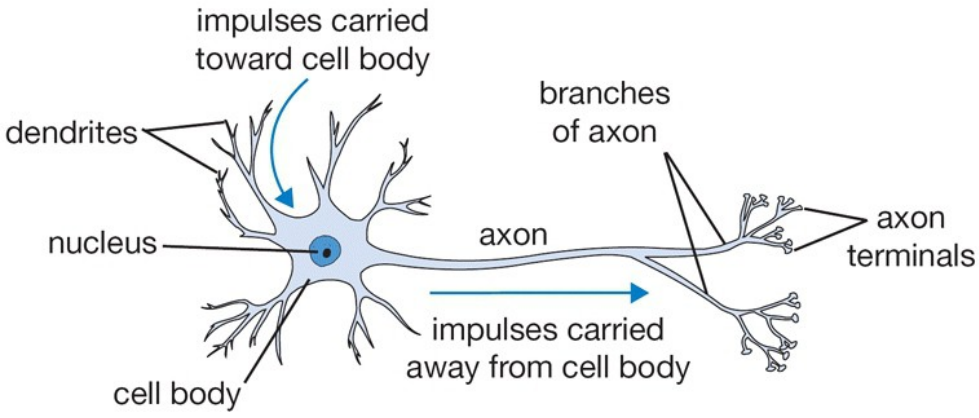
Introduction to Neural Networks

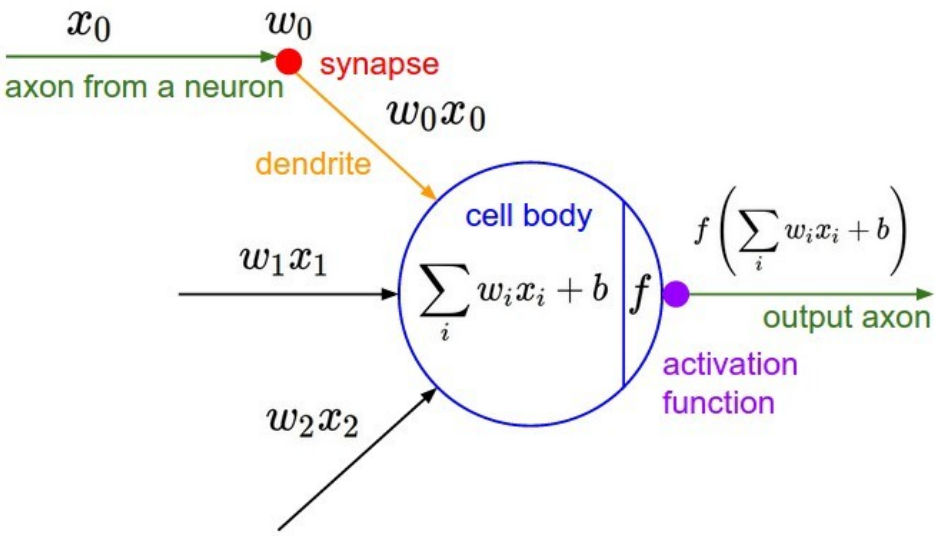
Jakob Verbeek

2017-2018

Biological motivation

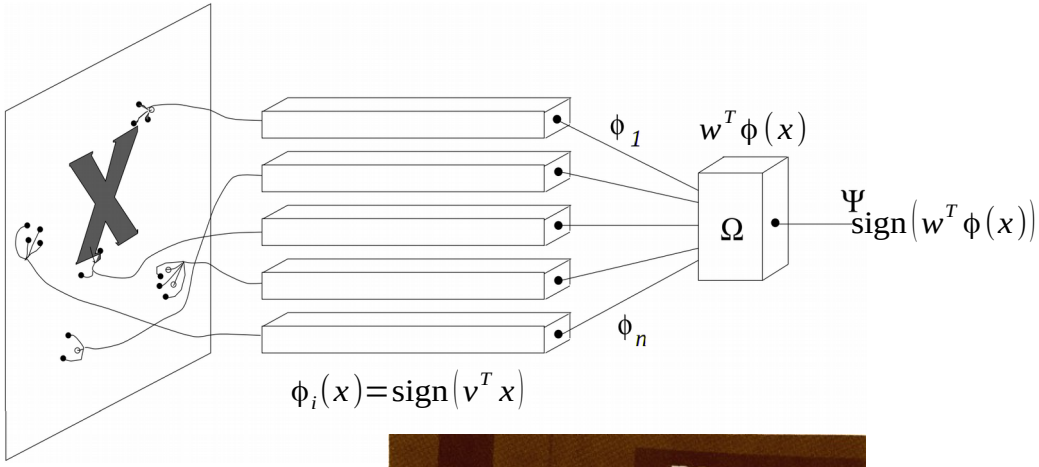
- Neuron is basic computational unit of the brain
  - about 10^11 neurons in human brain
- Simplified neuron model as linear threshold unit (McCulloch & Pitts, 1943)
  - Firing rate of electrical spikes modeled as continuous output quantity
  - Connection strength modeled by multiplicative weight
  - Cell activation given by sum of inputs
  - Output is non linear function of activation
- Basic component in neural circuits for complex tasks



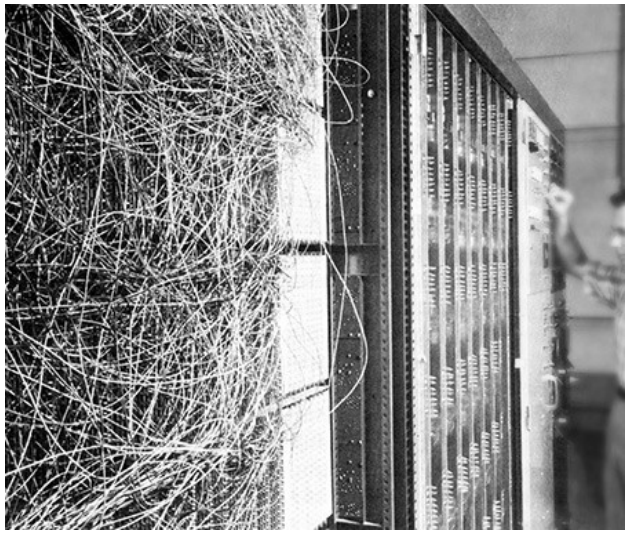


1957: Rosenblatt's Perceptron

- Binary classification based on sign of generalized linear function
  - Weight vector w learned using special purpose machines
  - Fixed associative units in first layer, sign activation prevents learning



20x20 pixel sensor



Random wiring of associative units

Rosenblatt's Perceptron

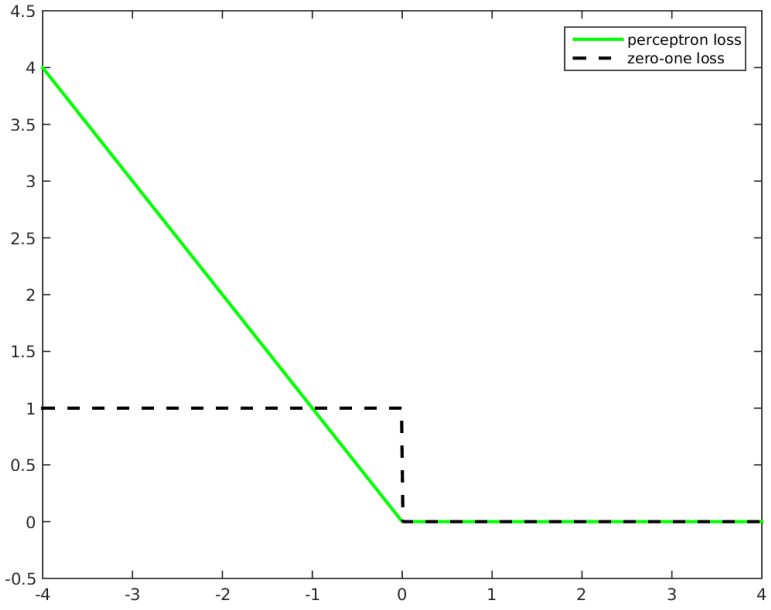
- Objective function linear in score over misclassified patterns $t_i \in \{-1, +1\}$

$$E(w) = -\sum_{t_i \neq sign(f(x_i))} t_i f(x_i) = \sum_i max(0, -t_i f(x_i))$$

- Perceptron learning via stochastic gradient descent

$$w^{n+1} = w^n + \eta \times t_i \phi(x_i) \times [t_i f(x_i) < 0]$$

  - Eta is the learning rate



Potentiometers as weights adjusted by motors during learning

Perceptron convergence theorem

- If a correct solution w* exists, then the perceptron learning rule will converge to a correct solution in a finite number of iterations for any initial weight vector
- Assume input lives in L2 ball of radius M, and without loss of generality that
  - w* has unit L2 norm
  - Some margin exists for the right solution $y\langle w^*, x\rangle > \delta$
- After a weight update $w' = w + yx$ we have $\langle w^*, w'\rangle = \langle w^*, w\rangle + y\langle w^*, x\rangle > \langle w^*, w\rangle + \delta$
- Moreover, since $y\langle w, x\rangle < 0$ for misclassified sample, we have

$$\begin{aligned}\langle w', w'\rangle &= \langle w, w\rangle + 2\, y\langle w, x\rangle + \langle x, x\rangle \\ &< \langle w, w\rangle + \langle x, x\rangle \\ &< \langle w, w\rangle + M\end{aligned}$$

- Thus after t updates we have

$$\langle w^*, w'\rangle > \langle w^*, w\rangle + t\,\delta$$
$$\langle w', w'\rangle < \langle w, w\rangle + tM$$

- Therefore $a(t) = \frac{\langle w^*, w(t)\rangle}{\sqrt{\langle w(t), w(t)\rangle}} > \frac{\langle w^*, w\rangle + t\,\delta}{\sqrt{\langle w, w\rangle + tM}}$, in limit of large t: $a(t) > \frac{\delta}{\sqrt{M}}\sqrt{t}$
- Since a(t) is upper bounded by construction by 1, the nr. of updates t must be limited.
- For start at w=0, we have that $t \le \frac{M}{\delta^2}$

Limitations of the Perceptron

- Perceptron convergence theorem (Rosenblatt, 1962) states that
  - If training data is linearly separable, then learning algorithm finds a solution in a finite number of iterations
  - Faster convergence for larger margin

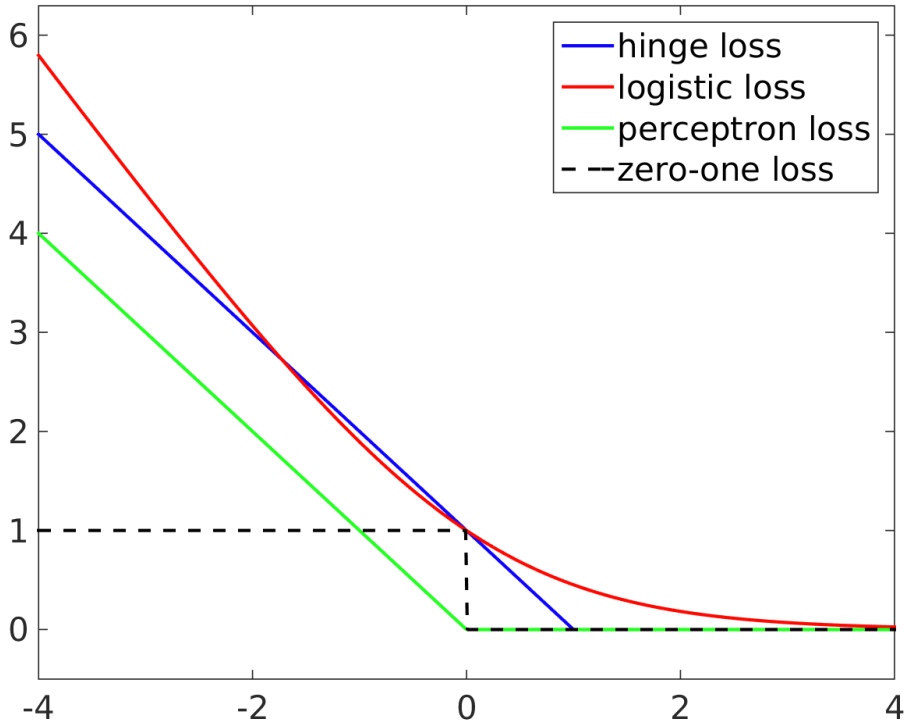
- If training data is linearly separable then the found solution will depend on the initialization and ordering of data in the updates
- If training data is not linearly separable, then the perceptron learning algorithm will not converge

- No direct multi-class extension
- No probabilistic output or confidence on classification

Relation to SVM and logistic regression

- Perceptron loss similar to hinge loss without the notion of margin
  - Not a bound on the zero-one loss
  - Loss is zero for any separator, not only for large margin separators
- All are either based on linear score function, or generalized linear function by relying on pre-defined non-linear data transformation or kernel

$$f(x) = w^T \phi(x)$$



Kernels to go beyond linear classification

- Representer theorem states that in all these cases optimal weight vector is linear combination of training data

$$w = \sum_i \alpha_i \phi(x_i)$$

$$f(x) = w^T \phi(x) = \sum_i \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

- Kernel trick allows us to compute dot-products between (high-dimensional) embedding of the data

$$k(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$$

- Classification function is linear in data representation given by kernel evaluations over the training data

$$f(x) = \sum_i \alpha_i k(x, x_i) = \alpha^T k(x, .)$$

Limitation of kernels

- Classification based on weighted “similarity” to training samples
  - Design of kernel based on domain knowledge and experimentation

$$f(x)=\sum_i \alpha_i k(x, x_i)=\alpha^T k(x,.)$$

  - Some kernels are data adaptive, for example the Fisher kernel
  - Still kernel is designed before and separately from classifier training

- Number of free variables grows linearly in the size of the training data
  - Unless a finite dimensional explicit embedding is available $\phi(x)$
  - Can use kernel PCA to obtain such a explicit embedding

- Alternatively: fix the number of “basis functions” in advance
  - Choose a family of non-linear basis functions
  - Learn the parameters of basis functions and linear function

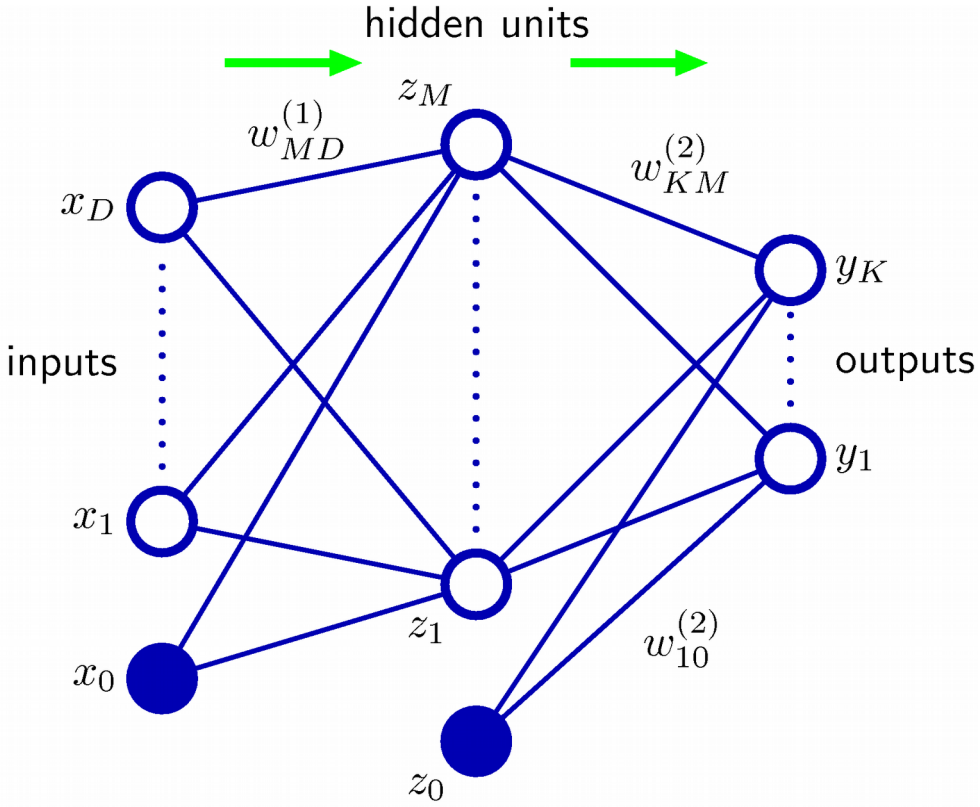
$$f(x)=\sum_i \alpha_i \phi_i(x;\theta_i)$$

Multi-Layer Perceptron (MLP)

- Instead of using a generalized linear function, learn the features as well
- Each unit in MLP computes
  - Linear function of features in previous layer
  - Followed by scalar non-linearity
- Do **not** use the "step" non-linear activation function of original perceptron

$$z_j = h\left(\sum_i x_i w_{ij}^{(1)}\right)$$
$$z = h\left(W^{(1)} x\right)$$

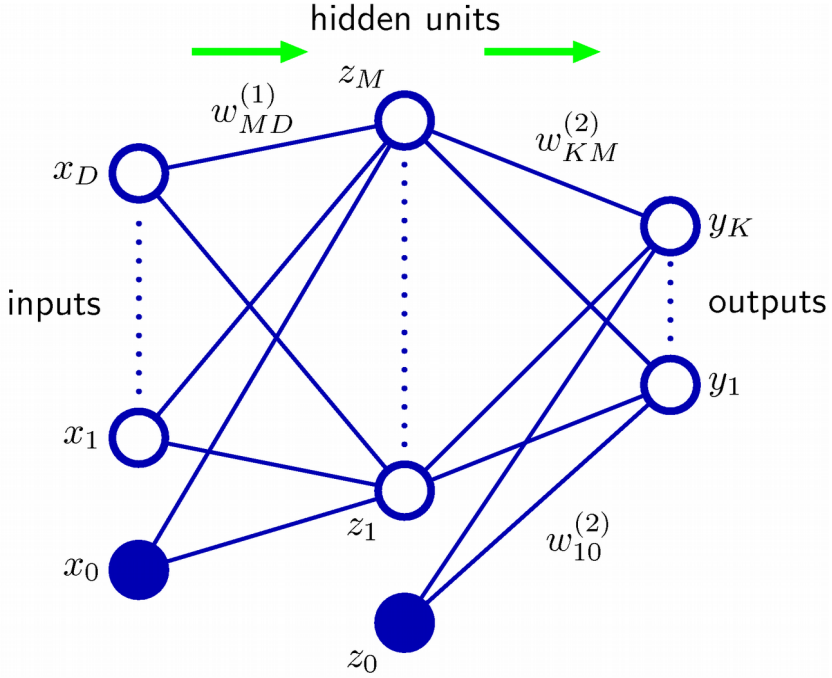
$$y_k = \sigma\left(\sum_j z_j w_{jk}^{(2)}\right)$$
$$y = \sigma\left(W^{(2)} z\right)$$



Multi-Layer Perceptron (MLP)

- Linear activation function leads to composition of linear functions
  - Remains a linear model, layers just induce a certain factorization

- Two-layer MLP can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units
  - Holds for many non-linearities, but not for polynomials



Classification over binary inputs

- Consider simple case with D binary input units
  - Inputs and activations are all +1 or -1
  - Total number of possible inputs is 2^D
  - Classification problem into two classes
- Create hidden unit for each positive sample x_m

$$z_m = \text{sign}(w_m^T x - D) \qquad \text{sign}(y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_m = x_m$$

  - Activation is +1 only if input equals x_m
- Let output implement an “or” over hidden units

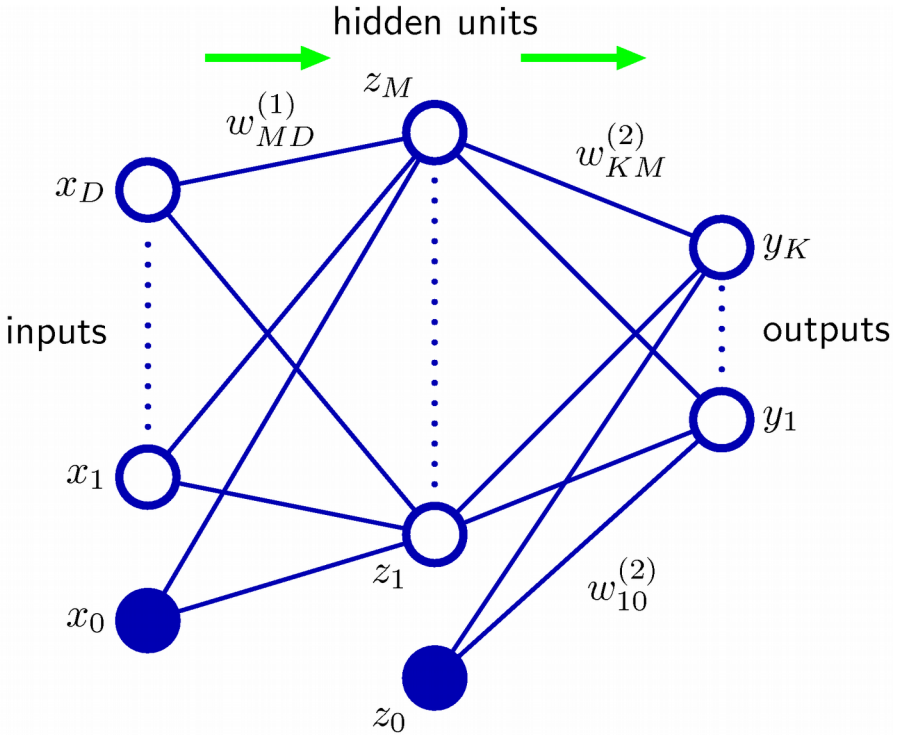
$$y = \text{sign}\left(\sum_{m=1}^{M} z_m - 1\right)$$

- MLP can separate any labeling over domain
  - But may need exponential number of hidden units to do so

Feed-forward neural networks

- MLP Architecture can be generalized
  - More than two layers of computation
  - Skip-connections from previous layers
- Feed-forward nets are restricted to directed acyclic graphs of connections
  - Ensures that output can be computed from the input in a single feed-forward pass from the input to the output

- Important issues in practice
  - Designing network architecture
    - Nr nodes, layers, non-linearities, etc
  - Learning the network parameters
    - Non-convex optimization
  - Sufficient training data
    - Data augmentation, synthesis



An example: multi-class classification

- One output score for each target class

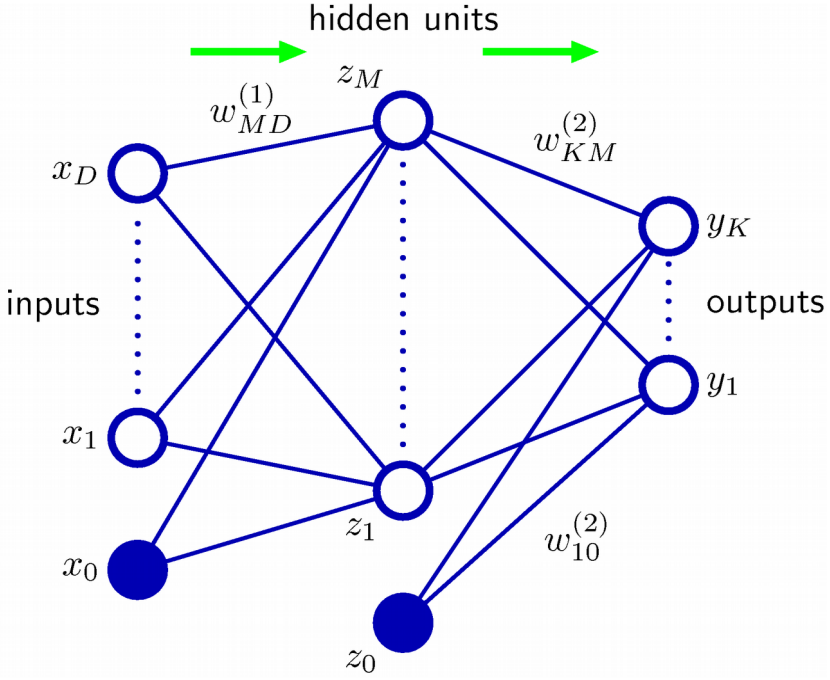
$$p(l=c|x)=\frac{\exp y_c}{\sum_k \exp y_k}$$

- Multi-class logistic regression loss (cross-entropy loss)
  - Define probability of classes by softmax over scores
  - Maximize log-probability of correct class

$$L=-\sum_n \ln p(l_n|x_n)$$

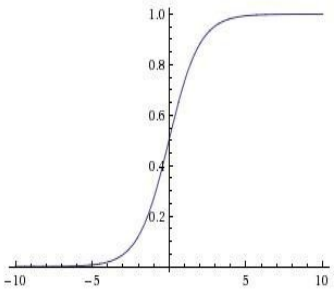
- As in logistic regression, but we are now learning the data representation concurrently with the linear classifier

- Representation learning in discriminative and coherent manner
- Fisher kernel also data adaptive but not discriminative and task dependent
- More generally, we can choose a loss function for the problem of interest and optimize all network parameters w.r.t. this objective (regression, metric learning, ...)

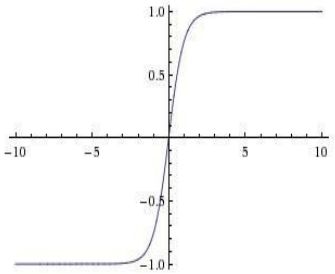


Activation functions

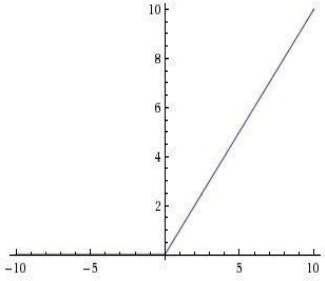
Sigmoid $1/(1+e^{-x})$



tanh

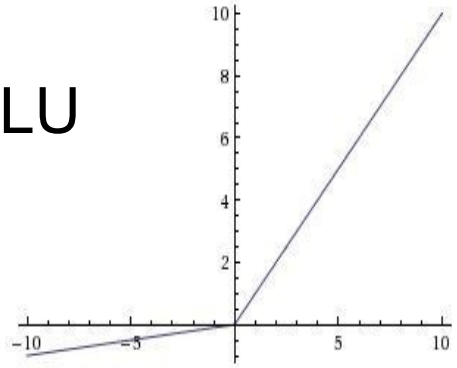


ReLU $\max(0, x)$

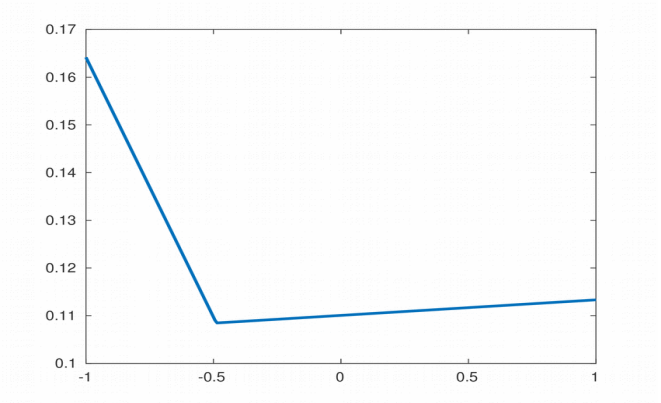


Leaky ReLU

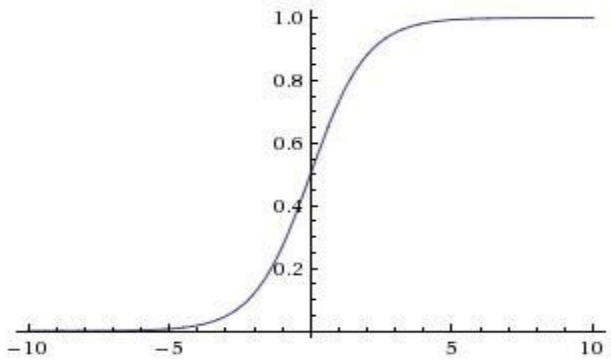
$\max(\alpha x, x)$



Maxout $\max(w_1^T x, w_2^T x)$

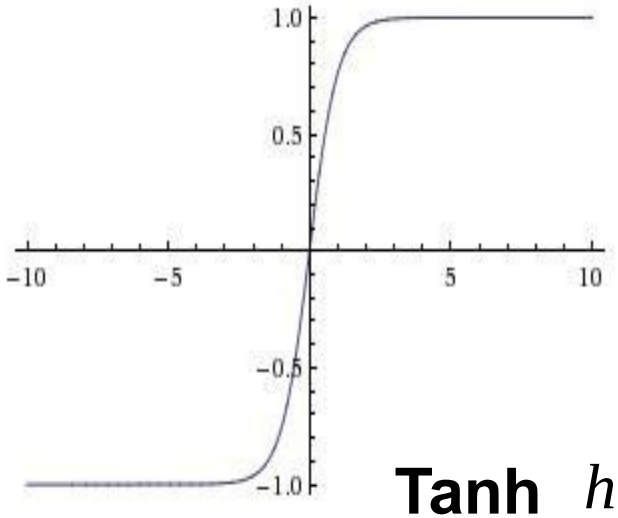


Activation Functions



- Squashes reals to range [0,1]
- Tanh outputs centered at zero: [-1, 1]
- Smooth step function
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Sigmoid $\sigma(x) = 1/(1+e^{-x})$

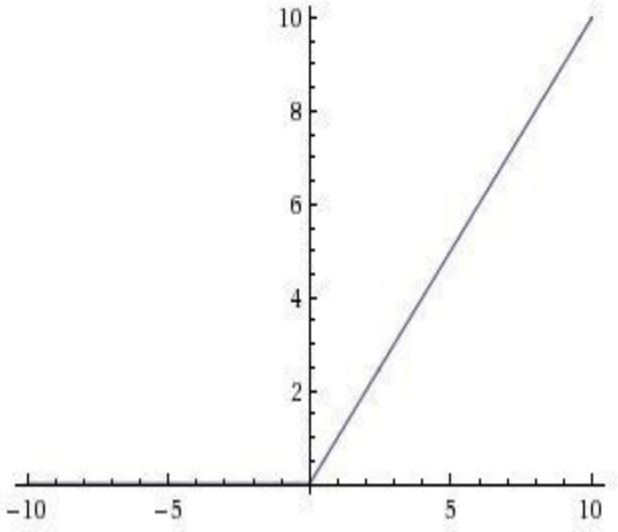


1. Saturated neurons "kill" the gradients, need activations to be exactly in right regime to obtain non-constant output
2. exp() is a bit compute expensive

Tanh $h(x) = 2\sigma(x) - 1$

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Computes **f(x) = max(0,x)**

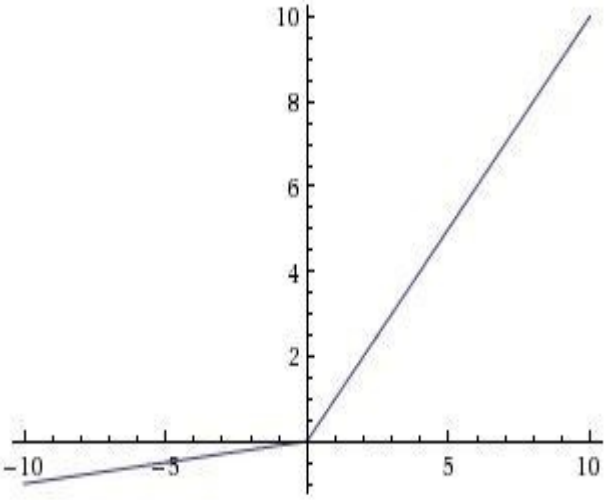
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Most commonly used today

