Recognition
Allows us to see what is in an image - used in tracking, identification, reading characters, controlling robots etc…

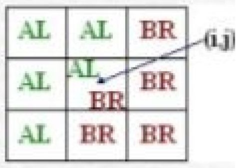
- **Template Matching**
  - **Applications**
    - Searching - locating known objects
    - Recognition - e.g. reading number plates (not a good approach)
    - Visual Inspection - e.g. golden template matching
      - Used in manufacturing: the perfect item is used as a template, compared to the items being produced, and a difference image is examined to detect flaws
    - Stereo Vision - determining how far away an object is using small templates
    - Tracking - assume an object changes slightly frame to frame, can use small templates to maintain the tracking
  - **Matching Algorithm**
    - Basic algorithm:
      - Input image and object
      - Evaluate a match criterion for each possible position of the object in the image
      - Search for local maxima above a threshold
    - **Problems**
      - To consider every possible position, do we evaluate at every rotation and scale also?
      - How do we identify a threshold? Too high and you miss genuine matches, too low you introduce false positives
      - What is the match criteria?
  - **Matching Criteria (metrics)**
    -
$$D_{SquareDifferences}(i,j) = \sum_{(m,n)} (f(i+m, j+n) - t(m,n))^2$$

$$D_{NormalisedSquareDifferences}(i,j) = \frac{\sum_{(m,n)}(f(i+m,j+n) - t(m,n))^2}{\sqrt{\sum_{(m,n)} f(i+m,j+n)^2 \sum_{(m,n)} t(m,n)^2}}$$

$$D_{CrossCorrelation}(i,j) = \sum_{(m,n)} f(i+m,j+n).t(m,n)$$

$$D_{NormalisedCrossCorrelation}(i,j) = \frac{\sum_{(m,n)} f(i+m,j+n).t(m,n)}{\sqrt{\sum_{(m,n)} f(i+m,j+n)^2 \sum_{(m,n)} t(m,n)^2}}$$

      - **Normalised square differences: 0 is optimal, 1 is the worst**
      - **Normalised cross correlation: 0 is the worst, 1 is optimal**
    - **Cross correlation ignores luminance**
    - What to do at boundaries?

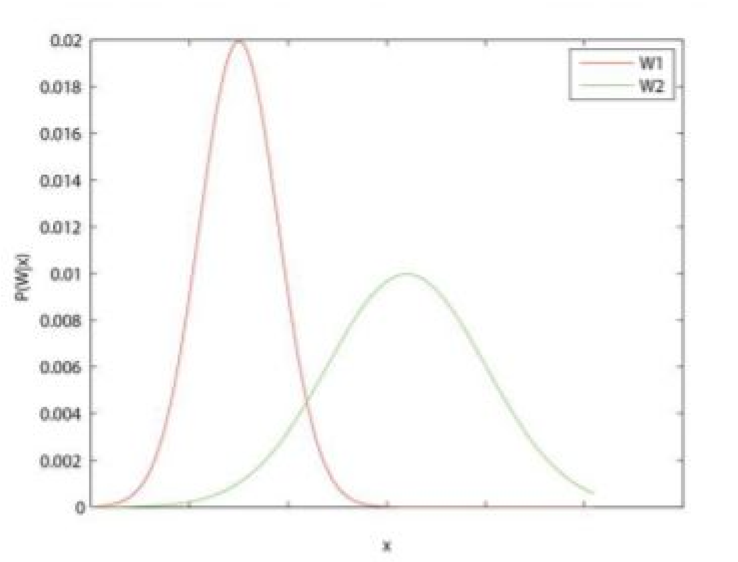
        - The degree-of-fit will be inconsistent with neighbouring values, so don’t compute generally
- **Control Strategies**
    - Localise close copies
        - Similar values in close proximity are probably matches to the same object so we select the greatest value in a local neighbourhood
        - The larger the neighbourhood the more dispersed the matches (typically 8)
    - Process using image hierarchy
        - **Use low res first, and limit the regions for high res matching based off this**
        - Not appropriate for very detailed templates
    - **Prioritise searches using the low res image matches as probability**
- **Chamfer Matching**
    - Template matching requires very close matches
        - Generally not possible
            - **Noise, Orientation**
    - **Need to be more flexible**
    - **A chamfer image is an encoding for each pixel of its distance to the nearest object (edge)**
    - Compute a chamfered image for a binary edge image
    -
```
For every point
    If (edge point)
        Set c(i, j) = 0
        Else Set c(i, j) = ∞
For j = min to max
    For i = min to max
        c(i, j) = min_{q∈AL} [distance( (i, j), q ) + f( q )]
For j = max to min
    For i = max to min
        c(i, j) = min_{q∈BR} [distance( (i, j), q ) + f( q )]
```


