Homework 2: Image Enhancement & Image Restoration

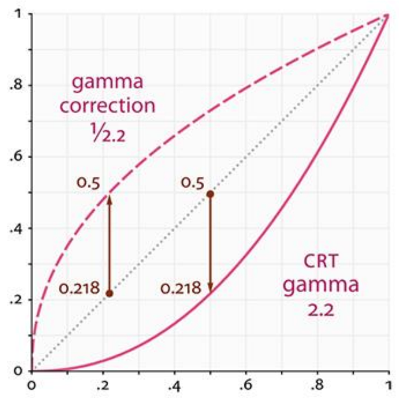
313553014 廖怡誠

Part I. Implementation:

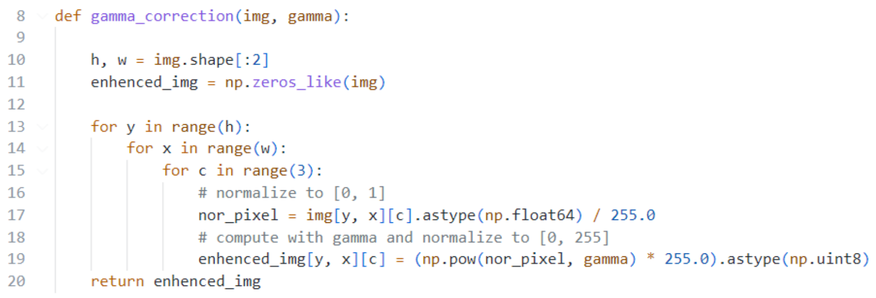
In this section, I will explain my implementations of image restoration tasks and image enhancement tasks. Image enhancement tasks includes gamma correction and histogram equalization and image restoration contains minimum mean square error (Wiener) filtering and constrained least squares restoration. Next, I will elaborate on each of these methods in the following discussion.

- **Gamma Correction**

Gamma correction involves mapping the original pixel values onto a gamma curve to adjust the brightness and darkness of an image while avoiding detail loss caused by linear adjustments. Within the pixel range of 0 to 255, the closer the original pixel value is to the midpoint (125), the greater the change when projected onto the gamma curve. Conversely, the closer the pixel value is to the extremes (0 or 255), the smaller the change, as illustrated in following picture.



In my implementation, I first normalized each pixel value to the range [0, 1] to facilitate computation. Then, I applied the gamma curve formula $P_{out} = {P_{in}}^{\gamma}$ and finally, I normalized the values back to the range [0, 255] and converted them to the uint8 data type.



```python
def gamma_correction(img, gamma):

    h, w = img.shape[:2]
    enhenced_img = np.zeros_like(img)

    for y in range(h):
        for x in range(w):
            for c in range(3):
                # normalize to [0, 1]
                nor_pixel = img[y, x][c].astype(np.float64) / 255.0
                # compute with gamma and normalize to [0, 255]
                enhenced_img[y, x][c] = (np.pow(nor_pixel, gamma) * 255.0).astype(np.uint8)
    return enhenced_img
```

- **Histogram Equalization**

The goal of Histogram Equalization is to adjust the brightness and contrast of an image. In the histogram distribution, brighter images tend to have pixel values concentrated in higher ranges, while darker images have pixel values concentrated in lower ranges. To balance brightness and contrast, we adjust the pixel values to achieve a uniform distribution, as illustrated in Figure 2.

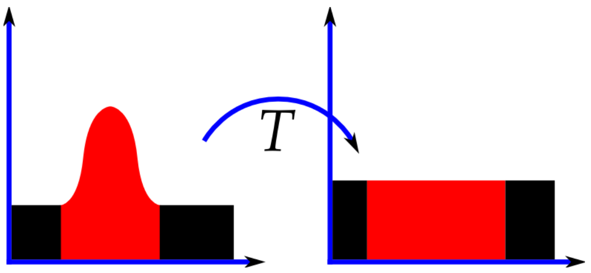


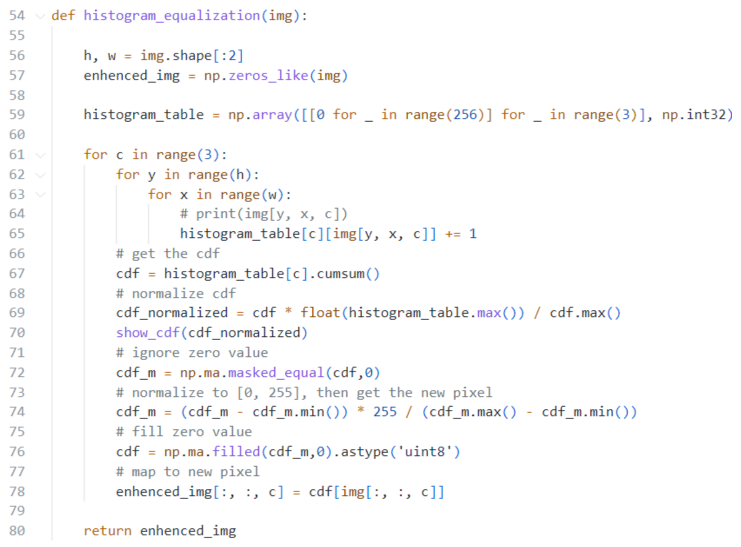
Fig 2. Histogram Equalization

In my implementation, the process is as follows:

1. Compute the frequency of occurrence of each pixel value in the image to obtain the histogram table.
2. Use the histogram table to calculate the cumulative distribution function (CDF) by summing up the occurrences of each pixel value.
3. Normalize the CDF values to the range [0, 255] using Equation (1) to generate new pixel values, ensuring a more evenly distributed brightness.

$$pixel_{new} = \frac{CDF(pixel_{org}) - \min(CDF)}{\max(CDF) - \min(CDF)} \quad (1)$$

4. Finally, map the original pixel values to the new values using the normalized CDF as a lookup table to produce the adjusted image.