ReLU
(Rectified Linear Unit)

[Nair & Hinton, 2010]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Leaky ReLU

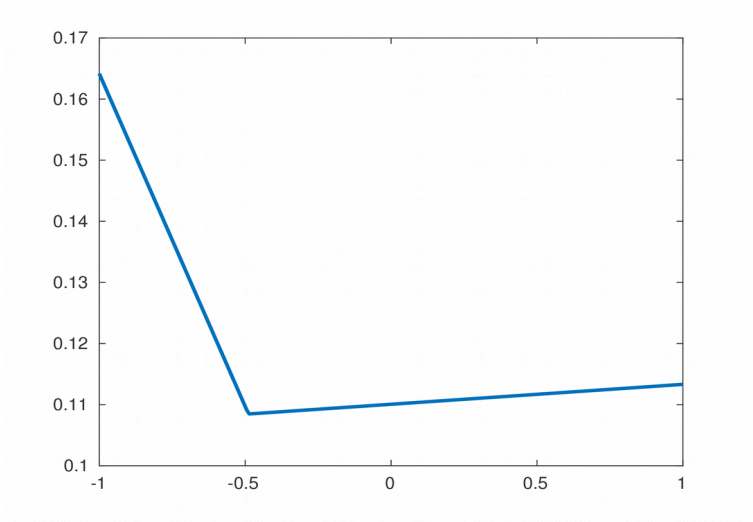
- Does not saturate: will not “die”
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)

$$f(x) = \max(0.01x, x)$$

[Mass et al., 2013] [He et al., 2015]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Activation Functions



Maxout

$$\max\left(w_1^T x, w_2^T x\right)$$

- Does not saturate: will not “die”
- Computationally efficient

- Maxout networks can implement ReLU networks and vice-versa
- More parameters per node

[Goodfellow et al., 2013]

Training feed-forward neural network

- Non-convex optimization problem in general
  - Typically number of weights is very large (millions in vision applications)
  - Seems that many different local minima exist with similar quality

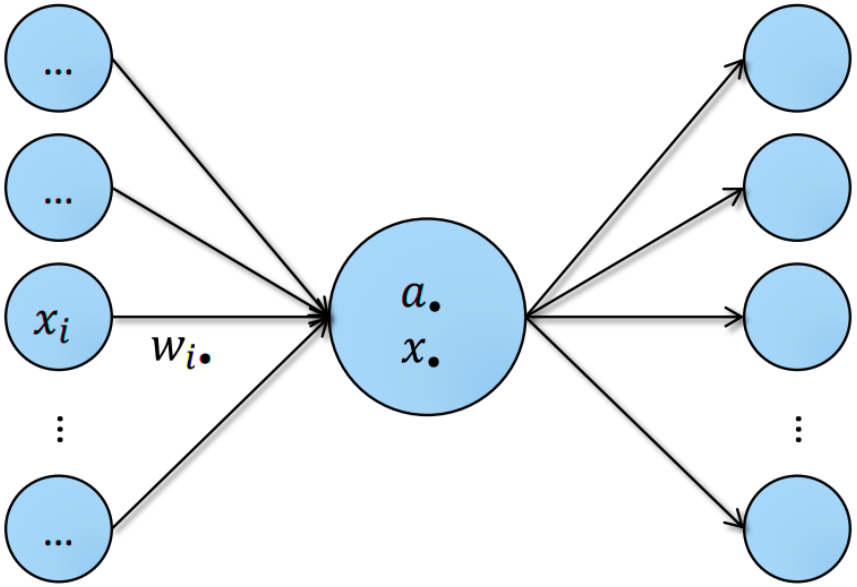
$$\frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i; W\right) + \lambda\, \Omega\left(W\right)$$

- Regularization
  - L2 regularization: sum of squares of weights
  - “Drop-out”: deactivate random subset of weights in each iteration
    - Similar to using many networks with less weights (shared among them)

- Training using simple gradient descend techniques
  - Stochastic gradient descend for large datasets (large N)
  - Estimate gradient of loss terms by averaging over a relatively small number of samples

Training the network: forward propagation

- Forward propagation from input nodes to output nodes
  - Accumulate inputs via weighted sum into activation
  - Apply non-linear activation function f to compute output
- Use Pre(j) to denote all nodes feeding into j



$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

Training the network: backward propagation

- Node activation and output

$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

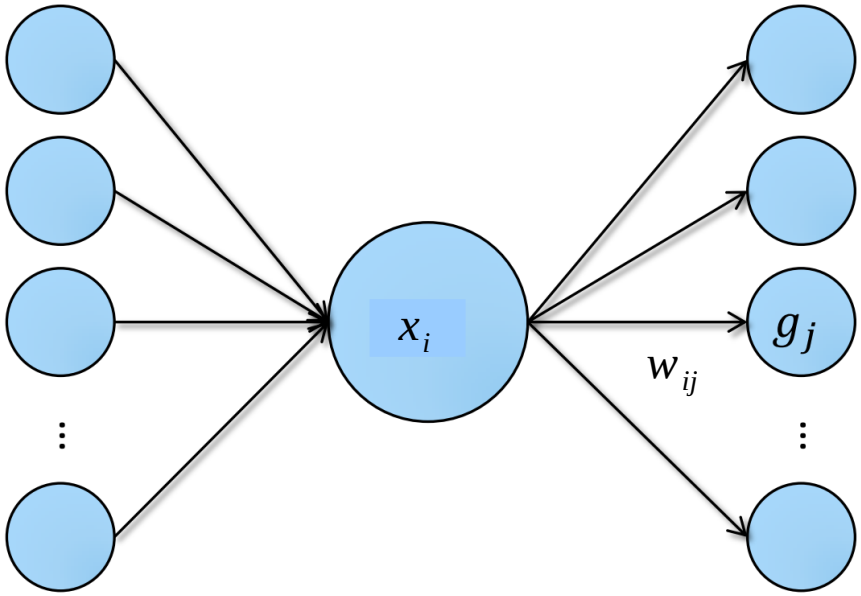
- Partial derivative of loss w.r.t. activation

$$g_j = \frac{\partial L}{\partial a_j}$$

- Partial derivative w.r.t. learnable weights

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

- Gradient of weight matrix between two layers given by outer-product of x and g



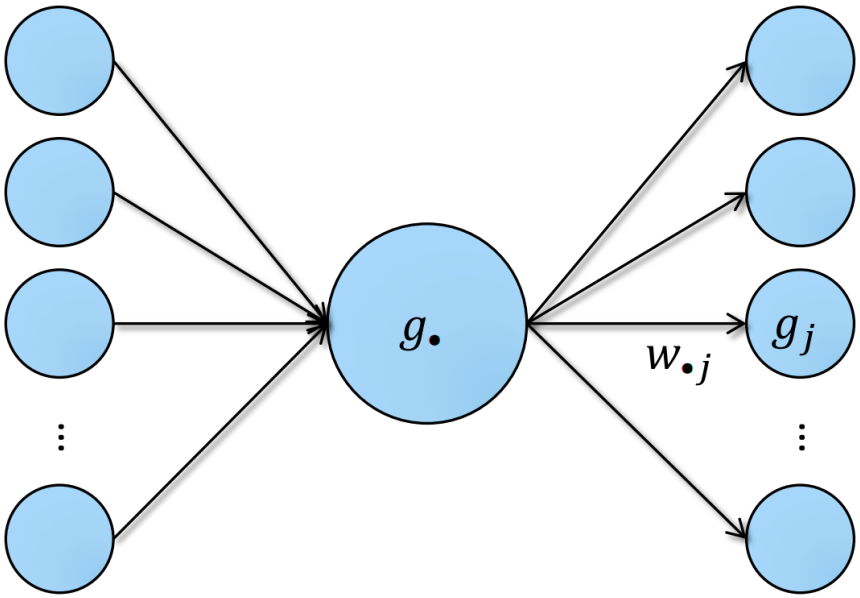
Training the network: backward propagation

- Back-propagation layer-by-layer of gradient from loss to internal nodes
  - Application of chain-rule of derivatives
- Accumulate gradients from downstream nodes
  - Post(i) denotes all nodes that i feeds into
  - Weights propagate gradient back
- Multiply with derivative of local activation function

$$a_j = \sum_{i \in Pre(j)} w_{ij} x_i$$

$$x_j = f(a_j)$$

$$g_i = \frac{\partial L}{\partial a_i}$$



$$\frac{\partial L}{\partial x_i} = \sum_{j \in Post(i)} \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial x_i}$$

$$= \sum_{j \in Post(i)} g_j w_{ij}$$

$$g_i = \frac{\partial x_i}{\partial a_i} \frac{\partial L}{\partial x_i}$$

$$= f'(a_i) \sum_{j \in Post(i)} w_{ij} g_j$$

Training the network: forward and backward propagation

- Special case for Rectified Linear Unit (ReLU) activations

$$f(a) = max(0, a)$$

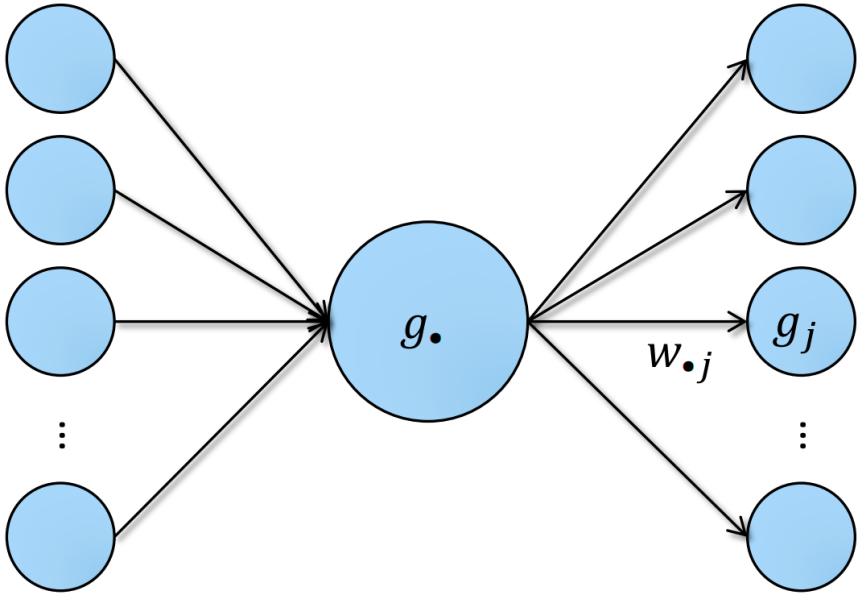
- Sub-gradient is step function

$$f'(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

- Sum gradients from downstream nodes

$$g_i = \begin{cases} 0 & \text{if } a_i \leq 0 \\ \sum_{j \in Post(i)} w_{ij} g_j & \text{otherwise} \end{cases}$$

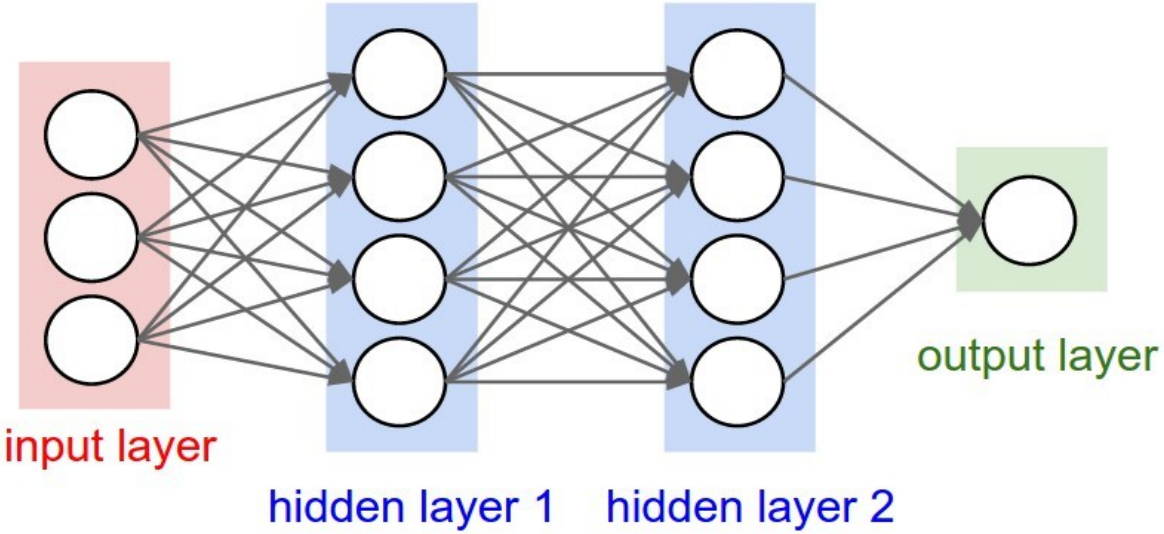
  - Set to zero if in ReLU zero-regime
  - Clip negative values in matrix vector product Wg



- Gradient on incoming weights is "killed" by inactive units
  - Generates tendency for those units to remain inactive

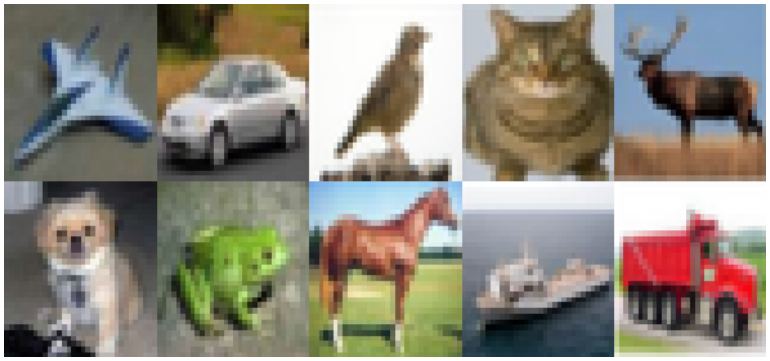
$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = g_j x_i$$

Convolutional Neural Networks



How to represent the image at the network input?

Input example : an image

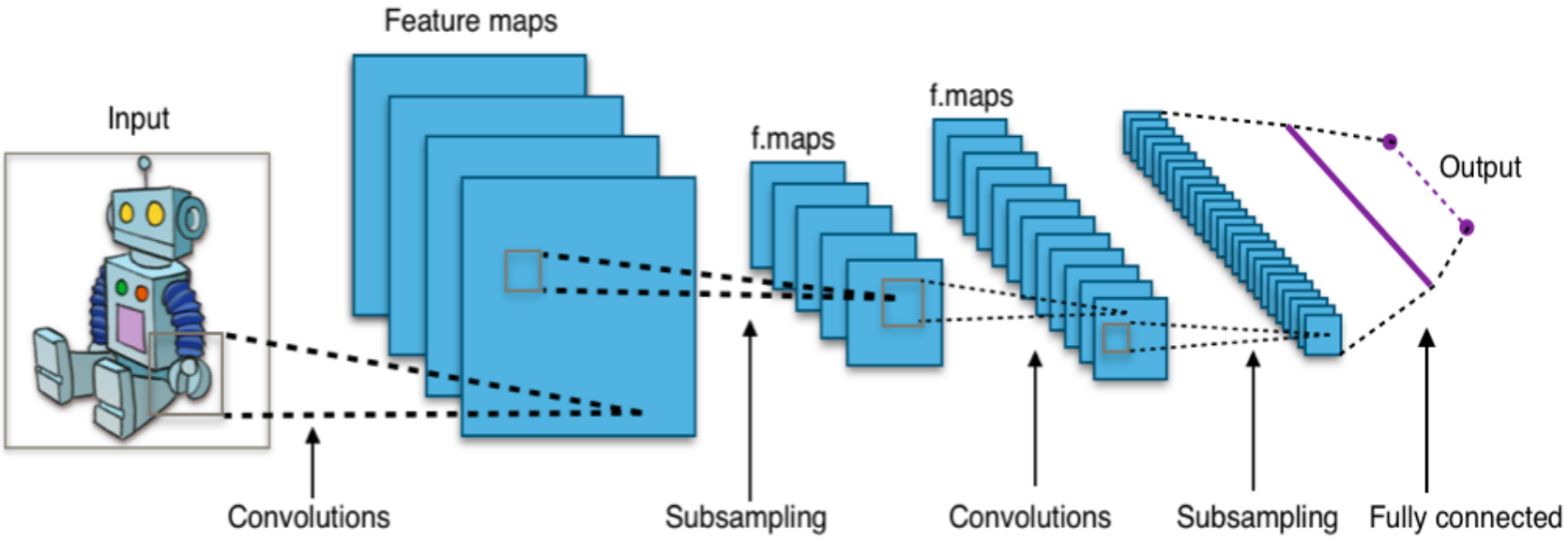


Output example: class label

airplane	dog
automobile	frog
bird	horse
cat	ship
deer	truck

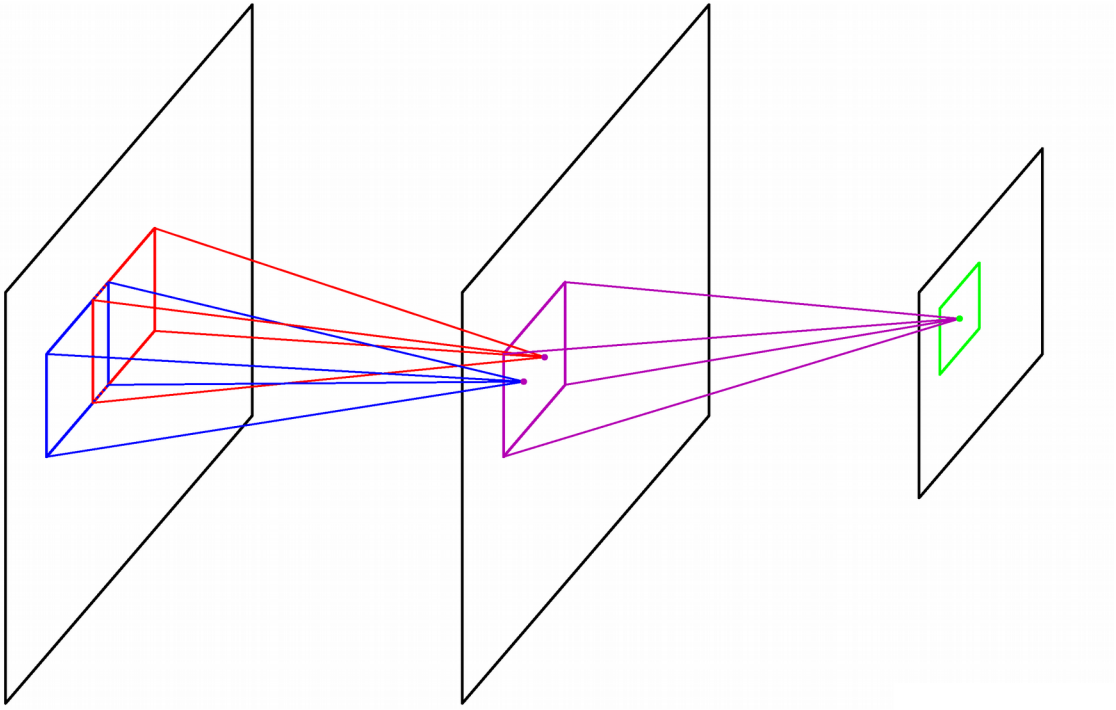
Convolutional neural networks

- A convolutional neural network is a feedforward network where
  - Hidden units are organizes into images or “response maps”
  - Linear mapping from layer to layer is replaced by convolution

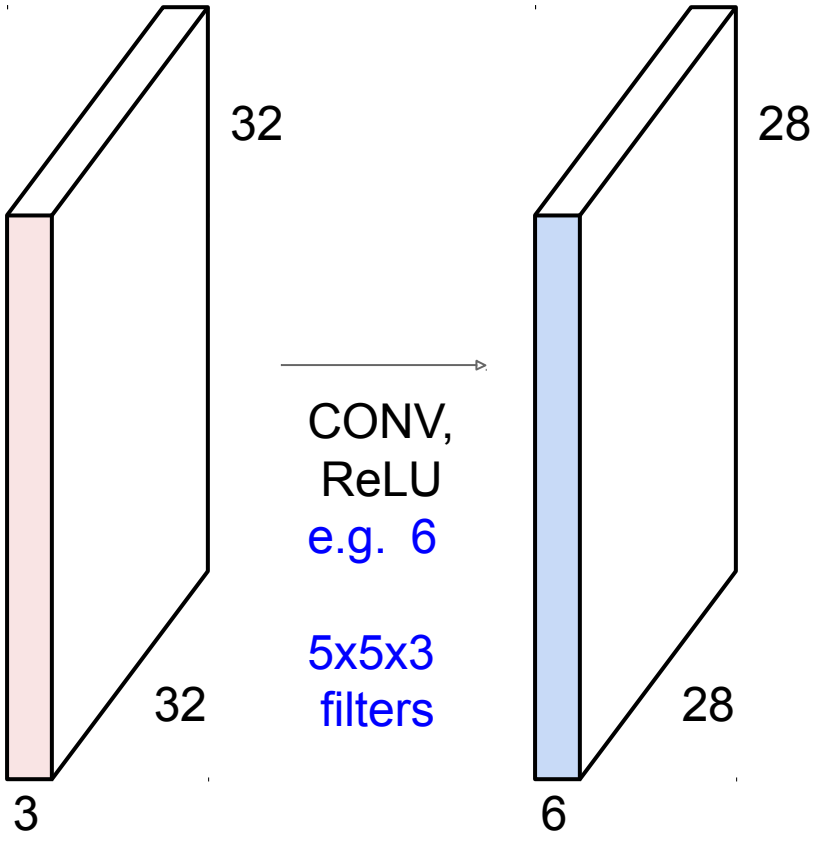


Convolutional neural networks

- Local connections: motivation from findings in early vision
  - Simple cells detect local features
  - Complex cells pool simple cells in retinotopic region
- Convolutions: motivated by translation invariance
  - Same processing should be useful in different image regions

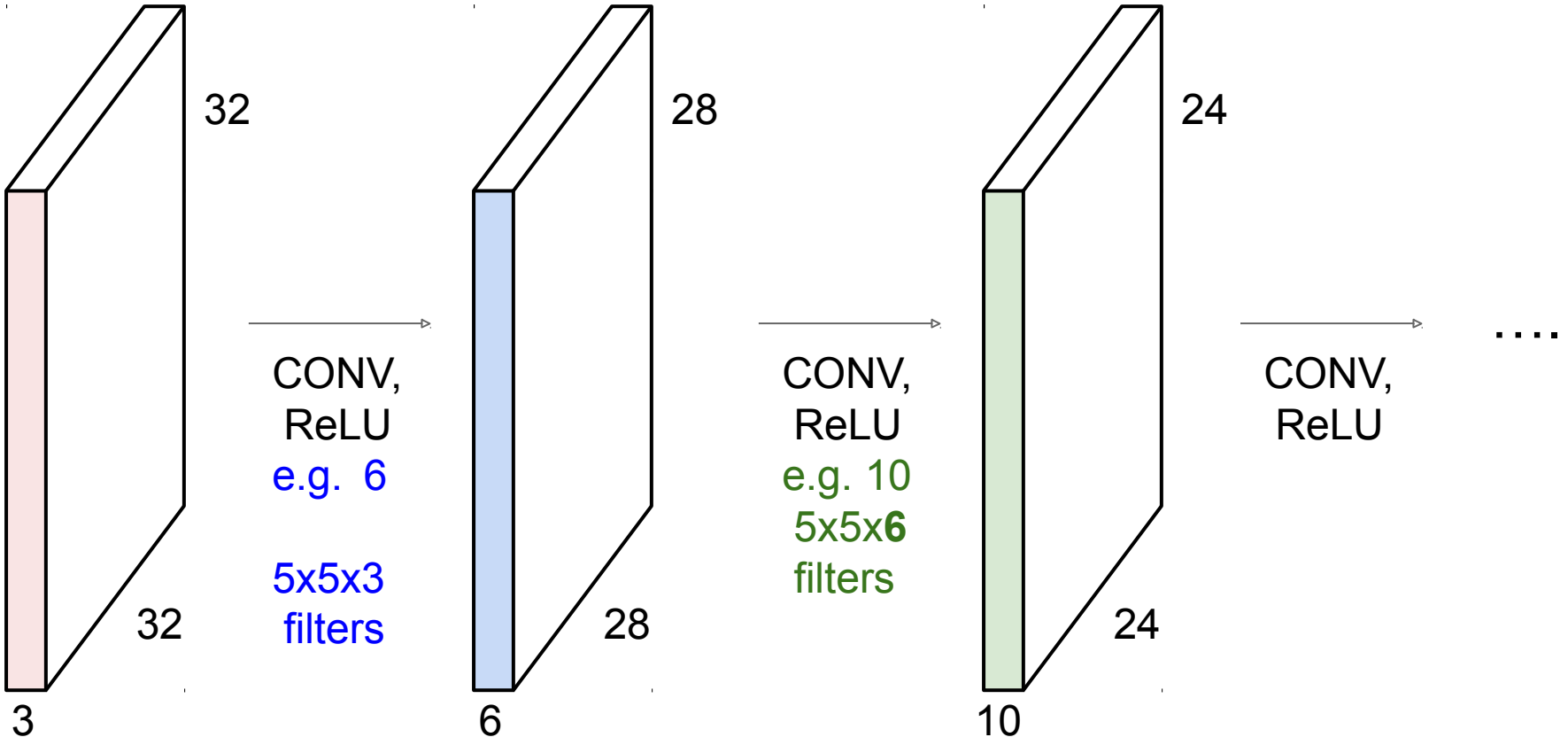


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



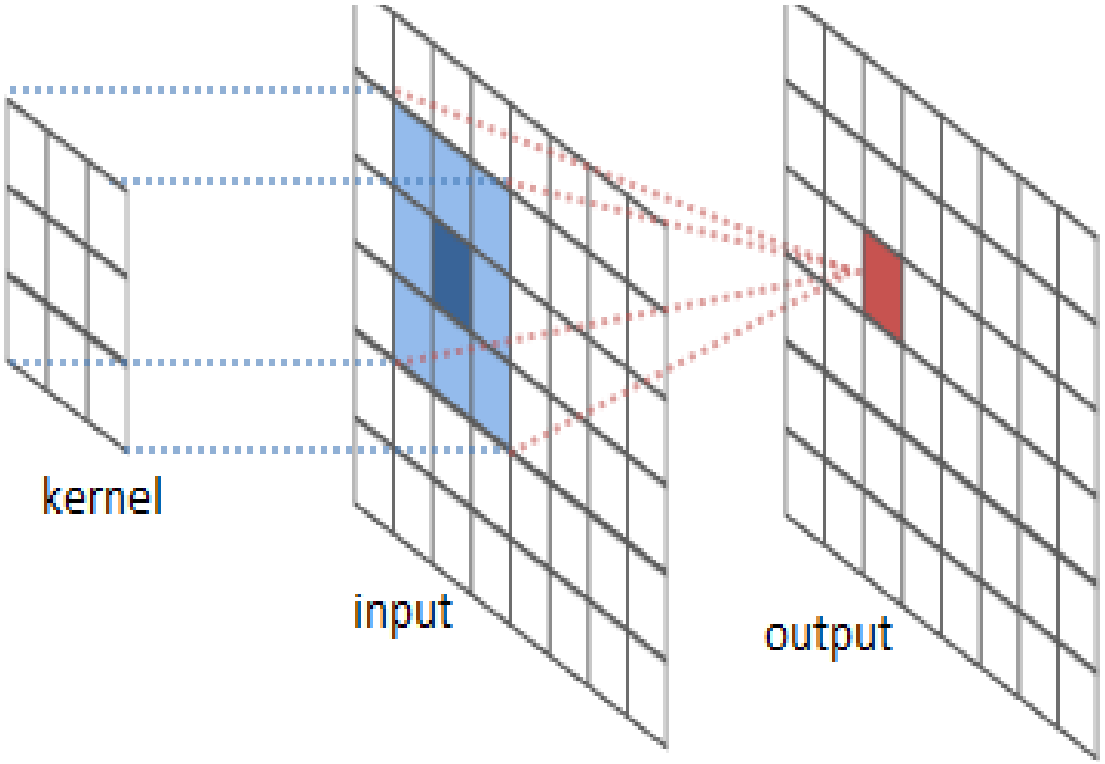
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

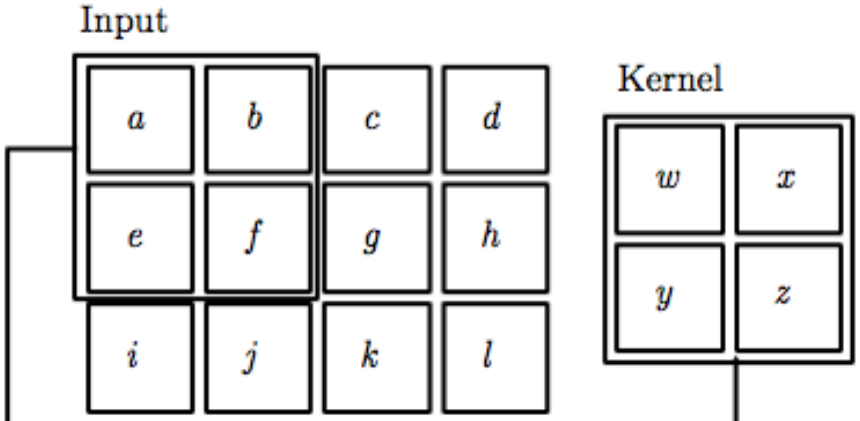


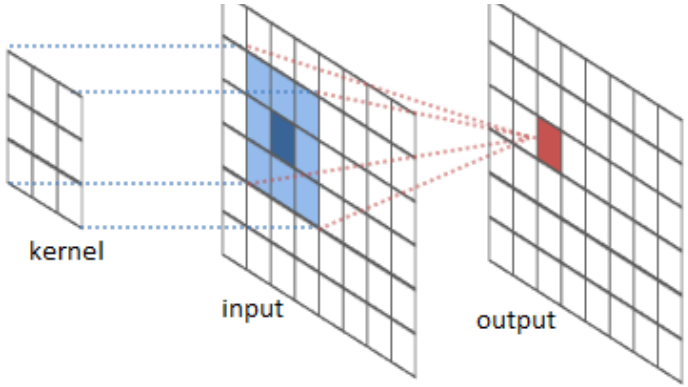
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

