

# Lecture Notes for **Neural Networks** and Machine Learning



CNN Visualization



# Logistics and Agenda

- Logistics
  - Syllabus Properly Revised for Labs
  - First Lab Due Next Week (Sunday)
- Agenda
  - Finish Ethics Demo
  - Paper Presentation:
    - ◆ The Limitations of Deep Learning, Chollet 2017
  - Visualizing Convolutional Architectures





# How to Make a Racist AI without really Trying



Robyn Speer, 2017

[http://blog.conceptnet.io/posts/2017/  
how-to-make-a-racist-ai-without-  
really-trying/](http://blog.conceptnet.io/posts/2017/how-to-make-a-racist-ai-without-really-trying/)

Follow Along in the Notebook:

[https://github.com/Azure/learnAnalytics-  
DeepLearning-Azure/blob/master/Students/12-  
biased-embeddings/how-to-make-a-racist-ai-  
without-really-trying.ipynb](https://github.com/Azure/learnAnalytics-DeepLearning-Azure/blob/master/Students/12-biased-embeddings/how-to-make-a-racist-ai-without-really-trying.ipynb)

Man is to Computer Programmer as Woman is to Homemaker?  
Debiasing Word Embeddings

Tolga Bolukbasi<sup>1</sup>, Kai-Wei Chang<sup>2</sup>, James Zou<sup>2</sup>, Venkatesh Saligrama<sup>1,2</sup>, Adam Kalai<sup>2</sup>

<sup>1</sup>Boston University, 8 Saint Mary's Street, Boston, MA

<sup>2</sup>Microsoft Research New England, 1 Memorial Drive, Cambridge, MA

tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

**Step 1: Identify gender subspace.** Inputs: word sets  $W$ , defining sets  $D_1, D_2, \dots, D_n \subset W$  as well as embedding  $\{\vec{w} \in \mathbb{R}^d\}_{w \in W}$  and integer parameter  $k \geq 1$ . Let

$$\mu_i := \sum_{w \in D_i} \vec{w} / |D_i|$$

be the means of the defining sets. Let the bias subspace  $B$  be the first  $k$  rows of  $\text{SVD}(\mathbf{C})$  where

$$\mathbf{C} := \sum_{i=1}^n \sum_{w \in D_i} (\vec{w} - \mu_i)^T (\vec{w} - \mu_i) / |D_i|.$$

**Step 2a: Hard de-biasing (neutralize and equalize).** Additional inputs: words to neutralize  $N \subseteq W$ , family of equality sets  $\mathcal{E} = \{E_1, E_2, \dots, E_m\}$  where each  $E_i \subseteq W$ . For each word  $w \in N$ , let  $\vec{w}$  be re-embedded to

$$\vec{w} := (\vec{w} - \vec{w}_B) / \|\vec{w} - \vec{w}_B\|.$$

For each set  $E \in \mathcal{E}$ , let

$$\begin{aligned}\mu &:= \sum_{w \in E} w / |E| \\ \nu &:= \mu - \mu_B\end{aligned}$$

$$\text{For each } w \in E, \quad \vec{w} := \nu + \sqrt{1 - \|\nu\|^2} \frac{\vec{w}_B - \mu_B}{\|\vec{w}_B - \mu_B\|}$$



# Paper Presentation

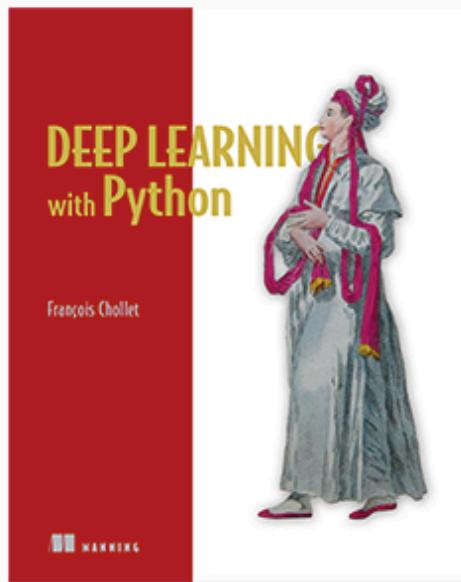
## The limitations of deep learning

This post is adapted from Section 2 of Chapter 9 of my book, [Deep Learning with Python](#) (Manning Publications).

Mon 17 July 2017

By [Francois Chollet](#)

In [Essays](#).



It is part of a series of two posts on the current limitations of deep learning, and its future.

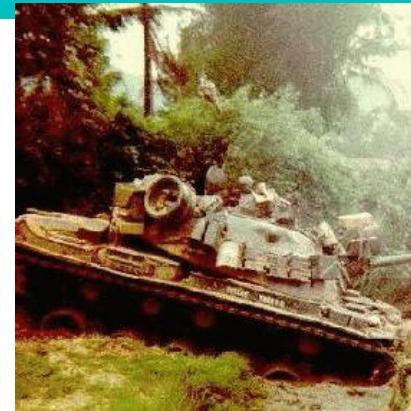


# CNN Visualization



# Why Visualize Trained CNN Architectures?

- **Reliability:**  
“1998 Military Project”  
Tanks in Trees  
Maybe an Urban Legend, but Plausible
- **Curiosity:** Answering why something works and trying to discover it is *research*
- **Improvement:** You cannot change what you do not understand



<https://neil.fraser.name/writing/tank/>

# Attribution versus Influence

- **Attribution:**  
The amount of the quantity of interest that can be attributed to a particular neuron. No need to assign “influential regions.”
- **Influence:**  
The sensitivity of the quantity of interest to a particular input or input distribution. Influential regions must be segmented before analysis.
- Difference is nuanced, but important in research community.

Leino, Klas, Shayak Sen, Anupam Datta, Matt Fredrikson, and Linyi Li. "Influence-directed explanations for deep convolutional networks." In *2018 IEEE International Test Conference (ITC)*, pp. 1-8. IEEE, 2018.



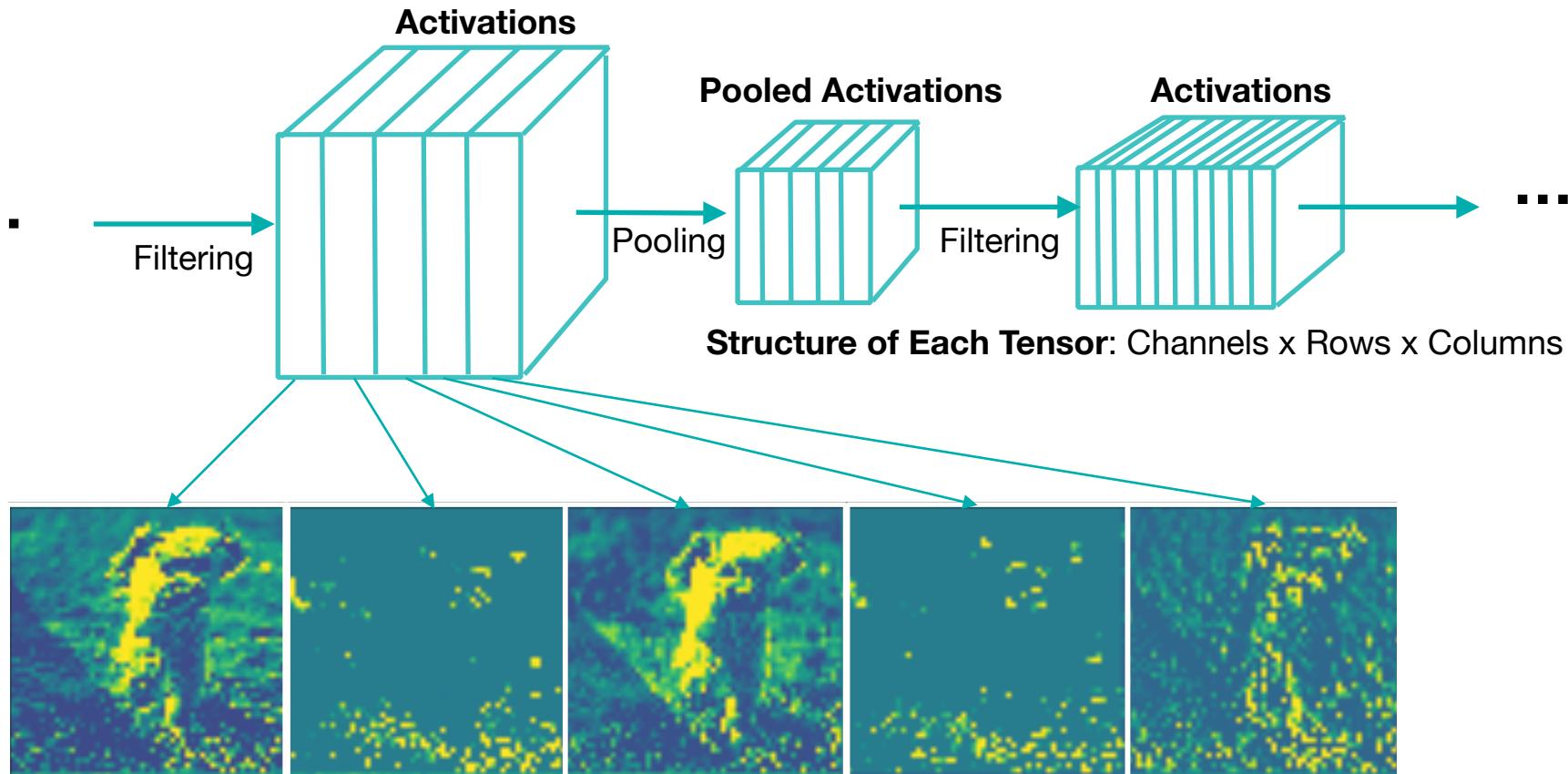
# Methods for Visualization

- No “Ideal” Technique, Depends on Question
- Visualize **Filter Activation Maps**
  - What parts of the images activate each filter?
- Visualize **Filters**
  - What is each filter most sensitive to?
- **Heatmaps** of Class Activation
  - What part of an input image most influences each final output?



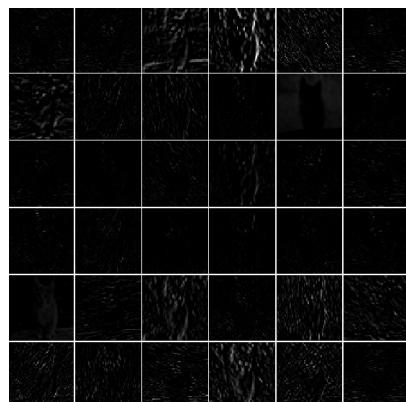
# Visualizing Intermediate Activations

- Look layer by layer
- **Assume:** each filter learns something independent

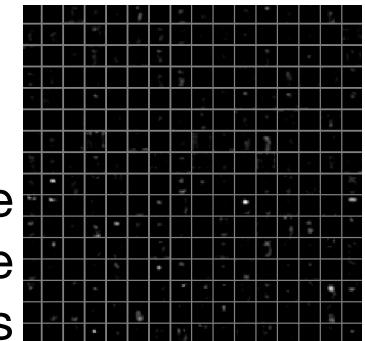


# Visualizing Intermediate Activations

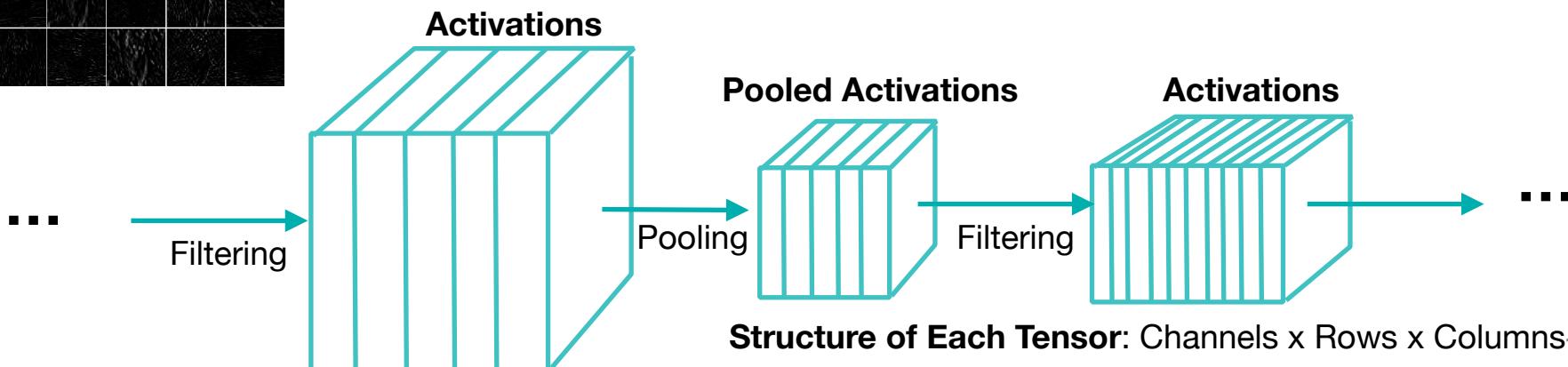
- **Recall:** general structure of most CNNs
  - Small kernels throughout (3x3)
  - Filtering followed by Pooling (spatial downsampling)
  - More filters in later layers



**Early Activations**  
are larger but not as  
numerous

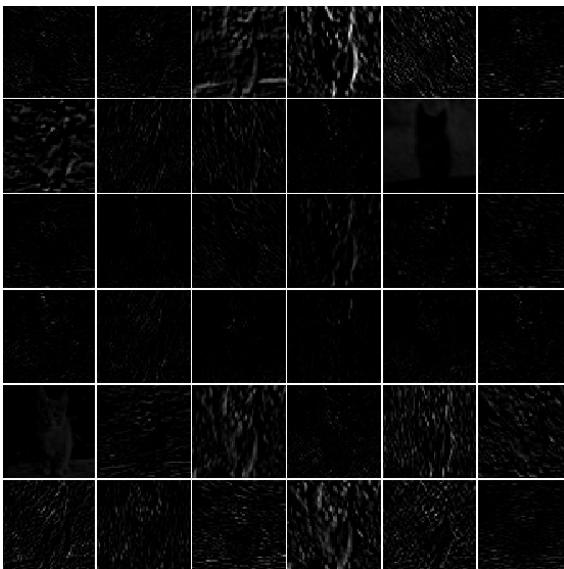


**Later Activations** are  
smaller and more  
numerous



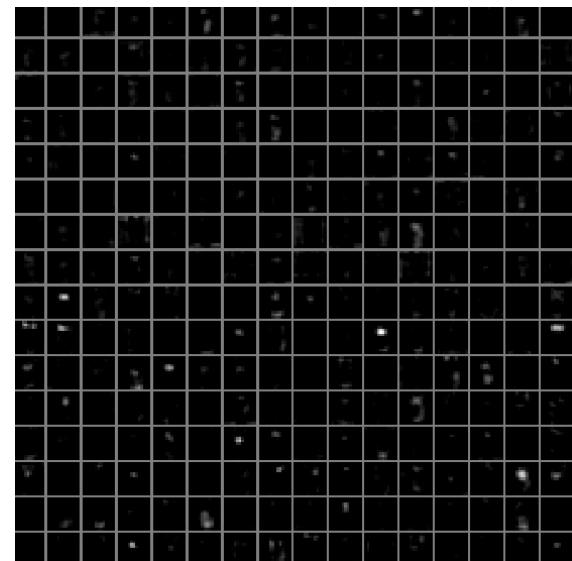
# Visualizing Intermediate Activations

- **Result:** Information Distillation Pipeline
  - Deeper layers have more abstract triggers
  - Deeper activations are increasingly sparse
  - Early layers are texture and edge detectors
  - Notion of “High Level Abstraction,” has biological motivation



