
Computer Vision with Convolutional Neural Networks

Convolution, LeNet, AlexNet, VGGNet, GoogleNet, Resnet, DenseNet, R-CNN, Fast R-CNN, YOLO, FCN, VQA LSTM CNN, Adversarial images

Aaditya Prakash
Apr 11, 2018
[Online version link \(with animations\)](#)

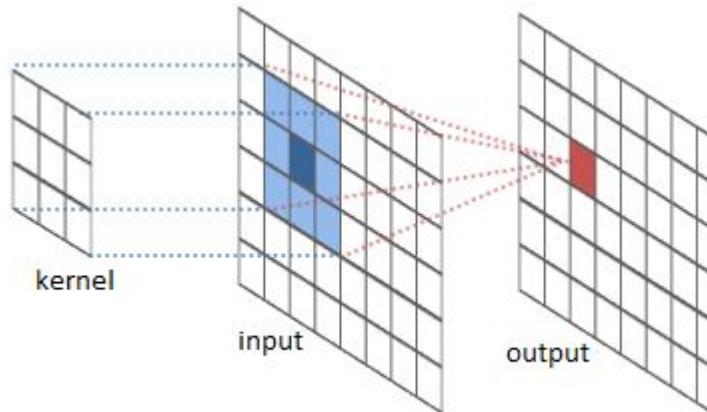
Convolution

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

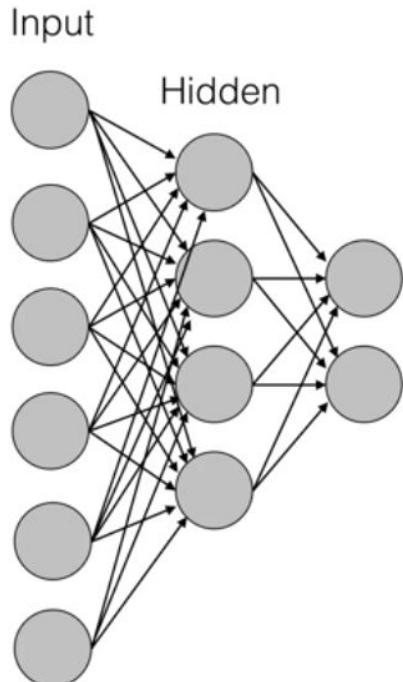
Image

4		

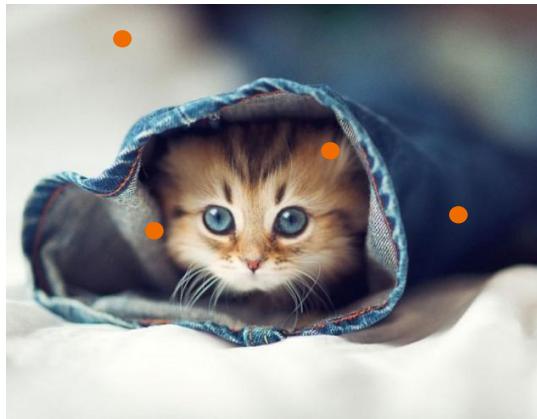
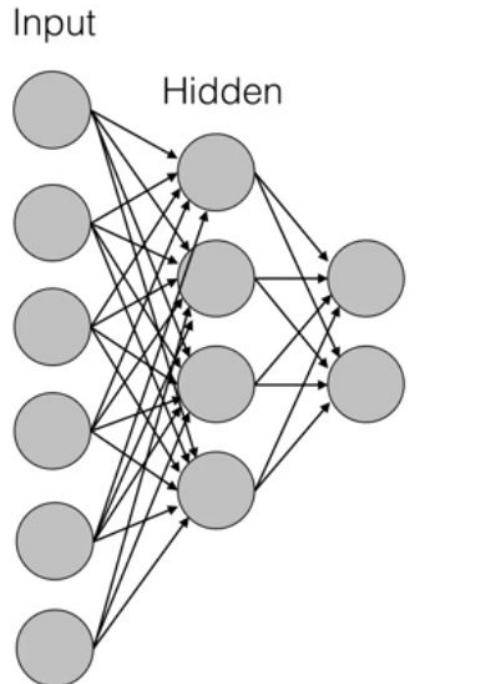
Convolved Feature



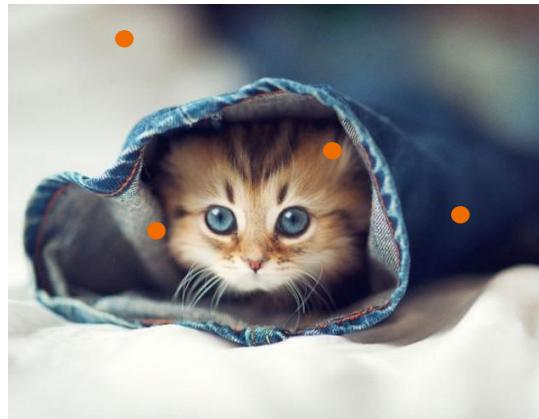
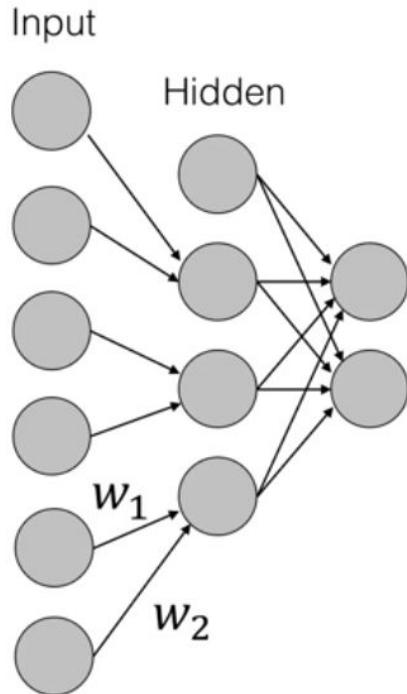
Convolution in Neural Networks



Convolution in Neural Networks

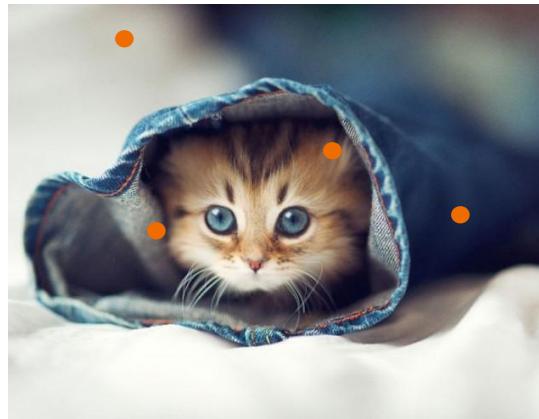
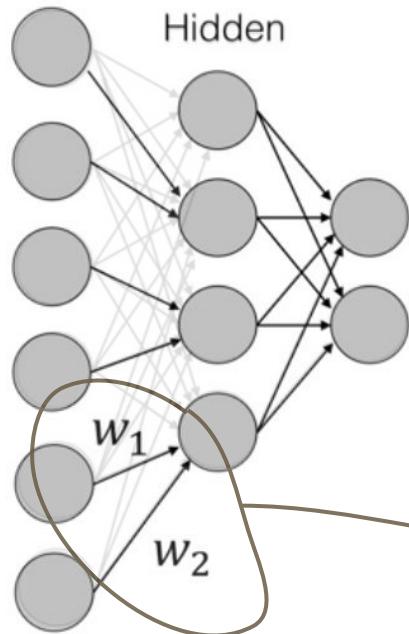


Convolution in Neural Networks



Convolution in Neural Networks

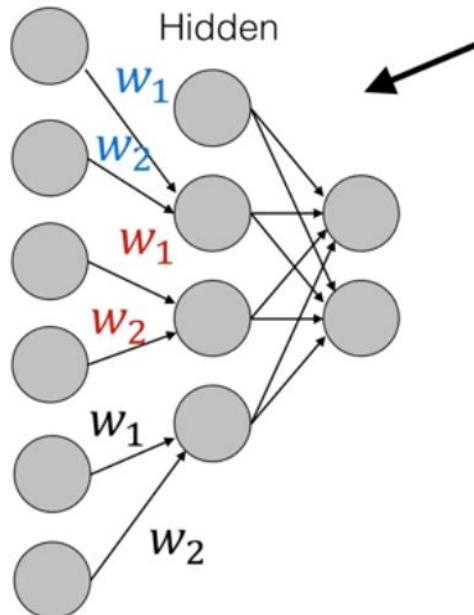
Input



$$y = w_1x_1 + w_2x_2$$

Stride 1-D

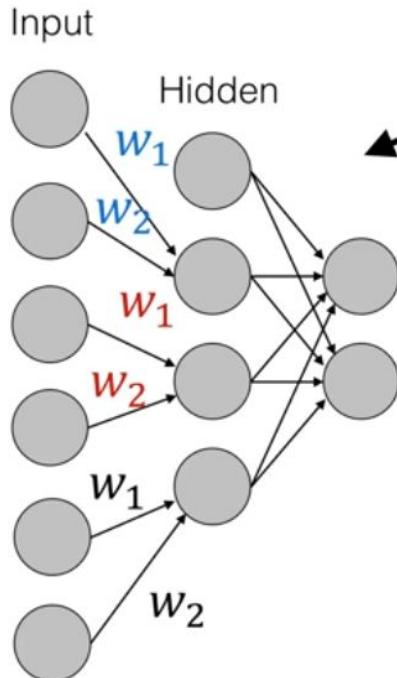
Input



1-d convolution with

- filters: 1
- filter size: 2
- stride: 2

Stride 1-D

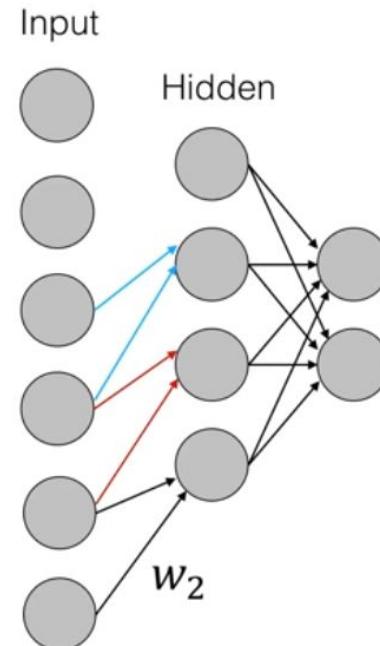


1-d convolution with

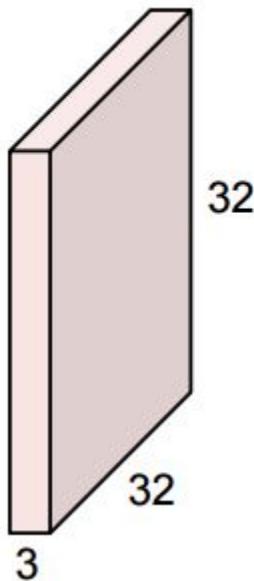
- filters: 1
- filter size: 2
- stride: 2

1-d convolution with

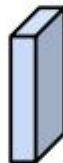
- filters: 1
- filter size: 2
- stride: **1**