The convolution operation



The convolution operation





Output

$$aw + bx + ey + fz$$

$$bw + cx + fy + gz$$

$$cw + dx + gy + hz$$

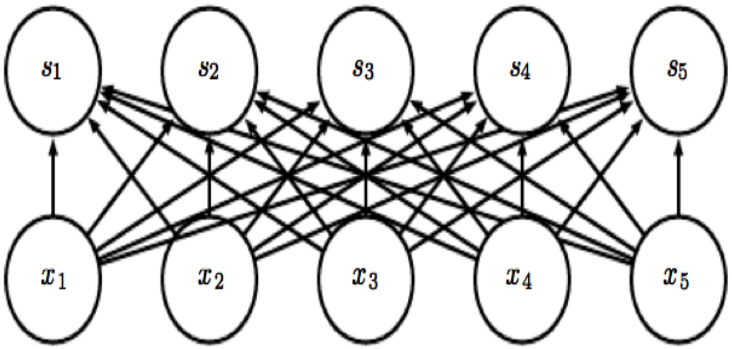
$$ew + fx + iy + jz$$

$$fw + gx + jy + kz$$

$$gw + hx + ky + lz$$

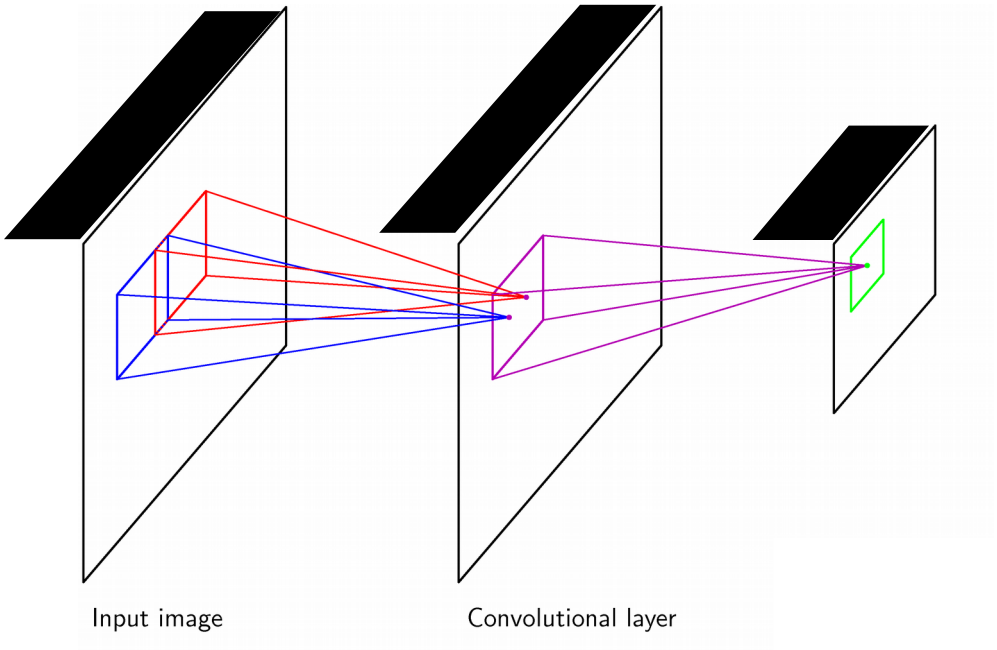
Local connectivity

Fully connected layer
as used in MLP



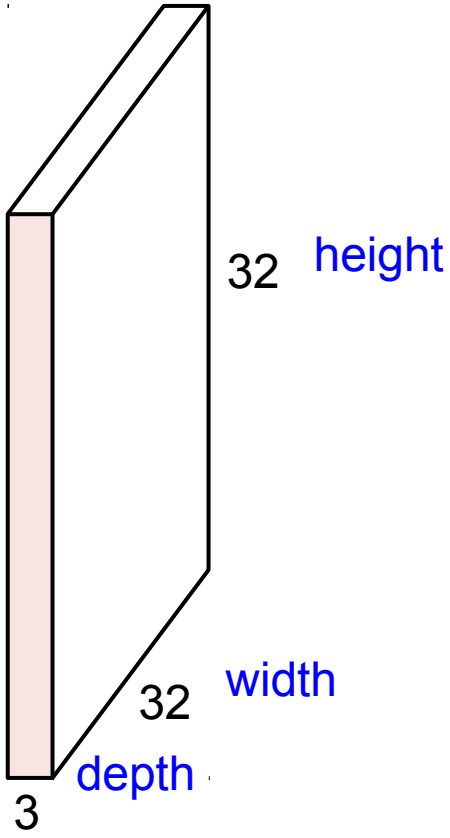
Convolutional neural networks

- Hidden units form another “image” or “response map”
  - Followed by point-wise non-linearity as in MLP
- Sharing of weights across spatial positions decouples the number of parameters from input and representation size
  - Enables training of models for large input images
  - Allows same model to be run over images of different size



Convolution Layer

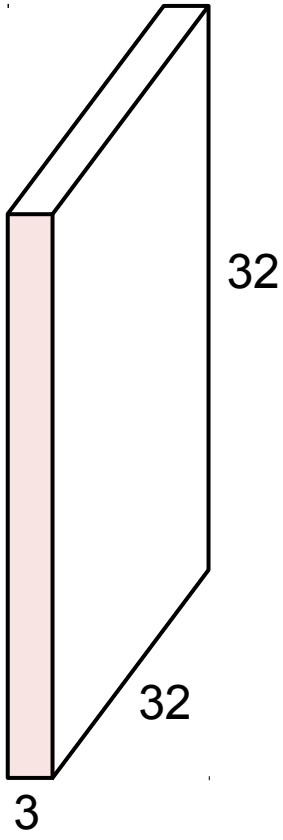
32x32x3 image



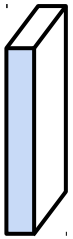
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

32x32x3 image



5x5x3 filter

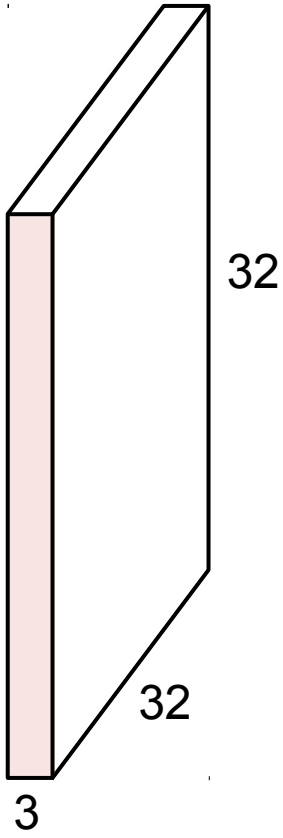


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

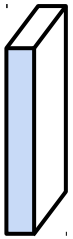
Convolution Layer

32x32x3 image



Filters always extend the full depth of the input volume

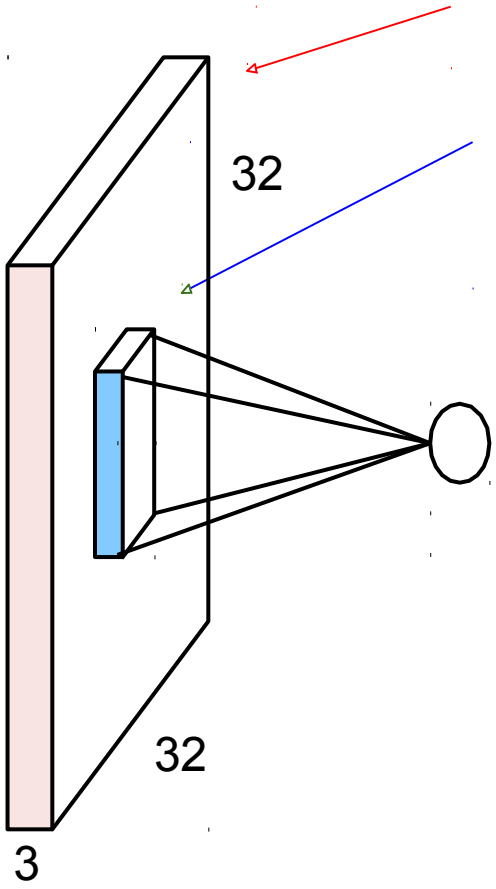
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

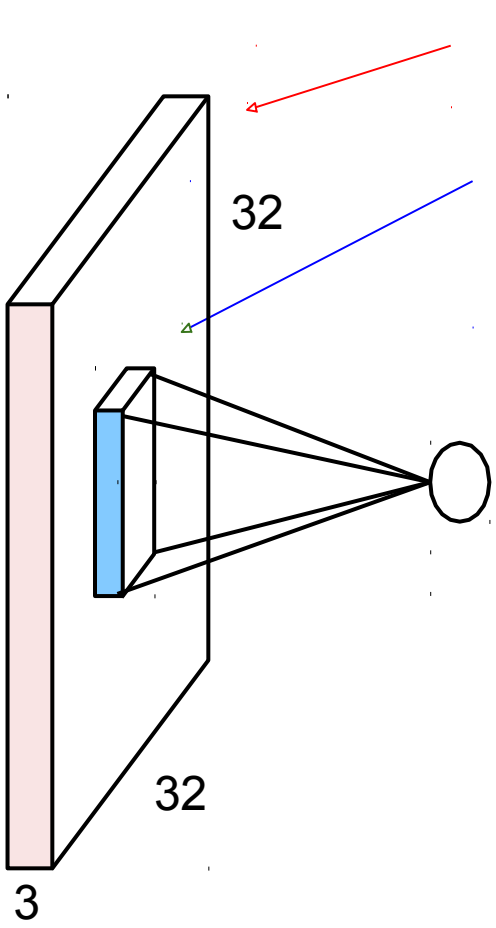


1 hidden unit:
dot product between 5x5x3=75 input patch and weight vector + bias

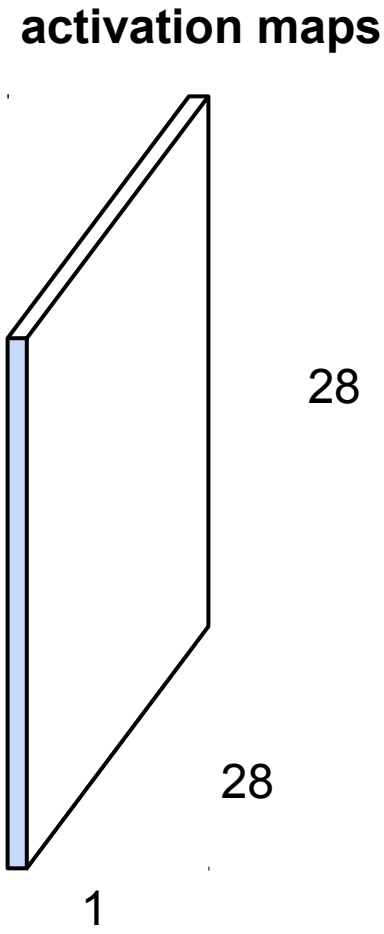
$$w^T x + b$$

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer

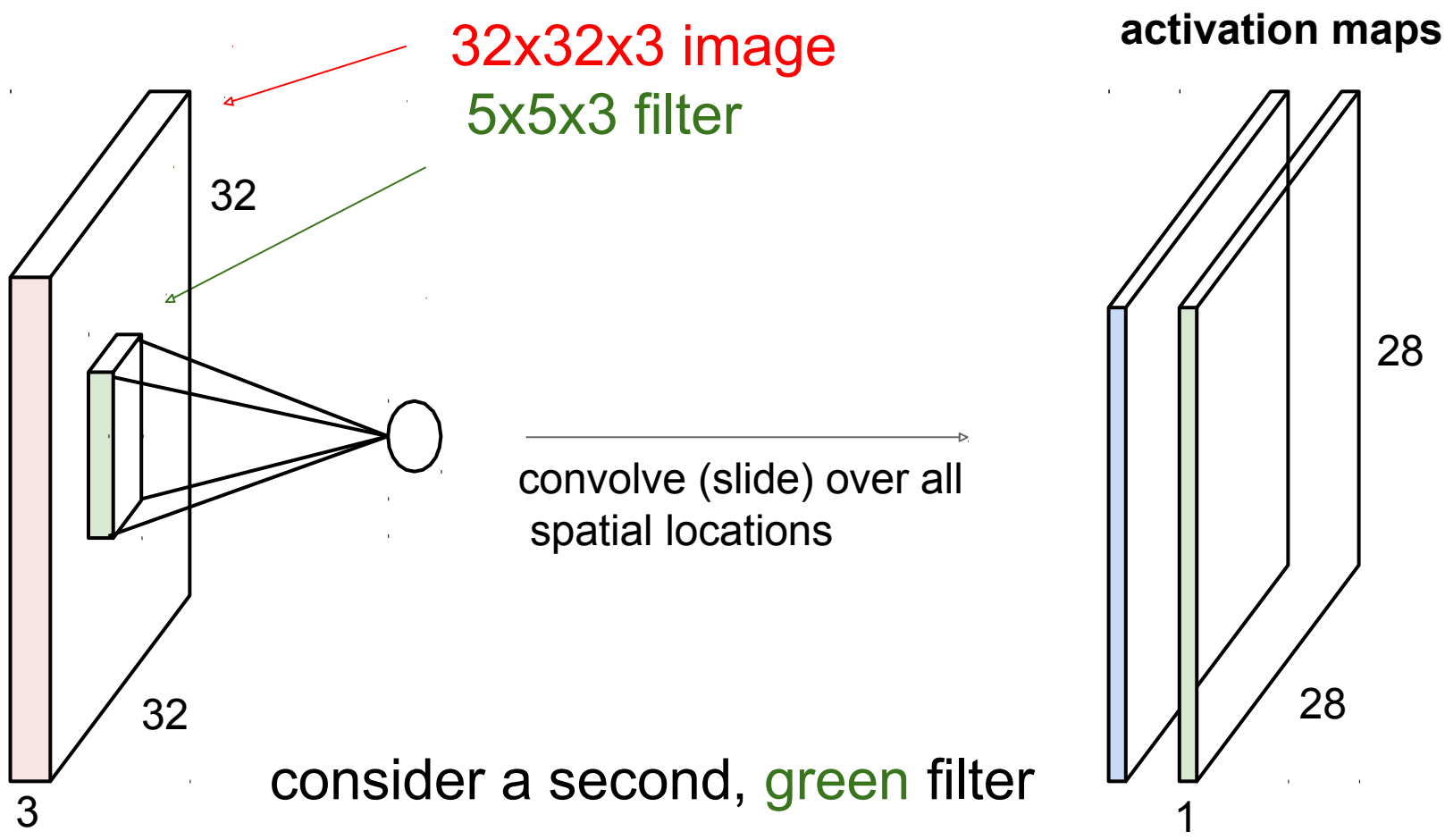


convolve (slide) over all spatial locations



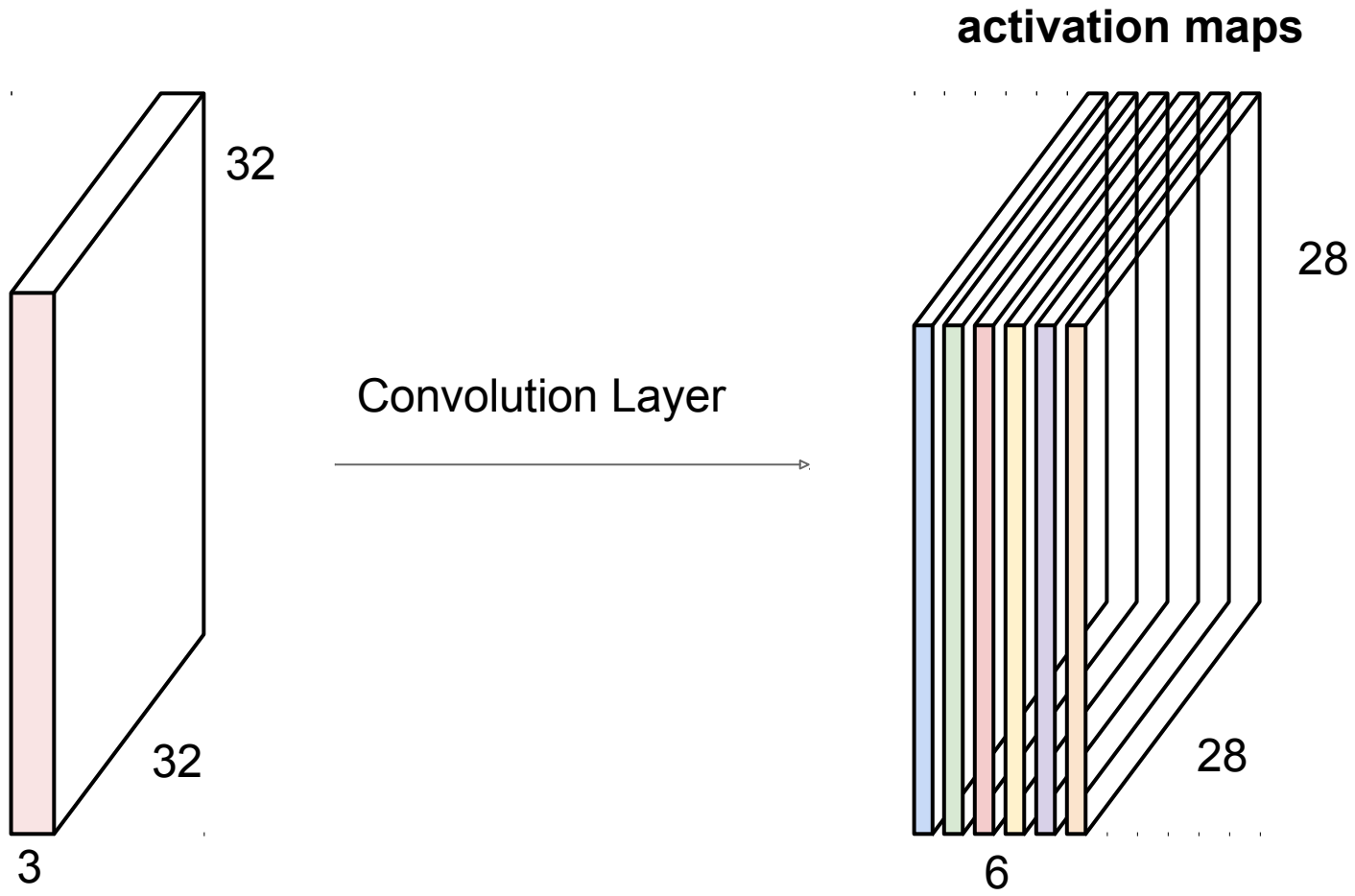
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Convolution Layer



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

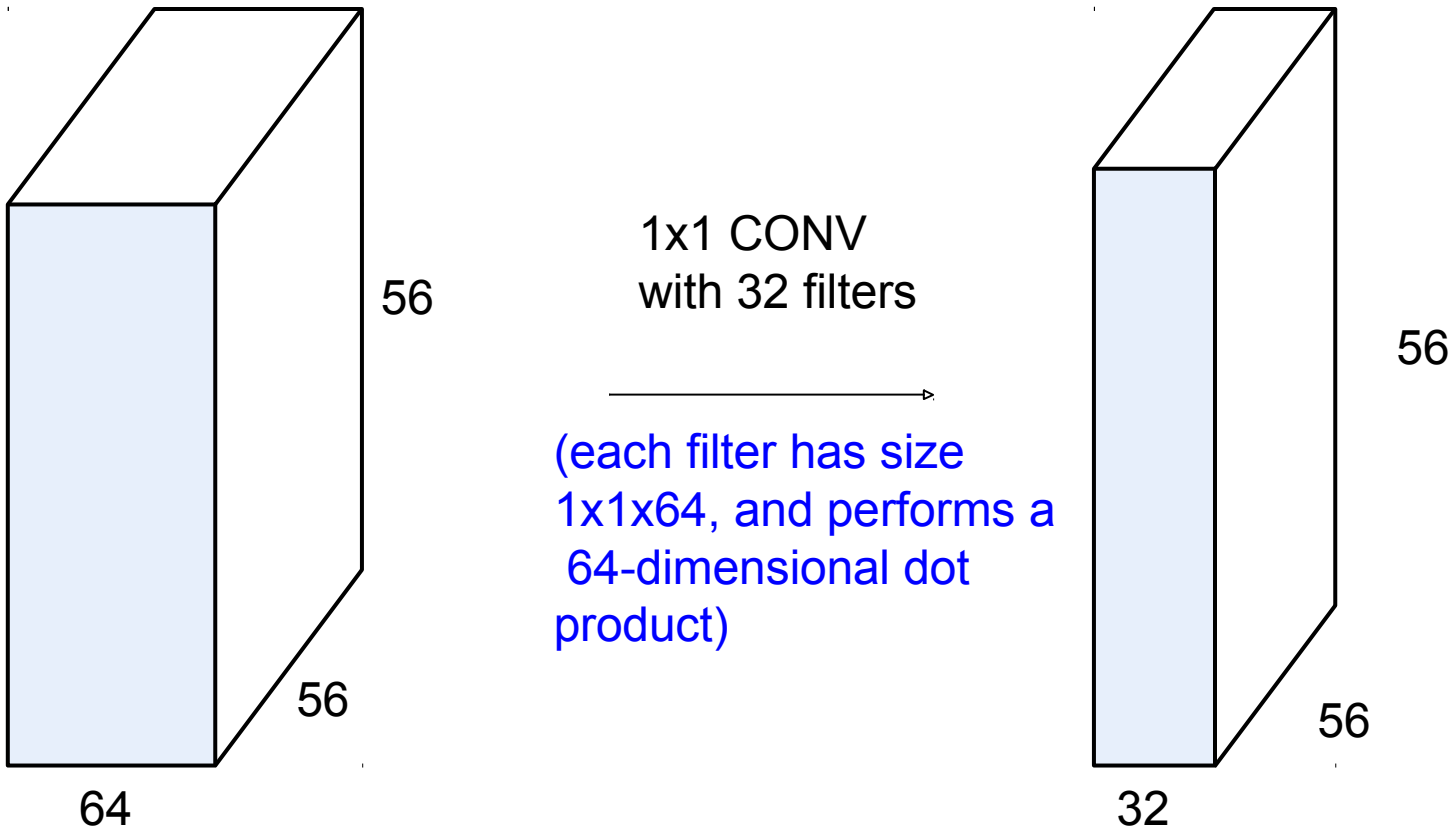
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

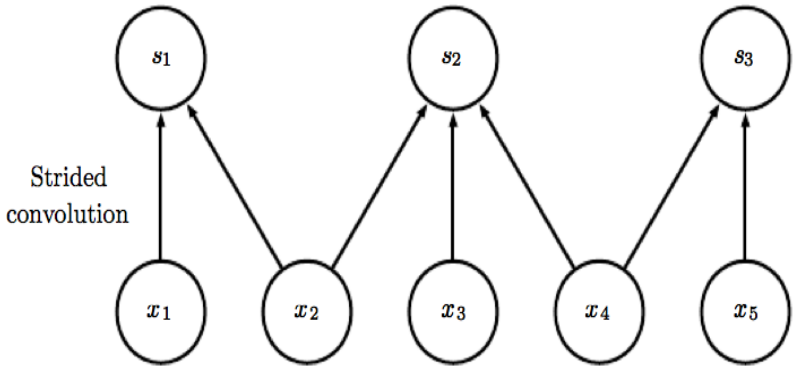
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

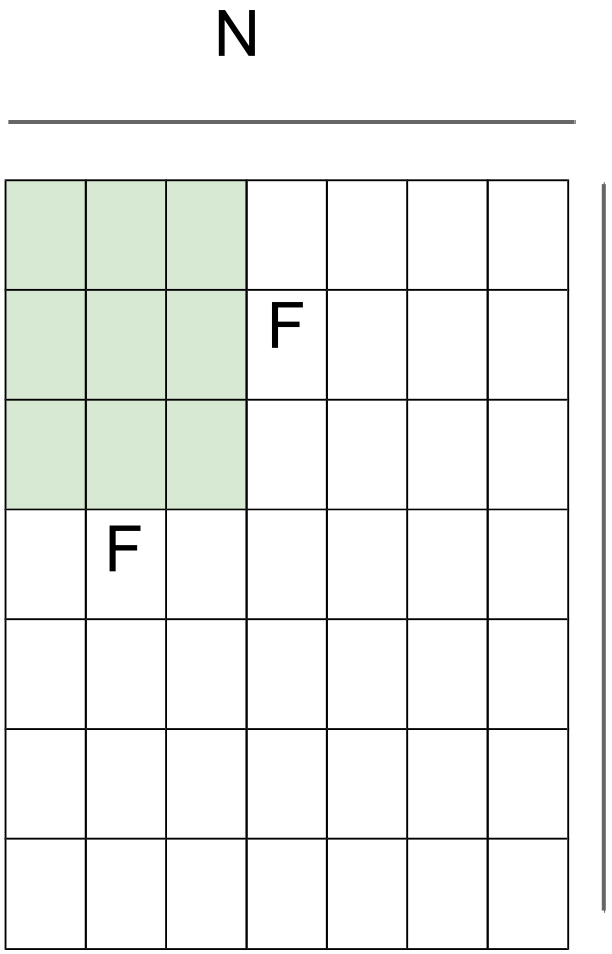
Convolution with 1x1 filters makes perfect sense



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Stride



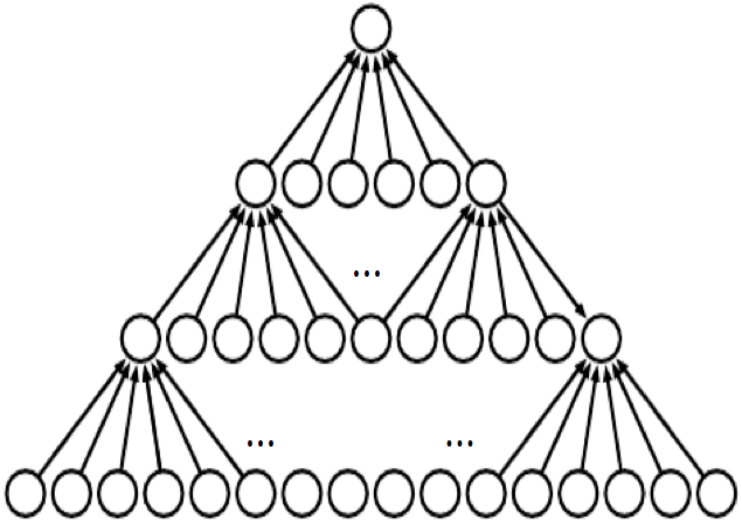


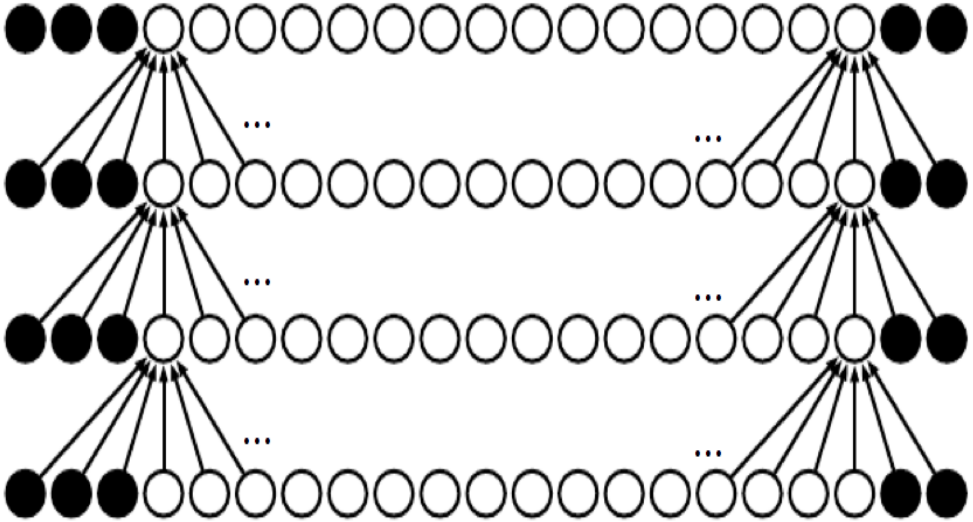
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

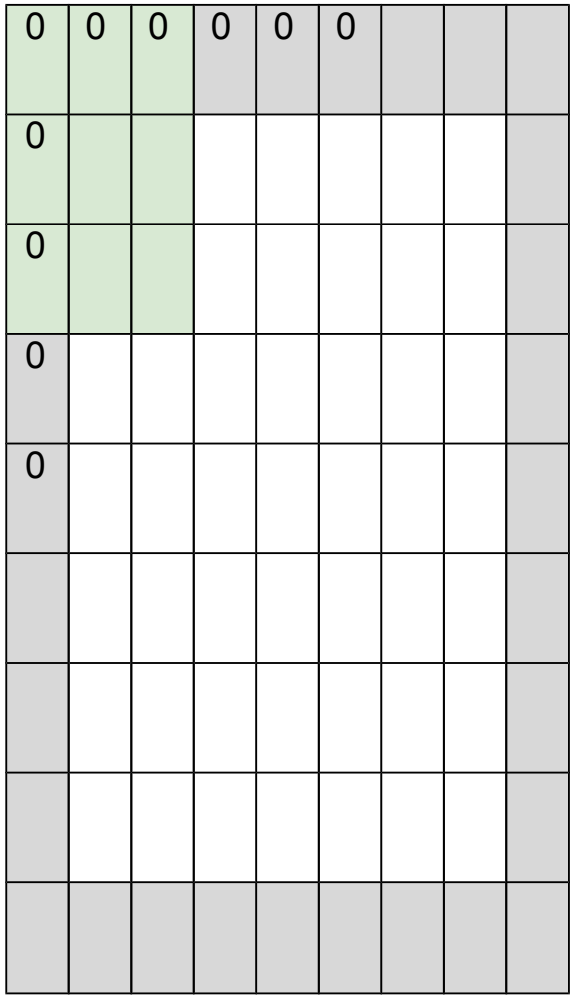
slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

(Zero)-Padding





Zero-Padding: common to the border



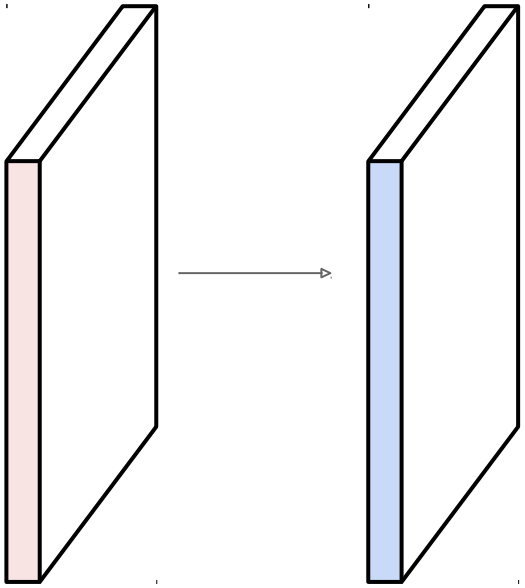
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

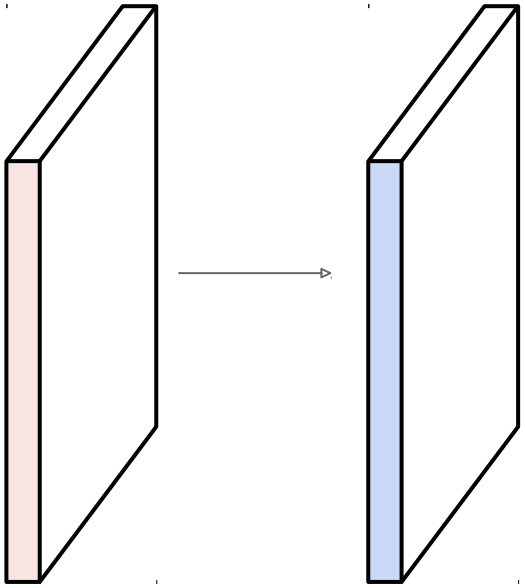
Example:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Example:

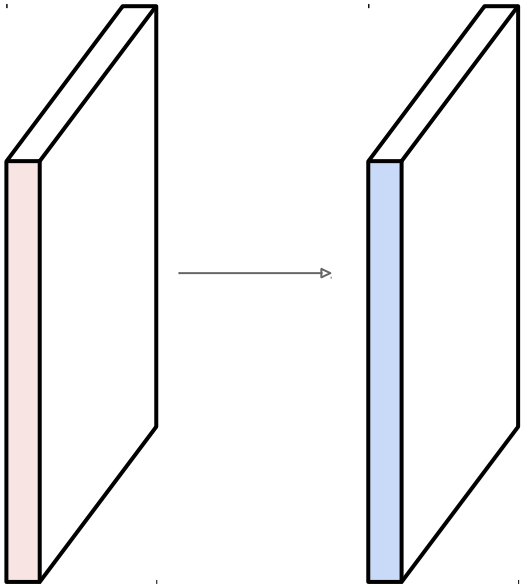


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

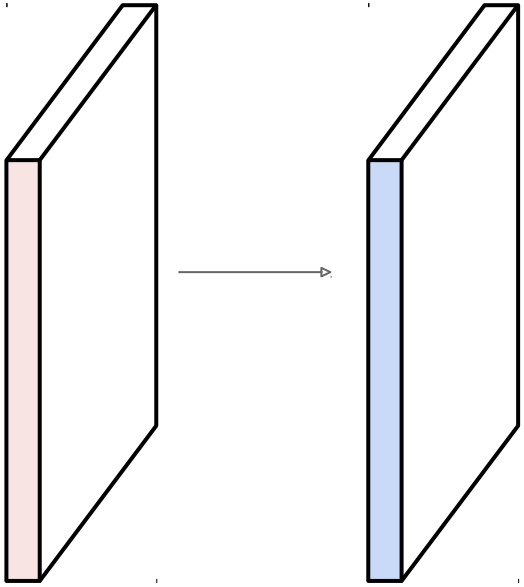
Example:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Example:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params
=> 76*10 = **760**

(+1 for bias)

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Common settings:

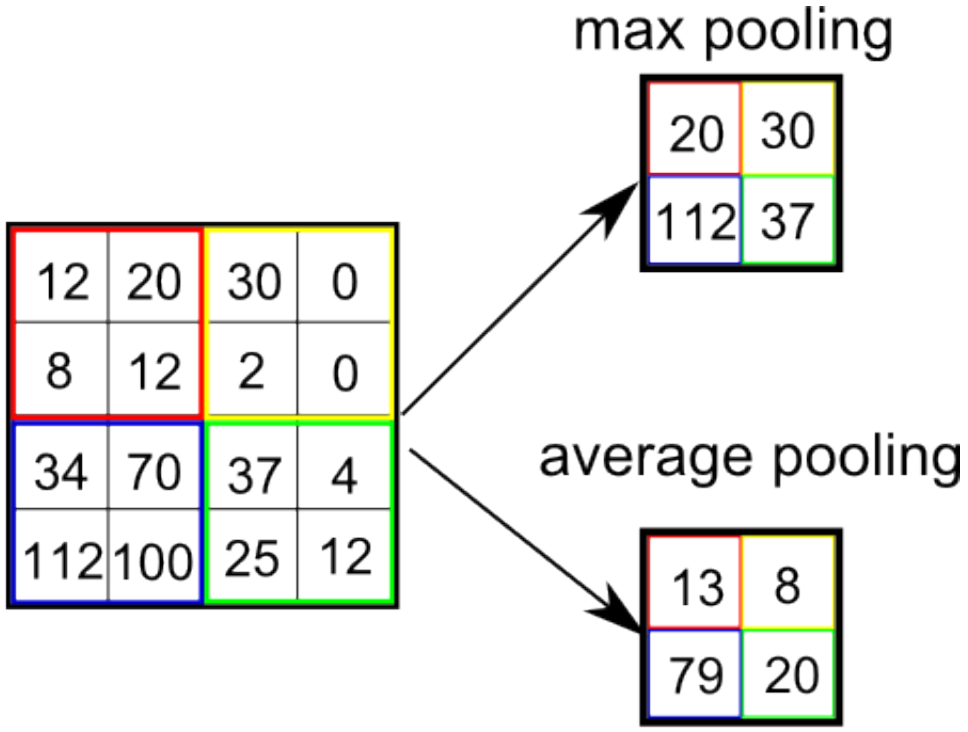
K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

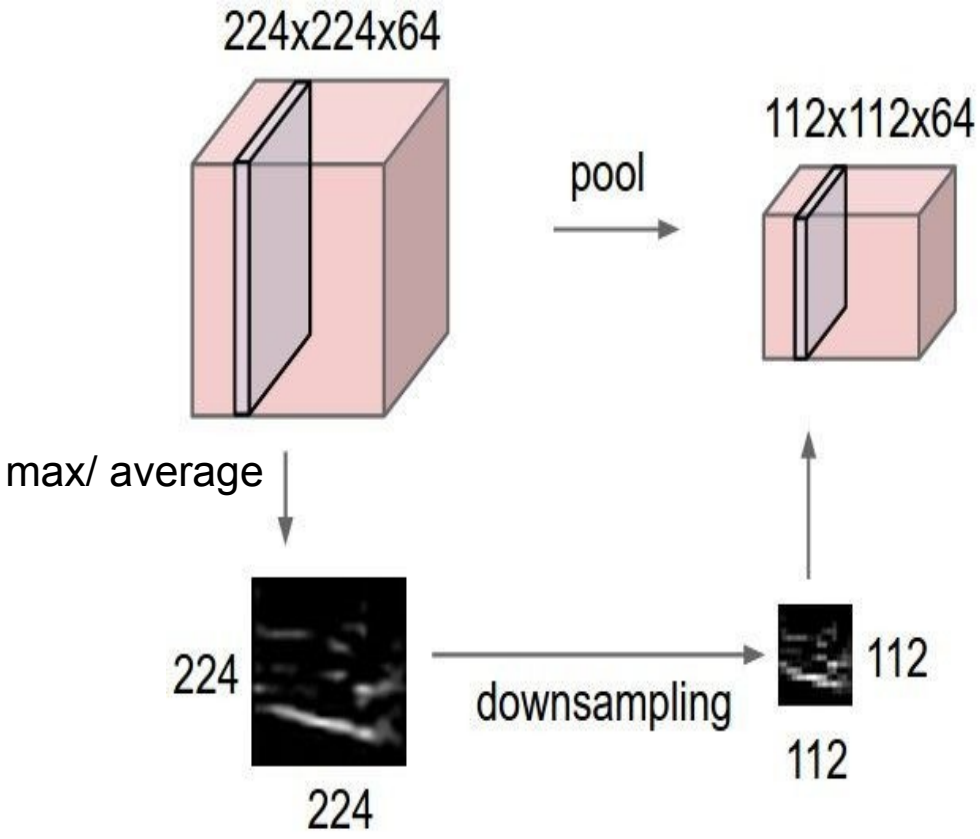
Pooling



Effect = invariance to small translations of the input

Pooling

- Makes representation smaller and computationally less expensive
- Operates over each activation map independently



slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Summary

Common settings:

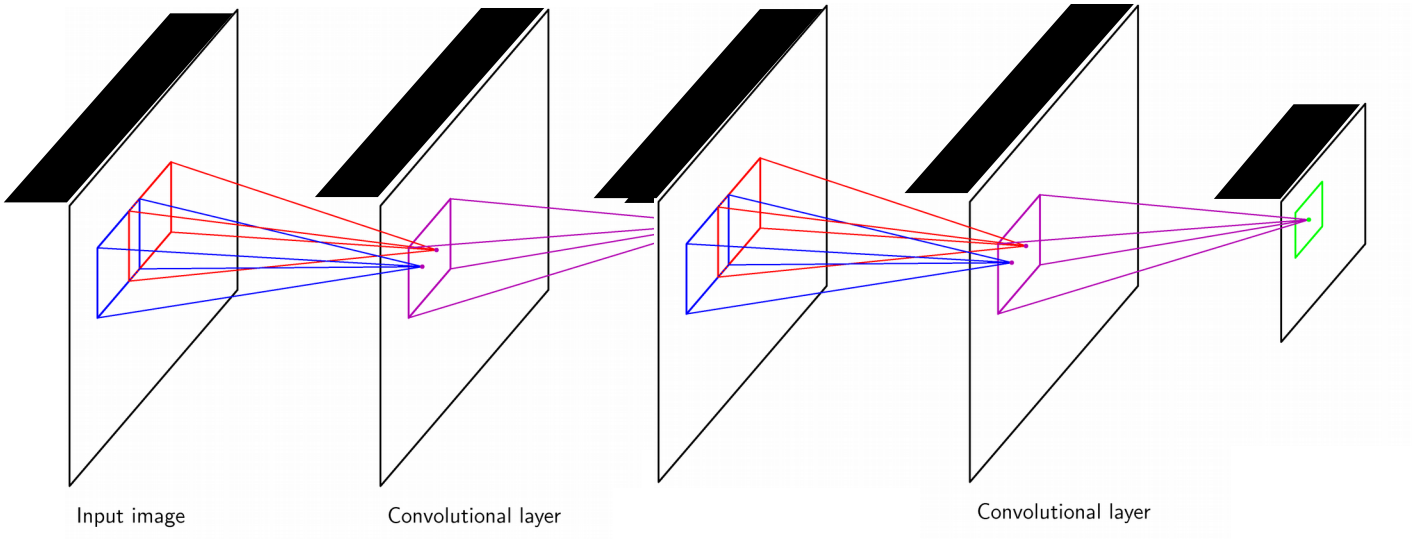
F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

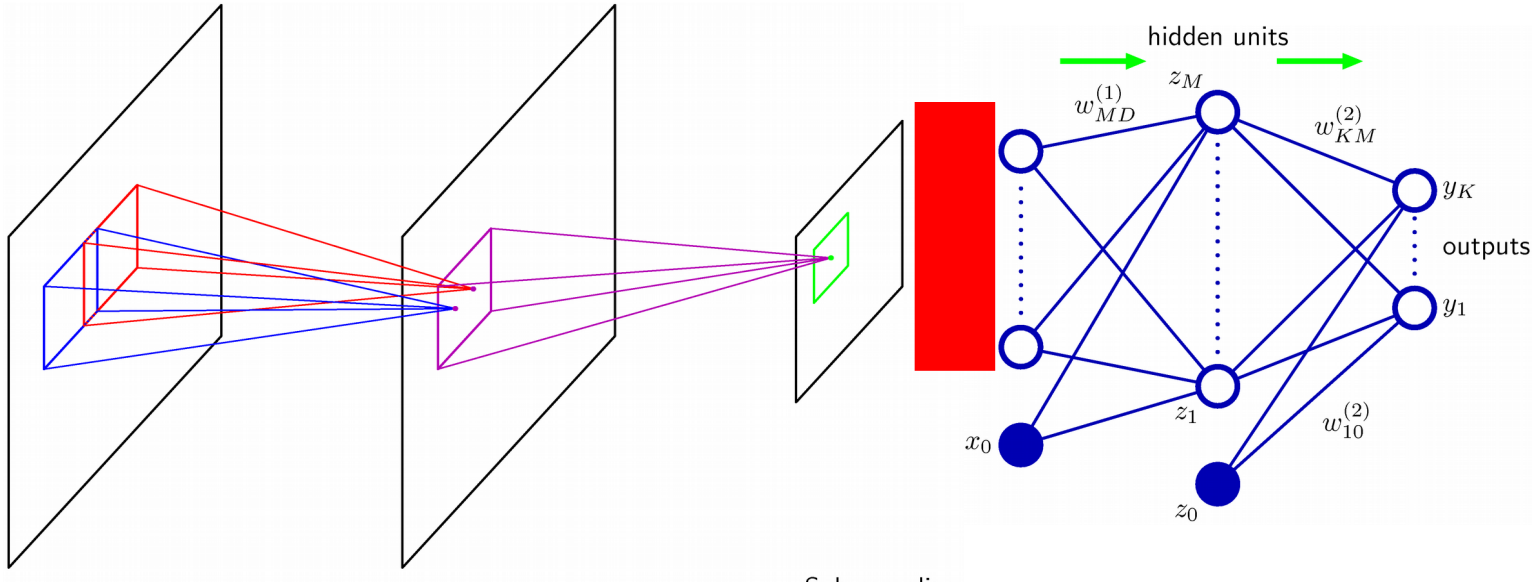
Receptive fields

- "Receptive field" is area in original image impacting a certain unit
  - Later layers can capture more complex patterns over larger areas
- Receptive field size grows linearly over convolutional layers
  - If we use a convolutional filter of size w x w, then each layer the receptive field increases by (w-1)
- Receptive field size increases exponentially over layers with striding
  - Regardless whether they do pooling or convolution



Fully connected layers

- Convolutional and pooling layers typically followed by several “fully connected” (FC) layers, i.e. a standard MLP
  - FC layer connects all units in previous layer to all units in next layer
  - Assembles all local information into global vectorial representation
- FC layers followed by softmax for classification
- First FC layer that connects response map to vector has many parameters
  - Conv layer of size 16x16x256 with following FC layer with 4096 units leads to a connection with 256 million parameters !
  - Large 16x16 filter without padding gives 1x1 sized output map



Weights initialization

- If the weights in a network start too small,
  then the signal shrinks as it passes through each layer until it's too tiny to be useful.
- If the weights in a network start too large,
  then the signal grows as it passes through each layer until it's too massive to be useful.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Weights initialization

- All zero initialization

- Small random numbers

- Draw weights from a Gaussian distribution
  with standard deviation of sqrt(2/n),
  where n is the number of outputs to the neuron
  - Ensures the (gradient) signal does roughly stays on the same scale from one layer to the next

Batch normalization

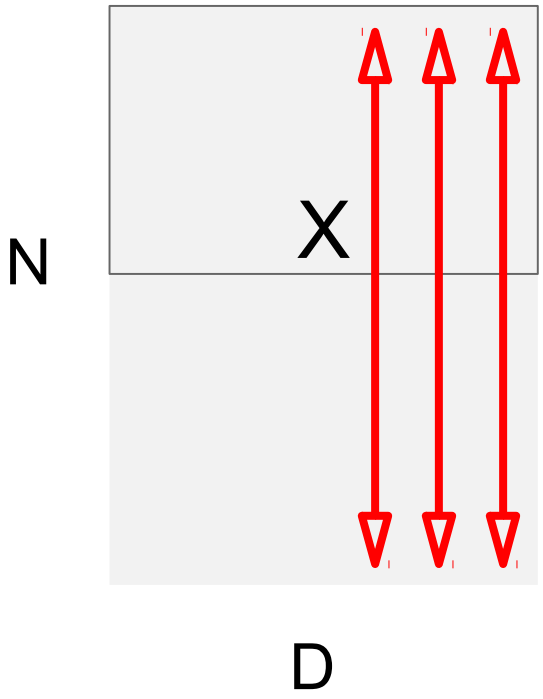
Initialization of NNs by explicitly forcing the activations throughout the network to take on a unit Gaussian distribution at the beginning of the training.

Normalization is a simple differentiable operation

[Ioffe and Szegedy, 2015]

Batch normalization

"you want unit gaussian activations? just make them so."



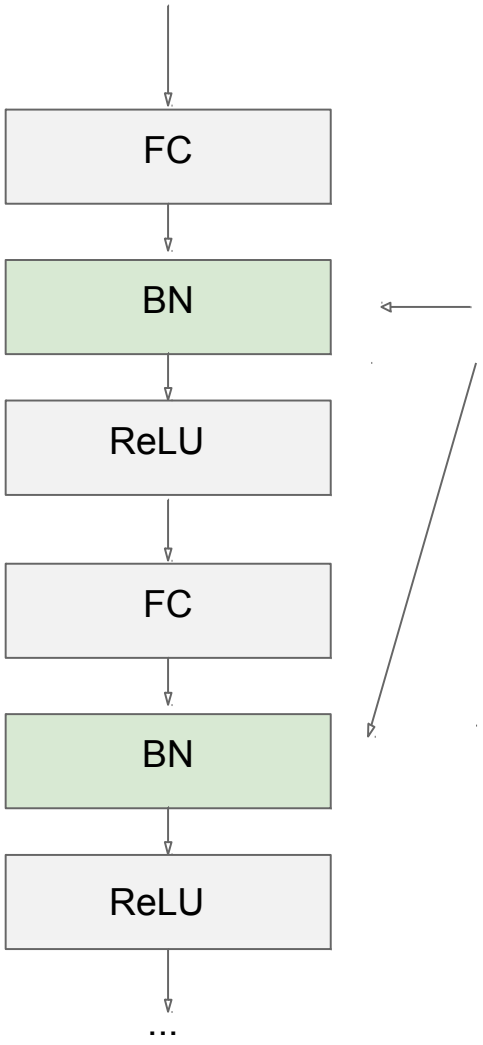
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Batch normalization



Usually inserted after Fully Connected and/or Convolutional layers, and before nonlinearity.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Batch normalization

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

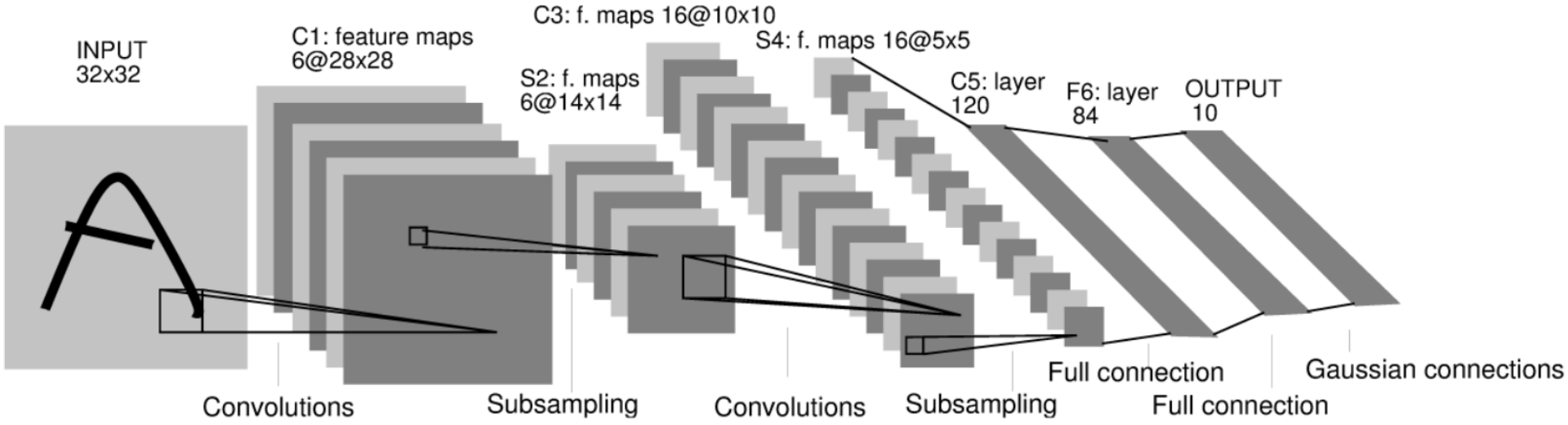
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Separates direction of weight vectors and their magnitude

- Instead of normalizing the activations, we can also normalize the weights to a similar effect [Salimans and Kingma, NIPS 2016]

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

CNN architectures: LeNet (1998)

- C1: 5x5 filters, outputs 6 channels, 156 = 6 (5x5 + 1) parameters
- S2: “average” pooling, times constant + bias, 12 parameters
- C3: 5x5 filters, outputs 16 channels, 1516 < 16 x 6 x (5x5 + 1) parameters
  - C3 features cannot see all S2 features
- S4: “average” pooling, times constant + bias, 32 parameters
- C5: 5x5 filters, outputs 120 channels, 49920 = 120 x16 (5x5 + 1) param.
- F6: fully connected, 84 outputs, 10080 = 84 x 120 parameters
- Final layer: 10 outputs, 840 = 84 x 10 parameters



[LeCun, Bottou, Bengio, Haffner, Proceedings IEEE, 1998]

What has changed

- Large training datasets for computer vision
  - 1.2 millions of 1000 classes in ImageNet challenge [Deng et al, CVPR'09]
  - 200 million faces to train face recognition nets [Schroff et al., CVPR 2015]
- GPU-based implementation: 1 to 2 orders of magnitude faster than CPU
  - Parallel computation for matrix products
  - Krizhevsky & Hinton, 2012: six days of training on two GPUs
  - Rapid progress in GPU compute performance
- Network architectures

Trends in recent influential CNN architectures

- More layers
- Smaller filters
- ReLU non-linearity
- Strided conv. rather than max/average pooling

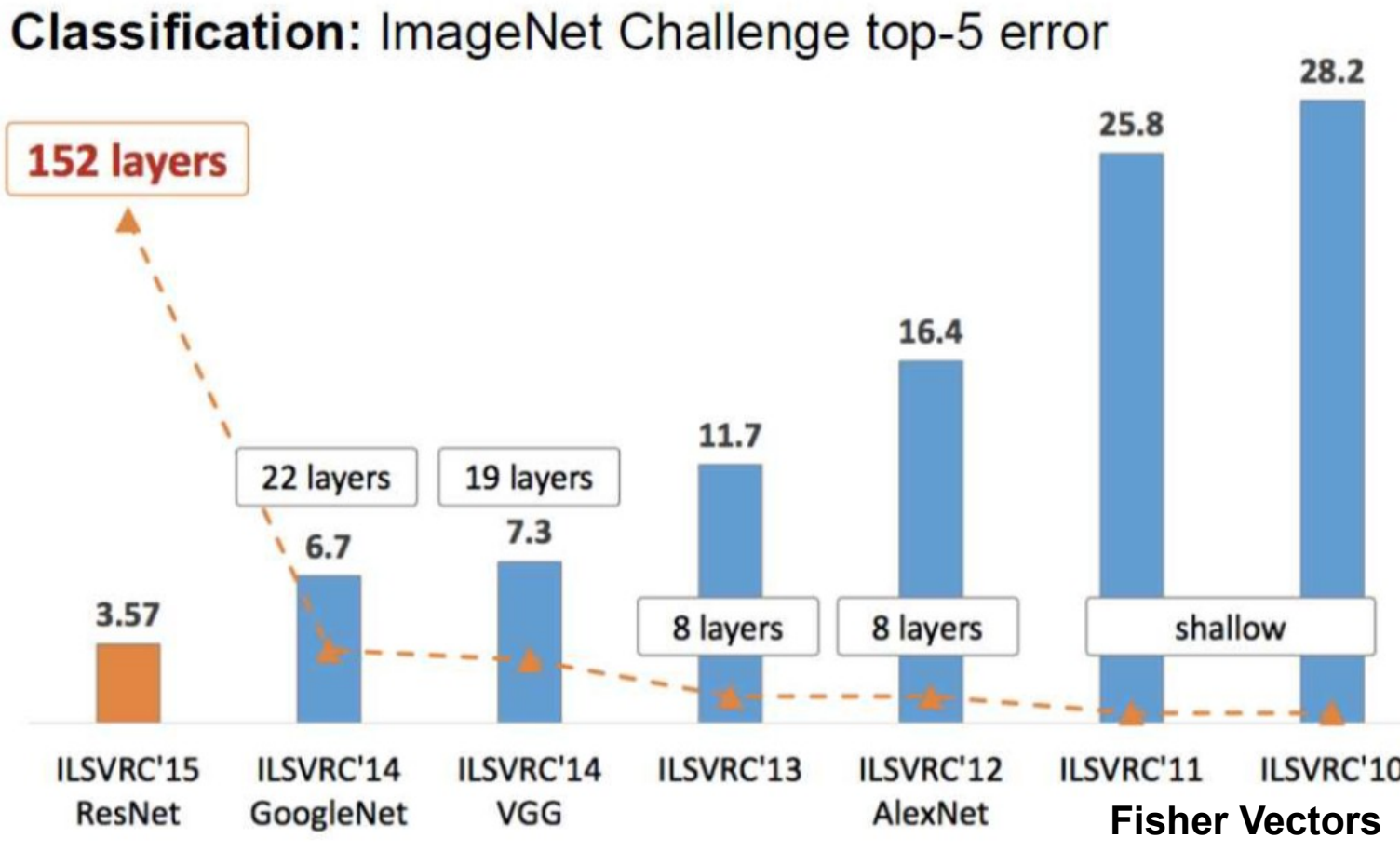
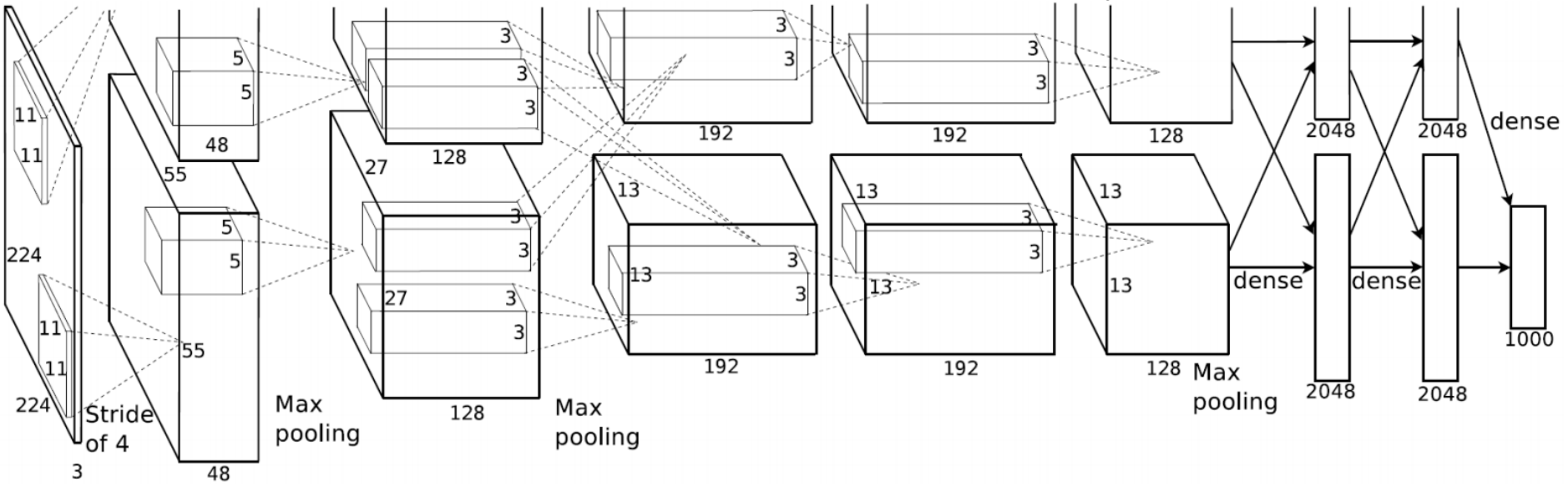


Figure: Kaiming He

AlexNet CNN Architecture (2012)

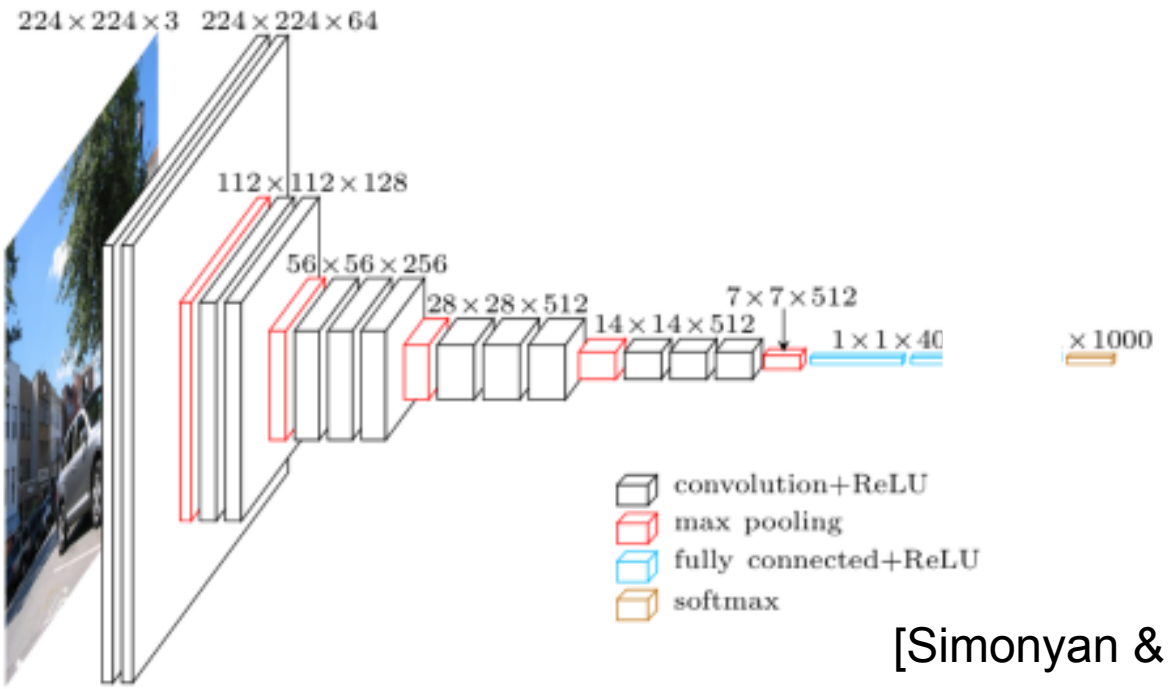
- Winner ImageNet 2012 image classification challenge, huge impact
  - CNNs improving “traditional” computer vision techniques on uncontrolled images, rather than datasets of small (eg 32x32) and controlled images
- Compared to LeNet
  - Inputs at 224x224 rather than 32x32
  - Distributed implementation over 2 GPUs
  - 5 rather than 3 conv layers
  - More feature channels in each layer
  - ReLU non-linearity

[Alex Krizhevsky & Geoff Hinton, NIPS 2012]



VGG “very-deep” CNN Architecture

- Double the number of layers 16 or 19 (from 8 in AlexNet)
- Only small 3x3 filters, rather than filters up to size 11 AlexNet
  - Large filters approximated by sequence of smaller ones, receptive field increases, smaller nr of parameters due to factorization of weights
- About 140 million parameters (AlexNet ~60 million)

