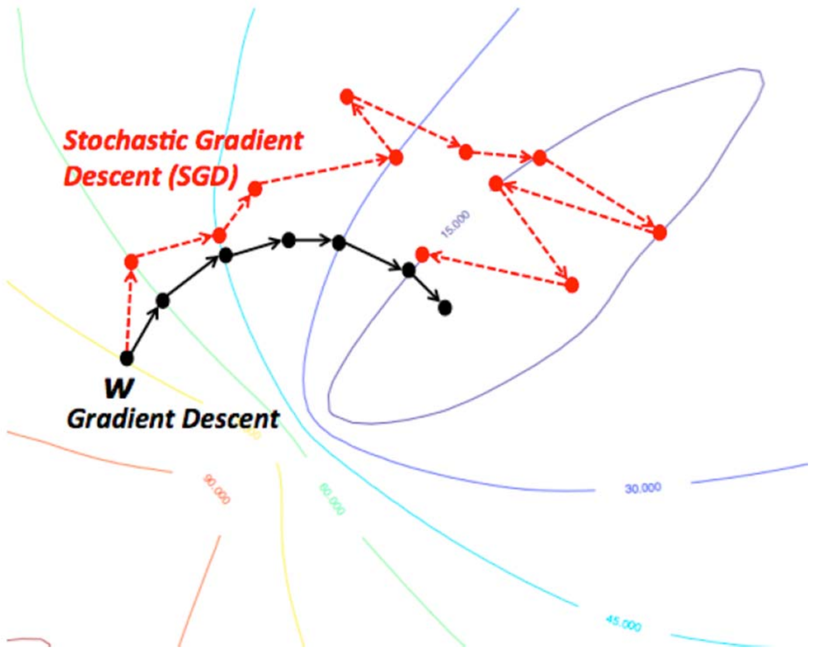
Introduction to Machine Learning



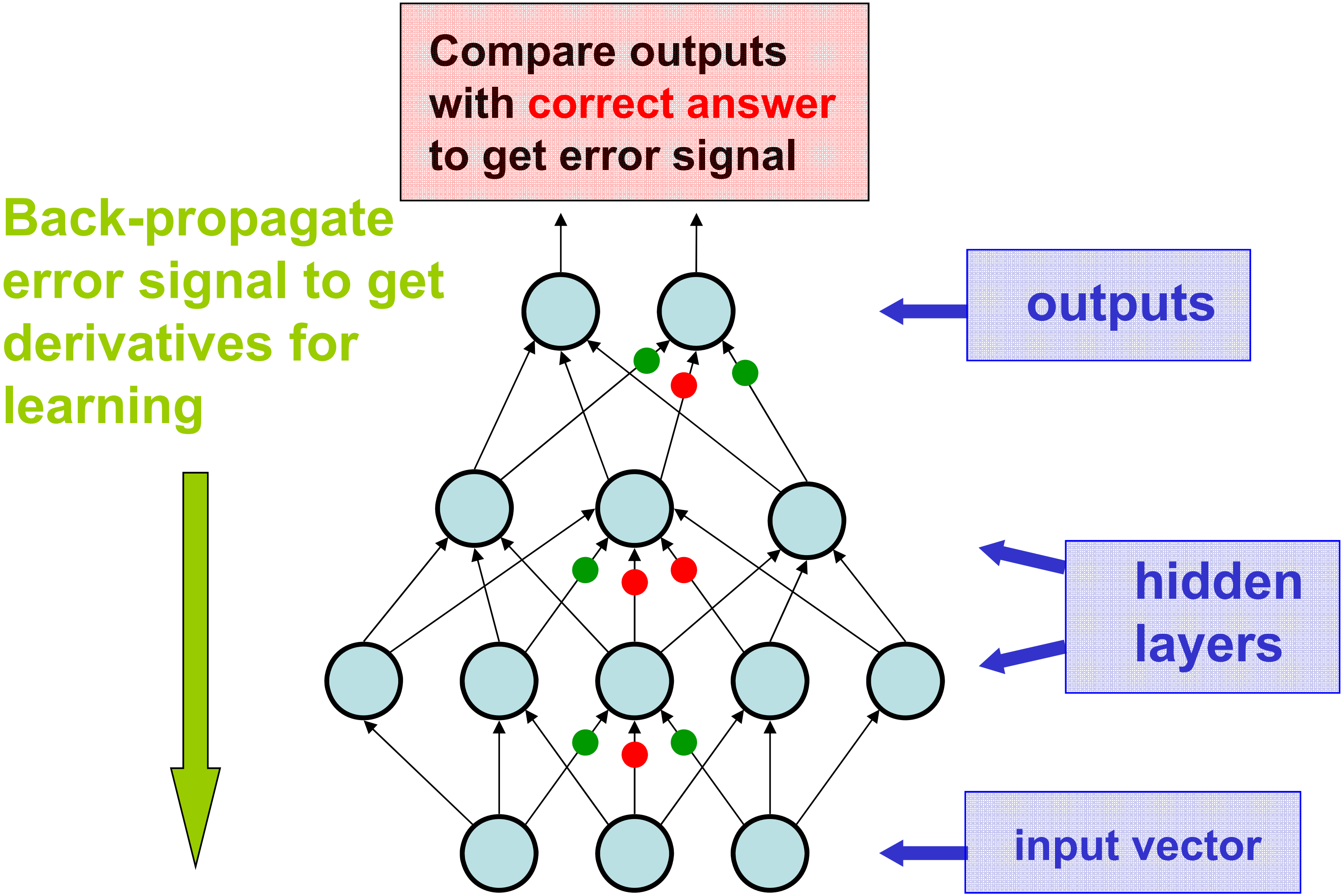
Week 6: Optimization, Regularization, and Applications of Deep Learning

Iasonas Kokkinos

i.kokkinos@cs.ucl.ac.uk

University College London

Multi-Layer Perceptrons (~1985)



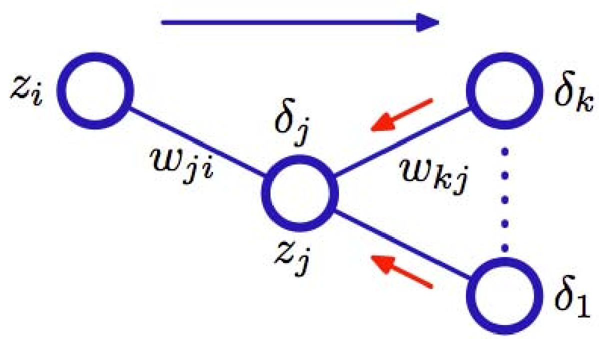
Slide credits: G. Hinton

Back-propagation in a nutshell

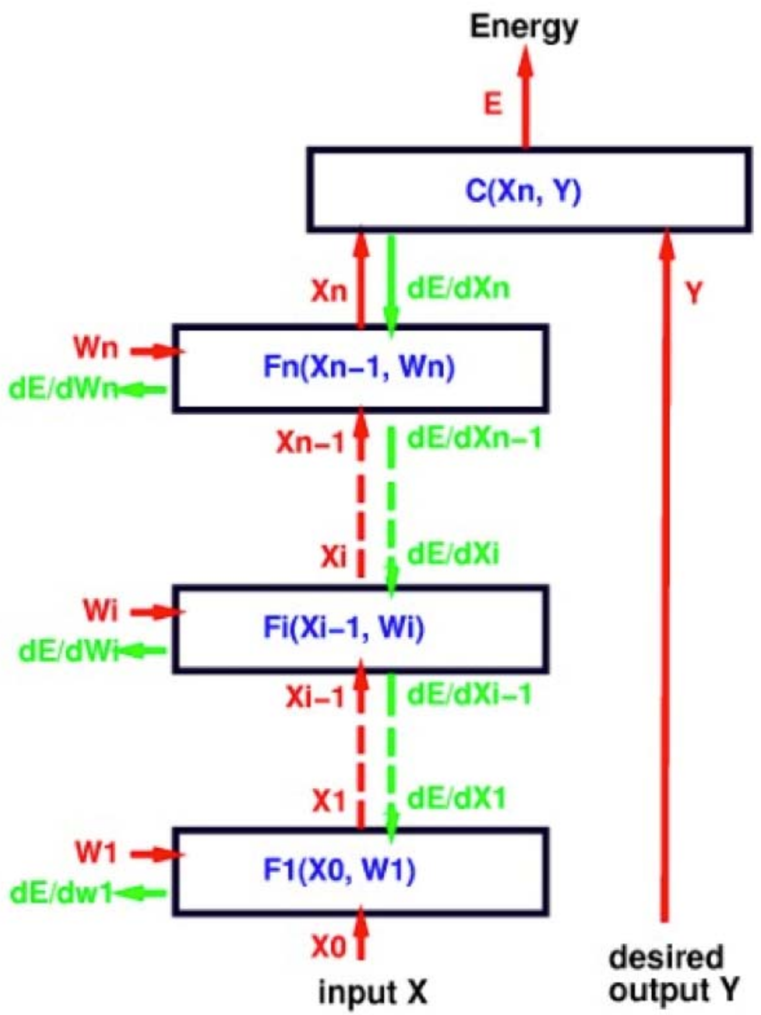
Error messages, ‘δ’, at current layer:
sensitivity of loss to activations of current layer

Every node estimates its error message by forming a weighted sum of the error messages of its recipients

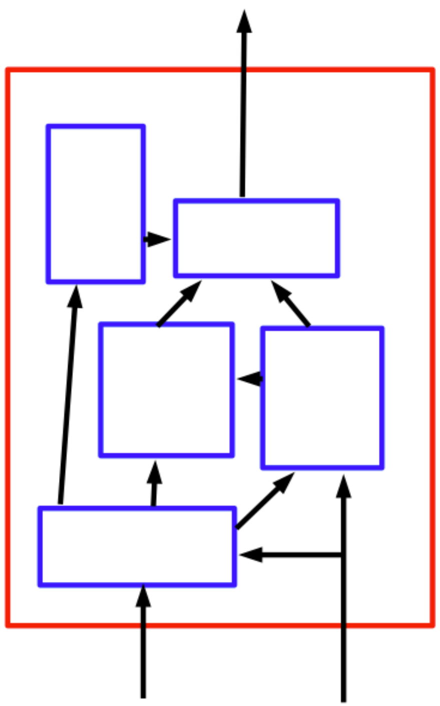
Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ’s from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



Forward/backward information flow



We can use the same algorithm to learn in any Directed Acyclic Graph!



Deep Learning: breakthrough for all of AI

VISION

$$x \rightarrow \boxed{max(0, W^1 \boldsymbol{x})} \xrightarrow{h^1} \boxed{max(0, W^2 \boldsymbol{h}^1)} \xrightarrow{h^2} \boxed{W^3 \boldsymbol{h}^2} \xrightarrow{o}$$

SPEECH

$$x \rightarrow \boxed{max(0, W^1 \boldsymbol{x})} \xrightarrow{h^1} \boxed{max(0, W^2 \boldsymbol{h}^1)} \xrightarrow{h^2} \boxed{W^3 \boldsymbol{h}^2} \xrightarrow{o}$$

NLP

This burrito place
is yummy and fun!

$$x \rightarrow \boxed{max(0, W^1 \boldsymbol{x})} \xrightarrow{h^1} \boxed{max(0, W^2 \boldsymbol{h}^1)} \xrightarrow{h^2} \boxed{W^3 \boldsymbol{h}^2} \xrightarrow{o}$$

Next 10 slides: Week 1 reminder

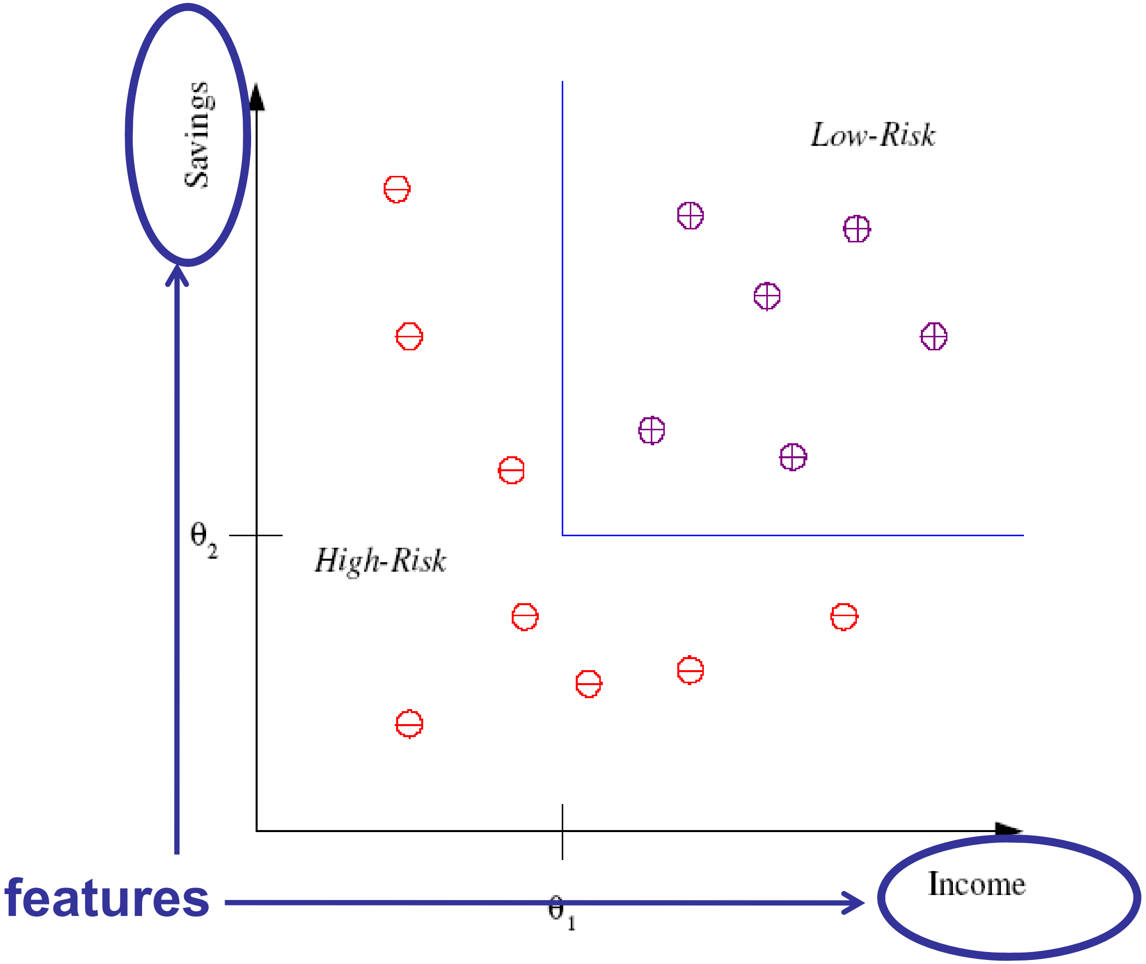
Our goal in this course: learn an input-output mapping

$$y = f_w(x) \ \ (= f(x, w))$$

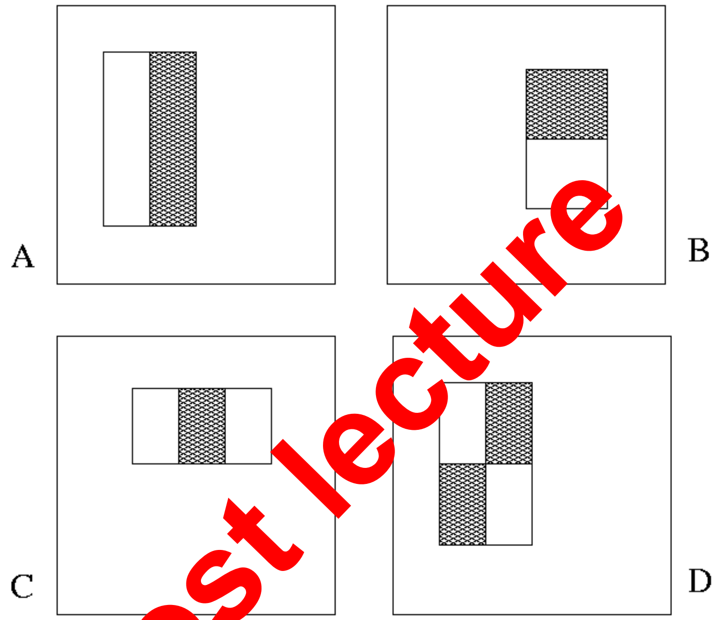
- Output: y
- Input: x
- Method: f
- Parameters: w

How to construct this function? $y = f_w(x)$

- Step 1: Determine its inputs, x



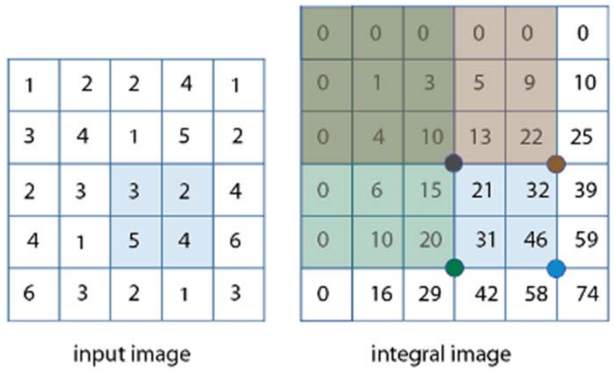
Feature example: Haar wavelets (NOT part of our course)



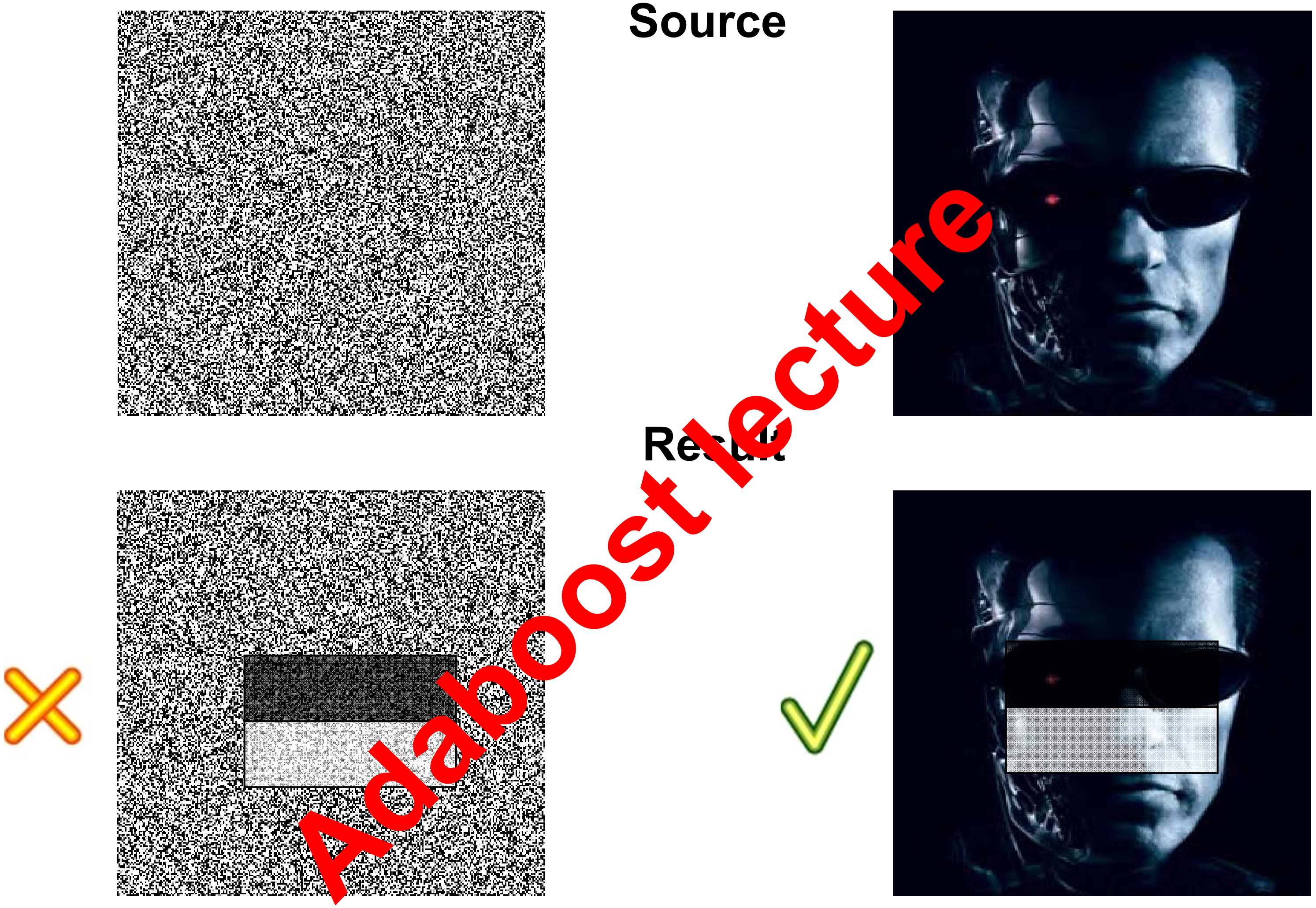
Value = ∑ (pixels in white area) – ∑ (pixels in black area)

Adaboost lecture

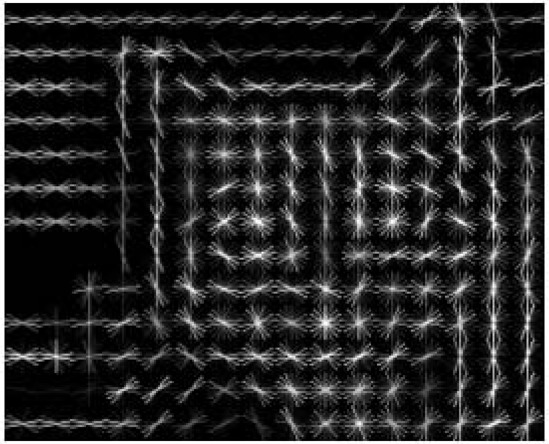
Why these features?
Extremely fast to compute
(4 pixel operations per box)



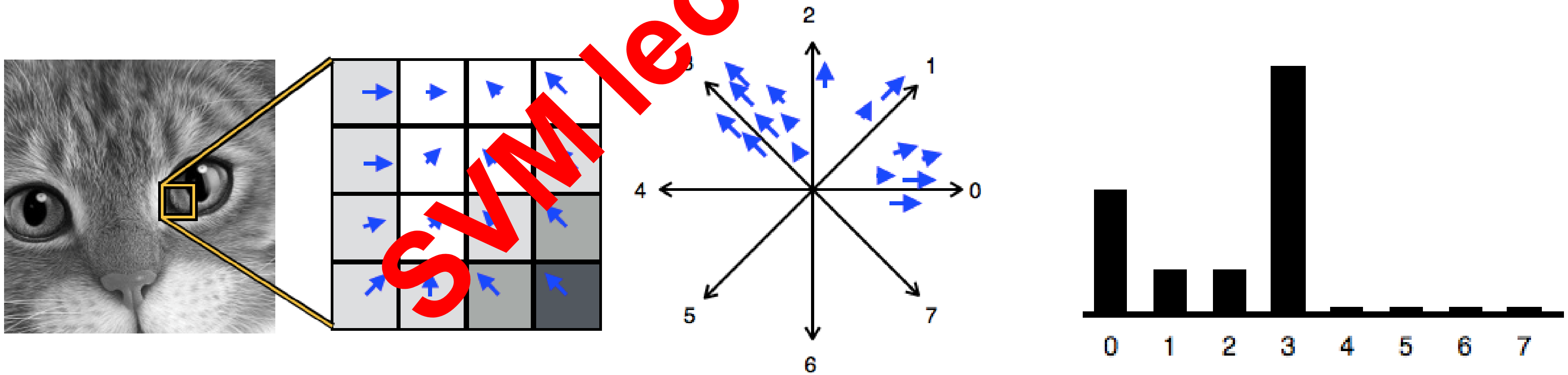
One Haar wavelet feature



Feature example: Histogram-of-gradient features (NOT part of our course)



SVM lecture



Feature example: Bag-of-word features (NOT part of our course)

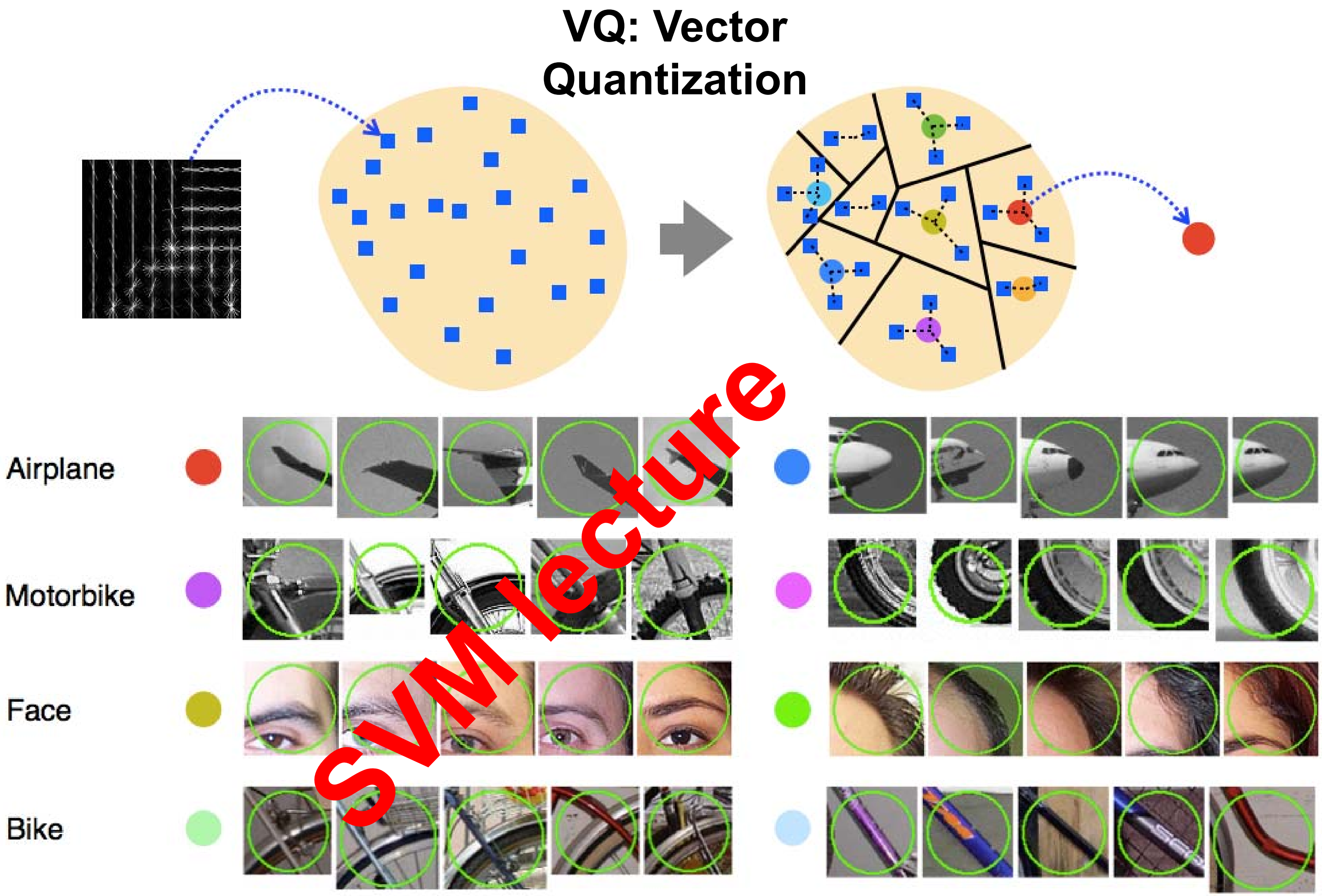
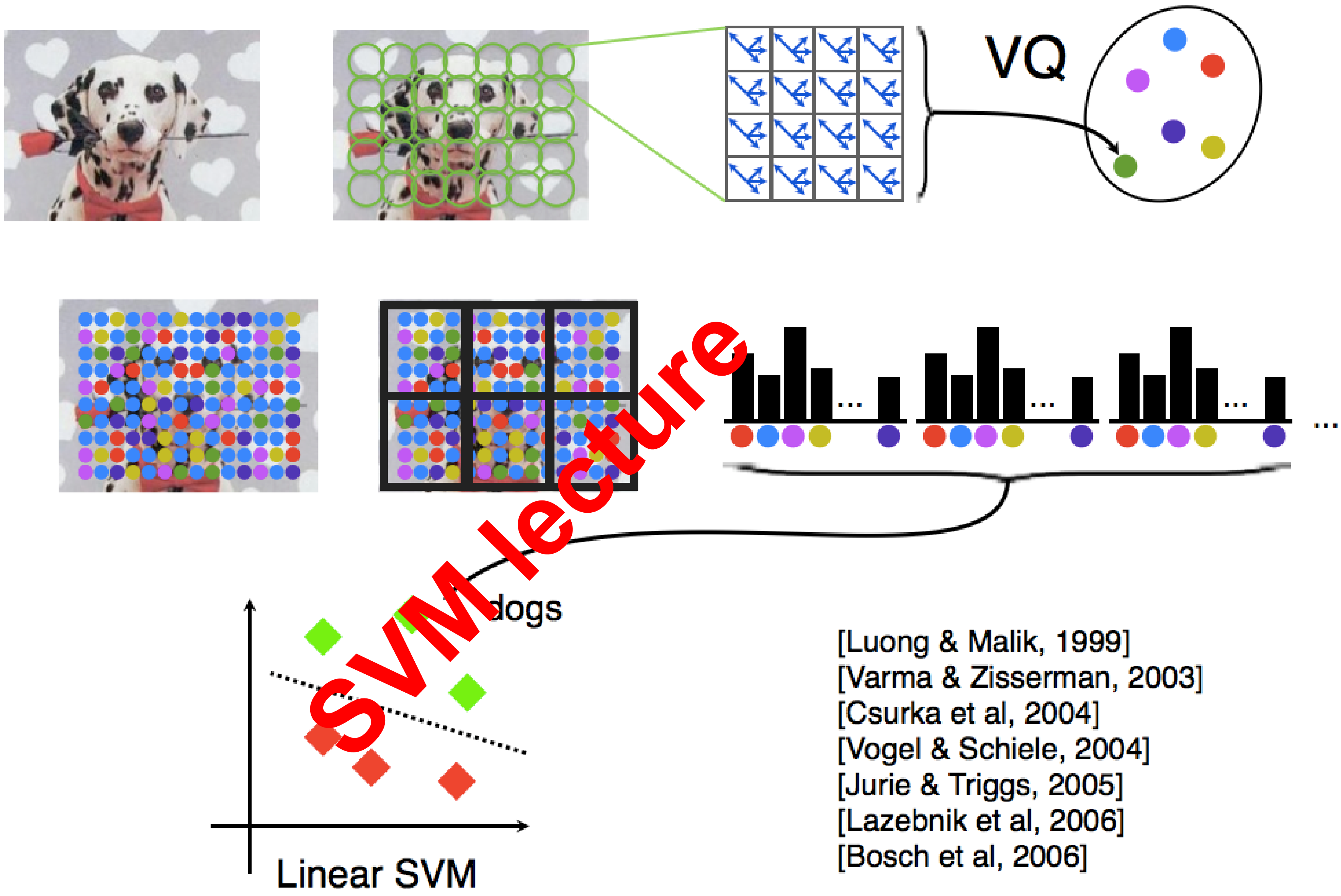
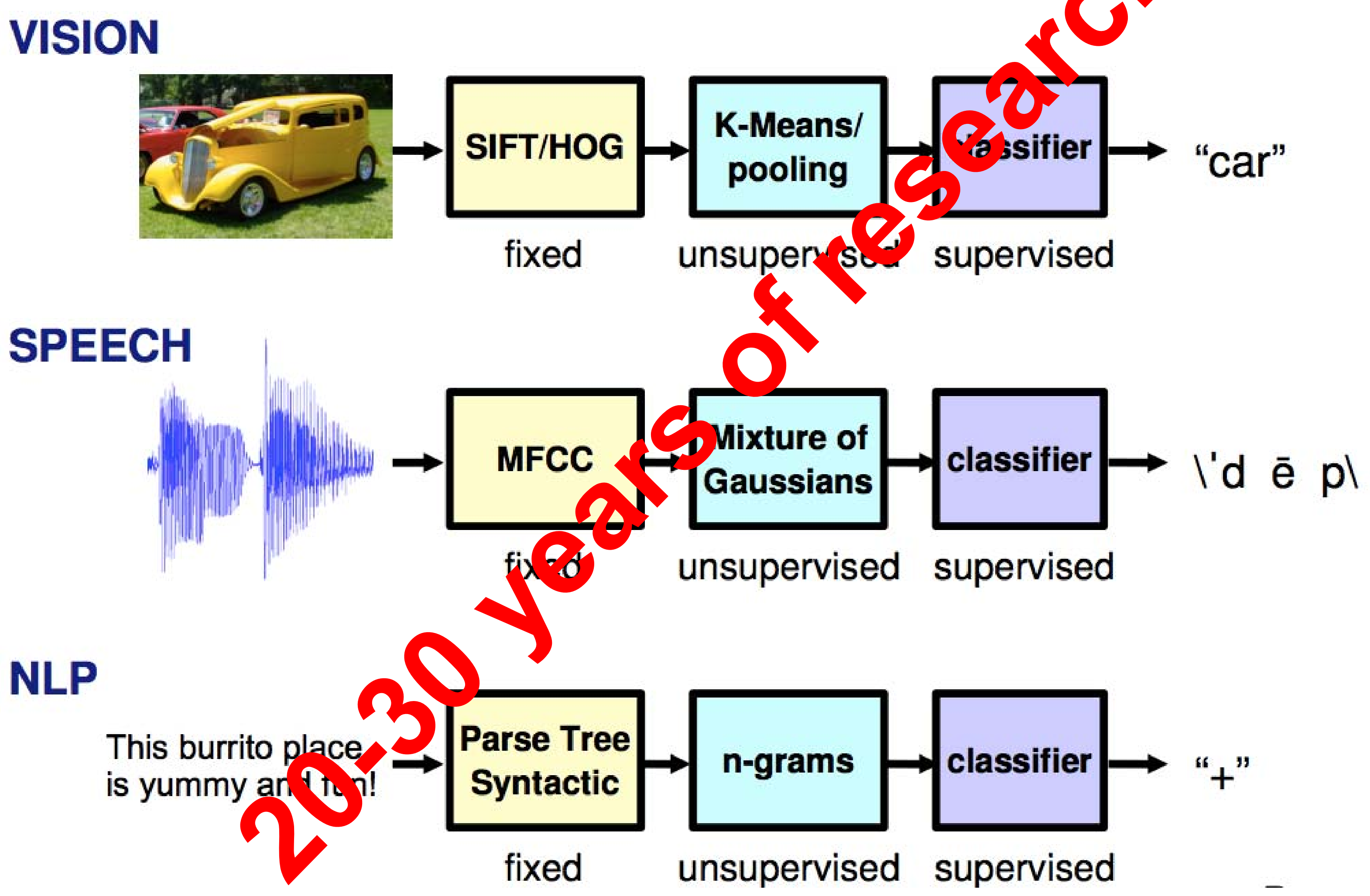


Image classification in a nutshell (NOT part of our course)

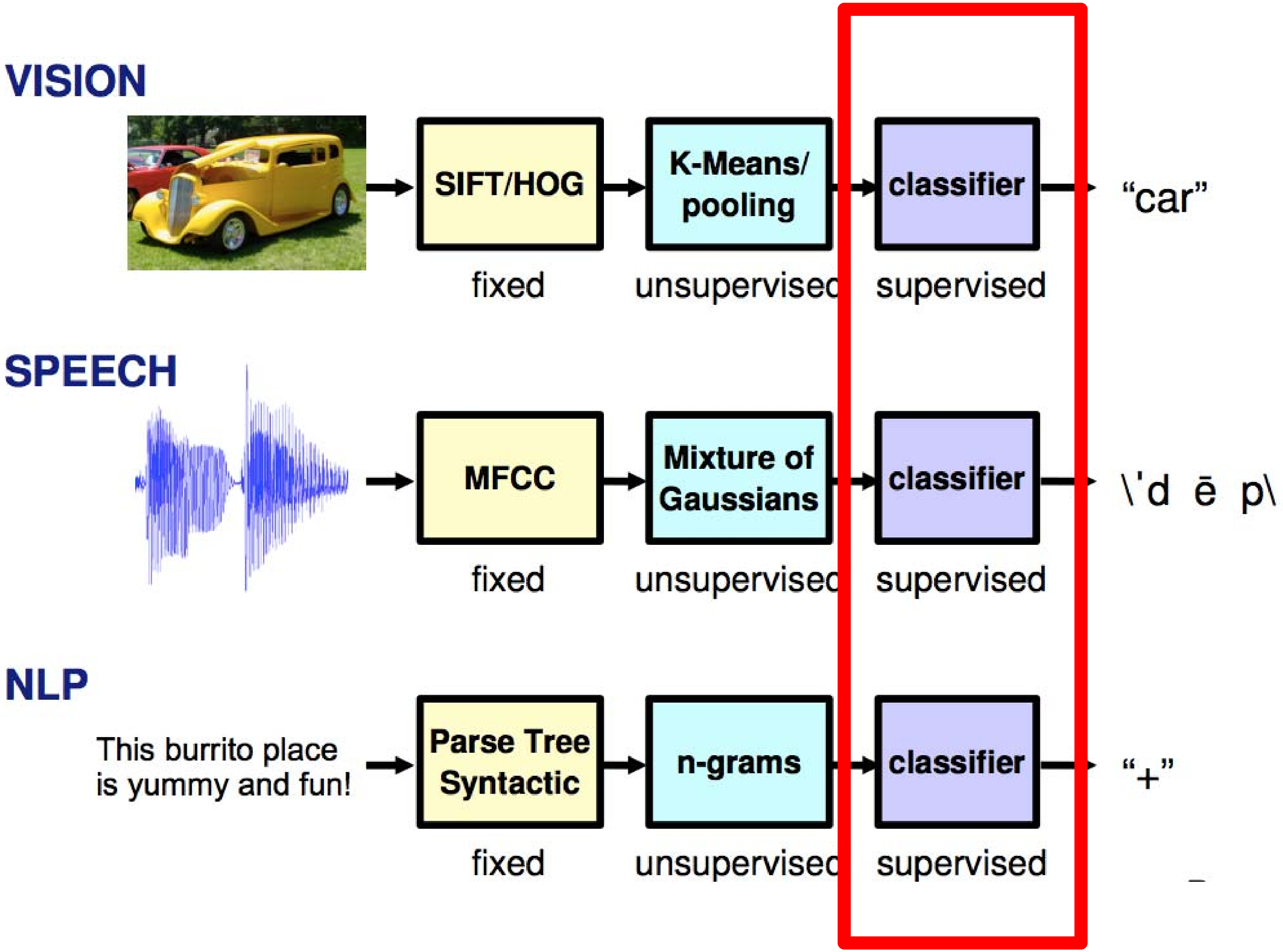


[Luong & Malik, 1999]
[Varma & Zisserman, 2003]
[Csurka et al, 2004]
[Vogel & Schiele, 2004]
[Jurie & Triggs, 2005]
[Lazebnik et al, 2006]
[Bosch et al, 2006]

Machine Learning for X: features for X + ML

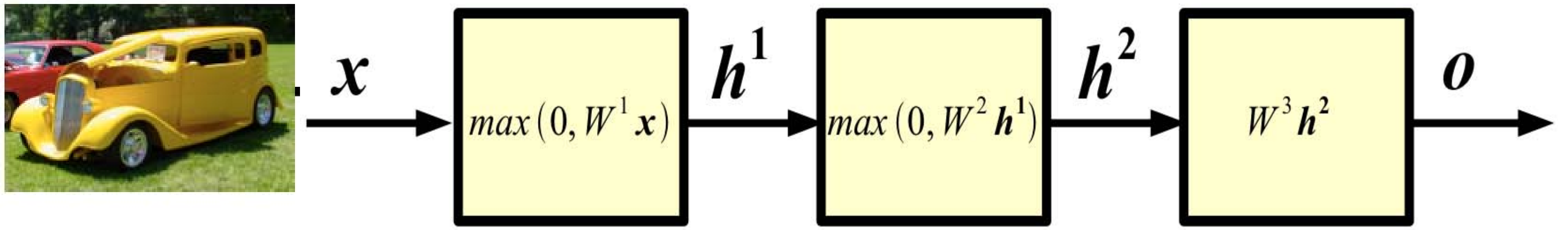


Our course

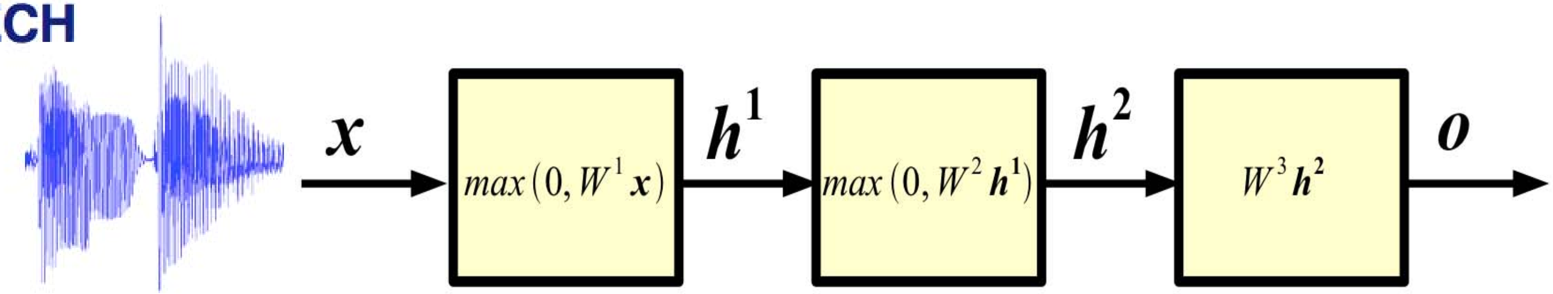

VISION
SIFT/HOG
K-Means/ pooling
classifier
"car"
fixed
unsupervised
supervised
SPEECH
MFCC
Mixture of Gaussians
classifier
\'d ē p\
fixed
unsupervised
supervised
NLP
This burrito place is yummy and fun!
Parse Tree Syntactic
n-grams
classifier
"+"
fixed
unsupervised
supervised

Deep Learning: learn the features!

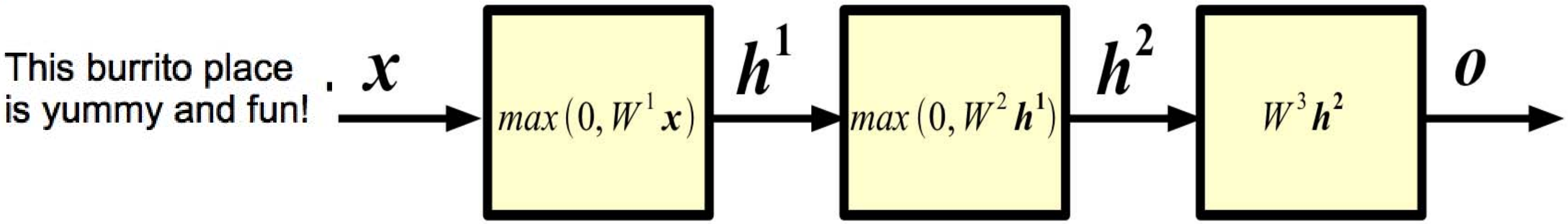
VISION



SPEECH

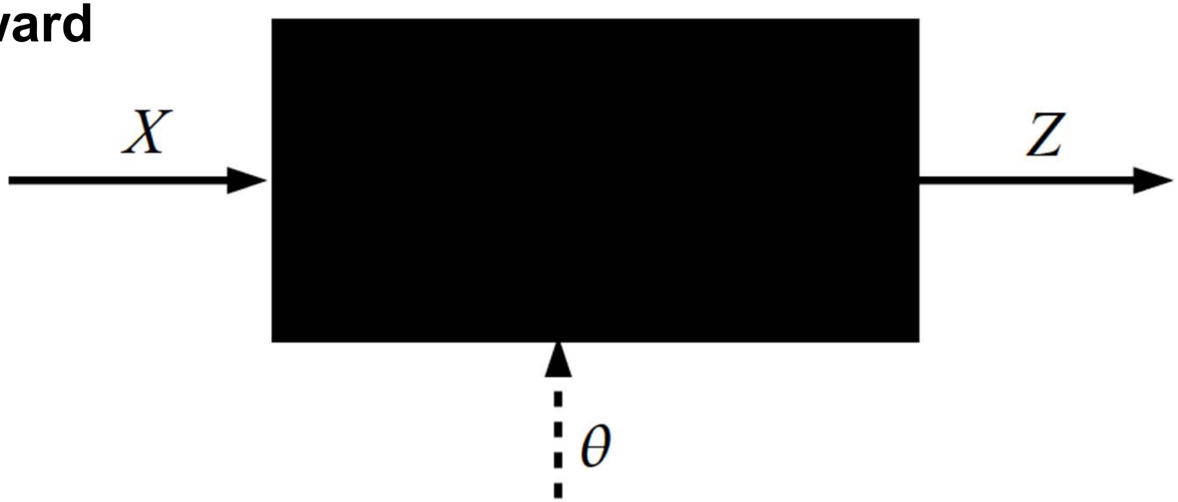


NLP

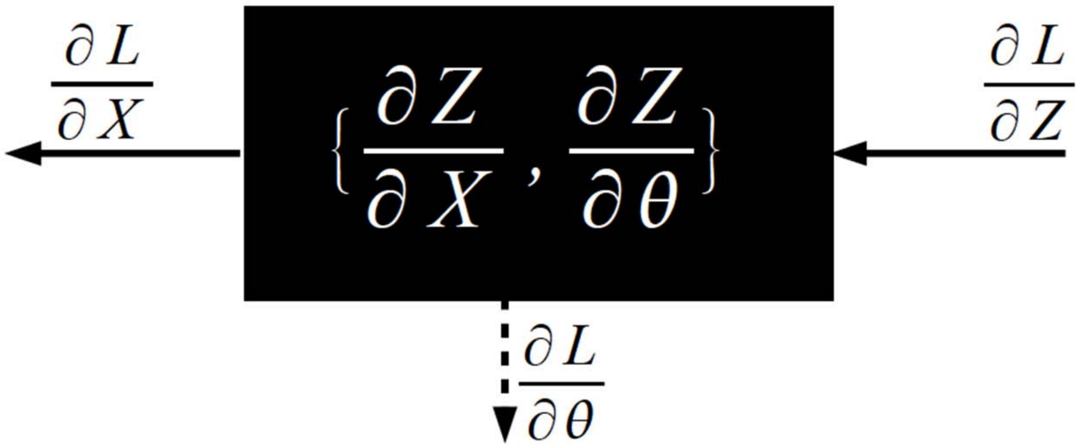


All you need is gradients

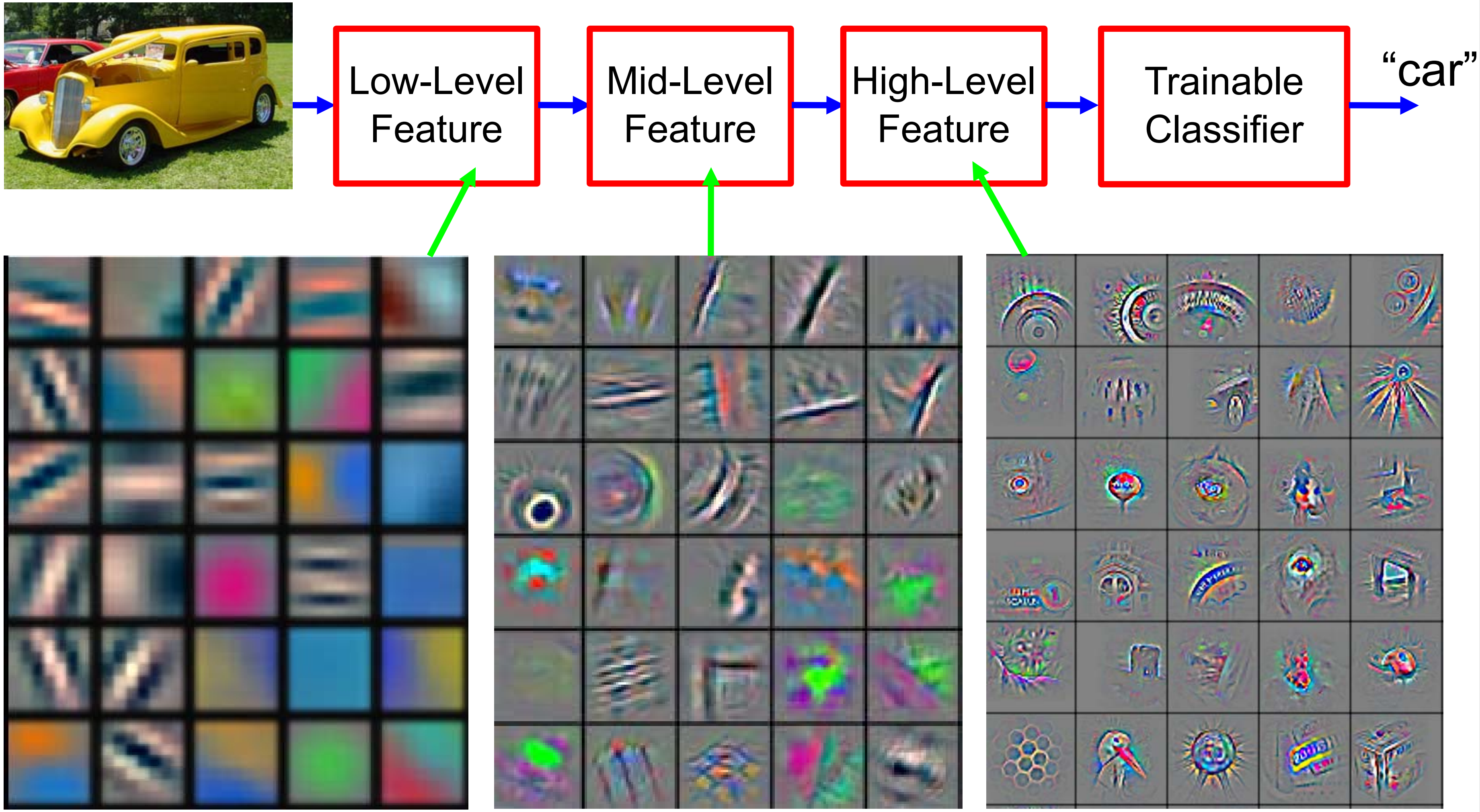
Forward



Backward

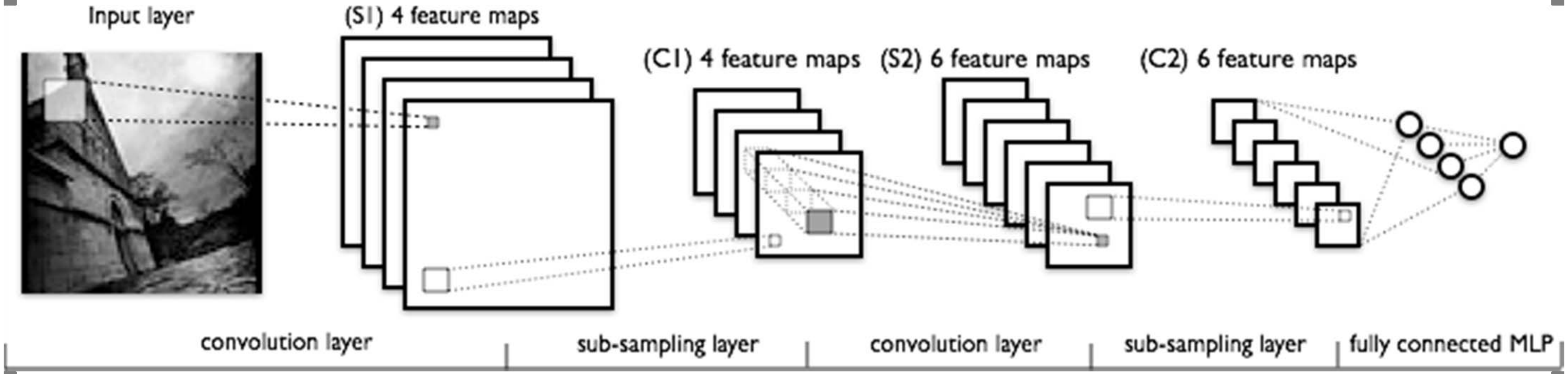


Deep Learning = Hierarchical Compositionality



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

Convolutional Nets



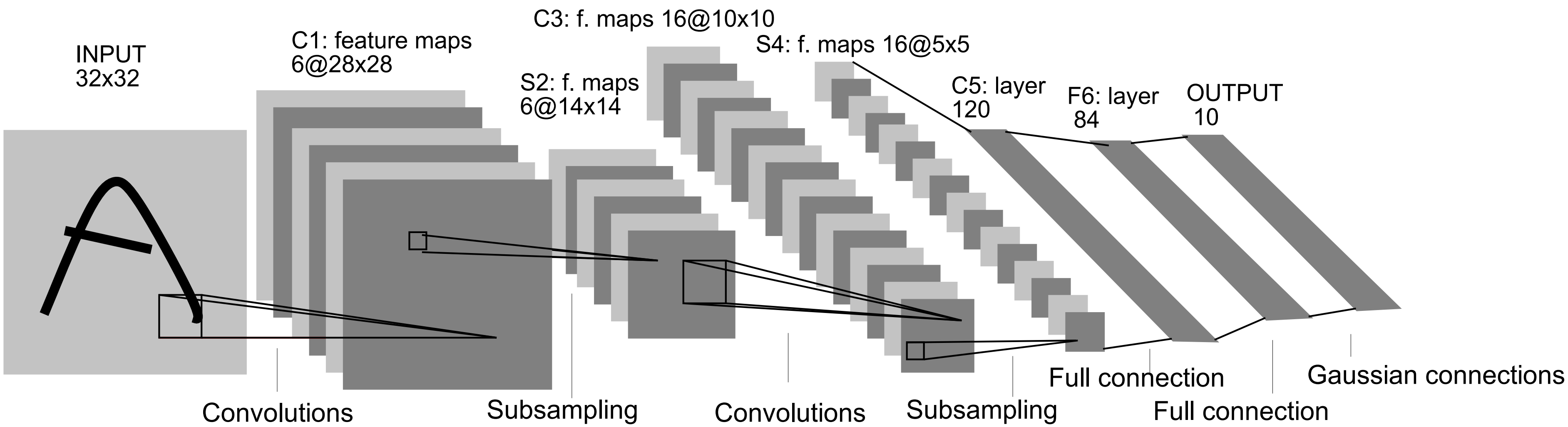
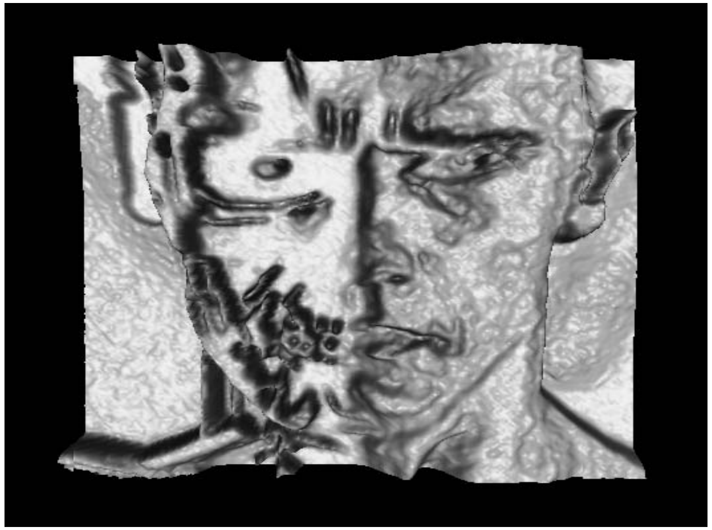
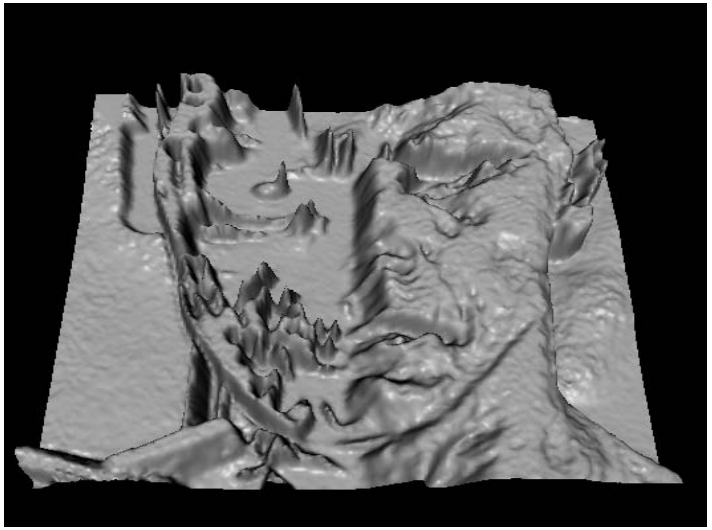
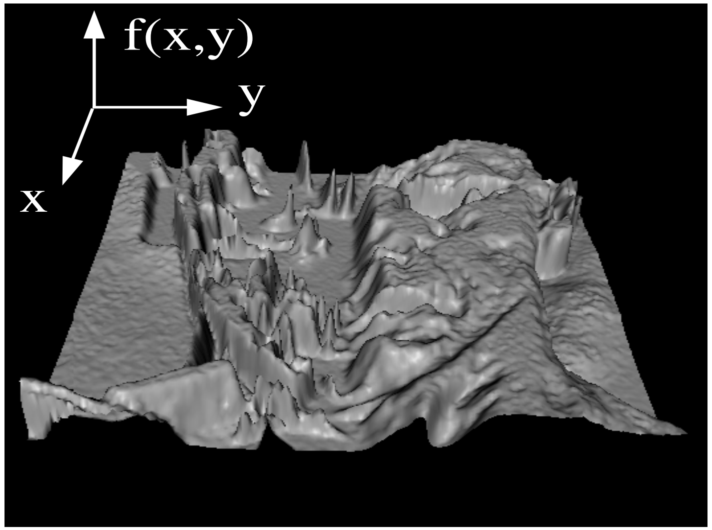


Image Credit: Yann LeCun, Kevin Murphy

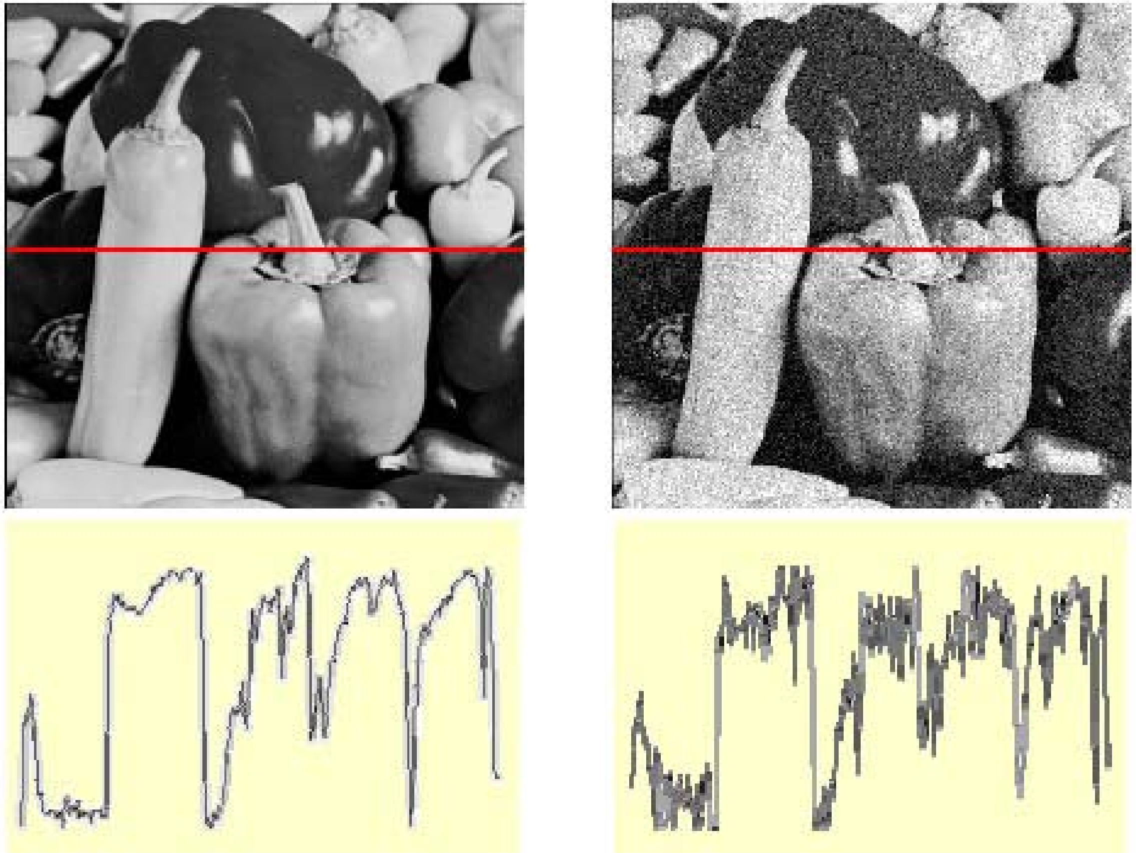
Taking a closer look: Images as functions







Gaussian Noise



$$f(x,y) = \overbrace{\hat{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

Averaging = Filtering

$$F[x,y] \otimes H[u,v] = G[x,y]$$

$F[x,y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$H[u,v]$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ ? & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

"box filter"

$G[x,y]$

	0	10	20	30	30				

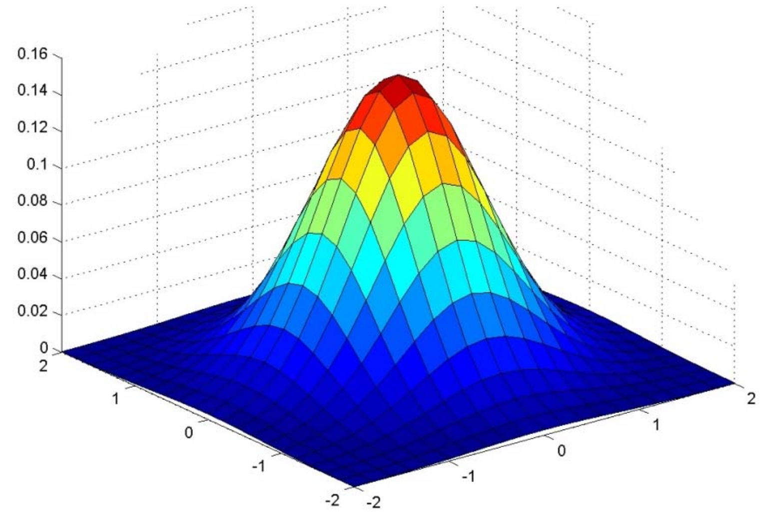
$$G = H \otimes F$$

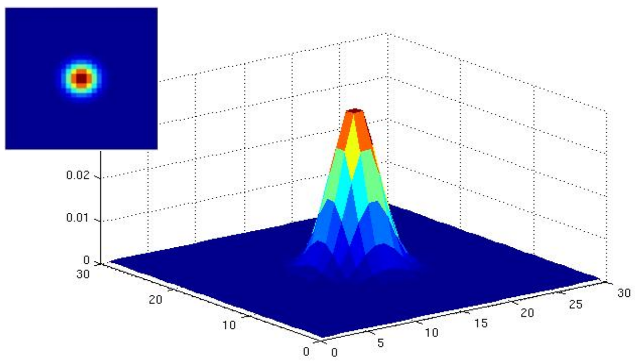
Gaussian Smoothing

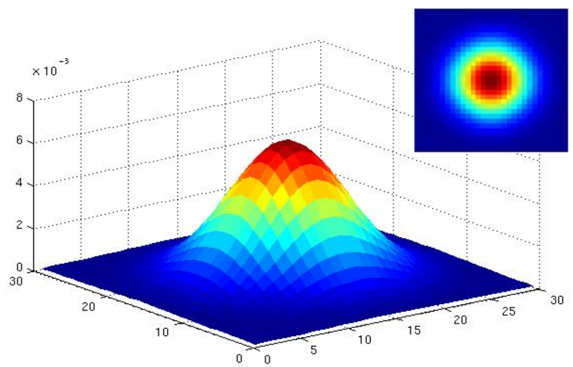
- Gaussian kernel

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Weighs nearby pixels more than distant ones







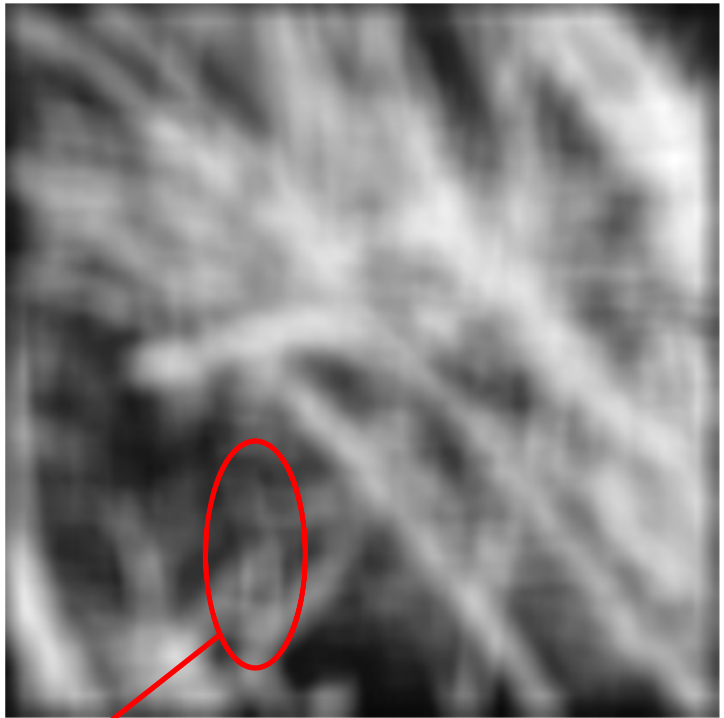
$\sigma = 2$ **with** 30×30 **kernel**

$\sigma = 5$ **with** 30×30 **kernel**

Smoothing by Averaging

depicts box filter:
white = high value, black = low value

Original



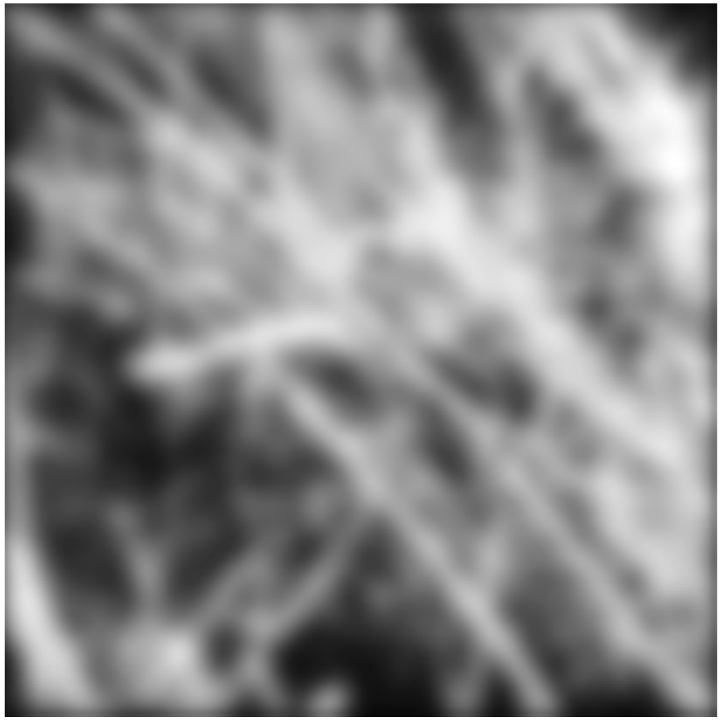
Filtered

“Ringing” artifacts!

