Basic Image Processing

Erdélyi Áron

2020.01.06.

Table of Contents

1 Introduction to image processing

1.1 History

1.1.1 Digital image processing:

Early 1920s: Bartlane cable picture transmission system

- Transmitting newspaper images via submarine cable between London and New York.
- Took about three hours to send a picture.
- First systems supported 5 gray levels.
- Not "real" image processing, since it was not created with a computer, only sent with one.

The real era of digital images only started, when the computers got powerful enough for the task.

Early 1960s:

- 1964: NASA's Jet Propulsion Laboratory started working on algorithms to improve images of the Moon.
  - Images were transmitted by the Ranger 7 probe
  - Corrections were desired for distortions inherent in the on-board camera.

Late 1960s, early 1970s:

- Medical imaging (CT)
- Remote Earth resources observations
- Astronomy

1.1.2 Computer vision

In 1966, Marvin Minsky (MIT) asked his students to "spend the summer linking a camera to a computer, and getting the computer to describe what it saw".

1.1.3 Neural networks

- **1958:** Fran Rosenblatt introduced the Preceptron model.
- **1970s:** Backpropagation algorithm for larger network training.
- **1980s:** Apperiance of CNNs.
- **1998:** First success of CNN.
- **2012:** Imagenet classification challenge won by deep CNN.

Since 2012 we are witnessing the golden age of CV.

1.2 Applications

A couple of examples of applications from the lecture slides:

- Early Hubble Space Telescope images were distorted by a flawed mirror and could be sharpened by deconvolution.
- Instagram filters (Convolution)
- Panoramic images (Feature matching)
- Medical image processing:
  - Ultrasound,

  - 3D imaging, MRI, CT, PET,
  - Image guided surgery, etc...
- Today face detection is a standard feature in smart phones, cameras. (Most popular: YOLO, or SSD with Non maxima supression)
- Facial retargeting
- Active Shape Models
- Emotion recognition
- Identity verification and recognition based on biometrics
- Video surveillance
- 3D scene understanding via laser scanning (Lidar) data:
  - Automated data filtering, correction, vectorization and interpretation.
  - Point cloud classification
- Advanced driver assistance systems:
  - Automatic parking,
  - Collision avoidance,
  - Driver monitoring system,
  - Emergency driver assistant,
  - Lane departue warning system,
  - Lane changing assistance,
  - Pedestrian protection system,
  - Traffic sign recognition, etc ...

1.3 Digital representation of an image

A digital image is a discrete representation of a continuous measurement, usually a 2 or 3 dimensional array. An element of this array is a pixel (picture element). A pixel has a position (its coordinates on the image) and an intensity value. A digital image is discretised both in space and intensity:

- Spatial discretisation is called sampling.
- Intensity discretisation is called quantization.

1.3.1 Sampling

Sampling is the reduction of a continuous signal to a discrete signal. A finite set of values, called samples, are selected to represent the original continuous signal.

In case of 2D signals (images) a grid is used for sampling. The grid points will be represented as pixels. The frequency of the sampling defines:

- The number of grid points,
- The resolution of the image,
- How detailed the discretised image is.

Sampling usually leads to information loss. We have to decide what the smallest detail that we want to keep.

1.3.2 Quantization

Intensity discretisation is referred to as quantization. The digital image quality is highly dependent on how many bits we use for coding the discrete intensity values:

- **Binary:** each pixel is represented by one bit (black or white)
- **Gray scale:** coded usually on 2^n bits.

1.4 The histogram of an image

The histogram $h(k)$ of the image is the number of pixels on the image with k intensity. The histogram normalized with the total number of pixels gives us the probability density function of the intensity values.

1.5 Color Spaces

Two main mixing models:

- Additive model
- Subtractive model

Color spaces specify a coordinate system and a subspace within that system, where each color is represented by a single point.

- RGB
- CMY, CMYK
- HSL/HSV/HSI
- YUV, YCbCr

1.5.1 RGB

This is the most common color model. In this model, there are 3 channels, Red, Green, Blue, hence the name RGB. All of the components depend on luminosity. They need to be coded with the same bandwidth. Changing the intensity in this model is not efficient, because all 3 channels have to be modified

1.5.2 CMY

This color space is most commonly used in printing. It is based on the subtractive model; it describes what kind of ink has to be applied for the desired wavelength (color) to emerge. (Cyan, Magenta, Yellow)

1.5.3 CMYK

Since there is no real black in CMY, in this color space, we add black as a 4th component. (Cyan, Magenta, Yellow, blacK)

1.5.4 CIE

This model is based on how we humans precieve color. It was developed to be completely independent of any device. (Comission Internationale de l'Eclairage)

1.5.5 CIELab, Lab, L*a*b

This space has three channels: one uminace (L), and two color channels (a and b). In this model, the color differences correspond to Euclidian distances.

The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b). The brightness (L) increases from the bottom to the top of the 3D model.

1.5.6 HSI

In this model, we have three channels: Hue, the dominant wavelength of the Spectral Power Distribution, Saturation, the relative purity or the amount of white light in the mixture, and Intensity, which indicates the dominant wavelength in the mixtue of light waves.

This model can be precieved as a cylinder:

- Hue: the angle around the vertical axis.
- Saturation: the distance from the central axis.
- Intensity: the height.

1.6 Dithering

With images there arises the problem of detail loss. Even so, if we want to reduce the number of bits we do the quantization with. Dithering might be a solution for a problem like this, because as humans we can precieve different densities of black and white pixel groups as grayscale.

1.6.1 Quantization for a single bit

Each pixel becomes black or white, the output is dependent on the pixels original gray level. For a grayscale image f let the domain of gray levels be $[0, F_{max}]$. A given pixel receives 1 (white) color, if the following condition holds:

$$f \leq r \cdot F_{max},$$

where $r \in [0, 1]$ is randomly generated for every pixel. With some rearrangement:

$$f + r \cdot F_{max} \leq F_{max}$$

We add random numbers, with values between 0 and F_{max} to the original image. After this we threshold the image with a threshold value of F_{max}.

1.6.2 Multiple bits

If the gray value of the output image can be represented by multiple bits, the result can be significantly better.

Let n be the number of different gray levels in the output image. We add random numbers $r \in [0, \frac{F_{max}}{n-1}]$ to the image pixel values. Then we threshold the image, with multiple threshold levels: Pixels with the modified gray levels larger than $k \cdot \frac{F_{max}}{n-1}$ but not larger than $(k+1) \cdot \frac{F_{max}}{n-1}$ receive the output value k.

2 2D convolution and its applications

2.1 Mathematical background

We look at the image as a 2D function: $f(x, y)$, where x, y are the pixel coordinates, and $f(x, y)$ is a gray level from $[0, 255]$. We can define different transformations:

- Intensity inversion:
$$g(x, y) = 255 - f(x, y)$$
- Intensity shift with constant:
$$g(x, y) = f(x, y) + c$$
- Weighting:
$$g(x, y) = f(x, y) \cdot w(x, y)$$
- Average on an N neighborhood:
$$g(x, y) = average\_N(f(x, y))$$

There are two important properties of the transformations we want to use on images: **linearity** and **shift invariance**.

2.1.1 Linearity:

$$T[f_1(x, y) + f_2(x, y)] = T[f_1(x, y)] + T[f_2(x, y)],$$
$$T[\alpha \cdot f(x, y)] = \alpha \cdot T[f(x, y)].$$

e.g.: weighting is linear, intensity shift is non-linear.

2.1.2 Spatial invariance

For any $[k, l]$ spatial shift vector,
$$T[f(x, y)] = g(x, y),$$
$$T[f(x - k, y - l)] = g(x - k, y - l)$$

eg.: weighting is not spatially invariant, but intensity inversion is.

Averaging on neighborhood is both linear, and spatially invariant: LSI.

2.1.3 Unit impulse function

2D unit impulse function (Delta function) on $\mathbb{Z}$:

$$\delta(x, y) = \begin{cases} 1 & x = 0, y = 0 \\ 0 & \text{otherwise} \end{cases}.$$

For any 2D function
$$f(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta(x - k, y - l) \cdot f(k, l).$$

2.1.4 Convolution

Impulse response is the output of an LSI transformation if the input was the Delta function:

$$T[\delta(x, y)] = h(x, y).$$

If T is an LSI transformation, then we can define convolution as follows:

$$g(x, y) = f(x, y) * h(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k, l) \cdot h(x - k, y - l).$$

Proof:

$$g(x,y) = T[f(x,y)] = T\left[\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)\sigma(x-k,y-l)\right] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)T[\sigma(x-k,y-l)] =$$

$$= \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} f(k,l)h(x-k,y-l).$$

2.2 Properties of convolution

- Commutative:
$$f * g = g * f.$$
- Associative:
$$f * (g * h) = f * g * h.$$
- Distributive:
$$f * (g + h) = f * g + f * h.$$
- Associative with scalar multiplication:
$$\alpha(f * g) = (\alpha f) * g.$$

2.3 Kernel size and boundary issues

In practice both the h kernel, and the image f have finite size. Typically the size of the kernel is much smaller than the image.

In general:

- Size of the input: $A \times B$,
- Size of the kernel: $C \times D$,
- Size of the output image: $(A + C - 1) \times (B + D - 1)$.

The border of the image is problematic. To convole an image with a $C \times D$ sized kernel, at the top and bottom of the image, we must add $\frac{C-1}{2}$ pixels, at the sides we have to add $\frac{D-1}{2}$ pixels. The question is, what values do we give to these pixels?

- Zero padding: fill the added pixels with 0s.
- Mirroring: mirror the existing pixels.
- Circular padding: Imagine many images next to each other, and cut out the picture, with the given size.
- Repeating border: repeat the pixel on the border.

2.4 Applications

Possible applications involve smoothing/noise reduction, edge detection, edge enhancement, etc... Depending on the task, the sum of the kernel can differ. E.g.: The sum of smoothing, and edge enhancement kernel matrices is 1, but edge detection kernel matrices summed are 0.

2.4.1 Example: Smoothing

Simple average:

$$g(x,y) = \frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r} f(x+i,y+j).$$

The 3×3 kernel would be:

$$h = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

2.5 Computational requirements

For k_s kernel size and P image size (in pixels) approximately k_sP operarions are needed. For larger kernel sizes, this may be slow.

2.6 Integral image

For blurring we can decrease the number of operations needed, if we utilize the integral image.

The integral image: $f->I_f$. The auxiliary representation:

$$I_f(x,y) = \sum_{i=1}^{x}\sum_{j=1}^{y} f(i,j).$$

We can calculate the integral image in P time, if we utilize the Auxiliary-auxiliary image:

$$t(x,y) = \sum_{j=1}^{y} f(x,j).$$

$$t(x,y) = f(x,1), x \in [1,w] \quad ... \quad t(x,y) = t(x,y-1) + f(x,y).$$

Using this we can calculate:

$$I_f(1,y) = t(1,y), y \in [1,h] \quad ... \quad I_f(x,y) = I_f(x-1,y) + t(x,y).$$

The sum of pixel values in an arbitrary sized sub-rectangle can be calculated by applying 3 additive operations using the integral image:

$$\sum_{i=a}^{c}\sum_{j=b}^{d} f(i,j) = I_f(c,d) - I_f(c,b-1) - I_f(a-1,d) + I_f(a,b).$$

The conclusion is, that wen we use the normal averaging, we have to do $(2r+1)^2$ addition and 1 division operations. If we use the integral image, we can reduce this to 3 additions, and 1 division operations, which is better in every case.

