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

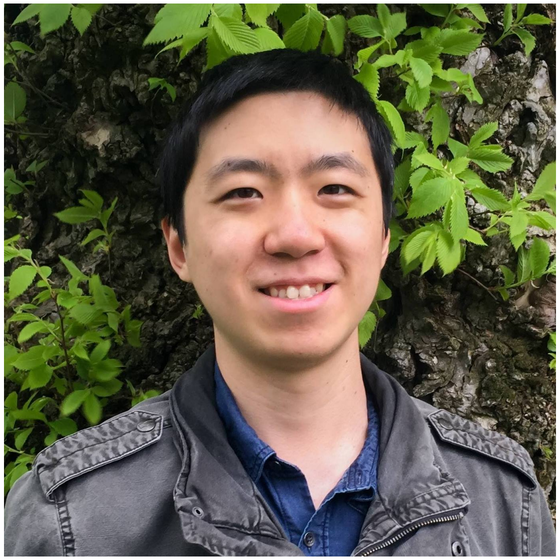
Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 09/26, 11:55pm
  - https://evalai.cloudcv.org/web/challenges/challenge-page/431/leaderboard/1200
- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8I5Zd3yOFs7E/edit?usp=sharing
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names

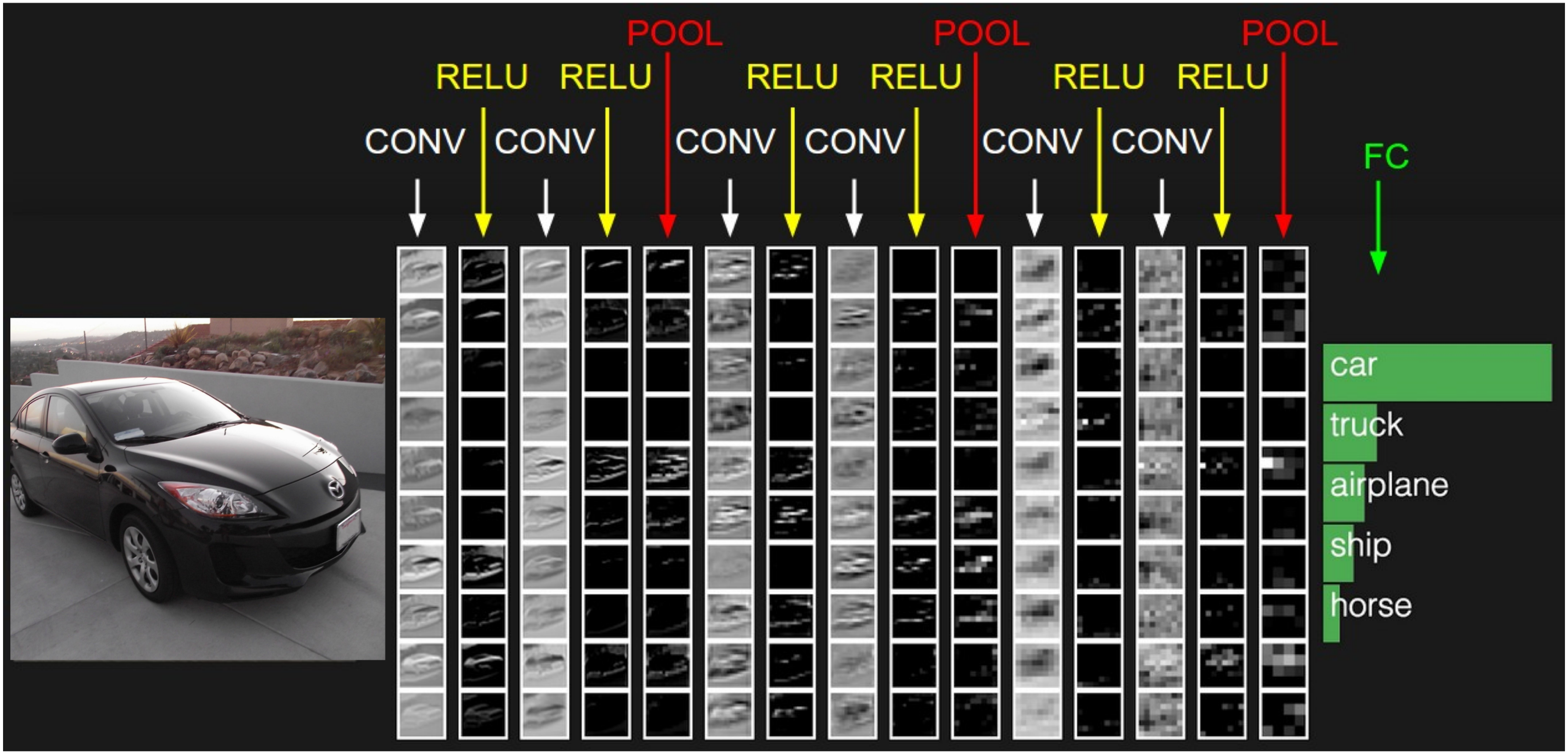
Administrativia

- Guest Lecture: Dr. Zhile Ren
  - Next class (09/26)
  - CNN architectures for 2D & 3D Detection & Segmentation



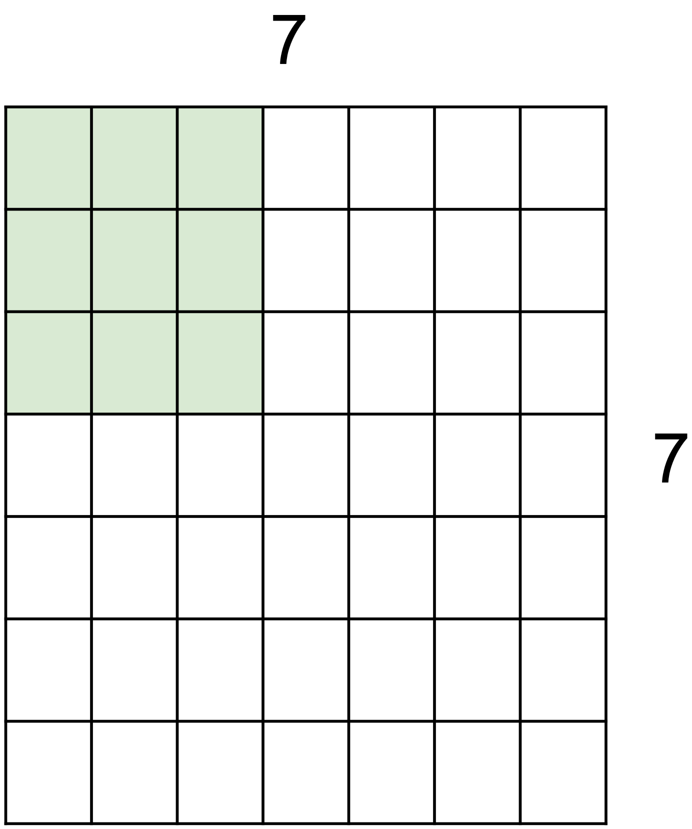
Recap from last time

preview:



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

A closer look at spatial dimensions:



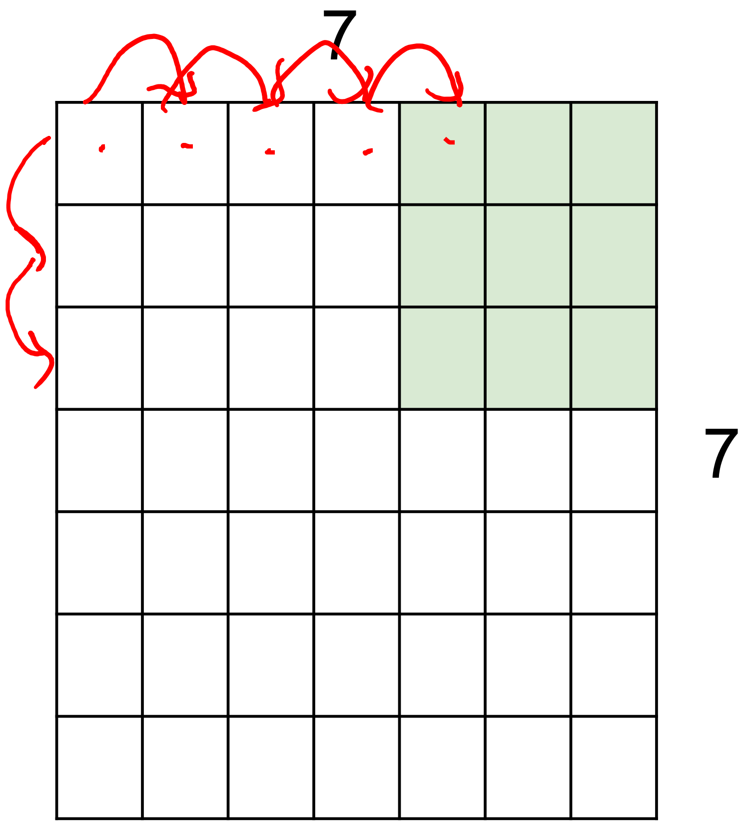
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

stride = 1

'valid'

'same'

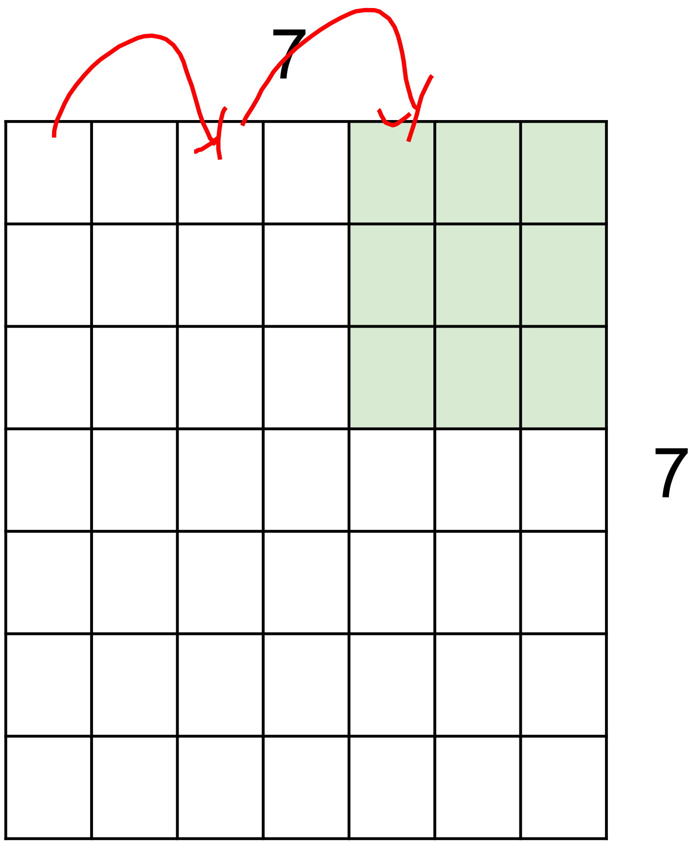


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

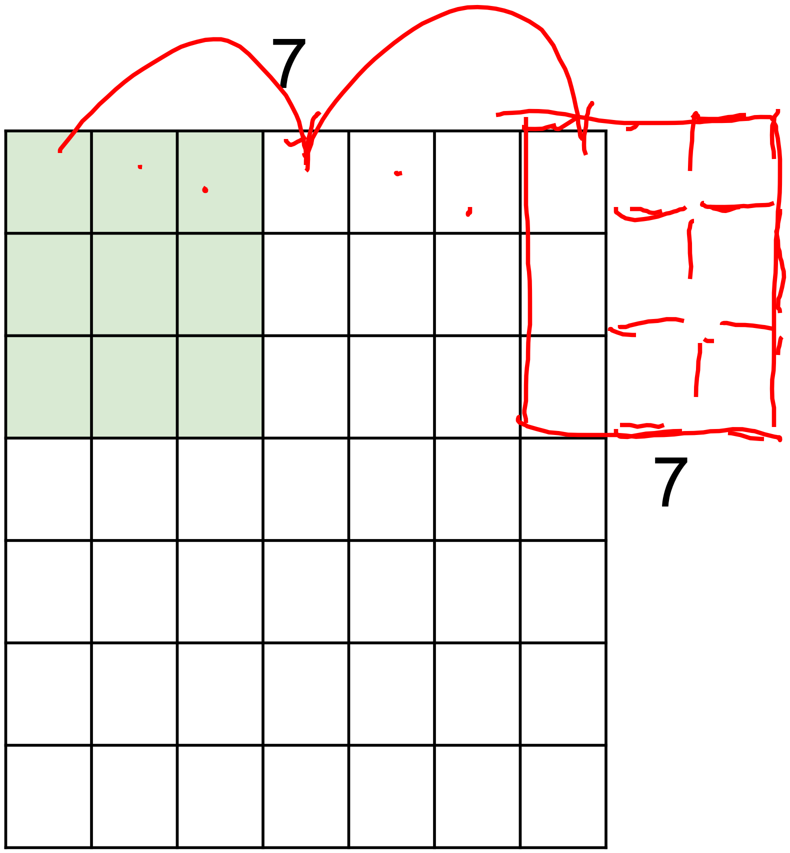
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

A closer look at spatial dimensions:



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

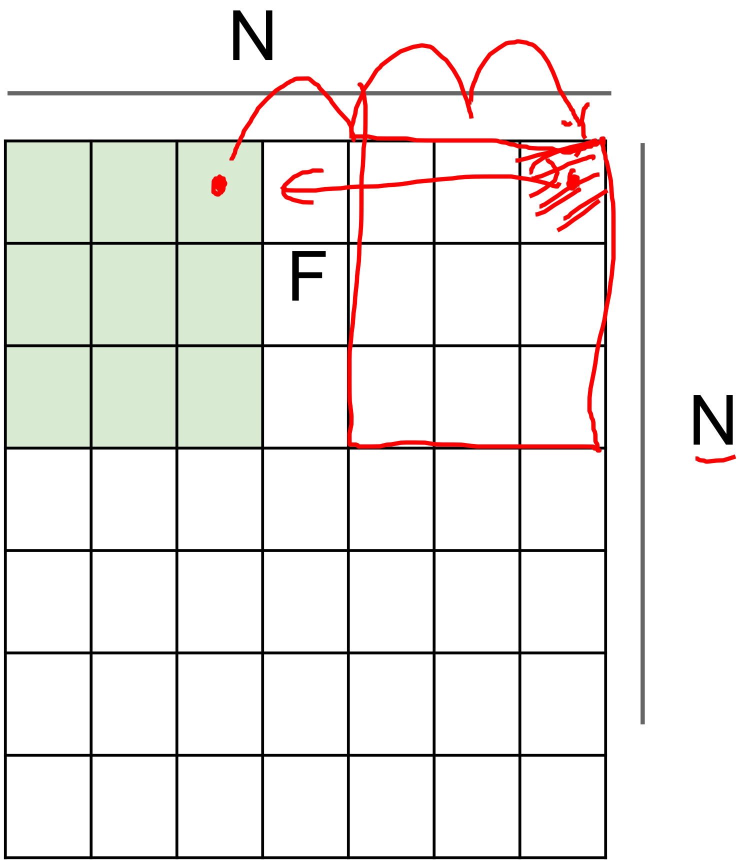
A closer look at spatial dimensions:



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

'valid'

$N-F \propto$ stride



Output size: $= \frac{N-F}{\text{stride}}$

(N - F) / stride + 1

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

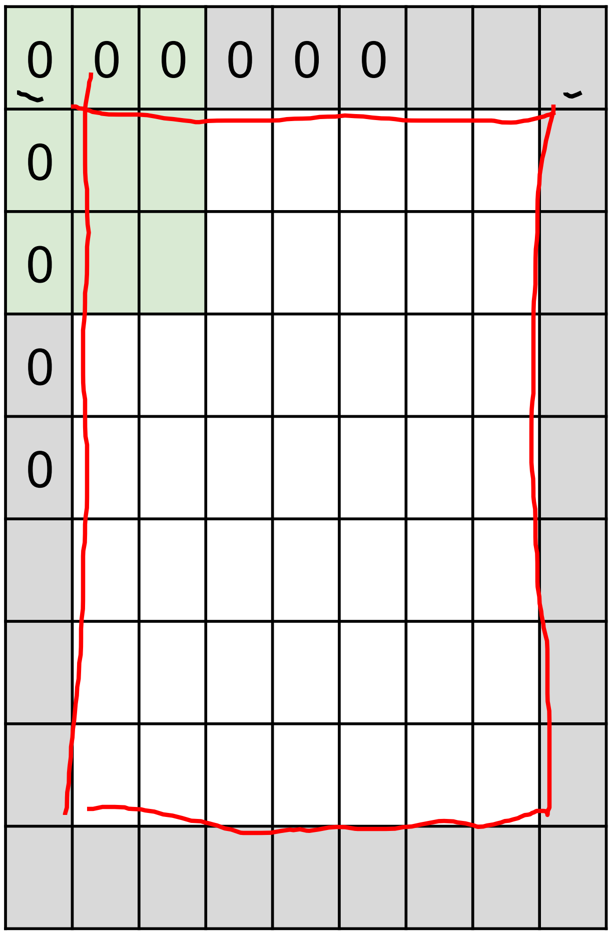
stride 1 => (7 - 3)/1 + 1 = 5

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

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

$$\frac{7-3}{4} + 1$$

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?

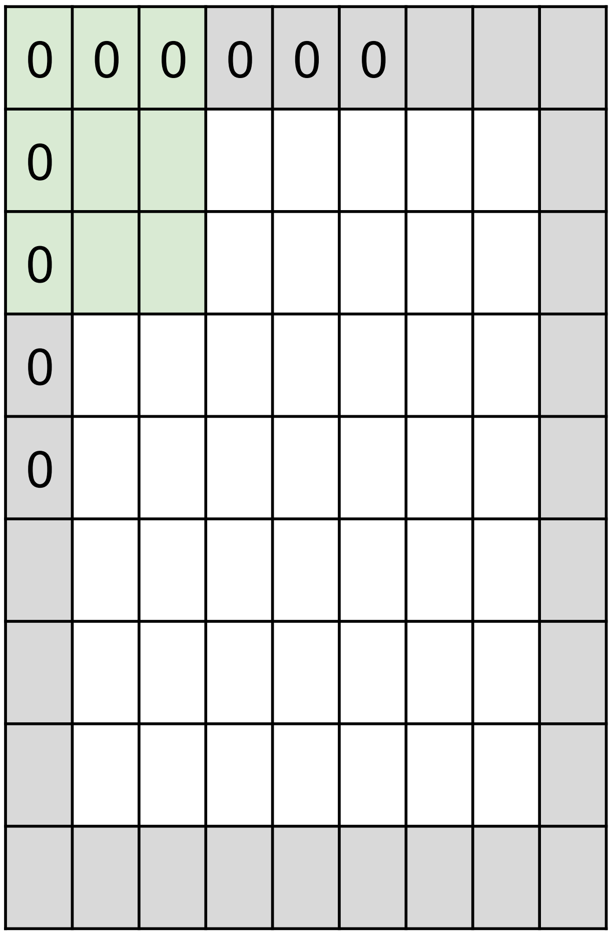
$$N' = \left\lfloor \frac{N + 2\,pad - F}{stride} + 1 \right\rfloor$$

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

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

$$N' = \frac{9-3}{1} + 1 = 6 + 1 = 7$$

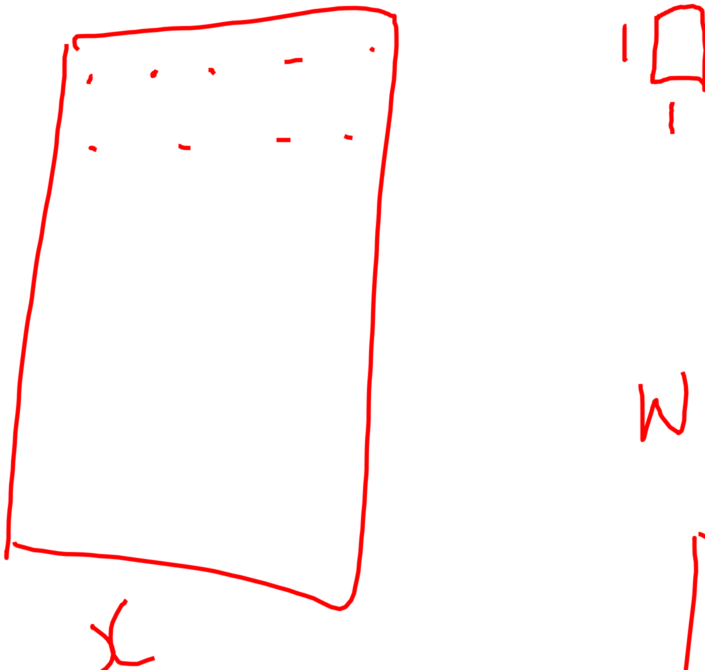
In practice: Common to zero pad the border

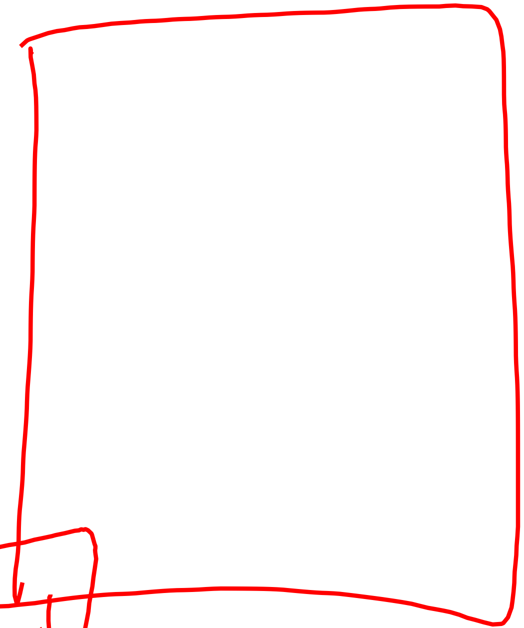


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Can we have $\boxed{1x1}$ filters?

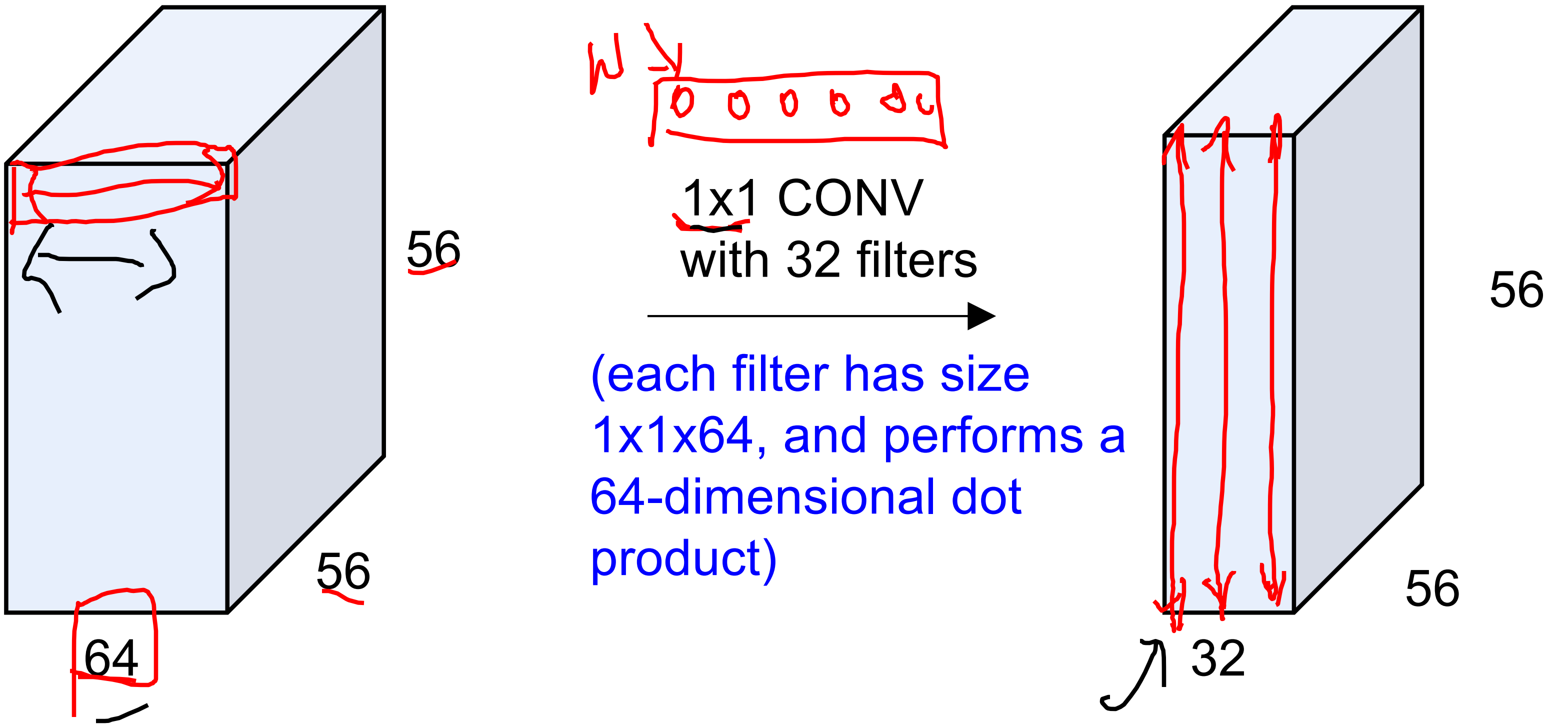




$$y[r,c] = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} x[r+a,\, c+b]\, w[a,b]$$

