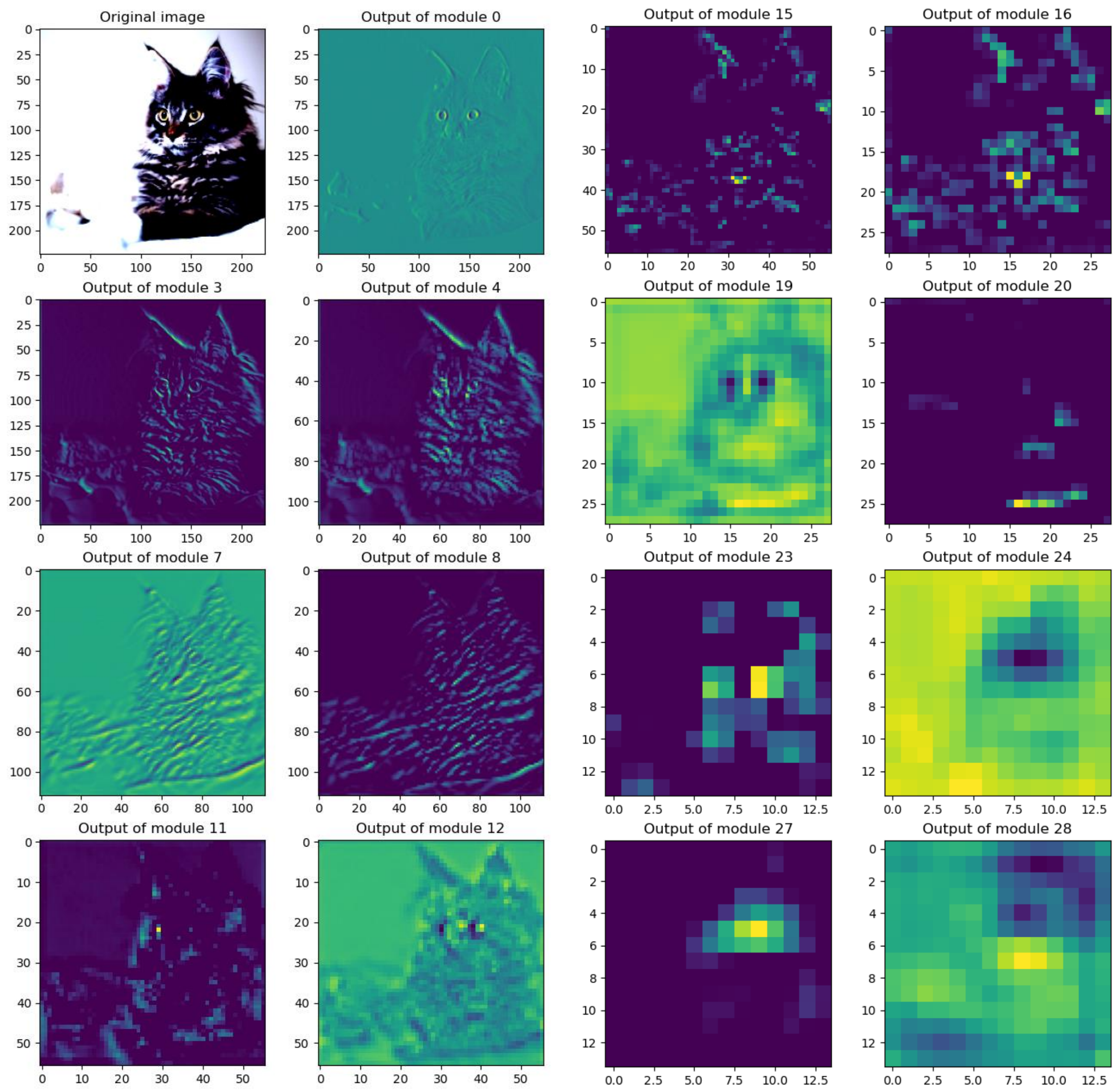
Convolutional Neural Networks

Alberto TONDA, Ph.D. (Senior permanent researcher, DR)

UMR 518 MIA-PS, INRAE, AgroParisTech, Université Paris-Saclay
UAR 3611, Institut des Systèmes Complexes de Paris Île-de-France

Outline

- Convolution
- Max pooling
- Architecture
- Applications
- Success stories



A landmark in Deep Learning history

- Between end of 1990s up to 2010, <u>nobody cared</u> about NNs
  - Considered suboptimal compared to ensembles of trees
  - Slow to train, not as effective on tabular data
  - Still, NN people did not give up
- CNNs emerged in ~2012 in a computer vision competition
  - ImageNet, 14M human-annotated images (classification)
  - For several years, champion performance stagnated
  - AlexNet, a CNN, got ~75% accuracy (runner-up ~65%)
  - Later in 2015 Microsoft used a 100-layer CNN to get ~96%

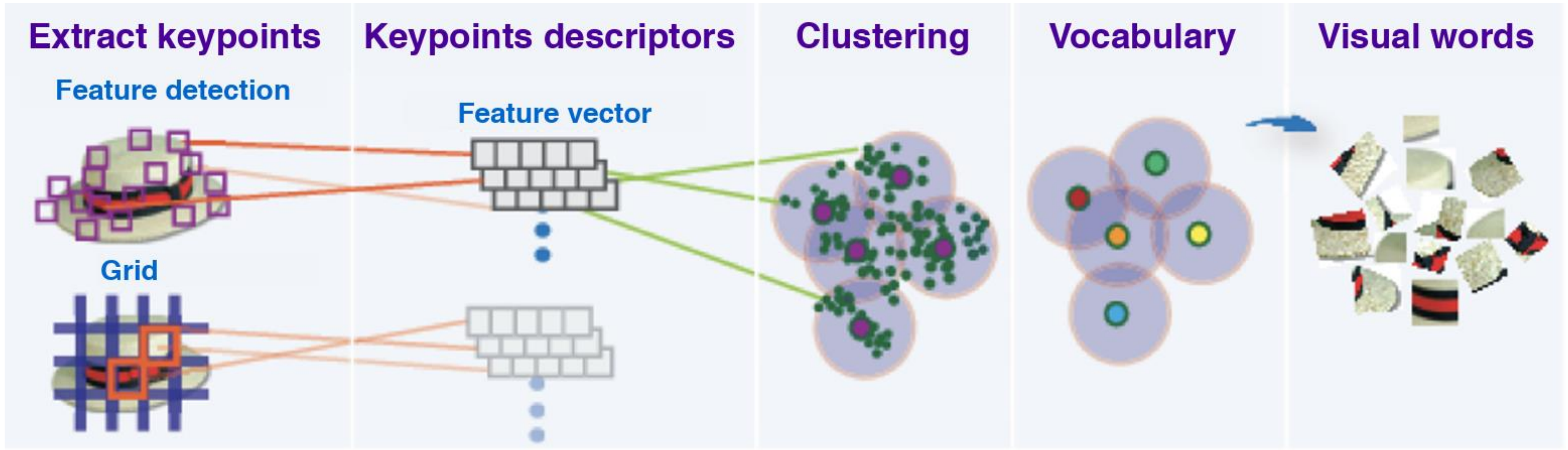
Convolution

- Originally from the domain of **computer vision**

Object Tracking
Pose Estimation
Object Detection
Action Recognition
Autonomous Navigation
3D Reconstruction
Crowd Understanding
Urban Scene Understanding
Indoor Scene Understanding
Multi-agent Collaboration
Human Training
Aerial Surveying
Image
Image Label
Depth/Multi-View
User Input
Video
Physics
Segmentation/Bounding Box
Camera Localization