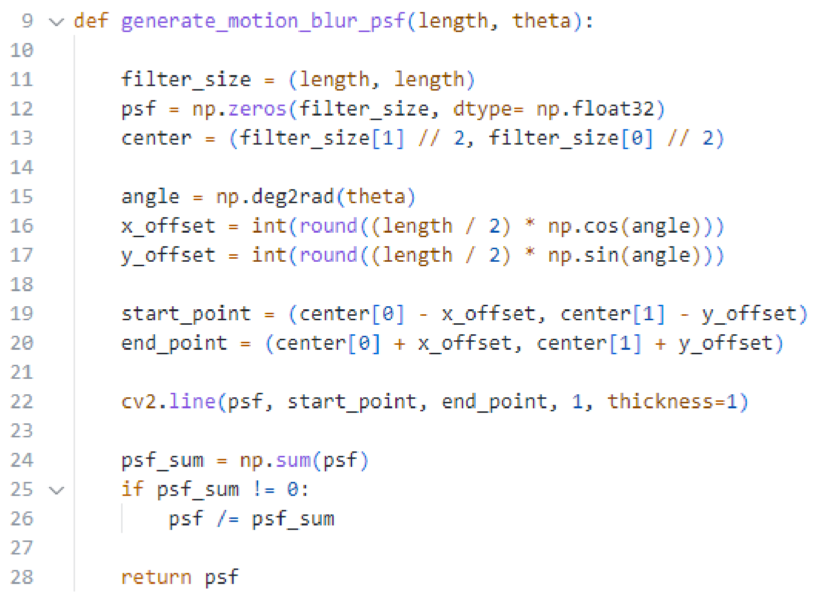
```python
def histogram_equalization(img):

    h, w = img.shape[:2]
    enhenced_img = np.zeros_like(img)

    histogram_table = np.array([[0 for _ in range(256)] for _ in range(3)], np.int32)

    for c in range(3):
        for y in range(h):
            for x in range(w):
                # print(img[y, x, c])
                histogram_table[c][img[y, x, c]] += 1
        # get the cdf
        cdf = histogram_table[c].cumsum()
        # normalize cdf
        cdf_normalized = cdf * float(histogram_table.max()) / cdf.max()
        show_cdf(cdf_normalized)
        # ignore zero value
        cdf_m = np.ma.masked_equal(cdf,0)
        # normalize to [0, 255], then get the new pixel
        cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
        # fill zero value
        cdf = np.ma.filled(cdf_m,0).astype('uint8')
        # map to new pixel
        enhenced_img[:, :, c] = cdf[img[:, :, c]]

    return enhenced_img
```

- **Motion Blur PSF Generation**

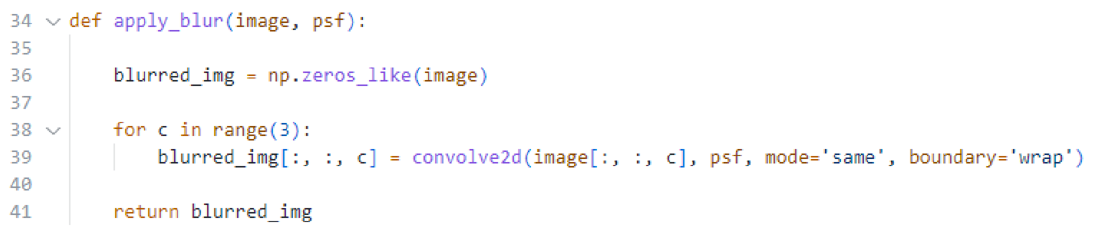
To simulate linear motion blur, I need to create a Point Spread Function (PSF) kernel that represents the motion trajectory of the blurred image. The process involves the following steps:

1. **Initialize the PSF matrix:** Create a zero matrix of size length x length to represent the PSF.
2. **Find the center:** Identify the center of the matrix, which will serve as the origin for the motion trajectory.
3. **Draw the motion trajectory:** Based on the given length and angle, draw a line within the matrix to represent the motion trajectory. This line corresponds to the motion blur direction.
4. **Normalize the PSF:** Ensure that the sum of all elements in the PSF matrix is equal to 1 to conserve image intensity during convolution.

Using this PSF, the blurred image can be restored by applying a deblurring algorithm that incorporates the corrected PSF.



```python
def generate_motion_blur_psf(length, theta):

    filter_size = (length, length)
    psf = np.zeros(filter_size, dtype= np.float32)
    center = (filter_size[1] // 2, filter_size[0] // 2)

    angle = np.deg2rad(theta)
    x_offset = int(round((length / 2) * np.cos(angle)))
    y_offset = int(round((length / 2) * np.sin(angle)))

    start_point = (center[0] - x_offset, center[1] - y_offset)
    end_point = (center[0] + x_offset, center[1] + y_offset)

    cv2.line(psf, start_point, end_point, 1, thickness=1)

    psf_sum = np.sum(psf)
    if psf_sum != 0:
        psf /= psf_sum

    return psf
```



```python
def apply_blur(image, psf):

    blurred_img = np.zeros_like(image)

    for c in range(3):
        blurred_img[:, :, c] = convolve2d(image[:, :, c], psf, mode='same', boundary='wrap')

    return blurred_img
```

