CS 4803 / 7643: Deep Learning

Topics:

- Convolutional Neural Networks
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers [Derived in notes]
  - Toeplitz matrices and convolutions = matrix-mult

Dhruv Batra

Georgia Tech

Administrativia

- HW2 Reminder
  - Due: 09/23, 11:59pm
  - https://evalai.cloudcv.org/web/challenges/challenge-page/684/leaderboard/1853
- Project Teams
  - https://gtvault-my.sharepoint.com/:x:/g/personal/dbatra8_gatech_edu/EY4_65XOzWtOkXSSz2WgpoUBY8ux2gY9PsRzR6KnglIFEQ?e=4tnKWI
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names

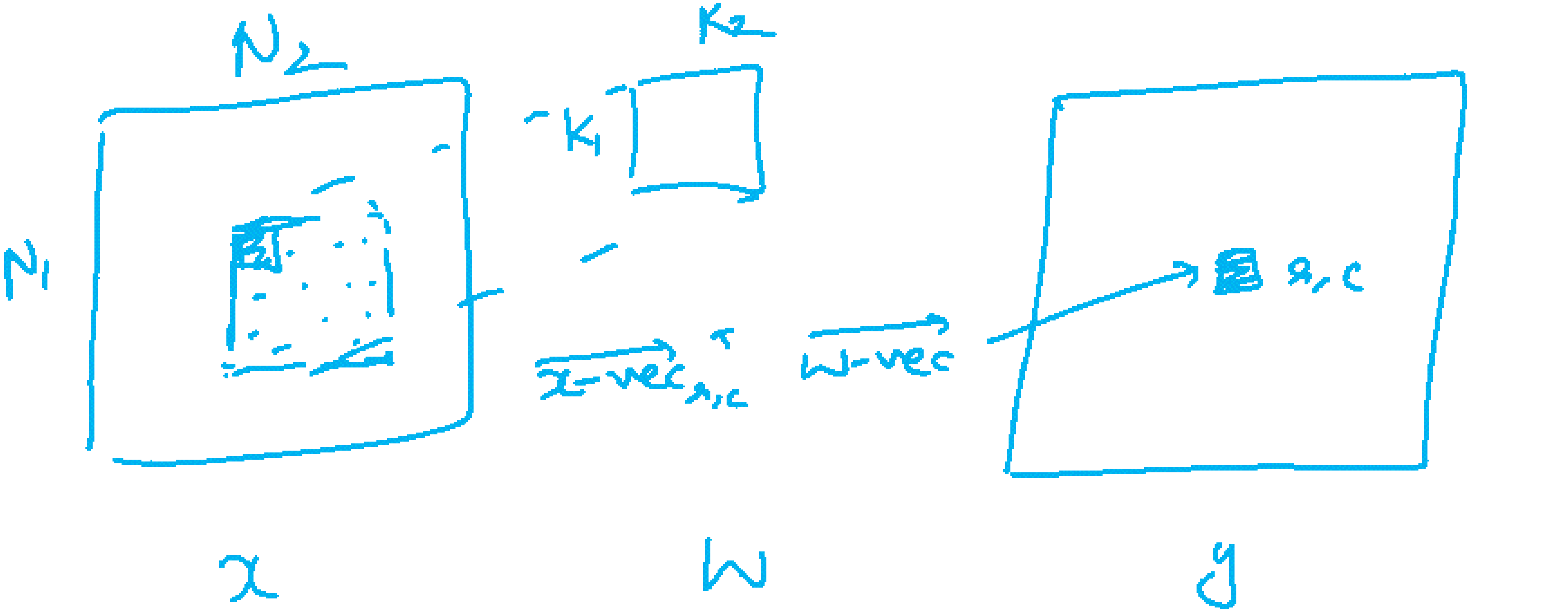
Recap from last time

Convolutions for programmers

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

rows

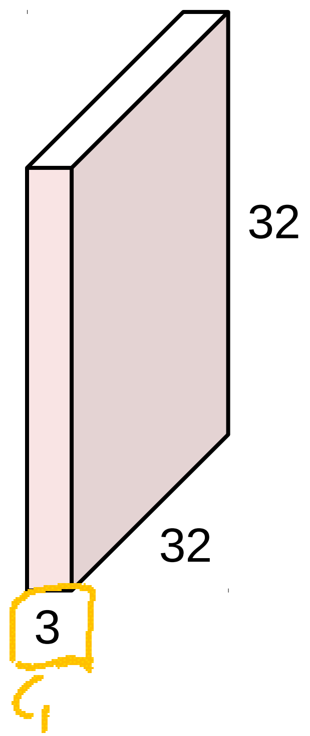
cols



Convolution Layer

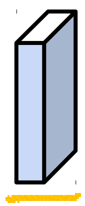
Filters always extend the full depth of the input volume

32x32x3 image



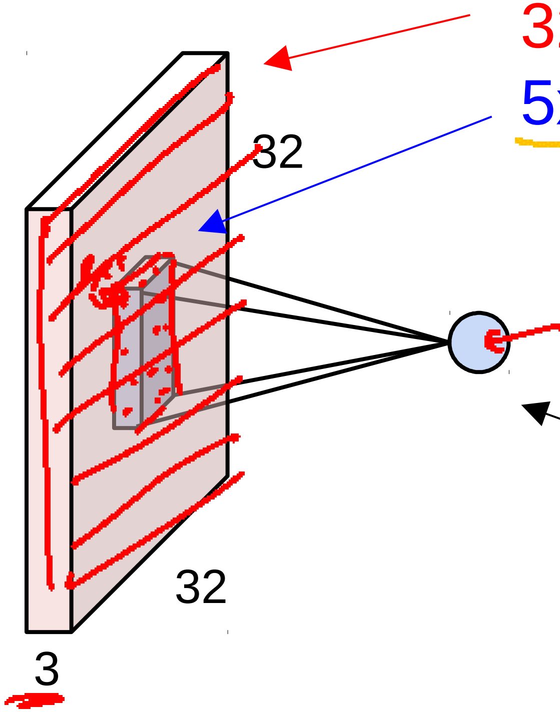
5x5x3 filter

c_1



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

Convolution Layer



32x32x3 image

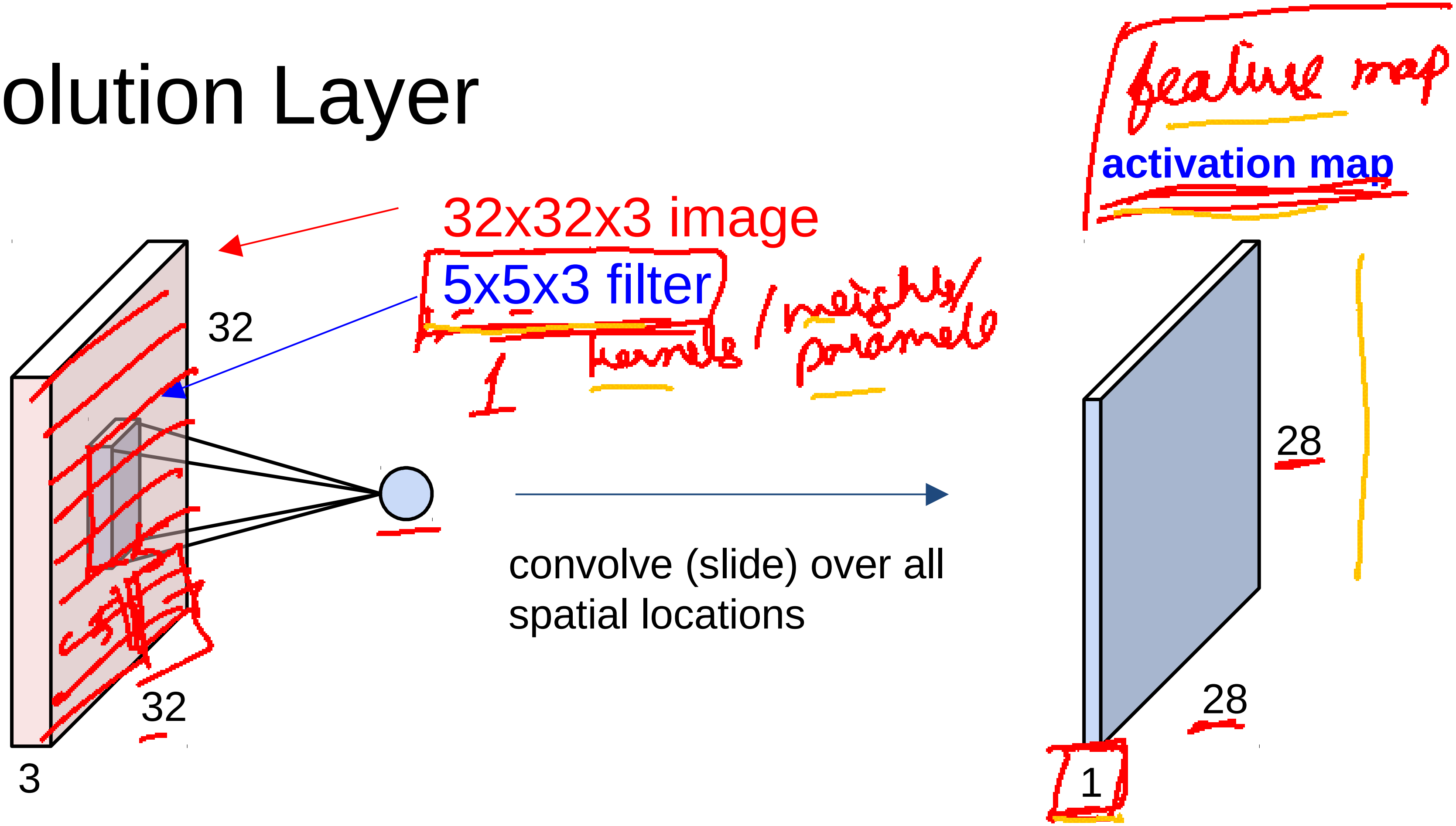
5x5x3 filter w

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)

$$w^T \boxed{x} + b$$

w-vec x-vec

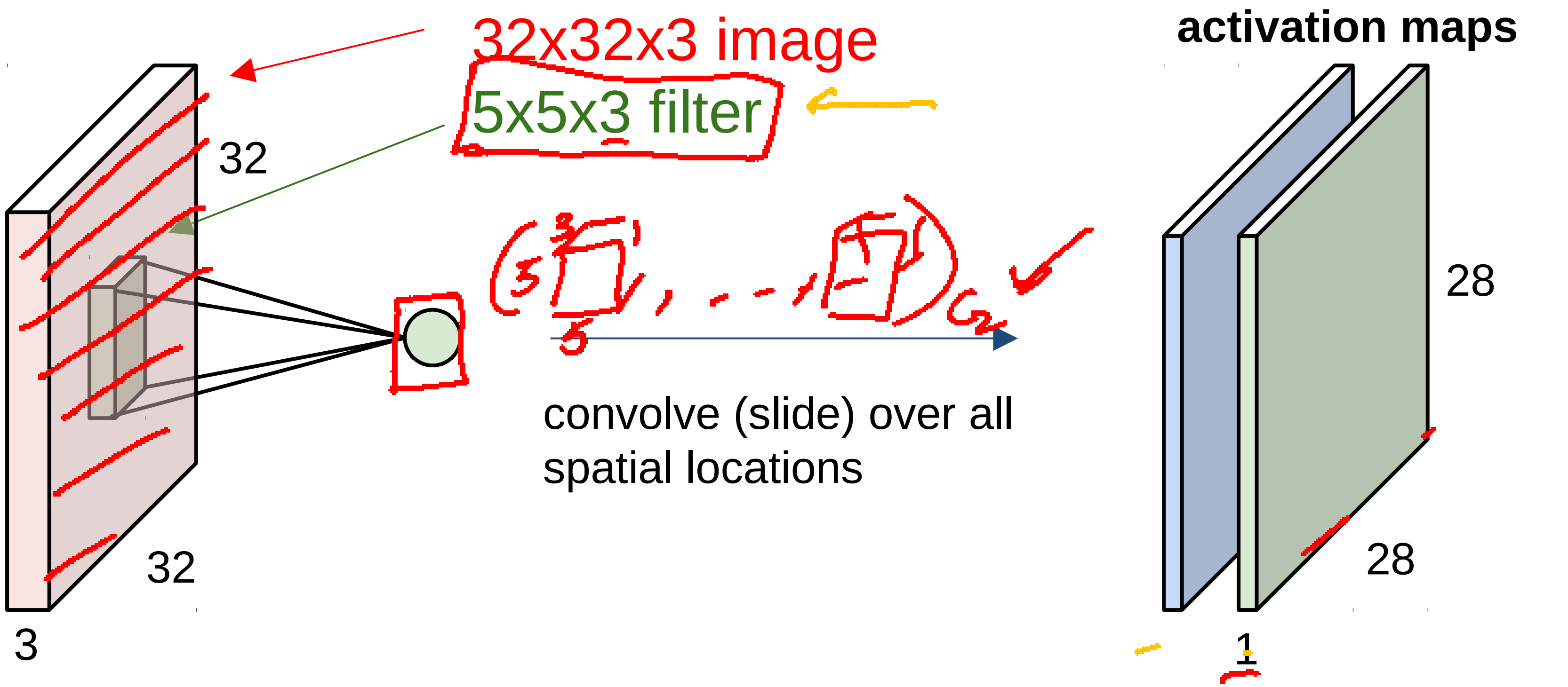
Convolution Layer



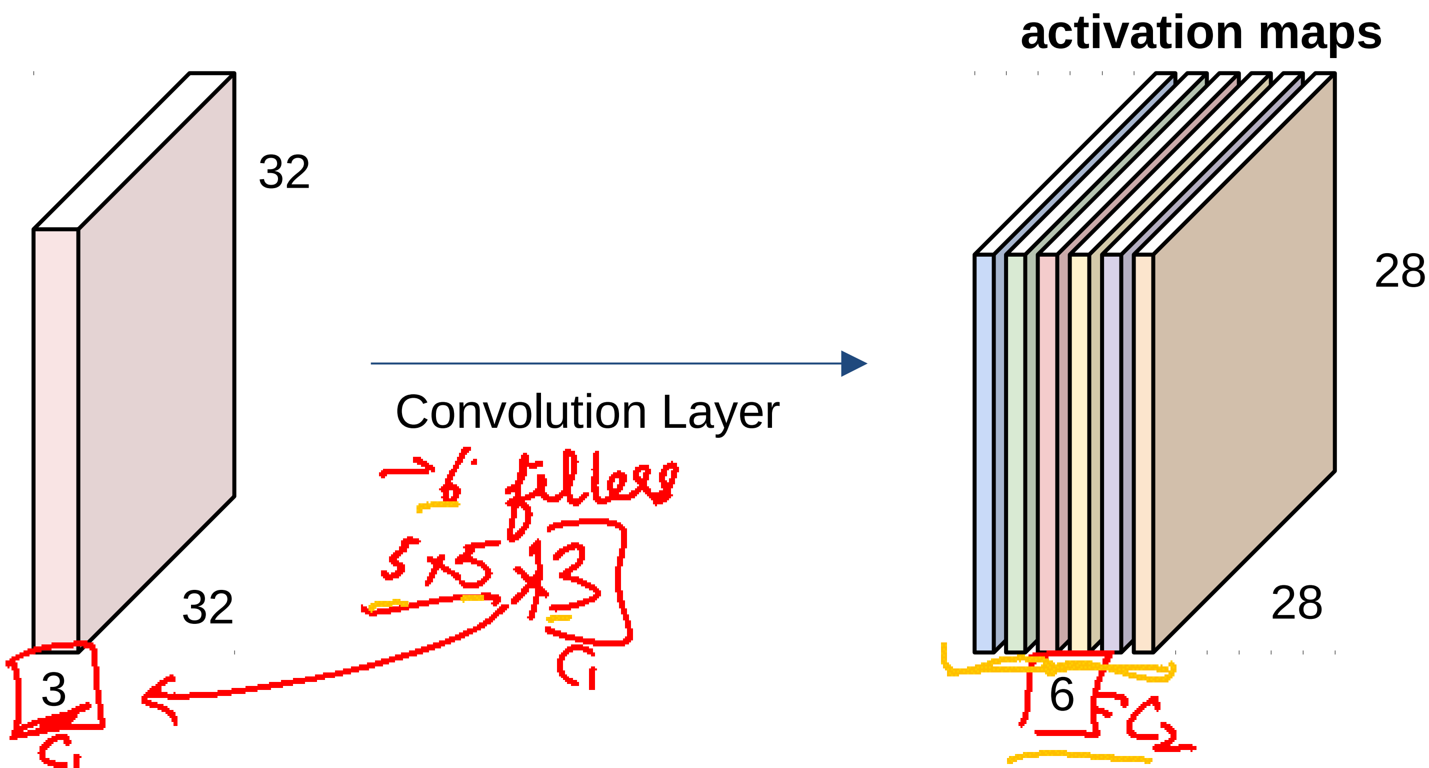
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Convolution Layer