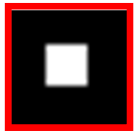
Image Source: Forsyth & Ponc

Smoothing by Gaussian filtering

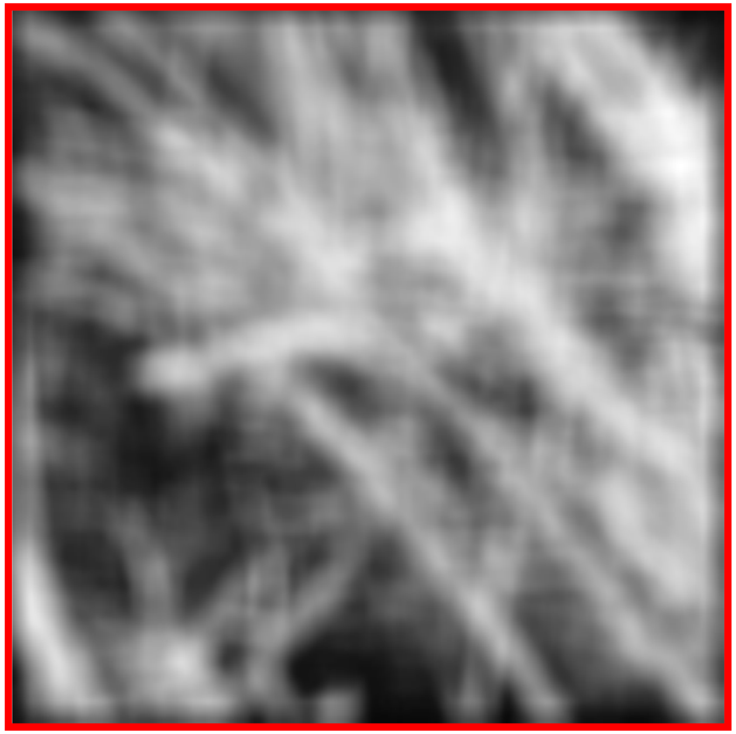
Original



Filtered



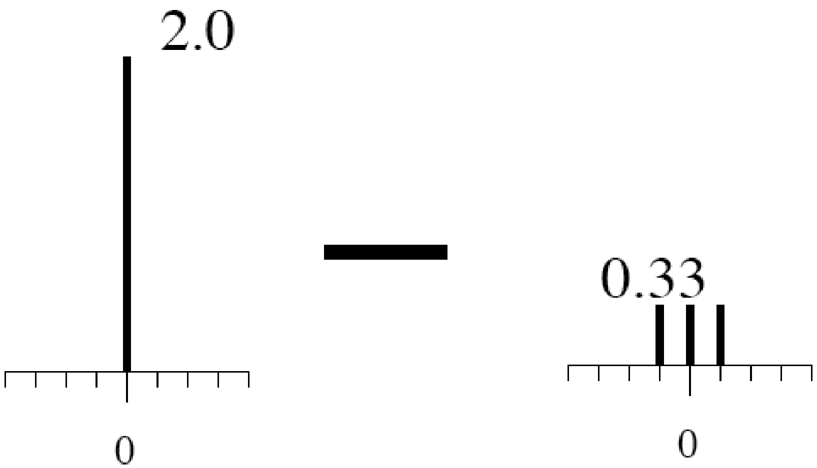
Original

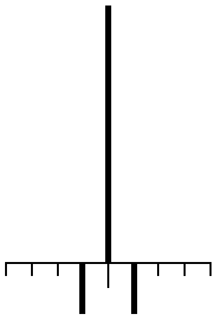


Filtered

Sharpening Filter

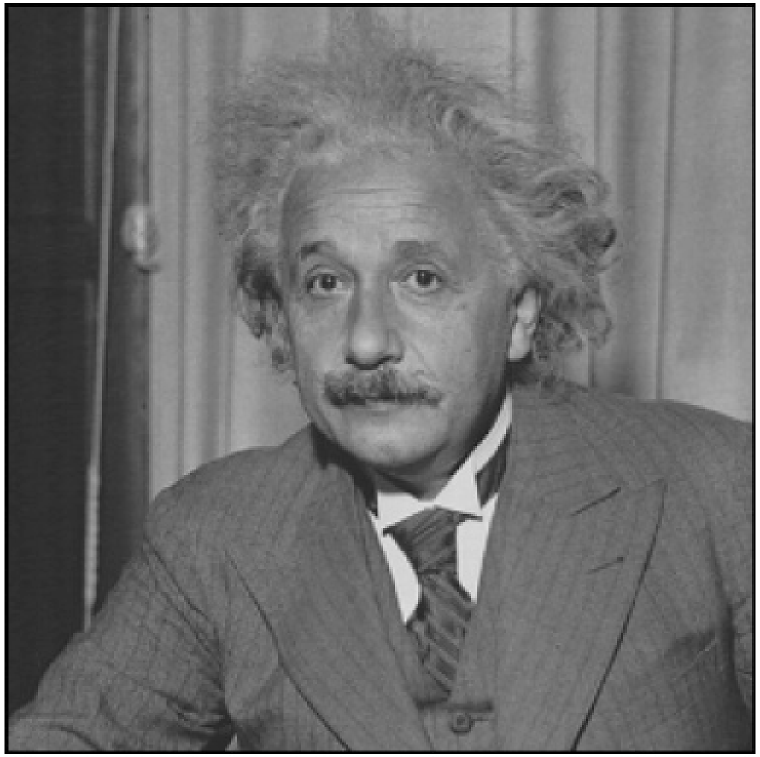
Original

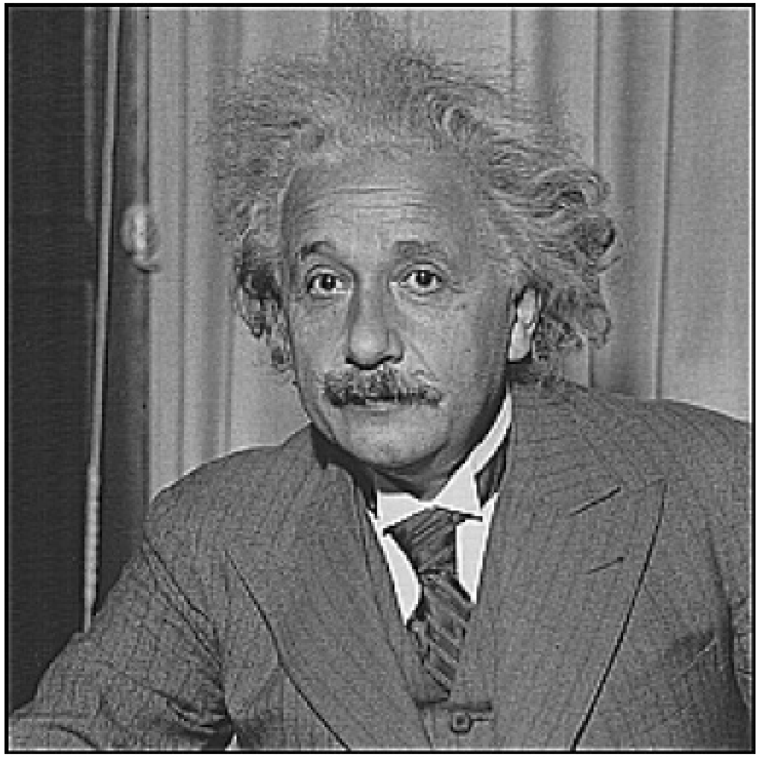




Sharpening filter

Sharpening Filter

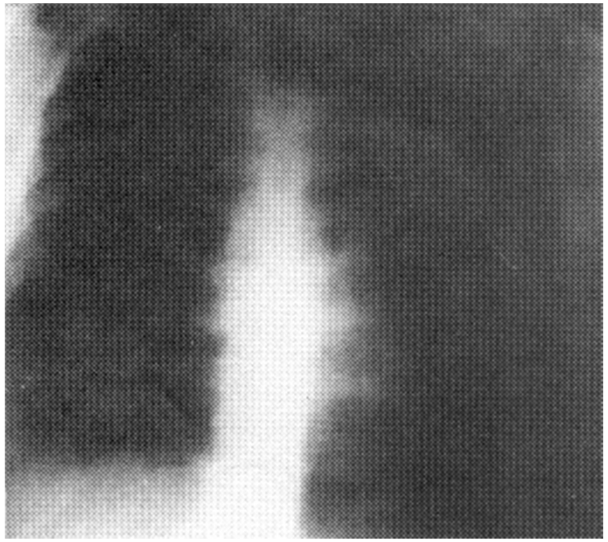


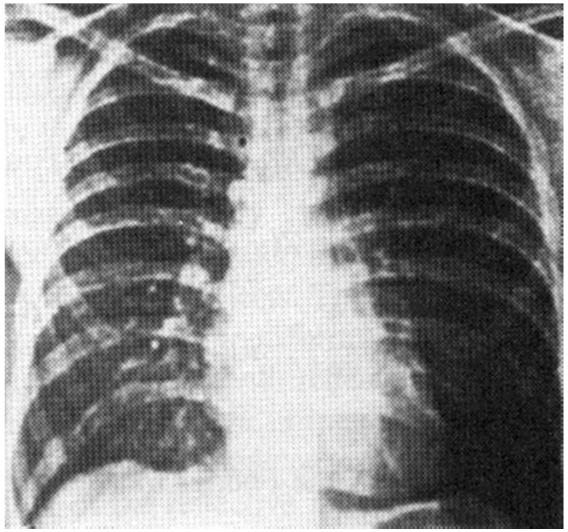


before after

Image processing application

Original





High Frequency Emphasis
+
Histogram Equalization

Filters, filters, filters

Decades of research in image processing

Demos:

http://setosa.io/ev/image-kernels/

Methods:

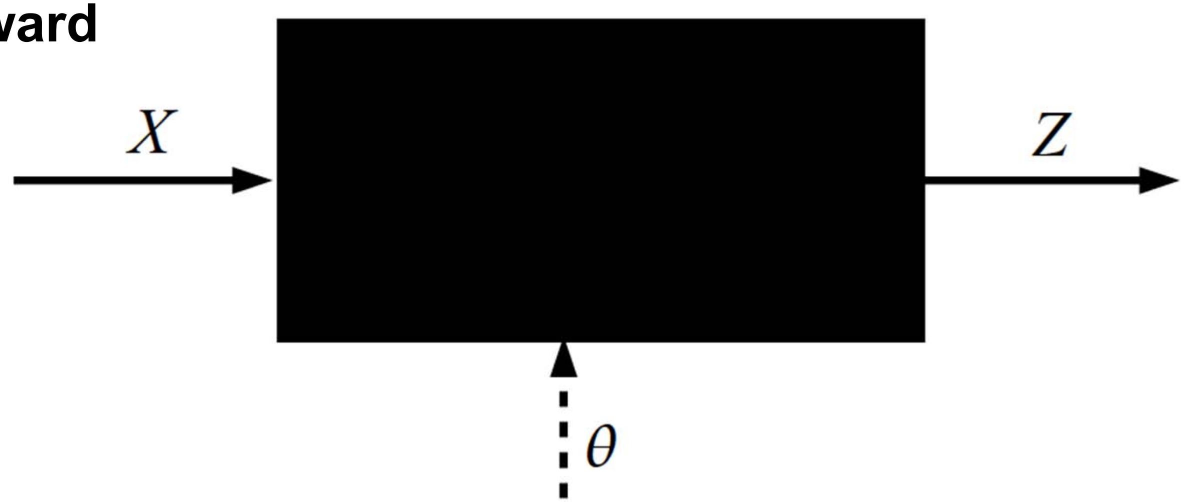
http://www.ipol.im/

Open-ended: how does it connect with a given application?

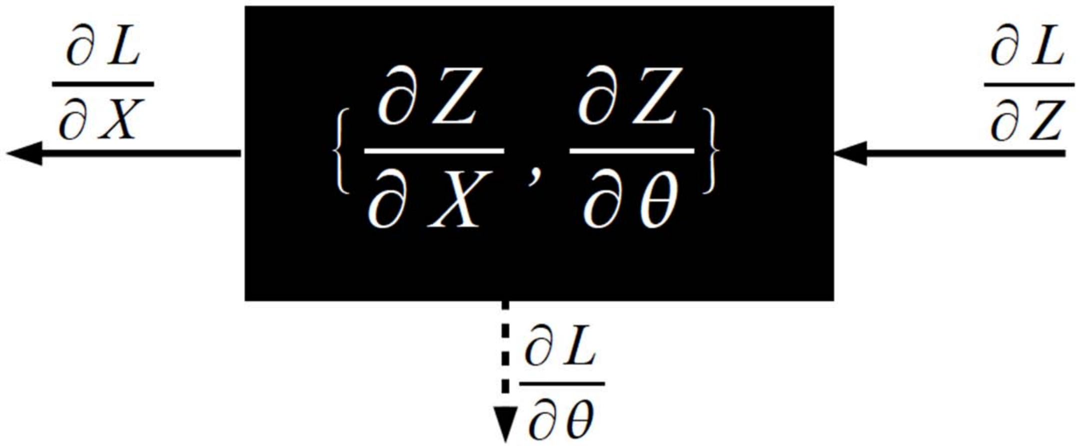
Question: can we learn how to do this?
(or any other type of image processing)

Answer: all you need is gradients

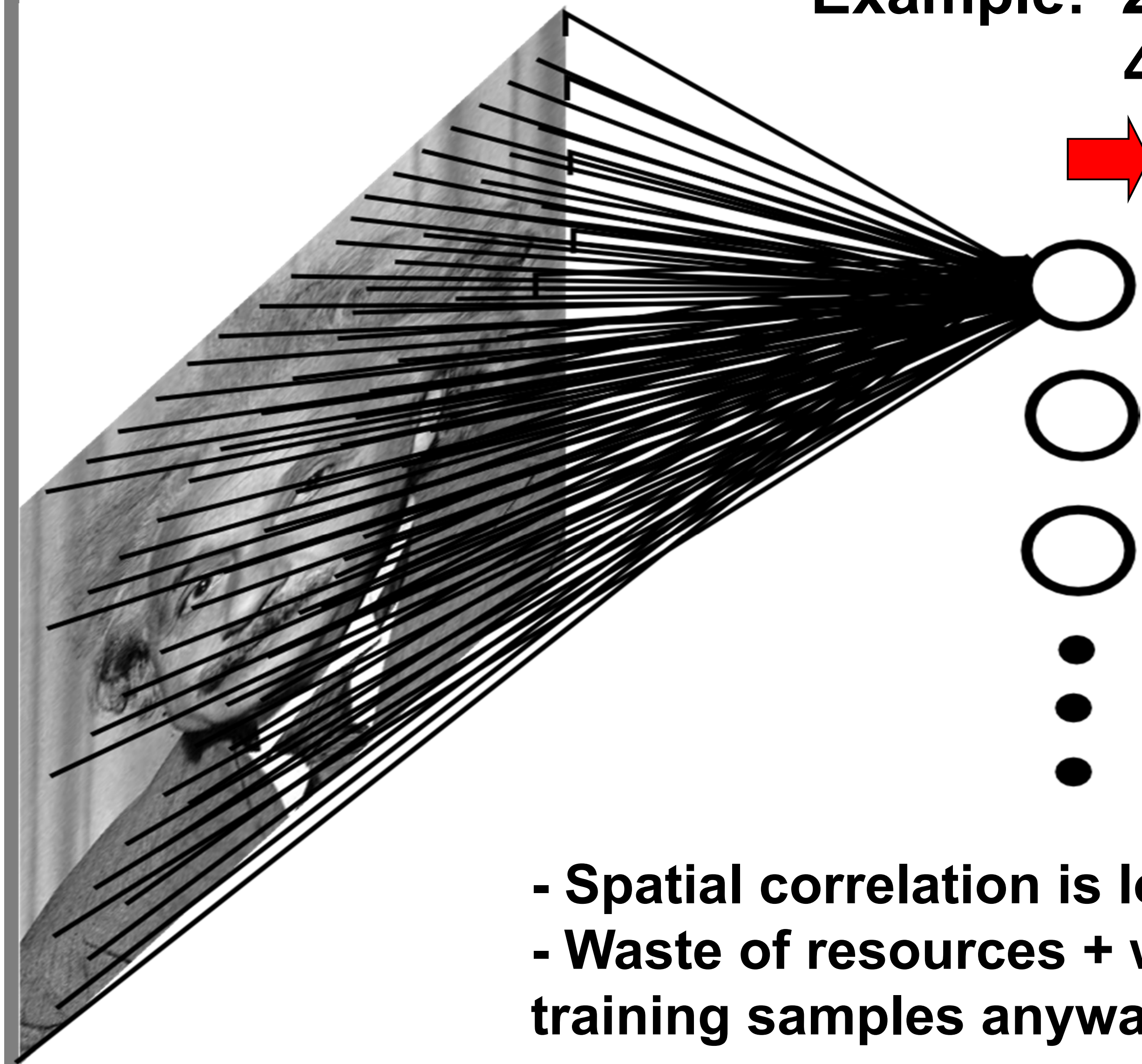
Forward



Backward



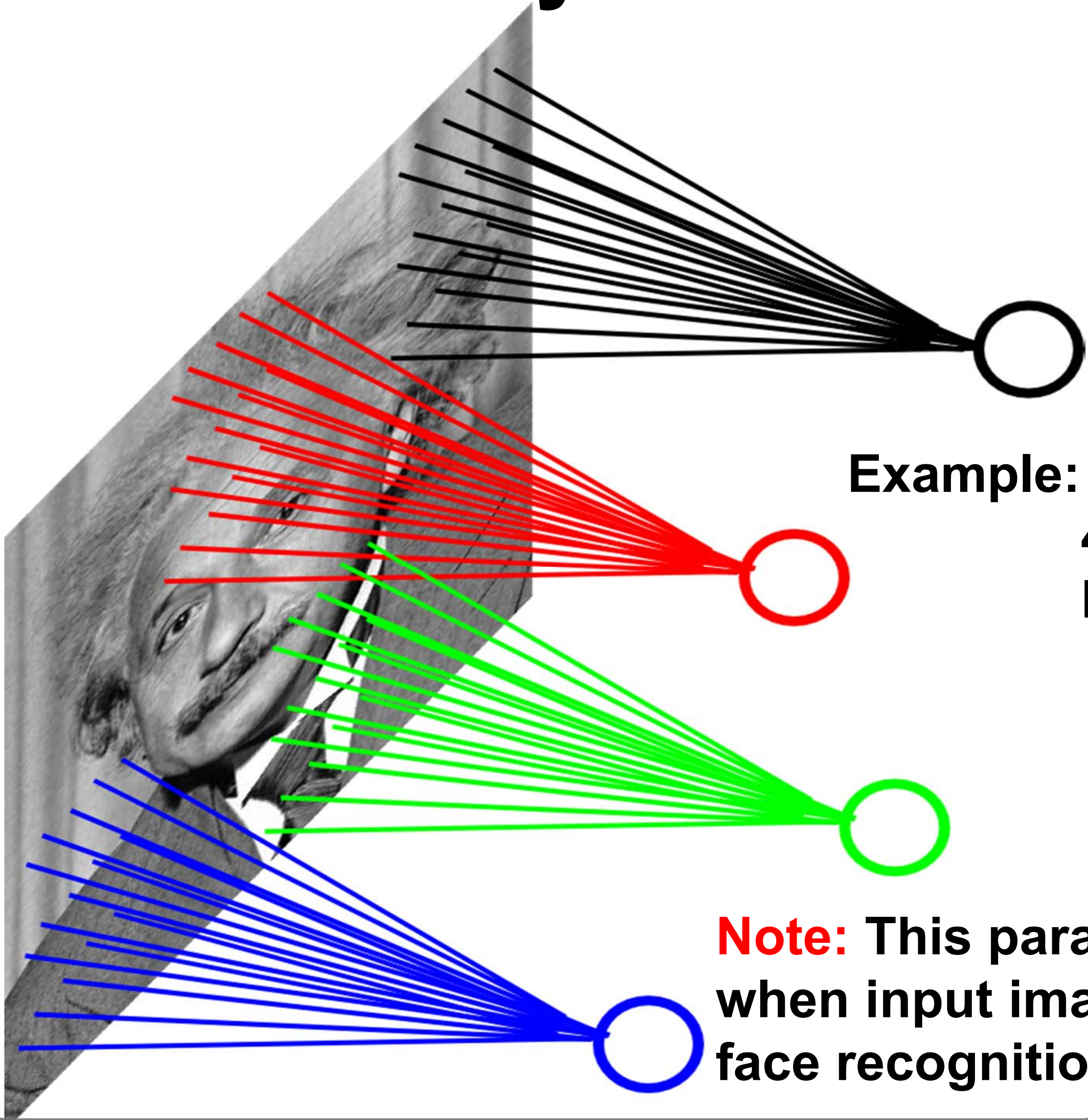
Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

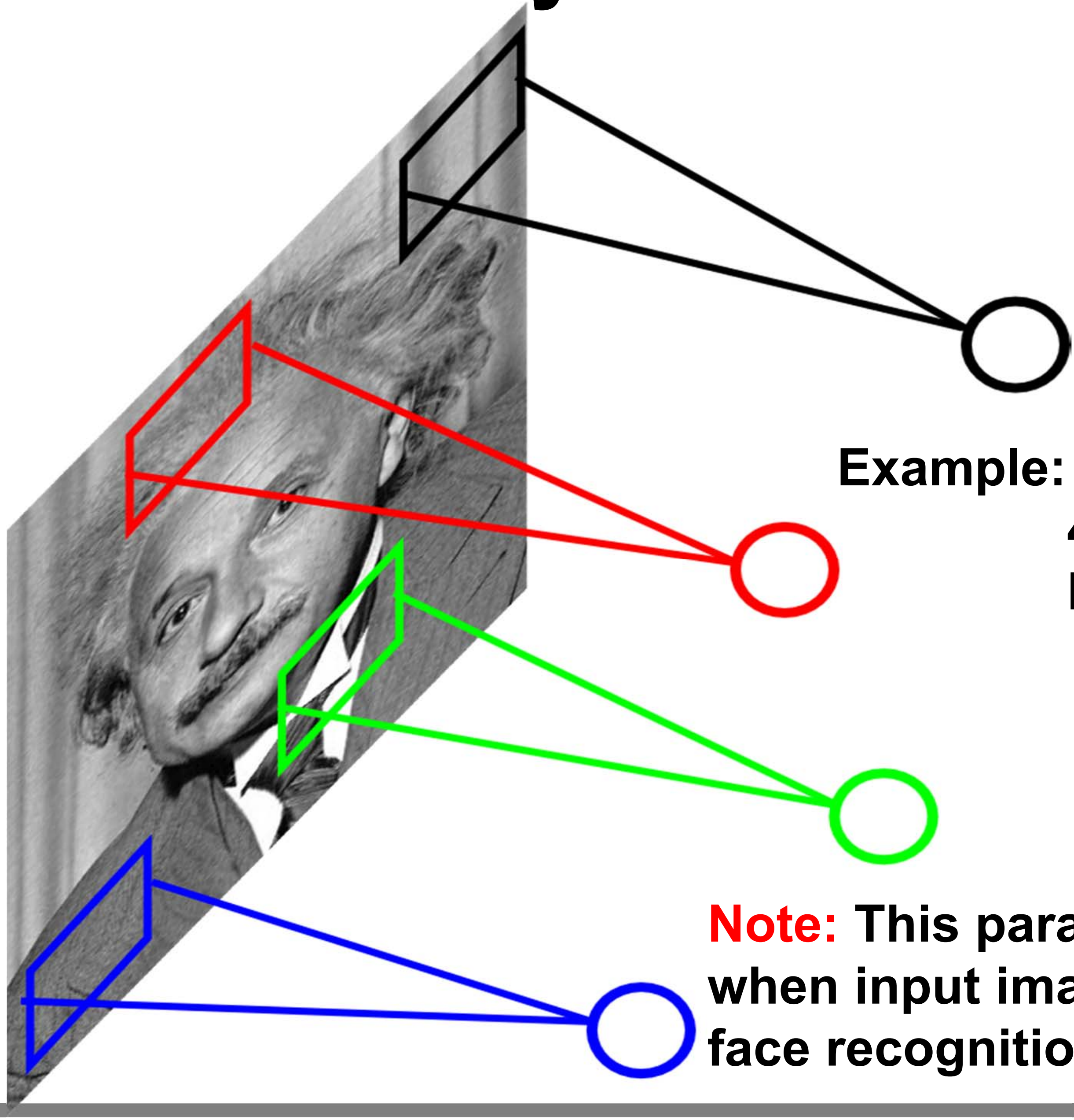
Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

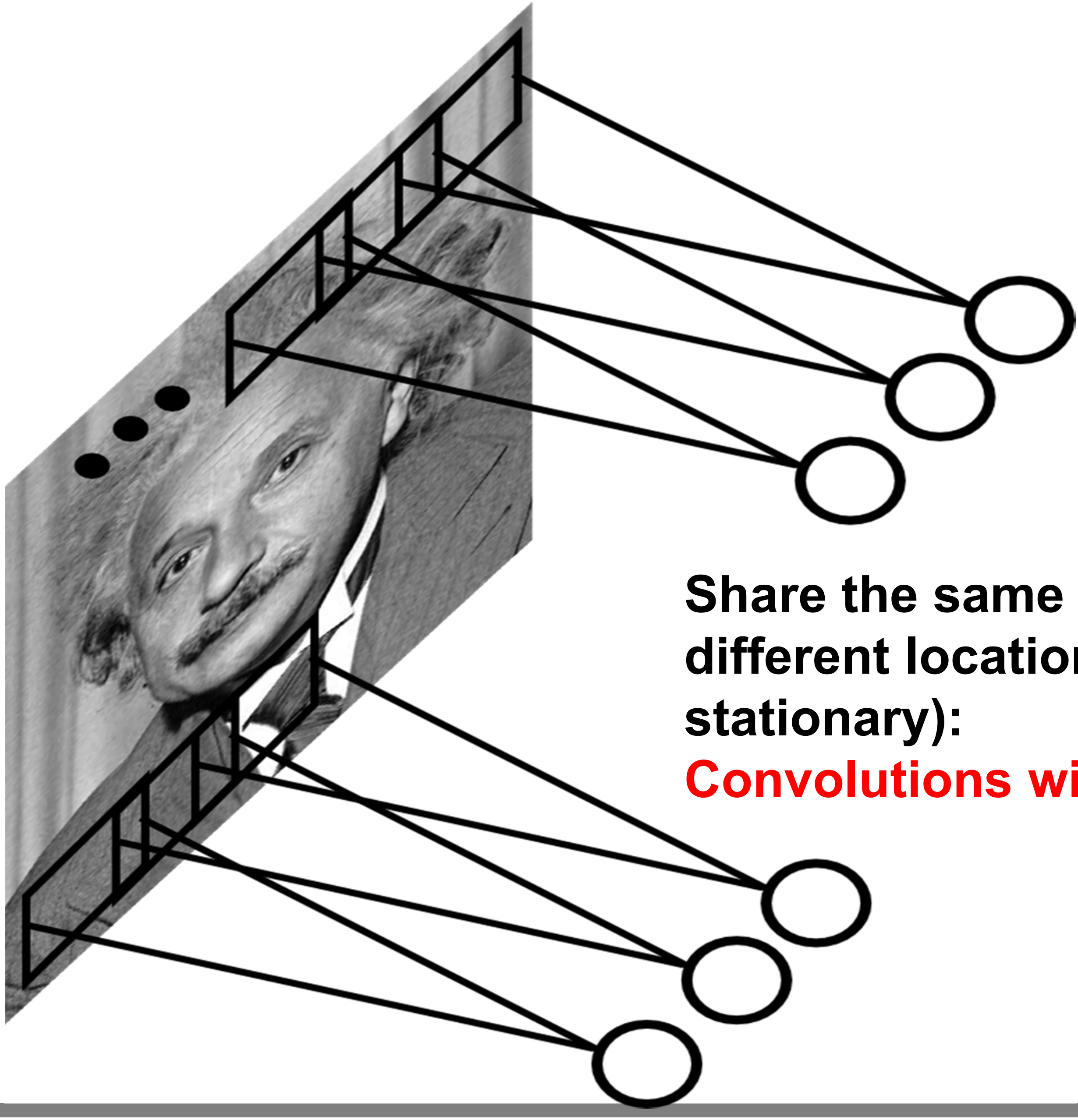
Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

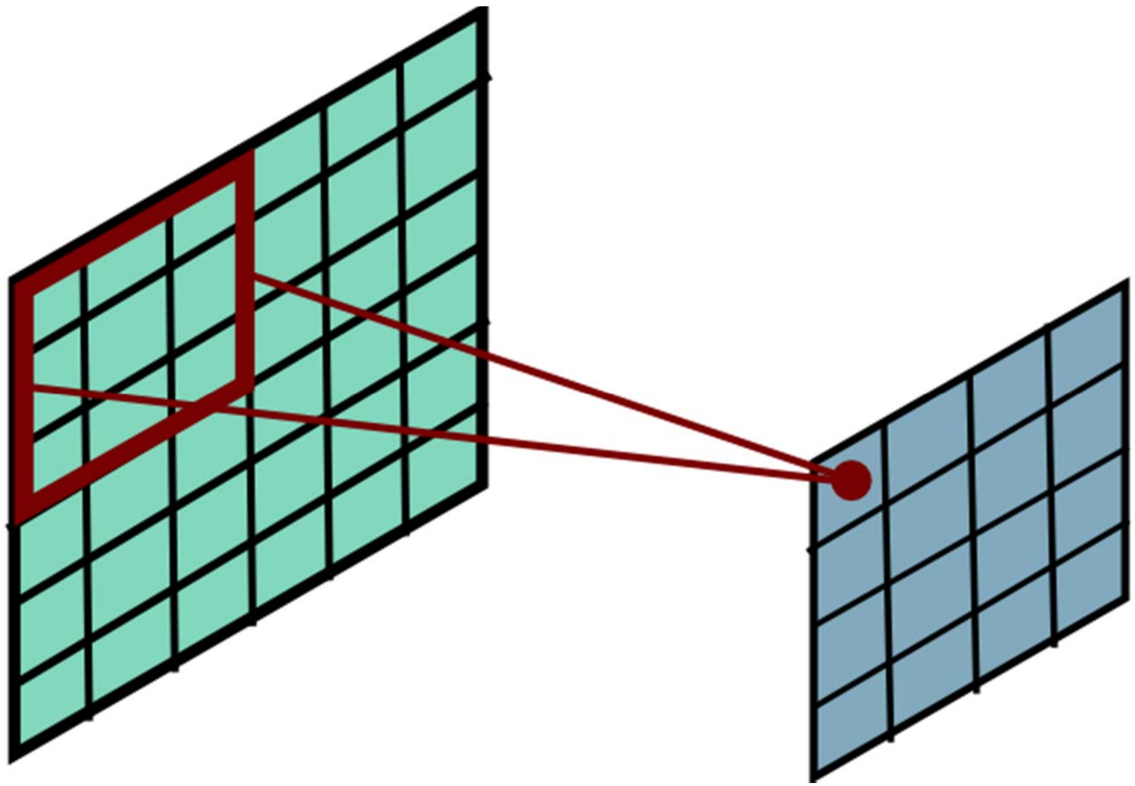
Note: This parameterization is good when input image is registered (e.g., face recognition).

Convolutional Layer

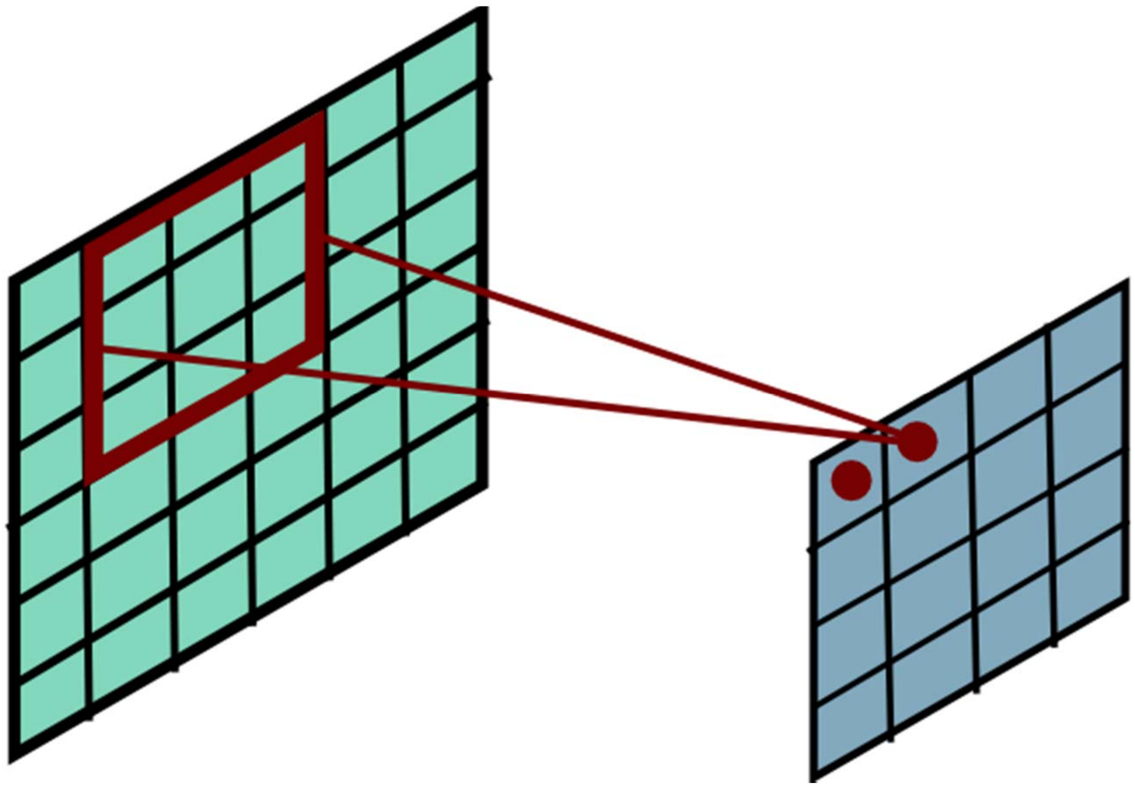


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

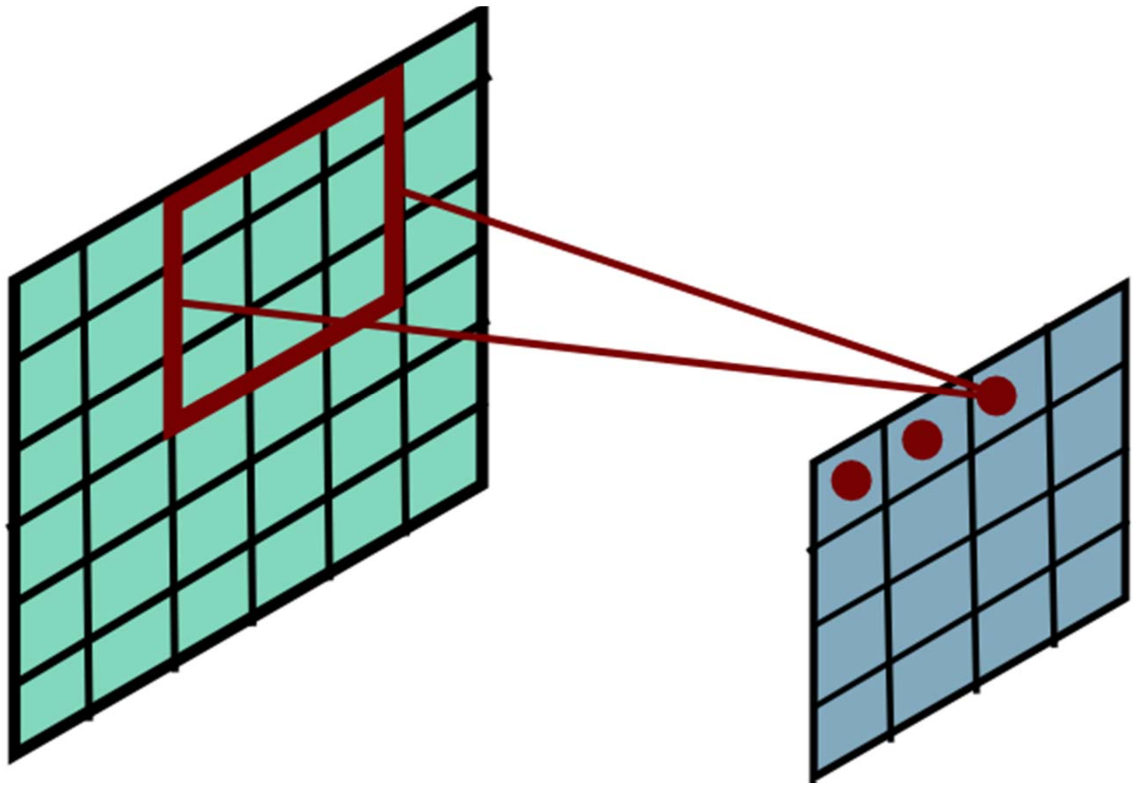
Convolutional Layer



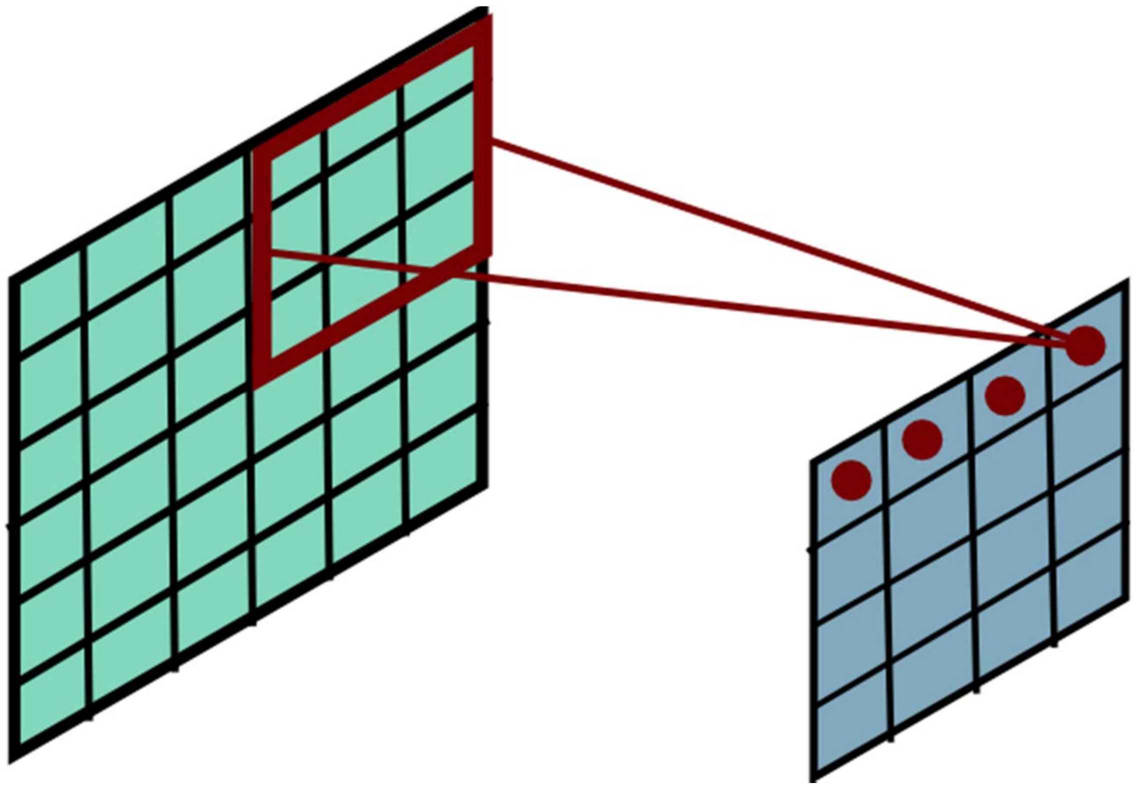
Convolutional Layer



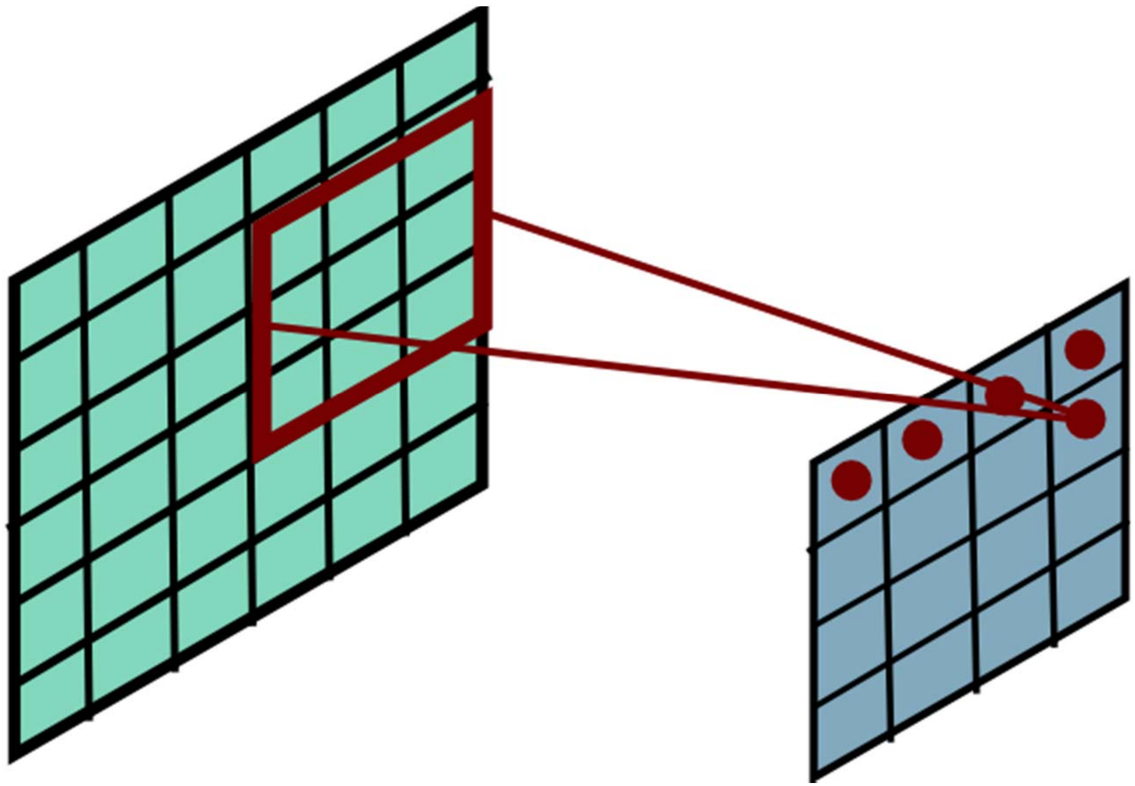
Convolutional Layer



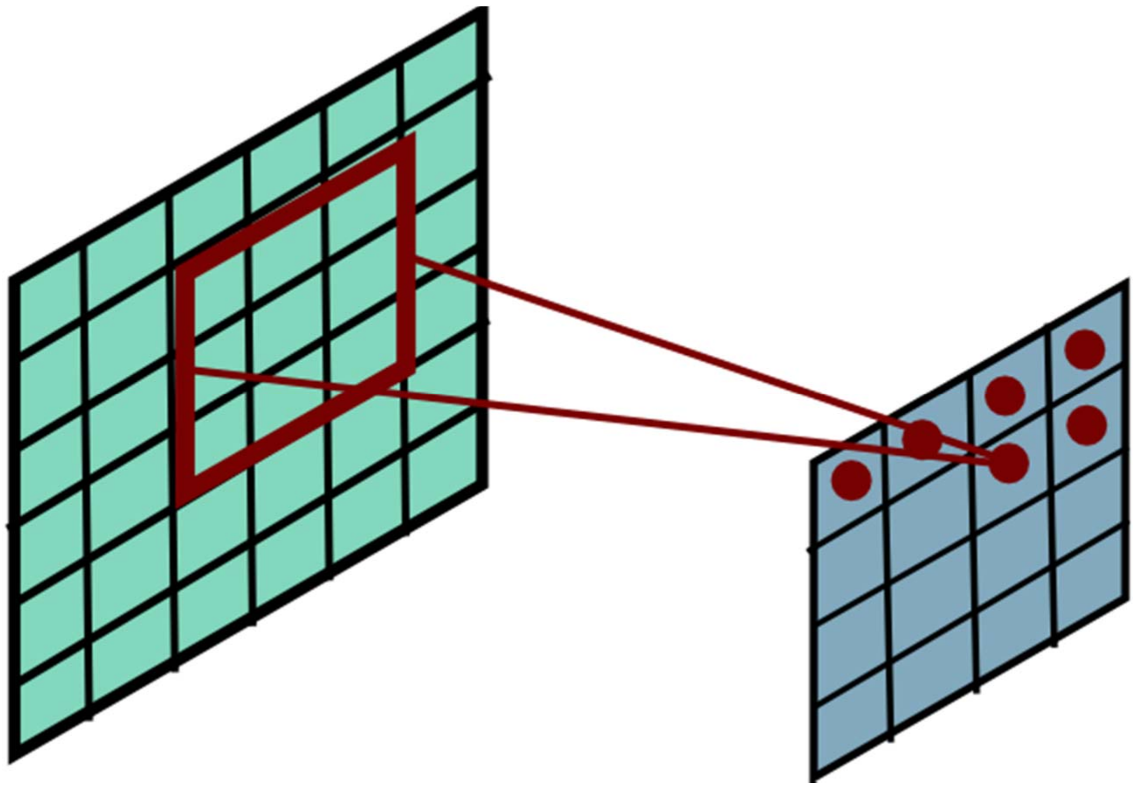
Convolutional Layer



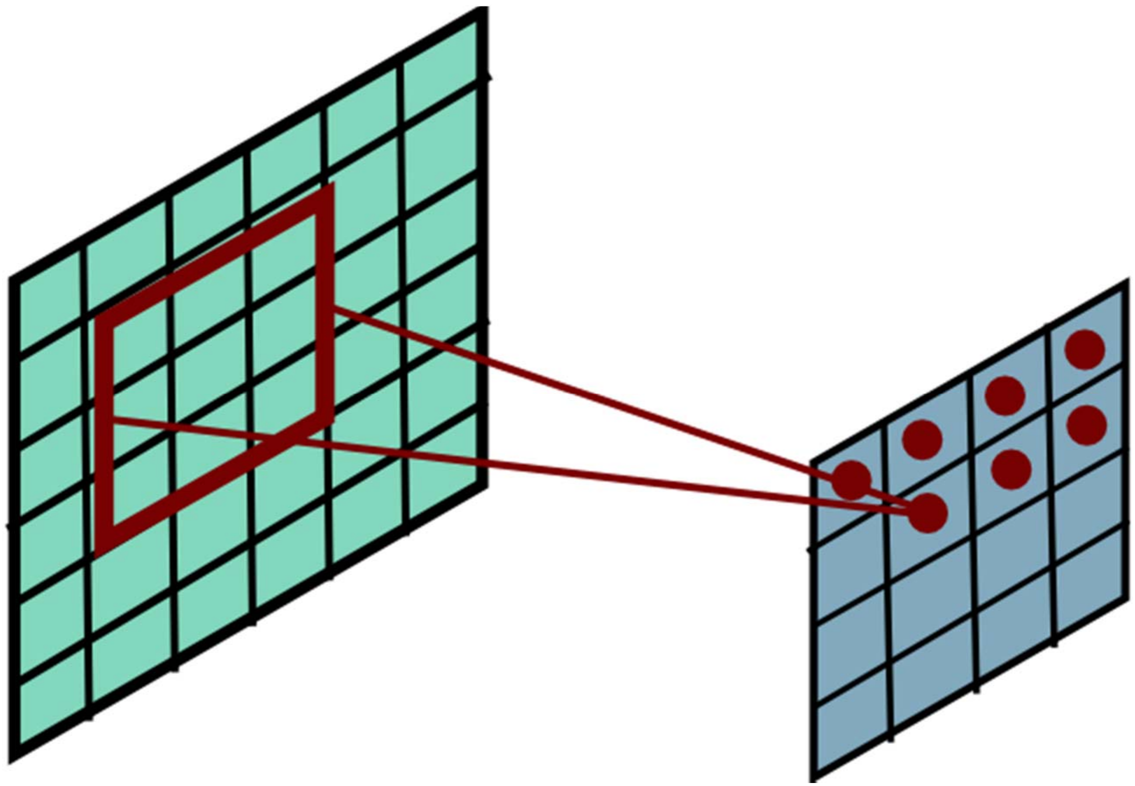
Convolutional Layer



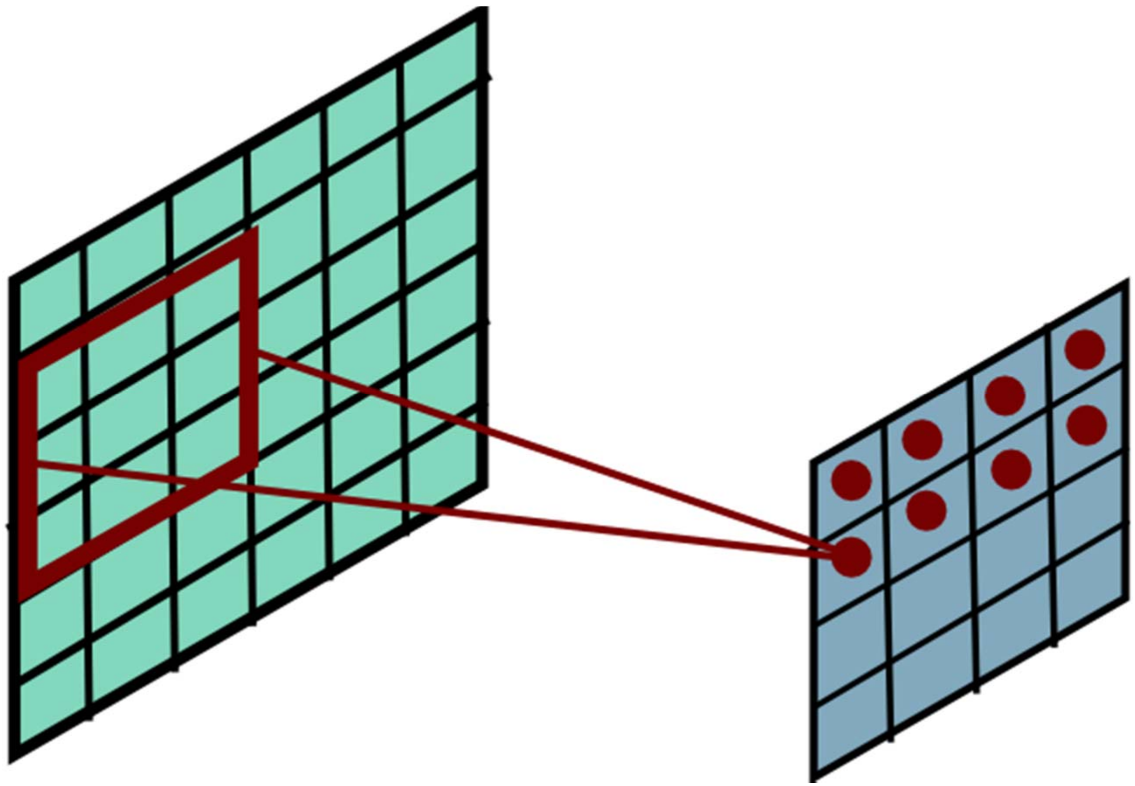
Convolutional Layer



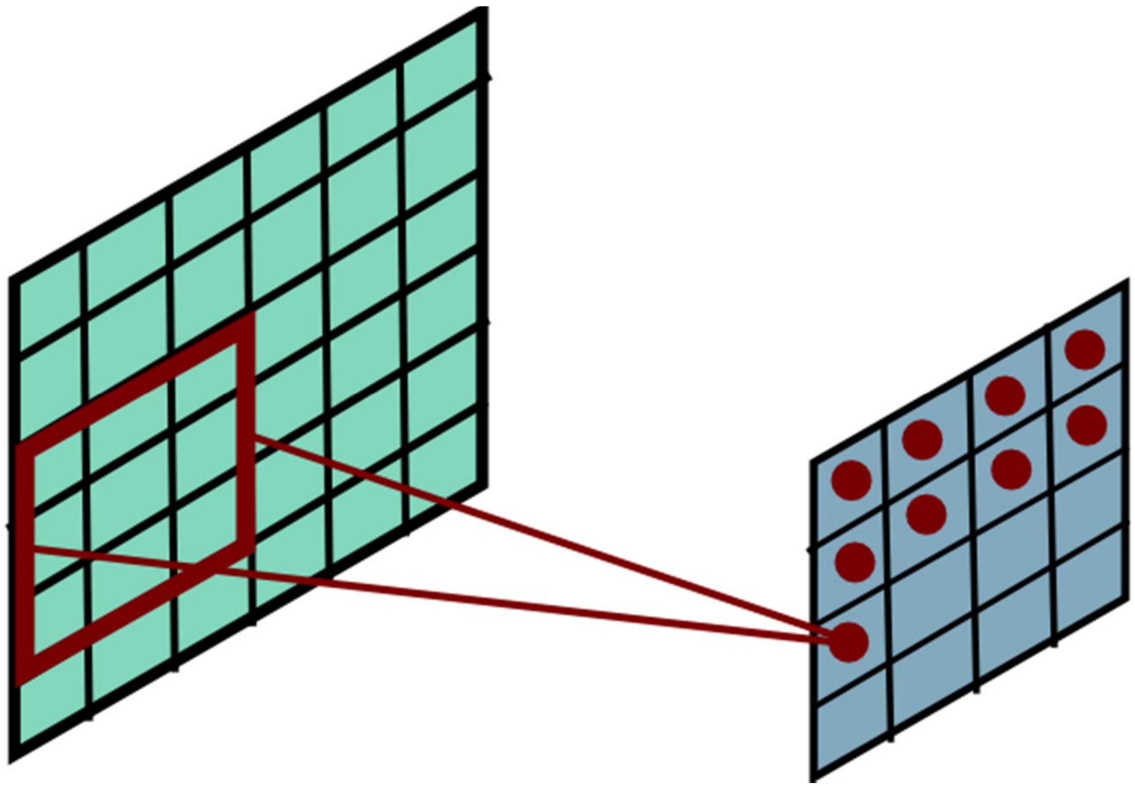
Convolutional Layer



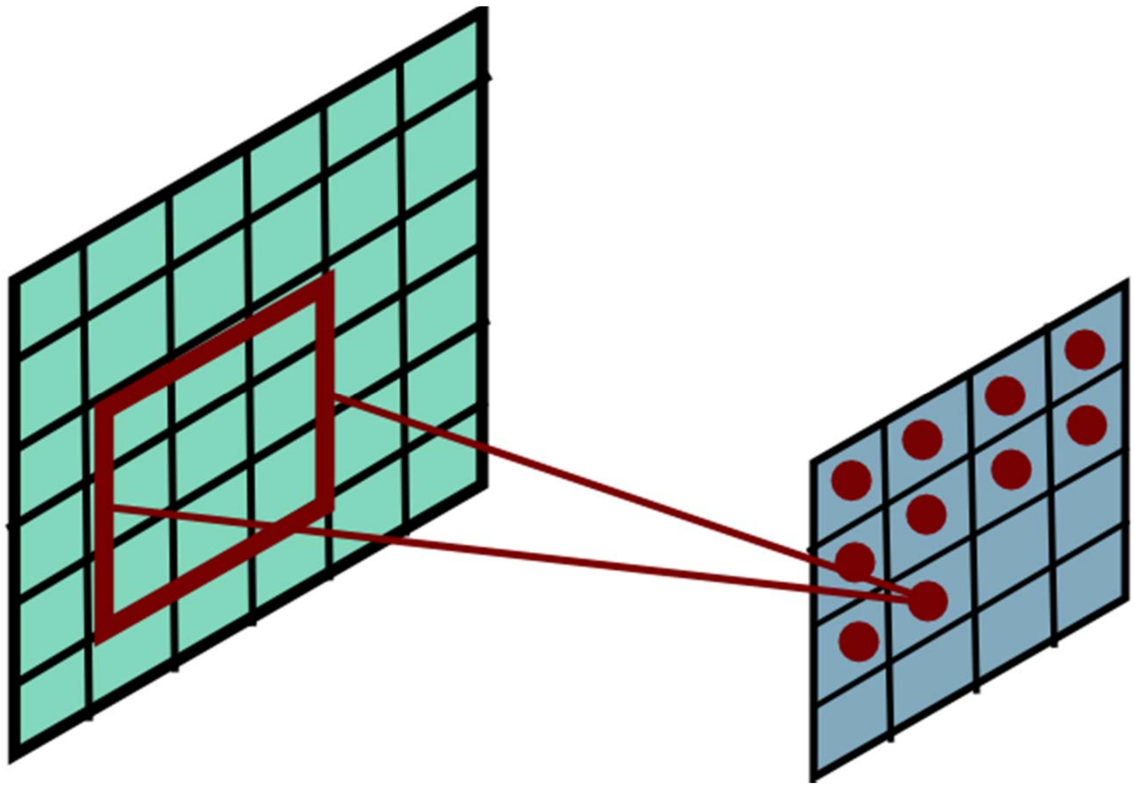
Convolutional Layer



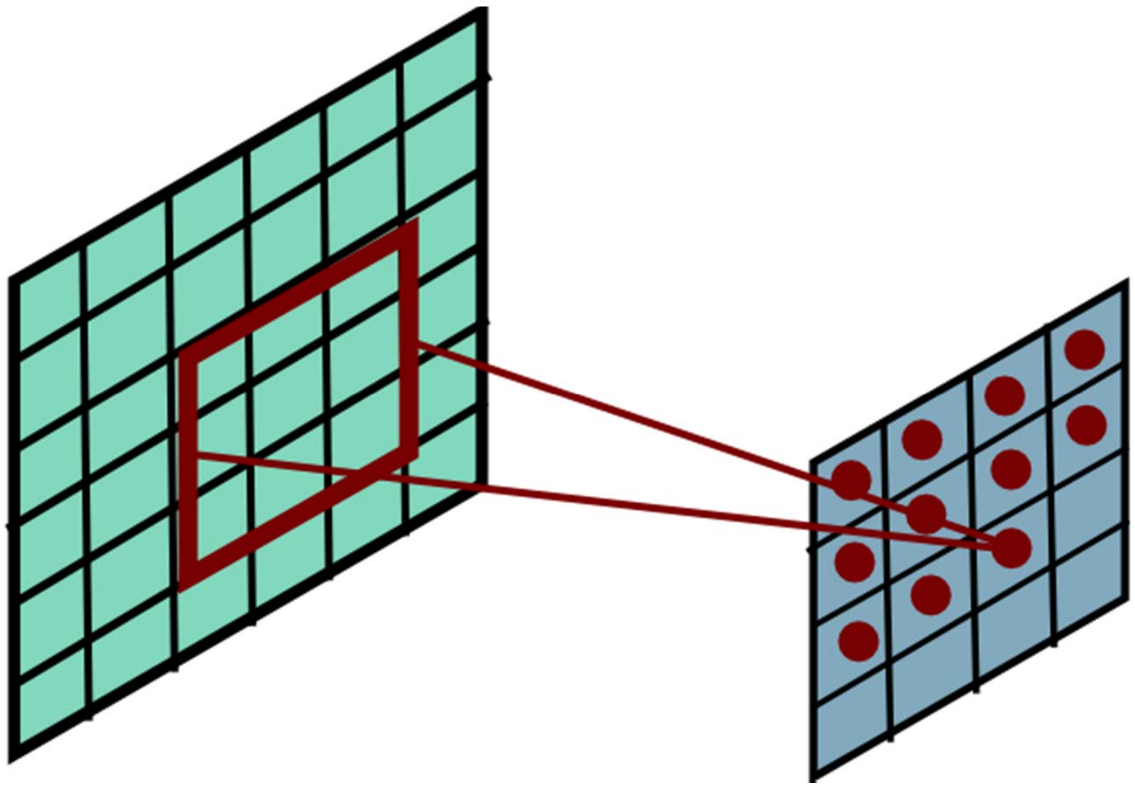
Convolutional Layer



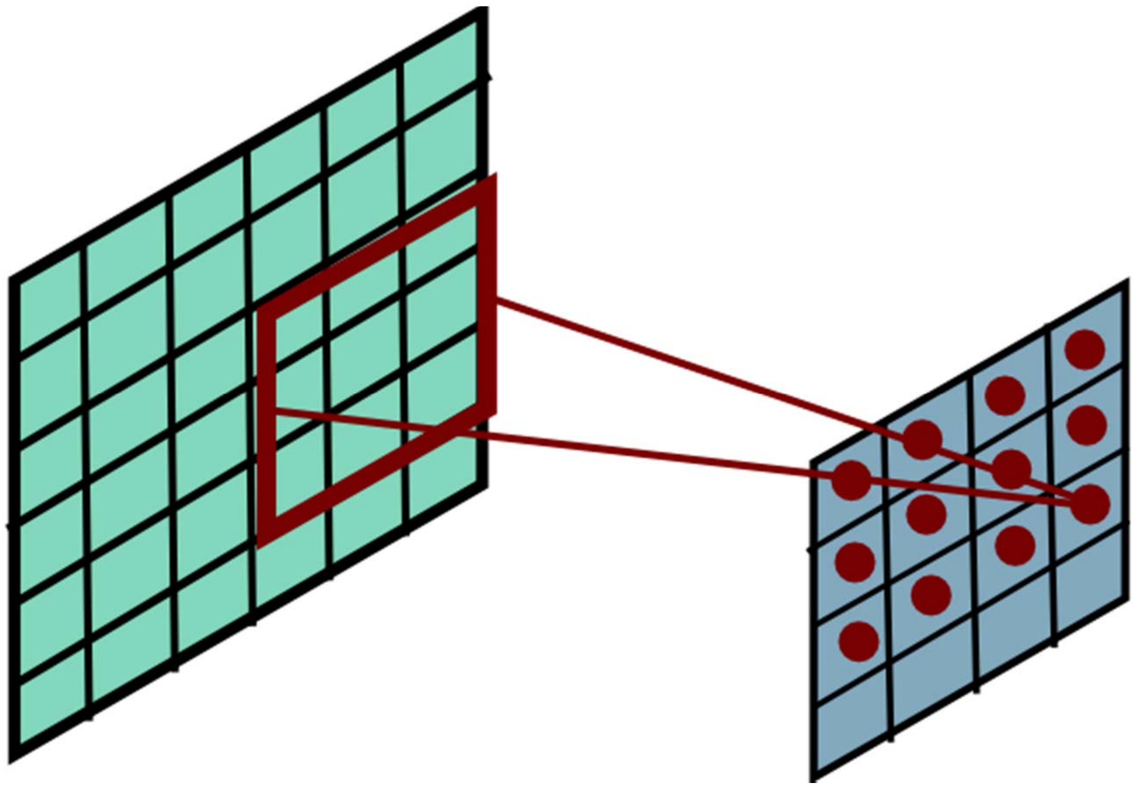
Convolutional Layer



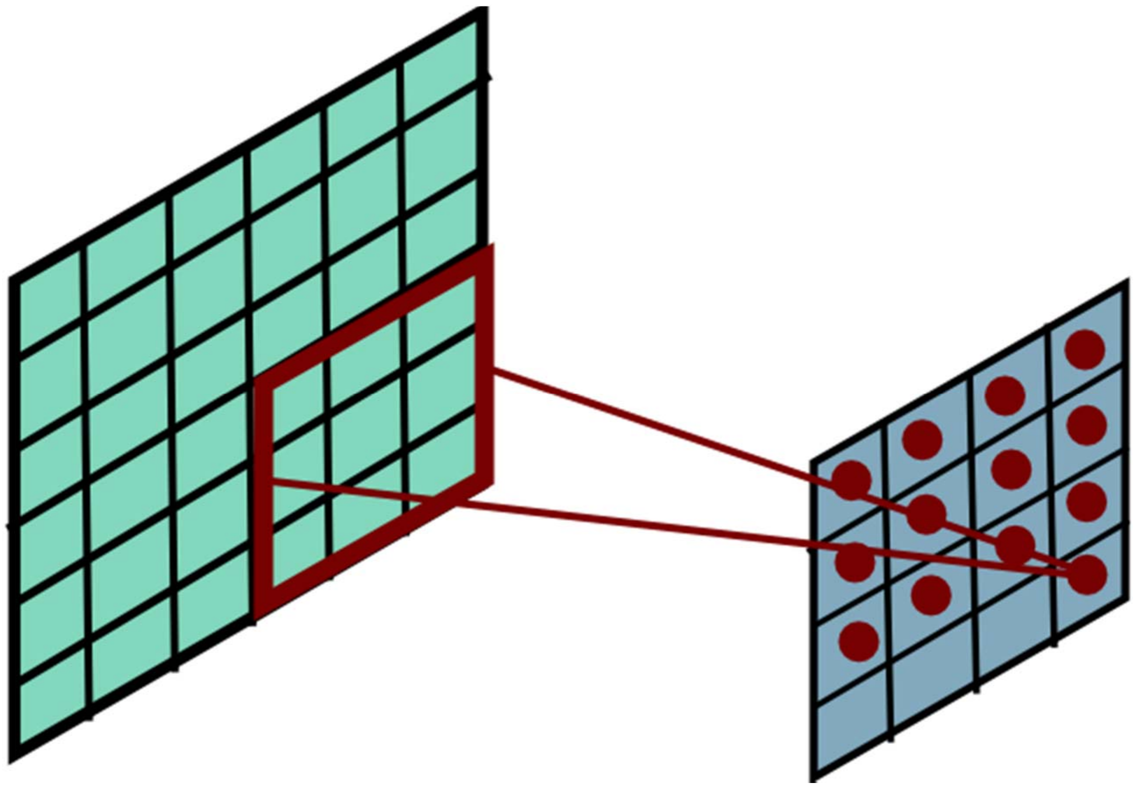
Convolutional Layer



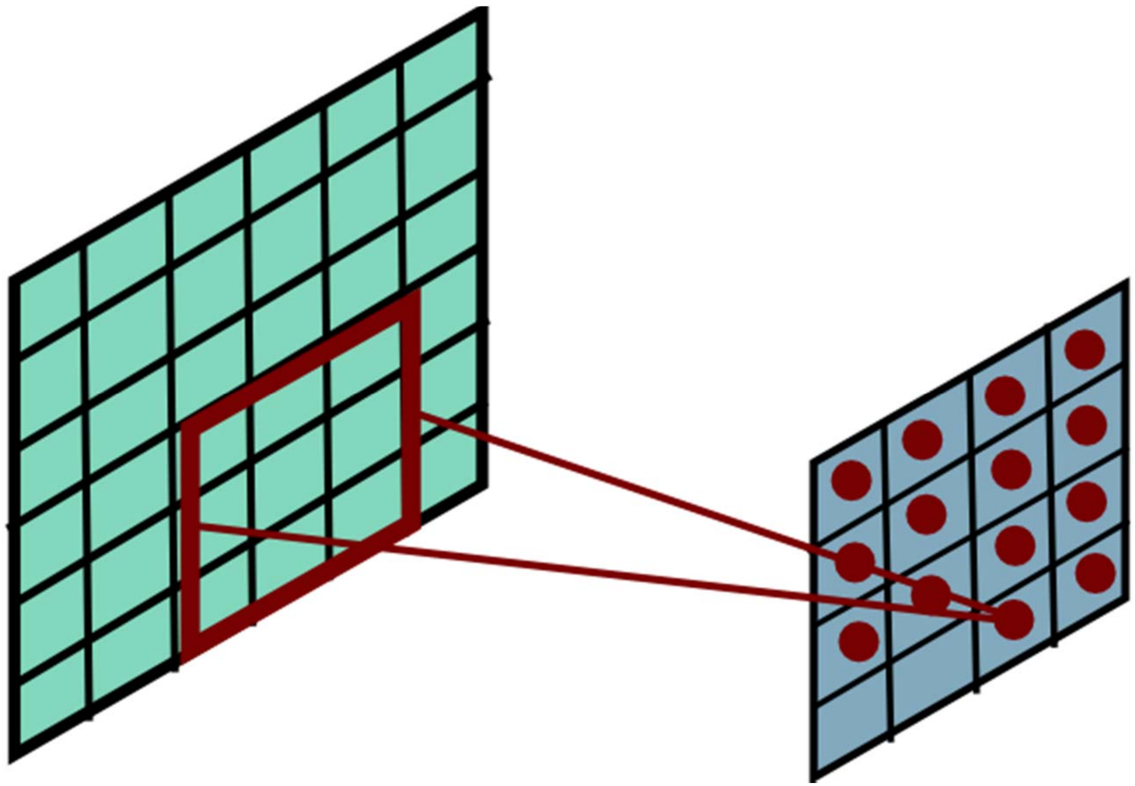
Convolutional Layer



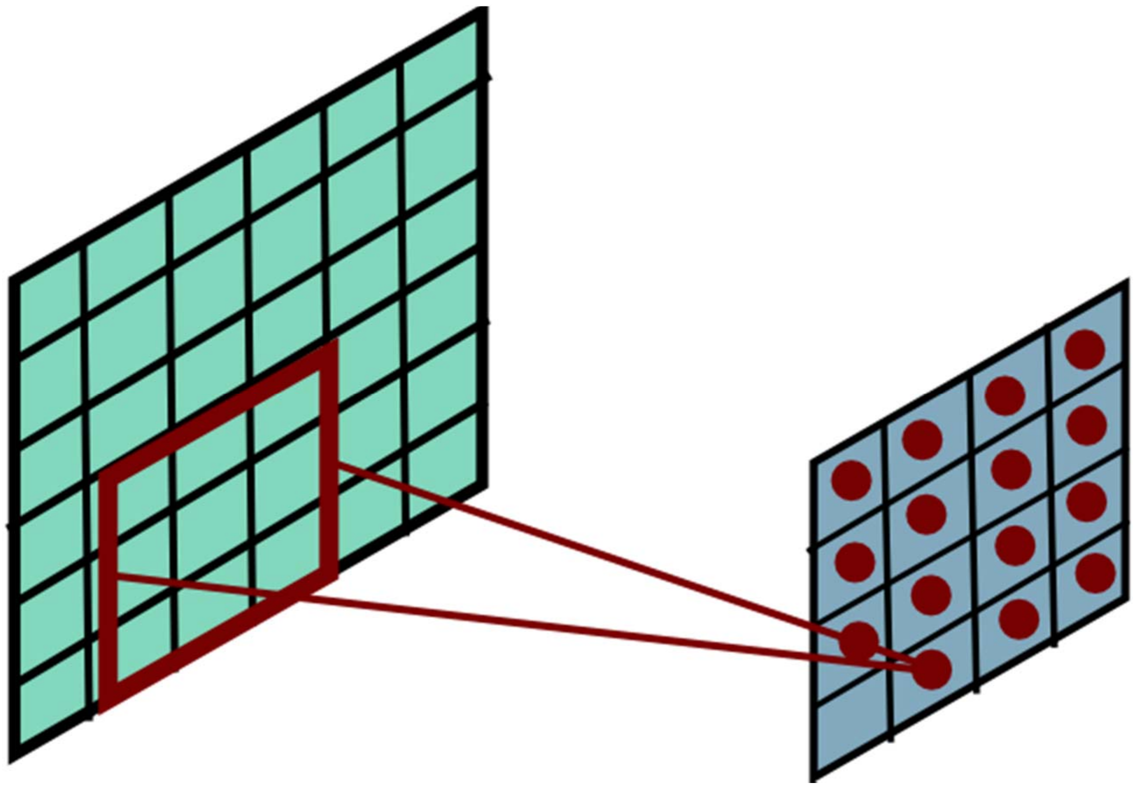
Convolutional Layer



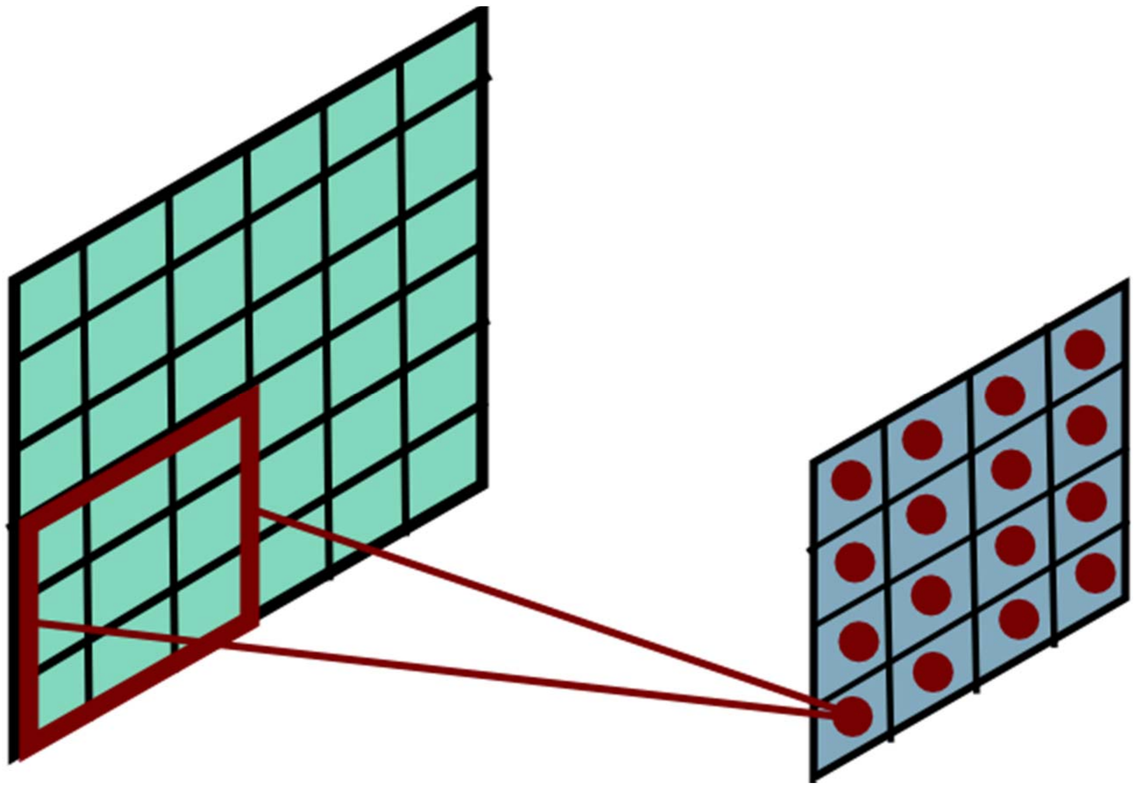
Convolutional Layer



Convolutional Layer

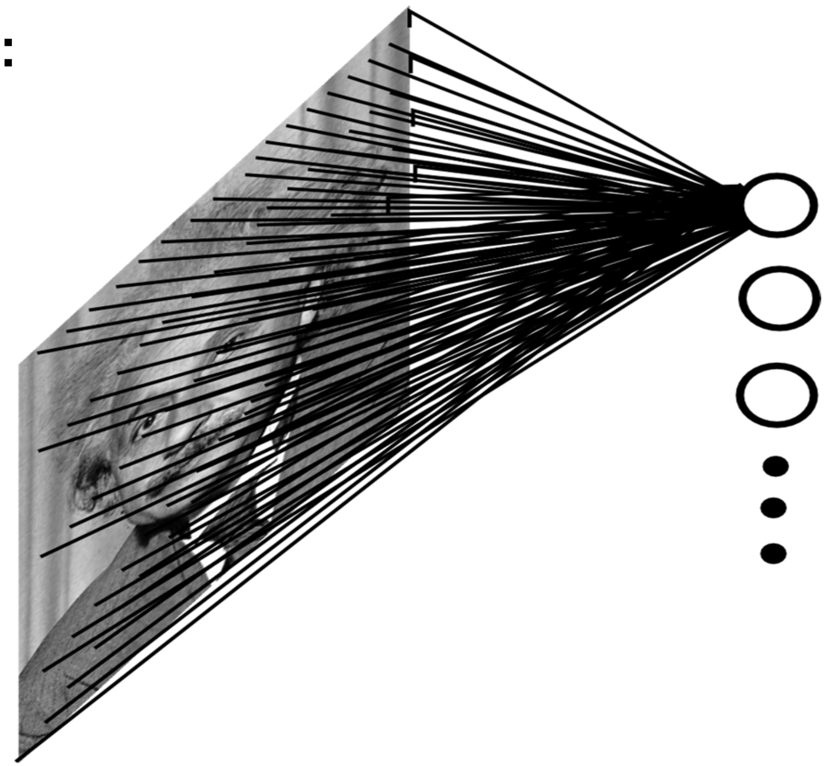


Convolutional Layer



"Fully-connected" layer

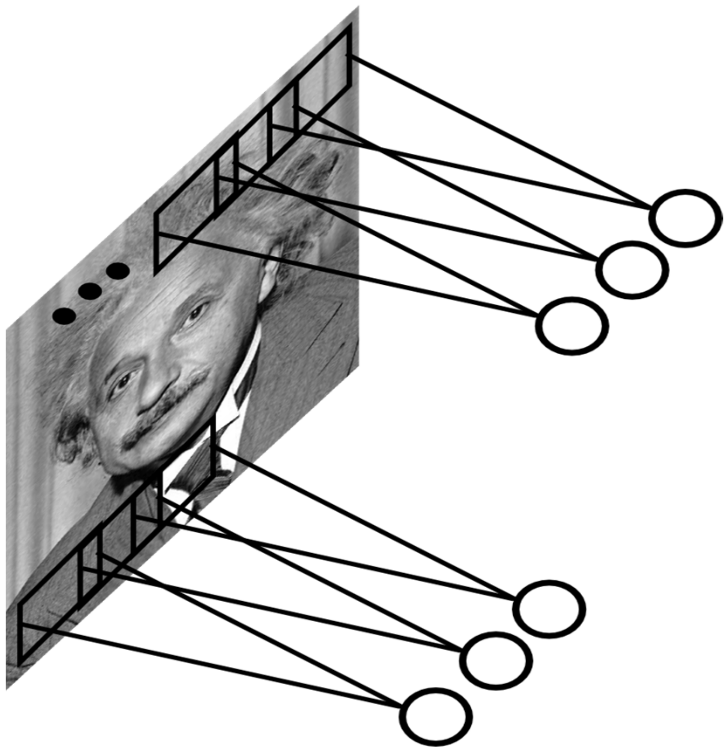
#of parameters:
K^2



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & \cdots & w_{1,K} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & \cdots & w_{2,K} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & \cdots & w_{3,K} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & \cdots & w_{4,K} \\ & & \vdots & & & \\ w_{K,1} & w_{K,2} & w_{K,3} & w_{K,4} & \cdots & w_{K,K} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