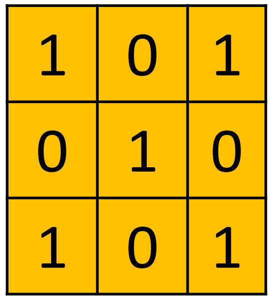
Convolution

- Before Deep Learning, considerations from computer vision
  - Considering an image as a grid of pixels is missing **semantics**
  - If one pixel is considered one feature, it's **not informative**
  - Too tied to its absolute position, **relative position** matters more



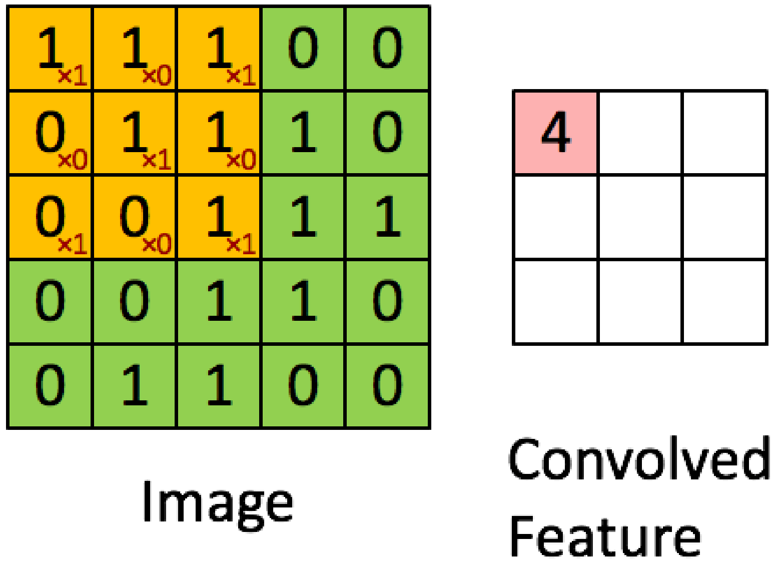
Convolution

- Convolutional filters
  - Small set of weights that are applied to the whole input image
  - “Sliding window”, same operation on different groups of pixels
  - Output: a tensor *more or less* of the same size



1	0	1
0	1	0
1	0	1

Convolutional filter



$1_{\times1}$	$1_{\times0}$	$1_{\times1}$	0	0
$0_{\times0}$	$1_{\times1}$	$1_{\times0}$	1	0
$0_{\times1}$	$0_{\times0}$	$1_{\times1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

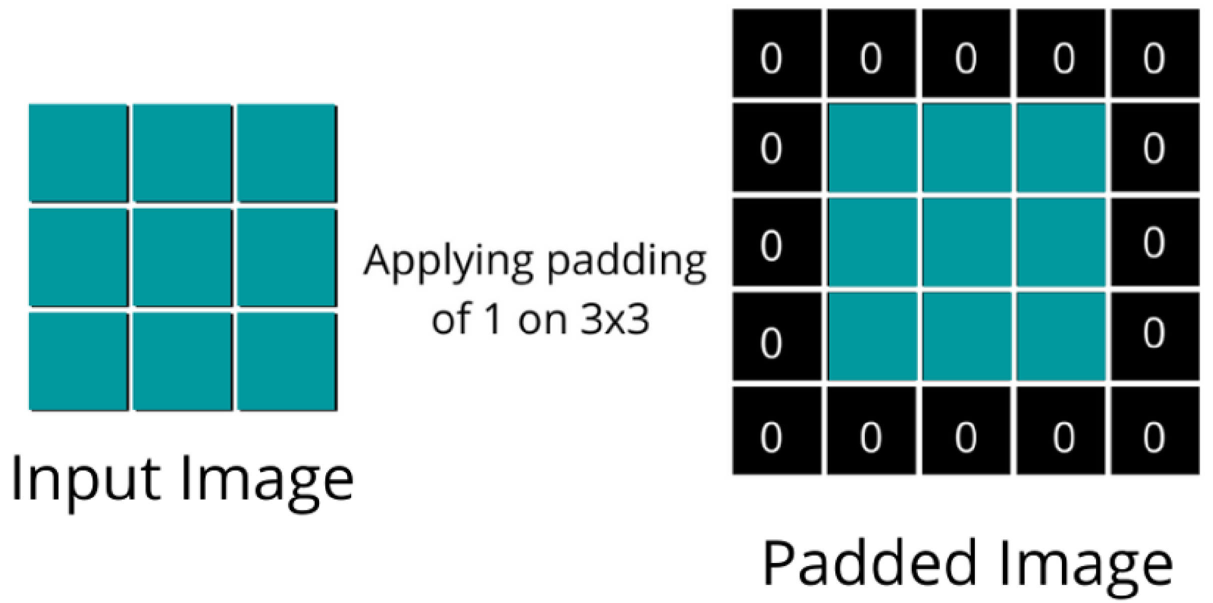
4		

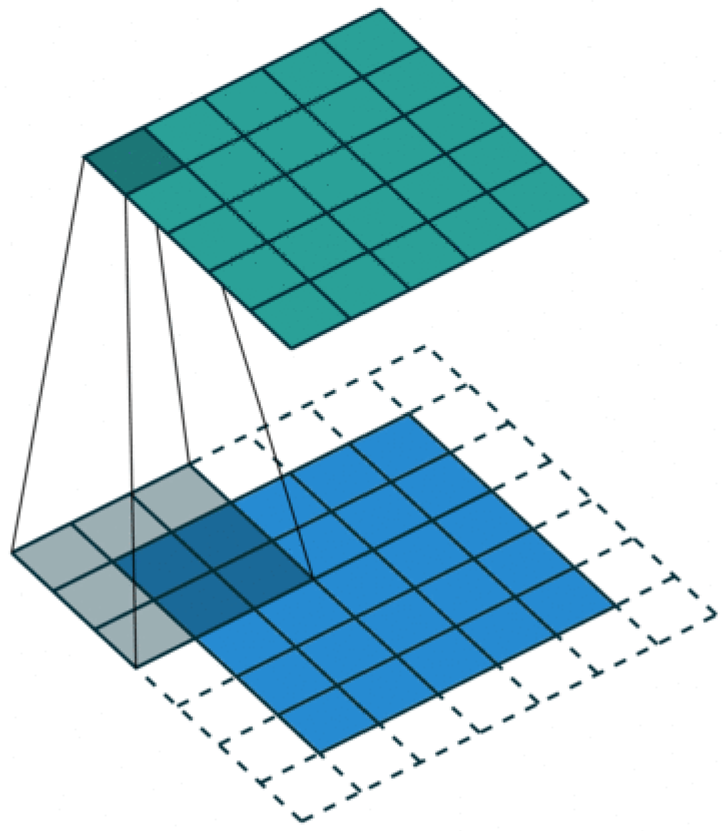
Convolved Feature

Convolution

- Convolutional filters
  - Learn patterns freely by adjusting weights
  - The patterns can appear **everywhere** in the image
  - Convolutional operation can be applied in **parallel**
- Still, it is not completely automatic: hyperparameters!
  - **Size** of the convolutional window (sometimes called **kernel**)
  - **Stride**, how often to apply the window
  - **Padding**, apply window even in the corners