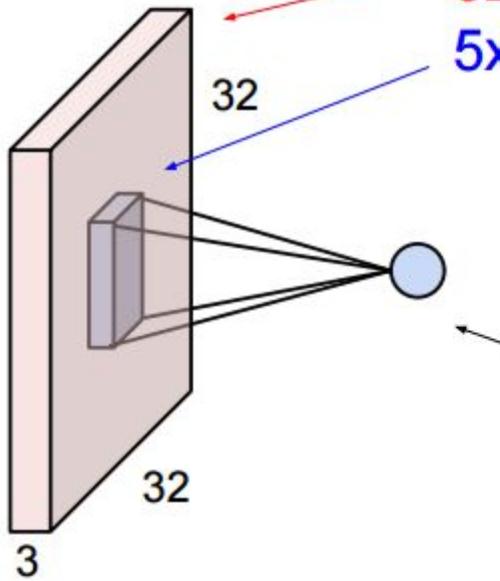
32x32x3 image



5x5x3 filter



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

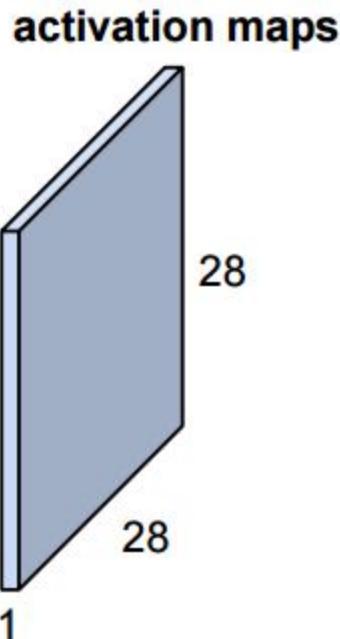
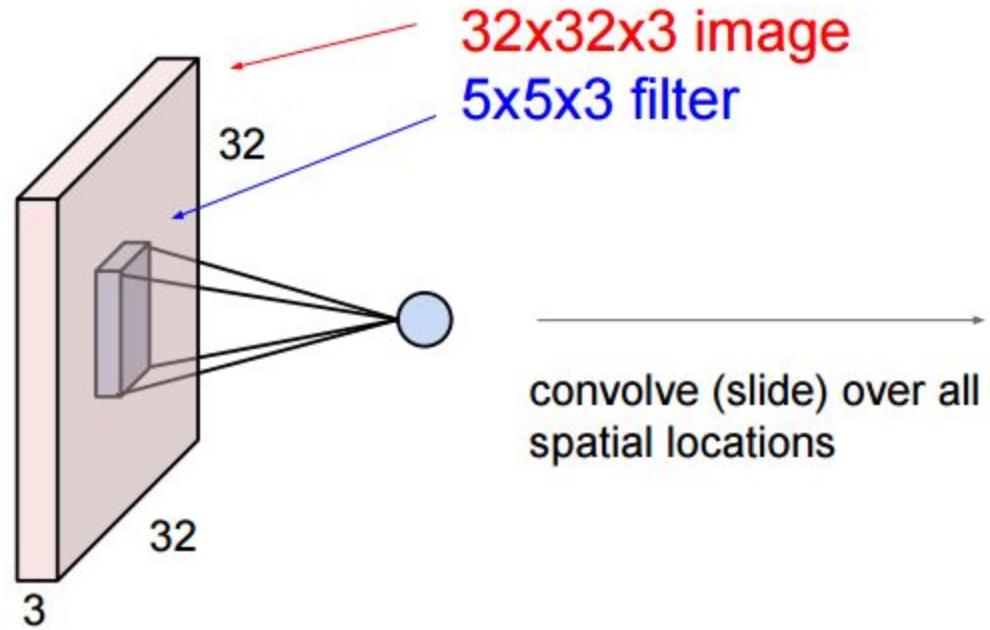


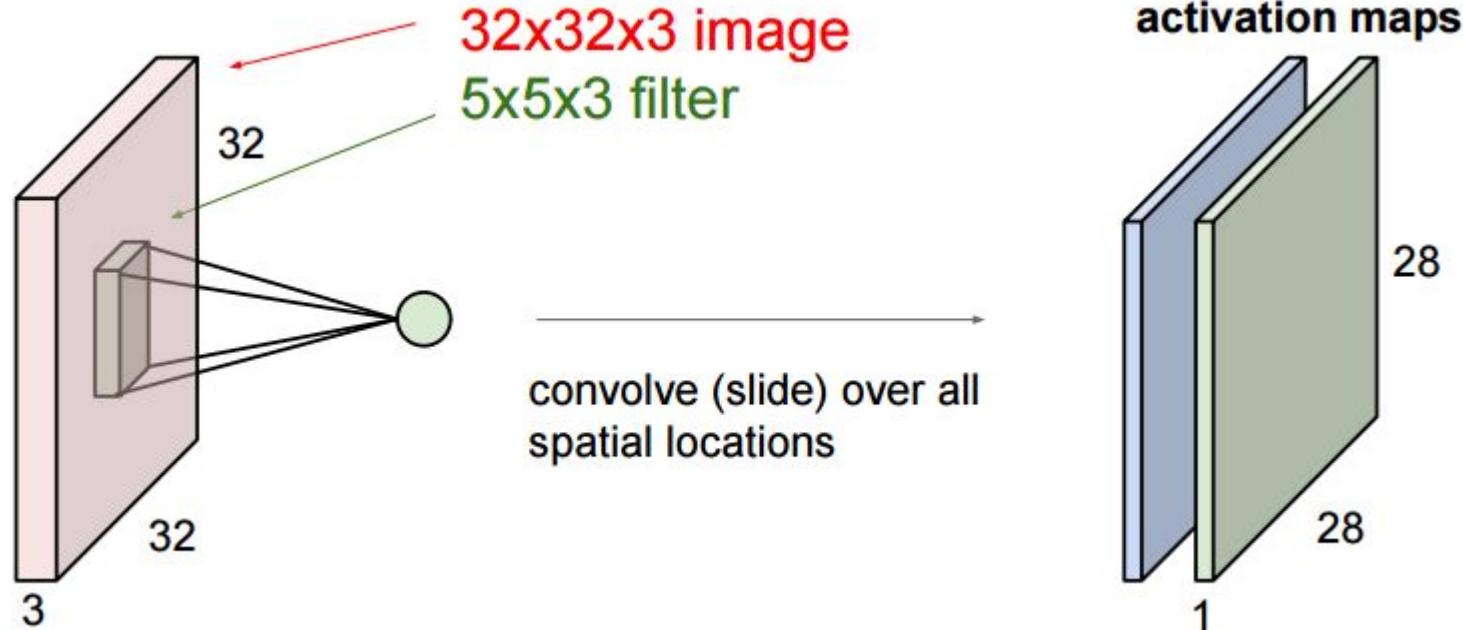
32x32x3 image
5x5x3 filter w

1 number:

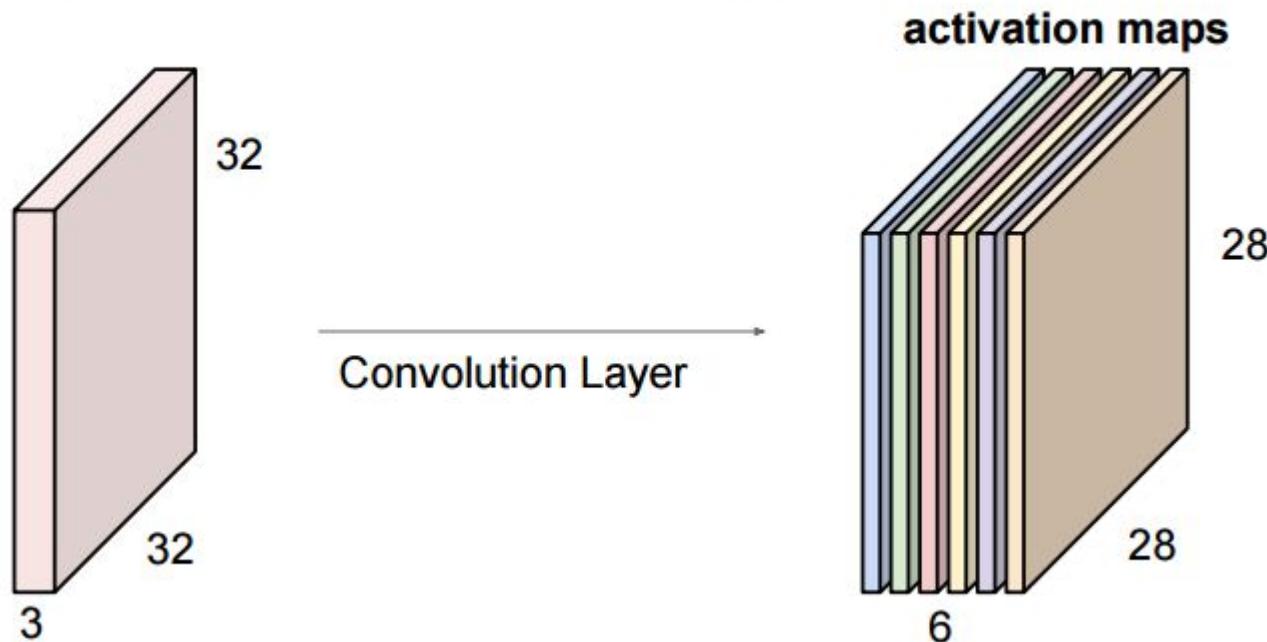
the result of taking a dot product between the
filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

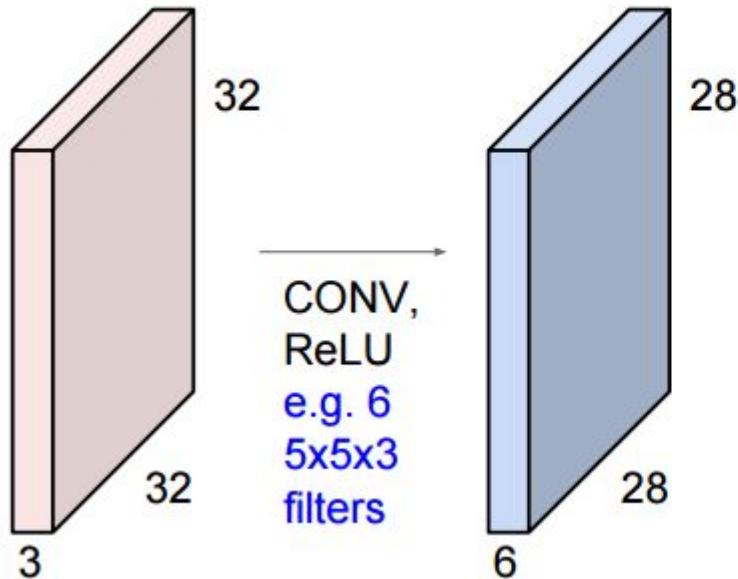




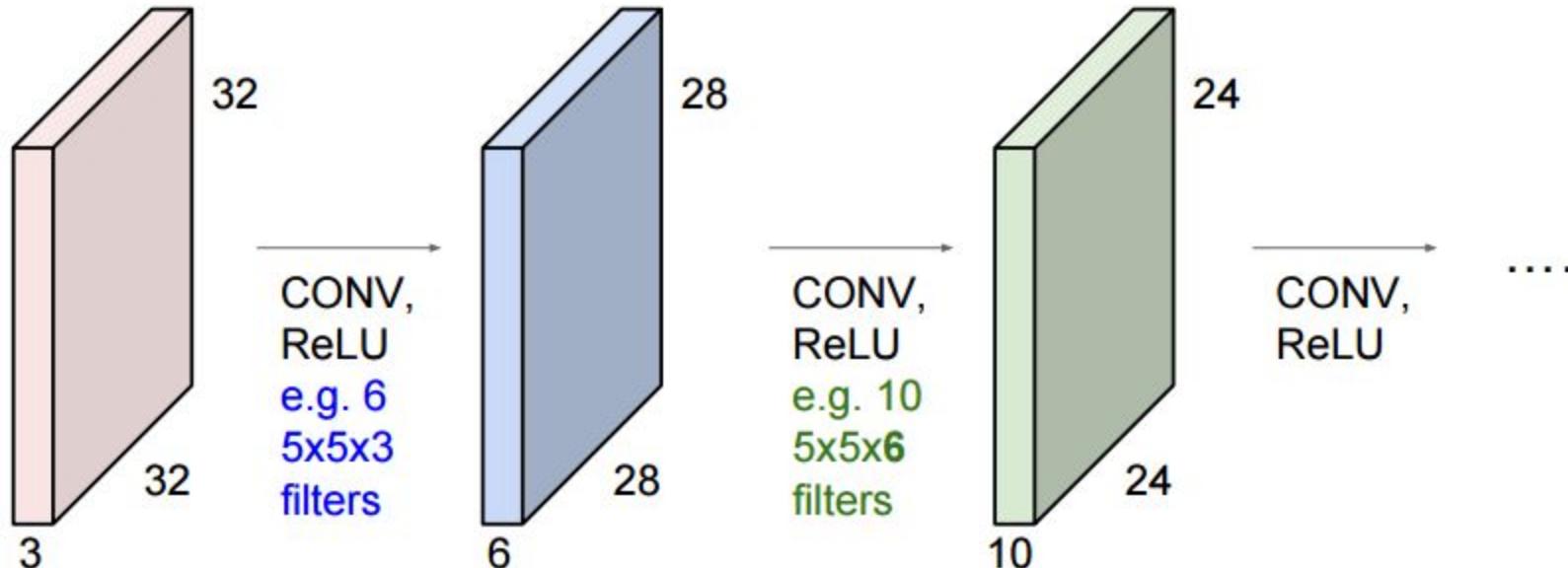
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



