Python Edge Enhancement

Liu, Zongsheng

Contents

Introduction

Edge enhancement was often referred to as edge sharpening. An image processing filter operated to enhance the contrast of the edges. The fundamental algorithm could be broken down into two parts; firstly, edge boundaries needed to be identified, and image contrast could be increased around the target boundaries. The report mainly focused on two mainstream methods Laplacian edge sharpening and unsharp mask filters. They had different algorithms for finding edges at the first stage. However, they had the same algorithm to enhance the edges. For Laplacian edge sharpening, it could be represented as $I_{enhanced}(x, y) = I_{input}(x, y) - \nabla^2 I_{input}(x, y)$ where ∇^2 was Laplacian edge detection filter. And, unsharp mask filters using a similar methodology as follow: $I_{enhanced}(x, y) = I_{input}(x, y) + kI_{edges}(x, y)$; since, k is a constant scaling factor to ensure that target edges were not $''oversharp''$.

Laplacian edge sharpening

The first method was named Laplacian edge sharpening since the first step of this process was using the Laplacian filter to identify the required edges. The human visual system measures the edges simply by knowing there was a great change in image details around the edges; it refers to a term$''discontinuities''$ in mathematical language. One of the most important techniques in edge detection is the derivative filter which follows the exact same algorithm. The assumption edges are the pixels with a fast rate of change in intensity level at some directions given by the gradient vectors. It implies that the derivative filter performed a gradient-based calculation as a fundamental algorithm. The Laplacian operator is a typical second derivative filter. Refer a simplified continuous model in $Figure1$ below, the assumption edges with a high rate of change in the intensity level would perform a peak which centred at the edge in the first derivative model; therefore, the required $''zero - crosing''$ could be found in the second derivative model respectively [3]. However, the basic algorithm of the $''zero - crosing''$ could also be applied to a discrete case. It has the following general form:

$$\nabla^2 f(x, y) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta x^2}.$$

Combining the discrete cases from $\frac{\delta^2 f}{\delta x^2}$ and $\frac{\delta^2 f}{\delta x^2}$, it would have following form when it applied to a discrete-space image:

$$\nabla^2 f(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y).$$

The corresponding 3×3 kernel be implemented by $M_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ or $M_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Consider two consecutive pixels q and p, and $\nabla^2(p)$ has different symbol with $\nabla^2(q)$, an edge lies between pixel p and q. p is classified as a zero-crossing if $|\nabla^2(p)| \leq |\nabla^2(q)|$. Since the second kernel M_2 is just a combination of the first kernel $M_1 =$ and its rotational version. Therefore, the second kernel $M_2 =$ would be used in the last code section since it included the diagonal direction and generally produced finer results. Once the required edges were identified, the next stage was the enhancement. It could be done artificially by subtracting the Laplacian from the original image. Therefore, it could be defined as follow:

$$I_{output}(x, y) = I_{input}(x, y) - \nabla^2 I_{input}(x, y).$$

In the last code section, a python edge enhancement example in Figure 3 would be presented based on the above algorithm.

Unsharp mask filter

An unsharp mask filter is an alternative approach rather than the Laplacian edge sharpening. Since it followed the same algorithm, the first stage detects the required edges. However, the edge detection used an alternative approach which represented as follow:

$$I_{edges}(x, y) = I_{original}(x, y) - I_{smoothed}(x, y).$$

As its name suggested, an unsharp or smoothed version of the original image was used in this process. There were several approaches to the smoothing process. However, the report was mainly focused on the Gaussian filter introduced in Workshop 4 [5]. The Gaussian filter is a typical weighted averaging filter where the kernel was derived from the Gaussian function:

$$f(x, y) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2 + y^2}{2\sigma^2}).$$

And, σ is a free parameter that represents the standard deviation of the Gaussian function. Since the Gaussian filter is a predefined Python function in the library [6].Thus, the Gaussian filter function was used directly from the library instead of building the own kernel in the relative codes in Figure 2. The smoothing level was only determined by the standard deviation σ , a relatively low value $\sigma = 2$ had be chosen in the code since the futures of sharp edges would be suppressed by a high value of σ [1]. Once the image had smoothed, the $I_{edges}(x, y)$ could be defined by subtracting $I_{smoothed}(x, y)$ from $I_{original}(x, y)$. The second stage was the enhancement which follows a similar methodology from Laplacian edge sharpening. Edge enhancement was reflected by combining the original image $I_{original}$ and I_{edges} together; it had the following form: $I_{enhanced}(x, y) = I_{input}(x, y) + kI_{edges}(x, y)$ where k is a parameter that generally in the range [0.2, 0.7] in order to ensure that the resulting image was not over sharped [1].

