RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences
Chair of Computer Science 13 (Computer Vision)
Prof. Dr. Bastian Leibe

Seminar Report

Linear and Nonlinear Filters

Alexander Skretting
Matriculation Number: 445457

Jose Rigel Soeryo Soebandoro
Matriculation Number: 444345

June 2023

Advisor: George Lydakis

Abstract

Contents

1 Introduction **3**

2 Linear Filters **3**

2.1 Correlation and Convolution Filtering 3
2.2 Padding (Border Effects) 4
2.3 Separable Filtering 5
2.4 Examples of Linear Filters 6
2.5 Sobel Filter 8
2.6 Band-pass and steerable Filter 8
2.7 Summed area table (integral image) 10

1 Introduction

Every picture created is different, from the camera they are taken with, the color space, to the subjects on the picture itself. As humans, we have trained our eyes to adjust our perceptions to understand the pictures better and take information as we need through the context gathered from day to day life. This is unfortunately not the case for computers. Varying qualites and scenarios makes it hard for computers to handle images and extract necessary informations.
This is where filtering is utilized to pass through and create a desired output image, catering to one' individual goals. The two major categories of filtering in Image processing are *Linear* and *Nonlinear filters*, with each having variations of their own, tailored to their specific use cases, whether for noise suppresion, edge detection or image enhancement[MW20]. The comparison between each filter is not only determining which filters produces the highest quality output, yet also defined by its computing complexity, both in space and time complexity. Optimizing these filters by means of different algorithms and strategies while maintaining its quality is highly preferable.
In this article, we will discuss the differences between different types of linear and non linear filters.[1]

2 Linear Filters

With **Linear Filtering**, or specifically linear spatial filtering, the function which is used to pass the image through must be linear and shift invariant. The function $\boldsymbol{L}$ is considered linear if there exists two constants a and b for any two input pixels $f_1(m,n)$ and $f_2(m,n)$ such that

$$\begin{aligned}
&\implies g_1(m,n) = \boldsymbol{L}[f_1(m,n)] \land g_2(m,n) = \boldsymbol{L}[f_2(m,n)] \\
&\implies a \cdot g_1(m,n) + b \cdot g_2(m,n) = \boldsymbol{L}[a \cdot f_1(m,n) + b \cdot f_2(m,n)]
\end{aligned}$$

also called the *superposition property of linear systems*[BA09]. Whereas a function is considered *shift invariant* when

$$\begin{aligned}
&\implies g(m,n) = \boldsymbol{L}[f(m,n)] \\
&\implies g(m-p,n-q) = \boldsymbol{L}[f(m-p,n-q)]
\end{aligned}$$

p and q implies a spatial shift to both the input and output pixels[BA09]. An intuitive way to think of these two properties is that the function has to behave the same throughout the entire picture.

2.1 Correlation and Convolution Filtering

A common formula for linear filtering is the *Correlation Filtering*[BA09].

$$g(i,j) = \sum_{l \in \mathcal{M}} \sum_{k \in \mathcal{N}} f(i+k, j+l) \cdot h(k,l)$$

or commonly notated as $g = f \otimes h$.
The desired output pixel $g(i,j)$, where i and j specify the coordinates of it, is based on a $M \times N$ sized neighborhood, meaning not only does one pixel define an output pixel, but also a specified number of its neighbors. The influence of each pixel in the neighborhood is defined by the filter coefficient $h(k,l)$, also called its *kernel* or *mask*[Sze22].

$$\underbrace{\begin{bmatrix} 128 & 34 & 123 \\ 68 & 54 & 73 \\ 100 & 95 & 17 \end{bmatrix}}_{\text{input neighborhood}} \otimes \underbrace{\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}}_{\text{kernel}} = 75$$

As the above example with a 3×3 kernel, a total of 9 pixels is needed to calculate a single output pixel.