2.7 Blurring

2.7.1 Gaussian blur

The kernel weights are defined by a 2D Gaussian function. The two parameters of the function are the window size, and the standard deviation.

2.8 First and second ordered edge detection

The goal is to extract the object contours. We can define edge points, by looking at the brightness change. If the brightness is changed sharply, there is probably an edge.

2.8.1 Properties of a good edge filter

- (Near) zero output in homogeneous regions.
- Detects as many edges as possible.
- Does not create false edges.
- Good localization: Detected edges should be as close to the real ones, as possible.
- Isotropic: filter response independent of edge directions; all edges are detected, regardless of their direction.

2.8.2 Basic structures

- **Edge:** sharp intensity change.
- **Line:** thin long region with approximately uniform width and intensity level.
- **Blob:** closed region with homogeneous intensity.
- **Corner:** breaking or direction change of a contour or edge.

2.8.3 Parameters of an edge

- **Edge normal:** vector, perpendicular to the edge, pointing toward the steepest intensity change. (Alt.: edge direction: points toward the direction of the edge)
- **Position:** center point.
- **Strength:** intensity ratio compared to neighborhood.

2.8.4 Edge detection with first order derivative

For edge detection, since we are looking for intensity changes, we can use the partial derivatives:

$$\frac{\partial f}{\partial x} = \lim \frac{f(x+dx,y) - f(x,y)}{dx}, \quad \frac{\partial f}{\partial y} = \lim \frac{f(x,y+dy) - f(x,y)}{dy}.$$

Since the smallest step we can take is 1 pixel, we can approximate the partial derivatives:

$$\frac{\partial f}{\partial x} = f(x+1,y) - f(x,y), \quad \frac{\partial f}{\partial y} = f(x,y+1) - f(x,y).$$

For better localization, we could use a symmetric formula around (x, y):

$$\frac{\partial f}{\partial x} = f(x+1,y) - f(x-1,y), \quad \frac{\partial f}{\partial y} = f(x,y+1) - f(x,y-1).$$

The corresponding convolution kernel would be:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}.$$

Using these we can calculate the x and y directional Prewitt operators:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}. \qquad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

After these calculation, we can use the L_2 norm to get the "norm" of the derivative of our image:

$$||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}.$$

Other first-order methods:

Sobel operator:

$$\frac{\partial}{\partial x}: \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{\partial}{\partial y}: \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Roberts operator (emphesized edges with 45 degree slopes):

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

2.8.5 Second order edge detection

Instead of extreme values, we are looking for zero crossing.

$$\nabla^2 f = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \cdot \nabla f = \frac{\partial^2}{\partial x^2} f + frac\partial^2 \partial y^2 f.$$

Approximation for x direction:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \cong \frac{\frac{f(x+1,y)-f(x,y)}{v} - \frac{f(x,y)-f(x-1,y)}{v}}{v} = \frac{1}{v^2}\big(f(x+1,y) - 2f(x,y) + f(x-1,y)\big),$$

where v is the distance of neighboring pixel centers. So we can say, that

$$\frac{\partial^2 f(x,y)}{\partial x^2} \cong f(x+1,y) - 2f(x,y) + f(x-1,y), \qquad \frac{\partial^2 f(x,y)}{\partial y^2} \cong f(x,y+1) - 2f(x,y) + f(x,y-1).$$

This means, that the kernel for the Laplace operator is

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

To elliminate weak edges, a threshold can be used on the gradient image.

2.9 LoG

2.9.1 Introduction, and first order case

The result of the differentiation can be very noisy. For this purpose, we can do a Gaussian blur and do the edge detection after. But there is no need for 2 convolutions, since the convolution is associative, we can reduce the number of convolutions to one; convole the image with the derivative of the Gaussian operator:

$$\left(\frac{\partial}{\partial x} h\right) * f.$$

2.9.2 Second order case

To reduce the number of convolutions, we can use the LoG (Laplacian of Gauss) operator:

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

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

$$LoG = \nabla^2 h_\sigma(u,v).$$

2.10 Edge crispening

Kernel for edge enhancement with Laplace operator:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

This means, that we subtract the second order derivative from the original pixel, making it a much sharper change.

3 Canny edge detector and Hough transformation

3.1 Canny edge detector

John F. Canny has developed an edge detector in 1986 to meet the good edge detector requirements.

3.1.1 Main steps

1. Gradient map calculation
   - **Noise reduction:** The original image is convolved with a Gaussian kernel to reduce image noise.
   - **Gradient intensity and direction calculation**:
     - The horizontal and vertical derivative image is calculated.
     - At each pixel (i, j) calculate:
       * $d(i, j)$ gradient magnitude (how sharp is the edge, compared to the gradient magnitude):
       $$||\nabla f||_{i,j} = \sqrt{d_x^2(i,j) + d_y^2(i,j)}.$$
       * $n(i, j)$ edge normal (perp. to the direction):
       $$n(i,j) = \arctan\left(\frac{d_x(i,j)}{d_y(i,j)}\right).$$
2. Non-max suppression
   - The goal is to thin the edges. Perpendicular to the edges, we should only mark a single point as the edge point. We will chose this point to be the brightest one.
   - The steps:
     (a) Each edge is categorized into one of 4 main edge directions $(0°, 45°, 90°, 135°)$, based on the gradient direction
     (b) At every pixely it supresses the edge, if it's magnitude is not greater than the magnitude of the two neighbors in the gradient direction.
   - Result: The edges become thin.
3. Thresholding:
   - Naive solution: Thresholding G map with a threshold t. Problems: If t is too small, we obtain many false edges. If it's too large, the valid edges may disappear.
   - Improved solution: hysteresis thresholding:
     - Using 2 thresholds, $t_1 < t_2$:
     - If $G(x, y) > t_2$, it is certainly an edge point.
     - If $G(x, y) < t_1$, it is certainly not an edge point.
     - If $t_1 \leq G(x, y) \leq t_2$, we mark it as an edge point, if and only if it has a neighboring edge point, in the direction perpendicular to the edge normal.

3.1.2 Expected results

The expected result is an image, containing one pixel thick edges of the original image.

3.1.3 Parameters

The three parameters of the Canny edge detector are the Gaussian parameters, and the thresholding parameters.

3.2 Hough transform

3.2.1 Goals

The objective of the Hough transformation is to find the lines on a binary image, from fragments/points of the line.

3.2.2 Polar representation of lines

A line can be written in the following form:

$$y = mx + b$$

where m is the slope of the line, and b is the y-intercept. A line in the image corresponds to a point in the "m-b" space. We can invert this, and then for a fixed y_0, x_0 point in the image space we get a line in the m-b space:

$$m = -\frac{1}{x_0}b + \frac{y_0}{y_0}.$$

The basic idea: For the points that lie in the same line in the Euclidian space, their corresponding line in the m-b space will cross each other in one point. This point will be $m = m_0$ and $b = b_0$, the slope and intercept of the line in the image space.

Problem: vertical lines can't be described.

Solution: To describe all possible lines with two scalar parameters, we will use a polar representation of the line.

Each line is described by (r, θ) instead of (m, b), where r is the perpendicular distance from the line to the origin, and θ is the angle this perpendicular makes with the x axis.

We can use the polar equation of the line, the Hesse normal form:

$$0 = P \cdot n_0 - r,$$

where P is an arbitrary point of the line and $n_0 = (\cos\theta, \sin\theta)$ is the normal vector of the line. A point in the Euclidian space is a sinusoid in the Hough space, described by the following equation:

$$r(\theta) = x \cdot \cos\theta + y \cdot \sin\theta.$$

All the sinusoid curves of the points in one line in the Euclidian space, cross each other in one point in the Hough space.

3.2.3 Algorithm

1. For all r, θ: initialize $H[r, \theta] = 0$.
2. For each edge point $I[x, y]$ in the image:
   - For $\theta = 0$ to 180
     - $r = x \cdot \cos\theta + y \cdot \sin\theta$
     - $H[r, \theta] += 1$
3. Find the value(s) of (r, θ), where $H[r, \theta]$ is the maximum.
4. The detected line in the image space is given by $r = x \cdot \cos\theta + y \cdot \cos\theta$.

4 Image enhancement

4.1 Definition

Image enhancement is the manipulation or transformation of the image to improve the visual appearance or to help further automatic processing steps. There is no general theory behind it, the result is highly application dependent and subjective. Enhancement is closely related to image recovery.

4.2 Main types

There are two main categories:

1. Spatial domain methods
2. Frequency domain methods

In spatial domain, we are directly manipulating pixel values through:

- Point-wise intensity transformation
- Histogram transformations
- Spatial filtering (LSI, Non-Linear)
- etc...

4.3 Histogram

The histogram $h(k)$ of the image is the number of pixels on the image with k intensity. The histogram normalized with the total number of pixels gives us the probability density function of the intensity values.

4.4 Point-wise Intensity Transformation

Point wise transformations are operating directly on pixel values, independently of the values of its neighboring pixels. We can describe the transformation as follows:

Let x and y be two greyscale images, and let T be a point-wise image enhancement transformation, that transforms x to y:

$$y(n_1, n_2) = T[x(n_1, n_2)].$$

4.4.1 Inverse transformation

$$y(n_1, n_2) = 255 - x(n_1, n_2).$$

4.4.2 Log transformation

$$y(n_1, n_2) = c \cdot \log\big(x(n_1, n_2) + 1\big).$$

This transformation expands low and compresses high pixel value range.

4.4.3 Power-law transformation

$$y(n_1, n_2) = c \cdot x^{\gamma}(n_1, n_2).$$

Commonly referred to as gamma transformation, it was originally developed to compesate the input-output characteristics of CTR displays.

4.5 Dynamic Range Expansion

Piecewise linear expansion/compression of predefined intensity ranges. Example: extracting the intensity values from the $[g_1, g_2]$ interval to a wider $[h_1, h_2]$ domain.In the selected interval, the contrast increases, everywhere else it decreases.

4.6 Histogram Stretching

With the following transformation we can stretch the intensity values so they use the whole available range:

$$y(n_1, n_2) = \frac{255}{x_{max} - x_{min}} \cdot \big(x(n_1, n_2) - x_{min}\big)$$

There are more ways to do this apart from the linear, there is the quadratic, and square root stretching.

4.7 Histogram Equalization

The goal is to increase the contrast, by distributing the occurrences of the intensity values evenly through the entire dynamic range:

$$\sum_{i=0}^{t_j} h[i] \approx P \cdot \frac{j}{c}, \quad j \in \{1..c\},$$

where c c is the number of different gray levels in the output image, and P is the total number of pixels.

Note: Adaptive histogram equalization is where we apply the histogram equalization independently to parts of the image.

