CS 4803 / 7643: Deep Learning

Topics:

- (Finish) Convolutional Neural Networks
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
  - Transposed convolutions

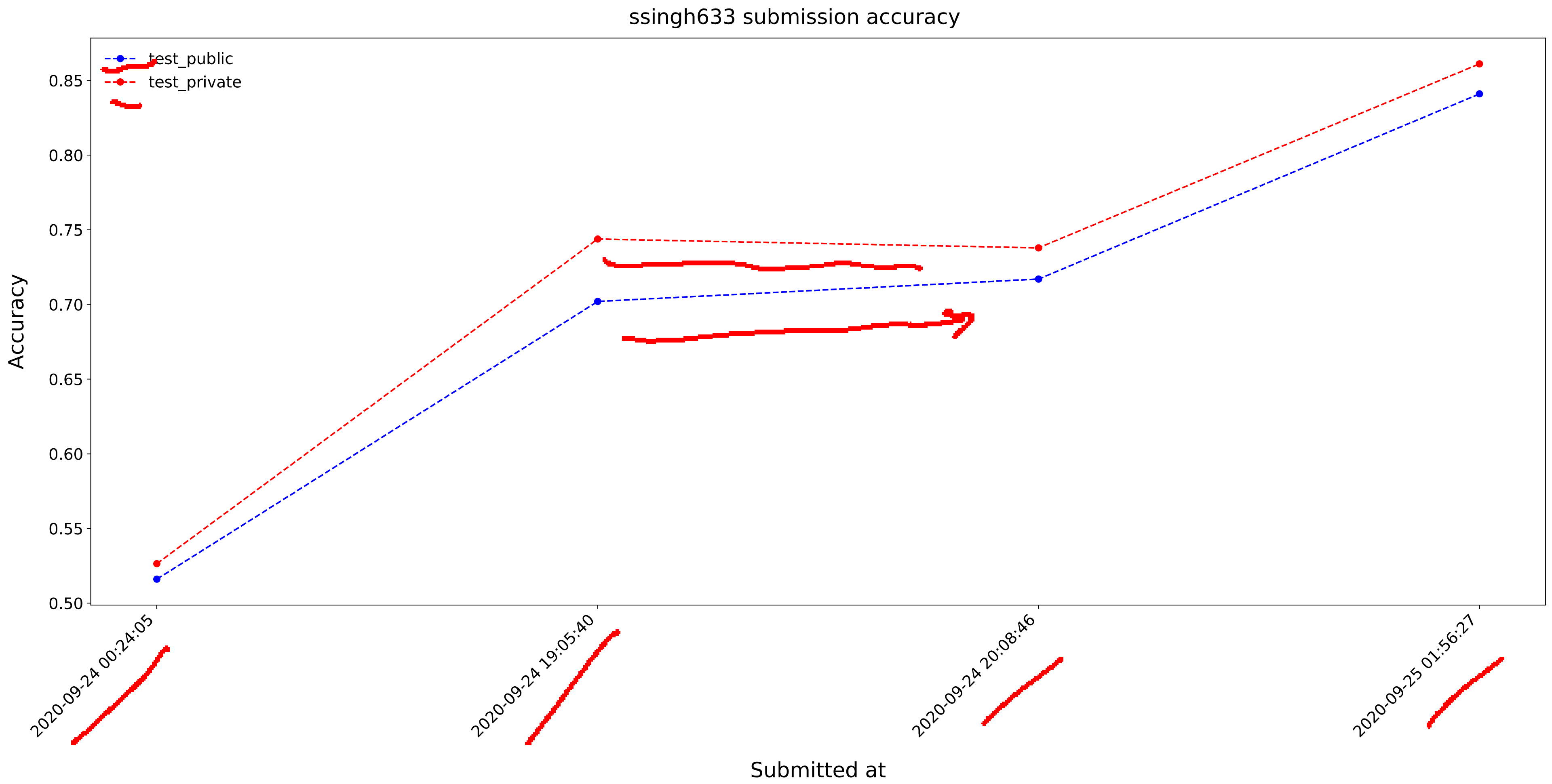
Dhruv Batra

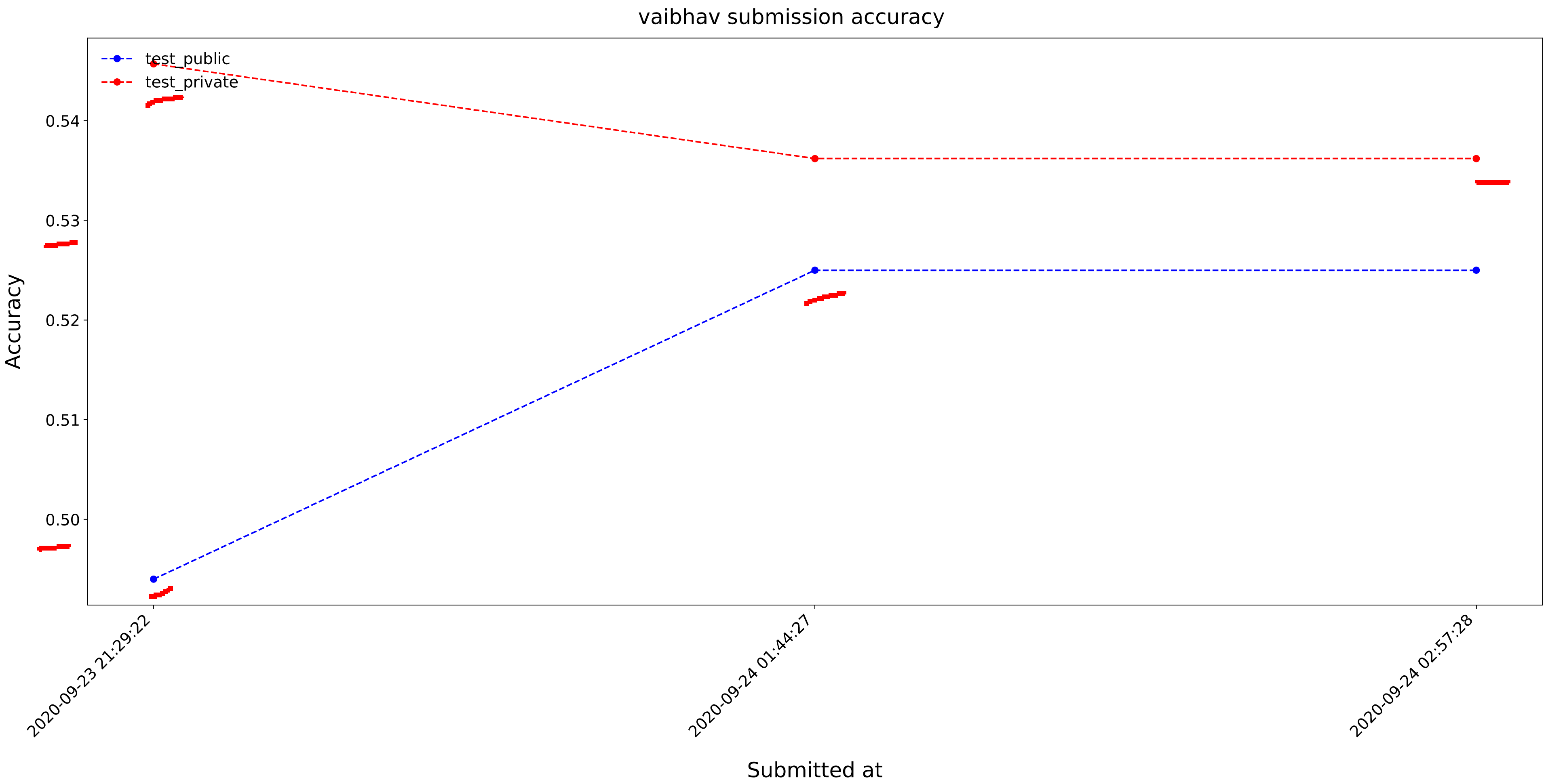
Georgia Tech

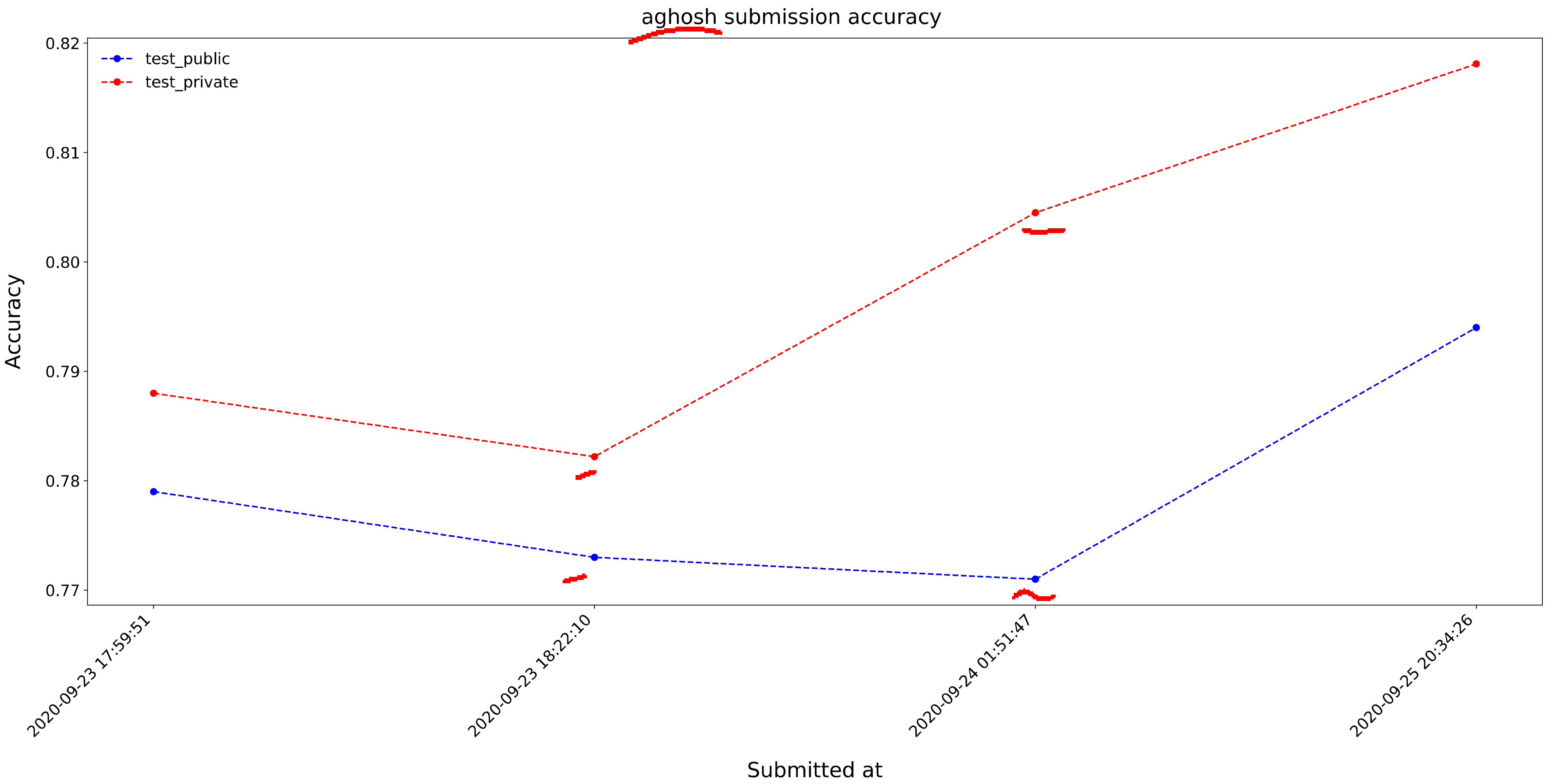
Administrativia

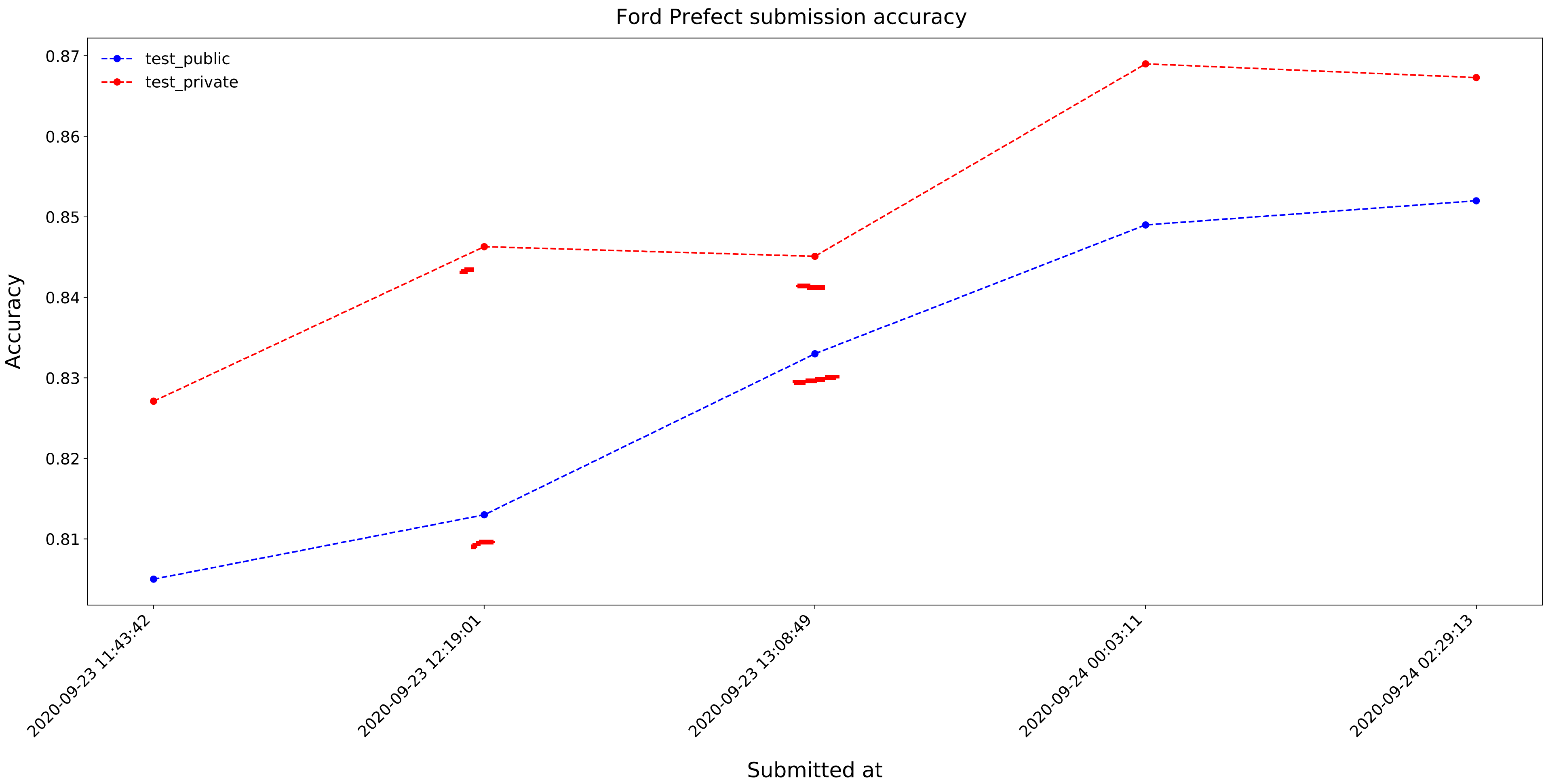
- HW2 Challenge Final Analysis
  - https://evalai.cloudcv.org/web/challenges/challenge-page/684/leaderboard/1853
  - Qualitative Trends

- HW3 Reminder
  - Due: 10/07 11:59pm
  - Theory: Convolutions, Representation Capacity, Double Descent
  - Implementation: Saliency methods (e.g. Grad-CAM) in Python and PyTorch/Captum


ssingh633 submission accuracy
test_public
test_private
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
2020-09-24 00:24:05
2020-09-24 19:05:40
2020-09-24 20:08:46
2020-09-25 01:56:27
Submitted at


vaibhav submission accuracy
test_public
test_private
0.54
0.53
0.52
0.51
0.50
Accuracy
2020-09-23 21:29:22
2020-09-24 01:44:27
2020-09-24 02:57:28
Submitted at


