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 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

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

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!

What is Deep-Learning?

Learning a **hierarchy** of increasingly abstract **representations**

Increasing level of abstraction
Each stage ~ trainable feature transform

Image recognition
Pixel $\rightarrow$ edge $\rightarrow$ texton $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object

Speech
Sample $\rightarrow$ spectral band $\rightarrow$ ... $\rightarrow$ phoneme $\rightarrow$ word

Text
Character $\rightarrow$ word $\rightarrow$ word group $\rightarrow$ clause $\rightarrow$ sentence $\rightarrow$ story
Importance of « features » in classical Machine-Learning

Examples of *hand-crafted* features

**Haar features**

**Control-points features**

**HoG** (Histogram of Gradients)

Deep-Learning vs. shallow Machine-Learning

**Shallow ML using*handcrafted* features**

**DL: jointly learn classification* and features**
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 $\mathbb{R}^{1000000}$ !!

• BUT:
  – 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
Features learning (before Deep-Learning)

Speech recognition: early 90’s – 2011
- MFCC (fixed)
- Mix of Gaussians (unsupervised)
- Classifier (supervised)

Object Recognition: 2006 - 2012
- SIFT (fixed)
- HoG
- K-means (Sparse Coding)
- Pooling
- Classifier (supervised)

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)

  - For inputs with correlated dims (2D image, 1D signal, ...)
  - Supervised learning

ConvNets (2)

- 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 “summatng with sigmoid activation”

Reminder on artificial “neurons”

**PRINCIPLE**

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

\( e_i \) - neurons

\( W_{ij} \) - connection weights

\( W_{0j} \) - bias

**ACTIVATION FUNCTIONS**

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

activation map

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

At sliding position \( i,j \)

\[ \sigma(i,j) = b + \vec{W} \cdot \vec{x}_{ij} \]

with \( \vec{x}_{ij} \) = 5x5 image patch in 3 colors
\( \rightarrow \) vector of dim 75, as filter coeffs in \( \vec{W} \)
### Convolution in action


<table>
<thead>
<tr>
<th>Input Volume (pad 1) (7x7x3)</th>
<th>Filter W0 (3x3x3)</th>
<th>Filter W1 (3x3x3)</th>
<th>Output Volume (3x3x2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x[1,1,0])</td>
<td>(w_0[1,1,0])</td>
<td>(w_0[1,1,1])</td>
<td>(o[1,1,0])</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>-1 0 1</td>
<td>0 1 -1</td>
<td>2 3 3</td>
</tr>
<tr>
<td>0 0 0 1 0 2 0</td>
<td>0 0 1</td>
<td>0 -1 1</td>
<td>3 7 3</td>
</tr>
<tr>
<td>0 1 0 2 0 1 0</td>
<td>1 -1 1</td>
<td>0 -1 1</td>
<td>8 10 3</td>
</tr>
<tr>
<td>0 1 0 2 2 0 0</td>
<td>-1 0 1</td>
<td>-1 0 0</td>
<td>-8 -8 -3</td>
</tr>
<tr>
<td>0 2 0 0 2 0 0</td>
<td>1 -1 1</td>
<td>1 -1 0</td>
<td>-3 1 0</td>
</tr>
<tr>
<td>0 2 1 2 2 0 0</td>
<td>0 1 0</td>
<td>1 -1 0</td>
<td>-3 8 -5</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example of typical results of convolution

![Input Image](image-url)
Each convolution FILTER is one set of neuron parameters

Each convolution LAYER is a set of \( \sim \text{imageSize} \) neurons, but they all have same \textit{SHARED} weights (perform \textit{SAME} convolution)

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

\( W_0 = \text{"bias"} \)
\( f = \text{activation function} \)

Convolutional v.s. Fully-connected
Convolutional layers

A convNet: succession of Convolution+activation Layers

One “activation map” for each convolution filter

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

Convolution of convolutions!
### 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: $\text{max}$, average, $L_p$, ...