"Convolutional" layer

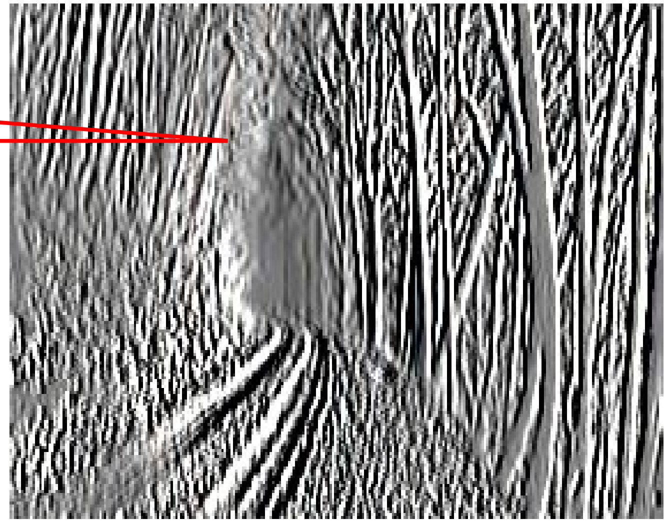
#of parameters: size of window



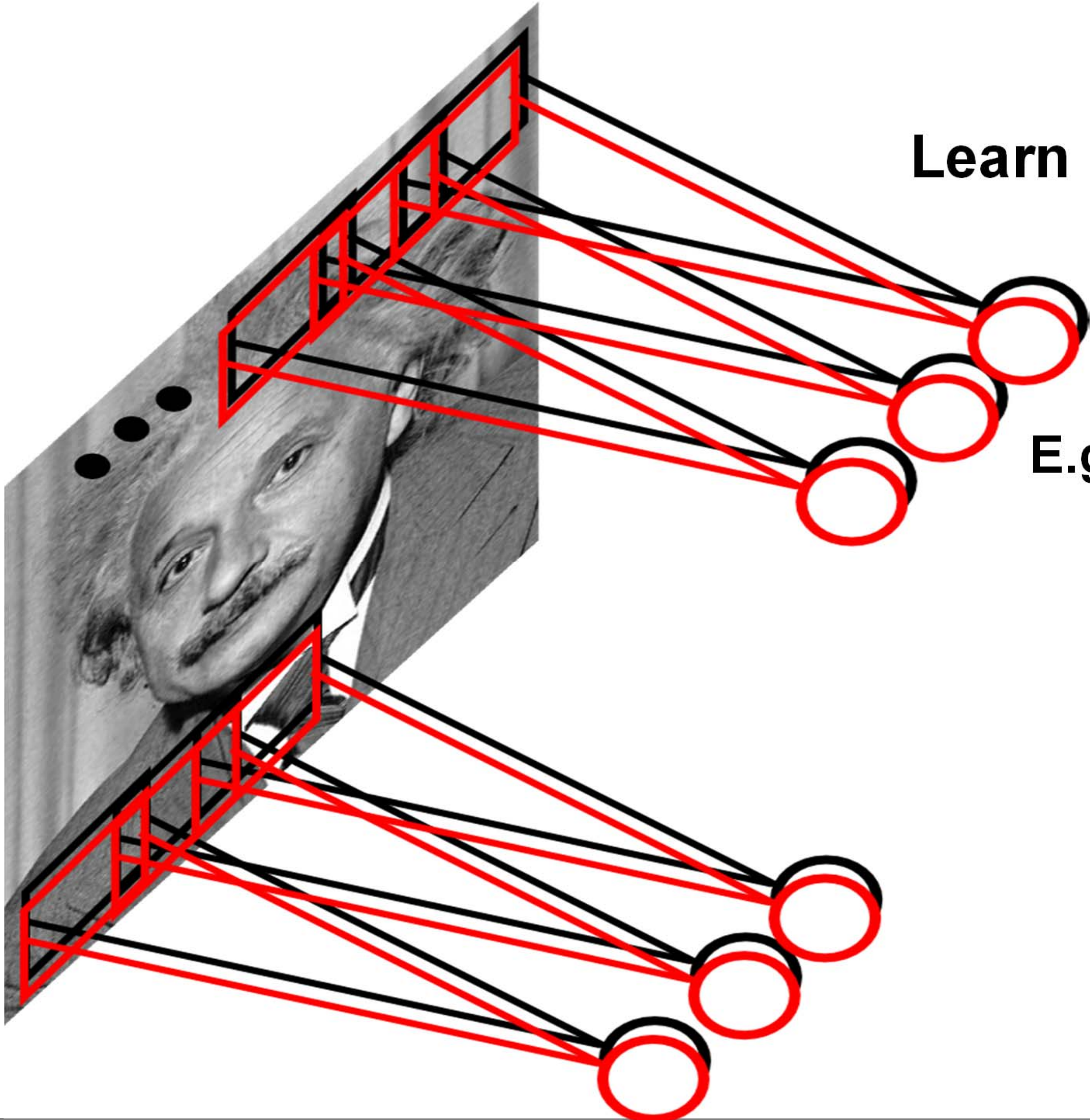
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_0 & w_1 & w_2 & 0 & \dots & 0 \\ 0 & w_0 & w_1 & w_2 & \dots & 0 \\ 0 & 0 & w_0 & w_1 & \dots & 0 \\ 0 & 0 & 0 & w_0 & \dots & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & w_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

Convolutional Layer

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



Convolutional Layer



Learn **multiple filters.**

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

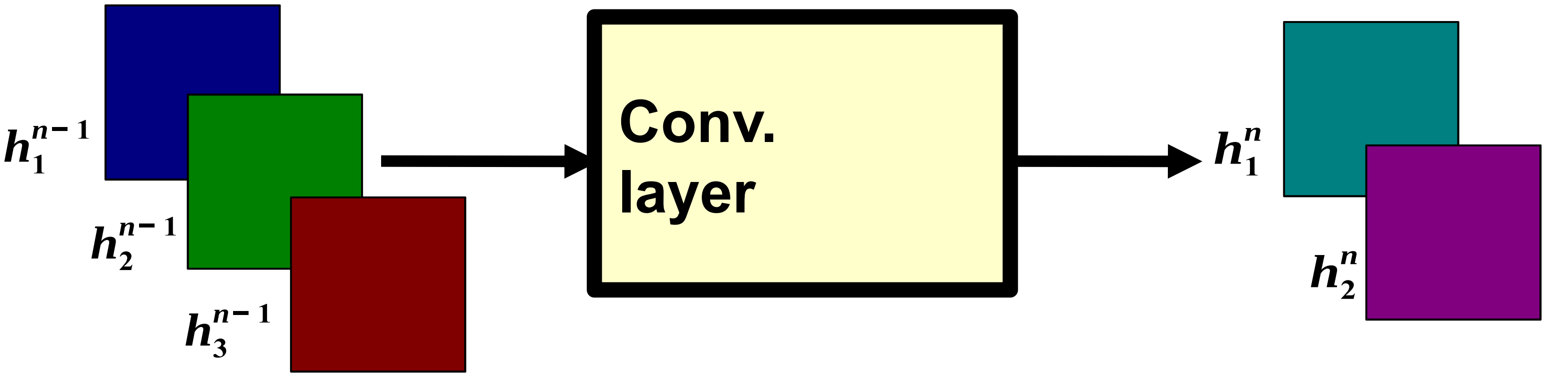
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

kernel



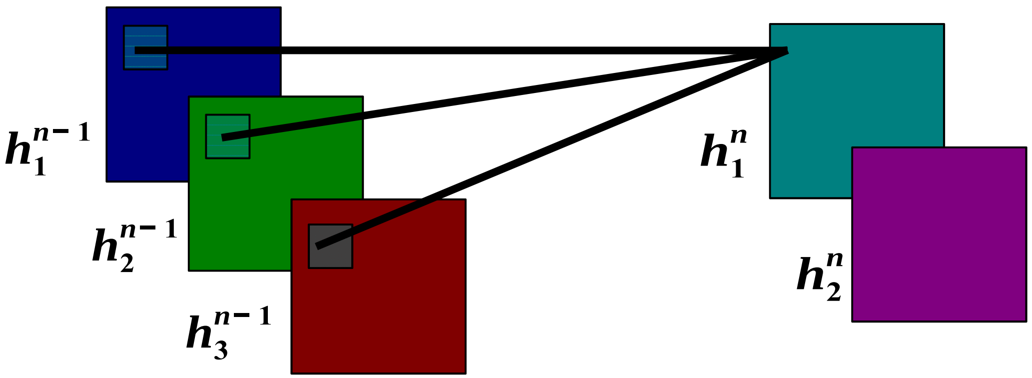
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

kernel



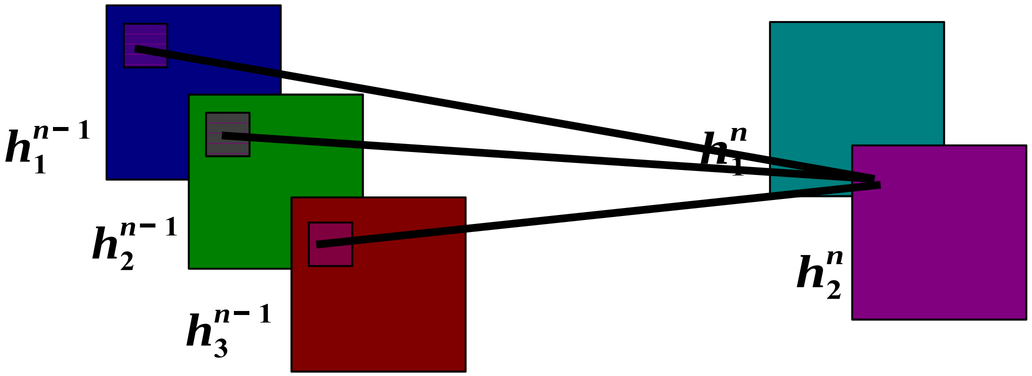
Convolutional Layer

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

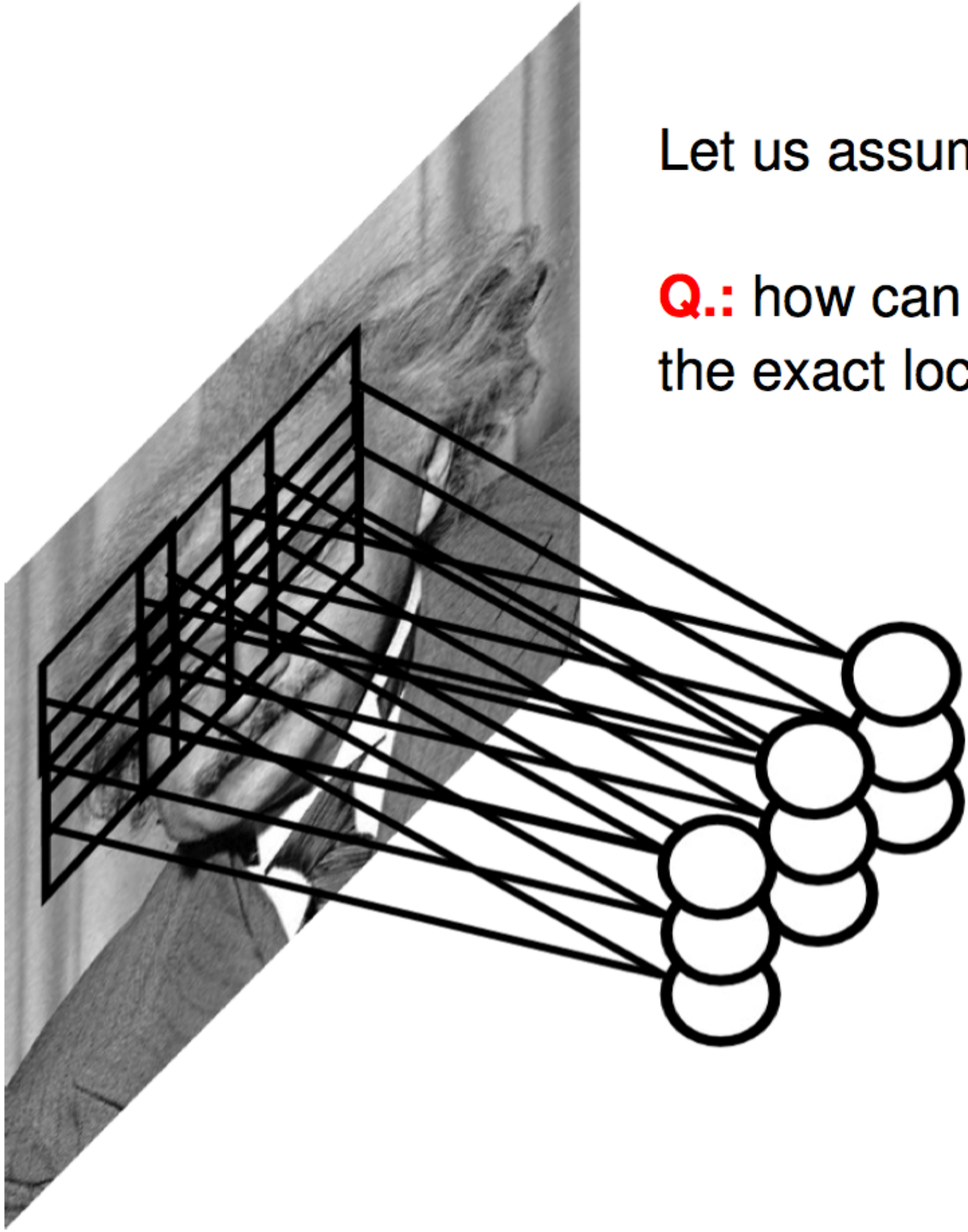
output feature map

input feature map

kernel



Pooling Layer

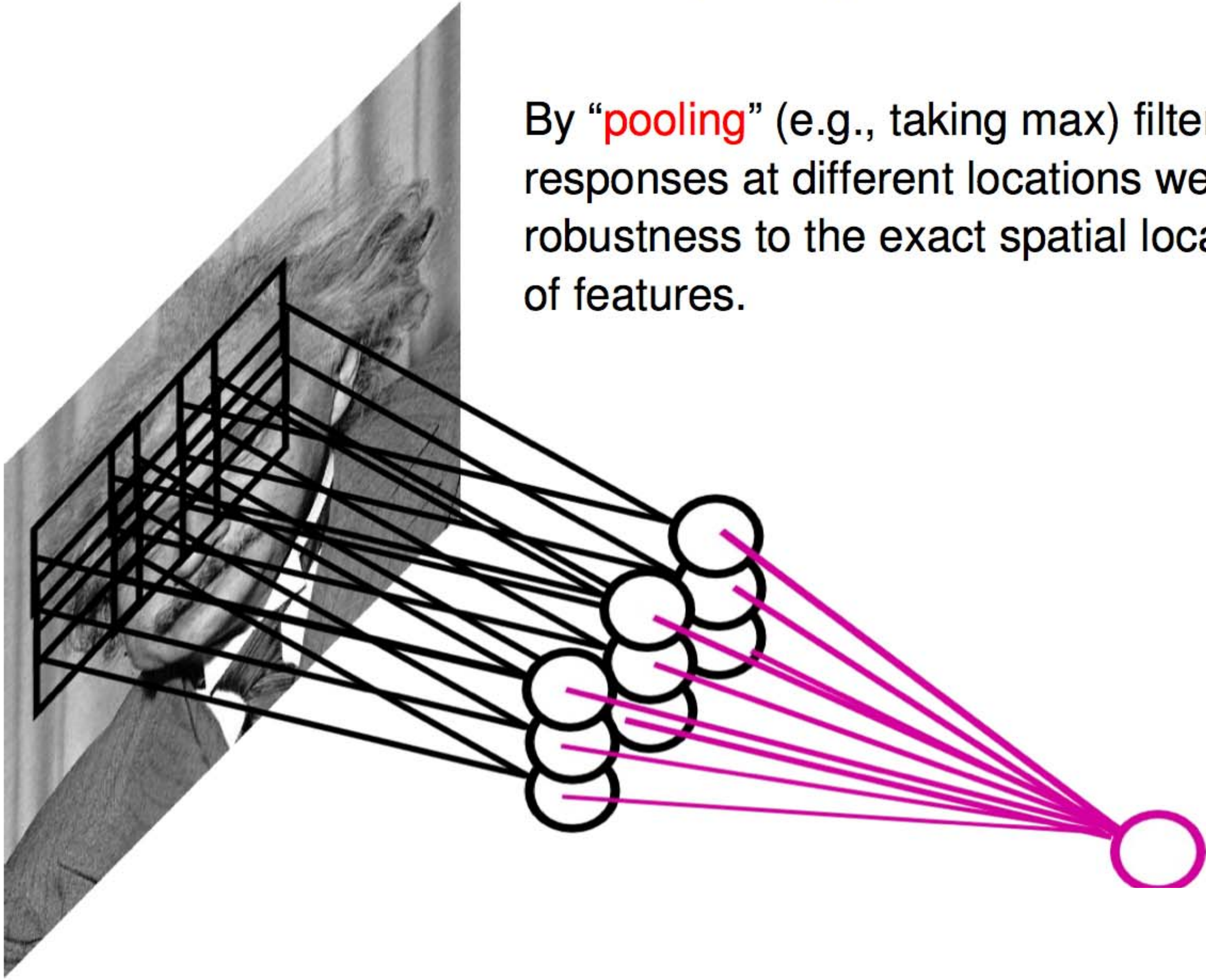


Let us assume filter is an “eye” detector.

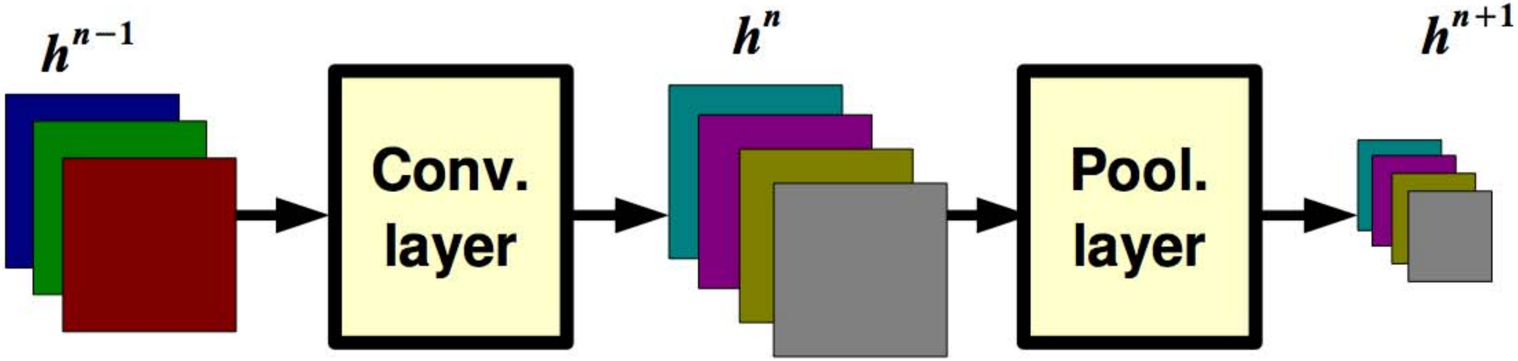
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer

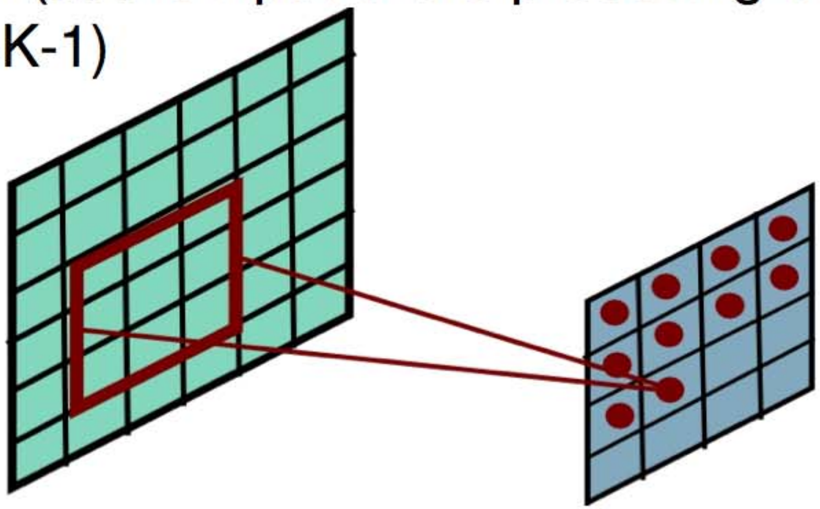
By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.



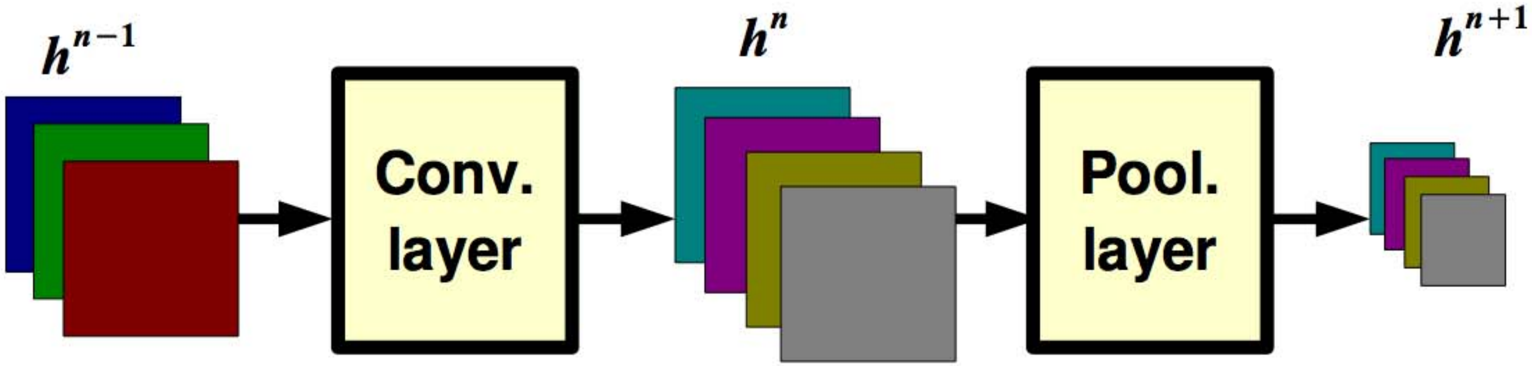
Pooling Layer: Receptive Field Size



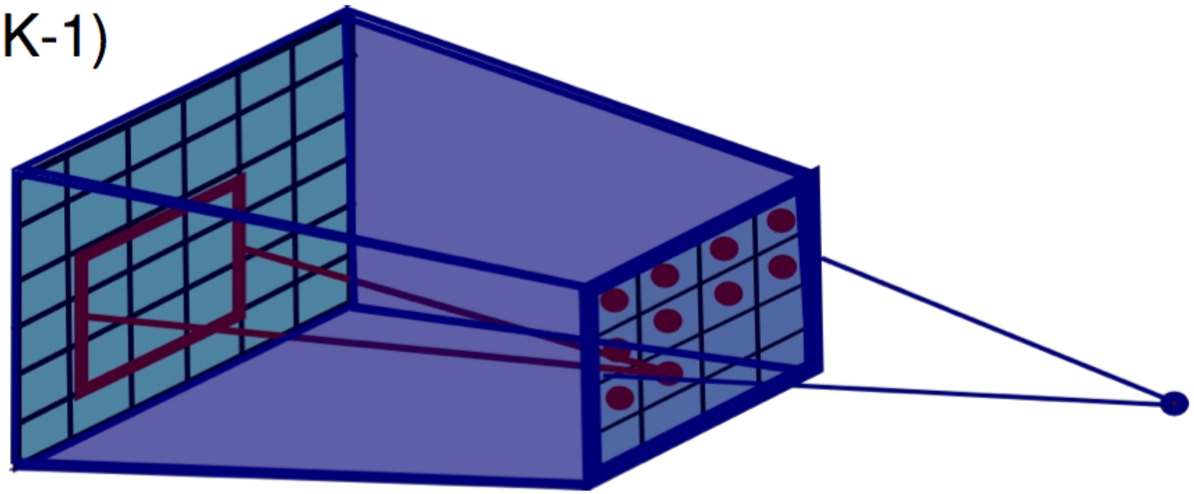
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Pooling Layer: Receptive Field Size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



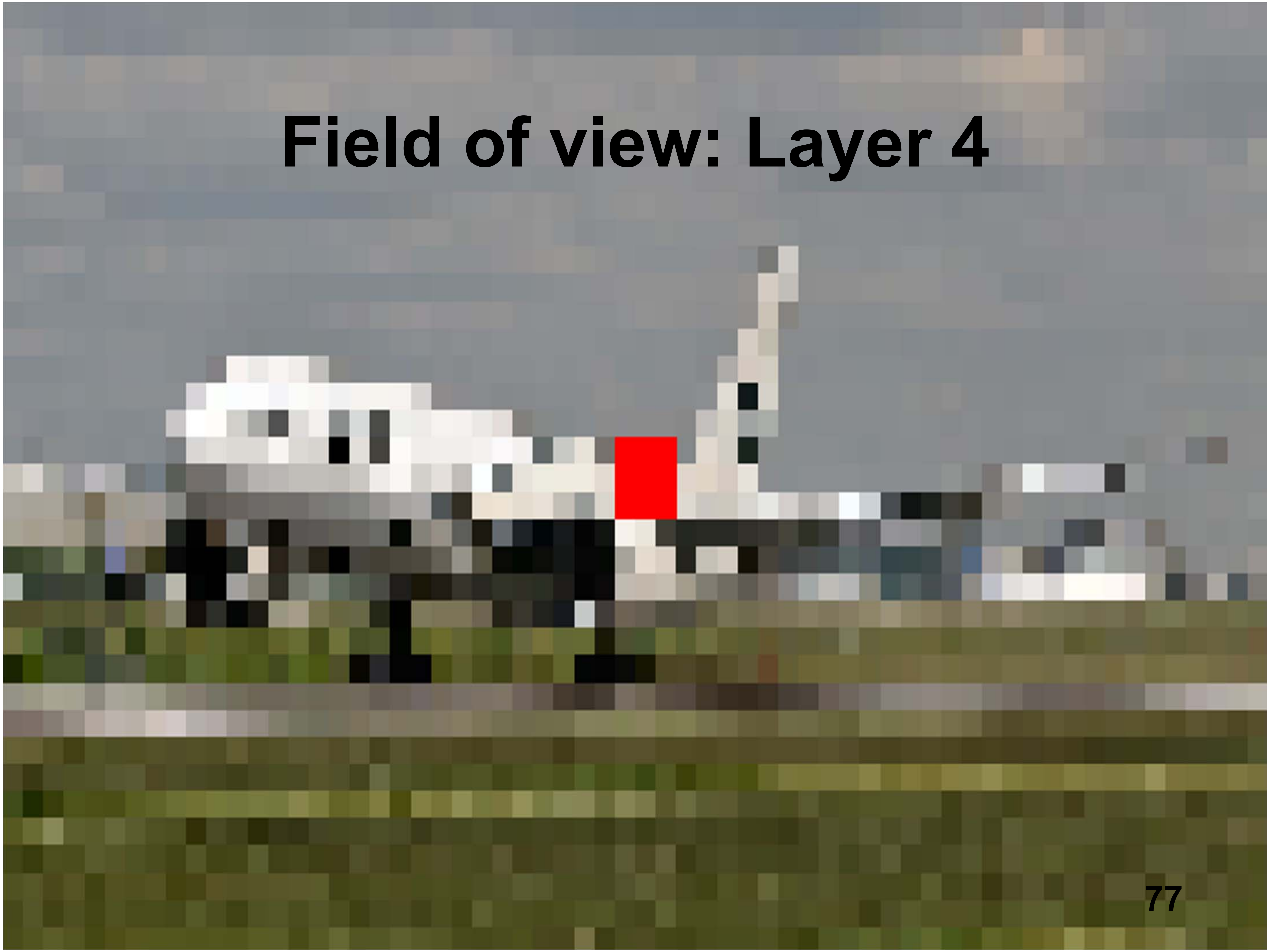
Field of view

Field of view: Layer 1

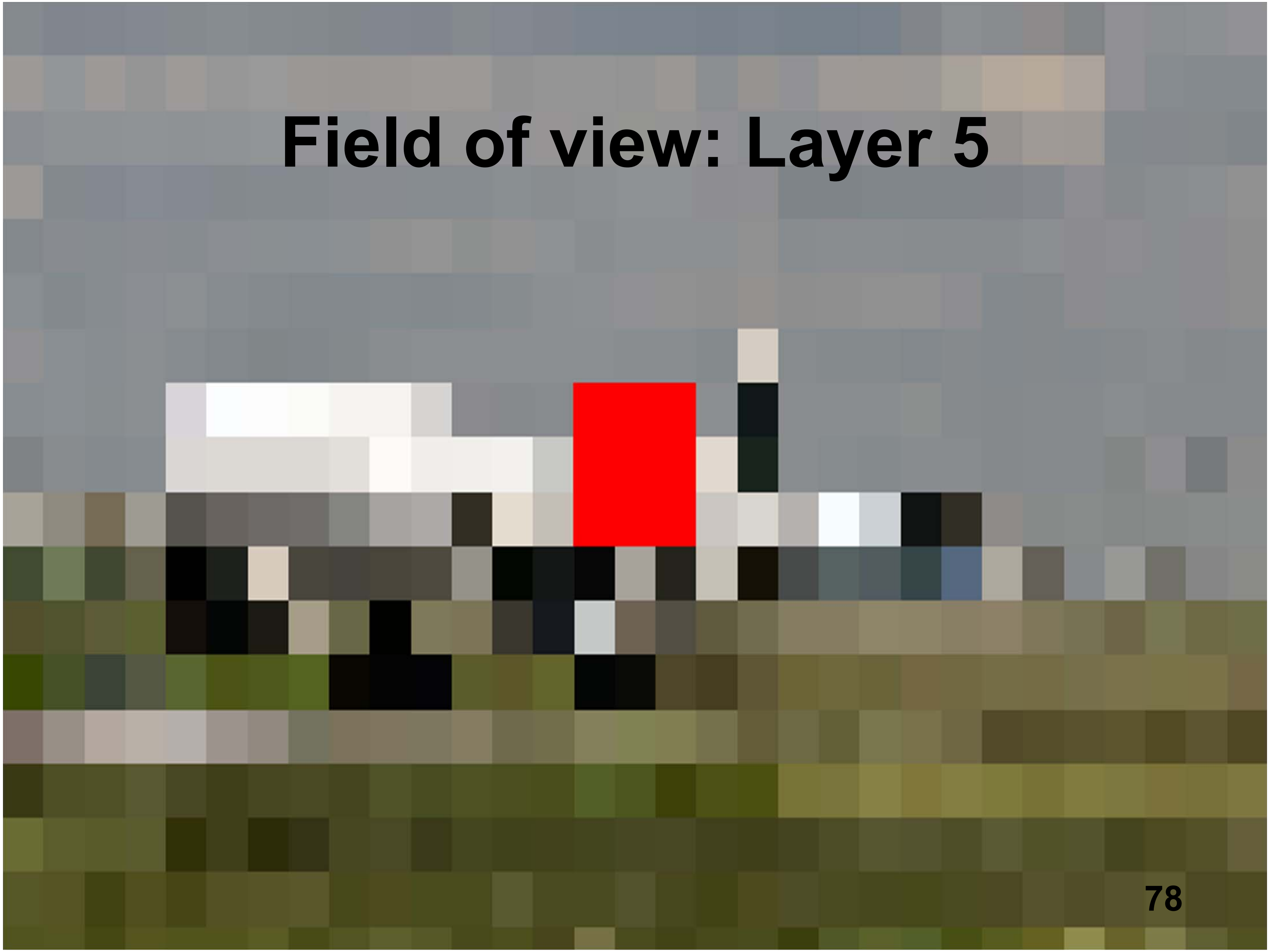
Field of view: Layer 2

Field of view: Layer 3

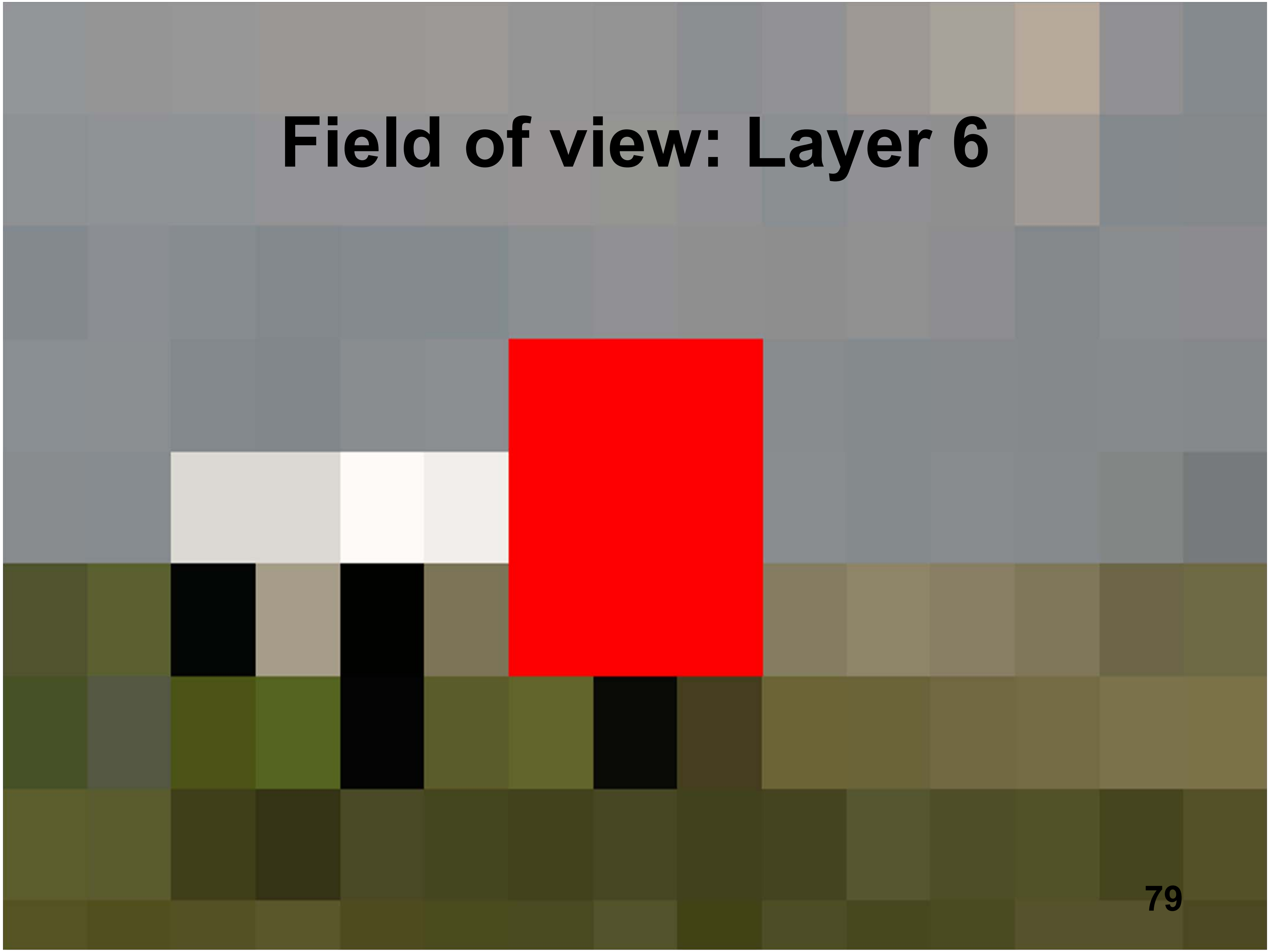
Field of view: Layer 4



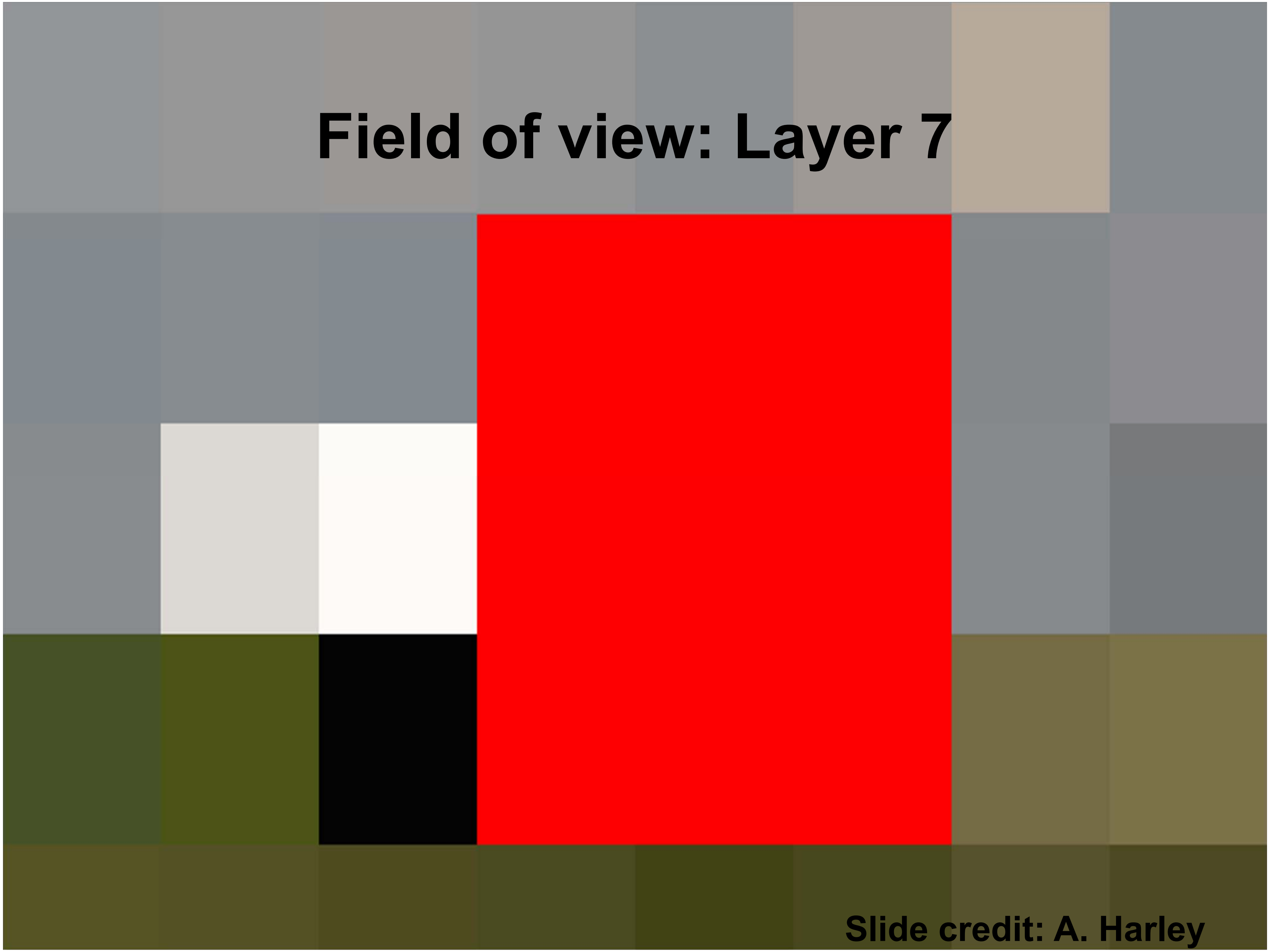
Field of view: Layer 5



Field of view: Layer 6



Field of view: Layer 7

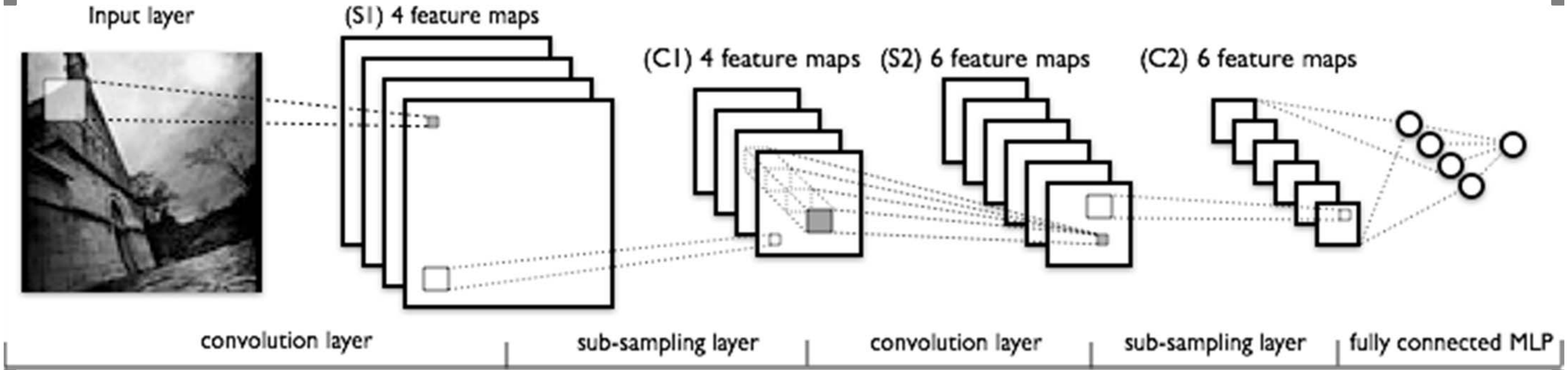


Slide credit: A. Harley

Field of view: Layer 8

Slide credit: A. Harley

Convolutional Nets



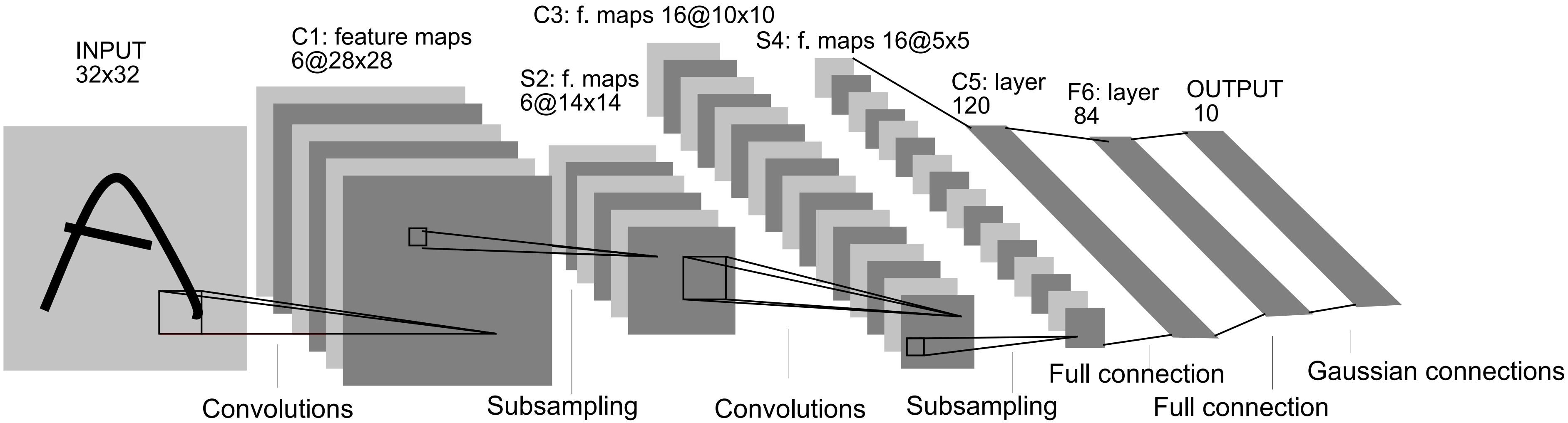


Image Credit: Yann LeCun, Kevin Murphy

Deep learning, AD 1993

http://yann.lecun.com/exdb/lenet/
(LeNet 5, 1998)

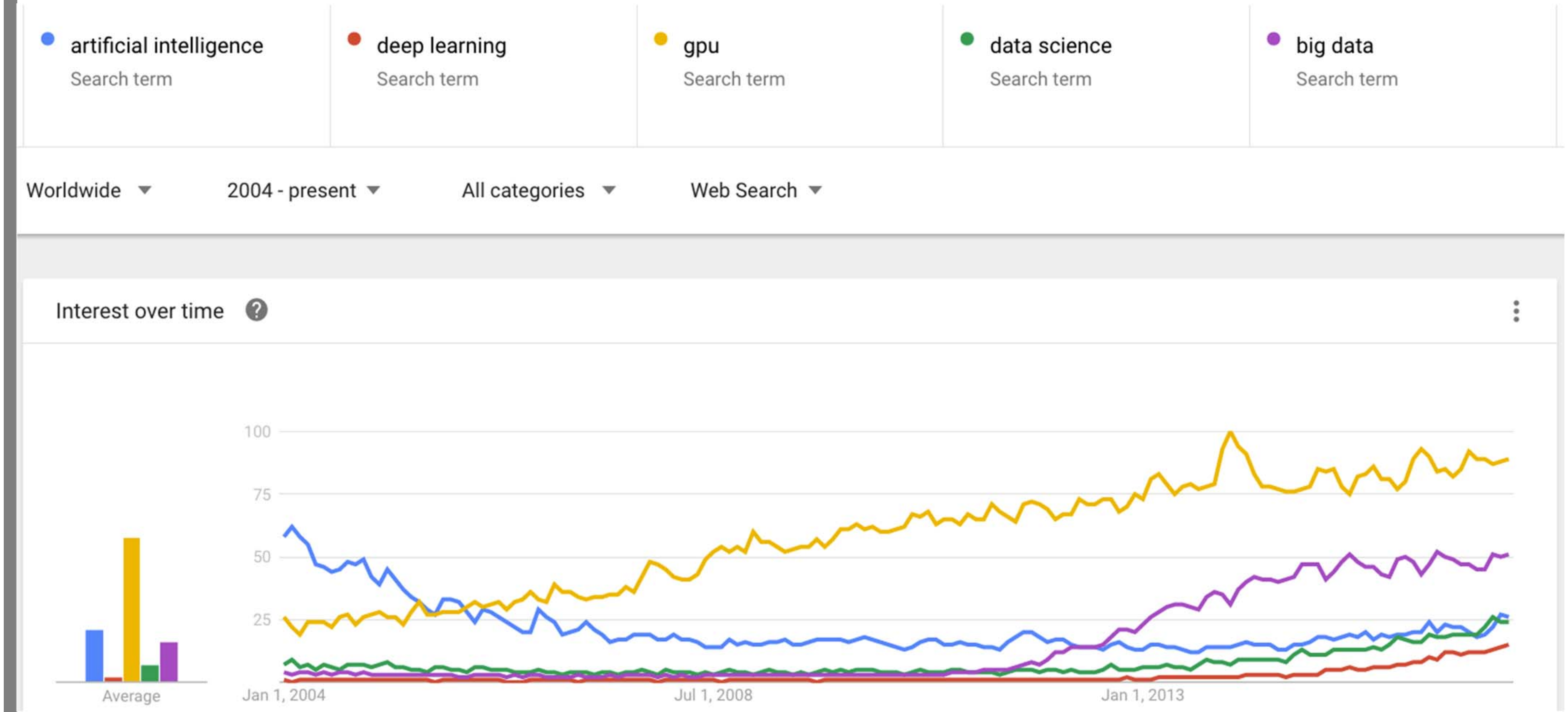
https://www.youtube.com/watch?v=FwFduRA_L6Q
(LeNet 1, 1993)

The 82 errors made by LeNet5

Notice that most of the errors are cases that people find quite easy.

The human error rate is probably 20 to 30 errors but nobody has had the patience to measure it.

What happened in between?



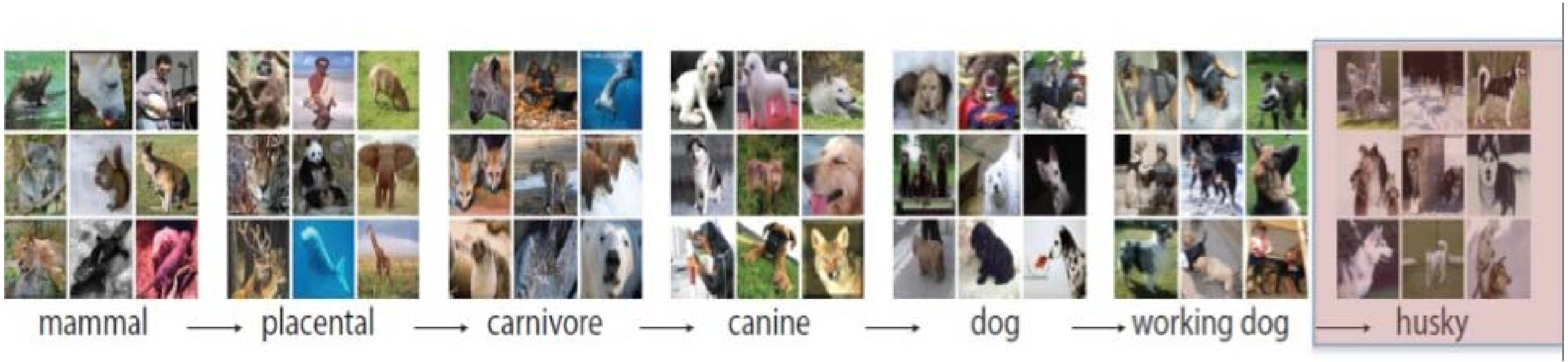
deep learning = neural networks (+ big data + GPUs)


WAIT

Computer Vision Data: Big and Complicated

http://www.image-net.org/

IMAGENET



Deng et. al, ImageNet: A Large-Scale Hierarchical Image Database, CVPR'09

Computer Vision Data: Big and Complicated

http://www.image-net.org/

IMAGENET

Overall

- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908

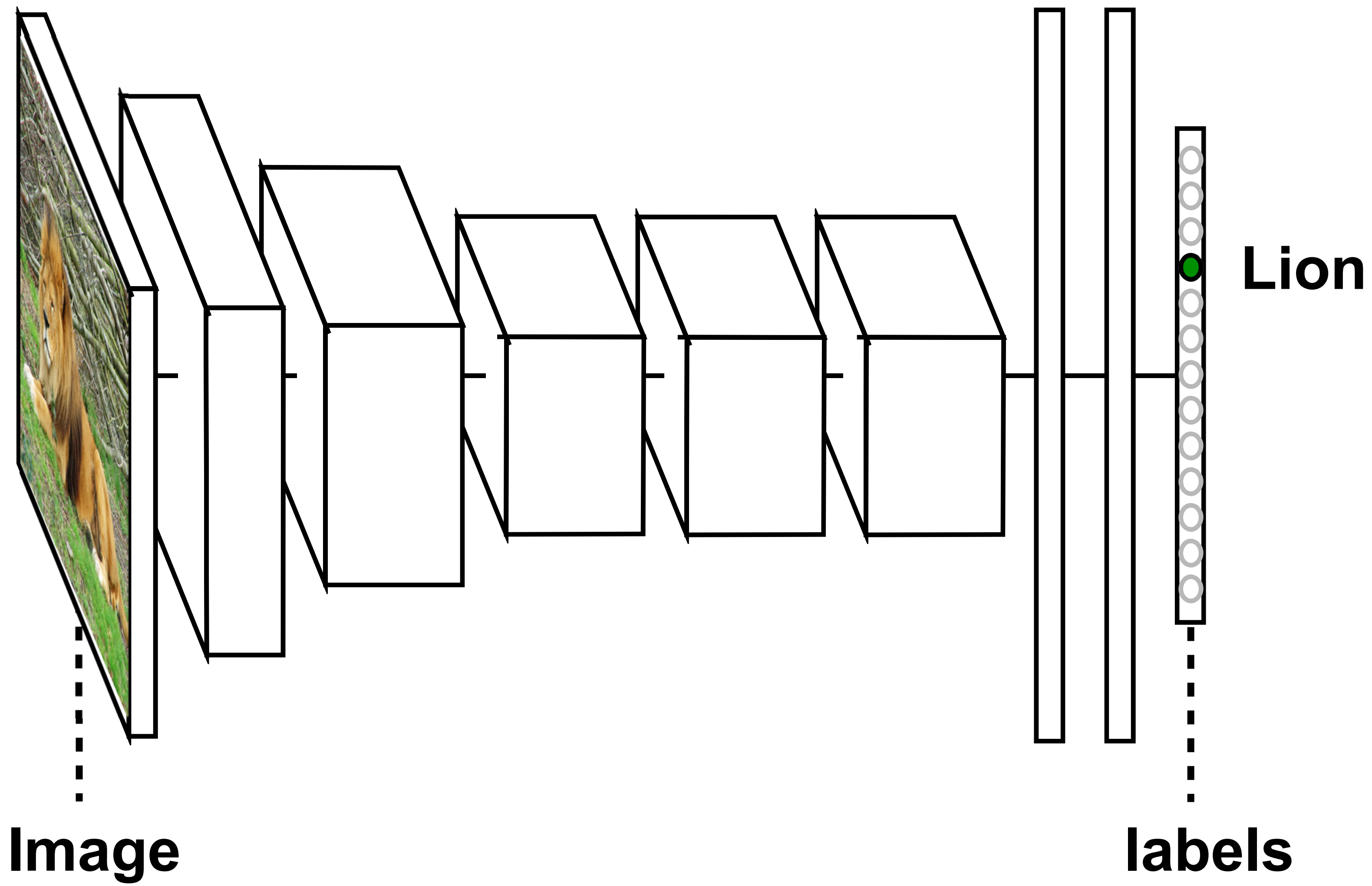
High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

Deng et. al, ImageNet: A Large-Scale Hierarchical Image Database, CVPR’09

Computer Vision Data: Big and Complicated

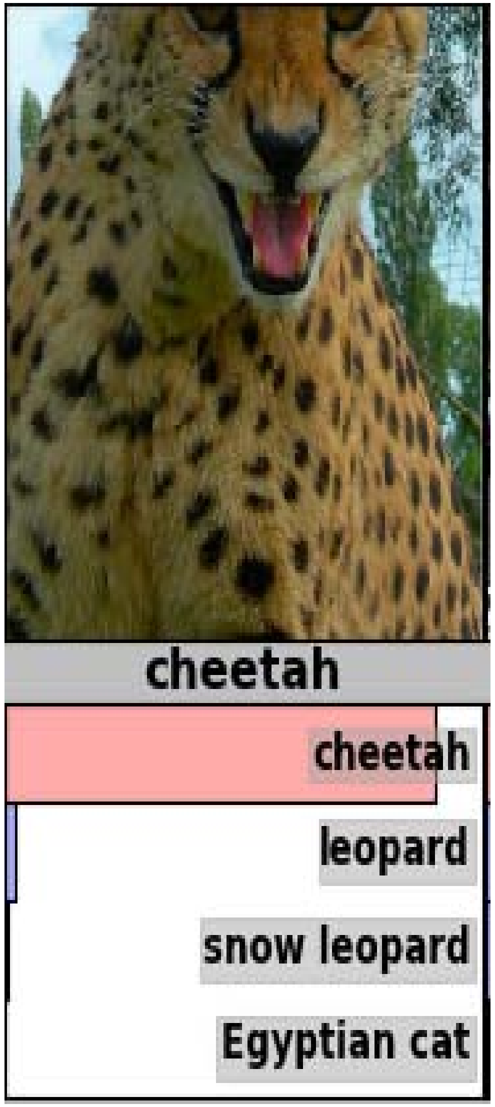
Examples of hammer:

Deep Learning for image classification



A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*13

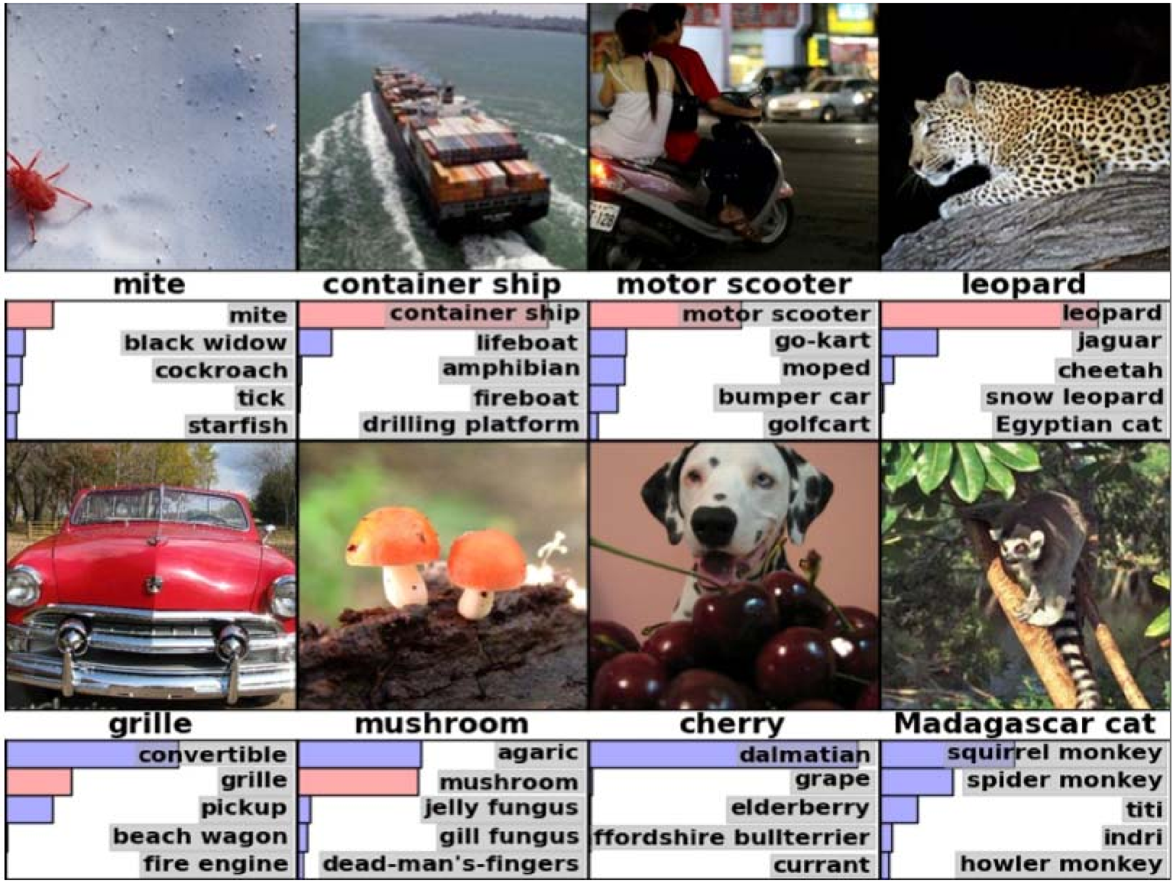
Examples from the test set (with the network's guesses)



Error rates on the ILSVRC-2012 competition

	classification
University of Toronto (Alex Krizhevsky)	16.4%
University of Tokyo	26.1%
Oxford University Computer Vision Group	26.9%
INRIA (French national research institute in CS) + XRCE (Xerox Research Center Europe)	27.0%
University of Amsterdam	29.5%

Imagenet top-5 error rates



Humans:5.4%

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*13 [16.4%] (best shallow competitor: 26%)