aghosh submission accuracy
0.82
0.81
0.80
0.79
0.78
0.77
Accuracy
test_public
test_private
2020-09-23 17:59:51
2020-09-23 18:22:10
2020-09-24 01:51:47
2020-09-25 20:34:26
Submitted at

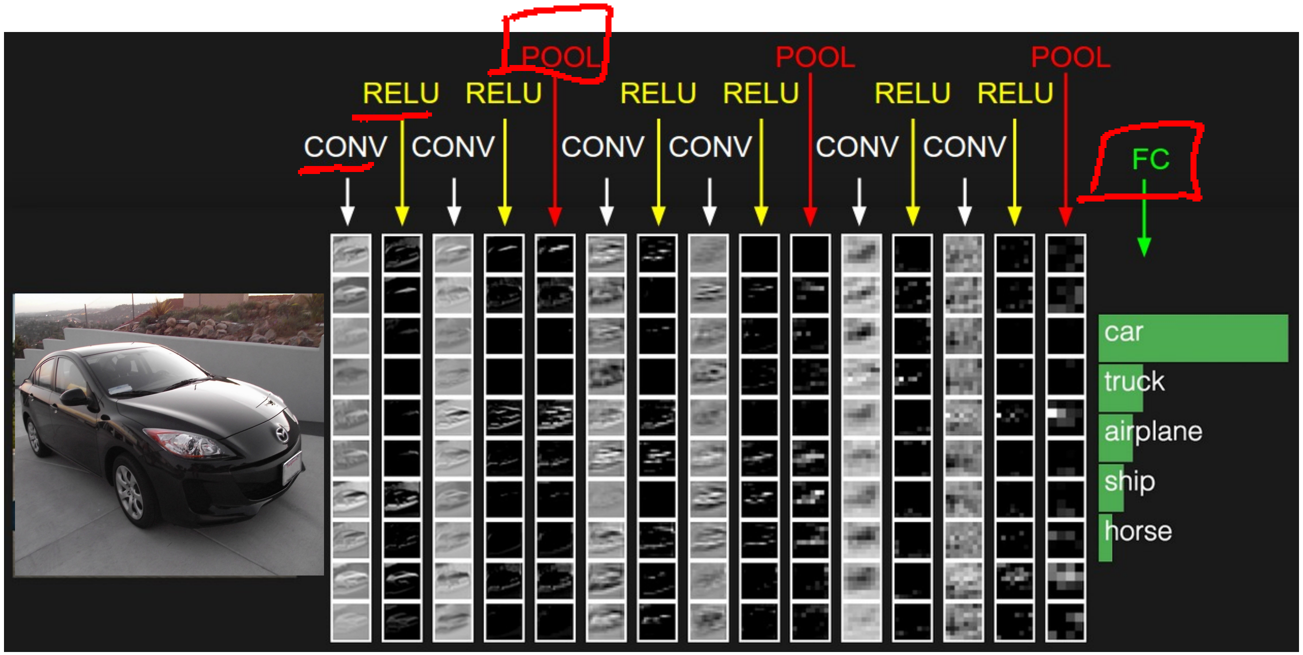

Ford Prefect submission accuracy
test_public
test_private
Accuracy
0.87
0.86
0.85
0.84
0.83
0.82
0.81
Submitted at
2020-09-23 11:43:42
2020-09-23 12:19:01
2020-09-23 13:08:49
2020-09-24 00:03:11
2020-09-24 02:29:13

Plan for Today

- (Finish) Convolutional Neural Networks
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
  - Transposed convolutions

Fully Connected Layer (FC layer)

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



Classical View

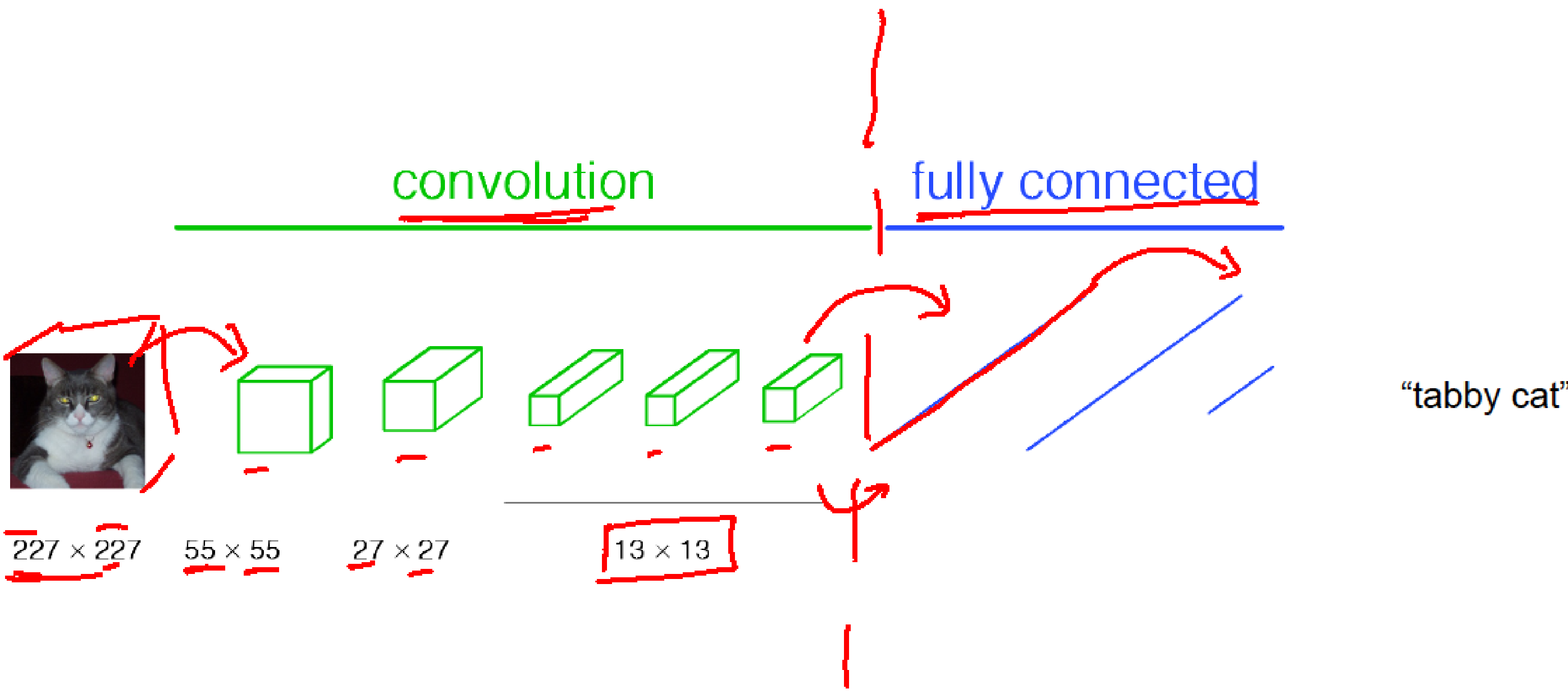


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

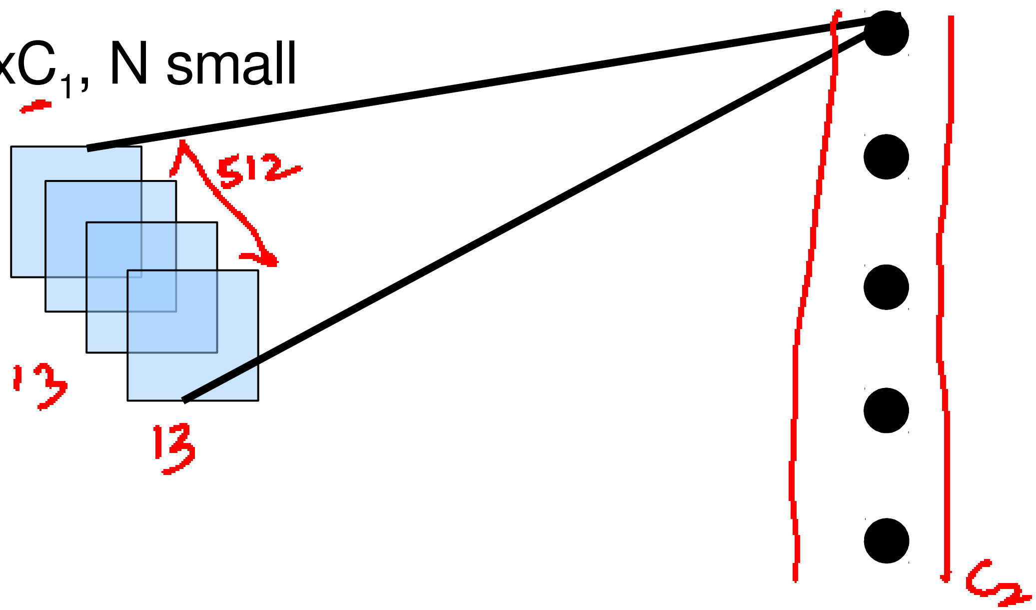
$h^{\ell-1}$-vec

$H^{\ell-1}(:)$

$W = \left[\longleftarrow \quad \longrightarrow \right]_{C_2 \times (N^2 C_1)}$

C_2 hidden units

NxNxC_1, N small



$h^{\ell} = W \left[h^{\ell-1} \right]_{N^2 C_1}$

Not applicable.