4.8 Spatial Filtering

4.8.1 Smoothing

Reduce the noise that may corrupt the image. Some noise types:

- Impulse noise aka salt and pepper noise,
- Additive Gaussian noise

4.8.2 Spatially adaptive noise smoothing

The smoothing takes into account the local characteristics of the image:

$$x(n_1, n_2) = \left(1 - \frac{\sigma_n^2}{\sigma_l^2}\right) \cdot x(n_1, n_2) + \frac{\sigma_n^2}{\sigma_l^2} \cdot \overline{x}(n_1, n_2),$$

where σ_n^2 is the variance of the noise and

$$\overline{x}(n_1, n_2) = \frac{1}{|N|} \sum_{(n_1, n_2) \in N} x(n_1, n_2),$$

$$\sigma_l^2 = \sum_{(n_1, n_2) \in N} \big(x(n_1, n_2) - \overline{x}(n_1, n_2)\big)^2.$$

4.8.3 Rank filter

1. Consider the actual pixel and neighborhood,
2. Sort the observed pixel values according to the gray level,
3. Take the k-th value from this row as the new pixel value.

4.8.4 Median filter

k is the middle pixel in the row:

$$k = \frac{(2W+1)^2 - 1}{2},$$

if W is the half side size of the neighborhood.

This method is very effective against salt and pepper noise, but not so effective against Gaussian noise.

4.8.5 Order statistic filtering

Based on the sorted pixel intensity levels in the analyzed neighborhood.

- Mid-point filtering:
  - works well on Gaussian or uniform noise:

$$y(n_1, n_2) = \frac{1}{2}\big(\max_{(m_1,m_2)\in N}\{x(m_1, m_2) + \min_{(m_1,m_2)\in N}\{x(m_1, m_2)\}\big)$$

- Alpha-trimmed mean filter:

$$y(n_1, n_2) = \frac{1}{|n| - \alpha} \sum_{(m_1,m_2)\in N_r} x(m_1, m_2),$$

  where N_r is a reduced neighborhood, not containing the lowest and highest elements on N.
  - If $\alpha = 0$, we get back the arithmetic mean.
  - If $\alpha = |N| - 1$, we get back the median filter.

4.9 Wallis Operator

The Wallis operator can help to adjust local contrast:

$$y(n_1, n_2) = \big(x(n_1, n_2) - \overline{x}(n_1, n_2)\big)\frac{A_{max}\sigma_d}{A_{max}\sigma_l(n_1, n_2) + \sigma_d} + \big(p\overline{x}_d + (1 - p)\overline{x}(n_1, n_2)\big),$$

where σ_l is the local contrast, $\overline{x}$ is the local average, σ_d is the desired local contrast, $\overline{x}_d$ is the desired mean value of all pixels, p is a weighting factor of the mean compensation, and A_{max} is maximizing the local contrast modification.

4.10 Anisortopic Diffusion

The anisotropic diffusion is a technique aiming at reducing image noise without blurring significant parts of the image content. This is a non-linear and space-variant transformation.

The main idea is that the effect of blurring in each direction is inversely proportional to the gradient value in that direction. This allows diffusion along the edges or in edge-free territories, but penalizes diffusion orthogonal to the edge direction.

This process is iterative.

4.11 TV Regularization

The goal of Total Variation based noise removal is to minimize the total variation of the image while keep the result as close to the original input image as possible.

The TV of the output image y is defined as the integral of the absolute gradient of the signal:

$$V(y) = \sum_{n_1,n_2} \sqrt{|y(n_1 + 1, n_2) - y_(n_1, n_2)|^2 + |y(n_1, n_2 + 1) - y(n_1, n_2)|^2}.$$

On the other hand, we also measure the difference between the original image x and the output image y by L_2 norm E:

$$E(x, y) = \sum_{n_1,n_2} \big(x(n_1, n_2) - y(n_1, n_2)\big)^2.$$

The goal function for Total Variation based regularization:

$$\hat{y} = \arg\min_y \big[E(x, y) + \lambda V(y)\big],$$

where λ is the regularization parameter.

5 Image analysis in the Fourier domain

The Fourier Transform changes between the representation in the time domain and in the frequency domain. The information is the same in both domains, only the representation is different. It is a reversible transform.

5.1 Definition of 2D Fourier transform

Forward transform: map an image $x(n_1, n_2)$ of size $N_1 \times N_2$ from the spatial domain into the $X(\omega_1, \omega_2)$ frequency domain:

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}.$$

Problems:

- Even if the image is real the spectrum is complex due to the complex exponential factors.
- ω_1, ω_2 are continous variables
- No computable representation

Solution: Discrete Fourier Transform, or DFT.

5.2 Discrete FT and IDFT

We sample one period of the Fourier transform in evenly spaced frequencies:

$$X(k_1, k_2) = X(\omega_1, \omega_2)\Big|_{\omega_1 = \frac{2\pi}{N1} k_1, \omega_2 = \frac{2\pi}{N_2} k_2},$$

where $k_1 = 0...N_1 - 1$, $k_2 = 0...N_2 - 1$.

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N1} k_1 n_1} e^{-j\frac{2\pi}{N_2} k_2 n_2}.$$

Inverse Fourier transform: maps from the discrete frequency domain back to the discrete spatial domain:

$$x(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{j\frac{2\pi}{N1} k_1 n_1} e^{j\frac{2\pi}{N_2} k_2 n_2}$$

5.3 Interpretation of Fourier coefficients

We can interpret the real part of an $X(k_1, k_2)$ Fourier coefficient as the amplitude of a cosine wave, while the imaginary part of the amplitude of a sinusoid wave.

The wavelength and orientation of the waves are encoded in the (k_1, k_2) position coordinates of the coefficients in the 2D Fourier map.

Spatial frequency:

$$f = \sqrt{\left(\frac{k_1}{N_1}\right)^2 + \left(\frac{k_2}{N_2}\right)^2}.$$

Wavelength:

$$\lambda = \frac{1}{f}.$$

Orientation:

$$\alpha = \arctan\left(\frac{k_2/N_2}{k_1/N_1}\right).$$

5.4 Centered DFT

Instead of positioning the Fourier map onto one fourth of the coordinate system, we can center it. This way we will be able to get the frequency, as a distance from the middle.

5.5 Image analysis in the Fourier domain (LPF, HPF)

In the transformed map, directions of strong lines are perpendicular to the major contours in the image.

There are many different filters, that can be applied to the Fourier map, but maybe the two most important ones are the Low-Pass Filter, which removes all low frequency waves, thus filtering out the large spatial frequencies, and blurring the image, and the High-Pass filter, which removes the high frequency waves, thus filtering out the low spatial frequencies, making a "kind-of" edge detector.

5.6 Fourier coeffcients for curves

1. Compley numbers from 2D curve points: $z_k = x_k + j \cdot y_k$
2. 1D DFT transform calculated for the complete closed curve:

$$C_n = \sum_{k=0}^{K-1} z_k \cdot e^{j\frac{2\pi}{K}n \cdot k}$$

3. Setting high frequency C_n coefficients to zero, then recovering of approximate contour points by inverse transform.

The more Fourier coefficients, the better the approximation is.

5.7 Homomorphic filtering

Motivation: image with large dynamic range, e.g. natural scene on brightly sunny day, recorded on a medium with small dynamic range results in image contrast significantly reduced especially in dark and bright regions. The goal is to reduce the dynamic range, and increase the contrast.

It simultaneously normalizes the brightness across an image and increases contrast.

Assumes the following image model: the image is formed by recording the light reflected from the objects illuminated by a light source:

$$x(n_1, n_2) = i(n_1, n_2) \cdot r(n_1, n_2).$$

Illumination is slowly varying, whereas reflectance is rapidly varying. We want to reduce the illumination, and increase the reflectance.

To seperate the two components we first use a log transformation:

$$\log\big(x(n_1, n_2)\big) = \log\big(i(n_1, n_2)\big) + \log\big(r(n_1, n_2)\big).$$

Since we assume that the illumination component varies slowly and the reflectance varies rapidly, we can get the two component by using (Fourier-based) low and high pass filters:

$$\log\big(i(n_1, n_2)\big) = LPF\big[\log\big(x(n_1, n_2)\big)\big],$$

$$\log\big(r(n_1, n_2)\big) = HPF\big[\log\big(x(n_1, n_2)\big)\big].$$

Weighting the two components:

$$\log\big(y(n_1, n_2)\big) = \gamma_1 \log\big(i(n_1, n_2)\big) + \gamma_2 \log\big(r(n_1, n_2)\big), \qquad \gamma_1 < 1, \gamma_2 > 1.$$

After this we transform back to the original range using the exponential transform.

6 Texture analysis 1

6.1 Definition of textures

Textures demonstrate the difference between an artificial world of objects whose surfaces are only characterized by their color and reflectivity properties to that of real world imagery. Texture is a repeating microstructure.

Some local "order" is repeated over a region which is large in comparison to the order's size. The order consists in the nonrandom arrangement of elementary parts. The parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region.

6.2 4 primary issues with examples

There are various primary issues in texture analysis:

- Texture classification
  - In texture classification, the problem is identifying the given textured region from a given set of texture classes.
  - The texture analysis algorithms extract distinguishing feature from each region to facilitate classification of such patterns.
- Texture segmentation
  - Unlike texture classification, texture segmentation is concerned with automatically determining the boundaries between various textured regions in an image.
  - Both reign-based methods and boundary-based methods have been attempted to segment texture images.
- Shape recovery from texture
  - Image plane variation in the texture properties, such as density, size and orientation of texture primitives, are the cues exploited by shape –from-texture algorithms.
  - Quantifying the changes in the shape of texture elements is also useful to determine surface orientation.
- Texture modelling
  - Specify a model that clearly identifies the given pattern sample.
  - Used to synthesize textures.

6.3 Techniques for texture extraction (with examples)

There are various techniques for texture extraction. Texture feature extraction algorithms can be grouped as follows:

- Statistical
- Geometrical
- Model based
- Signal Processing

6.3.1 Statical methods

1. Local features:
   - Gray level of certain pixels,
   - Average of gray levels in window,
   - Median,
   - Standard deviation of gray levels,

- Difference of maximum and minimum gray levels,
- Difference between average gray level in small and large windows,
- Kirsch feature,
- Combine features

2. Galloway: run length matrix
3. Haralick: Co-occurance matrix

6.3.2 Geometrical methods

Steps:

1. Threshold images into binary images of n grey levels.
2. Calculate statistical features of connected areas.

6.3.3 Model based methods

These involve building mathematical models, to describe textures:

- Markov random fields
- Fractals

6.3.4 Signal processing

These methods involve transforming original images using filters and calculating the energy of the transformed images.

- Law's masks,
- Laines-Daubechies wavelets,
- Fourier transform,
- Gabor filters.

6.4 GLCM and applications

The Gray Level Co-occurrence Matrix, also referred to as co-occurrence distribution, is the most classical second-order statistical method for texture analysis.

Gray Level Co-occurrence Matrix (GLCM) filters operate by computing, for each filter window position, how often specific pairs of image cell values occur in neighboring cell positions (such as one cell to the right).

The results are tabulated in a co-occurrence matrix, and specific statistical measures are computed from this matrix to produce the filtered value for the target cell.

