CNN: Convolutional Neural Nets

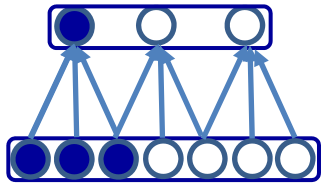
Introduction
Classification
Object detection
Image segmentation

CNNs

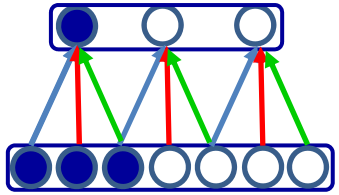
- CNNs were developped in the late 80ies for image and speech applications
- Deep CNNs were successfully used for image applications (classification and segmentation) in the 2010s – starting with the ImageNet competition, and for speech recognition.
  - Their use has been extended to handle several situations
  - They come now in many variants
  - They can often be used as alternatives to Recurrent NNs

CNNs
principle

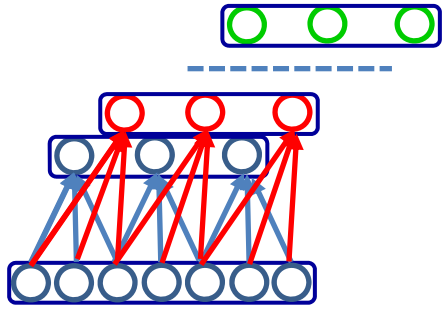
- Exploit local characteristics of the data via local connections
  - e.g. images (2 D), speech signal (1 D)



- Local connections are constrained to have shared weight vectors
  - This is equivalent to convolve a unique weight vector with the input signal
    - Think of a local edge detector for images
    - The 3 hidden cells here share the same weight vector
      - (blue, red, green weight values)

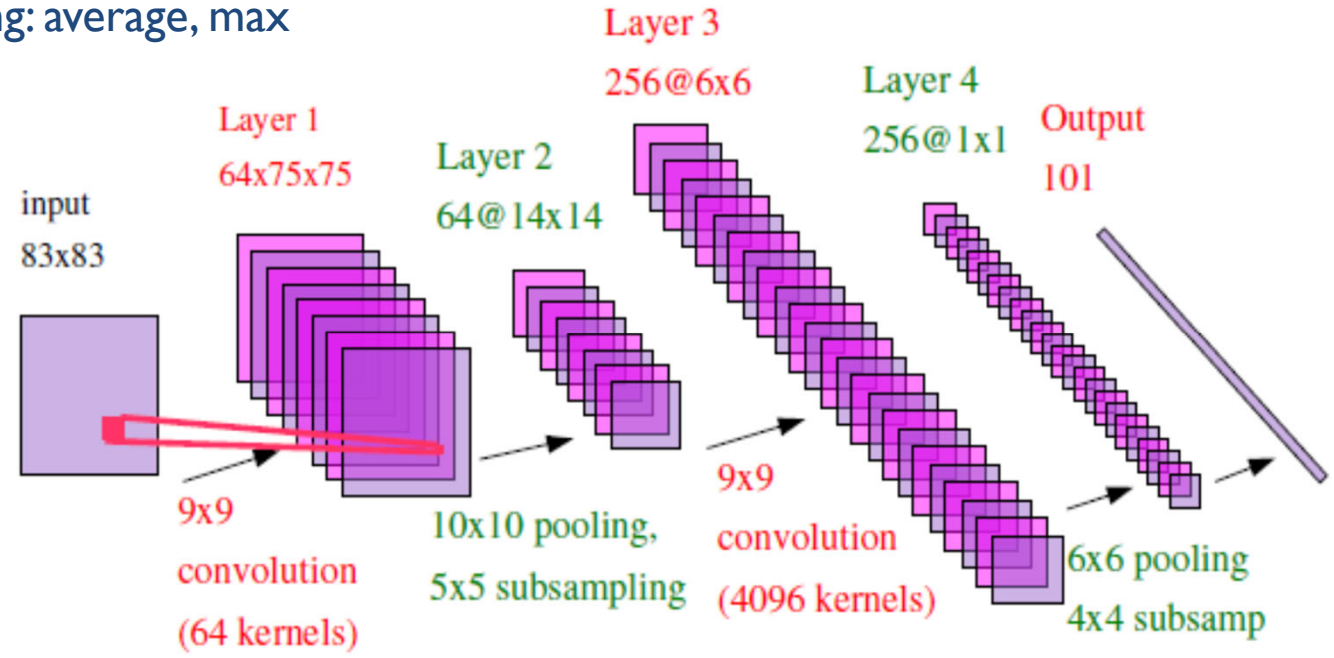


- Several convolution filters can be learned simultaneously
  - This corresponds to applying a set of local filters on the input signal
    - e.g edge detectors at different angles for an image
    - here colors indicate similar **weight vectors,** not weight values as above



CNNs
example

- ConvNet architecture (Y. LeCun since 1988)
  - Deployed at Bell Labs in 1989-90 for Zip code recognition
  - Character recognition
  - Convolution: non linear embedding in high dimension
  - Pooling: average, max



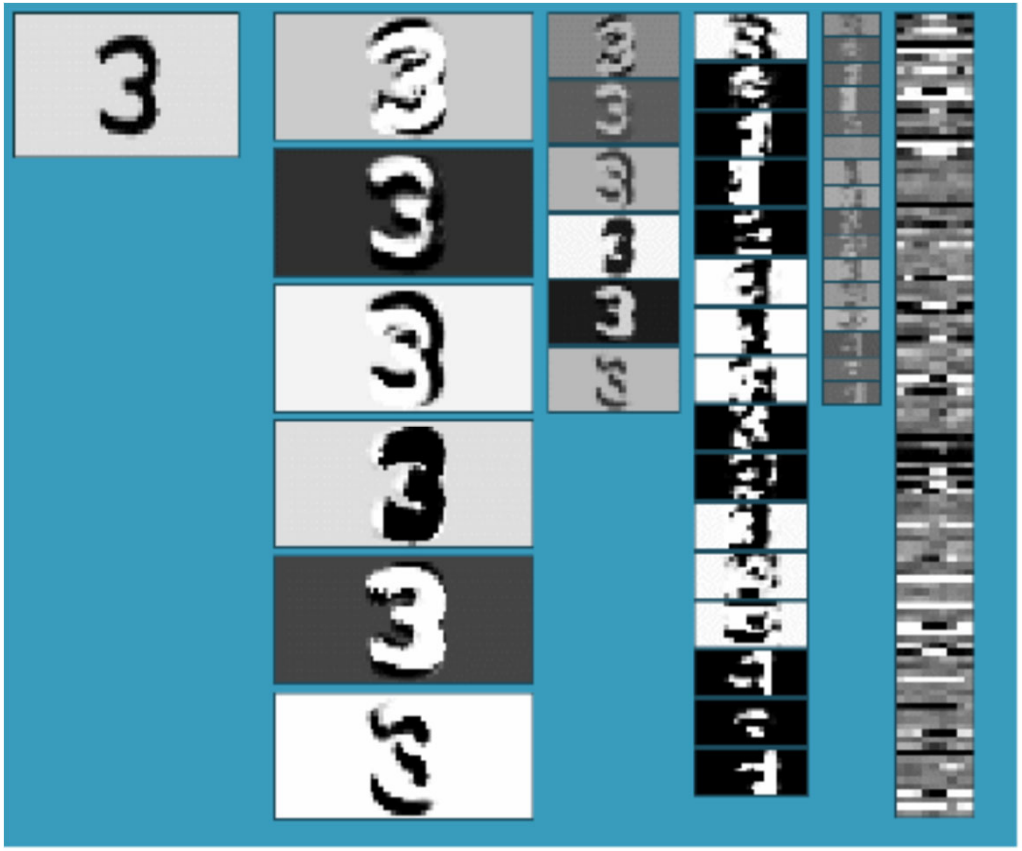
parameters 64x9x9=5184, 256x9x9=20736, 256x101 = 60916

CNNs

- In Convnet
  - The first hidden layer consists in 64 different convolution kernels over the initial input, resulting in 64 different mapping of the input
  - The second hidden layer is a sub-sampling layer with a pooling tranformation applied to each matrix representation of the first hidden layer
  - etc
  - Last layer is a classification layer, fully connected
- More generally
  - CNNs alternate convolution, and pooling layers, and a fully connected layer at the top.

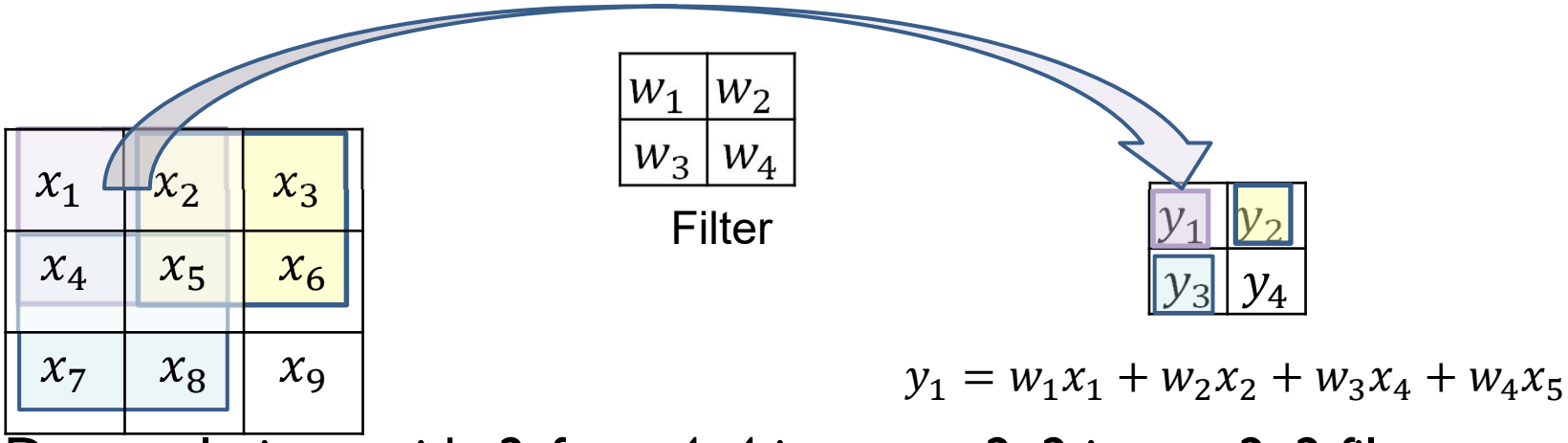
CNNs visualization

- Hand writing recognition (Y. LeCun Bell labs 1989)



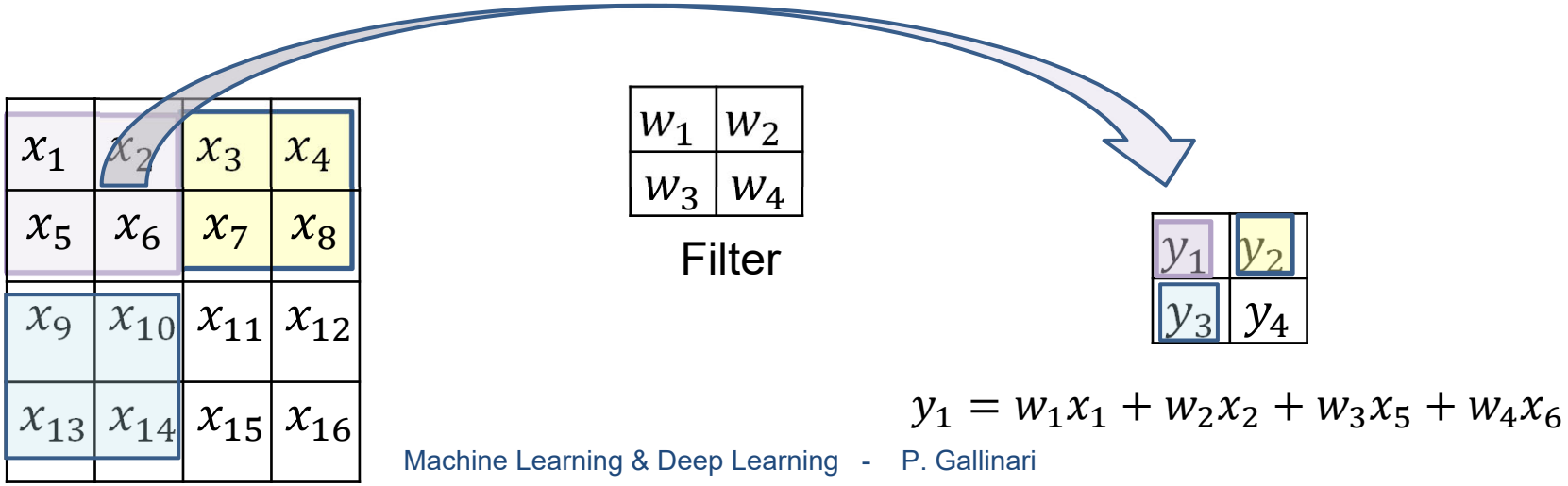
CNNs
Convolution: filter size and stride

- 2D convolution, stride 1, from 3x3 image to 2x2 image, 2x2 filter



$$y_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5$$

- 2 D convolution, stride 2, from 4x4 image to 2x2 image, 2x2 filter



$$y_1 = w_1x_1 + w_2x_2 + w_3x_5 + w_4x_6$$

CNNs
Padding

- Padding amounts at filling the border of the image, usually with 0
  - The width of the padding border depends on the filter characteristics

0	0	0	0	0	0
0	x_1	x_2	x_3	x_4	0
0	x_5	x_6	x_7	x_8	0
0	x_9	x_{10}	x_{11}	x_{12}	0
0	x_{13}	x_{14}	x_{15}	x_{16}	0
0	0	0	0	0	0

