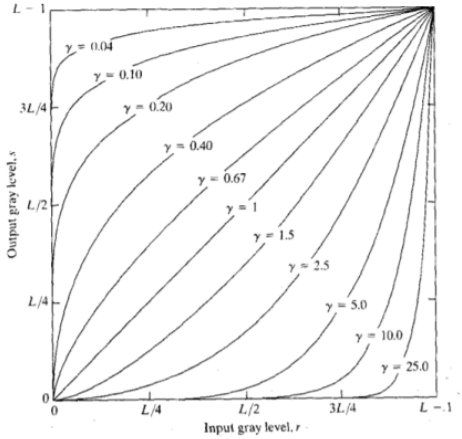
Image Process Assignment 1

310551067 吳岱容

- **Intoduction**
  The objective of this assignment is to use the knowledge we learne in class to implement image enhancement, such as contrast adjustment, color correction, and noise reduction.
  In this assignment, I implemented contrast adjustment, color correction, and noise reduction (with 3 methods).
- **Implementation**
  In this section, I will explain the methods I used.
  - Contrast adjustment
    - Power law transform



      The objective of power law transform is to adjust the bright part to be darker, and the dark part to be brighter. The gamma will decide this function.

      The general form:

      $s = c*r^{\gamma}$

      where s is output, r is input, c and γ are constant.

  - Color correction
    - Power law transform
      I use power law transform to adjust specific channels to implement color correction. By this, I can adjust the image's hue.
  - Noise reduction
    In noise reduction, I tried multiple filters and did the convolution in order to see what their effect is on different images.

- Median filter

2	2	3	2	1
3	4	3	2	3
3	5	5	6	6
4	5	6	7	5
4	4	6	6	4

3x3 median filter →

0	2	2	2	0
2	3	3	3	2
3	4	5	5	3
4	5	6	6	5
0	4	5	5	0

This method is to rank the masking digits and pick up the median number. Therefore, this method is usually used when the color of the noise is so different from the original image.

- Adaptive filter

Adaptive Expression:

$$\hat{f}(x,y)=g(x,y)-(\sigma_\eta^2/\sigma_L^2)[g(x,y)-m_L]$$

where σ_L^2 - Local variance of the local region

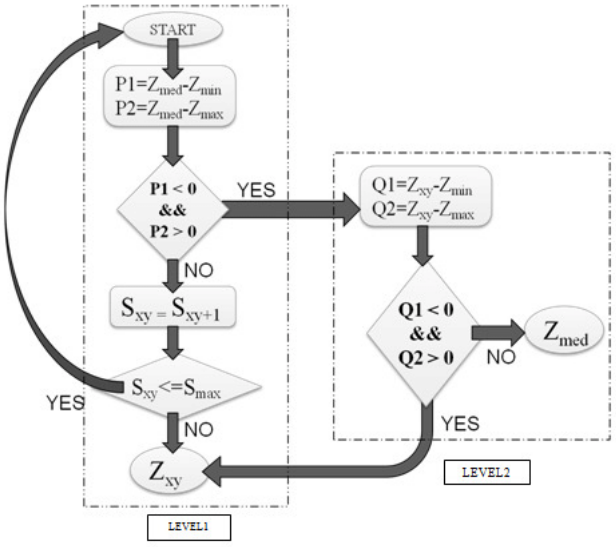
m_L - Local Mean

σ_η^2 -Variance of overall noise

g(x,y)- Pixel value at the position (x,y)

Adaptive filter is used to enhance or restore data by removing noise without significantly blurring the structures in the image.

- Adaptive mediam filter



Adaptive median filter compares each pixel to the surrounding ones. If the pixel differs a lot, it will be treated as noise and be replaced with the median value.

- **Result and Discussion**
  - Contrast
    - Power law transform
      - Results
        - Original picture

        - gamma = 0.5

        - gamma = 2.2

      - Discussion
        When gamma < 1, the picture seems to be brighter, and the contrast is also lower. Otherwise, when gamma > 1, the picture overall becomes darker, but the contrast is more obvious.
  - Color correction
    - Power law transform
      - Results

