Computer Vision

Xin Wang

March 6, 2022

Contents

Chapter 1

Image Formation

A digital image is formed from three components:

1. Lighting
2. Reflectance
3. Optics and Sensors

1.1 Lighting

Lighting has several properties of concern:

- Location
- Intensity of light
- Spectrum of light

There are many different sources of light. Four commonly used models to describe light sources are:

- **Point Light Source**: The light is inside the scene at a specific location only and it shines light equally in all directions. An example is be a table lamp.
- **Area Light Source**: The light source comes from a rectangular area and projects light from one side of the rectangle. An example is a florescent light fixture in a ceiling panel.
- **Sun Light Source**: The light is outside the scene and far enough away that all rays of light are basically from the same direction. An example is the sun in an outdoor scene.
- **Spotlight Light Source**: The light is focused and forms a cone-shaped envelop as it projects out from the light source. An example is a spotlight in a theatre.

1.2 Reflectance

A general model for modelling reflectance is the **Bidirectional Reflectance Distribution Function (BRDF)**. The model describes how much light arriving at incident direction is emitted in reflected direction.

$$f_r\left(\theta_i, \sigma_i, \theta_r, \sigma_r, \lambda\right) = \frac{dL_r}{dE_i}$$

where:

- θ_i and σ_i: Incident direction
- θ_r and σ_r: Reflected direction
- λ: Wavelength
- dL_r: Output power
- dE_i: Input power

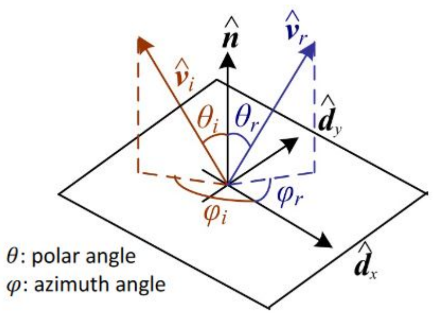
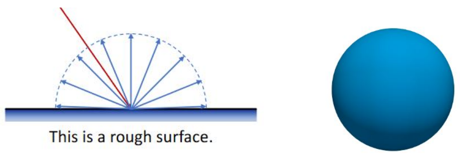


Figure 1.1: BRDF model

Considering the two ideal cases of reflection: **Diffuse reflection** and **Specular reflection**

- **Diffuse reflection**:

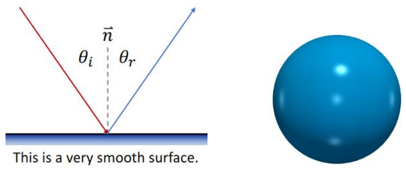
  

  The light is scattered uniformly in all directions so the BRDF is constant:

  $$f_r\left(\theta_i, \sigma_i, \theta_r, \sigma_r, \lambda\right) = f_r(\lambda)$$

  This effect on the viewed image is similar to the appearance of a rough surface.

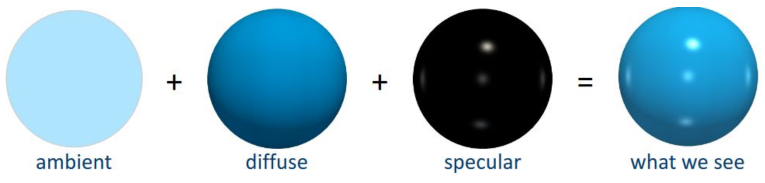
- **Specular reflection**:

  The light is reflected in a mirror-like fashion. The reflection and incident directions are symmetric with respect to the surface normal $\vec{n}$: $\theta_r = \theta_i$

For the majority of cases in the real world, there is a combination of **diffuse reflection**, **specular reflection** and **ambient illumination**. Ambient illumination accounts for general illumination which may be complicated to model such as the inter-reflection between walls in a room and distant light sources as seen in sunny outdoor environments.

This observation is formally state in the **Phong Reflection Model** which is an empirical model in computer graphics that describes how a surface reflects light as a combination of ambient, diffuse and specular components.



1.2.1 Computer graphics

There is an unique relationship between computer graphics and computer vision similar to the relationship of writing and reading. Computer graphics is concerned with generating images while computer vision is concerned with intepreting images.

This relationship is particularly valuable as computer graphics in photorealistic games can be used to generate data i.e. images + labels for training computer vision algorithms.

1.3 Optics

Camera sensors imitates the human eye which are human sensors. In the human eye, there are two types of neural cells in the retina:

- **Cone cells**: Colour vision and functions in the bright light.
- **Rod cells**: More sensitive to the light but monochromatic and functions in the dim light like night time.

Camera sensors are very much the same: **Charge-coupled Device (CCD)** and **Complementary Metal-oxide Semiconductor (CMOS)**.

1.3.1 Colour Filter Arrays (CFA)

A color filter array (CFA) or color filter mosaic (CFM) is a mosaic of tiny color filters placed over the pixel sensors of an image sensor to capture color information.

The most common CFA is the most single-chip digital image sensors used in digital cameras to create a colour image is the *Bayer Filter Mosaic*. The Bayer Filter Mosaic is arraged to mimic the human eyes i.e. most sensitive to green light with 50% green, 25% red and 25% blue.

The RGB of different cameras may be different, i.e. with different sensitivities to wavelengths. This different colour sensitivity is why the same picture with different cameras would look different.

1.3.2 Bayer Colour Filter

With its arrangement, only one colour is available at each pixel. The other two colours can be interpolated from neighbouring pixels. Through this interpolation at each pixel, the RGB values can be obtained.

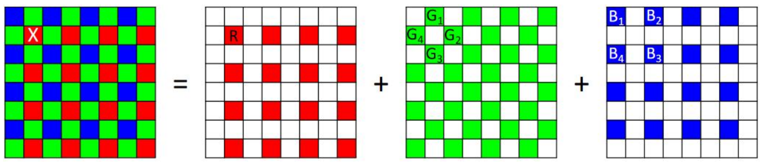
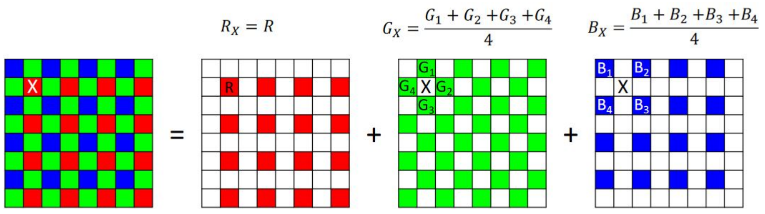


Figure 1.2: Bayer colour filter and interpolation

1.3.3 Demosaicing

A demosaicing algorithm is a digital image process used to reconstruct a full colour image from incomplete colour samples output from an image sensor overlaid with a CFA.

A simple method is bilinear interpolation where the red value of a non-red pixel is computed as the average of the two or four adjacent red pixels, and similarly for blue and green.


$R_X = R$
$G_X = \frac{G_1 + G_2 + G_3 + G_4}{4}$
$B_X = \frac{B_1 + B_2 + B_3 + B_4}{4}$
X
=
R
+
G_1 G_4 X G_2 G_3
+
B_1 B_2 X B_4 B_3

1.3.4 Colour spaces

Colour spaces are mathematical models describing the way colours can be represented. Colour spaces can be seen as a box containing all possible colours that can be produced by mixing primary colours of RGB.

The CIE 1931 XYZ was created by the International Commission on Illumination and represents the colour space using the primary colours X, Y and Z.

$$x = \frac{X}{X+Y+Z}$$
$$y = \frac{Y}{X+Y+Z}$$
$$z = \frac{Z}{X+Y+Z}$$

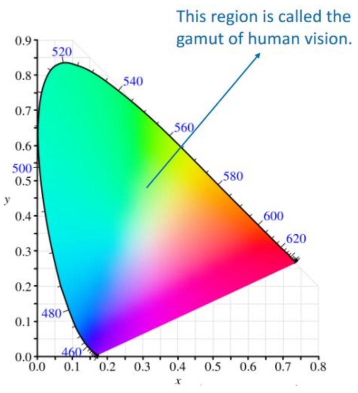


Figure 1.3: CIE 1931

There are various different forms colour spaces such as sRGB, HSV and CMYK. All these colour spaces can represent the same colour, but using different primary colours or coordinate systems.

1.3.5 Quantisation

Quantisation is the process that maps continuous signal to discrete signal. For colours, color quantization reduces the number of colors used in an image. This is important for displaying images on devices that support a limited number of colors and for efficiently compressing certain kinds of images.

It is important to note that numerical errors occur during quantisation, which depends on the number of bits used. The more bits, the less quantisation error.

Chapter 2

Image Filtering

The goal of using filters is to modify or enhance image properties and/or to extract valuable information from the pictures such as edges, corners, and blobs.

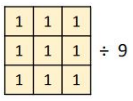
There are various types of image filters such as:

- Identity filter
- Low-pass/Smoothing filters: Moving average filter or Gaussian filter
- High-pass/Sharpening filters
- Denoising filter: Median filter

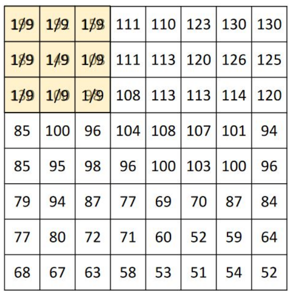
2.1 Smoothing Filters: Moving Average Filter

A moving average filter moves a window across the signal and calculates the average value within the window. A **filter kernal** is specified to indicate what values are averaged in the moving average filter. A filter kernal [1] is a small matrix used for blurring, sharpening, edge detection etc.

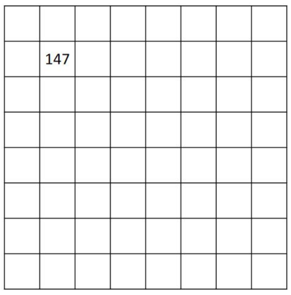
In images, the moving average filter removes high frequency signal e.g. noise or sharpness. This operation results in a smooth but blurry image. For example, consider a **box blur** kernal and its effect:



1	1	1	
1	1	1	÷ 9
1	1	1	

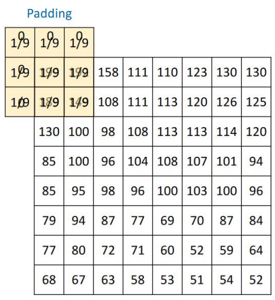


1/9	1/9	1/9	111	110	123	130	130
1/9	1/9	1/9	111	113	120	126	125
1/9	1/9	1/9	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52



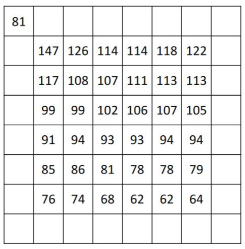
	147						

Due to the nature of this method of calculations, the output image will be smaller than the input image. The boundary pixels are generally dealt with using padding of various methods such as constant value, mirroring values etc. The following example uses 0 padding for its boundary pixels.



Padding

1/9	1/9	1/9						
1/9	1/9	1/9	158	111	110	123	130	130
1/9	1/9	1/9	108	111	113	120	126	125
	130	100	98	108	113	113	114	120
	85	100	96	104	108	107	101	94
	85	95	98	96	100	103	100	96
	79	94	87	77	69	70	87	84
	77	80	72	71	60	52	59	64
	68	67	63	58	53	51	54	52



81							
	147	126	114	114	118	122	
	117	108	107	111	113	113	
	99	99	102	106	107	105	
	91	94	93	93	94	94	
	85	86	81	78	78	79	
	76	74	68	62	62	64	

Input

Output

By increasing the size of the kernal e.g. into a 7×7 matrix, the image will become blurrier.