**Early Activations** are  
larger but not as  
numerous

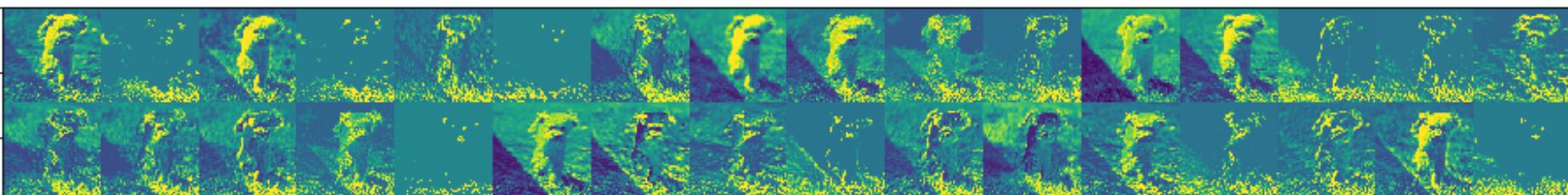
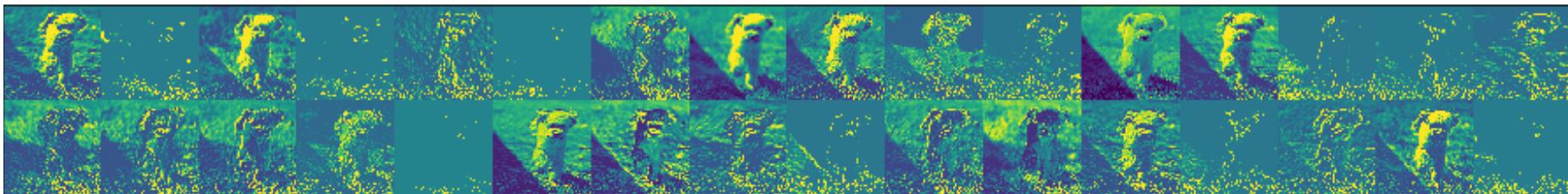
**Later Activations** are  
smaller and more  
numerous





# Visualizing ConvNets

Part One: Filter Activations



Follow Along: [LectureVisualizingConvnets.ipynb](#)  
activation-demo

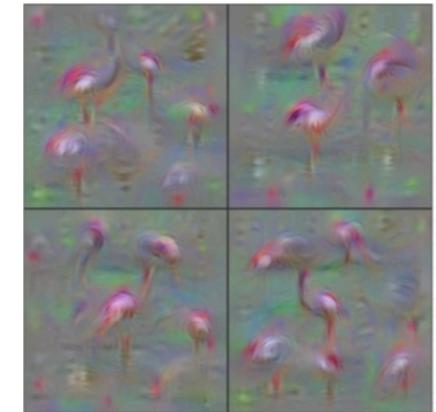
13



# Visualizing Filters: Class Neuron

- **Idea:** What Maximally Activates a Class Output?
  - Gradient Ascent in the Input Space

$$I \leftarrow I + \eta \nabla f_c(I)$$



where  $c$  is a specific neuron in output layer

$f$  is the activation before softmax

$I$  is the input image, init to zeros (or random)

gradient update is for  $I$

CNN weights stay unchanged



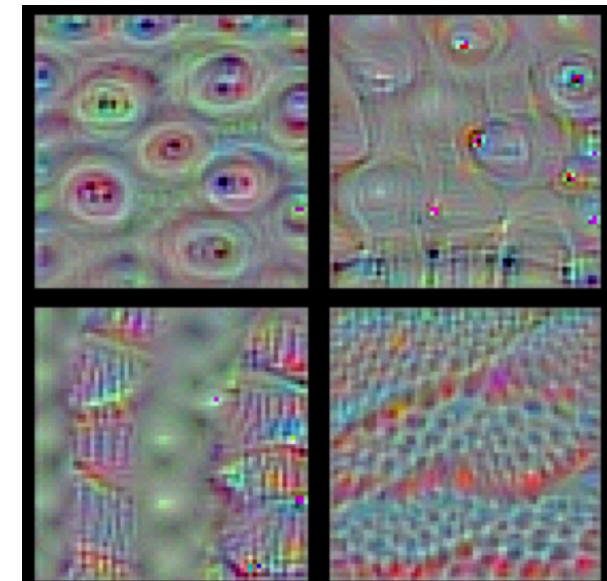
# Visualizing Filters: Maximal Activations

- **Idea:** What Maximally Activates a **Filter**?
  - **Again:** Gradient Ascent in the Input Space

$$I \leftarrow I + \eta \sum_{i,j} \nabla f_n(I)_{i,j}$$

“trick” must use norm of gradient

where  $n$  is a specific **filter** in a layer  
 $f$  is the activation of  $n^{th}$  filter in layer  
 $I$  is some random image, or zeros  
gradient update is for  $I$



# Visualizing Filters: Deep Dream

- **Idea:** How to generate new concepts (**dream**)?
  - Reinforce what is learned? How?

$$I \leftarrow \epsilon_{iter} \left( I + \eta \sum_{i,j} \nabla f_A(I)_{i,j} \right)$$

noise process

use inverse L1 norm of gradient

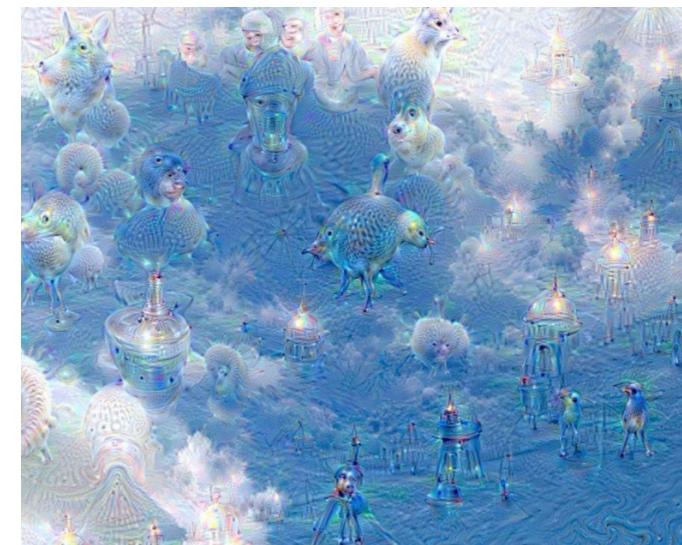
where  $A$  is a specific **conv layer**

$f$  is all **activations** of  $A^{th}$  layer

$I$  is init to **some input image**

gradient update is for  $I$

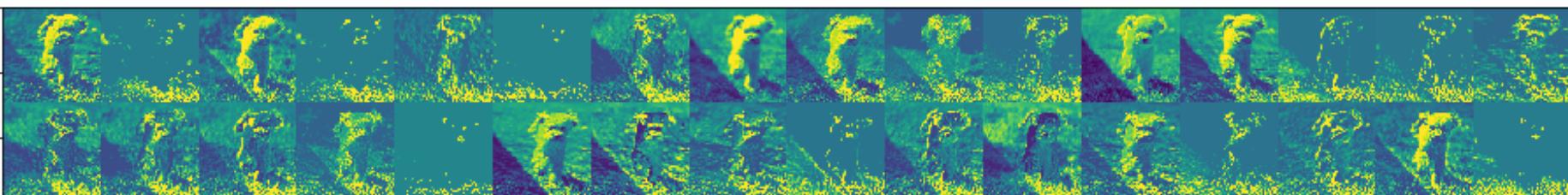
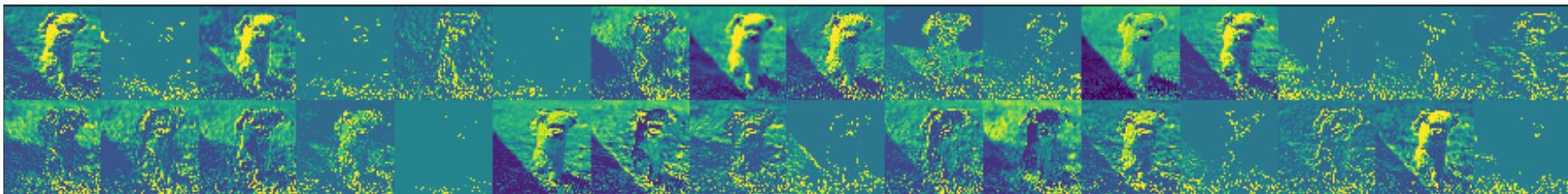
$\epsilon$  add randomness through scaling





# Visualizing ConvNets

Part Two: Image Gradients



Follow Along: [LectureVisualizingConvnets.ipynb](#)  
activation-demo

17



# Class Activation Mapping (CAM)

- **Idea:** What areas of the image contributed most to the classification result?
- Also, for each class, what areas of the image exhibit features of that class?
- Use change in output, w.r.t. final conv layer

$$\alpha_k^c = \frac{1}{|I \times J|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

final layer output in response to image  $I$   
 $c$  is class of interest

final convolutional layer,  $L$ , activations  
for row, column, channel

gradient weight for channel  $k$  and class  $c$  in layer  $L$   
 $k$  in  $1 \dots K$  activations in final layer



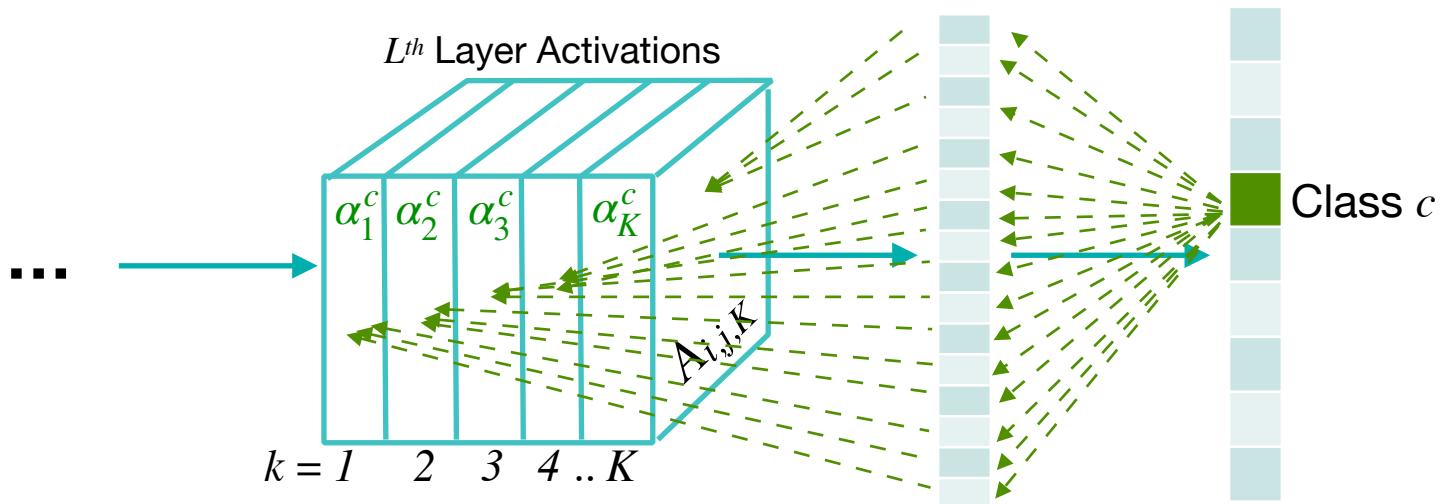
# Class Activation Mapping (CAM)

$$\alpha_k^c = \frac{1}{|I \times J|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

gradient weight for channel  $k$  and class  $c$  in layer  $L$   
 $k$  in  $1 \dots K$  activations in final layer

final layer output in response to image  $I$   
 $c$  is class of interest

final convolutional layer,  $L$ , activations  
for row, column, channel



**Sensitivity of Class to Activations**



# Class Activation Mapping (CAM)

$$\alpha_k^c = \frac{1}{|I \times J|} \sum_{i,j} \frac{\partial f_c(I)}{\partial A_{i,j,k}^{(L)}}$$

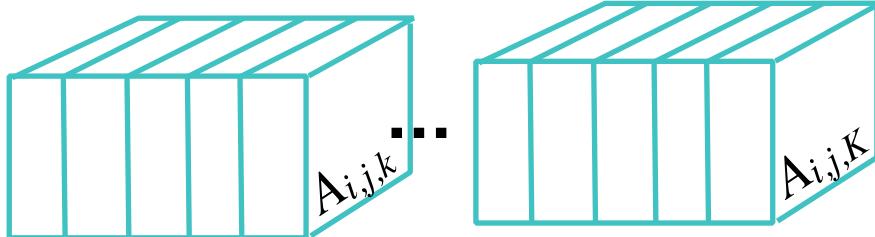
gradient weight for channel  $k$  and class  $c$  in layer  $L$   
 $k$  in  $1 \dots K$  activations in final layer

final layer output in response to image  $I$   
 $c$  is class of interest

final convolutional layer,  $L$ , activations  
for row, column, channel

**Heatmap,  $S$ , is the **weighted sum** of final layer activations:**

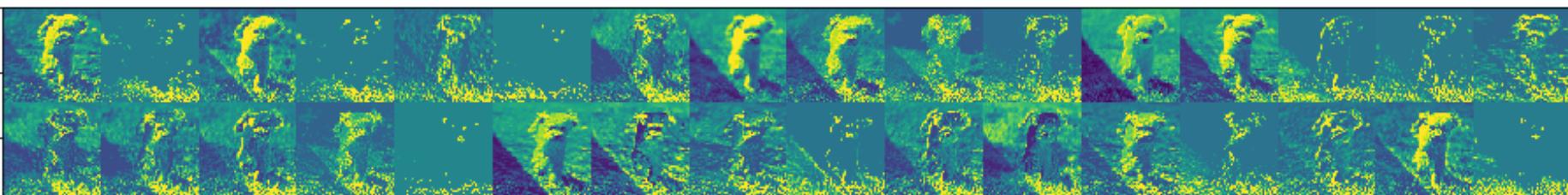
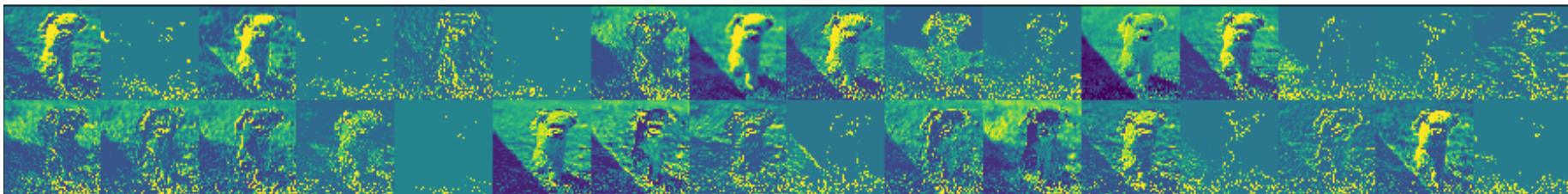
$$S_{i,j} = \frac{1}{S_{max}} \sum_k \alpha_k^c A_{i,j,k}^{(L)}$$





# Visualizing ConvNets

Part Three: Grad-CAM



Follow Along: [LectureVisualizingConvnets.ipynb](#)  
activation-demo

21



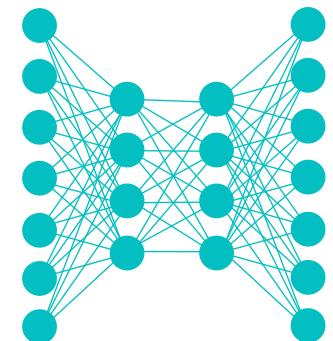
# Lecture Notes for Neural Networks and Machine Learning

