Deep-Learning: general principles + Convolutional Neural Networks

Pr. Fabien MOUTARDE
Center for Robotics
MINES ParisTech
PSL Université Paris

Fabien.Moutarde@mines-paristech.fr
http://people.mines-paristech.fr/fabien.moutarde

Acknowledgements

During preparation of these slides, I got inspiration and borrowed some slide content from several sources, in particular:

- Yann LeCun + MA Ranzato: slides on « Deep Learning » from the corresponding course at NYU
- Hinton+Bengio+LeCun: slides of the NIPS’2015 tutorial on Deep Learning
- Fei-Fei Li + A.Karpathy + J.Johnson: Stanford course lecture slides on « Convolutional Neural Networks »
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation
• Deep-Learning on 1D signal and 3D data
• Recent other image-based applications

Deep-Learning: general principles + convNets, Pr. Fabien MOUTARDE, Center for Robotics, MINES ParisTech, PSL, Nov.2020

Deep-Learning recent breakthroughs

Very significant improvement over State-of-the-Art in Pattern Recognition / Image Semantic Analysis:

• won many vision pattern recognition competitions (OCR, TSR, object categorization, facial expression,...)
• deployed in photo-tagging by Facebook, Google, Baidu,...

Similar dramatic progress in Speech recognition + Natural Language Processing (NLP)
Main application domains of Deep-Learning

Object recognition
Scene analysis
Robotics

Language processing
Speech recognition
Medical diagnosis & Bio-informatics

Is Deep-Learning « Large-Scale »?

Big and/or « Fat » data

\[
\begin{array}{cc}
\text{d dimensions} & \text{t tasks} \\
\text{n samples} & X & Y
\end{array}
\]

Deep-Learning: Large MODELS

State-of-the-Art Convolutional Neural Networks contain > 100 layers, millions of parameters
Importance of training data!

Dramatic recent progresses in image classification and visual object categorization not only due to Deep-Learning and convNets:

it was made possible largely thanks to ImageNet dataset, which is a HUGE collection of labelled general-purpose images (1000 categories, > 1 million examples)

Most powerful convNets have been trained on this huge dataset!

Importance of « features » in classical Machine-Learning

Traditional Machine Learning Flow

Examples of *hand-crafted* features

Haar features

Control-points features

HoG (Histogram of Gradients)
What is Deep-Learning?

Learning a hierarchy of increasingly abstract representations

- Increasing level of abstraction
  - Each stage ~ trainable feature transform

Image recognition
- Pixel → edge → texton → motif → part → object

Speech
- Sample → spectral band → ... → phoneme → word

Text
- Character → word → word group → clause → sentence → story

Deep-Learning vs. shallow Machine-Learning

**Shallow ML using handcrafted features**
- Input (raw) → Handcrafted features → Learnt transformation → Output

**DL: jointly learn classification and features**
- Input (raw) → Learnt low-level features → Learnt intermediate level features → Learnt high-level (abstract) features → Learnt transformation → Output
Why features should be learnt?

Real data examples for a given task are usually not spreaded everywhere in input space, but rather clustered on a low-dimension « manifold »

Example: Face images of 1000x1000 pixels ➔ « raw » examples are vectors in $\mathbb{R}^{1000000}$ !!

- position = 3 cartesian coord
- orientation 3 Euler angles
- 50 muscles in face
- Luminosity, color

➔ Set of all images of ONE person has $\leq 69$ dim

➔ Examples of face images of 1 person are all in a LOW-dim manifold inside a HUGE-dim space

Good features ~ « mapping » on manifold
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation
• Deep-Learning on 1D signal and 3D data
• Recent other image-based applications

Convolutional Neural Networks (CNN, or ConvNet)

• Proposed in 1998 by Yann LeCun (french prof.@ NYU, now also AI research director of Facebook)

CNN called LeNet by Yann LeCun (1998)

• For inputs with correlated dims (2D image, 1D signal,...)
• Supervised learning
• Wins most vision pattern recognition competitions (OCR, TSR, object categorization, facial expression, …)
• Deployed in photo-tagging by Facebook, Google, Baidu, …
• Also used in real-time video analysis for self-driving cars

