Computer Vision

Lectured by Wenjia Bai

Typed by Aris Zhu Yi Qing

March 21, 2022

Contents

1 Image Filtering

1.1 Definition

- **Kernel**: a small matrix used to apply effects, e.g. blurring.
- **Separable kernel**: kernels that can be separated as two or more simple filters.
- **Padding**: The action of adding pixels around the borders (e.g. with value 0) so that applying filters will not reduce the size of the image.
- **Low-pass (smoothing) filter**: filters that keep the low-frequency signals, e.g. MA filter
- **High-pass (sharpening) filter**: filters that highlight the high-frequency signals, e.g. (identity + (identity - MA)) filter, or

$$\begin{pmatrix} -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 2 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \end{pmatrix}.$$

- **Denoising filter**: filters to remove noise, e.g. median filter, non-local means, block-matching and 3D filtering (BM3D), etc.

1.2 Common Filters

1.2.1 Moving Average (MA) Filter

- In a 2D case, the MA kernel is a $\mathbb{R}^{K\times K}$ matrix in the following form

$$\frac{1}{K^2}\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

with a time complexity of $O(N^2K^2)$, where N is the legnth of image.

- MA kernel is separable, for instance

$$\begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix} * \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix},$$

reducing the time complexity to $O(N^2K)$.

- Purpose:
  - remove high-frequency signal (noise or sharpness)
  - result in a smooth but blurry image

1.2.2 Identity Filter

The identity filter kernel is

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

1.2.3 Gaussian Filter

- The Gaussian kernel is a 2D Gaussian distribution
  $$h(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$
  with $i, j = 0$ as the centre of the kernel.
- While its support is infinite, small values outside $[-k\sigma, k\sigma]$ can be ignored, e.g. $k = 3$ or $k = 4$.
- 2D Gaussian filter is separable with
  $$h(i,j) = h_x(i) * h_y(j)$$
  where
  $$h_x(i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}},$$
  because
  $$\begin{aligned} f[x,y] * h[x,y] &= \sum_i \sum_j f[x-i, y-j] h[i,j] \\ &= \sum_i \sum_j f[x-i, y-j] \left(\frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} \right) \\ &= \sum_i \left(\sum_j f[x-i, y-j] \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{j^2}{2\sigma^2}} \right) \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}} \\ &= \sum_i (f * h_y)[x-i] \frac{1}{\sqrt{2\pi}} e^{-\frac{i^2}{2\sigma^2}} \\ &= (f * h_y) * h_x \end{aligned}$$
- Derivative of Gaussian filter h is
  $$\frac{\mathrm{d}(f * h)}{\mathrm{d}x} = f * \frac{\mathrm{d}h}{\mathrm{d}x} = f * \frac{-x}{\sqrt{\pi}\sigma^3} e^{-\frac{x^2}{2\sigma^2}}.$$
  Thus, the smaller the σ, the more detail in the magnitude map; larger σ suppresses noise and results in a smoother derivative. Different σ help find edges at different scale.

1.2.4 Median Filter

- non-linear
- often used for denoising
- Move the sliding window, and replace the centre pixel using the median value in the window.

1.3 Impulse Response

- For continuous signal, an **impulse** is a Dirac delta function $\delta(x)$, with
  $$\delta(x) = \begin{cases} +\infty, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$$
  such that $\int_{-\infty}^{\infty} \delta(x) \mathrm{d}x = 1$. For discrete signal, an impulse is a Kronecker delta function $\delta[i]$, with
  $$\delta[i] = \begin{cases} 1, & \text{if } i = 0 \\ 0, & \text{otherwise.} \end{cases}$$
- The **impulse response** h is the output of a filter when the input is an impulse. It completely characterises a **linear time-invariant** filter.

  - shifting the input signal k steps corresponds to the same output signal but shifted by k steps as well, e.g. assuming $f[n] = \delta[n]$, $g[n] = h[n]$,
    - $g[n] = 10f[n]$ is time-invariant and amplifies the input by a constant.
    - $g[n] = nf[n]$ is *not* time-invariant since the amount it amplies the input depends on the
  - if input $f_1[n]$ leads to $g_1[n]$, $f_2[n]$ leads to $g_2[n]$, we will have

$$\text{output}(\alpha f_1[n] + \beta f_2[n]) = \alpha g_1[n] + \beta g_2[n].$$

1.4 Convolution

- **Convolution**: output g can be described as the convolution between an input f and impulse response h as

$$g[n] = f[n] * h[n] = \begin{cases} \displaystyle\sum_{m=-\infty}^{\infty} f[m]h[n-m] & \text{discrete} \\ \displaystyle\int_{m=-\infty}^{\infty} f(m)h(n-m) & \text{continuous} \end{cases}$$

- Note that if we describe input signal $f[n]$ as

$$f[n] = \sum_{i=0}^{n} f[i]\delta[n-i]$$

and we known the output of $\delta[n]$ is $h[n]$, we can write the output as

$$g[n] = \sum_{i=0}^{n} f[i]h[n-i]$$

- commutative, i.e.

$$f[n] * h[n] = h[n] * f[n]$$

- associative, i.e.

