

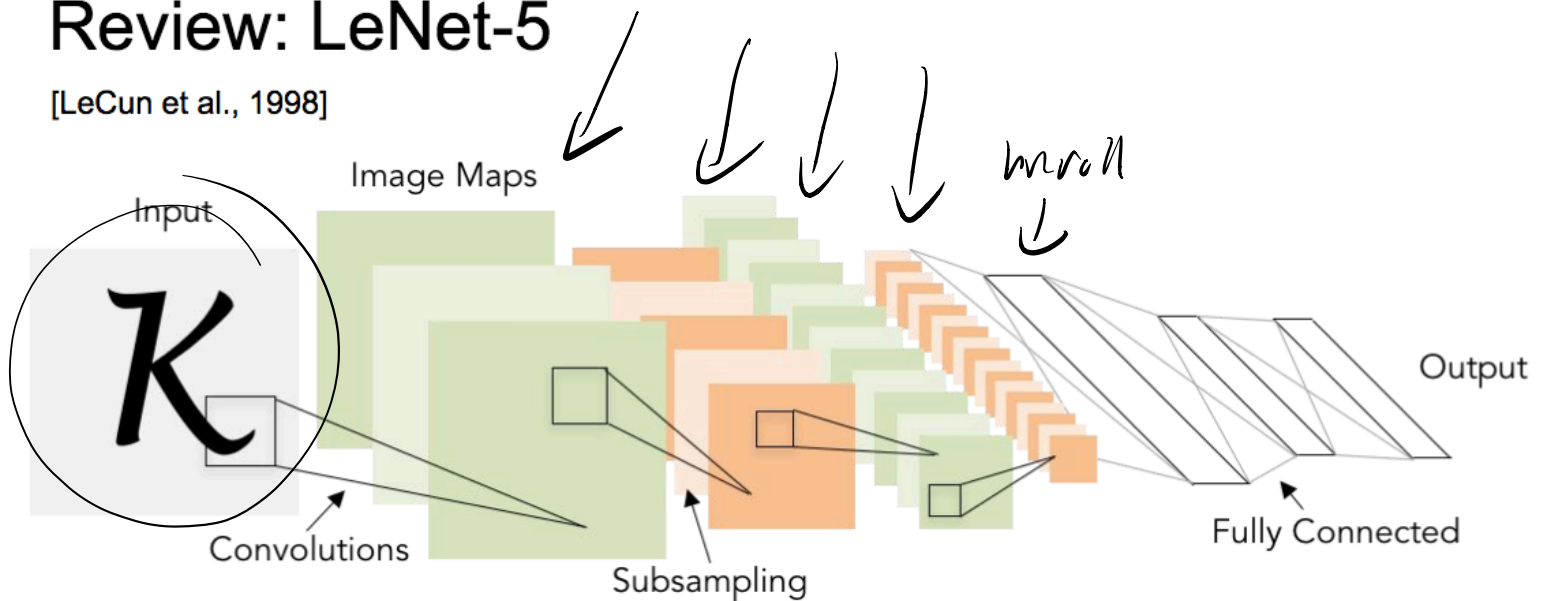


# Announcements

- Material starting today will not be on the final.
- No class tomorrow (happy Thanksgiving!)
- Last week of class – special/fun topics:
  - CNNs:
    - advanced architectures
    - other problems: object detection, style transfer  
deep dream...
  - (fast) bilateral filtering

# Review: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1  
Subsampling (Pooling) layers were 2x2 applied at stride 2  
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]



# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

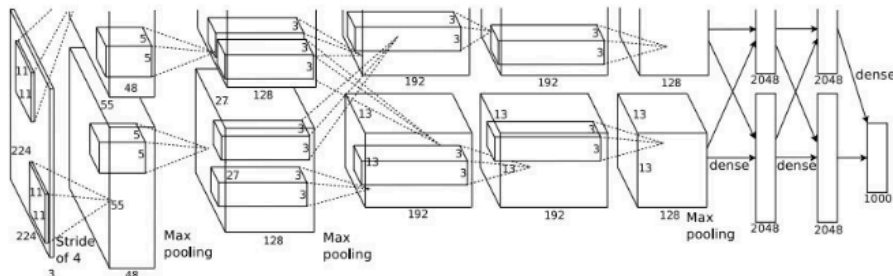
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)



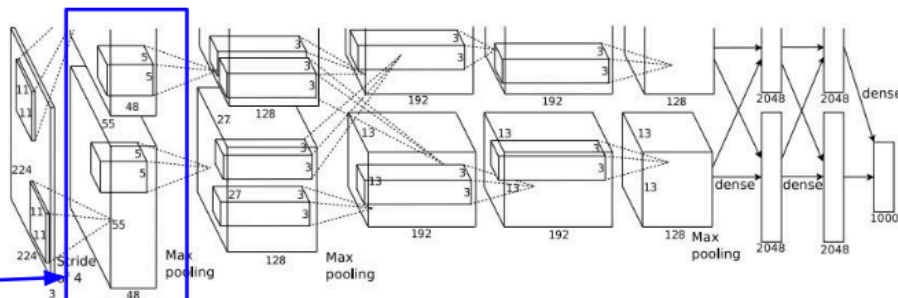
## Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- CNN ensemble: 18.2% -> 15.4%

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

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

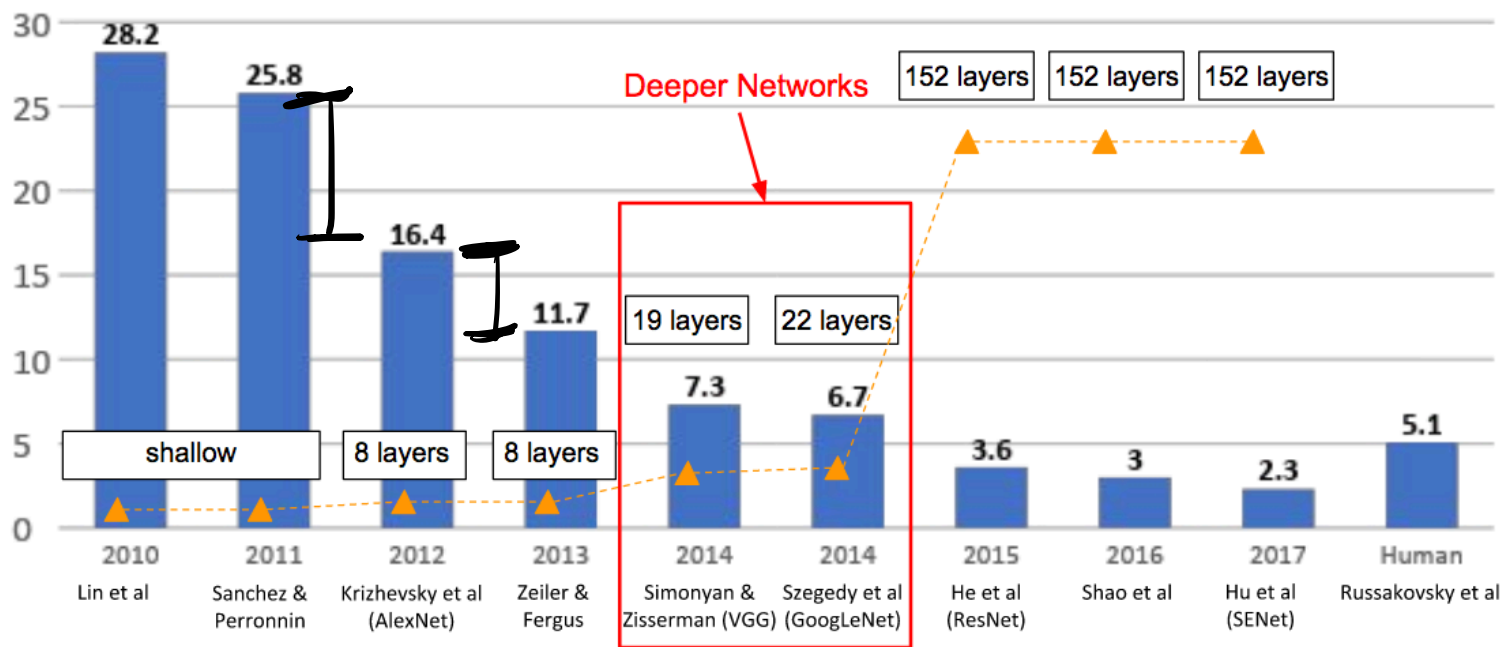
[1000] FC8: 1000 neurons (class scores)

[55x55x48] x 2

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

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

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners







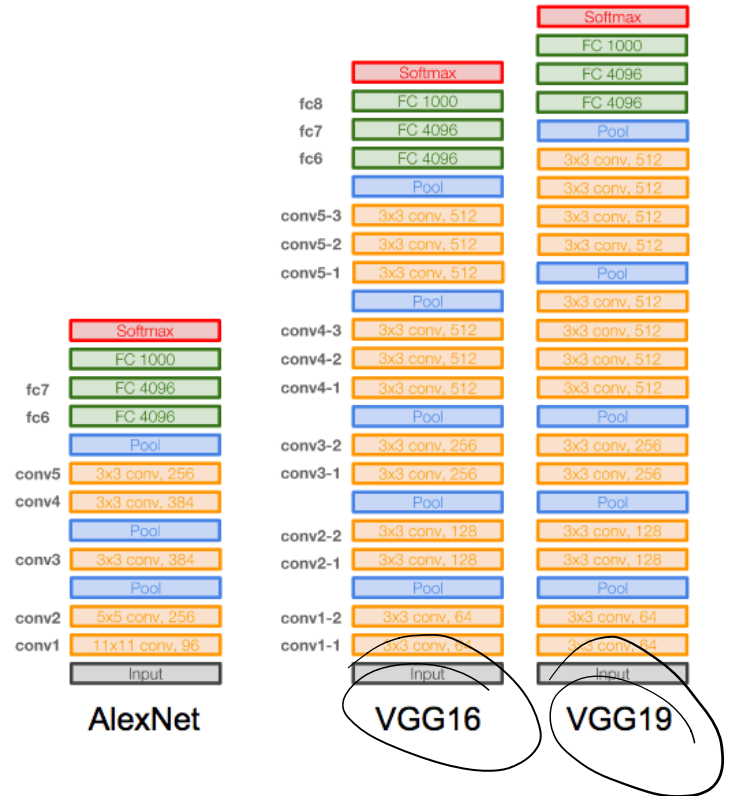


# Case Study: VGGNet

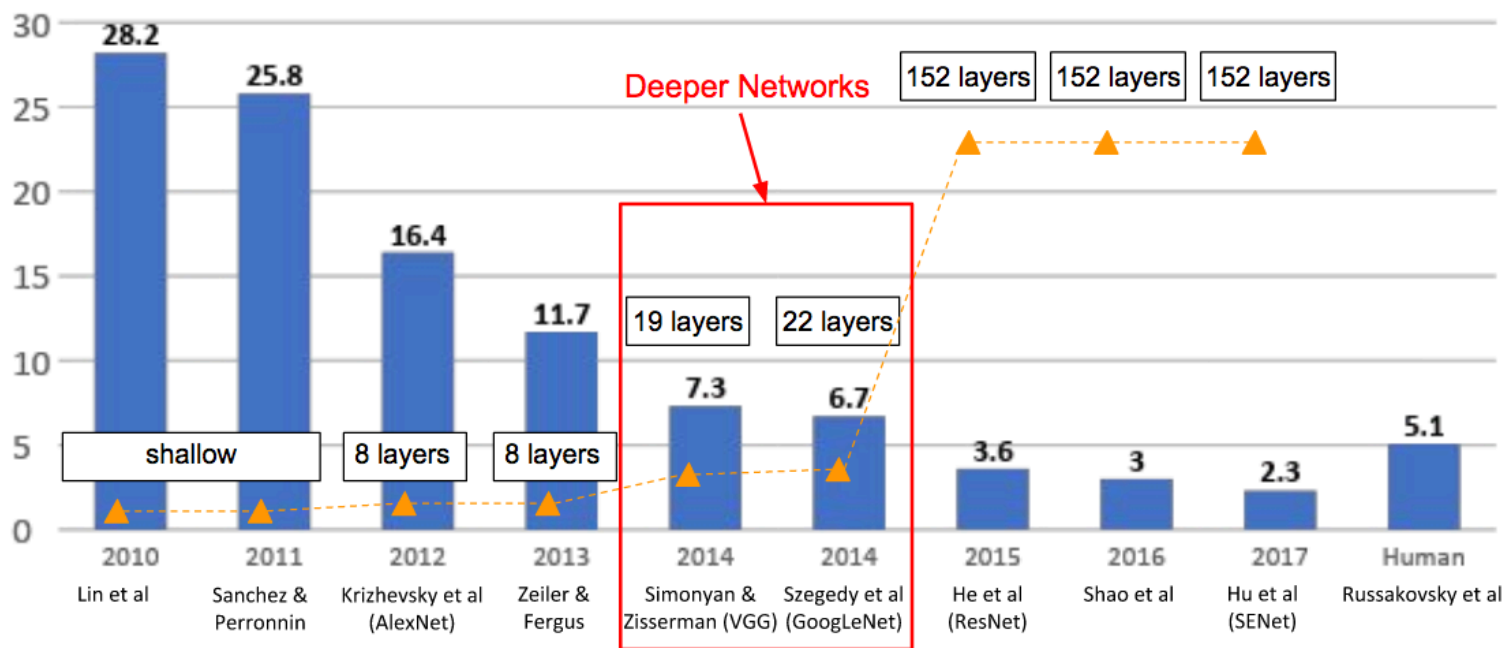
[Simonyan and Zisserman, 2014]

## Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



## ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners





**WE NEED TO GO  
DEEPER**

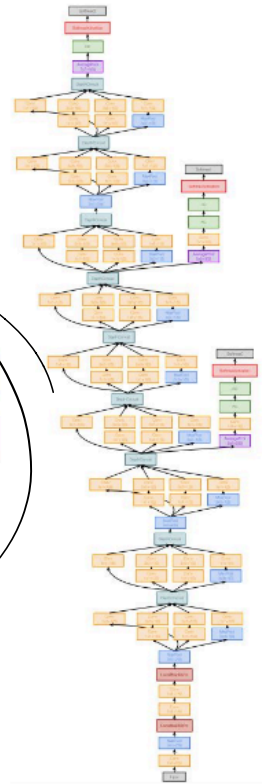
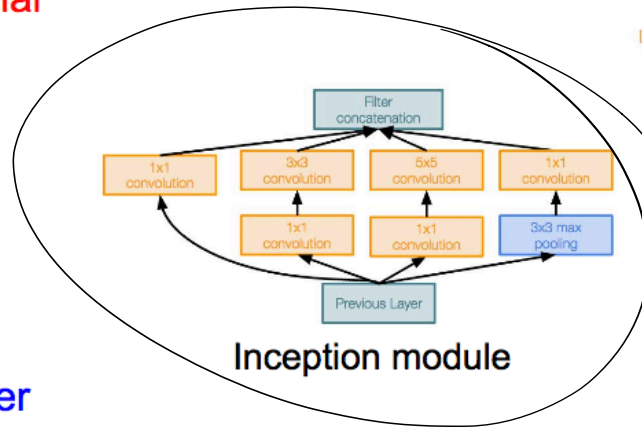


# Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

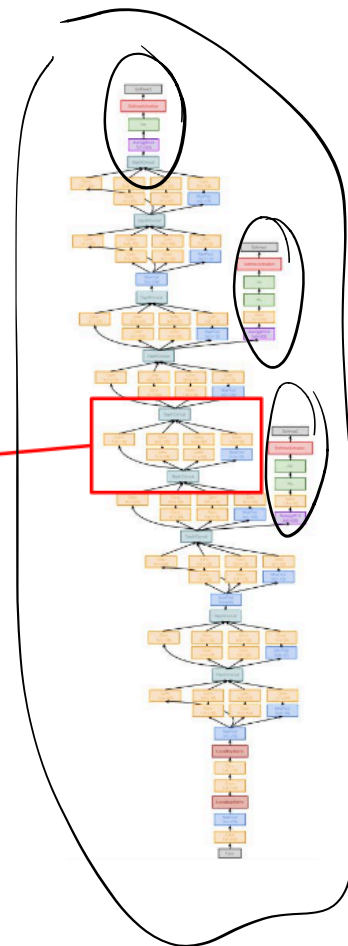
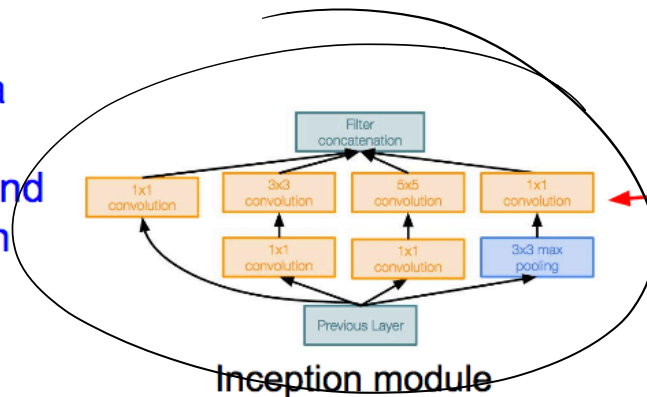
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!  
12x less than AlexNet
- ILSVRC’14 classification winner  
(6.7% top 5 error)



