MVA - Discrete Inference and Learning Lecture 7
-
Modern Learning

Yuliya Tarabalka
Inria Sophia Antipolis-Méditerranée - TITANE team
Université Côte d'Azur - France

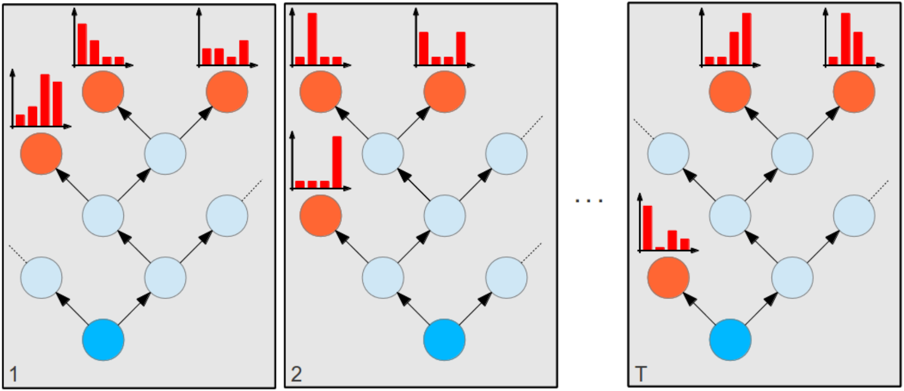
Overview

1. Random Forests
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
2. ConvNets
   - MLP to ConvNet
   - Convolution
   - Architectures
   - Auto Encoder
   - Frameworks
3. Dense labelling with CNNs
   - Fully convolutional networks
   - Enhancing outputs with RNNs
   - Yielding high-resolution outputs

Overview

1. Random Forests
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
2. ConvNets
   - MLP to ConvNet
   - Convolution
   - Architectures
   - Auto Encoder
   - Frameworks
3. Dense labelling with CNNs
   - Fully convolutional networks
   - Enhancing outputs with RNNs
   - Yielding high-resolution outputs

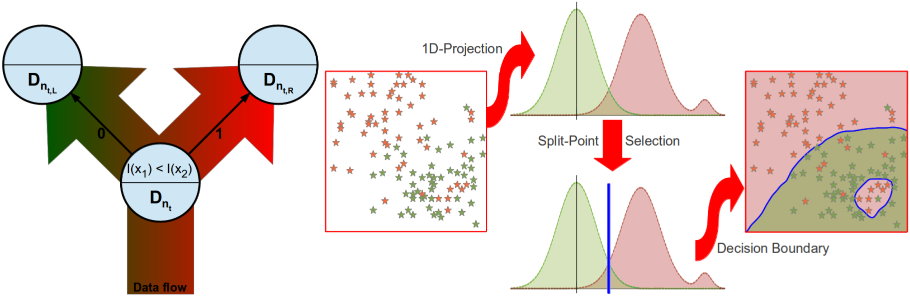
Recap



Set of decision trees

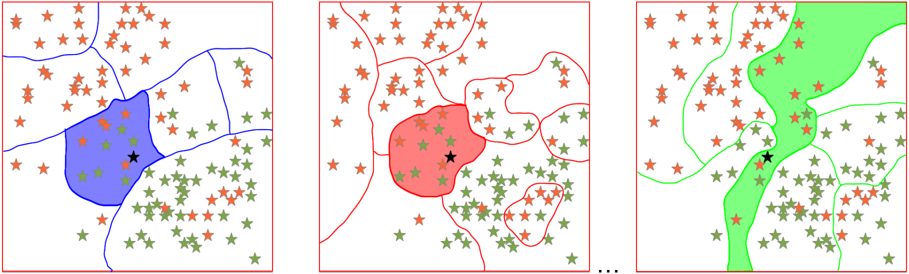
- Each tree t generated from training data $\mathrm{D}_t \subseteq \mathrm{D} \subset \mathbb{D}$
- Creation of one tree independent of all other trees
- Based on random processes to produce diverse set of trees

Data Propagation



- Data enters each tree in root node
- Each non-terminal / internal node performs a (simple) binary test
- Data propagation is based on test outcomes

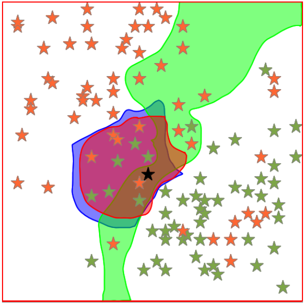
Tree Fusion



$t = 1$ $t = 2$ ... $t = T$

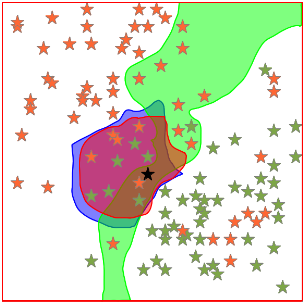
- Query sample is propagated through all trees
- Reaches exactly one leaf in each tree
- Information about target variable assigned to these leafs need to be combined

Tree Fusion



How to combine information assigned to individual leafs?

Tree Fusion



Random Forests

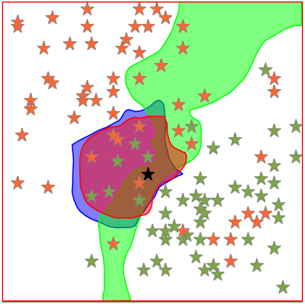
The simplest approach: voting scheme

$$P(c|x) = \frac{1}{T}\sum_{t=1}^{T}\delta(M(n_t), c),$$

$M(n)$ is dominant class of samples in leaf node n,
$\delta(\cdot,\cdot)$ is a Kronecker delta function,
$\delta(a, b) = 1$ if $a = b$, 0 otherwise

Drawback: Does not store category distribution in every leaf

Tree Fusion



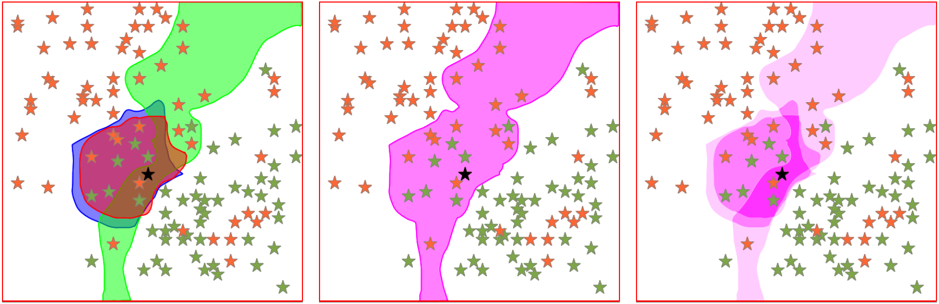
Randomized Trees

$$P(c|x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|x)$$

$P_t(c|x)$ is class posterior in leaf node n_t

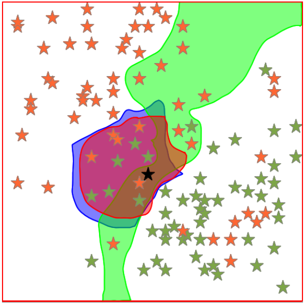
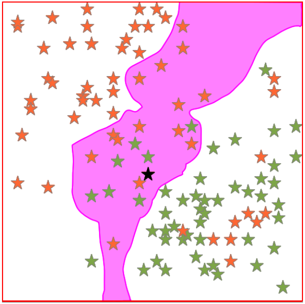
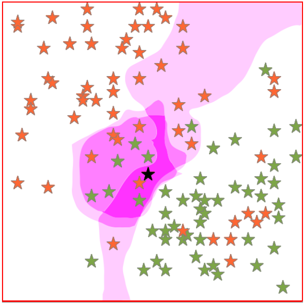
Drawback: Confidence about a correctness of estimation is lost

Tree Fusion



Randomized Trees	Fuse before estimation:	
$$P(c\|x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c\|x)$$	$$D_{L_x} = \bigcup_{t=1}^{T} D_{n_t}$$	$$P(c\|x) = \frac{P(L_x\|c)P(c)}{P(L_x)}$$
$P_t(c\|x)$ is class posterior in leaf node n_t	(Multi-set)	(Details in [Hänsch, 2014])

Tree Fusion

		
Randomized Trees	Fuse before estimation:	Weighted fusion.
$$P(c\|x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c\|x)$$	$$D_{L_x} = \bigcup_{t=1}^{T} D_{n_t}$$	$$P(c\|x) = \sum_{t=1}^{T} w_t P_t(c\|x)$$
$P_t(c\|x)$ is class posterior in leaf node n_t	(Multi-set)	(Details in [Hänsch, 2014])

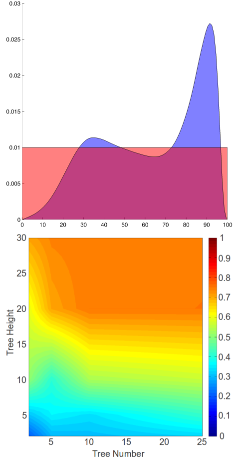
Random Forests - Split point selection

How to select split points?