consider a second, green filter



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

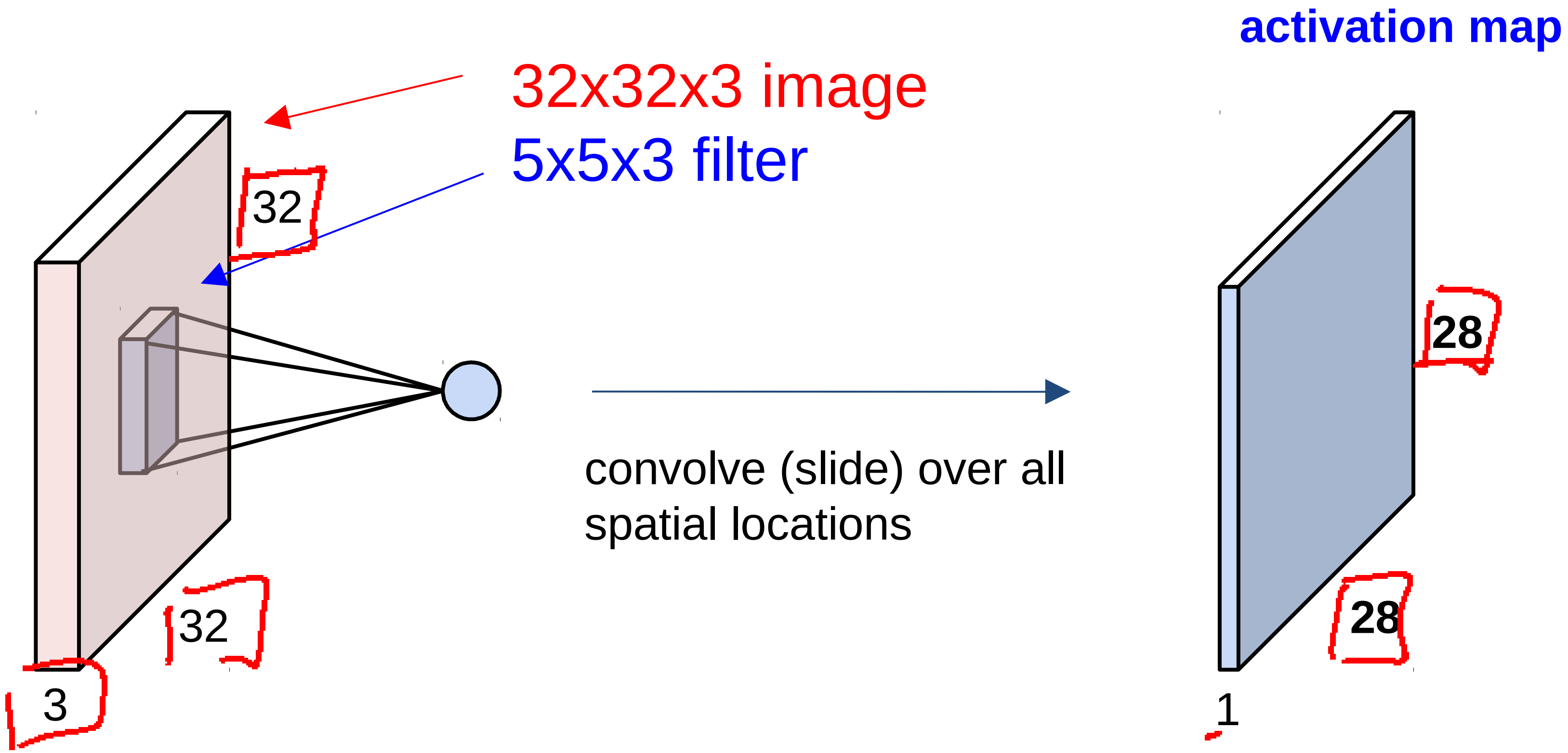


We stack these up to get a "new image" of size 28x28x6!

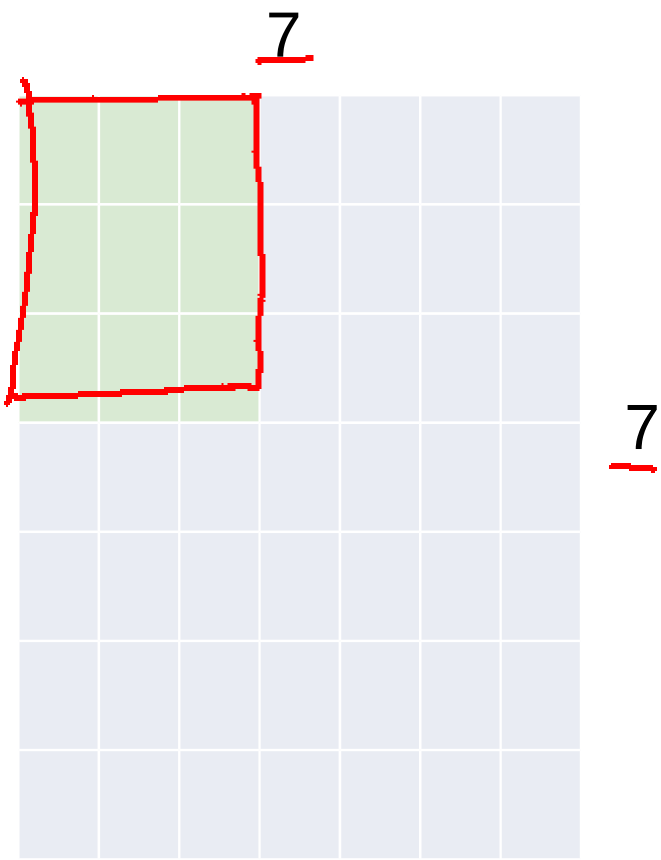
Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult

A closer look at spatial dimensions:

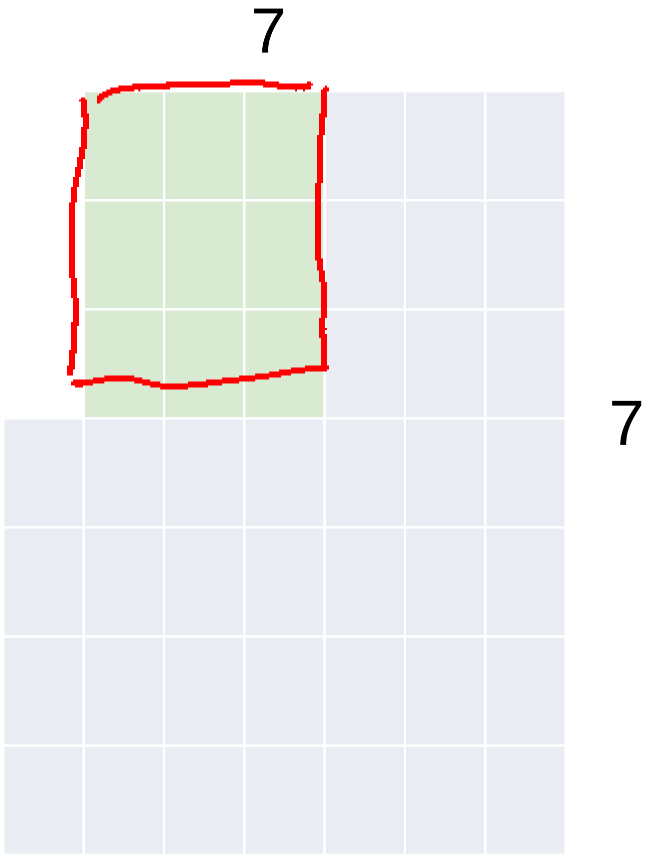


A closer look at spatial dimensions:



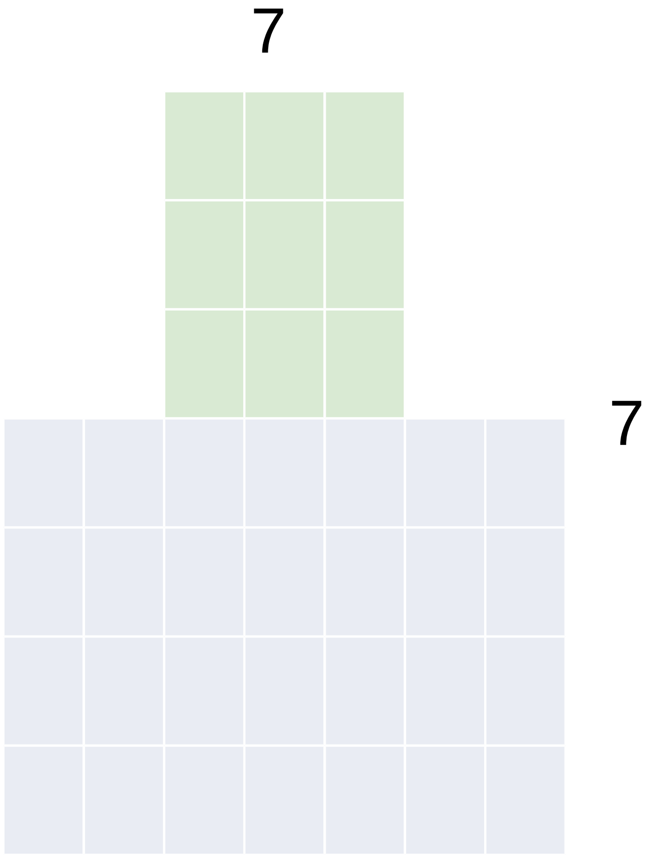
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



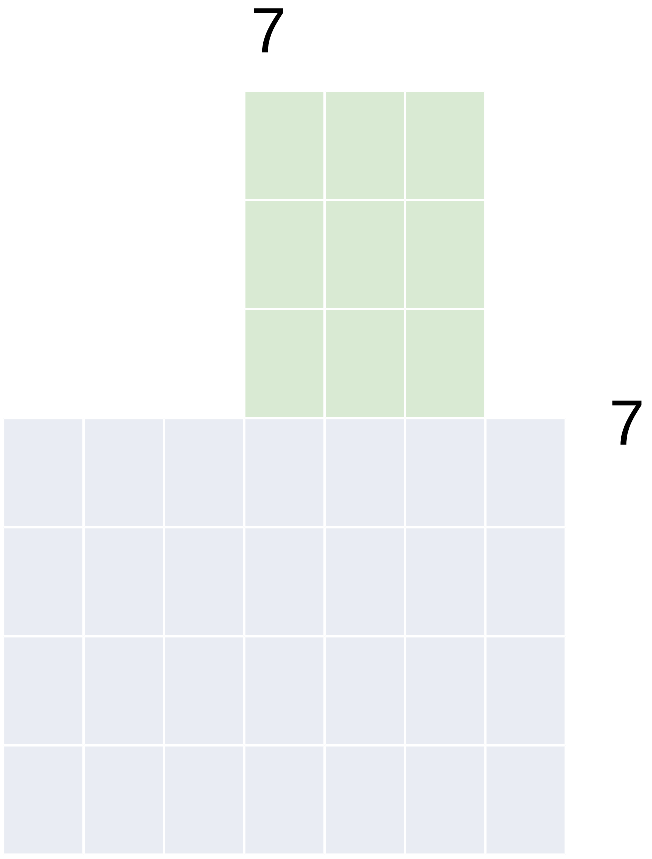
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

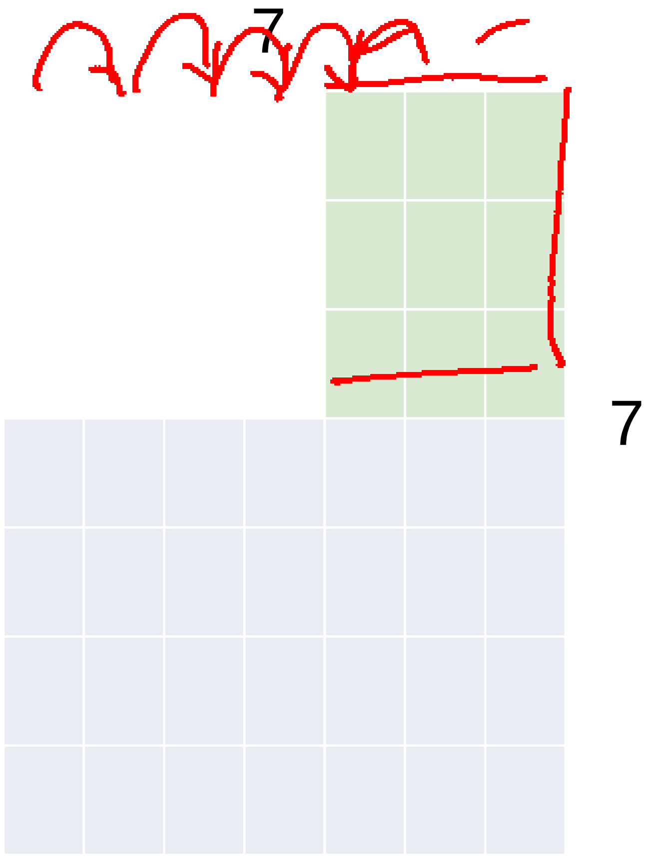
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

stride = 1

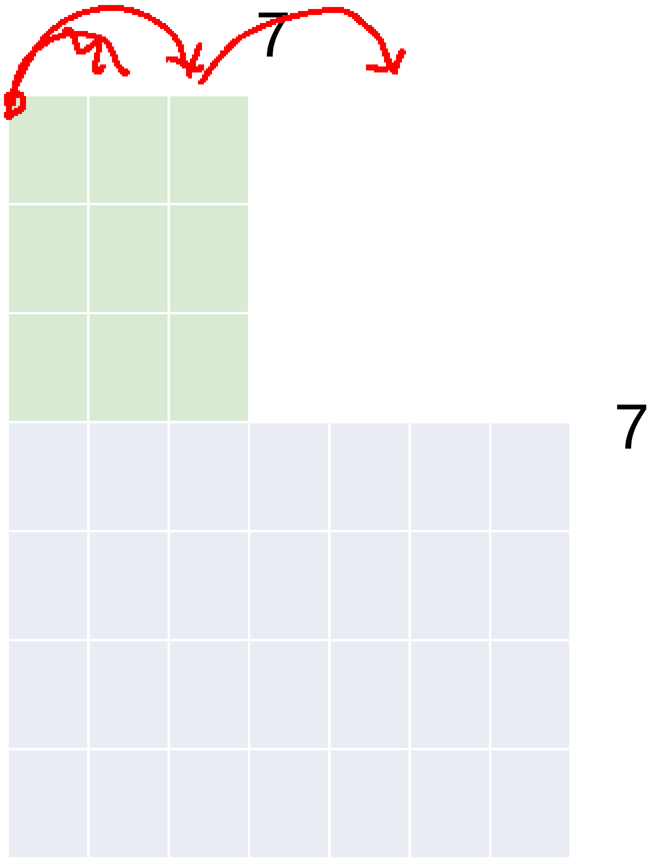
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

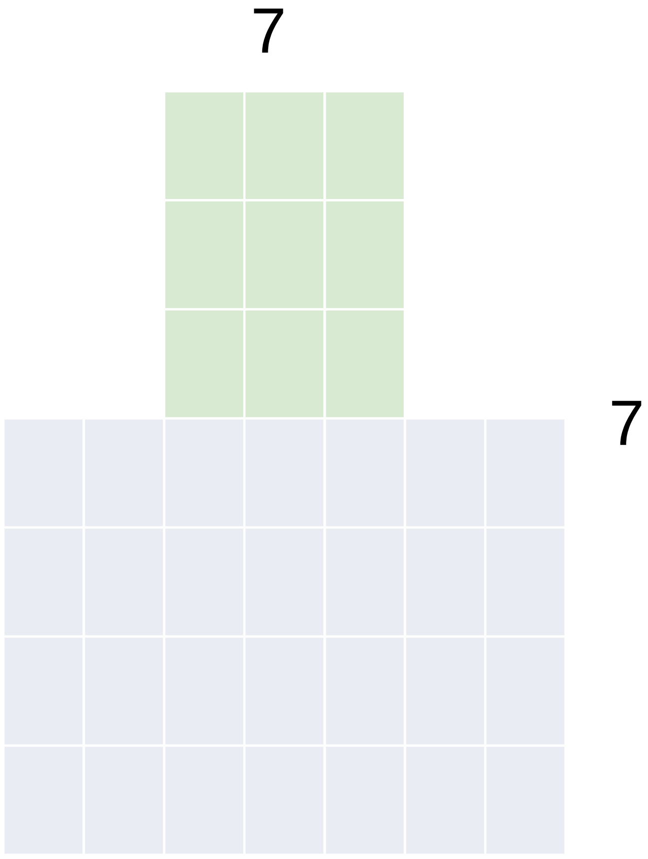
=> 5x5 output

A closer look at spatial dimensions:



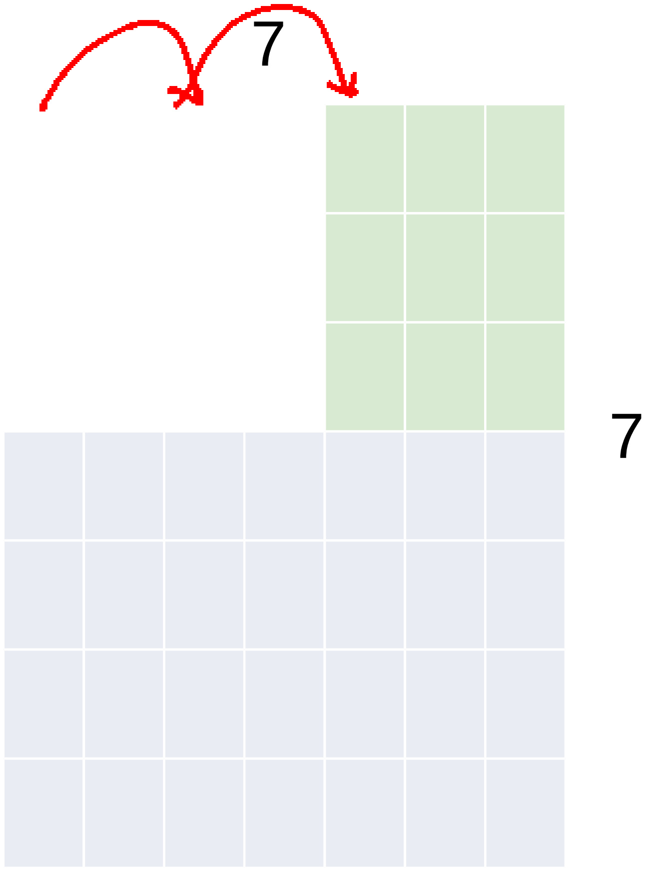
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

