Image formation: Images represented as array of pixels, set of 3 numbers (rgb). Objects have colour because of reflected light – modelled with bidirectional reflectance distribution function (BRDF) – $f_r(\theta_i, \gamma_i, \theta_r, \gamma_r, \lambda) = \frac{dL_r}{dE_i}$. Cameras sense light one of two ways: CCD (Charged-Coupled Device) or CMOS (Commonly in cameras). Full RGB colour space – Gamut of human vision. sRGB is triangle subset of human gamut – cannot produce all human visible colours.

Image filtering

Moving average filter: every value in kernel is the same. [[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]] is 3x3. To perform signal process using the kernel, set centre pixel to index 0. Scan over each pixel in image and make center pixel equal to elem-wise multiplication of both. Generally, work in greyscale as there is only 1 channel. RGB would essentially work red, then green, then blue. Complexity of this, with image NxN, kernel KxK, is $O(N^2, K^2)$. We can separate the filter using **Convolution**. To convolute 2 matrices, flip the kernel along x and y axis, and scan over each pixel with the kernel. Element wise multiply and sum all values gives new value for centre. Exam will not ask for padding. After separation, $O(N^2K)$.

Gaussian Filter: $h(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$. Small values outside $[-k\sigma, k\sigma]$ can be ignored (k = 3 or 4). Also separable; $h(i,j) = h(i) * h(j)$. $h(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$. **High pass filter**: $Identity\ (just\ a\ 1\ in\ center) + [Identity - moving\ average]_{\{High\ Frequency\}}$. Processing an image with a filter is done by convoluting the image with the kernel. Definitions of convolution will likely be given in the exam. Probably asked to prove – remember that m-i, n – j can be subbed with i, j. Convolution is associative.

Edge detection

Looks at a brightness gradient – needs differentiation. Recall $f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$. For discrete we can use finite difference: $f'[x] = f[x+1] - f[x] = f[x] - f[x-1] = \frac{f[x+1]-f[x-1]}{2}$. **Prewitt Filter**: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}_x, \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}_y$.
Sobel Filter: $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}_x, \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}_y$. Both are separable: $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$. Input f: $g_x = f * h_x, g_y = f * h_y, g = \sqrt{g_x^2 + g_y^2}, \theta = \arctan\left(\frac{g_y}{g_x}\right)$. If h gaussian: $\frac{dh}{dx} = \frac{-x}{\sqrt{2\pi\sigma^3}} e^{-\frac{x^2}{2\sigma^2}}$. Canny Edge Detection: 1. Perform gaussian smoothing. 2. Use Sobel filter to calc grad mag and direction. 3. Apply non-maximum suppression to get single response for each edge. M(x, y) = M(x, y) if local maximum, 0 otherwise. 4. Perform Hysteresis thresholding to find potential edges – define t_{low}, t_{high} – if less than low, reject. If above high, accept as edge. If between then weak edge: if connected to existing edge accept, if it is not then reject. Learning based edge detection: ML algorithm assumes paired data (images x and manual edge maps y). Finds a model with parameters θ that maps x to y s.t. $y = f(x \mid \theta)$. Integrates from multiple scales with coarse-scale edges to form a final output. Canny Edge detection uses single scale for edge detection, controlled by σ.

Hough Transform

Lines can be parameterised many ways. $y = mx + b, \frac{x}{a} + \frac{y}{b} = 1, x\cos\theta + y\sin\theta = p\ where\ \theta_{angle}, p_{distance}$. Hough transforms image space (edge map) to parameter space (a line). Each edge 'votes' for possible models in the parameter space. One way is to fit a line model to the edge points (x1, y1), (x2, y2), … $\min(m,b) \sum_i \left(y_i - (mx_i + b)\right)^2$. Parameter space is too large for m and b as both are infinite. Thus, we can use normal form (theta and p). whilst p can still be infinite, $\theta \in [0, \pi)$. The transform will look different – sinusoidal. To perform we do the following: Initialise bins $H(p,\theta) = 0$. For each edge point (x, y): for $\theta = [0, \pi)$, calculate $p = x\cos\theta + y\sin\theta$ and accumulate $H(p,\theta) = H(p,\theta) + 1$. Find (p, θ) where $H(p,\theta)$ is local maximum and larger than threshold. Done similarly to Canny. Reduces false positives. Detected lines are given by $p = x\cos\theta + y\sin\theta$. For circles: 1. Init bins $H(a,b,r) = 0$. $\forall r \in [r_{min}, r_{max}]$, let theta be gradient dir, calculate $a = x - r\cos\theta, b = y - r\sin\theta$. Accumulate as before.

