Recap of last lecture

Summary

- The Aperture Problem
- Cross-correlation (and its role in finding corners)
- The link between SSD and cross-correlation
- Scale invariance (and why its hard to achieve with corners)
- Blobs (and how to detect them with a LoG filter)
- Blobs and scales (and the importance of the scale-normalised LoG filter)
- Selecting the characteristic scale (and the role of scale space)
- Efficient scale space tricks (sparse scale sampling, incremental blurs, image pyramids, DoG)

VE Tag = Very Examinable

NE Tag = Non Examinable

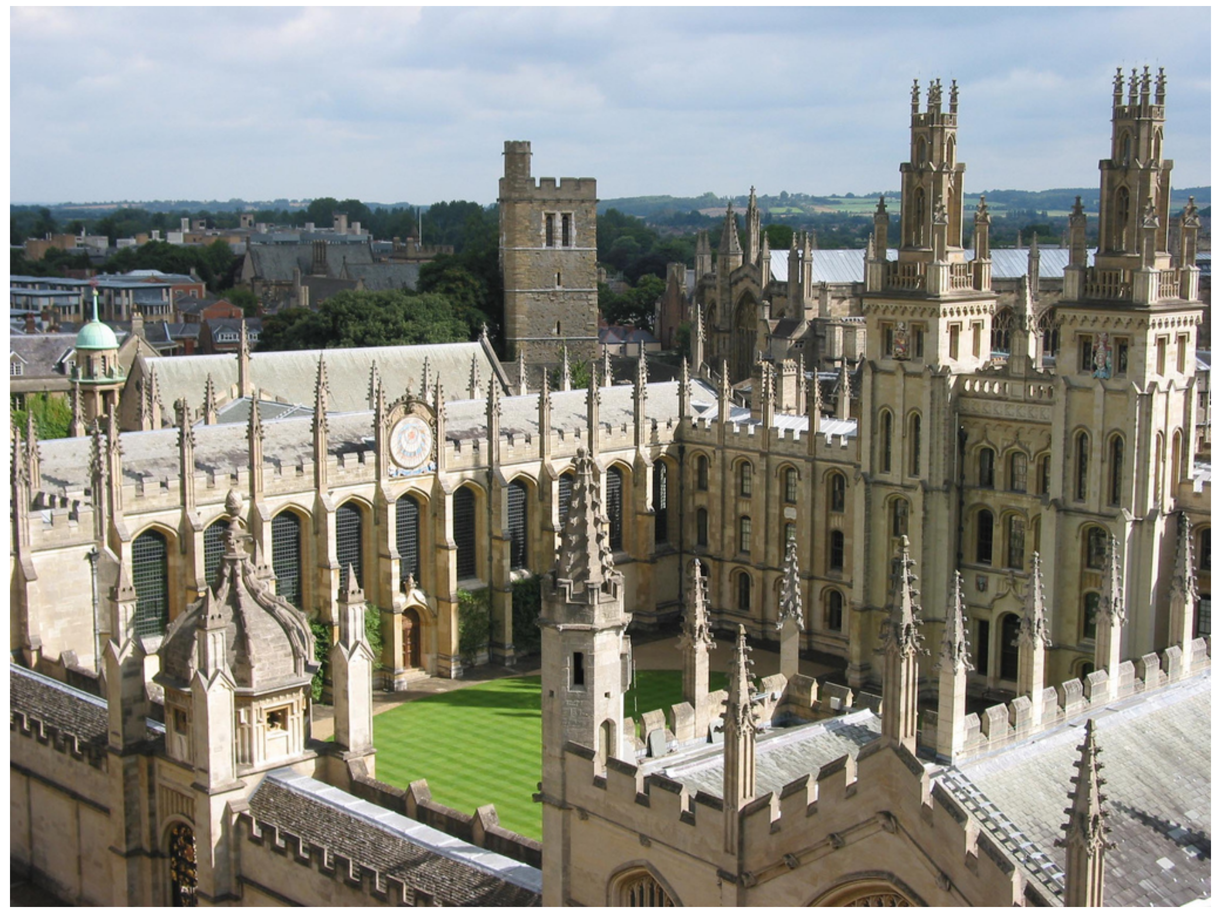
Matching and correspondence

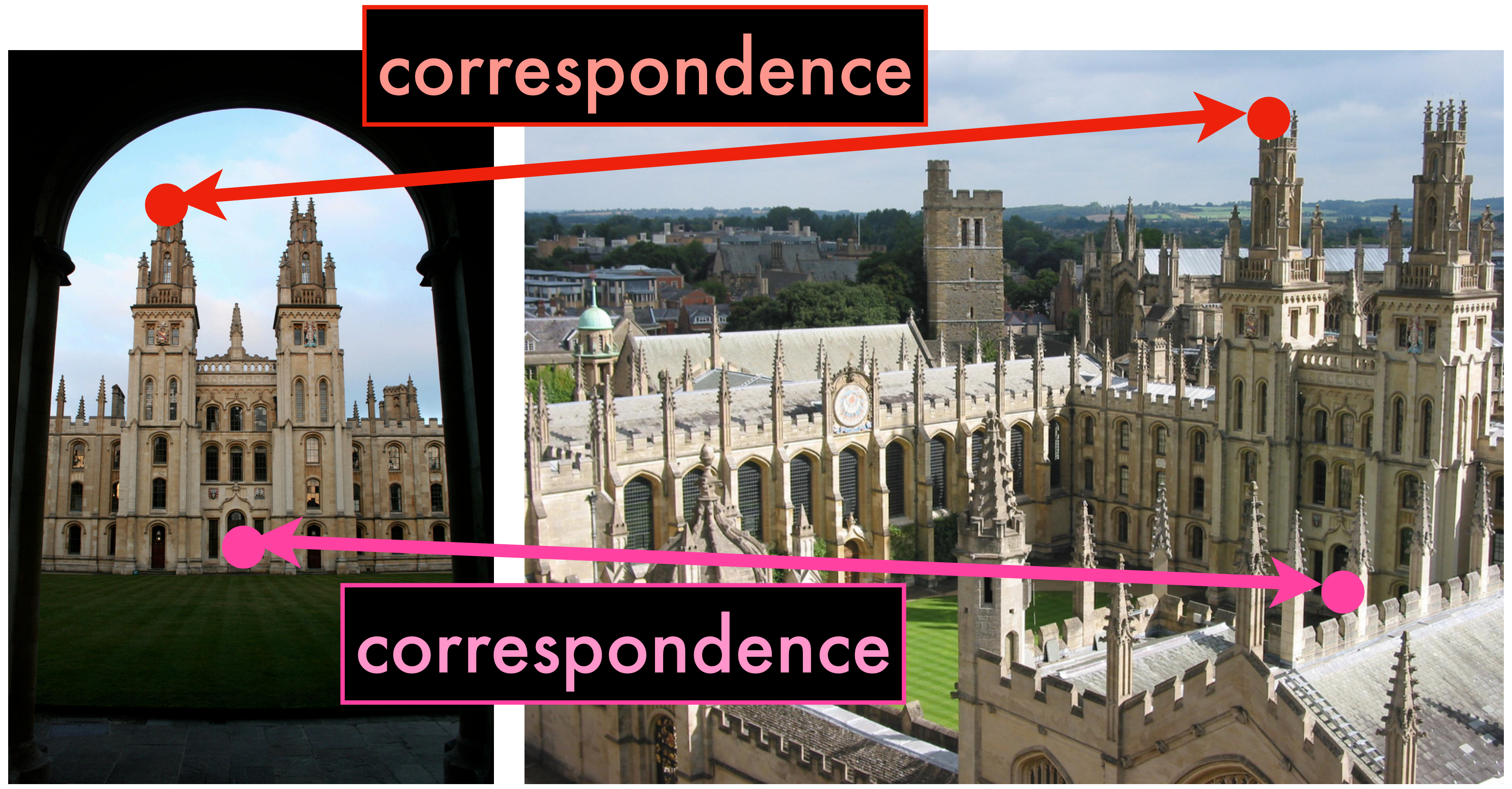
Feature Matching

The ability to match image structures (i.e. say whether regions from two images capture the same underlying object) is an important **primitive** in computer vision