[1] Written by Jose

Another common variant on the formula is having the signs of the offsets reversed.

$$g = f * h$$

$$g(i,j) = \sum_{l \in \mathcal{M}} \sum_{k \in \mathcal{N}} f(i-k, j-l) \cdot h(k,l)$$

With this formula, $*$ is called the *convolutional* operator, and the kernel h is called the *impulse response function*. An interesting note is that, when the kernel h is convolved with an impulse signal δ (an image with 0 everywhere except the origin), it reproduces the kernel itself $h * \delta = h$, whereas with correlational filtering, it produces the reflected signal (inverted signal in both dimensions).[Sze22] In cases where the kernel is symmetrical on both axis (e.g. box blur), the result of convolutional and correlational is the same.

(a) correlation (b) convolution

Figure 1: Difference between Correlation and Convolution Filtering by translating 32 pixels

By rotating the kernel 180°, it is possible to get the same output from convolution filtering with correlational filtering and vice versa[GW18].

An apparent problem from neighborhood filtering is that on the edges, the neighbors simply does not exist in one or two directions (e.g. a 1000×1000 image passed through a 3×3 kernel would produce a 998×998 image). There are a couple method to alleviate the calculation of the nonexistent neighbors.

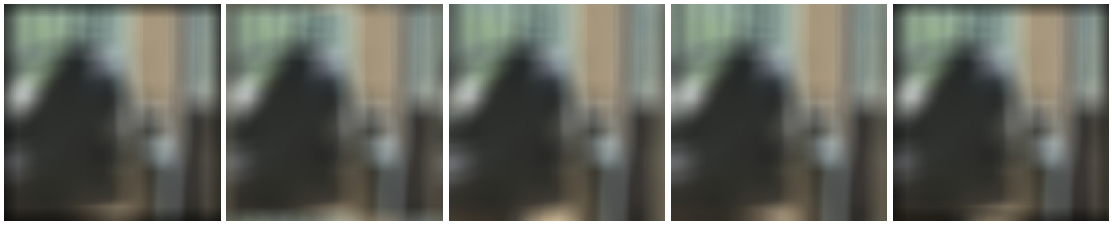
2.2 Padding (Border Effects)

(a) zero (b) wrap (c) clamp (d) mirror (e) extend

Figure 2: Different types of border padding

- **Zero:** Set all pixels outside the region to zero.
- **Constant:** This is similar to Zero, but instead a specific color is chosen to replace the out of bound pixels.
- **Clamp:** The edge pixels are repeated indefinitely.
- **(Cyclic) Wrap (Repeat or Tile):** The entire image is repeated similar to tiling it infinitely.
- **Mirror:** The image is mirrored accordingly on each of the edges.
- **Extend:** This is a combination of clamping and mirroring, where the clamped image is subtracted with the mirrored image.

This of course not only affects the unseen pixels, but the effects can bleed into the pixels inside the frame.



(a) zero (b) wrap (c) clamp (d) mirror (e) extend

Figure 3: Effects of different paddings with box blur

As seen from Figure 3, with a constant padding, the color chosen for the constant value bleeds uniformly in the edges. The wrapped image can be convincing at times, but drastic changes from one border to another may lead to undesirable bleeding edges. If we instead clamp the image, the repeated pixels are noticeably present, depending on the kernel size, this may still be desireable. In this use case, mirror seems to be the most convincing. It cannot however be said, that mirror is objectively the better method, as it depends on the scenarios and use cases (e.g. if a subject is placed in a white background, then the constant padding may be desirable).

There are other ways to handle the borders such as two-phase duplication, which is similar to clamp but alternates between the last 2 pixels. A change in the filter function per individual basis can also be implemented, but it might not be practical, as large amounts of additional logic may be needed[BA18].