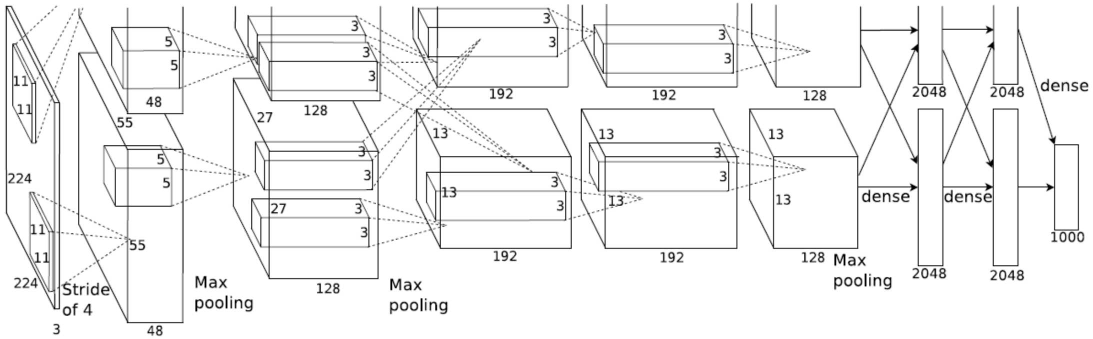
K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, ICCV 2015. [4.5%]

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015. [4.5%]

K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, CVPR, 2016 [3.6%]

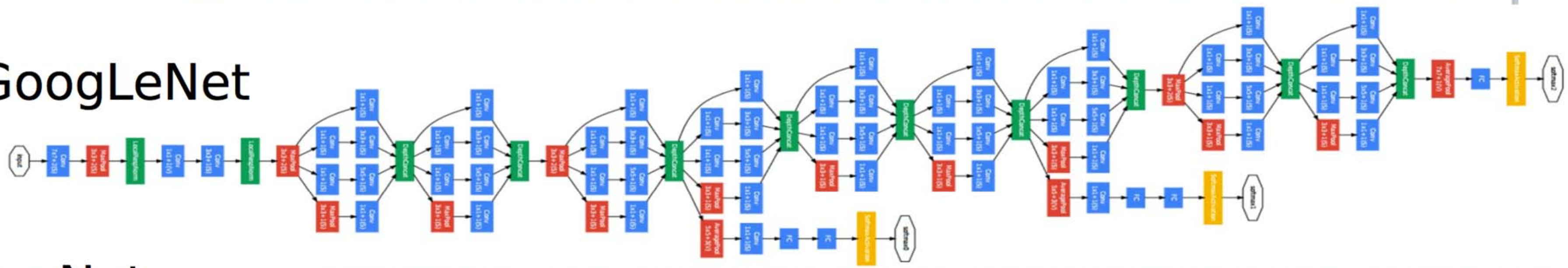
CNNs, 2010+

AlexNet



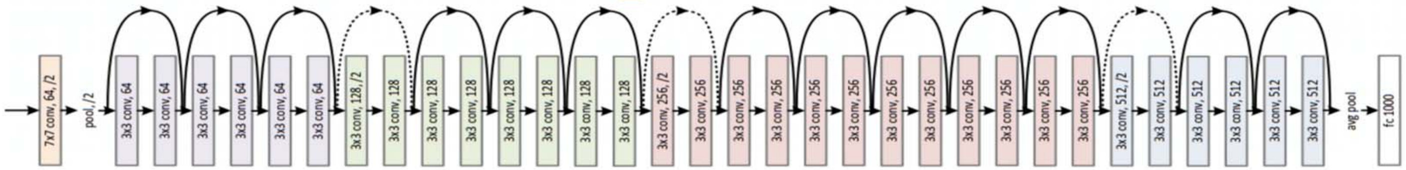
VGG

GoogLeNet



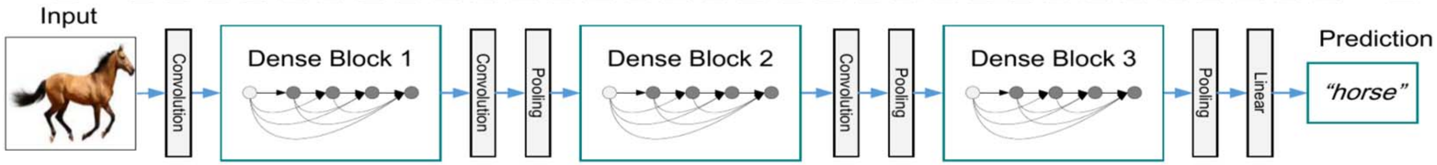
ResNet

[He et al. 2015]



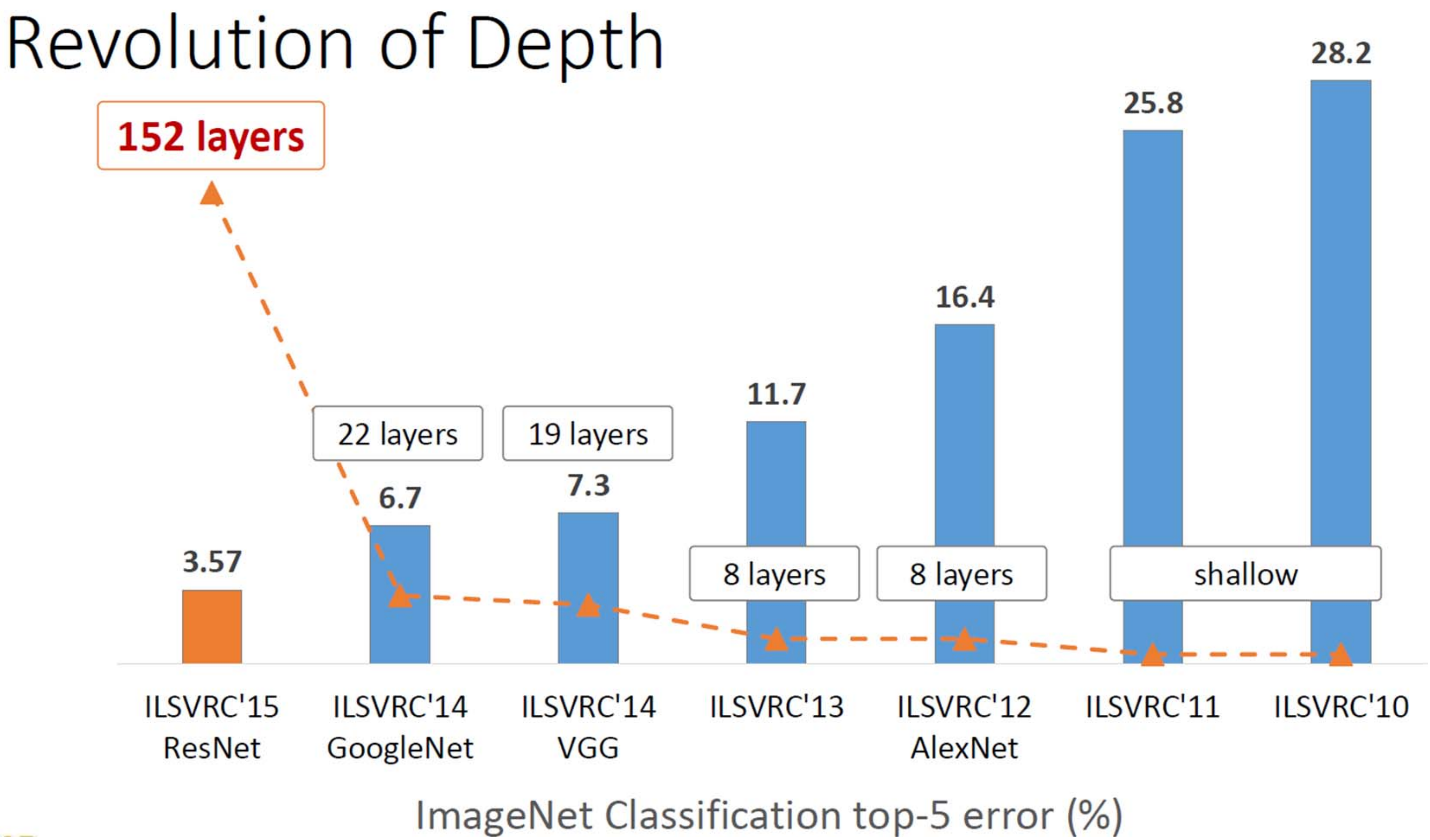
DenseNet

[Huang et al 2017]

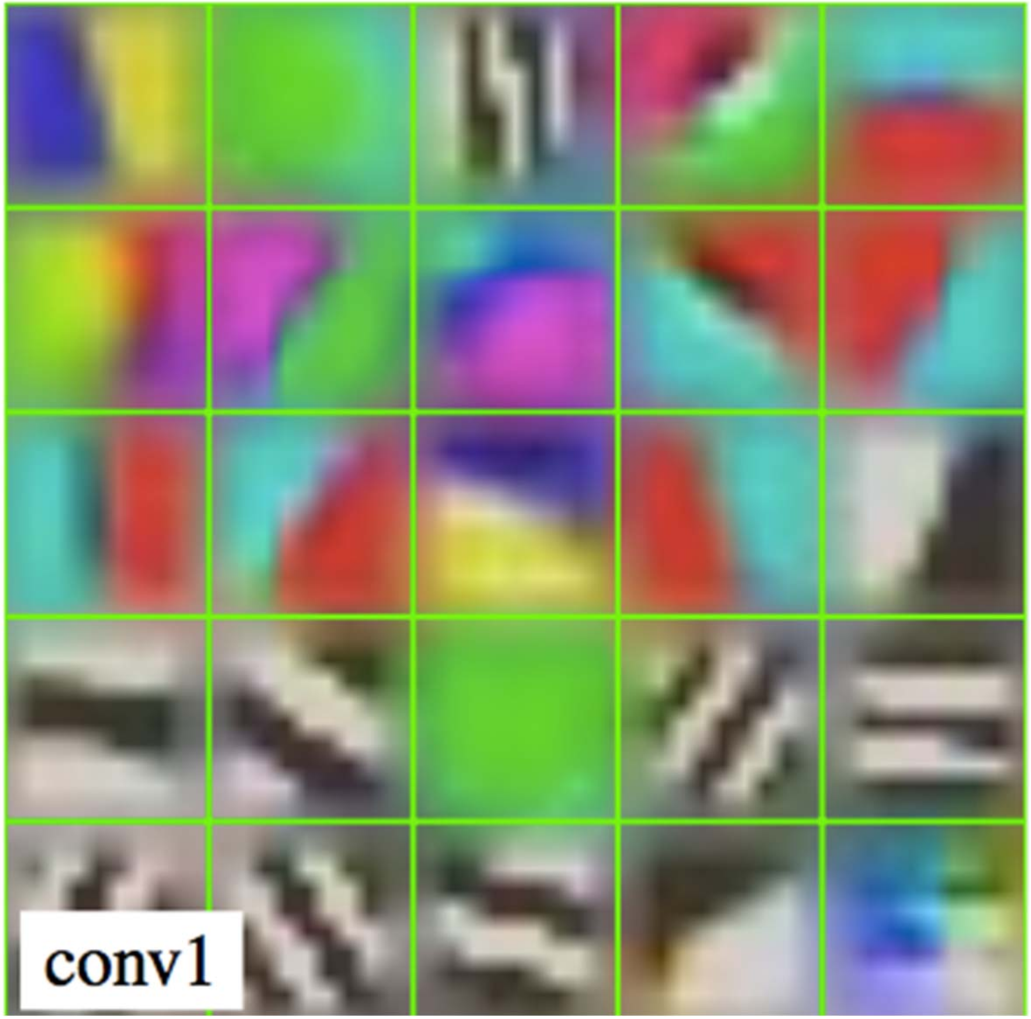


The deeper, the better

- Deeper networks can cover more complex problems
  - Increasingly large receptive field size
  - Increasingly non-linear patterns

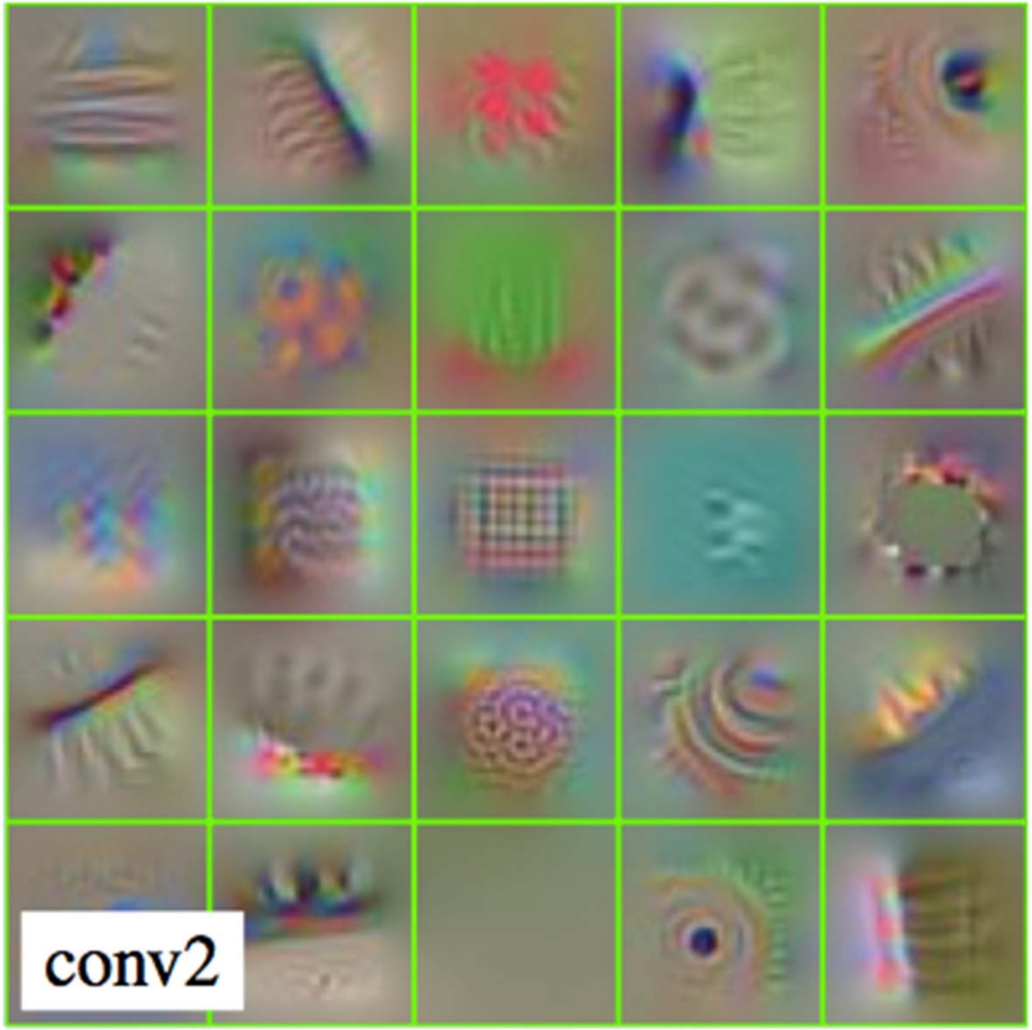


What is in the network?



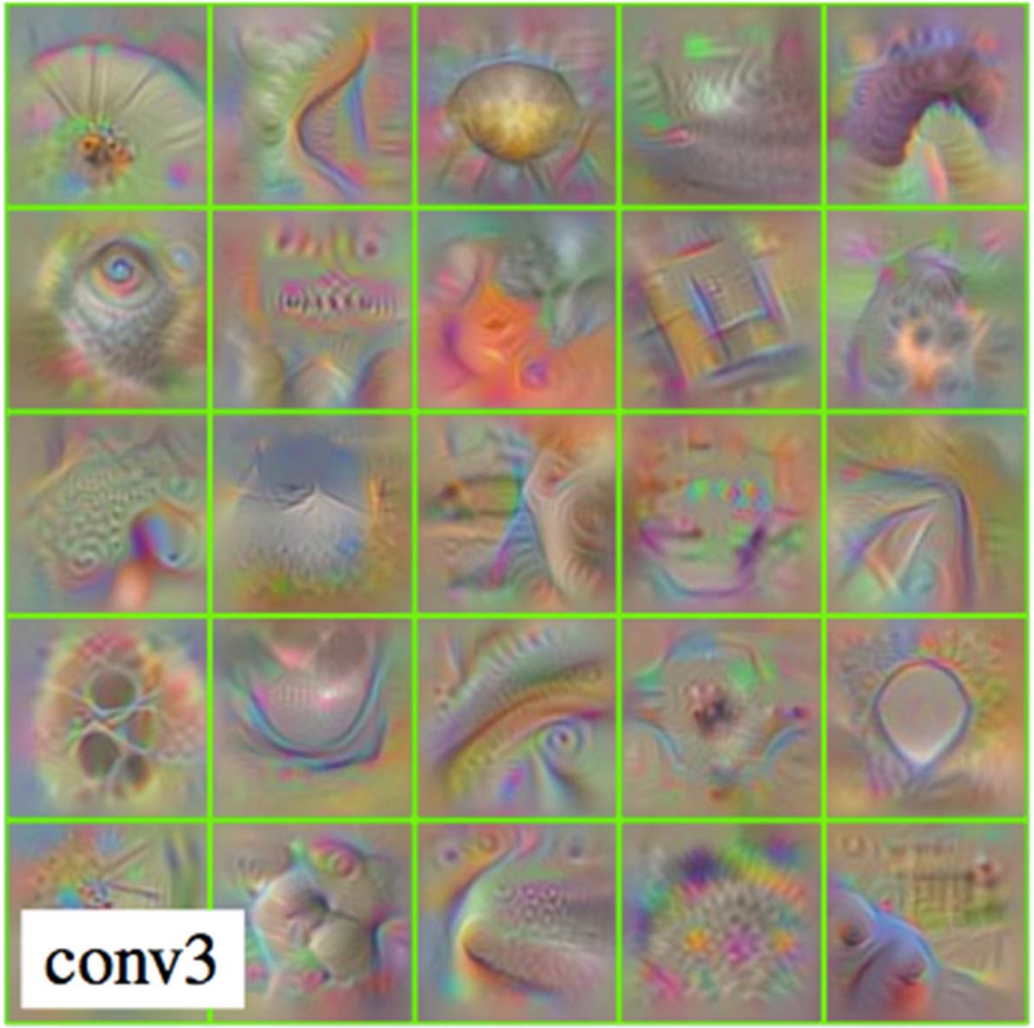
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



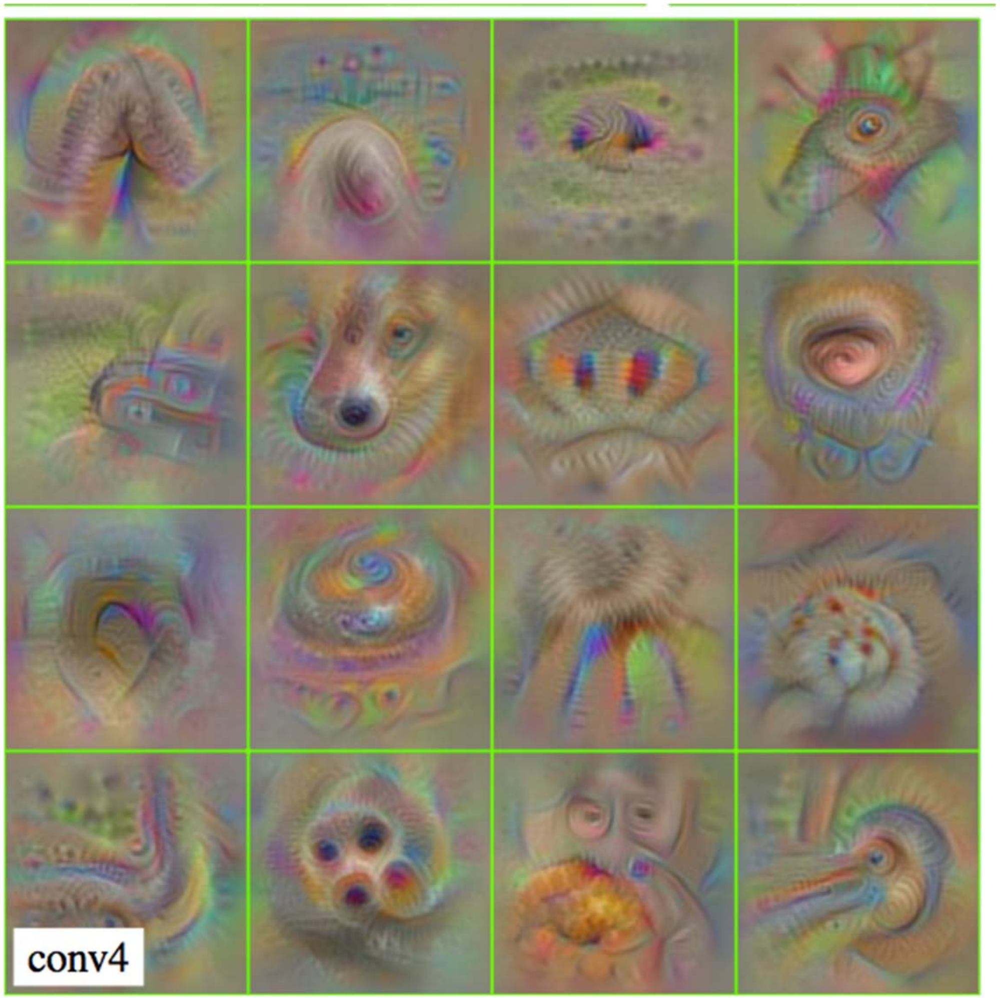
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



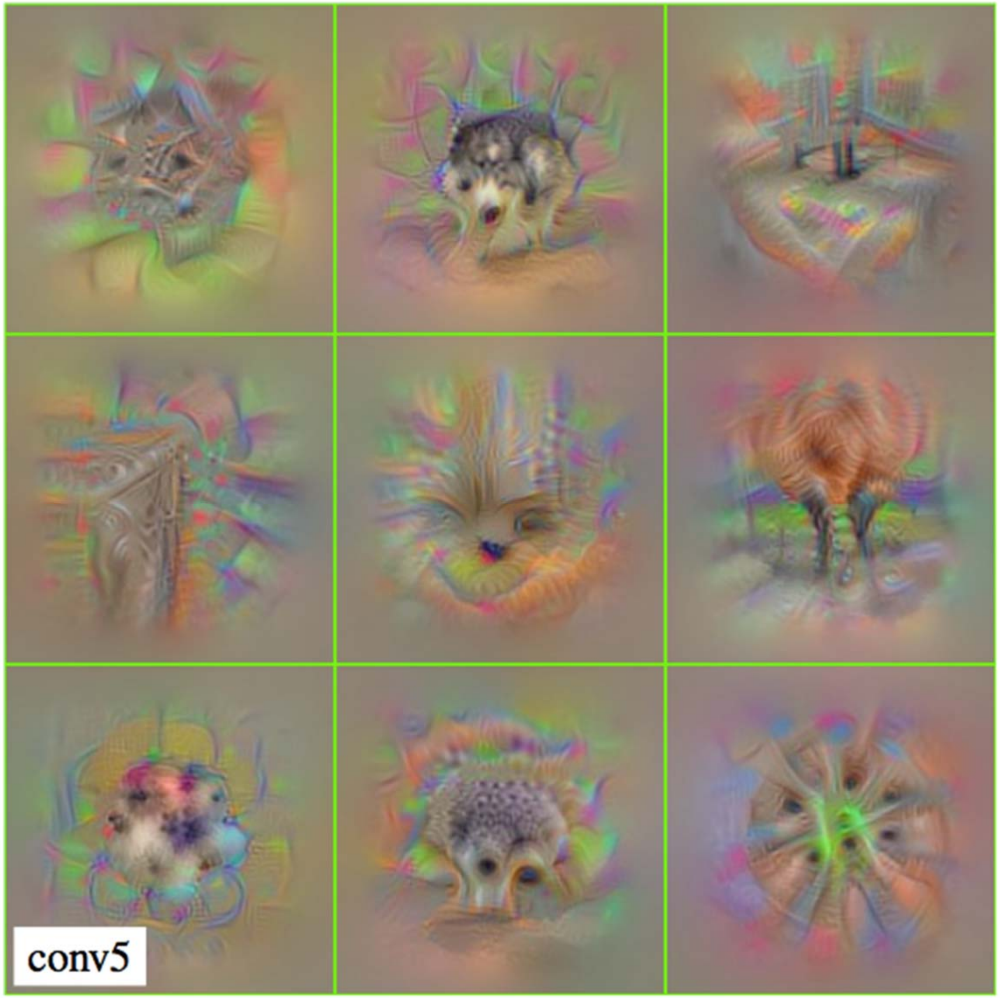
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



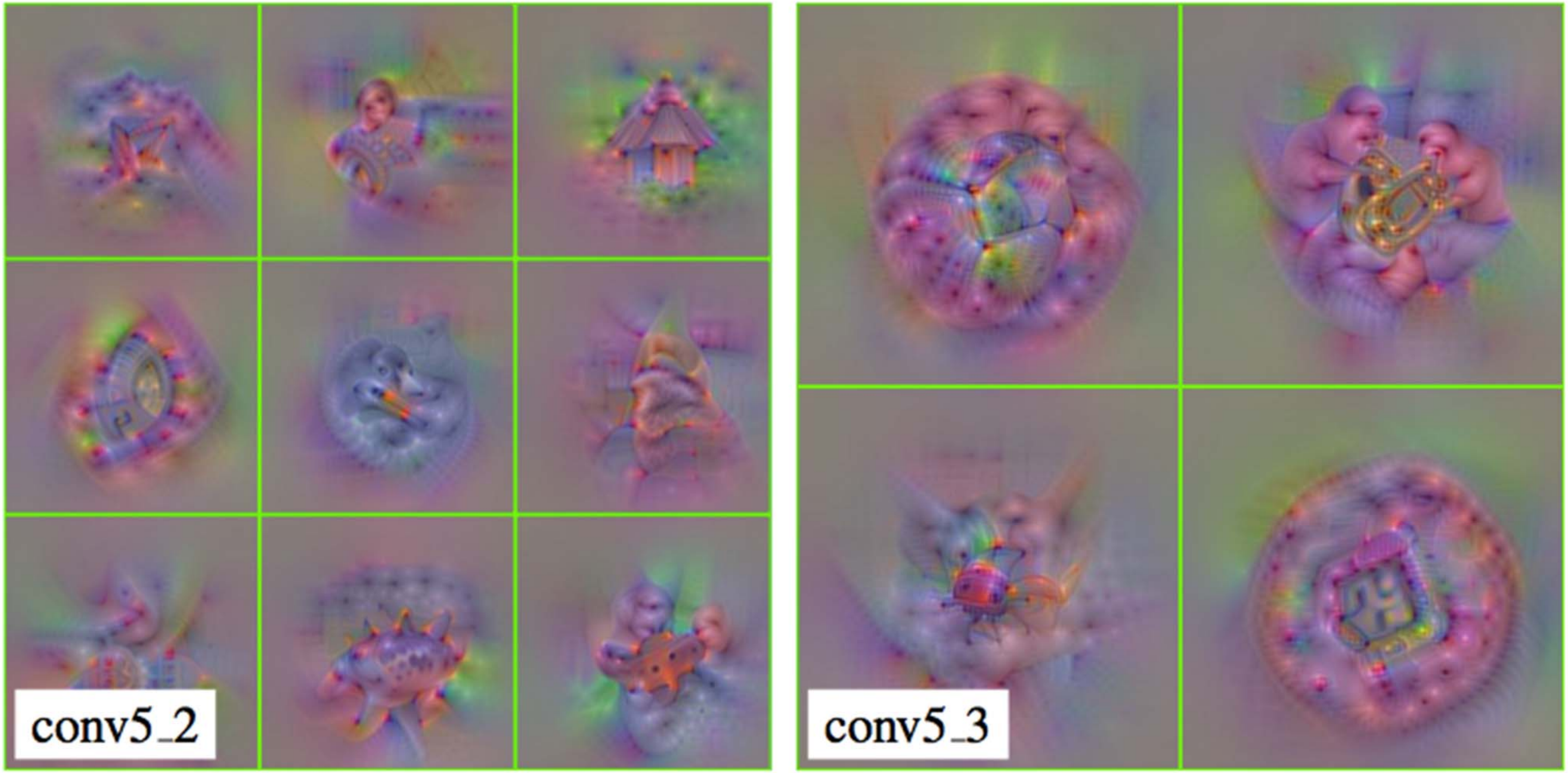
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



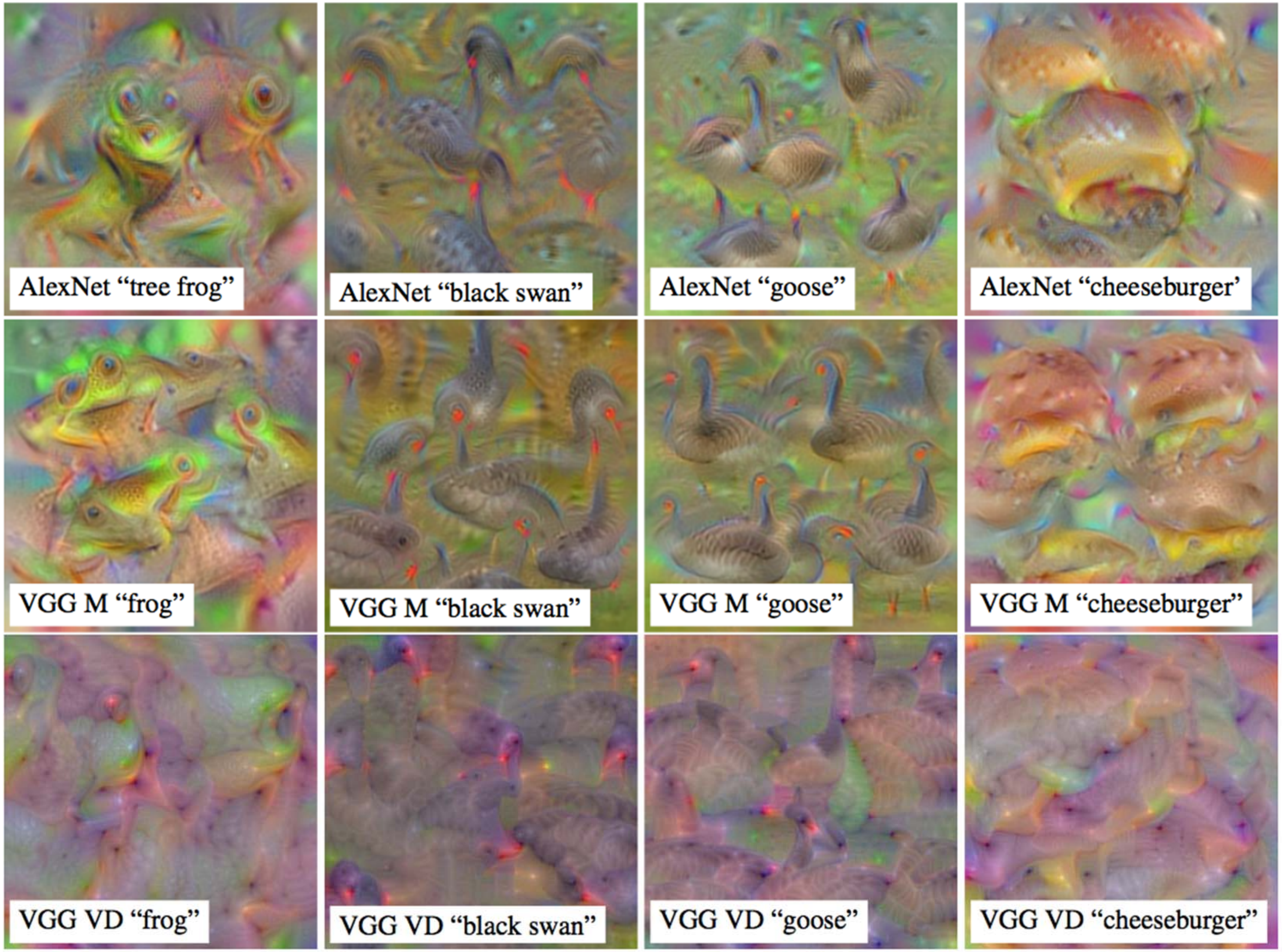
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

What is in the network?



Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

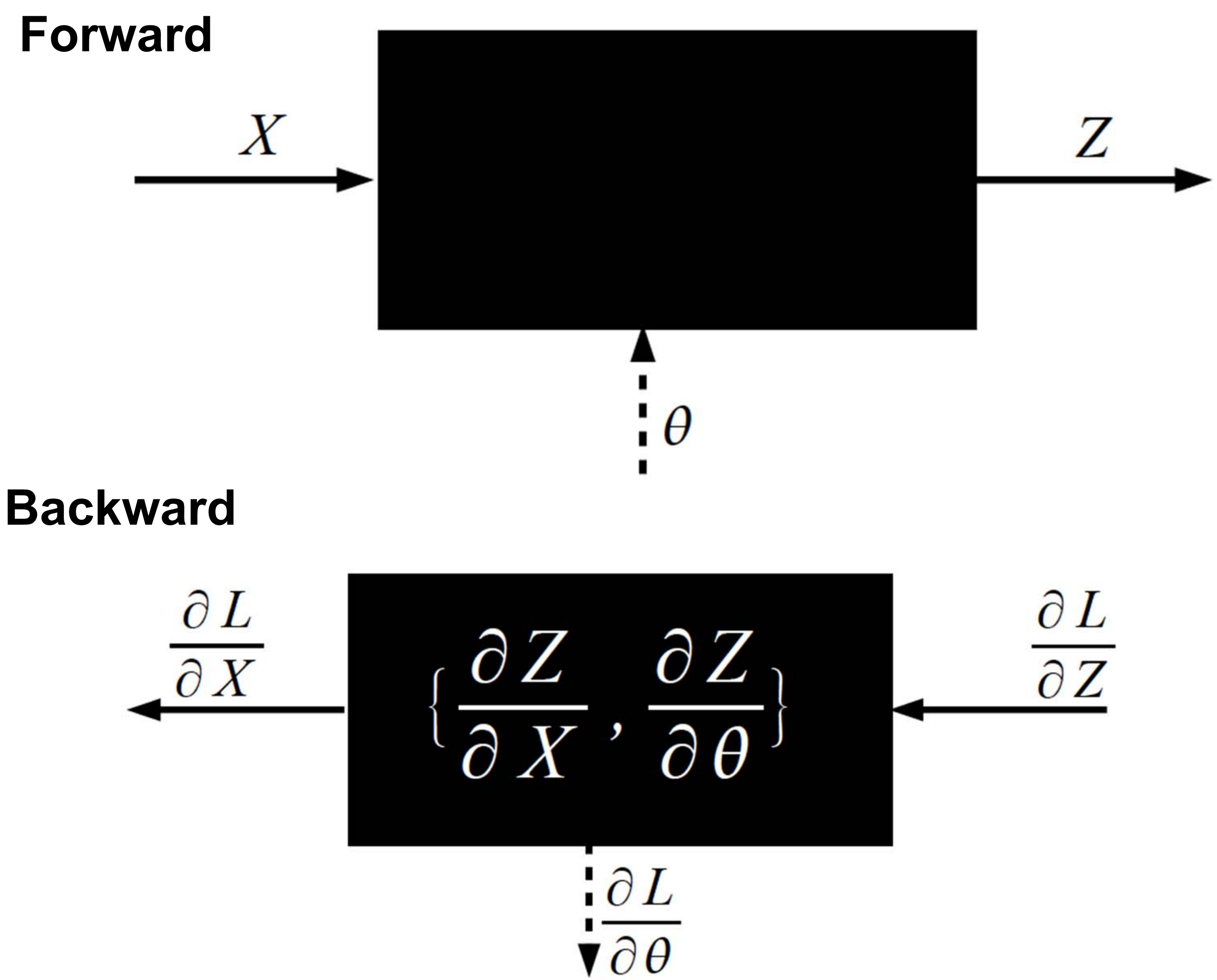
What is in the network?



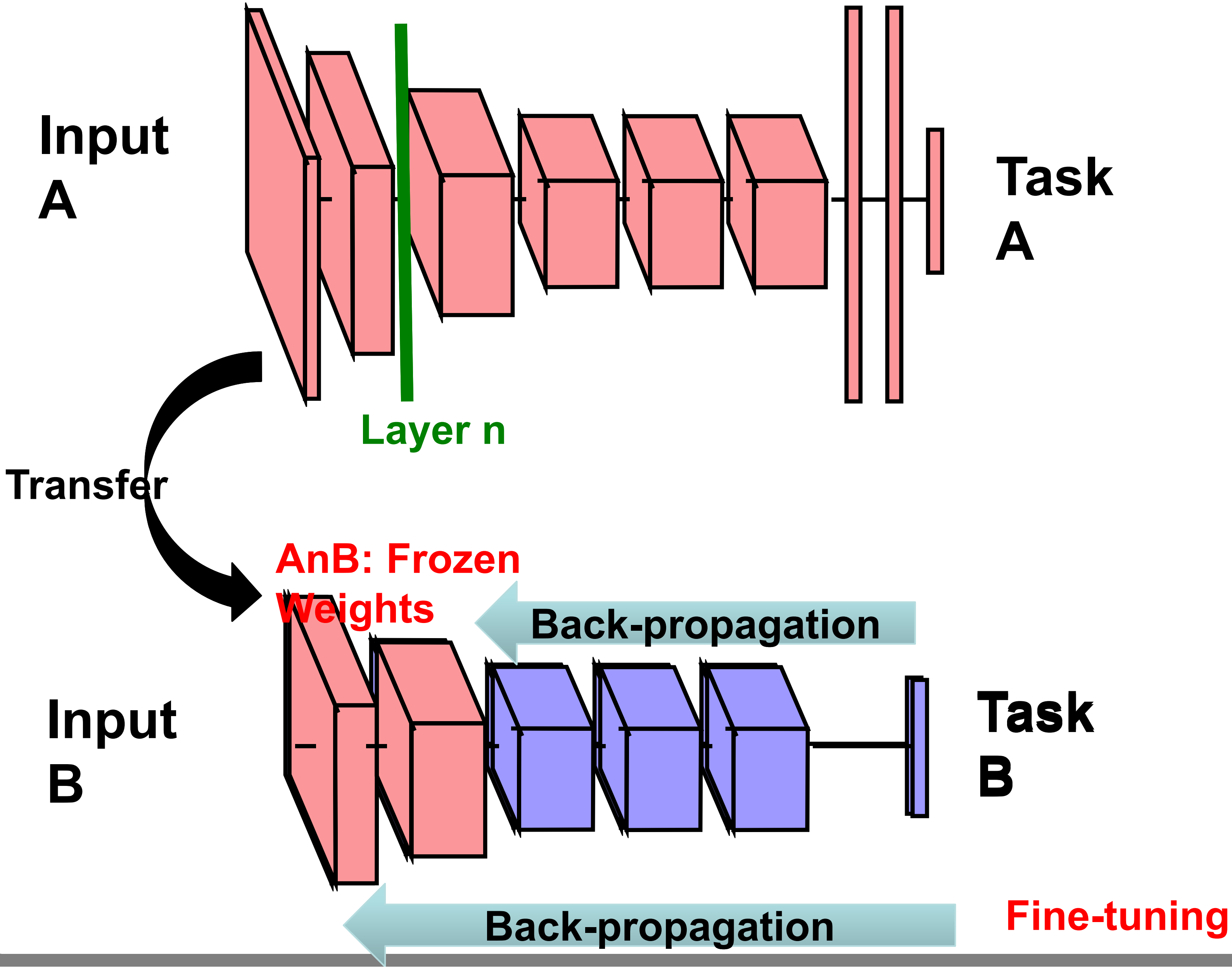
Visualizing deep convolutional neural networks using natural pre-images,
A. Mahindra and A. Vedaldi

Question: How did this happen?

Answer: All you need is gradients!

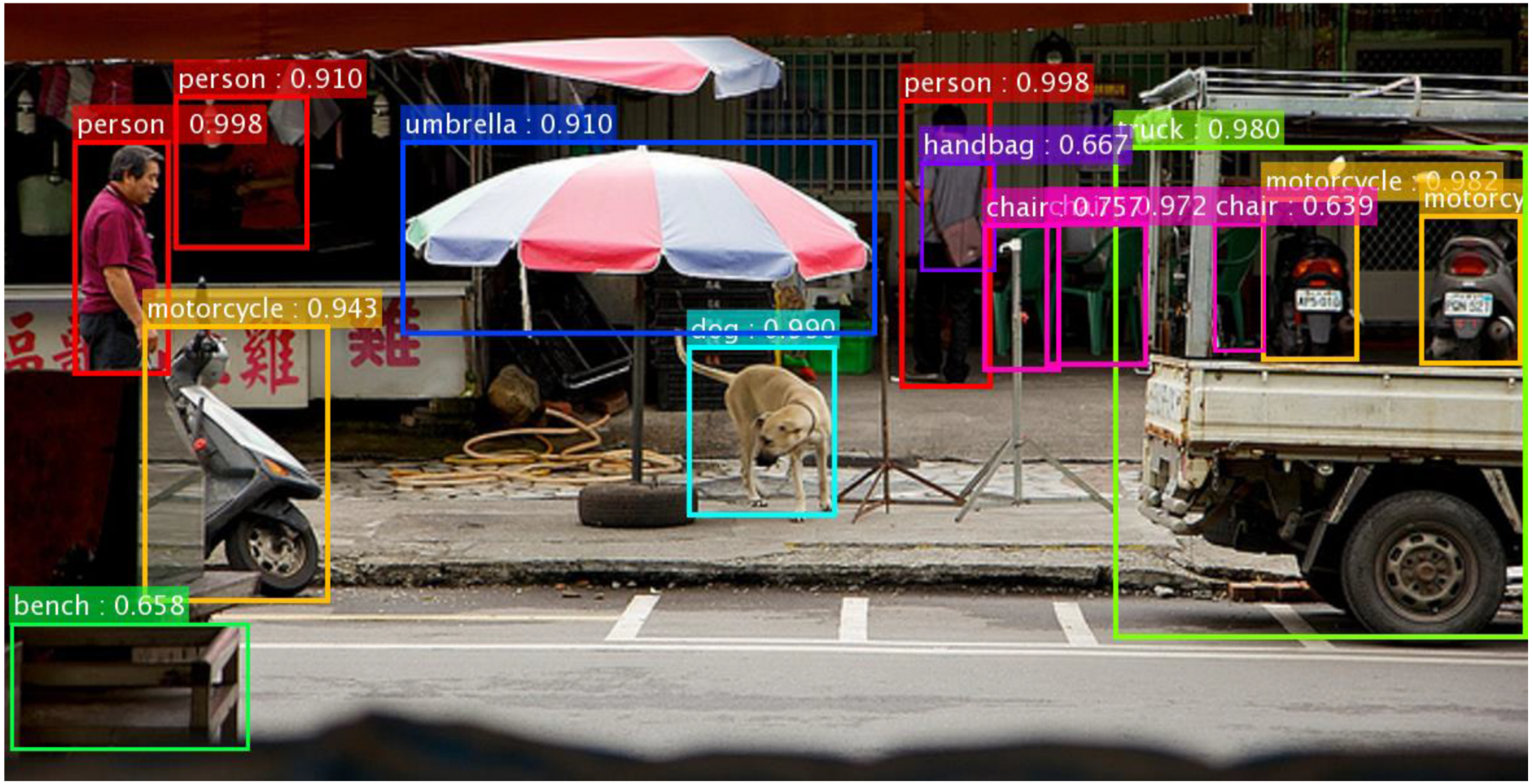


Transfer Learning



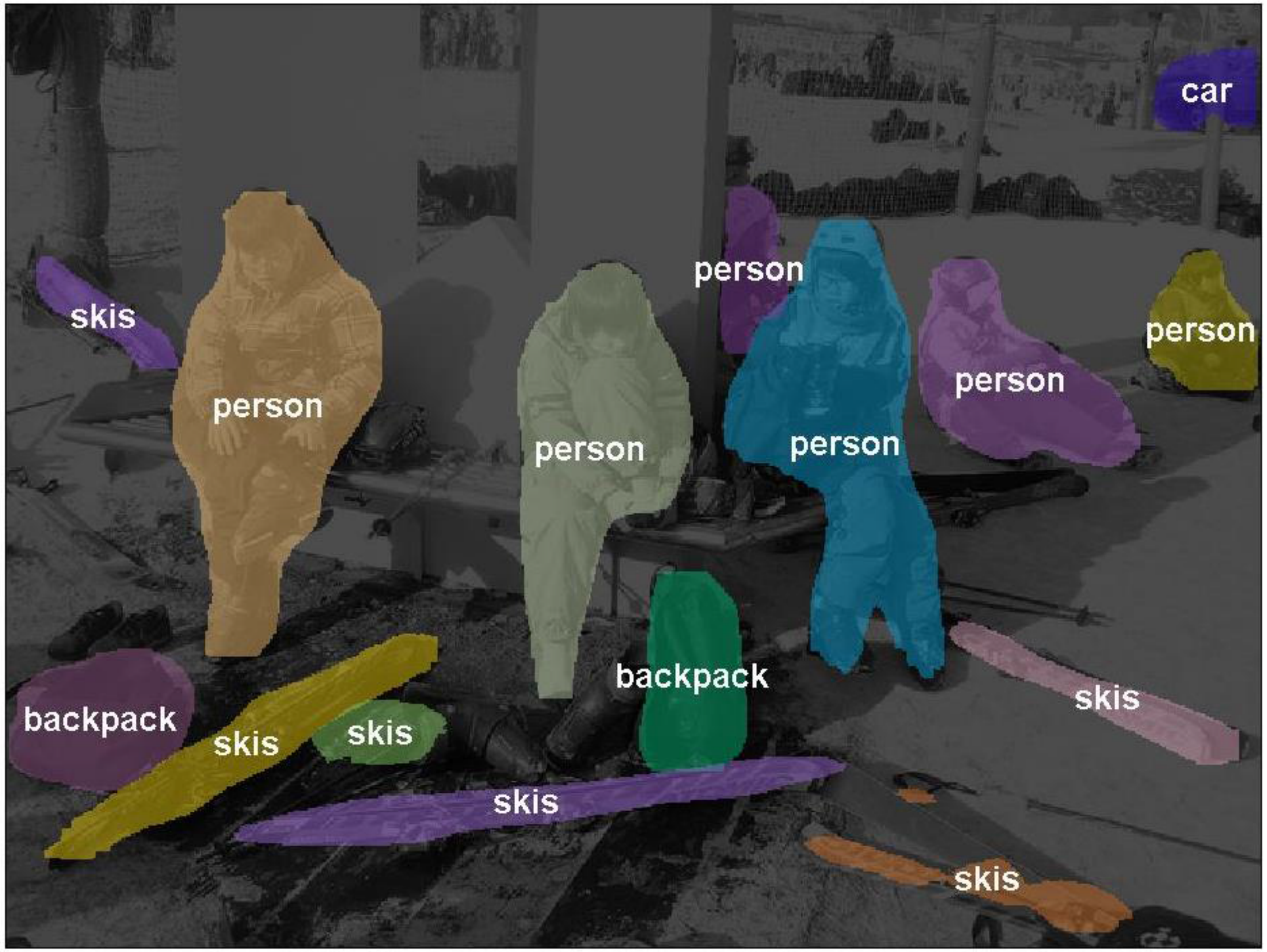
Qualitative Results

- Object detection
  - Faster R-CNN + ResNet



Qualitative Results

- Instance Segmentation



Deep learning, AD 1993

https://www.youtube.com/watch?v=FwFduRA_L6Q

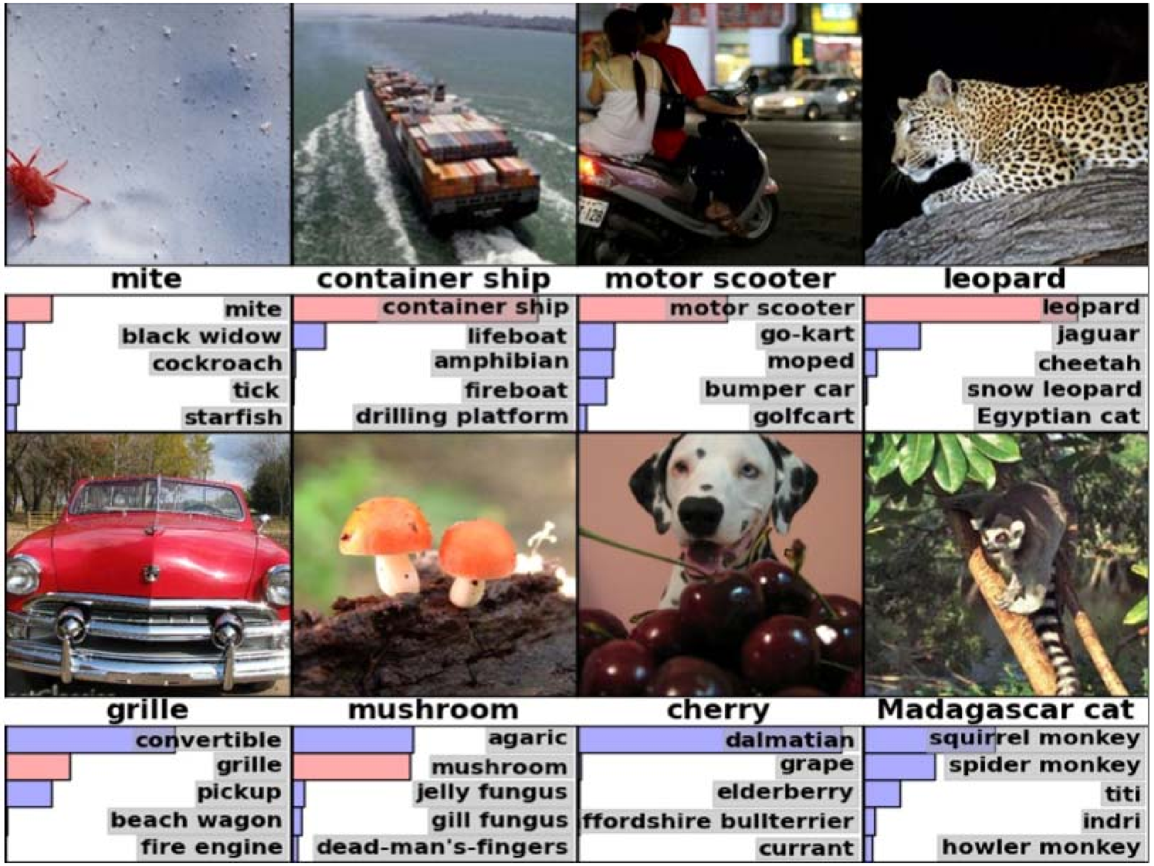
(LeNet 1, 1993)

http://yann.lecun.com/exdb/lenet/

(LeNet 5, 1998)

The 82 errors made by LeNet5

Deep Learning, AD 2016



Humans:5.4%

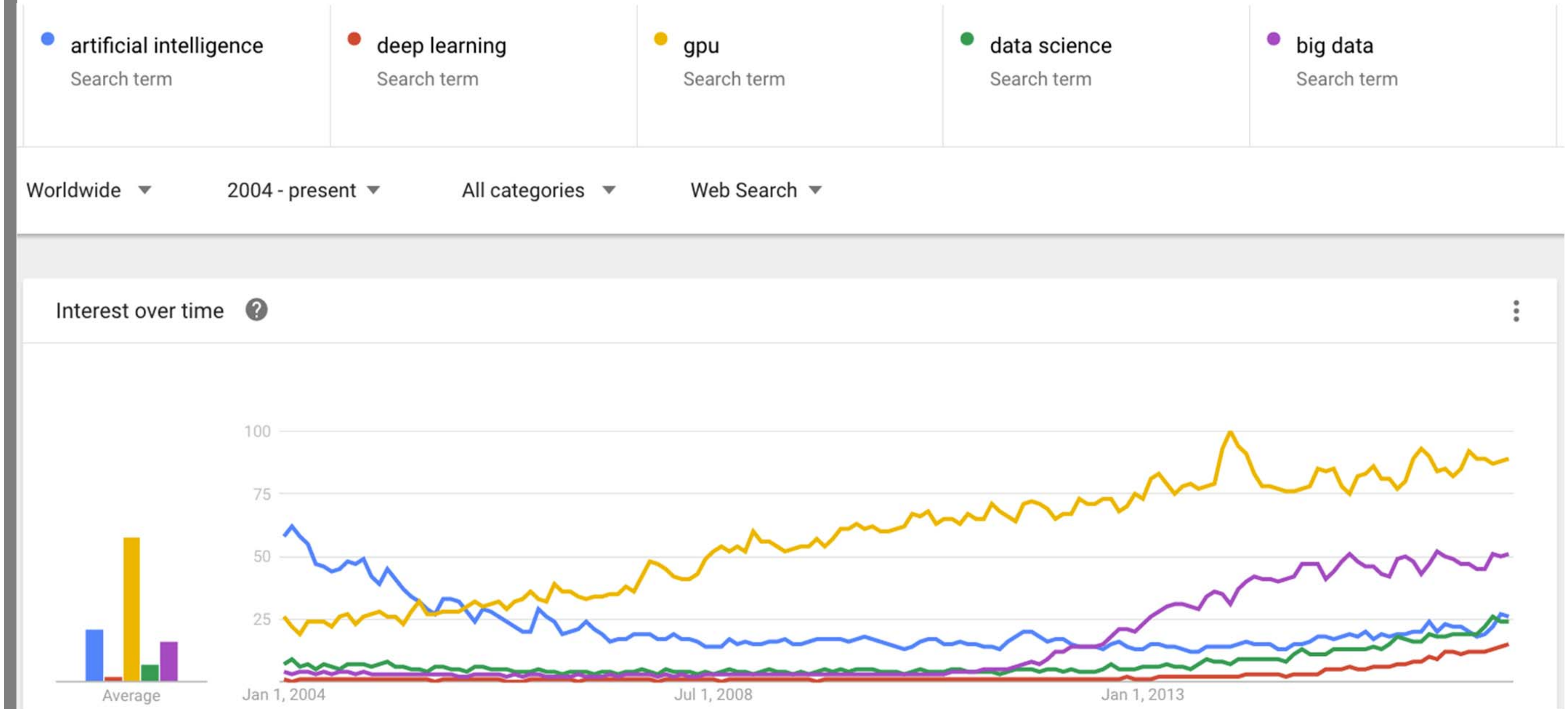
A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. ***NIPS*****13 [18%] (best shallow competitor: 36%)**

K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, http://arxiv.org/abs/1502.01852, 2015. [4.5%]

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, http://arxiv.org/abs/1502.03167, 2015. [4.5%]

K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, Arxiv, 2015 [3.6%]

What happened in between?



deep learning = neural networks (+ big data + GPUs)

+ a few more recent tricks!

Neural network training: old & new tricks

- Old: (80's)
  - Stochastic Gradient Descent, Momentum, "weight decay"
- New: (last 5-6 years)
  - Dropout
  - ReLUs
  - Batch Normalization
  - Residual Networks

Neural network training: old & new tricks

Old: (80's)

Stochastic Gradient Descent, Momentum, "weight decay"

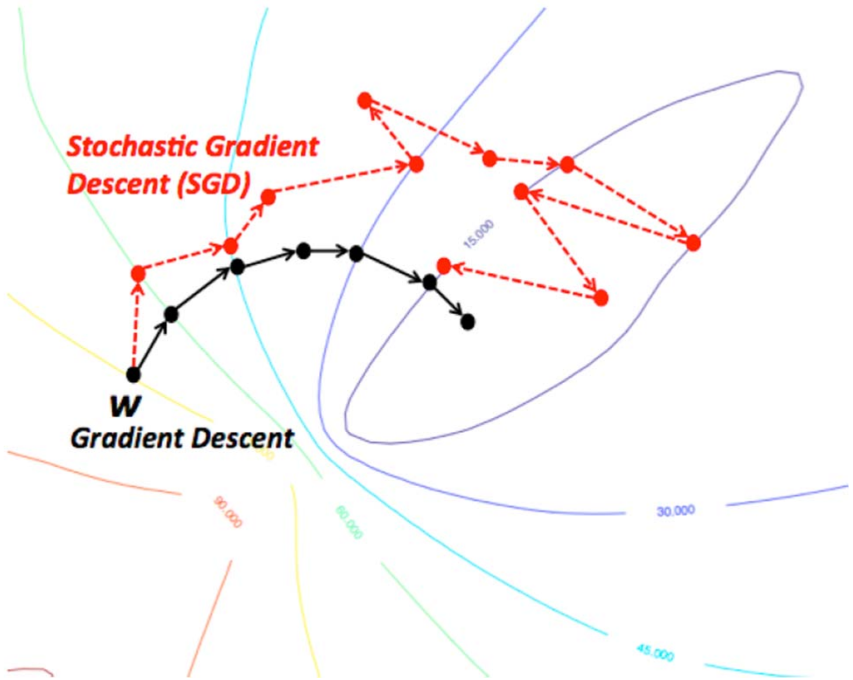
New: (last 5-6 years)

Dropout

ReLUs

Batch Normalization

Residual Networks



“Big data”: sometimes too big!

Challenges: fitting everything in memory

Keeping computational cost of training under control

“Large-Scale” Learning (checkout GI09: Applied Machine Learning)

Computer Vision Data: Big and Complicated

http://www.image-net.org/

IMAGENET

Overall

- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908

millions of images for academia

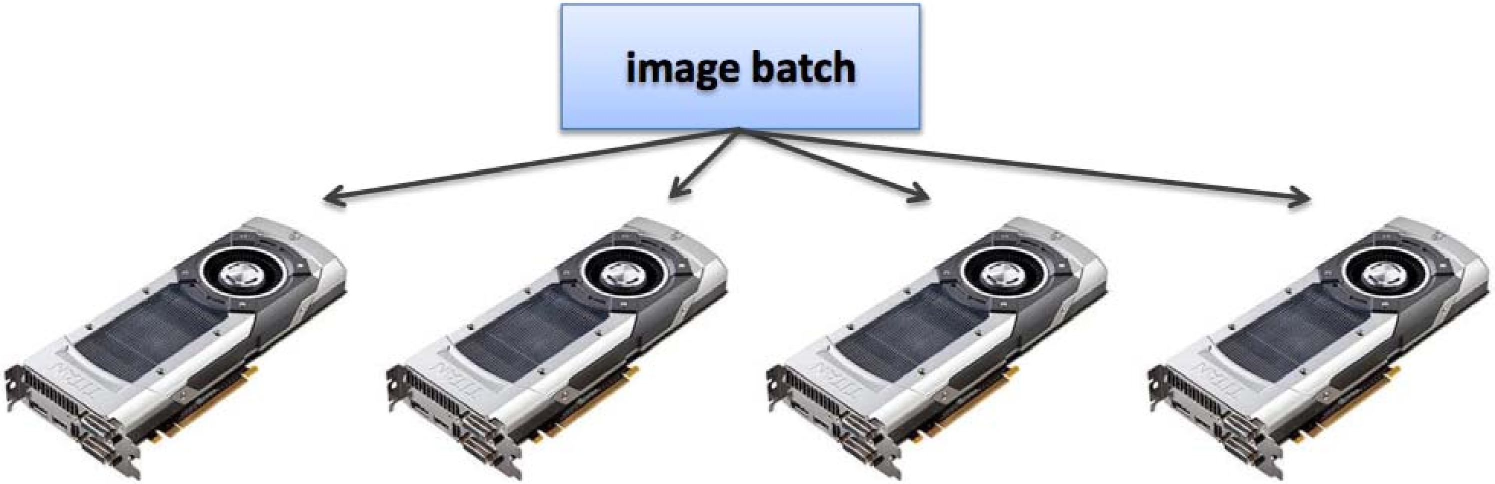
billions of images for industry

High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

Deng et. al, ImageNet: A Large-Scale Hierarchical Image Database, CVPR'09

How large is large?

- Heavily-modified Caffe C++ toolbox
- Multiple GPU support
  - 4 x NVIDIA Titan, off-the-shelf workstation
  - data parallelism for training and testing
  - ~3.75 times speed-up, 2-3 weeks for training



Simonyan & Zisserman, 2014

Training objective, multi-class case

Weeks 7-8:

One-hot label encoding: $\mathbf{y}^i = (0, 0, 1, 0)$

Likelihood of training sample: $(\mathbf{y}^i, \mathbf{x}^i)$

$$P(\mathbf{y}^i | \mathbf{x}^i; \mathbf{w}) = \prod_{c=1}^{C} \left(g_c(\mathbf{x}, \mathbf{W})\right)^{\mathbf{y}_c^i}$$

Optimization criterion:

$$L(\mathbf{W}) = -\sum_{i=1}^{N} \sum_{c=1}^{C} \mathbf{y}_c^i \log\left(g_c(\mathbf{x}, \mathbf{W})\right)$$

Parameter estimation: Gradient of L with respect to **W**

Training objective for classification

Likelihood of training sample's label: $P(\mathbf{y}^i|\mathbf{x}^i;\mathbf{w}) = \prod_{c=1}^{C} \left(g_c(\mathbf{x}^i, \mathbf{W})\right)^{\mathbf{y}_c^i}$

Cost function: $$L(\mathbf{W}) = -\sum_{i=1}^{N}\sum_{c=1}^{C} \mathbf{y}_c^i \log\left(g_c(\mathbf{x}^i, \mathbf{W})\right)$$

Normalize: $$L'(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left[-\sum_{c=1}^{C} \mathbf{y}_c^i \log\left(g_c(\mathbf{x}^i, \mathbf{W})\right)\right]$$

$$= \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) \qquad \hat{\mathbf{y}}^i = \begin{bmatrix} g_1(\mathbf{x}, \mathbf{W}) \\ \vdots \\ g_C(\mathbf{x}, \mathbf{W}) \end{bmatrix}$$

Add regularization: $$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_{l} \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

for all layers (l), and all input (k) –output (m) connection weights

Training objective for classification

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_{l} \lambda_l \sum_{k,m} (\mathbf{W}^l_{k,m})^2$$

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N} \boxed{\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Back-prop for i-th example

If N=10^6 , we will need to run back-prop 10^6 times to update **W** once!

Stochastic Gradient Descent (SGD)

Gradient:

Batch: [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy ('Stochastic') Gradient:

Minibatch: B elements

b(1), b(2),…, b(B): sampled from [1,N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \frac{1}{B}\sum_{i=1}^{B} \frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Epoch: N samples, N/B batches

Regularization in SGD: weight decay

Gradient:

Batch: [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy ('Stochastic') Gradient:

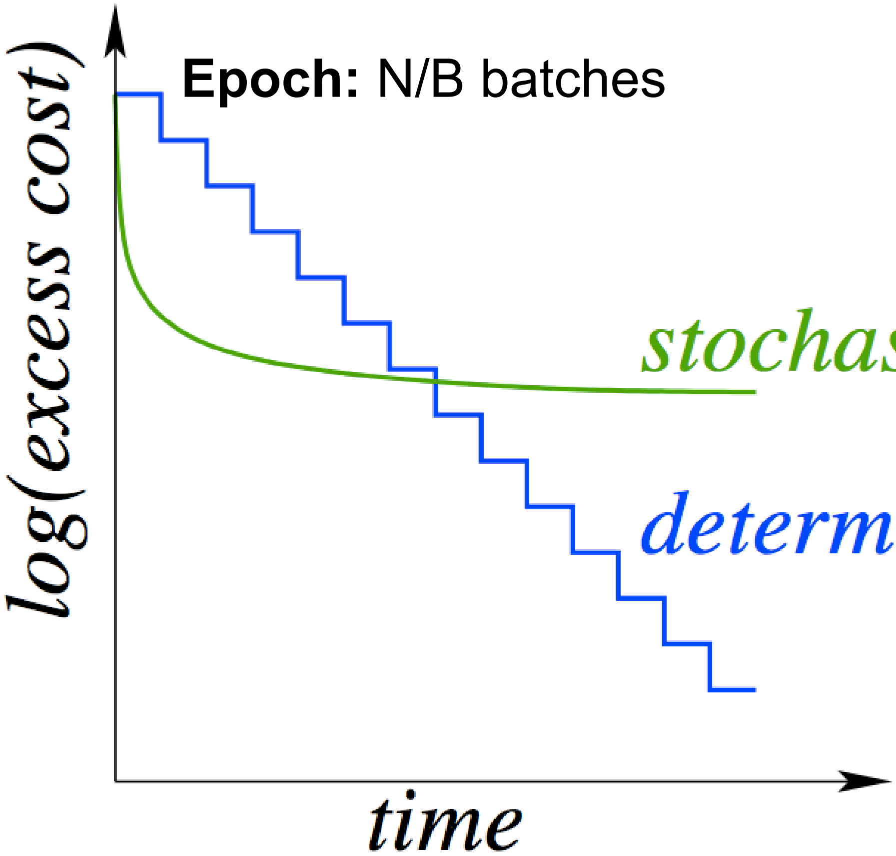
Minibatch: B elements

b(1), b(2),…, b(B): sampled from [1,N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \underbrace{\boxed{\frac{1}{B}\sum_{i=1}^{B}\frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}}}}_{\text{Back-prop on minibatch}} + \underbrace{\boxed{2\lambda_l \mathbf{W}^l_{k,m}}}_{\text{"Weight decay"}}$$

Epoch: N samples, N/B batches

Is Stochastic Gradient Descent faster?

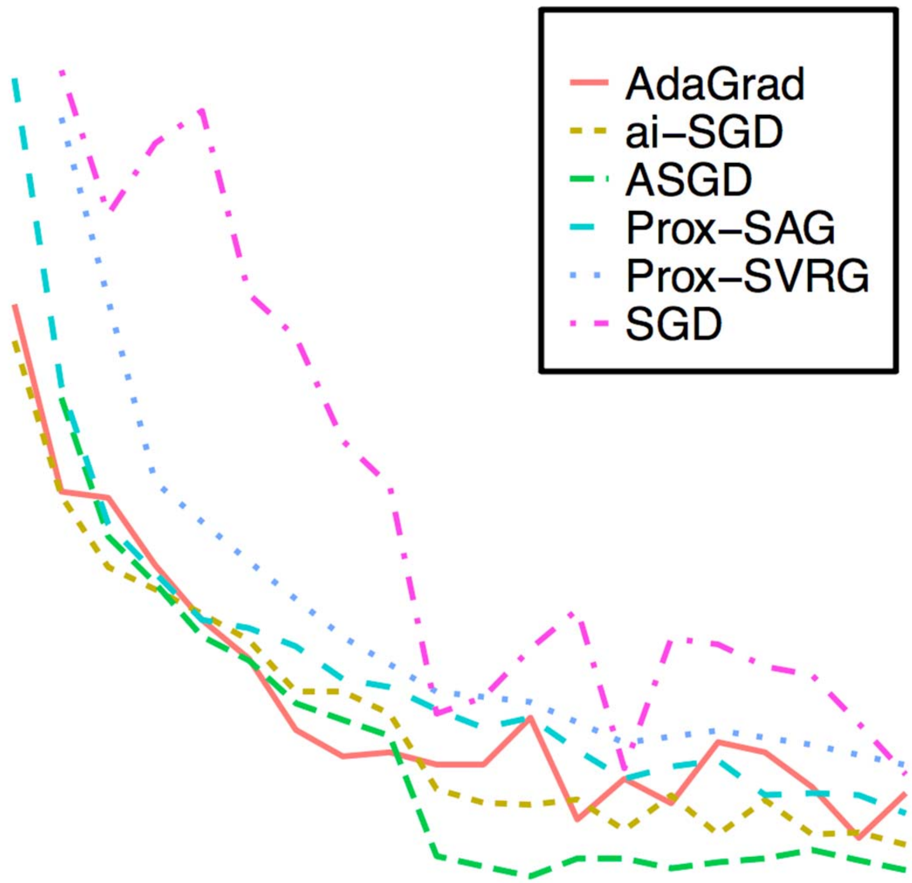


In one epoch SGD performs N/B more updates than GD!