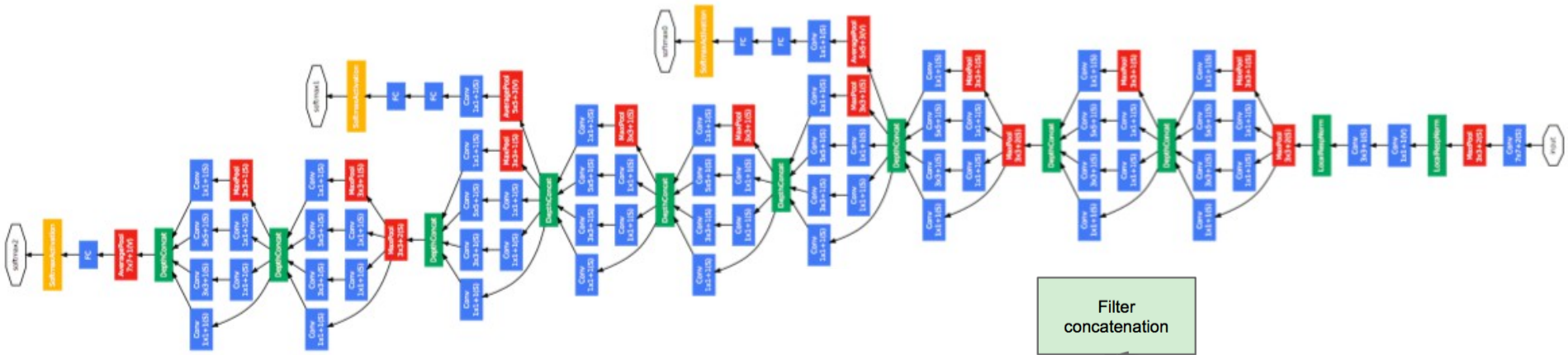


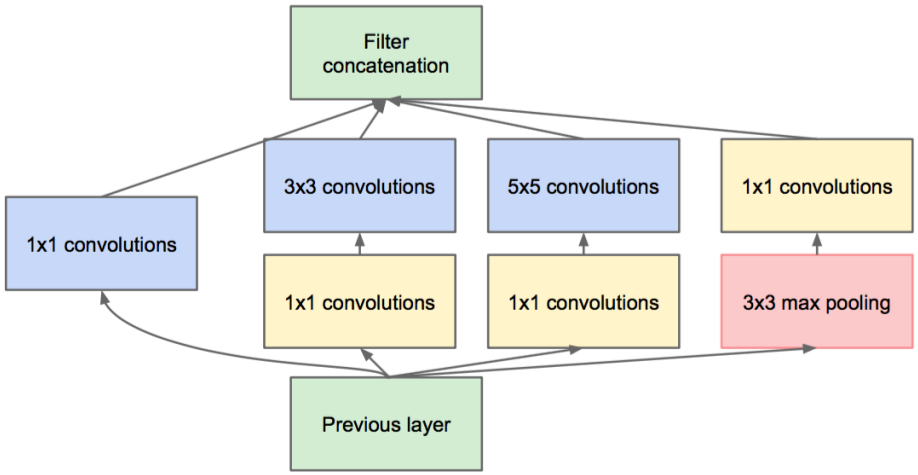
[Simonyan & Zisserman, ICLR ‘15]

Winner ImageNet 2014 challenge

GoogleNet Inception CNN Architecture

- Reduced number of parameters: 5 million (60m AlexNet, 140m VGG)
- Inception module: compress features before convolution
- Replaces fully-connected layer with average pooling
- Intermediate loss functions improve training of early layers





[Szegedy et al, CVPR 2015]

Winner ImagetNet 2015 challenge

ResNet: Residual CNN Architecture

[He et al. ECCV 2016]

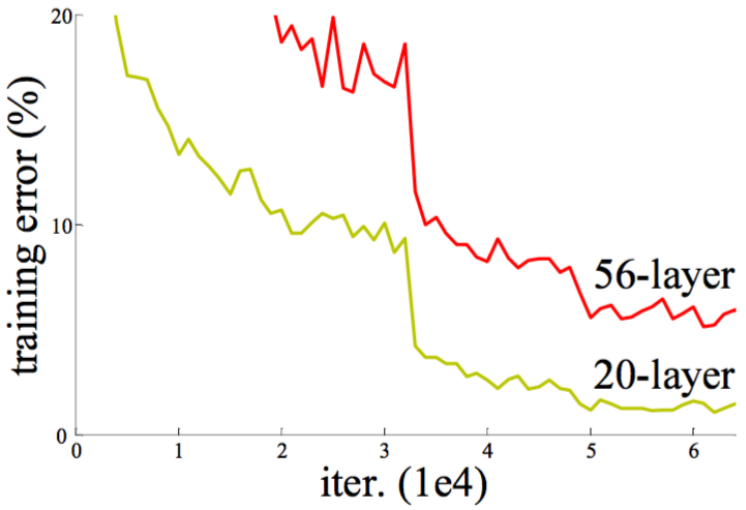
- Deeper networks are more difficult to train
  - Vanishing / exploding signal and gradients
  - Deeper nets lead to worse **train** errors: poor optimization!
- Residual connections sum output of layer with activation of previous layer

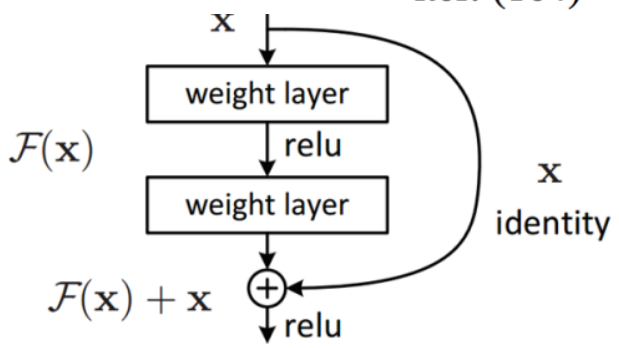
$$x_{l+1} = x_l + F(x_l, W_l)$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

- Acts a "shortcut" for gradient signal to train layers far from loss

$$\frac{\partial L}{\partial x_l} = \frac{\partial L}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial L}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i; W_i)\right)$$

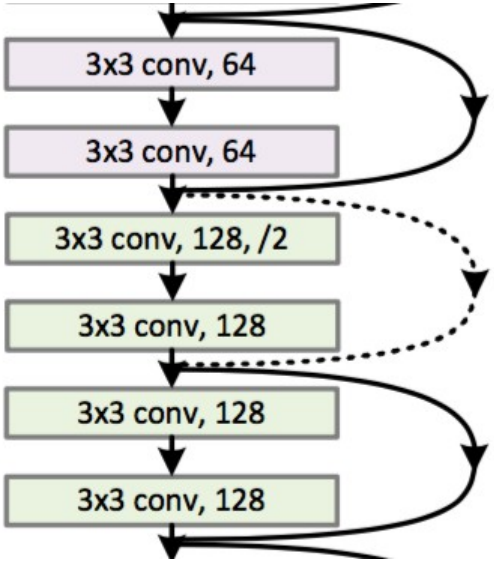


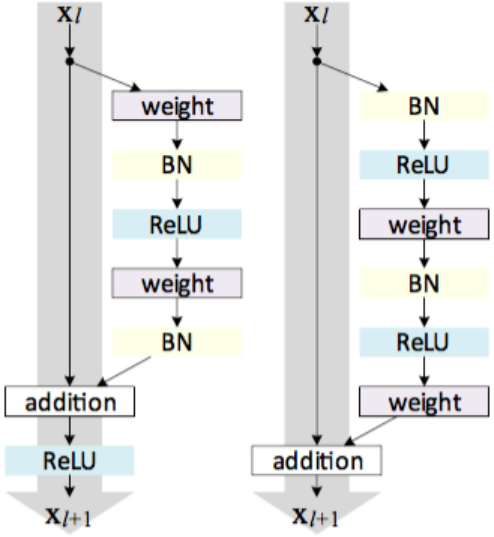


A residual block

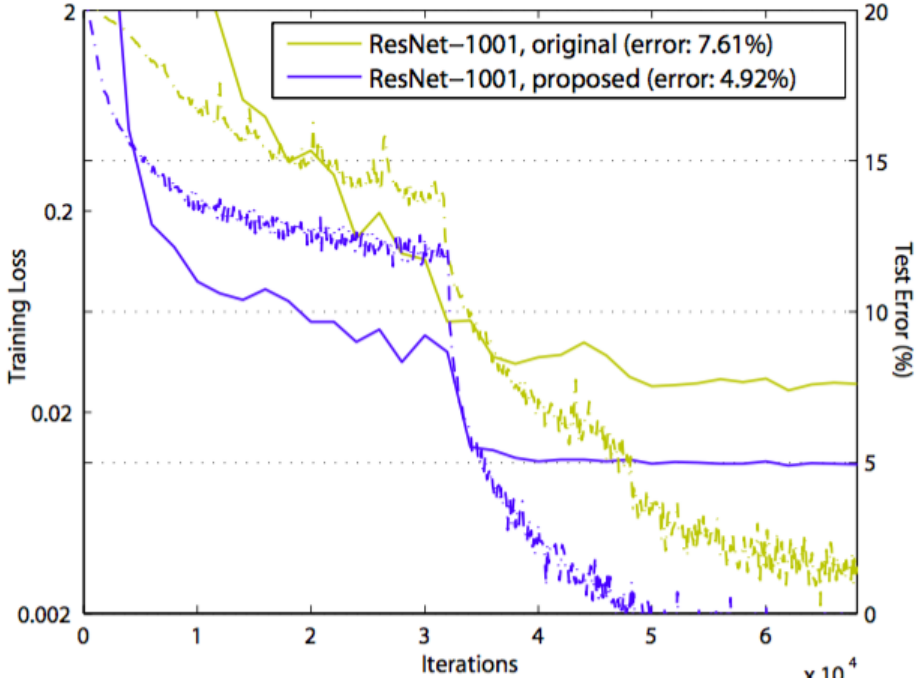
ResNet: Residual CNN Architecture

- Layers with striding: stride residual too
- Layers with increasing nr of feature channels
  - Residual on part of channels
  - Linear projection (1x1 conv) on residual connection
  - Both similarly effective (as is proj. on all residuals)
- More effective to place residual outside the non-linearity
- Successful training of networks with up to 1000 layers





(a) original (b) proposed



DenseNet: Dense layer connection CNN architecture

- ResNet sums output of layer with activation of previous layer

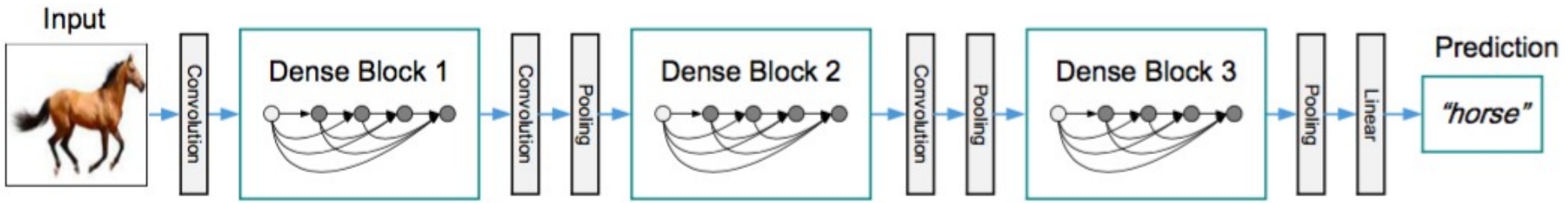
$$x_{l+1} = x_l + F(x_l, W_l)$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i)$$

- DenseNet takes input from all preceding ones, instead of only last one

$$x_L = F([x_l, x_{l+1}, \ldots, x_{L-2}, x_{L-1}], W_L)$$

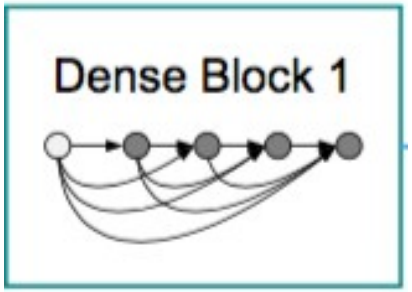
  - Maximally connected DAG at the layer level
  - Features computed in early layers useful for later layers do not need to be "carried along" by intermediate layers
  - Short effective depth to back-propagate error gradient signal

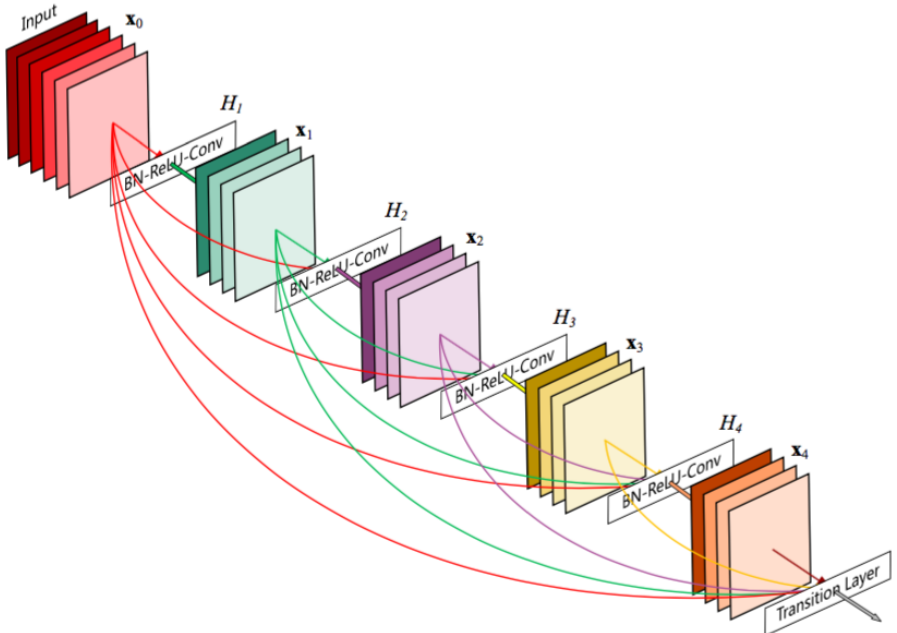


[Huang et al., CVPR'17]

DenseNet: Dense layer connection CNN architecture

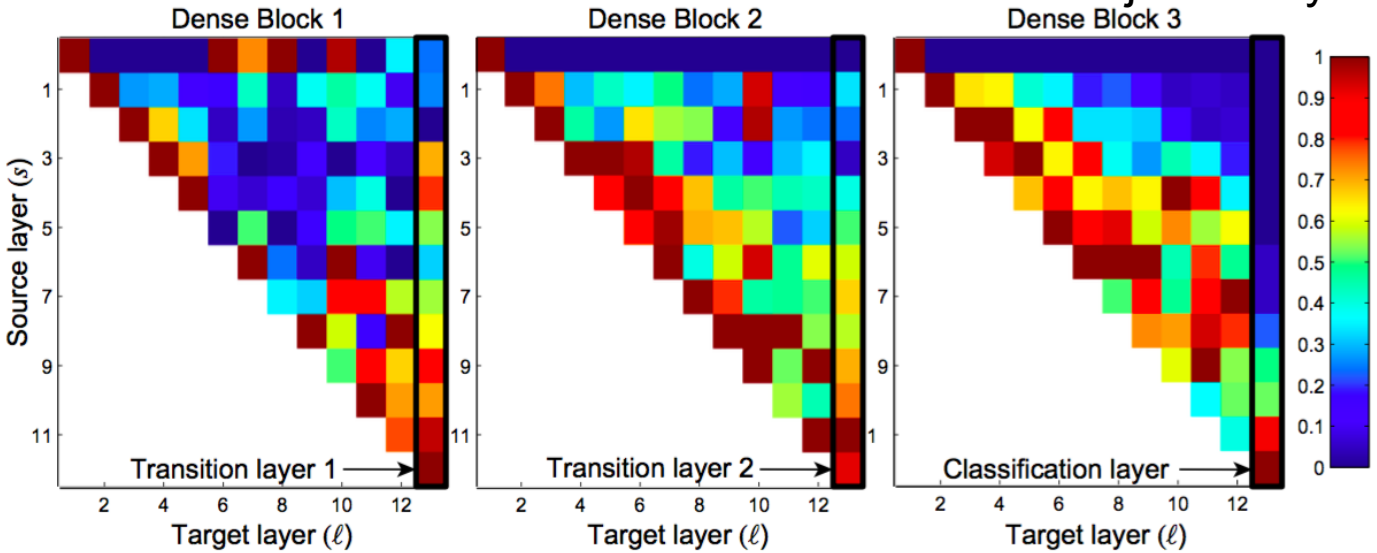
- Every layer computes a fixed nr. of output feature channels
  - Growth rate k, typically between 10 and 50
- Control number of parameters
  - Before 3x3 conv, use 1x1 conv to map input features to 4k features
  - Reduce the nr. of channels at pooling layers by a factor 2 (with 1x1 conv)
- Ultimate DenseNet computes a single feature map at a time
  - Not many more parameters: adds only features in own layer, at most k
  - Prevents parallel computation of features in a single layer



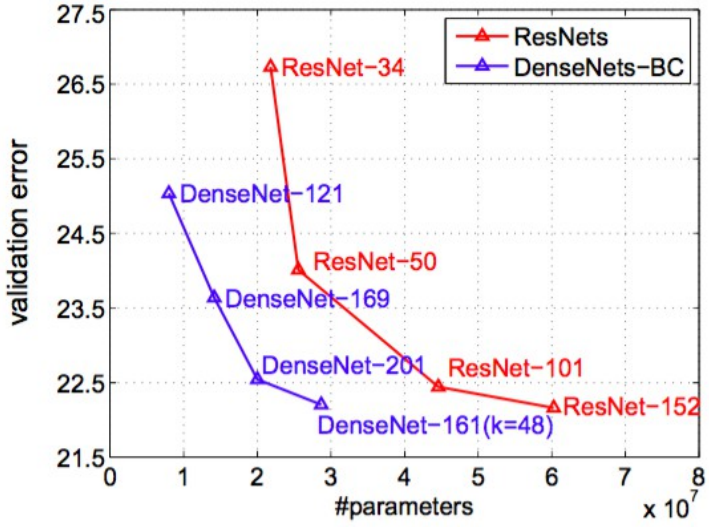


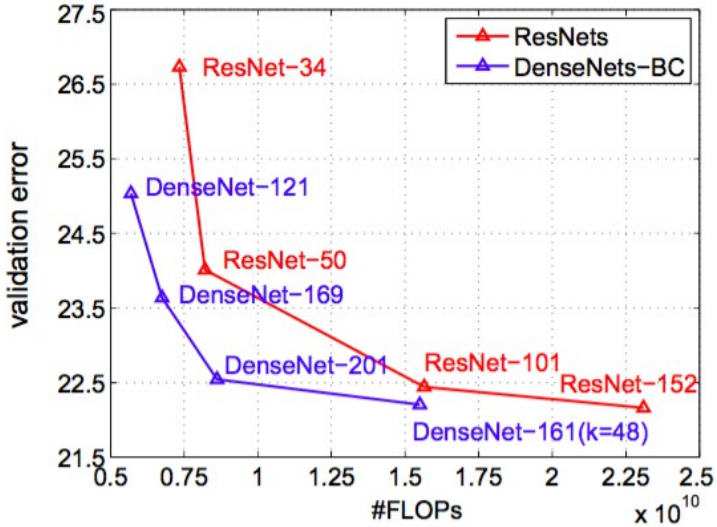
DenseNet: Dense layer connection CNN architecture

- Learned networks contain connections between non-adjacent layers



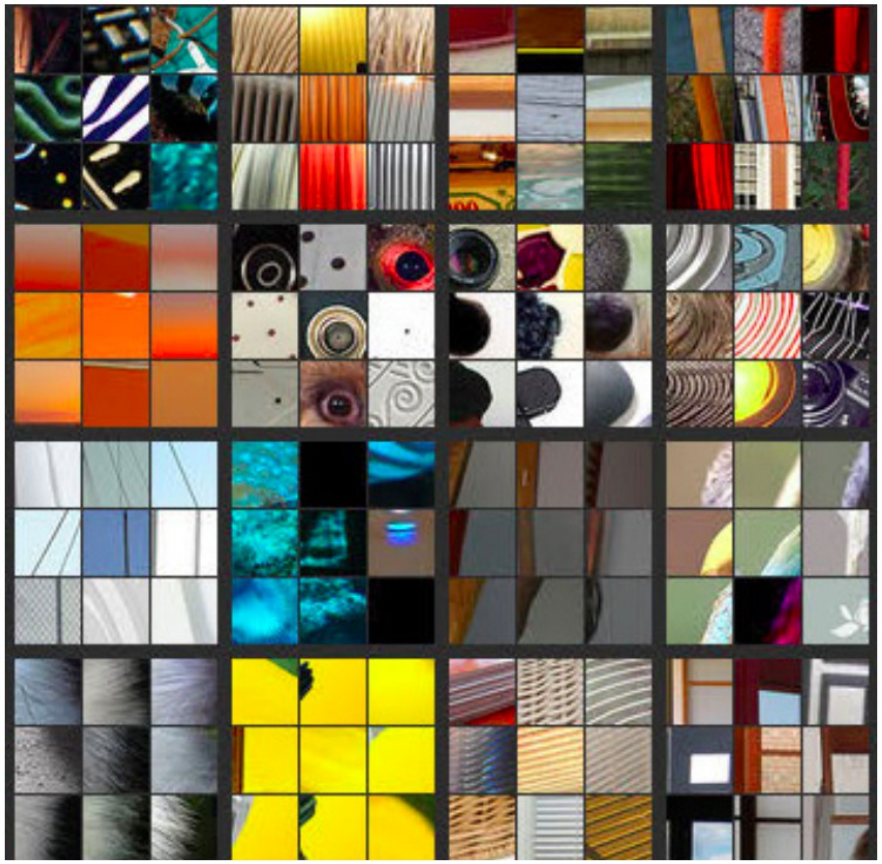
- More compute and memory efficient than residual networks





Understanding convolutional neural network activations

- Patches generating highest response for a selection of convolutional filters,
  - Showing 9 patches per filter
  - Zeiler and Fergus, ECCV 2014
- Layer 1: simple edges and color detectors



- Layer 2: corners, center-surround, ...

Understanding convolutional neural network activations

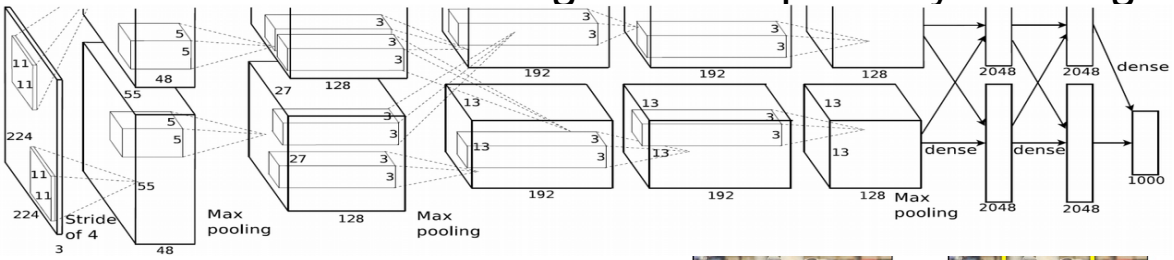
- Layer 3: various object parts

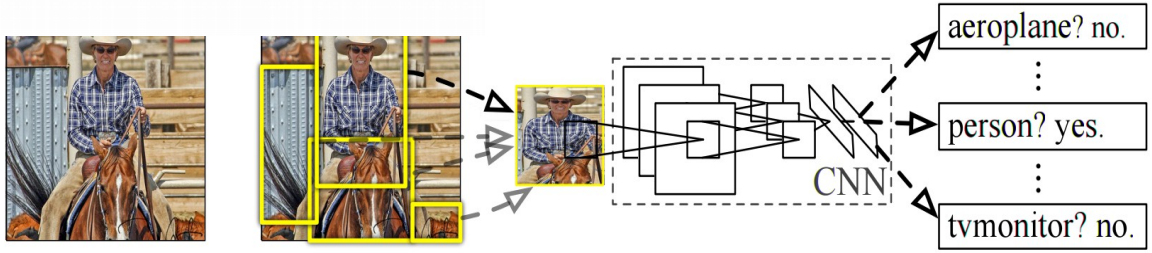
Understanding convolutional neural network activations

- Layer 4+5: selective units for entire objects or large parts of them

Finetuning pre-trained CNNs for other tasks

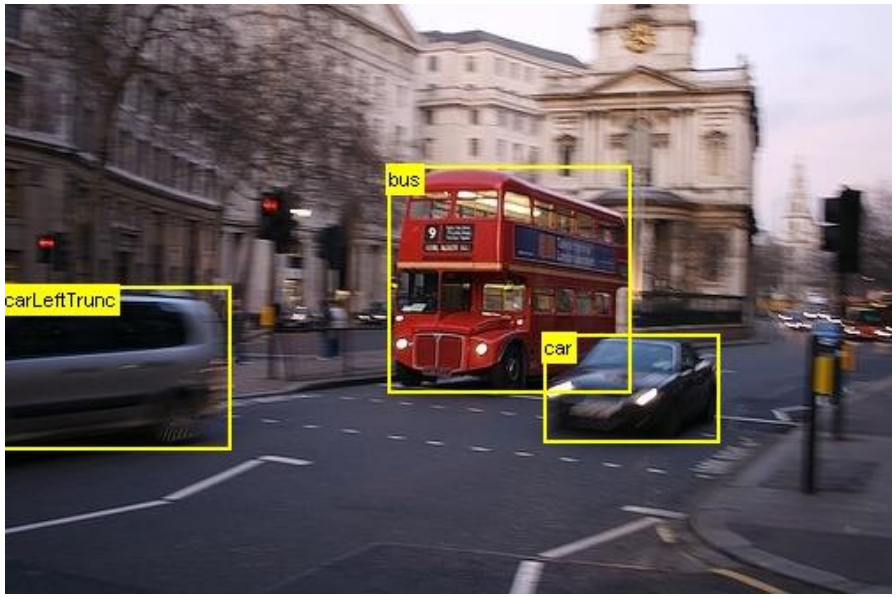
- Early CNN layers extract generic features that seem useful for different tasks
  - Object localization, semantic segmentation, action recognition, etc.
- On some datasets too little training data to learn CNN from scratch
  - For example, only few hundred objects bounding box to learn from
- **Pre-train** AlexNet/VGGnet/ResNet/DenseNet on large scale dataset
  - In practice mostly ImageNet classification: millions of labeled images
  - Also works with noisy image tags from Flickr [Joulin et al. ECCV 2016]
- **Fine-tune** CNN weights for task at hand, possibly with modifed architecture
  - Replace classification layer, add bounding box regression, …
  - Reduced learning rate and possibly freezing early network layers





Convolutional neural networks for other tasks

- **Object category localization**

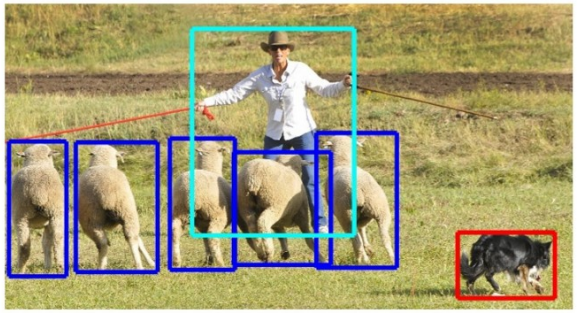


- Semantic segmentation

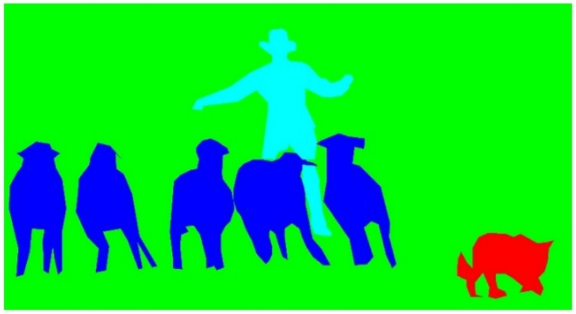
Object category localization with CNNs

- **Task:** given an image report a tight bounding box around every instance of an object category of interest
  - For example detect all people, sheep, dogs, … in an image
- **Problem formulation:** scoring hypothetical object locations
  - Avoid strong overlap between hypothesis with non-maximum suppression
  - Threshold on score to decide on number of objects

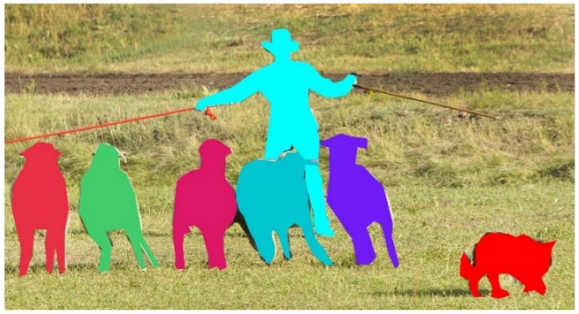
(a) Image classification



(b) Object localization



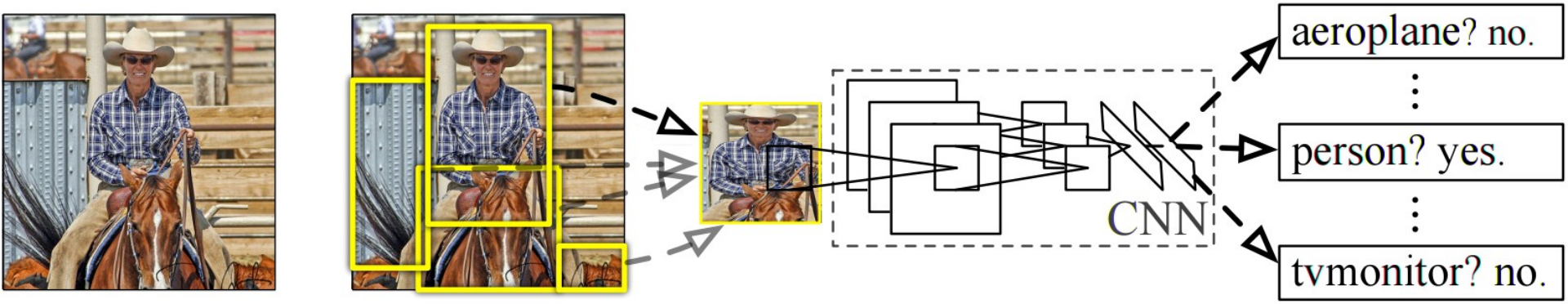
(c) Semantic segmentation



(d) Instance Segmentation

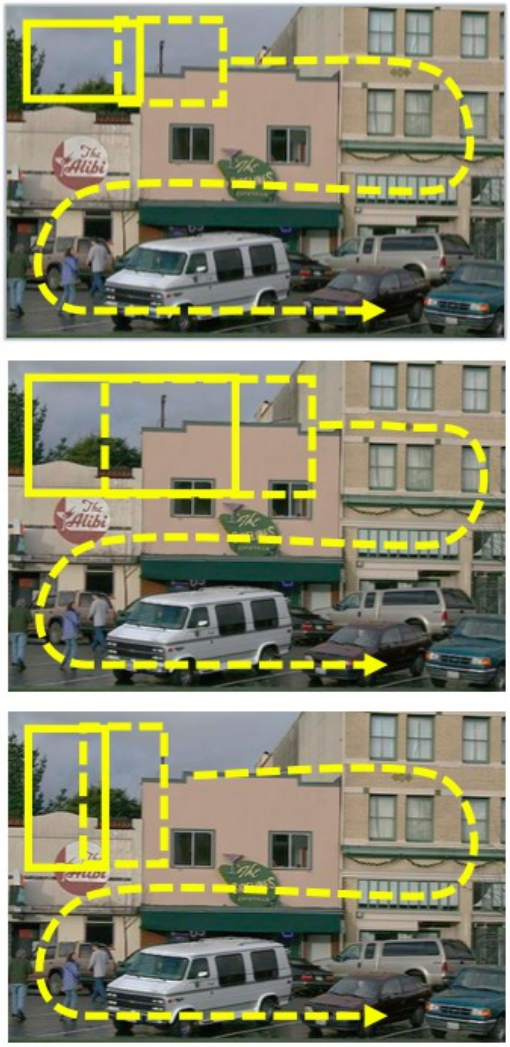
CNNs for object category localization

- Classify each possible dection window as being a tight bounding box for a pedestrian, car, sheep, …
  - Sliding window: translate windows of given size & aspect ratio over image
  - Crop detection window from image, feed to CNN image classifier
- Unreasonably many image regions to consider if applied in naive manner
  - Tremendous cost to evaluate CNN at many positions
- Solutions
  1) Use a smaller set of windows at plausible positions
  2) Share computations across different windows
  3) Do more than classification: bounding box regression

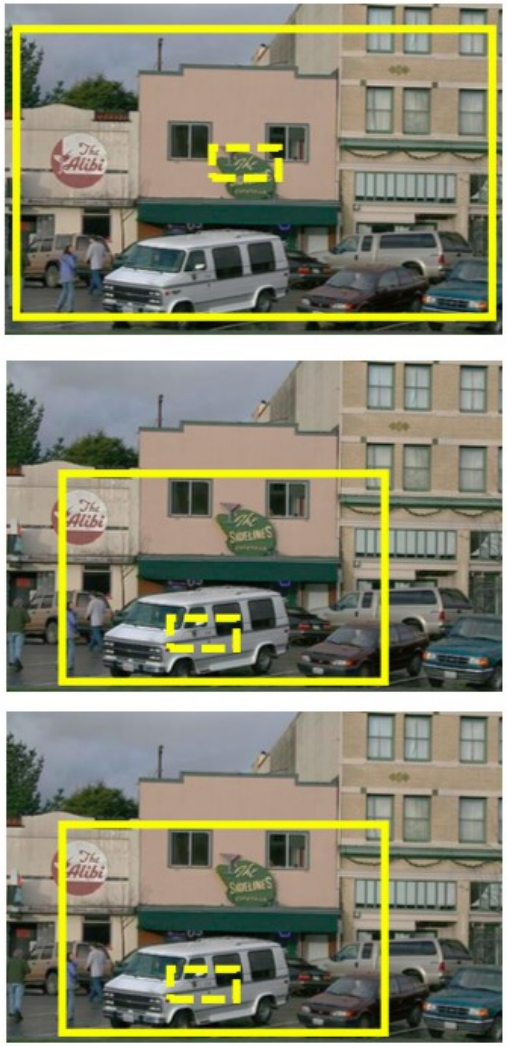


R-CNN, Girshick et al., CVPR 2014

1) How to avoid exhaustive sliding window search



Sliding window
(Viola and Jones 2002;
Felzenszwalb *et al*. 2008, ...)

