AIMS Big Data

Lecture 3: Deep Learning

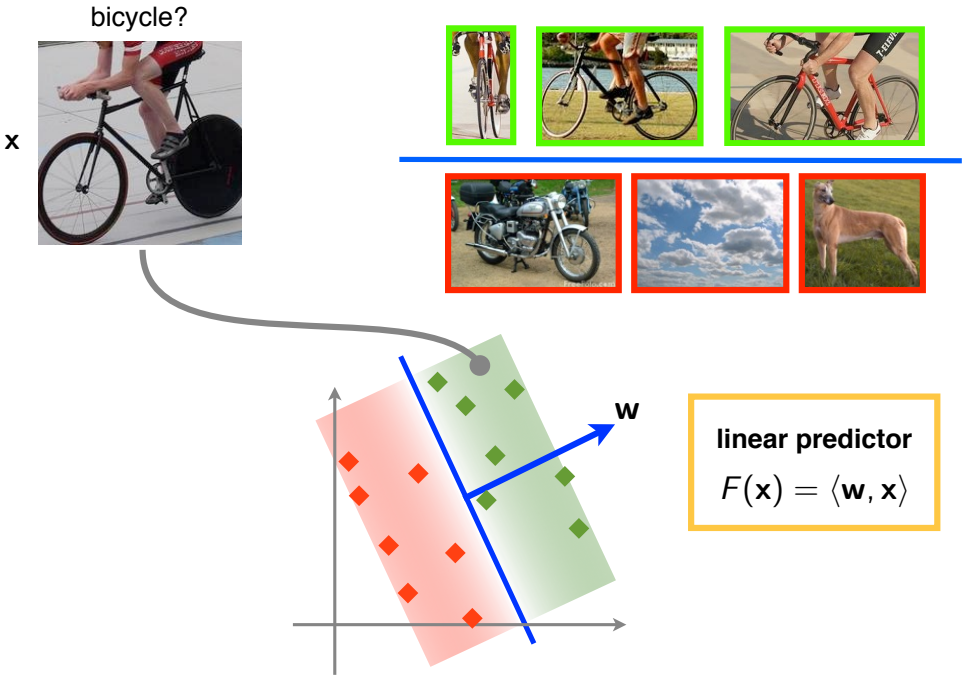
Andrea Vedaldi

For slides and up-to-date information:

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

UNIVERSITY OF OXFORD

Linear predictor

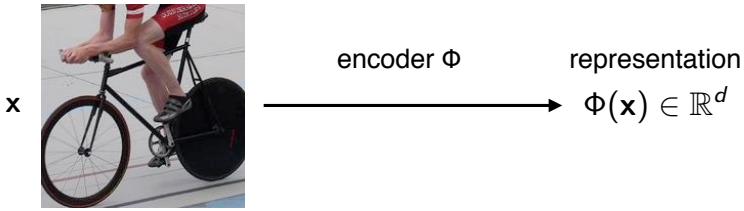


linear predictor

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

Data representations

Using linear predictors on non-vectorial data

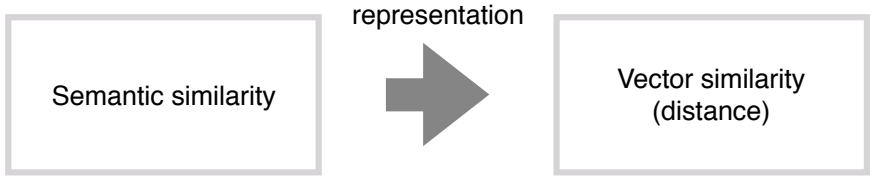


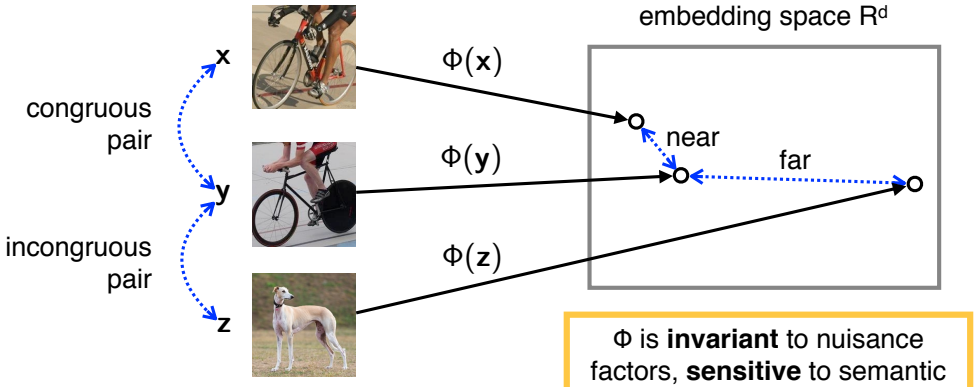
An **encoder** maps the data into a **vectorial representation**

Allows linear predictors to be applied to images, text, sound, videos, …

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

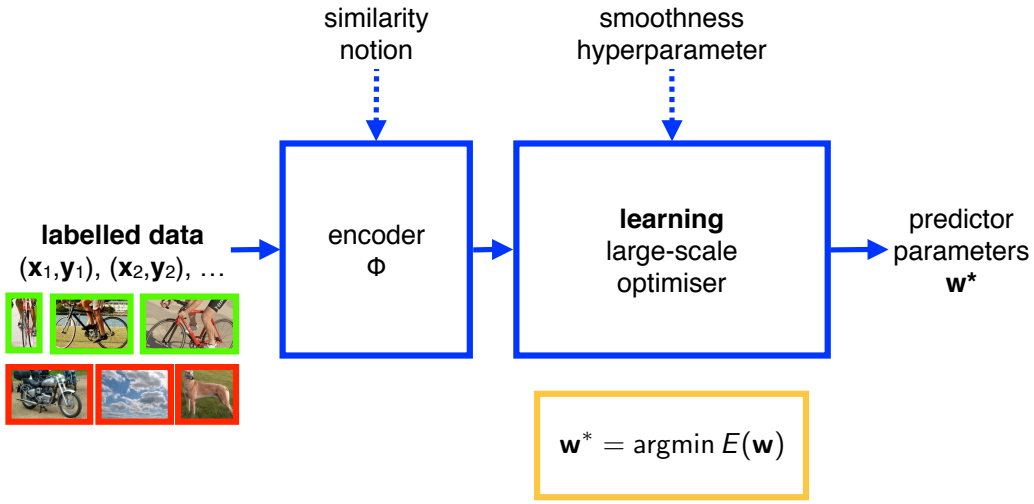
Meaningful representation



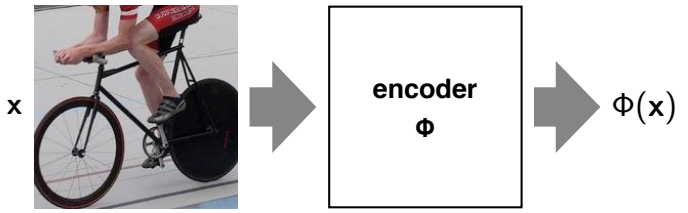


Φ is **invariant** to nuisance factors, **sensitive** to semantic variations

Learning predictors



Good representations

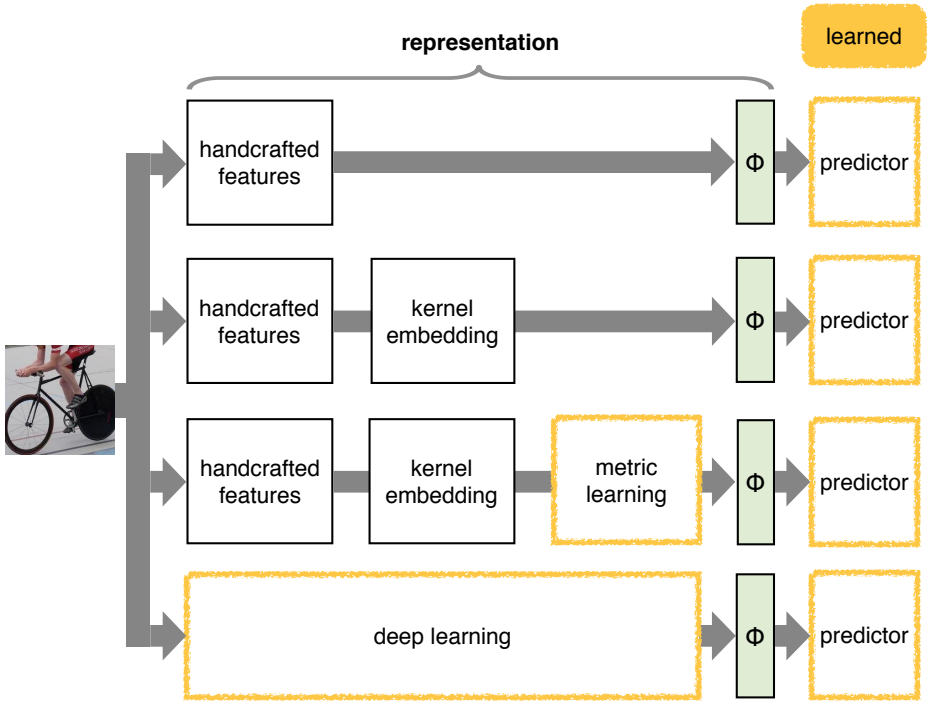


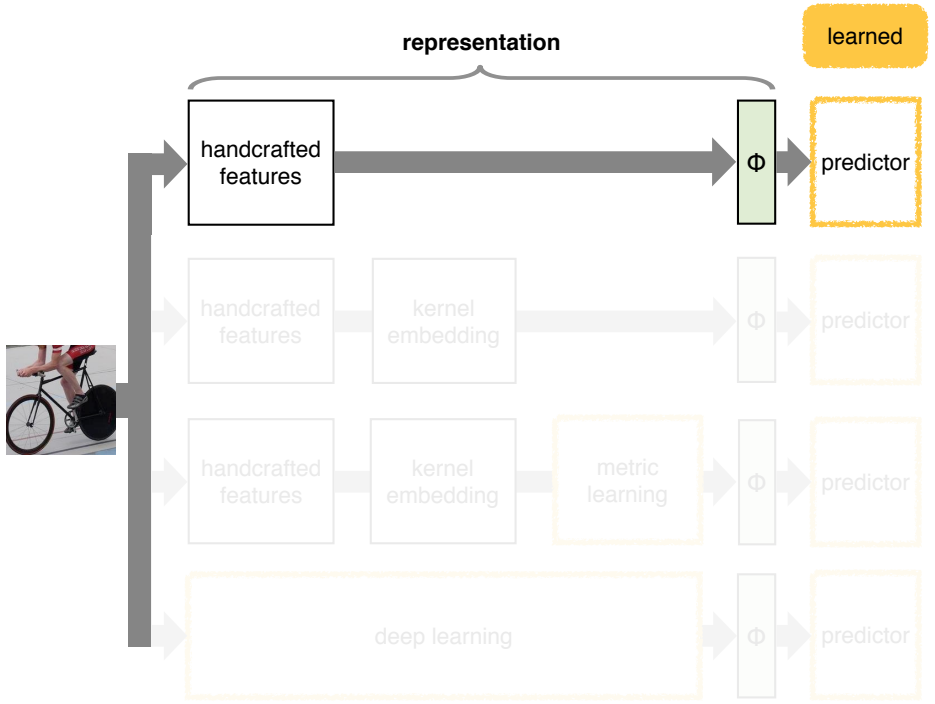
Main desiderata

- **Powerful:** meaningful similarity / untangles factors
- **Cheap:** fast to evaluate (can be computed on the fly)
- **Compact:** small code (takes little RAM, disk, IO)

Others

- Easy to learn (when not hand-crafted)
- Easy to implement

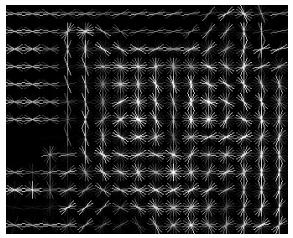


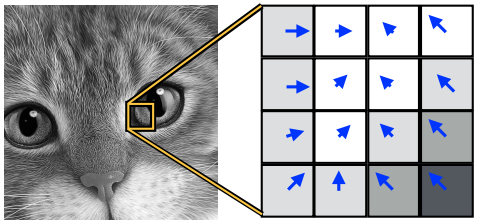


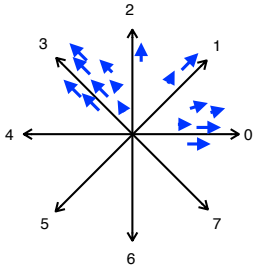
Histogram of Oriented Gradients

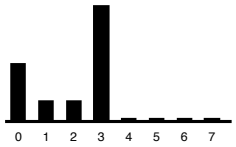
[Lowe 1999, Dalal & Triggs 2005]

HOG captures the local gradient (edge) orientations in the image





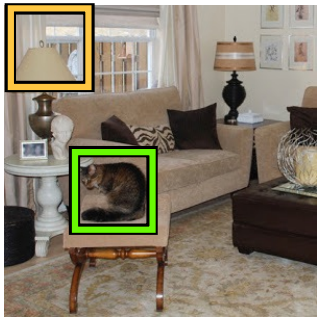


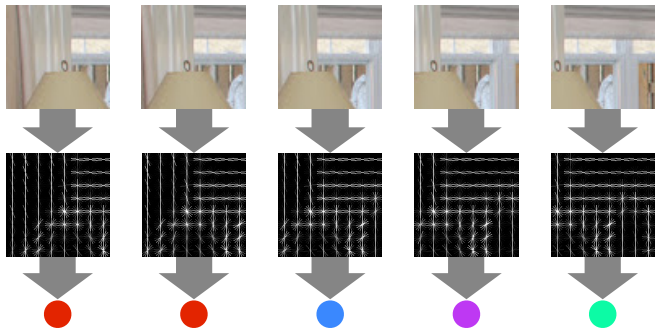


+ block l^2 normalisation

Bag of visual words

[Sivic & Zisserman 2003, Csurka *et al.* 2004, Nowak *et al.* 2006]