Random Forests - Random split point selection

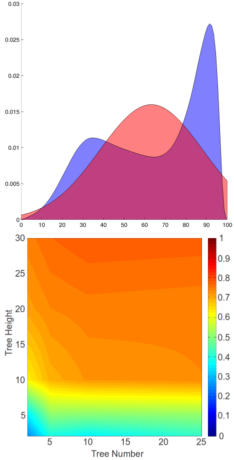
Uniform sampled

$\theta \sim U(\min(\hat{D}), \max(\hat{D}))$



Gaussian sampled

$\theta \sim N(\mu(\hat{D}), \sigma(\hat{D}))$



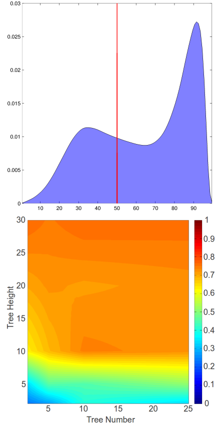
Random Forests - Naive split point selection

- Determine an optimal split point under usage of the marginal distribution of the data
  - Both labeled and unlabeled data points can be used
  - Fast to compute

Random Forests - Naive split point selection

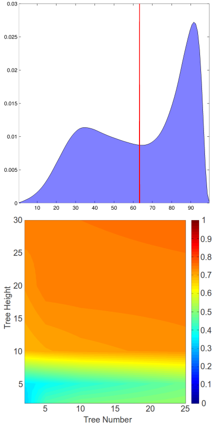
Interval center

$$\theta = \frac{min(\hat{D})+max(\hat{D})}{2}$$



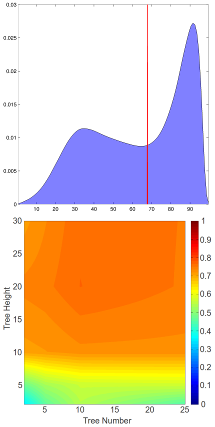
Mean value

$$\theta = \frac{1}{|\hat{D}|}\sum_{x\in\hat{D}}\hat{x}_i$$



Median value

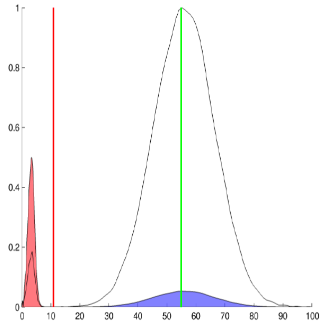
$$\theta = median(\hat{D})$$



Random Forests - Naive split point selection

- Determine an optimal split point under usage of the marginal distribution of the data
  - Both labeled and unlabeled data points can be used
  - Fast to compute
- No class-specific knowledge is used
  - Tend to give sub-optimal results, since all label-dependent (task-specific) information is ignored
  - Label-independent split points are not optimal in a Bayesian sense

Random Forests - Split point selection



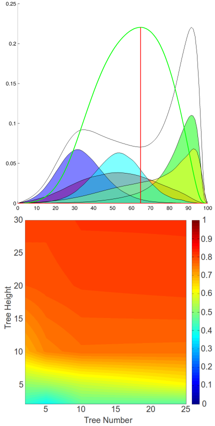
Class likelihood of two classes in red and blue, respectively,
Along with
Label-independent (green) and label-dependent (red) split points

Random Forests - Label-dependent split point selection

Max. drop of impurity $\theta = \arg\min_{\hat{\theta}} [P_L I(n_L) + P_R I(n_R) - I(n)]$

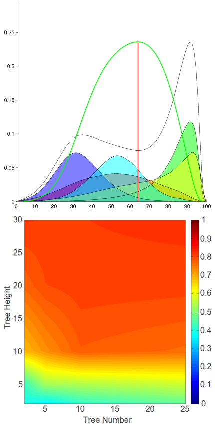
Entropy

$I(n) = -\sum_c P(c|n) \cdot \log P(c|n)$



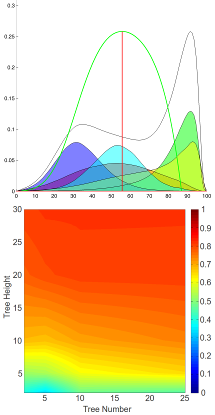
Gini

$I(n) = 1 - \sum_c P(c|n)^2$

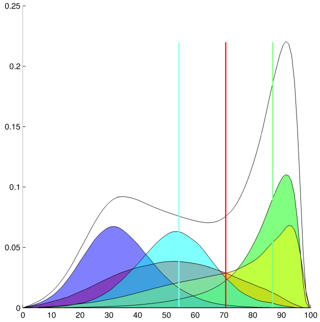


Misclassification

$I(n) = 1 - \max_c P(c|n)$



Random Forests - Split point selection



- Other possibilities available
  → Combine label-dependent & label-independent optimization methods
- Need for computational efficiency since selection is performed thousand to million times during training
- Avoid exhaustive search

Random Forests - Node optimization

- Generate m split candidates
  $\rightarrow$ "Traditionally": $m = \sqrt{d}$, where d is data dimension
  $\rightarrow$ "Modern" approaches: $m \approx 10^5$
  $\rightarrow$ Usually even $m = 2$ leads to performance increase
  $\rightarrow$ Trade-off between high performance and high correlation
- Select best split, reject all others
- Measure optimality of a split
  - Classification: "Purity" of child nodes (e.g. Gini, entropy, etc.)
  - Regression: e.g. variance
  - In general: How much better is the estimation of the child nodes (as a weighted average) than parent nodes?

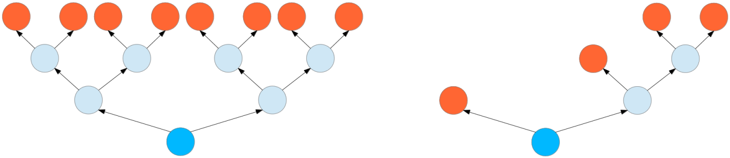
Random Forests - Node optimization

- Different energy functions allow simultaneous optimization of different targets
- Common example: Classification (Object class) **and** regression (Object position)
- e.g. **Hough Forests**[Gall et al., 2011]
  - Training Data: $D = \{P_i = (I_i, c_i, d_i)\}$
  - Randomly decide for one of two energy functions:
    Entropy of posterior: $U_1(A) = -|A| \cdot \sum_c P(c|A) log(P(c|A))$
    Variance of offset vectors: $U_2(A) = \sum_c \sum_{d \in D_C^A} ||d - \bar{d}_C^A||$
  - Select best test t according to $\arg\min_t(U_*(\{P_i|t = 0\}) + U_*(\{P_i|t = 1\}))$
  - Use offset vectors and class posterior to perform Hough voting during prediction

Random Forests - Structured prediction

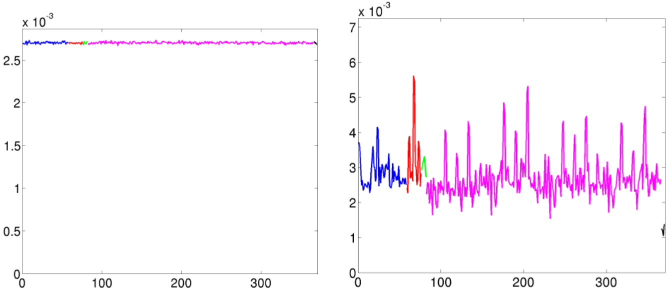
- Image data is structured
- Exploited already during structured projection within node tests
- Target variable can be structured as well
  $\rightarrow$ Offset vectors, label patch
- Enriched spatial estimate for image labeling
- Disadvantage: Increased memory footprint

Random Forests - Interpretation



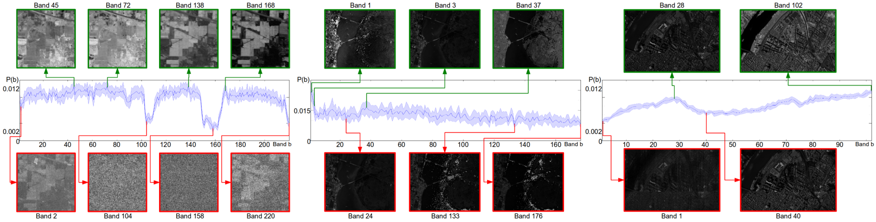
- Is maximum tree height reached?
- How balanced is a tree? $\frac{\#\text{nodes}}{2^{H+1}-1}$
- How large is largest leaf? $1 - \frac{\max_{n_t} |D_{n_t}|}{|D|}$
- How pure is largest leaf? $I(n^*)$ with $n^* = \arg\max_{n_t} |D_{n_t}|$
- Out-of-bag estimate for generalization error

Random Forests - Feature relevance



- RF for PolSAR image labeling, roughly 360 image features as input:
  (PolSAR-blue, SAR-red, color-green, grayscale-magenta, binary-black)
- Each feature has same probability to be used (as seen on the left)
- Each feature is actually selected by the RF with very unequal frequency (as seen on the right)
- Features that have been used often are more important / descriptive for the task at hand