Short reminder on what is a (multi-layer) Neural Network

For “Multi-Layer Perceptron” (MLP), neurons type generally “summating with sigmoid activation”
Reminder on artificial “neurons”

**PRINCIPLE**

\[ O_j = f \left( W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i \right) \]

- **Threshold** (Heaviside or sign) → *binary* neurons
- **Sigmoid** (logistic or tanh) → most common for MLPs
- **Identity** → *linear* neurons
- **ReLU** (Rectified Linear Unit)
- **Saturation**
- **Gaussian**

\[ W_{0j} = \text{"bias"} \]

Why MLP directly on pixels is generally a BAD idea?

Huge # of parameters, NO invariance at all
Why convolutions?

For image “semantic” classification, shift-invariance of features is useful

And ANY shift-invariant & linear system can always be expressed as a **CONVOLUTION**:

$$y[n] = \sum x[m] h[n-m]$$

(where $h[n]$ is the impulse response).

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
- Deep-Learning on 1D signal and 3D data
- Recent other image-based applications
Convolution: sliding a 3D filter over image

32x32x3 image

5x5x3 filter

At sliding position i,j
\[ \sigma(i,j) = b + \mathbf{W} \cdot \mathbf{x}_{ij} \]

with \( \mathbf{x}_{ij} \) = 5x5 image patch in 3 colors → vector of dim 75, as filter coeffs in \( \mathbf{W} \)

Non-linear activation:
\[ o(i,j) = f(\sigma(i,j)) \]
\( f= \tanh, \text{ReLU}, \ldots \)

Convolution in action

From http://cs231n.github.io/convolutional-networks/
Example of typical results of convolution

« Neural » view of convolution filters and layers

Each convolution FILTER is one set of neuron parameters

Each convolution LAYER is a set of ~imageSize neurons, but they all have same SHARED weights (perform SAME convolution)

\[ O = f \left( W_0 + \sum_{i=1}^{n} W_i e_i \right) \]

where

- \( O \) is the output
- \( f \) is the activation function
- \( W_0 \) is the "bias"
- \( W_i \) are the shared weights
- \( e_i \) are the inputs
A convNet: succession of Convolution+activation Layers

NB: each convolution layer processes **FULL DEPTH** of previous activation map
**Convolution of convolutions!**

- Image
- Convolutional layer
- Next layer

---

**Pooling layers**

**Goal:**
- Aggregation over space
- Noise reduction,
- Small-translation invariance,
- Small-scaling invariance
Pooling layers algorithm details

Parameters:
- pooling size (often 2x2)
- pooling stride (usually = pooling_size)
- Pooling operation: \textit{max}, average, L_p, ...

Example: 2x2 pooling, stride 2

Final classification layer: just a classical MLP

CNN called LeNet by Yann LeCun (1998)

AlexNet
Global architecture of convNets

Succession of Convolution (+ optional activation) layers and Pooling layers, which extract the hierarchy of features, followed by dense (fully connected) layer(s) for final classification

Typical convolutional filters after training

Architecture with a deep succession of layers processing coarser and coarser “images”

➡ Lower layer learns optimized low-level filters
  (detection of ~edges in L1, ~corners/arcs in L2)

➡ Higher level layers learn more “abstract” filters
  (~“texture types” in L3, ~object parts in L4)

➡ Last layer output a representation on which it is easy to discriminate between classes
ConvNet training

All successive layers of a convNet forms a Deep neural network (with weigh-sharing inside each conv. Layer, and specific pooling layers).