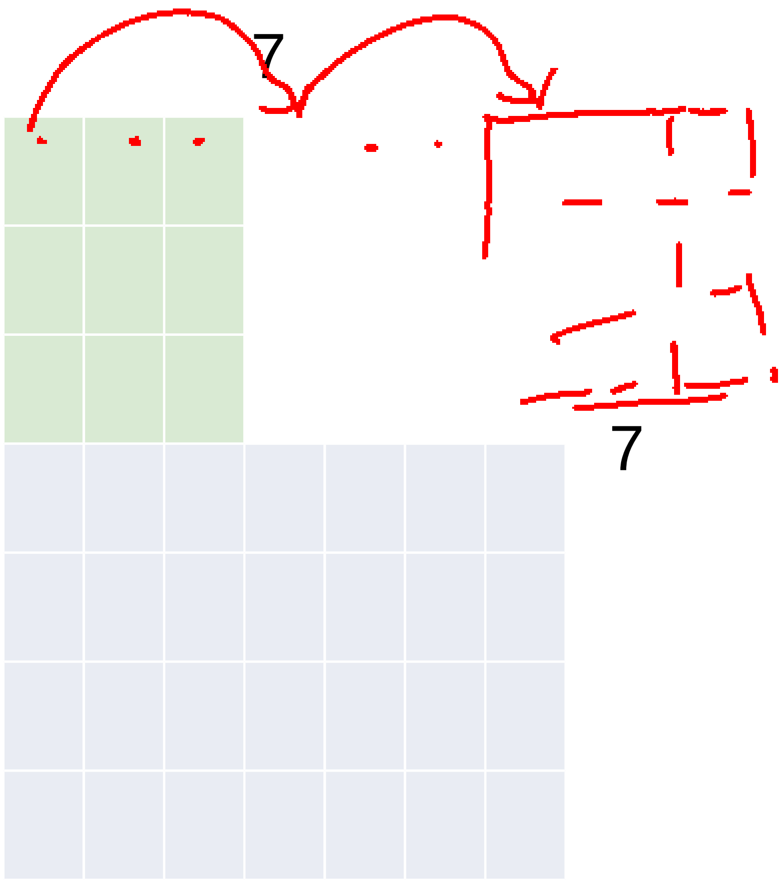
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

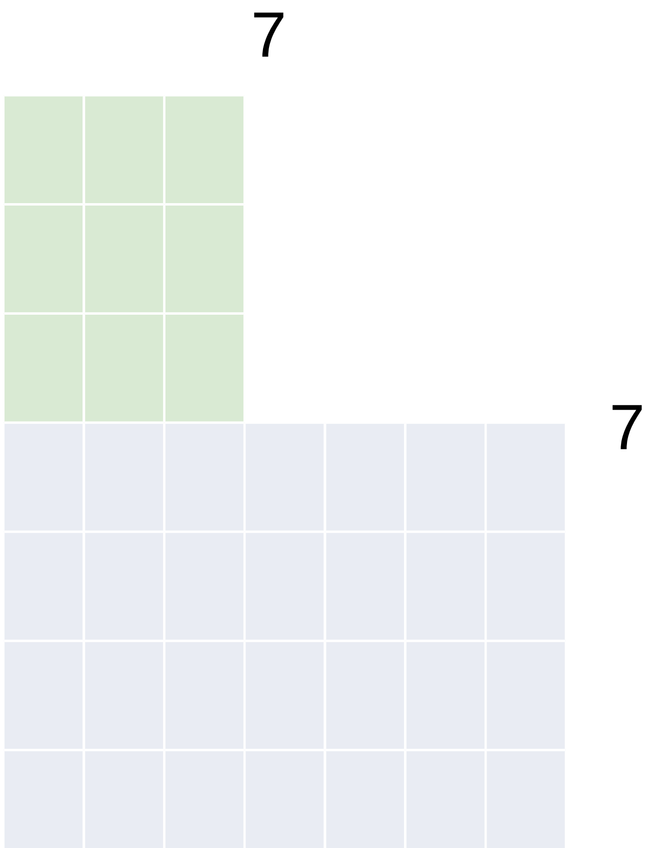
'valid'

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



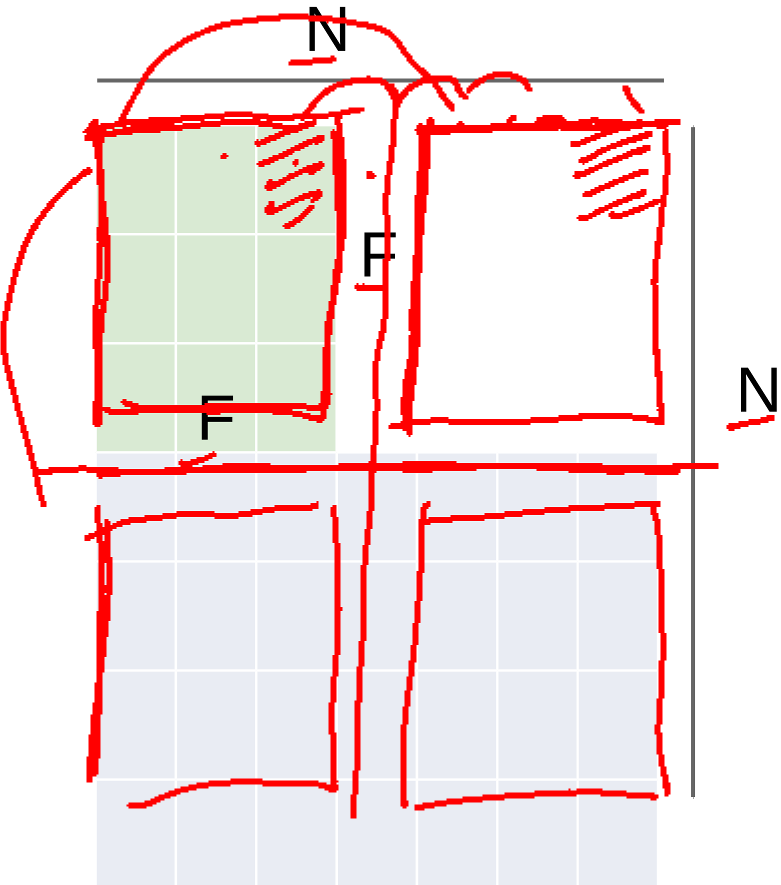
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

$N - F \propto$ stride

$\# \text{hops} = \frac{N-F}{\text{stride}}$

$\# \text{placements} = \# \text{hops} + 1$



Output size:

(N - F) / stride + 1

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

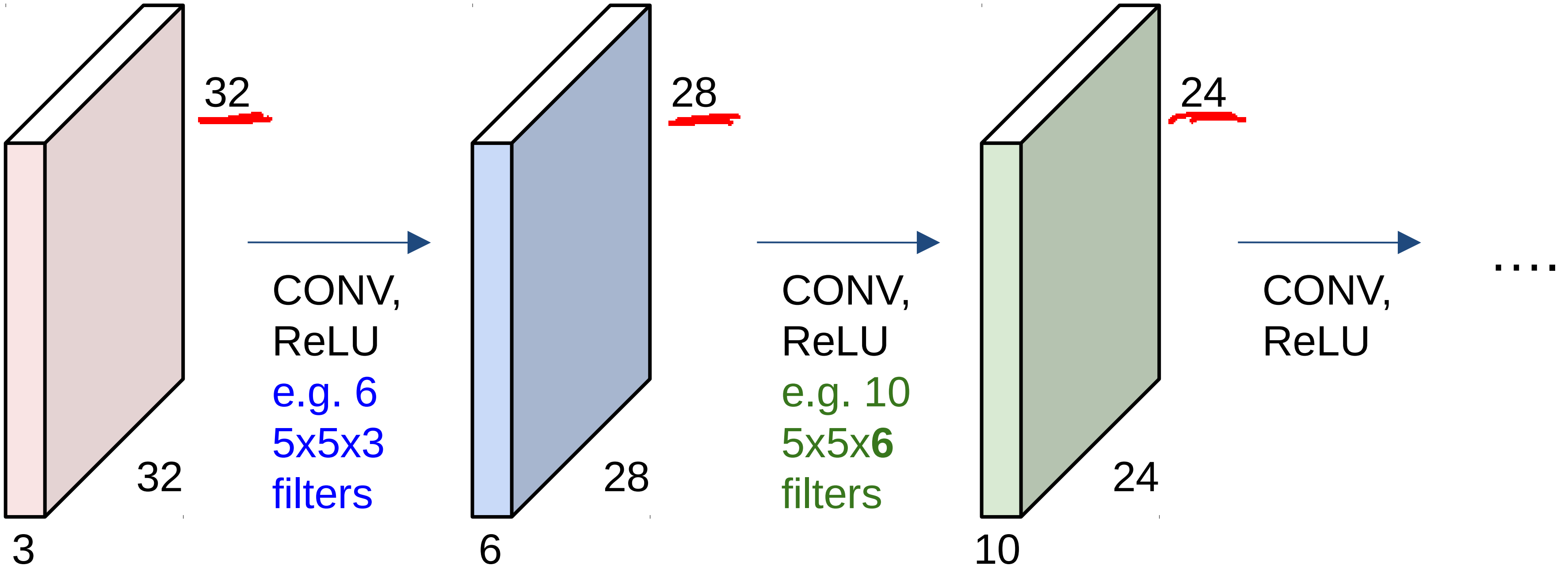
stride 1 => (7 - 3)/1 + 1 = 5

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

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

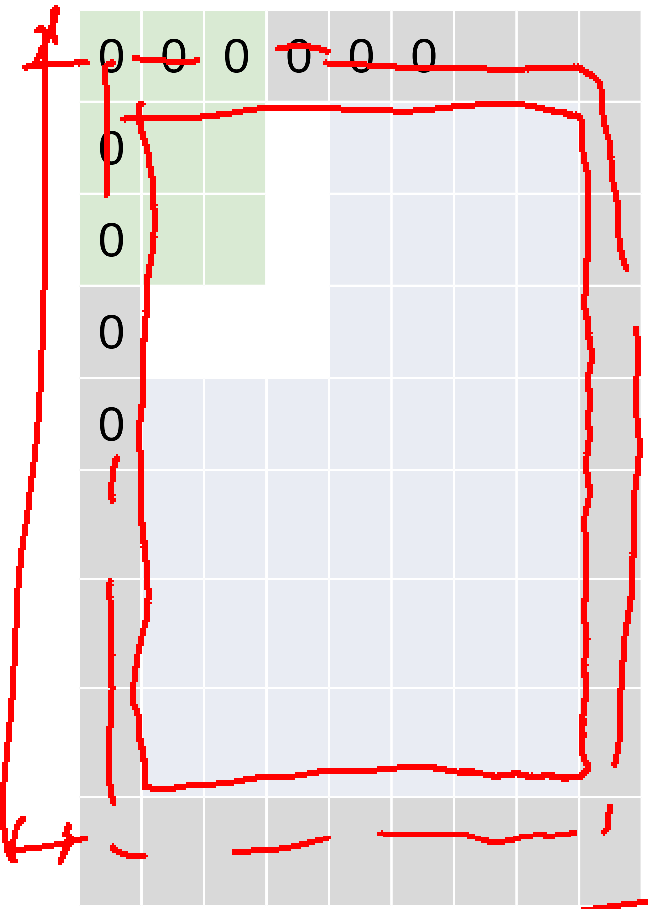
Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.



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

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?

$N \times N \rightarrow \tilde{N} \times \tilde{N}$

$\tilde{N} = N + 2 \cdot pad$

(recall:)
(N - F) / stride + 1

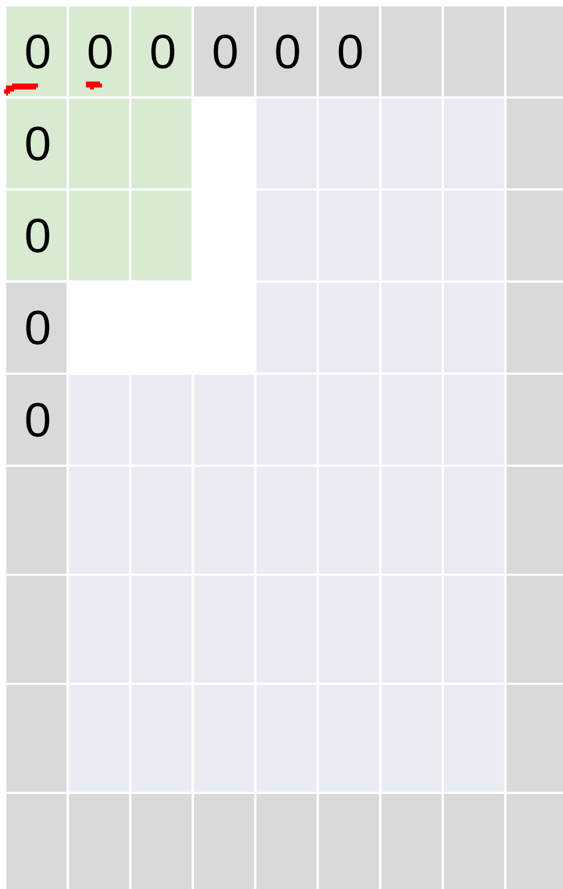
If stride=1 $\left\{ \frac{N + 2pad - F}{stride} + 1 \right.$

$\boxed{pad = \frac{F-1}{2}}$

$\frac{7+2-3}{1} + 1 = 7$

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

In practice: Common to zero pad the border

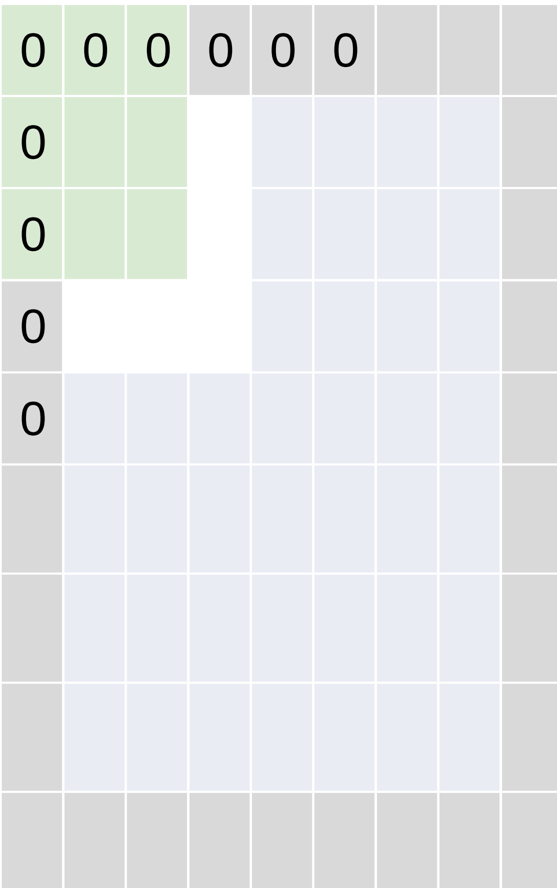


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

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

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 (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

Examples time:

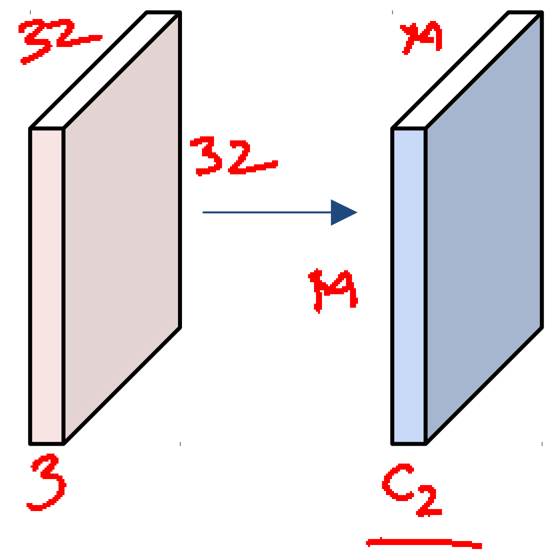
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

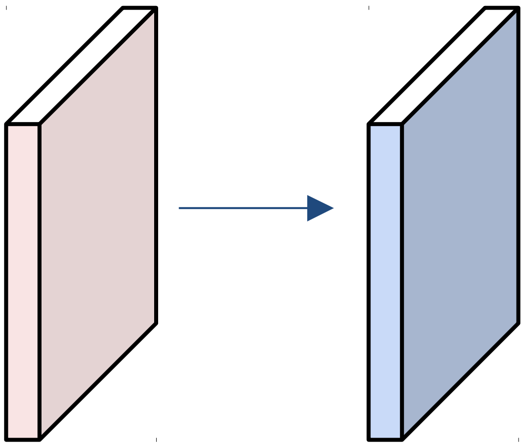
Output volume size: ?

$M \times M \times \boxed{C_2}$

$32 \times 32 \times 10$



Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

Examples time:

Input volume: **32x32x3**

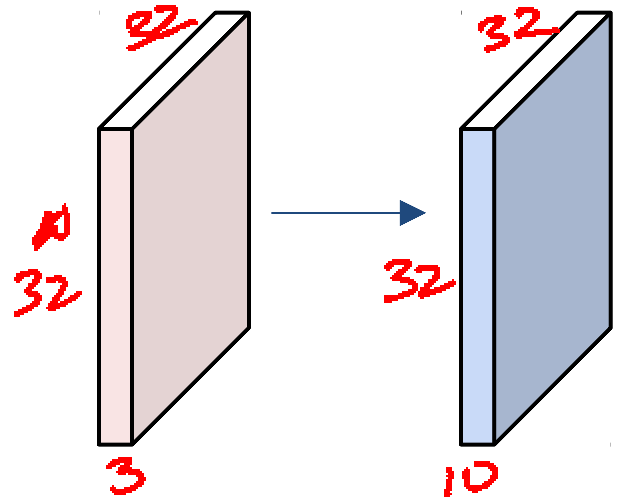
10 5x5 filters with stride 1, pad 2

(5x5x3)

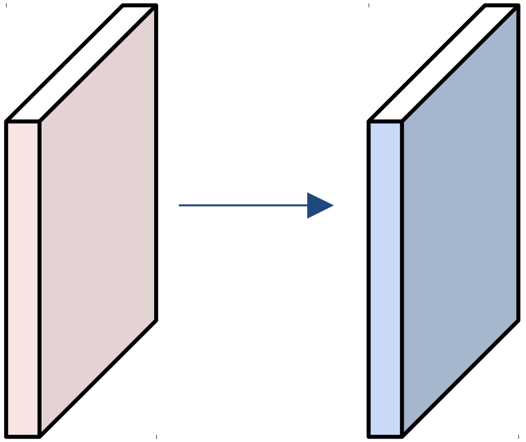
Number of parameters in this layer?

