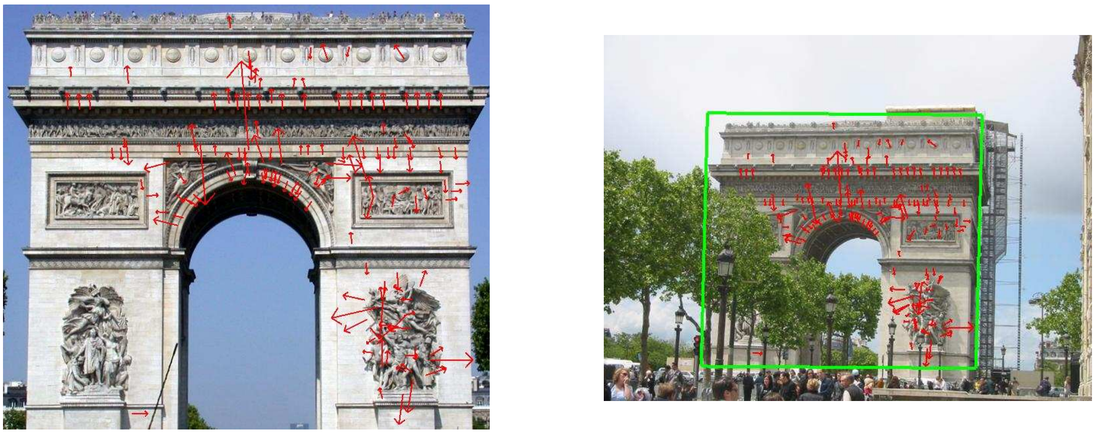
University of Cambridge
Engineering Part IB
Paper 8 Information Engineering