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

- distributivity, i.e.

$$f * (g + h) = f * g + f * h \quad \text{and} \quad \frac{\mathrm{d}(f * g)}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}x} * g = f * \frac{\mathrm{d}g}{\mathrm{d}x}$$

- In 2D discrete case for image filtering,

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

if the dimension of the kernel is $(2M+1) \times (2N+1)$, we can write

$$(f * h)[m,n] = \sum_{i=-M}^{M} \sum_{j=-N}^{N} f[m-i,n-j]h[i,j]$$

with $h[0,0]$ being the centre of the filter, (m,n) being the location in the image which the kernel's center is on.

- If a big filter f_b can be separated into convolution g and h, we can first convolve with g, then h

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

2 Edge Detection

2.1 Detection

	finite difference	convolution kernel
Forward difference	$f'[x] = f[x+1] - f[x]$	$[1, -1, 0]$
Backward difference	$f'[x] = f[x] - f[x-1]$	$[0, 1, -1]$
Central difference	$f'[x] = (f[x+1] - f[x-1])/2$	$[1, 0, -1]$

2.2 Edge Detection Filters

2.2.1 Prewitt Filter

Along the x-axis and the y-axis, we have respectively

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}.$$

They are separable, i.e.

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

2.2.2 Sobel Filter

Along the x-axis and the y-axis, we have respectively

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

They are also separable, i.e.

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} * \begin{pmatrix} -1 & 0 & -1 \end{pmatrix}$$

2.2.3 Magnitude and Orientation Calculation

Let h_x denotes the horizontal filter, h_y denotes the vertical filter, we can compute the magnitude and the orientation as

$$\begin{aligned} g_x &= f * h_x && \text{derivative along } x\text{-axis} \\ g_y &= f * h_y && \text{derivative along } y\text{-axis} \\ g &= \sqrt{g_x^2 + g_y^2} && \text{magnitude of the gradient} \\ \theta &= \arctan(g_y, g_x) && \text{angle of the gradient} \end{aligned}$$

2.3 Canny Edge Detection

2.3.1 Criteria for Good Edge Detector

- good detection: low probability of FP/FN on marking edge points
- good localisation: mark as close as the centre of true edge
- single response: only one response to a single edge

2.3.2 Algorithm

1. perform Gaussian filtering to suppress noise
2. calucalte the gradient magnitude $M(x, y)$ and direction
3. apply Non-Maximum Suppression (NMS) to get a single response for each edge

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

4. perform hyteresis thresholding to find potential edges with two thresholds t_{low} and t_{high}

$$\begin{cases} M(x, y) \geq t_{\text{high}} & \text{accept} \\ M(x, y) < t_{\text{low}} & \text{reject} \\ \text{Otherwise} & \text{iteratively check neighbouring pixels and} \\ & \text{accept if connected to an edge pixel.} \end{cases}$$

5. evaluation/performance
   - good detection — FP reduced by Gaussian smoothing and FN reduced by hysteresis thresholding to find weak edges
   - good localisation — NMS finds locations based on gradient magnitude and direction
   - single response — NMS finds one single point in the neighbourhood

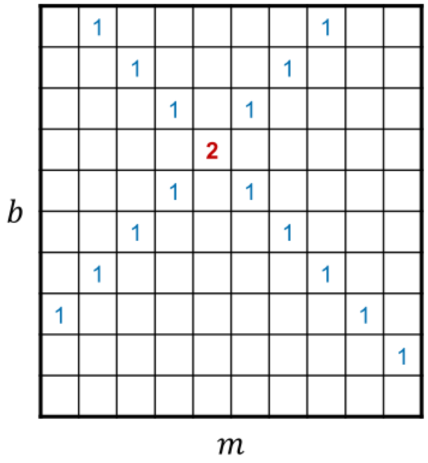

Figure 1: hough transform grid

3 Hough Transform

- **Hough transform** is a transform from image space to parameter space, e.g. from an edge map to the two parameters of a line.
- output is a parametric model, given the input edge points
- each edge point vote for possible models in the parameter space
- Example:
  - use slope intercept $b = y - mx$ to be the line model
  - each edge point poll vote for different b and m values (also lines in parameter space)
  - In practice, we use 2D bins to divide the parameter space; each point increases the vote by 1 in one of the bins, as shown in figure 1.
  - But the parameter space could be too large, $[-\infty, +\infty]$; use *normal form* instead

    $$x \cos\theta + y \sin\theta = \rho$$

    so at least for one dimension, $\theta \in [0, \pi)$.

