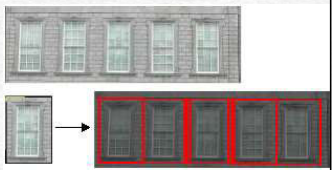
Recognition

- Template Matching
  - Chamfer Matching
- Statistical Pattern Recognition (SPR)
- Support Vector Machines (SVM)
- Cascade of Haar classifiers (Haar)
- Principal Components Analysis (PCA)

1

Template Matching - Topics

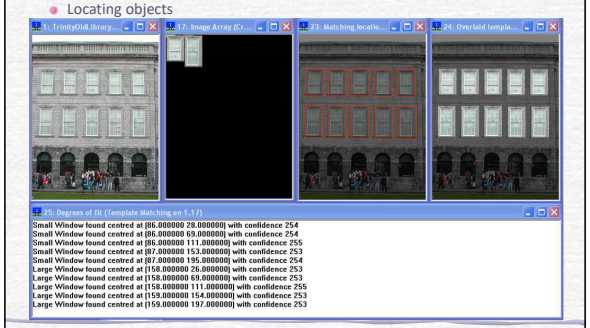
- Applications
- Matching criteria
- Control strategies
- Chamfer Matching



2

Template – Applications for template matching

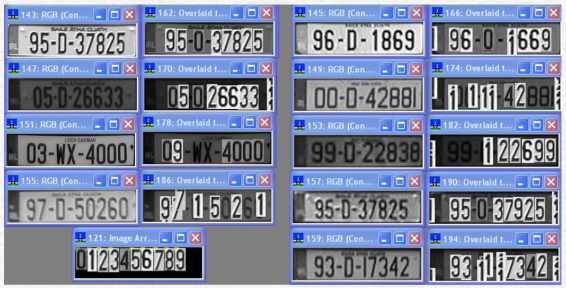
- Searching
  - Locating objects



3

Template – Applications for template matching

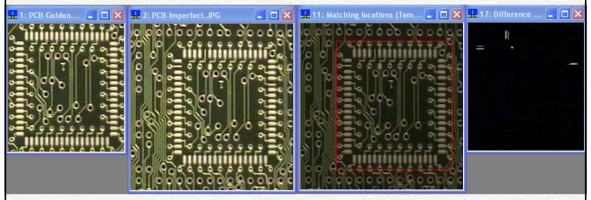
- Recognition



4

Template – Applications for template matching

- Visual inspection
  - Golden template matching



- Matching
  - Stereo Vision
  - Tracking

5

Template – Matching Algorithm

- Basic Algorithm
  - Inputs – Image & Object
  - For every possible position of the object in the image
    - Evaluate a match criterion
  - Search for local maxima of the match criterion above some threshold
- Problems
  - 'Every possible position'?
  - 'Match criterion'
  - 'Local maxima above some threshold'

6

Template – Matching criteria

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

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

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

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

7

Template – Matching criteria example

- Simple example
  - Matching criteria
  - Boundaries…

0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	1	1	1	1	0
0	0	0	1	1	0	1	0
0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0
0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0

Image

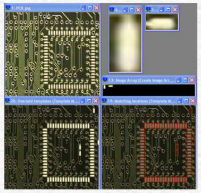
1	1	1	1
1	0	0	1
1	0	0	1
1	0	0	1
1	1	1	1

Template

11	13	14	11	14
10	14	16	8	14
9	12	12	1	10
10	14	15	7	15

Matching Space

- Real example
  - Uses normalised cross correlation.



8

Template Matching in OpenCV

```
Mat matching_space;
matching_space.create(
   search_image.cols–template_image.cols+1,
   search_image.rows–template_image.rows+1, CV_32FC1 );
matchTemplate( search_image, template_image,
                matching_space, CV_TM_CCORR_NORMED );
// Other measures: CV_TM_CCORR, CV_TM_SQDIFF,
//                 CV_TM_SQDIFF_NORMED
```


9

Template – Finding Local Maxima in OpenCV

- Local maxima – Dilate + Look for unchanged values + Threshold
- Local minima – Erode + Look for unchanged values + Threshold

```
Mat dilated, thresholded_matching_space, local_maxima,
                                thresholded_8bit;
dilate( matching_space, dilated, Mat());
compare( matching_space, dilated, local_maxima, CMP_EQ );
threshold( matching_space, thresholded_matching_space,
                    threshold, 255, THRESH_BINARY );
thresholded_matching_space.convertTo( thresholded_8bit,
                                CV_8U );
bitwise_and( local_maxima,thresholded_8bit,local_maxima );
```


10

Template – Control Strategies for Matching

- Goal: Localise close copies
  - Size, orientation
  - Geometric distortion
- Use an image hierarchy
  - Low resolution first
  - Limit higher resolution search
- Search higher probability locations first
  - Known / learnt likelihood
  - From lower resolution

11

Template – Chamfer matching

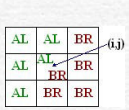
- Template matching requires very close matches
- Objects often appear very slightly different
  - Orientation
  - Noise
  - Sampling
- We want a more flexible approach

