CS 7643: Deep Learning

Topics:

- Toeplitz matrices and convolutions = matrix-mult
- Dilated/a-trous convolutions
- Backprop in conv layers
- Transposed convolutions

Dhruv Batra

Georgia Tech

Administrativia

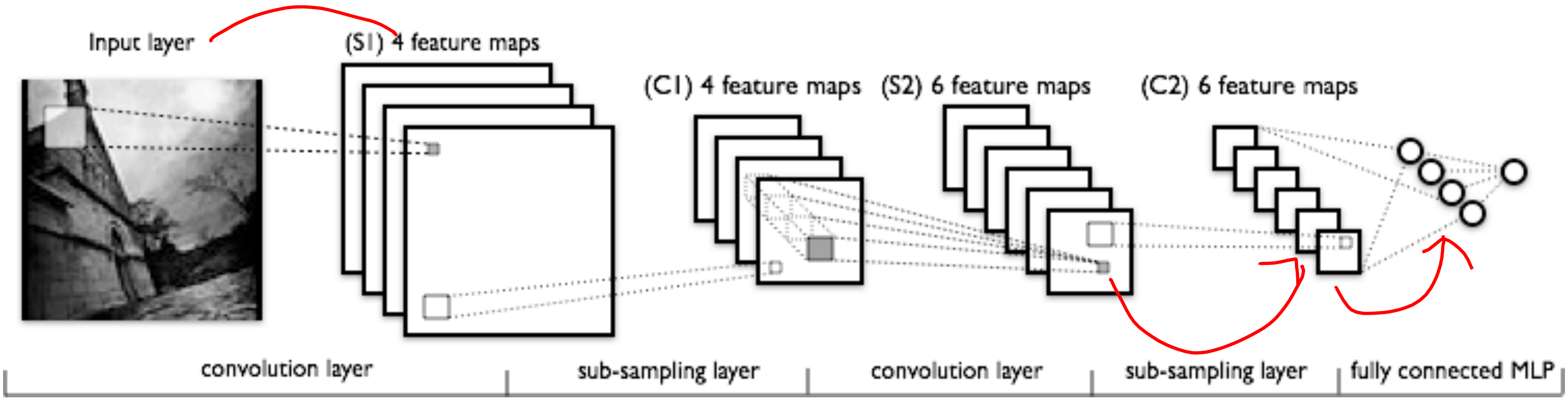
- HW1 extension
  - ~~09/22~~ 09/25
- HW2 + PS2 both coming out on ~~09/22~~ 09/25
- Note on class schedule coming up
  - Switching to paper reading starting next week.
  - https://docs.google.com/spreadsheets/d/1uN31YcWAG6nhjvYPUVKMy3vHwW-h9MZCe8yKCqw0RsU/edit#gid=0
- First review due: ~~Tue~~ Mon 09/26
- First student presentation due: Thr 09/28

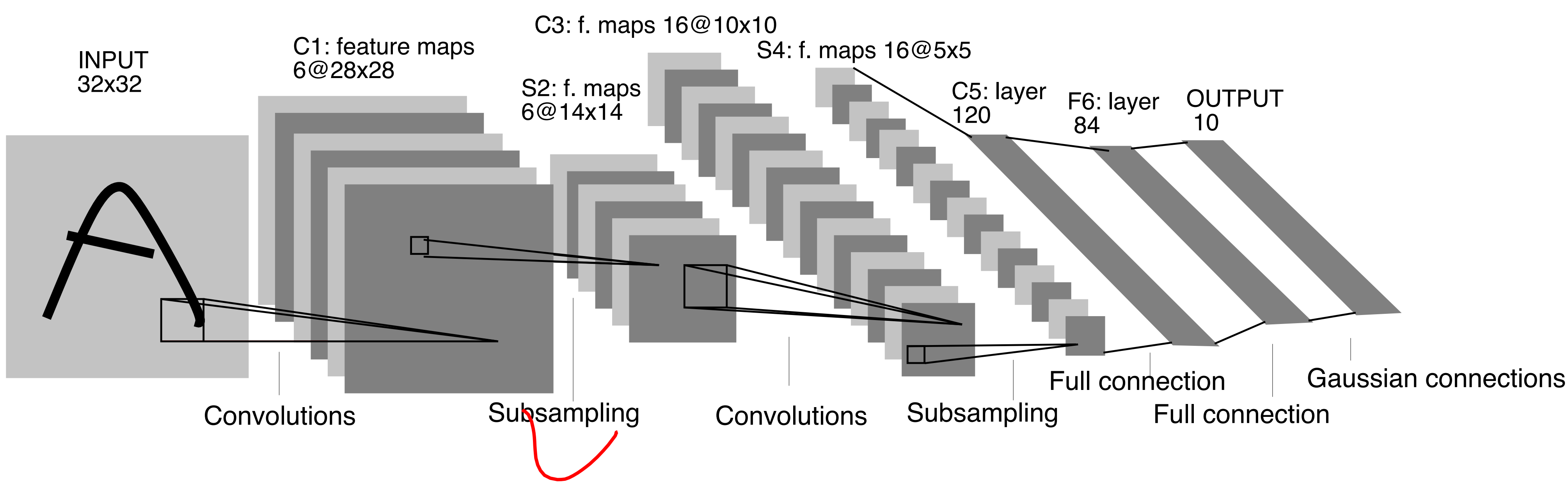
Recap of last time

Convolutional Neural Networks

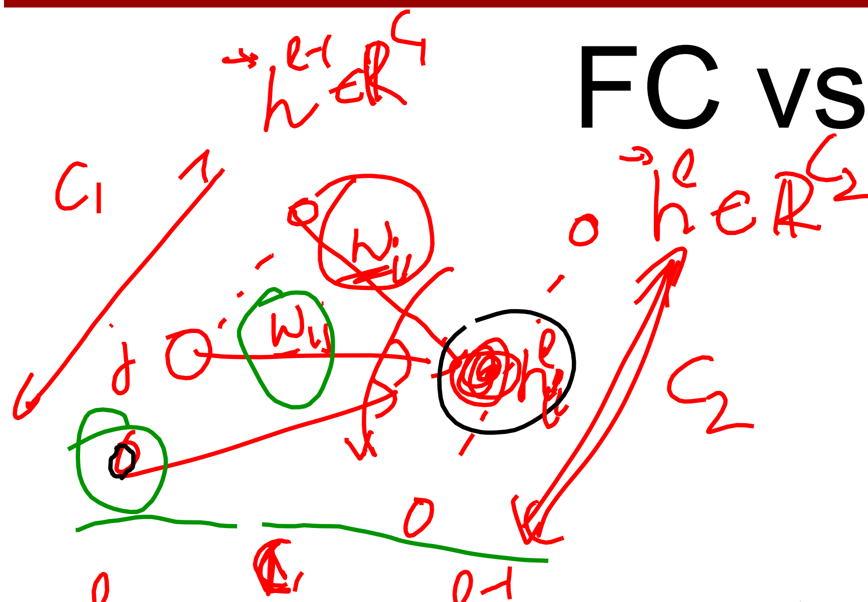
(without the brain stuff)

Convolutional Neural Networks





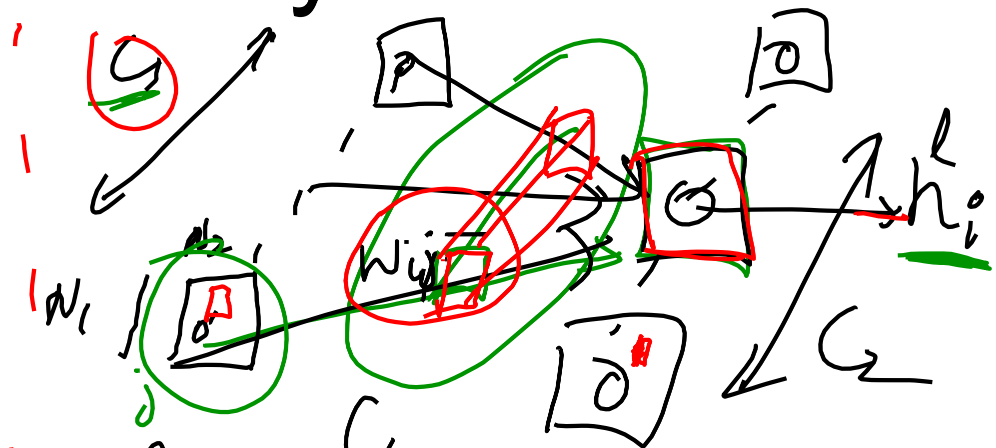
FC vs Conv Layer



$$h_i^l = \sum_{j=1}^{C_1} h_j^{l-1} \cdot W_{ij} + b_i$$

$$\vec{W}_i^T = \left[\longleftrightarrow \right] \quad b_i$$

$$(1 + C_1) \times C_2$$



$$h_i^l = \sum_{j=1}^{C_1} h_j^{l-1} * W_{ij} + b_i$$

$$h_i^l[r, c] = \sum_{j=1}^{C_1} \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} h_j^{l-1}[r+a, c+b] \, W_{ij}[a, b]$$

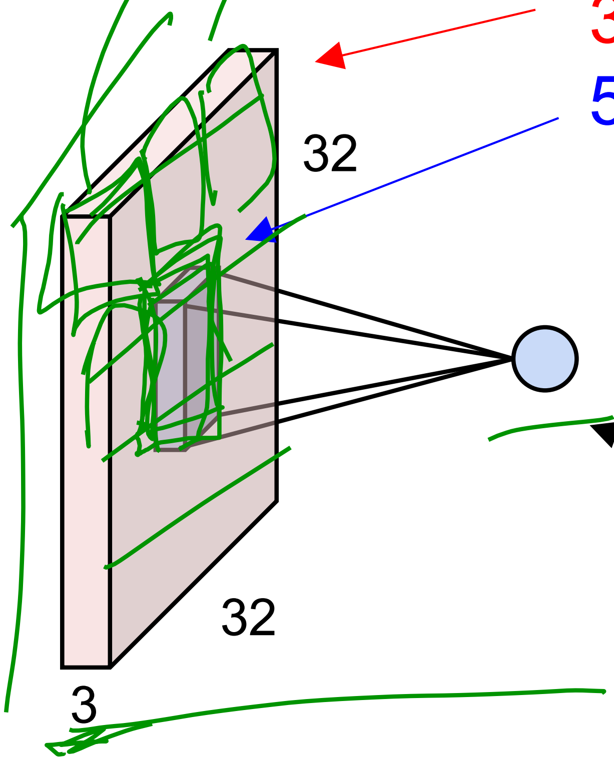
$$k_1 \times k_2 \times C_1 \quad b_i$$

$$(1 + k_1 \times k_2 \times C_1) \times C_2$$

C_2

Convolution Layer

W(:)



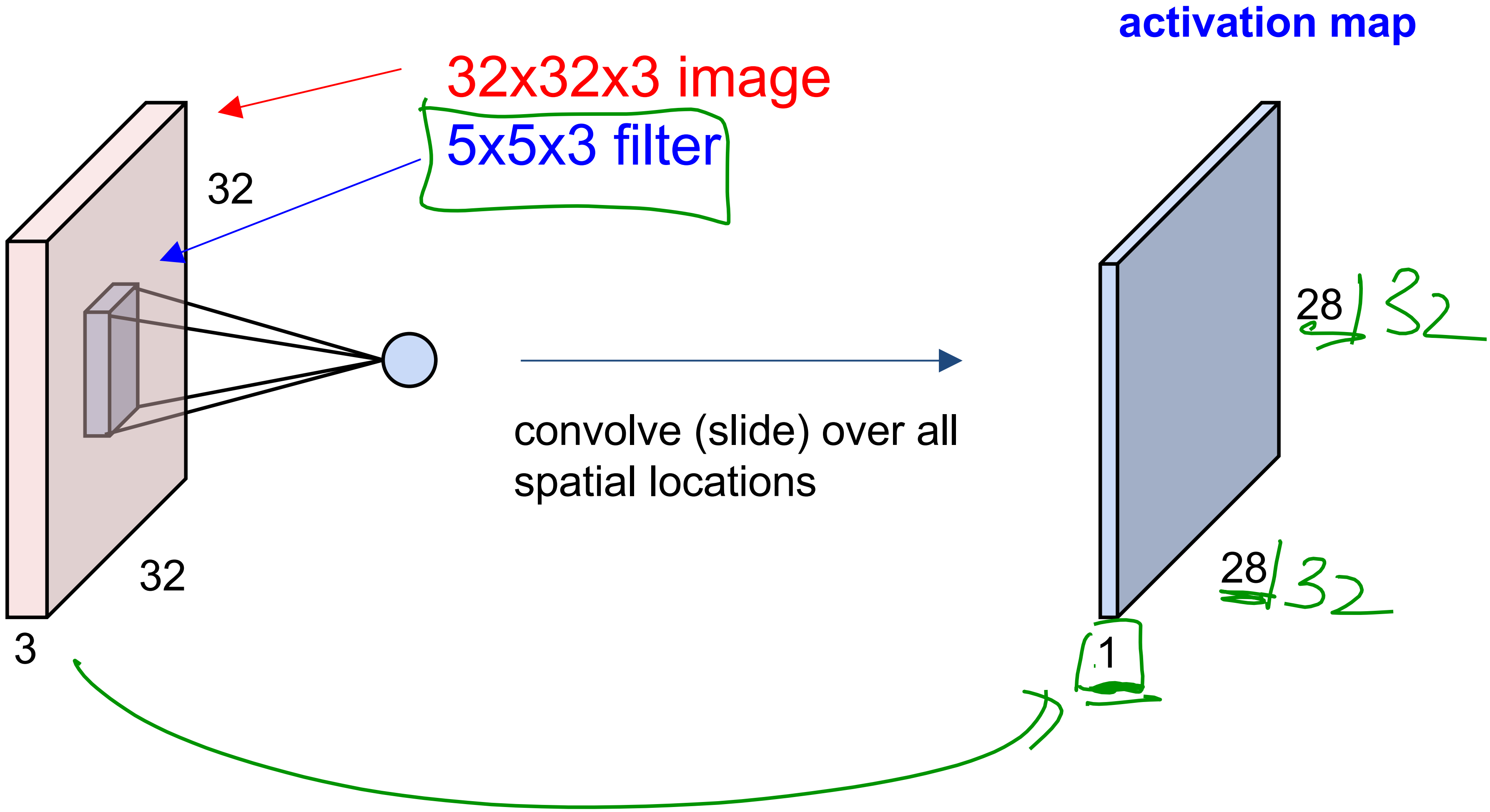
32x32x3 image

5x5x3 filter w

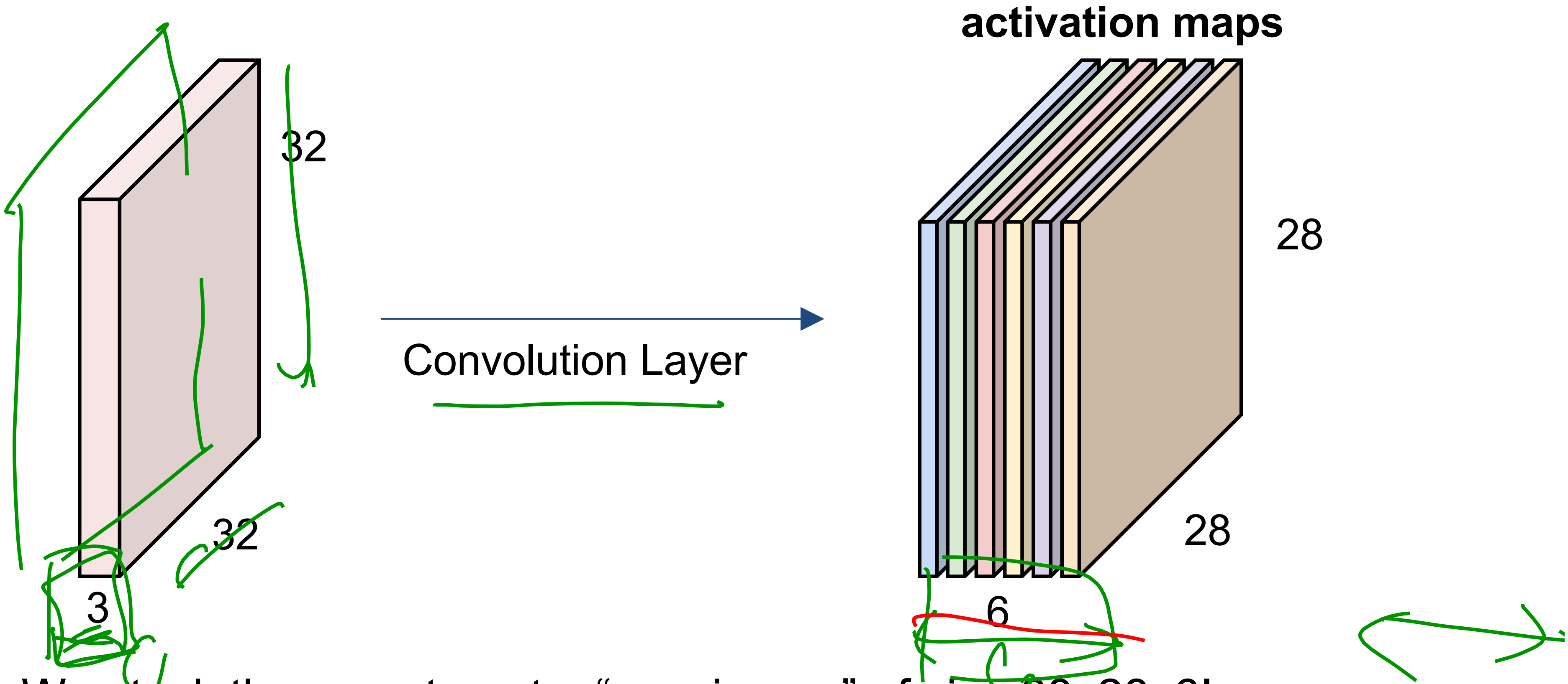
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