Fully conn. layer

Classical View

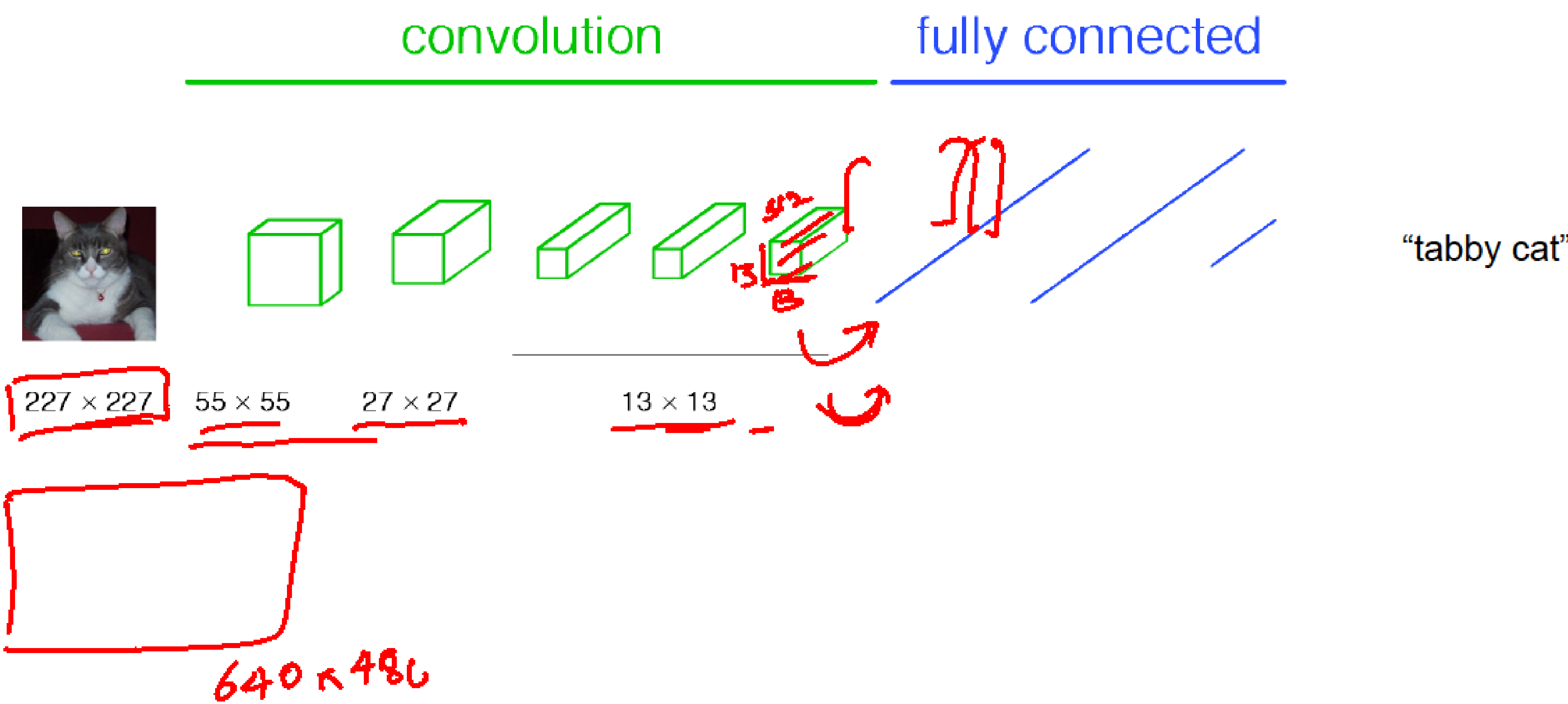
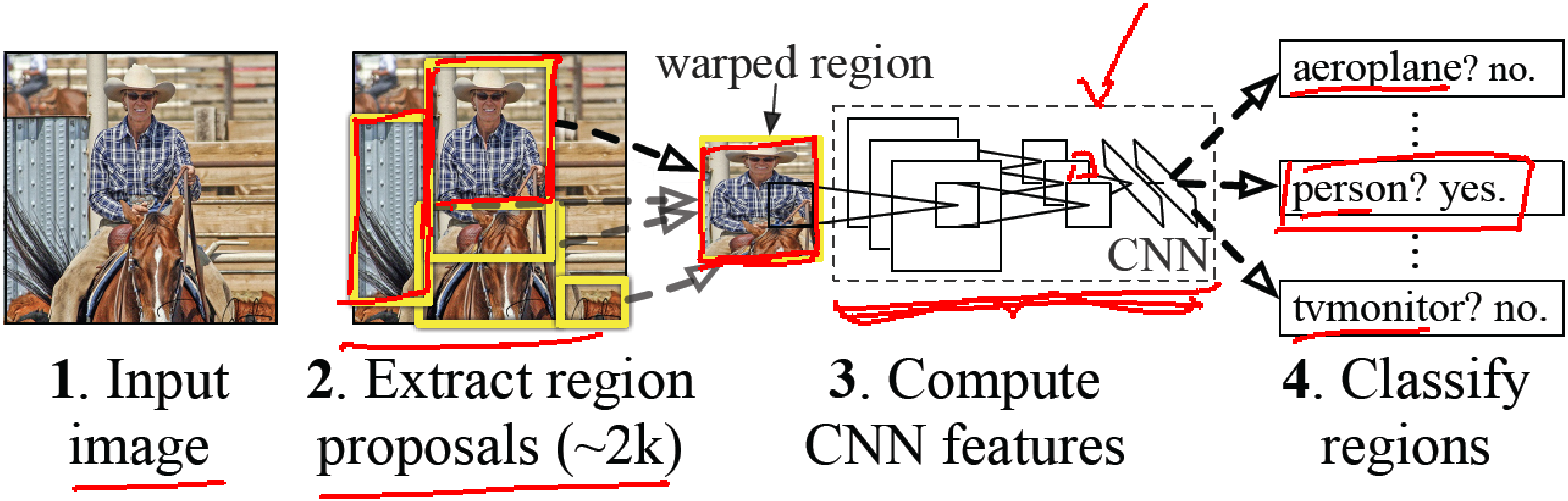


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Classical View = Inefficient



Classical View

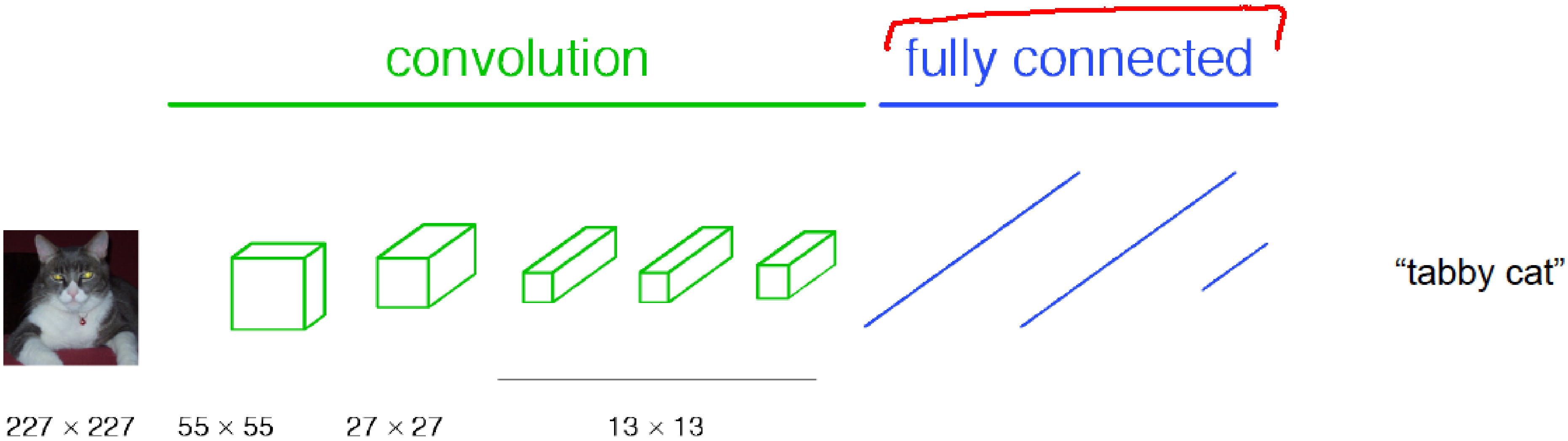


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

Re-interpretation

- Just squint a little!

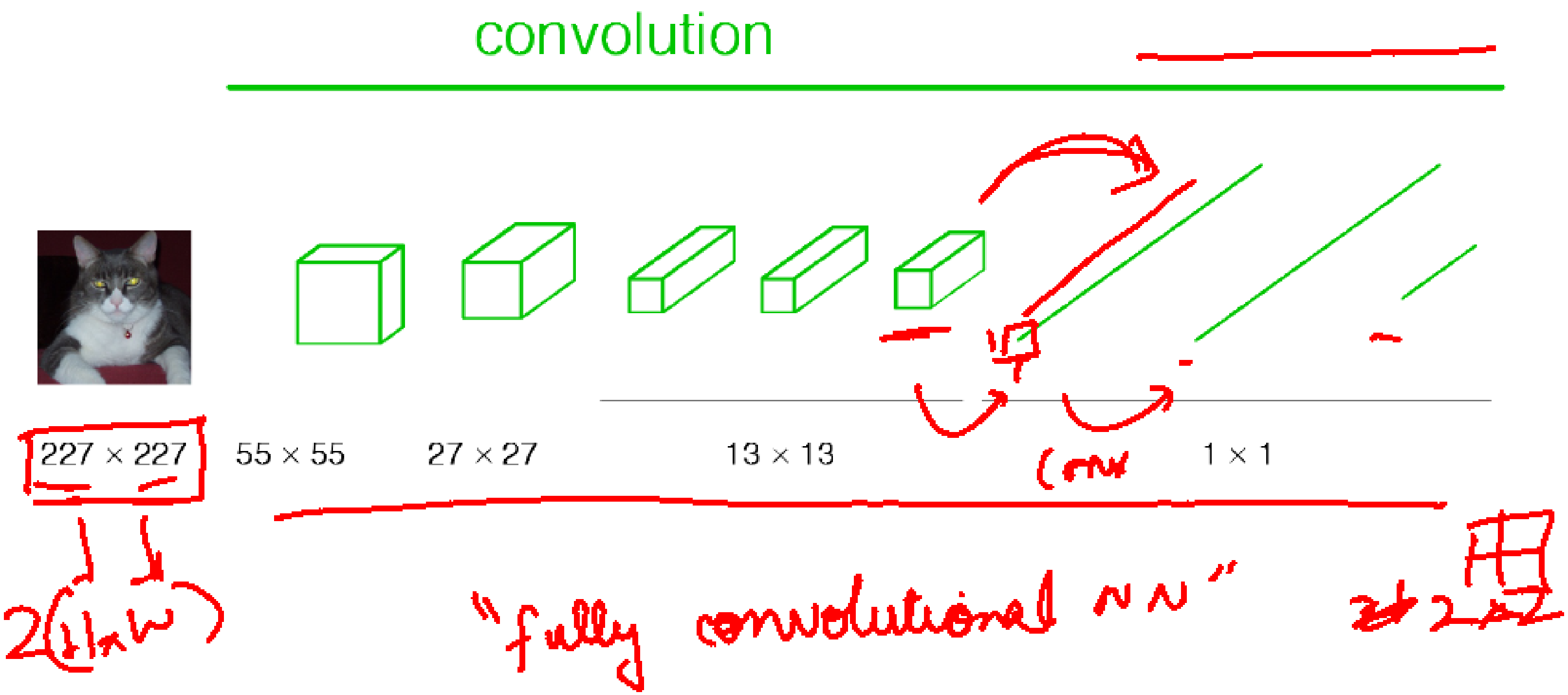
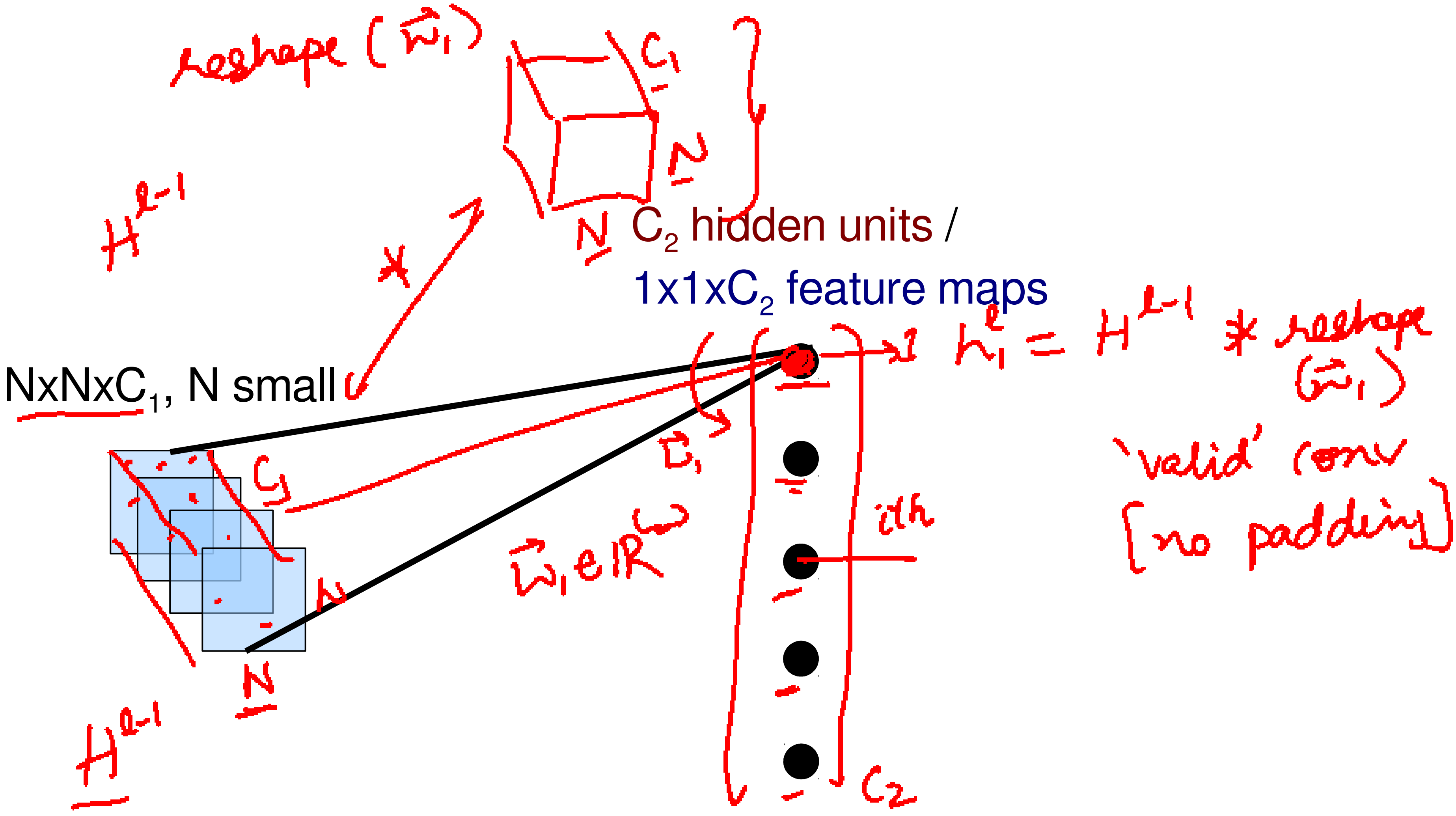


Figure Credit: [Long, Shelhamer, Darrell CVPR15]



Fully conn. layer /

Conv. layer (C_2 kernels of size NxNxC_1)

Re-interpretation

- Just squint a little!

