CS 7643: Deep Learning

Topics:

- Stride, padding
- Pooling layers
- Fully-connected layers as convolutions
- Backprop in conv layers

Dhruv Batra

Georgia Tech

Invited Talks

- Sumit Chopra on CNNs for Pixel Labeling
  - Head of AI Research @ Imagen Technologies
    - Previously Facebook AI Research
  - Tue 09/26, in class

Sumit Chopra

sumit [at] imagentechnologies [dot] com

Background

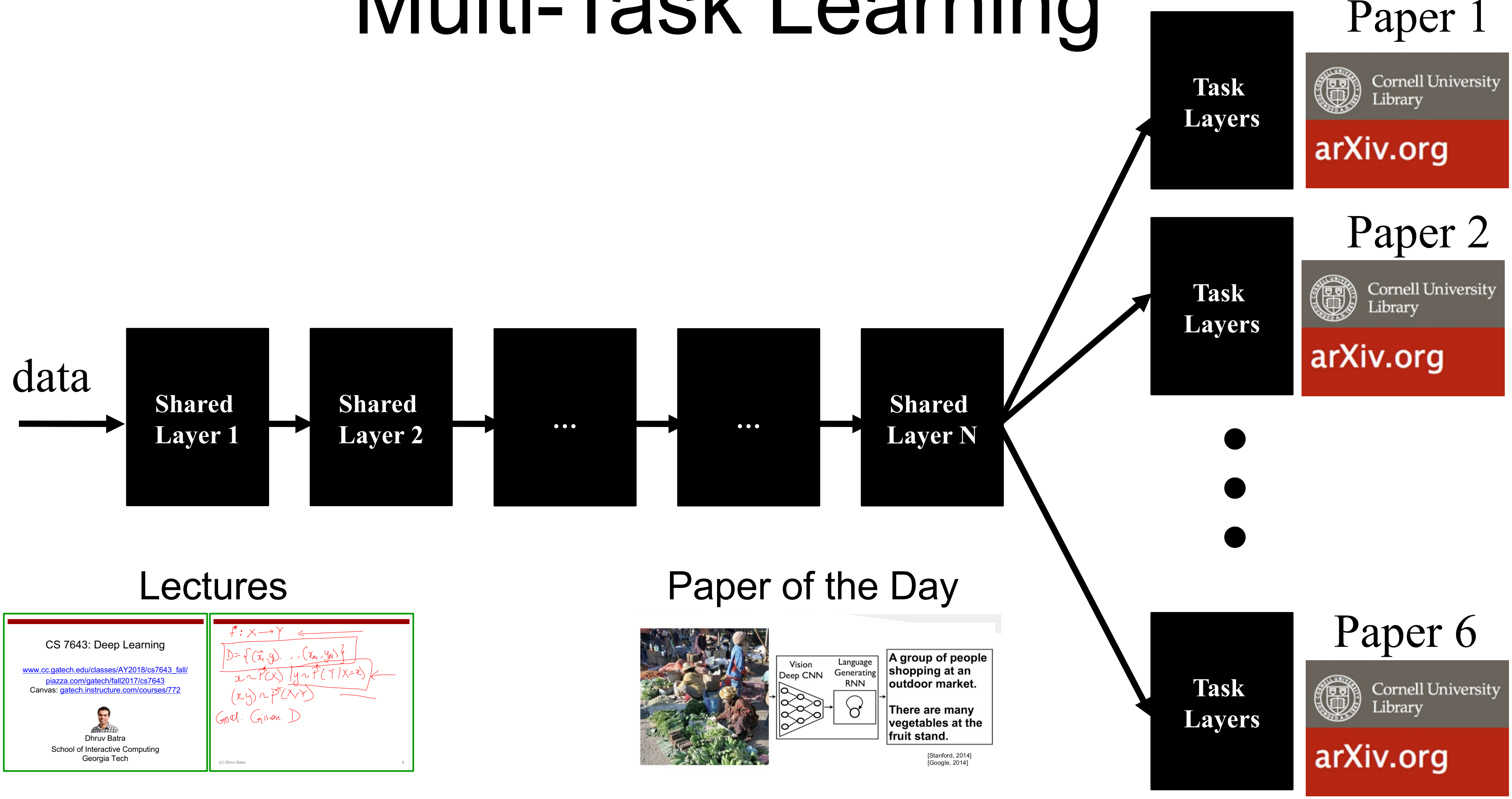
I am the head of A.I. Research at Imagen Technologies: a well funded stealth startup working towards transforming healthcare using artificial intelligence. I am interested in advancing AI research with a particular focus towards deep learning and healthcare.

Before Imagen, I was a research scientist at Facebook AI Research (FAIR), where I worked on understanding natural language. I graduated with a Ph.D., in computer science from New York University under the supervision of Prof. Yann LeCun. My thesis proposed a first of its kind neural network model for relational regression, and was a conceptual foundation for a startup for modeling residential real estate prices. Following my Ph.D., I joined AT&T Labs – Research as a scientist in the machine learning department. There I focused on building novel deep learning models for speech recognition, natural language processing, computer vision, and other areas of machine learning, such as, recommender systems, computational advertisement, and ranking.

Administrativia

- HW1 due soon
  - 09/22
- HW2 + PS2 both coming out on 09/22
- 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 09/26
- First student presentation due: Thr 09/28

Paper Reading Intuition: Multi-Task Learning



Paper Reviews

- Length
  - 200-400 words.
- Due: Midnight before class on Piazza

- Organization
  - Summary:
    - What is this paper about? What is the main contribution? Describe the main approach & results. Just facts, no opinions yet.
  - List of positive points / Strengths:
    - Is there a new theoretical insight? Or a significant empirical advance? Did they solve a standing open problem? Or is a good formulation for a new problem? Or a faster/better solution for an existing problem? Any good practical outcome (code, algorithm, etc)? Are the experiments well executed? Useful for the community in general?
  - List of negative points / Weaknesses:
    - What would you do differently? Any missing baselines? missing datasets? any odd design choices in the algorithm not explained well? quality of writing? Is there sufficient novelty in what they propose? Has it already been done? Minor variation of previous work? Why should anyone care? Is the problem interesting and significant?
  - Reflections
    - How does this relate to other papers we have read? What are the next research directions in this line of work?

Presentations

- Frequency
  - Once in the semester: 5 min presentation.
- Expectations
  - Present details of 1 paper
    - Describe formulation, experiment, approaches, datasets
    - Encouraged to present a broad picture
    - Show results, videos, gifs, etc.
  - Please clearly cite the source of each slide that is not your own.
  - Meet with TA 1 week before class to dry run presentation
    - Worth 40% of presentation grade

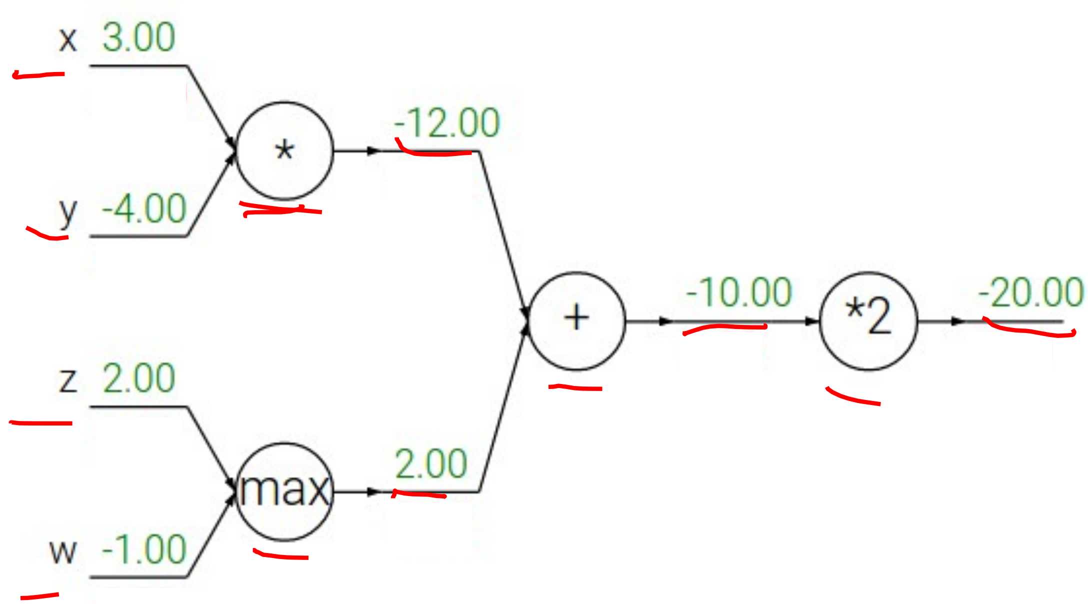
Administrativia

- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1AaXY0JE4lAbHvoDaWlc9zsmfKMyuGS39JAn9dpeXhhQ/edit#gid=0
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names + GT IDs

Recap of last time

Patterns in backward flow

$f(\ldots) = 2(xy + \max\{z, w\})$



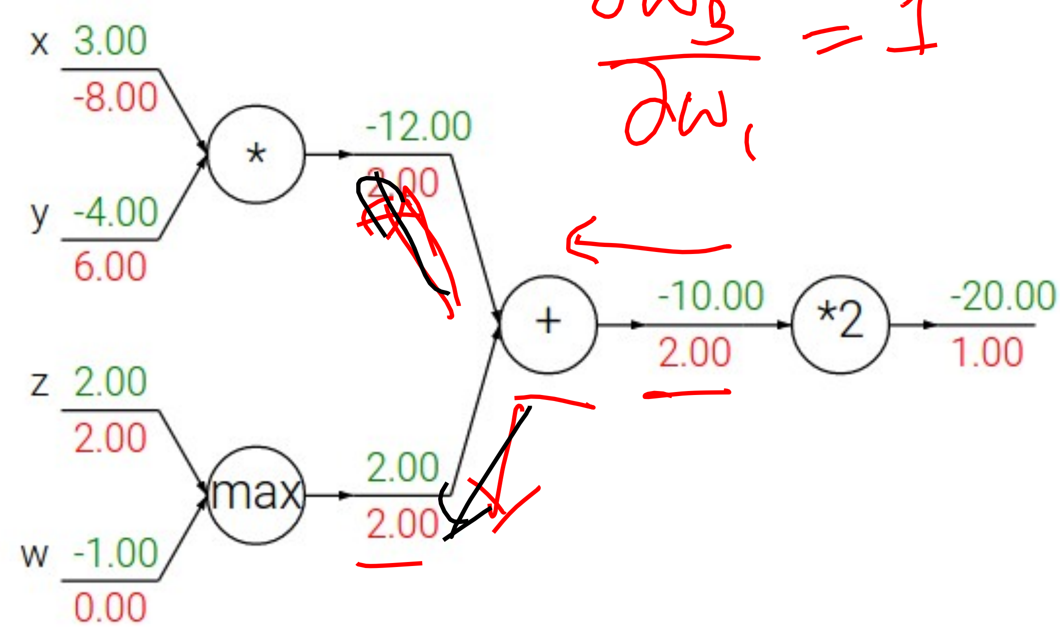
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

$$w_3 = w_1 + w_2$$

$$\frac{\partial w_3}{\partial w_1} = 1$$

add gate: gradient distributor

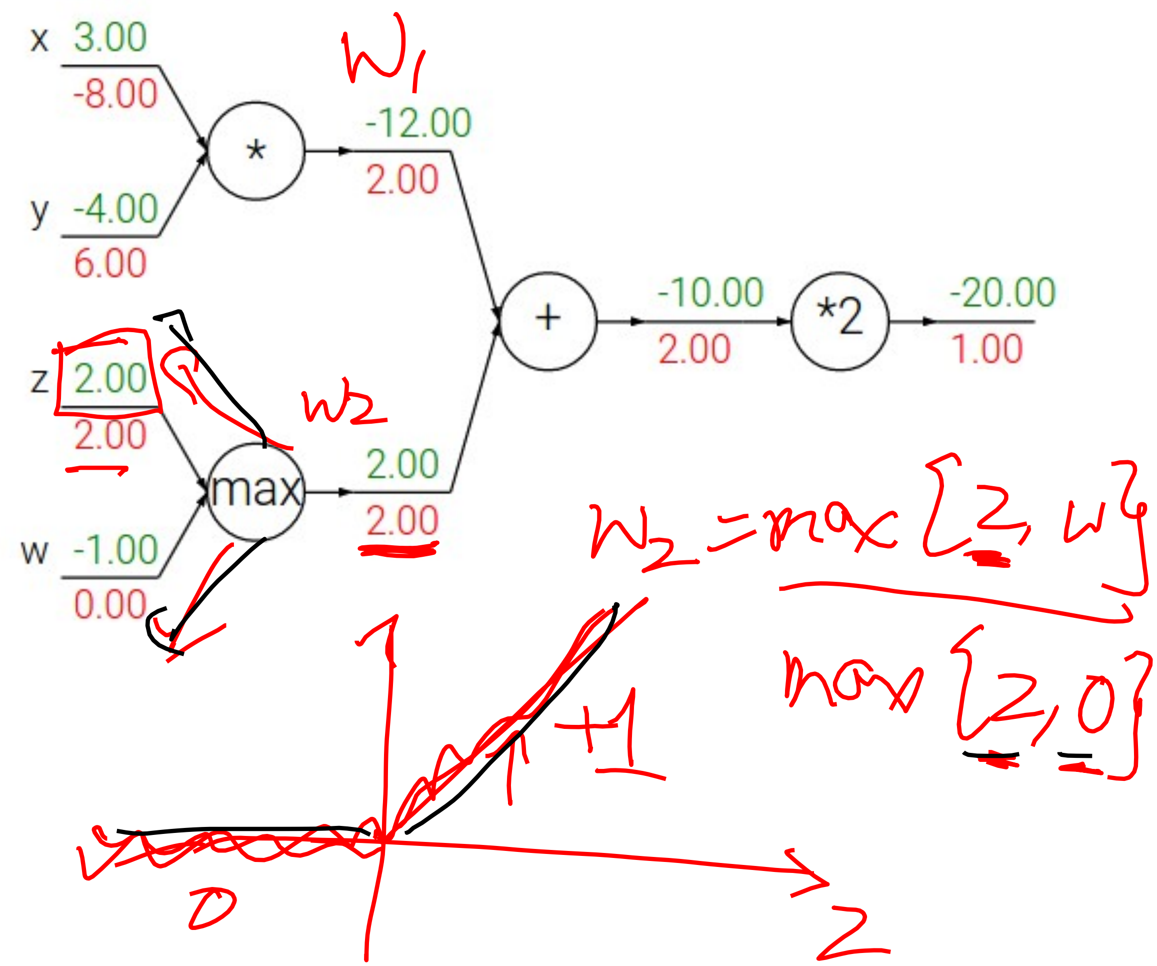


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Patterns in backward flow

add gate: gradient distributor

Q: What is a **max** gate?

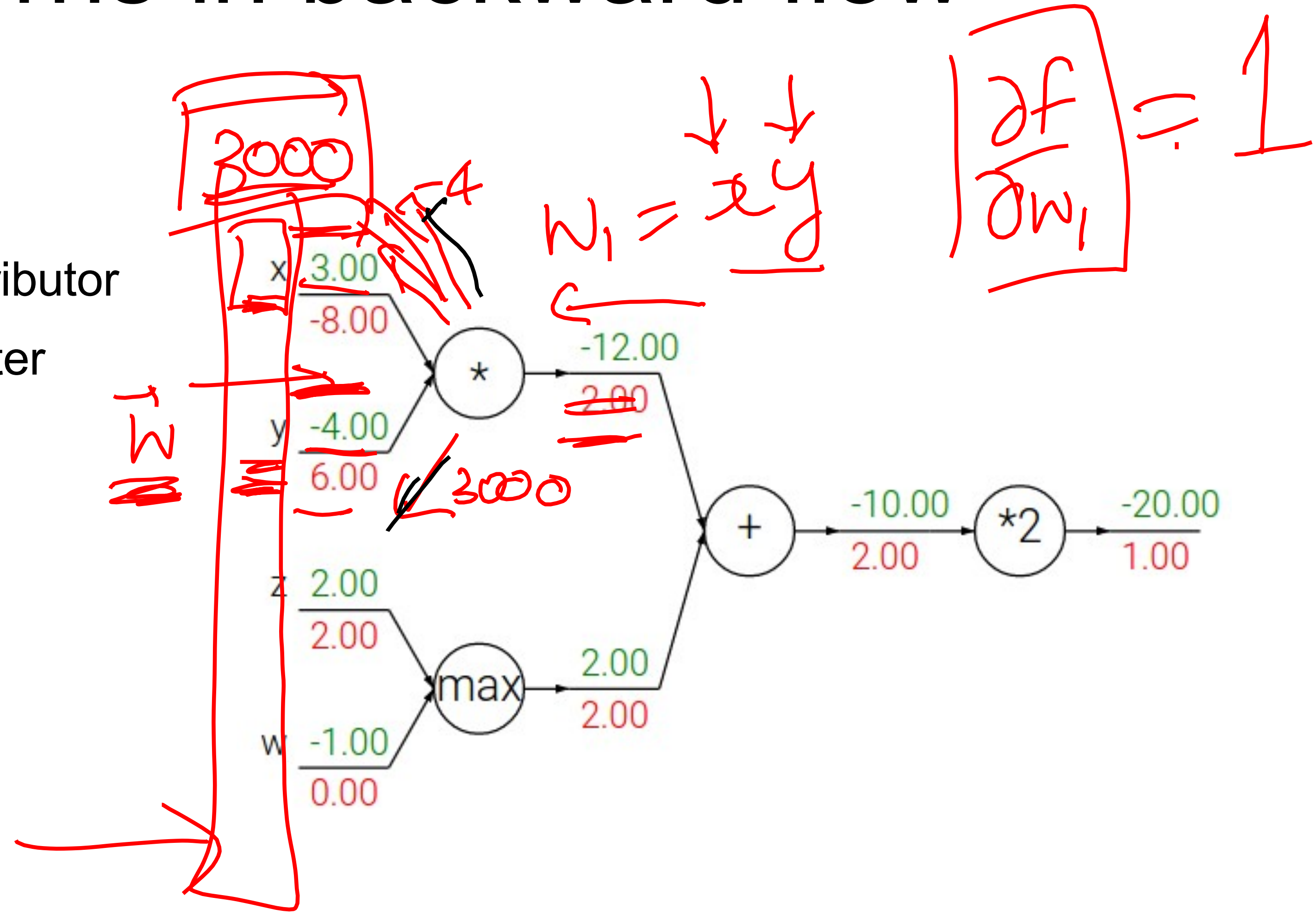


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

Q: What is a **mul** gate?

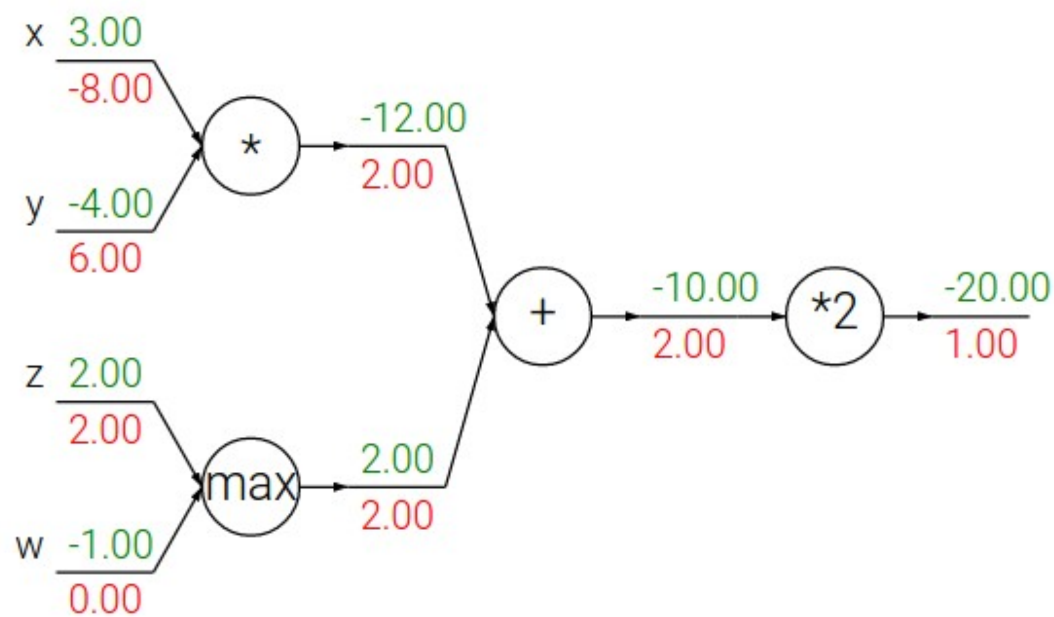


Patterns in backward flow

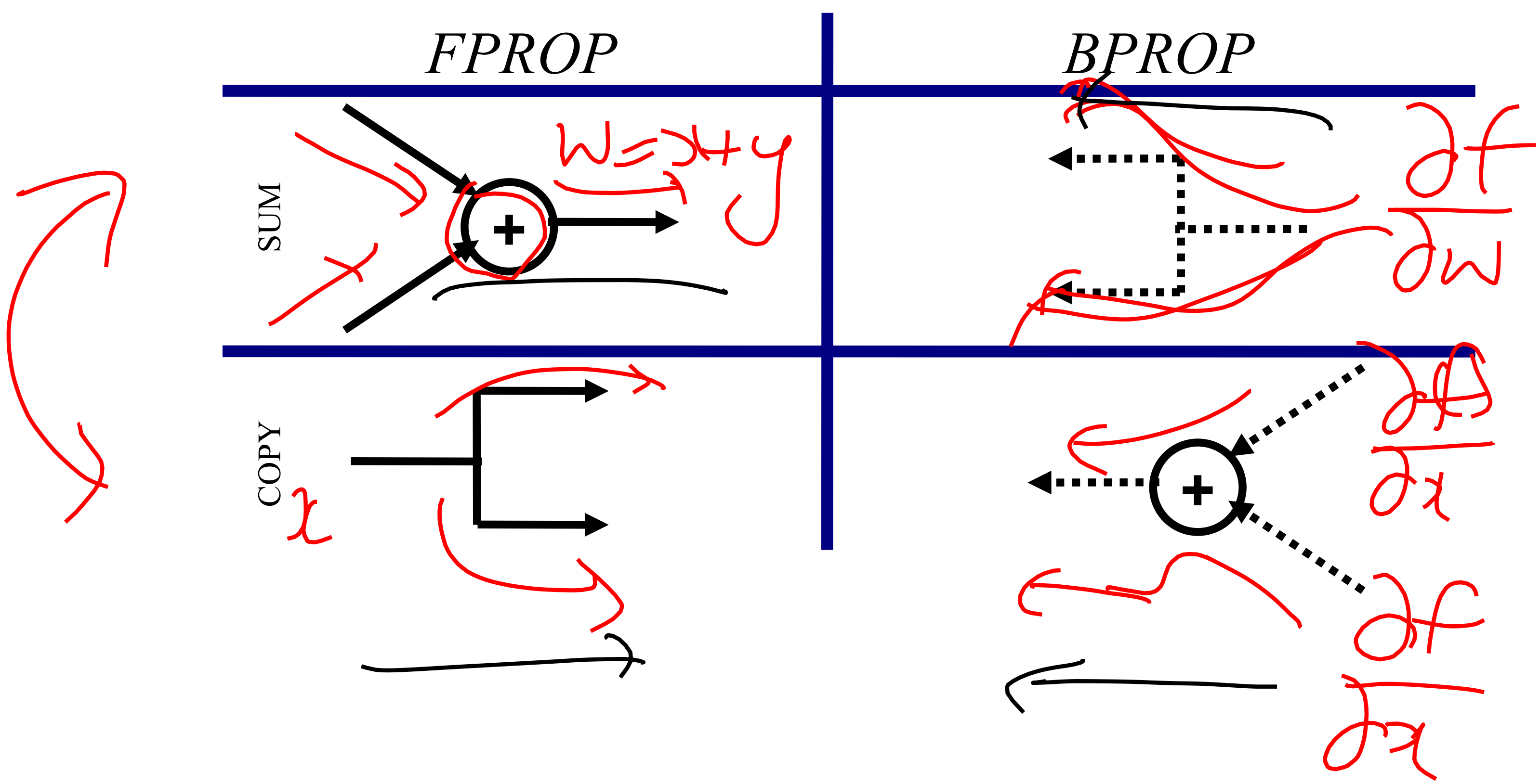
add gate: gradient distributor

max gate: gradient router

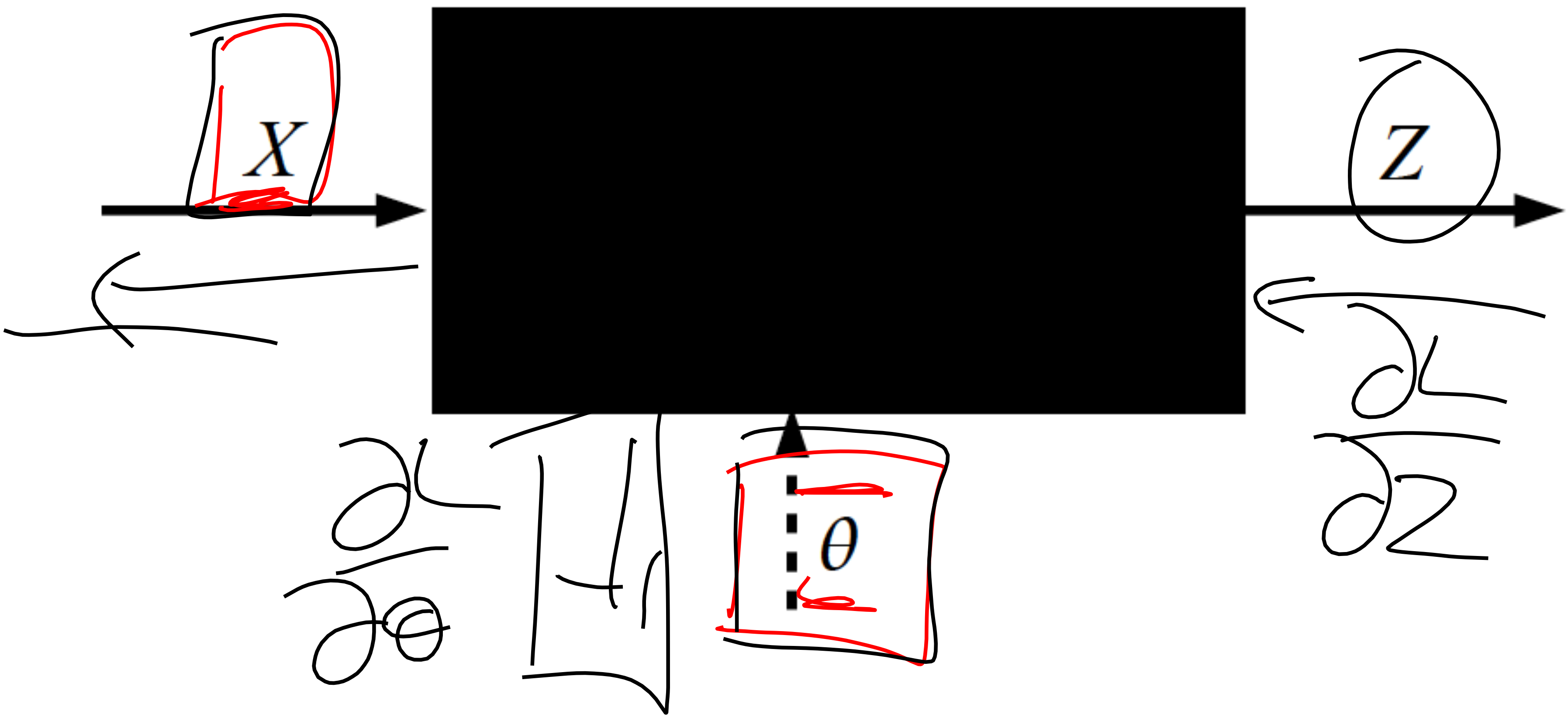
mul gate: gradient switcher

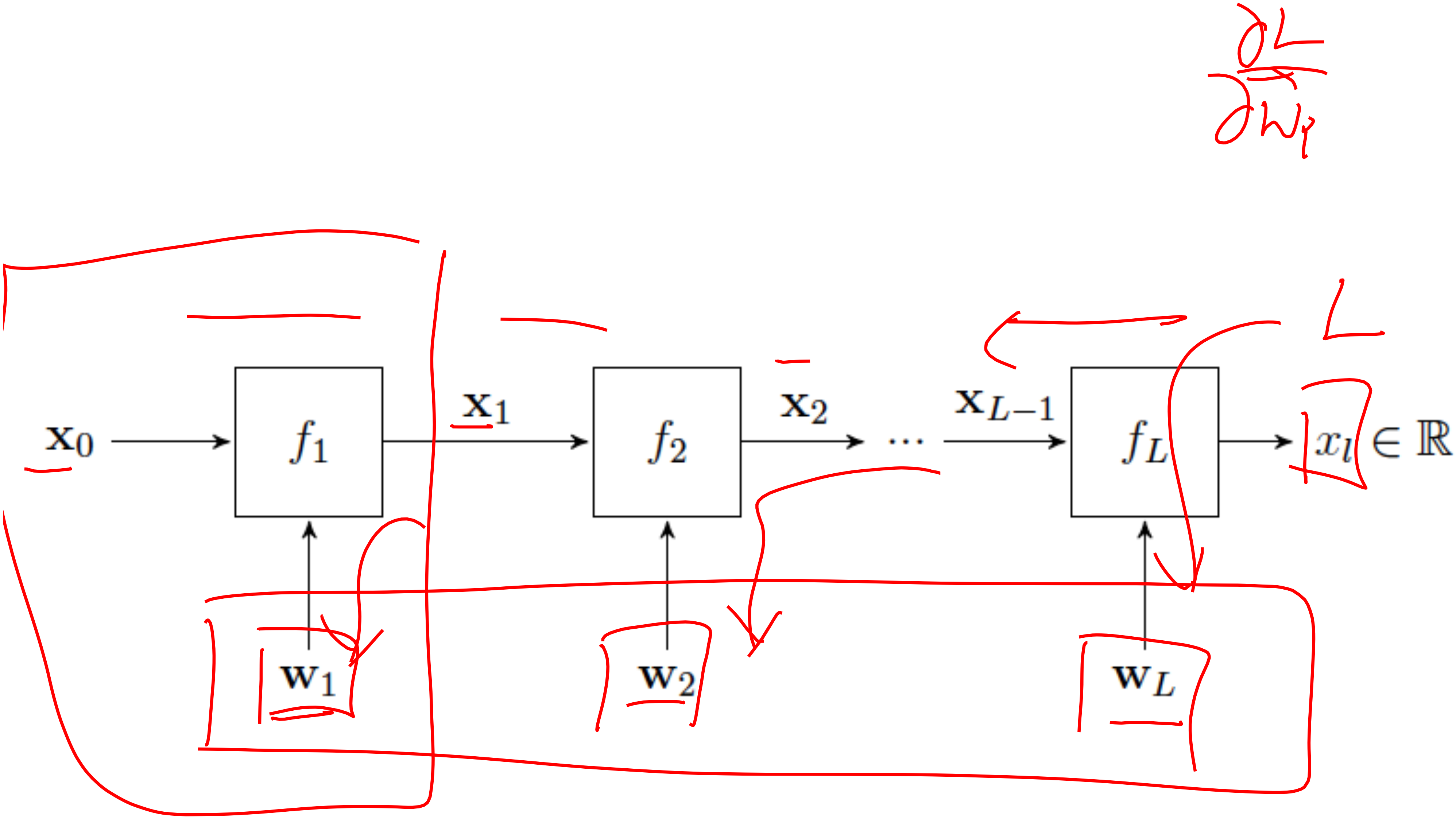


Duality in Fprop and Bprop



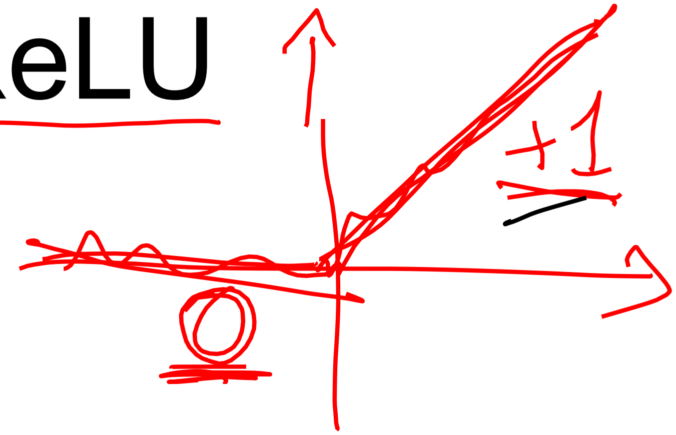
Key Computation in DL: Forward-Prop





$\frac{\partial L}{\partial w_l}$
L
$\mathbf{x}_0$
f_1
$\mathbf{x}_1$
f_2
$\mathbf{x}_2$
$\cdots$
$\mathbf{x}_{L-1}$
f_L
$x_l \in \mathbb{R}$
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_L$

Jacobian of ReLU



4096-d
input vector

f(x) = max(0,x)
(elementwise)

4096-d
output vector

Q: what is the size of the Jacobian matrix?

