Image Filters

Blurring: Filtering with Low Pass Filters applies a Blurring Effect (i.e. to extract image features)
Moving Average Filter: Moves a Window Across a Signal & Calculates the Average Value within the Window
Moving Average Window: Window that Calculates the Moving Average of all Pixels in the Window (i.e. average pixel intensity)
→ removes high frequency signals (noise/sharpness) → results in a smoother & blurrier image → gives average pixel intensity
Kernel: Set of Weights Given to Each Pixel in Window (i.e. 3x3 Moving Average Kernel has 1/9 for Each Pixel)
→ Place Filter Kernel at Every Possible Location in Image Array & Calculate Weighed Summation Across all Pixels in Window
Moving Average Window & Blurring: Moving Average Window used to Create a Blurred Version of Original Image by Averaging the Pixels Values in a Neighbourhood Around Each Pixel
→ a larger window means more pixels are included in the averaging process
→ when a large number of pixels are averaged - the differences between adjacent pixel values become less pronounced
→ reducing the variation in pixel intensity & leading to a smoother/blurrier image
→ a large window blurs distinctions between features in the image (i.e. edges/textures become less sharp & may lose fine details)
Padding: allows the application of filter to give output image with same dimensions as input image
→ without padding - the output image is smaller than the input image (missing boundary pixels)
→ padding deals with the boundary pixels & provides an output image with the same dimensions
→ can be padded by constant values / mirroring values (mirror the value on the other side of the window)
→ can apply filter kernel to edge of image - with padding pixels included in window
Computational Complexity (Image Size: NxN & Kernel Size: KxK): at each pixel, K^2 multiplications & K^2-1 summations (as multiplying each pixel in kernel with corresponding value in the kernel & summing the result of these multiplications)
→ done for N^2 pixels in image - giving N^2K^2 multiplications & $N^2(K^2-1)$ summations
→ $O(N^2K^2)$ (complexity scales quadratically with the size of the image (N) & size of kernel (K))
→ the larger the kernel - the more blurred the image - & the more computationally expensive the operation
Separable Filter: Filter can be Separated as the Consecutive Operations of Two Small Filters (First Apply Filter 1 & Then Filter 2)
→ optimises the computational complexity & accelerates the filtering operation
Separable Moving Average Filter: Separate 3x3 Moving Average Filter into 2 1-D Kernels & Reduce Computational Complexity
→ 3x3 Filter of 1/9 = 1x3 Row Filter of 1/3 Convoluted with 3x1 Column Filter of 1/3
→ Row Filter Computes the Average of the 3 Pixels in a Row for Each Pixel in the Image
→ Column Filter Computes the Average of Three Pixels in a Column for Each Pixel
→ Convolution Means Applying the Row Filter to Each Pixel in the Image (Blurring the Image Horizontally) - then Applying the Column Filter to Each Pixel in the Image (Blurring the Image Vertically) - with Final Image Equivalent to Applying 3x3 Filter
Complexity of Separable Filtering (Image Size: NxN & Kernel Size: 1xK & Kx1) at each pixel, K multiplications & K-1 summ.
→ done for N^2 pixels & twice for 2 kernels - $2N^2K$ multiplications & $2N^2(K-1)$ summations
→ $O(N^2K)$ (vs. original complexity of moving average window $O(N^2K^2)$) & hence separable filter more efficient
Types of Image Filter: Identity, Low-Pass/Smoothing (Moving Average, Gaussian), High-Pass/Sharpening, Denoising (Median)
Identity Filter: when applied to an image - leaves the image unchanged
→ when this filter is applied to image through convolution - each pixel in the output image is set to value of pixel in input image
→ Kernel has 1 assigned to central pixel & no value elsewhere
Gaussian Filter: uses the Gaussian Distribution to apply a filter - creating a smoothing effect (essential for reducing noise & detail)
→ Kernel $h(i,j) = 1/2\pi\sigma^2 \quad e^{-(i^2+j^2)/2\sigma^2}$
→ i & j denote the offset from center of window (pixel coordinates), σ denotes standard deviation & controls shape of filter
→ infinite support (region with non-0 values) - but often ignore small values outside of $[-k\sigma, k\sigma]$ (i.e. k=3)
Separable Gaussian Filter: the 2D Gaussian Filter is Equivalent to 2 1-D Gaussian Filters with the same σ
→ one along the x-axis (across row) & one along the y-axis (across column) (reducing the computational complexity)
→ $h(i,j) = h_x(i) * h_y(j)$ & $h_x(i) = 1/\sqrt{2\pi}\sigma \quad e^{-i^2/2\sigma^2}$ & $h_y(j) = 1/\sqrt{2\pi}\sigma \quad e^{-j^2/2\sigma^2}$
High Pass Filter 1: Identity + (Identity - Moving Average) = High-Pass
→ Identity Filter does not change image & so keeps both high-frequency & low-frequency signals
→ taking a low-frequency filter away from the Identity Filter (low & high) - gives a high-frequency filter
→ adding a high-frequency filter to an Identity Filter emphasises the high-frequency parts of the image
→ low & high + (low & high - low) = low & high + high → emphasising high-frequency signals
High Pass Filter 2: Identity + High-Frequency = High-Frequency
→ given a high-frequency filter, adding to an Identity Filter emphasises the high-frequency signals
→ high-frequency filter is finding difference between central & neighbouring pixels - emphasising high-freq parts of central pixel
Median Filter: a non-linear filter that is used for denoising
→ moving the sliding window & replacing the center pixel using the median value of the pixel intensities in the window
→ sort all values in the window & select the middle value
→ removes almost all salt & pepper noise in the input image (as salt & pepper noise introduces very low/high values into window)

Filtering & Convolution

Filtering: a filter h takes an input signal f - processes it- & generates an output signal g
→ Filtering can remove unwanted features of a signal/enhance wanted features, smooth/sharpen/denoise signals, can be 1D/2D..
Impulse Response: the impulse response (h) completely characterises a LTI System
→ as long as impulse response h is known - can calculate the output signal y - given any input x
Time-Invariant: when input shifted by time step k - otuput also shifts by k
Linear: given linear combination of input signals, same linear combination of output signals
→ given two input images - if we take a linear combination of two input images & then apply linear filter - equivalent to first applying linear filter to two images independently & linearly combining outputs
LTI System: $g[n] = f[n] * h[n]$ (input series of impulses at diff. times & output series of corresponding impulse responses)
Convolution: $g[n] = \sum_{-\infty}^{\infty} f[m]h[n-m] = \sum_{-infty}^{\infty} f[n-m]h[m] \quad f[n]*h[n] = h[n]*f[n]$
Flip & Shift: flip the impulse response ($-n$), shift it across the signal ($+m$), multiply with $f[m]$ & sum to get $g[n]$
2D Convolution: $g[m,n] = f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j]$
$= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[m-i,n-j]h[i,j]$ → summation across both x & y-dimension
2D Flip & Shift: flip the impulse response/kernel along x & y ($-i$ & $-j$), shift the kernel ($+m,+n$), multiply with $f[i,j]$ & sum to get $g[m,n]$
Convolution is Associative: $f * (g * h) = (f * g) * h$ (shown by expanding the left/right equations & using change of variables technique)
→ given associativity, if h_{big} equivalent to $h_1 * h_2$: $f * h_{big} = f * (h_1 * h_2) = (f * h_1) * h_2$ (separable filtering)
Prove Gaussian Filter is Separable: $f[x,y] * h[x,y] = \sum_i\sum_j f[x-i,y-j]h[i,j] = \sum_i\sum_j f[x-i,y-j]1/2\pi\sigma^2 \quad e^{-(i^2+j^2)/2\sigma^2} = \sum_i(\sum_j f[x-i,y-j]1/\sqrt{2\pi}\sigma \quad e^{-j^2/2\sigma^2})1/\sqrt{2\pi}\sigma \quad e^{-i^2/2\sigma^2} = \sum_i f * h_y[x-i]1/\sqrt{2\pi}\sigma \quad e^{-i/2\sigma^2} = (f * h_y) * h_x$
Differentiation Property: $d/dt[x(t)*y(t)] = dx(t)/dt * y(t)$ & $d^2/dt^2[x(t)*y(t)] = dx(t)/dt * dy(t)/dt$