Interest point detection

Interest points useful for processing and analysis. Classification, matching, retrieval, etc. Commonly corners or blobs where local structure rich. Aka key points, landmarks, low level feats. **Harris Detector**. Corners are intersections of edges. For corner detection just looking at gradient mag of a single pixel isn't enough – could be along an edge. Using a small window can tell us if it is a corner, however. Flat: no intensity change in either dir. Edge – intensity change 1 direction. Corner both directions. Define window W and window function w (1 if in window, 0 if not), where we have sum of squared differences; intensity in shifted window and original intensity $E(u,v) = \sum_{x,y \in W} w(x,y)[I(x+u, y+v) - (I(x,y)]^2$. w(x, y) may use Gaussian to put more focus on centre of window. $I(x+u, y+v) = I(x,y) + u\,I_x(x,y) + v\,I_y(x,y) + \cdots$ I_x is image derivative along x, same for y. $E(u,v) = [u\ v] \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$. If M = middle section (everything encompassed by uv either side), we get some simple cases: $M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ for flat region, $M = \begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix}$ for edge (large on one of diag.), $M = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$ for corner. If M is not diagonal matrix we can use Eigen decomposition, where λ_x is an eigenvalue of M, $M = P \wedge P^T, \wedge = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$. P is a matrix of eigenvectors of M. If R tells us whether a pixel is a corner, then R can be defined a number of ways. $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ was defined by Harris & stephens, where k is a small number i.e. 0.05. $R = \min(\lambda_1, \lambda_2)$ from Kanade, $R = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2} + \epsilon$ from Noble. Note that $\det M = \lambda_1\lambda_2$, trace $M = \lambda_1 + \lambda_2$, therefore $R = \det M - k(\text{trace } M)^2$. Note that Harris detector is **rotation invariant**. It is not invariant to scale, however. Circles that have been scaled down may be detected as corners. At a given scale, we can see whether the Harris detector gives the highest point. If it looks most like a corner at scale σ, there should be high response. Scale adapted Harris Detector: $M = \sum_{x,y} w(x,y)\sigma^2 \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y(\sigma) \end{bmatrix}$. Use this at different scales, determine scale with largest response – we now have a scale dimension, so interest point is (x, y, σ). Algo: $\forall\sigma$, perform gaussian smoothing, calc x, y derivatives $I_x(\sigma), I_y(\sigma)$ respectively. Compute M at each pixel. Calculate R. Detect interest points which are local maxima across scale and space with R > threshold. **Laplace of Gaussian (LoG)**: Performs Gaussian smoothing followed by Laplacian Operator. $\Delta f = \nabla^2 = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}$. Laplacian filter = $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. Second derivative more sensitive to noise: LoG filter is $f * \left(\frac{\delta^2 h}{\delta x^2} + \frac{\delta^2 h}{\delta y^2}\right)$. Since we know 2D gaussian, we have $\frac{\delta^2 h}{\delta x^2} + \frac{\delta^2 h}{\delta y^2} = -\frac{1}{\pi\sigma^4}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$. LoG can be good interest point detector – needs scale multiplication (second der. So twice). $LoG_{norm}(x,y,\sigma) = \sigma^2(I_{xx}(x,y,\sigma) + I_{yy}(x,y,\sigma))$. **Difference of Gaussian (DoG)**: Defined as difference of gaussians with different scales (suggested k = $\sqrt{2}$). $DoG(x,y,\sigma) = I * G(k\sigma) - I * G(\sigma)$. Approximates Normalised LoG: $DoG(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G(x,y,\sigma)$.