Comparison and Limitations

In Figure 4, there was a clear comparison between the two methods. The left column contains the results from the Laplacian edge sharpening, and the Unsharp mask filter results are in the right column. Since M2 kernel implied that every pixel needed to be compared with its eight immediate neighbours, which included the diagonal direction, the laplacian method on the left side may have a better performance on edge detection. Since the original image is a greyscale image with smooth boundaries, there was only a weak difference between the original image and the smoothed image. Thus, the edge-detection result from the Gaussian filter had a lack of performance in this case. Both methods had reflected some degrees of edge enhancement especially edges along the grid and the wood grain. However, the result from the Unsharp mask filter illustrated better performance with a more prominent contrast k=2 was set to be relatively more extensive than the normal range. However, if k was too large, it may cause an over-sharped image in Figure 5.

The edge enhancement algorithms from both methods were similar. So significant limitations on them were mainly caused by the edge detection process. For the unsharp mask method, when applying a Gaussian filter to smooth an image, the relatively unchanged smooth areas would not be impacted too much. It implies that $= I_{original}(x, y) - I_{smoothed}(x, y)$ produced some areas with high differences and also some areas with a low value. Therefore, when a scaling constant K is introduced in the enhancement process, some edges would be enhanced significantly, but some would not. The image used in Figure 3 was a smoothed image that may not illustrate noise's impacts. However, one of the most significant limitations of the Laplacian is the high sensitivity to noise. Since a noise point with a fast rate of change in intensity level would be miscalculated as a zero-crossing based on the algorithm and be misconducted as the fallacious edges [2]. In the practical application, a smoothing filter would be combined into the process in order to reduce the noise before the Laplacian edge detection process [4].

Conclusion

While there were many different approaches to the edge enhancement, two main edge-enhancement methods Laplacian edge sharpening and Unsharp mask filter had been considered in the report. Both of them had a similar approach to the enhancement process as follow:
Laplacian edge sharpening: $I_{enhanced}(x, y) = I_{input}(x, y) - \nabla^2 I_{input}(x, y)$
Unsharp mask filter: $I_{enhanced}(x, y) = I_{input}(x, y) + kI_{edges}(x, y)$.

However, they had different techniques to detect the edges in order to enhance them. Edges in the Unsharp mask filter method were found by computing the differences between the original image and its smoothed version produced by a Gaussian filter. Moreover, Laplacian methods involved a second derivative filter for finding zero-crossing as the assumption edges. Python returned a result that both methods had some level of effect on the edge enhancement. However, each method would have its suitable cases based on its limitations.

Reference

[1]Chris Solomon and Toby Breckon, Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab, John Wiley Sons, Ltd, 2011, sections 4.5–4.6

[2]Shrivakshan, G.T. Chandrasekar, Chandramouli. (2012). A Comparison of various Edge Detection Techniques used in Image Processing. International Journal of Computer Science Issues. 9. 269-276. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.1860rep= rep1type=pdf

[3]Phillip A. Mlsna, Jeffrey J. Rodríguez,Chapter 19 - Gradient and Laplacian Edge Detection,Editor(s): Al Bovik,The Essential Guide to Image Processing,Academic Press, 2009, Pages 495-524, ISBN 9780123744579, https://doi.org/10.1016/B978-0-12-374457-9.00019-6.

[4]Chandwadkar, R., Dhole, S., Gadewar, V., Raut, D. and Tiwaskar, S., 2013. Comparison of Edge Detection Techniques. Vishwakarma Institute of Information Technology, p.135

[5]James, Maddison, Week 4 Workshop Solutions, 2021, $https://www.learn.ed.ac.uk/webapps/blackboard/content/listContent.jsp?course_id =_8\ 7846_1 content_id =_6\ 629633_1$

[6]skimage.filters documentation, https://scikit-image.org/docs/stable/api/skimage.filters.html

