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

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*[Sze22a].

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

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).[Sze22a] 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.

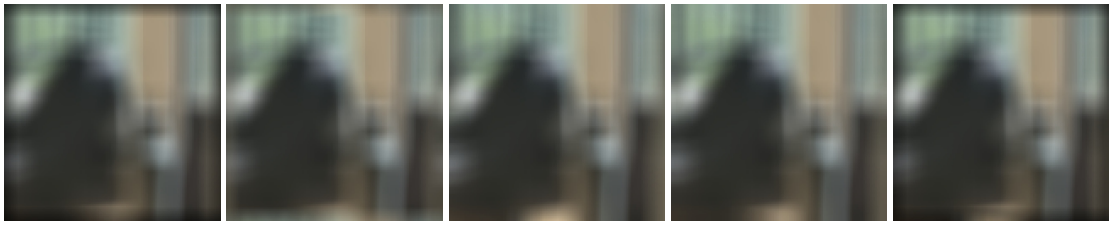
2.2 Padding (Border Effects)

(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[Sze22a].

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

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)* [Sze22a]

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

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.

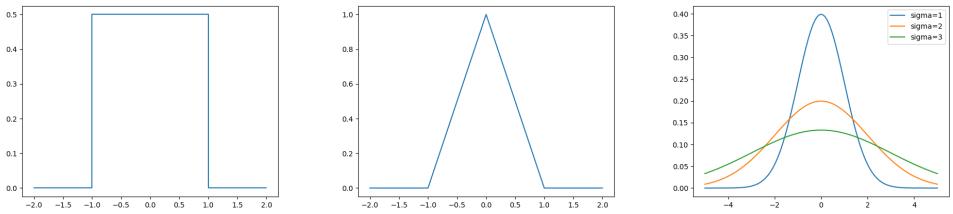
2.4 Examples of Linear Filters

- **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.



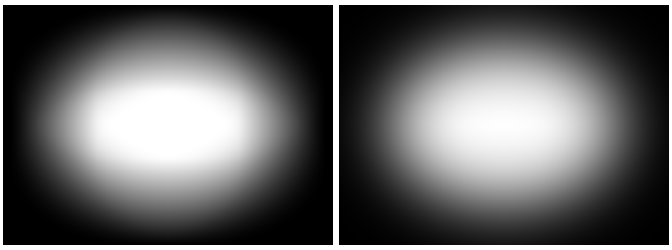
(a) Average (b) Bartlett (Tent) (c) Gaussian

Figure 4: 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 5: Comparison between gaussian and box blur

From Figure 5, 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*[Sze22a].

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

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

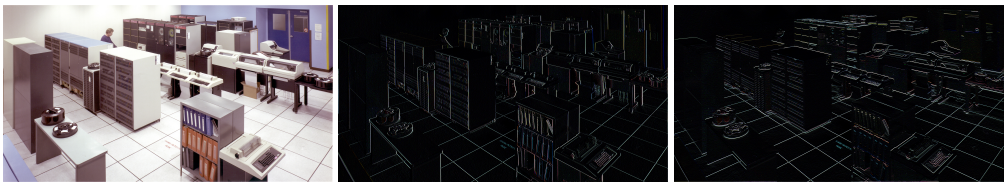
Figure 6: Unsharp masking with Gaussian blur

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*. [Sze22a]

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 7: 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* $\hat{\boldsymbol{u}} = (\cos\theta, \sin\theta)$[Sze22a].

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

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

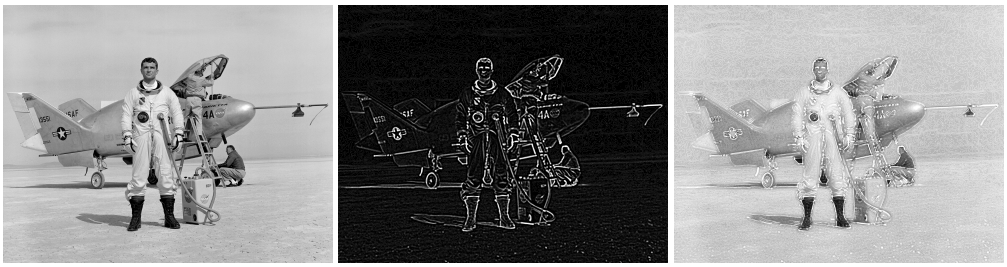
$$G_{\hat{\boldsymbol{u}}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}$$