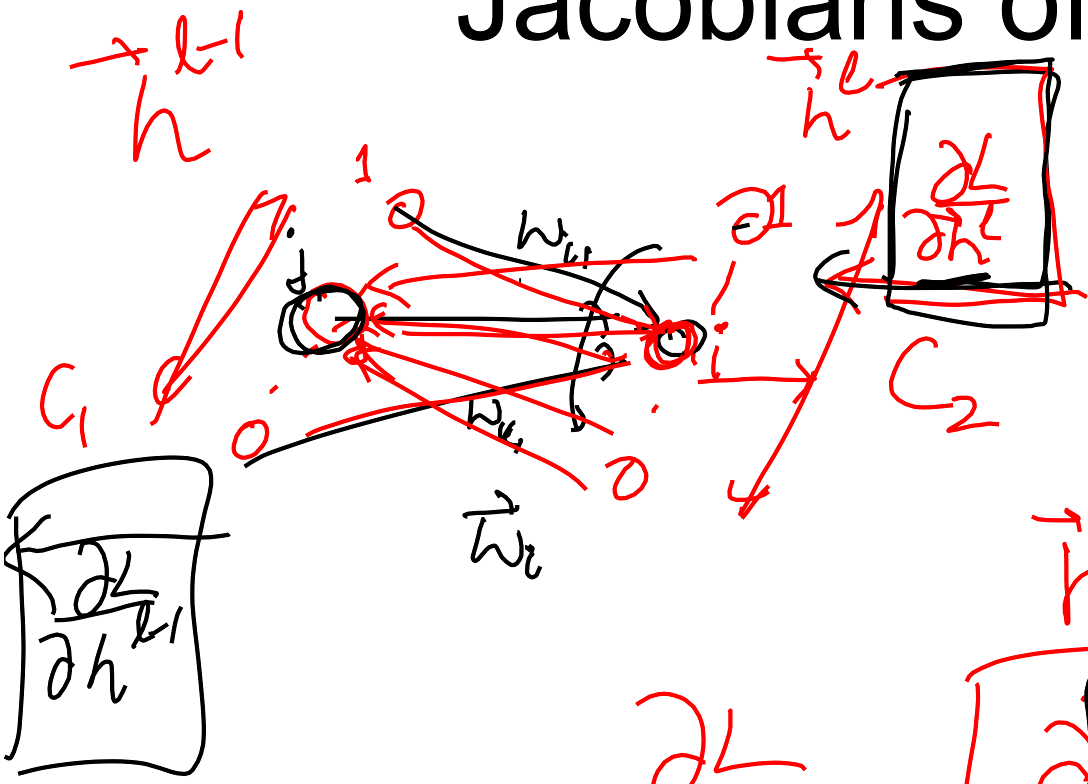
$\frac{\partial L}{\partial \vec{h}^{l+1}}$

4096

4096 x 4096

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

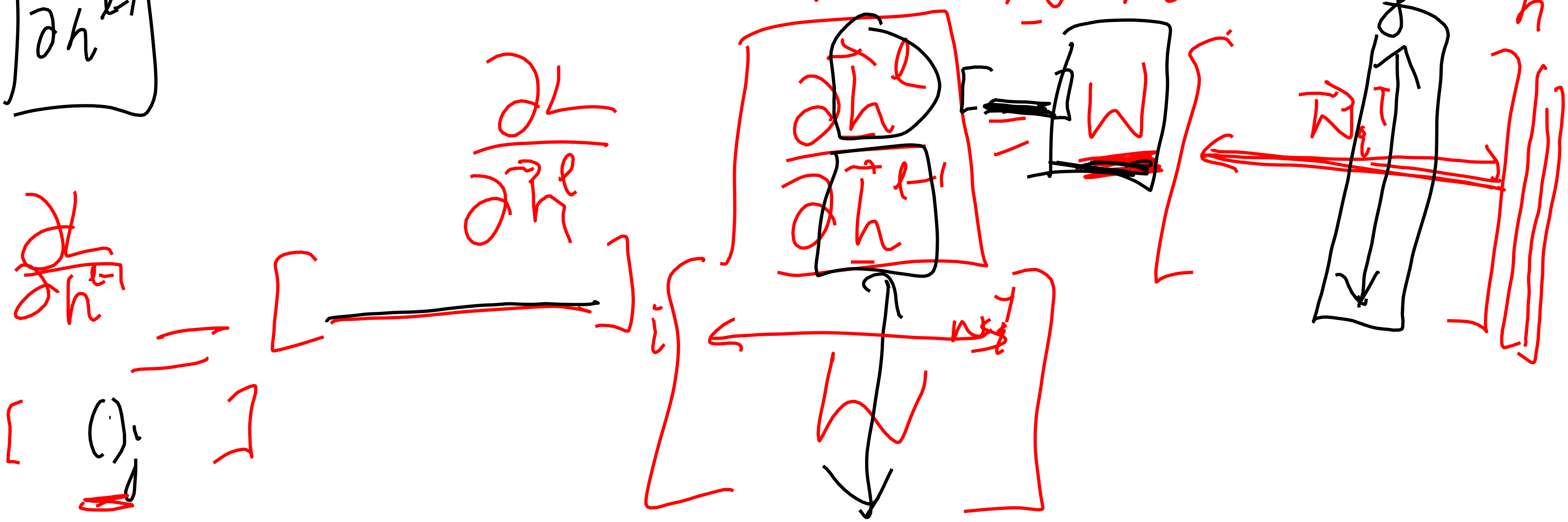
Jacobians of FC-Layer



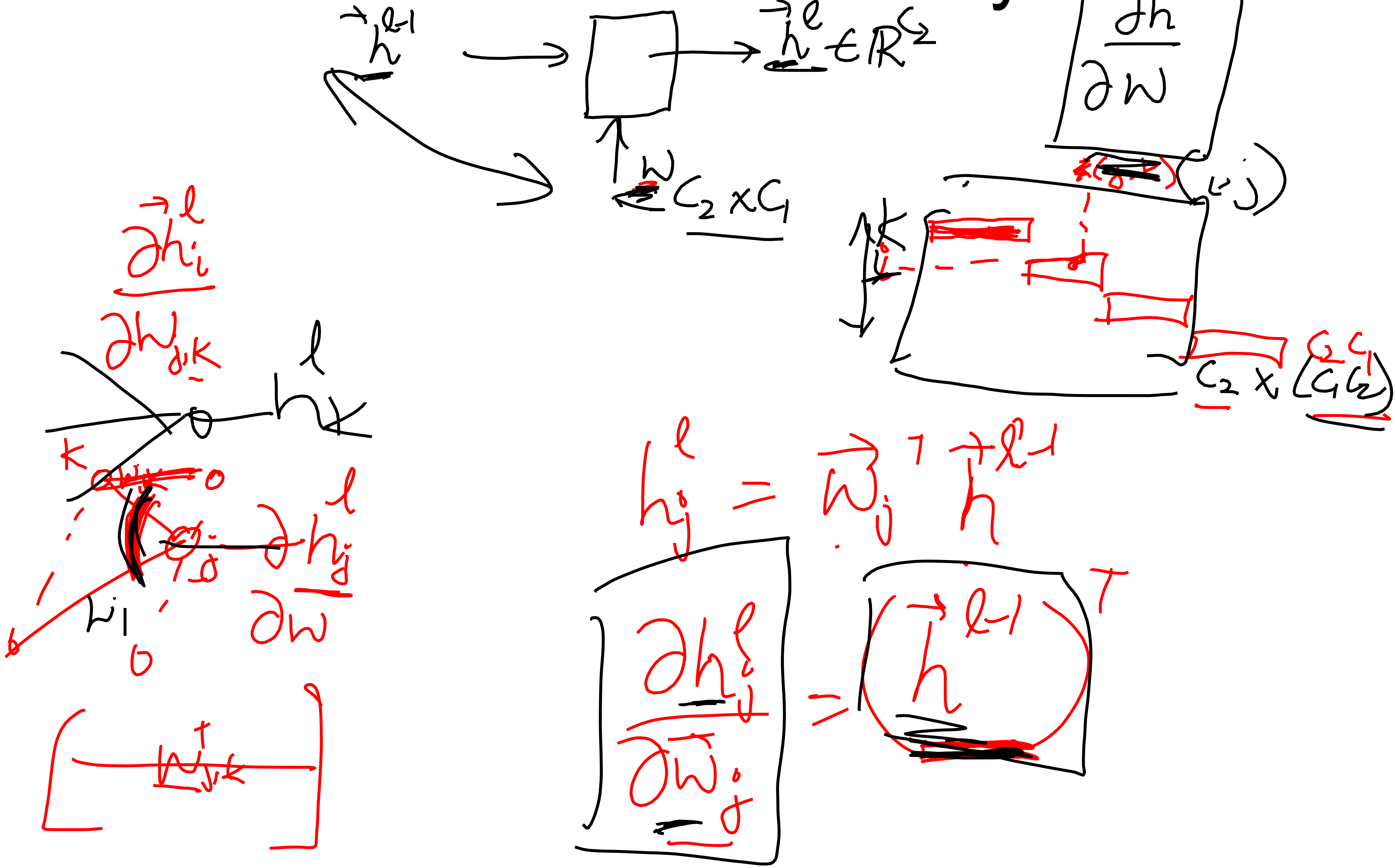
$$\vec{h}^{l}_i = (\vec{w}_i)^T \vec{h}^{l-1} + b_i$$

$$\vec{h}^{l} = W \vec{h}^{l-1}$$

$$\frac{\partial L}{\partial \vec{h}^{l-1}} = \frac{\partial L}{\partial \vec{h}^{l}} \frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}$$



Jacobians of FC-Layer



Convolutional Neural Networks

(without the brain stuff)

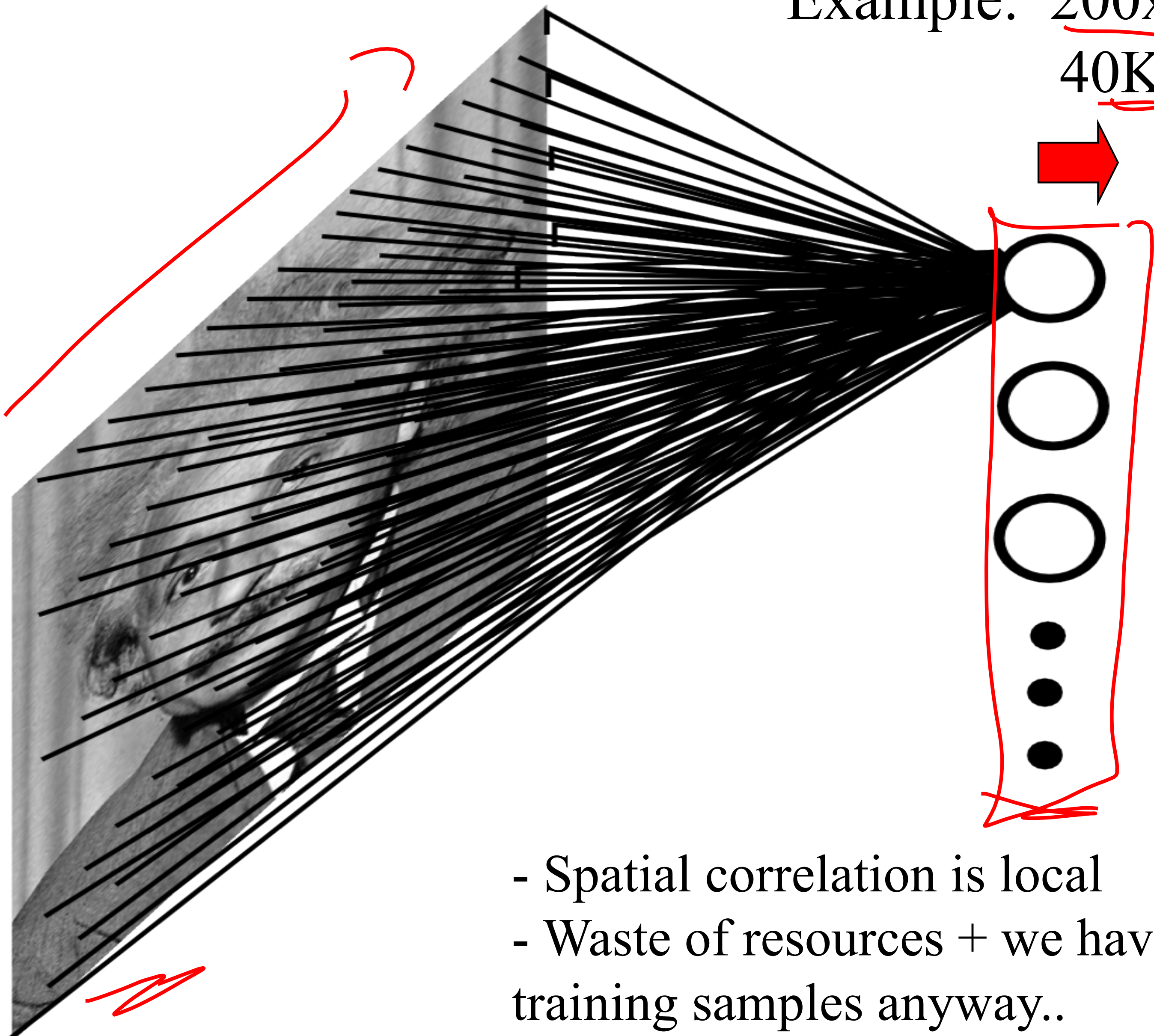
Fully Connected Layer

4×10^4

Example: 200x200 image

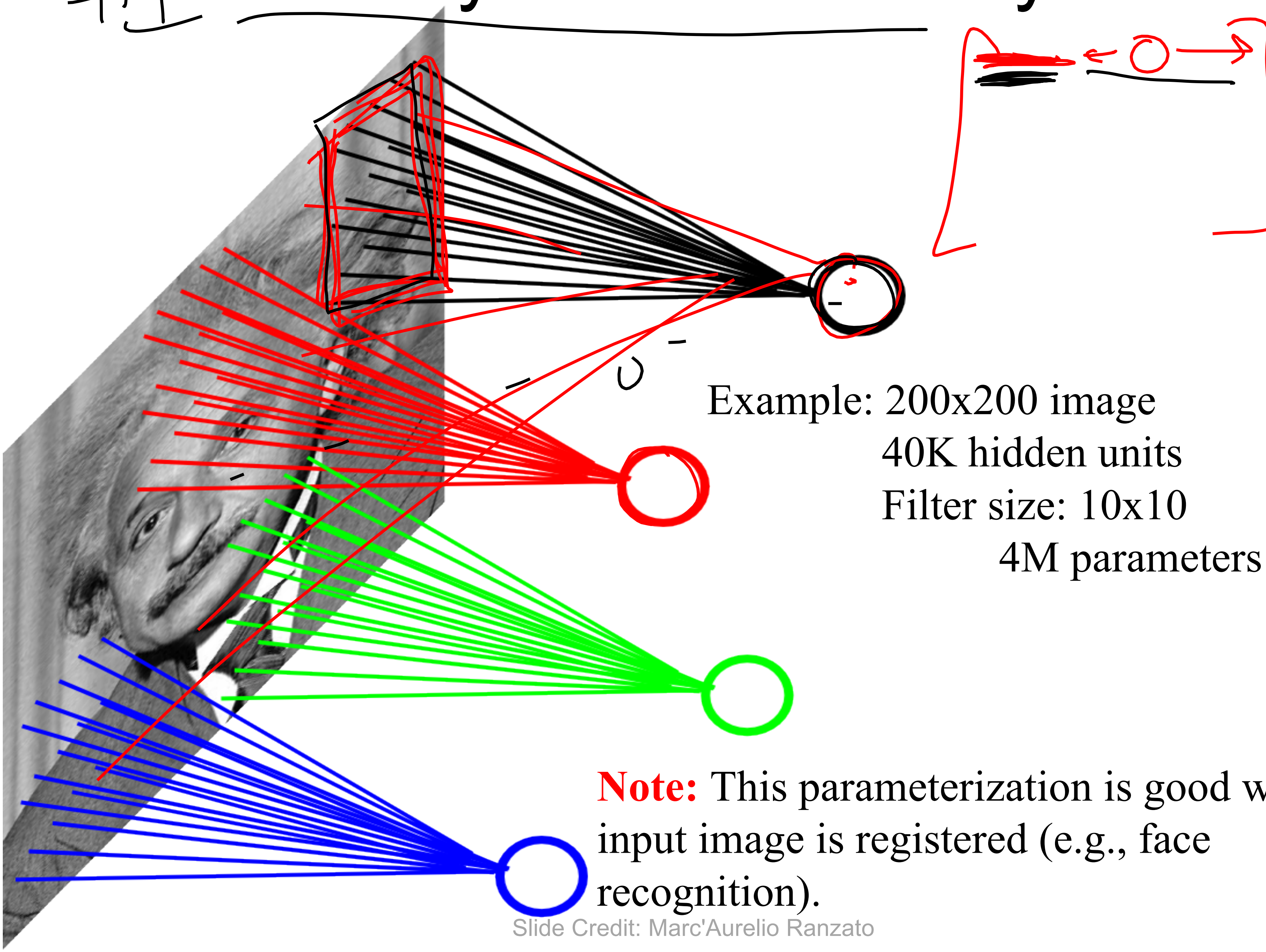
40K hidden units

~2B parameters!!!



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

#1 Locally Connected Layer

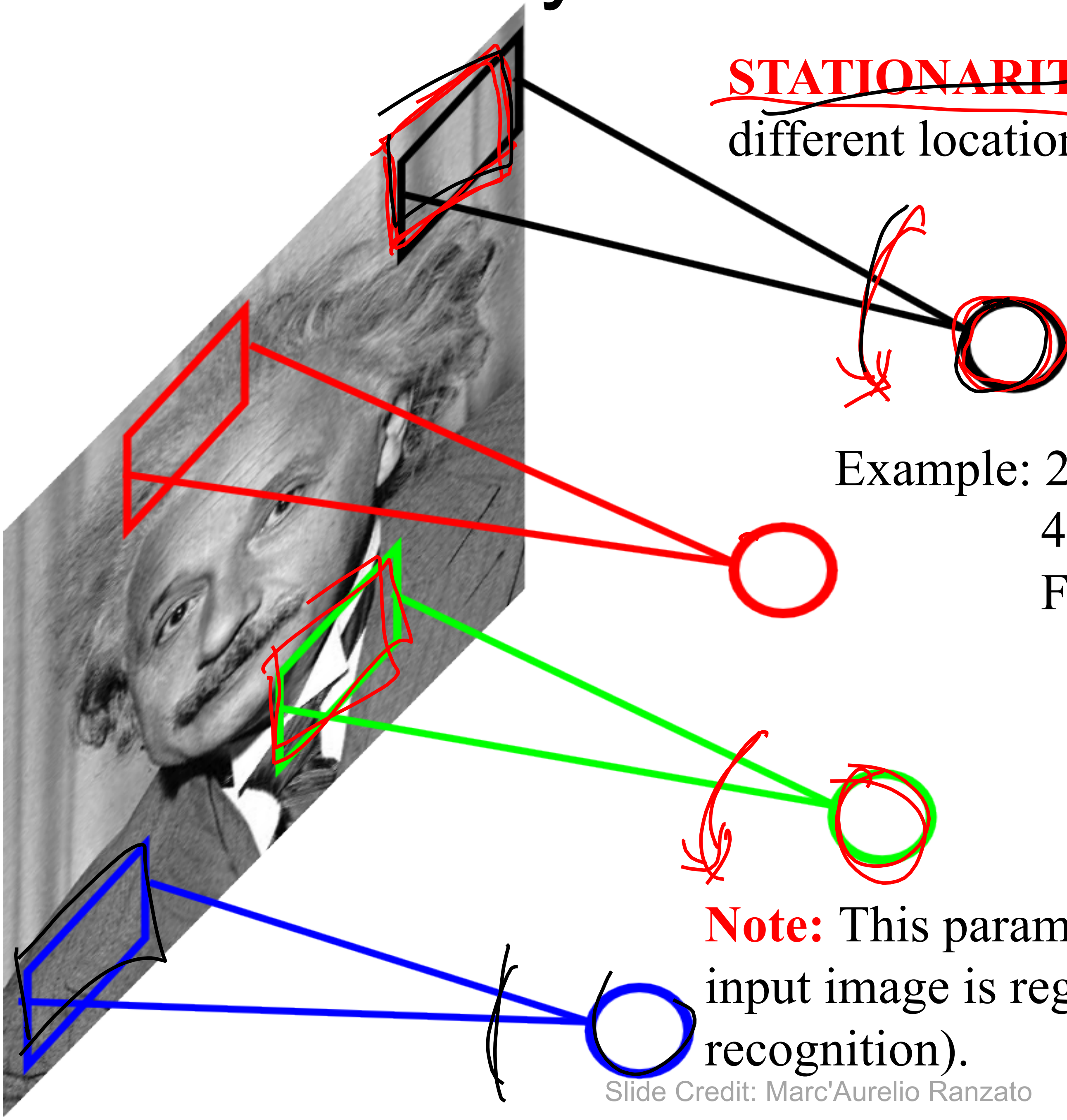


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

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

Slide Credit: Marc'Aurelio Ranzato

Locally Connected Layer

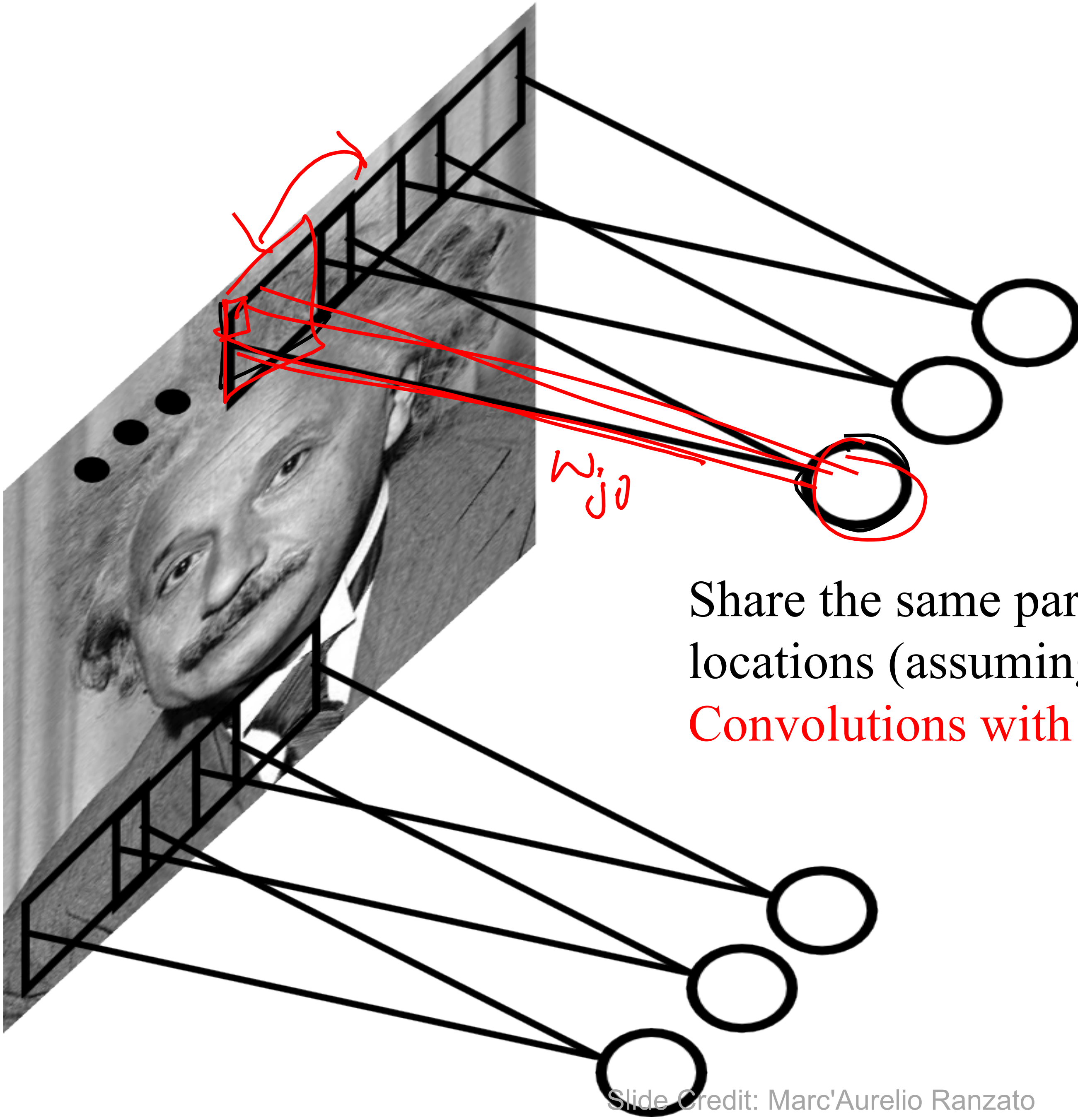


STATIONARITY? Statistics is similar at different locations

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

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

Convolutional Layer



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

Convolutions for mathematicians

$x(t) \quad y(t) \quad w(t)$

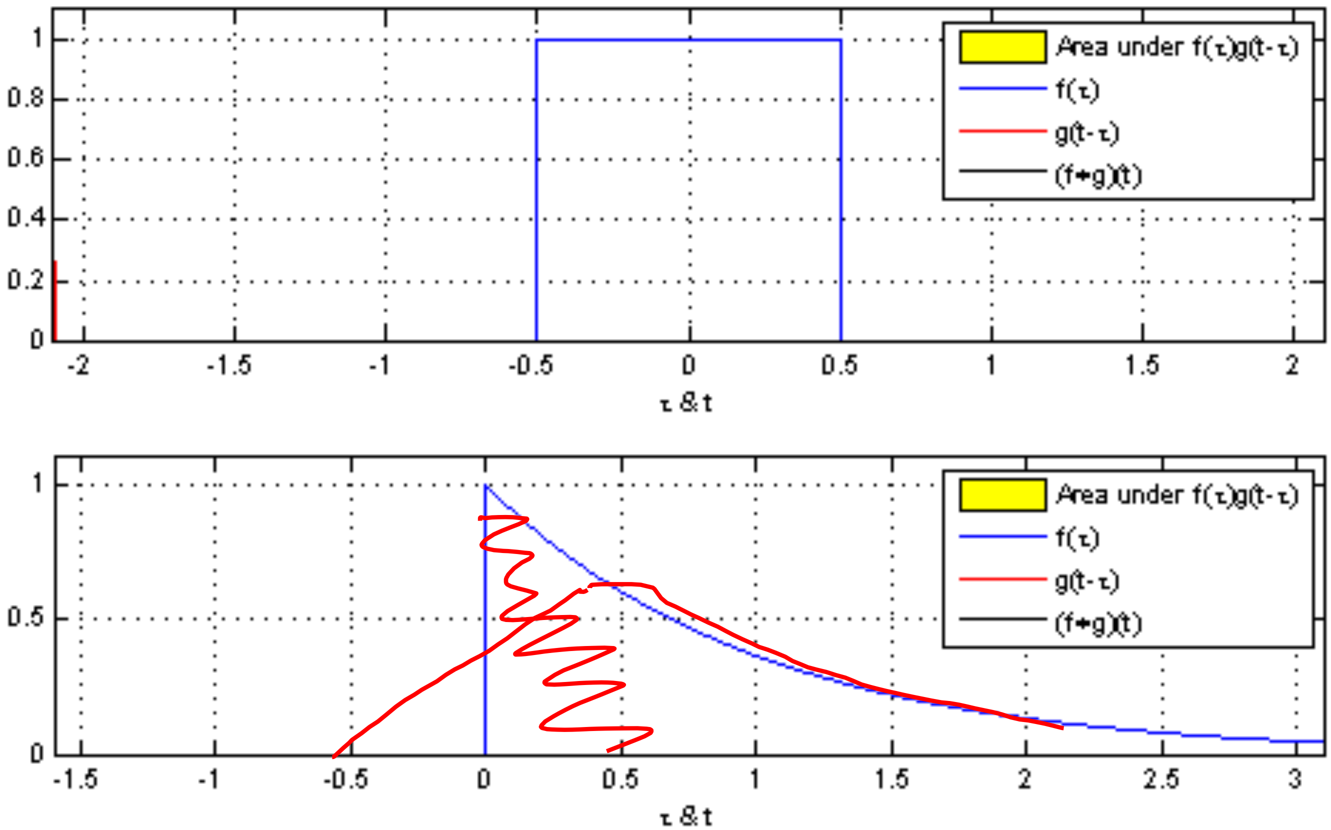
$$y(t) = (x * w)(t) = \int_{a=-\infty}^{\infty} x(t-a)\, w(a)\, da$$

$$= \int x(a)\, w(t-a)\, da$$

$w(a) \to w(-a)$

$w(-a) \to w(-(a-t))$

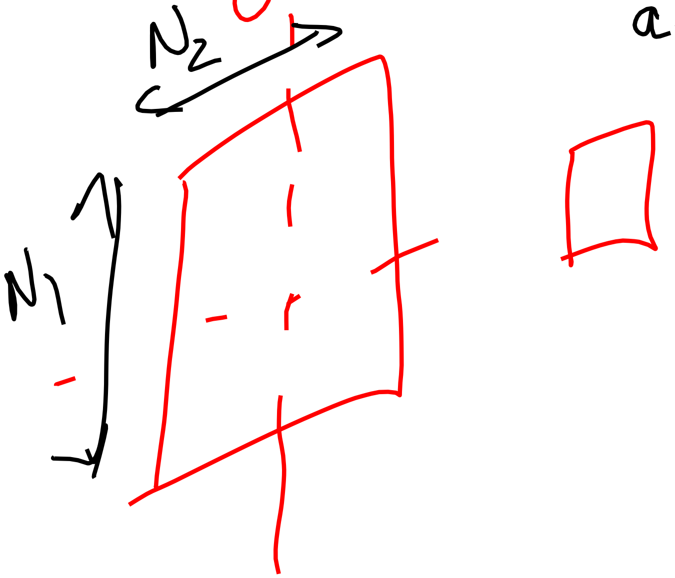
$$y(t_1,t_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(\underset{t_1-a}{\cancel{a-t_1}}, \underset{t_2-b}{\cancel{b-t_2}})\, w(a,b)\, da\, db$$



"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

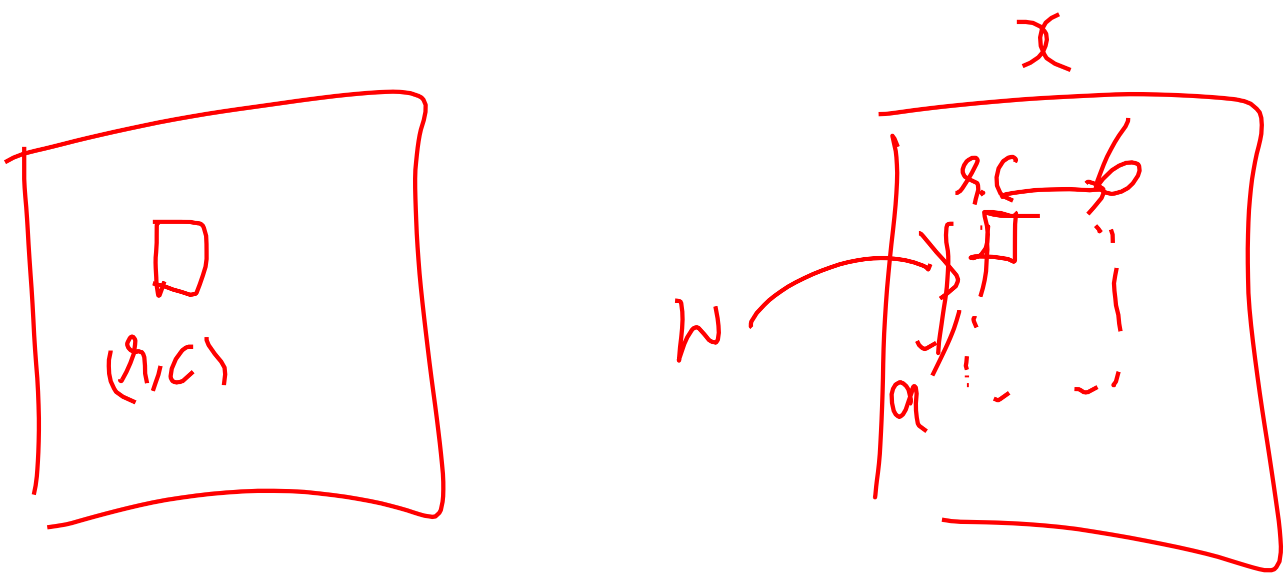
Convolutions for computer scientists

$$y[r,c] = \sum_{a=-\frac{N_1-1}{2}}^{\frac{N-1}{2}} \sum_{b=-\frac{N-1}{2}}^{+\frac{N-1}{2}} x[\underline{r-a}, \underline{c-b}]\, w[a,b]$$



Convolutions for programmers

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



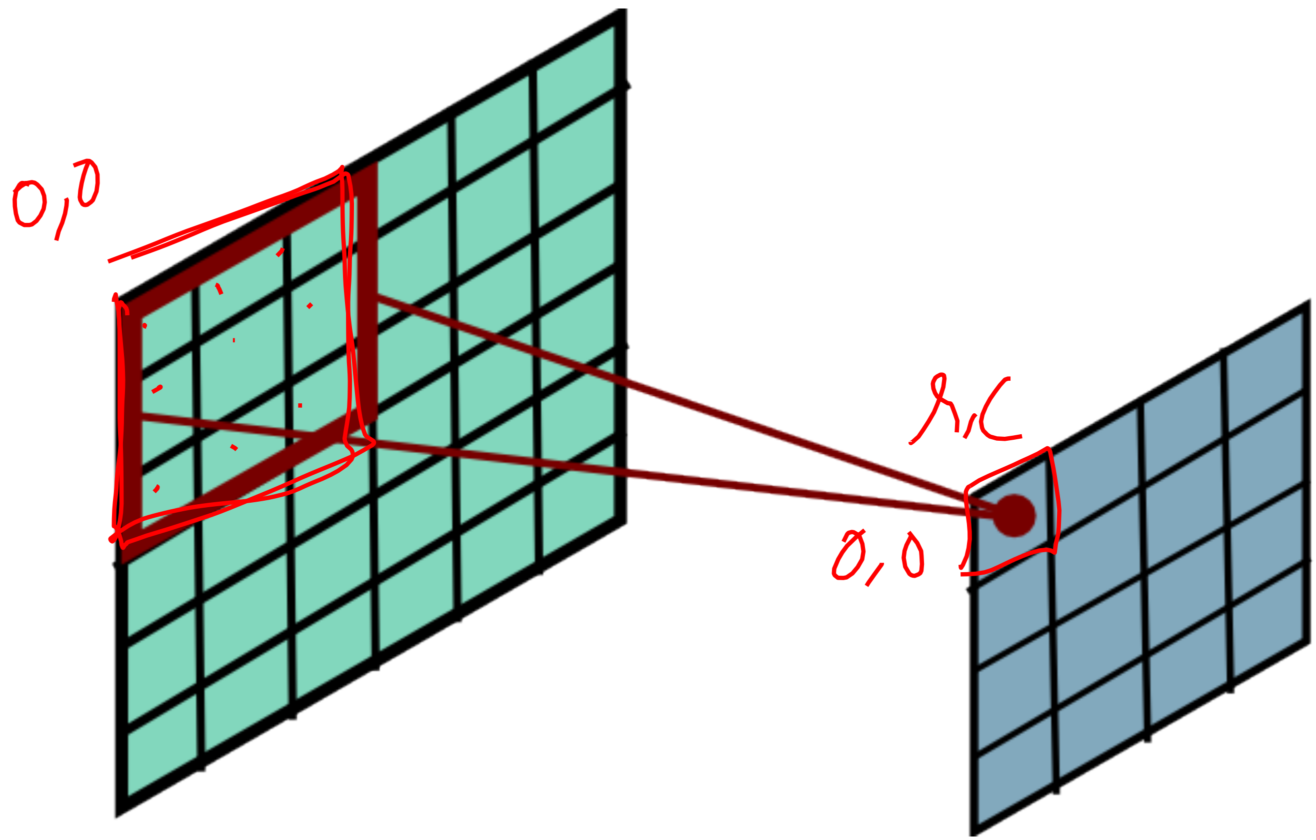
Convolution Explained

- http://setosa.io/ev/image-kernels/
- https://github.com/bruckner/deepViz

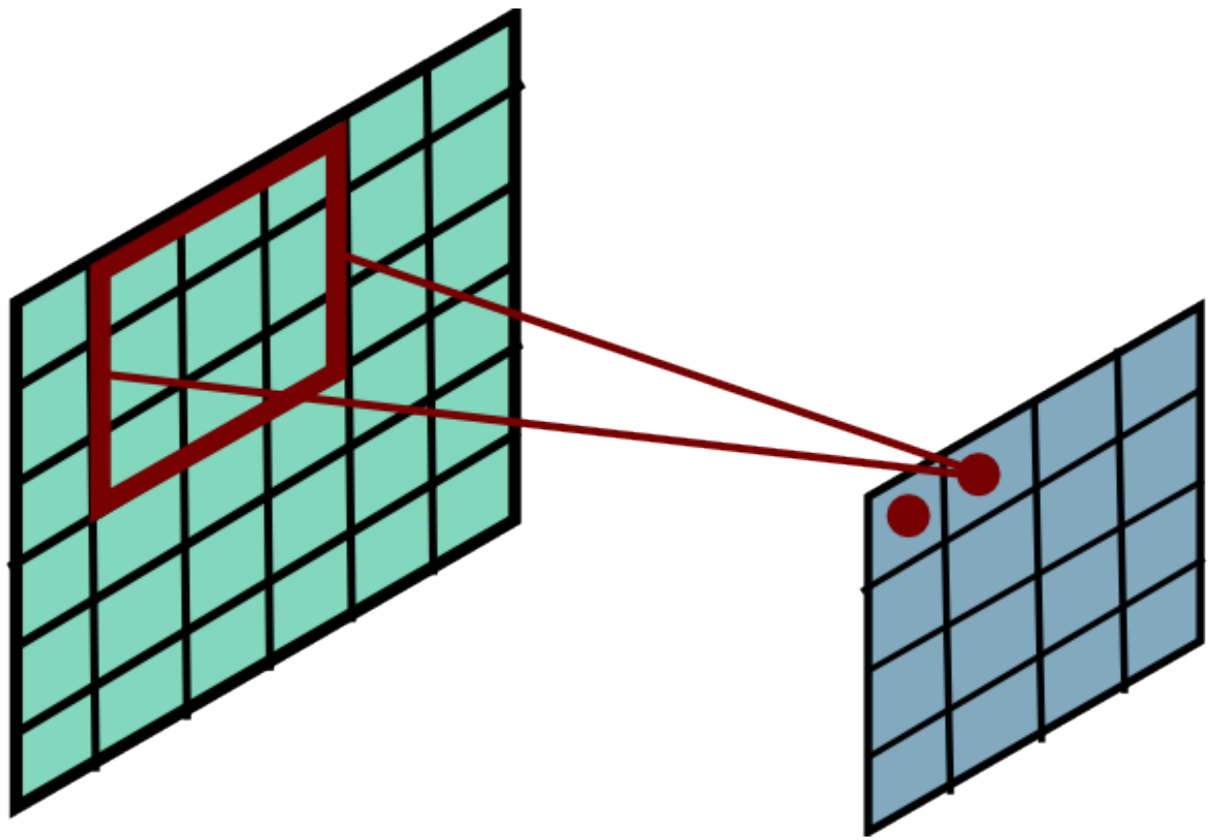
Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers

Convolutional Layer

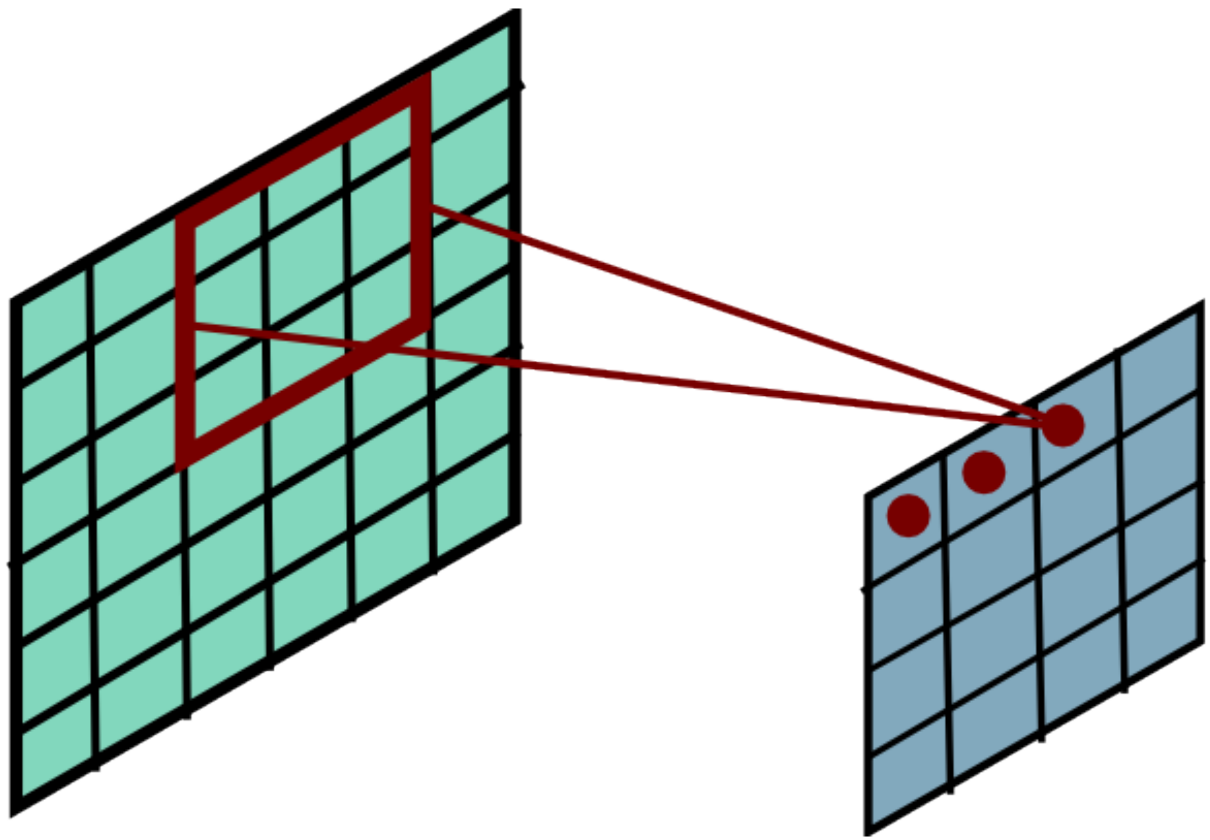


Convolutional Layer



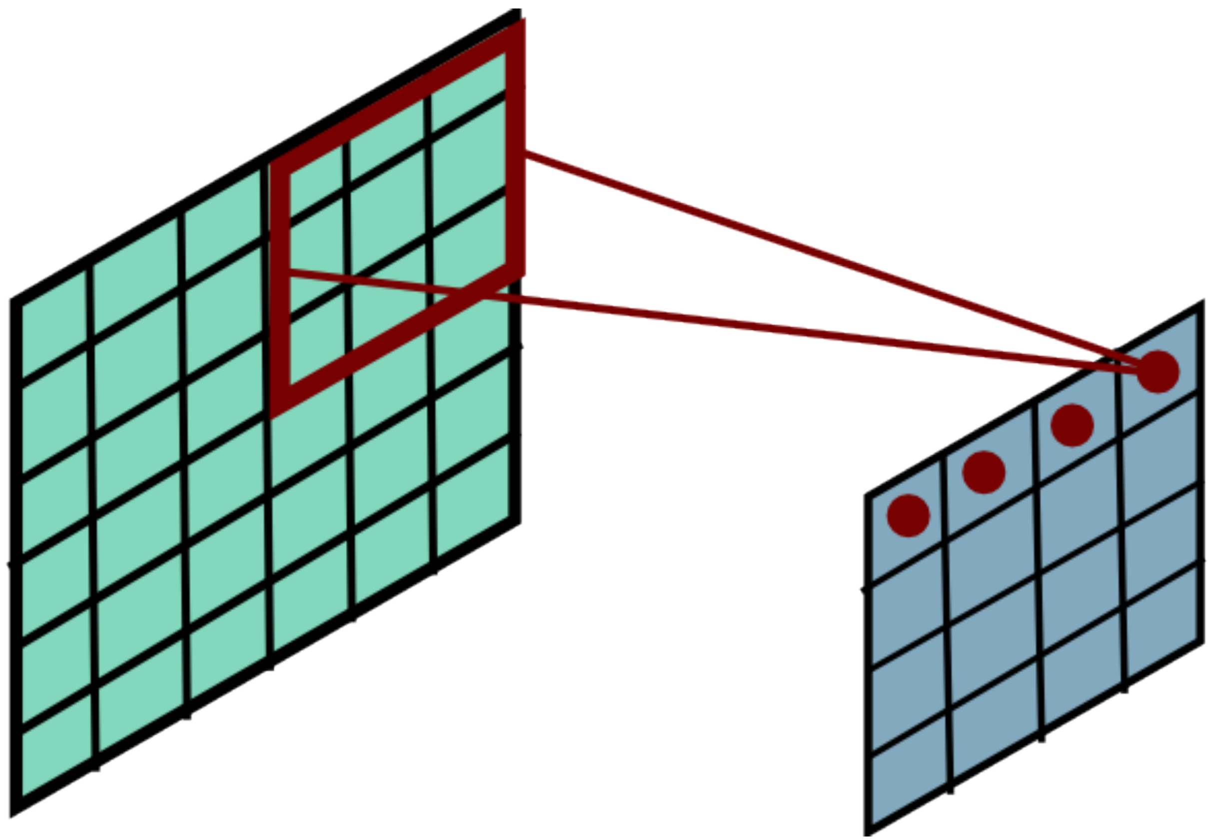
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

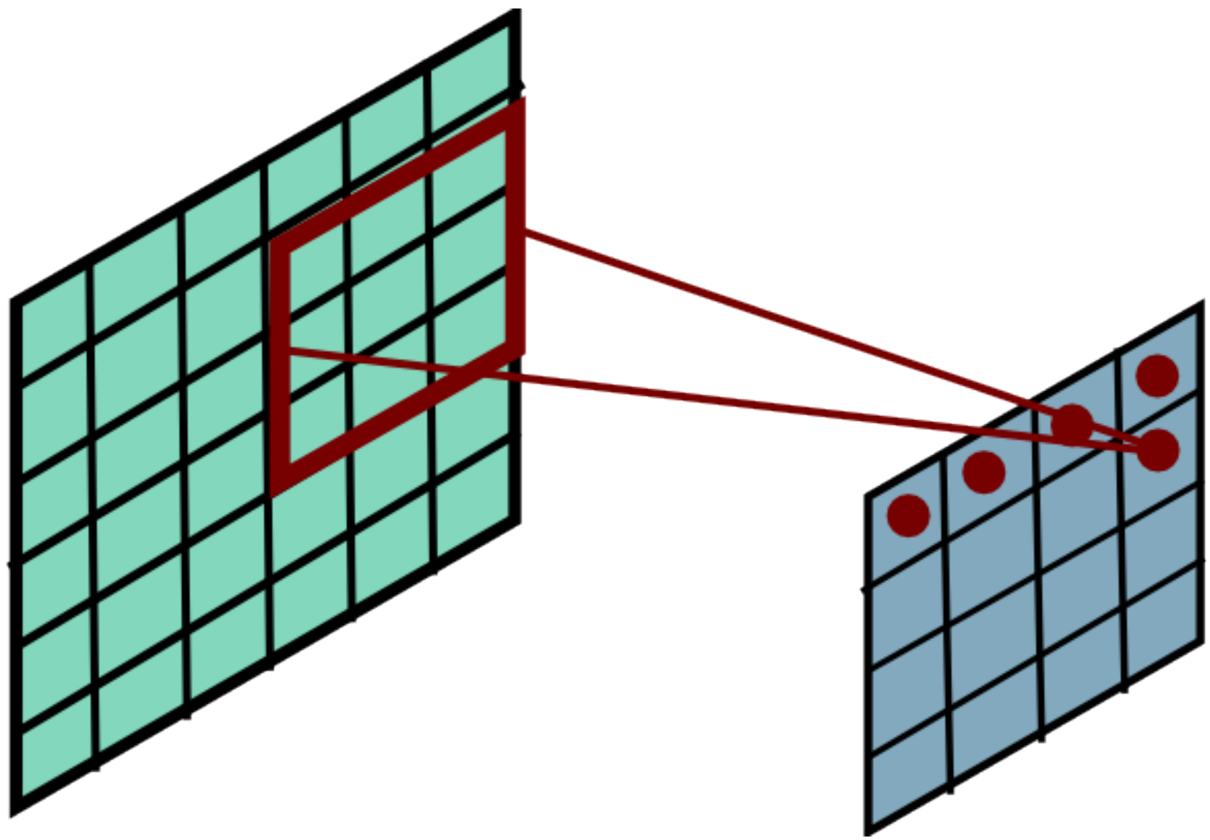


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

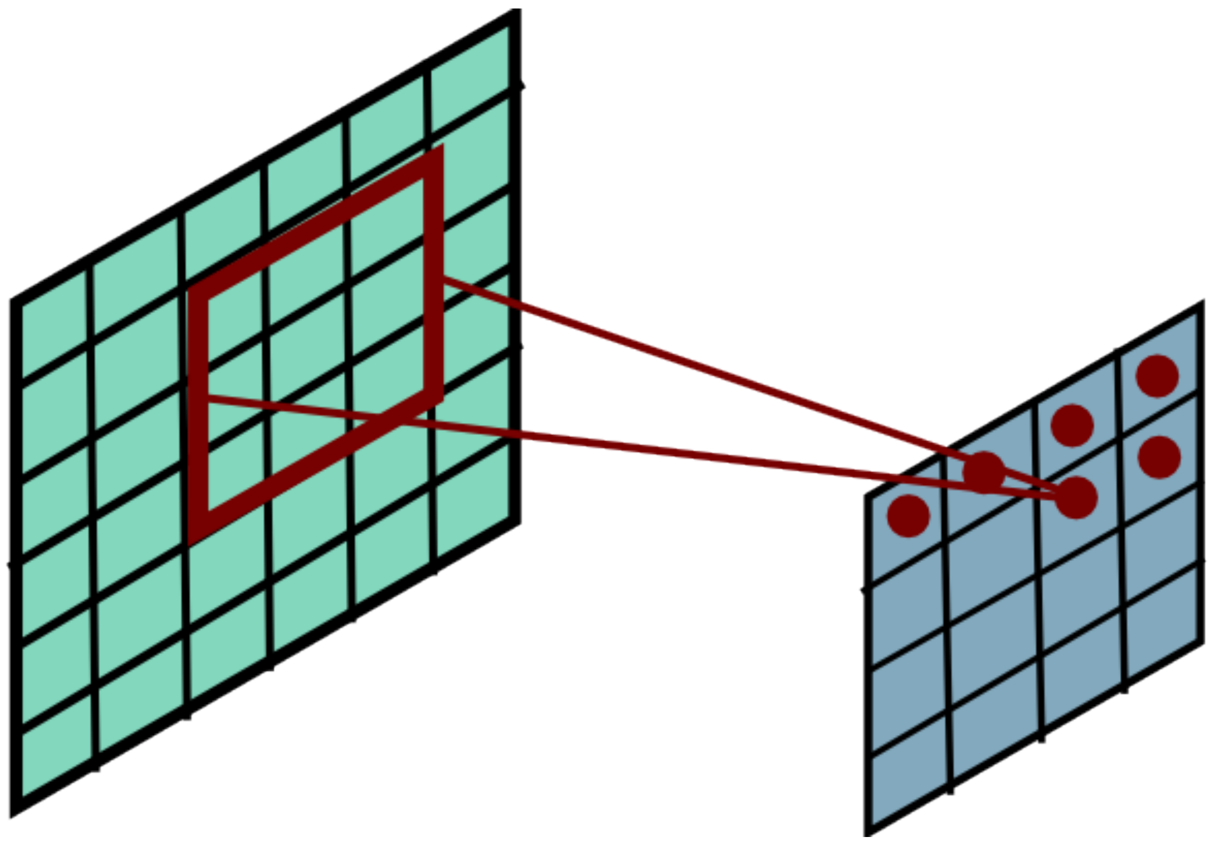


Convolutional Layer

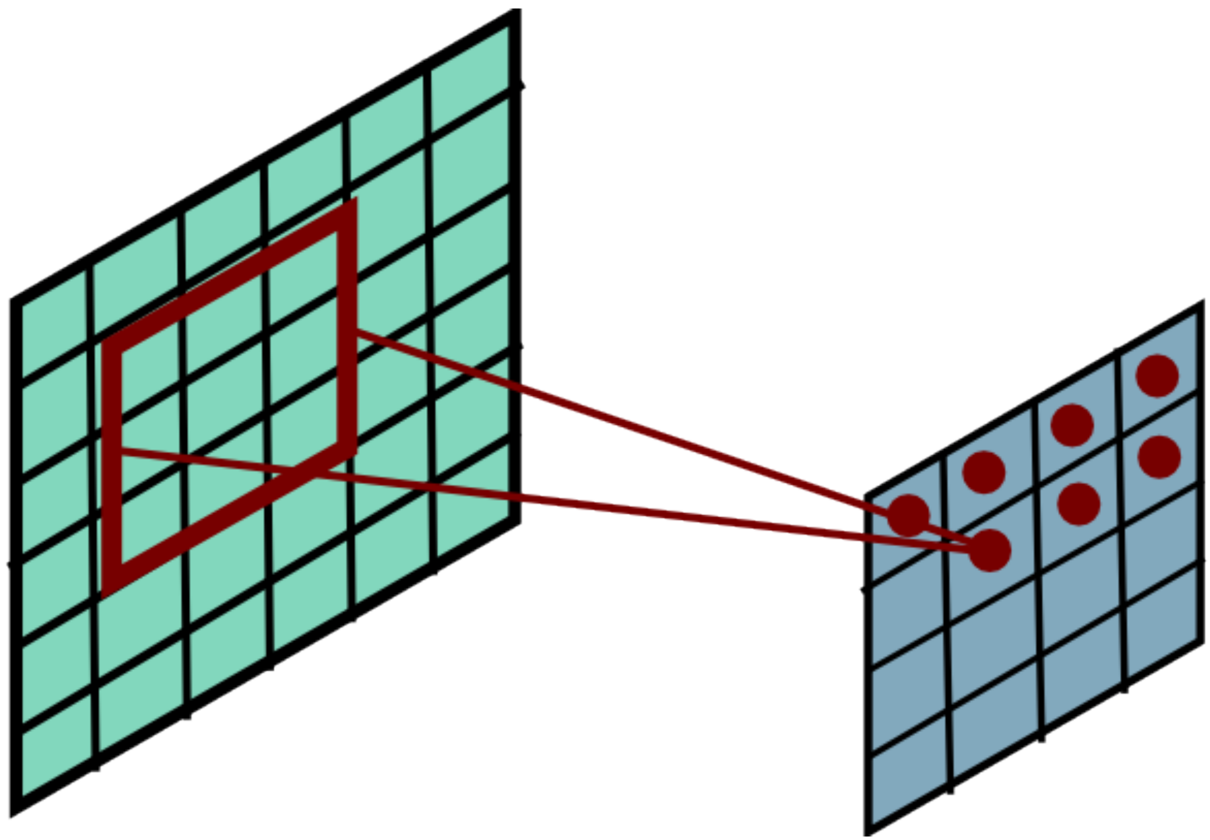


Slide Credit: Marc'Aurelio Ranzato

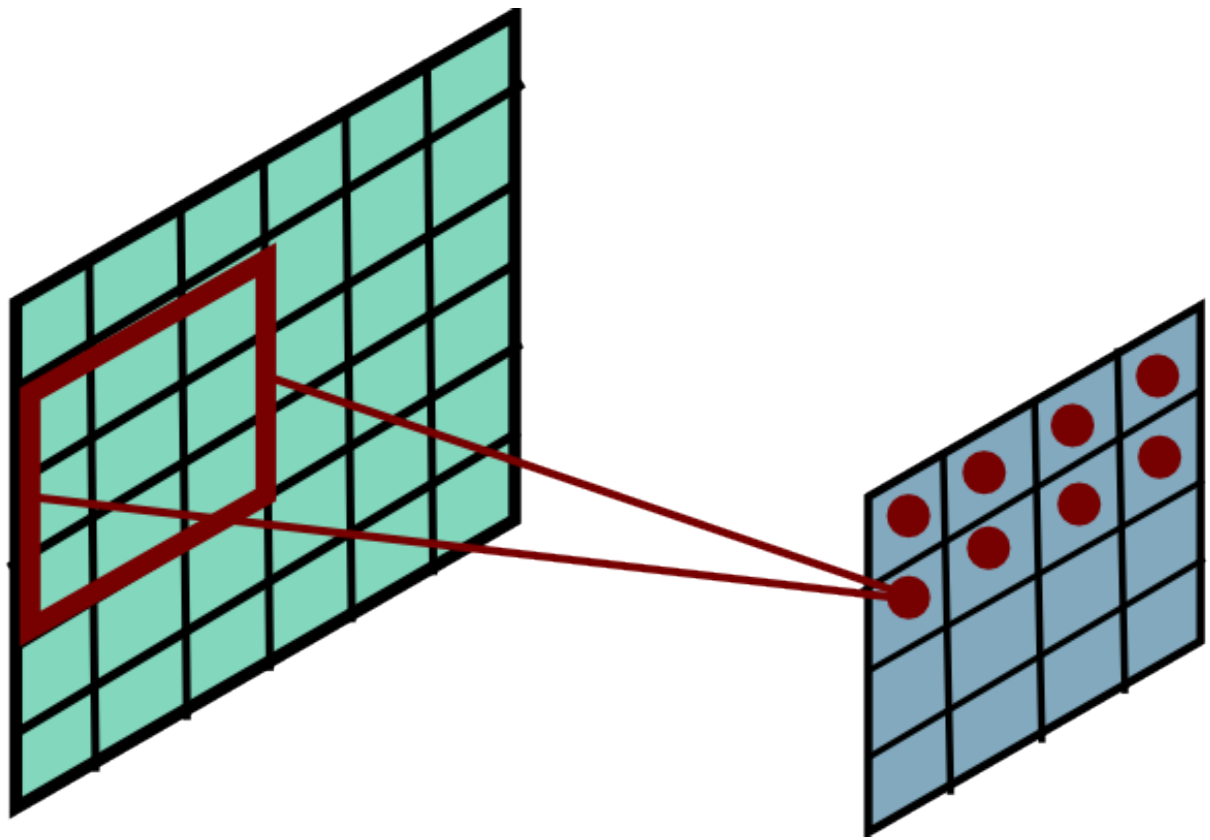
Convolutional Layer



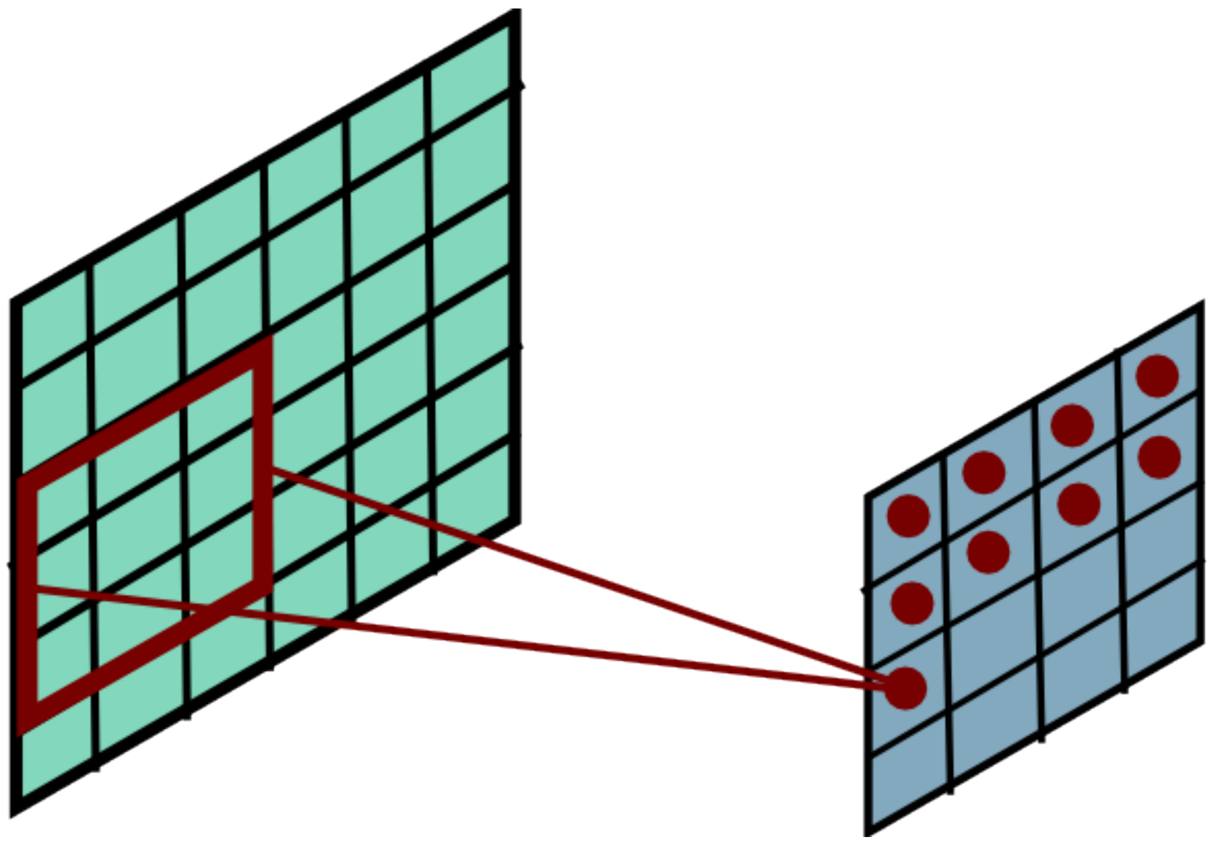
Convolutional Layer



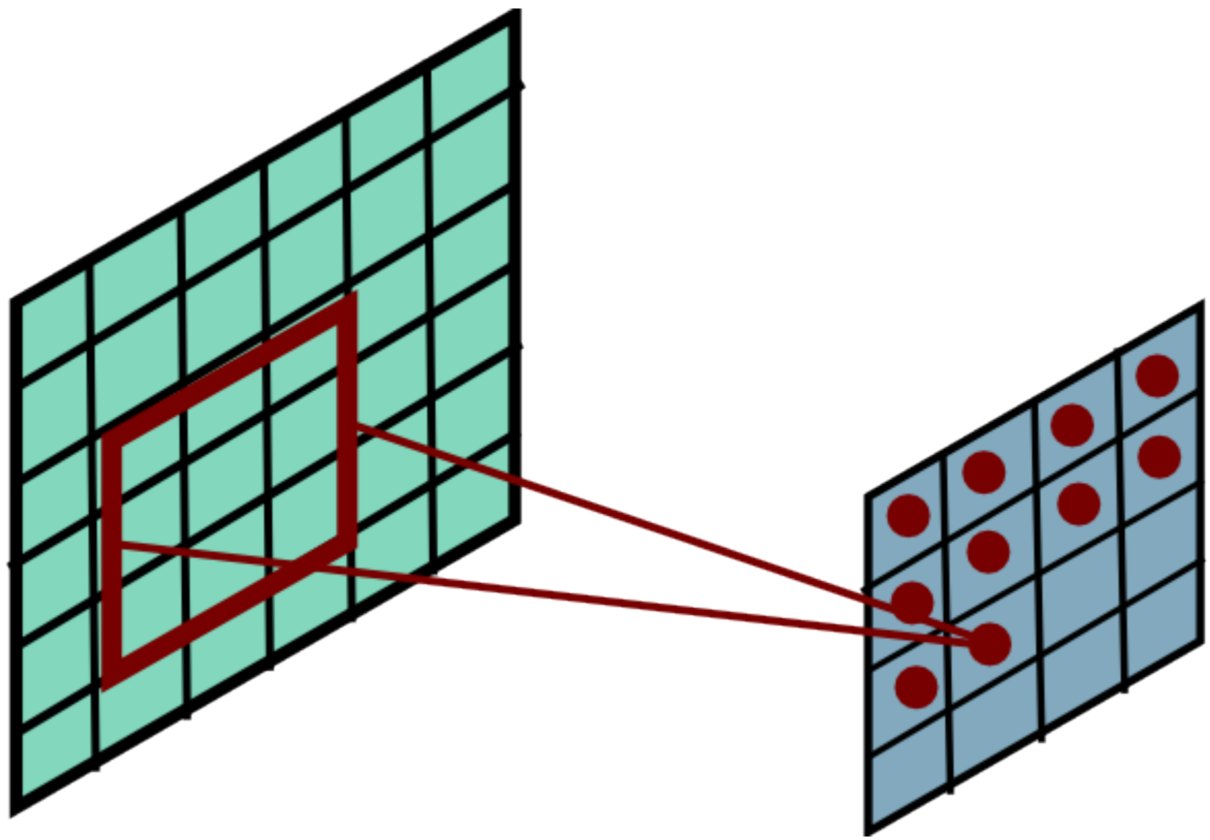
Convolutional Layer



Convolutional Layer

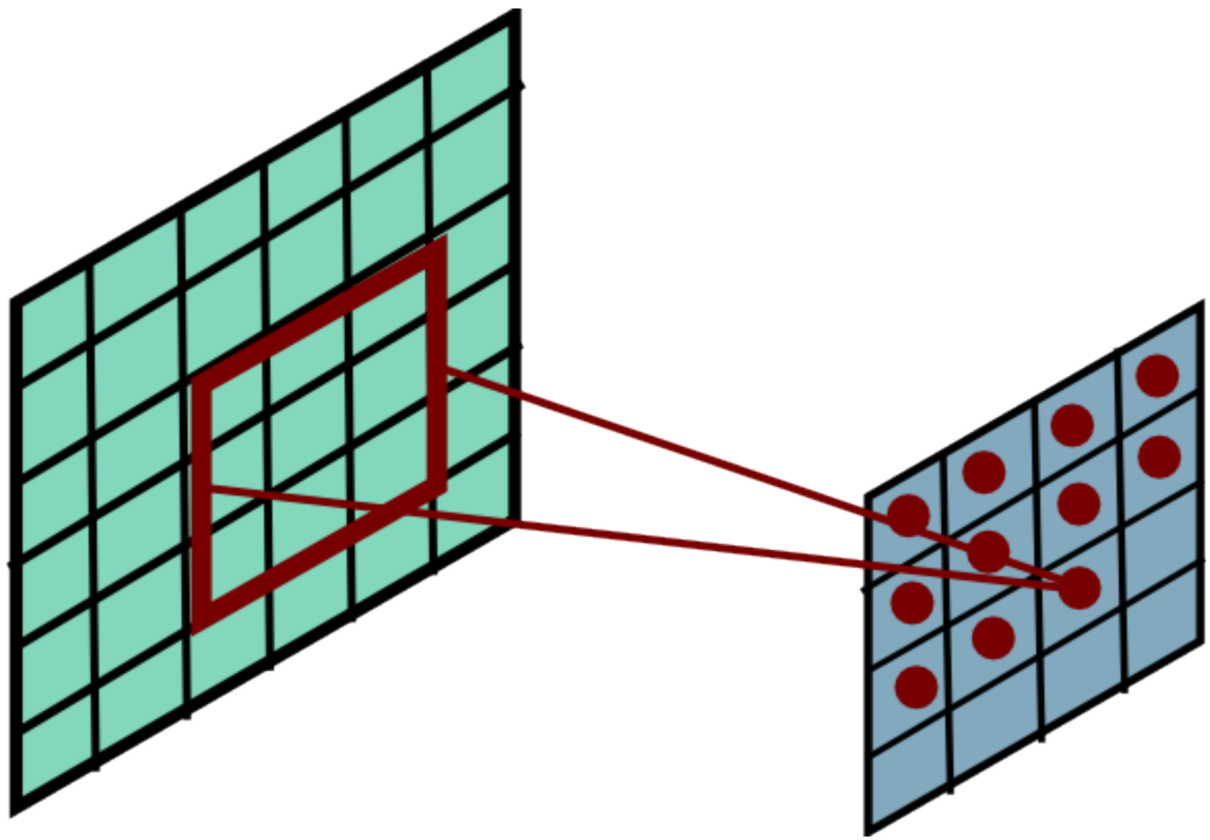


Convolutional Layer



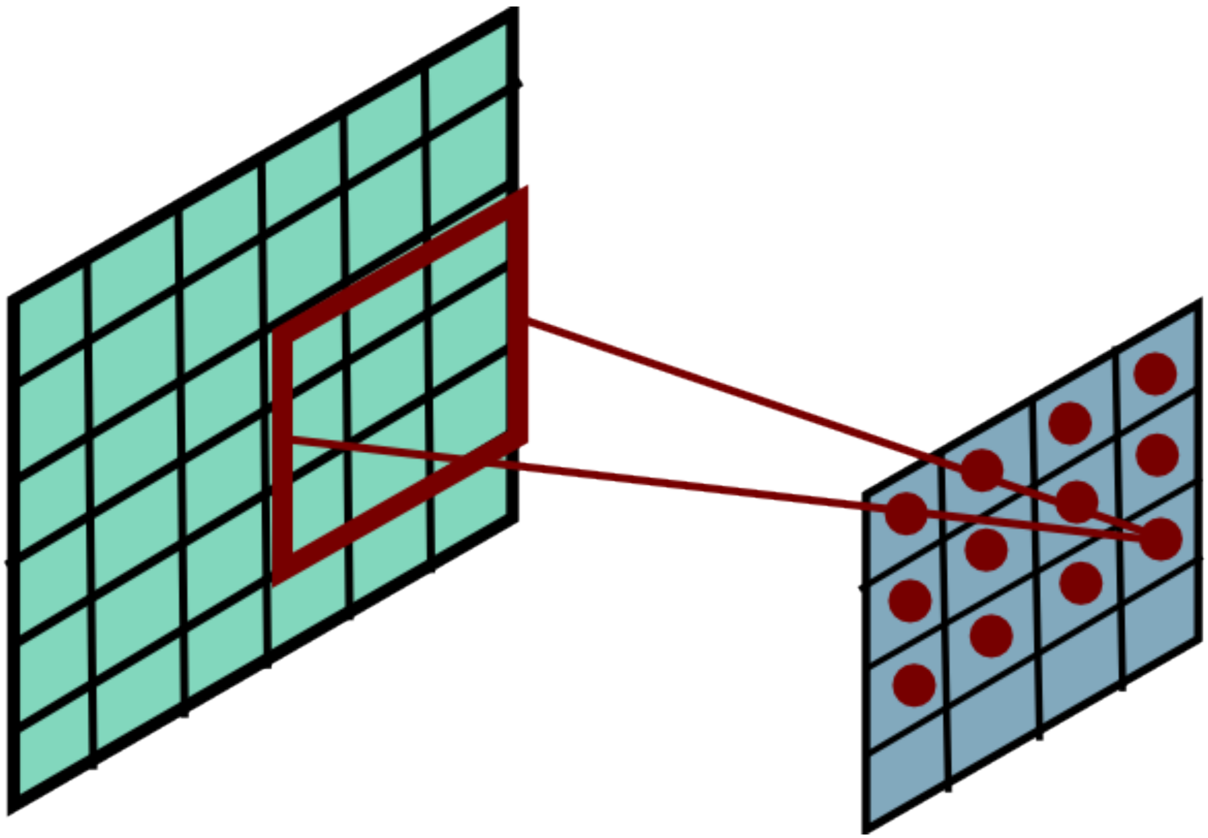
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

