Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 14

Object Recognition

Template matching:
normalized cross-correlation, chamfer matching

Image Analysis

- Automatically extracting useful information from an image of a scene
- We can also classify the types of analysis we wish to perform according to function
  - Inspection

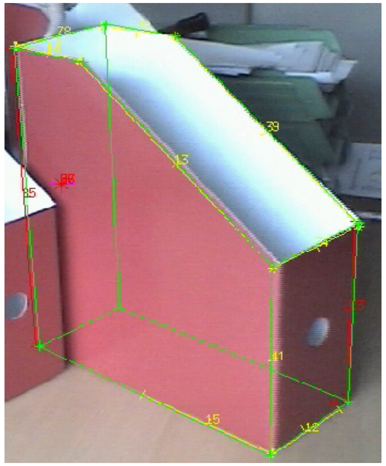
    Is the visual appearance of objects as it should be?
  - Location

    requires the specification of both position and orientation in either 2D or 3D.
    - image frame of reference (pixels)
    - real world frame of reference (e.g. millimetres) … calibration required
  - Identification of object type

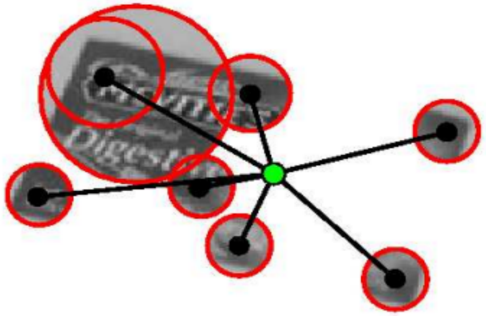
Approaches to Object Recognition

- Generic Gestalt Principles
  - The world is structured, extract features
  - perceptual grouping

- Model based
  - CAD model of object
  - Geometric features
  - Locate features and their arrangement



- Appearance based
  - Interest points / point features
  - or "whole" object

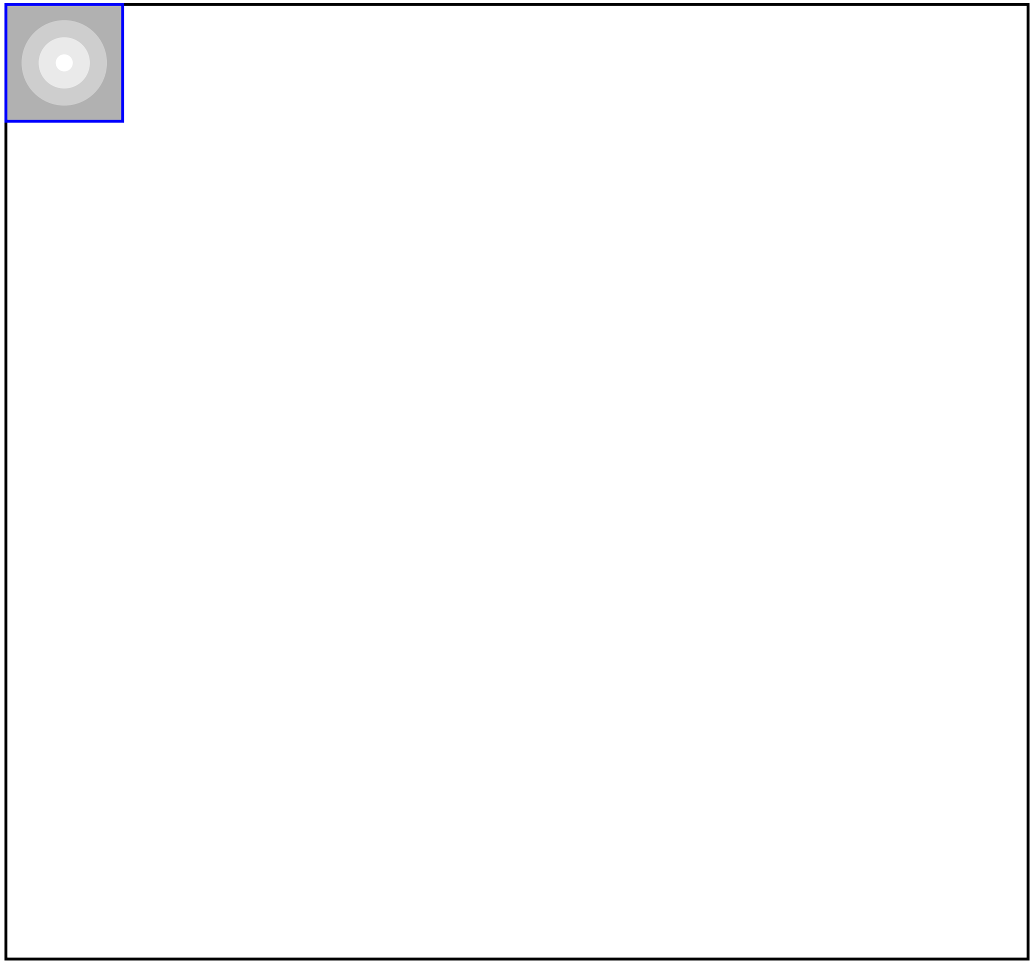


Credit: Markus Vincze, Technische Universität Wien

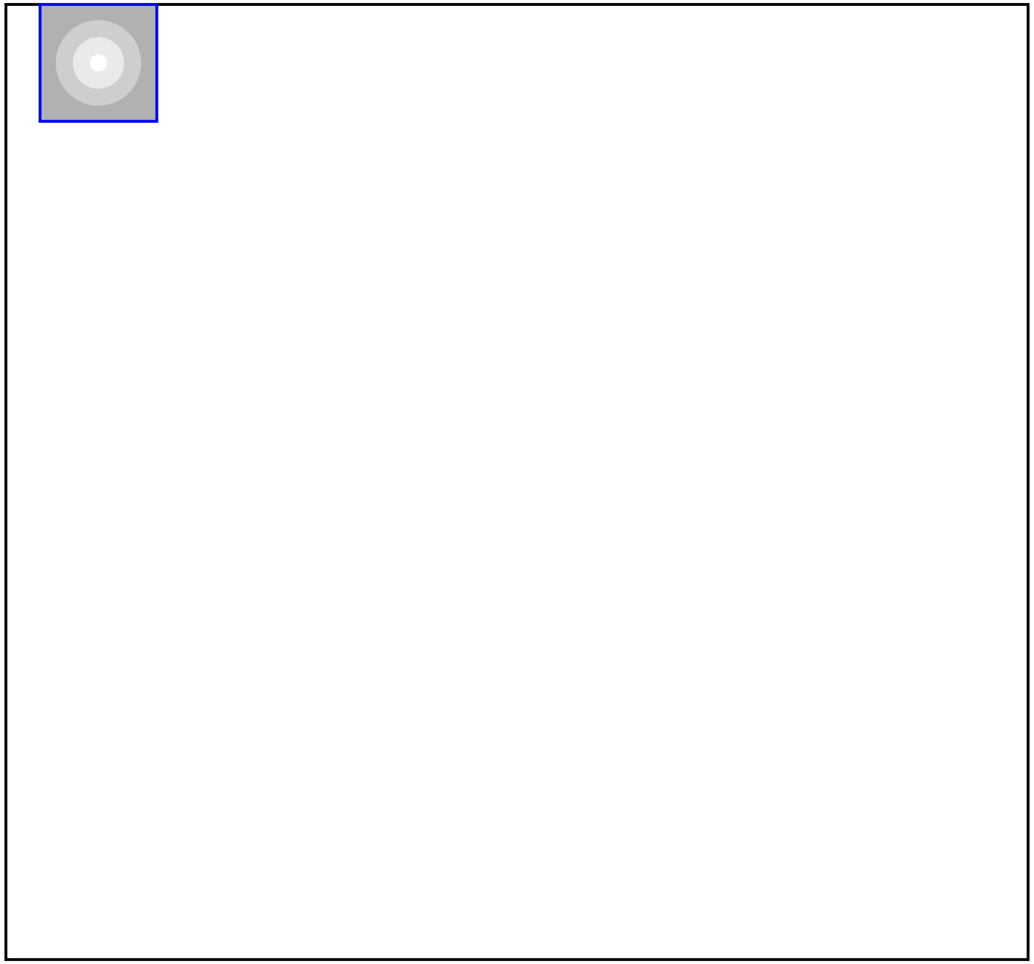
Template Matching

- Many applications of computer vision simply need to know whether an image contains some previously defined object
  - whether a pre-defined sub-image is contained within a test image
- This sub-image is called a Template
  - an ideal representation of the pattern/object which is being sought in the image