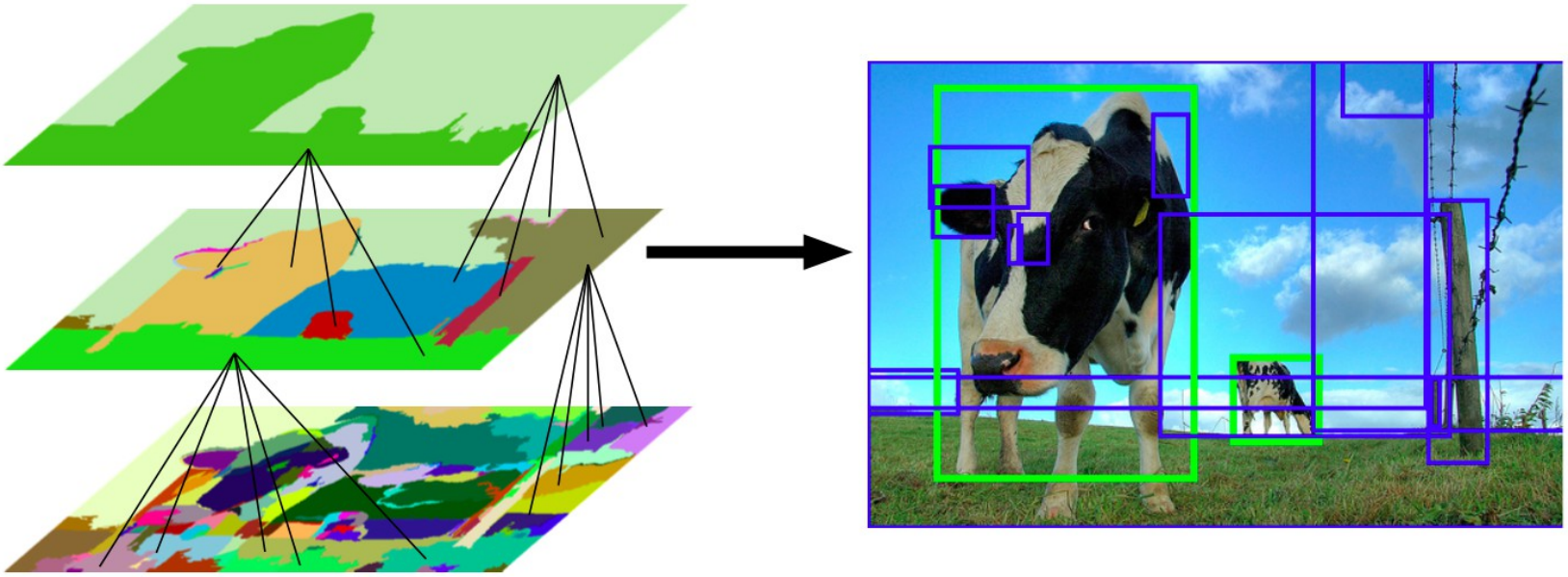


Branch & bound
(Lampert *et al.* 2008;
Lehmann *et al.* 2013)

Selective Search
(Alexe *et al.* 2010;
Sande *et al.* 2011)

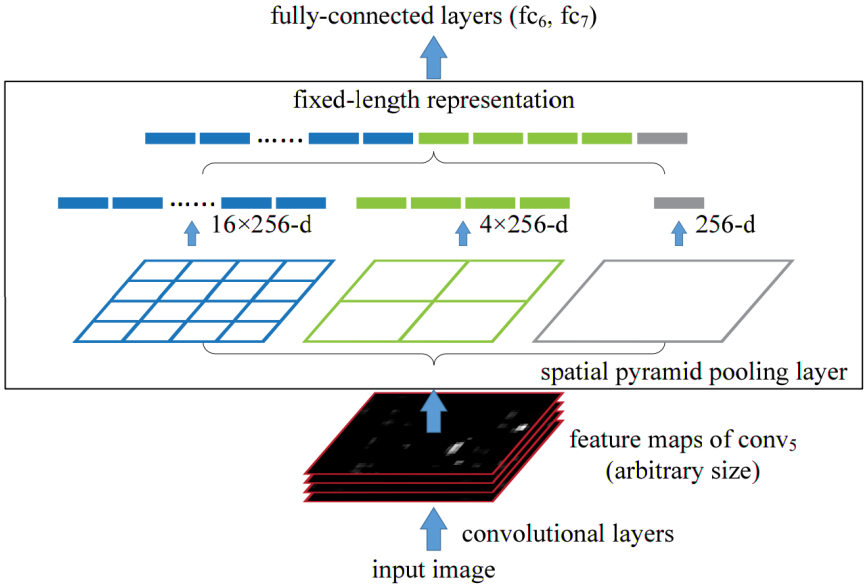
Detection proposal methods

- Many methods exist, some data driven learning based method
  [Alexe et al. 2010, Zitnick & Dollar 2014, Cheng et al. 2014]
- Selective search method [Sande et al. ICCV'11, Uijlings et al. IJCV'13]
  - Unsupervised multi-resolution hierarchical segmentation
  - Detections proposals generated as bounding box of segments
  - 1500 windows per image suffice to cover over 95% of true objects with sufficient accuracy



2) Sharing computations across object detection windows

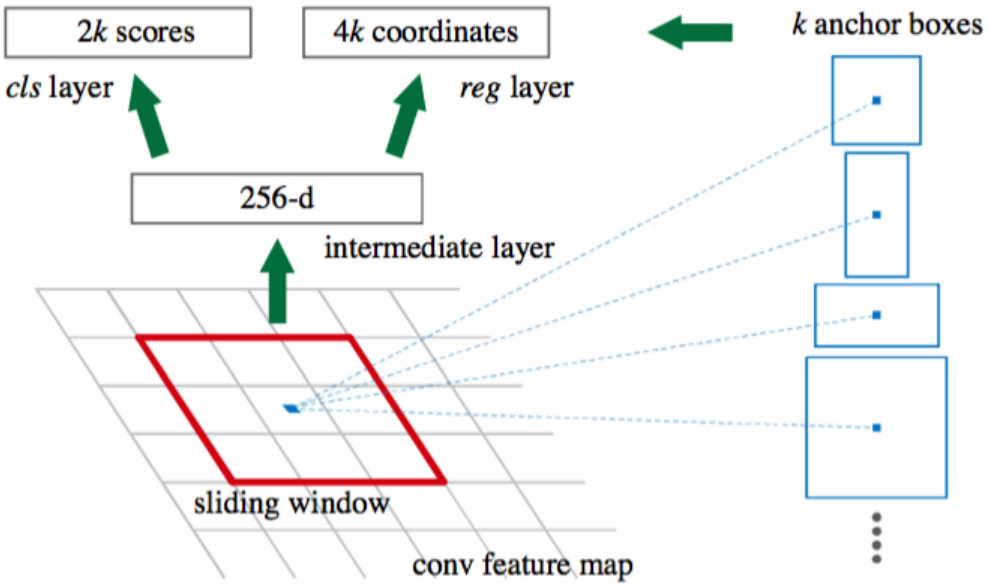
- Naively applying CNN across many cropped or warped windows is wasteful
  - At window overlap convolutions are computed multiple times
- Instead: compute convolutional layers only once across entire image
  - Pool features using max-pooling into fixed-size representation
  - Fully connected layers up to classification computed per window
- Speedup in practice about 2 orders of magnitude



[He et al. ECCV 2014, Girshick ICCV'15]

3) More than classification: Bounding box regression

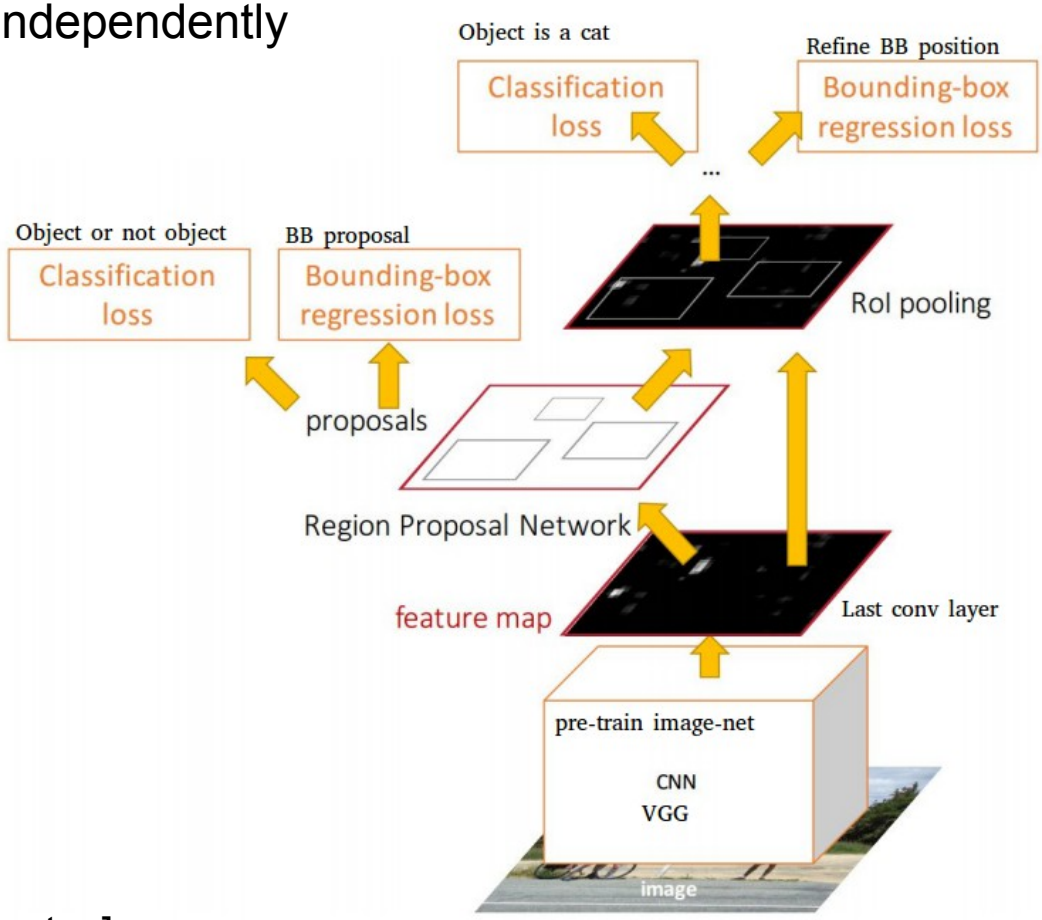
- Classification CNN only extracts a single scalar from every image window
- Additionally: predict the offset of the true object location with respect to the candidate detection window
  - Optionally for several “anchor” boxes



[Ren et al. ICCV’15]

Region of Interest pooling over regressed windows

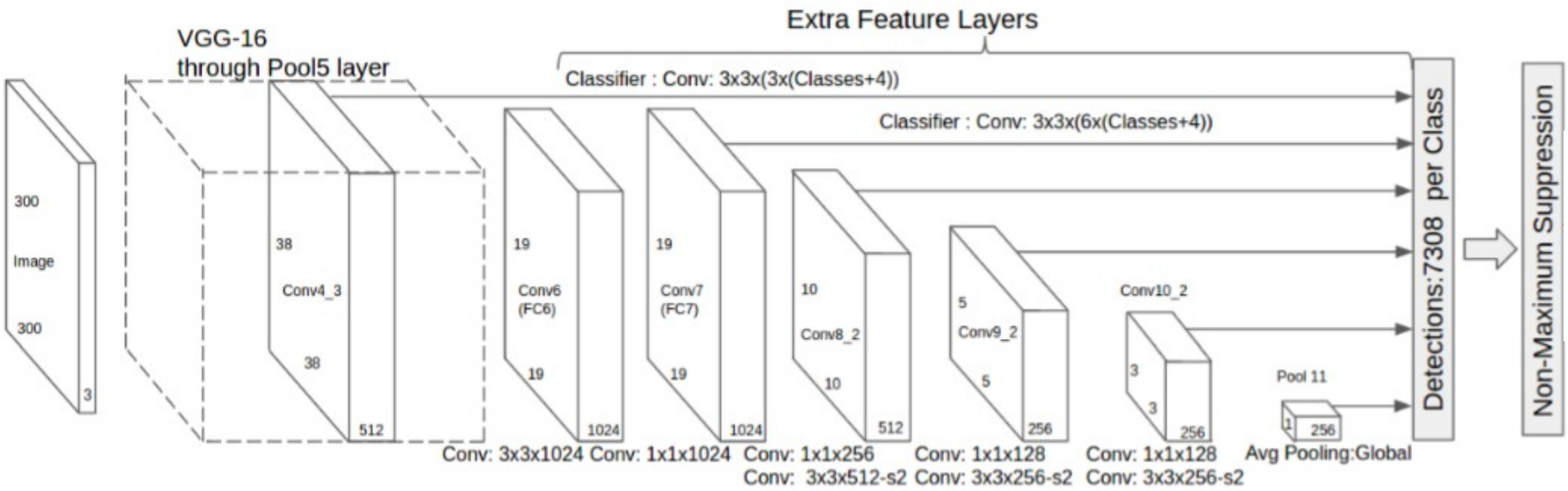
- Region proposal network returns regressed bounding boxes
  - Pool convolutional features across these boxes
  - Classify the regressed box with more CNN layers (regress again)
- RoI's are again processed independently



[Picture from Leonardo Araujo dos Santos]

Single-shot object window regression

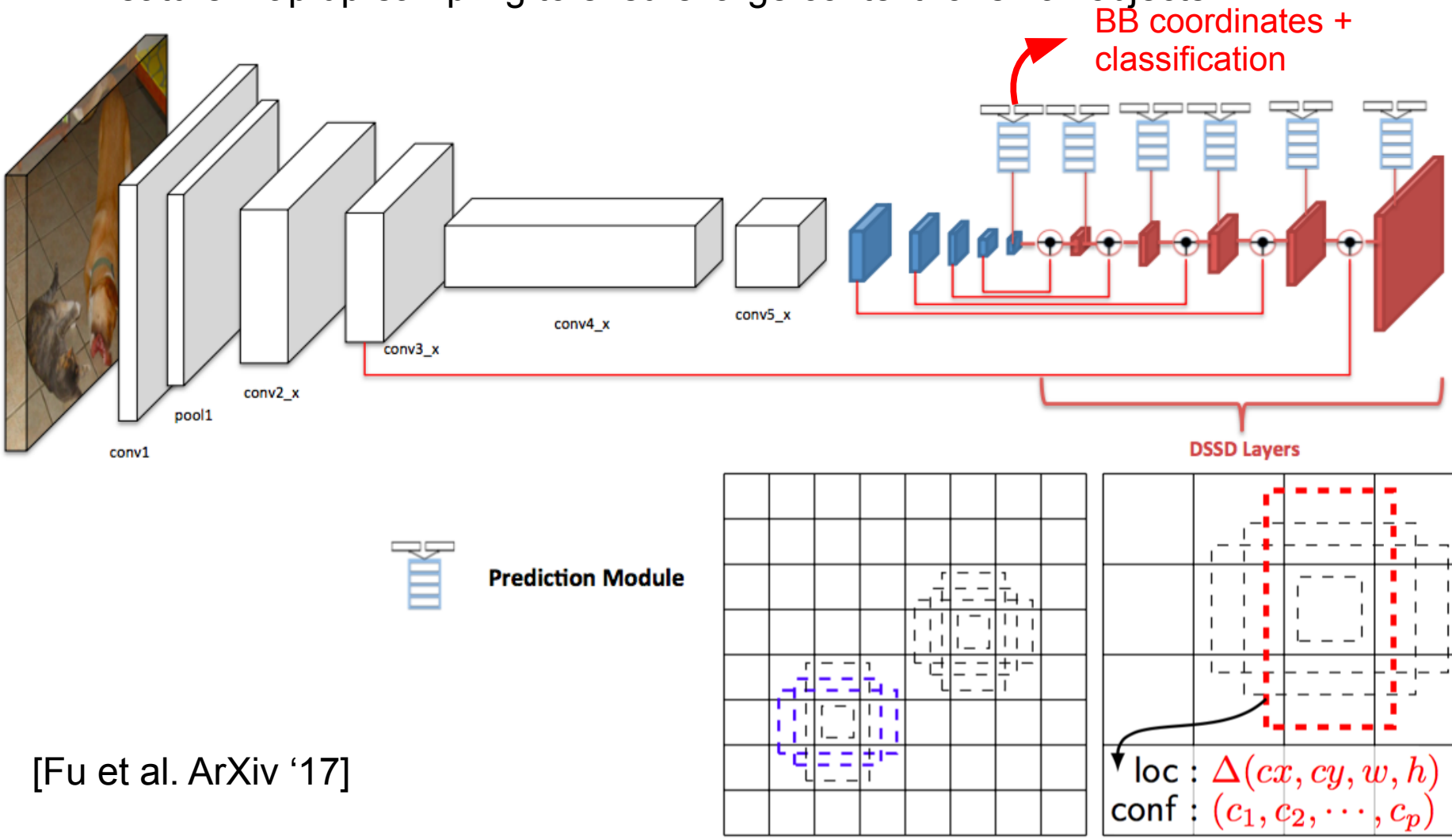
- Region proposal network directly returns regressed bounding boxes
- Detect anchor boxes at different scales and from different network layers
- Using K different “anchor” boxes, each layer of WxH activations outputs KxWxH regressed bounding boxes with corresponding scores
- No further per-box processing after regression: speedup



[Liu et al. ECCV’16]

Single-shot object window regression

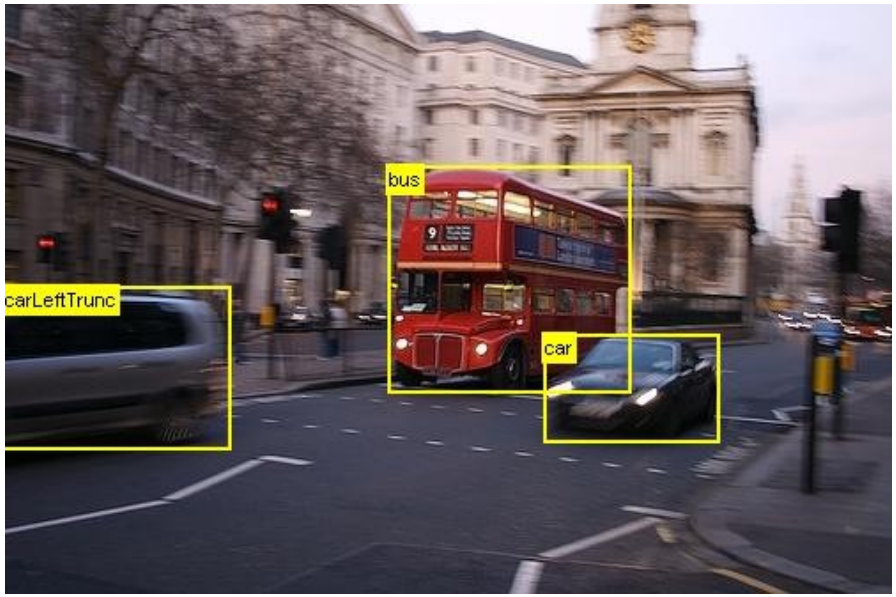
- Small object detected from early high-resolution layers
- Feature map up-sampling to ensure large context for small objects



[Fu et al. ArXiv '17]

Convolutional neural networks for other tasks

- Object category localization

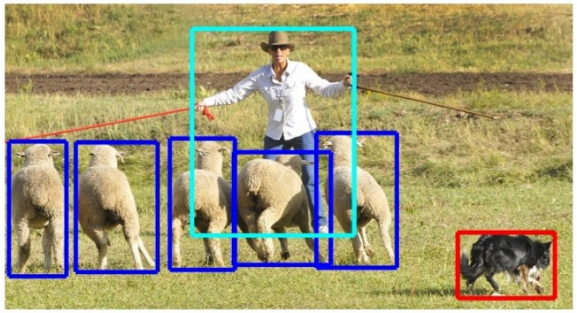


- **Semantic segmentation**

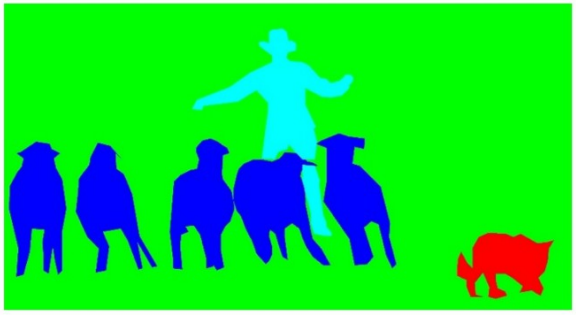
Semantic segmentation with CNNs

- **Task:** given an image assign every pixel to a category
  - For example: background, person, sheep, dog, etc
- **Problem formulation:** classify pixels independently from each other
  - Extract patch centered on pixel of interest, feed to classification CNN
  - Possibly ensure spatial consistency in post-processing step

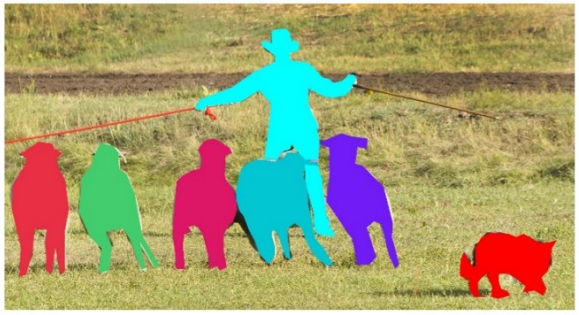
(a) Image classification



(b) Object localization



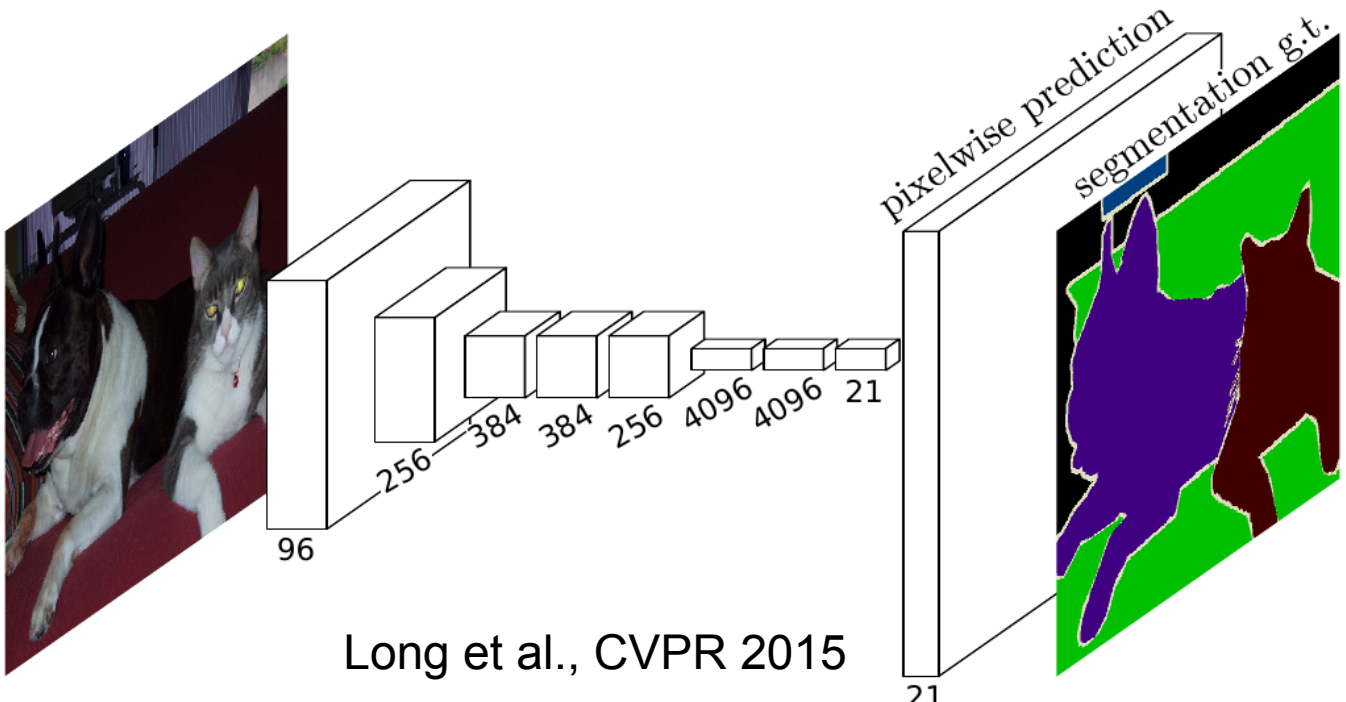
(c) Semantic segmentation



(d) Instance Segmentation

Application to semantic segmentation

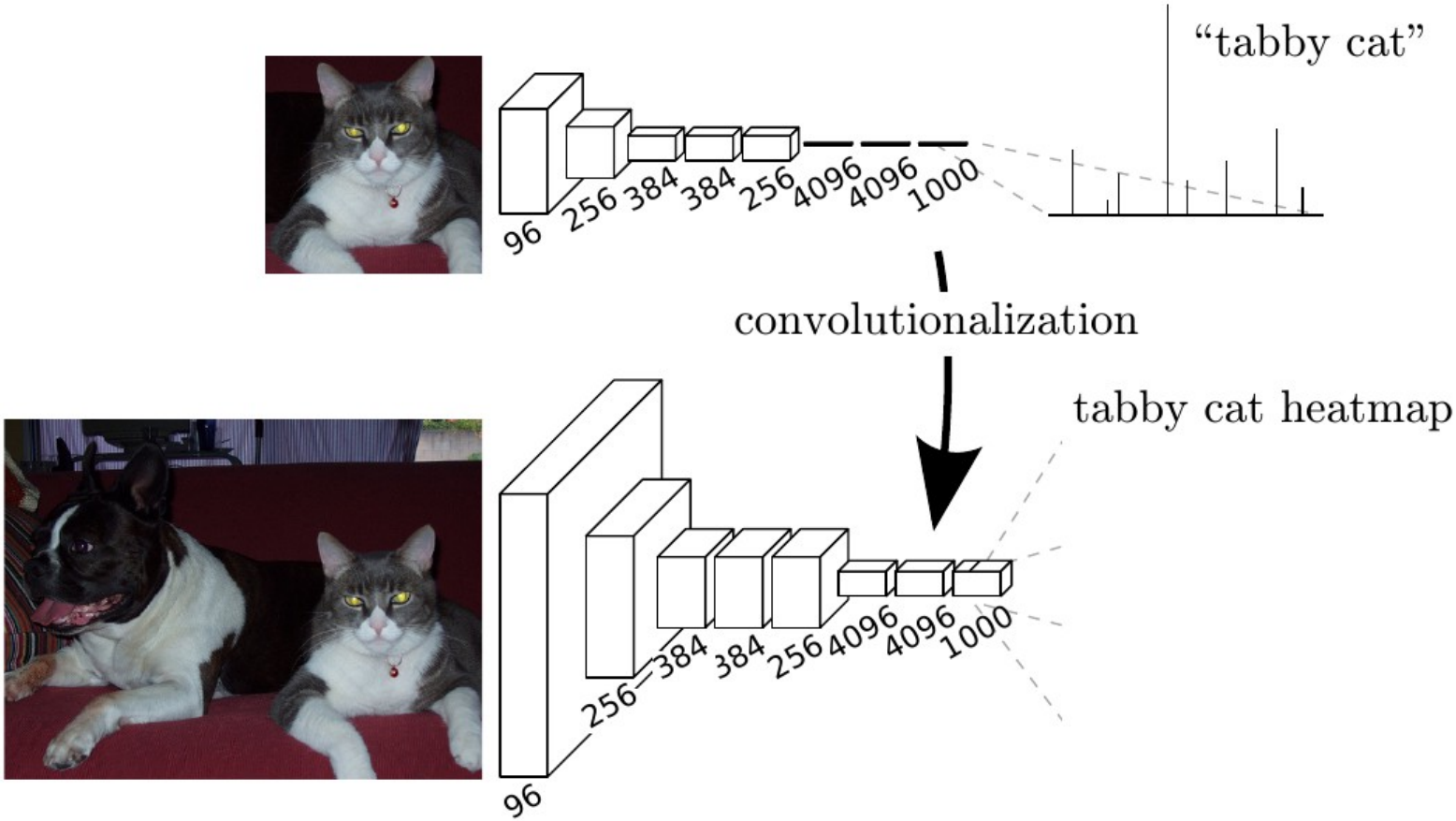
- Assign each pixel to an object or background category
  - Consider running CNN on small image patch to determine its category
  - Train by optimizing per-pixel classification loss
- Want to avoid wasteful computation of convolutional filters
  - Compute convolutional layers once per image
  - Here all local image patches are at the same scale
  - Many more local regions: dense, at every pixel