12

Template – Chamfering

- Compute 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
    $c(i, j) = \min_{q \in AL}$ [distance((i, j), q) + f(q)]
For j = max to min
  For i = max to min
    $c(i, j) = \min_{q \in BR}$ [distance((i, j), q) + f(q)]




```
Canny( gray_image, edge_image, 100, 200, 3);
threshold(edge_image,edge_image,127,255,THRESH_BINARY_INV);
distanceTransform(edge_image,chamfer_image,CV_DIST_L2,3);
```

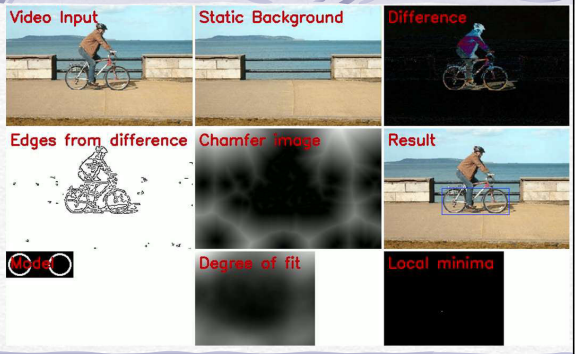
Object pixels (zeros)

After the first stage of processing

Chamfer Image

13

Template – Chamfer matching



14

Template – Chamfering for matching

- Compare with boundary template
  - Sum of chamfer values for boundary points
  - Low values best

Chamfer Image

Template

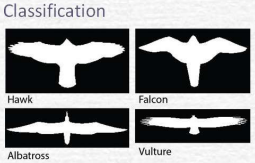
Matching Space

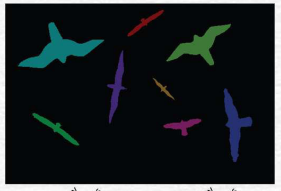
Chamfer matching is a simple routine to write (provided in the text);

15

SPR – Statistical Pattern Recognition

- Probability Review
- Features
- Classification





	Rectangularity	Elongatedness
Albatross	0.28	3.63
Vulture	0.45	5.56
Hawk	0.39	2.39
Falcon	0.33	2.11

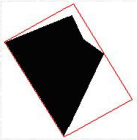
Rectangularity	Elongatedness	
0.33	2.11	Falcon
0.49	5.29	Vulture
0.30	3.67	Albatross
0.48	5.45	Vulture

Rectangularity	Elongatedness	
0.33	2.10	Falcon
0.31	3.61	Albatross
0.41	2.38	Hawk
0.37	2.72	Hawk

16

SPR – Features: Minimum Bounding Rectangle

- Turn rectangle through discrete steps
- Only one quadrant
- Metrics:
  - Length to Width ratio
    - Length / Width
  - Rectangularity
    - Area / (Length * Width)
  - Convex Hull to Minimum Bounding Rectangle area ratio
    - Area inside convex hull / (Length * Width)



```
RotatedRect min_bounding_rectangle =
                    minAreaRect(contours[contour_number]);
```


17

SPR – Features: Elongatedness

- CANNOT BE the ratio of the length and width of the minimum bounding rectangle
- Ratio of region area divided by the square of it's thickness
  - Erosion

$$elongatedness = \frac{area}{(2d)^2}$$

	Rectangularity	Elongatedness
Circle	0.79	1.00
Square	1.00	1.00
Rectangle	1.00	1.85

Rectangularity	Elongatedness	
1.00	1.82	Rectangle
0.99	1.81	Rectangle
0.39	1.82	Rectangle

Rectangularity	Elongatedness	
0.78	1.00	Circle
0.40	1.00	Circle
1.00	1.00	Square

18

SPR – Probability Review – Basics

- $P(A) = lim_{n \to \infty}\ N(A) / n$
- Probability of two events *A* and *B*:
  - Independent:
    $P(AB) = P(A)P(B)$
  - Dependent:
    $P(AB) = P(A|B)P(B)$
    - Conditional Probability $P(A|B)$
- Typical problem:
  - Given some evidence *x* from an unknown object.
  - What class W_i is the object?
  - Training
    - *A-priori* probability – $p(x \mid W_i)$
    - Relative probability – $p(W_i)$
  - *A-posteriori* probability – $p(W_i \mid x)$

19

SPR – Probability Review – Bayes Theorem

- For two classes *A* and *B* the *a-posteriori* probability is:

$$P(B|A) = P(A|B)P(B) / P(A)$$

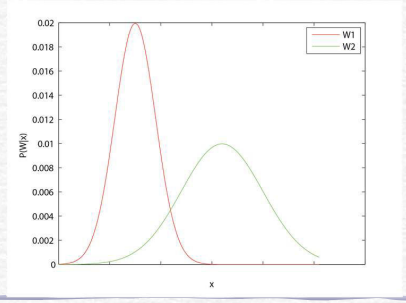
- Where W_i forms a partitioning of the event space:

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

20

SPR – Probability Density Functions

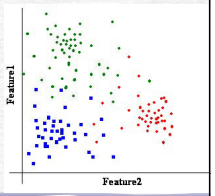
- Given a class (W_i) and an event (x)
  - Determine the probability of any particular value occurring...