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

1x1 convolution layers make perfect sense

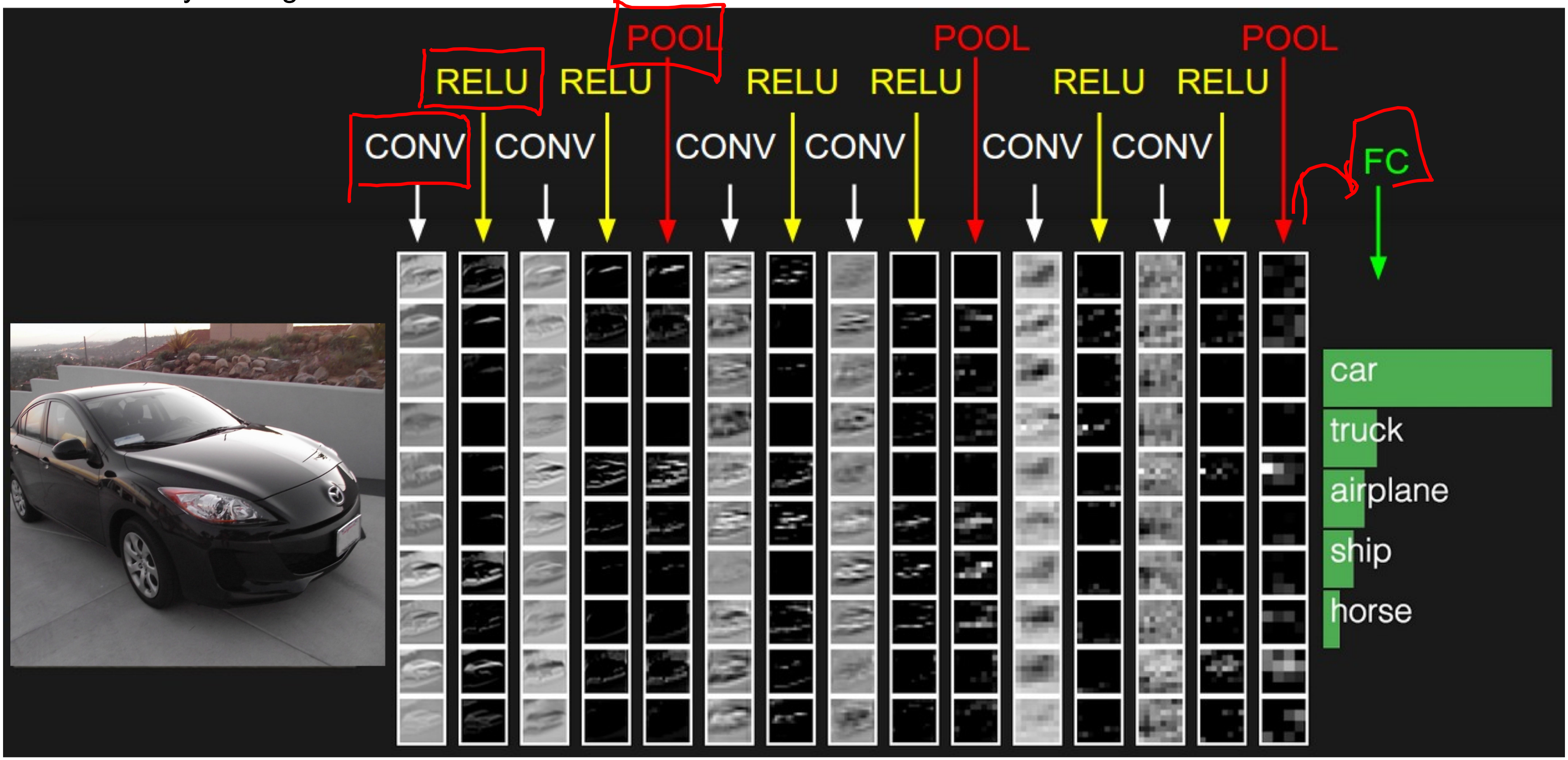


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Plan for Today

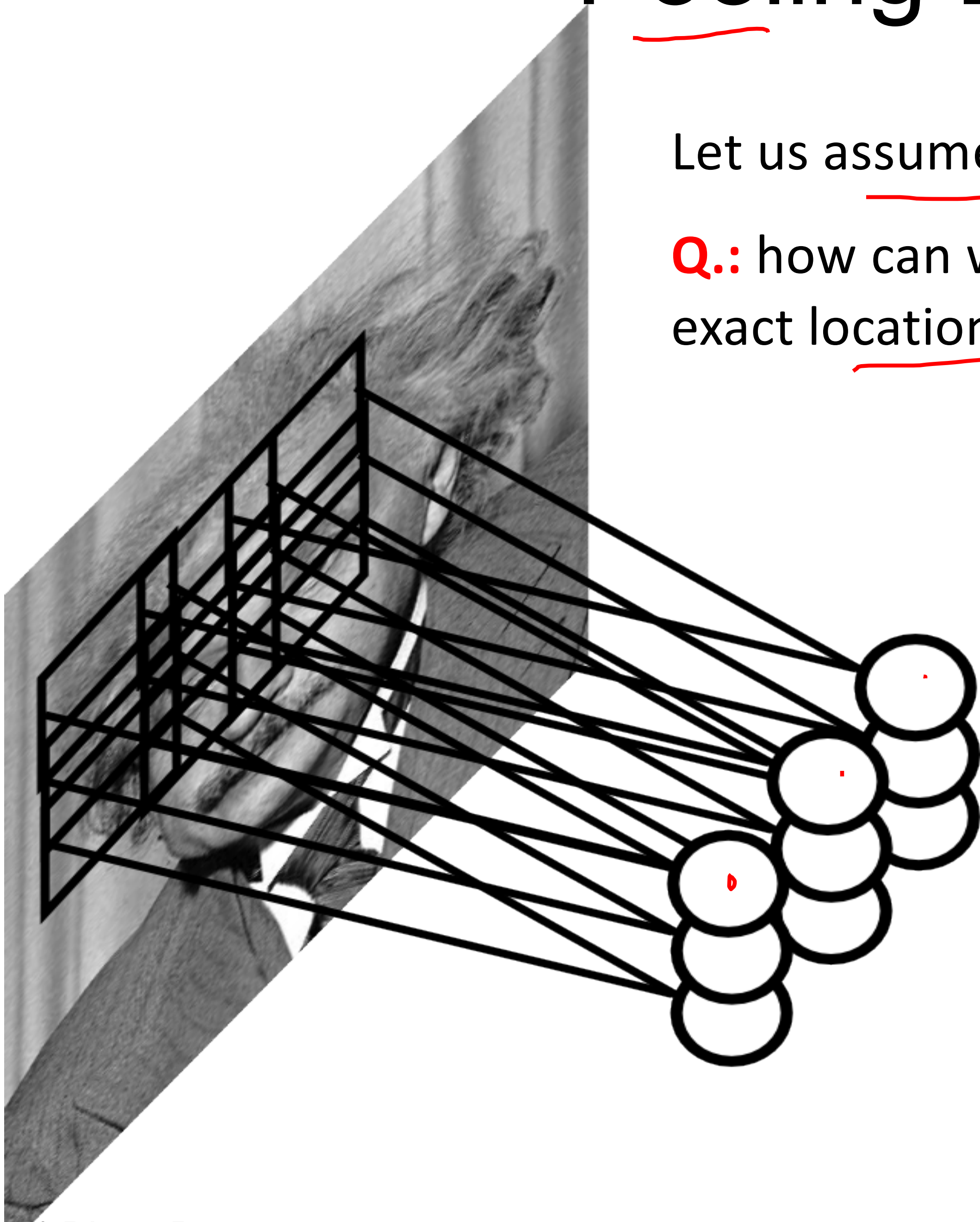
- Convolutional Neural Networks
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers [Derived in notes]
  - Toeplitz matrices and convolutions = matrix-mult

two more layers to go: POOL/FC



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

Pooling Layer

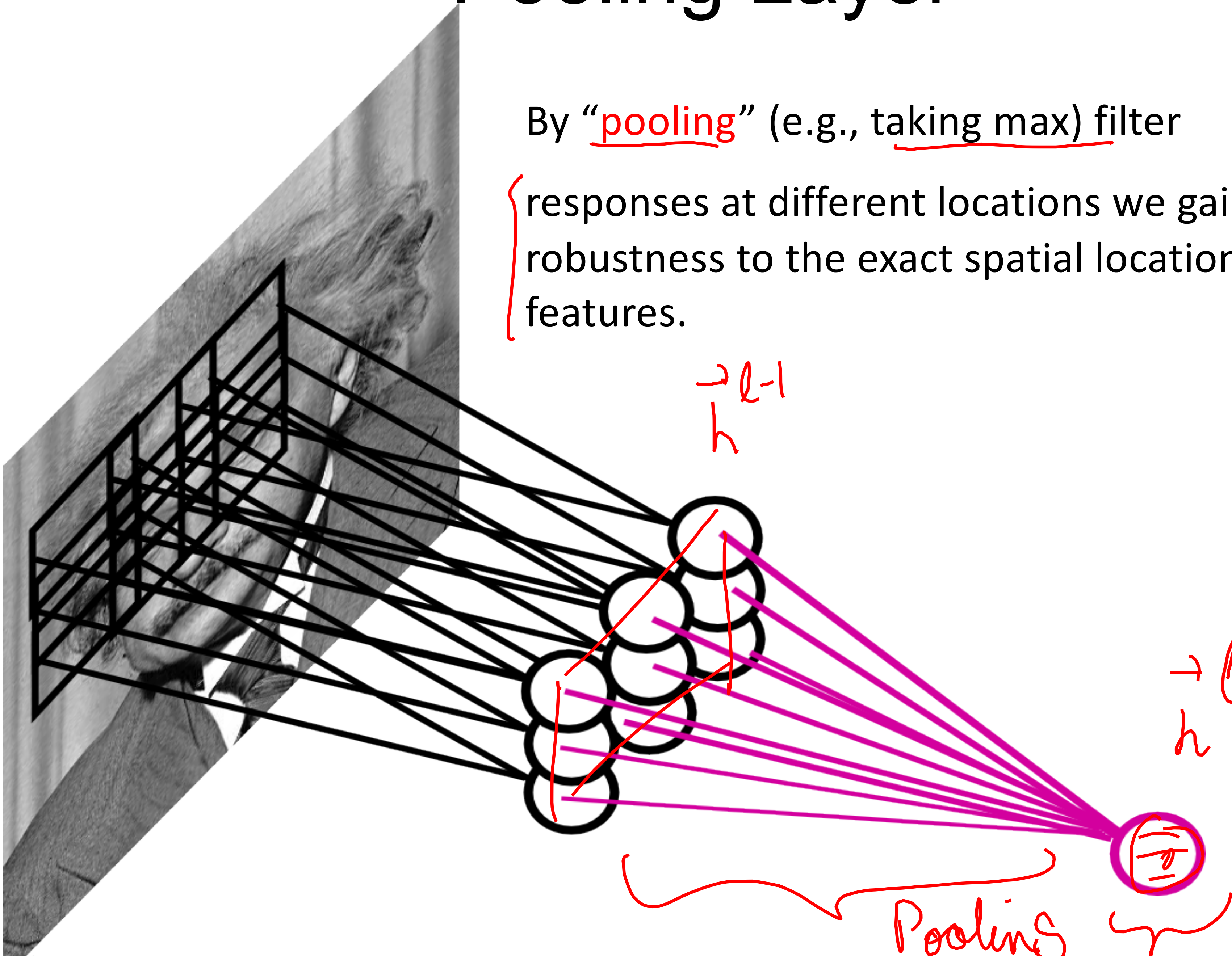


Let us assume filter is an “eye” detector.

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

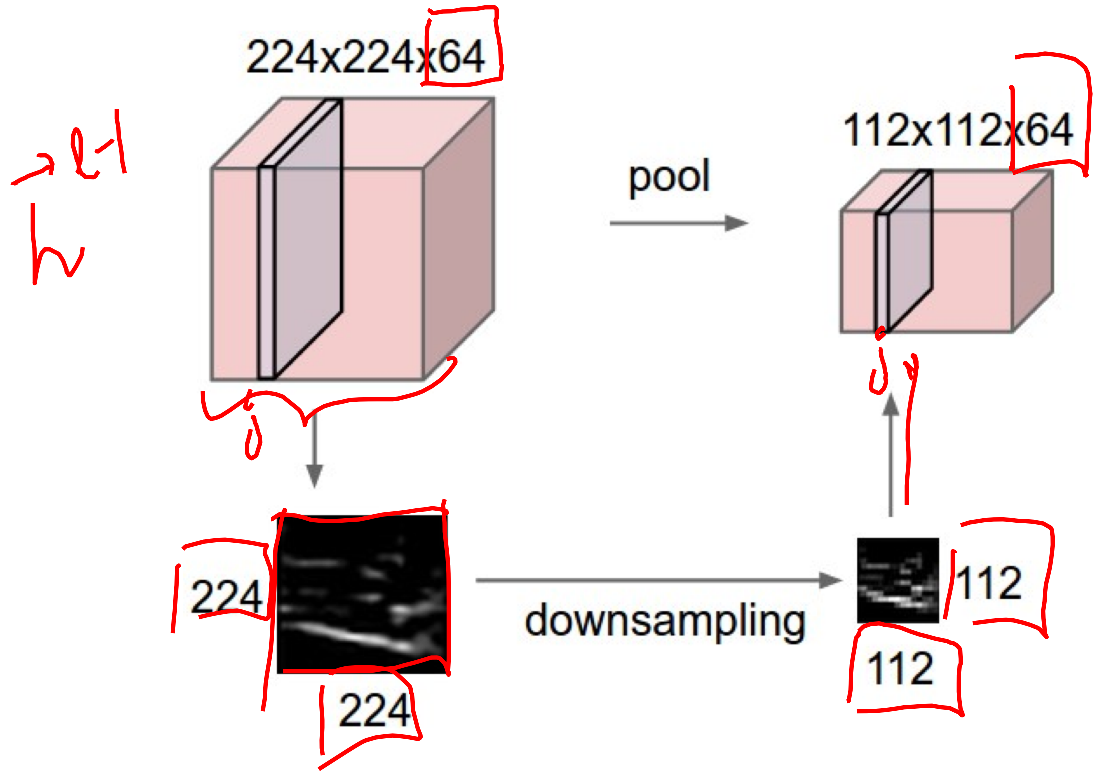
Pooling Layer

By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

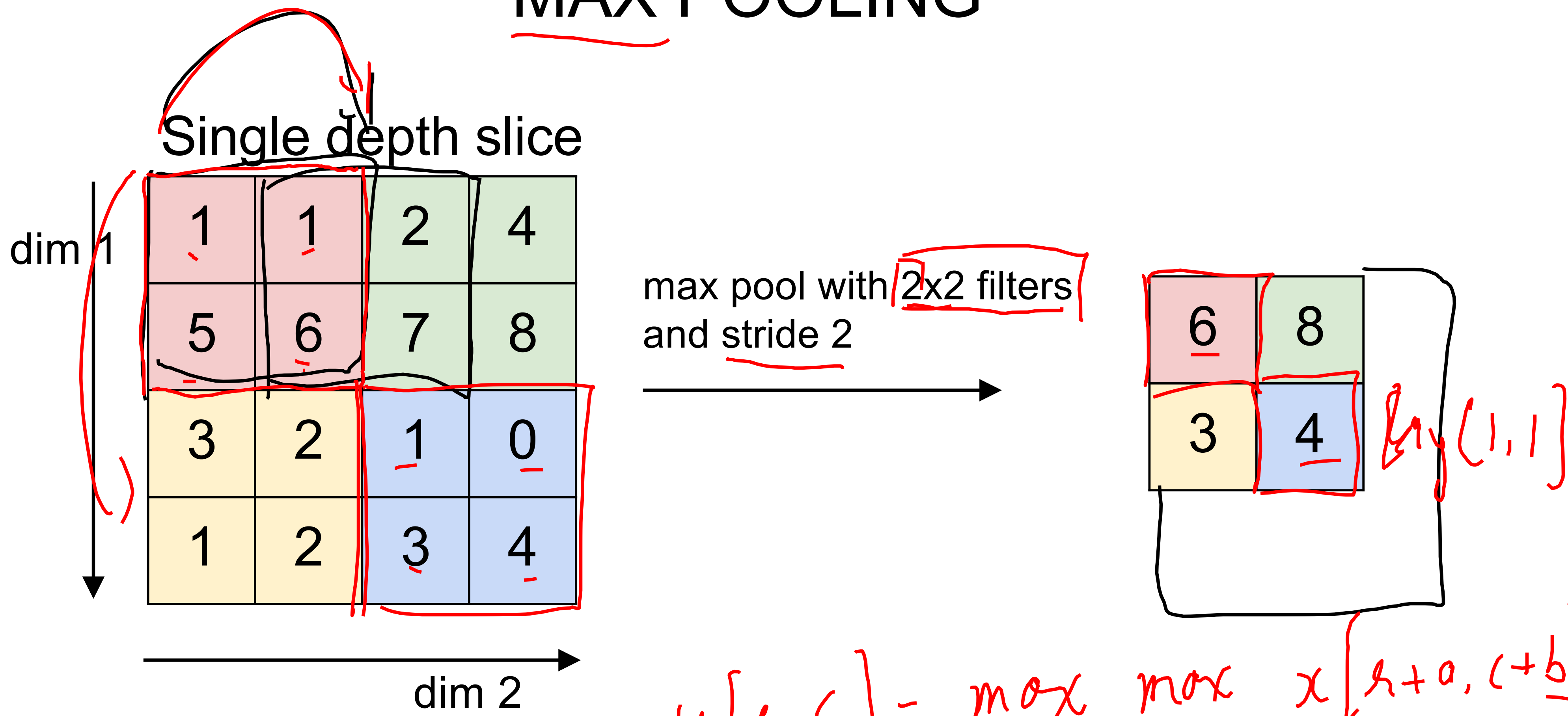


Pooling layer

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



MAX POOLING



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

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

Image courtesy: Justin Francis.