2.3 Separable Filtering

The time complexity to calculate a single pixel can be said to be $O(M \cdot N)$ as it is dependent on the size of the kernel. For each of the pixel in the kernel, it must be both multiplied and summed together with its neighbors. This can be in practice sped up by separating the horizontal and vertical operations, hence performing a one dimensional convolution horizontally, then one additional operation for the vertical convolution. This kernel in this case is the outer product of two one dimensional kernels[Sze22].

$$\boldsymbol{K} = v \otimes h = \begin{bmatrix} v_1h_1 & v_1h_2 & v_1h_3 & \dots & v_1h_n \\ v_2h_1 & v_2h_2 & v_2h_3 & \dots & v_2h_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_mh_1 & v_mh_2 & v_mh_3 & \dots & v_mh_n \end{bmatrix}$$

Also notated as $K = vh^{\boldsymbol{T}}$, where $\boldsymbol{T}$ denotes transposing h, since both are supposedly vertical tuples.

A method to check whether a given Kernel is separable is to use *Singular Value Decomposition (SVD)* [Sze22]

$$\boldsymbol{K} = \sum_i \sigma_i \cdot u_i \cdot v_i^{\boldsymbol{T}}$$

or more generally

$$\begin{aligned} \boldsymbol{K} &= \mathbf{U}_{m\times p}\boldsymbol{\Sigma}_{p\times p}\mathbf{V}^{\boldsymbol{T}}_{p\times n} \\ \boldsymbol{K} &= \begin{bmatrix} u_0 & \dots & u_{p-1} \end{bmatrix} \begin{bmatrix} \sigma_0 & & \\ & \ddots & \\ & & \sigma_{p-1} \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{p-1} \end{bmatrix} \end{aligned}$$

Note that it requires the number of rows and columns must be the same, it can however be solved by putting 0 as the missing rows/columns, to allow the kernel to be a square matrix.Then the first singular value σ_0 is checked. If it is the only one that is non zero, and the rest is zero, then the kernel is separable. As an

example, the sobel filter, which is used for edge detection, can be separated into two separate operations. In this case, this is for the horizontal derivative approximation. [CLLM23]

$$\frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\2\\1\end{bmatrix}$$

The normalization of the operators are easily calculated by dividing each operator with the absolute sum of the entries.

$$\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \Rightarrow |-1|+|0|+|1| = 2 \Rightarrow \frac{1}{2}$$

Then the normalized operator is

$$\frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$$

it applies to the vertical operator as well

$$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}$$

The following are more examples of common separable filtering.

$$\frac{1}{K^2}\begin{bmatrix}1 & 1 & \dots & 1\\ 1 & 1 & \dots & 1\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & \dots & 1\end{bmatrix} \quad \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \quad \frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} \quad \frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \quad \frac{1}{4}\begin{bmatrix}1 & -2 & 1\\ -2 & 4 & -2\\ 1 & -2 & 1\end{bmatrix}$$

$$\frac{1}{K}\begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix} \quad \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \quad \frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix} \quad \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \quad \frac{1}{2}\begin{bmatrix}-1 & 2 & 1\end{bmatrix}$$

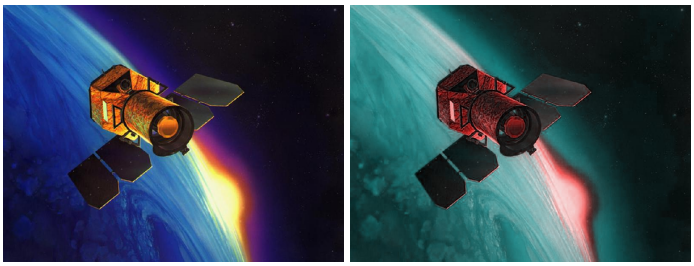
Box Bilinear Gaussian Sobel Corner

Note that for Corner filtering, the kernel is not normalized. Normalization is important as otherwise, a pixel may go beyond the allowed value range. In cases where the kernel has a negative value, it means that the pixel value is inverted, since an integer overflow occurs (e.g. $64(-1) = -64 \mod 256 = 192$ for one channel in an 8 bit pixel).'' With some filters, it is not always possible to separate it into just two operations, therefore it can be transformed into a series of 1 dimensional operations. However, this may be impractical as it increases the time and space complexity.

