CS 4803 / 7643: Deep Learning

Topics:

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

Dhruv Batra

Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 10/02, 11:55pm

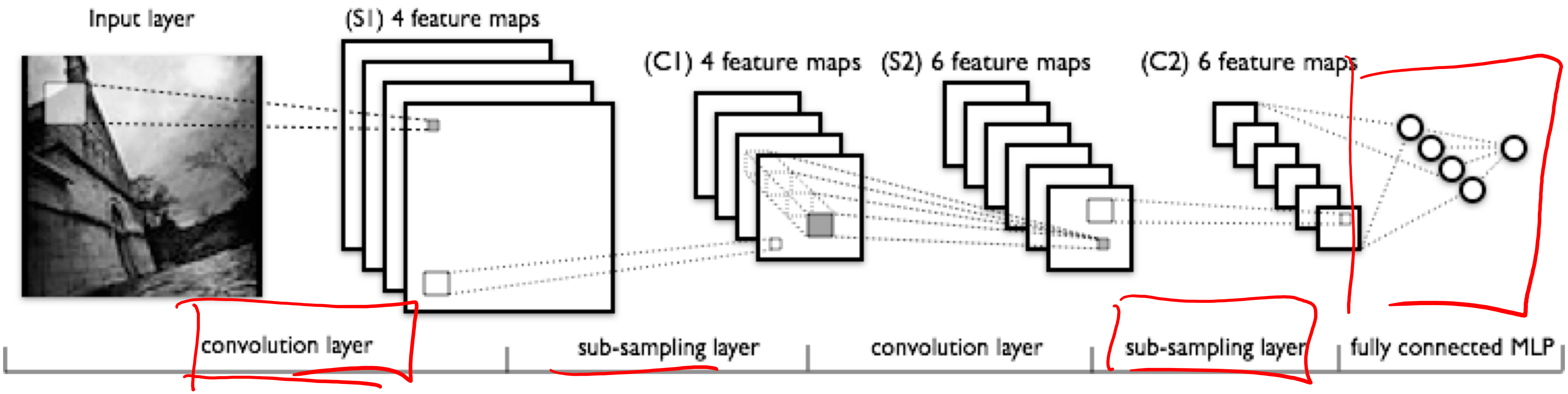
- Project Idea: ICLR19 Reproducibility Challenge
  - https://reproducibility-challenge.github.io/iclr_2019/
  - https://docs.google.com/spreadsheets/d/1BipWLvvWb7Fu6OSDd-uOCF1Lr_4drKOCRVdhxm_eSHc/edit#gid=0

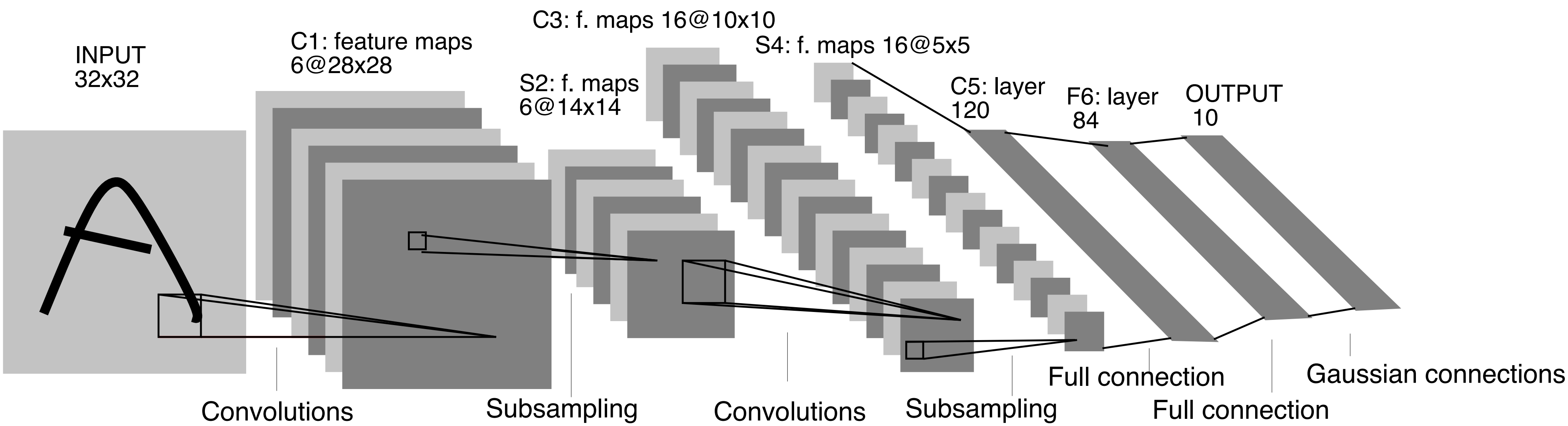
Recap from last time

Convolutional Neural Networks

(without the brain stuff)

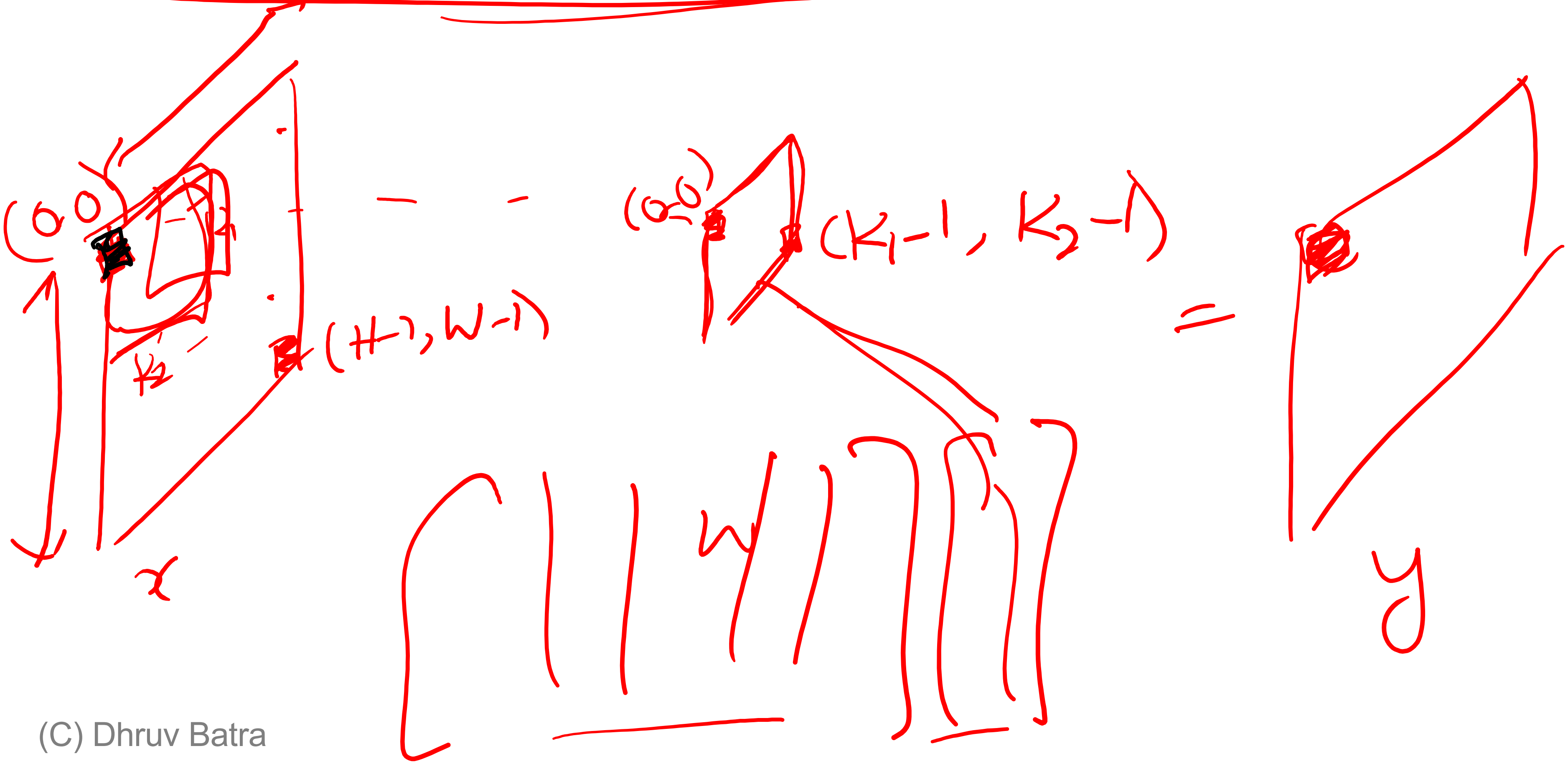
Convolutional Neural Networks



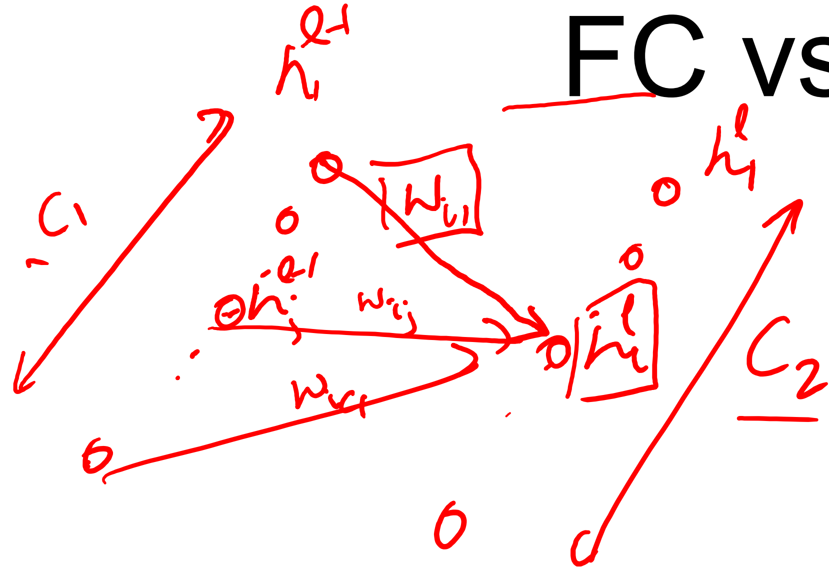


Convolutions for programmers

$$y[r,c] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} x[r+a, c+b]\, w[a,b]$$



FC vs Conv Layer

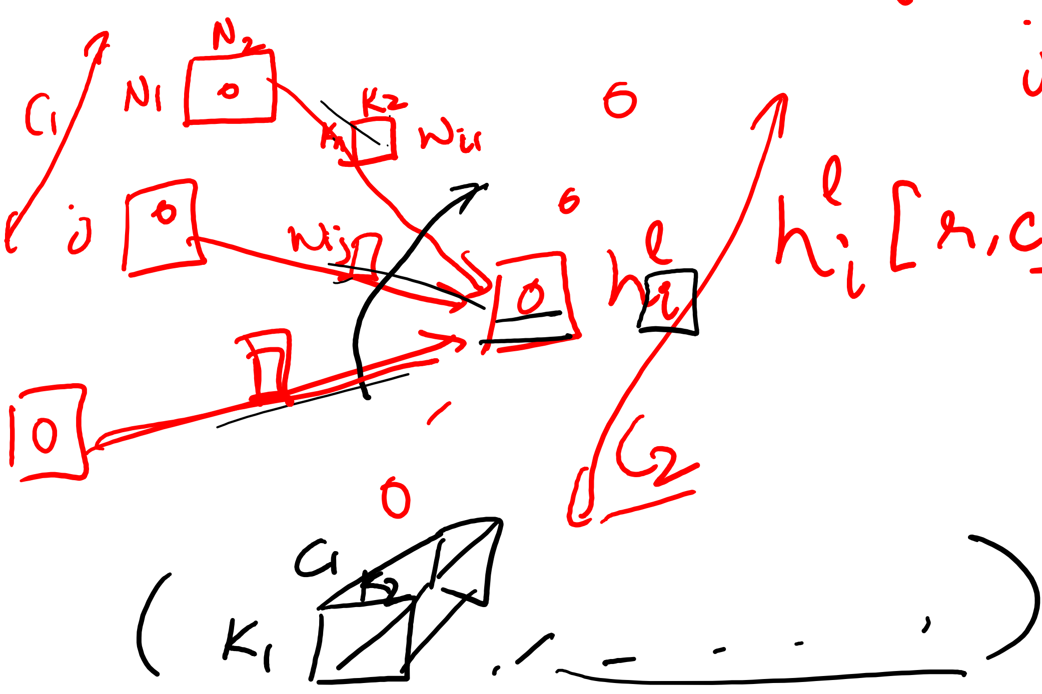


$$h_i^l = \sum_{j=1}^{C_1} h_j^{l-1} \cdot w_{ij} + b_i$$

Scalar product

conv

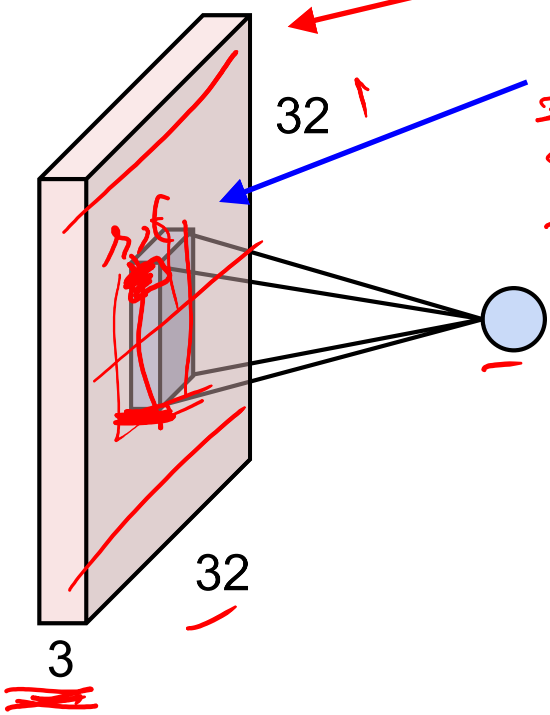
$$h_i^l = \sum_{j=1}^{C_1} h_j^{l-1} \ast W_{ij} + b_i$$



$$h_i^l[r,c] = \sum_{j=1}^{C_1} \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} h_j^{l-1}[r+a, c+b]\, w[a,b]$$

$$W \; (K_1 \times K_2 \times C_1 \times C_2)$$

Convolution Layer



32x32x3 image

5x5x3 filter w

3x3

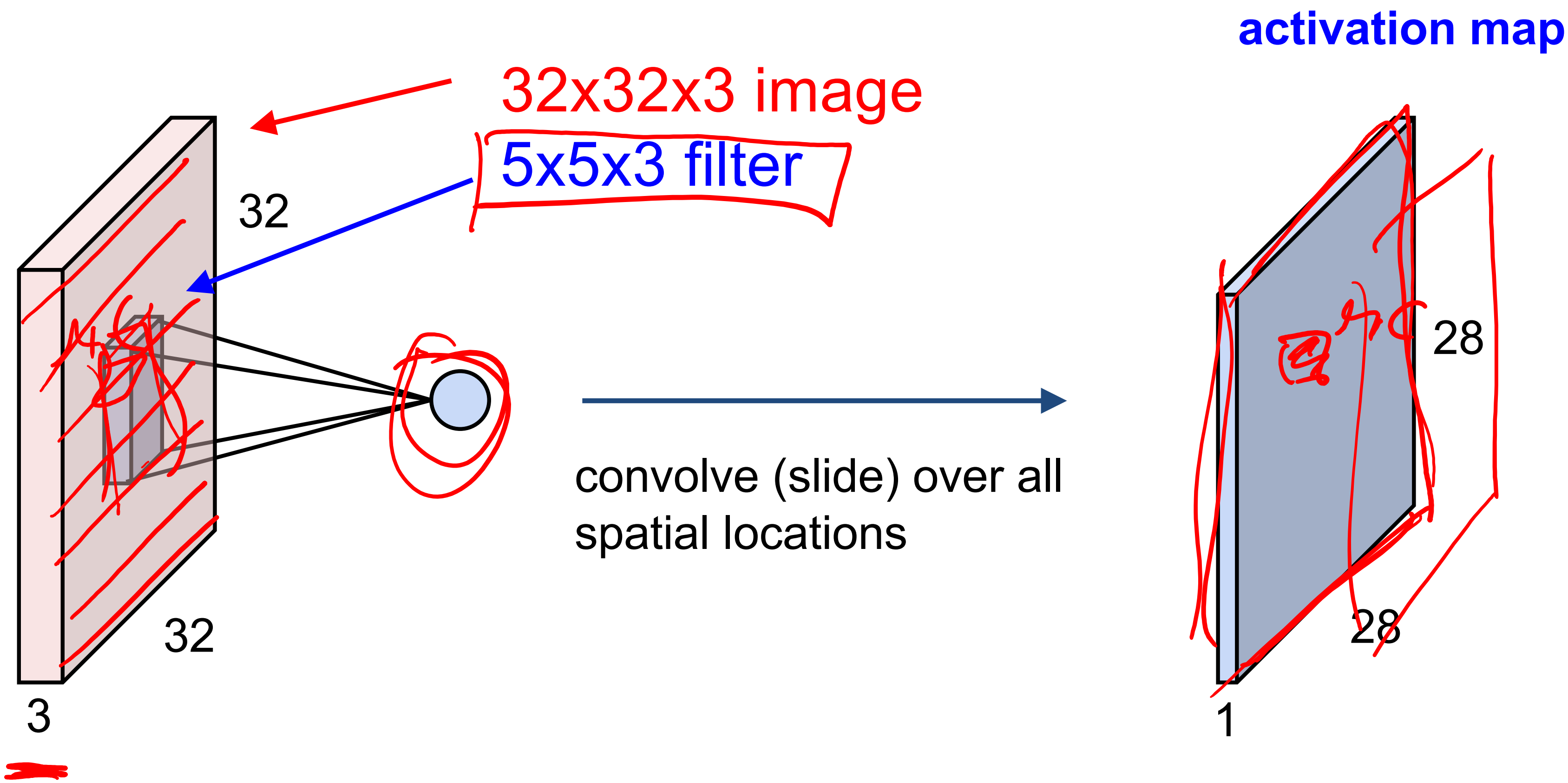
7x7

1 number:

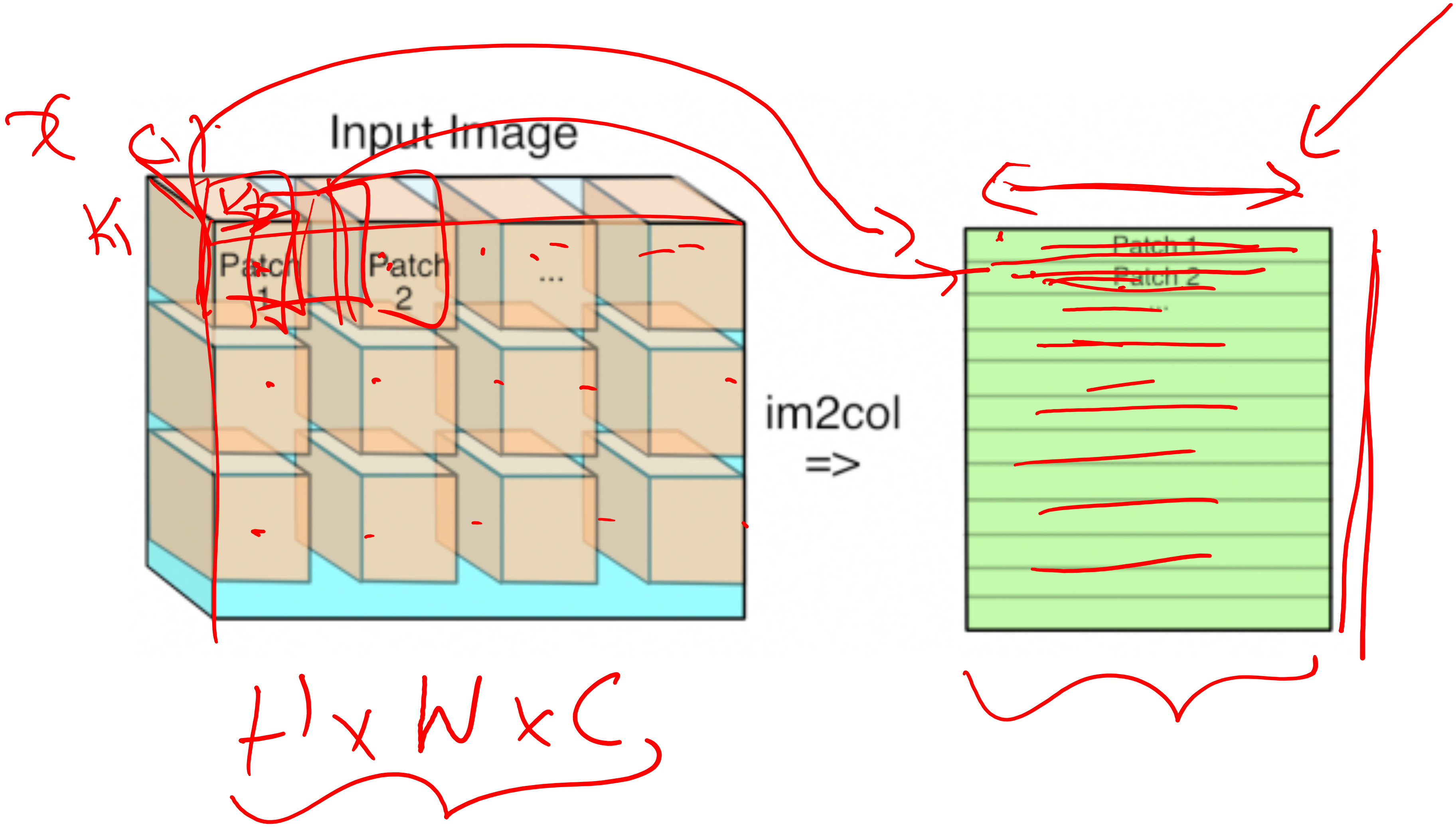
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer



Im2Col



GEMM

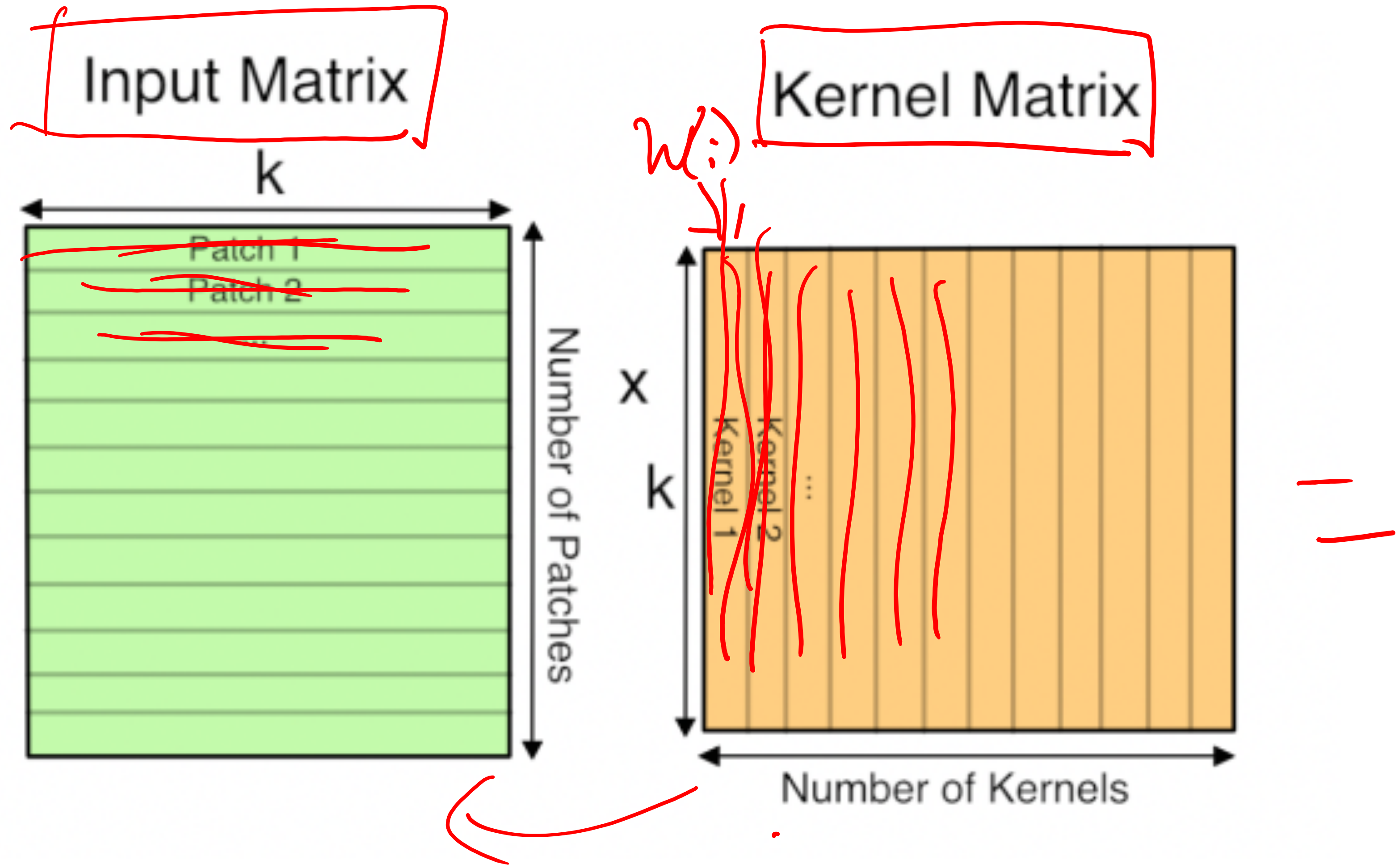
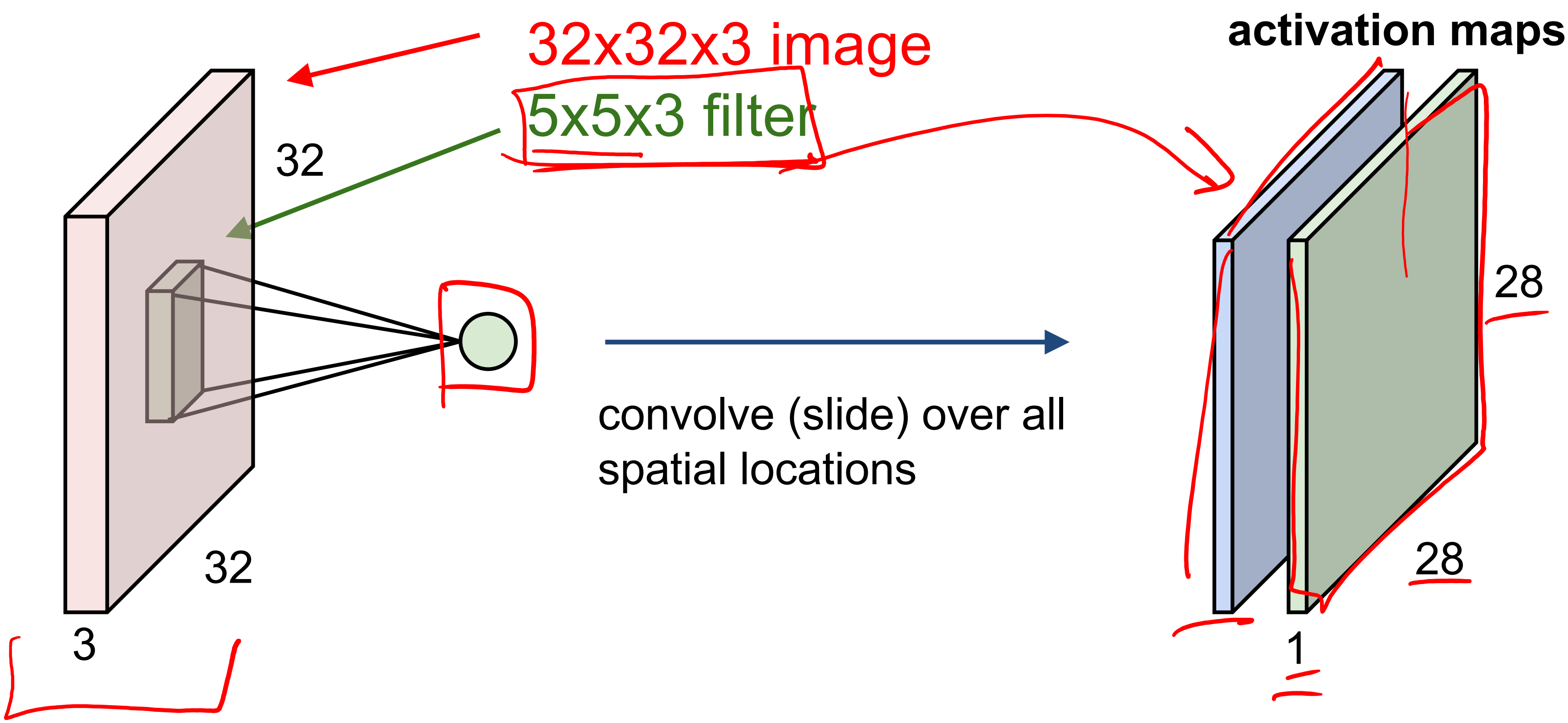


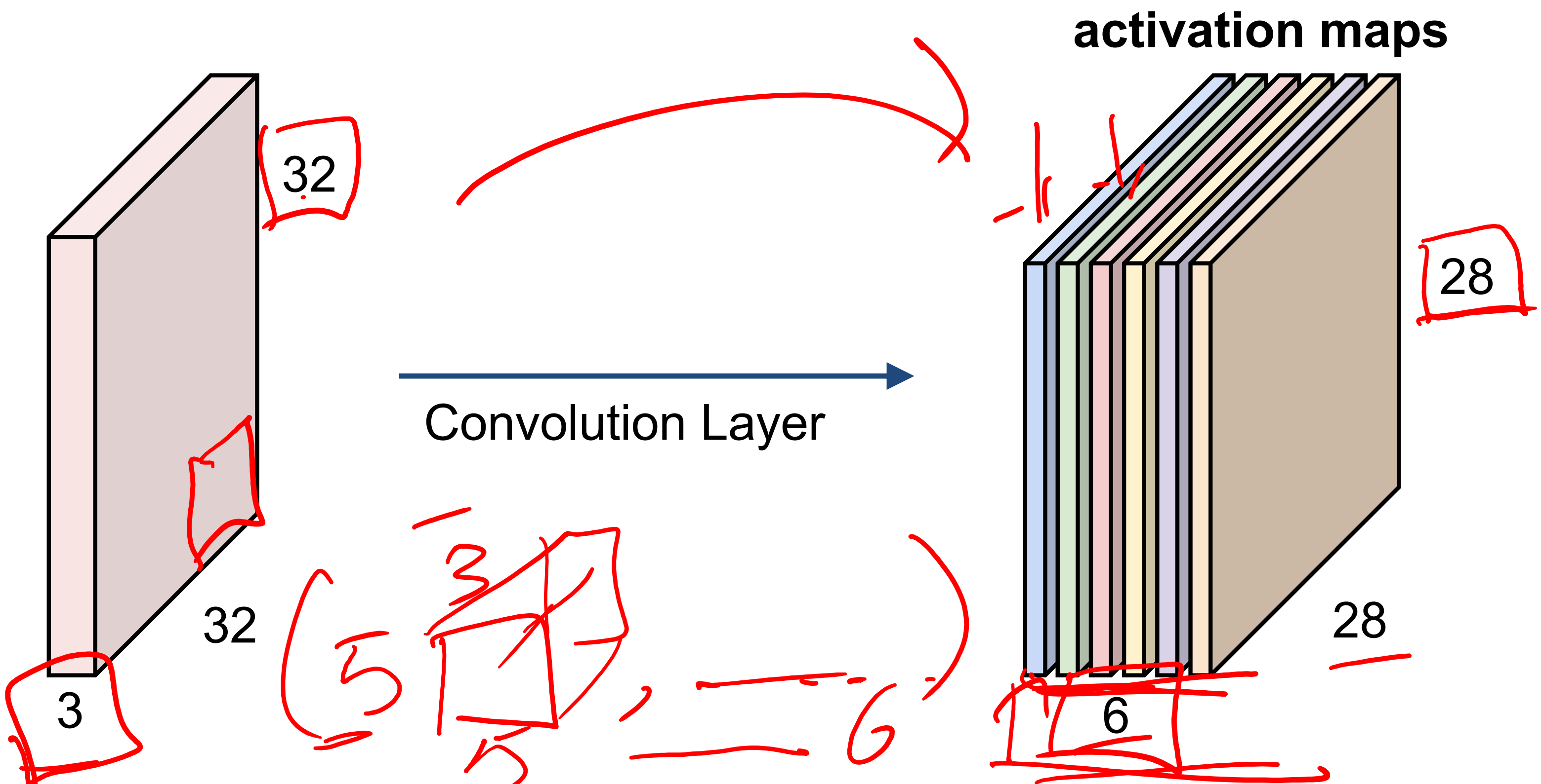
 Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Convolution Layer

consider a second, green filter

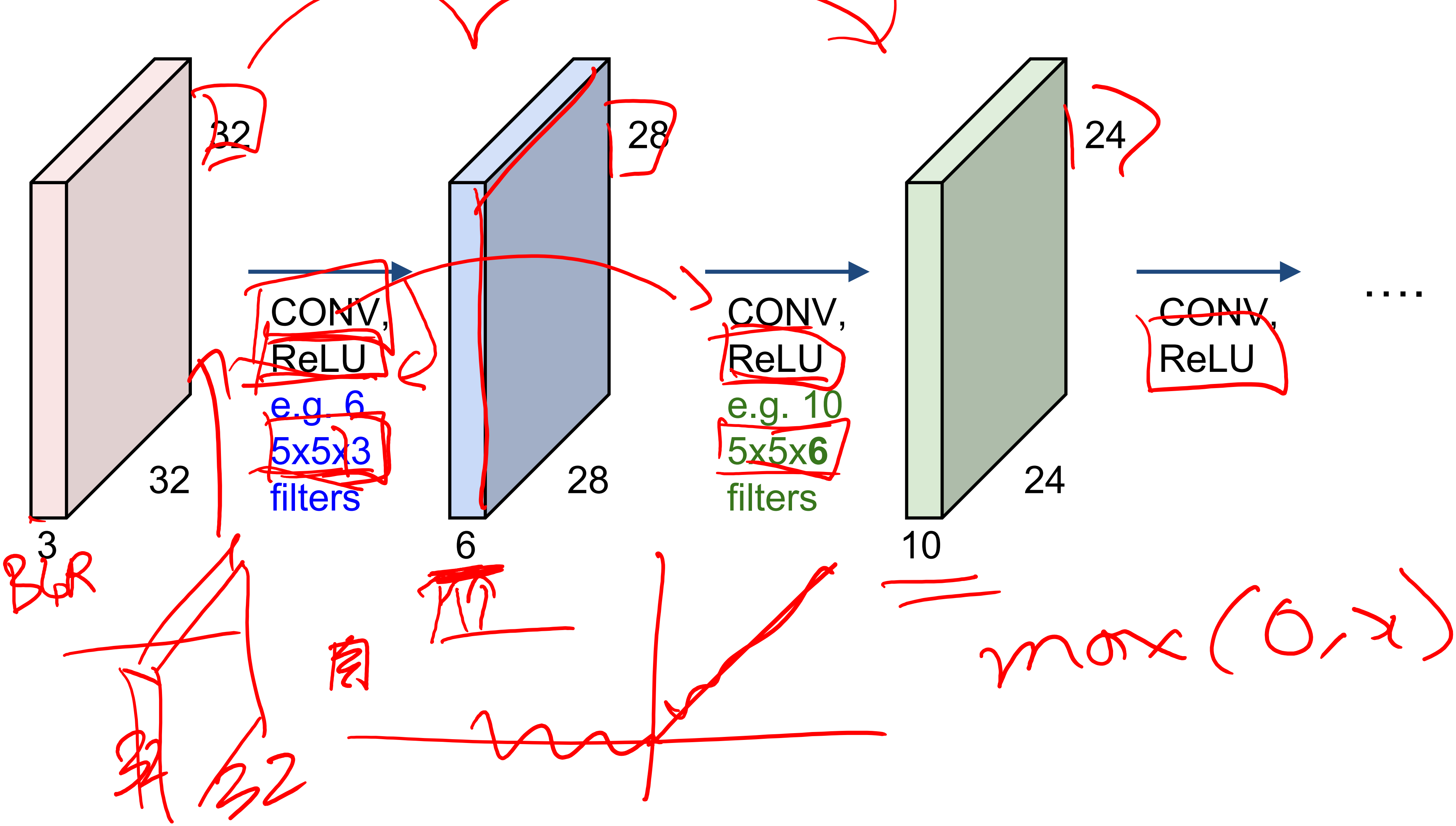


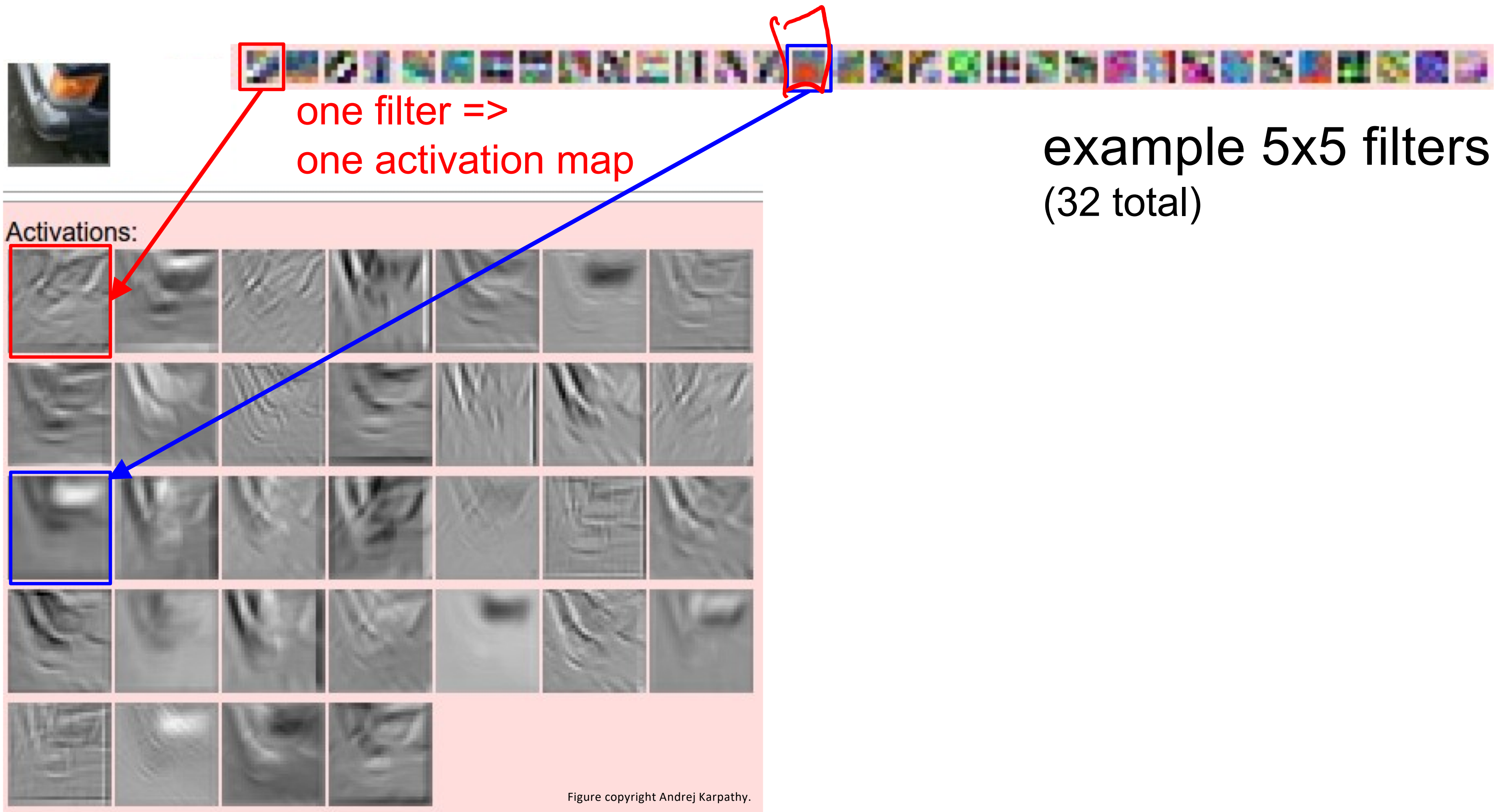
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