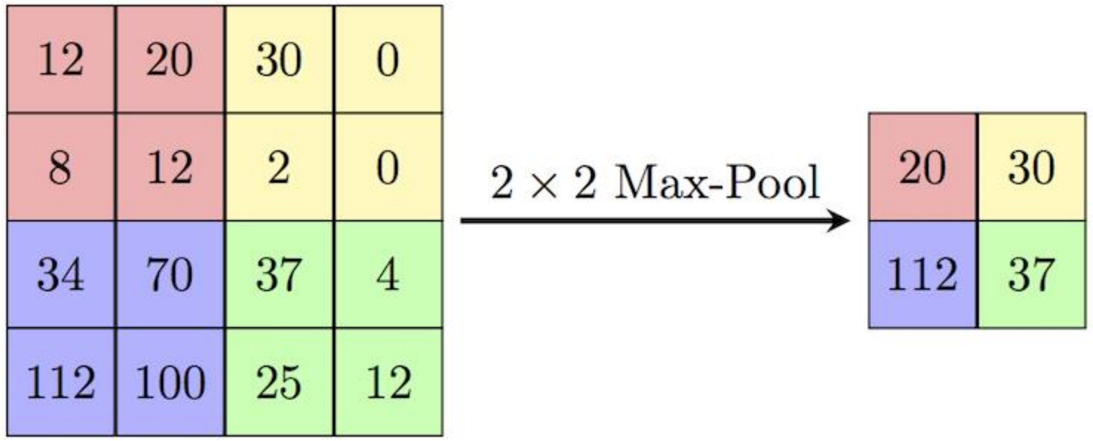
Padding

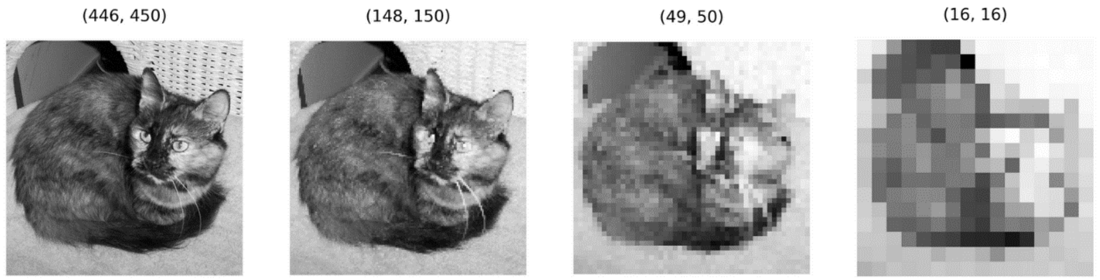

Input Image
Applying padding of 1 on 3x3
0 0 0 0 0
0 0
0 0
0 0
0 0 0 0 0
Padded Image



Pooling (Downsampling)

- After sending result of the convolution through **activation**
  - Pick the highest resulting value in a small grid applied to image
  - Idea: only pick the most important features created by filters



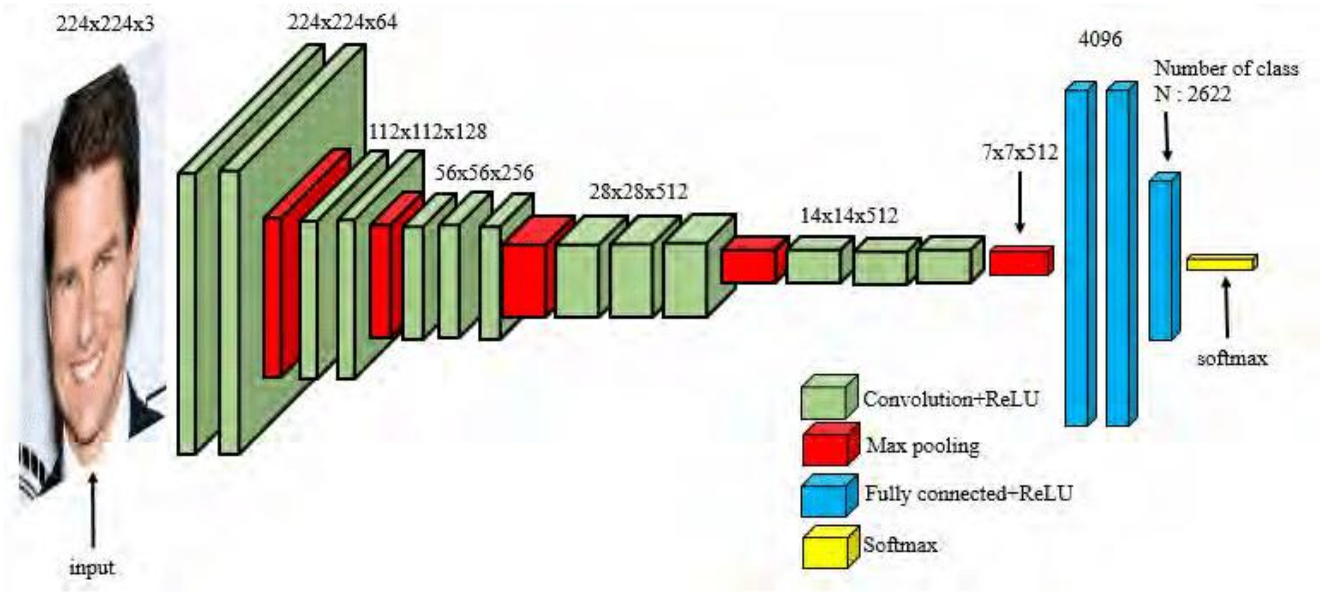


Pooling (Downsampling)

- Different possible choices of pooling
  - Get the highest value in window (MaxPool)
  - Average over values in window (AvgPool)
  - …

Architecture

- The idea is to create a *funnel* from high dimension to low
  - With sequences of convolution/activation/pooling
  - And final linear modules for classification or other
- Real architectures can be quite complex



CNNs for image classification

- The last part of the network is linear modules + activations
- The output is a tensor with one shape (n_classes)