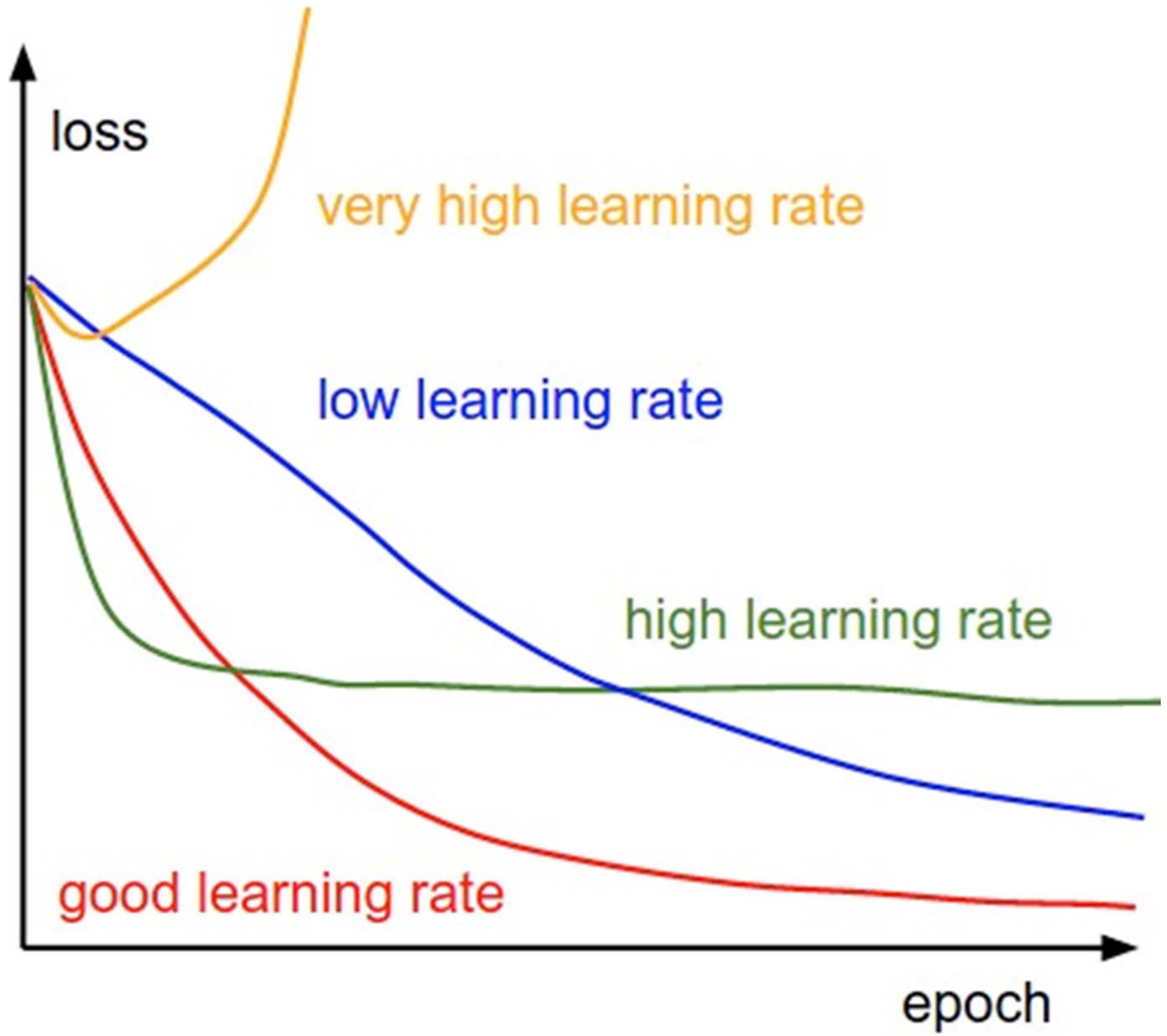
N: 10^6,B = 10: 100000 x more steps

Current research:
best of both worlds?
2nd order (Newton-Raphson-like) methods?

“Right” SGD: active research topic

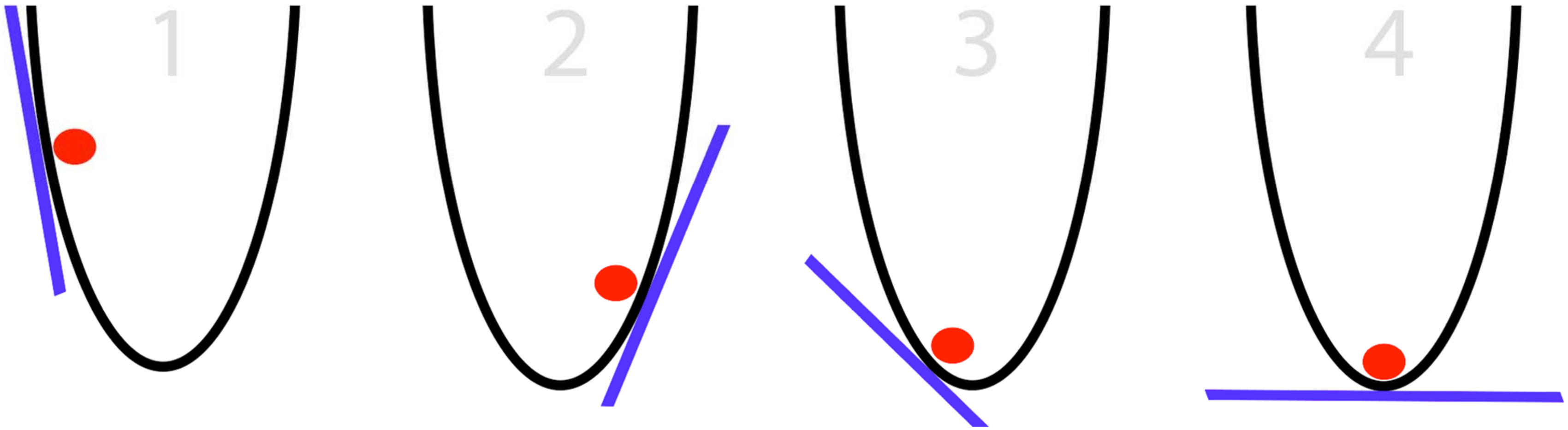


Learning rate

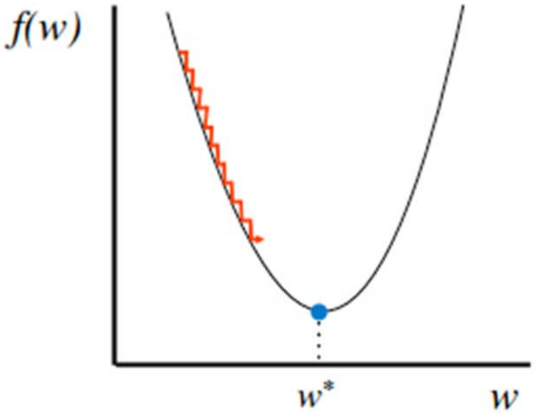


$$\mathbf{W}_{t+1} = \mathbf{W}_t - \boxed{\epsilon}\nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

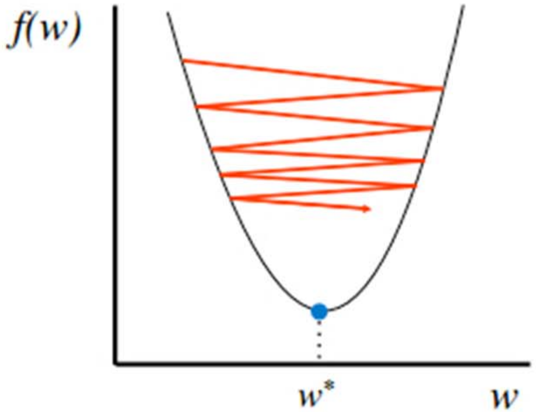
Gradient Descent



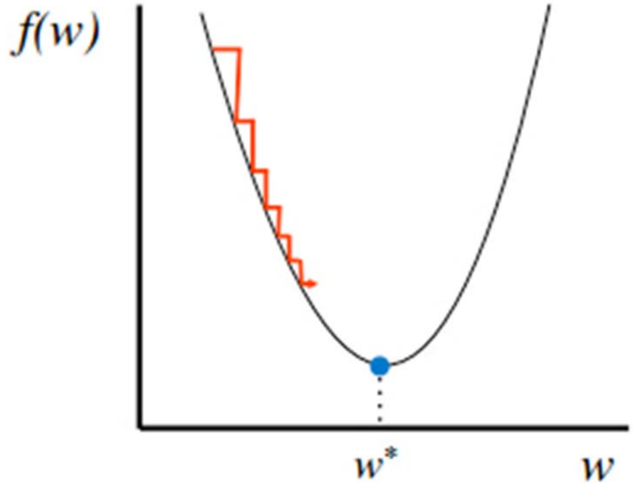
(S)GD with adaptable stepsize



Too small: converge very slowly



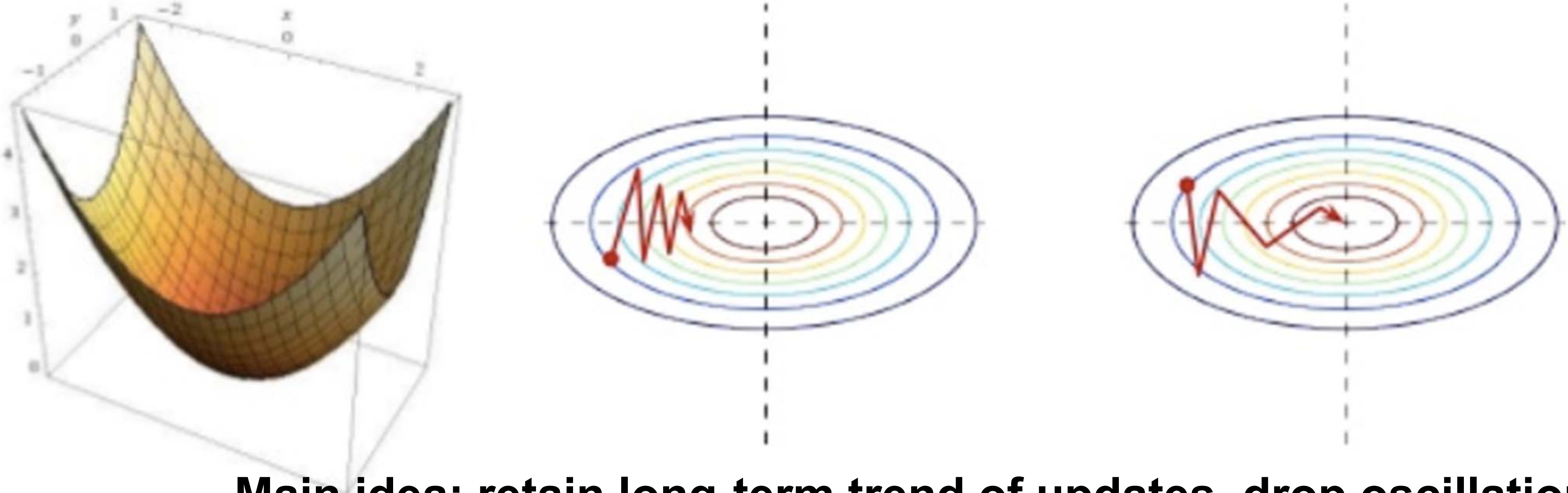
Too big: overshoot and even diverge



Reduce size over time

e.g. $$\epsilon_t = \frac{c}{t}$$

(S)GD with momentum



Main idea: retain long-term trend of updates, drop oscillations

(S)GD

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W})$$

(S)GD + momentum

$$\mathbf{V}_{t+1} = \mu \mathbf{V}_t + (1-\mu) \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \mathbf{V}_{t+1}$$

Neural network training: old & new tricks

Old: (80's)

- **Stochastic Gradient Descent, Momentum, "weight decay"**

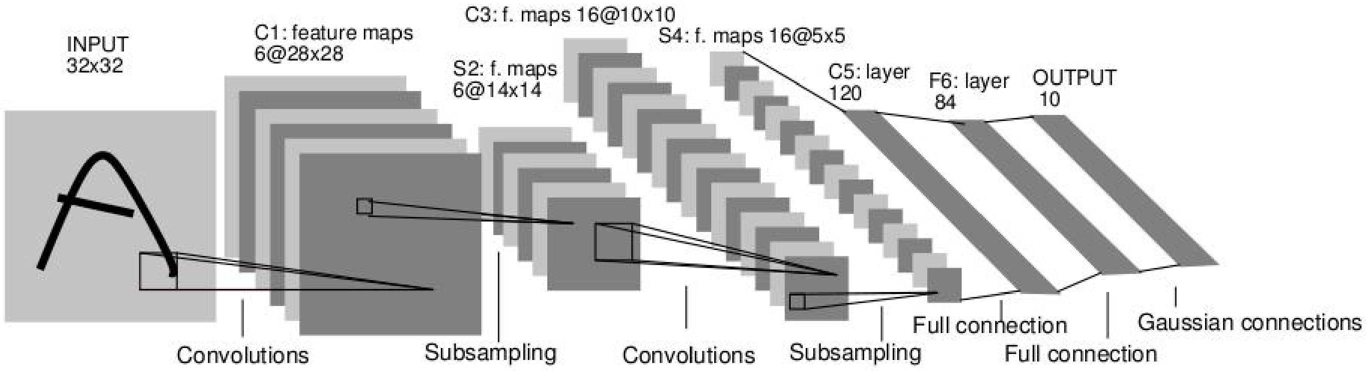
New: (last 5-6 years)

- **Dropout**
- **ReLUs**
- **Batch Normalization**
- **Residual Networks**

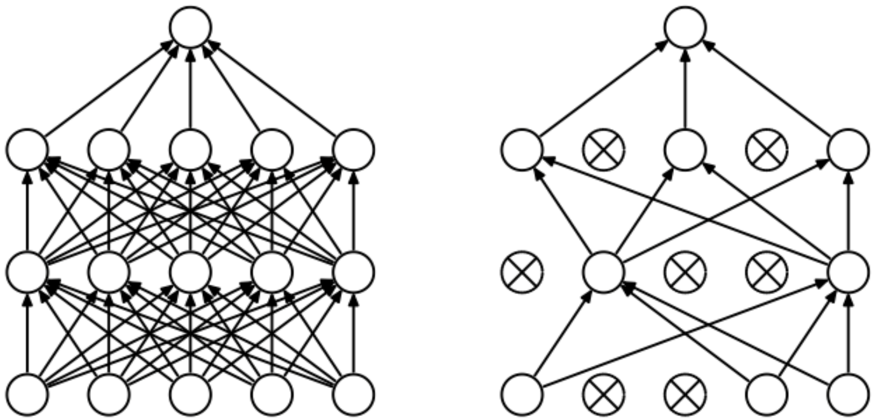
Regularization in Deep Learning

Weight Decay: just before

Convolutional Networks: last week

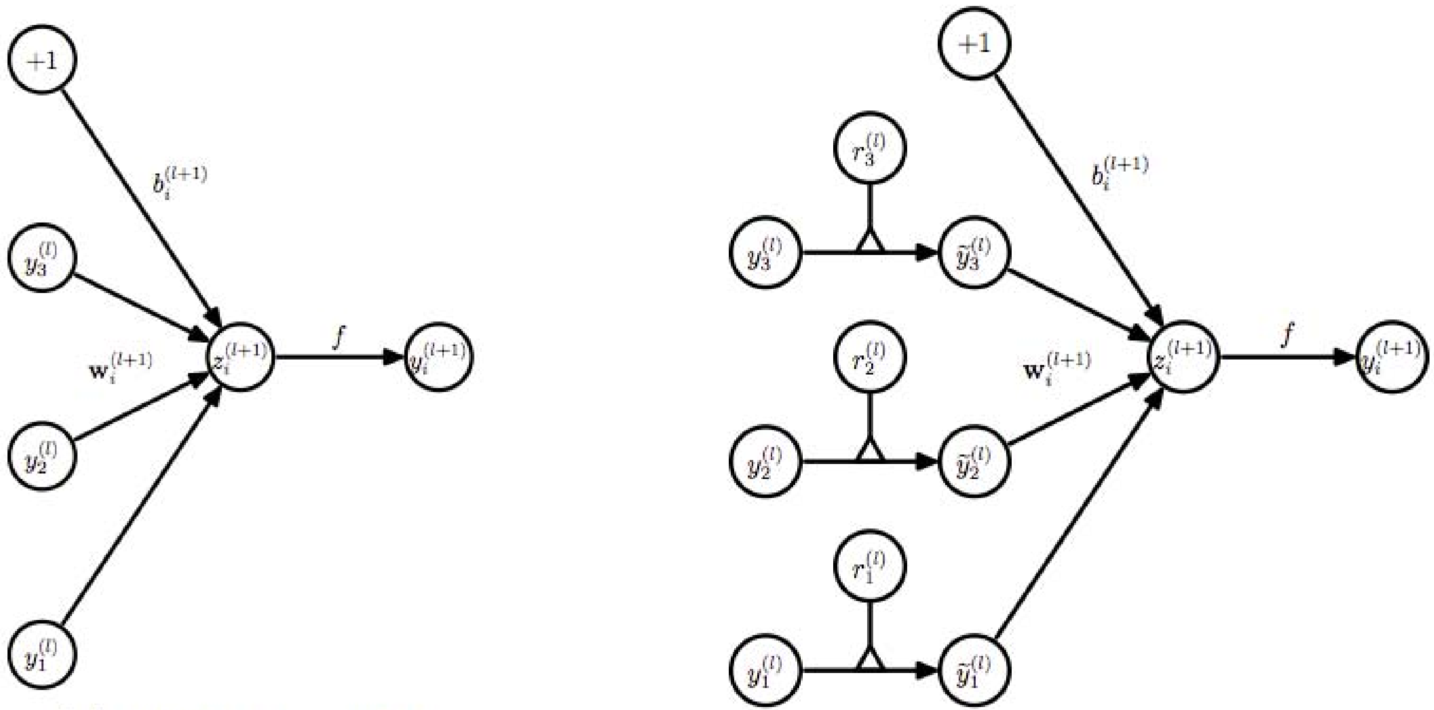


Dropout: now



(a) Standard Neural Net (b) After applying dropout.

Dropout



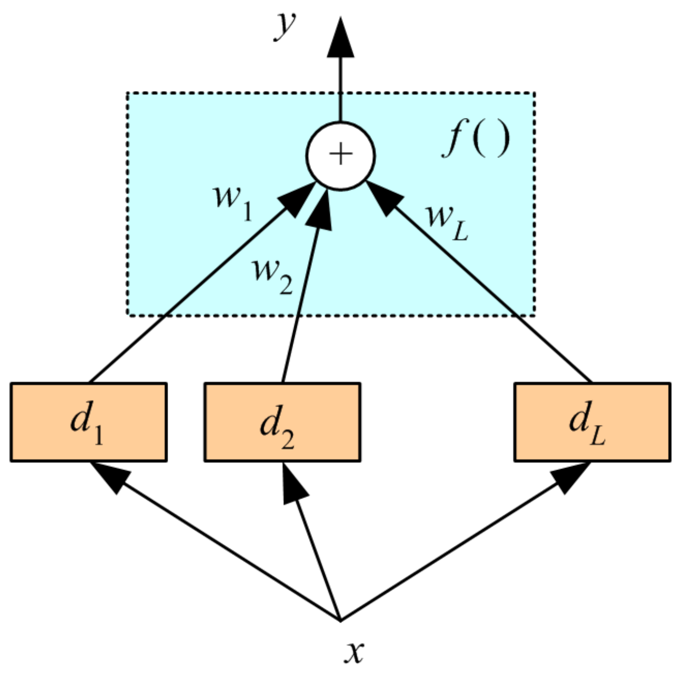
(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

Voting Methods

- Give up idea of building `the' classifier
- Generate a group of base-learners which has higher accuracy when combined
- Main tasks
  - Generating the learners
  - Combining them



Why should this work? Week 5 Lecture

- Committee of M predictors for target output

$$y_{COM}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$

- Output: true value + error $y(\mathbf{x}) = h(\mathbf{x}) + \epsilon(\mathbf{x})$

- Expected sum of squares error for m-th expert:

$$\mathbb{E}_{\mathbf{x}}[(y_m(\mathbf{x}) - h(\mathbf{x}))^2] = \mathbb{E}_{\mathbf{x}}[e_m(\mathbf{x})^2]$$

- Average error of individual members:

$$\mathbb{E}_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

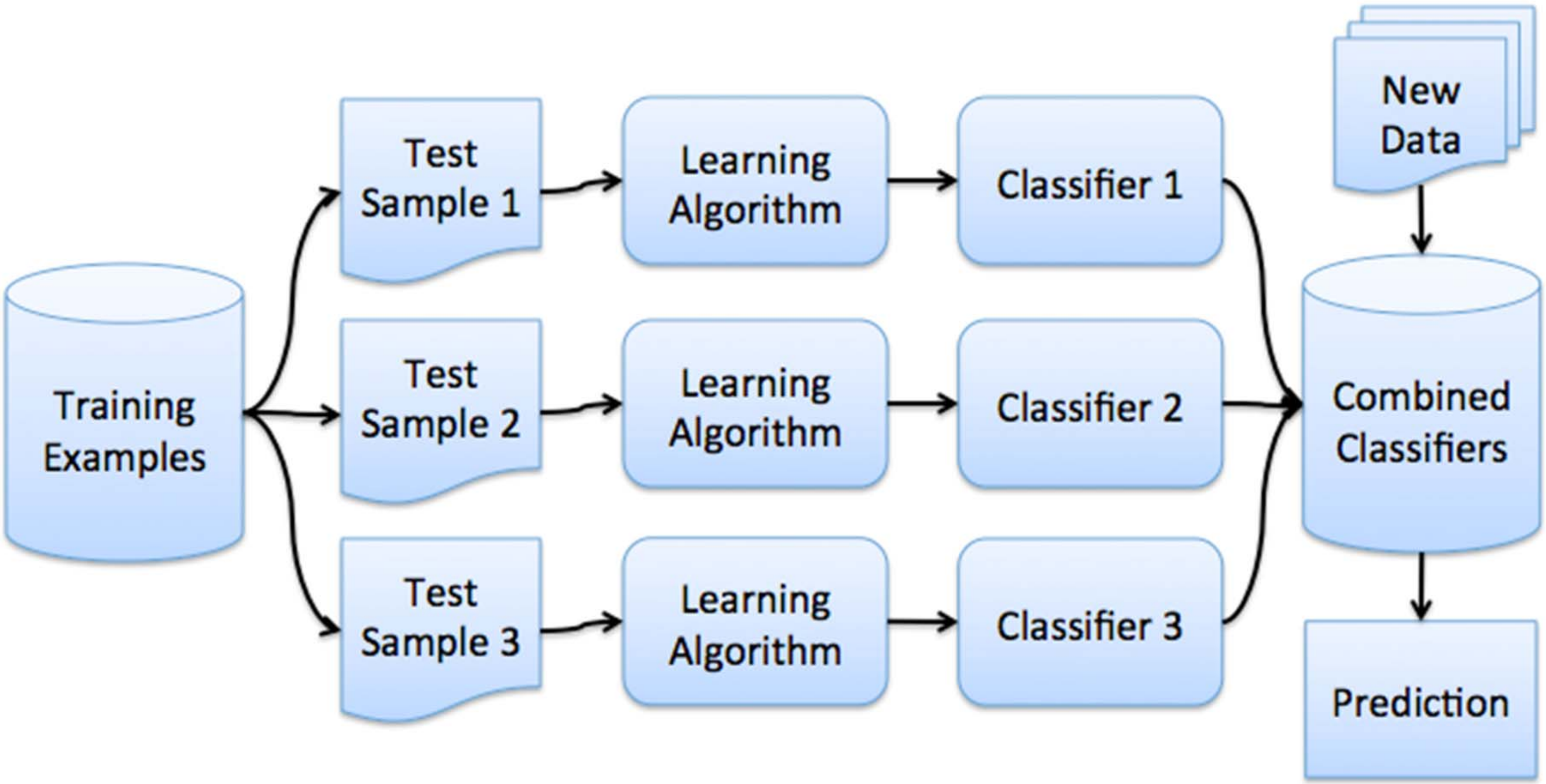
- Average error of committee:

$$\mathbb{E}_{COM} = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x})\right\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \epsilon_m(\mathbf{x})\right\}^2\right]$$

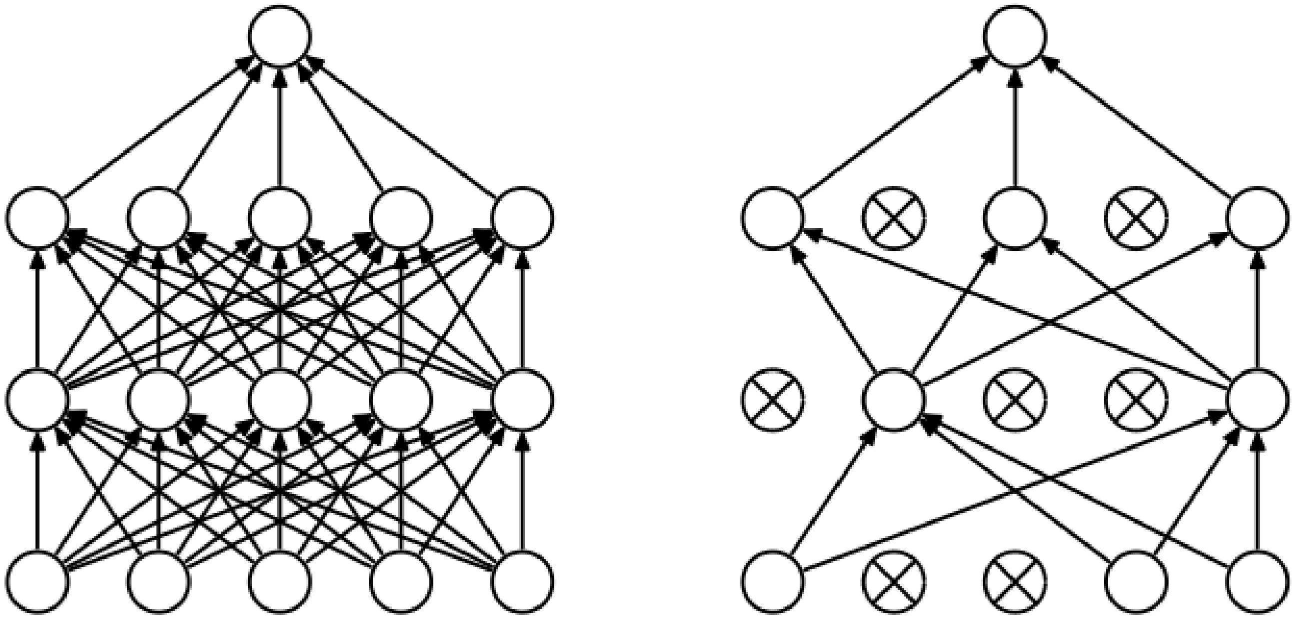
- **If** committee members have uncorrelated errors: $\mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_j(\mathbf{x})\right] = 0$

$$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$$

Bootstrapped AGGregatING (BAGGING)



Dropout



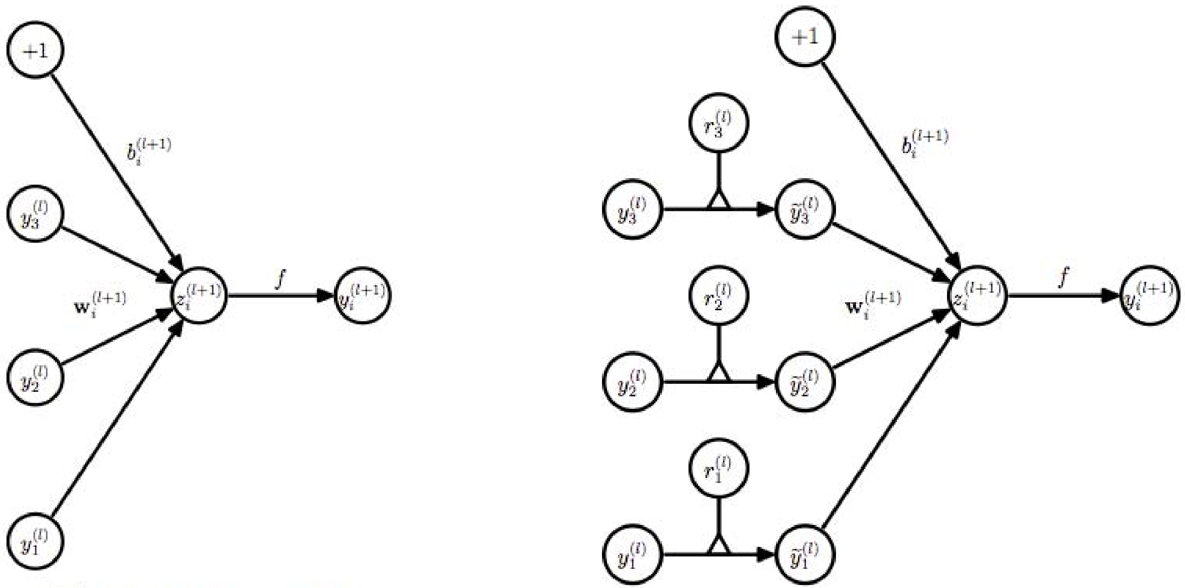
(a) Standard Neural Net (b) After applying dropout.

Each sample is processed by a 'decimated' neural net

- **Decimated nets: distinct classifiers**
- **But: they should all do the same job**

Improving neural networks by preventing co-adaptation of feature detectors
GE Hinton, N Srivastava, A Krizhevsky, I Sutskever, RR Salakhutdinov, arXiv, 2012, JMLR 2014
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Dropout block



(a) Standard network

(b) Dropout network

Figure 3: Comparison of the basic operations of a standard and dropout network.

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \mathrm{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

‘Feature noising’

Test time: Deterministic approximation

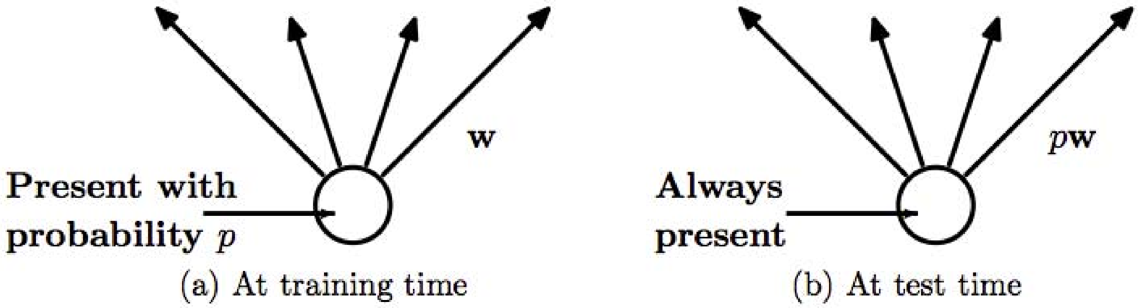
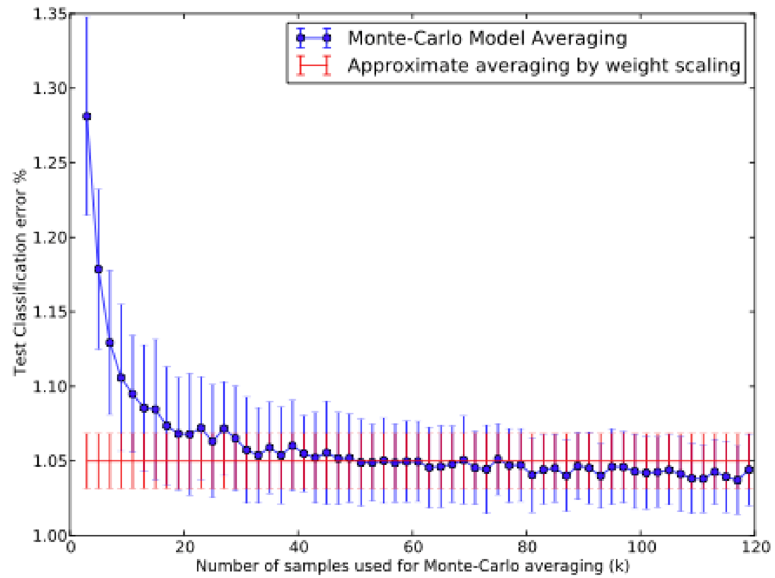


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights **w**. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

At test time, the weights are scaled as W(l) = pW(l) as shown in Figure 2. The resulting neural network is used without dropout.

An expensive but more correct way of averaging the models is to sample k neural nets using dropout for each test case and average their predictions. As k → ∞, this Monte-Carlo model average gets close to the true model average.

By computing the error for different values of k we can see how quickly the error rate of the finite-sample average approaches the error rate of the approximate model average.



Dropout performance

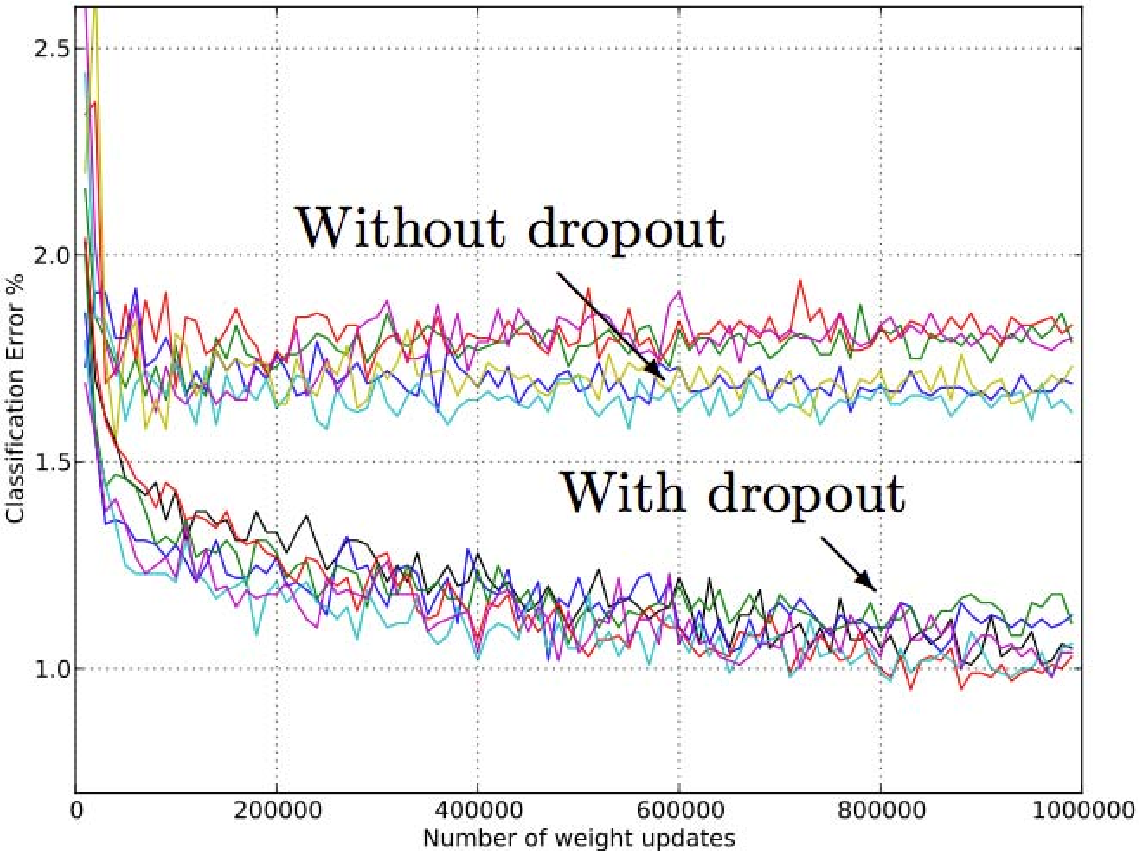


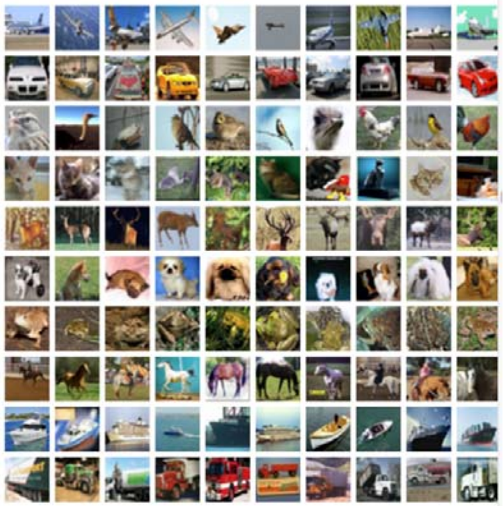
Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Dropout performance

(a) Street View House Numbers (SVHN)

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.



(b) CIFAR-10

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

Dropout performance

Figure 6: Some ImageNet test cases with the 4 most probable labels as predicted by our model. The length of the horizontal bars is proportional to the probability assigned to the labels by the model. Pink indicates ground truth.

Model	**Top-1**	**Top-5**
Sparse Coding (Lin et al., 2010)	47.1	28.2
SIFT + Fisher Vectors (Sanchez and Perronnin, 2011)	45.7	25.7
Conv Net + dropout (Krizhevsky et al., 2012)	37.5	17.0

Table 5: Results on the ILSVRC-2010 test set.

Model	**Top-1 (val)**	**Top-5 (val)**	**Top-5 (test)**
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Neural network training: old & new tricks

Old: (80's)

Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

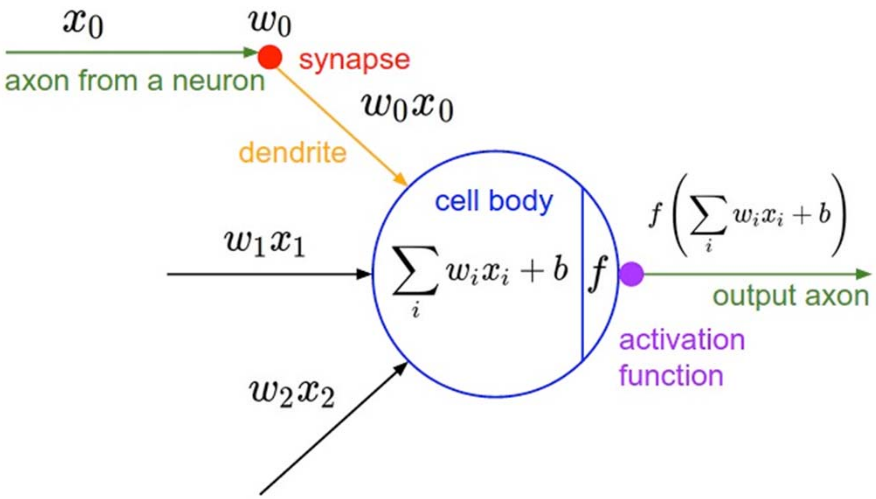
Dropout

ReLUs

Batch Normalization

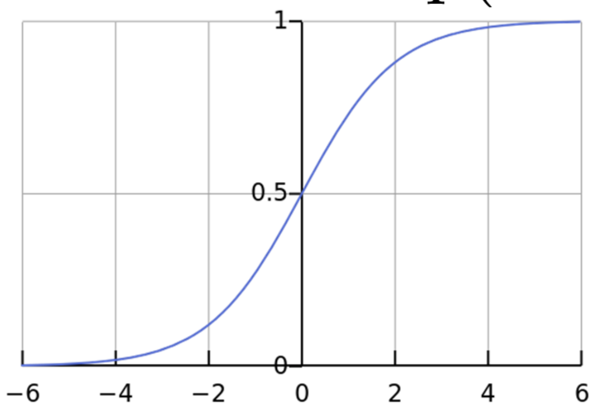
Residual Networks

‘Neuron’: cascade of linear and nonlinear function



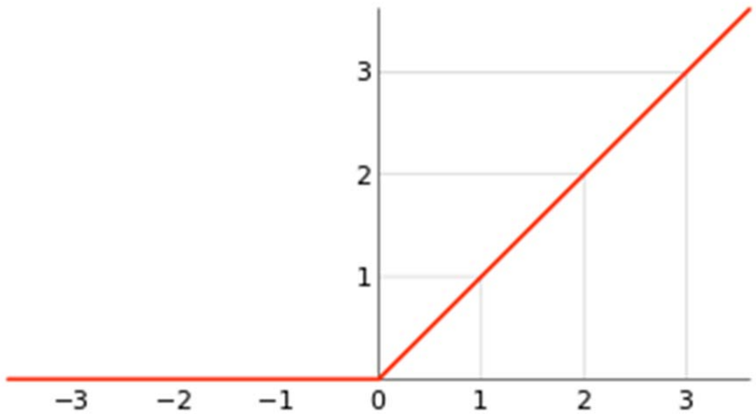
Sigmoidal (“logistic”)

$$g(a) = \frac{1}{1+\exp(-a)}$$



Rectified Linear Unit (RELU)

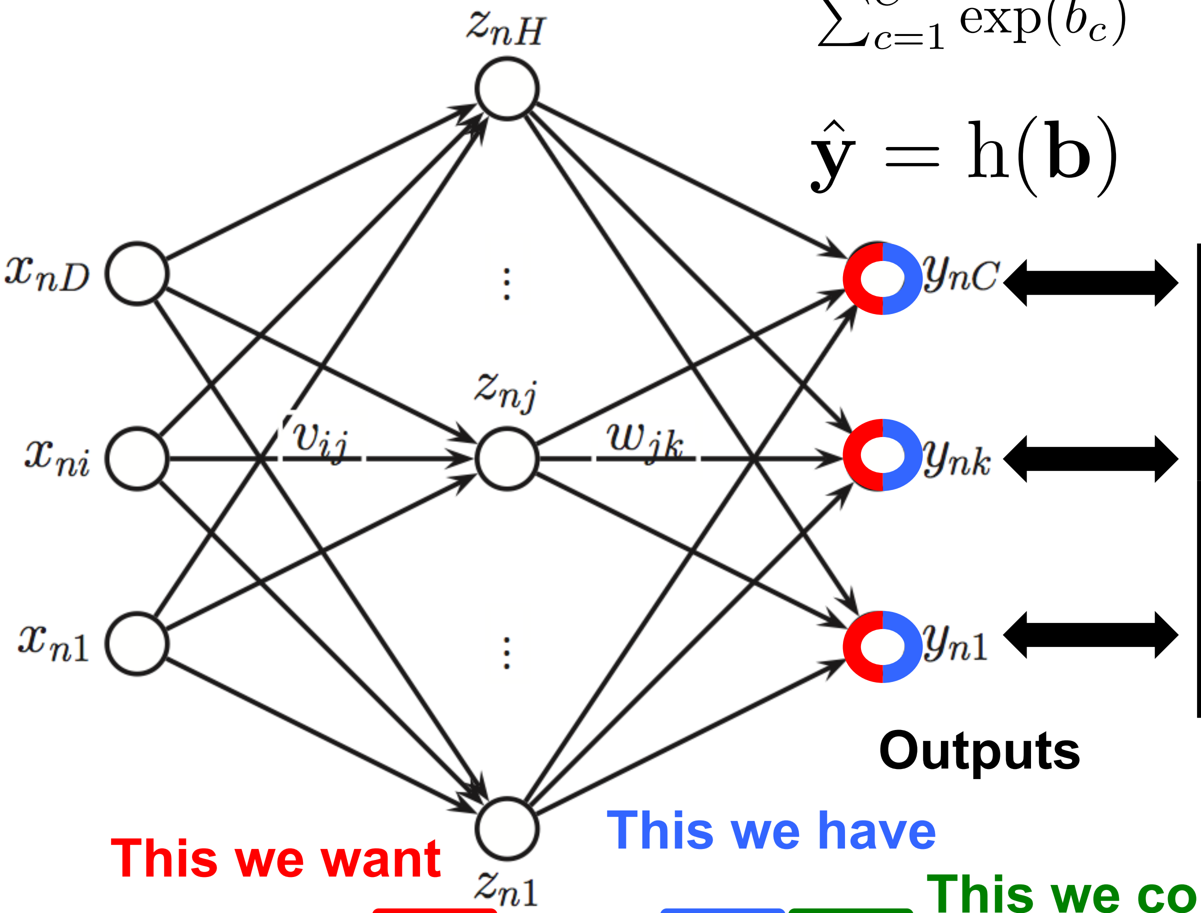
$$g(a) = \max(0, a)$$



A neural network in backward mode:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum_{c=1}^{C} \exp(b_c)}$$

$$\hat{\mathbf{y}} = \mathrm{h}(\mathbf{b})$$



$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

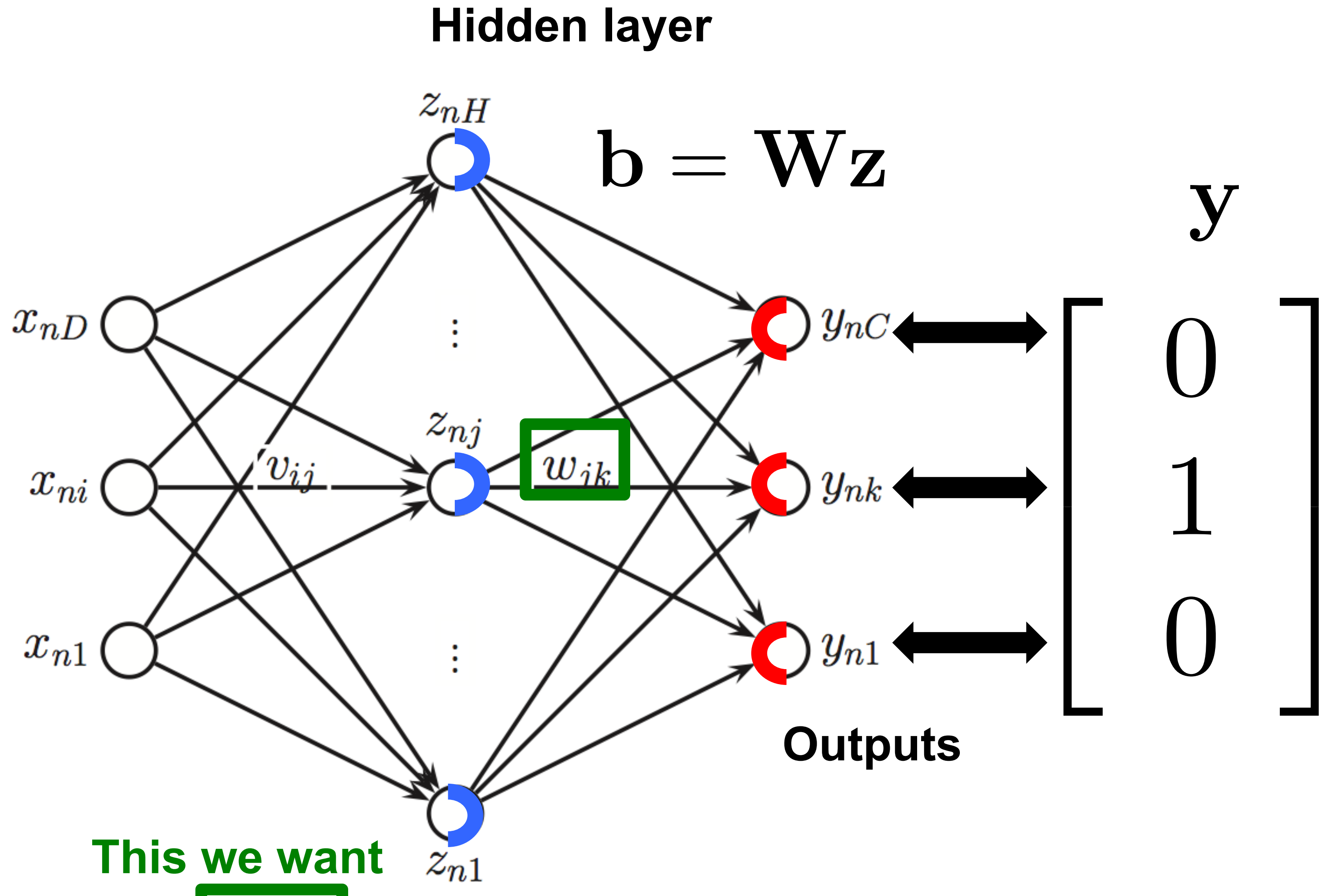
This we want

This we have

This we computed

$$\frac{\partial L}{\partial \hat{y}_c} = -\frac{y_c}{\hat{y}_c} \qquad \boxed{\frac{\partial L}{\partial b_k}} = \sum_c \boxed{\frac{\partial L}{\partial \hat{y}_c}} \boxed{\frac{\partial \hat{y}_c}{\partial b_k}} = \hat{y}_k - y_k$$

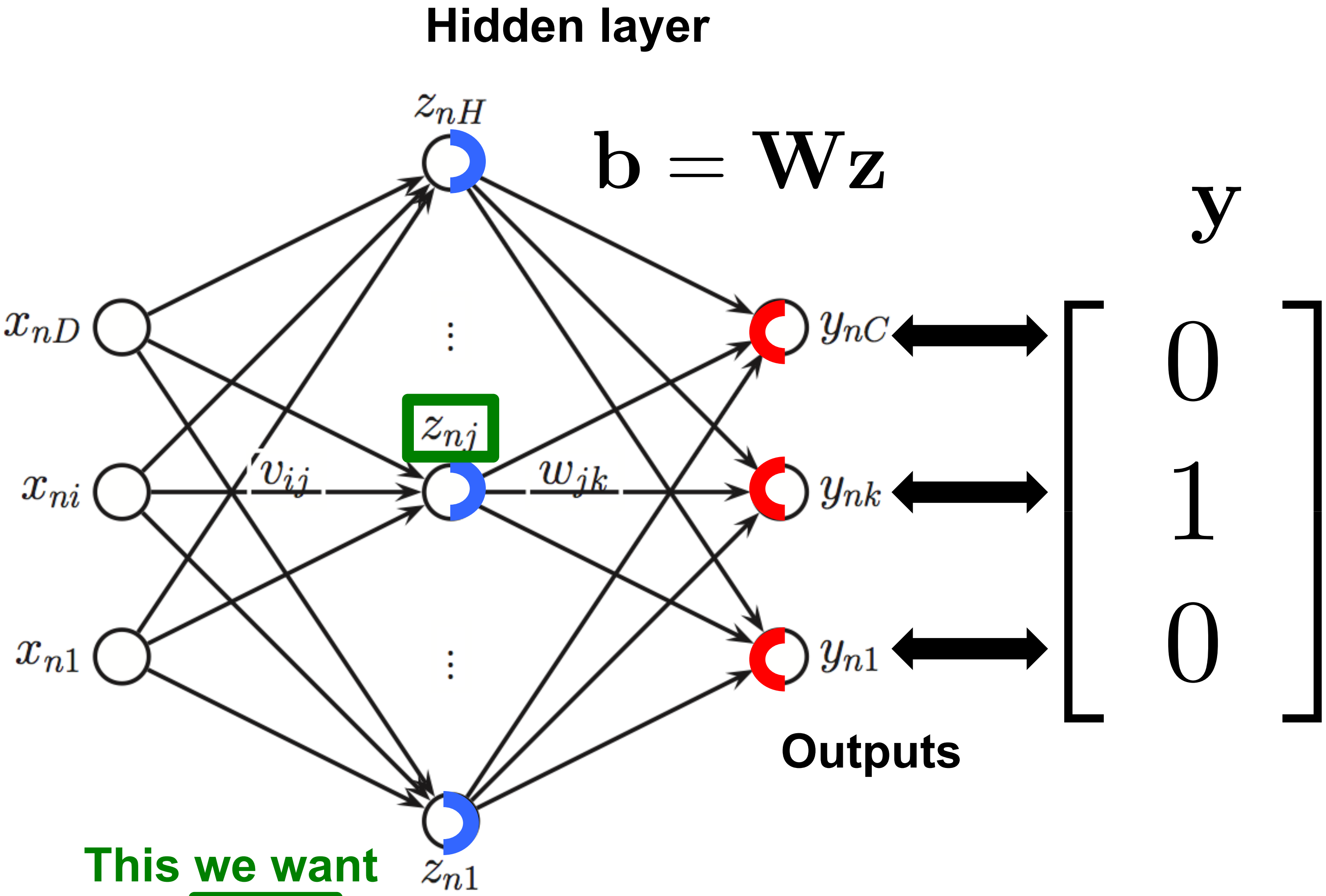
A neural network in backward mode:



This we want

$$\boxed{\frac{\partial l}{\partial w_{jk}}} = \text{?}$$

A neural network in backward mode:

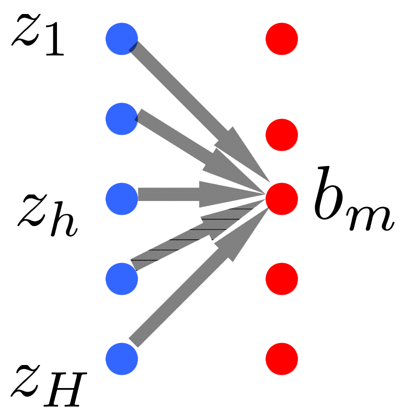


This we want

$$\boxed{\frac{\partial l}{\partial z_j}} = \ ?$$

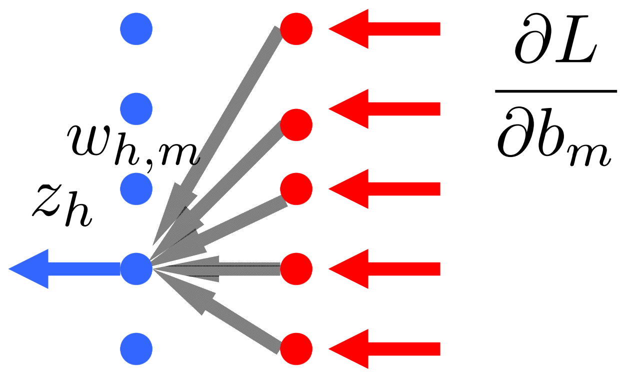
Linear layer in forward mode: all for one

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



Linear layer in backward mode: one from all

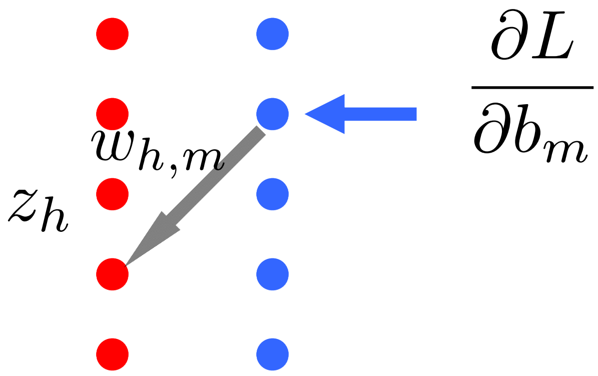
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} w_{h,c}$$

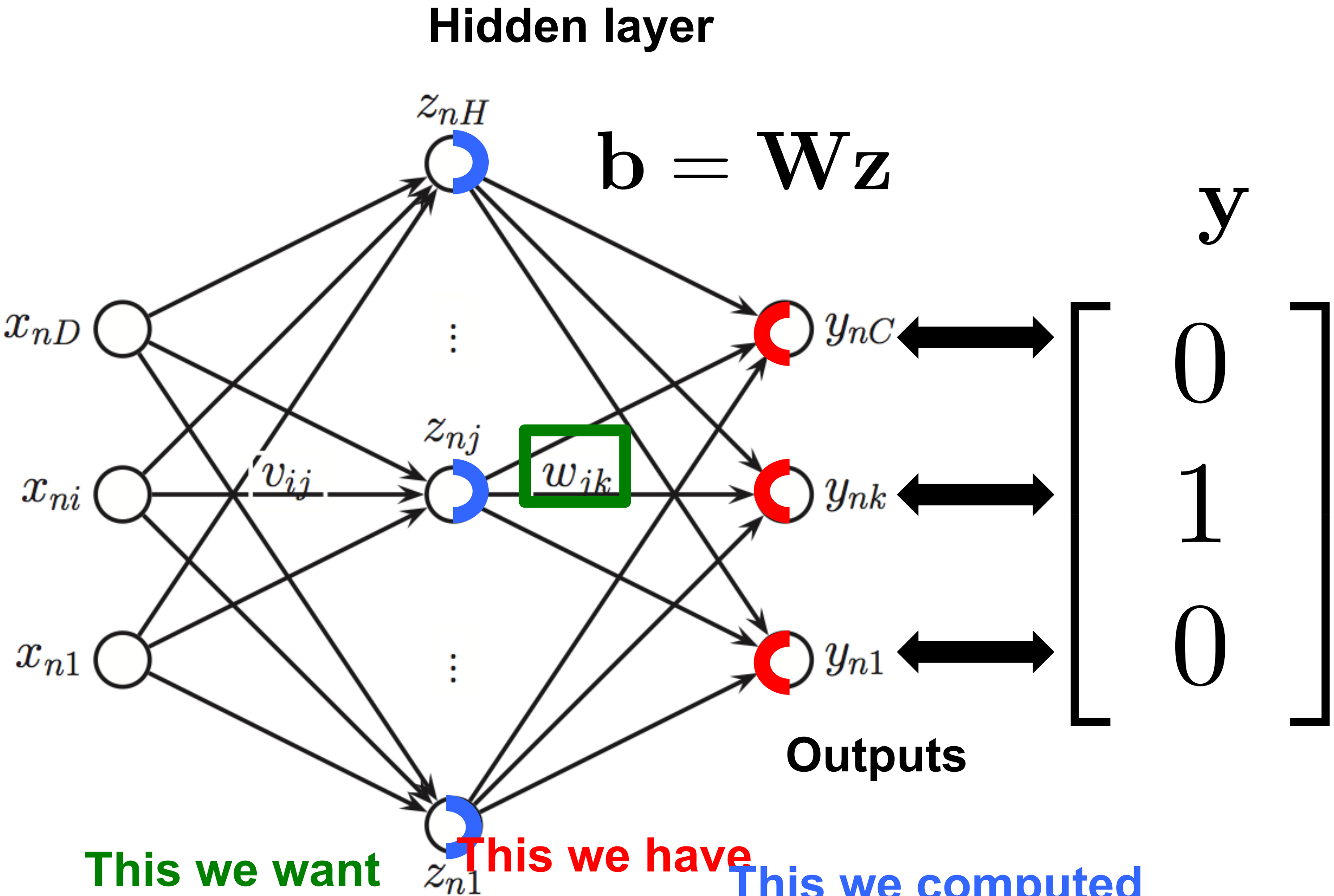
Linear layer parameters in backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial w_{h,m}} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial w_{h,m}} = \frac{\partial L}{\partial b_m} z_h$$

A neural network in backward mode:



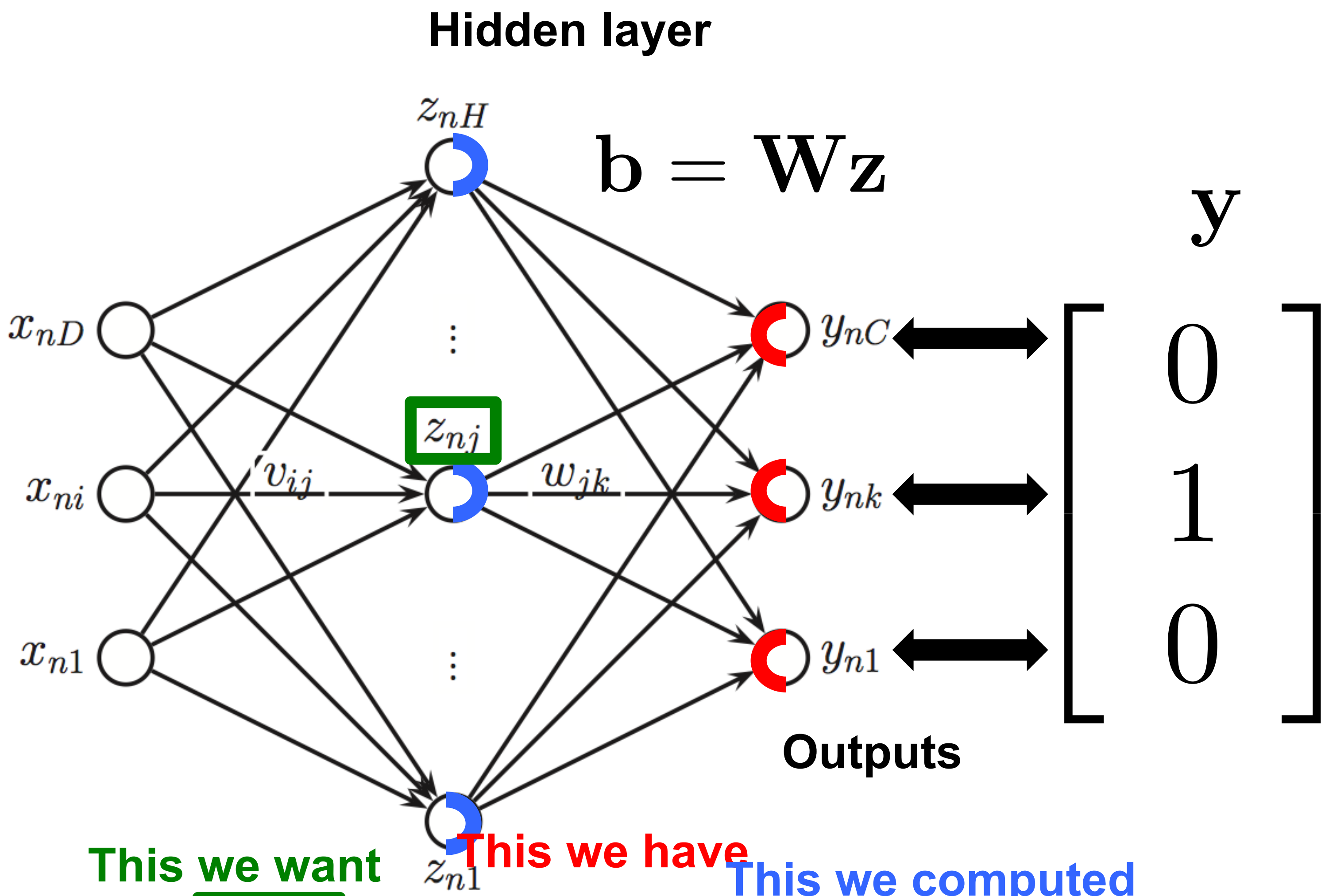
This we want

This we have

This we computed

$$\boxed{\frac{\partial l}{\partial w_{jk}}} = \sum_m \boxed{\frac{\partial l}{\partial b_m}} \boxed{\frac{\partial b_m}{\partial w_{jk}}} = \frac{\partial l}{\partial b_m} z_j$$

A neural network in backward mode:



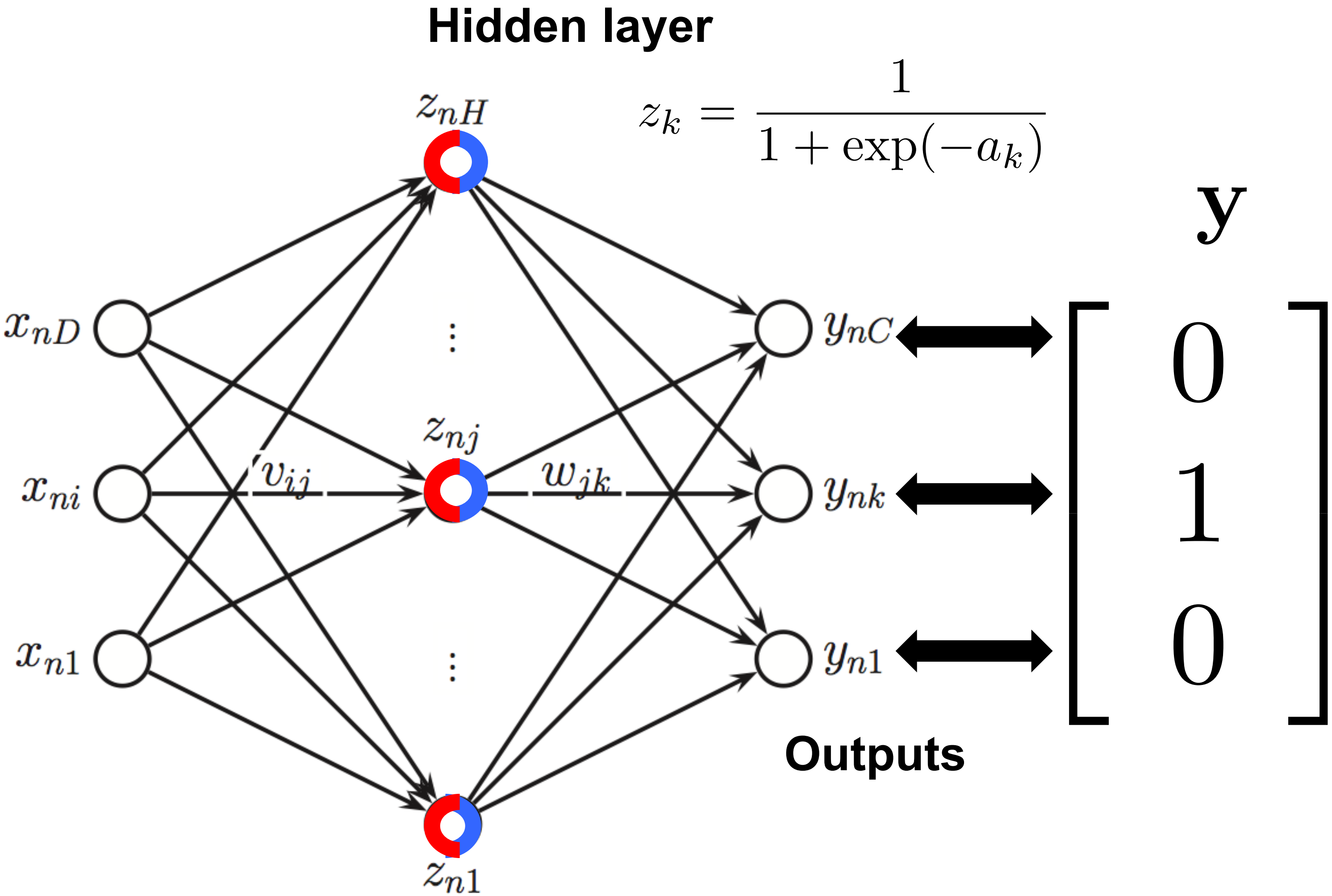
This we want

This we have

This we computed

$$\boxed{\frac{\partial l}{\partial z_j}} = \sum_m \boxed{\frac{\partial l}{\partial b_m}}\boxed{\frac{\partial b_m}{\partial z_j}} = \sum_m \frac{\partial l}{\partial b_m} w_{j,m}$$

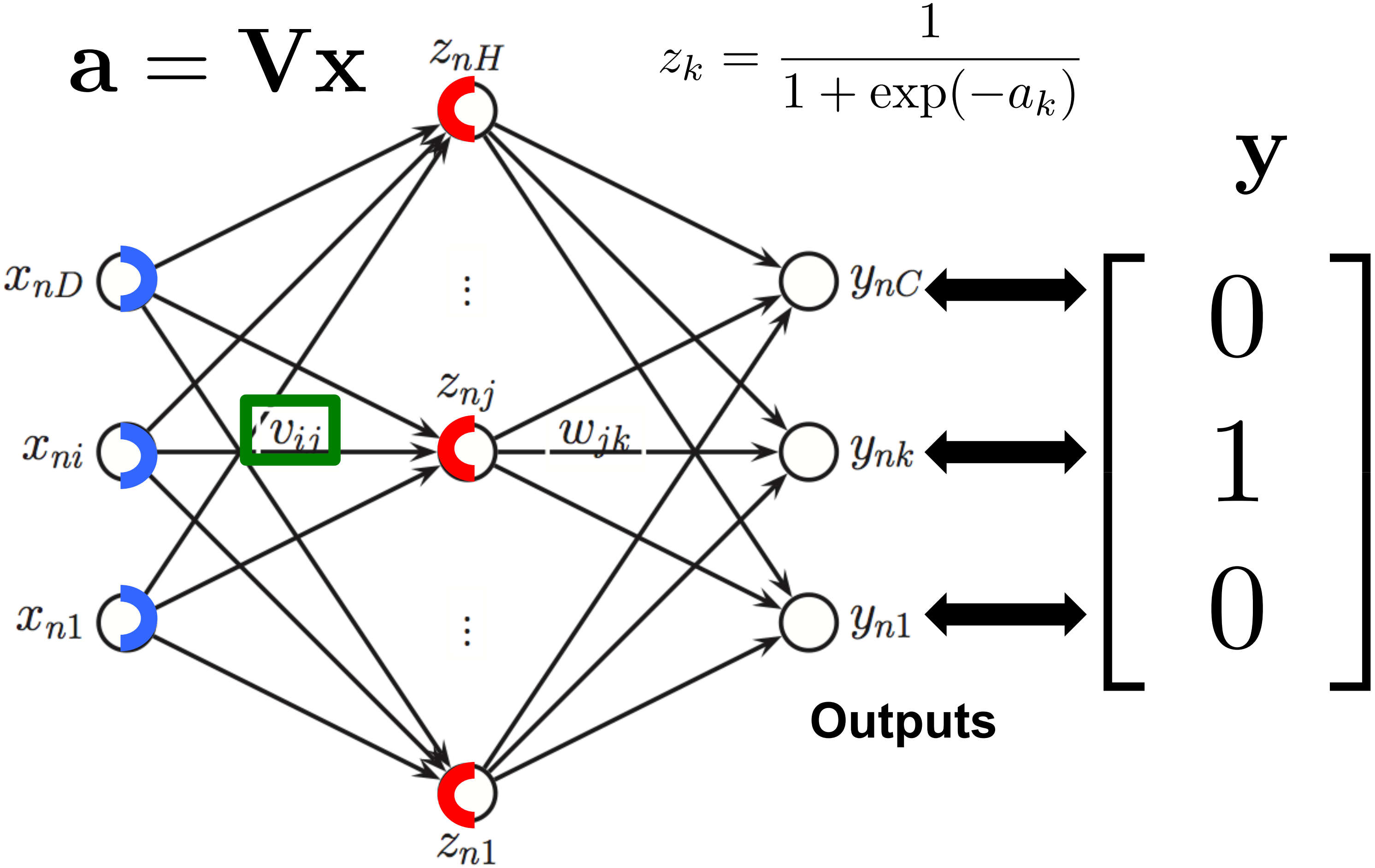
A neural network in backward mode:



$$z_k = \frac{1}{1 + \exp(-a_k)}$$

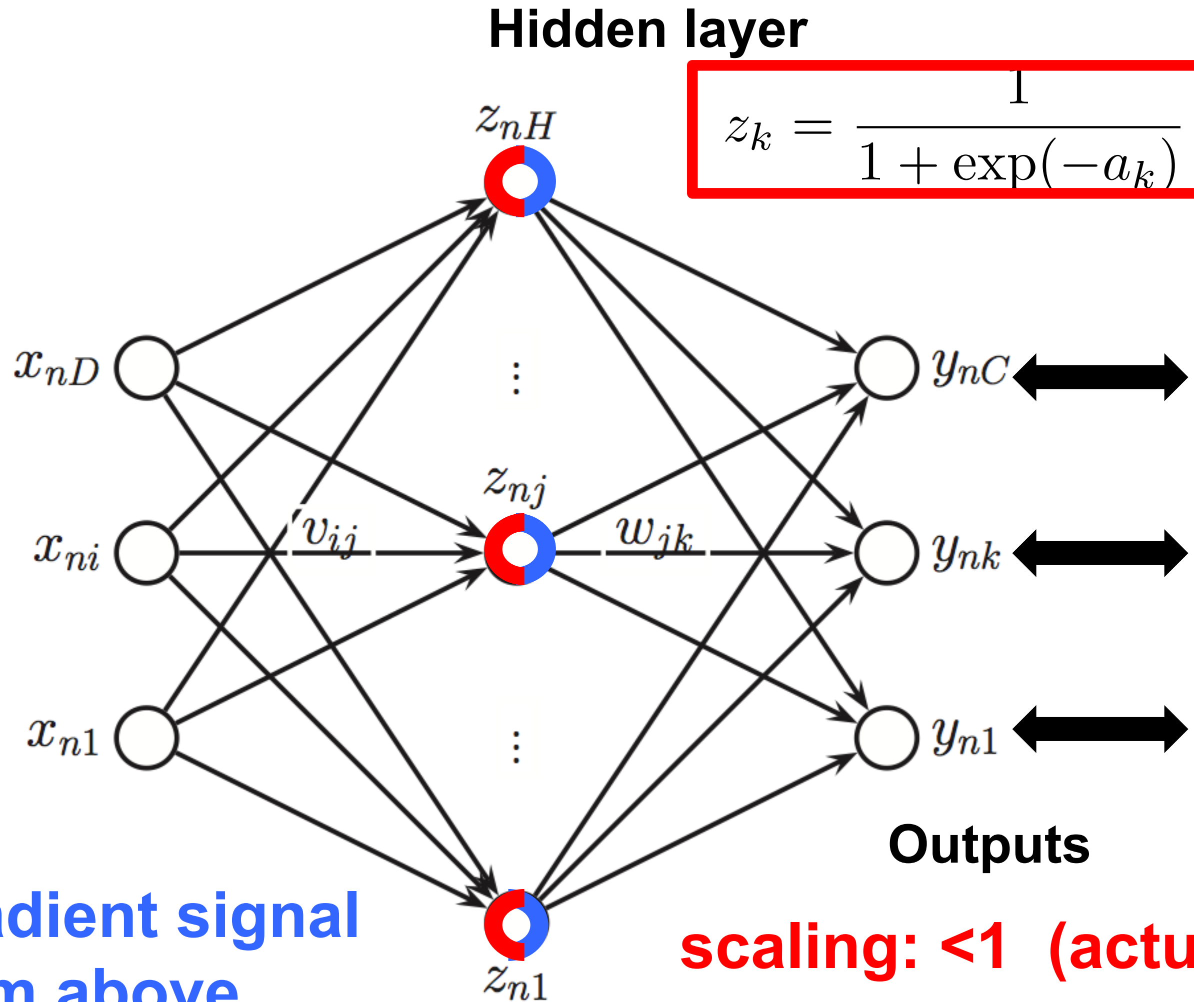
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

A neural network in backward mode:



$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k}\frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

A neural network in backward mode:



$$z_k = \frac{1}{1+\exp(-a_k)}$$

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Gradient signal from above

scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \boxed{\frac{\partial l}{\partial z_k}}\boxed{g'(a_k)} = \frac{\partial l}{\partial z_k}\boxed{g(a_k)(1-g(a_k))}$$

Vanishing gradients problem

Gradient signal from above

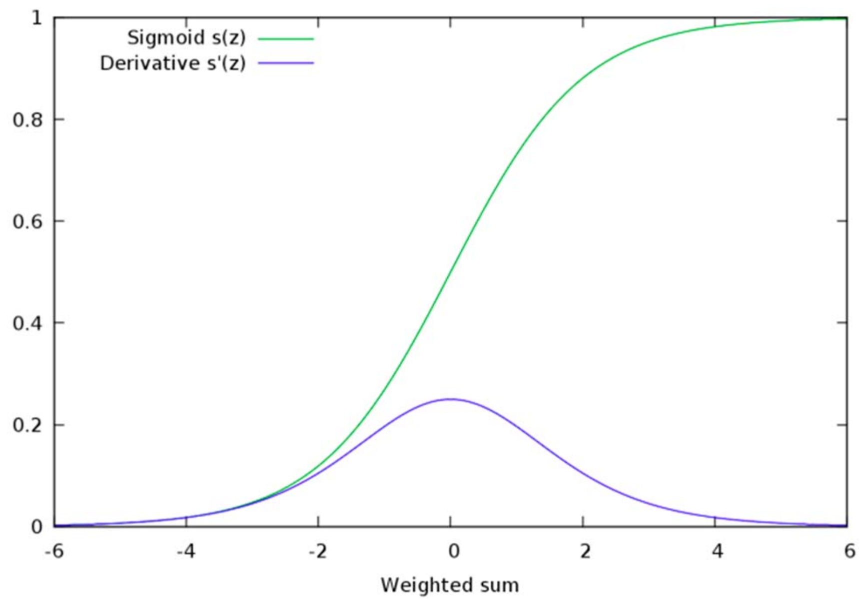
scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \boxed{\frac{\partial l}{\partial z_k}}\boxed{g'(a_k)} = \frac{\partial l}{\partial z_k}\boxed{g(a_k)(1-g(a_k))}$$

Do this 10 times: updates in the first layers get minimal

Top layer knows what to do, lower layers "don't get it"

Sigmoidal Unit:
Signal is not getting through!