[1]https://en.wikipedia.org/wiki/Kernel_(image_processing)

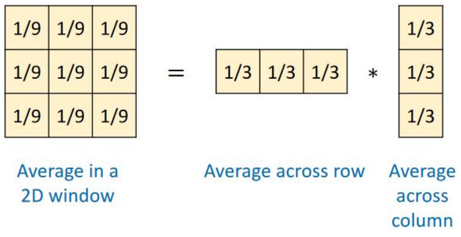
2.1.1 Brute Computational Complexity

Note that: *Image size*: $N \times N$ where N is the number of pixels and *Kernal size*: $K \times K$ where K is size of the filter kernal matrix

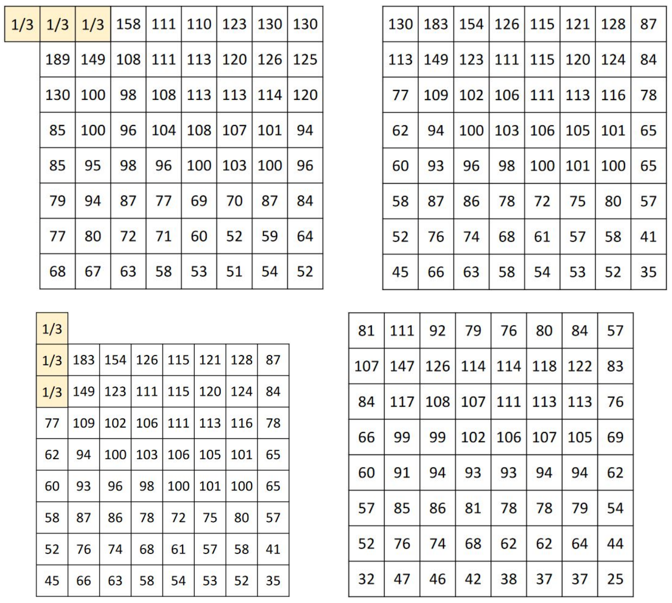
- There are N^2 pixels
- At each pixel, there are K^2 multiplications and $K^2 - 1$ summations.
- In total, there are: N^2K^2 multiplications and $N^2(K^2 - 1)$ summations
- Complexity is: $O(N^2K^2)$

2.1.2 Separable filter

If a big filter can be separated as the consecutive operation of two small filters, then the first filter can be applied to the the input image then the second filter. For example, consider the previous blur filter kernal divided into two smaller filters:



This calculation procedure results in the same result as the previous result:



1/3	1/3	1/3	158	111	110	123	130	130
	189	149	108	111	113	120	126	125
	130	100	98	108	113	113	114	120
	85	100	96	104	108	107	101	94
	85	95	98	96	100	103	100	96
	79	94	87	77	69	70	87	84
	77	80	72	71	60	52	59	64
	68	67	63	58	53	51	54	52

130	183	154	126	115	121	128	87
113	149	123	111	115	120	124	84
77	109	102	106	111	113	116	78
62	94	100	103	106	105	101	65
60	93	96	98	100	101	100	65
58	87	86	78	72	75	80	57
52	76	74	68	61	57	58	41
45	66	63	58	54	53	52	35

1/3							
1/3	183	154	126	115	121	128	87
1/3	149	123	111	115	120	124	84
77	109	102	106	111	113	116	78
62	94	100	103	106	105	101	65
60	93	96	98	100	101	100	65
58	87	86	78	72	75	80	57
52	76	74	68	61	57	58	41
45	66	63	58	54	53	52	35

81	111	92	79	76	80	84	57
107	147	126	114	114	118	122	83
84	117	108	107	111	113	113	76
66	99	99	102	106	107	105	69
60	91	94	93	93	94	94	62
57	85	86	81	78	78	79	54
52	76	74	68	62	62	64	44
32	47	46	42	38	37	37	25

2.1.3 Separable filter complexity

Note that: *Image size*: $N \times N$ where N is the number of pixels and there are two filter kernals: $1 \times K$ and $K \times 1$.

- There are N^2 pixels
- At each pixel, there are K multiplications and $K - 1$ summations.
- In total, there are: $2N^2K$ multiplications and $2N^2(K - 1)$ summations
- Complexity is: $O(N^2K)$ which is faster than the previous $O(N^2K^2)$

2.2 Identity Filter

The **Identity Filter Kernal** simply returns the same value of the image i.e. the input and output image is the same.

2.3 Smoothing Filters: Gaussian Filter

The Gaussian Filter uses a 2D Gaussian Distribution as its filter kernal:

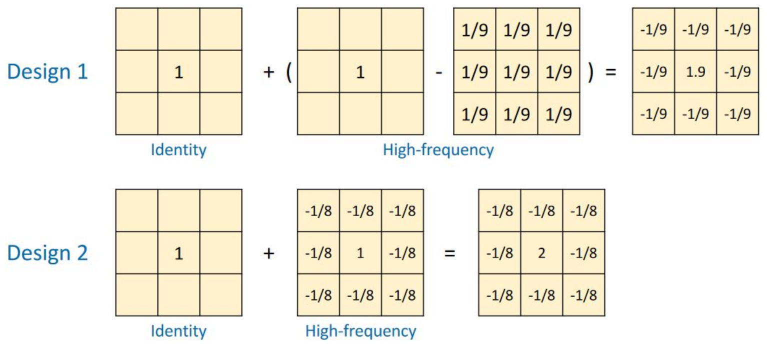
$$h(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

The 2D Gaussian filter is a separable filter, equivalent to two 1D Gaussian filters with the same σ, one along x-axis and the other along y-axis:

$$h(i,j) = h_x(i) * h_y(j)$$
$$h_x(i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}}$$

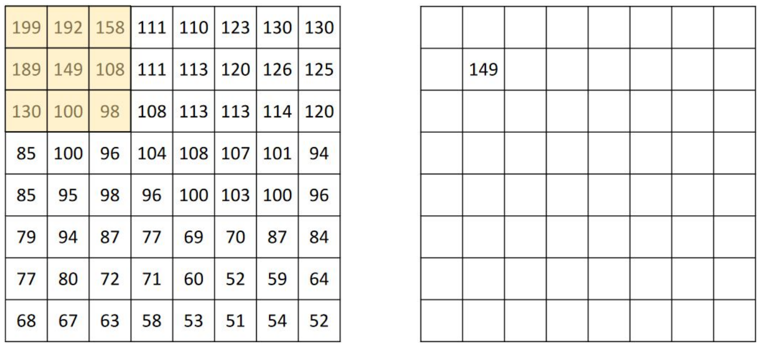
2.4 High-pass Filters

There are various methods of designing high-pass filters including using low-pass filters as seen in Design 1.



2.5 Denoising Filters: Median Filter

Median filters are non-linear filters that is often used for denoising an image. A common method of performing median filter is to move the sliding window and replacing the centre pixel using the *median value* in the window.



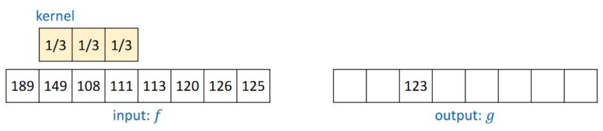
199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

	149						

Chapter 3

Image Filtering II

The concept of filtering can be described mathematically for example: where this operations can be described



as:

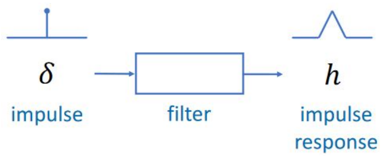
$$g[n] = \frac{1}{3}.f[n-1] + \frac{1}{3}.f[n] + \frac{1}{3}.f[n+1]$$

This concept of describing filters mathematically is extensively used in Signal Processing where it describes filters as:

A device/process 'h' that removes unwanted components/features from a input signal 'f' and generates an output signal 'g'

Impulse Response

Impulse responses are used to mathematically describe a filter. The impulse response h is the output of a filter when the input is an impulse signal δ.



- **Continuous signal**: An impulse signal is the *Dirac Delta function* $\delta(x)$:

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

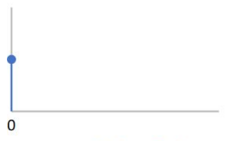
$$\int^{\infty} \delta(x)dx = 1$$

- **Discrete signal**: An impulse signal is the *Kronecker function* $\delta[i]$:

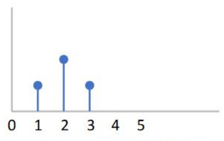
$$\delta[i] = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases}$$

The impulse response h completely characterises a *linear time-invariant* filter. Because any input signal can be form using impulses, if one knows how the system may response to one impulse then you know how the system will response to many impulses.

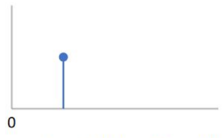
Time-invariant system: If a filter is a *time-invariant system*, when the input is shifted by time step k, the output will also shift by k. In the following example, note how the impulse response for $f[n]$ and $f[n-2]$ are the same but only shifted by 2 in a LTI system.



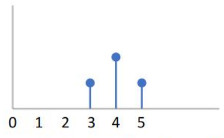
Input $f[n] = \delta[n]$



Output $g[n] = h[n]$

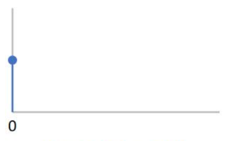


Input $f[n] = \delta[n-2]$

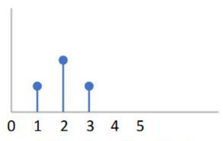


Output $g[n] = h[n-2]$

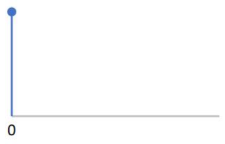
Linear system: If a filter is a *linear system*, when the input is multiplied by k, the output will also by multiplied by k.



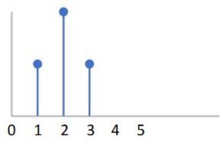
Input $f[n] = \delta[n]$



Output $g[n] = h[n]$

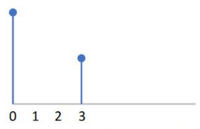


Input $f[n] = 2 \cdot \delta[n]$

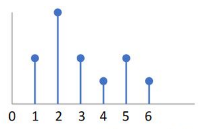


Output $g[n] = 2 \cdot h[n]$

Similarly, if a filter is a linear system, when combining two input signals linearly, their outputs will also be combined linearly.



Input $f[n] = 2 \cdot \delta[n] + \delta[n-3]$



Output $g[n] = 2 \cdot h[n] + h[n-3]$

3.1 Convolution and Linear Time-invariant System (LTI)

For a linear time-invariant system, the impulse response h completely characterises how the LTI system works. The output g can be described as the *convolution* between input f and impulse response h:

$$g[n] = f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[m]h[n-m]$$

Convolutions are usually implemented by:

1. Flip the kernel
2. Multiply the signal with the flipped kernel
3. Sum over the support of the kernel

$$\sum_{m=-\infty}^{\infty} f[m]h[n-m] = \sum_{m=-\infty}^{\infty} f[n+m]h[-m]$$

3.1.1 Propoerties of convolution

- **Commutativity**: $f * h = h * f$
- **Associativity**: $f * (g * h) = (f * g) * h$
- **Distributivity**: $f * (g + h) = (f * g) + (f * h)$
- **Differentiation:** $\frac{d}{dx}(f * g) = \frac{df}{dx} * g = f * \frac{dg}{dx}$

3.1.2 2D convolution

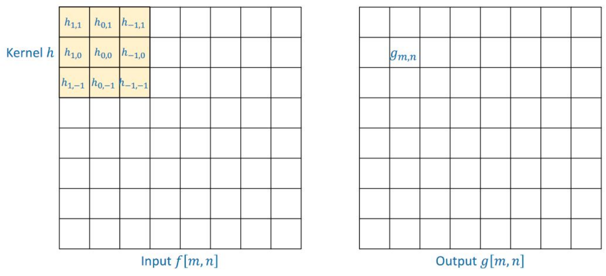
2D convolution is defined mathematically as:

$$g[m,n] = f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j]$$

2D convolutions are usually implemented by:

1. Flip the kernel both horizontally and vertically
2. Multiply the image patch centred at pixel (m,n) with the flipped kernel
3. Sum over the support of the kernel

$$g[m,n] = f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[m+i,n+j]h[-i,-j]$$