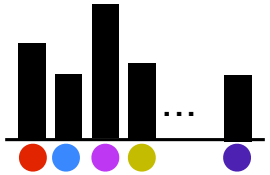




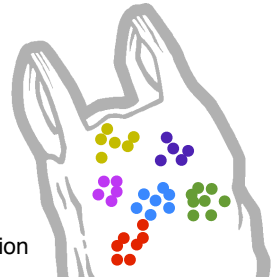
BoVW construction

1. Extract local descriptor densely
2. Quantise descriptors
3. Form histogram

2. **Discards spatial information**



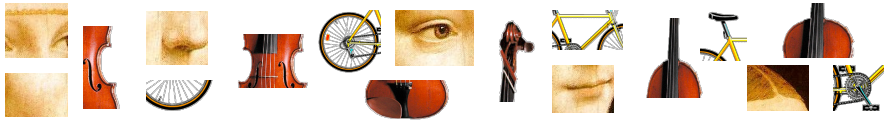
+ l^2 normalisation

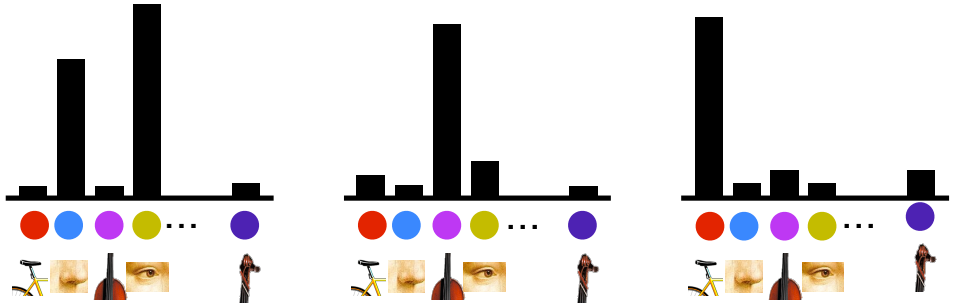


BoVW intuition

Discarding spatial information gives **lots of invariance**

Visual words represent **"iconic" image fragments**

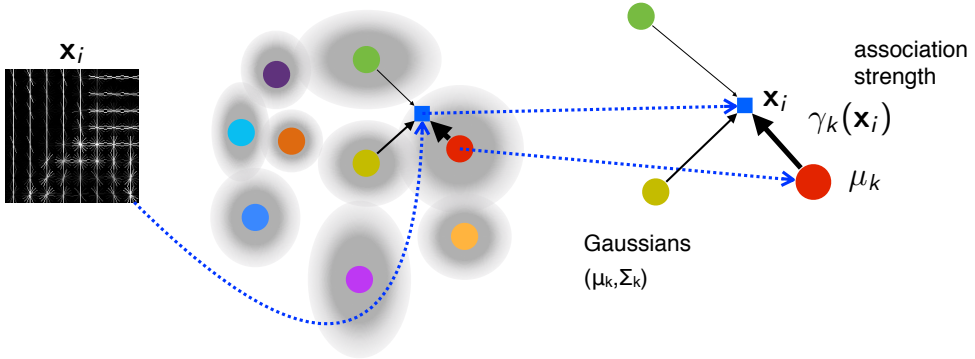




person | musical instrument | bike

Fisher Vector (FV)

[Perronnin et al. ECCV 201, Sharma Hussain Jurie ECCV 2010, Sanchez et al. 2103]



FV encoding $\Phi =$

+ sqrt-l^2 normalisation

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{u}_1 \\ \mathbf{v}_2 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{v}_K \\ \mathbf{u}_K \end{bmatrix}$$

first and second order statistics

$$\mathbf{v}_k = \frac{1}{M\sqrt{\pi_k}} \sum_{i=1}^{M} \gamma_k(\mathbf{x}_i) \frac{\mathbf{x}_i - \mu_k}{\sigma_i}$$

$$\mathbf{u}_k = \frac{1}{M\sqrt{2\pi_k}} \sum_{i=1}^{M} \gamma_k(\mathbf{x}_i) \left(\frac{\mathbf{x}_i - \mu_k}{\sigma_i} - 1 \right)^2$$

Some fundamental ideas

Local and translation invariant operators

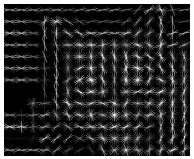
gradients, filters, visual words

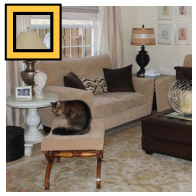
Untangling

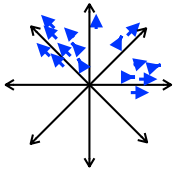
sparsity, quantisation

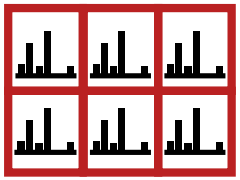
Pooling

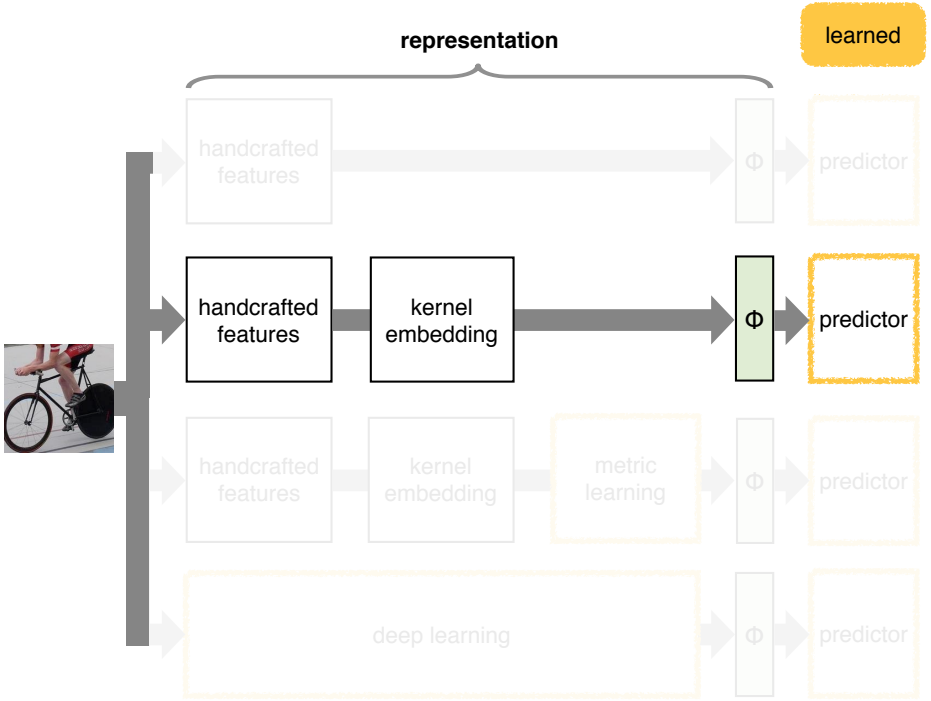
max, sum, spatial pooling







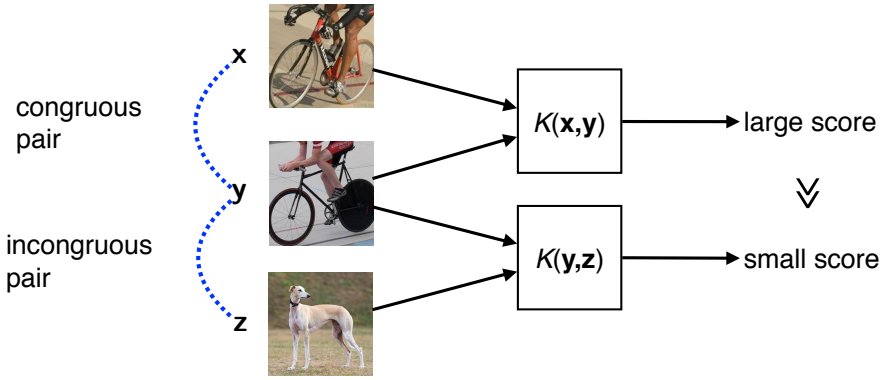




Kernels

A **kernel** *directly* encodes a notion of *data similarity*

$$K : (\mathbf{x}, \mathbf{y}) \mapsto \mathbb{R}$$



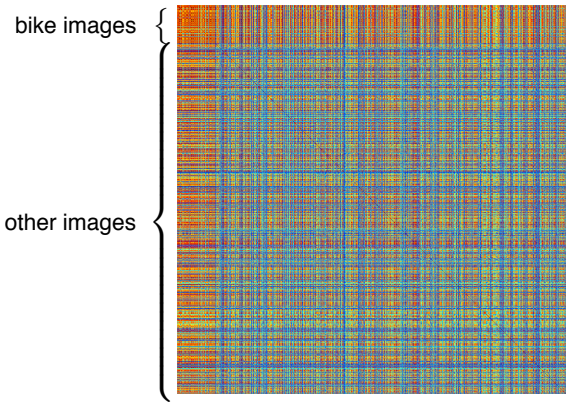
Similarity and kernels

Recall: the encoder ϕ(I) should embody a useful notion of similarity

Similarity can be measured by the inner product or kernel $\langle \phi(I), \phi(I') \rangle$

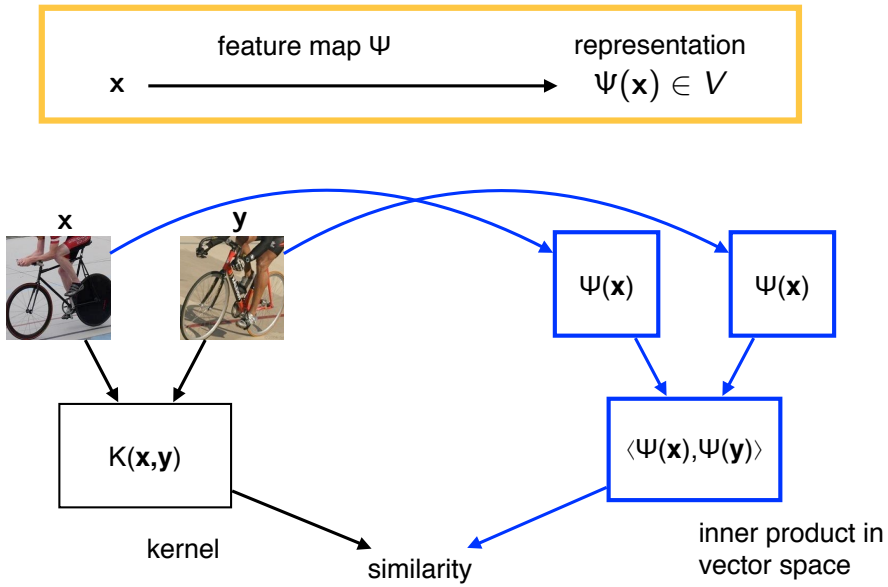
linear kernel

$\langle \Phi(I), \Phi(I') \rangle$



Kernel as **representations**

Positive definite kernel = inner product of **feature vectors**



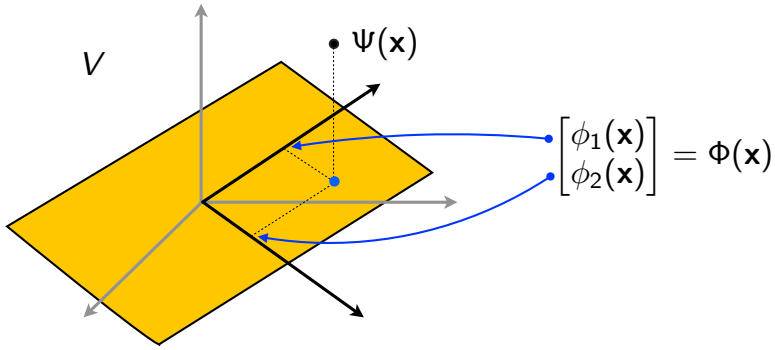
Finding kernel maps

- **Kernel maps**
  - often infinite dimensional
  - used implicitly (kernel trick)
  - theoretical

$$K(\mathbf{x},\mathbf{y}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle$$
$$\Psi(\mathbf{x}) \in V$$

- **Explicit kernel maps**
  - finite dimensional approximation
  - used explicitly
  - practical

$$K(\mathbf{x},\mathbf{y}) \approx \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$
$$\Phi(\mathbf{x}) \in \mathbb{R}^d$$



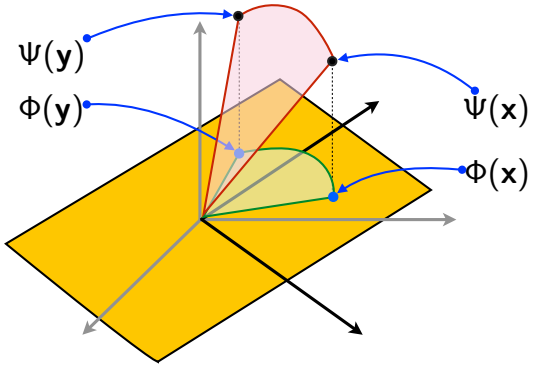
Finding kernel maps

- **Kernel maps**
  - often infinite dimensional
  - used implicitly (kernel trick)
  - theoretical

$$K(\mathbf{x},\mathbf{y}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle$$
$$\Psi(\mathbf{x}) \in V$$

- **Explicit kernel maps**
  - finite dimensional approximation
  - used explicitly
  - practical

$$K(\mathbf{x},\mathbf{y}) \approx \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$
$$\Phi(\mathbf{x}) \in \mathbb{R}^d$$



Example: Chi^2 map

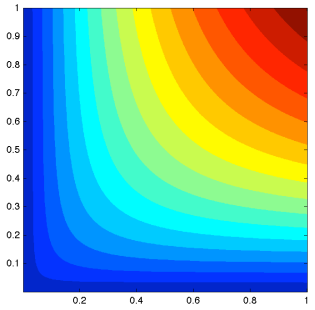
MATLAB code for Chi2 kernel

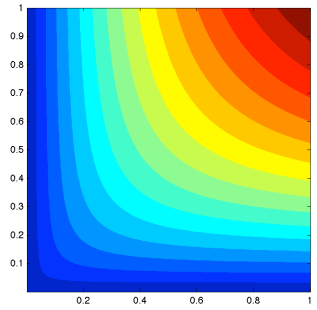
```
x = .01:.01:1 ;
for i = 1:100
  for j = 1:100
    K(i,j) = ...
      2*x(i)*x(j)/(x(i)+x(j));
  end
end
```

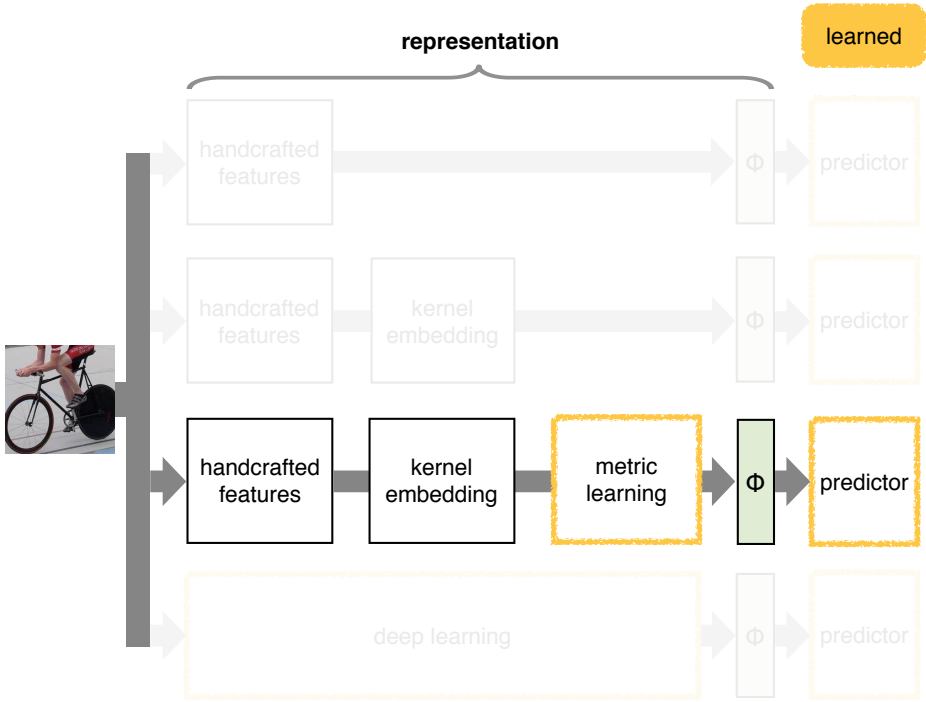
With the hom. kernel feature map

```
x = .01:.01:1 ;
psi = vl_homkermap(x,1) ;
K = psi'*psi ;
```

