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, ...

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

Meaningful representation

Semantic similarity

Vector similarity (distance)

embedding space $\mathbb{R}^d$

$\Phi$ is invariant to nuisance factors, sensitive to semantic variations

Linear predictor

$bicycle$?

$x$

$F(x) = \langle w, x \rangle$

An encoder $\Phi$ maps $x$ into a representation $\Phi(x) \in \mathbb{R}^d$

$x$

$\Phi(x)$

$\Phi(y)$

$\Phi(z)$

near

far

congruous pair

incongruous pair

$\Phi$ is invariant to nuisance factors, sensitive to semantic variations.
Learning predictors

(labelled data \((x_1, y_1), (x_2, y_2), \ldots\))

\(\Phi \rightarrow \text{learning} \rightarrow \Phi^* = \arg\min_w E(w)\)

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

Contents

Data representations: from shallow to deep

Deep learning

Representation generality and transfer learning

Toolkits and MatConvNet
Histogram of Oriented Gradients
[Lowe 1999, Dalal & Triggs 2005]

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

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

BoVW intuition
Discarding spatial information gives lots of invariance
Visual words represent “iconic” image fragments
Fisher Vector (FV)

FV encoding $\Phi = \left[ \begin{array}{c} v_1 \\ u_1 \\ v_2 \\ u_2 \\ \vdots \\ v_K \\ u_K \end{array} \right]$

First and second order statistics

\[
\begin{align*}
v_k &= \frac{1}{M\sqrt{\pi_k}} \sum_{i=1}^{M} \gamma_k(x_i) \frac{x_i - \mu_k}{\sigma_i} \\
u_k &= \frac{1}{M\sqrt{2\pi_k}} \sum_{i=1}^{M} \gamma_k(x_i) \left( \frac{x_i - \mu_k}{\sigma_i} - 1 \right)^2
\end{align*}
\]

A kernel directly encodes a notion of data similarity

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

Kernels
Similarity and kernels

Recall: the encoder $\phi(I)$ should embody a useful notion of similarity

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

Positive definite kernel = inner product of feature vectors
$K(x, y) = \langle \Psi(x), \Psi(y) \rangle$
$\Psi(x) \in V$

Finding kernel maps

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

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

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

Kernel as representations

linear kernel
$\langle \phi(I), \phi(I') \rangle$

Finding kernel maps

Finding kernel maps

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

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

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

$K(x, y) = \langle \Phi(x), \Phi(y) \rangle$
$\Phi(x) \in \mathbb{R}^d$
Example: Chi$^2$ map

With the hom. kernel feature map
\[ x = .01:.01:1; \]
\[ \psi = \text{vl_homkermap}(x,1); \]
\[ K = \psi^*\psi; \]

VLFeat Toolbox
http://www.vlfeat.org

MATLAB code for Chi$^2$ kernel
\[
x = .01:.01:1; \\
for i = 1:100 \\
  for j = 1:100 \\
    K(i,j) = ... \\
      2*x(i)*x(j)/(x(i)+x(j)); \\
  end \\
end
\]

For a thorough review: [Weinberger Saul JMLR 2009]

Learning to compare

Goal

▶ compare (rather than classify) objects $x, y$
▶ formally, learn a distance $d^2(x,y)$

Desiderata

▶ if $x$ and $y$ are congruous $\implies$ small distance
▶ if $x$ and $y$ are incongruous $\implies$ large distance

Parametrisation of the distance

Euclidean distance + linear projection $W$
\[ d^2_W(x, y) = \| Wx - Wy \|^2 \]

Classification-like constraints

For all object pairs $x, y$

▶ congruous $\implies$ distance smaller than threshold - margin
▶ incongruous $\implies$ distance larger than threshold + margin

\[ d^2_W(x, y) < b, \quad d^2_W(u, v) > b + 1 \]
Learning formulation

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

Input: training data
- congruous pairs $P$ (i.e., positive)
- incongruous pairs $N$ (i.e., negative)

Input: regulariser $R(W)$
- 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

$$d^2_W(x,y) = ||W_x - W_y||^2$$

Euclidean distance

Linear projection

$$x \in \mathbb{R}^n \rightarrow \bar{x} = W x \in \mathbb{R}^n$$

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

$W$ can also reduce the data dimensionality
- simply pick $m \ll n$

Contents

Data representations: from shallow to deep

Deep learning

Representation generality and transfer learning

Toolkits and MatConvNet
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

Activation function (sigmoid)

The gating function is a sigmoid.

