DIGITAL CULTURE
ITMO UNIVERSITY

Spatial Filtering

High School of Digital Culture
ITMO University
dc@itmo.ru

Contents

1 Basic image processing techniques

In this lecture we will consider basic image processing techniques. First of all, lets' recall what image processing is. Image processing includes methods and tasks where the input and output data are images. Why do we need image processing? In order to make an image better for human or computer perception: either from an aesthetic point of view or for further analysis, both visual or automated by computer.

Let's look at some examples of such methods. The simplest methods of image processing include methods of noise reduction, intensity changes, contrast enhancement, sharpening, color correction. This slide shows some examples how the image od Lena can be aesthetically "corrected" using simple image processing techniques. By the way, this photo appeared on this slide not by accident. This is a photo from Playboy that was selected by one of image processing experts to illustrate a new technique mentioned his article in the 1970s. After the the article was published, this picture became a standard test image for all kinds of image correction and processing methods, named after the model in the picture - Lena.

Another example of image processing for aesthetic purposes is HDR picture construction (High Dynamic Range). Such pictures are usually sewed from several pictures taken with different exposures from the same point at the same time.

Also, the use of various special effects can be attributed to the methods of image processing in order to improve it from an aesthetic point of view.

Here are more examples of image pre-processing methods that are widely used to simplify further image analysis: binarization (separation of the background and the objects), segmentation, contours or edges detection in an image. These are typical examples of image preprocessing that facilitate the task of subsequent semantic analysis.

1.1 Spatial filtering

Some basic gray level transformations Before diving into image processing techniques, let's recall how an image is usually presented. An image is essentially a two-dimensional function that maps spatial coordinates (x, y) to a vector representing the color of a point at a given position. In the case of a grayscale (black and white) image, the color of one point in the image can be expressed in a one-dimensional quantity, which is the value of intensity (or brightness) of a given point.

As we discussed earlier, In the case of a color image, the color is most often specified by a three-dimensional vector.

A discrete image is represented as a discrete function. The spatial plane of the image is represented as a coordinate grid. One grid cell corresponds to one pixel.

A variety of image processing techniques fall into two broad categories: spatial domain processing (spatial techniques) and frequency domain processing (frequency techniques). The term spatial domain refers to the image plane as such, and spatial image processing techniques are based on direct pixel manipulation. Frequency domain processing techniques are based on modifying the signal obtained by applying the Fourier transform to the original image. We will start with spatial methods, and then we will consider frequency representation of images, Fourier transform and frequency methods.

Many image processing techniques operate on pixel neighborhoods. The slide shows an example of a 3x3 neighborhood of a point (x, y). All spatial processing methods are described by the equation $g(x, y) = T[f(x, y)]$, where $f(x, y)$ is the original image, $g(x, y)$ is the processed image, and T is an operator over f, defined in some neighborhood of the point (x, y). T can also operate on a sequence of input images, for example, performing elementwise addition of several images.

The T operator is executed at each point (x, y) using the pixels within the neighborhood of the given point, resulting in the output g for that point. Neighborhoods are most often square or rectangular. In the simplest case, the neighborhood is 1x1 (i.e., it consists of one pixel). In this case, g depends only on the value of f at the point (x, y). Such transformations are called element-wise, or gradational. The operator T in this case becomes the mapping function $s = T(r)$, where in the case of grayscale images, for simplicity of notation, r and s are variables denoting the intensity values of the images $f(x, y)$ and $g(x, y)$ at each point (x, y).

For example, if $T(r)$ has the form shown in the chart on the left, then the result of such a transformation will be expressed in obtaining an image with a higher contrast than the original. Pixels with intensity values less than m will become darker, and pixels with intensity values greater than m will become lighter. In the extreme case shown on the right chart, the $T(r)$ transformation gives a binary (two-gradation) image - all pixels in the image darker than the threshold m will become black, and all pixels lighter than m will become white. Such a transformation is called a threshold function.

The chart on the screeen shows the most common gradationional transformations used to enhance images. The transformation of the image into a negative with brightness in the range $[0, L-1]$ is carried out using the function $s = L-1-r$. This flipping of the brightness levels creates the equivalent of a photographic negative.