Figure

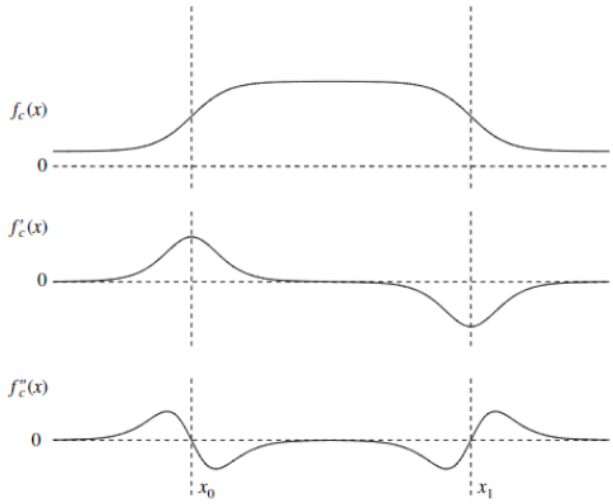

Figure 1: A simplified model in the 1D continuous case [3].
The assumption edges x_0 and x_1 with a fast rate of change in intensity level would perform local extremums in their first derivative model and also zero-crossings in the second derivative model.

```
def laplacian_edge_sharpening(img):
    """use laplacian operator to find and enhance the edges"""
    global img_enhanced1
    global img_edge1
    #set M2 as laplacian kernel using below
    M2=np.array([[1,1,1],[1,-8,1],[1,1,1]])
    height,width=img.shape
    ans=np.zeros((height-1,width-1))
    #apply the filter on every pixel
    for i in range(1,height-1):
        for j in range(1,width-1):
            ans[i,j]=np.sum(M2*img[i-1:i+2,j-1:j+2])
    #ans contains the required edges
    img_edge1=ans
    #compute the edge enhancement
    h,w=img.shape
    img_enhanced1=img[:h-1,:w-1]-img_edge1
laplacian_edge_sharpening(img)

def unsharp_mask_filter(img,k):
    global img_edge2
    global img_enhanced2
    #apply a guassion filter with stnadard div=2
    smooth_img= ip.gaussian_filter(img, sigma=2.0, truncate=4.0, mode="reflect")
    #edge detection process
    img_edge2=img-smooth_img
    #edge enhancement process
    img_enhanced2=img+img_edge2*k
unsharp_mask_filter(img,4)
```

Figure 2: Laplacian edge sharpening and Unsharp mask filter in Python
facets_improc, numpy and matplotlib.pyplot modules are imported as ip, np and plt respectively.