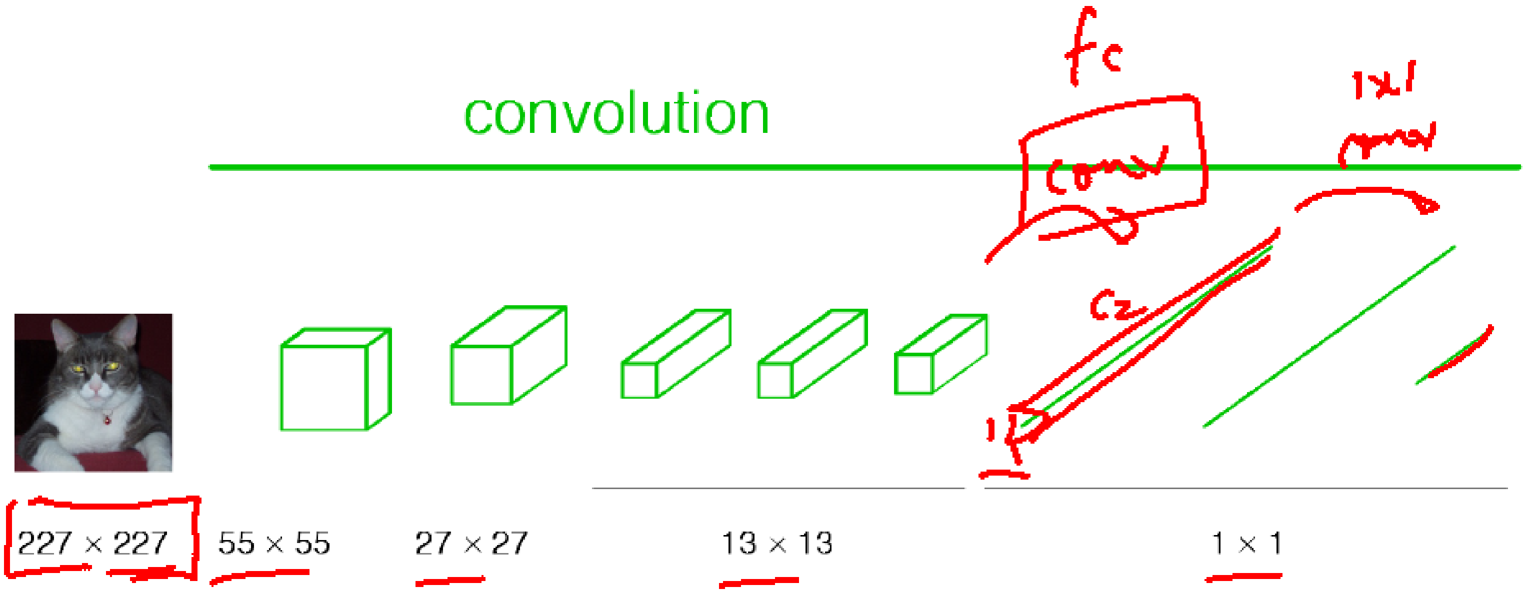
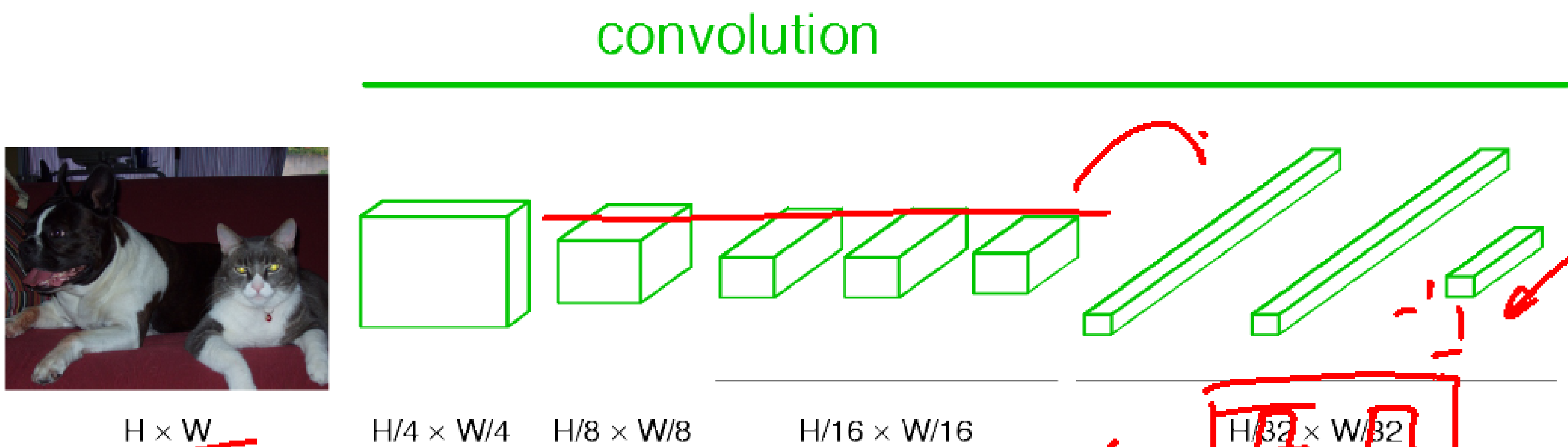


Figure Credit: [Long, Shelhamer, Darrell CVPR15]

"Fully Convolutional" Networks

- Can run on an image of any size!

[at test time]



classes

2x2

Benefit of this thinking

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

Plan for Today

- (Finish) Convolutional Neural Networks
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
  - Transposed convolutions

So far: Image Classification

This image is CC0 public domain

x

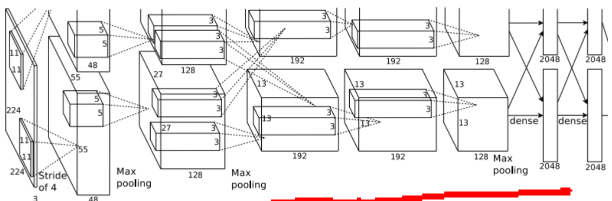


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector: 4096

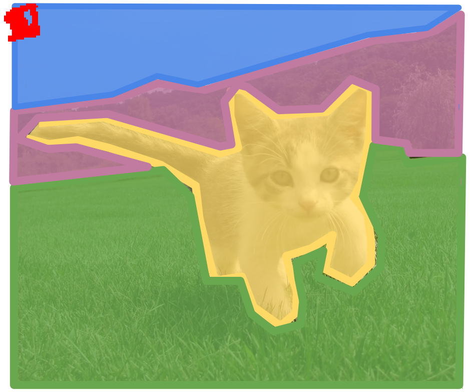
Fully-Connected: 4096 to 1000

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

y

Other Computer Vision Tasks

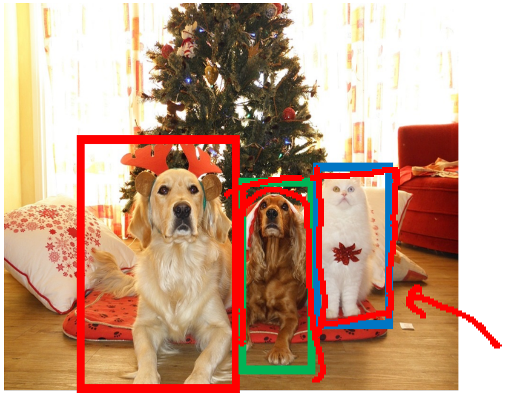
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