This is called the *directional* or *oriented* filter. Another filter is called the *Laplacian operator*, which is the second (undirected)derivative.[Sze22a]

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

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

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



(a) (b) (c)

Figure 8: Laplacian Filtering[NAS70] (a) Orinal image of NASA pilot; (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[Sze22a]. 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 7, 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* [Sze22a].
The common use case of steerable filtering is either for feature description (extracting interest points in images) or edge detection[Sze22a].

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 Section 2.2.
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 9: Summed area tables[Sze22a]

The above picture 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)[Sze22a].

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[Sze22a].[2]

3 Non-Linear Filters

Non-linear filters serve as additional ways of filtering images. Linear filters can work exceptionally well for certain cases, however one will encounter scenarios where using Non-Linear filters will lead to a better result both visually (Image Processing) and performance-wise. One example noise removal, but more on that later. First of all it is important to look at what non-linear filters actually are.

3.1 What are Non-Linear Filters?

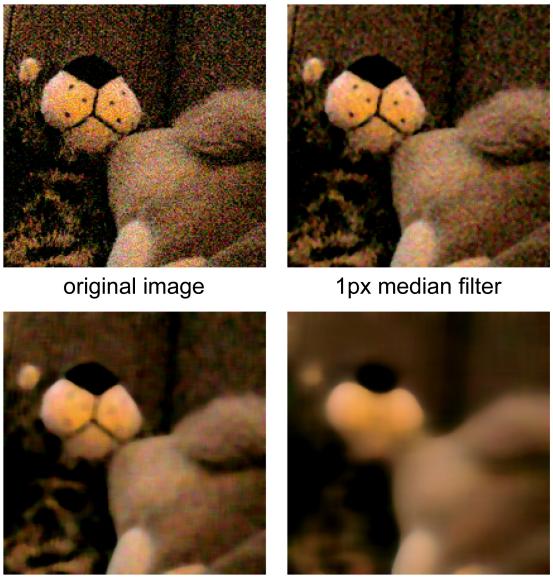
First and foremost, what are Non-Linear filters? As mentioned previously, linear filters are filters where the signal-response of two signals and the sum of the responses to the two signals individually are the same. Based on that one may probably guess that non-linear filters are filters where this criteria is not satisfied, and this is the case. This could mean that, in relation to the input data, the transformation could, for instance, be exponential as opposed to linear. This can lead to bigger difficulties concerning frequency response analysis,[Sze22b, p. 132] shift-invariance, etc.

3.2 Some different types of Non-Linear filters

3.2.1 Median Filtering

Consider a scenario with so-called salt and pepper noise, i.e. an image with noise consisting of sporadic black and white pixels. Were one to use for instance a Gaussian filter, one would quickly end up with a slightly blurred version of the original image. The noise would most likely still be there, simply in a slightly blurred format. Somewhat of a disappointment. There are however other methods of filtering, and for this scenario, a median filter would most likely do the trick.

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



h

Figure 11: Processing the image with a bigger kernel radius leads to less "crispness"

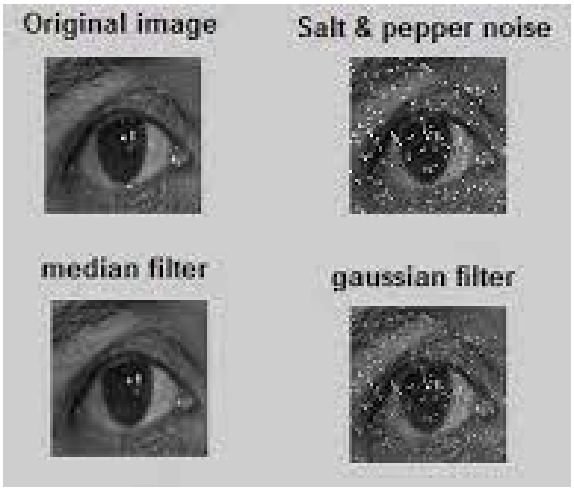


Figure 10: The same image processed with a median filter and a Gaussian filter. Both filters remove salt and pepper noise (in varying degrees) and produce a result that more accurately corresponds with how the image should look without noise