Linear stretching, logarithmic and power transformations are among the most commonly used. Logarithmic transform maps a narrow range of low intensity values in the original image to a wider range of output values. This allows to get more distinguishable details in the darkened areas of the image, but the bright areas of the image will lose details. Power transformation acts in a very similar way, but

unlike logarithmic functions, a whole family of curves of possible transformation arises here, obtained by simply changing the parameter $/gamma$.

Another common transformation is the use of piecewise linear functions. The main advantage of piecewise linear functions is that their shape can be arbitrarily complex. The main disadvantage is that a large number of parameters must be specified for their description. One of the simplest cases of using piecewise linear functions is a transformation that enhances contrasts. On the screen you can see a typical transform used to enhance contrast, the result of applying it to a low-contrast image, and the result of the thresholding.

2 Historgrams

The histogram of a digital image with gray levels in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$, where r_k is the k-th gray level and n_k is the number of pixels in the image having gray level r_k. It is common practice to normalize a histogram by dividing each of its values by the total number of pixels in the image, denoted by n. Thus, a normalized histogram is given by $p(r_k) = n_k/n$, for $k = 0, 1, ..., L-1$. Loosely speaking, $p(r_k)$ gives an estimate of the probability of occurrence of gray level r_k. Note that the sum of all components of a normalized histogram is equal to 1.

Histograms are the basis for numerous spatial domain processing techniques. Histogram manipulation can be used effectively for image enhancement, it provides useful image statistics which are useful for image compression, segmentation and used in many other image processing applications. Histograms are simple to calculate, thus making them a popular tool for real-time image processing.

Let's look at some examples of images and their histograms. On this slide we see two versions of the same image: dark and light, and corresponding histograms. The horizontal axis of each histogram plot corresponds to gray level values, r_k. The vertical axis corresponds to values of $h(r_k) = n_k$. Histograms slides are are non-normalized and simply plots of$h(r_k) = n_k$ versus r_k.

Note that the components of the histogram corresponding to the dark image are concentrated on the low (dark) side of the gray scale. Similarly, the components of the histogram of the bright image are biased toward the high side of the gray scale.

On the next slide there are two more examples - and image with low contrast and an image with high contrast and corresponding histograms. An image with low contrast has a histogram that is narrow and is centered toward the middle of the gray scale. For a monochrome image this implies a dull, washed-out gray look. Finally, we see that the components of the histogram in the high-contrast image cover a broad range of the gray scale and. Intuitively, it is reasonable to conclude

that an image whose pixels tend to occupy the entire range of possible gray levels and, in addition, tend to be distributed uniformly, will have an appearance of high contrast and will exhibit a large variety of gray tones. The net effect will be an image that shows a great deal of gray-level detail and has high dynamic range. It will be shown shortly that it is possible to develop a transformation function that can automatically achieve this effect, based only on information available in the histogram of the input image.

2.1 Histogram-based operations

Histogram-based operations, which change an image by transforming its histogram, are simple and effective methods for contrast adjustment. The purpose of a histogram-based operation is to enhance or suppress pixels of an image within a specific range of gray levels. Histogram-based operations can be linear and non-linear. Linear histogram stretching and compression are common examples of linear operations. With linear operation, for example, it is possible to "stretch" the histograms of too dark and too light images just by multiplying gray level of every pixel in a source image by some value. This historgam "stretching" will lead to a high contrast image.

But a linear transformation can rarely produce a uniform distribution, and thus won't get you to a really high contrast image. That's where non-linear transformation, such as histogram equalization, might become handy. It is the common example of a non-linear histogram-based operation.

The task of histogram equalization is to transform a histogram of any shape to a histogram which has the same frequency along the whole range of possible gray values, or to get a uniform distribution. So we want the resulting histogram to be "flat", as on the right plot on this slide.

This is realized by equally partitioning the cumulative histogram of the original image into a number pieces equal to a number of discrete gray levels. Each piece will correspond to one digital number in the equalized image. A plot on the left illustrates this cumulative cure. Let's now define this formally.