        - 2 pass algorithm
        - First pass considers all points AL (above and left) of the current pixel
        - Second pass considers all points BR (below and right) of the current pixel
    - **The template in Chamfer matching is a binary template in which only the object pixels are set and considered**
        - The metric is the sum of the overlapping values (0 is optimal)

- **Statistical Pattern Recognition (SPR)**
    - Derive features from known objects, and use these features to classify unknowns based on their similarity to these shapes

- **Probability**
    - P(A) = N(A)/n as n tends to infinity
    - **Independent events:** P(AB) = P(A)P(B)
    - **Dependent events:** P(AB) = P(A|B)P(B)
- Typical Probability Problem in this context
    - Having identified feature value x, given a class W_i, what's the probability that the object is of the class W_i? $P(x|W_i)$
    - This is in effect a **probability density function of the likelihood of a value occuring for that class**
        - We compute these for multiple classes
    - **However P(x|Wi) is a priori probability, it is reverse engineering (likelihood of a value being in this class, given this class), we care more about the probability of the class given some value [P(Wi|x)]**
    - **Bayes Theorem**
        - For two classes A and B, the a-posteriori probability is: P(B|A) = P(A|B)P(B)/P(A)
        - **Where W_i forms a partitioning of the event space:**

$$\frac{p(x \mid W_i)P(W_i)}{\Sigma_j\, p(x \mid W_j)P(W_j)}$$

    - 

        - Example probability density functions for two classes on a single feature x
        - The classes aren't linearly separable (as they overlap) - need another feature
    - **Example features:** Area, MinboundingRect, Convex hull(smallest convex region) , elongatedness (area divided by thickness squared), concavities and holes, perimeter length and circularity

- **Advantages of SPR: accurate**
- **Disadvantage: training, requires a decently sized training set**
- Learning strategies:
  - Supervised: probability density estimation
    - Training set includes class spec for every instance
  - Unsupervised: Cluster analysis
    - Look for similarities in feature space