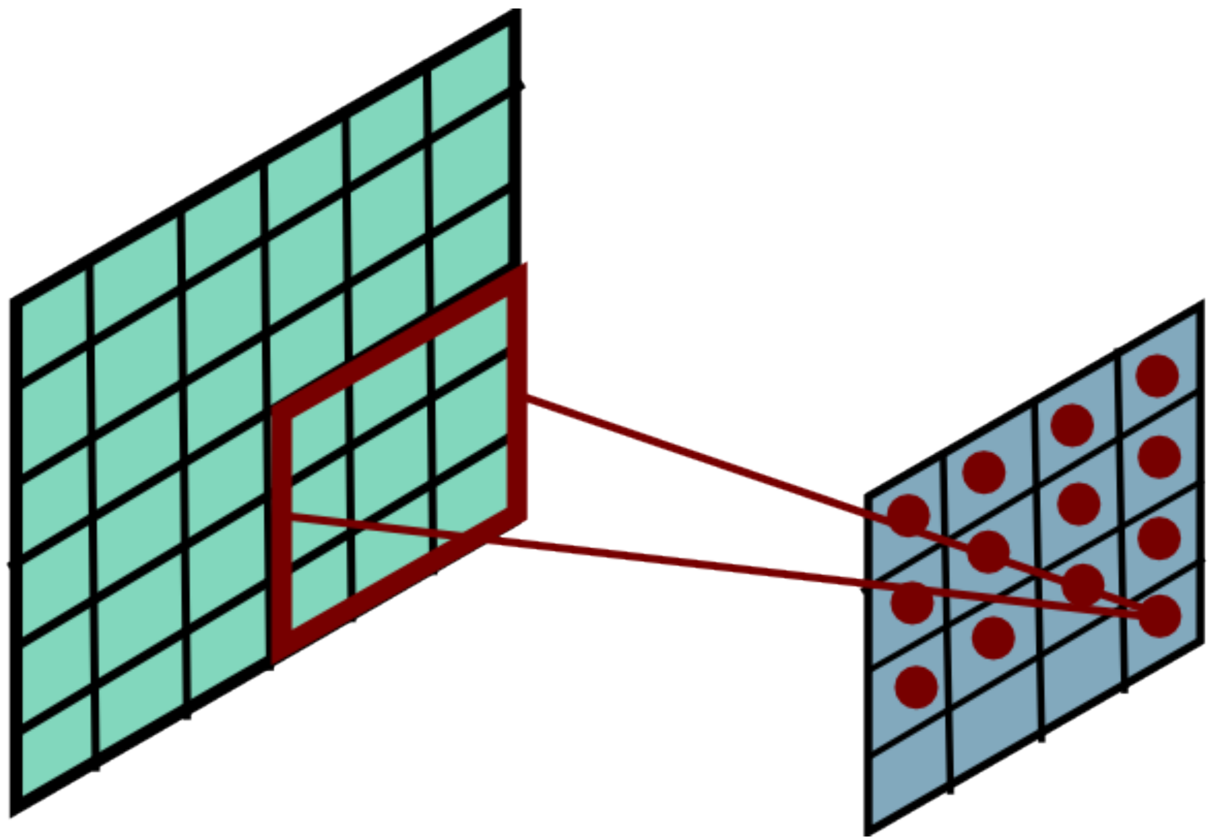


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

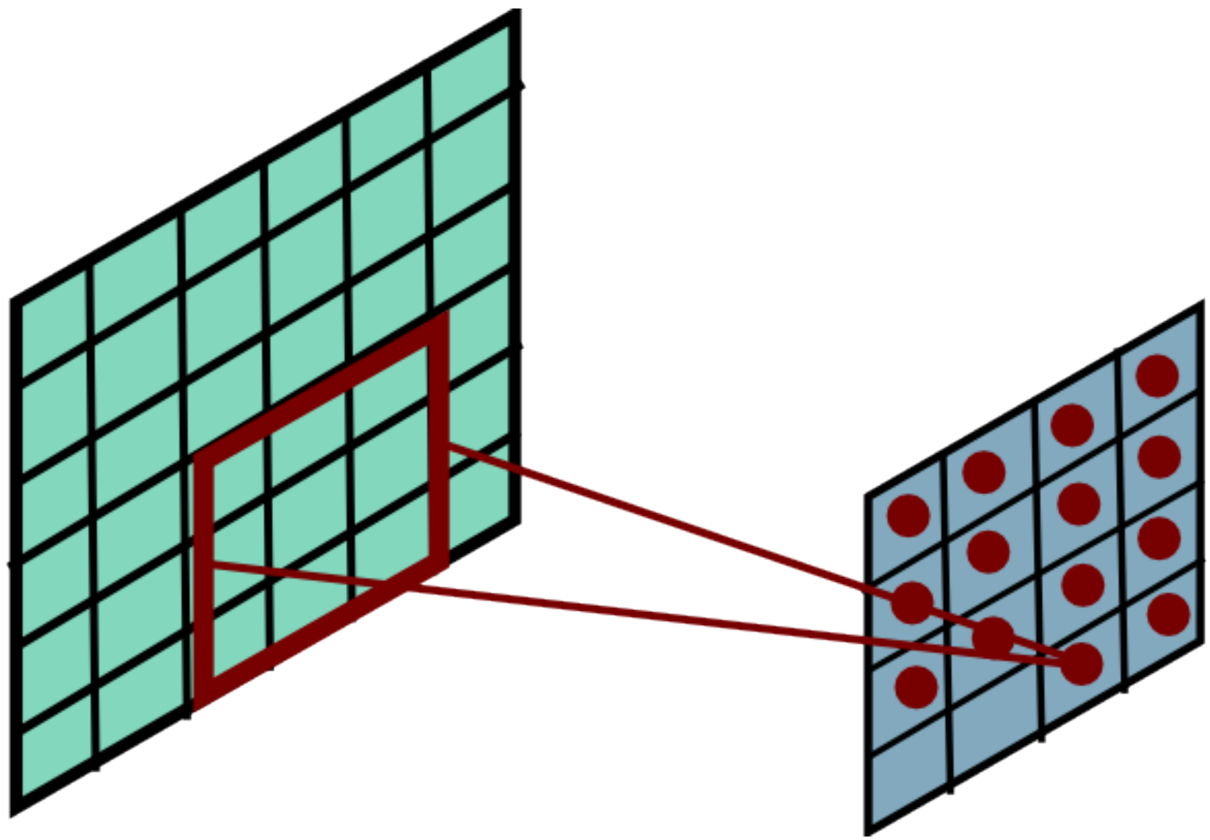


Convolutional Layer

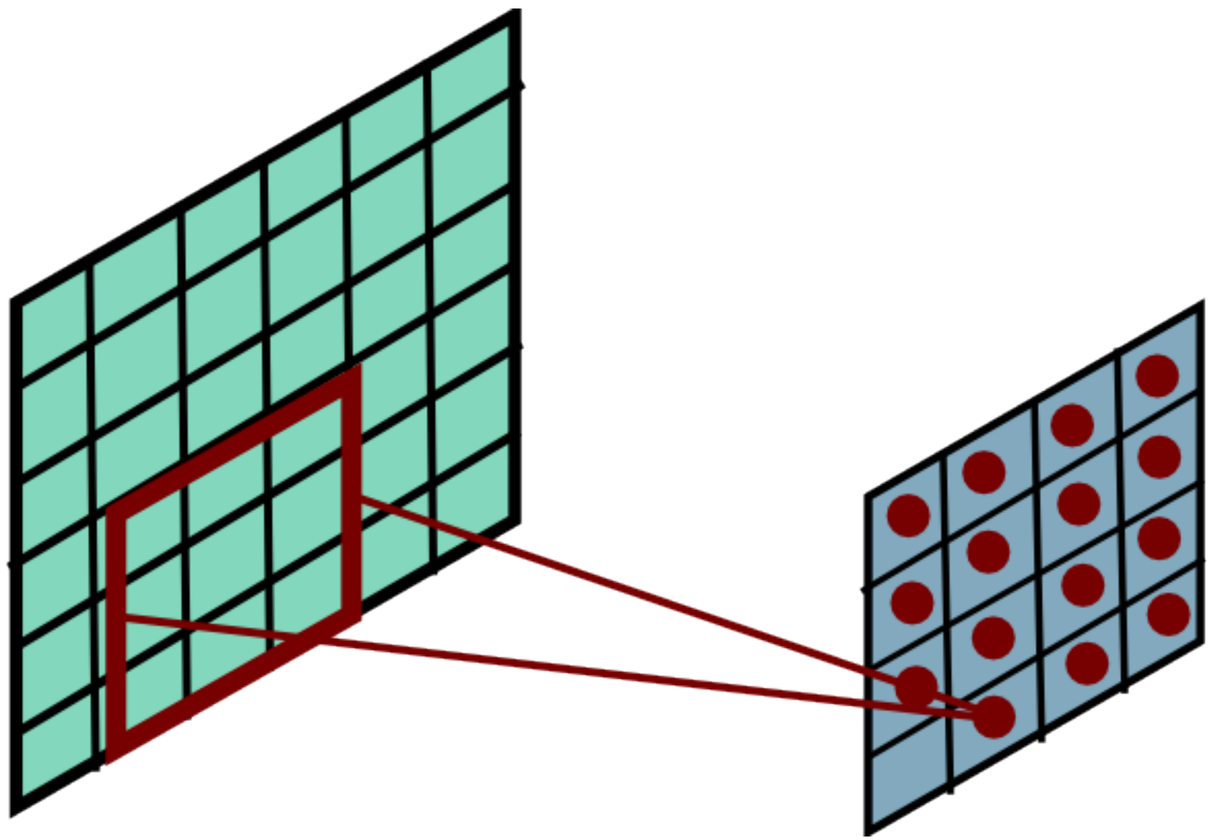


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

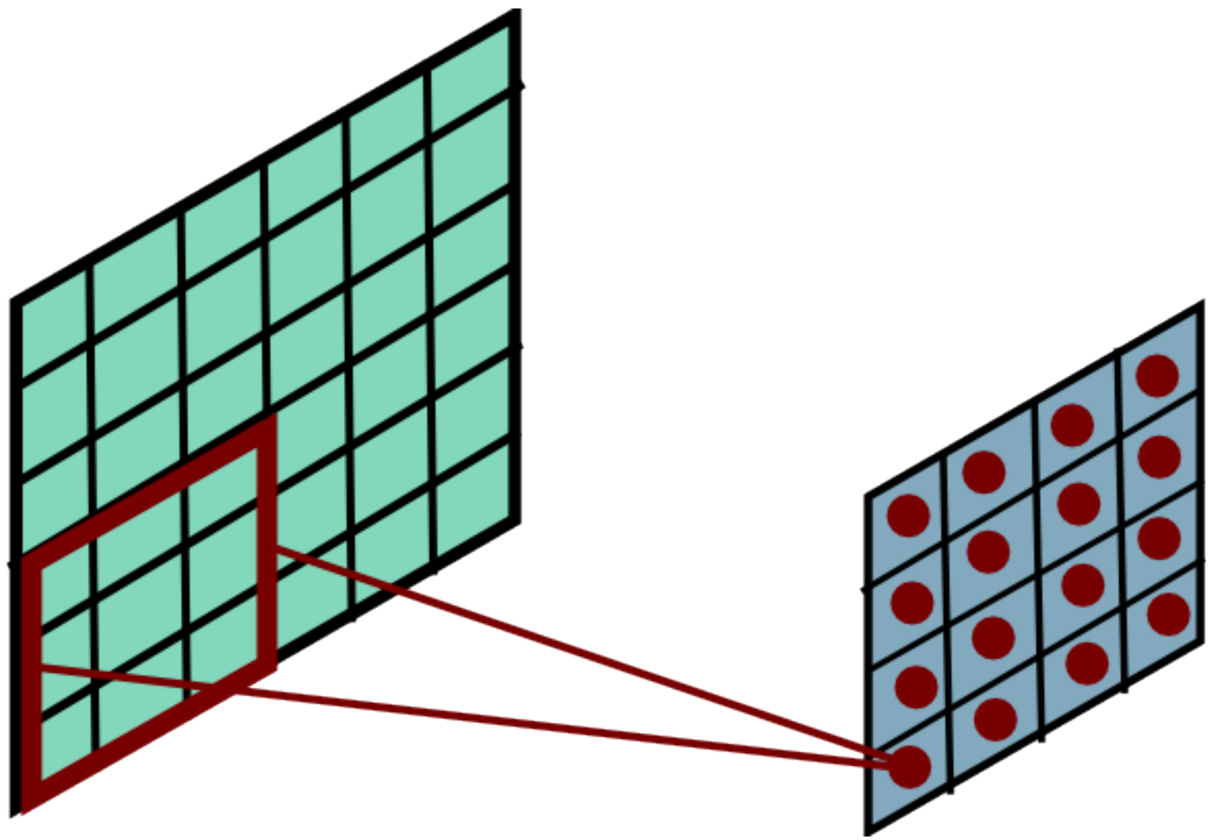


Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

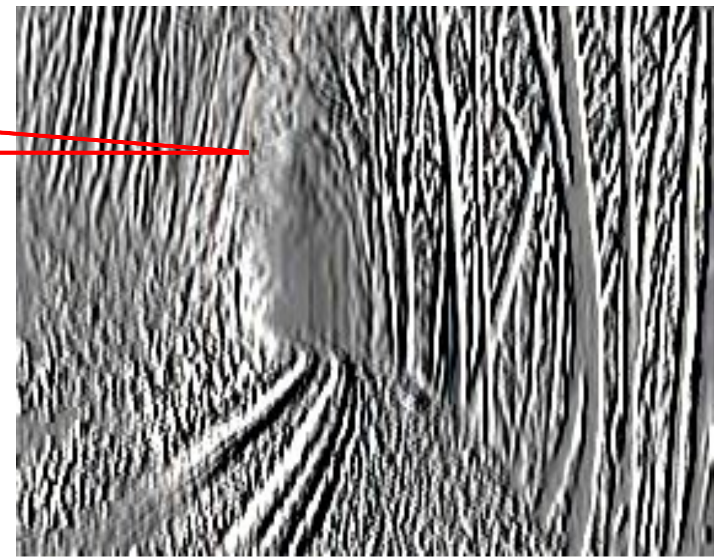
Convolutional Layer



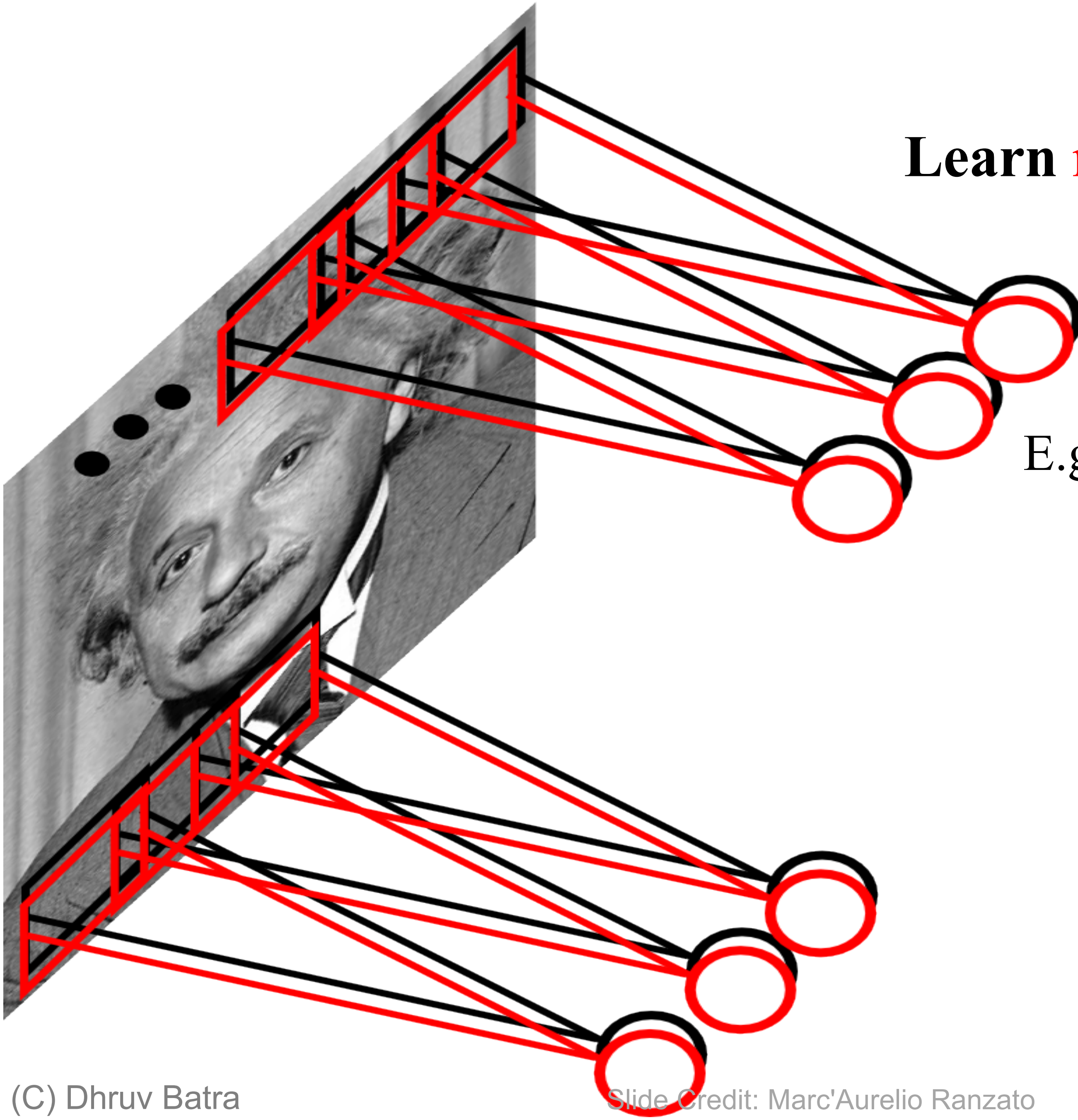
Mathieu et al. "Fast training of CNNs through FFTs" ICLR 2014

Convolutional Layer

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



Convolutional Layer



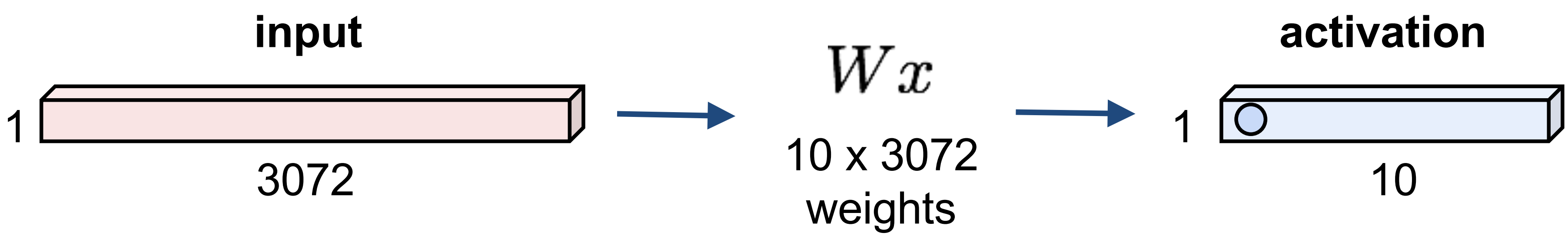
Learn multiple filters.

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

Slide Credit: Marc'Aurelio Ranzato

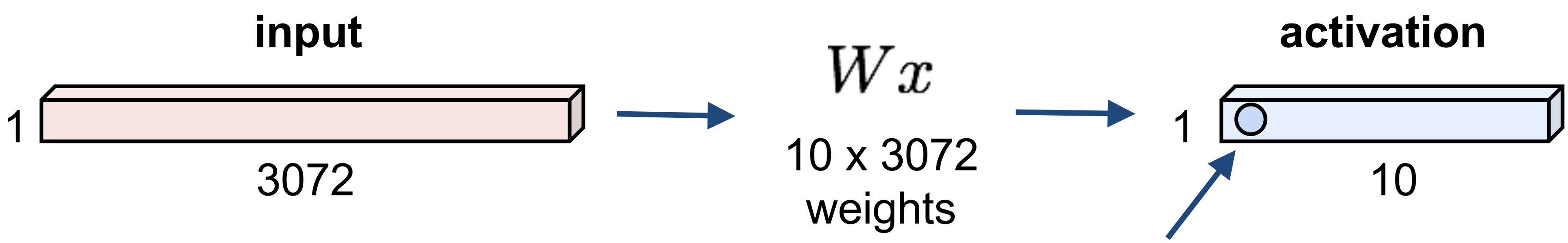
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

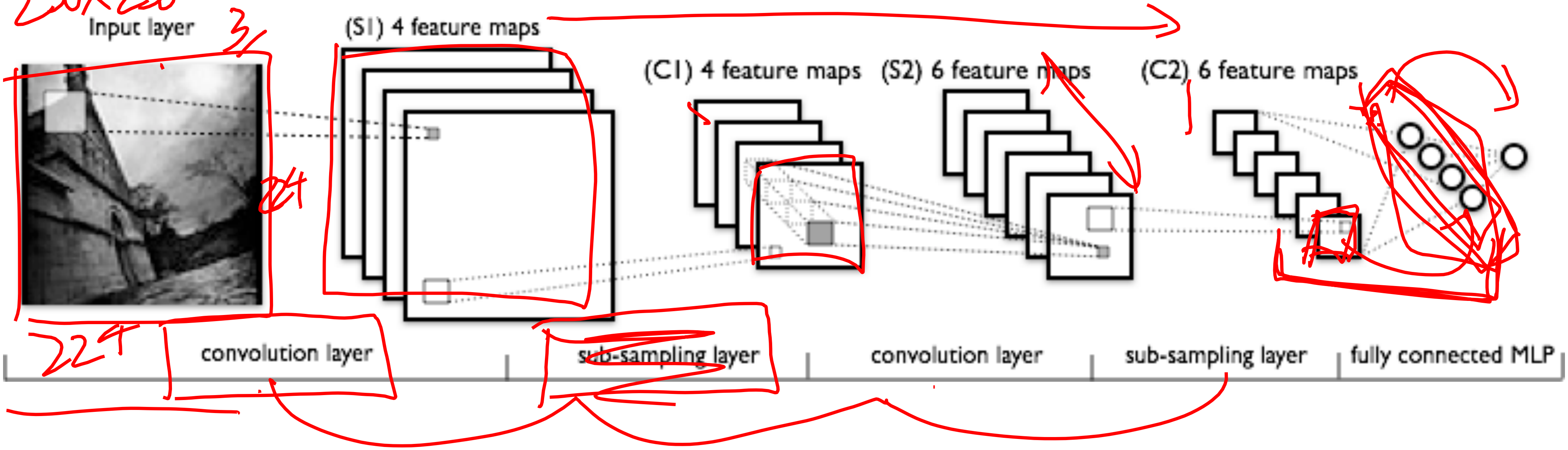
32x32x3 image -> stretch to 3072 x 1

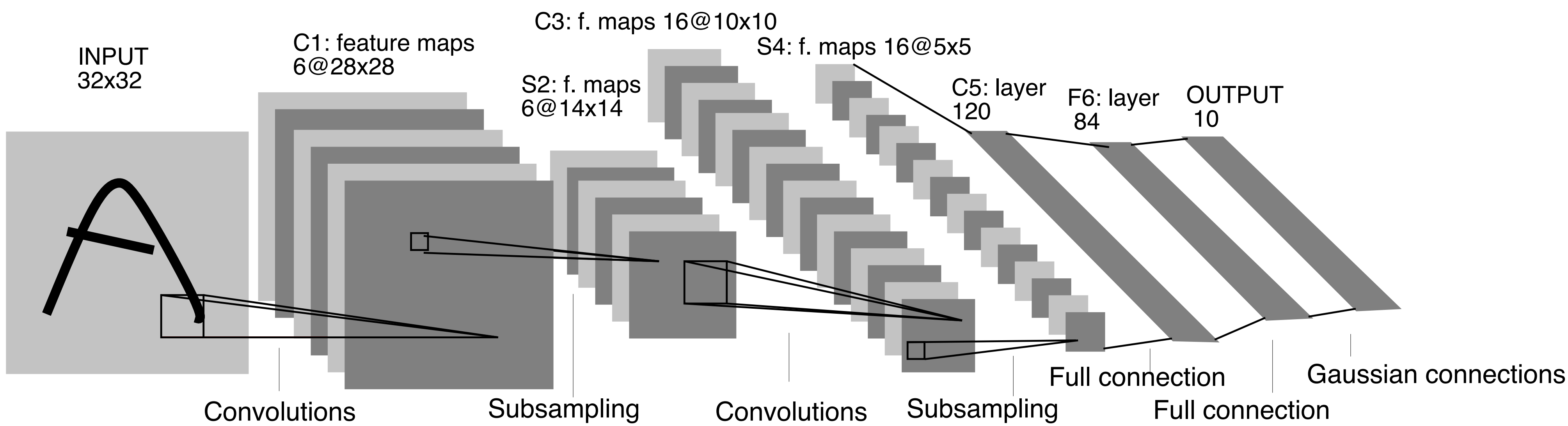


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Convolutional Neural Networks

256x256



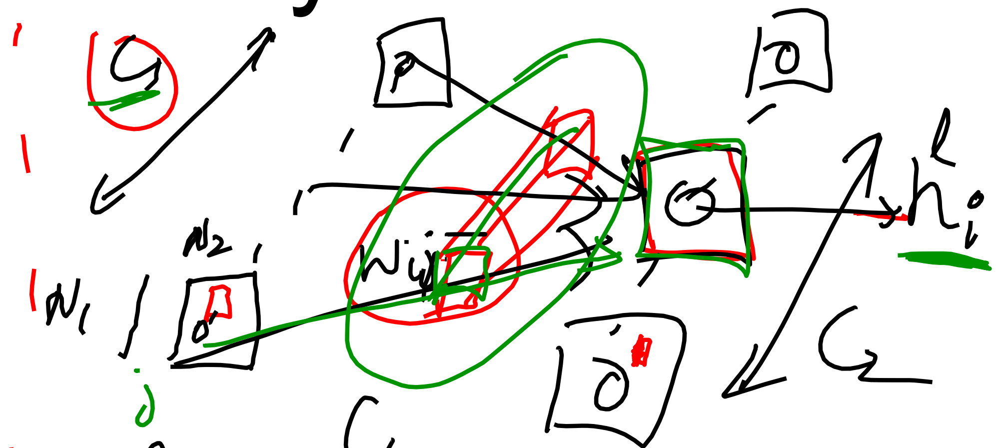


FC vs Conv Layer

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

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

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



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

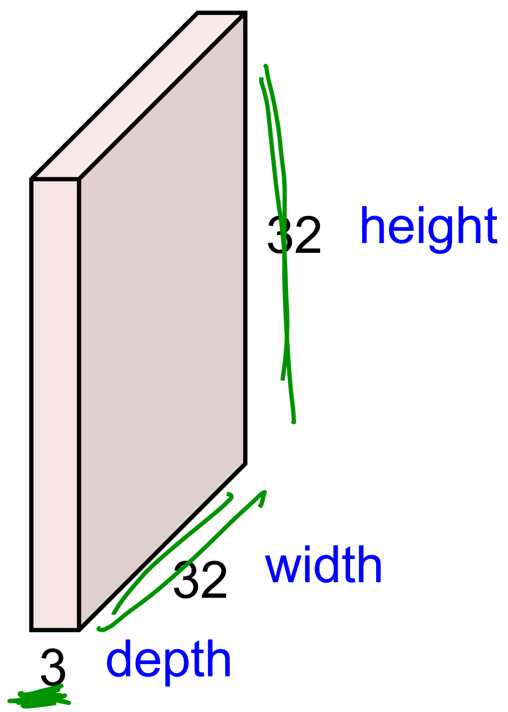
$$h_i^l[r, c] = \sum_{j=1}^{C_1} \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} h_j^{l-1}[r+a, c+b] \, W_{ij}[a, b]$$

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

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

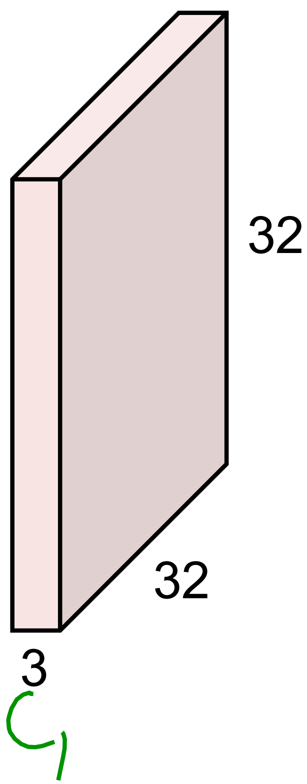
Convolution Layer

32x32x3 image -> preserve spatial structure



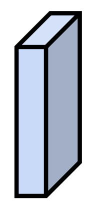
Convolution Layer

32x32x3 image



$k_1 \; k_2 \times C_1$

5x5x3 filter

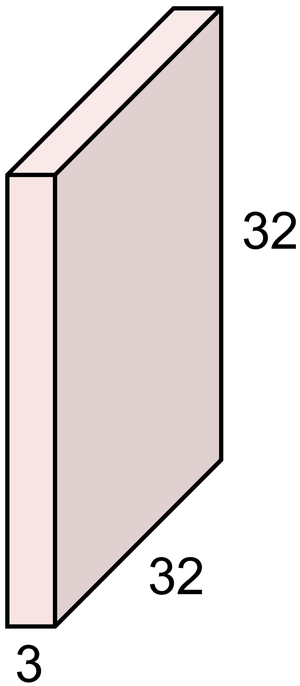


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

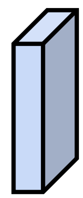
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter



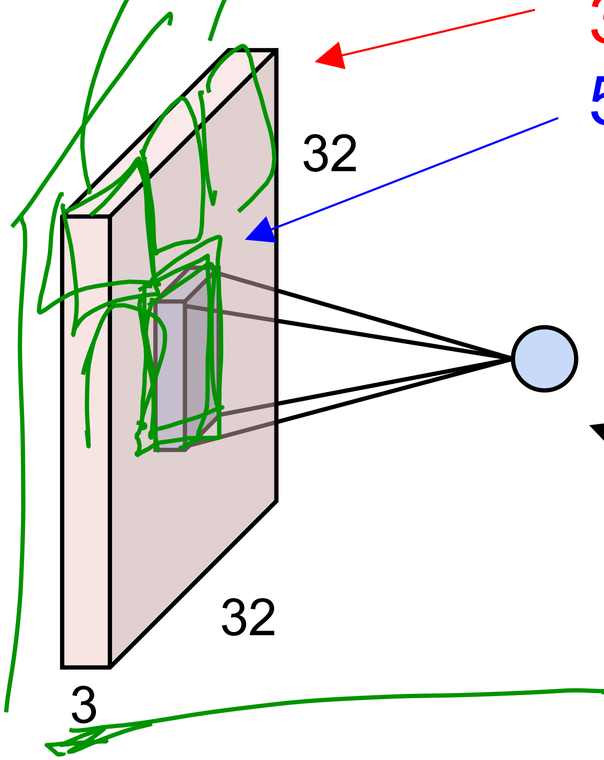
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

$W(:)$

32x32x3 image

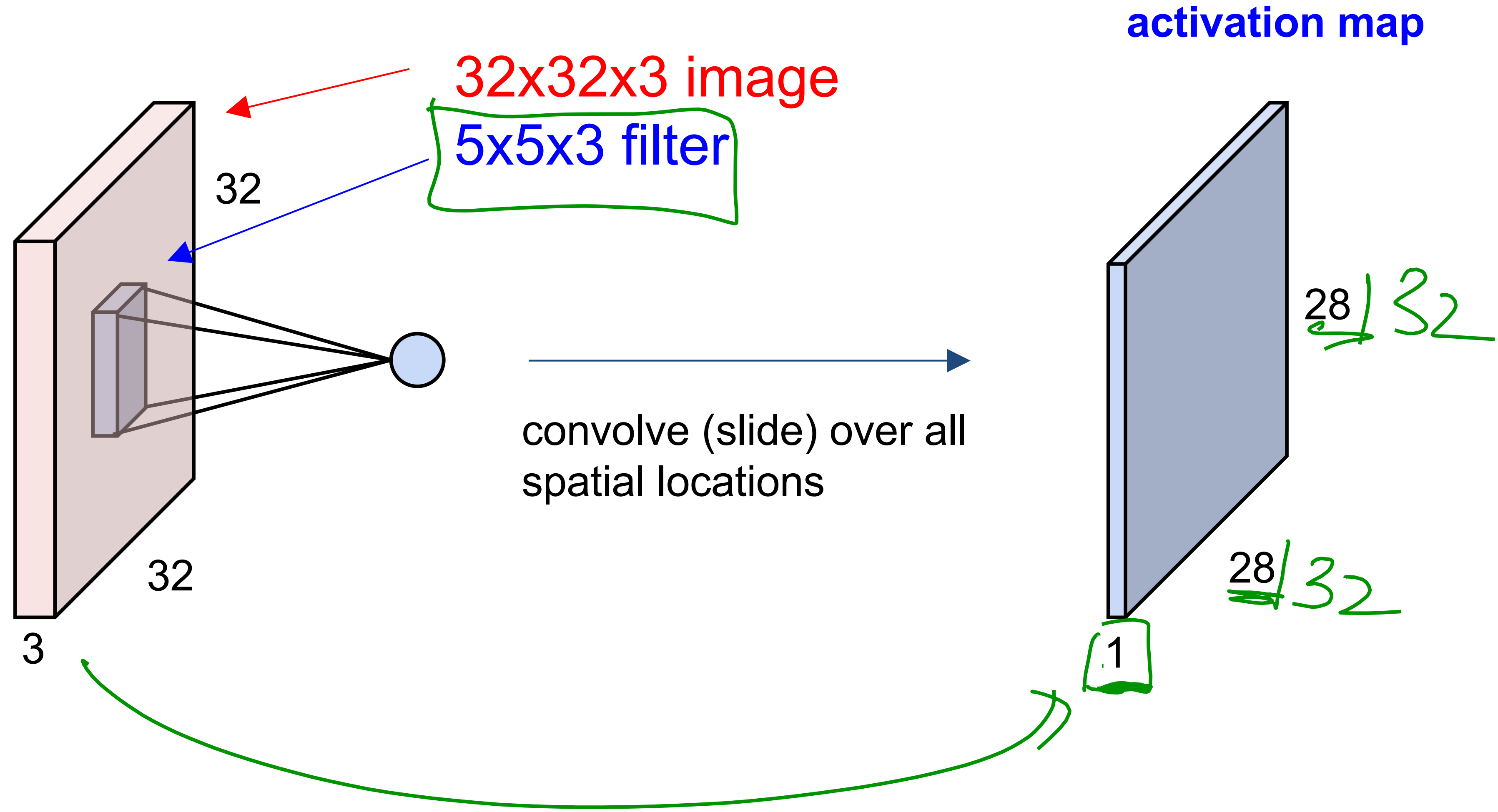
5x5x3 filter w



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)

$$w^T x + b$$

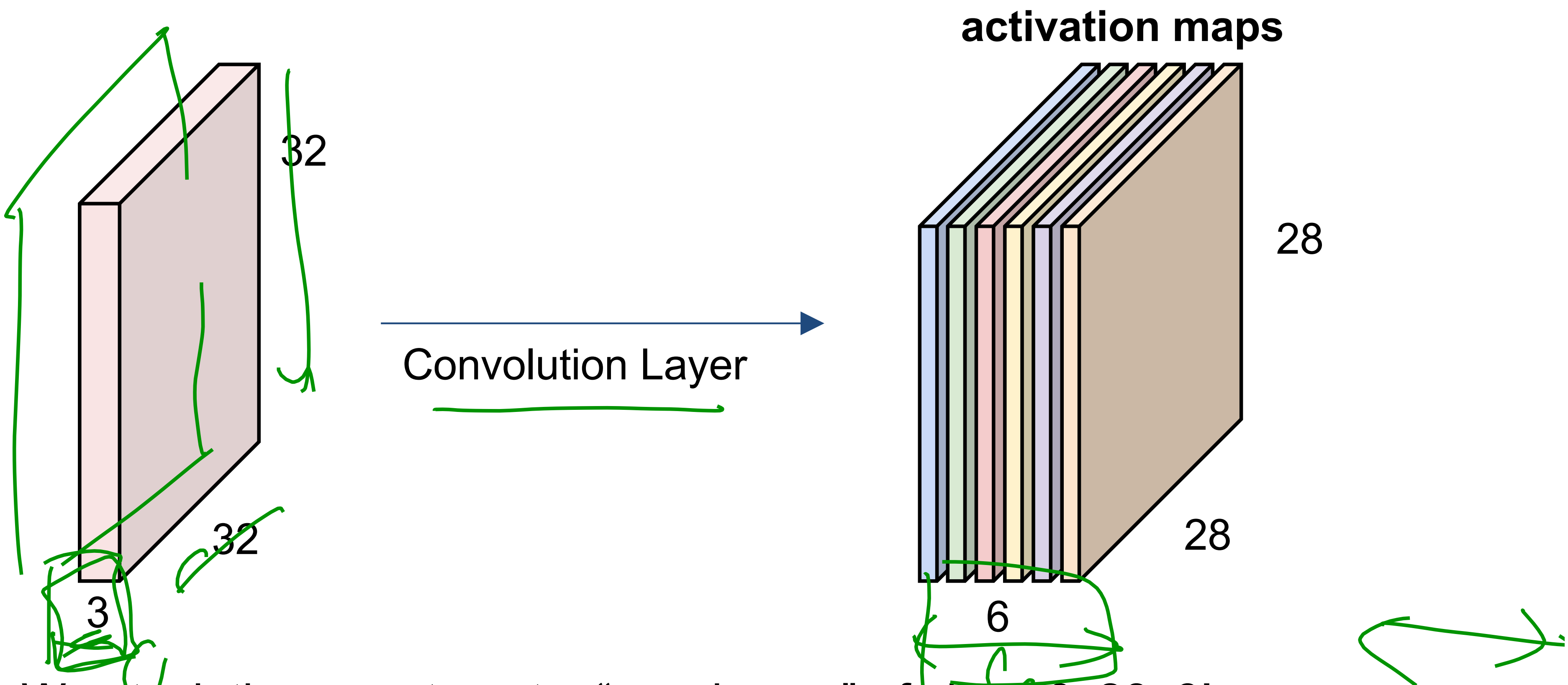
Convolution Layer



Convolution Layer

consider a second, green filter

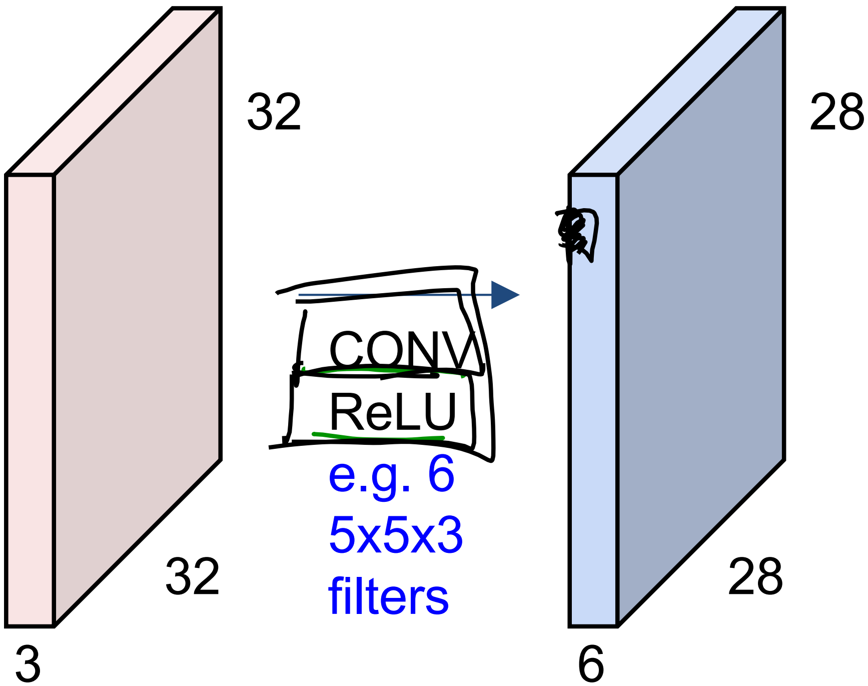
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