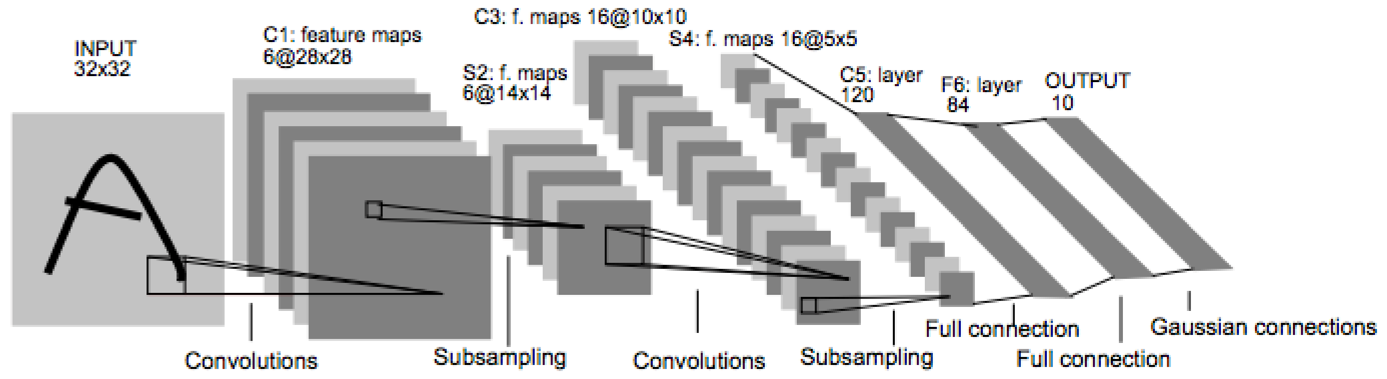
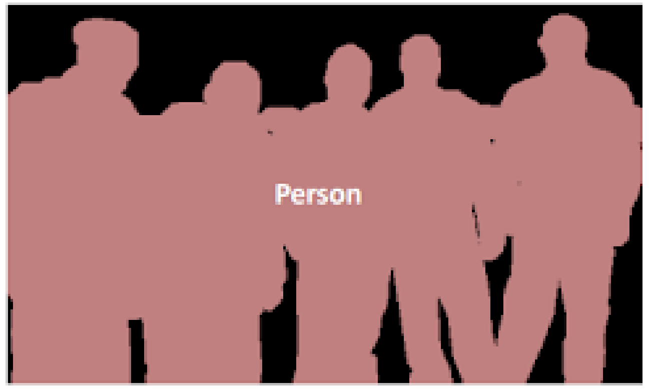


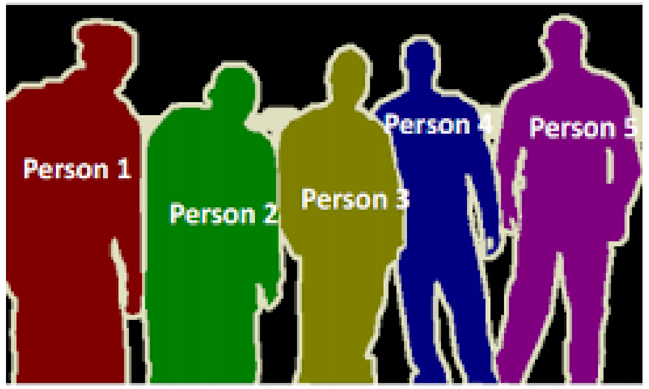
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

CNNs for image segmentation

- Different types of segmentation
  - **Semantic segmentation**, associate pixels to classes
  - **Instance segmentation**, associate pixel to object instances



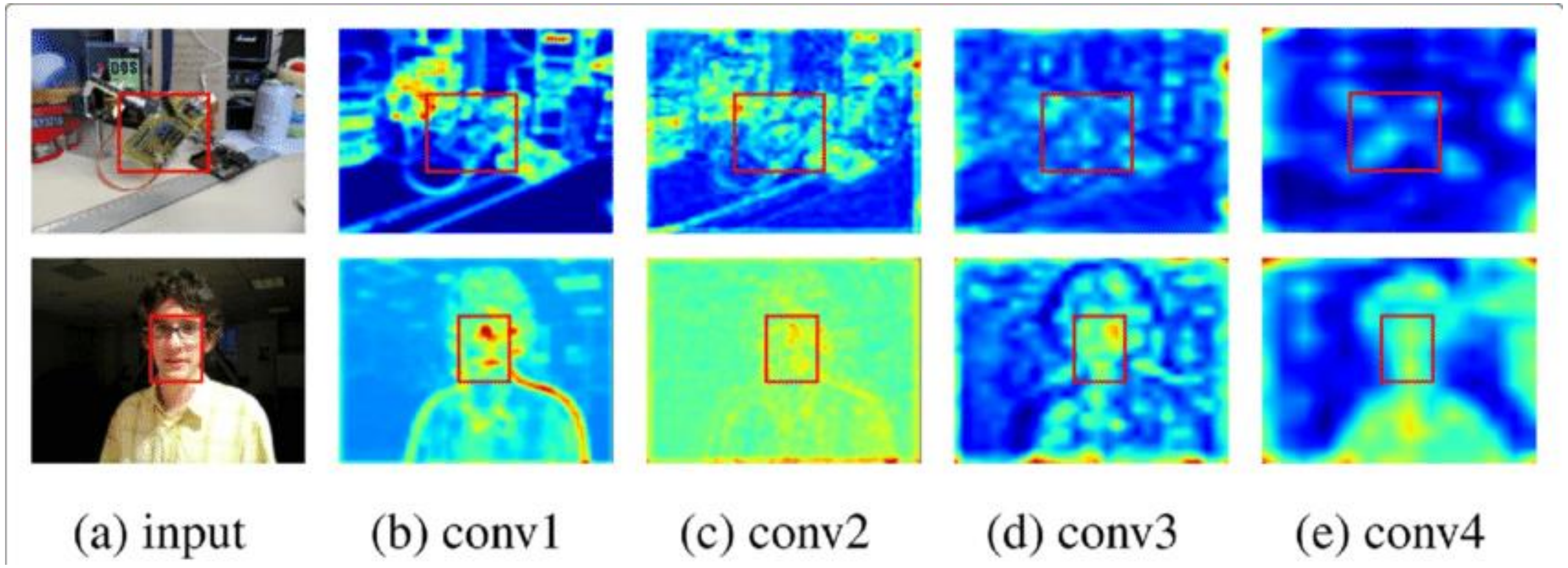
Semantic Segmentation



Instance Segmentation

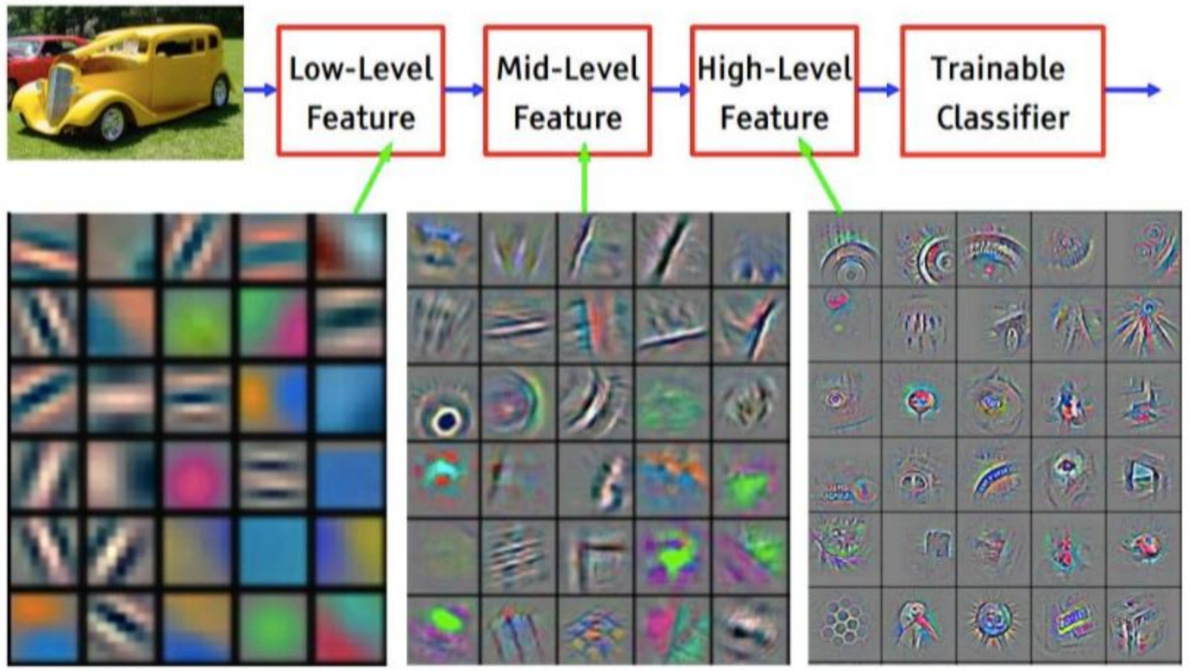
Looking inside a CNN

- Tensors in output to every module
  - Input has original meaning as an image
  - Visualize the intermediate tensors (convolutional and pooling)