VLFeat Toolbox
http://www.vlfeat.org







Learning to compare

For a thorough review: [Weinberger Saul JMLR 2009]

Goal

- compare (rather than classify) objects **x**, **y**
- formally, learn a distance $d^2(\mathbf{x},\mathbf{y})$

Desiderata

- if **x** and **y** are *congruous* $\Longrightarrow$ small distance
- if **x** and **y** are *incongruous* $\Longrightarrow$ large distance

Parametrisation of the distance

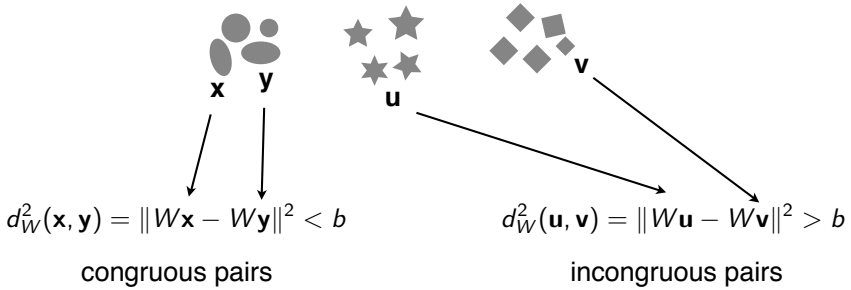
Euclidean distance + linear projection W

$$d_W^2(\mathbf{x},\mathbf{y}) = \|W\mathbf{x} - W\mathbf{y}\|^2$$

Classification-like constraints

For all object pairs **x**, **y**

- congruous $\Longrightarrow$ distance **smaller** than threshold - margin
- incongruous $\Longrightarrow$ distance **larger** than threshold + margin



$$d_W^2(\mathbf{x},\mathbf{y}) < b-1, \qquad d_W^2(\mathbf{u},\mathbf{v}) > b+1$$

Learning formulation

$$\min_{W,b} \mathcal{R}(W) + \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{P}} \max\{0, 1-b+d_W^2(\mathbf{x},\mathbf{y})\} + \sum_{(\mathbf{u},\mathbf{v})\in\mathcal{N}} \max\{0, 1+b-d_W^2(\mathbf{u},\mathbf{v}))\}$$

Input: training data

- congruous pairs $\mathcal{P}$ (i.e., positive)
- incongruous pairs $\mathcal{N}$ (i.e., negative)

Input: regulariser $\mathcal{R}(W)$

- controls which type of solution is found
- may induce smoothness, sparsity, group-sparsity, low rank

Output: projection matrix W

Algorithm and variants

- Convex + sparsity: regularized dual averaging
- Non-convex + fixed dimensionality: stochastic gradient descent

Compare & compress

Euclidean distance

$$d^2_W(\mathbf{x}, \mathbf{y}) = \|W\mathbf{x} - W\mathbf{y}\|^2$$

+

linear projection

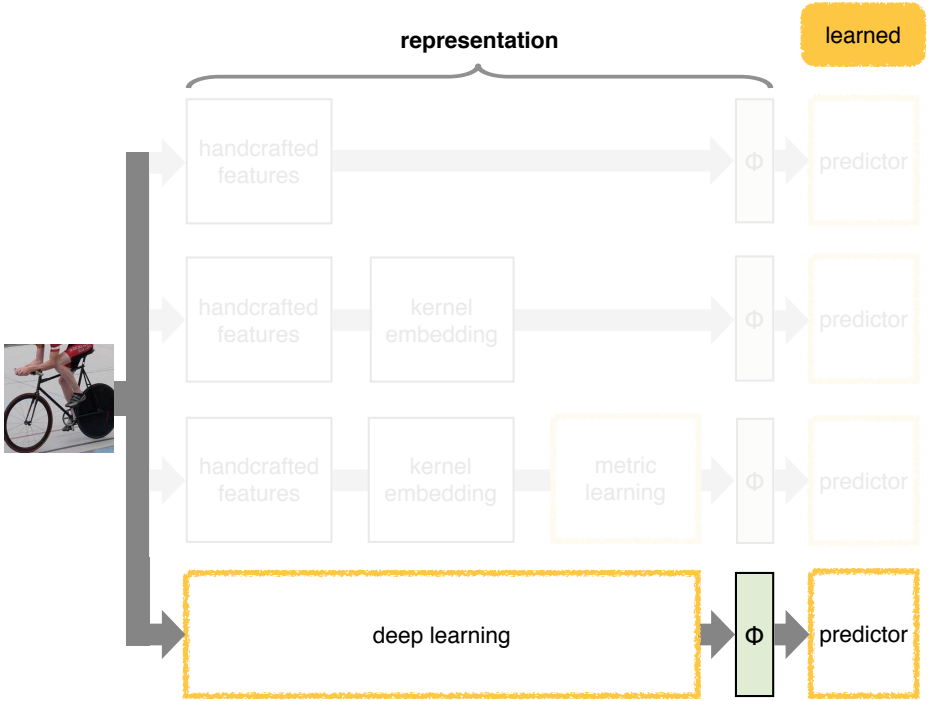
$$\mathbf{x} \in \mathbf{R}^n \xrightarrow{W \in \mathbf{R}^{m \times n}} \bar{\mathbf{x}} = W\mathbf{x} \in \mathbf{R}^m$$

W improves the data separation (= learns a meaningful similarity)

W can also **reduce the data dimensionality**

- simply pick m ≪ n

$$\bar{\mathbf{x}} = \boxed{W}\ \mathbf{x}$$



Deep learning overview

Neural networks

- Perceptron: as a classifier and regressor
- Multi-class perceptron and softmax
- Deeper: multi-layer perceptron

Deep architectures

- Discovery of oriented cells in the visual cortex
- Linear convolution and filter banks
- Gating functions
- Local normalisation
- Downsampling and pooling

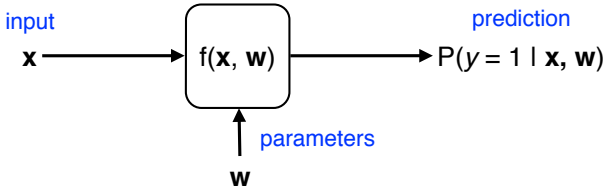
Learning deep architectures

- Autoencoders
- Convolutional neural networks

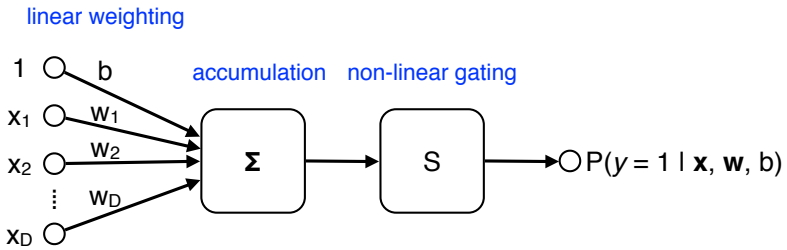
Perceptron

[Rosenblatt 57]

The goal is estimating the posterior probability of the binary label y of a vector $\mathbf{x}$:



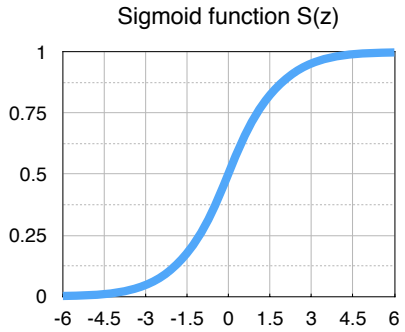
Perceptron: linear weighting + sum + non-linear gating:



Activation function (sigmoid)

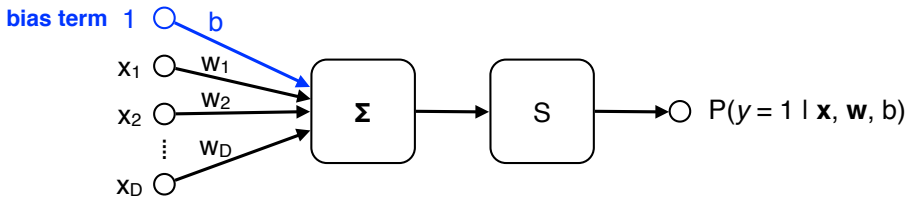
The gating function is a *sigmoid*.

It converts the range (-∞, +∞) into probability values (0, 1).



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

Functional form



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

Perceptron steps:

1. Map a vector **x** to a scalar score by an affine projection (**w**, b)
2. Transform the score monotonically but non-linearly by the sigmoid *S*()

Fitting to data

Training data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$

Treat as i.i.d. and compute the log-likelihood of the labels

- Likelihood that $y_i = 1$:

$$P(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = f(\mathbf{x}_i, \mathbf{w})$$

- Likelihood that $y_i = 1$ or 0:

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

- Negative log-likelihood

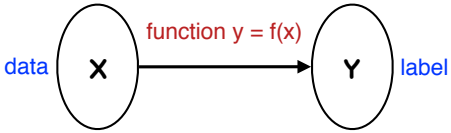
$$-\log P(y_i | \mathbf{x}_i, \mathbf{w}) = -y_i \log f(\mathbf{x}_i, \mathbf{w}) - (1 - y_i) \log(1 - f(\mathbf{x}_i, \mathbf{w}))$$

- Average over data points to obtain an **objective function** to minimise:

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

As a regressor

The perceptron can also be seen as a way of encoding a function from data **X** to labels **Y**



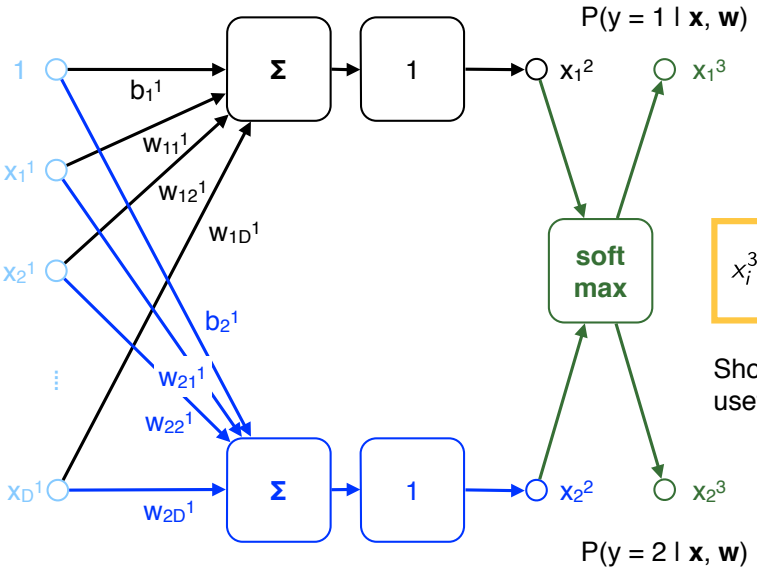
$$\mathbf{x} \longrightarrow y = S(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

Then learning the perceptron can be seen as fitting the function to data

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

Multi-class perceptron

Introducing the softmax layer

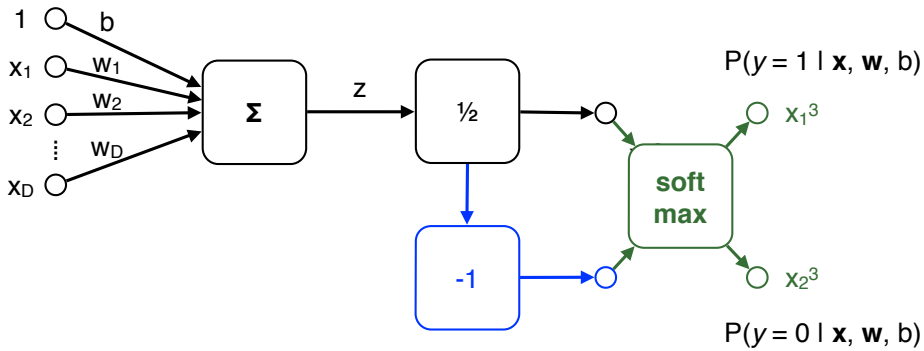


$$x_i^3 = \frac{e^{x_i^2}}{e^{x_1^2} + e^{x_2^2}}$$

Shown for 2-classes, useful for 3 or more

Softmax = sigmoid for 2 classes

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



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

Multi-class perceptron

Fitting to data

Training data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$

Negative log-likelihood of class $y_i = c$:

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

Objective function:

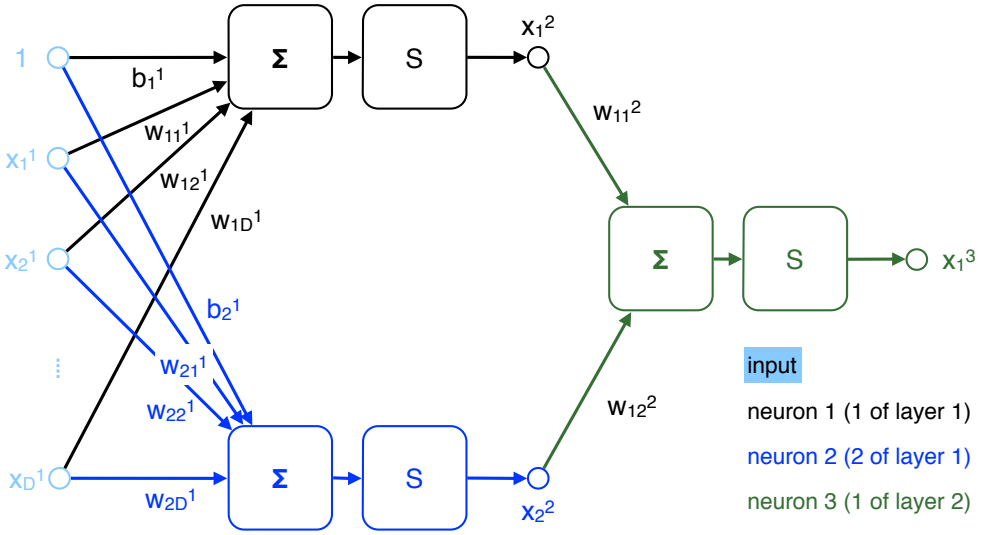
$$E(W) = \frac{1}{N} \sum_{i=1}^{N} \left(-\mathbf{w}_{y_i} \top \mathbf{x}_i - b_{y_i} + \log \sum_{q=1}^{C} e^{\mathbf{w}_q^\top \mathbf{x}_i + b_q} \right)$$

This is also the **cross entropy** $H(Q_i, P_i)$ between the

- empirical distributions $Q_i(y_i)$ and
- predicted distributions $P_i(y_i) = P(y_i \mid \mathbf{x}_i, \mathbf{w}, b)$

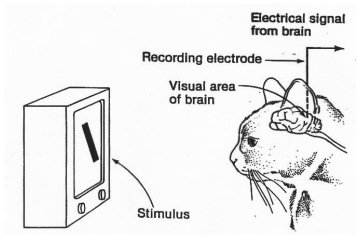
Multi-layer perceptron (MLP)