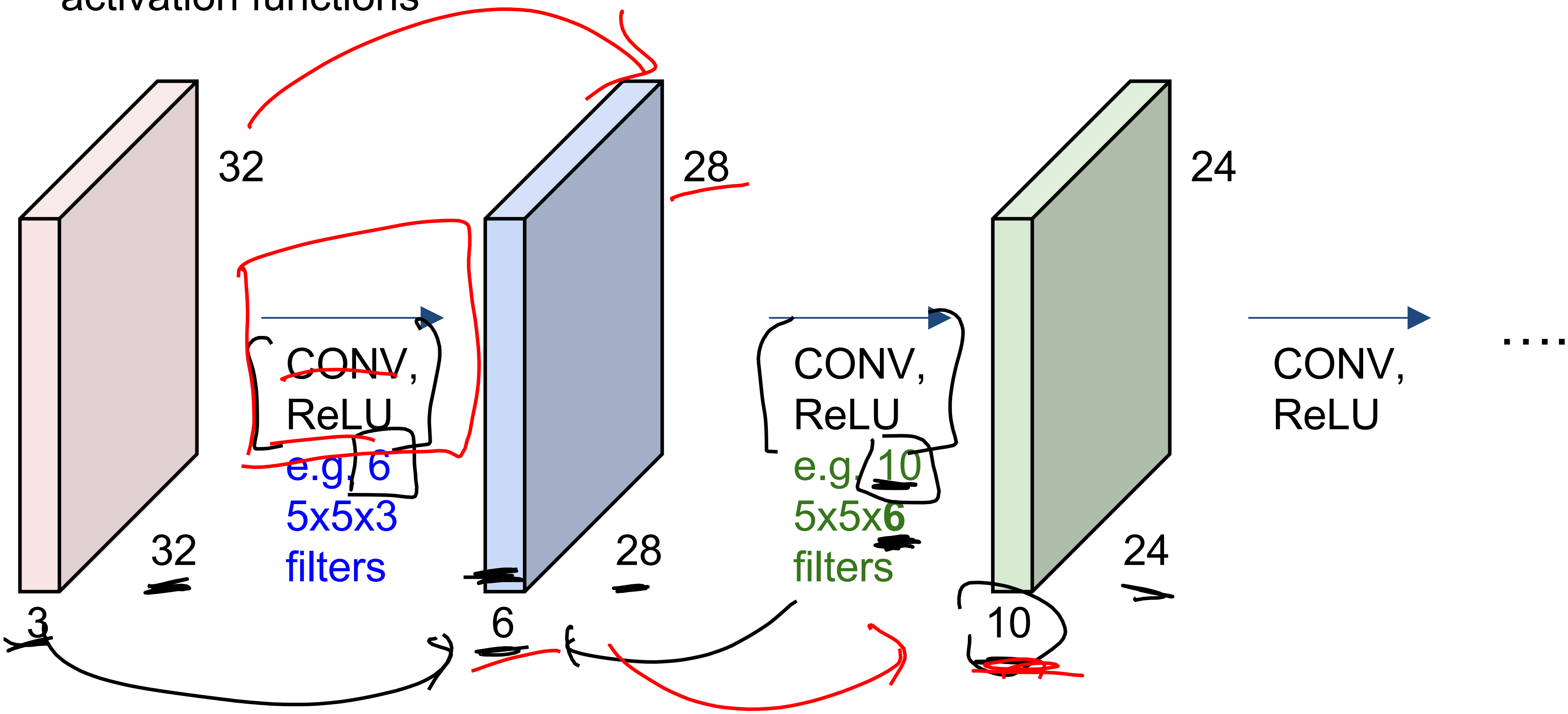


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



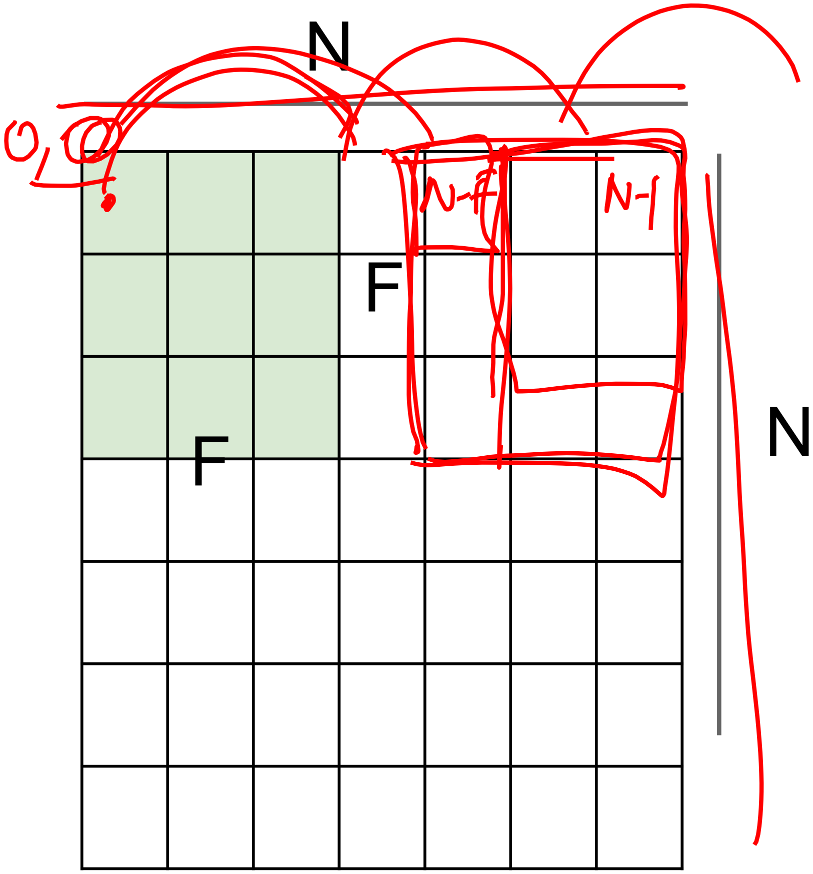
We stack these up to get a "new image" of size 28x28x6!

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



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

$(N-F) \propto \text{stride}$



Output size:

(N - F) / stride + 1

+2P

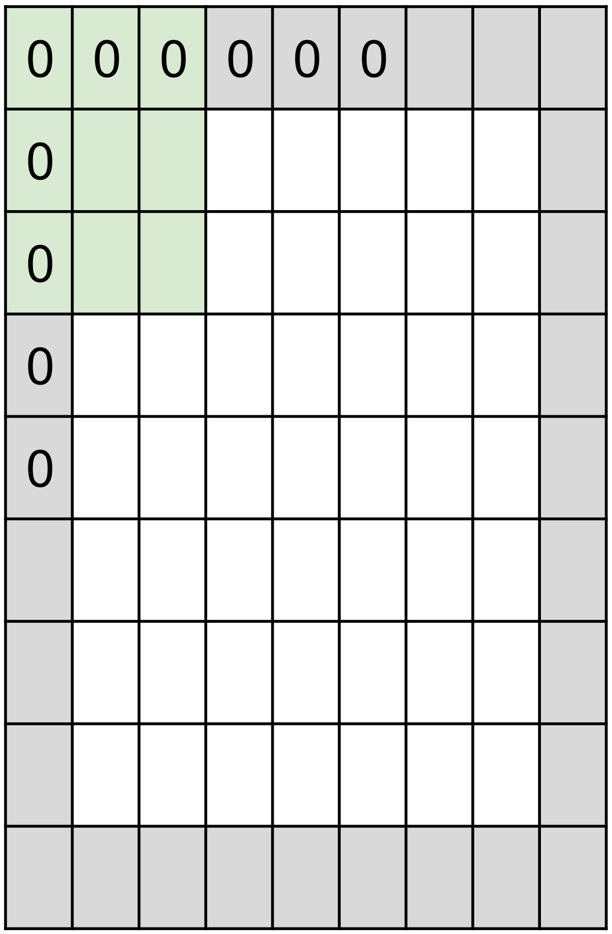
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border

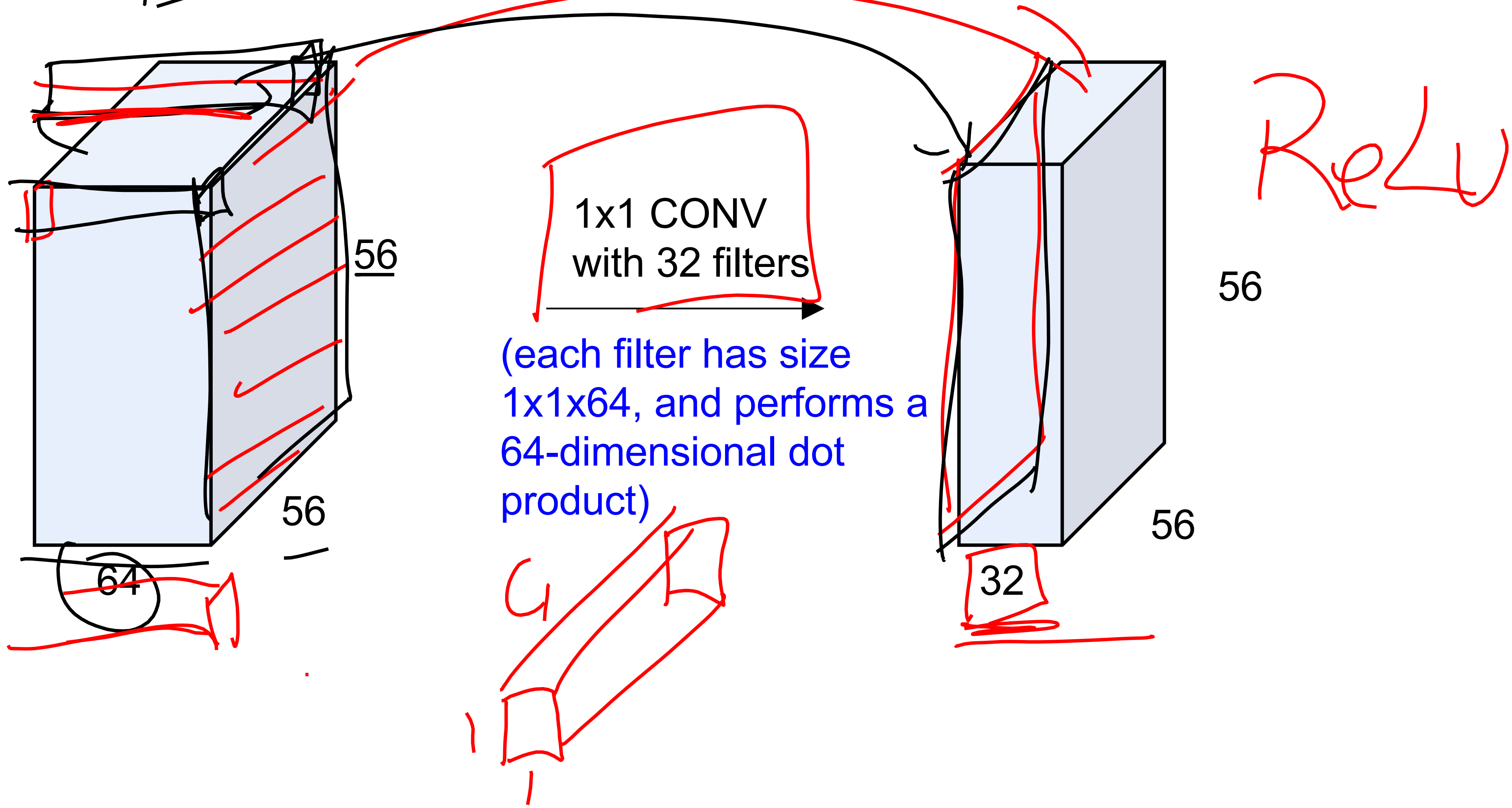


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

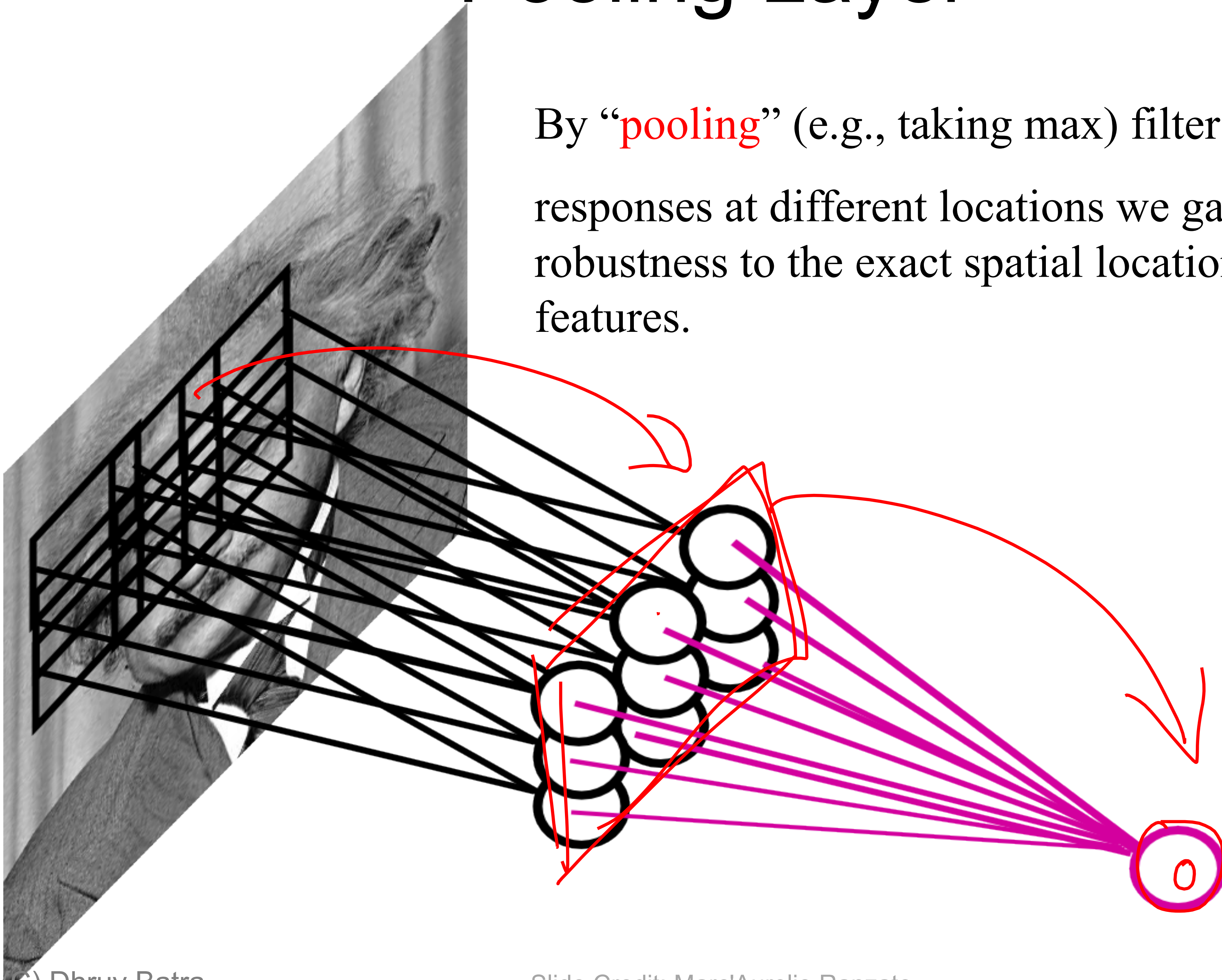
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with $\boxed{(F-1)/2}$. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

$[1 \times 1 \times C_1] \times C_2$

(btw, 1x1 convolution layers make perfect sense)



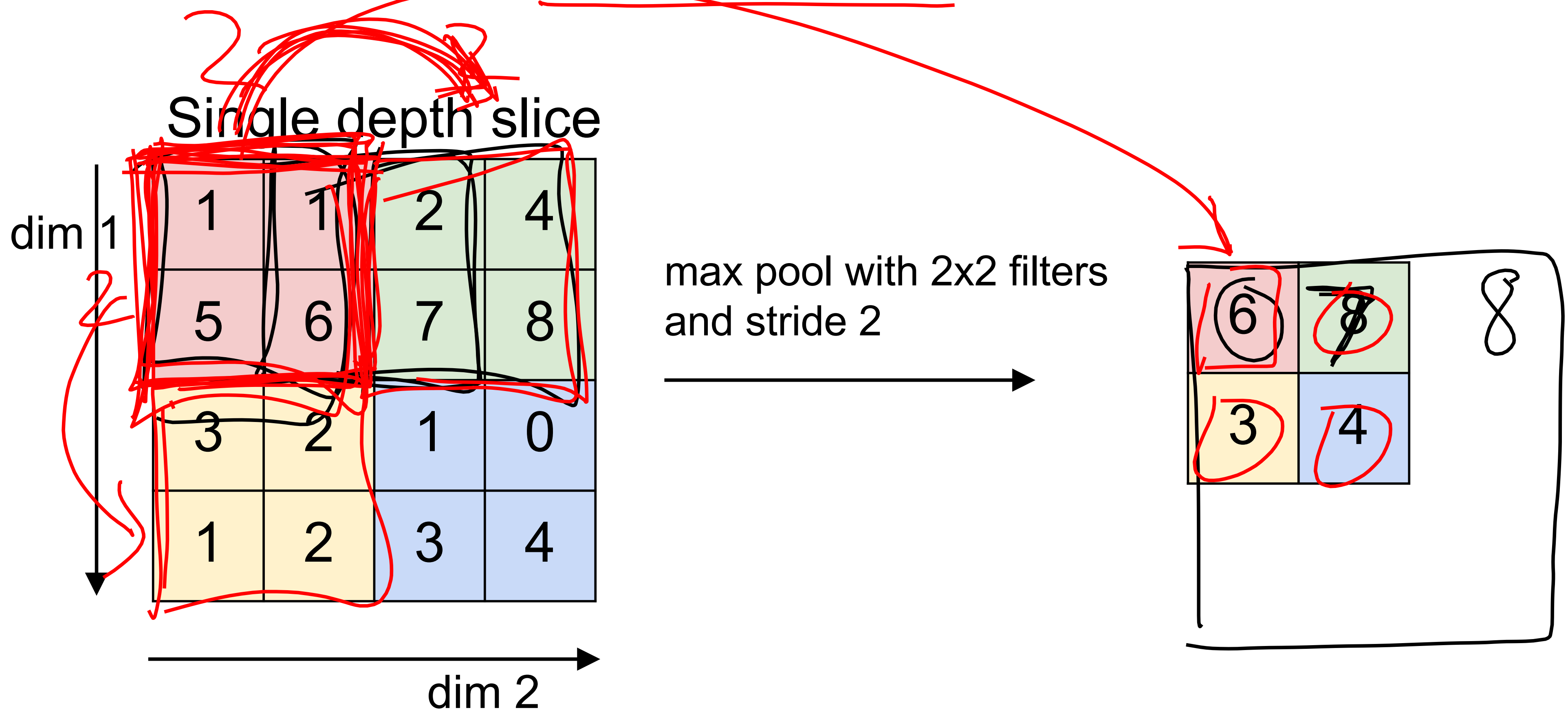
Pooling Layer



By "pooling" (e.g., taking max) filter

responses at different locations we gain robustness to the exact spatial location of features.