Now we have seen an example median filtering, compared to Gaussian filtering, for a specific scenario, and what it does, it is time to dive into what a *median filter actually is*. The algorithm works in this way: Given an input signal, in this case an image, it goes through every single pixel of the image, and replaces the value with the median value of the adjacent pixels. One can also let it go through the image whilst checking adjacent pixels in a 3px, 10px, or generally X-pixel radius around the current one, and replace it with the median of the pixels inside this X-pixel radius. The "X-pixel radius" is called a kernel. Increasing the kernel-size of the filter might lead to a "smeared" and somewhat blurry look in many scenarios. However, even though median filters work relatively well for this specific scenario, the cure for cancer is yet to be found. There are (several) scenarios in which median filters might produce sub-optimal results.

Figure 12: lllustration of how the mean filter algorithm works. The central pixel's value is changed from 80 to 40. 40 is the median of all the values within a 1px radius of the central pixel.

3.2.2 Bilateral Filtering

Another type of Non-Linear filter is the bilateral filter. In order to properly understand the bilateral filter, it is first important to understand a few other principles.
It would, at this point, be prudent to mention a certain property of filters that can be very much sought after, namely, *edge-preservation*. When filtering away noise one often runs the risk of losing the clarity of the edges in the image. Given an image with a coin, and some noise, one might filter away the noise with a filter, perhaps a Gaussian would work here. After applying the filter, the noise might have been cleared away, but one will often recognize a certain "smudgyness" by the edges. The, perhaps, once sharp clear edges where the coins meet the background have seemingly melted more into their surroundings. Somewhat problematic. Applying an even stronger filter might remove the clear edge of the coins completely, making the coins look more like the sun in the middle of day than the coins they supposedly are.

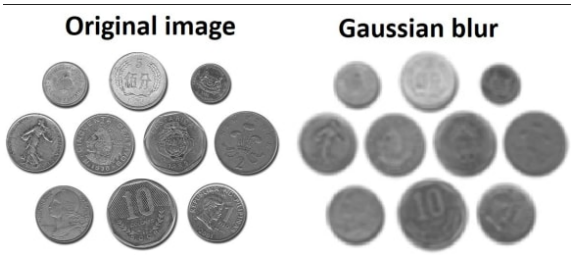


Figure 13: Applying a Gaussian-filter has made the edges of the coins less pronounced. This goes for the background-coin-edge, and also the edges of the coin details such as numbers and figures

Another important filtering method to recognize, at this point, is weighted kernel filtering. Weighted kernel filtering is an extension of the median filter. It works by *weighting* pixels based on their distance from the middle. I.e. the further the pixel is from the center, the smaller the effect of it, but importantly, it still has an influence!
Now to bilateral filters themselves. The base principle of bilateral filtering is applying the concept of weighted median filtering, yet with a tweak. The bilateral filter rejects values that are too much of an outlier for the current kernel, hence, often preserving edges better than the previously mentioned filters. Since a median filter uses all pixel-values in the kernel when processing an image, it may very well lead to blurred edges. Imagine, for instance, the aforementioned example with the coins (Figure 3.4). When the median filter does it's magic on the edges, for instance where the coin meets the background, it will use the values of the white background, and mix it into the values of the darker coins edges. Ergo, the edge of the darker coin will turn into a lighter colour because of the influence of the background. Apply the median filter several more times, and this will happen again and again and again, quickly leading to an unrecognizable blur, rather than a crisp clear image of coins with the noise removed (Note: This will most likely happen regardless of filter if one applies it an excessive amount of times, but the amount of times one must apply the filter for the image to become unrecognizably blurry or smudge-like is different).
Let's consider filtering the same image with a bilateral filter. When the bilateral filter considers pixels along the edge of the coins it will not consider outliers, hence, it will not consider the pixels of the strong white background, rather focusing on the pixels of the coin itself, as they are more similar to the relevant pixel. That is not to say that applying the bilateral filter an exorbitant amount of times won't lead to the

edges eventually blending into their surroundings, but the bilateral filter does, for most intents and purposes, preserve edges.