- **Minimum Mean Square Error (Wiener) Filtering**

Wiener filtering is a frequency-domain processing method used to restore the original image from noisy and blurred images. Its core purpose is to suppress noise by balancing noise and signal, addressing issues that arise during deconvolution due to unstable frequencies (such as zero or near-zero values). In the frequency domain, the relationship between the observed image $G(u,v)$ and the original image $F(u,v)$ can be expressed as:

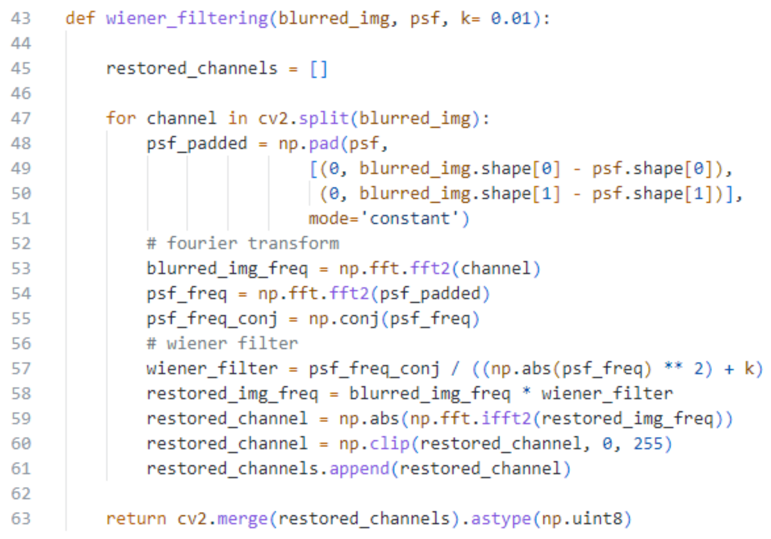
$$G(u,v) = H(u,v) \cdot F(u,v) + N(u,v)$$

where $G(u,v)$ is the observed (blurred and noisy) image in the frequency domain, $F(u,v)$ is the original (unknown) image in the frequency domain, $H(u,v)$is the frequency response of the blur (Point Spread Function, PSF), and $N(u,v)$ represents the noise in the image.

The inverse operation formula for Wiener filtering in the frequency domain is as follows:

$$F(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + SNR} \cdot G(u,v)$$

where $F(u,v)$ is the estimated original image in the frequency domain, $H^*(u,v)$ is the complex conjugate of $H(u,v)$, and SNR is the ratio of the noise to signal power ratio. In my implementation, I set the SNR to 0.01, which is k, and follow the above formula.



```python
def wiener_filtering(blurred_img, psf, k= 0.01):

    restored_channels = []

    for channel in cv2.split(blurred_img):
        psf_padded = np.pad(psf,
                            [(0, blurred_img.shape[0] - psf.shape[0]),
                             (0, blurred_img.shape[1] - psf.shape[1])],
                            mode='constant')
        # fourier transform
        blurred_img_freq = np.fft.fft2(channel)
        psf_freq = np.fft.fft2(psf_padded)
        psf_freq_conj = np.conj(psf_freq)
        # wiener filter
        wiener_filter = psf_freq_conj / ((np.abs(psf_freq) ** 2) + k)
        restored_img_freq = blurred_img_freq * wiener_filter
        restored_channel = np.abs(np.fft.ifft2(restored_img_freq))
        restored_channel = np.clip(restored_channel, 0, 255)
        restored_channels.append(restored_channel)

    return cv2.merge(restored_channels).astype(np.uint8)
```

- **Constrained Least Squares Restoration**

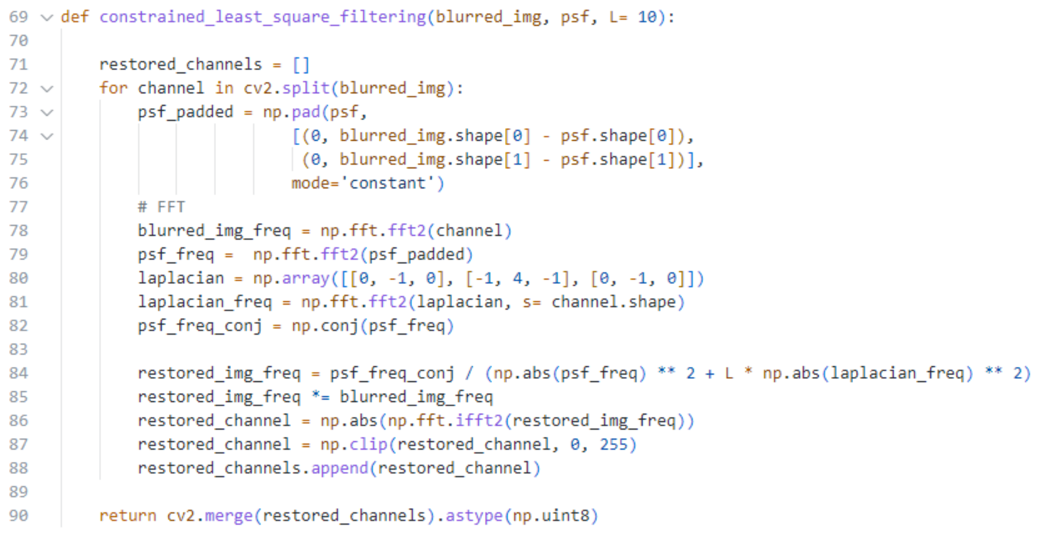
CLS (Constrained Least Squares) is a deblurring technique based on constrained optimization, aiming to find the best solution that matches the observed image while controlling the smoothness of the restored image to avoid noise amplification. The goal of CLS is to stabilize the inverse filtering problem by adding regularization constraints. The mathematical model for CLS can be expressed as:

$$\hat{F} = arg \min_{F} \{ \|H \cdot F - G\|^2 + \lambda \|L \cdot F\|^2 \}$$

where $\hat{F}$ is the restored (deblurred) image, G is the observed (blurred and noisy) image, H is the blur operator (Point Spread Function, PSF) in the spatial domain, L is a regularization operator (often a differential operator) that enforces smoothness in the restored image. λ is a regularization parameter that controls the trade-off between the fidelity to the observed image and the smoothness of the restored image. In the frequency domain, the formula of CLS can be expressed as:

$$F'(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + \lambda|L(u,v)|^2} \cdot G(u,v)$$

where $F'(u,v)$ is the estimated original image in the frequency domain, $H^*(u,v)$ is the complex conjugate of $H(u,v)$, and $L(u,v)$ is the frequency response of the regularization operator in the frequency domain (usually corresponding to a derivative operator in the spatial domain). In my implementation, I set $\lambda = 10$ and use the Laplacian operator $L(u,v)$ in the frequency domain for the CLS filter, following the formula above.



```python
def constrained_least_square_filtering(blurred_img, psf, L= 10):

    restored_channels = []
    for channel in cv2.split(blurred_img):
        psf_padded = np.pad(psf,
                            [(0, blurred_img.shape[0] - psf.shape[0]),
                             (0, blurred_img.shape[1] - psf.shape[1])],
                            mode='constant')
        # FFT
        blurred_img_freq = np.fft.fft2(channel)
        psf_freq =  np.fft.fft2(psf_padded)
        laplacian = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
        laplacian_freq = np.fft.fft2(laplacian, s= channel.shape)
        psf_freq_conj = np.conj(psf_freq)

        restored_img_freq = psf_freq_conj / (np.abs(psf_freq) ** 2 + L * np.abs(laplacian_freq) ** 2)
        restored_img_freq *= blurred_img_freq
        restored_channel = np.abs(np.fft.ifft2(restored_img_freq))
        restored_channel = np.clip(restored_channel, 0, 255)
        restored_channels.append(restored_channel)

    return cv2.merge(restored_channels).astype(np.uint8)
```

Part II. Results & Analysis:

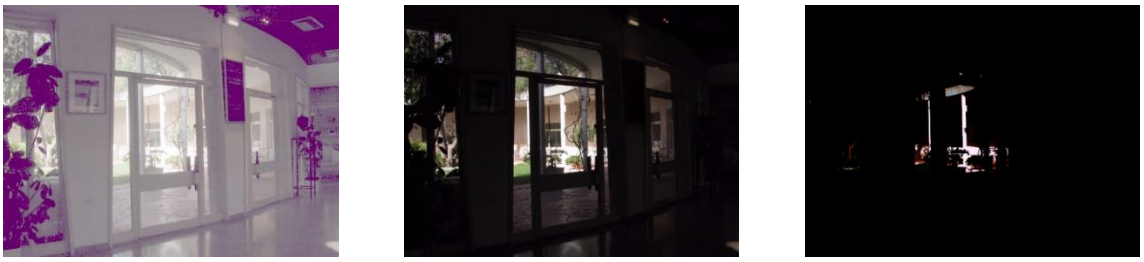
Task 1: Image Enhancement

- **Gamma Correction**

○ Please show the original image alongside the results. Try at least 3 different gamma values and compare the results

I observed that as gamma changes, the smaller the gamma, the brighter the image becomes, while the larger the gamma, the darker the image becomes. This is because when gamma is smaller, the curve formed is higher, meaning it tends to be closer to 255. Conversely, when gamma is larger, the curve tends toward 0.

Figure 1. The result of gamma changes



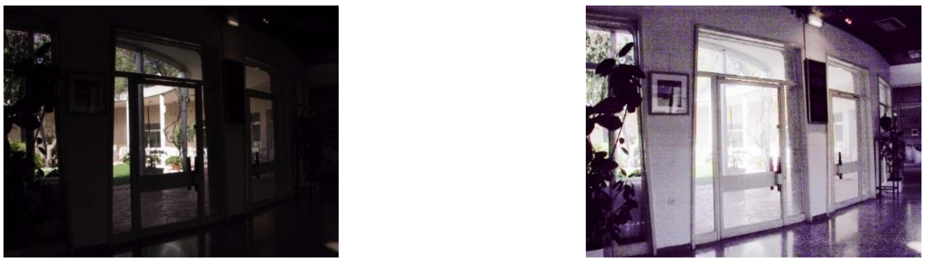
$\gamma = 0.2$ $\gamma = 1$ (Orginal image) $\gamma = 10$

- **Histogram Equalization**

○ Please show the original image alongside the results.

By averaging the distribution of the pixel values, I observe that the output image becomes lighter than the original image, but some objects still have dark colors and remain unclear. Additionally, there is a lot of noise in the output image. The reason is that the noise in the original image becomes more visible due to increased contrast. Additionally, since the pixel value distribution becomes more uniform, the differences between values are amplified, which causes some noisy pixels to be enhanced while others are suppressed.