Let the variable r represent the gray levels of the image to be enhanced. Let's assume that r has values in the interval $[0, L-1]$, with $r = 0$ representing black and $r = L-1$ representing white. The probability of occurrence of gray level r_k in an image is approximated by

$$p_r(r_k) = \frac{n_k}{n}, \quad k = 0, 1, 2, ..., L-1 \tag{1}$$

where n is a total number of pixels in the image, n_k is the number of pixels that have gray level r_k, and L is the total number of possible gray levels in the image. Histogram equalization (or histogram linearization) is a transformation of the form $s = T(r)$, that produce a level s for every pixel value r in the original

image.

$$s_k = T(r_k) = \sum_{i=0}^{k} p_r(r_i) = \sum_{i=0}^{k} \frac{n_i}{n} \tag{2}$$

So, a processed (output) image is obtained by mapping each pixel with level r_k in the input image into a corresponding pixel with level s_k in the output image via Eq. 2.

If we replace discrete histograms with continuous functions, probabilities with probability density functions and summations with integrals, it can be proved that such transformation will produce a uniform probability density function. In discrete case it cannot be proved, but this transformation does have the general tendency of spreading the histogram of the input image so that the levels of the histogram-equalized image will span a fuller range of the gray scale.

Let's look at some examples. This slide shows the result of performing histogram equalization on too dark and too light images. You can notice significant improvement in contrast for these images. On the right you can compare histograms of the original image and the result image, after histogram equalization. You can notice that after the equalization, histogram spans the full spectrum of the gray scale and it doesn't have any major spikes - it is very close to uniform.

And the next slide shows the result of performing histogram equalization on low- and high-contrast images. Note, that histogram equalization did not produce a significant visual difference in the last image because the histogram of this image already spans the full spectrum of the gray scale. It is of interest to note that, while all these histograms are different, the histograms of equalized images are visually very similar. This is not unexpected because the difference between the images is simply one of contrast, not of content.

These examples illustrate the power of histogram equalization as an adaptive enhancement tool. We discussed already many advantages of having gray-level values that cover the entire gray scale. In addition to producing gray levels that have this tendency, the histogram equalization method has the additional advantage that it is fully "automatic." In other words, given an image, the process of histogram equalization consists simply of implementing Eq. 2, which is based on information that can be extracted directly from the given image, without the need for further parameter specifications.

3 Thresholding

One of the image processing tasks which relies on histograms in binarization via thresholding. Binarization is a process ot separation of pixel values into two groups: objects and background. The results of binarization is a binary image. Binarization with thresholding is the simplest method of segmenting images.

Suppose that the intensity histograms shown on the slide correspond to some images. The histogram on the left corresponds to an image composed of light objects on a dark background, and the histogram on the right - to an image of two types of objects (one lighter than another) on a dark background. One obvious way to extract the objects from the background is to select a thershold T that separates the modes on the histogram (or a couple of thresholds T_1 and T_2 for the second image to separate all the modes).

When a threshold T depends only on gray-level values of the image (so it is the same for all pixel positions), the threshold is called global. If T depends on both gray-level values and coordinates of a given pixel, the threshold is called local. And if in addition T depends on some local property of a point (for example, the average gray level of a neighborhood centered in a given point), it is called adaptive.

One trivial way to choose a threshold is manually, by visual inspection of the image histogram. There are also many methods which can choose a threshold automatically. Here is an example of one of the simplest ones.

1. Select an initial estimate for T at random.
2. Segment the image using T. this will produce two groups of pixels: G_1 consisting of all pixels with gray level values $< T$ and G_2 consisting of pixels with values $\geq T$.
3. Compute the average intensity values μ_1 and μ_2 for the pixels in the regions G_1 and G_2.
4. Compute a new threshold value:

$$T = \frac{1}{2}(\mu_1 + \mu_2) \tag{3}$$

5. Repeat steps 2 through 4 until the difference in T in successive iterations is smaller than a predefined parameter.

This slides shows a few examples of thresholding methods applied to a gray-scale image. The original image is in the top left corner. The result of global thresholding is in the top right corner, and the result of adaptive thresholding is in the bottom right corner.