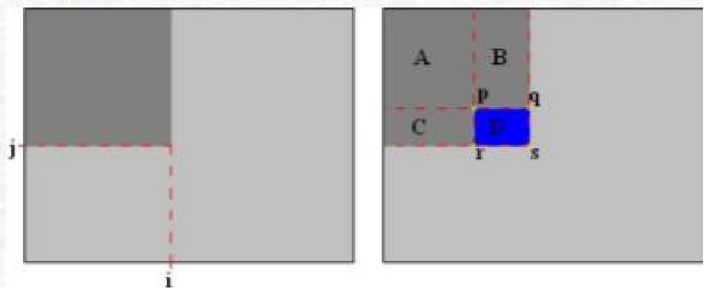
- **Support Vector Machines (SVM)**
  - Works for 2 class problem
  - Consider 2 linearly separable classes in n-dimensional feature space
  - **Finds the optimal hyperplane (max margin between classes)**
  - Given many n-dimensional training samples x_i with associated class identifiers $w_i = \{-1, 1\}$
    - Define separating hyperplanes $w.x + b = 0$, $w.x + b = 1$, $w.x + b = -1$
    - Constraint: $w_i(w.x_i + b) \geq 1$
    - If x^+ is a point on w.x+b=1 and x^- is the nearest point on w.x+b=-1
      - Then $\lambda w = x^+ - x^-$
      - Multiply by w…
      - $\lambda w.w = (1-b) - (-1-b) = 2$
      - Therefore $\lambda w.w = \lambda \|w\|^2 = 2$ as $\|w\| = \sqrt{w.w}$
    - $\|x^+ - x^-\| = \|\lambda w\| = \left\| {}^{2.w}/_{\|w\|^2} \right\| = \|{}^{2.w}/_{w.w}\| = \|{}^{2}/_{w}\| = {}^{2}/_{\|w\|}$
    - Hence we must minimise ||w|| to maximise the separation, subject to $w_i(w.x_i + b) \geq 1$
  - **Lagrangian Optimisation**
    - This is a minimisation problem subject to constraints which is amenable to solution by Lagrangian optimisation
    - $L(w, b, \alpha) = {}^{\|w\|}/_{2} - \sum_{i=1}^{n} \alpha_i [\omega_i.(w.x_i + b) - 1$
    - To find the minimums we set the partial differentials to 0
      - Classification based on:
      - Entering an unknown feature vector
      - Into the original equation $f(x_i) = w.x_i + b$
      - And classifying based on whether the value is positive or negative
  - What if classes aren't linearly separable?
    - Apply the kernel trick
      - Dot products are replaced by some non-linear kernel function
    - Soft margin or multiclass SVM

- **Cascade of Haar classifiers (Haar)**
    - Robust object detection using a cascade of classifiers
    - **Efficient calculation of features**
        - Only uses simple features
    - Learns based on a number of positive and negative samples
    - Selects a large number of features during training and creates classifiers with them to accept/reject
        - Classifiers are organised into a cascade(sequentially) where if the sub image is rejected by any of the classifiers, it is rejected
        - **Efficient: most sub-images stopped by first or second classifier**
    - System trains using particular scales, but classifiers are designed so that they are easily resized
    - **Features are determined as the difference of the sums of a number of rectangular regions**
        - Place the mask in a specific location
        - At a specific scale
        - Subtract the normalized sum of the white pixels from the normalized sum of the black pixels
        - Why does this work?
            - **Integral image**

              Every point $ii(i,j) = \Sigma_{i'=0..i} \Sigma_{j'=0..j} \text{image}(i',j')$

              Sum of points in rectangle D:

              $sum(D) = ii(p) + ii(s) - ii(q) - ii(r)$

            - **Features can be computed at any scale for the same cost**
    - **Training**
        - Needs a large number of positive and negative samples
        - Calculates hundreds of thousands of possible features
        - Uses training to identify which features are most important and at which points in the cascade
    - **Weak Classifiers**
        - Created by combining a specific feature with a threshold, making an accept or reject classification through comparison
        - $p_j feature_j(x) < p_j \vartheta_j$

          Tune threshold (ϑ_j)
    - **Strong Classifiers**
        - Combine a number of weak classifiers
        - Algorithm for this combination is AdaBoost

- **Principal Components Analysis (PCA)**
  - Statistical Technique for: Data Analysis, Compression, and Recognition
  - Analyses data covariance
  - Identifies principal directions
  - **Example**
    - Given 2D data, and N samples/vectors
    - Find the mean sample, the direction of maximum covariance, and the direction orthogonal to this
  - **Background - Eigenvalues and Eigenvectors**
    - Consider a square matrix A
    - The eigenvalues of A are the roots of the characteristic equation:
      - $$determinant(A - \lambda I) = |A - \lambda I| = 0$$
    - **For each λ there will be an eigenvector x such that Ax = λx**
    - An n x n matrix will have n eigenvalues
    - Consider n = 2 ; There will be 2 eigenvalues: λ_1, λ_2
      - With corresponding eigenvectors: x_1, x_2
      - Where $Ax_1 = \lambda_1 x_1$ and $Ax_2 = \lambda x_2$
      - Combining we get $A[x_1 \quad x_2] = [x_1 \quad x_2]\begin{bmatrix}\lambda_1 & 0 \\ 0 & \lambda_2\end{bmatrix}$

        and $\mathrm{A\Phi} = \Phi\Lambda$

        where $\Phi = [x_1 \quad x_2]$ and $\Lambda = \begin{bmatrix}\lambda_1 & 0 \\ 0 & \lambda_2\end{bmatrix}$