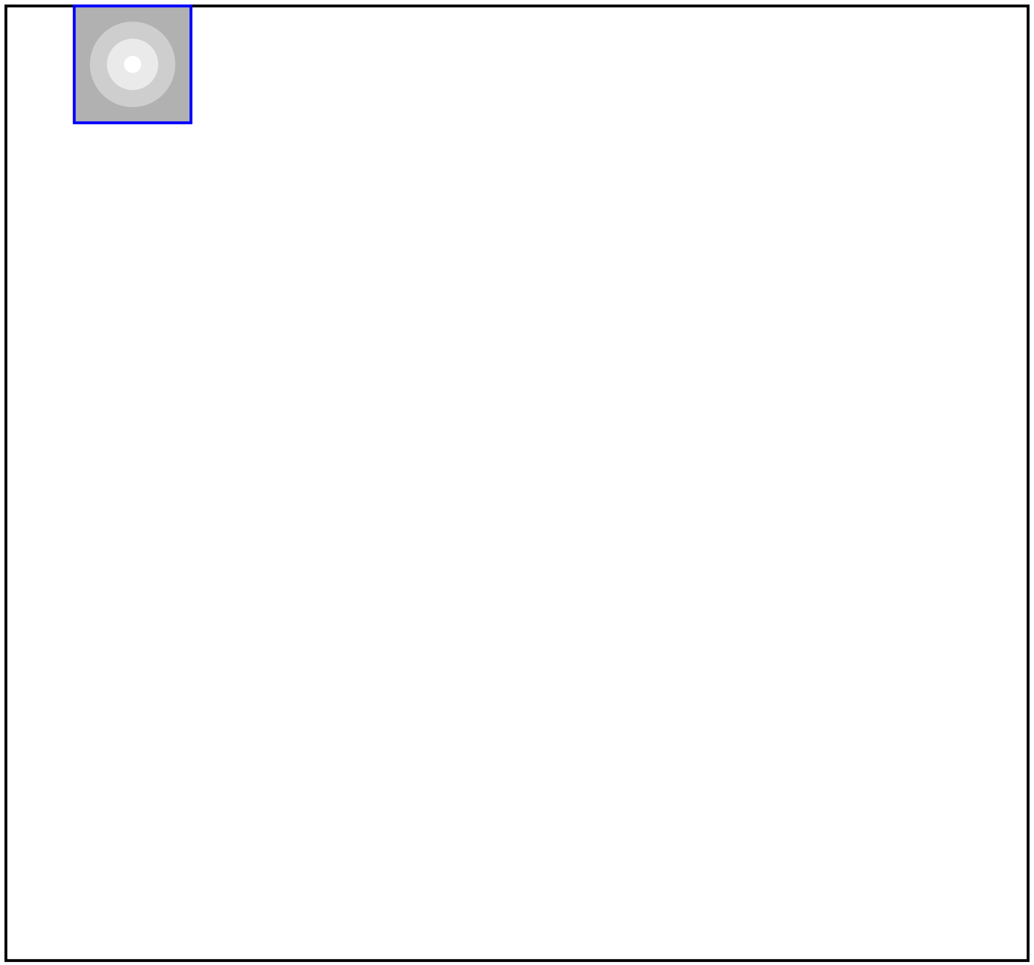
Template Matching



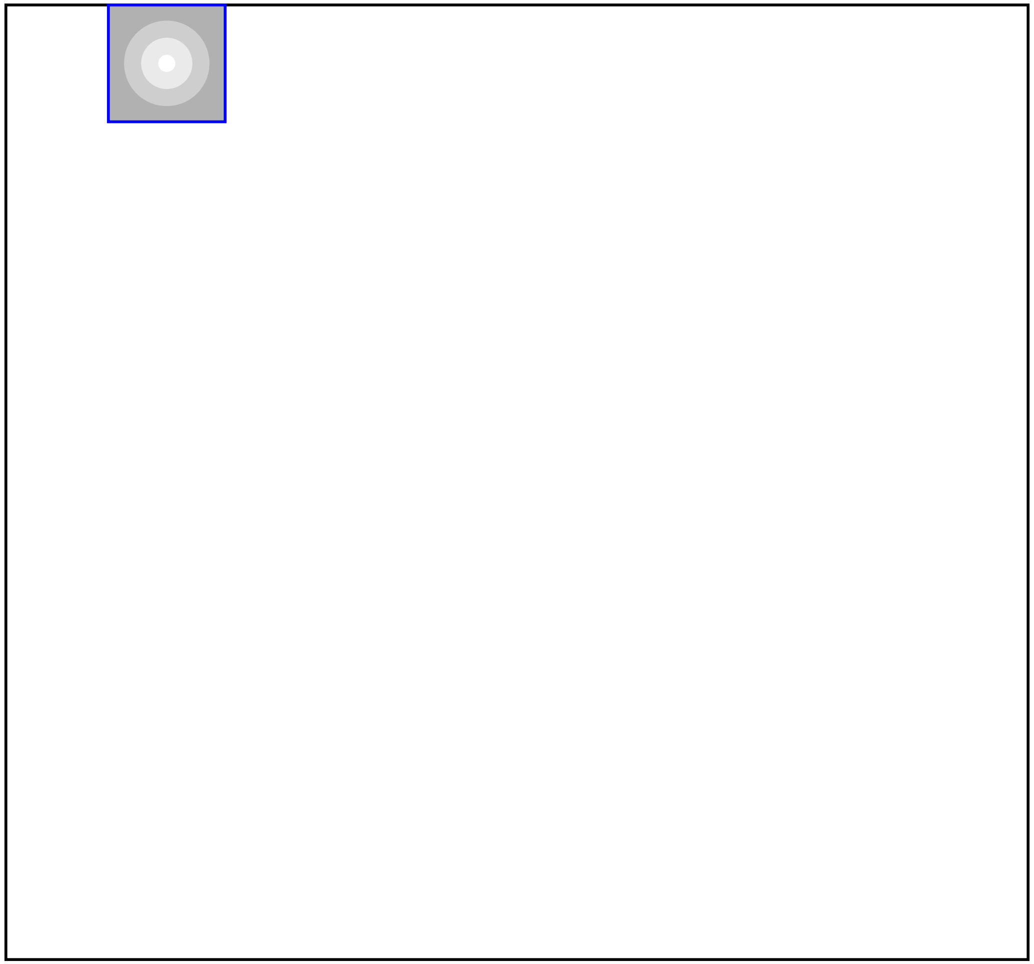
Template Matching



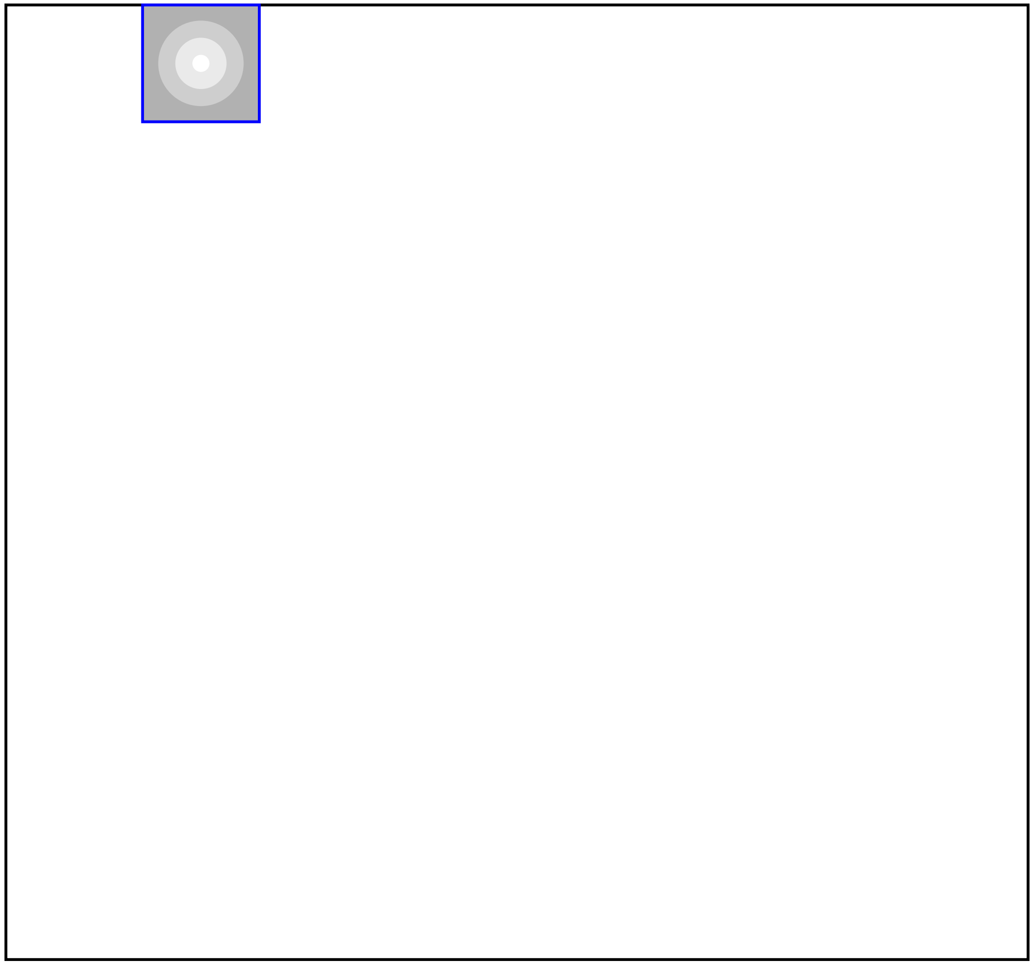
Template Matching



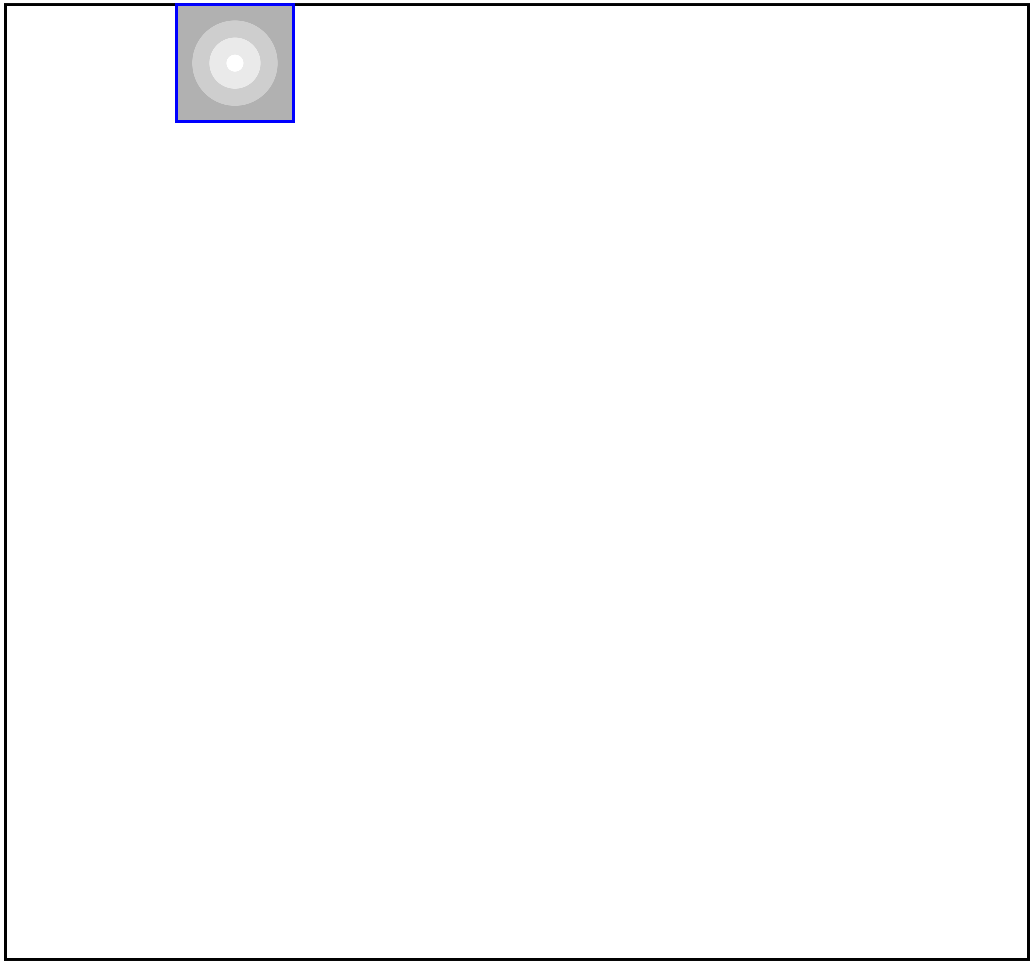
Template Matching



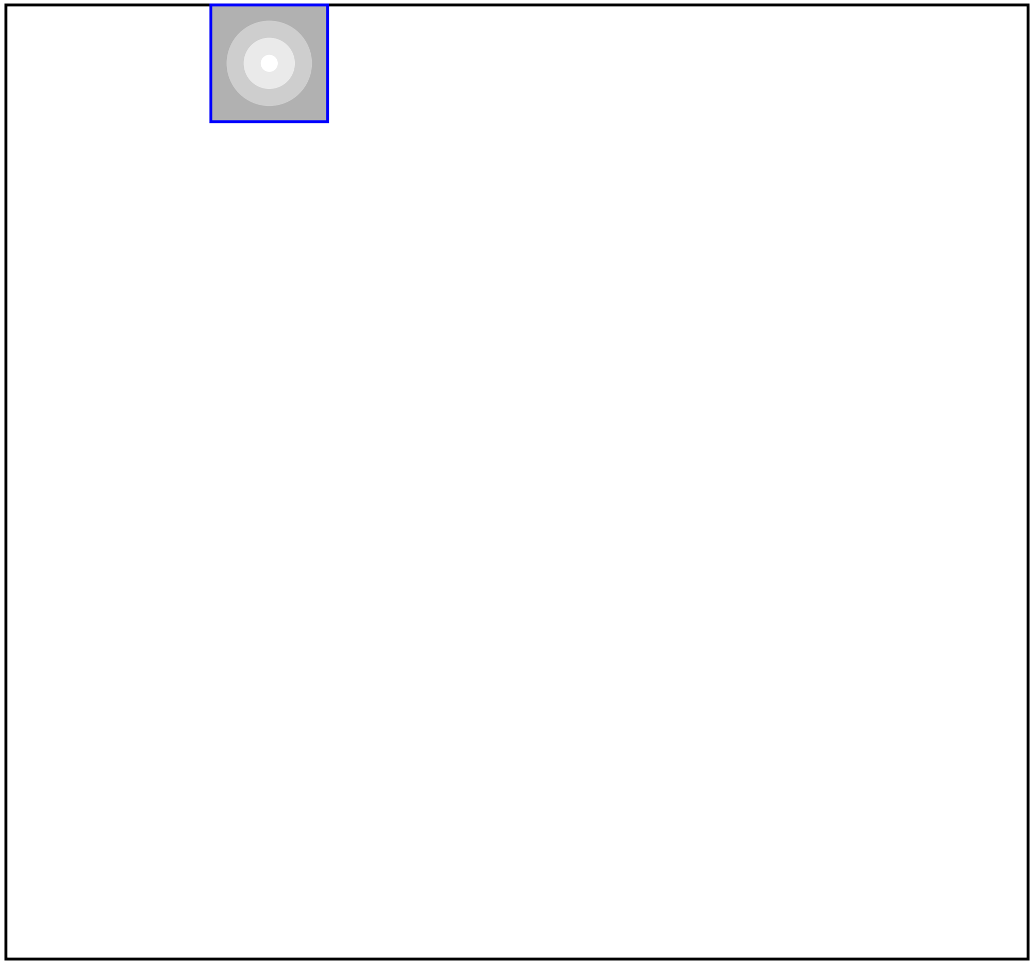
Template Matching



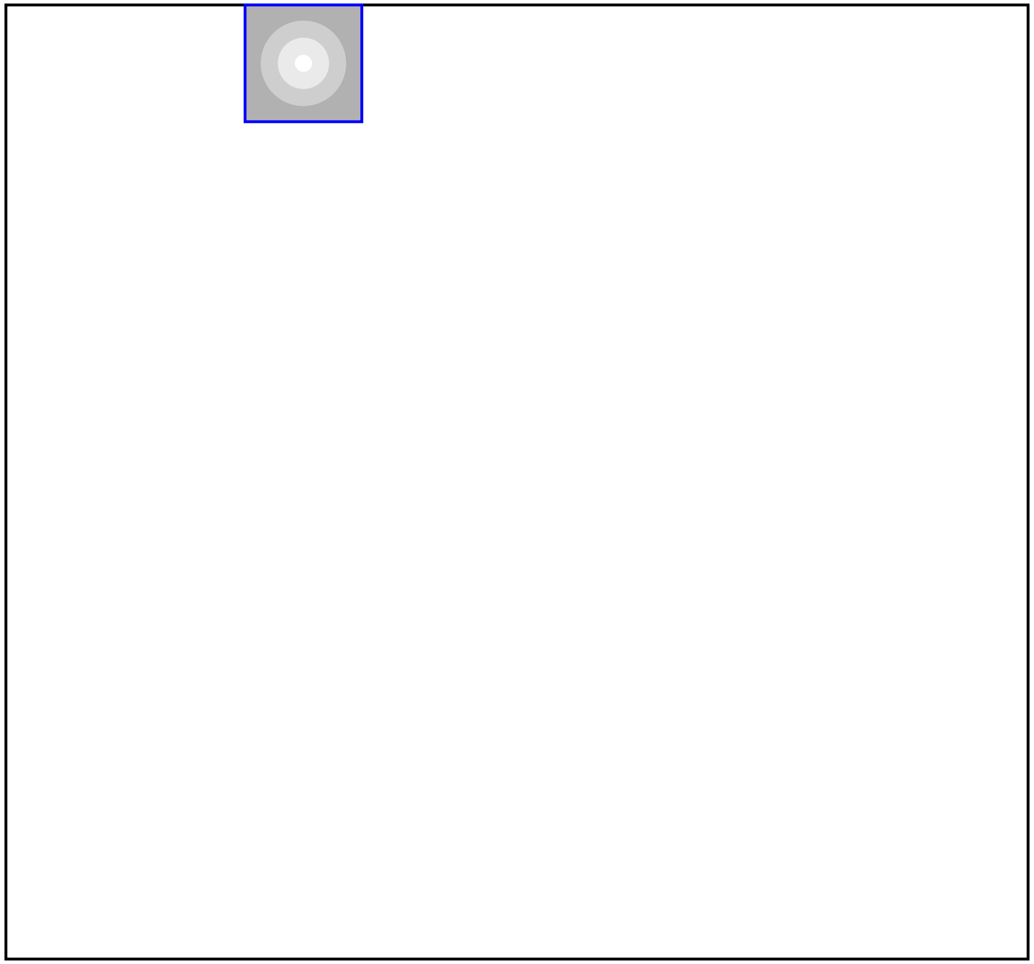
Template Matching



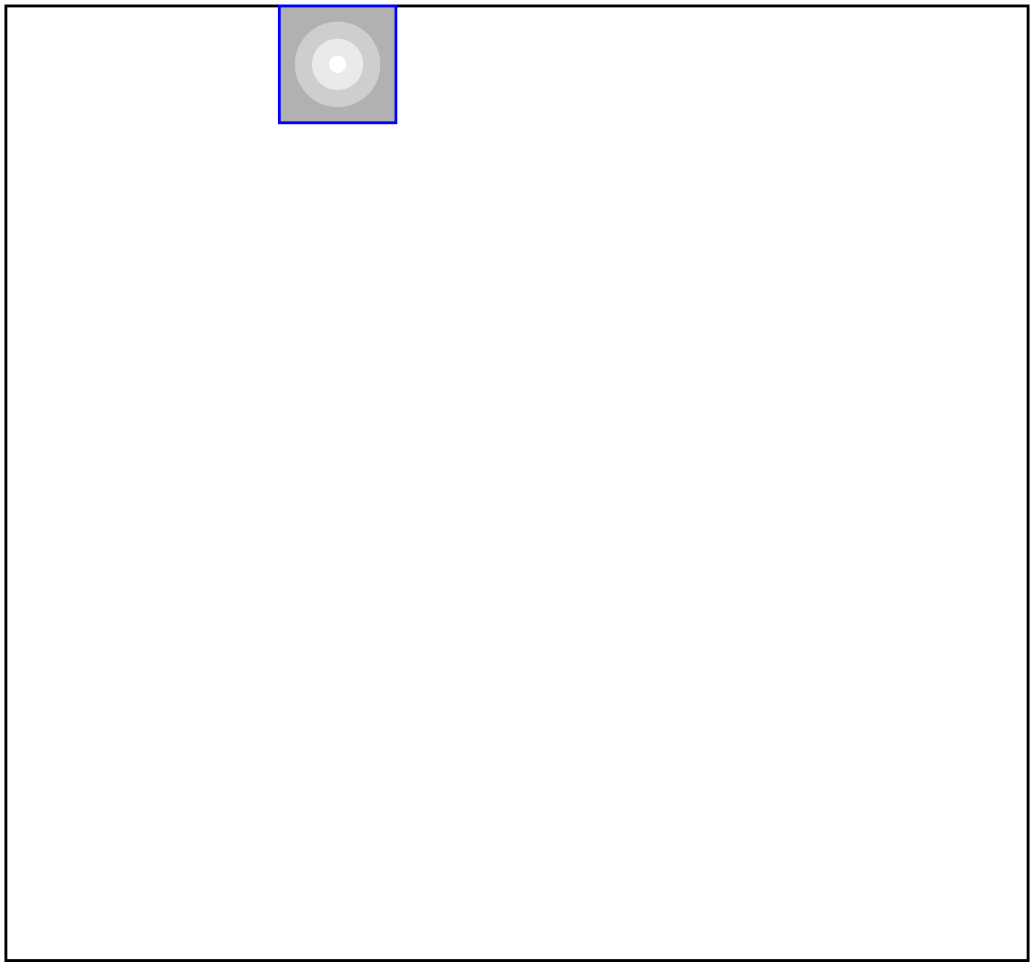
Template Matching



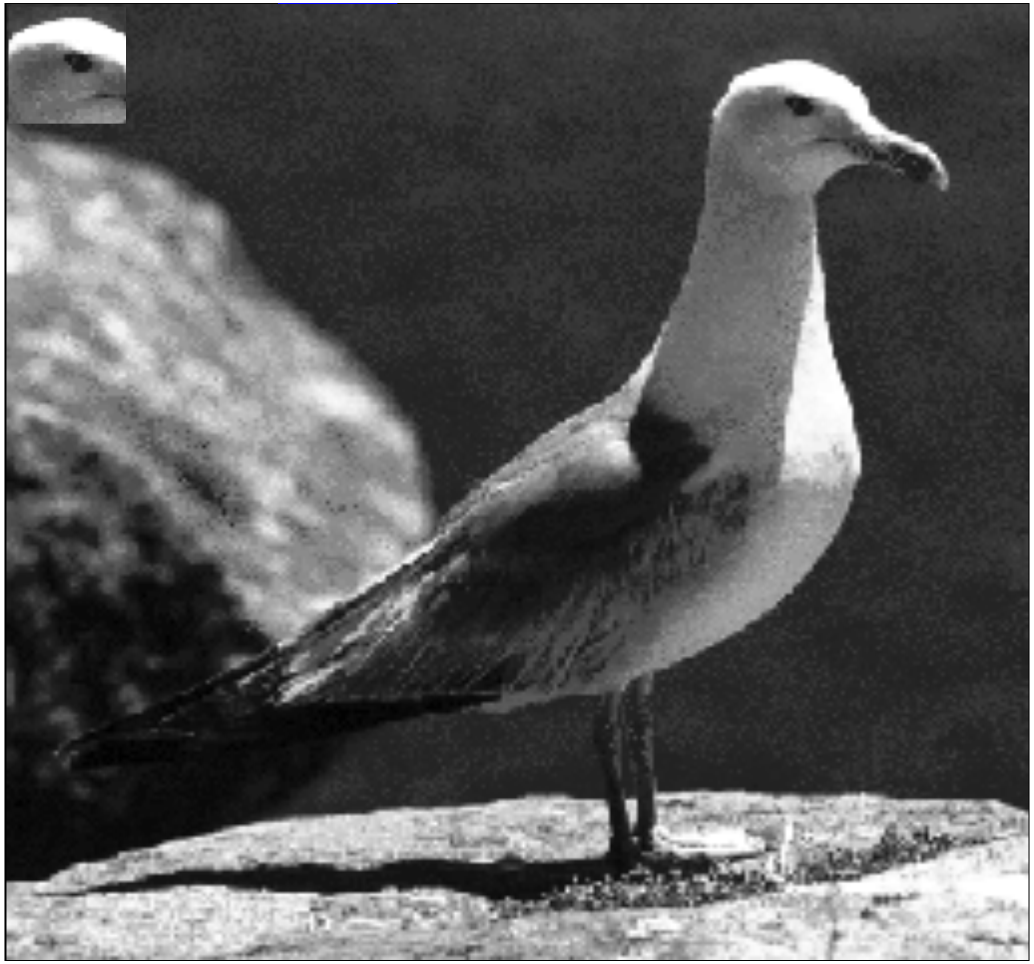
Template Matching



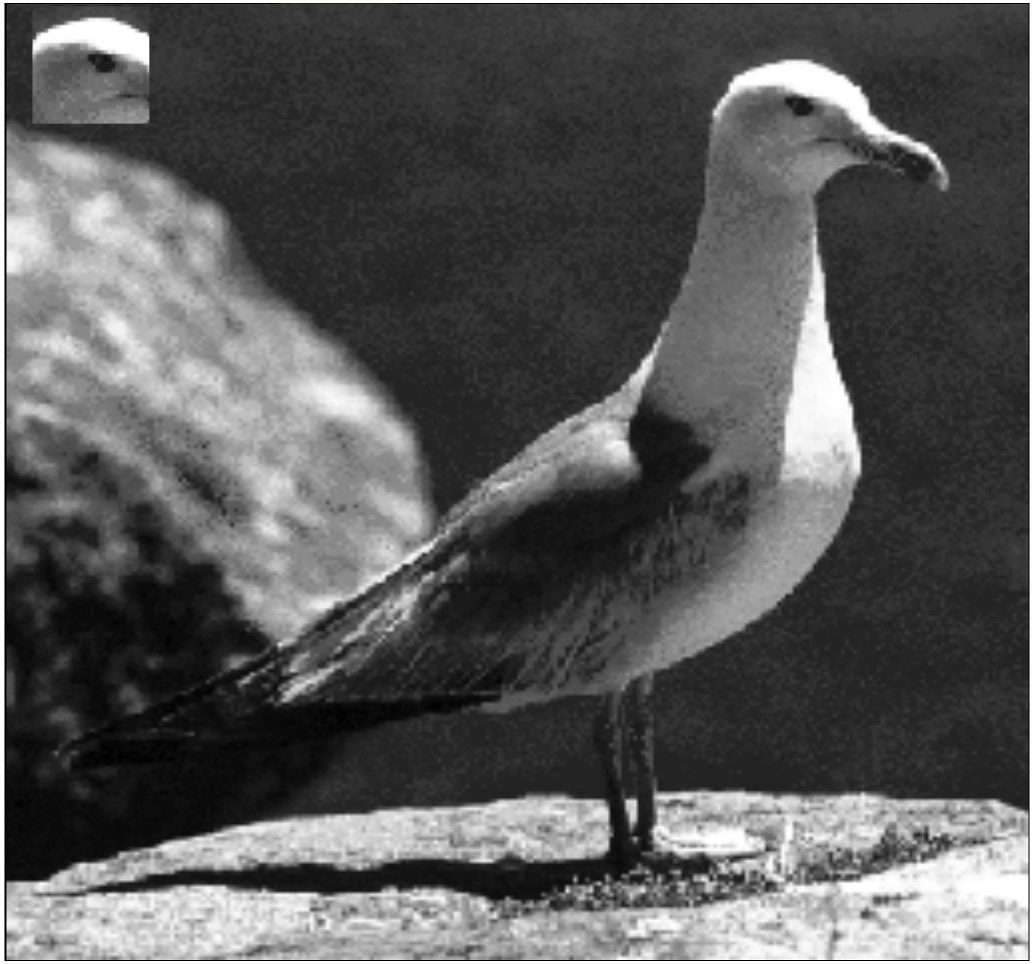
Template Matching



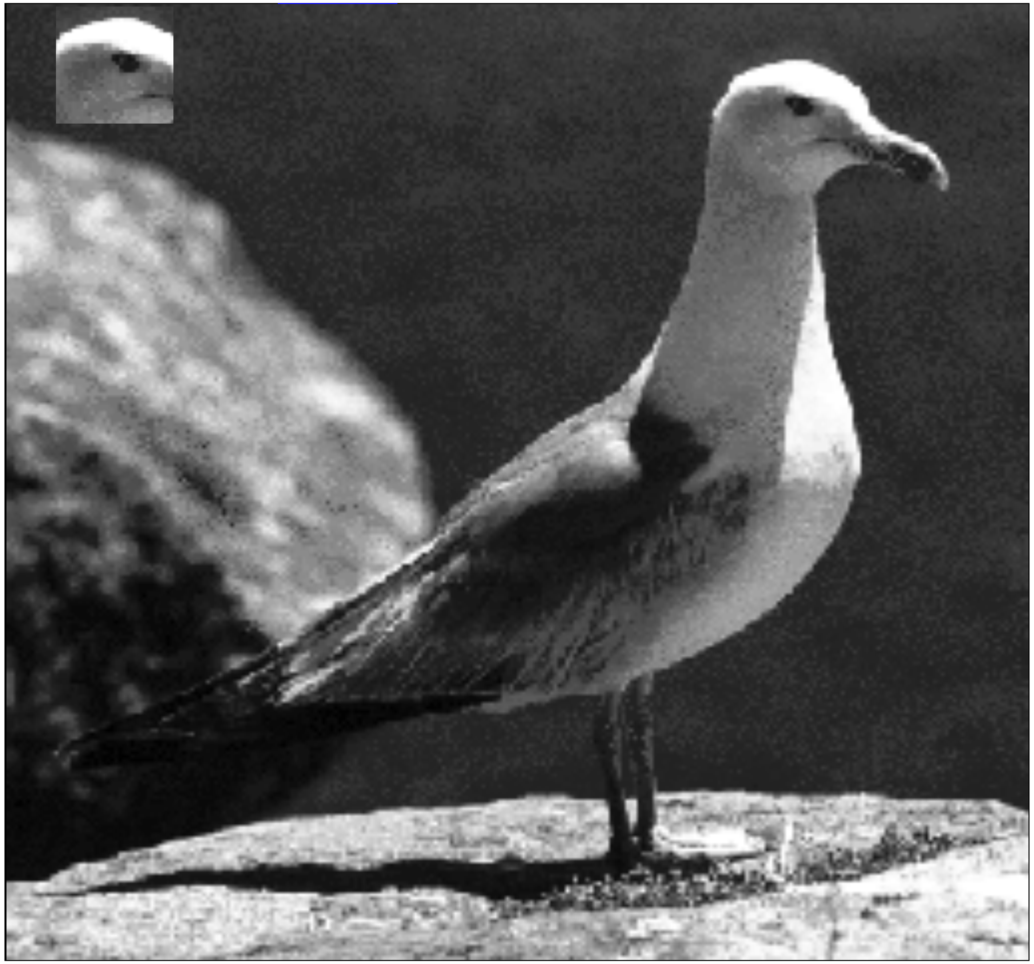
Template Matching



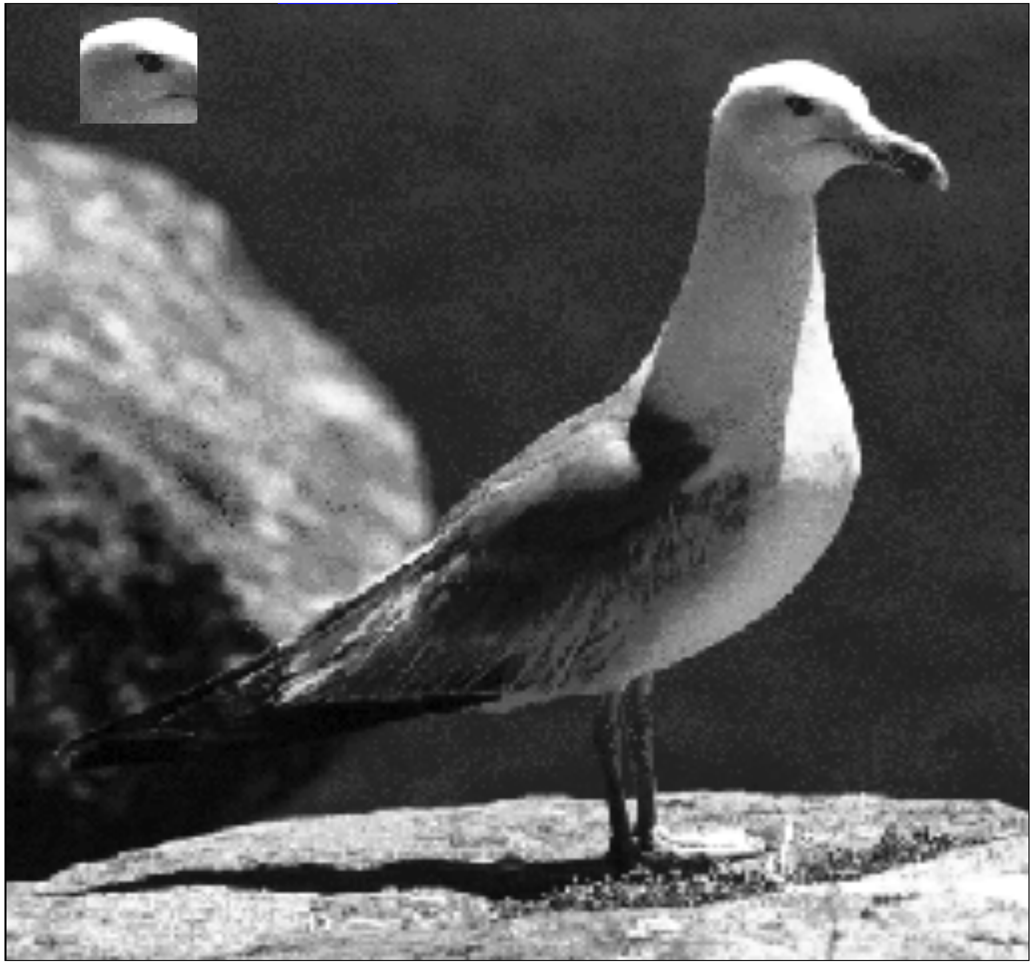
Template Matching



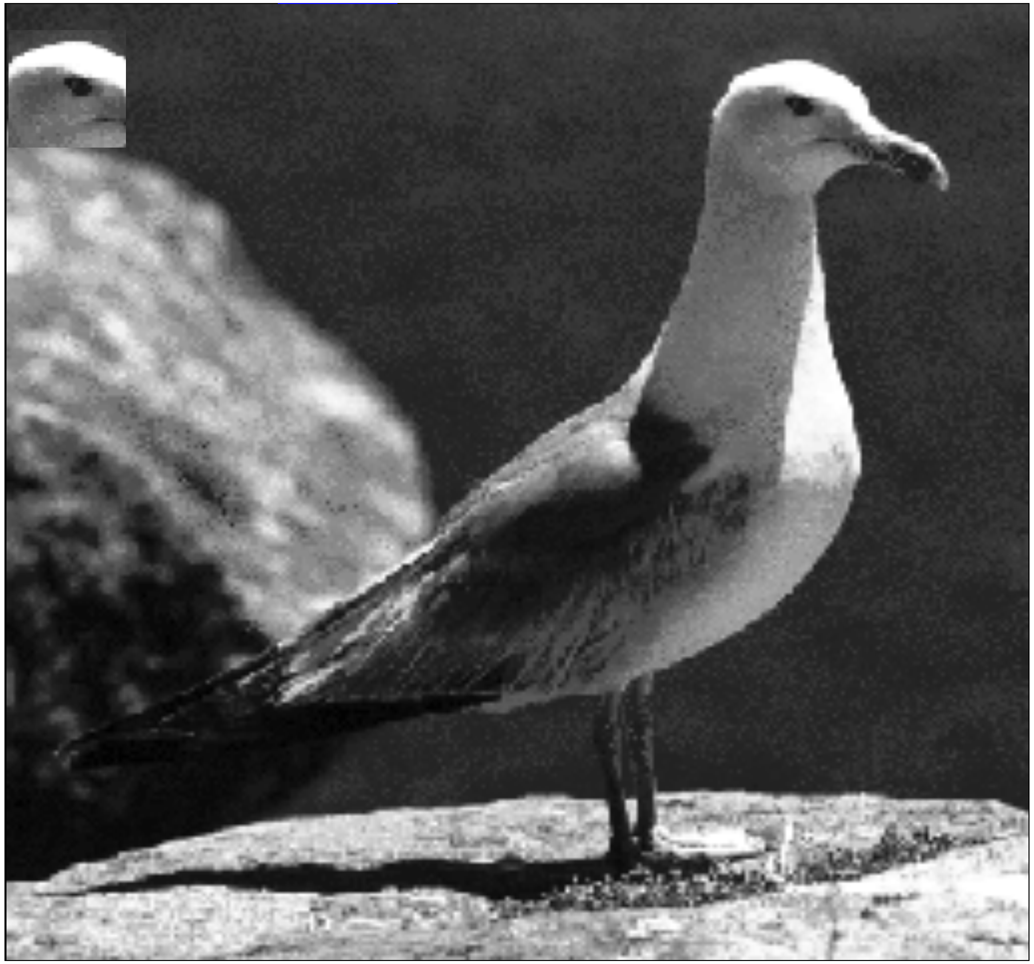