# Case Study: GoogLeNet

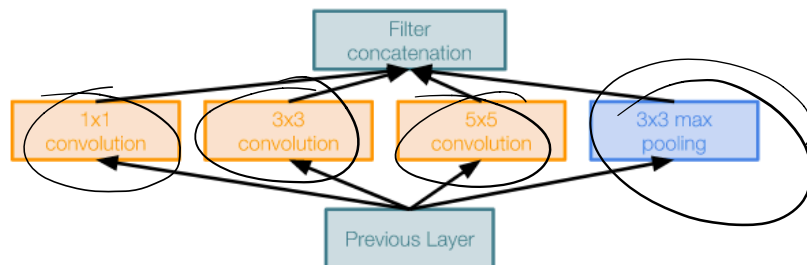
[Szegedy et al., 2014]

“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



# Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise



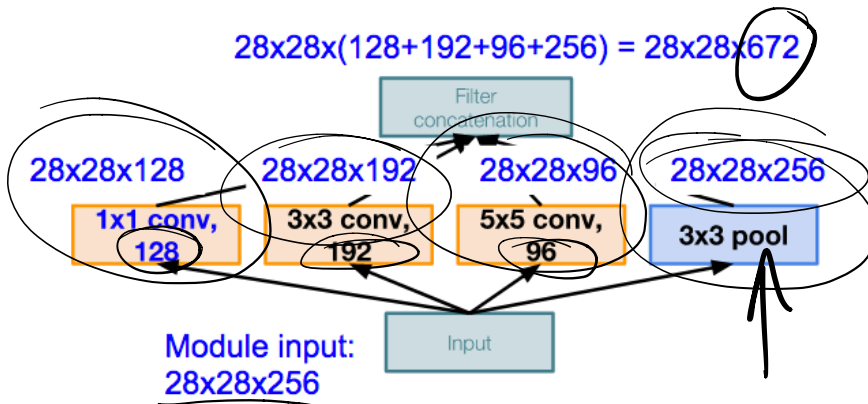
# Case Study: GoogLeNet

[Szegedy et al., 2014]

**Example:**

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Naive Inception module

Q: What is the problem with this?  
[Hint: Computational complexity]

**Conv Ops:**

[1x1 conv, 128]  $28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3x3 conv, 192]  $28 \times 28 \times 192 \times 3 \times 3 \times 256$

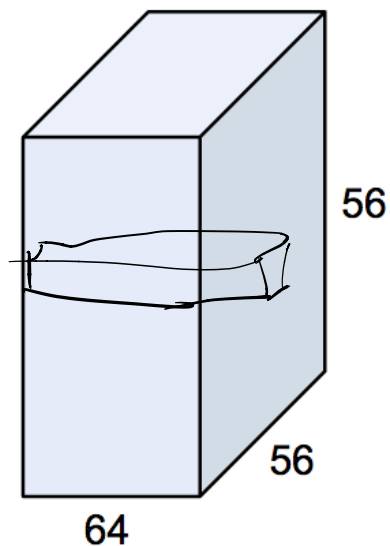
[5x5 conv, 96]  $28 \times 28 \times 96 \times 5 \times 5 \times 256$

**Total: 854M ops**

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

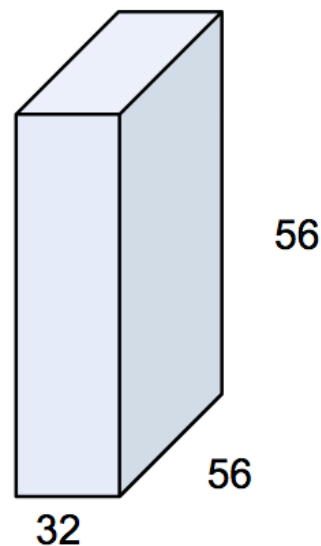
## Reminder: 1x1 convolutions



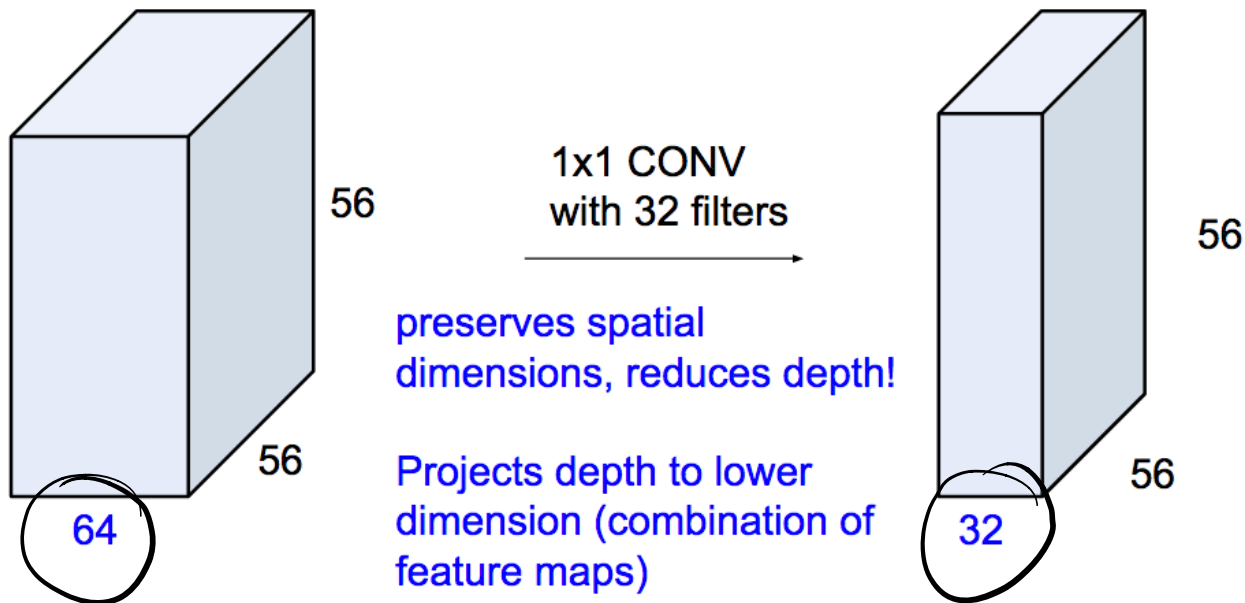
1x1 CONV  
with 32 filters

→

(each filter has size  
1x1x64, and performs a  
64-dimensional dot  
product)



## Reminder: 1x1 convolutions



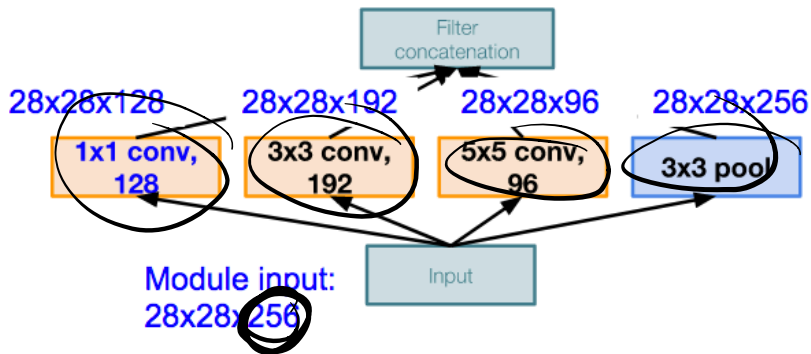
# Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

$$28 \times 28 \times (128 + 192 + 96 + 256) = 529k$$



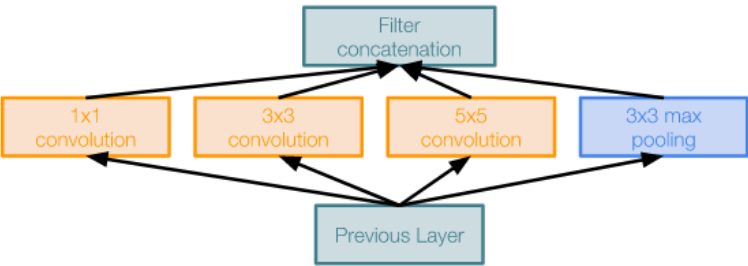
Naive Inception module

Q: What is the problem with this?  
[Hint: Computational complexity]

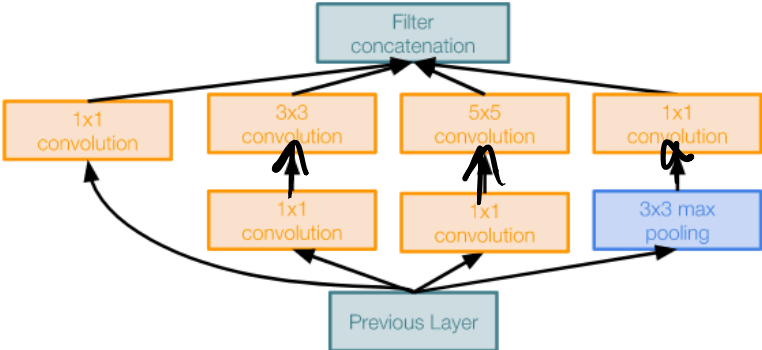
Solution: "bottleneck" layers that use  $1 \times 1$  convolutions to reduce feature depth

# Case Study: GoogLeNet

[Szegedy et al., 2014]



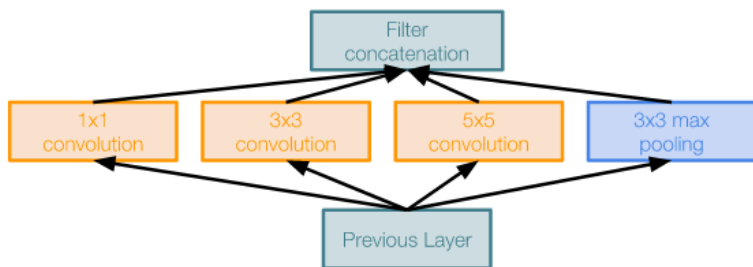
Naive Inception module



Inception module with dimension reduction

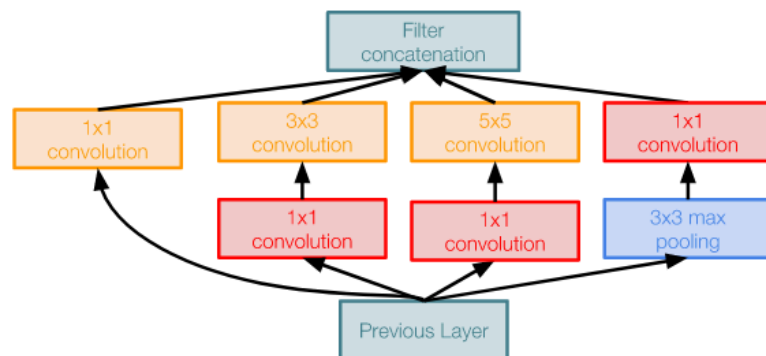
# Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

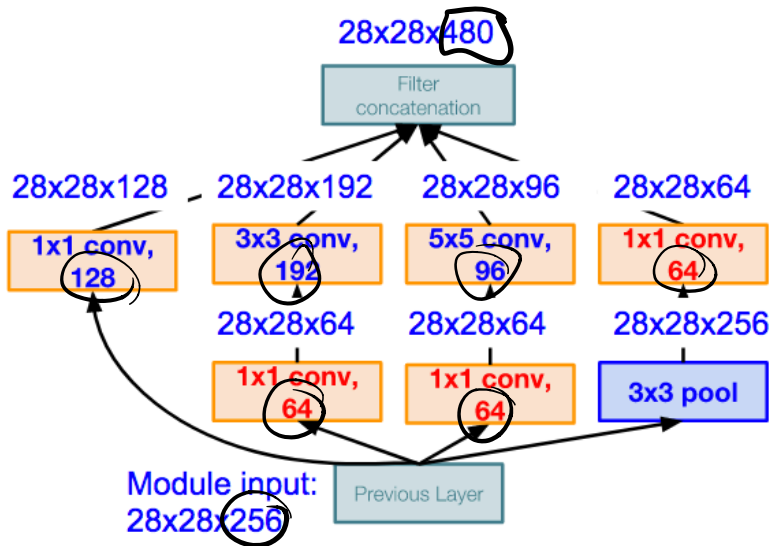
1x1 conv "bottleneck" layers



Inception module with dimension reduction

# Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

## Conv Ops:

- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256

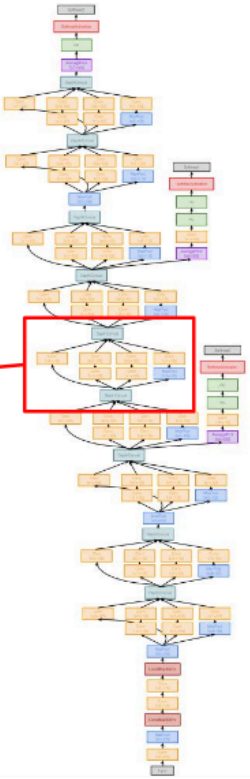
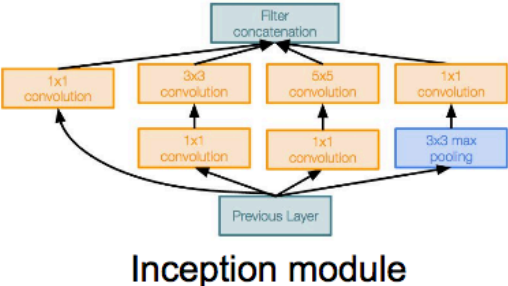
**Total: 358M ops**

Compared to 854M ops for naive version  
Bottleneck can also reduce depth after pooling layer

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules with dimension reduction on top of each other