Matching enables correspondence

Successful **matching** enables us to establish **correspondences** across views: to find pairs of regions for which "this bit" in one image matches "that bit" in another. The ability to find correspondences lies at the heart of computer vision, because it allows us to interpret the visual world.





Research trivia: It has been said[1] that when a grad student asked Prof. Takeo Kanade "What are the three most important problems in computer vision?", Kanade replied: "Correspondence, correspondence, correspondence!"

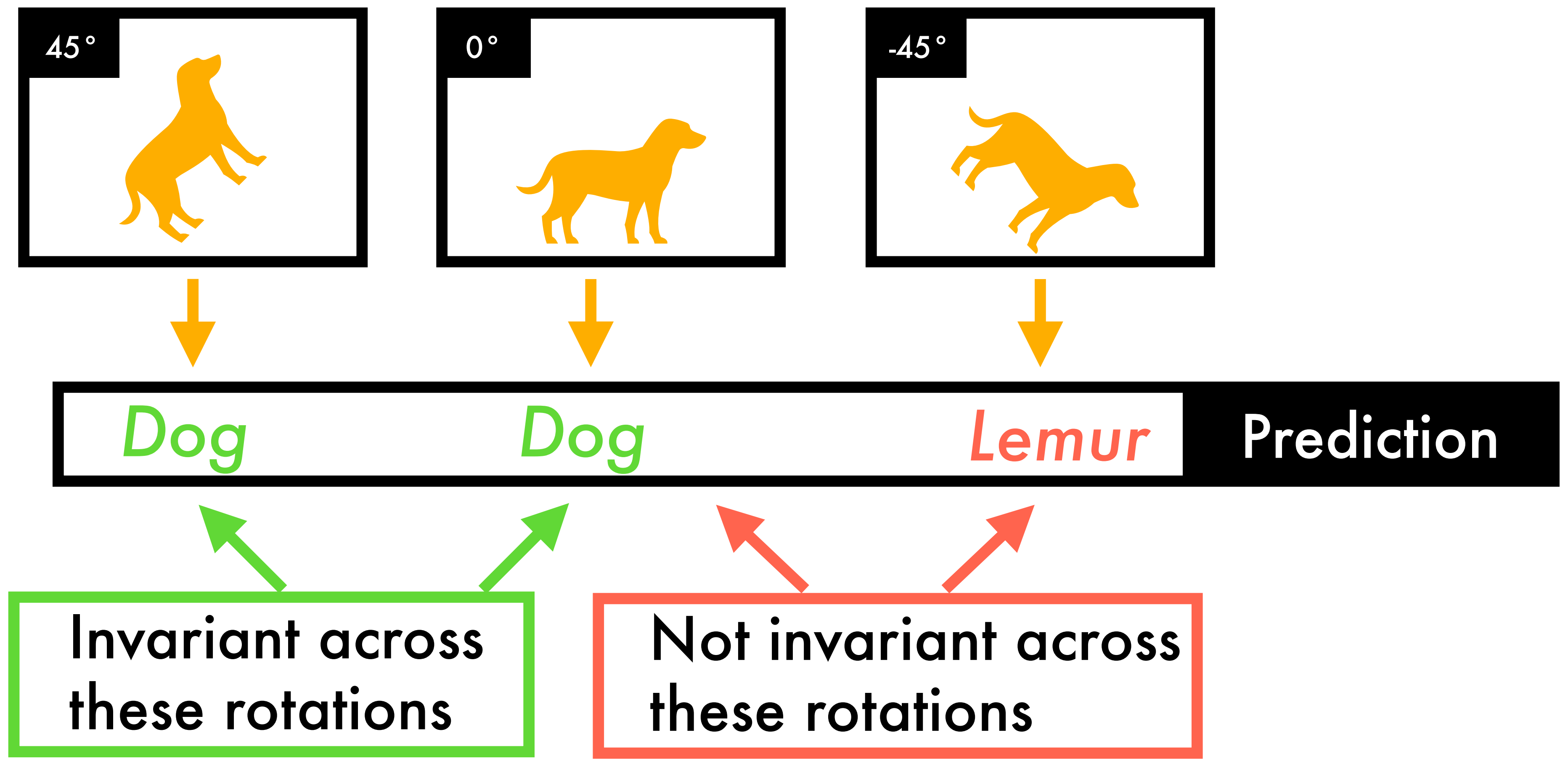
Images source: Philbin et al. "Object retrieval with large vocabularies and fast spatial matching." CVPR 2007