Long et al., CVPR 2015

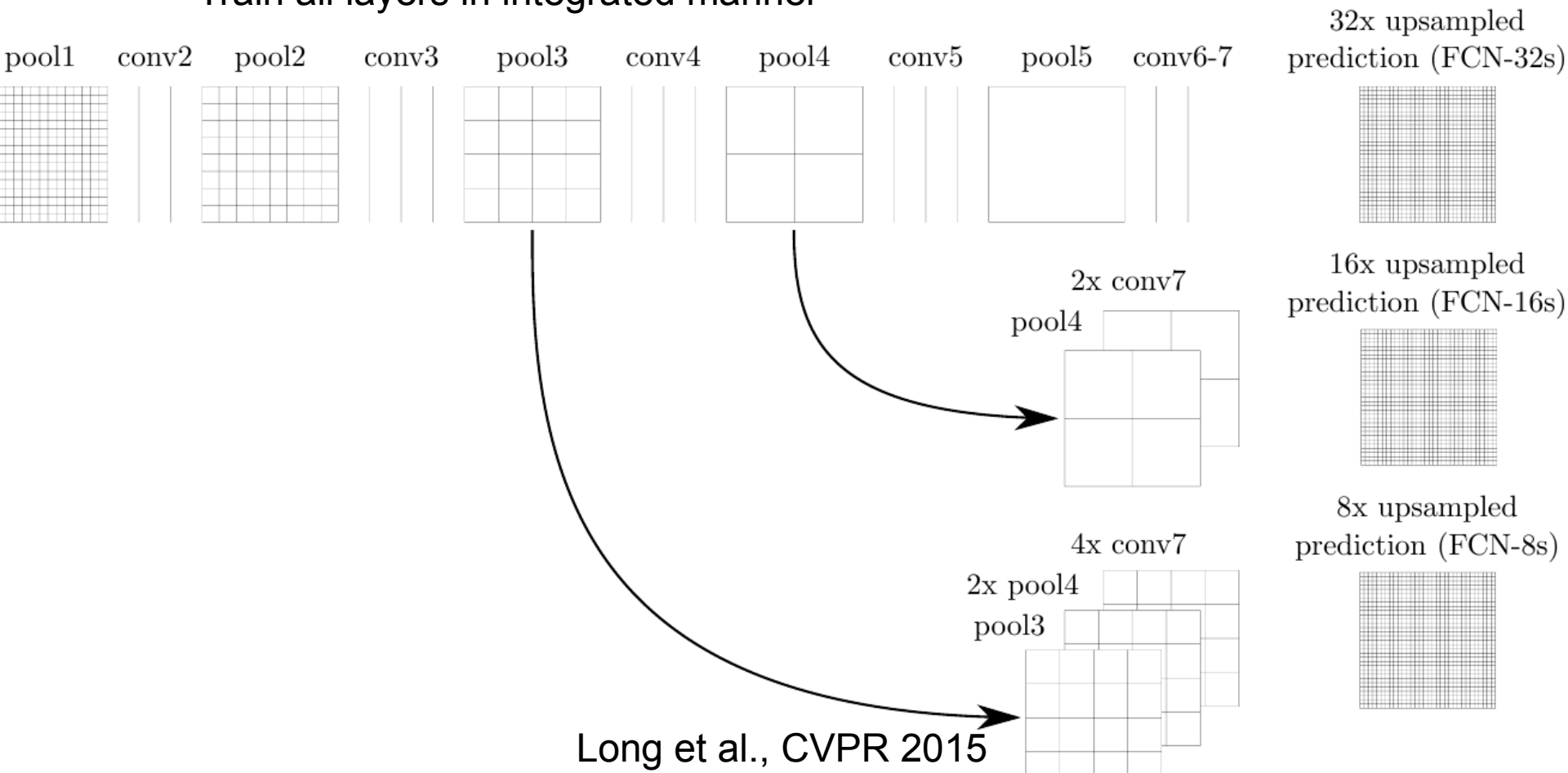
Application to semantic segmentation

- Interpret fully connected layers as 1x1 sized convolutions
  - Function of features in previous layer, but only at own position
  - Still same function is applied across all positions
- Five sub-sampling layers reduce the resolution of output map by factor 32



Application to semantic segmentation

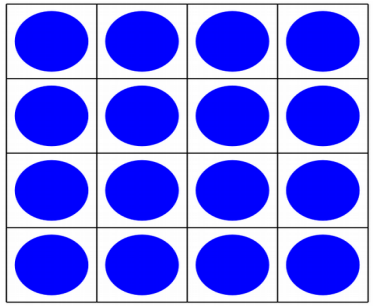
- Up-sampling via bi-linear interpolation gives blurry predictions
- Combine response maps at different resolutions
  - Upsampling of the later and coarser layers, concatenate with finer layers
  - Train all layers in integrated manner

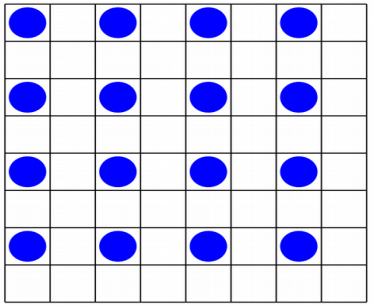


Long et al., CVPR 2015

Upsampling of coarse activation maps

- Simplest form: use bilinear interpolation or nearest neighbor interpolation
  - Note that these can be seen as upsampling by zero-padding, followed by convolution with specific filters, no channel interactions
- Idea can be generalized by learning the convolutional filter
  - No need to hand-pick the interpolation scheme
  - Can include channel interactions, if those turn out be useful



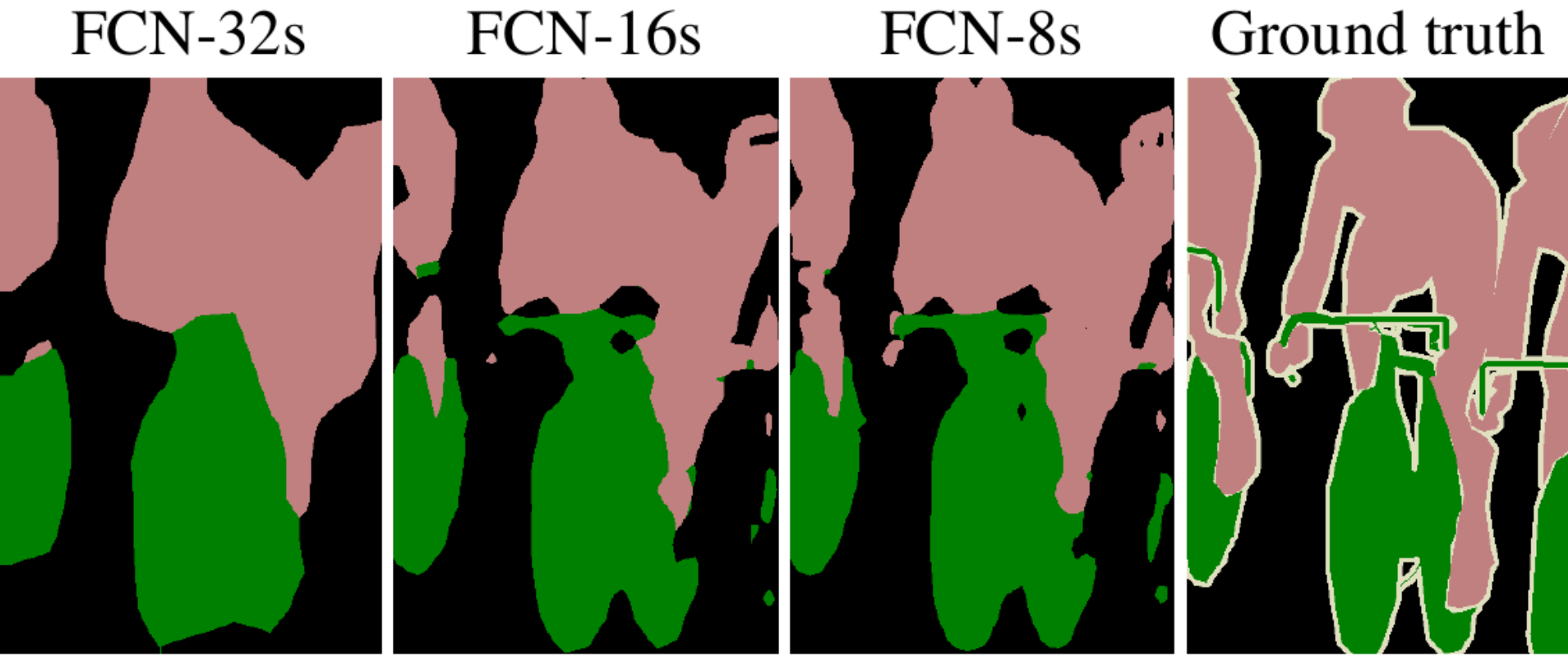


Bi-linear: $\frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ Nearest neighbor: $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

- Resolution-increasing counterpart of strided convolution
  - Similarly, average and max pooling can be written in terms of convolutions [Saxena & Verbeek, NIPS 2016]

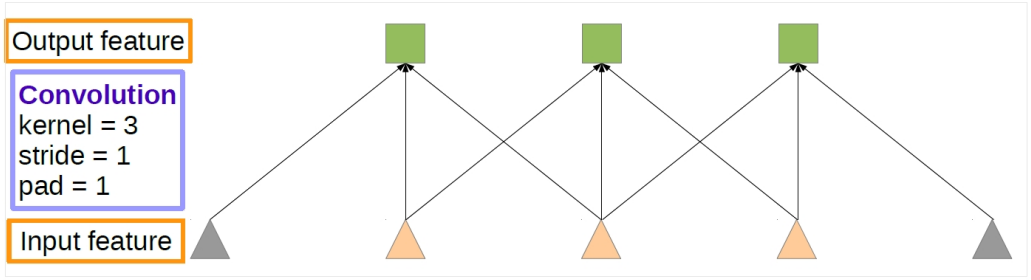
Application to semantic segmentation

- Results obtained using skip-connections from earlier layers
- Detail better preserved when using finer resolutions

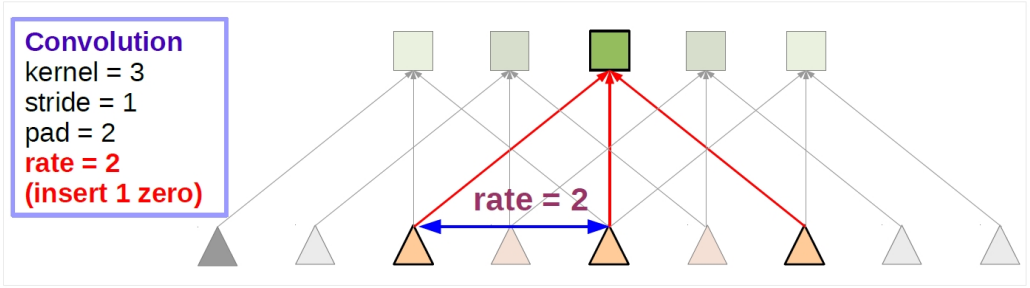


Dilated convolutions

- Filter size and number of parameters are normally coupled
- For fixed filter size, large field of view can be obtained by
  - More layers using a fixed filter: slow growth
  - Down sampling the signal: looses resolution
- Dilated convolution (“filtre à trous”): Large filter with many zeros
  - Large field of view without loosing resolution



(a) Sparse feature extraction



(b) Dense feature extraction

Dilated convolutions

- Decoupling field-of-view and the number of parameters in a filter

  [Yu & Koltun, ICLR '16]

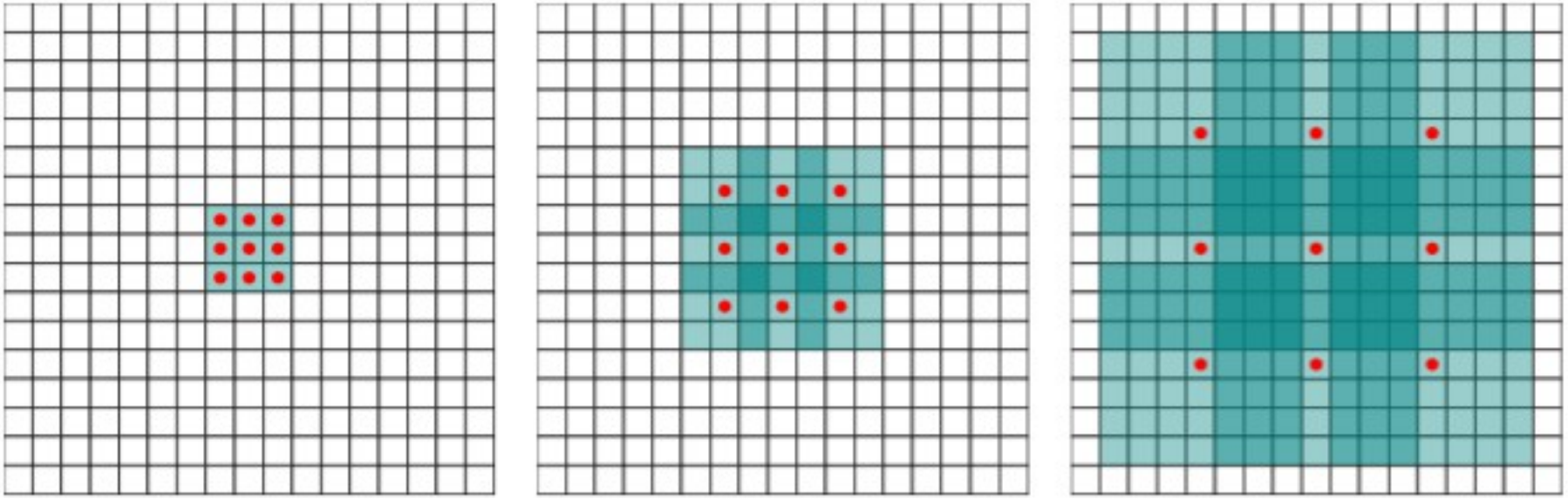
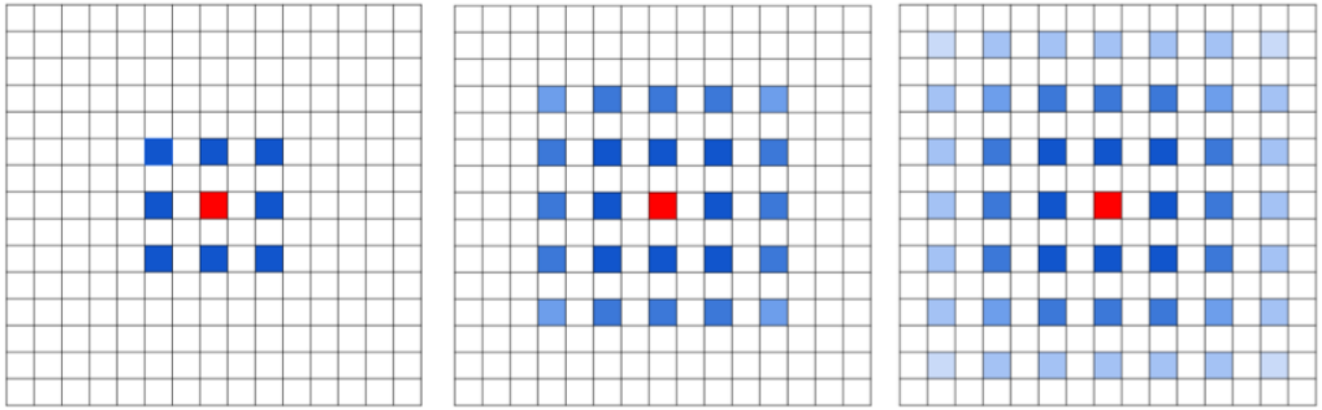


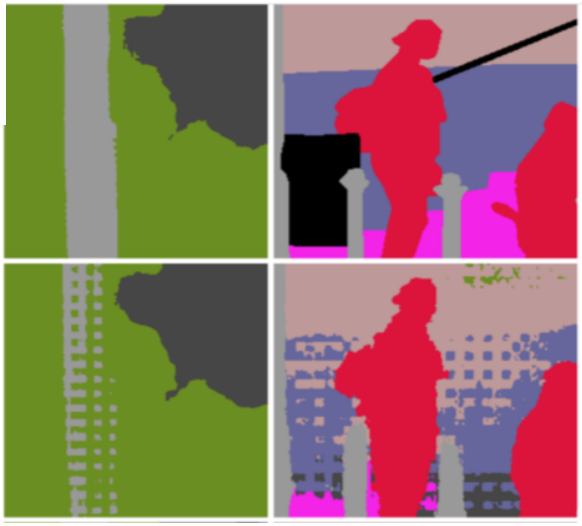
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element 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.

Dilated convolutions

- Similar to strided convolutions, but keeping full resolution in result
  - Can result in aliasing effect due to subsampling of high resolution features
- High-resolution layers are memory intensive
  - 4x more activations as compared to each factor 2 downsampling
  - Limits the number of feature channels that can be used

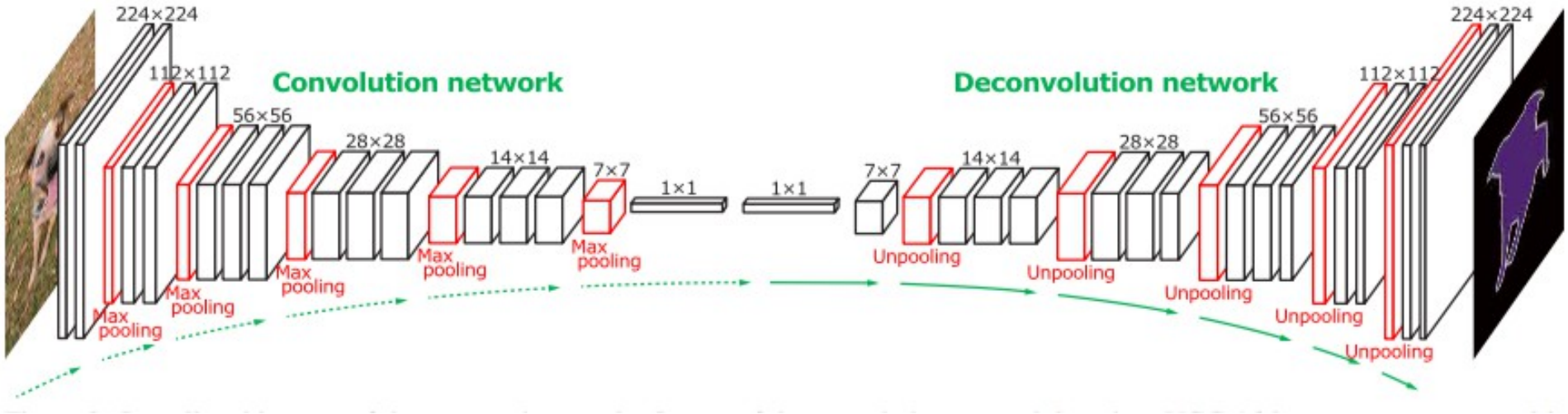


Receptive field of repeated 2-dilated convolutional layers

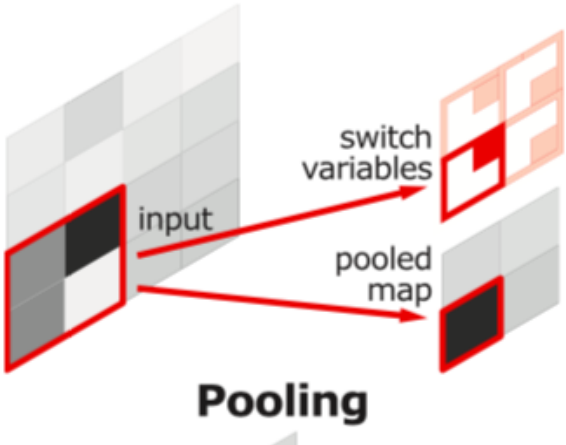


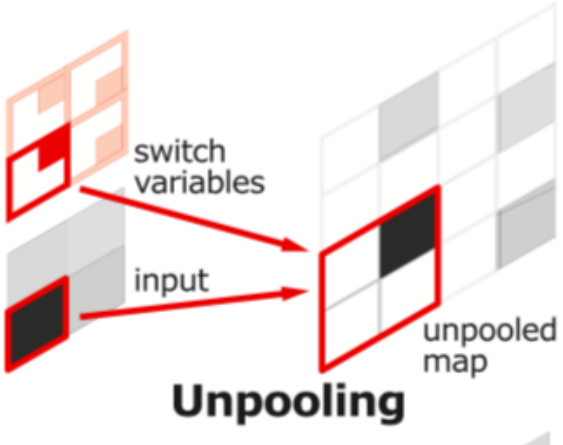
Convolution-Deconvolution architecture

- Use “un-pooling” operation to up-sample the signal, using max-pool locations



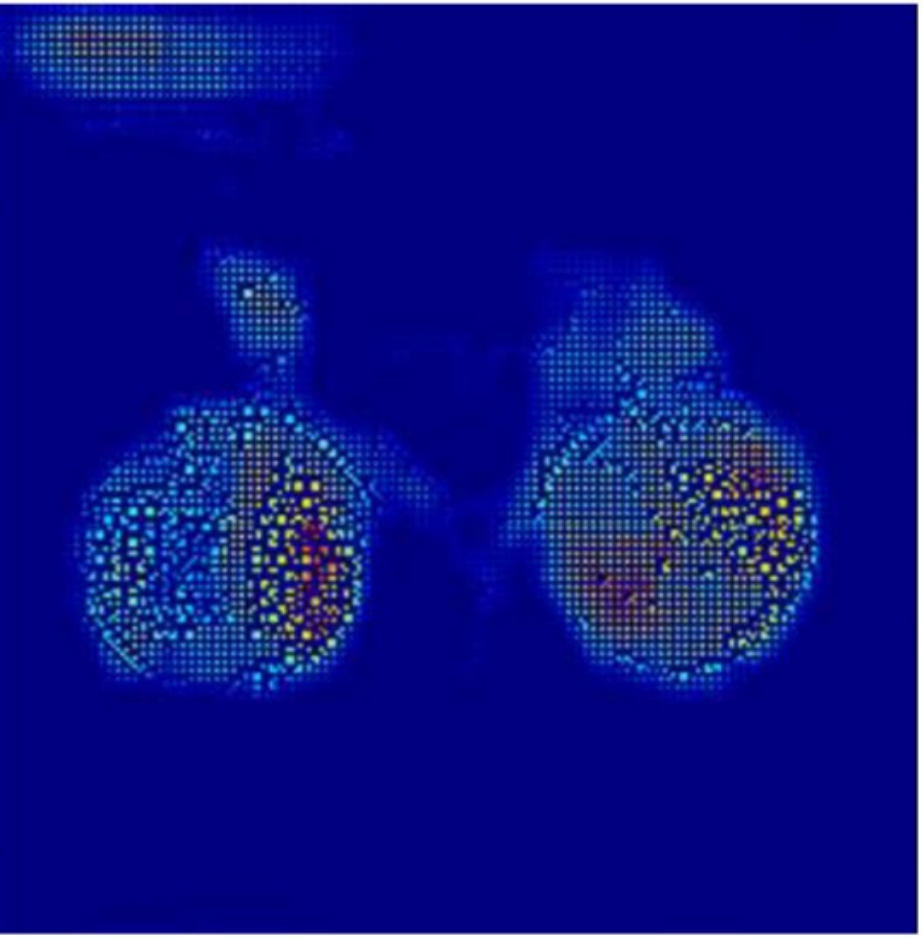
Noh et al., ICCV 2015



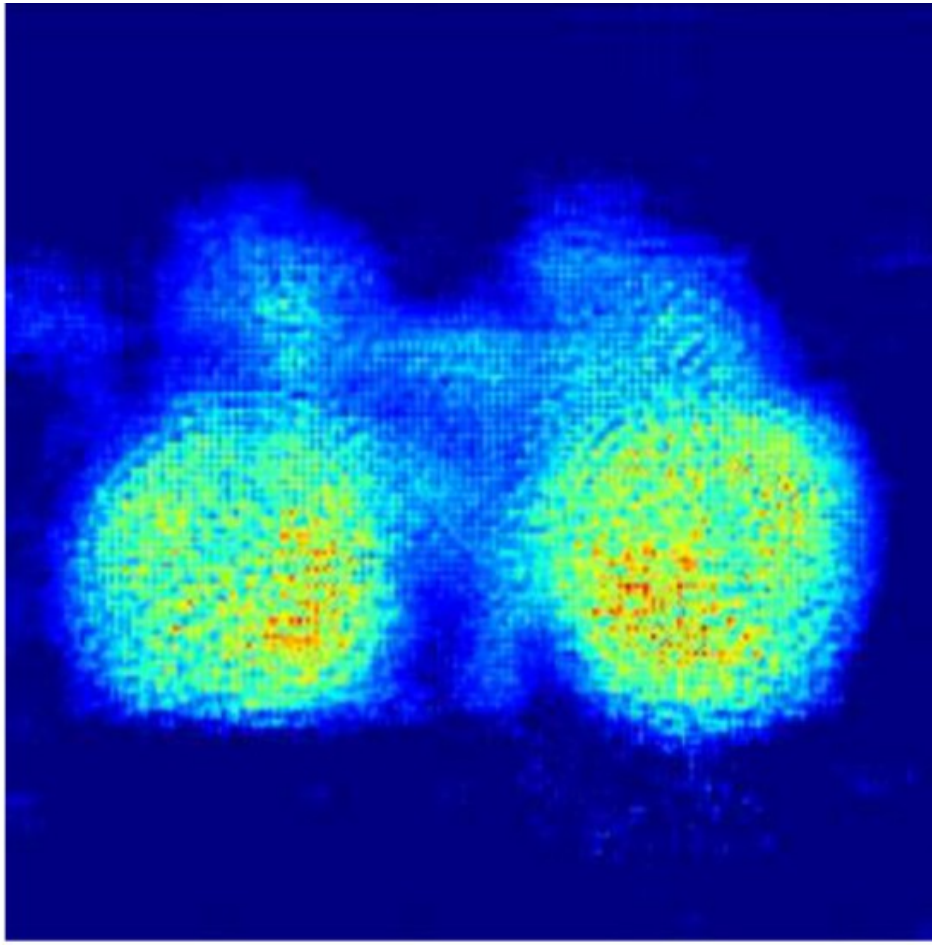


Convolution-Deconvolution architecture

- Gridding effects due to high frequency patters induced by un-pooling
  - Learned convolutions do not suppress these artifacts



Output of un-pooling

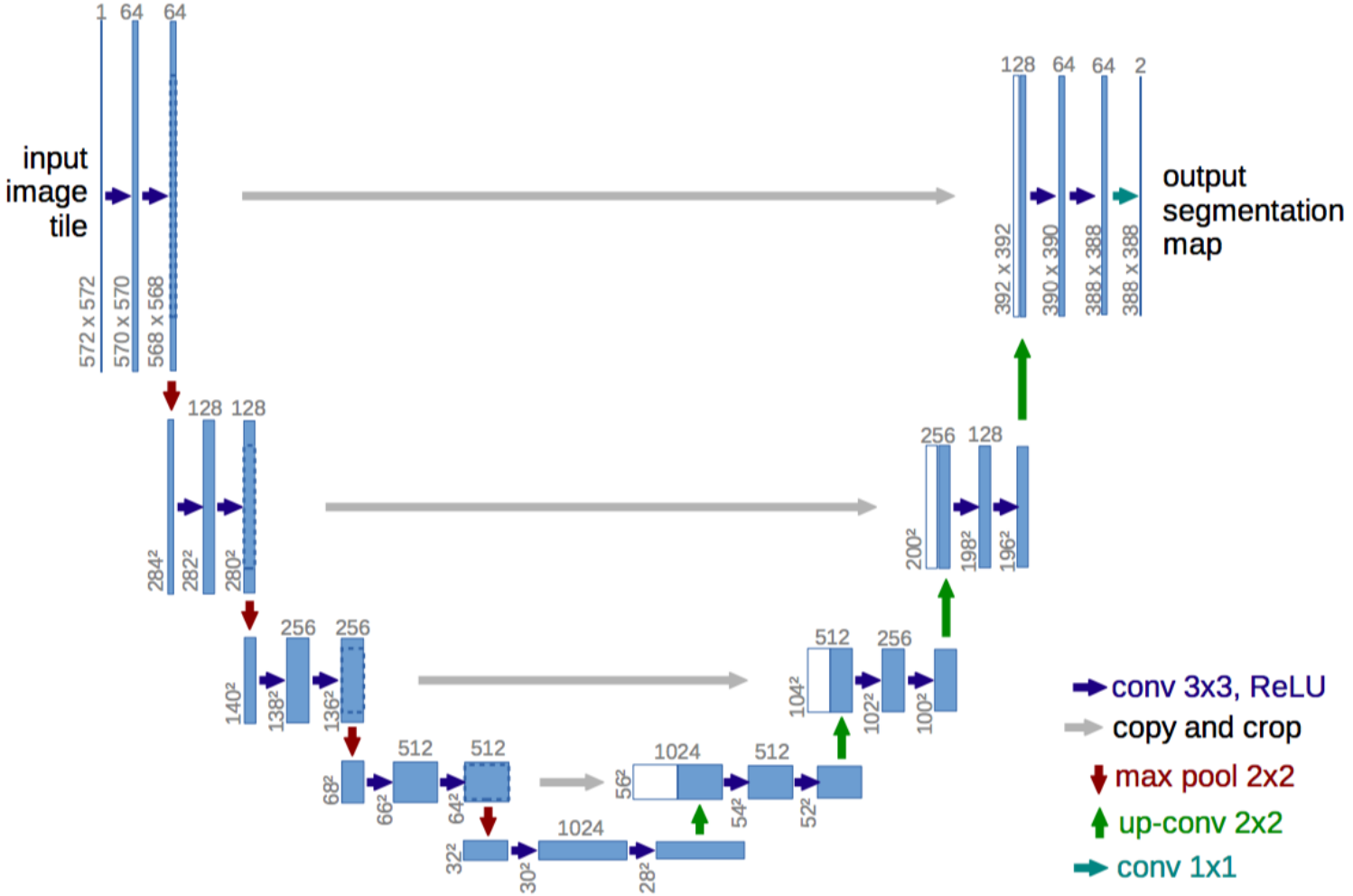


After a couple of conv layers

U-net architecture

- Combines ideas of skip connections and conv-deconv architecture
  - Skip connections to maintain high-resolution signal
  - Progressive upsampling from coarse to fine