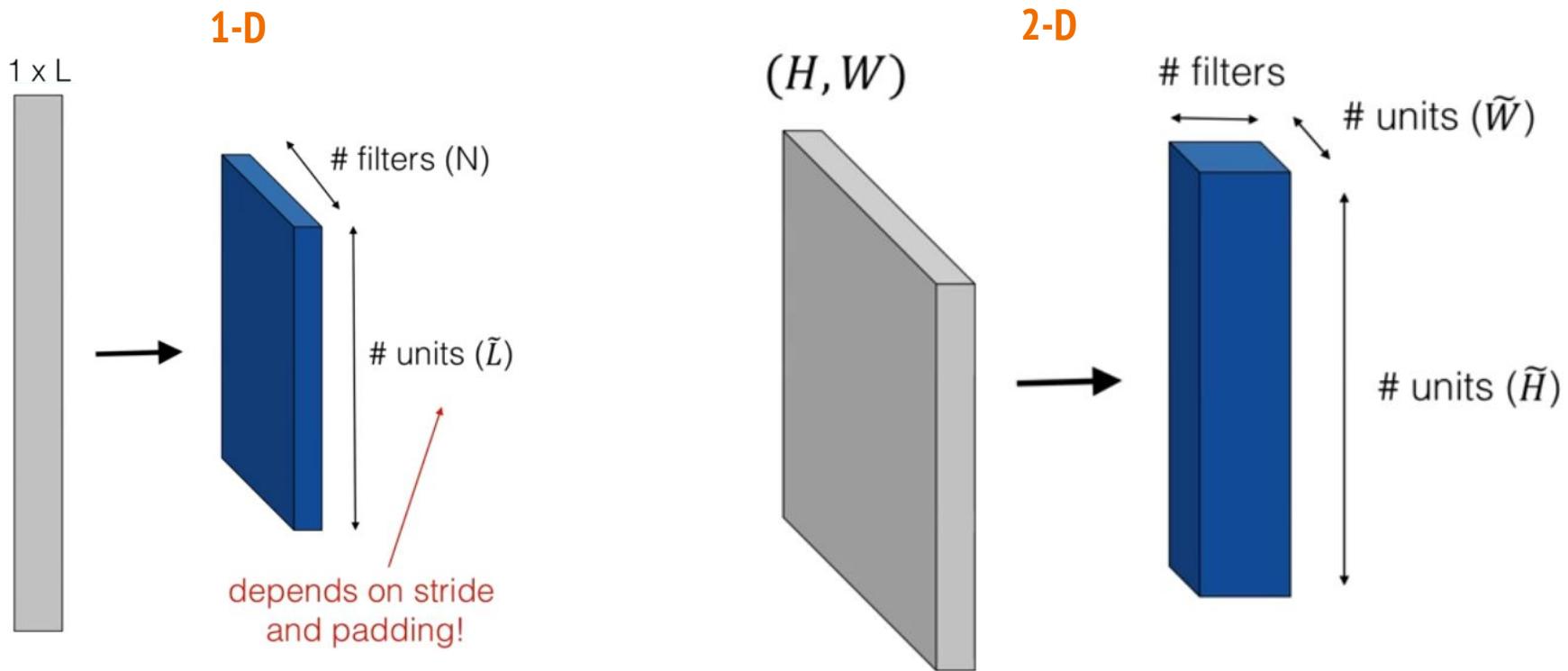
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



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



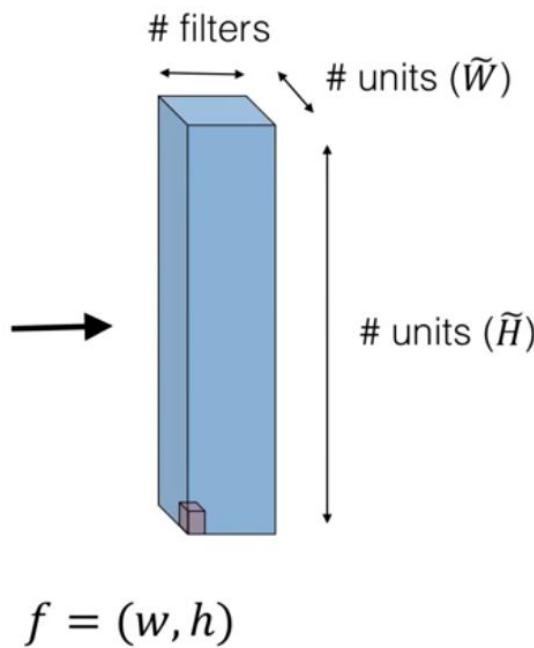
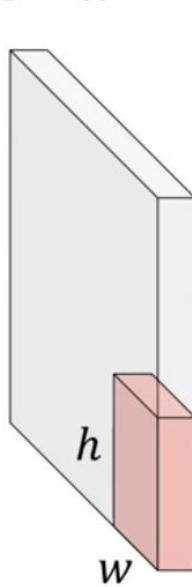
Features / Filters