Feature Description

Pixel Intensity: simply intensity of the pixel. Sensitive to Lighting conditions. Patch Intensities: Take some window representing local pattern. Images similar intensity range, roughly aligned, work well. Sensitive to absolute intensity values. Not rotation invariant. Gradient Orientation: not sensitive to intensity, but still rotation-variant. Histogram: take intensity histogram of patch. Binning intensity values, generating histogram from values. Robust to rotation and scale. Sensitive to intensity. **SIFT**: **1:**Detection of scale-space extrema: Search across scales and pixel locations, looking for interest points. DoG filter, scales continuously multiplied by $\sqrt{2}$. A point is of interest if it is local extremum along both scale and space. Both minima and maxima. If interest point is very bright, DoG < 0; vice versa for dark blob. **2:**Keypoint Localisation: initial SIFT simply locates key points at scale-space extrema. Improved version fits model onto nearby data improving accuracy. Quadratic func. can be fitted onto DoG response of neighbouring pixels, location and scale for extremum of QF can be estimated. Denote DoG response as $D(x,y,\sigma)$ or $D(\boldsymbol{x})$ where $\boldsymbol{x} = (x,y,\sigma)^\intercal$. From Taylor expansion we get $D(x + \Delta x) = D(x) + \frac{\delta D^\intercal}{\delta x}_{(1)}\Delta x + \frac{1}{2}\Delta x^\intercal \frac{\delta^2 D}{\delta x^2}_{(2)}\Delta x$. 1) first derivative estimated with finite difference: $\frac{\delta D}{\delta x} = \frac{D(x+1,y,\sigma) - D(x-1,y,\sigma)}{2}$. Second derivatives / Hessian also estimated with finite difference. Gives quadratic function for Δx, the shift to refined estimate. Therefore to find a refined extrema for the function: $\frac{\delta D(x+\Delta x)}{\delta\Delta x} = \frac{\delta D}{\delta x} + \frac{\delta^2 D}{\delta x^2}\Delta x = 0 \Rightarrow \Delta x = \frac{\delta^2 D}{\delta x^2}^{-1}\frac{\delta D}{\delta x}$. Shifting initial by Δx gives us **refined estimate**. This also gives us a refined scale, since it is 3D with a scale-dimension. **3:**Orientation assignment: Attempts to assign consistent orientation to each keypoint. Feature descriptors can be represented relative to this orientation, for rotation-invariance. We need to know the orientation of the feature, then we can perform sampling in a rotated coordinate system when we calculate features. We can create orientation histogram with 36 bins of 10°: each pixel in neighbourhood votes for orientation bin. We now have (x, y, σ), and orientation θ. We can draw samples using window size prop to sigma rotated by theta. **4:**Keypoint Descriptor: We need to describe the local image content. Histogram of Gradient Orientations. Made from calculated grad mags and orientation for sampling points. Calculated relative to dominant θ. In practise subregions are used, each with 4x4 samples. For each subregion, construct orientation histo with 8 bins, 45° each. Also represented with 8 arrows, length of arrow representing grad mag in given dir. In Lowe's impl, 16 subregions are used. This gives descriptor dim of 128 = 16 x 8. Each keypoint described with feature vector containing 128 elems. Robust to rotation due to consideration of dominant orientation. Similarly robust to scale since samples drawn from window prop to scale. Robust to intensity since grad orientations used for feat desc. **Keypoint Matching**: Matched by identifying nearest neighbours (Euclidean distance of SIFT descriptors). Each KP in image a identifies NN in database of KP's for image B. suppose we find kp (x, y) in A that corresponds with (u, v) in B. we assume relation via an affine transformation: $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$. With many pairs, we can extend the eq.. $\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ \dots & & \dots & & \dots & \end{bmatrix}\begin{bmatrix} m_1 \\ m_2 \\ \dots \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \dots \end{bmatrix}$. This can be written as a linear system, where m is the only unknown: **Am** = **b**. We can obtain least-square solution to linear system with Moore-Penrose inverse; minimising $\left|\left|\boldsymbol{Am} - \boldsymbol{b}\right|\right|^2; \boldsymbol{m} = (\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}\boldsymbol{A}^\intercal\boldsymbol{b}$. We have spatial transform between 2 images. **RANSAC**: squared diff sensitive to outliers. Points deemed corresponding that aren't. To ensure robust solution, we can use Random Sample Consensus: **1.** Randomly sample some points. **2.** Fit line along sampled points. **3.** Find number of outliers within threshold of line. **4.** Terminate if enough outliers found or we have reached certain iteration no. **SIFT** can be implemented on FPGAs or can be sped up using algorithmic techniques. 1. Feature Detection: no. of approaches to speed up including separable filtering, down-sampling or cropping Gaussian kernel. 2. Feature Description: can consider whether we can evaluate the Grad Orientation faster or use a different method. 3. Interest point matching: approximate nearest neighbour, use lower-dimensional feature vector. **SURF (Speeded-Up Robust Features)**: Only computes gradients along vertical and horizontal directions using Haar wavelets. Surf still uses same subregions and sample points, however, do not calculate gradients for arbitrary orientations. Two simple filters, d_y and d_x onto sample points. Summing pixel intensities with a weight of 1 or -1 is very fast, since its either addition or subtraction. For each subregion, sub haar wavelet responses over samples points, the descriptor for given region defined by 4 elems: $\sum d_x, \sum d_y, \sum|d_x|, \sum|d_y|$. If all 4 low, then homogenous region. If zebra-stripe like, then sum would be 0 but sum|d| would be large (in correct dir). If gradually increasing, then both sums in correct direction will be high. SURF still uses 16 subregions, with 64 dimensionality for each interest point. Approx. 5x faster than SIFT, due to simpler Haar wavelets. SURF performs as well as sift in matching. **BRIEF:** We can further shorten the descriptor by quantisation or binarization. Haar wavelets compare local region to another, giving float difference. BRIEF will give binary value (1 if q > p): $\tau(p,q) = 1\ if\ I(p) < I(q);\ \ 0\ \text{otherwise}$. BRIEF samples n_d pairs randomly for bin tests, random pattern determined once, same pattern applied to all interest points. N_d=256, then we get 256 tests, 1 bit output = (32 bytes, SIFT uses 128, SURF 64.) Fast to compute due to bit shifting. Uses less memory and compares only 2 numbers. Avoids calculating Euclidean distance, as we use Hamming distance (number of bits that differ). Efficient calculation as we can just XOR descriptors and take bit count. NOT rotation or scale invariant. Assumes images taken are only translational-movement. BRIEF is approx. 40x faster than SURF. Comparable to surf so long as rotation/scale is constant. Image similarity can be defined by number of matching points. If two images are similar, there should be large no. of matchable interest points. *VisualRank*. **HOG**: Histograms of Oriented Gradient extend idea of feature descriptors of large region or whole image. HOG divides large region into dense grid of cells, describes each cell, and concatenates to form global description. Divide image into equally spaced cells each 8x8 px. 4 cells form block, calculate Gradient orientation histogram for block (forming description for block). Description vector is normalised to form locally normalised descriptor. Small error value for stability. $\boldsymbol{v}_{norm} = \frac{\boldsymbol{v}}{\sqrt{||\boldsymbol{v}||_2^2 + \epsilon^2}}$. Block is then shifted along horizontally by one cell. Some overlap between blocks. HOG descriptor then formed by concatenating all normalised descriptors. HOG used to perform feature extraction from input image to a feature representation. Feat Extraction transforms image into low-dim vectors for easier comparison or matching. Classifier may be used to give output label to feature description.

Image Classification

ImageNet aims to collect 80,000 classes, with each approx. 500 – 1000 images. ILSVRC uses subset with only 1000 simpler classes. No overlap between classes since pairs of classes cannot be ancestors of each other. All classes organised into trees. 12 Subtrees below root: Mammal, bird, fish, reptile, Amphibian, Vehicle, Furniture, Musical Instrument, Geological formation, Tool, Flower / Fruit. Classes defined using concepts from natural language defined by wordnet (lexical db). Groups nouns into sets of cognitive synonyms each with distinct concept. Synnets form leaves. Classifiers classify images into groups. When labels are not available, we perform unsupervised training; a common technique is to perform clustering. **MNIST**: handwritten data recognition. 60,000 training samples, 10,000 for testing. Each sample is 28x28 image, classed 0-9. Preprocessing: find digit, normalise digit size to 28x28. Normalise location so mass centre is image centre. Slant correction may be performed, shifting each row so principal axis is vertical. Can extract features via either HOG or px intensity or learnt features such as CNNs. Afterward, perform classification with KNN for example. To define neighbours, use distance metric like Euclidean distance (hypotenuse). In our case we have 784 (28x28) dimensions. If different scales, can be better to normalise feature vector. Example is to normalise to Gaussian distribution N(0, 1), allowing dims to be treated fairly. Cosine Dist: $D(x,y) = \frac{x.y}{|x||y|} = \frac{x_1y_1 + \cdots + x_ny_n}{\sqrt{\sum_{i=1}^n x_i^2}\sqrt{\sum_{i=1}^n y_i^2}}$. Lp-norm: (Manhattan p = 1, Euclidean p = 2): $D_p(x,y) = \left(\sum_{i=1}^n |x_i - y_i|^p\right)^{\frac{1}{p}}$. Using k = 1 already works well although may still be inaccurate: 7 may be classified as 3, for example.