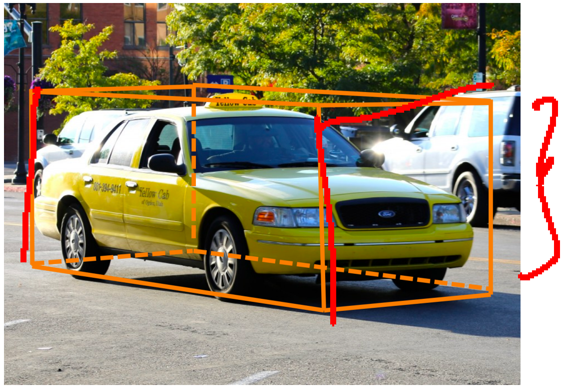
2D Object Detection



DOG, DOG, CAT

Object categories + 2D bounding boxes

3D Object Detection



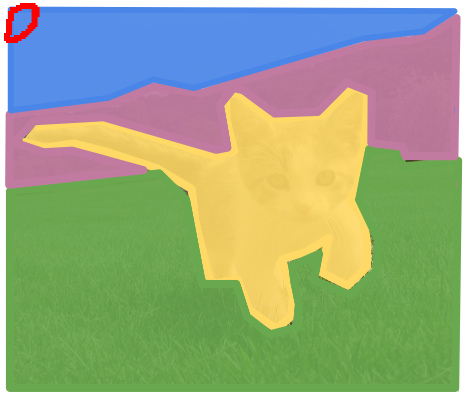
Car

Object categories + 3D bounding boxes

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

Semantic Segmentation

Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

2D Object Detection

DOG, DOG, CAT

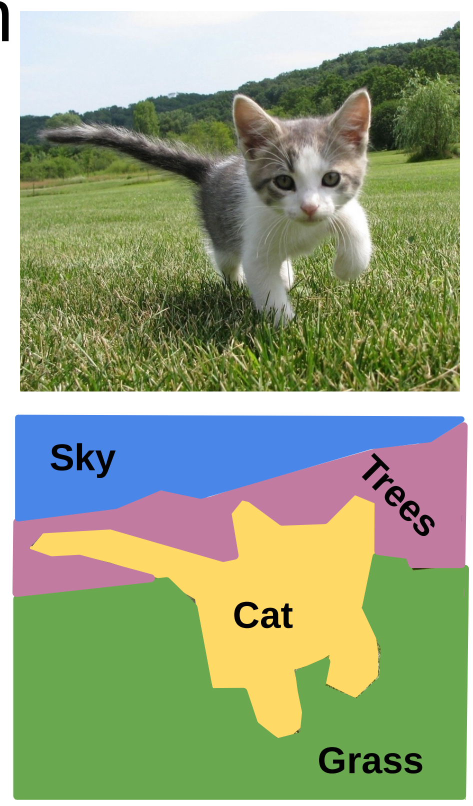
Object categories + 2D bounding boxes

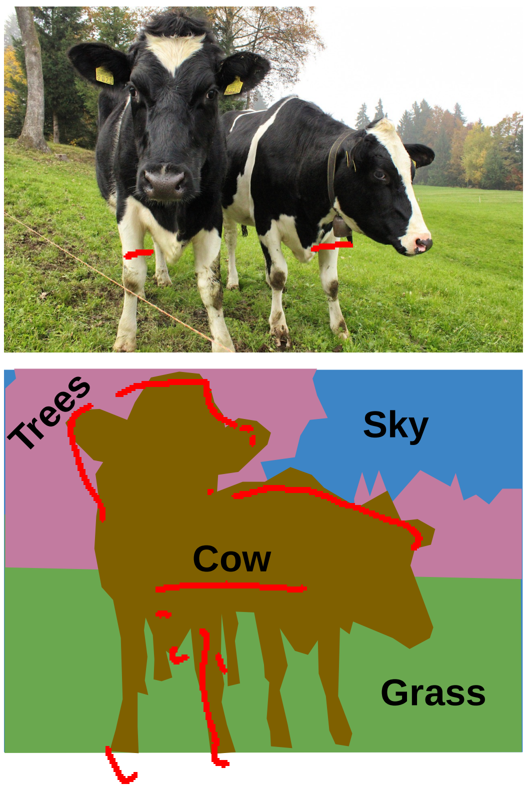
3D Object Detection

Car

Object categories + 3D bounding boxes

Semantic Segmentation

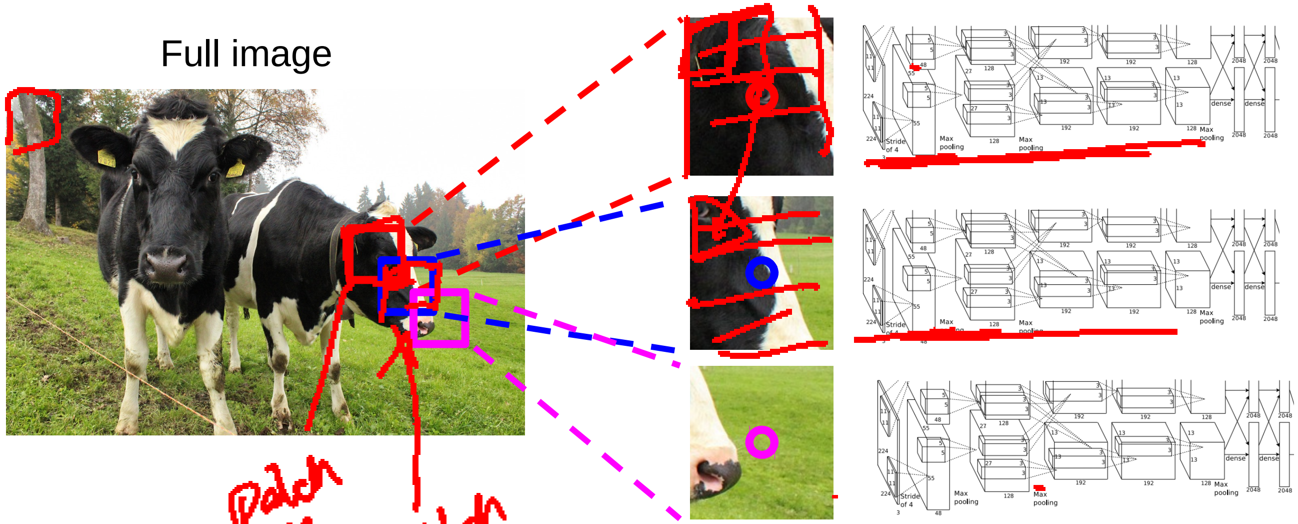




Label each pixel in the image with a category label

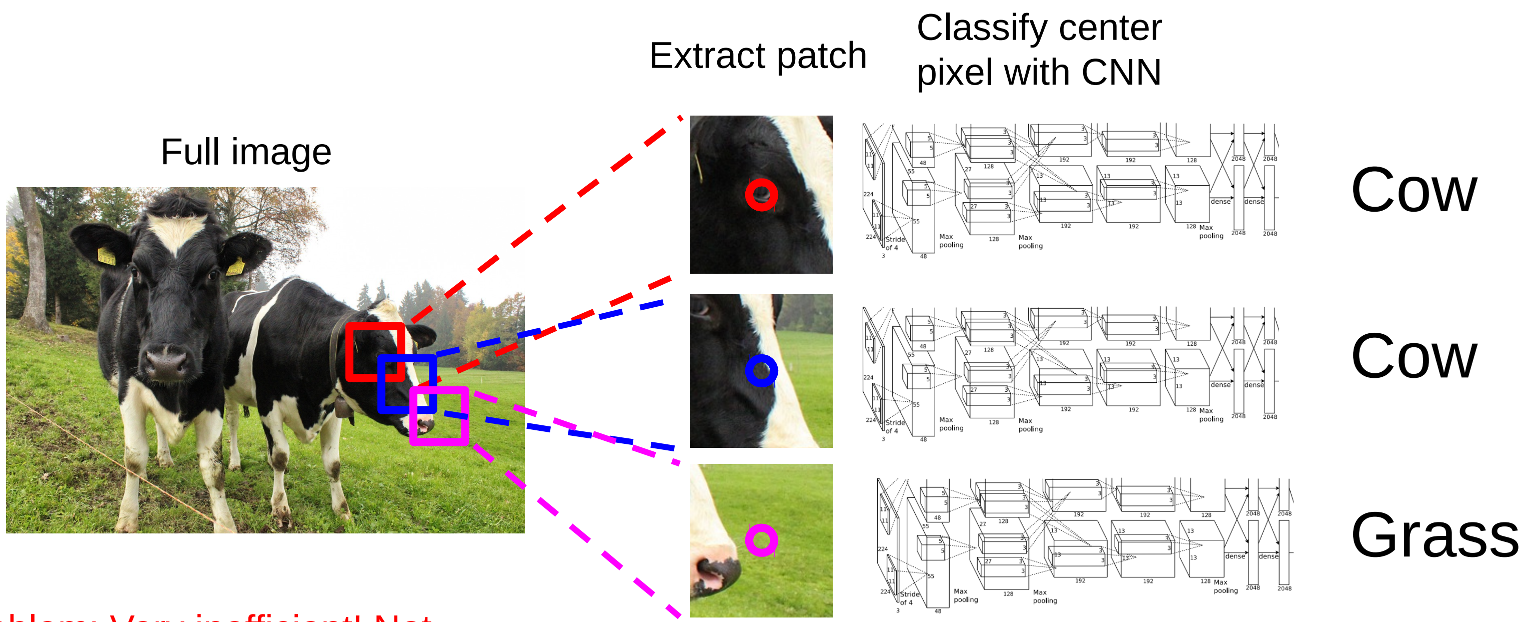
Don’t differentiate instances, only care about pixels

Semantic Segmentation Idea: Sliding Window



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

Semantic Segmentation Idea: Sliding Window



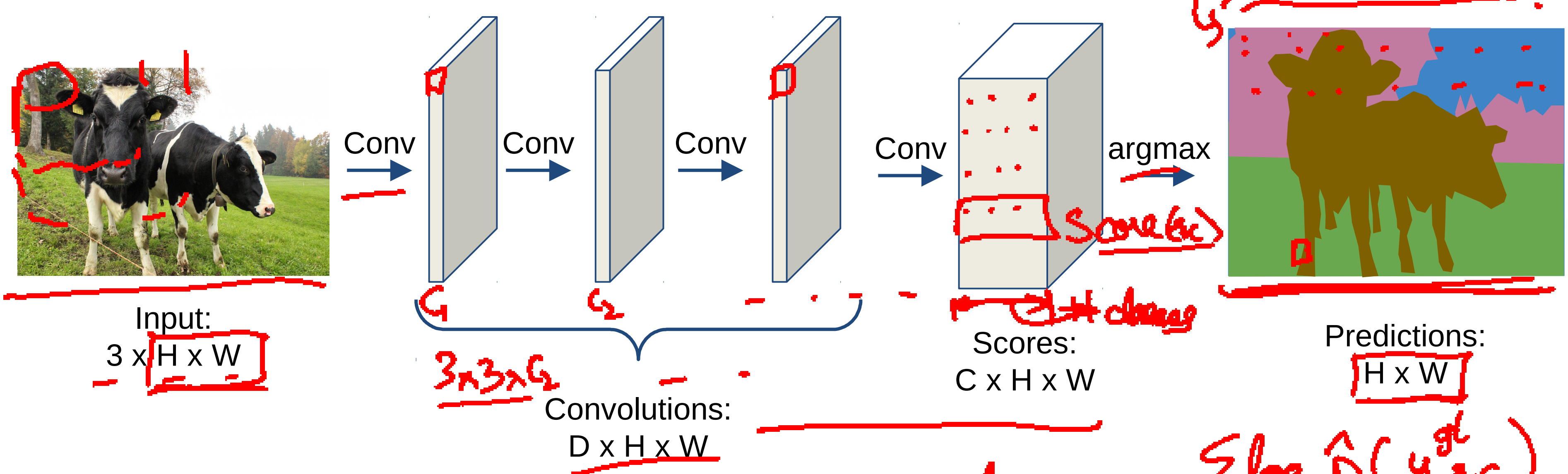
Problem: Very inefficient! Not reusing shared features between overlapping patches

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

Semantic Segmentation Idea: Fully Convolutional

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

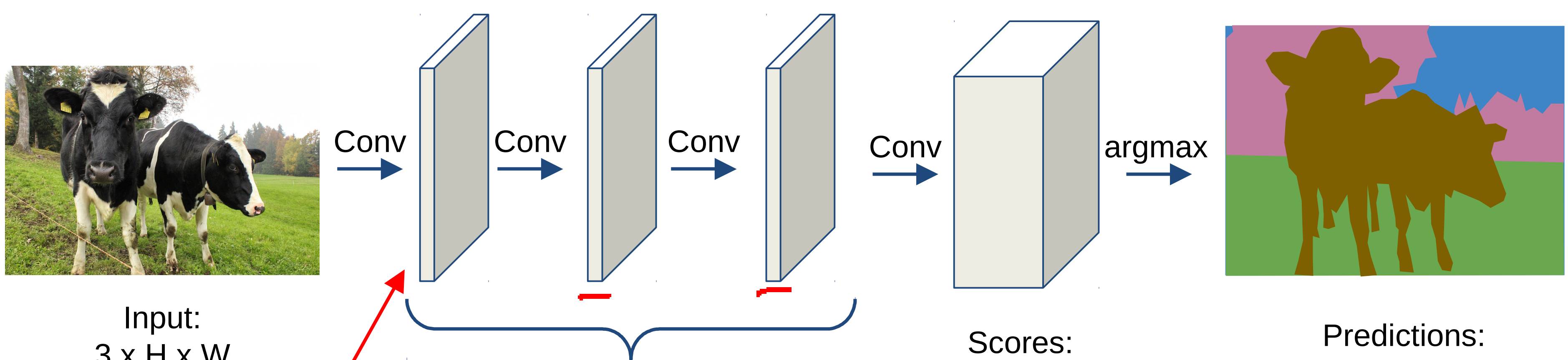
y^{gt} y



$$loss = -\sum_{x} \log \hat{p}(y^{gt}_{x,c})$$

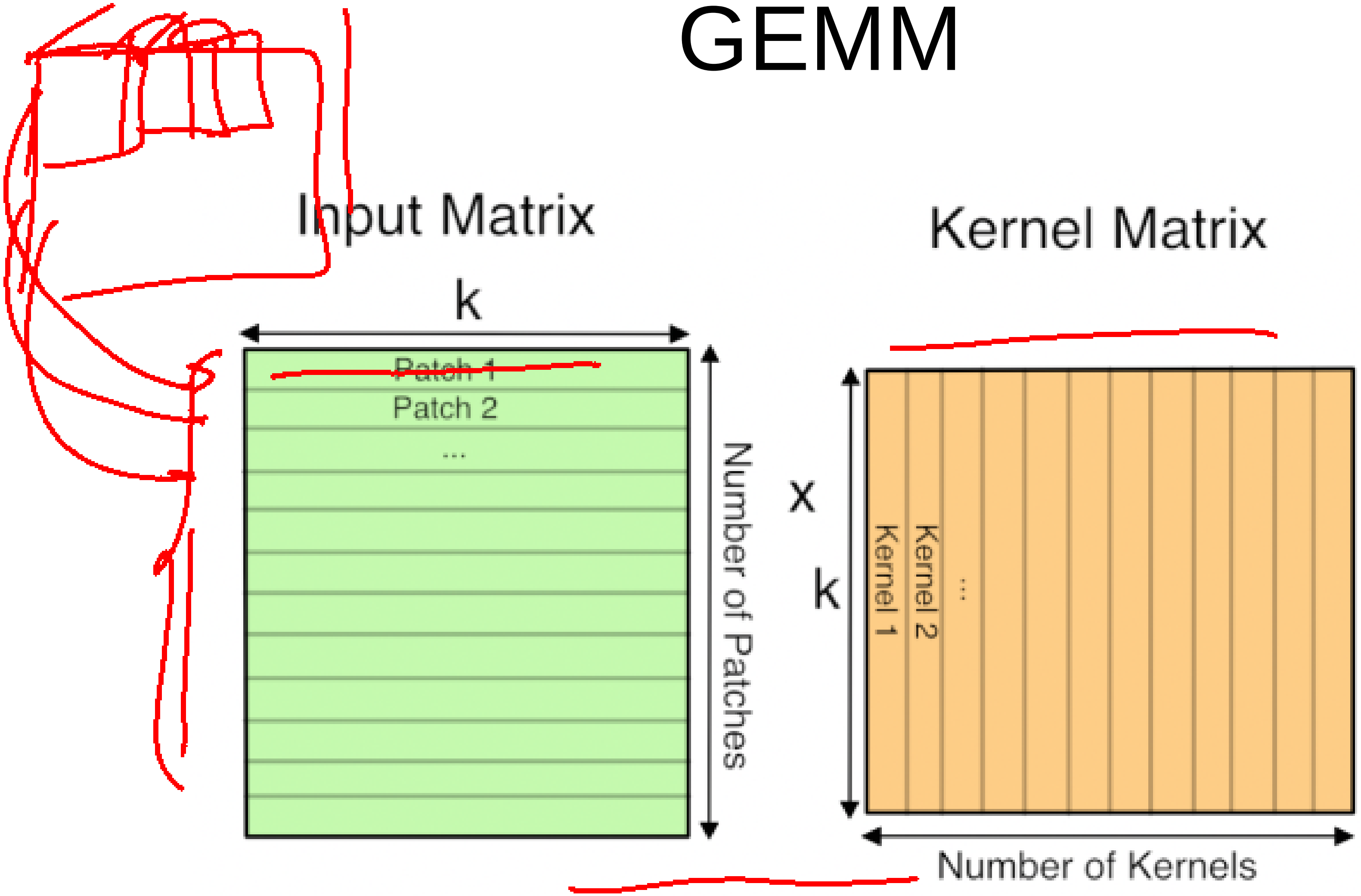
Semantic Segmentation Idea: Fully Convolutional

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



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

GEMM


Input Matrix
k
Patch 1
Patch 2
...
Number of Patches
x
Kernel Matrix
k
Kernel 1
Kernel 2
...
Number of Kernels