3.2.3 Minimum- and Maximum Filtering

Two more types of non-linear filters are minimum- and maximum filters. These two filters are often also called dilation and erosion filters, respectively. They are classified amongst the so called morphological filters. In many ways, the minimum and maximum filters are two sides of the same coin, they are very similar in all but one point. As the minimum filter moves along the image, processing as it should, it simply chooses the smallest value inside the current kernel and applies this value to the relevant pixel. Maximum filters work very similarly, but, as one might guess, they instead pick the largest value and assigns that to the relevant pixel. Given the two filters other names, dilation and erosion, it might not be too much of a leap to assume what an image processed by these filters would look like relative to the original. Minimum filters, or dilation filters, simply dilate the darker areas of images (since darker values tend to be assigned lower numbers). Effective if one wishes to thicken the borders of a dark-coloured letter on a light background in an image. Contrarily, the maximum filter, or erosion filter, will erode darker pixels from images. If a black pixel is surrounded by only white pixels, it will be gone after an erosion filter has been applied. Lastly on minimum and maximum filters, a bit about binary images. What are binary images? They are in fact part of what led to the creation of morphological filters such as the here discussed minimum and maximum ones. As one might deduce from the name "binary images" they operate with *only* binary values. In other terms, each pixel can only have one of two values, black and white.

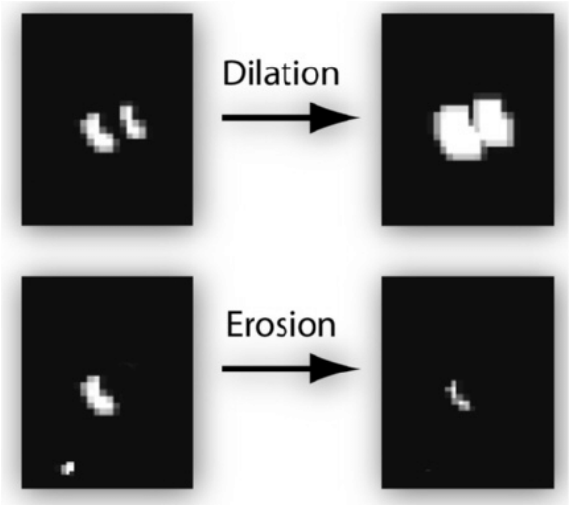


Figure 14: Illustrates the usage of minimum and maximum filters on a grayscale image. Using the minimum filter expands the area of the figure, whereas applying a maximum filter reduces it's size

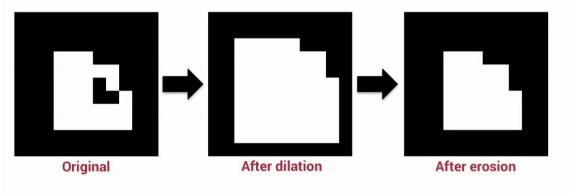


Figure 15: Illustration of three binary images. Using a minimum filter succeeded by a maximum filter fills the gaps in the figure and then decreases it's size

3.3 Further Non-Linear filter related techniques

3.3.1 Mean Shift

The mean shift algorithm is often referred to as a "hill climbing algorithm". Given for instance a scatter plot, or such, the mean shift algorithm will locate the area of the plot where the data-point density is at it's highest, metaphorically speaking, a hill (densest data cluster). Roughly, the algorithm will move (shift) from data point to data point based on the regional mean (mean of current cluster). What is considered in calculating the mean may vary from implementation to implementation. Some times calculating a weighted mean may yield more accurate results, depending on what one is searching for. How can one apply this concept in image processing? Mostly in image segmentation. As mean shift is quite prolific at finding and recognizing clusters, it is also quite good at recognizing "clusters" in images, i.e. different parts of an image. Segmenting an image, categorizing different parts of it, can allow you to perhaps apply different filtering algorithms for different areas of the image, maybe help with edge detection, and in general help preserve the intended look and data of the image. Furthermore it has applications in for instance computer vision and pattern recognition. Important (sub-)fields of image processing.