Figure 2. The result of histogram equalization



Orginal image Enhanced image

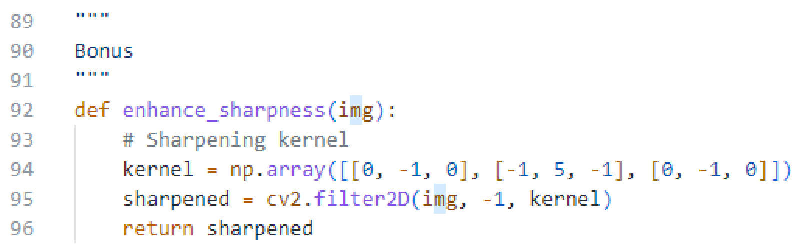
- Comparethe above two methods

By observing the results of the two methods, it can be seen that when $\gamma = 0.2$, the resulting image is overall brighter; in contrast, when $\gamma = 10$, the resulting image is overall darker. This indicates that gamma correction results in a more uneven distribution of brightness, but the noise is relatively less noticeable. However, while the histogram equalization method produces an image with a more even brightness distribution, the noise is more pronounced.

- **Bonus**

○ Please explain the used methods

I attempted to enhance image sharpness using a sharpening kernel, specifically a high-pass filter, which convolves with the image to increase the prominence of edges and fine details. This method enhances sharpness by highlighting details, making it particularly useful for images with blurry edges.

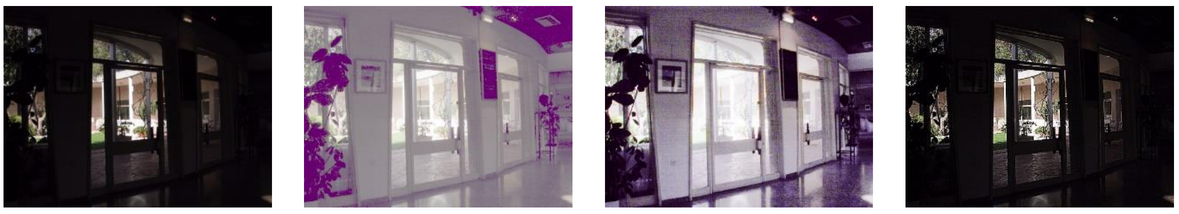


```
"""
Bonus
"""
def enhance_sharpness(img):
    # Sharpening kernel
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    sharpened = cv2.filter2D(img, -1, kernel)
    return sharpened
```

○ Please specify improvements in the image quality compared to the two enhancement methods mentioned above

The details and textures appear sharper compared to the other two methods, while the brightness remains largely unchanged.

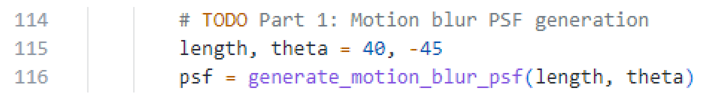
Figure 3. The comparison with sharpness method



Original image | Gamma correction | Histogram equalization | Sharpness

Task 2: Image Restoration

In this part, I set the length and angle of PSF to 40 and -45, and the PSF kernel size is set to (length by length). The results are shown in **Figure 5** and **Figure 6**.



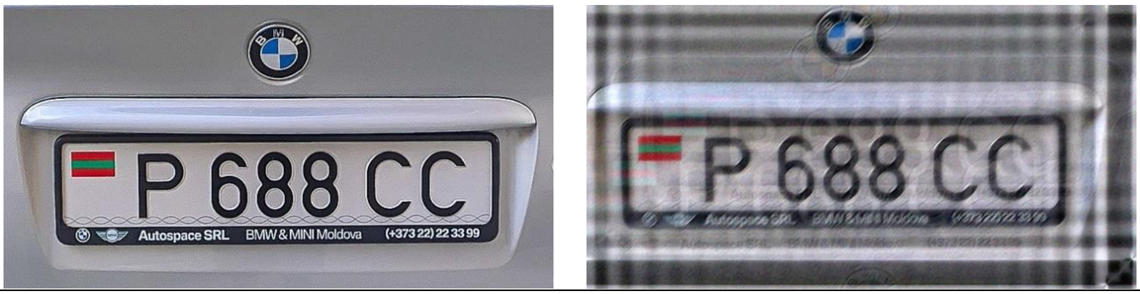
```
        # TODO Part 1: Motion blur PSF generation
        length, theta = 40, -45
        psf = generate_motion_blur_psf(length, theta)
```

● **Minimum Mean Square Error (Wiener) Filtering**

○ For both test cases, please show the original image alongside the results and also specify the used parameters (length, angle, K)

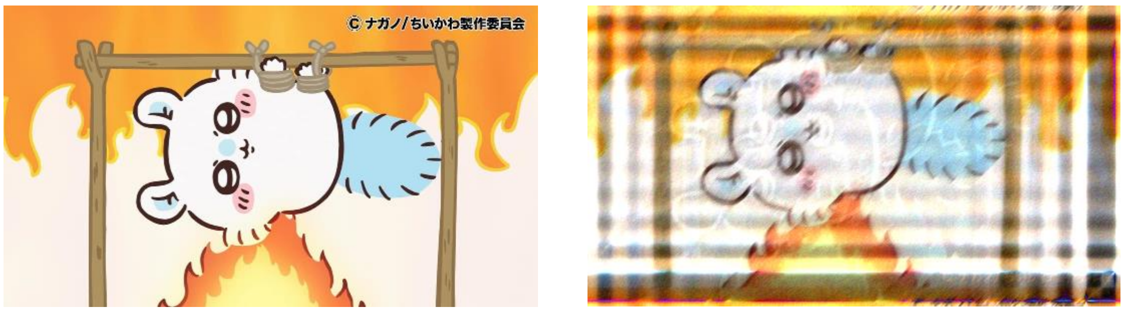
Based on the results shown in the image, the original image for testcase 1 is a car license plate with the text "P 688 CC." After applying Wiener filtering, the restored image exhibits significant distortions and stripe-like artifacts. For testcase 2, the original image depicts a cartoon character being roasted over flames, but the restored image contains grid-like distortions. The observed distortions in both cases could be due to incorrect parameter settings or an inaccurate estimation of the motion blur kernel, and we can observe **figure 6** and **figure 7**, which show the comparison of the original blurred image and PSF blurred image. They look similar but still have some micro-difference. In Wiener filtering, the effectiveness of restoration heavily depends on accurately estimating the PSF of the blur. If the length or angle of the blur is mismatched with the actual degradation, it can introduce severe artifacts, such as stripes or grids. Additionally, the K value, which controls the SNR, may have been set improperly, leading to overemphasis on certain frequencies and creating these noticeable patterns, shown in **figure 8** and **figure 9**.