4 Fundamentals of Spatial Filtering

So far we've mostly looked at point processing, where enhancement at any point in an image depends only on the gray level at that point. Larger neighborhoods allow considerably more flexibility. The general approach is to use a

function of the values of original image f in a predefined neighborhood of (x, y) to determine the value of output image g at (x, y). One of the principal approaches in this formulation is based on the use of so-called masks (also referred to as filters, kernels, templates, or windows). Basically, a mask is a small (say, 3×3) 2-D array, in which the values of the mask coefficients determine the nature of the process, such as image sharpening. Enhancement techniques based on this type of approach often are referred to as mask processing or filtering.

The name filter is borrowed from frequency domain processing. We'll talk about it a bit later. In frequency domain "filtering" refers to passing, modifying, or rejecting specified frequency components of an image. For example, a filter that passes low frequencies is called a lowpass filter. The net effect produced by a lowpass filter is to smooth an image by blurring it. We can accomplish similar smoothing directly on the image itself by using spatial filters.

Now let's look at filtering operations that are performed directly on the pixels of an image. We use the term spatial filtering to differentiate this type of process from the more traditional frequency domain filtering.

Spatial filtering modifies an image by replacing the value of each pixel by a function of the values of the pixel and its neighbors. If the operation performed on the image pixels is linear, then the filter is called a linear spatial filter. Otherwise, the filter is a nonlinear spatial filter.

The mechanics of linear spatial filtering using a 3×3 kernel are illustrated on the slide. The process consists simply of moving the filter mask from point to point in an image. At any point (x, y) in the image, the response of the filter the sum of products of the kernel coefficients and the image pixels encompassed by the kernel.

In general, linear spatial filtering of an image f of size $M \times N$ with a filter mask of size $m \times n$ is given by the expression on the slide. Here x and y are varied so that the center (origin) of the kernel visits every pixel in f once. For a fixed value of (x, y) we compute the sum of products for a kernel of arbitrary odd size. This equation is a central tool in linear filtering.

In mathematics, this operation on two functions (f - image, and w - filter) is called *spatial correlation*. Correlation consists of moving the center of a kernel over an image, and computing the sum of products at each location. Another very important operation is *spatial convolution*. The mechanics of spatial convolution are the same, except that the correlation kernel is rotated by 180°. Thus, when the values of a kernel are symmetric about its center, correlation and convolution yield the same result. For this reason, linear spatial filtering often is referred to as "convolving a mask with an image." Similarly, filter masks are sometimes called *convolution masks*. The term *convolution kernel* also is in common use.

The reason for rotating the kernel in convolution will become clear in the following discussion.

4.1 Correlation and convolution

The best way to explain the differences between correlation and convolution is by example. Let's look at this 1-D illustration.

The first row shows a $1 - D$ function, f, and a kernel, w. The kernel is of size 1×5. the only difference between correlation and convolution is that for convolution the kernel is pre-rotated by 180°.

The second row shows the starting position used to perform correlation and convolution, in which w is positioned so that its center coefficient is coincident with the origin of f.

The first thing we notice is that part of w lies outside f, so the summation is undefined in that area. A solution to this problem is to pad function f with enough 0's on either side. In general, if the kernel is of size $1 \times m$, we need $(m-1)/2$ zeros on either side of f in order to handle the beginning and ending configurations of w with respect to f. the next, third, row shows a properly padded function. In this starting configuration, all coefficients of the kernel overlap valid values.

The first correlation and convolution values are the sum of products with respectful kernels in this initial position. To obtain the rest of the values of correlation and convolution, we continue to shift the relative positions of w and f one step at a time, until we reach the end of f. Third row on this slide shows the correlation and convolution results.

Note that it took 8 steps to fully shift w past f so the center coefficient in w visited every pixel in f. Sometimes, it is useful to have every element of w visit every pixel in f. For this, we have to start with the rightmost element of w coincident with the origin of f, and end with the leftmost element of w being coincident the last element of f (additional padding is required here). the last row shows the result of this extended, or full, correlation and convolution. As you can see, we can obtain the "standard" correlation and convolution by cropping full, extended versions.

There are two important points to note. First, both correlation and convolution are functions of displacement of the filter kernel relative to the image. In other words, the first values of correlation and convolution correspond to zero displacement of the kernel, the second correspond to one unit displacement, and so on. The second thing to notice is that correlating a kernel w with a function that contains all 0's and a single 1 yields a copy of w, but rotated by 180°. A function that contains a single 1 with the rest being 0's is called a discrete unit impulse. Correlating a kernel with a discrete unit impulse yields a rotated version of the kernel at the location of the impulse.