Features / Filters

2-D

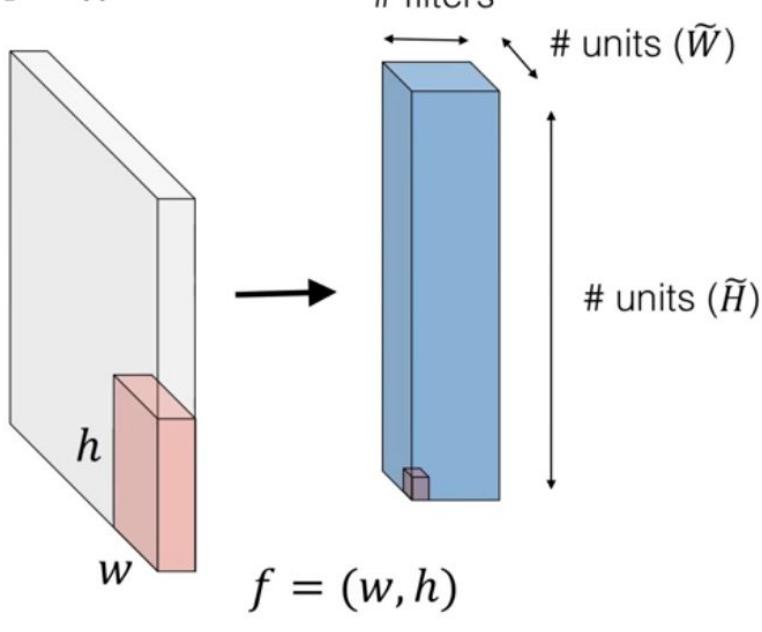
$H \times W$



Features / Filters

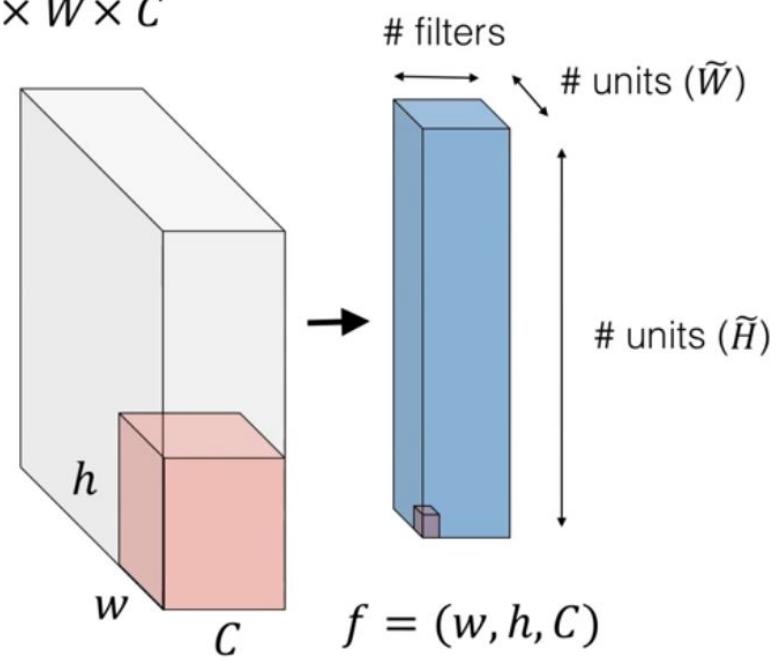
2-D

$H \times W$



3-D

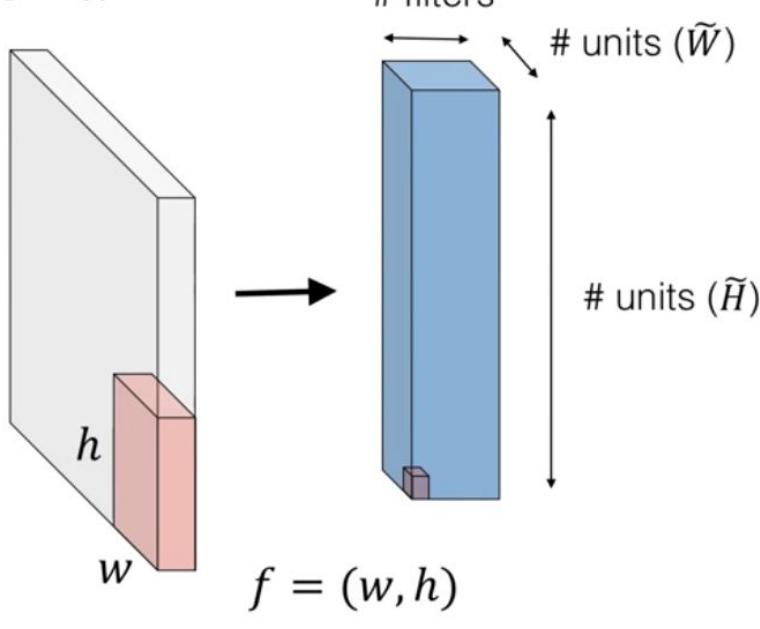
$H \times W \times C$



Features / Filters

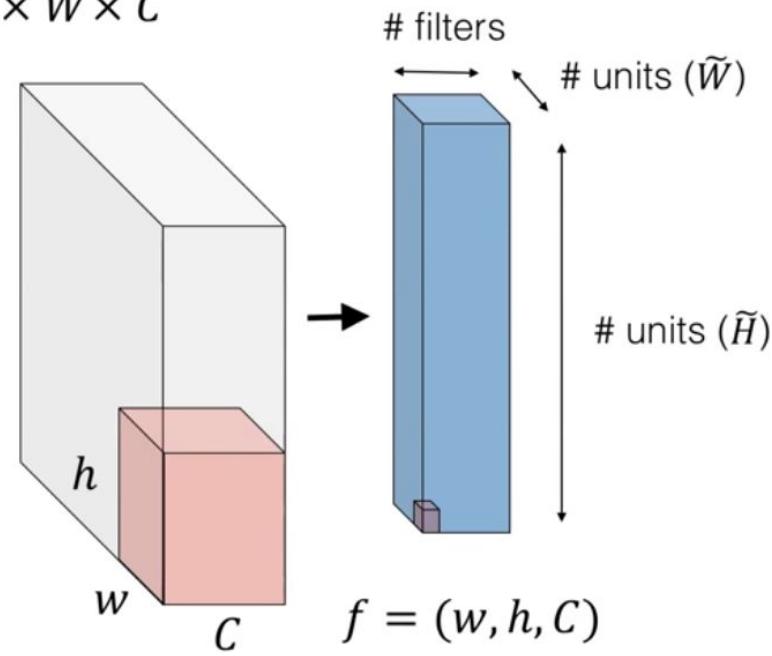
2-D

$H \times W$



3-D 2-D Multichannel

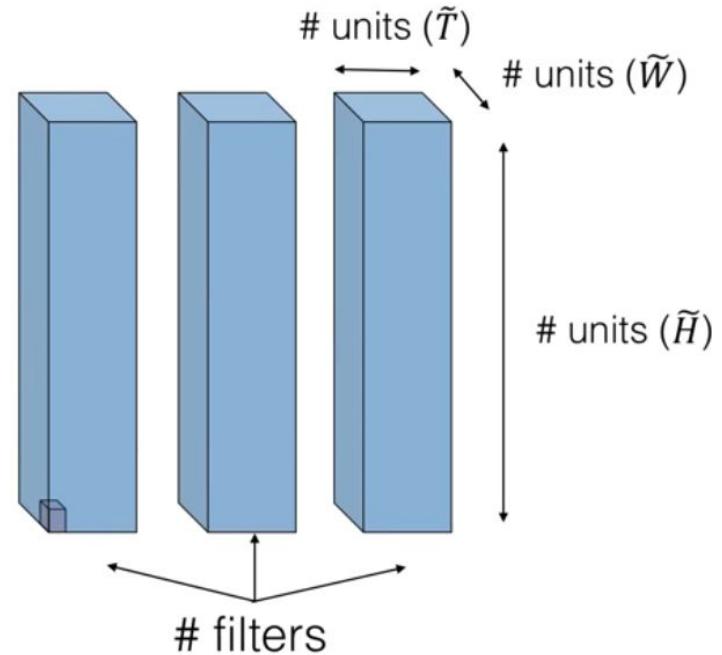
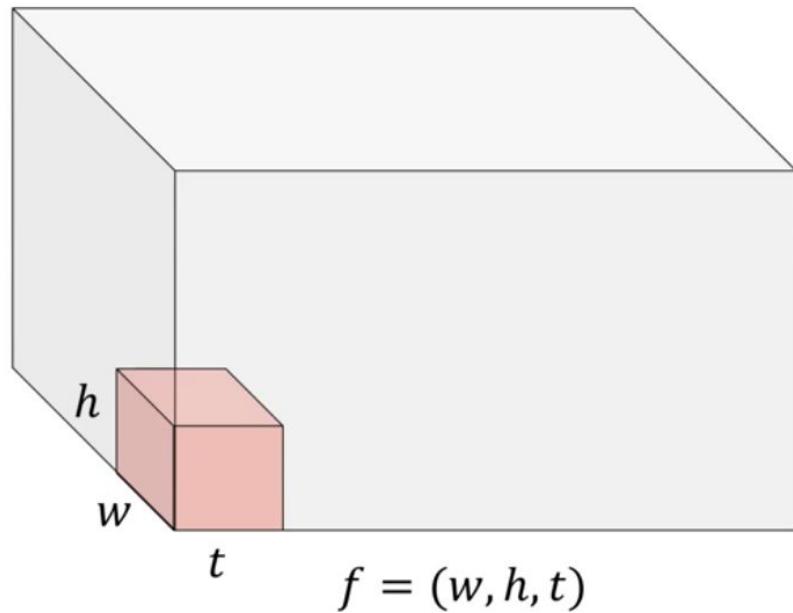
$H \times W \times C$



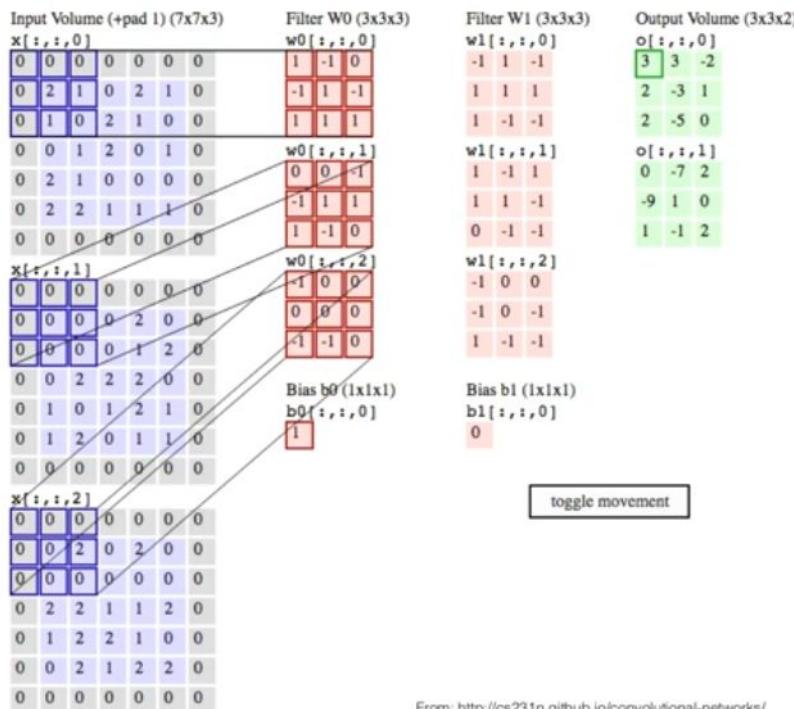
Features / Filters

3-D

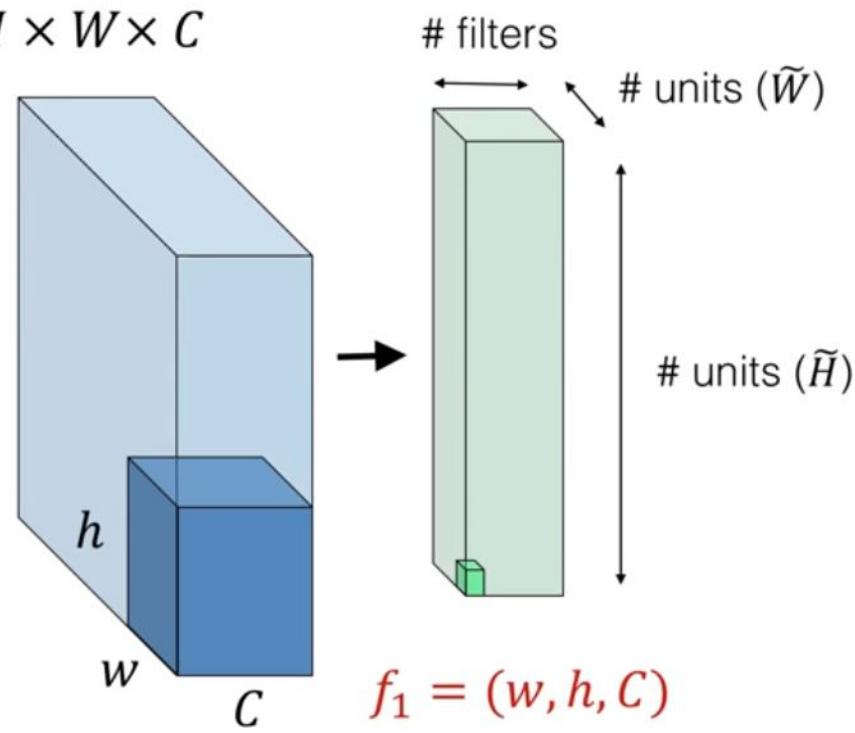
$H \times W \times T$



2-D Convolution



$H \times W \times C$

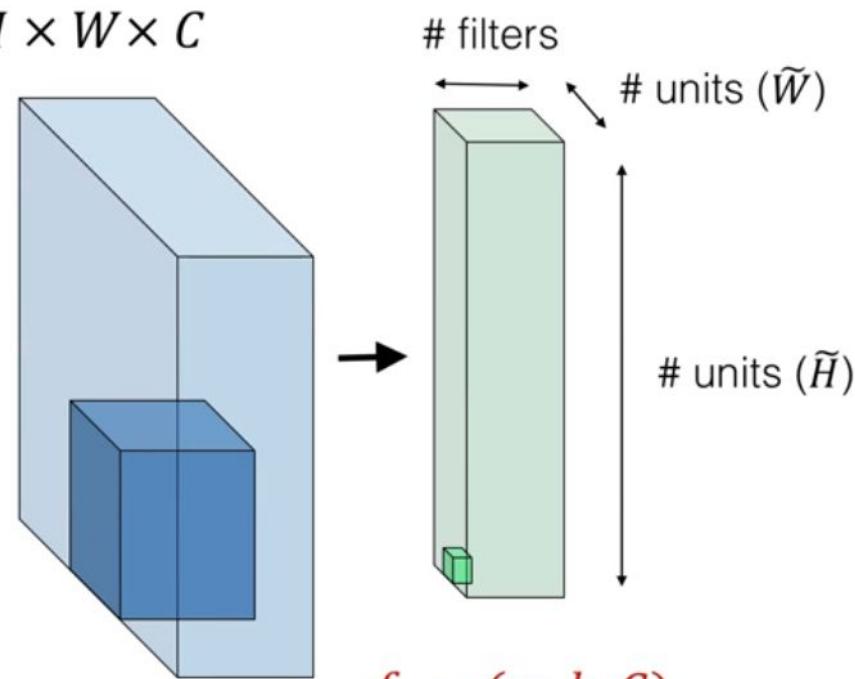


From: <http://cs231n.github.io/convolutional-networks/>

2-D Convolution

Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Output Volume (3x3x2)
x[:, :, 0]	w0[:, :, 0]	o[:, :, 0]
0 0 0 0 0 0 0 0 2 1 0 2 1 0 0 1 0 2 1 0 0 0 0 1 2 0 1 0 0 2 1 0 0 0 0 0 2 2 1 1 1 0 0 0 0 0 0 0 0 0	1 -1 0 -1 1 -1 1 1 1 0 0 -1 -1 1 1 1 -1 0 -1 0 0	3 3 -2 1 1 1 1 -1 -1 2 -3 1 2 -5 0 1 -1 1 0 -7 2 -9 1 0 1 -1 2
x[:, :, 1]	w0[:, :, 1]	o[:, :, 1]
0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 1 2 0 0 0 2 2 2 0 0 0 1 0 1 2 1 0 0 1 2 0 1 1 0 0 0 0 0 0 0 0 0	0 0 -1 0 0 0 -1 -1 0 -1 0 0 0 0 0 -1 0 -1 1 -1 -1	-1 0 0 -1 0 -1 1 -1 -1
x[:, :, 2]	w0[:, :, 2]	o[:, :, 2]
0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 2 2 1 1 2 0 0 1 2 2 1 0 0 0 0 2 1 2 2 0 0 0 0 0 0 0 0 0	b0 (1x1x1) b0[:, :, 0]	b1 (1x1x1) b1[:, :, 0]
		0
	toggle movement	