Random Forests - Feature relevance



[R. Hänsch, 2015a]

- RF for hyperspectral image labeling, $>$ 200 spectral bands as input
- Each band has same probability to be used
- Each band is actually selected by the RF with very unequal frequency
- Bands that have been used often are more important / descriptive for the task at hand

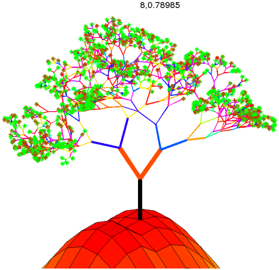
Random Forests - Visualization

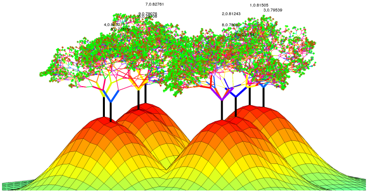
Important Characteristics of Random Forests

- Forest Level
  - Strength of the whole forest
    $\rightarrow$ e.g. classification accuracy
  - Correlation between trees
    $\rightarrow$ e.g. correlation of classification maps
- Tree Level
  - Strength of the individual tree
    $\rightarrow$ e.g. based on out-of-bag error
  - Structural layout of individual trees
    $\rightarrow$ e.g. balanced vs. degenerated chain
- Node Level
  - Node features
    $\rightarrow$ e.g. size, split dimension, drop of impurity, leaf impurity, etc.

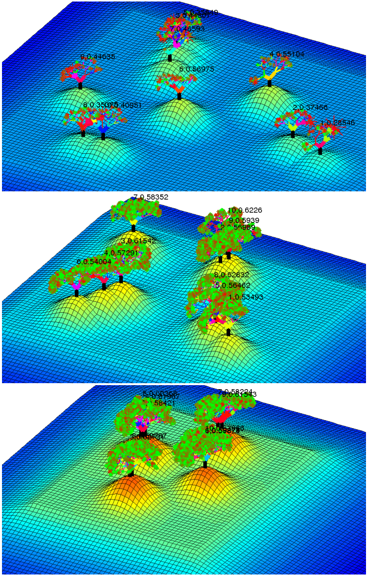
Random Forests - Visualization

- Branch
  - Color:
    (e.g.) split dimension
  - Thickness:
    Number of samples
  - Length:
    $l_{h+1}^{L/R} = l_h \cdot \kappa_2 \cdot ((f_{\max} - f_{\min}) + f_{\min})$
  - Orientation:
    $(\alpha, \beta)_{h+1}^{L/R} = (\alpha, \beta)_h \pm (30^\circ, \kappa_1 \cdot 45^\circ)$
- Leaf
  - Color: Leaf impurity
  - Size: Leaf size
- 2D position
  Based on pairwise correlation





Random Forests - Visualization



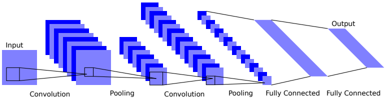
- Increasing tree height
- Trees getting higher
- Leafs getting purer
- Trees getting stronger
- Trees correlate stronger
- Forest gets stronger
- [R. Hänsch, 2015b]

Random Forests - Practical Considerations

- GPU implementations available
- Not all data samples in a node have to be used to define / select split point
- Accuracy usually grows faster with tree height than tree number
- But: Tree height limited by amount of training data
- Use features that are as diverse as possible
- Use simple split point definitions in combination with node optimization (i.e. selection)
- Check tree properties / visualize

Overview

1. Random Forests
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
2. ConvNets
   - MLP to ConvNet
   - Convolution
   - Architectures
   - Auto Encoder
   - Frameworks
3. Dense labelling with CNNs
   - Fully convolutional networks
   - Enhancing outputs with RNNs
   - Yielding high resolution



Recap MLP

MLPs

- Provide a mapping from $\mathcal{X} \rightarrow \mathcal{Y}$, i.e. from a feature space (usually $\mathcal{X} \equiv \mathbb{R}^n$) to a label space $\mathcal{Y}$
- Are based on concatenation of "simple" functions that depend on parameters (i.e. weights)
- Are optimized by gradient descent (and its modern extensions)

Recap MLP

MLPs

- Provide a mapping from $\mathcal{X} \rightarrow \mathcal{Y}$, i.e. from a features space (usually $\mathcal{X} \equiv \mathbb{R}^n$) to a label space $\mathcal{Y}$
- Are based on concatenation of “simple” functions that depend on parameters (i.e. weights)
- Are optimized by gradient descent (and its modern extensions)
- Work great, BUT:

ConvNets and Deep Learning

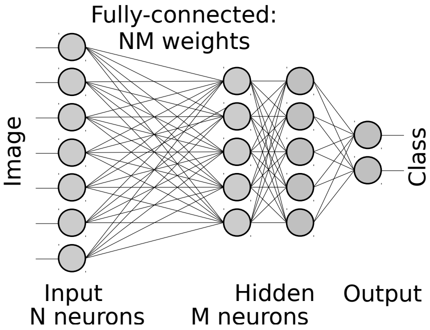
- Used by
  - Facebook: Automatic tagging
  - Google: Photo search
  - Amazon: Recommendations
  - Pinterest: Home feed personalization
  - Instagram: Search
- Buzzwords
  - Deep Learning, Deep Networks
  - Convolutional Neural Networks (CNNs), Convolutional Networks (ConvNets)
  - Note: There are more Deep Networks / Deep Learning approaches than ConvNets

ConvNets and Deep Learning

"CNNs are inspired by biological principles in the visual cortex."

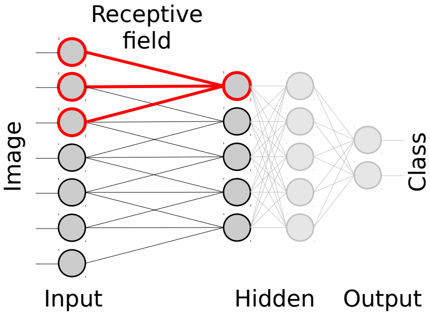
- Small regions of cells sensitive to specific regions within the visual field.
- 1962, Hubel and Wiesel
  - Neuronal cells fire only in the presence of certain structures e.g. edges of a specific orientation
  - Organized in columns
- Good selling point, BUT:
  - Extracting image features is neither new, nor the main point of ConvNets
  - Training works very differently

From FullyConnected (MLP) to Convolution (ConvNet)



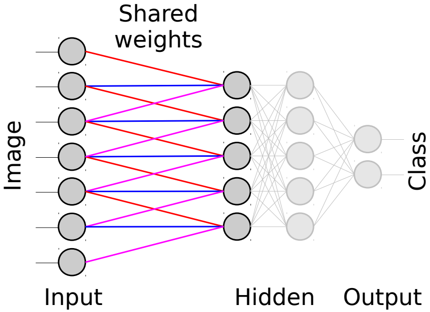
- Multiple layers of units
- All-to-all connection between two adjacent layers
- No lateral connections
- A tremendous amount of parameters in case of images $\rightarrow$ Untrainable

From FullyConnected (MLP) to Convolution (ConvNet)



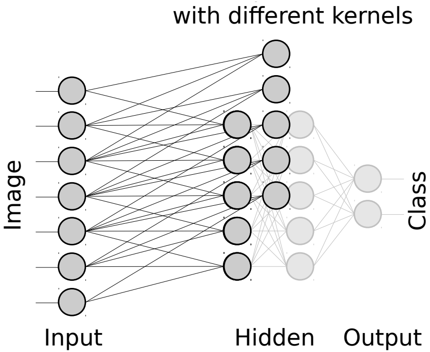
- Set most weights to zero and thus delete most connections and decrease parameters.

From FullyConnected (MLP) to Convolution (ConvNet)



- Set most weights to zero and thus delete most connections and decrease parameters.
- Use same values for weights of different neurons within a layer.
- The multiplication of the input with identical weights for different neurons corresponds to a convolution.
- The kernel of this convolution is automatically learned.

From FullyConnected (MLP) to Convolution (ConvNet)



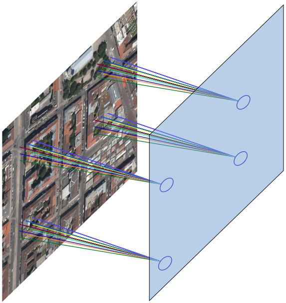
- Set most weights to zero and thus delete most connections and decrease parameters.
- Use same values for weights of different neurons within a layer.
- The multiplication of the input with identical weights for different neurons corresponds to a convolution.
- The kernel of this convolution is automatically learned.
- Use multiple convolutional layers to enable different kernels to be learned.

Convolutional neural networks (CNNs)

- Input: the image itself
- {*Convolutional* layers + *pooling* layers}* + MLP
- Jointly learn to extract features & conduct classification

Convolutional layer

Learned convolution filters → feature maps



Special case of fully connected layer:

- Only local spatial connections
- Location invariance

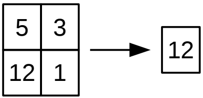
⇒ Makes sense in image domain (or text, time series,...)