Hyperparameter finds k = 3 ideal, 2.4% error rate. KNN no training step, simple (but effective) and multi-class capable. Expensive to use (storage training data, searching has high computation cost). Computational cost to classify M test images with N training is O(MN). Don't mind slower training, as can be performed ahead of time. Test time must be fast. Pixel intensities as feature vector is fine for simple pre-processed images, but not ideal since scale and rotation variant. **Linear Classifier**: in 2D has the form $w_1x_1 + w_2x_2 + b = 0$. Rule of classifier assigns class c based on $c = 1\ if\ (w_1x_1 + w_2x_2 + b \geq 0)\ else - 1$. Once we know **w** and b, we can discard training data. Much faster in test time. More generally (w: weights, x: data, b: bias): **w.x** + b = 0. Same rule to assign class. **Support Vector Machine**: To determine max margin hyperplane, only need innermost points, the "support vectors". Would like support vectors to fulfil w.x + b = 1 or -1 to allow other points to be easily classified. Maximise margin between wx + b = 1 and wx + b = -1. We can derive distance as follows: $\boldsymbol{n} = \frac{\boldsymbol{w}}{||\boldsymbol{w}||}$. SVM then optimises $\min_{w,b}||w||^2 + C\sum_{i=1}^N \xi_i$ subject to $\forall i$: $\xi_i \geq 0; y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) \geq 1 - \xi_i$. $\xi_i = 0$ when correctly classified or between 0 and 1 when within the margin. Otherwise distance to margin. C is regularisation term (small C means easily fulfilled constraints, large C leads to tighter margin. Can formulate slack variable as $\xi_i = \max\left(0, 1 - y_i(\boldsymbol{w}.\boldsymbol{x_i} + b)\right)_{hinge\ loss}$. Note that both are convex functions, hence we know there is a minimum, and that local min is global min. Non-negative weighted sum of two convex funcs is also convex. $\forall x_1, x_2 \in R, \forall t \in [0,1]\ [f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)]$. A line joining $(x_1, f(x_1))$ and $(x_2, f(x_2))$ will lie above the function curve. There are two approaches to optimise this. Gradient Descent: Used in large scale optimisation. Hinge loss is not differentiable, but we can use a more generalised concept for a gradient, the subgradient: $\nabla_w h = -\boldsymbol{y_i x_i}\ if\ (y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) < 1)\ else\ 0$. This is similar for $\nabla_b h$ but we mostly focus on w for the derivation. For each iteration, we move along the negative of the gradient by step length η: $\omega^{(k+1)} = \omega^{(k)} - \eta(2\omega^{(k)} + C\sum_{i=1}^N \nabla_w h)$. Can also use Lagrangian Duality: often used in svm libraries. At test time, we can classify using: $c = 1\ if\ (f(x) = \boldsymbol{w}.x + b \geq 0)\ else - 1$. Works well if data is linearly separable. If not separable in the original feature space, then it may be easier to transform features. Consider dataset with 2 classes. The first being a circle of points centred around some point $\mathbf{x}_c$, and another class, a ring around the first. Using transformation $\Phi(x) = (x - x_c)^2$, the first class has lower value (closer to $\mathbf{x}_c$ which is squared). The second class is much higher, thus separating points. Therefore, if we transform feature vector using $\boldsymbol{\Phi}$, the classifier also much change. $f(x) = \left(\sum_i a_i y_i \boldsymbol{\Phi}(x_i)\right).\boldsymbol{\Phi}$ x + b. We also define the kernel (not same as filtering) as: $k(x_i, x_j) = \Phi(x_i).\Phi(x_j)$. The classifier only contains the kernel: $f(x) = \sum_i a_i y_i k(x_i, x) + b$. The common kernels use for SVM are: linear $k(x_i, x_j) = x_i.x_j$, Polynomial $k(x_i, x_j) = (x_i.x_j)^d\ or\ (1 - x_i.x_j)^d$, or Gaussian $k(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}}$. Note that only binary classification at the moment. For multi-class, we can take several approaches: 1 v rest: we train classifier for each digit, first classifies 1 and not 1, then 2 and not 2, … The classifier which produces the highest response determines the result. 1 v 1 classifies for each pairing of digits i.e. 1 v 2, 1 v 3, … At test time, each classifier votes for the digit and majority vote wins. We need (for K classes) $\frac{K(K-1)}{2}$ classifiers. **Neural Networks** are an algorithm to learn and model from the data for classification or regression problems. Inspired by biological NNs. The simplest Neural network is a perceptron, which has a single layer and uses Heaviside step function as activation function: w, b optimised to match ground truth. y = 1 if w.x + b > 0; 0 otherwise. Note that the difference (comp to SVM) is that the non-linearity is added with an activation function f(w.x + b) whereas