CNN Visualization



**Next Time:**  
Fully Convolutional Learning

**Reading:** Chollet 5.4

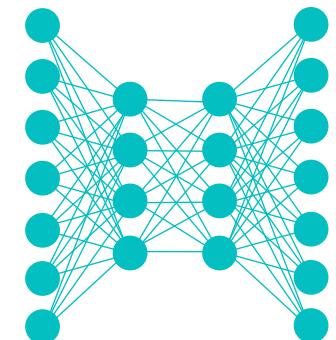




# Lecture Notes for **Neural Networks** and Machine Learning



Fully Convolutional Learning



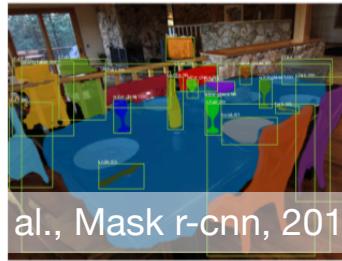
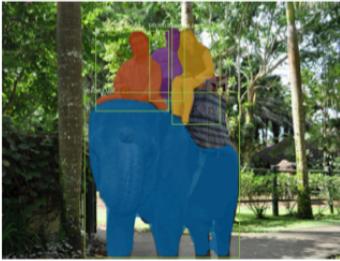
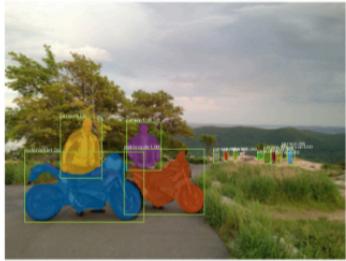
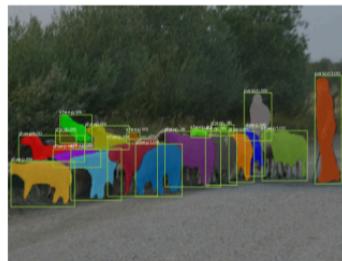
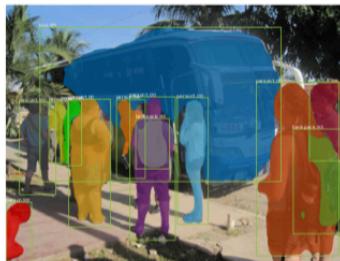
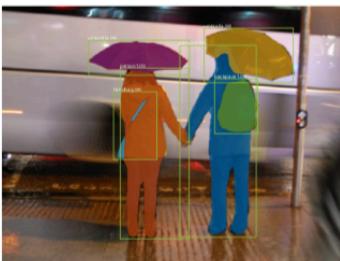
# Logistics and Agenda

- Logistics
  - Start on Lab One Now!
- Agenda
  - Types of CNN Learning Problems
  - R-CNN
  - YOLO
  - Mask R-CNN



# Types of Fully Convolutional Problems

- Semantic Segmentation
- Object Detection
- Instance Segmentation



He et al., Mask r-cnn, 2018

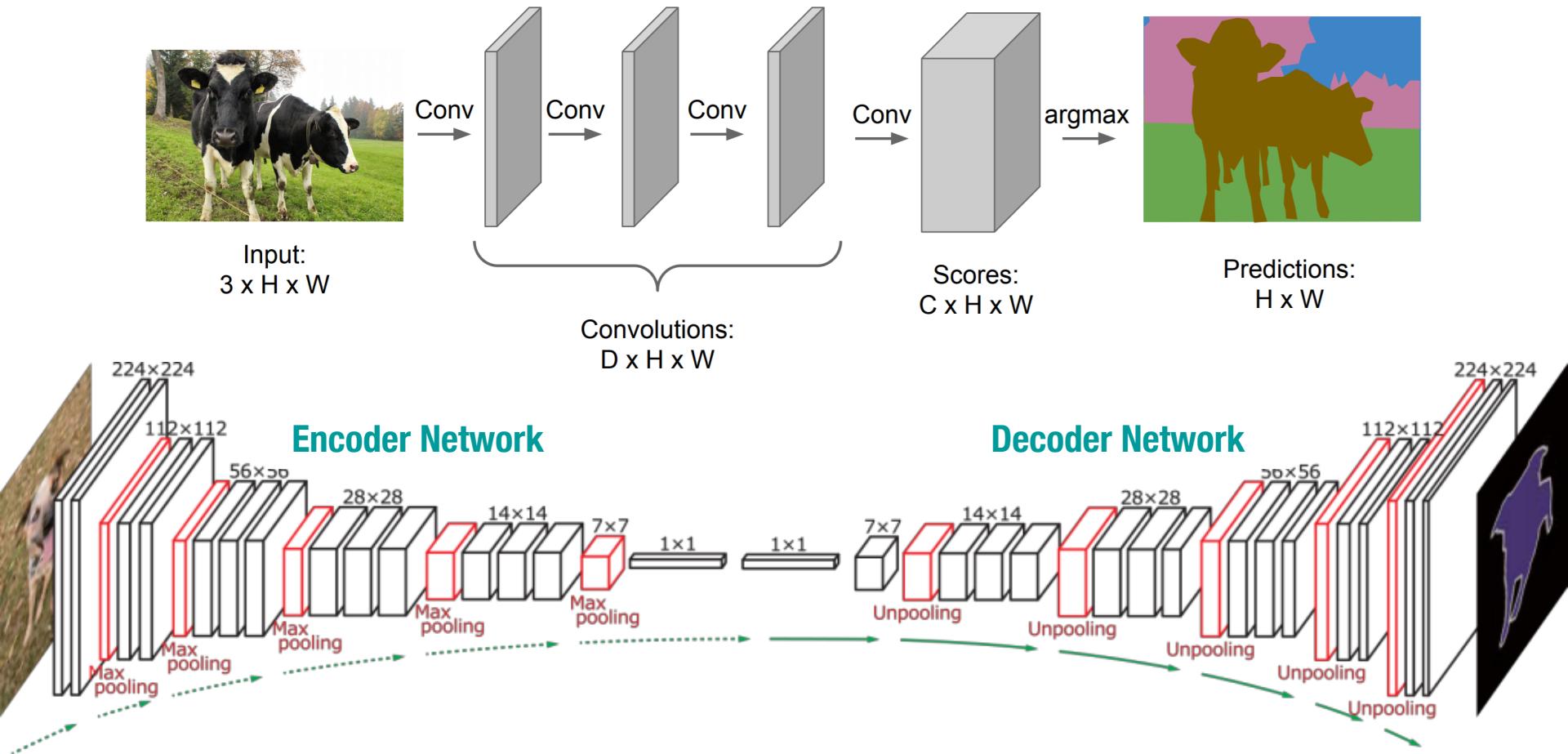


# Semantic Segmentation



# Semantic Segmentation

- Given a set of pixels, classify each pixel according to what instance it belongs



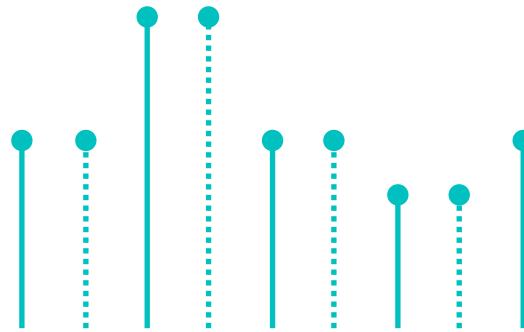
<https://tjmachinelearning.com/lectures/1718/fcn/>

28



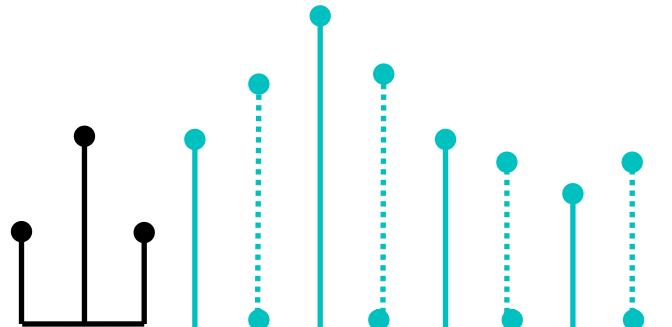
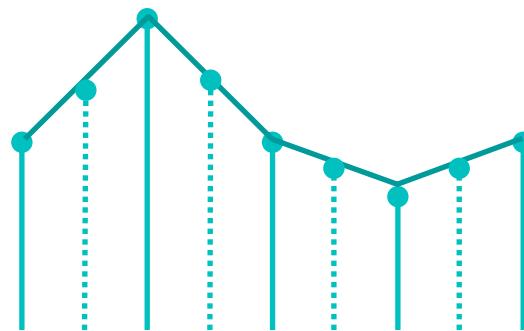
# Aside: Integer Upsampling via Interpolation

**Nearest Neighbor**

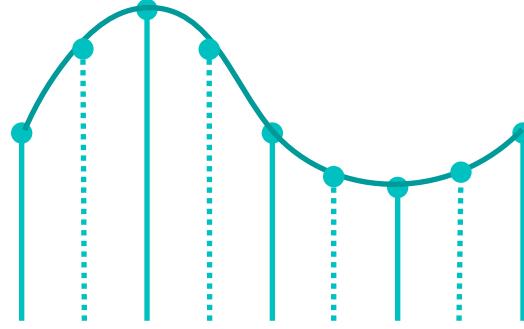


All are equivalent to inserting zeros and applying convolutional filter

**Linear**



**Cubic**



# Aside: Image Upsampling, Integer Factor

- Insert Zeros
- Convolve

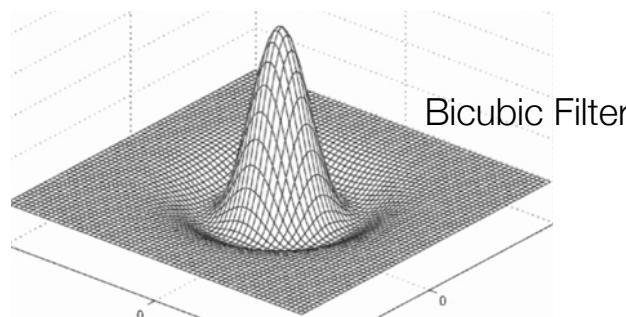
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



1		2		3		4	
5		6		7		8	
9		10		11		12	
13		14		15		16	

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

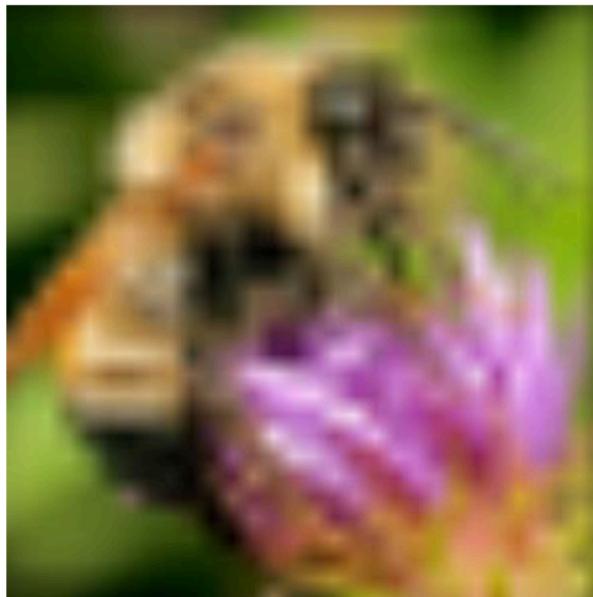
Bilinear Filtering



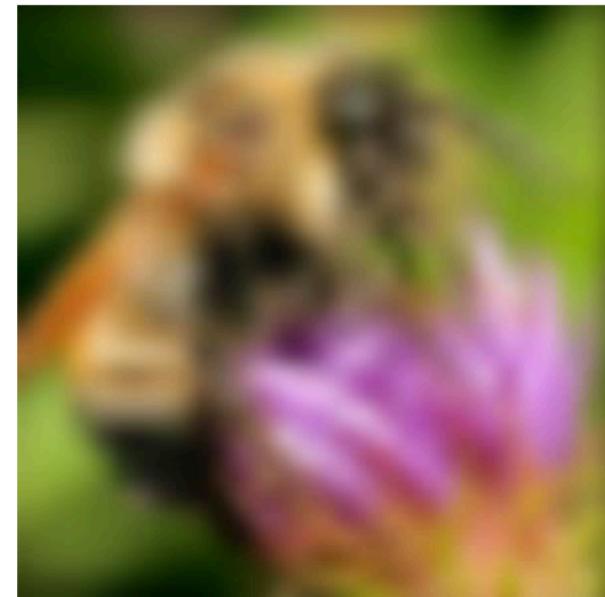
# Aside: Image Upsampling, Integer Factor



**Nearest Neighbor**



**Bilinear**



**Bicubic**

<https://www.cs.toronto.edu/~guerzhoy/320/lec/upsampling.pdf>



# Semantic Segmentation

## 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

5	6
7	8

Output: 2 x 2

## Max Unpooling

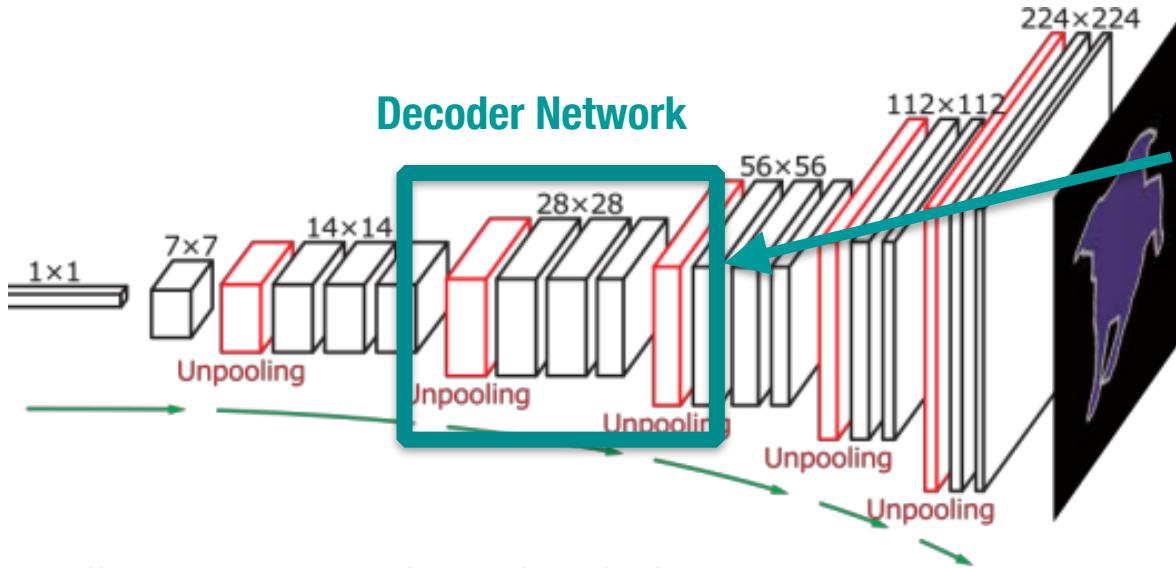
Use positions from pooling layer

1	2
3	4



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

Output: 4 x 4



Some knucklehead started calling this **deconvolution**. If you use that term in this class, **you fail**.