Training a NN = optimizing values of weights&biases

➤ Stochastic Gradient Descent (SGD), using back-propagation:
- Input 1 (or a few) random training sample(s)
- Propagate
- Calculate error (loss)
- Back-propagate through all layers from end to input, to compute gradient
- Update convolution filter weights
Computing gradient through cascade of modules

Cost function is $Q(t) = \sum_m \text{loss}(Y_m, D_m)$, where $m$ runs over training set examples

Usually, $\text{loss}(Y_m, D_m) = ||Y_m - D_m||^2$ [quadratic error]

Total gradient:
$$W(t+1) = W(t) - \lambda(t) \text{grad}_W(Q(t)) + \mu(t)(W(t) - W(t-1))$$

Stochastic gradient:
$$W(t+1) = W(t) - \lambda(t) \text{grad}_W(Q_m(t)) + \mu(t)(W(t) - W(t-1))$$

where $Q_m = \text{loss}(Y_m, D_m)$, is error computed on only ONE example randomly drawn from training set at every iteration and

- $\lambda(t) = \text{learning rate}$ (fixed, decreasing or adaptive),
- $\mu(t) = \text{momentum}$

Now, how to compute $dQ_m/dW_{ij}$?
Backprop through layers: chain rule derivative computation

If neuron j is output, \( \delta_j = \frac{dE_m}{ds_j} = \frac{dE_m}{dy_j} \frac{dy_j}{ds_j} \) with \( E_m = ||Y_m - D_m||^2 \)

so \( \delta_j = 2(y_j - D_j)f'(\sigma_j) \) if neuron j is an output

Otherwise, \( \delta_j = \sum_k \delta_k w_{jk} \frac{dy_j}{d\sigma_j} = \sum_k \delta_k w_{jk} \) if neuron j is “hidden”

\( \Rightarrow \) all the \( \delta_j \) can be computed successively from last layer to upstream layers by “error backpropagation” from output

Why gradient descent works despite non-convexity?

Error surface for neural net are NOT CONVEX!

- Local minima dominate in low-Dim...
- ...but recent work has shown that saddle points dominate in high-Dim

- Furthermore, most local minima are close to the global minimum
Saddle points in training curves

- Oscillating between two behaviors:
  - Slowly approaching a saddle point
  - Escaping it

Some ConvNet training « tricks »

- Importance of input normalization
  (zero mean, unit variance)
- Importance of weights initialization
  random but SMALL and prop. to 1/sqrt(nbInputs)
- Decreasing (or adaptive) learning rate
- Importance of training set size
  ConvNets often have a LARGE number of free parameters
  ➔ train them with a sufficiently large training-set!
- Avoid overfitting by:
  - Early Stopping of learning iterations
  - Use of L1 or L2 regularization (after some epochs)
  - Use « Dropout » regularization (esp. on large FC layers)
What is Overfitting?

Trying to fit too many free parameters with not enough information can lead to overfitting.

How to detect overfitting for iterative training?

Better = AVOID overfitting by REGULARIZATION

Avoid overfitting by EARLY STOPPING

• For Neural Networks, a first method to avoid overfitting is to STOP LEARNING iterations as soon as the validation_error stops decreasing

• Generally, not a good idea to decide the number of iterations beforehand. Better to ALWAYS USE EARLY STOPPING
Avoid overfitting using L1/L2 regularization

**Regularization** = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just the L2- or L1-norm of the vector of all weights:

\[ K = \sum_m \text{loss}(Y_m, D_m) + \beta \sum_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2)} \text{ or } p=1 \text{ (L1)} \]

→ name “Weight decay”

DropOut regularization for convNet training

At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~0.5), or re-installed with last values of weights
Batch Normalization

- Inputs are usually normalized, but for layers other than the first one, inputs are the outputs from previous layers, which may be far from staying centered around \([-1;1]\)

→ Idea of renormalizing between layers:
  - during training, normalize **within each mini-batch** (coordinate-wise removing of mean + normalizing by standard deviation → each component centered with \(\sigma=1\))
  - for inference after training, use a **fixed** normalization (based on whole training set)

Inserting BatchNorm layer just before or after each activation layer is generally good idea

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- **Useful pre-trained convNets**
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
- Deep-Learning on 1D signal and 3D data
- Recent other image-based applications
Examples of very successful ConvNets

- **LeNet**: 1st successful applications of ConvNets, by Yann LeCun in 1990’s. Used to read zip codes, digits, etc.