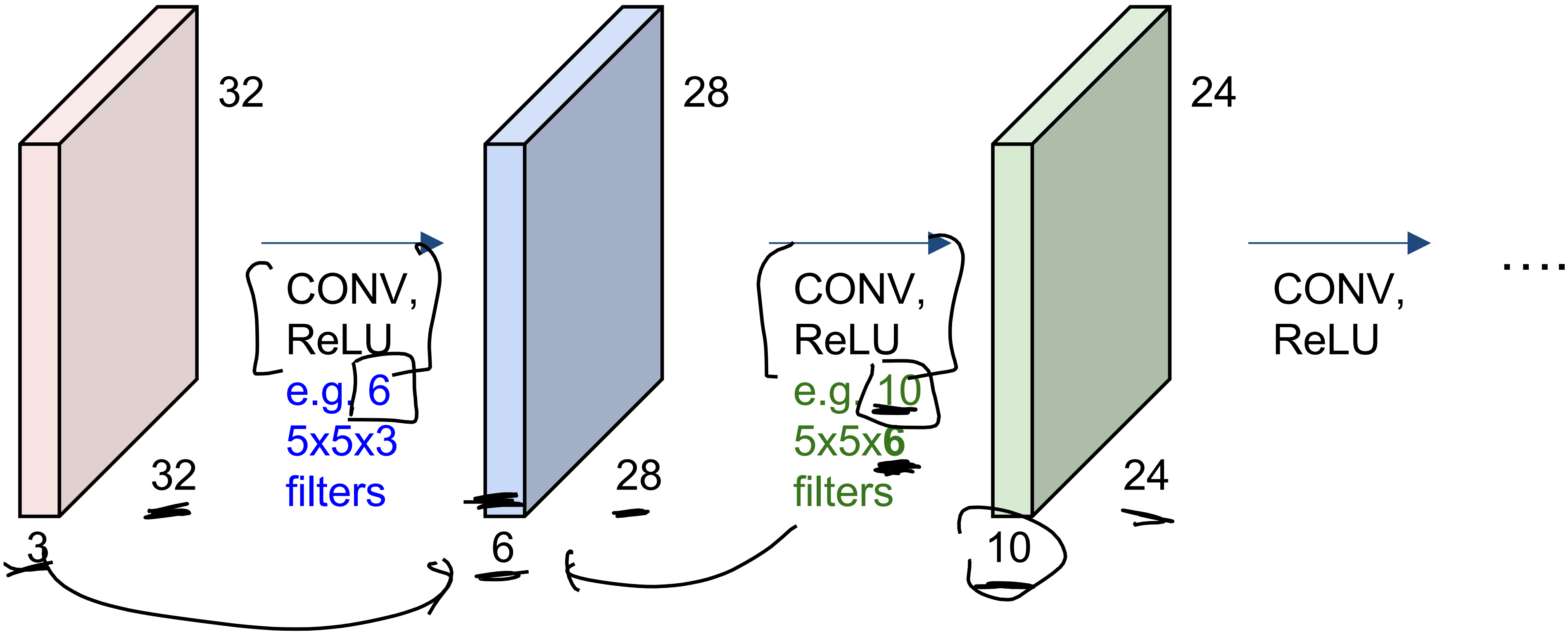
We stack these up to get a "new image" of size 28x28x6!

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

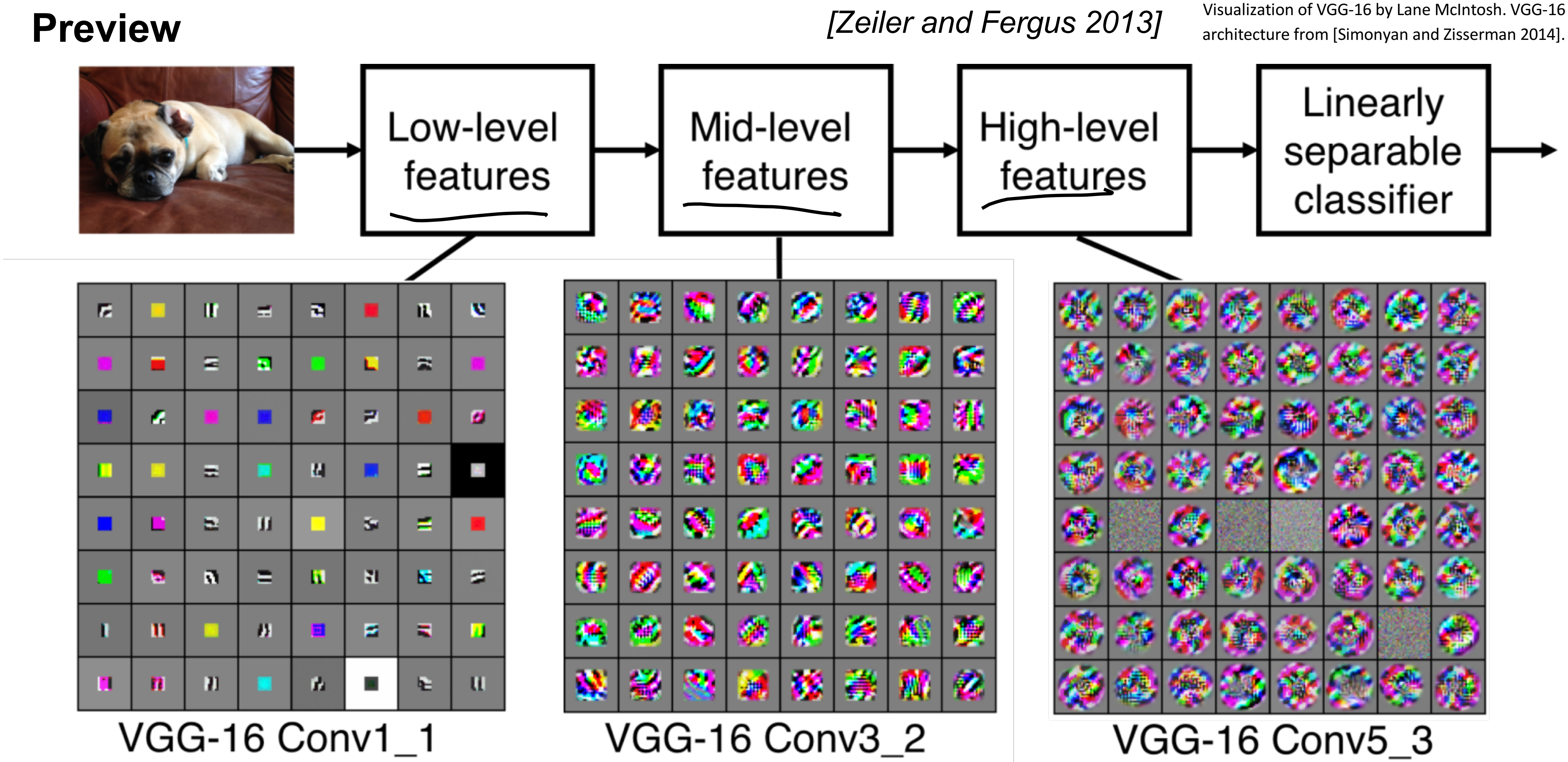
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

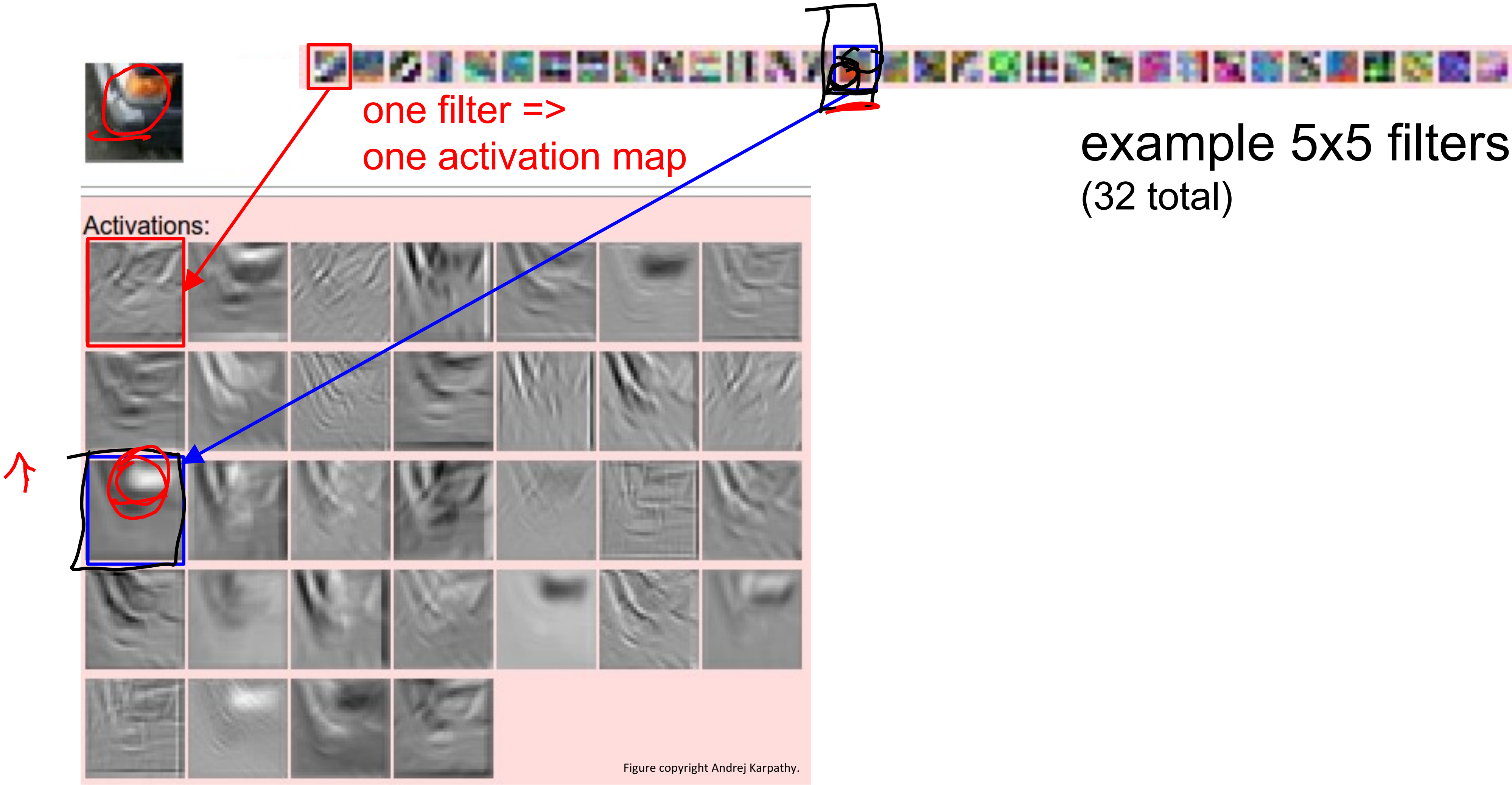


Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



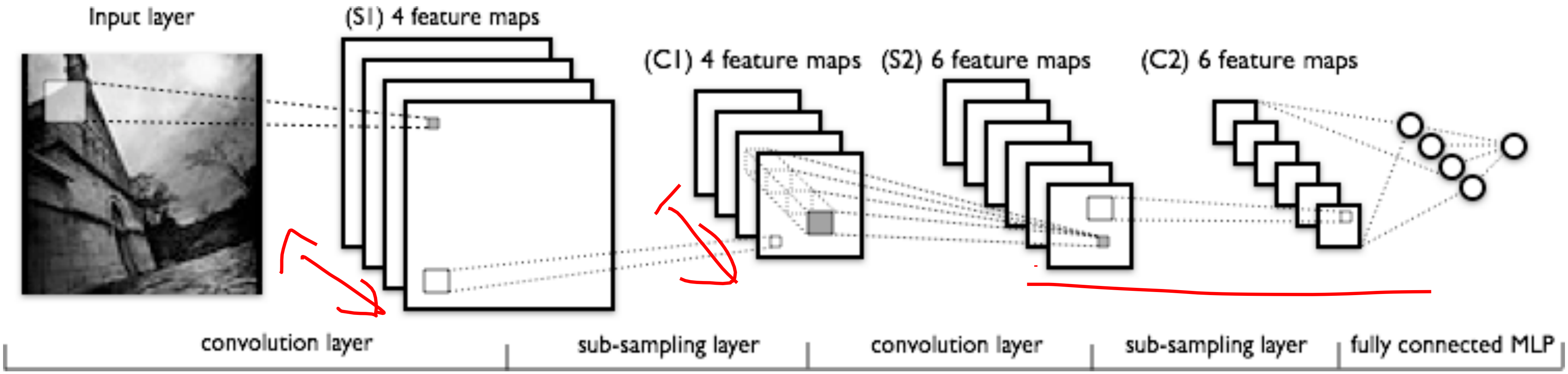
Preview


[Zeiler and Fergus 2013]
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3



example 5x5 filters
(32 total)

Convolutional Neural Networks



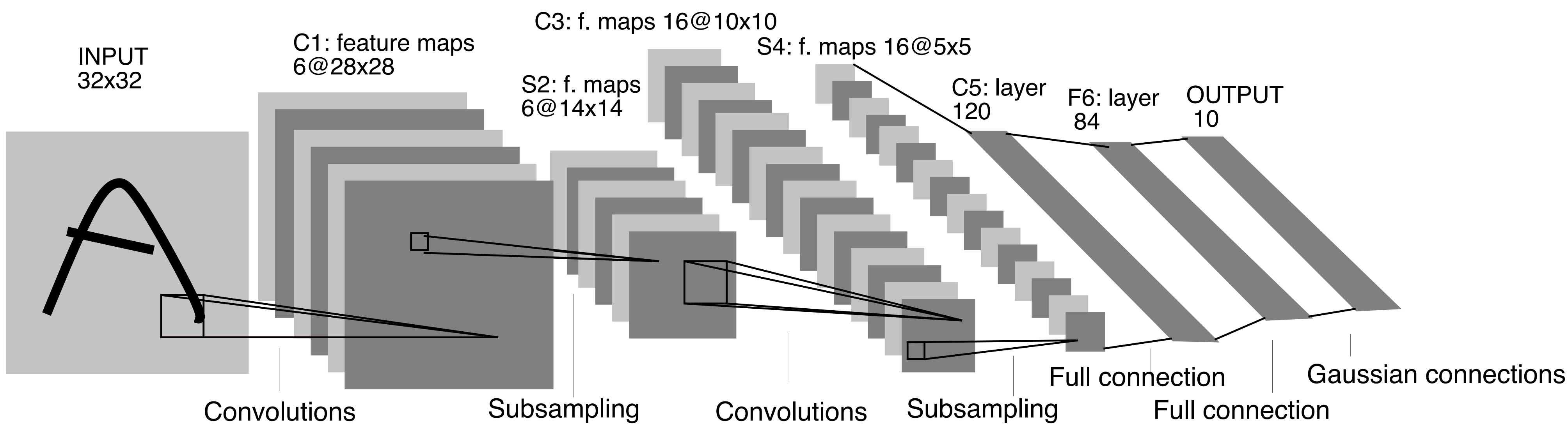
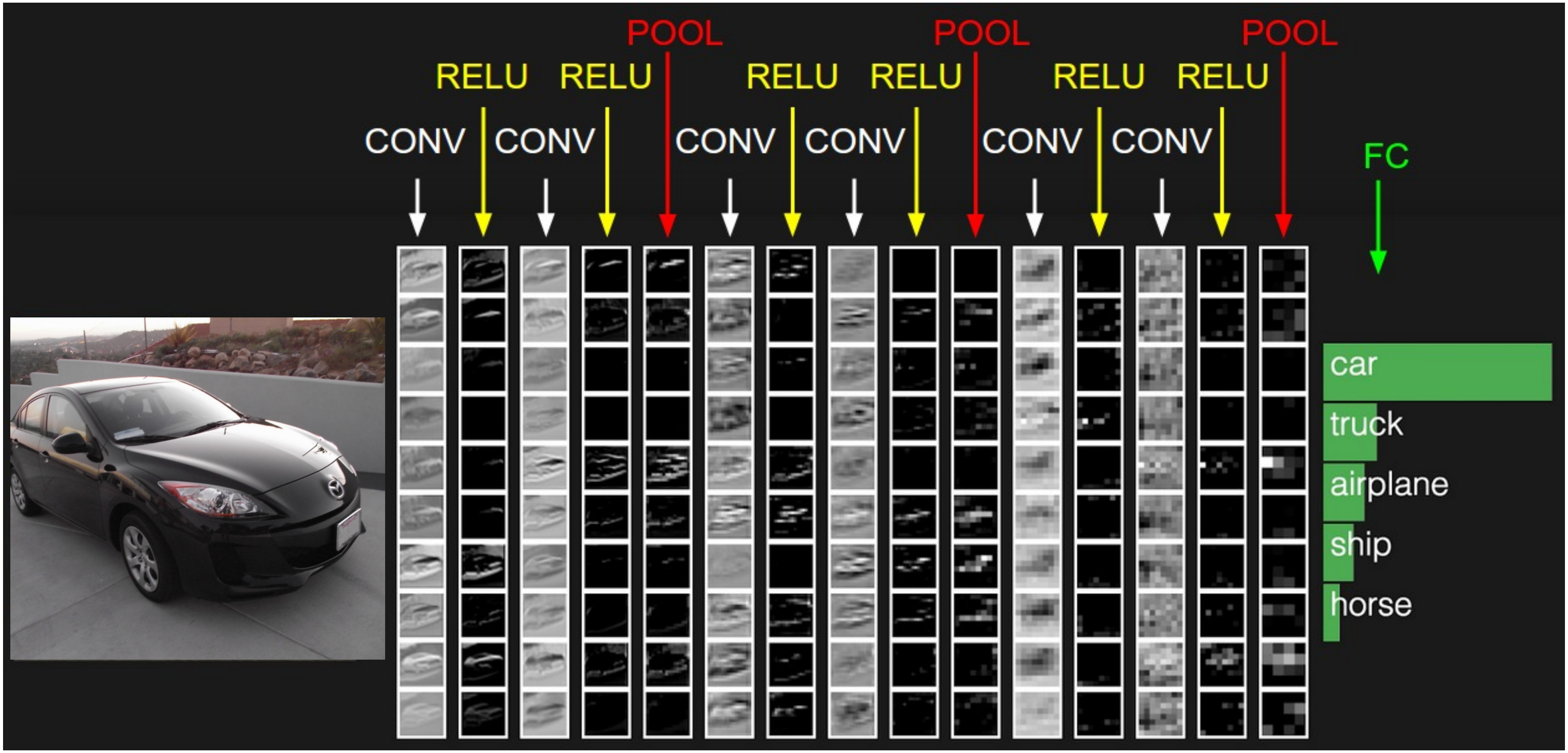
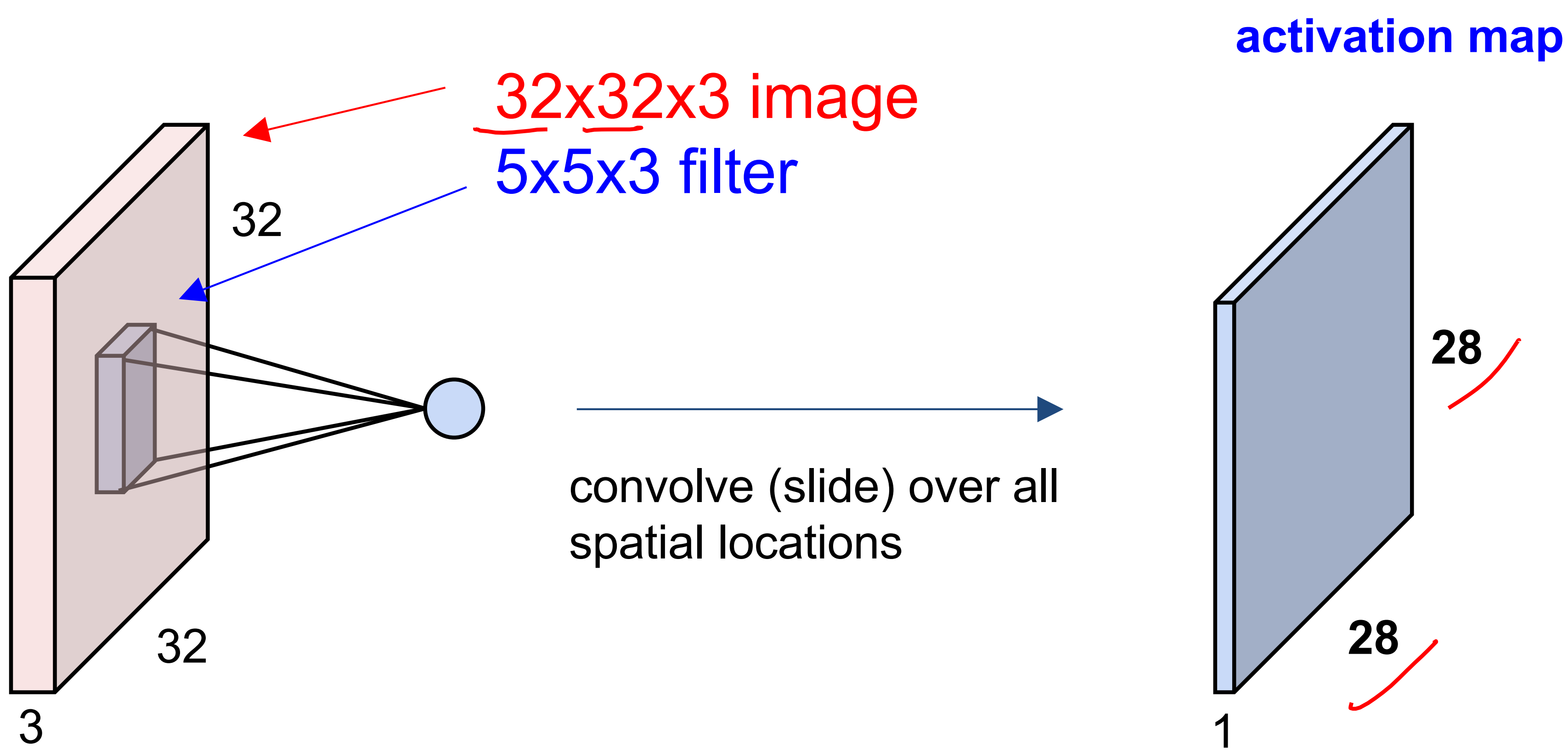


Image Credit: Yann LeCun, Kevin Murphy

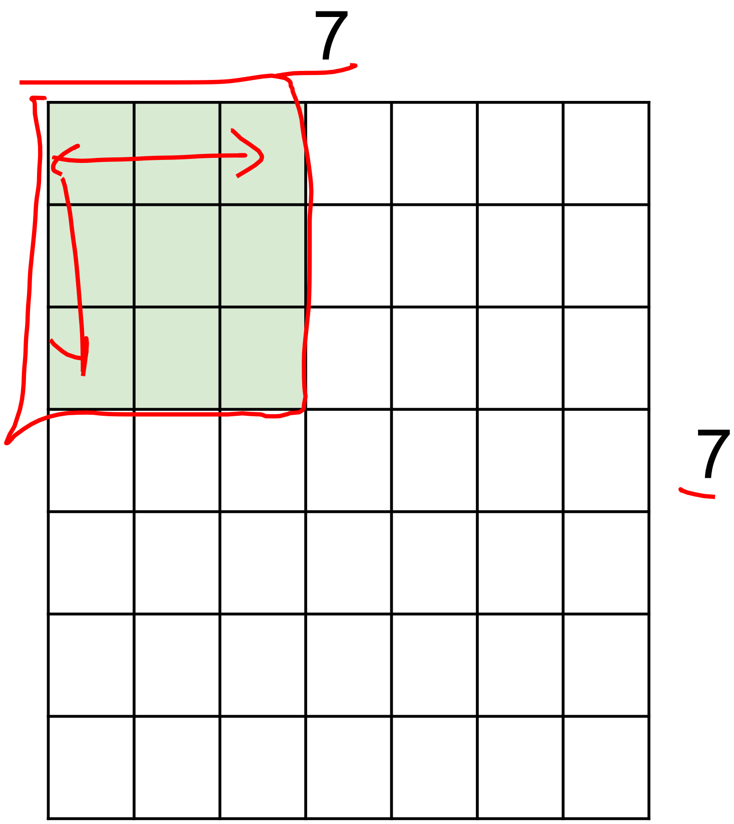
preview:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:

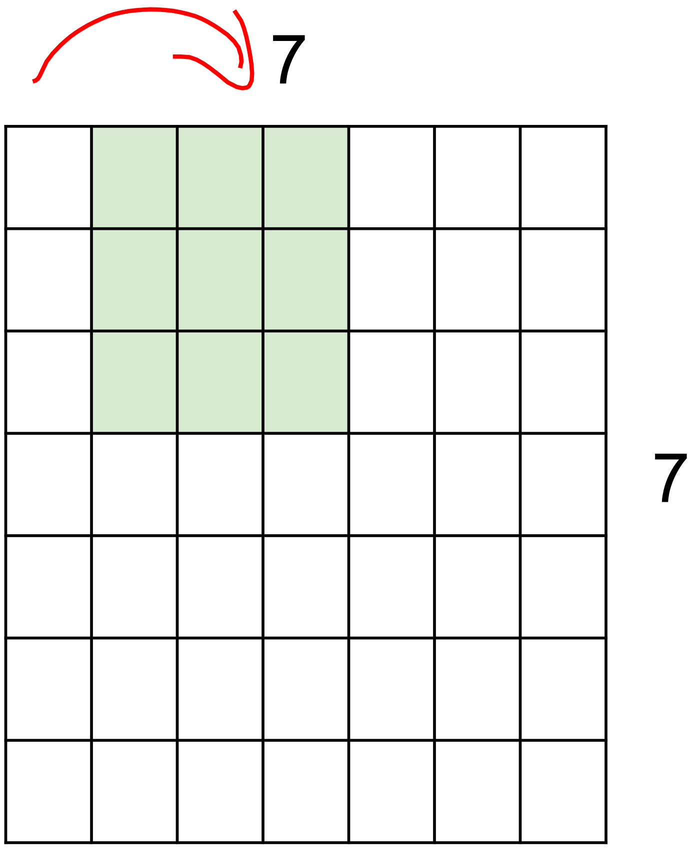


A closer look at spatial dimensions:



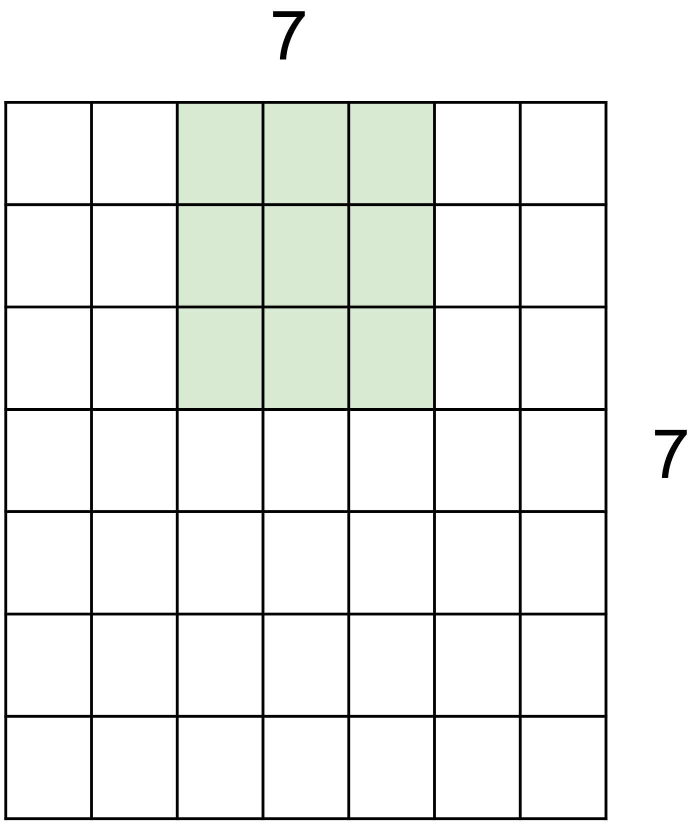
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



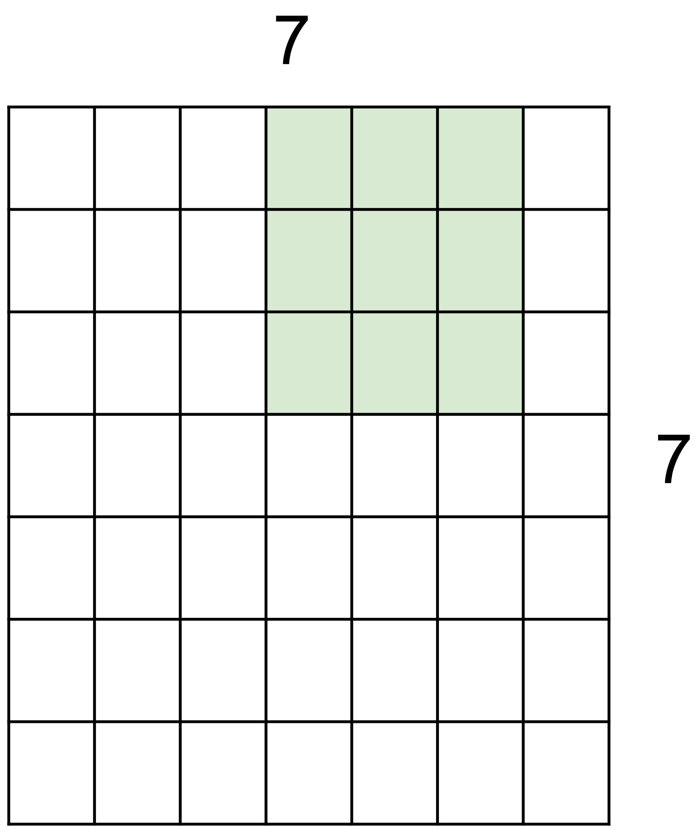
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



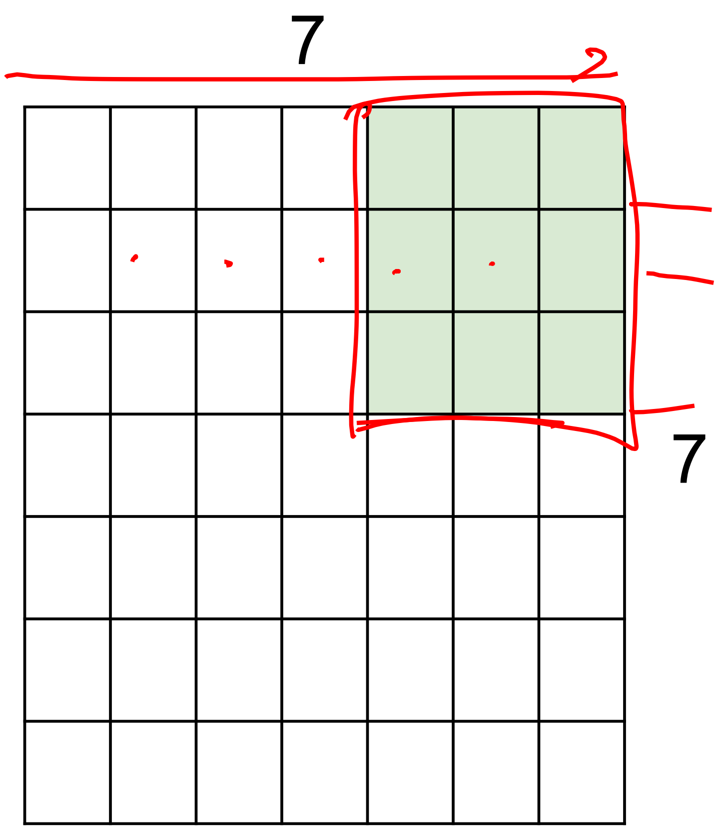
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

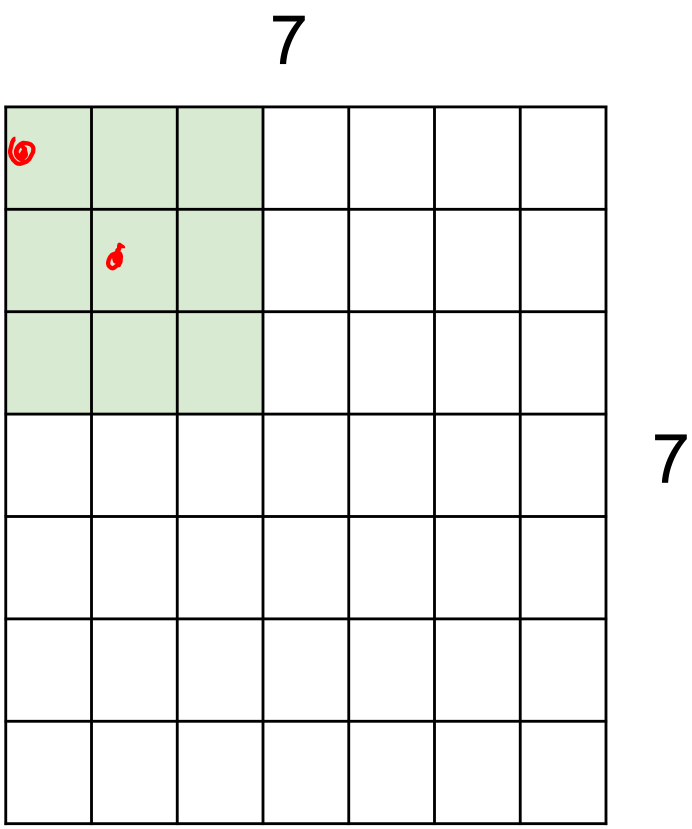
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