This is unpooling and then convolution, but **now the interpolation filters are learned!!**



# What about transpose convolution?

Regular Convolution

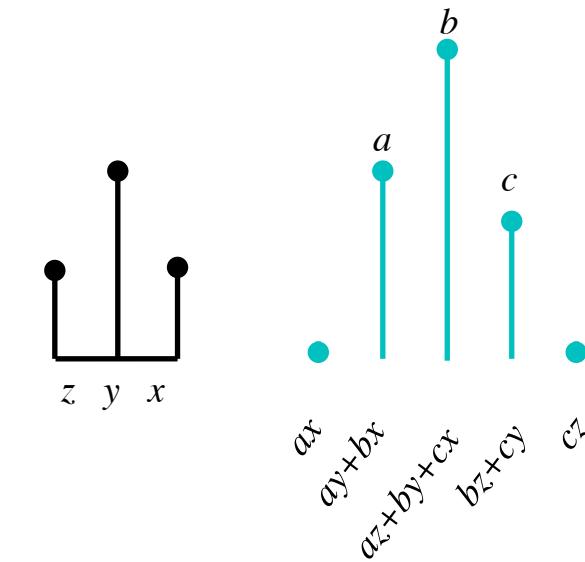
Convolution as Matrix Multiplication

$y$	$x$	0	0	0
$z$	$y$	$x$	0	0
0	$z$	$y$	$x$	0
0	0	$z$	$y$	$x$
0	0	0	$z$	$y$

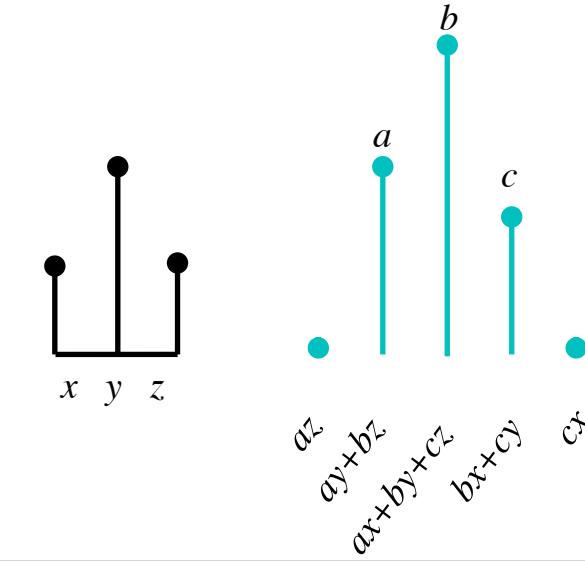
$$\times \begin{bmatrix} 0 \\ a \\ b \\ c \\ 0 \end{bmatrix} = \begin{bmatrix} ax \\ ay+bx \\ az+by+cx \\ bz+cy \\ cz \end{bmatrix}$$

Transpose

$y$	$z$	0	0	0
$x$	$y$	$z$	0	0
0	$x$	$y$	$z$	0
0	0	$x$	$y$	$z$
0	0	0	$x$	$y$

$$\times \begin{bmatrix} 0 \\ a \\ b \\ c \\ 0 \end{bmatrix} = \begin{bmatrix} az \\ ay+bz \\ ax+by+cz \\ bx+cy \\ cx \end{bmatrix}$$


Transpose Convolution



# Transpose Convolution: Strides

Regular Convolution

Strided Convolution as Matrix Multiplication

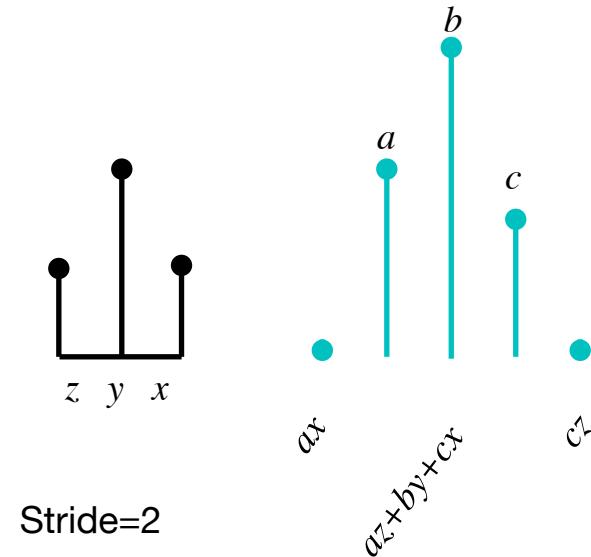
$$\begin{array}{|c|c|c|c|c|} \hline y & x & 0 & 0 & 0 \\ \hline 0 & z & y & x & 0 \\ \hline 0 & 0 & 0 & z & y \\ \hline \end{array} \times \begin{array}{|c|} \hline 0 \\ \hline a \\ \hline b \\ \hline c \\ \hline 0 \\ \hline \end{array} = \begin{array}{|c|c|} \hline ax \\ \hline az+by+cx \\ \hline cz \\ \hline \end{array}$$

↓

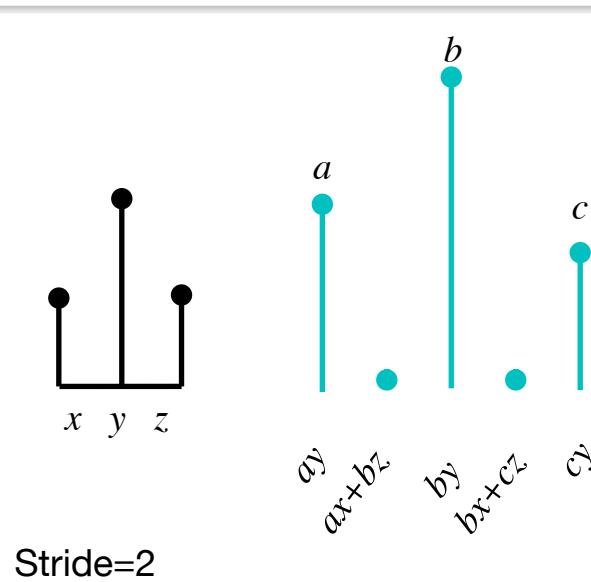
Transpose

$$\begin{array}{|c|c|c|} \hline y & 0 & 0 \\ \hline x & z & 0 \\ \hline 0 & y & 0 \\ \hline 0 & x & z \\ \hline 0 & 0 & y \\ \hline \end{array} \times \begin{array}{|c|} \hline a \\ \hline b \\ \hline c \\ \hline \end{array} = \begin{array}{|c|c|} \hline ay \\ \hline ax+bz \\ \hline by \\ \hline bx+cz \\ \hline cy \\ \hline \end{array}$$

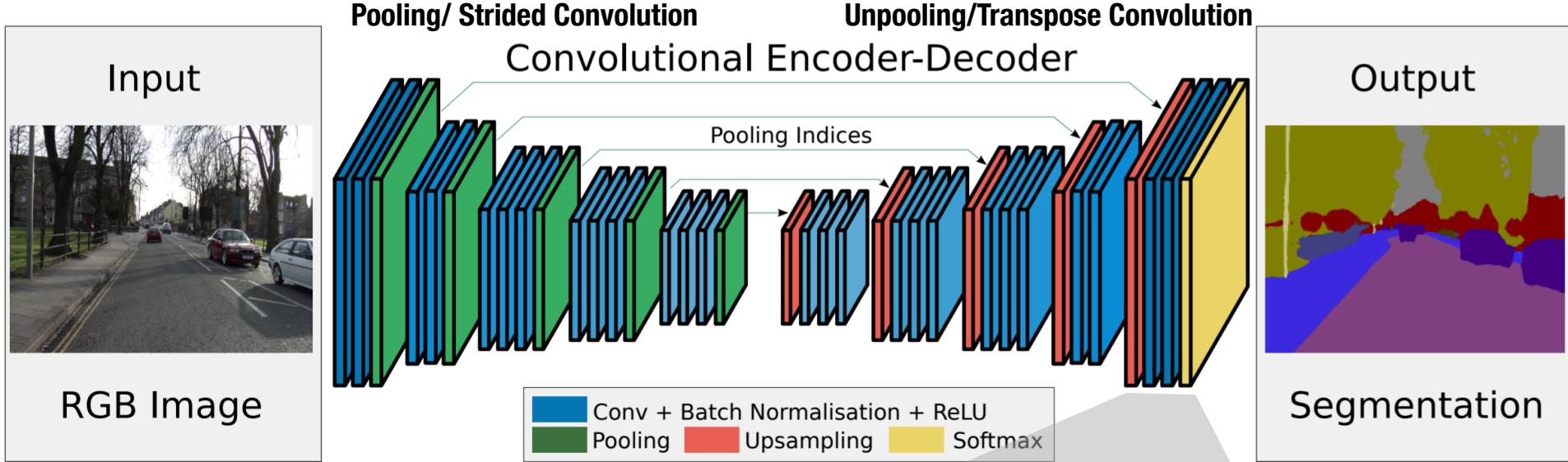
↓



Transpose Convolution  
Unpool + Convolution

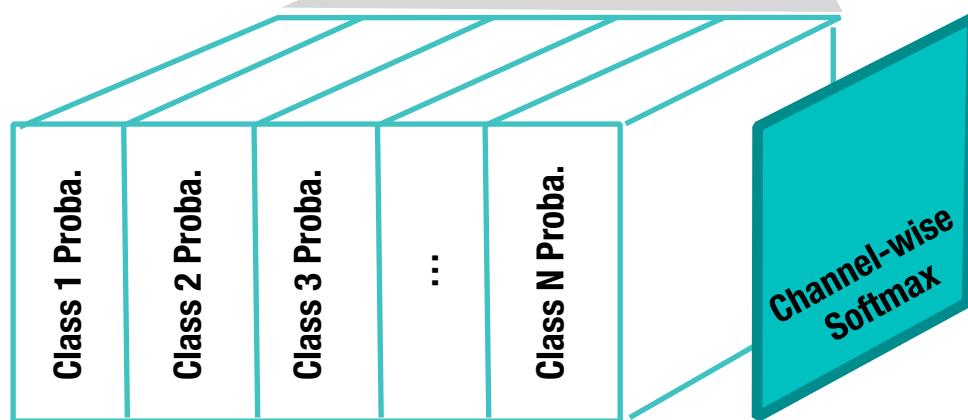


# Putting it all together



## Self Test:

Does it change the architecture if the Image input size changes?

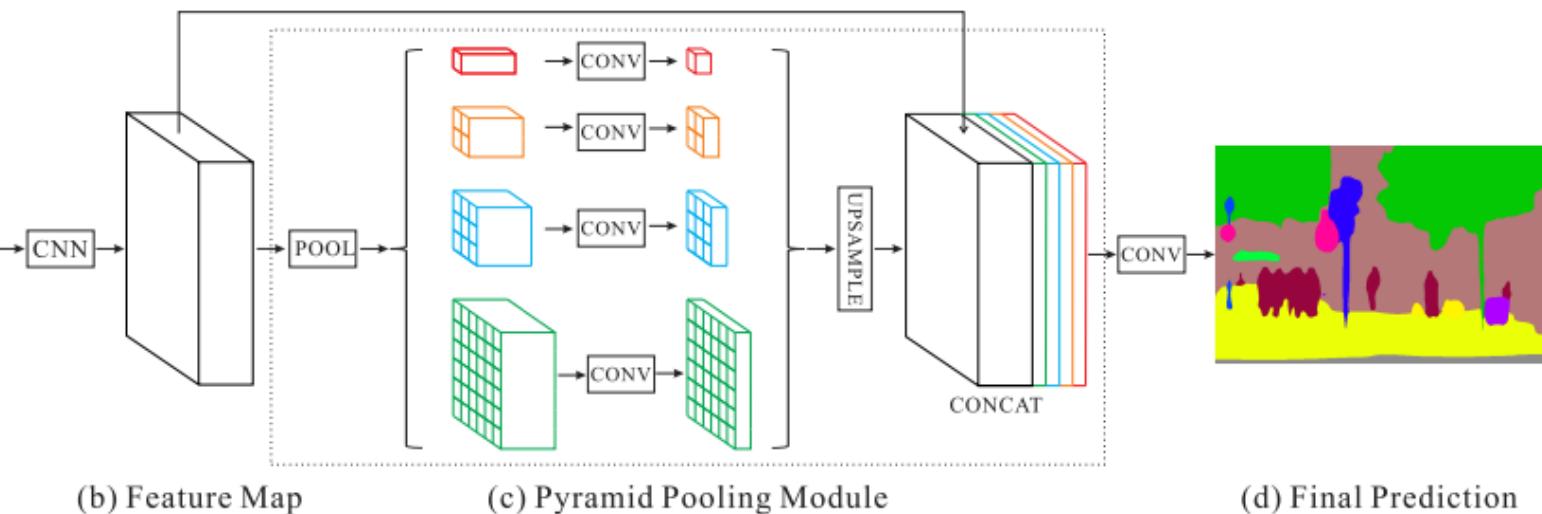


# Some Examples

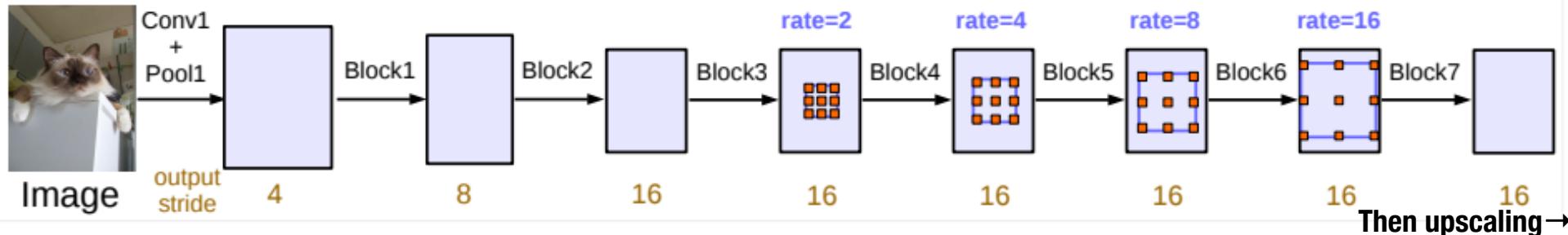
## Pyramid Scene Parsing Network (PSPNet)



(a) Input Image



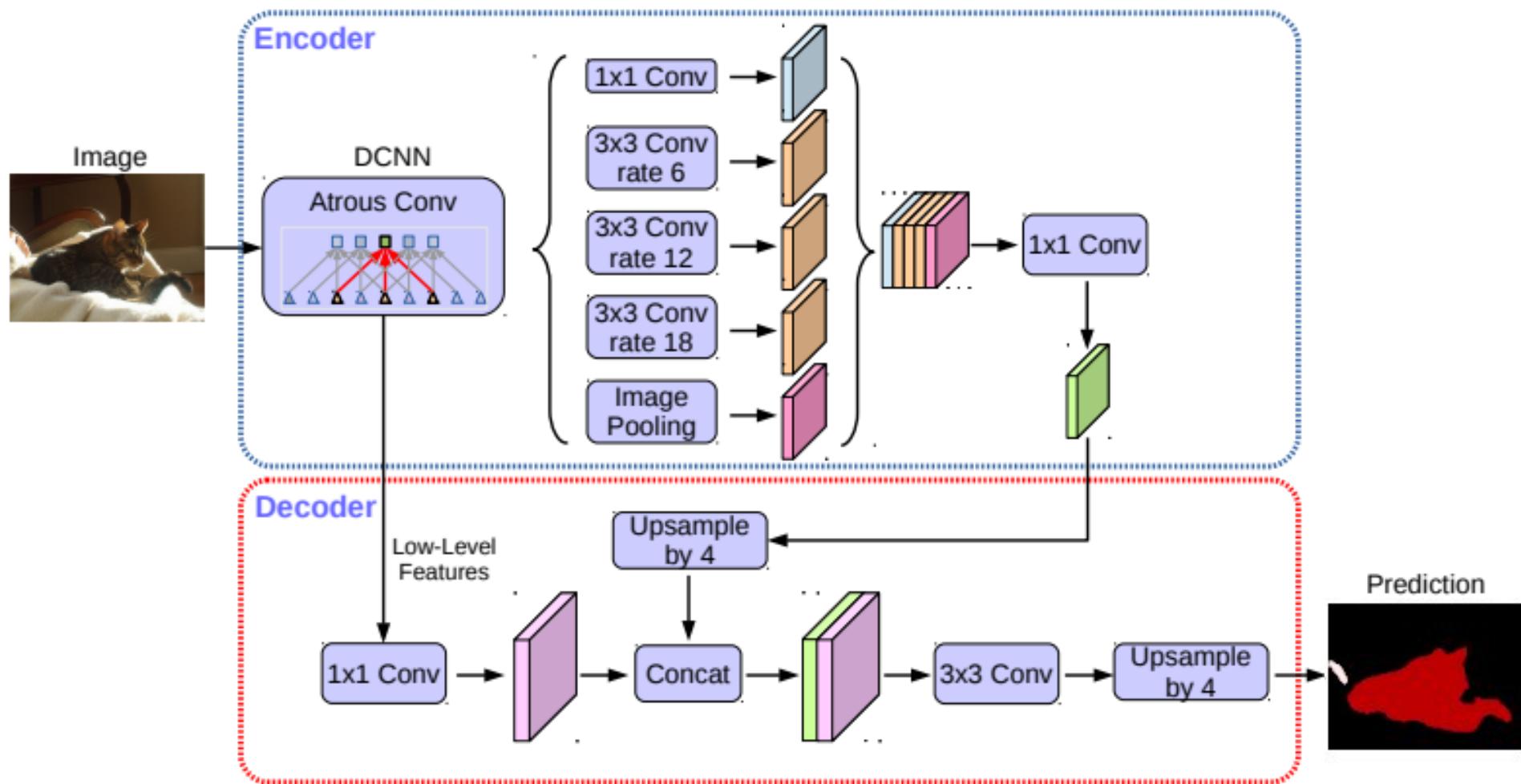
## DeepLabV3: Dilated Convolutions (Atrous Convolutions)