It is a matrix of frequencies at which two pixels, separated by a certain vector, occur in the image. Co-occurrence matrix is defined as,

$$p_{i,j}(\Delta x, \Delta y) = W \cdot Q(i, j|\Delta x, \Delta y),$$

where

$$W = \frac{1}{(M - \Delta x)(N - \Delta y)}, \quad Q(i, j|\Delta x, \Delta y) = \sum_{n=1}^{n-\Delta y} \sum_{m=1}^{M-\Delta x} A,$$

$$A = \begin{cases} 1 & f(m, n) = i, f(m + \Delta x, n + \Delta y) = j \\ 0 & otherwise \end{cases}$$

It has a size of $L \times L$, where L is the number of gray values.

Note: Polar representation of the offset: two parameters, d, θ instead of $\Delta x, \Delta y$, where d is the relative distance between the pixel pair, and θ is the relative orientation.

6.4.1 Example

Image matrix:

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 \end{bmatrix}.$$

Pixel values: 0, 1, 2, 3, this means that $L = 4$ thus the size of the matrix will be 4×4. Let $d = 1$, and $\theta = 0°$ (horizontal):

$$CM = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

6.4.2 Energy

Also called Uniformity or Angular second moment, it measures the textural uniformity that is pixel pair repetitions.

$$Energy = \sum_i \sum_j p_{ij}^2.$$

6.4.3 Entropy

Measures the disorder, or complexity of an image. The entropy is large when the image is not texturally uniform. Entropy is strongly, but inversely correlated to energy.

$$Entropy = -\sum_i \sum_j p_{ij} \log_2 p_{ij}.$$

6.4.4 Contrast

Measures the spatial frequency of an image and is difference moment of GLCM. It is the difference between the highest and the lowest values of a contiguous set of pixels. It measures the amount of local variations present in the image.

$$Contrast = \sum_i \sum_j (i-j)^2 p_{ij}$$

6.4.5 Homogenity

Also called as Inverse Difference Moment, it measures image homogenity as it assumes larger values for smaller gray tone differences in pair elements.

$$Homogenity = \sum_i \sum_j \frac{1}{1+(i-j)^2} p_{ij}.$$

6.4.6 Variance

This statistic is a measure of heterogeneity and is strongly correlated to first order statistical variable such as standard deviation. Variance increases when the gray level values differ from their mean.

$$Variance = \sum_i \sum_j (i-\mu)^2 p_{ij},$$

where μ is the mean of p_{ij}.

6.5 Autocorrelation and its application

Autocorrelation: Comparing the dot product (energy) of the non-shifted image with a shifted image.

$$R_{II}(\Delta x, \Delta y = \frac{\sum_{x=0}^{n}\sum_{y=0}^{N} I(x,y)I(x+\Delta x, y+\Delta y)}{\sum_{x=0}^{n}\sum_{y=0}^{N} I^2(x,y)})$$

6.5.1 Features

- Autocorrelation function can detect repetitive patterns of texels.
- Also defines the fineness/coarseness of the texture.

6.5.2 Autocorrelation calculation in the Fourier domain

Wiener-Khinchin Theorem: The autocorralation of the image I, is the inverse Fourier transform of the power spectrum of I:

$$R_{II}(\Delta x, \Delta y) = IDFT\big[|X(\omega_1, \omega_2)|^2\big].$$

7 Texture analysis 2

7.1 Definition of textures

Textures demonstrate the difference between an artificial world of objects whose surfaces are only characterized by their color and reflectivity properties to that of real world imagery. Texture is a repeating microstructure.

Some local "order" is repeated over a region which is large in comparison to the order's size. The order consists in the nonrandom arrangement of elementary parts. The parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region.

7.2 4 primary issues with examples

There are various primary issues in texture analysis:

- Texture classification
  - In texture classification, the problem is identifying the given textured region from a given set of texture classes.
  - The texture analysis algorithms extract distinguishing feature from each region to facilitate classification of such patterns.
- Texture segmentation
  - Unlike texture classification, texture segmentation is concerned with automatically determining the boundaries between various textured regions in an image.
  - Both reign-based methods and boundary-based methods have been attempted to segment texture images.
- Shape recovery from texture
  - Image plane variation in the texture properties, such as density, size and orientation of texture primitives, are the cues exploited by shape –from-texture algorithms.
  - Quantifying the changes in the shape of texture elements is also useful to determine surface orientation.
- Texture modelling
  - Specify a model that clearly identifies the given pattern sample.
  - Used to synthesize textures.

7.3 Laws filters

The goal is to enhance micro-structure of the texture. Main elements:

- Averaging
- Edges
- Points

$$L_3 = \frac{1}{6}\begin{bmatrix}1\\2\\1\end{bmatrix}, \quad E_3 = \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix}, \quad S_3 = \frac{1}{2}\begin{bmatrix}1\\-2\\1\end{bmatrix},$$

where L is a level detection filter, E is an edge detection filter, and S is a spot detection filter.

To get Laws filters, we multiply these filters together, resulting in 9 filters. With these matrices, we will train with texture models. Each training texture will be convoled with every Laws filter. Now every texture is represented with 9 matrices. We take these matrices, and average the squares:

$$M = \frac{1}{hw}\sum_{x,y}(T * H_n)^2(x,y),$$

where T is the image of the training texture, $H_n, n = 1..9$ is a Laws filter. This way every trained texture will be represented with 9 scalars.

In the recognition phase, we convole the input image with the 9 filters, and preform a squared blurring. To get the class map:

$$ClassMap(x, y) = \arg \min_{j=1...n} \sum_{i=1...9} \left|P_i(x, y) - M_i^j\right|,$$

where n is the number of different textures, P_i is the squared blur of the convoled input image.

7.4 Gabor filtering

We get Gabor filters, by multiplying sinusoid waves with the Gaussian filter.

7.4.1 Gabor wavelets

$$\Psi_c(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi u_0 x)$$

7.4.2 Applications

Gabor filters can selectively highlight specific image elements according to their appropriately set frequency, orientation and phase parameters. Invariant for additive changes of illumination (in case of asymmetric sinusoid functions)

7.5 Texture synthesis

The goal is, that given a small sample, generate larger realistic versions of the texture.

A good solution may be to try to find a window, in the image, that matches the neighborhood, and assign that value to the pixel. This may not work though if there are no exact matches, so we find the best matches using SSD error and randomly choosing between them, preferring the better matches with higher probability.

For block-based texture syntesis, since the neighbor pixels are highly correlated, we can create a much more optimal algorithm, by synthesizing blocks of pixels at once.

This may be correct for some cases, in others, it will have a "copy-paste" effect. To reduce this, we overlap these blocks, and calculate a minimal error boundary cut, where we take the square of the difference of the overlapping boundaries, and select a path where the error is minimal.

7.6 Texture transfer

Each patch satisfies a desired correspondence map C as well as satisfying the texture synthesis requirements. C is a spatial map of some corresponding quantity over both the texture source image and a controlling target image. E.g.: image intensity, blurred image intensity, local image orientation angles, etc...

Take the texture from one image and "paint" it onto another object. Same algorithm as before with additional term:

- do texture synthesis on the texture image
- add term to match intensity of the other image.

8 Image segmentation: Intro, K-means and Morphology

8.1 Definition of image segmentation

Gestalt definition: a configuration or pattern of elements so unified as a whole that it cannot be described merely as a sum of its parts.

8.2 Types of image segmentation

The segmentation can be knowledge-driven (top-down) or data-driven (bottom-up). Knowledge driven segmentation methods builds prior knowledge into the segmentation algorithm, and group tokens that likely belong to the same object:

- Hard to implement
- Cannot stand alone, needs cues from bottom-up segmentation.

Data-driven methods builds on the raw pixel data, and group tokens with similar features:

- Easier to implement.
- Often fail on real life images.

There is the so-called semantic gap between the two approach. The complex, high level definitions of top-down methods are hard to embed efficiently into low level algorithms.

- Intensity level based segmentation
  - Thresholding
  - Otsu's method
- Region-based segmentation
  - Region growing
  - Region splitting and merging
- Clustering in feature space

8.3 Intensity level based segmentation

8.3.1 Thresholding

Assumtion is that the image parts can be seperated based on their intensity level.

$$s(n_1, n_2) = \begin{cases} \text{object} & x(n_1, n_2) < T \\ \text{background} & x(n_1, n_2) \geq T \end{cases},$$

where $s(n_1, n_2)$ is the cluster of the (n_1, n_2) pixel of the x image, and T is a threshold.

8.3.2 Otsu's method

Automatically determines the optimal global threshold by minimizing the intra-class variance. The intra class variance is defined, as follows:

$$\sigma_w^2(k) = \omega_1(k)\sigma_1^2(k) + \omega_2(k)\sigma_2^2(k),$$

where ω_i and σ_i are the probability and variance of the two classes seperated by the threshold k.

Otsu showed, that minimizing the intra-class variance is the same as maximising the inter-class variance:

$$\sigma_b^2(k) = \sigma^2 - \sigma_w^2(k) = \omega_1(k)\omega_2(k)\big(\mu_1(k) - \mu_2(k)\big)^2,$$

where μ_i are the means of the two classes seperated by the threshold k.

To calculate ω_i and μ_i, the normalized histogram of the image is used:

$$\omega_1(k) = \sum_{i=0}^{k} p_i, \qquad \mu_1(k) = \frac{\sum_{i=0}^{k} ip_i}{\omega_1},$$

$$\omega_2(k) = \sum_{i=k+1}^{L-1} p_i, \qquad \mu_2(k) = \frac{\sum_{i=k+1}^{L-1} ip_i}{\omega_2},$$

where p_i is the i-th entry in the normalized histogram of the image. The otsu threshold is the value that maximises the inter-class variance.

8.4 Region-based segmentation

Let R be the entire image region, and $R_1, ..., R_n$ are subregions. We want to find a segmentation, that is

- Complete: $\bigcup_{i=1}^{n} R_i = R$,
- Points in the region $R_i (i = 1...n)$ are connected,
- The regions are disjoint: $R_i \cap R_j = \emptyset$, $\forall i \neq j$,
- All the pixels in a region has common properties, that they don't share with pixels from other regions.

8.4.1 Region growing

The method is initialized with a set of seed points as regions. We start growing the regions by adding neighboring pixels to the region if they has similar predefined properties as the seed points. The seeds can be selected based on prior information, or evenly, or random.

The similarity criteria is usually depending on the segmentation result we want. (Commonly used properties are the intensity level, color, texture, motion,...)

This is simple, and works well on images with clear edges, prior knowledge can be easily utilized, robust noise, but it's time consuming.

8.4.2 Region splitting and merging

Let R represent the entire image region and P be a predicate. The splitting and merging steps are alternating.

- We split the region R_i into 4 subregions if $P(R_i) = false$.
- We merge two neighboring regions R_i and R_j if $P(R_i \cup R_j) = true$.

The minimum region size must be selected.

8.5 K-means algorithm

K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications. Each cluster is represented by the centre of the cluster and the algorithm converges to stable centroids of clusters. Each sample will belong to the cluster with the nearest mean.

The objective is to minimize the within-cluster sum of squares:

$$\arg\min_{s} \sum_{i=1}^{K} \sum_{x \in S_i} d^2(x, \mu_i), \qquad d^2(x, \mu_i) = ||x - \mu_i||^2 = \sum_{n=1}^{N} (x_n - \mu_{in})^2,$$

where $x \in \mathbb{R}^N$ are the data samples, μ_i is the mean of the points in the cluster $S_i (i = 1...K)$.