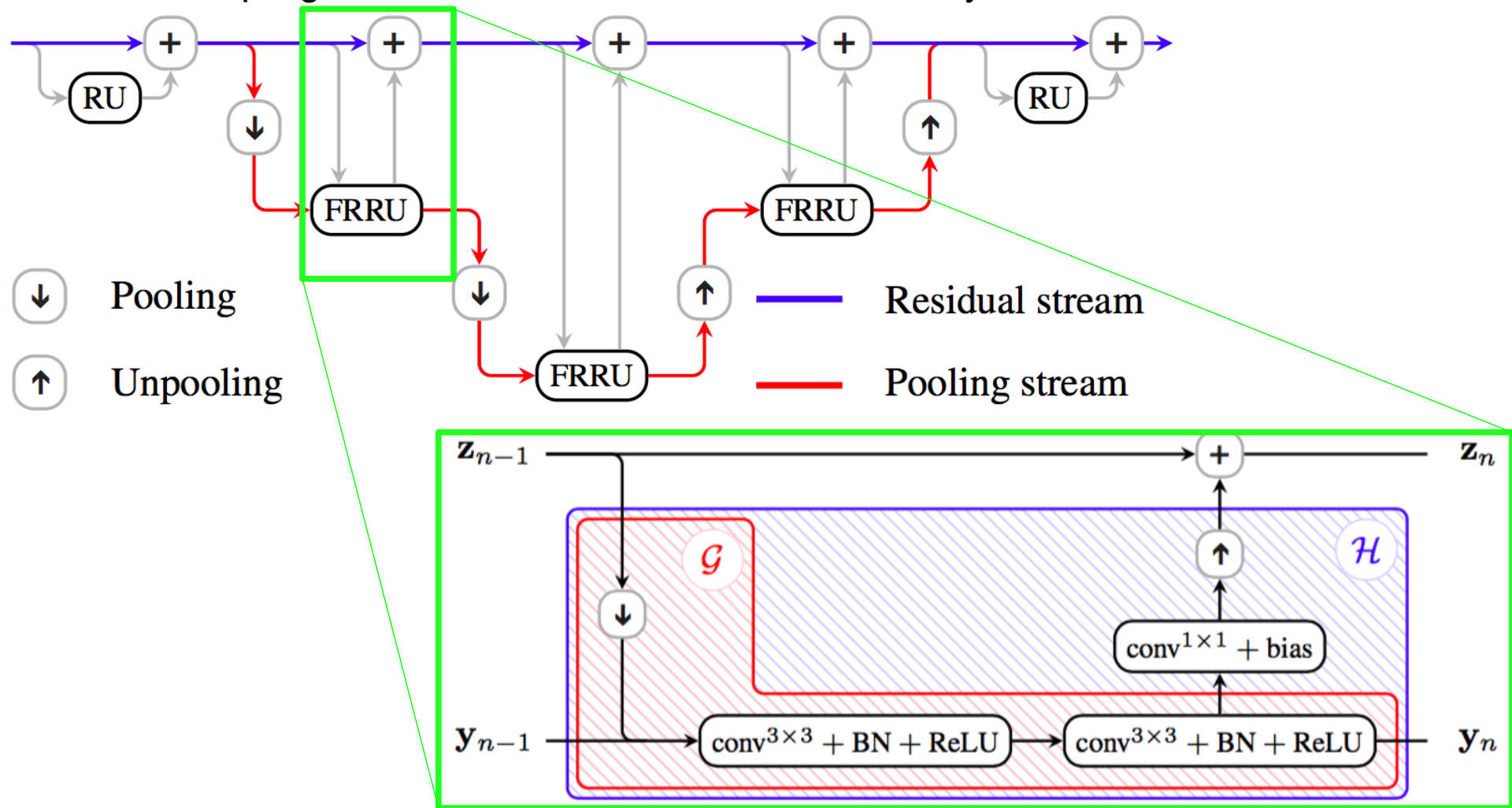
[Ronneberger et al. 2015]



Full-Resolution Residual Networks

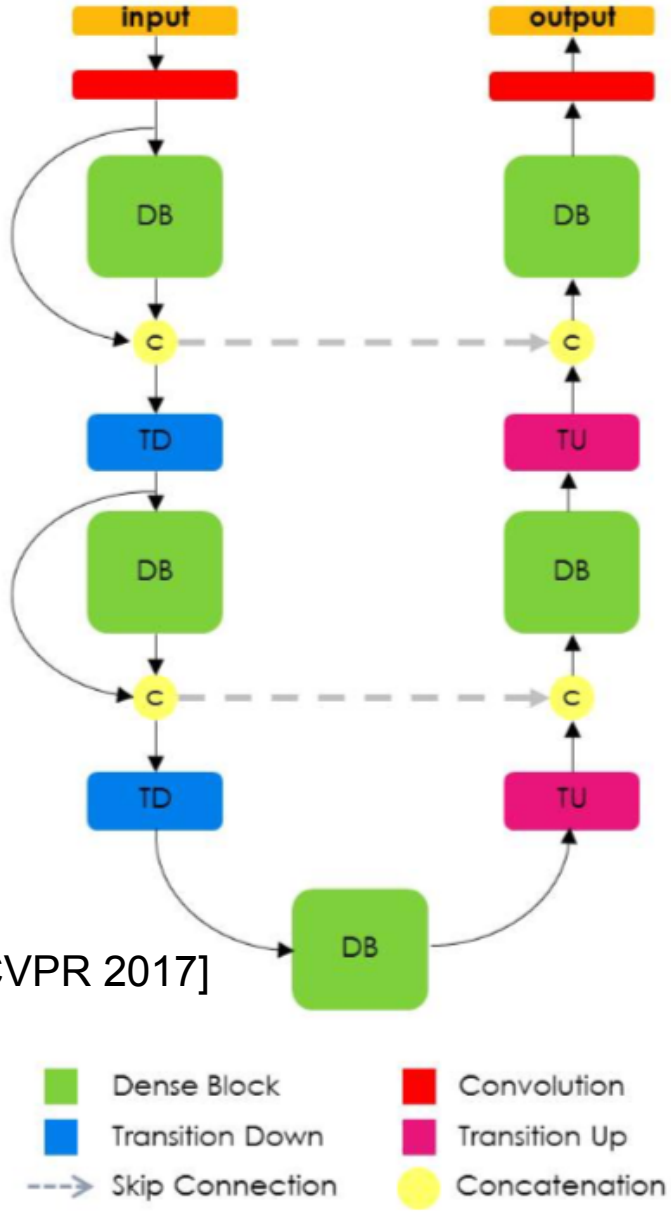
- Combine a full resolution stream with another conv-deconv stream
  - Residual connections in full res stream
  - Coupling full res and low-res streams in each layer

[Pohlen et al., CVPR 2017]



The One Hundred Layers Tiramisu

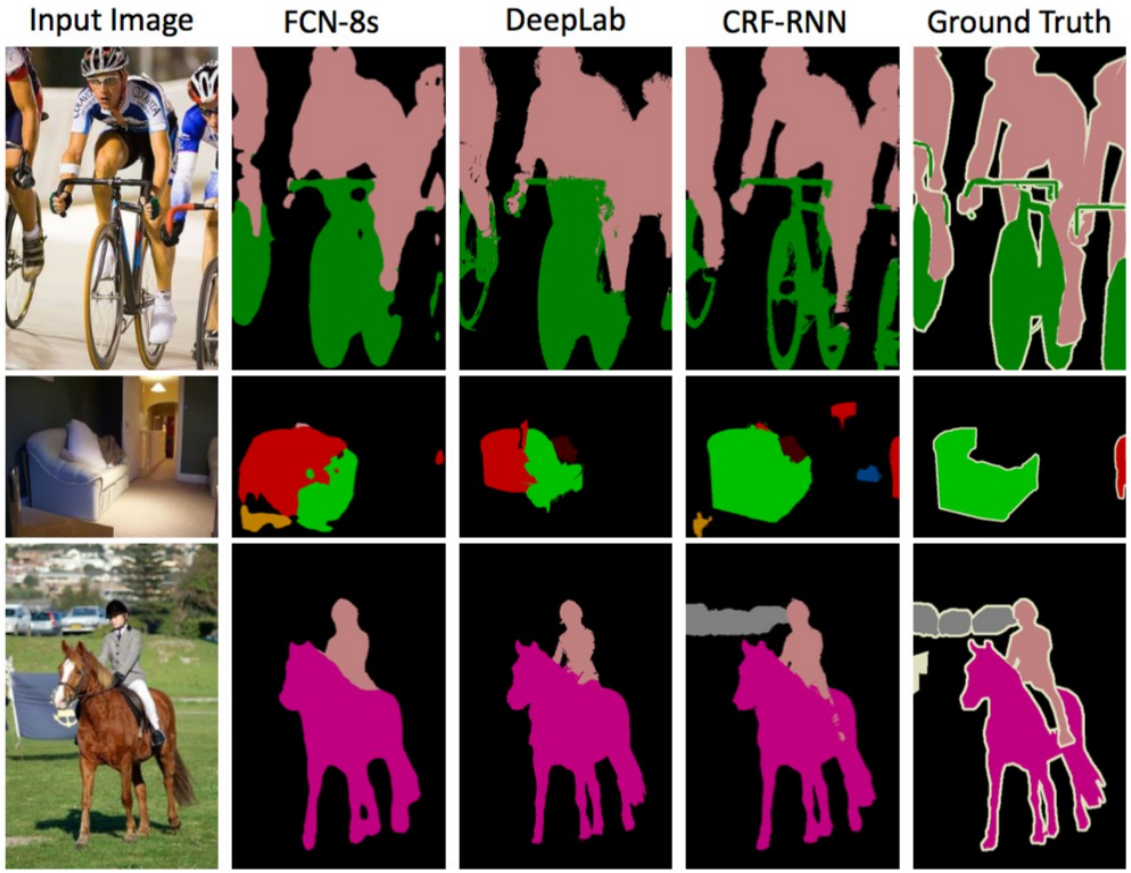
- Similar to U-net architecture
- Builds on densely connected blocks
- Concatenates features in downsampling
- No up concatenation to avoid explosion of number of input feature maps
- Semantic segmentation network with 103 layers trained from scratch (i.e. no ImageNet pre-training)



[Jegou et al., CVPR 2017]

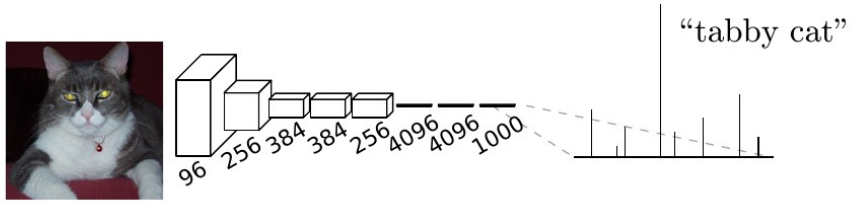
Semantic segmentation: further improvements

- Beyond independent prediction of pixel labels
- Conditional random fields (CRF): encourage nearby and similar pixels to take the same label value
- Efficient inference for fully connected CRFs (all pixel pairs are connected [Krahenbuhl & Koltun, NIPS'11]
- Integrate CRF model within CNN training [Zheng et al., ICCV'15]



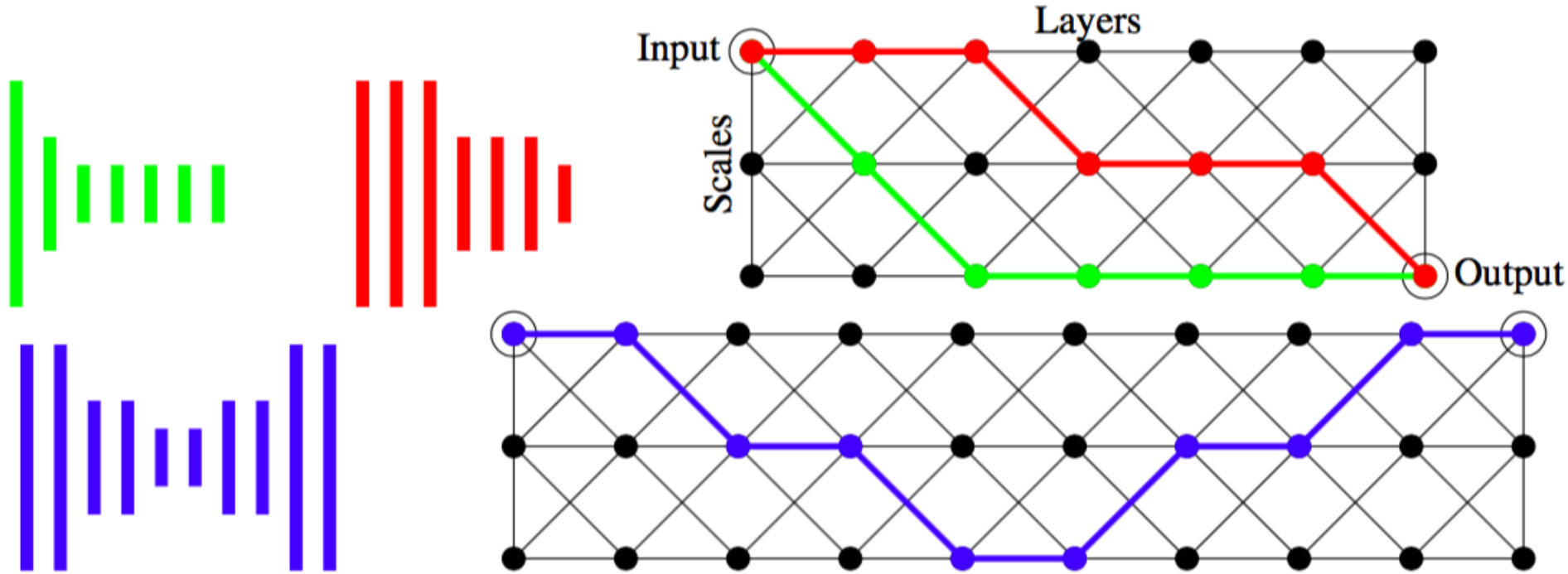
Scale, size and resolution in convolutional networks

- Classification CNN goes from full-res input to 1x1 classification signal
  - Chain of convolution and pooling layers from input to output
- Dense prediction problems require high resolution and large field-of-view
  - Semantic segmentation, object localization, optical flow prediction, etc
- What are the right architectures?
  - Filter sizes, positioning of convolutions vs pooling, type of pooling, etc
  - Are chain-structured networks the best for classification ?



Multi-scale network architectures

- Grid of network layers across multiple scales
  - Feed-forward across the horizontal “layer axis”
  - Nothing new in training: standard back-prop gradient calculation
- Chain-structured networks (eg for classification) and other networks (such as Unet for segmentation) are special cases of this more general structure

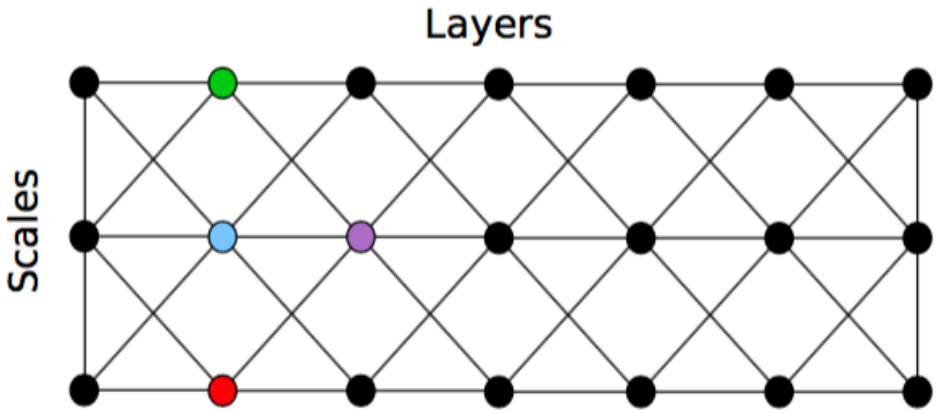


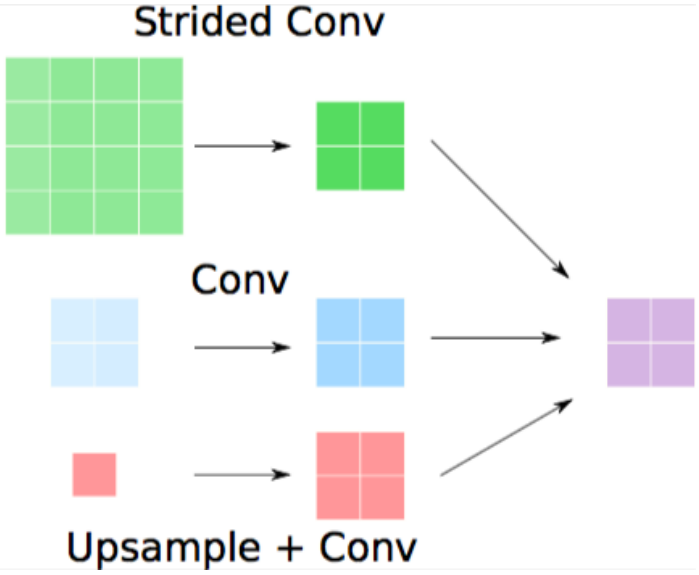
[Saxena & Verbeek, NIPS 2016]

Convolutional neural fabrics

[Saxena & Verbeek, NIPS 2016]

- Each feature map receives input from three others
  - Scale finer: strided convolution
  - Scale coarser: stride coarse activations on finer resolution, then covolution
  - Same scale: standard convolution
- Generalizes very large class of networks with “standard” layers
- With enough layers and feature channels, 3x3 convolutions suffice for
  - Average pooling, max-pooling, and strided convolition
  - Nearest-neighbor, bi-linear, and general deconvolution up-sampling
  - Filters of any size by distribution over layers



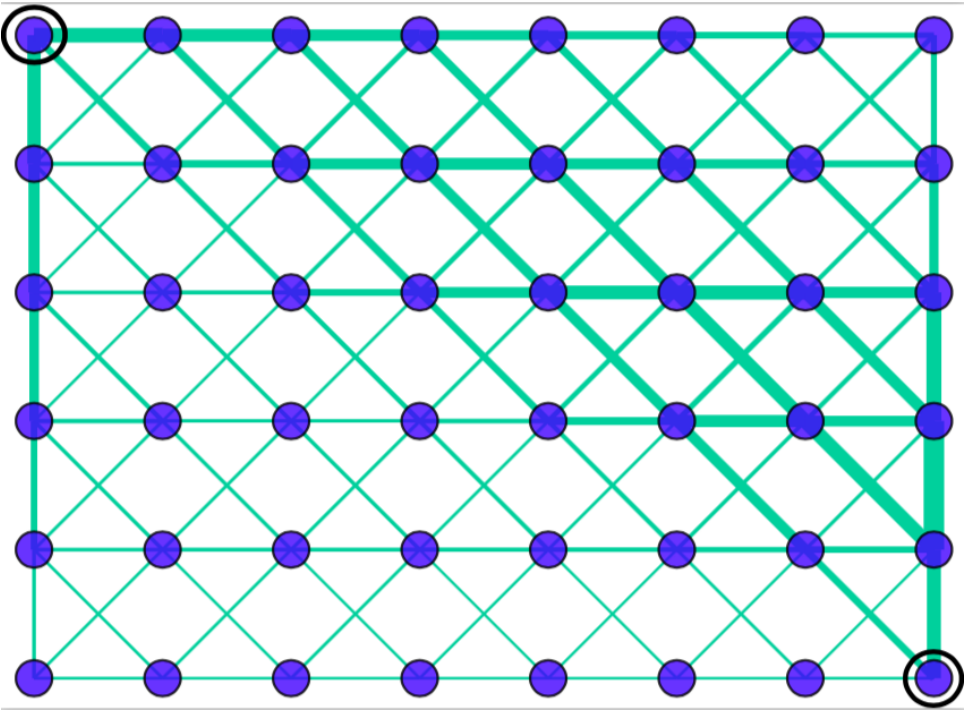


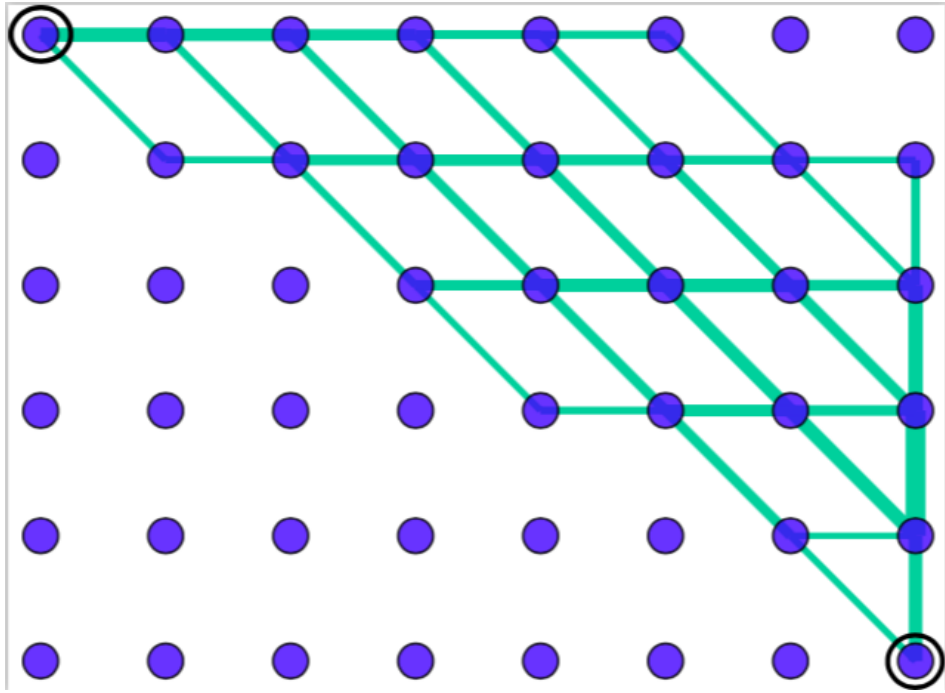
Convolutional neural fabrics

[Saxena & Verbeek, NIPS 2016]

- Connection strengths in a fabric learned for image classification
- Weak connections may be suppressed
  - CIFAR-10: reduce nr. of connections by factor 3, error up from 7.4% to 8.1%
- Search over cost effective networks can be integrated in training

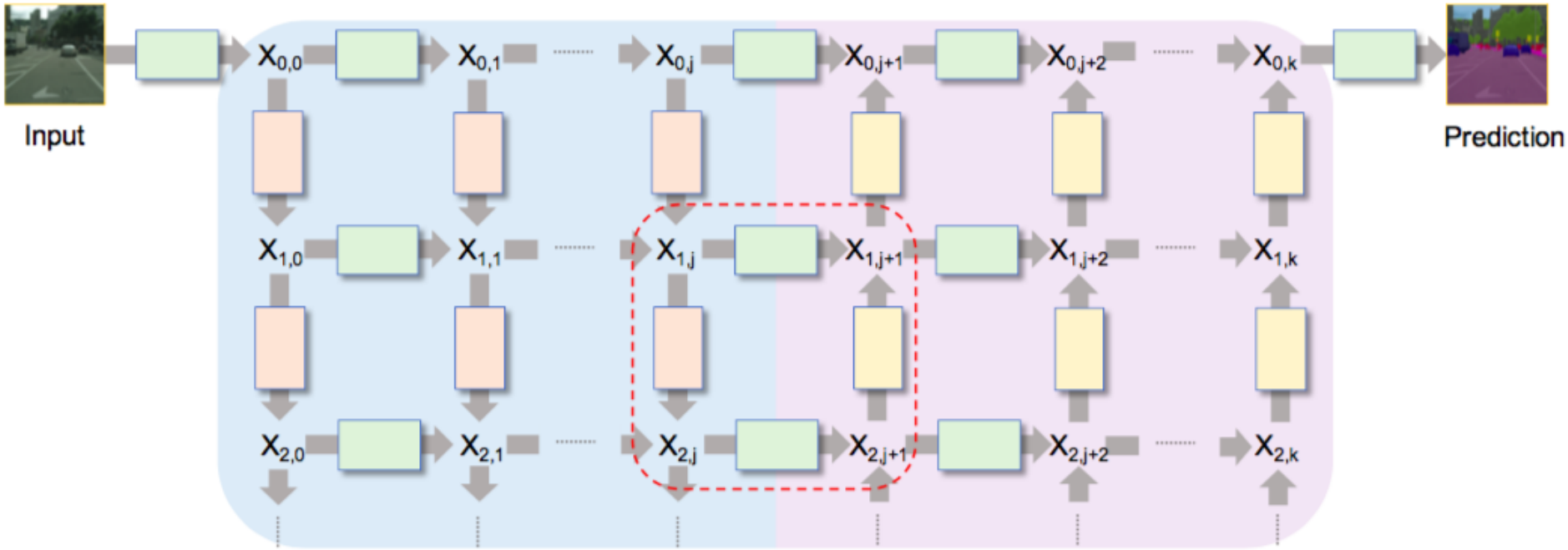
  [Veniat & Denoyer, arXiv'17]





Residual conv-deconv grid network for segmentation

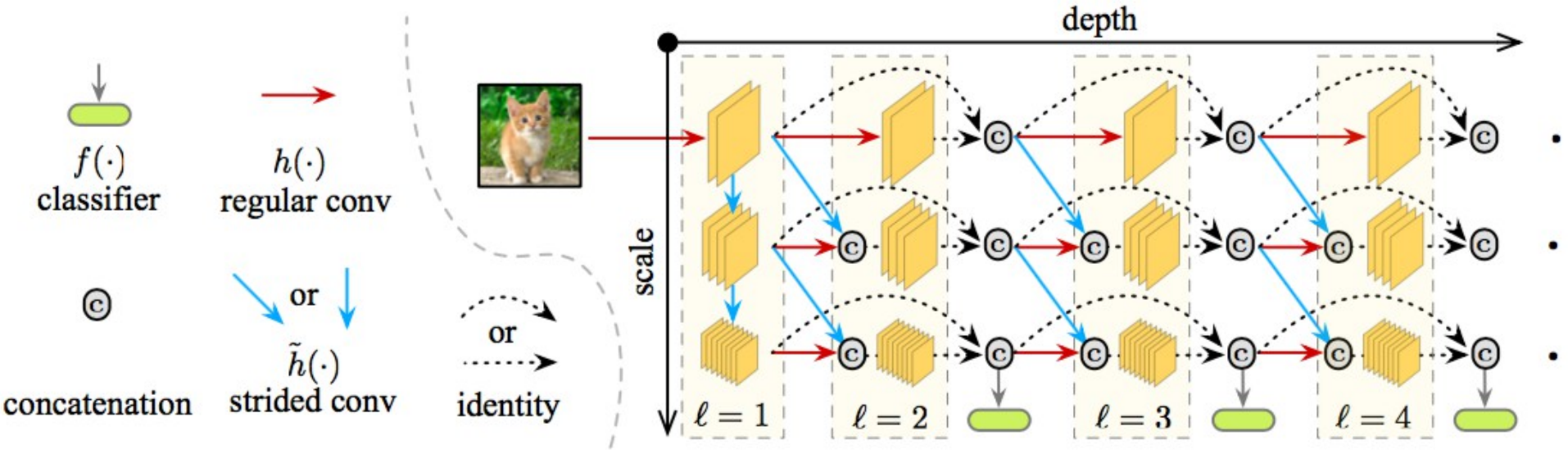
- Grid of network layers across multiple scales
  - Feed-forward and residual across the horizontal “layer axis”
- Down-sampling block followed by up-sampling block
- Accuracy close to state of the art (similar to FRRN), trained “from scratch”
  - Few thousand training images, instead of pre-trained ImageNet classification



[Fourure et al, BMVC 2017]

Multi-scale Dense Convolutional Networks

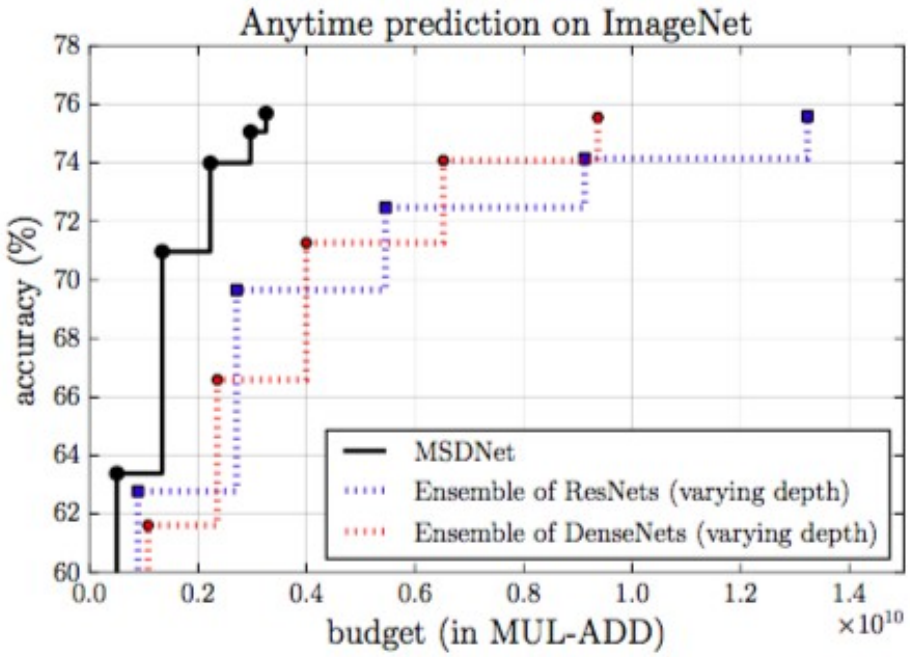
- Grid of network layers across multiple scales
  - Feed-forward and dense connections across the horizontal “layer axis”
- Down-sampling across all layers for classification
- Intermediate classifiers for any-time prediction

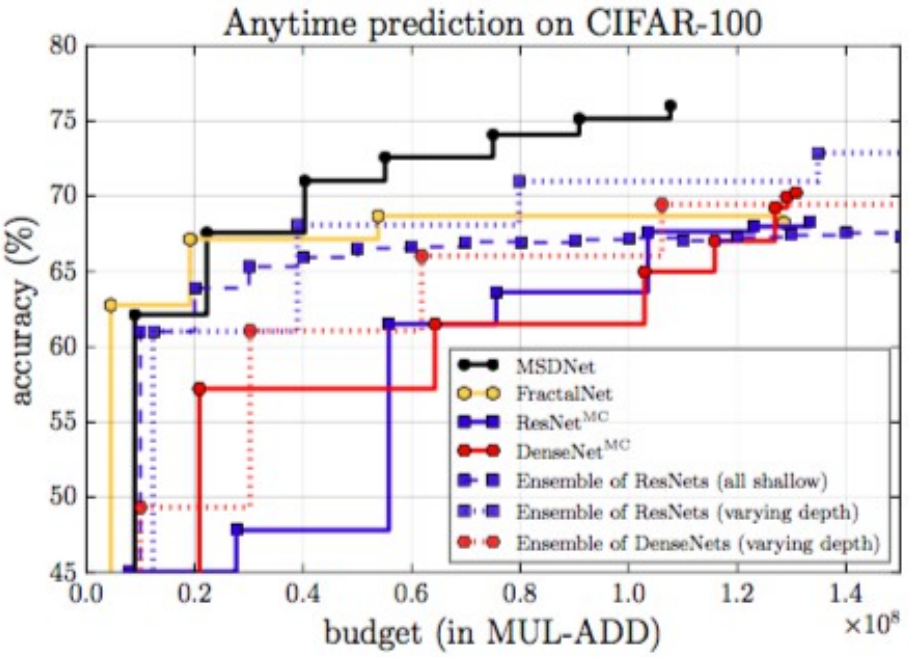


[Huang et al, arXiv 2017]

Multi-scale Dense Convolutional Networks

- Efficient any-time prediction model
  - Features computed for early classifiers are re-used for later classifiers





[Huang et al, arXiv 2017]