Template Matching



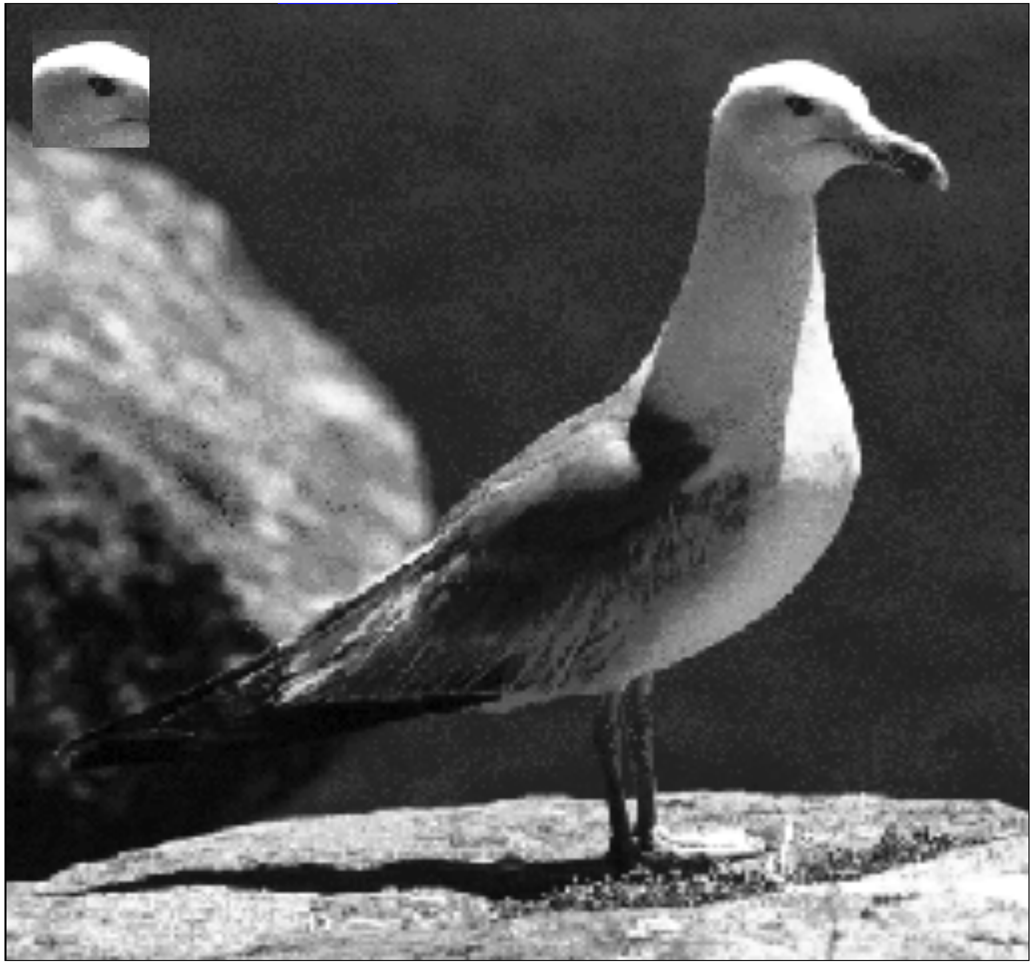
Template Matching



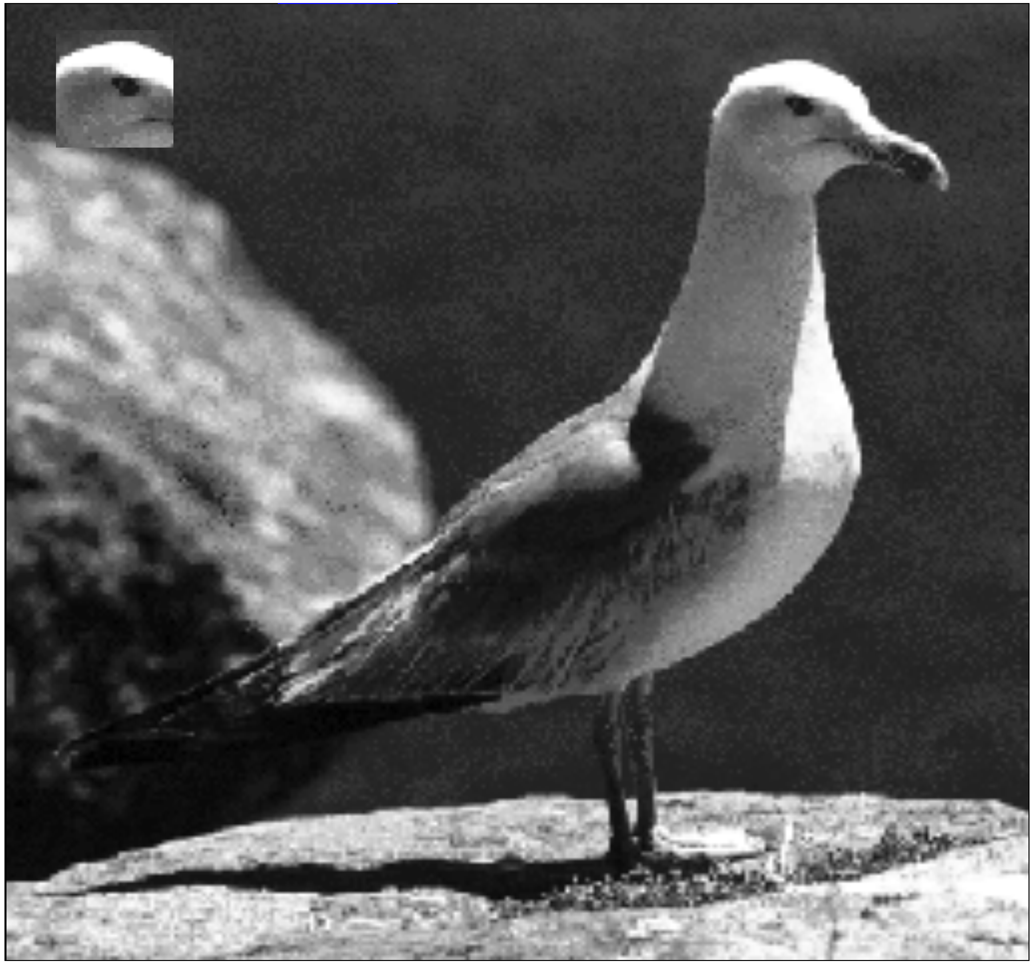
Template Matching



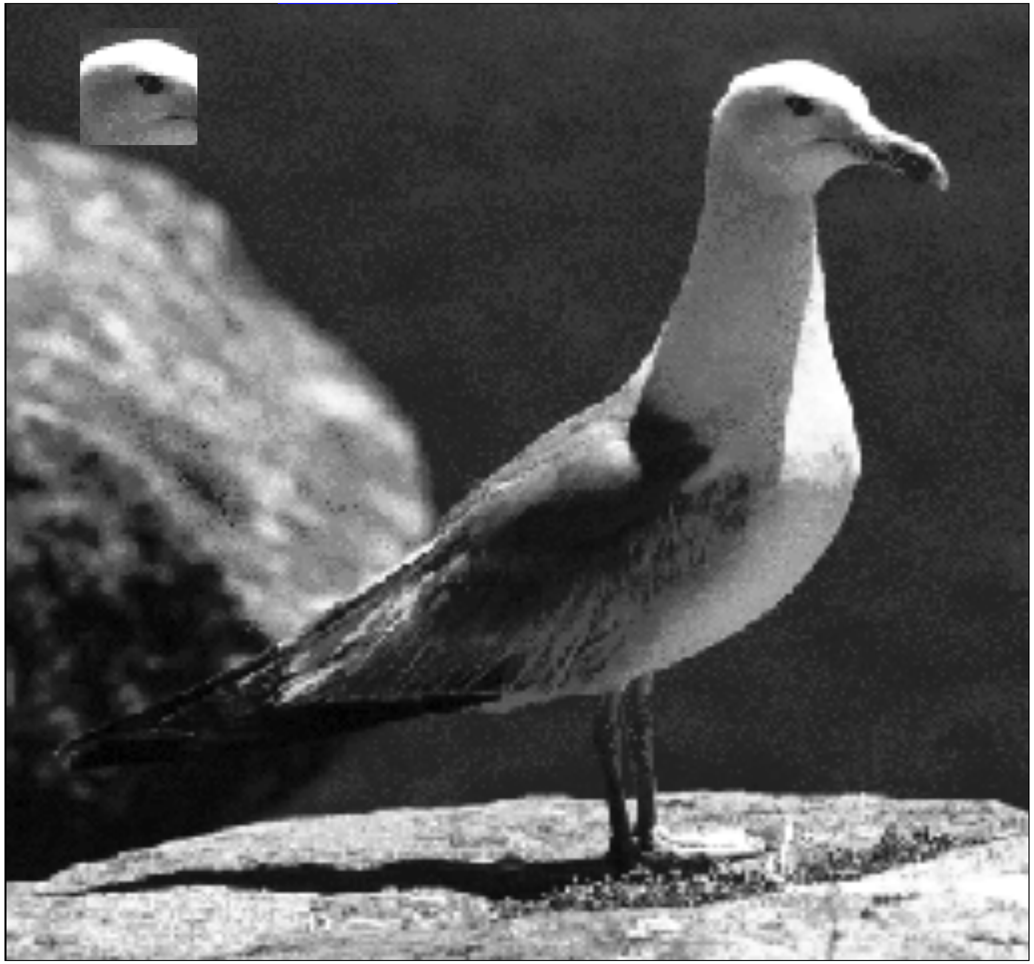
Template Matching



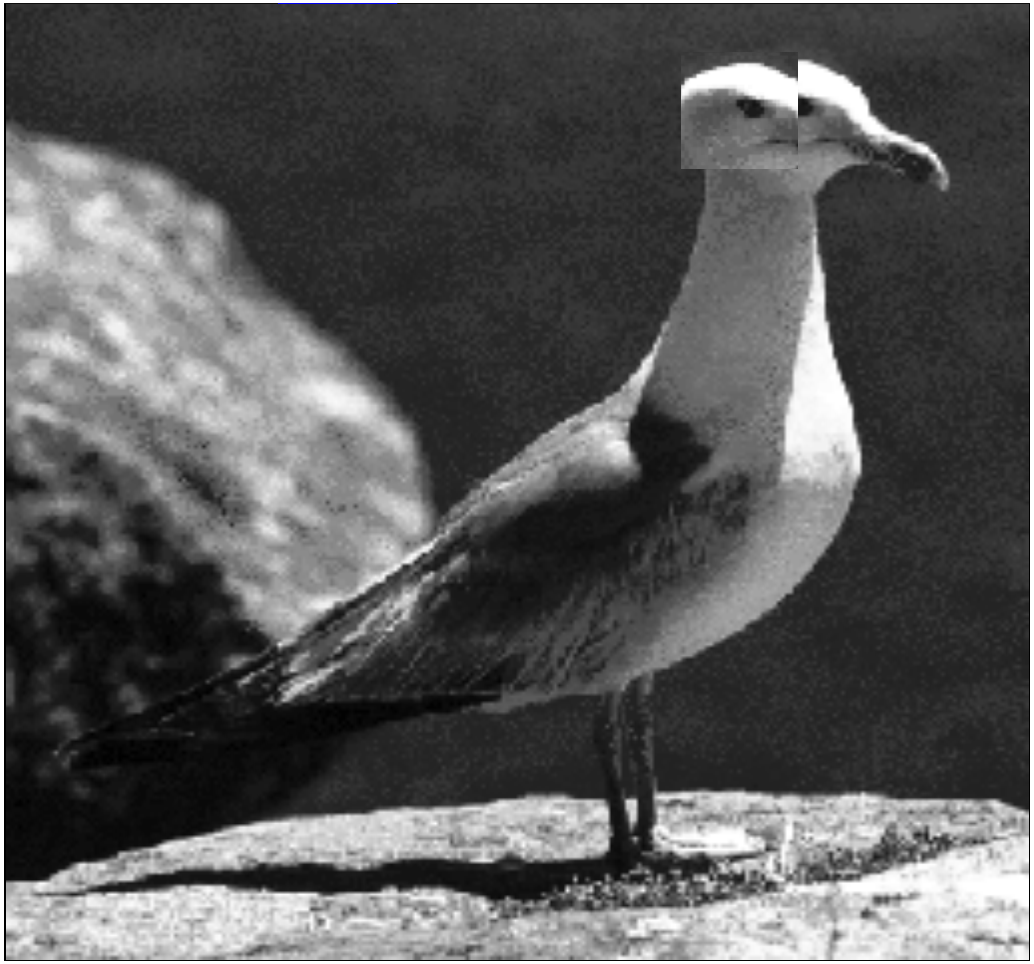
Template Matching



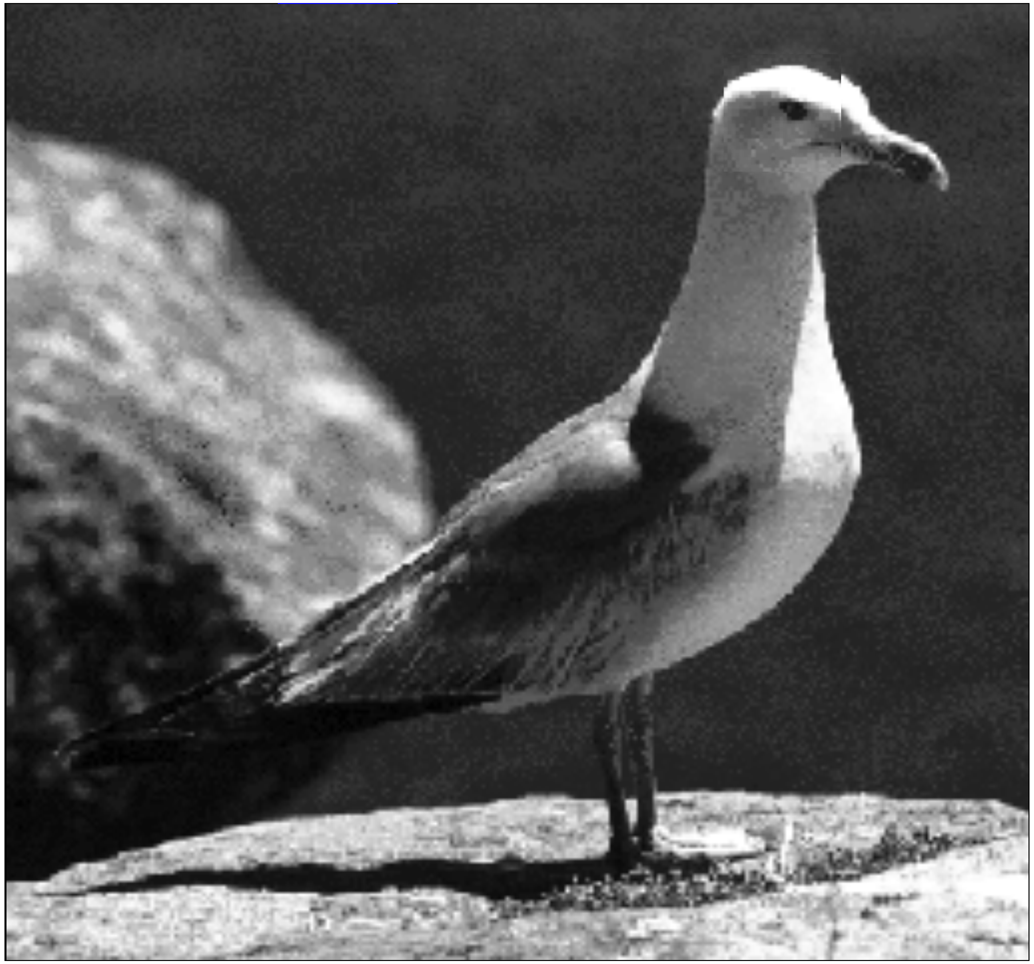
Template Matching



Template Matching



Template Matching



Template Matching

The template matching technique :

- Translate the template to every possible position in the image
- Compute a measure of the match between the template and the image at that position
- If the similarity measure is large enough then the object can be assumed to be present

Template Matching

- Global Template Matching