Convolutional neural networks (CNNs)

Pooling layers

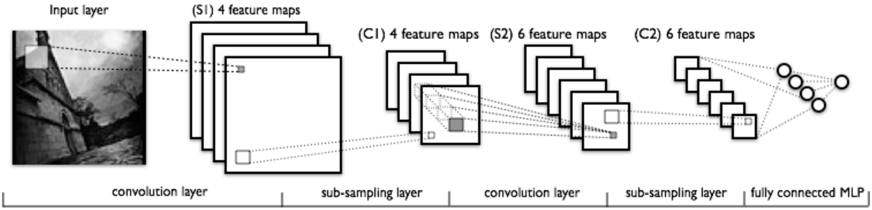
Subsample feature maps

- Increase *receptive field* ☺
- Downgrade resolution
  - Robustness to spatial variation ☺
  - Not good for *pixelwise* labeling ☹



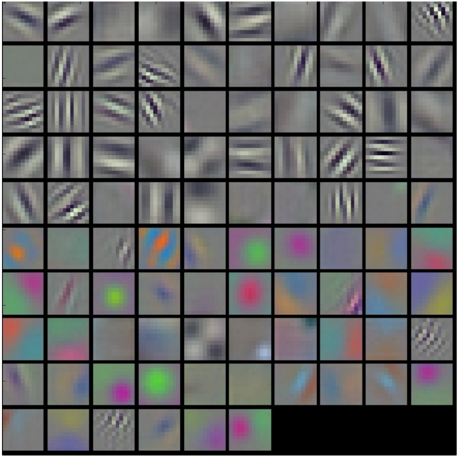
Max pooling

Overall categorization CNN



Source: deeplearning.net

Example of First Level Filters



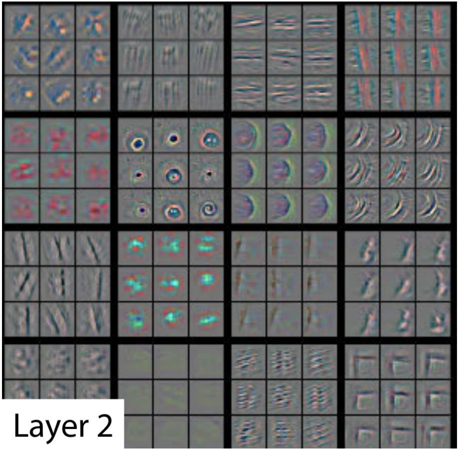
- Learned kernels of first convolutional layer of a ConvNet (AlexNet).
- Correspond mostly to edges and corners of different orientations.
- Note: Grouping is caused by network architecture (two independent streams were used to handle the large amount of data).

Example of Higher Level Filters

Layer 1

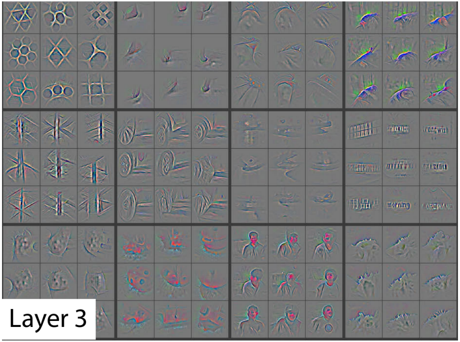
- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

Example of Higher Level Filters



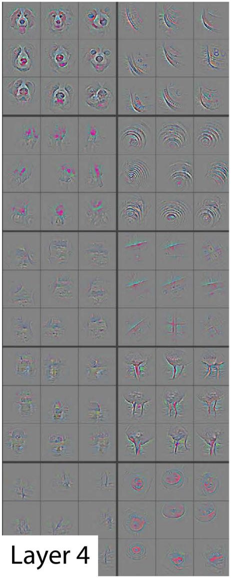
- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

Example of Higher Level Filters



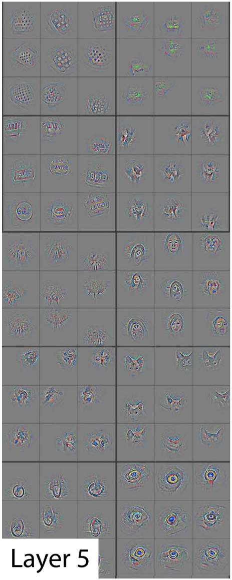
- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

Example of Higher Level Filters



- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

Example of Higher Level Filters



- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

Architectures

LeNet (1998)

- One of the first successful applications of ConvNets
- Digital digit / character recognition

Architectures

LeNet (1998)

- One of the first successful applications of ConvNets
- Digital digit / character recognition

AlexNet (2012)

- Similar to LeNet, but deeper and bigger
- Stacked conv-layers
- Image classification (ImageNet Large-Scale Visual Recognition Challenge)
- Trained on 15 million annotated images from over 22,000 categories
- Trained on two GTX 580 GPUs for five to six days

Architectures

LeNet (1998)

- One of the first successful applications of ConvNets
- Digital digit / character recognition

AlexNet (2012)

- Similar to LeNet, but deeper and bigger
- Stacked conv-layers
- Image classification (ImageNet Large-Scale Visual Recognition Challenge)
- Trained on 15 million annotated images from over 22,000 categories
- Trained on two GTX 580 GPUs for five to six days

ZF Net (2013)

- Similar to AlexNet
- Trained on 1.3 million annotated images
- Trained on a GTX 580 GPU for twelve days

Architectures

VGG Net (2014)

- Simple and deep: Only 3x3 filters and 2x2 pooling
- Stacked conv-layers to increase effective receptive field size
- Used Caffe toolbox
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks

Architectures

VGG Net (2014)

- Simple and deep: Only 3x3 filters and 2x2 pooling
- Stacked conv-layers to increase effective receptive field size
- Used Caffe toolbox
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks

GoogLeNet (2015)

- 22 layers
- Proposed inception module: Running multiple filter operations in parallel
- 12x fewer parameters than AlexNet
- Trained on multiple high-end GPUs for a week

Architectures

VGG Net (2014)

- Simple and deep: Only 3x3 filters and 2x2 pooling
- Stacked conv-layers to increase effective receptive field size
- Used Caffe toolbox
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks

GoogLeNet (2015)

- 22 layers
- Proposed inception module: Running multiple filter operations in parallel
- 12x fewer parameters than AlexNet
- Trained on multiple high-end GPUs for a week

Microsoft ResNet (2015)

- 152 layers
- Trained on an 8 GPUs for two to three weeks
- 3.6% error on ImageNet LSVRC (AlexNet: 15.4%)

Common Architectures and Tricks

- Designing good architecture somewhat tricky
- Some designs, or parts of designs, exist that work well
- Usually a good idea to look at papers of common architectures
  - Most of the time, at least some intuition or motivation for choice of layers

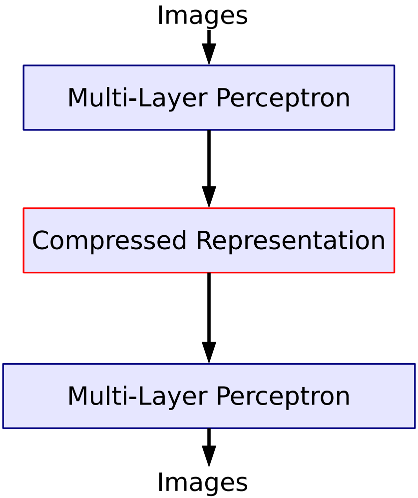
Network in a Network

- Alternate between actual conv layers, and conv layers of size 1x1
- Use the (per pixel) FC layers to compress (reduce channels)
  - Next (actual) convolution faster
  - Deeper network, but less parameters

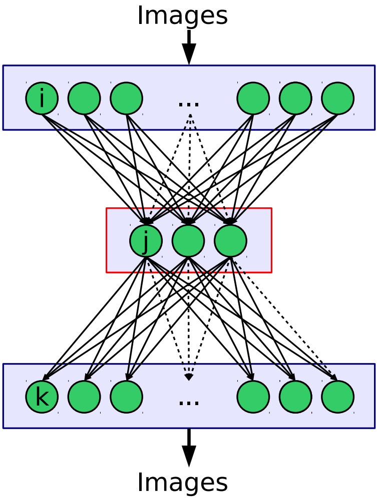
Inception Module

- Used by Google
- Multiple versions
- Compilation of multiple ideas
- Network in a Network
- Use of small filters (only 3x3)
- Using two 3x3 filters same receptive field size as one 5x5 filter
    - But less parameters