Reference: [1]Wang et al. "Learning correspondence from the cycle-consistency of time." CVPR, 2019

Invariances beyond scale

Rotation *invariance*

In addition to scale invariances, we often also want our vision systems to be invariant to rotation



Just as in the case of scale invariance, we can achieve rotation invariance if we can:

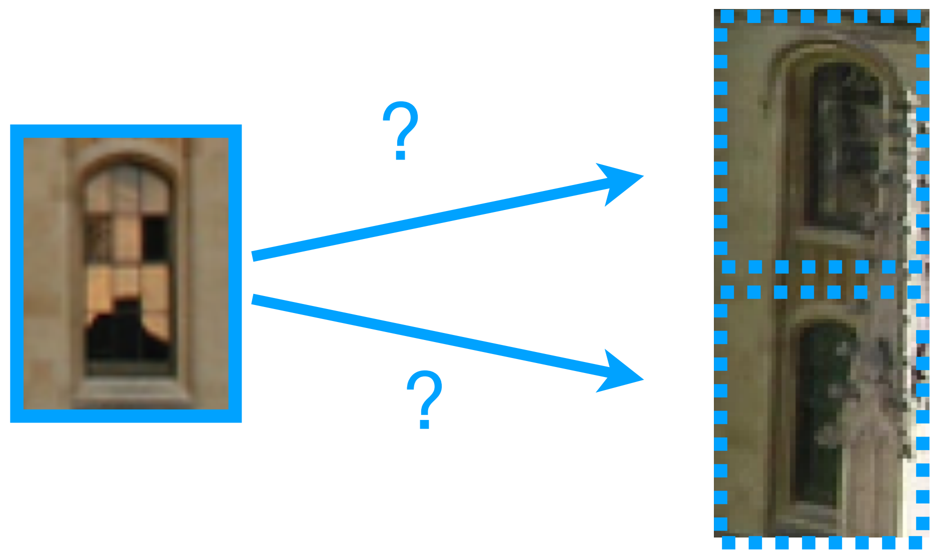
1. Accurately *estimate the rotation* of an object
2. Normalise (i.e. rotate) all objects to a common rotation

Further challenges for matching

Rotation and scale invariance are very useful, but they are not enough on their own, for two reasons

1. We need robustness via **additional invariances** to factors like partial occlusion, and changes in 3D viewpoint and illumination

2. *Too much invariance is bad* (a function that maps every patch to the zero vector is invariant to everything, but not useful)! We also need to create features that are **distinctive**



Which window from the second image matches the window in the first image?

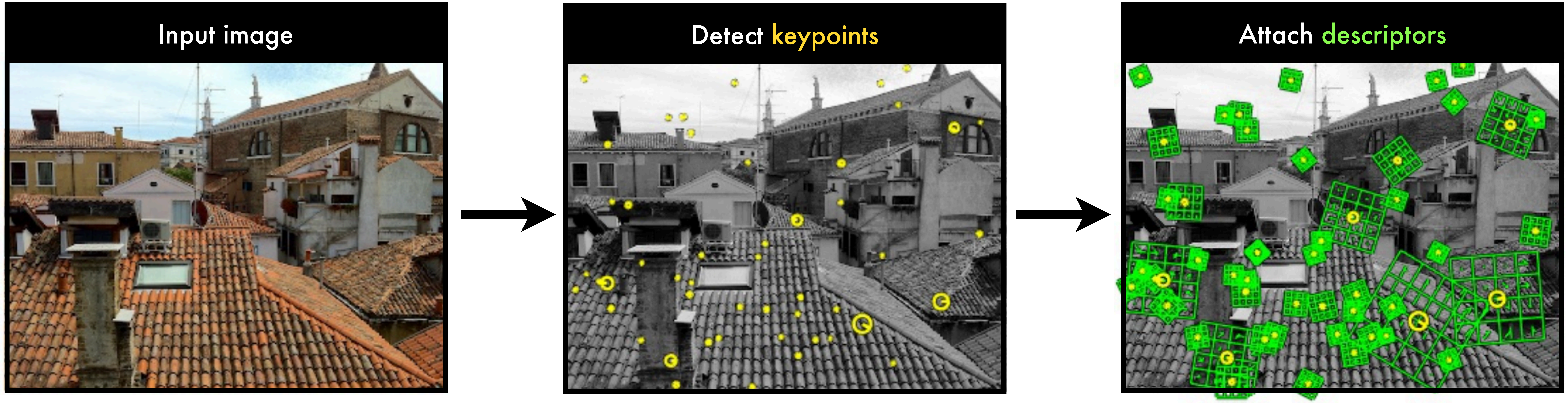
We need distinctive features to know the answer.

There is a natural tension between invariance and distinctiveness

Strategy: use keypoints <u>and</u> descriptors

Achieving invariance and distinctiveness

We will use a two-pronged strategy to achieve invariance and distinctiveness for robust matching



- The <u>keypoints</u> enable us to estimate (and therefore normalised and achieve **invariance** to) scale and rotation
- The <u>descriptors</u> enhance **distinctivenss**, while supporting *partial invariance* to changes in 3D view, occlusion and illumination

Keypoints help us efficiently select the subset of points that are "most interesting" to describe