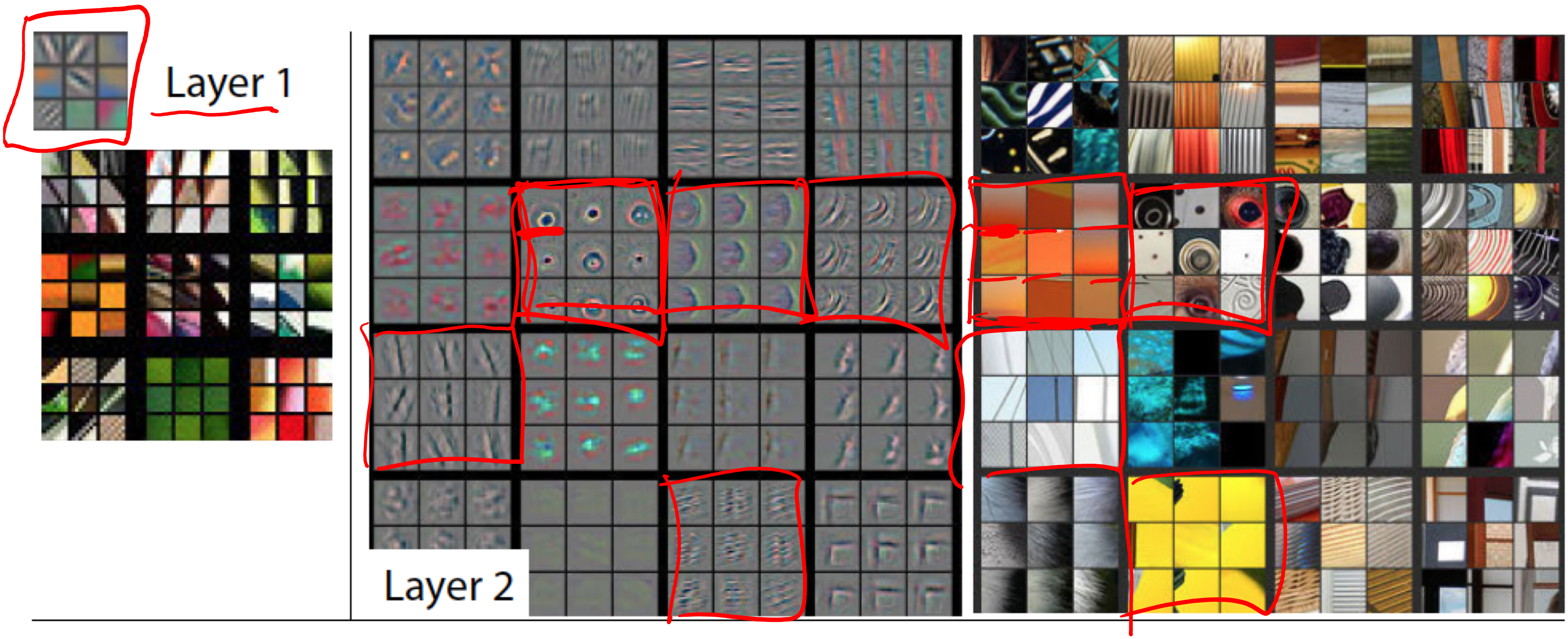
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions





example 5x5 filters
(32 total)

Visualizing Learned Filters



Visualizing Learned Filters

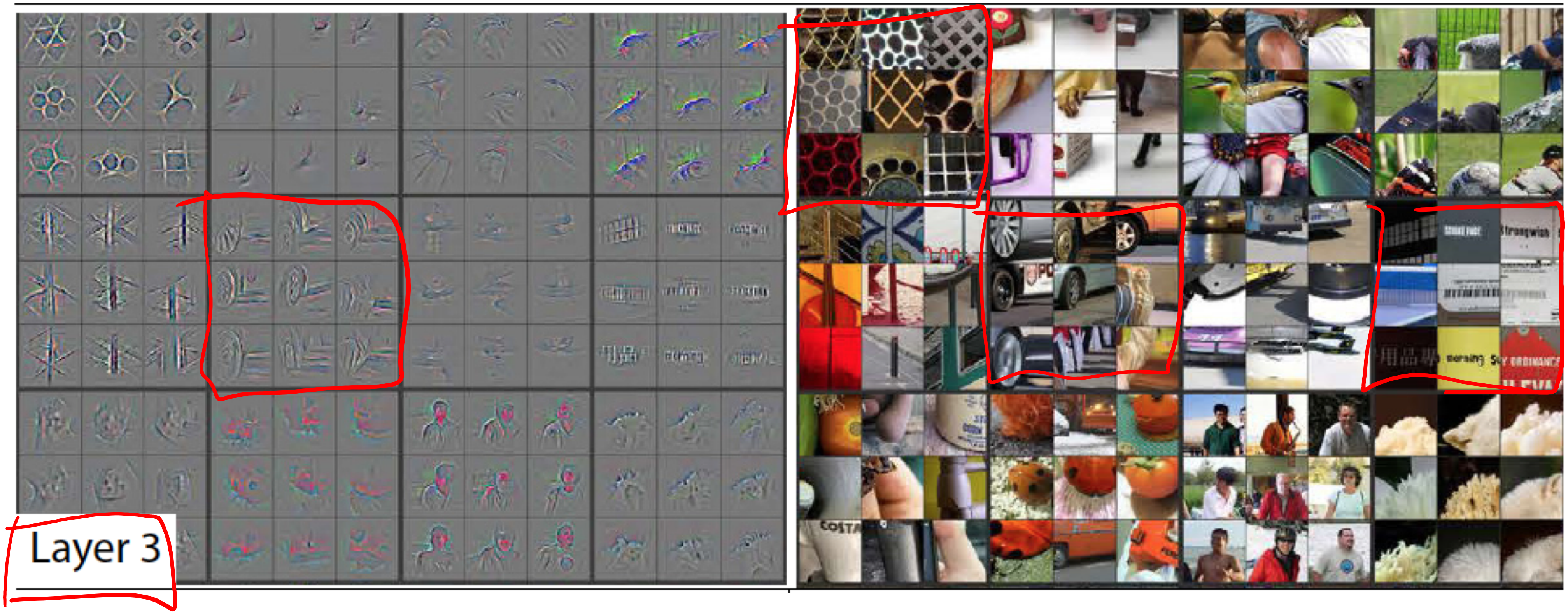
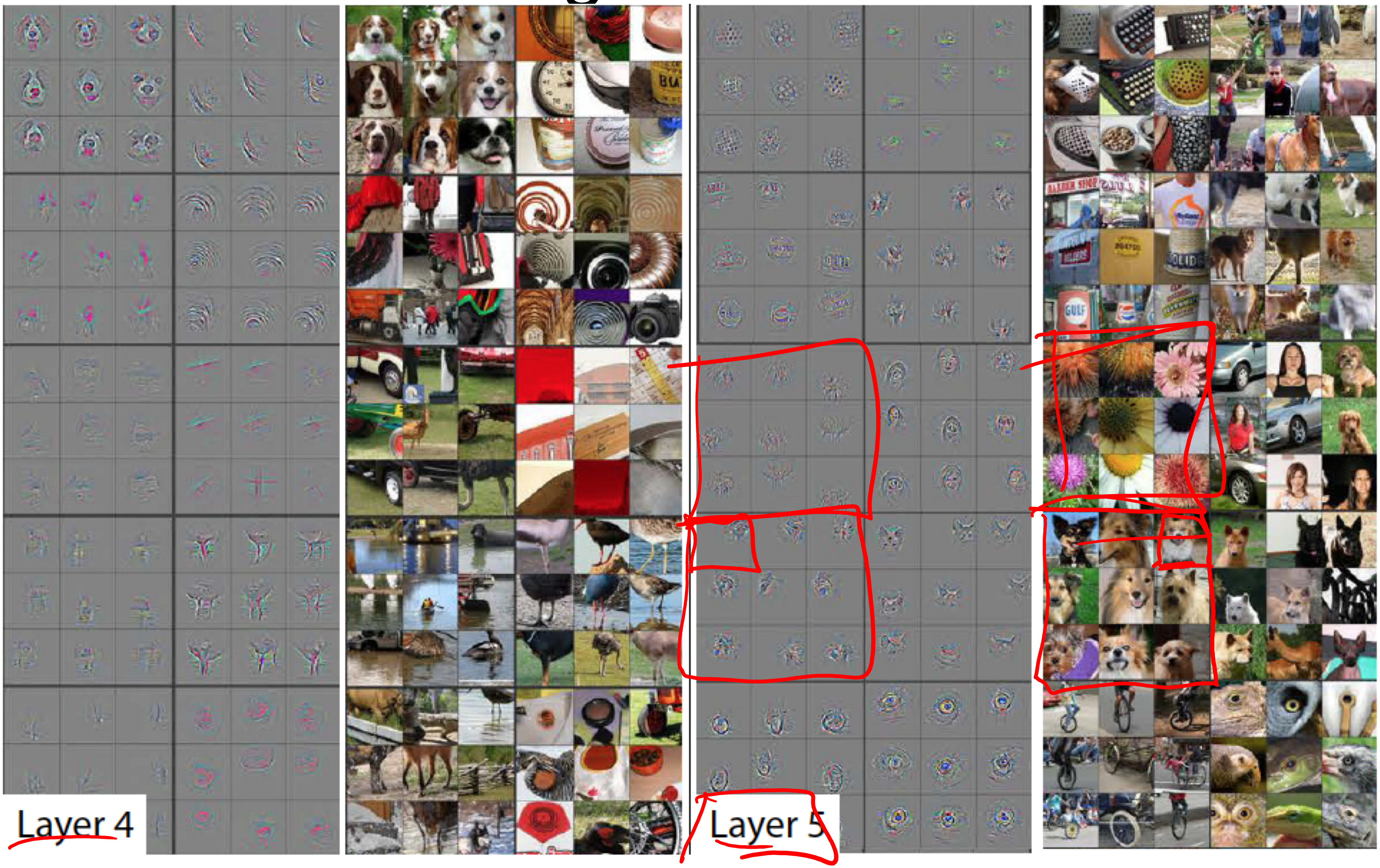


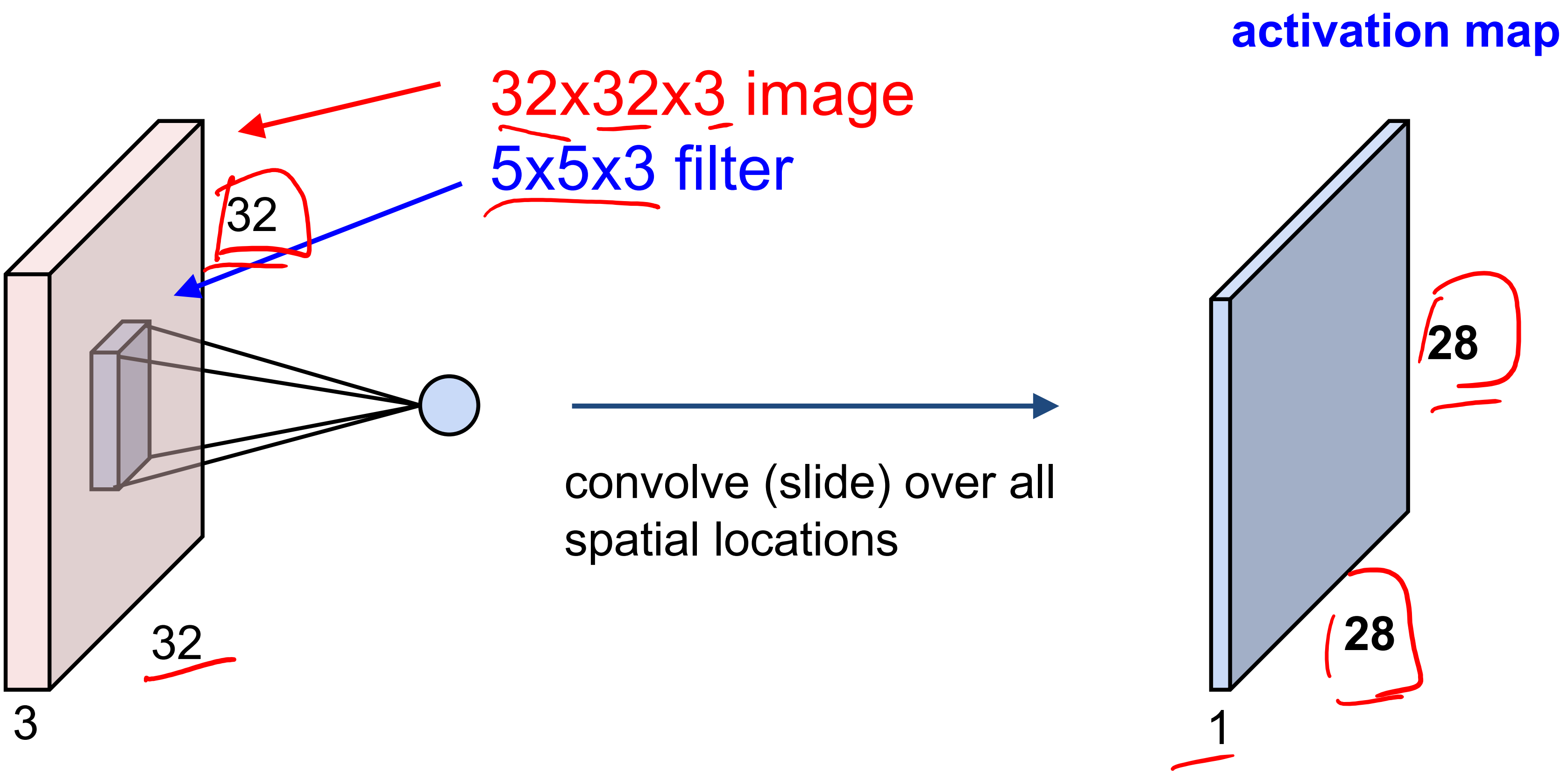
Figure Credit: [Zeiler & Fergus ECCV14]

Visualizing Learned Filters

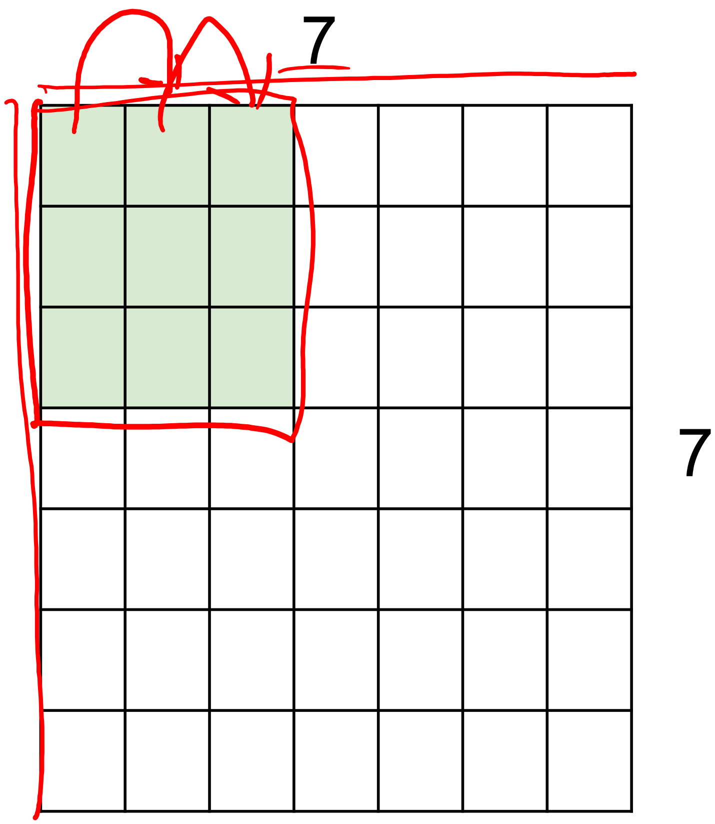
256



A closer look at spatial dimensions:



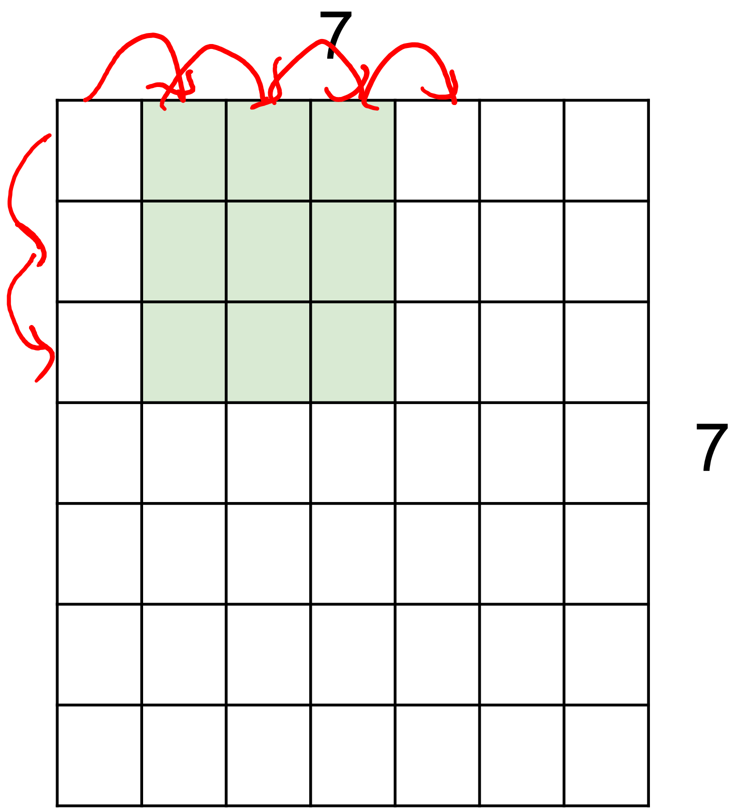
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