(Convolutional) Auto Encoder

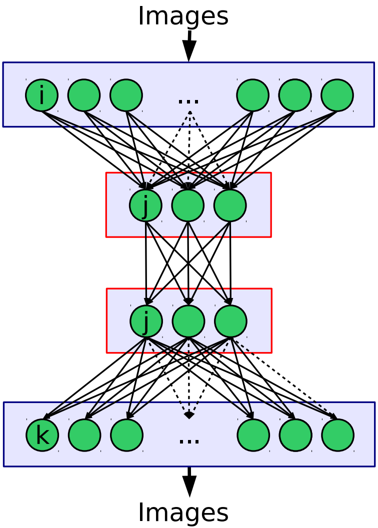


(Convolutional) Auto Encoder



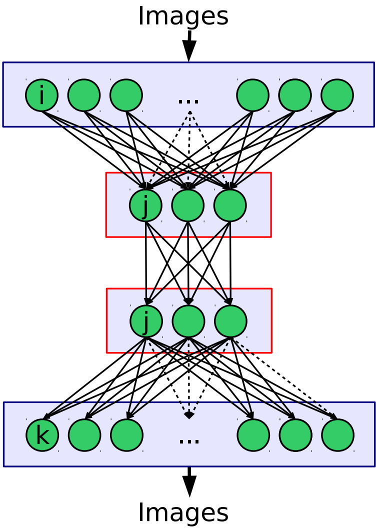
(Convolutional) Auto Encoder

- Stacked Autoencoder



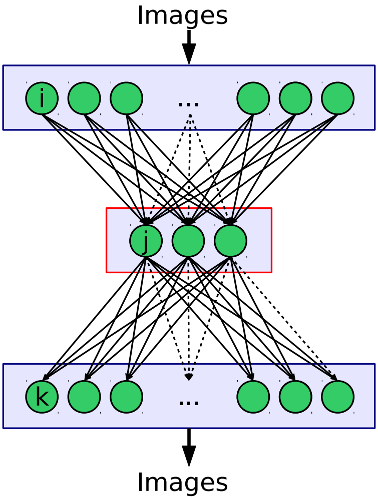
(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients



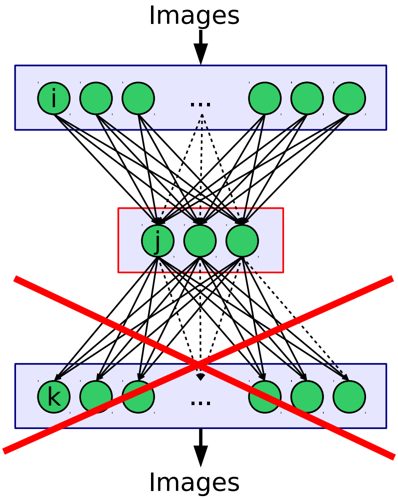
(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training

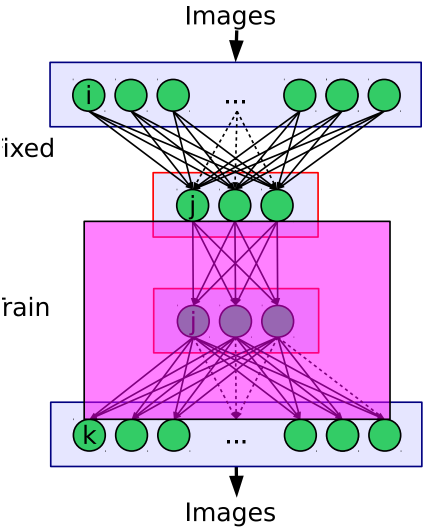


(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



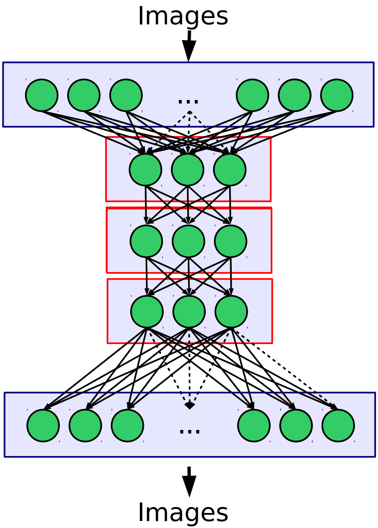
(Convolutional) Auto Encoder



- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training

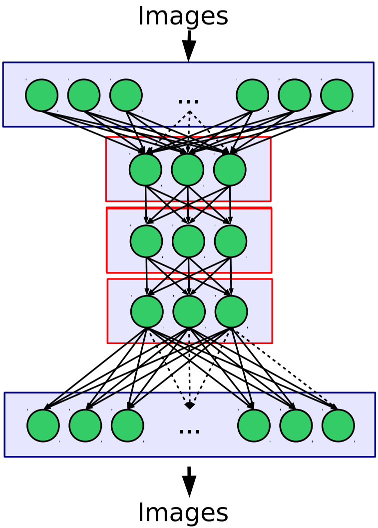
(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



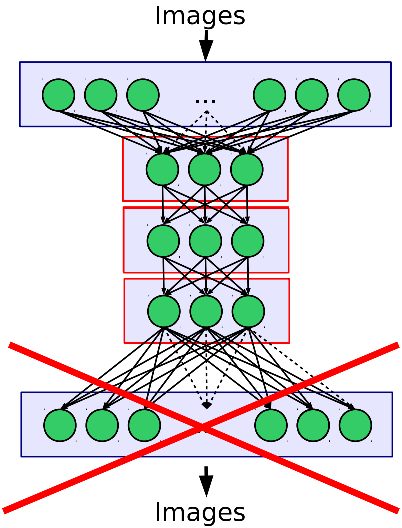
(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training
- Application: Deep Learning



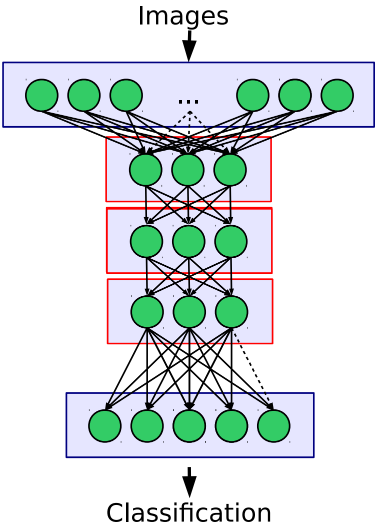
(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training
- Application: Deep Learning



(Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training
  $\rightarrow$ Learn "reasonable" features from unlabeled data
- Application: Deep Learning
  $\rightarrow$ Supervised learning (via Backpropagation) only as refinement



Frameworks

- Implementing fast, multi-channel convolutions just as hard as implementing fast matrix multiplications
- *Use existing tools!*
    - Caffe
    - Tensorflow
    - Torch
- For larger datasets you want to use a (good) GPU!

Caffe

Caffe

Deep learning framework by BAIR

- Started by Yangqing Jia at UC Berkeley
- Maintained by Berkeley AI Research and many contributers
- Backend in C++, frontends for Python and Matlab
- `http://caffe.berkeleyvision.org/`
- `https://github.com/BVLC/caffe`
- Version 2 now available

Tensorflow

- Developed by Google Brain team
- Python frontend
- `https://www.tensorflow.org/`
- `https://github.com/tensorflow`

Torch

- Lua frontend
- `http://torch.ch/`
- `https://github.com/torch/torch7`

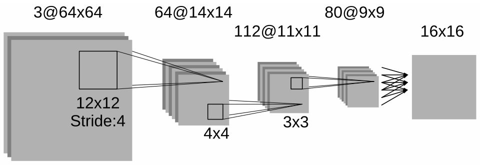
Outline

1. Random Forests
   - Fusion
   - Node Tests
   - Interpretation
   - Application Tips
2. ConvNets
   - MLP to ConvNet
   - Convolution
   - Architectures
   - Auto Encoder
   - Frameworks
3. Dense labelling with CNNs
   - Fully convolutional networks
   - Enhancing outputs with RNNs
   - Yielding high-resolution outputs

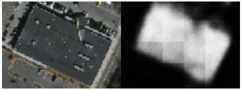
Example: *dense* labeling with CNNs in remote sensing

Pioneering works:

1. Predict an entire patch centered in input patch (Mnih, 2013)



- Allows to learn "in-patch location" priors
  → Patch border artifacts



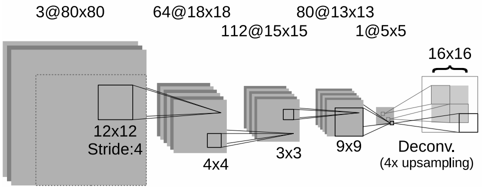
2. Predict the central pixel in the patch and shift one by one
   (e.g., Paisitkriangkrai et al., CVPR Earthvision 2015)
   - Too many redundant computations

State of the art: fully convolutional network (FCN)

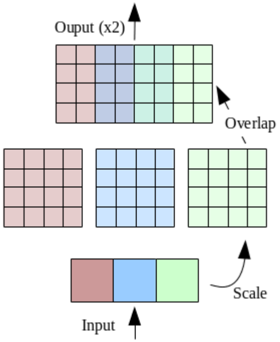
Fully convolutional networks (FCNs)

[Long et al., CVPR 2015]

- Convolutions & subsampling
- “Deconvolutional” layer to upsample



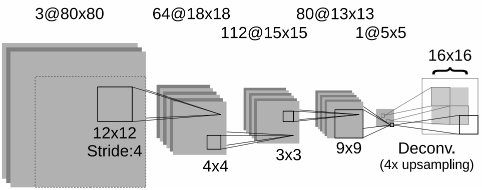
Proposed FCN for remote sensing



Deconv. layer

[Maggiori et al, TGRS 2017]

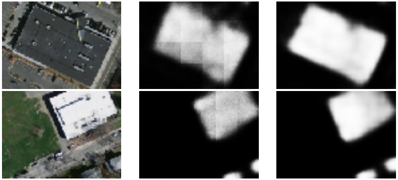
State of the art: fully convolutional network (FCN)



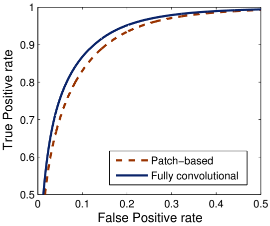
- Output size varies with input size (with fixed number of parameters)
- Location invariant (same logic used to compute every output)
- Avoid redundant computations
- *Especially* relevant in remote sensing (arbitrary tiling, azimuth)

FCN: experiment

- Patch artifacts removed by construction
- More accurate
- 10x faster



Massachusetts dataset (Mnih, 2015)



Once again...

