C18 Machine Vision and Robotics

Computer Vision

Introduction

Dr Andrea Vedaldi
4 lectures, Hilary Term

For lecture notes, tutorial sheets, and updates see
http://www.robots.ox.ac.uk/~vedaldi/teach.html

C18 Computer Vision

Overview

Prof. Andrea Vedaldi (4 lectures)

- Lecture 1: Matching, indexing, and retrieval
- Lecture 2: Convolutional neural networks
- Lecture 3: Backpropagation and automated differentiation
- Lecture 4: Applications

Prof. Victor Prisacariu (4 lectures)

- 3D vision

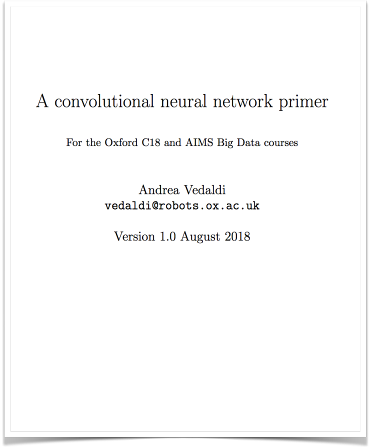
Feedback form

C18 materials

Notes, handout and tutorial sheet

Look for materials in WebLearn or at

`http://www.robots.ox.ac.uk/~vedaldi/teach.html`



A convolutional neural network primer

For the Oxford C18 and AIMS Big Data courses

Andrea Vedaldi
vedaldi@robots.ox.ac.uk

Version 1.0 August 2018

C18 Machine Vision and Robotics

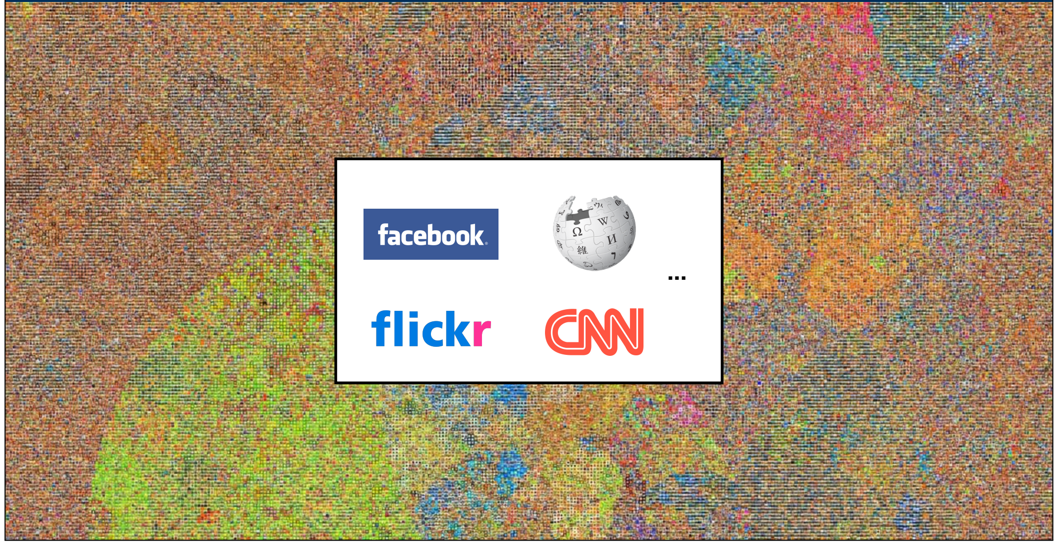
Computer Vision

Lecture 1: Matching, indexing, and retrieval

Dr Andrea Vedaldi
4 lectures, Hilary Term

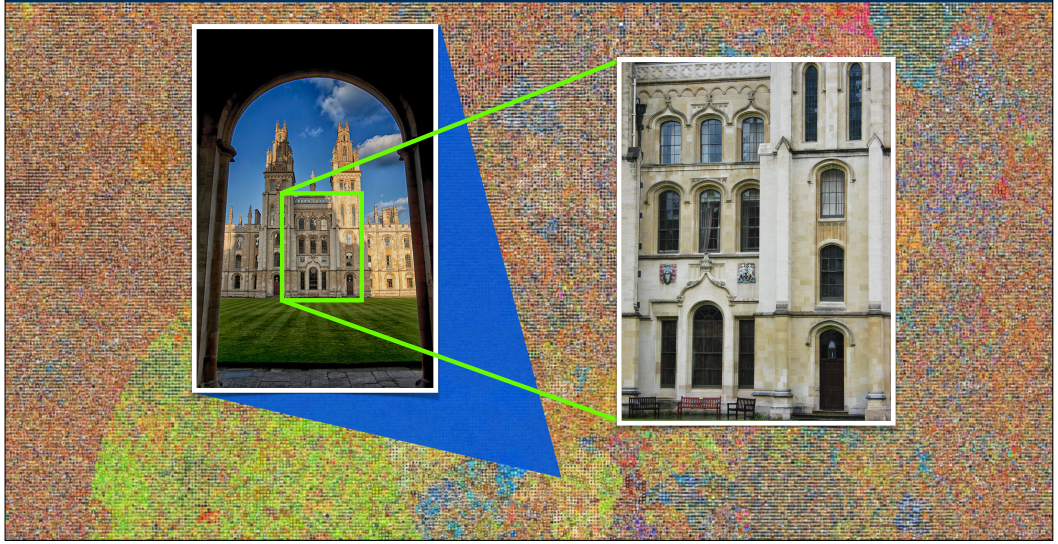
For lecture notes, tutorial sheets, and updates see
http://www.robots.ox.ac.uk/~vedaldi/teach.html

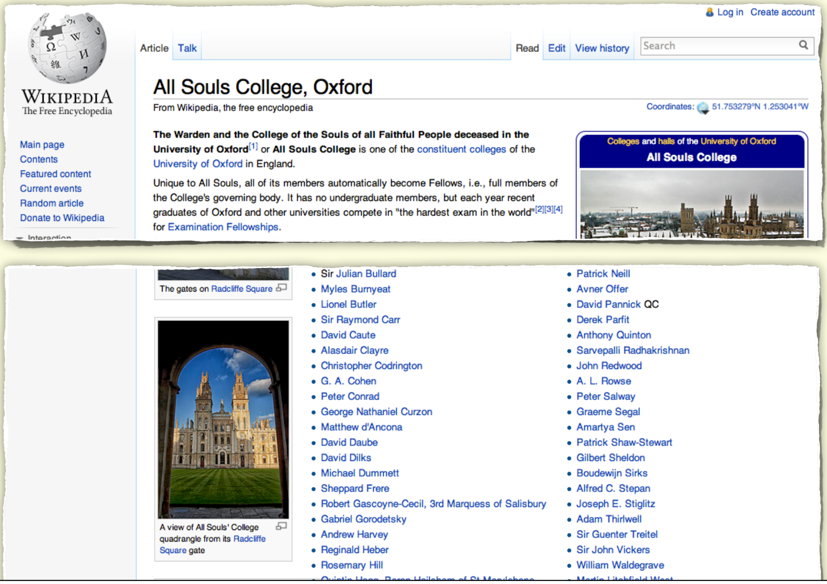
The Internet: 50 billion images and counting ...



It may **not** contain the picture **you just took** …

.. but it likely contains a similar one!





All Souls College, Oxford

From Wikipedia, the free encyclopedia

Coordinates: 51.753279°N 1.253041°W

The Warden and the College of the Souls of all Faithful People deceased in the University of Oxford[1] or **All Souls College** is one of the constituent colleges of the University of Oxford in England.

Unique to All Souls, all of its members automatically become Fellows, i.e., full members of the College's governing body. It has no undergraduate members, but each year recent graduates of Oxford and other universities compete in "the hardest exam in the world"[2][3][4] for Examination Fellowships.

Colleges and halls of the University of Oxford

All Souls College

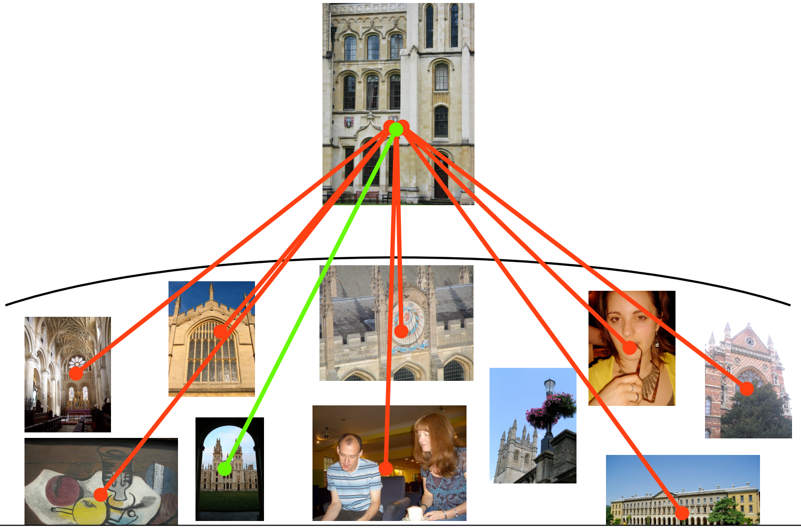
The gates on Radcliffe Square

A view of All Souls' College quadrangle from its Radcliffe Square gate

- Sir Julian Bullard
- Myles Burnyeat
- Lionel Butler
- Sir Raymond Carr
- David Caute
- Alasdair Clayre
- Christopher Codrington
- G. A. Cohen
- Peter Conrad
- George Nathaniel Curzon
- Matthew d'Ancona
- David Daube
- David Dilks
- Michael Dummett
- Sheppard Frere
- Robert Gascoyne-Cecil, 3rd Marquess of Salisbury
- Gabriel Gorodetsky
- Andrew Harvey
- Reginald Heber
- Rosemary Hill
- Quintin Hogg, Baron Hailsham of St Marylebone

- Patrick Neill
- Avner Offer
- David Pannick QC
- Derek Parfit
- Anthony Quinton
- Sarvepalli Radhakrishnan
- John Redwood
- A. L. Rowse
- Peter Salway
- Graeme Segal
- Amartya Sen
- Patrick Shaw-Stewart
- Gilbert Sheldon
- Boudewijn Sirks
- Alfred C. Stepan
- Joseph E. Stiglitz
- Adam Thirlwell
- Sir Guenter Treitel
- Sir John Vickers
- William Waldegrave
- Martin Litchfield West

Goal: search a large collection for an image of the **same object**



Matching local features

Global geometric verification

Indexing using visual words

Evaluating retrieval systems

Matching local features

Global geometric verification

Indexing using visual words

Evaluating retrieval systems

Define a similarity function between images

$F(\mathbf{I}_1, \mathbf{I}_2)$ = confidence that the **object is the same**

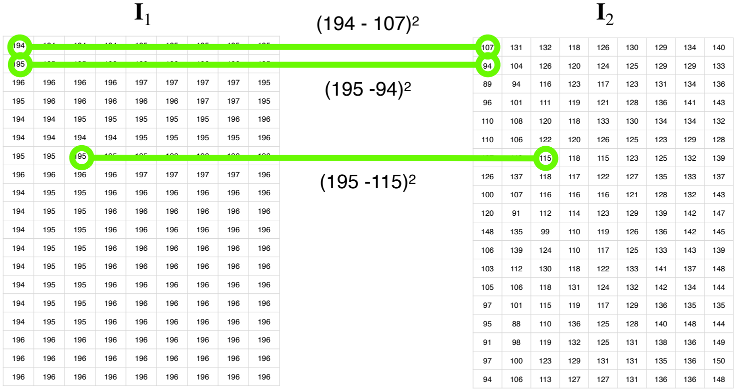
$\mathbf{I}_1$

$\mathbf{I}_2$

Image similarity (I)

Compare images as vectors of pixels

$$F(\mathbf{I}_1, \mathbf{I}_2) = -\|\mathbf{I}_1 - \mathbf{I}_2\|^2$$



$\mathbf{I}_1$

194	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
195	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
196	196	196	196	197	197	197	197	195
195	196	196	196	196	196	197	197	195
194	194	195	195	195	195	195	196	196
194	194	194	194	195	195	195	195	196
195	195	195	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
196	196	196	196	197	197	197	197	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
194	195	195	196	196	196	196	196	196
196	196	196	196	196	196	196	196	196
196	196	196	196	196	196	196	196	196
196	196	196	196	196	196	196	196	196

$(194 - 107)^2$

$(195 - 94)^2$

$(195 - 115)^2$

$\mathbf{I}_2$

107	131	132	118	126	130	129	134	140
94	104	126	120	124	125	129	129	133
89	94	116	123	117	123	131	134	136
96	101	111	119	121	128	136	141	143
110	108	120	118	133	130	134	134	132
110	106	122	120	126	125	123	129	128
[illegible]	[illegible]	115	118	115	123	125	132	139
126	137	118	117	122	127	135	133	137
100	107	116	116	116	121	128	132	143
120	91	112	114	123	129	139	142	147
148	135	99	110	119	126	136	142	145
106	139	124	110	117	125	133	143	139
103	112	130	118	122	133	141	137	148
105	106	118	131	124	132	142	134	144
97	101	115	119	117	129	136	135	135
95	88	110	136	125	128	140	148	144
91	98	119	132	125	131	138	136	149
97	100	123	129	131	131	135	136	150
94	106	113	127	127	131	136	136	148

Why do pixel values differ so much?

Nuisance factors

Viewpoint **Visibility** **Illumination** **Camera** **Noise**

I_1

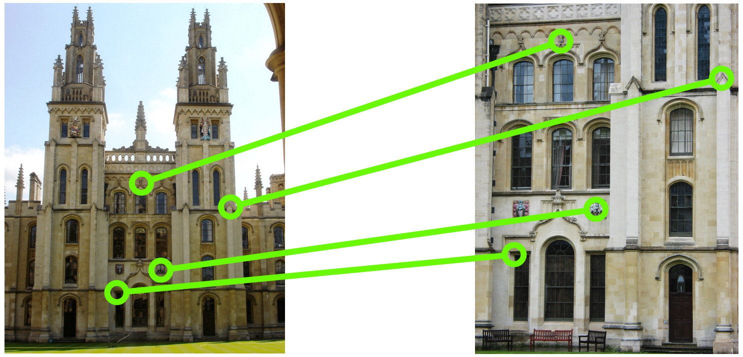
I_2

Viewpoint and visibility

Handling a variable viewpoint

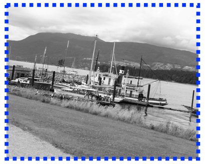
- As viewpoint changes pixels “move around” or even appear/disappear
- We need to **match corresponding pixels** before we can compare them

I_1 I_2



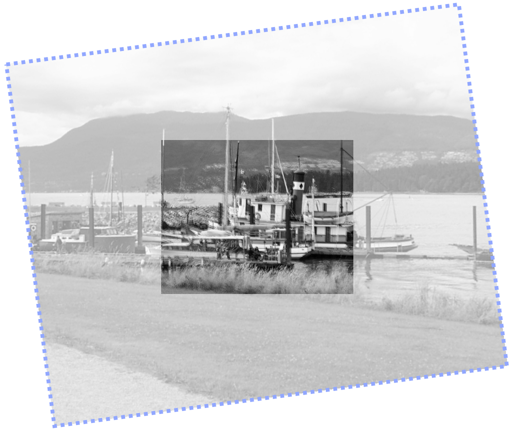
Matching and transformation

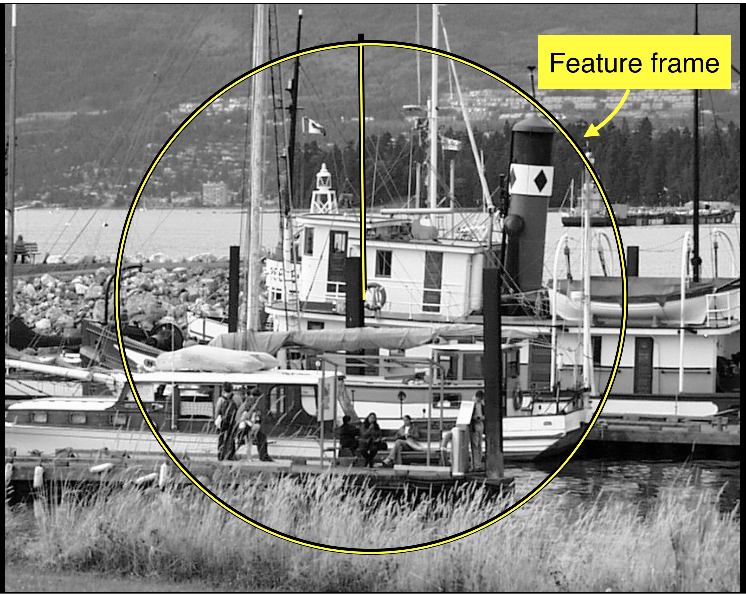
Matching can be seen as transforming or warping an image to another

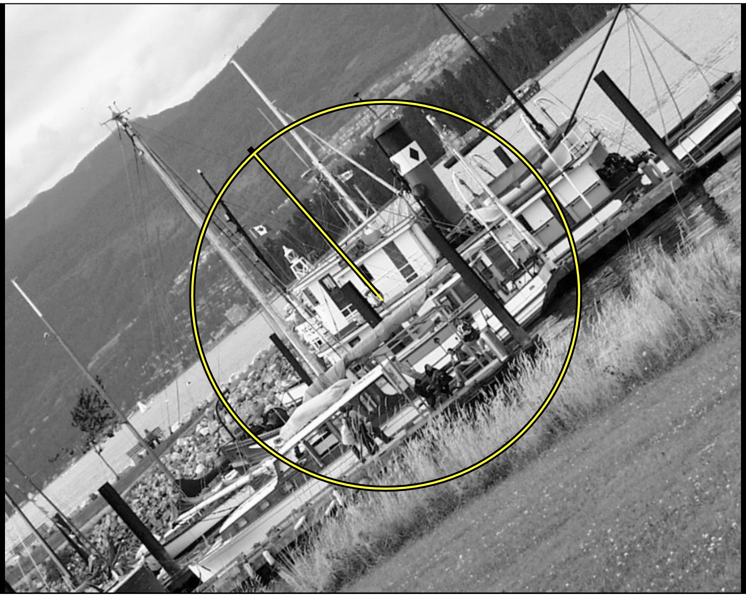


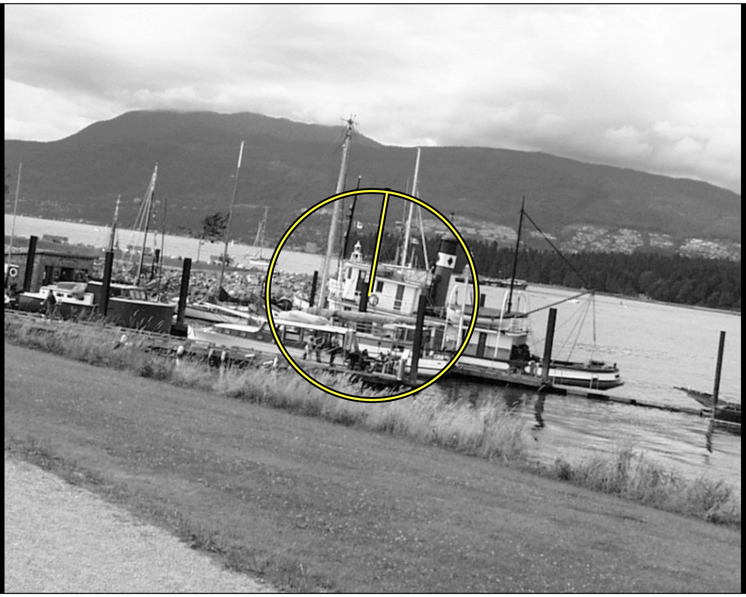
Matching and transformation

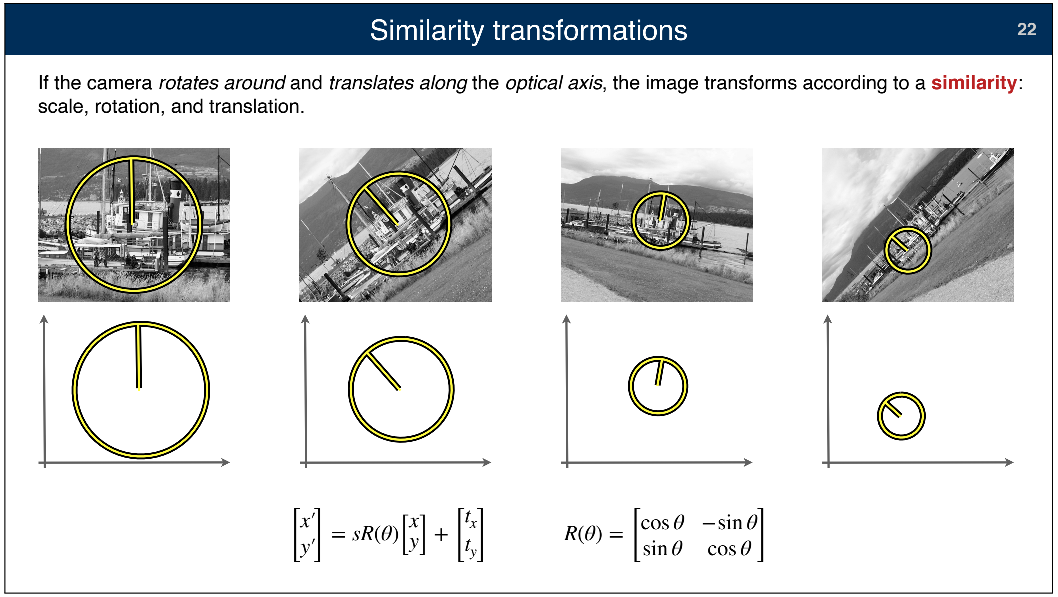
Matching can be seen as transforming or warping an image to another