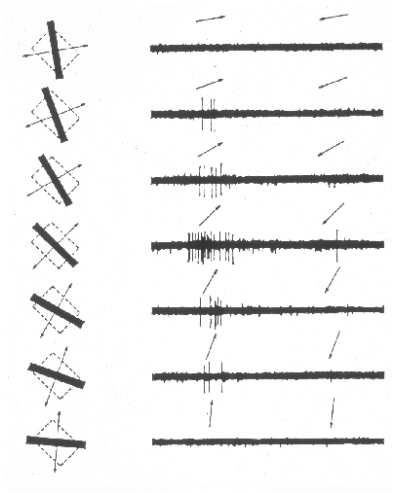
Deep architectures

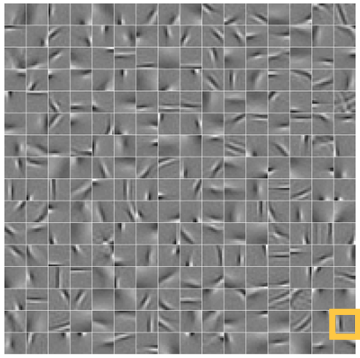


Discovery of oriented cells in the visual cortex

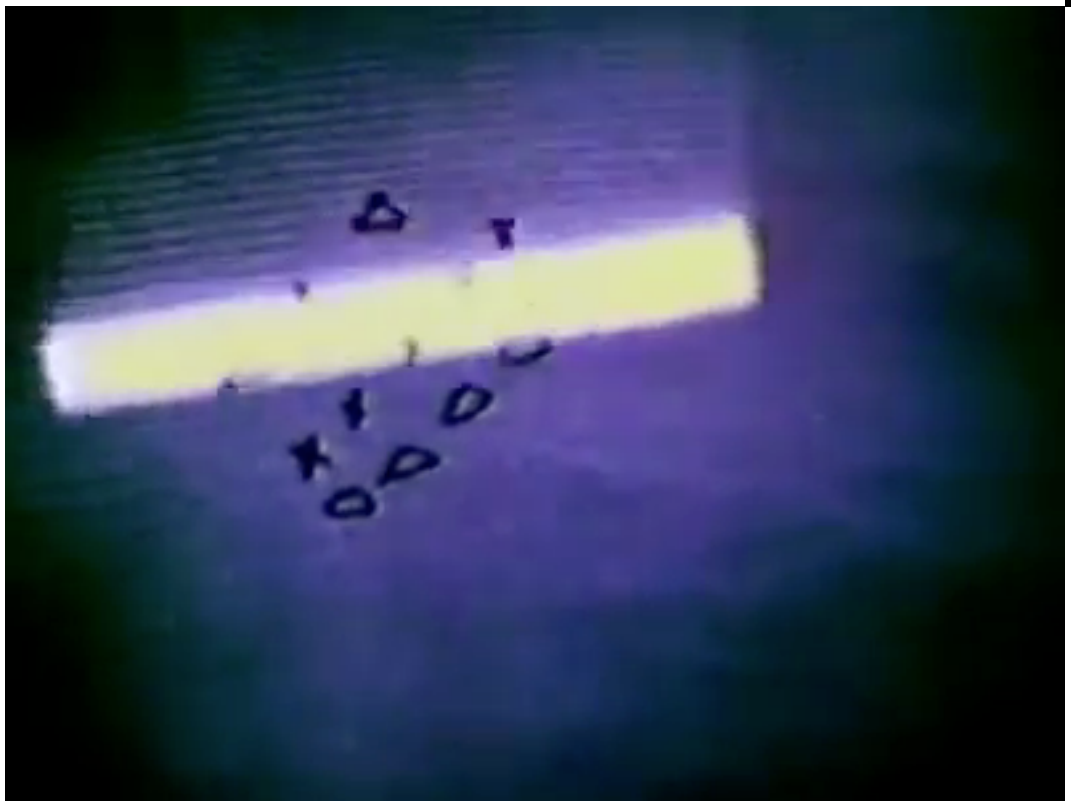
[Hubel and Wiesel 59]





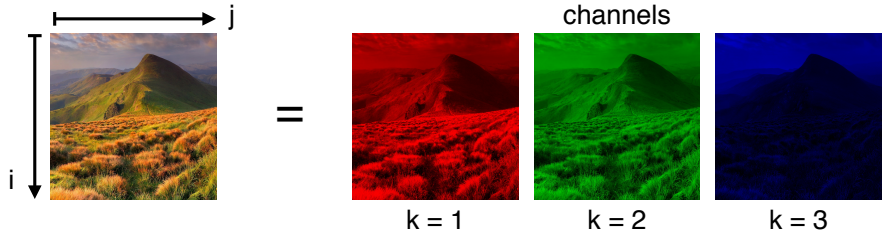


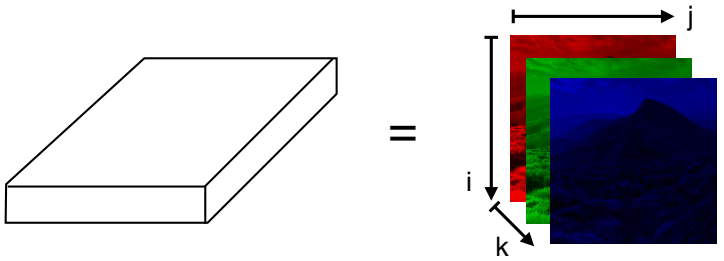
oriented filter



A signal processing notation

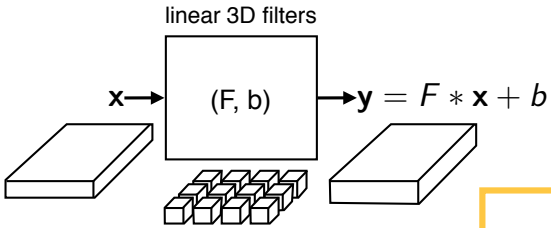
Data and intermediate neural computations are **discrete vector fields**





Linear convolution

As a filter bank



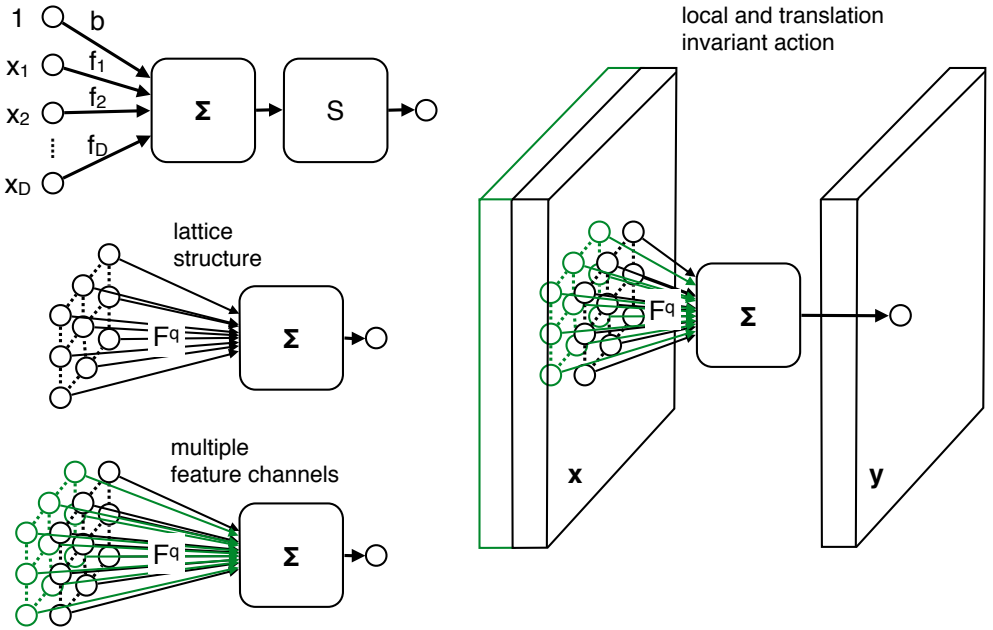
$$y_{ijq} = b_q + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{k=1}^{K} x_{u+i,v+j,k} f_{u,v,k,q}$$

Linear, translation invariant, local:

- Input **x** = H × W × K array
- Filter bank F = H' × W' × K × Q array
- Output **y** = (H - H' + 1) × (W - W' + 1) × Q array

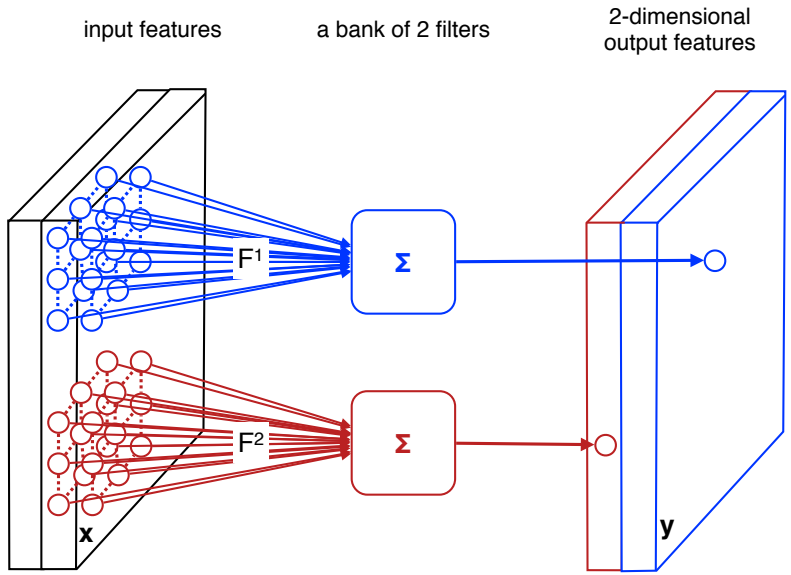
Linear convolution

As a neural network



Linear convolution

As a neural network



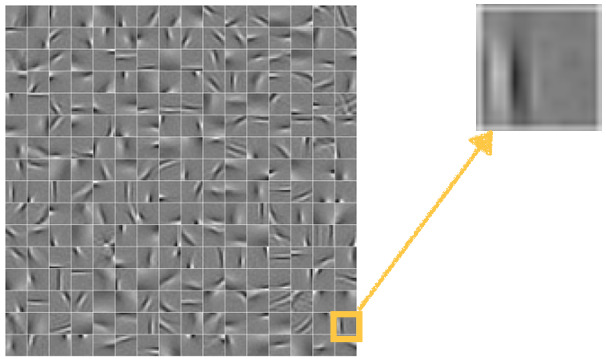
Linear convolution

Filter bank example

A bank of 256 filters (learned from data)

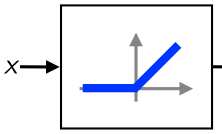
Each filter is 1D (it applies to a grayscale image)

Each filter is 16 × 16 pixels



Gating functions

Component-wise non-linearity



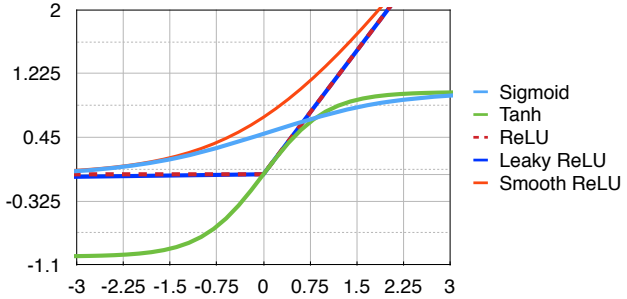
$$y = \frac{1}{1+e^{-x}} \quad \text{sigmoid}$$

$$y = \tanh(x) \quad \text{hyperb. tan}$$

$$y = \max\{0, x\} \quad \text{ReLU}$$

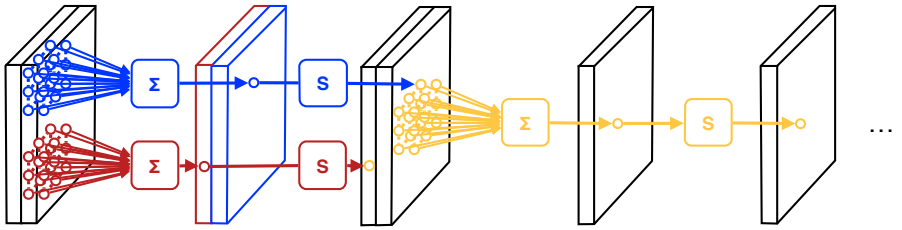
$$y = \log(1+e^{x}) \quad \text{Soft ReLU}$$

$$y = \epsilon x + (1-\epsilon)\max\{0, x\} \quad \text{Leaky ReLU}$$



Multiple layers

Convolution, gating, convolution, …

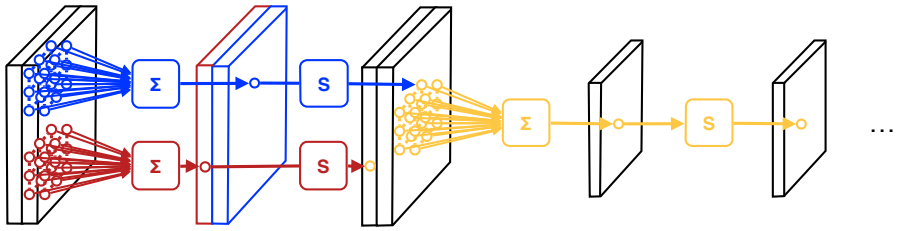


Filters are followed by non-linear operators (e.g. gating, but see later)

Multiple such layers are chained together

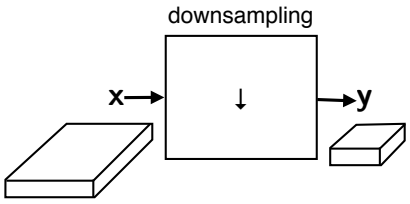
Multiple layers

Downsampling



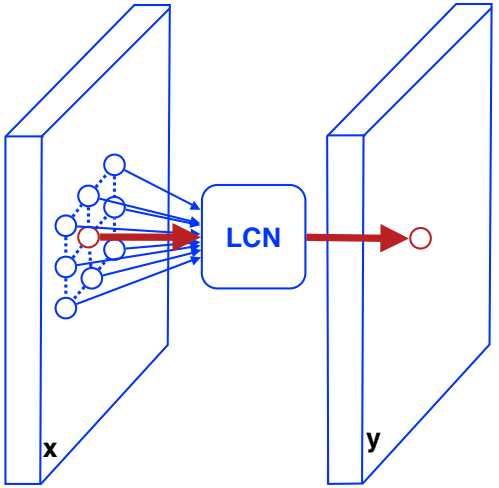
Filters are often followed (or incorporate) downsampling

This is often compensated by an increase in the number of feature channels (not shown)