Images credit: Venice "Roof", Vedaldi, SIFT tutorial

Descriptor: intensity patches

Using raw pixel intensity patches as descriptors

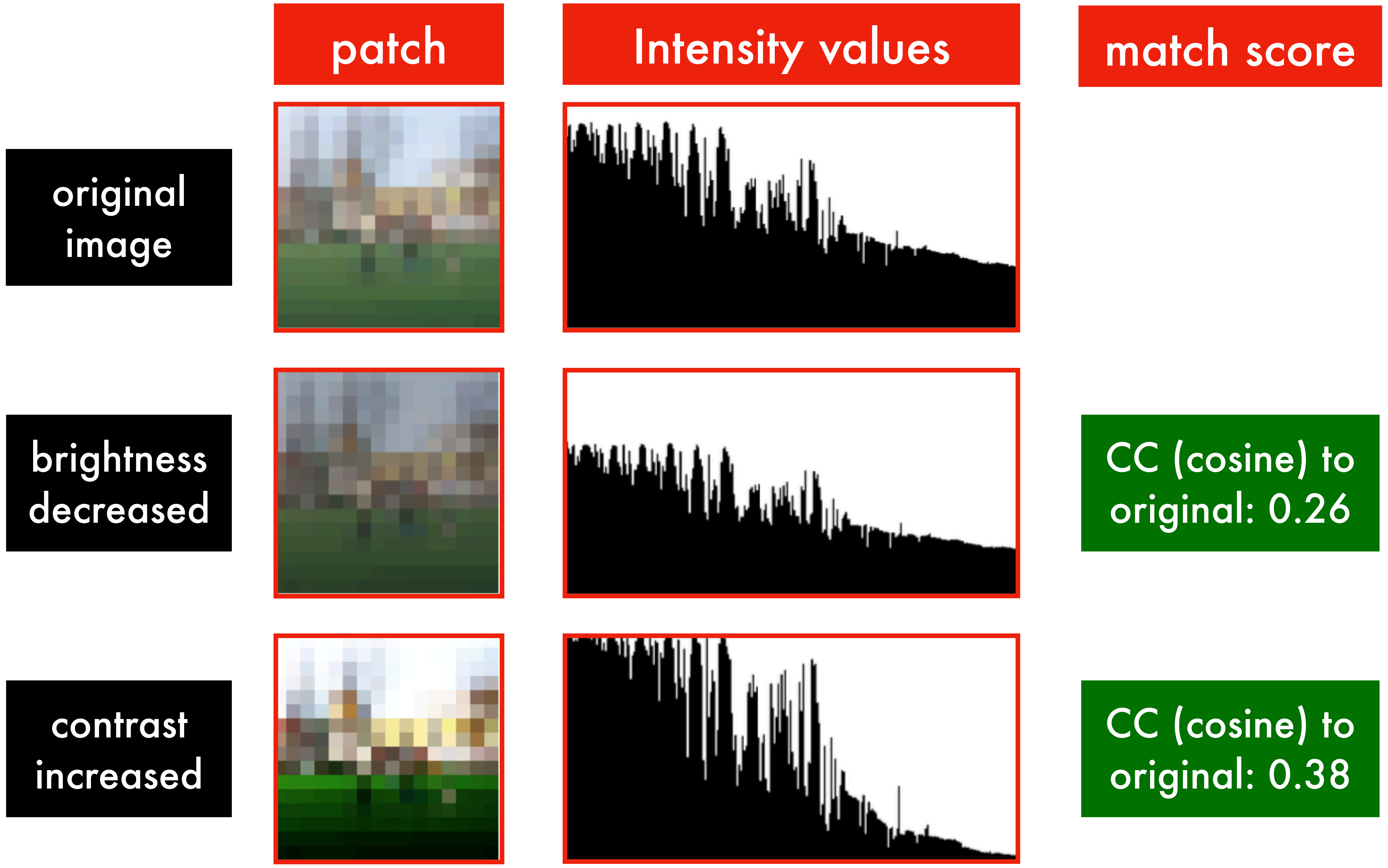
The simplest way to "describe" a patch of N pixels in an image is just to store the N intensity values, $P[i]$

You can then compare patches directly using (unnormalised) cross-correlation (CC) to find a match:

$$CC(P_1, P_2) = \sum_{i=1}^{N} P_1[i]P_2[i]$$

Problem: this raw form of description is not very robust to changes in lighting

The influence of colour changes



Unnormalised cross-correlation is sensitive to lighting changes, so raw intensity patches would not support robust matching under this similarity measure

Descriptor: Zero-Normalised intensity patches

Zero-Normalised Patches

Brightness changes are essentially changes in the **mean** brightness value. While the mean changes, the distribution of the intensity values around the mean stays the same.

By giving the intensity values a zero mean, they become relatively immune to brightness change:

$$\boxed{\mu = \frac{1}{N}\sum_{x,y} I(x,y)}$$

$$\boxed{Z(x,y) = I(x,y) - \mu}$$

However, the intensity values are still affected by contrast changes. A contrast change is essentially a change in the **variance** of the distribution of the intensity values around the mean.

To deal with contrast all that is required is to divide each value by the standard deviation of the intensity value distribution:

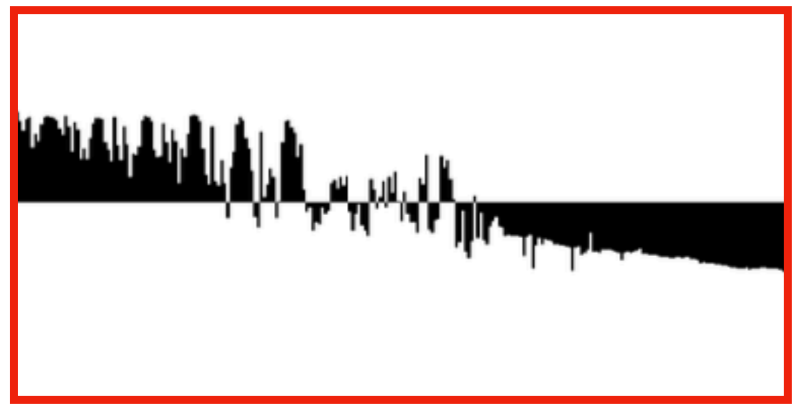
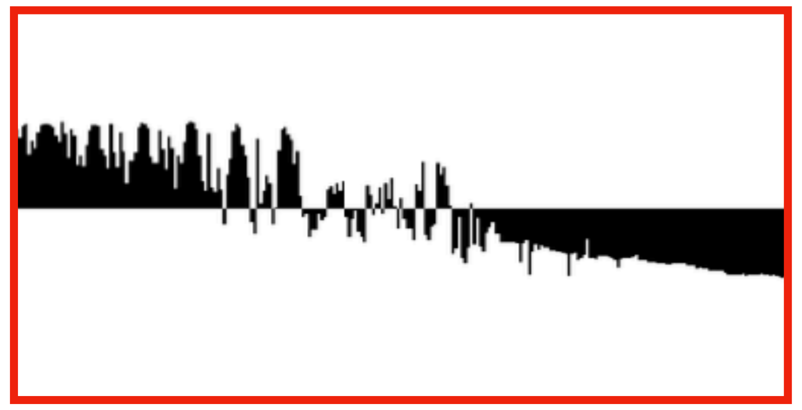
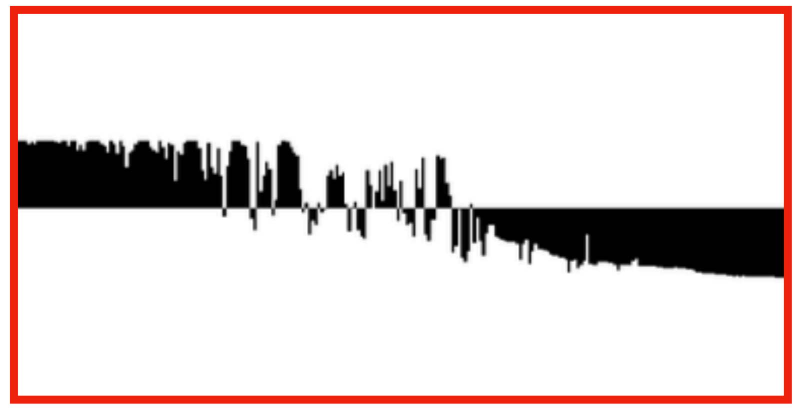
$$\boxed{\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2}$$