Figure 16: Four different images, all segmented by a mean shift algorithm. The segmentation enables one to analyze different areas in different ways. Maybe one would rather filter noise away from water-areas or different coloured areas with different filtering techniques

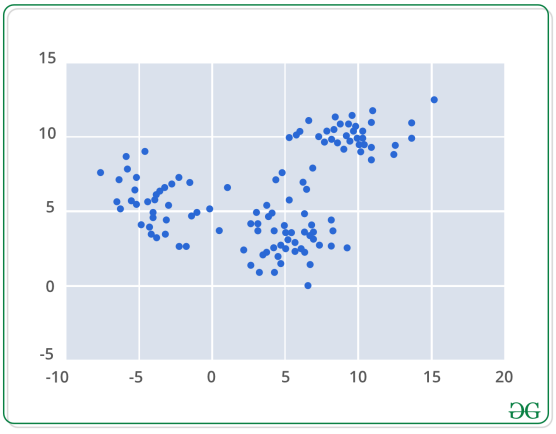


Figure 17: A typical graph for which the mean shift algorithm could be applied. The clusters can be viewed as hills. Increased data point density would imply a "steeper hill". One can roughly recognize the outline of three main clusters here

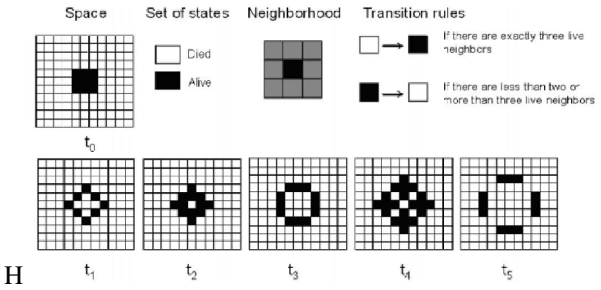


Figure 18: An illustration of Conway's game of life. An important milestone in the study of Cellular Automata, and also a good example of what Cellular Automata can look like

3.3.2 Cellular Automata

Cellular automata is something one probably has heard about in other contexts than image processing. Tracing it roots back to the late 40s, as the brainchild of the two late great scientists Stanislaw Ulam, and of course, the father of computer science himself Neumann Janos, commonly known as John von Neumann. Von Neumann was working on self-replicating-systems when he, with some help from Ulam, came up with his Cellular Automaton. Proving that his self-replicating automaton would work. But how can one apply this seemingly unrelated part of computer science to image processing? First of all, let's assess how cellular automata work.
In essence, cellular automata are grids of cells, each containing what can be considered a finite state machine. They can be very useful in the analysis of many different systems, as well as in the fields of physics and biology. A famous example/application of cellular automata is Conway's game of life[Gar]. Now how specifically does one apply a cellular automaton to image processing? As of now one of the most major application areas is noise removal in binary images. [SH10] As demonstrated by Rosin and Sun[RS11] it can also be used for edge detection, which, as previously mentioned, is an *essential* part of image processing.

3.4 Anisotropic Diffusion

As discussed, there are several scenarios in which non-linear filters can help improve the clarity of images, but sometimes one may actually wish to blur certain areas of an image. Say, for instance, one wishes to recognize the outline of a person, an animal, a car, or similar in an image. Using edge-detection one might well find the edges accurately, but what when the edge-detection algorithm also detects features of the figures as edges, and not only the outline? This is where (anisotropic) diffusion can come into play. Diffusion in general will "smudge out" the image, letting edges and borders flow into one another creating an image where shapes may be hard to distinguish from one another. This is useful, as this is, in a sense, what we wish to do with the "inner details" of the figure we are trying to find the outline of. Still, there is somewhat of a problem remaining, the fact that only diffusing the image will lead to *all* the edges and borders being hard to detect. Where anisotropic diffusion veers of from standard diffusion is by preserving the strongest edges or borders, the outlines. By controlling what is diffused and not, we diffuse only the contents inside the borders, hence making the most prominent edges all the more prominent relative to the rest of the image. This will allow us to figure out the outline of a figure more accurately, as the outline tends to be the most prominent border.