$k_1 \times k_2 \qquad C_1 \qquad C_2$

$$\left(5\times5\times3 + 1\right)\times 10 = 76\times10 = 760$$



Examples time:



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 (+1 for bias)
=> 76*10 = **760**

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.

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.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

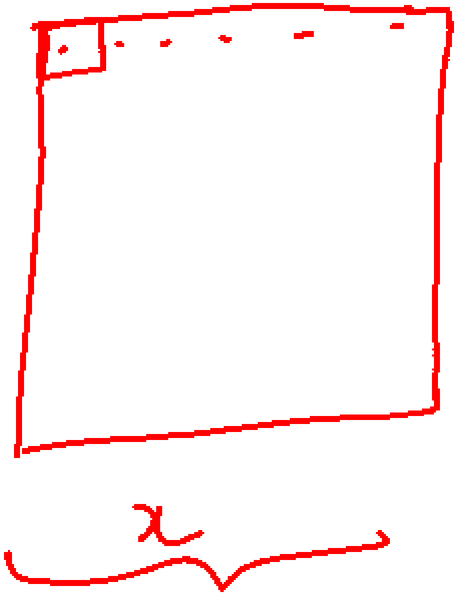
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

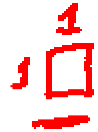
Plan for Today

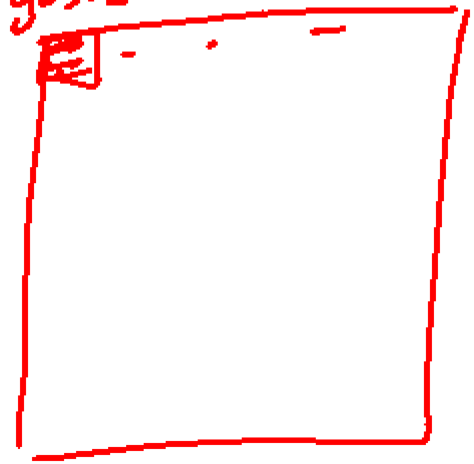
- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult

Can we have 1x1 filters?

$y[0,0] = x[0,0]\,w[0,0]$



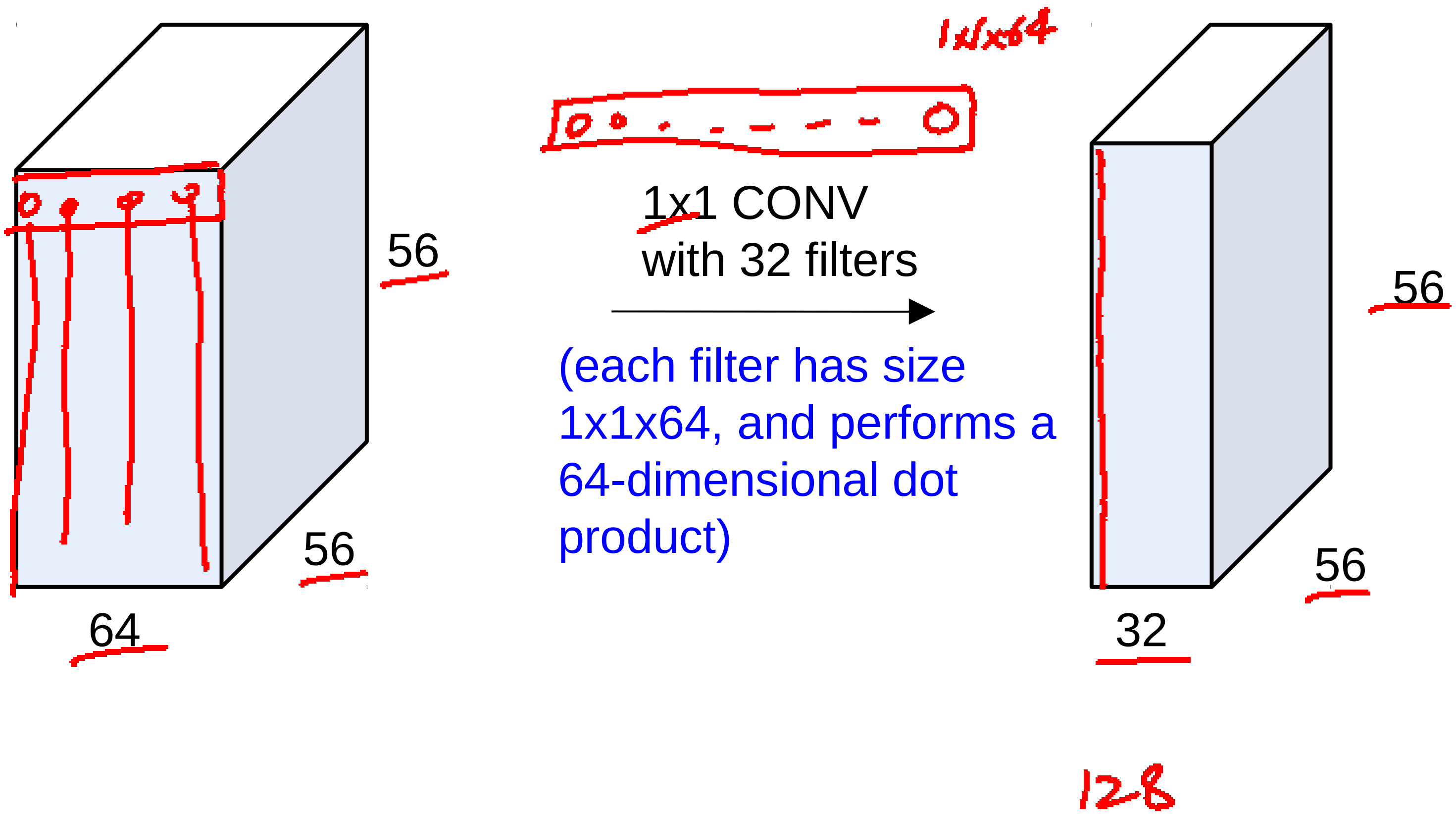




$$y[r,c] = \cancel{\sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1}} x[r+a, c+b]\,w[a,b]$$

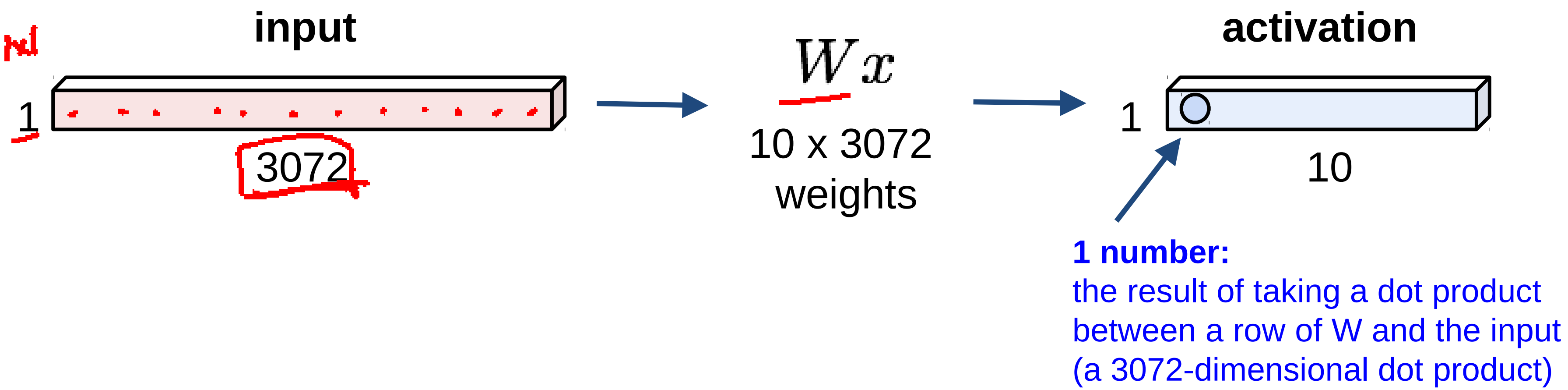
$$= x[r,c]\,w[0,0]$$

1x1 convolution layers make perfect sense



Fully Connected Layer as 1x1 Conv

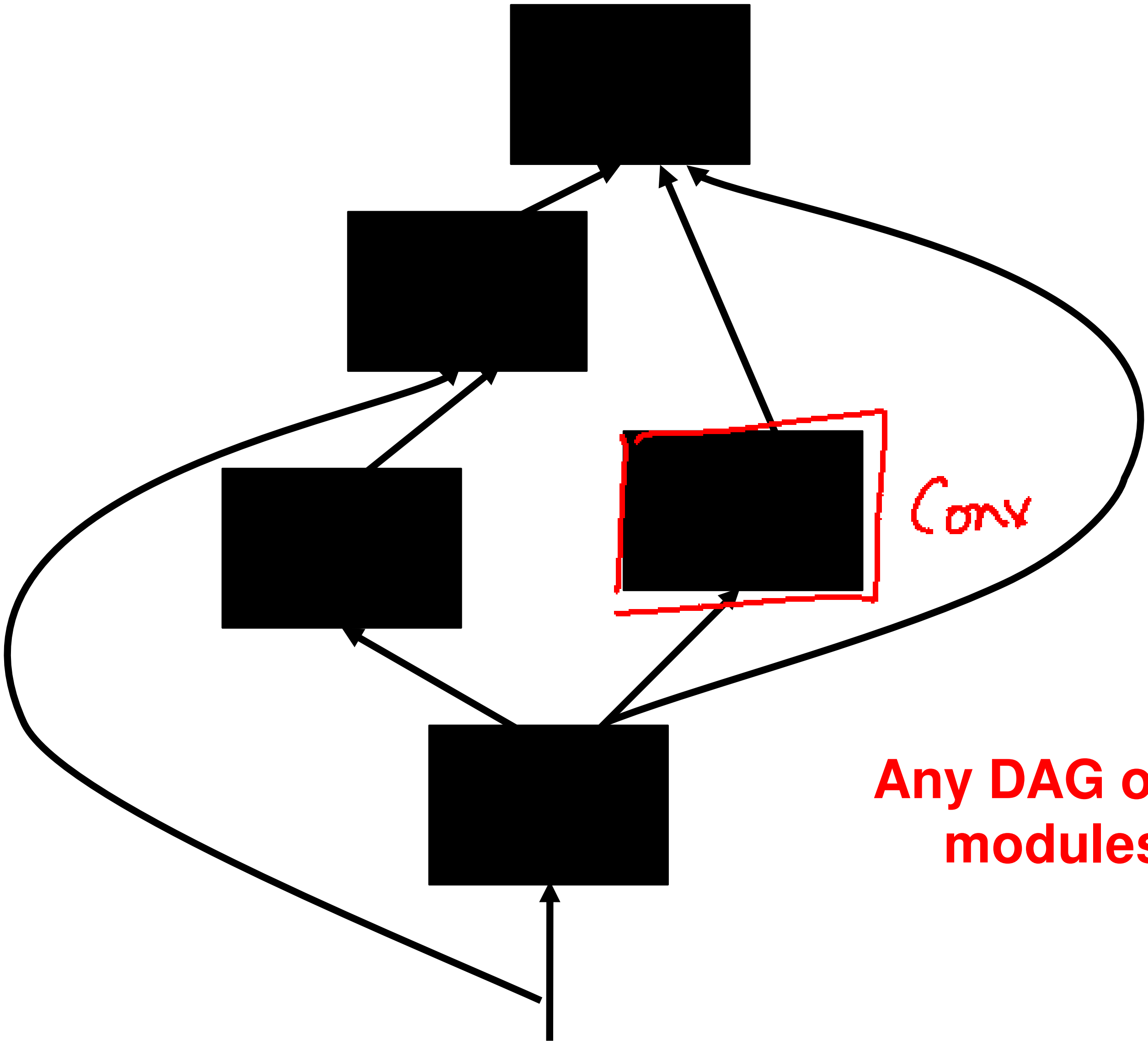
32x32x3 image -> stretch to 3072 x 1



Plan for Today

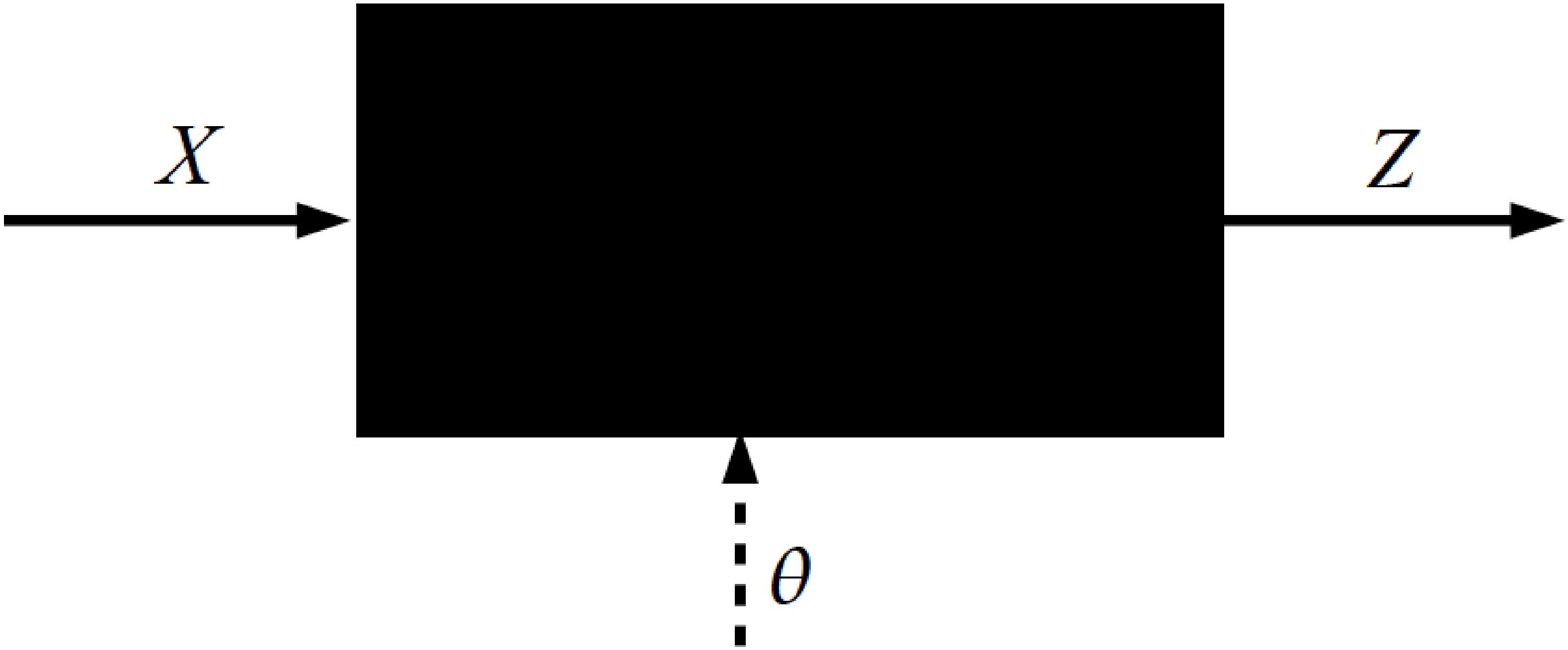
- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult

Computational Graph

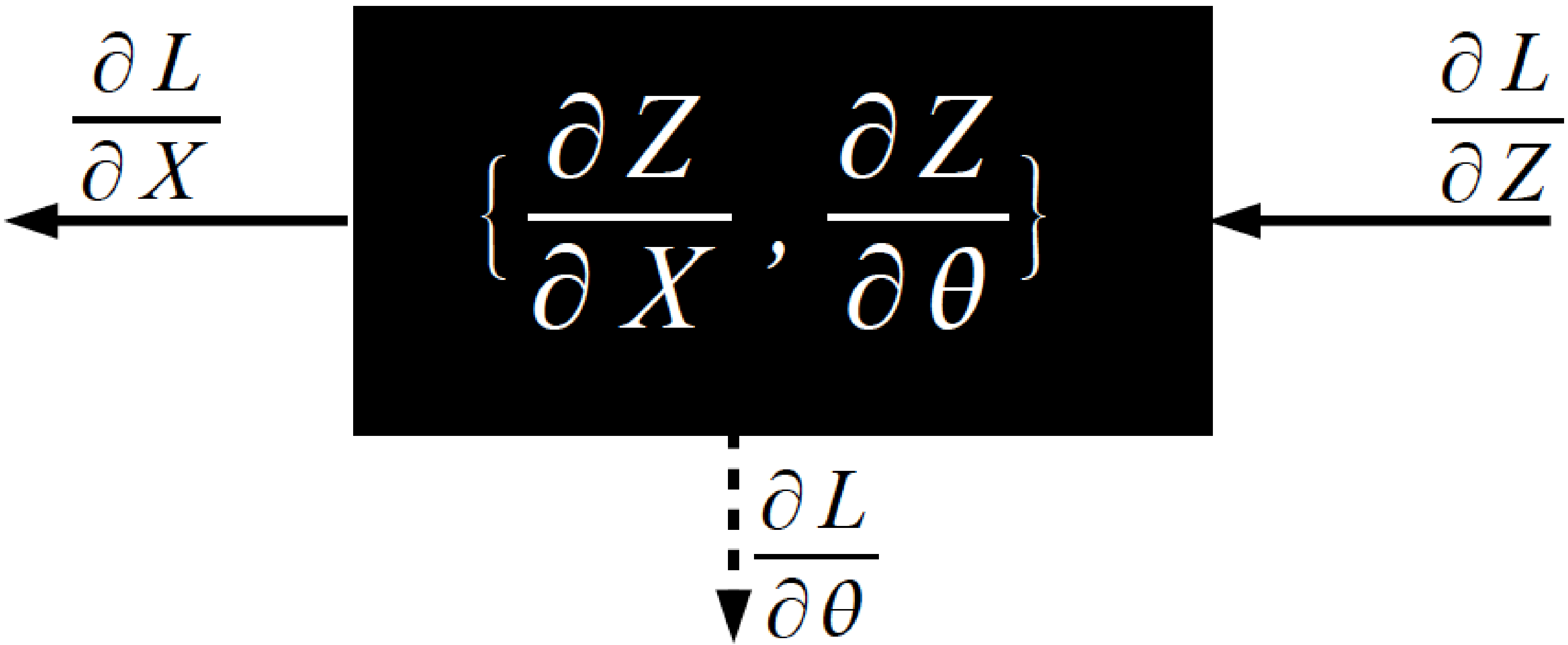


Any DAG of differentiable modules is allowed!