Imposing sensible restrictions

- improves the learning process,
- reduces execution times.

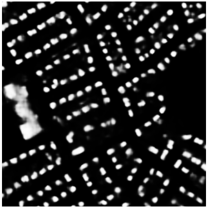
FCN: experiment

Massachusetts dataset

[Dataset: Mnih, 2013]

Color input

Reference



FCN

SVM

- Classification of 22.5 km^2 (1 m resolution): 8.5 seconds

Dealing with imperfect training data

Frequent misregistration/omission in large-scale data sources:

Pléiades image + OpenStreetMap (OSM) over Loire department

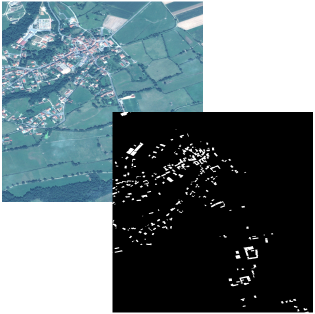
Possible strategy

Two-step training process:

1. Pretrain on large amounts of imperfect data
   → Learn dataset generalities
2. Fine-tune on a small piece of manually labeled reference

Imperfect training data: experiment

1. Pretrain on 22.5 km^2 Pléiades + OpenStreetMap data
2. Fine-tune on a manually labeled tile (2.5km^2, 3000$\times$3000 px.)



Fine-tuning tile

Close-up

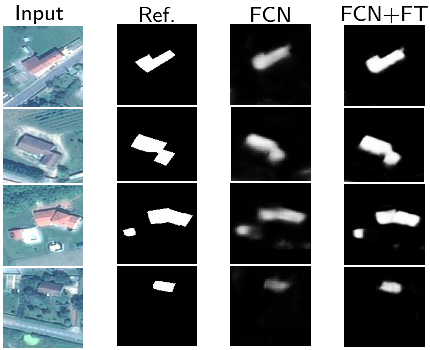
E. Maggiori, Y. Tarabalka, G. Charpiat, P. Alliez. "Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification", TGRS 2017.

Imperfect training data: experiment

Test on a different manually labeled tile

Test tile

Results



Method	Accuracy	AUC*	IoU
FCN	99.13%	0.98154	47%
FCN + FT	99.57%	0.99836	72%

*AUC: area under the ROC curve

Concluding remarks

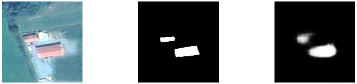
- FCNs have now become the standard dense labeling architecture

Recognition/localization trade-off

Subsampling:

- increases the receptive field (improving recognition)
- reduces resolution (hampering localization)

⇒ “Blobby” objects

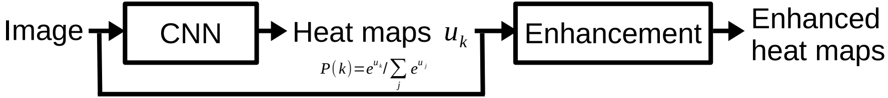


Input Ref. CNN

Solutions

1. Post-process the CNN’s output (e.g., CRF)
2. Use innovative (e.g., multiscale) architectures

Enhancing CNNs' outputs



Recent approaches

- CNN + Fully connected CRF (Chen et al., ICML 2015)
- CNN + Fully connected CRF as RNN (Zheng et al., CVPR 2015)
- CNN + Domain transform (Chen et al., CVPR 2016)

In remote sensing:

- CNN + CRF (Paisitkriangkrai et al., CVPR Worshops 2015)
- CNN + Fully connected CRF (Marmanis et al., ISPRS 2015; Sherrah 2016,...)

Goal

Learn iterative enhancement process

Partial differential equations (PDEs)

- **One strategy:** progressively enhance the score maps by using partial differential equations
- Given heat maps u_k, image I:
  - Heat flow
    (Smooths out u_k)

$$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(\nabla u_k(x))$$

- **Divergence** represents the volume density of the outward flux of a vector field from an infinitesimal volume around a given point

Partial differential equations (PDEs)

Given heat maps u_k, image I:

- Heat flow
  (Smooths out u_k)
  $$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(\nabla u_k(x))$$
- Perona-Malik
  Edge-stopping function $g(\nabla I, x)$
  $$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(g(\nabla I, x)\nabla u_k(x))$$
- Anisotropic diffusion
  Diffusion tensor $D(I, x)$
  $$\frac{\partial u_k(x)}{\partial t} = \mathrm{div}(D(\nabla I, x)\nabla u_k(x))$$
- Geodesic active contours
  Edge-stopping function $g(\nabla I, x)$
  $$\frac{\partial u_k(x)}{\partial t} = |\nabla u_k(x)|\mathrm{div}\left(g(\nabla I, x)\frac{\nabla u_k(x)}{|\nabla u_k(x)|}\right)$$
- ...

Partial differential equations (PDEs)

- Different PDE approaches can be devised to enhance classification maps
- Several choices must be made to select the appropriate PDE and tailor it to the considered problem
  - For example, edge-stopping function $g(\nabla I, x)$ must be chosen

Partial differential equations (PDEs)

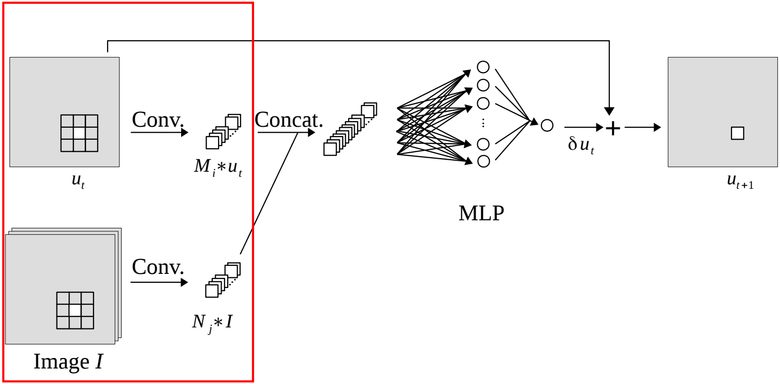
- Different PDE approaches can be devised to enhance classification maps
- Several choices must be made to select the appropriate PDE and tailor it to the considered problem
  - For example, edge-stopping function $g(\nabla I, x)$ must be chosen
- Can we let a machine learning approach discover by itself a useful iterative process?

Partial differential equations (PDEs)

- Different PDE approaches can be devised to enhance classification maps
- Several choices must be made to select the appropriate PDE and tailor it to the considered problem
  - For example, edge-stopping function $g(\nabla I, x)$ must be chosen
- Can we let a machine learning approach discover by itself a useful iterative process?
- PDEs are usually discretized in space by using finite differences
  - Derivatives as discrete convolution filters

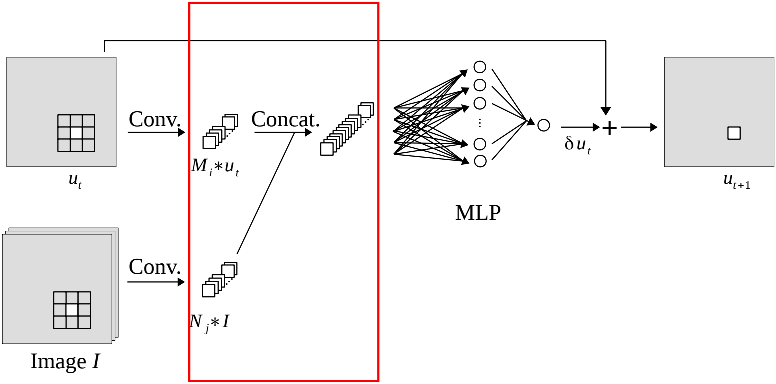
A generic enhancement process

- Differential operations $\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial^2}{\partial x \partial y}, \frac{\partial^2}{\partial x^2}, \ldots\right)$ applied on u_k and image I
- Implemented as convolutions: $M_i * u_k$, $N_j * I$
  $\{M_1, M_2, \ldots\}$, $\{N_1, N_2, \ldots\}$ conv. kernels (e.g., Sobel filters)