MAX POOLING



Single depth slice
dim 1
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
dim 2
max pool with 2x2 filters
and stride 2
6 8
3 4
8

Pooling Layer: Examples

Max-pooling:

$$h_i^n(r,c) = \max_{\bar{r} \in N(r),\ \bar{c} \in N(c)} h_i^{n-1}(\bar{r},\bar{c})$$

Average-pooling:

$$h_i^n(r,c) = \operatorname*{mean}_{\bar{r} \in N(r),\ \bar{c} \in N(c)} h_i^{n-1}(\bar{r},\bar{c})$$

L2-pooling:

$$h_i^n(r,c) = \sqrt{\sum_{\bar{r} \in N(r),\ \bar{c} \in N(c)} h_i^{n-1}(\bar{r},\bar{c})^2}$$

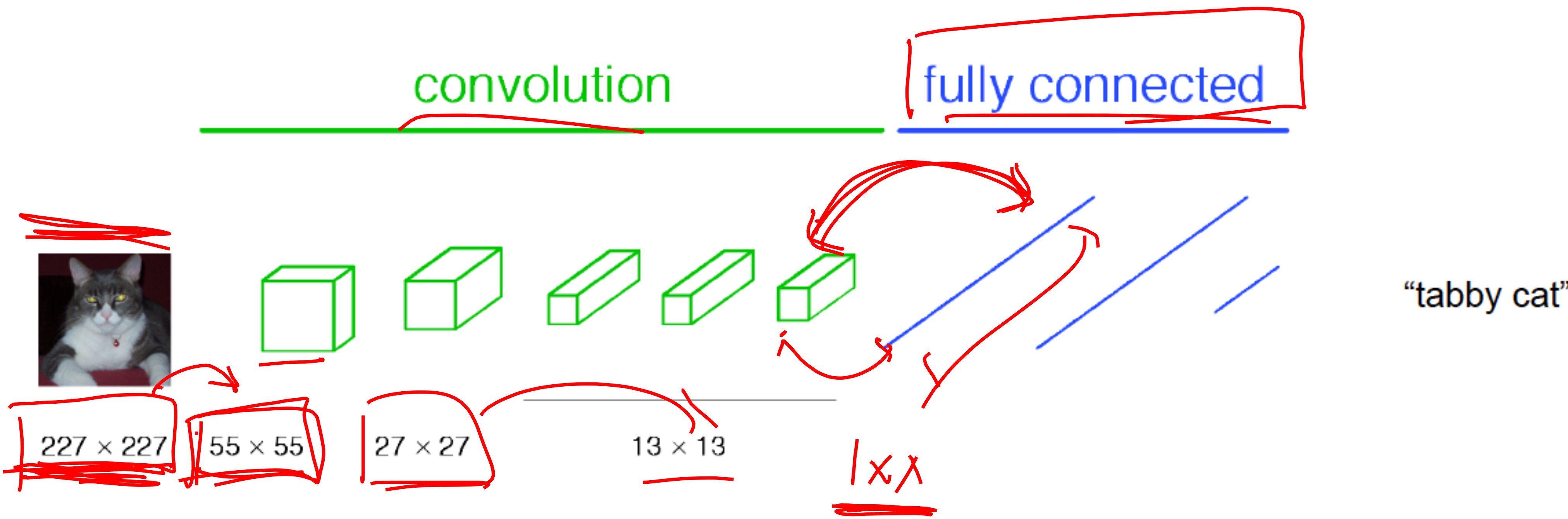
L2-pooling over features:

$$h_i^n(r,c) = \sqrt{\sum_{j \in N(i)} h_i^{n-1}(r,c)^2}$$

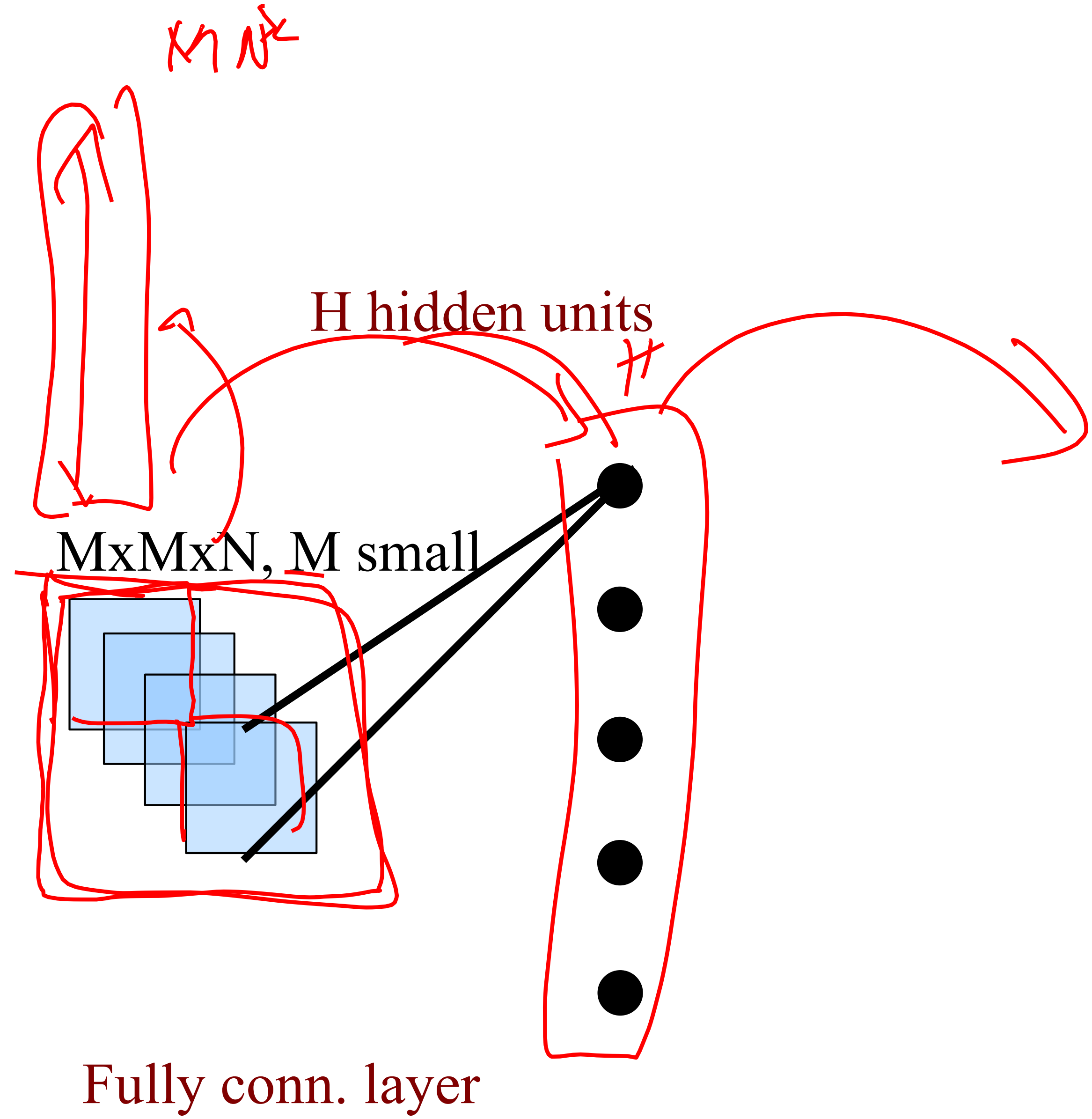
2

3

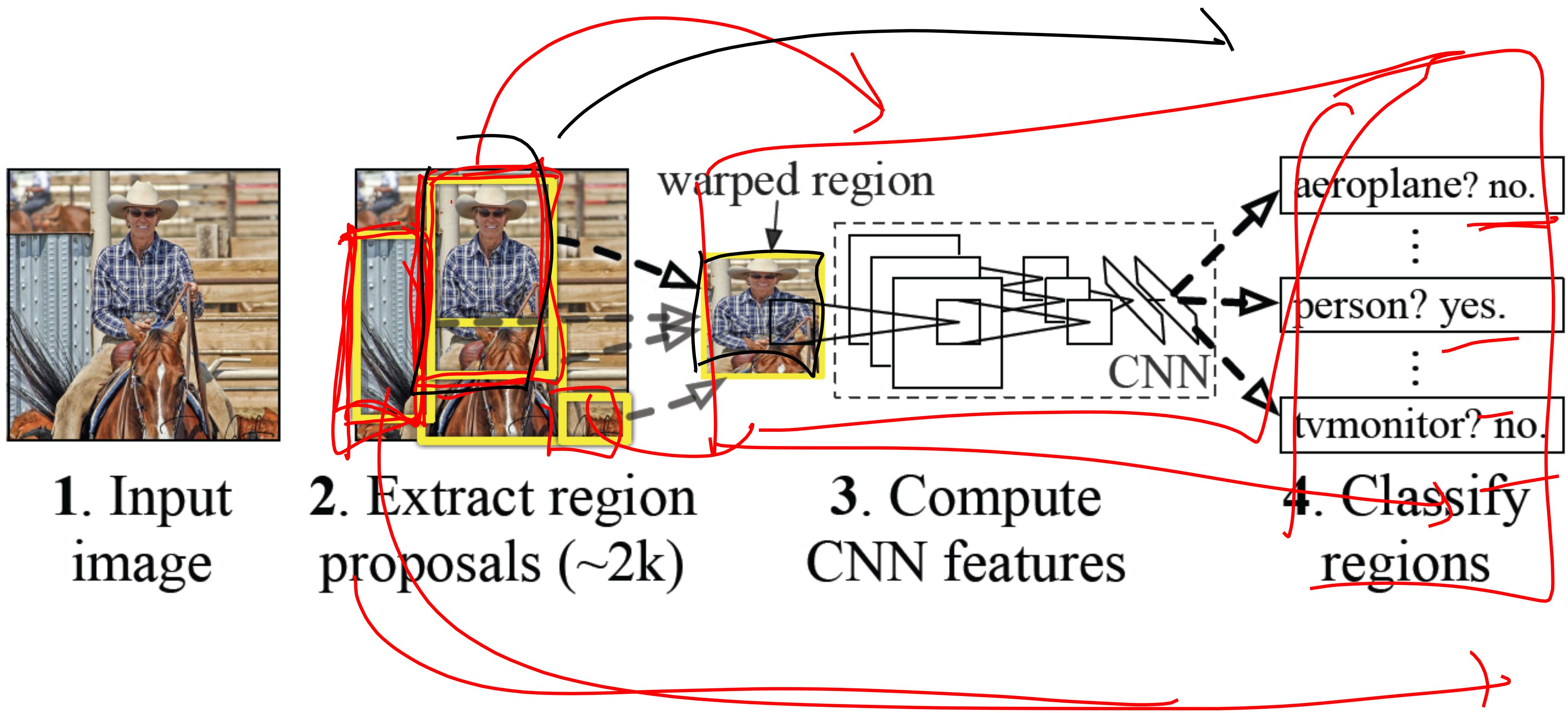
Classical View



512x512


H hidden units
MxMxN, M small
H
Fully conn. layer

Classical View = Inefficient



Classical View

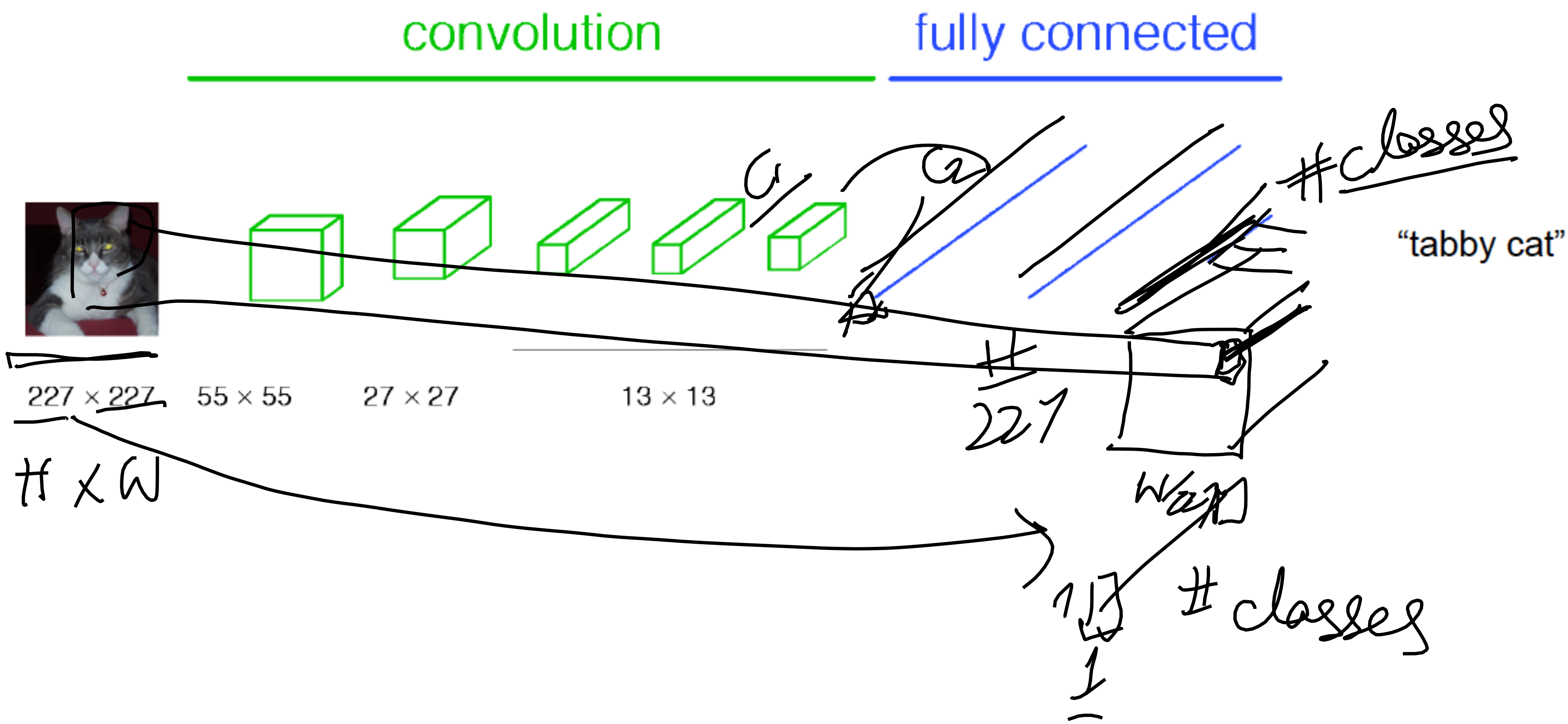


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

- Just squint a little!

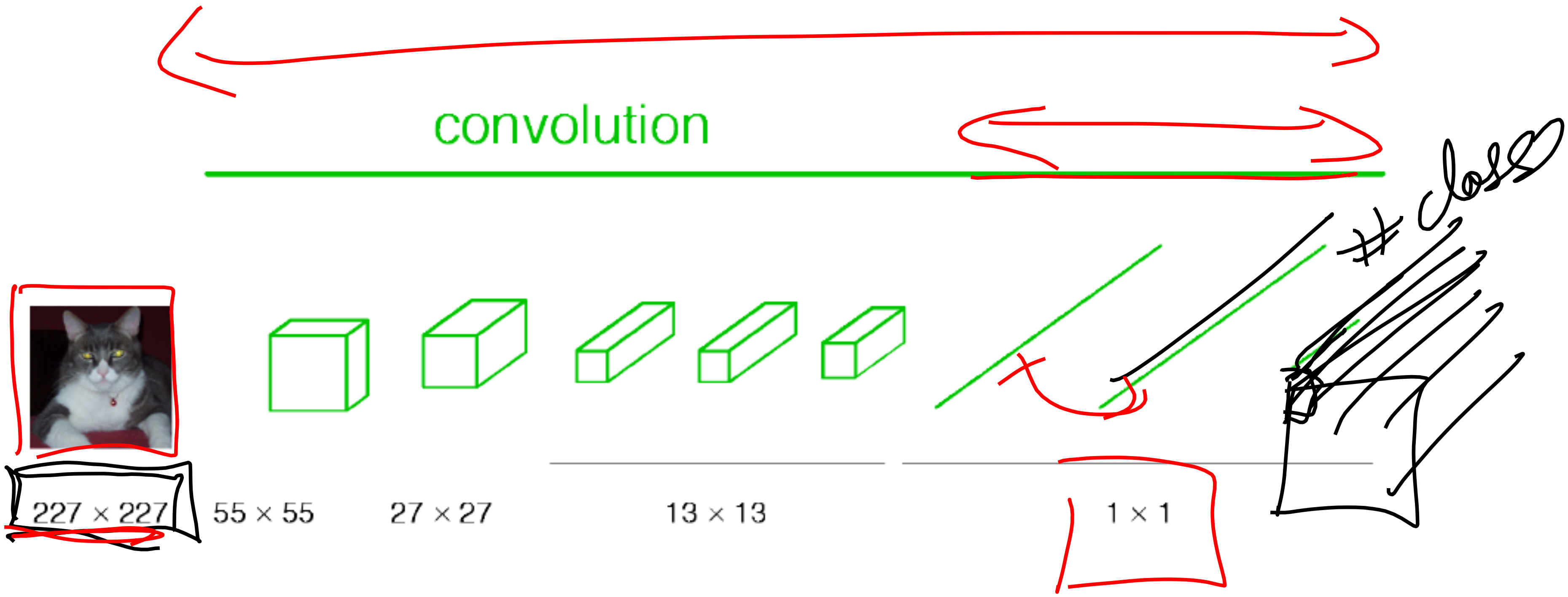
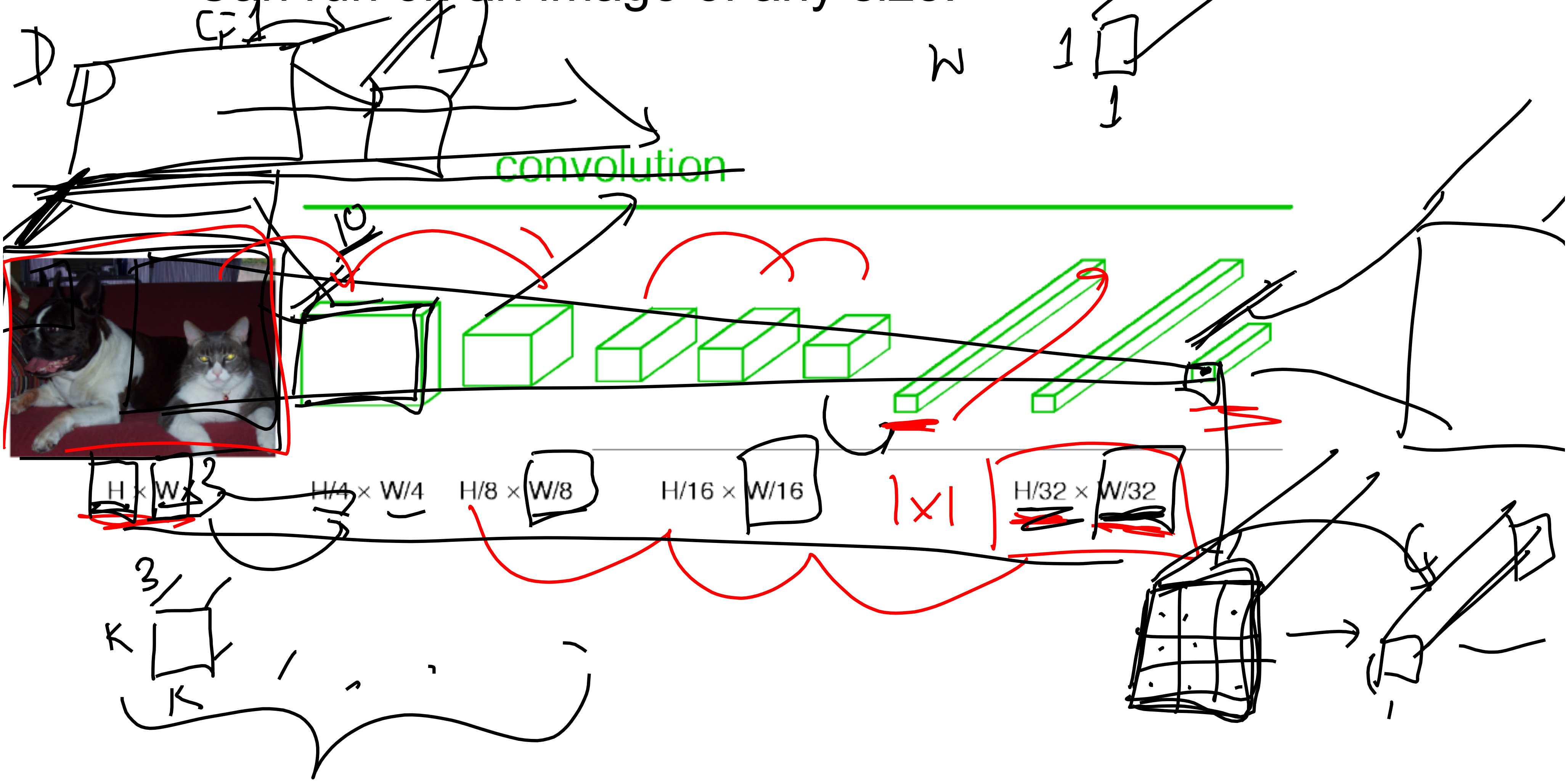
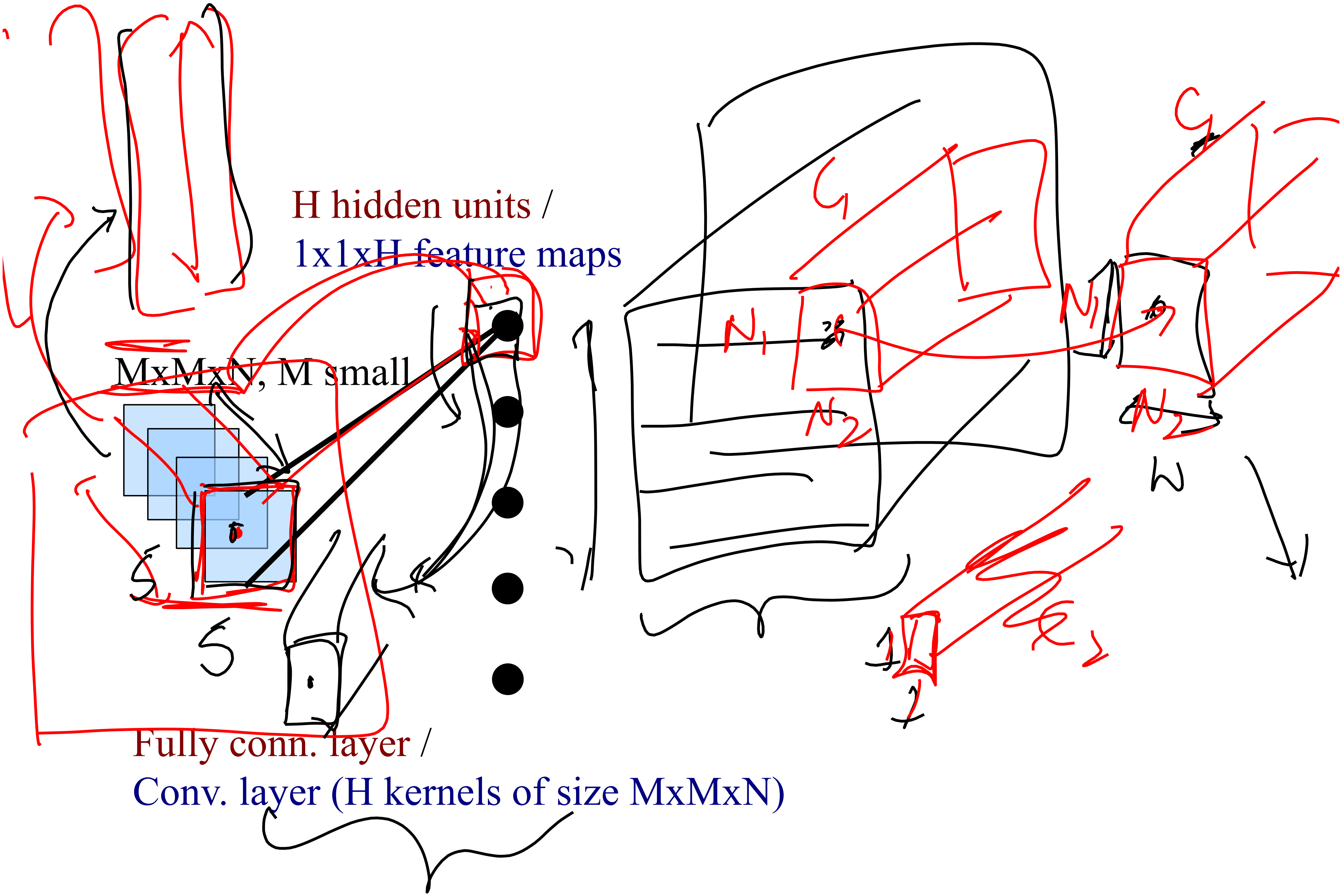