8.5.1 Algorithm

Given the cluster number K, the K-means algorithm is carried out in three steps after initialization:

1. **Initialization:** set K cluster seed points (randomly).
2. **Assignment step:** Assign each object to the cluster of the nearest seed point measured with a specific distance metric.
3. **Update step:** Compute new seed points as the centroinds of the clusters of the current partition:
$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^t} x_j.$$
4. **Go back** to the assignment step. Stop, when there are no chacnges compared to the previous state.

Efficient in computation: $O(tKn)$, where n is the number of objects, K is the number of clusters, and t is the number of iterations.

8.5.2 Issues

- Number of clusters has to be known in advance.
- Sensitive to initial seed points
- Not suitable for discovering non-convex shapes.

8.6 Morphology

Once segmentation is complete, morphological operations can be used to remove imperfections in the segmented image and provide information on the form and structure.

Morphological image processing (or morphology) describes a range of image processing techniques that deal with the shape (or morphology) of features in an image. They are typically applied to remove imperfections introduced during segmentation, and so typically operate on bi-level images.

The basic idea in binary morphology is to probe an image with a structuring element (a simple, pre-defined shape), drawing conclusions on how this shape fits or misses the shapes in the image.

8.6.1 Fit/Hit

Fit: All on pixels in the structuring element cover on pixels in the image.

Hit: Any on pixel in the structuring element covers an on pixel in the image.

All morphological processing operations are based on these simple ideas. The structuring element is moved across every pixel in the original image to give a pixel in a new processed image. The value of this new pixel depends on the operation performed.

8.6.2 Erosion

Erosion of image f by structurinf element s is given by $f \ominus s$. The structuring element s is positioned with its origin at (x, y), and the new pixel value is determided using the following rule:

$$g(x,y) = \begin{cases} 1 & s \text{ fits } f \\ 0 & \text{otherwise} \end{cases}.$$

This is used to plit apart joinded objects, and to remove extrusions.

8.6.3 Dilation

Dilation of image f by structuring element s is given by $f \oplus s$. The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1 & s \text{ hits } f \\ 0 & \text{otherwise} \end{cases}.$$

Dilation is used to repair breaks, and repair intrusions.

8.6.4 Compound operations

More interesting morphological operations can be performed by performing combinations of erosions and dilations The most widely used of these compound operations are:

- Opening
  - The opening of image f by structuring element s denoted $f \circ s$ is simply an erosion followed by a dilation:
$$f \circ s = (f \ominus s) \oplus s.$$
- Closing
  - The closing of image f by structuring element s denoted $f \bullet s$ is simply a dilation followed by an erosion:
$$f \bullet s = (f \oplus s) \ominus s.$$

8.6.5 Boundary extraction

Extracting the boundary (or outline) of an object is often extremely useful. The boundary canbe given simply as
$$\beta(A) = A - (A \ominus B).$$

8.6.6 Region filling

Given a pixel inside a boundary, region filling attempts to fill that boundary with object pixels. The key equation for region filling is:
$$X_k = (X_{k-1} \oplus B) \cap A^C,$$
where

- A is the original image,
- X_0 is simply the starting point inside a boundary,
- B is a simple structuring element,
- A^C is the complement of A.

This equation is repeated until $X_k = X_{k-1}$

9 Markov Random Fields

9.1 Main principle

The main principle of Markov Random Fields in image processing is to map the image to a graph, where nodes are assigned to different pixels, and the edges connect pixels which are in interaction.

In this case we can start thinking of segmentation as pixel labeling, each pixel gets a class label from a task-dependent label set $\wedge$.

Instead of finding a direct algorithm to find the optimal labeling, we construct a probability function which assigns a likelihood value to each possible global segmentation, then an optimization process to find the labeling with the highest confidence.

This probability function is dependent on

- Local feature vectors at each pixel
- label consistency constraints between neighboring pixels.

9.2 Labeling and MAP

Each pixel s in the image has a feature vector $\overline{f}_s$. For the whole image, we have $f = \{\overline{f}_s : s \in S\}$, a global observation. Each pixel is assigned a label $\omega_s \in \wedge$. For the whole image, we have $\omega = \{\omega_s : s \in S\}$, a global labeling.

Ω is a set of all possible ω global labelings. For an $N \times M$ image, there are $|\Omega| = |\wedge|^{NM}$ possible goal labelings. The question is: which one is the correct one?

We define a probability measure on the set of all possible global labeling and select the most likely one. $P(\omega|f)$ measures the probability of a global labeling ω, given the observed features f. Our goal is to find an optimal labeling $\hat{\omega}$, which maximizes $P(\omega|f)$. This is called the Maximum a Posteriori (MAP). Estimate:

$$\hat{\omega} = \arg\max_{\omega \in \Omega} P(\omega|f).$$

9.3 Bayesian Framework

The Bayes theorem says

$$P(\omega|f) = \frac{P(f|\omega)P(\omega)}{P(f)} \propto P(f|\omega)P(\omega),$$

since $P(f)$ is constant. We need to define $P(f|\omega)$ and $P(\omega)$ in our model.

9.4 Definition - Neighbors

For each pixel, we can define some surrounding pixels as it's neighbors. Example: first-, and second-order neighbors.

9.5 Definition - MRF

The labeling field X can be modelled as a Markov Random Field if

1. $\forall \omega \in \Omega : P(X = \omega) > 0$
2. $\forall s \in S, \forall \omega \in \Omega : P(\omega_s|\omega_r, r \neq s) = P(\omega_s|\omega_r, r \in N_s)$ where N_s denotes the neighbors of s.

9.6 Hammersley-Clifford Theorem

The Hammersly-Clifford theorem states, that a random field is a MRF if and only if $P(\omega)$ follows a Gibbs distribution.

$$P(\omega) = \frac{1}{Z} e^{-U(\omega)} = \frac{1}{Z} e^{-\sum_{c \in C} V_C(\omega)},$$

where $Z = \sum_{\omega \in \Omega} e^{-U(\omega)}$ is a normalization constant. The practical consequences:

- Probability functions of MRFs can be factorized into small terms $V_c(\omega)$, called clique potentials, which can be locally calculated on the graph.
- Because of this property it is possible to design the $P(w)$ probability function in a modular way, and enables using efficient iterative optimization techniques.

9.7 Clique potentials

The H-C theorem provides us an easy way of defining MRF models via clique potentials. A subset $C \subseteq S$ is called a clique if every pair of pixels in this subset are neighbors.

A clique containing n pixels is called n^{th} order clique, denoted by C_n. The set of cliques is denoted by

$$C = C_1 \cup C_2 \cup ... \cup C_K.$$

For each clique c in the image, we can assign a value $V_C(\omega)$ which is called clique potential of c, where ω is the configuration of the labelig field.

The sum of potentials of all cliques gives us the energy $U(\omega)$ of the configuration ω.

$$U(\omega) = \sum_{c \in C} V_C(\omega) = \sum_{i \in C_1} V_{C_1}(\omega_i) + \sum_{(i,j) \in C_2} V_{C_2}(\omega_i, \omega_j) + ...$$

9.8 MRF segmentation model

Pixel labels are represented by (for example) Gaussian distributions:

$$P(f_s|\omega_s) = \frac{1}{\sqrt{2\pi}\sigma_{\omega_s}} e^{-\frac{(f_s - \mu_{\omega_s})^2}{2\sigma^2_{\omega_s}}}.$$

Clique potentials:

- Singleton: proportional to the likelihood of features given ω : $\log P(f|\omega)$.
- Doubleton: favors similar labels at neighboring pixels - smoothness prior

$$V_{C_2}(i,j) = \beta\delta(\omega_i, \omega_j) = \begin{cases} -\beta & \omega_i = \omega_j \\ \beta & \omega_i \neq \omega_j \end{cases}.$$

As β increases, regions become more homogenous.

9.9 Model parameters

- Doubleton potential β
  - Less dependent on the input
  - Can be fixed a priori
- number of labels $|\wedge|$
  - Problem dependent
  - Usually given by the user or inferred from some higher level knowledge.
- Each label $\lambda \in \wedge$ is represented by a Gaussian distribution $N(\mu_\lambda, \sigma_\lambda)$ estimated from the input image.
- The class statistics (mean and variance) can be estimated via the empirical mean and variance.

9.10 Energy function

Now we can define the energy function of our MRF model:

$$U(\omega) = \sum_s \left(\ln\left(\sqrt{2\pi}\sigma_{\omega_s}\right) + \frac{(f_s - \mu_{\omega_s})^2}{2\sigma^2_{\omega_s}} \right) + \sum_{s,r} \beta\delta(\omega_s, \omega_r).$$

Hence

$$\hat{\omega} = \arg\max_{\omega \in \Omega} P(\omega|f) = \arg\min_{\omega \in \Omega} U(\omega).$$

9.11 Optimization

The problem is now reduced to the minimization of a non-convex energy function.

9.11.1 Iterated Conditional Mode

This is a gradient descent approach.

1. Start at a "good" initial configuration ω_0 and set $k = 0$.
2. For each configuration which differs at most in one element from the current configuration (they are denoted by N_{ω_k}) ω_k, compute the energy $U(\eta), \eta \in N_{\omega_k}$.
3. From the configurations N_{ω_k} select the one wich has the minimal energy:
$$\omega_{k+1} = \arg \min_{\eta \in N_{\omega_k}} U(\eta.)$$
4. Go to step 2, with $k = k + 1$, untill convergence is obtained.

Con: Gets stuck in local minima.

9.12 Modified Metropolis Dynamics

1. Set $k = 0$ and initialize ω randomly. Choose a sufficiently high initial temperature $T = T_0$.
2. Construct a trial perturbation η from the current configuration ω such that η differs only in one element from ω.
3. (Metropolis criteria) Compute $\Delta U = U(\eta) - U(\omega)$ and accept η if $\Delta U < 0$. Else accept with probability $e^{\frac{-\Delta U}{T}}$:
$$\omega = \begin{cases} \eta & \Delta U \leq 0 \\ \eta & \Delta U > 0 \text{ and } \xi < e^{-\frac{\Delta U}{T}} \\ \omega & \text{otherwise} \end{cases},$$
where ξ is a uniform random number in $[0, 1[$.
4. Decrease the temperature $T = T_{k+1}$ and goto step 2 with $k = k + 1$ until the system is frozen.

10 Mean shift

10.1 Mean shift intuitive description

Imagine a distribution of identical billiard balls. The objective is to find the densest region. We start by placing a somewhat random starting point, with a predefined region of interest. We then proceed to to calculate the center of mass within the region of interest, and move the point, aswell as the region of interest to the center of mass. The vector of this movement is called the mean shift vector.

We do this, untill there is only minimal, or no change at all.

10.2 Mean shift vector

We have the data points, and we have to approximate the center of mass, then determine the shift vector from the initial mean:

$$m_h(y_0) = \left[\frac{\sum_{i=1}^{n_x} w_i(y_0) \cdot x_i}{\sum_{i=1}^{n_x} w_i(y_0)} \right] - y_0,$$

where n_x is the number of points in the kernel, y_0 is the initial mean location, x_i are the data points, h is the kernel radius, and w_i are the weights determined by different kernels (uniform, Gaussian, Epanechnikov).