21

SPR – Classification

- Object recognition
  - Classes (w_1, w_2, ... w_R)
- Classifier
  - Input Pattern / features (x_1, x_2, ... x_n)
- Feature space
  - Choosing the features (Example)
  - Clusters in feature space
- Separability
  - Hyper-surfaces
  - Linear separability
  - Inseperable classes
- Probabilistic Classifier



22

SPR – Probabilistic classifier

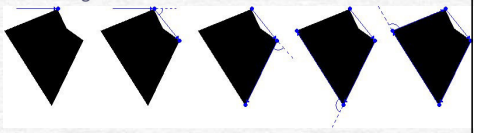
- Optimal classifier
- Based on Probability Density Functions
  - Remove normalisation from Bayes rule... $\sum_j p(x|W_j)P(W_j)$
  - Resultant Discrimination function: $g_r(x) = p(x|W_r)P(W_r)$
  - $g_r(x) \geq g_s(x)$
  - Unknown class: $g_r(x) >$ threshold
  - Mean Loss Function: $J(q) = \int_x \sum_{s=1..R} \lambda[d(x,q)|W_s]p(x|W_s)P(W_s)\,dx$

    where $\lambda[W_r|W_s] = \begin{cases} 0 & if\ r = s \\ 1 & otherwise \end{cases}$
- Advantages / Disadvantages
  - Accuracy
  - Extensive training

23

SPR – Features: Convex hull

- Smallest convex region



- Algorithm
  1. Start with
     - Any point on convex hull
     - Previous vector direction
  2. Search all other boundary points
     - Find point with least angle to previous vector
  3. Switch to new point and vector
  4. Go to 2 unless new point = start point.

24

SPR – Features: Convex hull

```
vector<vector<Point>> hulls(contours.size());
for (int contour=0; (contour<contours.size()); contour++)
{
  convexHull(contours[contour], hulls[contour]);
}
```


25

SPR – Features: Concavities and Holes

- Identify concavities from the convex hull
- Identify holes in the binary shape

	Shape	Height / Width	Hull / Box	# holes	Hole Areas	# concavities	Concavity Details
0	2.46	0.85	1	4.1		0	
1	4.71	0.58	0			2	0.4@243 0.03@261
2	2.83	0.85	0			5	3.6@213 2.5@45
3	2.83	0.82	0			3	5.7@185 0.2@354
4	2.38	0.62	1	0.6		3	0.9@101 0.1@207
5	2.54	0.85	0			4	3.1@140 2.4@302
6	2.13	0.79	1	1.5		3	3.1@317 0.03@94
7	2.36	0.57	0			9	4.3@205 0.1@294
8	2.83	0.80	2	1.8	1.6	2	0.3@205 0.2@355
9	2.62	0.81	1	1.9		4	3.0@143 0.1@240

	Shape	Height / Width	Hull / Box	# holes	Hole Areas	# concavities	Concavity Details
9	2.62	0.81	1	1.8		3	2.9@137 0.03@66
5	2.54	0.85	0			4	3.0@140 2.4@302
0	2.54	0.87	1	4.5		3	0.4@171 0.1@116
3	2.62	0.81	0			3	5.7@184 0.4@343
7	2.36	0.59	0			4	4.2@211 0.4@0
8	2.62	0.81	2	1.6	1.5	3	0.3@354 0.3@203
2	2.43	0.84	0			3	3.9@201 2.8@40
5	2.62	0.83	0			4	3.0@143 2.4@300

26

SPR – Features: Concavities and Holes

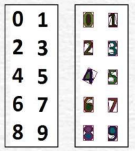
```
vector<vector<int>> hull_indices(contours.size());
vector<vector<Vec4i>> convexity_defects(contours.size());
for (int contour=0; (contour<contours.size()); contour++)
{
  convexHull( contours[contour], hull_indices[contour] );
  convexityDefects( contours[contour], hull_indices[contour],
                                        convexity_defects[contour]);
}
```


27

SPR – Features: Perimeter Length and Circularity

- Perimeter length
  - Approximately the number of boundary elements

    `contours[contour].size()`
  - Should really take into account direction
- Circularity = Perimeter length / (4*π*Area)



	Shape	Perimeter	Area	Bounding Box	Convex Hull	Hu Moments			Hole Areas	
0	80	406	651	555	0.17	0.00	0.00	175		
1	101	274	522	429	0.47	0.13	0.01			
2	111	347	570	528	0.42	0.07	0.00			
3	121	351	522	530	0.40	0.06	0.00			
4	85	347	592	442	0.20	0.01	0.00	60		
5	122	351	570	504	0.43	0.06	0.00			
6	102	392	620	534	0.23	0.01	0.00	76		
7	94	267	551	394	0.46	0.09	0.04			
8	65	438	651	571	0.20	0.01	0.00	45	62	
9	101	389	589	524	0.23	0.01	0.00	74		