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

"Fully Convolutional" Networks

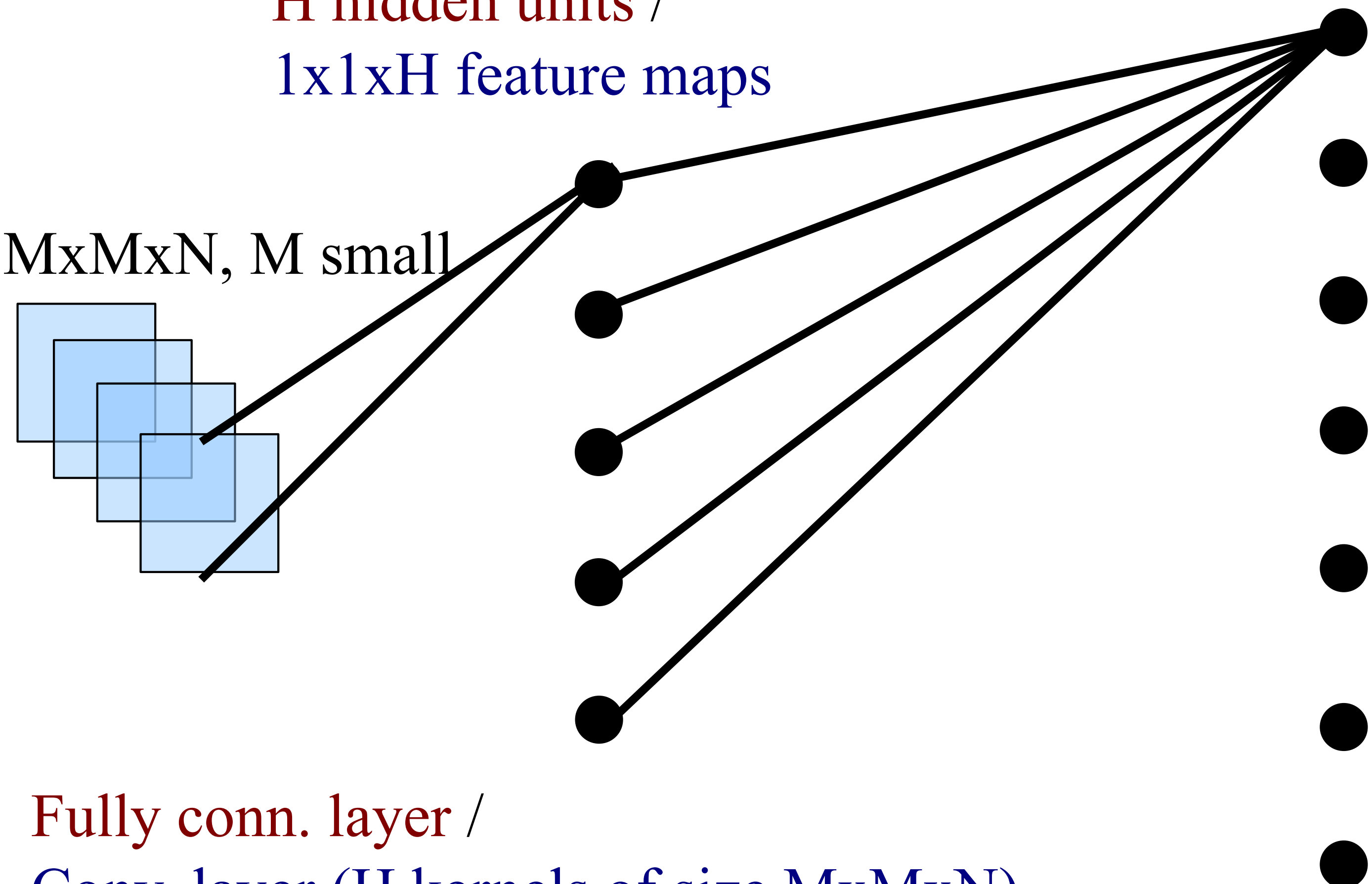
- Can run on an image of any size!




H hidden units /
1x1xH feature maps
MxMxN, M small
Fully conn. layer /
Conv. layer (H kernels of size MxMxN)

K hidden units /
1x1xK feature maps

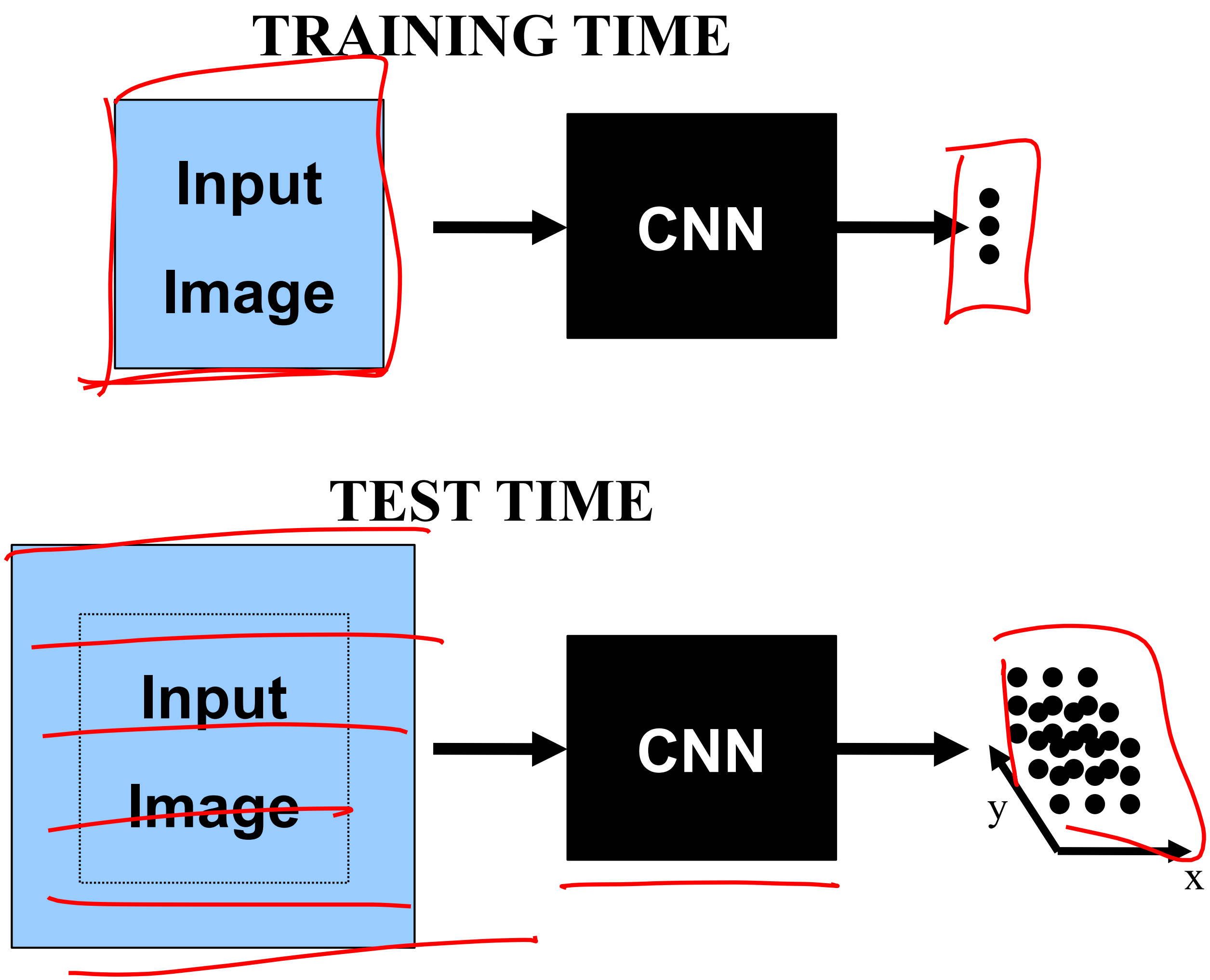
H hidden units /
1x1xH feature maps



Fully conn. layer /
Conv. layer (H kernels of size MxMxN)

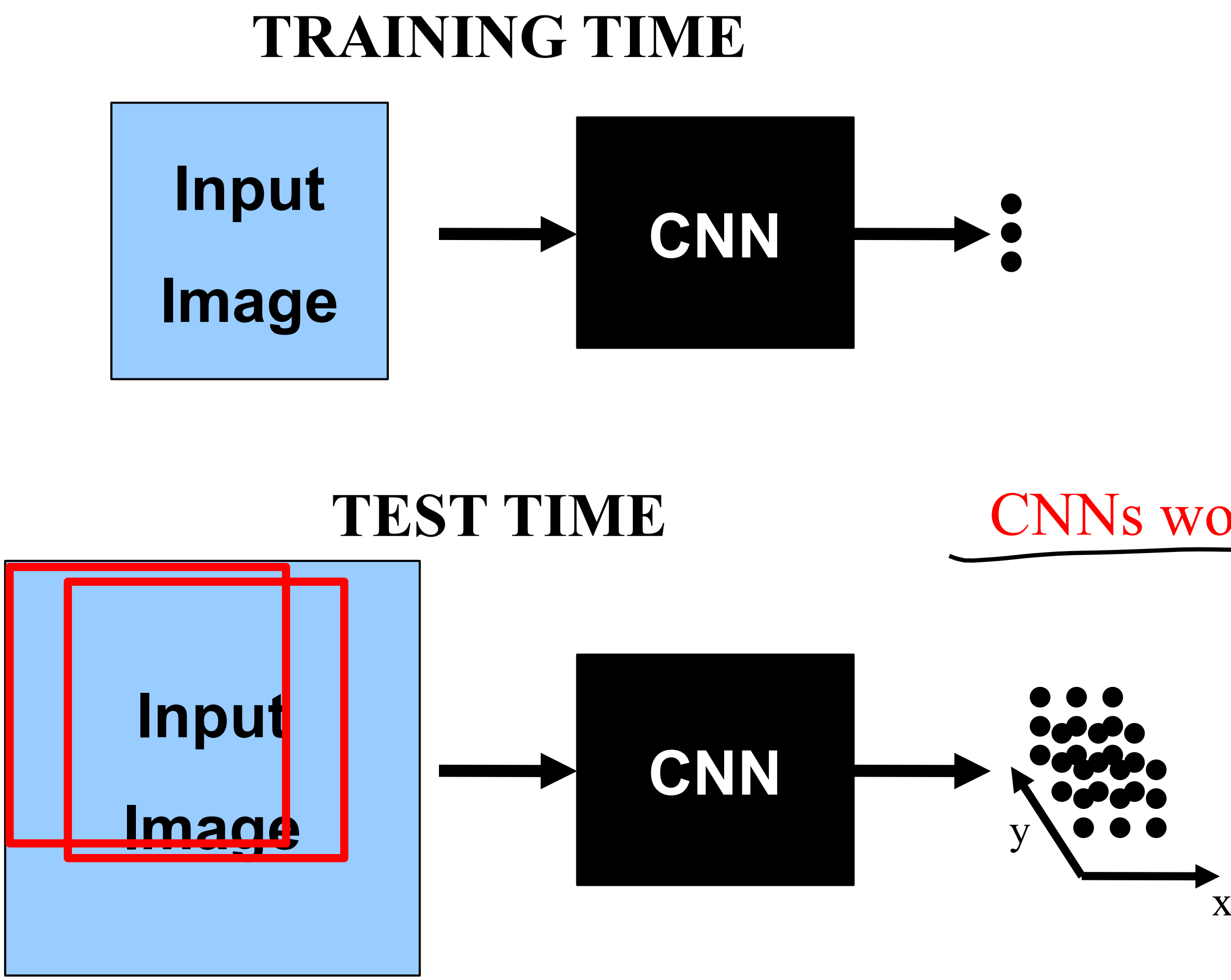
Fully conn. layer /
Conv. layer (K kernels of size 1x1xH)

Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).



Slide Credit: Marc'Aurelio Ranzato

Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).



CNNs work on any image size!

Unrolling is order of magnitudes more eficient than sliding windows!

Benefit of this thinking

- Mathematically elegant
- Efficiency
  - Can run network on arbitrary image
  - Without multiple crops

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX
  where N is usually up to ~5, M is large, 0 <= K <= 2.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm

Plan for Today

- Convolutional Neural Networks
  - Toeplitz matrices and convolutions = matrix-mult
  - Dilated/a-trous convolutions
  - Backprop in conv layers
  - Transposed convolutions

Toeplitz Matrix

- Diagonals are constants

$$\begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}.$$

- $A_{ij} = a_{\boxed{i-j}}$

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

Why do we care?