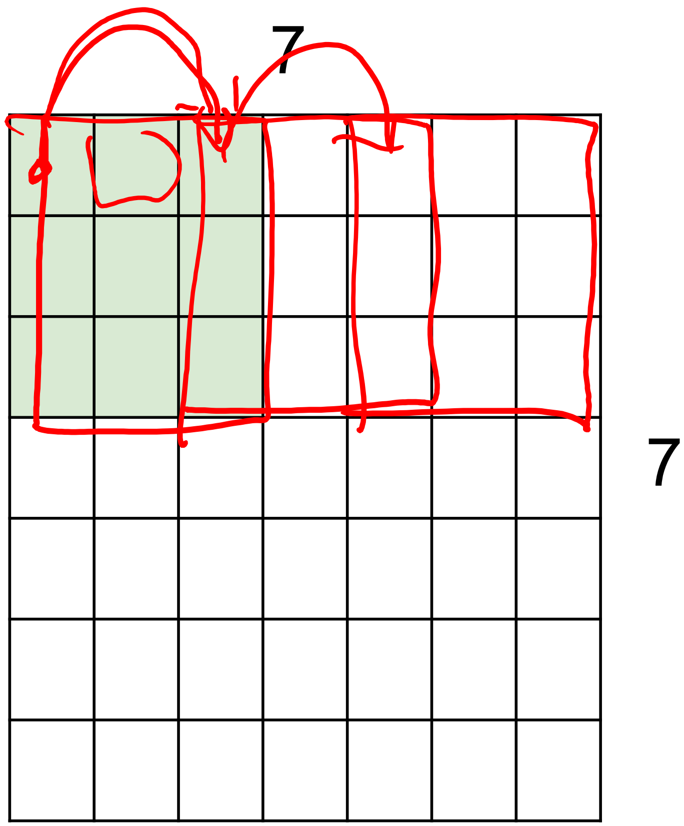
A closer look at spatial dimensions:

stride = 1



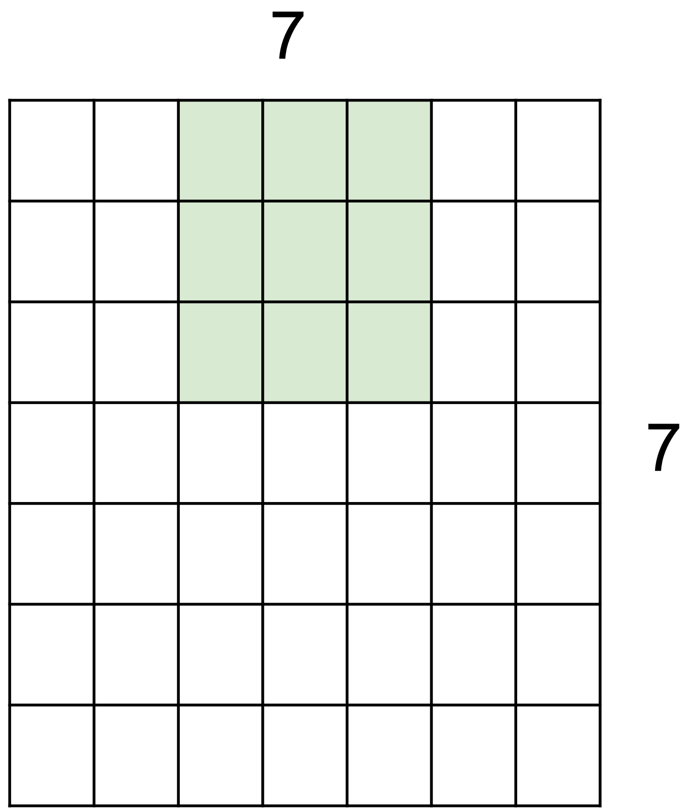
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



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

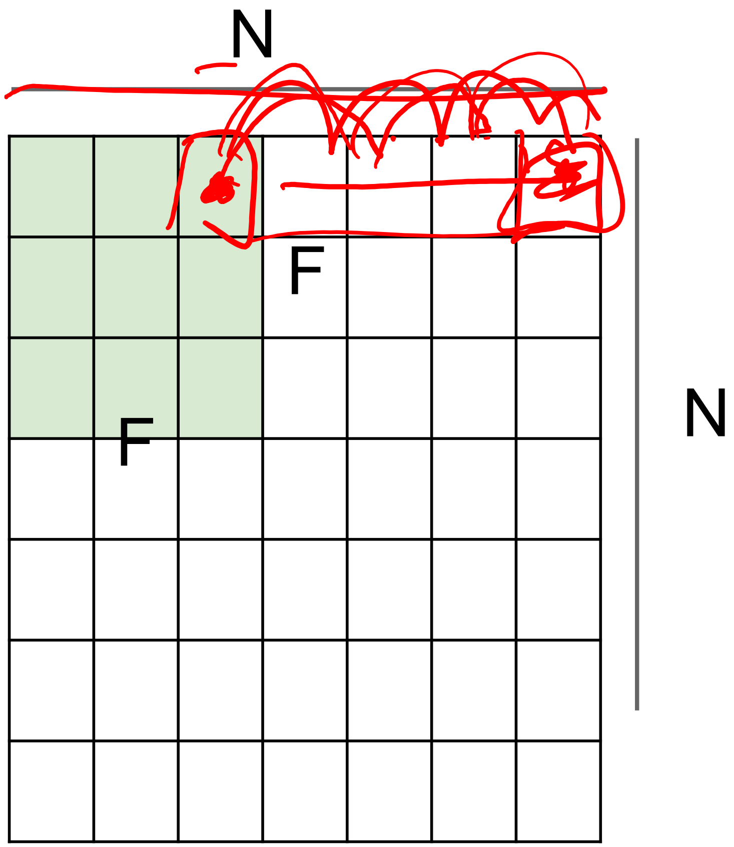
A closer look at spatial dimensions:



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

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

$\frac{N-F}{\text{stride}} + 1$



Output size:

$\boxed{\textbf{(N - F) / stride + 1}}$

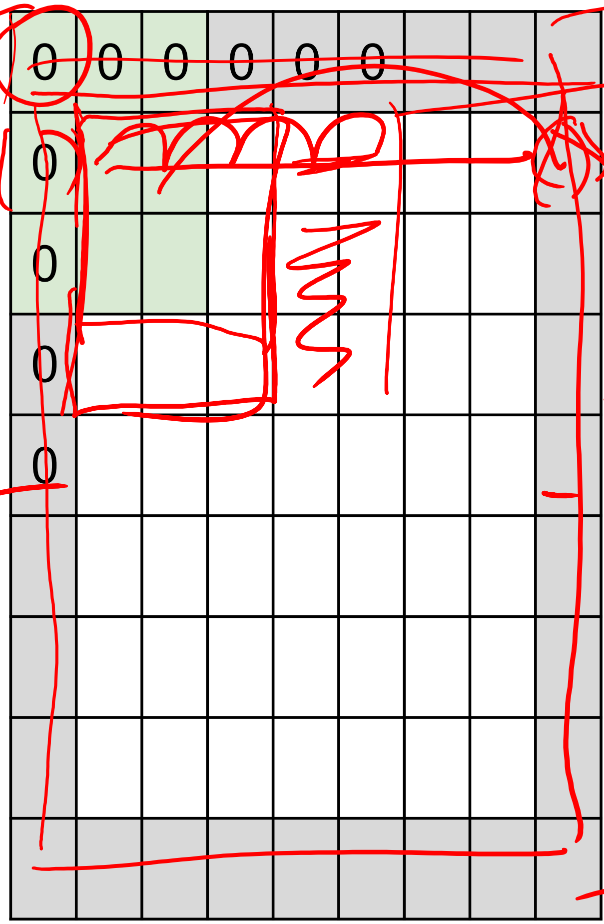
e.g. N = 7, F = 3:

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

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

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

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?

$N \Rightarrow \hat{N}$

$7\times7 \to 9\times9$

$\frac{9-3}{1} + 1$

$\boxed{7\times7}$

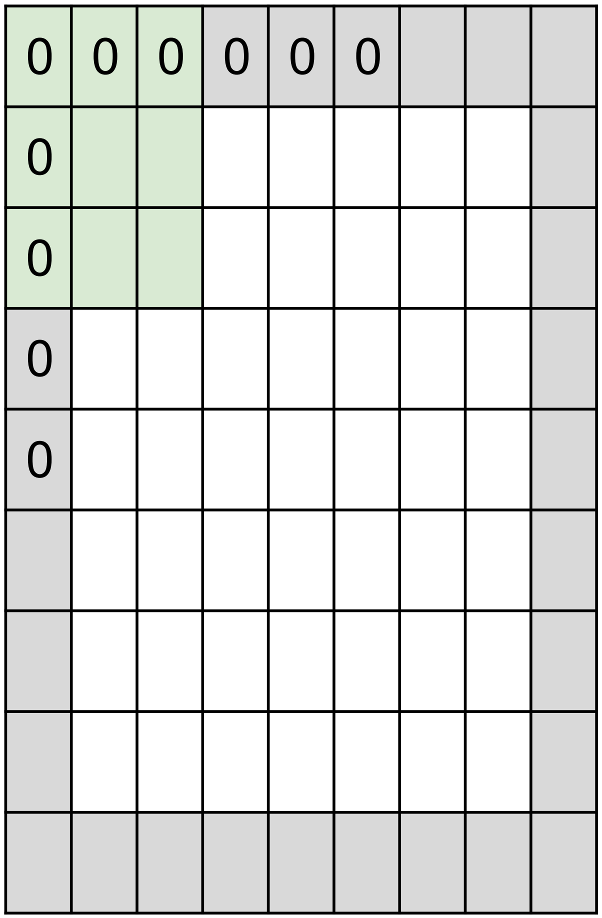
(recall:)

(N - F) / stride + 1

$\frac{N - F + 2\,pad}{stride} + 1$

$\frac{F-1}{2}$

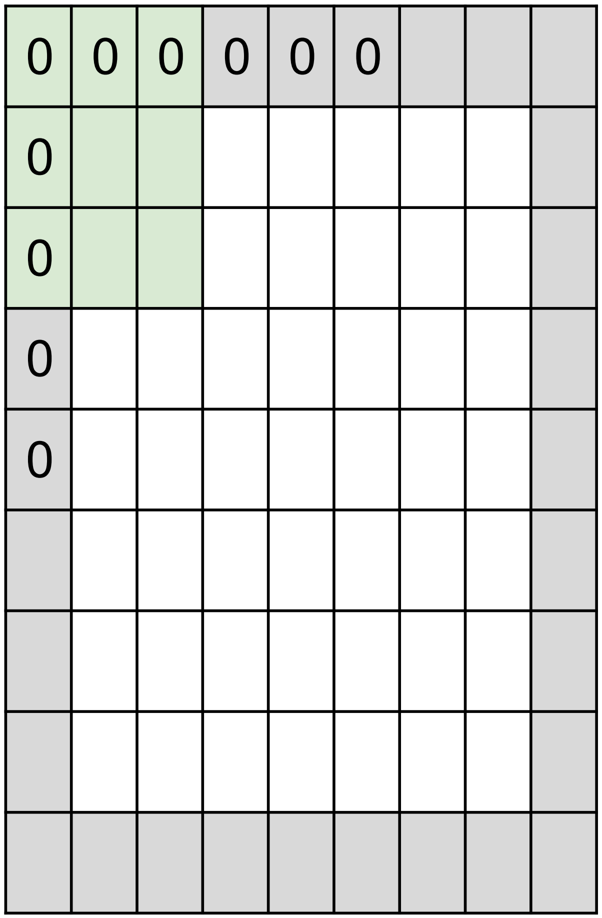
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

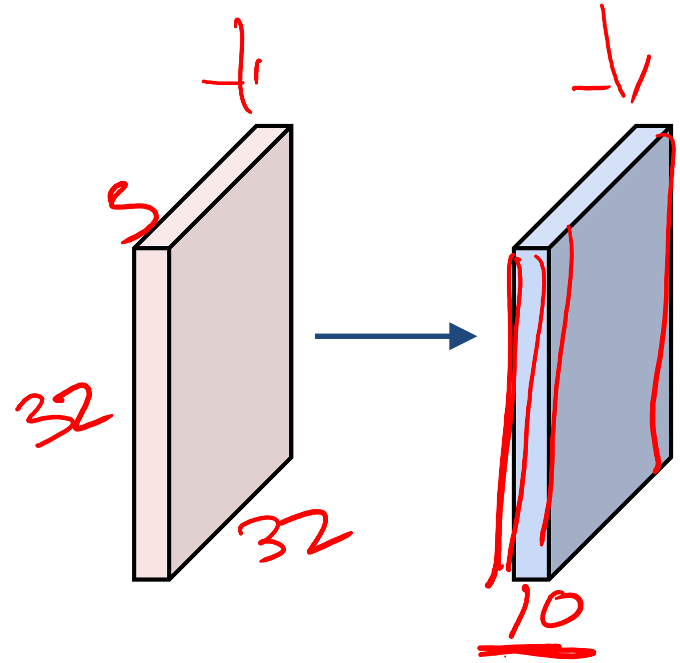


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

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

Examples time:

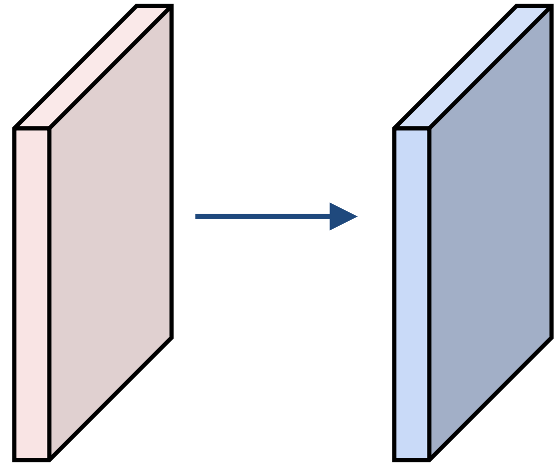


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

$$\underbrace{(32)}_{N_1} \times \underbrace{(32)}_{N_2} \times \underbrace{(10)}_{C_2}$$

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

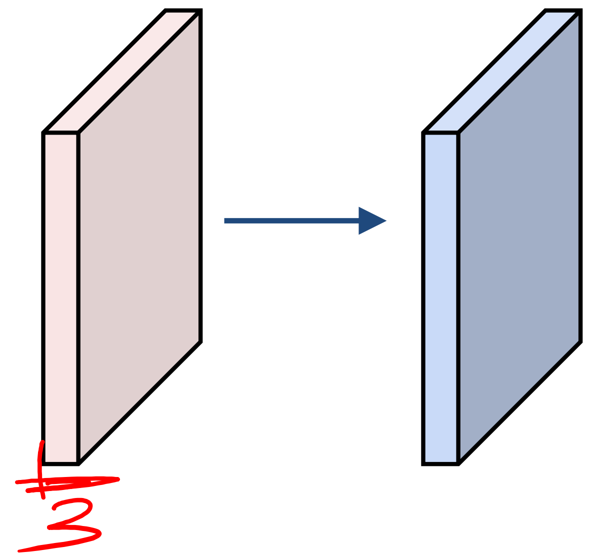
32x32x10

Examples time:

C_1

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

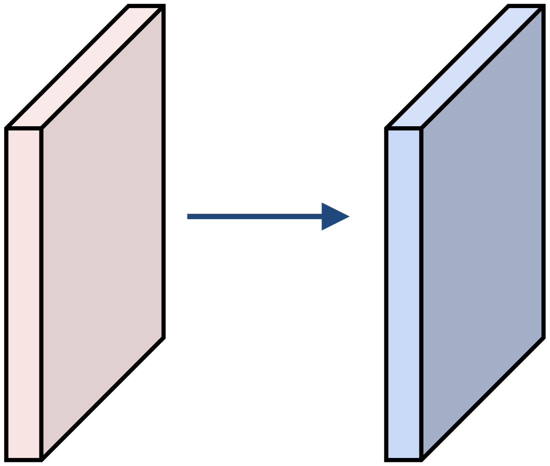
$(5\times5\times3+1)\times10 = 760$

$(\ )\times(\ )\times(\ )\times(\ )$

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 $\boxed{(F \cdot F \cdot D_1) \cdot K}$ 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:

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.

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

Example: CONV layer in Torch

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```


Summary. To summarize, the Conv Layer:

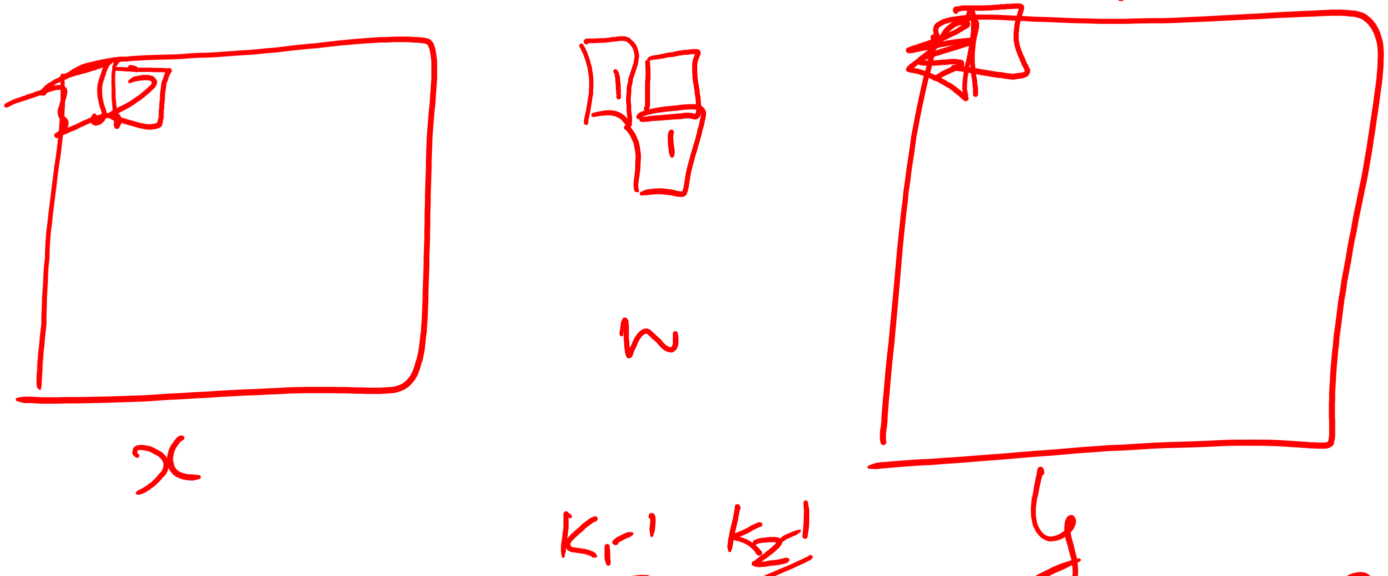
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

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

Plan for Today

- Convolutional Neural Networks
  - 1x1 convolutions
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers
  - Toeplitz matrices and convolutions = matrix-mult
  - Dilated/a-trous convolutions

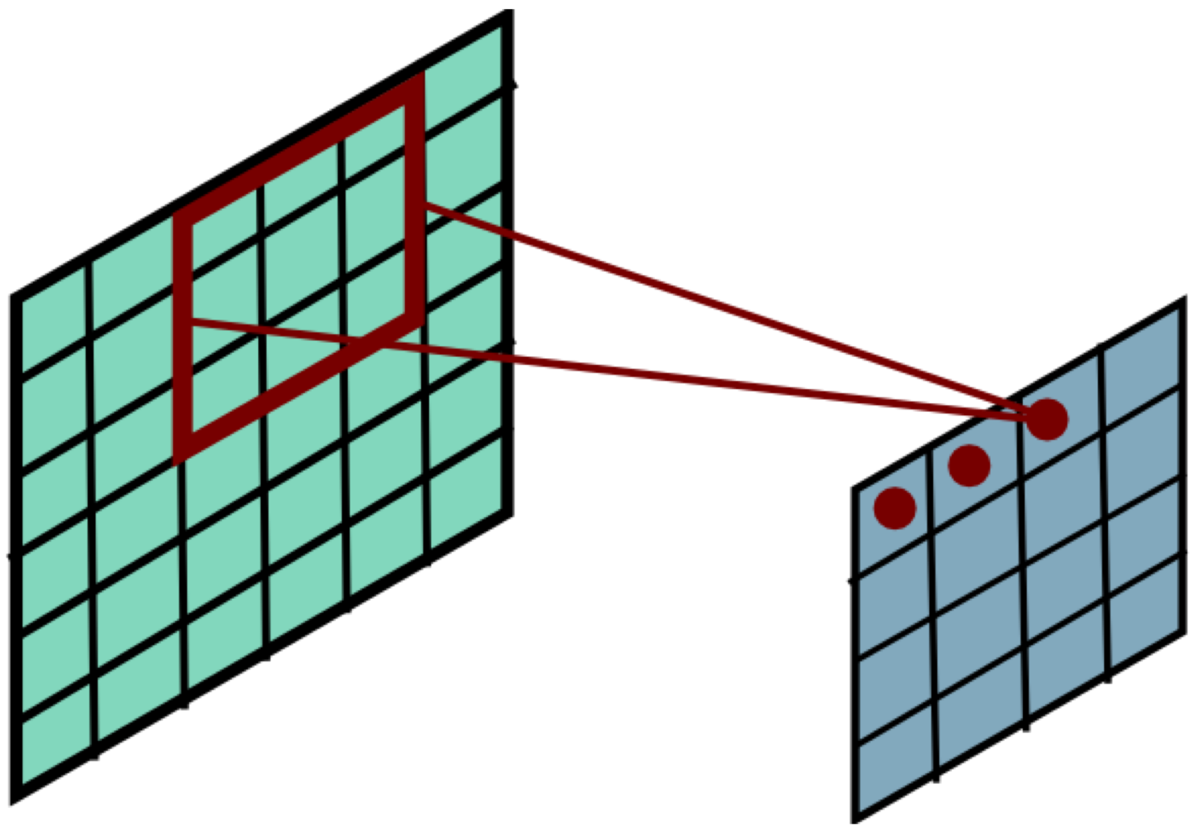
Can we have $\boxed{1\text{x}1}$ filters?