Local contrast normalisation

Normalise image/feature patches



$$y_{ijq} = \frac{x_{ijq} - \mu_{ijq}}{\sigma_{ijq}}$$

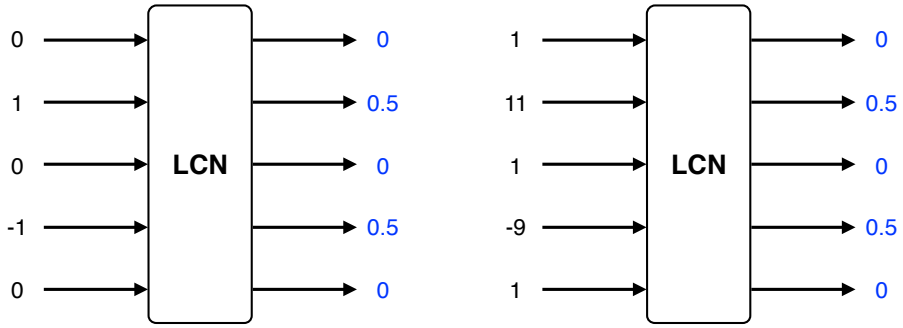
$$\mu_{ijq} = \frac{1}{|\mathcal{N}(i,j)|} \sum_{(u,v)\in\mathcal{N}(i,j)} y_{uvq}$$

$$\sigma^2_{ijq} = \frac{1}{|\mathcal{N}(i,j)|} \sum_{(u,v)\in\mathcal{N}(i,j)} (y_{uvq} - \mu_{ijq})^2$$

Local contrast normalisation

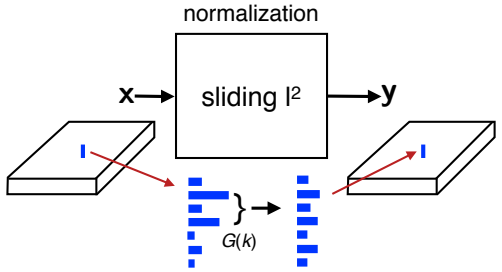
Example

It has a local equalising effect:



Local feature normalisation

Across feature channels rather than spatially



$$y_{ijk} = x_{ijk}\left(\kappa + \alpha \sum_{q \in G(k)} x_{ijq}^2\right)^{-\beta}$$

Operates at each spatial location independently

Normalise groups G(k) of feature channels

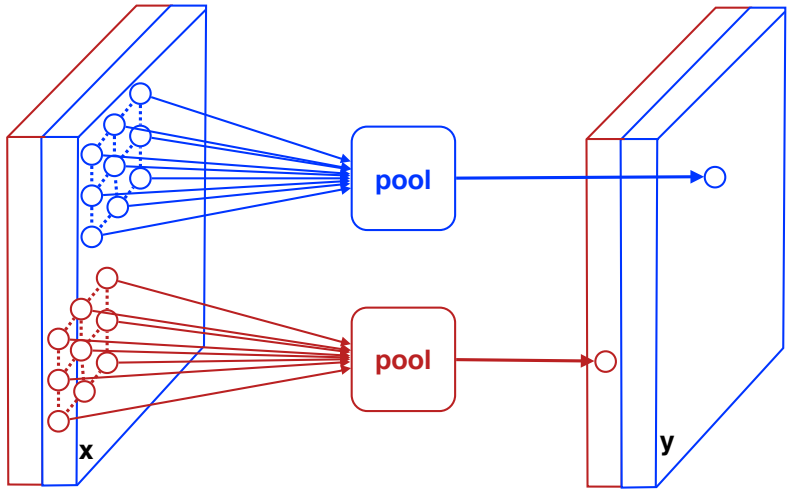
Groups are usually defined in a **sliding window manner**

Spatial pooling

Reduce dependency on precise location

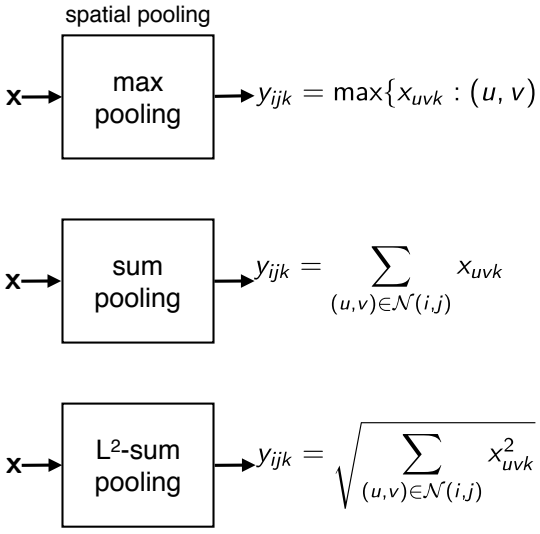
Pooling compute the average / max of the features in a neighbourhood.

It is applied channel-by-channel.



Spatial pooling

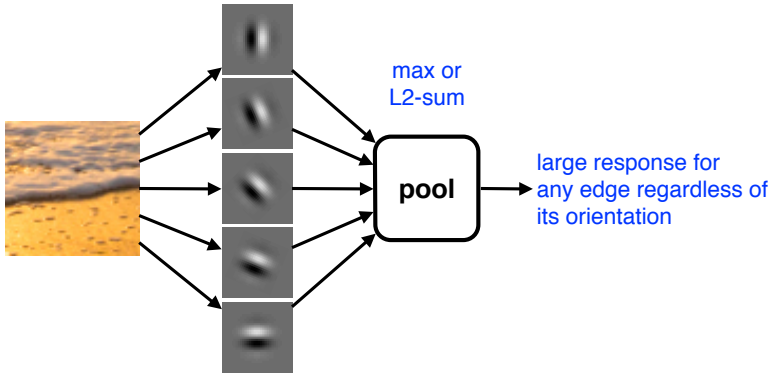
Variants



Feature pooling

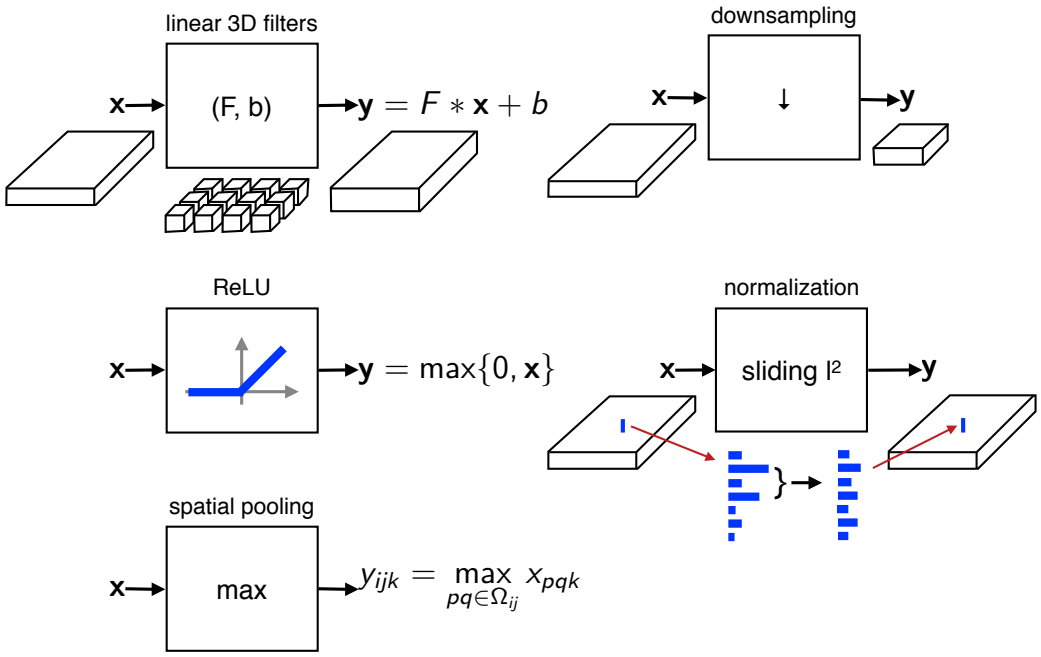
Across feature channels, not in space

Pooling across feature channels (filter outputs) can achieve invariance



L2 pooling, in particular, is invariant to the **sign of the edge filter too**

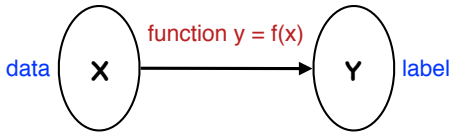
CNN components summary



Possible learning goals

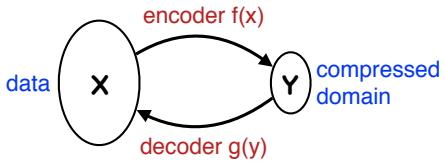
Discriminative training (neural networks)

- Classification / regression
- Solve a task (e.g. object recognition)

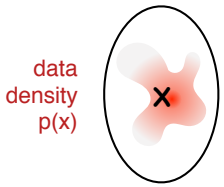


Generative training (autoencoders, Boltzman machines, …)

- Reconstruct the image from a compressed representation (autoencoder)

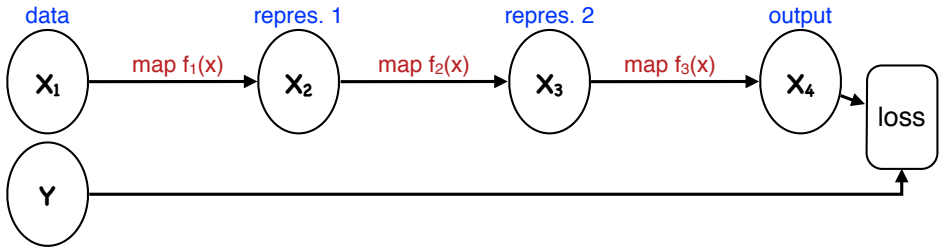


- Model the distribution of the data (Boltzman machines, …)

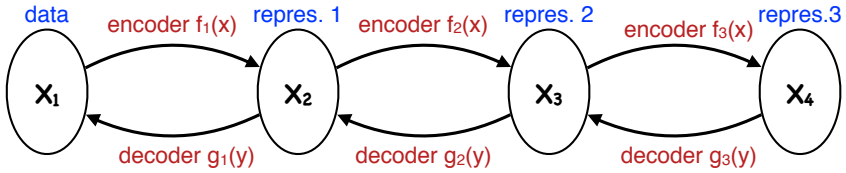


Stage-wise generative training

A key difficult in learning deep models is the complex interaction between all layers. Direct optimisation of a regression loss is difficult:



Generative training allows to train layer by layer using as a target the reconstruction of the layer before:



Autoencoders

Learn to encode and reconstruct the data

Notation

- Let $\mathbf{x}^{(i)}$ be a *vectorised* **image patch**
- Let $\mathbf{w}_j$ be the vector representing the j-th filter (for the vectorised image)
- Let W = [$\mathbf{w}_1^\top \mathbf{w}_2^\top \ldots \mathbf{w}_K^\top$] be a **bank** of K filters

Define:

- Encoder: $\mathbf{y}^{(i)} = W\,\mathbf{x}^{(i)}$
- Decoder: $\mathbf{x}^{(i)} \cong W^\top \mathbf{y}^{(i)}$

Learning objective:

$$E(W) = \sum_{i=1}^{N} \| W^\top W \mathbf{x}^{(i)} - \mathbf{x}^{(i)} \|^2 + \sum_{i=1}^{N} \sum_{j=1}^{K} g(\mathbf{w}_j^\top \mathbf{x}^{(i)})$$

reconstruction error (linear autoencoder) — sparsity term

Autoencoders

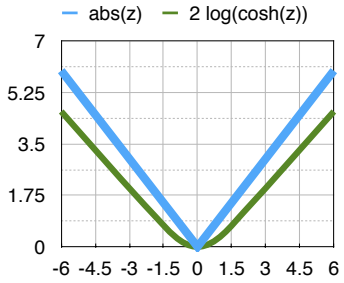
Non-linear autoencoders

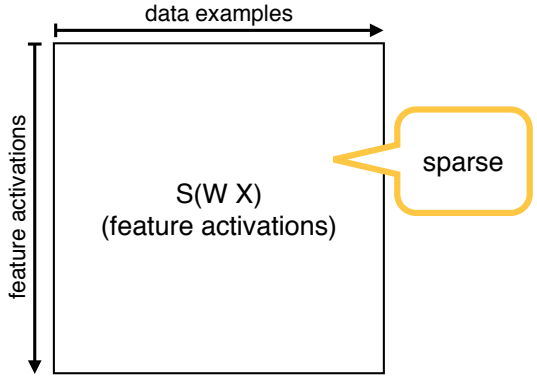
- non-linear gating and/or multiple layers

$$E(W) = \sum_{i=1}^{N} \| S(W^\top S(W \mathbf{x}^{(i)})) - \mathbf{x}^{(i)} \|^2 + \sum_{i=1}^{N} \sum_{j=1}^{K} g(\mathbf{w}_j^\top \mathbf{x}^{(i)})$$

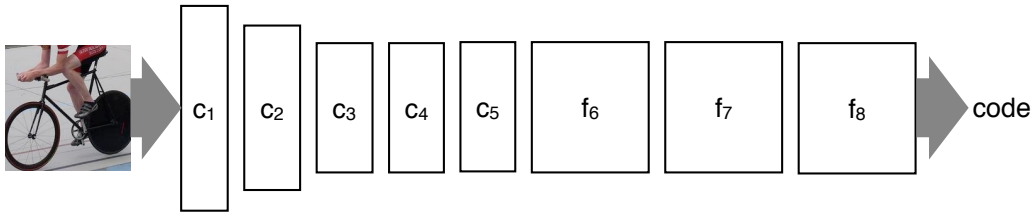
Sparsity term

- L^1 regulariser g(z) = | z |
- Smooth L^1 log(cosh(z))





Convolutional neural networks (CNNs)



x → CNN(x)

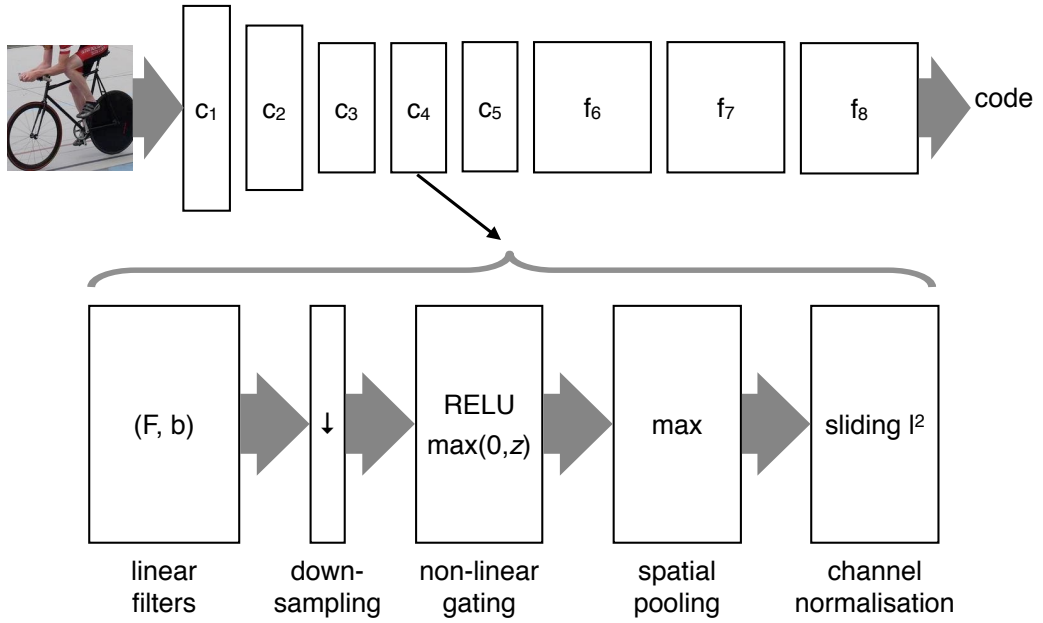
From left to right

- decreasing spatial resolution
- increasing feature dimensionality

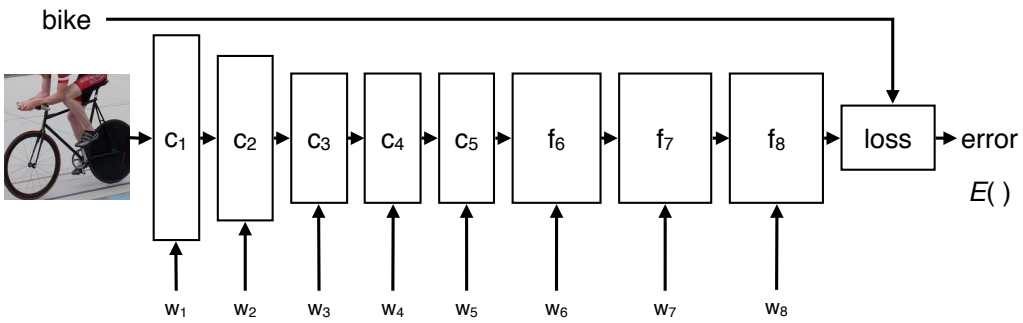
Fully-connected layers

- same as convolutional, but with 1 × 1 spatial resolution
- contain most of the parameters

Convolutional layers



Learning a CNN



$$\operatorname{argmin} E(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_8)$$

Stochastic gradient descent
(with momentum, dropout, …)

Learning CNNs classifiers

Challenge

- many parameters, prone to overfitting

Key ingredients

- very large annotated data
- heavy regularisation (dropout)
- stochastic gradient descent
- GPU(s)

IMAGENET

- 1K classes
- ~ 1K training images per class
- ~ 1M training images

Training time

- ~ 90 epochs
- days—weeks of training
- requires processing ~150 images/sec

What do CNNs learn?