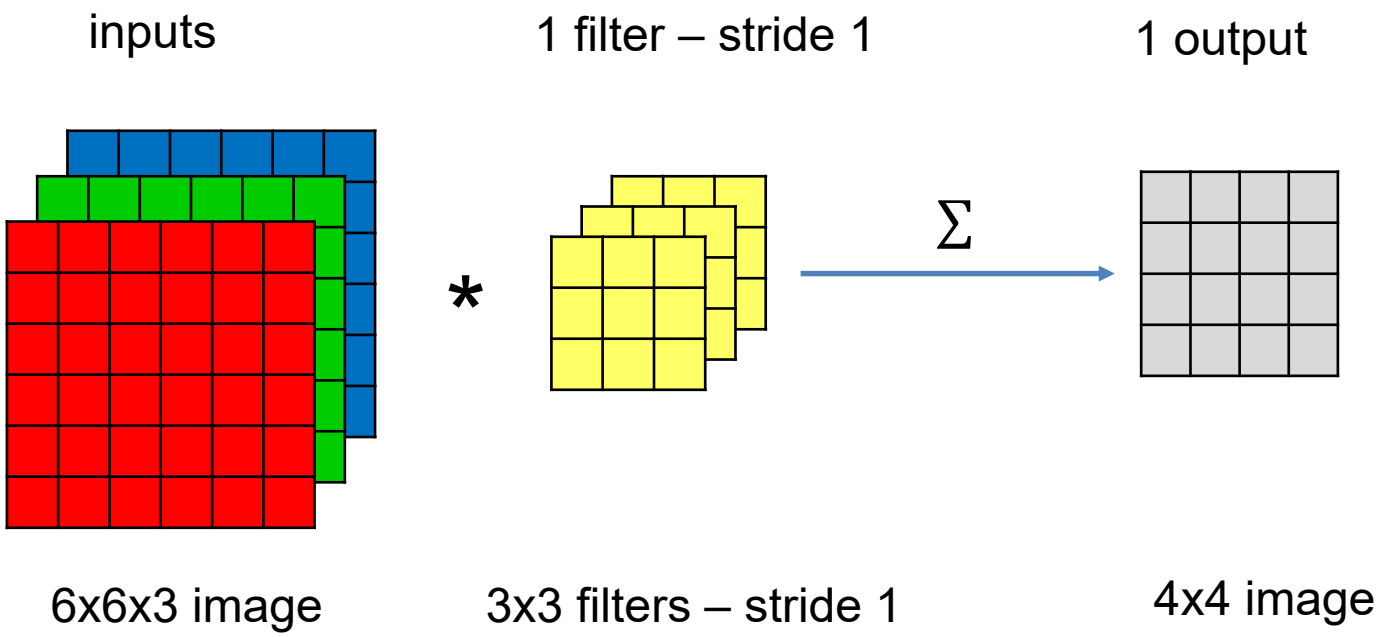
CNNs
Convolutions arithmetics

- Input image nxn, filter $f\,\mathrm{x}f$, padding p, stride s
- Output image is $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ x $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$
- Floor function [.]
  - in some cases a convolution will produce the same output size for multiple input sizes. If $i + 2p - k$ is a multiple of s, then any input size $j = i + a, a \in \{0, \ldots, s-1\}$ will produce the same output size. This applies only for $s > 1$.

Note: more in (Dumoulin 2016), a guide to convolution arithmetic for Deep Learning

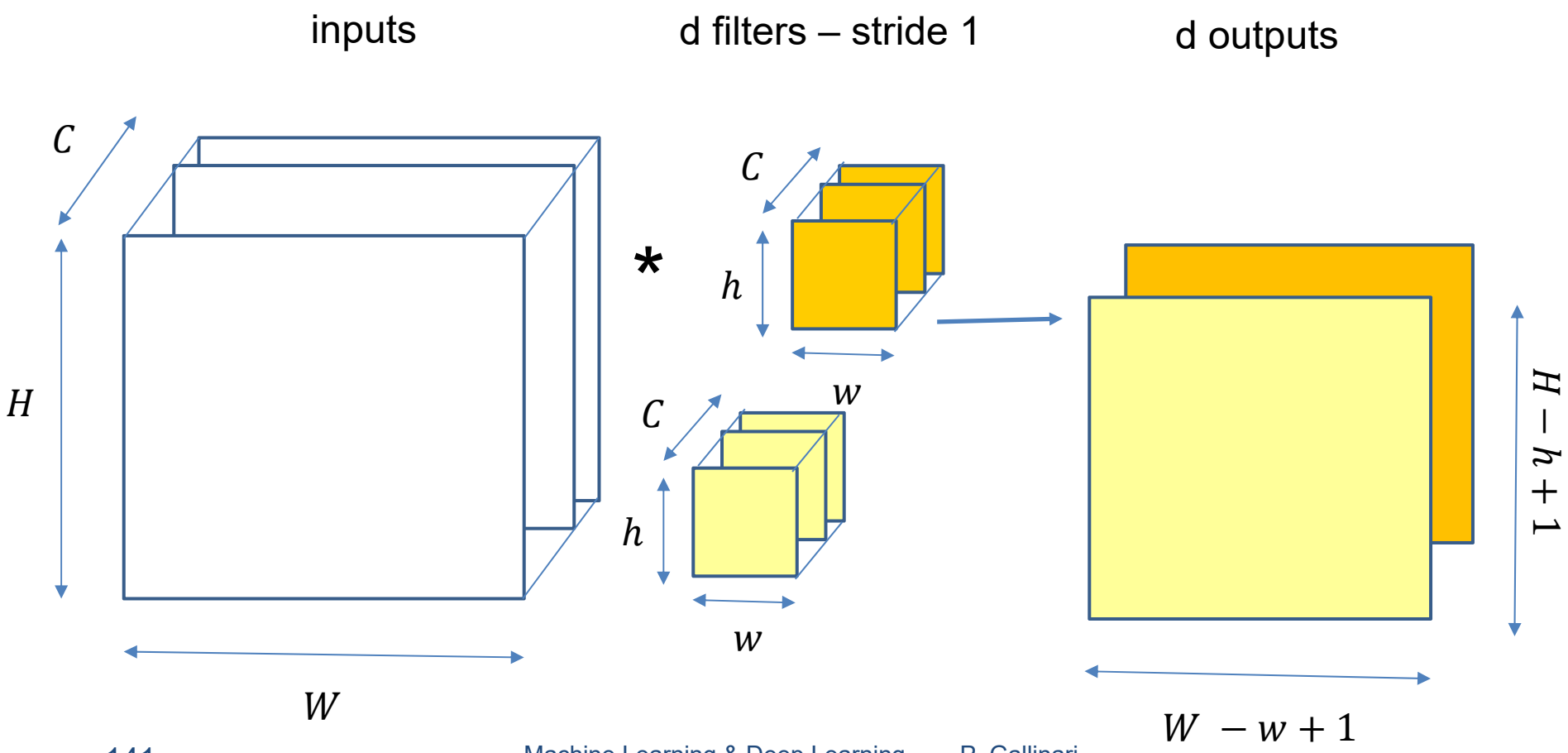
CNNs
on multiple channels, e.g. RGB images

- Convolution generalizes to multiple channels. For images, the input is usually a 3 D tensor, and the output is a 2 D tensor: the filter is not swipped across channels usually, but only across rows and columns of the corresponding channel.



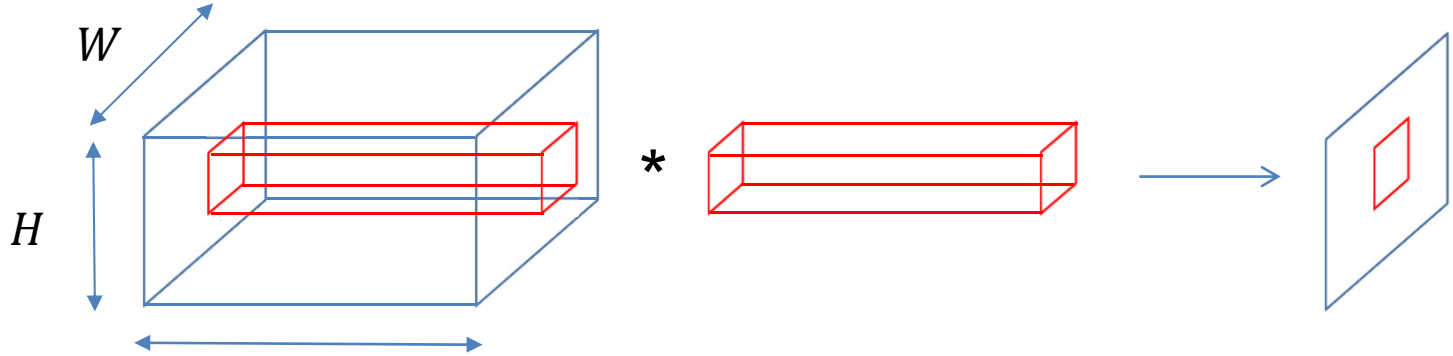
CNNs
on multiple channels

- This generalizes to any number of input channels, and filters
  - Below C input channels and 2 outputs



CNNs
1x1 convolutions on multiple channels

- 1x1 convolutions, perform a pixel wise weighted sum on several channels
  - They are used to reduce the size of a volume
    - e.g. transforming a HxWxC volume to a HxWxC' volume with $C' < C$, by using C', 1x1 convolutions

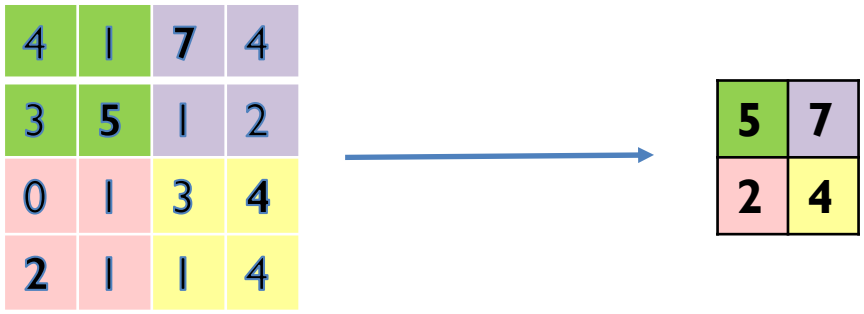


HxWxC 1x1xC HxWx1

$C' = 1$ convolution in this example

CNNs
Pooling

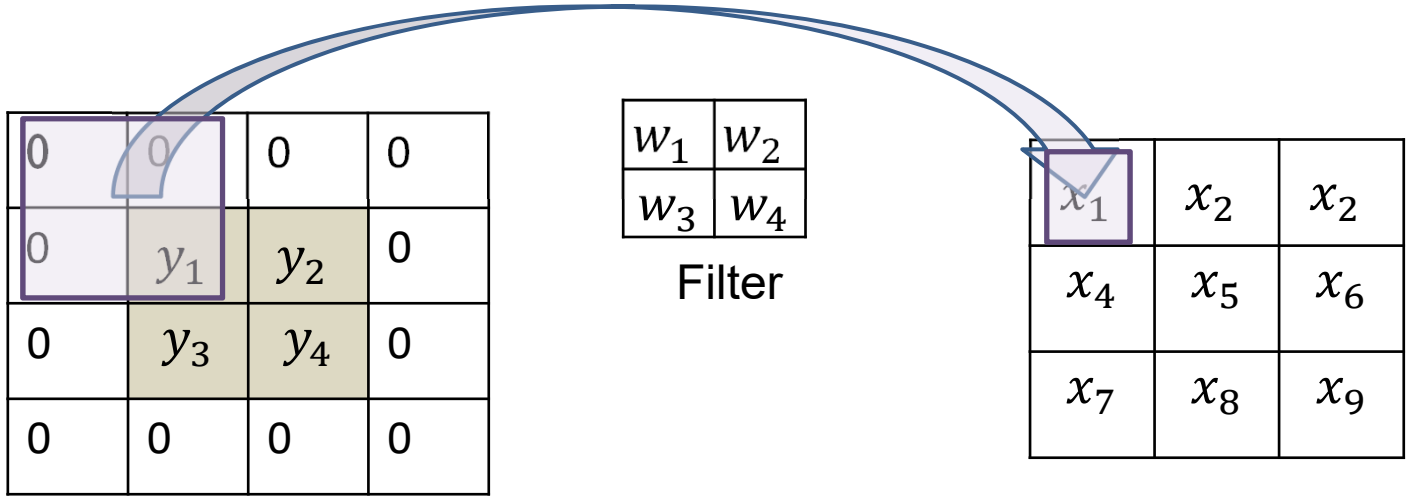
- Pooling
  - Used to aggregate information from a given layer
  - Usually Mean or Max operators are used for pooling
  - Example: Max pooling, stride 2



- Pooling provides some form of invariance to input deformations
- Pooling arithmetics

CNNs
Transposed convolution

- This is the reverse operation – to a convolution
  - Increases the input image size
    - Used for auto-encoders, object recognition, segmentation
  - Example: from 2x2 image to 3x3 image, 2x2 filter, Stride 1 with Padding



Note: more in (Dumoulin 2016), a guide to convolution arithmetic for Deep Learning