**Example:** 2x2 pooling, stride 2
Final classification layer: just a classical MLP

CNN called LeNet by Yann LeCun (1998)

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

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
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 = optimizing values of weights & biases
Method used = gradient descent

→ 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

- A practical Application of Chain Rule
- Backprop for the state gradients:
  - $dC/dX_i = dC/dX_i \cdot dX_i/dX_{i-1}$
  - $dC/dX_i-1 = dC/dX_i \cdot dF_i(X_{i-1},W_i)/dX_{i-1}$
- Backprop for the weight gradients:
  - $dC/dW_i = dC/dX_i \cdot dX_i/dW_i$
  - $dC/dW_i = dC/dX_i \cdot dF_i(X_{i-1},W_i)/dW_i$
Recall of back-prop principle

Smart method for efficient computing of gradient (w.r.t. weights) of a Neural Network cost function, based on chain rule for derivation.

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 \( \frac{dQ_m}{dW_{ij}} \)?

**Backprop through layers:**
chain rule derivative computation

\[
\sigma_i \rightarrow y_i \rightarrow \sigma_j \rightarrow y_j \rightarrow \sigma_k \rightarrow \text{Let } \delta_j = \frac{dE_m}{d\sigma_j}. \text{ Then } W_{ij}(t+1) = W_{ij}(t) - \lambda(t)y_i \delta_j
\]

If neuron \( j \) is output, \( \delta_j = (\frac{dE_m}{d\sigma_j}) = (\frac{dE_m}{dy_j})(\frac{dy_j}{d\sigma_j}) \) 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, \( \delta_j = (\frac{dE_m}{d\sigma_j}) = \sum_k (\frac{dE_m}{d\sigma_k})(\frac{d\sigma_k}{d\sigma_j}) = \sum_k \delta_k \frac{d\sigma_k}{d\sigma_j} = \sum_k \delta_k W_{jk}(\frac{dy_j}{d\sigma_j}) \)

so \[ \delta_j = (\sum_k W_{jk} \delta_k)f'(\sigma_j) \] 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 $1/\sqrt{\text{nblInputs}}$
• 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:
  – 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 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:

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

\( \Rightarrow \) 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 \( \sim 0.5 \)),

or re-installed with last values of weights
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**: 1\textsuperscript{st} 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) → 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 !...
Input: 224x224x3 image

ZFnet
[Zeiler & Fergu, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

GoogleNet
[Szegedy et al., 2014]

Inception module
ILSVRC 2014 winner (6.7% top 5 error)
ResNet (Residual Net), by Microsoft [He et al., 2015]

- ILSVRC 2015 \textit{large} winner in 5 main tracks (3.6% top 5 error)
- 152 layers!!
- But novelty = "skip" connections

ResNet global architecture

- 2-3 weeks of training on 8 GPU machine !!
- However, at runtime faster than a VGGNet! (even though it has 8x more layers)
Summary of recent ConvNet history

But most important is the choice of **ARCHITECTURAL STRUCTURE**

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)
- **Caffe** [http://caffe.berkeleyvision.org/](http://caffe.berkeleyvision.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
- **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/)

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 a 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!

Transfer-Learning even improves performances!

[ transfer learning improves performance ]

[Yosinski, Clune, Bengio, Lipson, "How transferable are features in deep neural networks?", ICML'2014]
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!

Transfer Learning code example in Keras

```python
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(...)```
Deep-Learning for visual object DETECTION

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)**

Both are faster, but less accurate, than Faster R-CNN

---

Recent comparison of object detection convNets

![Graph showing performance comparison of different object detection methods.](https://www.slideshare.net/TaegyunLeon/pr12-you-only-look-once-yolo-unified-realtime-object-detection)
Mask R-CNN architecture (left) extracts detailed contours and shape of objects instead of just bounding-boxes.

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-serie)
Deep Gesture Recognition

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

  Multiview
  (Su et al., 2015)

• ConvNet on 2D DEPTH image(s)

• Convolutions of 3D points

  PointNet++ (Qi et al., 2017)
  PointCNN (Li et al., 2018)

• 3D convolutions on voxels (see next slide)
Deep-Learning with 3D convolutions on voxels

Voxel grid (3D + channels) 32 × 32 × 32 × 1

3D - CNN

3D object

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

Input RGB Image → Convolutional Neural Network (GoogLeNet) → 6-DOF Camera Pose

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
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)

Van + diff = Ostrich!!

• **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?