<https://towardsdatascience.com/semantic-segmentation-with-deep-learning-a-guide-and-code-e52fc8958823>



# DeepLabV3+



<https://towardsdatascience.com/semantic-segmentation-with-deep-learning-a-guide-and-code-e52fc8958823>



# Object Segmentation

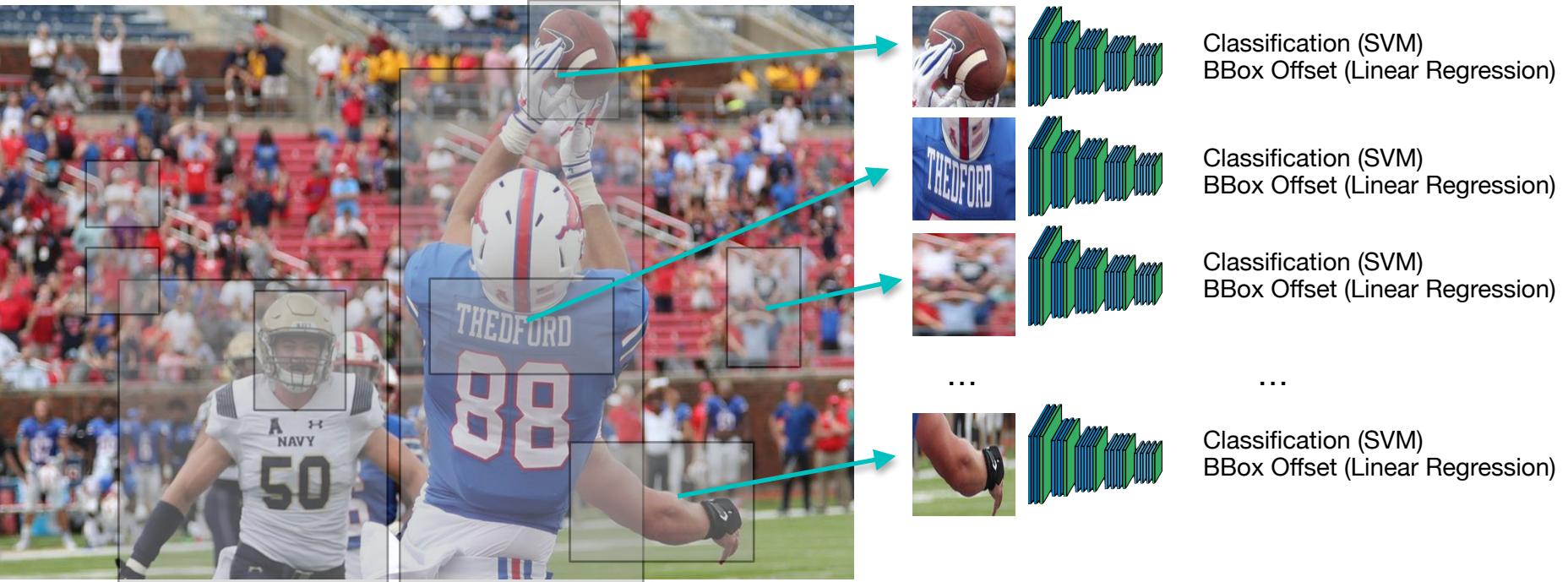


**Research!**

**A history in naming one network five different times  
with five different papers  
each time changing one thing about the architecture**



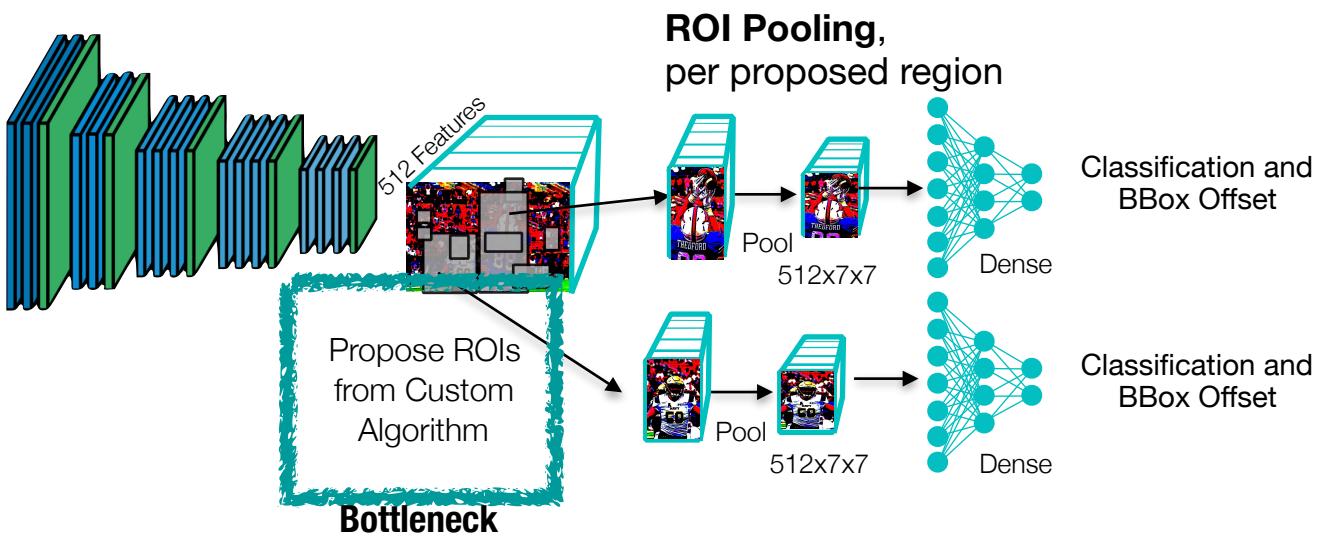
# 2014: R-CNN



- Too Slow to Be Useful
- SVM and BBox Regression Trained Separately
- Fine Tuned Existing ConvNet (for Warped Images)
- ~50 Seconds per Image when Deployed



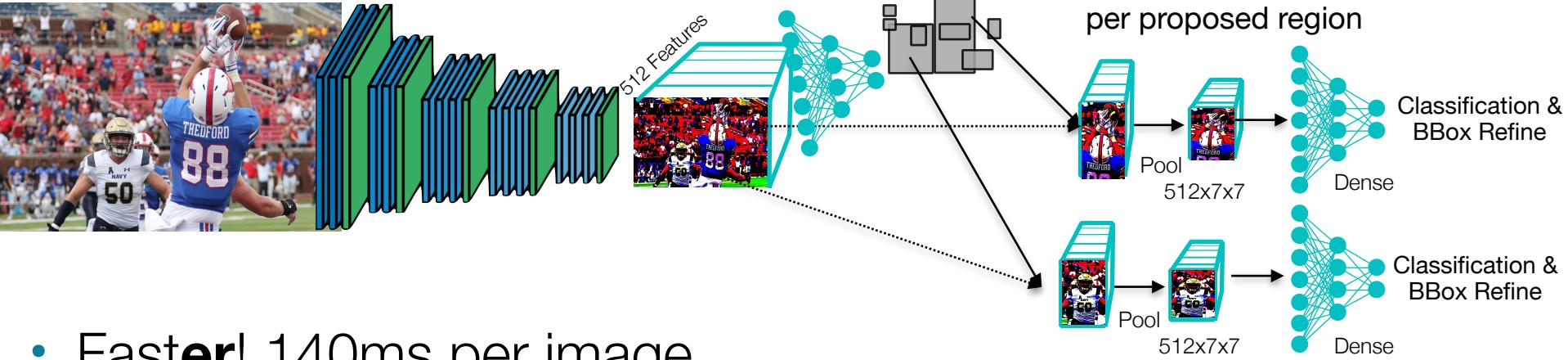
# 2015: Fast R-CNN



- Fast! 2.3 seconds per image (not ~50)
- But still not real time...



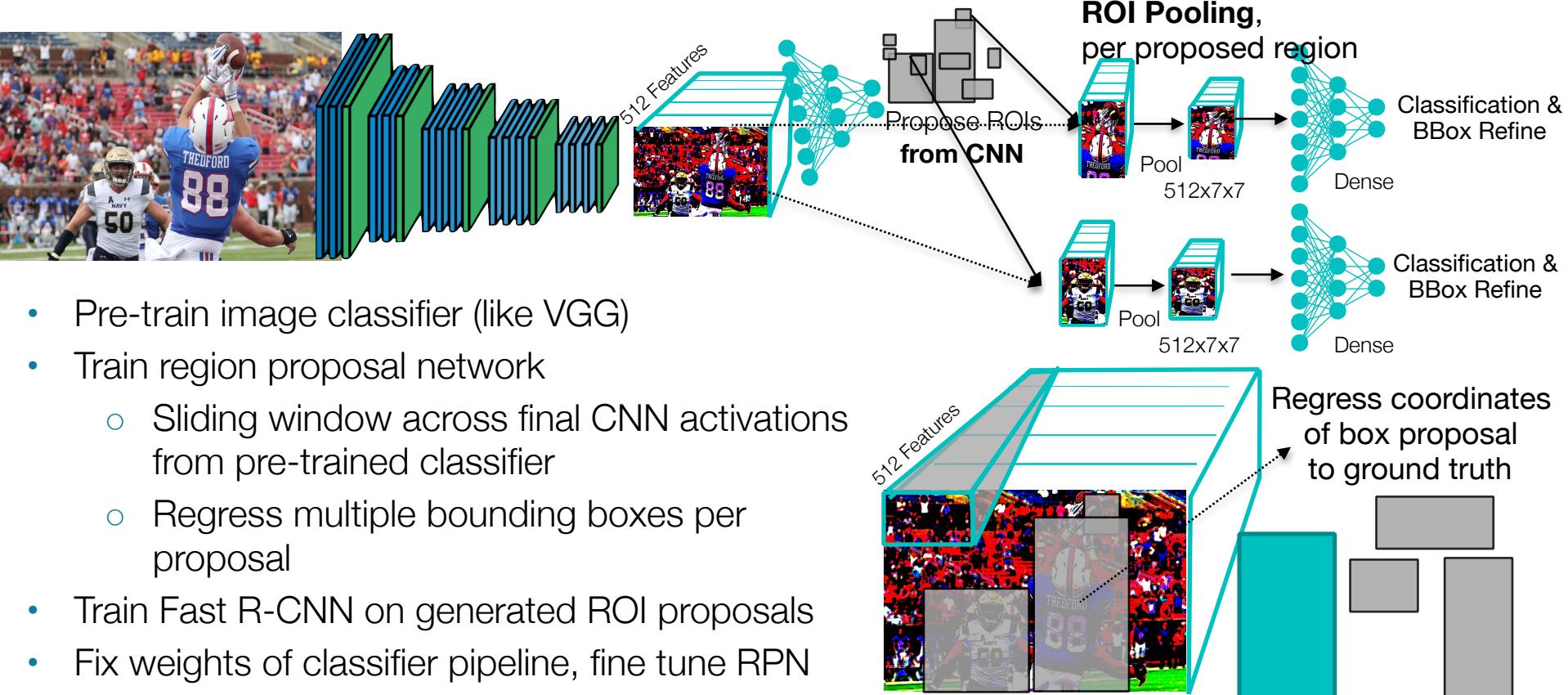
# 2015: Faster R-CNN



- **Faster!** 140ms per image (7 FPS)
- Highly Accurate



# 2015: Faster R-CNN, Training



$$l_{box} = \sum_i \mathbf{1}_i \left[ (x - \hat{x}_i)^2 + (y - \hat{y}_i)^2 + (\log w - \log \hat{w}_i)^2 + (\log h - \log \hat{h}_i)^2 \right]$$

$$l_{class} = \sum_c \text{entropy}(c, \hat{c})$$



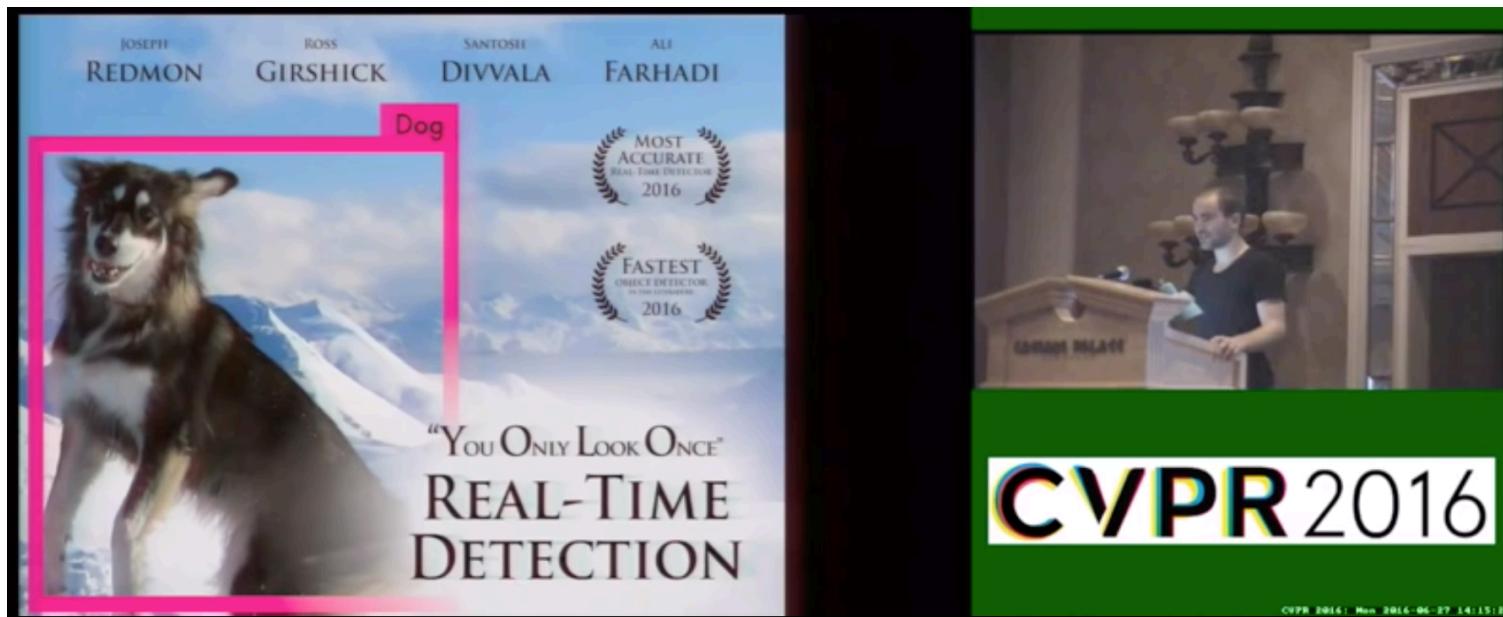
# In Parallel...

