac{1}{Z} \exp\left(-\frac{1}{\sigma_p^2} \sum_{x,y} \|\nabla f(x, y)\|\right)$$

그럼 전체 에너지 함수  $E(l)$ 을 다음과 같이 정리할 수 있다.

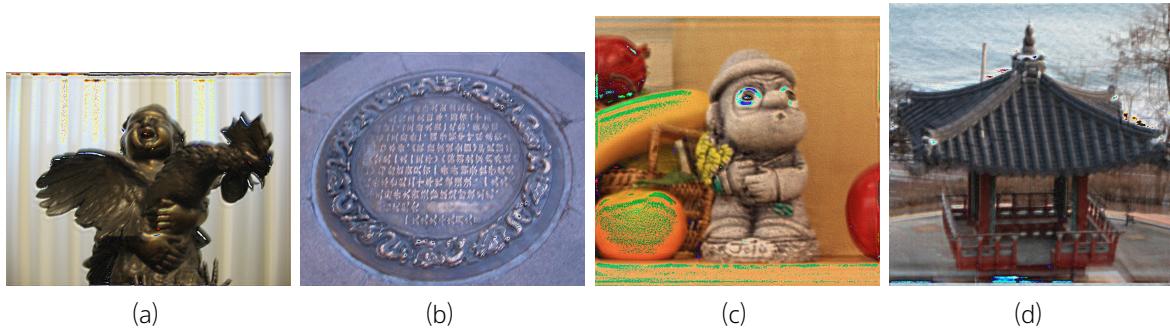
$$\begin{aligned} E(l) &= \|b - kl\|^2 - \log(p(l)) \\ &= \|b - kl\|^2 - \log\left(\frac{1}{Z} \exp\left(-\frac{1}{\sigma_p^2} \sum_{x,y} \|\nabla f(x, y)\|\right)\right) \\ &= \|b - kl\|^2 + \frac{1}{\sigma_p^2} \sum_{x,y} \|\nabla f(x, y)\| - \log \frac{1}{Z} \end{aligned}$$

결국 MAP 문제의 목표 또한 다음과 같이 정리할 수 있다.

$$\begin{aligned} l &= \min_l [-\log(p(b|l, k)) - \log(p(l)) + \log(p(b))] \\ &= \min_l [\|b - kl\|^2 + \frac{1}{\sigma_p^2} \sum_{x,y} \|\nabla f(x, y)\| - \log \frac{1}{Z}] \\ &= \min_l [\|b - kl\|^2 + \alpha \sum_{x,y} \|\nabla f(x, y)\|] \end{aligned}$$

이는 결국 MSE 문제에서의 Regularization Term으로  $l$ 의 prior  $p(l)$ 이 붙은 결과와 같다라는 사실을 확인할 수 있다.  $\alpha$ 를 통해 Regularization의 Preference를 조절하면서, 동시에  $l$ 의 prior  $p(l)$ 의 분포를 예측하는 것과 같다.

아래는 다양한  $\alpha$  값에 따른 TV Deconvolution의 결과이다.

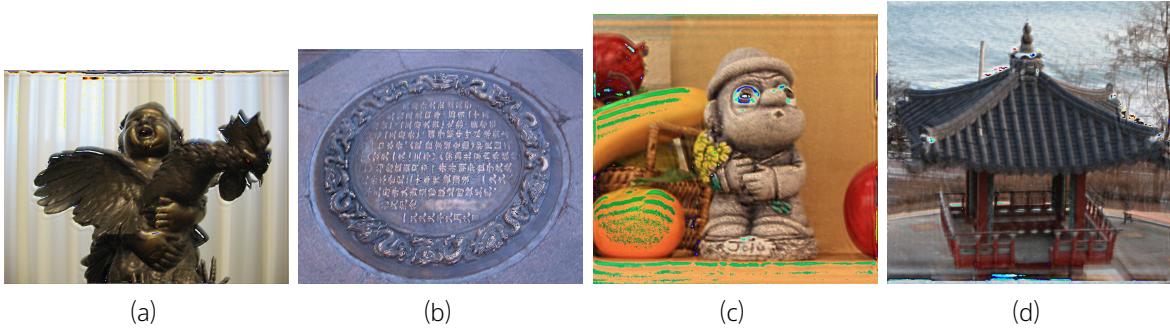


Result of TV Deconvolution( $\alpha=0.10$ )

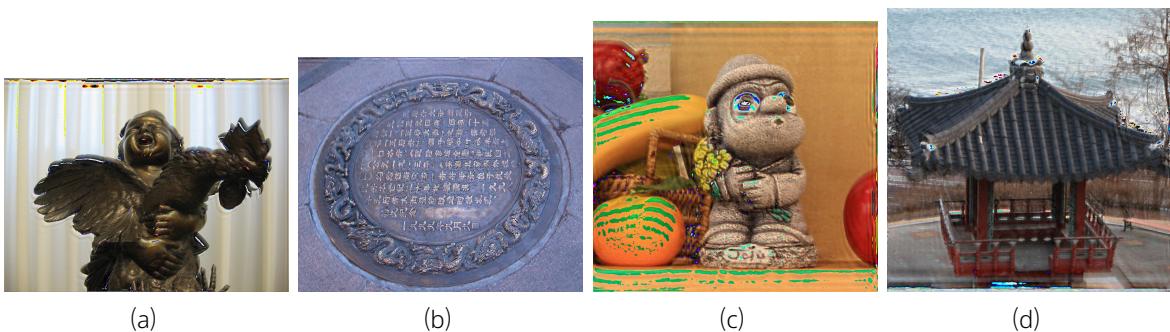
$\alpha$  값에 따라 다른 Deconvolution 결과를 보인다.

## Limitation

TV Deconvolution은 Wiener Deconvolution에 비해 정교한 Deconvolution 결과를 출력할 수 있지만, 시간이 오래 걸린다는 단점이 있다. Iteration의 수를 결정하여 적정 품질에 빠르게 도달할 수 있도록 하는 것이 중요하다고 생각된다. 또한, conjugate gradient solver의 tolerance 값과 IRLS 방법의  $\tau$  값을 적당히 조절하여 빠른 수렴을 할 수 있도록 하는 것이 중요하다. Non-Linear optimization을 해결하기 때문에, 빠른 optimization이 불가능하다는 단점이 있지만, Linear optimization으로의 근사가 가능하다면 속도를 더 개선시킬 수 있을 것으로 생각된다. 전체 이미지에 대한 gradient를 사용하기 때문에, downsampling한 이미지의 gradient를 이용해 원본 이미지의 gradient를 추정하는 방법 또한 유효할 것으로 생각된다.



Result of TV Deconvolution( $c=0.001$ )



Result of TV Deconvolution( $c=0.0001$ )