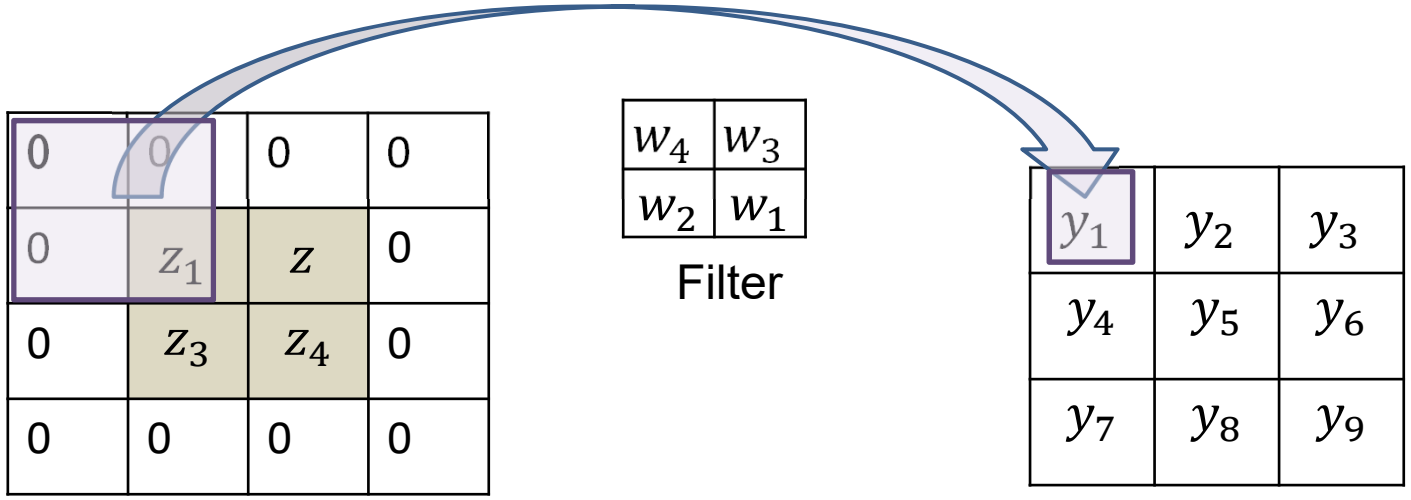
Transposed convolutions

- Convolution
  - $x * w = z$, with $x \in R^9, z \in R^4$
  - $x = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix}, \mathrm{w} = \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix}, \mathrm{z} = \begin{pmatrix} z_1 & z_2 \\ z_3 & z_4 \end{pmatrix}$
  - Convolution in matrix form
    - Lets flatten the vectors, the CNN convolution can be written in matrix form as:
    - $\mathrm{W}x = z$
    - $x = \begin{pmatrix} x_1 \\ \vdots \\ x_9 \end{pmatrix}, \mathrm{W} = \begin{pmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{pmatrix}, \mathrm{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix}$

- Transposed convolution
  - Transposed convolution in matrix form $y = W^T z, \quad z \in R^4 \; and \; y \in R^9$
  - $\mathrm{W}^{\mathrm{T}} = \begin{pmatrix} w_1 & 0 & 0 & 0 \\ w_2 & w_1 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ w_3 & 0 & w_1 & 0 \\ w_4 & w_3 & w_2 & w_1 \\ 0 & w_4 & 0 & w_2 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & w_4 & w_3 \\ 0 & 0 & 0 & w_4 \end{pmatrix}$

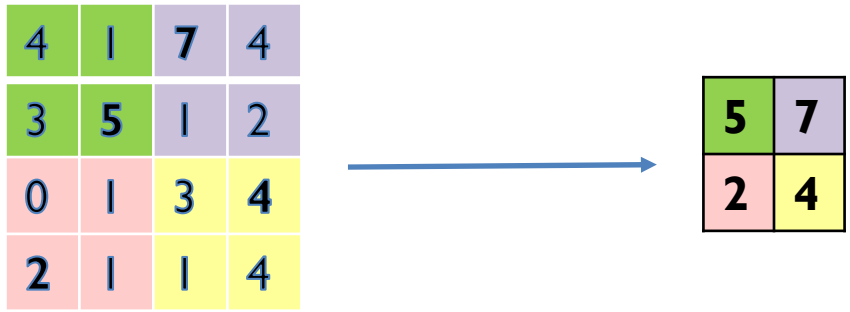
Transposed convoution

- Transposed convolution in convolutional form $y = z * w$

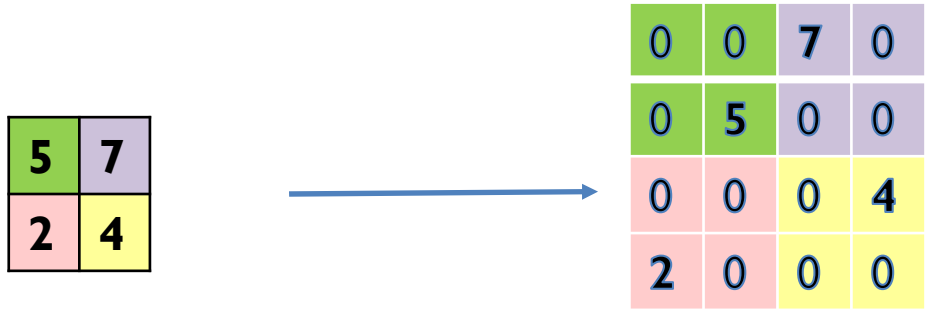


CNNs
Unpooling

- Reverse pooling operation
- Different solutions, e.g. unpooling a max pooling operation

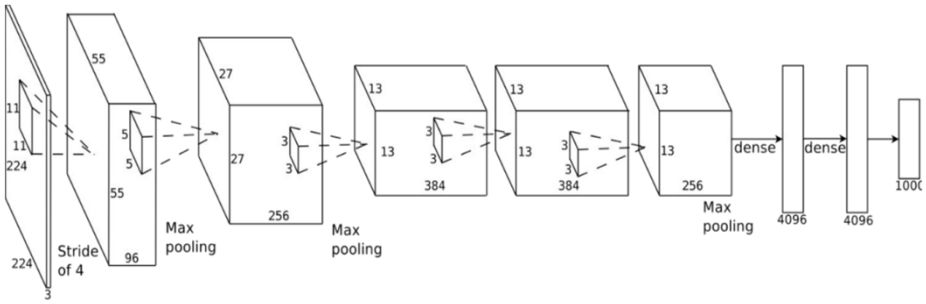


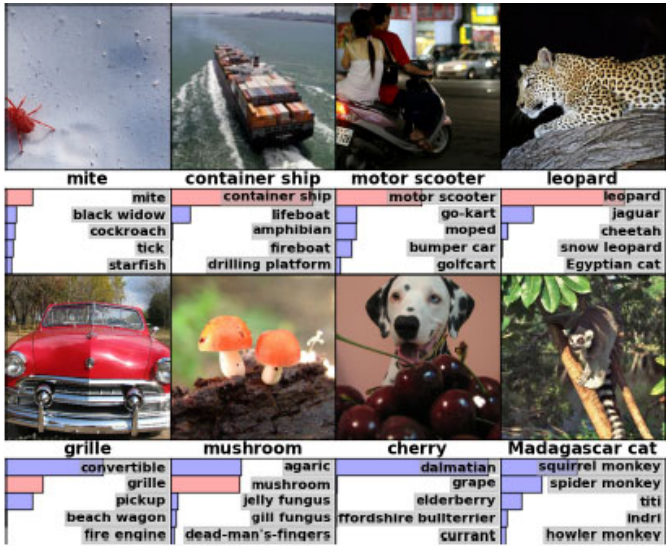
- Remember the positions of the max and fill the other positions with 0



CNNs–Classification (Krizhevsky et al. 2012)

- A landmark in object recognition - AlexNet
- ImageNet competition
  - Large Scale Visual Recognition Challenge (ILSVRC)
  - 1000 categories, 1.5 Million labeled training samples
  - Method: large convolutional net
  - 650K neurons, 630M synapses, 60M parameters
  - Trained with SGD on GPU



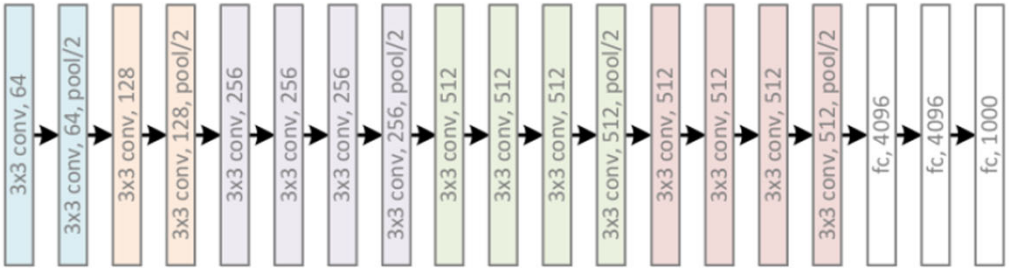


CNNs
Very Deep Nets trained with GPUs

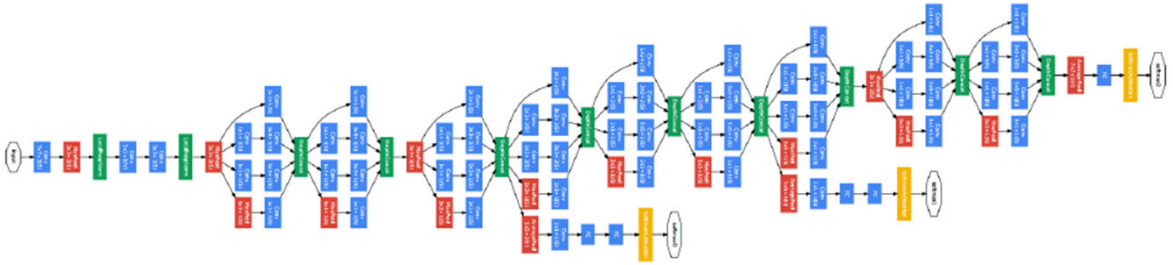
Deeper Nets with small filters – training time several days up to 1 or 2 weeks on ImageNet

Oxford, [Simonyan 2014], Parameters 138 M

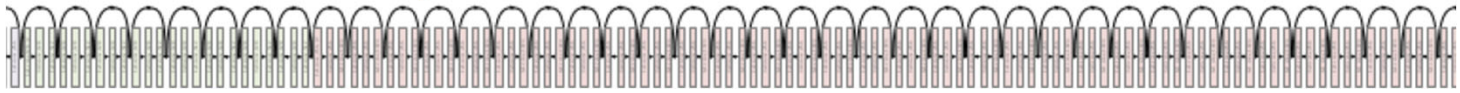
VGG, 16/19 layers, 2014



GoogleNet, 22 layers, 2014 Google, [Szegedy et al. 2015], Parameters 24 M

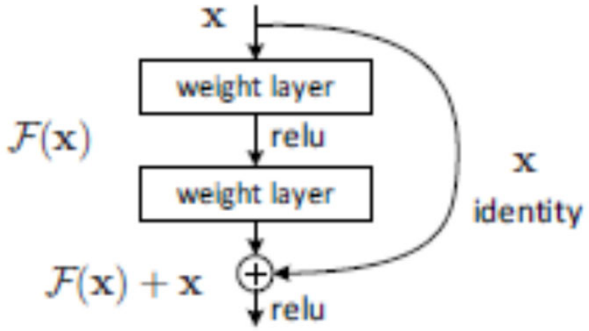


ResNet, 152 layers, 2015 MSRA, [He et al. 2016] , Parameters 60 M

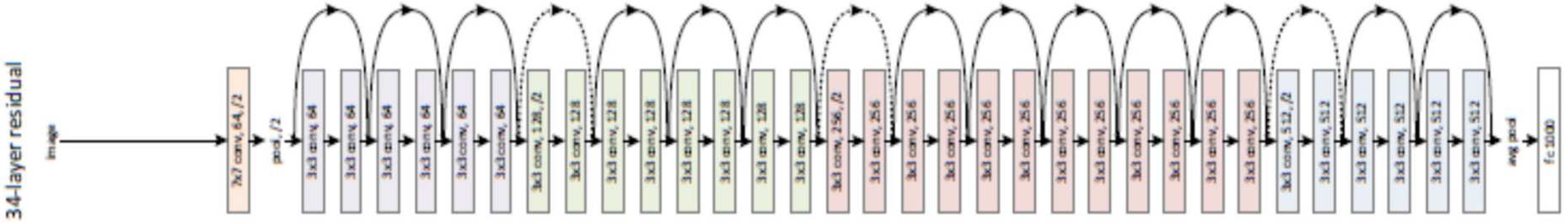


CNNs
ResNet [He et al. 2016]

- 152 ResNet 1st place ILSVRC classification competition
- Other ResNets 1st place ImageNet detection, 1st place ImageNet localization, MS-COCO detection and segmentation
- Main characteristics
  - Building block
    - Identity helps propagating gradients
    - Reduces the vanishing effect
    - $F(x)$ is called the residual
    - Similar ideas used in other models



  - Deep network with small convolution filters
    - Mainly 3x3 convolutional filters



CNNs
ResNet [He et al. 2016b]

- ResNet block
  - $x_{t+1} = x_t + F(x_t, W_t)$
  - $x_T = x_t + \sum_{i=t}^{T-1} F(x_i, W_i)$

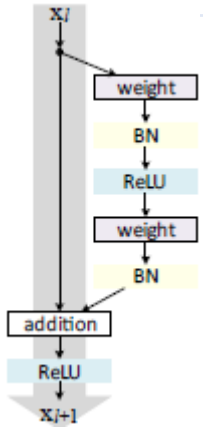


Fig. He 2016, original ResNet block

  - The feature x_T on the last layer can be represented as the feature x_t of layer t plus a residual $\sum_{i=t}^{T-1} F(x_i, W_i)$

- ResNet Backward equation
  - $\frac{\partial C}{\partial x_t} = \frac{\partial C}{\partial x_T}\frac{\partial x_T}{\partial x_t} = \frac{\partial C}{\partial x_T}(1 + \frac{\partial}{\partial x_t}\sum_{i=t}^{T-1} F(x_i, W_i))$
  - Gradient $\frac{\partial C}{\partial x_t}$ can be decomposed in two additive term
    - $\frac{\partial C}{\partial x_T}$ propagates this gradient to any unit
    - $\frac{\partial}{\partial x_t}\sum_{i=t}^{T-1} F(x\_i, W\_i)$ propagates through the weight layers

CNNs
ResNet as a discretization scheme for ODEs (Optional)

- Ordinary Differential Equation
  - $\frac{dX}{dt} = F(X(t), \theta(t)), X(0) = X_0$ (1)
- Resnet module can be interpreted as a numerical discretization scheme for the ODE:
  - $X_{t+1} = X_t + G(X_t, \theta_t)$ - ResNet module (2)
  - $X_{t+1} = X_t + hF(X_t, \theta_t), h \in [0,1]$ (simple rewriting of (2) replacing $G()$ with $hF()$
  - $\frac{X_{t+1} - X_t}{h} = F(X_t, \theta_t)$
    - Forward Euler Scheme for the ODE (1)
    - h time step
  - Note: this type of additive structure (2) is also present in LSTM and GRU units (see RNN section)
- Resnet
  - Input X_t, output X_{t+1}
  - Multiple Resnet modules implement a discretization scheme for the ODE $\frac{dX}{dt} = F(X(t), \theta(t))$
    - $X(t_1) = X(t_0) + hF\left(X(t_0), \theta_{t_0}\right)$
    - $X(t_2) = X(t_1) + hF\left(X(t_1), \theta_{t_1}\right), \dots$

CNNs
Resnet as a discretization scheme for ODEs

- This suggests that alternative discretization schemes will correspond to alternative Resnet like NN models
  - Backward Euler, Runge-Kutta, linear multi-step …
- Example (Lu 2018) linear multi-step discretization scheme
  - $X_{t+1} = (1 - k_t)X_t + k_t X_{t-1} + F(X_t, \theta_t)$

Fig. (Lu 2018)

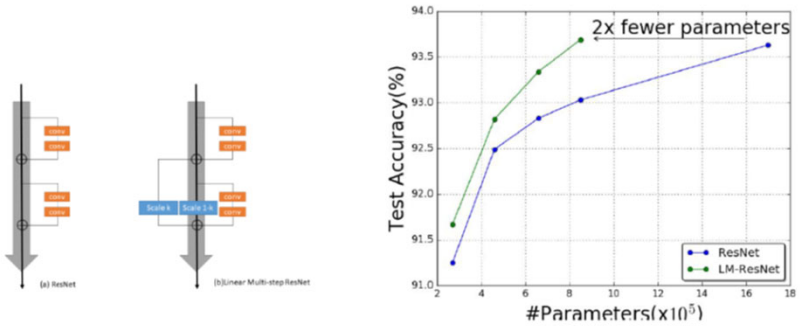


Figure 2: LM-architecture is an efficient structure that enables ResNet to achieve same level of accuracy with only half of the parameters on CIFAR10.