2.4 Examples of Linear Filters

Due to the fact that linear filter in essence is the sum of pixels in a neighborhood with varying weights and sizes, there are infinitely many different kernels that can define operators which classified as linear and invariant. It is also even possible to factor in different color channels. As an example, it is possible to shift the hue of an image. In this example, using a kernel of $\begin{bmatrix}0 & 0 & 1\end{bmatrix}$, the hue is shifted to the left (negative).

```
img = cv.imread("color.jpg")
kernel = np.array([[[0, 0, 1]]])
convolved = ndimage.convolve(img, kernel, mode='reflect')
cv.imwrite('../img/colorShift.jpg', convolved)
```

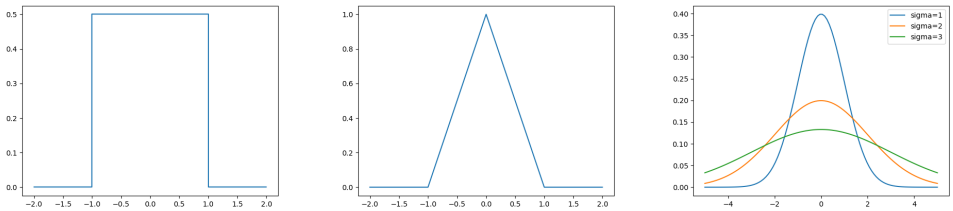


(a) original (b) shifted

Figure 4: Hue shifting using linear filters of the GALEX telescope image[NAS23]

- **Box Filter:** each pixel is determined by the average of a $K \times K$ neighborhood of pixels. It is also often called the *moving average*, as the calculation can be done with scanlines, subtracting the old pixel and adding the newest pixels to the running sum. As the pixel is influenced equally by a square neighborhood, this may not be ideal as some 'square artifacts' can be seen.
- **Bartlett Filter:** one solution to this is by separably convolving the image with a linear 'tent' function (having its peak in the middle). A 3×3 version of this is called the bilinear kernel.
- **Approximate Gaussian Filter:** if the Bartlett kernel is convolved with itself, this will produce an approximate version of the gaussian kernel.

Essentially, the three above filters are different mathematical functions applied both horizontally and vertically.



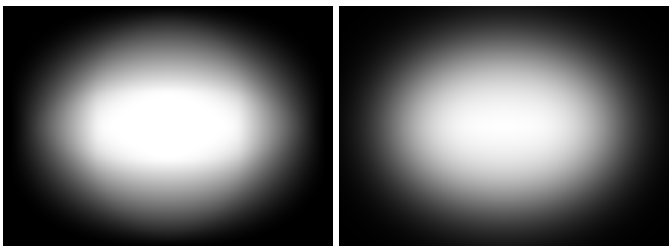
(a) Average (b) Bartlett (Tent) (c) Gaussian

Figure 5: One dimensional graphs of the filters

As its name suggests, the Gaussian Filter is based on the Gaussian function (also commonly known as the normal function). Two dimensionally, it can be expressed as [MW20]

$$G_{2D}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

It can be seen from the graph that the lower the sigma value, the narrower the neighborhood becomes.