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

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

Sigmoid function $S(z)$

Perceptron steps:
1. Map a vector $\mathbf{x}$ to a scalar score by an affine projection $(\mathbf{w}, b)$
2. Transform the score monotonically but non-linearly by the sigmoid $S()$
Fitting to data

Perceptron

Training data: \((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\)

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

- Likelihood that \(y_i = 1\):
  \[ P(y_i = 1 | x_i, w) = f(x_i, w) \]

- Likelihood that \(y_i = 1\) or \(0\):
  \[ P(y_i | x_i, w) = f(x_i, w)^y_i (1 - f(x_i, w))^{1-y_i} \]

- Negative log-likelihood
  \[ -\log P(y_i | x_i, w) = -y_i \log f(x_i, w) - (1 - y_i) \log (1 - f(x_i, w)) \]

- Average over data points to obtain an objective function to minimise:
  \[ E(w) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log f(x_i, w) + (1 - y_i) \log (1 - f(x_i, w)) \]

Multi-class perceptron

Introducing the softmax layer

Softmax = sigmoid for 2 classes

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

\[ \frac{e^{\xi_i}}{e^{\xi_i} + e^{\xi_j}} \]

Shown for 2-classes, useful for 3 or more

As a regressor

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

\[ \text{data} \xrightarrow{\text{function } y = f(x)} \text{label} \]

\[ x \rightarrow y = S(\langle w, x \rangle + b) \]

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

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

Softmax = sigmoid for 2 classes

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

\[ \frac{e^{\xi_i}}{e^{\xi_i} + e^{\xi_j}} \]

\[ \frac{e^{\xi_i}}{e^{\xi_i} + e^{\xi_j}} = \frac{e^{\xi_i}}{e^{\xi_i} + e^{-\xi_j}} = \frac{1}{1 + e^{-z}} = S(\langle w, x \rangle + b) \]
Fitting to data

Multi-class perceptron

Training data: \((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\)

Negative log-likelihood of class \(y_i = c\):

\[
- \log P(y_i = c | x_i, W) = - \log \frac{e^{w_c^\top x_i + b_c}}{\sum_{q=1}^{C} e^{w_q^\top x_i + b_q}} = -w_c^\top x - b_c + \log \sum_{q=1}^{C} e^{w_q^\top x + b_q}
\]

Objective function:

\[
E(W) = \frac{1}{N} \sum_{i=1}^{N} \left(-w_{y_i}^\top x_i - b_{y_i} + \log \sum_{q=1}^{C} e^{w_q^\top x_i + b_q} \right)
\]

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

- empirical distributions \(Q_i(y_i)\) and
- predicted distributions \(P_i(y_i) = P(y_i | x_i, w, b)\)

Deep architectures

Multi-layer perceptron (MLP)

Discovery of oriented cells in the visual cortex

[Hubel and Wiesel 59]
Data and intermediate neural computations are **discrete vector fields**

\[
\begin{array}{c}
\text{channels} \\
\text{channels} \\
\text{channels}
\end{array}
\]

\[
\begin{array}{c}
k = 1 \\
k = 2 \\
k = 3
\end{array}
\]

As a filter bank

**Linear convolution**

**As a neural network**

Linear, translation invariant, local:

- **Input** \( x = H \times W \times K \) array
- **Filter bank** \( F = H' \times W' \times K \times Q \) array
- **Output** \( y = (H - H' + 1) \times (W - W' + 1) \times Q \) array
As a neural network

Linear convolution

input features → a bank of 2 filters → 2-dimensional output features

Gating functions

Component-wise non-linearity

\[ y = \frac{1}{1 + e^{-x}} \] sigmoid
\[ y = \tanh(x) \] hyperb. tan
\[ y = \max\{0, x\} \] ReLU
\[ y = \log(1 + e^x) \] Soft ReLU
\[ y = \epsilon x + (1 - \epsilon) \max\{0, x\} \] Leaky ReLU

Convolution, gating, convolution, ...