Pooling Layer: Examples

Max-pooling:

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

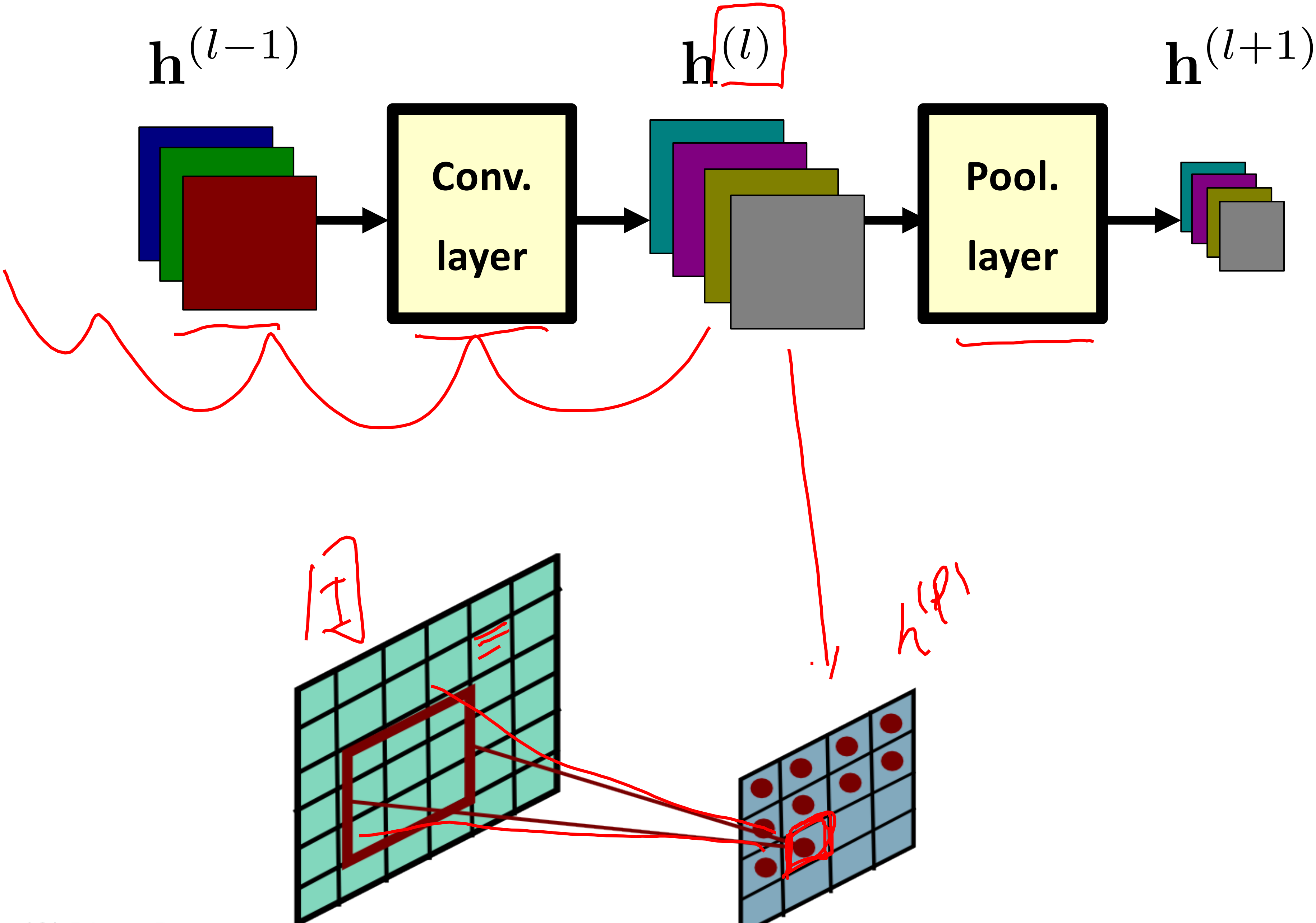
Average-pooling:

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

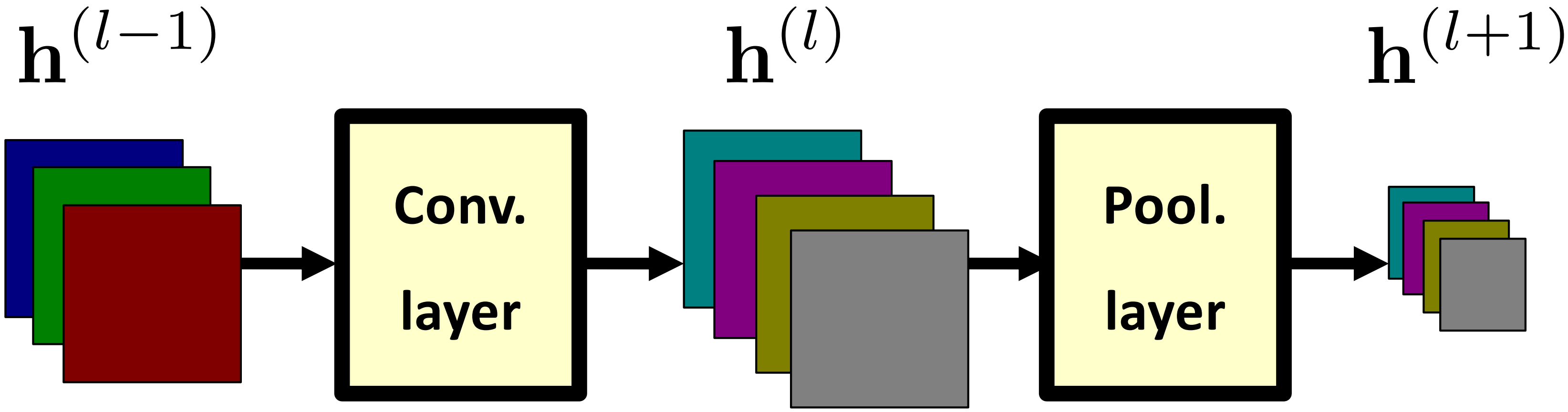
L2-pooling:

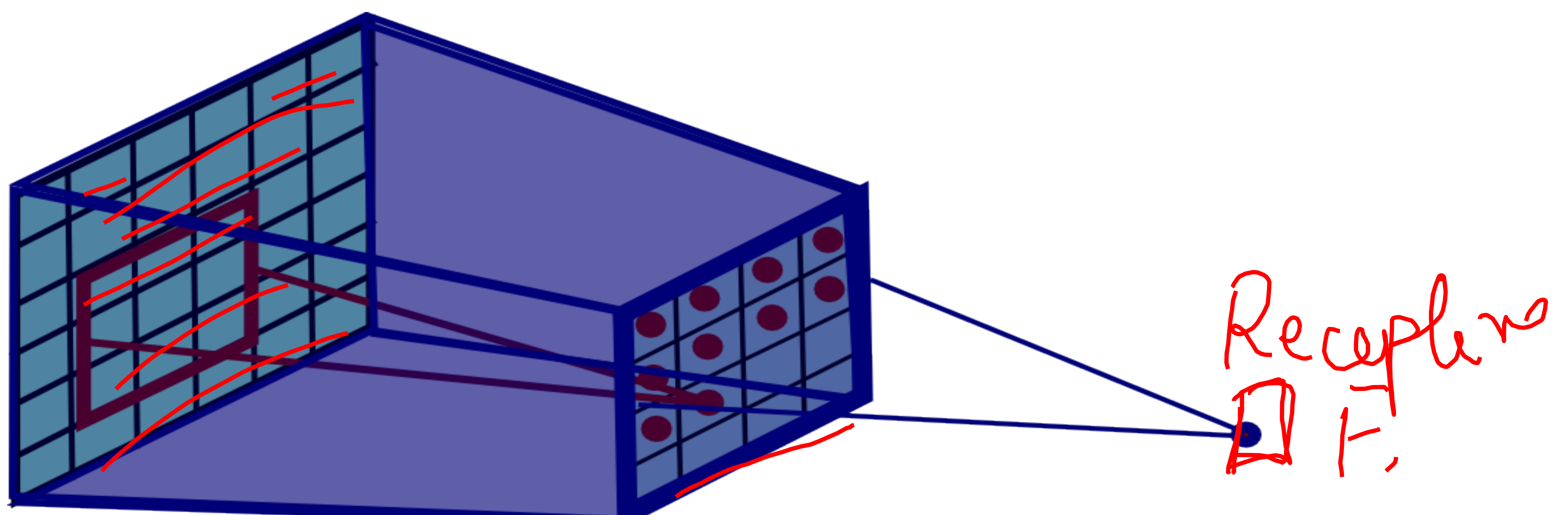
$$h_i^n(r,c) = \sqrt{\sum_{\bar{r} \in N(r),\ \bar{c} \in N(c)} h_i^{n-1}(\bar{r},\bar{c})^2}$$