Features constructed by the CNN

- Visualize features created by CNN
  - Create images that maximize module output (tensor values)
  - More details in class on visualization (later)

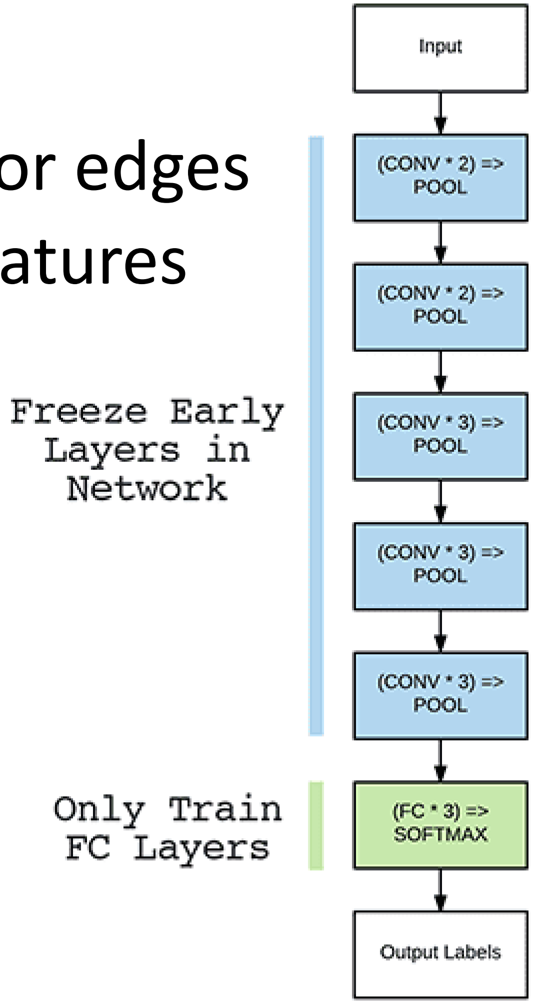


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Features constructed by the CNN

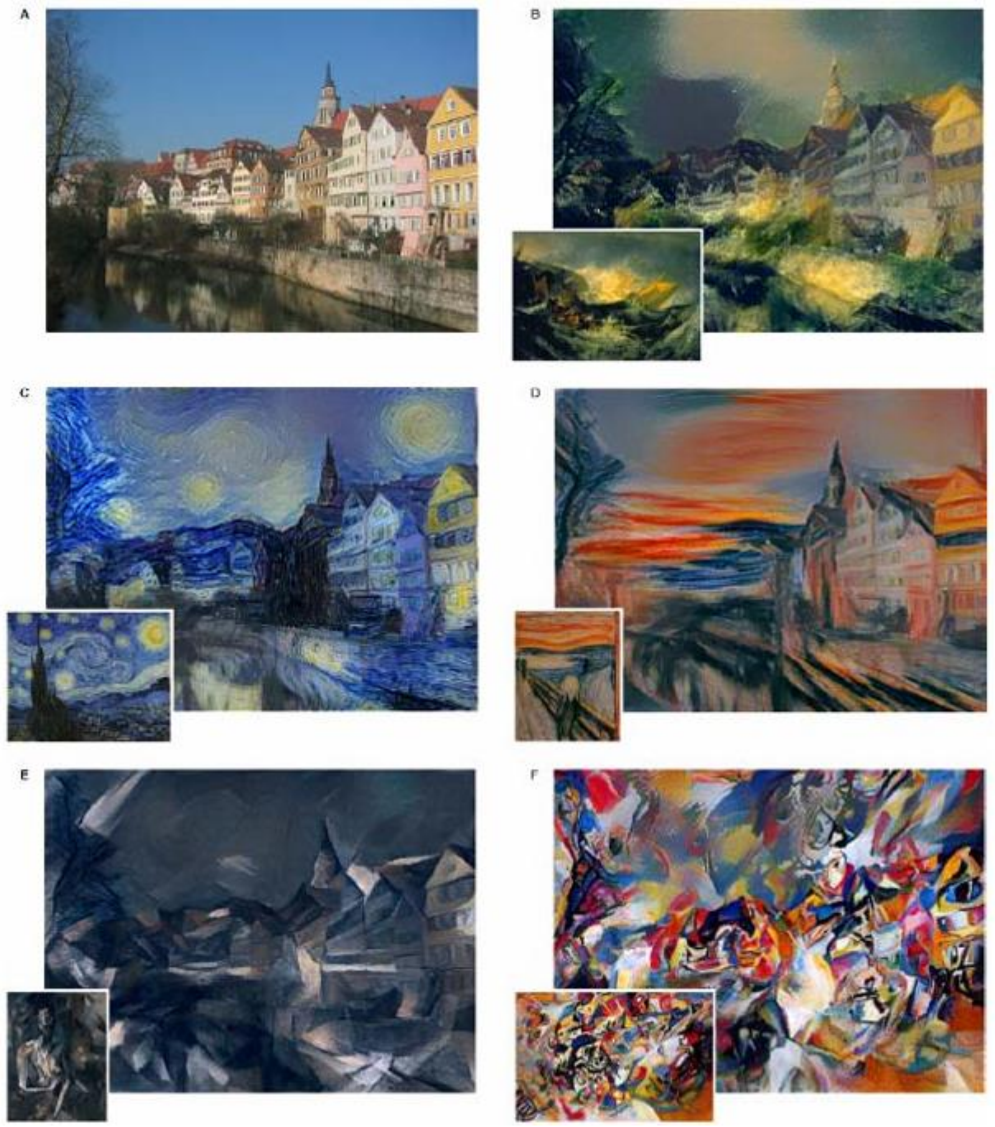
- If this is true (very likely true for large models)
  - Knowledge encoded in first layers can be **re-used**
  - No need to re-learn from scratch features for lines or edges
  - What should be re-learned are mid-to-high-level features
  - Or maybe only the **final part**! (linear + activations)