- Some of the same researchers started working on a similar, but radically different output approach



# Don't want to listen to me explain it?

- Check out Joseph Redmon's Talk at CVPR 2016:
  - <https://www.youtube.com/watch?v=NM6lrxy0bxS>
  - This is how you give a Technical presentation, plus he is from the school where I did graduate school... so he is clearly superior by grace of proximity...



# 2018: Everybody has a Gimmick



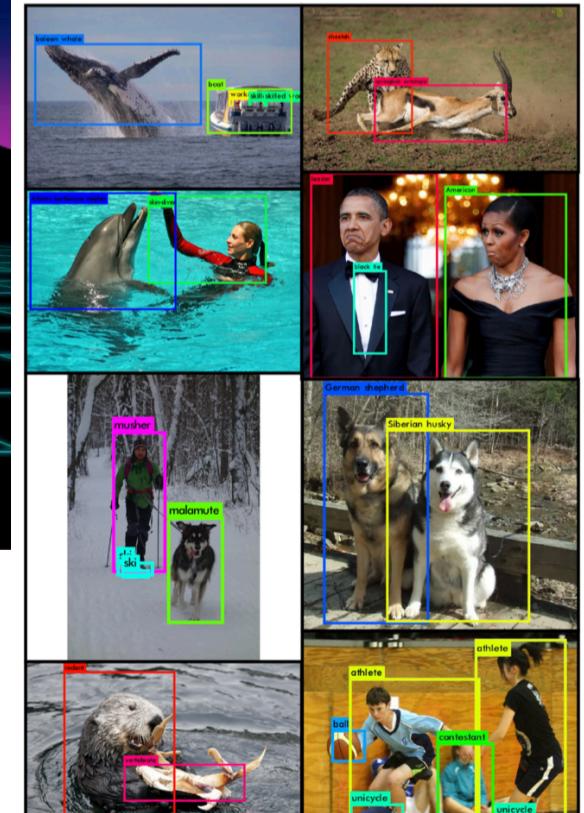
- YOLO ~40-60 FPS
- Slightly more Accurate than **Faster R-CNN**

## YOLO9000: Better, Faster, Stronger

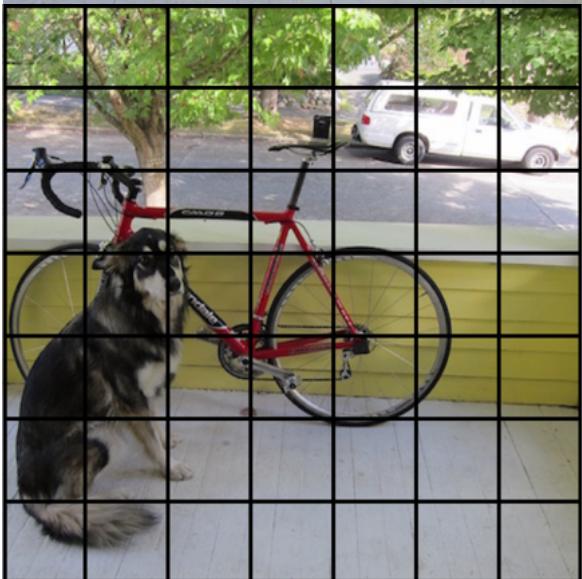
Joseph Redmon<sup>\*†</sup>, Ali Farhadi<sup>\*†</sup>

University of Washington<sup>\*</sup>, Allen Institute for AI<sup>†</sup>

<http://pjreddie.com/yolo9000/>

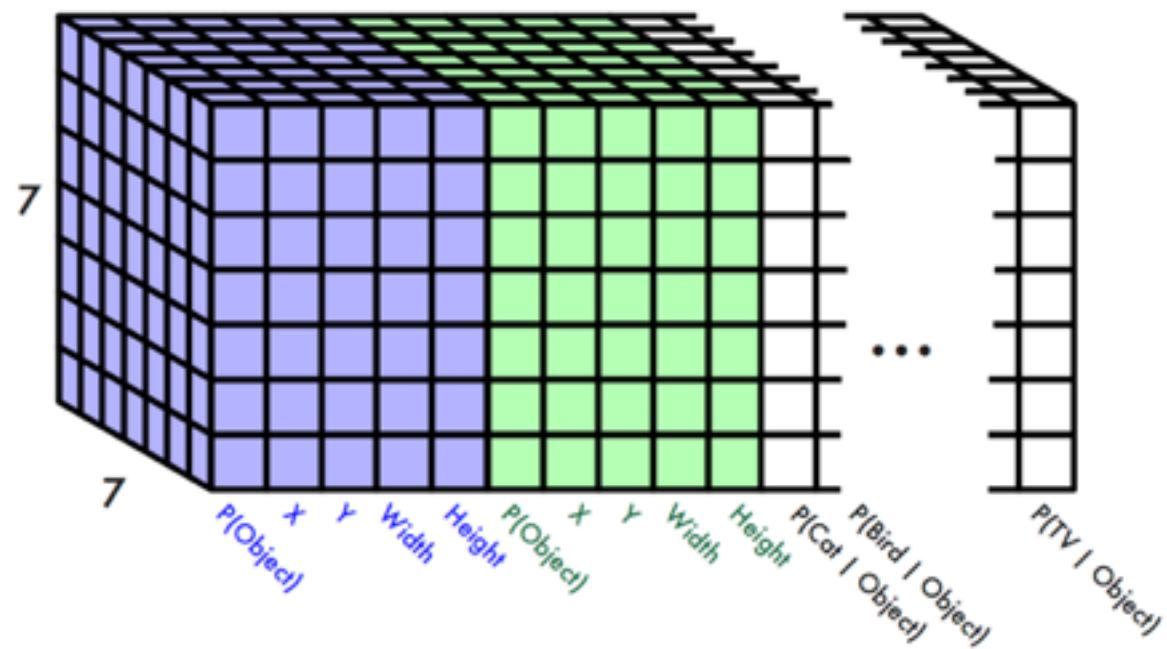


# The YOLO Output Tensor



## The Output Tensor

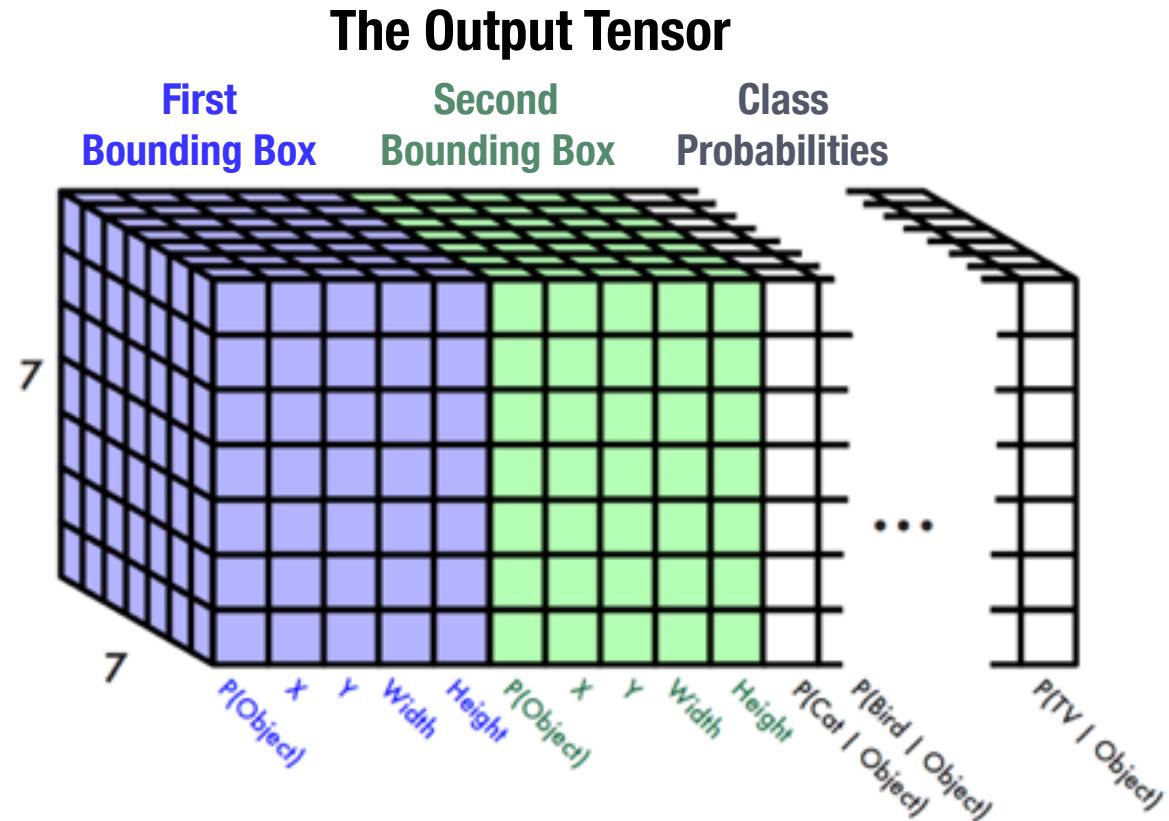
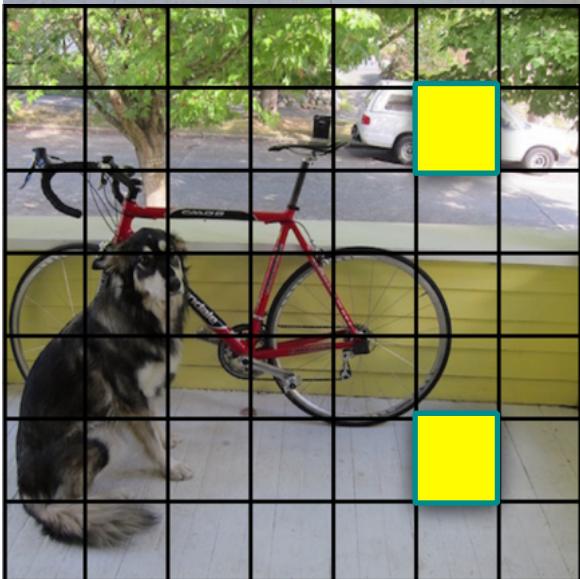
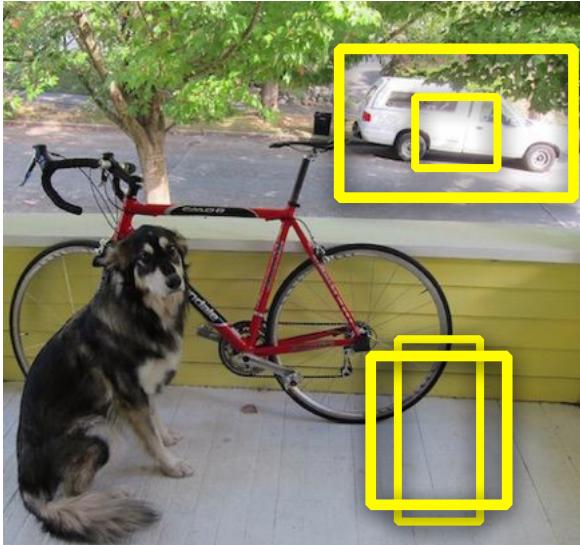
First Bounding Box      Second Bounding Box      Class Probabilities



Redmon and Farhadi, YOLO9000: Better, Faster, Stronger, 2016,  
December 25 — Merry Christmas?



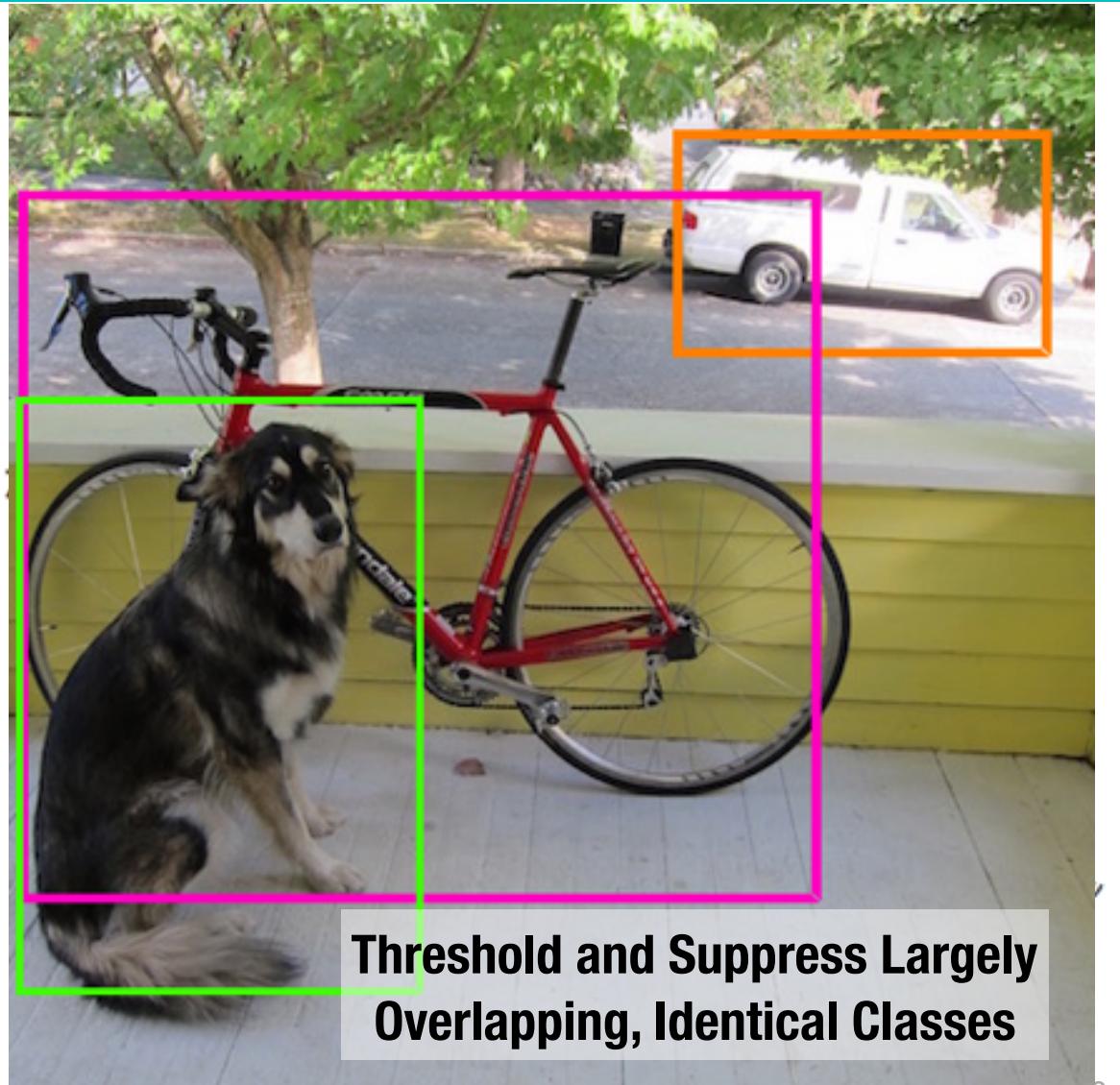
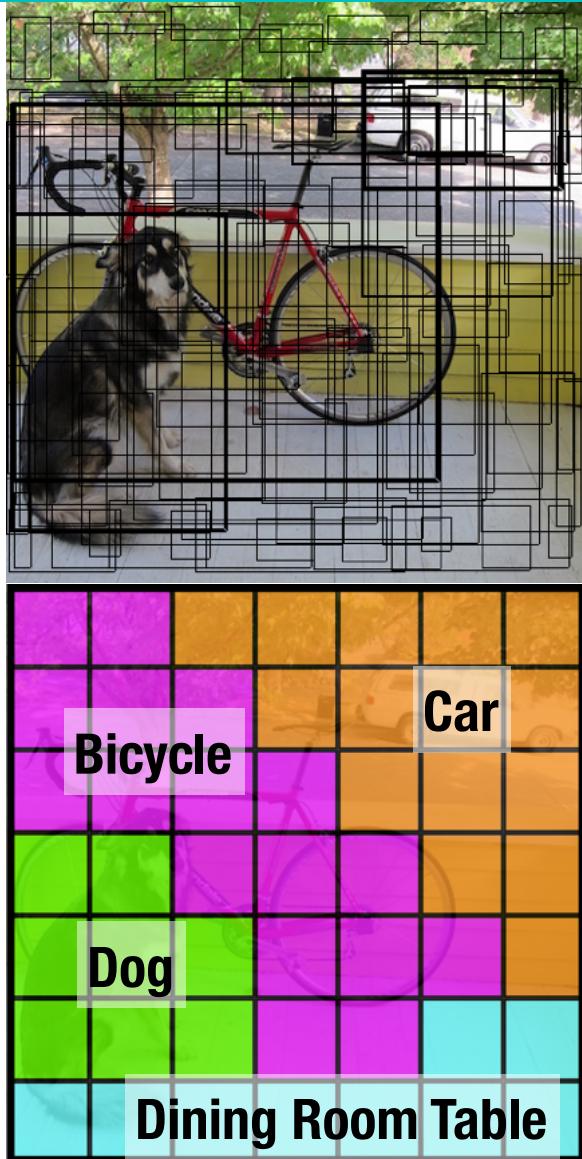
# The YOLO Output Tensor



Redmon and Farhadi, YOLO9000: Better, Faster, Stronger, 2016,  
December 25 — Merry Christmas?