Figure 4. The result of testcase 1 with Wiener Filtering



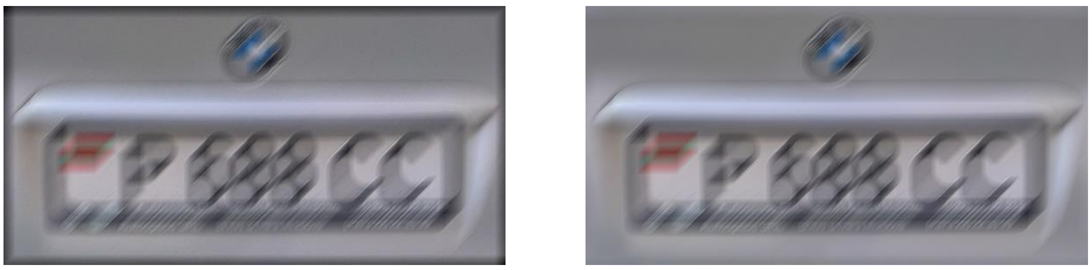
Original image Restored image

Figure 5. The result of testcase 2 with Wiener Filtering



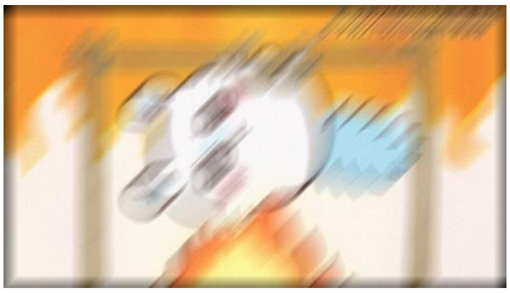
Original image Restored image

Figure 6. The comparison of blurred image of testcase 1



Original blurred image PSF blurred image

Figure 7. The comparison of blurred image of testcase 2


Original blurred image

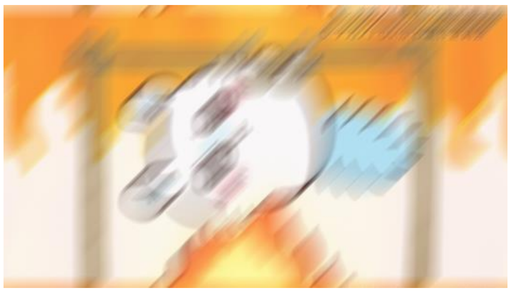

PSF blurred image

Figure 8. The comparison between different k of testcase 1.

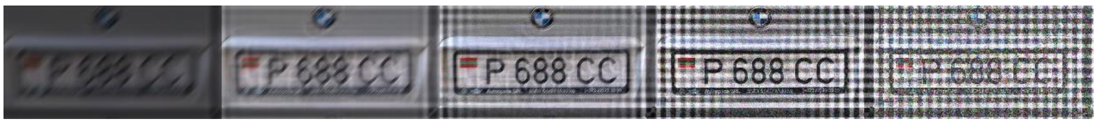
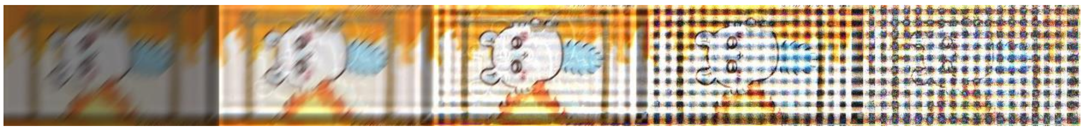

$k = 1$ $k = 0.1$ $k = 0.01$ (our method) $k = 0.001$ $k = 0.0001$

Figure 9. The comparison between different k of testcase 2.

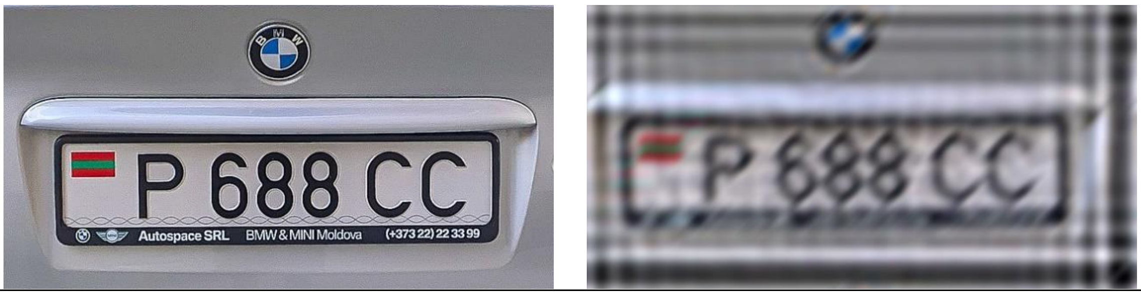

$k = 1$ $k = 0.1$ $k = 0.01$ (our method) $k = 0.001$ $k = 0.0001$

- **Constrained Least Squares Restoration**

  - For both test cases, please show the original image alongside the results and also specify the used parameters (length, angle, gamma).