Multiple layers

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

Multiple such layers are chained together
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**

**Example**

It has a local equalising effect:

\[
\begin{array}{cccccc}
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0.5 & 1 & 0.5 & 0.5 \\
-1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0.5 & 0 & 0 & 0.5 \\
0 & -9 & 0.5 & 0 & 0 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
y_{ijq} = \frac{x_{ijq} - \mu_{ijq}}{\sigma_{ijq}}
\]

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

\[
\sigma_{ijq}^2 = \frac{1}{|N(i,j)|} \sum_{(u,v) \in N(i,j)} (y_{uvq} - \mu_{ijq})^2
\]

**Local feature normalisation**

Across feature channels rather than spatially

Operates at each spatial location independently

Normalise groups \(G(k)\) of feature channels

Groups are usually defined in a sliding window manner

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

Reduce dependency on precise location

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

It is applied channel-by-channel.

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

Spatial pooling

Variants

Spatial pooling

\[ y_{ijk} = \max_{(u,v) \in \mathcal{N}(i,j)} x_{uvk} \]

Sum pooling

\[ y_{ijk} = \sum_{(u,v) \in \mathcal{N}(i,j)} x_{uvk} \]

L2-sum pooling

\[ y_{ijk} = \sqrt{\sum_{(u,v) \in \mathcal{N}(i,j)} x_{uvk}^2} \]

CNN components summary

(F, b) linear 3D filters

\[ x \rightarrow y = F \ast x + b \]

downsampling

\[ \downarrow \]

ReLU

\[ y = \max\{0, x\} \]

max or L2-sum

large response for any edge regardless of its orientation
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, …)

---

Autoencoders

**Learn to encode and reconstruct the data**

**Notation**
- Let $x^{(i)}$ be a vectorised image patch
- Let $w_j$ be the vector representing the $j$-th filter (for the vectorised image)
- Let $W = [w_1^T \ldots w_K^T]$ be a bank of $K$ filters

**Define:**
- Encoder: $y^{(i)} = W x^{(i)}$
- Decoder: $x^{(i)} = W^T y^{(i)}$

**Learning objective:**

$$E(W) = \sum_{i=1}^{N} \| W^T W x^{(i)} - x^{(i)} \|^2 + \sum_{i=1}^{N} \sum_{j=1}^{K} g(w_j^T x^{(i)})$$

- **reconstruction error**
  (linear autoencoder)
- **sparsity term**

---

**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:

---

**Non-linear autoencoders**

- non-linear gating and/or multiple layers

$$E(W) = \sum_{i=1}^{N} \| S(W^T S(W x^{(i)})) - x^{(i)} \|^2 + \sum_{i=1}^{N} \sum_{j=1}^{K} g(w_j^T x^{(i)})$$

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

Notation
- Let $x^{(i)}$ be a vectorised image patch
- Let $w_j$ be the vector representing the $j$-th filter (for the vectorised image)
- Let $W = [w_1^T \ldots w_K^T]$ be a bank of $K$ filters

Define:
- Encoder: $y^{(i)} = W x^{(i)}$
- Decoder: $x^{(i)} = W^T y^{(i)}$

Learning objective:

$$E(W) = \sum_{i=1}^{N} \| W^T W x^{(i)} - x^{(i)} \|^2 + \sum_{i=1}^{N} \sum_{j=1}^{K} g(w_j^T x^{(i)})$$

- **reconstruction error**
  (linear autoencoder)
- **sparsity term**

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$

---

**Autoencoders**

**Sparsity term**

- L¹ regulariser $g(z) = |z|$
- Smooth L¹ $\log(\cosh(z))$
Convolutional neural networks (CNNs)

From left to right
▶ decreasing spatial resolution
▶ increasing feature dimensionality

Fully-connected layers
▶ same as convolutional, but with $1 \times 1$ spatial resolution
▶ contain most of the parameters

Learning a CNN

Learning CNNs classifiers

Challenge
▶ many parameters, prone to overfitting

Key ingredients
▶ very large annotated data
▶ heavy regularisation (dropout)
▶ stochastic gradient descent
▶ GPU(s)

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

\[ E(w) = \frac{1}{N} \sum_{i=1}^{N} E_i(w) \]

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

\[ w_{t+1} = w_t - \eta_t \nabla E_i(w_t), \quad i \sim U(\{1, 2, \ldots, N\}) \]

Refinements:

- **Epochs**: all points are visited sequentially, but in random order
- **Validation**: evaluate \( E(w) \) 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:

\[ m_{t+1} = 0.9 m_t + \eta_t \nabla E_i(w_t), \quad w_{t+1} = w_t - m_{t+1} \]

Chain rule

**Naive application**

\[ \frac{d x_4}{d w_1} = \frac{d x_4}{d x_3} \times \frac{d x_3}{d x_2} \times \frac{d x_2}{d w_1} \]

**The backprop way**

\[ \frac{d x_4}{d w_1} = \frac{d x_4}{d x_3} \times \frac{d x_3}{d x_2} \times \frac{d x_2}{d w_1} \]

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

Backpropagation

Compute derivatives using the chain rule
Modular algorithms for backprop

\[ f(x; w) \]

\[ g(y) \]

\[ z \in \mathbb{R} \]

**Contents**

Data representations: from shallow to deep

Deep learning

Representation generality and transfer learning

Toolkits and MatConvNet

**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**

**Zeiler & Fergus**

(YNU)

General purpose features, deconvolution, …

Krizhevsky & Hinton

Toronto

Winner ImageNet 2012

CUDA ConvNet

Sermanet & LeCun

(NYU, Facebook)

OverFeat

DeCAF, Caffe

UC Berkeley

General purpose features

[Oquab et al. 2014]

INRIA

State-of-the-art

PASCAL classification

[Razavian et al. 2014]

KTH

More applications
Evaluating deep and shallow encoders

A preview of Tuesday talk

Deep or shallow?

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]

Reference implementations

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

Main differences
- Number of filters
- downsampling factors

<table>
<thead>
<tr>
<th>Name</th>
<th>Speed</th>
<th>s/image</th>
<th>Similar to</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-S</td>
<td>Slow</td>
<td>1.82</td>
<td>OverFeat</td>
</tr>
<tr>
<td>CNN-M</td>
<td>Medium</td>
<td>1.33</td>
<td>Zeiler &amp; Fergus</td>
</tr>
<tr>
<td>CNN-F</td>
<td>Fast</td>
<td>0.6</td>
<td>Krizhevsky &amp; Hinton</td>
</tr>
</tbody>
</table>

[Models by Karen Simonyan]

Deep vs. shallow

Reference implementations performance

ILSVRC 2012

<table>
<thead>
<tr>
<th>Model</th>
<th>top-5 accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-F</td>
<td>83.9</td>
</tr>
<tr>
<td>CNN-M</td>
<td>83.9</td>
</tr>
<tr>
<td>CNN-S</td>
<td>83.9</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus</td>
<td>83.9</td>
</tr>
<tr>
<td>Razavian et al.</td>
<td>83.9</td>
</tr>
<tr>
<td>Oquab et al.</td>
<td>82.6</td>
</tr>
</tbody>
</table>

VOC 2007

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-F + Tuning</td>
<td>85.2</td>
</tr>
<tr>
<td>[Wei et al. 2014] + extra data</td>
<td>85.2</td>
</tr>
</tbody>
</table>

VOC 2012

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-F,M,S</td>
<td>82.8</td>
</tr>
<tr>
<td>CNN-F,M,S use a modified Caffe</td>
<td>82.8</td>
</tr>
</tbody>
</table>

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.

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

![Data augmentation example](image)

### Data augmentation: CNNs

<table>
<thead>
<tr>
<th>Augmentation Method</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No augmentation</td>
<td>77.4</td>
</tr>
<tr>
<td>Sample train, average test</td>
<td>79.8</td>
</tr>
<tr>
<td>Sample train, max test</td>
<td>79.4</td>
</tr>
<tr>
<td>Average train &amp; test</td>
<td>79.4</td>
</tr>
<tr>
<td>Stack train &amp; test</td>
<td>79.0</td>
</tr>
<tr>
<td>Sample train only</td>
<td>78.1</td>
</tr>
<tr>
<td>Sample train only, only flips</td>
<td>77.4</td>
</tr>
</tbody>
</table>

### Dimensionality reduction

Tested on PASCAL VOC 2007

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP (%)</th>
<th>Encoding Dimension (log)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-M 4K</td>
<td>79.8</td>
<td>4000</td>
</tr>
<tr>
<td>CNN-M 2K</td>
<td>80.1</td>
<td>2000</td>
</tr>
<tr>
<td>CNN-M 1K</td>
<td>79.8</td>
<td>1000</td>
</tr>
<tr>
<td>CNN-M 128</td>
<td>78.2</td>
<td>128</td>
</tr>
</tbody>
</table>

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

Tested on PASCAL VOC 2007

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-trained on ImageNet</td>
<td>79.6</td>
</tr>
<tr>
<td>Fine Tuned on PASCAL</td>
<td>82.4</td>
</tr>
</tbody>
</table>

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?