$h_{1,1}$ $h_{0,1}$ $h_{-1,1}$
Kernel h
$h_{1,0}$ $h_{0,0}$ $h_{-1,0}$
$h_{1,-1}$ $h_{0,-1}$ $h_{-1,-1}$
$g_{m,n}$
Input $f[m,n]$
Output $g[m,n]$

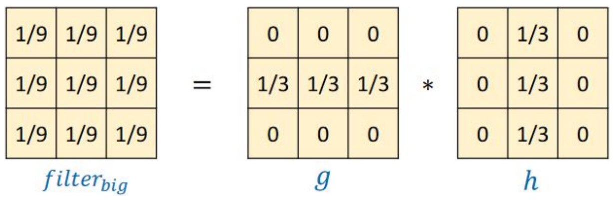
3.1.3 Asspciativity and separable filters

Note the associativity property of convolution:

$$f * (g * h) = (f * g) * h$$

If a big filter can be separated as the convolution of two small filters, such as $g * h$, then it is possible to first convolve f with g then with h.

$$f * filter_{big} = f * (g * h) = (f * g) * h$$



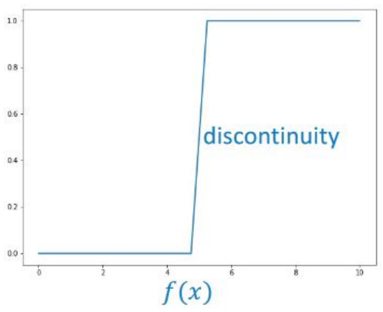
Chapter 4

Edge Detection I

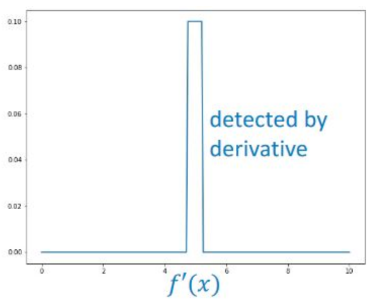
In computer vision, an edge refers to lines where image brightness changes sharply and has discontinuities. Edges capture important properties of the world and are important low-level features for analysing and understanding images.

To detect edges, we note that an image can be regarded as a function of pixel position. As known from mathematics, derivatives characterises the discontinuities of a function which can be used to help detect edges. For example, consider the definition of a continuous function:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



(a) Continuous function



(b) Differentiated function showing detected discontinuity

For a discrete function, the finite difference can be calculated using:

- Forward difference: $f'[x] = f[x+1] - f[x]$
- Backward difference: $f'[x] = f[x] - f[x-1]$
- Central difference: $f'[x] = \frac{f[x+1]-f[x-1]}{2}$

This finite difference can be performed using convolution with kernals such as:

- $h = [1, -1, 0]$
- $h = [0, 1, -1]$
- $h = [1, 0, -1]$

4.1 Convolution

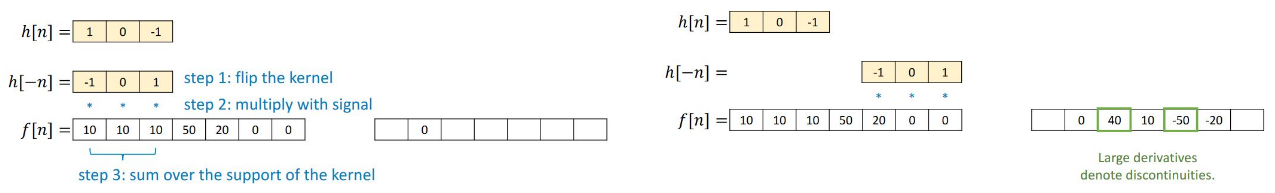
As mentioned earlier, convolution is often implemented:

$$g[n] = f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m] = \sum_{m=-\infty}^{\infty} f[n+m]h[-m]$$

The *central difference*, without accounting for $\frac{1}{2}$, can be defined as:

$$\begin{aligned} f'[x] &= f[x+1] - f[x-1] \\ &= f[x+1] * 1 + f[x] * 0 + f[x-1] * (-1) \\ &= f[x+1] * h[-1] + f[x] * h[0] + f[x-1] * h[1] \end{aligned}$$

The convolution kernal is this defined as: $h = [1, 0, -1]$. The process of convolution with central difference $f'[x] = f[x] * h[x]$ can be seen as shown below:



$h[n] =$ 1 0 -1
$h[-n] =$ -1 0 1
step 1: flip the kernel
step 2: multiply with signal
$f[n] =$ 10 10 10 50 20 0 0
0
step 3: sum over the support of the kernel
$h[n] =$ 1 0 -1
$h[-n] =$ -1 0 1
$f[n] =$ 10 10 10 50 20 0 0
0 40 10 -50 -20
Large derivatives denote discontinuities.

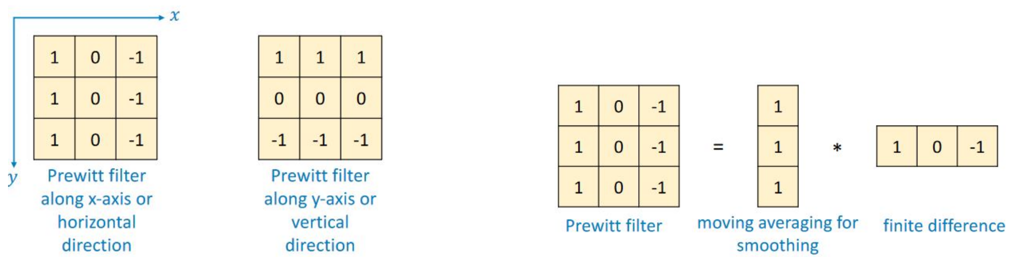
Note the following:

- The kernal $h[n]$ is flipped $h[-n]$ as mentioned earlier on how convolution implemented
- Large derivatives denote discontinuities

The following methods only enable discontinuities to be detected in 1D. Similar filters have to be designed to extend edge detection to a 2D image.

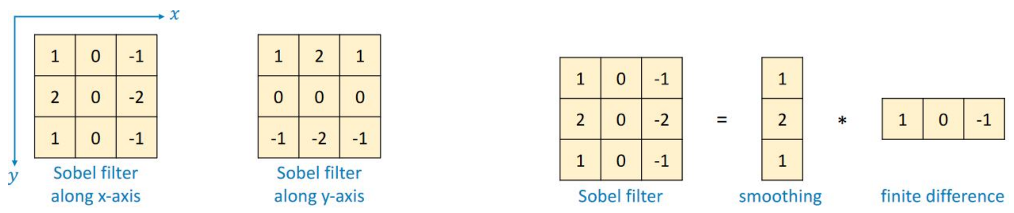
4.2 Edge detection filters

4.2.1 Prewitt filter



x
y
1 0 -1
1 0 -1
1 0 -1
Prewitt filter along x-axis or horizontal direction
1 1 1
0 0 0
-1 -1 -1
Prewitt filter along y-axis or vertical direction
1 0 -1
1 0 -1
1 0 -1
=
1
1
1
*
1 0 -1
Prewitt filter
moving averaging for smoothing
finite difference

4.2.2 Sobel filter



4.3 Image Gradient

There are two outputs from Sobel filters that are combined to describe edges:

- describing discontinuity along x-axis
- describing discontinuity along y-axis

The two outputs are combined to describe the two properties of edges: *Magnitude* and *Orientation* using the following process:

1. Compute the derivatives along x-axis and y-axis:

$$\begin{aligned} g_x &= f * h_x \\ g_y &= f * h_y \end{aligned}$$

2. Compute the magnitude of the gradient:

$$g = \sqrt{g_x^2 + g_y^2}$$

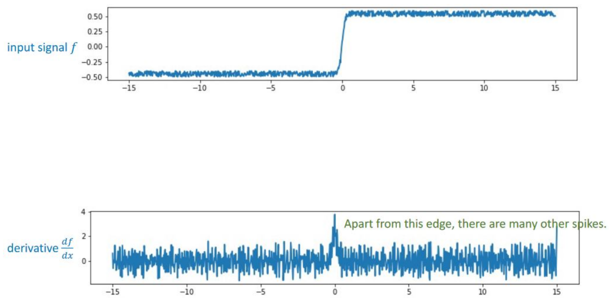
3. Compute the orientation or angle of the gradient:

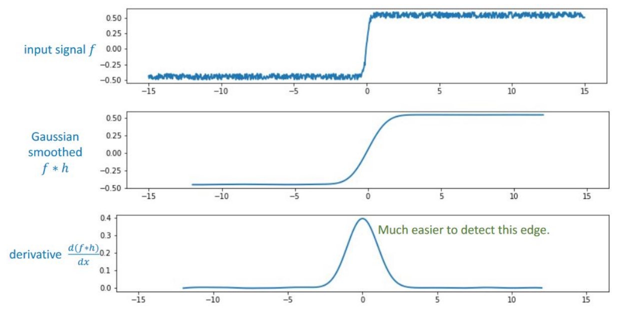
$$\theta = \arctan 2(g_y, g_x)$$

4.4 Smoothing

Derivatives are sensitive to noise thus using a smoothing kernel beforehand to suppress the noise would result in a better result. This can be seen in the Prewitt and Sobel filter which have a smoothing kernel built in.

An alternative is to use the Gaussian kernel for smoothing before calculating the derivatives. The following images show the ease of edge detection after smoothing compared to no smoothing performed beforehand.





4.4.1 Derivative of Gaussian Filter

Derivative of Gaussian filter is an operator that performs Gaussian smoothing before taking the derivative.

$$g[x] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad \text{Gaussian kernel}$$

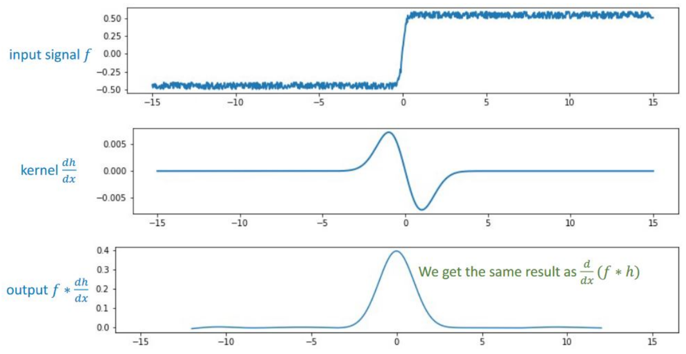
$$\frac{d}{dx}(f * h) \quad \text{Derivative of Gaussian filter}$$

Using the differentiation of convolution:

$$\frac{d}{dx}(f * h) = \frac{df}{dx} * h = f * \frac{dh}{dx}$$

The analytical form for the derivative of Gaussian kernel can be defined as:

$$\frac{d}{dx}(\underset{\text{signal}}{f} * \underset{\text{Gaussian kernel}}{h}) = f * \frac{dh}{dx} = f * \underbrace{\frac{-x}{\sqrt{2\pi}\sigma^3} e^{-\frac{x^2}{2\sigma^2}}}_{\text{derivative of Gaussian kernel}}$$



- 1D Gaussian filter: $h[x] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$
- 2D Gaussian filter: $h[x, y] = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

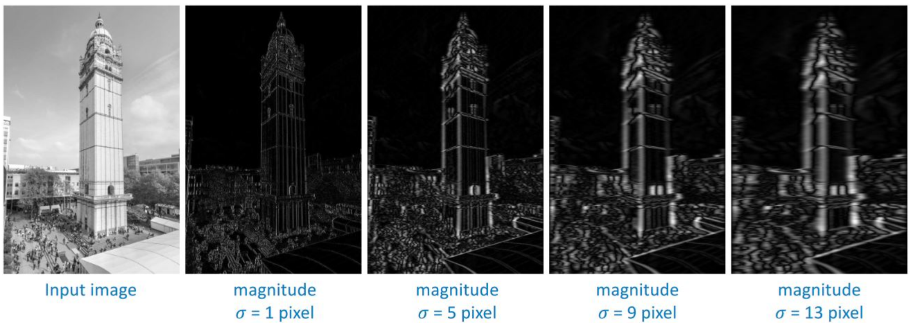
It is a separable filter and equivalent to convolution of two 1D Gaussian filters. This means that 2D Gaussian smoothing can be accelerated using separable filtering.