$H \times W \times C$



From: <http://cs231n.github.io/convolutional-networks/>

Pooling

1-D

0	1	4	9
3	2	5	8
1	2	3	1
3	1	7	4

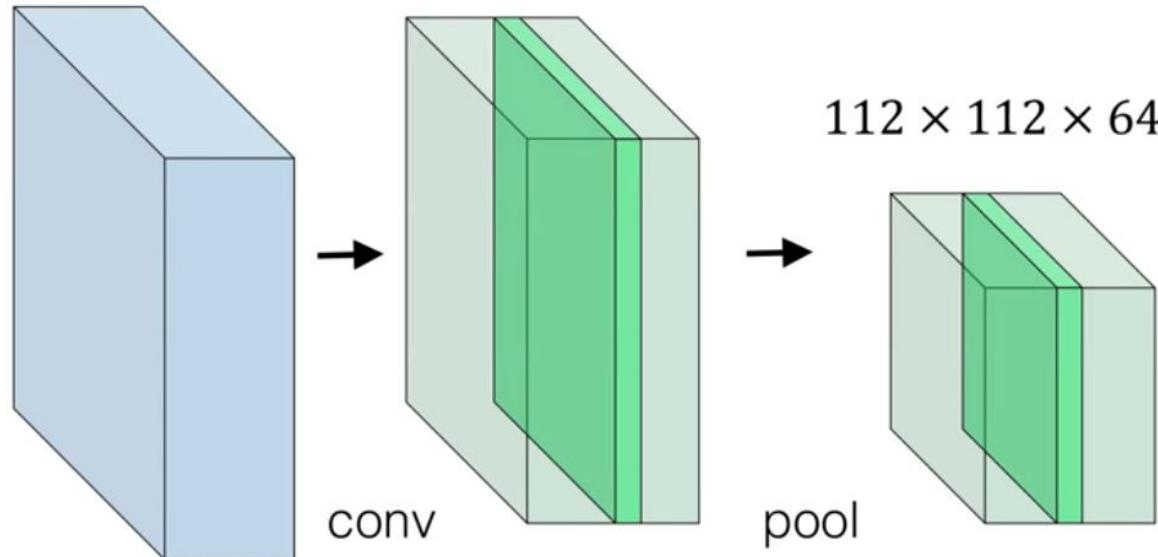
Max pool:
2x2 filters
Stride 2

3	9
3	7

$224 \times 224 \times 3$

2-D

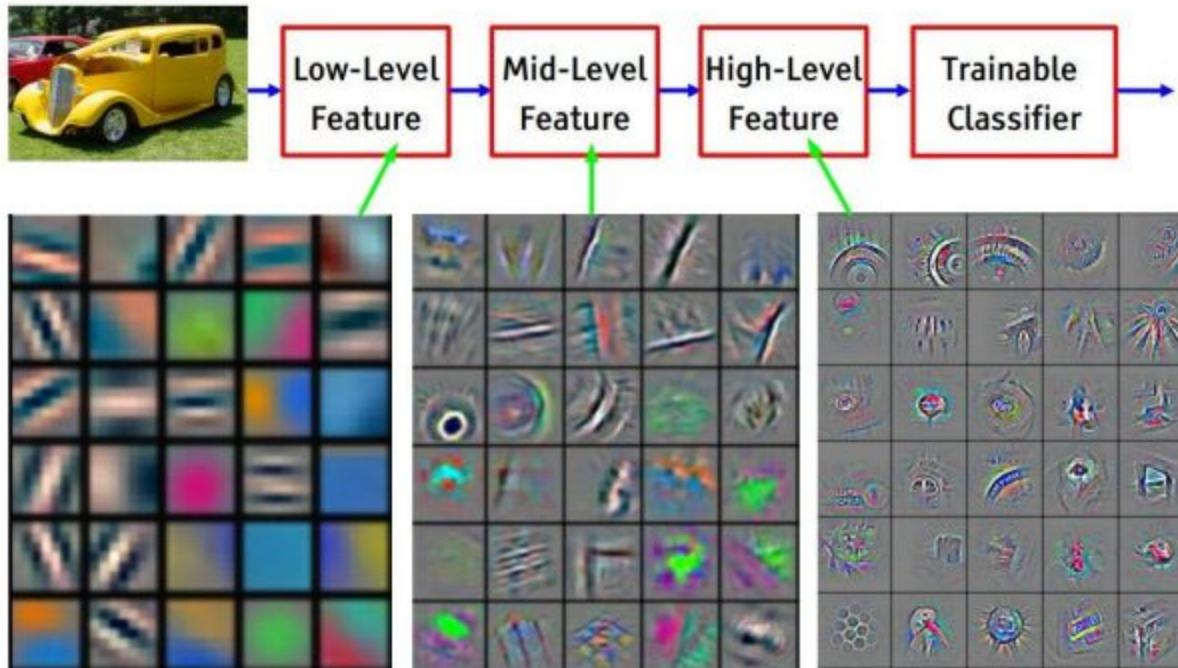
$224 \times 224 \times 64$



Inside Convolutional Neural Network



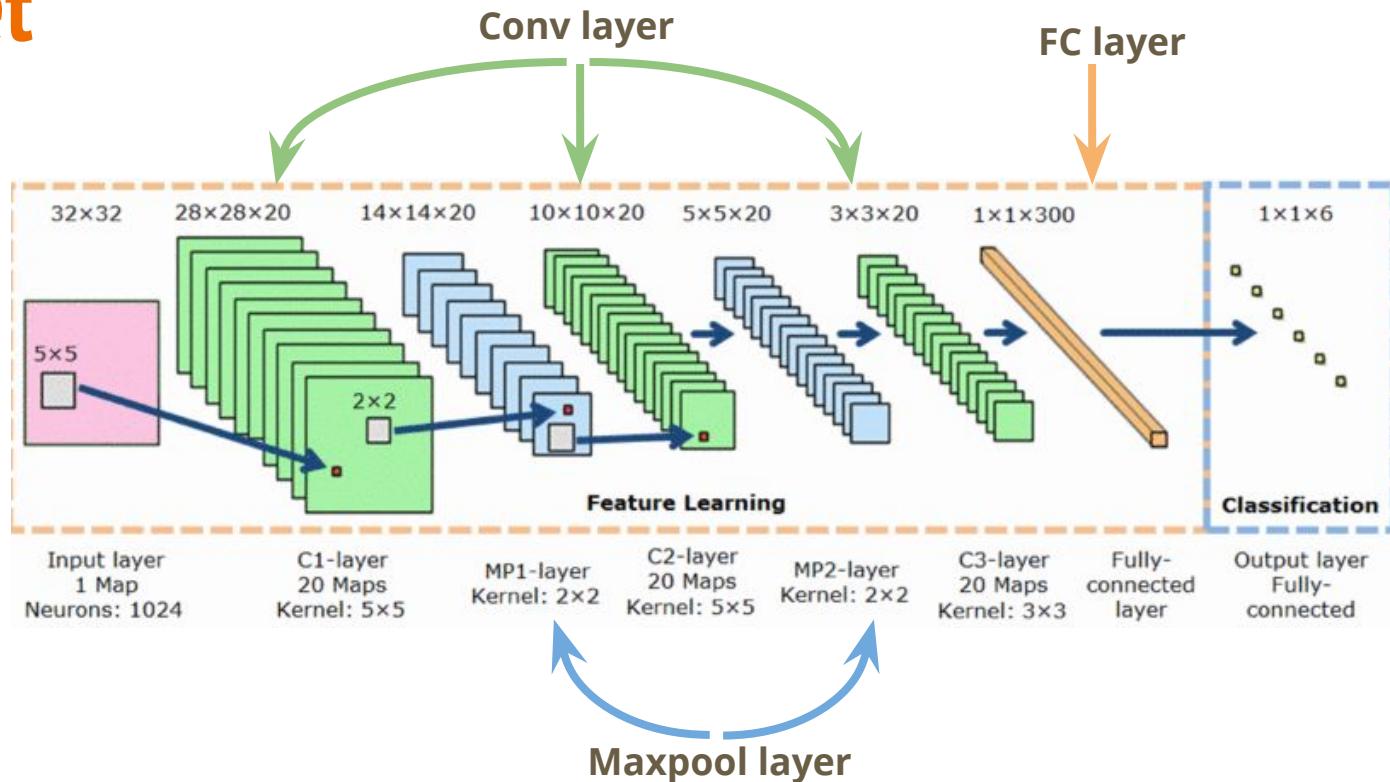
Inside Convolutional Neural Network



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



LeNet



*Original LeNet-5 has two FCL at the end, and filter sizes are slightly different

AlexNet (2012 Winner)

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

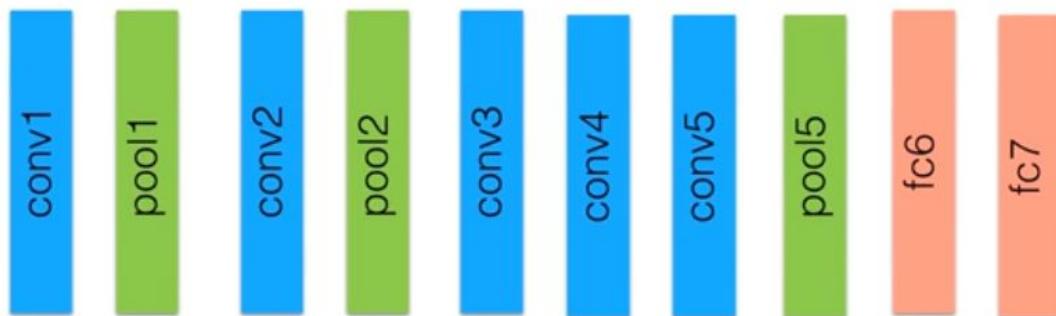
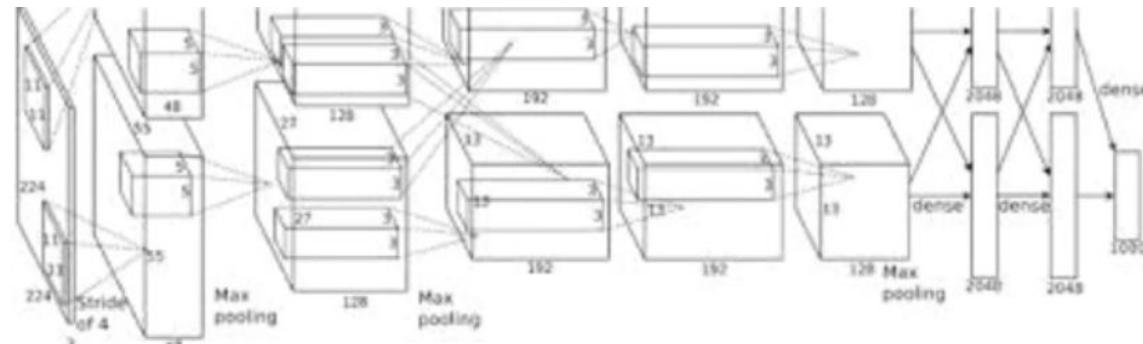
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2 [4096]

FC6: 4096 neurons [4096]

FC7: 4096 neurons [1000]

FC8: 1000 neurons (class scores)



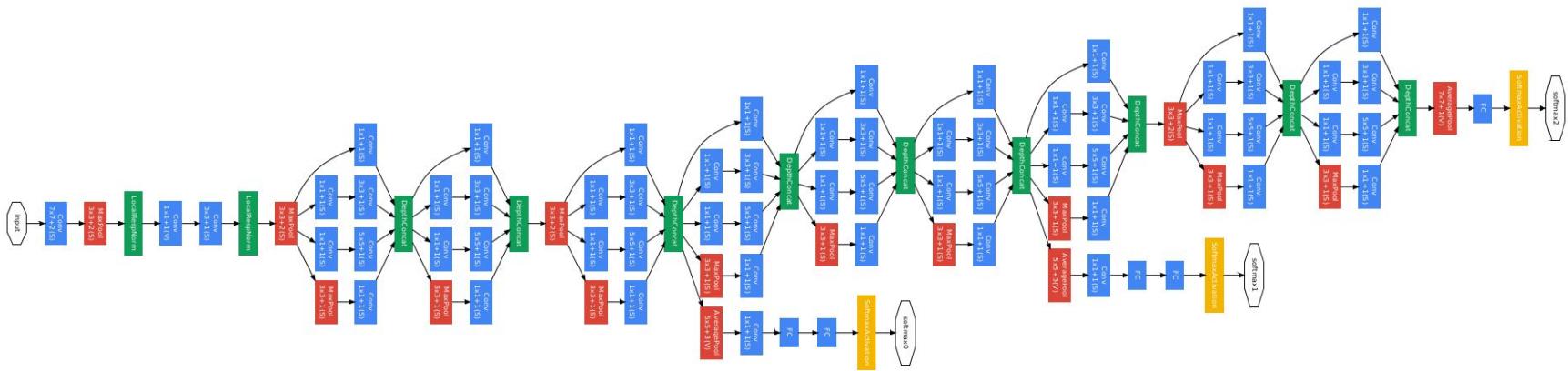
VGG Net

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
 CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$
 POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$
 CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$
 POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
 POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
 POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0
 FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$
 FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$
 FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
conv3-64	conv3-64	conv3-64
conv3-64	conv3-64	conv3-64
maxpool		
conv3-128	conv3-128	conv3-128
conv3-128	conv3-128	conv3-128
maxpool		
conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256
conv1-256		conv3-256
maxpool		
conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512
conv1-512		conv3-512
maxpool		
conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512
conv1-512		conv3-512
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		