Stochastic gradient descent

The loss is an average over many data points

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

Key idea: approximate the gradient sampling a point at a time:

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

uniform distribution

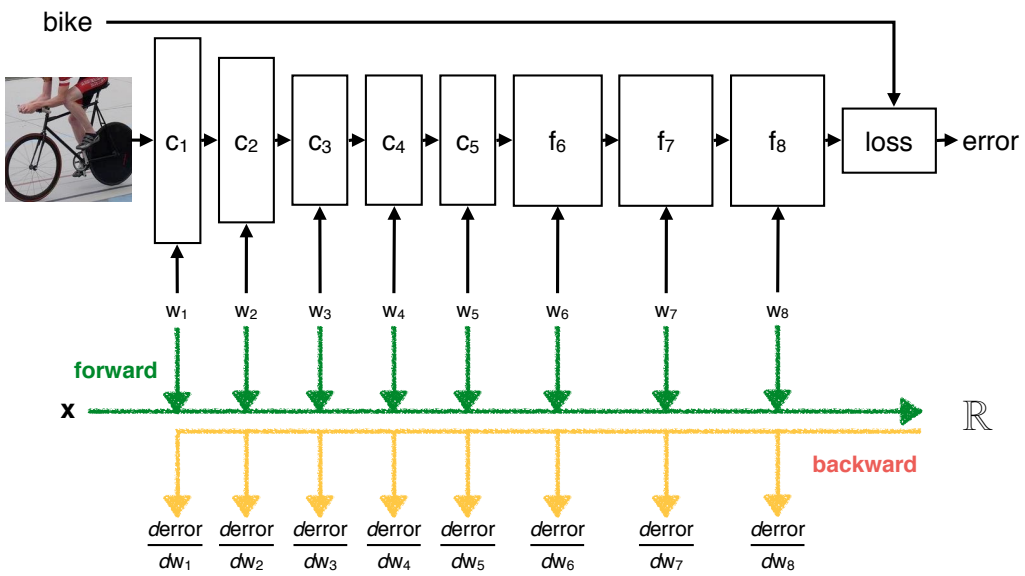
Refinements:

- **Epochs**: all points are visited sequentially, but in random order
- **Validation**: evaluate E($\mathbf{w}_t$) on an held-out validation set to diagnose objective decrease
- **Learning rate**: decreased tenfold once the objective stops decreasing
- **Momentum**: smooth gradient using a moving average:

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

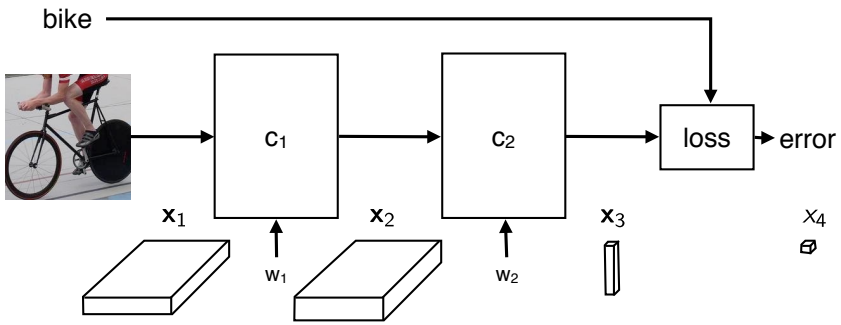
Backpropagation

Compute derivatives using the chain rule



Chain rule

Naive application



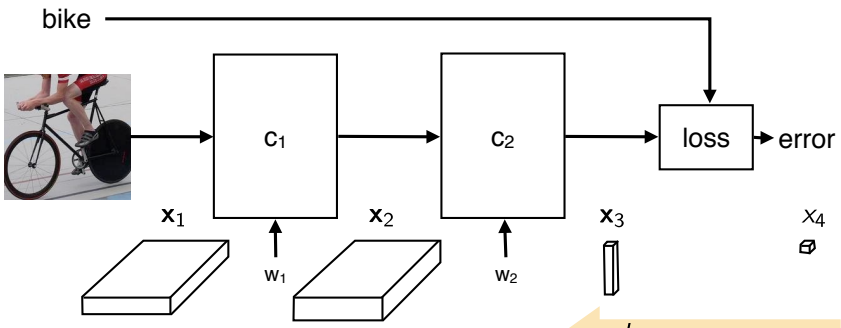
$$\frac{dx_4}{d\mathbf{w}_1^\top} = \frac{dx_4}{d\,\mathrm{vec}\,\mathbf{x}_3^\top} \times \frac{d\,\mathrm{vec}\,\mathbf{x}_3}{d\,\mathrm{vec}\,\mathbf{x}_2^\top} \times \frac{d\,\mathrm{vec}\,\mathbf{x}_2}{d\mathbf{w}_1\top}$$

derivative matrix dimension	$1 \times \dim(\mathbf{w}_1)$	$1 \times H_3W_3K_3$	$H_3W_3K_3 \times H_2W_2K_2$	$H_2W_2K_2 \times \dim(\mathbf{w}_1)$
	✔	✔	✘	✘

E.g. $H_3=H_2=W_3=W_2=64$ and $K_3=K_2=256$ $\Longrightarrow$ 8GB for a single derivative!

Chain rule

The backprop way



explicitly computed

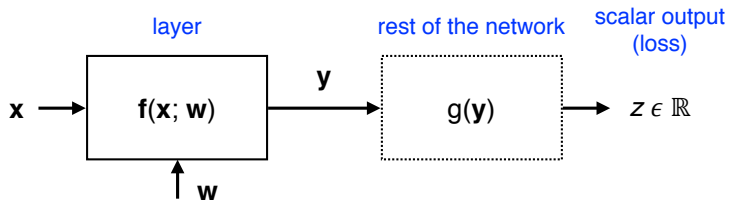
not computed explicitly

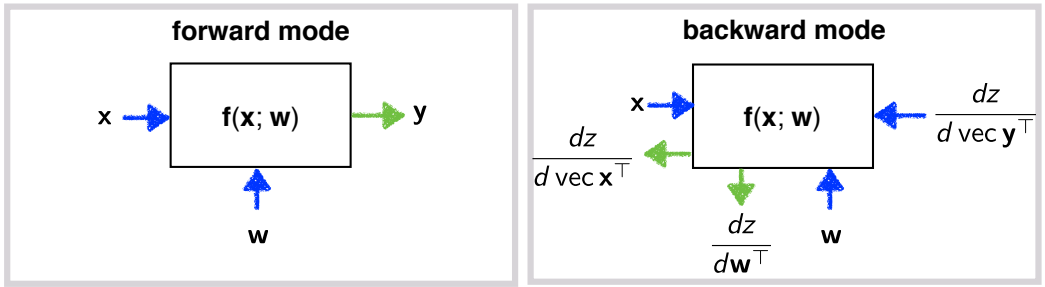
$$\frac{dx_4}{d\,\mathrm{vec}\,\mathbf{x}_3^\top}$$

$$\frac{dx_4}{d\,\mathrm{vec}\,\mathbf{x}_2^\top} = \frac{dx_4}{d\,\mathrm{vec}\,\mathbf{x}_3^\top} \times \frac{d\,\mathrm{vec}\,\mathbf{x}_3}{d\,\mathrm{vec}\,\mathbf{x}_2^\top}$$

$$\frac{dx_4}{d\mathbf{w}_1^\top} = \frac{dx_4}{d\,\mathrm{vec}\,\mathbf{x}_2^\top} \times \frac{d\,\mathrm{vec}\,\mathbf{x}_2}{d\mathbf{w}_1\top}$$

Modular algorithms for backprop





Deep learning

Applications and examples

Visualisation

- Deep dreams
- Deconvolution networks

Transfer learning

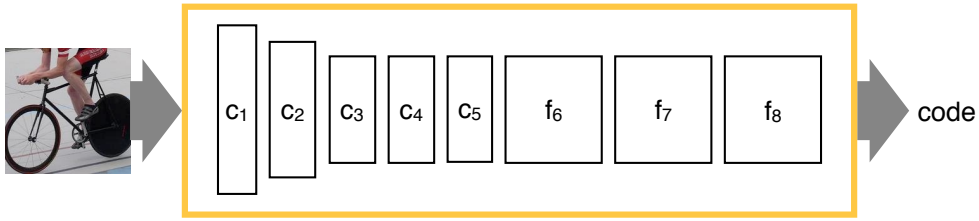
- Generic feature representations
- Comparison with conventional representations

Practical notes

- Toolkits
- MatConvNet

Deep dreams

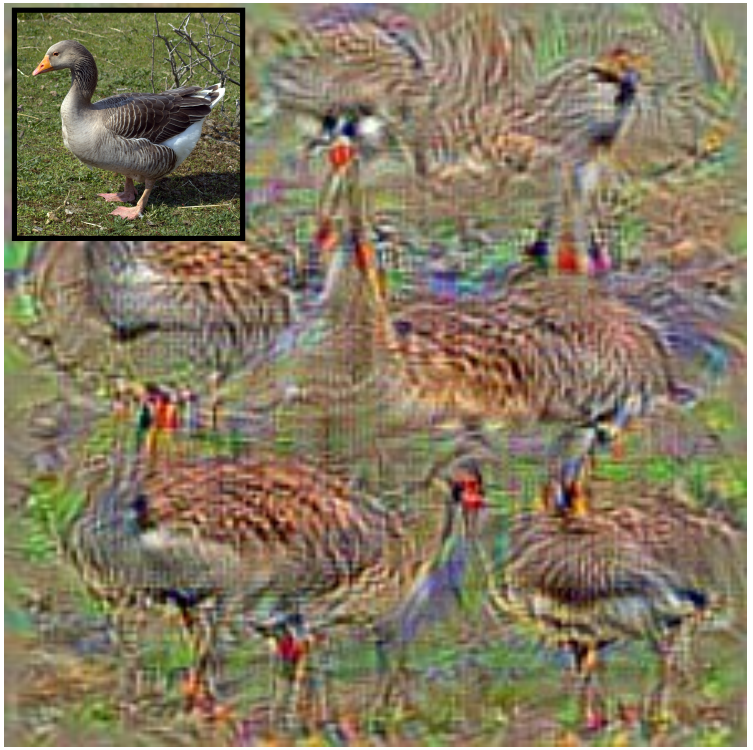
[Erhan *et al.* 2009, Simonyan *et al.* ICLR 2014]

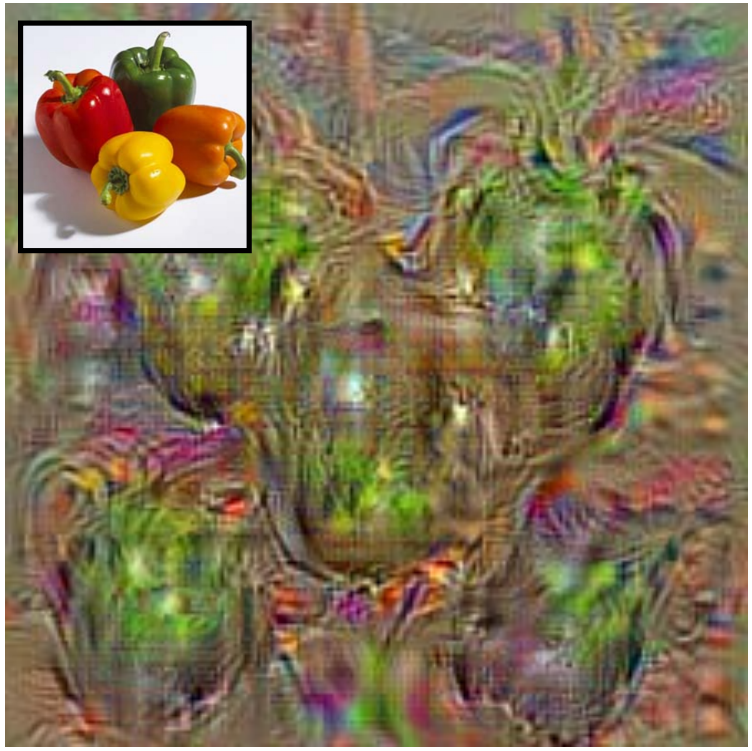


What does deep learning learn?