Part B: Image Features and Matching

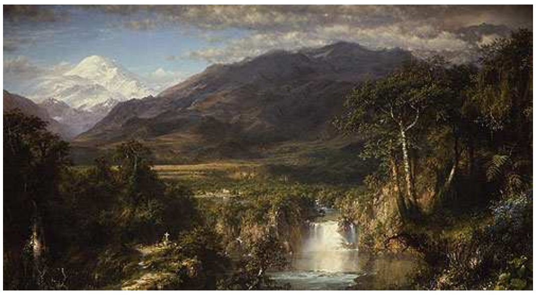
Handout 1: Introduction

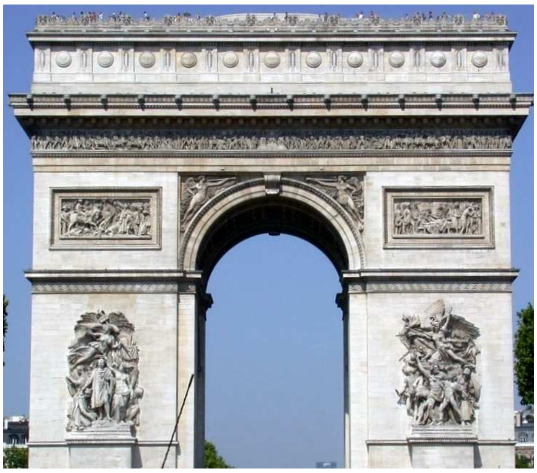


Roberto Cipolla
May 2013

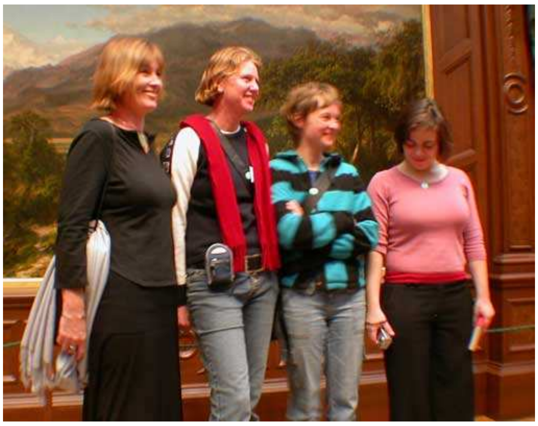
Image Matching and Retrieval

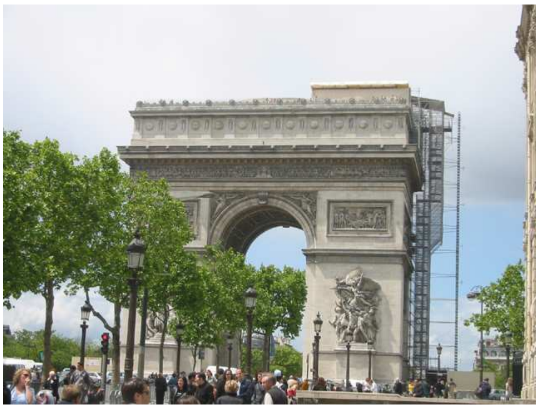
How do we build a system that can store thousands of reference images like the ones shown below





and then given a queries like this





correctly identify the object from the database and locate it in the photograph?

How can we build a system which can return an answer in less than a second?

The Basics

What ingredients are needed to achieve this result?

- A way of keeping track of what the system has seen
- A way to get the information out for comparison with queries
- A method to compare queries to the known objects

Data Reduction

How does the system keep track of what objects it has seen? Can it simply store every pixel of the reference images? Well, it will be apparent after a quick back-of-the-envelope calculation. How many bytes of storage are required to store the image data for all the works of art owned by the Louvre Museum?

✏ 1,000,000 images ×
$1000 \times 1000 = 1,000,000$ pixels per painting ×
3 bytes per pixel =
$3,000,000,000,000$ bytes $\approx$ 3 Terabytes

What do we keep?

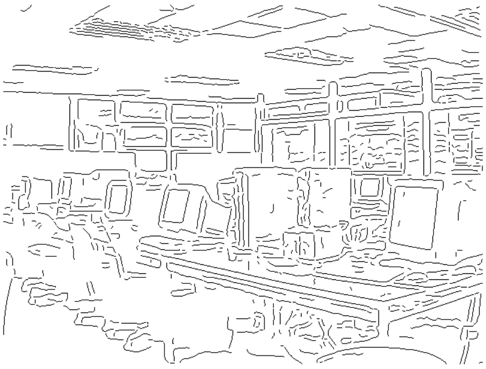
The system can't store every pixel, so what do we keep? Well, what kinds of problems exist in the queries that the system will need to deal with?

- Unreliable colour
- Different lighting
- Occlusion
- Scale change
- Orientation change
- Perspective distortion

How do we reduce an image but still deal with these problems?

Feature Extraction - Interest Points

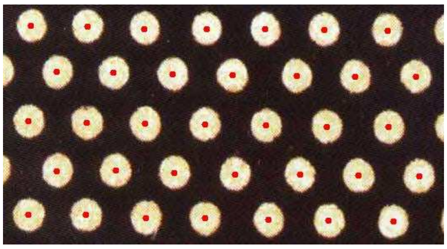
In handout 2 we look at feature extraction. The aim is to reduce the data content of the images while preserving the useful information they contain.



The most commonly used features are **edges**, which are detected along 1-dimensional intensity discontinuities in the image. Automatic edge detection algorithms produce something resembling a line drawing of the scene.

Corner or interest point detection is also common. Corner features are distinctive features that can be localised in the image and are particularly useful for motion analysis.

Blob detection focuses on finding isolated areas of uniform intensities in the image. Due to the inherent scale of these features they are useful for matching images across scales.



Feature Descriptors

We have a way of finding interesting points in images, but how do we describe those points to the system? There needs to be a way of taking the neighborhood of the interest point (the pixels around it) and turning it into an n-tuple/vector. This transformation should have a few properties:

Robust: Changes in the following should have little effect:

- Contrast
- Intensity
- Colour
- Resolution

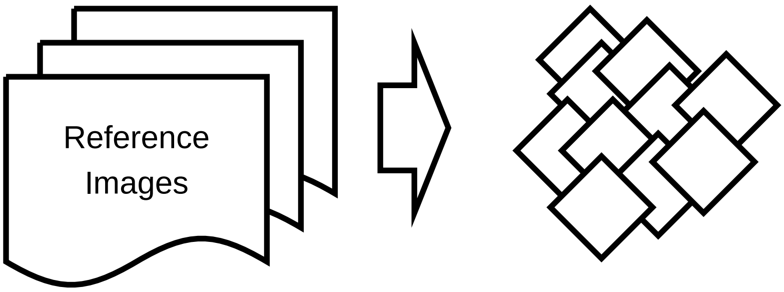
Ideally, small location and perspective changes should also have little effect.