- Line detection by Hough transform:
  1. Initialize the bins $H(\rho, \theta)$ to all zeros.
  2. For each (x, y) and each θ, calculate ρ and $H(\rho, \theta) + +$.
  3. Find (ρ, θ) where $H(\rho, \theta)$ is a local maximum and larger than a threshold.
     - local maximum so there can be multiple solutions, to reduce FN
     - larger than a threshold so as to reduce FP
  4. The detected lines are given by $\rho = x \cos\theta + y \sin\theta$.
- robust to noise/occlusion
  - edge map is often generated after image smoothing
  - broken/unoccluded edge points can still vote and contribute to line detection
- Circle detection by Hough transform
  - parameter space also forms circles
  - If radius r is also unknown, then 3D parameter space $H(a, b, r)$
  - parameterize to be $x = a + r\cos\theta$ and $y = b + r\sin\theta$, since we know θ form edge detection, we can narrow the voting area to move along θ for a distance r.
- Pros and Cons:
  + detects multiple instances
  + robust to image noise
  + robust to occlusion
  - Computational complexity is quite high. For each edge point, we need to vote to a 2D or even 3D parameter space.
  - need to carefully set parameters such as those for edge detectors, the threshold for the accumulator or range of circle radius.

4 Interest Point Detection

4.1 Definition

- **Interest point**: points that we are interested in and are useful for subsequent image processing and analysis. They are also called *key-points*, *landmarks*, *low-level* features.

4.2 Harris Detection

- small windows to tell if it is a corner
  - flat — change of intensity in neighter direction
  - edge — change of intensity along just one direction
  - corner — change of intensity along both direction
- If the window shifts by $[u, v]$, together with Taylor expansion

  $$I(x+u, y+v) = I(x,y) + uI_x(x,y) + vI_y(x,y) + \cdots$$

  the change of intensities is given by

  $$\begin{aligned} E(u,v) &= \sum_{(x,y)\in W} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \\ &= \sum w(x,y)[uI_x(x,y) + vI_y(x,y)]^2 \\ &= \sum w(x,y)(u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2) \\ &= \sum w(x,y) \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\ &= \begin{pmatrix} u & v \end{pmatrix} \sum w(x,y) \begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \\ &:= \begin{pmatrix} u & v \end{pmatrix} M \begin{pmatrix} u \\ v \end{pmatrix} \end{aligned}$$

- E is large if image derivatives are large
- We can infer the directions of change from M
  - If $M = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$, this is a *flat* region.
  - If $M = \begin{pmatrix} 10 & 0 \\ 0 & 0.1 \end{pmatrix}$, large change if shift along $u \Rightarrow$ an *edge*.
  - If $M = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}$, large change in whichever way $\Rightarrow$ a *corner*.
  - If more complicated M, we can do **eigen decomposition** to obtain diagonal matrix Λ, and compare against the above 3 matrices.
- **Cornerness** metrics:
  - Harris and Stephens:

    $$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(M) - k(\mathrm{trace}(M))^2$$

    where k is a small number, $k = 0.05$.
  - Kanade and Tomasi:

    $$R = \min(\lambda_1, \lambda_2)$$

  - Noble:

    $$R = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2 + \epsilon}$$

- also detects blobs and textures
- algorithm:
  1. compute x and y derivative of an image

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

     where G can be e.g. sobel filter
  2. at each pixel, compute

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

  3. calculate detector response and detect interest point which are local maxima and whose R is above threshold.

- *scaled* Harrison detector:

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

  so that

  - at each scale σ, determine the scale which gives us the largest detector response
  - prevent ”the larger the scale, the smaller the derivative magnitude” problem

- Algorithm: perform orignal Harrison detector steps for each scale, and then determine the interest point.

4.3 Laplacian of Gaussian (LOG)

- First perform Gaussian smoothing, then Laplacian Operator, i.e.

  $$\Delta(f*h) = \frac{\partial^2(f*h)}{\partial x^2} + \frac{\partial^2(f*h)}{\partial y^2} = f * \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right)$$

  where h is the Gaussian kernel and $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, with the following laplacian filter

  $$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

  Since we know the formation of a 2D Gaussian, we can derive the following:

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

- To ensure comparability between scales, we do

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

  similar to Harrison detector.

4.4 Difference of Gaussian (DoG)

- DoG is defined as

  $$\mathrm{DoG}(x,y,\sigma) := I * G(k\sigma) - I * G(\sigma) \approx (k-1)\sigma^2\nabla^2 G(x,y,\sigma)$$

  as an approximation of $\mathrm{LoG}_{\mathrm{norm}}$ using Gaussian filters at different scales.

- DoG filters are used in SIFT, which is a pipeline for detecting and describing interest points.
- ALso **scale-invariant**, and follow similar procedures as above.