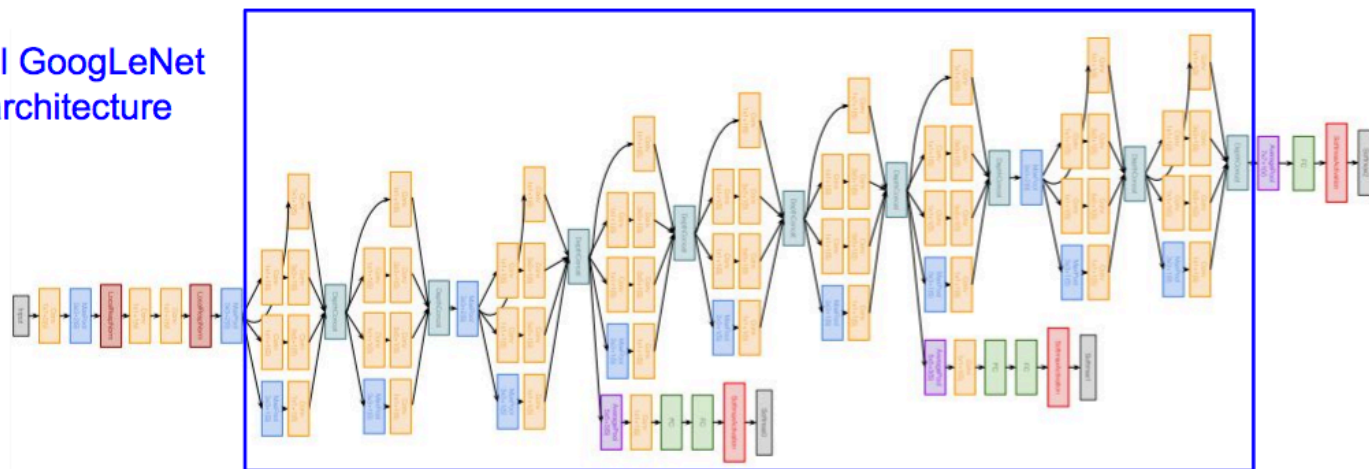




# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture

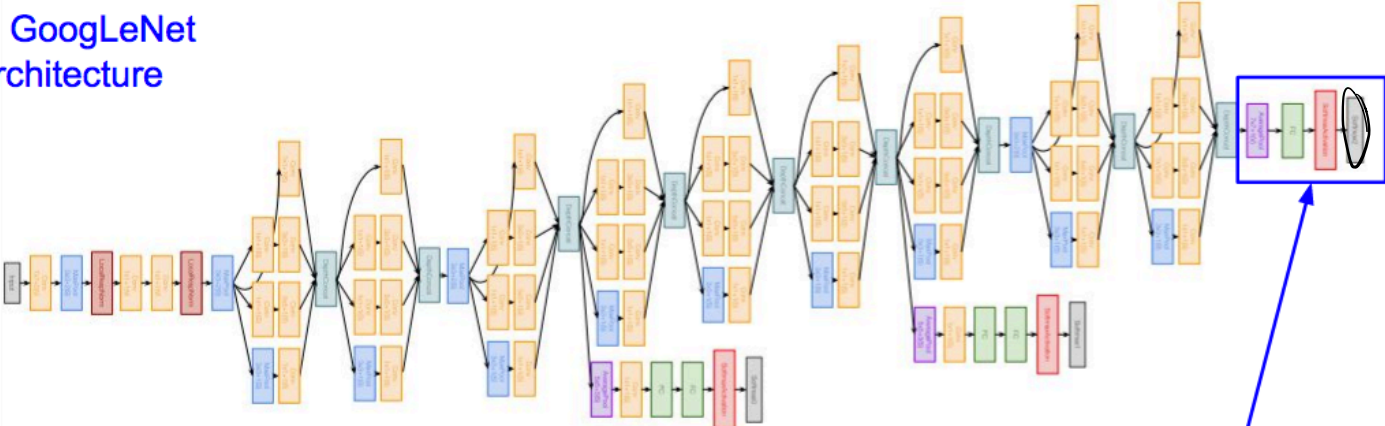


Stacked Inception  
Modules

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture

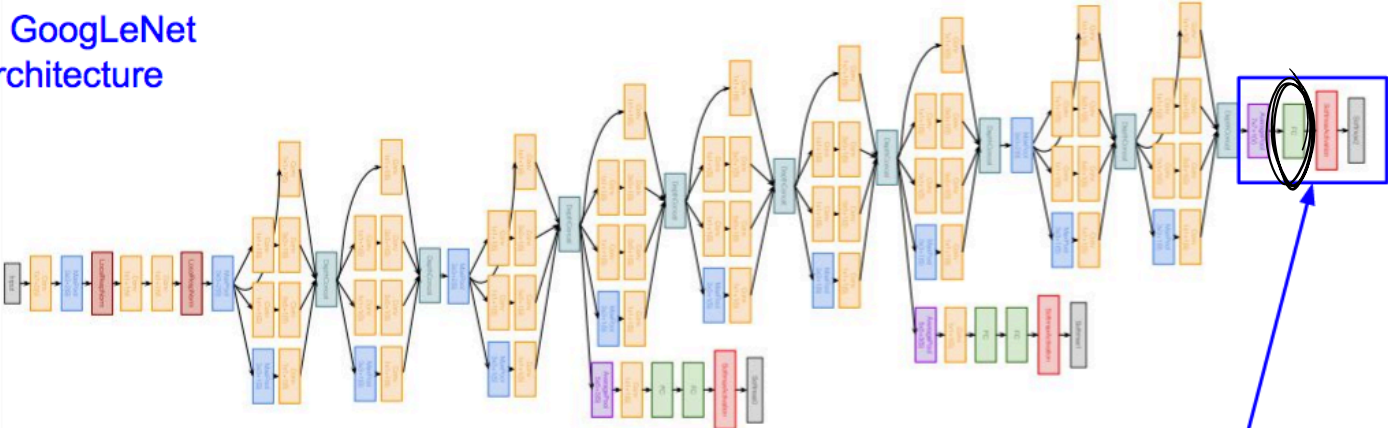


Classifier output

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture

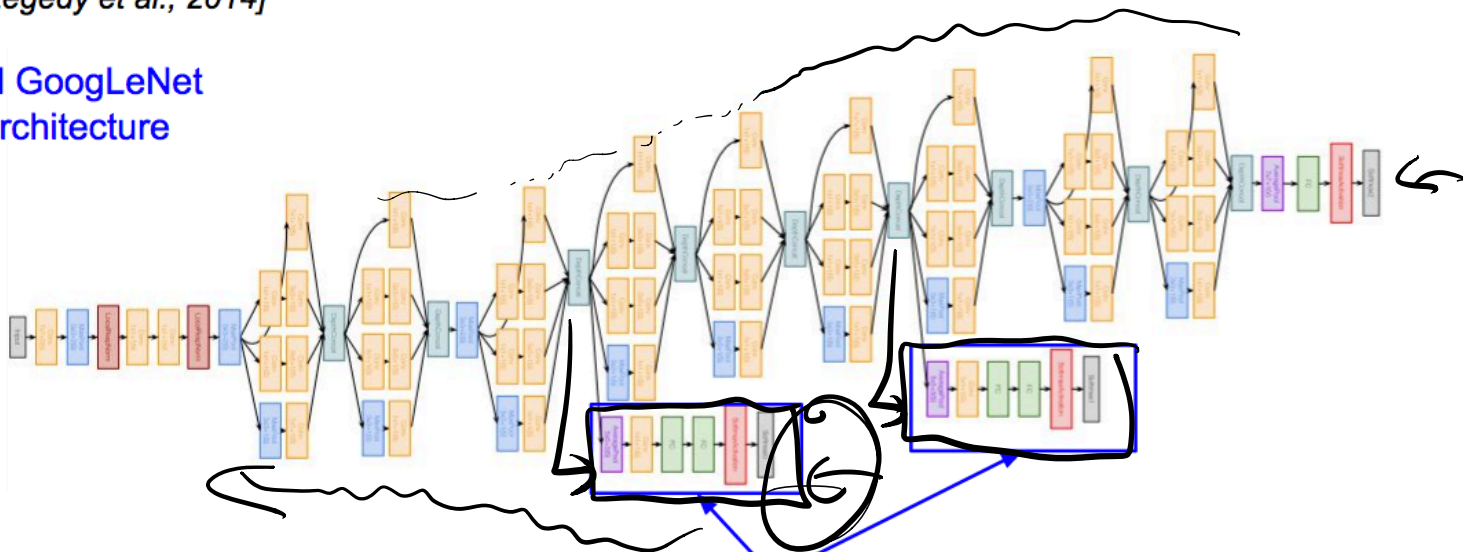


Classifier output  
(removed expensive FC layers!)

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet  
architecture

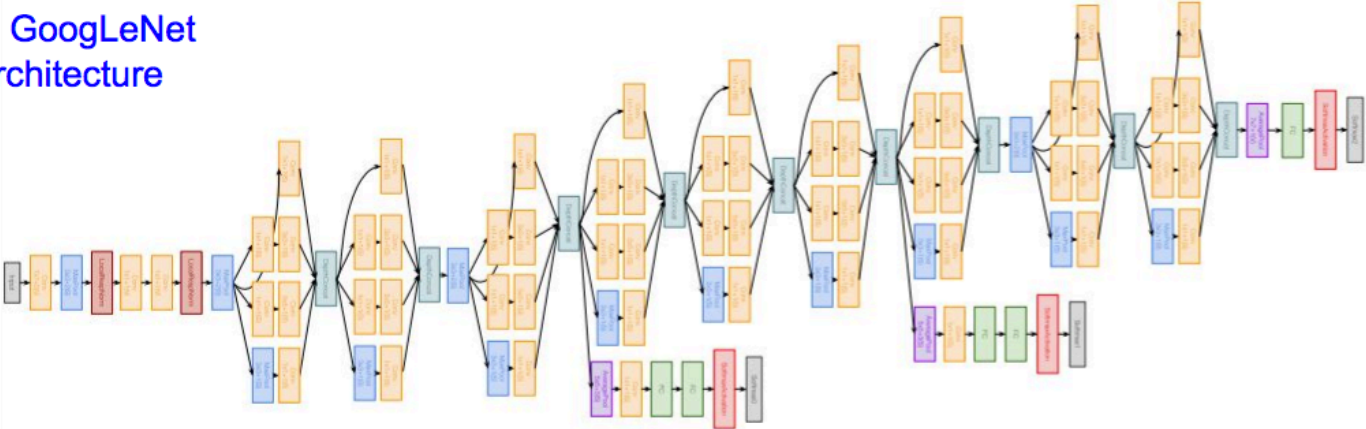


Auxiliary classification outputs to inject additional gradient at lower layers  
(AvgPool-1x1Conv-FC-FC-Softmax)

# Case Study: GoogLeNet

[Szegedy et al., 2014]

## Full GoogLeNet architecture



22 total layers with weights

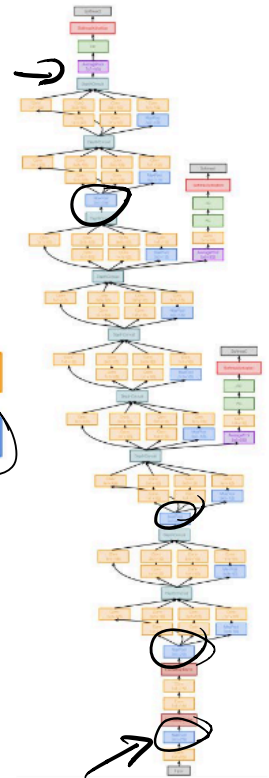
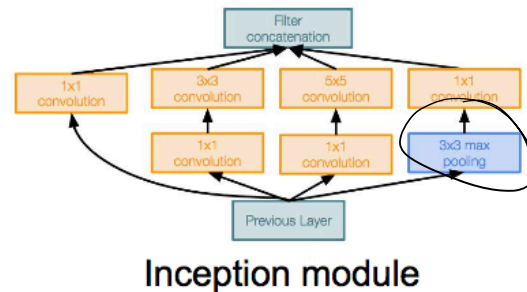
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

# Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



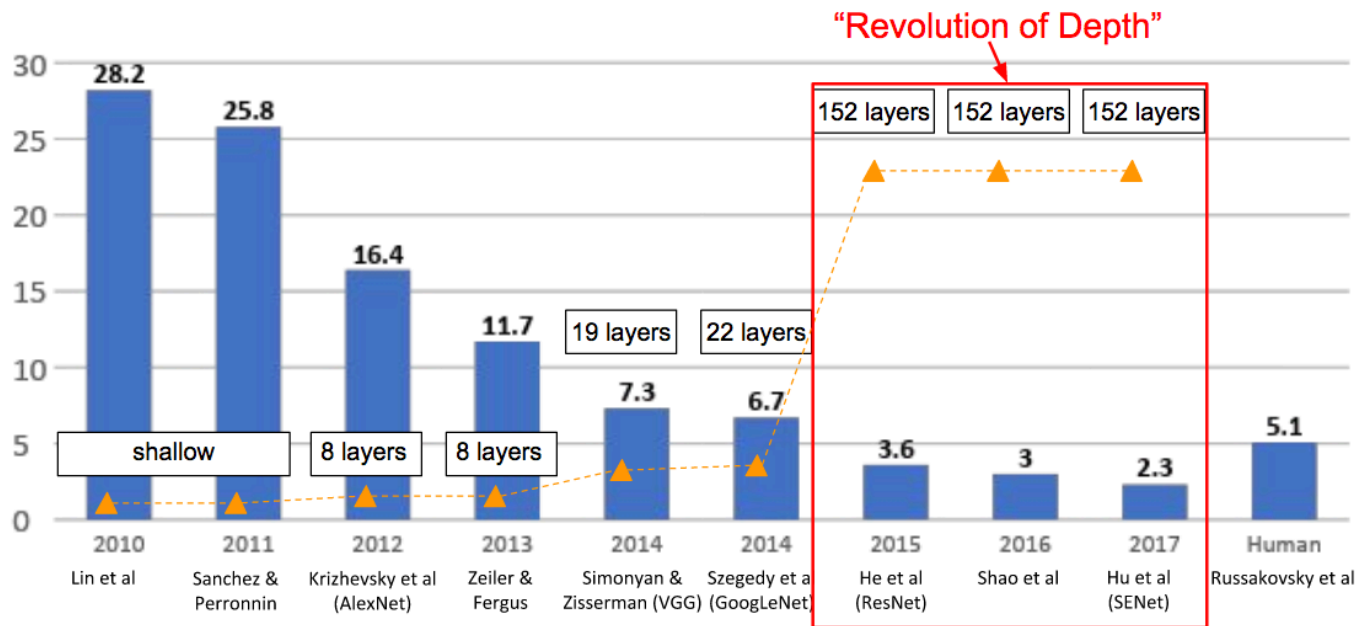


**WE NEED TO GO  
DEEPER**





# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



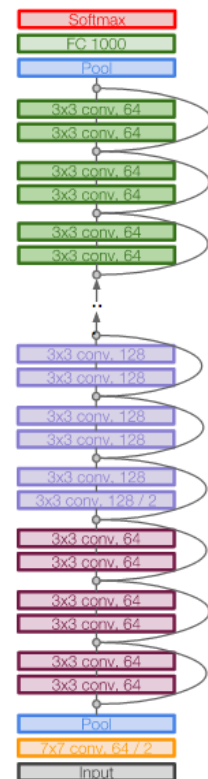
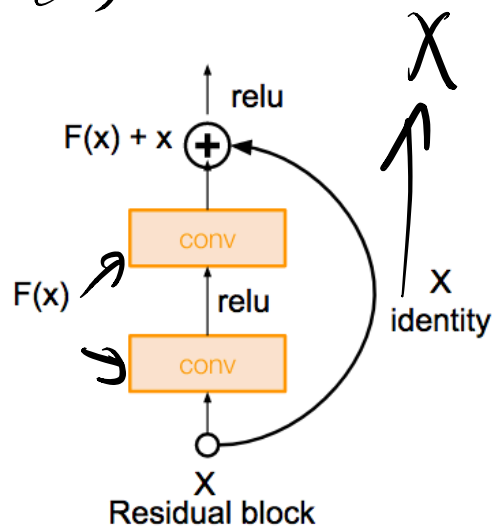
# Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

$$f'(x) - x$$



# Case Study: ResNet

*[He et al., 2015]*

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

# Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



**Q: What's strange about these training and test curves?**

**[Hint: look at the order of the curves]**

56-layer model performs worse on both training and test error-> The deeper model performs worse, but it's not caused by overfitting!

# Case Study: ResNet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

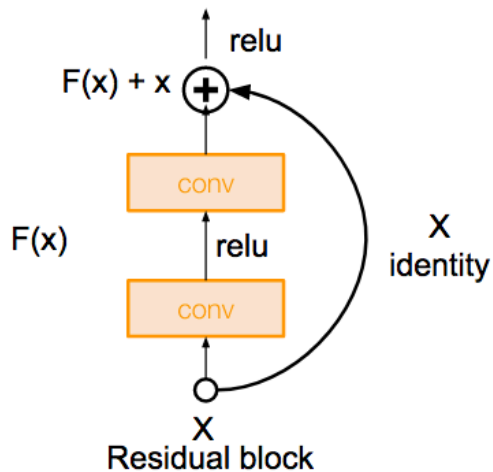
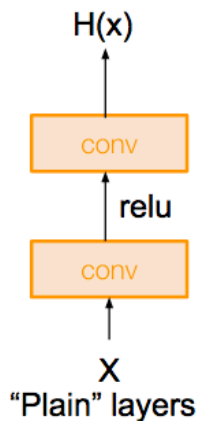
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

# Case Study: ResNet

[He et al., 2015]

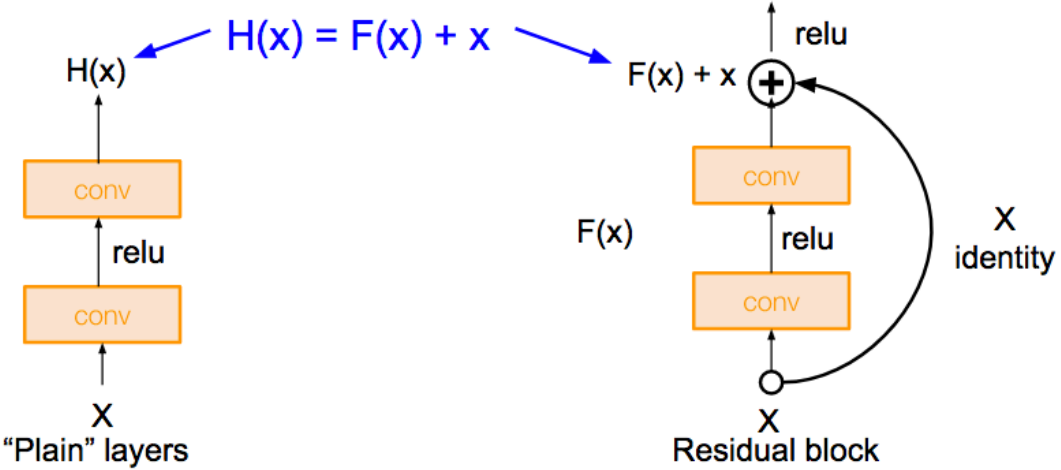
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



# Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual  $F(x) = H(x) - x$  instead of  $H(x)$  directly