Camera Model

Camera Model: X = PX (camera matrix P maps 3D coordinate X to 2D image x)
Pinhole Camera: models how a 3D coordinate X = (X,Y,Z) is mapped to x = (x,y) on an image plane
→ image plane upside down - for convenience - rotate image plane by 180 degrees & put as virtual image plane in front of pinhole
→ using the virtual image plane to represent the image & simplifying the model
→ by similar triangles: (X,Y,Z) in 3D maps to (fX/Z,fY/Z) in 2D (f is focal lengt/distance from cam. origin to virt. img. plane)
→ using homogeneous coordinate, 3D to 2D mapping: (X,Y,Z,1) → (fX/Z,fY/Z,1) or (fX,fY,Z)

Camera Eq.: $\begin{bmatrix} fX + p_xZ \\ fY + p_yZ \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow x = [K|0]X_{cam} \quad K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$

World Coordinate System: the camera coordinate system is related to the world coordinate system by a 3D rotation & translation
→ $X_{cam} = R(X - C)$ **Pinhole Cam Matrix:** (maps from world coordinate to camera & then to image)

$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0} & 1 \end{bmatrix} \cdot \mathbf{X}$ → Pinhole

Camera Matrix p has 9 DoF (intrinsic parameters f, p_x, p_y have 3 DoF & extrinsic parameters have 6 DoF

+ Skew & Conversion to Pixels: $\mathbf{P} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \end{bmatrix}$ (11 DoF)

→ k_x & k_y ratios for conversion, $\alpha_x = k_x fX/Z$, $x_0 = k_x p_x$, $\alpha_y = k_y fY/Z$, $y_0 = k_y p_y$
Camera Calibration/Pose Estim.: $p = \text{argmin}_p ||Ap||^2$ s.t. $||p||^2 = 1$
→ solved by performing SVD for A & solution is column of V that corresponds to smallest singular value

Edge Detection

Edges: lines where image brightness changes sharply & has discontinuities (colour/depth/surface normal discontinuity)
→ **edges capture important properties & are important low-level features for analysing images**
Detecting Edges: image represented as function of pixel position, derivatives characterise discontinuities of function & detect edges
Derivative of a Discrete Function: (as Convolution $f'[x] = f[x] * h[x]$)
→ **Central Difference:** $f'[x] = f[x+1] - f[x-1]/2$ Kernel: $h = [1, 0, -1]$
Edge Detection Filters: Prewitt Filter, Sobel Filter, Smoothing Filters
Prewitt Filter: 2D Convolution Kernel computes the derivative (central difference) in 2D (finds discontinuities along x & y-axis)

$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ along x-axis $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ along y-axis

→ Separable into a Moving Average Filter & Finite Difference Filter

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

→ performs derivative across x-dim (for each row) & performs moving average across three rows (for each column) to smooth
Sobel Filter: similar to Prewitt Filter - assigning more weight to central pixels (performs finite difference in 2D to find discontinuities)

$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ along x-axis $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$ along y-axis

→ **Separable into Weighted Average & Finite Difference Filter**

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

→ applying along x-axis gives output describing the derivative along the x-axis & shows vertical discontinuities/edges
→ applying along the y-axis gives output describing the derivative along the y-axis & shows horizontal discontinuities/edges
Image Gradient:
→ at each pixel - two outputs from Sobel Filters (one describing derivative along x-axis & another the derivative along y-axis)
→ this is the gradient - $f = (\delta f/\delta x, \delta f/\delta y)$ - & is descibed by magnitude & orientation
→ **Compute the Derivatives along x-axis & y-axis by applying the Sobel Filter:** $g_x = f * h_x$ & $g_y = f * h_y$

Gradient Magnitude: $g = \sqrt{g_x^2 + g_y^2}$ & **Gradient Orientation:** $\theta = tan^{-1}(g_y/g_x)$

→ can convert magnitude map into a binary map using thresholding
Smoothing: Prewitt & Sobel contain MAV Filters, as derivatives are sensitive to noise, smoothing kernel helps suppress noise
→ for 2D image - perform smoothing with a 2D Gaussian Filter & then apply Prewitt/Sobel, & calculate gradient mag/orient.
→ 2D Gaussian: $h[x,y] = 1/2\pi\sigma^2 \quad e^{-(x^2+y^2)/2\sigma^2}$
→ derivatives are sensitive to noise & so much easier to detect edge of Gaussian Smoothed Signal (clearer peak signal)
Canny Edge Detection: perform **gaussian filtering** to suppress noise - calculate the **gradient magnitude & direction**, apply **non-maximum suppression (NMS)** to get a single response for each edge - perform **hysteresis thresholding** to find potential edges
→ carefully designs the image features (gaussian filtering & image gradient) & procedures (NMS & Thresholding)
→ Gaussian Suppresses Noise, Gradient Magnitude & NMS Find Location of Edges, Hysteresis Thresholding Finds Weak Edges
Non-Maximum Suppression (NMS): aims to get a single response for an edge (width of edge = single pixel)
→ **Idea:** edge occurs where the gradient magnitude reaches the maxmum (suppressing the non-max values)
→ **To Check if Pixel q is Local Max. Along Gradient Direction:** move to pixel r & compare the gradient magnitudes between q & r - move to pixel p & compare the gradient magnitudes between q & p - if pixel q is the local maximum - it is an edge point & can suppress the other non-maximum pixel
Suppression: $M(x,y) = M(x,y)$ if local max, $M(x,y) = 0$ otherwise
→ in practice - when comparing gradient magnitudes of q to r & p - pixels r & p may not be located on pixel lattice & so need to perform image interpolation of pixels r & p (using nearest neighbour or linear interpolation) to interpolate the gradient magnitudes of r & p (i.e. average of gradient magnitudes of four neighbours on pixel lattice)
→ alternatively - can round the gradient direction into 8 possible angles representing directions towards 8 surrounding pixels (0/45/90..) (only comparing gradient magnitudes along these directions & so no interpolation needed)
Thresholding: some pixels may be local maxmima but have low magnitudes - want to detect edges with high magnitudes
→ & then can convert magnitude map into a binary image
→ **Simple Thresholding:** 1 if Intensity ¿ T, & 0 otherwise
→ **Hysteresis Thresholding:** defines two thresholds, t_{low} & t_{high}
→ if a pixels gradient magnitude is $\geq t_{high}$ - accepted as an edge pixel → if $< t_{low}$ - rejected
→ if a pixels gradient magnitude is between - it is a weak edge pixel & will be accepted if connected to a neighbouring edge pixel (performed iteratively until all pixels are either accepted or rejected) → allows for weak edges connected to strong edges to be accepted as an edge
Effect of σ: a large σ detects large-scale edges & smooths fine details / a small σ detects fine features
Improving Edge Detection Accuracy: utilising richer features (colour/texture), multi-scale features, enforcing smoothness, utilising machine learning (i.e. learning mapping from image to edge directly from paired data)
→ a ML model can integrate features from multiple scales - fusing fine-scale edges with coarse-scale edges to form the final output

Motion

Video: 2D-t image sequence captured over time - in motion tracking, at each (x,y) at time t, want to estimate (x+u,y+v) at t+1
Optic Flow: the motion (flow) of brightness patterns (optic) in video
→ the output of optic flow methods is a dense pixel-wise motion field - describing the displacement vector for each pixel
Assumes: brightness constancy (pixel has constant brightness across time), small motion (between frames), spatial coherence (motion of neighbouring pixels is similar / flow is constant in a small neighbourhood)
Constraint: Intensity at time t+1 = Intensity at time t: $I(x+u, y+v, t) = I(x,y,t)$
→ performing Taylor Expansion: $\approx I(x,y,t) + \delta I/\delta x \; u + \delta I/\delta y \; v + \delta I/\delta t$
→ combining two equations: $I_xu + I_yv + I_t = 0$ **(Optical Flow Constraint Equation)** (1 Eq. of 2 Unknowns - No Soln.)
→ as this is based on 1st Order Taylor Exp. - works well if u & v small - otherwise not (small motion assump.)
Lucas-Kanade Method: introduces spatial coherence assumption to introduce another equation in u & v
→ allows us to take a small KxK window & transform underdet. eq. to overdet. linear sys. of eqs. (more eqs. than unknowns)
→ compute the image gradients I_x, I_y (finite difference between neighbouring pixels) & I_t (finite difference between neighbouring time frames)
→ for each pixel:

. → calculate $A^TA = \sum_p \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$ & $A^Tb = -\sum_p \begin{bmatrix} I_xI_t \\ I_yI_t \end{bmatrix} \rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = \left(A^TA\right)^{-1} A^Tb$

Optical Flow Relation to Corner Response: where there is a high corner response - optic flow estimation is more robust
→ if we observe motion from a small neighbourhood - the motion at the corner is easier to track compared to edge/flat region
Aperture Problem: when looking through an aperture (hole) the motion of a line is ambiguous (corner clearer to define)
→ the motion component parallel to a line cannot be inferred based on visual input
Large Motion: introduce a multi-scale (res) framework (although motion large in original res. - small in downsampled region)
Multi-Scale Framework: corse-to-fine motion estimation (start estimation at scale 4 & then 3..2..1)
→ for next scale down - estimate incremental flow - using the estimated flow from last scale (final flow = sum increment. flows)
for scale l from coarse to fine:
→ initial estim. at scale l by upsampling flow estimate at prev. scale $l+1$: $u^{(l)} = 2u^{(l+1)}$ & $v^{(l)} = 2v^{(l+1)}$
→ compute the warped & shifted image: $J^l_{warped} = J^l(x + u^{(l)}, y^{(l)}_v)$
→ for image I^l & J^l_{warp} at this scale - compute image grads I_x, I_y, I_t: I_x, I_y from I_l & $I_t = J^l_{warped}(x,y) - I^l(x,y)$
→ estimate the incremental flow $(A^TA)^{-1}A^Tb$
→ update the flow at this scale: $u^{(l)} = 2u^{(l+1)} + u^\delta$ & $v^{(l)} = 2v^{(l+1)} + v^\delta$
Object Tracking: estimating the motion for an object of interest (rather than motion for pixels in Optic Flow method)
Lucas Kanade Tracker: define (u,v) for one object only rather than each pixel (assum. constant brightness & small motion)

Optimise SSD: $min_{u,v}E(u,v) = \sum_x\sum_y [I(x,y) - J(x+u, y+v)]^2$ (SSD of Motion from I to Image J)

→ **(optim. has same soln. as optic flow method - but summing over pixels in template image, rather than a small region)**
→ can always start from some template in the first time frame - estimate the displacement between two time frames - & propagate the location across a sequence of time frames
Correlation Filter: aims to maximise the correlation between template features & image features in next frame
→ more robust to illumination changes than SSD in Lucas Kanade (& does not need brightness constancy to hold & learns global features for template - unlike Lucas Kanade, which uses pixel intensities) (max. correlation features can be learned from CNNs)
Siamese Network: template & search window passed to CNNs (search window contains all possible locations template could move to) - CNNs learn some features & produce feature maps - perform correlation between two feature maps - network parameters trained to predict ground truth score map - maximal score indicates object location → trained on a large dataset of video sequences with annotates subjects of interest (must include various objects & scenarios to ensure generalisable & robust model)
Image Stabilisation: compute optical flow between successive frames using Lucas Kanade - identify & match keypoints between frames, estimating motion vector of keypoints, calculate motion trajectory of camera by integrating motion vectors over time, apply smoothing filter to remove jitter - compute transformation model to aligm frames - apply transformation

Hough Transform

want to obtain a parametric representation of line → two parameters provides much more efficient representation
Slope-Intercept Form: $y = mx + b$, m - slope, b - y-intercept
Double-Intercept Form: $x/a + y/b = 1$, a - x-intercept b - y-intercept
Normal Form: $xcos(\theta) + ysin(\theta) = \rho$, θ - angle from origin to closest point on line, ρ - distance (origin to closest point)
→ Derivation: x-int: $a = \rho/cos(\theta)$, y-int: $b = \rho/sin(\theta)$ & plug into double intercept form

Model Fitting: (m, b) could be estimated by minimising the fitting error: $min_{m,b}\sum_i [y_i - (mx_i + b)]^2]$

→ minimise SSD between the real y-coordinates of edge points & the estimated y-coordinates of edge points from model
Hough Transform: a Transform from Image Space to Parameters Space (i.e. from edge map to parameters of line)
→ Input Edge Map & Output Parametric Model - Each Edge Point Votes for Possible Parameters in Parameter Space
→ Rewrite $y = mx + b$ as $b = y - mx$ & Vote for Each Edge Point (i.e. First Point Votes $b = y_1 - mx_1$)
→ Image Space: (x, y) & Parameter Space: (m, b)
→ **Bins:** in practice - parameter space divided into 2D Bins - each point increments vote by 1 in Corresponding Bin
→ **Problem with Slope-Intercept Form:** Large Parameter Space - $m, b \in [-\infty, \infty]$ & need a lot of bins
→ **Solution:** use Normal Form - although $\rho \in [-\infty, \infty]$ - $\theta \in [0, \pi)$ (& in practice, ρ limited by image size)
→ **Vote Result:** point in parameter space with the most votes (i.e. intersection of three sinusoidal curves)
Line Detection by Hough Transform: use voting results from parameter space to detect lines in image space
→ **initialise the bins $H(\rho, \theta)$ all to 0 → for each edge point (X, y)**
. → **for θ from 0 to π** . → **calculate $\rho = xcos(\theta) + ysin(\theta)$** . → **accumulate $H(\rho,\theta) = H(\rho,\theta) + 1$**
→ **find points in parameter space (ρ, θ) where $H(\rho, \theta)$ is a local maximum & larger than threshold**
→ **the detected lines are given by $\rho = xcos(\theta) + ysin(\theta)$**
→ when multiple sinusoidal curves intersect in parameter space - indicates that multiple edge points align along same line
→ Hough Transform builds representation of all possible lines through edge point - & voting process accumulates evidence
→ take local maximum as only want lines of one-pixel width (similar to NMS in edge detection)
→ perform thresholding as to ignore lines with only a few votes (likely to be caused by noise/irrelevant features)
Model Fitting vs Hough: Model Fitting: One Line Only & Hough: Can Simultaneously Detect Multiply Lines
Circle Detection: → $(x-a)^2 + (y-b)^2 = r^2$
→ image space (x, y) is transformed into the parameter space (a, b, r) (large 3D search space & many bins)
If the Radius is Known: for each edge point (x, y) - only need to vote for possible values of (a, b) in 2D parameter space
→ (x, y) are known coordinates of edge point & (a, b) is unknown center of parameterised circle
→ rewrite as $(a-x)^2 + (b-y)^2 = r^2$ (circle centered at (x, y) with radius r in parameter space
→ **Vote Result:** point in parameter space with most votes (i.e. point at which circles overlapping)
If the Radius is Unknown: → 3D Paramaeter Space $H(a, b, r)$
→ set a range for the radius r & for each $r \in [r_{min}, r_{max}]$ (i.e. 1 to 10 pixels) → for each edge point (x, y)
. → vote for possible values (a, b) & accumulate $H(a, b, r)$
Parametric Form of Circles: $x = a + rcos\theta$, $y = b + rsin(\theta)$
→ **Acceleration:** if know angle θ (direction) from edge point (x, y) to circle center (a, b) - can more accurately vote in parameter space - **only voting along this direction**
Voting Along Gradient Direction: edge point from edge detection - providing gradient magnitude & direction - so θ is known
→ narrows down voting area in parameters space $H(a, b, r)$ → simply move along θ (or opposite θ) for a distance of r
→ **if do not know θ - vote to a whole circle & if do know θ - vote along gradient direction (in parameter space)**
Circle Detection by Hough Transform:
→ initialise bins $H(a, b, r)$ to all 0s → for each possible radius $r \in [r_{min}, r_{max}]$. → for each edge point (x, y)
. → let θ be the gradient direction . → calculate $a = x - rcos\theta$ & $b = y - rsin\theta$. → accumulate
→ find (a, b, r) where local maximum & larger than threshold → detected circles $x = a + rcos\theta$ & $y = b + rsin\theta$
Advantages of Hough: can simultaneously detect multiple lines, robust to noise (broken edge points still vote & contribute to line detection), & is robust to object occlusion (the remaining edge points still contribute to line detection)
Limitations of Hough: computational complexity can be high (as need to vote in 2D/3D parameter space for each edge point) - & need to carefully set parameters (such as parameters for edge detector / threshold for accumulator / range of circle radius)