Receptive Field

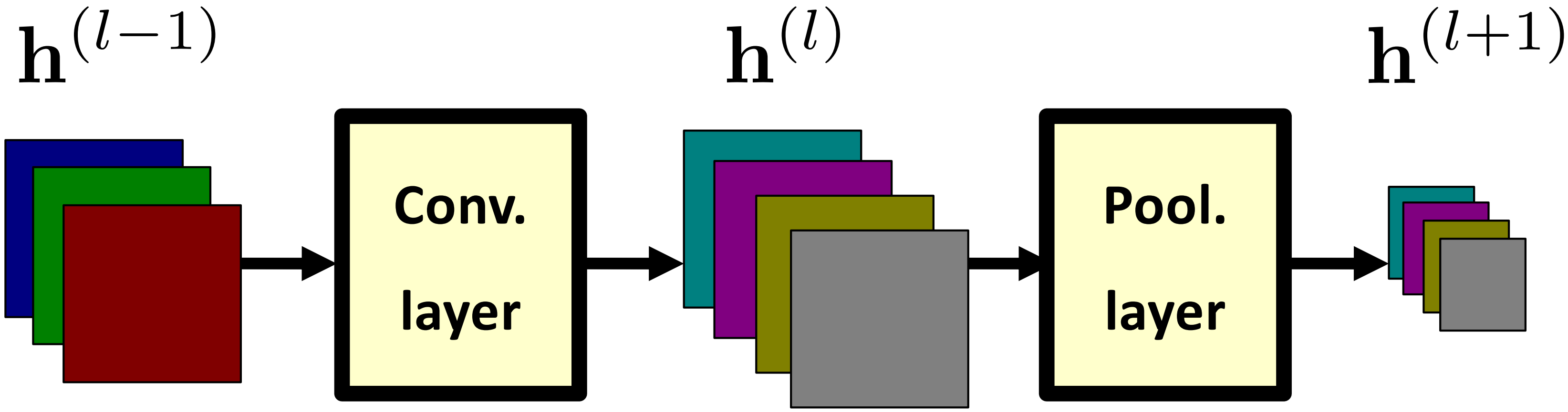


Pooling Layer: Receptive Field Size

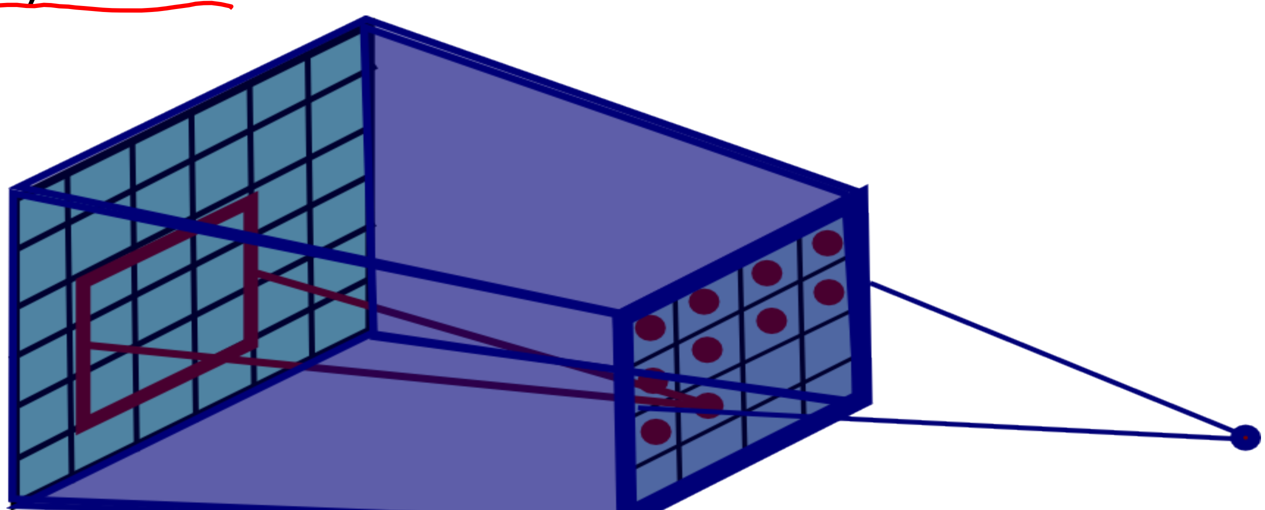




Pooling Layer: Receptive Field Size



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

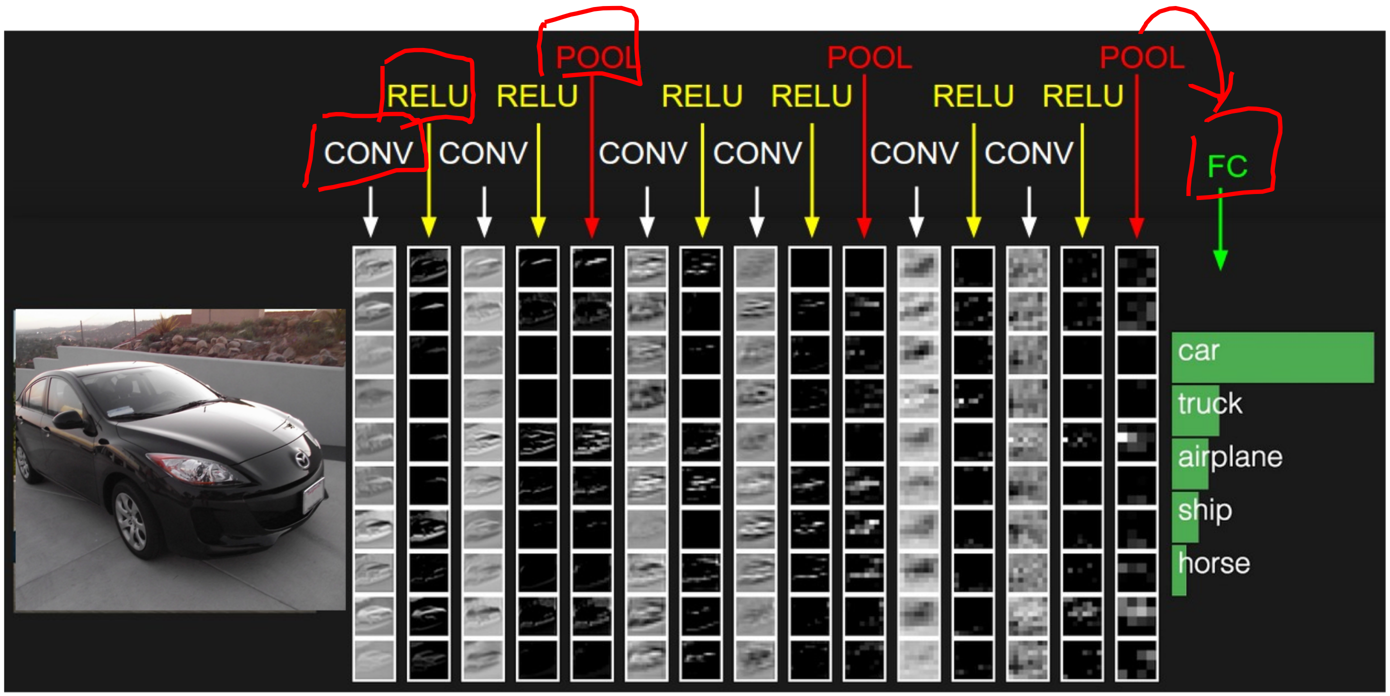
Common settings:

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

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

Fully Connected Layer (FC layer)

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



Convolutional Neural Networks

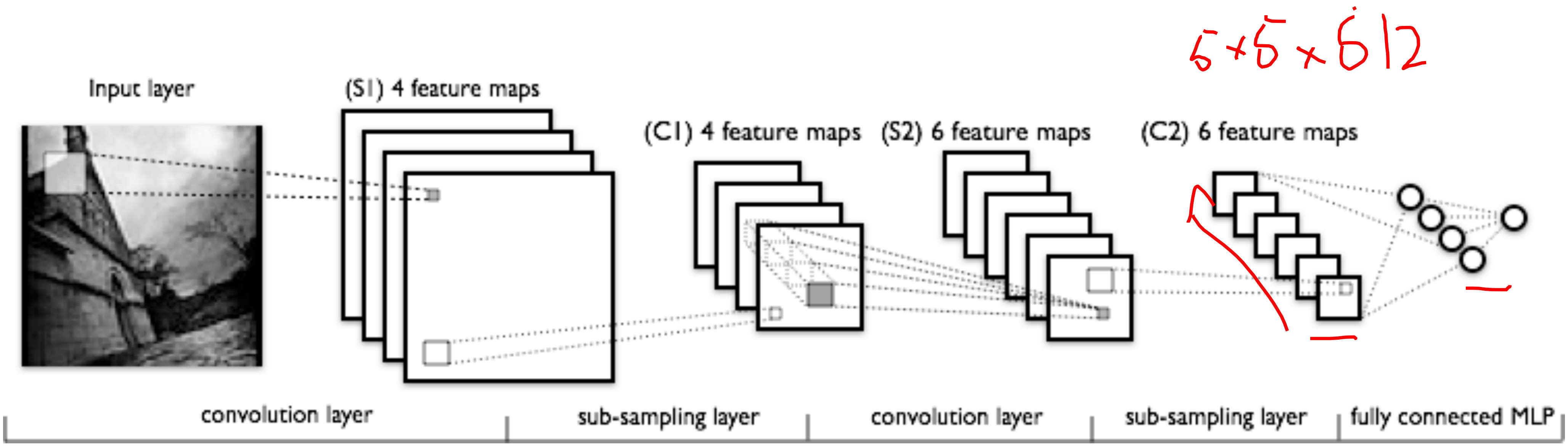


Image Credit: Yann LeCun, Kevin Murphy

Classical View

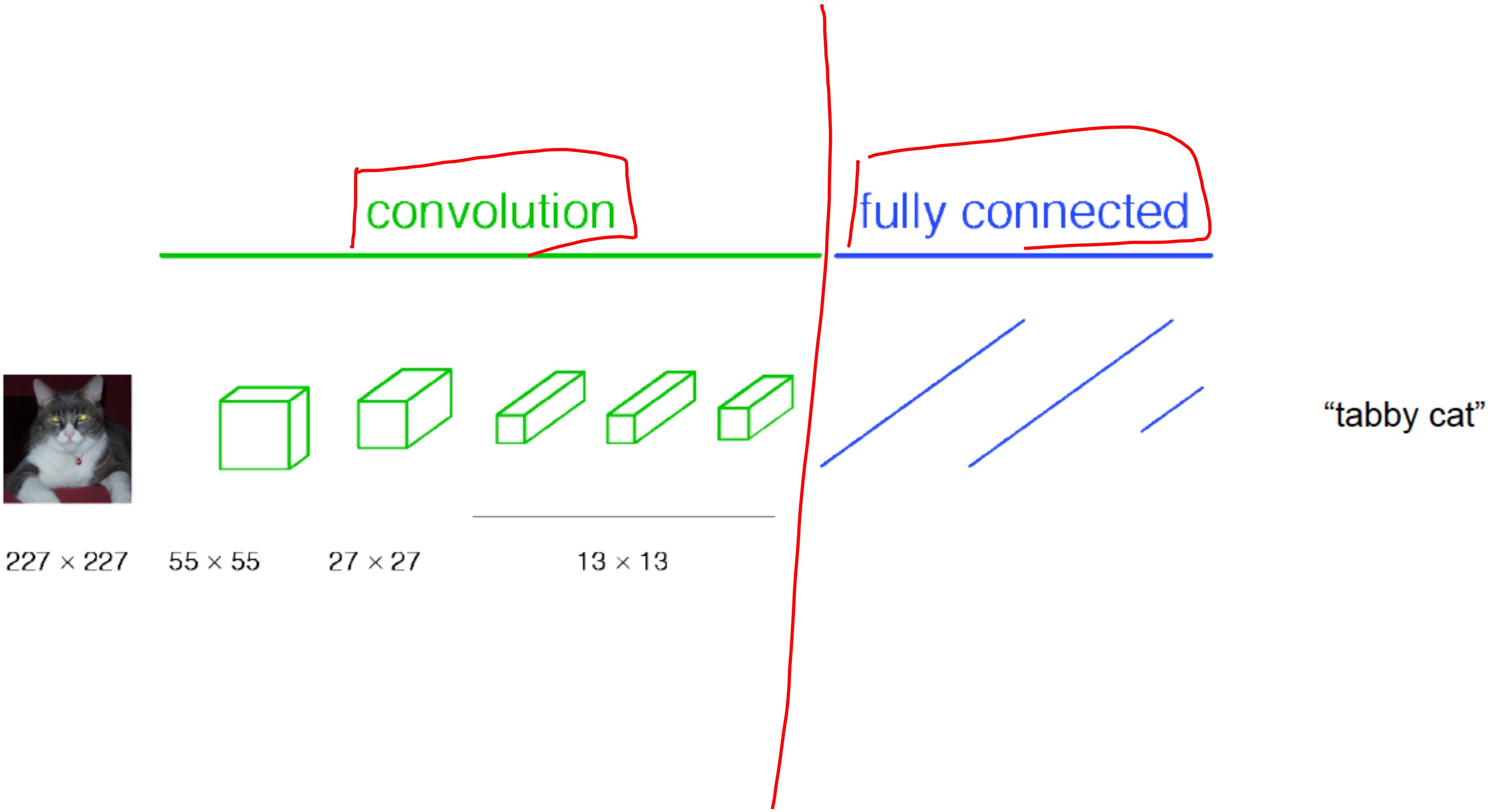
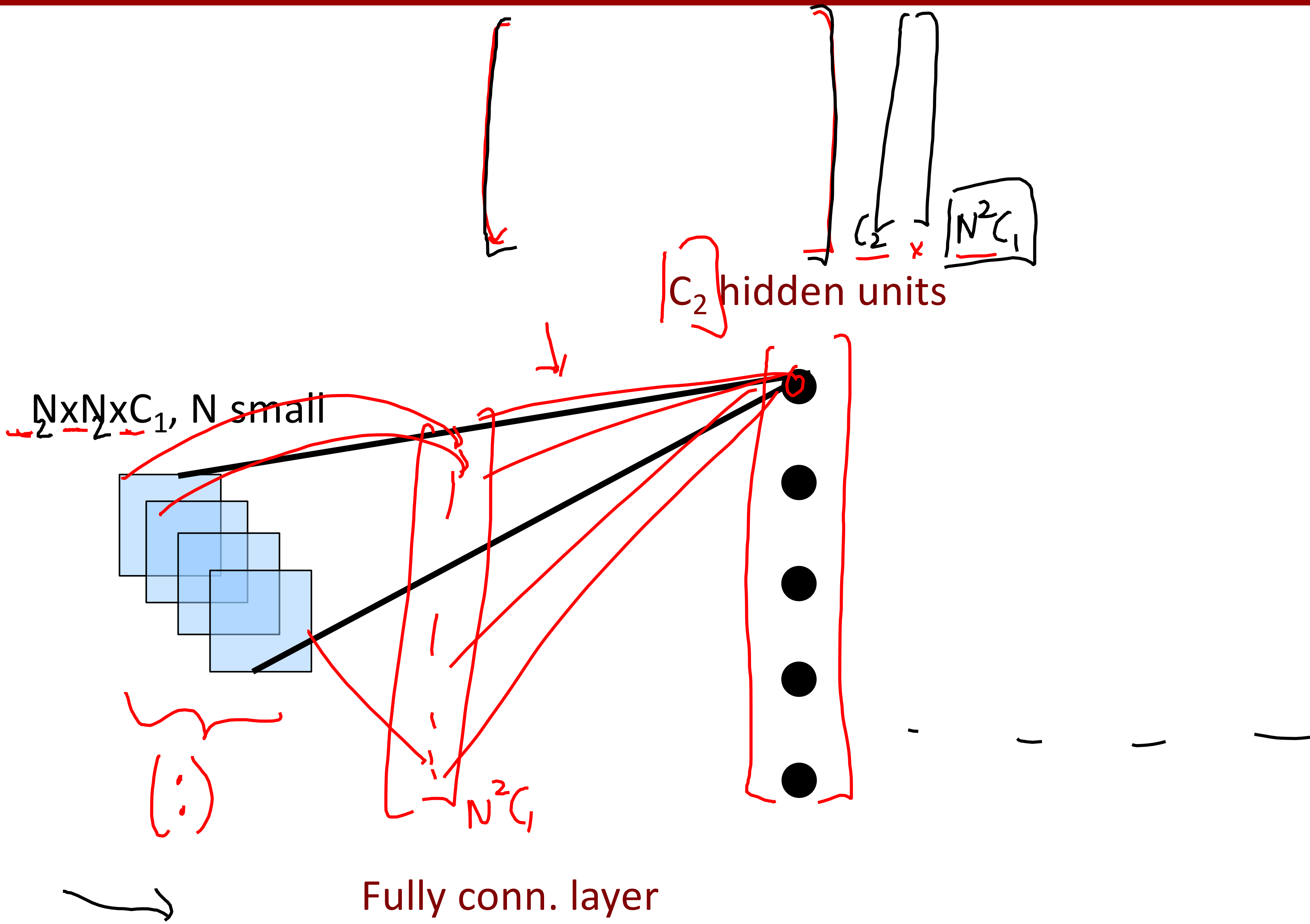


Figure Credit: [Long, Shelhamer, Darrell CVPR15]