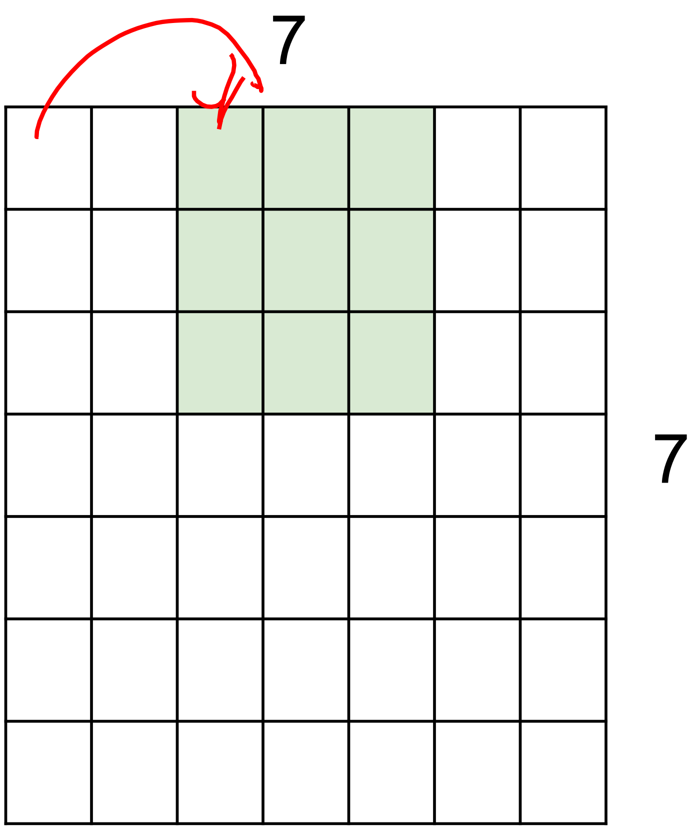
=> 5x5 output

A closer look at spatial dimensions:



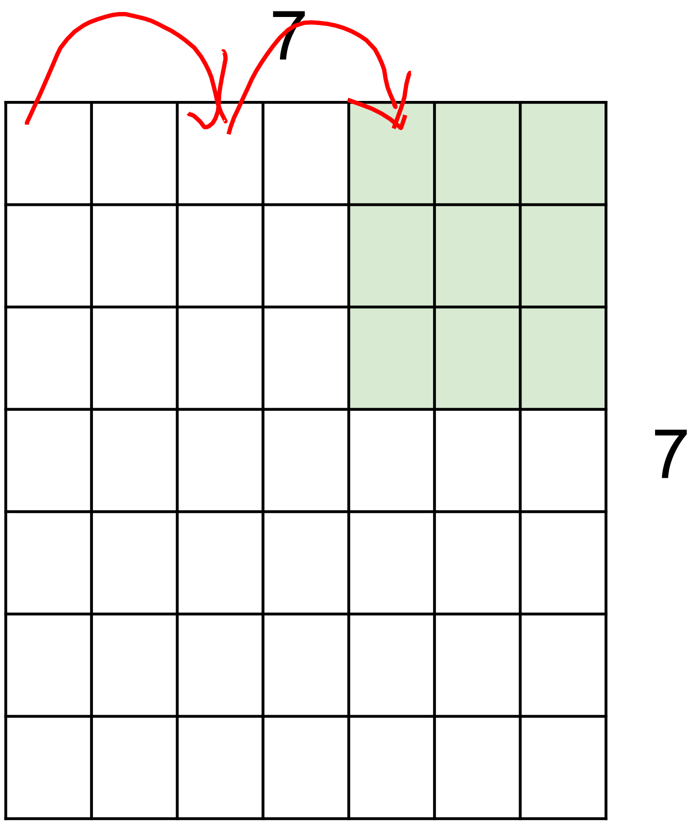
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



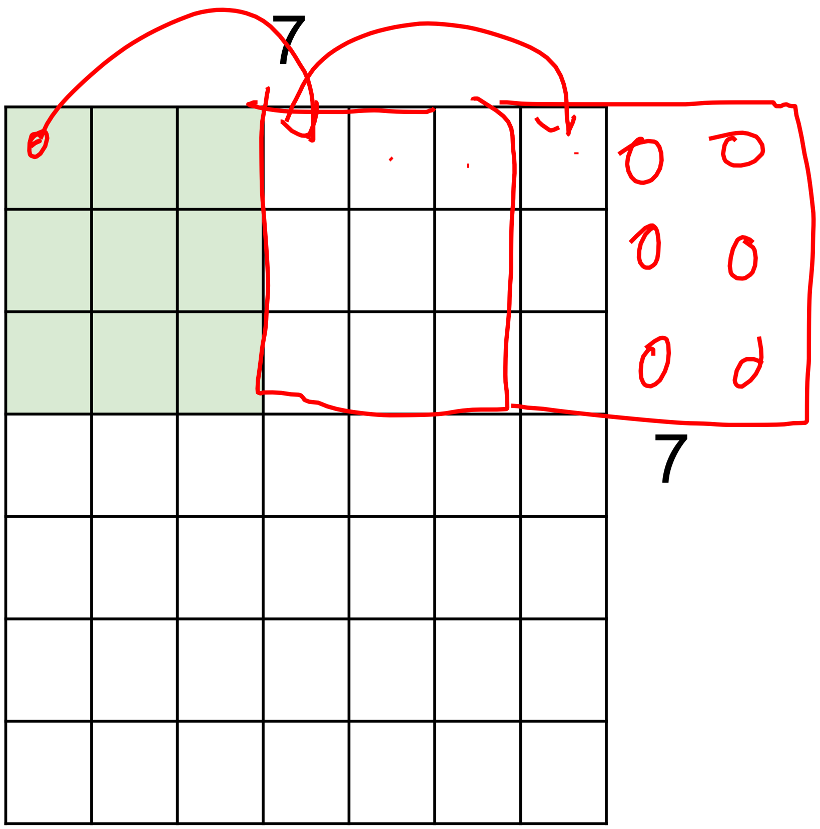
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



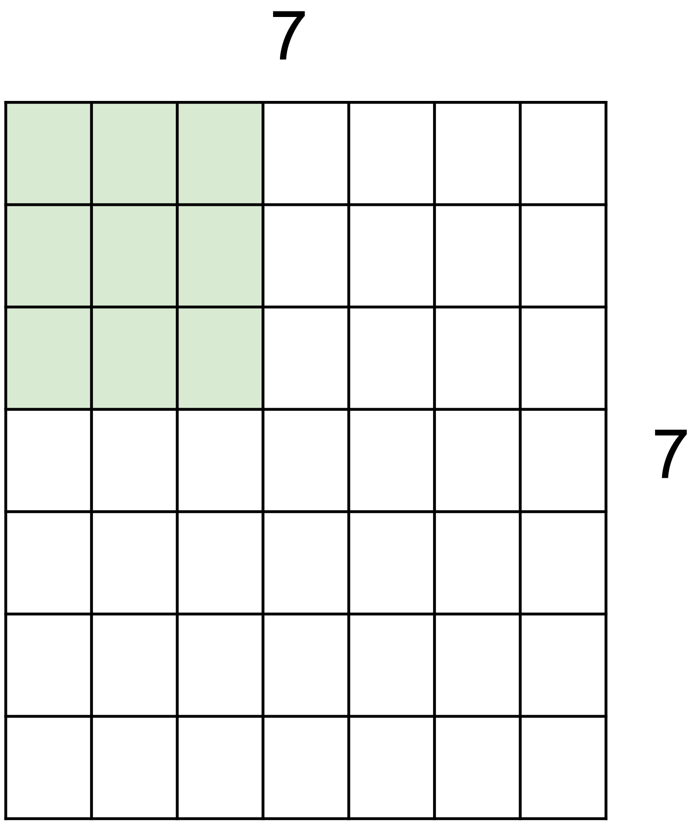
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

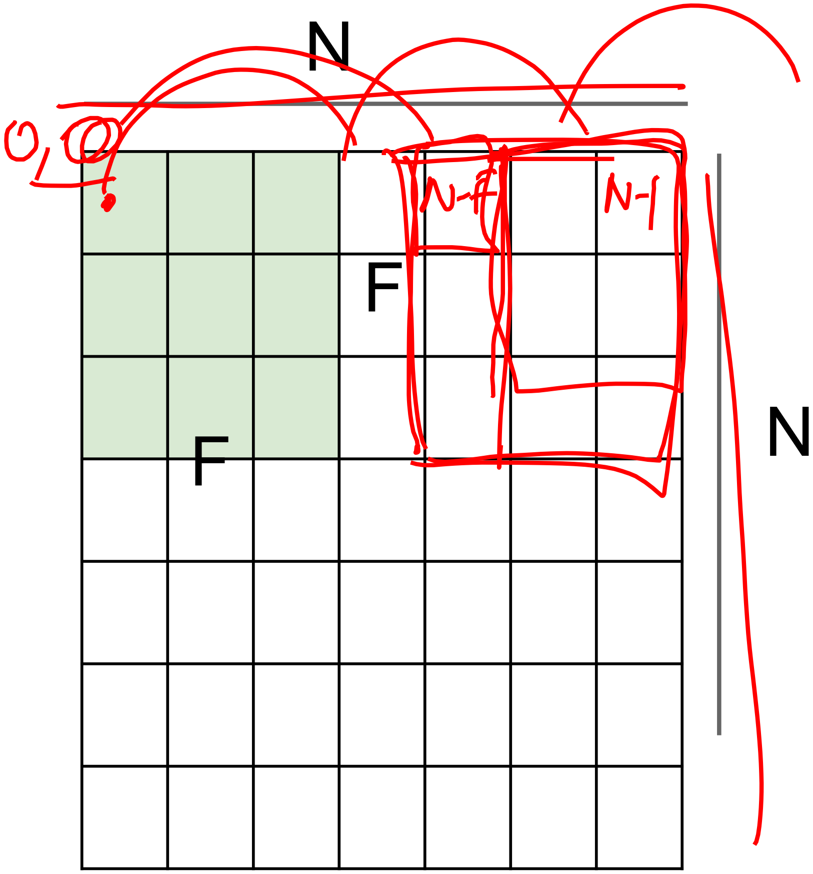
A closer look at spatial dimensions:



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

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

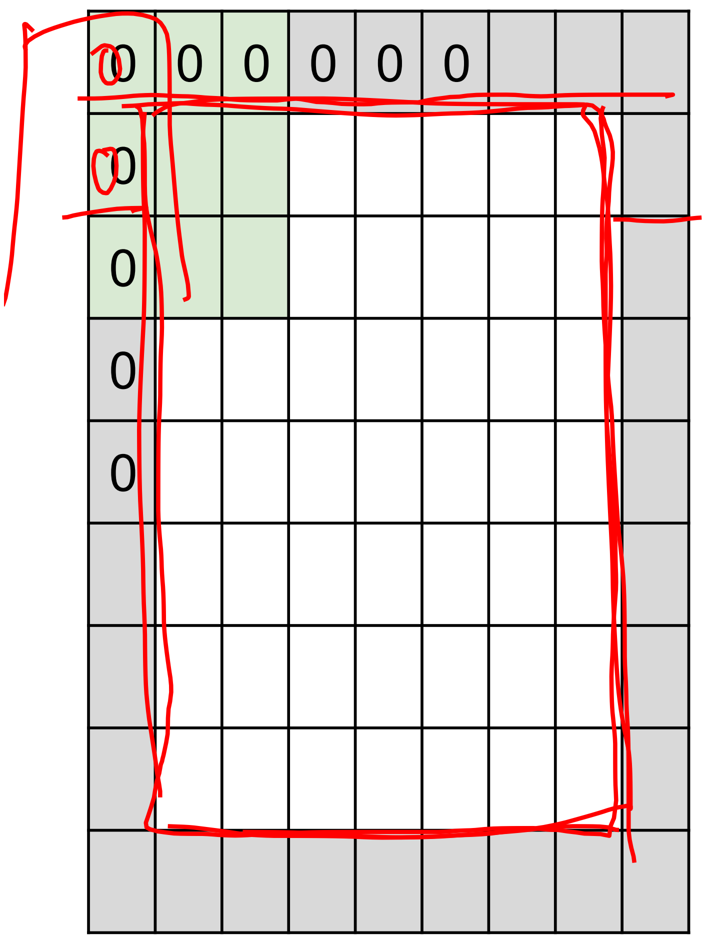
$(N - F) \propto \text{stride}$



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 :\

In practice: Common to zero pad the border



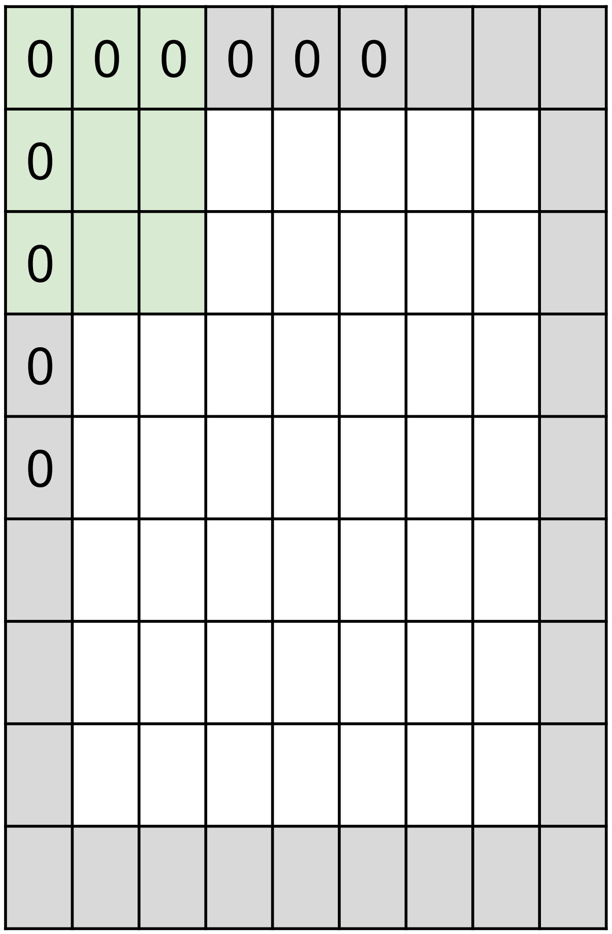
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

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

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

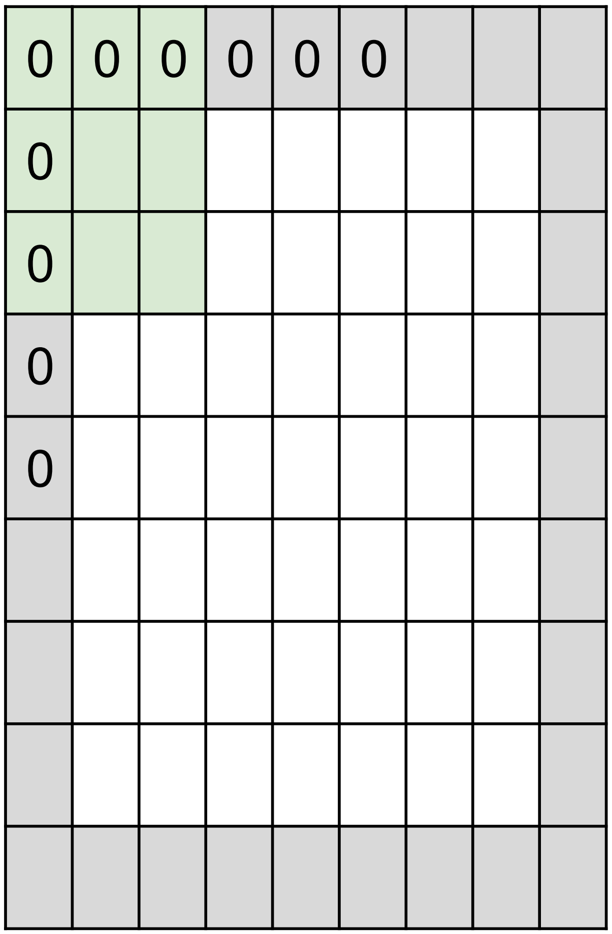
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

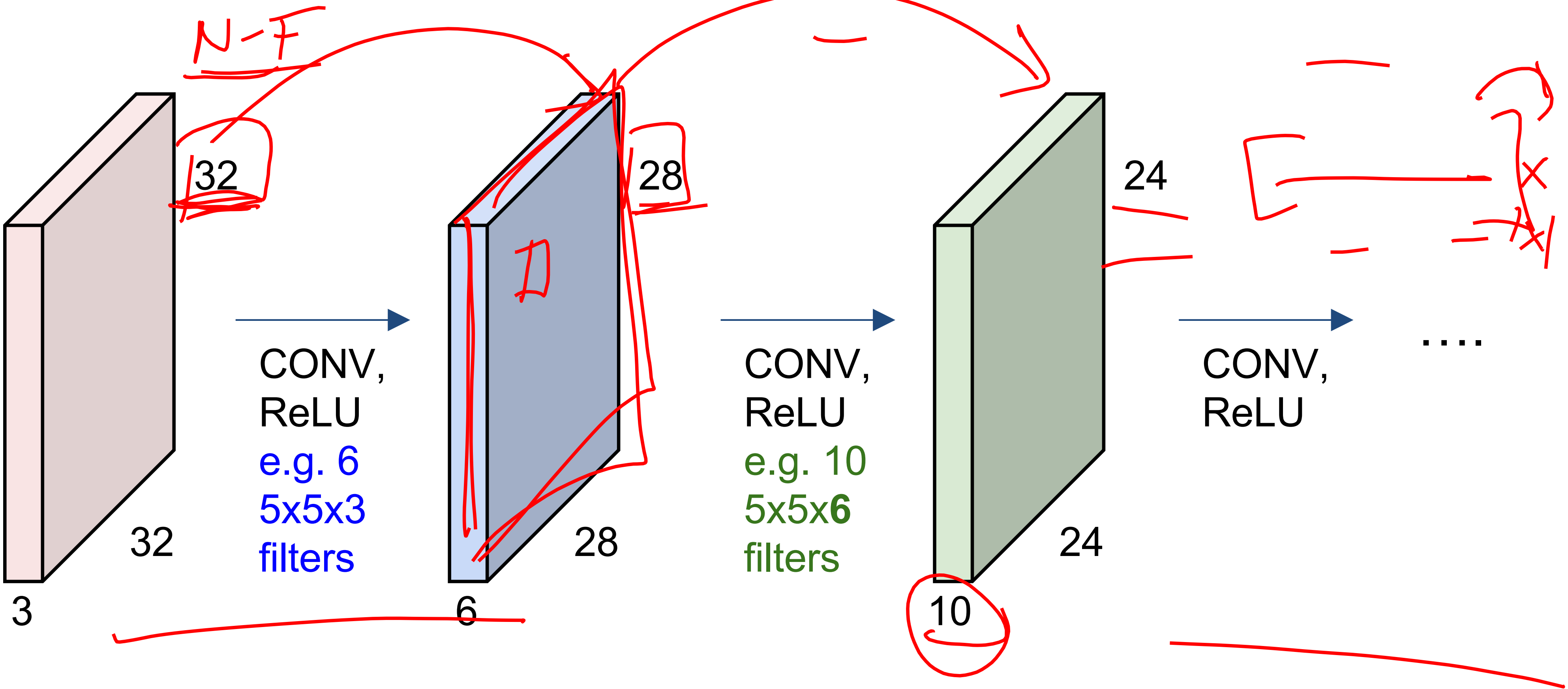


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Remember back to…

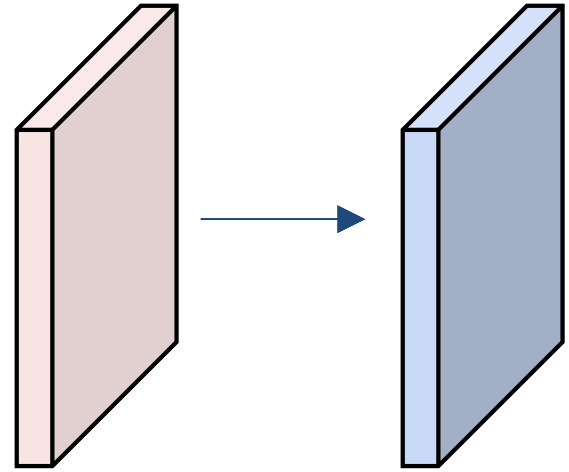
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



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

$$\frac{N-F}{St} + 1$$

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

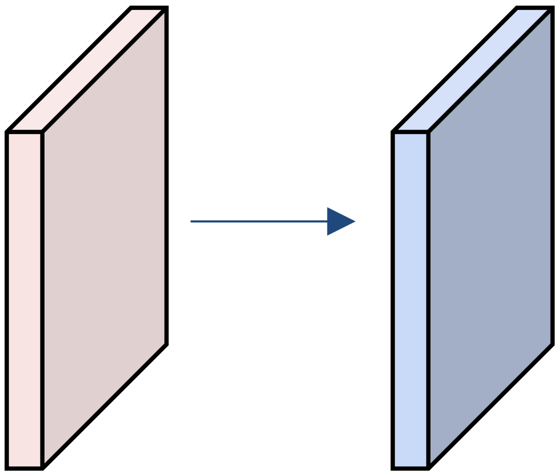
Output volume size: ?

$K_1 \times K_1 \times C_2$

$32 \quad 32 \times 10$

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

Examples time:



Input volume: **32x32x3**

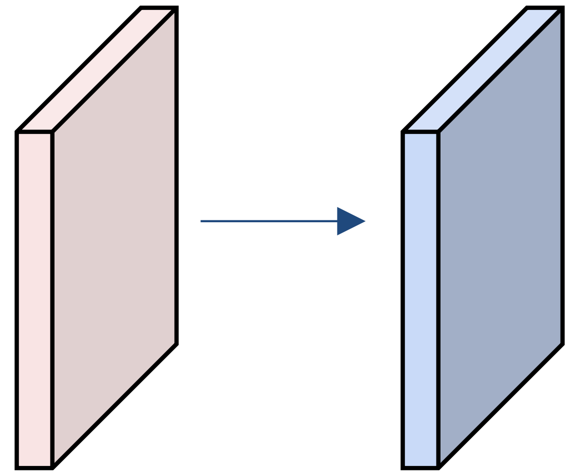
10 5x5 filters with stride 1, pad 2

Output volume size:

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

32x32x10

Examples time:



Input volume: **32x32x3**

$\boxed{\text{10 5x5 filters}}$ with stride 1, pad 2

Number of parameters in this layer?

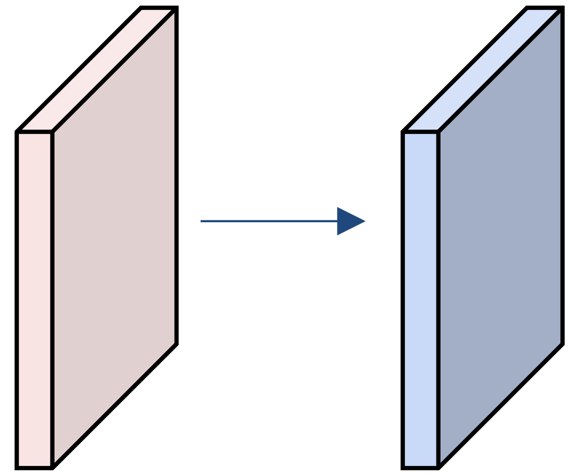
750

$5 \times 5 \times 10 \times 3 \qquad 10$

$[k_1 \times k_2 \times c_1 + 1] \times c_2 = 760$

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

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:
  - $\boxed{W_2 = (W_1 - F + 2P)/S + 1}$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $\boxed{D_2 = K}$
- 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)

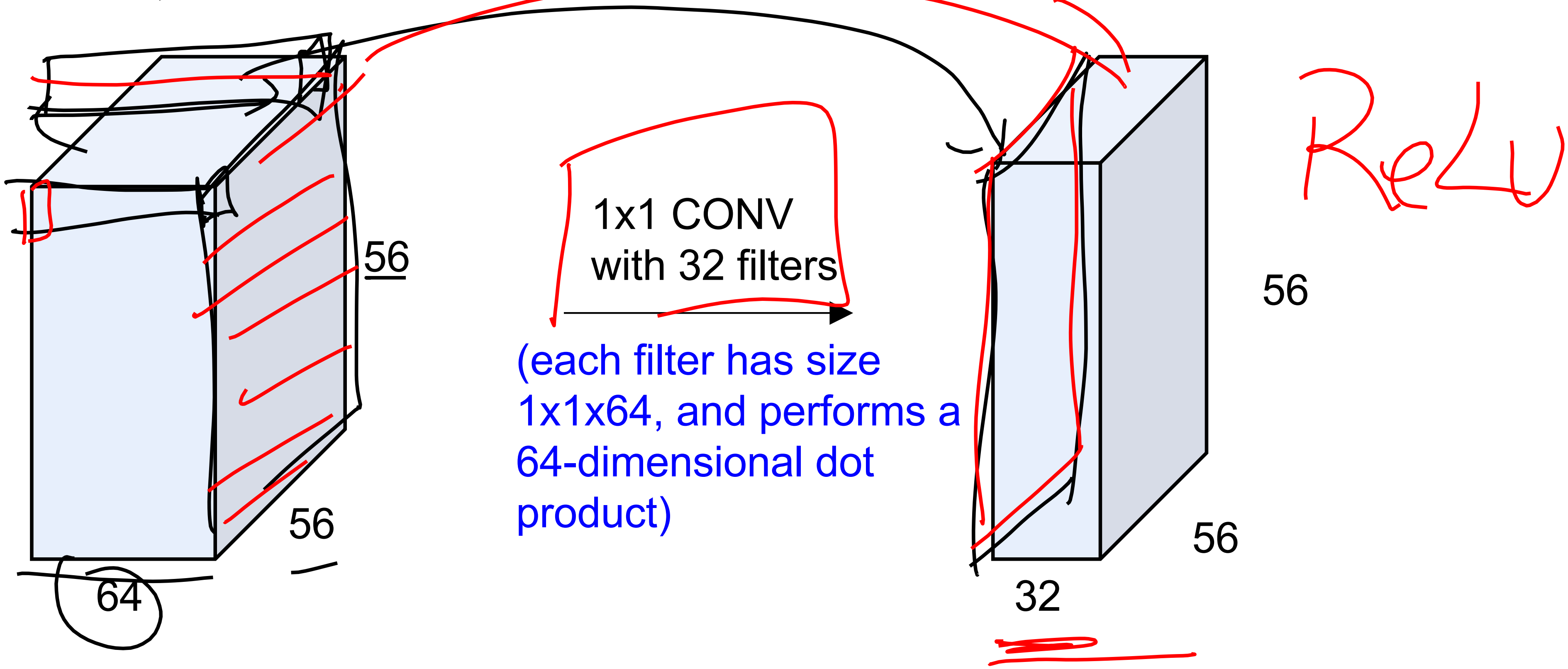
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

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.

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

(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in Torch

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.