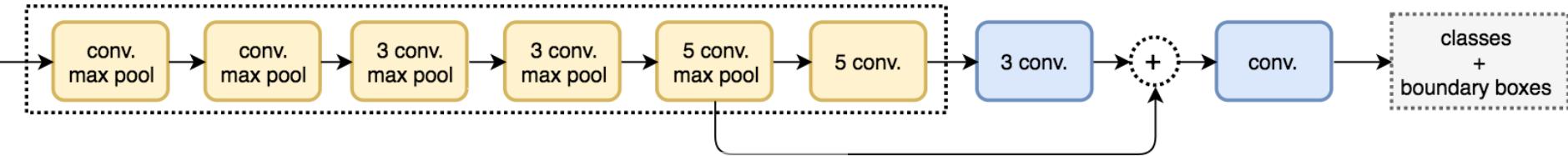


# The YOLO Output Tensor



# The YOLO Architecture

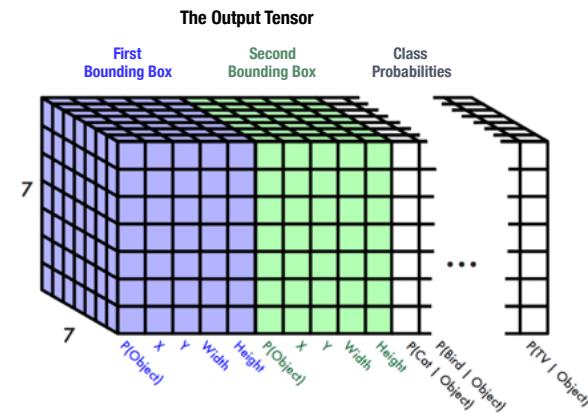
DarkNet



**Trained from Traditional Image Dataset.  
Architecture usually: DarkNet on ImageNet**

	pjreddie guys one of my beehives died :-(
	guys one of my beehives died :-(
	SELU activation and yolo openimages
	GUYS I THINK MAYBE IT WAS BROKEN ON OPENCV IDK
	YO DAWG, I HEARD YOU LIKE LICENSES
	generate own license, totally legal :verified:

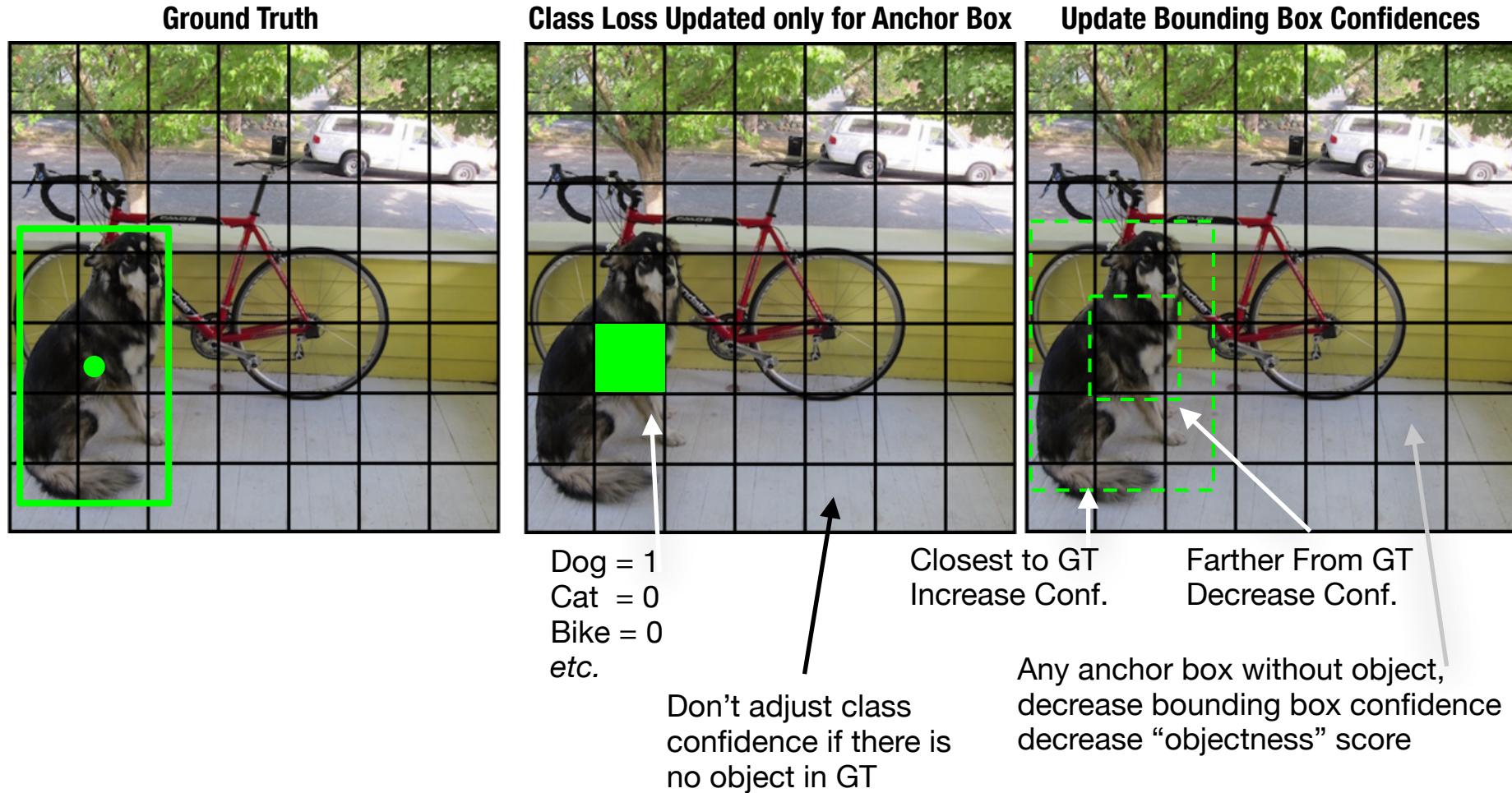
**Last layers:  
Trained on images  
with bounding boxes**



[https://medium.com/@jonathan\\_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088](https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088)

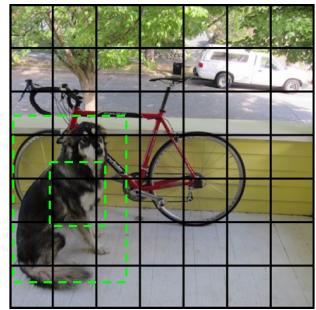


# Training the YOLO Architecture

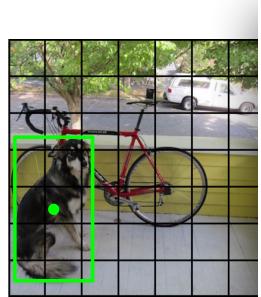


# The YOLO Loss Function

## Update Bounding Box



$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_{ij})^2 + (y_i - \hat{y}_{ij})^2 \right] \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_{ij}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{ij}})^2 \right]$$



$S \times S$  cells,  $i^{\text{th}}$  cell

$B$  boxes per cell,  $j^{\text{th}}$  box

$\mathbb{1}^{\text{obj}}$  indicator function, from GT

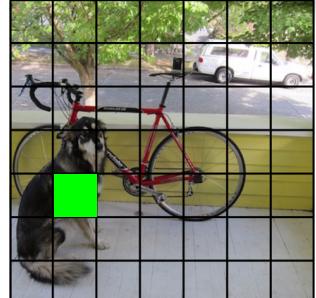
$\hat{C}$  is confidence per box

$\hat{p}(c)$  softmax output, per class

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (\hat{C}_i - \hat{C}_{ij})^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (\hat{C}_i - \hat{C}_{ij})^2$$

## Class Loss



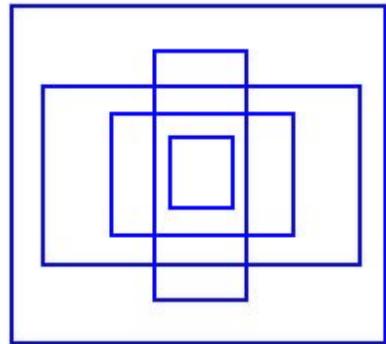
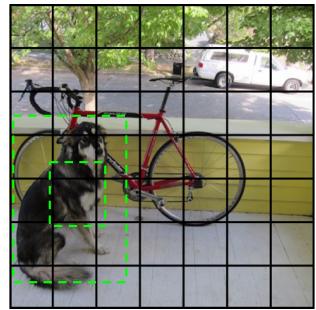
$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

[https://medium.com/@jonathan\\_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088](https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088)

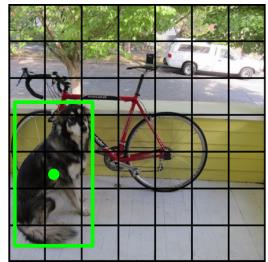


# Updated YOLO Localization

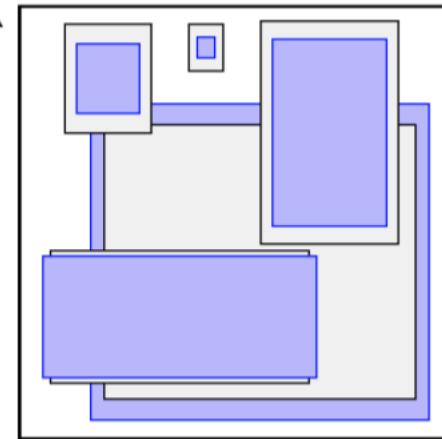
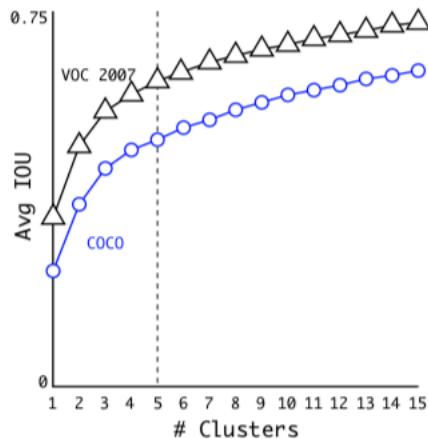
## Update Bounding Box



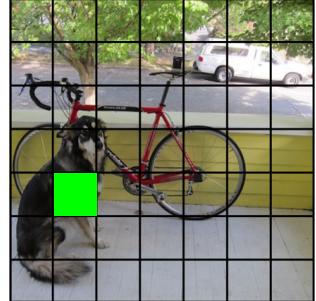
- Define 5 pre-defined box shapes
- Regress  $x, y$  offset from cell center
- Bound  $x$  and  $y$  to the bounds of the cell
- And  $w$  and  $h$  scaling of predefined shape



“Good” Shape Priors  
Found via Clustering



## Class Loss



$$\begin{aligned} & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_{ij} \right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_{ij} \right)^2 + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \end{aligned}$$

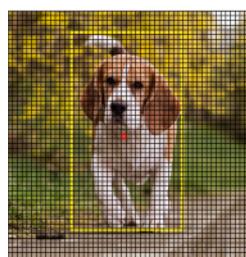
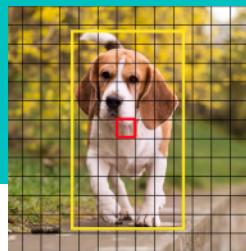
[https://medium.com/@jonathan\\_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088](https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088)

## Localization Loss

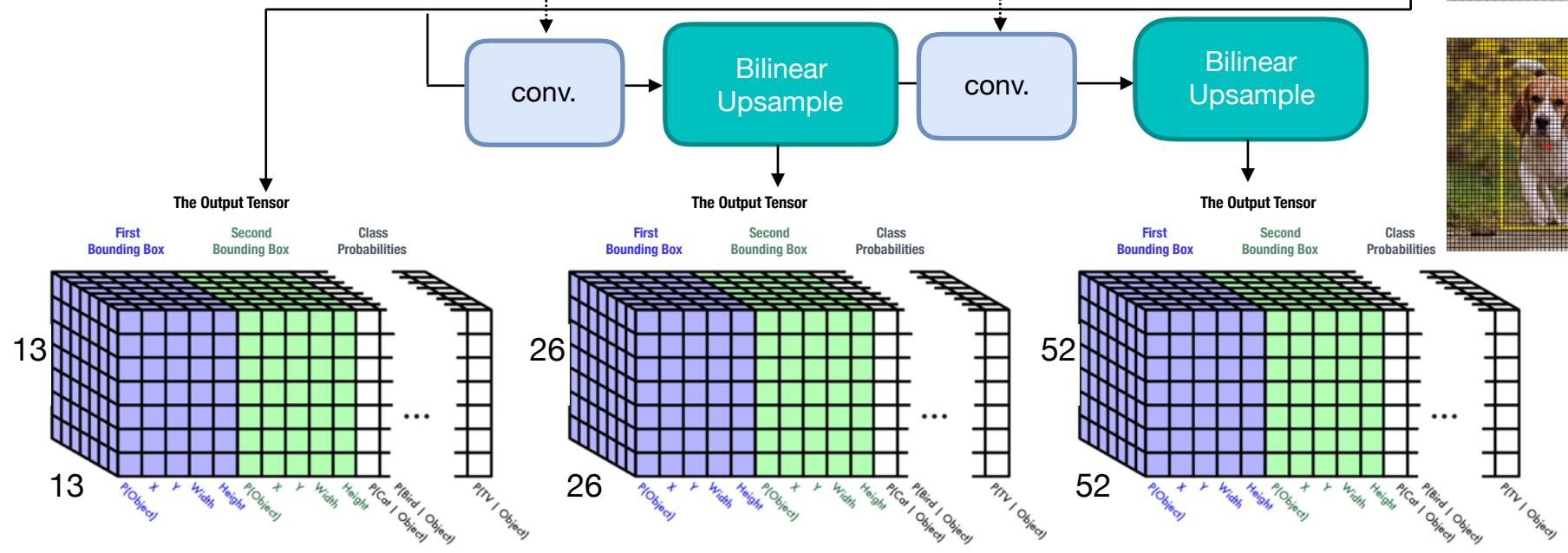
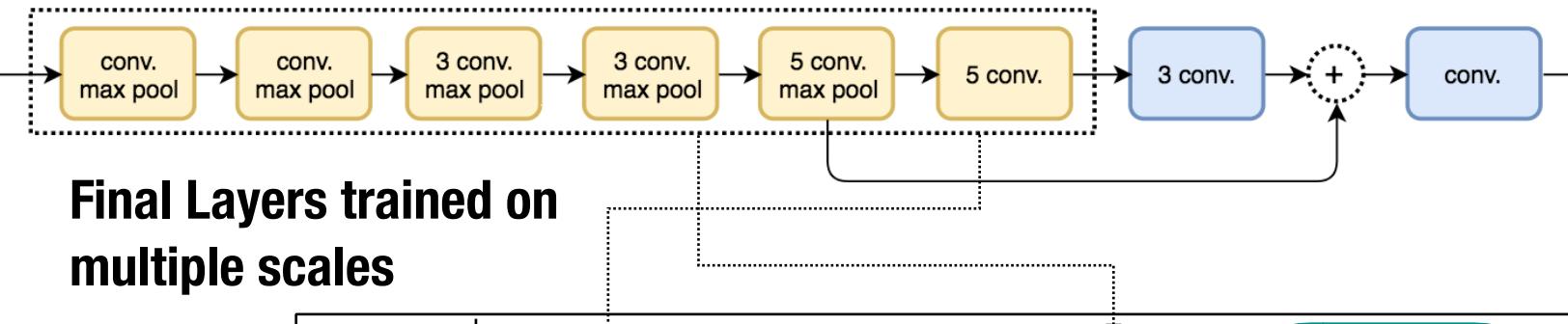
## Classification Loss



# The YOLOv3 Architecture



**Final Layers trained on  
multiple scales**



[https://medium.com/@jonathan\\_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088](https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088)



# A closing thought from YOLOv3 Report

## Abstract

*We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP<sub>50</sub> in 51 ms on a Titan X, compared to 57.5 AP<sub>50</sub> in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at <https://pjreddie.com/yolo/>.*

But maybe a better question is: “What are we going to do with these detectors now that we have them?” A lot of the people doing this research are at Google and Facebook. I guess at least we know the technology is in good hands and definitely won’t be used to harvest your personal information and sell it to.... wait, you’re saying that’s exactly what it will be used for?? Oh.