Time Distribution of AlexNet

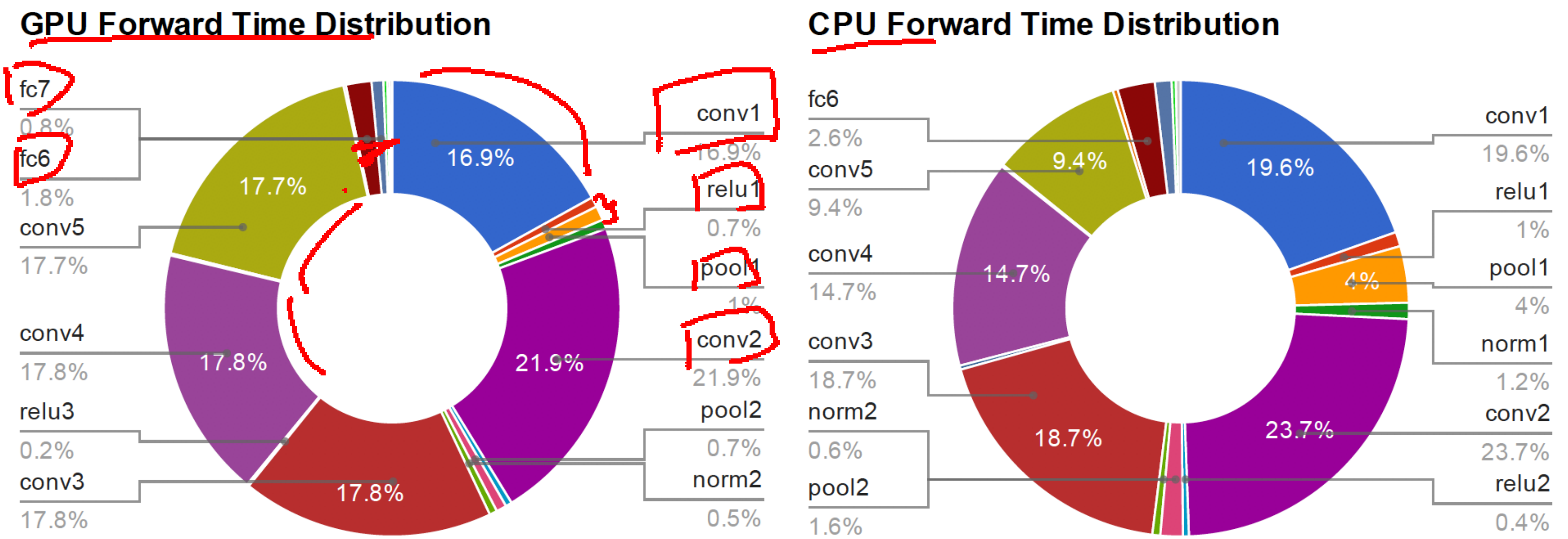
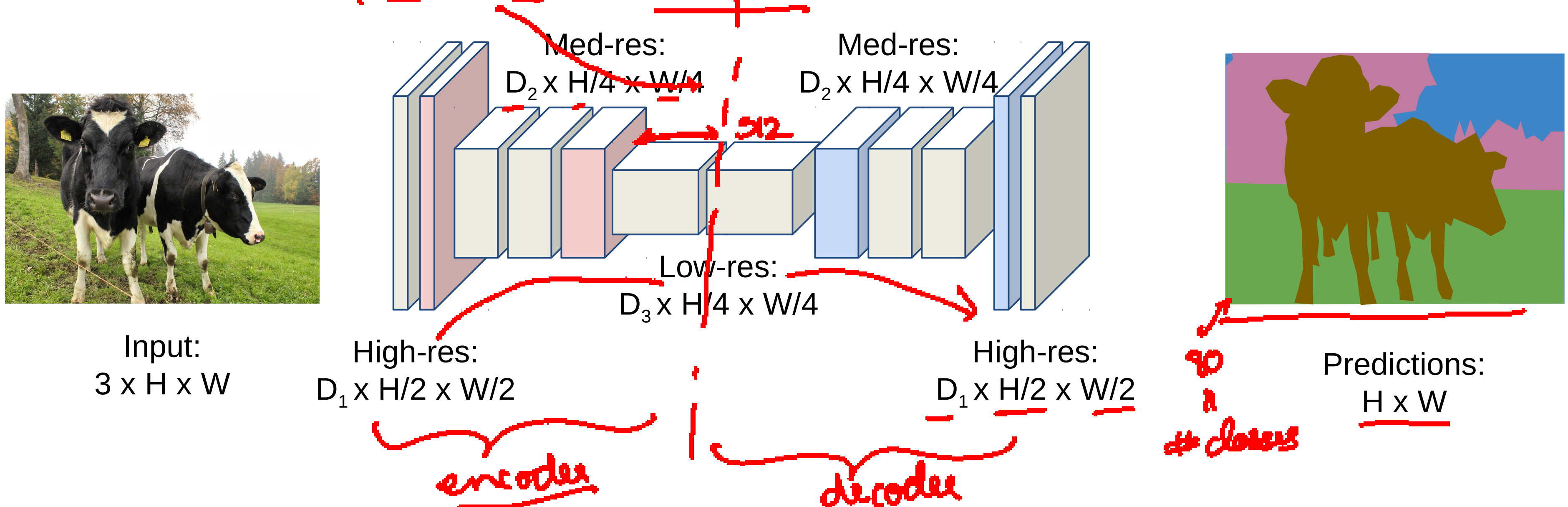


Figure Credit: Yangqing Jia, PhD Thesis

Semantic Segmentation Idea: Fully Convolutional

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



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

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

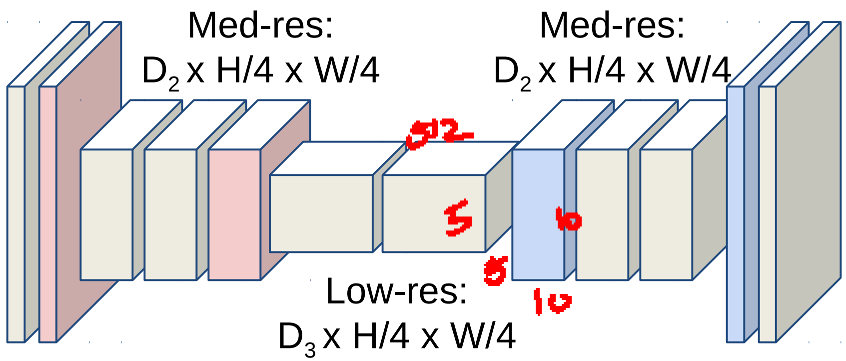
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

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

Upsampling:
???

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

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

In-Network upsampling: "Unpooling"

Nearest Neighbor

1,1

1	2
3	4

Input: 2 x 2

h^{l-1}

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

h^{l}

"Bed of Nails"

1	2
3	4

Input: 2 x 2

0,0

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

In-Network upsampling: “Max Unpooling”

stride=2

2x2

Max Pooling
Remember which element was max!

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

Input: 4 x 4

argmax x,c

5	6
7	8

Output: 2 x 2

… Rest of the network

Max Unpooling
Use positions from pooling layer

1	2
3	4

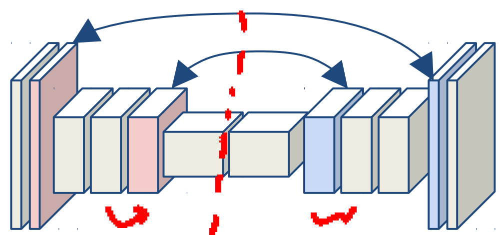
Input: 2 x 2

argmax x,c

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers



max pool

max unpooling

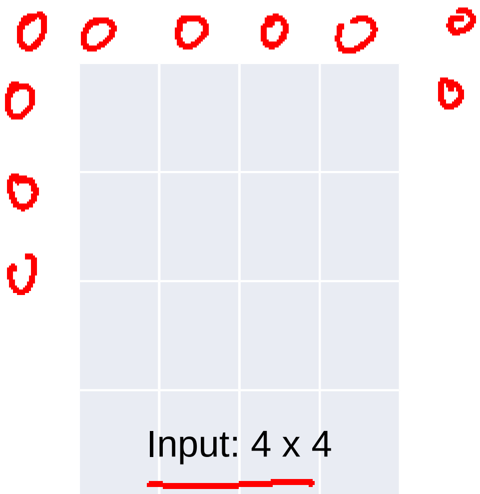
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

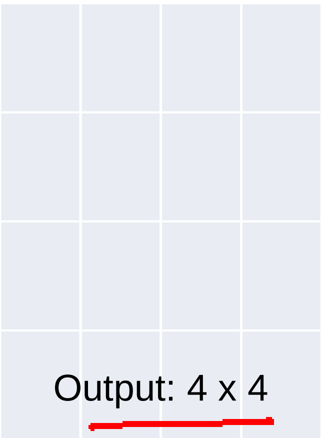
Transposed Convolutions

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

Learnable Upsampling: Transpose Convolution

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

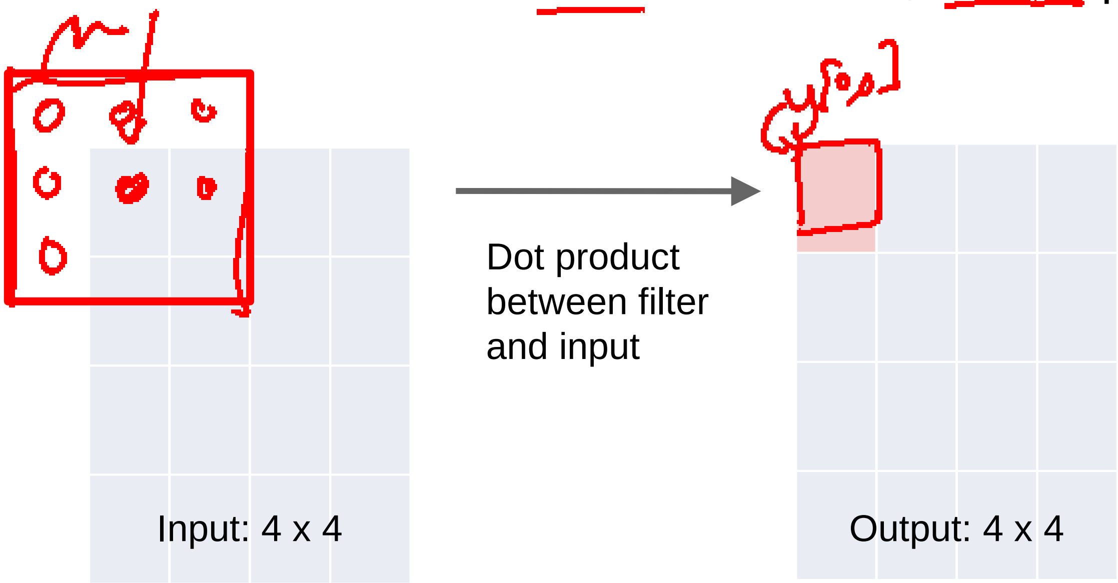




Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

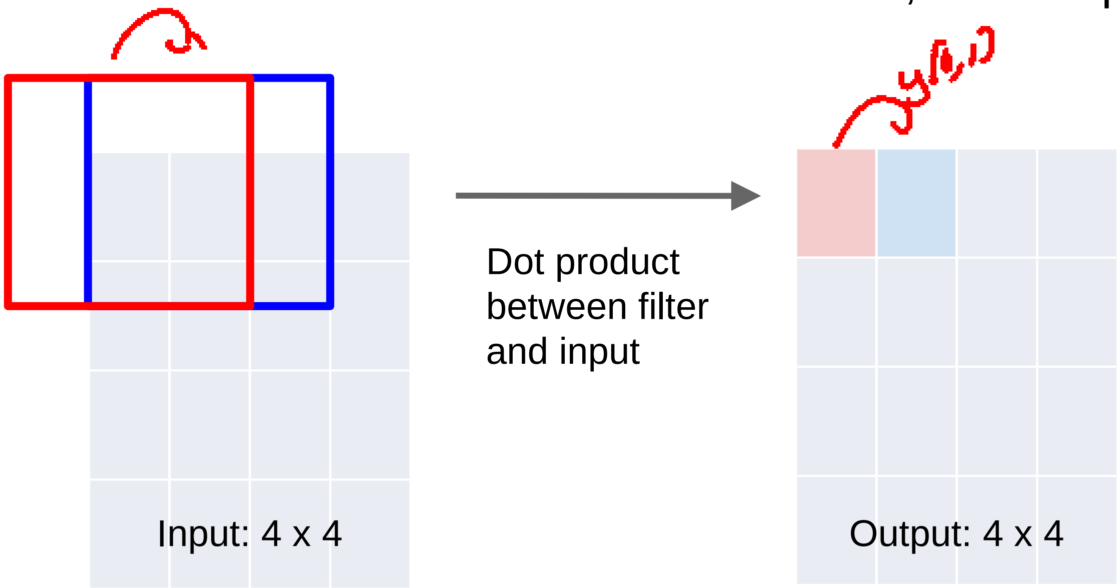
Learnable Upsampling: Transpose Convolution