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

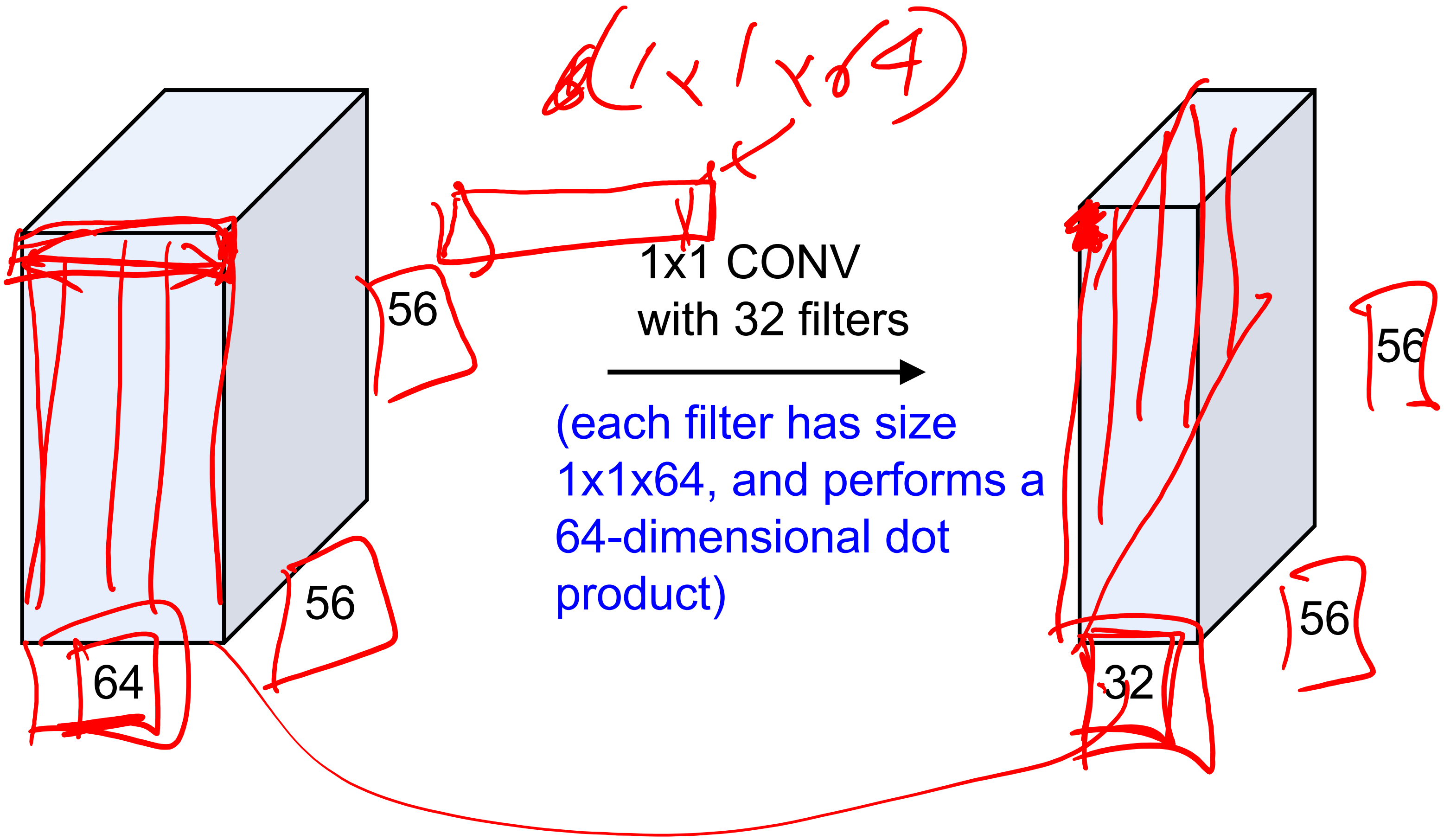
$$\boxed{y[r,c] = x[r,c]} \cdot \underbrace{w[0,0]}$$

Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

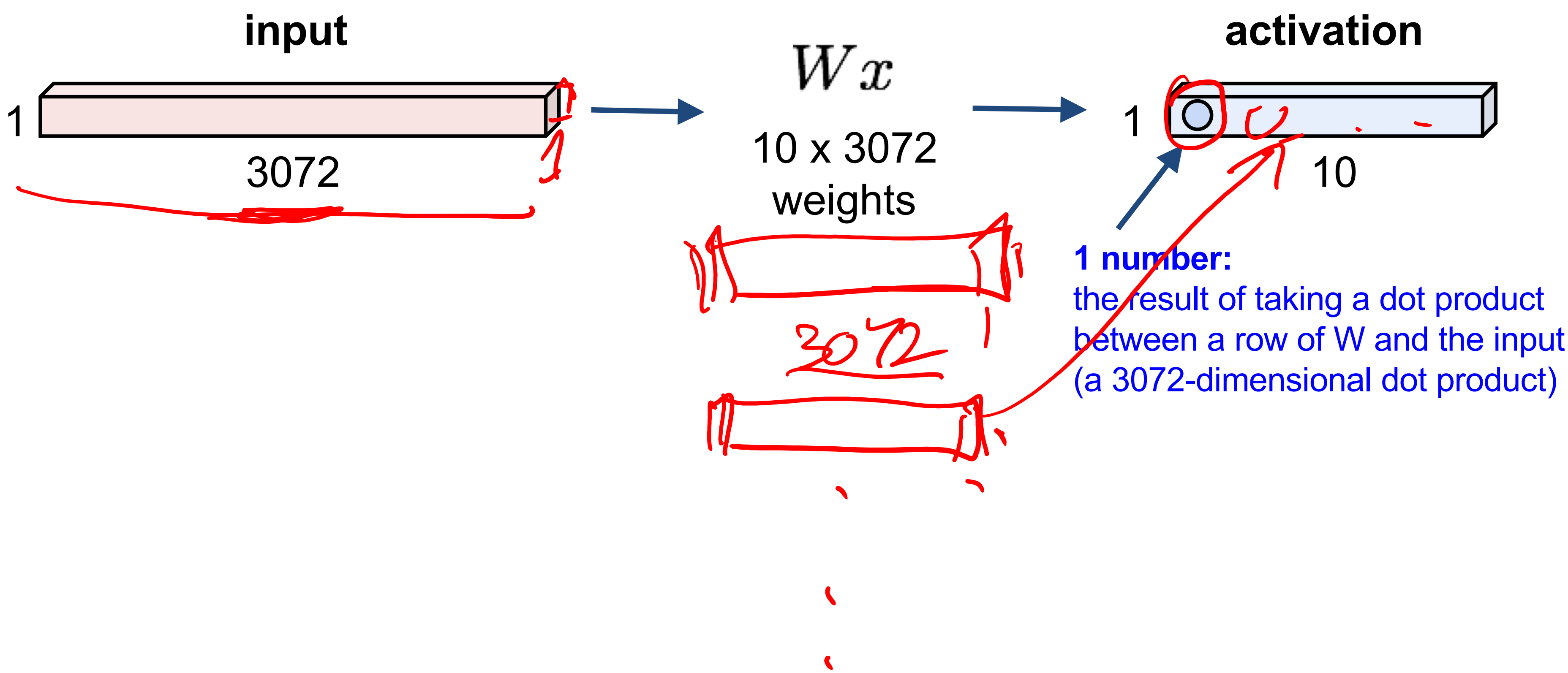
(btw, 1x1 convolution layers make perfect sense)



Reminder: Fully Connected Layer

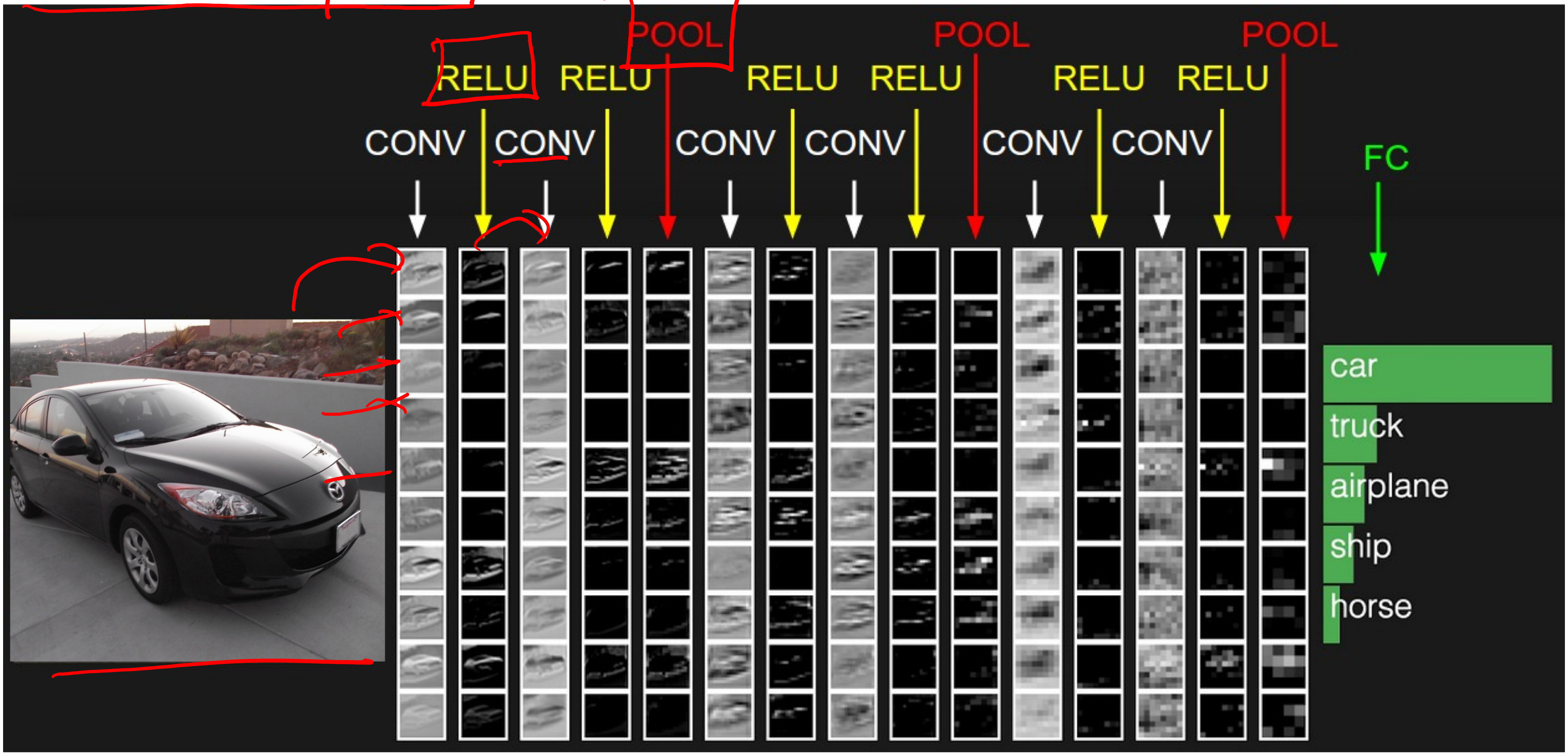
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



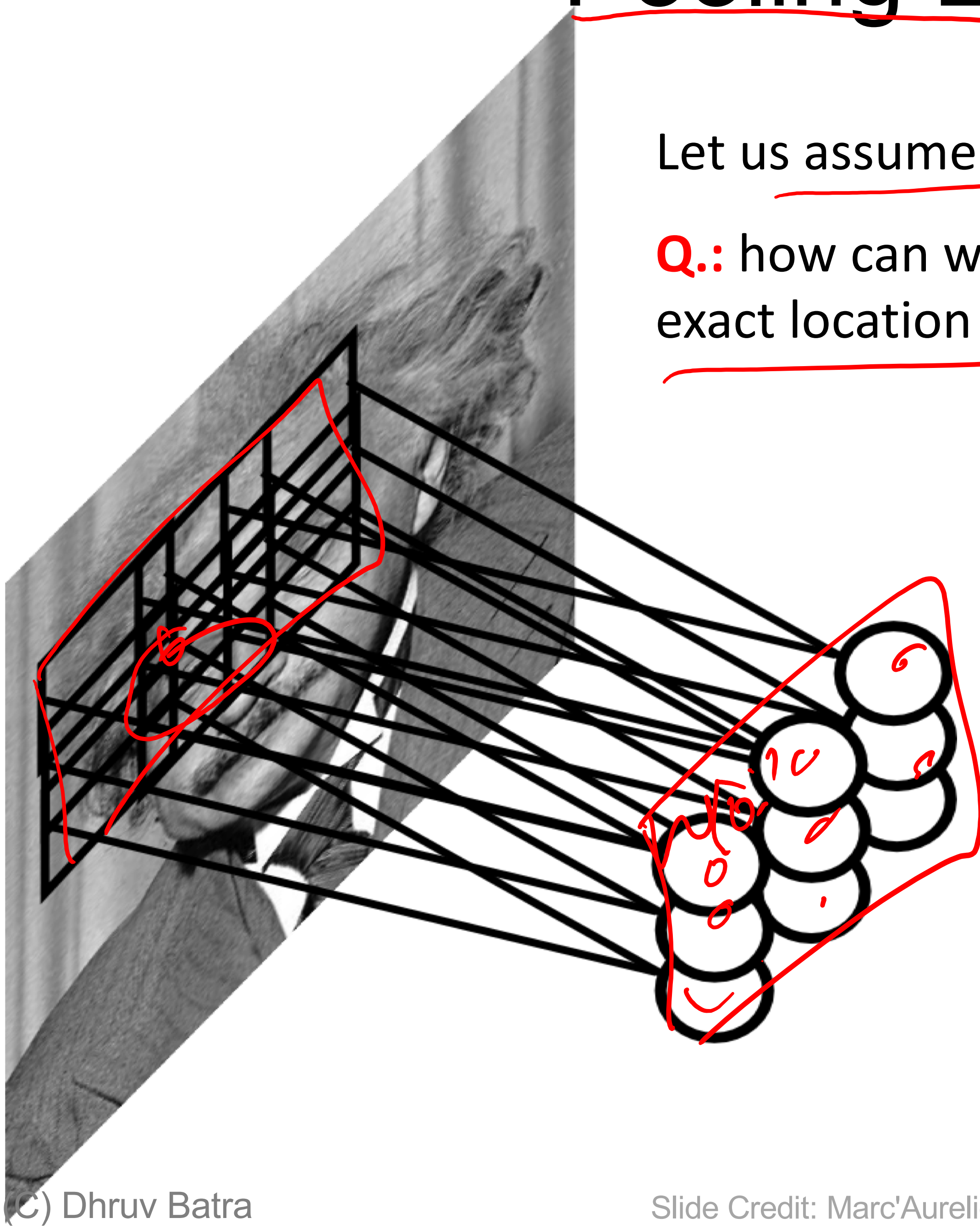
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

two more layers to go: POOL/FC



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

Pooling Layer



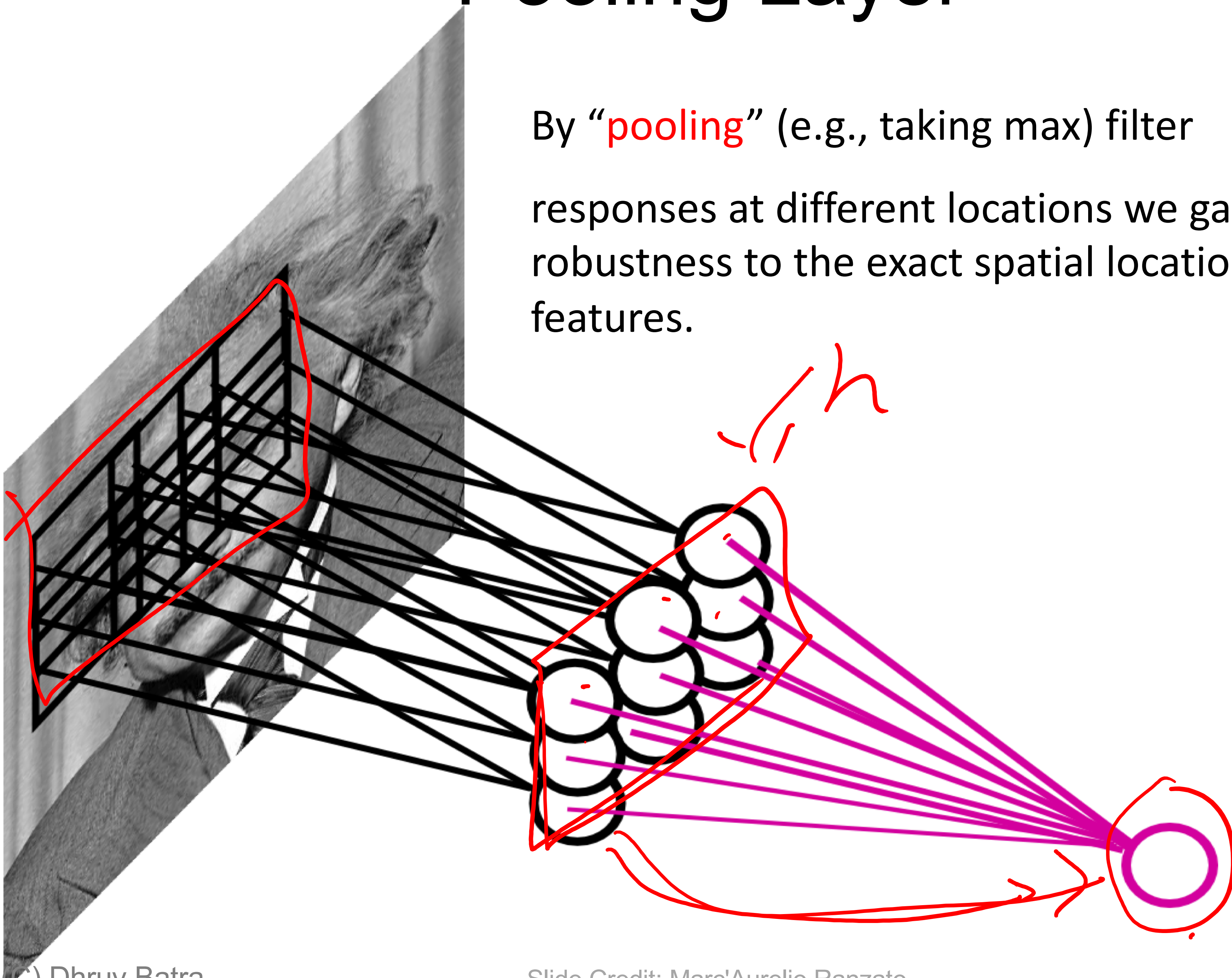
Let us assume filter is an "eye" detector.