  If the template represents the complete object

- Local template matching

  Uses several templates of local features of the object, *e.g.* corners in the boundary or characteristic marks, to represent the object

Template Matching

Several similarity measures

- some based on the summation of differences between the image and template
- other based on cross-correlation techniques

Template Matching

Euclidean distance between the template $t(i, j)$ and the test image $g(i, j)$

$$E(m,n) = \sqrt{\sum_i \sum_j \left[g(i,j) - t(i-m, j-n)\right]^2}$$

- The summation is evaluated for all i, such that $(i\text{-}m)$ is a valid co-ordinate of the template sub-image
- This definition amounts to translating the template $t(i, j)$ to a position (m, n) along the test image and evaluating the similarity measure at that point
- The position (m, n) at which the smallest value of $E(m, n)$ is obtained corresponds to the best match for the template

Template Matching

- To compare the difference in size of two 1D objects we
  - subtract the values
  - square the difference
  - take the square root of the result
  - leaving us with the absolute difference in size

$$d = \sqrt{\left(s_1 - s_2\right)^2}$$

- Extending this to the 2D case, we might wish to see how far apart two objects are on a table, *i.e.* to compute the distance between them

$$d = \sqrt{\left(\left(x_1 - x_2\right)^2 + \left(y_1 - y_2\right)^2\right)}$$

- Similarly, in 3D

$$d = \sqrt{\left(\left(x_1 - x_2\right)^2 + \left(y_1 - y_2\right)^2 + \left(z_1 - z_2\right)^2\right)}$$

Template Matching

- We can extend this to n dimensions by just making each co-ordinate an independent variable which characterizes the entities we are comparing
- For example, a 10*10 image template comprises 100 independent pixels, each of which specifies the template sub-image
- Thus, we are now dealing with a 10 * 10 = 100 dimensional comparison and the difference between the two sub-images is

$$d = \sqrt{\left(\left(image(1,1) - template(1,1) \right)^2 + \ldots + \left(image(10,10) - template(10,10) \right)^2 \right)}$$

which is identical to our definition of the Euclidean metric.