(a) Box ($k = 133$) (b) Gaussian ($k = 233$)

Figure 6: Comparison between gaussian and box blur

From Figure 6, using the gaussian blur, as opposed to the box filter produces a more appealing image, as it

preserves the shape of the image better as opposed to creating this 'halo' effect and producing a squashed rectangular shape in the center. One disadvantage to Gaussian however is, to produce an equally 'blurred' image, a larger kernel is required for Gaussian, which can be argued to then be slower than using the box filter[GW18].
These filters are mainly used to smoothen(blur) the image, where the low frequencies are let through as well as reducing the effect of the high frequencies (high frequencies and low frequencies both referring to the brightness/darkness of each pixel). In practical applications, this is done to reduce high frequency noises. A creative application to smoothing the images, is that the smoothen images can be used as a mask to sharpen the image by adding the difference between the original and blurred image. This method is called *unsharp masking*[Sze22].

$$g_{sharp} = f + \gamma(f - h_{blur} * f)$$

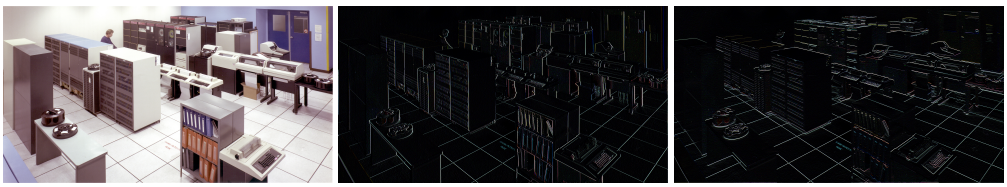
(a) original (b) unsharp mask (c) difference (d) sharpened

Figure 7: Unsharp masking with Gaussian blur

The result of the image traces a bright outline througout the image between the darker(shadow) and lighter part. It can also be seen, that the difference between the blurred image and the original results in an approximate edge detection. A better way to extract edges by means of linear filtering is by the use of the *Sobel filter*. [Sze22]

2.5 Sobel Filter

As per the example in the previous section, the horizontal Sobel operator is calculated by combining the *horizontal central difference* (which calculates the second horizontal derivative of the pixel and places it in the center itself) and a vertical linear tent filter to smooth the result. Respectively, it can also be done for the vertical derivative.
The two sobel operator emphasizes the horizontal or vertical edges respectively, although as seen in the images, it can also highlight diagonal edges.



(a) original (b) horizontal (c) vertical

Figure 8: Vertical and Horizontal Sobel filtering[NAS90]

2.6 Band-pass and steerable Filter

In the above example, the sobel filter is calculated either for the horizontal or vertical axis. Nevertheless, it is possible to calculate filters (including for sobel) for any angles in between using a specified *directional derivative* $\nabla_{\hat{\boldsymbol{u}}} = \frac{\partial}{\partial \hat{\boldsymbol{u}}}$ and a *unit direction* $\hat{\boldsymbol{u}} = (\cos\theta, \sin\theta)$[Sze22].

$$\hat{\boldsymbol{u}} \cdot \nabla(G * f) = \nabla_{\hat{\boldsymbol{u}}}(G * f) = (\nabla_{\hat{\boldsymbol{u}}} G) * f$$

Then it is also preferable to smooth it with the Gaussian filter.

$$G_{\hat{\boldsymbol{u}}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}$$

This is called the *directional* or *oriented* filter.
A similar to operator to the sobel filter is called the *Prewitt operator*, which instead of convolving it with a perpendicular tent filter, it instead uses an average operator.[NA20]

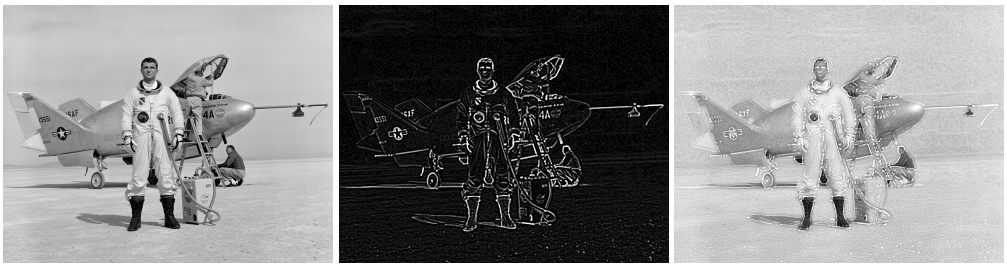
$$\underset{\text{(a)}}{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{\text{(b)}}{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}} \quad \underset{\text{(c)}}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}}$$