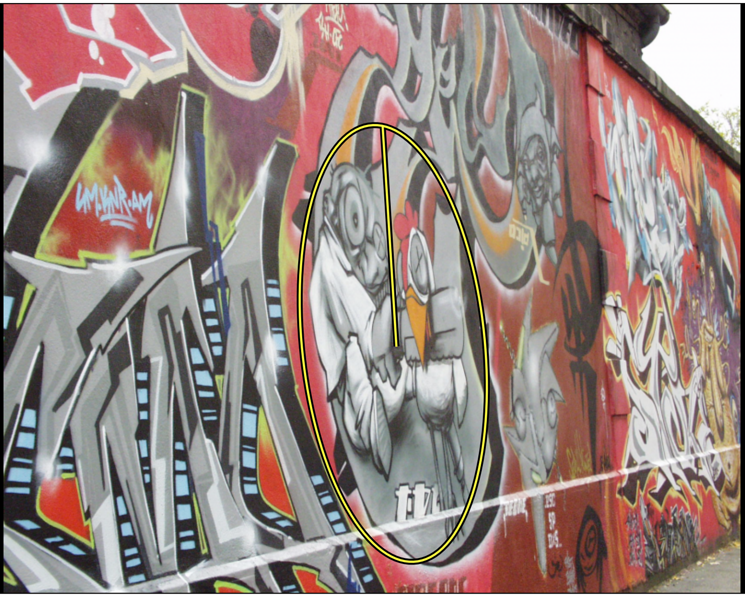




Similarity transformations

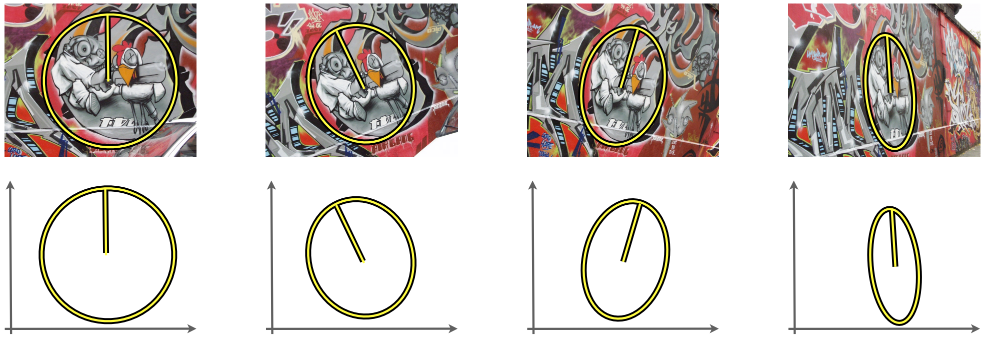
If the camera *rotates around* and *translates along* the *optical axis*, the image transforms according to a **similarity**: scale, rotation, and translation.

$$\begin{bmatrix}x'\\y'\end{bmatrix} = sR(\theta)\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}t_x\\t_y\end{bmatrix} \qquad R(\theta) = \begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}$$



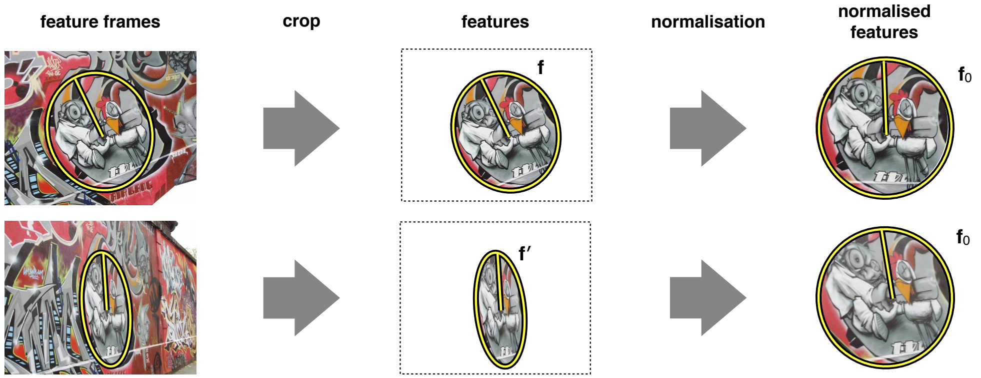
Homography/affine transformations

For *pure camera rotation* **or** if the *object is planar*, then the image transforms with an **homography** (approximated as an **affine transformation**).



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Comparing local features using normalisation



feature frames

crop

features

normalisation

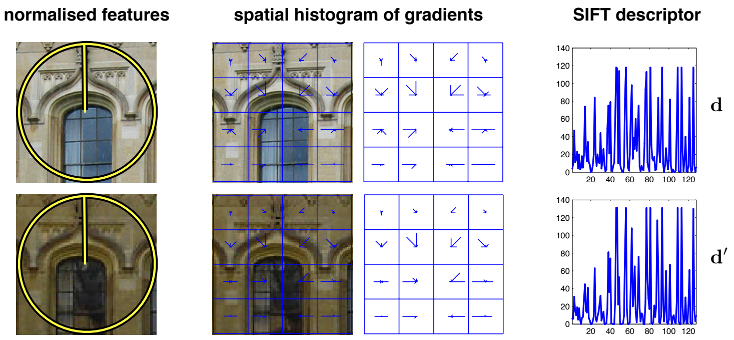
normalised features

Consider corresponding feature frames **f** and **f′**.

Then **normalisation** *undoes the effect of a viewpoint change.*

After normalisation, pixels are in correspondence (matched) and can be compared directly.

Descriptors: SIFT

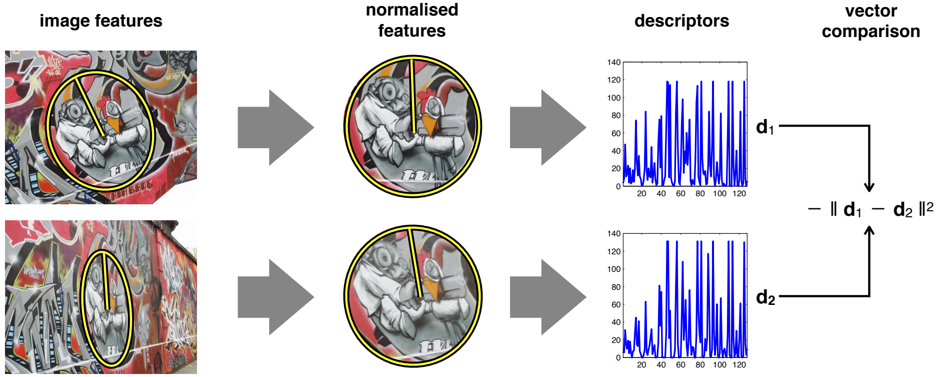


In practice, one **compares descriptors** rather than pixels. Descriptors:

- handle residual distortions, noise, illumination;
- make the representation more compact.

The most important example is the **SIFT descriptor**.

Summary: descriptors

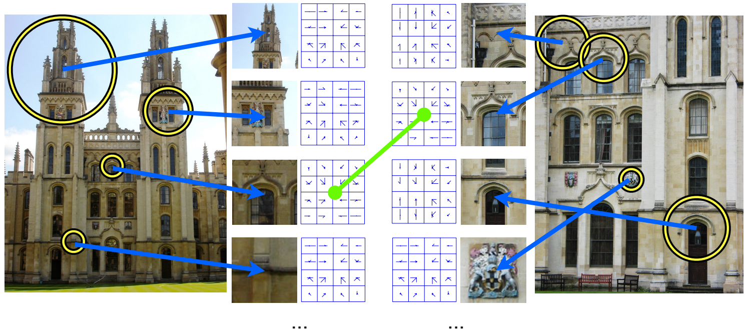


For each pair of image features

- Extract and normalize the corresponding image patches
- Compute their descriptor vectors
- Compare descriptors using the Euclidean distance

Question: how do we get the features in the first place?

Exhaustive matching

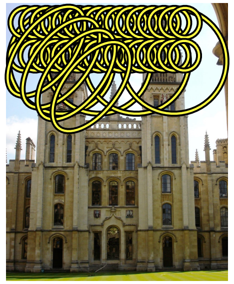


Exhaustive approach:

- Extract all possible features (all circles or all ellipses) from both images
- Test all feature pairs for possible matches

Testing all features guarantees that, if the “same feature” is visible in both images, then the corresponding patches are considered for matching.

Why exhaustive matching is unfeasible



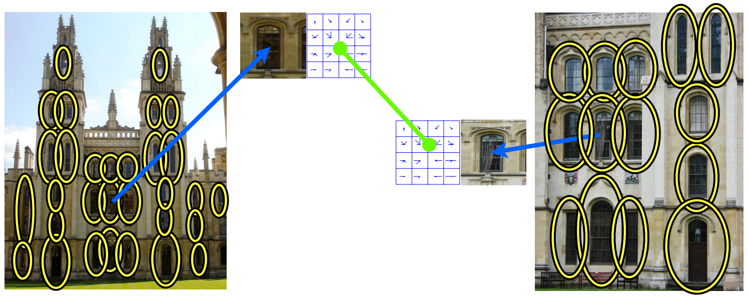
We need a method to **select** a small subset of features to match.

The cost of exhaustive matching is $O(N_1 N_2)$ where N_i is the number of features extracted from image $\mathbf{I}_i$.

Even after sampling the search space, the number of all possible features N_i is very large (~10^6).

Exhaustive matching is just too expensive.

Co-variant feature detectors

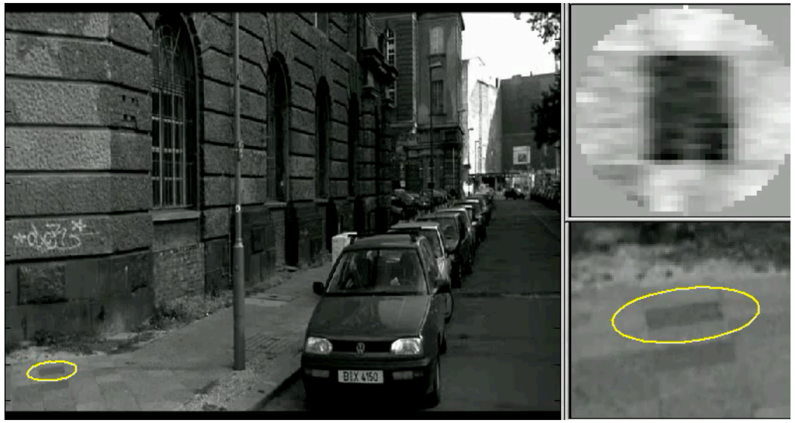


A **detector** is a rule that **selects a small subset of features** for matching.

The key is **co-variance**: the selection mechanism must pick the “same” (i.e. corresponding) features after an image transformation.

Example of a co-variant detection rule: “pick all the dark blobs”.

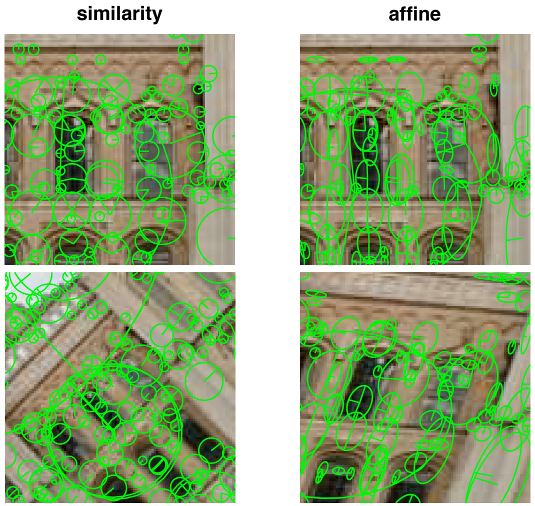
Co-variant detection, invariant descriptor



A feature extracted by the Harris-Affine detector independently from different frames of a video.

Note that the feature seems “glued on” the scene.

Co-variant detector types



similarity

affine

Properties of a detector

- repeatability
- generality
- speed

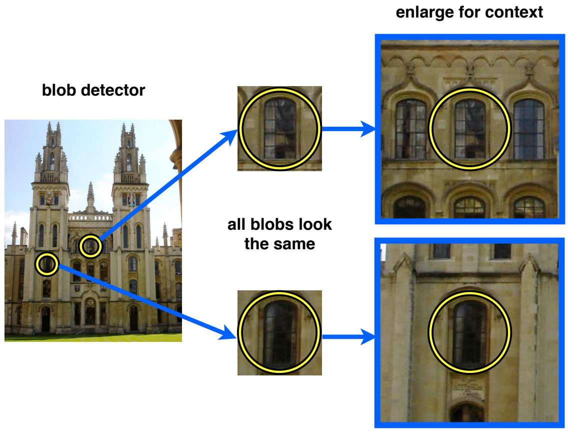
Benefits of increased covariance

- handle more general motions / objects

Cons of increased covariance

- less robust
- slower

Discriminability and support



enlarge for context

blob detector

all blobs look the same

In practice, descriptors are computed in a region **surrounding the feature.**

This is because the feature “visual anchors” (e.g. blobs) look the same and would be confused during matching.

Matching local features

Global geometric verification

Indexing using visual words

Evaluating retrieval systems

From local to global matching

Local matching

So far we have detected and then matched **local features**.

This is because normalisation is only possible if features are unoccluded and approximately planar.

Small features are much more likely to satisfy such assumptions.

On the contrary, the image as a whole is non-planar and contains plenty of self-occlusions.

Global matching

However, our goal is to compare images as a whole, not just individual patches.

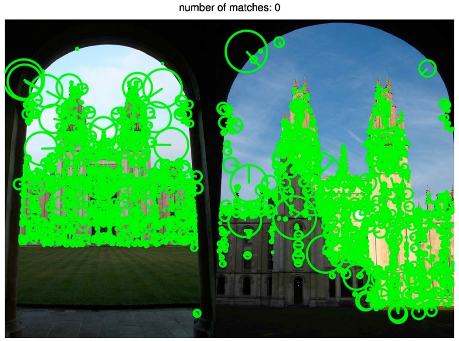
Next, we will see how to build a **global similarity score** from patch-level local comparisons.

Matching all local features

Step 0: get an image pair

Matching all local features

Step 1: detect local features **f** and extract descriptors **d**

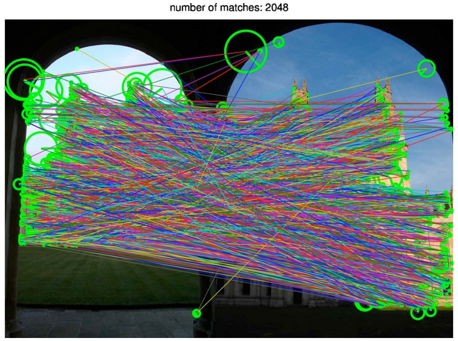


The left image has *m* features $(\mathbf{f}_1, \mathbf{d}_1), \ldots, (\mathbf{f}_m, \mathbf{d}_m)$

Right image has *n* feature $(\mathbf{f}'_1, \mathbf{d}'_1), \ldots, (\mathbf{f}'_n, \mathbf{d}'_n)$

Matching all local features

Step 2: match each descriptor to its closets one

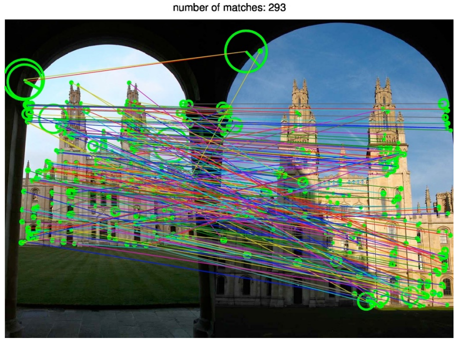


Match the *i*-th left feature to its right nearest-neighbour nn(*i*), where:

$$\mathrm{nn}(i) = \operatorname*{argmin}_{j=1,\ldots,m} \|\mathbf{d}_i - \mathbf{d}'_j\|^2$$

Matching all local features

Step 3: reject ambiguous matches using the **2nd-nn test**

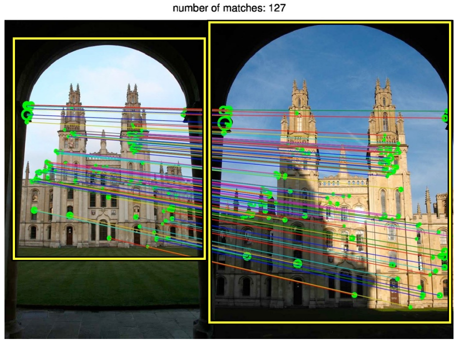


Accept a match $i \longmapsto \mathrm{nn}(i)$ only if it is at least a fraction $\tau = 0.9$ away from other possible matches:

$$\|\mathbf{d}_i - \mathbf{d}'_{\mathrm{nn}(i)}\|^2 < \tau \operatorname*{argmin}_{j \neq \mathrm{nn}(i)} \|\mathbf{d}_i - \mathbf{d}'_j\|^2$$

Matching all local features

Step 4: geometric verification

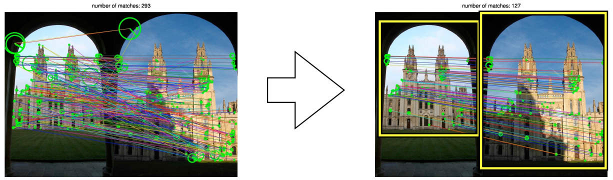


The final step is to test whether matches are consistent with an overall image transformation.

Inconsistent matches are rejected (see RANSAC).

RANSAC: optimization robust to outliers

(RANdom SAmple Consensus)



Input: M tentative feature matches ($\mathbf{x}_1$, $\mathbf{x}'_1$), …, ($\mathbf{x}_M$, $\mathbf{x}'_M$).

Output: affine transformation (A*,T*) with the largest number of inlier matches:

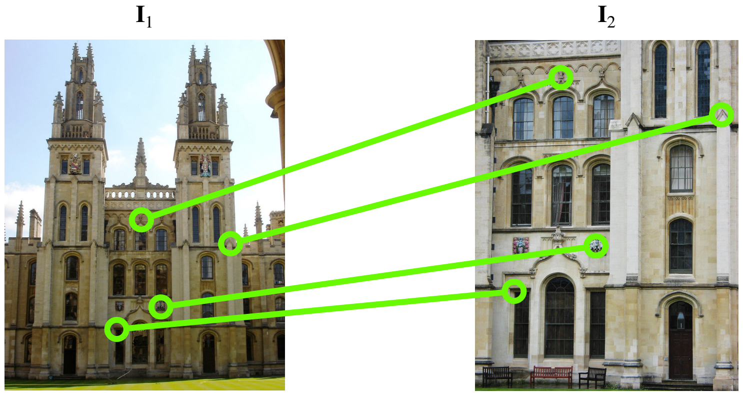
$$(A^*, T^*) = \operatorname{argmax}_{A,T} \left| \left\{ i : \|\mathbf{x}'_i - A\mathbf{x}_i - T\| < \epsilon \right\} \right|$$

1. Repeat a large number of times:
   - A. Randomly sample a **minimal subset** of matches sufficient to estimate (A,T).
   - B. Find **inliers**, i.e. other matches that are compatible with (A,T).
2. Return (A*,T*) as the pair (A,T) with the largest number of inliers.

Image similarity (II)

By counting number of verified local feature matches

$F(\mathbf{I}_1, \mathbf{I}_2)$ = # of matches after geometric verification



Matching local features

Global geometric verification

Indexing using visual words

Evaluating retrieval systems

From image matching to image search

Our matching strategy can be used to search a handful of images exhaustively. However, this is far to slow to **search a database of a billion or more images** such as Flickr, Facebook, or the Internet.

Example:

L images in the database	e.g. 10^6 - 10^{10} (Facebook)
N features per image (incl. query)	e.g. 10^3 (~ SIFT detector)
D dimensional feature descriptor	e.g. 10^2 (~ SIFT descriptor)
Exhaustive search cost: $O(N^2 L D)$	10^{11} - 10^{15} ops = 100 days - 300 years
Memory footprint: $O(NLD)$	1TB - 1PB

Goal: develop a method to search a million or more images on a single computer in under a second (and many more on computer clusters).