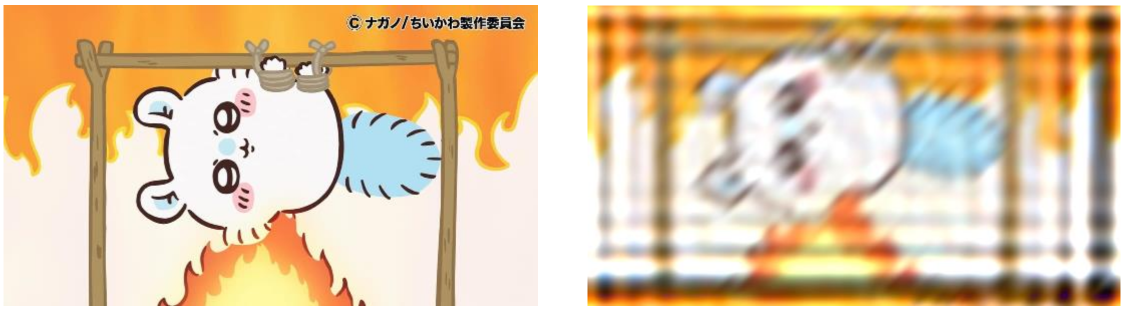
From the provided results, **Figure 10** shows that for testcase 1, the original image of the car license plate "P 688 CC" becomes heavily distorted with stripe-like artifacts after restoration using CLS, while **Figure 12** reveals that the choice of λ significantly impacts the outcome, with $\lambda = 10$ yielding the most accurate restoration by balancing noise reduction and detail preservation. Similarly, **Figure 11** demonstrates that for testcase 2, the original cartoon image is restored with severe grid-like distortions, and as shown in **Figure 13**, using $\lambda = 10$ achieves the best result, suggesting that inappropriate parameter selection, particularly λ, is a key factor behind the distortions. These results highlight the sensitivity of Wiener filtering and CLS algorithms to parameter tuning, emphasizing the need for precise adjustment to avoid over-smoothing or noise amplification.

Figure 10. The result of testcase 1 with CLS



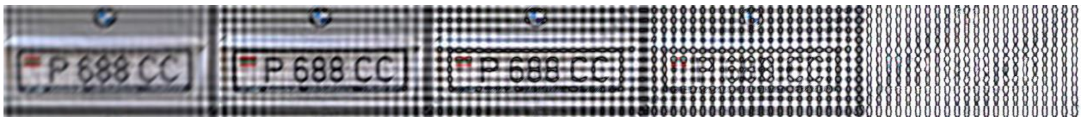
Original image | Restored image

Figure 11. The result of testcase 2 with CLS



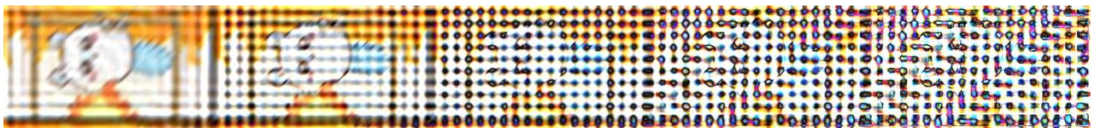
Original image | Restored image

Figure 12. The comparison between different lambda of testcase 1.



$\lambda = 10$ (our method) | $\lambda = 1$ | $\lambda = 0.1$ | $\lambda = 0.01$ | $\lambda = 0.001$

Figure 13. The comparison between different lambda of testcase 2.



$\lambda = 10$ (our method) | $\lambda = 1$ | $\lambda = 0.1$ | $\lambda = 0.01$ | $\lambda = 0.001$

- **Compare the above two methods on different test cases**

Overall, the images restored using Wiener filtering have better visual quality with less noise compared to those restored by the CLS method. As observed in **Table 1**, the PSNR values for testcase 1 and testcase 2 are approximately 11 for Wiener filtering, while the PSNR values for the CLS method are around 10. Additionally, CLS method is more challenging in terms of parameter adjustment, as **Figures 12** and **13** show that the output images are highly sensitive to changes in the λ parameter.

Table 1. The PSNR of testcase 1 and testcase 2.

	Wiener Filtering		**CLS**	
	Testcase 1	Testcase 2	Testcase 1	Testcase 2
PSNR	11.0190	11.5046	10.1872	10.5747

- **Bonus**

  - **Please explain the used methods**

I attempt to use the inverse filtering, which is the simple method of restore motion blurred image. The formula is

$$F'(u,v) = \frac{G(u,v)}{H(u,v)} \cdot$$

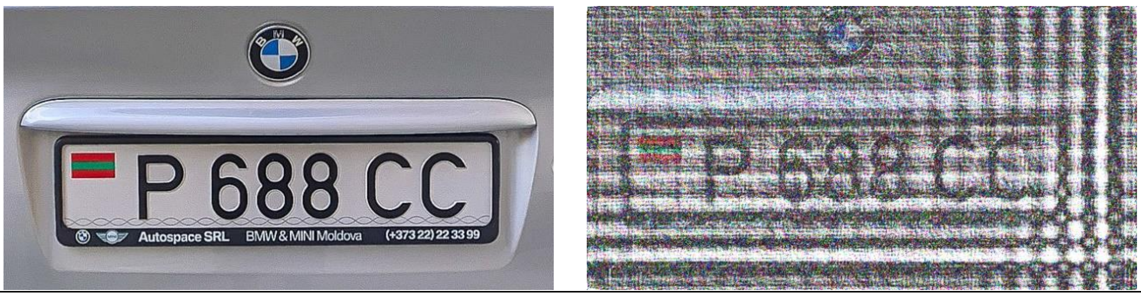
where $F'(u,v)$ is the estimated original image in the frequency domain, $G(u,v)$ is degraded image, and $H(u,v)$ is the degradation function (PSF). However, this method has drawback of sensitivity of noisy when $H(u,v)$ is small or close to zero, which leads to instability and artifacts in the restored image, therefore, we need to add a constant ϵ to solve this problem, the updated formula is