Distinctive: Different-looking areas have different descriptors.

Some descriptors excel at one or the other of these properties, but we will concentrate on three which do quite well at all three in handout 3:

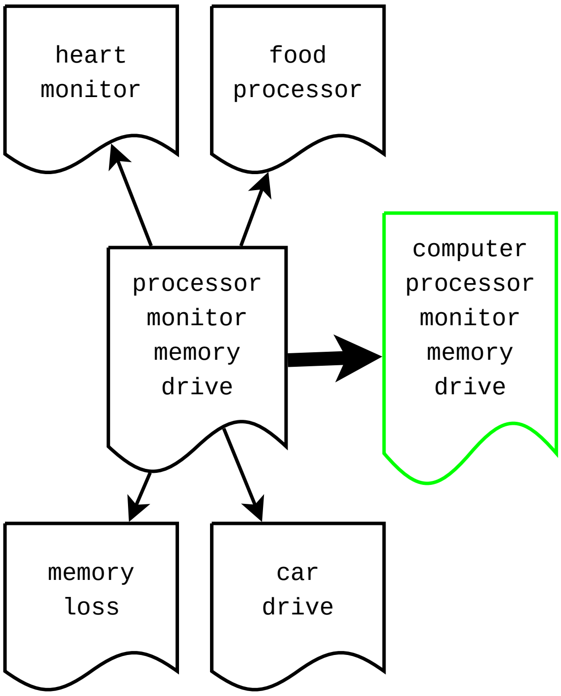
- Patches of normalized pixel intensities
- Output of a Filter Bank
- Orientation Histograms and SIFT

Searching Documents



So far we have discussed interest points and descriptors, but we haven't really touched upon how they will be used to retrieve images. For a moment, pretend that we are writing a search engine for text documents. The difference is that with this search engine you enter in an entire document that is like the one you want, and the engine goes through its database looking for matches.

How would this engine work? Well, it would pick out particular words from the document you gave it, probably a subset of distinctive nouns and verbs, and look for other documents which had those words. The document which had the most matching words would then be chosen.