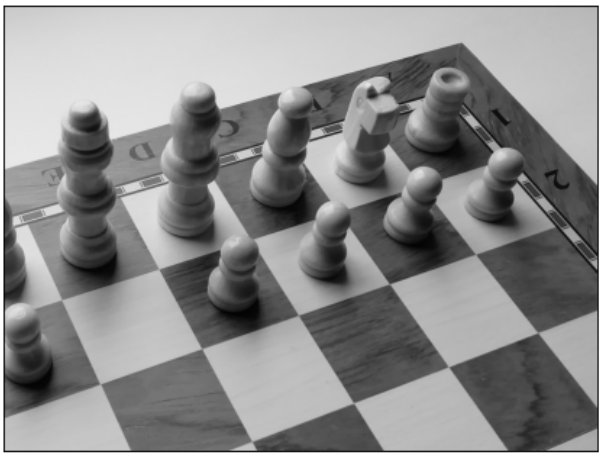

Figure 3: Original image

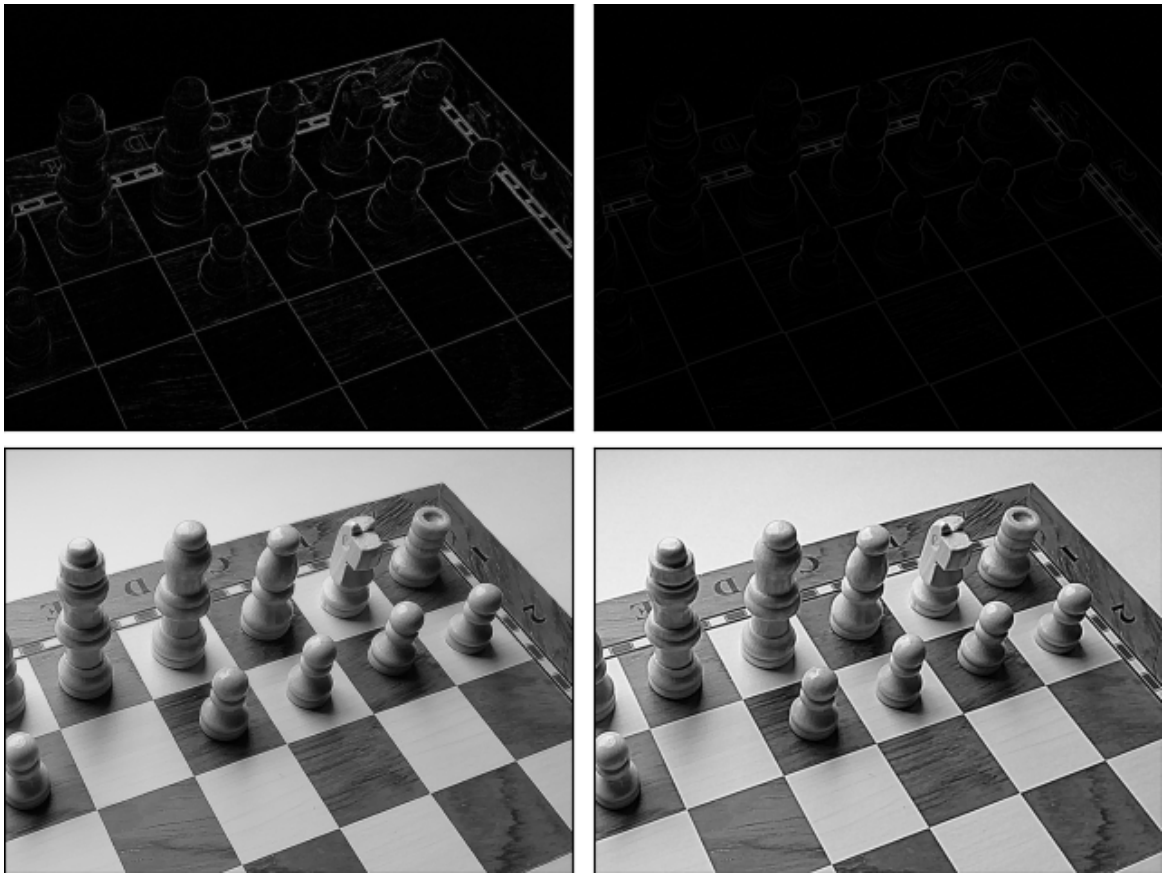

Figure 4: Laplacian edge sharpening VS Unsharp mask filter.
The left column contains the edge-detection result and the edge-enhancement result from the Laplacian edge sharpening; the right column compares the results from the unsharp mask filter with k=2 and $\sigma = 2$

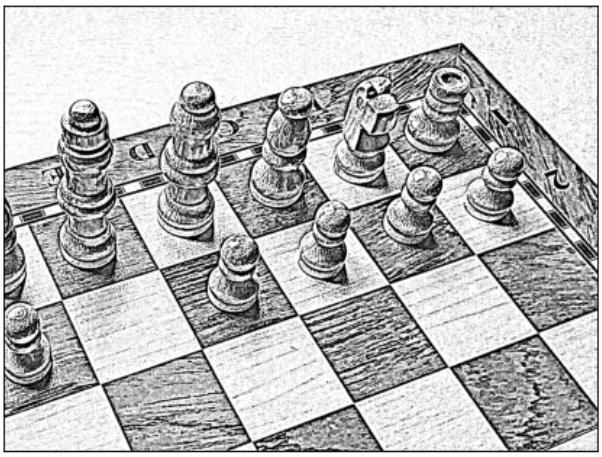

Figure 5: Example of an over-sharped image with k=80

```
%matplotlib notebook
import facets_improc as ip
import numpy as np
import matplotlib.pyplot as plt
img=ip.imread("chess_1024x768_gs.png")

def laplacian_edge_sharpening(img):
    """use laplacian operator to find and enhance the edges"""
    global img_enhanced1
    global img_edge1
    #set M2 as laplacian kernel using below
    M2=np.array([[1,1,1],[1,-8,1],[1,1,1]])
    height,width=img.shape
    ans=np.zeros((height-1,width-1))
    #apply the filter on every pixel
    for i in range(1,height-1):
        for j in range(1,width-1):
            ans[i,j]=np.sum(M2*img[i-1:i+2,j-1:j+2])
    #ans contains the required edges
    img_edge1=ans
    #compute the edge enhancement
    h,w=img.shape
    img_enhanced1=img[:h-1,:w-1]-img_edge1
laplacian_edge_sharpening(img)

def unsharp_mask_filter(img,k):
    global img_edge2
    global img_enhanced2
    #apply a guassion filter with stnadard div=2
    smooth_img= ip.gaussian_filter(img, sigma=2.0, truncate=4.0, mode="reflect")
    #edge detection process
    img_edge2=img-smooth_img
    #edge enhancement process
    img_enhanced2=img+img_edge2*k
unsharp_mask_filter(img,4)

#comparing results in a 2*2 block
fig, ax = plt.subplots(2, 2)
plt.subplots_adjust(hspace=0.04, wspace=0.04, top=1, bottom=0, left=0, right=1)
plt.sca(ax[0, 0])
ip.imshow(img_edge1, new_figure=False)
plt.sca(ax[0, 1])
ip.imshow(img_edge2, new_figure=False)
plt.sca(ax[1, 0])
ip.imshow(img_enhanced1, new_figure=False)
plt.sca(ax[1, 1])
ip.imshow(img_enhanced2, new_figure=False)

ip.imshow(img)
unsharp_mask_filter(img,80)
ip.imshow(img_enhanced2)
```

Figure 6: Full python code