Key Computation: Forward-Prop



Key Computation: Back-Prop

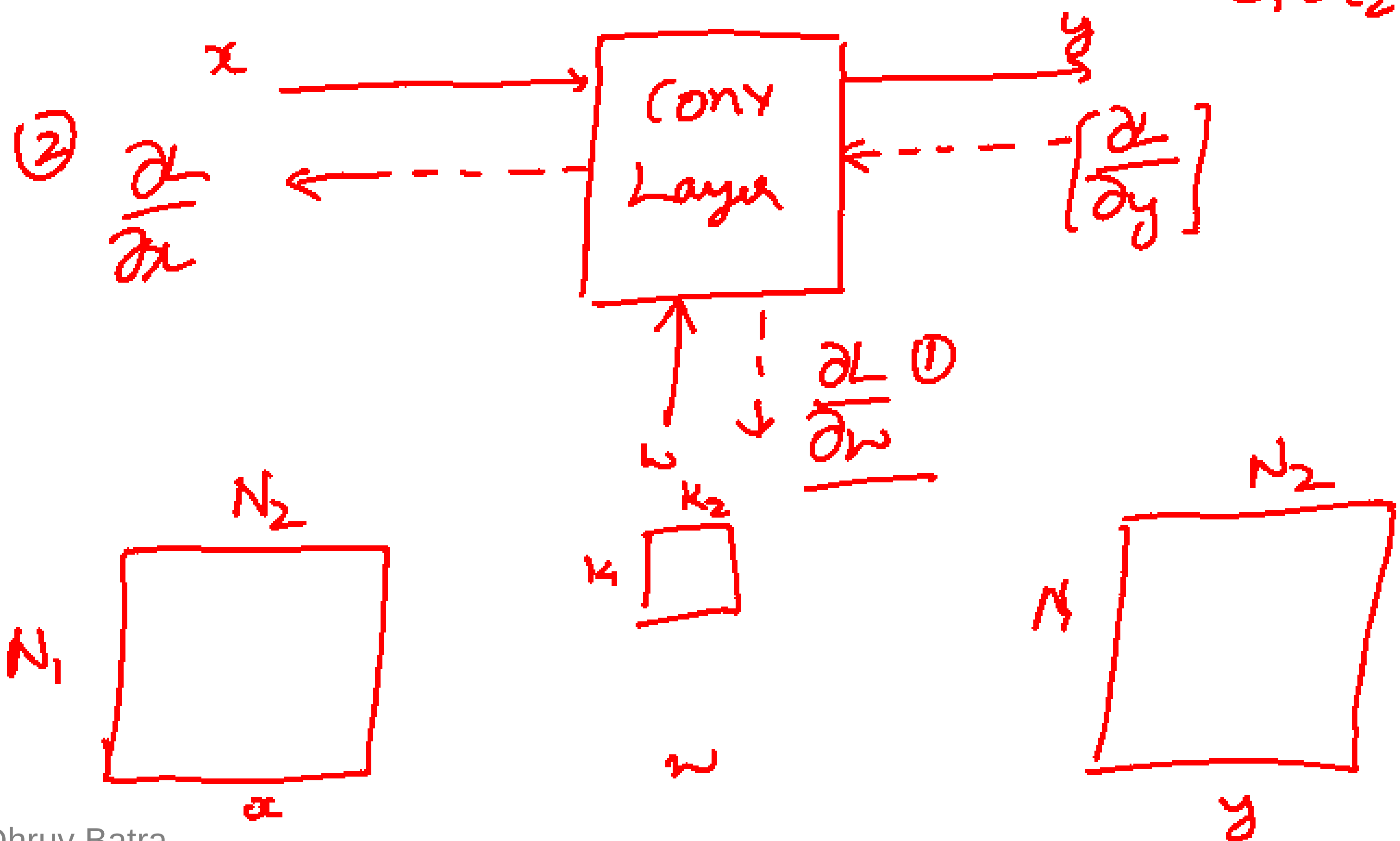


Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Backprop in Convolutional Layers

- Notes
  - https://www.cc.gatech.edu/classes/AY2021/cs7643_fall/slides/L11_cnns_backprop_notes.pdf

$C_1 = C_2 = 1$



Backprop in Convolutional Layers

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

Input

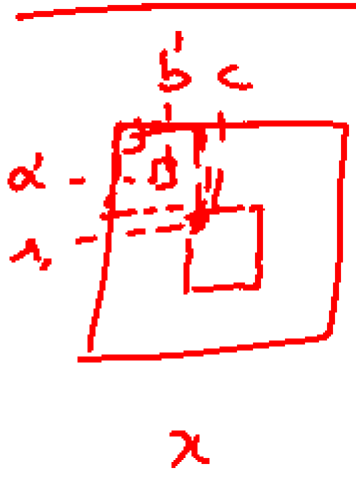
component of the Jacobian

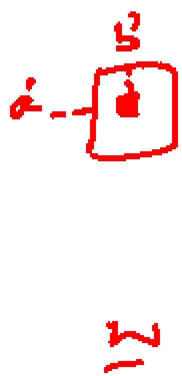
Input: $\left[\frac{\partial L}{\partial y}\right]$

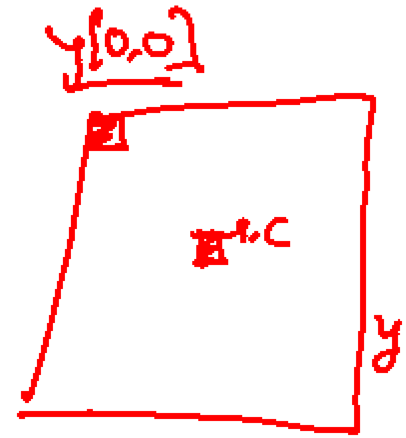
Output: $\frac{\partial L}{\partial w}$

$$\frac{\partial L}{\partial w[a',b']} = \sum_{r=0}^{N_r-1} \sum_{c=0}^{N_c-1} \frac{\partial L}{\partial y[r,c]} \boxed{\frac{\partial y[r,c]}{\partial w[a',b']}}$$

$= x[r+a', c+b']$







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

$= x[r+a', c+b']$

Backprop in Convolutional Layers

$$\boxed{\frac{\partial L}{\partial w[a',b']}} = \sum_{r=0}^{N_1-1} \sum_{c=0}^{N_2-1} \frac{\partial L}{\partial y[r,c]} \, x[r+a', c+b']$$

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

$$\frac{\partial L}{\partial w} = x * \frac{\partial L}{\partial y}$$

[with appropriate padding]

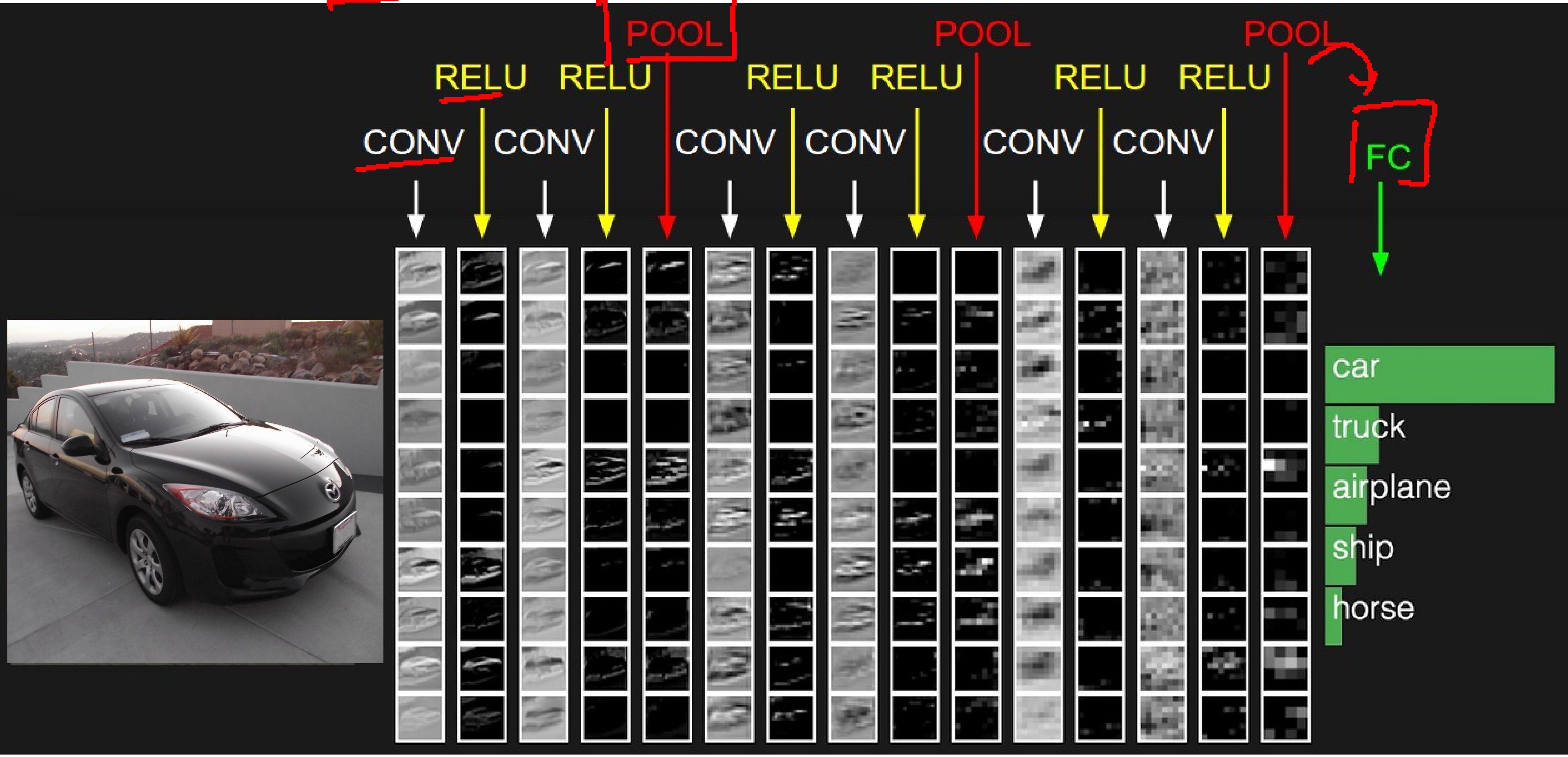
$$[\quad]_{N_1 \times N_2} \; [\quad]_{N_1 \times N_2}$$

Backprop in Convolutional Layers

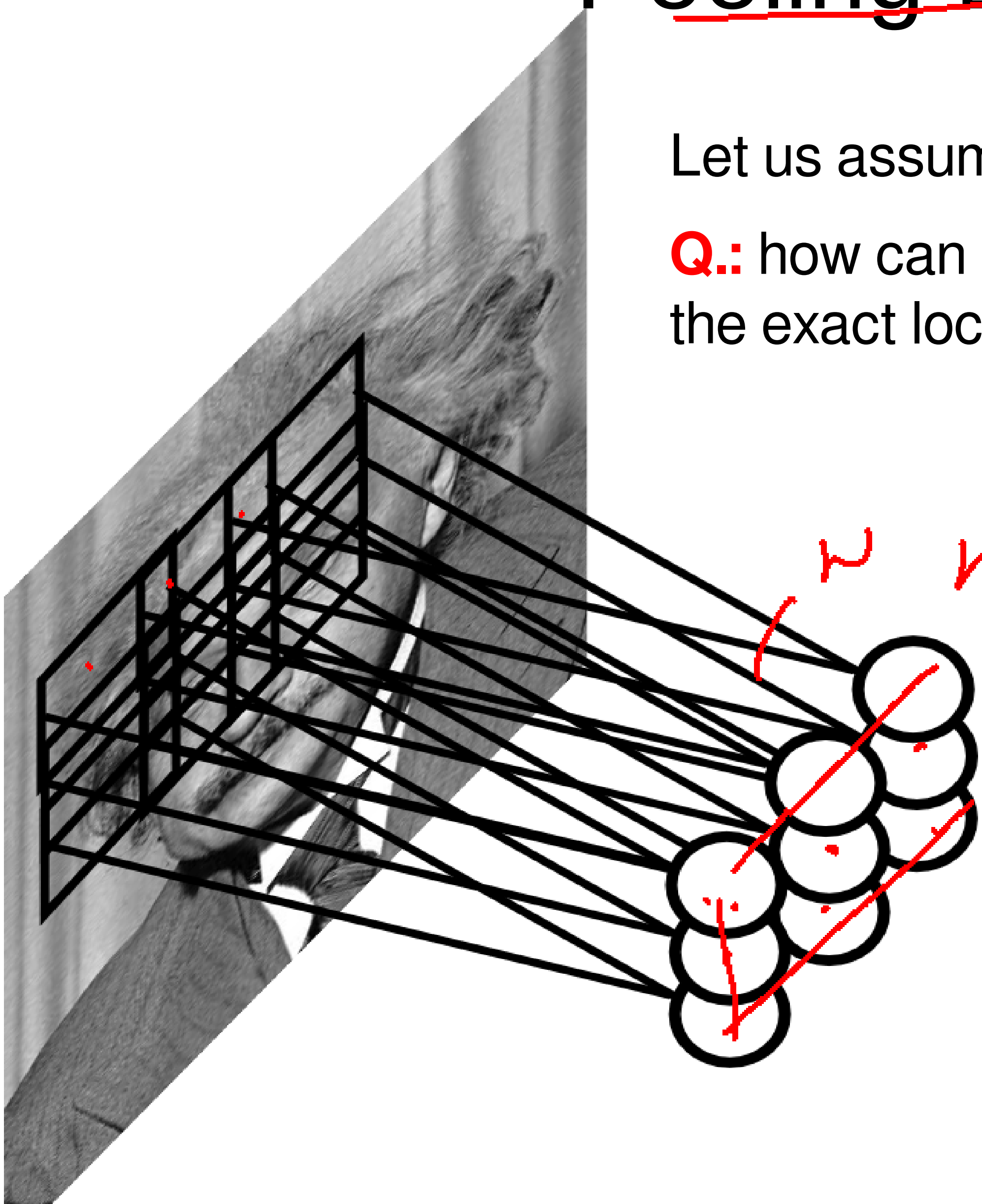
Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult

two more layers to go: POOL/FC



Pooling Layer



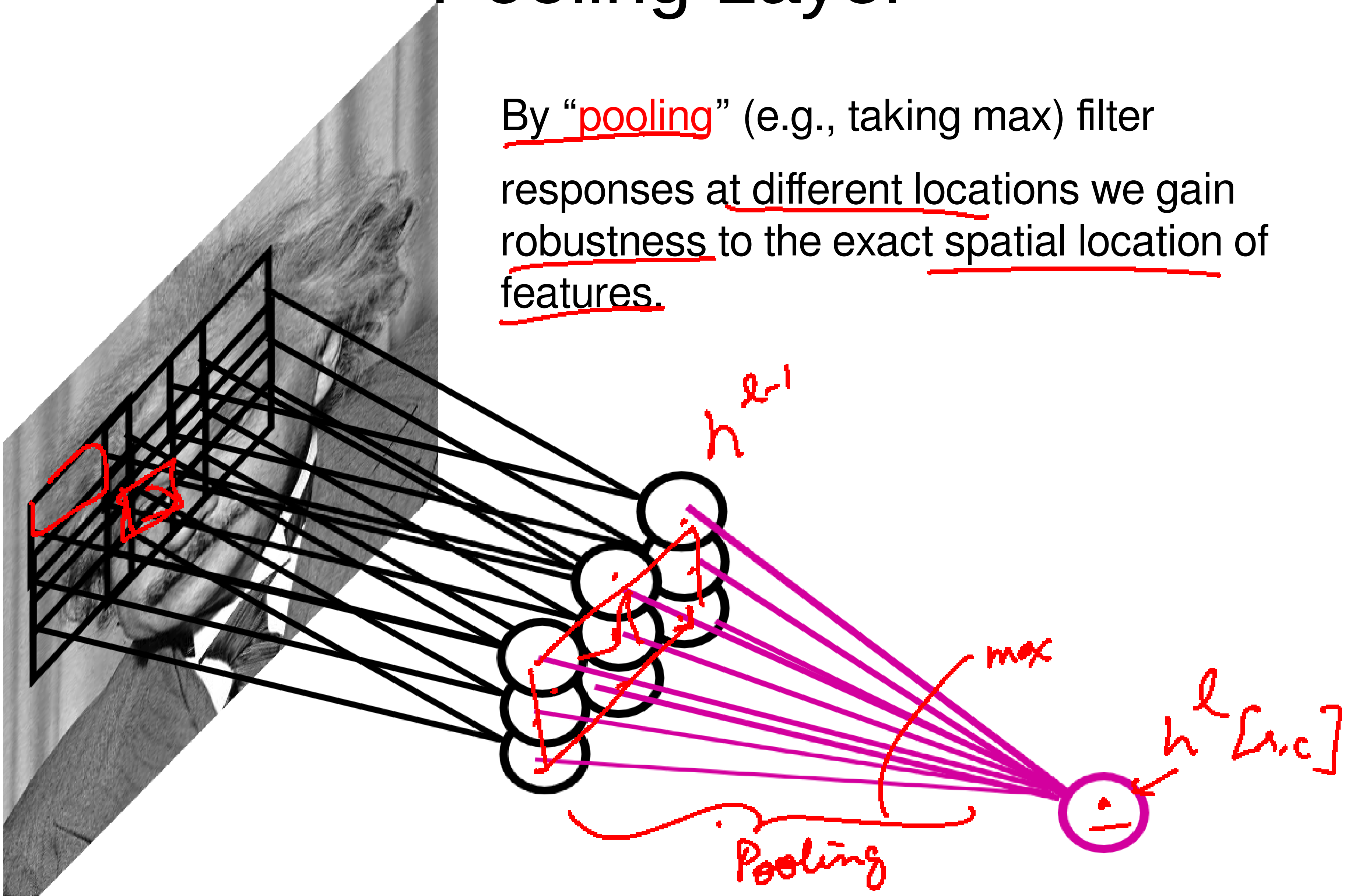
Let us assume filter is an “eye” detector.