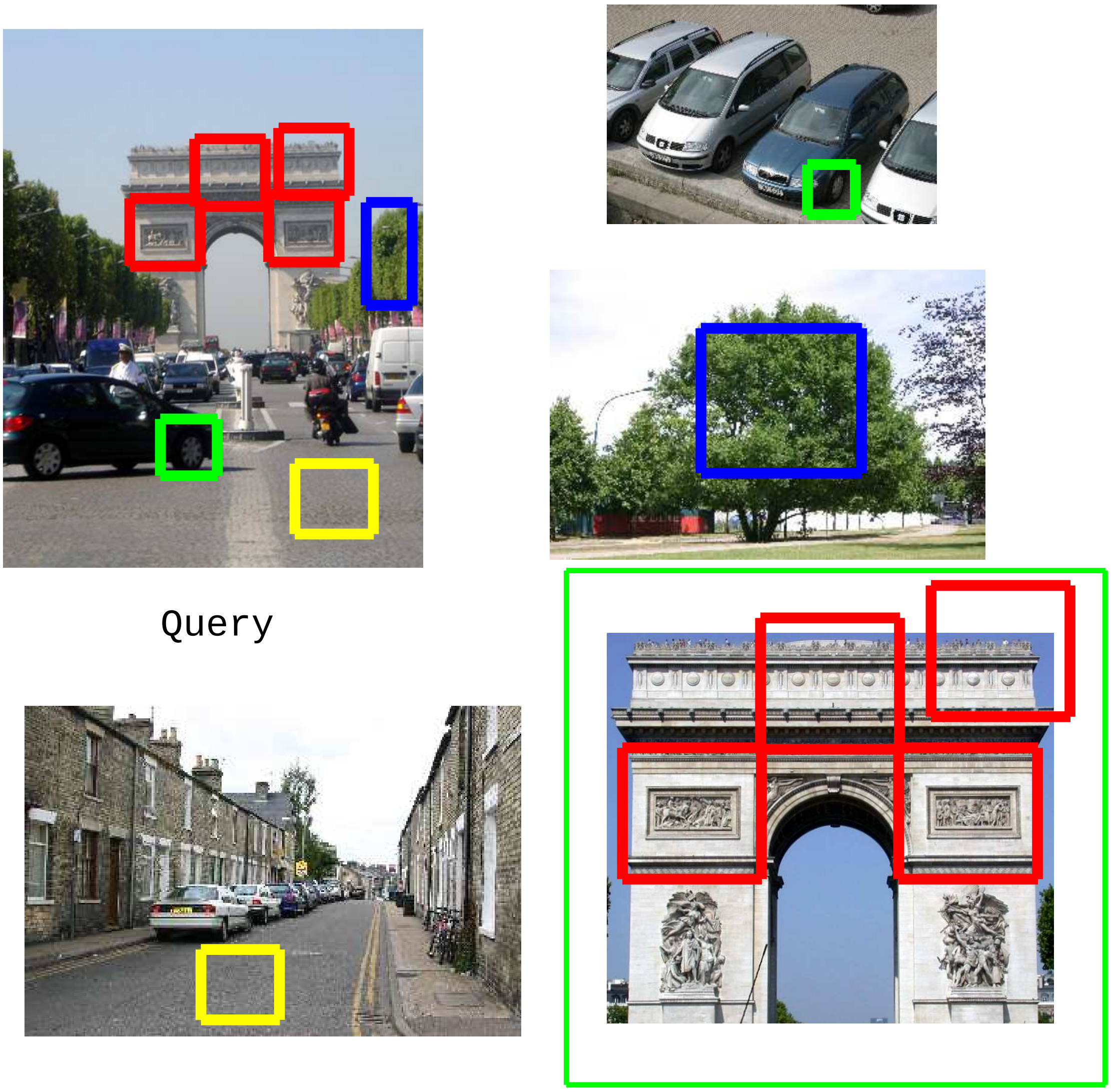


Matching words

Initially, it seems like matching words is simple. Aside from verbs, which have various conjugations, nouns should be easy to match. This isn't the case, unfortunately. Take, for example, different members of the same class. "dalmation" and "poodle" are very similar words, as they are both dogs, but don't look anything alike. Only by looking at their definitions can we match them properly. Thus, both the words from a document and their definitions can be used to find documents from the database which match.

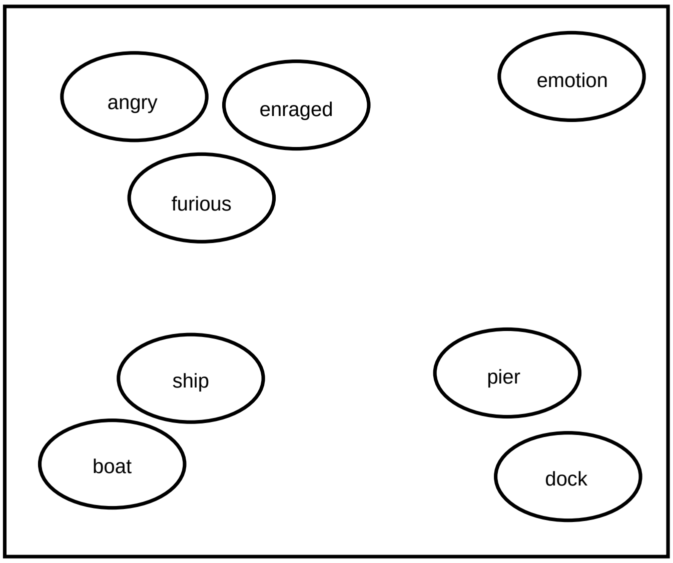
It ends up that this simple idea of matching documents based on the words they contain and their definitions carries over remarkably well into the problem of searching images. We are going to use the same technique to match images to images in a database. Interest points will be used as **visual words** with descriptors as their definitions. What remains is a need of a way to find the synonyms of a visual word.

Visual words



Nearest Neighbour Matching

Finding synonyms without a thesaurus is quite difficult. It requires scanning definitions of words for those with the same meaning. Imagine that instead of word-based definitions, though, each word was described by a unique point in a 2D plane, and that words that meant the same thing were near to each other.