- Original picture

- gamma = 0.5 in R channel

- gamma = 0.5 in G channel

- gamma = 0.5 in B channel

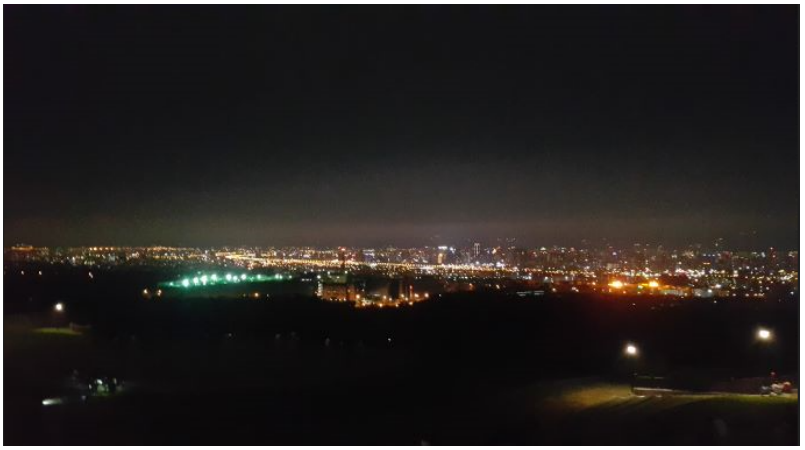
- Discussion
  As from the results, we can use power law transform to adjust the hue of the images.

- Noise reduction
  In this section, the results of 3 filters will be shown. The picture included 2 noise-added pictures (one is with Gaussian noise, and the other is with impulse noise) and one picture taken at night (noise is not manually added).
  - Original pictures

- Gaussian noise

- Impulse noise

- Night photo

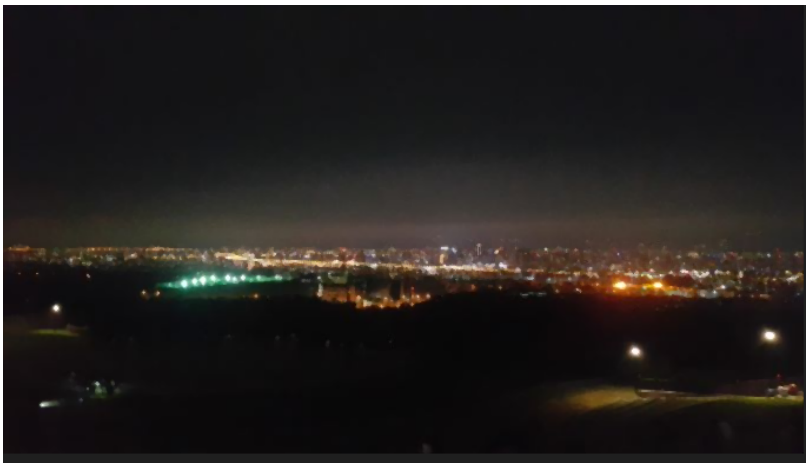


  - Median filter
    - Results

- Gaussian noise

- Impulse noise

- Night photo

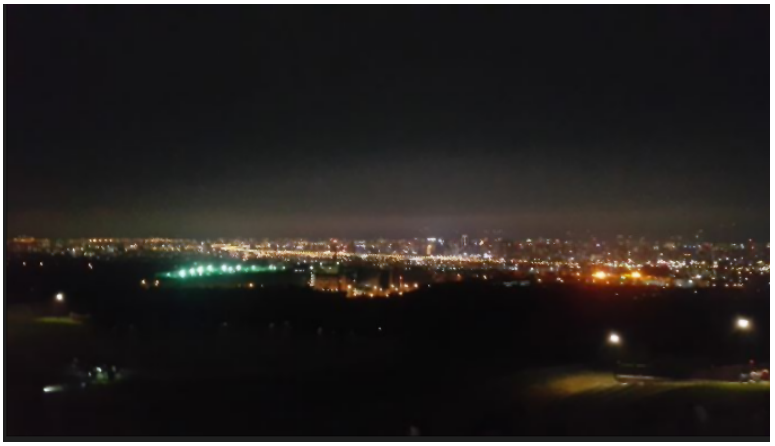


- Adaptive filter
  - Results

- Gaussian noise

- Impulse noise

- Night photo

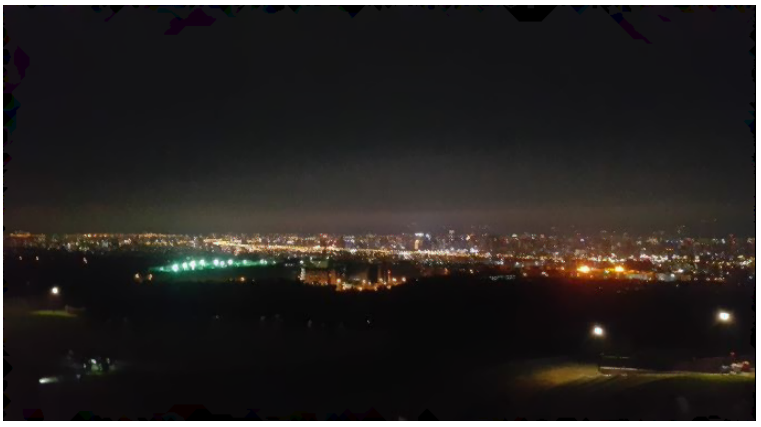


- Adaptive median filter
  - Results

- Gaussian noise

- Impulse noise

- Night photo



  - Discussion
    Three filters show different effects on the images. Especially, for the manually added impulse noise, they act very differently. Overall, the adaptive median filter seems to have the least blurring effect on each image. The results tell us that to reduce noise, we need to see the condition to choose the method we use.