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

Pooling Layer

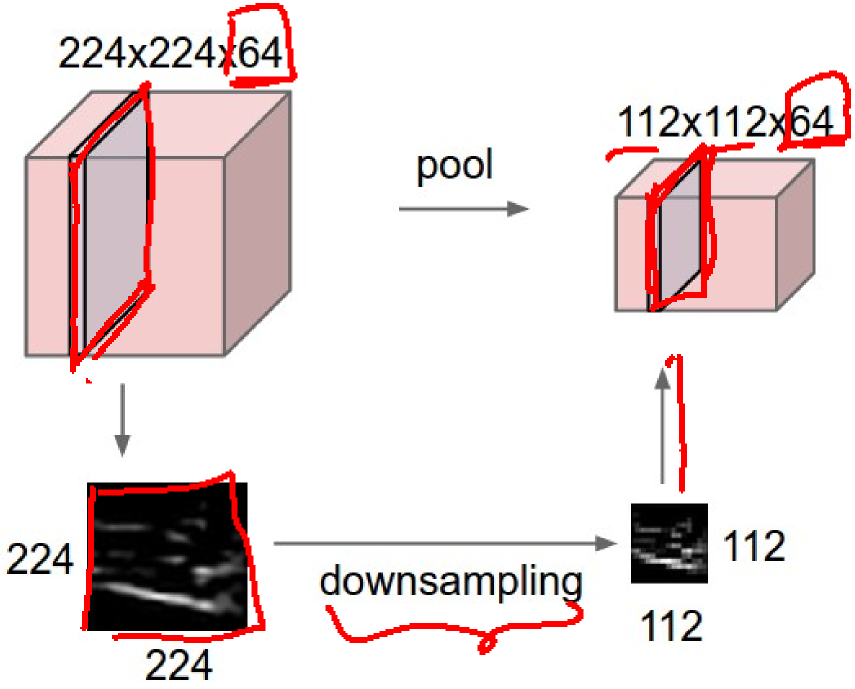
By "pooling" (e.g., taking max) filter

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

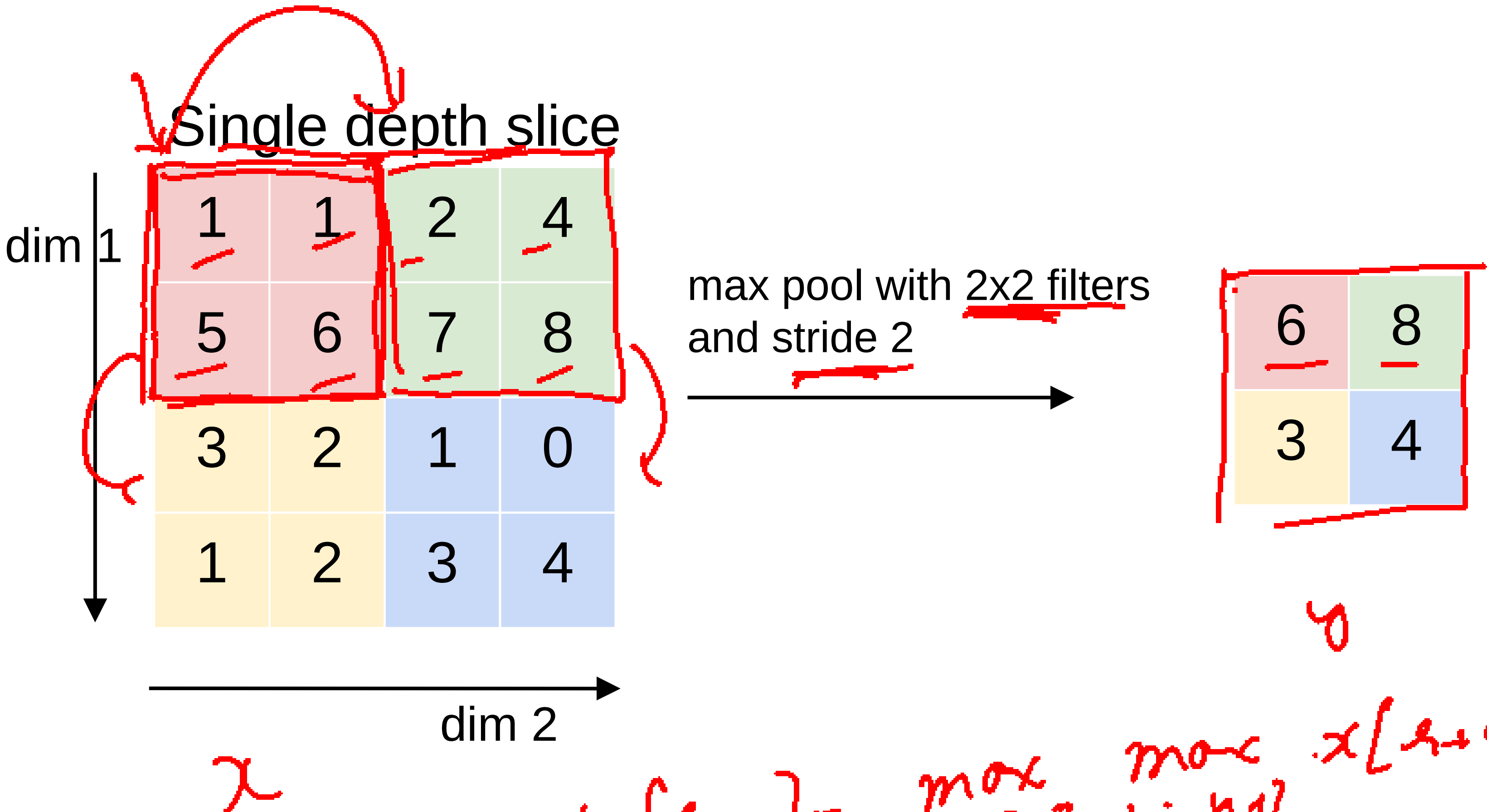


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING



$$y[r, c] = \max_{a \in \{0,1\}} \max_{b \in \{0,1\}} x[r+a, c+b]$$

1	3	2	9
7	4	1	5
8	5	2	3
4	2	1	4

4x4

Pooling Layer: Examples

Max-pooling:

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

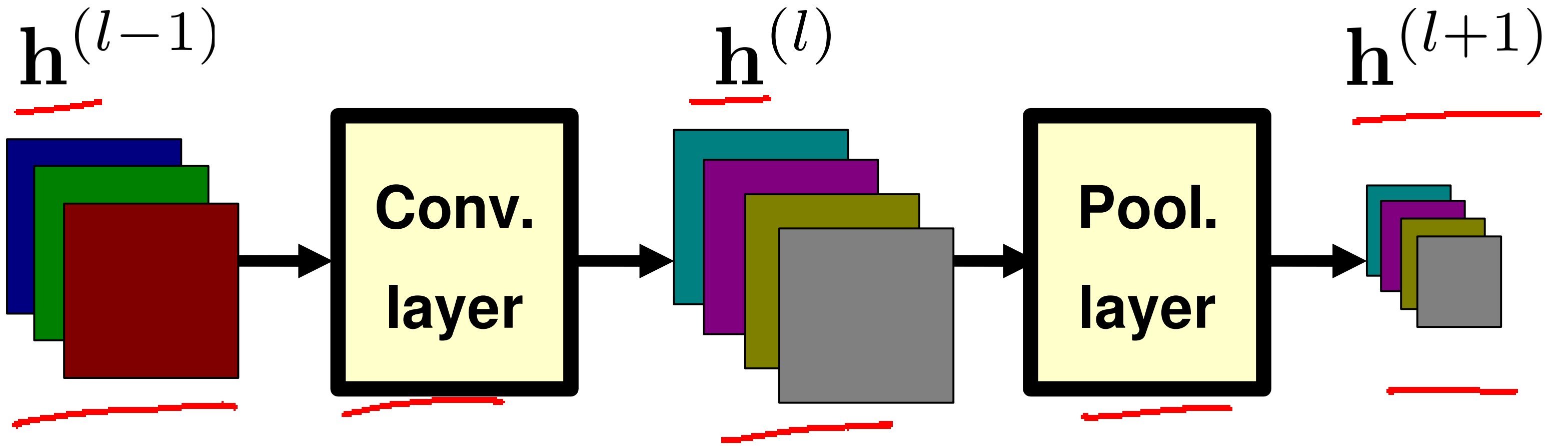
Average-pooling:

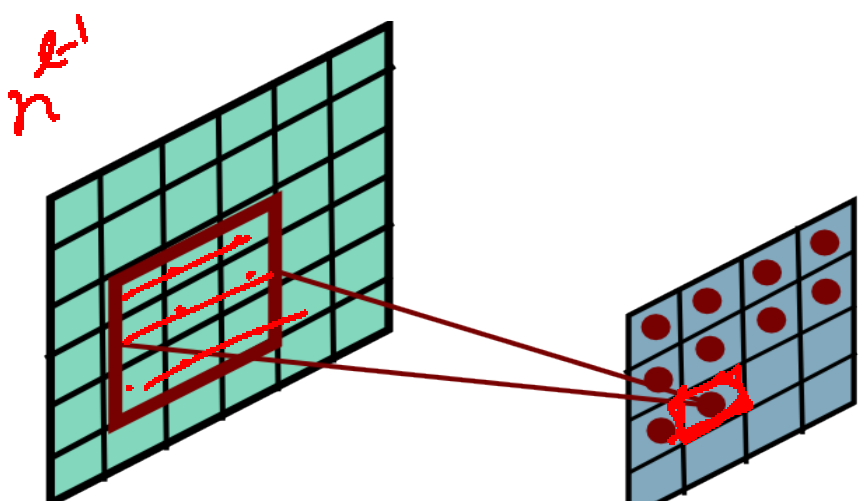
$$h_i^n(r,c) = \boxed{\operatorname*{mean}_{\bar{r}\in N(r),\ \bar{c}\in N(c)}} h_i^{n-1}(\bar{r},\bar{c})$$

L2-pooling:

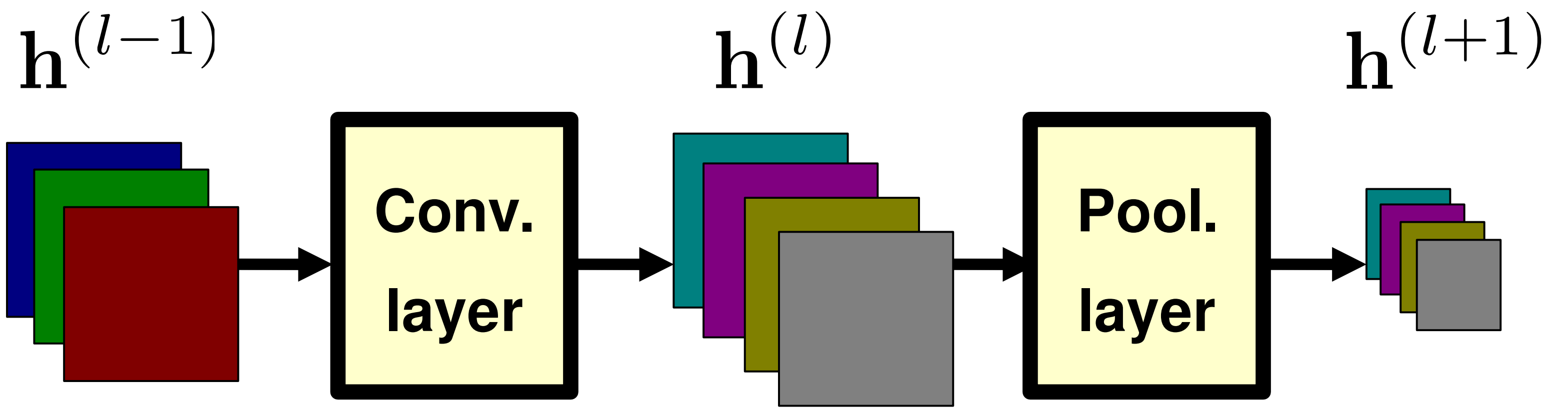
$$h_i^n(r,c) = \sqrt{\sum_{\bar{r}\in N(r),\ \bar{c}\in N(c)} h_i^{n-1}(\bar{r},\bar{c})^2}$$

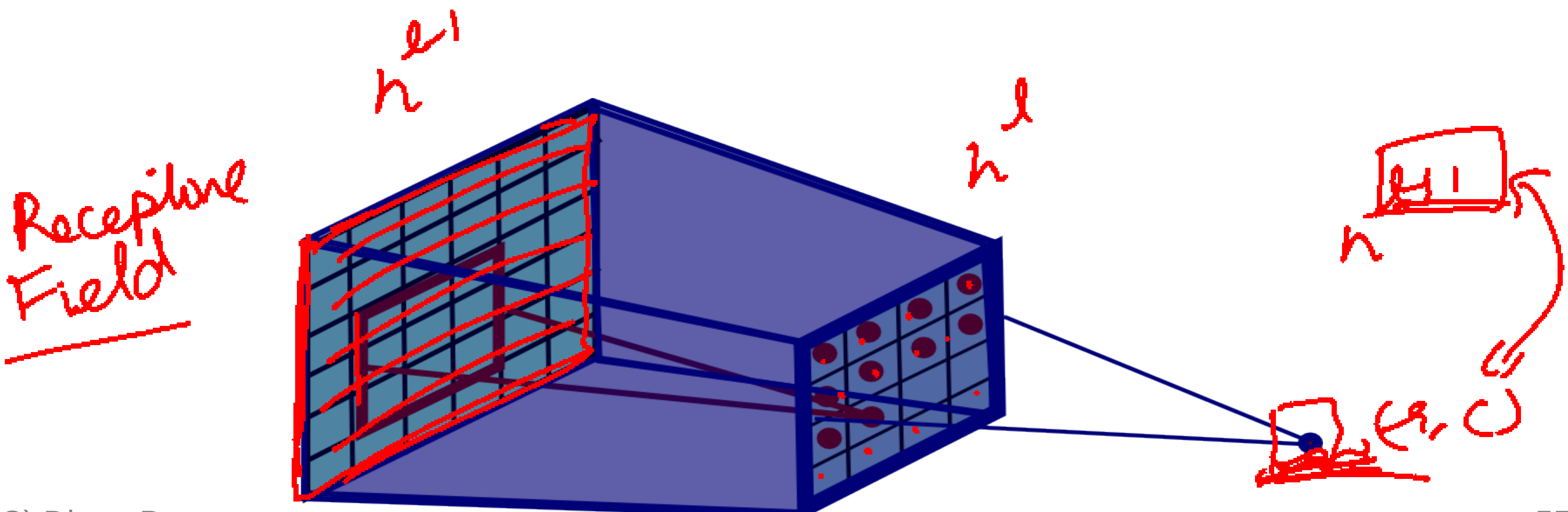
Receptive Field



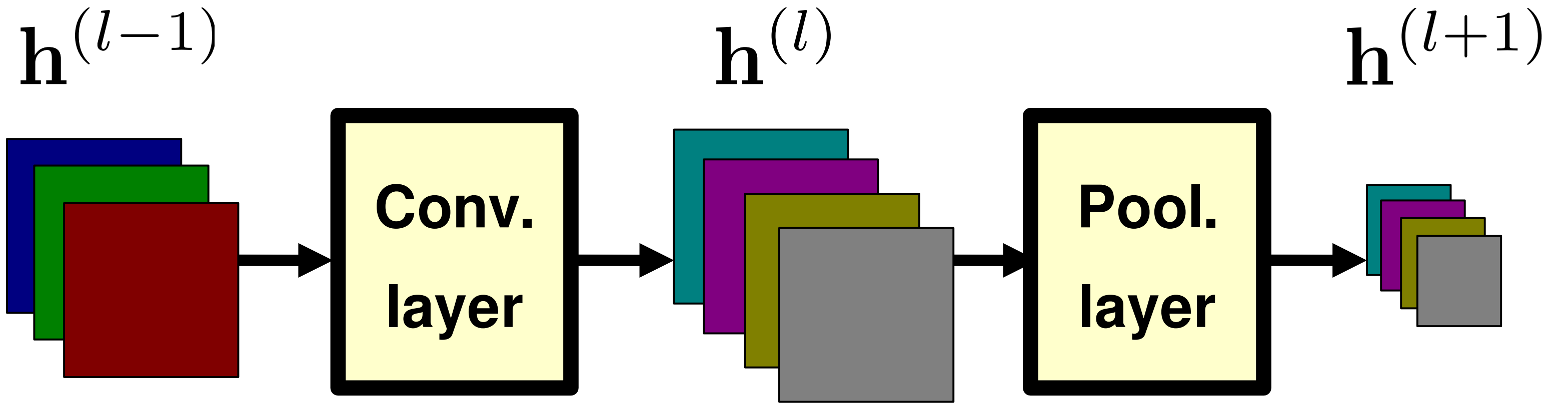


Pooling Layer: Receptive Field Size

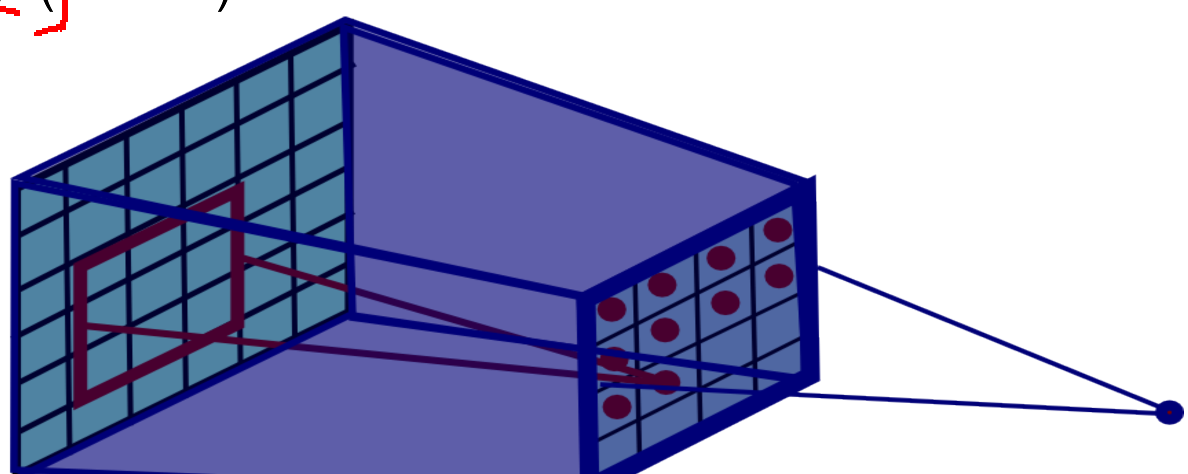




Pooling Layer: Receptive Field Size



If convolutional filters are FxF and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch in $\mathbf{h}^{(l-1)}$ of size: (P+F-1)x(P+F-1)