Vanishing gradients problem: ReLU solves it

Gradient signal from above

Scaling: {0,1}

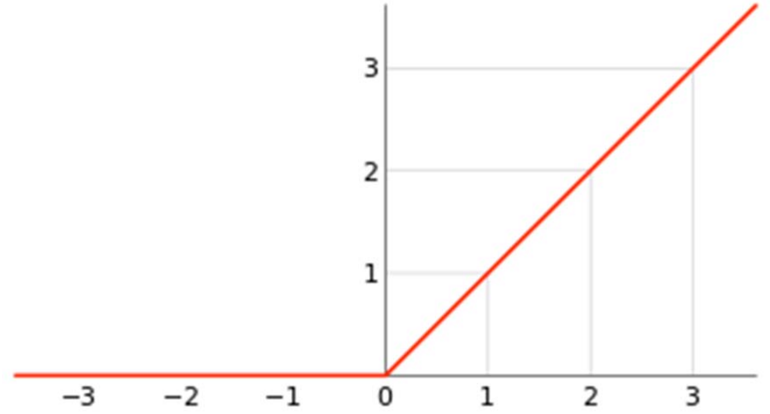
$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \boxed{\frac{\partial l}{\partial z_k}} \boxed{g'(a_k)}$$

Do this 10 times: updates in the first layers can remain large

Top layer knows what to do, lower layers " get it"

Rectified Linear Unit (RELU)

$$g(a) = \max(0, a)$$



$$g'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases}$$

Vinod Nair and Geoffrey Hinton (2010). Rectified linear units improve restricted Boltzmann machines. ICML.

Neural network training: old & new tricks

Old: (80's)

- **Stochastic Gradient Descent, Momentum, "weight decay"**

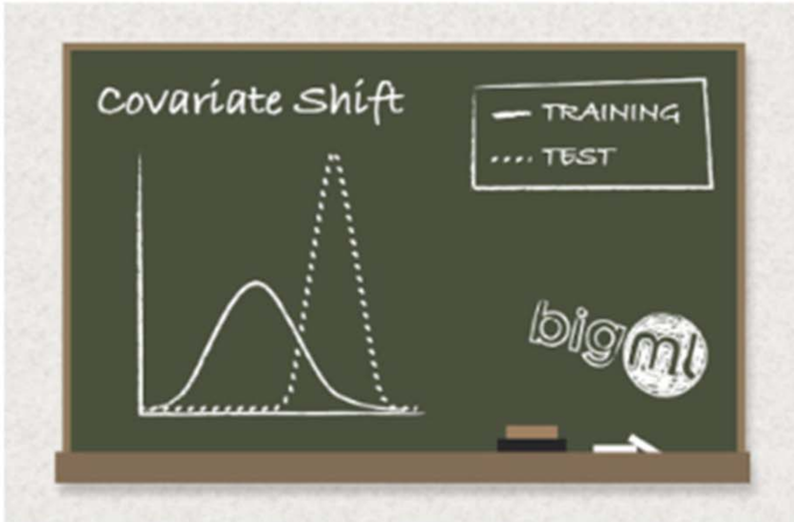
New: (last 5-6 years)

- **Dropout**
- **ReLUs**
- **Batch Normalization**
- **Residual Networks**

Covariate Shift Problem

Train speech recognition system with British speakers, test with Americans

- Traditional machine learning must contend with *covariate shift* between data sets.



blog.bigml.com

Covariate shift in a single day

10 am

2pm

7pm

"Whitening": set mean = 0, variance = 1

Photometric transformation: $I \rightarrow a\,I + b$

Original Patch and Intensity Values

Brightness Decreased

Contrast increased,

- Make each patch have zero mean:

$$\mu = \frac{1}{N}\sum_{x,y} I(x,y)$$

$$Z(x,y) = I(x,y) - \mu$$

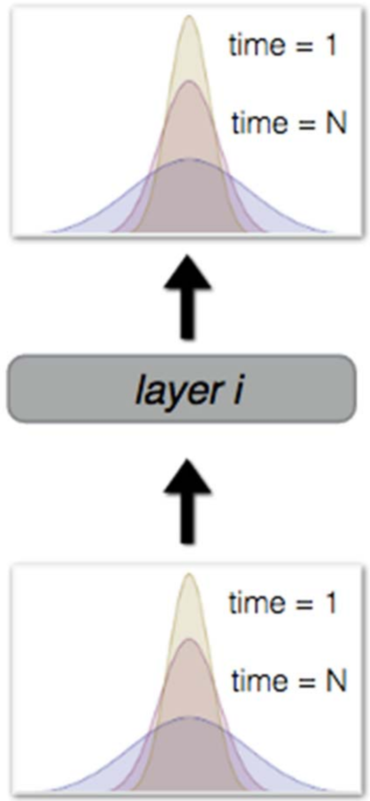
- Then make it have unit variance:

$$\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2$$

$$ZN(x,y) = \frac{Z(x,y)}{\sigma}$$

Internal Covariate Shift

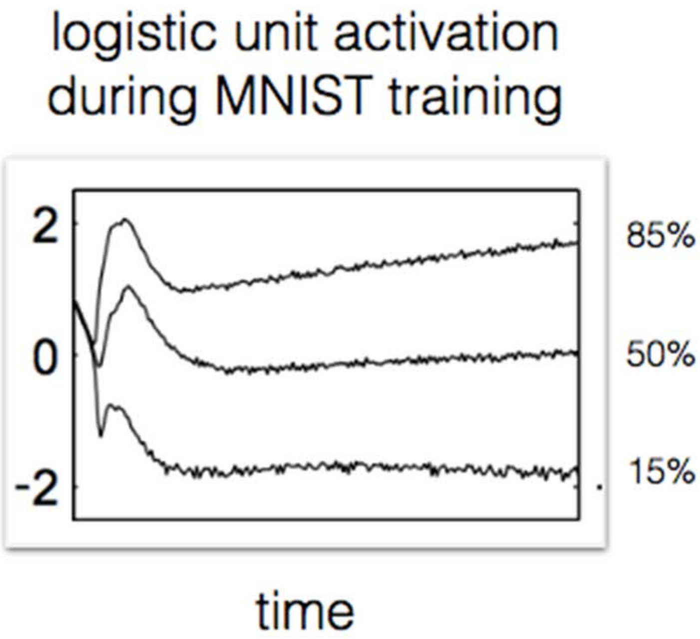
Neural network activations: moving target



Internal Covariate Shift

Neural network activations: moving target

- Covariate shifts occur across layers in a deep network.
- Performing domain adaptation or whitening is impractical in an online setting.



Batch Normalization

Whiten-as-you-go:

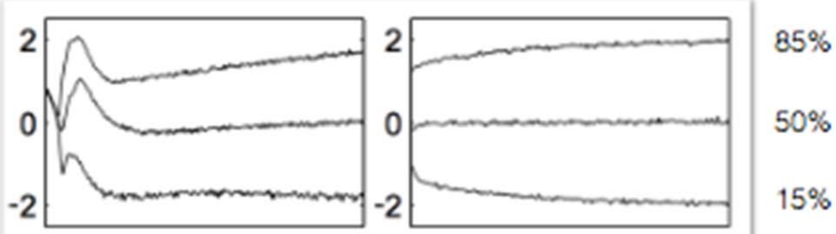
- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance (γ, β) of each layer as parameters

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



(b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
S Ioffe and C Szegedy (2015)

Batch Normalization

Whiten-as-you-go:

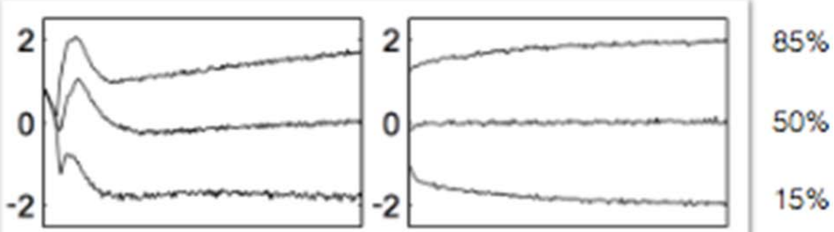
- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance (γ, β) of each layer as parameters

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

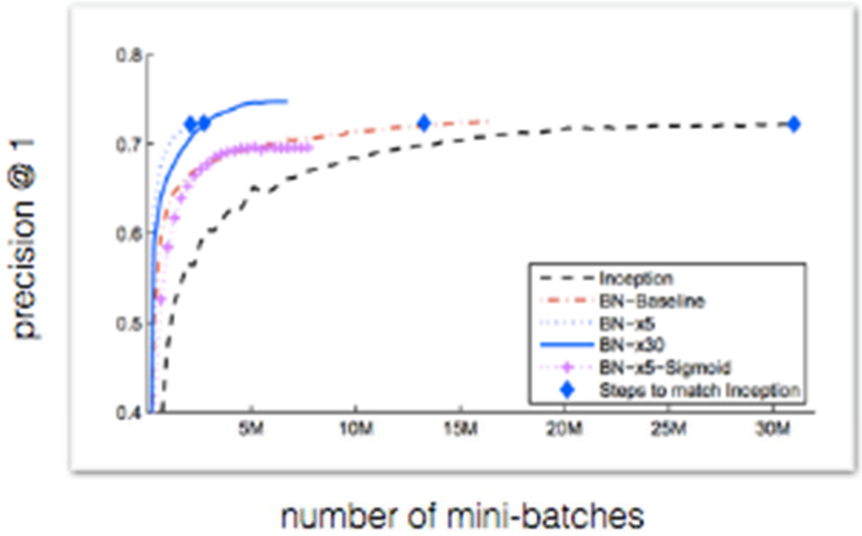


(b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
S Ioffe and C Szegedy (2015)

Batch Normalization: used in all current systems

- Multi-layer CNN's train faster with fewer data samples (15x).
- Employ faster learning rates and less network regularizations.
- Achieves state of the art results on ImageNet.



Neural network training: old & new tricks

Old: (80's)

Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

Dropout

ReLUs

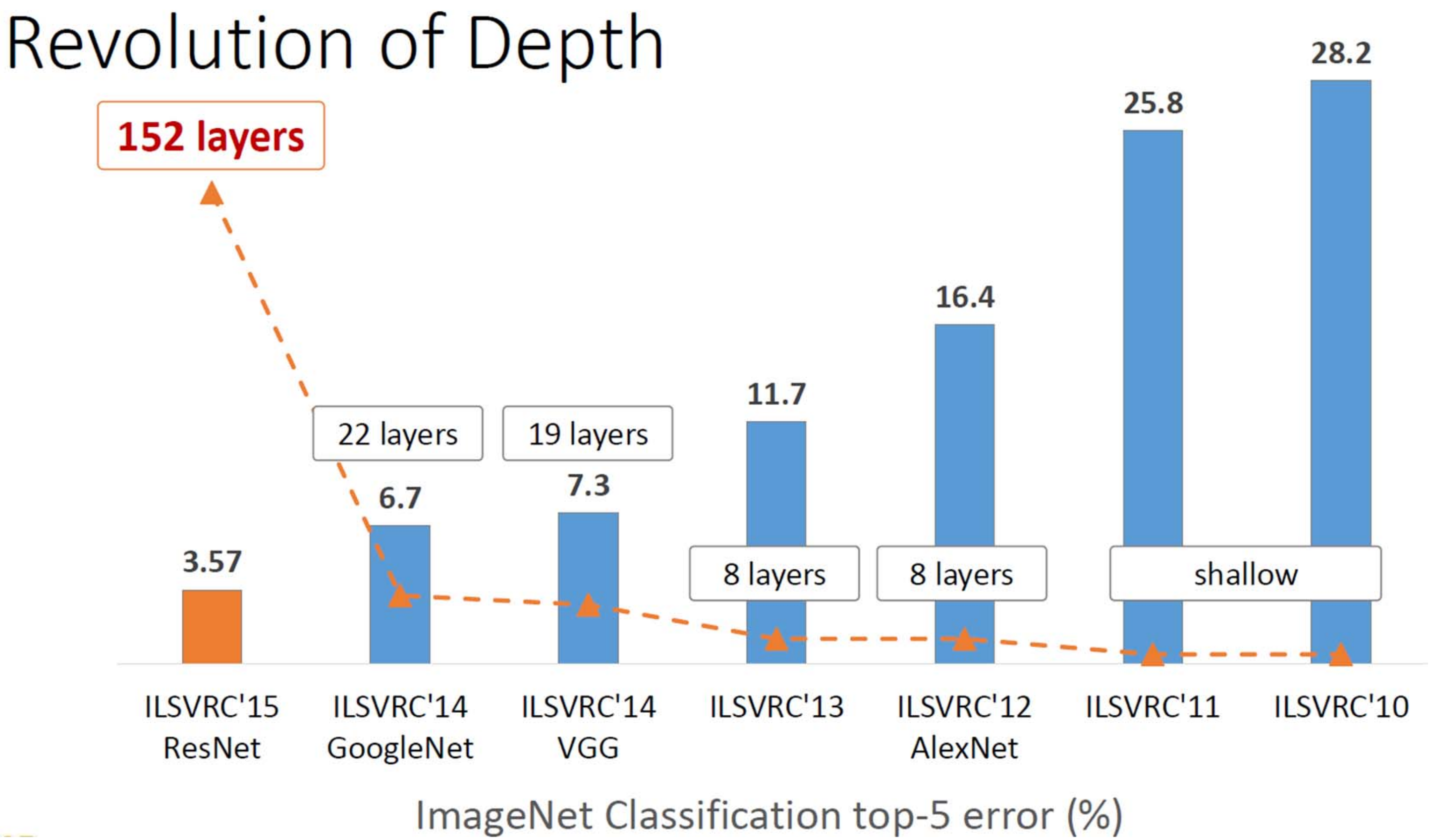
Batch Normalization

Residual Networks

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016 (best paper award).

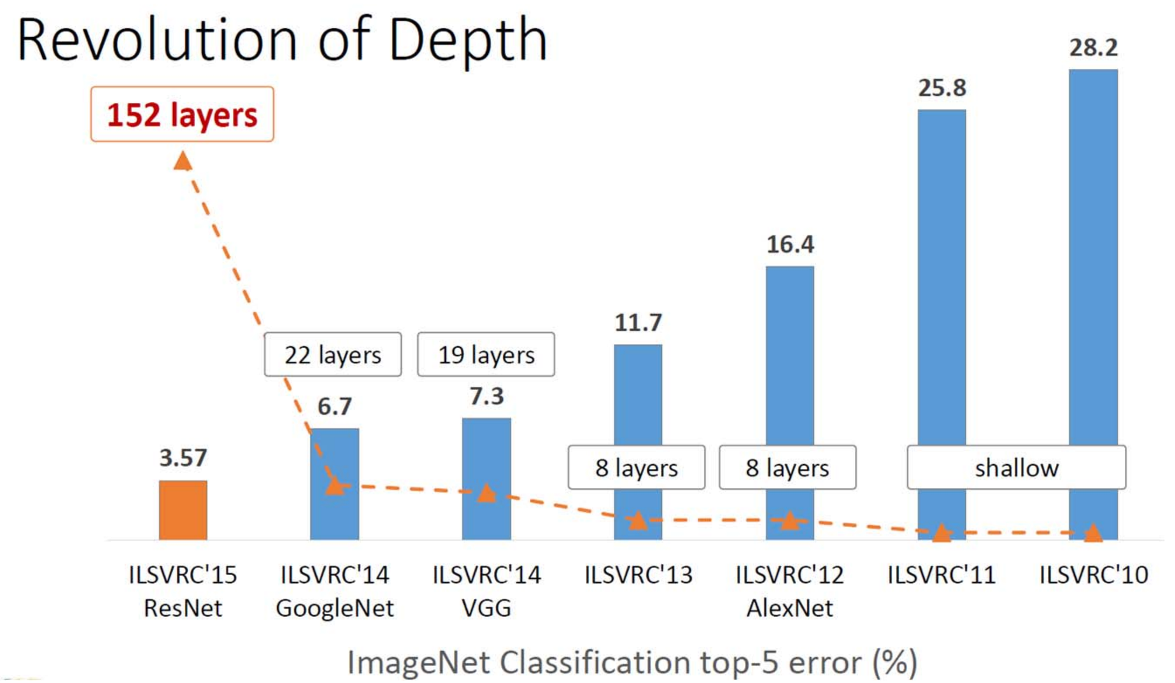
The deeper, the better

- Deeper networks can cover more complex problems
  - Increasingly large receptive field size
  - Increasingly non-linear patterns



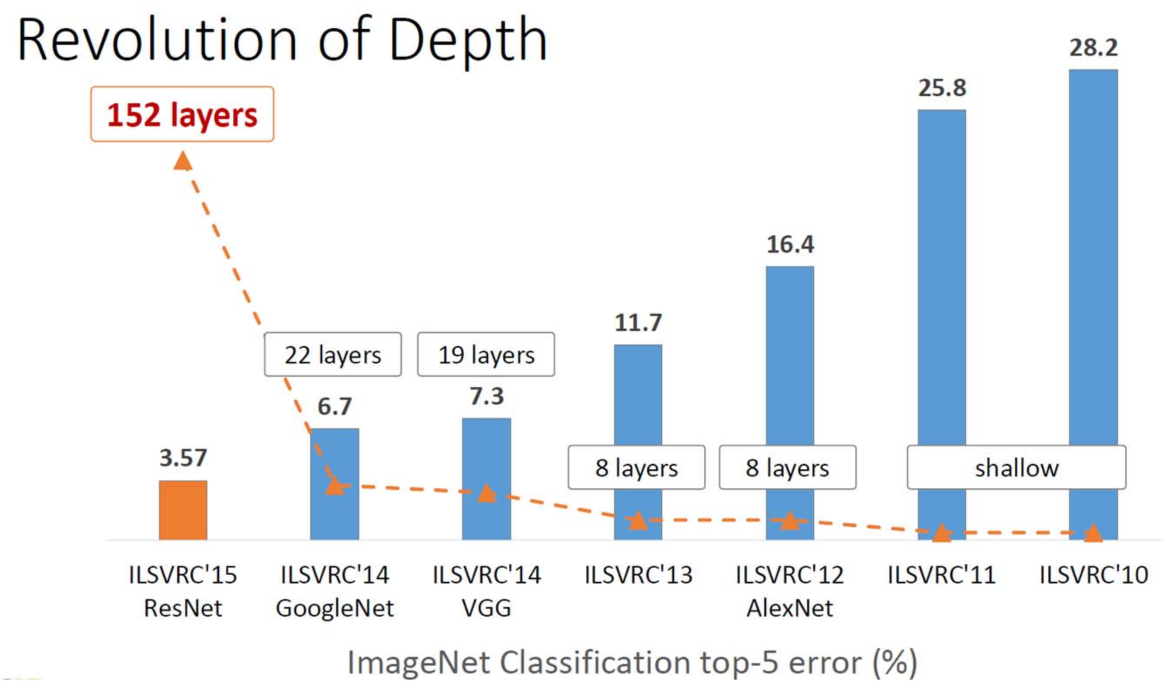
Going Deeper

- From 2 to 10: 2010-2012
  - ReLUs
  - Dropout
  - …



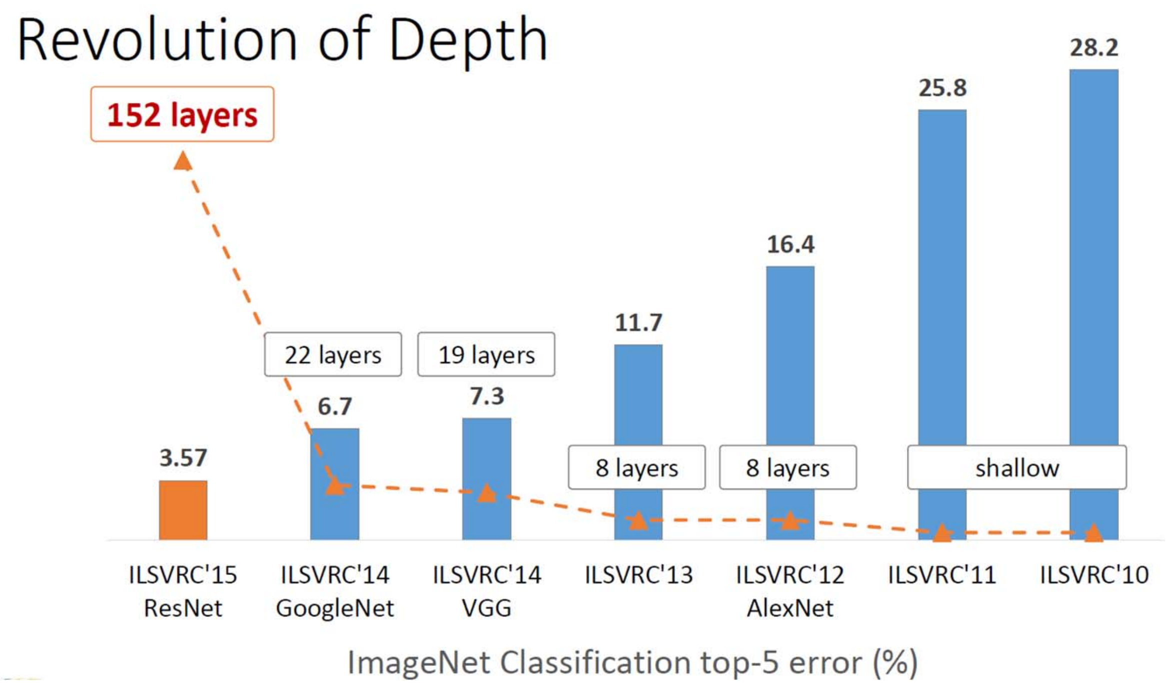
Going Deeper

- From 10 to 20: 2015
  - Batch Normalization



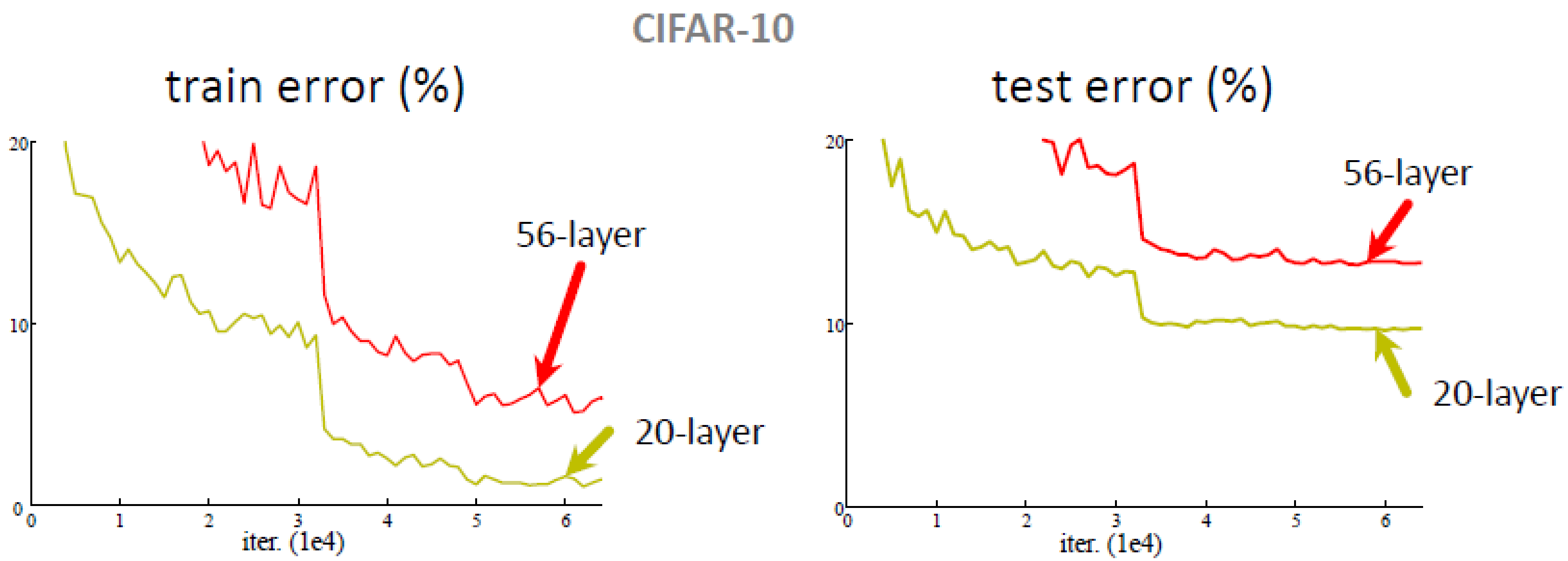
Going Deeper

- From 20 to 100/1000
  - Residual networks



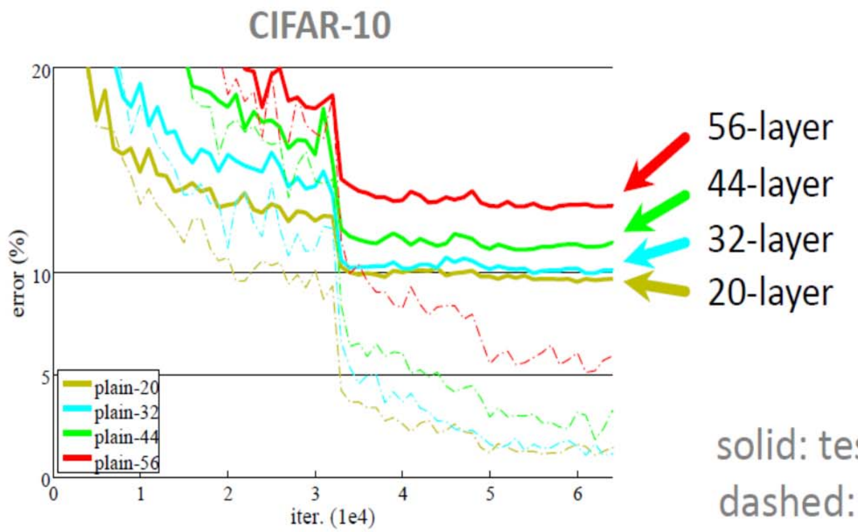
Plain Network

- Plain nets: stacking 3x3 conv layers
- 56-layer net has higher training error and test error than 20-layer net

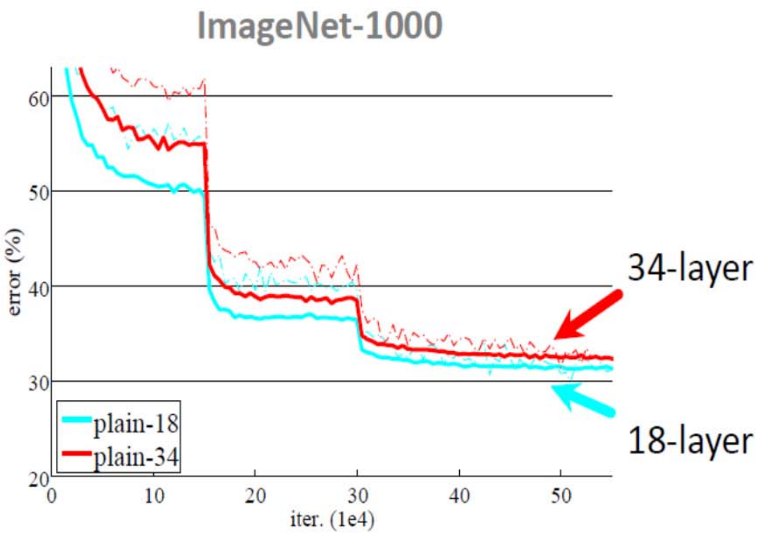


Plain Network

- “Overly deep” plain nets have higher training error
- A general phenomenon, observed in many datasets
- Counterintuitive: deeper network should be stronger, right?
  - Idea: make sure deeper network can fallback to the shallow solution.

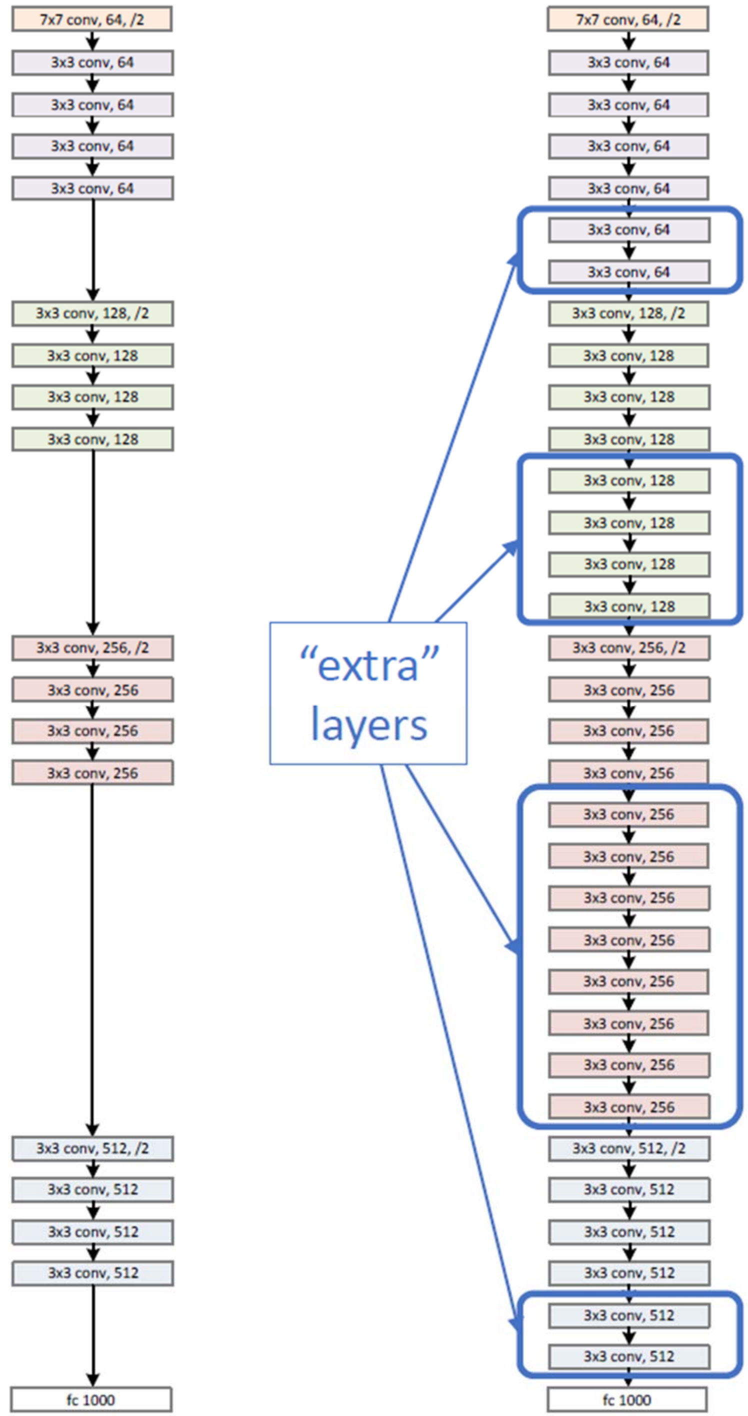


solid: test/val
dashed: train



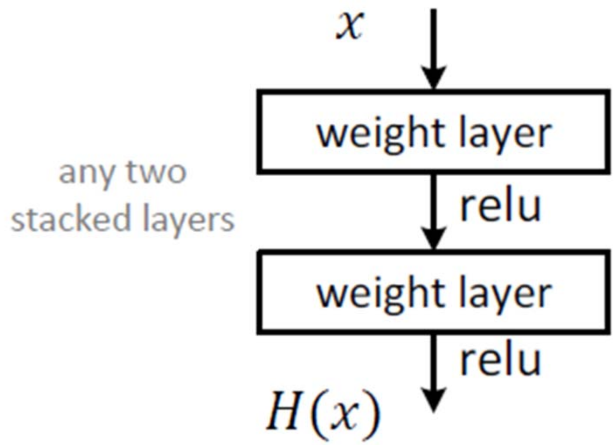
Residual Network

- Naïve solution
  - If extra layers are an identity mapping, then training errors can not increase



Residual Network

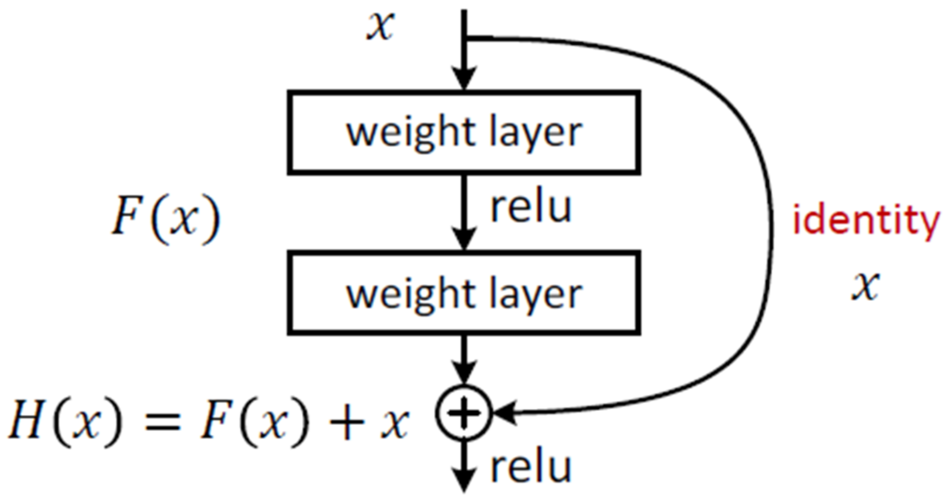
- Plain block
  - Difficult to make identity mapping because of multiple non-linear layers



Residual Network

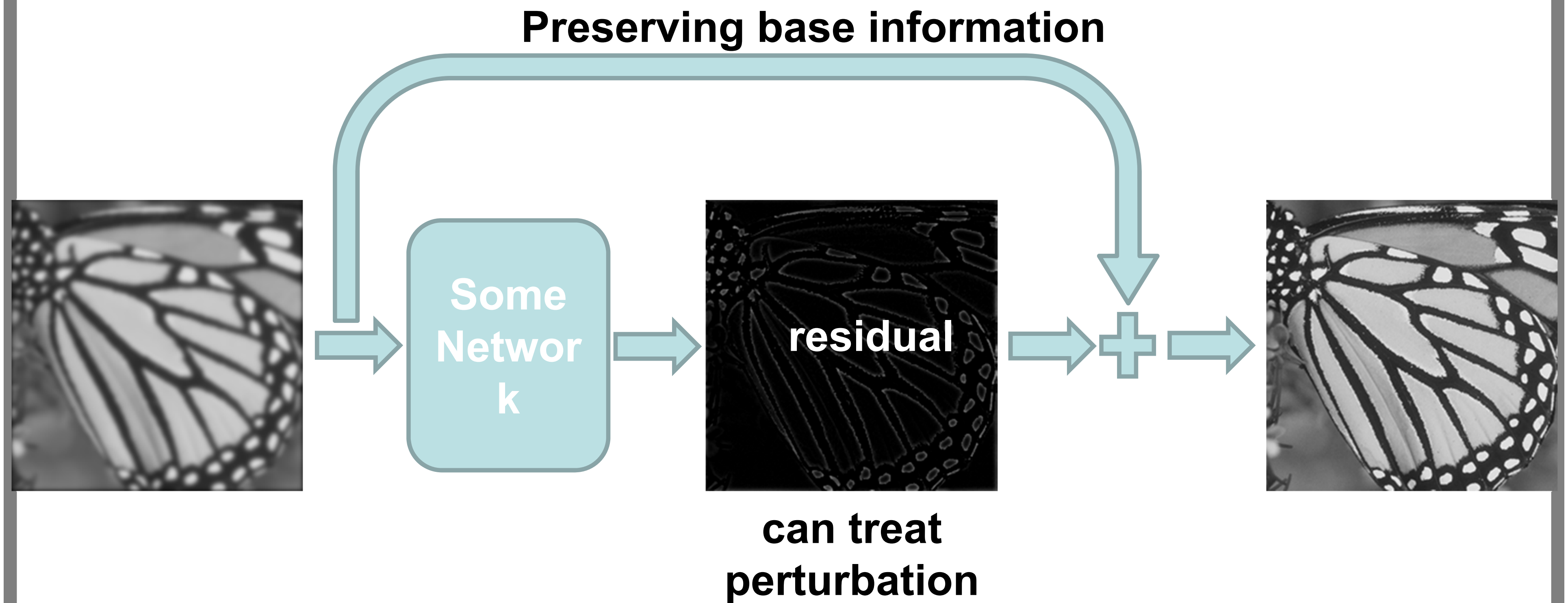
- Residual block
  - If identity were optimal, easy to set weights as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations

-> Appropriate for treating perturbation as keeping a base information



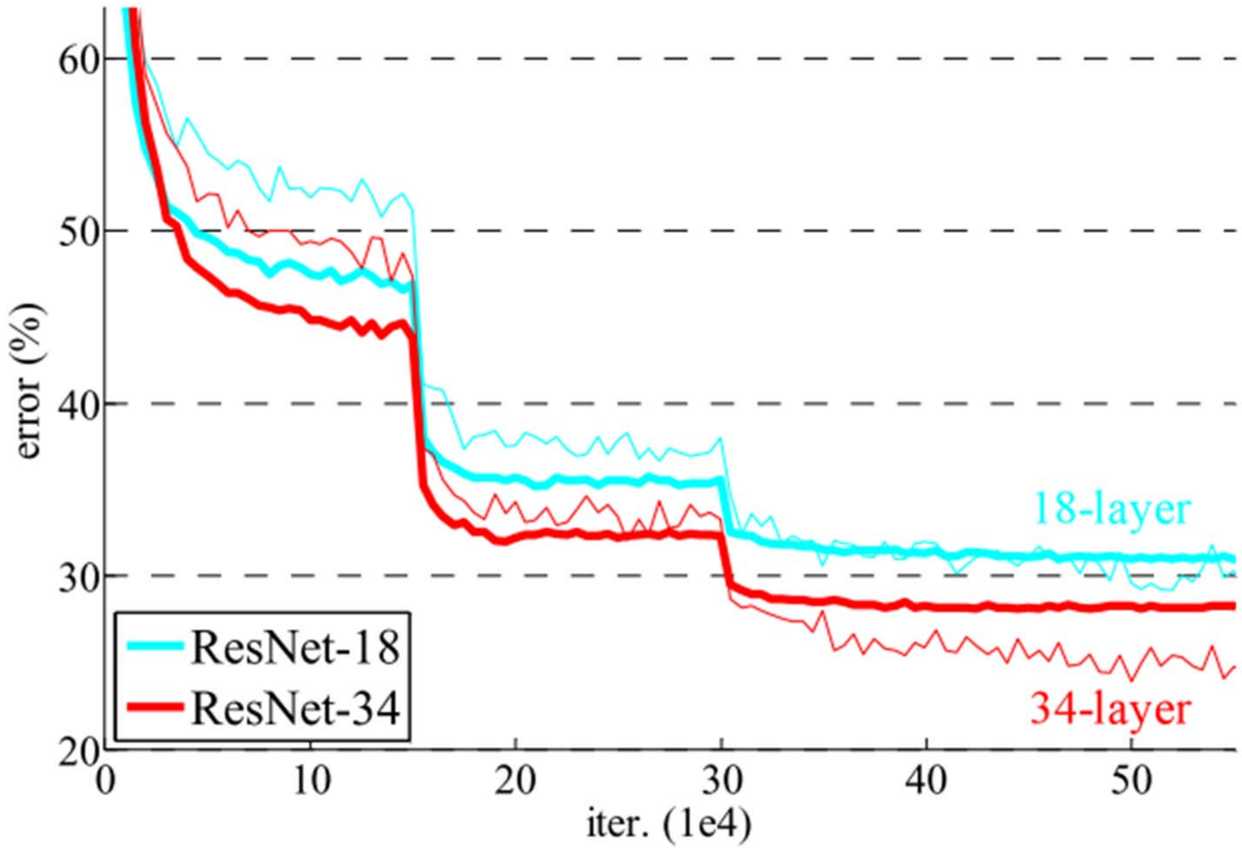
Residual modelling: basic idea in image processing

- Difference between an original image and a changed image



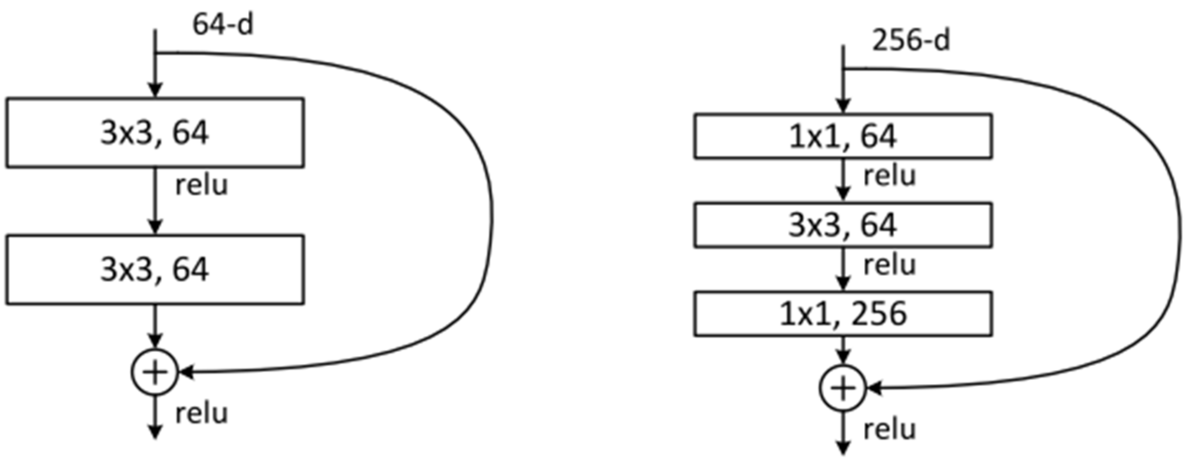
Residual Network

- Deeper ResNets have lower training error



Residual Network

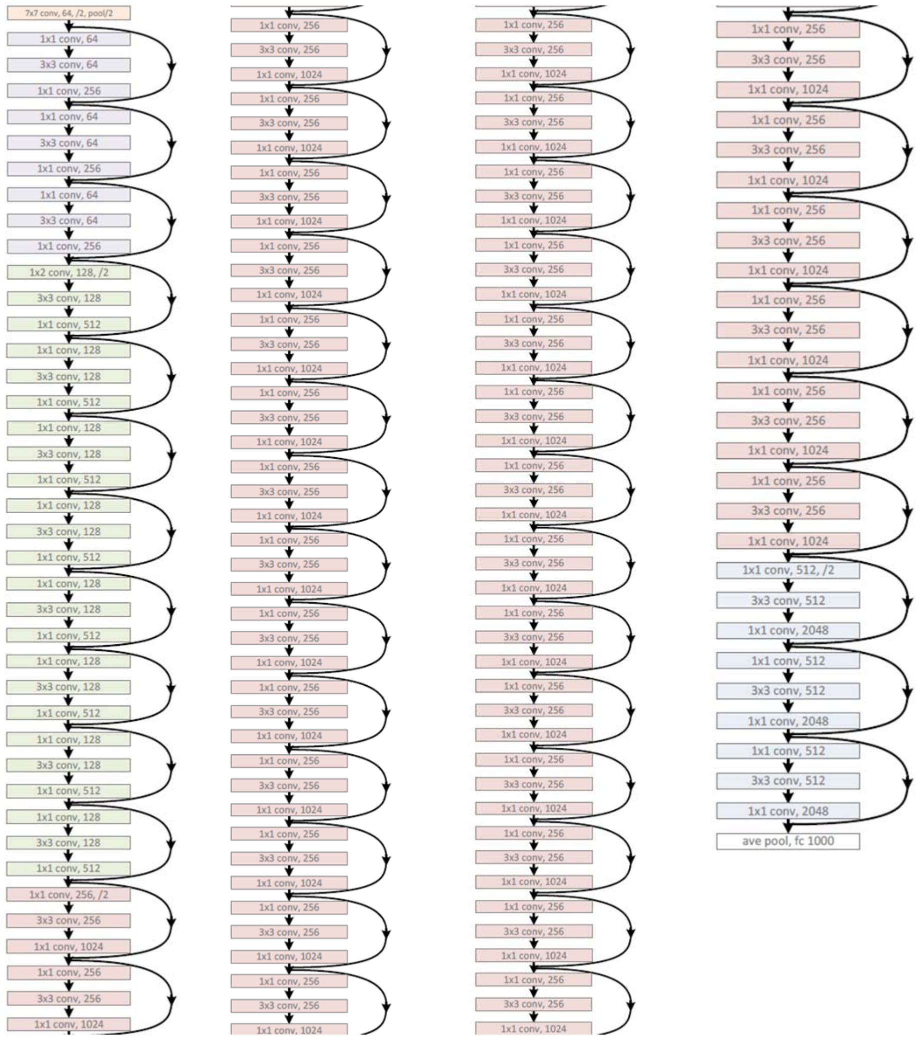
- Residual block
  - Very simple
  - Parameter-free



A naïve residual block **"bottleneck" residual block (for ResNet-50/101/152)**

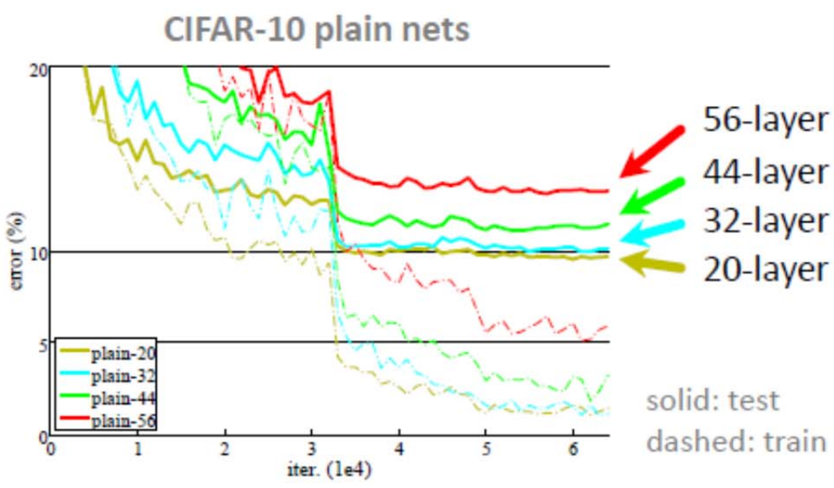
Network Design

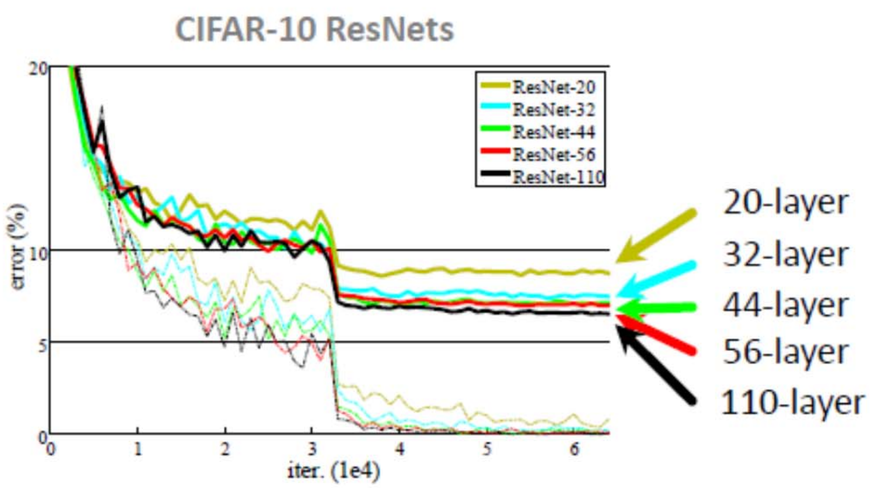
- ResNet-152
  - Use bottlenecks
  - ResNet-152(11.3 billion FLOPs) has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs)



Results

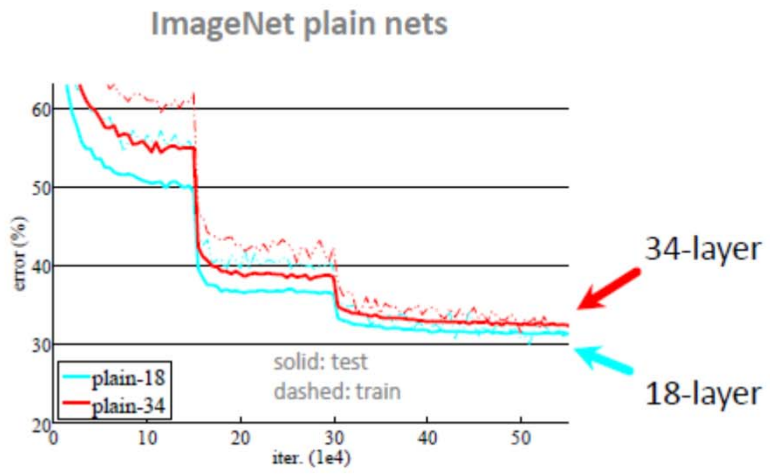
- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error

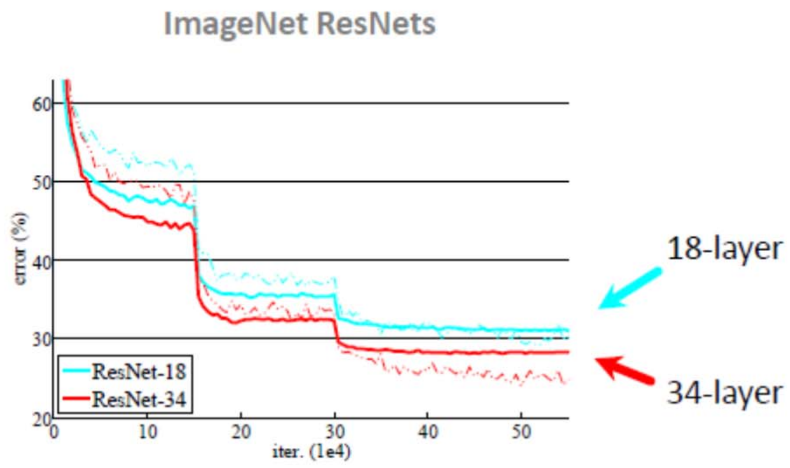




Results

- Deep Resnets can be trained without difficulties
- Deeper ResNets have lower training error, and also lower test error



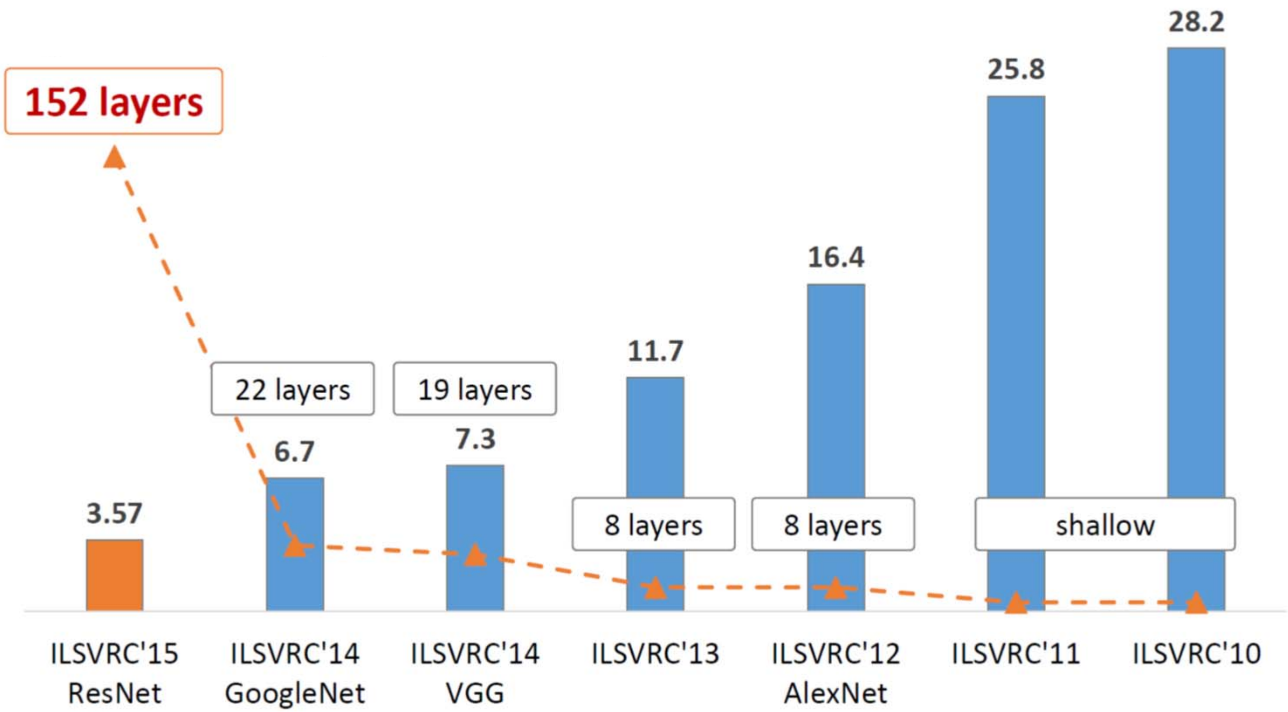


Results

- 1st places in all five main tracks in “ILSVRC & COCO 2015 Competitions”
  - ImageNet Classification
  - ImageNet Detection
  - ImageNet Localization
  - COCO Detection
  - COCO Segmentation

Quantitative Results

- ImageNet Classification



Result

- Performances increase absolutely

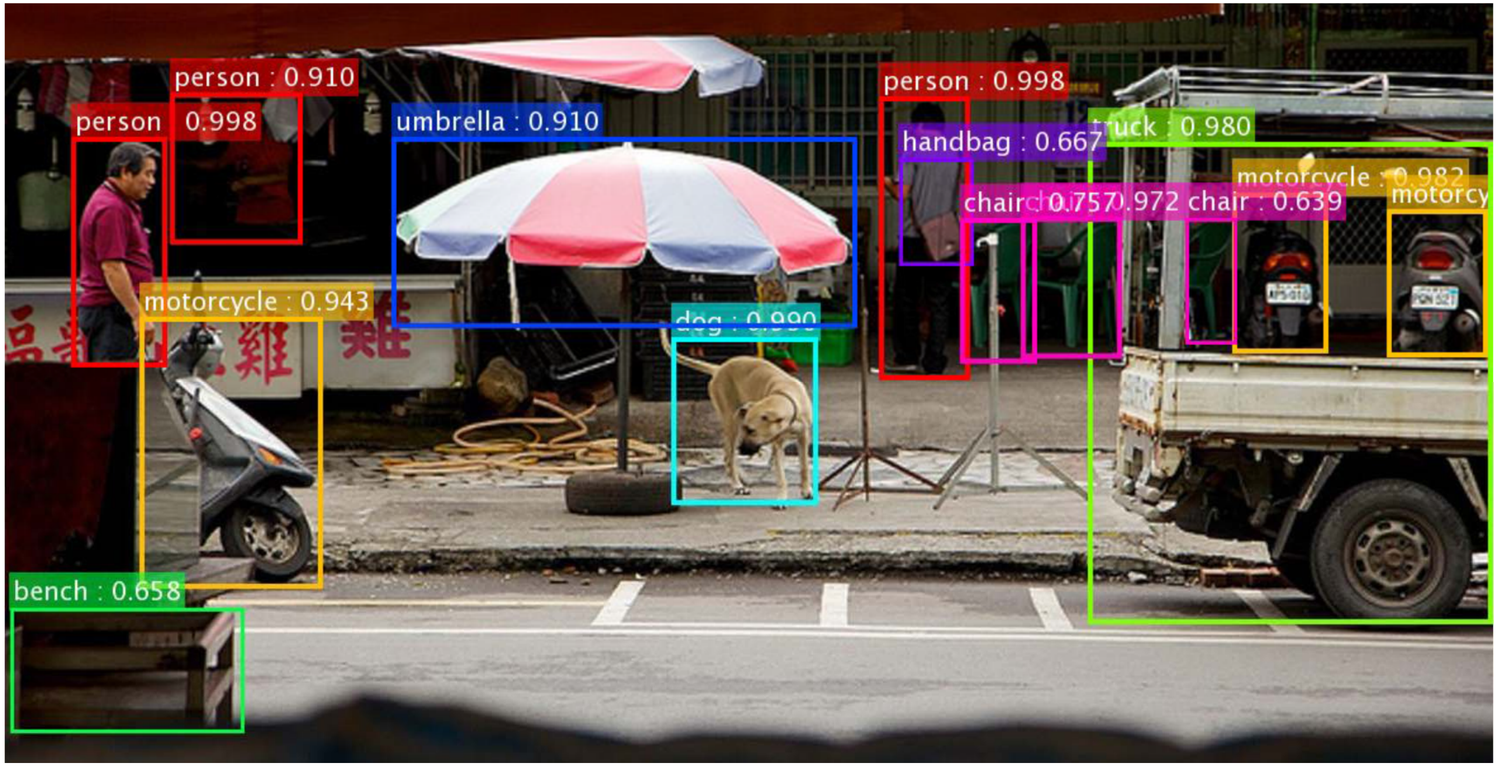
task	2nd-place winner	MSRA	margin (relative)
ImageNet Localization (top-5 error)	12.0	9.0	27%
ImageNet Detection (mAP@.5)	53.6	62.1	16%
COCO Detection (mAP@.5:.95)	33.5	37.3	11%
COCO Segmentation (mAP@.5:.95)	25.1	28.2	12%

absolute 8.5% better!