        If we normalise the eigenvectors then $\Phi\Phi^T = \Phi^T\Phi = I$

        so $\Phi^T \mathrm{A} \Phi = \Phi^T \Phi \Lambda = \Lambda$

        and $\mathrm{A} = \mathrm{A}\Phi\Phi^T = \Phi\Lambda\Phi^T$
    - **In PCA we have N samples/vectors in some n-dimensional space**
    - **Combine into data matrix D**
      - $$\mathrm{D} = \begin{bmatrix} x_1^1 & x_2^1 & .. & x_n^1 \\ x_1^2 & x_2^2 & .. & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & .. & x_n^N \end{bmatrix}$$
      - Each row is a sample
    - **Determine the mean of the samples**
      - $$\mu = \left[\sum_{i=1}^N x_1^i/_N \quad \sum_{i=1}^N x_2^i/_N \quad .. \quad \sum_{i=1}^N x_n^i/_N\right]$$
    - **Compute mean-centred data U**

- $$U = D - \begin{bmatrix} \mu \\ \mu \\ \vdots \\ \mu \end{bmatrix} = \begin{bmatrix} x_1^1 - \sum_{i=1}^{N} x_1^i / N & x_2^1 - \sum_{i=1}^{N} x_2^i / N & .. & x_n^1 - \sum_{i=1}^{N} x_n^i / N \\ x_1^2 - \sum_{i=1}^{N} x_1^i / N & x_2^2 - \sum_{i=1}^{N} x_2^i / N & .. & x_n^2 - \sum_{i=1}^{N} x_n^i / N \\ \vdots & \vdots & & \vdots \\ x_1^N - \sum_{i=1}^{N} x_1^i / N & x_2^N - \sum_{i=1}^{N} x_2^i / N & .. & x_n^N - \sum_{i=1}^{N} x_n^i / N \end{bmatrix}$$
- Using mean-centred data U, compute the covariance matrix Σ
    - $$\Sigma = U^T U / (N-1)$$
- We can then determine $\Phi^T \Sigma \Phi = \Lambda$
    - **Where the eigenvectors are in an orthogonal matrix - Φ**
        - We can consider Φ as a linear transformation
    - **And the eigenvalues are in an ordered diagonal matrix - Λ**
        - The amplitudes of the eigenvalues in Λ are proportional to the percentage of overall variance accounted for by the associated eigenvector
- **Facial recognition with PCA**
    - **Common example application of PCA**
    - Each face image is treated as a vector with n=rows*columns dimensions
    - **Approach**
        - Create a matrix D
            - Where each face image is a row
            - D is an N x n matrix
        - Create mean centred data U
            - Determine the mean of the rows (face images)
            - Subtract from all the rows
        - Compute the covariance matrix
        - Solve for eigenvalues and eigenvectors
        - Normalise the eigenvectors
        - Select m Eigenvectors (with the largest m Eigenvalues)
            - Usually 20-50 for facial recognition
        - For an unknown image find its location in PCA space
            - Look for closest match
- **Performance**
    - **Two aspects**
        - **Computation Time**
            - How long did it take?
        - **Success and Failure Rates**
            - What is the correct answer? We need **Ground Truth**
                - **Ground truths have to be manually computed - very difficult to get agreement**
            - How do we assess success? We need **Metrics**
                - **Compute TP TN FP FN**
                - **Then Compute:**

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{Total\ Samples}$$

$$Specifcity = \frac{TN}{FP + TN}$$

$$F_\beta = (1 + \beta^2).\frac{Precision.Recall}{(\beta^2.Precision) + Recall}$$

-

- **Can use these metrics to guide system tuning**