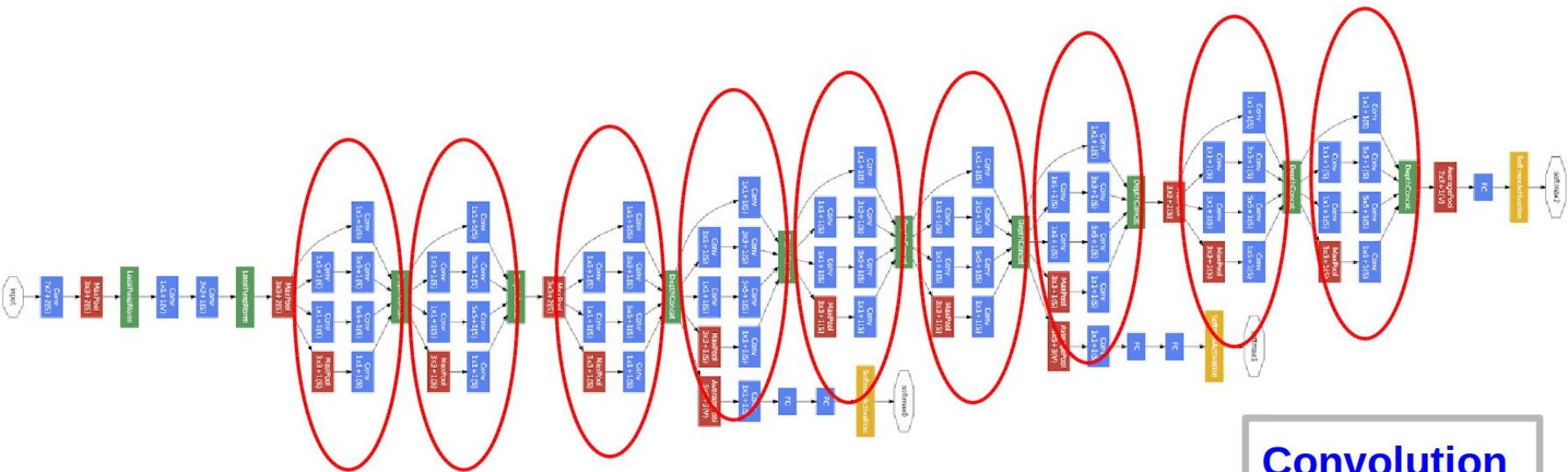
GoogLeNet (2014 winner)



Peasant's network vs Google's

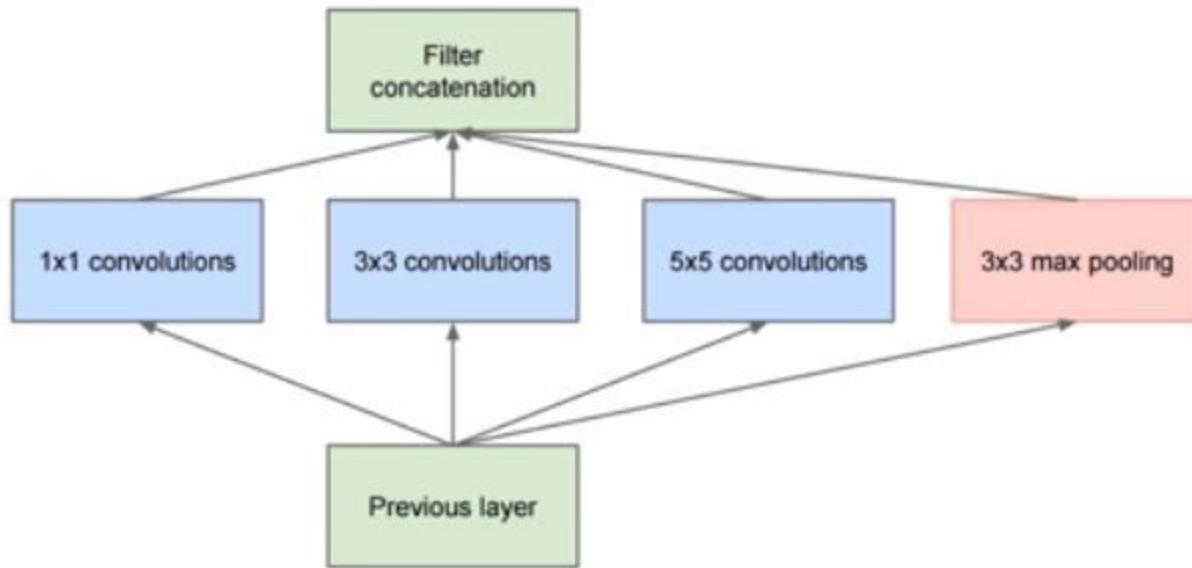


GoogLeNet (2014 winner)

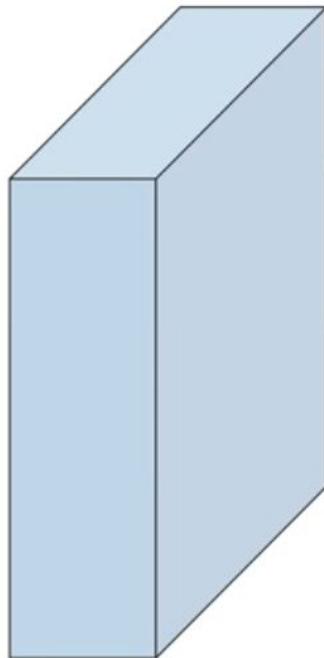


Convolution
Pooling
Softmax
Concat/Normalize

Inception



Inception

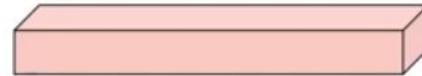


$1 \times 1?$

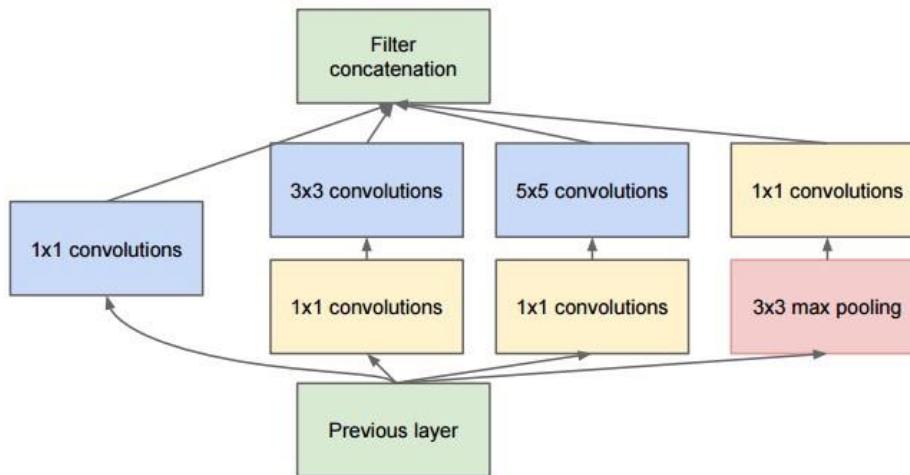
$3 \times 3?$

$5 \times 5?$

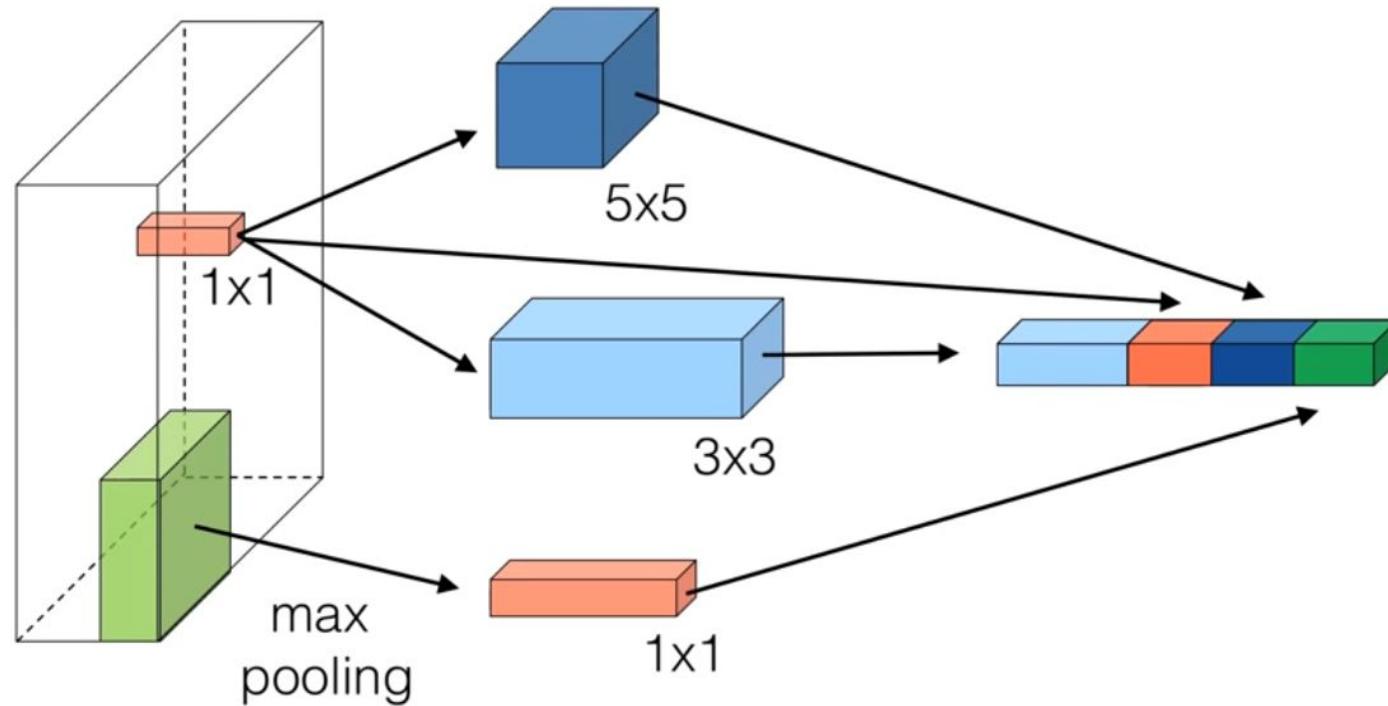
Pooling?