$C_2 \times N^2C_1$
C_2 hidden units
NxNxC1, N small
N^2C_1
Fully conn. layer

Classical View

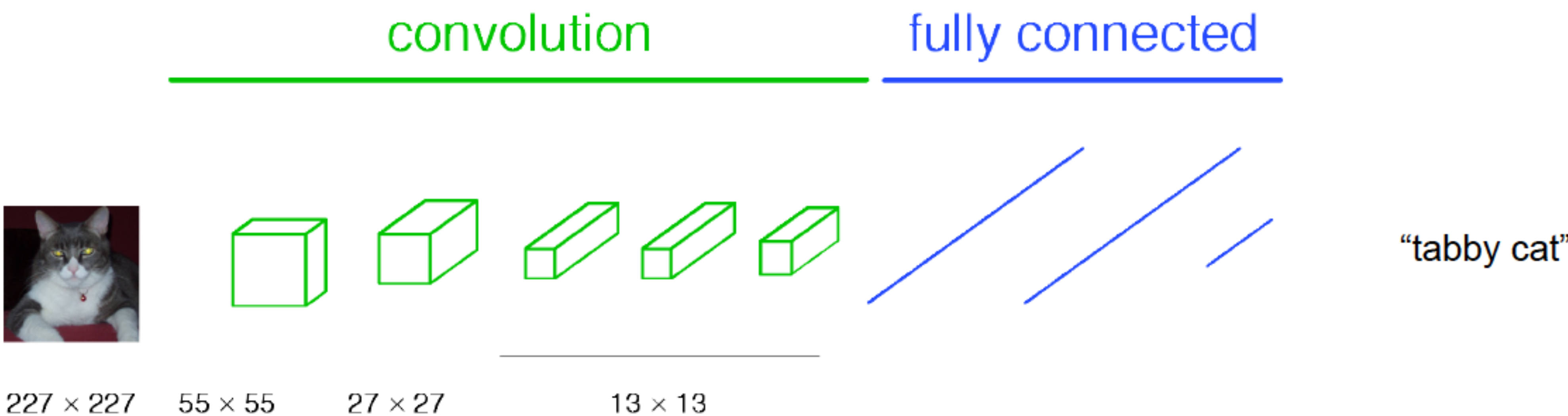
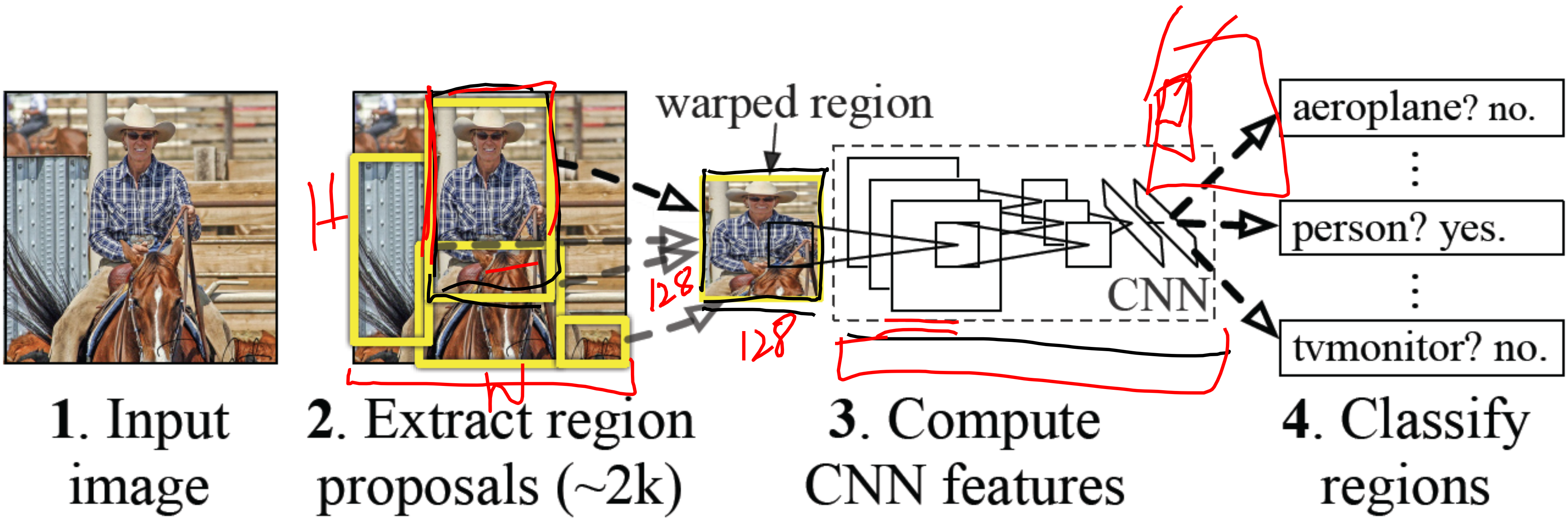


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Classical View = Inefficient



Classical View

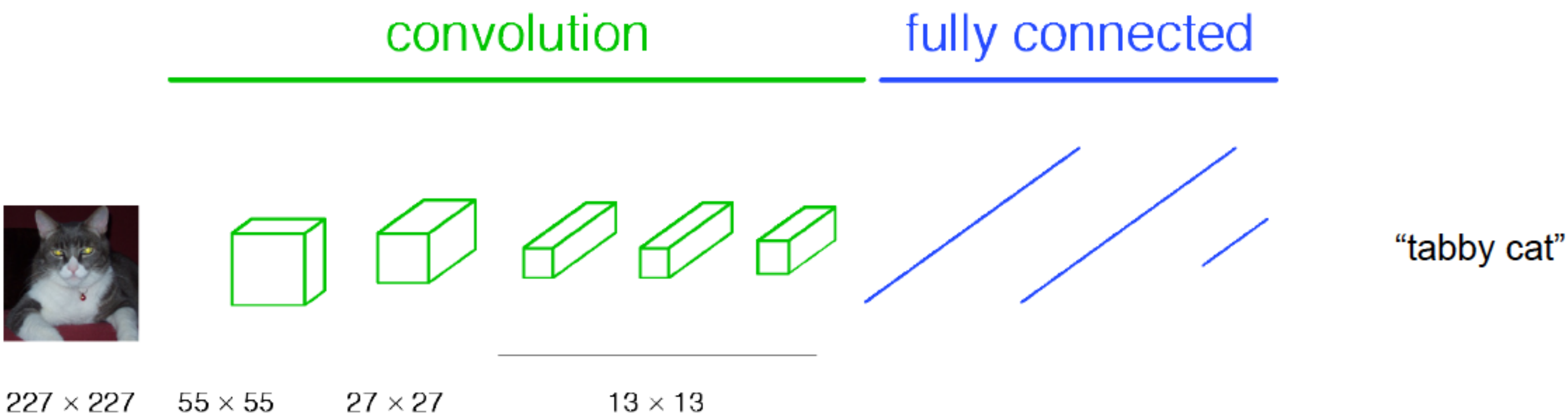
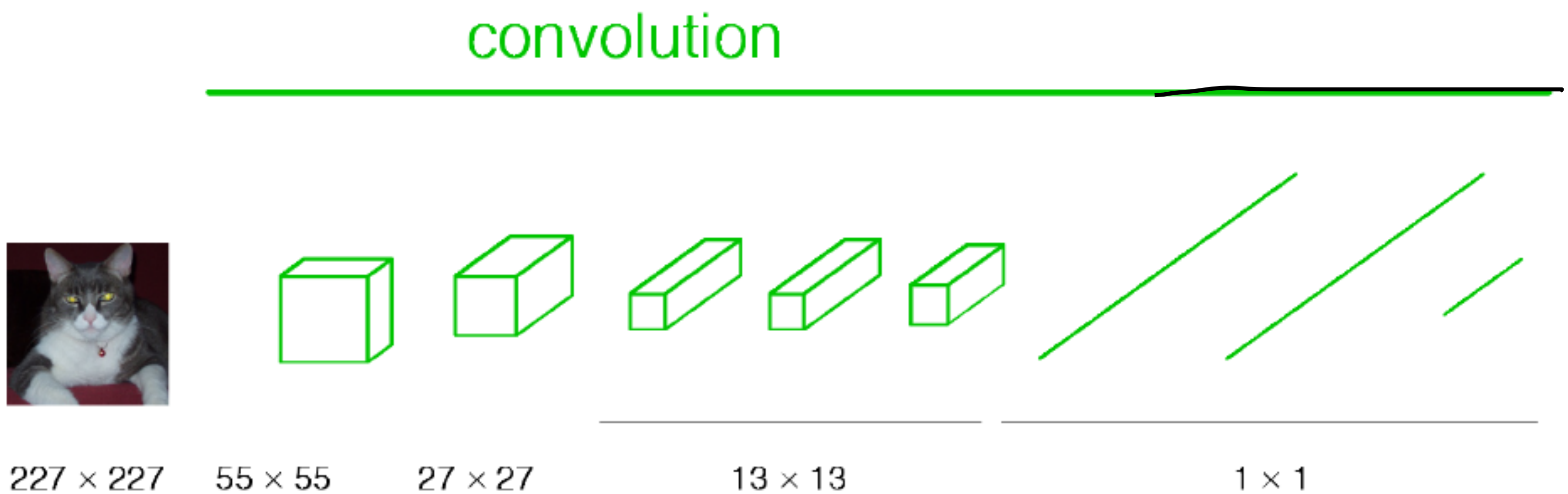
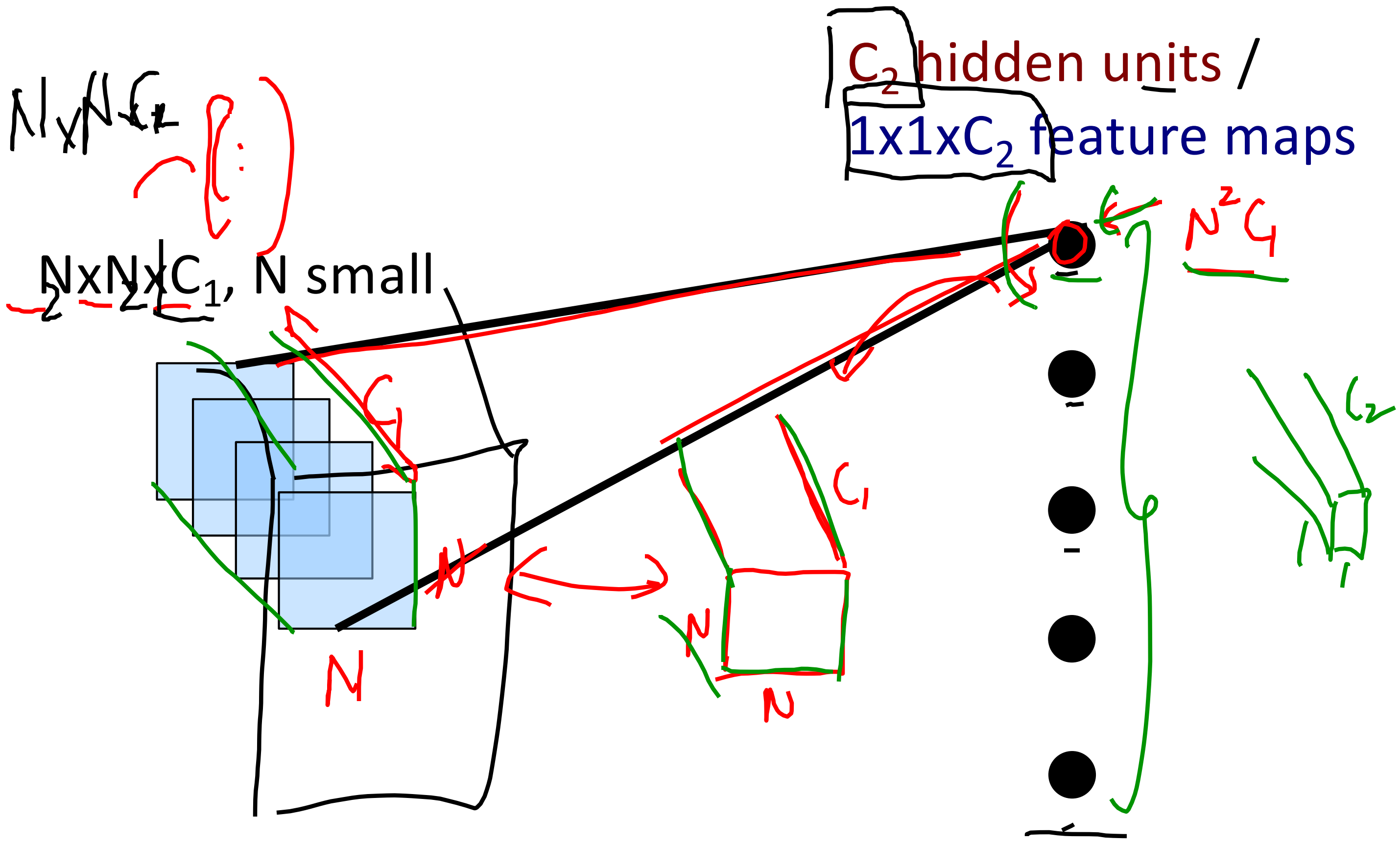


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

- Just squint a little!





Fully conn. layer /

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

Re-interpretation

- Just squint a little!