- Based on ResNet-101
- Existing techniques can use residual networks or features from it

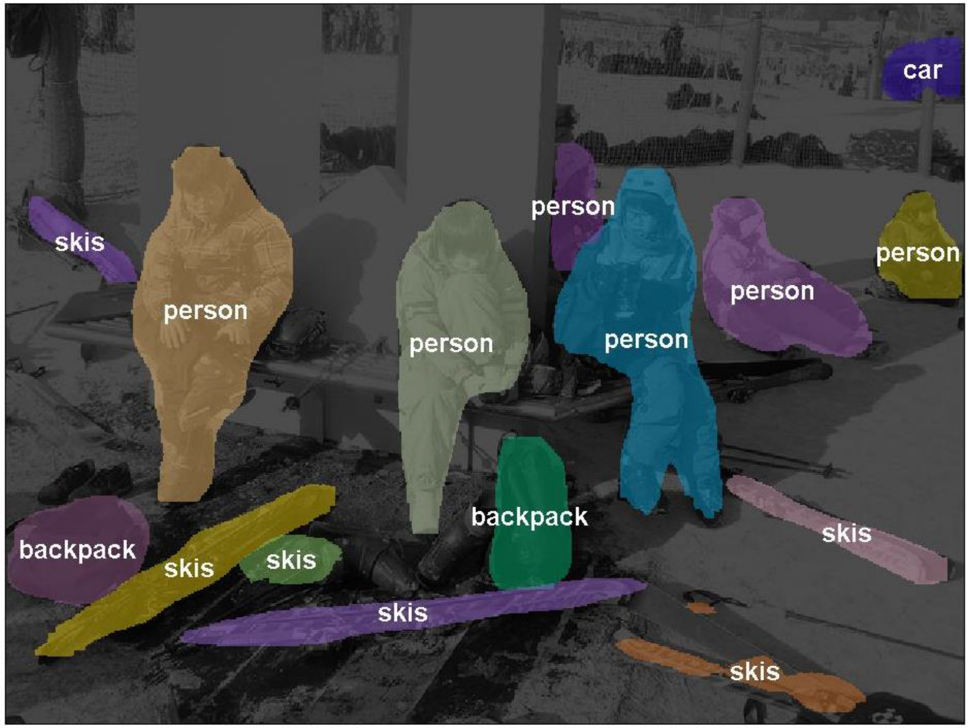
Qualitative Result

- Object detection
  - Faster R-CNN + ResNet



Qualitative Results

- Instance Segmentation



DCNNs and Vision

2012 onwards: all about DCNNs

If you have a hammer, you treat everything like a nail

-Classification & Detection

-Semantic Segmentation

-Boundary Detection

-Feature Descriptors

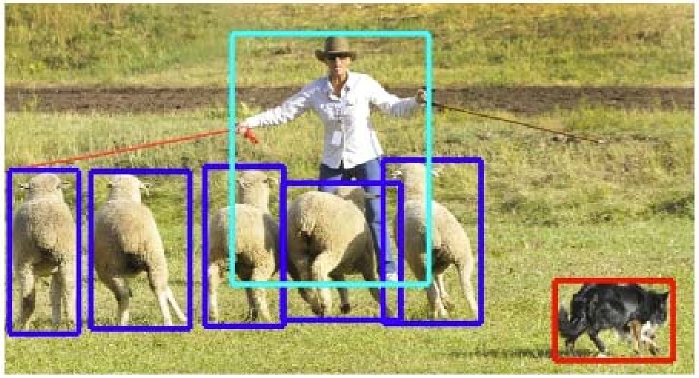
-…

Semantic segmentation task

Beyond detection

http://mscoco.org/ **Microsoft**

(a) Image classification



(b) Object localization

(c) Semantic segmentation

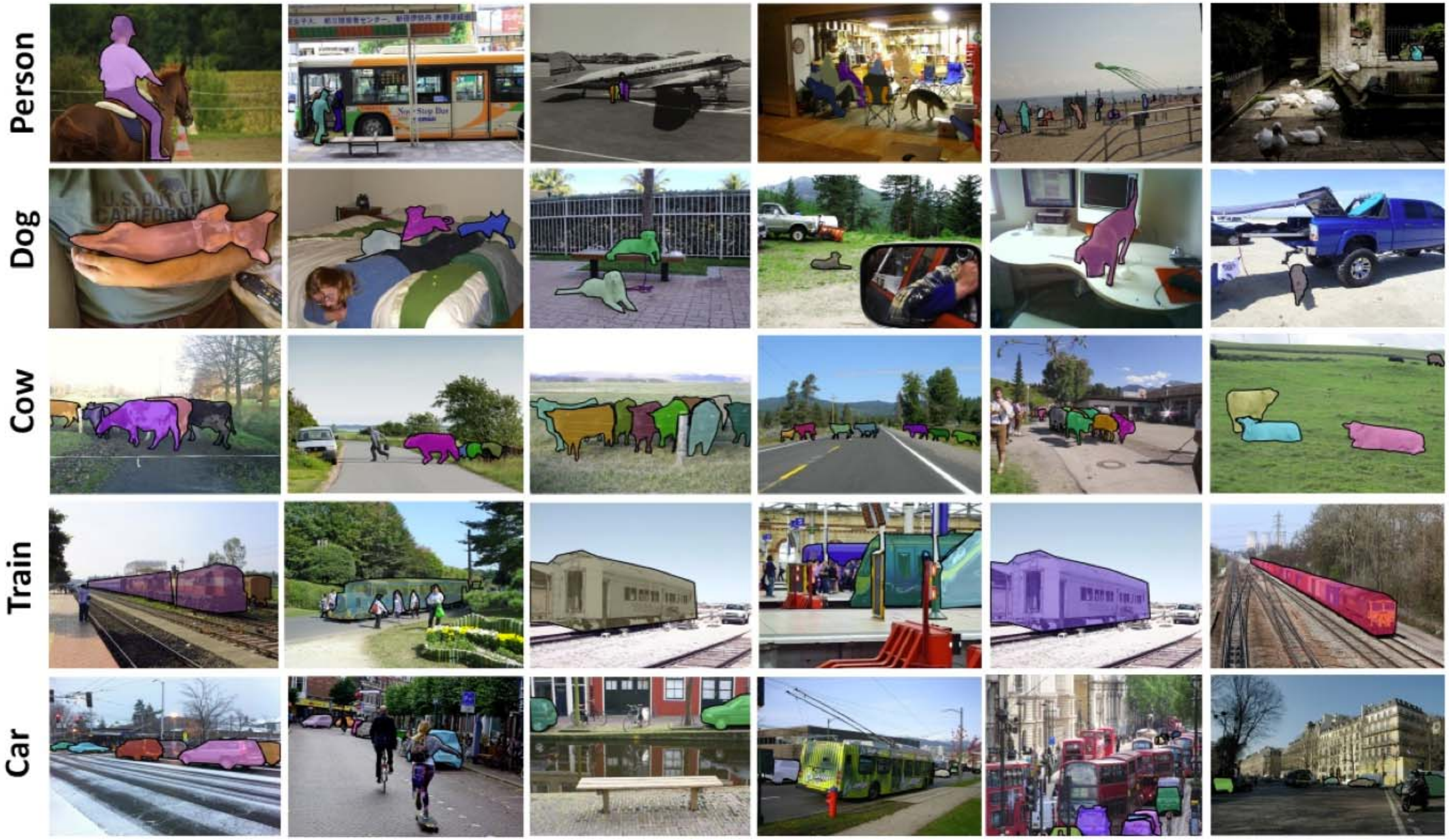
(d) This work

Beyond detection

http://mscoco.org/ **Microsoft** **Pascal VOC**

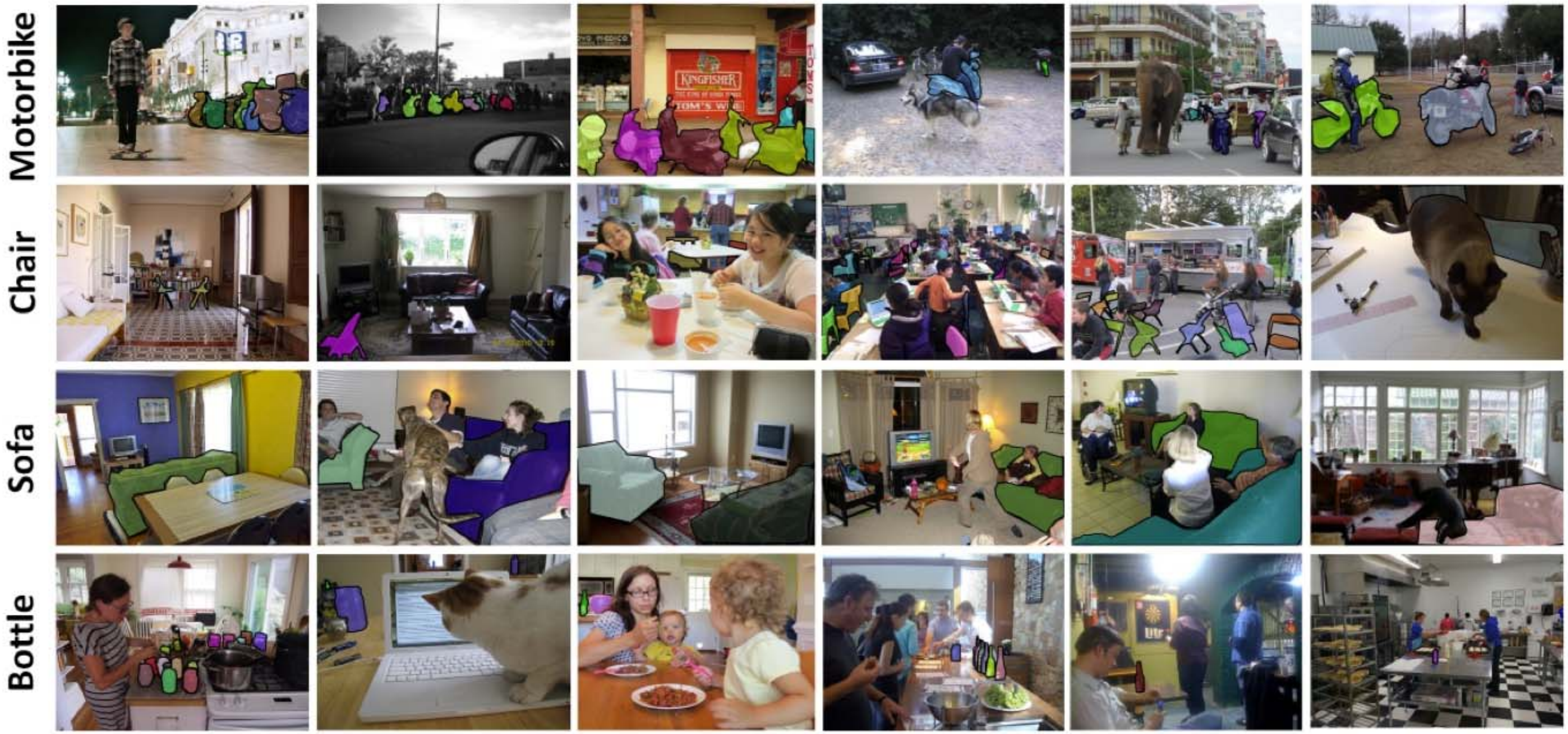
Beyond detection

http://mscoco.org/ **Microsoft**



Beyond detection

http://mscoco.org/ **Microsoft**



Fully Convolutional Networks for Segmentation

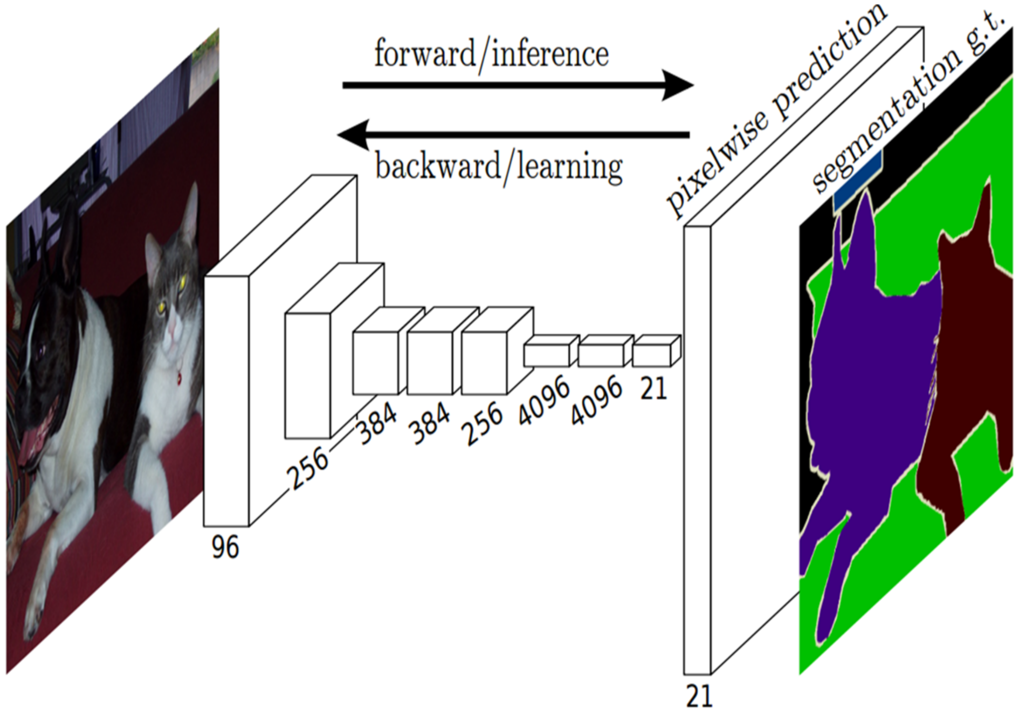
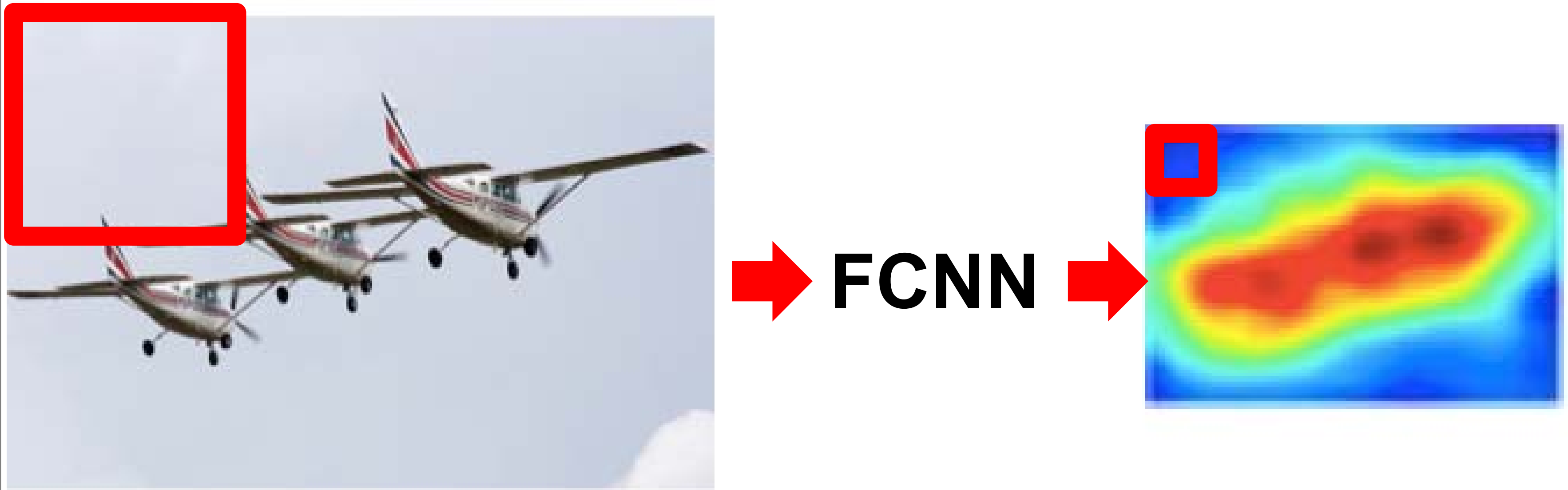


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation.
***CVPR*, 2015**

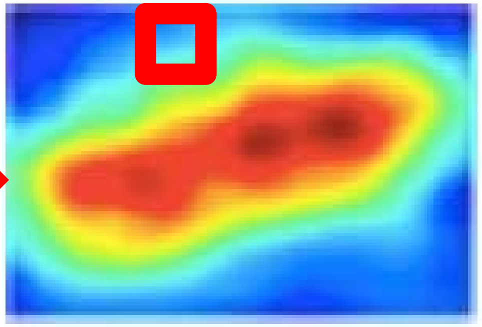
Fully convolutional neural networks



Fully convolutional neural networks

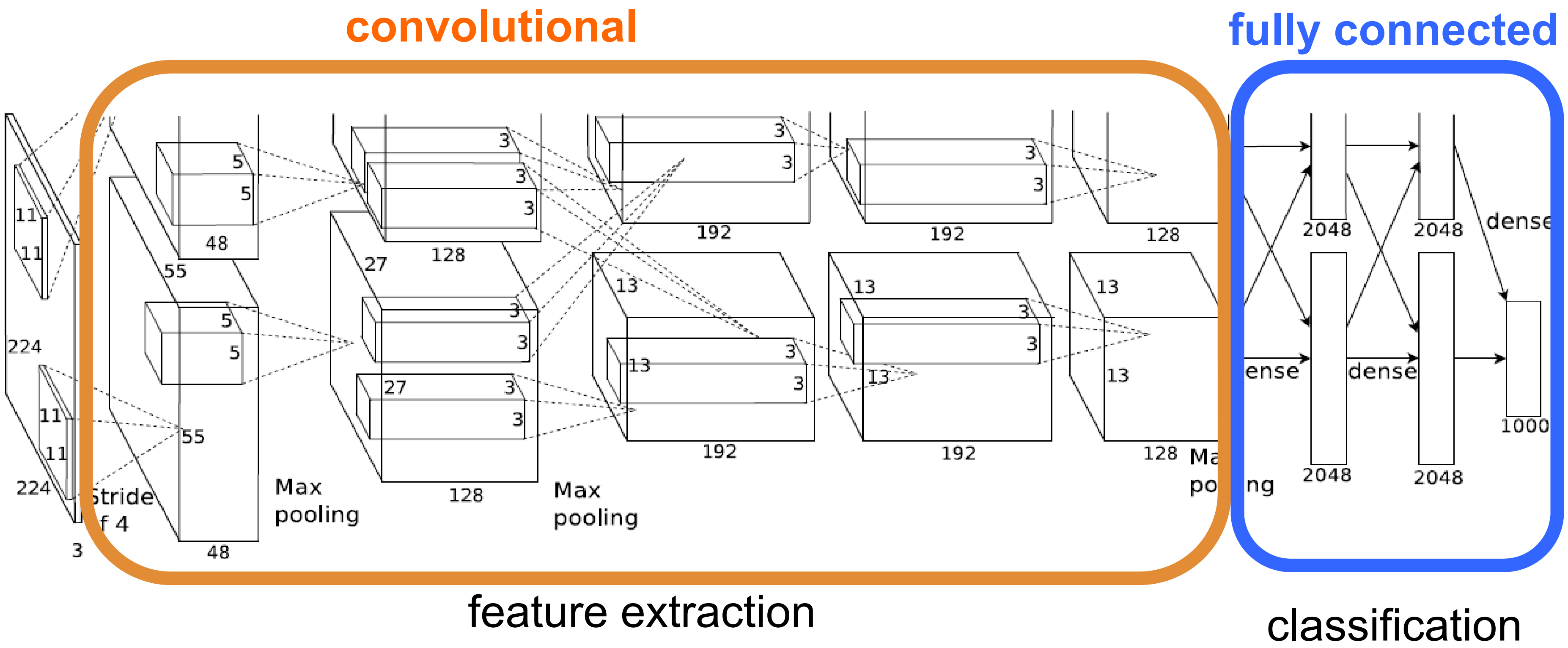
Fully convolutional neural networks

FCNN



Fully convolutional neural networks

Convolutional/Fully Connected DCNN layers



AlexNet

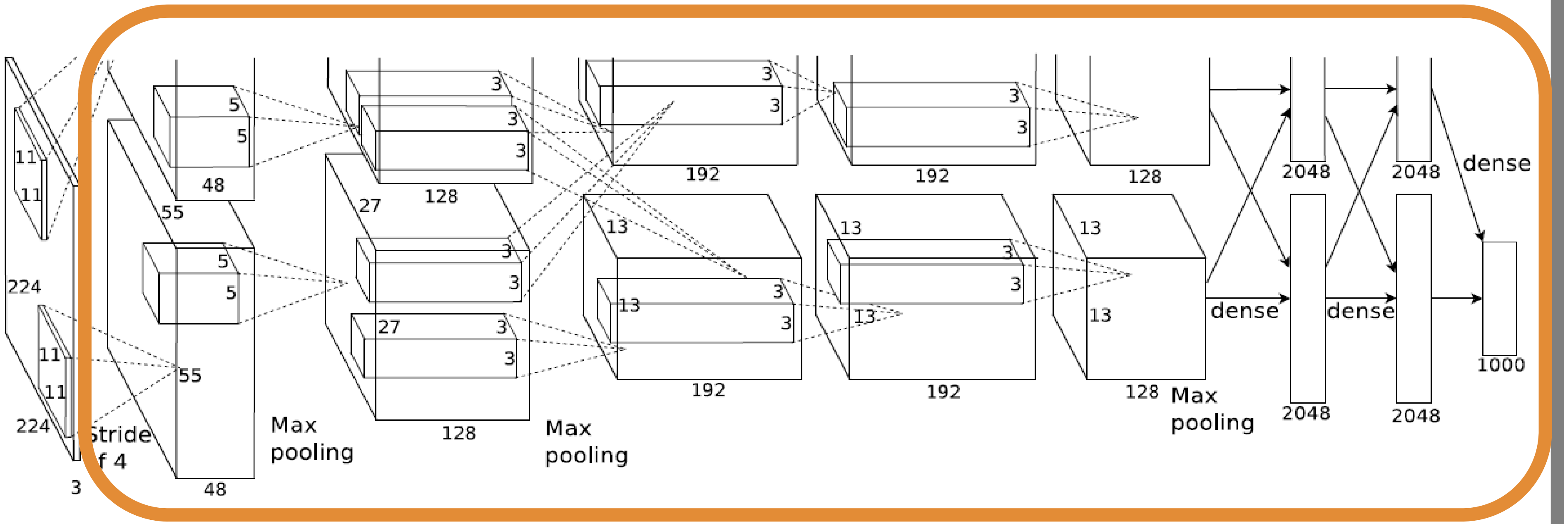
A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. NIPS13

VGG network

K. Simonyan and A. Zisserman. Very deep CNNs for large-scale image recognition, ICLR 2015

Fully convolutional neural networks

convolutional



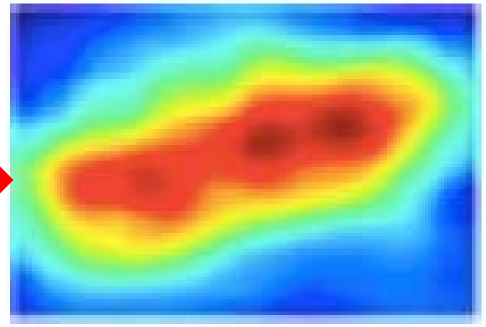
Fully connected layers: 1x1 spatial convolution kernels

“FCNNs” (2015) or "Space Displacement Neural Nets” (1998)

Y. LeCun, et al, Gradient-Based Learning Applied to Document Recognition, Proc. IEEE98
Sermanet et al, Overfeat: Integrated Recognition, Localization and Detection, ICLR 14
G. Papandreou et al, Modelling Deformations in Deep Learning, CVPR 15
J. Long, et al., Fully Convolutional Networks for Semantic Segmentation, CVPR15

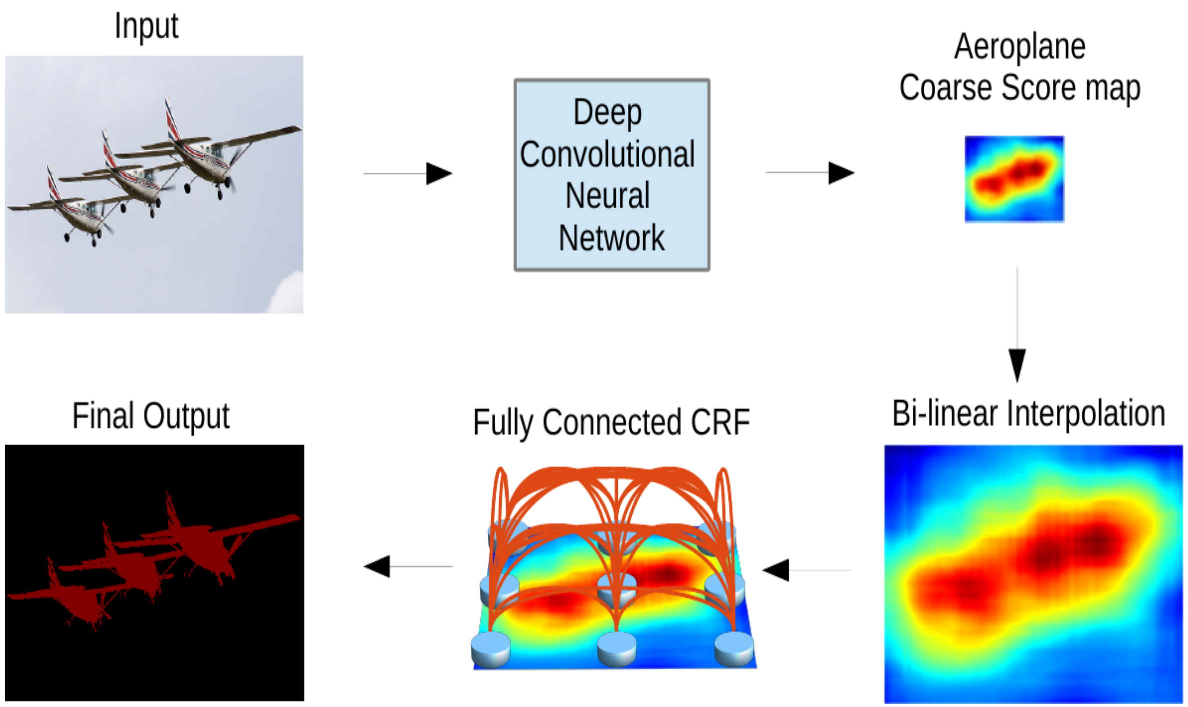
Fully convolutional neural networks

➡ FCNN ➡



Fast (shared convolutions)
Simple (dense)

Deeplab: Atrous Convolution + Structured Prediction




L-C. Chen


G. Papandreou


A. Yuille


K. Murphy

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. Yuille, Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs, ICLR 2015
L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. Yuille, Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, PAMI 2016
L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking Atrous Convolution for Semantic Image Segmentation, Arxiv 2017

Comparison to state-of-the-art, 2014

Method	mean IOU (%)
MSRA-CFM	61.8
FCN-8s	62.2
TTI-Zoomout-16	64.4
DeepLab-CRF (our)	66.4
DeepLab-MSc-CRF (our)	67.1

Pre-CNN: Up to 50% → **CNN: 60-64%** → **CNN + CRF: >67%**

Pascal Train: 67% → **Coco + Pascal 71%**

G. Papandreou, et al, Weakly- and Semi-Supervised Learning of a DCNN for Semantic Image Segmentation, arxiv 2015

Updated results (Deeplab v2, 2016)

Method	mIOU
DeepLab-CRF-LargeFOV-COCO [58]	72.7
MERL_DEEP_GCRF [88]	73.2
CRF-RNN [59]	74.7
POSTECH_DeconvNet_CRF_VOC [61]	74.8
BoxSup [60]	75.2
Context + CRF-RNN [74]	75.3
QO_4^{mres} [66]	75.5
DeepLab-CRF-Attention [17]	75.7
CentraleSuperBoundaries++ [18]	76.0
DeepLab-CRF-Attention-DT [63]	76.3
H-ReNet + DenseCRF [89]	76.8
LRR_4x_COCO [90]	76.8
DPN [62]	77.5
Adelaide_Context [40]	77.8
Oxford_TVG_HO_CRF [87]	77.9
Context CRF + Guidance CRF [91]	78.1
Adelaide_VeryDeep_FCN_VOC [92]	79.1
DeepLab-CRF (ResNet-101)	79.3
Ensemble DeepLab-CRF (ResNet-101)	79.6

TABLE 5: Performance on PASCAL VOC 2012 *test* set. We have added some results from recent arXiv papers on top of the official leadearboard results.

Updated results (Deeplab v3, 2017)

Method	mIOU
Adelaide_VeryDeep_FCN_VOC [76]	79.1
LRR_4x_ResNet-CRF [21]	79.3
DeepLabv2-CRF [10]	79.7
CentraleSupelec Deep G-CRF [7]	80.2
HikSeg_COCO [71]	81.4
Deep Layer Cascade (LC) [45]	82.7
TuSimple [75]	83.1
Large_Kernel_Matters [60]	83.6
Multipath-RefineNet [47]	84.2
ResNet-38_MS_COCO [77]	84.9
PSPNet [84]	85.4
IDW-CNN [74]	86.3
DeepLabv3	85.7
DeepLabv3-JFT	86.9

Table 7. Performance on PASCAL VOC 2012 *test* set.

L.-C. Chen, G. Papandreou, F. Schroff, H. Adam , Rethinking Atrous Convolution for Semantic Image Segmentation, Arxiv, 2017
C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *arXiv:1707.02968*, 2017.

Deeplab v2 results

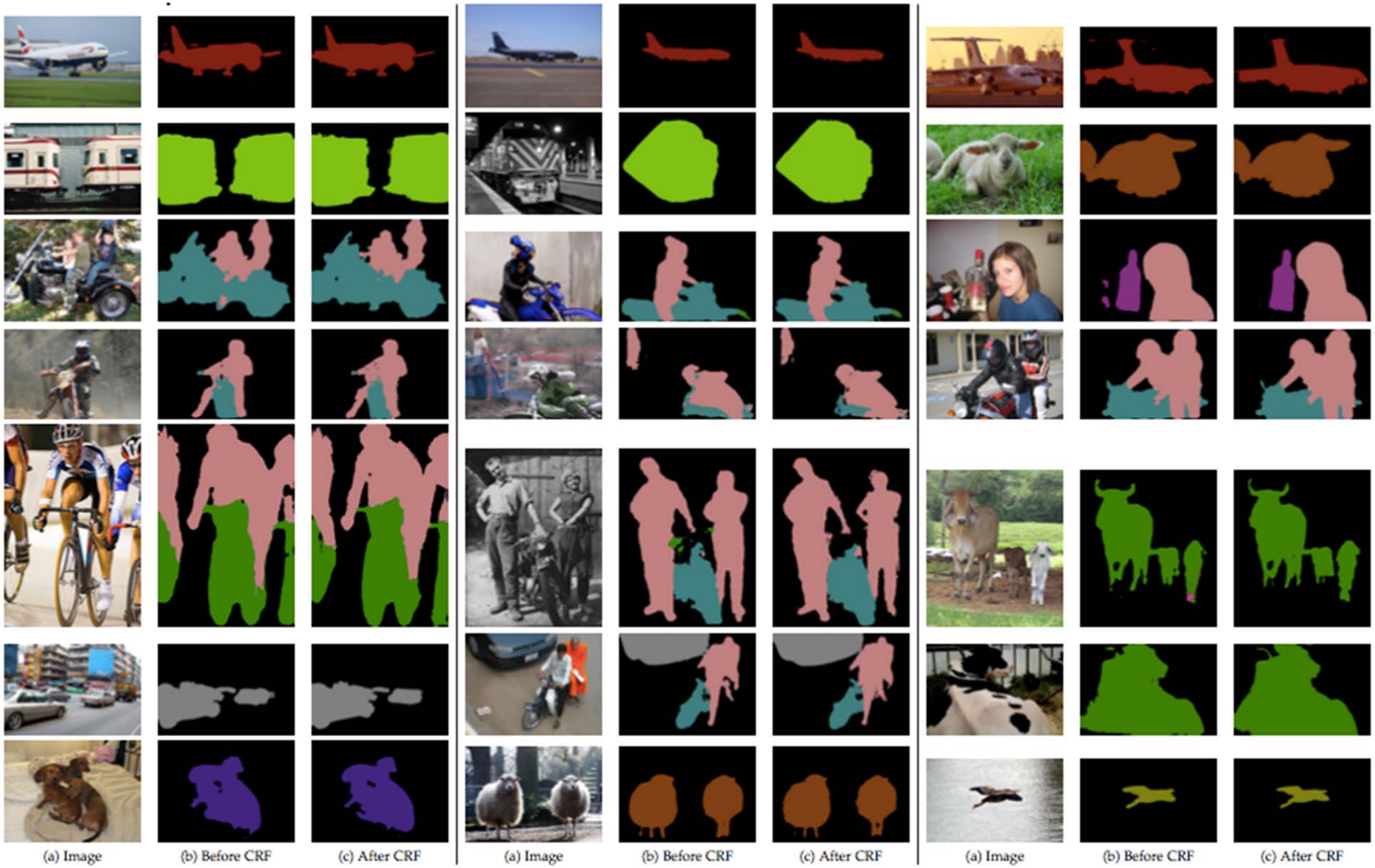
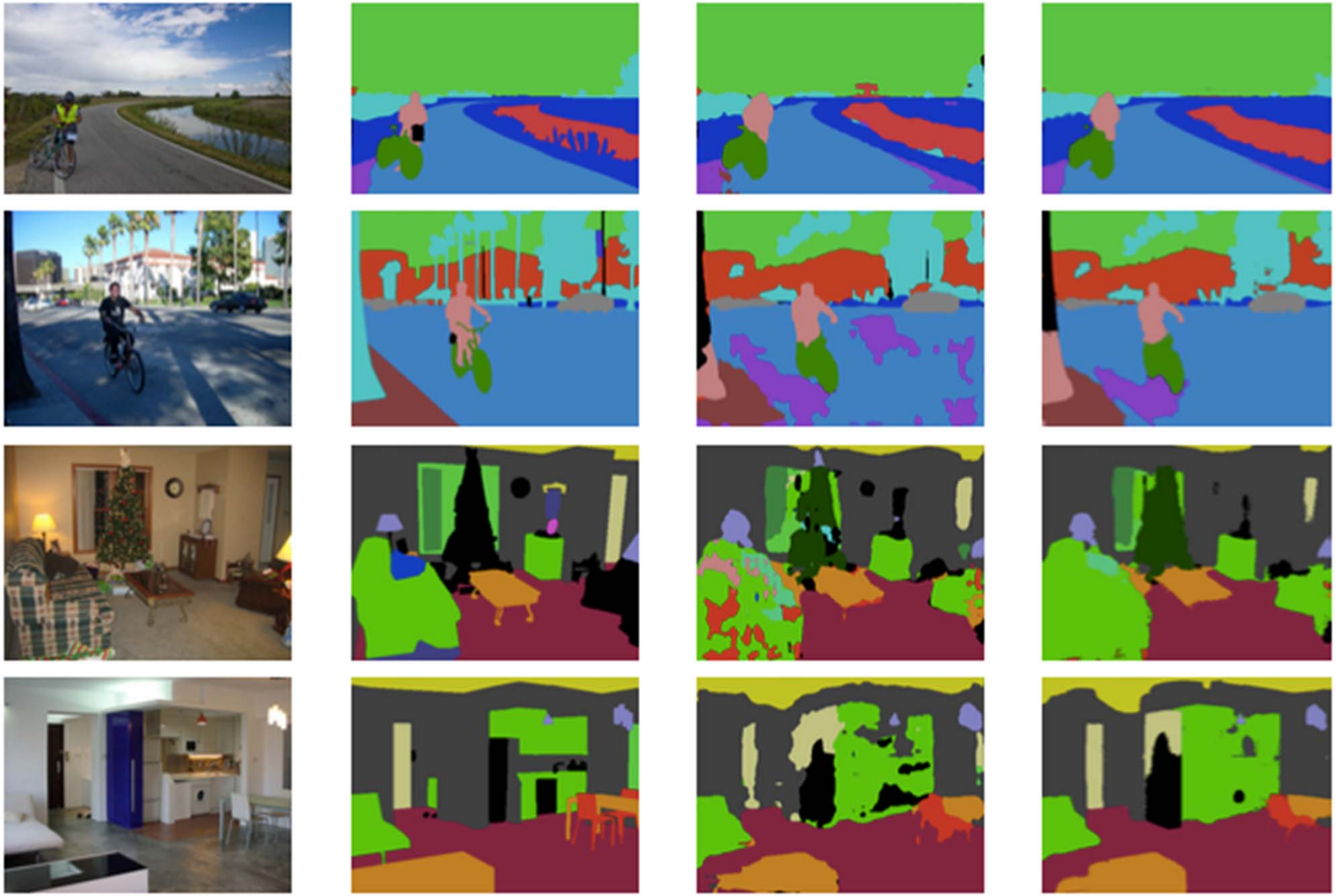


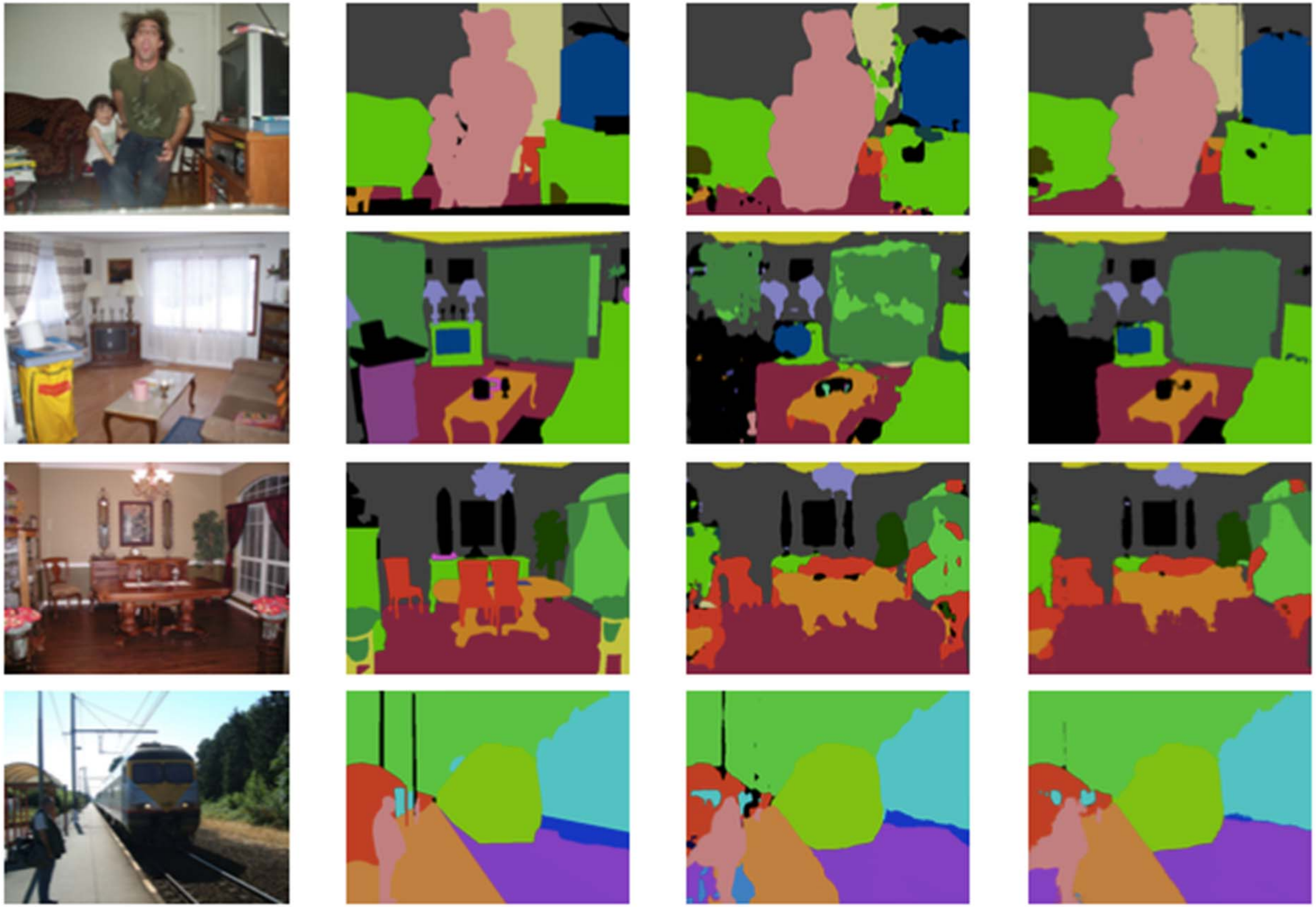
Fig. 7: Visualization of some VOC 2012 *val* results. For each row, we show the input image, the segmentation result before CRF, and the refined segmentation result after Fully Connected CRF (DeepLab-CRF).

Deeplab v2 results



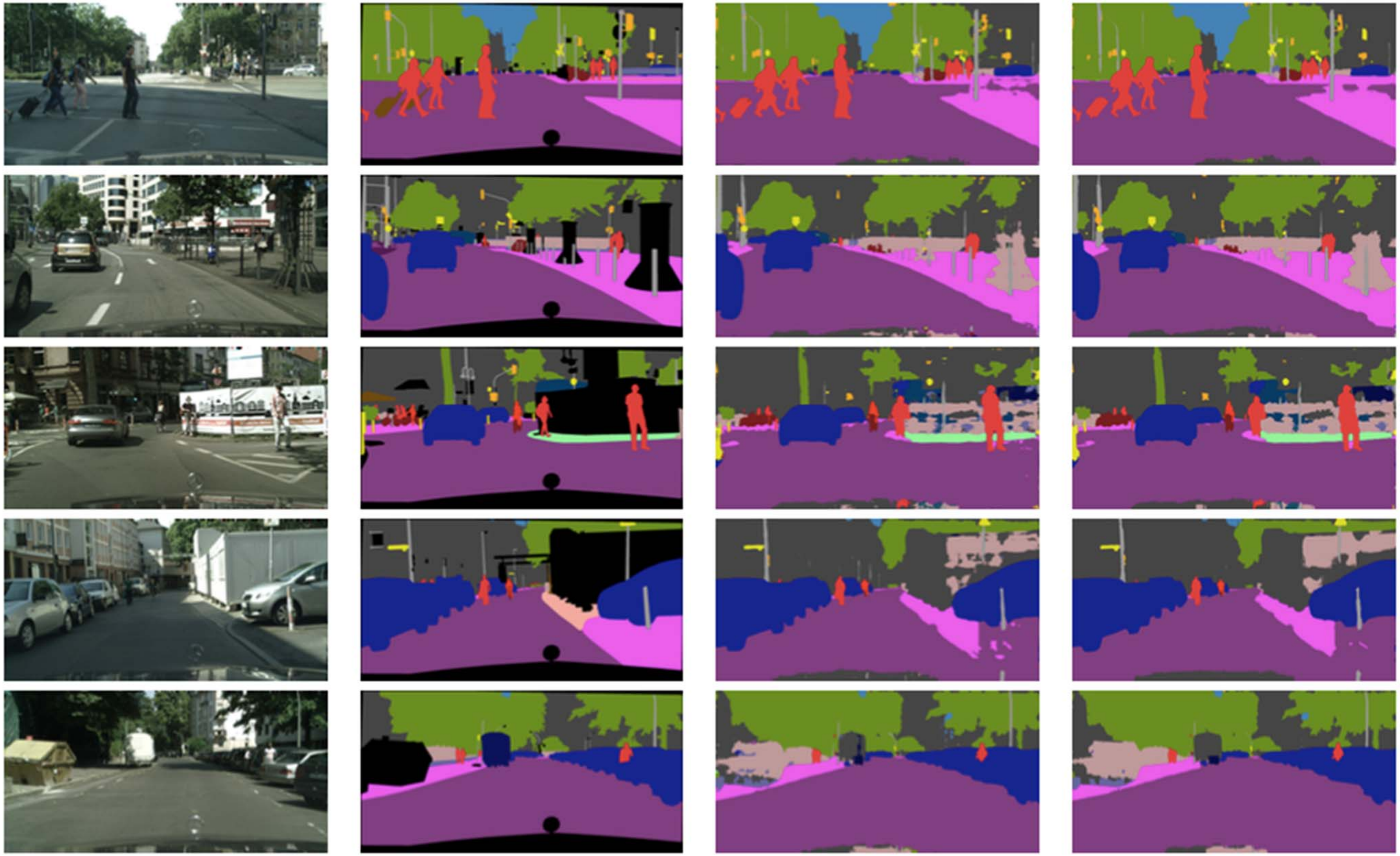
Ground truth FCNN FCNN-DCRF

Deeplab v2 results



Ground truth FCNN FCNN-DCRF

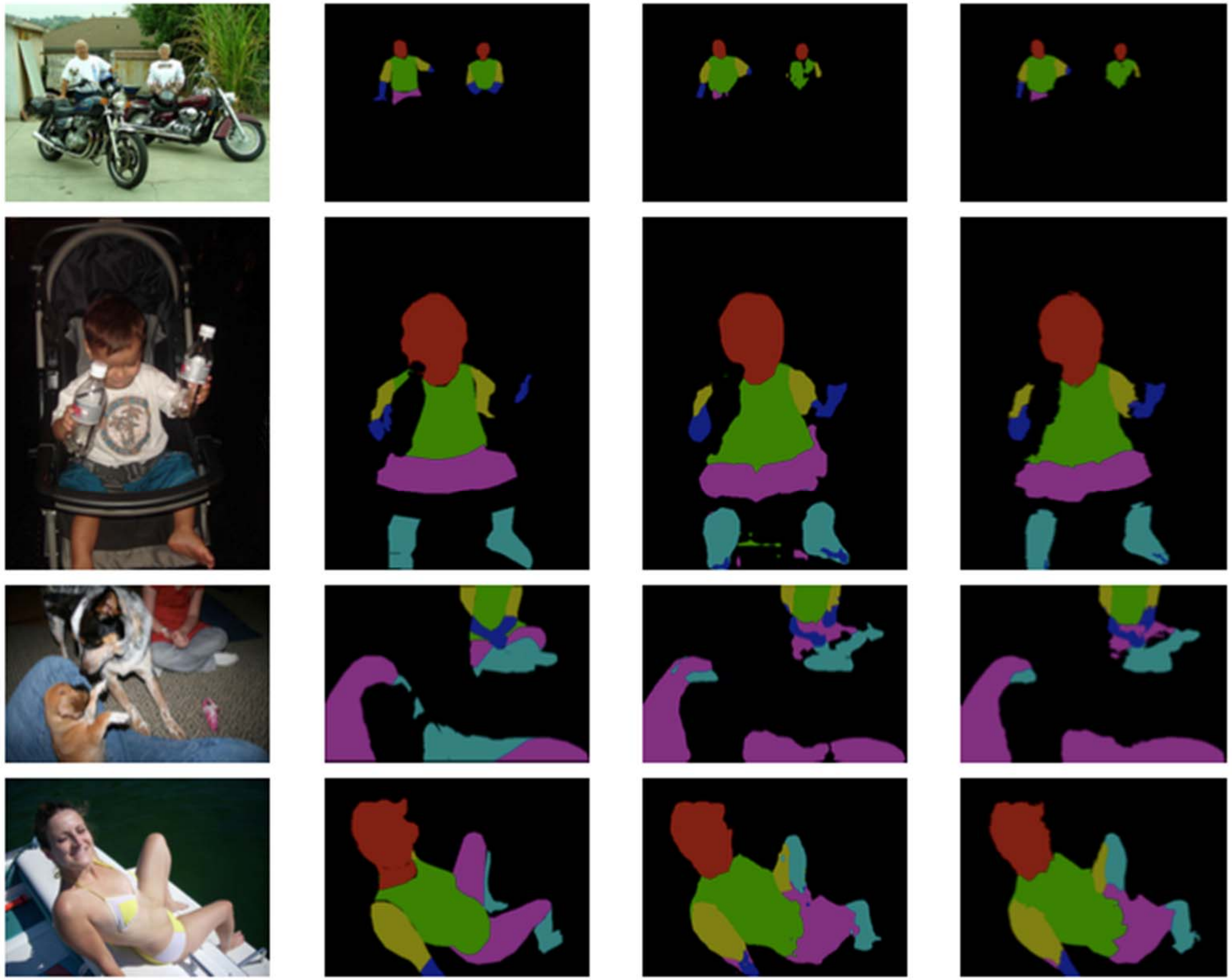
Deeplab v2 results



(a) Image (b) G.T. (c) Before CRF (d) After CRF

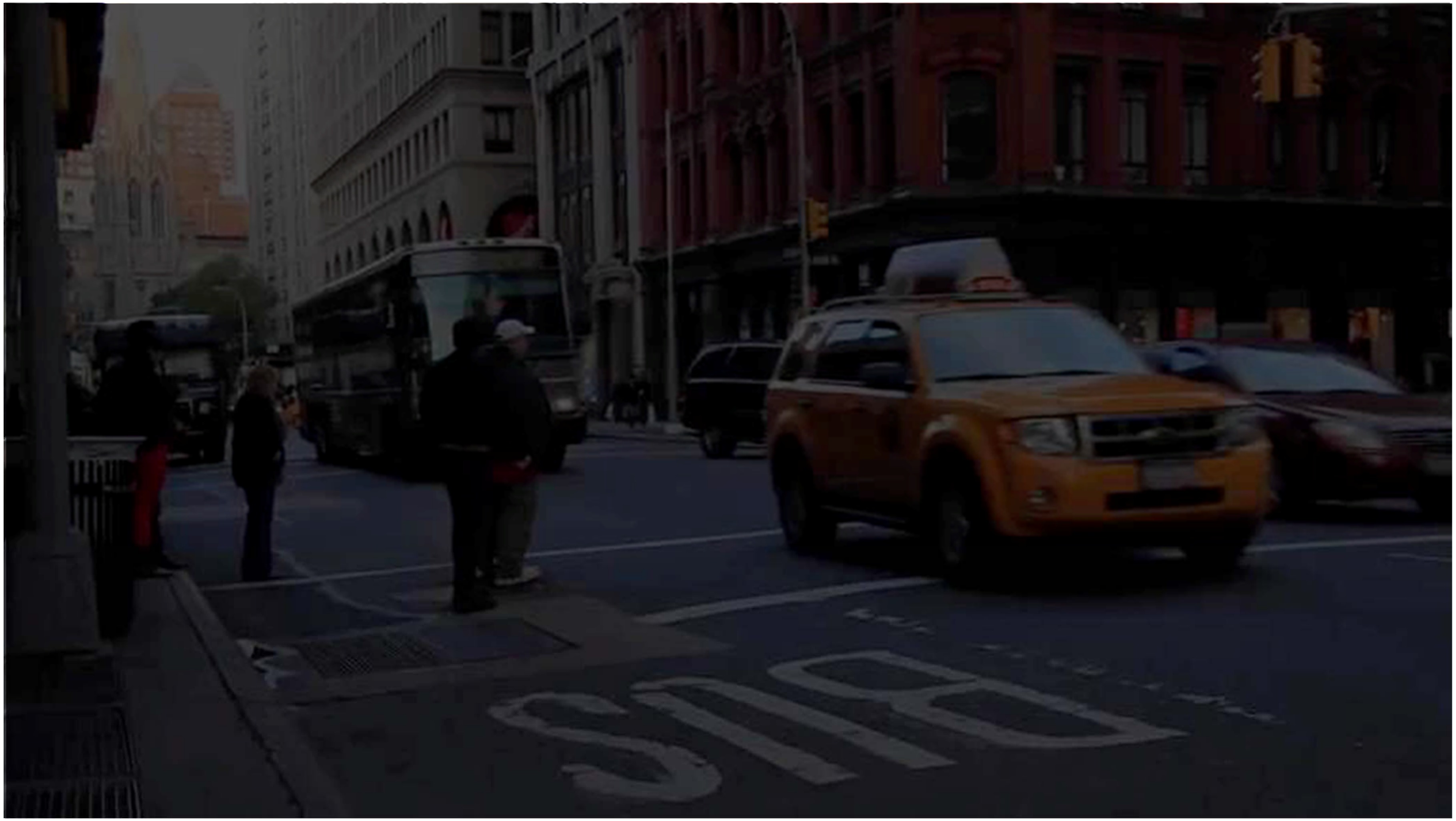
Fig. 11: Visualization results on Cityscapes. For each row, we show the input image, the ground-truth, and our DeepLab results before and after CRF.

Deeplab v2 results

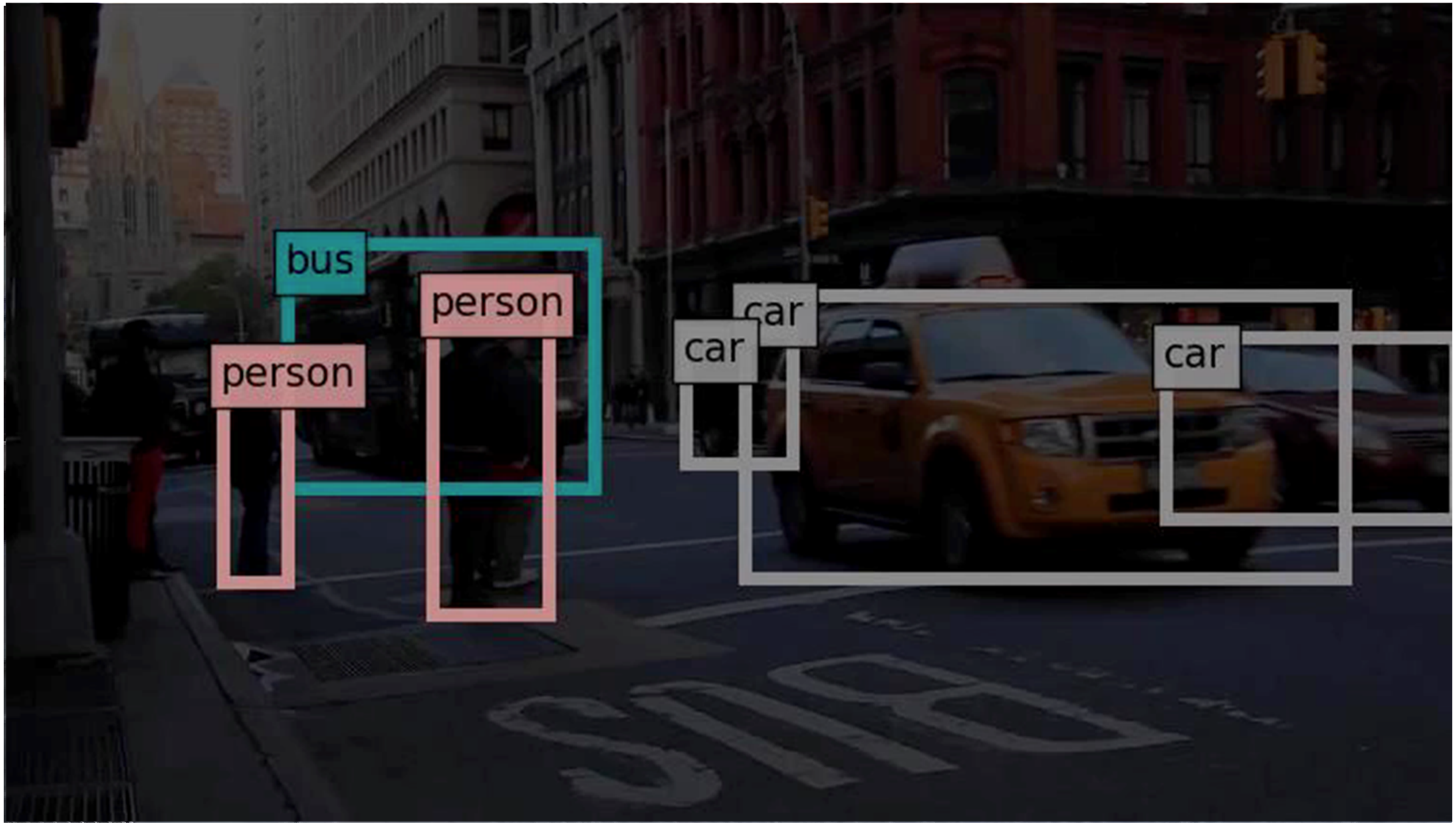


See also: S. Tsogkas, G. Papandreou, I. Kokkinos, and A. Vedaldi, Semantic Part Segmentation using high-level guidance, Arxiv, 2015

What can we get out of an image?

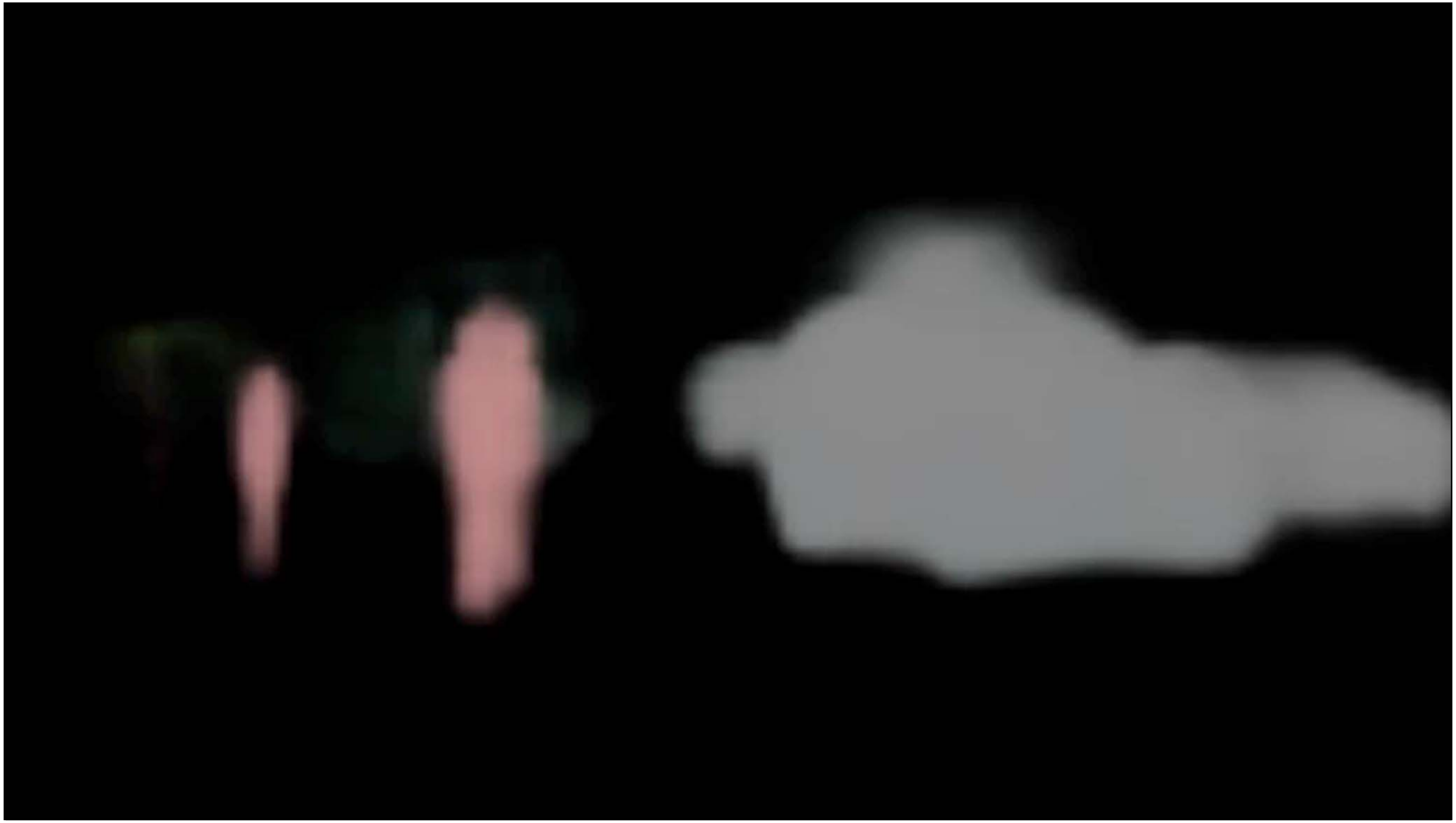


What can we get out of an image?



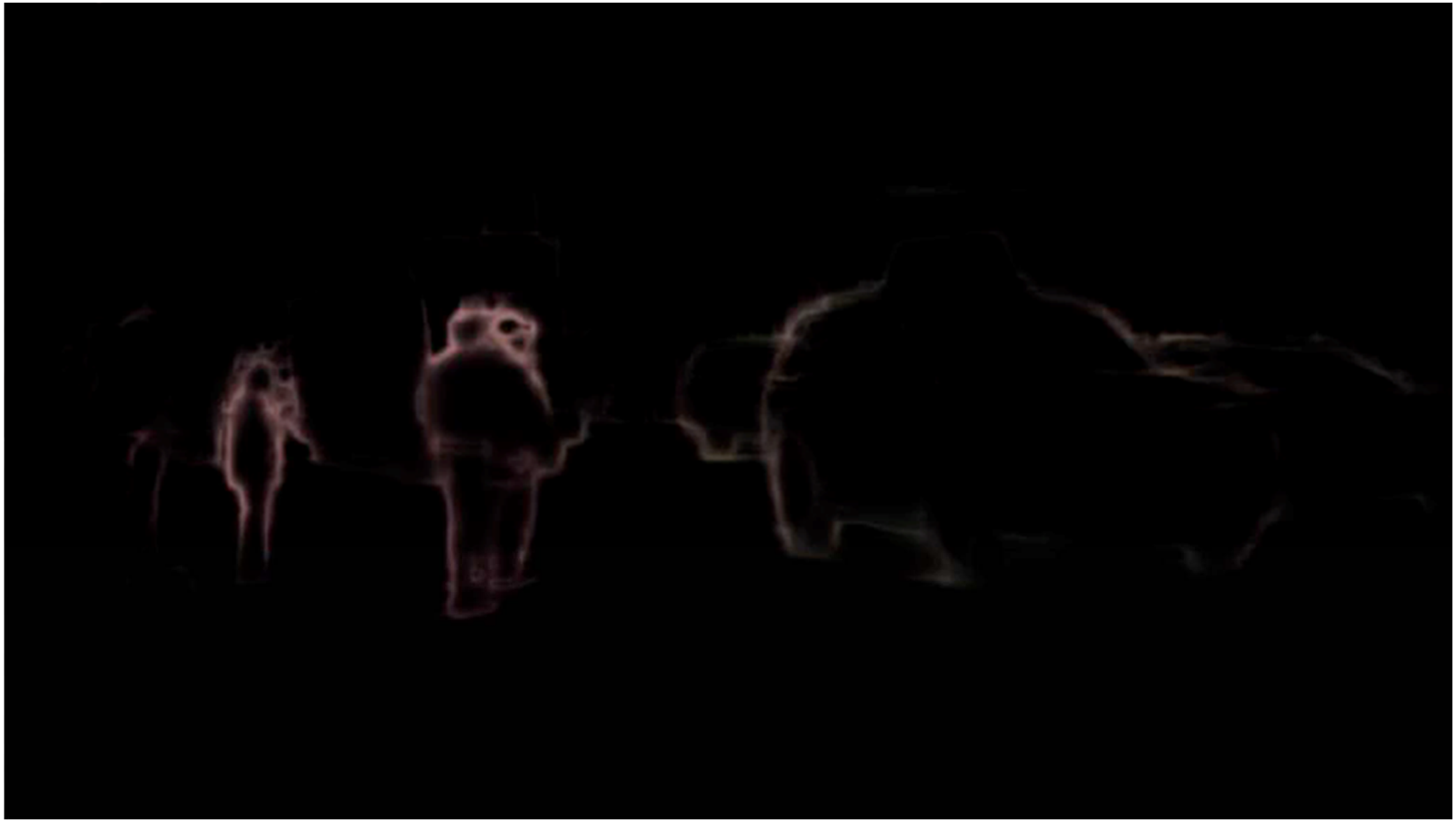
Object detection

What can we get out of an image?



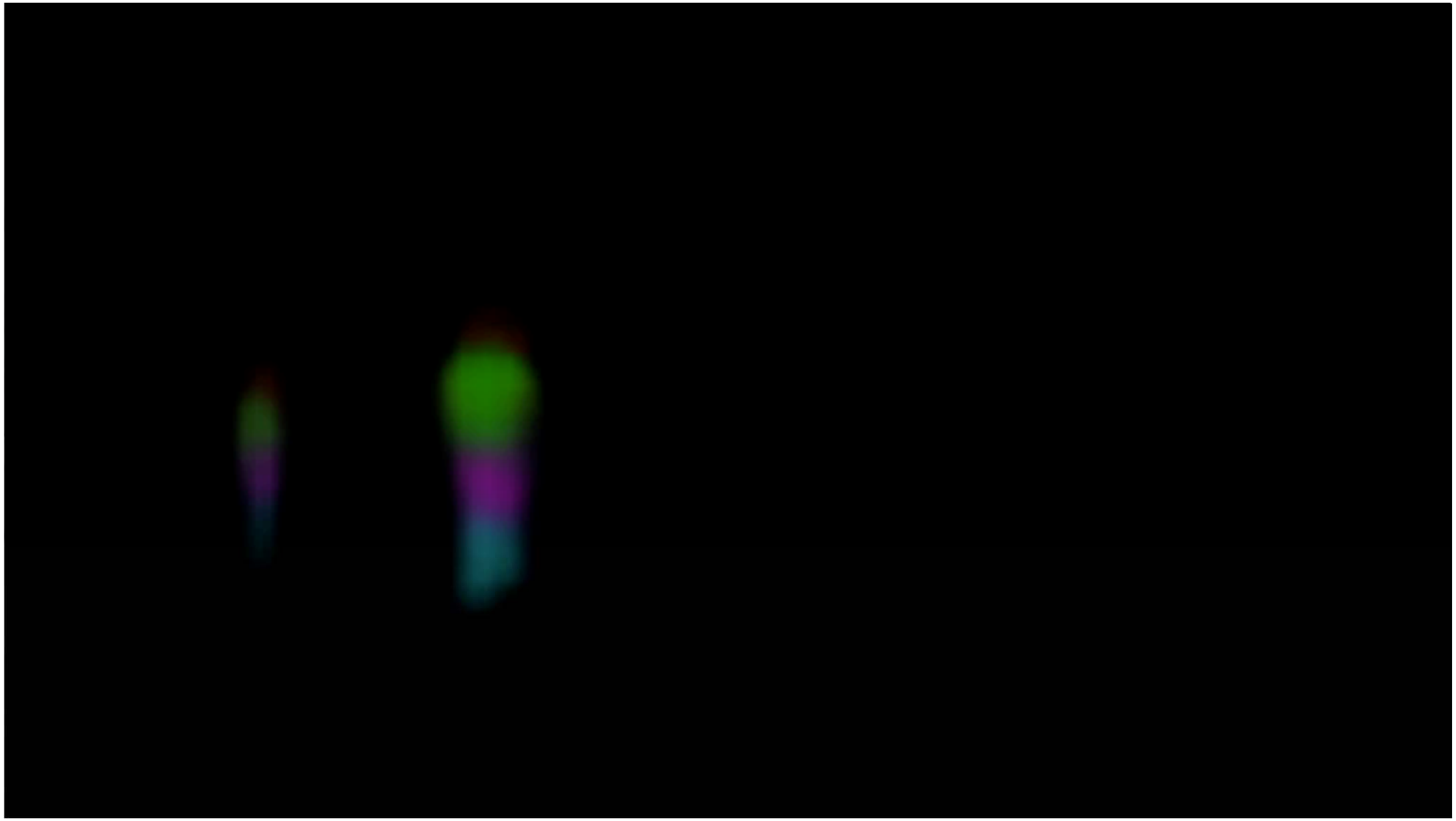
Semantic segmentation

What can we get out of an image?



Semantic boundary detection

What can we get out of an image?



Part segmentation

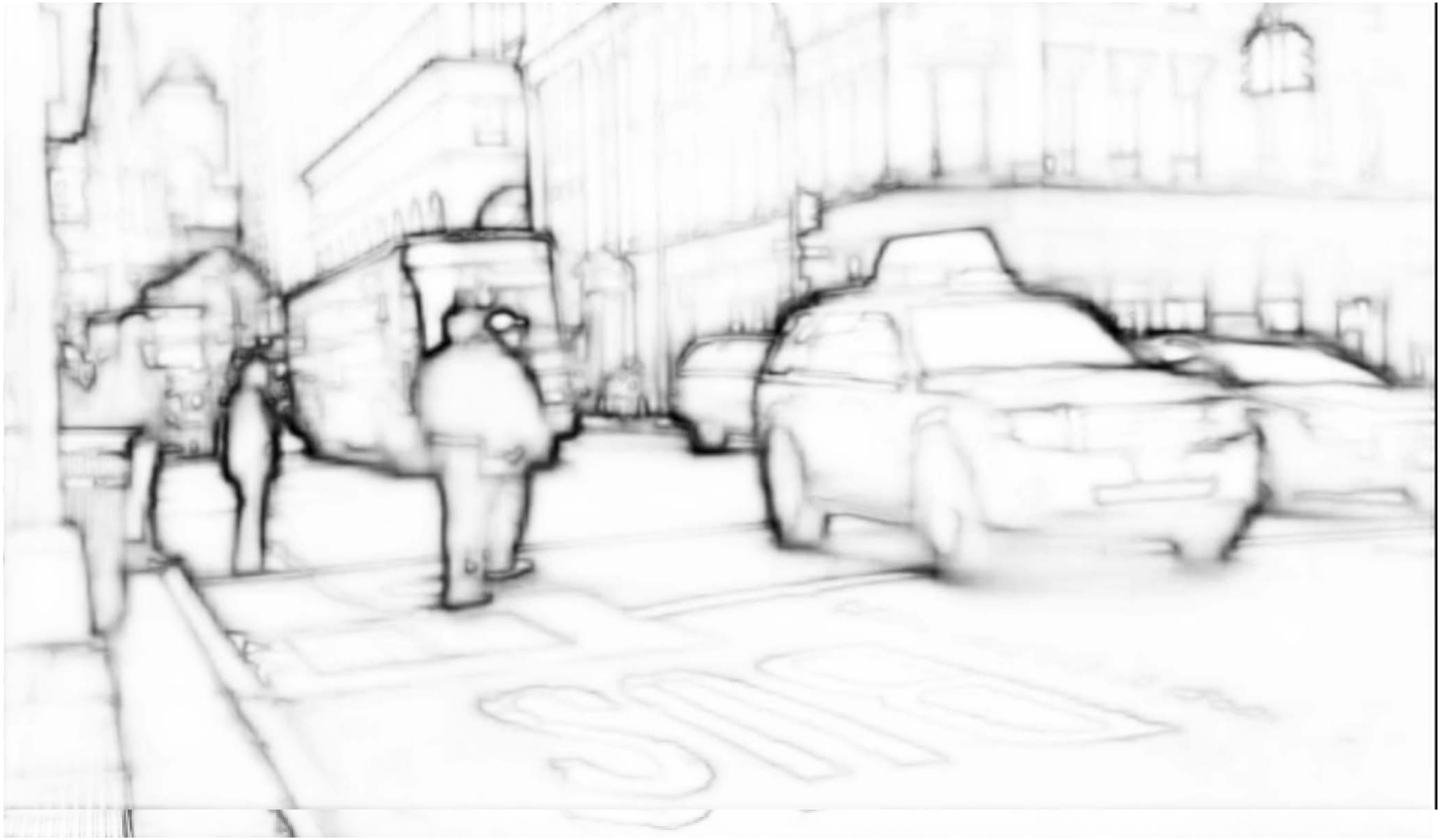
What can we get out of an image?

Surface normal estimation

What can we get out of an image?

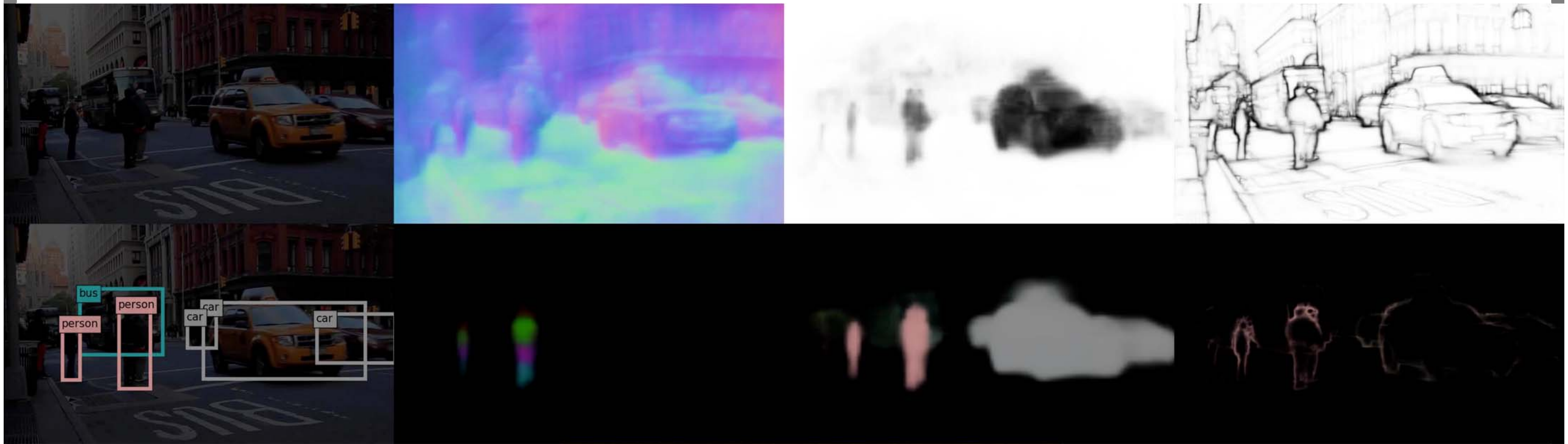
Saliency estimation

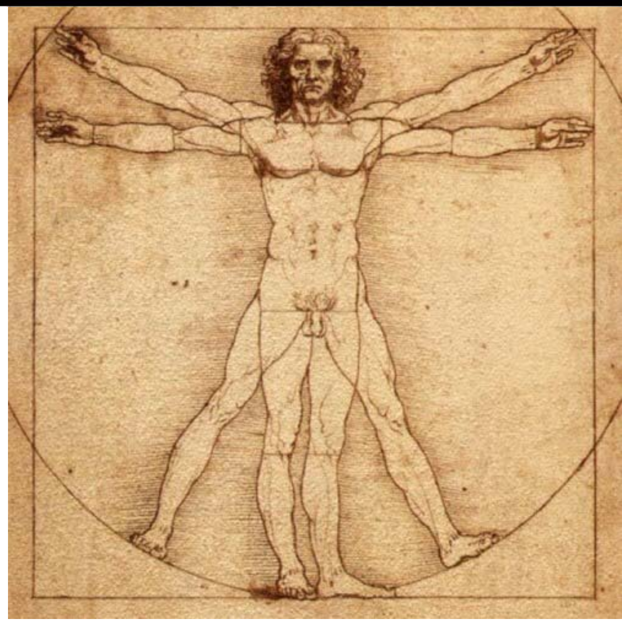
What can we get out of an image?