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

Pooling Layer

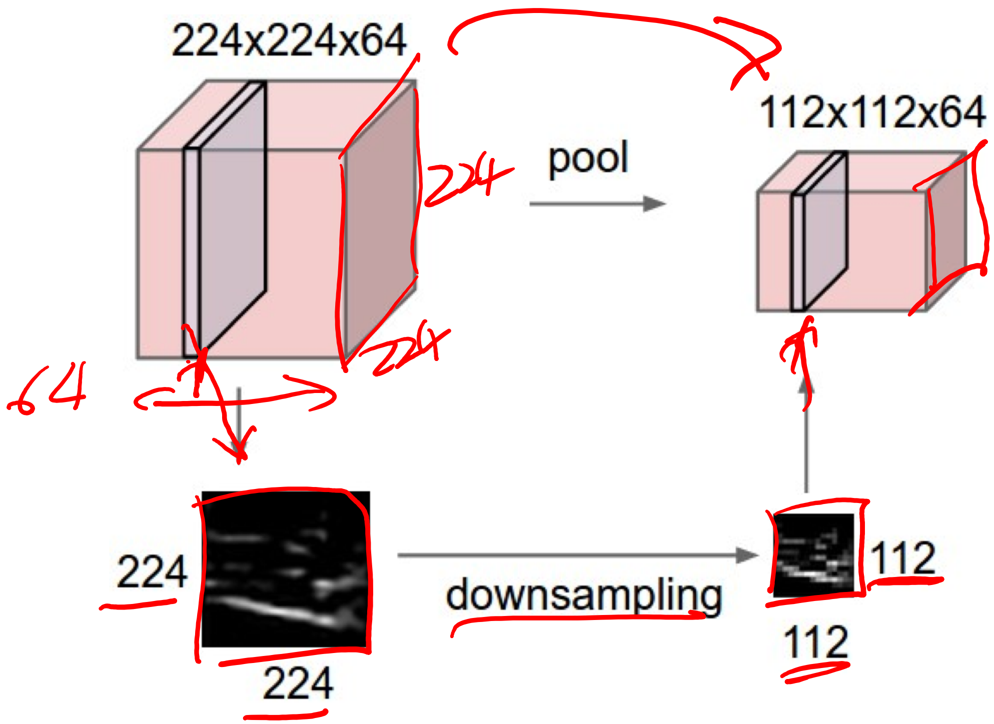
By “pooling” (e.g., taking max) filter

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:



stride=2

MAX POOLING 2x2

Single depth slice

dim 1 ↓, dim 2 →

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2 →

6	8
3	4

$$y[r,c] = \boxed{\max_a \max_b} \, x[r+a, c+b]$$

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

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:

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

- 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

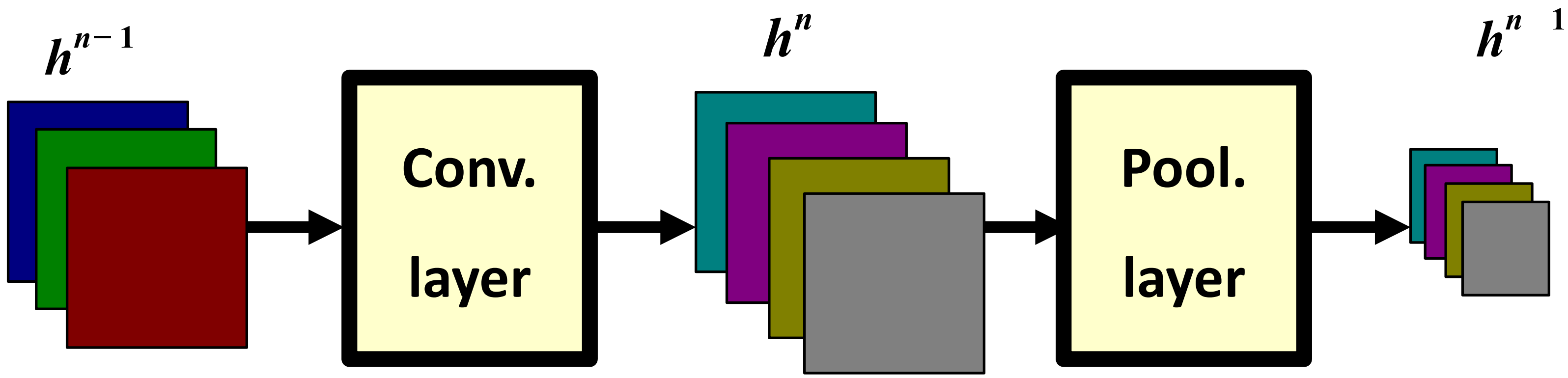
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since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

Common settings:

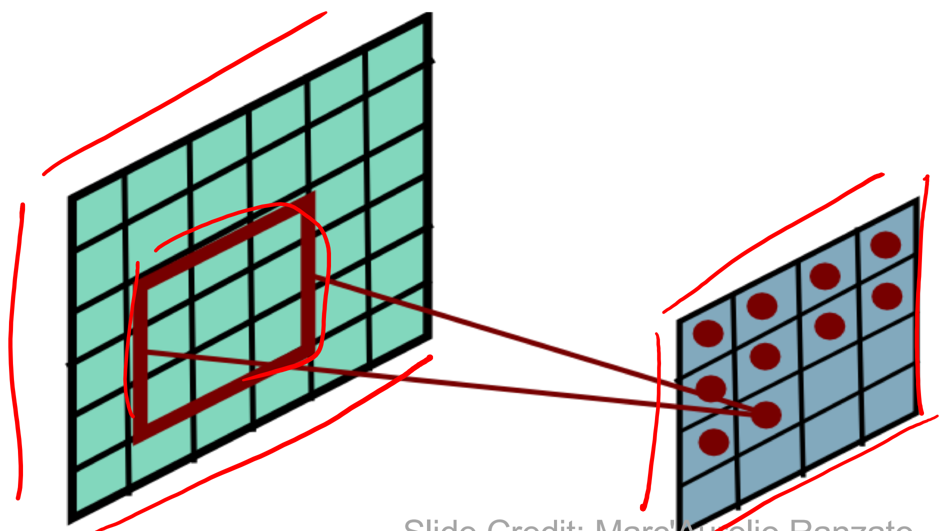
F = 2, S = 2
F = 3, S = 2

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

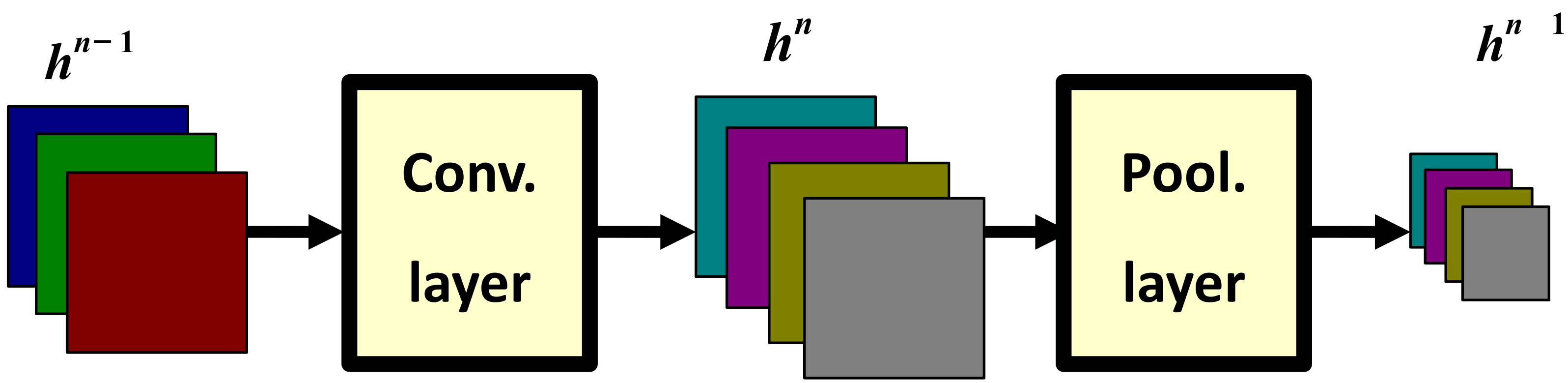
Pooling Layer: Receptive Field Size



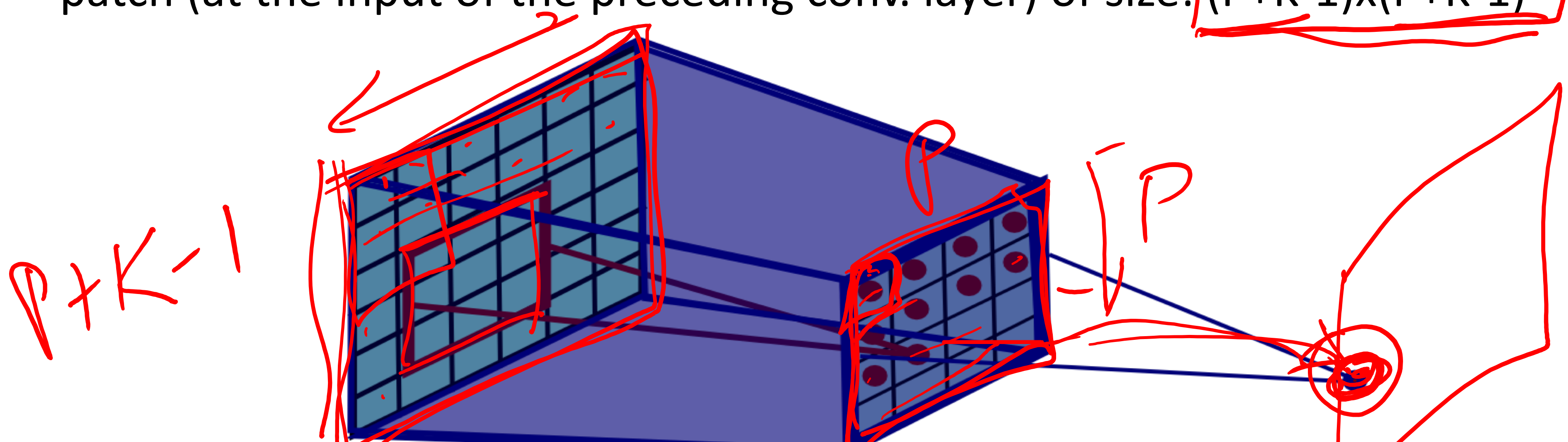
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Pooling Layer: Receptive Field Size



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

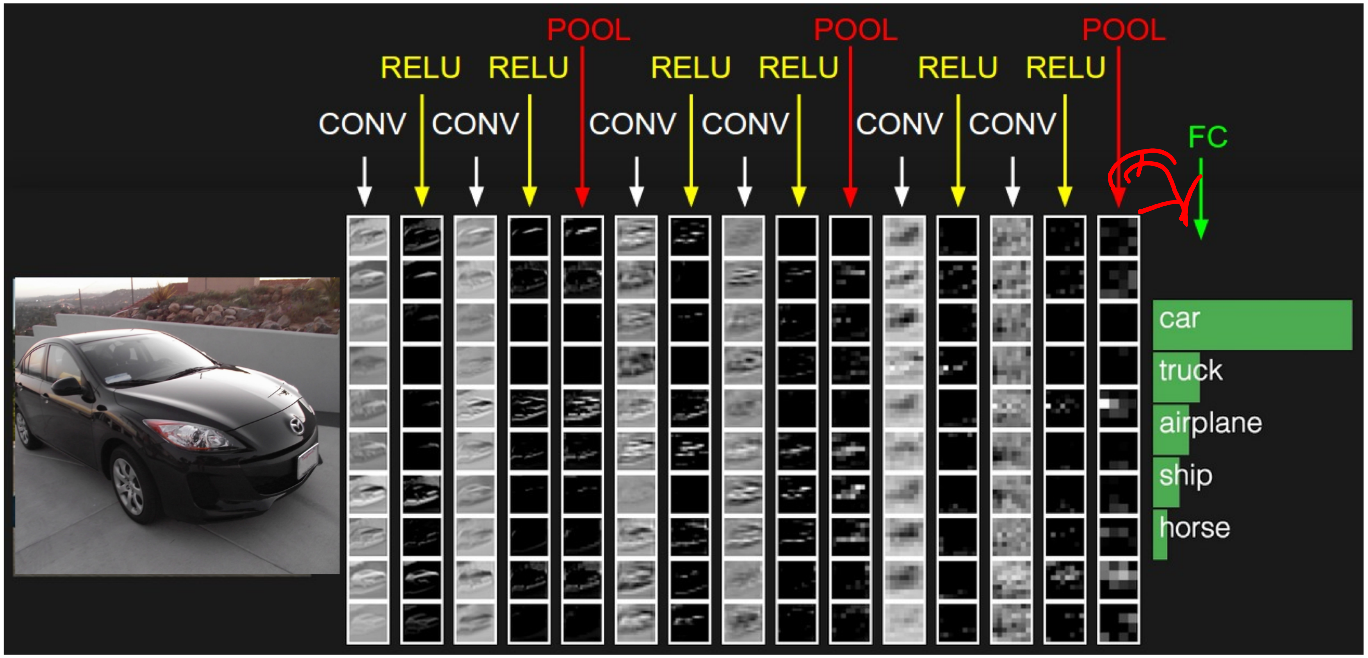


Plan for Today

- Convolutional Neural Networks
  - 1x1 convolutions
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers

Fully Connected Layer (FC layer)

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



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

Classical View

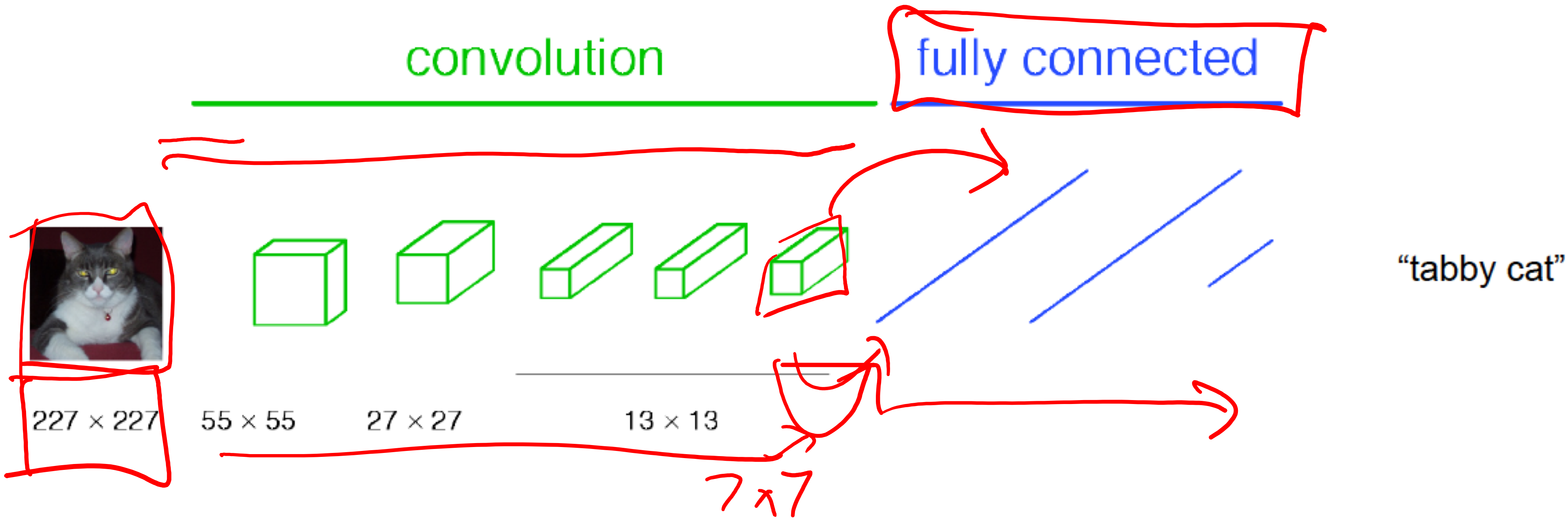
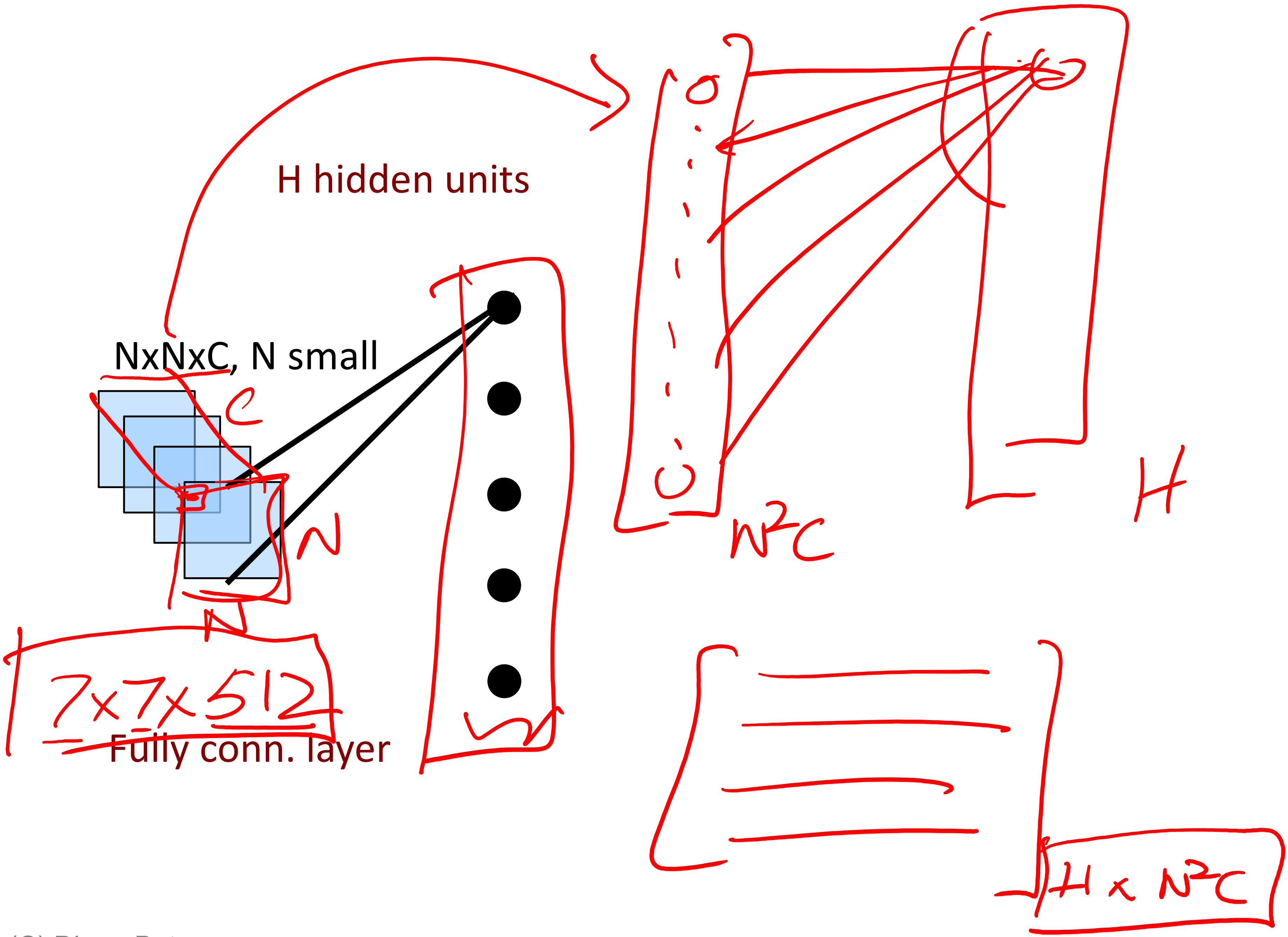
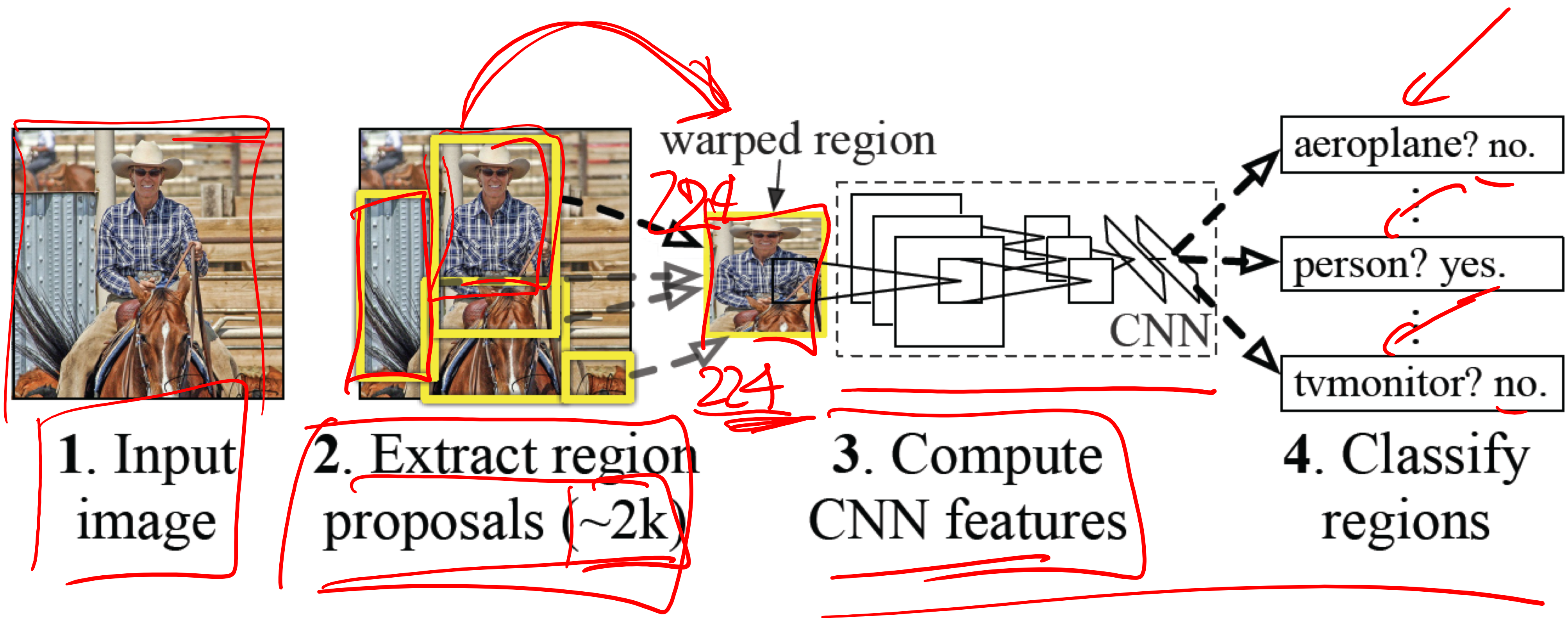