5 Feature Description

5.1 Simple Descriptors

- Pixel intensity
  - - sensitive to absolute intensity value — same object with different illumination can have different intensity
  - - not very discriminative — a single pixel doesn’t represent any local content
- Patch intensities
  - + respresent local pattern
  - + perform well if the images are of similar intensities and roughly aligned
  - - sensitive to absolute intensity value
  - - not rotation-invariant
- Gradient orientation
  - + sensitive to intensity changes
  - + orientation is robust to scaling
  - - not rotation-invariant
- Histogram

+ robust to rotation
+ robust to scaling
- sensitive to intensity changes

5.2 Scale-Invariant Feature Transform (SIFT)

- detects and describe local features in images
- 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.
- Algorithm:

  1. detection of scale-space extrema (minima/maxima)

     search across scales and pixel locations, looking for interest points using DoG.

  2. keypoint localisation

     refine the estimate coordinates by fitting a quadratic curve to the neighbouring pixels and compute the new extrema.

     Denote the DoG response as $D(x, y, \sigma)$ or $D(\mathbf{x})$, by Taylor expansion we have

$$D(\mathbf{x}+\Delta\mathbf{x}) = D(\mathbf{x}) + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\Delta\mathbf{x}\frac{\partial^2 D}{\partial \mathbf{x}^2}\Delta\mathbf{x}$$

$$\Rightarrow \frac{\partial D(\mathbf{x}+\Delta\mathbf{x})}{\partial \Delta\mathbf{x}} = \frac{\partial D}{\partial \mathbf{x}} + \frac{\partial^2 D}{\partial \mathbf{x}^2}\Delta\mathbf{x} = 0 \Rightarrow \Delta\mathbf{x} = -\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1}\frac{\partial D}{\partial \mathbf{x}}$$

  3. orientation assignment

     determine the dominant orientation θ for a neighbourhood of a keypoint

     (a) an orientation histogram with 36 bins covering 360 degrees is created
     (b) each pixel votes for an orientation bin, weighted by the gradient magnitude
     (c) keypoint will be assigned an orientation, which is the peak of the historgram

     So now we have the location (x, y), scale σ, and dominant orientation θ.

  4. keypoint descriptor

     Compute a histogram of gradient orientations. In practice, subregions are used and each subregion has 4x4 samples.

     – each subregions has one orientation histogram
     – these multiple histograms together describe what it looks like around the keypoint.
     – each subregion has an orientation historgram with 8 bins in 8 directions.

     If 16 subregions are used, the descriptor (feature vector) has a dimension of $128 = 16 \times 8$.

- robust to rotation, scaling, and changes in illumination

  – rotation — relative to dominant direction
  – scaling — draw samples from a window proportional to size
  – changes in illumination — using gradient orientations

5.3 Keypoint matching

Suppose we find a keypoint (x, y) in A that corresponds to a keypoint (u, v) in B. We assume that they are reltaed with an *affine* transformation:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

with many pairs of corresponding keypoints, we can write the equation as:

$$\begin{pmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}\begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \end{pmatrix}$$

which can be written as a linear system

$$A\mathbf{m} = \mathbf{b}$$

where only $\mathbf{m}$ is unknown, and solve the least-square problem as

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

5.4 RANSAC

However, the squared difference, $\|A\mathbf{m} - \mathbf{b}\|^2$, can be sensitive to outliers(noise) — points that are deemed to be corresponding but actually aren't. To ensure the robustness to outliers. RANSAC(Random Sample Consensus) deals with this.

1. randomly sample some points
2. fit a line along the sampled points
3. find the number of **inliers** within a threshold to the line
4. terminate if enough inliers have been found, or we have reached a certain number of iteration

5.5 Acceleration

5.5.1 Speeded-Up Robust Features (SURF)

- SURF only computes the gradients along horizontal and vertical directions using **Haar wavelets**.
- SURF applies very simple filters d_x and d_y

$$d_x = \begin{pmatrix} -\mathbf{1} & \mathbf{1} \end{pmatrix}, d_y = \begin{pmatrix} -\mathbf{1}^T \\ \mathbf{1}^T \end{pmatrix}$$

  where $\mathbf{1}^T = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}$ called the Haar wavelets

- Summing pixel intensities with weigths 1 and -1 is very fast, so the result for each subregion is defined by (the sum of values and the sum of absolute values)

$$\left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right)$$

  - all four are low — homogeneous subregion
  - zebra pattern/strips — $\sum |d_x|$ or $\sum |d_y|$ is high
  - gradually increasing intensities — $\sum |d_x|$ and $\sum d_x$ are high

- Now we need $16 \times 4 = 64$ dimensions, and around 5 times faster than SIFT with Haar wavelets.

5.5.2 Binary Robust Independent Elementary Features (BRIEF)

- we compare an interest point p to another interest point q and get a binary value as output

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

- randomly sample n_d pairs of points for binary tests; random pattern is only determined once and same pattern applied to all intersect points.
  - If $n_d = 256$, we have 256 tests in total, with a total of 32 bytes (SURF uses 64 bytes).
  - fast to compute with bit-shifting operator << and >>.
  - use bitwise XOR between two descriptors to get the **hamming distance** instead of Euclidean distance, which is the bit count of the result.
- ignores rotation and scaling, assume images are taken from a moving camera with only translation.
- about 40-fold faster than SURF, which is around 200 times faster than SIFT.