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)
```

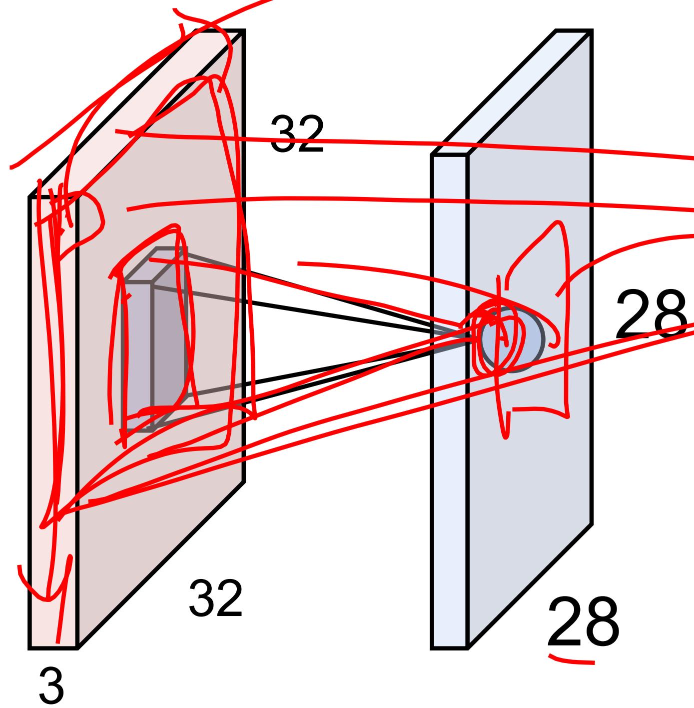
Example: CONV layer in Caffe

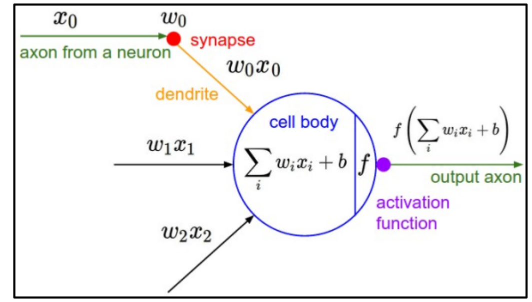
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 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.

The brain/neuron view of CONV Layer





An activation map is a 28x28 sheet of neuron outputs:

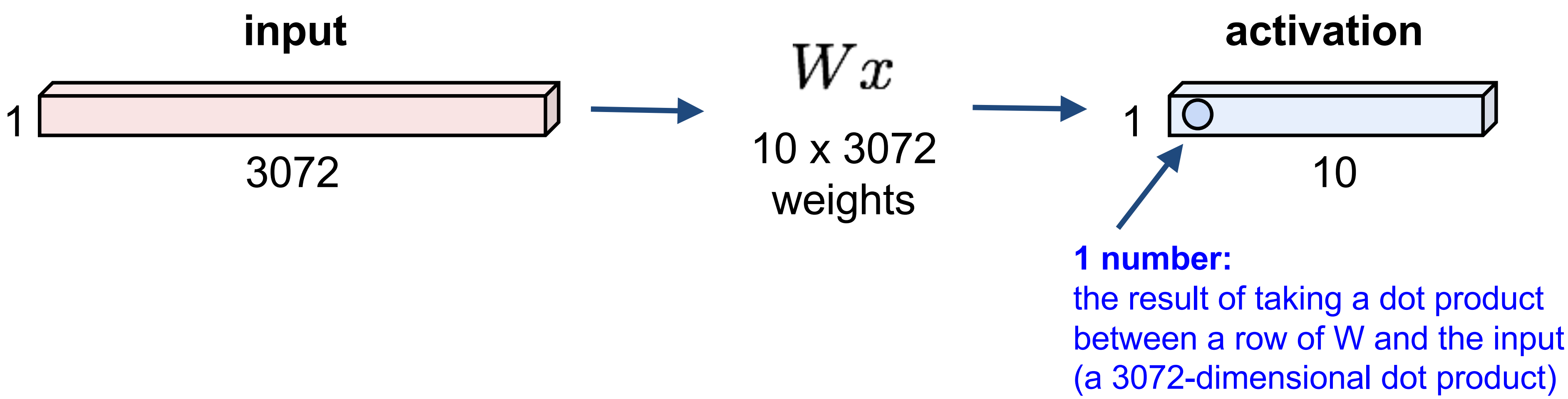
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

Reminder: Fully Connected Layer

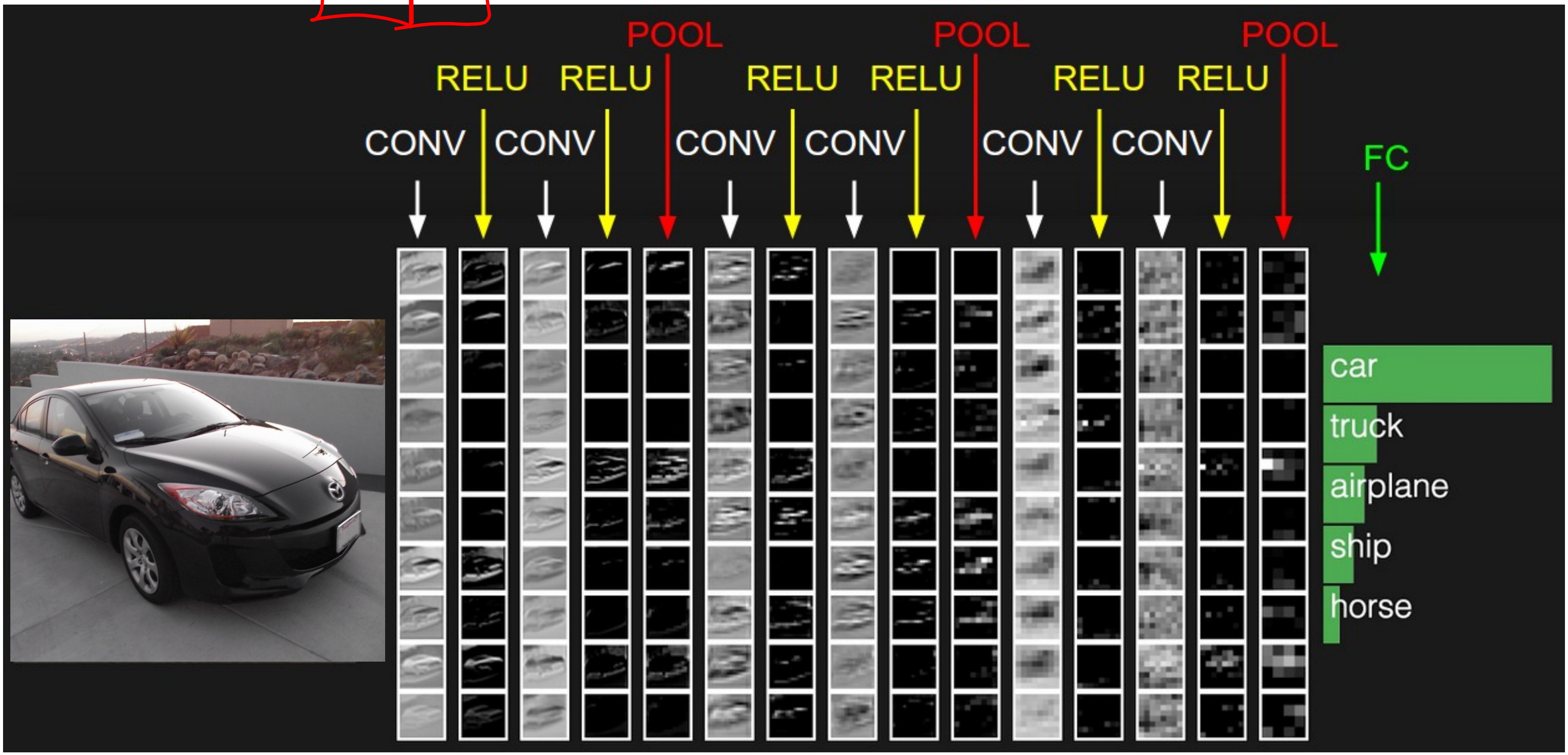
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



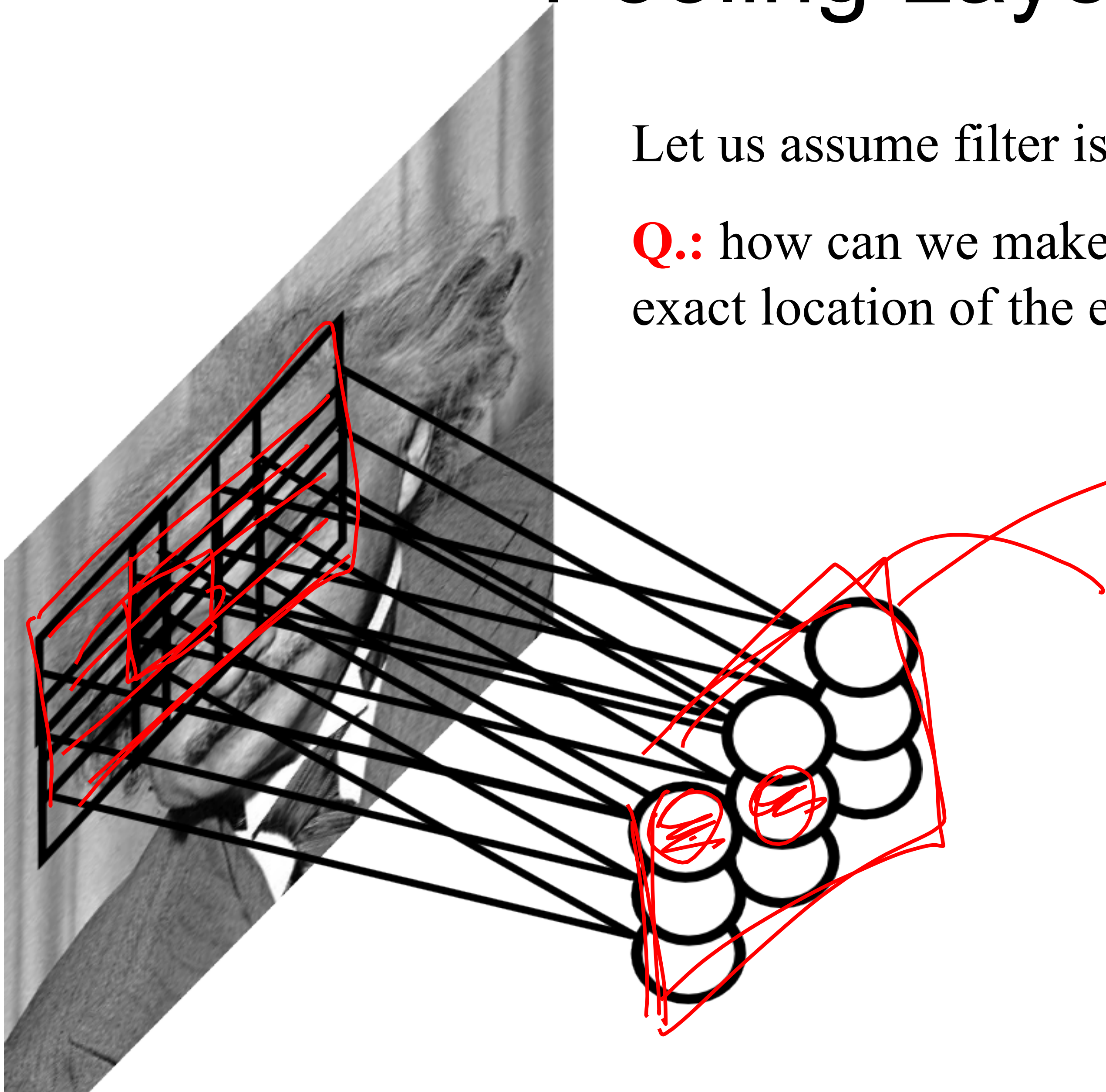
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

two more layers to go: POOL/FC



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

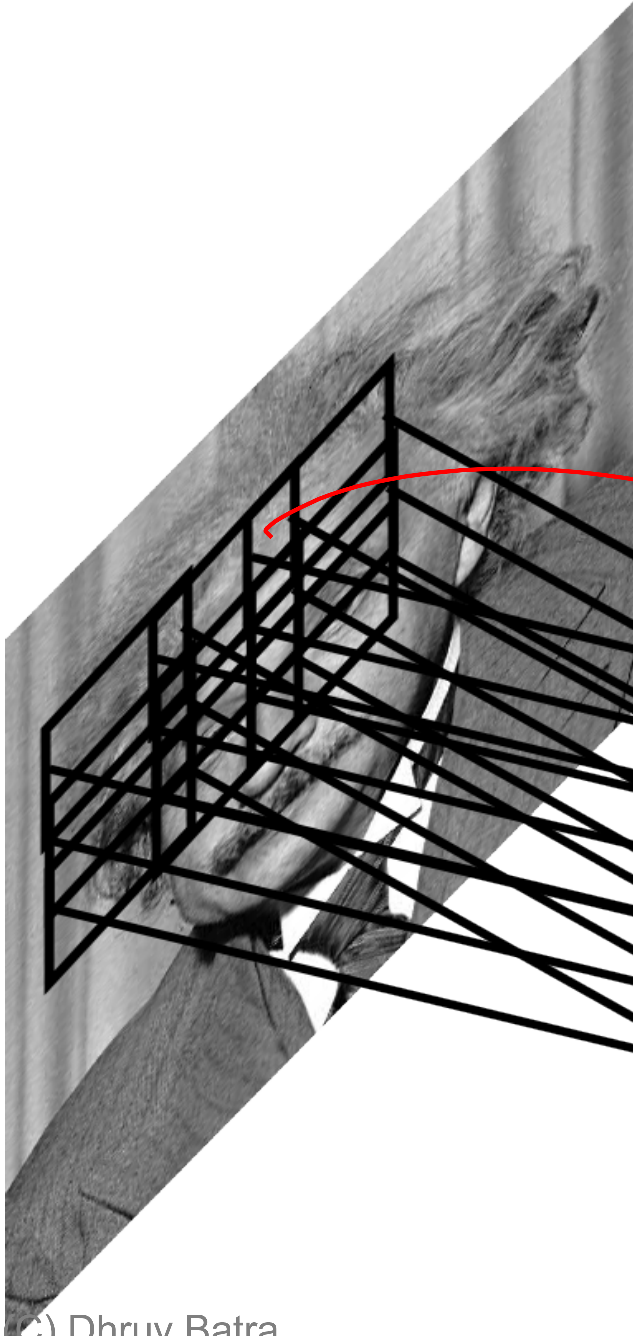
Pooling Layer



Let us assume filter is an “eye” detector.

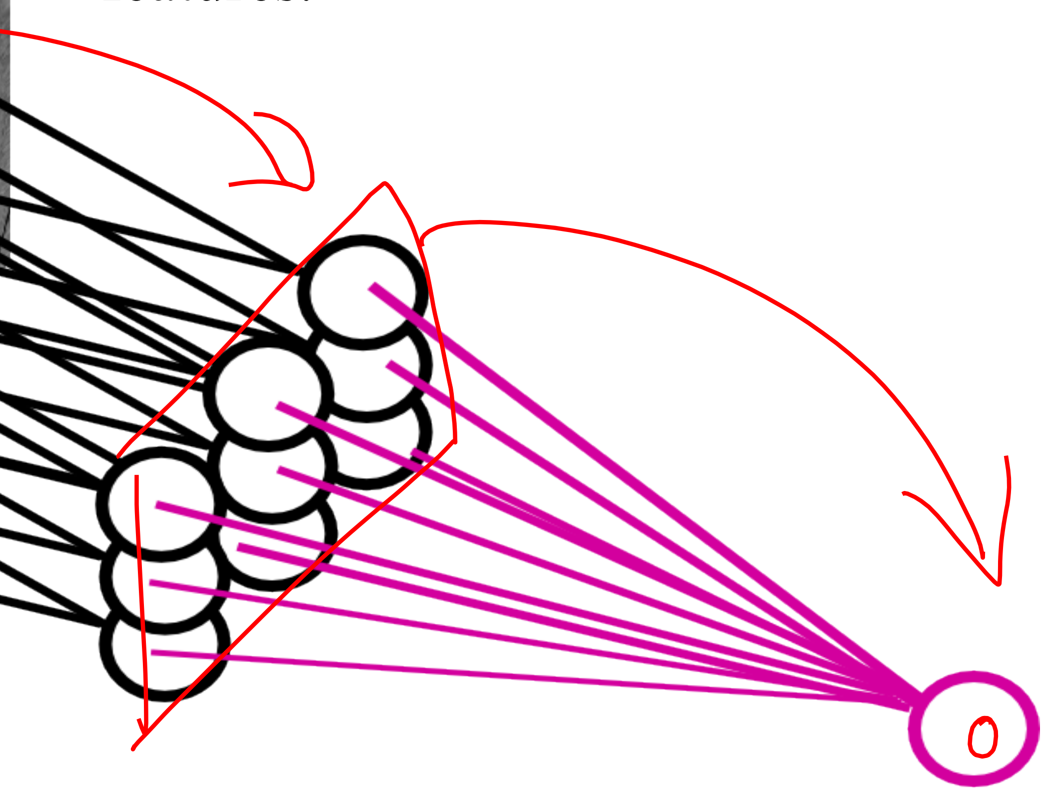
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



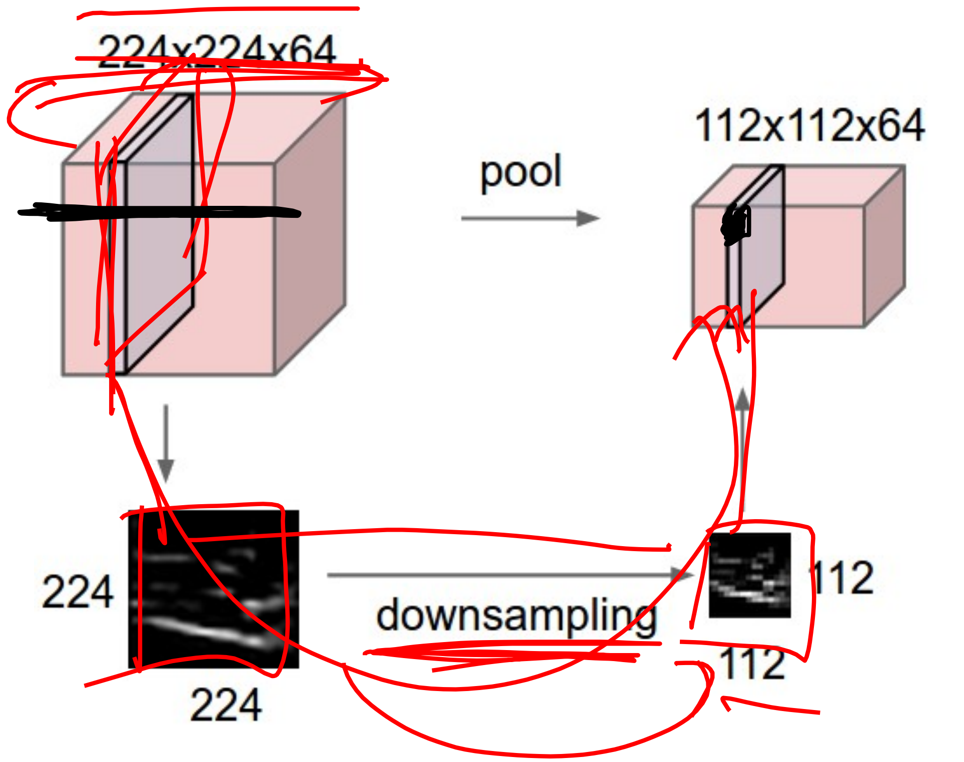
By "pooling" (e.g., taking max) filter

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



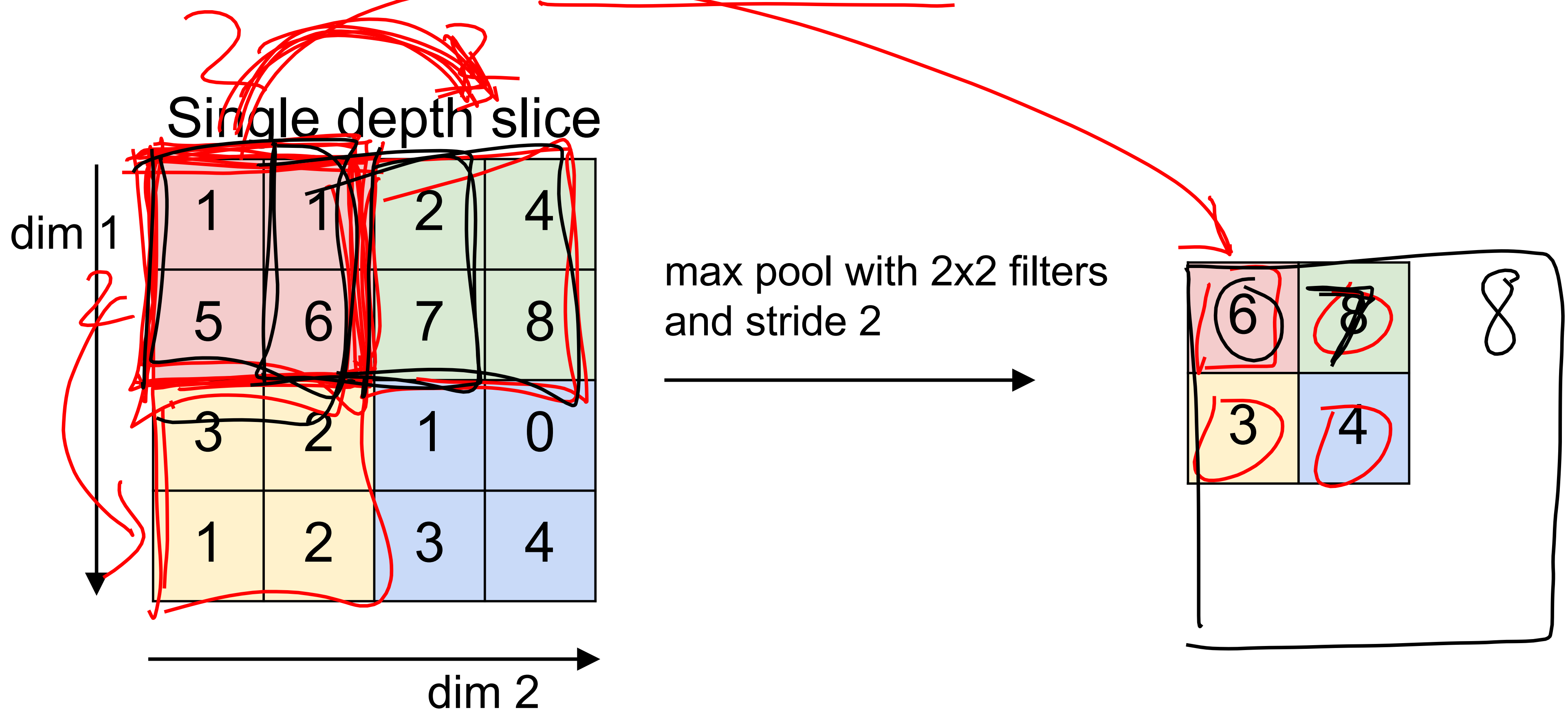
Pooling layer

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



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

MAX POOLING



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
8

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:

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

L2-pooling:

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

L2-pooling over features:

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

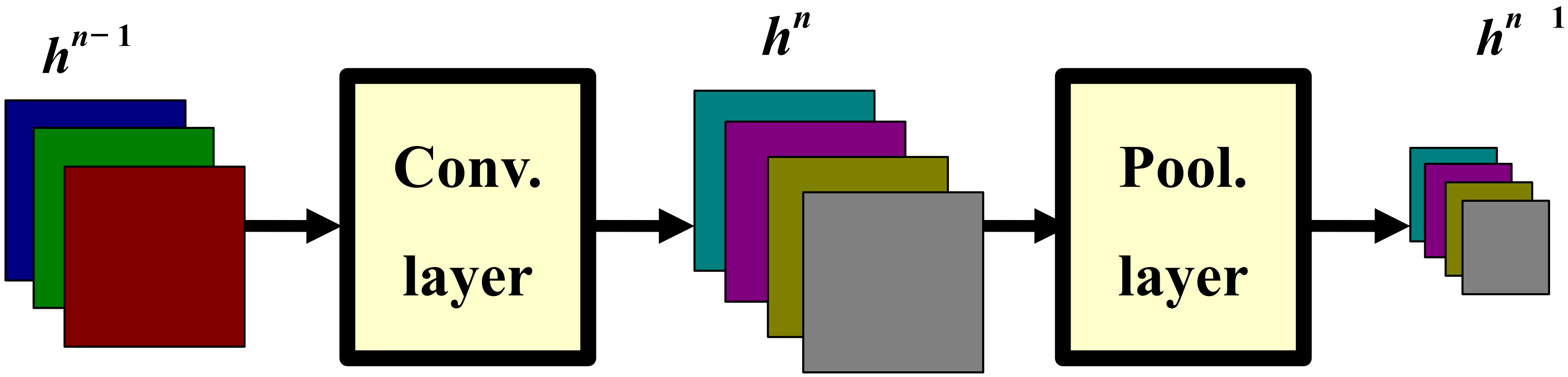
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since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

Common settings:

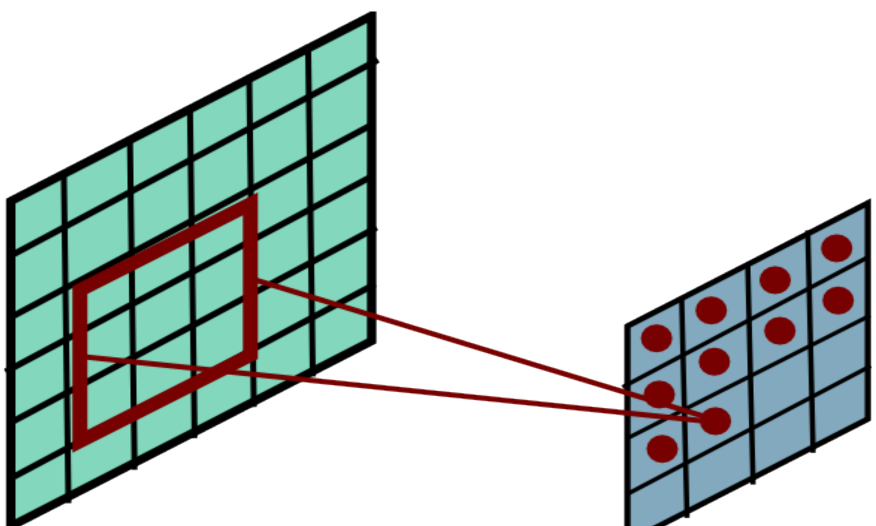
F = 2, S = 2
F = 3, S = 2

- 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

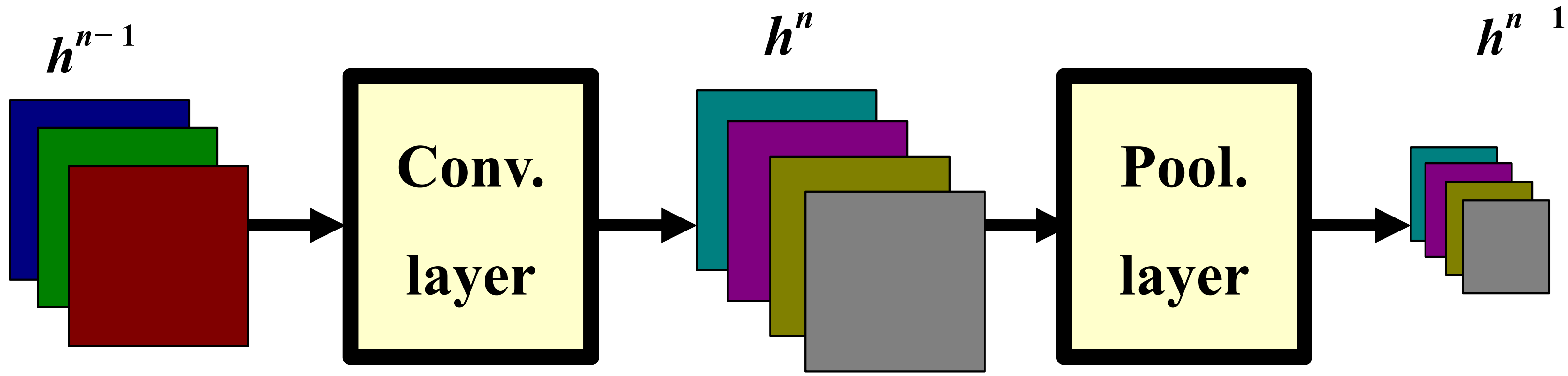
Pooling Layer: Receptive Field Size



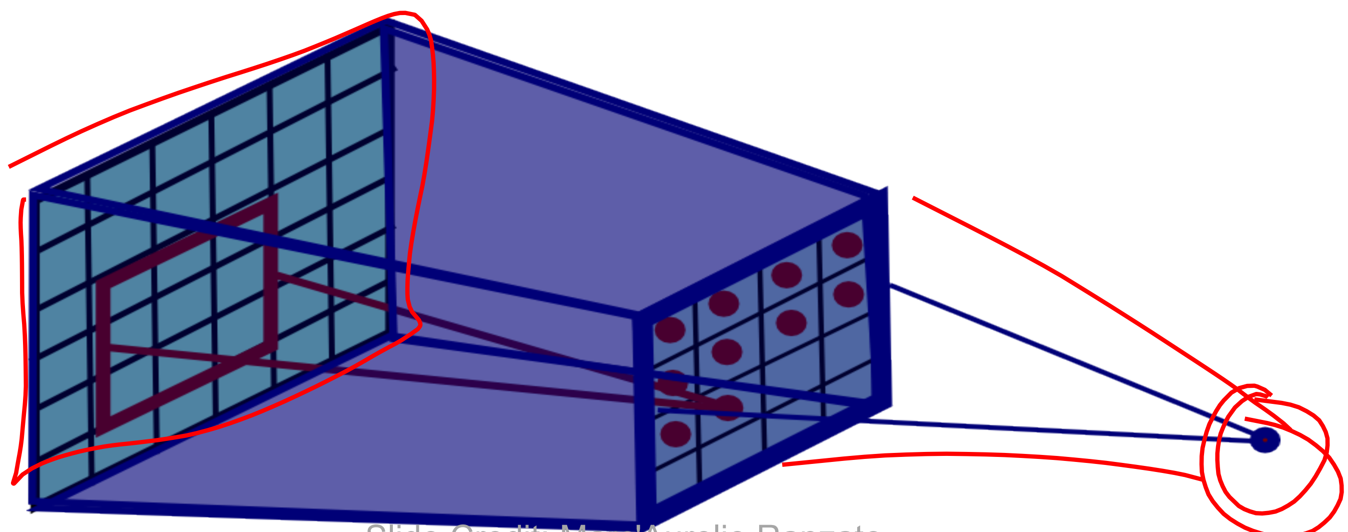
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Pooling Layer: Receptive Field Size

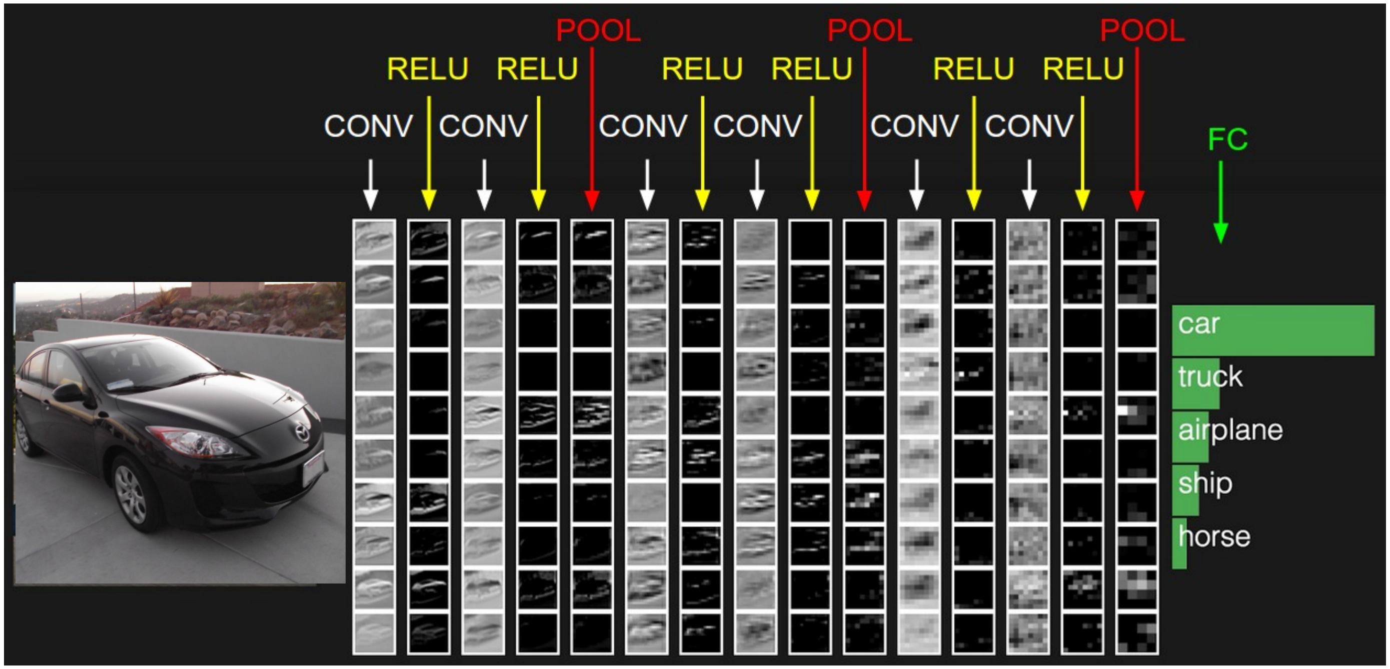


If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



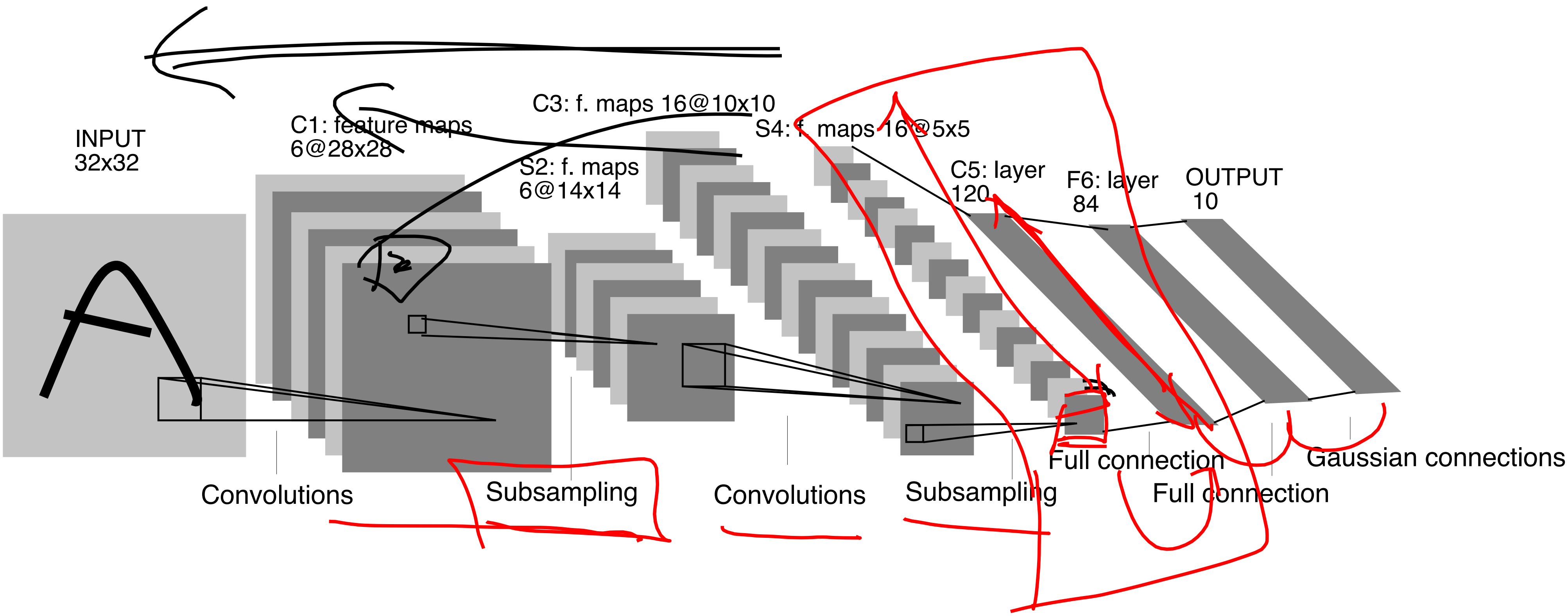
Fully Connected Layer (FC layer)

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



Convolutional Nets

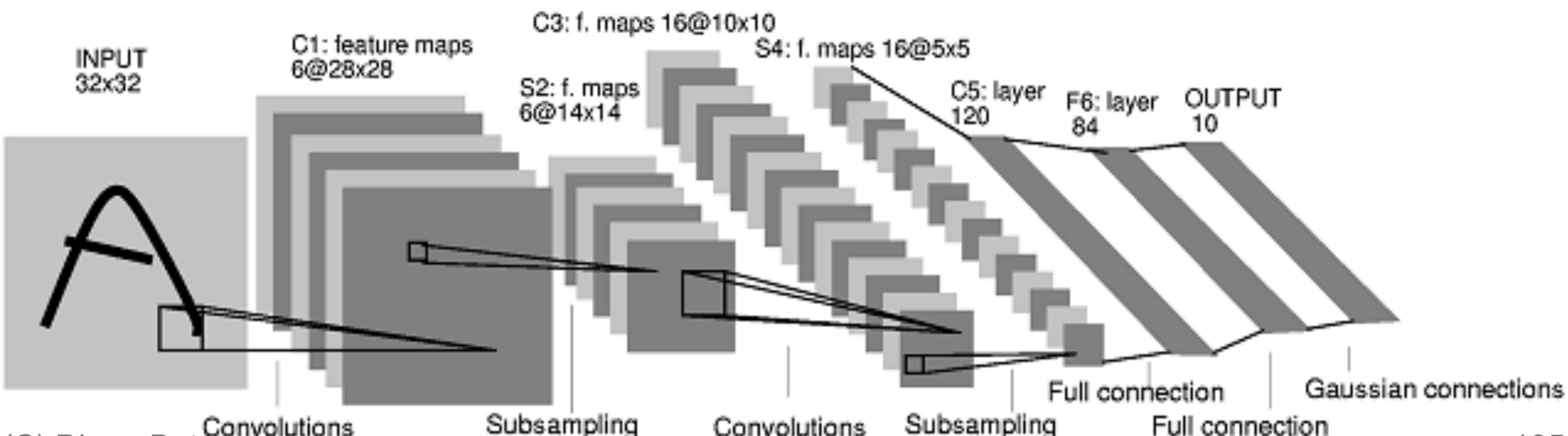
- Example:
  - http://yann.lecun.com/exdb/lenet/index.html