Interest Point Detectors

Interest Points: points that we are interested in that are useful for subsequent image processing (efficient image representation)
Image Matching: establishing the correspondence between two images (i.e. combining two images of same object with some spatial transformation between them) (can be performed by pixels, using all information / by interest points, using little information)
Image Restoration: optimising a similarity metric based on all pixels ($max_T = Similarity(Image_A, Image_B(T))$)
→ Evaluating Similarity Metric for all Pixels is Computationally Expensive
Matching by Interest Points: find some landmarks that describe commonalities between images, find a common spatial transformation that maps landmarks in one image to second image (much less computationally expensive - not evaluating for all pixels)
Corner Detection: → for corner detection - shift a small window of pixels & calculate the change of intensities for the window
→ **edge: change of intensity in one direction / corner: change of intensity along both directions**
Harris Corner Detector: → if the window shifts by $[u, v]$ - the change of intensities is given by Sum of Squared Difference (SSD):

→ $E(u,v) = \sum_{(x,y)\in W} w(x,y)[I(x+u, y+v) - I(x,y)]^2$

→ Window Function can be Uniform or Gaussian (emphasis on central pixels)
→ by performing Taylor Exp. on $I(x+u, y+v) = I(x,y) + uI_x(x,y)_vI_y(x,y)$ & substituting SSD becomes:

$E(u,v) \approx [u, v] \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}$, $M = \sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$

→ can find Image Gradients I_x & I_y (x & y-direction) by applying Sobel Filter along x & y-axis for a 2D Image
→ M can infer the directions of change in an image & encapsulates structure of image in a local neighbourhood
→ **by analysing eigenvalues & eigenvectors of M - can deduce the directionality of edges within a window of an image**
→ if the image gradients are large - SSD/M will be large - & implies strong change in intensity - so corner/edge
For Diagonal M: (Diagonal Values of M determine if Flat/Edge/Corner)
Flat Region: M all 0s - no matter which values of u/v (which direction shifted) - no change in I_x or I_y - no change in SSD
Edge: M has one large diagonal element & one small diagonal element - large change of intensity/SSD in one direction - edge
Corner: M has both large diagonal elements - large change in intensity/SSD in both u & v - corner
Eigenvalue Decomposition: if M not diagonal - need to find Eigenvalues
→ $M = P\Lambda P^T$ (as M is symmetric $P^{-1} = P^T$ & eigenvalues are orthogonal)
→ the eigenvalues λ_1 & $_2$ represent magnitude of the change of intensity in directions of corresp. eigenvectors (columns of P)
Edge: if large change as shift along first eigenvector (large λ_1) - but small change as shift along the second eigenvector (small λ_2)
→ & vice-versa - large gradient in one direction & small gradient in perpendicular direction - edge
→ the direction of the edge is the direction of the eigenvector that has a small change in intensity (small eigenvalue)
Corner: large change along either eigenvector (both eigenvalues are large)
→ as the two eigenvectors are orthogonal to each other - [u, v] cannot be orthogonal to both eigenvectors & so a shift [u, v] will be in a direction containing one of the eigenvectors & change of intensity can be detected
$\lambda_1 \approx 0$, $\lambda_2 \approx 0$: **flat /** λ_2 **» λ_1 or λ_1 » λ_2: edge / λ_1 & λ_2 »: corner**
Harris: $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ (k is a small number, i.e. 0.05)
→ if only want values of $\lambda_1\lambda_2$ & $\lambda_1 + \lambda_2$ - do not need to perform eigenvalue decomposition
→ can use properties for determinant & trace - for $M = P\Lambda P^T$
$det(M) = det(P\Lambda P^T) = det(P)det(\Lambda)det(P^T) = det(\Lambda) = \lambda_1\lambda_2$ as $det(AB) = det(A)det(B)$
$trace(M) = trace(P\Lambda P^T) = trace(PP^T\Lambda) = trace(\Lambda) = \lambda_1 + \lambda_2$ as $trace(AB) = trace(BA)$
→ $R = det(M) - k(trace(M))^2$ (cornerness can be found by evaluating determinant & trace)
Harris Corner Detector: → **compute x & y derivatives of an image ($I_x = G_x * I$ & $I_y = G_y * I$) (G can be Sobel Filter)**
→ **at each pixel - compute the matrix M & calculate the detector response**
→ **detect interest points whose R above a threshold & which are local maxima (NMS: response higher than 4 neighbours)**
→ also finds strong responses at blobs & textures → is rotation-invariant (same SSD when shifting window) - but not scale-invariant

Object Detection

→ **predict set of bounding boxes (x, y, w, h) & labels for object** (applying image class. model to different regions of image)
→ at each location perform classification & localisation (predicting bounding box coordinates) (both with same conv. network)
→ too expensive to apply conv. net. to each pixel - perform class. & local. on feat. map output of conv. net. (much smaller res.)
Faster R-CNN: performs class. & local. on feat. map in two stages: Region Proposal Network (RPN) & Detection Network
RPN: applies to each pixel in conv. feat. map to propose possible regions of interest (applying 3x3 slid. wind.)
→ using feat. in window - perform bin. class. (0/1 not int./int.) & predict obj. size (k preds. for diff. scales & asp. ratio)
→ **Anchors:** predictions (0/1) of varying size (scale & asp. ratio)

→ **Loss:** $\sum L_{cls}(p_i, p_i^*) + \lambda\sum p_i^* L_{loc}(t_i, t_i^*)$ (weighted sum of class. & local. loss - p_i^* is ground truth predictor)

Detection Network: takes proposed region with more accurate feats. & performs multi-class.(without anchors - rough size known)
ROI Pooling: normalises size of proposals (through resampling) - mapping proposal feats. to standard fixed size
→ each proposal (i.e. 32x32) overlayed onto feature map (giving 32x32xD feature array)
→ performing interpolation to transform to a normalised 7x7xD feature map & passing mapped features to classifier (i.e. NN)
→ for each ROI - classifiers in the detect. net. uses normalised feat. map to predict label class & refine bound. box estimate
→ **RPN moves 3x3 Window & Binary Classifies, Proposes Region, Detect. Net. takes Proposed Feats., Pools to Fixed Window, & Multi-Class Classifies, then Refining the Bounding Box**
One-Stage Detection: changes binary class. loss func. in stage 1 to multi-class loss func. (directly perform. local. & multi-class.)
SSD: at each location on feat. map (found by sliding window) - evaluates a set of boxes at different scales & aspect rations - performing multi-class classification & localisation (faster R-CNN, but binary class. in stage 1 is multi-class class.)
→ faster R-CNN is more accurate (refinement) & slower / SSD is much faster & less accuracy (trade-off for accuracy & speed)