5.5.3 Histograms of Oriented Gradient(HOG)

- we can extend the idea of feature descriptors to describe the feature of a large region, even a whole image.
- HOG divides a large region into a (dense) grid of cells, describes each cell, and concatenates the local descriptions to form a global description.
- The description vector $\mathbf{v}$ for each cell is normalized to form a locally normalized descriptor as

$$\mathbf{v}_{\text{norm}} = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|_2^2 + \epsilon^2}}$$

  with ϵ to ensure numerical stability

6 Image Classification

6.1 Classification concepts

See Intro to ML notes for k-NN, distance metrics, parameter tuning, cross-validation, stachastic gradient descent, multi-layer perceptron, forward/backward propagation

6.2 Support Vector Machine (SVN)

- If used as a *linear* SVN, the model is formulated as

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

  The rule is to assign a class c to data x with

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

  so we need to estimate $\mathbf{w}$ and b to determine the decision boudnary or *hyperplane*.

- The problem could be transformed to an optimisation problem of

  $$\min_{\mathbf{w},b} \|\mathbf{w}\|^2$$

  subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ for $i = 1, 2, \ldots, N$. Or formulate as

  $$\min_{\mathbf{w},b} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \max\left(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b)\right)$$

  since we cannot guarantee the data to be linearly separable. The second term of the above is equation is called the **hinge loss**. The gradient of the loss function is

  $$\nabla_{\mathbf{w}} L = 2\mathbf{w} - C \sum_{i=1}^{N} \nabla_{\mathbf{w}} h$$

  where

  $$\nabla_{\mathbf{w}} h = \begin{cases} -y_i \mathbf{x}_i & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i + b < 1) \\ 0 & \text{otherwise} \end{cases}$$

  so at each iteration,

  $$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{(k)}, b^{(k)})$$

6.3 Convolutional Neural Network

- In CNN, each neuron only see a small local region in the layer before it, called the **receptive field**.
- e.g. A neuron can depend on a 5x5xC cube of the input, where C refers to the number of color channels.
- More output neurons can be added as well, to form a Dx1x1 cube, where D denotes depth.
- Mathematically, a **convolution layer** is

  $$a_d = f\left(\sum_{ijk} W_{dijk} x_{ijk} + b_d\right)$$

  where d is the depth of output, ij is the dimension coordinate, and k is the input channel/depth.

- operations during convolution:
  - **padding**
  - **stride**
  - **dilation**: make a neuron look at a larger region (increase receptive field) by downsample image or increase kernel size

- **Pooling layer**:
  - operation to make feature maps/representations smaller
  - similar to image downsampling, e.g. if max pooling,

    $$\begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} \Rightarrow (6)$$

    pick the max element, and reduce the dimension

- issue:
  - *exploding* gradient — clip it, such as clip by value

    $$\mathbf{g}_i = \begin{cases} v_{\min} & \text{if } g_i < v_{\min} \\ v_{\max} & \text{if } g_i > v_{\max} \\ \mathbf{g}_i & \text{otherwise} \end{cases}$$

or clip by norm

$$\mathbf{g} = \begin{cases} \frac{\mathbf{g}}{\|\mathbf{g}\|} v & \text{if } \|\mathbf{g}\| > v \\ \mathbf{g} & \text{otherwise} \end{cases}$$

- *vanishing* gradient — use different activation functions, e.g. ReLU, leaky RelU, or
  - ∗ Parametric ReLU

$$f(z) = \begin{cases} az & z < 0 \\ z & z \geq 0 \end{cases}$$

  - ∗ Exponential Linear Unit

$$f(z) = \begin{cases} a(e^z - 1) & z < 0 \\ z & z \geq 0 \end{cases}$$

- Representative CNN: LeNet, AlexNet, VGG

7 Object Detection

7.1 Approaches

- general ideas
  - at each sliding window, perform two tasks.
  - Firstly, classification (whether this is a cat or not)
  - secondly, localisation (bounding box coordinates) using CNN to predict bounding coordinates
  - however, too expensive to apply CNN on each pixel of the image
- two-stage detection:
- make an initial guess, propose some possible regions of interest
- classify what these regions are and predict the bounding boxes
- One-stage detection:
  - do not guess, but simply divide the image into grid cells
  - classify these grid cells and predict the bounding boxes

7.2 Two-stage Object Detection Methods

7.2.1 Selective Search

We can look at image features such as greyscale or gradients to separate the image into regions with similar features.