- **AlexNet**: Beginning of ConvNet “buzz”: largely outperformed competitors in ImageNet_ILSVRC2012 challenge. Developed by Alex Krizhevsky et al., architecture similar to LeNet (but deeper+larger, and some chained ConvLayers before Pooling). 60 M parameters!

- **ZF Net**: ILSVRC 2013 winner. Developed by Zeiler & Fergus, by modif of AlexNet on some architecture hyperparameters.

- **GoogLeNet**: ILSVRC 2014 winner, developed by Google. Introduced an Inception Module, + AveragePooling instead of FullyConnected layer at output. Dramatic reduction of number of parameters (4M, compared to AlexNet with 60M).

- **VGGNet**: Runner-up in ILSVRC 2014. Very deep (16 CONV/FC layers) \(\rightarrow\) 140M parameters !


---

LeNet, for digits/letters recognition [LeCun et al., 1998]

**Input:** 32x32 image

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]
AlexNet, for image categorisation  
[Krizhevsky et al. 2012]

Input: 224x224x3 image

60 million parameters !...

GoogleNet  
[Szegedy et al., 2014]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
• ILSVRC 2015 large winner in 5 main tracks (3.6% top 5 error)
• 152 layers!!!
• But novelty = "skip" connections

ResNet (Residual Net), by Microsoft [He et al., 2015]

• 2-3 weeks of training on 8 GPU machine !!
• However, at runtime faster than a VGGNet! (even though it has 8x more layers)
Performance comparison of usual convNet architectures

Summary of recent ConvNet history

But most important is the choice of ARCHITECTURAL STRUCTURE
Current SoA convNets

- ResNeXt
- Xception
- MobileNet
- NASNet
- SqueezeNet
- PyramidNet
- DenseNet
- etc!!...

DenseNet principle:
connect each convolution layer to ALL PREVIOUS LAYERS

Current SoA convNets:
performances on ImageNet
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation
• Deep-Learning on 1D signal and 3D data
• Recent other image-based applications

convNets and GPU

Good convNets are very big (millions of parameters!)
Training generally performed on BIG datasets

⇒ Training time more manageable using GPU acceleration for ultra-parallel processing
Programming environments for Deep-Learning

- TensorFlow  [https://www.tensorflow.org](https://www.tensorflow.org)
- KERAS  [https://keras.io](https://keras.io)
  
  Python front-end APIs mapped either on Tensor-Flow or Theano back-end

- PyTorch  [https://pytorch.org/](https://pytorch.org/)
  
  C++ library, hooks from Python → notebooks

- Theano  [http://www.deeplearning.net/software/theano/](http://www.deeplearning.net/software/theano/)
- Lasagne  [http://lasagne.readthedocs.io](http://lasagne.readthedocs.io)
  
  lightweight library to build+train neural nets in Theano

All of them handle transparent use of GPU, and most of them are used in Python code/notebook

---

Example of convNet code in Keras

```python
model = Sequential()
# 1 set of (Convolution+Pooling) layers, with Dropout
model.add(Convolution2D(conv_depth_1, kernel_size, kernel_size,
    border_mode='valid', input_shape=(depth, height, width)))
model.add(MaxPooling2D(pool_size=(pooling_size, pooling_size)) )
model.add(Activation('relu'))
model.add(Dropout(drop_prob))
# Now flatten to 1D, and apply 1 Fully_Connected layer
model.add(Flatten())
model.add(Dense(hidden_size1, init='lecun_uniform'))
model.add(Activation('sigmoid'))
# Finally add a Softmax output layer, with 1 neuron per class
model.add(Dense(num_classes, init='lecun_uniform'))
model.add(Activation('softmax'))
# Training "session"
sgd = SGD(lr=learning_rate, momentum=0.8) # Optimizer
model.compile(loss='categorical_crossentropy', optimizer=sgd)
model.fit(X_train, Y_train, batch_size=32, nb_epoch=2, verbose=1,
    validation_split=valid_proportion)
# Evaluate the trained model on the test set
model.evaluate(X_test, Y_test, verbose=1)
```
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• **Transfer Learning**
• Object localization and Semantic segmentation
• Deep-Learning on 1D signal and 3D data
• Recent other image-based applications