Issues:

- memory footprint
- matching cost (time)
- precision and recall

The inverted index

Used by Google to search the Web instantaneously

inverted index

term t	f_t	Inverted list for t
and	1	$\langle 6, 2\rangle$
big	2	$\langle 2, 2\rangle$ $\langle 3, 1\rangle$
dark	1	$\langle 6, 1\rangle$
did	1	$\langle 4, 1\rangle$
gown	1	$\langle 2, 1\rangle$
had	1	$\langle 3, 1\rangle$
house	2	$\langle 2, 1\rangle$ $\langle 3, 1\rangle$
in	5	$\langle 1, 1\rangle$ $\langle 2, 2\rangle$ $\langle 3, 1\rangle$ $\langle 5, 1\rangle$ $\langle 6, 2\rangle$
keep	3	$\langle 1, 1\rangle$ $\langle 3, 1\rangle$ $\langle 5, 1\rangle$
keeper	3	$\langle 1, 1\rangle$ $\langle 4, 1\rangle$ $\langle 5, 1\rangle$
keeps	3	$\langle 1, 1\rangle$ $\langle 5, 1\rangle$ $\langle 6, 1\rangle$
light	1	$\langle 6, 1\rangle$
never	1	$\langle 4, 1\rangle$
night	3	$\langle 1, 1\rangle$ $\langle 4, 1\rangle$ $\langle 5, 2\rangle$
old	4	$\langle 1, 1\rangle$ $\langle 2, 2\rangle$ $\langle 3, 1\rangle$ $\langle 4, 1\rangle$
sleep	1	$\langle 4, 1\rangle$
sleeps	1	$\langle 6, 1\rangle$
the	6	$\langle 1, 3\rangle$ $\langle 2, 2\rangle$ $\langle 3, 3\rangle$ $\langle 4, 1\rangle$ $\langle 5, 3\rangle$ $\langle 6, 2\rangle$
town	2	$\langle 1, 1\rangle$ $\langle 3, 1\rangle$
where	1	$\langle 4, 1\rangle$

Inverted index

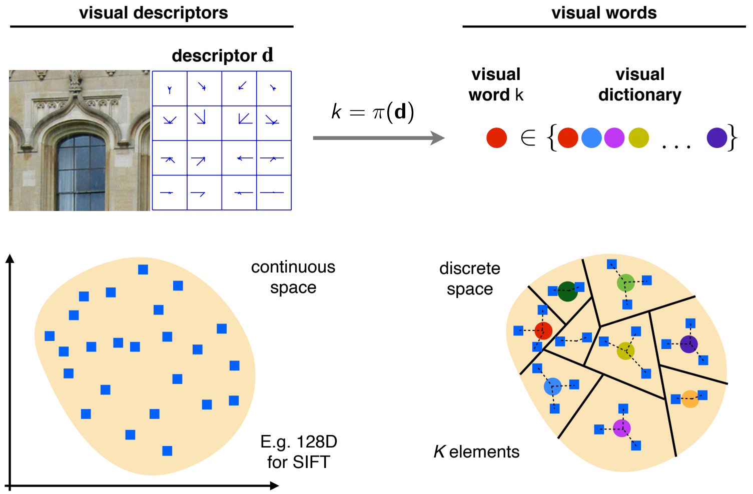
- For each word, lists all documents containing it as pairs 〈DocID, WordCount〉
- Efficient query resolution: given a word, return the corresponding list

Indexing images

- Image = document
- Word = ?

The key is to understand how to extract **"words"** from **images**

Visual words



The K-means algorithm

For learning a visual words vocabulary

The visual vocabulary is obtained by forming K **clusters** of example descriptors $(\mathbf{d}_1, \ldots, \mathbf{d}_M)$. Here M may be in the order of a 1M, and K in the order of 10^4 - 10^5.

The K cluster means ($\mu_1,\ldots,\mu_K$) are randomly initialised. Then the K-means algorithm alternates two steps:

- Find for each descriptor $\mathbf{d}_i$ the index $\pi(\mathbf{d}_i)$of its closest mean:

$$\pi(\mathbf{d}_i) = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{d}_i - \mu_k\|^2$$

- Recompute each mean μ_k from the descriptor assigned to it:

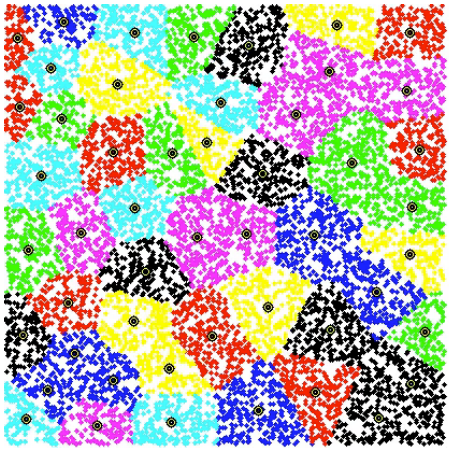
$$\mu_k = \operatorname{average}\{\mathbf{d}_i : \operatorname{nn}(\mathbf{d}_i) = k\}$$

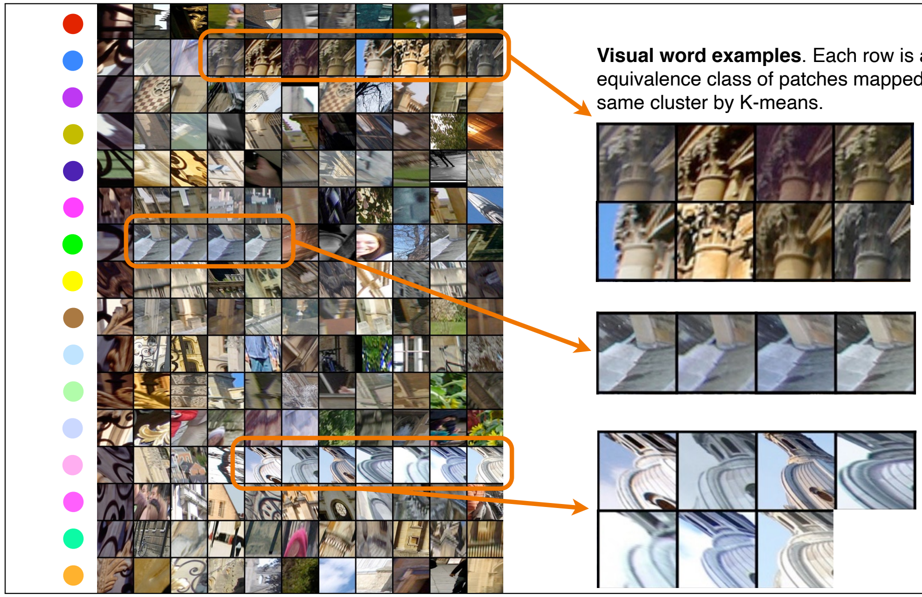
Once the means are trained, new descriptors d are quantised by mapping them to the closest mean:

$$\pi(\mathbf{d}) = \underset{k=1,\ldots,K}{\operatorname{argmin}} \|\mathbf{d} - \mu_k\|^2$$

K-means example

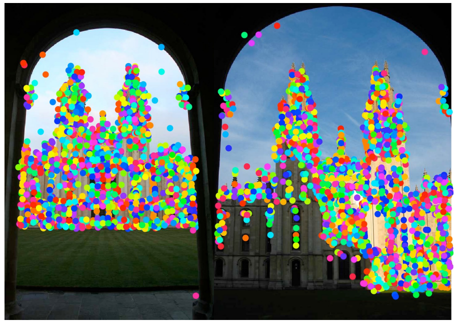
Clustering a 2D dataset





Visual word examples. Each row is an equivalence class of patches mapped to the same cluster by K-means.

From local features to visual words

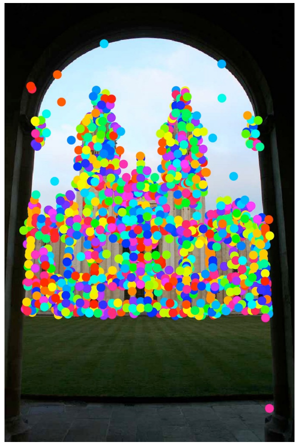


Two steps:

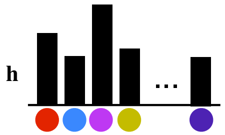
- **Extraction**. Extract local features and compute corresponding descriptors as before.
- **Quantisation**. Then map the descriptors to the K-means cluster centres to obtain the corresponding visual words.

Histogram of visual words

A simple but efficient global image descriptor



The **histogram of visual words** is the vector of the number of occurrences of the K visual words in the image:



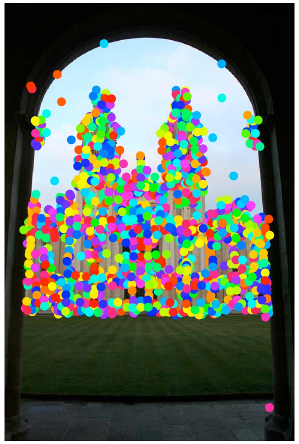
$$h_k = |\{\mathbf{d}_i : \pi(\mathbf{d}_i) = k\}|$$

If there are K visual words then $\mathbf{h} \in \mathbb{R}_+^K$.

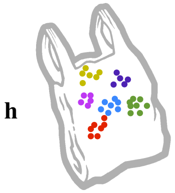
The vector $\mathbf{h}$ is a **global image descriptor**.

Histogram of visual words

A simple but efficient global image descriptor



This is also called a **bag of visual words** because it does not remember the relative positions of the features, just the number of occurrences.



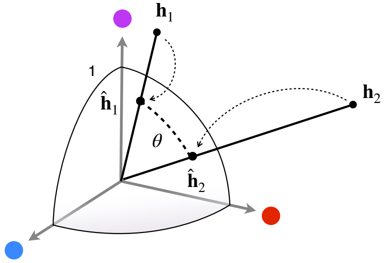
Hence, $\mathbf{h}$ **discards spatial information.**

Pros: more invariant to viewpoint changes and other nuisance factors.

Cons: less discriminative.

Comparing histograms

Cosine similarity



$$F(\mathbf{I}_1, \mathbf{I}_2) = \cos\theta = \langle \hat{\mathbf{h}}_1, \hat{\mathbf{h}}_2 \rangle$$

$$\hat{\mathbf{h}}_1 = \frac{\mathbf{h}_1}{\|\mathbf{h}_1\|}$$

$$\hat{\mathbf{h}}_2 = \frac{\mathbf{h}_2}{\|\mathbf{h}_2\|}$$

Histogram of visual words can be compared as vectors.

The relative distribution of visual words is more informative than their absolute number of occurrences.

This intuition is captured by the **cosine similarity**, which computes the angle of the L^2-normalised histograms.

Image similarity (III)

By comparing bag-of-words descriptors

$$F(\mathbf{I}_1, \mathbf{I}_2) = \langle \hat{\mathbf{h}}_1, \hat{\mathbf{h}}_2 \rangle$$

$\mathbf{I}_1$

$\mathbf{I}_2$

Search as sparse matrix multiplication

Goal: given a query vector **h**, quickly compute its similarity with all the L vectors **h**₁, **h**₂, **h**₃, ..., **h**L in the database (one per indexed image).

Express this as a vector-matrix multiplication:

$$\begin{array}{c} \mathbf{h} \\ \begin{bmatrix} 0 & 0.1 & 0.2 & 0 & \dots & 0 & \dots & 0.1 \end{bmatrix} \\ \begin{matrix} ● & ● & ● & ● & & ● & & ● \end{matrix} \end{array} \times \begin{array}{cccccc} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \dots & \mathbf{h}_L & \\ 0 & 0 & 0 & \dots & 0.1 & ● \\ 0 & 0.1 & 0 & \dots & 0 & ● \\ 0.2 & 0 & 0 & \dots & 0 & ● \\ 0.1 & 0 & 0.3 & \dots & 0.1 & ● \\ \dots & \dots & \dots & \dots & \dots & \\ 0 & 0 & 0.1 & \dots & 0.2 & ● \\ \dots & \dots & \dots & \dots & \dots & \\ 0.01 & 0.1 & 0 & \dots & 0 & ● \end{array}$$

The naive **multiplication cost** is O(K L), where K is the number of visual words and L is the database size.

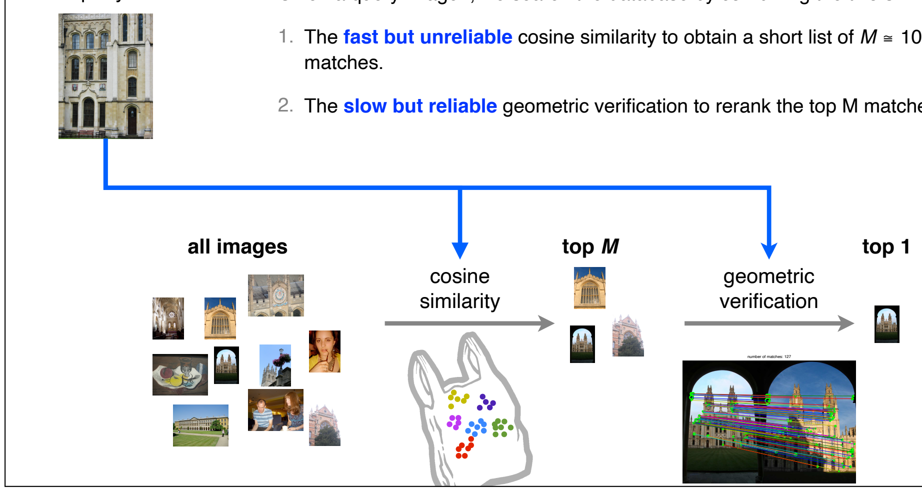
However, histograms are often highly sparse. If only a fraction $\rho \ll 1$ of entries is non-zero, then the cost reduces to $O(\rho\ K\ L)$ or even $O(\rho^2\ K\ L)$.

The **space required** i is also only $O(\rho\ K\ L)$.

Summary: image indexing and retrieval

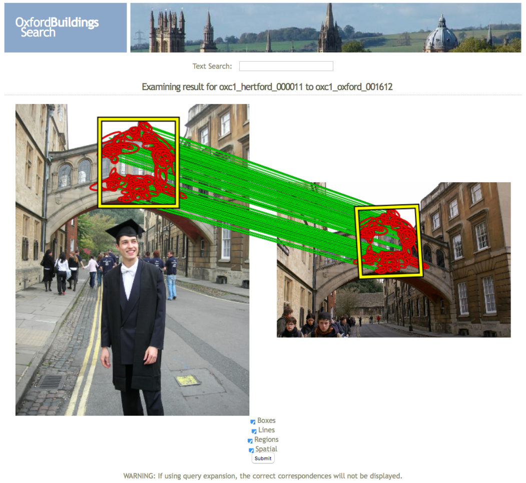
Given a query image **I**, we search the database by combining the two similarities:

1. The **fast but unreliable** cosine similarity to obtain a short list of *M* ≅ 100 possible matches.
2. The **slow but reliable** geometric verification to rerank the top M matches.



Demo

http://www.robots.ox.ac.uk/~vgg/demo/



Text Search:

Examining result for oxc1_hertford_000011 to oxc1_oxford_001612

☑ Boxes
☑ Lines
☑ Regions
☑ Spatial

WARNING: If using query expansion, the correct correspondences will not be displayed.

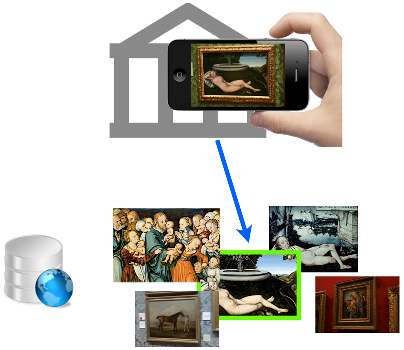
Matching local features

Global geometric verification

Indexing using visual words

Evaluating retrieval systems

Evaluating of a retrieval system



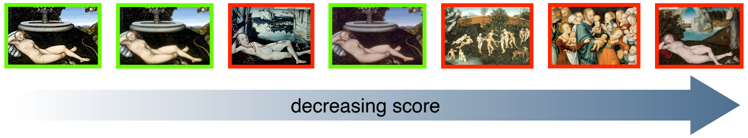
We now have a system that can match a given picture to a large database of images (e.g. Wikipedia).

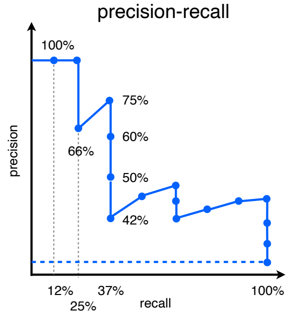
Besides speed, a **good retrieval system** must have two fundamental properties:

1. **Precision**, i.e. the ability to return **only** images that match the query.
2. **Recall**, i.e. the ability to return **all** the images that match the query.

Precision-recall curves

Assess the quality of a ranked result list





Consider all images up to rank *r* in the list:

- **Precision** @*r*: fraction of correct results in the top *r*.
- **Recall** @*r*: fraction of relevant database images that are contained in the top *r*.

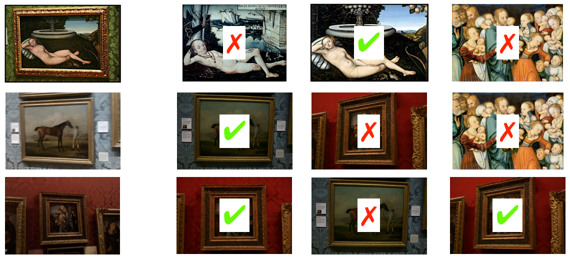
The **Average-Precision** (AP) is (roughly) the area under the PR curve.

AP is a single number summarising the overall quality of the result list.

Evaluating an image retrieval system

A benchmark usually has 1) a large image database and 2) a number of test queries for which the correct answer (relevant/irrelevant images) is known.

The retrieval system is evaluated in term of **mean average precision** (mAP), which is the mean AP of the test queries.



query	**retrieval results**			**AP**
	✗	✓	✗	35%
	✓	✗	✗	100%
	✓	✗	✓	75%
…	…	…	…	…
	mean average precision (mAP)			53%

Example benchmark: Oxford 5K

http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/

Dataset content

- ~ 5K images of Oxford
- An optional additional set of confounder (irrelevant) images
- 58 test queries

C18 Machine Vision and Robotics

Computer Vision

Lecture 2: Convolutional neural networks

Dr Andrea Vedaldi
4 lectures, Hilary Term

For lecture notes, tutorial sheets, and updates see
http://www.robots.ox.ac.uk/~vedaldi/teach.html

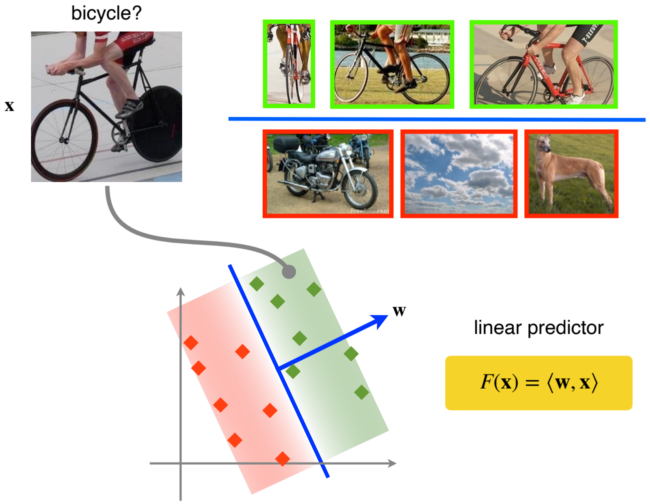
Linear predictors

We would like to build a **predictor** that can tell if an image $\mathbf{x}$ contains a certain object (say a "bicycle").

We **learn** this function from example images that do and do not contain the object.

In the simplest case, the function is a **linear predictor $F(\mathbf{x})$**:

- Images are interpreted as (high-dimensional) vectors.
- $F(\mathbf{x})$ dots $\mathbf{x}$ and a **parameter vector $\mathbf{w}$** to obtain the **score** for the positive hypothesis (bicycle).
- The **sign** of $F(\mathbf{x})$ is used as prediction.



linear predictor

$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$

Data representations

Linear predictors beyond vector inputs

Beyond vector data

A linear predictor applies to **vector data**.

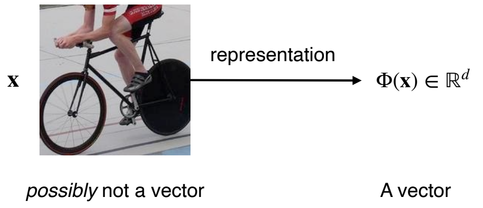
However, we want to process images, text, videos, or sounds that are not necessarily vectors.

For this, we use a **representation function Φ**, which maps data to vectors.

Non-linear classification

Representations are used even if the data $\mathbf{x}$ is already a vector.

They result in a non-linear classifier function which can be significantly **more expressive** than a linear one.



linear predictor

$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$

non-linear predictor

$F(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$

Meaningful representations

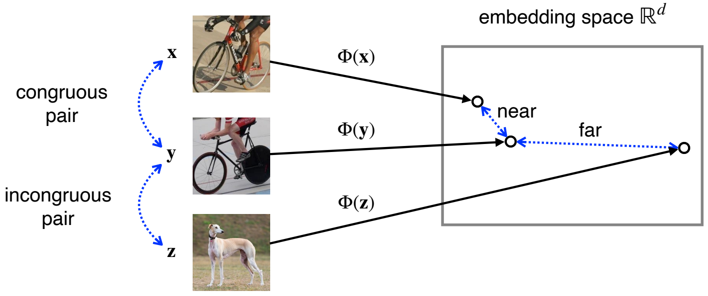
A representation should help the linear classifier to perform discrimination.

The goal is to map the **semantic similarity** between data points to a corresponding **vector similarity**.

A good representation is:

- **invariant** to nuisance factors
- **sensitive** to semantic factors

Semantic similarity → representation → Vector similarity (distance)



The perceptron

Convolutional networks

Learning via SGD

Evaluation

The perceptron

Convolutional networks

Learning via SGD

Evaluation

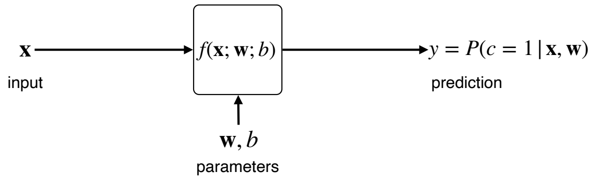
The perceptron

An early neural network by Rosenblatt (1957)

What

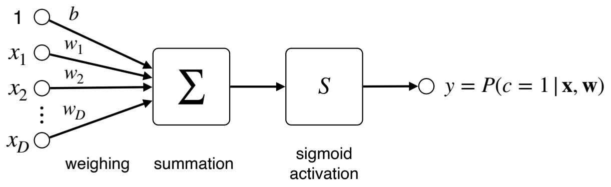
The **perceptron** maps an **input vector** $\mathbf{x}$ to a **probability value** y.

For example, y could be the probability that $\mathbf{x}$ is an image of a "bicycle" rather than not.