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



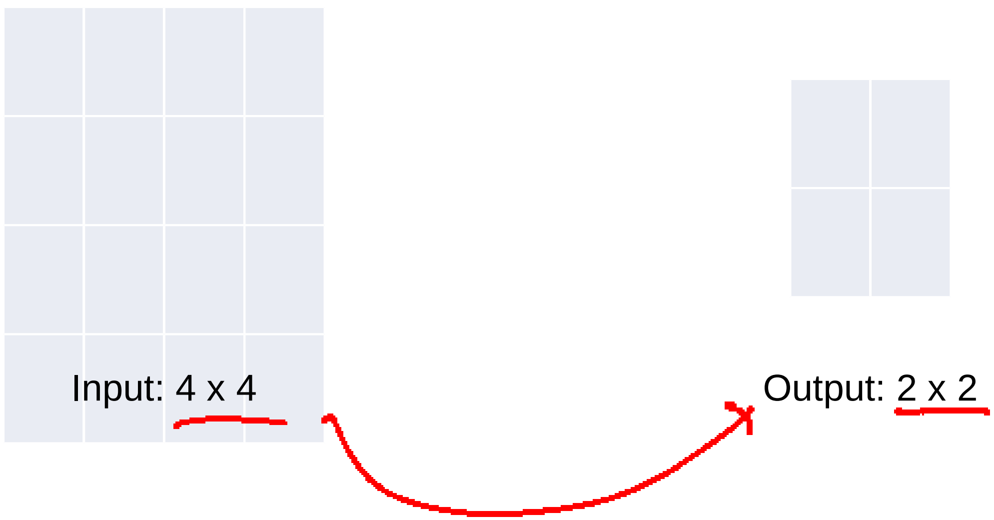
Learnable Upsampling: Transpose Convolution

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



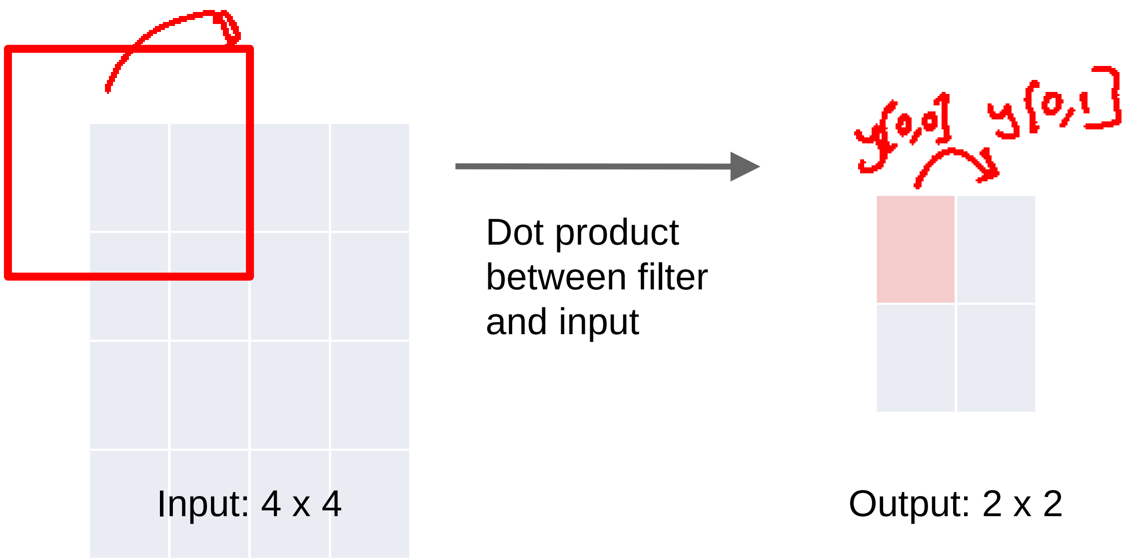
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, $\boxed{\text{stride 2}}$ pad 1



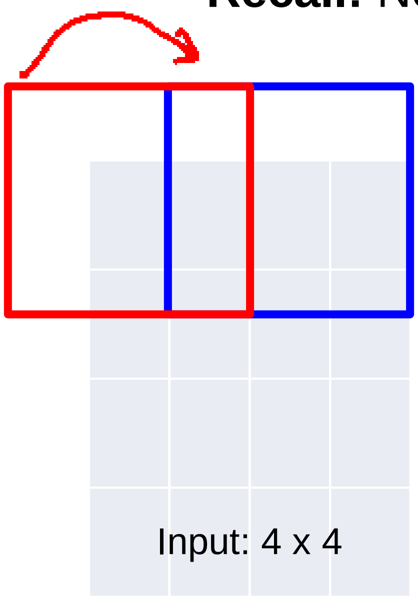
Learnable Upsampling: Transpose Convolution

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



Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1



Dot product between filter and input

y[0,0] y[0,1]

Output: 2 x 2

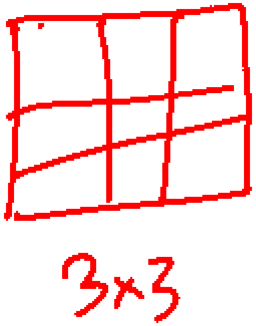
Filter moves 2 pixels in the input for every one pixel in the output

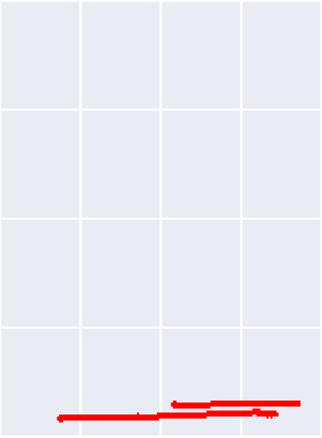
Stride gives ratio between movement in input and output

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

Learnable Upsampling: Transpose Convolution

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

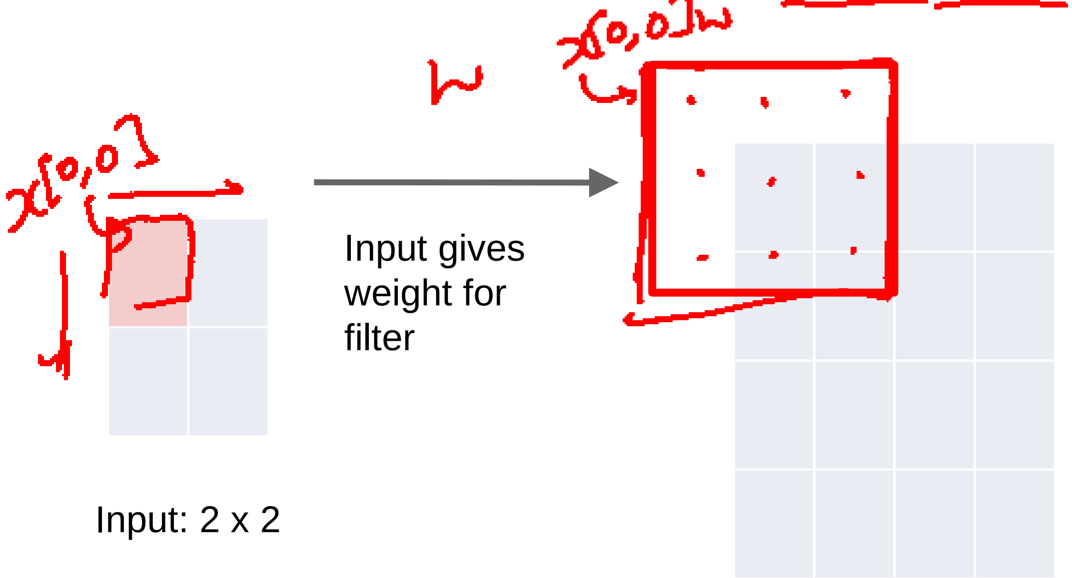




Input: 2 x 2

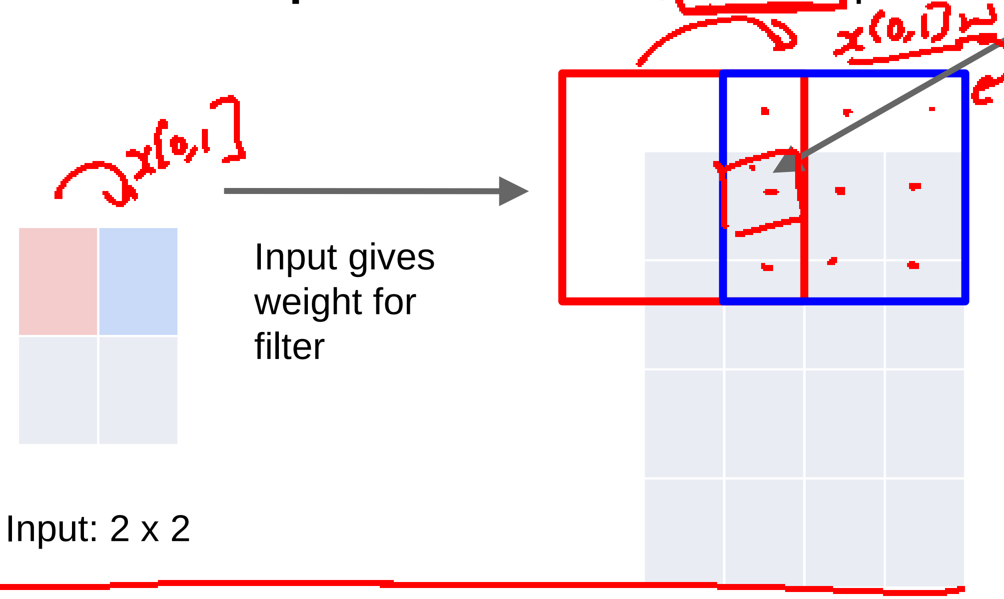
Learnable Upsampling: Transpose Convolution

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



Learnable Upsampling: Transpose Convolution

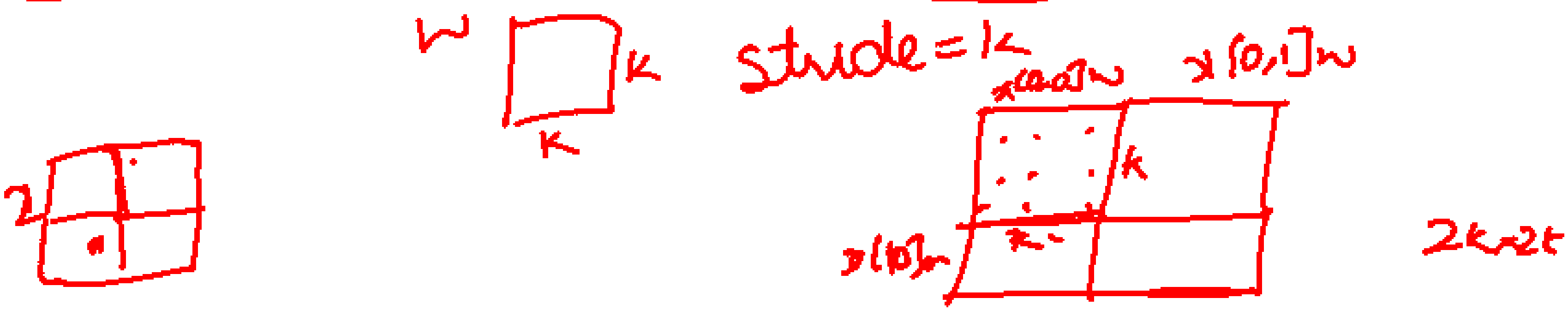
3 x 3 **transpose** convolution, stride 2 pad 1



Sum where output overlaps

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

Stride gives ratio between movement in output and input



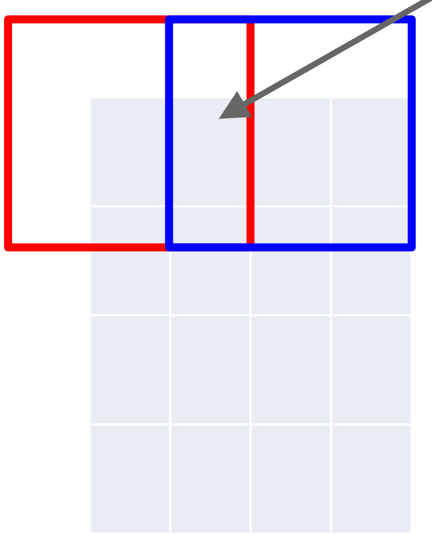
Learnable Upsampling: Transpose Convolution