Template Matching

- A frequently-used and simpler template-matching metric is based on the absolute difference of $g(i, j)$ and $t(i\text{-}m, j\text{-}n)$ rather than the square of the difference
- It is defined by

$$S(m,n) = \sum_i \sum_j \left| g(i,j) - t(i-m, j-n) \right|$$

- As before, the summation is evaluated for all i and j, such that $(i\text{-}m, j\text{-}n)$ is a valid co-ordinate of the template sub-image
- Note that the summation of the last term is constant since it is a function of the template only and is evaluated over the complete domain of the template

Template Matching

- Returning again to the root of sum of squares measure

$$E(m,n) = \sqrt{\sum_i \sum_j \left[g(i,j) - t(i-m, j-n) \right]^2}$$

- If we expand the square, we get terms in $g^2()$, $-g()\ t()$, $t^2()$

- The summation of the last term is constant

  – it is a function of the template only
  – it is evaluated over the complete domain of the template

- If it is assumed that the first term is also constant, or that the variation is small enough to be ignored, then $E^2(m,n)$ is small when the summation of the middle mixed term is large

Template Matching

- Thus, a new similarity measure might be $R(m, n)$, given by

$$R(m,n) = \sum_i \sum_j g(i,j)\,t(i-m, j-n)$$

- again summing over the usual range of i and j

- $R(m, n)$ is the familiar cross-correlation function

- The template $t(i\text{-}m, j\text{-}n)$ and the section of $g(i, j)$ in the vicinity of (m, n) are similar when the cross-correlation is large

Template Matching

- If the assumption that the summation of $g(i, j)$ is independent of m and n is not valid, an alternative to computing R is to compute the normalised cross-correlation $N(m, n)$, given by

$$N(m,n) = \frac{R(m,n)}{\sqrt{\sum_i \sum_j g(i,j)^2}}$$

… summing over the usual range *i* and *j*

Note that (by the Cauchy-Schwarz inequality)

$$N(m,n) \leq \sqrt{\sum_i \sum_j t(i-m, j-n)^2}$$

Template Matching

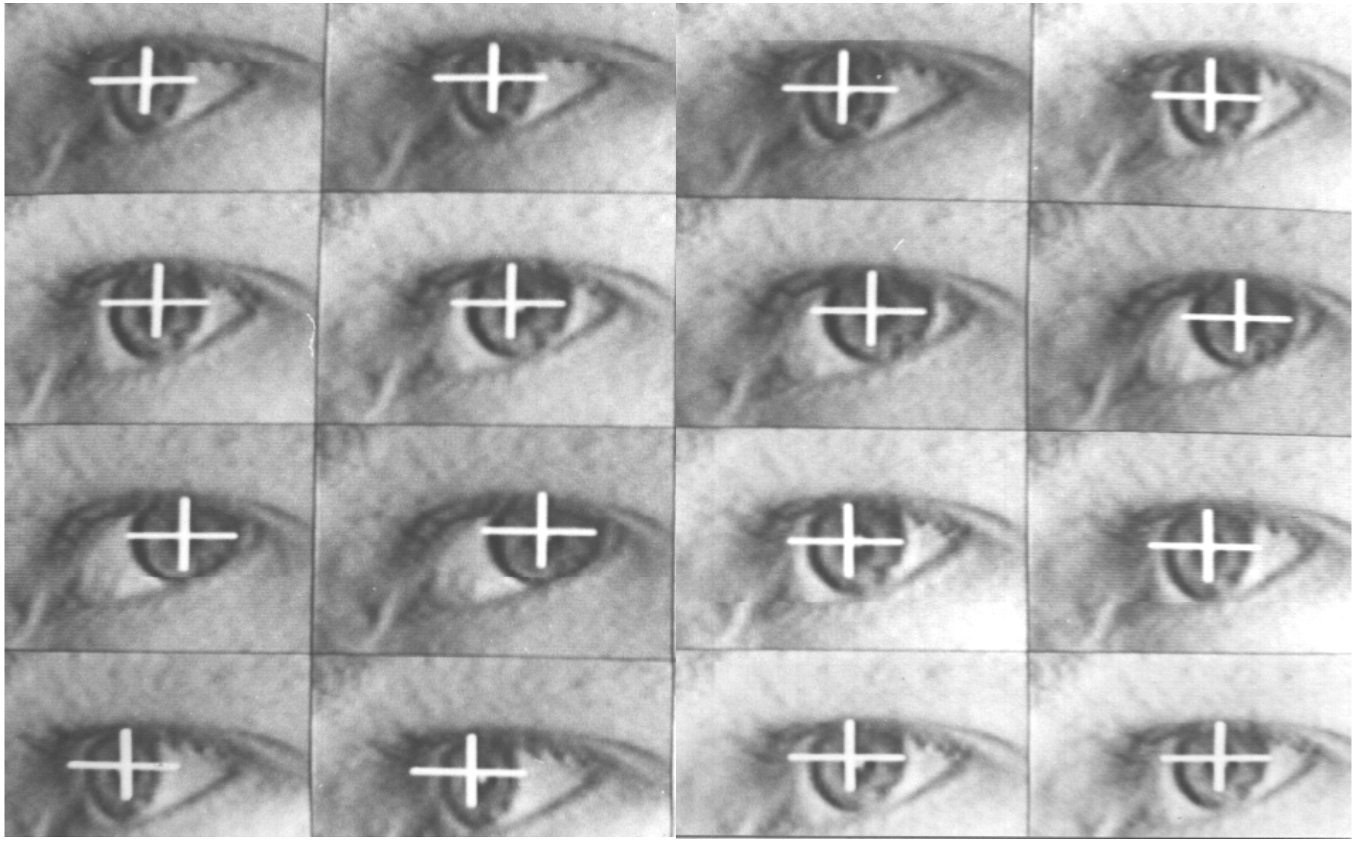
- Hence, the normalised cross-correlation may be scaled so that it lies in the range 0 to 1 by dividing it by the above expression

- Thus, the normalised cross-correlation may be redefined

$$N(m,n) = \frac{R(m,n)}{\left(\sqrt{\sum_i \sum_j g(i,j)^2}\sqrt{\sum_i \sum_j t(i-m,j-n)^2}\right)}$$

Template Matching

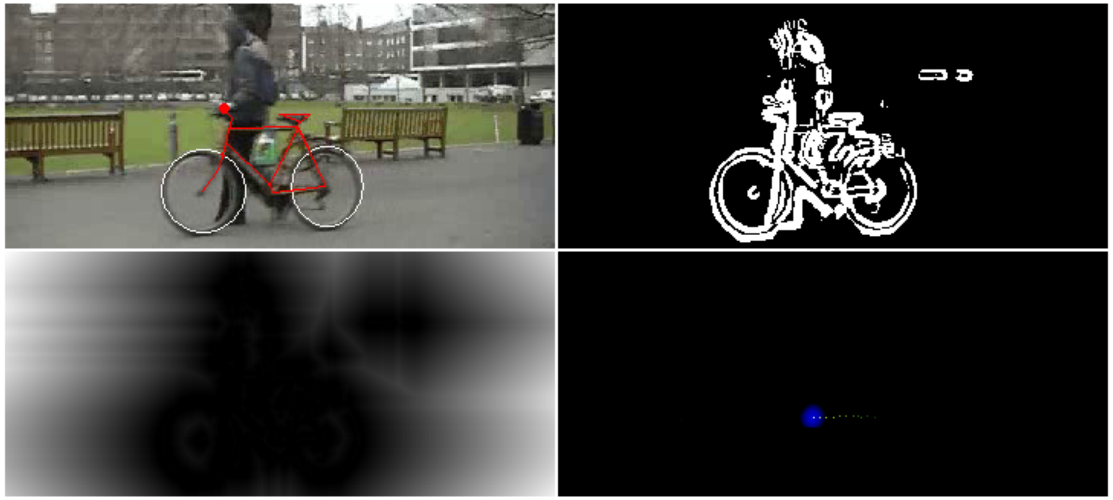
Eye-tracking using Normalised Cross-correlation



Template Matching

Chamfered Matching

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



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

Compute chamfered image for a binary edge image

- Image value = distance to closest edge pixel
- Also known as the distance transform
- cf. the medial axis transform

∞	∞	∞	∞	∞	0	∞	∞
∞	∞	∞	∞	∞	0	∞	∞
∞	∞	∞	0	0	0	0	∞
∞	∞	∞	0	0	∞	0	∞
∞	∞	∞	0	∞	∞	0	∞
∞	∞	∞	0	∞	∞	0	∞
∞	∞	∞	0	0	0	0	∞
∞	∞	∞	∞	∞	∞	∞	∞

Object pixels (zeros)

∞	∞	∞	∞	∞	0	1	2
∞	∞	∞	∞	1.4	0	1	1.4
∞	∞	∞	0	0	0	0	1
∞	∞	∞	0	0	1	0	1
∞	∞	∞	0	1	1.4	0	1
∞	∞	∞	0	1	1.4	0	1
∞	∞	∞	0	0	0	0	1
∞	∞	∞	1	1	1	1	1.4