- **Conclusion**
  In this assignment, we use the knowledge from the class to implement several image enhancement methods. Through the assignment, I think I learned more

about this part. Also, there are multiple methods to do the enhancement task, but it is very difficult to find an algorithm that works with all the conditions.

- **Code section**

```
[1] from PIL import Image
    import numpy as np
    import matplotlib.pyplot as plt
    import math
    import cv2
    from google.colab.patches import cv2_imshow
    ##lib
```

```
    ##image reading
    inputImg = []
    inputImg.append(cv2.imread('./img1.jpg'))
    inputImg.append(cv2.imread('./img2.jpg'))
    inputImg.append(cv2.imread('./img3.png'))
    inputImg.append(cv2.imread('./img4.png'))
```

```
[3] ##constract
    def powerLaw (img, gamma=1, c=1):
        newImg = np.zeros_like(img)
        newImg = np.array(c*255*(img/255)**gamma, dtype = 'uint8')
        cv2_imshow(newImg)
        print(newImg.shape)
        return newImg
```

```
[4] ##color adjust
    def colorCorrect (img, gamma=[1,1,1], c=[1,1,1]):
        newImg = np.zeros_like(img)
        for i in range(3):
            newImg[:,:,i] = powerLaw(img[:,:,i], gamma[i], c[i])
        cv2_imshow(newImg)
        return newImg
```

```
##denoising
class filter:
    def __init__(self, array=None, name='', size=None):
        self.array = array
        self.name = name
        self.shape = self.array.shape if size == None else size


def mediumFilter(size = (3,3)):
    return filter(name = 'Medium', size = size)

def adaptiveFilter(size = (3,3)):
    return filter(name = 'Adaptive', size = size)

def adaptiveMedianFilter(size = (3,3)):
    return filter(name = 'Adaptive Medium', size = size)
```

```
def convolution2D (img, kernel):
    ##zero padding
    h = (kernel.shape[0]-1)//2
    w = (kernel.shape[1]-1)//2
    img = np.pad(img, ((h,h),(w,w)), 'constant')

    imgHeight, imgWidth = img.shape
    kernelHeight, kernelWidth = kernel.shape
    outImg = np.zeros((imgHeight-kernelHeight+1, imgWidth-kernelWidth+1)).astype(int)
    noiseVar = np.var(img)
```

```
for x in range(imgHeight-kernelHeight+1):
    for y in range(imgWidth-kernelWidth+1):
        if kernel.name == 'Medium':
            outImg[x,y] = np.median(img[x:x+kernelHeight,y:y+kernelWidth])
        elif kernel.name == 'Adaptive':
            tmp = img[x+(kernelHeight-1)//2,y+(kernelHeight-1)//2]
            localVar = np.var(img[x:x+kernelHeight,y:y+kernelWidth])
            if (noiseVar >localVar):
                localVar = noiseVar
            outImg[x,y] = tmp-(noiseVar/localVar)*(tmp-np.mean(img[x:x+kernelHeight,y:y+kernelWidth]))
        elif kernel.name == 'Adaptive Medium':
            tmp = img[x+(kernelHeight-1)//2,y+(kernelWidth-1)//2]
            i = 0
            while(x-i>=0 and x+kernelHeight+i<imgHeight and y-i>=0 and y+kernelWidth+i<imgWidth):
                min = np.min(img[x-i:x+kernelHeight+i,y-i:y+kernelWidth+i])
                max = np.max(img[x-i:x+kernelHeight+i,y-i:y+kernelWidth+i])
                med = np.median(img[x-i:x+kernelHeight+i,y-i:y+kernelWidth+i])

                if(not(min<med and med<max)):
                    i+=1
                else:
                    if(min<tmp and tmp<max):
                        outImg[x,y] = tmp
                    else:
                        outImg[x,y] = med
                    break;
```

```
outImg = np.clip(outImg, 0, 255)
return outImg
```

```
def convolution (img, kernel):
    outImg = np.zeros_like(img)
    imgHeight, imgWidth, imgChannel = img.shape
    for i in range(imgChannel):
        outImg[:,:,i] = convolution2D(img[:,:,i],kernel)
    cv2_imshow(outImg)
    return outImg
```