---

Power and Generality of learnt representation

By removing *last layer(s)* (those for classification) of a convNet trained on ImageNet, one obtains a *transformation of any input image into a semi-abstract representation*, which can be used for learning SOMETHING ELSE (« transfer learning »):

– either by just **using learnt representation as features**
– or by creating new convNet output and perform **learning of new output layers + fine-tuning of re-used layers**
Transfer learning and fine-tuning

- SoA convNets trained on ImageNet are image CLASSIFIERS for one object per image
- Many object categories can be irrelevant (e.g. boat in an office)
- For each application, models are usually obtained from state-of-the-art ConvNets pre-trained on ImageNet (winners of yearly challenge, eg: AlexNet, VGG, Inception, ResNet, etc...)

Adaptation is performed by Transfer Learning, ie modification+training of last layers and/or fine-tuning of pre-trained weights of lower layers

Transfer Learning with few training examples

- Using a CNN pre-trained on a large dataset, possible to adapt it to another task, using only a SMALL training set!

<table>
<thead>
<tr>
<th># Train</th>
<th>Acc % 15/class</th>
<th>Acc % 30/class</th>
<th>Acc % 45/class</th>
<th>Acc % 60/class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sohn et al. [16]</td>
<td>35.1</td>
<td>42.1</td>
<td>45.7</td>
<td>47.9</td>
</tr>
<tr>
<td>Bo et al. [3]</td>
<td>40.5 ± 0.4</td>
<td>48.0 ± 0.2</td>
<td>51.9 ± 0.2</td>
<td>55.2 ± 0.3</td>
</tr>
<tr>
<td>Non-pretr.</td>
<td>9.0 ± 1.4</td>
<td>22.5 ± 0.7</td>
<td>31.2 ± 0.5</td>
<td>38.8 ± 1.4</td>
</tr>
<tr>
<td>ImageNet-pretr.</td>
<td>65.7 ± 0.2</td>
<td>70.6 ± 0.2</td>
<td>72.7 ± 0.4</td>
<td>74.2 ± 0.3</td>
</tr>
</tbody>
</table>


Deep-Learning: general principles + convNets, Pr. Fabien MOUTARDE, Center for Robotics, MINES ParisTech, PSL, Nov.2020
Transfer-Learning even improves performances!


Some transfer-learning applications

• Learning on simulated synthetic images + fine-tuning on real-world images
• Recognition/classification for OTHER categories or classes
• Training an objects detector (or a semantic segmenter)

• Precise localization (position+bearing) = PoseNet
• Human posture estimation = openPose
• End-to-end driving (imitation Learning)
• 3D informations (depth map) from monovision!
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)

# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(input=base_model.input, output=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit_generator(...)

Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation
• Deep-Learning on 1D signal and 3D data
• Recent other image-based applications
The high-level representation computed by last convolution layer can be analyzed for detection and localization (bounding-boxes) of all objects of interesting categories.

Region Proposal Network (RPN) on top of standard convNet.

End-to-end training with combination of 4 losses
Example of visual DETECTION & categorization with Faster_R-CNN

ConvNets are currently state-of-the-art
ALSO for visual objects detection

Object visual detection
without proposal

Solve detection as a regression problem
(“single-shot” detection)

YOLO and SSD

YOU ONLY LOOK ONCE(YOLO)
SINGLE SHOT MULTIBOX DETECTOR(SSD)

Images from: https://www.slideshare.net/Taegyunleon1/pr12-you-only-look-once-yolo-unified-realtime-object-detection

Both are faster, but less accurate, than Faster_R-CNN
Recent comparison of object detection convNets

Mask R-CNN architecture (left) extracts detailed contours and shape of objects instead of just bounding-boxes
Semantic segmentation

Convolutional Encoder-Decoder

Input

RGB Image

Convolutional Encoder-Decoder

Pooling Indices

Conv + Batch Normalisation + ReLU

Segmentation

Output

Feature extractor

Shape generator
Many competitors for DL of semantic segmentation

Many competitors for semantic segmentation by deep-learning:

- SegNet (2015)
- U-Net (2015)
- RefineNet (2016)
- ICnet (2017)
- DeepLab
- ...