In case of convolution, the result of pre-rotating the kernel is that now we have an exact copy of the kernel at the location of the unit impulse. It is a foundation of linear system theory, that convolving a function with an impulse

yields a copy of the function at the location of the impulse.

This 1-D concepts extend easily to images.

4.2 Correlation and convolution of a 2-D kernel

For a kernel of size $m \times n$, we pad the image with a minimum of $(m-1)/2$ rows of 0's at the top and bottom and $(n-1)/2$ columns of 0's on the left and right. In the example on this slide, m and n are equal to 3, so we pad f with one row of 0's above and below and one column of 0's to the left and right.

You can also see on the slide the initial position of the kernels for performing correlation and convolution, and the final result after the center of w visits every pixel in f, computing a sum of products at each location. As before, the result of the correlation is a copy of the kernel, rotated by 180°. For convolution, we pre-rotate the kernel as before and repeat the sliding sum of products. You see again that convolution of a function with an impulse copies the function to the location of the impulse. As noted earlier, correlation and convolution yield the same result if the kernel values are symmetric about the center.

Summarizing this all in equation form, the correlation of a kernel w of size $m \times n$ with an image $f(x, y)$, is given by top equation on this slide:

$$(wf)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t) \tag{4}$$

As before, here $a = (m-1)/2$, $b = (n-1)/2$, and we assume that f has been padded appropriately.

In a similar manner, the convolution of a kernel w of size $m \times n$ with an image $f(x, y)$ is defined as

$$(w \star f)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \tag{5}$$

Here the minus signs align the coordinates of f and w when one of the functions is rotated by 180°. This equation implements the sum of products process to which we'll refer as linear spatial filtering. That is, linear spatial filtering and spatial convolution are synonymous.

Properties Convolution operation satisfies the following algebraic properties:

- Commutativity
- Associativity
- Distributivity

- Associativity with scalar multiplication

Correlation is neither commutative nor associative.

Because convolution is commutative, it is immaterial whether w or f is rotated, but rotation of the kernel is used by convention.

5 Smoothing

5.1 Smoothing – Linear Filters

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called averaging filters.

The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in gray levels. Because random noise typically consists of sharp transitions in gray levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable features of an image) also are characterized by sharp transitions in gray levels, so averaging filters have the undesirable side effect that they blur edges. A major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant" we mean pixel regions that are small with respect to the size of the filter mask.

On the slide we show two 3×3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask. A spatial averaging filter like this one, in which all coefficients are equal is sometimes called a box filter.

The second mask shown on the slide is a little more interesting. This mask yields a so-called weighted average, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown on the right the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask. This strategy of weighing the center point the highest is simply an attempt to reduce blurring in the smoothing process. We could have picked other weights to accomplish the same general objective. In practice, it is difficult in general to see

differences between images smoothed by using either of the masks on the slide, or similar ones, because the area these masks span at any one location in an image is so small.

5.2 Moving average

Let's look how moving average transforms an image. Here, F - a source image, G - a resulting image. We start with a sliding mask in the left upper corner. The resulting value for a position highlighted in the image G is an average of all pixels under the mask in the image F. Since all values under the mask in this position are 0, the resulting value for this position in G is also 0.

Then we move the mask one pixel to the right and compute a new average for this position. In this case we have one pixel with value 90, so the resulting average is 10.

We will continue to move this sliding window and compute values for all internal positions in the image G.

5.3 Dealing with edges

You probably noticed, that we can't compute values along the edges of G. This situation is common for all neighborhood-based filters, and there are several ways to handle it. The simplest is do not compute resulting values for positions along the edge. The resulting filtered image will be smaller than the original, but all the pixels in the filtered image will have been processed with the full mask. If the result is required to be the same size as the original, a common approach is to "pad" the source image along the edges with 0's 9or other constant gray level), or padding by replicating or mirroring edge columns and rows. Number of padded pixels depends on the filter size. For 3×3 filter as in this example we need to pad one row and one column on all sides of the original image. Often the following terminology is used to describe padding.