7.2.2 Faster R-CNN

RPN

- A **Region Proposal Network** (RPN) is used.
- The input image is fed into a CNN, gives the image's feature map.
- The feature map goes into a RPN, outputing interesting regions to be looked at.
- Using AlexNet/VGG-16 as **backbone networks**, using their last convolution layer as a feature map, we can use a 3×3 sliding window to perform binary classification at each location, giving it 0 if it isn't interesting, and 1 if it is.
- it handles objects of different sizes / aspect ratio by making k predictions(bounding boxes). e.g. 128^2, 256^2, 512^2, $1:1$, $1:2$, $2:1$. These bounding boxes are called **anchors**.
- A bounding box can be described by its centre coordinates (x, y) and size (w, h).
- For a convolution feature map of $W \times H$, $W \times H \times k$ anchors are predicted, and only the highest scoring boxes are kept.
- The loss is defined as

$$L(p, t) = \underbrace{\sum_{i=1}^{n_{\text{anchor}}} L_{\text{cls}}(p_i, p_i^*)}_{\text{classification loss}} + \lambda \underbrace{\sum_{i=1}^{n_{\text{anchor}}} 1_{y=1} L_{\text{loc}}(t_i, t_i^*)}_{\text{localisation loss}}$$

  where * denotes the ground truth

- we predict how to transform this anchor into the ground truth bounding box using:

$$\begin{aligned}\text{anchor} &= (x_a, y_a, w_a, h_a)\\ \text{predicted bounding box} &= (x, y, w, h)\\ \text{predicted transformation} &= (t_x, t_y, t_w, t_h)\\ t_x &= \frac{x - x_a}{w_a}\\ t_y &= \frac{y - y_a}{h_a}\\ t_w &= \log\frac{w}{w_a}\\ t_h &= \log\frac{h}{h_a}\end{aligned}$$

RoI

- Combining classification and localisation, we have features for each region known as **RoI** (Region of Interest). We can use an RoI pooling layer, which we can calculate RoI location and size on the feature map and convert to a fixed size to be provided to the classifier.
- For each RoI, the classifier predicts the label class and refines the bounding box estimate:

  $$L(p, t) = L_{\text{cls}}(p, y) + \lambda \cdot 1_{y \geq 0} L_{\text{loc}}(t, t^*)$$

  where y denotes the ground truth, and is now a multi-class classification problems.

Comparison between RPN and Detection Network
Please see table 1.

7.3 One-stage Object Detection Methods

- e.g. YOLO, SSD. The RPN is changed from using a binary value for the classification loss to using a multi-class classifier, which predicts the object for each anchor.
- Faster R-CNN is more accurate but slower, and vice versa for SSD, since RPN estimates the regions size before looking closely at features and then refining it.

Table 1: RPN v.s. RoI

RPN	Detection network
The input to RPN is a 3 × 3 window on conv5 feature map.	The input to detection network is a proposed region, thus contains more accurate features.
It is class-agnostic. It only checks whether this is an RoI or not.	It classifies the region into a number of classes.
RPN needs to use a lot of anchors, because we do not know what the object looks like yet	It does not need to use anchors. We already know a rough size from the proposal.

- Backbone networks (VGG/AlexNet) can be changed out to improve performance.

8 Image Segmentation

8.1 Thresholding

This converts a greyscale image to a binary label map. At each pixel, the label is defined as

$$f(x) = \begin{cases} 1 & \text{if } I(x) \geq \text{threshold} \\ 0 & \text{otherwise.} \end{cases}$$

This doesn't require any training data, and only needs threshold as the parameter.

8.2 K-means

- unsupervised method
- Represents each cluster by its centre. Each data point (pixel intensity) is associated to the nearest cluster centre.
- It can be formulated as

$$\min \sum_{k=1}^{K} \sum_{x \in C_k} (x - \mu_k)^2 \quad \text{or} \quad \min \sum_{k=1}^{K} \sum_{x} \delta_{x,k} (x - \mu_k)^2 \tag{1}$$

where $\delta_{x,k}$ denotes membership of x in cluster k, μ_k denotes the centre of cluster k.

- Algorithm
  1. initialize μ_k, where $k = 1, 2, \dots, K$.
  2. For each iteration
     (a) compute $\delta_{x,k}$ for each data point, assigning x to the nearest cluster centre μ_k.
     (b) update μ_k according to the membership $\delta_{x,k}$.
     (c) repeat until $\delta_{x,k}$ no longer changes or the maximum number of iterations is reached.
- To determine which K to use, plot equation (1) with different K, and use the "elbow" method to determine the best value.
- clustering can also be performed base on
  - color similarity
  - position + color similarity
  - other features

  so that x becomes a feature vector instead of a scalar.

8.3 Gaussian Mixture Model (GMM)

- unsupervised method
- GMM performs a *soft* assignment by assuming a Gaussian distribution for each cluster, formulated as

$$P(y_j = k | x_j, \pi_k, \mu_k, \sigma_k) = \pi_k \cdot \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$$

  where x_j is the intensity of feature for data point j, π_k is the mixing component for class k, y_j denotes the group which point j belongs to.
- See intro to ML for detailed algorithm.