3.5 Typical applications of Non-Linear filters

The subject of where one can apply Non-Linear filters has already been touched upon. Likely, the most prominent application area of non-linear filters in image processing is in noise filtering. Filtering away noise can be invaluable for increasing the clarity of a given image. A further application of non-linear filters is edge detection. Edge detection can be of vital importance (quite literally) in image processing, especially when applied in the context of computer vision, and for instance, autonomous cars. Non-Linear filters may also help in image enhancement and restoration, mostly through aforementioned applications such as

H

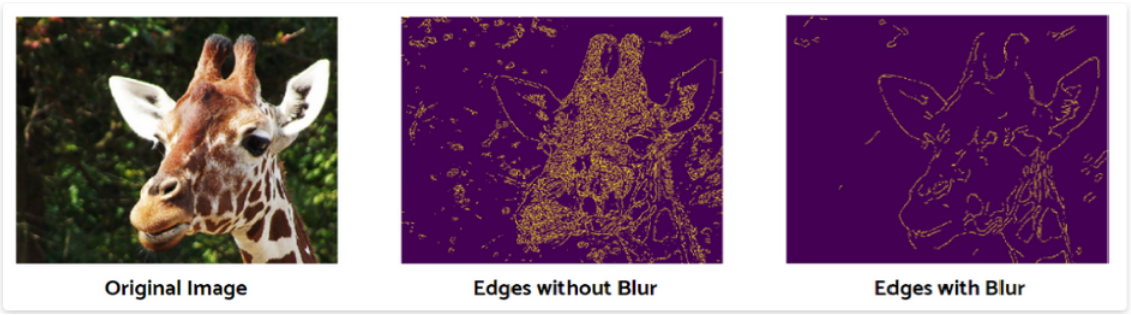


Figure 19: Filtering can be a double edged-sword. Depending on what data one is searching for, one may not need to preserve every single edge in an image. For this scenario, if one is trying to recognize a clear outline of the giraffe, it is easier to find in a blurred version of the image

H

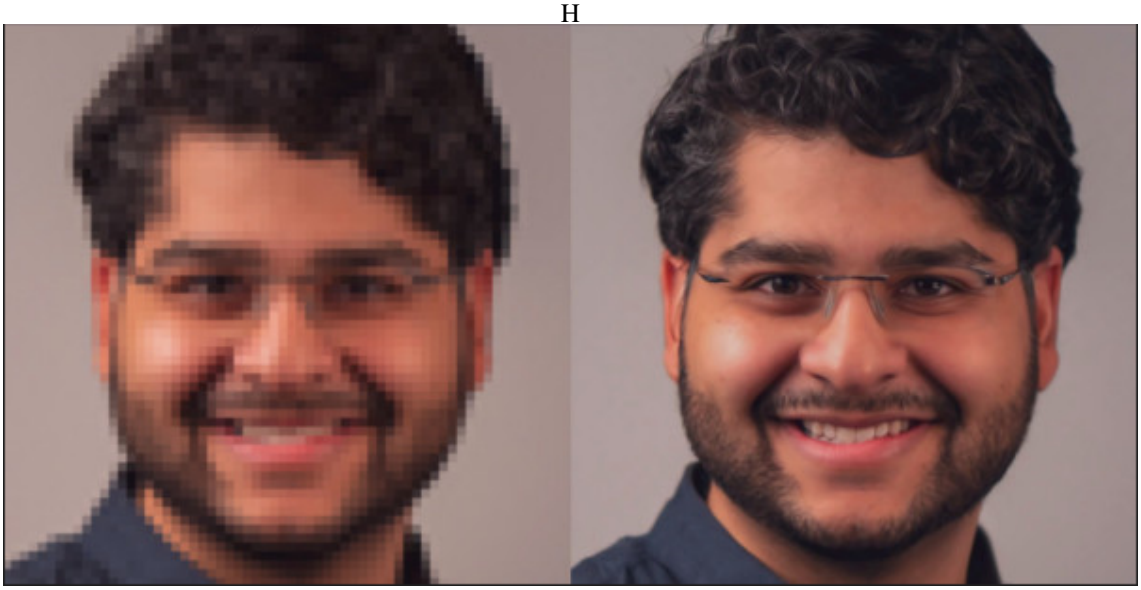


Figure 20: Non-Linear filtering can be a vital component in image enhancement and upscaling. In this photo the image has been upscaled from 64x64 to an impressive 1024x1024

noise-removal, pattern recognition (edge detection/segementation) and further techniques such as increasing contrast. Image restoration and enhancement can even be used to increase the resolution (upscaling) of images and videos, allowing old films to be re-released in higher resolutions. From the old graininess of VHS to HD and full HD all the way to UHD and 4k/8k resolutions. This process is of course not magic though, and it cannot create something that does not exist, hence it also cannot recover completely lost information[SL16]