- Applications
  - Classification (a la ResNet)
  - Modeling dynamical systems

Convolutional Nets
ILSVRC performance over the years

- Imagenet 2012 classification challenge

Rank	Name	Error rate	Description
1	**U. Toronto**	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

Object recognition over 1,000,000 images and 1,000 categories (2 GPU)

- ImageNet 2014 – Image classification challenge

Rank	Name	Error rate	Description
1	Google	0.06656	Deep learning
2	Oxford	0.07325	Deep learning
3	MSRA	0.08062	Deep learning

- ImageNet 2014 – object detection challenge

Rank	Name	Mean Average Precision	Description
1	Google	0.43933	Deep learning
2	CUHK	0.40656	Deep learning
3	DeepInsight	0.40452	Deep learning
4	UvA-Euvision	0.35421	Deep learning
5	Berkley Vision	0.34521	Deep learning

- ImageNet 2013 – image classification challenge

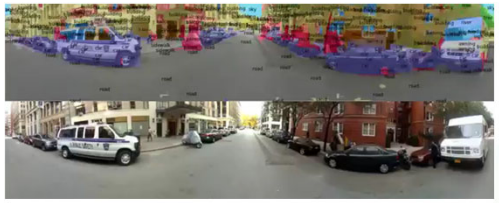
Rank	Name	Error rate	Description
1	NYU	0.11197	Deep learning
2	NUS	0.12535	Deep learning
3	Oxford	0.13555	Deep learning

MSRA, IBM, Adobe, NEC, Clarifai, Berkley, U. Tokyo, UCLA, UIUC, Toronto .... Top 20 groups all used deep learning

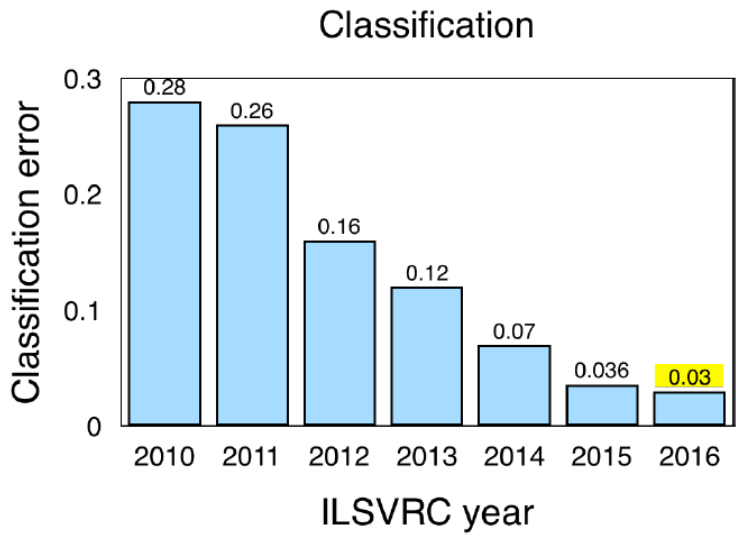
- ImageNet 2013 – object detection challenge

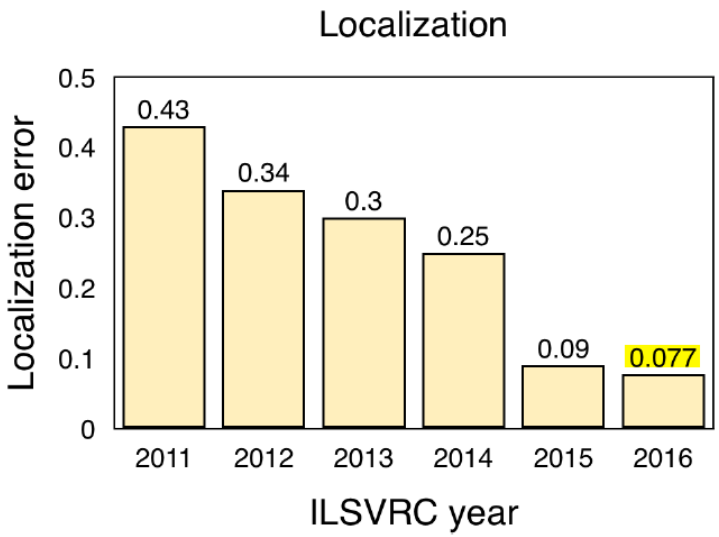
Rank	Name	Mean Average Precision	Description
1	UvA-Euvision	0.22581	Hand-crafted features
2	NEC-MU	0.20895	Hand-crafted features
3	NYU	0.19400	Deep learning

CNN examples



Convolutional Nets
ILSVRC performance over the years

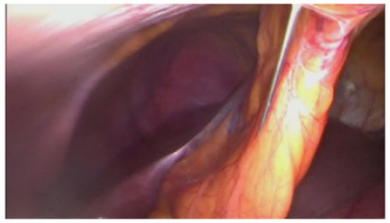


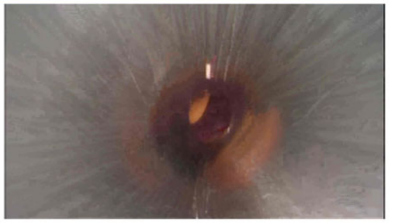


Classification
CNNs and Transfer Learning

- Training large NN requires
  - large amount of labeled data
  - Large GPU clusters
- Large labeled datasets are not available for all applications
- Deep Networks **pretrained** with large datasets like ImageNet are used for other applications after some retraining/ fine tuning:
  - Classification of images from different nature
  - Classification of objects in large size images
  - Object detection, Segmentation
  - Learning latent representations of images
- Remark
  - CNN trained on ImageNet have specific characteristics
    - e.g. input: 224x224 images, centered on the objects to be classified
    - How to adapt them to other collections?

Classification - Transfer learning - CNNs - Images from different nature,M2CAI Challenge (Cadene 2016)





- Endoscopic videos (large intestine)
  - resolution of 1920 x 1080, shot at 25 frame per second at the IRCAD research center in Strasbourg, France. 27 training videos ranging from 15mn to 1hour, 15 testing videos
- Used for: monitor surgeons, Trigger automatic actions
- Objective: classification, 1 of 8 classes for each frame
  - TrocarPlacement, Preparation, CalotTriangleDissection, ClippingCutting, GallbladderDissection, GallbladderPackaging, CleaningCoagulation, GallbladderRetraction
- Resnet 200 pretrained with ImageNet -> reaches 80% correct classification

Model	Input	Param.	Depth	Implem.	Forward (ms)	Backward (ms)
Vgg16	224	138M	16	GPU	185.29	437.89
InceptionV3 [2]	399	24M	42	GPU	**102.21**	311.94
ResNet-200 [3]	224	65M	200	GPU	273.85	687.48
InceptionV3	399	24M	42	CPU	19918.82	23010.15

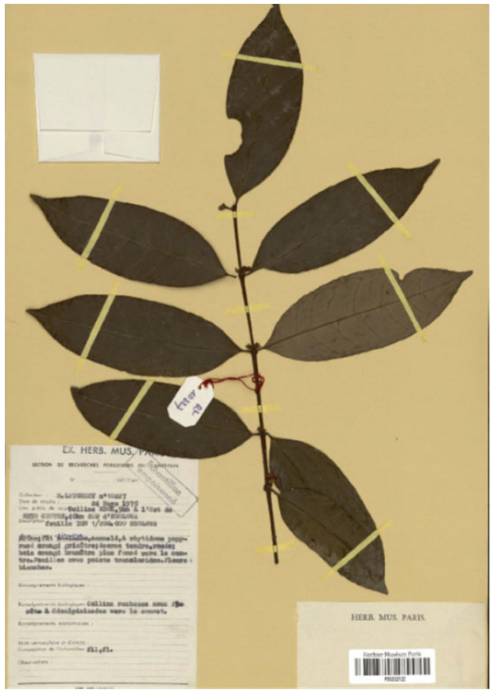
Table 1: Forward+Backward with batches of 20 images.

InceptionV3	Extraction (repres. of ImageNet)	60.53
InceptionV3	From Scratch (repres. of M2CAI)	69.13
InceptionV3	Fine-tuning (both representations)	79.06
ResNet200	**Fine-tuning (both representations)**	**79.24**

Table 2: Accuracy on the validation set.

Classification - Transfer learning - CNNs - Images from different nature, Plant classification (Wu 2017)

- Digitized plant collection from Museum of Natural History – Paris
- Largest digitized world collection (8 millions specimens)
- Goal
  - Identify plants characteristics for automatic labeling of worlwide plant collections
  - O(1000) classes, e.g. opposed/alternate leaves; simple/composed leaves; smooth/with teeth leaves, ….
- Pretrained ResNet



Classification - Fully convolutional nets

CNNs – Classification of large images (Fig. Durand 2016)

How to deal with complex scenes?

Pascal VOC style

ImageNet style

- Working on datasets with complex scenes (large and cluttered background), not centered objects, variable size, ...

VOC07/ 12

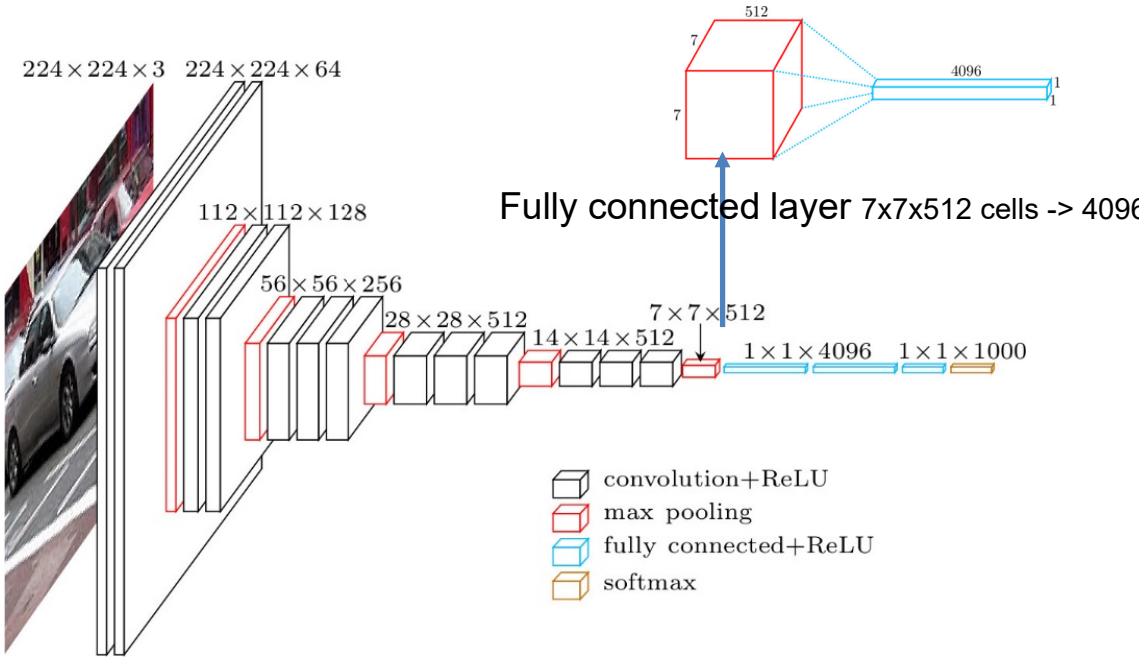
MIT67

15 Scene

COCO

VOC12 Action

Classification - CNNs – Classification of large images (Durand 2016)
Sliding window => Convolutional Layers



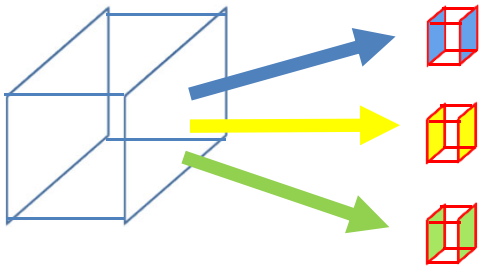
ImageNet trained CNN

- Sliding window:
  - Use the ImageNet trained CNN as a sliding window (a convolution filter) on the large image
  - In order to do that, one must **convert the fully connected layer**
    7x7x512 cells → 4096 cells
    into a convolutional layer

Converting Fully Convolutional Nets (FCN) to CNN

- Fully connected layers can be converted to convolutional nets
  - The following scheme is equivalent to 3 output cells fully connected to the input cells, but is expressed as a convolution
  - Colors correspondance below