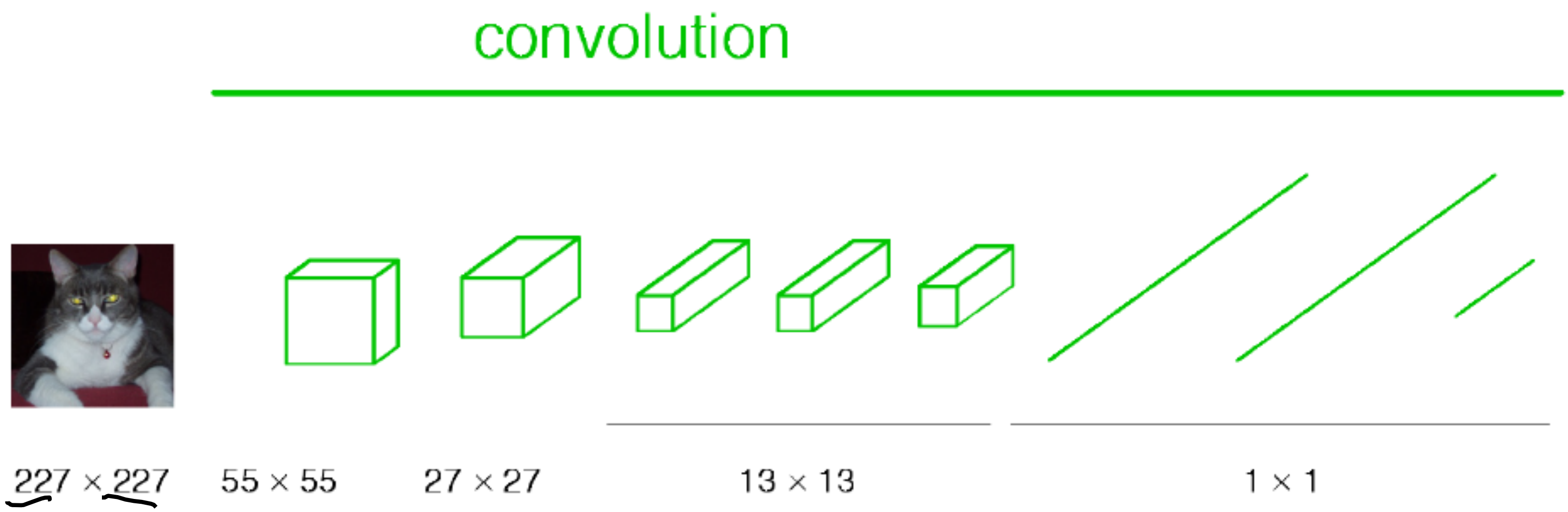
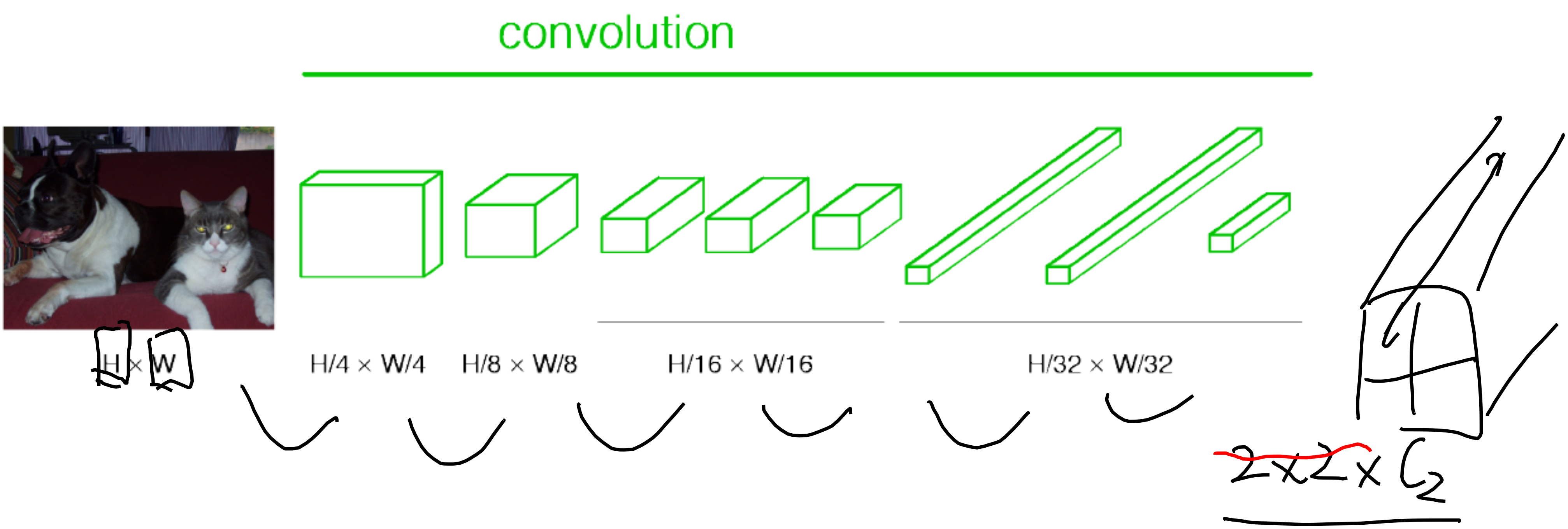


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

"Fully Convolutional" Networks

- Can run on an image of any size!



Benefit of this thinking

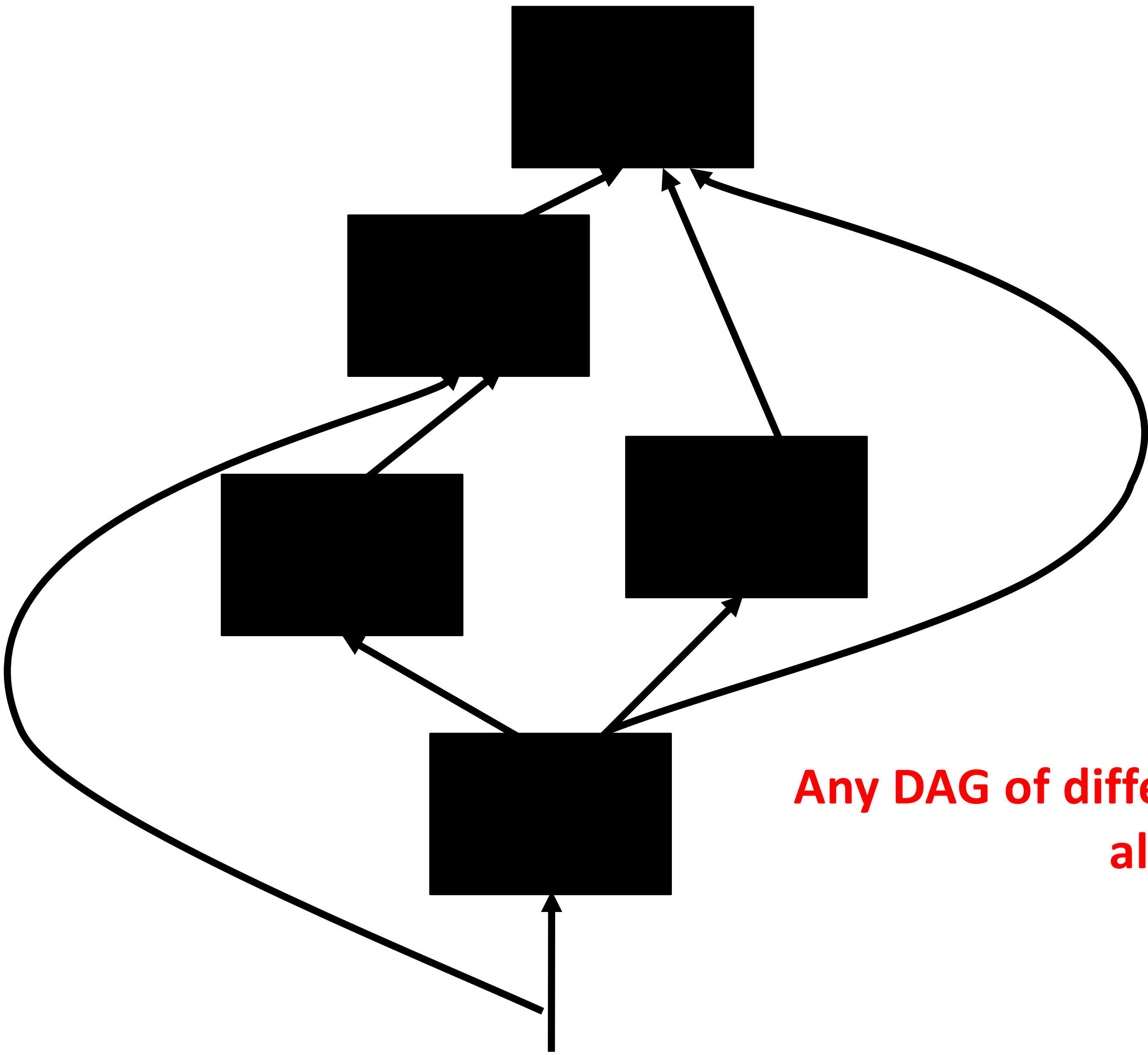
- Mathematically elegant

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

Plan for Today

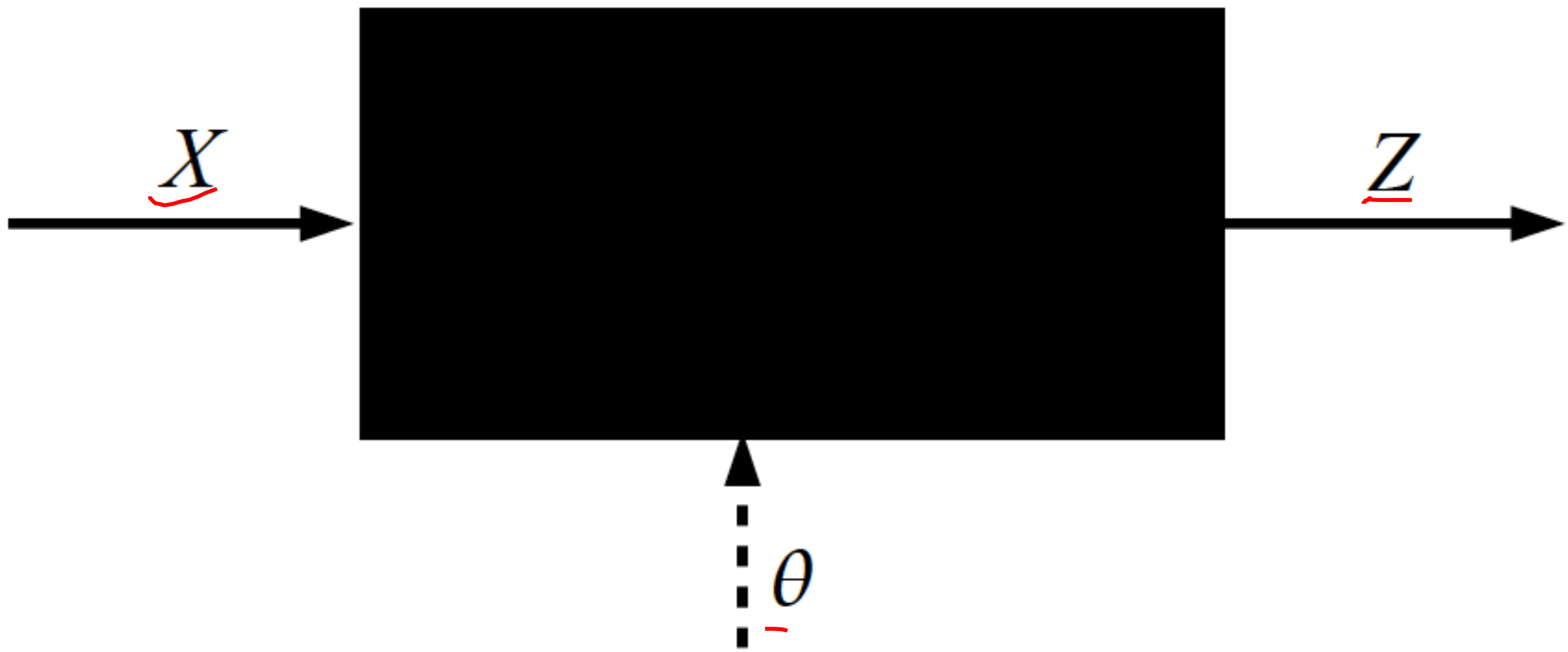
- Convolutional Neural Networks
  - Pooling layers
  - Fully-connected layers as convolutions
  - Backprop in conv layers [Derived in notes]
  - Toeplitz matrices and convolutions = matrix-mult

Computational Graph



Any DAG of differentiable modules is allowed!

Key Computation: Forward-Prop



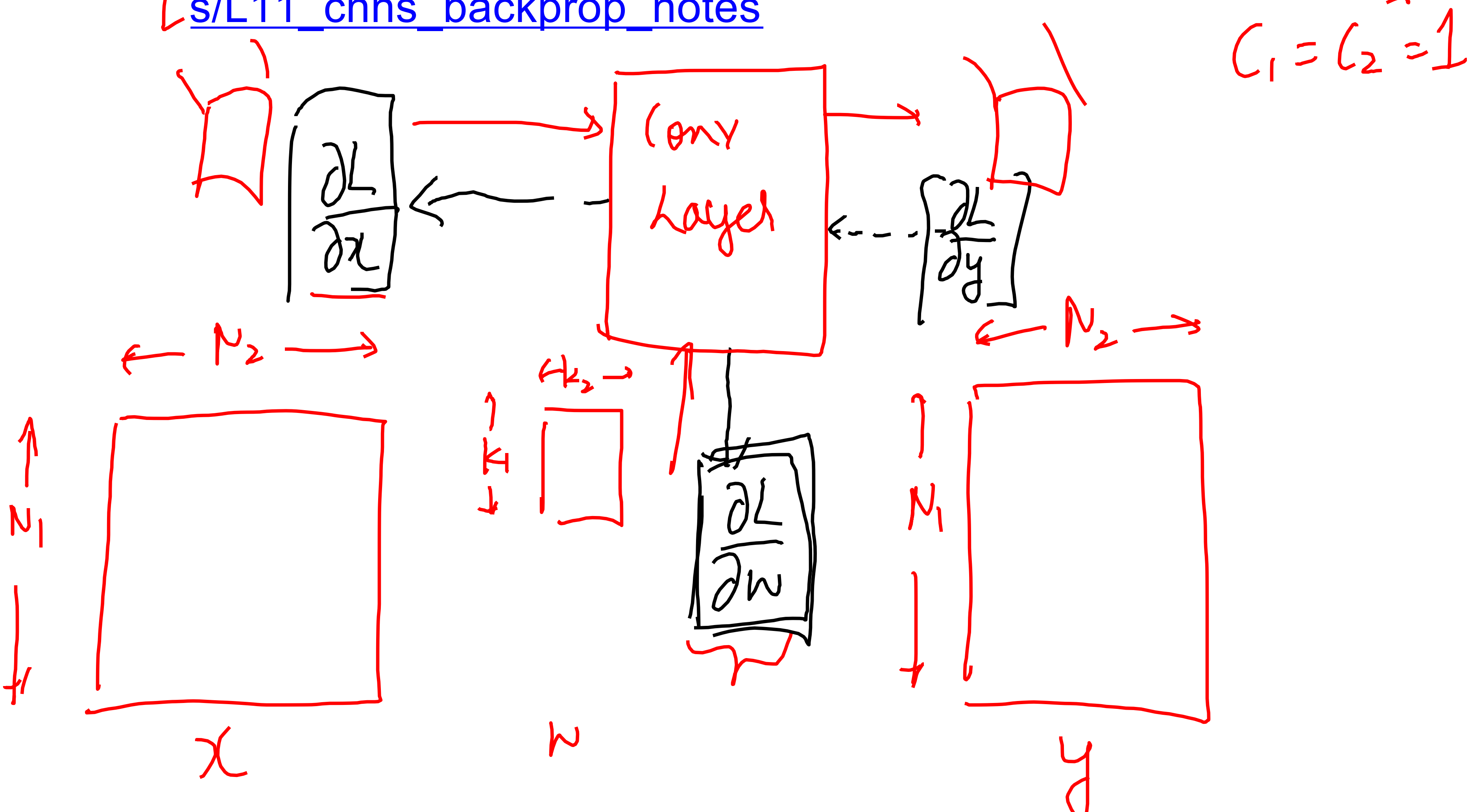
Key Computation: Back-Prop

$$\frac{\partial L}{\partial X} \longleftarrow \left\{ \frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta} \right\} \longleftarrow \frac{\partial L}{\partial Z}$$

$$\frac{\partial L}{\partial \theta}$$

Backprop in Convolutional Layers

- Notes
  - https://www.cc.gatech.edu/classes/AY2020/cs7643_fall/slides/L11_cnns_backprop_notes