Figure Credit: [Long, Shelhamer, Darrell CVPR15]


H hidden units
NxNxC, N small
C
N
N
7x7x512
Fully conn. layer
N²C
H
H x N²C

Classical View = Inefficient



Classical View

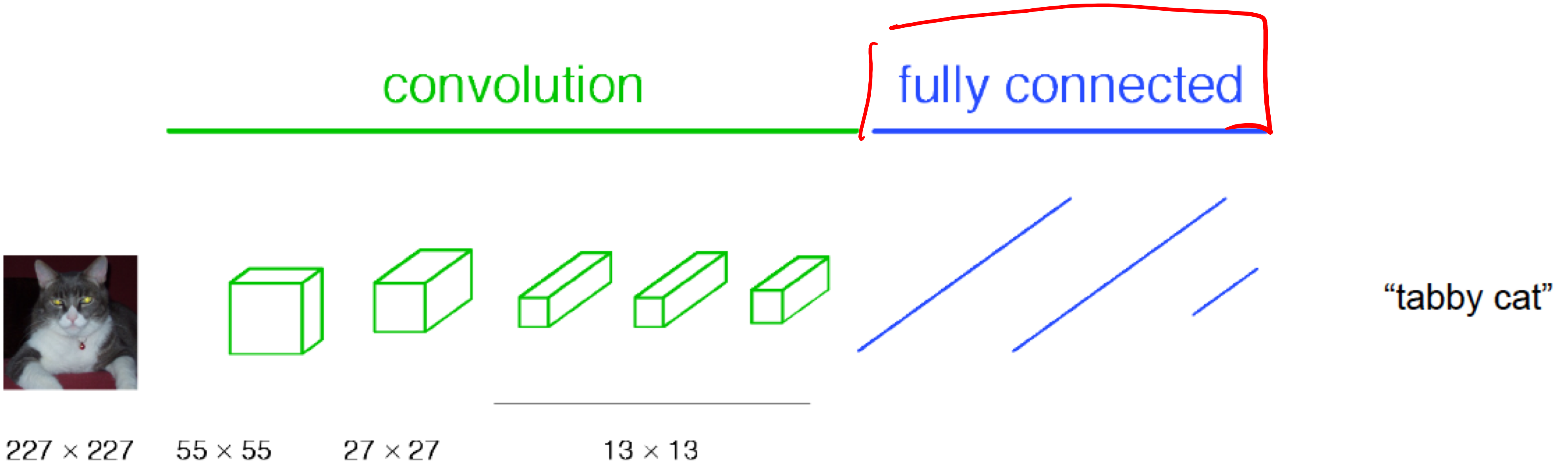


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

- Just squint a little!

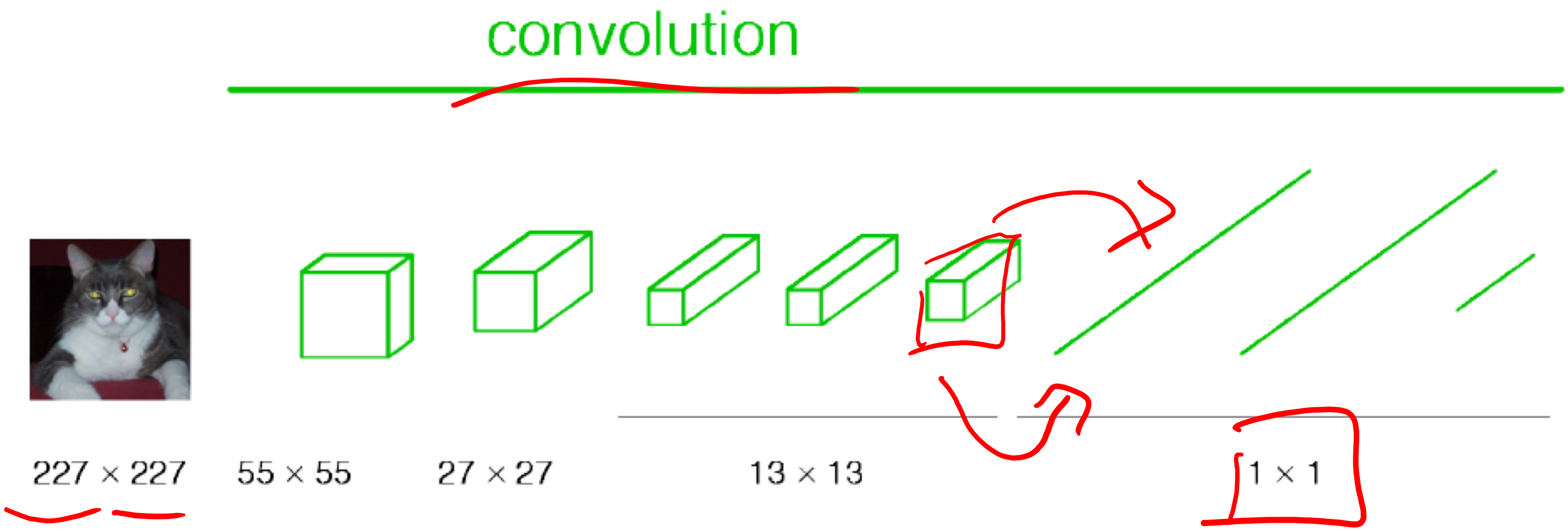


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

"Fully Convolutional" Networks

- Can run on an image of any size!

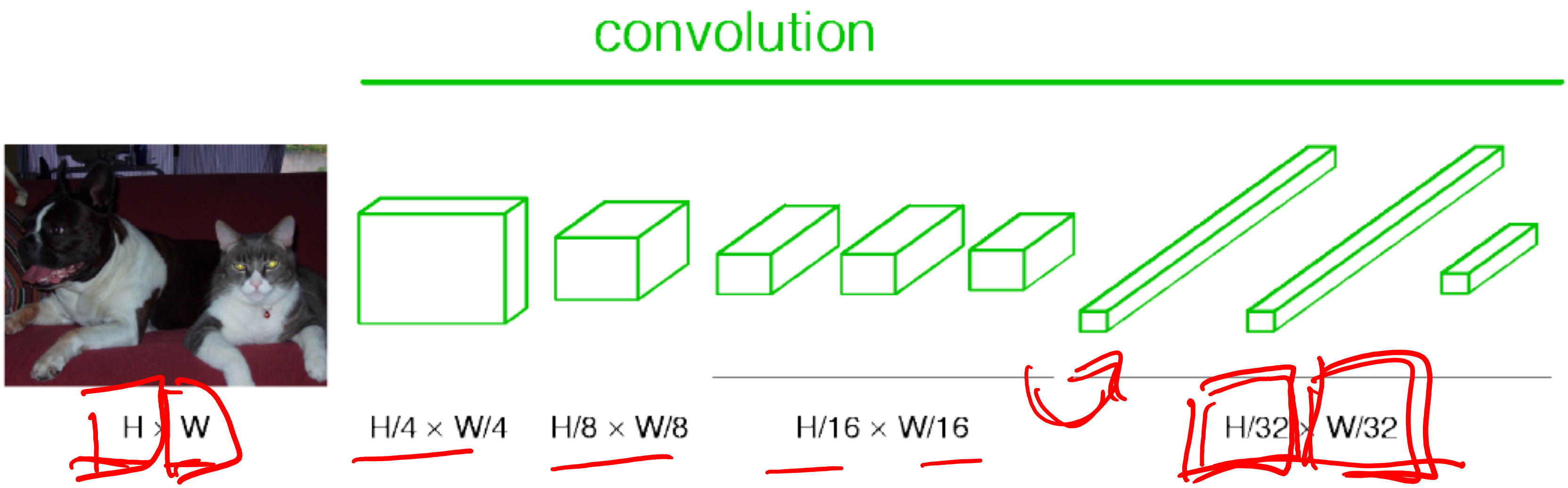
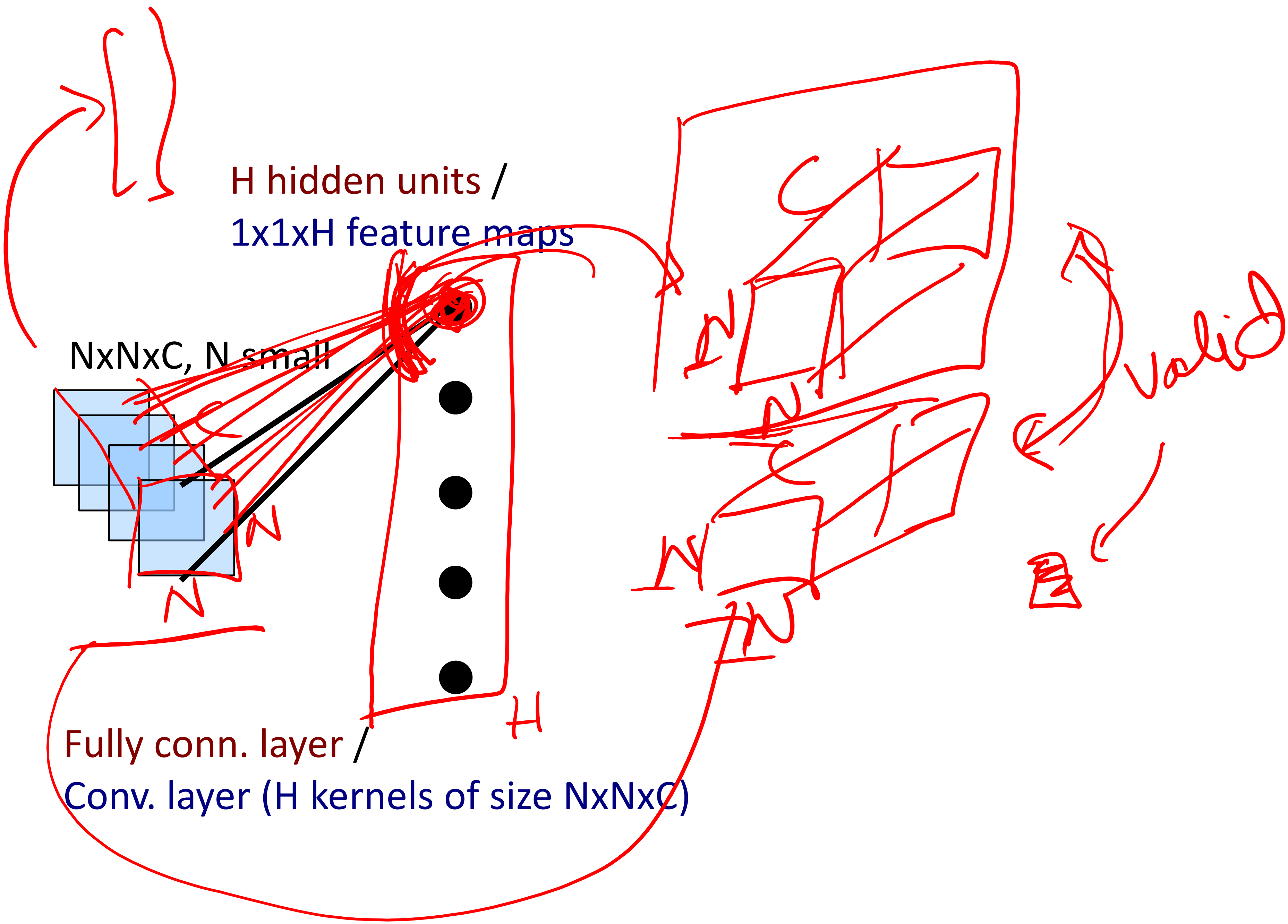
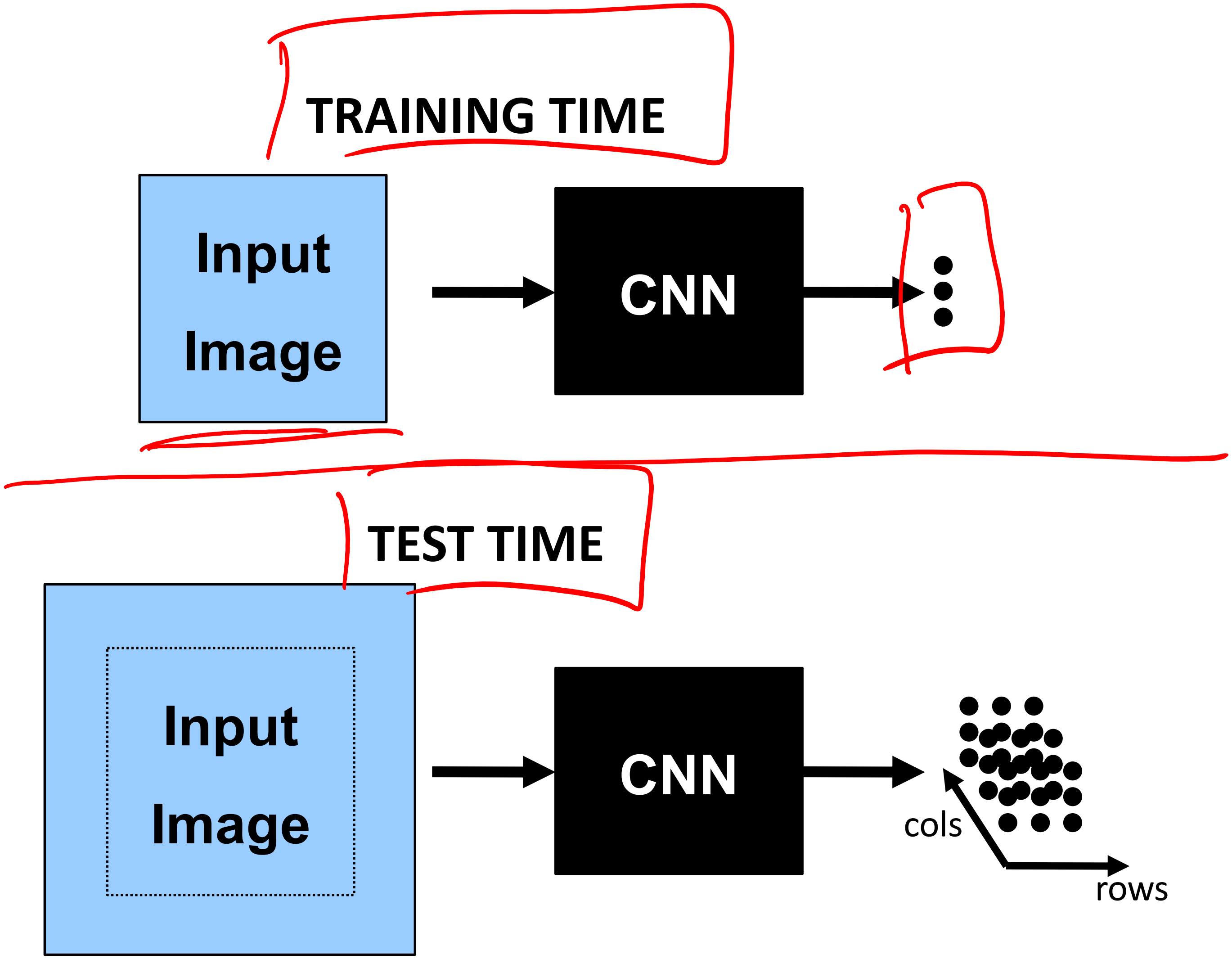


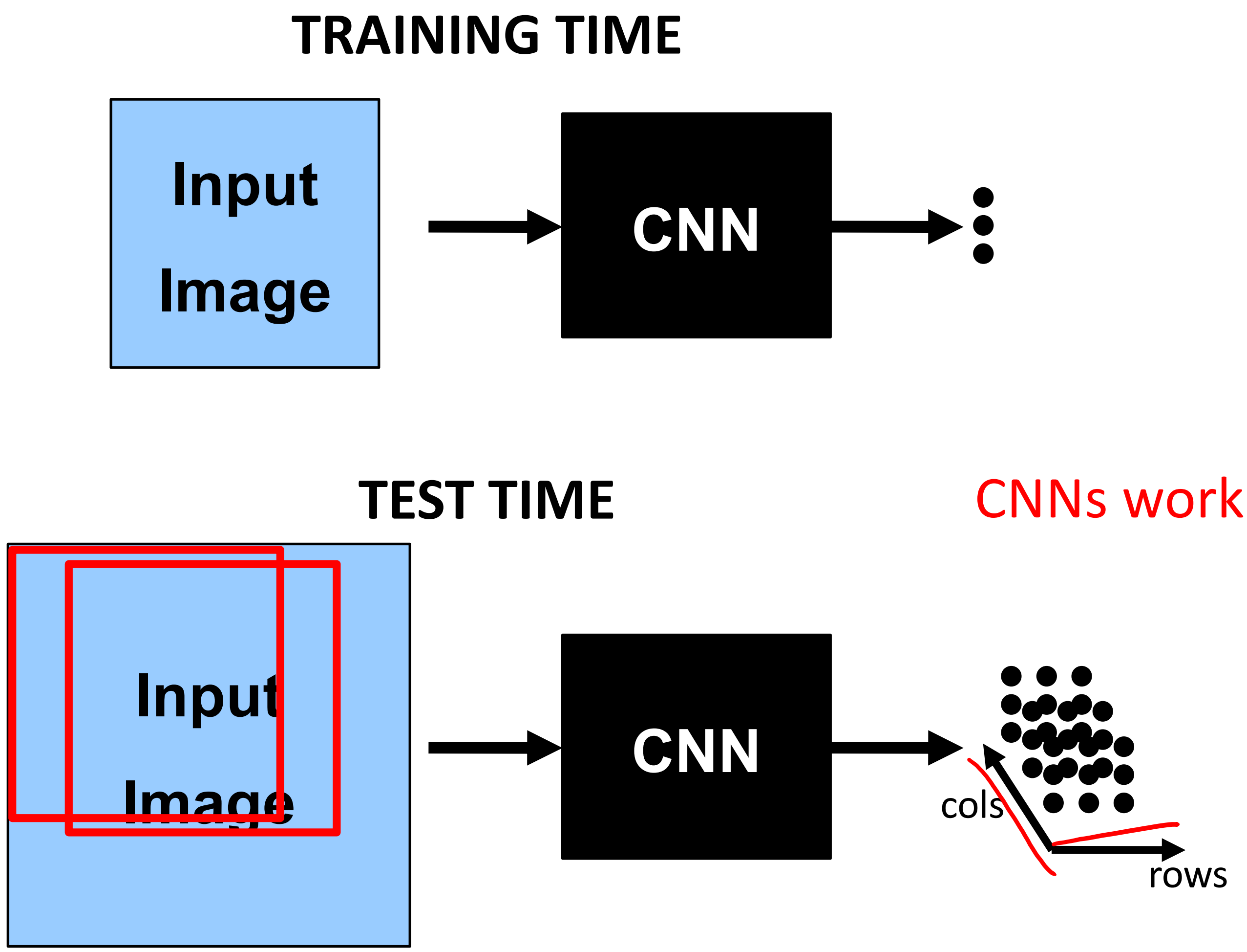
Figure Credit: [Long, Shelhamer, Darrell CVPR15]


H hidden units /
1x1xH feature maps
NxNxC, N small
Fully conn. layer /
Conv. layer (H kernels of size NxNxC)
H
N
C
Valid

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).