FCN classical view

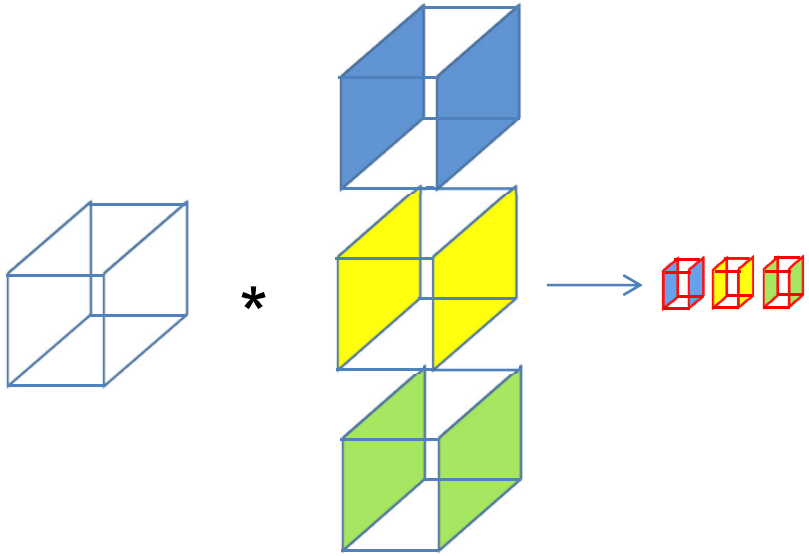


Cell layer
$nxnxC$

Each weight
vector is
$nxnxC$

Outputs
3 cells

FCN convolutional view



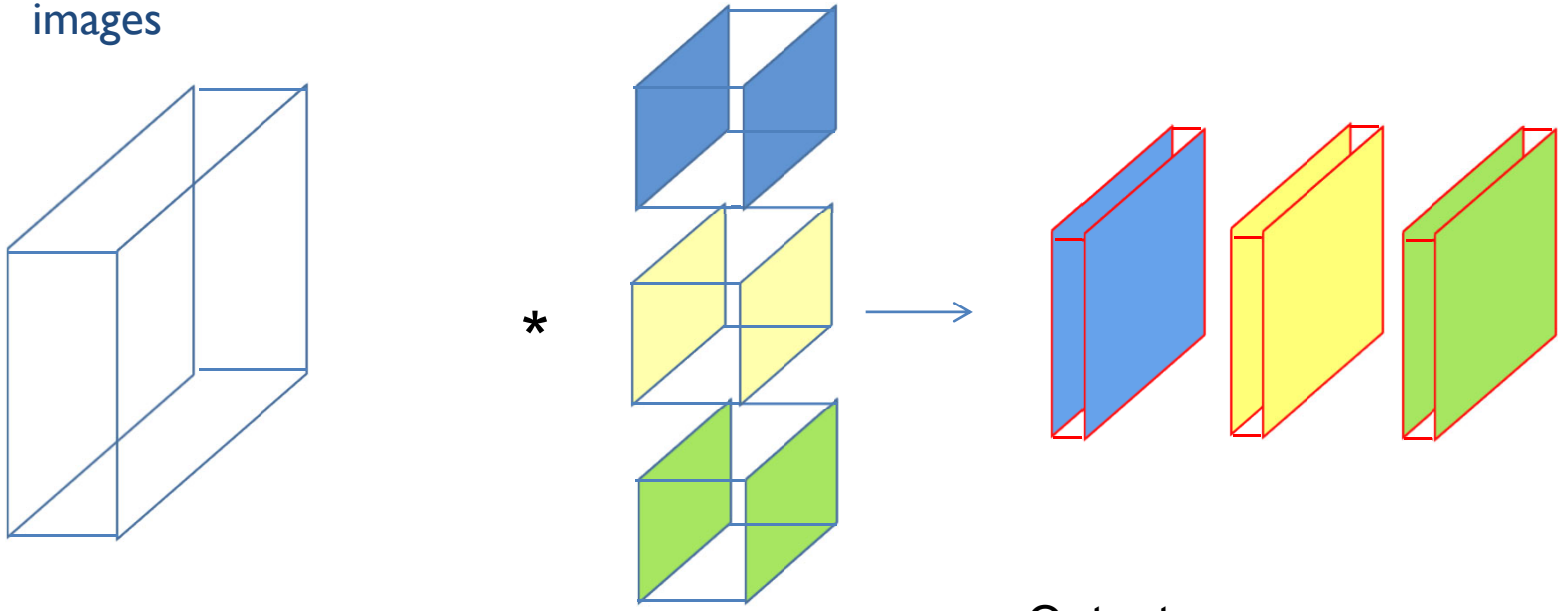
Cell layer
$nxnxC$

Weight layer
each is $nxnxC$

Outputs
3 cells

Converting Fully Convolutional Nets (FCN) to CNN

- Fully connected layers can be converted to convolutional nets
  - This does not change anything if the input size is the size of the weight layer
  - It can be used as a convolution for larger input sizes, and then produces larger outputs
  - In this way, pre-trained networks can be used without retraining for larger images

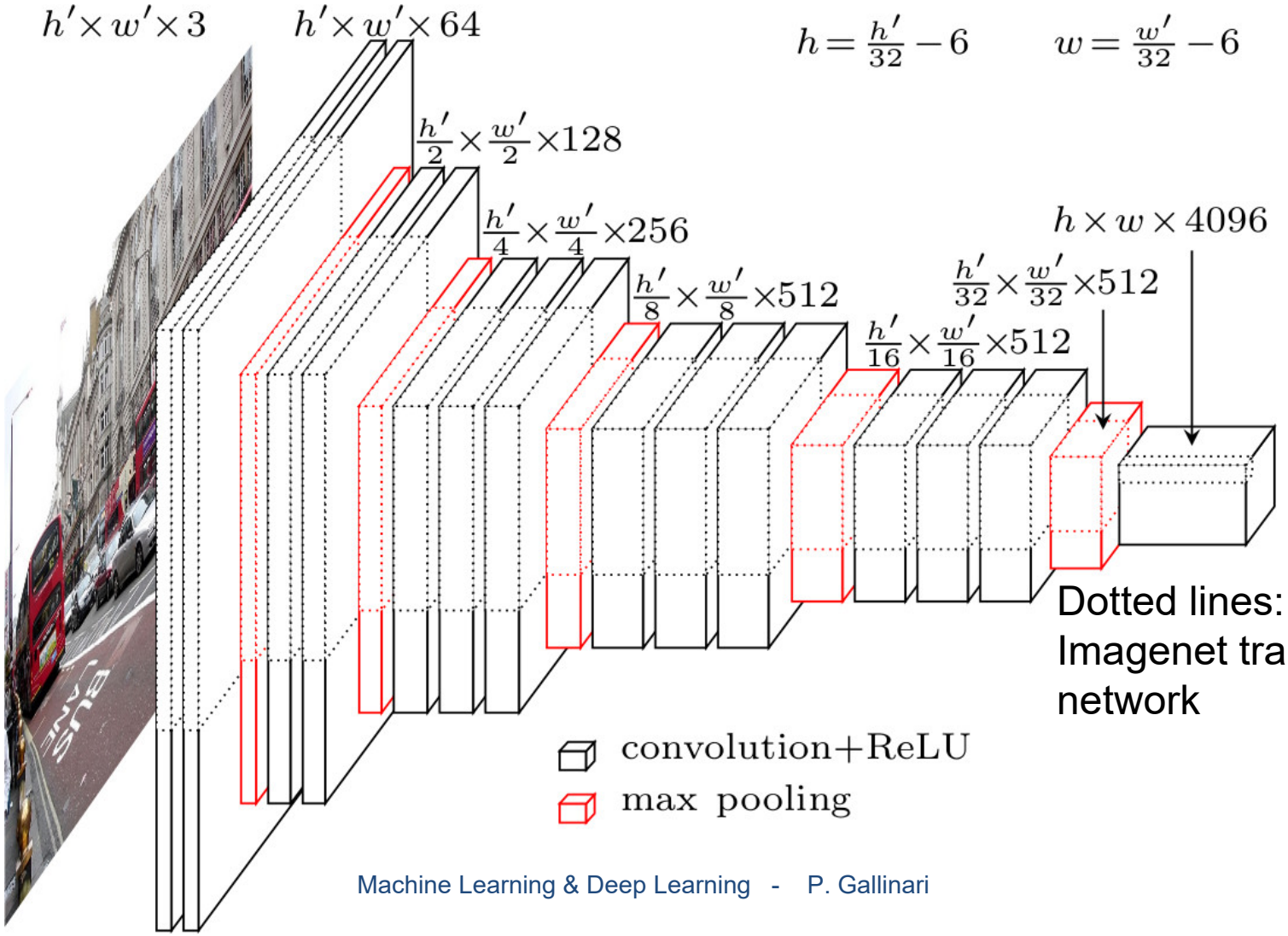


Cell layer
NxNxC

Weight layer
nxnxC each

Outputs
$(N-n+1)$x$(N-n+1)$x1 each

CNNs – Classification of large images (Durand 2016)
Sliding window => Convolutional Layers



Dotted lines: initial Imagenet trained network

CNNs – Classification of large images (Sermanet et al. 2014)
Sliding window => Convolutional Layers

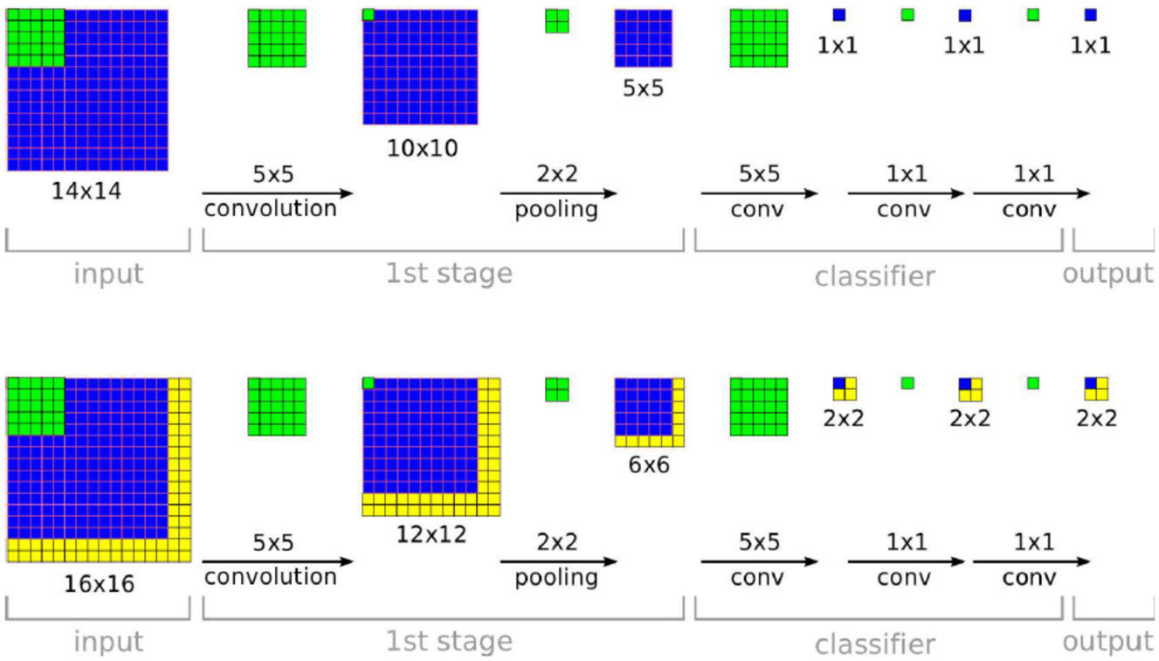
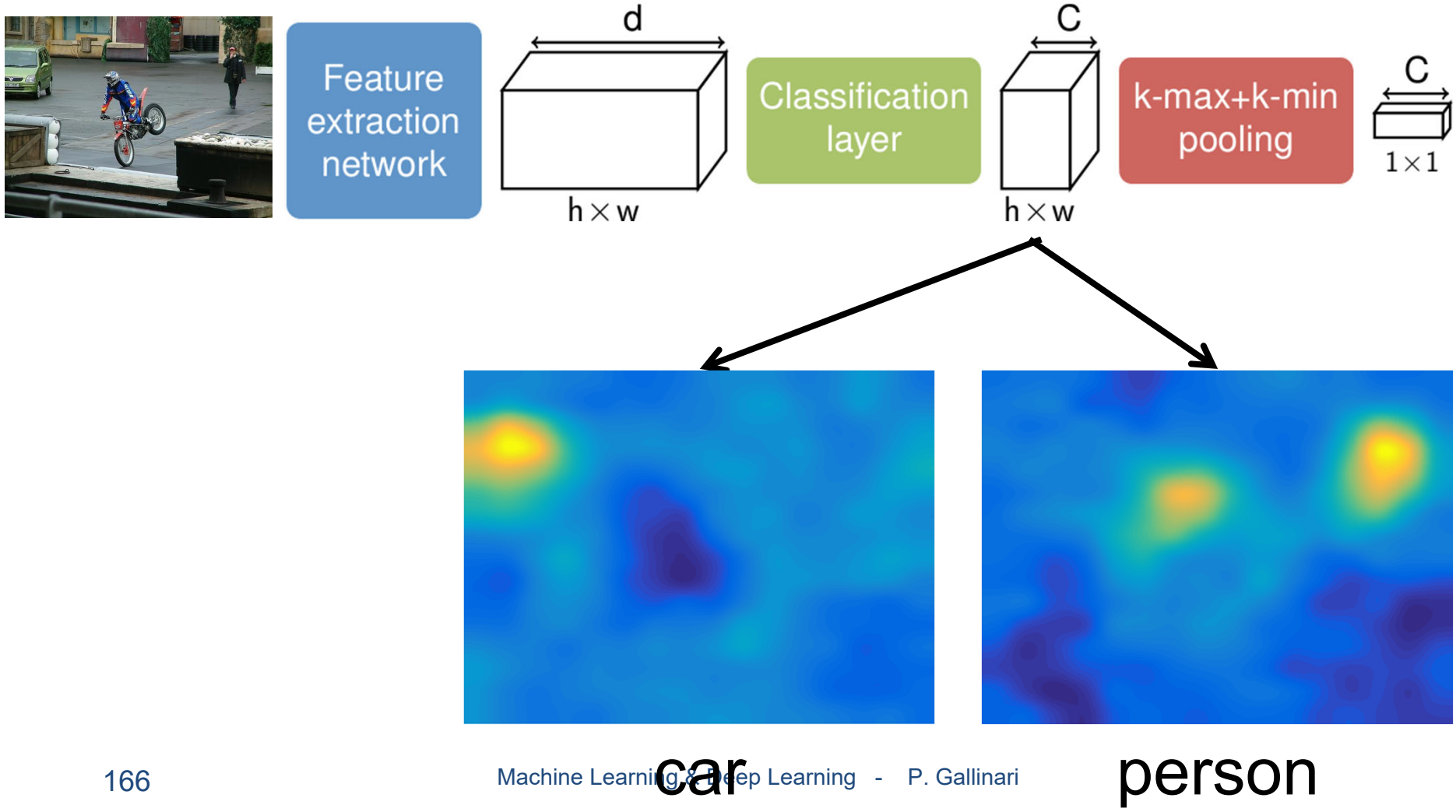


Figure 5: **The efficiency of ConvNets for detection.** During training, a ConvNet produces only a single spatial output (top). But when applied at test time over a larger image, it produces a spatial output map, e.g. 2x2 (bottom). Since all layers are applied convolutionally, the extra computation required for the larger image is limited to the yellow regions. This diagram omits the feature dimension for simplicity.