Scale-Invariant Detectors

Scale-Adapted Harris Detector / Laplacian of Gaussian (LoG) / Difference of Gaussian (DoG)
→ can apply Harris Detector at multiple scales & find cornerness response at most suitable scale
Getting Images at Multiple Scales: → Gaussian Smoothing with Different σ (variance/kernel size)
→ Small σ - can detect small objects with fine details & large σ - can detect larger objects (more blurred)
→ Sampling (Downsampling or Upsampling) with Different Spatial Resolutions (Different Sampling Factors)
How to Determine Scale at Each Pixel: Harris Detector gives High R for Regions Most Corner-Like - Select Scale with Highest R
→ Problem: Direct Comparison of Harris Detector R Across Scales may not be fair
→ the R depends on the eigenvalues of M, the eigenvalues of M are determined by the gradients $I_x(\sigma)$, $I_y(\sigma)$ which are inversely proportional to scale - **the larger the scale σ, the smaller the magnitude of the gradient, smaller M, & smaller R**
→ consider $f(x)$ & $g(x)$ where $f(x) = g(sx)$ (i.e. image of different zoom factors - more zooming = greater variance)
→ when calculating the derivative/gradient along the x-dimension - the peak magnitudes differ (with same shape) - & so comparing gradient magnitudes is not fair (the larger the scale - the smaller the magnitude of the gradient & the smaller the response)
$df/dx = s \;\; dg/dx$ (magnitude of derivative of f & g differ by scaling factor s)
→ **to make the derivative magnitude comparable - need to multiply derivative by scale, s**
Scale-Adapted Harris Detector: multiply image gradient by scaling factor σ (variance of gaussian filter/size of gaussian kernel)
→ σ is called a normaliser - normalised the image gradient & allows them to be comparable

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

→ calculate scale-adapted detector R for a series of σ (small to large scale), at each pixel - determine the scale which gives largest R, for interest point detection - look for maxima across both scale & space
→ **for each scale σ**
. → **perform gaussian smoothing with σ**
. → **calculate the x & y derivatives/gradients of the smoothed image $I_x(\sigma)$ & $I_y(\sigma)$**
. → **at each pixel - compute the matrix M (multiplying by σ^2**
. → **calculate the detector response $R = \lambda_1\lambda_2 - k(_1 + \lambda_2)^2$**
→ **detect interest points which are local maxima across both scale & space & R above threshold (scale-space extrema)**
Laplacian: sum of second derivatives - $\Delta f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

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

→ Second Derivative is more Sensitive to Noise than First Derivative & LoG must perform Gaussian Smoothing first
LoG: first convolve the input f with a Gaussian h & then calculate the Laplacian (calculates the sum of second derivatives)

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

→ & as the 2D Gaussian is formulated as $h(x,y) = 1/2\pi\sigma^2 \; e^{-x^2+y^2 \,/\, 2\sigma^2}$

→ $\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$ LoG performs Convolution with this Kernel (High Weight at Center)

→ can then detect local maxima & perform thresholding on response map to detect interest points
LoG Response at Scale σ: $LoG(x,y,\sigma) = I_{xx}(x,y,\sigma) + I_{yy}(x,y,\sigma)$ (approximated by DoG)
Normalised LoG Response at Scale σ: $LoG_{norm}(x,y,\sigma) = \sigma^2(I_{xx}(x,y,\sigma) + I_{yy}(x,y,\sigma))$
DoG: $DoG(x,y,\sigma) = I * G(k\sigma) - I * G(\sigma)$ (difference between two Gaussian-Smoothed Images, i.e. k=$\sqrt{2}$)

Descriptors & SIFT

Simple Descriptors: Pixel Intensity, Patch Intensities, Gradient Orientation, Histogram (Advantages & Disadvantages)
Pixel Intensity: Describe Features Simply using the Intensity of a Single Pixel
→ **Sensitive to Absolute Intensity Value**: intensity of same point will change under illuminations (i.e. same image with different illuminations will change intensity values of single pixels) & therefore will not be able to compare & match interest points
→ **Not Very Discriminative**: the greyscale intensity ranges from 0 to 255 - there are thousands of pixels with the same intensity
→ **a single pixel does not represent any local pattern** (i.e. pixel at edge / blob etc.)
Patch Intensities: Describe Features using Intensities of a Patch of Pixels
→ **Better Than a Single Pixel**: represents the local pattern (i.e. uniform region / edge at certain orientation)
→ performs well if the images are of similar intensities & roughly aligned (as can match patches of two images)
→ **Sensitive to Absolute Intensity Values** (& so Changes of Illuminations)
→ **Not Rotation Invariant**: after rotation - patch will also be rotated & original matching patches no longer match
Gradient Orientation: Describe Features using the Gradient Orientation of a Patch of Pixels
→ **Not Sensitive to Intensity Changes** → **Not Rotation Invariant**
Histogram: describes features using a histogram of pixel intensities in a patch
→ for each input patch - draw a histogram which divides pixel intensity into bins & counts intensity of pixels (i.e. 0-40, 40-80..)
→ **Robust to Rotation (Rotation-Invariant)**: count into each intensity bin remains the same
→ **Robust to Scaling (Scaling-Invariant)**: as normalisation is performed by diving the no. of pixels in each bin by total no. pixels
SIFT: combines advantages of gradient orientation (robust to change in intensity) & histogram (robust to rotation & scaling)
→ **use a histogram to describe the gradient orientations** → providing detection & description of local features
→ transforms an image into a large set of interest points - each of which is described by a feature vector
SIFT Algorithm: Detection of Scale-Space Extrema, Keypoint Localisation, Orientation Assignment, Keypoint Descriptor
Detection of Scale Space Extrema: DoG Filter used to Search Across Scale & Space to Identify Potential Interest Points
→ calculate the DoG for different scales σ - from fine to coarse (i.e. σ, $\sqrt{2}\sigma$, 2σ, $2\sqrt{2}\sigma$, 4σ, ...)
→ implemented by first calculating a stack of guassian smoothed images & then calculating the difference between successive σ's
→ i.e. calculating stack $G(\sigma)$, $G(k\sigma)$, $G(k^2\sigma)$, $G(k^3\sigma)$, $G(k^4\sigma)$ & finding difference (DoG) between successive G
→ i.e. a stack of 5 G gives 4 DoG Response Maps ($DoG(\sigma)$, $DoG(k\sigma)$, $DoG(k^2\sigma)$, $DoG(k^3\sigma)$, $DoG(k^4\sigma)$)
→ stacking response maps form a 3D feature space (2D space + scale dimension)
→ **X is detected as an extreme point if a local extreumum both along scale dimension (appropriate scale) & across space**
→ Negative Resp. & Min.: Light Blob on Dark Background → Positive Resp. & Max.: Dark Blob on Light Background
Keypoint Localisation: Refine Localisation & Scale by Fitting a Model onto Nearby Data & Trying to Find Sub-Pixel Location to Achieve the Max/Min Value for These Features (achieving sub-pixel accuracy)
→ a quadratic function is fitted to the DoG Response of neighbouring pixels (can easily find local extrema of quadratic function)
Fitting the Quadratic Function: denote DoG Response as $D(x)$ where $x = (x, y, \sigma)^T$ (3D Vector with both location & scale)
→ assume at each location x can move by Δx & calculate feature map at $D(x + \Delta x)$
→ this quadratic function $D(x + \Delta x)$ describes the feature function $D(x)$ in the local neighbourhood
→ **by Taylor Expansion (to second order):** $D(x + \Delta x) = D(x) + \delta D^T/\delta x \; \Delta x + 1/2 \; \Delta x^T \; \delta^2 D/\delta x^2 \; \Delta x$
→ D is response map of DoG (known), x is initial state of local maximum (known), Δx is shift to refined estimate (unknown)
→ the first derivative of response map of DoG D can be estimated using finite difference (looking at response map & moving x by one pixel either side, looking at difference, & dividing by two) → the second derivative also estimated using finite difference
To Estimate Refined Extrema for this Function: $\delta D(x + \Delta x)/\delta\Delta x = \delta D/\delta x + \delta^2 D/\delta x^2 \; \Delta x = 0$
→ therefore, $\Delta x = -\delta^2 D^{-1}/\delta x^2 \;\; \delta D/\delta x$
→ as $x = (x, y, \sigma)^T$ - we get a refined estimate for both location & scale (sub-pixel & sub-scale accuracy)
Orientation Assignment: determining the dominant orientation for a keypoint
→ to make SIFT invariant to scale & rotation, need to know scale & orient. at keypoint - scale known - find dominant orientation
→ the gradient orientations of all pixels in a neighbourhood vote for the dominant orientation (an orientation histogram with 36 bins covering 360 created - each pixel votes in the orientation bin, weighted by gradient magnitude - the keypoint will be assigned an orientation corresponding to the maximum of the histogram)
Keypoint Descriptor: Describing a Feature (Vector) for Each Keypoint
→ for patch centered at x with 16 subregions, in each subregion: sample 16 points & calculate gradient orientations
→ gradient orientation of each point ϕ must be adjusted by θ (dominant orientation) - i.e. $\phi - \theta$ (& SIFT Rotation Invariant)
→ calculate histogram for each subregion with 8 bins (orientation: 0-45, ...) - each bin sums gradient magnitude for an orientation
→ uniform regions will have small arrows & edges will have strong arrows in particular direction
→ Feature Vector has 128 Elements for 16 Subregions x 8 Bins (128x1 Vector)
SIFT Feature Descriptor Robust to Scaling/Rotation/Change of Illumination, as: sampling window proportional to scale, considering dominant orientation, use of gradient orientations for feature description, use of histograms
Keypoint Matching: Keypoints Between Two Images are Matched by Identifying the Nearest Neighbours
→ the distance is defined by the euclidean distance of SIFT descriptors
→ **after finding corresponding keypoints - want to find spatial transformation that relates the two images (assumed affine)**