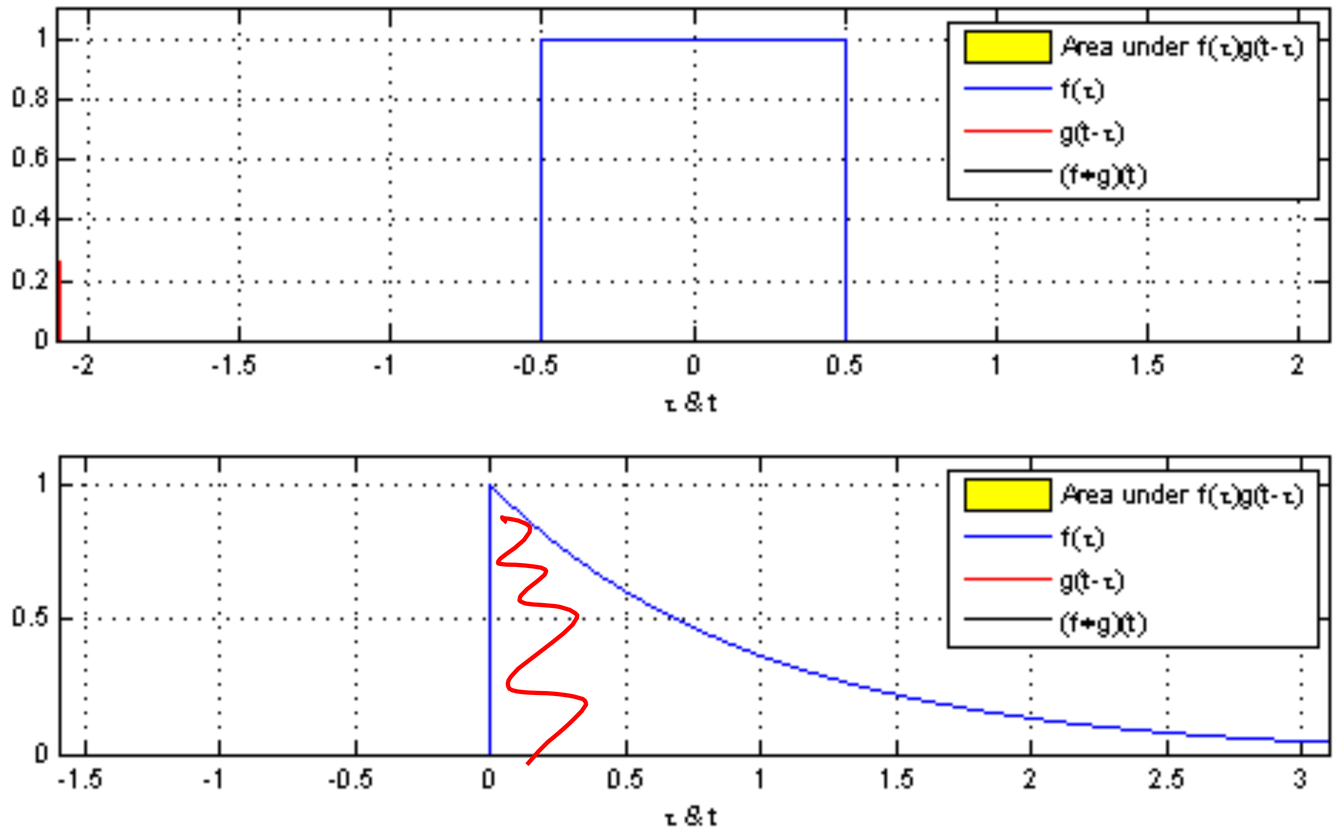
- (Discrete) Convolution = $\boxed{\text{Matrix Multiplication}}$
  - with Toeplitz Matrices

$[w_1 \ldots w_k]$

$$y = w * x \quad \begin{bmatrix} w_k & 0 & \ldots & 0 & 0 \\ w_{k-1} & w_k & \ldots & 0 & 0 \\ w_{k-2} & w_{k-1} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_1 & w_{k-2} & \ldots & w_k & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & w_1 & \ldots & w_{k-1} & w_k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \vdots & w_1 & w_2 \\ 0 & 0 & \vdots & 0 & w_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

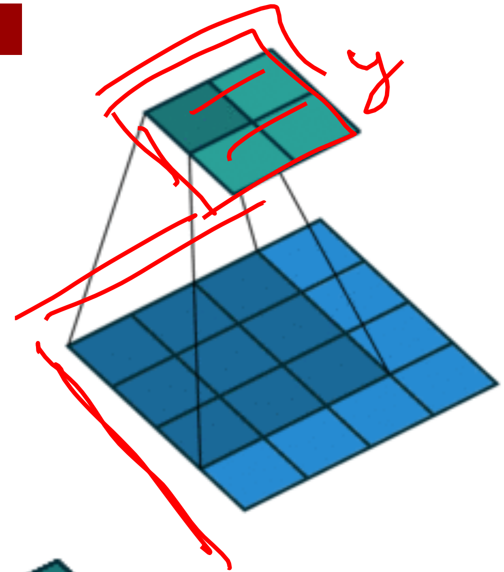
w_k

y

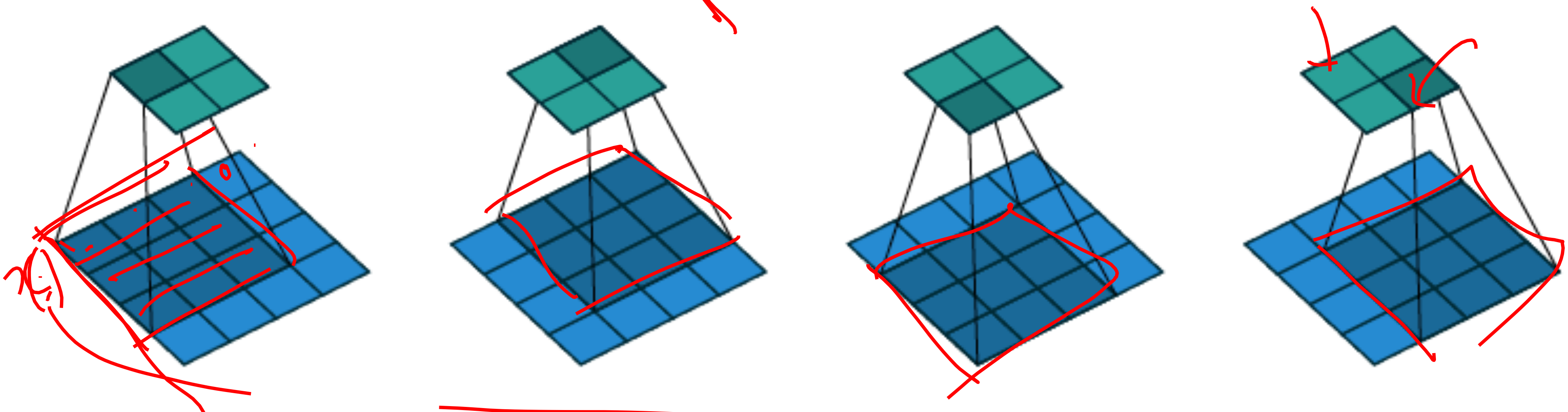


"Convolution of box signal with itself2" by Convolution_of_box_signal_with_itself.gif: Brian Ambergderivative work: Tinos (talk) - Convolution_of_box_signal_with_itself.gif. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Convolution_of_box_signal_with_itself2.gif#/media/File:Convolution_of_box_signal_with_itself2.gif

$y[r,c] = \sum x(\ ) \, w(\ )$



W^T



$$\underset{4\times 1}{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}} = \underset{4\times 16}{\begin{pmatrix}
w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\
0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\
0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2}
\end{pmatrix}} \underset{16\times 1}{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}$$

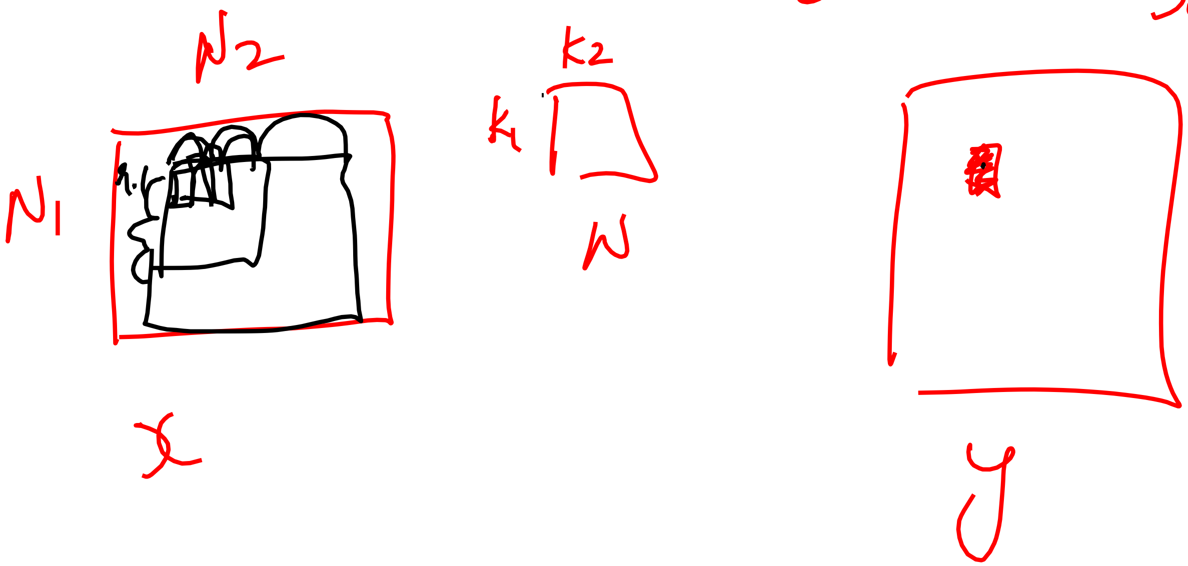
$y = Wx$

Plan for Today

- Convolutional Neural Networks
  - Toeplitz matrices and convolutions = matrix-mult
  - Dilated/a-trous convolutions
  - Backprop in conv layers
  - Transposed convolutions

Dilated Convolutions

$$y[r,c] = \sum_{a=0}^{k_1-1}\sum_{b=0}^{k_2-1} x[r+\underbrace{2a}_{5a},\ c+\underbrace{b}_{5b}]\, w[a,b]$$



Dilated Convolutions

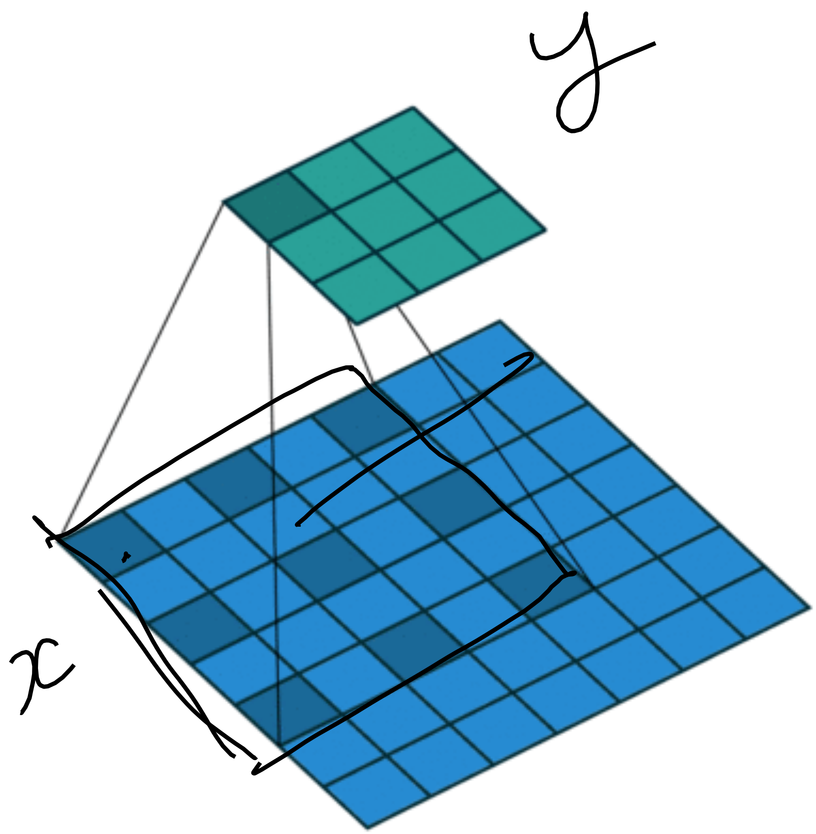
5×5

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

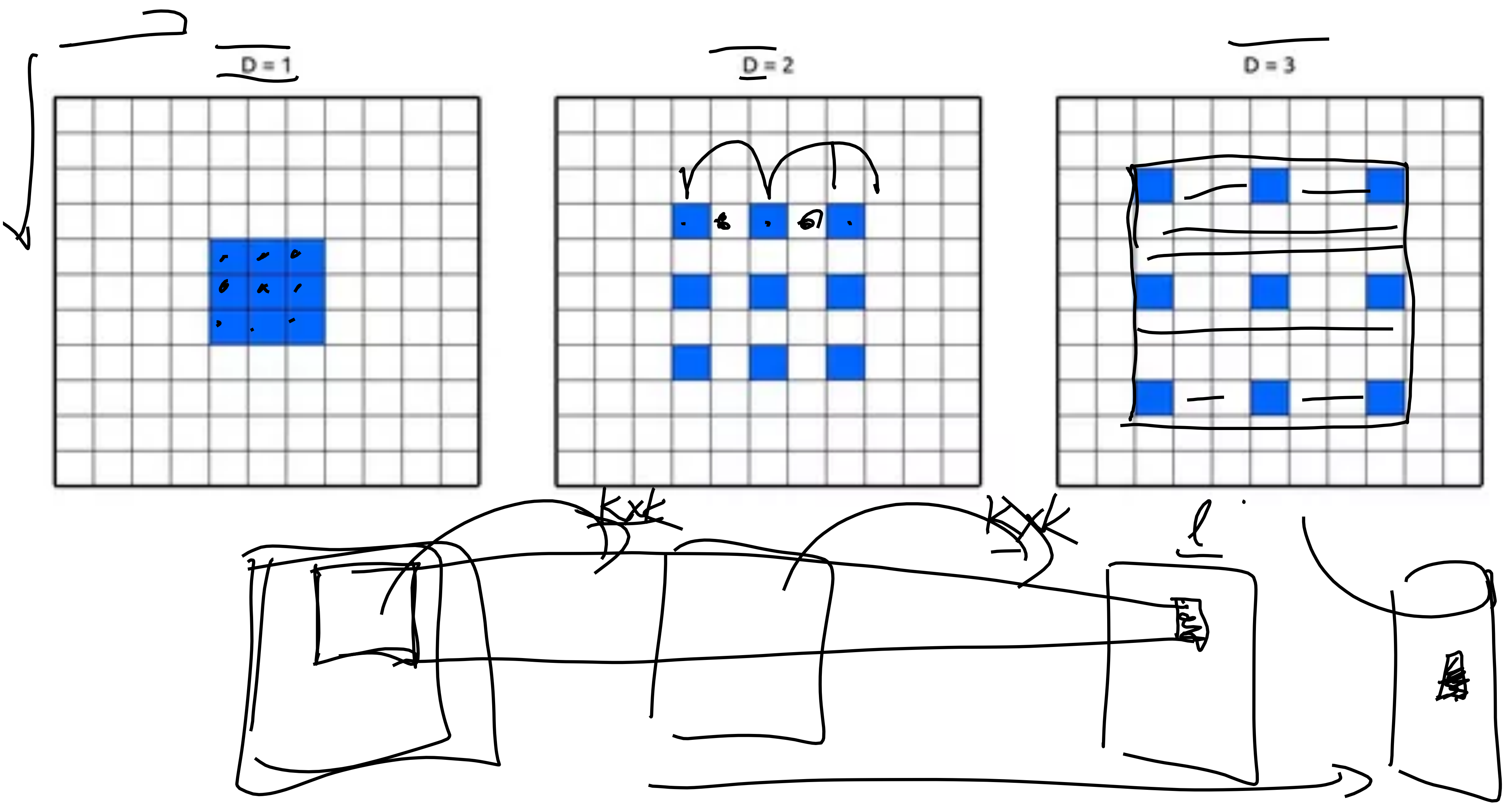
5×5

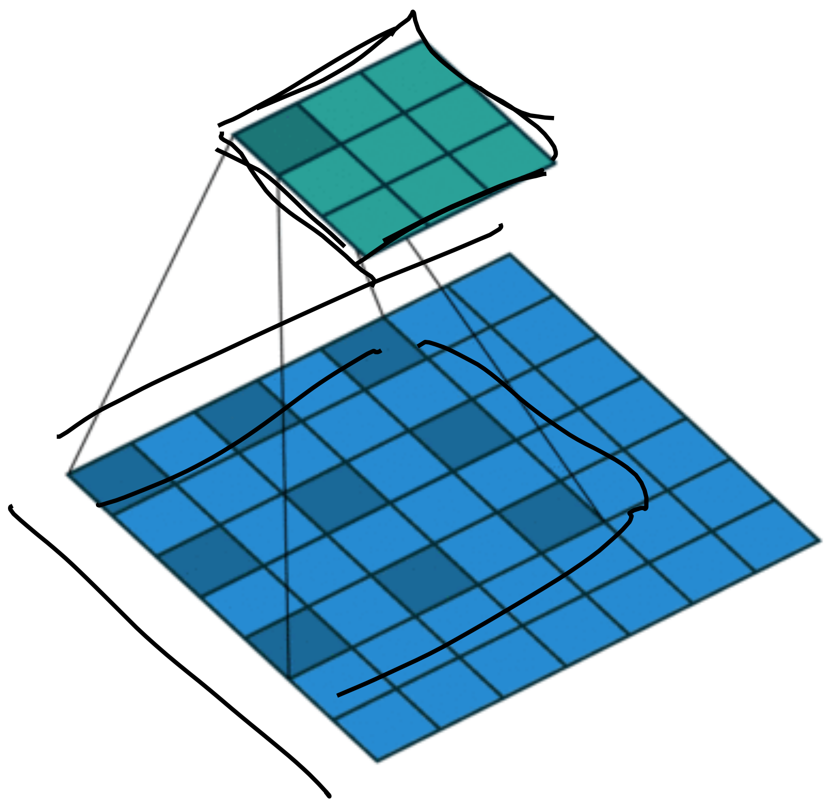
$\boxed{d = 2}$ $(d-1)$

3×3



$k \times k \rightarrow \left[k + (k-1)(d-1)\right] \times \left[\underline{\quad 3 \times 3 \quad}\right]$





(recall:)