The process of a Gaussian Derivative Filter is defined as:

1. Smooth the input image with a 2D Gaussian filter
2. Take the derivative along x-axis and y-axis
3. Calculate the magnitude and orientation

Parameter σ in derivative of Gaussian

Large σ value suppresses noise and results smoother derivative. Different σ values finds edges at different scales.



Chapter 5

Edge Detection II

As mentioned earlier, edge detection is a fundamental problem in image processing and computer vision that aims to identify points in an image that are edges i.e. where discontinuities occur.

There are two solutions to the problem:

- **Canny approach**: Know explicitly the criteria of edge points, can design computational procedures to satisfy these
- **Machine Learning**: Collect data which represent what to achieve e.g. manually drawn edge maps and train a machine learning model to map from images to final outputs.

5.1 Canny Edge Detection

John Canny defines a good edge detector:

- **Good detection**: Low probability of failing to mark real edge points and falsely marking non-edge points
- **Good localisation**: Points marked as edge points by operator should be as close as possible to the centre of the true edge
- **Single response**: Only one response to a single edge

Canny edge detection algorithm is summarised into the following:

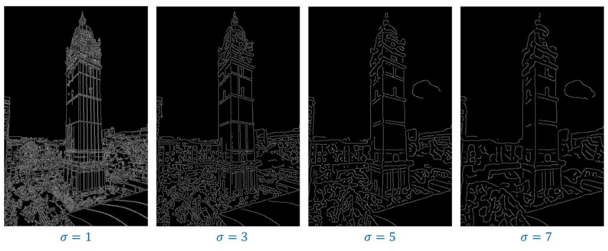
1. Perform Gaussian filtering to suppress noise
2. Calculate the gradient magnitude and direction
3. Apply *non-maximum suppression* (NMS) to get a single response for each edge
4. Perform *hysteresis thresholding* to find potential edges

5.1.1 Gaussian filtering

Gaussian filtering is performed to smooth image and suppress noise. The effect of smoothing can be adjusted by the parameter σ in the Gaussian kernel.

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

The effect of Gaussian smoothing parameter σ on edge detection is:



The choice of σ depends on desired edge detection behavior:

- Large σ: Detects large-scale edges and smooths out fine details.

- Small σ: Detects fine features.

5.1.2 Image gradient

The image gradient is calculated after Gaussian filtering. Common filters are *Prewitt* and *Sobel*.

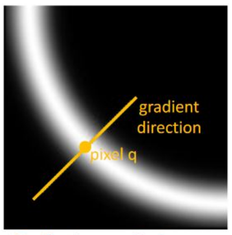
5.1.3 Non-maximum suppression (NSM)

The aim of NMS is to get a single response for an edge. The main concept of NMS: Edge occurs where the gradient magnitude reaches the maximum.

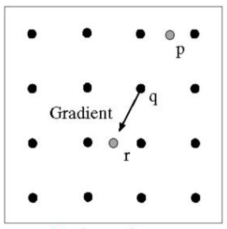
The NMS process is defined:

1. Check whether a pixel q is a local maximum along the gradient direction
2. Move to pixel r and compare the gradient magnitudes between q and r
3. Move to pixel p and compare the gradient magnitudes between q and p
4. If pixel q is the local maximum, it is an edge point and suppress the other pixels, i.e. non-maximum pixels

$$M(x,y) = \begin{cases} M(x,y) & \text{if local maximum} \\ 0 & \text{otherwise} \end{cases}$$



Gradient magnitude map



Pixels on the map

In implementation, we need to perform image interpolation for pixels p and r, if they are not located on the pixel lattice using nearest neighbour or linear interpolation.

5.1.4 Thresholding

Thresholding is performed to ensure edges have high magnitudes. When performing NMS for all pixels on the gradient magnitude map, many pixels may have low magnitude even though it is a local maxima.

The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity $I_{i,j}$ is less than some fixed constant T (that is $I_{i,j} < T$), or a white pixel if the image intensity is greater than that constant. This is the method of segmenting images.

$$\text{binary}(x,y) = \begin{cases} 1 & \text{if} I(x,y) \leq t \\ 0 & \text{otherwise} \end{cases}$$

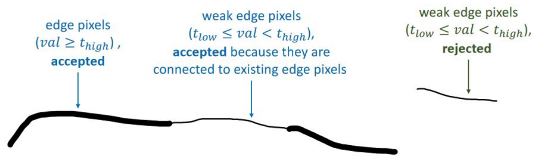
From a grayscale image, thresholding can be used to create binary images.

A common alternative is **Hysteresis thresholding** which is done by the canny edge detector.

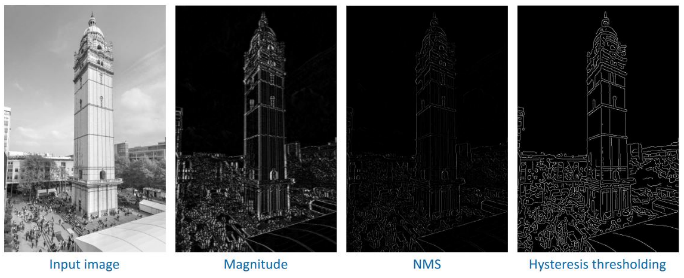
Hysteresis Thresholding

Hysteresis thresholding defines two thresholds t_{low} and t_{high} for edge detection:

- If a pixel's gradient magnitude is $\leq t_{high}$, pixel is accepted as an edge pixel.
- If a pixel's gradient magnitude is $< t_{low}$, pixel is rejected.
- If it is in between, it is a weak edge pixel i.e. may be an edge or may not be.
  - Check neighbouring pixels, pixel is accepted if it is connected to an edge pixel


edge pixels ($val \geq t_{high}$), accepted
weak edge pixels ($t_{low} \leq val < t_{high}$), accepted because they are connected to existing edge pixels
weak edge pixels ($t_{low} \leq val < t_{high}$), rejected

The result can be seen in the image:



5.1.5 Canny Edge Detection and its goals

John Canny's initial goal is achieved by:

- **Good detection**:
  - Gaussian smoothing to suppress noise (reducing false positives)
  - Hysteresis thresholding to find weak edges (reducing false negatives)
- **Good localisation**:
  - Finding the location of edges using non-maximum suppression, which is based on gradient magnitude and direction
- **Single response**:
  - Non-maximum suppression (NMS)

5.2 Improving Edge Detection

The Canny Edge Detection approach is designed by the following approach:

1. Think of what criteria edges need to satisfy
2. Carefully design the *image features* (e.g. Gaussian filtering + image gradient) and *procedures* (e.g. NMS, hysteresis thresholding) to achieve that criteria.

There are many improvements that can be made in improving edge detection accuracy such as:

- Utilising richer features, e.g. colour, texture
- Integrating multi-scale features
- Enforcing smoothness
- Machine learning, e.g. learning the mapping from image to edge directly from paired data

5.2.1 Learning-based Edge detection

Develop a machine learning model for edge detection based on:

- Assuming that paired data (images x + manually defined edge maps y) are available
- Common problem is how to find a model f that maps x to y, i.e. $y = f(x|\theta)$, where θ denotes the model parameters

For example, consider this model:

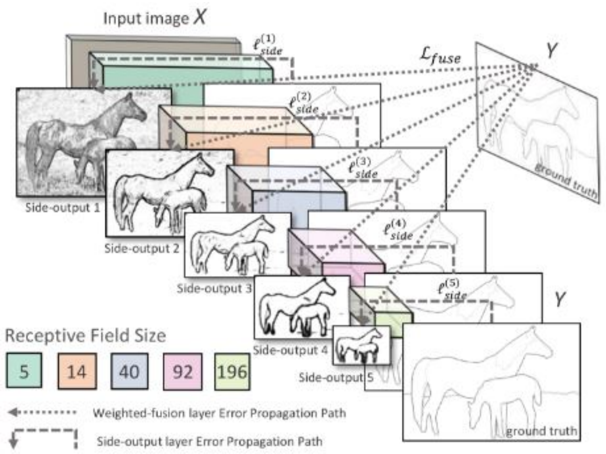


Figure 5.1: The colourful blocks are a machine learning model (e.g. neural network) that maps an input image into an edge map.

This machine learning model integrates features from multiple scales, fusing fine-scale edges with coarse-scale edges to form the final output. This is vastly different compared to the Canny edge detector which only uses a single scale for edge detection, controlled by the Gaussian parameter σ.

Chapter 6

Hough Transform

With edge detection,a binary edge map which contains a lot of edge points is obtained. If these edge points belong to a line, how is a parametric representation of the line obtained?

Hough transform is a transform from image space to parameter space e.g. from an edge map to the two parameters of a line. The result is a parametric model, given the input edge points. The aim of this transform is to detect shapes and objects by a voting procedure.

Advantage:

- It detects multiple instances.
- Robust to image noise.
- Robust to occlusion.

Limitations:

- The computational complexity can be high. For each edge point, we need to vote to a 2D or even 3D parameter space.
- Need to carefully set some parameters, such as the parameters for the edge detector, the threshold for the accumulator or the range of circle radius

6.1 Line parameterisation

A line can be represented in several forms:

- Slope intercept form:
  $$y = mx + b$$
  where m is the *gradient* and b is the *y-axis intercept*
- Double intercept form:
  $$\frac{x}{a} + \frac{y}{b} = 1$$
  where a is the *x-intercept* and b is the *y-intercept*
- Normal form:
  $$x\cos(\theta) + y\sin(\theta) = \rho$$
  where θ is the *angle* and ρ is the *distance from origin*

6.2 Model fitting

A common alternative to Hough transform is to fit a line model onto edge points:

- Suppose there is set of points $(x_1, y_1), (x_2, y_2), \ldots$ to fit a line model $y = mx + c$
- m and b can be estimated by minimising the fitting error:
  $$\min_{m,b} \sum_i [y_i - (mx_i + b)]^2$$
  where:

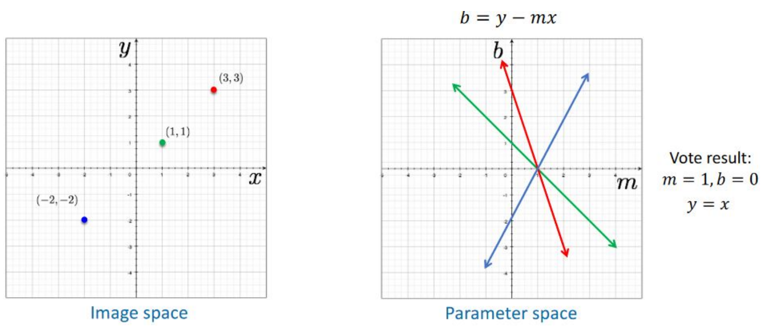
  - y_i - Real y of a point
  - $mx_i + b$ - $\hat{y}$ estimated by our line model

6.3 Hough Transform

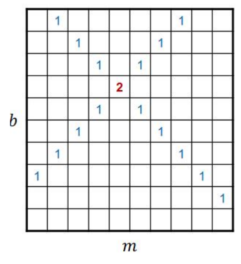
Define the slope intercept form for a line model:

$$y = mx + b \rightarrow b = y - mx$$

Each edge points in the image space $(x_1, y_1), (x_2, y_2), (x_3, y_3), ...$ votes for a line model in the parameter space e.g. the first point will vote for $b = y_1 - mx_1$



With this, the parameter space is divided into 2D bins where each point increments the vote by 1 in one of the bins as seen below:



The problem with the slope intercept form is that the parameter space is too large where $m \in [-\infty, \infty]$ and $b \in [-\infty, \infty]$. Alot of bins would be needed.