- Valid padding - no padding done; the resulting image is smaller than the source one.
- Same padding - source image is padded along the sides so that filtering doesn't change image size.

Now let's look at an example of smoothing with moving average. On the left - a source image, on the right - a result of smoothing with moving average. Notice, that the resulting image has some edge effects. This is due to the fact that all the pixels in the neighborhood have the same influence on the resulting value. To eliminate these edge affects, we can choose another filter, such as weight contribution of neighborhood pixels correlates with their closeness to the center.

5.4 Smoothing – Gaussian filter

One particular filter which has this property is a Gaussian filter, also often called a Gaussian blur. This is another very commonly used smoothing filter. It uses a kernel that represents the shape of a Gaussian 'bell-shaped' hump. The idea of Gaussian smoothing is to use this 2-D distribution as a 'point-spread' function. The kernel coefficients diminish with increasing distance from the kernel's centre. Central pixels have a higher weighting than those on the periphery.

Mathematically, applying a Gaussian filter to an image is the same as convolving the image with a Gaussian function. This is also known as a two-dimensional Weierstrass transform.

The Gaussian kernel is continuous. To apply this filter to digital image, we need a discrete equivalent. Discrete approximation to the Gaussian function theoretically requires an infinitely large convolution kernel, as the Gaussian distribution is non-zero everywhere. Fortunately the Gaussian distribution approaches very close to zero at about three standard deviations from the mean. 99% of the distribution falls within 3 standard deviations. This means we can normally limit the kernel size to contain only values within three standard deviations of the mean. Most commonly, the discrete equivalent is the sampled Gaussian kernel that is produced by sampling points from the continuous 2D Gaussian function.

The standard deviation of a filter can be interpreted as a measure of its size, and has direct influence on the amount of "blurring". Larger values of σ produce a wider peak (greater blurring). It's important to note that kernel size must increase with increasing σ to maintain the Gaussian nature of the filter. At the edge of the mask, coefficients must be close to 0.

This slide shows example of image blurring with Gaussian filters. A filter with greater σ produces more blur, and it also requires larger size of a mask.

And here we see a comparison of filtering results with a mean (averaging) filter and a Gauss filter.

5.5 Smoothing – Median Filter

Other commonly used spatial filters are order-statistics non-linear filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result. The best-known example in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the gray levels in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median). Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective

in the presence of impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.

Let's look at a left most image on the slide. It is an X-ray image of a circuit board heavily corrupted by bipolar impulse noise (also well known as salt-and-pepper noise). To illustrate the point about the superiority of median filtering over average filtering in situations such as this, in the center we show the result of processing the noisy image with a 3*3 neighborhood averaging mask, and the right most image is the result of using a 3*3 median filter. The image processed with the averaging filter has less visible noise, but the price paid is significant blurring.The superiority in all respects of median over average filtering in this case is quite evident. In general, median filtering is much better suited than averaging for the removal of additive salt-and-pepper noise.

5.6 Sharpening

Now, when we know how to blur an image, it's easy to get sharpening. Image sharpening is also widely used - from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems. The principle idea of sharpening is to highlight fine detail in an image or enhance detail that has been blurred. What do we lose when blur an image? We lose fine-grained details. If we add these details to the original image, we'll get a sharpened image. This idea is used in a simple sharpening technique called "unsharp filter" or "unsharp masking". Its name derives from the fact that the technique uses a blurred, or "unsharp", negative image to create a mask of the original image. The unsharp mask is then combined with the original image, creating an image that is less blurry than the original.

We will come back to image sharpening later in more details, when we will be discussing detection of discontinuities - isolated points, lines and edges.

sent by neurons. Each channel has its weight (or, say, the relative importance of the connection or connection strength). For example, if the number 10 passes through the connection and the connection weight is 0.5, the neuron input is $0.5 \cdot 10 = 5$. Neurons are organized in layers to structure connections. Neurons of one layer are not interconnected, because they connect only to neurons of the immediately preceding and immediately following layers. In reality, it is represented in matrices and calculated using linear algebra. But we will leave it out of the way for now.