zero-normalised patch

$$ZN(x,y) = \frac{Z(x,y)}{\sigma}$$

The influence of colour changes on ZN patches

Zero-normalised patches often can be **accurately matched** using simple cross-correlation

	patch	ZN intensity values	match score
original image			
brightness decreased			CC with original image: 0.999..
contrast increased			CC with original image: 0.97

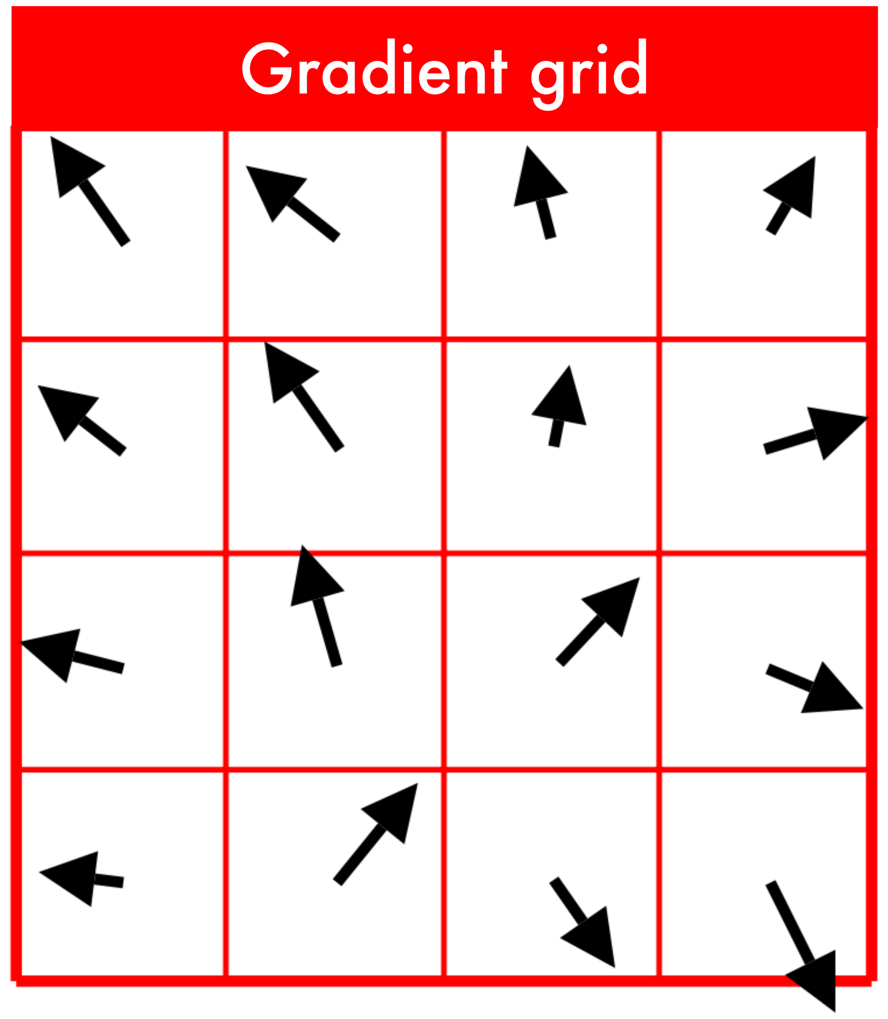
Note: The size of the descriptor grows with the size of the patch, and thus can be quite big. Even so, while not a data reduction, it is a useful way to represent these areas

Histogram of Oriented Gradients

What about gradients?

If you look at the gradient of each pixel in the patch, each will have its own distinct:

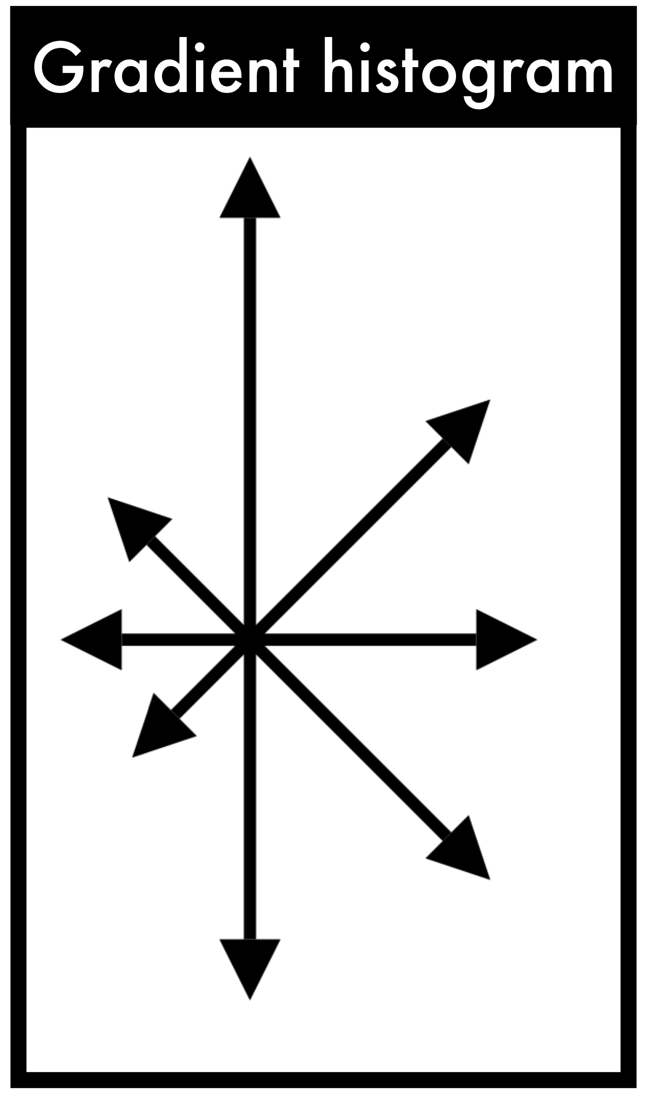
- orientation/direction, or way that it is facing
- size/strength (gradient magnitude)