10.3 Parametric and nonparametric distributions

10.3.1 Parametric distributions

Parametric distributions have a closed formula for the probability density function (PDF) with a few parameters. If we estimate the PDF from the samples, we can then forget the samples and use the PDF directly for probability calculation.

10.3.2 Nonparametric distributions

We do not have the closed formula for the PDF. Instead, we need to store the samples, and use them directly to model the PDF.

10.4 Kernel density estimation

Our assumption is that the data points are sampled from an underlying PDF:

$$PDF(x) = \sum_{i=1}^{n} c_i e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}},$$

where μ_i is the ith sample. Each sample point contributes to the PDF with an additive term (here: Gaussian).

The role of kernels is to determine the weights of nearby points in the density calculation.

$$P(x) = \frac{1}{n} \sum_{i=1}^{n} K(x - x_i),$$

where K is the kernel. Some famous kernels:

- Uniform kernel:
$$K_U(x) = \begin{cases} c & ||x|| \leq 1 \\ 0 & \text{otherwise} \end{cases},$$
- Normal kernel:
$$K_N(x) = c \cdot e^{-\frac{1}{2}||x||^2},$$
- Epanechnikov kernel:
$$K_E(x) = \begin{cases} c(1 - ||x||^2) & ||x|| \leq 1 \\ 0 & \text{otherwise} \end{cases}.$$

Radially symmetric kernel:

$$K(x) = ck(||x||^2),$$

where k is called the profile. Using the profile, and that $g(x) := -k'(x)$,

$$m(x) = \left[\frac{\sum_{i=1}^n x_i g_i}{\sum_{i=1}^n g_i} - x\right]$$

Proof: Mean shift vector is proportional to the gradient of the nonparametric pdf, therefore it is appropriate for mode seeking:

$$m(x) = \frac{\nabla P(x)}{\frac{c}{n}\sum_{i=1}^n g_i}$$

$$P(x) = \frac{c}{n}\sum_{i=1}^n k(||x - x_i||^2) \Longrightarrow \nabla P(x) = \frac{c}{n}\sum_{i=1}^n \nabla k(||x - x_i||^2)$$

$$\nabla P(x) = \frac{2c}{n}\sum_{i=1}^n (x_i - x) g(||x - x_i||^2)$$

$$\nabla P(x) = \frac{2c}{n}\sum_{i=1}^n x_i g(||x - x_i||^2) - \frac{2c}{n}\sum_{i=1}^n x g(||x - x_i|^2|)$$

$$\nabla P(x) = \frac{2c}{n}\sum_{i=1}^n g(||x - x_i||^2)\left[\frac{\sum_{i=1}^n x_i g(||x - x_i||^2)}{\sum_{i=1}^n g(||x - x_i||^2)} - x\right].$$

10.5 Mean shift clustering algorithm

Main steps:

1. A density estimation window is placed on each sample point.
2. Within each window the mean shift vector is calculates, which points toward the maximum density:

$$m_h(x) = \frac{\sum_{i=1}^n x_i g\left(\left|\left|\frac{x - x_i}{h}\right|\right|^2\right)}{\sum_{i=1}^n g\left(\left|\left|\frac{x - x_i}{h}\right|\right|^2\right)} - x,$$

   where h is the bandwidth parameter of the kernel.
3. The window is shifted with the mean shift vector.
4. Step 2, and 3 are repeated until convergence to a local density maximum.
5. The sample points, that converged to the same local maximum, will belonng to the same cluster.

The region for which all trajectories lead to the same mode is called the attraction basin. All data points in an attraction basin of a mode form the cluster.

10.6 Application for image segmentation

Similar color and texture regions will be segmented in the same segment. We can take into account if the pixels are spatially connected. The grayscale model: Each pixel is a "billiard ball" x in the 3D joint spatial-intensity space:

$$x = [x, y, z(x, y)] \in \mathbb{R}^3,$$

where $z(x, y)$ is the gray level. The segmentation process is to find the densest areas in this distribution.

11 Watershed algorithm

11.1 Definition

A mathematical morphology based approach on image segmentation. A grey-level image may be seen as a topographic surface, where the grey level of a pixel is interpreted as its altitude in the surface.

The goal of the algorithm is to find the "watersheds" that are separating the "catchment basins" from each other.

Basic definitions:

- I:2D gray level image.
- Path P of length l between p and q in I is an $(l+1)$ tuple of pixels, where $p = p_0$, $q = p_l$, such that p_i, p_{i+1} are adjacent. $l(P)$ is the length of a given path P.
- Minimum: A minimum $M \subset I$ is a connected plateau of pixels from which it is impossible to reach a point of lower altitude, without having to climb (local minimum).

11.2 Basic steps

- We "pierce holes" in each regional minimum.
- The 3D topography is flooded gradually from below.
- When rising water in distinct catchment basins is about to merge, a dam is built, to prevent the merging.

Instead ow working on the image, this technique is often applied to it's gradient image.

11.3 Types of points

There are three types of points:

- Points belonging to a regional minimum,
- Catchment basin: watershed of a single minimum, where the minimum is trivial.
- Divide lines: watershed lines, points where the minimum is not trivial, crest lines on the surface.

We are interested in the third type.

11.4 Dam construction

At each step of the algorithm, the binary image in obtained in the following manner:

1. Initially set pixels with minimum gray level to 1, others to 0.
2. In each subsequent step, we flood the 3D topography from below and the pixels covered by the rising water are set to 1.

A dam is constructed by the points on which the dilation would cause the "water sets" to megre. The result is a one-pixel thick connected path. We set these dam pixels to a gray level value greater than the max gray value.

11.5 Distance transform

The input is a binary image, showing the foreground, and background, and the result is a gray level image, where the gray level is determined by the distance of the boundary. The closer to the boundary, the darker.

This is implemented with morphological operations, and is often used for the input of the watershed transform.

11.6 Segmentation examples

- Using the gradient magnitude as the segmentation function is a good practice. The gradient is high at borders, and low mostly inside the object.
- Obtaining good foreground markers: regional maxima of the morphology enhanced input image:
  1. Threshold the morphology enhanced image.
  2. Using the watershed transform of the distance transform, and then looking for the watershed ridge lines of the result
- Works very well on masonry wall images.

11.7 Use of markers

The problem with over-segmentation, is that we can't overcome it with a simple filtering. To solve this problem, we can limit the number of regions by specifying the objects of interest.

The two types of markers:

- Internal marker: marks an object, that we want selected.
- External marker: marks the background. Two areas marked with external markers are allowed to be merged.

We only flood from the markers, not the minimum. This way the flooding will produce as many catchment basins as there are markers.

12 Image recovery

12.1 Definition

The modeling and removal of the degradation the image is subjected to, based on some optimality criteria.

In case of recovery there is a degradation we want to remove, lost information we want to recover (e.g. make blurred text readable again). It is done by modeling the degradation and making assumptions about the degradation and the original image.

12.2 Examples: sources of degradation and forms of recovery

Sources of degradation:

1. Motion
2. Atmospheric turbulence
3. Out-of-focus lens
4. Finite resolution of the sensor
5. Limitations of the acquisition system
6. Transmission error
7. Quantization error
8. Noise

Forms of restoration:

1. Restoration/Deconvolution
2. Removal of compression artifacts
3. Super-Resolution
4. Inpainting/Concealment
5. Noise smoothing

12.3 Degradation and restoration model

The original image goes through a system H, that introduces some type of degradation resulting the observed image y:

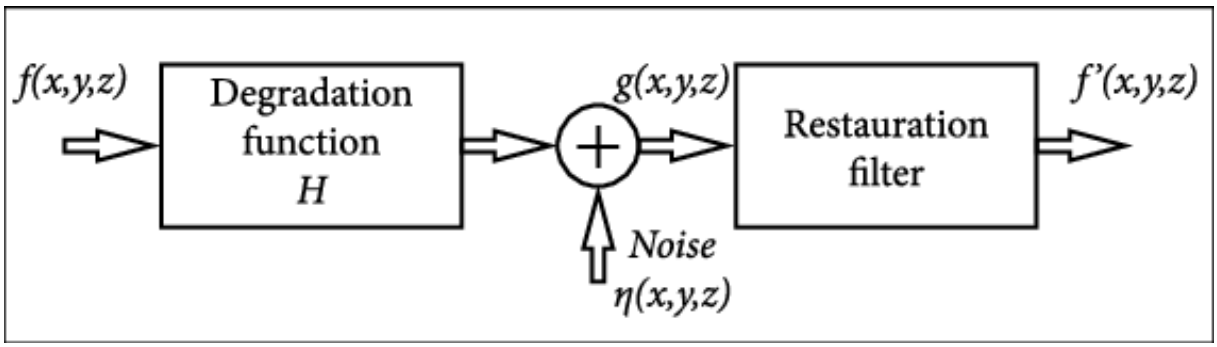
$$x(n_1, n_2) \to H \to y(n_1, n_2).$$

The objective is to reconstruct x based on

- y and H: recovery
- y: blind recovery
- y and partially H: semi-blind recovery.

If we know x and

- y: system identification
- H: system implementation



The model of degredation for restoration problems:

$$y(n_1, n_2) = H\big[x(n_1, n_2)\big] + n(n_1, n_2).$$

If an LSI degradation system is assumed, with signal independent additive noise:

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) + n(n_1, n_2).$$

The restoration problem is called deconvolution.

12.4 Matrix-vector form

The Point Spread Function (PSF) is the system's impulse response, an image $h(n_1, n_2)$ which describes the response of an imaging system to a point source or a point object.

The observed image $y(n_1, n_2)$ can be taken as the convolution of the object and the PSF:

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2).$$

While 2D convolution in a naive form is quite slow ($O(N^4)$), a 1D convolution can be represented in a matrix vector form:

$$y(n) = x(n) * h(n) = \sum_k x(k)h(n-k),$$

where $x : 1 \times N$, $h : 1 \times L$, $y : 1 \times (N + L - 1)$:

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N+L-2) \end{bmatrix} = \begin{bmatrix} h(0) & 0 & 0 & \cdots & 0 \\ h(1) & h(0) & 0 & \cdots & 0 \\ h(2) & h(1) & h(0) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h(L-1) \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}.$$

This means that if we reorganize the image to be a 1D vector, we could use the matrix form:

$$y = Hx + n$$

12.5 Operation in the Fourier domain

The convolution theorem says, that convolution in the spatial domain becomes a simple element wise multiplication in the Fourier domein. The degredation model in the Fourier domain becomes:

$$Y(\omega_1, \omega_2) = X(\omega_1, \omega_2) \cdot H(\omega_1, \omega_2) + N(\omega_1, \omega_2).$$

12.6 Inverse filter

This is the simplest deconvolution filter, developed for LSI systems. It can also be simply implemented in the frequency domain.