$\mathbf{x}$
input
$f(\mathbf{x};\mathbf{w};b)$
$y = P(c=1\,|\,\mathbf{x},\mathbf{w})$
prediction
$\mathbf{w}, b$
parameters

How

The perceptron computes this probability by **weighing** the vector components, **summing** them, and then applying a non-linear **sigmoid activation** function.



The sigmoid activation function

Makes the perceptron non-linear

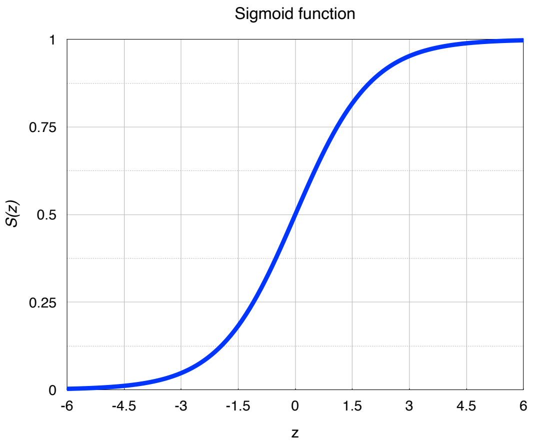
The activation function in the perceptron is a **sigmoid**

$$S(z) = \frac{1}{1+e^{-z}}$$

The sigmoid converts **real scores** z in the range $(-\infty, \infty)$ into **probability values** in the range $(0, 1)$.

It has several remarkable properties, such as the following identity for its derivative

$$\frac{dS}{dz} = S(z)(1 - S(z)) = S(z)S(-z)$$



The perceptron as a parametric function

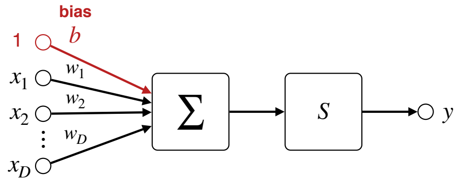
Perceptron = linear classifier + sigmoid

The perceptron is a function $f(\mathbf{x}; \mathbf{w}; b)$ parametrized by a weight vector $\mathbf{w}$ and a bias b.

The function:

1. Maps a vector $\mathbf{x}$ to a scalar score using the linear function $\langle \mathbf{x}, \mathbf{w} \rangle + b$.
2. Transforms the score into a **probability value** by applying the sigmoid function $S(z)$.

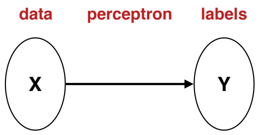
There usually is a constant **bias term** b added to the score. This can be implemented by extending the input vector with a constant element equal to 1 and including b in $\mathbf{w}$.



$$f(\mathbf{x}; \mathbf{w}, b) = S\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) = \frac{1}{1 + \exp(-w_1 x_1 - \ldots - w_D x_D - b)}$$

Training the perceptron: least square

Regard the perceptron as a parametric function from an input space **X** to an output space **Y**:



$$\mathbf{x} \longmapsto y = S\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right)$$

The parameters $(\mathbf{w}, b)$ of the perceptron are **learned empirically** by fitting the function to **example data** $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_N, y_N), .$

This can be done by solving a least-square problem:

$$E(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \left(S(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - y_i\right)^2$$

This problem is **non-linear** due to the activation function S. It needs to be solved by an iterative method such as gradient descent.

Cross-entropy loss

Better than least square for classification problems

Given the probabilistic nature of the perceptron output, usually the fitting criterion is not least square, but **maximum log-likelihood**.

The log-likelihood is computed as follows:

- The posterior probability of the 0/1 label y_i can be expressed as

$$P(y_i \mid \mathbf{x}_i; \mathbf{w}) = f(\mathbf{x}_i; \mathbf{w})^{y_i} (1 - f(\mathbf{x}_i; \mathbf{w}))^{1-y_i}$$

- The negative log-likelihood of the parameters is

$$-\log P(y_i \mid \mathbf{x}_i; \mathbf{w}) = -y_i \log f(\mathbf{x}_i; \mathbf{w}) - (1 - y_i) \log(1 - f(\mathbf{x}_i; \mathbf{w}))$$

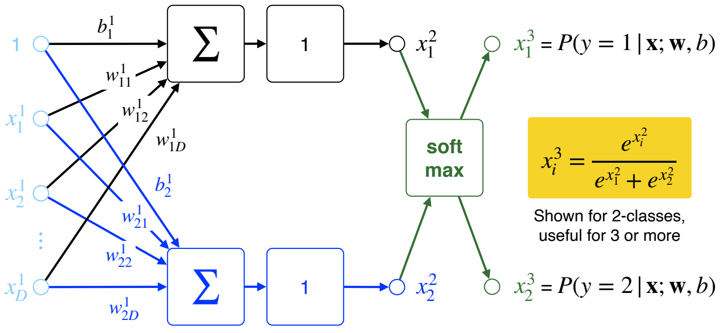
The empirical negative log-likelihood is obtained by averaging the negative log-likelihood over all the training data points

$$E(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log f(\mathbf{x}_i; \mathbf{w}) + (1 - y_i) \log(1 - f(\mathbf{x}_i; \mathbf{w}))$$

Just like the squared objective of least square, this objective function can be minimised by using an iterative method such as gradient descent.

Multi-class perceptron

Softmax layer



$$x_i^3 = \frac{e^{x_i^2}}{e^{x_1^2} + e^{x_2^2}}$$

Shown for 2-classes, useful for 3 or more

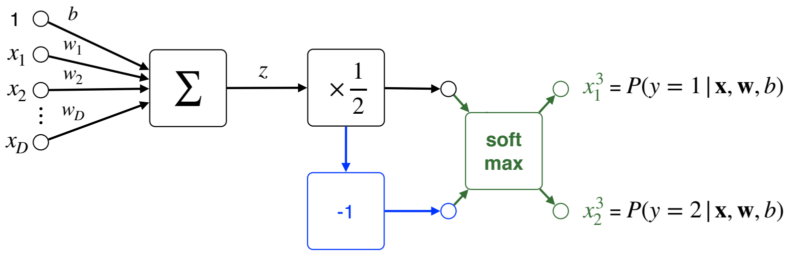
Multiple perceptrons can be combined to predict more than two classes.

Each perceptron computes the score x_c^2 for a class hypothesis $c = 1,\ldots,C$.

The vector of scores $\mathbf{x}^2$ is mapped to a vector of probabilities $\mathbf{x}^3$ using the **softmax** operator, which is a generalisation of the sigmoid.

Softmax = sigmoid for 2 classes

In the **binary case**, the softmax is the same as the sigmoid


1
b
x_1
w_1
x_2
w_2
w_D
x_D
Σ
z
$\times \frac{1}{2}$
-1
soft max
$x_1^3 = P(y = 1 \,|\, \mathbf{x}, \mathbf{w}, b)$
$x_2^3 = P(y = 2 \,|\, \mathbf{x}, \mathbf{w}, b)$

$$x_1^3 = \frac{e^{x_1^3}}{e^{x_1^3} + e^{x_2^3}} = \frac{e^{\frac{z}{2}}}{e^{\frac{z}{2}} + e^{-\frac{z}{2}}} = \frac{1}{1 + e^{-z}} = S(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

Multi-class perceptron

Learning from example data

The log-likelihood and objective function for a multi class perceptron are given by:

$$-\log P(y = y_i \,|\, \mathbf{x}_i, W) = -\log \frac{e^{\mathbf{w}_{y_i}^\top \mathbf{x} + b_{y_i}}}{\sum_{q=1}^{C} e^{\mathbf{w}_q^\top \mathbf{x} + b_q}} = -\mathbf{w}_{y_i}^\top \mathbf{x} - b_{y_i} + \log \sum_{q=1}^{C} e^{\mathbf{w}_q^\top \mathbf{x} + b_q}$$

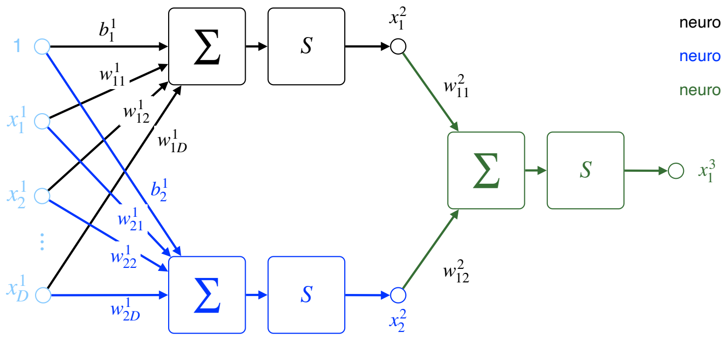
$$E(W) = \frac{1}{N} \sum_{i=1}^{N} \left(-\mathbf{w}_{y_i}^\top \mathbf{x}_i - b_{y_i} + \log \sum_{q=1}^{C} e^{\mathbf{w}_q^\top \mathbf{x}_i + b_q} \right)$$

This loss function is sometimes called **cross-entropy**. It measures the discrepancy between

- the empirical posterior distributions $Q(c \,|\, \mathbf{x}_i) = \delta(c - y_i)$and
- the predicted posterior distributions $P(c \,|\, \mathbf{x}_i) = P(y = c \,|\, \mathbf{x}_i, W)$.

Multi-layer perceptron (MLP)

Deep architectures



input

neuron 1 (1 of layer 1)

neuron 2 (2 of layer 1)

neuron 3 (1 of layer 2)

Perceptrons can also be chained, resoling in a so-called **deep neural network**. Depth refers to the fact that the function decomposes as a long (“deep”) chain of simpler perception-like functions.

The perceptron

Convolutional networks

Learning via SGD

Evaluation

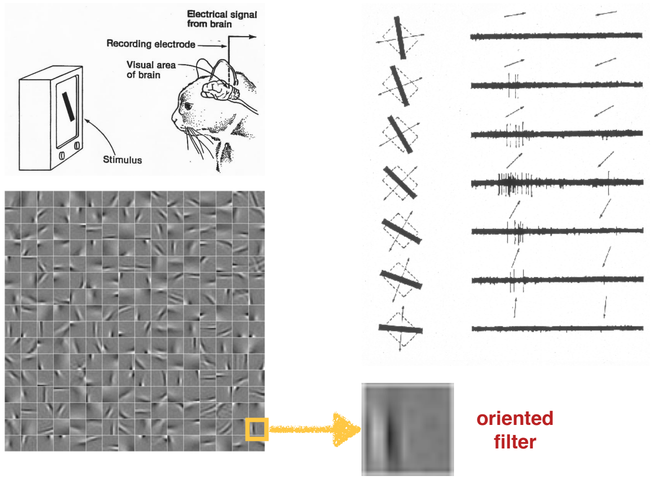
The discovery of oriented cells in the visual cortex

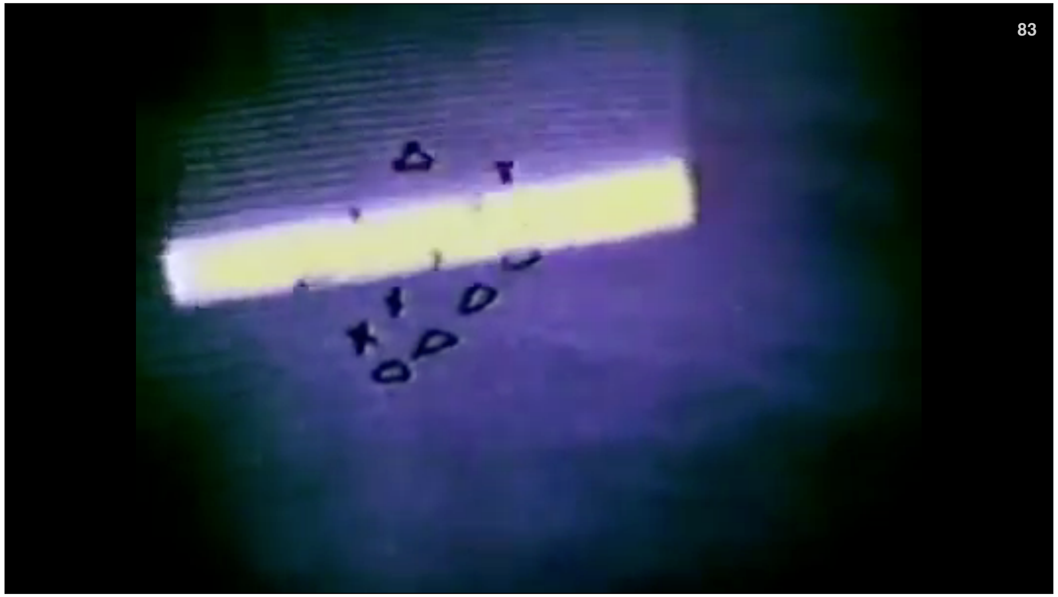
Hubel and Wiesel 1959

In 1959, Hubel & Wiesel conducted seminal experiments on the visual cortex of mammals (Nobel Prize in Physiology and Medicine in 1981).

They discovered the existence of neurons that respond to specific orientations and locations in the retina.

These neurons form a local and (statistically) translation invariant image operator.





Tensors

Variables in CNNs are usually **tensors**, i.e. **multi-dimensional array**.

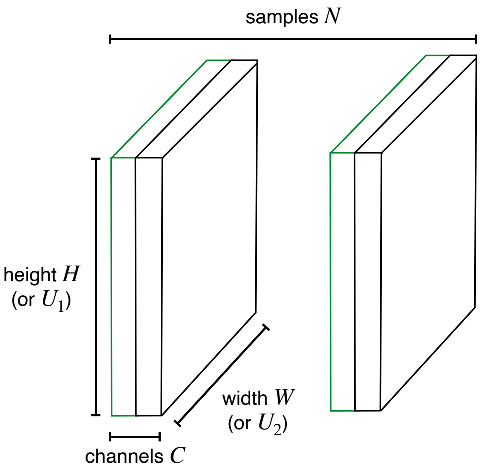
Conventionally, the dimensions are $N \times C \times U_1 \times \ldots \times U_D$ where

- N is the **batch size**, i.e. the number of data samples represented by the tensor.
- C is the number of **channels**.
- $U_1 \times \ldots \times U_D$ are the **spatial dimensions**.

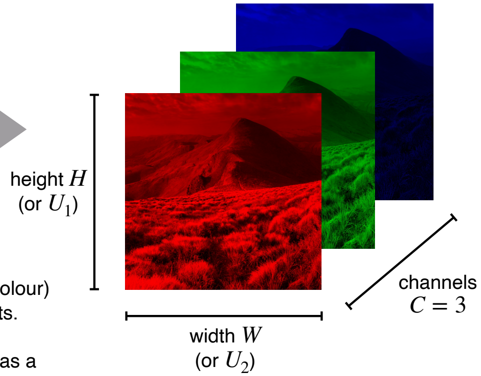
The number of spatial dimensions D can vary. E.g.:

- $D = 2$ is used to represent 2D data such as images.
- $D = 3$ is used to represent 3D data such as volumes.

In general, it is possible to assign any meaning to the dimensions (e.g. time), as required by the application.



Example: images as tensors



A **color image** can be interpreted as a tensor with $C = 3$ (colour) channels, one for each of the R, G, and B colour components.

More in general, any $C \times H \times W$ tensor can be interpreted as a $H \times W$ **field** of C-dimensional **feature vectors**.

The meaning of the feature channels is often not obvious.

Tensor indexing

Tensor elements x_{ncu} are identified via indexes, one for each dimension:

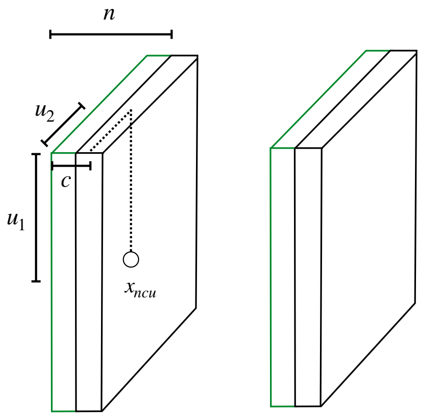
- n is the sample index in the batch
- c is the feature channel index
- u is the spatial index

The spatial index u is in fact a **multi-index**, a shorthand notation for $u = (u_1, \dots, u_D)$.

Indexes are **0-based**:

- $0 \leq n < N$
- $0 \leq c < C$
- $0 \leq u < U = (U_1, \dots, U_D)$

Generally, whenever you see a spatial multi-index, just pretend there is only one spatial dimension ($D = 1$). The extension to $D > 1$ is almost always trivial.



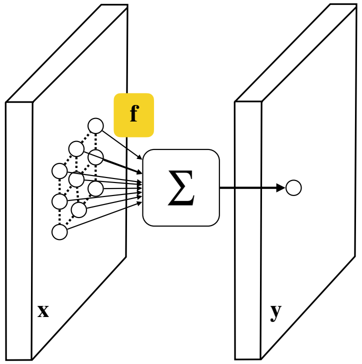
Linear convolution

A simple filtering operation

A linear filter $\mathbf{f}$ computes the weighted summation of a window of the input tensor $\mathbf{x}$.

Key properties:

- **Linearity**: the operation is linear in the input and the filter parameters.
- **Locality**: the operator looks at a small window of data.
- **Translation invariance**: all windows are processed using the same filter weights.



Linear convolution

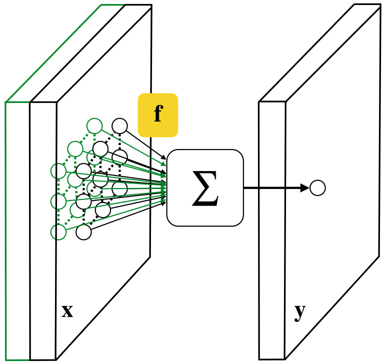
Multiple input channels

A linear filter $\mathbf{f}$ computes the weighted summation of a window of the input tensor $\mathbf{x}$.

Key properties:

- **Linearity**: the operation is linear in the input and the filter parameters.
- **Locality**: the operator looks at a small window of data.
- **Translation invariance**: all windows are processed using the same filter weights.

The filter has one channel for each input tensor channel.



Linear convolution

Multiple output channels and filter banks

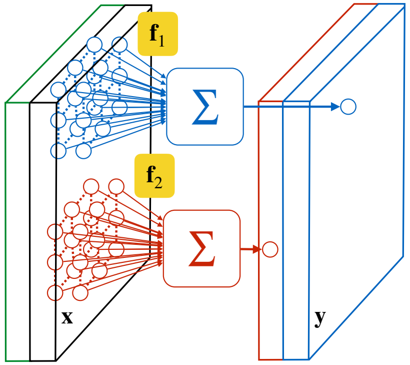
A linear filter $\mathbf{f}$ computes the weighted summation of a window of the input tensor $\mathbf{x}$.

Key properties:

- **Linearity**: the operation is linear in the input and the filter parameters.
- **Locality**: the operator looks at a small window of data.
- **Translation invariance**: all windows are processed using the same filter weights.

The filter has one channel for each input tensor channel.

A **bank of filters** is used to generated multiple output channels, one per filter.



Linear convolution

As a neural network operator

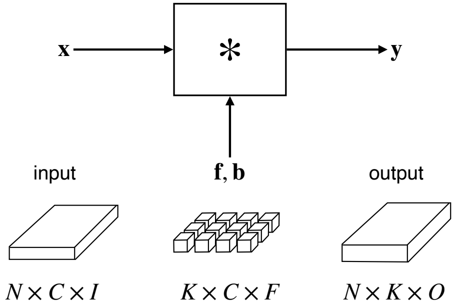
A convolutional layer is an operator that takes an **input** a tensor $\mathbf{x}$ a **filter bank** $\mathbf{f}$ and a **bias** vector $\mathbf{b}$ and produces as **output** a new tensor $\mathbf{y}$.

Dimensions:

- The **batch size** N is the same for input and output.
- Input and filters have the same **number of channels** C.
- The number of output channels K is the same as the **number of filters** in the bank.
- The output dimension O is given by

$$O = I - F + 1$$

Recall that $O = (O_1, O_2)$, $F = (F_1, F_2)$, and $I = (I_1, I_2)$ as we are using the multi-index shorthand.



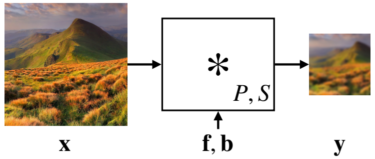
$$y_{nkv} = b_k + \sum_{c=0}^{C-1} \sum_{u=0}^{F-1} f_{kcu} \cdot x_{n,c,v+u}$$

Linear convolution

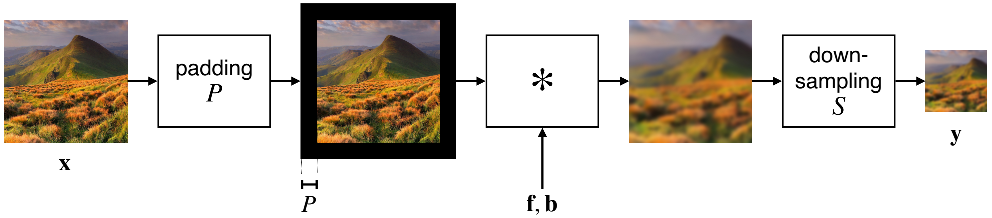
Padding and downsampling

Padding extends a tensor $\mathbf{x}$ with a border P filled with zeros.

Downsampling retain one every S pixels in a tensor, where S is called the **stride**.