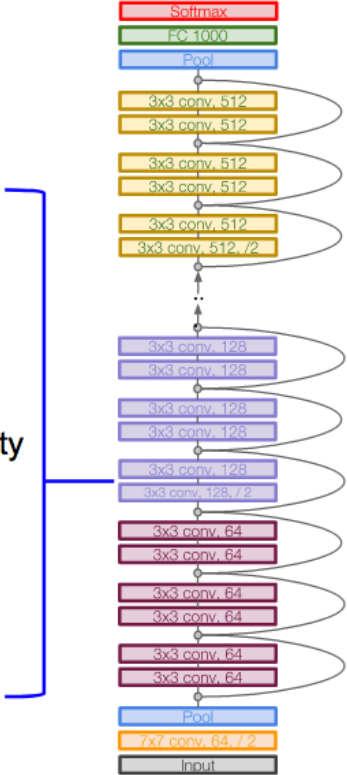
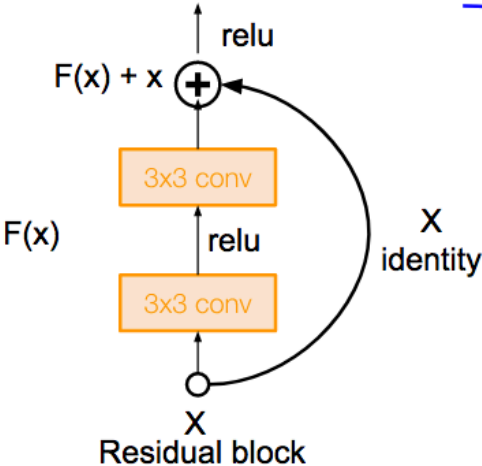


# Case Study: ResNet

[He et al., 2015]

### Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

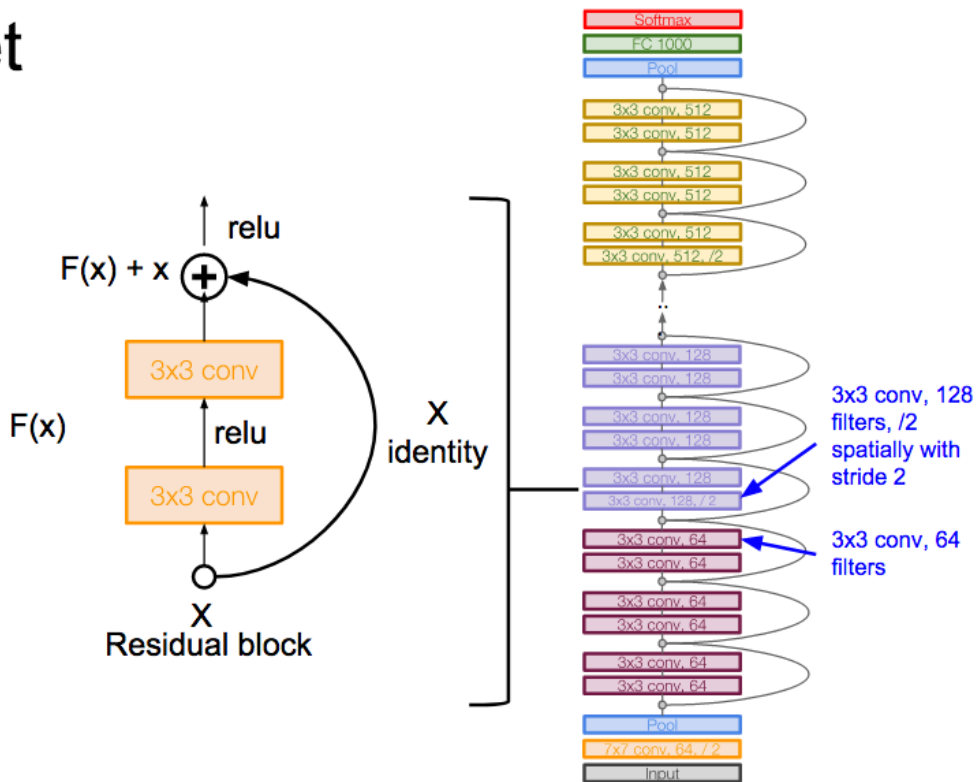


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)

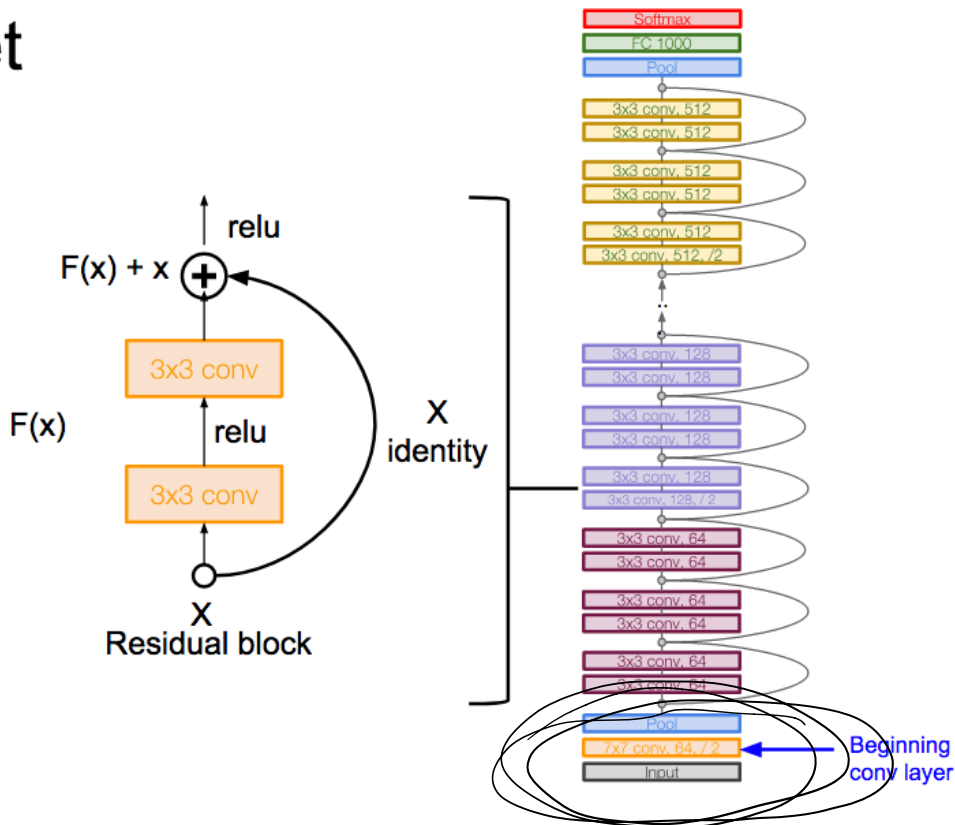


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning

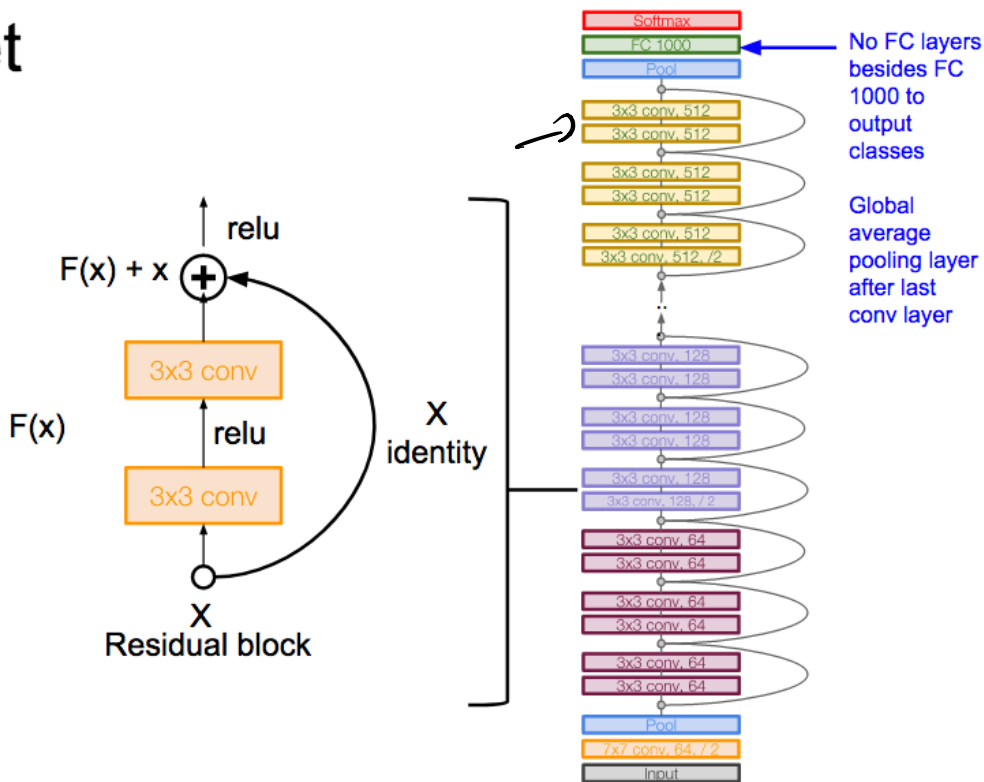


# Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



# Case Study: ResNet

[He et al., 2015]

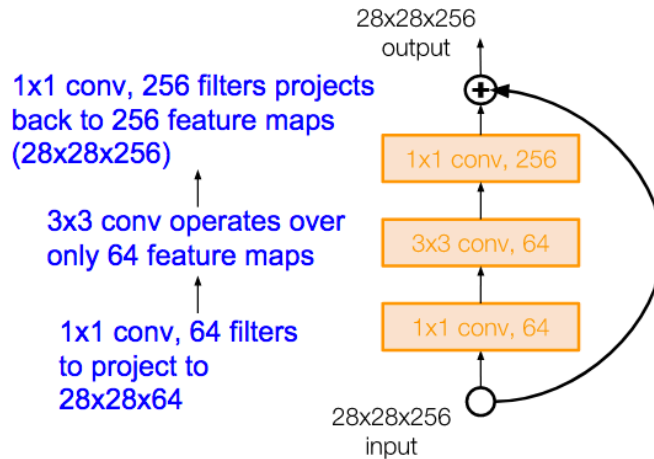
Total depths of 34, 50, 101, or 152 layers for ImageNet



# Case Study: ResNet

[He et al., 2015]

For deeper networks  
(ResNet-50+), use “bottleneck”  
layer to improve efficiency  
(similar to GoogLeNet)



# Case Study: ResNet

[He et al., 2015]

## Experimental Results

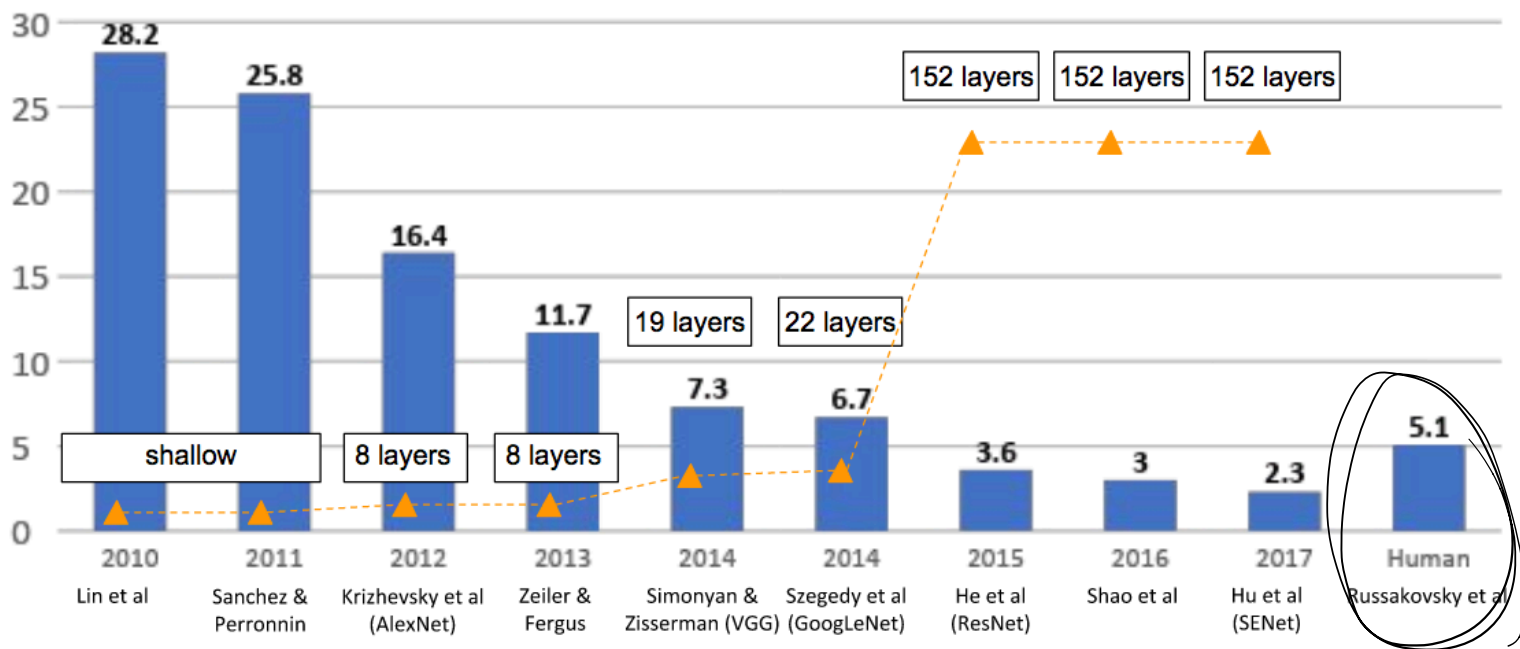
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowering training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

## MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**

- ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

## ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

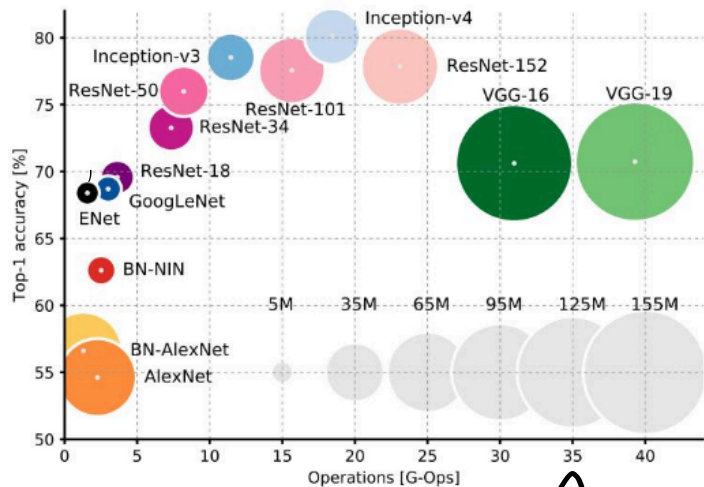
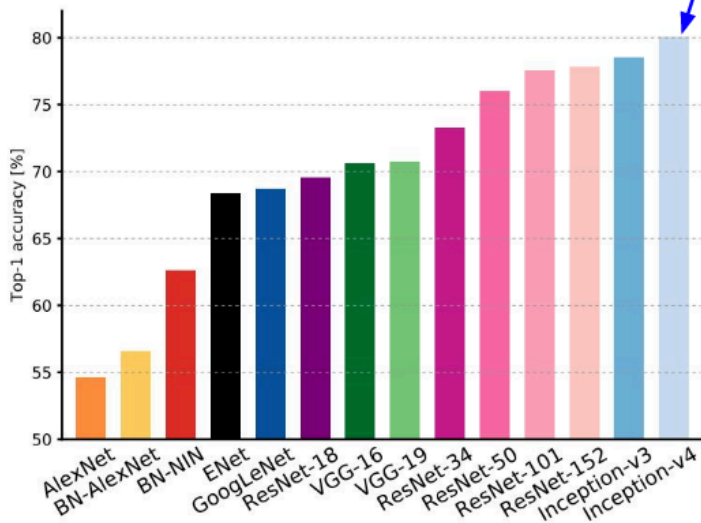






# Comparing complexity...

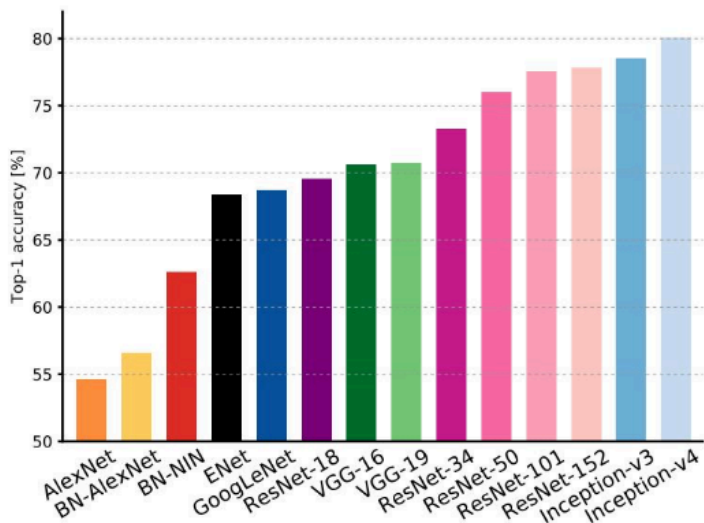
Inception-v4: Resnet + Inception!