(N - k) / stride + 1

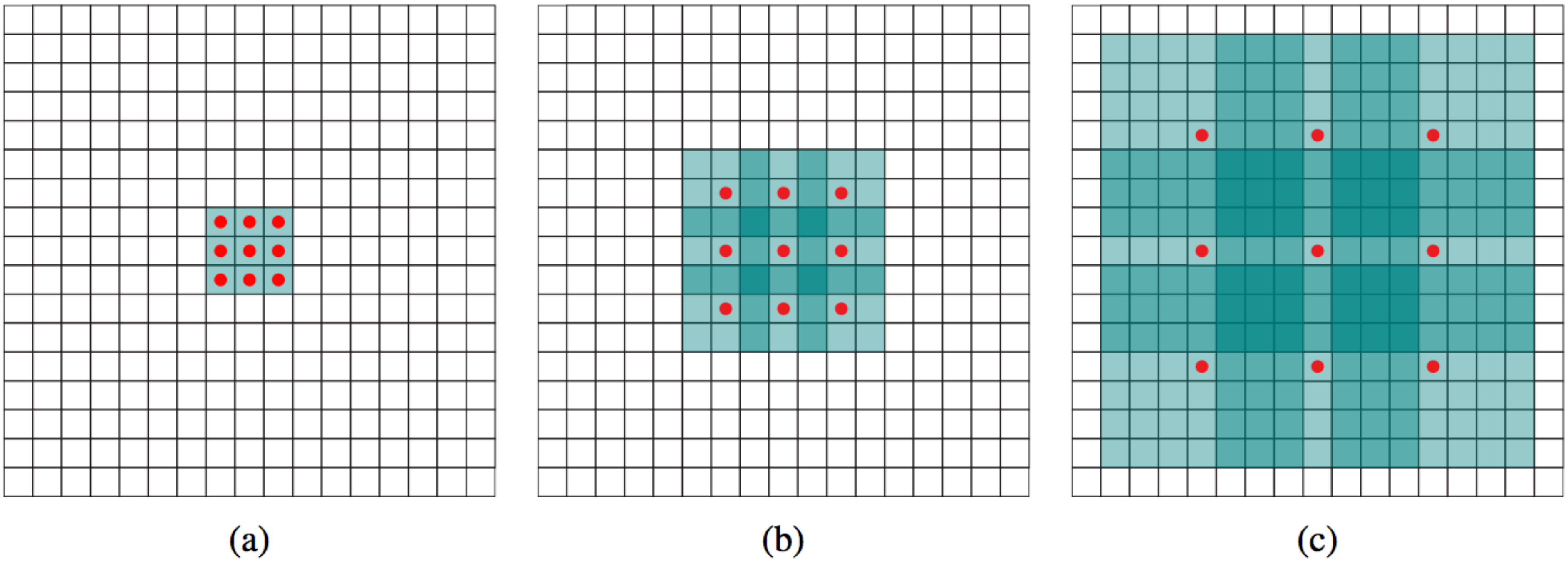


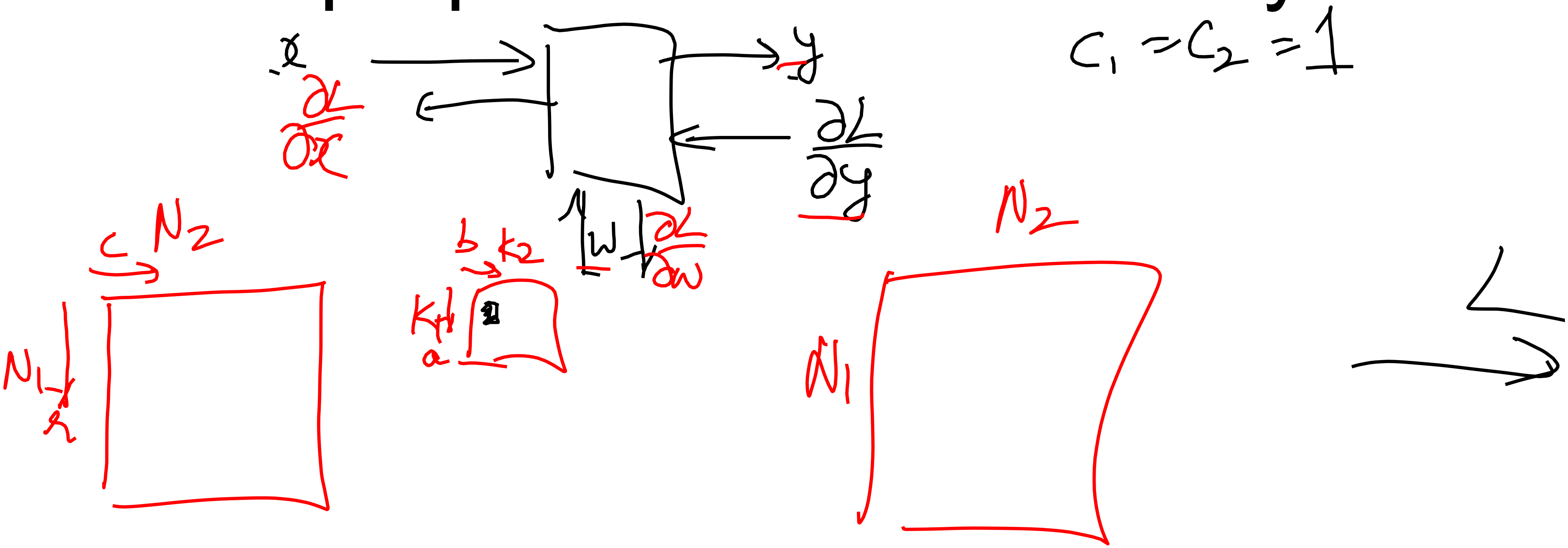
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element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

Figure Credit: Yu and Koltun, ICLR16

Plan for Today

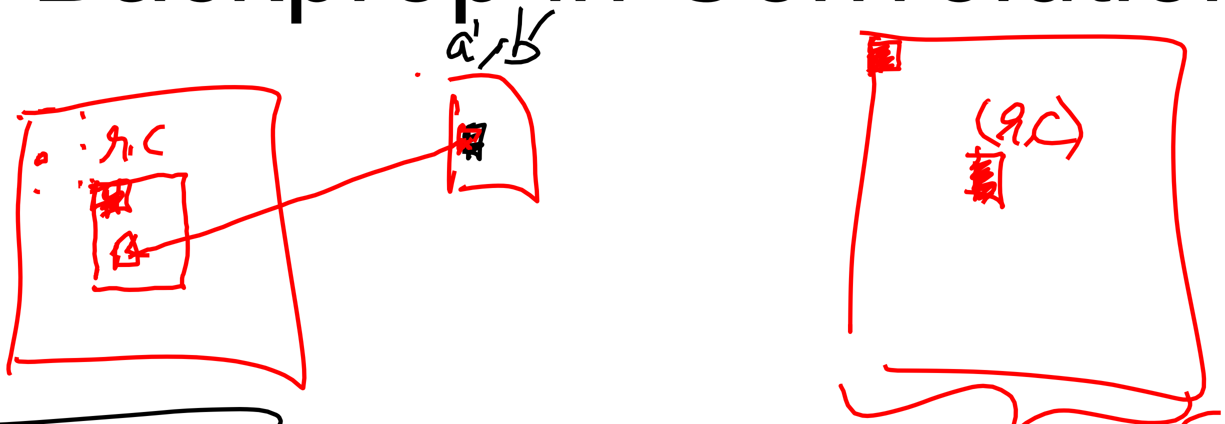
- Convolutional Neural Networks
  - Toeplitz matrices and convolutions = matrix-mult
  - Dilated/a-trous convolutions
  - Backprop in conv layers
  - Transposed convolutions

Backprop in Convolutional Layers



$$\frac{\partial L}{\partial w[a', b']} \qquad \frac{\partial L}{\partial x[r', c']}$$

Backprop in Convolutional Layers



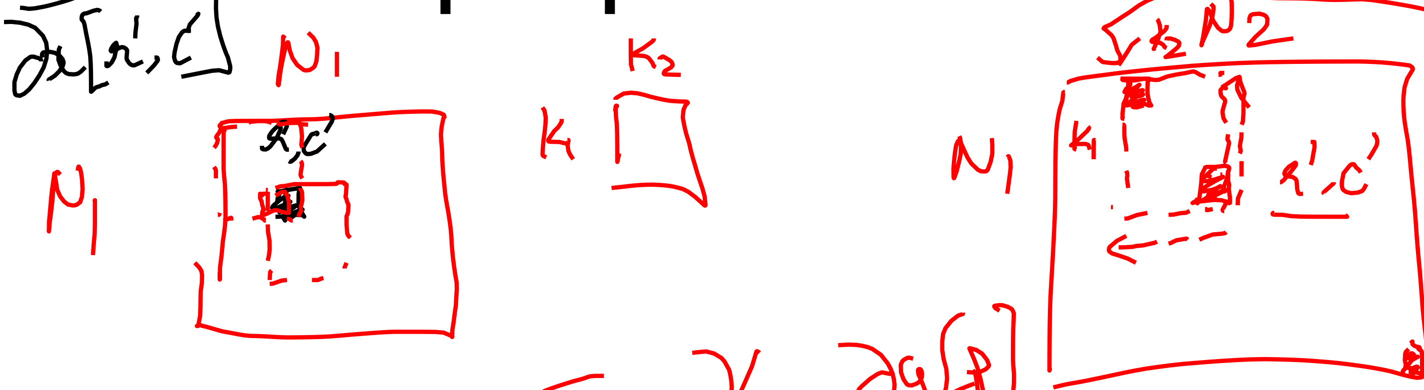
$$\frac{\partial L}{\partial w[a',b']} = \sum_{\substack{\text{pixels}\\ \text{in } y\\ \text{affected}}} \frac{\partial L}{\partial y[p]} \underbrace{\frac{\partial y[p]}{\partial w[a',b']}}$$

$[K_1, K_2]$

$$= \sum_{r=0}^{N_1-1} \sum_{c=0}^{N_2-1} \underbrace{\frac{\partial L}{\partial y[r,c]}}_{\text{input}} \frac{\partial y[r,c]}{\partial w[a',b']}$$

$$\frac{\partial y[r,c]}{\partial w[a',b']} = x[r+a', c+b']$$

$\frac{\partial L}{\partial x[r',c']}$ Backprop in Convolutional Layers



$(r'-k_1+1,\ c'-k_2+1)$

$$\frac{\partial L}{\partial x[r',c']} = \sum_{\substack{\text{pixels}\\\text{affected}}} \frac{\partial L}{\partial y[p]} \frac{\partial y[p]}{\partial x[r',c']} \quad y$$

$$= \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y[r'-a,\ c'-b]} \frac{\partial y[r'-a,\ c'-b]}{\partial x[r',c']}$$

$$y[r'-a,\ c'-b] = \sum_{a'=0}^{k_1-1} \sum_{b'=0}^{k_2-1} x[r'+a'-a,\ c'+b'-b]\ \cancel{w}[a',b']$$

$w[a,b]$

Backprop in Convolutional Layers

$$\frac{\partial L}{\partial x[r', c']} = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} \frac{\partial L}{\partial y[r' + a, c' + b]} \, w[a, b]$$

180° $+a$ $+b$

$$\begin{bmatrix} w[1,1] & w[1,0] \\ w[0,1] & w[0,0] \end{bmatrix} \xleftarrow{180^\circ} \begin{bmatrix} w[0,0] & w[0,1] \\ w[1,0] & w[1,1] \end{bmatrix}$$

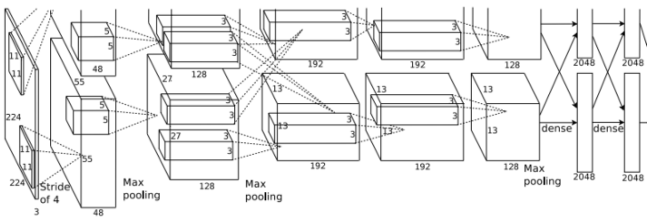
Plan for Today

- Convolutional Neural Networks
  - Toeplitz matrices and convolutions = matrix-mult
  - Dilated/a-trous convolutions
  - Backprop in conv layers
  - Transposed convolutions

Transposed Convolutions

- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

So far: Image Classification

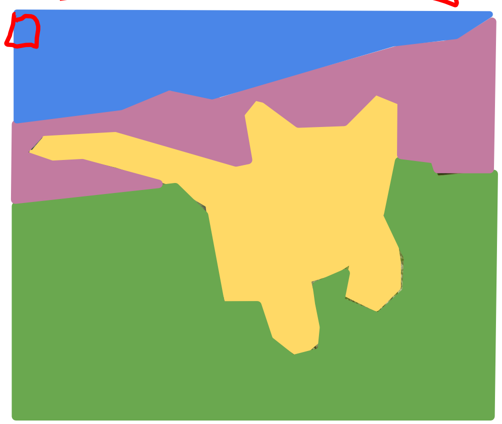
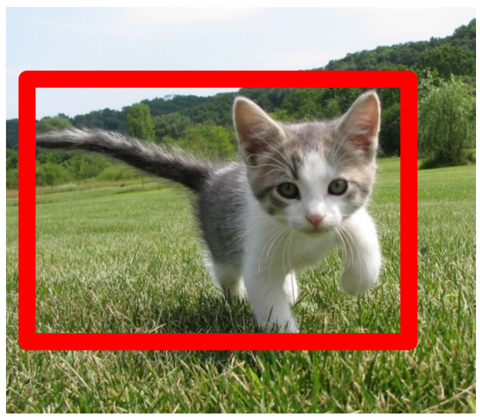
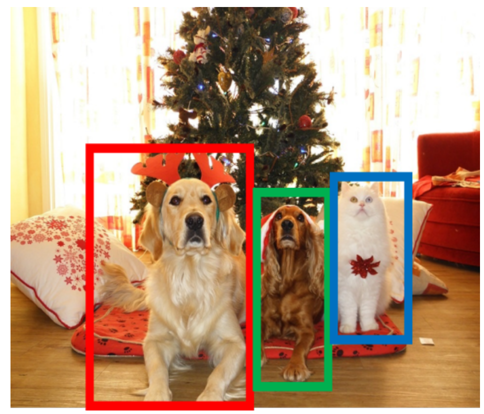
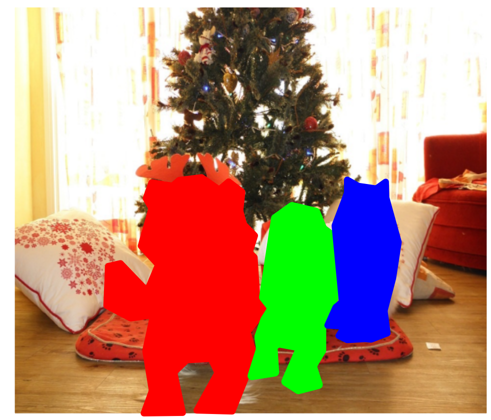


Vector: 4096

Fully-Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Other Computer Vision Tasks

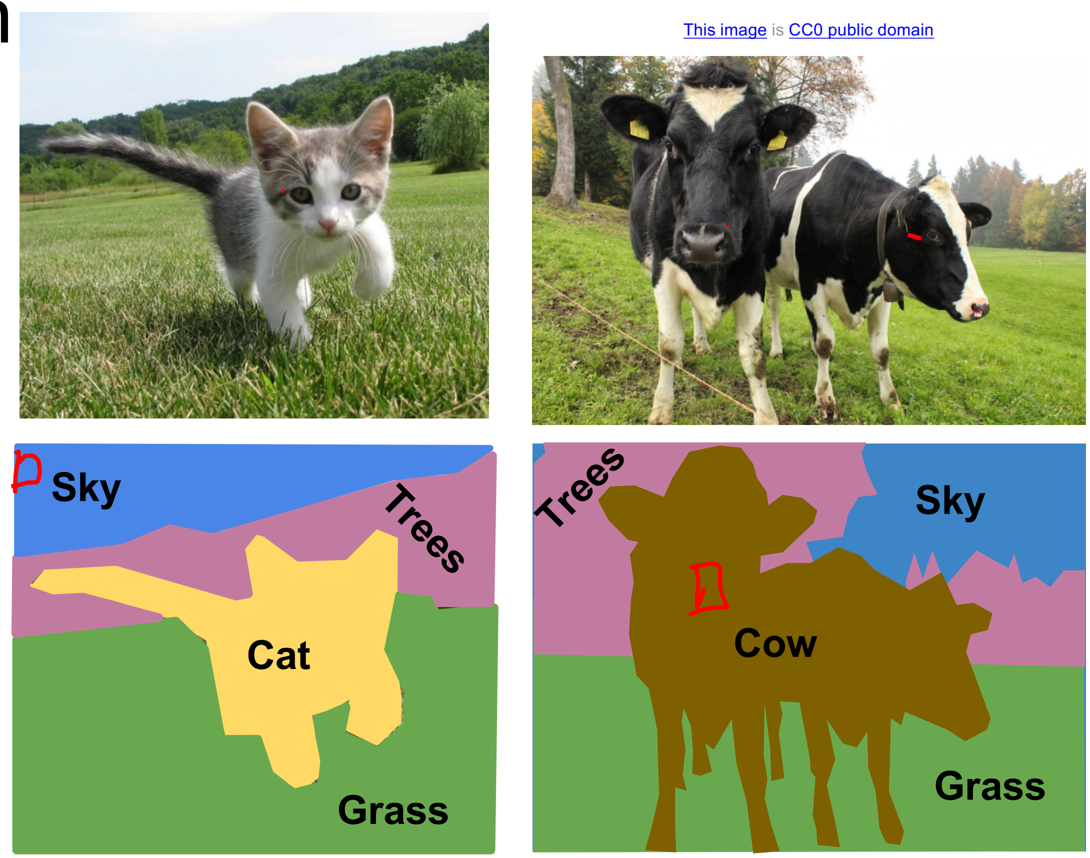
Semantic Segmentation	**Classification + Localization**	**Object Detection**	**Instance Segmentation**
			
GRASS, CAT, TREE, SKY	**CAT**	**DOG, DOG, CAT**	**DOG, DOG, CAT**
No objects, just pixels	Single Object	Multiple Object	

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

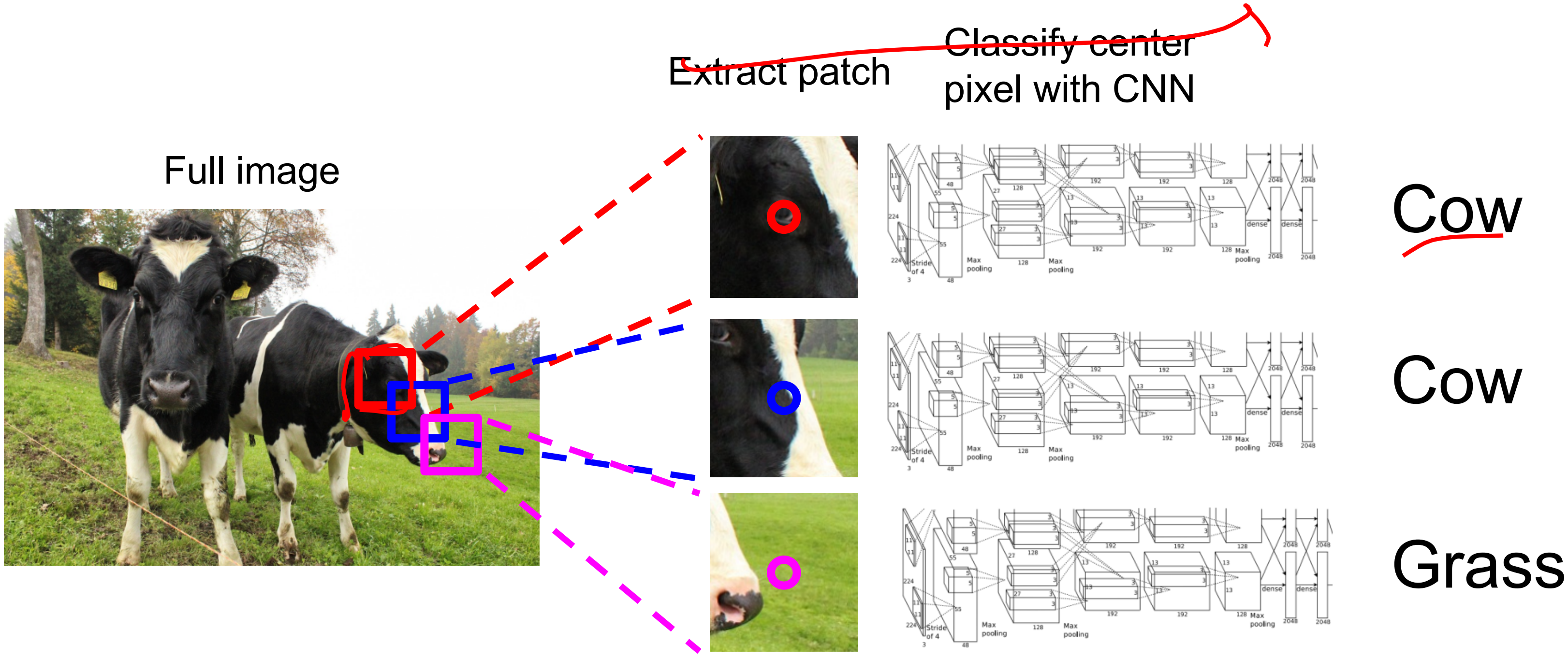
Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels



Semantic Segmentation Idea: Sliding Window



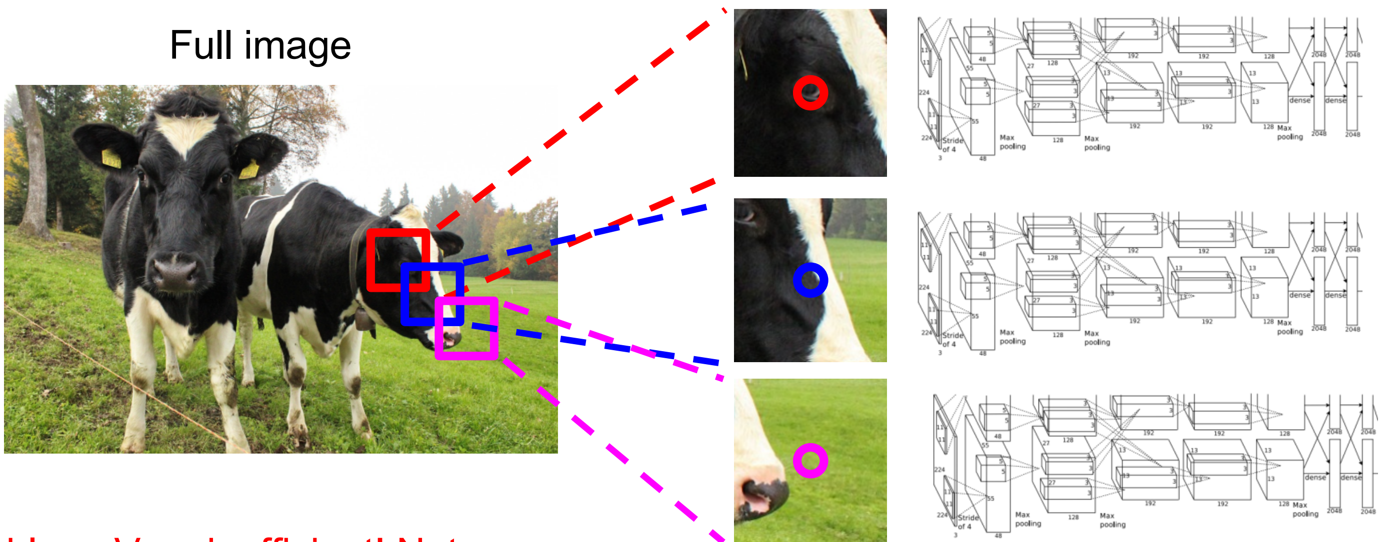
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Sliding Window

Full image

Extract patch

Classify center pixel with CNN



Cow

Cow

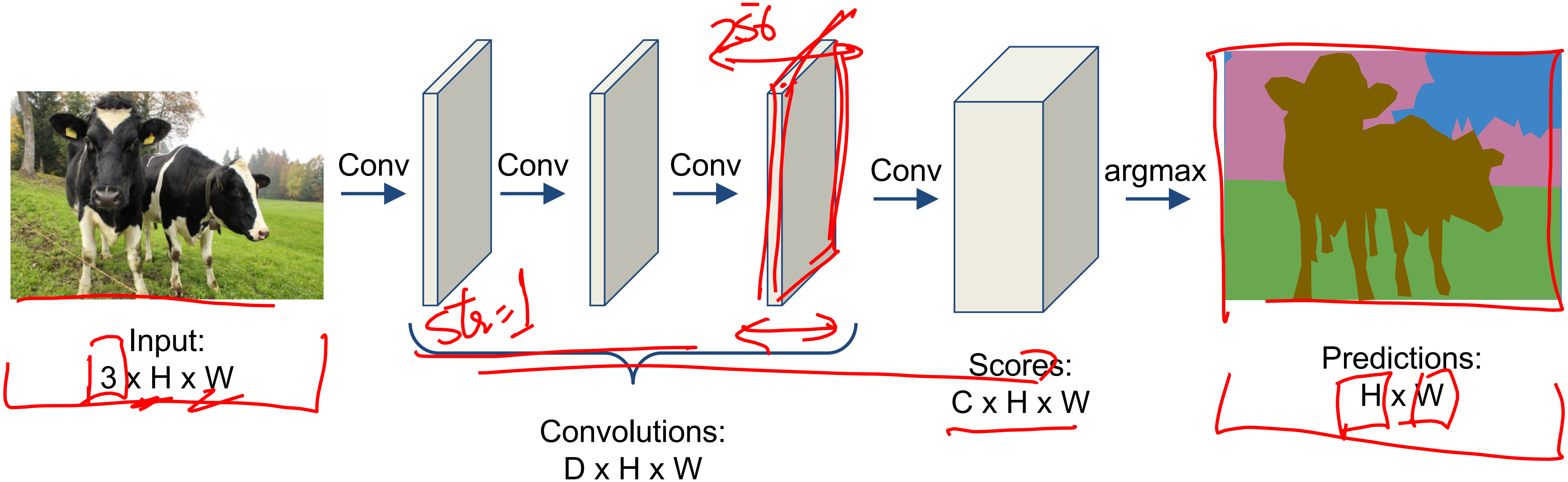
Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

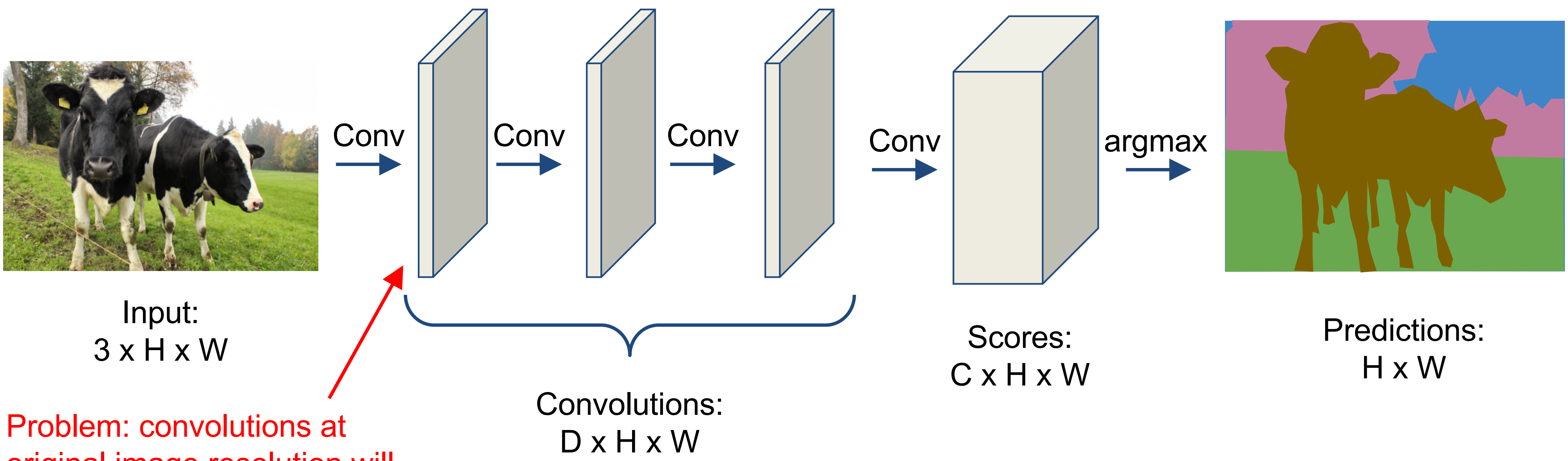
Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



Semantic Segmentation Idea: Fully Convolutional

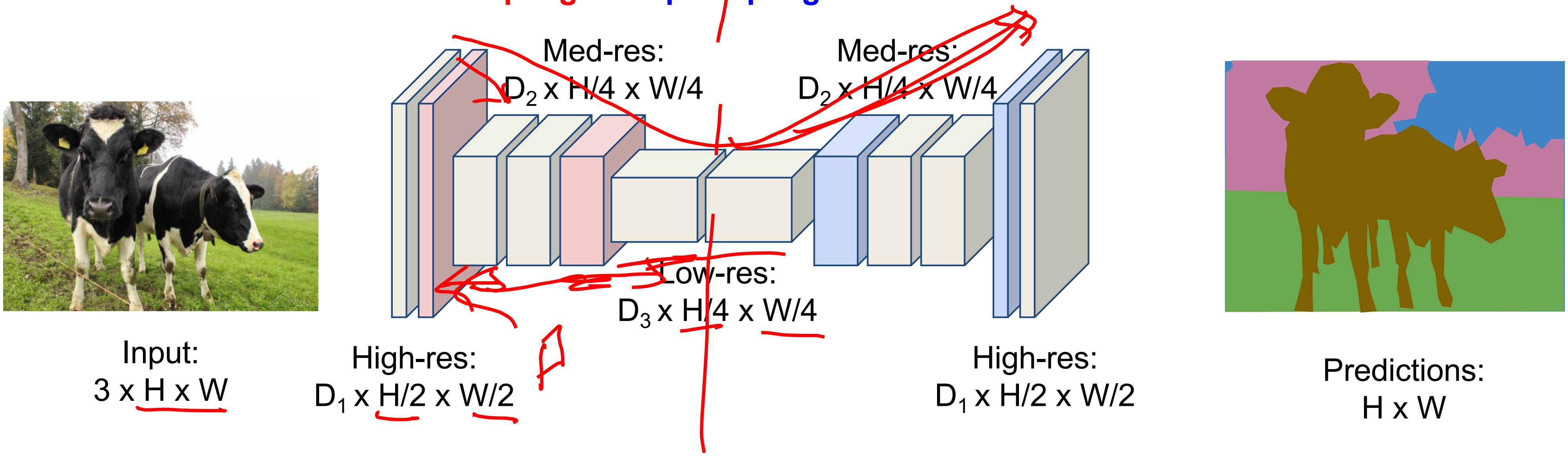
Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



Problem: convolutions at original image resolution will be very expensive ...

Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

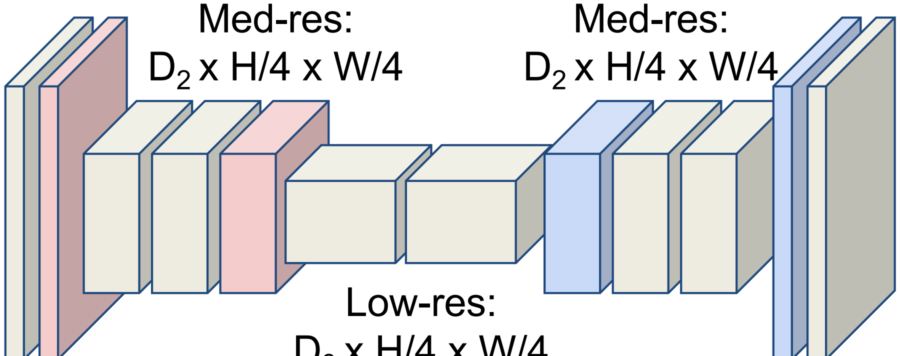
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided
convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
???

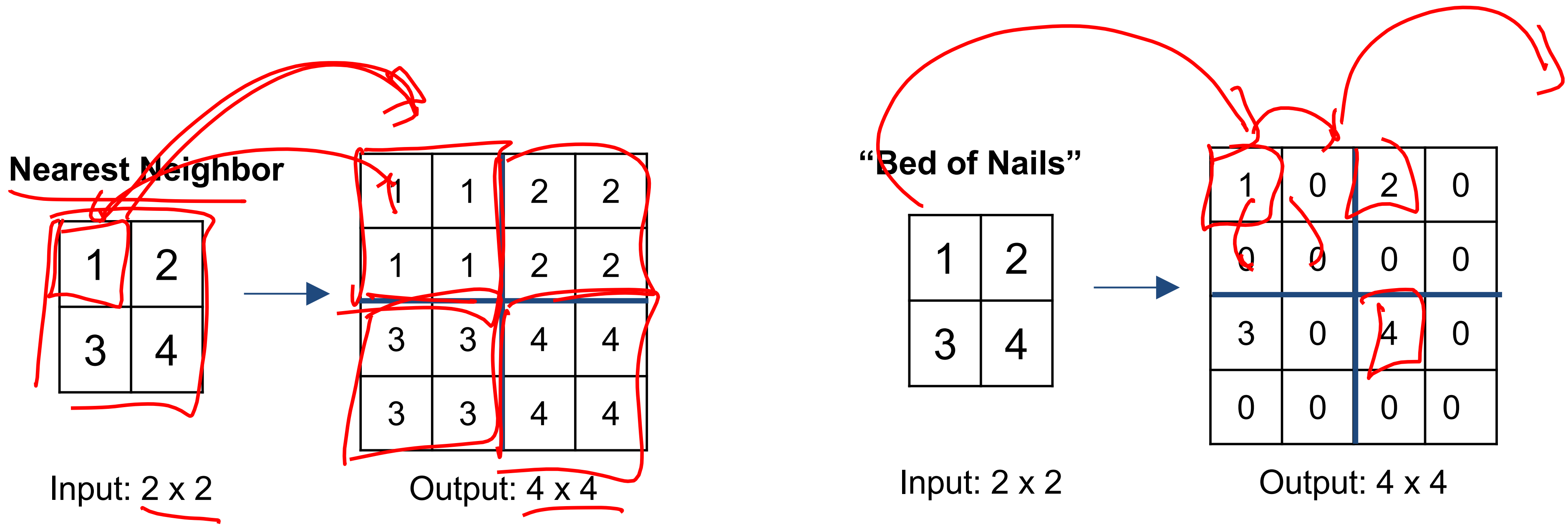
Input:
$3 \times H \times W$



Predictions:
$H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

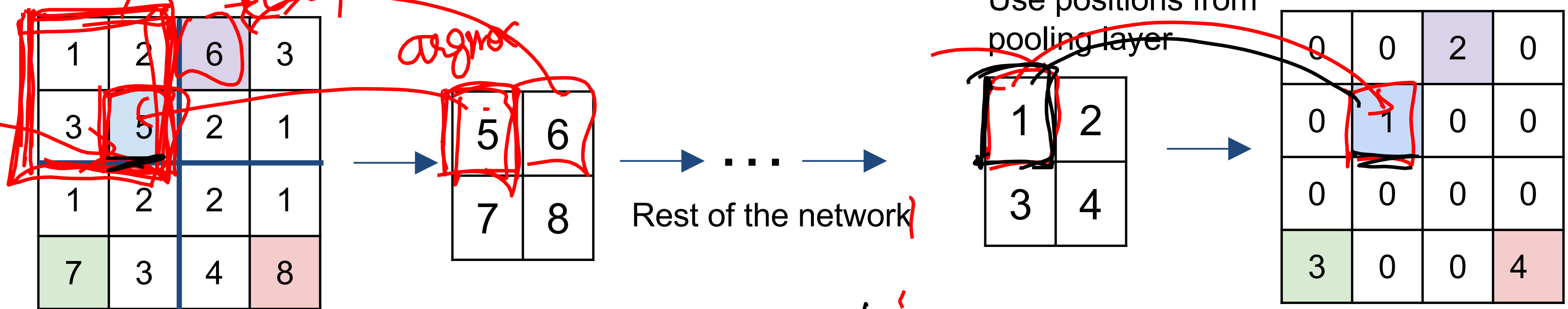
In-Network upsampling: “Unpooling”



In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

Max Unpooling
Use positions from pooling layer



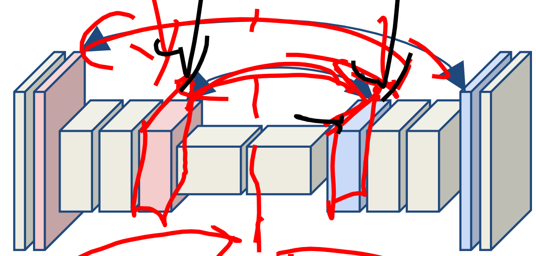
Input: 4 x 4

Output: 2 x 2

Input: 2 x 2

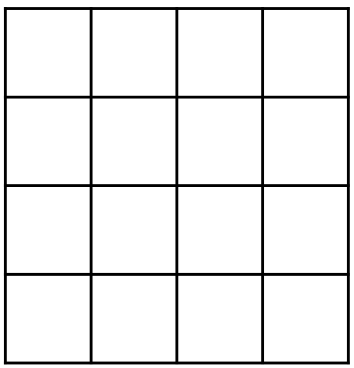
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