Fig: Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks, 2014

CNNs – Classification of large images (Durand 2016)
Sliding window => Convolutional Layers



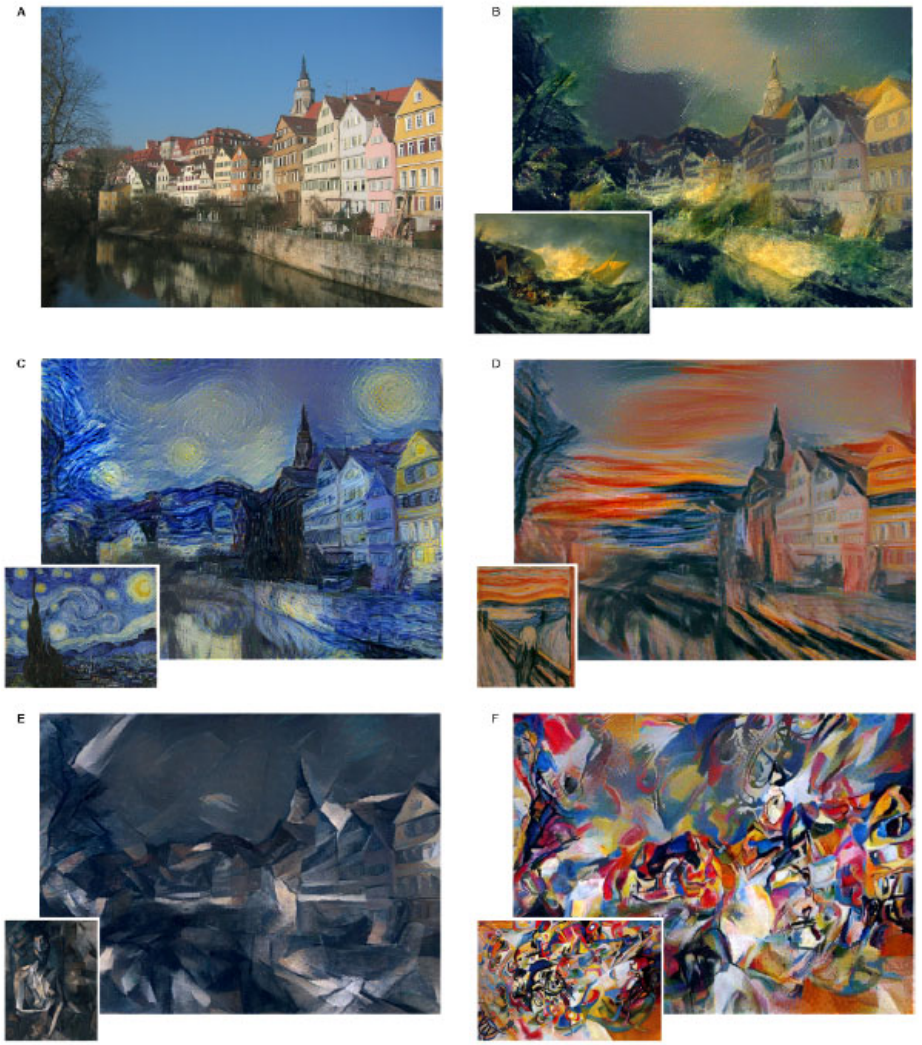
CNN : A neural algorithm of Artistic Style (Gatys et al. 2016)

Generate images by combining content and style

Makes use of a discriminatively trained CNN

Image generation

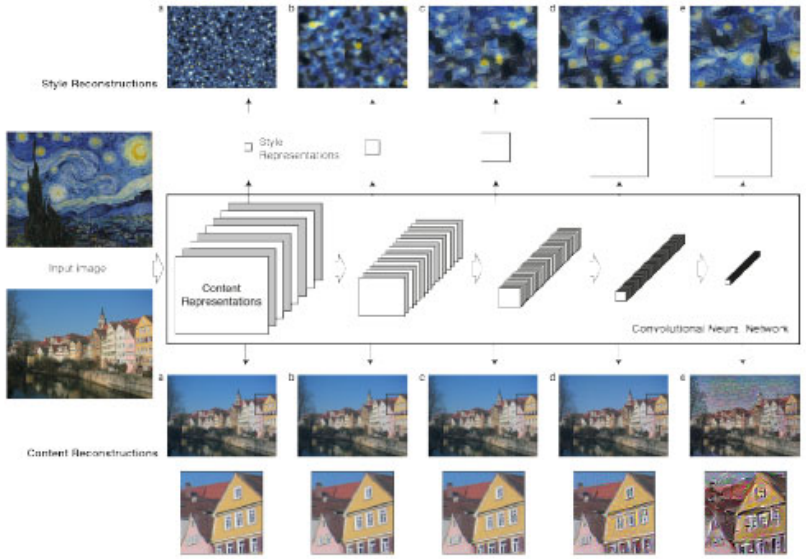
- inverse problem on the CNN



https://deepart.io

CNN : A neural algorithm of Artistic Style (Gatys et al. 2016)

- Idea (simplified)
  - Use a pre-trained ImageNet NN
  - $\boldsymbol{c}$ input content image, F_c a filter representation of $\boldsymbol{c}$
  - $\boldsymbol{a}$ input art image, G_a a filter correlation representation of $\boldsymbol{a}$
  - $\boldsymbol{x}$ a white noise image, F_x and G_x the corresponding filter and filter correlation representations
  - loss:
    - $L = \|F_c - F_x\|^2 + \alpha\|G_a - G_x\|^2$
- Generated image
  - Solve an inverse problem
    - $\hat{\boldsymbol{x}} = argmin_x(L)$
    - Solved by gradient



CNN : A neural algorithm of Artistic Style (Gatys et al. 2016)

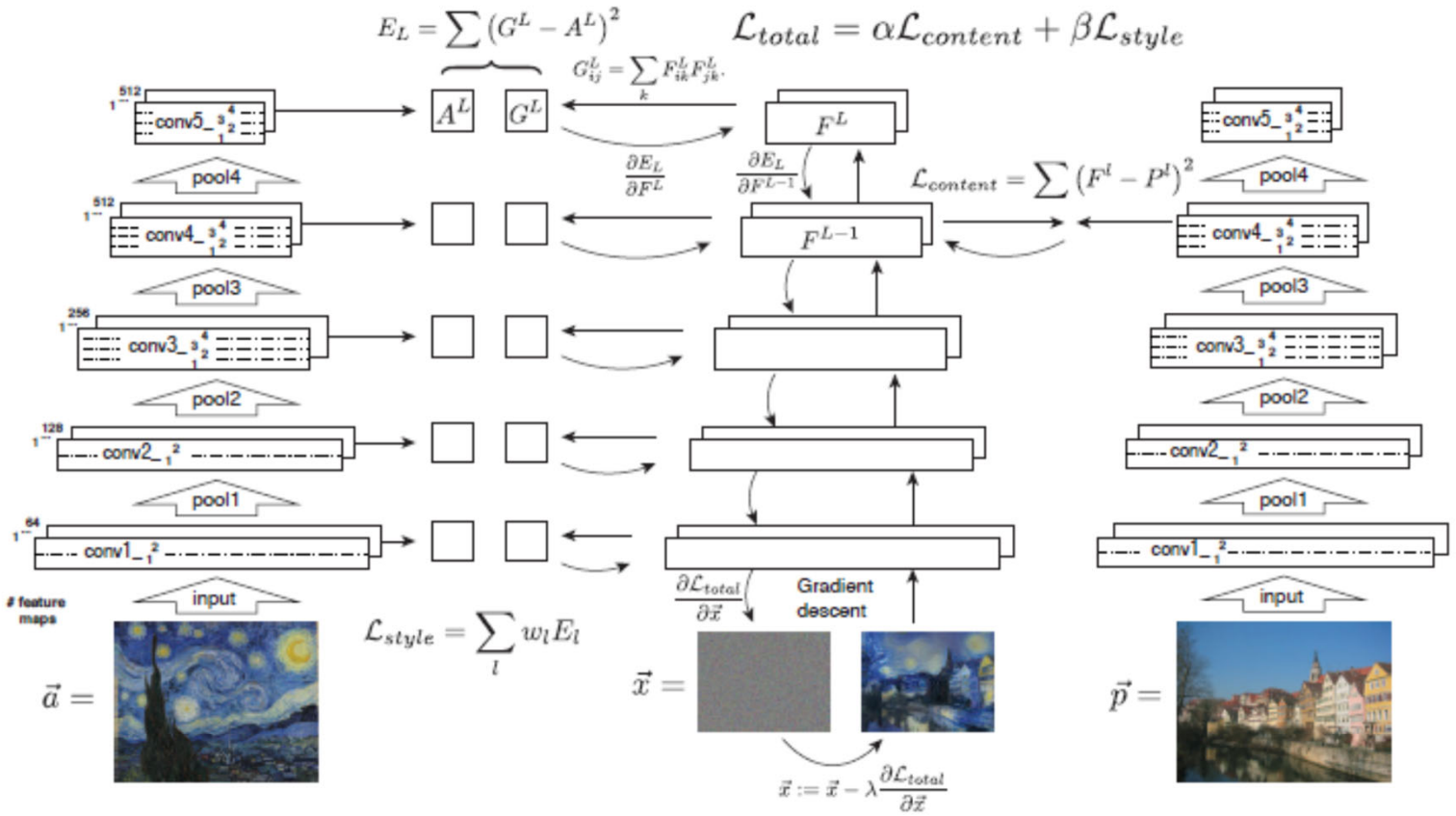


Figure 2. Style transfer algorithm. First content and style features are extracted and stored. The style image $\vec{a}$ is passed through the network and its style representation A^l on all layers included are computed and stored (left). The content image $\vec{p}$ is passed through the network and the content representation P^l in one layer is stored (right). Then a random white noise image $\vec{x}$ is passed through the network and its style features G^l and content features F^l are computed. On each layer included in the style representation, the element-wise mean squared difference between G^l and A^l is computed to give the style loss $\mathcal{L}_{style}$ (left). Also the mean squared difference between F^l and P^l is computed to give the content loss $\mathcal{L}_{content}$ (right). The total loss $\mathcal{L}_{total}$ is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle). This gradient is used to iteratively update the image $\vec{x}$ until it simultaneously matches the style features of the style image $\vec{a}$ and the content features of the content image $\vec{p}$ (middle, bottom).

Object detection

- Objective: predicting classes and location of objects in an image
  - Usually the output of the predictor is a series of bounding boxes with an object class label
- Performance measure
  - Let B a target bounding box and $\hat{B}$ the predicted one
  - Intersection over Union: $IoU = \frac{area(B \cap \hat{B})}{\text{area}(B U \hat{B})}$
- Training
  - Supervised training, e.g. Pascal Voc Dataset

PASCAL Annotation Version 1.00 Image filename : "TUDarmstadt/PNGImages/motorbike-testset/motorbikes040-rt.png"
Image size (X x Y x C) : 400 x 275 x 3
Database : "The TU Darmstadt Database«
Objects with ground truth : 2 { "PASmotorbikeSide" "PASmotorbikeSide" }
Note that there might be other objects in the image # for which ground truth data has not been provided.
Top left pixel co-ordinates : (1, 1)
Details for object 1 ("PASmotorbikeSide")
Original label for object 1 "PASmotorbikeSide" : "motorbikeSide«
Bounding box for object 1 "PASmotorbikeSide" (Xmin, Ymin) - (Xmax, Ymax) : (57, 133) - (329, 265)
Details for object 2 ("PASmotorbikeSide")
Original label for object 2 "PASmotorbikeSide" : "motorbikeSide«
Bounding box for object 2 "PASmotorbikeSide" (Xmin, Ymin) - (Xmax, Ymax) : (153, 95) - (396, 218)

- Teaser YOLO démos
  - First paper 2015 (J.Redmon who developed V1 to V3)
  - YOLOV2 - https://www.youtube.com/channel/UC7ev3hNVkx4DzZ3LO19oebg?app=desktop&cbrd=1&ucbcb=1
  - YOLOV3 - https://www.youtube.com/watch?v=MPU2HistivI
  - Other actors developed further versions, YOLOV5, V6

CNNs for Object detection
Case study: YOLO (Redmon 2015), https://goo.gl/bEs6Cj

- Classical CNN architecture
- Divides the input image into a SxS grid
  - Each grid cell predicts
    - B bounding boxes and confidence for these boxes
      - 5 numbers per box: (x, y): $box\ center$, (w, h): $box\ dimension$, $confidence$
      - $confidence = P(Object).IoU(target, pred)$
        - $P(Object)$ is the probability that an object appears in a grid cell
    - The class probability for the object if any (only one object/ cell grid), i.e. 1 prediction / cell
      - $P(Class|Object)$
      - Note: at inference time they use the following score
        - $P(Class|object).P(Object).IoU(target, pred)$ instead of $P(Class|Object)$
          - This includes confidence
      - Only the boxes/classes with the higher score are kept

CNNs for Object detection
Case study: YOLO (Redmon 2015)