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

→ 2 Equations of 3 Unknowns (more unknowns than equations & overdetermined system) (cannot solve - find least square solution)
→ which can be written as a linear system $Am = b$ with least square solution: $m = (A^TA)^{-1}A^Tb$ (min $||Am - b||^2$)
Smallest No. Interest Point Pairs: Unknowns: Scale Factor (if present), Rotation Angle, Tx, Ty (4: 4 DoF) - 1 Interest Point Pair = 2 Equations - Want 4 (No. Unknowns) Equations to Find Estimate = 2 Interest Point Pairs
RANSAC: outliers may exist in matching - want to improve robustness of matching - consider outliers in model fitting
→ **for each iteration: select random samples - fit model - check no. of inliers - terminate after no. iterations/enough inliers**
RANSAC for Keypoint Matching: find Eucl. Dist. between Feat. Vects. & Pair Keypoints - then RANSAC finds Am=b soln.
→ **in each iteration: estimate the unknown parameters m based on a random set of interest point pairs & check no. inliers (using affine transformation equation)** → **for correct transformation to be found - all point pairs in random set must be inliers**
Applications of Keypoint Matching: Image Stitching, Panoramas, Image Blending (Weighted Average of Overlapping Regions)

Faster Feature Description

Accelerating SIFT: SURF / BRIEF & **Describing Feature for Image:** HOG
Speed of SIFT: calculating gradient magnitudes & orientations can be slow (for real-time performance)
Faster Feature Description: use faster hardware (FPGA) / improving software & algorithm (decompose pipelines into steps, evaluate cost for each step, improve each step)
Pipeline for Image Matching: Feature Detection with DoG - Feature Desription with SIFT - Interest Point Matching
Faster Feature Detection: use seperable filtering, downsample, truncated Gaussian Kernel
Faster Feature Description: evaluate gradient orientations faster, describe local content without histogram of grad. orient.
Faster Interest Point Matching: approximate nearest neighbour (rather than Euclidean Near. Neigh.), use lower-dim. feat. vector
SURF: finds grads. along horizontal & vertical directions using haar wavelets (gradients in x & y for each sample point in subregion)
→ do not need to calculate gradients for arbitrary orientations (as done in SIFT) & can be 5x faster
Haar Wavelets: simple filters d_x & d_y applied to each sample point in a subregion to calculate gradients in x & y (intensity different between two halves in either x or y-direction) (evaluate image grads. by applying weight +1/-1 to either half & summing)
→ only sum haar wavelet response over all sample points in subregion (much faster) - rather than finding grad. mag & orient.

Descriptor: $(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$ (summation/absolute summation of Haar Wavelets)

SIFT: 16x8=128-D (16 grad. orient. of 8 bins) / SURF: 16x4=64-D (16 subreg. of 4 No. Descriptor) - SURF much faster & efficient
→ each dimension in a FP Number (4 Bytes) & to compare to feature vectors - calculate Euclidean Distance
Further Shorten Descriptor: quantise/binarise FP Number (convert into discrete number/convert into binary)
BRIEF: rather than comparing two halves to each other & calculating their distance (a FP Number) as in SURF - BRIEF compares intensities of a point p to another point q - giving a binary value as output
→ do not need to compute difference between intensities - just use binary to represent which intensity greater
→ no subregions - only binary tests & comparisons
→ randomly sample n_d pairs of points (p, q) for binary test - perform binary tests on pairs & get 0 or 1
→ the BRIEF descriptor is a n_d dimensional bit string - containing the results of binary tests
→ to match interest points - compare difference between bitstrings
→ the random sampling pattern around an interest point is determined once & then applied to all interest points
→ computation is much faster as comparing two numbers - no gradient orientation or intensity difference
Hamming Distance: when comparing two BREIF descriptors - can measure difference using Hamming Distance - computed effectively using bitwise XOR operation & bitcount (can achieve 40x speedup over SURF, 200x over SIFT)
→ if no rotation/scaling involved - BRIEF achieves performance comparable to SURF in interest point matching accuracy
→ BRIEF ignores rotation & scale invariance - assumes images taken with moving camera only involving translation
HOG (Histogram of Orientated Gradient): describes feature of entire image/large region (using gradient orientation histograms)
→ divides large region into dense grid of cells, describes each cell, concatenates these local descriptions to form a global description
→ multiple cells form a block, gradient orientation histogram describes this block, & block moved through image
→ there is an overlap between blocks & for each block (4 cells) - the descriptor vector v (concat. of four histograms) is normalised

Locally Normalised Descriptor: $v_{\text{norm}} = v/\sqrt{\|v\|_2^2 + \epsilon^2}$ (ϵ - small value) → concatenate these to form global desc.

→ allows performing image class. based on image feats. & can detect if region contains features/retrieve similar images by features
→ HOG robust to illum. changes, invariant to contrast changes (normalis.), & can be rotation-invariant if dominant orient. used
Image Retrieval: find HOG descriptors, store descriptors, at retrieval time compare Euclidean Distances (similarity), match images

Classifiers

Image → **Feature Extraction** → Feature Representation → **Classifiers** → Output Label
Feature Extraction: transforms input images into low-dimensional vectors that can be easily compared & matched
Split Dataset into Two: Training Set (training classifier) & Test Set (evaluating performance of classifier on unseen data)
→ Hand-Crafted Features (i.e. pixel intensities / HOG) / Learnt Features (CNN / automatically learn by algorithm)
Classifiers: KNNs, SVM, Neural Network, Vision Transformer
KNNs: compute KNNs & assign test datapoint to class by majority voting
→ **Finding the Best K:** train classifier using training set & search parameters on a validation set (separate from held-out test set)
→ **Advantages:** no training, simple & effective, can perform multi-class classification
→ **Disadvantages:** all training data must be stored & searched (expensive for large datasets)
→ at training time - fine with slow training, at test time - want faster inference (KNN slow inference due to search & high cost)

→ **Distance:** Euclidean: $D = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$ LP Norm: $D_p(x, y) = (\sum |x_i - y_i|^p)^{1/p})$