Invert a CNN by finding the image that maximises the output of a class

$$\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \, \text{CNN}_c(\mathbf{x})$$

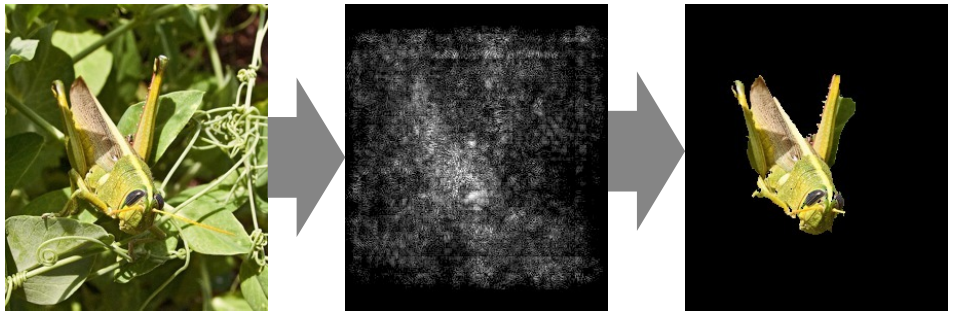




Weakly-supervised detectors

This can be used to **segment objects**

Remarkably, *no object segmentation or bounding box is given during training*



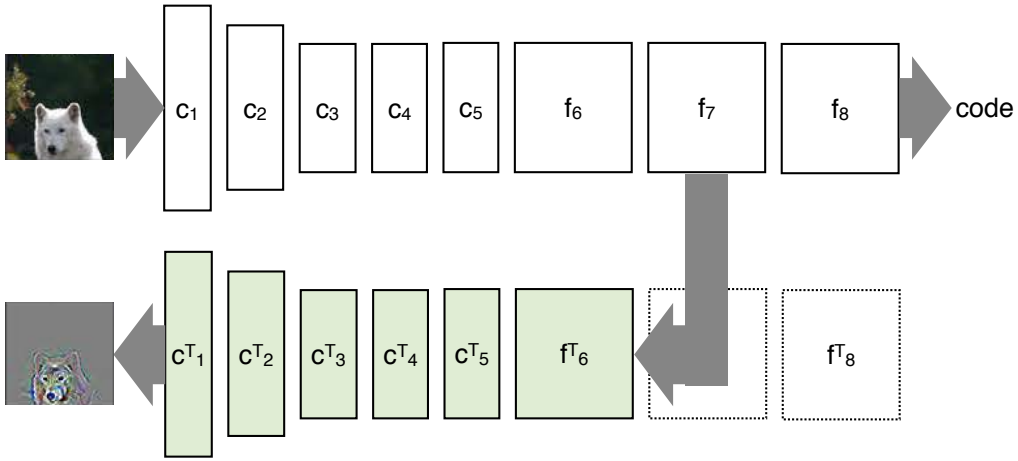
input image | input saliency | grabcut

[Simonyan *et al.* ICLR 2014]

De-convolutional networks

[Zeiler Fergus ECCV 2014]

"Transpose" the architecture to go from activations back to image

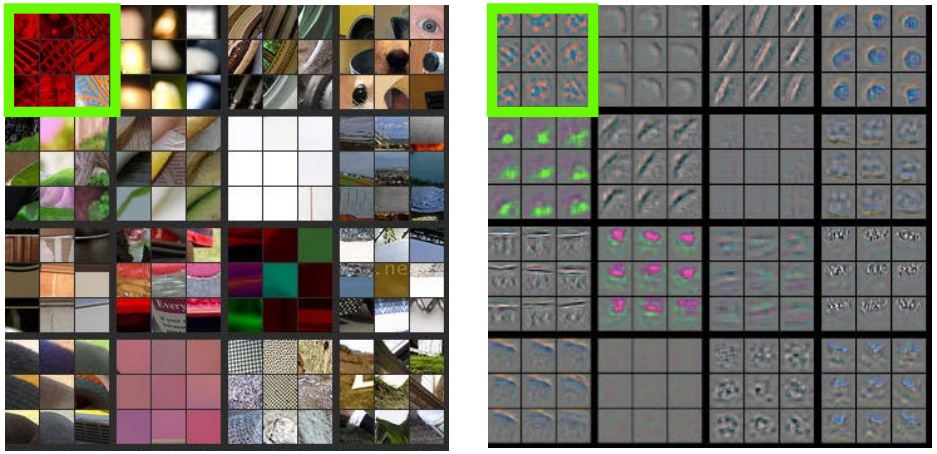


Deconvent visualization

Visualize sample images that excite a given neuron the most

Layer 1

filter response



top 9 exciting patches for each neuron

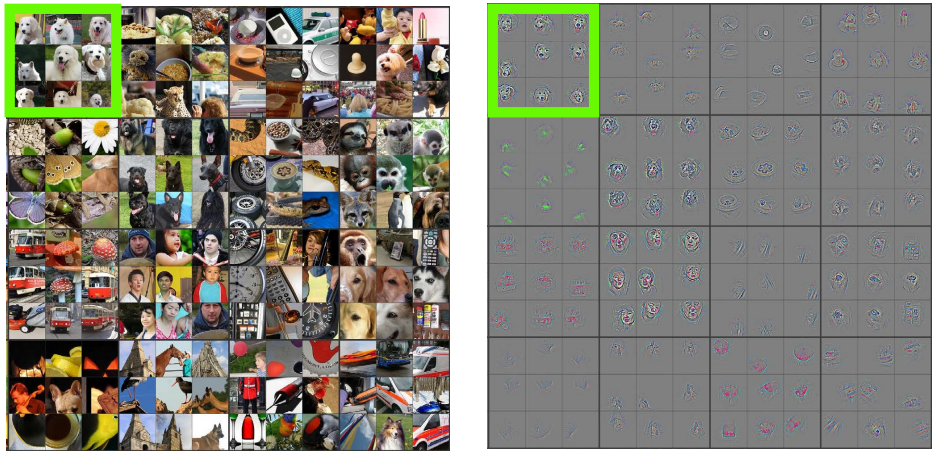
their deconvnet reprojection

Deconvent visualization

Visualize sample images that excite a given neuron the most

Layer 5

filter response



top 9 exciting patches for each neuron

their deconvnet reprojection

What is a deconvnet?

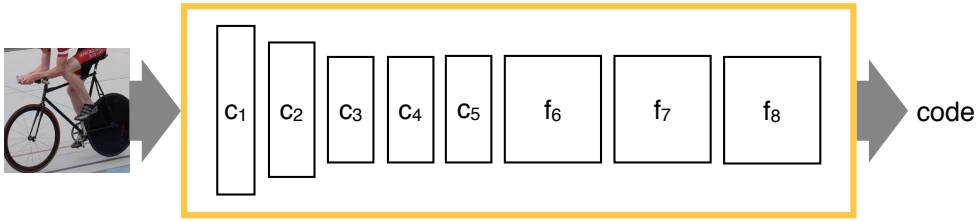
The "transpose" of the CNN

- transpose of the filters (as linear operators)
- max-pooling: remembers activations from forward pass

Alternative interpretation
[Simonyan *et al.* 2014]

- backpropagation applied to the maximum activation problem is neary the same as a deconvnet
- approximate equivalence of "deep dreams" and deconvnets

CNNs as general purpose encoders



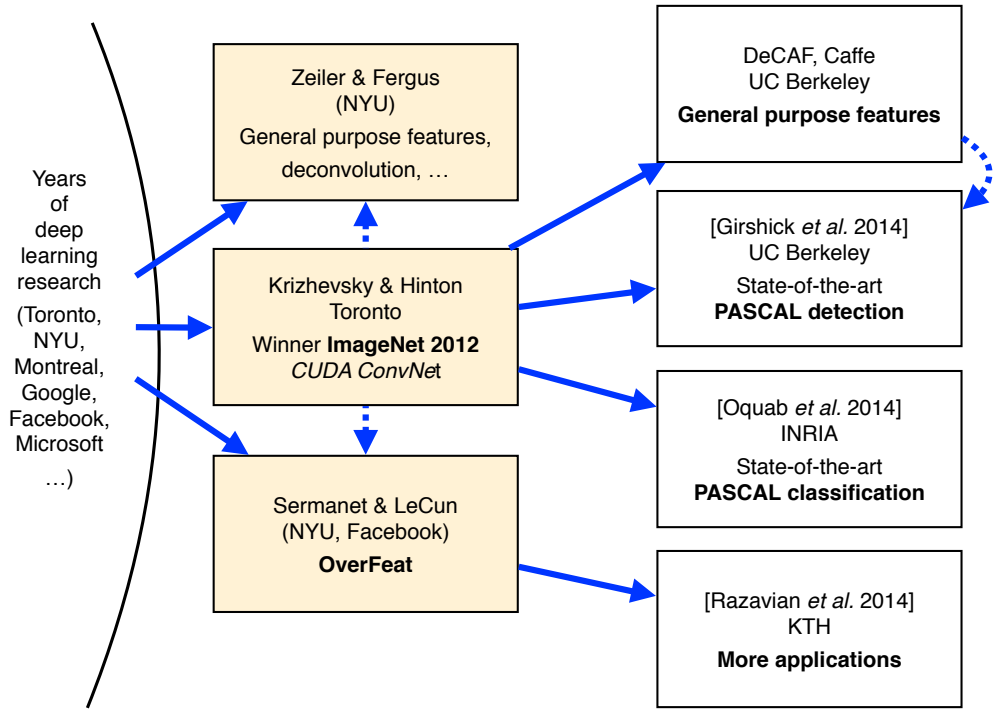
Pre-trained CNN encoders

- Architecture trained on ~ 1M ImageNet images
- Last softmax layer chopped off
- Output used as image encoding

Used as general-purpose features

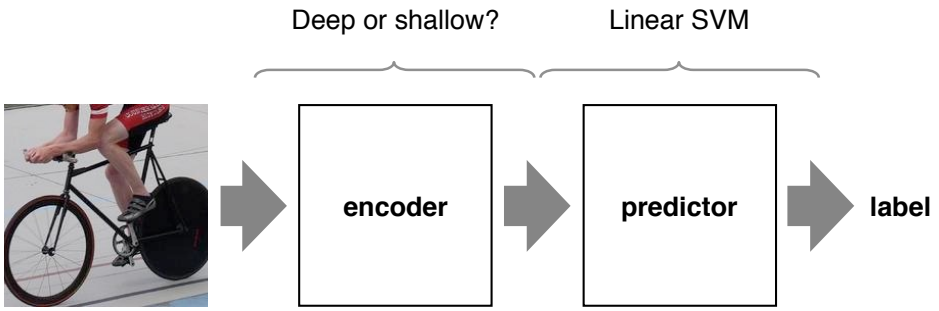
- Applied to PASCAL VOC, Caltech, UCSD Birds, MIT Scene 67, …
- [Zeiler & Fergus, DeCAF, Caffe, …]

Deep visual encodings



Evaluating deep and shallow encoders

A preview of Tuesday talk



Shallow encoder

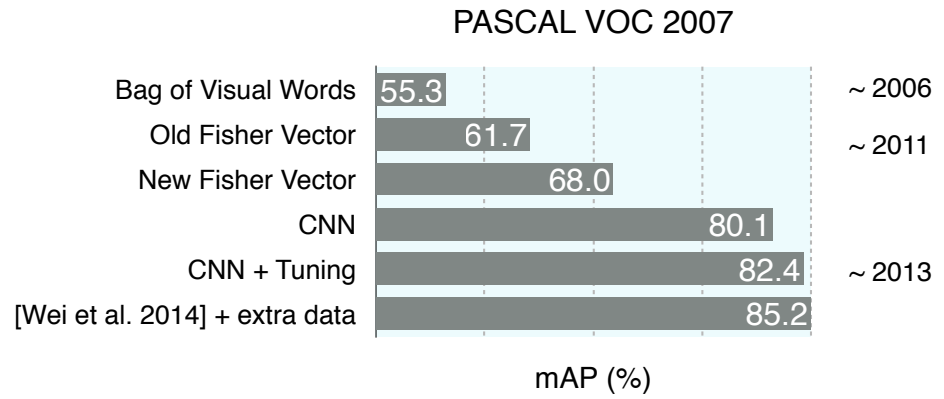
- Further Improved Fisher Vector

Deep encoders

- CNN Fast (CNN-F)
- CNN Medium (CNN-M)
- CNN Slow (CNN-S)

[Return of the devil is in the details, *Chatfield et al.* 2014]

Deep *vs.* shallow



CNNs

- Outperform shallow encodings
- Are expensive to train, but fast to evaluate
- Do provide low-dimensional, general-purpose codes
- Will definitely get much better

Reference implementations

Name	Speed	s/image[1]	Similar to
CNN-S	Slow	1.82	OverFeat
CNN-M	Medium	1.33	Zeiler & Fergus
CNN-F	Fast	0.6	Krizhevsky & Hinton

[Models by Karen Simonyan]

Types

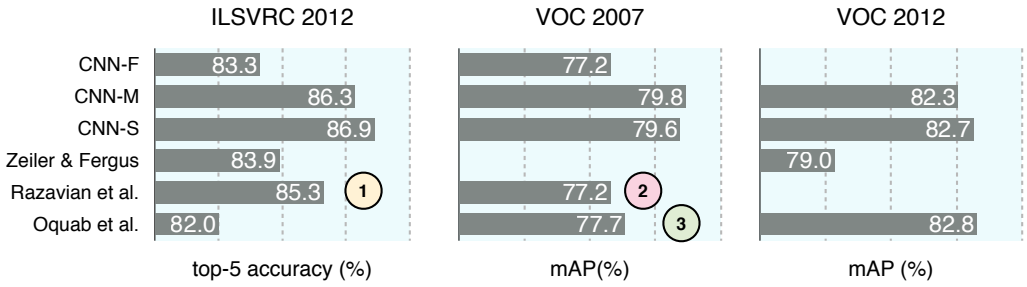
- Inspired by existing implementations
- Trained in-house using one uniform setup

Main differences

- Number of filters
- downsampling factors

[1]time incorporates 10-fold data augmentation

Reference implementations performance



1

Excellent performance[1] on the ImageNet challenge data (~ state-of-the-art).

2

CNN-F,M,S use a modified Caffe

Yet better than other using DeCAF, Caffe, OverFeat

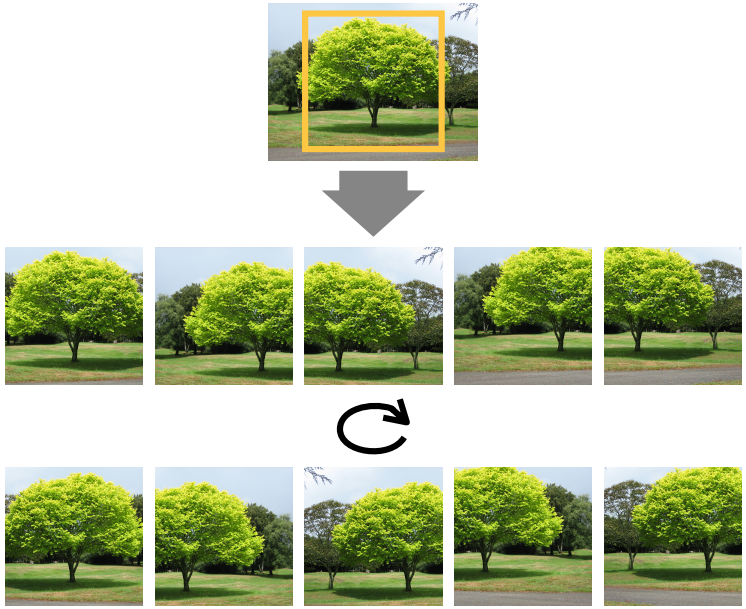
3

Simpler and yet **better** or **equal** than alternative ways of using the encoders.