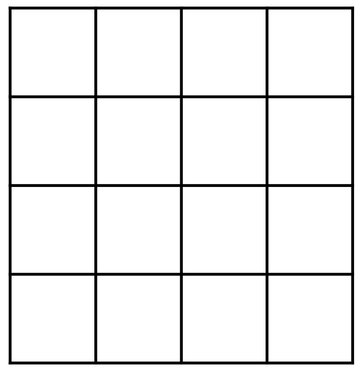


Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, stride 1 pad 1



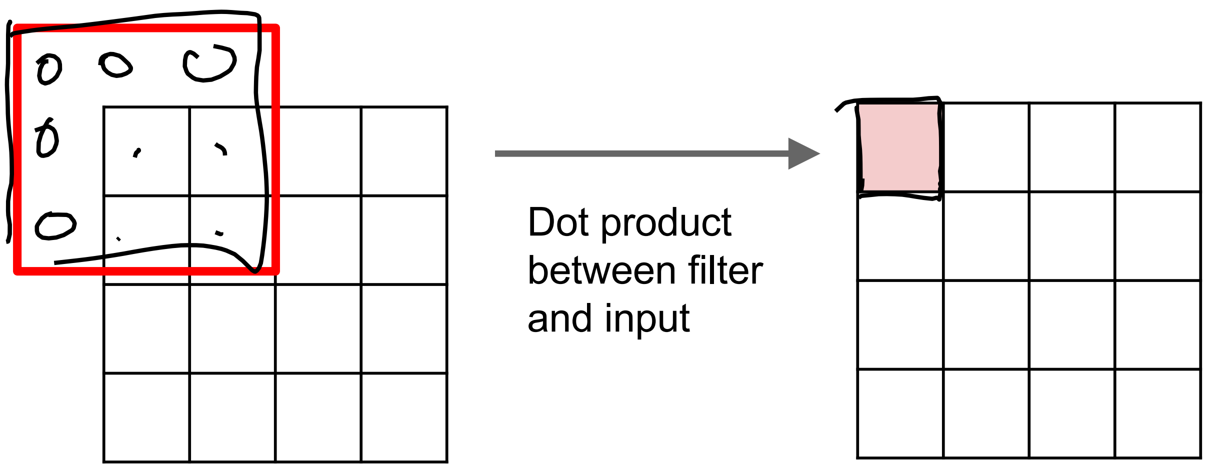
Input: 4 x 4



Output: 4 x 4

Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

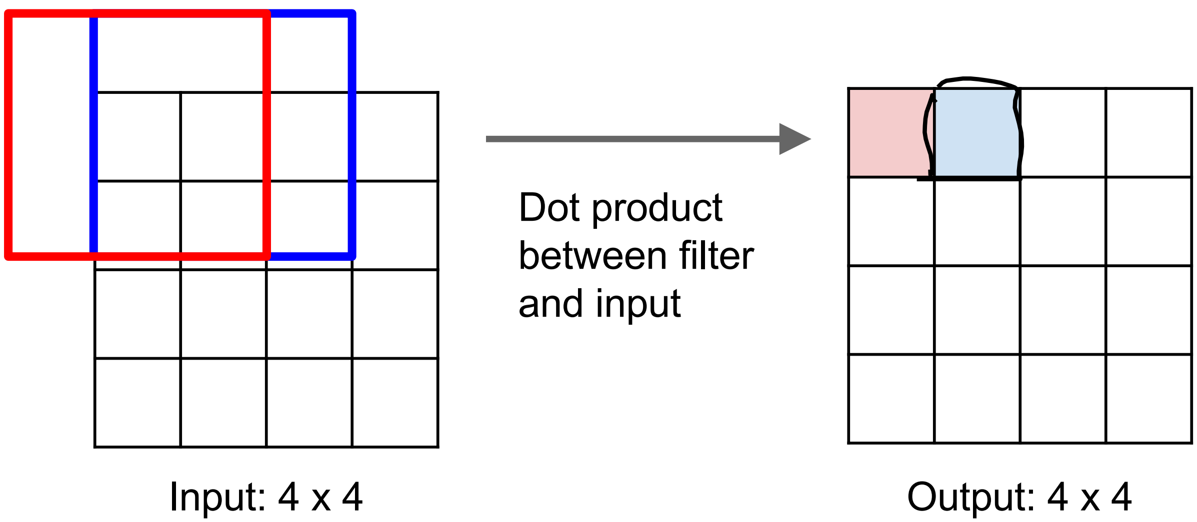


Input: 4 x 4

Output: 4 x 4

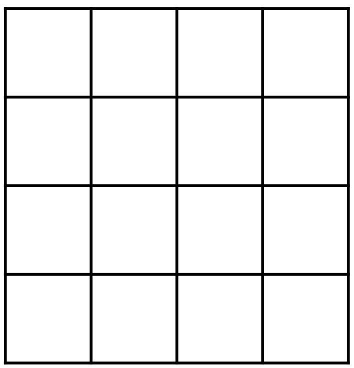
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

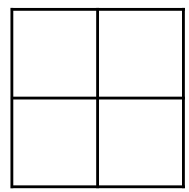


Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

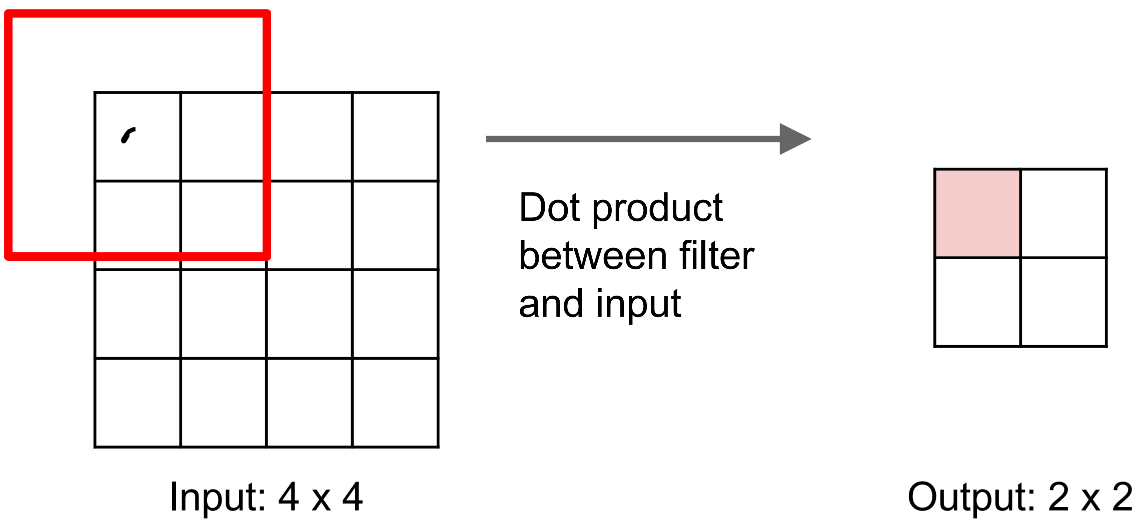


Output: 2 x 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

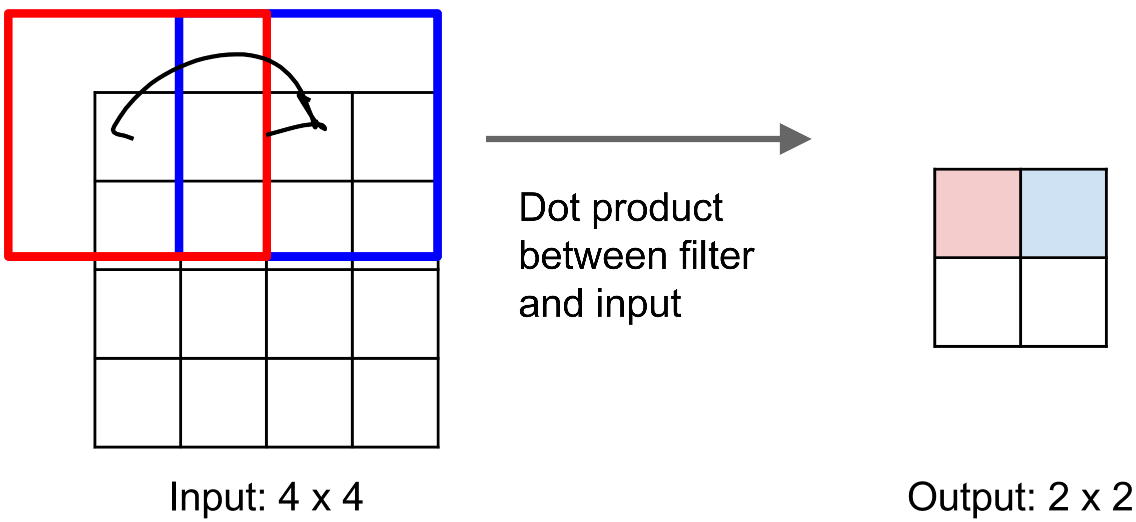
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1

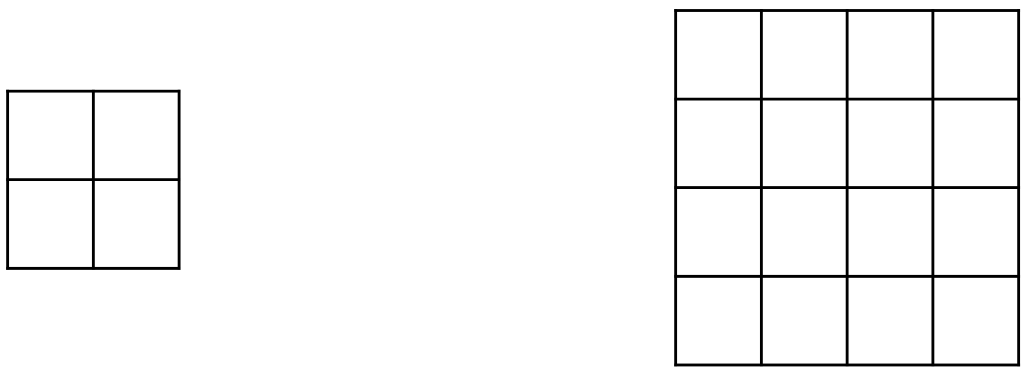


Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



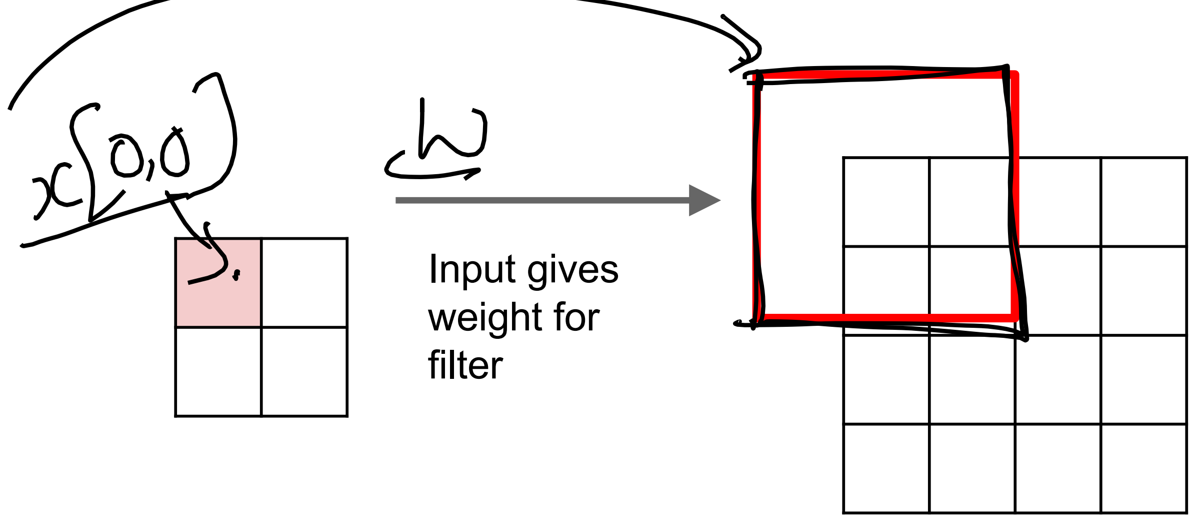
Input: 2 x 2

Output: 4 x 4

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

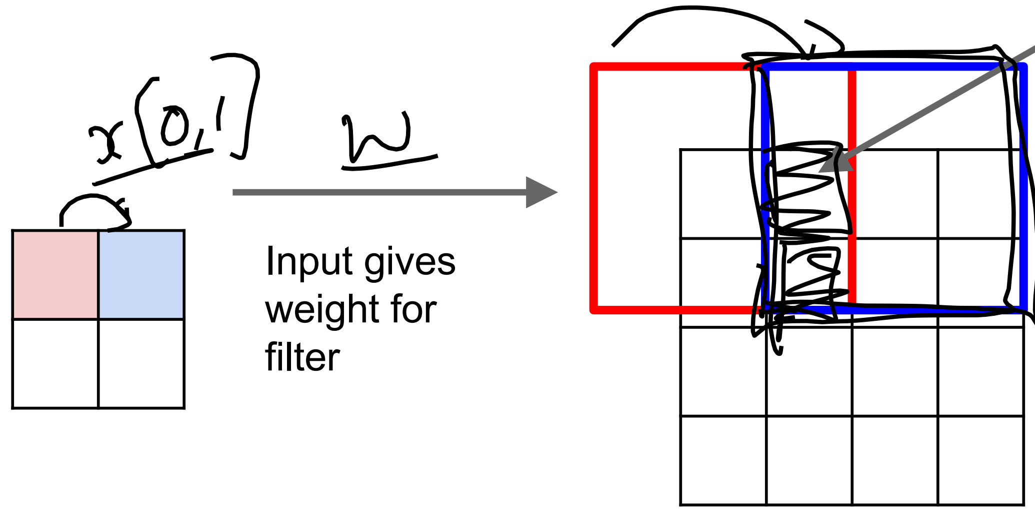


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



Input: 2 x 2

Output: 4 x 4

Sum where output overlaps

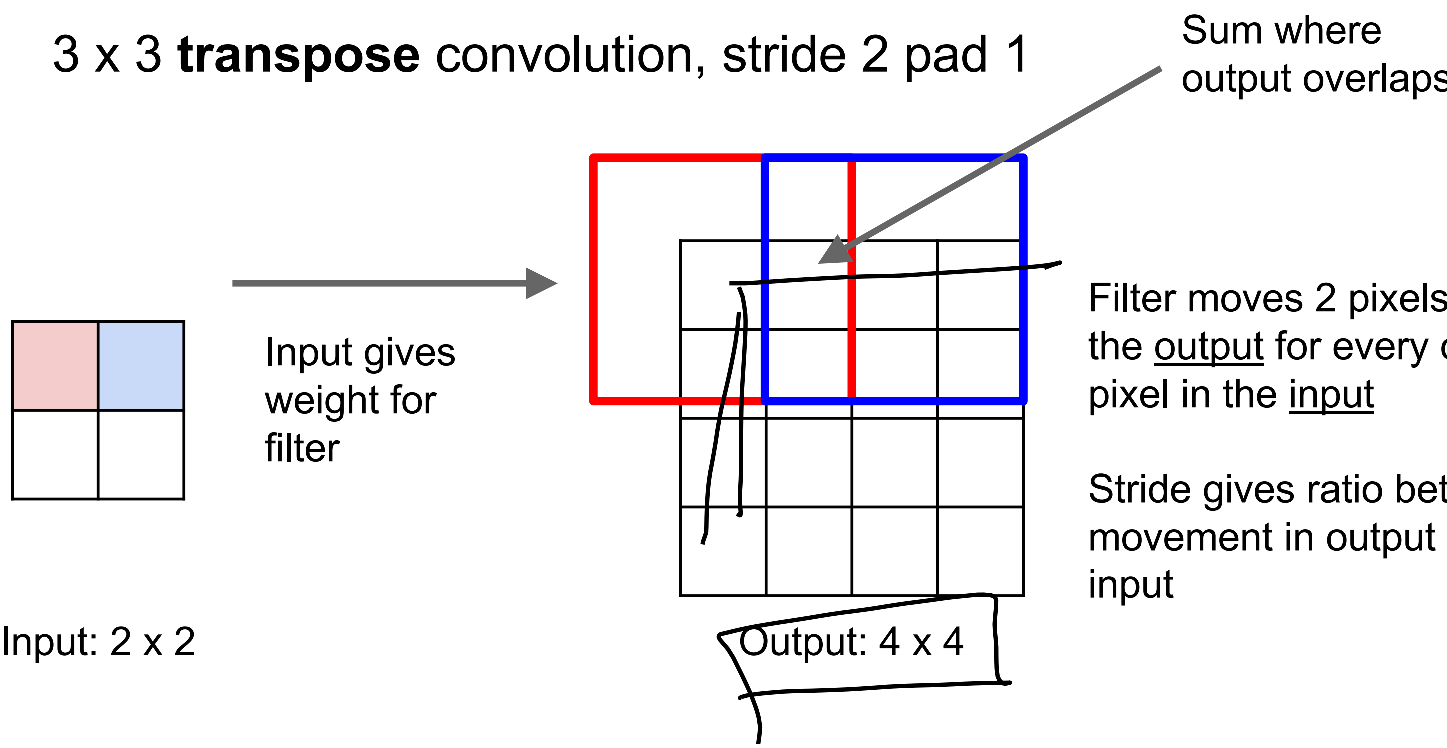
Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Learnable Upsampling: Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

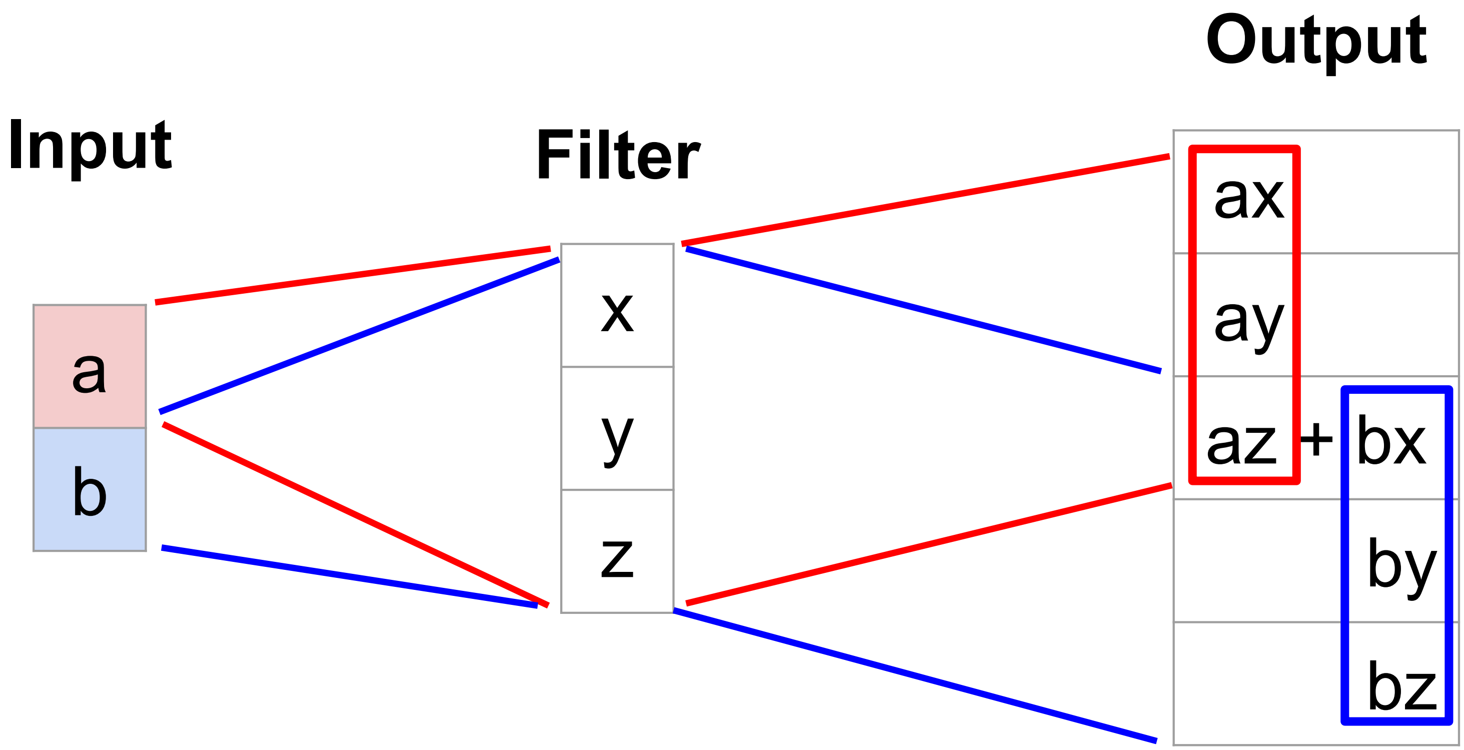
3 x 3 **transpose** convolution, stride 2 pad 1



Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

Transpose Convolution: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

Transposed Convolution

- https://distill.pub/2016/deconv-checkerboard/