Linear Classifier: simplest form of SVM - a line that separates two different classes - for 2D case, line: $w_1x_1 + w_2x_2 + b = 0$
→ assigns a class c to data x: $c = 1(\text{if } w_{1x1} + w_2x_2 + b \geq 0)$ & $c = -1$ (otherwise)
→ general case: $w \cdot x + b = 0$ (in HOG, x would be concatenation of Histogram of Gradient Orientations for all large regions)
→ Linear Classifier can discard training data once w & b found - much faster inference than KNN at test time
→ selecting w & b: selecting the best separating hyperplane (line that is far from both classes/maximum margin hyperplane)
→ to determine the maximum margin hyperplane - only need innermost points - **support vectors**
→ want the support vectors to fulfill the equations $wx + b = 1$ & $wx + b = -1$ - such that all points easily classified
→ want to maximise the margin between $wx + b = 1$ & $wx + b = -1$
→ **maximum margin:** let $wx_1 + b = -1$, move x_1 along n for distance of margin m, arrive at x_2 on $wx_2 + b = 1$ → $w(x_{1\,m}n) + b = 1$, $mw \cdot n = 2$, $n = w/||w||$, & thus $m = 2/||w||$ where $||w||$: L2 Norm
SVM: aims to maximise margin between support vectors, w.r.t unknown parameters w & b & find best separating hyperplane
during training: $max_{w,b} 2/||w||$ s.t. $w \cdot x_i + b \geq +1$ if $y_i = +1$ & $w \cdot x_i \leq -1$ if $y_i = -1$
during classification: for a new unlabelled point, predicts whic hside of the hyperplane the point lies on
Reformulated as Optimisation: $min_{w,b}||w||^2$ s.t. $y_i(w \cdot x_i + b) \geq 1$
→ quadratic optimisation problem - subject to linear constraints - can be optimised analytically or using gradient descent

Further Reformulated: $min_{w,b}L(w,b) = ||w||^2 + C\sum max(0, 1 - y_i(w \cdot x_i + b)$

→ term in summation is called the hinge loss & is loss function optimised by gradient descent

Gradients: $\nabla_w L = 2w + C\sum \nabla h$ & $\nabla_w h = -y_ix_i$ (if $y_i(w \cdot x_i + b) < 1$) & $= 0$ (otherwise)

Gradient Descent: $w^{(k+1)} = w^{(k)} - \eta_w L(w^{(k)}, b^{(k)}) = w^{(k)} - \eta(2w^{(k)} + C\sum \nabla_w h)$ where η : learning rate

→ HOG used for Feature Extraction & SVM for Classification - $wx + b = 0$, x with HOG & params. w/b using grad. des.
One vs Rest: (Multi-Class) decompose to multiple binary class. problems (between class 1 & others, class 2 & others...) - during testing, apply all classifiers to test data - classifier which produces highest resp (data further away) determines result
One vs One: train a classifier between each pair of classes (class 1 & 2, class 1 & 3, ...) - for K classes, K(K-1)/2 classifiers
→ at test time - each classifier votes for a class & count the vote for each class, performing majority voting
Underfitting: Classifier too simple to capture complex features & not trained for enough epochs
Overfitting: Classifier learns training data too well, including noise, trained for too many epochs
Class Imbalance: some classes with much higher no. in data, training biases towards that class
Precision: $TP/(TP + FP)$ & **Recall:** $TP/(TP + FN)$
(TP: Items Correctly Detected as Positive Class, FP: Items Wrongly Detected, FN: Missed in Detection but Positive)
Threshold: to evaluate Precision/Recall of Classification, Threshold Set (true images have a probability value greater than threshold)
Precision-Recall Curve: calculate precision & recall for possible values of threshold & plot (precision: y & recall: x)
Scale-Robust SVM: use RPN to generate region proposals of diff. sizes, pass to SVM classifiers to determine if region contains object (reducese no. windows processes by SVM & increases detection speed) vs. image pyramid & slid. window (+ NMS/Threshold)
Handling Overlapping Detection Results: use NMS (keeps only the bound. box with highest class. score for each detect. object)
Class. Accuracy: TP+TN/Total Preds **Localisation Acc.:** IoU, Overlap A/Union A (Pred. Bound. Box & Grouth T Bound. Box)

Neural Networks

Sigmoid Activation: $f(x) = 1/1 + e^{-z}$ **Perceptron:** single layer & step activation ($y = 1$ if $wx + b > 0$)
MLP: several layers of neurons - output of a neuron layer is input to next layer - optimised with backpropagation
→ **Improvements:** deeper / better hardware / larger datasets / activation functions / data augmentation / data normalisation

MSE Loss: $1/M\sum 1/2||a_m - y_m||^2$ (minimise MSE between output & ground truth label)

→ use backpropagation to calculate the gradient & perform stochastic gradient descent for optimisation of W & b
Binary Classification: one output neuron, uses sigmoid activation function in output layer & cross-entropy loss
Multi-Class Classification: for K classes, K output neurons - use softmax activation & cross-entropy loss

Cross-Entropy: $= -\sum p_i log(q_i)$ & **Cross-Entropy Loss:** $= -\sum y_i log(f(z_i))$ (y: true probability, f(z): estimate)

Softmax Activation: $f(z_i) = e^{z_i}/\sum e^{z_k}$ **Error Rate:** 100% - Classification Accuracy

Generating Augmented Samples: apply affine transformations to original samples (translation/scaling/squeezing/shearing)

Preprocessing Object Detection with HOG + SVM: Resizing (ensure all images are same size to give feature extraction consistency), **Greyscale Conversion** (HOG Features often extracted from Greyscale to reduce computational complexity & focus on structural information), **Noise Reduction** (apply Gaussian Filter to smooth image & improve performance of gradient calculations used in HOG Descriptor), **Normalisation** (apply global image normalisation to adjust dynamic range & constrast - reducing influence of lighting variations), **Edge Detection** (can improve HOG effectiveness), **Data Augmentation** (improves generalisation), **Splting Dataset** (training, validation, test)

CNNs

Limitations with MLP:
→ **uses many parameters** - even for 3-Layer Network (784-300-10) uses 784x300 + 300x10 = 238,200 weights
→ **may not scale up to bigger images** - for a 224x224 RGB image (224x224x3 channels) = 150,527 Neurons for Layer 1
→ **2D image considered as a flattened vector - without considering 2D nature** - need appropriate operator for 2D images
CNNs: assume that the inputs are images & encode certain properties (local connectivity / weight sharing etc.) into the architecture - which will make the computation more efficient & substantially reduce the no. of parameters
→ each Neuron only sees a small local region in the layer before it (called the Receptive Field)
Local Connectivity: a Neuron only sees a small local region in layer before it (in MLP - a Neuron sees all Neurons in Prev. Layer)
Weight Sharing: each local region shares the same weights to the convolution layer
→ **Local Connectivity + Weight Sharing substantially reduces no. of parameters**
Convolutional Layer: Input: XxYxC (X & Y: Dimensions, C: No. Channels)
→ Each Neuron Depends on XxYxC Cube of Input (& has XxYxC Parameters + 1 for Bias)

→ Neuron Output: $z_d = \sum_{ijk} W_{ijk}x_{ijk} + b, \quad a = f(z_d)$ (i/j/k subscripts for X/Y/C dimensions)

→ for Multiple Neurons in Convolutional Layer - Neurons Have Different Weights & Biases (Different Connections)
→ adding more neurons forms a 1x1xD cube of output (D: Depth) & no. parameters becomes XxYxCxD + D (weights + bias)
→ moving the small window across the image: output cube of XxYxD - each window has the same weights (weight sharing)
→ 2D Convolution Kernel has Four Dimensions: Kernel Width/Height & Input/Output Depth
→ the Output Cube is the Feature Map, 3 Dimensions: Width/Height, Output Depth
Padding: used to keep the size of the original image in the output
Stride: move the window faster & look at a downsampled grid (information of a large image to be condensed to small region)
Dilation: increase the receptive field of original image without increasing no. of parameters
Pooling Layer: an operation to make feature maps/representations smaller
→ similar to image downsampling, after pooling - neurons in next layer can see larger region of image
Max Pooling: select max value in each window for output feature map (combines information of multiple pixels & condenses large images) (convolution & pooling layers obtain a feature map - fully-connected layers condense feature map into output)
Deep Neural Nets: efficient use of GPUs / large datasets / improvement of optimisation & network architecture
Exploding Gradient: in optimisation - gradient becomes very large - preventing the algorithm from converging
→ **Solution: Gradient Clipping**, clip by value (min & max gradient value) / clip by norm ($g = g/||g|| \; v$, if $||g|| > v$)
→ minimise loss taking small steps to minimum point - in vicinity of cliff gradient very large & overshoots - clipping prevents
Vanishing Gradient: gradient becomes vanishing small - preventing weights from changing values
→ occurs if sigmoid activation used - $f'(z) = f(z)(1 - f(z))$ - if f(z) saturates to 0 or 1 - f'(z) becomes almost 0
→ causes problem in backpropagating the gradient - solved using a Leaky ReLU
Leaky ReLU: $f(z) = 0.01z(\text{if } z < 0) = z(\text{if } z \geq 0)$
Exponential Linear Unit (ELU): $f(z) = a(e^z - 1)(\text{if } z < 0) = z(\text{if } z \geq 0)$
Parametric ReLU: $f(z) = az(\text{if} z < 0) = z(\text{if} z \geq 0)$ (a is a learned parameter)
Size of Feature Map (Conv): ((input size - kernel size + 2 x padding) / stride) + 1 **(Pool):** ((input size - kernel size)/stride) + 1
Data Size of Feature Map: feature map dimensions x no. filters x 4 bytes