- 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

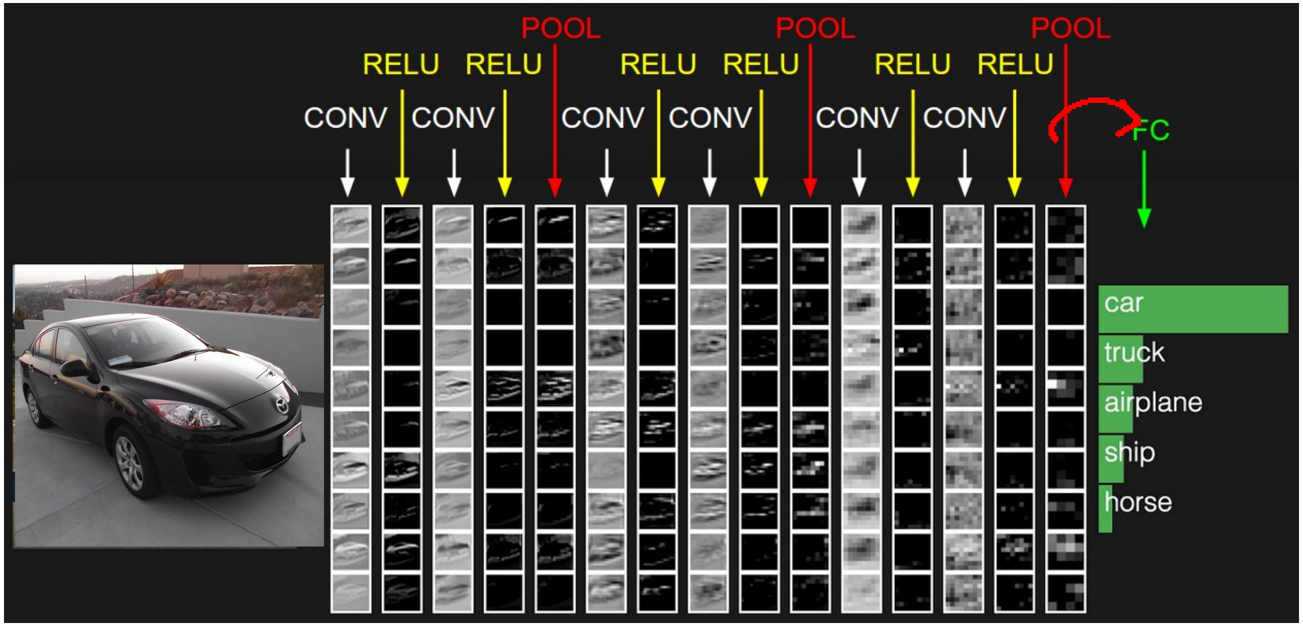
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since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

Common settings:

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

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Convolutional Neural Networks

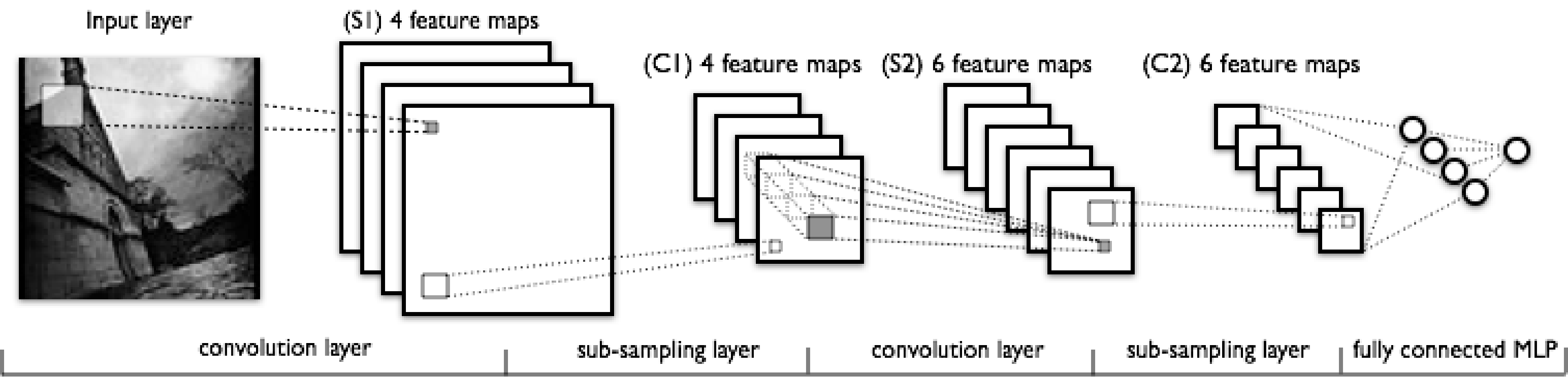


Image Credit: Yann LeCun, Kevin Murphy

Classical View

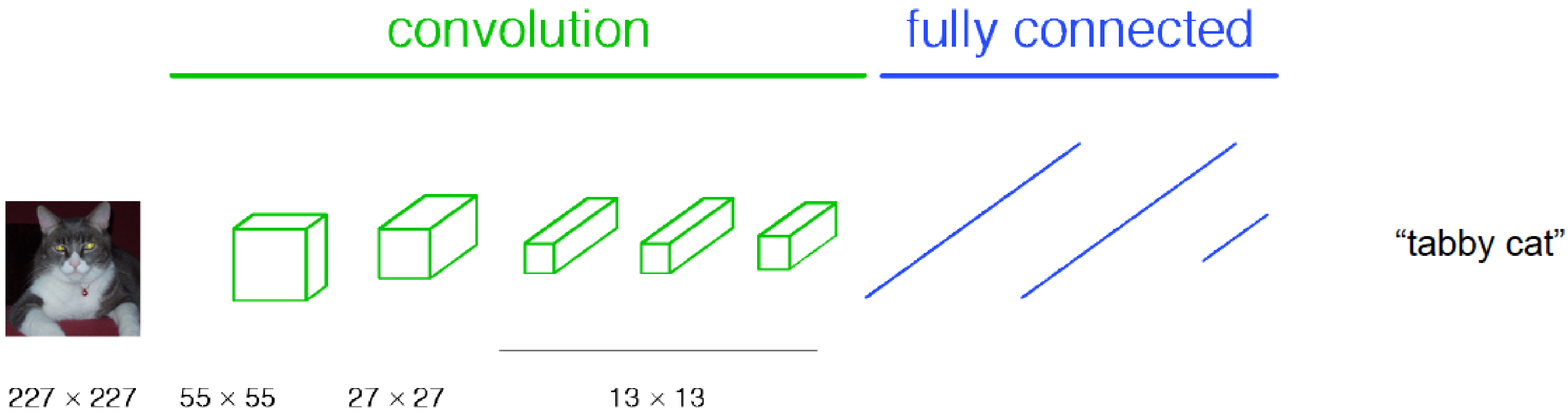
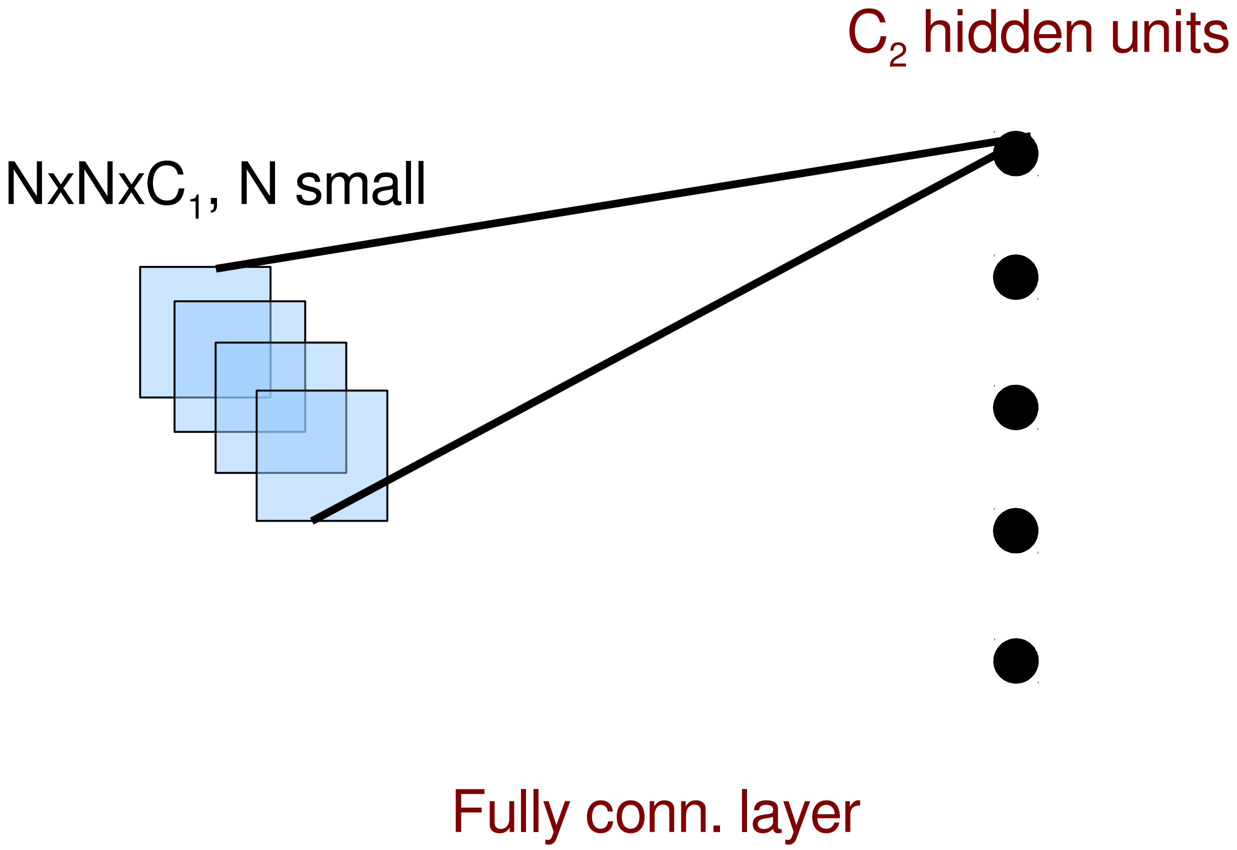


Figure Credit: [Long, Shelhamer, Darrell CVPR15]


C_2 hidden units
$NxNxC_1$, N small
Fully conn. layer

Classical View

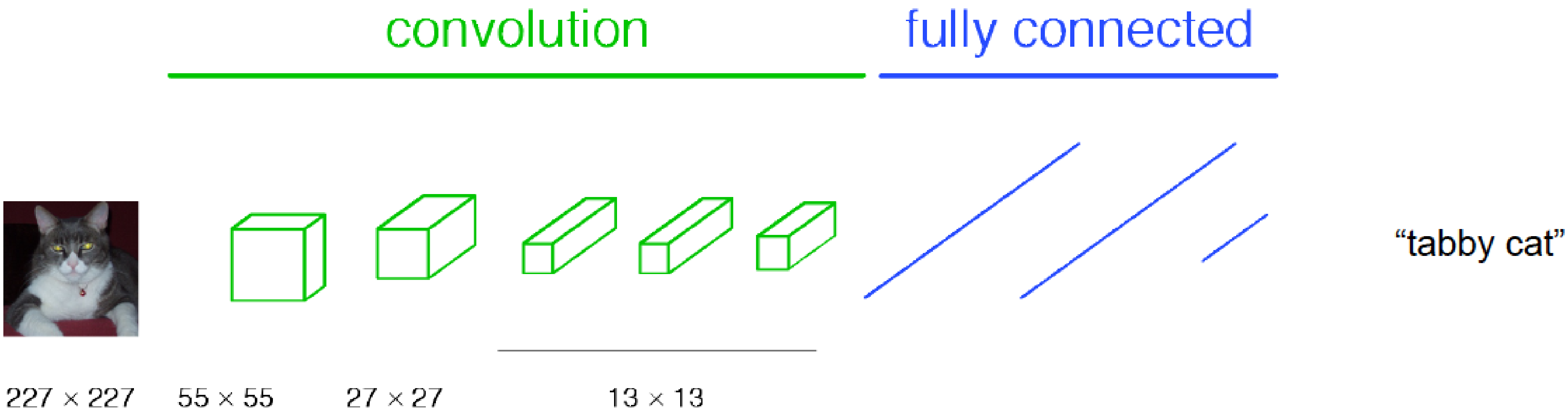
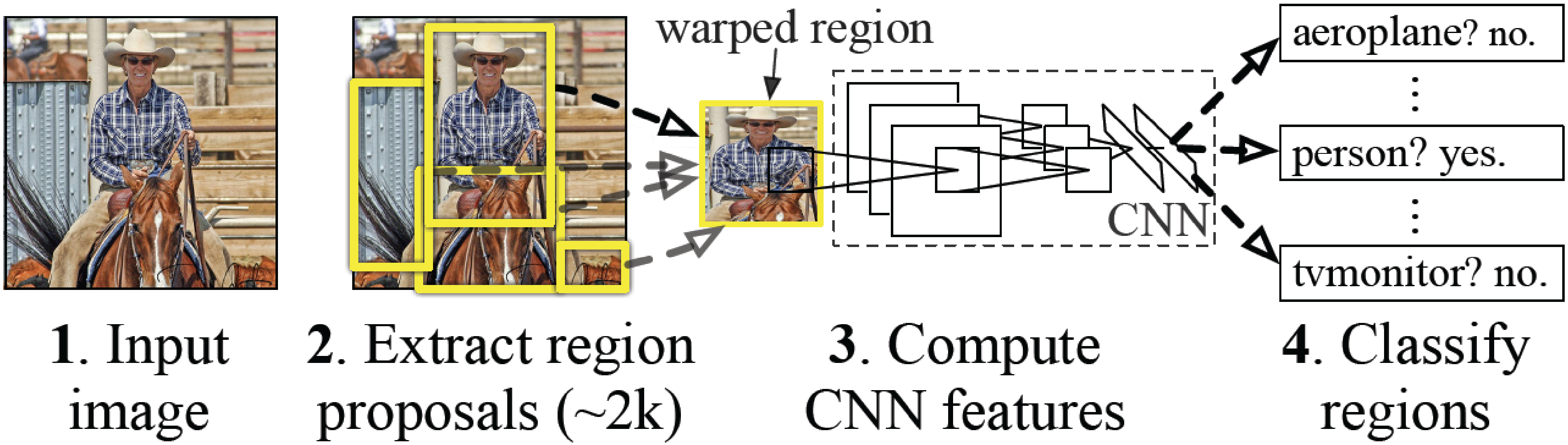


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Classical View = Inefficient