Basically, this is it. So, we've discussed the main problems and schemes of machine learning. Let's turn to the minimally required apparatus of probability theory and mathematical statistics because we will need it further on.

4 Some Statistical Theory

So, let's talk about the elements of statistical data analysis we will need further on. In general, we will discuss relevant statistical methods right before using them. At the moment, it makes sense to recall how to identify the main characteristics of univariate samples.

It all begins with a statistical population. A population ξ is a random variable we are dealing with, and it also gives us data. So here's the example. Let the population ξ be the amount of money a person spent in a supermarket. The amount of money spent (the value ξ) varies from a person to person and from a visit to visit. In practice, a set of values ξ is always finite, because, for example, we consider the data for the past month (or some period), and a finite number of customers is served within that month.

We suppose that a random variable ξ obeys a law. We will call it the distribution of the random variable. Perhaps you recall from statistics the meaning of such concepts as Bernoulli distribution, as well as normal, uniform, and binomial distribution. For now, you don't need a thorough knowledge of the distribution concept. You just need to remember that we assume there's a law of changes in the variable ξ. And how do the laws differ? Correct! They differ in probabilities.

In our example, of course, an amount of money spent cannot be negative. Hence, the distribution of our random variable is such that the probability of assigning negative values to the possible values is zero. At the same time, people rarely spend more than $ 500 in a supermarket. Therefore, the probabilities that ξ takes the values greater than $ 500 are extremely low. But what is within the boundaries? It would be curious to find out. How do we do it? By using a sample.

Definition 4.0.1 *A sample of size n from a statistical population ξ is a set of independent random variables $X_1, X_2, ..., X_n$ having the same distribution as ξ.*

The justification remains the same. A change in a day or an hour of an observation causes a change in the sample. That's why the sample elements are random variables. In a particular observation, a sample is a set of n numbers $x_1, x_2, ..., x_n$.

It turns out that a sample allows estimating expected value, variance, and mean deviation of a random variable, etc. Let's remind ourselves how to do this.

What is the expected value? To put it simply, it is the average probability value of a given random variable. To clarify it, let's simplify the example. We will consider only those people who have spent a lot in a supermarket or haven't purchased anything. Sometimes customers just don't buy. It isn't rare. Meanwhile, people rarely spend more than $ 500 on groceries. Let's ask how much the average person spends on a purchase (but we still consider only those who spend a lot or don't purchase). The first thing that comes to mind is an arithmetic mean. That is,

$$\text{Average purchase amount} = \frac{0 + 500}{2} = 250?$$

Well, that's odd. For example, 99 of 100 customers spend zero amount and only one spends 500. It's reasonable to write it as follows:

ξ	0	500
P	$\frac{99}{100}$	$\frac{1}{100}$

.

The table describes the distribution of a discrete random variable. The first line of the table contains values, and the second line probabilities (calculated based on the frequency). What can be called $\mathsf{E}\xi$? Apparently, the value:

$$\mathsf{E}\xi = 0 \cdot \frac{99}{100} + 500 \cdot \frac{1}{100} = 5.$$

Don't you think the obtained value is nearer the truth?

It's all good in theory, but an issue arises in practice because we don't know the probability values. The only option is to estimate the true expected value. The obvious choice is a sample mean

$$\overline{X} = \frac{X_1 + X_2 + ... + X_n}{n},$$

that, on a particular observation $(x_1, x_2, ..., x_n)$, turns into the arithmetic mean of sample elements:

$$\overline{X} = \frac{x_1 + x_2 + ... + x_n}{n}.$$

For example, we can consider the sample:

$$X = (0, 11, 2, 3, 9, 2, 8, 6, 3.4, 8, 7.5, 9, 4, 8, 6).$$

Thus, the sample mean is calculated as follows:

$$\overline{X} = \frac{0+11+2+3+9+2+8+6+3.4+8+7.5+9+4+8+6}{15} \approx 5.793.$$

Figure 14 shows the sample elements denoted by blue points. The red point corresponds to the sample mean $\overline{X}$. As you can see, the red point is almost in the middle of our blue points (the sample elements). It can be proved that the sample mean better approximates the true expected value of the population ξ as the sample size increases.