3.6 Conclusion

Now that we have discussed some different image processing algorithms, specifically several linear and non-linear filtering ones it is time to assess their use cases, and to compare them. A first obvious question that may arise is whether any specific type of filtering is simply objectively better. As with many situations in life, and different fields of computer science, this is hard to find. Just as it may be hard to determine a specific sorting algorithm as objectively the best in any given scenario, it is equally hard to determine an objectively best filtering algorithm. In some programs or prototypes it may be prudent to apply a simple filtering algorithm as nothing too extensive is needed. For ease of use and understanding, many linear

filters may well hold the edge over most non-linear filters. Many non-linear filters can seem less intuitive, and hence be harder to understand and implement, possibly being overly complex for certain scenarios. It is also important to note that in most use cases, be that in pure image processing, signal processing or computer vision, many filters are used in combination to achieve optimal results. Applying specific filtering techniques for specific scenarios and then weaving all the data they produce together may well produce far more accurate data than taking a "one-size fits all" approach to filters.
Filtering can be an incredibly helpful tool in processing images. Removing noise from images, increasing clarity, detecting edges and such have applications in the closely related field of computer vision as well, but none of this comes without it's drawbacks. As we have seen, applying a wrong filter can easily lead to a decrease in clarity of images, perhaps only succeeding in making an image more blurry. Moreover, even if applying "the optimal" filtering algorithm it by no stretch of the imagination guarantees a completely noise and error free result. Filtering is not magic, certain images may be past a point in which it physically cannot get any clearer. This also leads into another issue, namely over-processing. Passing a signal or an image through a filter one time (or more) too many can lead to results which no longer resemble the original, what it was supposed too look like, or lead to unnatural results. In these cases, one may very well consider whether the application of a filter actually helped. If one in certain scenarios realizes that the answer for this case was actually no, one may already have spent far too many resources and hours on something that ended up being producing nothing practical. Resource management is very often of the essence.
Even with all the potential drawbacks that filtering can bring, the pros most likely heavily outweigh the cons. As technology advances, new techniques and new equipment will lead to even more advanced image processing. With the advent of AI, many of it's techniques may be appended to already existing techniques, or spawn new ones entirely. This will all surely lead to even more advanced filtering techniques, and with the blistering increase of data (big data) out there, there is no lack in training data for all AI/ML/Neural Network's needs. This will probably be the most major source of improvement for future filtering techniques. Another important area of improvement is efficiency. Increased hardware performance will lead to faster filtering algorithms working even more efficiently. Further research, and development of the algorithms will also lead to a better (lower) time-complexity, hence increasing the filtering algorithms efficiency further. This is of the utmost importance in, for instance, computer vision for autonomous cars. Faster processing and filtering of the image data a self-driving car collects can be the difference between life and death, and can drastically improve the safety of these self-driving cars.

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.

[Gar] Martin Gardner. Mathematical games-the fantastic combinations of john conway's new solitaire game, life, 1970. *Scientific American, October*, pages 120–123.

[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.

[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].

[RS11] Paul Rosin and Xianfang Sun. *Cellular Automata as a Tool for Image Processing*, pages 233–251. 09 2011.

[SH10] P. Selvapeter and Wim Hordijk. Cellular automata for image noise filtering. pages 193 – 197, 01 2010.

[SL16] T. Sharmila and Leon Leo. Image upscaling based convolutional neural network for better reconstruction quality. pages 0710–0714, 04 2016.

[Sze22a] Richard Szeliski. *3.2 Linear filtering*, page 119–131. Springer, 2nd edition, 2022.

[Sze22b] Richard Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* Springer, 2022.