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).

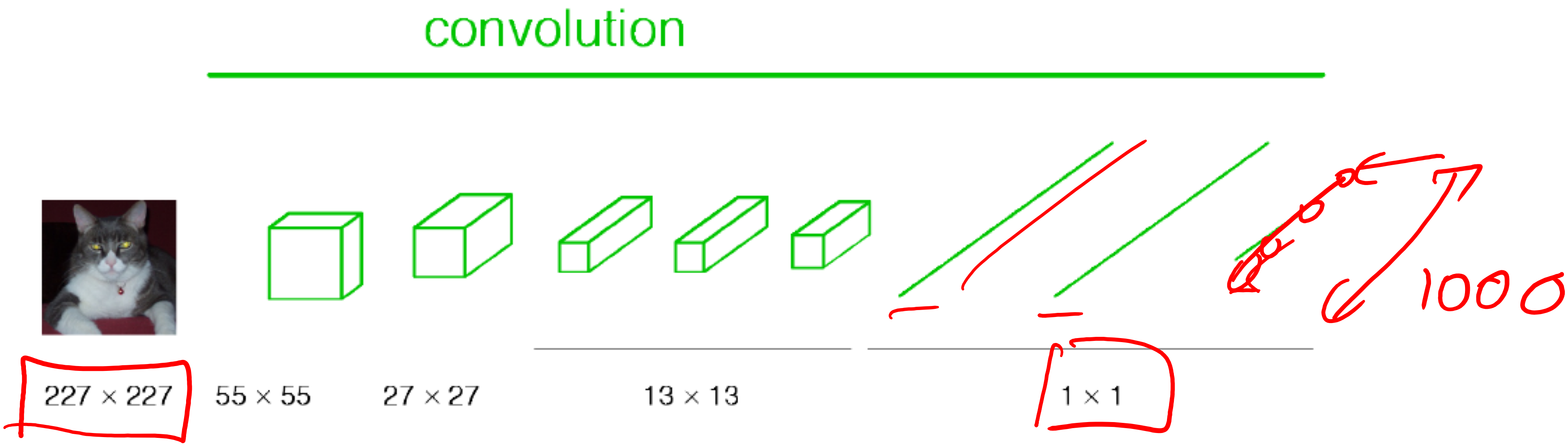


CNNs work on any image size!

Unrolling is order of magnitudes more eficient than sliding windows!

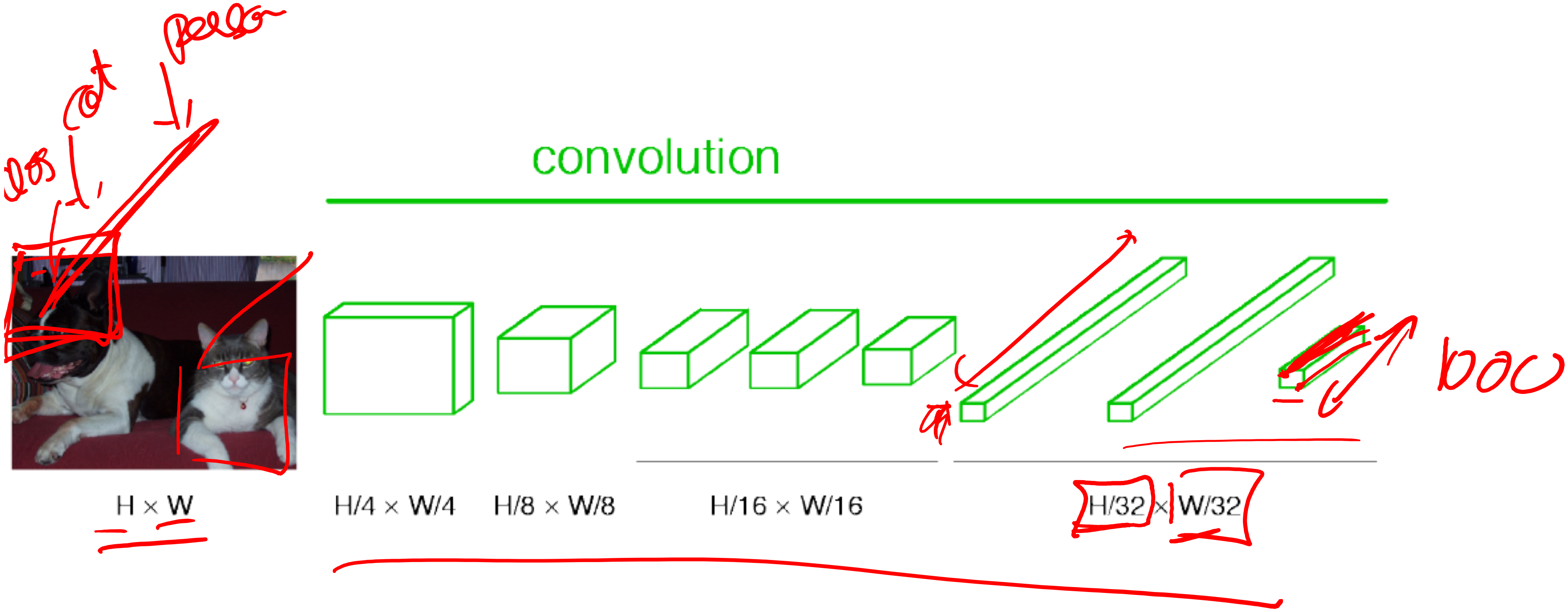
Training time

- Fixed-size images



Testing time

- Can run on an image of any size!



Benefit of this thinking

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

"Fully Convolutional" Networks

- Up-sample to get segmentation maps

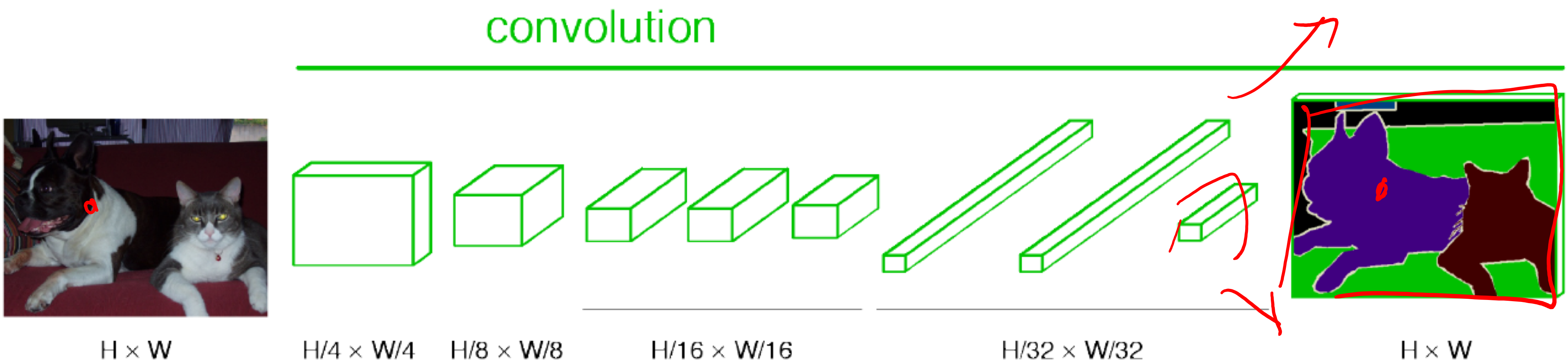
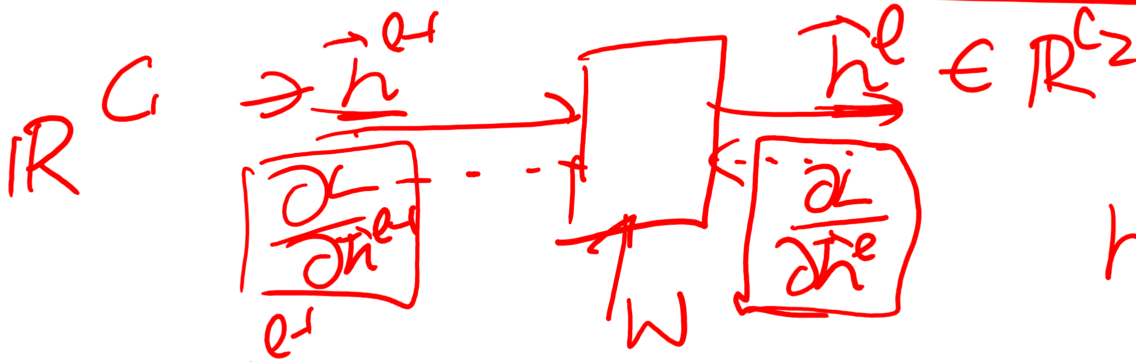


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

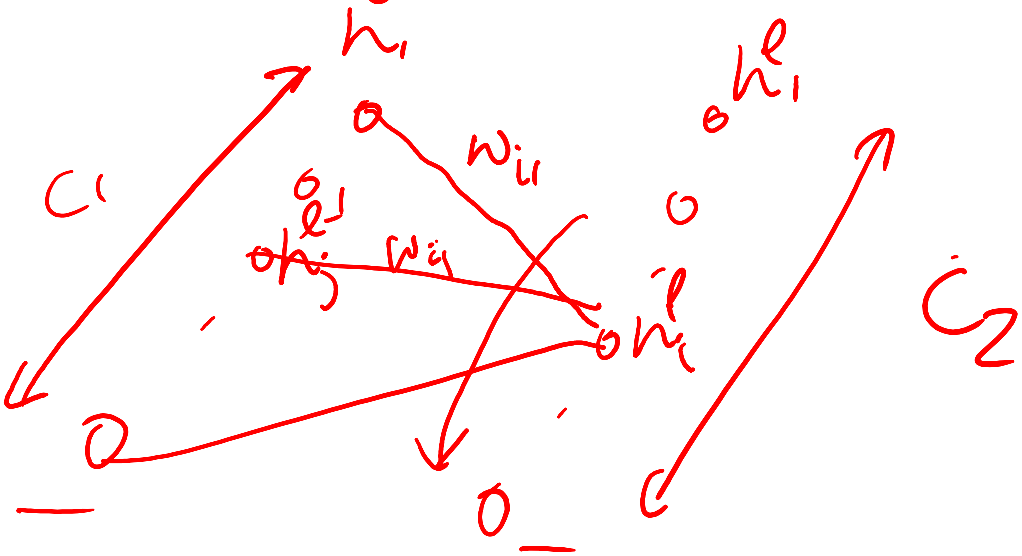
Plan for Today

- Convolutional Neural Networks
  - 1x1 convolutions
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers

Jacobians of FC-Layer



$h_i^{\ell} = \vec{w}_i^{T} \vec{h}^{\ell-1}$



$\underbrace{\begin{bmatrix} \text{---} \\ \text{---}\ \vec{w}_i^{T}\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}}_{W} \begin{bmatrix} \vec{h}^{\ell-1} \end{bmatrix}$

$\vec{h}^{\ell} = \boxed{W}\, \vec{h}^{\ell-1}$ F-Pass

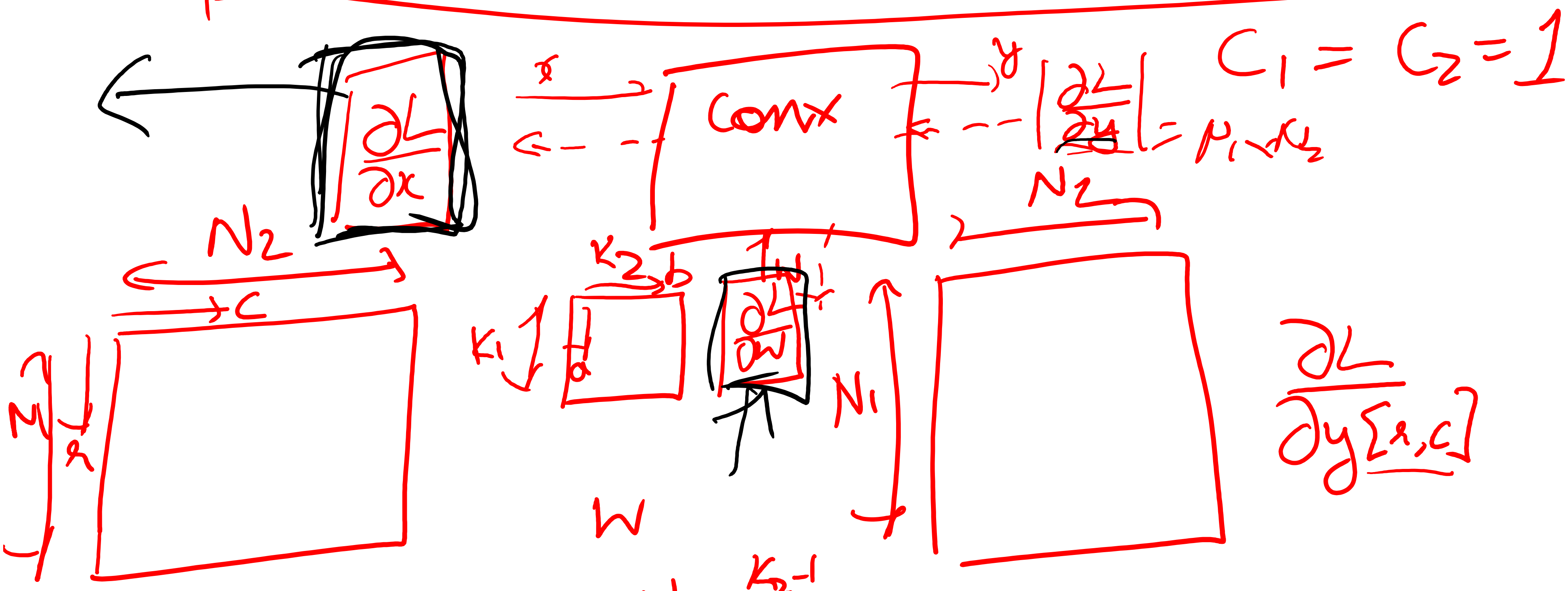
$\vec{w}_i$

$$\boxed{\frac{\partial L}{\partial \vec{h}^{\ell-1}}} = \boxed{\frac{\partial L}{\partial \vec{h}^{\ell}} \cdot \underbrace{\frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}}}_{W}}$$

B-Pass

Backprop in Convolutional Layers

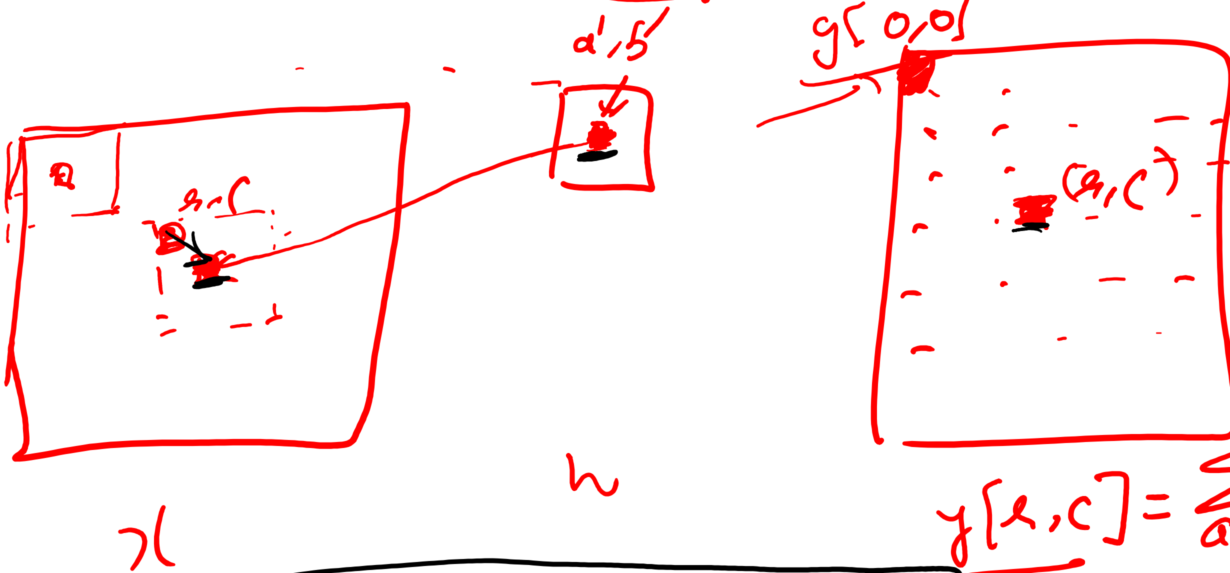
- Notes
  - https://www.cc.gatech.edu/classes/AY2018/cs7643_fall/slides/L6_cnns_backprop_notes.pdf



$$y[r,c] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} x[r+a,\, c+b]\, w[a,b]$$

Backprop in Convolutional Layers

$$\frac{\partial L}{\partial w[a',b']} = \sum_{\text{all pixels } p \text{ affected}} \underbrace{\frac{\partial L}{\partial y[p]}}_{\text{input}} \underbrace{\frac{\partial y[p]}{\partial w[a',b']}}_{\text{Jacobian}}$$



$$y[r,c] = \sum_a \sum_b x[r+a, c+b]\, w[a,b]$$

$$= x[r+0, c+0]\, w[0,0] + x[r+0, c+1]\, w[0,1] + \dots + x[r+a', c+b']\, w[a',b']$$

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

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