Histograms as descriptors

The pixel gradients can be binned together into a "**histogram of oriented gradients**" **(HOG)**

This histogram is built using gradients/ edges (which are robust to contrast and brightness changes) and can be detected at different scales, and also incorporate discriminative orientation data



Length indicates bin value for each orientation

These properties makes the histogram a very strong candidate, both (1) as a **descriptor** and (2) for estimating the orientation of keypoints

Reference: Lowe, David G. "Distinctive image features from scale-invariant keypoints." IJCV (2004)

Dominant Orientations for rotation estimation

Fine-grained HOGs

We can find the dominant orientation by looking at the histogram of oriented gradients at the appropriate low-pass filtered image in the Image Pyramid

We can build a histogram (typically with 36 bins covering 360 degrees) of all of the edge orientations *weighted by their gradient magnitudes* in the neighbourhood of the keypoint

Note: this needs to be smoothed (low-pass filtered with a 2D gaussian of size 1.5σ scale for the keypoint)

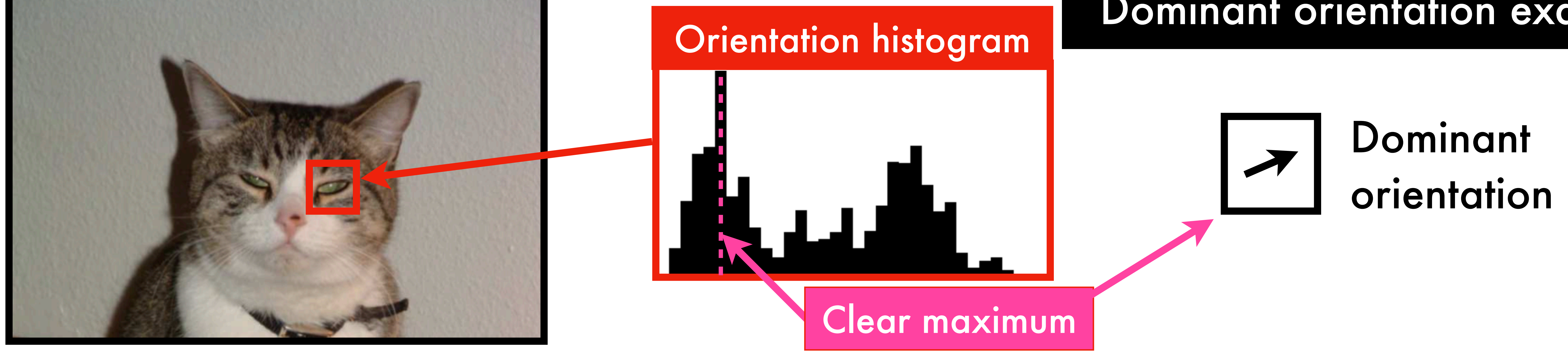
Finding the dominant orientation

The highest peak in the histogram[1] will approximate the **dominant orientation**. We can use this orientation to estimate the rotation of a **keypoint**, and then normalise to achieve rotation invariance.

Note: a better estimate can be found through *interpolation* (by fitting a parabola to the values of the bin and its two neighbours).

If there is *no clear maximum*, then the keypoint is given several dominant orientations (i.e. several copies of the keypoint with different orientations are used.)

Dominant orientation example



[1]In most implementations, a further smoothing step is applied to the bin values in the histogram to improve the robustness of peak finding (see e.g. https://www.vlfeat.org/api/sift.html)