28

SPR – Classifier learning

- Training set
  - Must be representative
  - Must be inductive
    - Can use Kernel Density Estimation
- Training set size
  - Training set provides the unknown statistical information
  - Size will typically have to be increased several times
- Learning strategies
  - Supervised:
    - Probability density estimation – estimating $p(x|w_r)$ & $P(w_r)$
    - Training set includes class specification for every instance
  - Unsupervised:
    - Cluster Analysis
    - Look for similarities in feature space

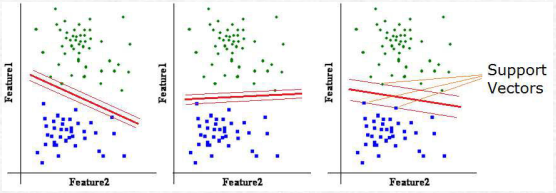
SPR: Real example

	Shape	Height / Width	Hull / Box	# holes	Hole Areas	# concavities	Concavity Details
0	2.46	0.85	1	4.1		0	
1	4.71	0.58	0			2	0.4@243 0.03@261
2	2.83	0.85	0			5	3.6@213 2.5@45
3	2.83	0.82	0			3	5.7@185 0.2@354
4	2.38	0.62	1	0.6		3	0.9@101 0.1@207
5	2.54	0.85	0			4	3.1@140 2.4@302
6	2.13	0.79	1	1.5		3	3.1@317 0.03@94
7	2.36	0.57	0			9	4.3@205 0.1@294
8	2.83	0.80	2	1.8 1.6		2	0.3@205 0.2@355
9	2.62	0.81	1	1.9		4	3.0@143 0.1@240

	Shape	Height / Width	Hull / Box	# holes	Hole Areas	# concavities	Concavity Details
9	2.62	0.81	1	1.8		3	2.9@137 0.03@66
5	2.54	0.85	0			4	3.0@140 2.4@302
0	2.54	0.87	1	4.5		3	0.4@171 0.1@116
3	2.62	0.81	0			3	5.7@184 0.4@343
7	2.36	0.59	0			4	4.2@211 0.4@0
8	2.62	0.81	2	1.6 1.5		3	0.3@354 0.3@203
2	2.43	0.84	0			3	3.9@201 2.8@40
5	2.62	0.83	0			4	3.0@143 2.4@300

Support Vector Machines

- Popular technique for two-class problem
- Consider two linearly separable classes in *n*-dimensional feature space
- Many hyper-surfaces usually exist
- Optimal classification
  - Maximise the margin between the classes.



Support Vector Machines – Example

From *Vision-Based Traffic Data Collection Sensor for Automotive Applications* by Llorca DF, Sánchez S, Ocaña M, Sotelo MA - (2010) http://openi.nlm.nih.gov/detailedresult.php?img=3270873_sensors-10-00860f12&req=4

Support Vector Machines – Formulation

- Given many *n*-dimensional training samples x_i with associated class identifiers $\omega_i = \{-1, 1\}$
- Define the separating hyper-planes $w.x + b = 0$
  $w.x + b = 1$ $w.x + b = -1$
- Constraint: $\omega_i(w.x_i + b) \geq 1$
- 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 $\lambda w.w = x^+.w - x^-.w = (1 - b) - (-1 - b) = 2$
  - Therefore $\lambda w.w = \lambda\|w\|^2 = 2$ as $\|w\| = \sqrt{w.w}$ and...

$$\|x^+ - x^-\| = \|\lambda w\| = \left\|{}^{2.w}/_{\|w\|^2}\right\| = \|{}^{2.w}/_{w.w}\| = \|{}^{2}/_{w}\| = {}^{2}/_{\|w\|}$$

- Hence we must minimise $\|w\|$ to maximise the separation, subject to $\omega_i(w.x_i + b) \geq 1$

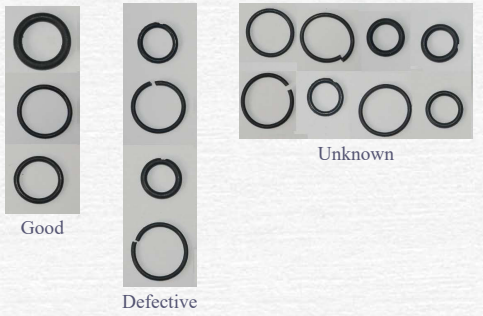
Support Vector Machines – Lagrangian Optimisation

- 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 is simply based on
  - entering an unknown feature vector x_i
  - into the original equation $f(x_i) = w.x_i + b$
  - and classifying based on whether the value is positive or negative (preferably >= 1).