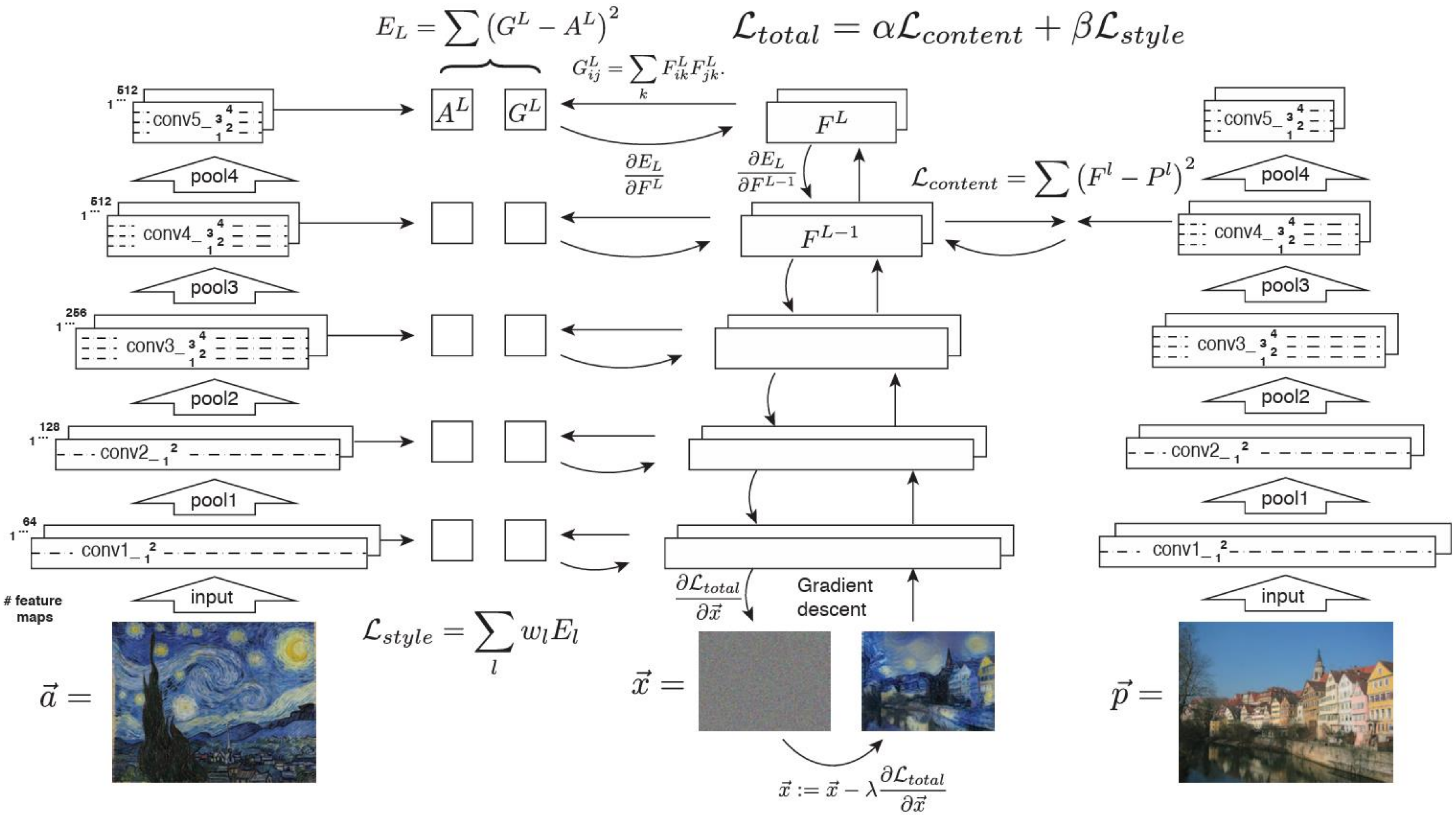
- First seeds of *transfer learning*
  - Transfer learning? Fine-tuning? LoRA?
  - Very similar concepts



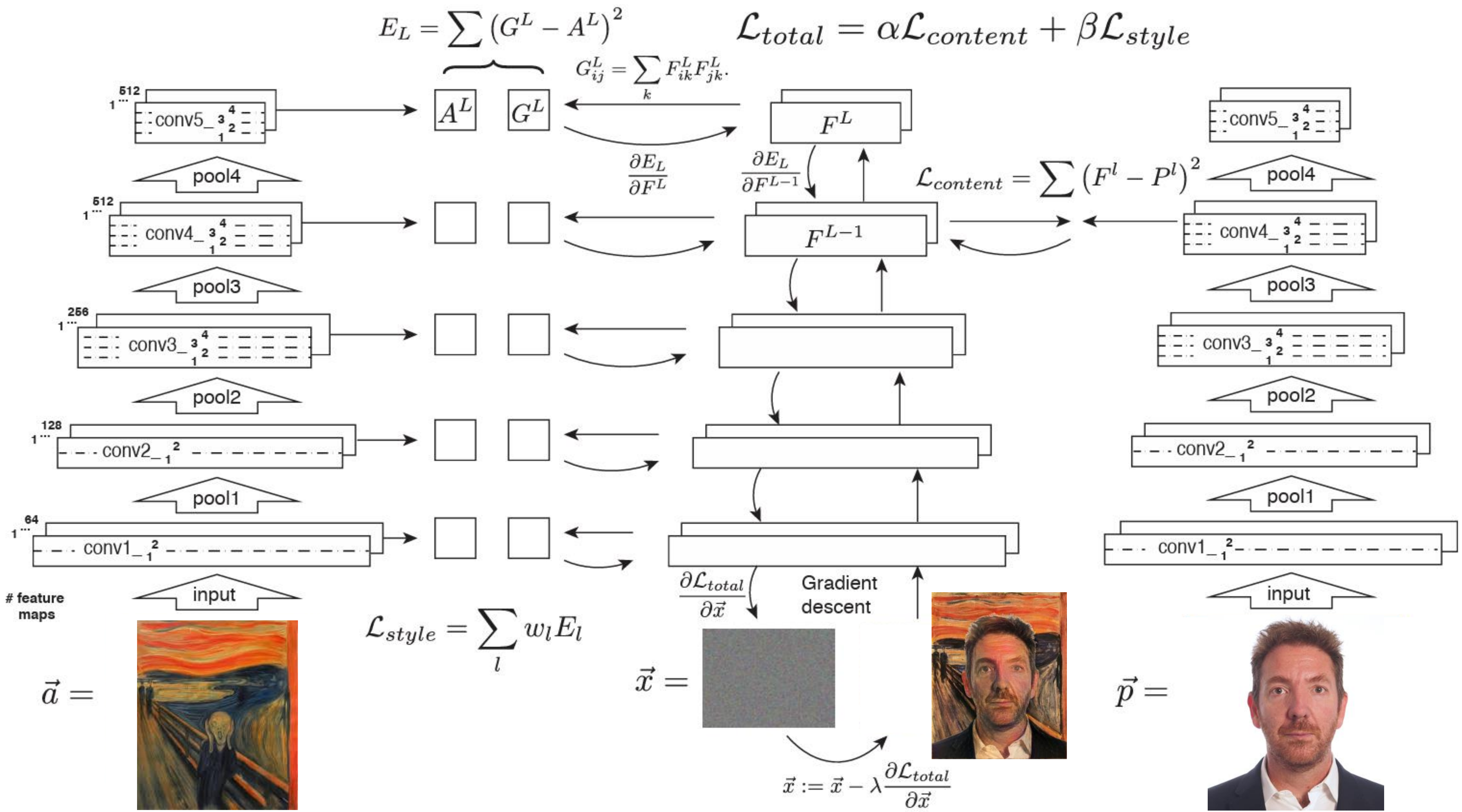
Style transfer

- Using the fact that weights in modules capture features
- "Style" is a feature
- Apply style to content