A spread is the second important characteristic. The figure shows that blue points are not close to each other. Therefore, we can ask how they are scattered on average from the mean in every direction. It is like target shooting. We all want to score a bullseye, but many of us only hit the circle around it.

In the purchase example, we can approximately understand how large the spread from the mean is. The spread allows us to predict sales over some period in the worst-case and best-case scenario. Let's move on to calculations. In the supermarket example, we considered the random variable ξ having the following distribution:

ξ	0	500
P	$\frac{99}{100}$	$\frac{1}{100}$

and the expected value:

$$\mathsf{E}\xi = 0 \cdot \frac{99}{100} + 500 \cdot \frac{1}{100} = 5.$$

We can assume that the spread equals $\max|\xi - \mathsf{E}\xi| = 495$, but it doesn't seem right because the value 500 is highly unlikely. Shouldn't we take the average spread? Here comes the variance that, as you know, is defined as

$$\mathsf{D}\xi = \mathsf{E}\left(\xi - \mathsf{E}\xi\right)^2.$$

The distribution of the random variable $\left(\xi - \mathsf{E}\xi\right)^2 = \left(\xi - 5\right)^2$ is described by the table:

$(\xi-5)^2$	25	495^2
P	$\frac{99}{100}$	$\frac{1}{100}$

.

Thus,

$$\mathsf{D}\xi = \mathsf{E}\left(\xi - 5\right)^2 = 25 \cdot \frac{99}{100} + 495^2 \cdot \frac{1}{100} = 2475.$$

That's a large number. It is much greater than those obtained naively. Are we missing something? Look, we compute the expected value of the square and obtain

something like an average squared spread. The spread itself can be estimated using the so-called standard deviation σ that is equal to

$$\sigma = \sqrt{\mathsf{D}\xi}.$$

In our example, $\sigma = \sqrt{2475} \approx 49.75$, so it is nearer the truth.

Since the probability distribution is usually unknown, we want to estimate the variance. A good variance estimator equals

$$S^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2.$$

The following estimator is also common:

$$S_0^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2.$$

It is a so-called unbiased variant of the estimator S^2. This is how we obtain the estimators of the deviation σ:

$$\sigma^* = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}, \quad \sigma_0^* = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2}$$

Let's return to our synthetic example. The following sample:

$$X = (0, 11, 2, 3, 9, 2, 8, 6, 3.4, 8, 7.5, 9, 4, 8, 6).$$

The calculations show that $S^2 \approx 9.63$ and $\sigma \approx 3.1$. It means that the values of our random variable should fall on average within the range:

$$\left(\overline{X} - \sigma^*, \overline{X} + \sigma^*\right).$$

The figure shows the sample, which elements are represented by blue points, a sample mean that is a red point, and the interval endpoints $\overline{X} \pm \sigma^*$ shown in green. As you can see, most sample elements fall within the interval.

5 Conclusion

So, now you know what machine learning is and what problems you can solve using its main methods. It is also important to note that digitalization is gaining ground, and machine learning is one of the main engines of this process. The described methods find many applications in practice. Today, machines help

doctors analyze medical images and diagnose diseases. Many services implement chatbots or voice assistants. Text analysis, marketing, economy, and manufacture are some of the examples of the domains where AI and ML achieved great results.

These technologies are available to everyone. Many tools offer ML methods that you can use to solve different problems (even without complicated calculations). Moreover, some of these tools require no profound knowledge in computer science or math, and you only need to know what to do and what you can get. Coding skills enable you to solve more complicated problems by way of combining and adjusting the methods to your needs. The amazing thing is that many libraries offer vast collections of machine learning methods and algorithms.

Our advice to you is to find what best fits your needs and skills but never miss opportunities. Good luck! :)

Figure 3: Machine learning map

Figure 4: Classical learning

Figure 5: The relationship between the price increment and proximity of the nearest metro station

Figure 6: Naive estimate of true distribution

Figure 7: Regression

Figure 8: Child's classifier

Figure 9: About Clustering

Figure 10: Reinforcement learning.

Figure 11: Q-learning

Figure 12: Ensemble methods.

Figure 13: Neuron

Figure 14: Sample X and its mean $\overline{X}$

Figure 15: Sample X and its parameters