SVMs add non-linearity via feature transformation. Sigmoid (or logistic fn): $f(z) = \frac{1}{1+e^{-z}}$. A neural network is formed by connecting neurons, where outputs of neurons are the inputs of other neurons. The output a can be compared with the ground truth y in a loss function J (aim to minimise.) example Loss function is MSE: $J(W,b) = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{2}\left\|a_m - y_m\right\|^2$. To calculate the output of network a given x, use forward propagation. Parameters **W** and **b** can be found with gradient descent and back propagation used to calc gradient. We perform a calculation as follows: $z_i^{(2)} = W_{i,1}^{(1)}a_1^{(1)} + W_{i,2}^{(1)}a_2^{(1)} + W_{i,3}^{(1)}a_3^{(1)} + b_i^{(1)}; a_i^{(2)} = f\left(z_i^{(2)}\right);\quad z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}; a^{(l+1)} = f\left(z^{(l+1)}\right)$. Gradient descent: $W = W - \alpha\frac{\delta J}{\delta W}$, $b = b - \alpha\frac{\delta J}{\delta B}$. Back propagation is from chain rule. $z = g\left(f(x)\right), y = f(x), z = g(y); \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$. Using running example, we can derive $\frac{\delta J}{\delta a}$ *and* $\frac{\delta a}{\delta W^{(2)}}$, therefore we can use chain rule to get $\delta J/\delta W^{(2)}$. **Stochastic Gradient Descent**: Addresses comp expense of for/back propagation of all samples in one go. A back of B samples are selected for performing propagation. For each iteration, 1. Select random batch of B samples. Calculate Grads $\frac{\delta J_B}{\delta W}$ *and* $\frac{\delta J_B}{\delta b}$ for this batch. Update parameters using before eqn. ($J = J_B$). Note that MSE works for regression problems, where y is continuous. However, this is not optimal for classification problems, which can be binary or multiclass. We need single neuron in output for binary. Activation could be sigmoid, output in probability range. If we predict 0.9 for class 1, ground truth = 1. Distance metric between predicted prob and true prob can be used (cross entropy). Cross entropy: $H(p,q) = -\sum_i p_i \log(q_i)$. In general case, we have p = [y, 1 - y] and q = [f(z), 1 – f(z)] hence: $H(p,q) = -[y\log f(z) + (1-y)\log(1-f(z))]$. For K classes, there are K neurons in output layer. Activation function used would be softmax, to create prob dist: $f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K}e^{z_k}}$. The true probability is represented with 1-hot encoding: p = $[y_1, \ldots, y_i, \ldots, y_K] = [0, \ldots, 1, \ldots, 0]$ Performance: data augmentation can be used to make more training samples. Applies affine transformations to OG samples, translating, squeezing, scaling and shearing. Key limitation of MLP is high no of parameters required. Likely to scale poorly for large images. Caused by flattening of 2D image into single vector rather than performing on 2D images. **Convolutional Neural Networks**: CNNs assume input of 2D image and encode properties like local connectivity and weight sharing into the arch, reducing param no and making computation more efficient. When we look at an image, we first look at regions, process this info, then combine info from different regions to form global understanding. In CNNs, each neuron sees a receptive field in the layer before it. This is local connectivity. **Convolutional Layer**: basic building blocks. Consider input of XxYxC. If neuron required a 5x5xC cuboid of input, it would have 5x5xCx1 (1 for bias), in/output of neuron: $z = \sum_{i,j,k}W_{i,j,k}x_{i,j,k} + b\,; a = f(z)$. We can add more neurons to form Dx1x1 output (D=depth) - $a_n = f(\sum_{i,j,k}W_{i,j,k}x_{i,j,k} + b_n)$. Window can be moved across input, gives output of XxYxD. All windows use same weights. Convolutional kernel W has 4 dims: output depth, kernel width & height, input depth: $a_d = f(\sum_{i,j,k}W_{d,i,j,k}x_{i,j,k} + b_d)$. Output feature map has 3 dimensions: out depth, width, height. Params for layer: Padding (zeroes around edge so output is larger): output shape: $X_{out} = X_{in} + 2p - k + 1$. Stride (moving more than 1 px at a time): gives stride s x s: $X_{out} = \left\lfloor\frac{X_{in}+2p-k}{s}\right\rfloor + 1$. Dilation (increase region size without downsampling or more params): Using 3x3 kernel, look at a square of 5x5 in image and take every other pixel when convoluting. **Pooling Layer**: takes a block of pixels from image and takes single value from, i.e. max or average. Can choose pooling window size. **LeNet-5**: 7 layers. C=conv,S=pool,F=fully connected: C1: 28x28, depth6, 5x5Conv. S2: 14x14: depth6, max pooling. C3: 10x10, depth16, 5x5 conv. S4: 5x5, depth16. C5: 120 neurons. F6: 84 Neurons. Output: 10 classes. Same technique used to optimise loss fn. **Deep Neural Networks**: GPU improvements = better DNNs. Big issues with previous: exploding gradients – Too large for algorithm to converge; and vanishing gradient – too small for algorithm to make progress. One solution is gradient clipping: $g_i = \{v_{min}\ if\ (g_i < v_{\min}); v_{max}\ if\ (g_i > v_{max}); g_i\ otherwise$. Or; clip by l2 norm: $g = \{\frac{g}{\|g\|}v\ if\ (\|g\| > v); g\ otherwise$. Vanishing problem caused on either extreme of sigmoid. Sigmoid derivative: $f'(z) = \frac{e^{-z}}{(1-e^{-z})^2} = f(z)(1-f(z))$. Propagated der: $\frac{\delta J}{\delta z^{(l)}} = \left(\left(W^{(l)}\right)^T\frac{\delta J}{\delta z^{(l+1)}}\right)\circ f'\left(z^{(l)}\right)$. Can be addressed with different activation functions. ReLU: $f(z) = \{0\ if\ (z<0)\ else\ z$. Leaky ReLU: $f(z) = \{0.01z\ if\ (z<0)\ else\ z$. PReLU: $f(z) = \{az\ if\ (z<0)\ else\ z$ (a learnt in training). Exponential ReLU: $f(z) = \{\ a(e^z - 1)\ if\ (z<0)\ else\ z$. AlexNet similar to LeNet but deeper. Performs data augmentation by cropping random 224x224 regions from og 256x256 images. Horizontal reflections and pertubing RGB vals. VGG only uses 3x3 conv kernels. Conv of 2 3x3 kernels is eq to 5x5, 3 is 7x7 and so on. Deeper and reduces no of param. More non-linearity; activation functions after every small kernel.