The objective is to find the x that minimizes the Euclidian norm of the error:

$$\arg\min_x \big(J(x)\big) = \arg\min_x \big(||y - Hx||_2^2\big).$$

The first derivative of the minimization function must be set to 0:

$$\frac{\partial J(x)}{\partial x} = 0 \Rightarrow \frac{\partial}{\partial x}\big(y^T y - 2x^T H^T y + x^T H^T H x\big) = -2H^T y + 2H^T H x = 0$$

$$H^T H x = H^T y$$

$$x = \big(H^T H\big)^{-1} H^T y.$$

In the frequency domain:

$$|H(\omega_1,\omega_2)|^2 X(\omega_1,\omega_2) = H^*(\omega_1,\omega_2)Y(\omega_1,\omega_2)$$

$$X(\omega_1,\omega_2) = \frac{H^*(\omega_1,\omega_2)Y(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2}$$

Problem: since $H(\omega_1,\omega_2)$ is a low pass filter, it is very likely to drop off rapidly as the distance increases.

Simplification: consider a system, where $H(\omega_1,\omega_2)$ is real. In this case, the invers filter output is calculated as:

$$\hat{X}_{inv}(\omega_1,\omega_2) = \frac{Y(\omega_1,\omega_2)}{H(\omega_1,\omega_2)}.$$

If we add noise:

$$Y(\omega_1,\omega_2) = H(\omega_1,\omega_2)X(\omega_1,\omega_2) + N(\omega_1,\omega_2)$$

$$X(\omega_1,\omega_2) = \frac{Y(\omega_1,\omega_2) - N(\omega_1,\omega_2)}{H(\omega_1,\omega_2)} = \frac{Y(\omega_1,\omega_2)}{H(\omega_1,\omega_2)} - \frac{N(\omega_1,\omega_2)}{H(\omega_1,\omega_2)}.$$

12.7 Noise amplification issue

The problem is that $H(\omega_1,\omega_2)$ is 0, or very small at certain frequency pairs, and $N(\omega_1,\omega_2)$ is large. We can see that $\frac{N(\omega_1,\omega_2)}{H(\omega_1,\omega_2)}$ (the error estimation) can be huge.

The drawbacks of this method is called the strong amplification of noise. A solution may be to threshold $H(\omega_1,\omega_2)$. This method is called the pseudo-inverse filtering.

12.8 CLS filter and Wiener filter

12.8.1 Constrained Least Square Methods

The objective is to reduce the noise amplification effect of the inverse filter by adding extra constraints about the restored image. We still have the term:

$$\arg\min_x \big(J(x)\big) = \arg\min_x \big(||y - Hx||^2\big),$$

but we also have a second term, describing a prior knowledge about the smoothness of the original image:

$$||Cx||_2^2 < \varepsilon.$$

Putting the two together with the introduction of a regularization parameter α:

$$\arg\min_x \big(||y - Hx||_2^2 + \alpha||Cx||_2^2\big) \Rightarrow x = (H^T H + \alpha C^T C)^{-1} H^T y,$$

where C is a high pass filter.

12.8.2 Wiener filter

Treat the image as a sample from a 2D random field. The image is part of a class of samples, realizations of the same random field.

Autocorrelation:

$$R_{ff}(n_1,n_2,n_3,n_4) = E\big[f(n_1,n_2)f^*(n_3,n_4)\big],$$

the expected value over many realizations of the 2D random field.

Power-spectrum (Wide Sense Stationary (WSS) input):

$$P_{ff}(\omega_1,\omega_2) = DFT\big\{R_{ff}(d_1,d_2)\big\},$$

where $d_1 = n_1 - n_3$, and $d_2 = n_2 - n_4$. The objective:

$$\tilde{x}(n_1, n_2) = \arg \min_{x(n_1,n_2)} E\big[|x(n_1, n_2) - \tilde{x}(n_1, n_2)|^2\big].$$

We look for the solution in the following format:

$$\tilde{x}(n_1, n_2) = r(n_1, n_2) * y(n_1, n_2),$$

where the input image is assumed to be WSS with autocorrelation $R_{XX}(n_1, n_2)$. The recovered image is obtained in the Fourier domain:

$$X(\omega_1, \omega_2) = R(\omega_1, \omega_2) \cdot Y(\omega_1, \omega_2).$$

13 Descriptors overview, Moravec, Harris, and correspondence matching

13.1 Local feature descriptors

There are different types of use of the descriptors:

- Descriptors and matching keypoints:
  1. Feature/keypoint detection
  2. Local feature description around the keypoints
  3. Keypoint matching
- Bag-of-Features (or bag-of-words)
  1. Feature detection
  2. Feature description
  3. Frequency histogram constructionfor image, or image part description
- Description on a specific area
  1. Find the Region of Interest (ROI)
  2. Description of the ROI
  3. Classification/Clustering of ROI descriptor

13.2 Application examples

The detection and description of local features has an important role in many applications:

- Object recognition/detection/tracking
- Image and video retrieval
- Image registration, motion estimation
- Wide baseline matching
- Texture classification
- Structure from Motion

13.3 Interest points

- **0D structure:** single points. Not useful for matching
- **1D structure:** lines. Edges can be localized in 1D
- **2D structure:** corners, interest points.

Finding featured points is based on the idea of auto-correlation: sum of squared differences (SSD) matching.

13.4 Moravec corner detector

- Take a window W in the image
- Shift in four directions:
$$[\Delta x, \Delta y] \in \big\{[1,0],[0,1],[1,1],[-1,1]\big\}$$
- compute SSD difference for each direction:
$$SSD(\Delta x, \Delta y) = \sum_{(x_k,y_k)\in W} \big(I(x_k,y_k) - I(x_k+\Delta x, y_k+\Delta y)\big)^2$$

- Find the min difference at each pixel
- Local maxima in the min image are corners.

A limitation to this method, that the corners are not isotropic: if an edge is present that is not in the direction of the neighbors (horizontal, vertical, or diagonal), then the smallest SSD will be large and the edge will be incorrectly chosen as an interest point.

13.5 Harris corner detector

Auto-correlation function (SSD) for a point (x, y), and an arbitrary shift $(\Delta x, \Delta y)$ (not only 4 directions):

$$f(x,y) = \sum_{(x_k,y_k)\in W} \big(I(x_k,y_k) - I(x_k+\Delta x, y_k+\Delta y)\big)^2$$

Discrete shifts can be avoided with the auto-correlation matrix (Taylor approximation):

$$I(x_k+\Delta x, y_k+\Delta y) = I(x_k,y_k) + \begin{bmatrix} I'_x(x_k,y_k) & I'_y(x_k,y_k)\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix},$$

then

$$f(x,y) = \sum_{(x_k,y_k)\in W}\left(\begin{bmatrix} I'_x(x_k,y_k) & I'_y(x_k,y_k)\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix}\right)^2 =$$

$$= \sum_{(x_k,y_k)\in W}\begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix} I'_x(x_k,y_k)\\ I'_y(x_k,y_k)\end{bmatrix}\begin{bmatrix} I'_x(x_k,y_k) & I'_y(x_k,y_k)\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} =$$

$$= \sum_{(x_k,y_k)\in W}\begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix} I_x'^2 & I'_xI'_y\\ I'_xI'_y & I_y'^2\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} =$$

$$= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\sum_{(x_k,y_k)\in W}\begin{bmatrix} I_x'^2 & I'_xI'_y\\ I'_xI'_y & I_y'^2\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix} =$$

$$= \begin{bmatrix}\Delta x & \Delta y\end{bmatrix}\begin{bmatrix}\sum_{(x_k,y_k)\in W} I_x'^2 & \sum_{(x_k,y_k)\in W} I'_xI'_y\\ \sum_{(x_k,y_k)\in W} I'_xI'_y & \sum_{(x_k,y_k)\in W} I_y'^2\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta y\end{bmatrix},$$

where the middle matrix M is the auto-correlation matrix of the local gradient map, which captures the structure of the local neighborhood, and the measures are based on the eigenvalues λ_1, λ_2:

- 0 strong eigenvalues: uniform region
- 1 strong eigenvalue: contour
- 2 strong eigenvalues: interest point (corner)

Threshold the eigenvalues, then find the maximum for localization.

To avoid eigenvalue computation, we can alternatively use:

$$R = Det(M) - \kappa Tr^2(M),$$

where

$$Tr(M) = a_{11} + a_{22}, \qquad \kappa \in [0.04, 0.15].$$

If $R >> 0$, we have an interest point, if $R << 0$ we have a contour.

13.6 Detemining corresponences

To determine correspondences compare the vectors using a distance measure. Let W_1 and W_2 be two rectangular windows in I_1 and I_2 respectively. To compare W_1 and W_2, we can use the sum-square difference of the values of the pixels in a square neighborhood about the points being compared. This is the simplest measure.

13.7 Correlation based feature matching

Basic feature matching with Harris corners and correlation, we get very good results in the presence of occlusion and clutter. There is no invariance to scale and affinite changes, limited invariance to image rotation.

14 SIFT

14.1 SIFT motivation

The Harris operator is not invariant to scale and correlation is not invariant to rotation. For better image matching, Lowe's goal was to develop an interest operator that is invariant to scale and rotation.

Also, Lowe aimed to create a descriptor that was robust to the variations corresponding to typical viewing conditions. The descriptor is the most-used part of SIFT.

Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters.

14.2 Advantages

- Invariant to translation, scaling, and rotation.
- Robust to illumination changes, noise, minor changes in viewpoint.
- Robust to local geometric distortion.
- Highly distinctive.
- SIFT based object detectors are robust to partial occlusion.

14.3 Introduction of the four main steps

The 4 steps of the algorithm:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint description

14.4 Including Gaussian pyramid

Blurring image with a Gaussian kernel: Loosing details i.e. transforming the image into a different scale:

$$L(x, y, k \cdot \sigma) = G(x, y, k\sigma) * I(x, y),$$

where

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

increasing k will increase the amount of blur. At certain level of blur, the image can be spatially downscaled. If we then apply the blur again and again to the downscaled image, we can downscale it arbitrarily. This is called the Gaussian pyramid.

14.5 LoG-DoG approximation

Laplacian of Gauss (LoG) kernel:

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

It is extremely useful, it can find stable features, and give excellent notion of scale, but the calculation is very costly.

Approximation of LoG:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{\sigma(k-1)}$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$$

The Difference of Gaussian (DoG) approximates LoG:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma).$$

14.5.1 Scale-space extrema detection

Key point detection with DoG:

- $I(x, y)$ is the original image
- $G(x, y, k\sigma)$ is the gaussian blur at scale $k\sigma$
- The original image convoled with Gaussian kernel at different scales:

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y)$$

- The convoled images are grouped by octave. The difference of consecutive convoled images is taken in an octave:

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma)$$

Scale space is seperated by octaves:

- Octave 1 uses scale σ
- Octave 2 uses scale 2σ
- etc...

In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images. Adjacent Gaussians are subtracted to produce the DOG. After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image $\frac{1}{4}$ the size to start the next level.

14.6 Keypoint localization and filtering

Detect maxima and minima of difference-of-Gaussian in scale space. Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below.

Localization is done with sub pixel accuracy, based on the interpolation of nearby data.

It rejects weak candidates:

- Low contrast points.
- Poorly localized points along edges.

Similar approach to Harris, but here the ratio of the Tr^2 and determinant of the Hessian detector matrix is calculated.

We then assign orientation to these points, to a 10° approximation with orientation histograms.