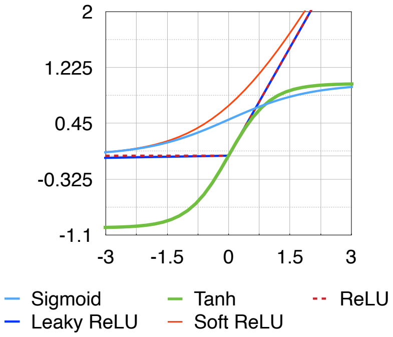
Padding and downsampling can be interpreted as additional layers before and after standard convolution:

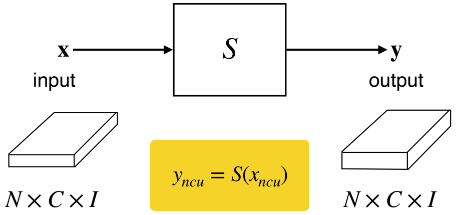


Activation functions

The non-linearity in deep networks

Activation functions are scalar non-linear functions S(z) that are applied element-wise to an input tensor **x** to generate an output tensor **y** (with the same dimensions).





$$y_{ncu} = S(x_{ncu})$$

$$\begin{cases} z = \max\{0,\ z\}, & \text{rectified linear unit (ReLU)}, \\ z = \log(1 + e^z), & \text{soft ReLU}, \\ z = \epsilon z + (1-\epsilon)\max\{0,\ z\}, & \text{leaky ReLU}, \\ z = (1 + e^{-z})^{-1}, & \text{sigmoid}, \\ z = \tanh(z), & \text{hyperbolic tangent}, \end{cases}$$

Pooling

Parameter-less non-linear filters

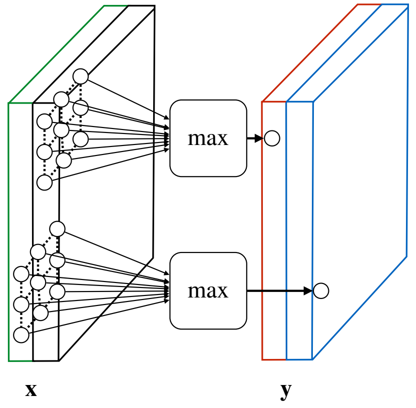
The **max pooling** operator is similar to linear filter, operating transitively on $F = (F_1, F_2)$ sized windows.

The operator extracts the maximum response for each channel and window

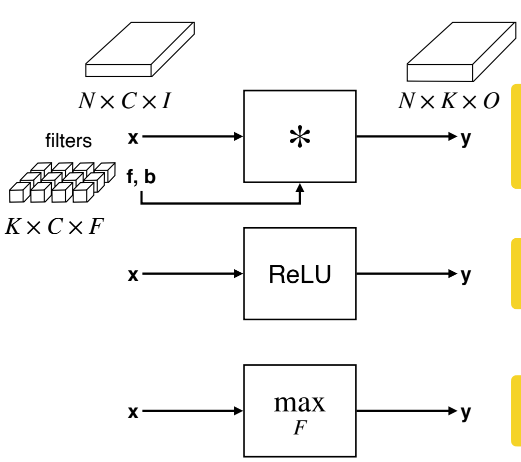
$$y_{ncv} = \max_{0 \le u < F} x_{n,c,v+u}$$

Pooling can use other operators, for example **average**

$$y_{ncv} = \frac{1}{F_1 \cdot F_2} \sum_{0 \le u < F} x_{n,c,v+u}$$



CNN layers summary



expression	dimensions
$y_{nkv} = b_k + \sum_{c=0}^{C-1} \sum_{u=0}^{F-1} f_{kcu} \cdot x_{n,c,v+u}$	$O = I - F + 1$
$y_{ncu} = \max\{0, x_{ncu}\}$	$K = C, \quad O = I$
$y_{ncv} = \max_{0 \le u < F} x_{nc,v+u}$	$O = I - F + 1$

Deep convolutional neural networks

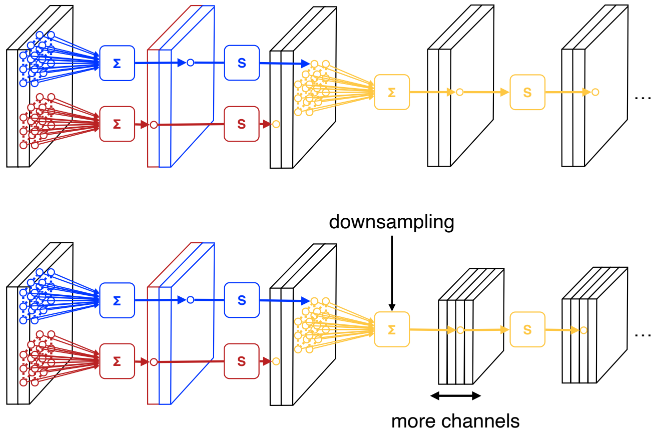
A long sequence of layers

A **deep convolutional neural network** is a chain of several layers.

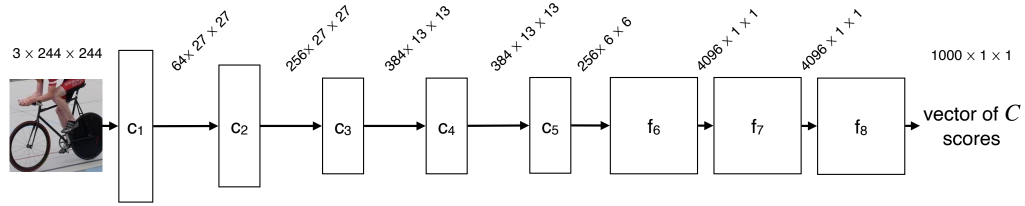
The typical pattern is to alternate linear convolution and non-linear activation, usually ReLU.

The other typical pattern is to gradually reduce the spatial resolution (via downsampling) and increase the number of feature channels.

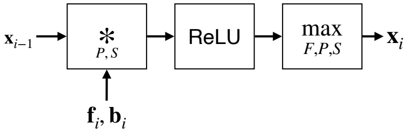
Max-pooling is often used, in combination with downsampling, to reduce resolution further.



AlexNet: a CNN for image classification



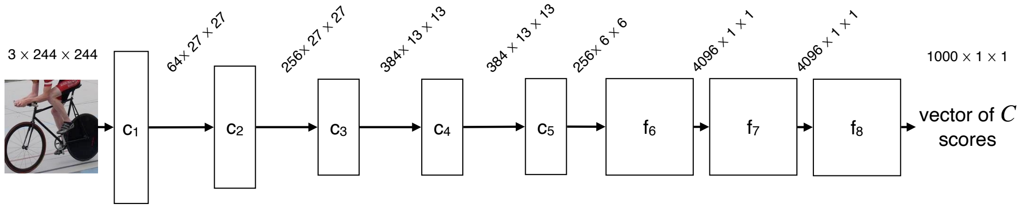
	c₁	c₂	c₃	c₄	c₅	f₆	f₇	f₈
K filters number	96	256	384	384	256	4096	4096	4096
F filter size	11	5	3	3	3	6	1	1
S filter stride	4	1	1	1	1	1	1	1
P filter padding	0	2	1	1	1	0	0	0
F' pooling size	3	3	-	-	3	-	-	-
S' pooling stride	2	2	-	-	2	-	-	-
P' pooling padding	1	0	-	-	0	-	-	-



AlexNet contains 8 **blocks**, each formed by:

- A linear convolution operator (with padding/downsampling)
- A ReLU operator (except for f₈)
- An optional max pooling operator (with padding/downsamplin

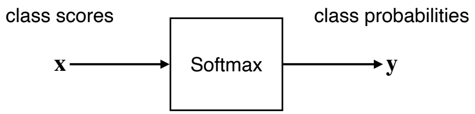
AlexNet: a CNN for image classification



The output is a 1000 × 1 × 1 tensor.

Each entry represents the score for the hypothesis that the image contains one out of a 1000 possible classes (defined in ImageNet).

Class scores are converted into probabilities by using the **softmax layer** (multi-class generalization of the sigmoid)



$$y_c = \frac{e^{x_c}}{\sum_{k=0}^{C-1} e^{x_k}}$$

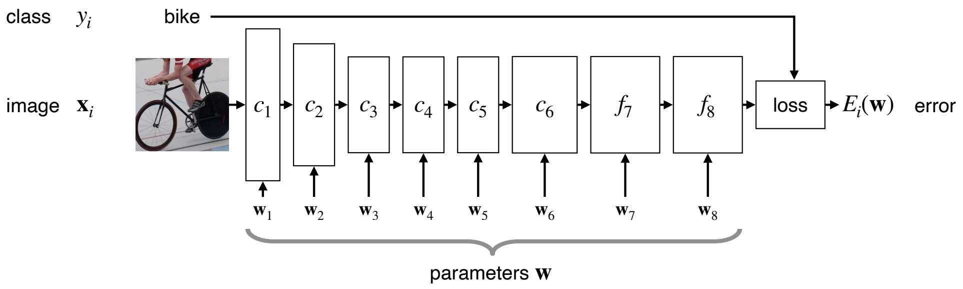
The perceptron

Convolutional networks

Learning via SGD

Evaluation

Learning a CNN



Given a dataset $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_N, y_N)$ the total error is obtained by averaging the cross-entropy loss.

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} E_i(\mathbf{w}), \qquad E_i(\mathbf{w}) = \ell(y_i, \Phi(\mathbf{x}_i))$$

The goal is to optimize this energy over the model parameters $\mathbf{w}$.

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, E(\mathbf{w})$$

Learning a CNN

ImageNet benchmark data

A CNN classifiers has millions of parameters. Hence, **learning requires massive amounts of data**.

ImageNet is a large collection of labelled image.

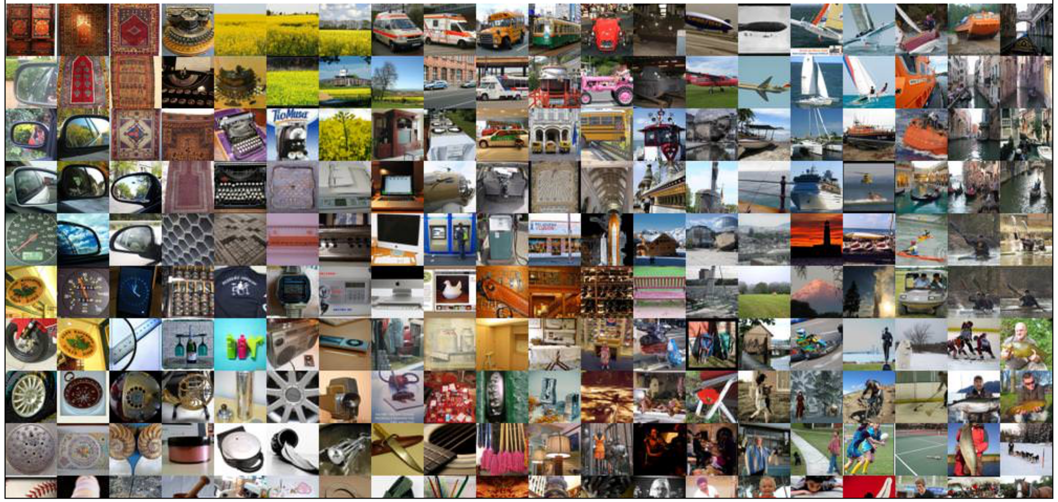
The standard subset (ILSVRC12) contains

- 1,000 object classes
- ~1,000 example images for each class
- 1.2M training images in total

Without ImageNet (or a similar dataset) it would have been impossible to develop modern deep neural networks for computer vision.

Learning a CNN

ImageNet benchmark data



Stochastic gradient descent

The objective function is an average over $N = 1.2$M data points, and so is the gradient. The cost of a single gradient descent update is way too large to be practical.

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} E_i(\mathbf{w}) \quad \Rightarrow \quad \nabla E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla E_i(\mathbf{w})$$

Stochastic gradient

Approximate the gradient **by sampling a single data point** (or a small batch of size N' << N). Perform the gradient update using the approximation.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla E_i(\mathbf{w}_t), \qquad i \sim U(\{1,2,\ldots,N\})$$

uniform distribution

Momentum

SGD can be accelerated by denoising the gradient estimate using a moving average. This average is called **momentum**.

$$\mathbf{m}_{t+1} = 0.9\,\mathbf{m}_t + \eta_t \nabla E_i(\mathbf{w}_t), \quad \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{m}_{t+1}$$

Learning a CNN

Further details and practical notes

Epochs & mini-batches

In practice, the data is visited not randomly, but in random order (without repetitions). A full pass is called an **epoch**.

Gradients are estimated by averaging **mini-batches** of 10-1000 examples. This takes advantage of parallel hardware such as GPUs.

Annealing schedule

The learning rate η_t is gradually reduced over time, usually by a factor 10 when no progress is observed.

This allows SGD to slow down and more accurately land on an optimum as the latter is approached.

Time required

On a fast GPU, it is possible to process ~1k images per second for AlexNet.

An epoch thus lasts for 20 minutes. 40-100 epochs are required, requiring 13-33 hours (faster training requires tricks such as batch normalization).

On a CPU, this could be 100 x slower (four months).

Some networks are much slower (10 - 50 x).

The perceptron

Convolutional networks

Learning via SGD

Evaluation

Evaluating deep networks

General approach

Evaluation is similar to any other machine learning method, such as SVMs or the perceptron.

Evaluation must always be done on a **held-out validation or test set**. This is because we need to test generalization, not just model fitting.

$$E(\Phi) = \frac{1}{|\mathscr{D}_{\text{validation}}|} \sum_{(\mathbf{x},y)\in\mathscr{D}_{\text{validation}}} \text{err}(\Phi(\mathbf{x}), y)$$

Most benchmarks provide validation data for this purpose.

Evaluation can use the same loss used for training. However, it is not uncommon to evaluate with respect to other, more meaningful losses **err** as well.

Top-k error

For classification problems, there are two popular losses.

Classification error: the percentage of incorrectly classified images in the validation set.

Top-k error: the percentage of images whose ground truth class is not contained in the top-k more likely classes according to the model.

The top-k error requires the network to estimate confidences. Top-1 is the same as the classification error.

C18 Machine Vision and Robotics

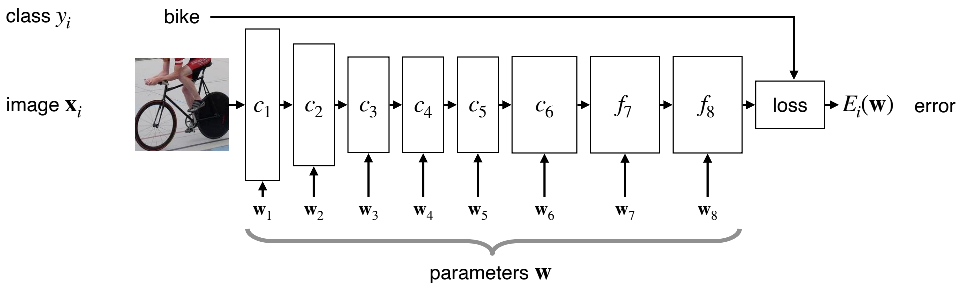
Computer Vision

Lecture 3: Backpropagation and automatic differentiation

Dr Andrea Vedaldi
4 lectures, Hilary Term

For lecture notes, tutorial sheets, and updates see
http://www.robots.ox.ac.uk/~vedaldi/teach.html

The need for gradients



In order to train a neural network we minimise the average prediction error

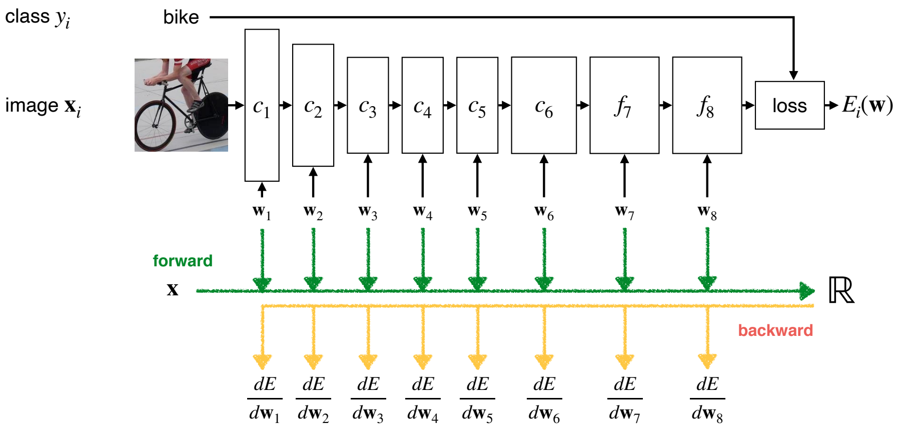
$$\operatorname*{argmin}_{\mathbf{w}_1,\ldots,\mathbf{w}_8} E(\mathbf{w}_1, \ldots, \mathbf{w}_8)$$

In order to do so, we require the **gradients of the error** with respect to all parameters

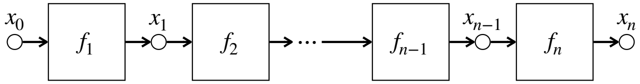
$$\nabla E = \left(\frac{dE}{d\mathbf{w}_1}, \ \cdots, \ \frac{dE}{d\mathbf{w}_8}\right)$$

Backpropagation

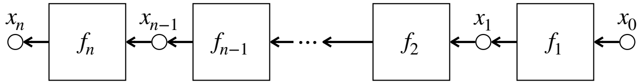
An efficient algorithm to compute the gradients



Chain rule: scalar version



Chain rule (scalar version)



A composition of n functions

$$\begin{tikzcd}
x_n & = & (f_n \arrow[d] & \circ & f_{n-1} \arrow[d] & \circ & \cdots & \circ & f_2 \arrow[d] & \circ & f_1) \arrow[d] & (x_0) \\
\frac{dx_n}{dx_0} & = & \frac{df_n}{dx_{n-1}} & \times & \frac{df_{n-1}}{dx_{n-2}} & \times & \cdots & \times & \frac{df_2}{dx_1} & \times & \frac{df_1}{dx_0} &
\end{tikzcd}$$

The derivative is obtained by using the chain rule

The vec operator

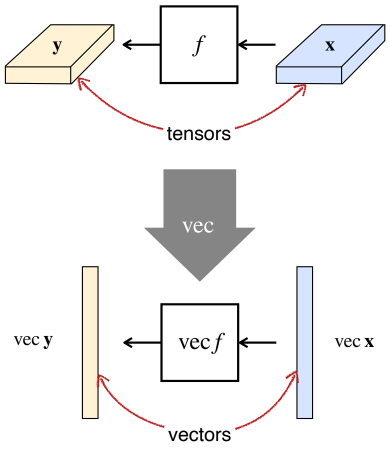
Reshaping tensors into vectors

The vec **operator** rearranges the elements of a tensor as a column vector, unrolling the tensor dimensions.

The order of unrolling is not essential, but a consistent convention must be used. PyTorch uses the **row major** convention:

$$\operatorname{vec}\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix} = \begin{bmatrix} y_{00} \\ y_{01} \\ y_{10} \\ y_{11} \end{bmatrix}$$

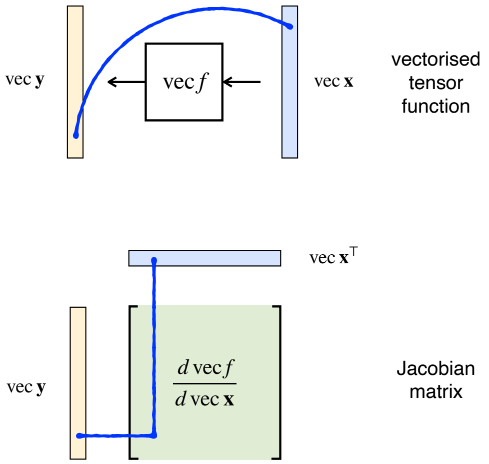
By reshaping tensors in this manner,
a tensor layer $\mathbf{y} = f(\mathbf{x})$ can be thought of as
a vector layer $\operatorname{vec} \mathbf{y} = f(\operatorname{vec} \mathbf{x})$.



Derivative of tensor-valued functions

We use the vec operator to reduce a tensor derivative to a Jacobian matrix:

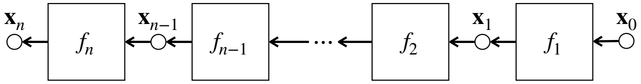
1. vec converts the tensor function $\mathbf{y} = f(\mathbf{x})$ to a vector function $\operatorname{vec} \mathbf{y} = (\operatorname{vec} f)(\operatorname{vec} \mathbf{x})$.
2. The derivative of a vector function is its Jacobian matrix.
3. The Jacobian matrix contains the derivative of each element of the output vector $\operatorname{vec} \mathbf{y}$ with respect to each element of the input vector $\operatorname{vec} \mathbf{x}$.