Inception



Inception



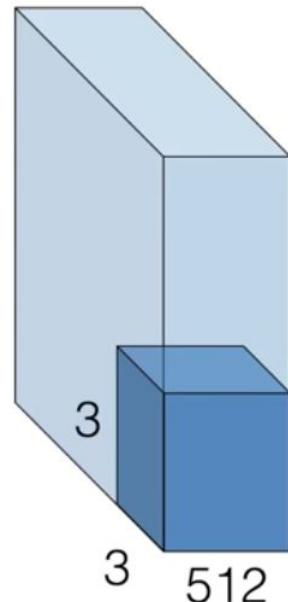
Inception

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M

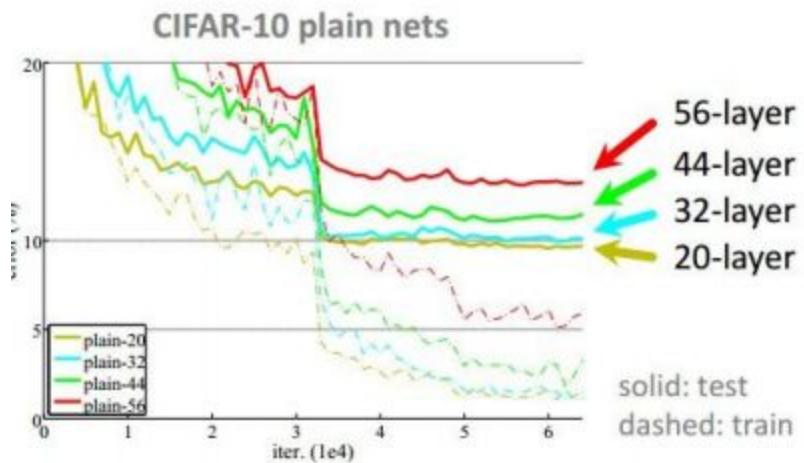
If we used (3 × 3, 512) convolution:

(3 × 3 × 512 × 512) parameters = 2.359 million parameters

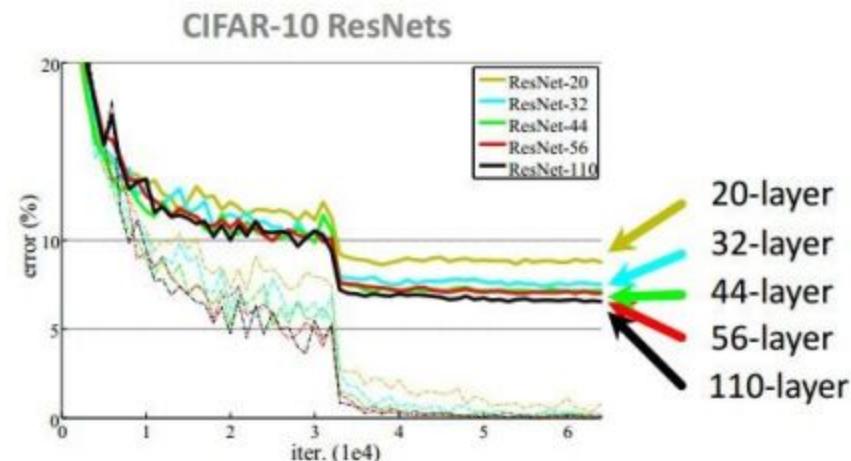
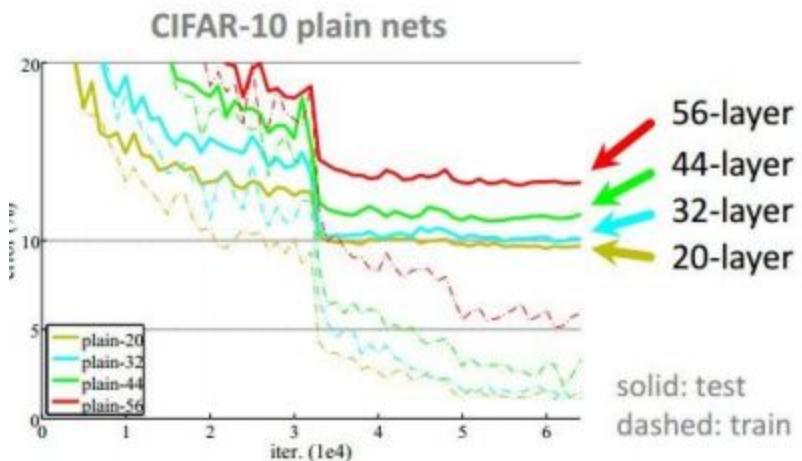
Inception module: 437K parameters



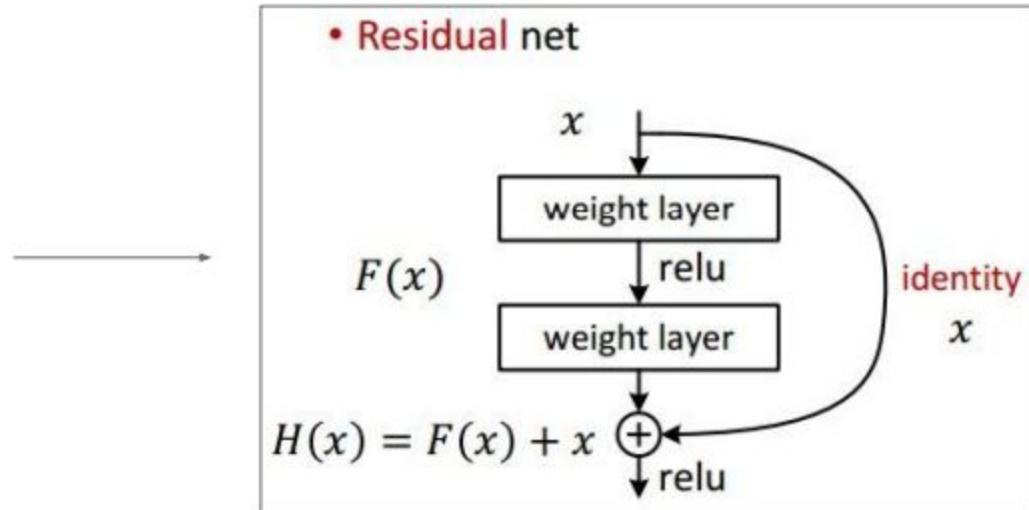
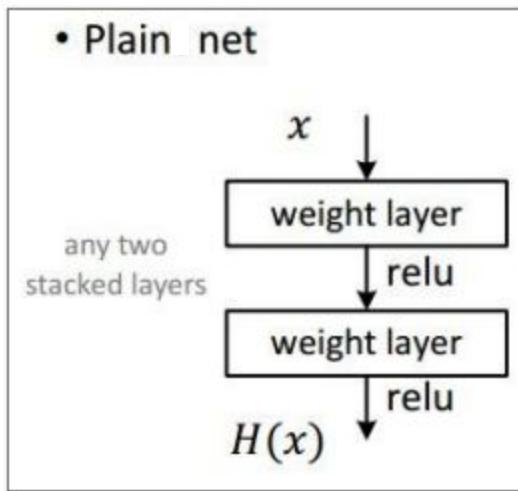
Going Deeper



Going Deeper



Resnet (2015 winner)