Note: After several stages of convolution-pooling, the spatial resolution is greatly reduced (usually to about 5x5) and the number of feature maps is large (several hundreds depending on the application).

It would not make sense to convolve again (there is no translation invariance and support is too small). Everything is vectorized and fed into several fully connected layers.

If the input of the fully connected layers is of size 5x5xN, the first fully connected layer can be seen as a conv. layer with 5x5 kernels.
The next fully connected layer can be seen as a conv. layer with 1x1 kernels.



Classical View

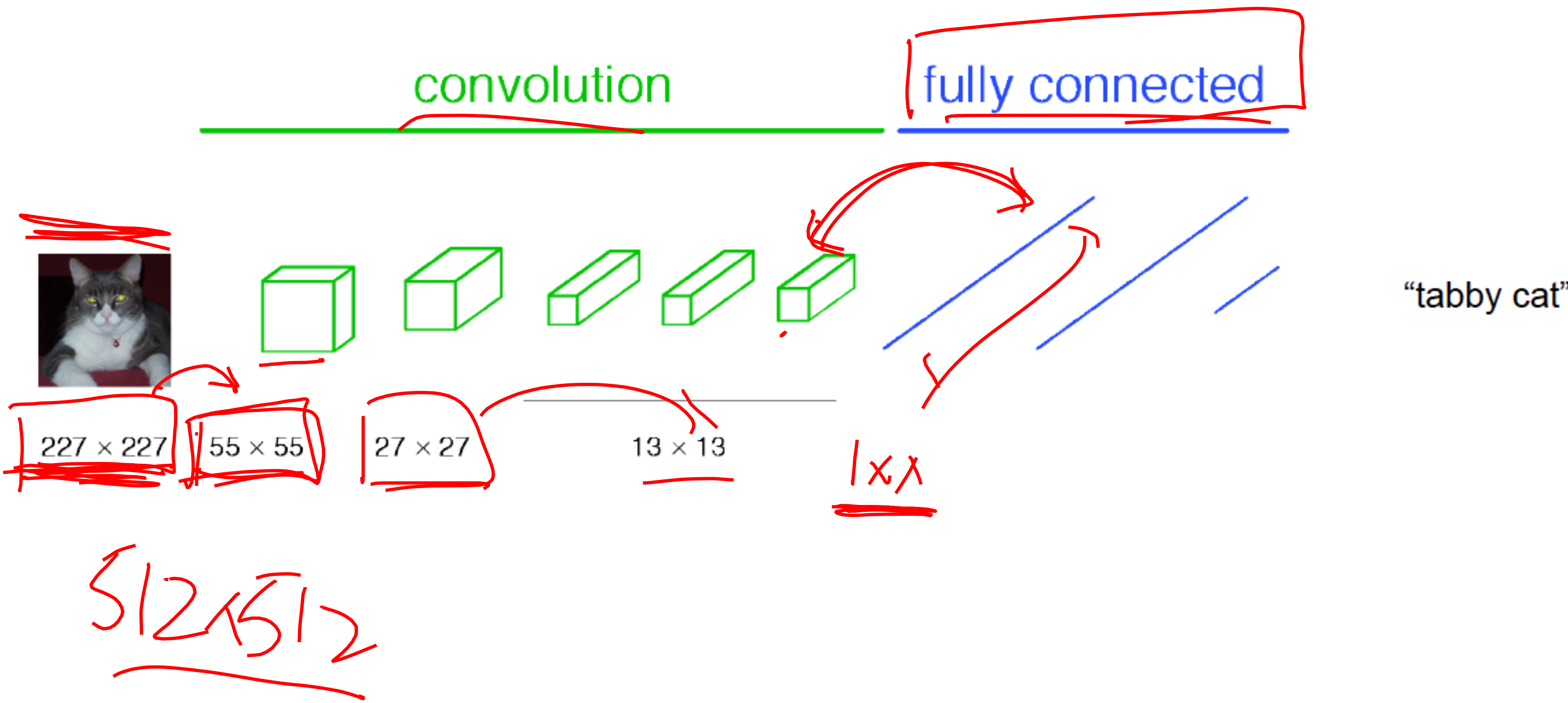
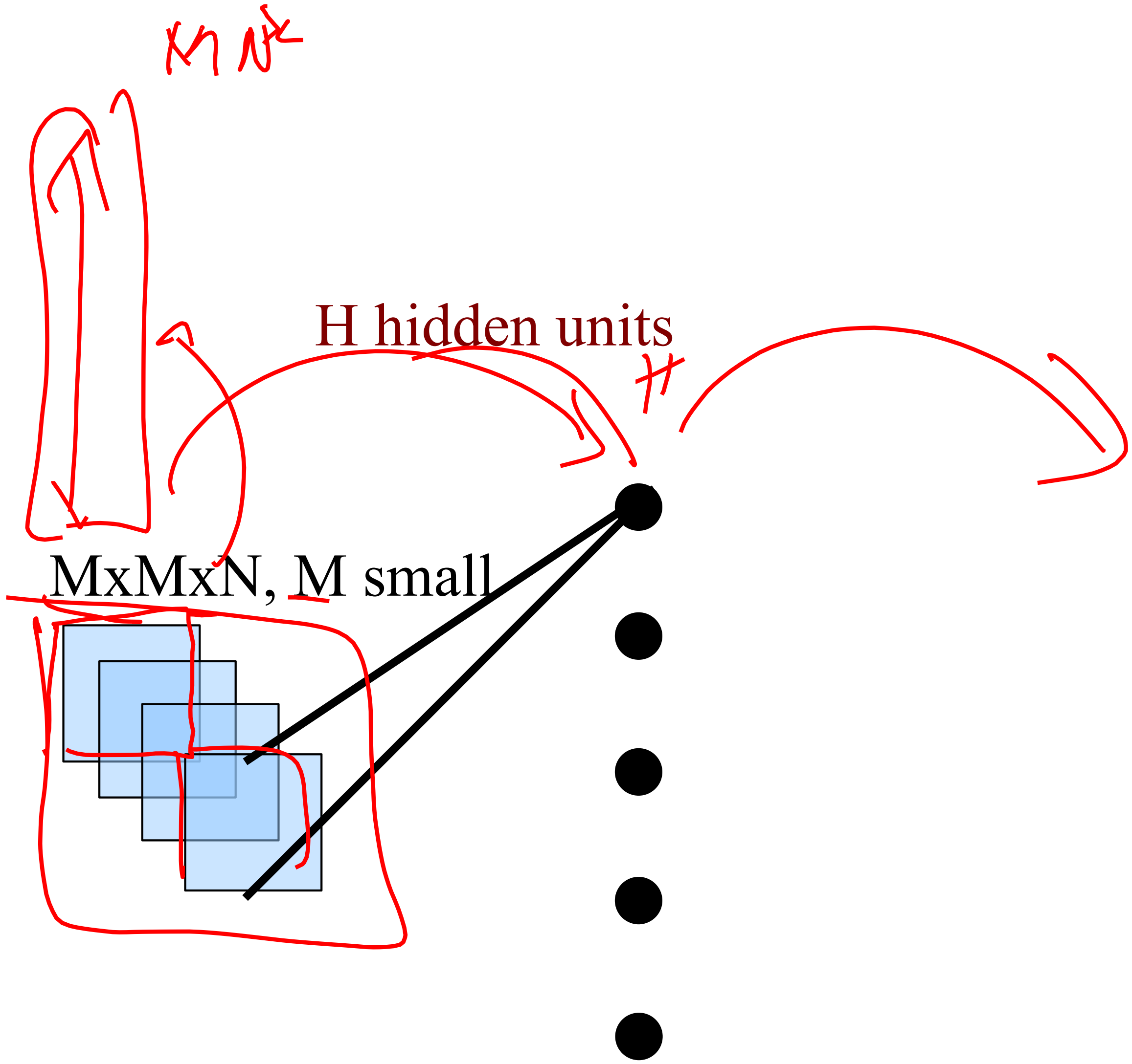
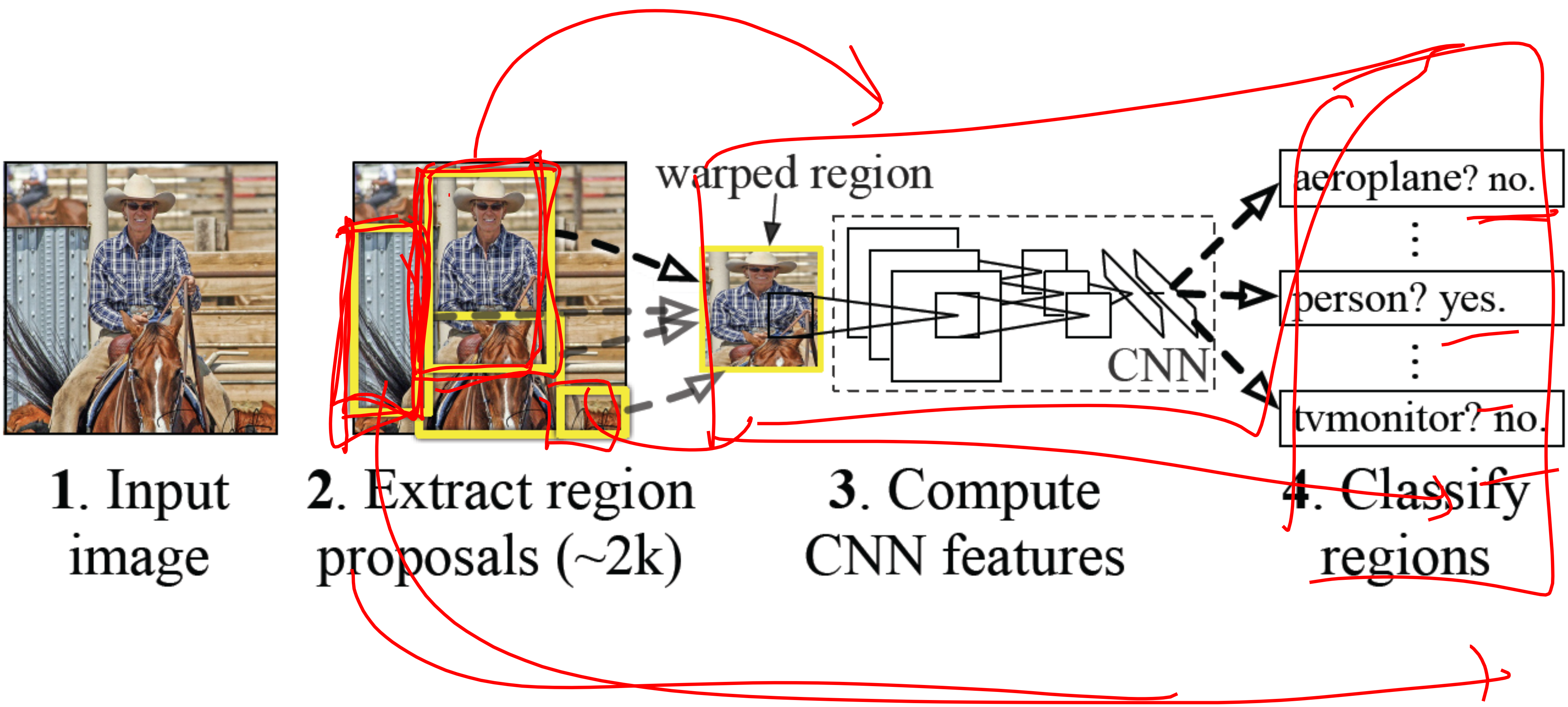


Figure Credit: [Long, Shelhamer, Darrell CVPR15]



Fully conn. layer

Classical View = Inefficient



Classical View

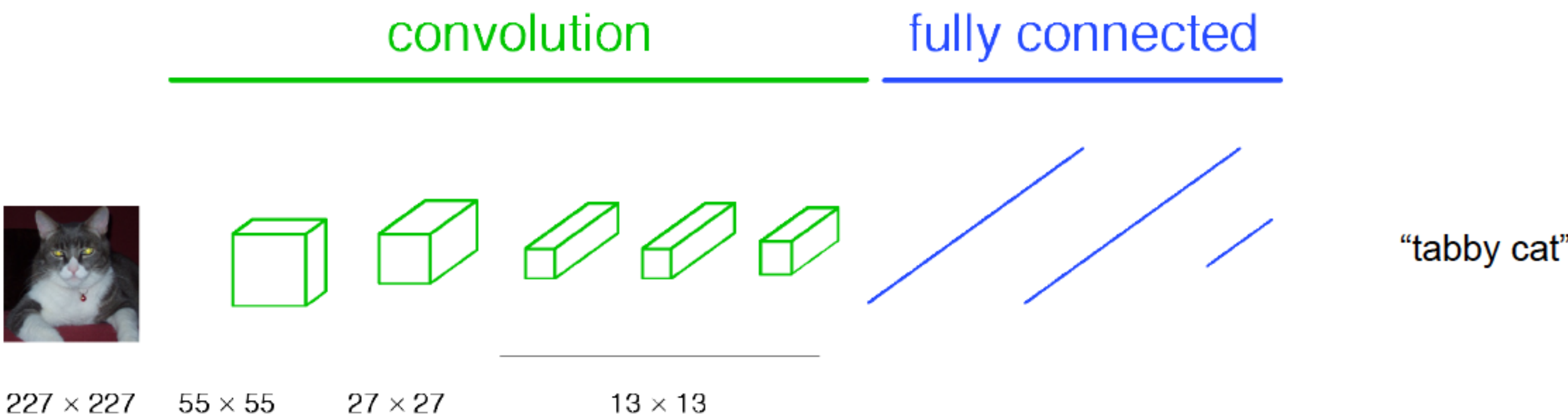


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

- Just squint a little!

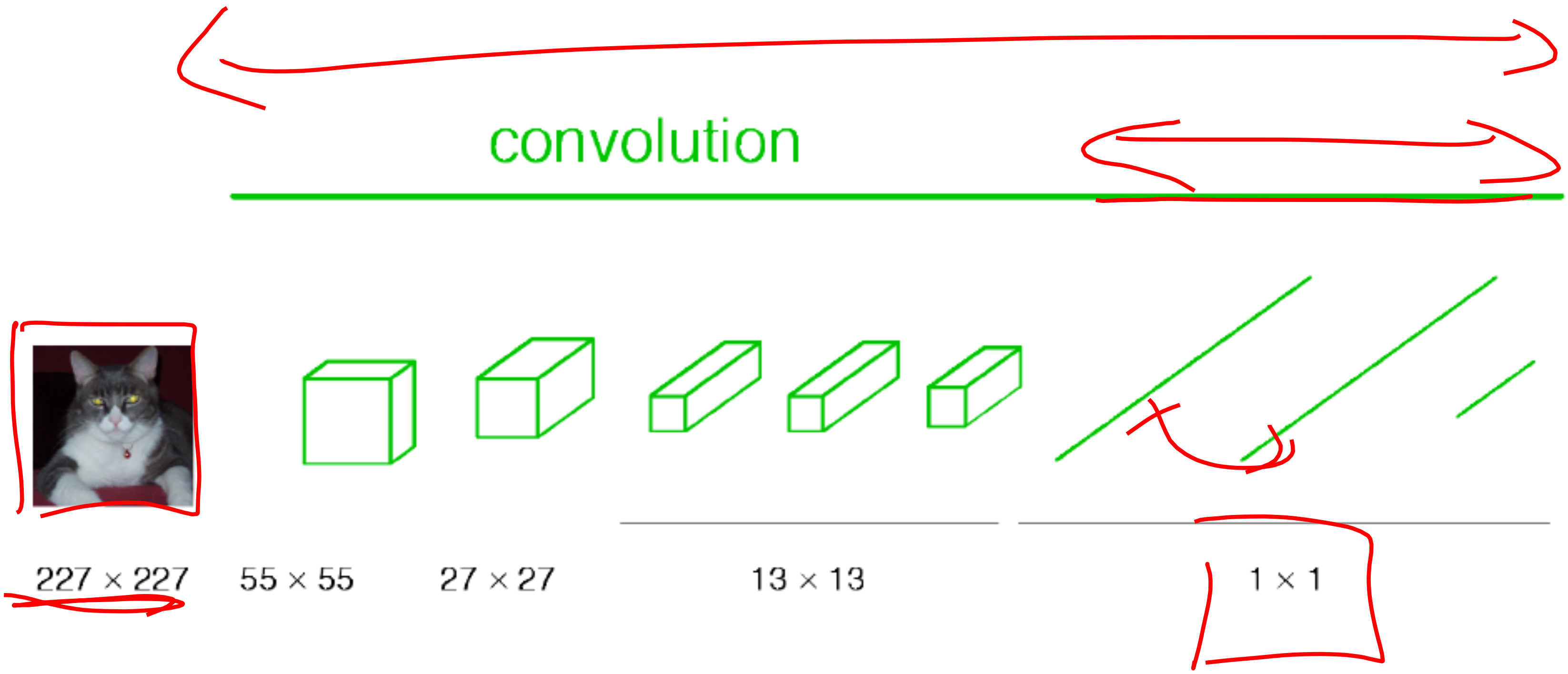
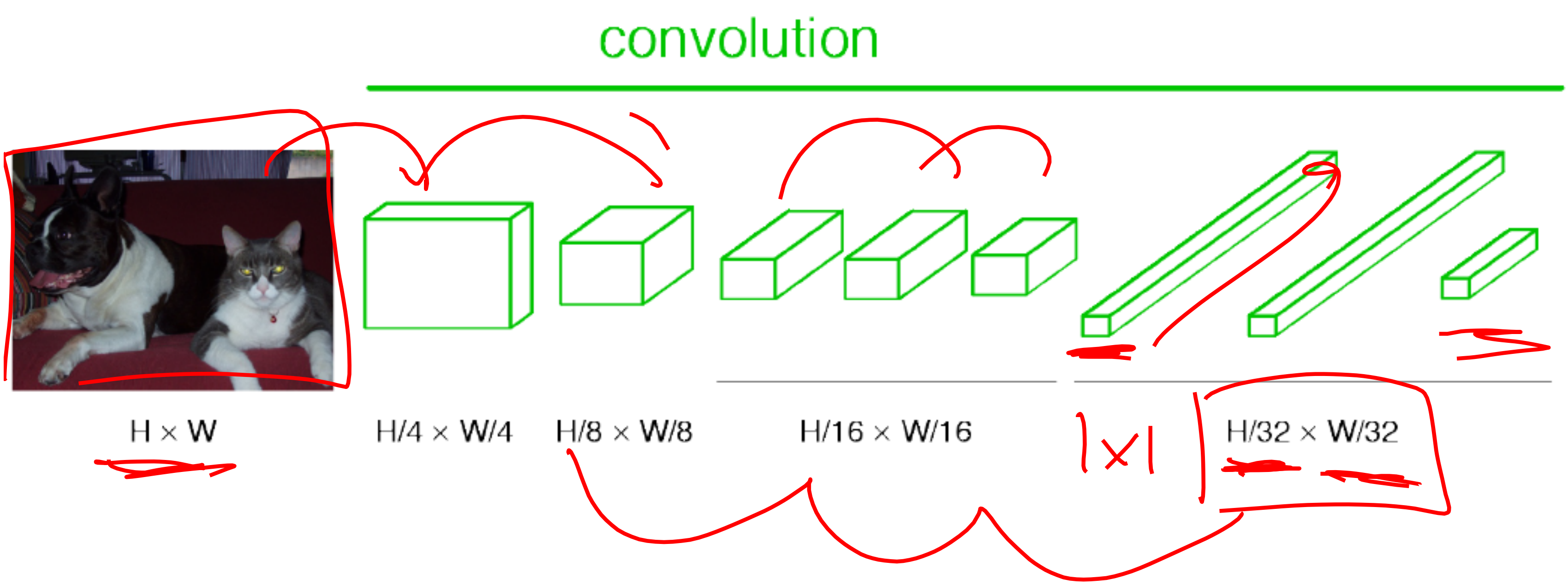


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

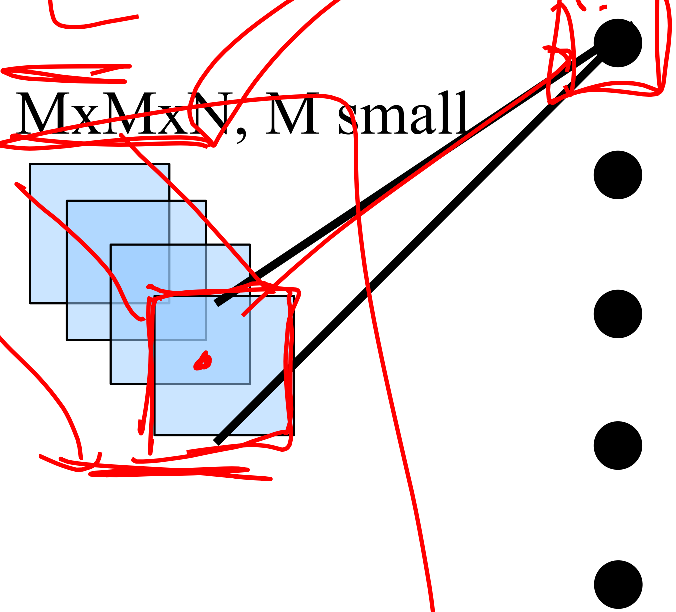
"Fully Convolutional" Networks

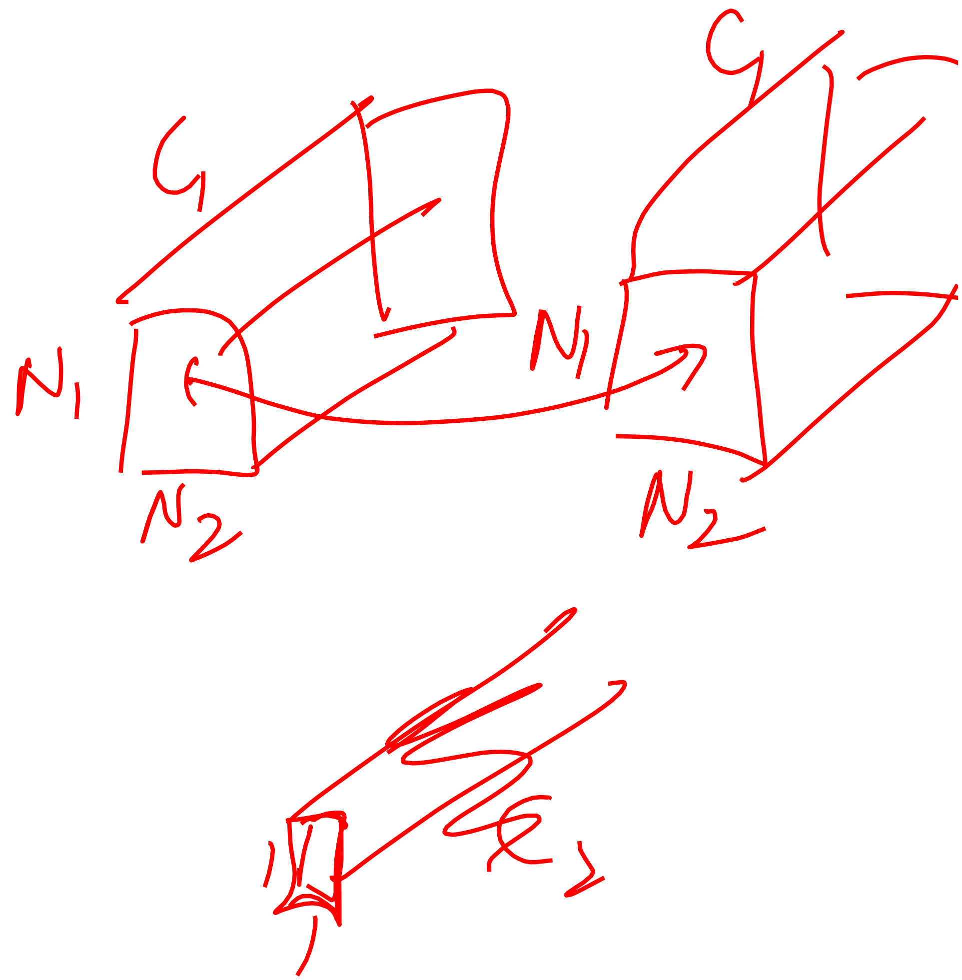
- Can run on an image of any size!



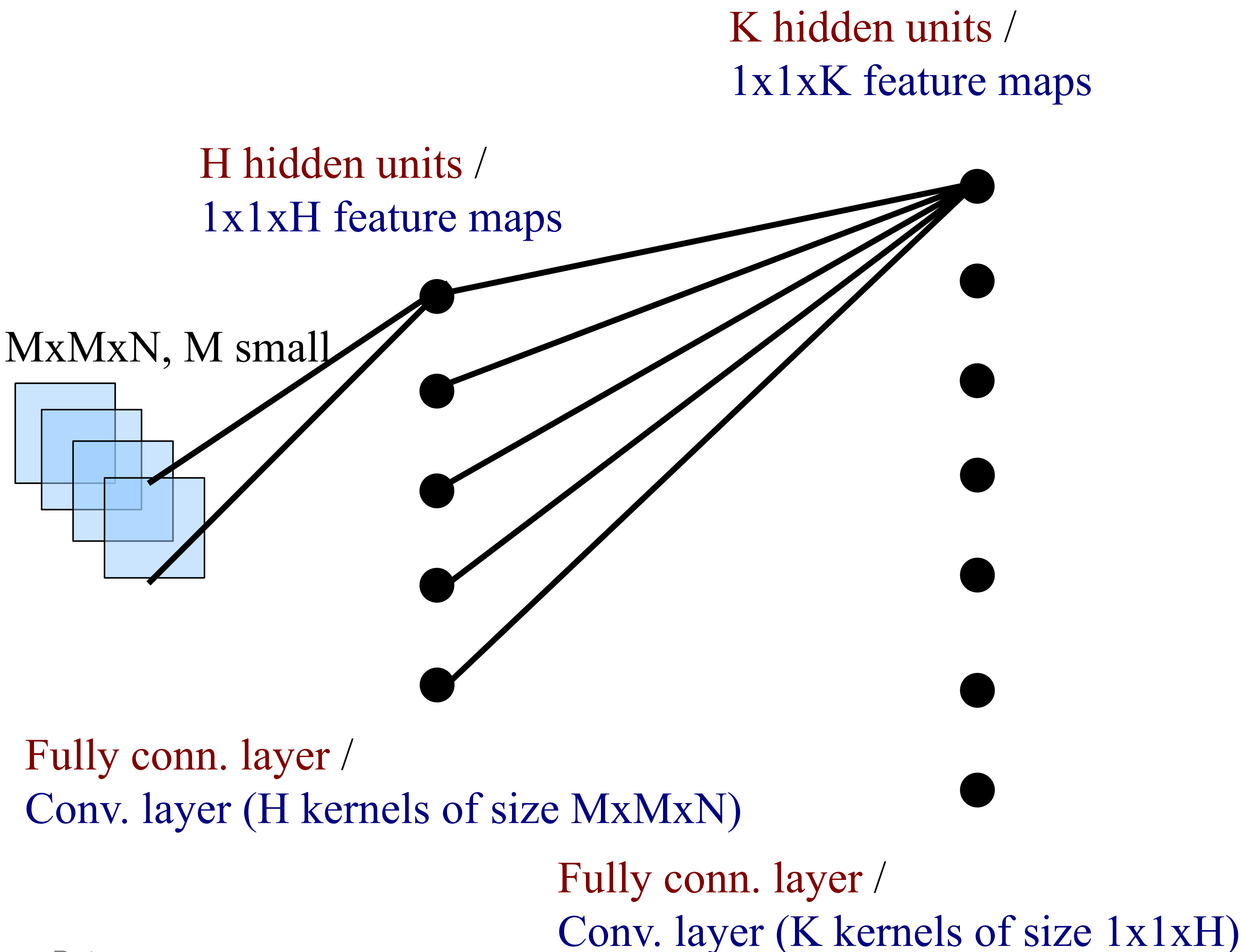
H hidden units /
1x1xH feature maps

MxMxN, M small

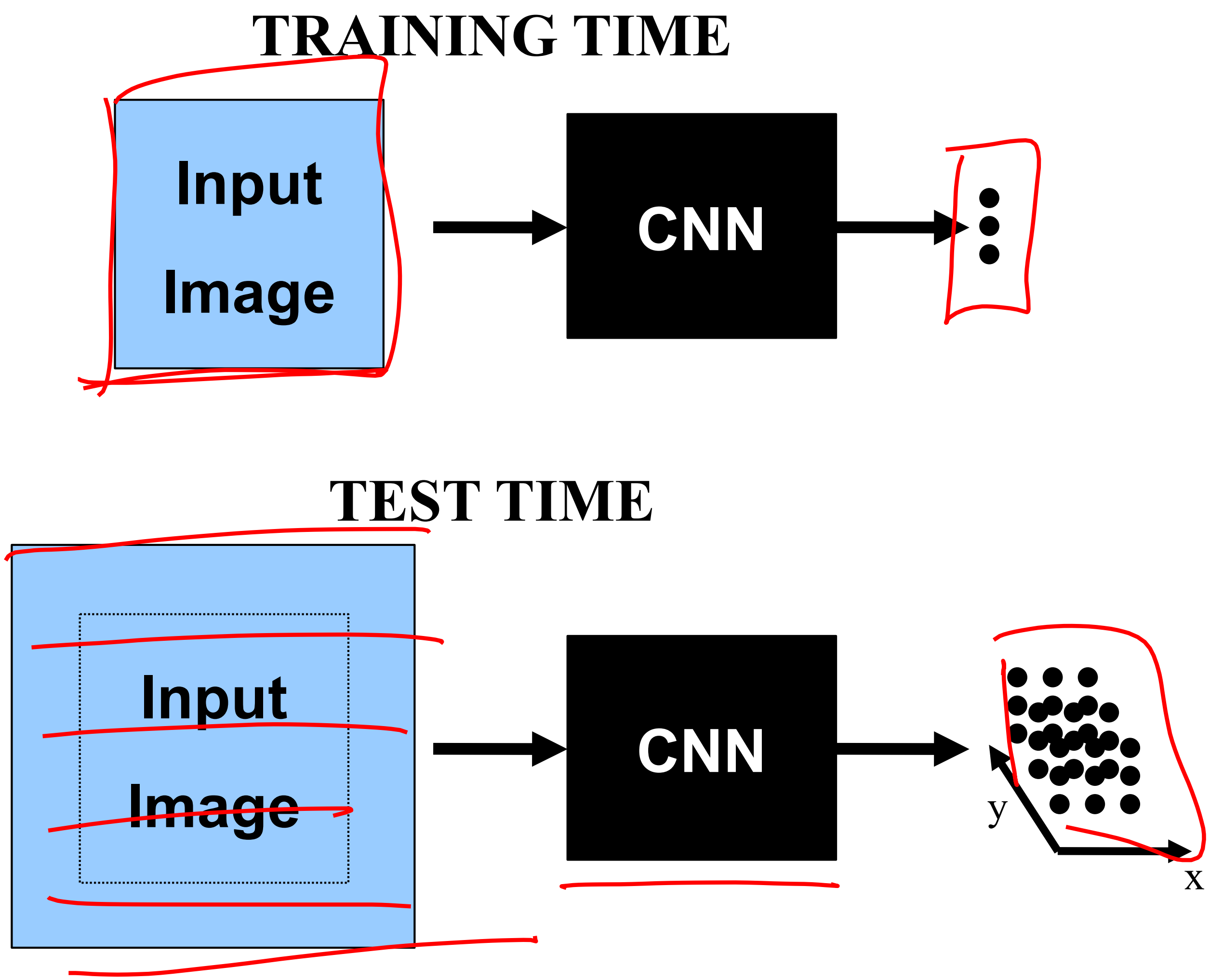




Fully conn. layer /
Conv. layer (H kernels of size MxMxN)

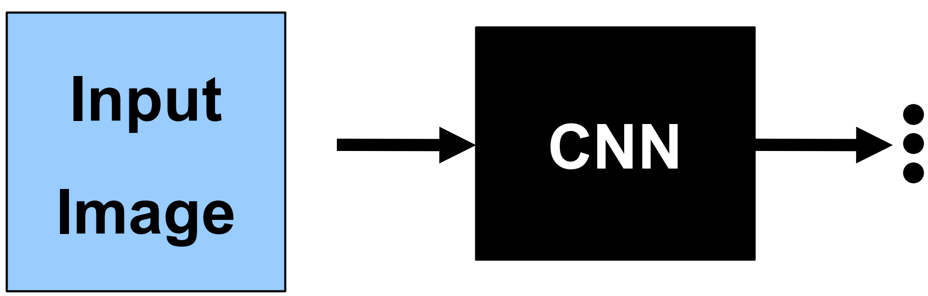

K hidden units /
1x1xK feature maps
H hidden units /
1x1xH feature maps
MxMxN, M small
Fully conn. layer /
Conv. layer (H kernels of size MxMxN)
Fully conn. layer /
Conv. layer (K kernels of size 1x1xH)

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).



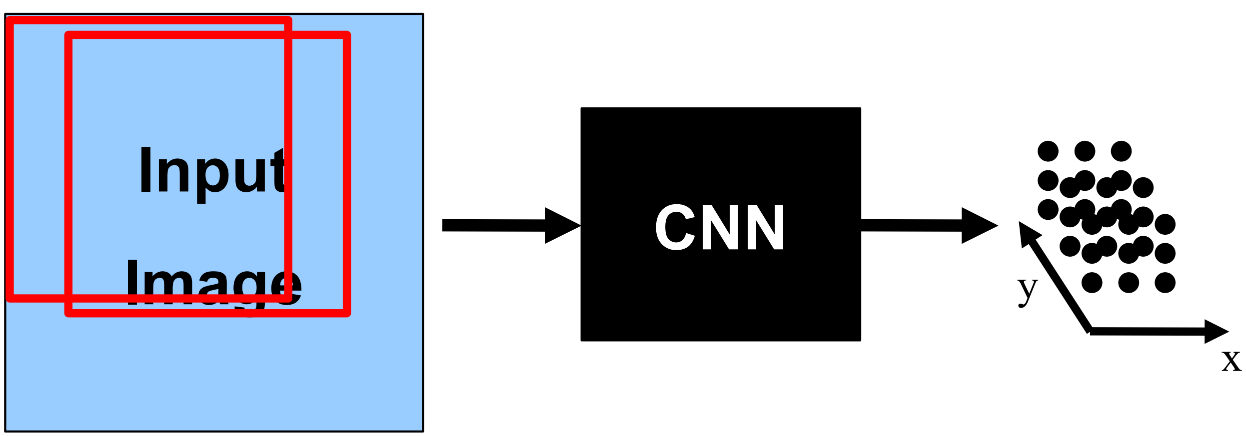
Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).

TRAINING TIME



TEST TIME

CNNs work on any image size!



Unrolling is order of magnitudes more eficient than sliding windows!

Re-interpretation

- Just squint a little!

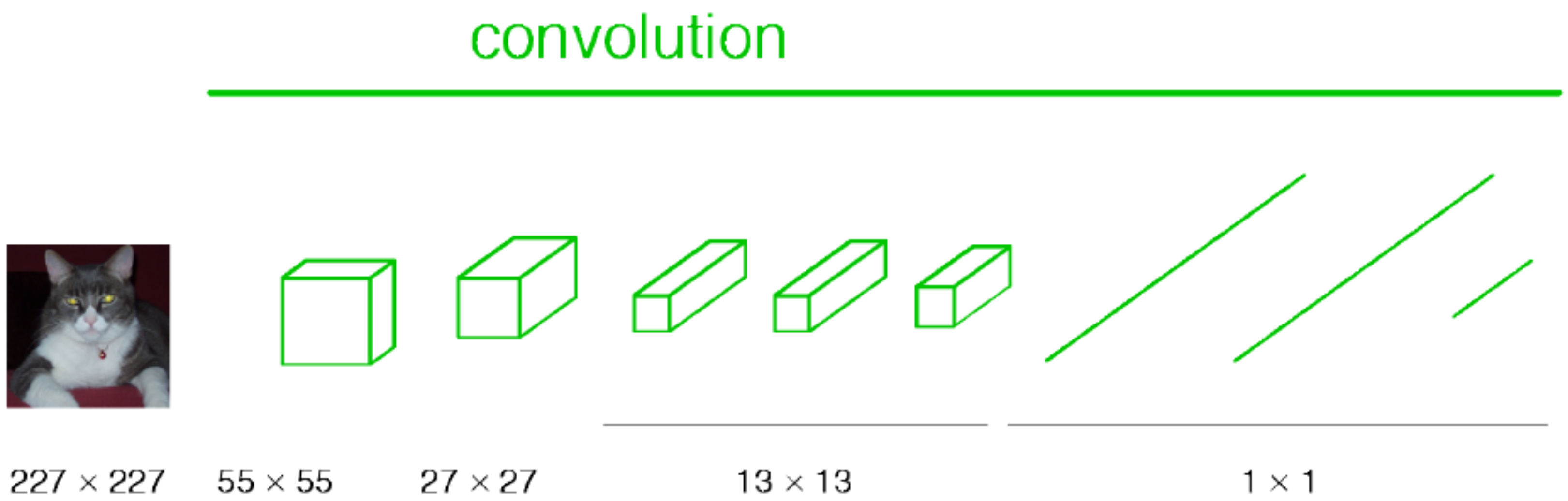


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

“Fully Convolutional” Networks

- Can run on an image of any size!

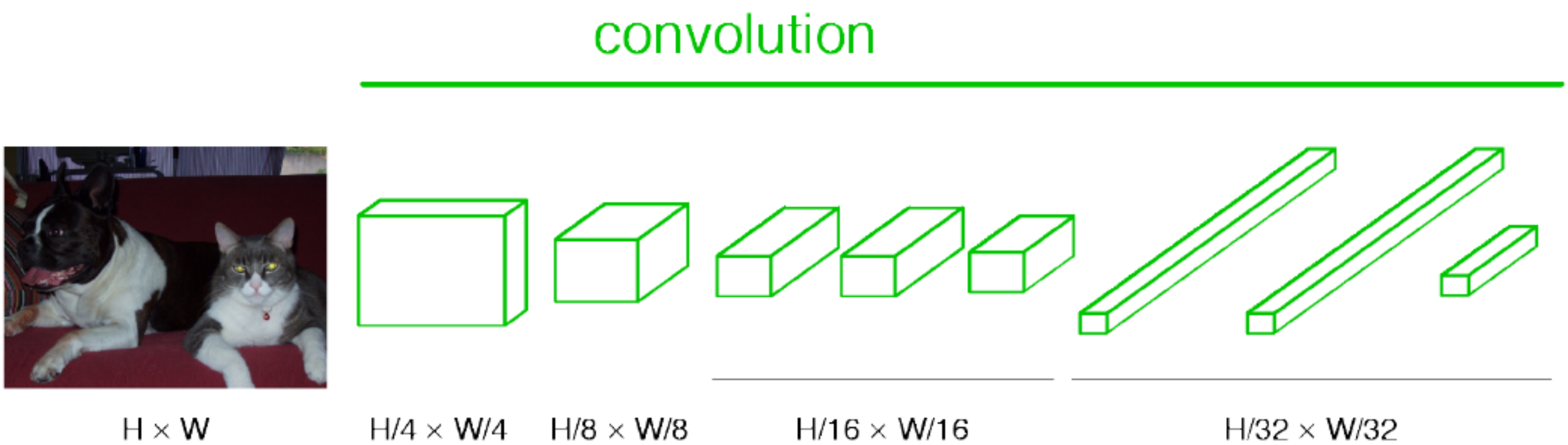


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

“Fully Convolutional” Networks

- Up-sample to get segmentation maps

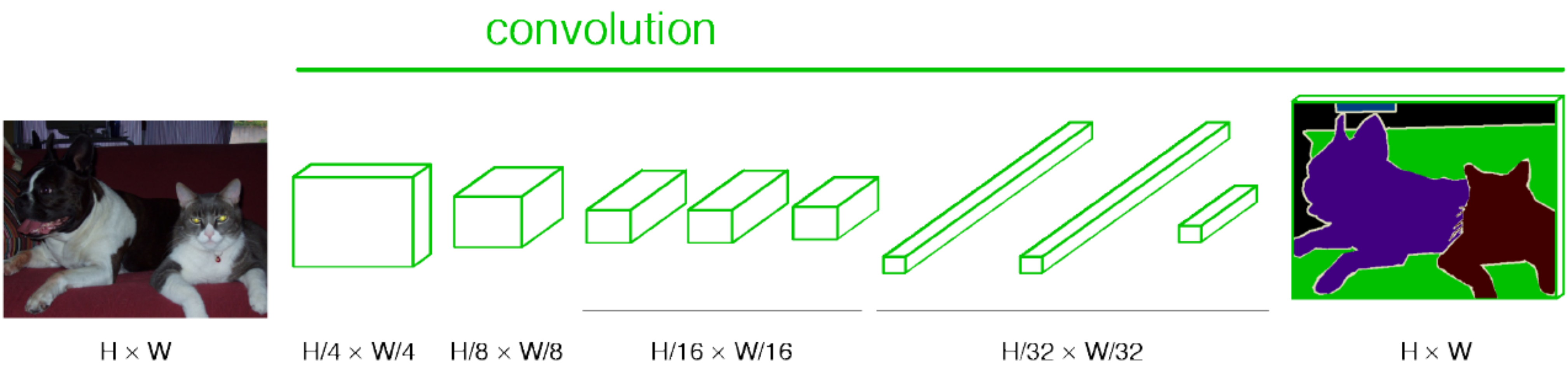


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

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

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