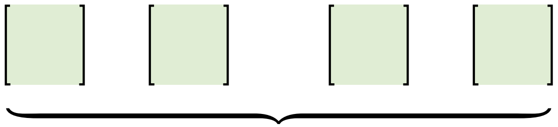
$\operatorname{vec} \mathbf{y}$
$\operatorname{vec} f$
$\operatorname{vec} \mathbf{x}$
vectorised tensor function
$\operatorname{vec} \mathbf{x}^\top$
$\operatorname{vec} \mathbf{y}$
$\frac{d \operatorname{vec} f}{d \operatorname{vec} \mathbf{x}}$
Jacobian matrix

Chain rule (tensor version)

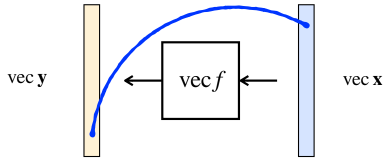
Using vec and matrix notation



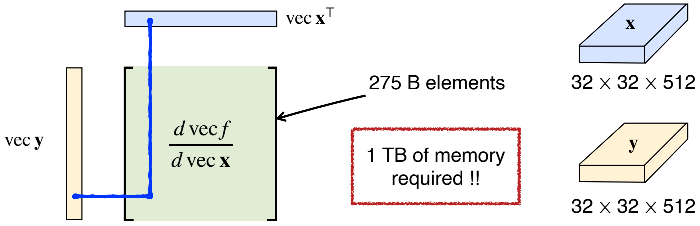
$$\underbrace{\frac{d\,\mathrm{vec}\,f_n}{d\,\mathrm{vec}\,\mathbf{x}_{n-1}} \times \frac{d\,\mathrm{vec}\,f_{n-1}}{d\,\mathrm{vec}\,\mathbf{x}_{n-2}} \times \cdots \times \frac{d\,\mathrm{vec}\,f_2}{d\,\mathrm{vec}\,\mathbf{x}_1} \times \frac{d\,\mathrm{vec}\,f_1}{d\,\mathrm{vec}\,\mathbf{x}_0}}_{\frac{d\,\mathrm{vec}\,\mathbf{x}_n}{d\,\mathrm{vec}\,\mathbf{x}_0}}$$



The (unbearable) size of tensor derivatives



The size of these **Jacobian** matrices is **huge**. Example:



vec $\mathbf{x}^\top$

vec $\mathbf{y}$

$\frac{d\,\mathrm{vec}\,f}{d\,\mathrm{vec}\,\mathbf{x}}$

275 B elements

1 TB of memory required !!

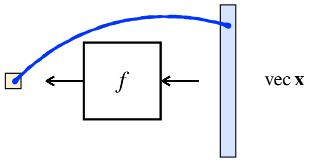
$\mathbf{x}$ 32 × 32 × 512

$\mathbf{y}$ 32 × 32 × 512

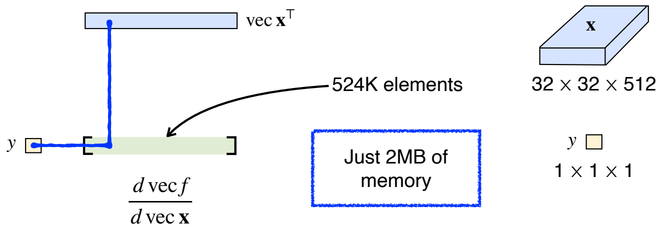
Unless the output is a scalar

Scalar

This is always the case
if the last layer
is the **loss function**



Now the Jacobian reduces to a **gradient** and has the same size as $\mathbf{x}$. Example:



vec $\mathbf{x}^\top$

y

$\frac{d\,\mathrm{vec}\,f}{d\,\mathrm{vec}\,\mathbf{x}}$

524K elements

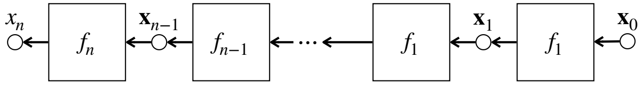
Just 2MB of memory

$\mathbf{x}$ 32 × 32 × 512

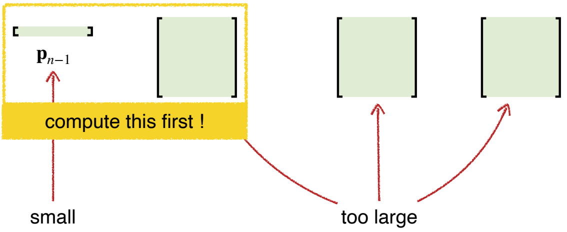
y 1 × 1 × 1

Backpropagation

Assume that x_n is a scalar



$$\frac{d\,\mathrm{vec}\,f_n}{d\,\mathrm{vec}\,\mathbf{x}_{n-1}} \times \frac{d\,\mathrm{vec}\,f_{n-1}}{d\,\mathrm{vec}\,\mathbf{x}_{n-2}} \times \cdots \times \frac{d\,\mathrm{vec}\,f_2}{d\,\mathrm{vec}\,\mathbf{x}_1} \times \frac{d\,\mathrm{vec}\,f_1}{d\,\mathrm{vec}\,\mathbf{x}_0}$$



$\mathbf{p}_{n-1}$

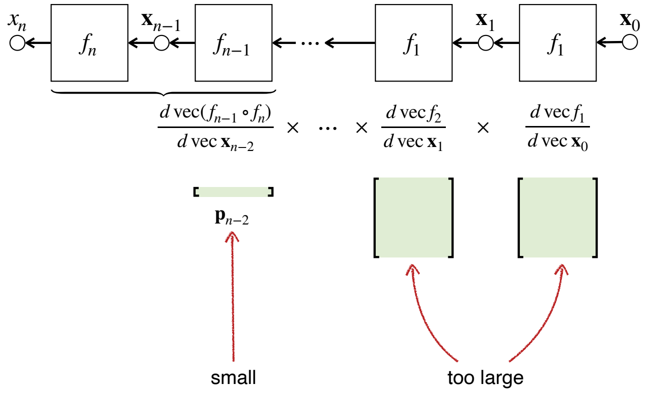
compute this first !

small

too large

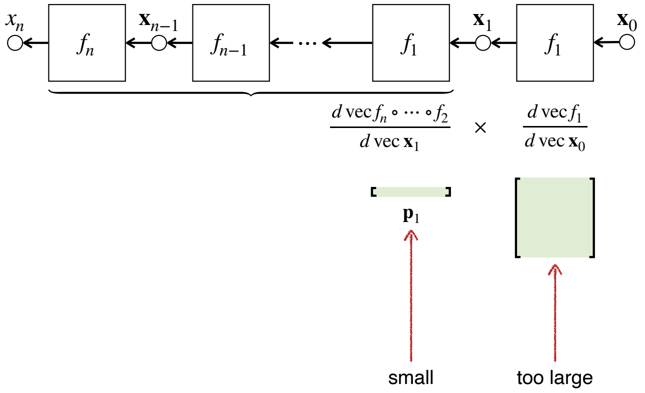
Backpropagation

Assume that x_n is a scalar


x_n
f_n
$\mathbf{x}_{n-1}$
f_{n-1}
$\cdots$
f_1
$\mathbf{x}_1$
f_1
$\mathbf{x}_0$
$\frac{d\,\mathrm{vec}(f_{n-1} \circ f_n)}{d\,\mathrm{vec}\,\mathbf{x}_{n-2}} \times \cdots \times \frac{d\,\mathrm{vec}\,f_2}{d\,\mathrm{vec}\,\mathbf{x}_1} \times \frac{d\,\mathrm{vec}\,f_1}{d\,\mathrm{vec}\,\mathbf{x}_0}$
$\mathbf{p}_{n-2}$
small
too large

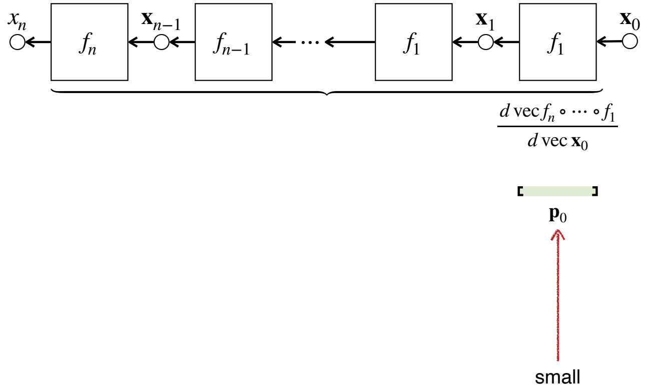
Backpropagation

Assume that x_n is a scalar


x_n
f_n
$\mathbf{x}_{n-1}$
f_{n-1}
$\cdots$
f_1
$\mathbf{x}_1$
f_1
$\mathbf{x}_0$
$\frac{d\,\mathrm{vec}\,f_n \circ \cdots \circ f_2}{d\,\mathrm{vec}\,\mathbf{x}_1} \times \frac{d\,\mathrm{vec}\,f_1}{d\,\mathrm{vec}\,\mathbf{x}_0}$
$\mathbf{p}_1$
small
too large

Backpropagation

Assume that x_n is a scalar



Vector-Jacobian product f^{BP}

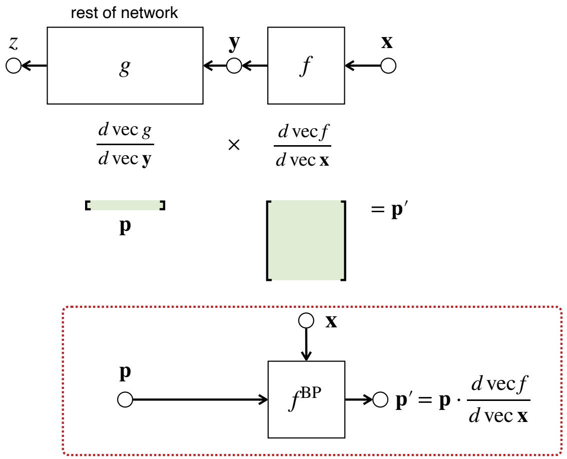
The key step is the calculation of the **vector-Jacobian product**

$$\mathbf{p}' = f^{\mathrm{BP}}(\mathbf{p}; \mathbf{x}) = \mathbf{p} \cdot \frac{d\,\mathrm{vec}\,f}{d\,\mathrm{vec}\,\mathbf{x}}$$

The result $\mathbf{p}'$ is a vector that has the same size as $\mathbf{x}$, so not too large.

The Jacobian matrix is still too large to explicitly compute.

The key idea is to use layer-specific optimisation to compute f^{BP} *without* computing the Jacobian matrix explicitly.


rest of network
z
g
$\mathbf{y}$
f
$\mathbf{x}$
$\frac{d\,\mathrm{vec}\,g}{d\,\mathrm{vec}\,\mathbf{y}} \times \frac{d\,\mathrm{vec}\,f}{d\,\mathrm{vec}\,\mathbf{x}}$
$\mathbf{p}$
$= \mathbf{p}'$
$\mathbf{x}$
$\mathbf{p}$
f^{BP}
$\mathbf{p}' = \mathbf{p} \cdot \frac{d\,\mathrm{vec}\,f}{d\,\mathrm{vec}\,\mathbf{x}}$

An example of f^{BP}

Sigmoid layer

Assume that $\mathbf{x}$ is a vector (otherwise use vec).

Let $\mathbf{y} = f(\mathbf{x})$ be the **sigmoid activation** layer:

$$f(\mathbf{x}) = \begin{bmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_C) \end{bmatrix}, \quad \sigma(x) = \frac{e^x}{e^x + e^{-x}}.$$

The Jacobian is then given by:

$$\frac{df}{d\mathbf{x}} = \begin{bmatrix} \frac{d\sigma(x_1)}{dx_1} & \frac{d\sigma(x_1)}{dx_2} & \cdots & \frac{d\sigma(x_1)}{dx_C} \\ \frac{d\sigma(x_2)}{dx_1} & \frac{d\sigma(x_2)}{dx_2} & \cdots & \frac{d\sigma(x_2)}{dx_C} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d\sigma(x_C)}{dx_1} & \frac{d\sigma(x_C)}{dx_2} & \cdots & \frac{d\sigma(x_C)}{dx_C} \end{bmatrix}.$$

Most derivatives are equal to zero:

$$\frac{d\sigma(x_c)}{dx_k} = \begin{cases} \dot{\sigma}(x_c), & c = k, \\ 0, & c \neq k. \end{cases}, \qquad \dot{\sigma}(x) = \frac{d\sigma}{dx}(x).$$

The *Jacobian* is the diagonal matrix

$$\frac{df}{d\mathbf{x}} = \begin{bmatrix} \dot{\sigma}(x_1) & 0 & \ldots & 0 \\ 0 & \dot{\sigma}(x_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \dot{\sigma}(x_C) \end{bmatrix}.$$

f^{BP} is then given by

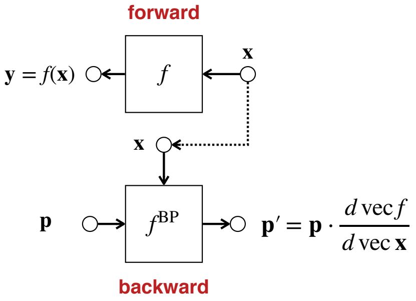
$$f^{\mathrm{BP}}(\mathbf{p}; \mathbf{x}) = \mathbf{p} \cdot \frac{df}{d\mathbf{x}} = \begin{bmatrix} p_1\dot{\sigma}(x_1) & p_2\dot{\sigma}(x_2) & \cdots & p_C\dot{\sigma}(x_C) \end{bmatrix}.$$

f^{BP} as a reversed layer

The function f is a **forward layer** $\mathbf{y} = f(\mathbf{x})$.

The function f^{BP} defines a **backward layer** operating in the reverse direction $\mathbf{p}' = f^{\mathrm{BP}}(\mathbf{p}; \mathbf{x})$.

This generates a new mirror block diagram; the forward diagram feeds into the backward diagram via $\mathbf{x}$.


forward
$\mathbf{y} = f(\mathbf{x})$
f
$\mathbf{x}$
$\mathbf{x}$
$\mathbf{p}$
f^{BP}
$\mathbf{p}' = \mathbf{p} \cdot \frac{d \operatorname{vec} f}{d \operatorname{vec} \mathbf{x}}$
backward

f^{BP} computes gradients

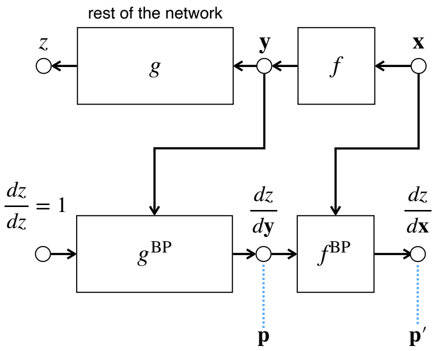
So what are these vectors $\mathbf{p}$ anyways?

Each $\mathbf{p}$ is the **gradient** of the network output z with respect to the corresponding variable $\mathbf{x}$:

$$\mathbf{p}' = \frac{dz}{d\mathbf{x}} \quad \text{or even just} \quad \mathbf{p}' = d\mathbf{x}$$

Thus f^{BP} computes a gradient out of another gradient:

$$\mathbf{p} = \frac{dz}{d\mathbf{y}} \quad \Rightarrow \quad \mathbf{p}' = f^{\mathrm{BP}}(\mathbf{p}; \mathbf{x}) = \frac{dz}{d\mathbf{x}}$$



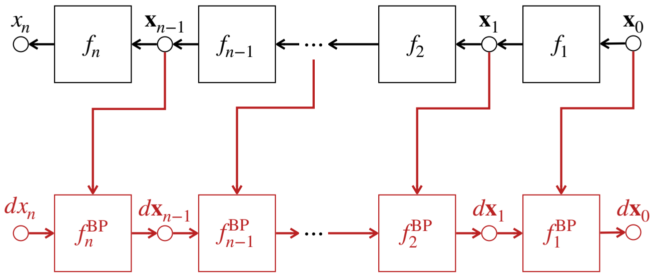
Compute graphs

Keeping track of calculations for automatic differentiation

The **compute graph** is a mechanism to keep track of the calculations in a program.

It can be used to automatically deduce which computations are required to compute the gradients.

These computations can then be added to the graph and the process repeated to obtain higher-order derivatives.



Compute graphs

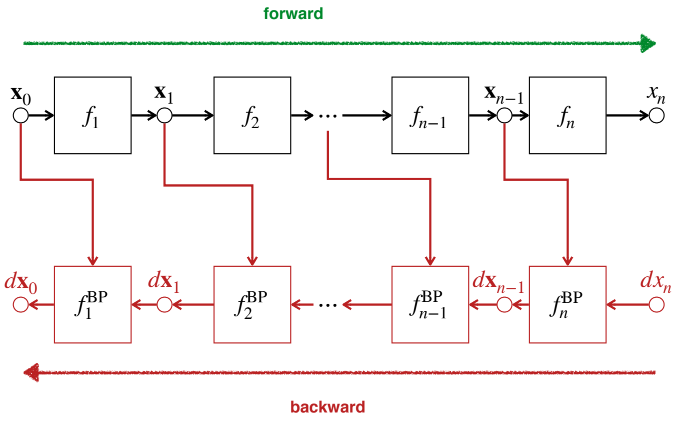
Keeping track of calculations for automatic differentiation

The **compute graph** is a mechanism to keep track of the calculations in a program.

It can be used to automatically deduce which computations are required to compute the gradients.

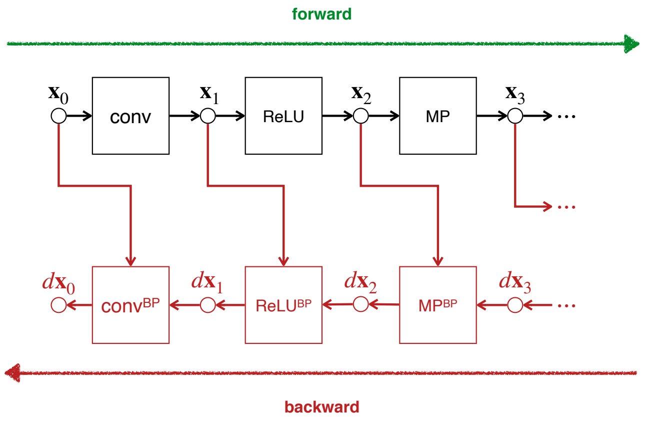
These computations can then be added to the graph and the process repeated to obtain higher-order derivatives.

The graph is more commonly shown the other way around, with the forward direction left to right.



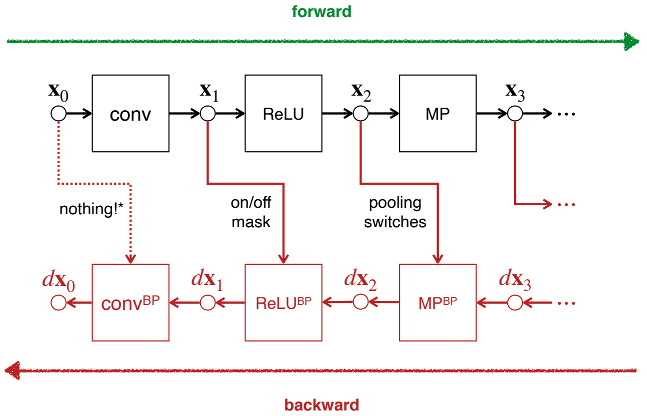
Backpropagation network

Conv, ReLU, MP and their transposed blocks



Sufficient statistics and bottlenecks

Sometimes much less information is needed



* Unless the gradients w.r.t. the filter parameters are also needed

Automatic differentiation (AutoDiff)

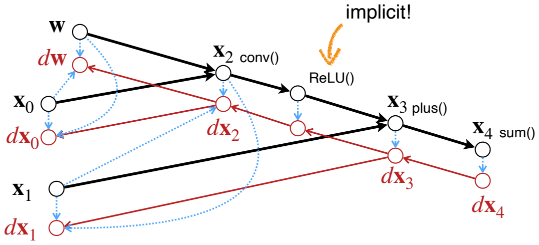
A PyTorch example

Modern machine learning toolboxes provide **AutoDiff**.

This means that calculations can be performed as normal in a programming language.

Underneath, the toolbox builds a compute graph.