$$F'(u,v) = \frac{G(u,v)}{H(u,v) + \epsilon} \cdot$$

  - **Please specify improvements in the image quality compared to the two enhancement methods mentioned above**

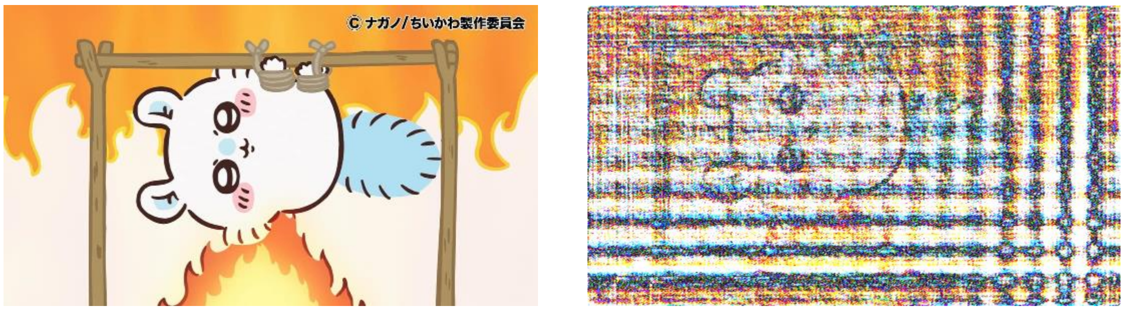
The inverse filtering method doesn't perform well when the image has a lot of noise. Compared to the other two enhancement methods, Wiener and CLS filtering can efficiently reduce noise and preserve detail. Still, inverse filtering is just appropriate to apply to images with little noise.

Figure 14. The result of testcase 1 with inverse filtering



Original image　　　　Restored image

Figure 15. The result of testcase 2 with inverse filtering



Original image

Restored image

Part III. Answer the questions:

1. Please describe a problem you encountered and how you solved it.

When I first implemented image restoration, I overlooked the issue of pixel data types when converting from the frequency domain back to the spatial domain. This caused the output image to exhibit significant abnormal noise, as shown in **Figure 16.** The problem can be resolved simply by converting the final image data type back to uint8 or by using cv2.merge().

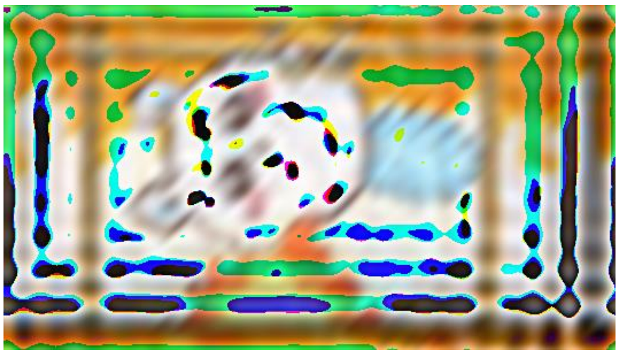


Figure 16. The wrong result with CLS.

2. What potential limitations might arise when using Minimum Mean Square Error (Wiener) Filtering for image restorations? Please suggest possible solutions to address them.

Minimum Mean Square Error (Wiener) Filtering, while effective in image restoration, has notable limitations. One key challenge is its dependency on accurate estimates of the noise and signal power spectra. If these estimates are imprecise, the filter may either over-smooth the image, losing fine details, or under-smooth, leaving residual noise. Additionally, Wiener Filtering assumes linearity and stationarity of the noise, which may not hold for complex real-world scenarios, leading to suboptimal restoration in images with spatially varying noise or non-linear distortions. To mitigate these issues, adaptive Wiener Filtering techniques can be employed, where the filter parameters are dynamically adjusted based on local image statistics.

Alternatively, integrating Wiener Filtering with advanced machine learning approaches or combining it with domain-specific priors (such as edge or texture preservation constraints) can enhance its performance in more diverse and challenging restoration tasks.

3. **What potential limitations might arise when using Constrained Least Squares Restoration for image restorations? Please suggest possible solutions to address them.**

Constrained Least Squares Restoration, while effective in balancing noise suppression and detail preservation, faces limitations primarily due to its sensitivity to the choice of the regularization parameter and the accuracy of the degradation model. If the regularization parameter is not optimally tuned, the restoration may result in over-smoothing or insufficient noise suppression, failing to preserve essential image details. Furthermore, the method assumes a precise understanding of the blurring kernel and noise characteristics, which can be difficult to obtain in practical scenarios. To address these challenges, automated parameter selection techniques, such as cross-validation or Bayesian inference, can be used to optimize the regularization parameter. Additionally, incorporating adaptive regularization methods or leveraging machine learning models to estimate degradation parameters dynamically can improve restoration performance, especially in real-world settings with varying noise and blur conditions.

Part IV. Command Line:

Image enhancement	python image_enhancement.py
Image restoration	python image_restoration.py