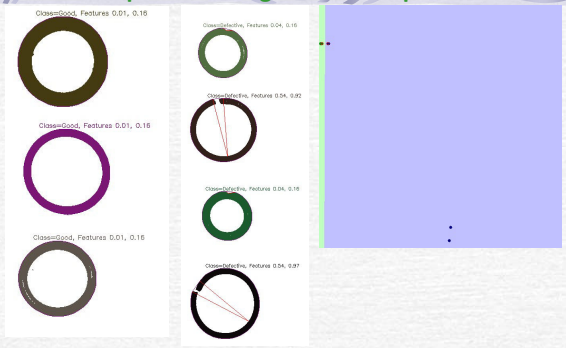
Support Vector Machines – Example



SVM Example – Training

```
Ptr<SVM> svm = ml::SVM::create();
svm->setType(SVM::C_SVC);
svm->setKernel(SVM::LINEAR);
Mat labelsMat(number_of_samples, 1, CV_32SC1, labels);
Mat trainingDataMat(number_of_samples, 2, CV_32FC1,
                                                training_data);
Ptr<ml::TrainData> tData = ml::TrainData::create(trainingDataMat,
                        ml::SampleTypes::ROW_SAMPLE, labelsMat);
svm->train(tData);
```

SVM Example – Training & Feature Space



37

SVM – Support vectors

```
Mat support_vectors = svm->getSupportVectors();
for (int support_vector_index = 0; support_vector_index <
support_vectors.rows; ++support_vector_index)
{
  const float* v = support_vectors.ptr<float>(support_vector_index);
  circle(feature_space, Point((int) v[0],(int) v[1]), 3,Scalar(0,0,255));
}
```


38

SVM Example – Prediction

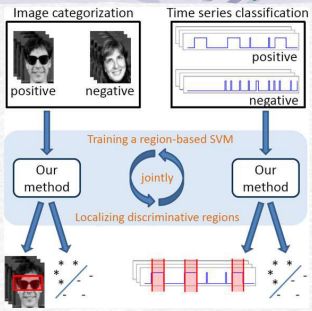
```
// Try to predict the class
Mat sampleMat = (Mat_<float>(1,2) << feature[0], feature[1]);
float prediction = svm->predict(sampleMat);
class_id = (prediction > 0.0) ? 1 : (prediction < 0.0) ? 2 : 0;
```


39

SVM Example – Prediction

40

Support Vector Machines – Example

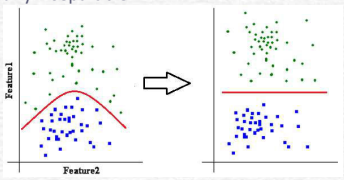


From *Weakly supervised discriminative localization and classification: a joint learning process.*
by Nguyen, M.H., Torresani, L., De la Torre, F., & Rother, C. (2009) Proceedings of International Conference on Computer Vision.
http://www.robots.ox.ac.uk/~minhhoai/projects/segsvm.html

41

Support Vector Machines – Nonlinear classification

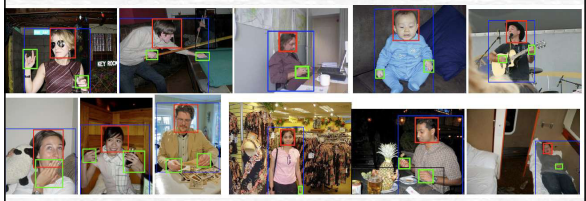
- What if classes are linearly inseparable



- Apply the kernel trick
  - Dot products replaced by some non-linear kernel function
    - E.g. $k(x_i, x_j) = (x_i . x_j)^d$
- Other technique
  - Soft Margin
  - Multiclass SVM

42

Support Vector Machines – Example



From *Taxonomic Multi-class Prediction and Person Layout using Efficient Structured Ranking*
by A. Mittal, M. B. Blaschko, A. Zisserman, P. H. S. Torr , in *European Conference on Computer Vision, 2012*
http://www.robots.ox.ac.uk/~vgg/software/struct_rank/

43

Haar – Robust Object Detection using a cascade of classifiers.

- Features
  - Efficient calculation
- Training
- Weak & Strong Classifiers
  - Adaboost
- Recognition

Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

44

Haar – In OpenCV.

```
CascadeClassifier cascade;
if( !cascade.load(
            "haarcascades/haarcascade_frontalface_alt.xml" )
{
 vector<Rect> faces;
 equalizeHist( gray_image, gray_image );
 cascade.detectMultiScale( gray_image, faces, 1.1, 2,
                    CV_HAAR_SCALE_IMAGE, Size(30, 30) );
}
```


45

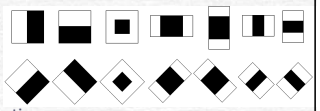
Haar – Overview

- Training using a number of positive and negative samples
- Uses simple (Haar like) features
  - Efficient calculation…
- Selects a large number of these features during training to create strong classifiers
- Links a number of strong classifiers into a cascade for recognition
  - Efficiency…
- Can work at different scales

46

Haar – Features

- Features determined as the difference of the sums of a number of rectangular regions

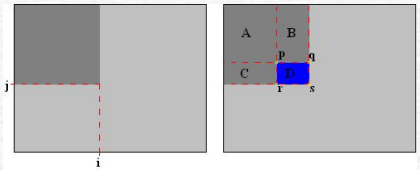


- Place the mask in a specific location
- and at a specific scale
- Then subtract the sum of the 'white pixels'
- from the sum of the 'black pixels'
- Why does this work?