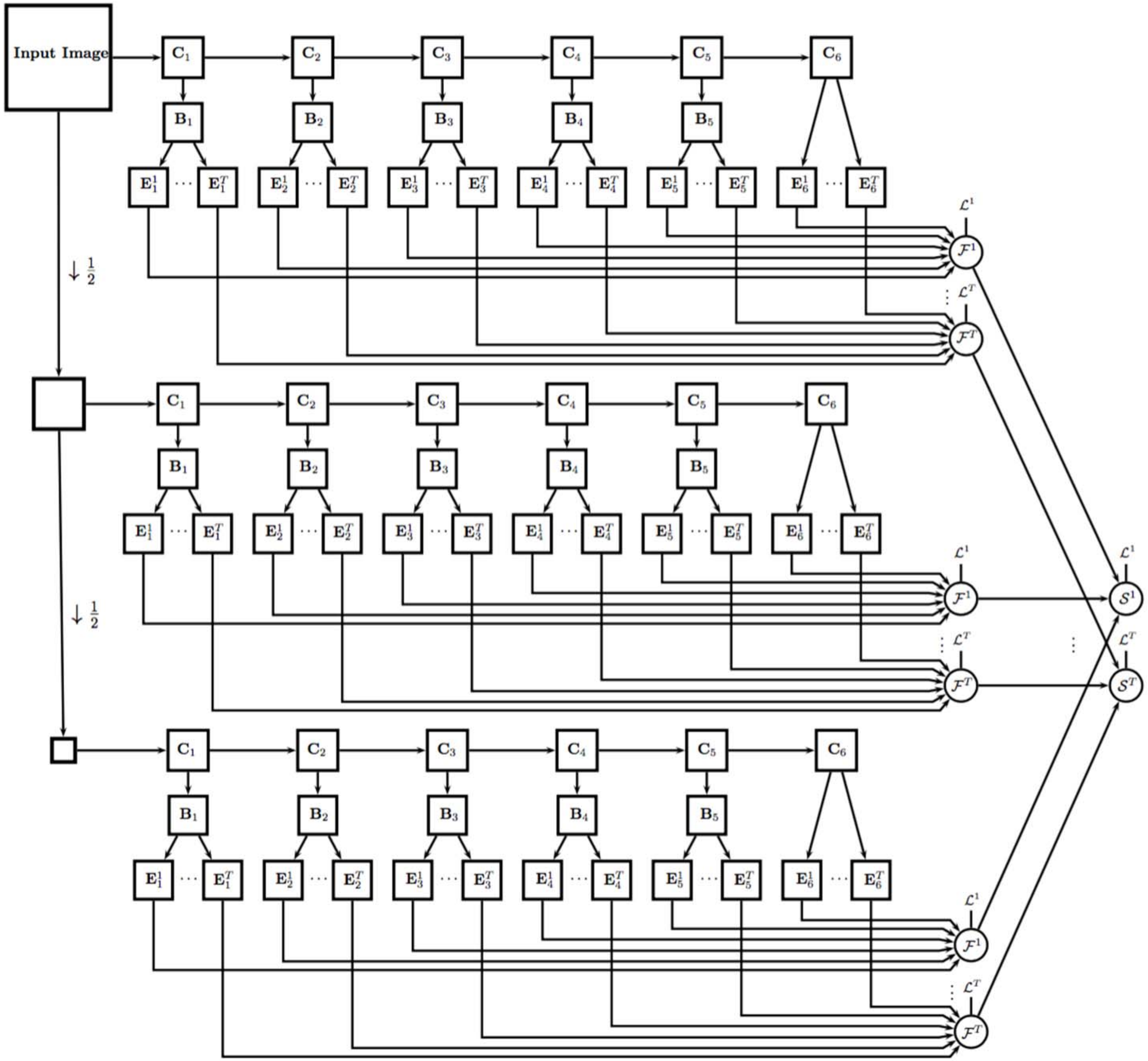
Boundary detection

UberNet: a single “universal” network for all tasks

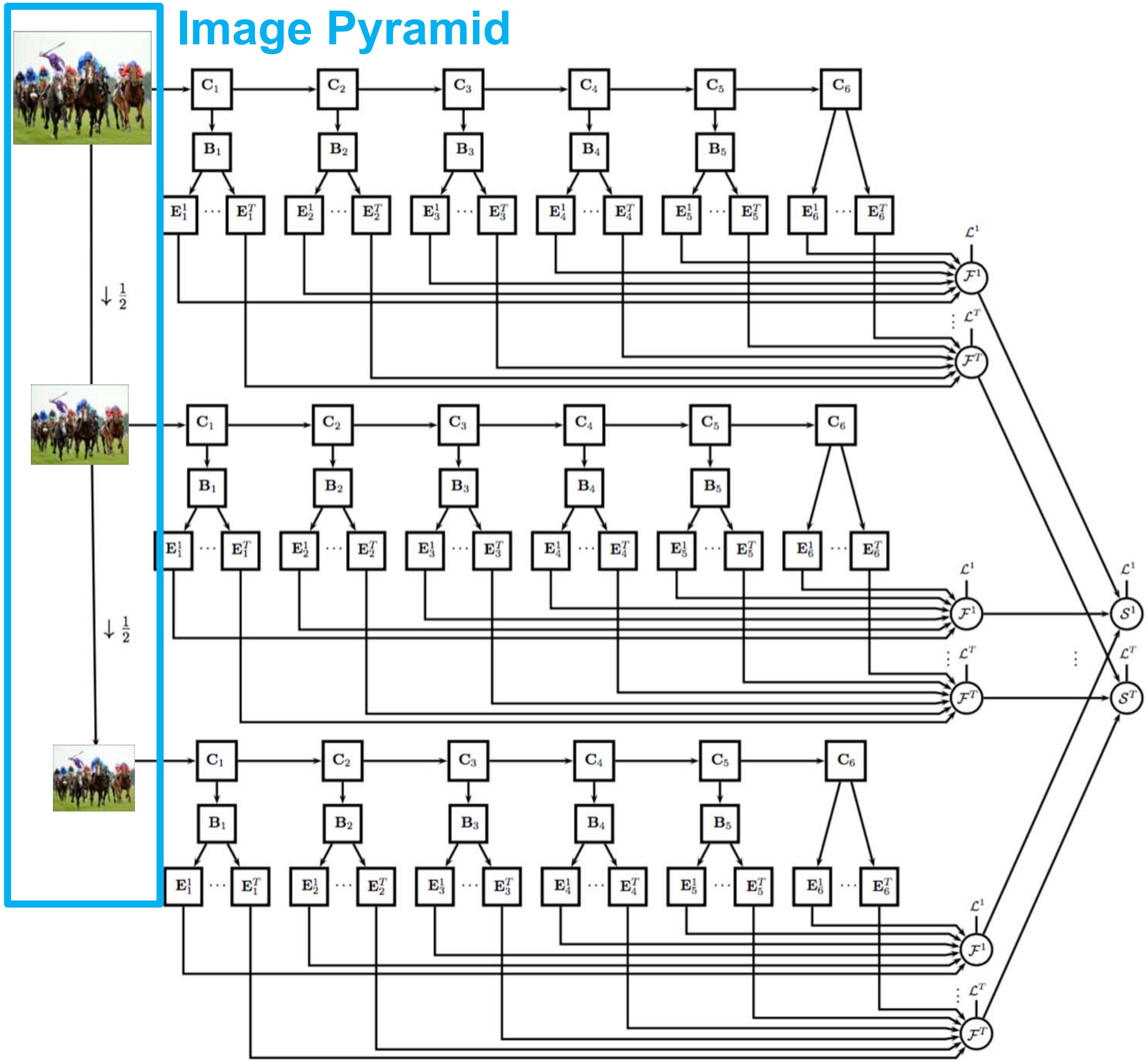




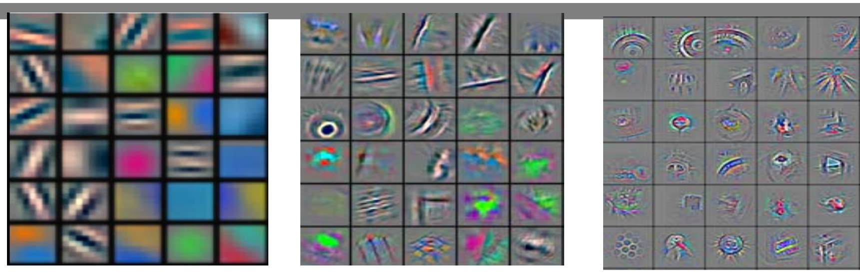
UberNet architecture

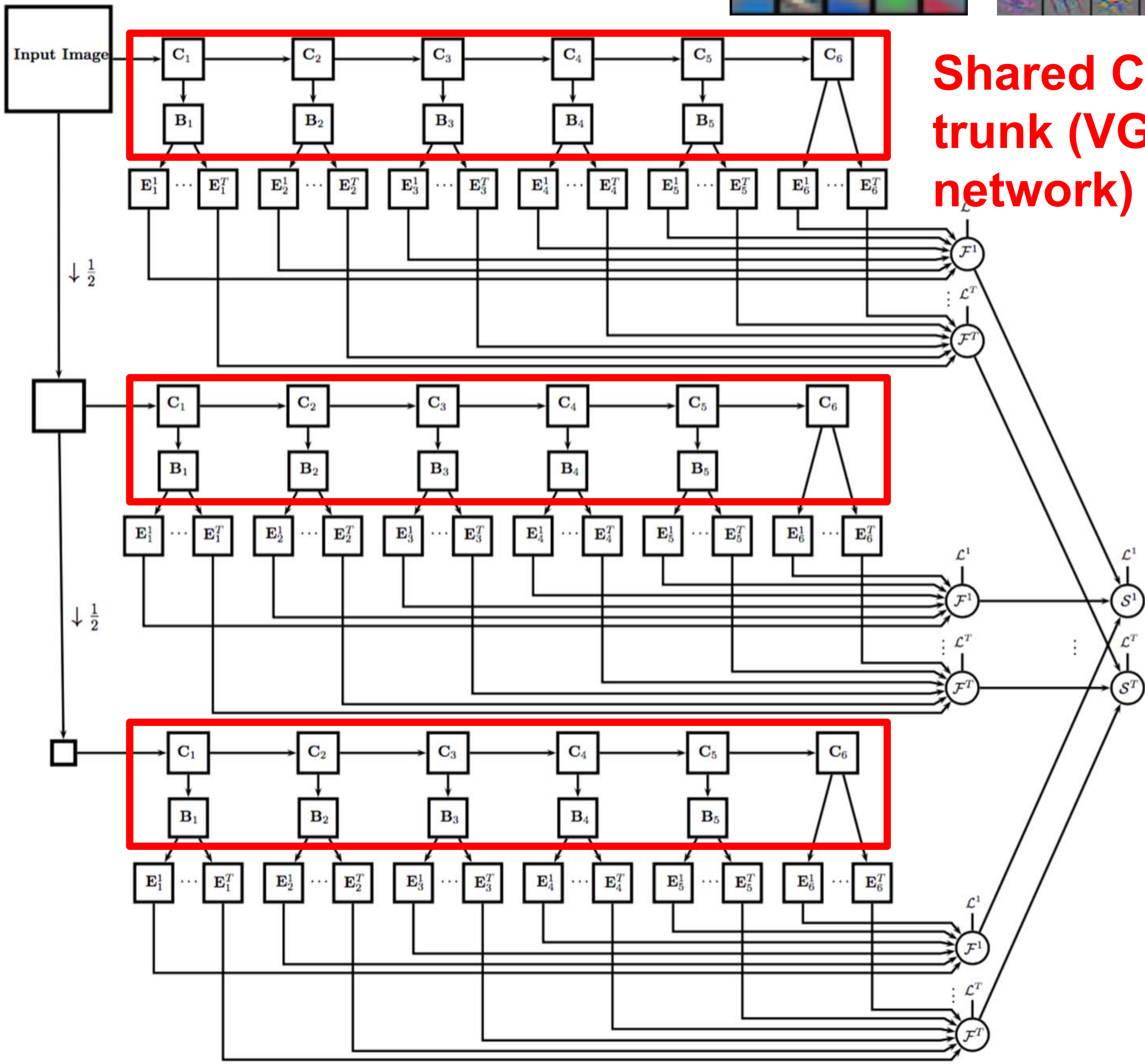


UberNet architecture



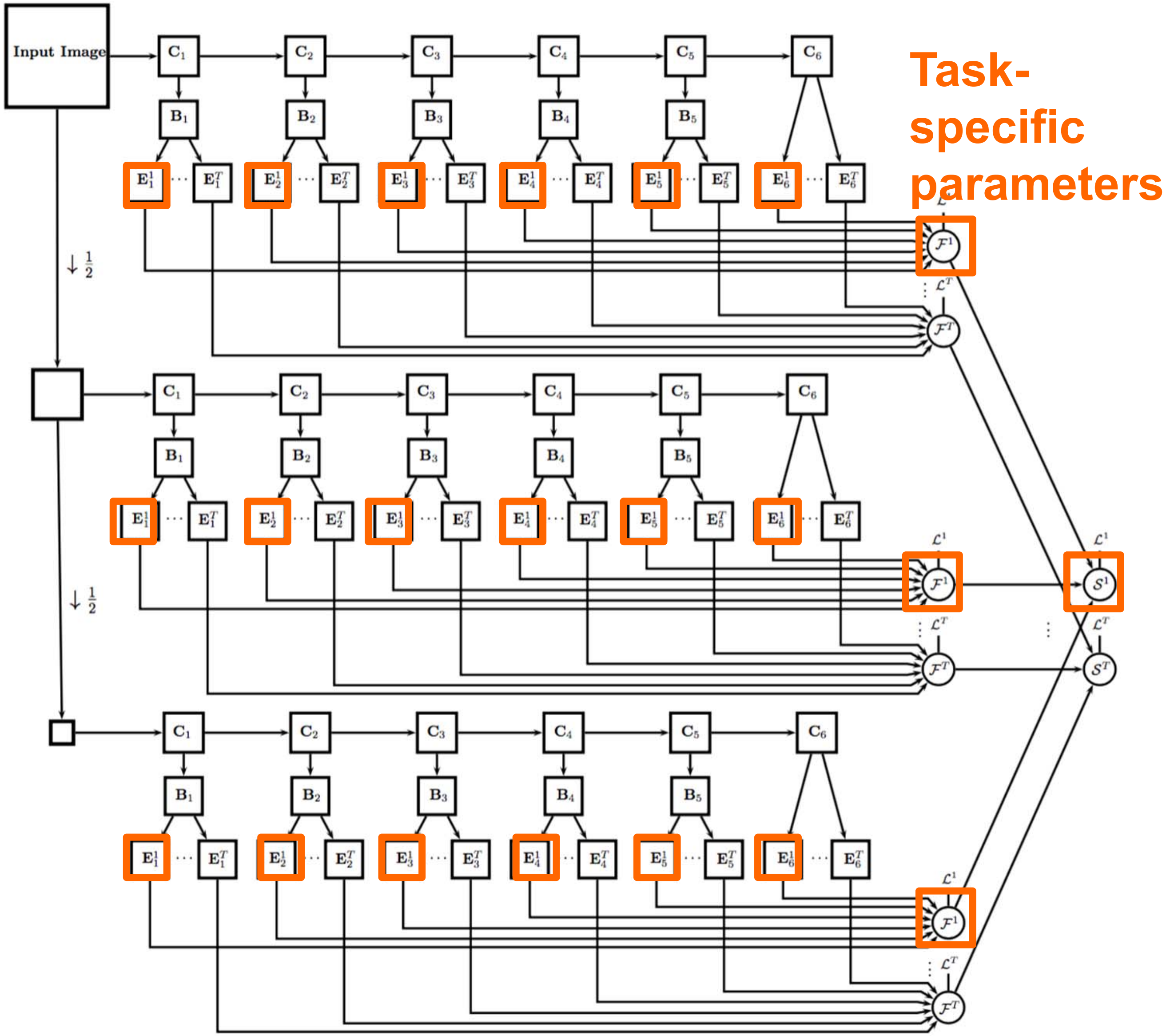
UberNet architecture



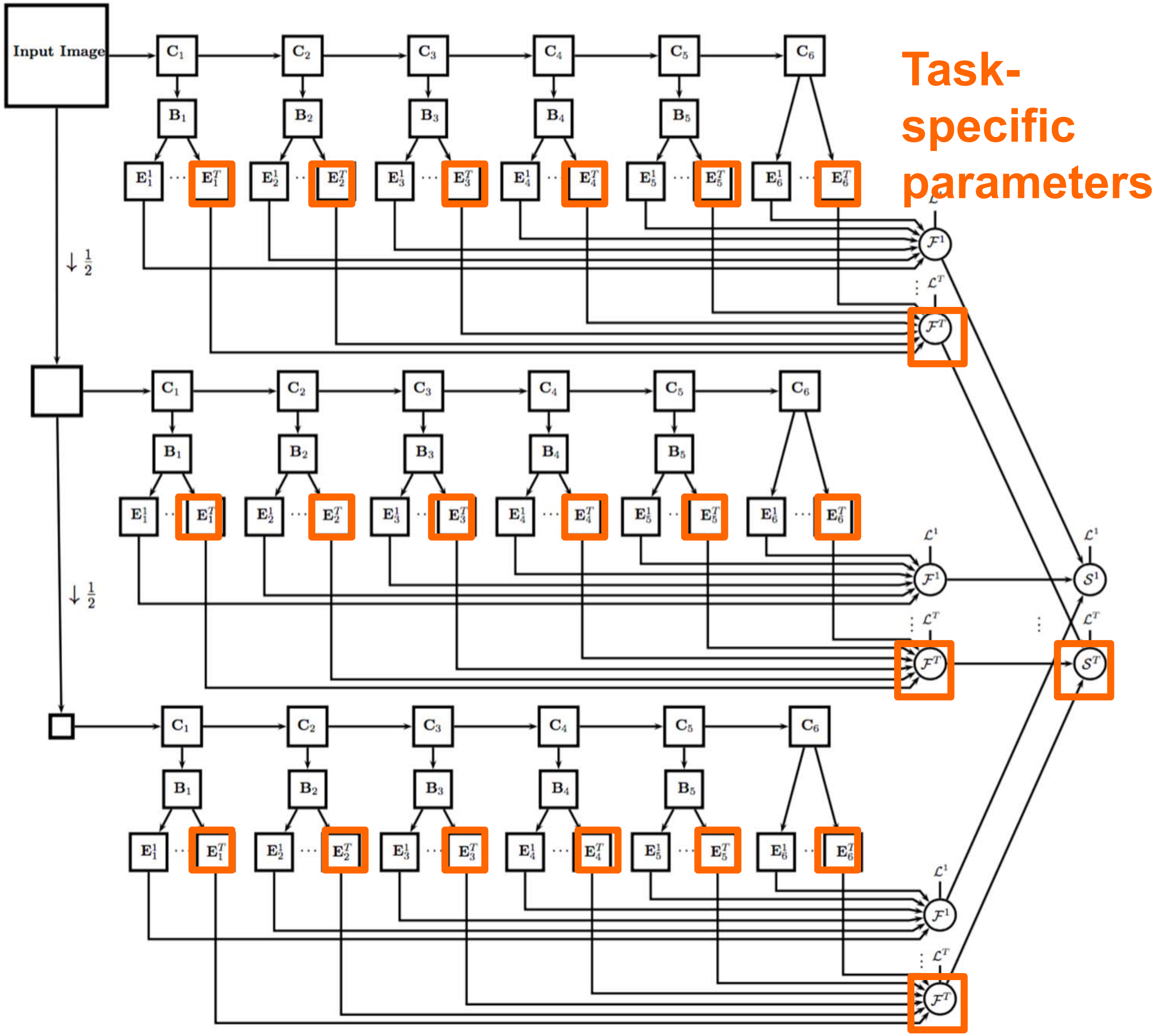


Shared CNN trunk (VGG16 network)

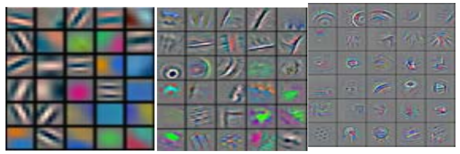
UberNet architecture

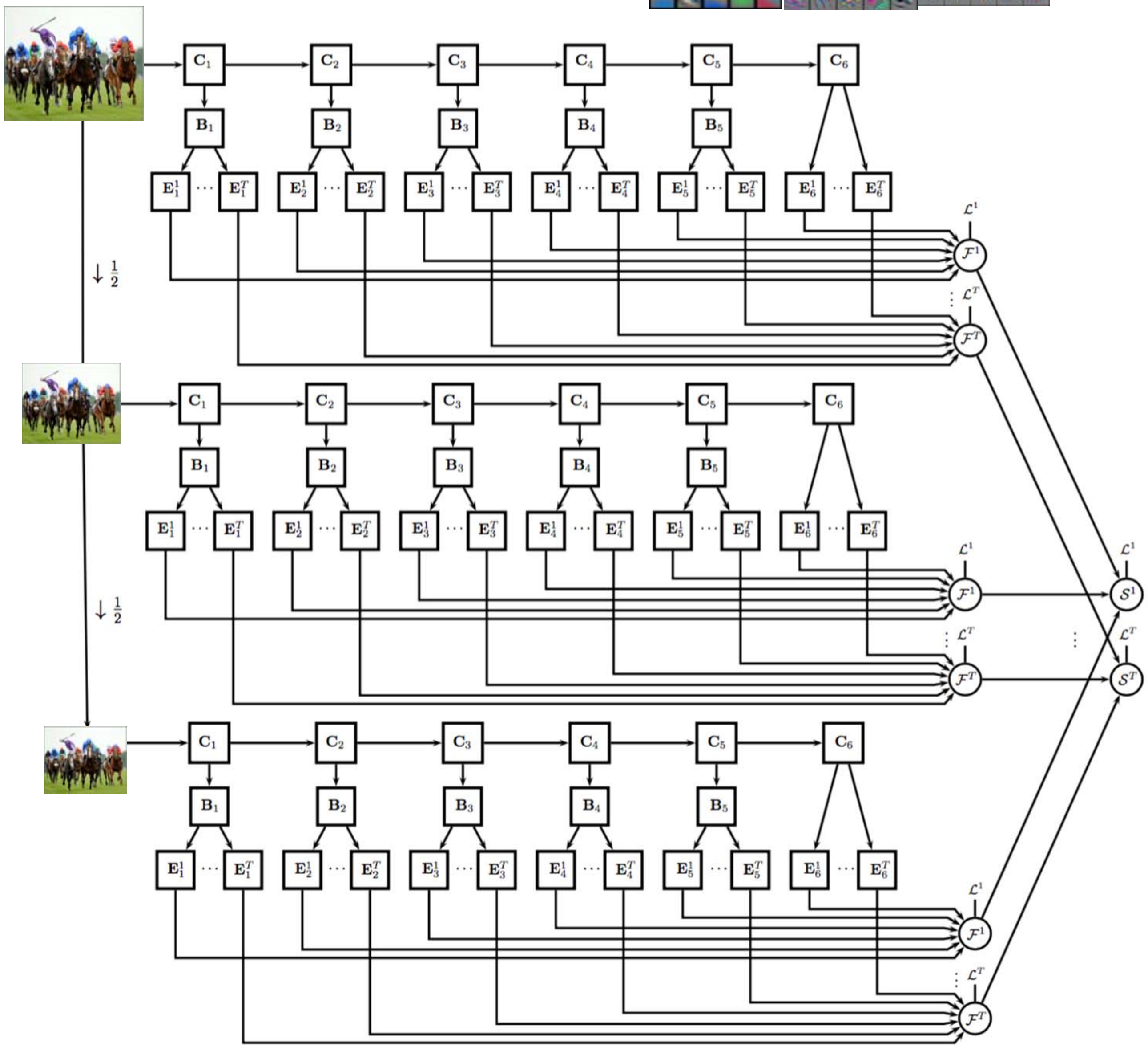


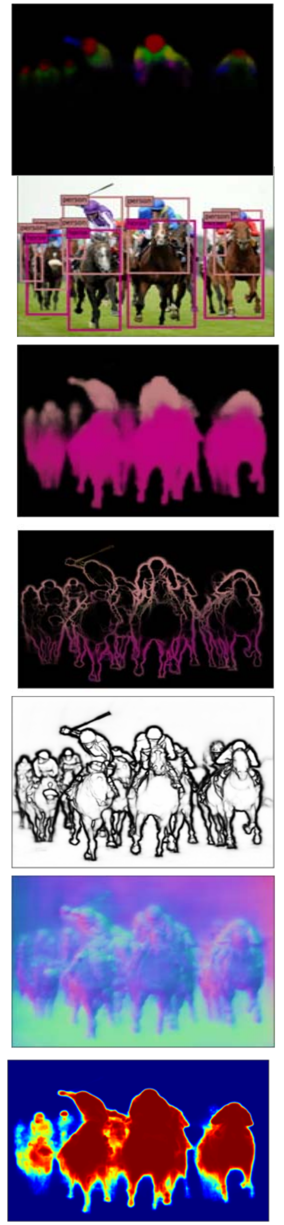
UberNet architecture



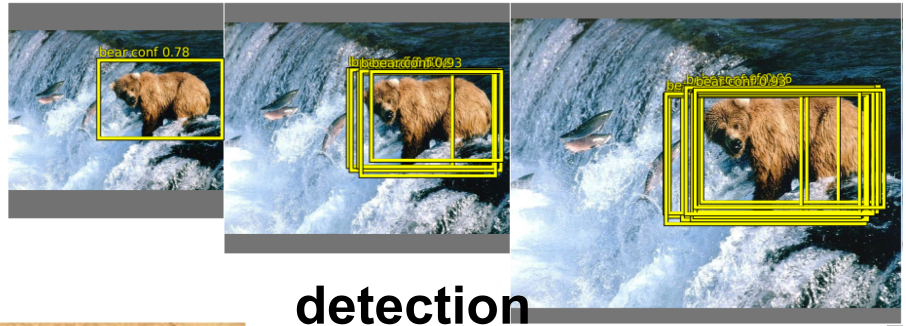
UberNet architecture



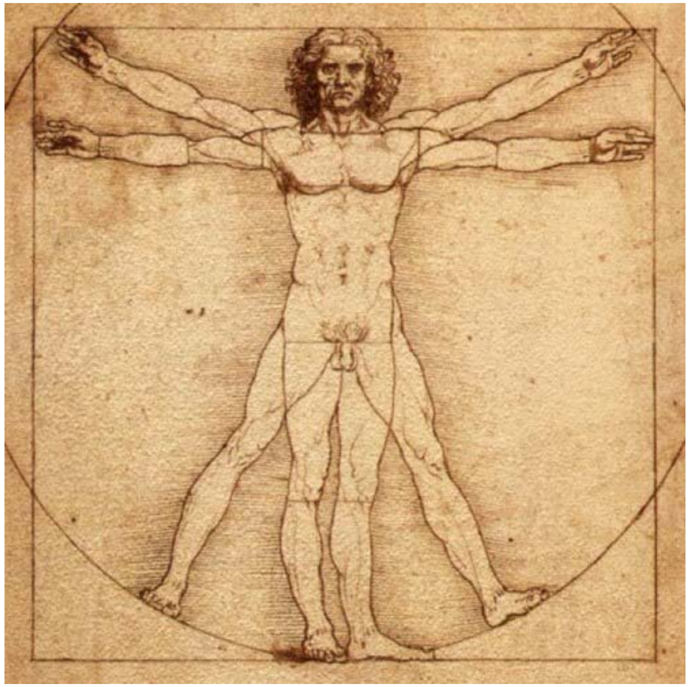




OverFeat, ICLR 2014



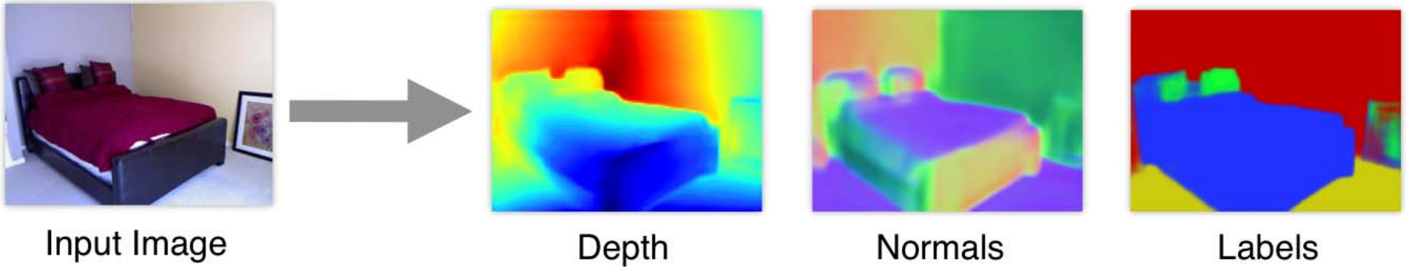
detection

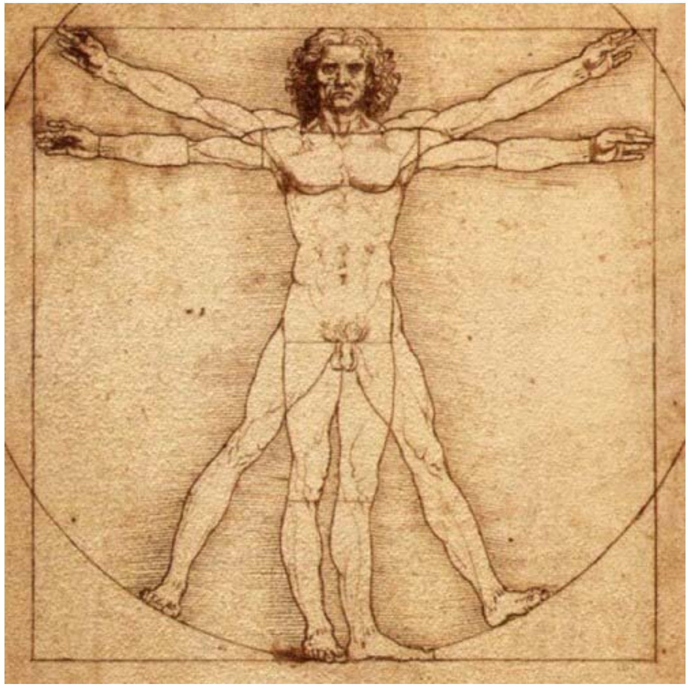


classification

P. Sermanet, D. Eigen, X. Zhiang, M. Mathieu, R. Fergus, and Y. LeCun, OverFeat: Integrated Recognition, Localization and Detection using CNNs, ICLR 2014

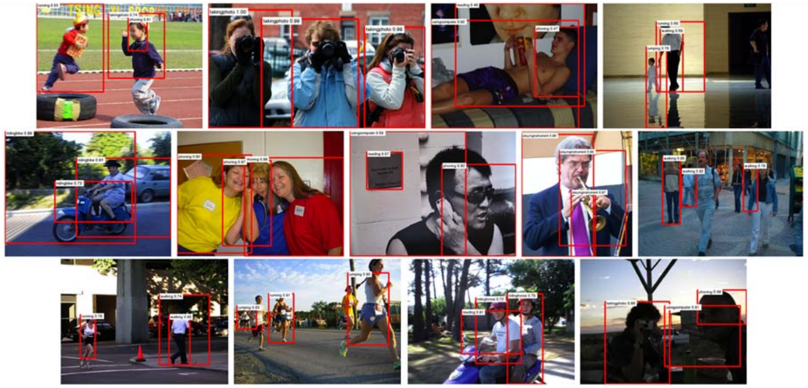
Eigen & Fergus, ICCV 2015





D. Eigen and R. Fergus, Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture, ICCV 2015

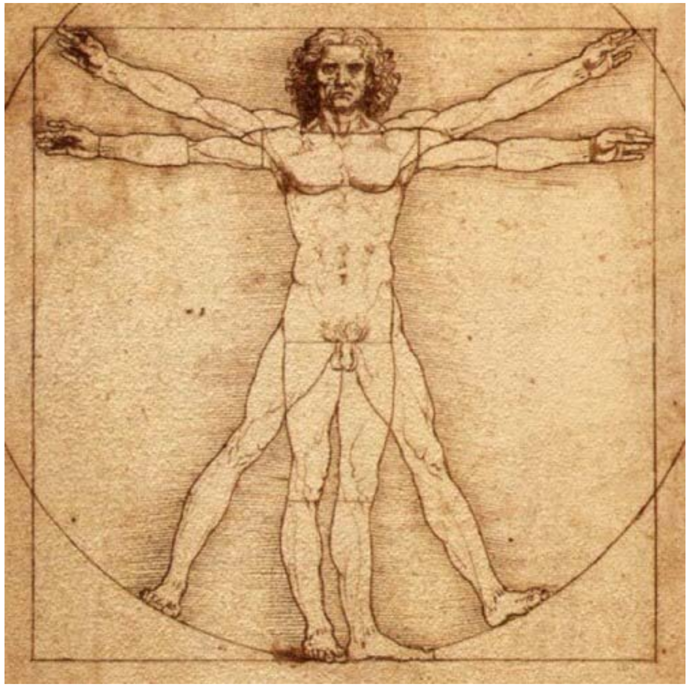
Gkioxari et al ICCV 2015



detection

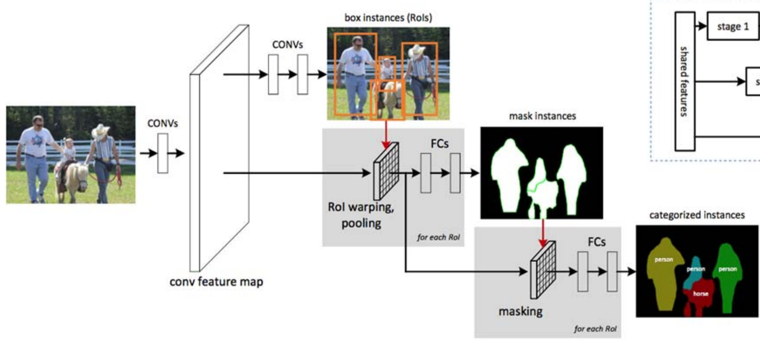
action recognition

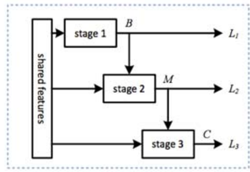
pose estimation



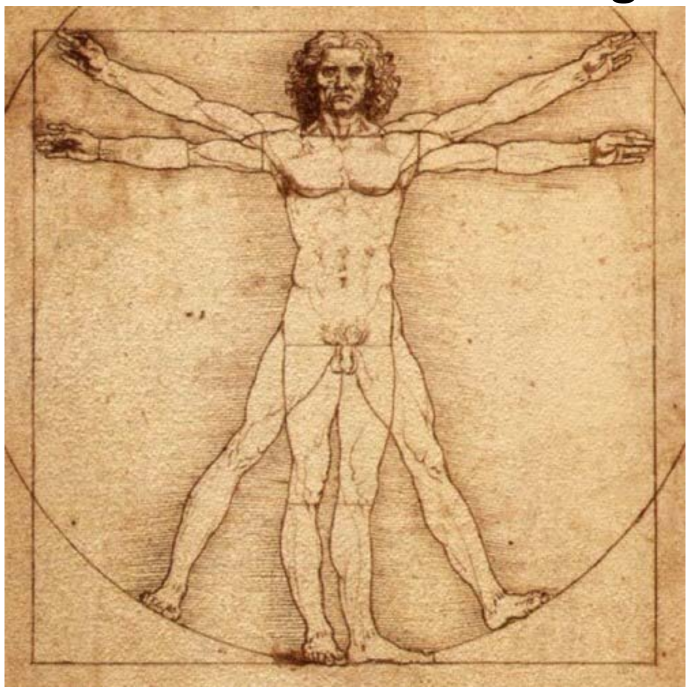
G. Gkioxari, B. Harihanan, R. Girshick and J. Malik, R-CNNs for Pose Estimation and Action Detection, ICCV 2015

Dai et al, CVPR 2016





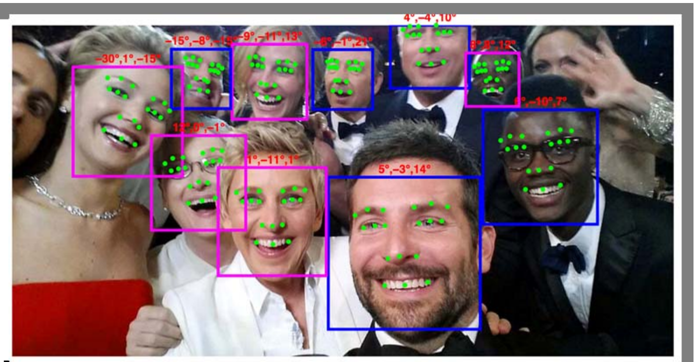
instance segmentation



detection

J. Dai, K. He, and J. Sun, Instance-aware Semantic Segmentation via Multi-task Network Cascades, CVPR 2016

Ranjan et al, 2016

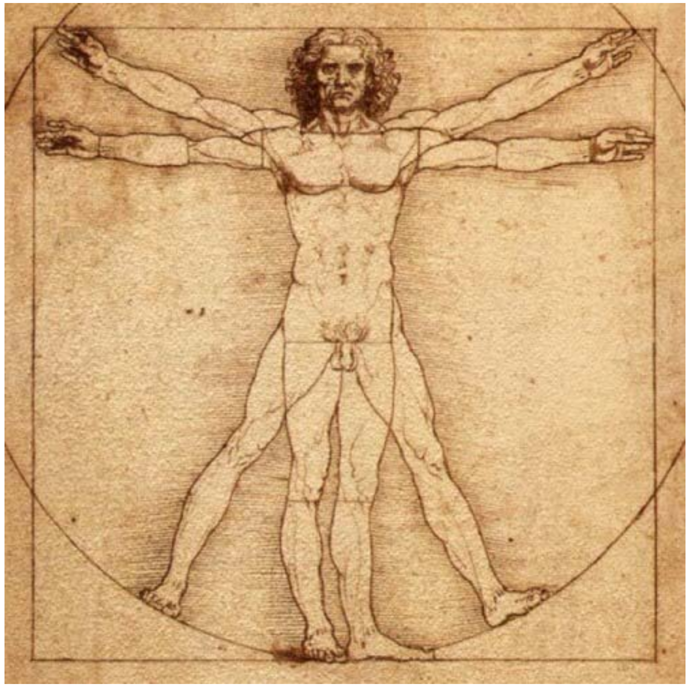


detection

landmarks

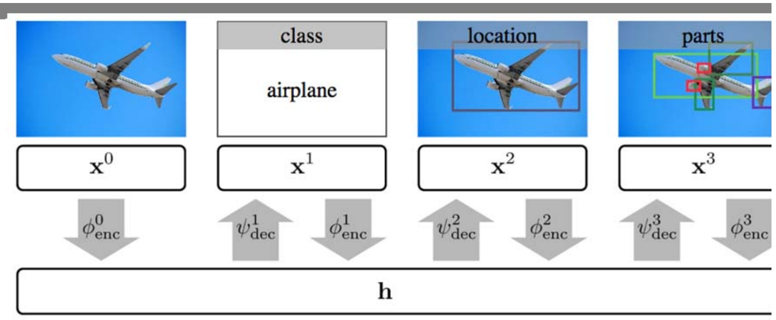
pose

gender

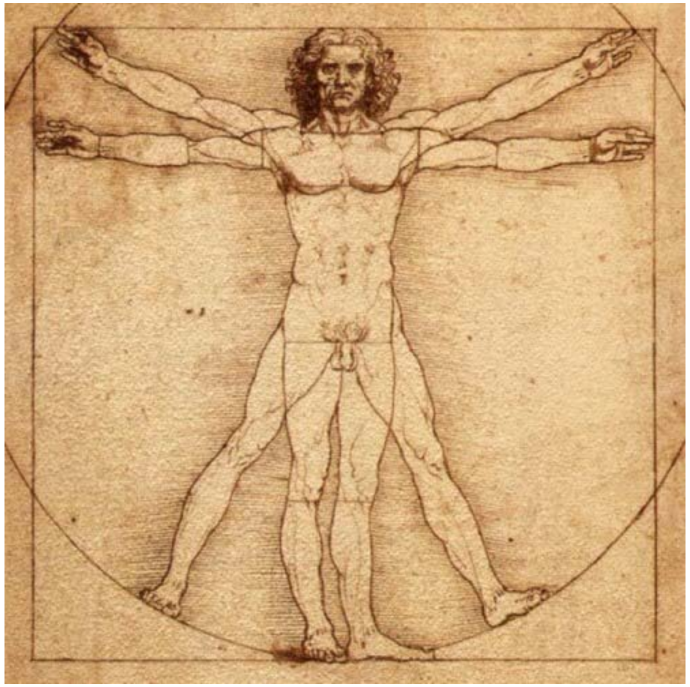


R. Ranjan, et al. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender

Bilen and Vedaldi, NIPS 2016



part



detection

classification

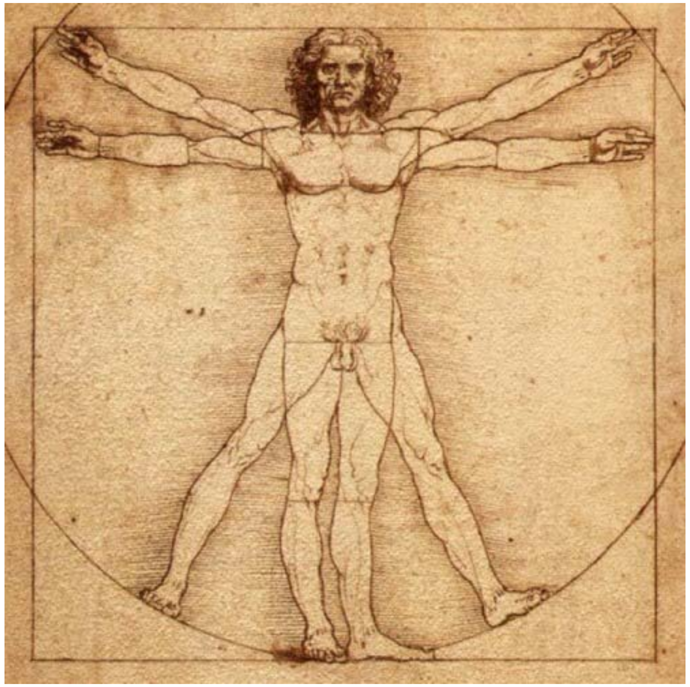
H. Bilen, and A. Vedaldi, Integrated perception with recurrent multi-task neural networks, NIPS 2016

He et al, ICCV 2017

instance segmentation

keypoints

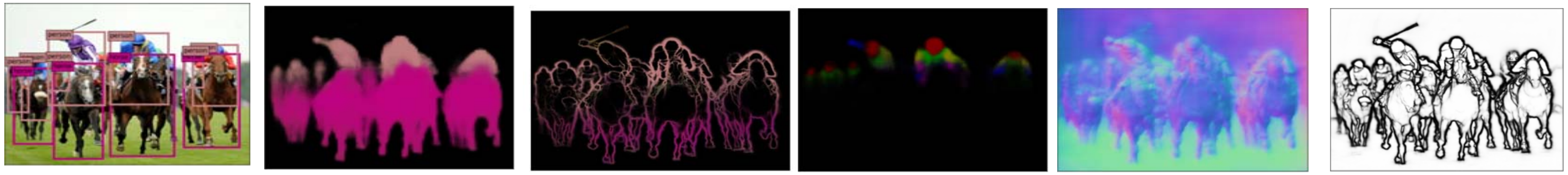
detection



K. He, P. Dollar, G. Gkioxari and R. Girshick, Mask-RCNN, ICCV 2017

UberNet: Training a Universal CNN for *Low- Mid- and High-Level* Vision using Diverse Datasets and Limited Memory

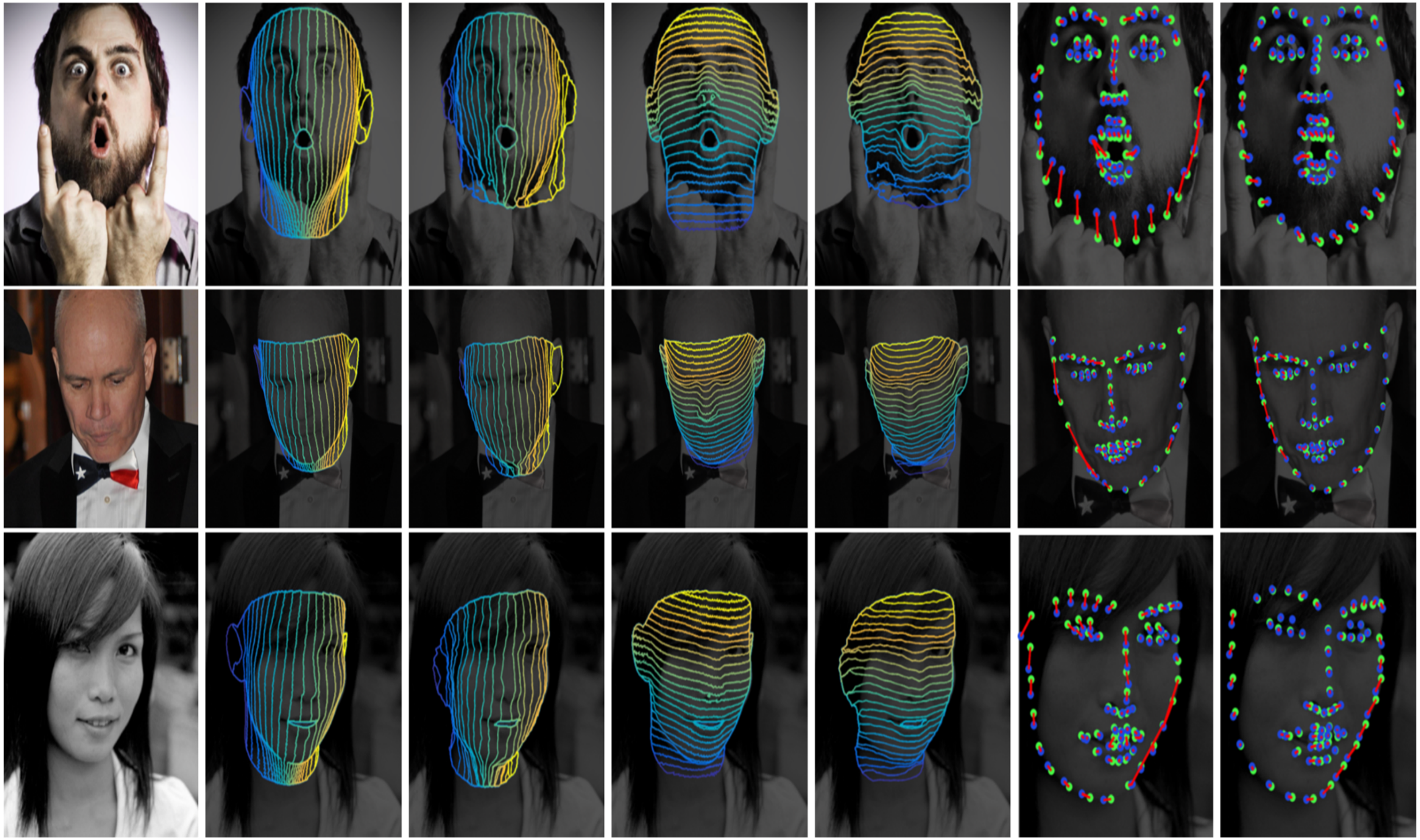
Low- Mid- and High-Level: **broad spectrum of tasks**

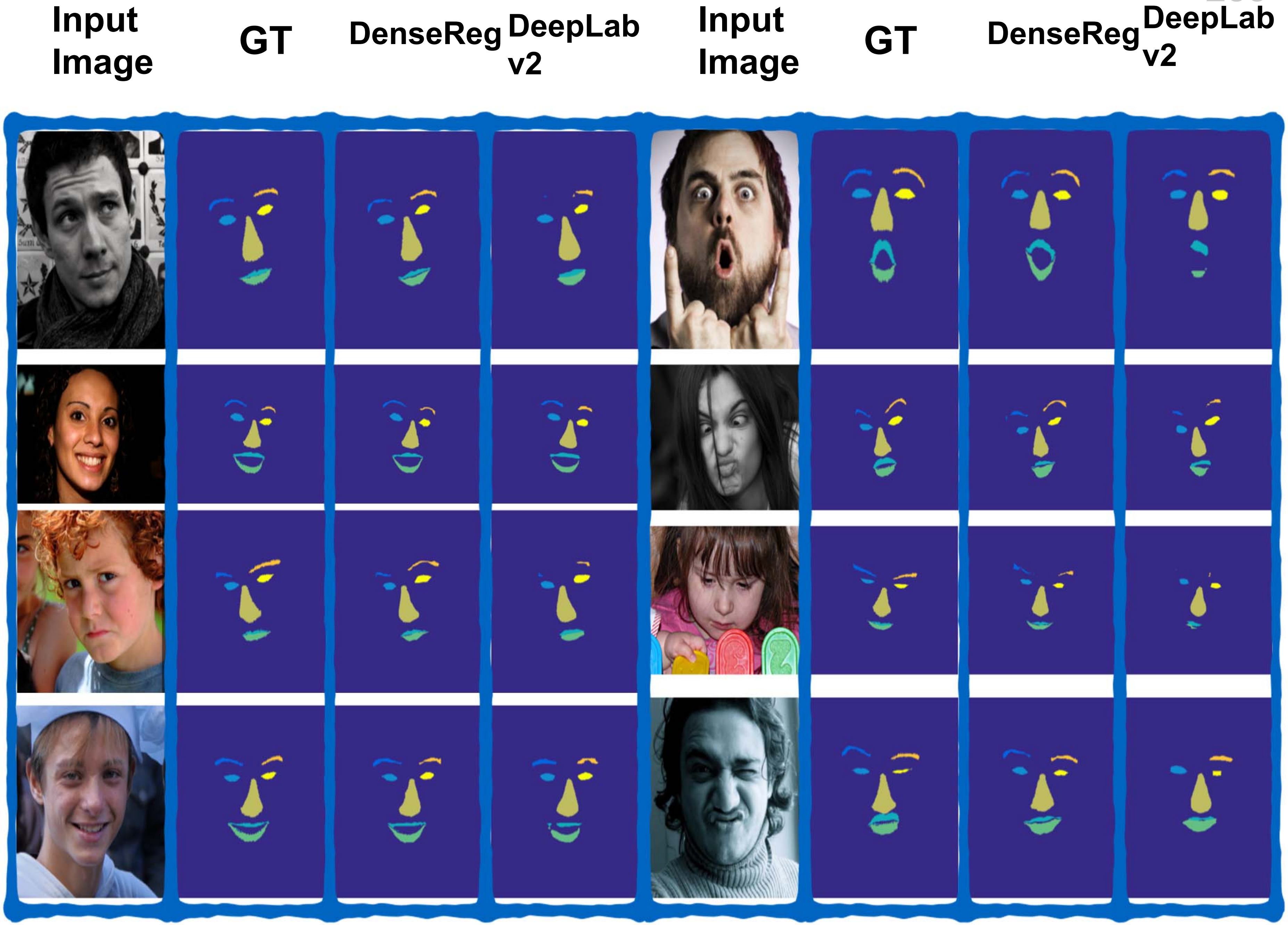


Diverse Datasets: **no single dataset**

Limited Memory: **too many tasks to fit in memory**

Input Image	GT(horz)	DenseReg (horz)	GT(vert)	DenseReg (vert)	DenseReg landmarks	DenseReg + MDM




Input Image
GT
DenseReg
DeepLab v2
Input Image
GT
DenseReg
DeepLab v2

http://alpguler.com/DenseReg.html

DenseReg:

Fully Convolutional Dense Shape Regression In-the-Wild

Rıza Alp Güler
George Trigeorgis
Epameinondas Antonakos
Patrick Snape
Stefanos Zafeiriou
Iasonas Kokkinos

informatics mathematics Inria
CentraleSupélec
Imperial College London
UCL

Why did this work now?

- Bigger computers and GPUs.
- Big data
- Tailored network architectures
- Better optimisation (Rectified Linear Units, ResNets)
- Regularisation (Dropout, batch normalisation)
- **Dedicated software libraries**

Deep Learning envorinments

	Performance	Usability	Flexibility	OS	Language	Community
Caffe	Very fast	(+) Easy for simple networks (plug and play) (-) not goof for RNN (-) cumbersome for big networks	(-) high-level: need to code in C++/ CUDA for new GPU layers	All	C++, but Python or Matlab interface available	(+) a lot of pre-trained models (-) community shrinking (-) slow development
TensorFlow	Slow	(+) easy Multi-GPUs (+) TensorBoard for visualisation (-) not goof for RNN	(+) relatively low-level: flexible enough to design your own layers in python.	Bad support for Windows	Python	(+) community growing (+) fast development (-) not many pretrained models
Theano	Slow	(+) nice for RNN (-) single GPU (-) harder to debug	(+) low-level by design, so easy to write your own layers (+) high-level wrappers (kearas, lasagne)	All	Python	(-) not many pretrained models
Torch	Fast	(+) nice for RNN (+) easy Multi-GPUs (-) Need to learn Lua	(+)once you know Lua, easy to write you own layers	Bad support for Windows	Lua	(+) Community growing (+) fast development (+) many pretrained models

See also: https://deeplearning4j.org/compare-dl4j-torch7-pylearn#keras

PYTORCH **http://pytorch.org/**

Why did this work now?

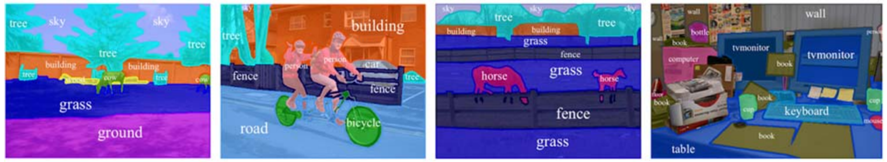
- Bigger computers and GPUs.
- **Big data**
- Tailored network architectures
- Better optimisation (Rectified Linear Units, ResNets)
- Regularisation (Dropout, batch normalisation)
- Dedicated software libraries
- **Pretraining/Finetuning/Transfer Learning**

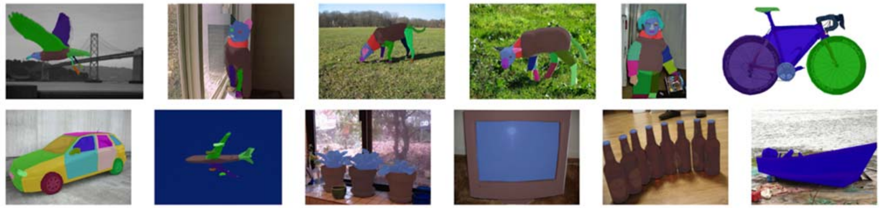
ImageNet

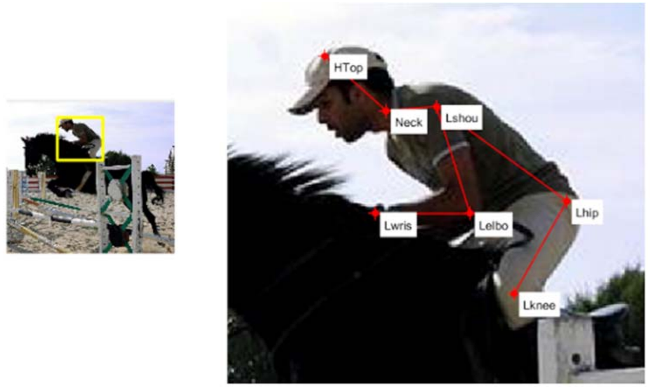
1000 Classes, 1M images

Classification

PASCAL



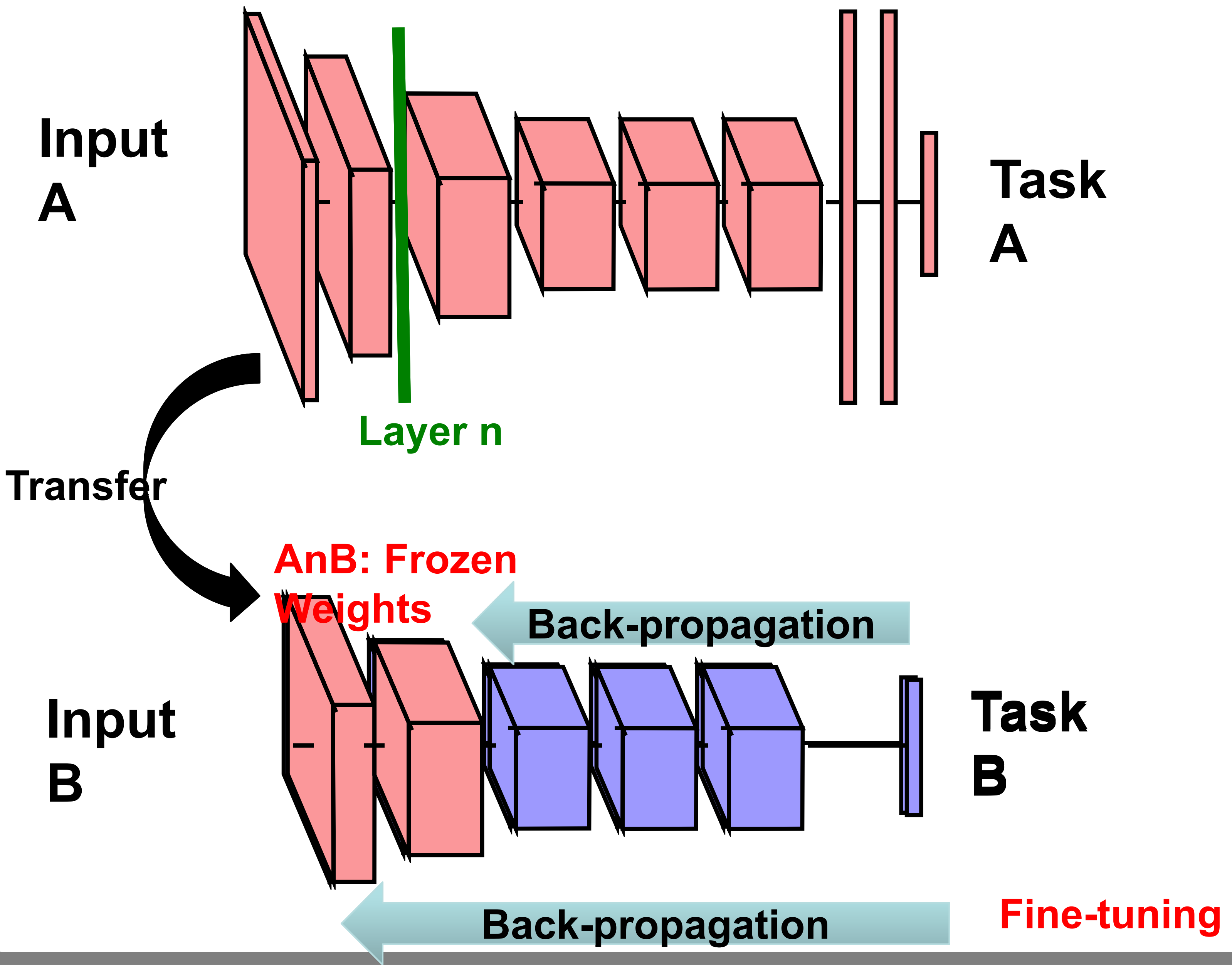




20 Classes, 10K images, N tasks

Object Detection,
Semantic
Segmentation,
Part Segmentation,
Instance

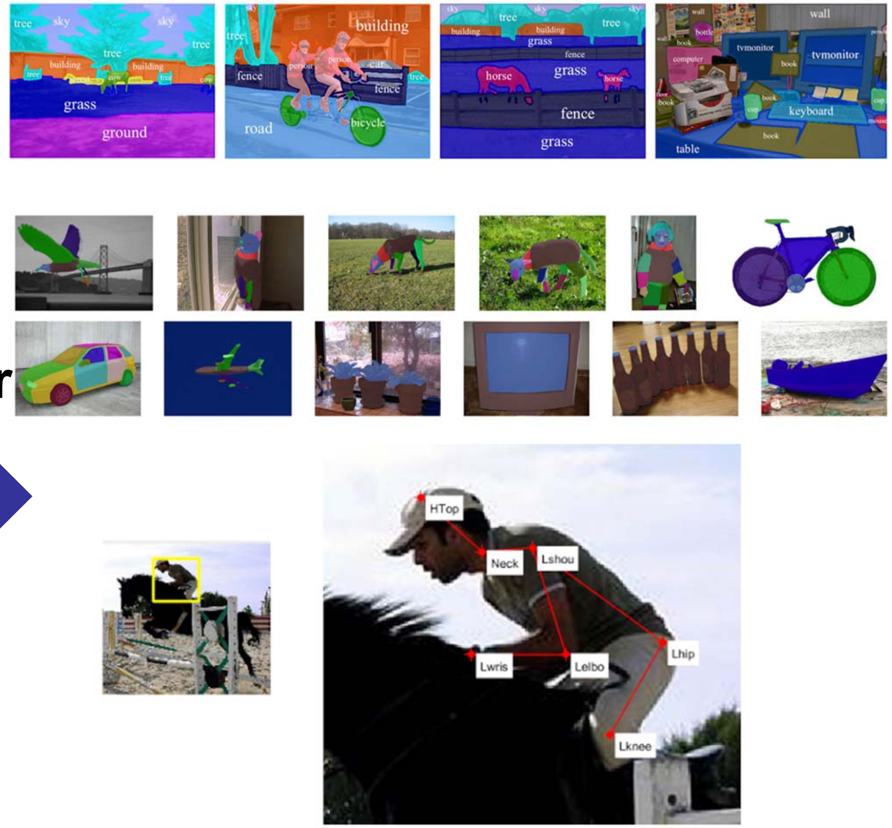
Transfer Learning Overview



ImageNet

PASCAL

Transfer



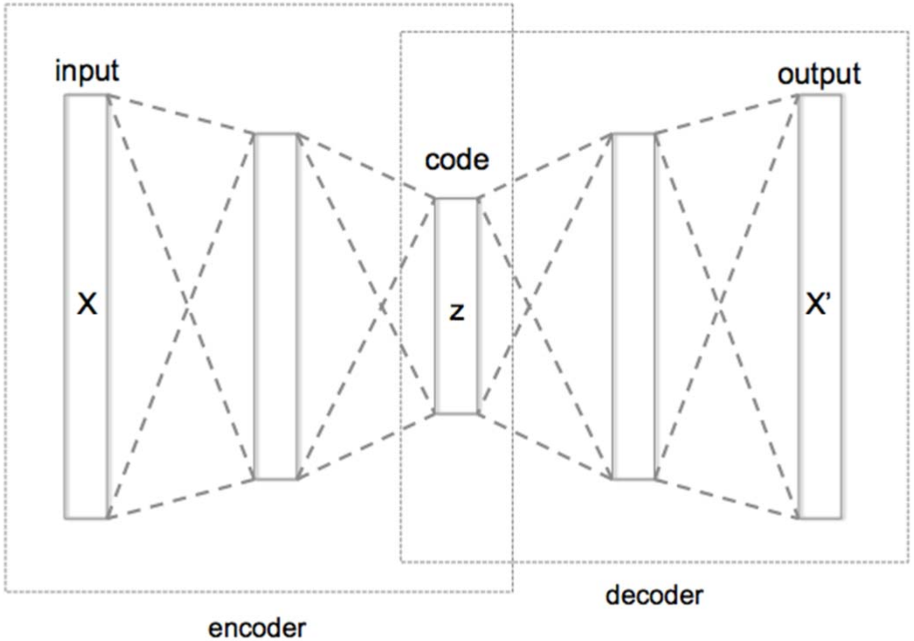
1000 Classes, 1M images

Classification

20 Classes, 10K images, N tasks

Object Detection,
Semantic Segmentation,
Part Segmentation,

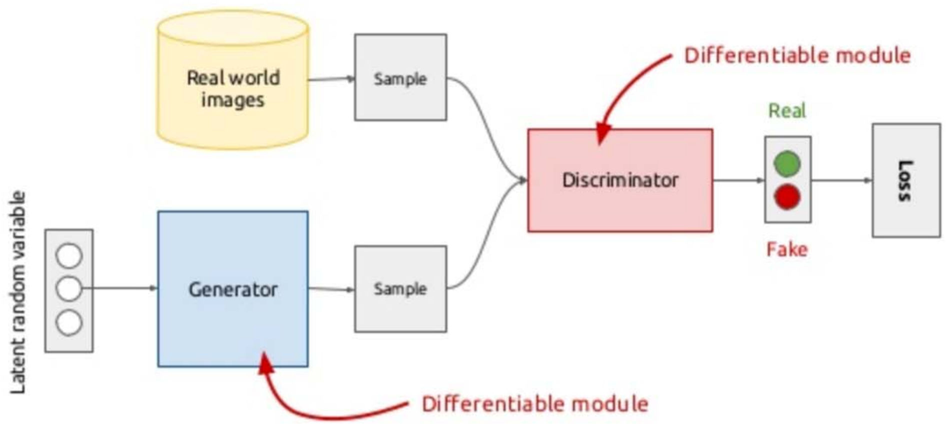
Unsupervised Learning: what if there is no ImageNet?

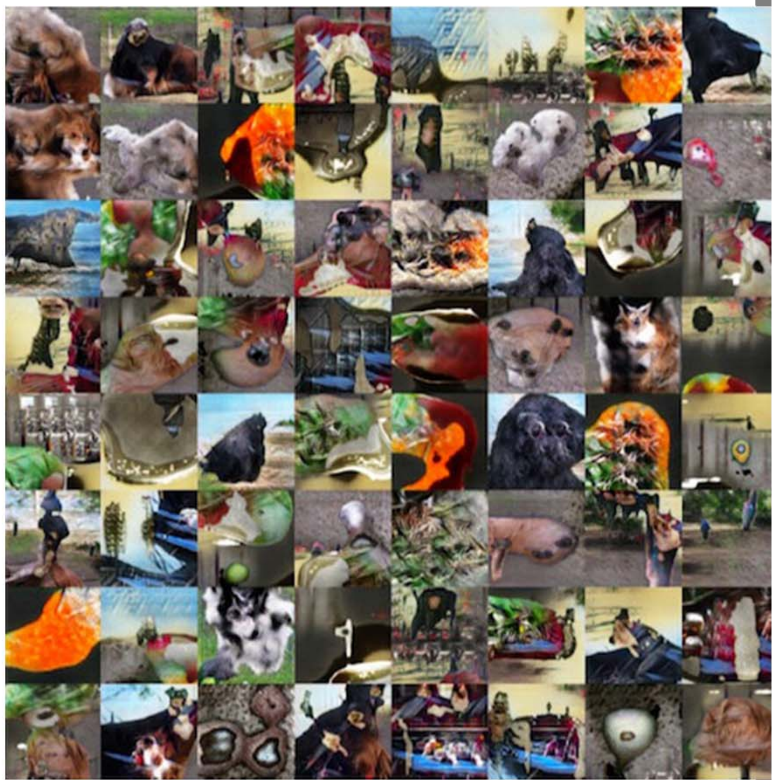


Autoencoder: Distill image to low-dimensional representation

Auto-Encoding Variational Bayes, Kingma, D.P. and Welling, M.,2013

Unsupervised Learning: what if there is no ImageNet





GANs: Learn to create realistic data

Samples of natural images

Goodfellow, et. al., "Generative Adversarial Networks". 2014