- **Object classification** (PASCAL VOC)

- **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 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

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

Example: text spotting

Automatically detect & recognise text in natural images

Also known as PhotoOCR
Tasks are learned synergistically

A CNN toolbox for MATLAB
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)

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)

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

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

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

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

Backward computation

- require network derivatives from block downstream

\[ W, b \]

\[ z(\cdot) \in \mathbb{R} \]

\[ x \rightarrow \text{vl\_nnconv} \rightarrow y \rightarrow z(\cdot) \]

\[ \frac{dz}{dx}, \frac{dz}{dy}, \frac{dz}{dW}, \frac{dz}{db} \]

- chain rule

Example

MatConvNet

```matlab
% 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));
```

Wrapping up

Representations

- from features
- from kernels
- from metric learning
- from deep learning

Handcrafted

Learned

Contents

Data representations: from shallow to deep

Deep learning

Representation generality and transfer learning

Toolkits and MatConvNet
MatConvNet
A computer vision-oriented deep learning library

Huge variety of applications
Big networks for image classification

E.g. AlexNet, VGG verydeep, GoogLeNet

Huge variety of applications
Dense networks for semantic segmentation

E.g. Fully-Convolutional Neural networks
[Long et al. 2015]
Siamese networks for face recognition/verification

Huge variety of applications

E.g. VGG-Face

Text spotting

Huge variety of applications

E.g. SynText and VGG-Text

Getting started

In 1 slide

% 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));

Demo
Modular: every part can be used directly

Design

Portable
C++ / CUDA
Core

Building Blocks

Wrappers

Could be used directly or in other languages (e.g.
Python or Lua)
Atomic operations
Reusable
Flexible
GPU support
Pack a CNN model
Simple to use

Atomic operations
Reusable
Flexible
GPU support
Pack a CNN model
Simple to use

Anatomy of a building block

forward (eval)

\[ y = vl\_nnconv(x, W, b) \]
Anatomy of a building block

**forward (eval)**

\[ y = \text{vl}_n\text{ncov}(x, W, b) \]

**backward (backprop)**

\[ dzdx = \text{vl}_n\text{ncov}(x, W, b, dzdy) \]

Modular: every part can be used directly

Very fast implementations
Native MATLAB GPU support

Design

Portability
C++ / CUDA

Building Blocks
- Convolution
- Pooling
- Normalization
- cuDNN, ...

Wrappers
SimpleNN
DagNN

Could be used directly or in other languages (e.g., Python or Lua)

Atomic operations
Reusability
Flexible
GPU support

Pack a CNN model
Simple to use
Design

Modular: every part can be used directly

Portable C++ / CUDA Core
Convolution, pooling, normalization, cuDNN, ...

Wrappers
SimpleNN DagNN

Build Blocks

Atomic operations
Reusable
Flexible
GPU support

Could be used directly or in other languages (e.g. Python or Lua)

Pack a CNN model
Simple to use

DagNN wrapper
Complex CNN topologies
Example: GoogLeNet
Alternatives
So many toolboxes!
Torch, Caffe, Caffe 2, Theano, CUDA-Convnet, CUDA-Convnet 2, LibCCV, MXNet, SIGNA, ...

MatConvNet

Demo

Fast
Similar speed as Caffe
MATLAB
Popular in many circles
Ease of use
Documentation

Low barrier of entry
Simply hackable
Modular (take what you need)
Solid roots (VLFeat)
Actually used in CV research
Momentum

Wrapping Up

Represent & predict
A good representation captures a useful notion of similarity
Works as a prior in prediction

Representations types
Hand-crafted: HOG, BoVW, VLAD, Fisher Vectors
Kernels (explicit feature maps)
Metric learning (compare & compress)
Deep learning

Representations from deep learning
Excellent performance
Modular architecture
Large scale training
Feature generality