Object detection

Want to find bounding box (x, y, w, h) of object within an image. Specifies (x, y) of top left and width, height. NN can perform classification. Move sliding window across image and perform classification and bounding box localisation. Window gives localisation, refine with CNN feats. CNN gives 2 branch output – fully connected to K classes, compared against ground truth class label (cross entropy). And another connected to four variables for bounding box, compared against ground truth bounding box. Added together and optimised. Expensive to apply CNN to each pixel of the image. Object detection therefore split into 2-stage detection: 1. Region proposal (initial guesses to propose possible regions) and detector: predict bounding box of said regions. One stage detection skips guessing by dividing image into grid cells and classifying on them cells. **Selective Search**: rarely used, traditional approach. **Faster R-CNN**: region proposal network used instead. Input fed into CNN giving feature map describing image. Feature map fed into RPN which gives regions to consider. Classifier / detector looks closely at proposed regions, can refine shape of bounding box. AlexNet/VGG-16 known as backbone networks (often pre-trained) with last conv layer being used as convolutional feature map. Both stages rely on feature map. Every pixel of feat map is high-dim feat vector, describing content of small region in input image. Can use 3x3 window to perform classification, giving 0 if isn't interesting, 1 if it is. Handles object of different sizes / aspect ratios by making k predictions. For example, 3 scales and 3 aspect ratios. These are anchors. The loss is defined as: $L(p,t) = \sum_{i=1}^{n_{anchor}}L_{cls}(p_i, p_i^*) + \lambda\sum_{i=1}^{n_{anchor}}1_{y=1}L_{loc}(t_i, t_i^*)$. Classification Loss L_{cls} calculated with cross entropy, comparing pred objectness p_i with ground truth p_i*. The localisation loss attempts to refine bounding box. Either directly predict values for bound box or the transformation parameter t from the anchor into ground truth. Note t=0 means pred bound box is anchor. Anchor: (x_a, y_a, w_a, h_a), Pred bounding box: (x, y, w, h). Pred transformation: (t_x, t_y, t_w, t_h). $t_x = \frac{x - x_a}{w_a},\ t_y = \frac{y - y_a}{h_a}, t_w = \log\left(\frac{w}{w_a}\right), t_h = \log\left(\frac{h}{h_a}\right)$. Easier to pred transformation rather than bound box direction, since pred box components may be arbitrary. Note that localisation loss compares vectors of continuous no's, therefore MSE can be used as loss fn. We can look closely at locations and object size in feat map. Features can be used in Region of Interest pooling – loc and size calced from feat map: converted into fixed size to be provided into classifier. Classifier predicts label class and refines bb estimate. $L(p,t) = L_{cls}(p,y) + \lambda\,.\,1_{y\geq0}L_{loc}(t, t^*)$. Classification loss now multi-class prob. RPN and detection network similar. Input to RPN is 3x3 window, input to detection is proposed region. Detection network classifies region into no. of classes, RPN is class-agnostic. Detection network doesn't need anchors. One stage Object detection uses methods such as YOLO and SSD. RPN in 2-stage uses binary classifier. This changes to multi-class which predicts object for each anchor. Faster R-CNN more accurate & slower, SSD vice-versa. Backbone networks can be used to improve performance. Meta arch combined with backbone can be used to observe trade-offs (speed v accuracy).

Image segmentation

Even more detailed understanding than bounding box – label for each pixel (generates segmentation mask). Instance segmentation gives each instance unique label. Thresholding: simplest method of segmentation: converts greyscale image to binary label map. Label defined as $f(x) = 1\ if\ (I(x) \geq threshold); 0\ otherwise$. Requires no training data, only parameter is threshold. Relies on intensity histogram split into 2 classes. **K-means**: Generally, more than 2 classes. Simple method for clustering by estimating params. Each cluster represented by centre; each px intensity associated with nearest centre. Optimal centres minimise intra-class variance: C_k = data point assoc. with k: $\min\sum_{k=1}^{K}\sum_{x\in C_k}(x - \mu_k)^2$. Reformulated with delta denoting membership. $\min\sum_{k=1}^{K}\sum_{x\in C_k}\delta_{x,k}(x - \mu_k)^2$. If we know delta, we can work out mu, and vice versa. Can be done iteratively. To determine no. of clusters k, can calculate class-variance for each k. can be performed on other features like colour-similarity. **Gaussian Mixture Model (GMM)**: GMMs allows for soft assignment by assuming each cluster is gaussian dist. Prob that $x_j\ in\ C_k \Rightarrow P(y_j = k \mid x_i, \pi_k, \mu_k, \sigma_k) = \pi_k.\frac{1}{\sqrt{2\pi\sigma_k}}e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$. Process for updating very similar to k-means. $\pi_k = \frac{\sum_j P(y_j=k)}{N}, \mu_k = \frac{\sum_j P(y_j=k).x_j}{\sum_j P(y_j=k)}, \sigma_k^2 = \frac{\sum_j P(y_j=k).(x_j - \mu_k)^2}{\sum_j P(y_j=k)}$. Class assigned by max prob. PowerPoint uses this. **CNNs**: segmentation map manually annotated. We assume we have this data. Ideas same as before. Each pixel in feat map of AlexNet encodes info about 16x16 region in input image. We can obtain classification-results from feat map. Convolutionalisation replaces fully-connected layers with conv layers enabling network to produce pixel-wise prob map (each class has heat map). Note depth of final network represents no of classes. Note that upsampling must be performed to obtain pixel-wise prediction since feat map is downsampled. Transposed convolution can achieve this. A convolution in 1D can be represented as: $\begin{bmatrix} w_1 & w_2 & w_3 & 0 \\ 0 & w_1 & w_2 & w_3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$. Transposed can then be $\begin{bmatrix} w_1 & 0 \\ w_2 & w_1 \\ w_3 & w_2 \\ 0 & w_3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$. Classification can be turned into segmentation by replacing fully connected layers with conv and transposed conv layers. Pixel-wise classification problem. Each pixel we can define classification loss (cross-entropy) and seg-loss = average. Possible for same network to segment and perform object detection.