The solution is to use the *normal form* $x\cos(\theta) + y\sin(\theta) = \rho$ where $\theta \in [0, \pi)$ and $\rho \in [-\infty, \infty]$. Note that although *rho* is infinite, it is technically limited by the image size.

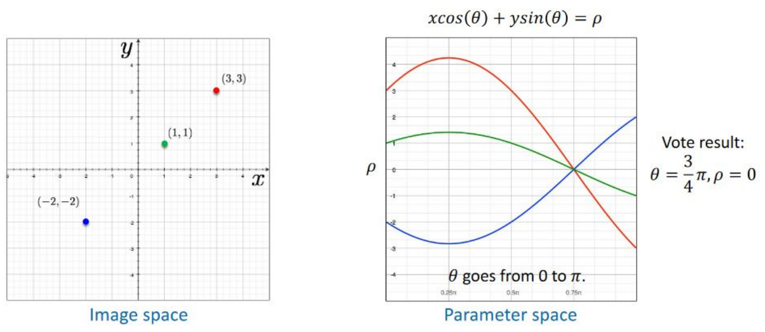


Figure 6.1: Conversion from *slope intercept* to *normal form*

6.3.1 Line detection by Hough transform

The algorithm is defined as:

1. Initialise bins $H(\rho, \theta)$ to all zeros
2. For each edge point (x, y):
   (a) For θ from 0 to π:

    - Calculate $\rho = x\cos(\theta) + y\sin(\theta)$
    - Accumulate $H(\rho, \theta) = H(\rho, \theta) + 1$

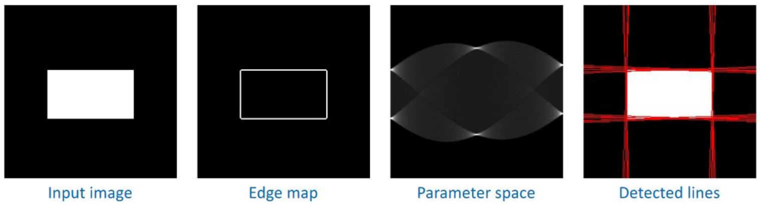
3. Find (ρ, θ) where $H(\rho, \theta)$ is a *local maximum* and larger than a *threshold*
    - Local maximum is chosen as it is similar to NMS in edge detection
    - Thresholding is required as a few random points would not lead to a line being detected
4. Detected lines are defined by $\rho = x\cos(\theta) + y\sin(\theta)$

This approach is different from model fitting where (m, b) is defined by:

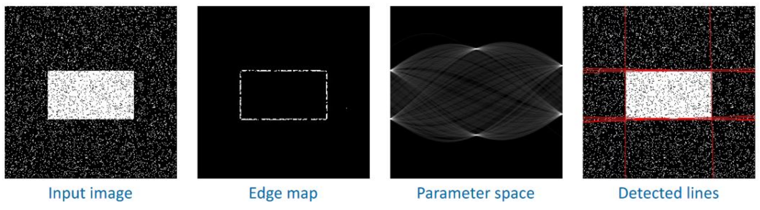
$$\min_{m,b} \sum_i [y_i - (mx_i + b)]^2$$

where only one line will be detected.

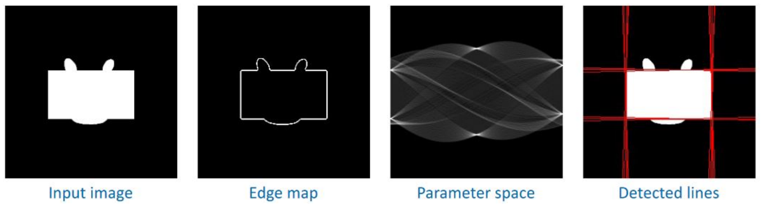
Shapes: Unlike model fitting, Hough Transform can simultaneously detect multiple lines as long as the lines are local maxima above a threshold. This allows Hough transform to detect other shapes such as Ellipses, Planes in 3D space etc



Noise: Hough transform is robust to noise. An edge map is often generated after image smoothing and broken edge points can still vote and contribute to line detection.



Occlusion: Hough transform is robust to object occlusion e.g. rectangle covered by a bunny. The remaining edge points vote and contribute to line detection.

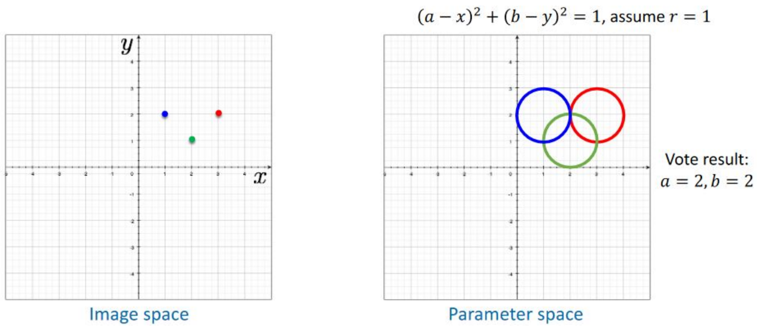


6.4 Circle detection

Hough Transform can be used to detect other shapes such as circles. The circle canbe parameterised as:

$$(x - a)^2 + (y - b)^2 = r^2$$

where the image space (x, y) is transformed into the parameter space (a, b, r).



If the radius r is known, then for each edge point (x, y), there is only the need to vote for possible values of (a, b) i.e. still in parameter space $H(a, b)$.

6.5 Circle detection by Hough Transform

The algorithm is defined as:

1. Initialise the bins $H(a, b, r)$ to all zeros
2. For each possible radius $r \in [r_{min}, r_{max}]$
   - For each edge point (x, y)
     - Let θ to be gradient direction, or opposite gradient direction
     - Calculate $a = x - r\cos(\theta), b = y - r\sin(\theta)$
     - Accumulate $H(a, b, r) = H(a, b, r) + 1$
3. Find (a, b, r) where $H(a, b, r)$ is a local maximum and larger than a threshold
4. Detected circles are defined: $x = a + r\cos(\theta), y = b + r\sin(\theta)$

6.6 Generalised Hough Transform Idea

Weights: When voting in the parameter space, it is possible to add *weights*. Instead of using equal vote for each edge point, the vote can be weighted by the gradient magnitude so stronger edge points get higher weights.

Spaces: There are two spaces, the image space and the parameter space i.e. Hough space.

- If the shape can be described by some parameters using an analytical equation, we use this equation to perform voting to the parameter space
- If it is not a simple shape without an analytical equation, as long as we have a model to describe it, we can still vote

Chapter 7

Interest Point Detection I

Interest points (keypoints, landmarks, low-level features) are image points useful for subsequent image processing and analysis such as Image classification, Image matching, Image retrieval etc. Interest points are mostly corners or blobs, where local image structures are rich.

7.1 Facial Image Analysis

A common use of interest points where interest points (landmarks) can be defined to represent a face.

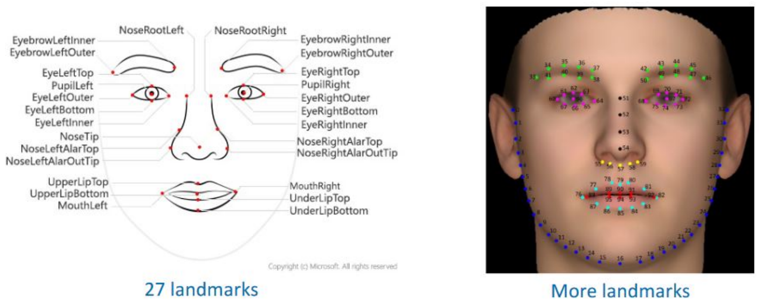


Figure 7.1: Examples of landmarks representing a human face

These interest points are used to detect faces and the locations of interest points can be used for mood analysis, augmented reality etc.

7.2 Image matching (Image alignment, Image registration)

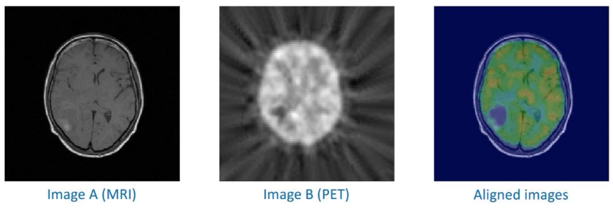
Image matching can be used to stitch images together by overlaying the matching parts. The correspondance i.e. similarity between two images can be found using three methods:

- Pixels: Using *all* information

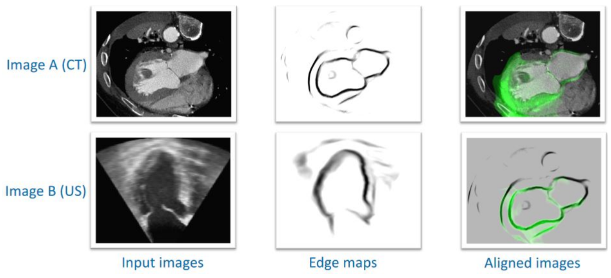
  Optimising a similarity metric based on all image pixels:

$$\max_T Similarity(Image_A, Image_B(T))$$

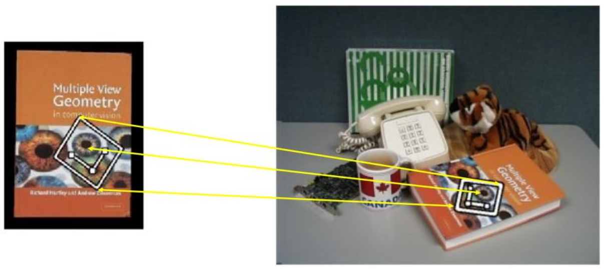
  where T denotes spatial transformation



- Edges: Using *some* information



- Interest points: Using *very little but important* information



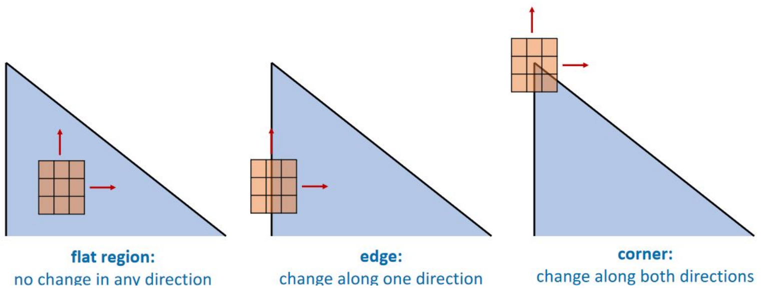
7.3 Interest Point Detection

Interest points are generally corners i.e. intersections of edges and the Harris Corner Detector is an algorithm that could detect it.

7.3.1 Harris Corner Detection

Edge detection works by calculating the magnitude of image gradient at each pixel. Edge pixels correspond to high magnitude of gradient.