Receptive Field Size: receptive field of previous layer + (kernel size - 1) + $\prod$ strides of all prev. layers

→ **receptive field of a neuron represents the size of the region in the input image that can affect this neuron**

Segmentation Methods

Image Segmentation: a process of partitioning an image into multiple regions - each region consisting of pixels with similar properties (i.e. intensity/colour/texture) or semantics (person/car/building) **(pixel-wise classification problem)**
→ **Unsupervised (no training data/easy/fast):** thresholding, K-Means Clustering, Gaussian Mixture Model
→ **Supervised (training data needed):** CNNs
Thresholding: converts greyscale image into a binary label map (only parameter is threshold & assumes pixel intensities can be grouped into two clusters, with a threshold to seperate) → at each pixel x - the label is: $f(x) = 1$ if $I(x) \geq$ threshold / or 0
K-Means Clustering: iteratively estimate cluster parameters & pixel associations **(for general cases with more than 2 clusters)**
→ initialise the cluster center μ_k $k = 1, 2, \ldots, K$ & for each iteration:
. → compute $\delta_{x,k}$ for each point - assigning x to nearest cluster center μ_k → update μ_k according to the membership $\delta_{x,k}$
. → repeat until $\delta_{x,k}$ no longer changes or the maximum no. of iterations reached
→ **selects the optimise cluster centers that minimise the intra-class variance (MSE between point & center of cluster)**

→ $min\sum_K\sum_x \delta_{x,k}(x - \mu_k)^2$

→ can also be performed on (other than intensities): colour, position + colour, features (x becomes a feature vector - not scalar)
GMM: performs a soft assignment by assuming a gaussian distribution for each cluster (K-Means is hard assignment, 0 or 1)
→ Probability that x_j belongs to cluster k: $= \pi_k \times 1/\sqrt{2\pi}\sigma_k \;\; e^{-(x-\mu_k)^2/2\sigma_k^2}$
→ x_j : intensity for data point j / j_j: class for data point j / π_k, μ_k, σ_k : parameters for cluster k
→ initialise the parameters π_k, μ_k, σ_k for $k = 1, 2, \ldots, K$ & for each iteration
→ compute the membership for each datapoint x_j (cluster with highest probability) & update the gaussian mixture model

→ $\pi_k = \sum_j P(y_j = k)/N$ (proportion of cluster k) & $\mu_k = \sum_j P(y_j = k)x/\sum_j P(y_j = k)$ & $\sigma_k^2 = \sum_j P(y_j = k)(x - \mu_k)^2/\sum_j P(y_j = k)$

→ repeat until parameter change is trivial or maximum no. of iterations reached (can then assign class to each pixel by max. prob.)
GrabCut: iteratively - estimate GMM parameters for two clusters & perform segmentation with smoothness constraint
CNNs: train model that first extracts convolutional features from the input images & then classify each pixel into some classes
→ since segmentation is pixel-wise classification - apply classification model to each pixel of the image
→ classify the local region & assign the classification value to the center pixel - repeating for every pixel in the image
→ **Concerns:** expensive to apply NN to each pixel / need a NN that outputs a pixel-wise probability map, rather than a single probability vector for the entire image / need to transform the classification network into a segmentation network
→ **transform fully connected layers of CNN into convolutional layers to generate a pixel-wise probability map**
→ pixel-wise probability map has a downsampled size of the original image (& performs segmentation at a coarse resolution)
Downsampling in CNN: strided convolution & max pooling / **Upsampling in CNN:** transposed convolution (& restore original size)
Convolution (Stride 2): generates a downsampled feature map by a factor of 2 (i.e. 8x8 to 4x4)
Transposed Convolution (Stride 2): each pixel in the input contributes to a 3x3 window in the output with some weights (generating an upsampled feature map by a factor of 2, 4x4 to 8x8) (stride of 2 as output window moves by 2)
→ can use different depth in transposed convolutional layer, may want to reduce depth for larger feature map to fit into memory
Training: at each pixel, define class. loss (cross-entropy) - seg. loss is average class. loss for all pixels - train by optimising seg. loss

Improving Image Segmentation

FCN: details of the boundary not well delineated - due to upsampling (performs upsampling to original res. in single step)
To Accurately Delineate: combine local (pixel intens. / edges) & global features (what object is / context)
→ initial layers of NN extract local features (applied to image in original res. before downsampling/max. pooling) & deep layers encode global features (image has been downsampled to a smaller res. & each pixel in feat. map gives greater window in image)
Seg. Networks: input in original res. - passes through conv. laters & max pool. (downsampling) - smaller feature map - + transp. conv. layers to upsample back - upsampled pred. can be blurry - **want to combine local & global features when upsampling**
DeepLabv3+: extract local features & concatenate (in skip connection) with global features (from spatial pyramid pooling)
Skip Conn.: if concatenating XxYxD1 & XxYxD2 with 1x1 Conv - Kernel applied across 1x1 window, depth D1+D2, & weighted average taken across depth (& gives upsampling in two 4x steps - creating smaller factor & less blurriness in output)
Image Pyramid: repres. of image at multiple scales (res. levels) → more downsampling = smaller scale & represents more global
Multi-Scale Feature Extraction: conv. kernels with diff. sizes (larger kernel = more global) / conv. kernels with diff. dilations (larger region without incr. in params.)
Spatial Pyramid Pooling: extracts features at multiple scales & gives aggregated (concatenated/weighted sum) feat. map
→ uses dilated conv.: 1x1 Conv (Pixel-Wise Feat.), 3x3 (with Increasing Dilation Rate to Increase Scale of Feat.) & provides 4 feat. maps with different dilation levels that can be concatenated (using 1x1 Conv) into an aggregated feature map, with 4x depth
→ 1x1 Convolution: equivalent to performing weighted sum at each pixel level & aggregates multiple feature maps
U-Net: CNN with contracting & expanding path (encoder & decoder)
→ tries to improve resolution of segmentation map with skip level at every level & splitting upsampling into smaller factors
→ encoder decreases image size & increases no. channels/depth of feat. map (series of conv. layers & pooling to downsample)
→ deeper layers compensate for loss of spatial info. & allow network to maintain high repres. capacity
→ strided conv. is a learned downsample. approach & has parameters (more train. time) / max pooling is fixed operation
→ decoder increases image size & decreases no. channels/depth of feat. map (series of conv. & upconv/transpos. conv)
→ upconv. upsamples feature map & increases spatial dimensions, conv. refines features after concat.
→ final 1x1 conv maps deep feature maps to desired no. of outputs
HRNet: concats. features at four diff. resolution levels (first upsample. & then concat. & conv.) (learn relation. between levels)
UNet++: series of nested dense skip pathways connect encoder & decoder subnetworks (aggregation of features from different resolution levels) (features of multiple scales are fused multiple times) (provides more precise localisation & delineation)
UNETR: aggregates local features learnt by convolutions & global features learnt by transformer (self-attention of image patches)
Instance Segmentation: defines a segmentation for each instance (object detection followed by image segmentation)
Panoptic Segmentation: providing segmentation for each thing (instance seg. & stuff (seman. seg.) & combining seg. results