Motion

Optic Flow: estimates movement of brightness/intensity in videos. Output is a flow field, same dims as video and describes px displacement. Video: 2D image sequence captured over time. Function of (x, y, t). for each (x, y) at time t, we want to estimate (x + u, y + v) at t + 1. 3 Assumptions: Brightness constancy: from 2 time frames from a video: assume two corresponding points will have same greyscale intensities. Small motion: objects wont move much per frame. Spatial coherence: neighbouring pixels should move similarly. **Optic Flow Constraint**: $I(x+u, y+v, t+1) = I(x, y, t)$. We can FO taylor expand on LHS, based on small motion: $I(x+u, y+v, t+1) \approx I(x,y,t) + \frac{\delta I}{\delta x}u + \frac{\delta I}{\delta y}v + \frac{\delta I}{\delta t} \Rightarrow \frac{\delta I}{\delta x}u + \frac{\delta I}{\delta y}v + \frac{\delta I}{\delta t} = 0$. Alternative notation: $I_x u + I_y v + I_t = 0$. **Lucas-Kanade**: there are 2 unknowns: u, v. using spatial coherence: $\left[I_x(p)\ \ I_y(p)\right]\begin{bmatrix} u \\ v \end{bmatrix} = -I_t(p)$. Assume u, v are same for small neighbourhood, then we have sys of linear eq: $\boldsymbol{A} = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \ldots & \ldots \\ I_x(p_N) & I_y(p_N) \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} u \\ v \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \ldots \\ I_t(p_N) \end{bmatrix}$. Unknowns estimated with least square: $x = argmin_x\left\|\boldsymbol{Ax} - \boldsymbol{b}\right\|^2$. Least square uses Moore-Penrose method to find solution. $\boldsymbol{x} = (\boldsymbol{A^T A})^{-1}\boldsymbol{A^T b}$. Note first seen in Harris detector. Related to cornerness. $cond(A^T A) = \frac{|\lambda_{max}|}{|\lambda_{min}|}$. Motion of corner easier to track than edge. **Aperture Problem**. Motion of line ambiguous through aperture. Must calc image gradients either with finite diff, or sobel/prewitt. As well as finite difference between time frames. Then apply least square as above. Assumption fails when motion is large. Use Multi-scale or multi-resolution network. Motion will look smaller in downsampled image. Flow can be estimated using scale pyramids on time-frames. Flow field obtained by summing all incr. flows. Multi-scale Lucas Kanade: for all scale l: 1. Upsample flow estimate from previous scale $u^{(l+1)} = 2u^{(l+1)}$, same for v. 2. Compute warped image. $J_{warped}^{l} = J^{l}(x + u^{(l+1)}, y + v^{(l+1)})$. 3. Compute partial der. At new scale: $I_t = J_{warped}^{l}(x,y) = I^{l}(x,y)$. 4. Estimate inc. flow: $\begin{bmatrix} u^{\delta} \\ v^{\delta} \end{bmatrix} = (A^T A)^{-1}A^T b$. 5. Update Flow: $u^{(l)} = u^{(l+1)} + u^{\delta}$, same for v. **Horn-Schunck:** Optimisation based method: global energy function for flow. $E(u,v) = \int\int_{(x,y)\in\Omega}\left(I_x u + I_y v + I_t\right)^2 + \alpha(|\nabla u|^2 + |\nabla v|^2)dxdy$. First part relates to optic flow, we want to minimise. Second half is smoothness term, minimising grad for u, v. Achieves a smooth flow field. Integrated over every pixel in omega. $u|v = \frac{I(x|y)(I_x\bar{u} + I_y\bar{v} + I_t)}{I_x^2 + I_y^2 + \alpha}$ 1. Compute gradients. 2. Initialise flow field u = 0, v = 0. 3. For each iteration k: 3.1 Calculate Average flow field $\bar{u}^{(k)}, \bar{v}^{(k)}$. 3.2: update flow field. 4. Terminate if change of val is smaller than threshold or max iterations. **Tracker**: aims to estimate motion from template image I to J (next frame). Assume template moves by u, v and assume will become same as image J. $\min_{u,v}E(u,v) = \sum_x\sum_u\left(I(x,y) - J(x+u, y+v)\right)_{(1)}^2$. Goal is to find how template moves from frame to frame. Can diff E wrt u,v and set to 0: $\frac{\delta E}{\delta u} = -2\times(1)\frac{\delta J}{\delta x} = 0, \frac{\delta E}{\delta v} = -2\times(1)\frac{\delta J}{\delta y} = 0$. Note that we apply the chain rule to obtain the $\frac{\delta J}{\delta x}$ term. (Same for 2nd eqn). We can use small motion assumption here too. We can approximate J' with I'. $= \frac{\delta E}{\delta u} = -2\sum_x\sum_y\left[-I_t - I_x y - I_y v\right]I_x = 0, \frac{\delta E}{\delta v} = -2\sum_x\sum_y\left[-I_t - I_x y - I_y v\right]I_y = 0$. We also have matrix form. $\begin{bmatrix} u \\ v \end{bmatrix} = \left(\sum_x\sum_y\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}\right)^{-1}\sum_x\sum_y\begin{bmatrix} I_x & I_t \\ I_y & I_t \end{bmatrix}$. Compared to Lucas Kanade optic flow, we sum over pixels within template image, whereas former sums over small neighbourhood. In general case where motion more complex, we can use parametric model for motion: $W(x,y;p) = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix}[x\quad y] + \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \min_{u,v}E(u,v) = \sum_x\sum_y\left(I(x,y) - J\left(W(x,y;p)\right)\right)^2$. Lucas-Kanade is classical method with assumptions that may not hold. **Correlation Filter**: aim to max correlation between template feat and image feats. $(f*h)[n] = \sum_{m=-\infty}^{\infty}f[m]h[n+m]$. Correlation can be more robust to illumination changes, compared to sum sq diff. In 2D images, we have a search window (in next time frame) and a box with the same size as the template. We can calculate correlation for each position (taking max). Use more discriminative features, such as those learnt from CNNs. Can use Siamese (two) networks to compare features. Each pixel in score map relation to position of search window. Can be trained with supervised learning. Paired images can be extracted from videos and annotated. This defines ground truth for score map. **Tracking by Detection**: Object tracking is performing object detection at each frame. In tracking, typically have initial bounding box, and only care about how single object moves, without considering objects in background. Since we are only interested in object in initial bounding box, we can learn specific features. Can perform online learning (using info from video as it's streamed), compared to offline (uses data from existing dataset). For object tracking, more interested in online learning. At each sliding window, we perform classification and localisation. +ve and -ve samples are extracted from first time frame. Note that hard samples are objects that look quite similar to the object we want to track. Can perform data augmentation to obtain more training data. More samples from more time frames = more to train with. Majority of network can be pre-trained with large datasets, only last few layers need online training. Fewer parameters in classification and localisation branches, trained with fewer samples.