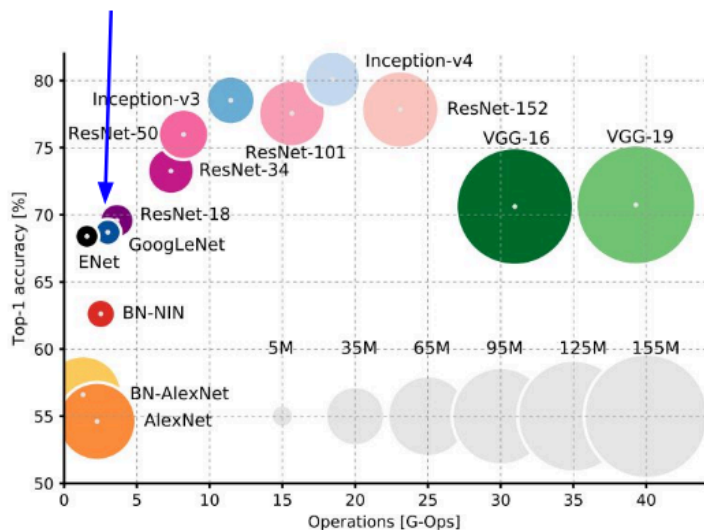
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

## Comparing complexity...



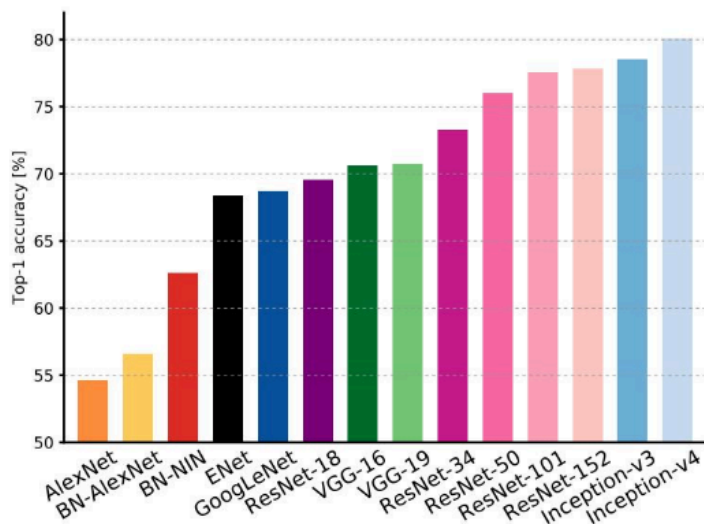
GoogLeNet:  
most efficient



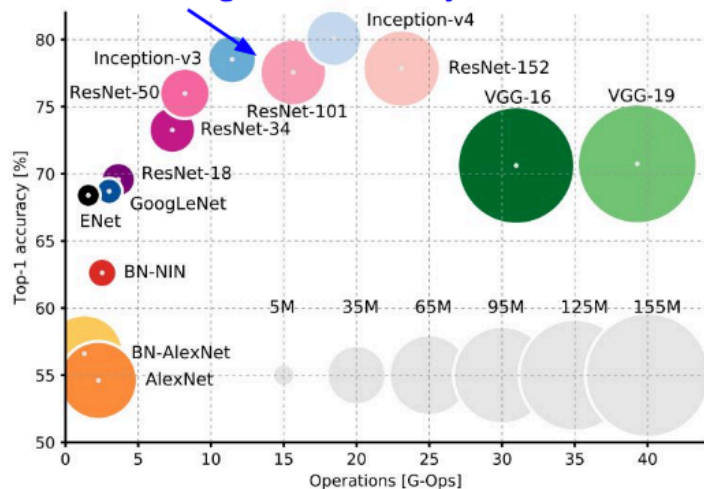
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

## Comparing complexity...



ResNet:  
Moderate efficiency depending on model, highest accuracy



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

# And so on and so forth...

- Dizzying number of papers since then proposing more architecture tricks and hacks.
- A couple notable examples:
  - FractalNet
  - DenseNet

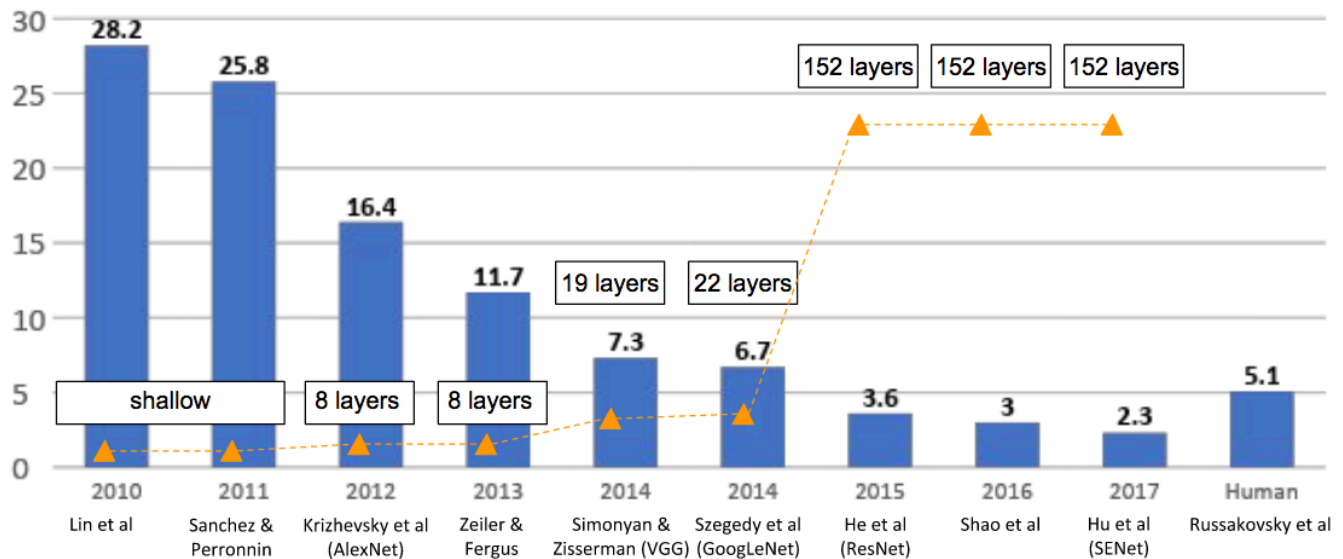


Do we though?  
(open question)

# And so on and so forth...

- So we've beat the crap out of ImageNet... what now?

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# And so on and so forth...

- So we've beat the crap out of ImageNet... what now?
  - Can we do image classification on other datasets?
  - Can we do things other than image classification?



# And so on and so forth...

- So we've beat the crap out of ImageNet... what now?
  - **Can we do image classification on other datasets?**
  - Can we do things other than image classification?

# Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

# Transfer Learning

“You need a lot of data if you want to train/use CNNs”

well... sort of

**BUSTED**

# Transfer Learning with CNNs

## 1. Train on Imagenet



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

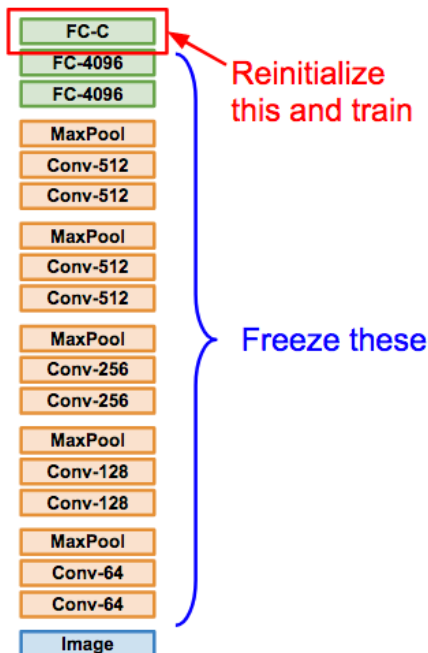
# Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

## 1. Train on Imagenet



## 2. Small Dataset (C classes)



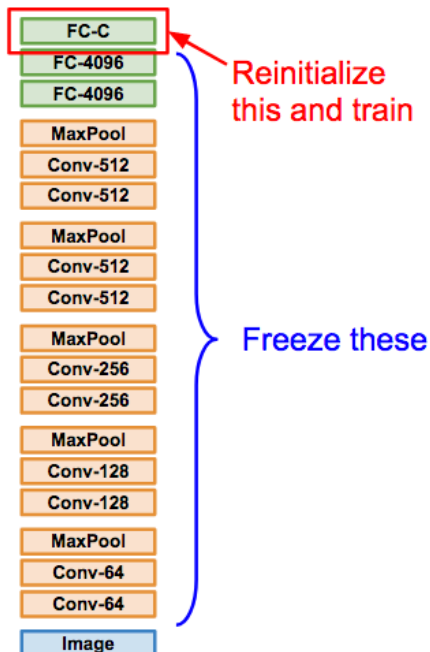
# Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

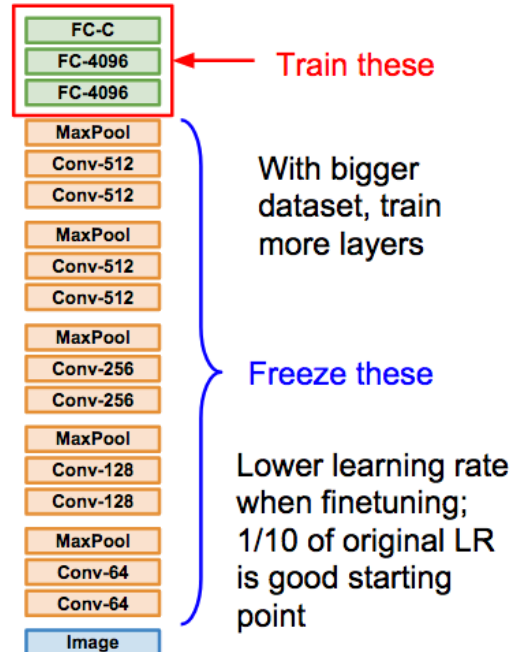
## 1. Train on Imagenet



## 2. Small Dataset (C classes)



## 3. Bigger dataset



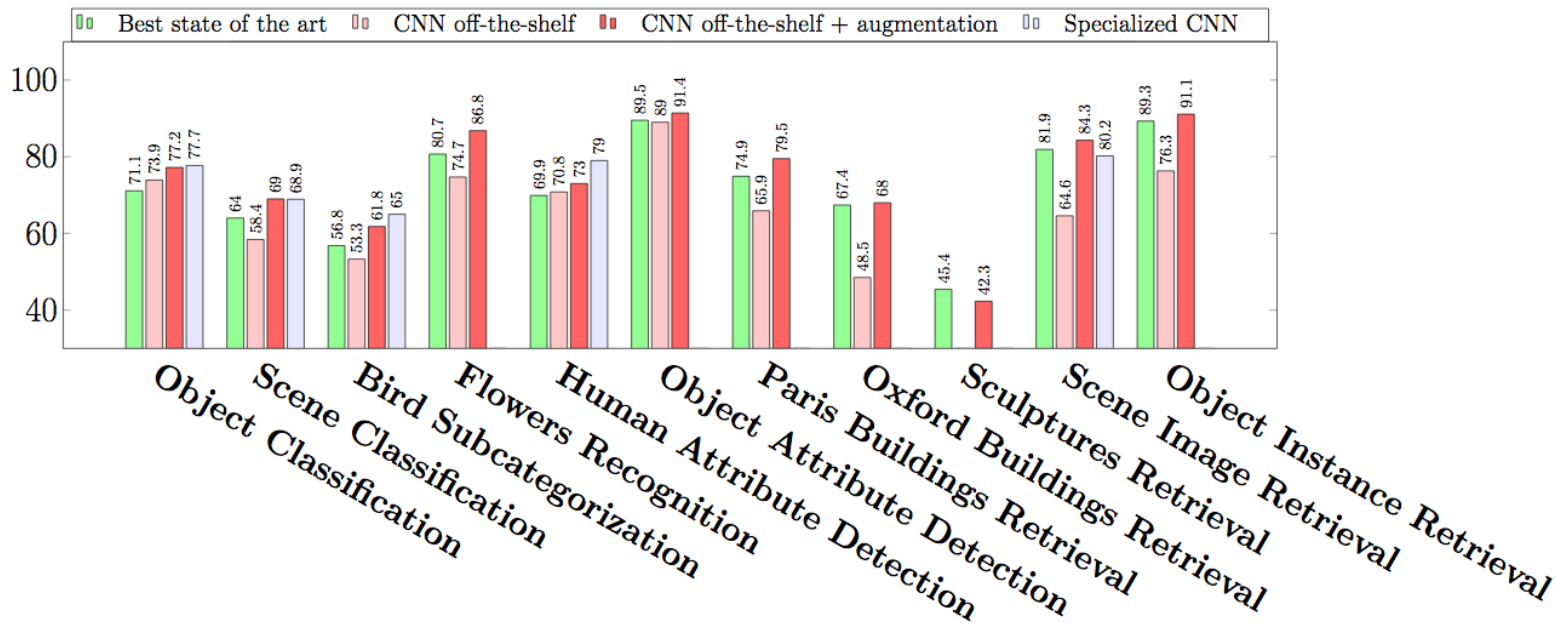
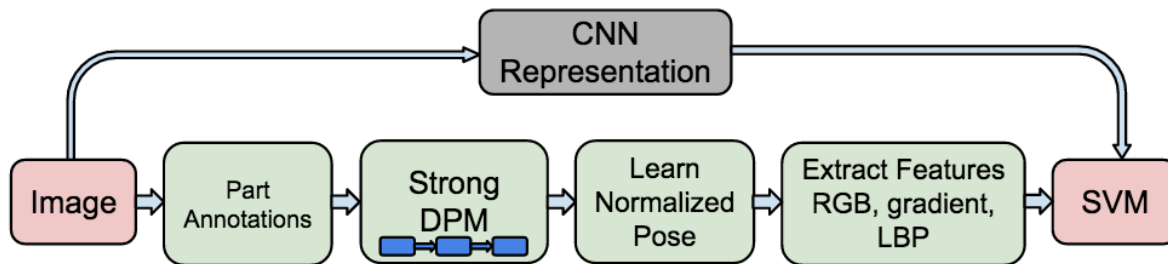
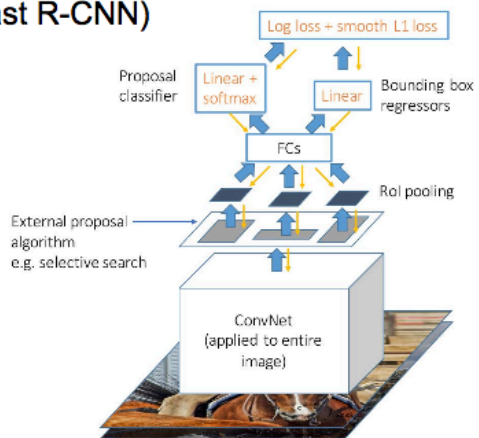


Figure: Razavian et al.: CNN Features off-the-shelf: an Astounding Baseline for Recognition

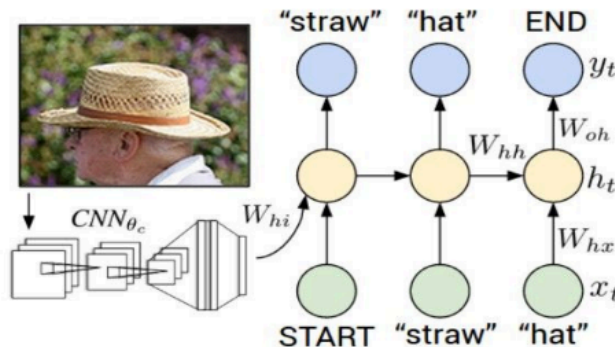
<https://arxiv.org/pdf/1403.6382.pdf>

# Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

## Object Detection (Fast R-CNN)



## Image Captioning: CNN + RNN



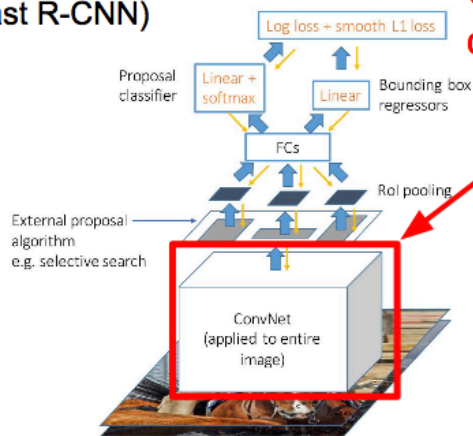
Girshick, "Fast R-CNN", ICCV 2015  
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for  
Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.