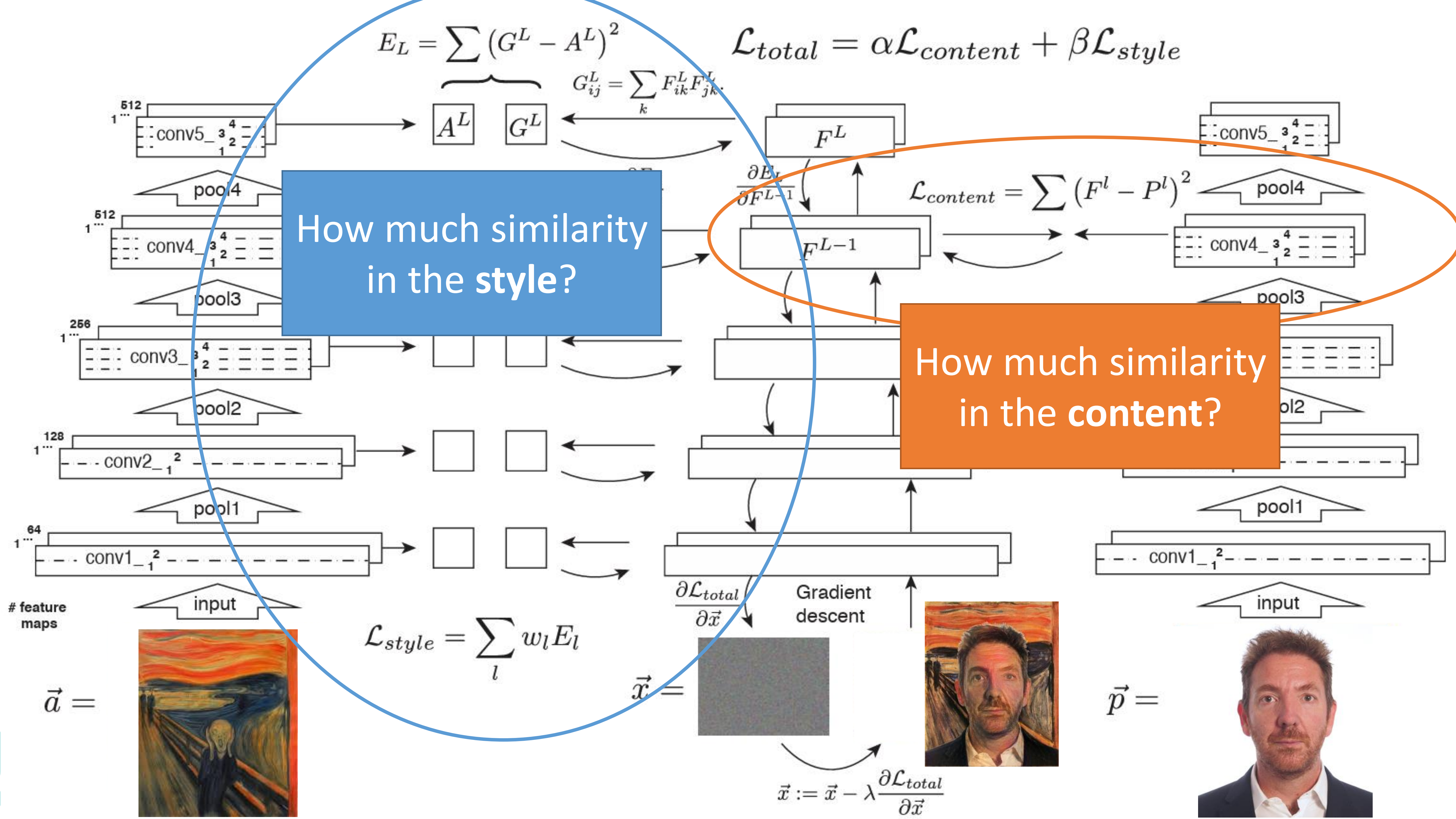




$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}.$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
512
256
128
64
conv5_1 2 3 4
conv4_1 2 3 4
conv3_1 2 3 4
conv2_1 2
conv1_1 2
pool4
pool3
pool2
pool1
input
# feature maps
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$



$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}.$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
512
conv5_1 2 3 4
pool4
512
conv4_1 2 3 4
pool3
256
conv3_1 2 3 4
pool2
128
conv2_1 2
pool1
64
conv1_1 2
input
# feature maps
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
A^L
G^L
F^L
$\frac{\partial E_L}{\partial F^{L-1}}$
F^{L-1}
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
conv5_1 2 3 4
conv4_1 2 3 4
conv3_1 2 3 4
conv2_1 2
conv1_1 2
pool4
pool3
pool2
pool1
input
512
256
128
64
# feature maps
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

How much similarity in the **style**?

How much similarity in the **content**?

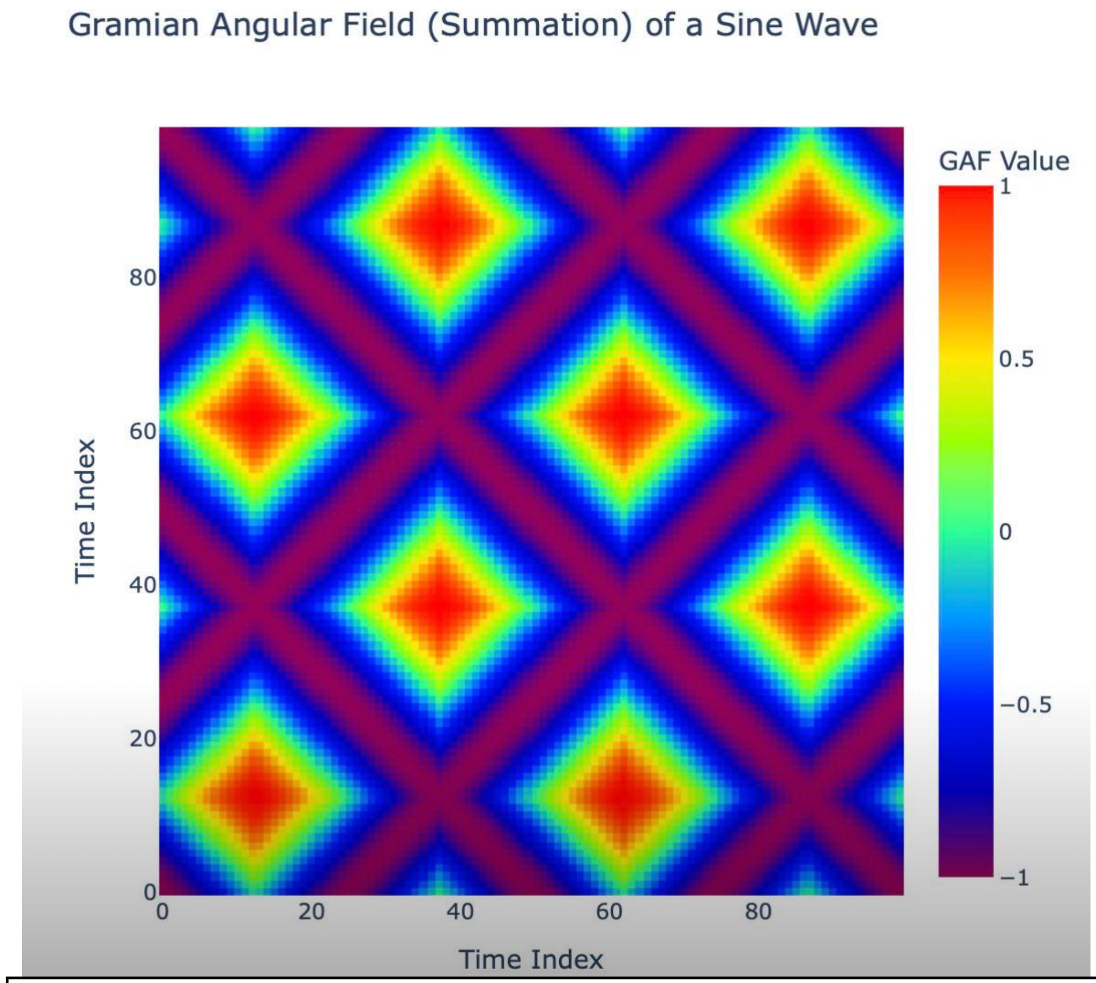
CNNs for non-images?