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

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

Input: 2 x 2

Input gives weight for filter

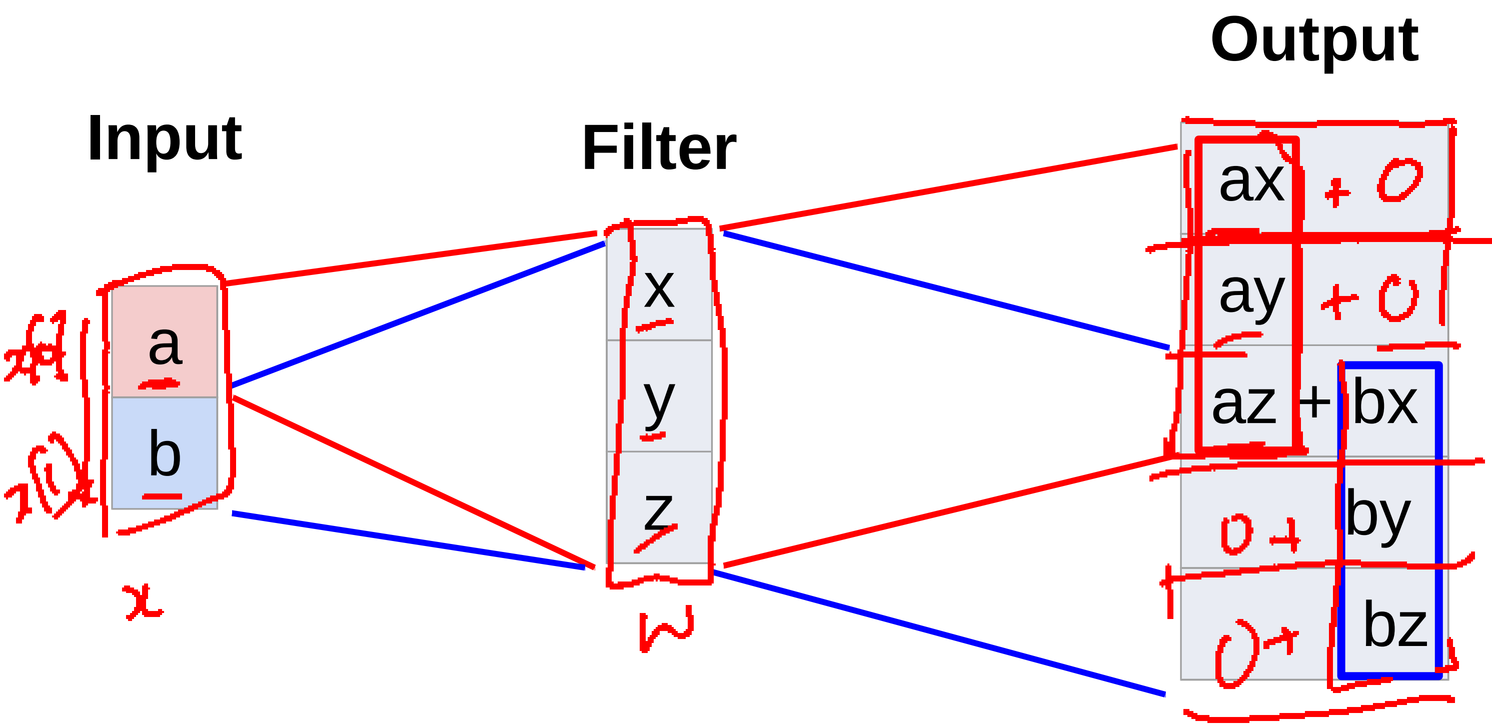


Sum where output overlaps

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

Stride gives ratio between movement in output and input

Transpose Convolution: 1D Example



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

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

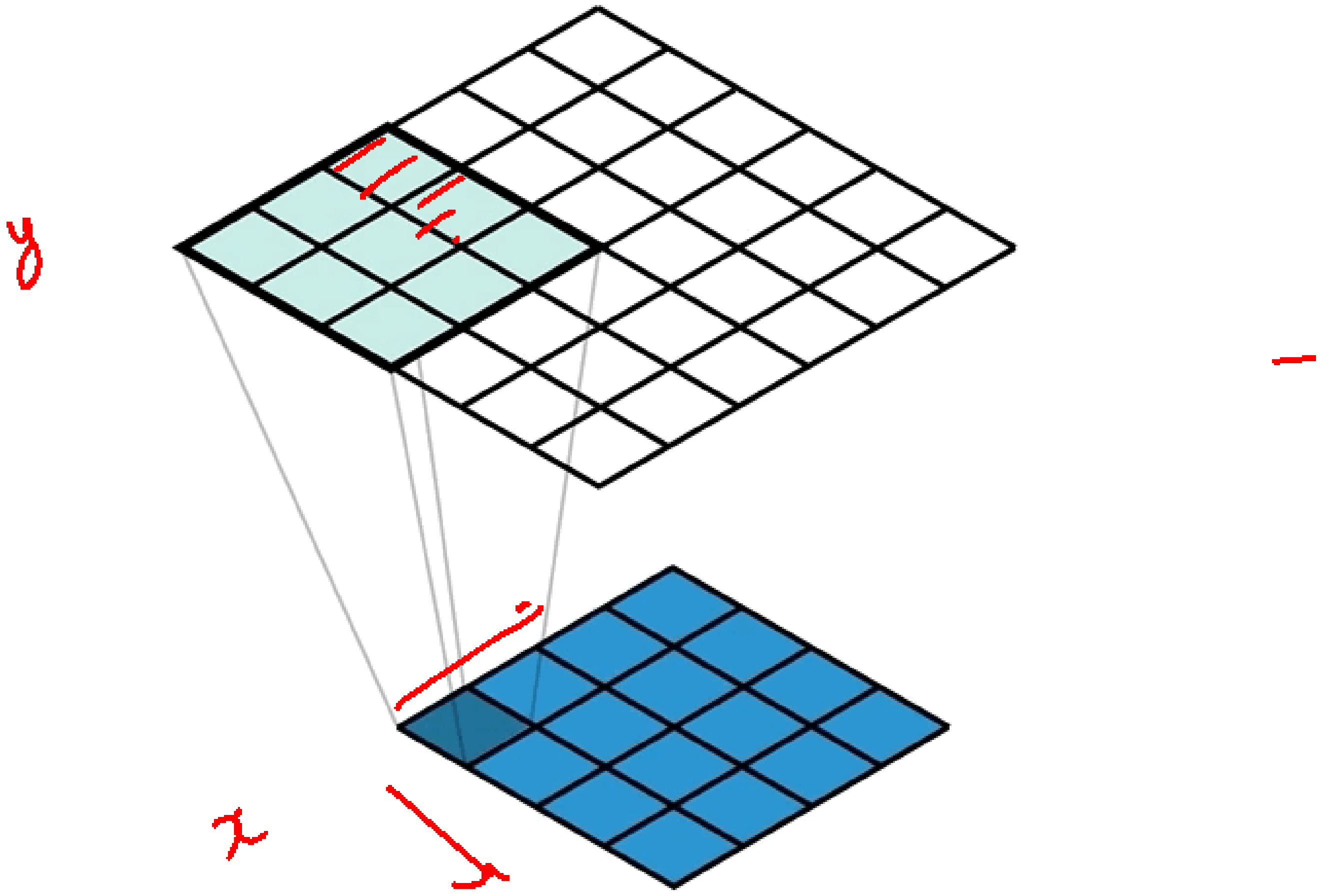


Figure Credit: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

Transposed Convolution

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

In-Network upsampling: “Unpooling”

w

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

Nearest Neighbor

$x(i,j)$

1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

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

“Bed of Nails”

1	2
3	4

Input: 2 x 2

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

$$w^* = \arg\min_{w} L(\quad)$$

Why this operation?

What is deconvolution?

- (Non-blind) Deconvolution

Given $\underline{y} = \underline{x} * \underline{w}$ and w

find $\underline{x}$

$\vec{y} = \underline{W} x$ $\hat{x} = W^{-1} y$

- Given w & y find y

Toeplitz Matrix

- Diagonals are constants

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

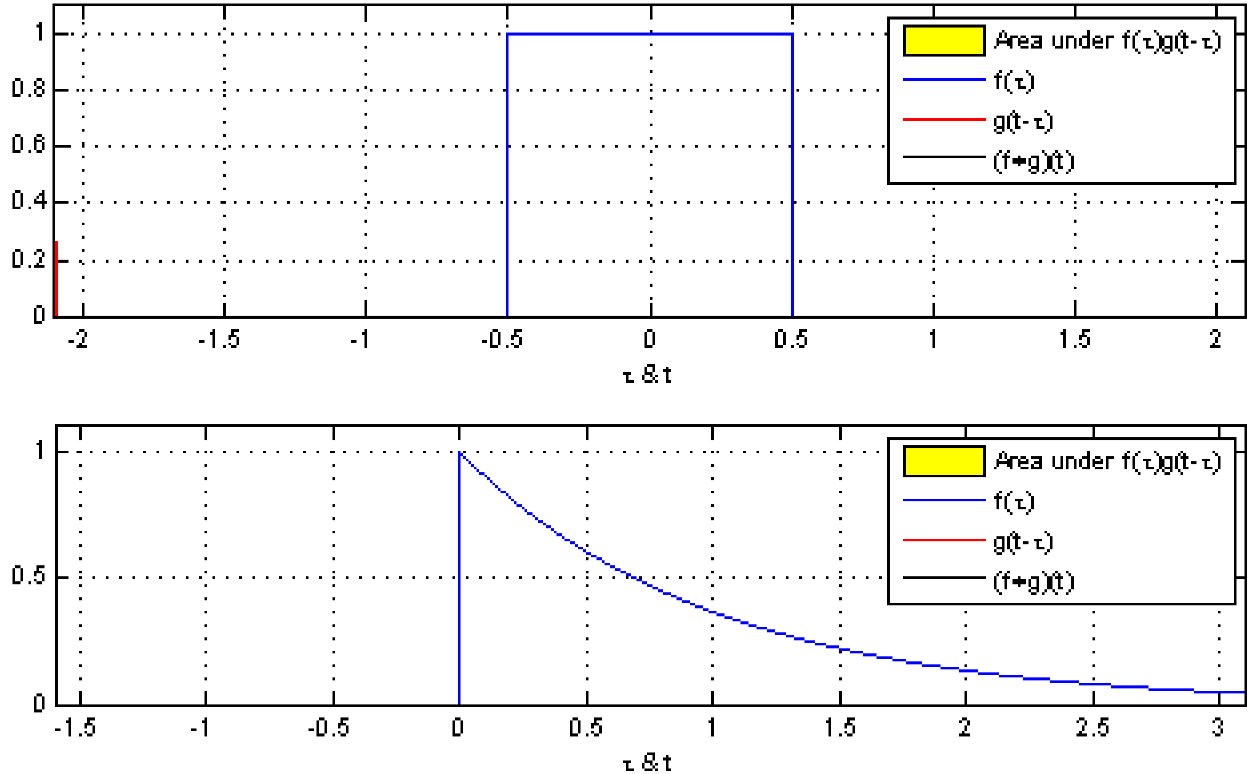
- $A_{ij} = a_{i-j}$

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

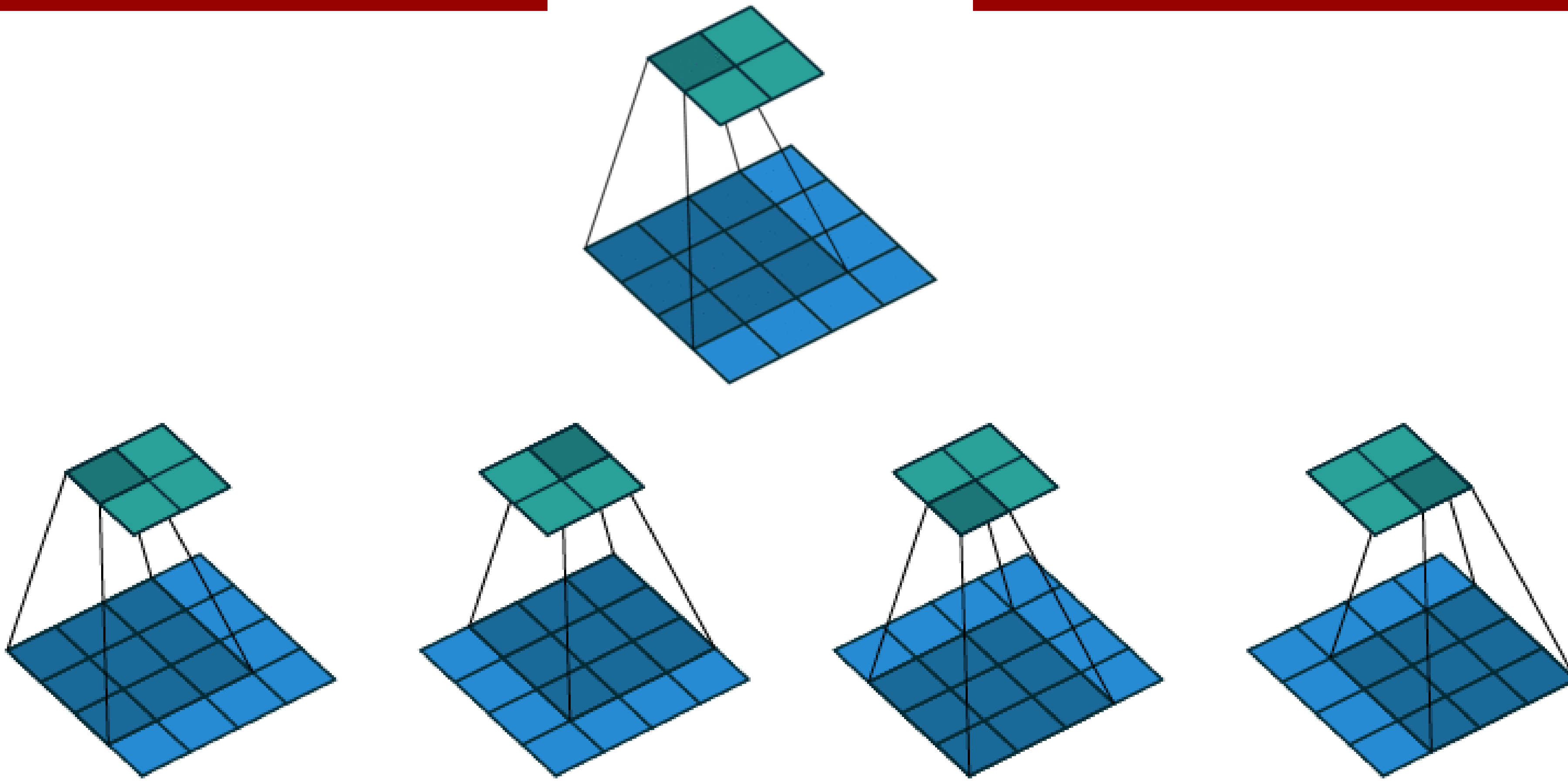
Why do we care?

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

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



"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



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

What is deconvolution?

- (Non-blind) Deconvolution

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

What does “deconvolution” have to do with “transposed convolution”?

“transposed convolution” is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

“transposed convolution” is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

"transposed convolution" is a convolution!

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)