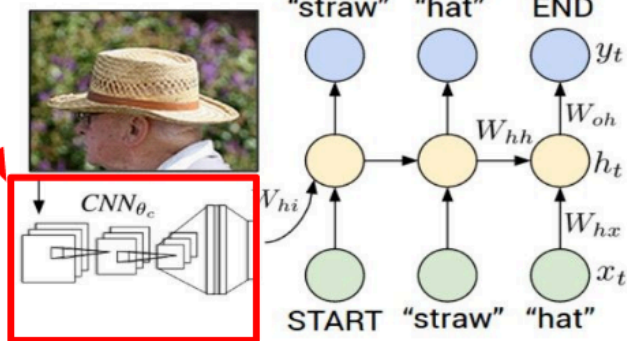
# Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

## Object Detection (Fast R-CNN)



**CNN pretrained  
on ImageNet**

## Image Captioning: CNN + RNN





# And so on and so forth...

- So we've beat the crap out of ImageNet... what now?
  - Can we do image classification on other datasets?
  - **Can we do things other than image classification?**

# Other Computer Vision Tasks

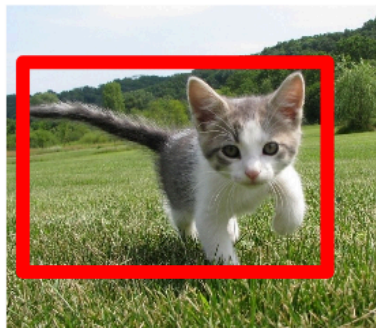
## Semantic Segmentation



GRASS, CAT,  
TREE, SKY

No objects, just pixels

## Classification + Localization



CAT

Single Object

## Object Detection



DOG, DOG, CAT

Multiple Object

## Instance Segmentation



DOG, DOG, CAT

This image is CC0 public domain

# Other Computer Vision Tasks

## Semantic Segmentation



GRASS, CAT,  
TREE, SKY

No objects, just pixels

## Classification + Localization



CAT

Single Object

## Object Detection



DOG, DOG, CAT

Multiple Object

## Instance Segmentation



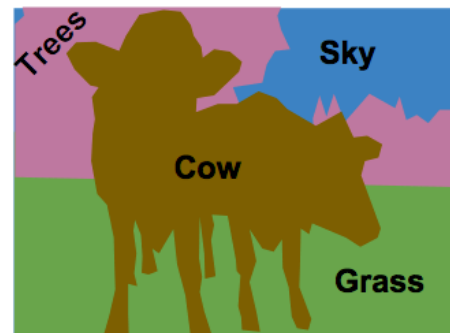
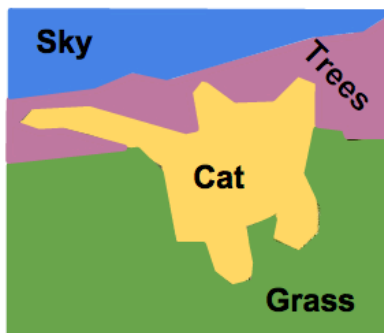
DOG, DOG, CAT

This image is CC0 public domain

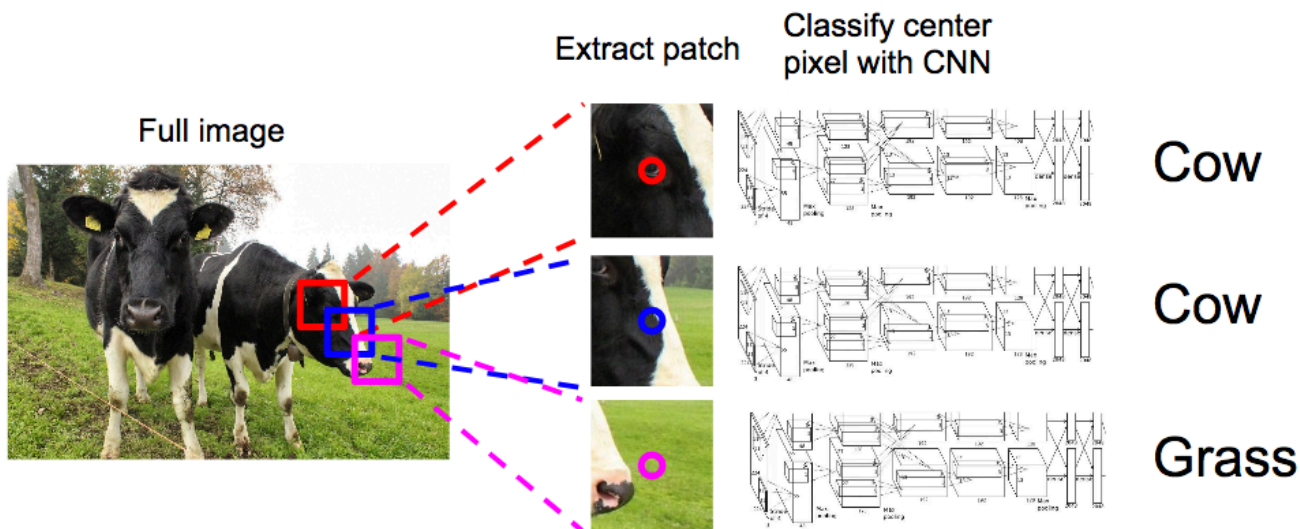
# Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels



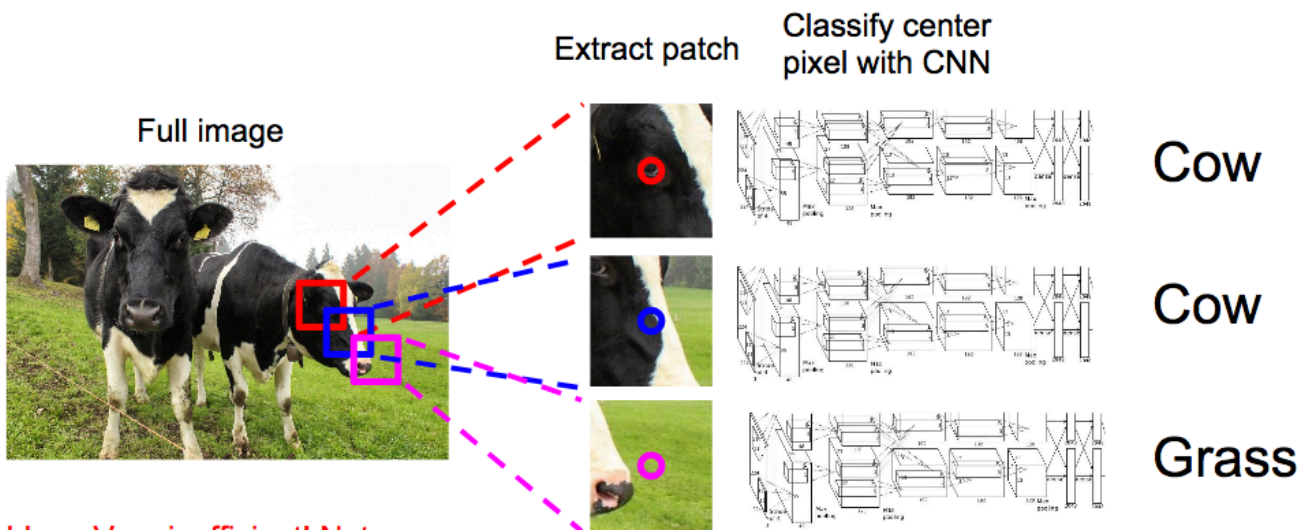
# Semantic Segmentation Idea: Sliding Window



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



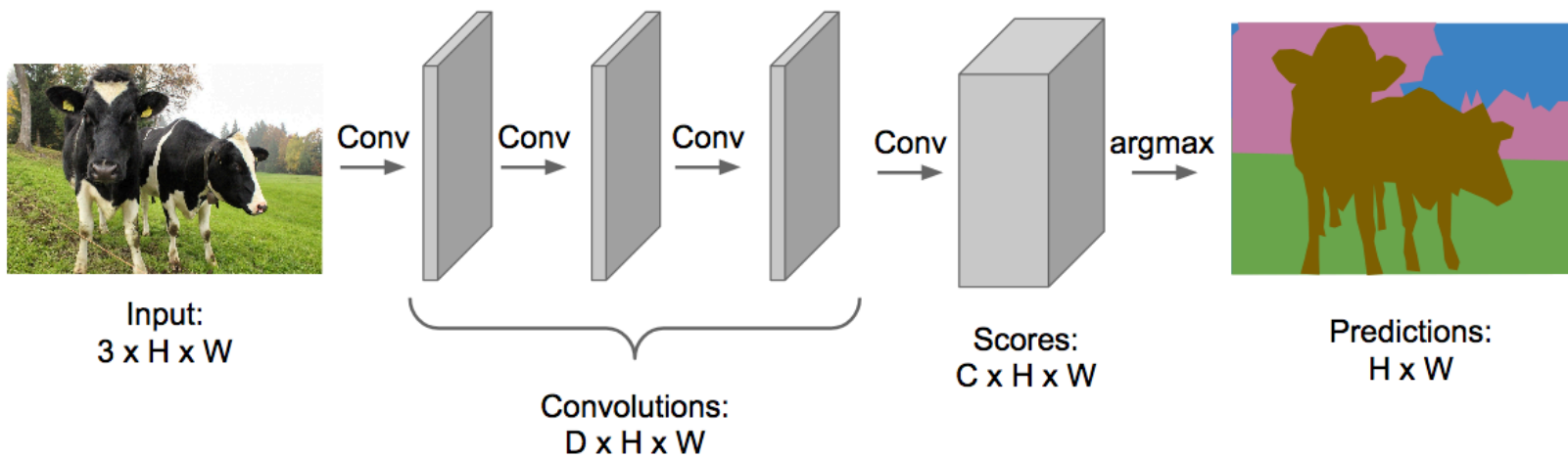
**Problem: Very inefficient! Not reusing shared features between overlapping patches**

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



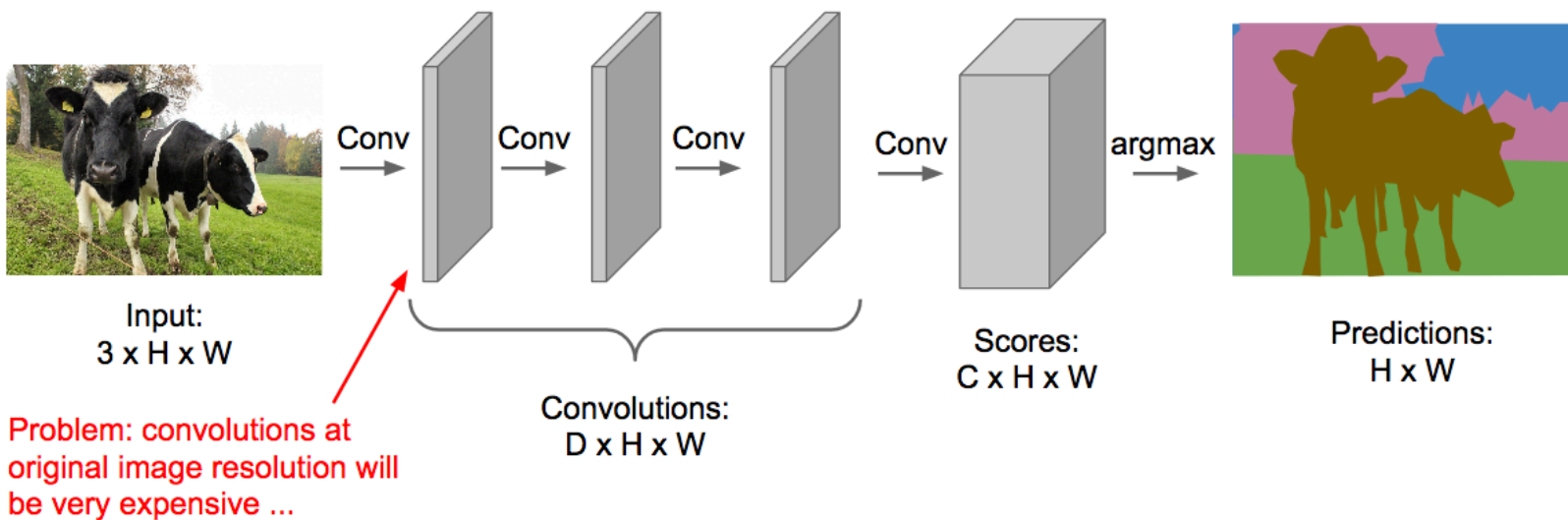
# Semantic Segmentation Idea: Fully Convolutional

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



# Semantic Segmentation Idea: Fully Convolutional

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

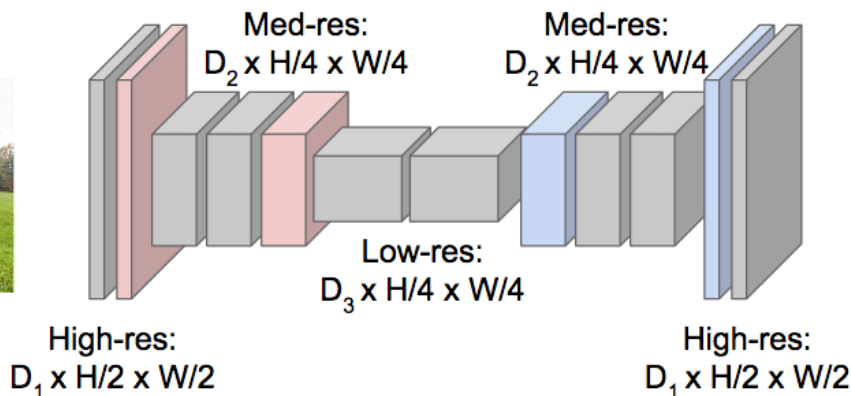


# Semantic Segmentation Idea: Fully Convolutional

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



Input:  
 $3 \times H \times W$



Predictions:  
 $H \times W$

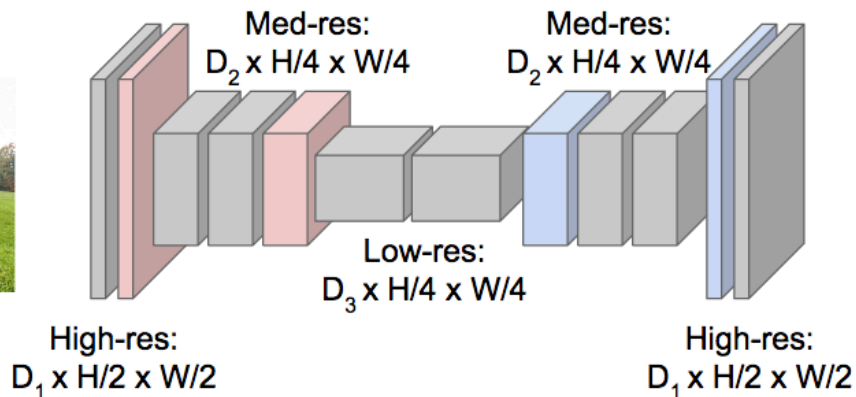
# Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**  
Pooling, strided  
convolution



Input:  
 $3 \times H \times W$

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



**Upsampling:**  
???



Predictions:  
 $H \times W$

# In-Network upsampling: “Unpooling”

**Nearest Neighbor**

|   |   |
|---|---|
| 1 | 2 |
| 3 | 4 |



|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

Input: 2 x 2

Output: 4 x 4

**“Bed of Nails”**

|   |   |
|---|---|
| 1 | 2 |
| 3 | 4 |



|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 |

Input: 2 x 2

Output: 4 x 4

# In-Network upsampling: “Max Unpooling”

## 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



Rest of the network

## Max Unpooling

Use positions from pooling layer

|   |   |
|---|---|
| 1 | 2 |
| 3 | 4 |

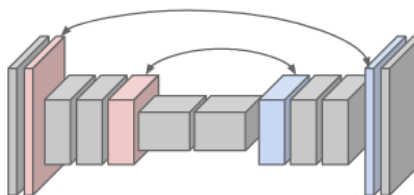
Input: 2 x 2



|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 2 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 4 |

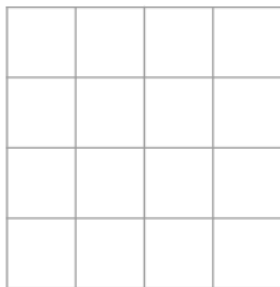
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