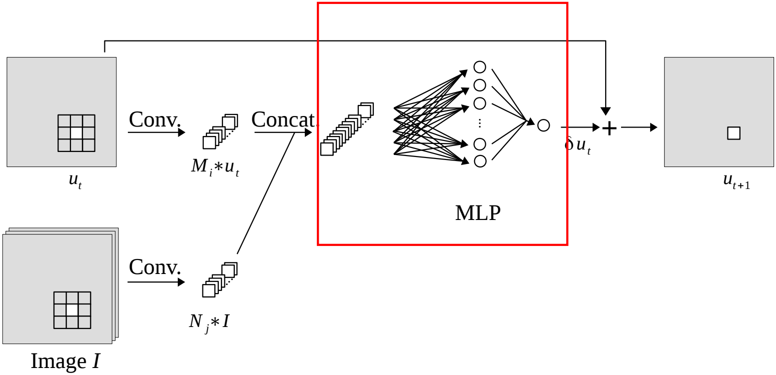
A generic enhancement process

- $\Phi(u_k, I) = \{M_i * u_k,\ \ N_j * I\ ;\ \ \forall i, j\}$, set of responses



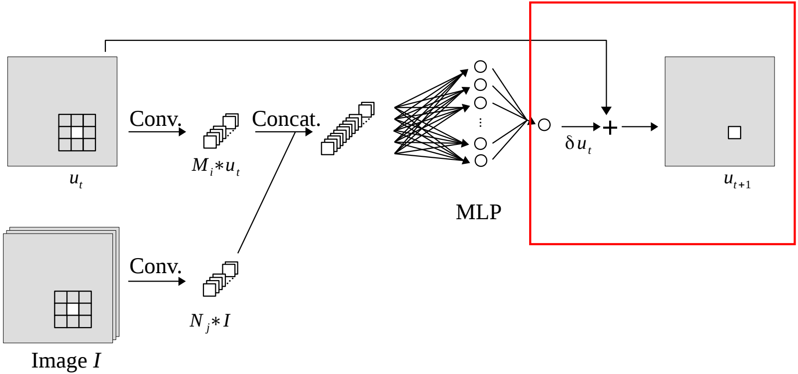
A generic enhancement process

- Overall update on u_k at x: $\delta u_k(x) = f_k\left(\Phi(u_k, I)(x) \right)$
- Class-specific f_k, implemented as multilayer perceptron
- M_i and N_j convey spatial reasoning (e.g., gradients), f_k their combination (e.g., products)



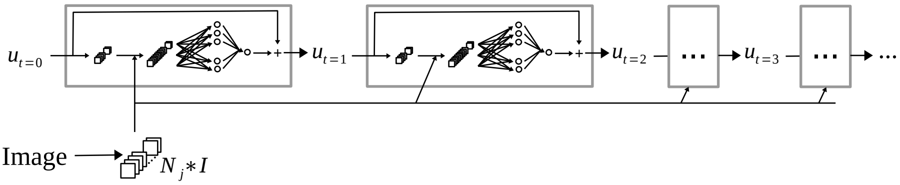
A generic enhancement process

- Discretized in time:
  $u_{k,t+1}(x) = u_{k,t}(x) + \delta u_{k,t}(x)$, overall update δ



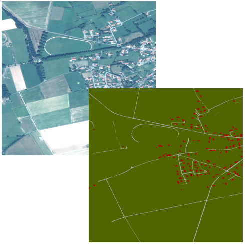
Iterative processes as recurrent neural networks (RNNs)

- "Unroll" iterations
- Enforce weight sharing along iterations
- Train by backpropagation as usual ("through time")
- Every iteration is meant to progressively refine the classification maps



Experiments

- FCN trained on Pléiades satellite images + OSM data
- Manually labeled tiles for RNN training/testing
- Unroll 5 iterations
- 32 M_i and 32 N_j
- MLP: 1 hidden layer, 32 neurons



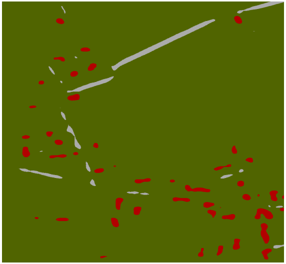
Building, Road, Background

E. Maggiori, G. Charpiat, Y. Tarabalka, P. Alliez. "Recurrent Neural Networks to Correct Satellite Image Classification Maps". TGRS 2017.

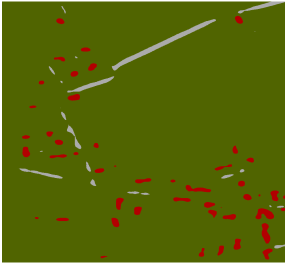
Experiments

Color input

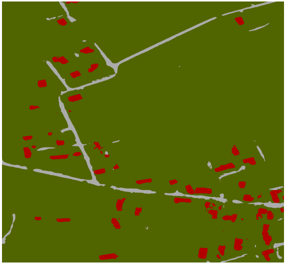
Reference



Coarse CNN

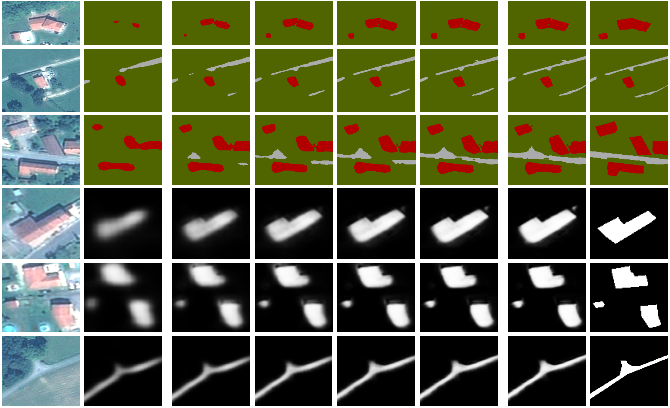


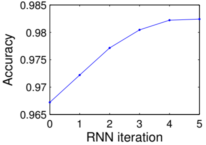
→ RNN enhancement →



RNN output

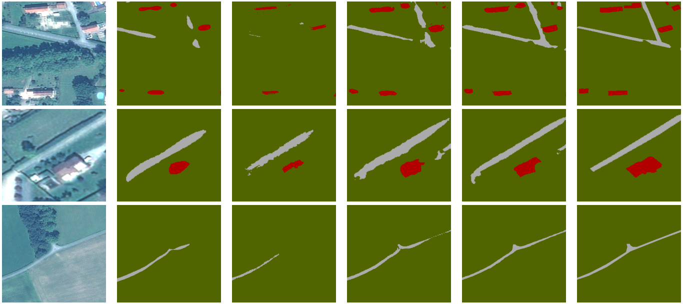
Experiments





Experiments

Comparison

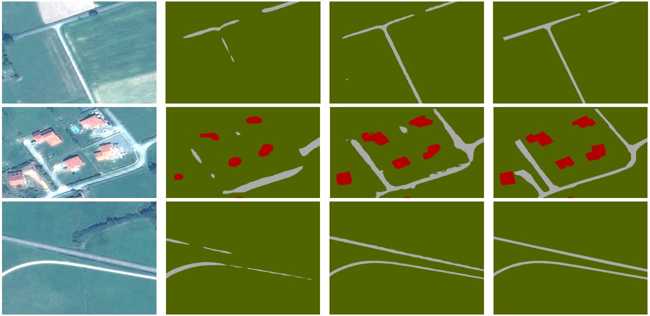


Color image | Coarse CNN | CNN+CRF | Class-agnostic CNN+RNN | CNN+RNN | Reference

Method	Overall accuracy	Mean IoU	Class-specific IoU Build.	Road	Backg.
CNN	96.72	48.32	38.92	9.34	96.69
CNN+CRF	96.96	44.15	29.05	6.62	96.78
Class-agn. CNN+RNN	97.78	65.30	59.12	39.03	97.74
CNN+RNN	**98.24**	**72.90**	**69.16**	**51.32**	**98.20**

Experiments

More examples



Color image | Coarse CNN | RNN output | Reference

Concluding remarks

- A small set of accurately labeled data can be used to enhance classification maps
- We can *learn* the specifics of an iterative enhancement process
- Removing the recurrence constraint significantly deteriorates results

Yielding high-resolution outputs

Very recent works

Four families of architectures:

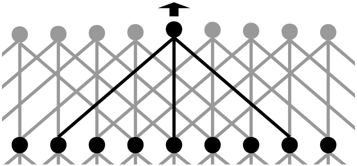
- *Dilation* (Chen et al., 2015; Dubrovina et al., 2016,...)
- *Unpooling/deconv.* (Noh et al., 2015; Volpi and Tuia, 2016,...)
- *Skip networks* (Long et al., 2015; Badrinarayanan et al., 2015,...)
- *MLP network* (Maggiori et al., 2017,...)