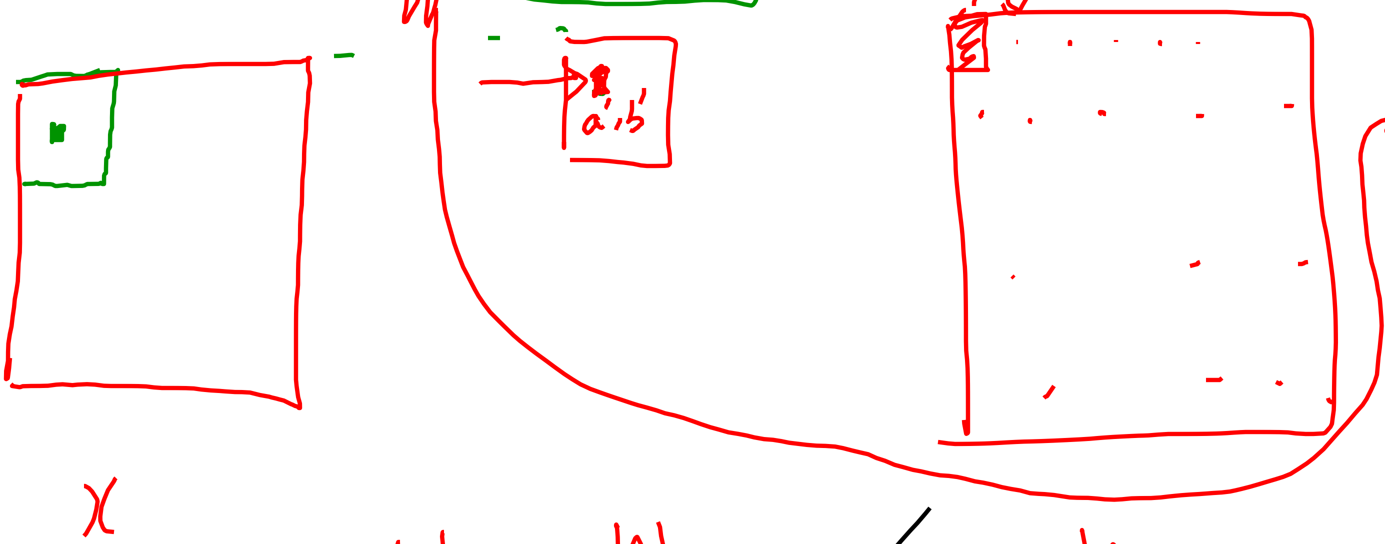
Backprop in Convolutional Layers

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

(1) (2) Input Jacobian

$$y[0,0] = \sum_a \sum_b x[0+a, 0+b]\, w[a,b]$$

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



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

Backprop in Convolutional Layers

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

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

Backprop in Convolutional Layers

Plan for Today

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

Toeplitz Matrix

- Diagonals are constants

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

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

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

Why do we care?

1D

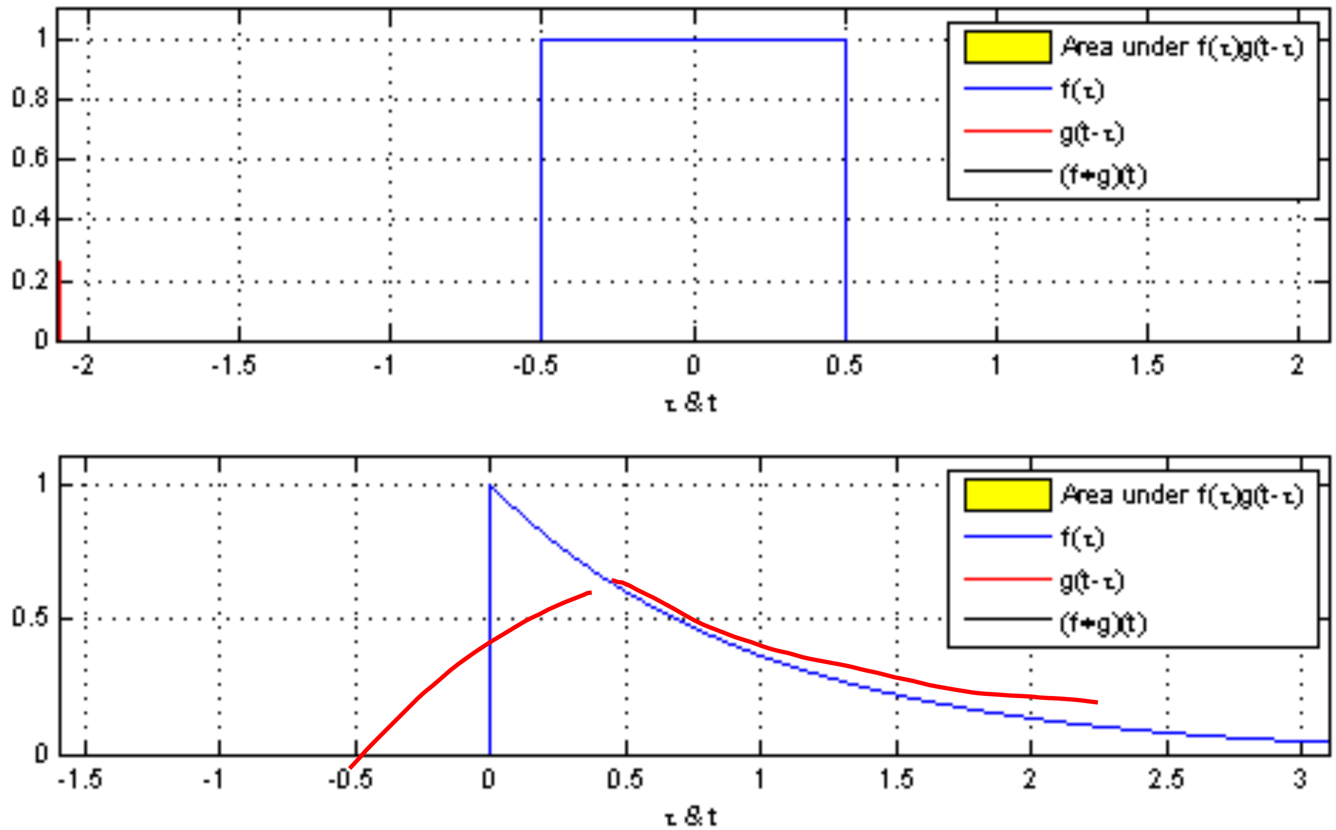
- (Discrete) Convolution = Matrix Multiplication
  - with Toeplitz Matrices

W

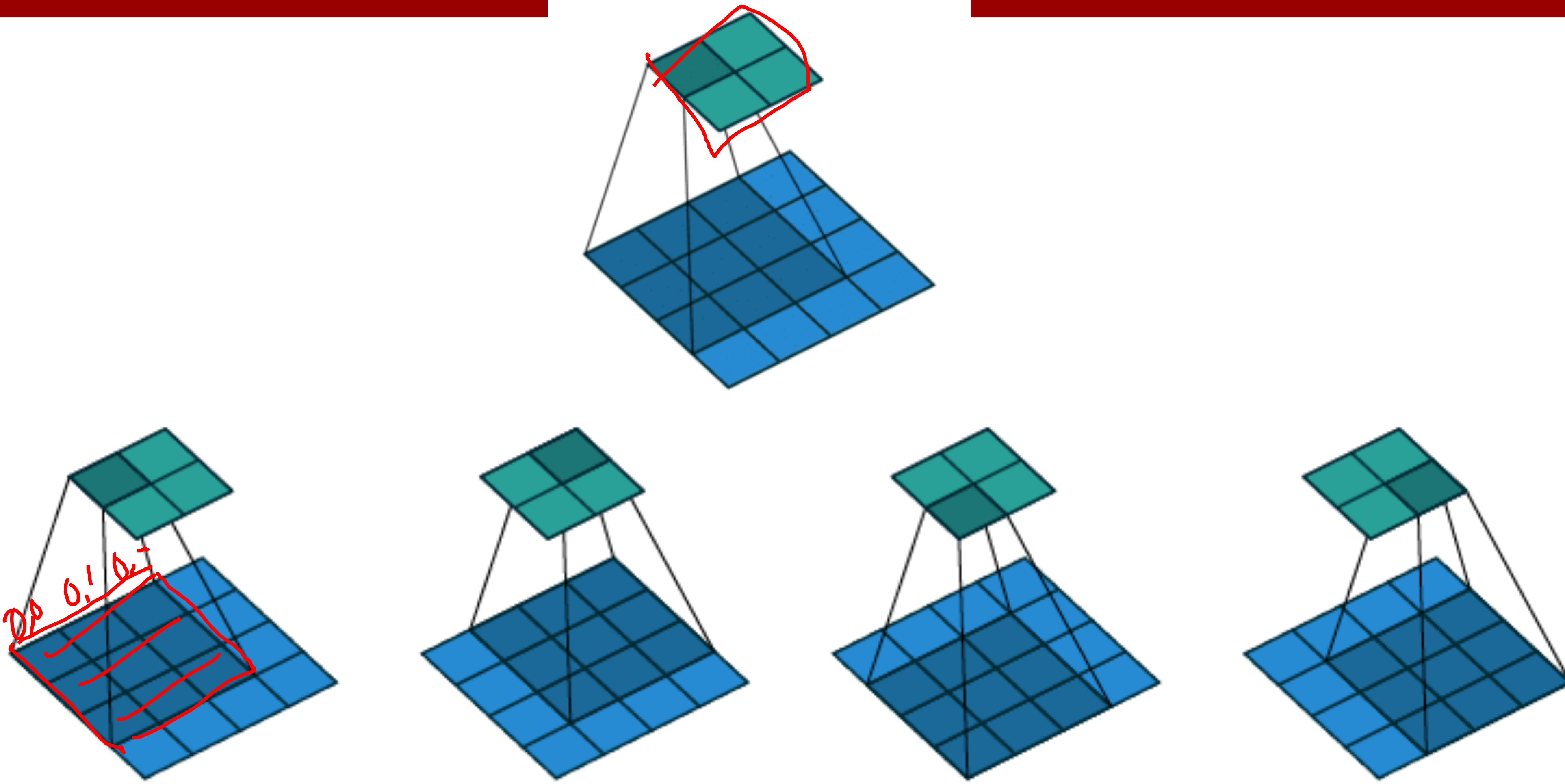
$w_1 --- w_k$

$x(:)$

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



"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



x[0,0]

$$\text{y-vec} = \underbrace{\begin{pmatrix}
w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\
0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\
0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2}
\end{pmatrix}}_{W} \overset{x(:)}{\begin{bmatrix} \uparrow \\ \\ \\ \downarrow \end{bmatrix}}$$

So far: Image Classification

This image is CC0 public domain

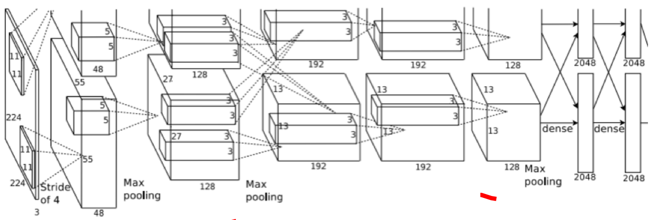


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

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

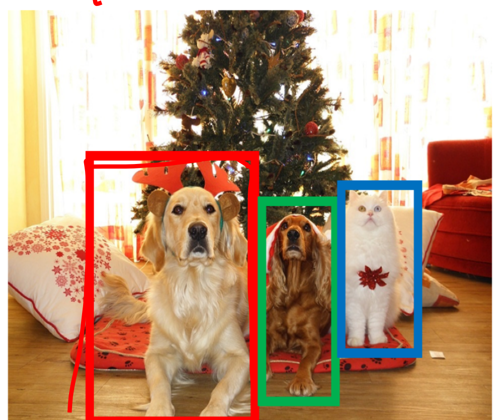
Other Computer Vision Tasks

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

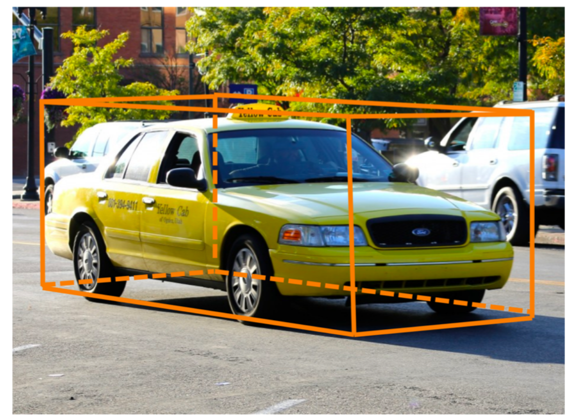
2D Object Detection



DOG, DOG, CAT

Object categories + 2D bounding boxes

3D Object Detection



Car

Object categories + 3D bounding boxes