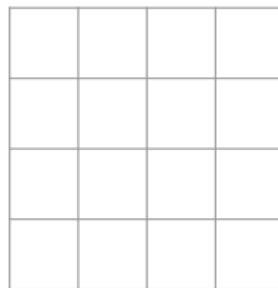


# Learnable Upsampling: Transpose Convolution

**Recall:** Typical 3 x 3 convolution, stride 1 pad 1



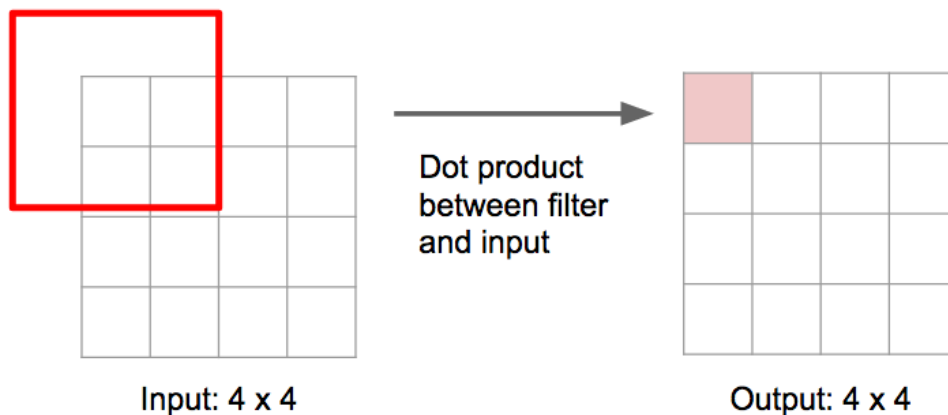
Input: 4 x 4



Output: 4 x 4

# Learnable Upsampling: Transpose Convolution

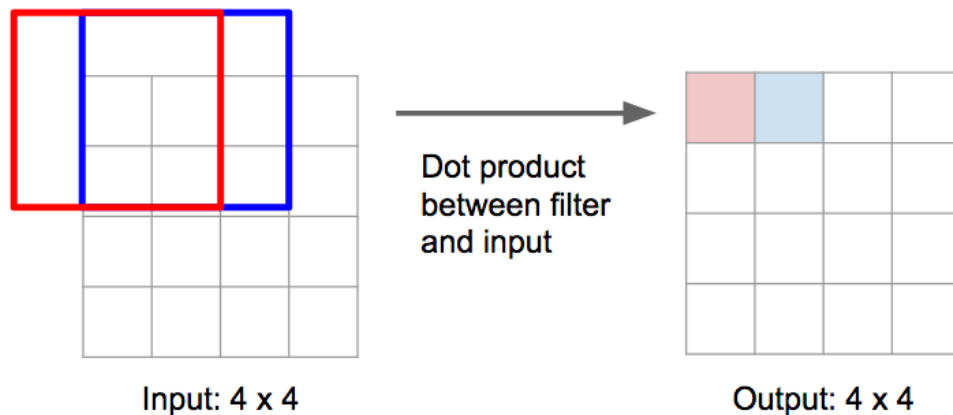
**Recall:** Normal 3 x 3 convolution, stride 1 pad 1





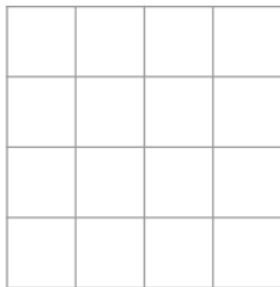
# Learnable Upsampling: Transpose Convolution

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



# Learnable Upsampling: Transpose Convolution

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



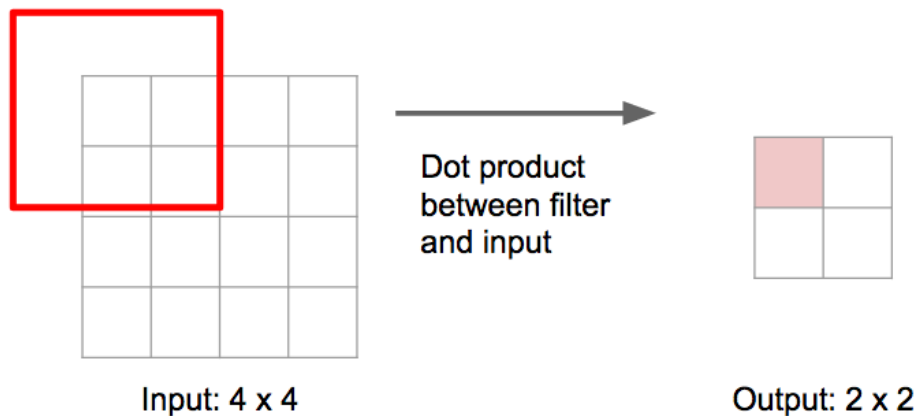
Input: 4 x 4



Output: 2 x 2

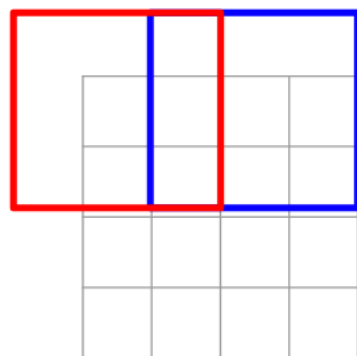
# Learnable Upsampling: Transpose Convolution

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



# Learnable Upsampling: Transpose Convolution

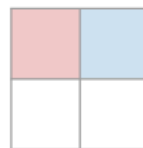
**Recall:** Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4



Dot product  
between filter  
and input



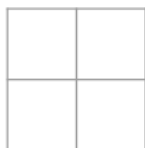
Output: 2 x 2

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

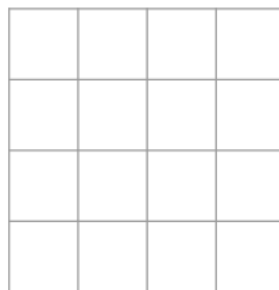
Stride gives ratio between  
movement in input and  
output

# Learnable Upsampling: Transpose Convolution

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



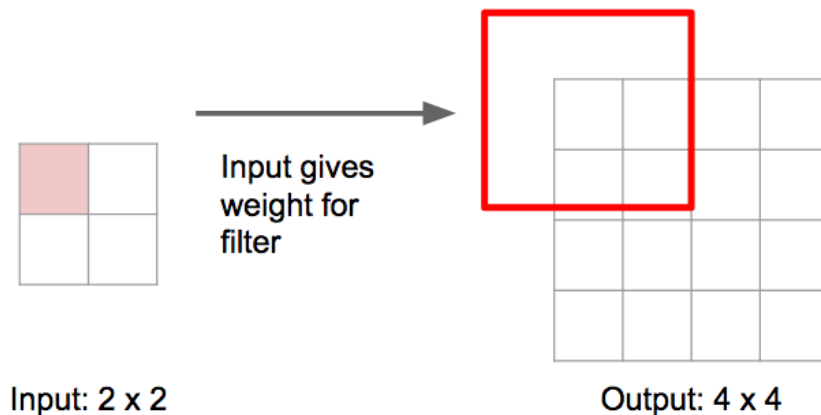
Input: 2 x 2



Output: 4 x 4

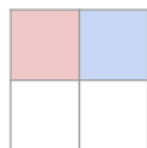
# Learnable Upsampling: Transpose Convolution

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



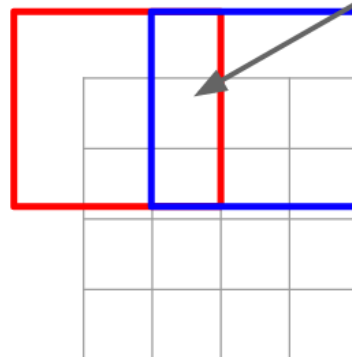
# Learnable Upsampling: Transpose Convolution

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



Input: 2 x 2

Input gives weight for filter



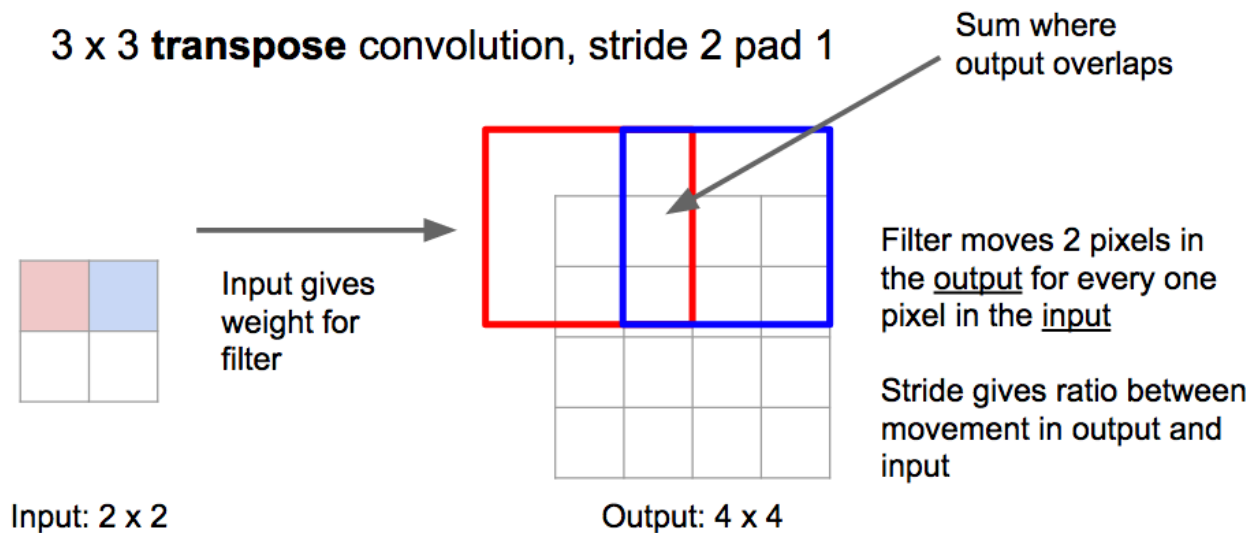
Output: 4 x 4

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



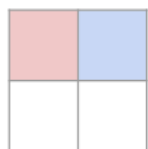


# Learnable Upsampling: Transpose Convolution

## Other names:

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

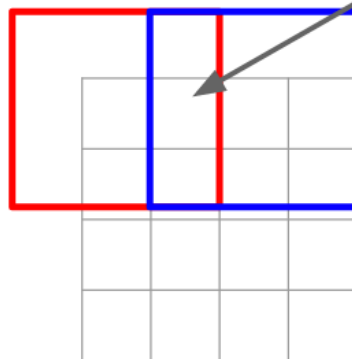
3 x 3 **transpose** convolution, stride 2 pad 1



Input: 2 x 2



Input gives weight for filter



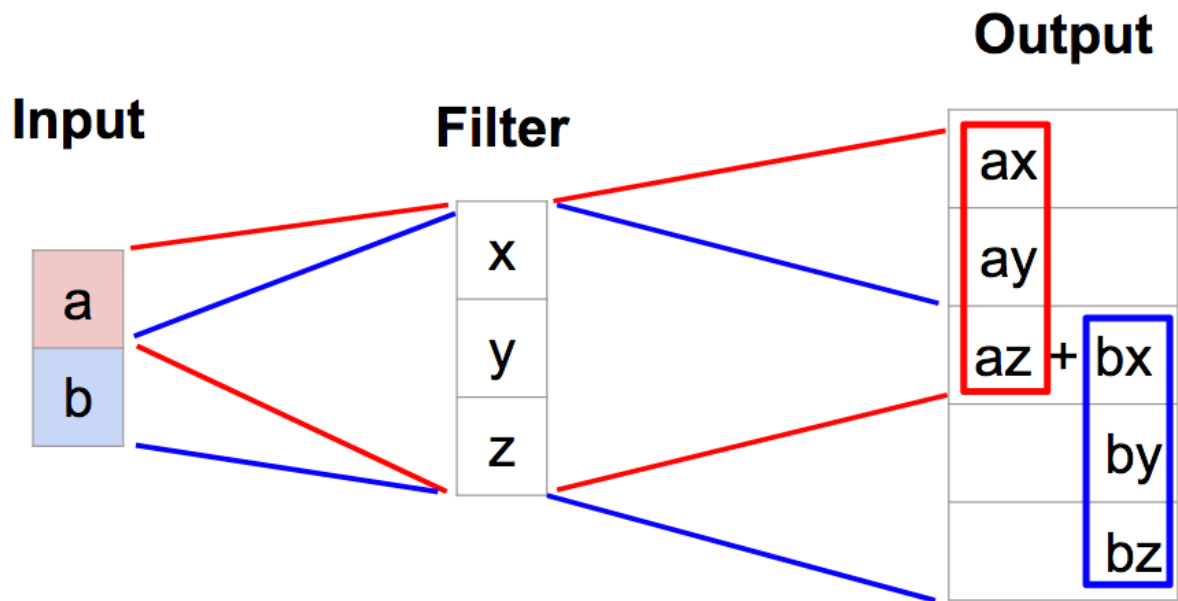
Output: 4 x 4

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: 1D Example



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

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



# 2D Object Detection

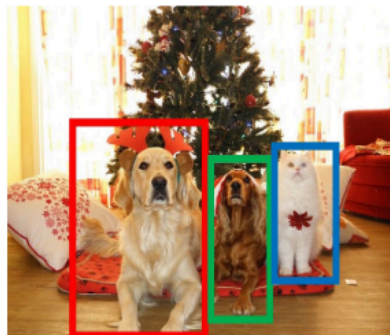
## Semantic Segmentation



GRASS, CAT,  
TREE, SKY

No objects, just pixels

## 2D Object Detection



DOG, DOG, CAT

Object categories +  
2D bounding boxes

## 3D Object Detection



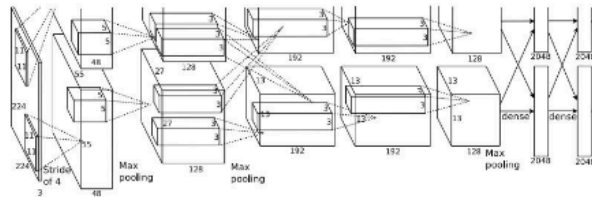
Car

Object categories +  
3D bounding boxes

This image is CC0 public domain

# Object Detection as Classification: Sliding Window

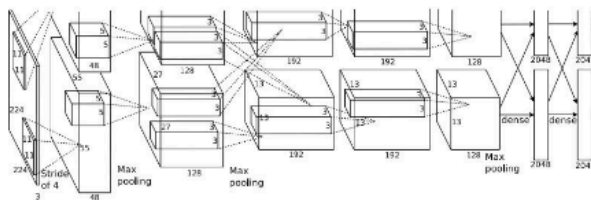
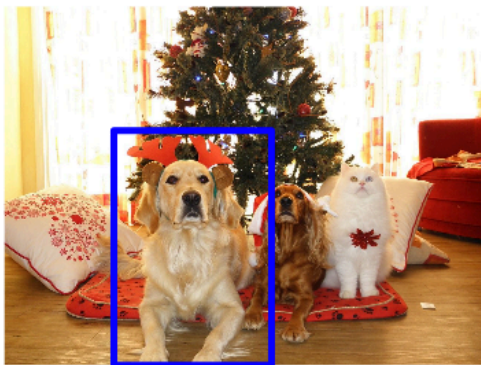
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO  
Cat? NO  
Background? YES

# Object Detection as Classification: Sliding Window

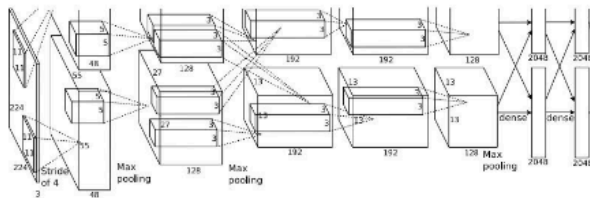
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES  
Cat? NO  
Background? NO

# Object Detection as Classification: Sliding Window

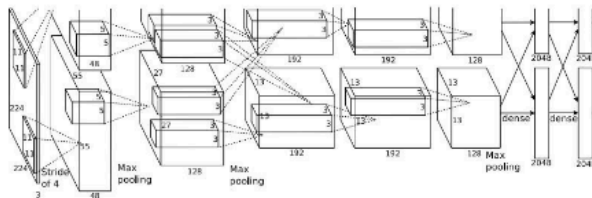
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES  
Cat? NO  
Background? NO

# Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

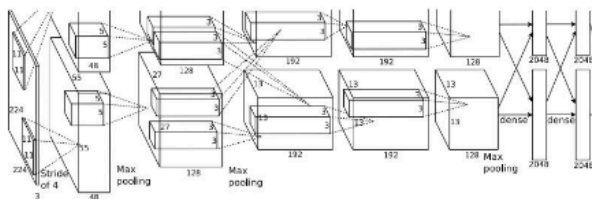
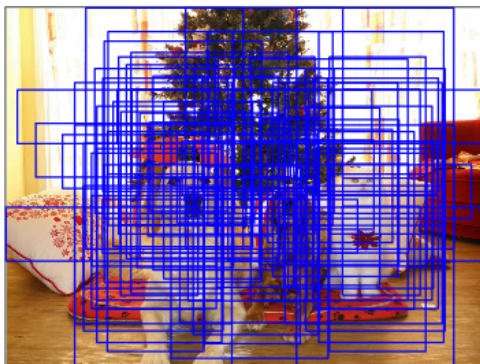