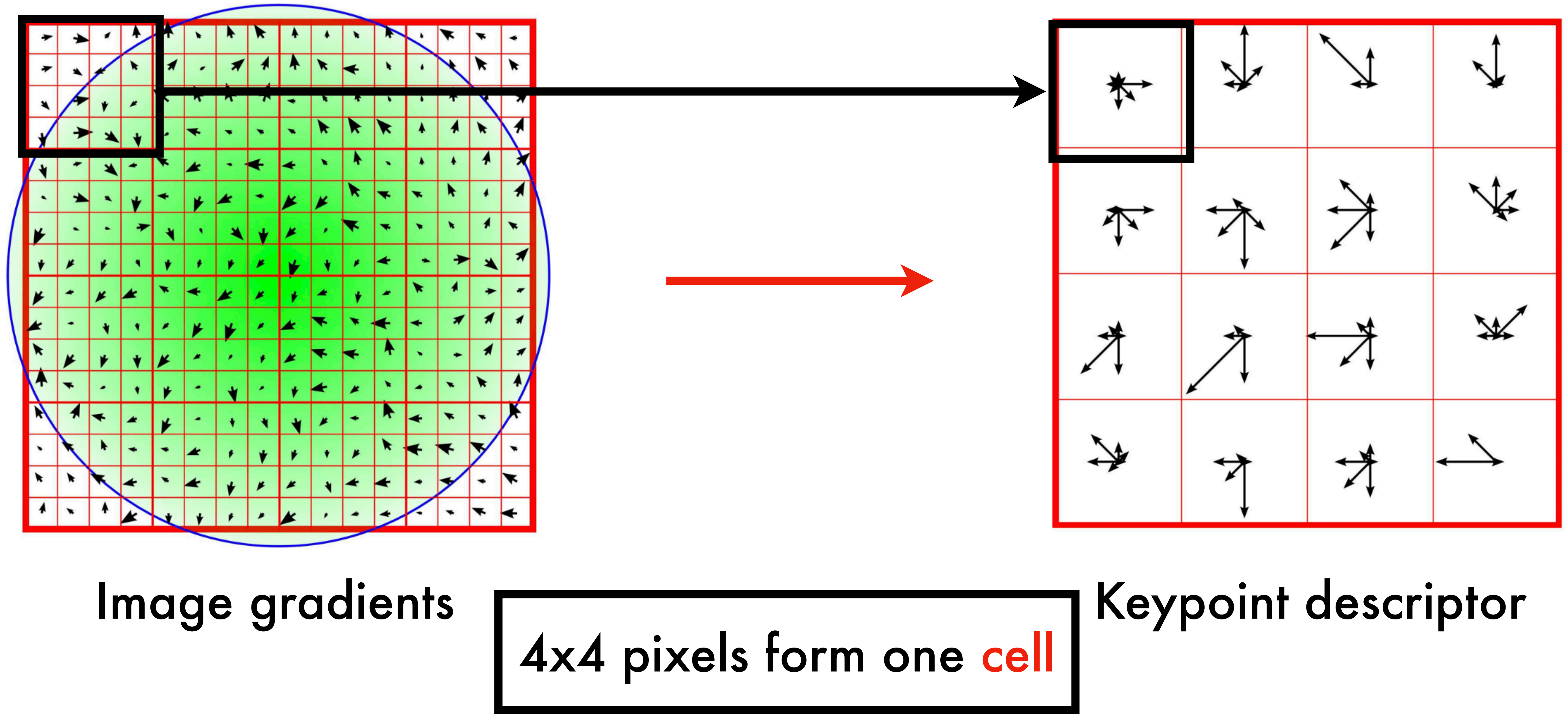
The SIFT keypoint descriptor

The SIFT keypoint descriptor

SIFT stands for **S**cale-**I**nvariant **F**eature **T**ransform. It uses a collection of orientation histograms to create a robust and descriptive representation of a patch

This $N \times N$ patch (typically, $N = 16$) is extracted at the scale of the keypoint, and its gradient orientations are stored **relative** to the dominant orientation of the keypoint, making the overall descriptor scale invariant and rotation invariant

SIFT details



The $N \times N$ patch is split into c cells (typically with N pixels in each cell) and the directions are binned into a histogram weighted by their magnitude and a Gaussian window with a σ of 0.5 times the scale of the keypoint at the centre of the patch

The Gaussian weights the inner pixels (those closer to the keypoint) to minimise the influence of partial occlusions

Reference: Lowe, David G. "Distinctive image features from scale-invariant keypoints." IJCV (2004)

The SIFT keypoint descriptor - more details

Further SIFT details

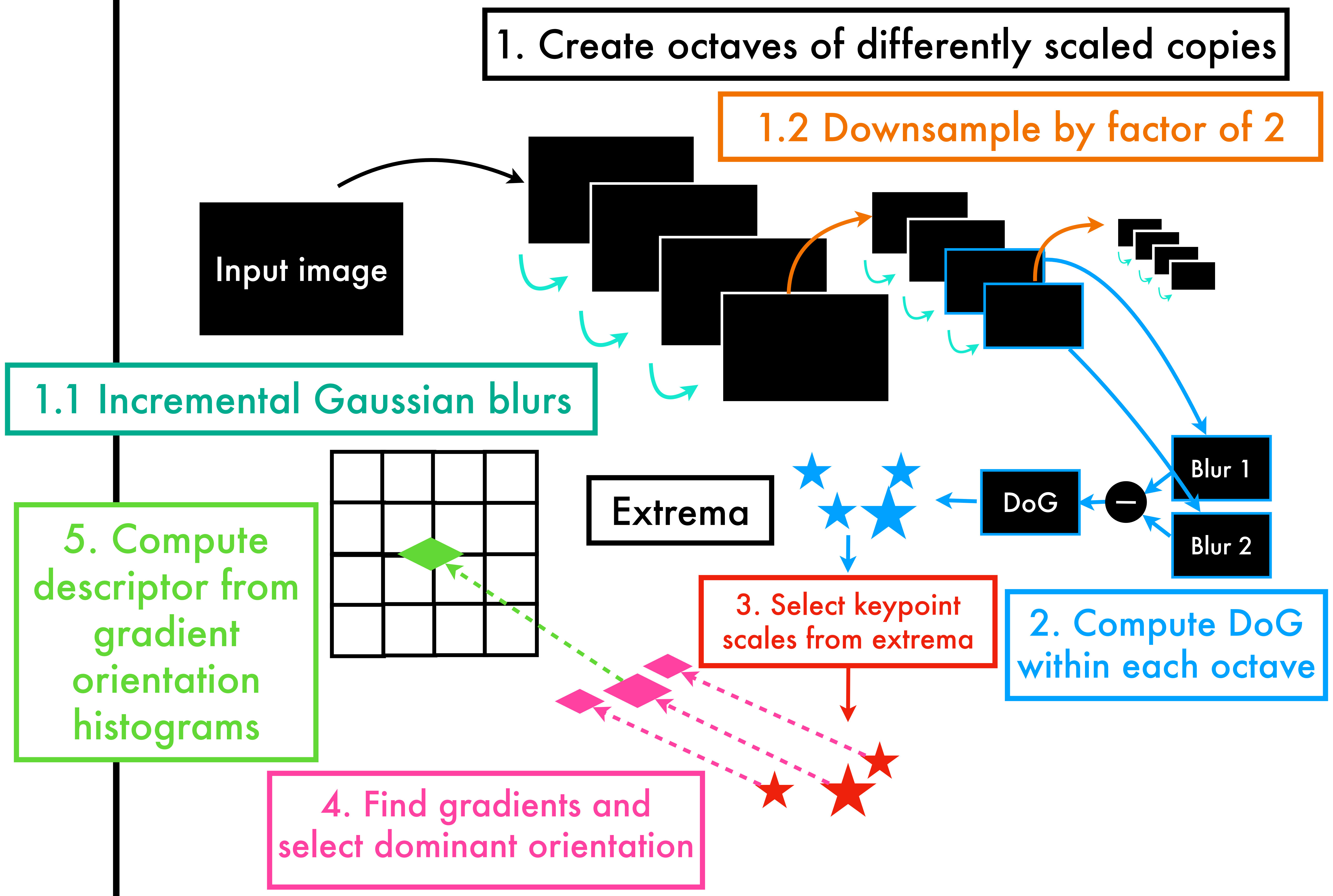
Descriptor size: If the bins are centred on d directions (typically 8) in each of c cells (typically 16), the resulting descriptor is a $d \times c$ vector (typically 128D).