Then, finding the best synonym would be a simple case of finding the point in the plane which is the **nearest neighbour** to the query word.

This is essentially the task that a feature descriptor transform is meant to perform, namely a transformation that takes the input data (an interest point) and outputs a vector which describes that data such that other, similar data members will be nearby.

Data Structures

"Nearby" is usually defined as a small distance away as measured by an Euclidean distance metric, measured using the following formula

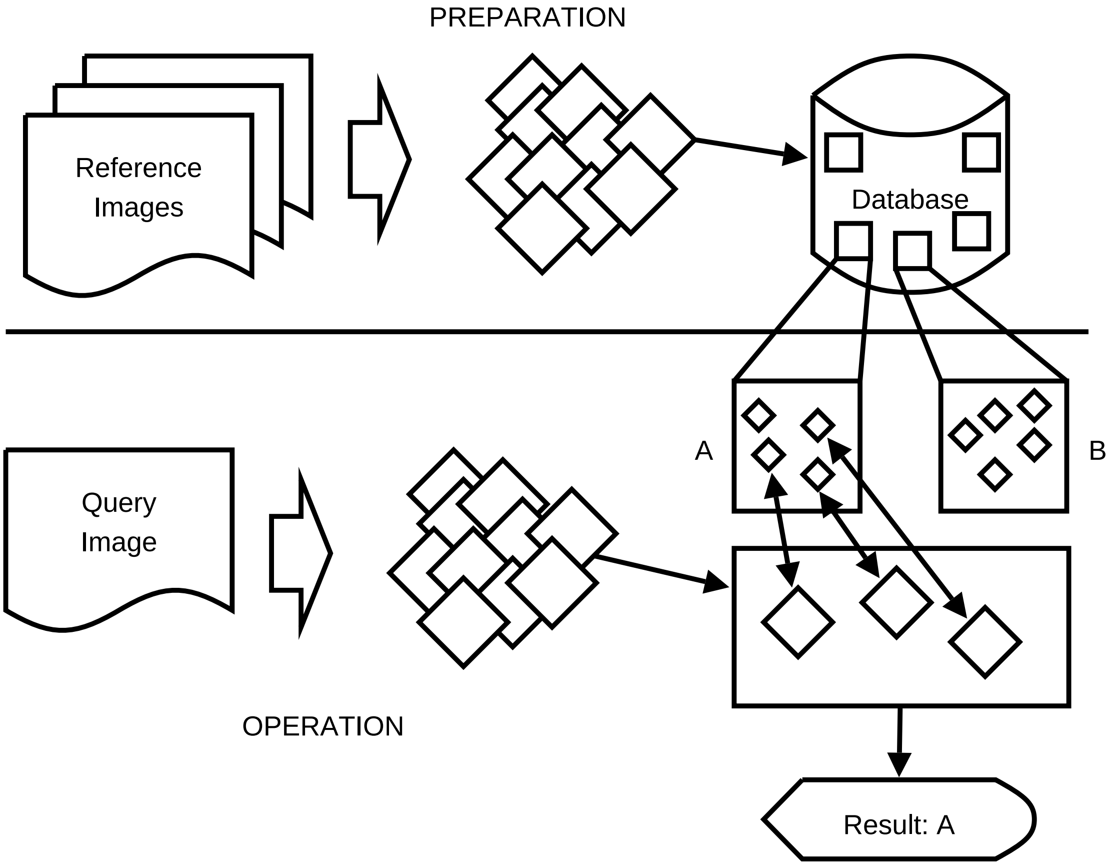
$$E(\vec{x}, \vec{y}) = \sqrt{\sum_d (\vec{x}_d - \vec{y}_d)^2}$$

So, one way of solving the problem is to search through all the feature points in the database images for the best match of a query feature. This is called a **linear search**, and is prohibitively expensive to compute. Instead, we must find a way of storing the data to make searching faster.

A data structure organizes data such that it is more efficient to store, access and search. The simplest data structure is a list of items, such as an array of numbers. The most complex are almost impossible to visualize. One solution is to use tree-based data structures to tackle the problem of nearest neighbour retrieval.

The Final System and Demo

The final system the course will develop is described below:


PREPARATION
Reference Images
Database
A
B
Query Image
OPERATION
Result: A