47

Haar – Efficient Calculation – Integral Image

- Integral Image:
  - Every point ii(i,j) = $\Sigma_{i'=0..i} \Sigma_{j'=0..j}$ image(i',j')
  - Sum of points in rectangle D:
    - sum(D) = ii(p) + ii(s) – ii(q) – ii(r)



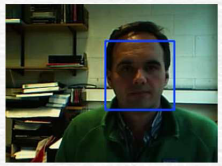
- Features can be computed at any scale for the same cost

An **integral** function is provided which computes the integral image both normally and at 45°

48

Haar – Training

- Number of possible features.
  - the variety of feature types
  - allowed variations in size and position
- Training must
  - Identify the best features to use at each stage
  - To do this positive and negative samples are needed…



49

Haar – Weak & Strong Classifiers

- Weak Classifier
  - $p_j feature_j(x) < p_j \vartheta_j$
  - Tune threshold (ϑ_j)
- Strong Classifiers
  - Combine a number of weak classifiers…
  - E.g. 100% true positives with only 40% false positives using only two face features…

50

Haar – Strong Classifiers – Adaboost

Given n example images $x_1..x_n$ together with classifications $y_1..y_n$
where $y_i = 0, 1$ for negative and positive examples respectively.

Initialise weights $w_{1,i} = 1 / (2*(m*(1- y_i)+l*y_i))$
where m and l are the number of negative and positive examples respectively.

For $t=1,...,T$

1. Normalize the weights (i.e. for all i): $w_{t,i} = w_{t,i} / (\Sigma_{j=1..n} w_{t,j})$
2. For each feature, j, train a weak classifier $h_j(x)$ and evaluate the error taking into account the weights: $\varepsilon_j = \Sigma_i w_{t,i} \mid h_j(x_i) - y_i \mid$
3. Select the classifier, $h_j(x)$, with the lowest ε_j, save as $c_t(x)$ with error E_t
4. Update the weights (i.e. for all i): $w_{t+1,i} = w_{t,i} \beta_t^{(1-e_i)}$
   where $e_i = \mid c_t(x_i) - y_i \mid$ and $\beta_t = E_t / (1–E_t)$

The final strong classifier is: $h(x) = 1$ if $\Sigma_{t=1..T} \alpha_t c_t(x) \geq \frac{1}{2} \Sigma_{t=1..T} \alpha_t$,
0 otherwise
where $\alpha_t = \log 1/\beta_t$

51

Haar – Classifier cascade

- Object recognition is possible with a single strong classifier
- To improve detection rates AND to reduce computation time:
  - A cascade of strong classifiers can be used
  - Each stage either accepts or rejects
  - Only those accepted pass to the next stage
    - Efficient computation…
  - Strong classifiers trained using AdaBoost
    - On the remaining set of data

52

Haar – Recognition

- Face recognition
  - 38 stages
  - 6000+ features
  - 4916 positive samples
  - 9544 negative samples
  - Scale independence

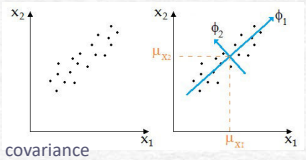
53

Principal Component Analysis (PCA)

- Statistical Technique for
  - Data analysis
  - Data compression
  - Recognition
- Analyses data covariance
- Identifies principal directions

For example:

- Given *2* dimensional data
- and *N* samples/vectors
- Find
  - the mean sample
  - the direction of maximum covariance
  - the direction orthogonal to this



54

PCA – 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 eigenvalue λ there will be an eigenvector x such that

$$Ax = \lambda x$$

  - Many possible values of x

- 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 $Ax_2 = \lambda_2 x_2$

55

PCA – Background – Eigenvalues and Eigenvectors

- 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 $A\Phi = \Phi\Lambda$
  - 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 $\Phi^T A\Phi = \Phi^T\Phi\Lambda = \Lambda$
  - and $A = A\Phi\Phi^T = \Phi\Lambda\Phi^T$