Fig. Redmon 2015

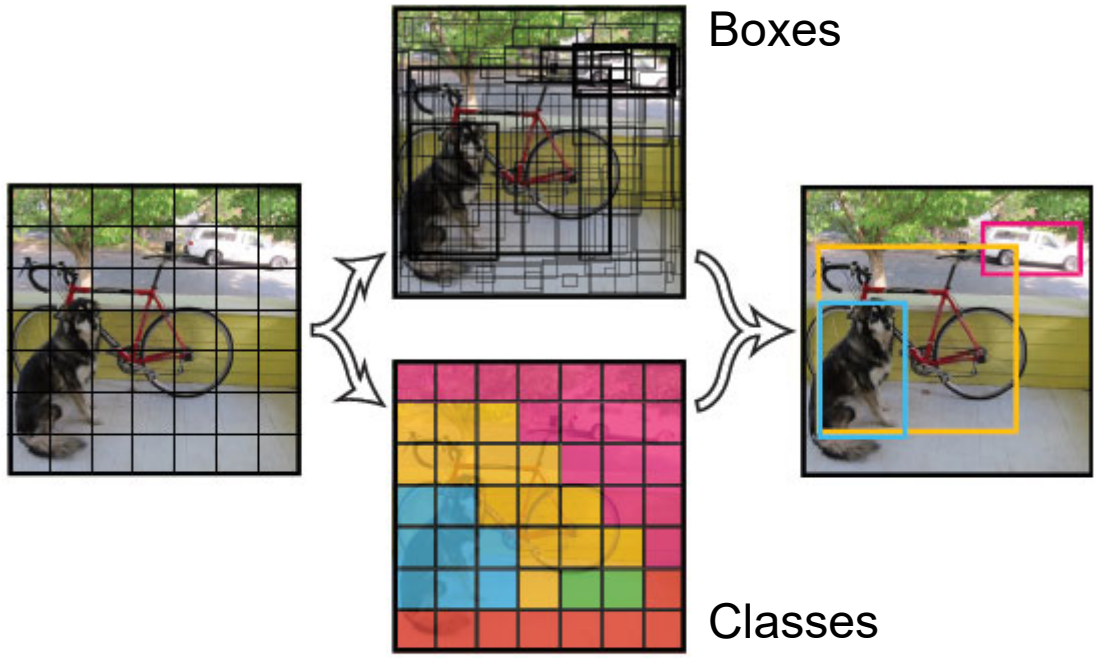


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an even grid and simultaneously predicts bounding boxes, confidence in those boxes, and class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

CNNs for Object detection
Case study: YOLO (Redmon 2015) - Network Design

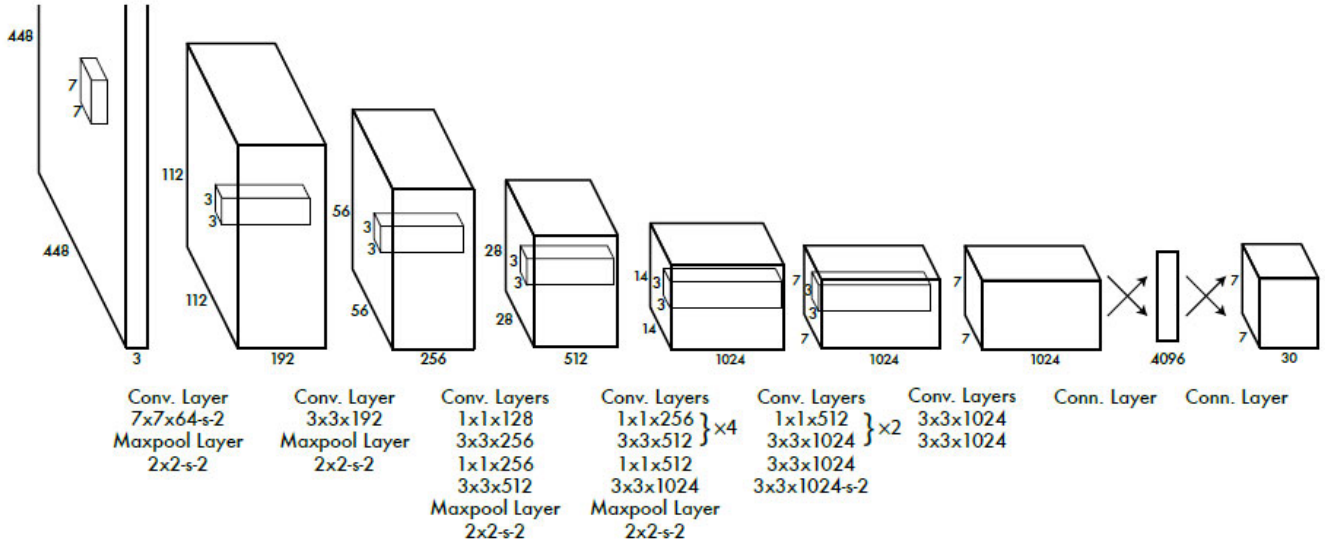


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Output : SxSx(Bx5 + C) tensor

for Pascal Voc dataset: SxSx(Bx5 + C) = 7x7x(2x5 + 20)

With B: # boxes and C: # classes

Several 1x1xn convolutional structures to reduce the feature space from preceeding layers

CNNs for Object detection
Case study: YOLO (Redmon 2015) - Design and Training

- Pretrained on ImageNet 1000 class
- Remove classification layer and replace it with 4 convolutional layers + 2 Fully Connected layers
- Activations: Linear for the last layer, leaky reLu for the others
- Requires a lot of know-how (design, training strategy, tricks, etc)
  - Not described here – see paper…
- Improved versions followed the initial paper
- Generalizes to other types of images:

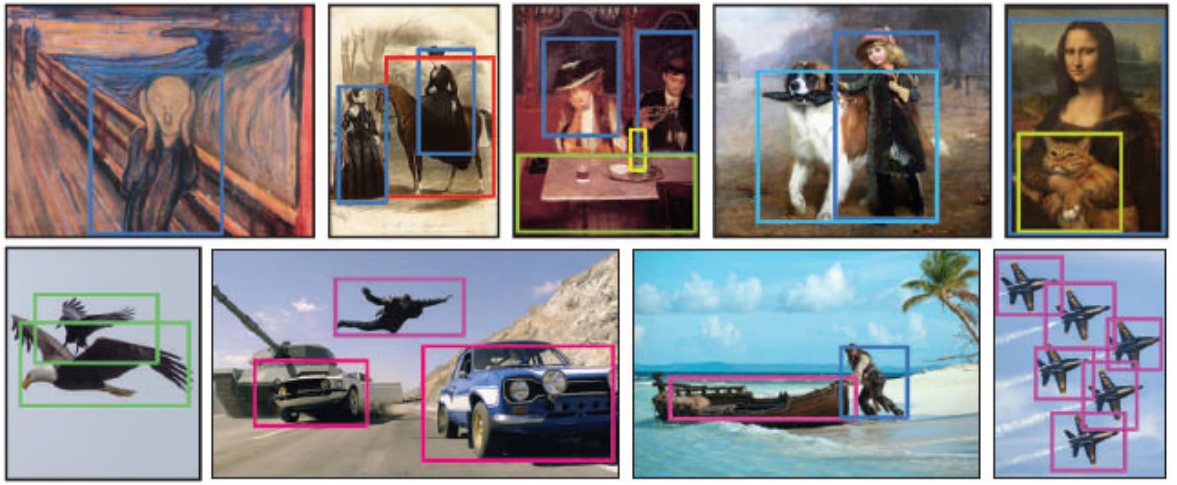
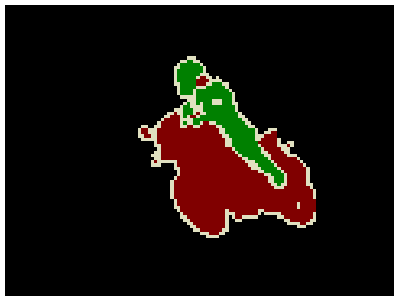


Figure 6: **Qualitative Results.** YOLO running on artwork and natural images. It is mostly accurate although it does think one person in an image is an airplane.

Image Semantic Segmentation

- Objective
  - Identify the different objects in an image



  - Microsoft demo 2015 https://www.youtube.com/watch?v=FroRjEejA30
- Deep learning
  - handles segmentation as pixel classification
  - re-uses network trained for image classification by making them fully convolutional
  - Currently, SOTA is Deep Learning
- Main datasets
  - Voc2012, http://host.robots.ox.ac.uk/pascal/VOC/voc2012/
  - MSCOCO, http://mscoco.org/explore/

CNNs for Image Semantic Segmentation

- DL for segmentation massively re-uses CNN architectures pretrained for classification
  - This is another example of transfer learning
  - Here the goal is to generate classification **at the pixel level** and not at the global image level
    - Means that the output should be the same size (more or less) as the original image, with each pixel labeled by an object Id.
    - Full connections: too many parameters
      - How to keep a pixelwise precision with a low number of parameters
  - Two solutions have been developped
    - Encoder – Decoder architectures with skip connections
      - Encoder are similar to the ones used for classification and decoders use Transpose Convolutions and Unpooling
    - Dilated or a Trous convolutions : remove the Pooling/Unpooling operation

CNNs for Image Semantic Segmentation
Encoder-Decoder - Fully Convolutional Nets (Shelhamer 2016)

- One of the first contribution to DL semantic segmentation, introduces several ideas
- Auto-encoder with skip connections

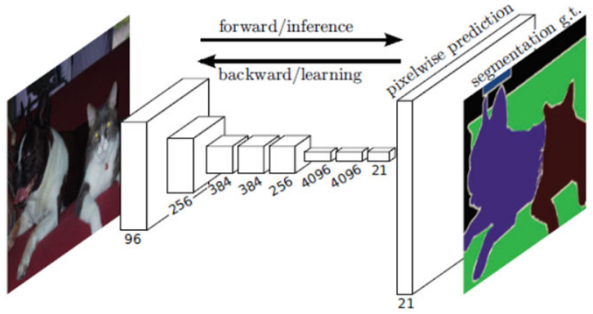


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

- Fully connected -> convolutional trick

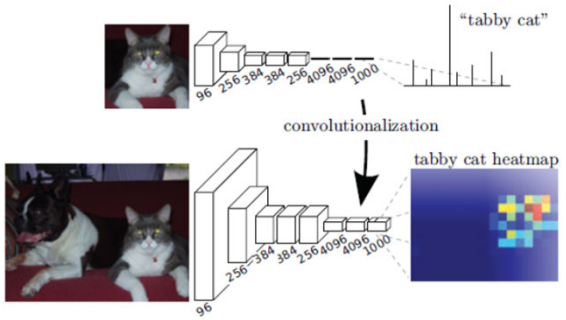


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.

- End to end training for segmentation

CNNs for Image Semantic Segmentation
Encoder-Decoder - Fully Convolutional Nets (Shelhamer 2016)

- FCN architecture: **upsampling** and **skip connections**
  - Training loss = per pixel cross entropy
  - Their initial pipeline (red rectangle) requires x 32 upsampling
  - Improved results where obtained by combining several resolutions in the DNN

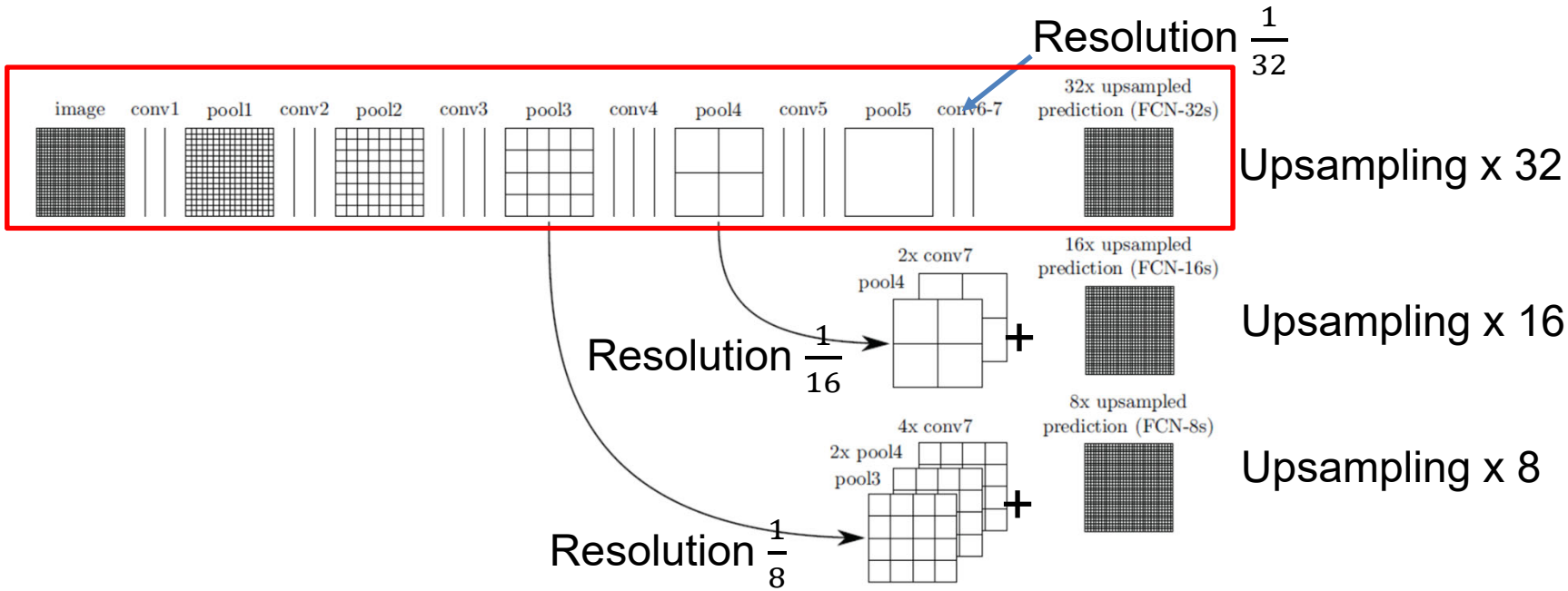


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.