Camera Model

Closely related to motion, mapping 3D world to 2D image. Denote 3D cords as **X** and 2D cords as **x**. Mapping is described with camera matrix P: $x = \boldsymbol{PX}$. **Pinhole Camera**: simplest case, can be generalised to others. Pinhole is very small hole. Note dark room on left side; image is flipped. Want to mode how **X** is mapped to **x** on image plane. Image plane is rotated by 180° and put as virtual image plane in from of pin hole. More convenient, image no longer flipped. Pinhole is camera origin, and distance from camera origin to image plane is focal length f. In one dimension, we can use similar triangles: $(X,Y,Z) \Rightarrow (f\frac{X}{Z}, f\frac{Y}{Z})$. Homogenous cords mean (x, y, 1) is same as (kx, ky, k) for any k. in homogenous cords, (X,Y,Z,1) ⇒ (fX, fY, Z). **Principle point offset**: in CCD array, image origin define as bottom/top left, hence image coordinate system origin may differ from Principle point p (centre of image plane). To account, (X,Y,Z,1) ⇒ $\left(f\frac{X}{Z} + p_x, f\frac{Y}{Z} + p_y, 1\right)$.can be matrified. **World Coordinate System**: if rotation, need to factor in 3D rotation matrix R: $\boldsymbol{X}_{cam} = \boldsymbol{R}(\boldsymbol{X} - \boldsymbol{C})$. Note inhomogenous coords; Using Homogenous, $\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix}_{(X_{cam})} = \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RC} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{(X)}$. **Camera Matrix:** map world coord **X** to image coord **x**. We have the following: $\boldsymbol{x} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{R} & -\boldsymbol{RC} \\ 0 & 1 \end{bmatrix}\boldsymbol{X}$. Camera matrix can be written more concisely as: $\boldsymbol{P} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}[\boldsymbol{R}\quad -\boldsymbol{RC}] = \boldsymbol{K}[\boldsymbol{R} \mid -\boldsymbol{RC}] = \boldsymbol{KR}[\boldsymbol{I} \mid -\boldsymbol{C}]$. Total 9 degrees of freedom. 3 intrinsic (inside camera), 3 for 3D rotation, 3 for 3D translation. Still representing image coordinates in same units as camera coordinates. We want conversion ratio k_x, k_y like pixels per millimetre: $\left(f\frac{X}{Z} + p_x, f\frac{Y}{Z} + p_y, 1\right) \Rightarrow \left(k_x\left(f\frac{X}{Z} + p_x\right), k_y\left(f\frac{Y}{Z} + p_y\right), 1\right)$. Converts camera to a_x, a_y instead of f and x_0, y_0 in place of p_x, p_y. Can be increased to 11 degrees of freedom when considering skew. **Calibration**: Assume we know 3D structure, we want to estimate camera matrix: Given set of points $\{\boldsymbol{X_i}, \boldsymbol{x_i}\}$ we want to estimate **P**, similar to many previous problems. Mapping: $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{p_1^T} \\ \boldsymbol{p_2^T} \\ \boldsymbol{p_3^T} \end{bmatrix}\boldsymbol{X}$. The second form allows us to have the following: $x = \frac{p_1^T X}{p_3^T X}$, y = $\frac{p_2^T X}{P_3^t}$. Rearranged we object multiple equations: Can develop an overdetermined matrx such as: $\begin{bmatrix} \boldsymbol{X_1^T} & 0 & -\boldsymbol{X_1^T}x_1 \\ 0 & \boldsymbol{X_1^T} & -\boldsymbol{X_1^T}y_1 \\ \ldots & \ldots & \ldots \\ \boldsymbol{X_n^T} & 0 & -\boldsymbol{X_n^T}x_n \\ 0 & \boldsymbol{X_n^T} & -\boldsymbol{X_1^T}x_1 \end{bmatrix}$. Due to this form, solution p is null space of A. This can be resolved adding a constraint P = $argmin_p\left\|\boldsymbol{AP}\right\|^2$. This can be solved using SVM for A. You can't use this information to track drones, for example. Once info gathered, to id correspondences between points.