Well the other people heavily funding vision research are the military and they’ve never done anything horrible like killing lots of people with new technology oh wait.....<sup>1</sup>

I have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it, like counting the number of zebras in a national park [13], or tracking their cat as it wanders around their house [19]. But computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it. We owe the world that much.

In closing, do not @ me. (Because I finally quit Twitter).

---

<sup>1</sup>The author is funded by the Office of Naval Research and Google.



# A closing thought from YOLOv3 Report

## The Rebuttal I wish I could Write:

Reviewer #2 AKA Dan Grossman (lol blinding who does that) insists that I point out here that our graphs have not one but two non-zero origins. You're absolutely right Dan, that's because it looks way better than admitting to ourselves that we're all just here battling over 2-3% mAP. But here are the requested graphs. I threw in one with FPS too because we look just like super good when we plot on FPS.

Reviewer #4 AKA JudasAdventus on Reddit writes "Entertaining read but the arguments against the MSCOCO metrics seem a bit weak". Well, I always knew you would be the one to turn on me Judas. You know how when you work on a project and it only comes out alright so you have to figure out some way to justify how what you did actually was pretty cool? I was basically trying to do that and I lashed out at the COCO metrics a little bit. But now that I've staked out this hill I may as well die on it.

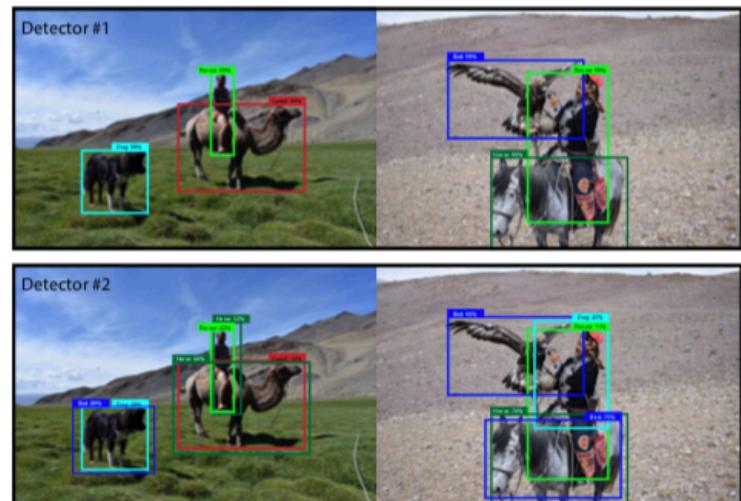
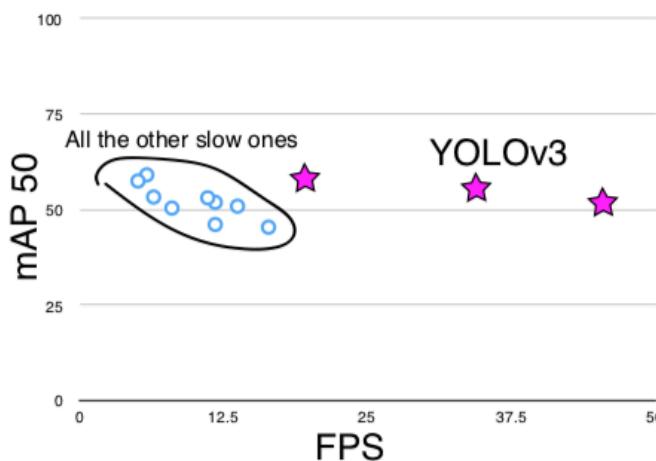


Figure 5. These two hypothetical detectors are perfect according to mAP over these two images. They are both perfect. Totally equal.

Now this is OBVIOUSLY an over-exaggeration of the problems with mAP but I guess my newly retconned point is that there are such obvious discrepancies between what people in the "real world" would care about and our current metrics that I think if we're going to come up with new metrics we should focus on these discrepancies. Also, like, it's already mean average precision, what do we even call the COCO metric, average mean average precision?

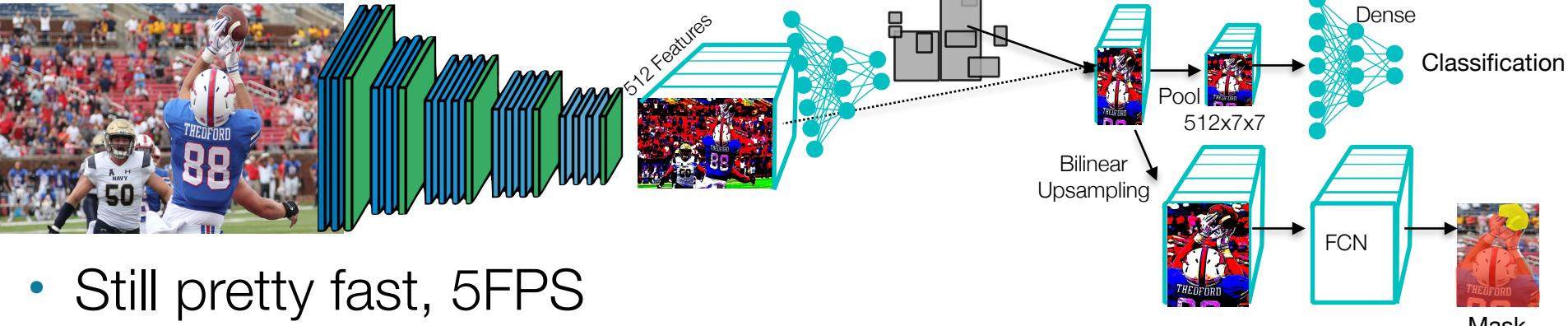
Here's a proposal, what people actually care about is given an image and a detector, how well will the detector find and classify objects in the image. What about getting rid of the per-class AP and just doing a global average precision? Or doing an AP calcu-

lation per-image and averaging over that?

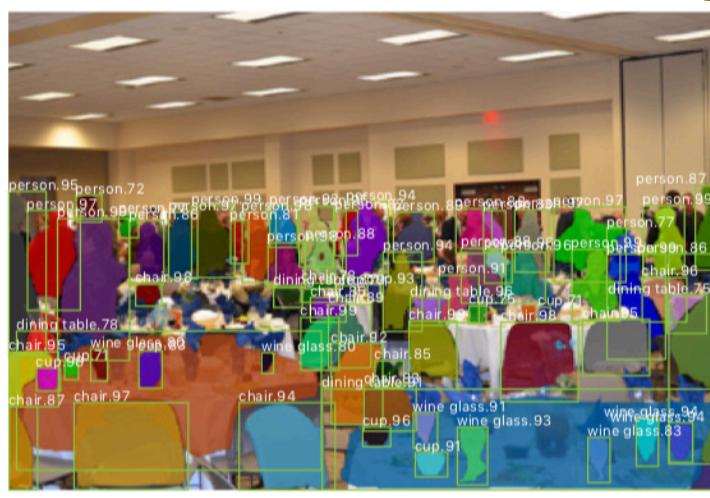
Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them.



# 2018: Mask R-CNN



- Still pretty fast, 5FPS
- Excellent results

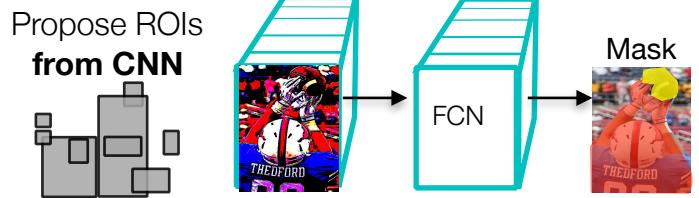


Ren et al. Mask R-CNN, 2018

56



# 2018: Mask R-CNN



Can also provide **key point detection** from same FCN features (not real time, post processed)

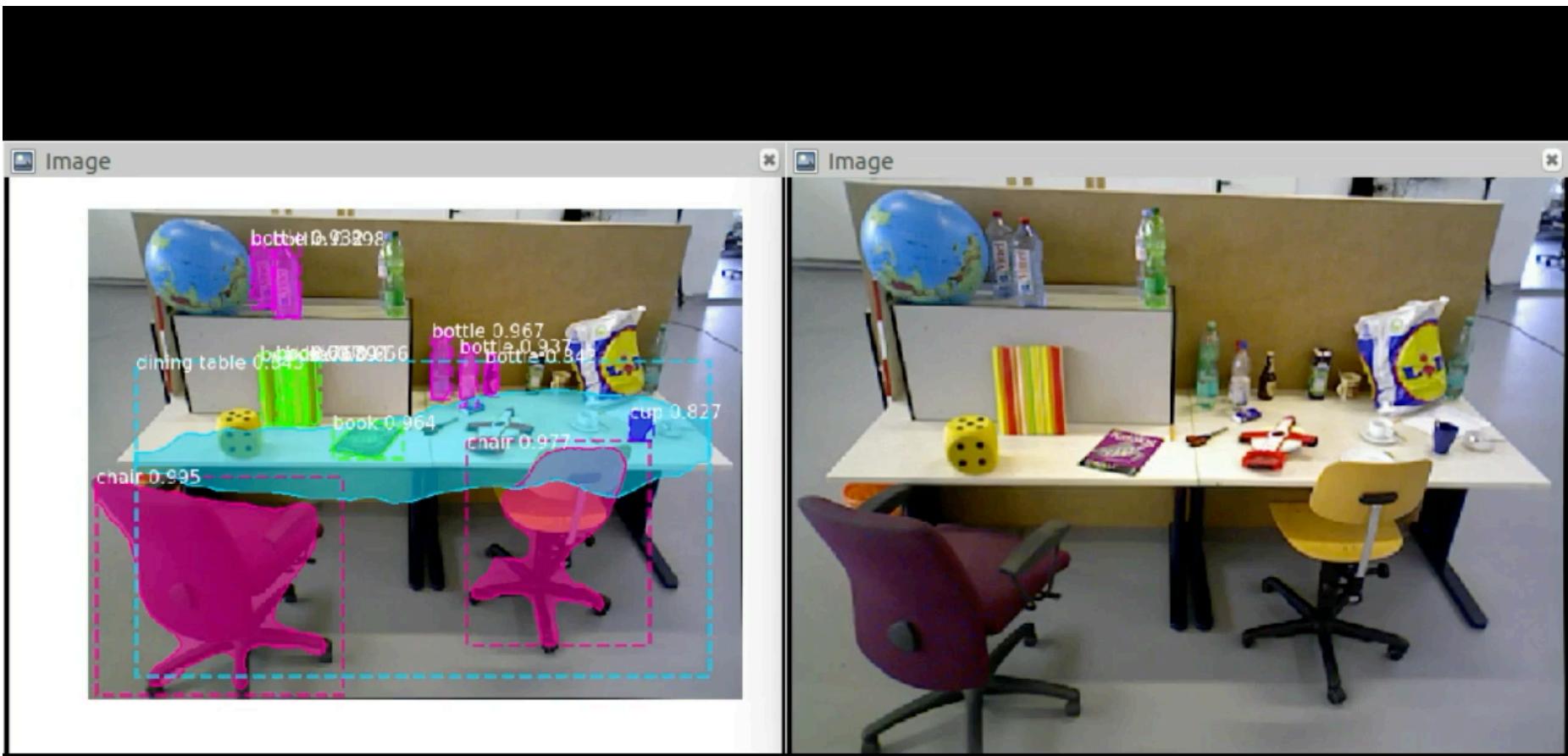
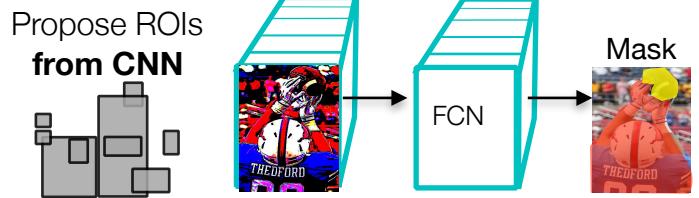


ICCV17



# 2018: Mask R-CNN

- Real time, Mask R-CNN



<https://www.youtube.com/watch?v=nEug0-pD0Ms>



# In summary

- Semantic segmentation through FCN is active research area
  - DeepLabV3 is excellent choice
- Object segmentation is good, ready for use in industry (Apple's ObjectDetector uses YOLO variant)
  - Already deployed in a number of Apps
  - At 60 FPS, supports tracking applications and AR
  - Can backoff to CPU only at about 5 FPS
- Instance Segmentation is ready for deployment in a number of areas, but is still 5 FPS, using a good deal of the hardware





# YOLOv3 in Keras

Excellent implementation with  
... not very many comments

Look at “YOLOv3” and “TrainBottleneck”

Self-Guided: <https://github.com/qzwweeee/keras-yolo3>

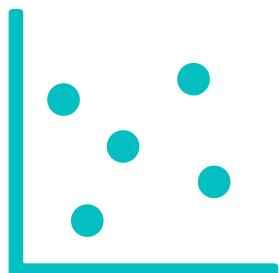
Another Great resource, with Excellent Documentation (PyTorch).  
Step-by-Step Layer Construction and Loss Explanation

<https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/>

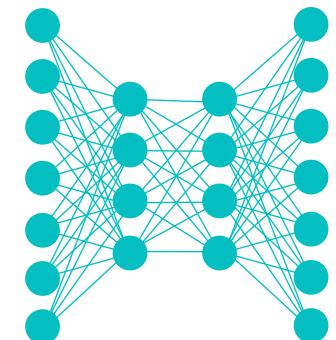


# Lecture Notes for Neural Networks and Machine Learning

Fully Convolutional Learning



**Next Time:**  
Image Style Transfer  
**Reading:** Chollet 8.1– 8.3



# Backup slides



# Title Between Topics



# Example Slide





# Title

Subtitle

Follow Along: Notebook Name

65