VERY HOT TOPIC !!!

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
- Deep-Learning on 1D signal and 3D data
- Recent other image-based applications
Deep-TEMPORAL Convolution for multivariate time-series

MC-DCNN model
(separate 1D temporal convolution of each time-series)

Deep Gesture Recognition

Hand gesture recognition: 90% accuracy (vs 83% baseline)

Work in progress at center for Robotics of MINES ParisTech
(PhD thesis of Guillaume Devineau)

Potential applicability to other kinds of time-series!
Deep-Learning on 3D data

Possible to use:

- ConvNets on 2D images of multiple views

- ConvNet on 2D DEPTH image(s)

- Convolutions of 3D points

Multiview (Su et al., 2015)

- 3D convolutions on voxels (see next slide)

Deep-Learning: general principles + convNets, Pr. Fabien MOUTARDE, Center for Robotics, MINES ParisTech, PSL, Nov.2020 82

Deep-Learning with 3D convolutions on voxels

Voxel grid (3D + channels)

32 × 32 × 32 × 1

3D - CNN

VoxNet (Maturana et al., 2015)

3D ShapeNets (Wu et al., 2015)
Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
- Deep-Learning on 1D signal and 3D data
- Recent other image-based applications

What can Deep Convolutional Networks perform?

- Image classification
- Visual **object detection and categorization**
- **Semantic segmentation** of images
- ...

AND ALSO:
- Image-based **localization**
- Estimation of **Human pose**
- Inference of **3D** (depth) from monocular vision
- Learning **image-based behaviors**
  - End-to-end driving from front camera
  - Learning robot behavior from demonstration/imitation
PoseNet: 6-DoF camera-pose regression with Deep-Learning

Figure 4: Map of dataset showing training frames (green), testing frames (blue) and their predicted camera pose (red). The testing sequences are distinct trajectories from the training sequences and each scene covers a very large spatial extent.


Human posture estimation by Deep-Learning

Real-time estimation of Human poses on RGB video

OpenPose [Realtime Multi-Person 2D Pose Estimation using Part Affinity Field, Cao et al., CVPR’2017 [CMU]]
**Inference of 3D (depth) from monocular vision**

*Unsupervised monocular depth estimation with left-right consistency*

C Godard, O Mac Aodha, GJ Brostow - CVPR’2017 [UCL]

---

**End-to-end driving from camera by Deep-Learning**

ConvNet input: Cylindrical projection of fisheye camera

ConvNet output: steering angle

Imitation Learning from Human driving on real data

End-to-end driving via Deep *Reinforcement* Learning

*thèse CIFRE Valeo/MINES-ParisTech en cours*
Robot task learning using Reinforcement Learning

Demonstration of the task via kinesthetic teaching

Learning complex behavior with Deep Reinforcement Learning

SAC-Q active intention: LIFT(green)

SAC-Q for real robot experiments: lift and bring

Work by Google DeepMind

[Learning by Playing Solving Sparse Reward Tasks from Scratch, Riedmiller et al. (ICML’2018)]
Summary on ConvNets & Deep-Learning

- Proven advantage of learning features empirically from data
- Large ConvNets require huge amounts of labelled examples data for training
- Current research/progresses = finding efficient global architecture of ConvNets
- Enormous potential of TRANSFER-LEARNING on small datasets for restricted/specialized problems
- ConvNets also for multivariate time-series (1D temporal convolutions) and for 3D data (3D conv on voxels, etc…)
- ConvNets can potentially infer from image ANYTHING for which information is in the image (3D, movement, planning, …)

Perspectives on Deep-Learning

Next frontiers:
- Theoretical aspects
- Robustness issues (cf. adversarial examples)
- **Unsupervised** deep-learning on unlabelled data
- Deep **Reinforcement** Learning (DRL)
- Deep **Recurrent** Neural Networks (LSTM, GRU, etc…) for sequence processing (NLP!) or modeling behavior & dynamics
Any QUESTIONS ?