Eventually, gradients can be requested.



```
import torch

# Define two random inputs, both requiring grads
x0 = torch.randn(1,3,20,20, requires_grad=True)
x1 = torch.randn(1,10,18,18, requires_grad=True)

# Get a convolutional layer. It contains
# a parameter tensor conv.weight with requires_grad=True
conv = torch.nn.Conv2d(3,10,3)

# Intermediate calculations
x2 = conv(x0)
x3 = torch.nn.ReLU()(x2) + x1
x4 = x3.sum() # Scalar!

# Invoke AutoGrad to compute the gradients
x4.backward()

# Print the gradient shapes
print(x0.grad.shape)
print(x1.grad.shape)
print(conv.weight.grad.shape)
```

C18 Machine Vision and Robotics

Computer Vision

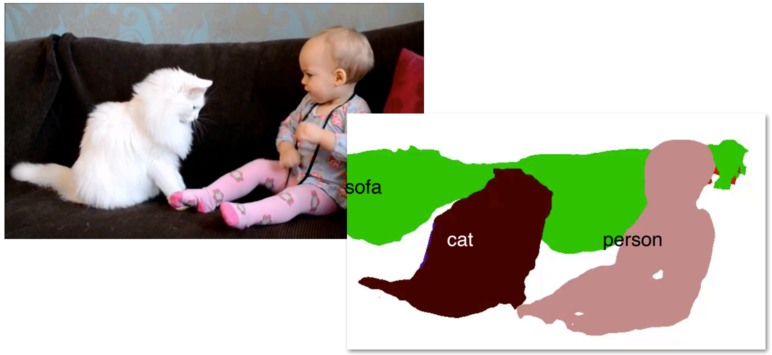
Lecture 4: Applications

Dr Andrea Vedaldi
4 lectures, Hilary Term

For lecture notes, tutorial sheets, and updates see
http://www.robots.ox.ac.uk/~vedaldi/teach.html

Semantic image segmentation

Label individual pixels



Face analysis

Detection, verification, recognition, emotion, 3D fitting

E.g. VGG-Face

Text spotting

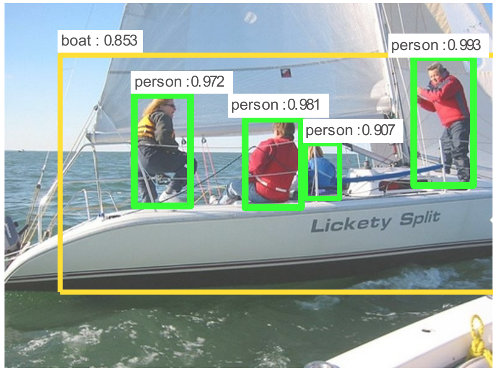
Detection, word recognition, character recognition

E.g. SynthText and VGG-Text

http://zeus.robots.ox.ac.uk/textsearch/#/search/

Object detection

Extract individual object instances



Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation
R. Girshick, J. Donahue, T. Darrell, J. Malik, CVPR 2014

Architectures

Segmentation

Detection

Tracking

Architectures

Segmentation

Detection

Tracking

Neural network architectures

Evolution

AlexNet (2012)

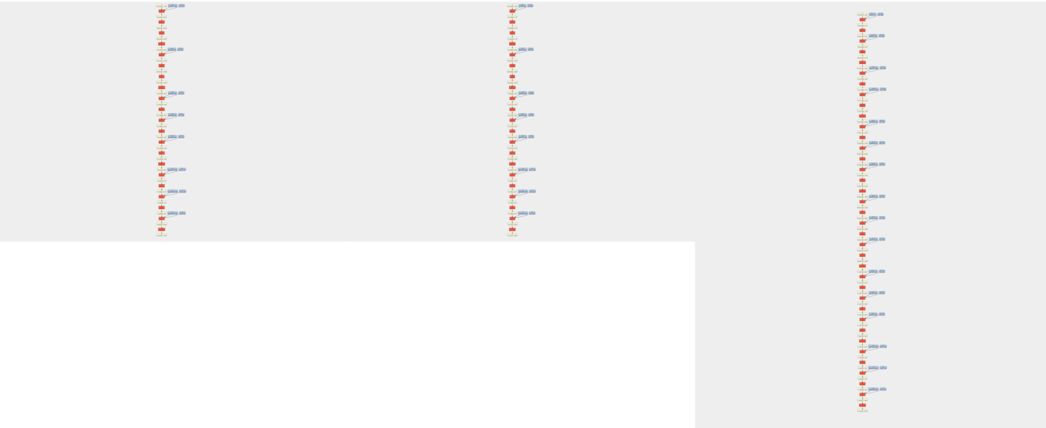
5 convolutional layers

3 fully-connected layers

Neural network architectures

Evolution

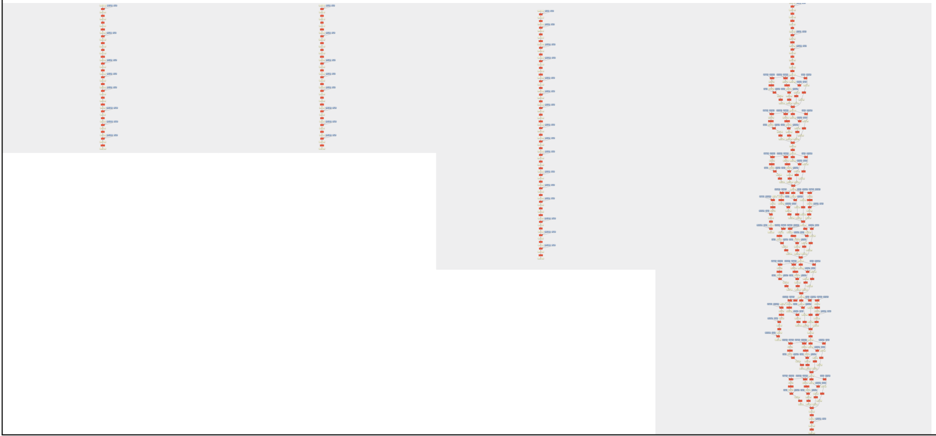
AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014)



Neural network architectures

Evolution

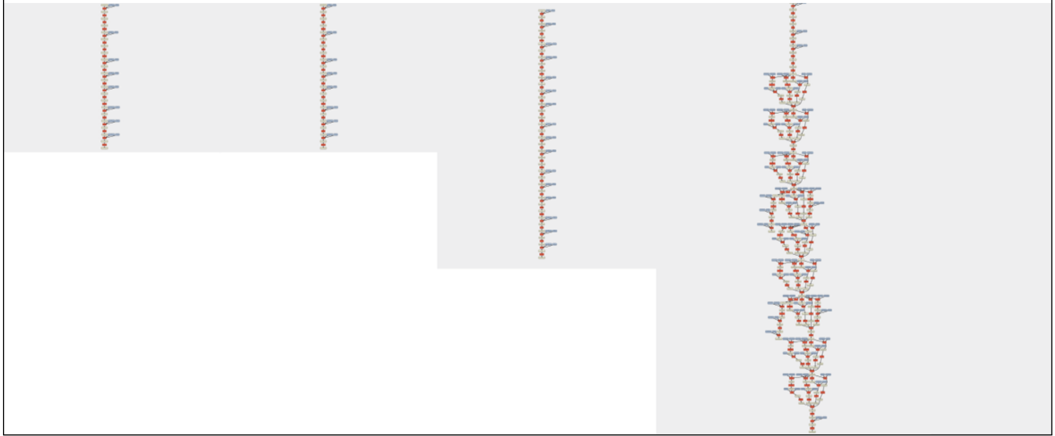
AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)



Neural network architectures

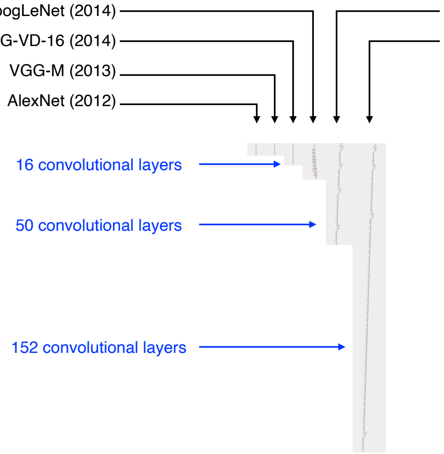
Evolution

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)



Neural network architectures

Evolution



GoogLeNet (2014)

VGG-VD-16 (2014)

VGG-M (2013)

AlexNet (2012)

ResNet 50 (2015)

ResNet 152 (2015)

16 convolutional layers

50 convolutional layers

152 convolutional layers

Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks.* In Proc. NIPS, 2012.

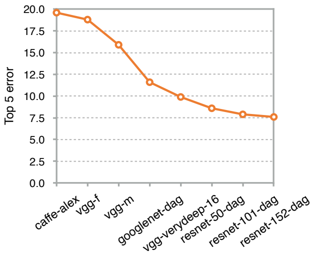
C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions.* In Proc. CVPR, 2015.

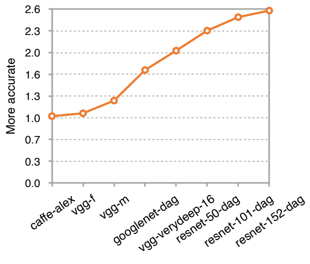
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition.* In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition.* In Proc. CVPR, 2016.

Accuracy

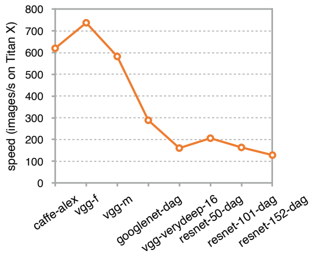
3 × more accurate in 3 years

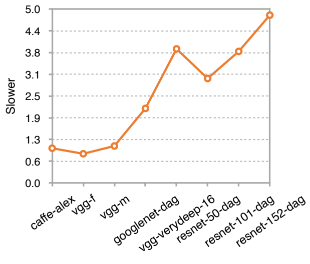




Speed

5 × slower





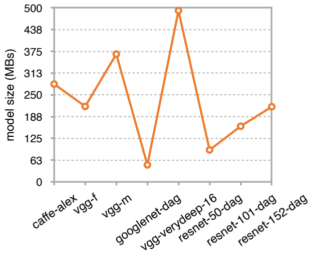
Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

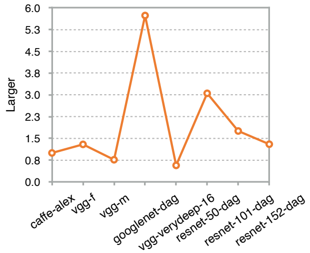
Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture

Model size

Num. of parameters is about the same

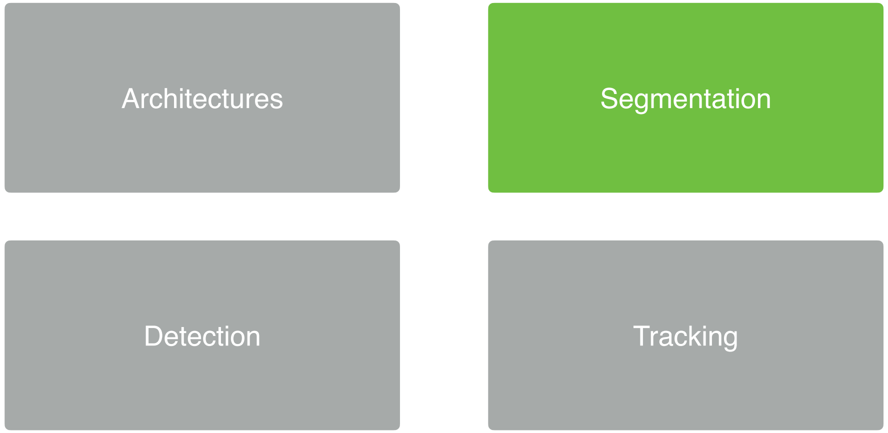




Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture



Architectures

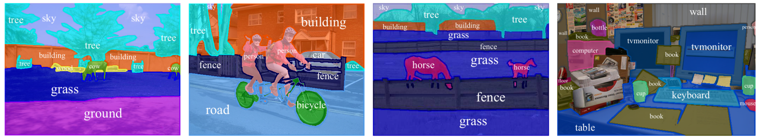
Segmentation

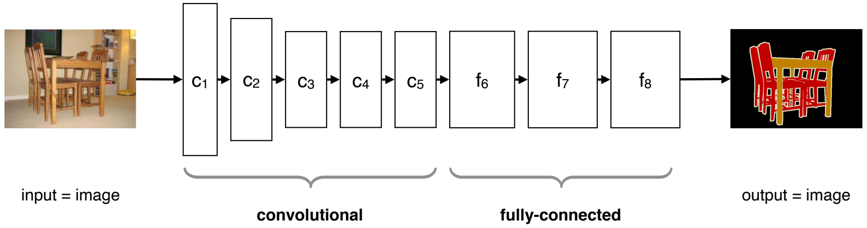
Detection

Tracking

Semantic image segmentation

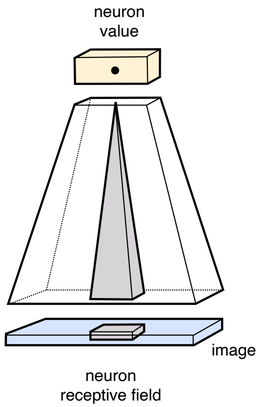
Label individual pixels





Receptive field

The part of the image looked at by a neuron



Receptive Field (RF) of a neuron

- The subset of the image affecting the value of a neuron

Small vs large RFs

- Small RF: spatially specific, but can only account for small visual structures
- Large RF: spatially a-specific, but can account for large visual structure

How to make the RF large

- Use large filters
- Chain several filters
- Interleave downsampling along the chain
  E.g. downsampling 2x increases the RF size 2x.

Convolutional vs fully connected layers

Comparing the receptive fields

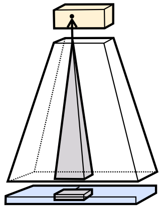
Convolutional layers

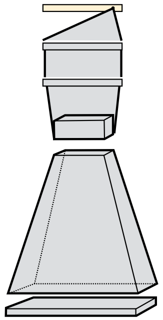
Neurons are spatially selective, can be used to localize things.

Fully connected layers

Neurons are global, do not characterize well position.

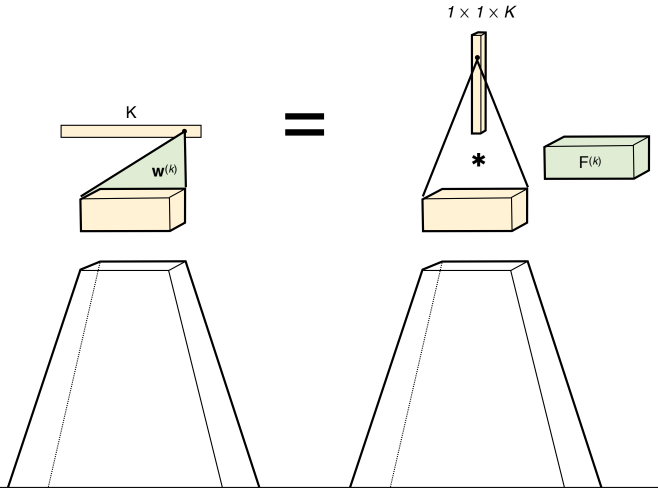
Which one is more useful for pixel level labelling?



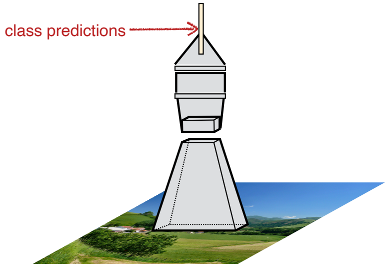


A fully connected layer is just a large filter

The filter support fills the entire input tensor

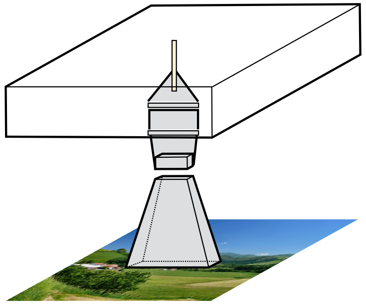


Fully-convolutional neural networks



J. Long, E. Shelhamer, and T. Darrell. *Fully convolutional models for semantic segmentation.* In Proc. CVPR, 2015

Fully-convolutional neural networks



Dense evaluation

- Apply the whole network convolutional
- Computes a vector of class probabilities at each pixel

Downsampling

- For efficiency, the input data is substantially down sampled in the network
- The output is fairly low resolution (e.g. 1/32 of original)

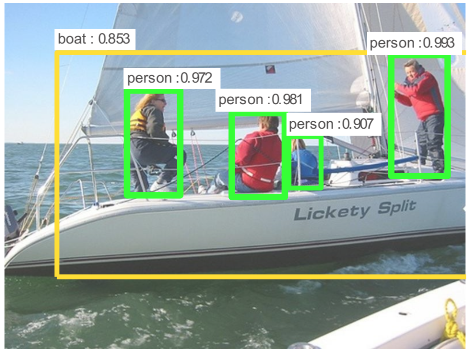
Architectures

Segmentation

Detection

Tracking

The object detection problem



The goal of **object detection** is to simultaneously classify, enumerate, and localise known object types in an image.

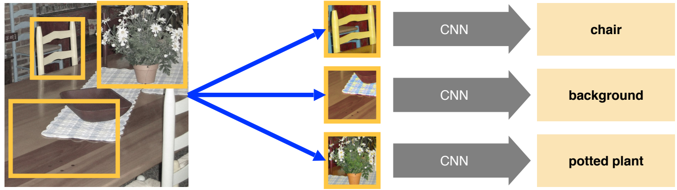
A key challenge is that the number of object instances is not known a priori.

Detections with CNNs

Region-based Convolutional Neural Network (R-CNN)

CNNs compute a fixed number of image features. A new computational mechanism is needed in order to detect a variable number of objects.

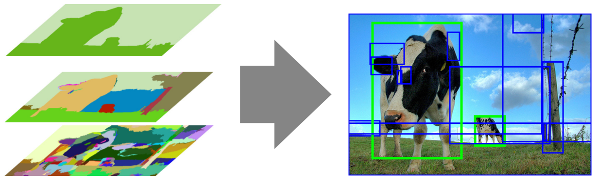
Region-based CNN (R-CNN) use a **region proposal algorithm** to extract a large number of potential object regions, and then a CNN to assess each one of them.



Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation
R. Girshick, J. Donahue, T. Darrell, J. Malik, CVPR 2014

Region proposal algorithm

Obtain a shortlist of regions that may contain objects



A **region proposal algorithm** produces a shortlist of regions that are likely to contain whole objects.

The *Selective Search* method by [van de Sande, Uijlings et al.]):

- Uses hierarchical segmentation based on colour uniformity and image edges.
- Produces about ~ 2000 regions / image with a > 95% probability of hitting any relevant object in the image.

From proposals to CNN features

Dilate, crop, reshape

Propose

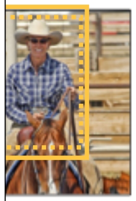
Dilate

Crop & scale
Anisotropic
227 x 227

A region proposal is slightly dilated to capture some visual context and then cropped and resized in order to be passed to a CNN.

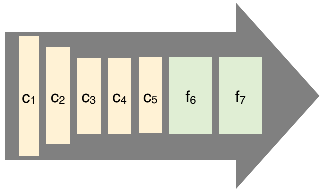
From proposals to CNN features

Evaluate CNN

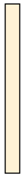


Dilate

Scale
Anisotropic
227 x 227



CNN features
Up to FC-7
AlexNet

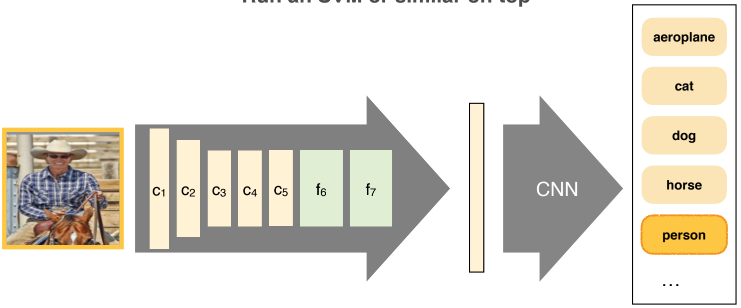


Feature vector
4096 D

The cropped and resize region is passed through a CNN to extract a corresponding **feature vector** (or image representation).

Classification of a region

Run an SVM or similar on top

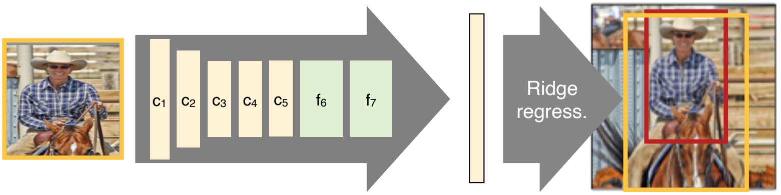


Scale	**CNN features**	**Feature vector**	**Label**
Anisotropic 227 x 227	Up to FC-7 AlexNet	4096 D	One out of *N*

The feature vector is then classified by means of a linear predictor (or a multi-layer perceptron). There are C + 1 possible object types, including “no object” (background).

Region adjustment

Bounding-box regression

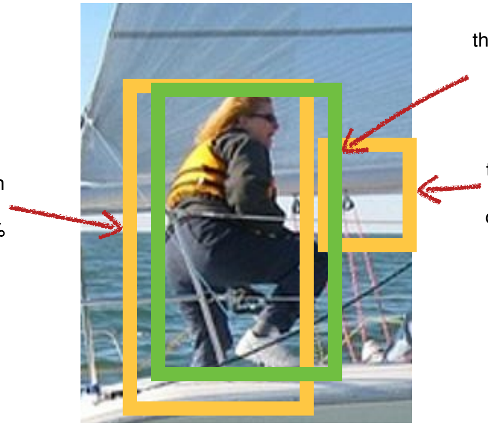


Scale	**CNN features**	**Feature vector**	**Box adjustment**
Anisotropic 227 x 227	Up to FC-7 AlexNet	4096 D	dx1, dx2, dy1, dy2

A second linear regression is used to **refine the bounding box** location. In the example, the person’s legs were not included in the proposal, but regression can fix this mistake.

Positive and negative training regions

Based on overlap with ground truth bounding box



a **positive**
training region

overlap > 70%

the **ground-truth**
region

a **negative**
training region

overlap < 30%

Ren, He, Girshick, & Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. NeurIPS 2015

R-CNN results on PASCAL VOC

At the time of introduction (2013)

Despite its conceptual simplicity, at the time of introduction R-CNN was substantially better than all existing methods.

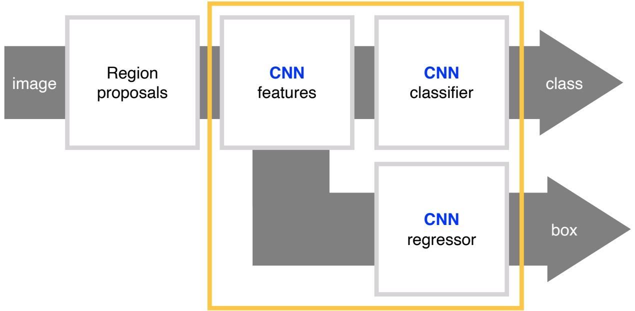
This is due to the power of the CNN classifier.

Importantly, the CNN is **pre-trained** on the ImageNet data (1M images) for classification (using only image-level labels), then **fine-tuned** on PASCAL VOC data (5K images) for object detection (using region-level labels).