56

PCA – Theory – Organise Data

- In PCA we have *N* samples/vectors in some *n*-dimensional space
- Combine into a data matrix *D*
  - Each row is a sample
- Determine the mean of the samples
- Compute mean-centred data *U*

$$D = \begin{bmatrix} x_1^1 & x_2^1 & .. & x_n^1 \\ x_1^2 & x_2^2 & .. & x_n^2 \\ : & : & & : \\ x_1^N & x_2^N & .. & x_n^N \end{bmatrix}$$

$$\mu = \begin{bmatrix} \sum_{i=1}^{N} x_1^i /_N & \sum_{i=1}^{N} x_2^i /_N & .. & \sum_{i=1}^{N} x_n^i /_N \end{bmatrix}$$

$$U = D - \begin{bmatrix} \mu \\ \mu \\ : \\ \mu \end{bmatrix} = \begin{bmatrix} x_1^1 - \sum_{i=1}^{N} x_1^i /_N & x_2^1 - \sum_{i=1}^{N} x_2^i /_N & .. & x_n^1 - \sum_{i=1}^{N} x_n^i /_N \\ x_1^2 - \sum_{i=1}^{N} x_1^i /_N & x_2^2 - \sum_{i=1}^{N} x_2^i /_N & .. & x_n^2 - \sum_{i=1}^{N} x_n^i /_N \\ : & : & & : \\ x_1^N - \sum_{i=1}^{N} x_1^i /_N & x_2^N - \sum_{i=1}^{N} x_2^i /_N & .. & x_n^N - \sum_{i=1}^{N} x_n^i /_N \end{bmatrix}$$

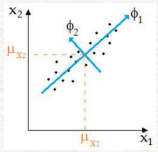
57

PCA – Theory – Covariance matrix

- Using the mean-centred data *U*, compute the covariance matrix Σ

$$\Sigma = {}^{U^T U}/_{(N-1)}$$

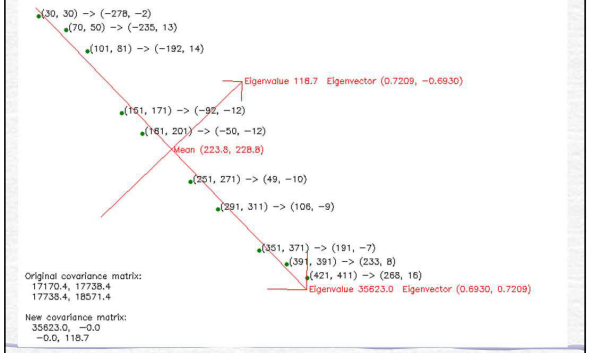
- We can then determine $\Phi^T \Sigma \Phi = \Lambda$
  - where the eigenvectors are in an orthogonal matrix Φ
  - and the eigenvalues are in an ordered diagonal matrix Λ



- We can consider Φ as a linear transformation
  - e.g. in our example it transforms $[x_1, x_2]$ to $[\phi_1, \phi_2]$
- The amplitudes of the eigenvalues in Λ are proportional to the percentage of overall variance accounted for by the associated eigenvector.

58

PCA – Example



59

PCA – Basics

```
float samples[][2] = { {30,30}, {70,50}, …, {421,411}, {391,391} };
int number_of_samples = sizeof(samples)/sizeof(int[2]);
Mat samples_matrix(number_of_samples, 2, CV_32FC1, samples);
…
PCA pca(samples_matrix, Mat(), 0, 2 );
Mat eigenvalues = pca.eigenvalues;
Mat eigenvectors = pca.eigenvectors;
Mat mean = pca.mean;
```

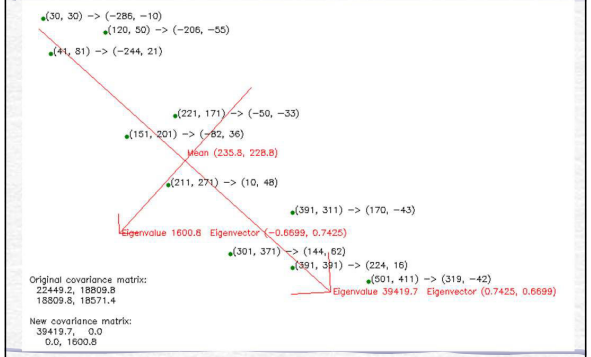

60

PCA – Basics

```
Mat transformed_samples_matrix = samples_matrix.clone();
for (int sample_no=0; sample_no<number_of_samples; sample_no++)
{
  Mat sample = samples_matrix.row(sample_no);
  Mat transformed_sample;
  pca.project(sample, transformed_sample);
  transformed_sample.row(0).copyTo(
                                    transformed_samples_matrix.row(sample_no));
}
Mat covariance,average,new_covariance,new_average;
calcCovarMatrix(samples_matrix,covariance,average,
                                    CV_COVAR_NORMAL | CV_COVAR_ROWS);
calcCovarMatrix(transformed_samples_matrix,new_covariance,
                    new_average,CV_COVAR_NORMAL | CV_COVAR_ROWS);
```


61

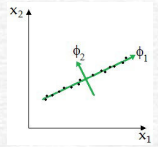
PCA – Example



62

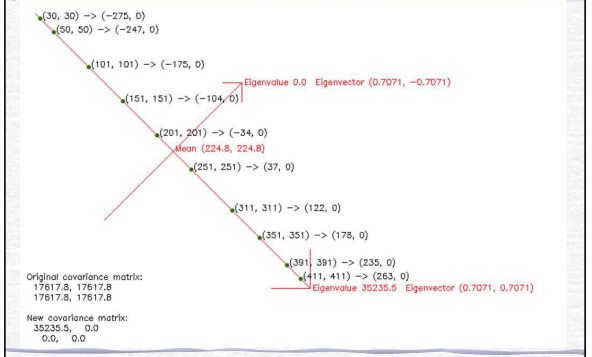
PCA – Compression

- If the variance on an axis is very small
  - Variance caused by noise?
  - Drop this dimension?