Segmentation
Encoder-Decoder - Other models based on the same ideas

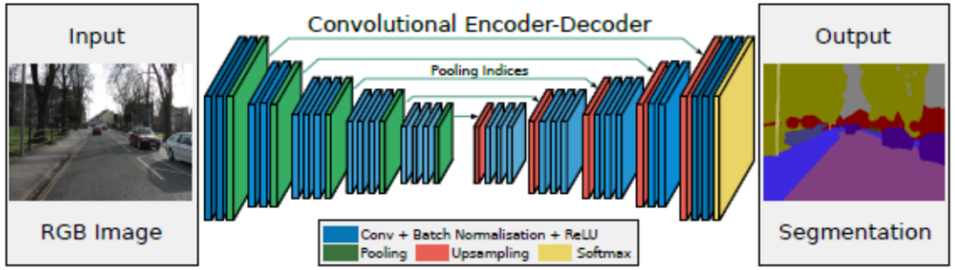


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

SegNet – (Badrinarayanan 2017)

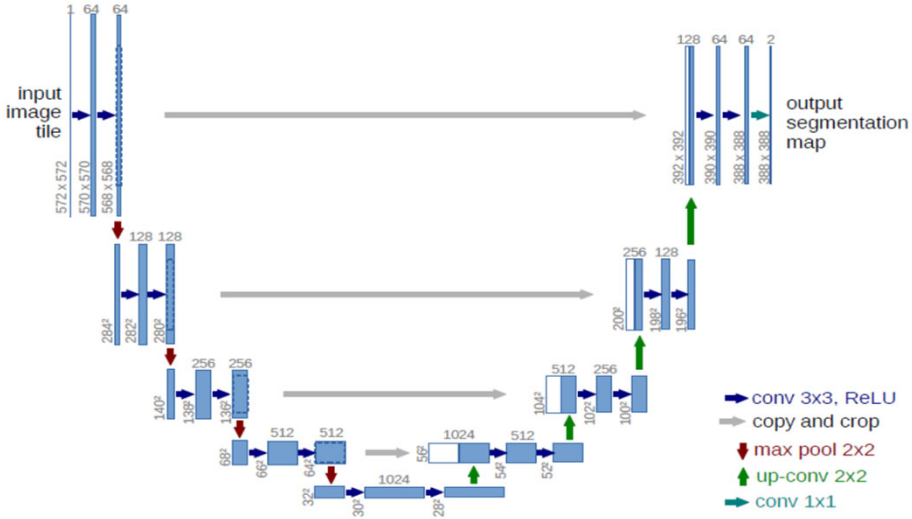


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

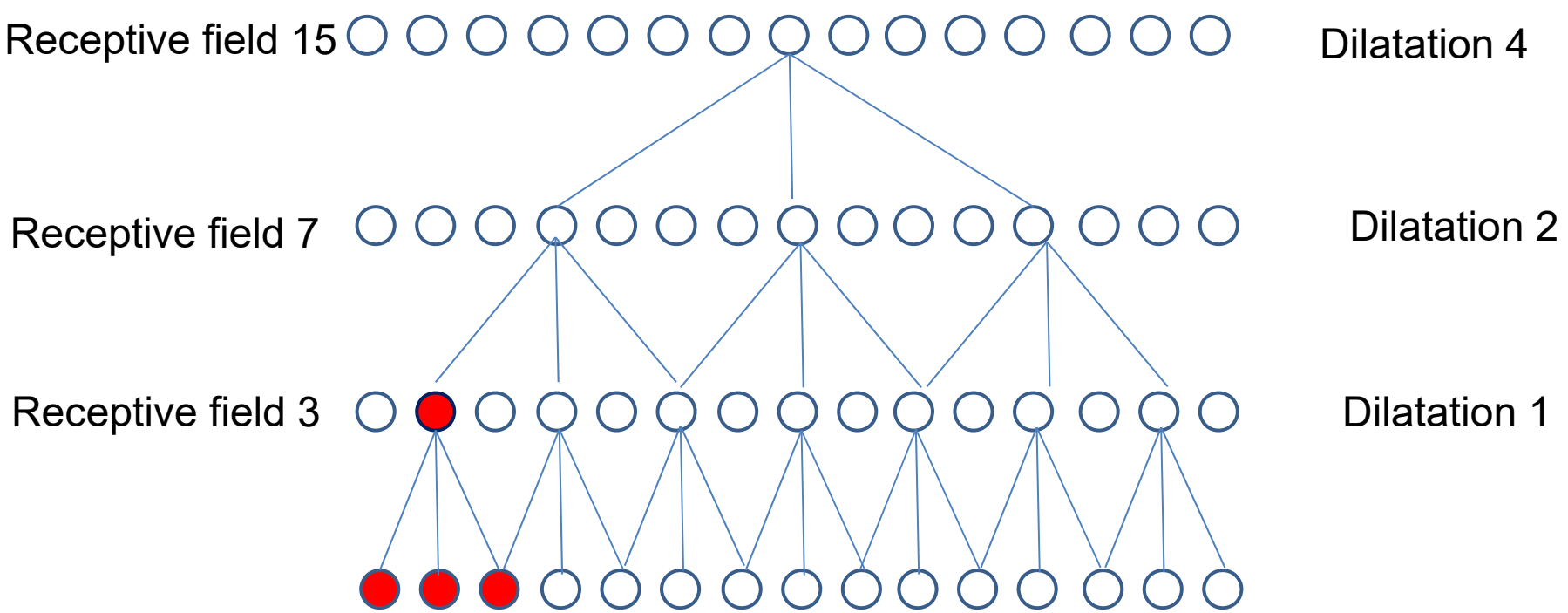
Popular U-Net, (Ronneberger 2015)

Segmentation
Dilated convolutions (Yu 2016)

- Pooling used for classification is not adapted to segmentation
  - The link with individual pixels is lost
- Proposed method
  - Start from a Deep CNN trained from classification.
  - Remove the last Fully Connected and Pooling layers
  - Replace them with Dilated Convolution layers
    - Dilated convolution layers organized hierarchically allow to keep large feature maps for individual neurons with a « small » number of connections
    - Size of the input is the same as the size of the output
      - No downsampling as with pooling, i.e. keep the resolution

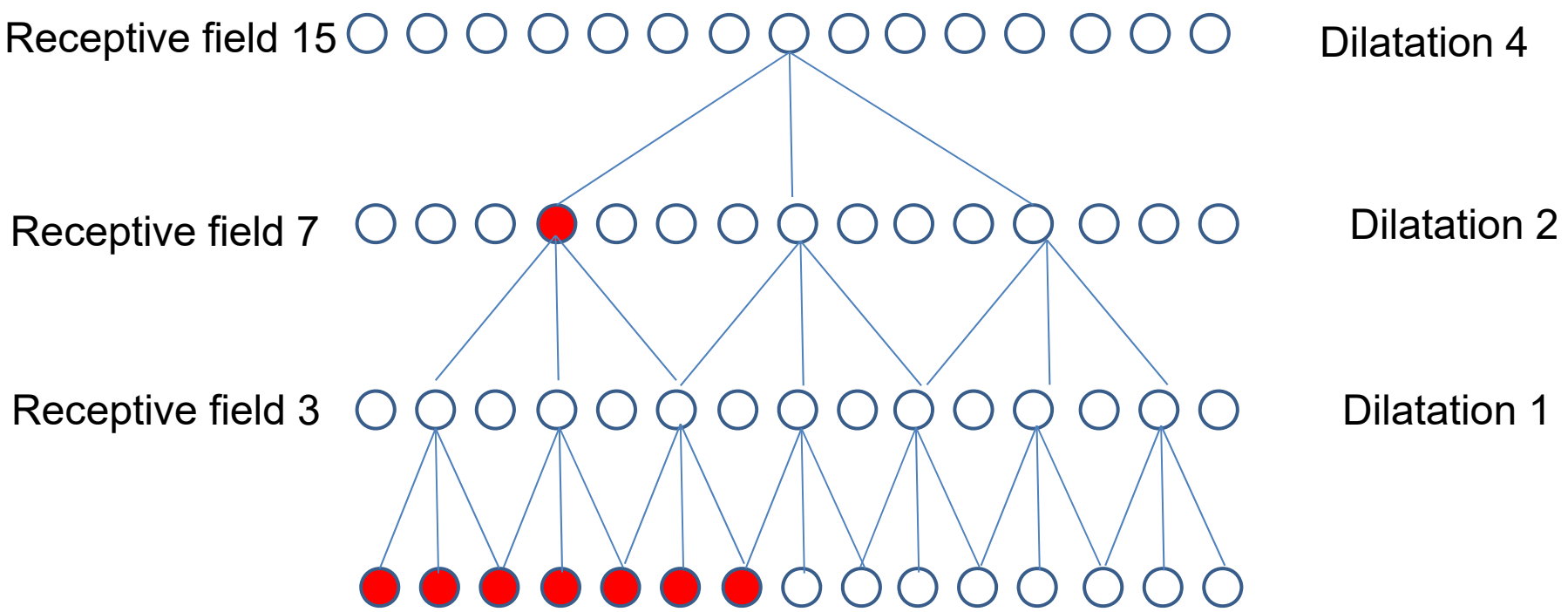
Segmentation
Dilated convolutions (Yu 2016)

- 1 D example



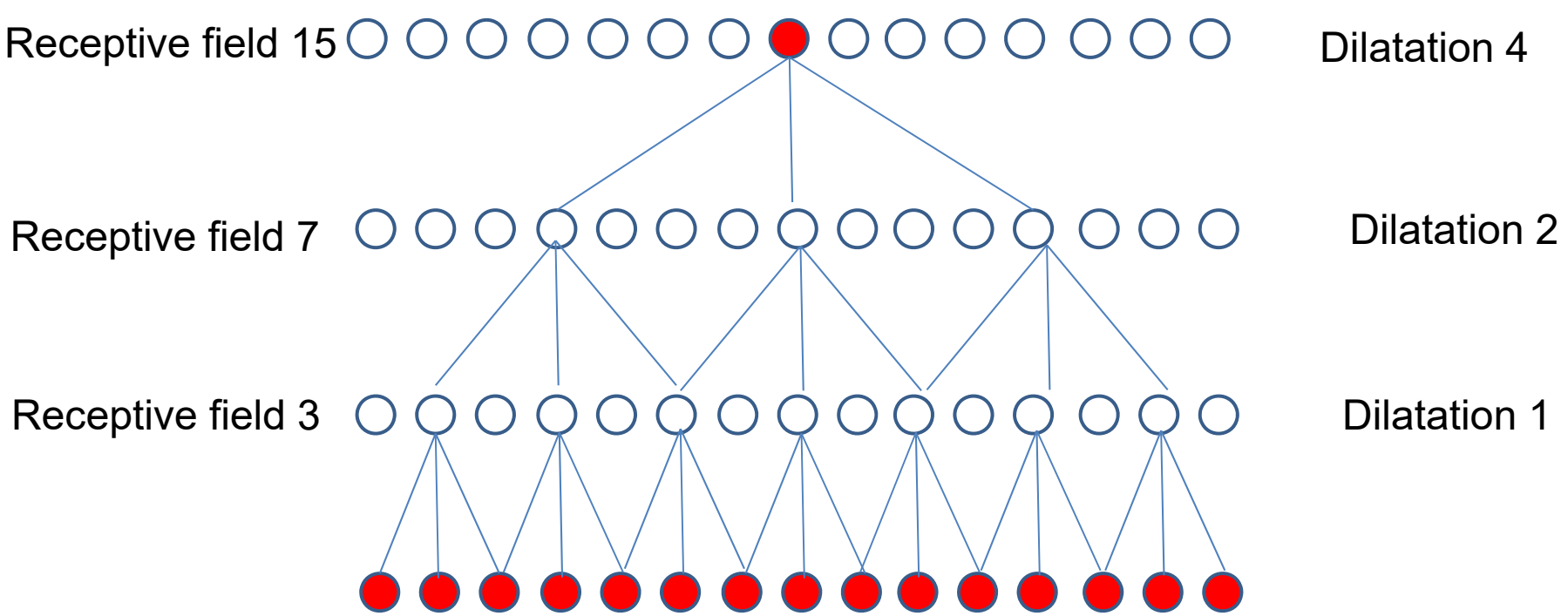
Segmentation
Dilated convolutions (Yu 2016)

- 1 D example



Segmentation
Dilated convolutions (Yu 2016)

- 1 D example



Segmentation
Dilated convolutions (Yu 2016)

- In 2 D

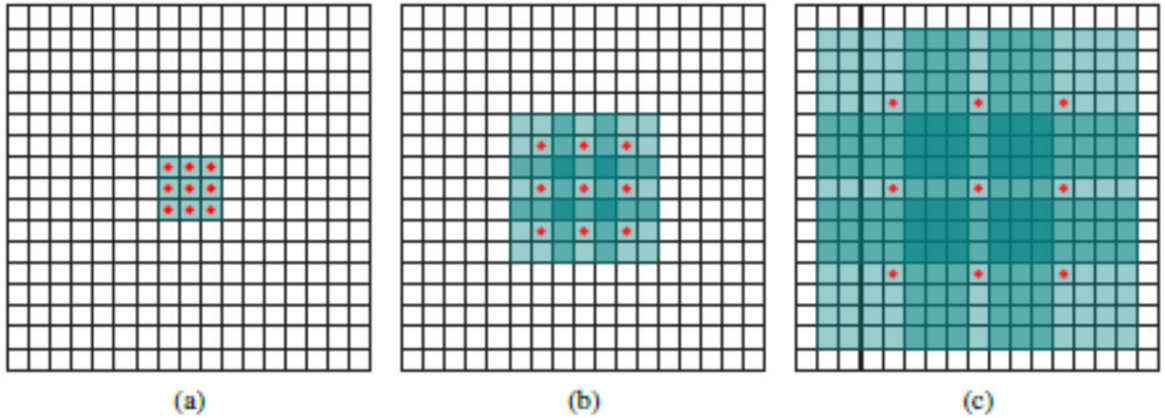


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Fig from (Yu 2016)

- More recent architectures use improved versions of these two ideas

- Noisy data for vision
  - Random rotations
  - Random flips
  - Random shifts
  - Random “zooms”
  - Recolorings