The difference in corner detection is that edge detection methods could not differentiation between edge and corner pixels. But, by using a small window, the difference between edge pixels and corner pixels could be detected.


flat region:
no change in any direction
edge:
change along one direction
corner:
change along both directions

By shifting the window and calculating the change of intensities for the window:

- **Edges**: Change of intensity along just one direction
- **Corner**: Change of intensity along both direction

Mathematics can be used to describe the change of pixel intensities in a given window W. Given a window shifting by $[u, v]$, the change intensities is given by: where the window function $w(x, y)$ can be a Step or Gaussian function

$$\underbrace{E(u,v)}_{\text{sum of squared difference (SSD)}} = \sum_{(x,y)\in W} \underbrace{w(x,y)}_{\text{window function}}[\underbrace{I(x+u,y+v)}_{\text{intensity in the shifted window}} - \underbrace{I(x,y)}_{\text{original intensity}}]^2$$

7.3.2 Change of Intensities

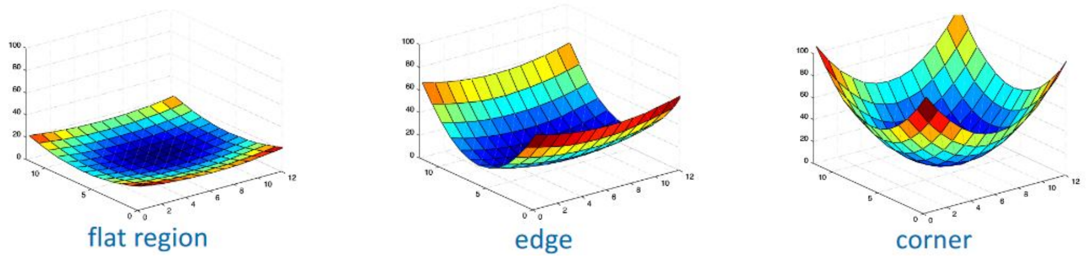
A small shift $[u, v]$ is defined with the following bilinear approximation

$$E(u,v) \approx [u,v].M.\begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix derived from the window function and product of image derivatives

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

Based on $E(u,v) \approx [u,v].M.\begin{bmatrix} u \\ v \end{bmatrix}$:



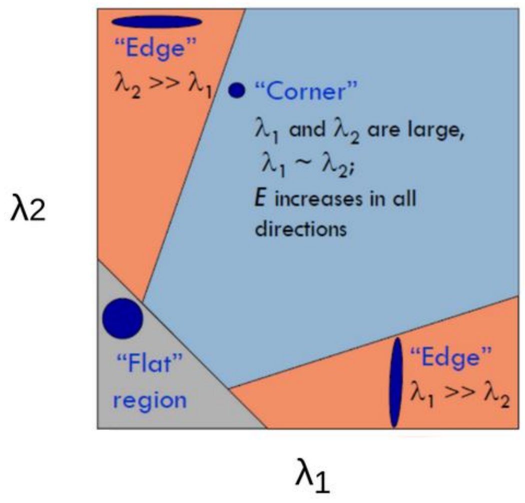
- $M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$: Flat region
- $M = \begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix}$: Edge - A large change if shifting window along u but little change along v
- $M = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$: Corner - A large change whichever way window shift occurs

If M is more complicated, the matrix M can be made simpler by performing eigen decomposition by using the face that eigenvectors are orthogonal to each other. Since M is a real symmetrical matrix, M can be decomposed by

$$M = \underbrace{P}_{\text{each column is a eigenvector}} \underbrace{\Lambda}_{\text{diagonal matrix with eigenvalues } \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}} \underbrace{P^T}_{\text{each row is a eigenvector}}$$

7.3.3 Intepreting Eigenvalues

Getting the two eigen values of hessian matrix for all points will show the category of that point.



Corners are defined by eigenvalues are defined variously such as $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$, $R = \min(\lambda_1, \lambda_2)$, $R = \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2+\in}$, etc

7.3.4 Cornerness

Since only the values $\lambda_1\lambda_2$ and $\lambda_1 + \lambda_2$, there is no need to perform eigen-decomposition. Instead the properties of matrix determinant and trace of $M = P\Lambda P^T$ defined as:

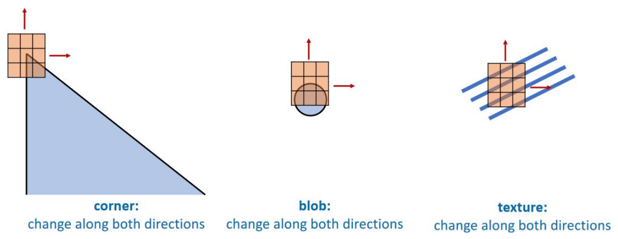
$$R = \det(M) - k(\text{trace}(M))^2$$

can be used:

$$\det(M) = \det(P\Lambda P^T) = \det(\Lambda) = \lambda_1\lambda_2$$
$$\text{trace}(M) = \text{trace}(P\Lambda P^T) = \text{trace}(\Lambda) = \lambda_1 + \lambda_2$$

7.4 Harris Detector

This detector finds strong responses at corners, blobs and textures.



corner:
change along both directions
blob:
change along both directions
texture:
change along both directions

Algorithm:

1. Compute x and y derivatives of an image

$$I_x = G_x * I$$
$$I_y = G_y * I$$

2. At each pixel, compute the matrix M

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

3. Calculate the detector response
$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$
4. Detect interest points which are local maxima and whose response R are above a threshold.

7.4.1 Properties

Invariant to rotation: Harris detector is rotation-invariant i.e. if a corner is rotated, you can still get the same change of intensities when you shift the window along a rotated direction.

Not invariant to scale: Harris corner detector is not invariant to scale. However, we can apply Harris detector at multiple scales and find the response at the most suitable scale.

Multi-scale images are obtained through:

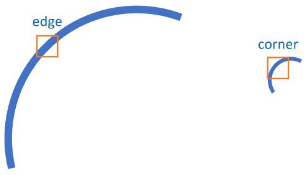
- Gaussian smoothing with different σ
- Sampling with different spatial resolutions

Chapter 8

Interest Point Detection II

8.1 Scale

The Harris detector is not invariant to scale as seen in this following example:



Different scales are required as images convolved with Gaussian kernels of different σ provide information at different scale as seen below:

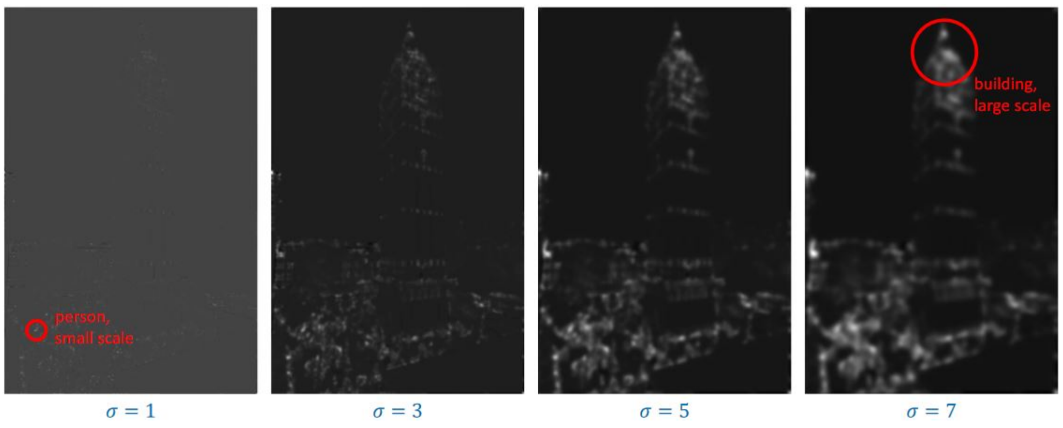


Figure 8.1: Harris detector response when different Gaussian kernels are used for calculating image derivatives

How the scale is determined at each pixel is the problem. The goal is to be able to choose a scale such that the region that looks most like a corner **results in a high Harris detector response**.

8.1.1 Response at different scales

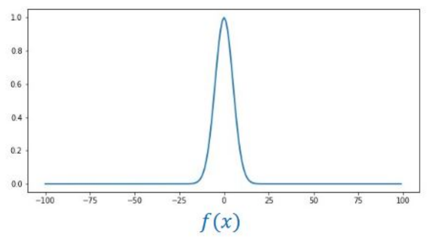
The Harris detector response is calculated using the eigenvalues of M

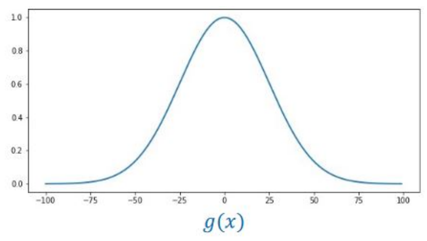
$$M = \sum_{x,y} w(x,y) = \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y^2(\sigma) \end{bmatrix}$$

which are in turn determined by the derivatives $I_x(\sigma)$ and $I_y(\sigma)$. The derivatives are inversely proportional to scale σ i.e. the larger the scale σ, the smaller the derivatives magnitude.

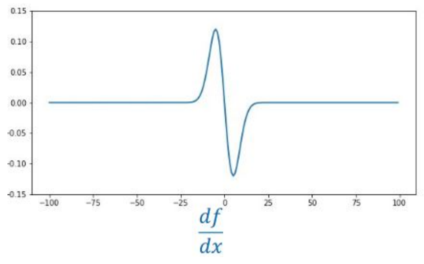
8.1.2 Derivatives at different scales

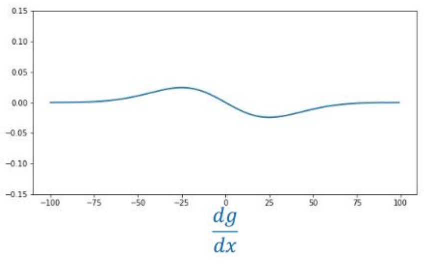
Consider two signals $f(x)$ and $g(x)$ which are related by $f(x) = g(sx)$ i.e. only differing by scale s and have the same peak magnitude. This can happen, for example, when a picture of the same object is taken with different zoom factors.





The peak magnitudes of the calculated derivatives differ as well





The relationship between the two derivatives are defined as:

$$\frac{df}{dx} = s.\frac{dg}{dx}|_{sx}$$

This equation shows that to make the derivative magnitude comparable across scales, multiply the derivative by its scale s. The same object will give same magnitude of response, regardless of the zoom factor.

8.1.3 Scale adapted Harris detector

The scale adapted Harris detector is defined as

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} \sigma^2 I_x^2(\sigma) & \sigma^2 I_x(\sigma)I_y(\sigma) \\ \sigma^2 I_x(\sigma)I_y(\sigma) & \sigma^2 I_y^2(\sigma) \end{bmatrix} = \sum_{x,y} w(x,y)\sigma^2 \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y^2(\sigma) \end{bmatrix}$$

At each pixel, determine the scale which gives the largest detector response (e.g. at this scale, it looks most like a corner)

The *scale adapted Harris detector* composes of two steps:

- Calculates the scale adapted detector response for a series of σ, from small-scale to large-scale
- Perform interest point detection and look for local maxima both across space and across scale

The algorithm, **for each** σ, is defined as:

1. Perform Gaussian smoothing with σ
2. Calculate the x and y derivatives of the smoothed image $I_x(\sigma)$ and $I_y(\sigma)$
3. At each pixel, compute the matrix M

$$M = \sum_{x,y} w(x,y)\sigma^2 \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y^2(\sigma) \end{bmatrix}$$

4. Calculate the detector response

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

5. Detect interest points which are local maxima across both scale and space and whose response R are **above a threshold**.

8.2 Other interest point detectors

8.2.1 Laplacian of Gaussian (LoG)

The Laplacian is the sum of second derivatives. For 2D image, it is

$$\Delta f = \bigtriangledown^2 f = \frac{\sigma^2 f}{\sigma x^2} + \frac{\sigma^2 f}{\sigma y^2}$$

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 1 & -2 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \underbrace{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}}_{\text{Laplacian filter}}$$

Note that the second derivative is more sensitive to noise. For this reason, the image is smoothed using a Gaussian kernel before calculating the Laplacian.

$$\text{Laplacian: } \Delta f = \frac{\sigma^2 f}{\sigma x^2} + \frac{\sigma^2 f}{\sigma^2 y}$$

$$\text{LOG: } \Delta(f * h) = \frac{\sigma^2 (f * h)}{\sigma x^2} + \frac{\sigma^2 (f * h)}{\sigma y^2}$$

The analytical form for the LoG kernel is defined as:

$$\text{LoG: } \Delta(f * h) = \frac{\sigma^2 (f * h)}{\sigma x^2} + \frac{\sigma^2 (f * h)}{\sigma y^2} = f * \left(\frac{\sigma^2 h}{\sigma x^2} + \frac{\sigma^2 h}{\sigma y^2} \right)$$

Noting that 2D Gaussian is defined as $h(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$, the LoG can be derived

$$\frac{\sigma^2 h}{\sigma x^2} + \frac{\sigma^2 h}{\sigma y^2} = -\frac{1}{\pi\sigma^4} \left(1 - \frac{x^2 + y^2}{2\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Similarly to Harris detector, if we want to determine the optimal scale at each pixel, we need to make sure the LoG response is comparable between scales.