Figure 9: Sobel and Prewitt operator comparison (a) Horizontal Sobel operator; (b) Horizontal Prewitt operator; (c) Vertical Prewitt operator
Another filter is called the *Laplacian operator*, which is the second (undirected)derivative.[Sze22]

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

and again, it can be smoothed with the Gaussian filter. It is equivalent to the *Laplacian of Gaussian*(LoG) filter.

$$\nabla^2 G(x, y, \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x, y, \sigma)$$



(a) (b) (c)

Figure 10: Laplacian Filtering (a) Original image of NASA pilot[NAS70]; (b) Output of the laplacian filter with Gaussian blur (c) Combined image of the orientation map and original
Since the laplacian filter does not rely on a specific direction, it can be seen at the above picture that the edges are agnostic to the direction and not biased to a certain angle. Bear in mind that smoothing the picture (in this case by means of the Gaussian filter) is necessary as it subdues the noises from being detected as edges.
The aforementioned filters are more sophisticated as they require the second or first derivatives and they can filter both the low and high frequencies. These filters are called *band-pass filters*. Since the filters are based on the gradient fields, which are undirected, it can be 'steered' by freely choosing a unit direction, hence the name *steerable* filter. The advantage of this is that multiple variations of directions can easily be calculated with minimal cost as the derivative must only be calculated once. In case of higher order derivatives, the calculation must be done repeatedly with the same unit vector. This can although be calculated with a relatively small number of functions[Sze22]. Consider the example below for the second-order derivatives, which requires 3 basis functions. Although the basis filters are not in itself separable, it can be expressed as a linear combination of multiple separable filters.

$$G_{\hat{\boldsymbol{u}}\hat{\boldsymbol{u}}} = u^2 G_{xx} + 2uvG_{xy} + v^2 G_{yy} \text{with } \hat{\boldsymbol{u}} = (u, v)$$

With higher order steerable filters, it is possible to create filters that is more selective with the orientation (unlike in Figure 8, where the lines placed at an angle of 45° is highlighted). Alternatively, it can also be

possible to create filters that responds to more than one type of edge, whether bar edges (thin lines) or step edges (stark transitions between dark and light pixels), this however requires the *Hilbert transform pairs* [Sze22].

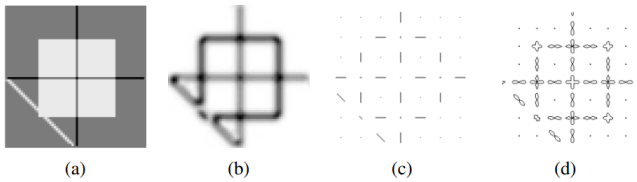


Figure 11: Fourth-order steerable filter (a) original test pattern (b) average oriented output (c) dominant orientation (d) oriented energy as a function of angle (polar plot)
[Sze22]

Steerable filtering are most commonly used for edge detection, but another common use case is *corner detection*. This requires multiple steerable operators in different directions as corners are defined as points with a sharp change in direction (high curvature)[NA20]. Feature description is also one application of steerable filter[Sze22].

2.7 Summed area table (integral image)

To calculate pixels in a neighborhood, some pixels needs to be accessed multiple times and as the kernel size increases, this problem also magnifies as well. To better process the convolution of multiple box filters of different sizes and locations, preprocessing it can practically reduce the time complexity. This can be done with the *summed area table*, or the running sum of all the pixel values from the origin. It can be computed recursively through the raster-scan algorithm.