Robustness: By dividing the patch into cells, a particular gradient can move around to some degree within the descriptor window and still contribute to the same directional histogram.

Normalisation: Once the $d \times c$ vector has been extracted, it is L2-normalised to provide invariance to gradient magnitude change.

Truncation: One final step is performed to help minimise the effects of non-affine lighting changes: the values are truncated so that all values in the unit vector are less than 0.2 (to reduce the effect of single elements such as those coming from very strong specular highlights) and then renormalising.

Keypoint and Descriptor Framework Overview



Framework overview

Finding Keypoint Scale and location

To produce a descriptor which is invariant to scale (and location) we first normalise for scale in the image (by sampling image intensities at the appropriate level of the image pyramid)

Finding Keypoint Orientation

We can then normalise for orientation (by sampling pixels after computing a characteristic reference/ dominant orientation for the patch of pixels using a histogram of gradients)

Computing the Feature Descriptor

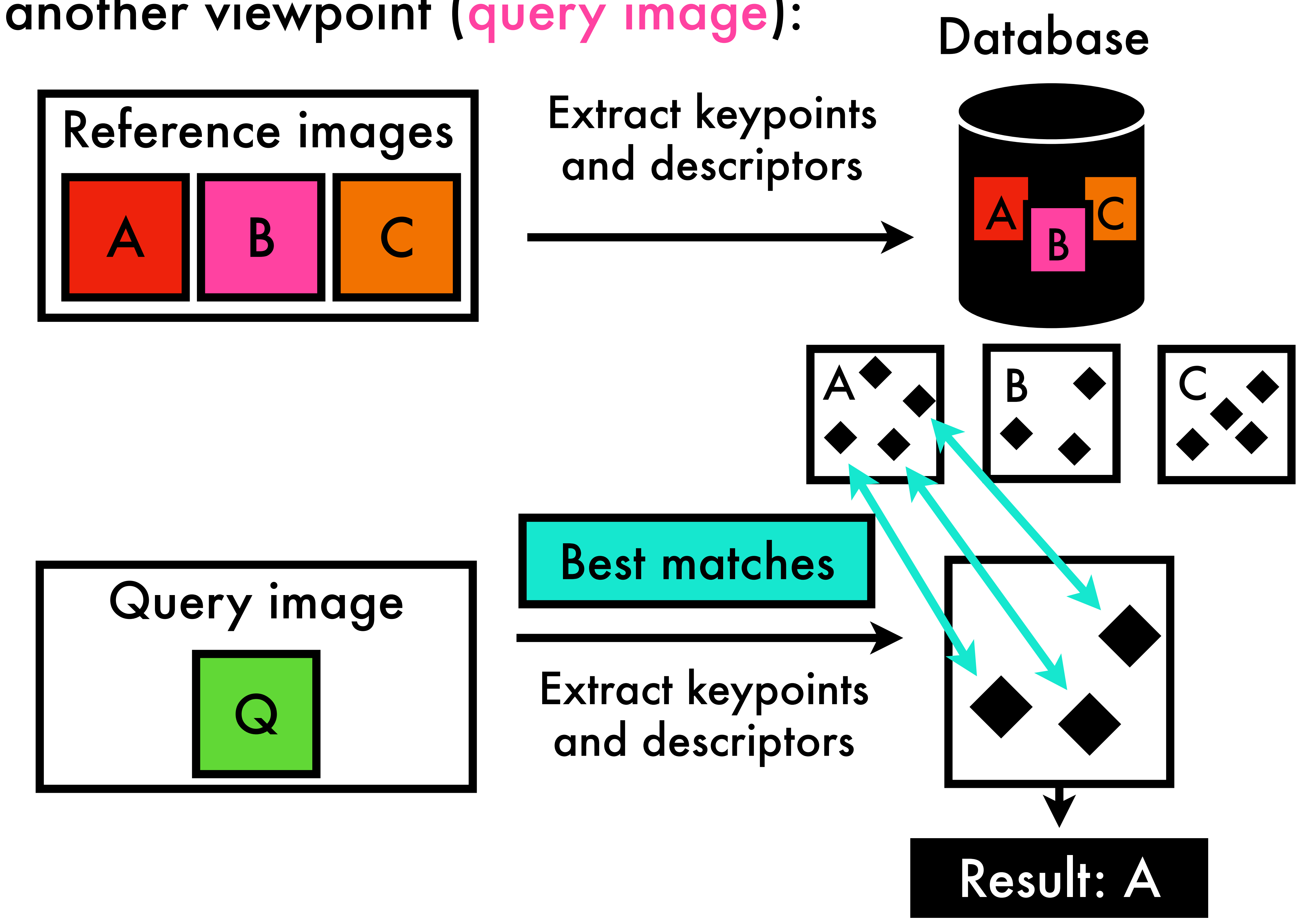
Finally, we can then compute an appropriate descriptor for each feature by either raw or normalised intensities, edges or the Scale Invariant Feature Transform (SIFT)

SIFT struggles with: (i) large changes in viewpoint, (ii) background clutter at object boundaries

Matching features over multiple views

Finding correspondences

We can use our design of **keypoints** and their **descriptors** to build a system to *recognise* a target object (specified by a reference image) from another viewpoint (query image):



Matching descriptors

A good match[1] is usually defined as one which is a small distance away in feature descriptor space ($d = 128$ for **SIFT**) as measured by Euclidean distance, $E(\mathbf{x}, \mathbf{y})$:

$$\boxed{E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}}$$

One way of solving the **correspondence problem** is to search through all the keypoint descriptors in the database images for the best match of a query feature.

Data structures can be used to organise 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, traversed with linear search. Another solution is to use tree-based data structures such as k-d trees to tackle the problem of nearest neighbour retrieval.

[1]The very best matches are those for which the nearest neighbour is much closer than the second nearest neighbour.