8.4 CNN

- supervised method
- we can e.g. remove the fully connected layers and directly infer pixel-wise classification results from the 13 × 13 feature map
- **Fully convolutional network**: all layers are convolutional layers.
- **Convolutionalization**: replaces the fully connected layers with convolutional layers
- Need to apply **upsampling** operation to obtain a pixel-wise prediction
  - e.g. transposed convolution. For $\mathbb{R}^{4\times4} \mapsto \mathbb{R}^{8\times8}$, we can have one input correspond to a 3×3 region in the output, with some weight, with a stride of 2.
  - convolution in 1D as matrix operation v.s. 2D, see figure 2.

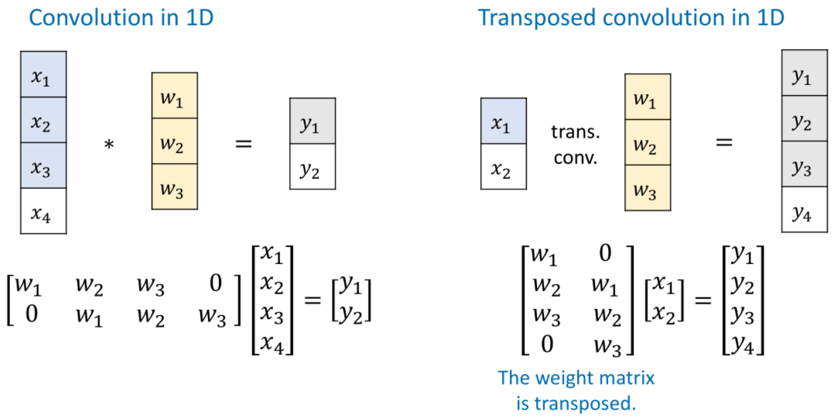
Convolution in 1D
Transposed convolution in 1D
x_1 x_2 x_3 x_4
$*$
w_1 w_2 w_3
$=$
y_1 y_2
trans. conv.
y_1 y_2 y_3 y_4
$\begin{bmatrix} w_1 & w_2 & w_3 & 0 \\ 0 & w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$
$\begin{bmatrix} w_1 & 0 \\ w_2 & w_1 \\ w_3 & w_2 \\ 0 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$
The weight matrix is transposed.

Figure 2: convolution v.s. transposed convolution in Matrix operation

- Thus a classification network can be transformed to a segmentation network by replacing the fully connected layers with convolutional and transposed convolutional layers.
- **Mask R-CNN**

  - the input is the convlution feature map
  - fed into two branches: detection and segmentation, so perrforming two tasks at the same time
  - segmentation branch is fully convolutional

9 Motion

9.1 Definition

- **Optic Flow**: The motion (flow) of brightness patterns (optic) in videos.
- The output of optic flow methods is a flow field, which describes the displacement vector for each pixel in the image.
- Assumption in optic flow:
  - brightness constancy
  - small motion
  - spatial coherence

9.2 Optic flow constraint

- Based on brightness constancy assumption, we have

$$I(x+u, y+v, t+1) = I(x, y, t)$$

where I is the intensity, (x, y, t) is the spatial and temporal coordinate, and (u, v) is the displacement

- Based on small motion assumption, using first-order Taylor expansion we can approximate

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \quad \text{or} \quad I_x u + I_y v + I_t = 0$$

9.3 Lucas-Kanade method

- At each pixel p, we have the optic flow constraint equation

$$\begin{pmatrix} I_x(p) & I_y(p) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -I_t(p)$$

- For a small neighbourhood, e.g. a 3×3 window, we have a system of linear equations,

$$\begin{pmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_N) & I_y(p_N) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_N) \end{pmatrix}$$

equivalently a problem of the form $Ax = b$.

- solution is given by

$$x = (A^T A)^{-1} A^T b \tag{2}$$

where

$$x = \begin{pmatrix} u \\ v \end{pmatrix}, A^T A = \sum_p \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}, A^T b = -\sum_p \begin{pmatrix} I_x I_t \\ I_y I_t \end{pmatrix}$$

- Note that the matrix $A^T A$ also appears in the Harris detector 4.2.
- Using concepts in linear algebra, since

$$\text{cond}(A^T A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

for flat regions, we have large condition number $\Longrightarrow$ unstable calculation of the inverse; for corner regions, we have larger $\lambda_{\min}$, so smaller condition number $\Longrightarrow$ more stable inverse calculation.

Intuitively, the motion of flat/edge region couldn't be figured around a small neighbourhood accurately, but a corner is fine.

- **multi-scale**: motion at different scale. For each scale l from coarse to fine

1. upsample the flow estimate at previous scale $l+1$ with
$$u^{(l)} = 2u^{(l+1)}, v^{(l)} = 2v^{(l+1)}$$
2. compute the warped image
$$J^l_{\text{warped}} = J^l(x + u^{(l)}, y + v^{(l)})$$
3. for image I^l and J^l_{warped} at this scale, compute the image gradients I_x, I_y and I_t.
   - I_x and I_y are caluclated from image I^l.
   - $I_t = J^l_{\text{warped}}(x,y) - I^l(x,y)$.
4. estimate the incremental flow using
$$\begin{pmatrix} u^\delta \\ v^\delta \end{pmatrix} = \left(A^T A\right)^{-1} A^T b$$
5. update the flow at scale l: $u^{(l)} = 2u^{(l+1)} + u^\delta$, $v^{(l)} = 2v^{(l+1)} + v^\delta$.