14.7 Defining the descriptor

At this point each key point has

- location
- scale
- orientation

Next step is to compute a descriptor for the local image region about each key point, that is highly distinctive, and invariant as possible to variations such as changes in viewpoint and illumination.

Steps:

- Normalization:
  - Rotate the window to standard orientation
  - Scale the window size based on the scale at which the point was found.
- Take a 16×16 point neighborhood around the keypoint and divide it into a 4×4 gradient window.
- Compute gradient magnitude and orientation at each point in the region.
- Build the orientation histogram of the 4x4 samples in each window with 8 direction bins.
- 4×4 times 8 directions gives a vector of 128 values.

15 Machine Learning - supervised algorithms

15.1 Introduction

"Field of study that gives computers the ability to learn without being explicitly programmed.", "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

15.1.1 Machine learning algorithms

- Supervised learning
  - The supervised algorithms are trained on labeled data, where the desired output is known. The goal is to train a classifier that can work on previously unknown data.
    - Regression: Prediction of continuous valued output
    - Classification: prediction of discrete valued output.
- Unsupervised learning
  - In case of unsupervised learning the training data is not labeled.
  - Applications:
    - Social Network Analysis
    - Market segmentation
    - Compression
    - Image segmentation
- Reinforcement learning
  - The goal is to get an agent to act in the world so as to maximize its rewards

15.2 Linear regression

Representation of h for linear regression with one variable:

$$h_\theta = \theta_0 + \theta_1 x.$$

The question is how to find the best values for θ. The idea is to find the parameters (θ_0, θ_1) so that they minimize a cost function:

$$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \big(h_\theta(x_i) - y_i\big)^2.$$

There are other cost functions such as the cross-entropy cost function.

15.3 Gradient descent

We will use the gradient descent method to find the minimum of $J(\theta_0, \theta_1)$. Start with arbitrary initial values, and in each iteration change θ_0 and θ_1 so that J is reduced, untill it reaches a minimum value.

To achieve this, the folowing update rule is used:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1),$$

where α is the learning rate. The update is done simultaneously for all the θ_j.

15.4 Logistic regression

This is a supervised classification algorithm. From the input features, it predicts a discrete output.

The training data y is a vector of 0's and 1's for single class classification, but there may be more, implying that there are more classes.

Logistic regression produces answers betweenn $[0, 1]$. To achieve this we take the sigmoid function of $\theta^T x$:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

The interpretation of the result is a probability of x being in the class.

15.5 Decision boundary

To predict binary class labels we use a threshold: 0.5. We can interpret this threshold as a boundary in the nth space, which separates the elements of a class from everything else.

We will need another cost function, because we want a convex function. For this purpose, we define the cost function:

$$cost\big(h_\theta(x), y\big) = \begin{cases} -\log\big(h_\theta(x)\big) & y = 1 \\ -\log\big(1 - h_\theta(x)\big) & y = 0 \end{cases}.$$

This way, if $y = 1$ and $h_\theta(x) = 1$, then the cost is 0, otherwise as it gets smaller and smaller, it goes to infinity. Same with the other case. To unify this cost function, we need to sum it up:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \Big(y_i \log\big(h_\theta(x_i)\big) + (1 - y_i) \log\big(1 - h_\theta(x_i)\big) \Big).$$

15.6 Softmax regression

If we have more the one class, we need another way to represent affiliation. For this we use onehot encoding. In this method every label is represented with a vector, which is only 1 at the class' position, everywhere else it is 0.

For K exclusive categories we can use Softmax classifier, where the hypothesis function maps the input x to the following a K-dimensional hypothesis vector:

$$h_\theta = \frac{1}{\sum_{j=1}^{K} e^{\theta_j^T x_j}} \begin{bmatrix} e^{\theta_1^T x} \\ e^{\theta_2^T x} \\ \vdots \\ e^{\theta_K^T x} \end{bmatrix}.$$

Not θ is a matrix, and each of its columns is the parameterization of the x feature vector for one class. The kth element the hypothesis vector can be interpreted as probability of membership of the kth category:

$$P(y_i = k | x_i, \theta) = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{K} e^{\theta_j^T x_j}}.$$

The cost function (cross-entropy) is the following:

$$J(\theta) = -\sum_{i=1}^{n} \sum_{k=1}^{K} 1\{y_i = k\} \log P(y_i = k | x_i, \theta).$$

We cannot solve for the minimum of $J(\theta)$ analytically, thus we'll resort to an iterative optimization algorithm.

15.7 Regularization

If we have too few features, we can't learn a hypothesis, that fits the data well. This is called underfitting. To handle underfitting we can introduce new features.

If we have too many features we can learn a hypothesis that fits the training data very well, but fails on new samples. This is called overfitting.To handle overfitting:

- We can reduce the number of features
- We can apply regularization:
  - We can keep all the features but we reduce their magnitude.
  - Works well if we have a lot of features and each contributes a little bit to predict y.
  - The idea is to keep the parameters low, to get a simpler hypothesis function, which is less prone to overfitting.

We can regularize, wit modifying the cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost\big(h_\theta(x_i), y_i\big) + \lambda \sum_{j=1}^{n} \theta_j^2.$$

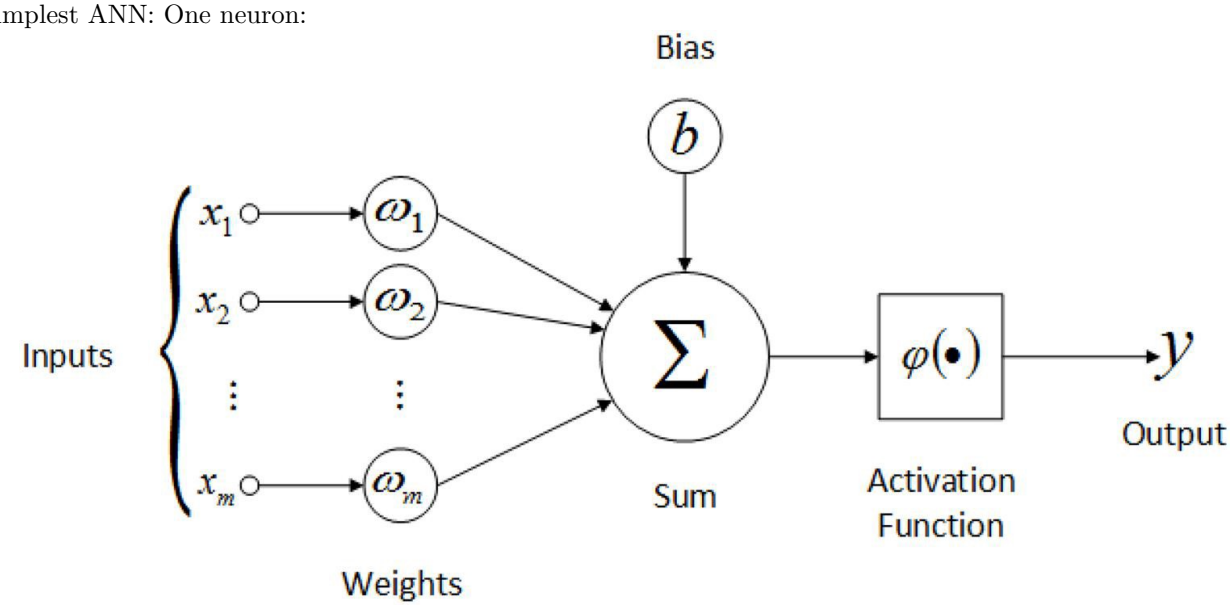
The regularization parameter λ controls the trade-off between two goals: fitting the data well, and keeping the parameters low, to avoid overfitting.

If λ is too large and the parameters will be close to 0, the model won't fit the data, we will see underfitting.

16 Introduction to Deep Learning

16.1 Artificial neural networks

Simplest ANN: One neuron:



$$y = \varphi(z) = \varphi\Big(\sum_{i=1}^{n} x_i w_i + b\Big) = \varphi(xw + b),$$

where x is the input vector, W is the vector of weights, and b is the bias.

These simple computational units (the artificial neurons) can be organized into networks. (Like the "real" neurons in the human nervous system). Feed-Forward networks only contain forward connections: the neurons in layer k are only connected to neurons in layer $k+1$. No backward or within-layer connections.

We call layers that have neurons connecting to every neuron in the next layer, fully connected layers.

16.2 Forward pass

The Forward Pass is the calculation of the response of the network to an input. Assuming you have a trained network, all trainable parameters (θ) are tuned for the task, it can be calculated as a series of matrix-vector operations:

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} = \varphi\left(\begin{bmatrix} \cdots & w_1 & \cdots \\ \cdots & w_2 & \cdots \\ \vdots & & \\ \cdots & w_m & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) = \varphi(Wx + b).$$

16.3 Activation function

The activation function is a non-linear function applied in each neuron on z, the weighted sum of the neuron's input.

Commonly used activation functions:

- Sigmoid
- Tanh
- Relu

16.4 Training

Forward propagation:

$$\tilde{y} = g\big(W_3\varphi(W_2\varphi(W_1x + b_1) + b_2) + b_3\big).$$

For loss function we can use cross-entropy for classification, and L^2 loss for regression.

To optimize the weights and biases, to minimize the loss, we use gradient descent algorithm:

- The weights and biases are initialized as small random numbers.
- Each parameter (weights and biases) are modified simultaneously in each iteration:

$$W_{i,j}^{(l)} = W_{i,j}^{(l)} - \alpha\frac{\partial}{\partial W_{i,j}^{(l)}}J, \qquad b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J.$$

We can compute the gradient of loss for each parameter with the backpropogation algorithm, which is based on the chain rule:

$$F(x) = f(g(x)) \Rightarrow F'(x) = f'(g(x))g'(x).$$

Regularization with L^2 or L^1 norms. Other ways of regularization:

- Dropout
  - During training each neuron can be deactivated with a certain probability.
  - In each iteration of the training, a different sub-network is optimized.
- Early stopping:
  - Stop the training if the performance drops on the validation set

16.5 Convolutional neural networks

Fully connected layers on raw pixels are not efficient because:

- they don't scale well to full images
- dosen't take into account the correlation.

16.5.1 Fully connected layers

Each pixel is one input for the network, so the number of parameters can be very high even for a medium sized image.

16.5.2 Convolutional layers

The weights are reused in different parts of the image, so much less parameters. Also practical convolutional kernels will be learned. Each kernel has 3 dimensions.

Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels).

The neurons along the depth are all looking at the same region of the image. But they learn a different set of weights.

16.5.3 Pooling

It is common practice to insert a Pooling layer in-between successive convolutional layers. The goal is to gain robustness against small changes in the location of a feature, and also to reduce dimensionality (downsample along the spatial dimensions).

They introduce no parameters, since it computes a fixed function. Types of pooling:

- Max pooling
- Average pooling

16.5.4 Transfer learning

The models that were trained on a big dataset could be partly reused on other tasks:

- Reuse a convnet as fixed feature extractor: Take a trained model, remove its final fully connected layer and use the rest as a fixed feature extractor for a classifier (like a linear SVM or a softmax).
- Fine tuning: use the network pre-trained on an other data and use it as initialization for the training on the data of interest. Usually this fine tuning is done with low learning rate, or even with the first few layers kept fixed.

Using pre-trained models also helps against over fitting.