After the first stage of processing

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

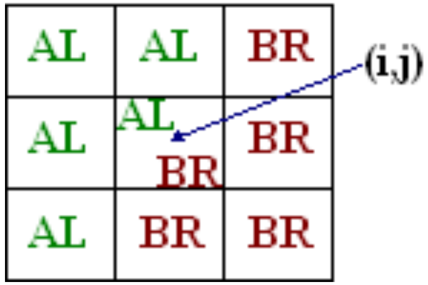
Template Matching

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 is AL (|(i, j), q)| , f( q ))

for j = max to min
    for i = max to min
       c(i, j) = min q is BL (|(i, j), q| , f( q ))
```



∞	∞	∞	∞	∞	0	∞	∞
∞	∞	∞	∞	∞	0	∞	∞
∞	∞	∞	0	0	0	0	∞
∞	∞	∞	0	0	∞	0	∞
∞	∞	∞	0	∞	∞	0	∞
∞	∞	∞	0	∞	∞	0	∞
∞	∞	∞	0	0	0	0	∞
∞	∞	∞	∞	∞	∞	∞	∞

Object pixels (zeros)

∞	∞	∞	∞	∞	0	1	2
∞	∞	∞	∞	1.4	0	1	1.4
∞	∞	∞	0	0	0	0	1
∞	∞	∞	0	0	1	0	1
∞	∞	∞	0	1	1.4	0	1
∞	∞	∞	0	1	1.4	0	1
∞	∞	∞	0	0	0	0	1
∞	∞	∞	1	1	1	1	1.4

After the first stage of processing

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

The chamfer matching technique :

- Generate a binary model image
- Compute the chamfer image / distance transform
- Compute the match image:
  - translate the model. to every possible position in the image
  - at each position, compute the sum of chamfer values at locations where the model is non-zero
- Find local minima in match image
- If the minimum is small enough then the object can be assumed to be present

Template Matching

The chamfer matching technique

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

1	1	1	1
1			1
1			1
1			1
1	1	1	1

Template

27.4	19.6	12.8	8	11.4
23.2	16.8	10.4	5	10.4
21	14	8	0	7
23.2	16.8	10.4	5	11.4

Matching Space

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

The chamfer matching technique

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

1	1	1	1
1			1
1			1
1			1
1	1	1	1

Template

27.4	19.6	12.8	8	11.4
23.2	16.8	10.4	5	10.4
21	14	8	0	7
23.2	16.8	10.4	5	11.4

Matching Space

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

The chamfer matching technique

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

1	1	1	1
1			1
1			1
1			1
1	1	1	1

Template

27.4	19.6	12.8	8	11.4
23.2	16.8	10.4	5	10.4
21	14	8	0	7
23.2	16.8	10.4	5	11.4

Matching Space

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

The chamfer matching technique

3.8	2.8	2.4	2	1	0	1	2
3.4	2.4	1.4	1	1	0	1	1.4
3	2	1	0	0	0	0	1
3	2	1	0	0	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	1	1	0	1
3	2	1	0	0	0	0	1
3.4	2.4	1.4	1	1	1	1	1.4

Chamfer Image

1	1	1	1
1			1
1			1
1			1
1	1	1	1

Template

27.4	19.6	12.8	8	11.4
23.2	16.8	10.4	5	10.4
21	14	8	0	7
23.2	16.8	10.4	5	11.4

Matching Space

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

The chamfer matching technique

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Template Matching

- Maxima and Minima Detection
  - Depends on the distance metric
    - maxima: normalized cross-correlation
    - minima: chamfer matching
  - Local maxima
    - Dilate, identify unchanged values, threshold
  - Local minima
    - Erode, identify unchanged values, threshold

Template Matching

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

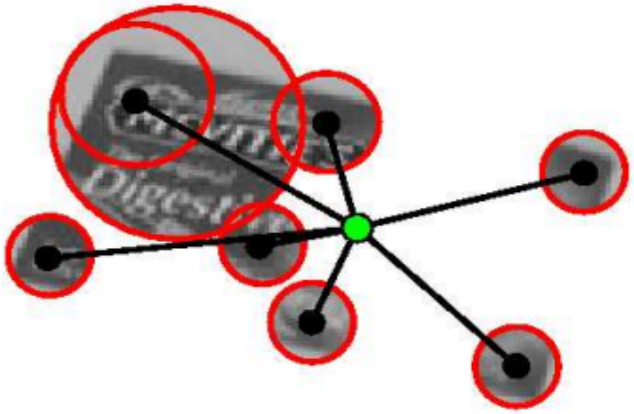
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Local Template Matching

- One of the problems of template matching is that each template represents the object or part of it as we expect to find it in the image
  - No cognisance is taken of variations in scale or in orientation
- If the expected orientation can vary, then we will require a separate template for each orientation and each one must be matched with the image
- Thus template matching can become computationally expensive, especially if the templates are large

Local Template Matching

- Use much smaller local templates to detect salient features in the image which characterise the object we are looking for
- The spatial relationship between occurrences of these features are then analysed
- We can infer the presence of the object if valid distances between these features occur



- The SIFT descriptor can also be used as the local template (more on the use of SIFT for local template matching later)

Demos

The following code is taken from the templateMatching project in the lectures directory of the ACV repository

See:

```
templateMatching.h
templateMatchingImplementation.cpp
templateMatchingApplication.cpp
```

```
/*
  Example use of openCV to perform template matching with normalized cross-correlation
  ---------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  7 June 2017
*/

#include "templateMatching.h"

/*
 * function templateMatching
 * Trackbar callback - threshold user input
*/

void templateMatching(int, void*) {

   extern Mat   inputImage;
   extern Mat   templateImage;
   extern int   thresholdValue;
   extern char* input_window_name;
   extern char* correlation_window_name;
   extern char* maxima_window_name;
   extern char* template_window_name;
   Mat          outputImage;

   outputImage = inputImage.clone();
```

```
/*****************************************************************************************************/
/*
 * The following is derived from code provided as part of "A Practical Introduction to Computer Vision with OpenCV"
 * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
 */

   Mat correlation_image;
   double min_correlation, max_correlation;
   Mat matched_template_map;
   int result_columns =  inputImage.cols - templateImage.cols + 1;
   int result_rows = inputImage.rows - templateImage.rows + 1;

   printf("%d %d, %d %d, %d %d\n",inputImage.cols,inputImage.rows, templateImage.cols, templateImage.rows, result_columns,result_rows);

   correlation_image.create( result_columns, result_rows, CV_32FC1 );

   matchTemplate( inputImage, templateImage, correlation_image, CV_TM_CCORR_NORMED ); //CV_TM_CCORR, CV_TM_SQDIFF, CV_TM_SQDIFF_NORMED

   minMaxLoc( correlation_image, &min_correlation, &max_correlation );

   FindLocalMaxima( correlation_image, matched_template_map, max_correlation *  ((double)thresholdValue/100) ); // DV thresholdValue/100 :
                                            cv::Mat matched_template_map
   Mat matched_template_display;
   cvtColor(matched_template_map, matched_template_display, CV_GRAY2BGR);

   Mat correlation_window = convert_32bit_image_for_display( correlation_image, 0.0 );

   DrawMatchingTemplateRectangles( outputImage, matched_template_map, templateImage, Scalar(0,0,255) );
/*****************************************************************************************************/


   imshow(template_window_name, templateImage);
   imshow(correlation_window_name, correlation_window);
   imshow(maxima_window_name, matched_template_display);
   imshow(input_window_name, outputImage);
}
```

Demos

The following code is taken from the chamferMatching project in the lectures directory of the ACV repository

See:

```
chamferMatching.h
chamferMatchingImplementation.cpp
chamferMatchingApplication.cpp
```

```
/*
  Example use of openCV to perform chamfer matching
  -------------------------------------------------
  Implementation file

  David Vernon
  10 June 2017
*/
#include "chamferMatching.h"

/*
 * function templateMatching
 * Trackbar callback - threshold user input
*/

void chamferMatching(int, void*) {
   extern Mat   model_image;
   extern Mat   background_image;
   extern Mat   foreground_image;
   extern int   thresholdValue;
   extern char* foreground_window_name;
   extern char* background_window_name;
   extern char* difference_window_name;
   extern char* model_window_name;
   extern char* model_edges_window_name;
   extern char* foreground_edges_window_name;
   extern char* chamfer_window_name;
   extern char* minima_window_name;
   extern char* match_window_name;
   Mat         outputImage;
   outputImage = foreground_image.clone();
```

```
/*****************************************************************************************/
/*
 * The following is derived from code provided as part of "A Practical Introduction to Computer Vision with OpenCV"
 * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014.  All rights reserved.
 */

   Mat model_gray,model_edges,model_edges2;

   cvtColor(model_image, model_gray, CV_BGR2GRAY);
   threshold(model_gray,model_edges,127,255,THRESH_BINARY);             // hardcoded literal value ... better to be input by user

   Mat result_image = foreground_image.clone();

   Mat difference_image, difference_gray, current_edges;
   absdiff(foreground_image,background_image,difference_image);
   cvtColor(difference_image, difference_gray, CV_BGR2GRAY);
   Canny(difference_image, current_edges, 100, 200, 3);                 // hardcoded literal value ... better to be be input by user

   vector<vector<Point> > results;
   vector<float> costs;
   threshold(model_gray,model_edges,127,255,THRESH_BINARY);
   Mat matching_image, chamfer_image, local_minima;
   threshold(current_edges,current_edges,127,255,THRESH_BINARY_INV);    // hardcoded literal value ... better to be input by user
   distanceTransform( current_edges, chamfer_image, CV_DIST_L2 , 3);
   ChamferMatching( chamfer_image, model_edges, matching_image );
   FindLocalMinima( matching_image, local_minima, thresholdValue);      // DV: replaced hardcoded literal value with input by user
   DrawMatchingTemplateRectangles( result_image, local_minima, model_edges, Scalar( 255, 0, 0 ) );
   Mat chamfer_display_image = convert_32bit_image_for_display( chamfer_image );
   Mat matching_display_image = convert_32bit_image_for_display( matching_image );
/*****************************************************************************************/

   imshow(foreground_window_name, result_image);
   imshow(background_window_name, background_image);
   imshow(difference_window_name, difference_image);
   imshow(model_window_name, model_image);
   imshow(model_edges_window_name, model_edges);
   imshow(foreground_edges_window_name, current_edges);
   imshow(chamfer_window_name, chamfer_display_image);
   imshow(minima_window_name, local_minima);
   imshow(match_window_name, matching_display_image);
}
```

Reading

D. Vernon, *Machine Vision: Automated Visual Inspection and Robot Vision*, Prentice-Hall, 1991.

Section 6.2 Template matching