- For *object tracking*, the optimisation problem becomes
$$\min_{u,v} E(u,v) = \sum_x \sum_y \left[I(x,y) - J(x+u, y+v)\right]^2$$
where x, y are pixels in the **template image**, u, v indicates the motion from template I to image J. The solution is the same as equation (2).

9.4 Horn-Schunck Method

- an optimisation-based method, defining a global energy **functional** (a cost function) for the flow for all the pixels
$$E(u,v) = \iint_{(x,y)\in\Omega} \underbrace{\left(I_x u + I_y v + I_t\right)^2}_{\text{optic flow constraint}} + \underbrace{\alpha}_{\text{weight}} \underbrace{\left(|\nabla u|^2 + |\nabla v|^2\right)}_{\substack{\text{smoothness term} \\ \text{for the flow}}} \, \mathrm{d}x\mathrm{d}y$$
$$:= \iint_{(x,y)\in\Omega} L(x, y, u, v, u_x, u_y)\mathrm{d}x\mathrm{d}y$$
where $u = u(x,y)$ and $v = v(x,y)$ are the unknown **functions** defined on the image pixels. They are solved by minimising the energy function $E(u,v)$.

- minismizing the functional comes down to solving its associated Euler-Lagrange equation,
$$\frac{\partial L}{\partial u} - \frac{\partial}{\partial x}\frac{\partial L}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial u_y} = 0$$
$$\frac{\partial L}{\partial v} - \frac{\partial}{\partial x}\frac{\partial L}{\partial v_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial v_y} = 0$$
this follows that
$$(I_x u + I_y v + I_t)I_x - \alpha \Delta u = 0$$
$$(I_x u + I_y v + I_t)I_y - \alpha \Delta v = 0$$
where $\Delta := \partial_{xx} + \partial_{yy}$.

- In practice, the Laplacian Δu can be approximated using finite differences
$$\Delta u = \overline{u} - u$$
where $\overline{u}$ is the local average within a small neighbourhood.

- Plug in the approximation for Δu and reorganize, we obtain
$$(I_x^2 + \alpha)u + I_x I_y v = \alpha\overline{u} - I_x I_t$$
$$I_x I_y u + (I_y^2 + \alpha)v = \alpha\overline{v} - I_y I_t$$
becoming an equation system for u and v, which can be solved as
$$u = \overline{u} - \frac{I_x(I_x\overline{u} + I_y\overline{v} + I_t}{I_x^2 + I_y^2 + \alpha}$$
$$v = \overline{v} - \frac{I_y(I_x\overline{u} + I_y\overline{v} + I_t}{I_x^2 + I_y^2 + \alpha}.$$
thus u, v, $\overline{u}$, $\overline{v}$ can be estimated iteratively.

9.5 Learning-based Flow Method

- A CNN takes in an input (two frames of an image), and outputs a flow field, visualized with a displacement vector at each pixel.

- The average End-point Error (EE) can be used to evaluate the performance of an optic flow method if the ground truth is available
$$\text{EE} = \frac{1}{N}\sqrt{\left(u(x,y) - u_{\text{GT}}(x,y)\right)^2 + \left(v(x,y) - v_{\text{GT}}(x,y)\right)^2}$$

9.6 Correlation Filter

- In a correlation filter, we aim to maximise the correlation between tempalte features and image features in the next time frame

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

- The correlation can be more robust to illumination changes than sum of squared difference.
- In 2D images, we have a search window (in the next time frame) and a box with the same size as the template. We can find where the correlation achieves the maximum between the template features and features in a search window.
- The features can be learnt from CNNs.

9.7 Tracking by Detection

- tracking v.s. detection
  - tracking concerns about videos
  - In tracking, we have initial bounding box. We are only interested in how this object moves. We do not need to detect every object.
- **online learning** and **offline learning**
  - online: using information from this video, while it is streamed.
  - offline: using information from an existing trainign set.

  Prefered method here is online learning.
- At each sliding window, we perform two tasks
  1. perform **binary** classification
  2. localisation to enable us to adjust the size of the object during tracking
- The positive and negative smaples for binary classification are extracted from the first time frame. More samples can be generated by
  - data augmentation: translation, rotation, etc.
  - using more time frames while the video is streamed.
- Most layers of the convolutional network are pre-trained offline using large datsets such as ImageNet.
- Only the last few layers are trained online, using a lot of samples extracted from the video.

9.8 Action Recognition

We use both image and motion information as inputs

- image: static features extracted from a single image
- motion: dynamic features extracted from videos, e.g. optic flow fields

to train a classifier for action recognition.