The LoG response at scale σ is defined as:

$$LoG(x, y, \sigma) = I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma)$$

The normalised LoG response at scale σ is defined as:

$$LoG_{norm}(x, y, \sigma) = \sigma^2 (I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$$

8.2.2 Difference of Gaussian (DoG)

Difference of Gaussians (DoG) filter is defined as

$$DoG(x, y, \sigma) = I * G(k \times \sigma) - I * G(\sigma)$$

where k is a arbitary constant e.g. $k = \sqrt{2}$

DoG approximates the normalised Laplacian of Gaussian (LoG)

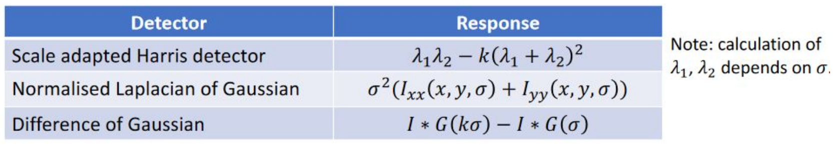
$$LoG_{norm}(x, y, \sigma) = \sigma^2 (I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$$

DoG is a good approximation to the normalised Laplacian of Gaussian (LoG). It provides some convenience in calculating the response across different scales.

8.2.3 Summary of interest point detectors

- Scale-variant
  - Harris detector
- Scale-invariant
  - Scale adapted Harris detector
  - Normalised Laplacian of Gaussian
  - Difference of Gaussian

  The scale-invariant detectors follow similar procedures:
  - Calculate the detector response across scales
  - Find local extrema both across scale and across space



Detector	Response
Scale adapted Harris detector	$\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$
Normalised Laplacian of Gaussian	$\sigma^2(I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$
Difference of Gaussian	$I * G(k\sigma) - I * G(\sigma)$

Note: calculation of λ_1, λ_2 depends on σ.

Chapter 9

Feature Description I

Feature description describes the feature for each point so that the points can be compared and matched.

9.1 Descriptors

There are several types of descriptors:

- Pixel intensity
- Patch intensities
- Gradient orientation
- Histogram
- SIFT descriptor

9.1.1 Pixel intensity

Describes features using intensity of a single pixel.

Problems:

- Sensitive to absolute intensity value. Intensity of the same point will change under different illumination.
- Not discriminative. The grayscale intensity ranges from 0 to 255. There are thousands of pixels with exactly the same intensity.

9.1.2 Patch intensities

Better than single pixel intensity and represents the local pattern. This performs well if the images are of similar intensities and roughly aligned.

Problems:

- Sensitive to absolute intensity value
- Not rotation-invariant

9.1.3 Gradient orientation

This type is robust to changes in intensity.

Problem:

- Not rotation invariant

9.1.4 Histogram

The intensity histogram of a patch. This is robust ot rotation and scaling.

Problem:

- Sensitive to intensity changes

9.2 SIFT algorithm

Combination of the gradient orientation (robust to intensity changes) and histogram (robust to rotation and scaling). The algorithm is used for detecting and describing local features in images.

This algorithm transforms an image into a large set of interest points, each of which is described by a feature vector that is invariant to translation, scaling and rotation.

The SIFT algorithm is defined into the following steps:

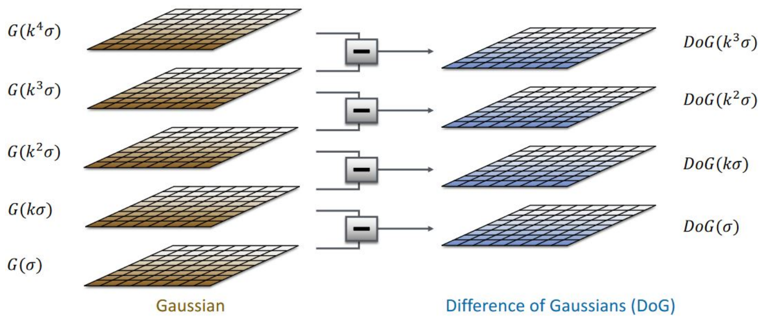
1. Detection of scale-space extrema [Detection]
2. Keypoint localisation
3. Orientation assignment [Description]
4. Keypoint descriptor

9.2.1 Detection of scale-space extrema

Search across scales and pixel locations, looking for interest points. Difference of Gaussian (DoG) filter is used to identity potential interest points.

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

The difference of Gaussian for different scales σ is calculated from fine to coarse e.g. $\sigma, \sqrt{2}\sigma, 2\sigma, \ldots$. This is implemented as first calculating a stack of Gaussian smoothed images then calculating their differences.



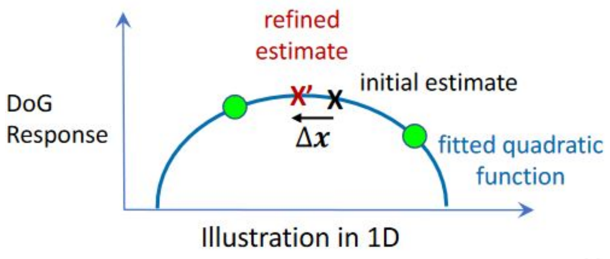
9.2.2 Keypoint localisation

SIFT can be improved to refine the localisation and scale by fitting a model onto nearby data which can improve the accuracy for image matching:

- A quadratic function is fitted to the DoG response of neighbouring pixels
- Location and scale of extremum for this quadratic function can be estimated

Denote the DoG response by $D(x, y, \sigma)$ or $D(x)$ where $x = (x, y, \sigma)^T$ is a 3D vector containing both location and scale. The refined estimate is defined as:

$$\Delta x = -\frac{\partial^2 D}{\partial x^2}^{-1} \frac{\partial D}{\partial x}$$



9.2.3 Orientation assignment

Assigns a consistent orientation to each keypoint, based on local image properties. As a result, the feature descriptor can be represented relative to this orientation and achieve invariance to image rotation.

The key idea being that if the orientation of the image, it is possible to sample in a rotated coordinate system before calculating features. The dominant orientation θ for a neighbourhood of a keypoint found by:

- Calculate the gradient orientation for pixels in this neighbourhood
- An orientation histogram with 36 bins covering 360 degrees is created
- Each pixel votes for an orientation bin, weighted by the gradient magnitude
- Keypoint will be assigned an orientation that is the peak of the histogram

Local image descriptor

With sample points, it is possible to describe the local image content through:

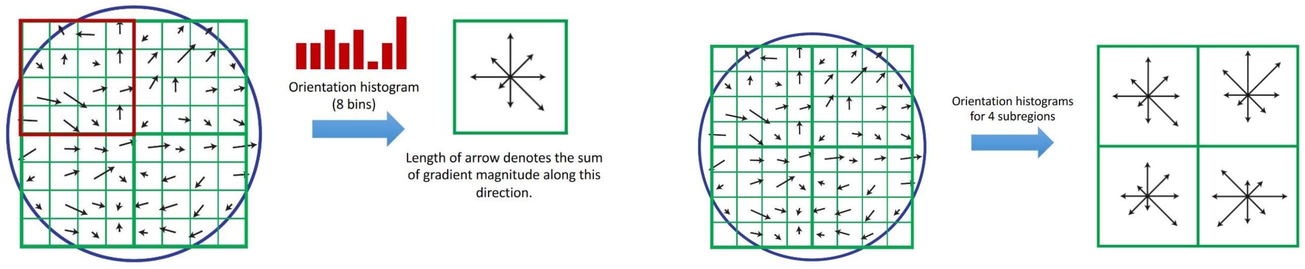
- Pixel intensities
- Gradient orientations
- Histograms

SIFT uses the histogram of gradient orientations:

- Calculate the gradient magnitudes and orientations for these sampling points. Orientations are calculated relative to the dominant orientation θ
- Create a histogram of gradient orientations

In practice, *subregions* are used and each subregion has 4×4 samples:

- Each subregion will have one orientation histogram
- Obtain a number of histograms, which together describe what it looks like around the keypoint


Orientation histogram
(8 bins)
Length of arrow denotes the sum of gradient magnitude along this direction.
Orientation histograms for 4 subregions

This descriptor is robust to rotation, scaling and change of illuminations:

- Consideration of dominant orientation
- A sampling window proportional to scale
- Use of gradient orientations for feature description
- Use of histograms

9.2.4 Keypoint matching

Keypoints between two images are matched by identifying the nearest neighbours:

- For each keypoint in image A, identify its nearest neighbours in the database of keypoints for image B
- The distance is defined by the Euclidean distance of SIFT descriptors

The exact neighbours do not need to be found. There are faster algorithms for finding approximate nearest neighbours.

After nearest neighbour search, a keypoint is found (x, y) in image A corresponds to another keypoint (u, v) in image B, the images are related by an affine transformation:

$$\begin{bmatrix}u\\v\end{bmatrix} = \underbrace{\begin{bmatrix}m_1 & m_2\\ m_3 & m_4\end{bmatrix}}_{\text{rotation, scaling ...}} \cdot \begin{bmatrix}x\\y\end{bmatrix} + \underbrace{\begin{bmatrix}t_x\\t_y\end{bmatrix}}_{\text{translation}}$$

For many pairs of corresponding keypoints, the equation can be written as a linear system

$$Am = b$$

where m - unknown affine parameters

The least-square solution to the linear system is defined by the Moore-Penrose inverse:

$$m = (A^T A)^{-1} A^T b$$

which minimises the squared difference $||Am - b||^2$

Once the affine parameters are solved, we know the spatial transformation between the two images.

9.3 RANSAC

The squared difference $||Am - b||^2$ is sensitive to outliers. To improve the robustness of matching, we can employ methods which consider outliers in model fitting, such as Random Sample Consensus (RANSAC).

For each iteration:

1. Select random sample points
2. Fit a model
3. Based on the model, check how many points are inliers
4. Terminate after a certain number of iterations or enough inliners have been found

Chapter 10

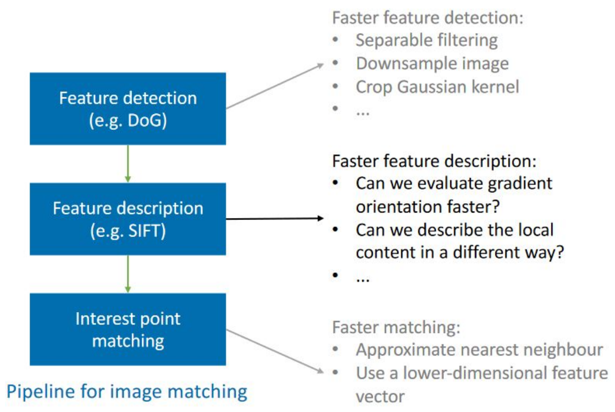
Feature Description II

As mentioned earlier the SIFT descriptor at each interest point is formed by concatenating the histograms of gradient orientations for 16 subregions. However, calculating the gradient magnitudes and orientations is computationally slow.