Classical View

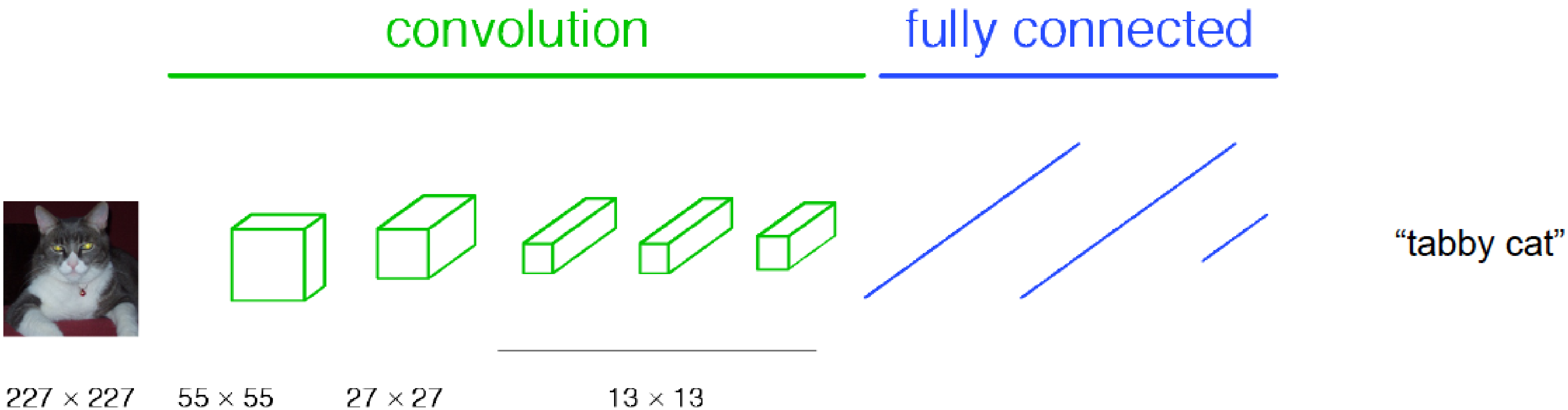
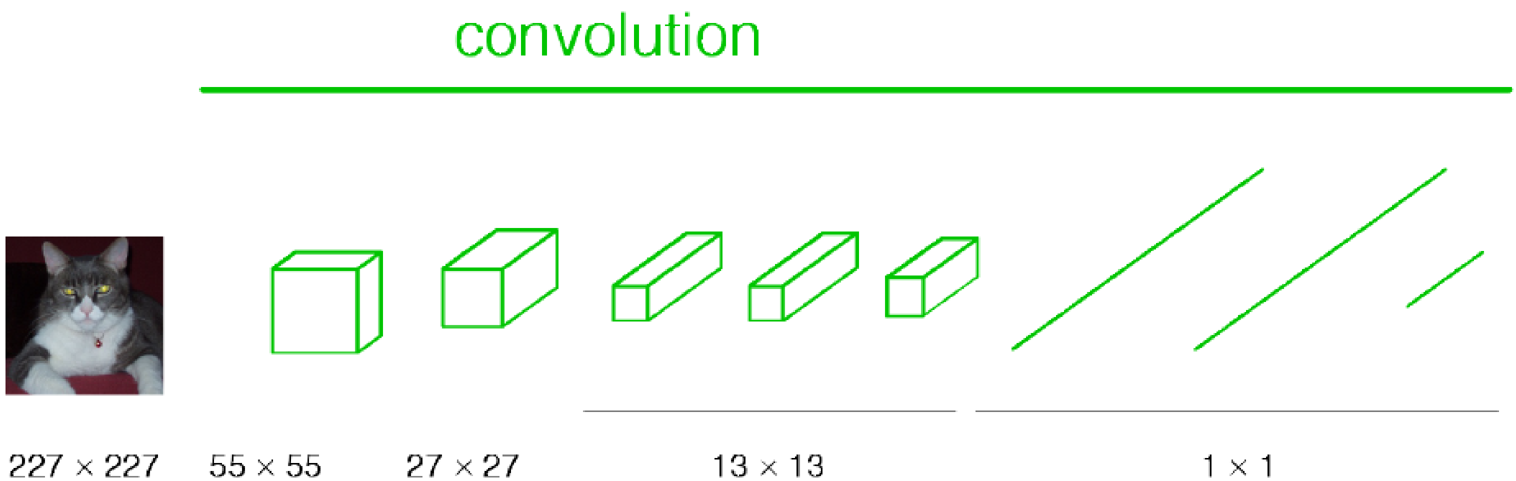


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

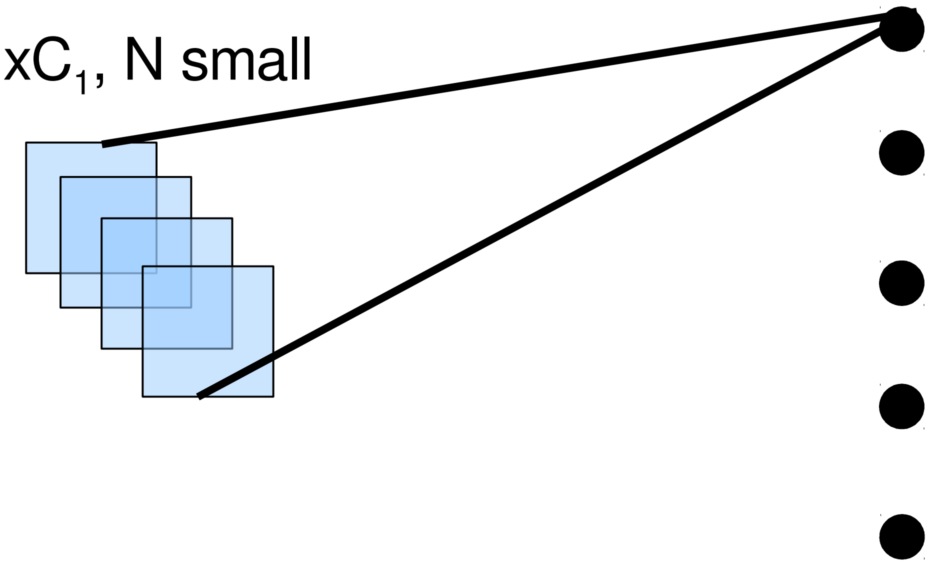
- Just squint a little!



C_2 hidden units /

$1x1xC_2$ feature maps

$NxNxC_1$, N small



Fully conn. layer /

Conv. layer (C_2 kernels of size $NxNxC_1$)

Re-interpretation

- Just squint a little!

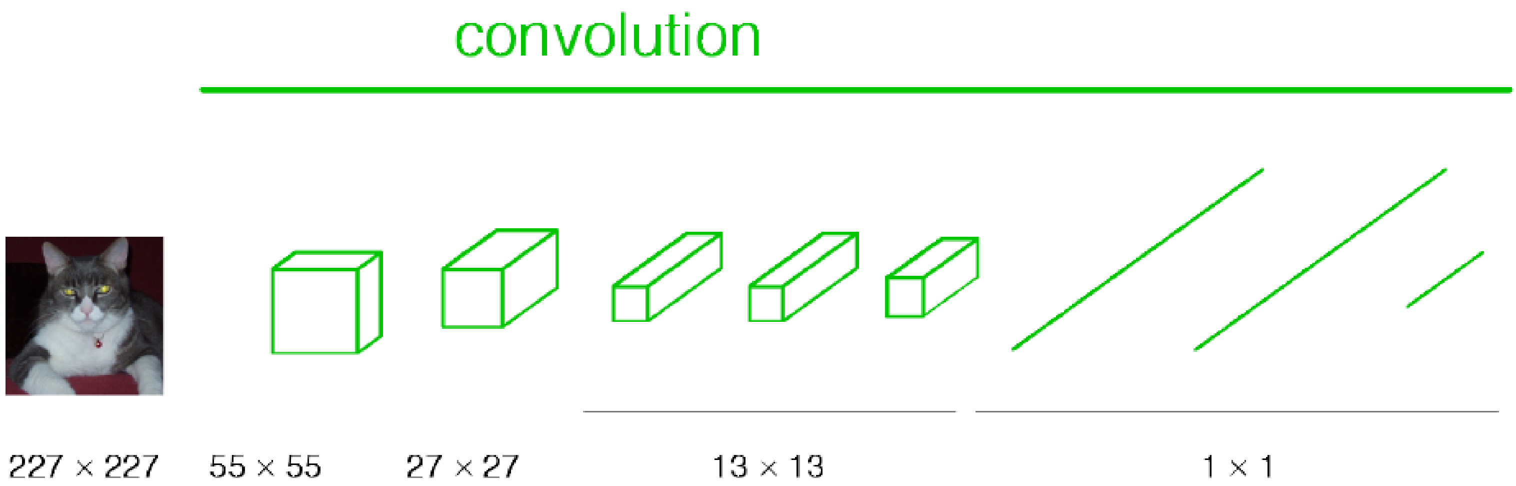


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

“Fully Convolutional” Networks

- Can run on an image of any size!

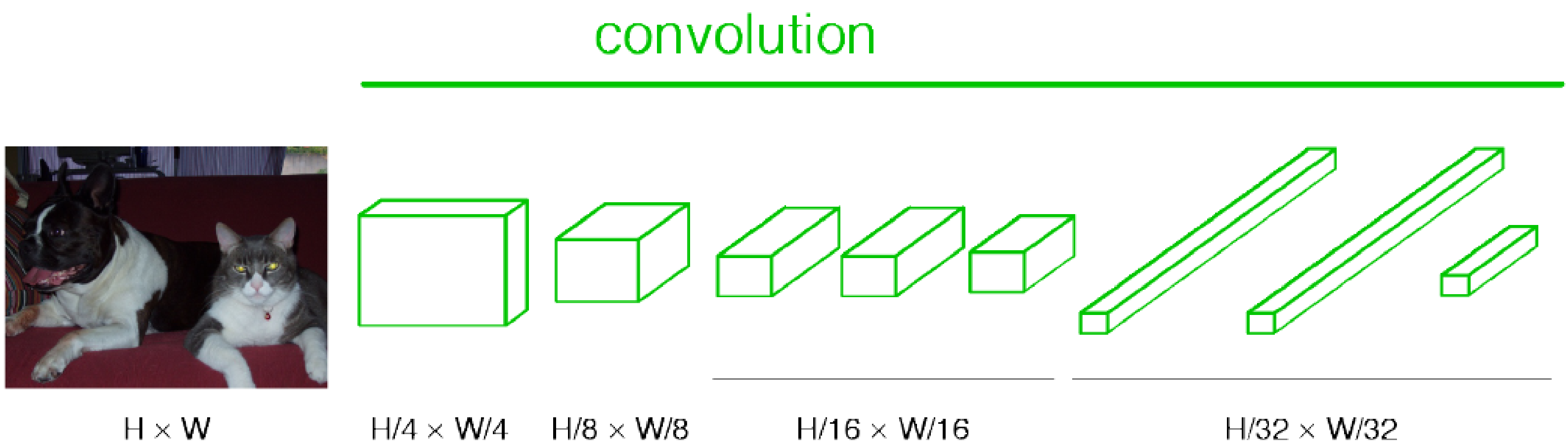


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Benefit of this thinking

- Mathematically elegant

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

Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult

Toeplitz Matrix

- Diagonals are constants

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

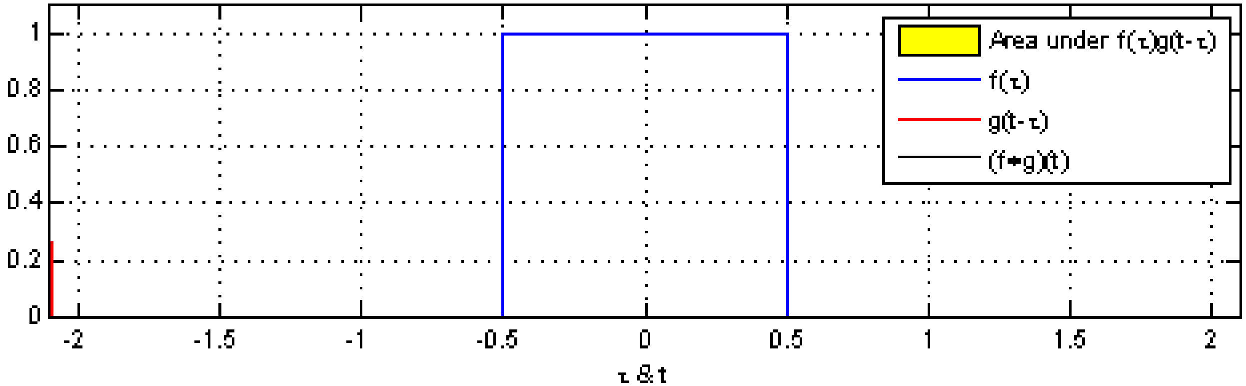
- $A_{ij} = a_{i-j}$

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

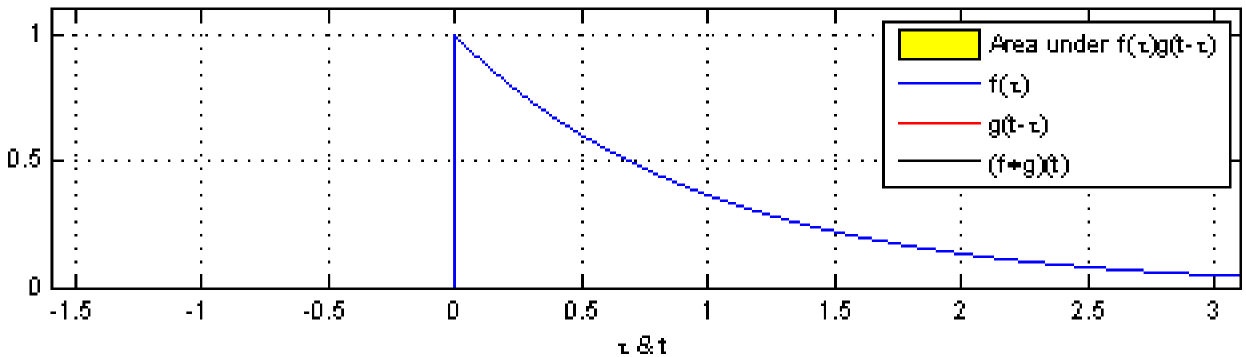
Why do we care?

- (Discrete) Convolution = Matrix Multiplication
  - with Toeplitz Matrices

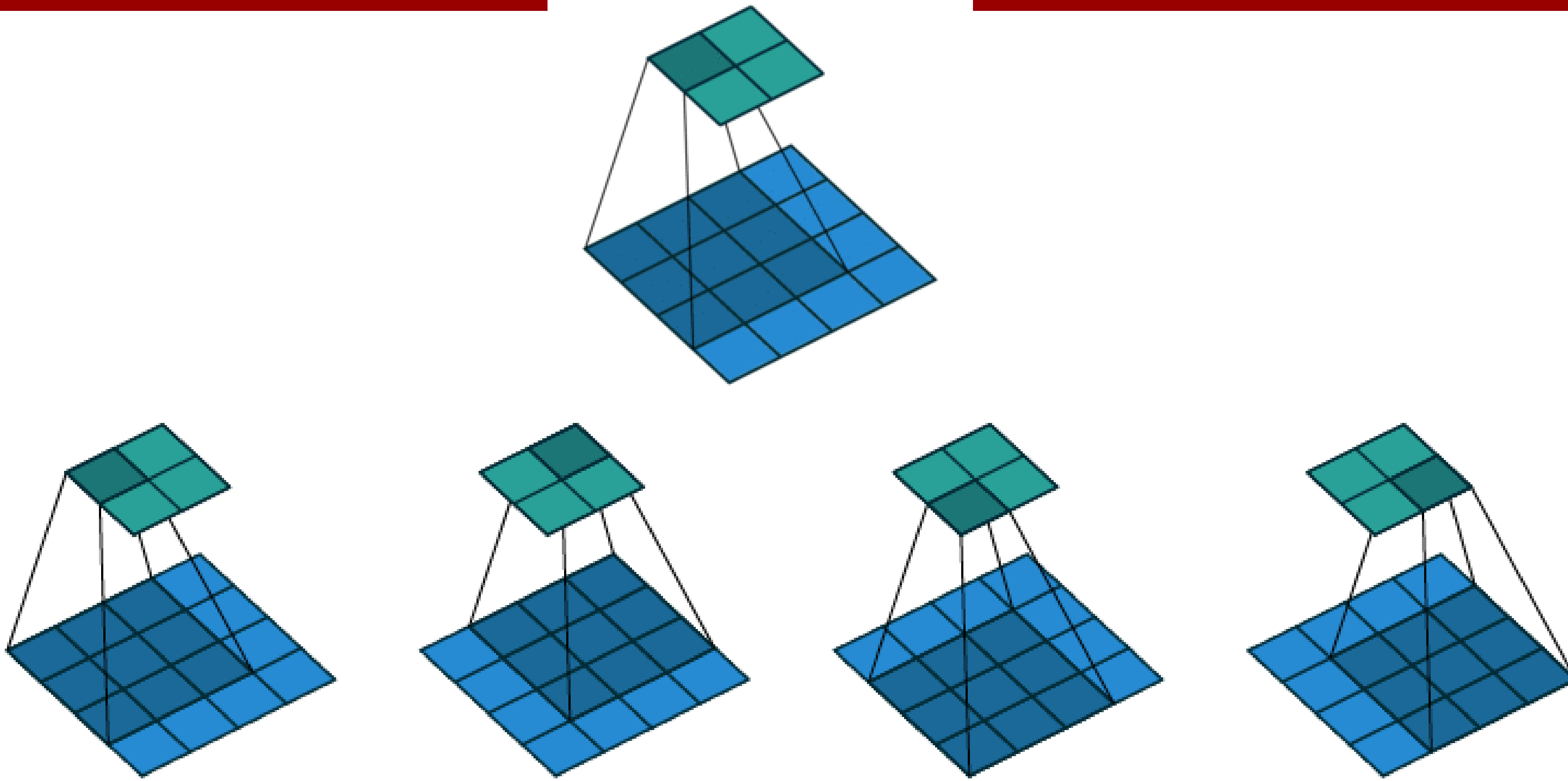
$$y = w * x \begin{bmatrix} w_k & 0 & \dots & 0 & 0 \\ w_{k-1} & w_k & \dots & 0 & 0 \\ w_{k-2} & w_{k-1} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_1 & \dots & \dots & w_k & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & w_1 & \dots & w_{k-1} & w_k \\ \vdots & \vdots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



1
0.8
0.6
0.4
0.2
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
τ & t
Area under f(τ)g(t-τ)
f(τ)
g(t-τ)
(f*g)(t)



1
0.5
0
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
τ & t
Area under f(τ)g(t-τ)
f(τ)
g(t-τ)
(f*g)(t)



$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Figure Credit: Dumoulin and Visin, https://arxiv.org/pdf/1603.07285.pdf