63

PCA – Example

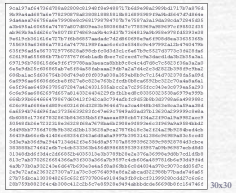


64

PCA – Face Recognition

- Common example application of PCA
- Each face image is treated as a vector
  - with *n=rows*columns* dimensions!

=



30x30

= [9c, 1a, 97, af, …, 74, 6f]$_{900}$

- In the following example… *n = 10304*

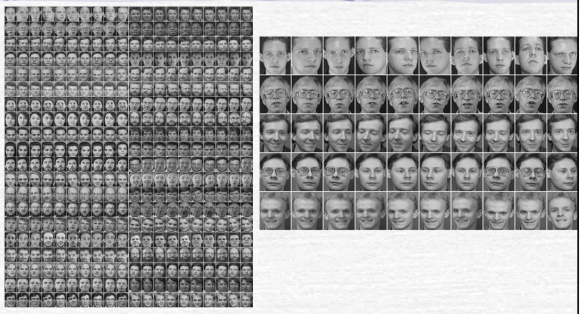
65

PCA – Face Recognition

- Create a matrix *D*
  - where each face image is a row
  - *D* is a *N x n* matrix
- Create mean centred data *U*
  - Determine the mean of the rows (face images)
  - Subtract from all of the rows
- Compute the covariance matrix $\Sigma = {U^T U}/{(N-1)}$
- Solve for Eigenvectors and Eigenvalues as before.
- Normalise the Eigenvectors
- Select *m* Eigenvectors (with the largest *m* Eigenvalues)
  - Usually 20-50 for face recognition.
- For an unknown image find its location in PCA space
  - Look for the closest match $p_\phi = (p_x - \mu_x).\Phi_{pca}^T$

66

PCA – Face Recognition – Database



Original images provided by AT&T Laboratories Cambridge
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

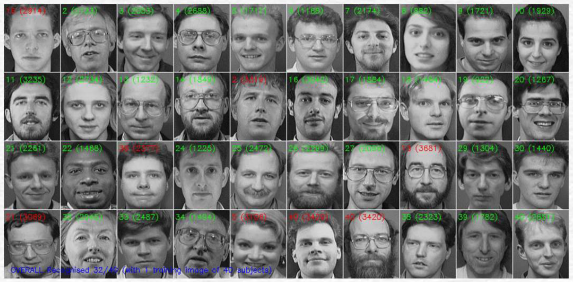
67

PCA – Face Recognition – Some known faces

Original images provided by AT&T Laboratories Cambridge
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

68

PCA – Face Recognition – Recognised faces



Original images provided by AT&T Laboratories Cambridge (Composite recognised images © Kenneth Dawson-Howe
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

69

PCA – Face Recognition – Training & Prediction

```
vector<Mat> known_images;          vector<int> known_labels;
Mat current_image, unknown_image;
int current_label;
...
known_images.push_back(current_image);
known_labels.push_back(current_label);
...
Ptr<FaceRecognizer> recogniser = createEigenFaceRecognizer();
recogniser->train(known_images, known_labels);
...
int predicted_label = 0;          double confidence = 0.0;
recogniser->predict(unknown_image,predicted_label,confidence);
```


70

PCA – Image Reconstruction using Eigenvectors

- We can reconstruct images from PCA space

$$p_x = p_\phi . \Phi_{pca}^T + \mu_x$$

- Using some number (*m*) of principal components
- So that some percentage of variance is accounted for

$$\%\ of\ Total\ Variance = 100. \sum_{i=1}^{m} \lambda_i \Big/ \sum_{j=1}^{n} \lambda_j$$

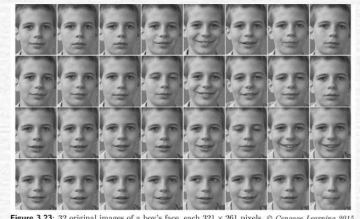


Figure 3.23: 32 original images of a boy's face, each 321 × 261 pixels. © *Cengage Learning 2015.*

Figure 3.24: Reconstruction of the image from four basis vectors $\mathbf{b}_i$, $i = 1, \ldots, 4$ which can be displayed as images. The linear combination was computed as $q_1\mathbf{b}_1 + q_2\mathbf{b}_2 + q_3\mathbf{b}_3 + q_4\mathbf{b}_4 = 0.078\,\mathbf{b}_1 + 0.062\,\mathbf{b}_2 - 0.182\,\mathbf{b}_3 + 0.179\,\mathbf{b}_4$. © *Cengage Learning 2015.*

71