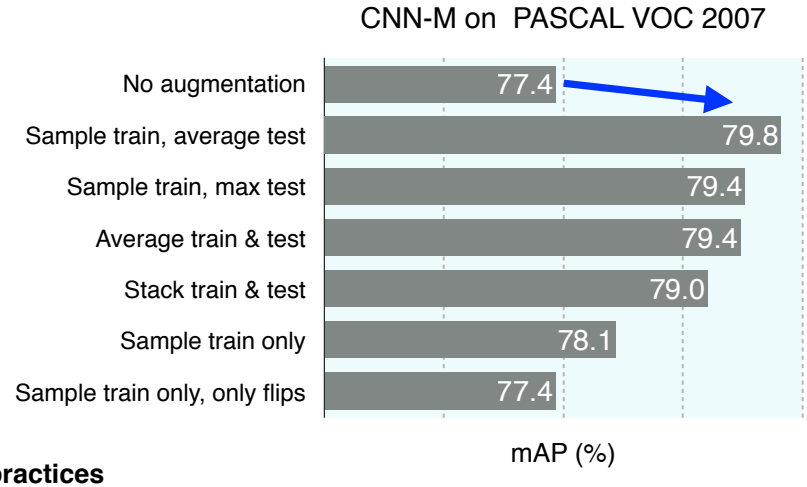
[1] A bit better than OverFeat, probably due to slightly different data augmentation (crops from the whole image & test set augmentation)

Data augmentation

Augment the training data by adding jittered versions of each image



Data augmentation: CNNs

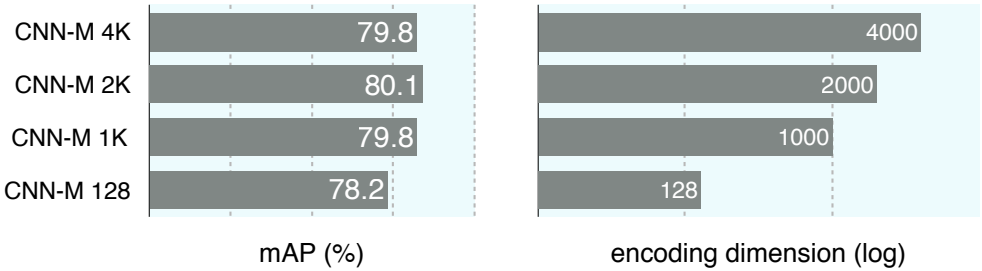


Best practices

- **Sample** *training* and **average** *test*
- Only flipping is insufficient
- Further augmentation has diminishing returns

Dimensionality reduction

Tested on PASCAL VOC 2007



Encodings are often **highly redundant**

CNN

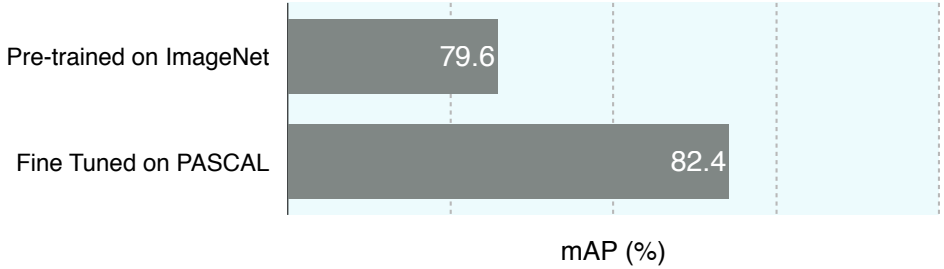
- **reduce dimension 31 times**, ~ same performance
- (re-learn last layer using a multi-class loss and PASCAL VOC)

FV dimensionally reduction

- similar compression possible
- (use e.g. WSABIE [Weston *et al.* 2011])

CNN fine-tuning

PASCAL VOC 2007



Pre-trained CNNs can be **tuned on target dataset**

- Use target data to provide more training images
- Remark: tuning in PASCAL requires a multi-class loss

Often (but not always) yields a nice improvement

Feature generality

How large a gap can pre-trained features jump?

ImageNet pre-training

Object classification (PASCAL VOC)

- [Chatifield et al. 2014, Razavian et al. 2014, Zeiler et al. 2014]

Object detection (PASCAL VOC)

- R-CNN [Girshick et al. 2014]
- Requires region proposals and adaptation for accurate localisation

Fine-grained classification (UCSD birds)

- Part-R-CNN [Zhang et al. 2014]

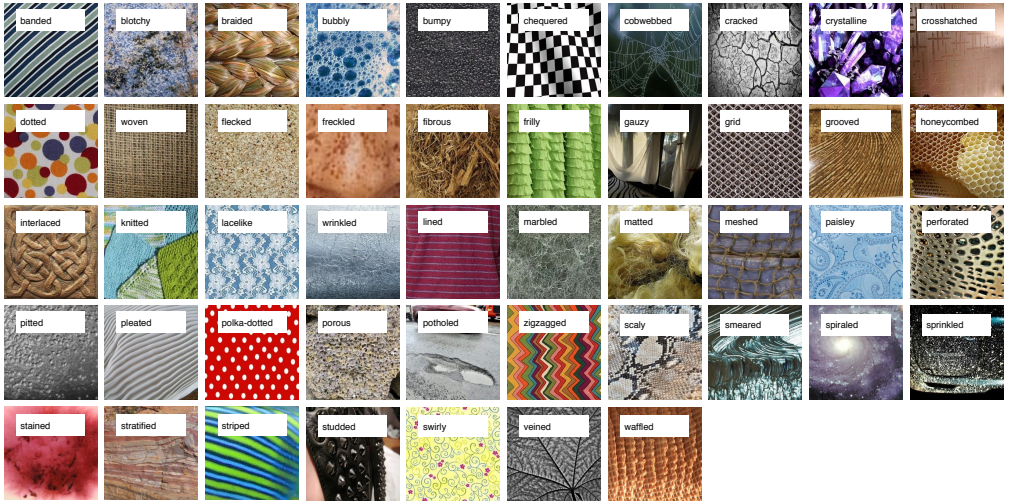
MIT 67 scene classification

- [Razavin et al. 2014]

Beyond objects?

Feature generality

ImageNet pre-trained features achieve **state-of-the-art material recognition** and **texture naming** (but similar to Fisher Vector) [Cimpoi et al. 2014]



[Describable textures dataset]

Feature sharing

The **same** CNN-based **representations** apply to **different tasks**

- ImageNet classification
- object category classification & detection
- scene recognition
- fine-grained bird classification
- texture recognition

Not dissimilar from SIFT, HOG

Can we learn features jointly from multiple tasks?

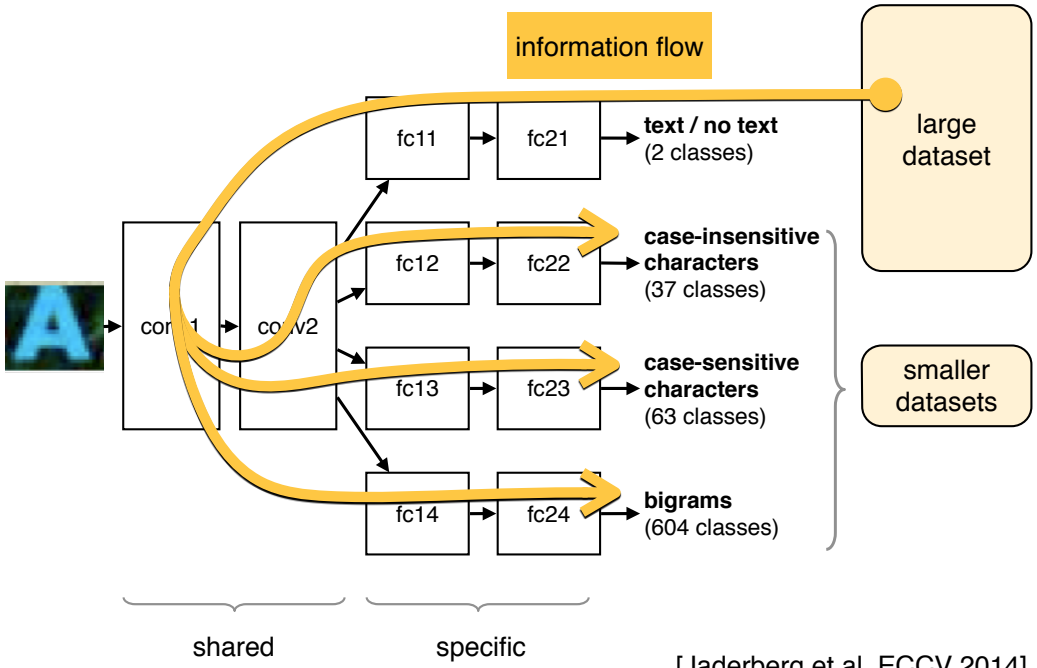
See e.g. [Bengio Courville Vincent PAMI 2013] for a great overview

Example: text spotting

Automatically detect & recognise text in natural images

Also known as PhotoOCR

Tasks are learned synergistically

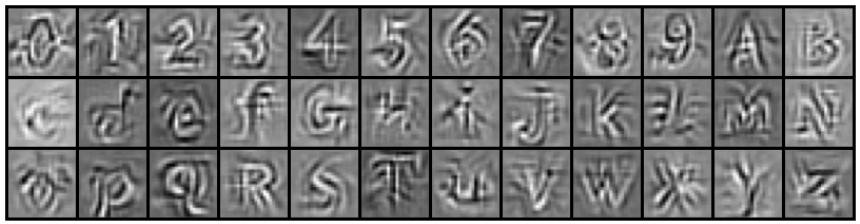


[Jaderberg et al. ECCV 2014]

What have we learned?

Diagnose the model

Use the “deep dreams” trick to visualise the learned character classes:



[Jaderberg et al. ECCV 2014]

Do it yourself

Software

- **CUDA-Convnet 1 & 2**
  https://code.google.com/p/cuda-convnet/
- **Overfeat / Torch** [Lua]
  http://cilvr.nyu.edu/doku.php?id=code:start
- **Berkeley Caffe** [Python]
  http://caffe.berkeleyvision.org
- **Theano** [Python]
  http://deeplearning.net/software/theano/
- **LibCCV**
  http://libccv.org

Pre-trained models

- Return of the Devil in the Details
  http://www.robots.ox.ac.uk/~vgg/research/deep_eval/
- Caffé reference models
  http://caffe.berkeleyvision.org/getting_pretrained_models.html

MatConvNet

http://www.vlfeat.org/matconvnet

A MATLAB toolbox for CNNs

- Similar in spirit to VLFeat.org
- Expose the fundamental computational blocks as MATLAB functions
- Designed for quick experimentation in this environment

Flexibility

- Can run Caffe models
- Pre-trained models form Caffe and VGG

Efficiency

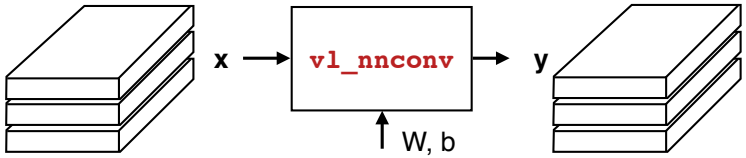
- Computations are inspired by Berkeley Caffe
- *Native MATLAB GPU support*
- 60-70% training speed of Caffe (and improving)

MatConvNet

A CNN toolbox for MATLAB

Forward computation

- operates on a *stack of images*
- each image has *d* feature channels



Available blocks

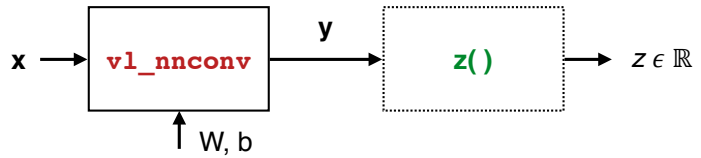
- convolution, pooling, normalization, loss, ReLU, softmax, dropout
- easily extensible (often directly in MATLAB code)

MatConvNet

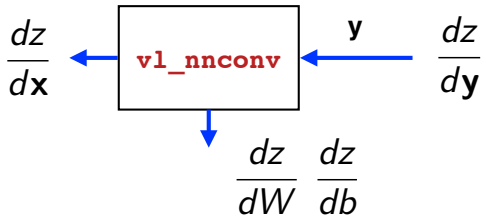
A CNN toolbox for MATLAB

Backward computation

- require network derivatives from block downstream



- chain rule



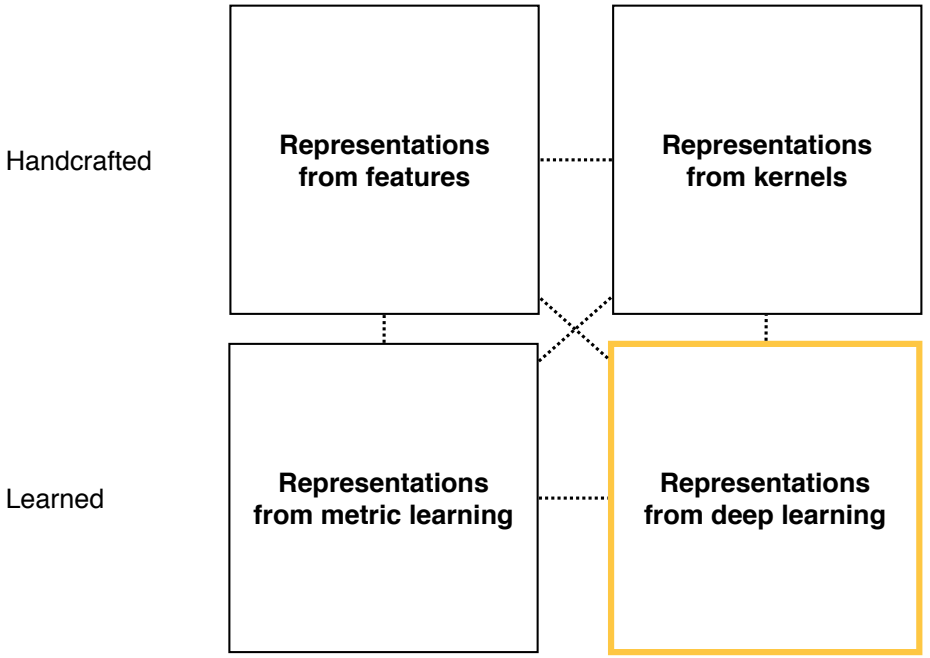
Example

```
% download a pre-trained CNN from the web
urlwrite(...
    'http://www.vlfeat.org/matconvnet/models/imagenet-vgg-f.mat', ...
    'imagenet-vgg-f.mat') ;
net = load('imagenet-vgg-f.mat') ;

% obtain and preprocess an image
im = imread('peppers.png') ;
im_ = single(im) ;
im_ = imresize(im_, net.normalization.imageSize(1:2)) ;
im_ = im_ - net.normalization.averageImage ;

% run the CNN
res = vl_simplenn(net, im_) ;

% show the classification result
scores = squeeze(gather(res(end).x)) ;
[bestScore, best] = max(scores) ;
figure(1) ; clf ; imagesc(im) ;
title(sprintf('%s (%d), score %.3f',...
   net.classes.description{best}, best, bestScore)) ;
```

Represent & predict

- A good representation captures a useful notion of similarity
- Works as a prior in prediction

Representations from hand-crafted features

- HOG, BoVW, VLAD, Fisher Vectors

Representations from kernels

- Derive implicit and explicit representation from a concept of similarity

Representations from metric learning

- Compare & compress with metric learning

Representations from deep learning

- Visualisation, transfer learning, feature sharing
- Excellent performance