Ultimate goal: CNN architecture that addresses recognition/localization trade-off

Analysis of SoA: E. Maggiori, Y. Tarabalka, G. Charpiat, P. Alliez. "High-Resolution Semantic Labeling with Convolutional Neural Networks", arXiv, Nov. 2016.

Dilation networks

- Based on the shift-and-stitch approach:
    - Conduct predictions at different offsets to produce low-resolution outputs
    - Interleave these outputs to compose the final high-resolution result
- Such an interleaving can be implemented as convolutions on non-contiguous locations

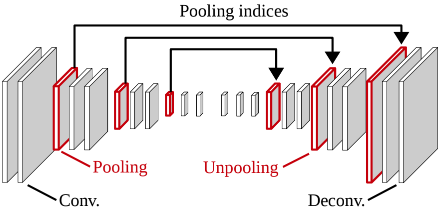


⇒ Larger context without introducing more parameters

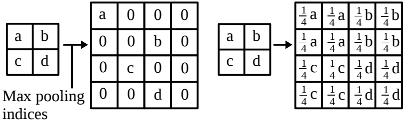
- Not robust to spatial deformation
  (e.g., detect road located *exactly* 5px away)

Unpooling/deconvolution networks

- The CNN is "mirrored" to learn the deconvolution:



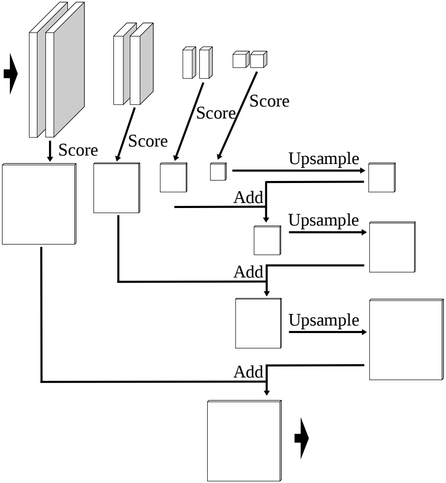
- Max (left) and average (right) unpooling



- The depth of deconv. networks is significantly larger ($\sim$ twice FCN)

Skip networks

1. Extract intermediate features
2. Classify
3. Upsample/add (pairwise)



- Addresses trade-off
- Inflexible/arbitrary at combining resolutions

MLP network

Premise

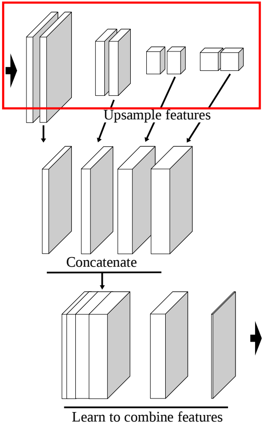
- CNNs do not need to "see" everywhere at the same resolution
- E.g., to classify central pixel:

Full resolution context

Full resolution only near center

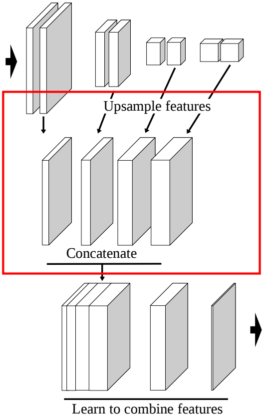
⇒ Combine resolutions to address trade-off, in a flexible way

MLP network



Base FCN

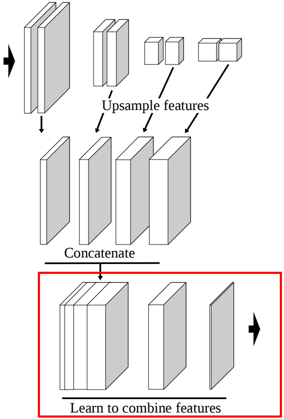
MLP network



- Extract intermediate features
- Upsample to the highest res.
- Concatenate

⇒ Pool of features
(e.g., edge detectors, object detectors)

MLP network



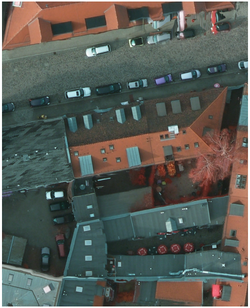
- Multi-layer perceptron (MLP) learns how to combine those features
  $\Rightarrow$ Output classif. map
- Pixel by pixel (series of 1×1 convolutional layers)
  $\Rightarrow$ 128 hidden neurons, nonlinear activation
- Addresses trade-off in a flexible way

Experiments

Datasets

ISPRS 2D semantic labeling contest:

Vaihingen (9 cm)



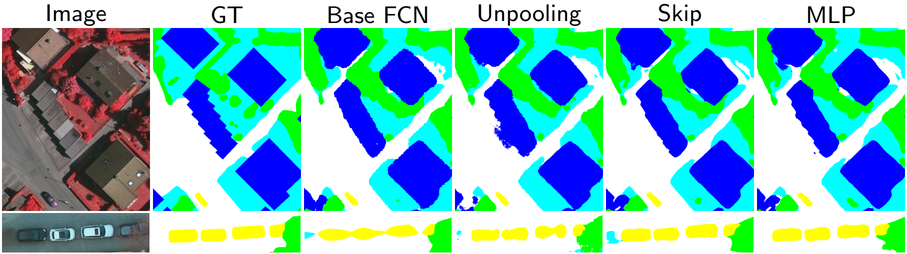
Potsdam (5 cm)

- Color infra-red + Elevation model

Results: Base FCN vs derived architectures

Vaihingen	Imp. surf.	Building	Low veg.	Tree	Car	Mean F1	Acc.
Base FCN	91.46	94.88	79.19	87.89	72.25	85.14	88.61
Unpooling	91.17	95.16	79.06	87.78	69.49	84.54	88.55
Skip	91.66	95.02	79.13	88.11	77.96	86.38	88.80
MLP	**91.69**	**95.24**	**79.44**	**88.12**	**78.42**	**86.58**	**88.92**

Potsdam	Imp. surf.	Building	Low veg.	Tree	Car	Clutter	Mean F1	Acc.
Base FCN	88.33	93.97	84.11	80.30	86.13	75.35	84.70	86.20
Unpooling	87.00	92.86	82.93	78.04	84.85	72.47	83.03	84.67
Skip	89.27	94.21	84.73	**81.23**	93.47	75.18	86.35	86.89
MLP	**89.31**	**94.37**	**84.83**	81.10	**93.56**	**76.54**	**86.62**	**87.02**



Classes: Impervious surface (white), Building (blue), Low veget. (cyan), Tree (green), Car (yellow), Clutter (red).

Results: Comparison with other methods

Vaihingen	Imp. surf.	Build.	Low veg.	Tree	Car	F1	Acc.
CNN+RF	88.58	94.23	76.58	86.29	67.58	82.65	86.52
CNN+RF+CRF	89.10	94.30	77.36	86.25	71.91	83.78	86.89
Deconvolution						83.58	87.83
Dilation	90.19	94.49	77.69	87.24	76.77	85.28	87.70
Dilation + CRF	90.41	94.73	78.25	87.25	75.57	85.24	87.90
MLP	**91.69**	**95.24**	**79.44**	**88.12**	**78.42**	**86.58**	**88.92**

Submission of the MLP-network results to ISPRS server

- Overall accuracy: 89.5%
- Second place (out of 29) at the time of submission
- Significantly simpler and faster than other methods

Concluding remarks

- Modern CNN architertures address well recognition/localization trade-off
- Good generalisation potential
- How to implement?
    - You can use ready frameworks
- New architectures become popular
    - Example: U-net

Concluding remarks

Key to CNNs' success

Imposing *sensible* restrictions to neuronal connections reduces optimization search space w.l.o.g:

- Better minima $\rightarrow$ better accuracy
- Computational efficiency

$\Rightarrow$ Win-win

A recurrent pattern: simpler is better

- FCNs $\rightarrow$ More accurate and 10x faster
- RNNs $\rightarrow$ Removing recurrence significantly degrades results
- MLP net $\rightarrow$ More accurate than more complicated models

Concluding remarks

The "no free lunch" principle in machine learning (Wolper, 1996)

There is no such thing as a universally better classifier. A classifier is better under certain assumptions.

- CNNs exploit the properties of images particularly well
- Shifting efforts from feature engineering to network engineering
- Good *payoff* of the efforts,
  e.g., learning better features than handmade ones,
  convolutions $\rightarrow$ GPUs, borrowing pretrained network

Gall, J., Yao, A., Razavi, N., Gool, L. V., and Lempitsky, V. (2011). Hough forests for object detection, tracking, and action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2188–2202.

Hänsch, R. (2014). *Generic object categorization in PolSAR images - and beyond*. PhD thesis, TU Berlin.

R. Hänsch, O. H. (2015a). Feature-independent classification of hyperspectral images by projection-based random forests. In *WHISPERS 2015*.

R. Hänsch, O. H. (2015b). Performance assessment and interpretation of random forests by three-dimensional visualizations. In *IVAPP 2015*.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901.

I