10.1 Faster feature description

SIFT can be improved using two approaches:

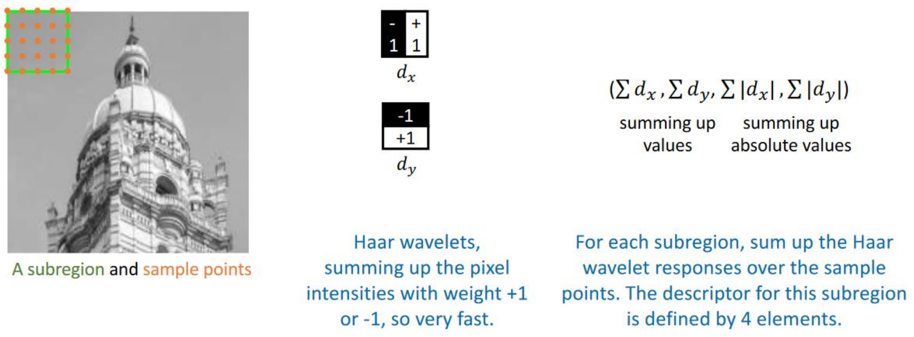
- **Hardware**: Use faster hardware e.g. FPGA instead of CPU
- **Software**: Improving software or algorithm e.g. decomposing pipeline into several steps, and reconsider and evaluate each step


Feature detection (e.g. DoG)
Feature description (e.g. SIFT)
Interest point matching
Pipeline for image matching
Faster feature detection:
Separable filtering
Downsample image
Crop Gaussian kernel
...
Faster feature description:
Can we evaluate gradient orientation faster?
Can we describe the local content in a different way?
...
Faster matching:
Approximate nearest neighbour
Use a lower-dimensional feature vector

10.1.1 Speeded-Up Robust Features (SURF)

SIFT uses histograms of gradient orientations for describing local features. To accelerate computation, there is no need to calculate gradients for arbitrary orientations. SURF only calculates the gradients along with horizontal and vertical directions using the Haar wavelets.

SURF applies very simple filters d_x and d_y (Haar wavelets) onto the sample points to calculate gradients along x and y.


A subregion and sample points
-1 +1
d_x
-1
+1
d_y
Haar wavelets, summing up the pixel intensities with weight +1 or -1, so very fast.
$(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$
summing up values
summing up absolute values
For each subregion, sum up the Haar wavelet responses over the sample points. The descriptor for this subregion is defined by 4 elements.

The dimensions of the SURF descriptor:

- 4×4 subregions
- 4 elements per subregion
- Feature descriptor: 16 subregions $\times$ 4 elements = 64

10.1.2 Binary Robust Independent Elementary Features (BRIEF)

When we use Haar wavelets in SURF, we compare a local region to another and calculate the difference, which is a floating point number.

In BRIEF, we compare a point p to another point q and get a binary value as output:

$$\tau(p,q) = \begin{cases} 1 & \text{if} I(p) < I(q) \\ 0 & \text{otherwise} \end{cases}$$

BRIEF:

1. Randomly sample n_d pairs of points for binary tests
   - The random pattern is determined only once. Afterwards, the same pattern is always applied to all interest points.
2. If $n_d = 256$, perform 256 tests and each test gives 1 bit (0 or 1)
3. BRIEF descriptor is described as a n_d-dimensional bitstring e.g. $[1, 0, 0, ...]$

Assuming $n_d = 256$, the BRIEF descriptor is 256-bit long i.e. 32 bytes long. This is shorter than SIFT (512 bytes) and SURF (256 bytes). This makes BRIEF faster since:

- Only compare two numbers, without calculating the gradient orientation (SIFT) or intensity difference (SURF)
- When comparing two BRIEF descriptors, it is also faster as we do not need to calculate the Euclidean distance
- BRIEF ignores rotation- and scale-invariance. It assumes that the images are taken from a moving camera that only involves translation, and it does not account for rotation or scaling. BRIEF uses a upright window of fixed size (e.g. 48 x 48) for sampling.

Fast to compute the BRIEF descriptor

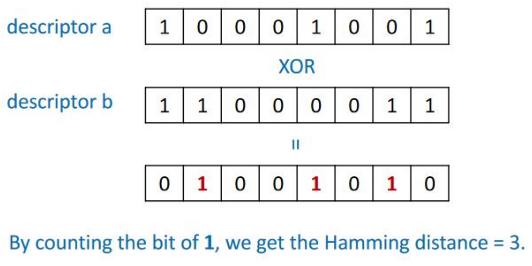
For example, to perform 8 binary tests and get one byte of descriptor:

$$\begin{aligned} descriptor = {} & ((I(p1) < I(q1)) << 7) + ((I(p2) < I(q2)) << 6) + ((I(p3) < I(q3)) << 5) \\ & + ((I(p4) < I(q4)) << 4) + ((I(p5) < I(q5)) << 3) + ((I(p6) < I(q6)) << 2) \\ & + ((I(p7) < I(q7)) << 1) + ((I(p8) < I(q8)) << 0) \end{aligned}$$

where p and q are pre-defined sampling points and $<< n$ means shifting bit by n places

Fast to compare two BRIEF descriptors

When performing image matching and comparing two BRIEF descriptors, measure their difference using the Hamming distance. Hamming distance can be computed very efficiently with a bitwise XOR operation followed by a bit count



10.2 Histograms of Oriented Gradient (HOG)

HOG is similar to SIFT. They both use gradient orientation histograms for feature description, the only difference is that HOG describes features for a large image region, instead of just around a point.

By describing a full image, we are able to perform image classification based on image features e.g. does it contain a human or not or retrieve similar images.

The idea of HOG is that it divides a large region into a dense grid of cells, describes each cell, then concatenate these local descriptions to form a global description:

- Divide the image into equally spaced cells
  - Each cell contains 8×8 pixels
  - 4 cells form a block

  The orientation histograms of this block describes a section of this image
- Move to the next block and similarly describe the content there using orientation histograms. Note there is an overlap between blocks.
- For each block, the descriptor vector v (concatenation of 4 histograms) is normalised:

$$v_{norm} = \frac{v}{\sqrt{||v||_2^2 + \epsilon^2}}$$

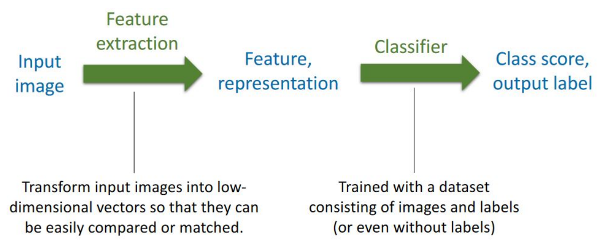
- The HOG descriptor is formed by concatenating the normalised local descriptors for all blocks to describe a full image

Chapter 11

Image Classification I

As previously mentioned, it is possible to describe the global feature of an image. Based on this, it is possible to start talking about *image classification*.

11.1 How image classification works


Input image
Feature extraction
Feature, representation
Classifier
Class score, output label
Transform input images into low-dimensional vectors so that they can be easily compared or matched.
Trained with a dataset consisting of images and labels (or even without labels)

The image dataset is split into two parts:

- Training set - Training the classifier
- Test set - Evaluating the performance of the classifier on data never seen

When learning with training set, there are several types:

- Supervised learning: Labels are available for training data
- Unsupervised learning: Labels are not available, clustering is often for classification
- Semi-supervised learning: Labels are available for parts of the training data

11.1.1 Preprocessing

Several processing is done such as:

- Detect where the digit is in a much bigger image
- Normalise the size of each digit
- Normalise the location, place the mass centre of the digit in the centre of the image
- Sometimes, slant correction may also be performed, which shifts each row in the image so that the principal axis becomes vertical

11.1.2 Image classification

Feature extraction includes:

- Hand-crafted (e.g. pixel intensities, HOG, or other manually defined feature descriptors)
- Learnt features (e.g. CNN, automatically learnt by the algorithm)

The classifier can have several types:

- K nearest neighbours (KNN)
- Support vector machine
- Neural network

11.2 K-nearest neighbours (KNN)

It is a non-parametric classifier:

- $K = 1$: Each test data point is assigned the class of its nearest neighbour
- $K > 1$: Compute K nearest neighbours and the test data point is assigned the class given by majority voting.

In KNN, neighbours are defined by a *distance metric* between data points x and y. There are many types of distance metrics:

- Euclidean distance
- Cosine distance
- Euclidean distance (L^2 norm)
- Manhattan distance (L^1 norm)

KNN have advantages and disadvantages:

- Advantage:
  - No training step at all
  - Simple but effective
  - Multi-class classification
- Disadvantage:
  - Storage and search are expensive

Computational-wise, KNN has N training images and M test images. At test time, the computational cost to classify M test images is

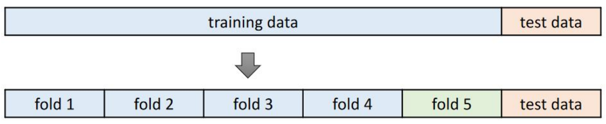
$$O(MN)$$

11.2.1 Hyperparameter K

Hyperparameters like K value are problem dependent and are established through parameter tuning.

Cross-validation

A common practice is to split the training set into 2 parts: a training set and a validation set, then tune the hyper-parameters on the validation set.



The dataset split is summarised into:

- Training set: training the model
- Validation set: deciding what type of model and which hyperparameters are the best
- Test set: getting a final estimate of model performance

It is expected the test performance to be possibly worse than the validation performance

- The reason is that we have been trying hard to fit our model onto the validation set, not the test set. They may have different distributions

Chapter 12

Image Classification II

K-nearest-neighbours are not the only models for image classification. There are other image classification models like Support Vector Machine and Neural Networks.

12.1 Support Vector Machine (SVM)

A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. In other words, it is a line that separates two different classes.

12.1.1 Linear classifier

The linear model is formulated as:

$$\mathbf{w}.\mathbf{x} + b = 0$$

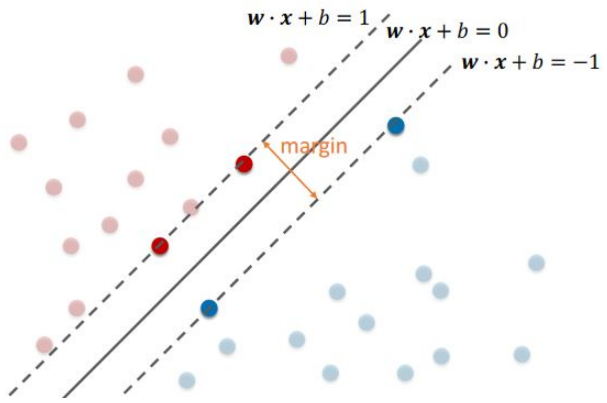
where $\mathbf{w}$ is the weights, $\mathbf{x}$ is the data and b is the bias

The rule of the linear classifier is to assign a class c to data x:

$$c = \begin{cases} +1 & \text{if } \mathbf{w}.\mathbf{x}{+}b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

To train the classifier, the parameters $\mathbf{w}$ and b determines the decision boundary and needs to be estimated.

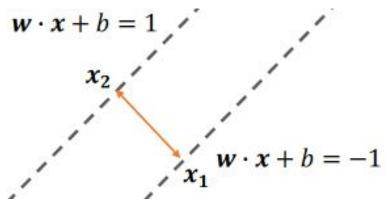
Intuitively, the best hyperplane (line) is one that is far from both classes of points i.e. **maximum margin hyperplane**. This line can be determined from only the innermost points i.e. **support vectors**.



Ideally, the support vectors should fulfil the equations $\mathbf{w}.\mathbf{x} + b = 1$ or -1, so that all the other pointsare easily classified

Maximum margin

The margin between $\mathbf{w}.\mathbf{x} + b = 1$ and $\mathbf{w}.\mathbf{x} + b = -1$ can be derived:



- Let x_1 be a point on the plane $\mathbf{w}.\mathbf{x} + b = -1$, $\mathbf{w}.\mathbf{x}_1 + b = -1$
- Move $\mathbf{x}_1$ along the normal direction $\mathbf{n}$ for the distance of the margin m, shoudl arrive at the point $\mathbf{x}_2$ on the other plane $\mathbf{w}.\mathbf{x} + b = 1$
- $\mathbf{w}.(\mathbf{x}_1 + \mathbf{n}) + b = 1$
- $\mathbf{w}.\mathbf{n} = 2$
- Normal: $\mathbf{n} = \frac{w}{||w||}$
- Margin: $m = \frac{2}{||w||}$

SVM aims to maximise the margin:

$$\max_{\mathbf{w},b} \frac{2}{||\mathbf{w}||}$$