	VOC 2007	VOC 2010
DPM v5 (Girshick et al. 2011)	33.7%	29.6%
UVA sel. search (Uijlings et al. 2013)		35.1%
Regionlets (Wang et al. 2013)	41.7%	39.7%
SegDPM (Fidler et al. 2013)		40.4%
R-CNN (TorontoNet)	54.2%	50.2%
R-CNN (TorontoNet) + bbox regression	58.5%	53.7%
R-CNN (VGG-VD)	62.1%	
R-CNN (ONet) + bbox regression	66.0%	62.9%

R-CNNs as a complex CNN

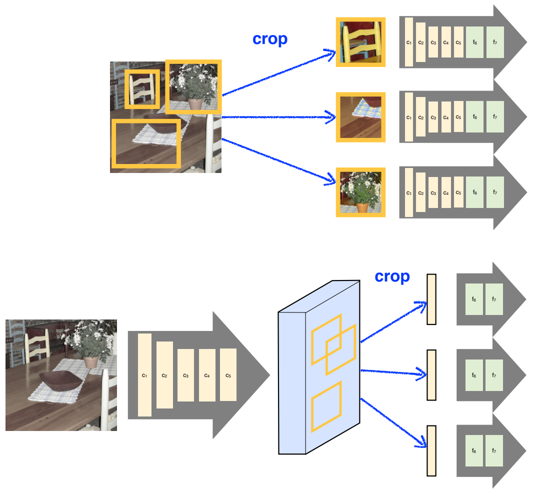
Integrate more of the blocks as CNN components



R-CNN can be improved substantially in three ways:

- By integrating all blocks in a end-to-end trainable CNN
- By accelerating region-specific computations
- By replacing region proposal generation with something better

Accelerating R-CNN



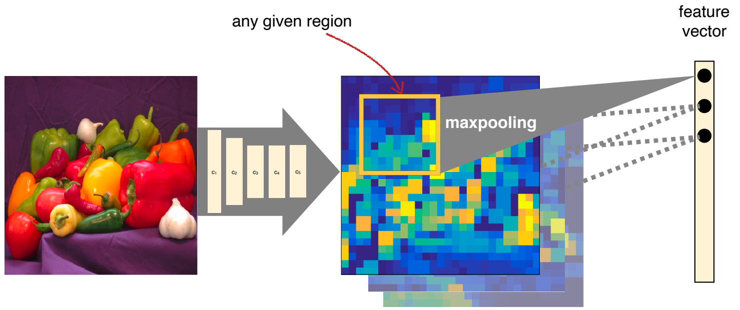
Problem: The fundamental bottleneck is evaluating the CNN from scratch for each image region.

Solution: compute all the convolutional features just once, and then *crop directly the resulting feature map*.

Only the fully-connected layers are evaluated for each region.

How: spatial pooling layer.

The Spatial Pooling (SP) layer



The **spatial pooling layer** (SP) max-pools the convolutional feature responses in a given region.

This can be used to extract many region-specific feature vectors by reusing the same convolutional features.

He, Zhang,Ren & Sun, “Spatial Pyramid Pooling (SPP) in Deep Convolutional Networks for Visual Recognition”, ECCV 2014

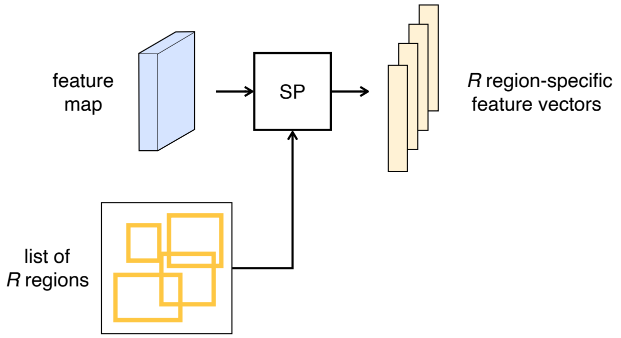
The Spatial Pooling (SP) layer

As a building block

The SP layer extracts a feature vector for each of the R regions.

The output are thus R tensor of size $1 \times 1 \times C$.

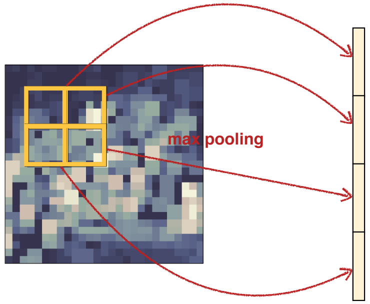
Alternatively, this can be seen as a single $1 \times 1 \times C \times R$ tensor.



He, Zhang,Ren & Sun, “Spatial Pyramid Pooling (SPP) in Deep Convolutional Networks for Visual Recognition”, ECCV 2014

The Spatial **Pyramid** Pooling Layer

SP with multiple subdivisions

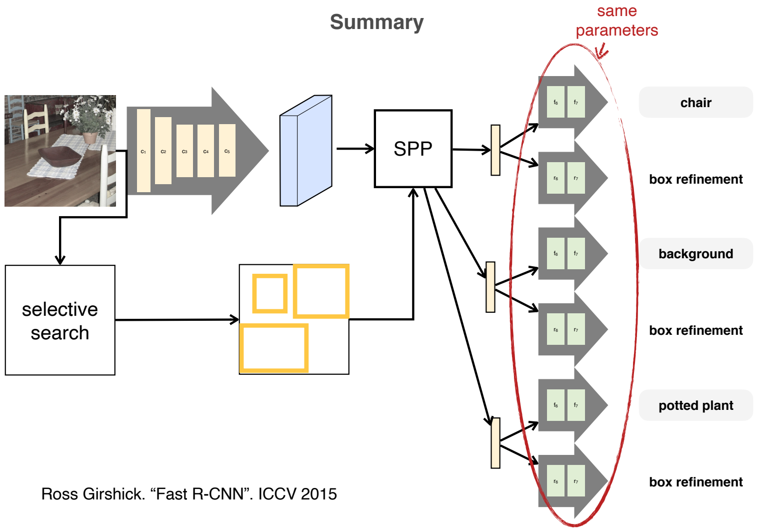


SPP is similar to SP, but pools features in the tiles of a **grid-like subdivision** of the region.

The resulting feature vector **captures the spatial layout** of the original region.

Fast R-CNN

Summary



Ross Girshick. “Fast R-CNN”. ICCV 2015

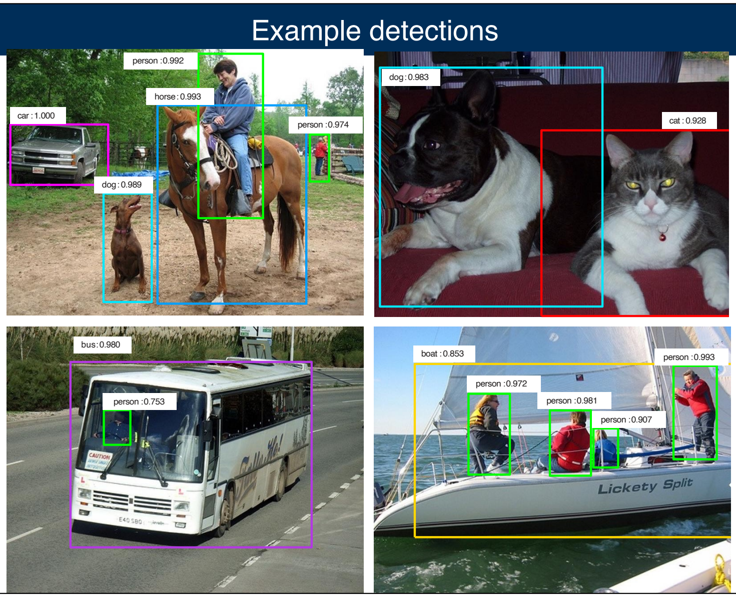
Fast and Faster R-CNN performance

Both faster and better!

Detection mAP on PASCAL VOC 2007, with VGG-16 pre-trained on ImageNet.

Method	Time / image	mAP (%)
R-CNN	~50s	66.0
Fast R-CNN	~2s	66.9
Faster R-CNN	198ms	69.9

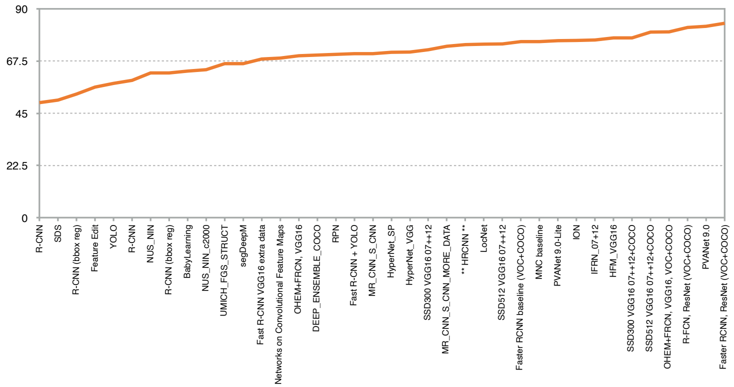
Example detections



PASCAL VOC Leaderboards

Detection challenge (comp4: train on own data)

http://tinyurl.com/h7uzkov



2014 4 × improvement in accuracy 2016

Architectures

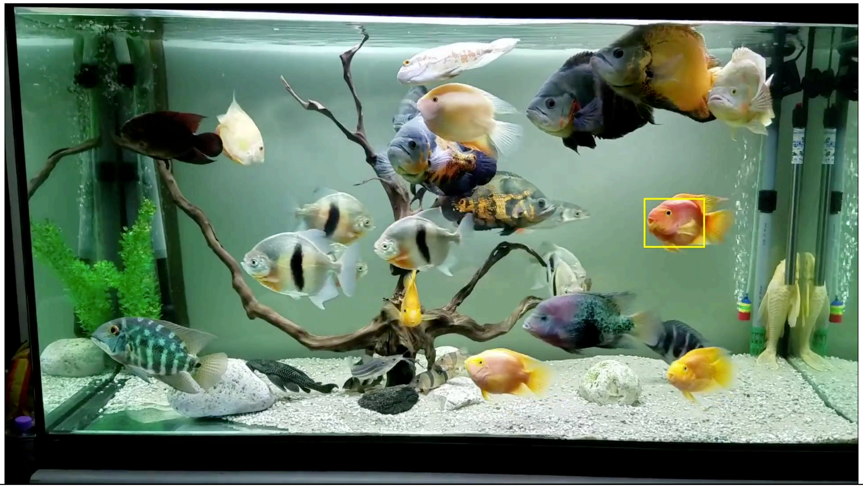
Segmentation

Detection

Tracking

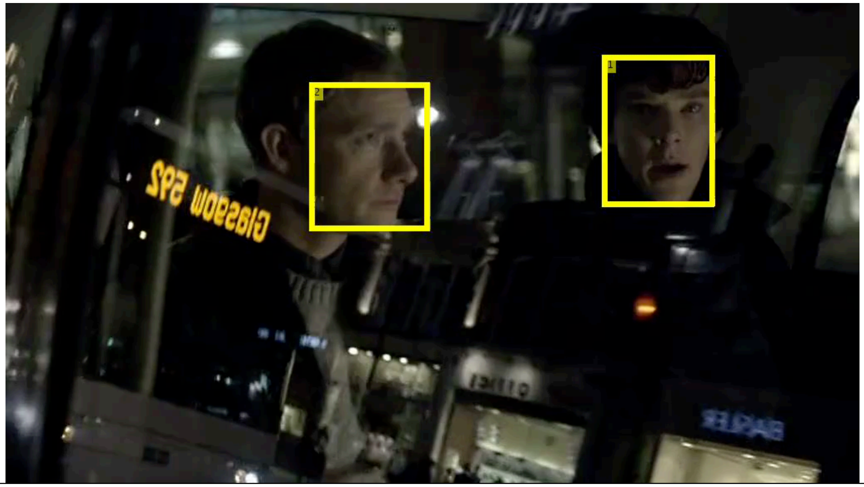
Tracking 1/2: select & track

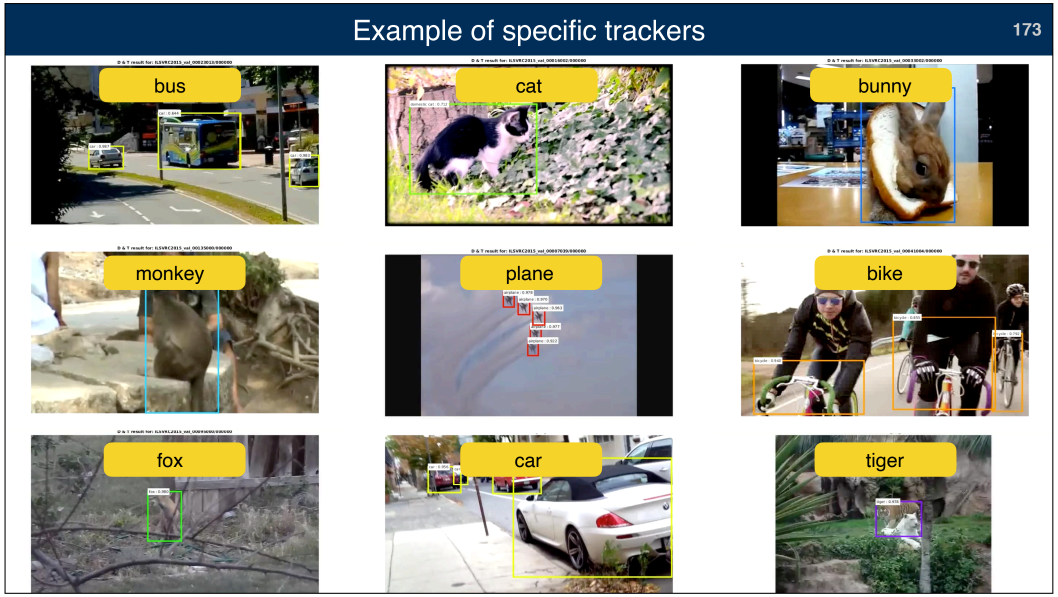
Draw a bounding box first, then track it automatically



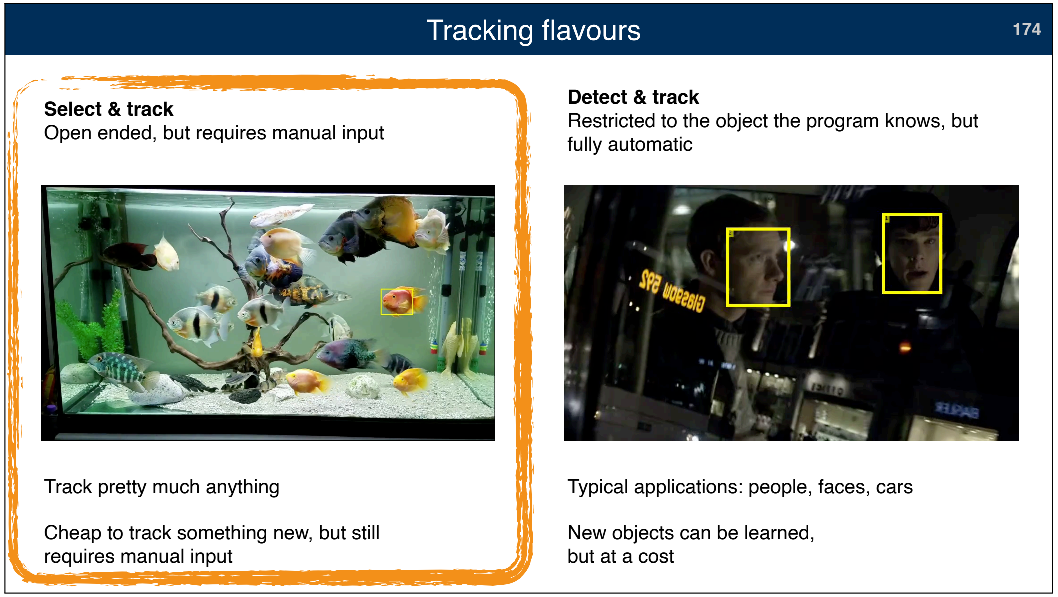
Tracking 2/2: detect & track

Track pre-programmed objects (e.g. faces) fully automatically (no manual selection required)





Example of specific trackers



Tracking flavours

Select & track
Open ended, but requires manual input

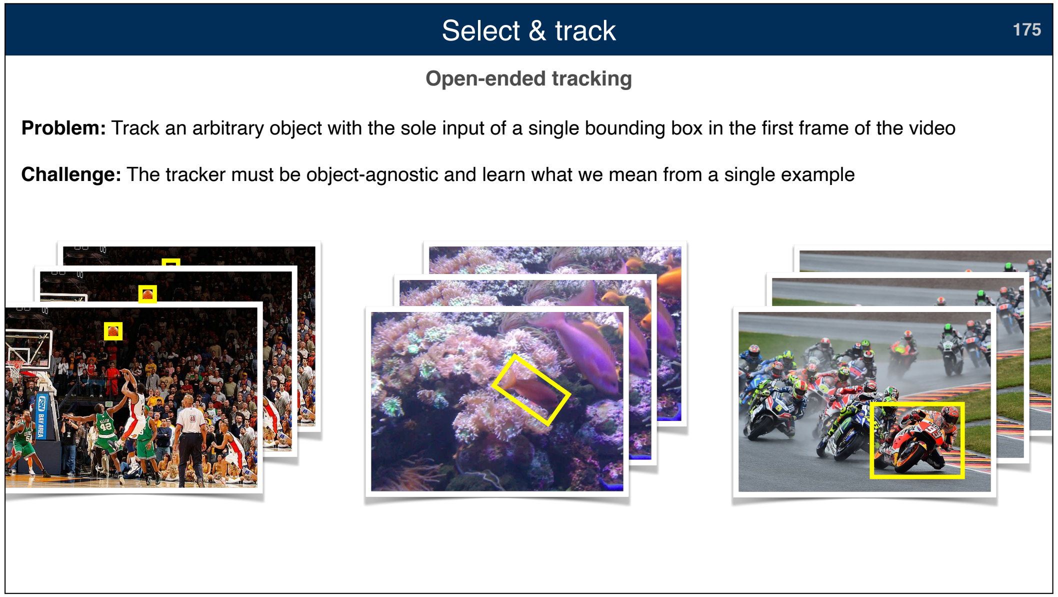
Track pretty much anything

Cheap to track something new, but still requires manual input

Detect & track
Restricted to the object the program knows, but fully automatic

Typical applications: people, faces, cars

New objects can be learned, but at a cost

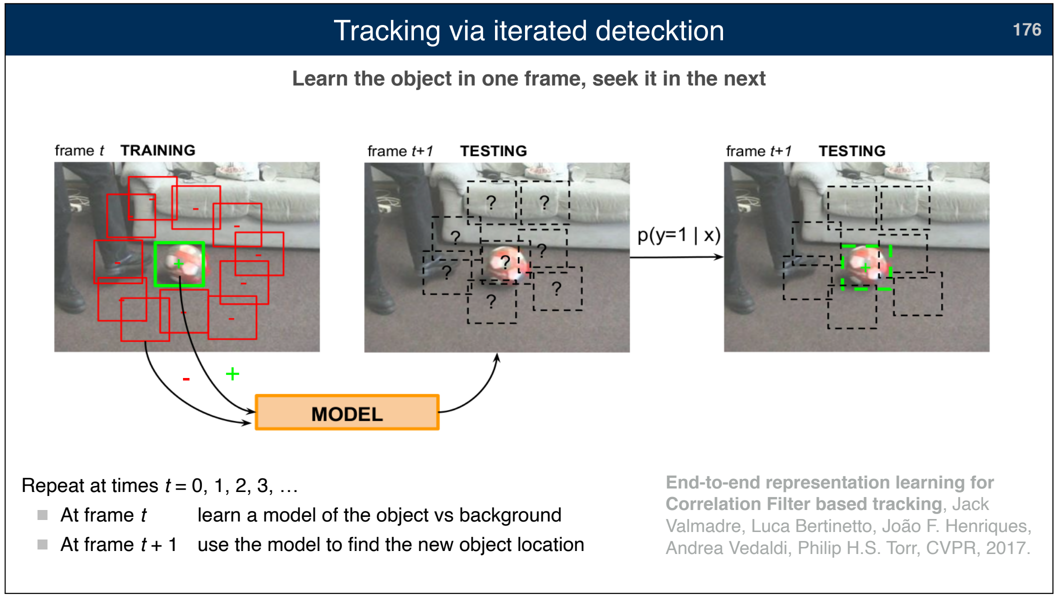


Select & track

Open-ended tracking

Problem: Track an arbitrary object with the sole input of a single bounding box in the first frame of the video

Challenge: The tracker must be object-agnostic and learn what we mean from a single example



Tracking via iterated detecktion

Learn the object in one frame, seek it in the next

Repeat at times $t = 0, 1, 2, 3, \ldots$

- At frame t learn a model of the object vs background
- At frame $t + 1$ use the model to find the new object location

End-to-end representation learning for Correlation Filter based tracking, Jack Valmadre, Luca Bertinetto, João F. Henriques, Andrea Vedaldi, Philip H.S. Torr, CVPR, 2017.

How our tracker works

Describe and match

Descriptor computation

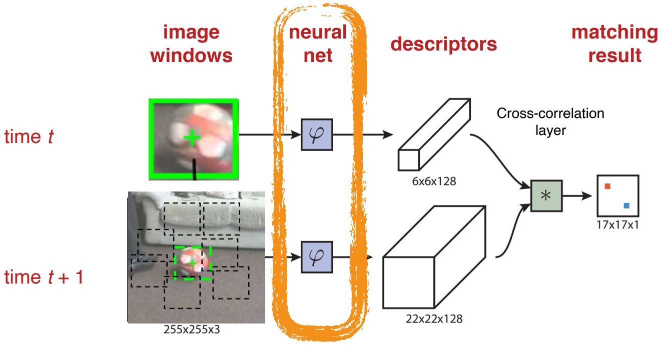
A **neural network** ϕ maps each image window to a visual descriptor

Two images of different sizes

- small: exemplar at time *t*
- big: search area at time *t* + 1

Descriptor matching

Computes the descriptor similarity at all translated sub-windows



Training data

ImageNet Video

Official task is object detection from video - can be easily adapted to arbitrary object tracking

Almost `4,500 videos` and `1,200,000 bounding boxes`!

30 classes: mostly animals (~75%) and some vehicles (~25%)

C18 Computer Vision

Recap

Prof. Andrea Vedaldi (4 lectures)

- Lecture 1: Matching, indexing, and retrieval
- Lecture 2: Convolutional neural networks
- Lecture 3: Backpropagation and automated differentiation
- Lecture 4: Applications

Prof. Victor Prisacariu (4 lectures)

- 3D vision