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

This is also an example of recursive filtering, which takes a previous calculated value of a pixel and calculate it further for the next pixel. In this case, this can theoretically go on forever, but since no picture is infinite, the edges can be calculated using the padding strategies in Section2.2.
Since the calculations are based on previously calculated pixels, this attempts to reduce the amount of times a pixel is used for calculations, as it immediately takes the already calculated value instead. The image $s(i,j)$, often called *Integral image* can be computed using two additions per pixel. The summed area inside a chosen rectangl can be calculated by sampling 4 pixels from the summed area table.

$$S(i_0 \dots i_1, j_0 \dots j_1) = s(i_1,j_1) - s(i_1,j_0-1) - s(i_0-1,j_1) + s(i_0-1,j_0-1)$$

3	2	7	2	3
1	5	1	3	4
5	1	**3**	5	1
4	3	2	1	6
2	4	1	4	8

(a) Original image(S = 24)

3	5	12	14	17
4	*11*	**19**	24	31
9	**17**	28	38	46
13	24	37	48	62
15	30	44	59	81

(b) Summed area table(S = 28)

3	5	12	*14*	17
4	11	19	24	31
9	17	28	38	46
13	24	37	**48**	62
15	30	44	59	81

(c) Computation of area sum (S = 24)

Figure 12: Summed area tables[Sze22]

The above table shows the relation of how the area is calculated. For summed area table, the red pixel is calculated by adding itself to the two highlighted pixels in bold while subtracting the italicised symbol After the entire summed area table is calculated, it is now cost effective to calculate the area sum as the sum of the top left and bottom right numbers only needs to be subtracted with the other diagonal (remember that the starting indices requires it to go one pixel beyond the corresponding rectangle).
Although with this preprocessing it reduces the time complexity, the space complexity increases, since each pixel's range has now increased due to summing multiple pixel into a single pixel, hence each of them now requires $\log M + \log N$ additional bits (M and N being the width and height)[Sze22].

A common use case for summed area table is for feature detections, computing simple multi-scale low-level features. In practice, it may be preferrable to instead use moving average filters, for example in computing the sum in the sum of squared differences stereo (SSD) and motion algorithms[Sze22].[2]

References

[BA09] Alan C. Bovik and Scott T. Acton. Chapter 10 - basic linear filtering with application to image enhancement. In Al Bovik, editor, *The Essential Guide to Image Processing*, pages 225–239. Academic Press, Boston, 2009.

[BA18] Donald G. Bailey and Anoop S. Ambikumar. Border handling for 2d transpose filter structures on an fpga, Nov 2018.

[CLLM23] Qiong Chang, Xiang Li, Yun Li, and Jun Miyazaki. Multi-directional sobel operator kernel on gpus. *Journal of Parallel and Distributed Computing*, 177:160–170, 2023.

[GW18] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, 4th edition, 2018.

[MW20] Siddharth Misra and Yaokun Wu. Chapter 10 - machine learning assisted segmentation of scanning electron microscopy images of organic-rich shales with feature extraction and feature ranking. In Siddharth Misra, Hao Li, and Jiabo He, editors, *Machine Learning for Subsurface Characterization*, pages 289–314. Gulf Professional Publishing, 2020.

[NA20] Mark S. Nixon and Alberto S. Aguado. 4 - low-level feature extraction (including edge detection). In Mark S. Nixon and Alberto S. Aguado, editors, *Feature Extraction and Image Processing for Computer Vision (Fourth Edition)*, pages 141–222. Academic Press, fourth edition edition, 2020.

[NAS70] NASA. Pilot major cecil powell and the x-24a on lakebed, Nov 1970. [Online; accessed June 24, 2023].

[NAS90] NASA. 40 x 80 foot data acquisition system, Feb 1990. [Online; accessed June 24, 2023].

[NAS23] NASA. Galex (galaxy evolution explorer): launched april 28, 2003, Apr 2023. [Online; accessed June 24, 2023].

[Sze22] Richard Szeliski. *3.2 Linear filtering*, page 119–131. Springer, 2nd edition, 2022.

[2]Entire section of linear filtering is written by Jose