- Can you imagine to use the feature construction ability of CNNs to applications that are *not* related to vision?

CNNs for time series analysis

- CNNs perform **automatic feature construction**
- There are alternatives, using expert-designed features
- For example, python library **tsfresh**
  - Transform the dynamic problem into a series of static features
  - This is *tabular data*! Apply classic ML algorithms
  - (IMHO) Decent for classification, not so much for forecasting

CNNs for time series analysis



https://medium.com/analytics-vidhya/encoding-time-series-as-images-b043becbdbf3

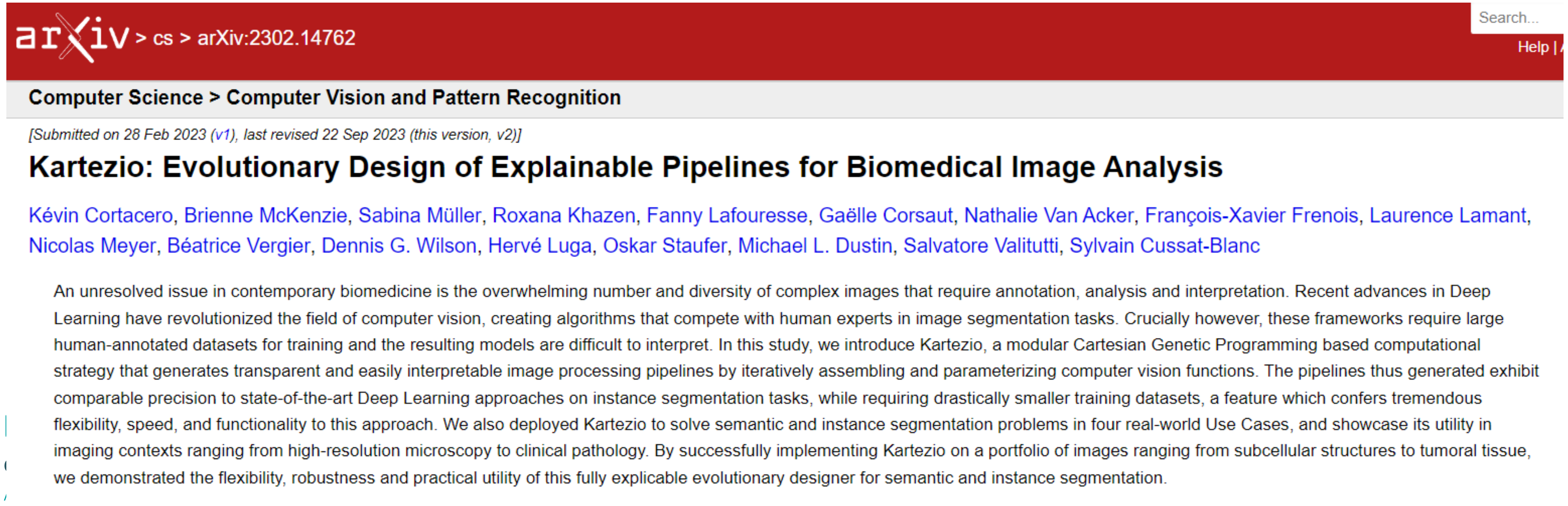
Relational data

- In general, every data where adjacency has **meaning**
  - DNA/RNA, for example
  - Signals coming from spectrometry
  - Voxels from 3D scans
  - Adjacent nodes in a graph (Convolutional Graph Neural Networks)

- The only big difference is the **shape** of convolutional filters

Alternatives to CNNs for images?

- Currently, CNNs are the state of the art
  - Recent results claim that **Visual Transformers** are better
- More human-readable approaches?
  - White-box ML approach by Cussat-Leblanc et al., **Kartezio**



Computer Science > Computer Vision and Pattern Recognition

[Submitted on 28 Feb 2023 (v1), last revised 22 Sep 2023 (this version, v2)]

Kartezio: Evolutionary Design of Explainable Pipelines for Biomedical Image Analysis

Kévin Cortacero, Brienne McKenzie, Sabina Müller, Roxana Khazen, Fanny Lafouresse, Gaëlle Corsaut, Nathalie Van Acker, François-Xavier Frenois, Laurence Lamant, Nicolas Meyer, Béatrice Vergier, Dennis G. Wilson, Hervé Luga, Oskar Staufer, Michael L. Dustin, Salvatore Valitutti, Sylvain Cussat-Blanc

An unresolved issue in contemporary biomedicine is the overwhelming number and diversity of complex images that require annotation, analysis and interpretation. Recent advances in Deep Learning have revolutionized the field of computer vision, creating algorithms that compete with human experts in image segmentation tasks. Crucially however, these frameworks require large human-annotated datasets for training and the resulting models are difficult to interpret. In this study, we introduce Kartezio, a modular Cartesian Genetic Programming based computational strategy that generates transparent and easily interpretable image processing pipelines by iteratively assembling and parameterizing computer vision functions. The pipelines thus generated exhibit comparable precision to state-of-the-art Deep Learning approaches on instance segmentation tasks, while requiring drastically smaller training datasets, a feature which confers tremendous flexibility, speed, and functionality to this approach. We also deployed Kartezio to solve semantic and instance segmentation problems in four real-world Use Cases, and showcase its utility in imaging contexts ranging from high-resolution microscopy to clinical pathology. By successfully implementing Kartezio on a portfolio of images ranging from subcellular structures to tumoral tissue, we demonstrated the flexibility, robustness and practical utility of this fully explicable evolutionary designer for semantic and instance segmentation.

Questions?

Bibliography

Cortacero, K., ... & Cussat-Blanc, S. (2023). Kartezio: Evolutionary design of explainable pipelines for biomedical image analysis. arXiv preprint arXiv:2302.14762. https://arxiv.org/abs/2302.14762

Cortacero, K., McKenzie, B., Müller, S. et al. Evolutionary design of explainable algorithms for biomedical image segmentation. Nat Commun 14, 7112 (2023). https://doi.org/10.1038/s41467-023-42664-x

Images and videos: unless otherwise stated, I stole them from the Internet. I hope they are not copyrighted, or that their use falls under the Fair Use clause, and if not, I am sorry. Please don't sue me.