Dog? NO  
Cat? YES  
Background? NO



# Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

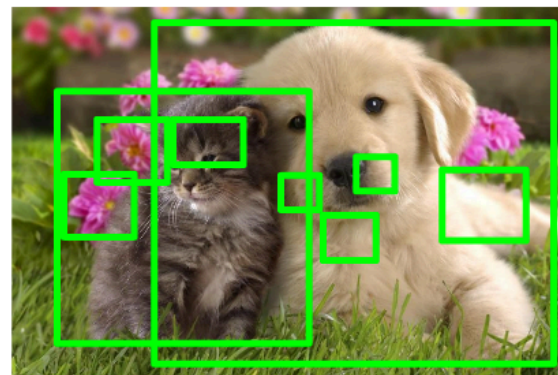


Dog? NO  
Cat? YES  
Background? NO

**Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!**

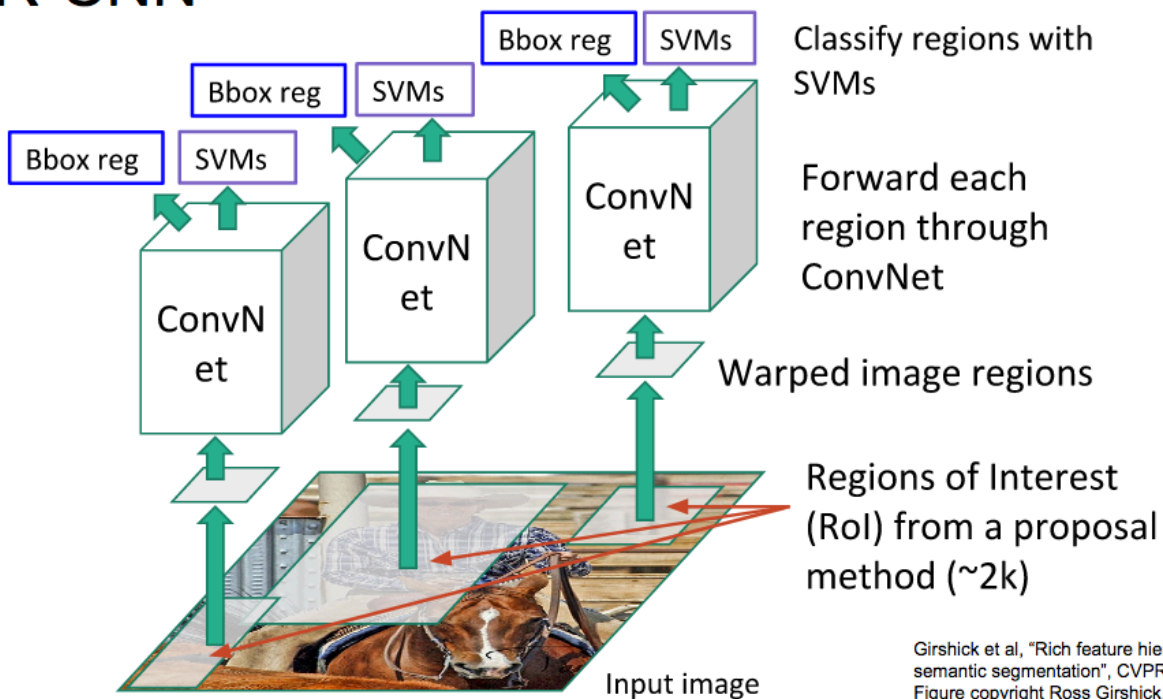
# Region Proposals / Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



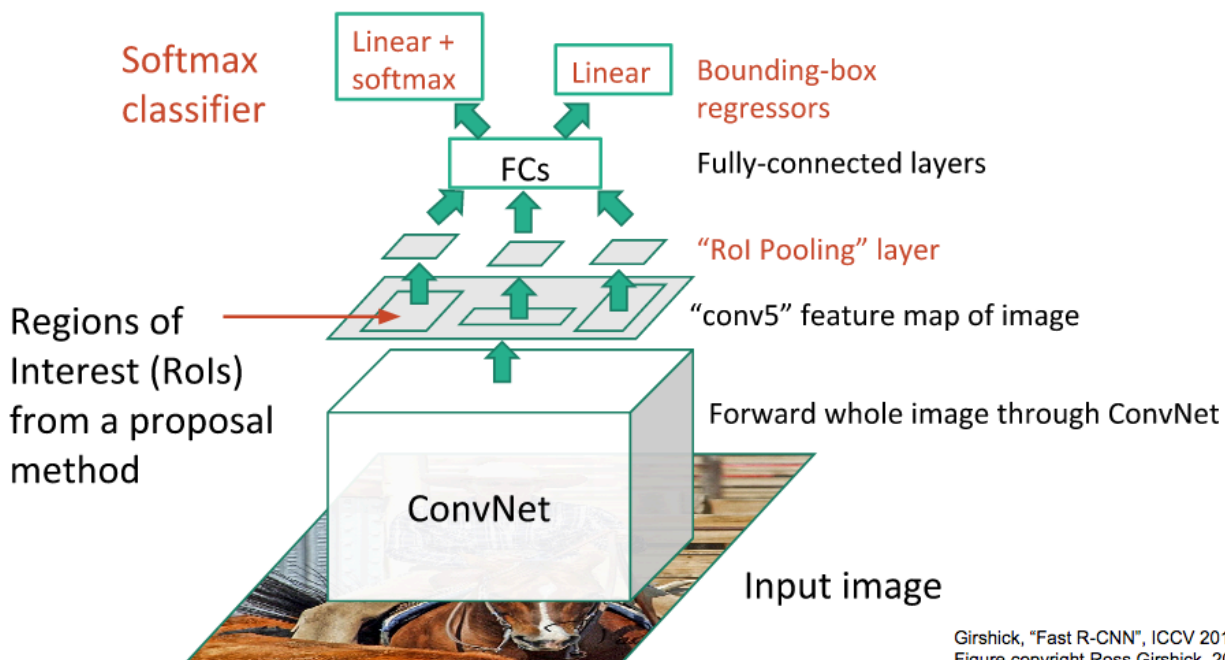
Alexe et al, "Measuring the objectness of image windows", TPAMI 2012  
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013  
Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014  
Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

# R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

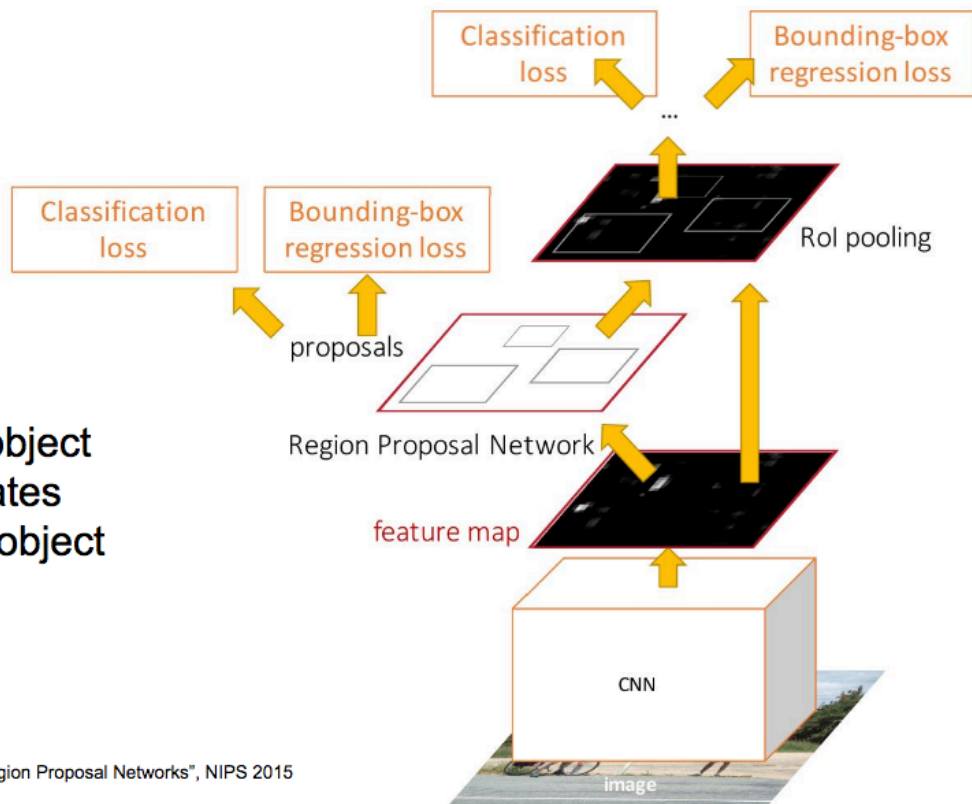
# Faster R-CNN:

Make CNN do proposals!

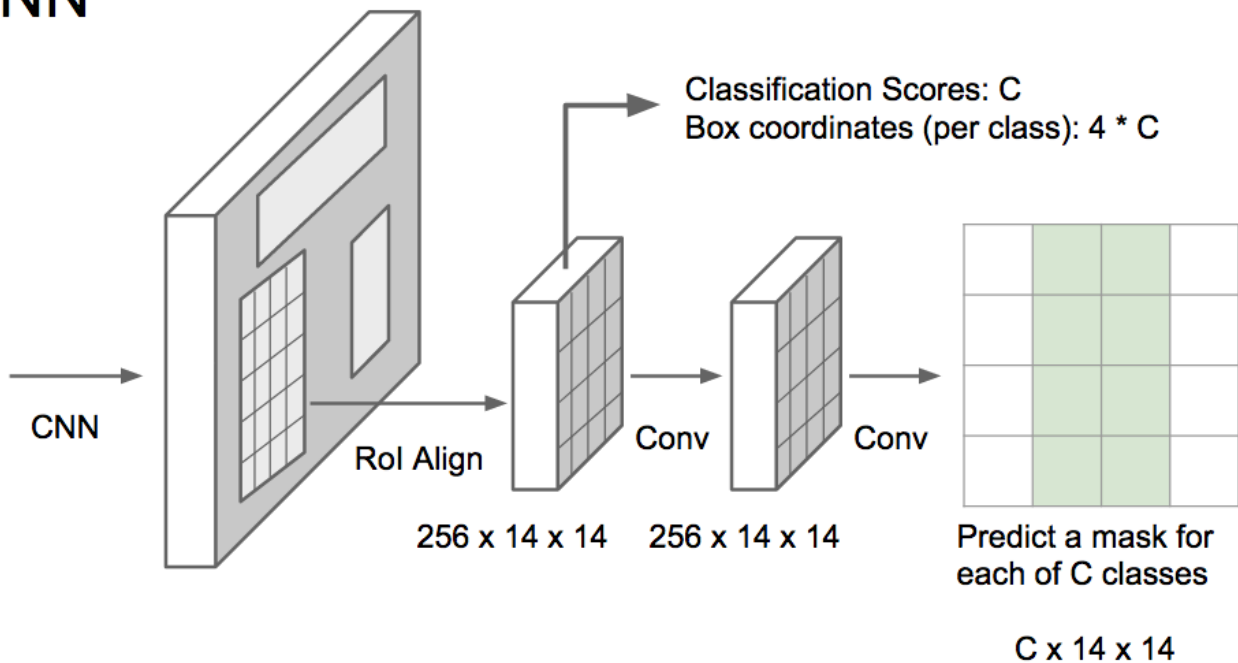
Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

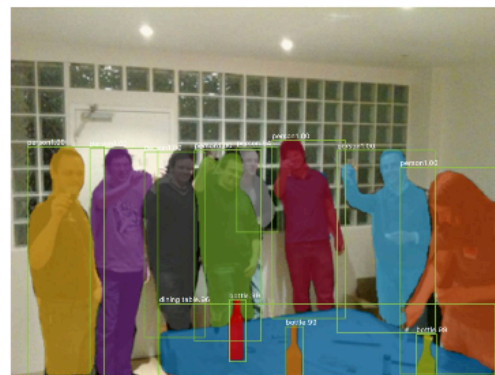
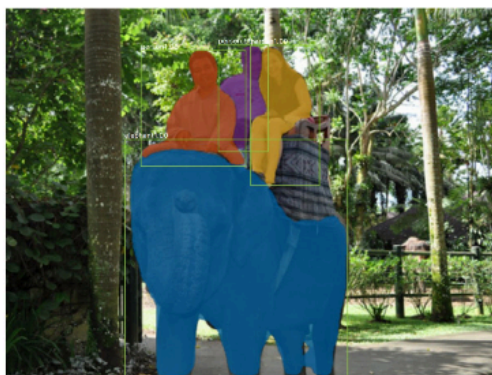
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



# Mask R-CNN



# Mask R-CNN: Very Good Results!



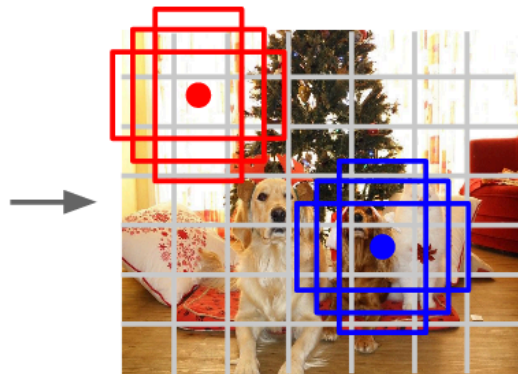
He et al, "Mask R-CNN", arXiv 2017  
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.  
Reproduced with permission.

# Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network! →



Input image  
 $3 \times H \times W$



Divide image into grid  
 $7 \times 7$

Image a set of **base boxes**  
centered at each grid cell  
Here  $B = 3$

Within each grid cell:

- Regress from each of the  $B$  base boxes to a final box with 5 numbers:  $(dx, dy, dh, dw, \text{confidence})$
- Predict scores for each of  $C$  classes (including background as a class)

Output:  
 $7 \times 7 \times (5 * B + C)$



# Object Detection: Impact of Deep Learning

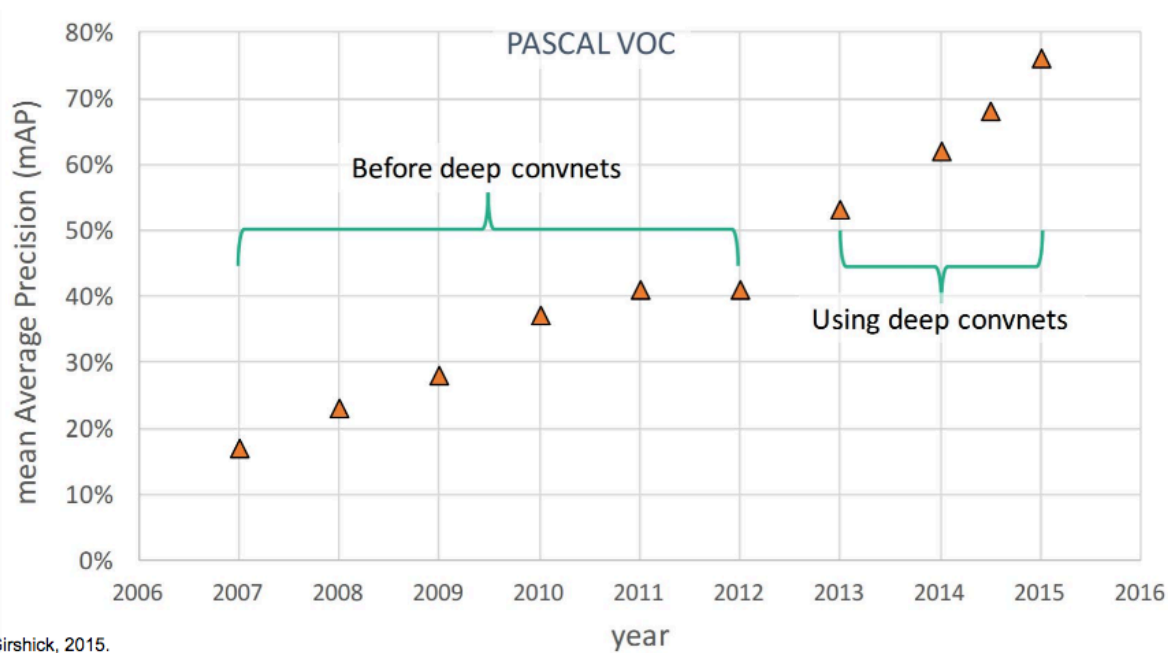


Figure copyright Ross Girshick, 2015.  
Reproduced with permission.

# Other Problems

- Fine-grained recognition (e.g., dog/bird species)
- Instance segmentation
- Face detection and recognition
- Motion estimation
- Feature detection and description
- Depth estimation
- Novel view synthesis
- ...and many others