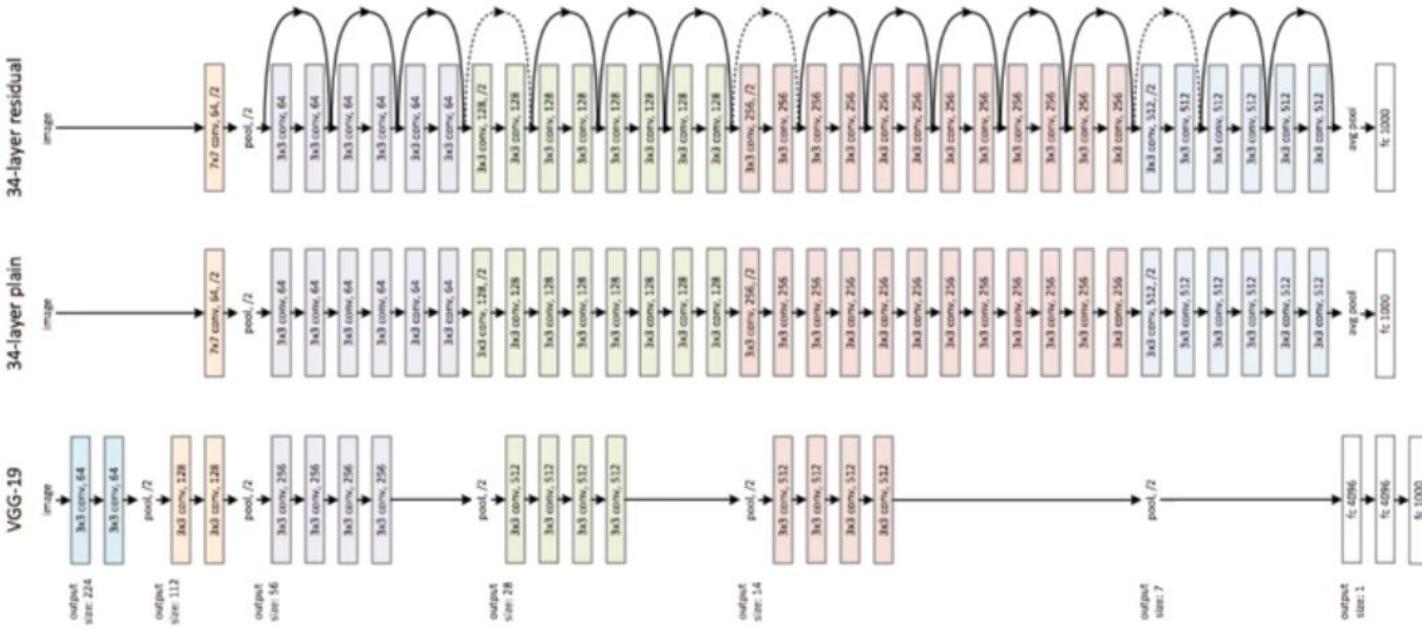


Resnet - comparison with other nets

34-residual

34-plain

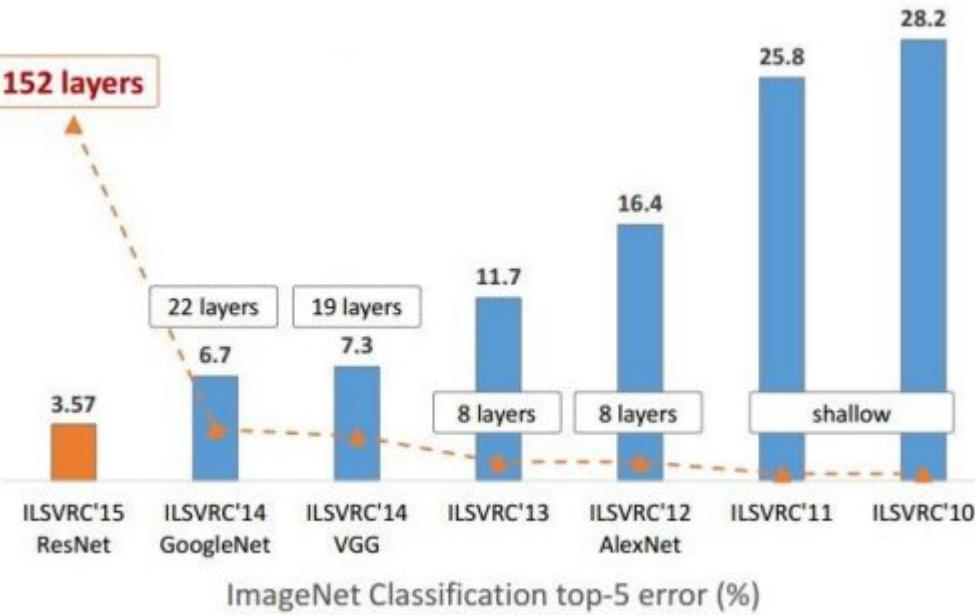
VGG



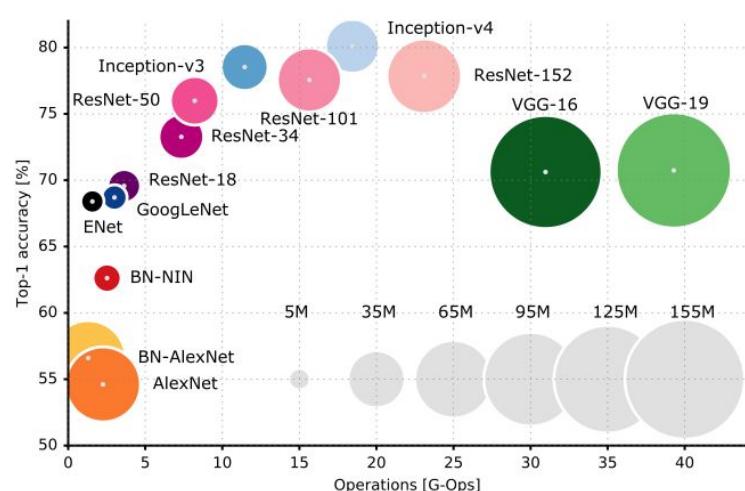
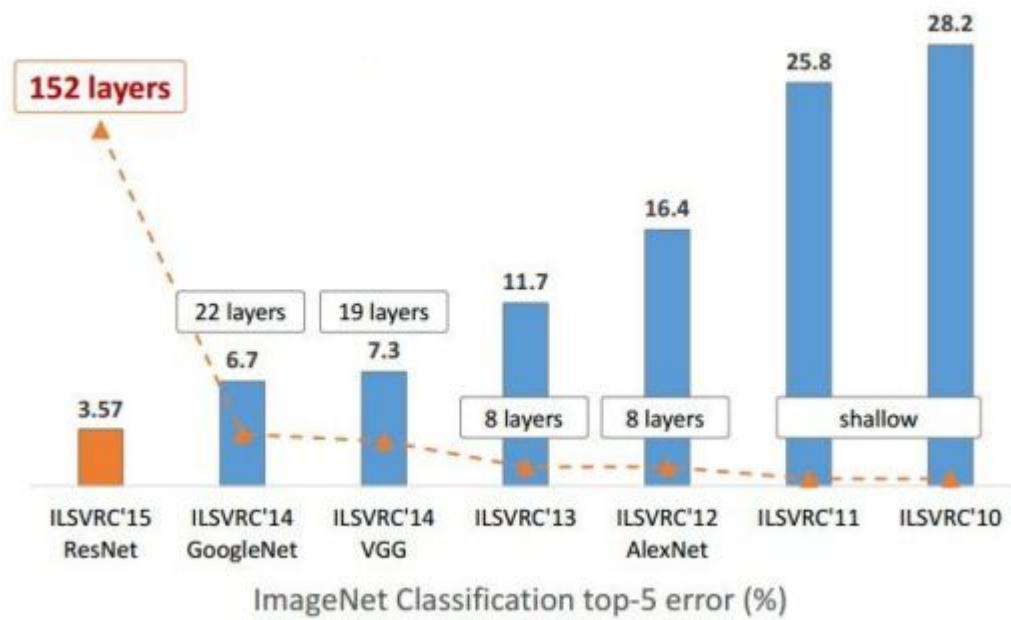
Resnet - Number of layer comparison

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

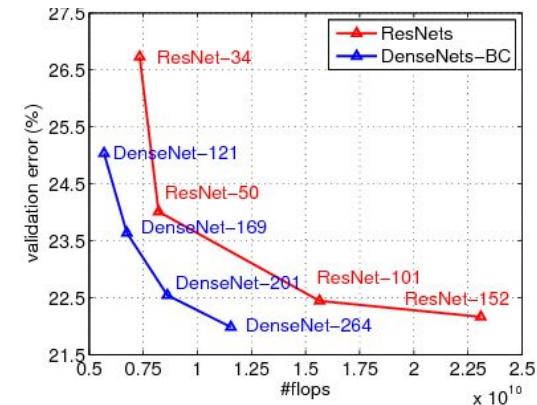
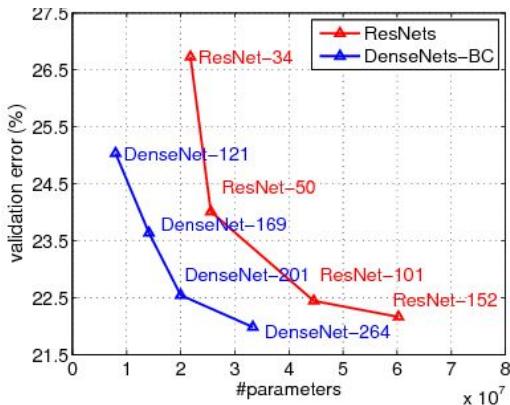
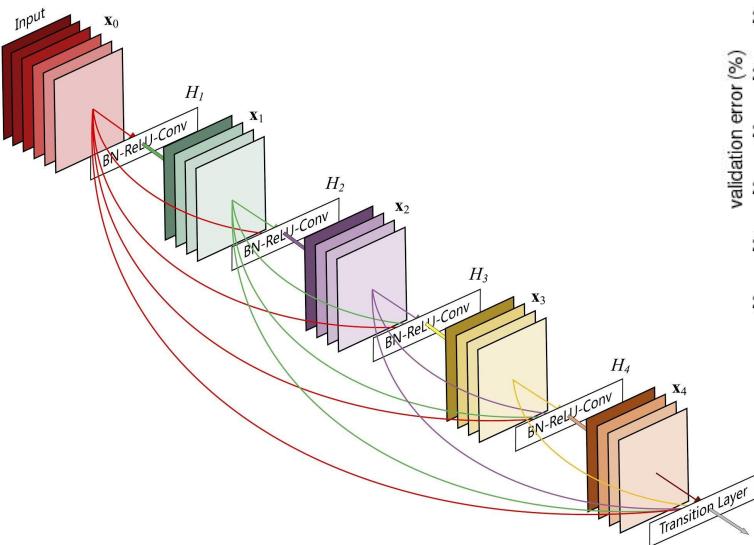
Depth - The Stigma



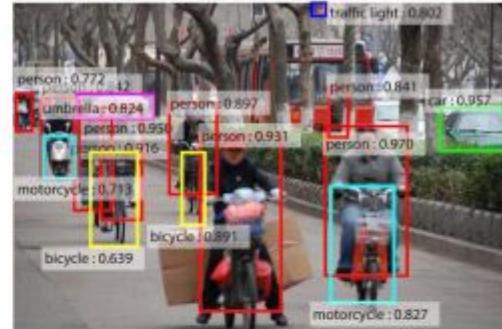
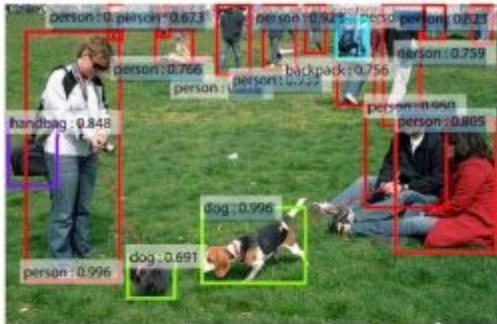
Depth - The Stigma



DenseNet



Localization and Detection



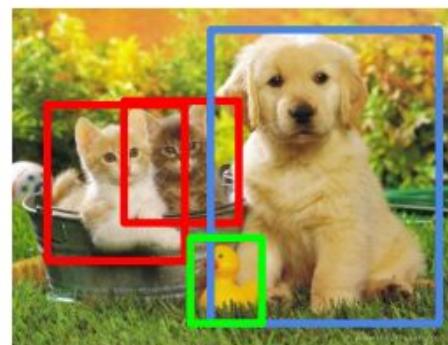
Classification vs Localization vs Detection vs Segmentation



CAT



CAT



CAT, DOG, DUCK

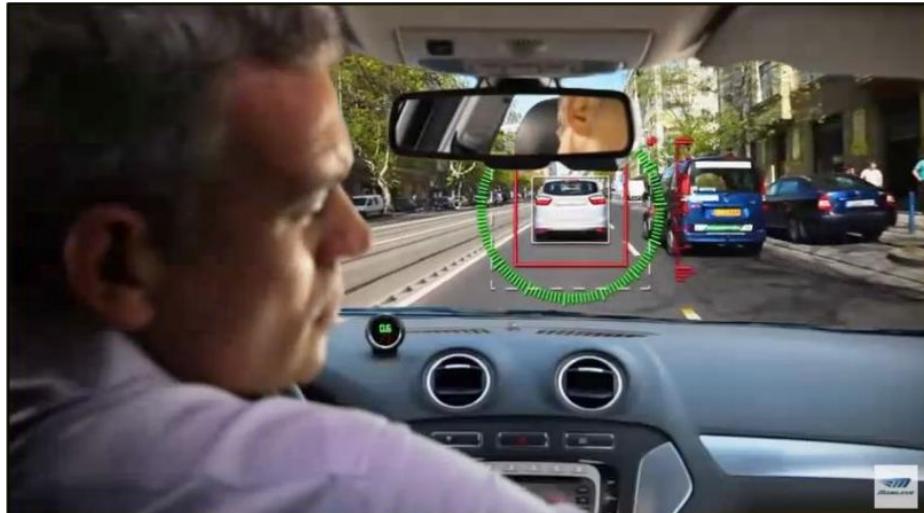


CAT, DOG, DUCK

Single object

Multiple objects

Detection - Fast and Accurate

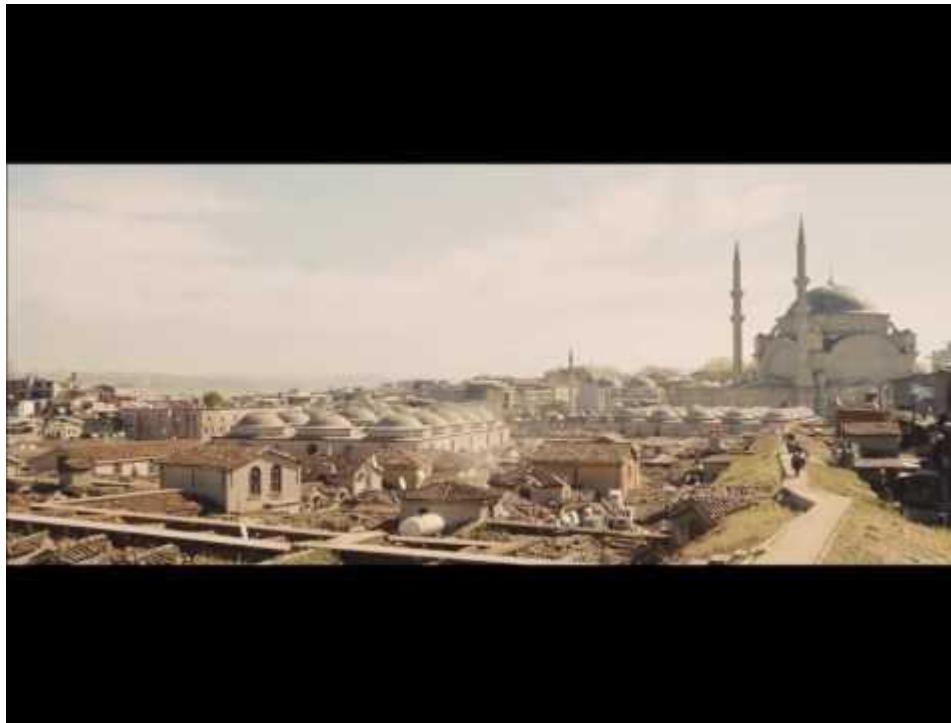


MobilEye ([video](#))

Nikon S60

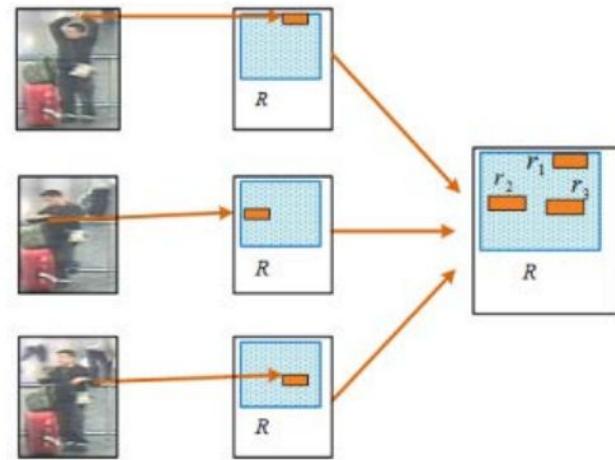


Demo

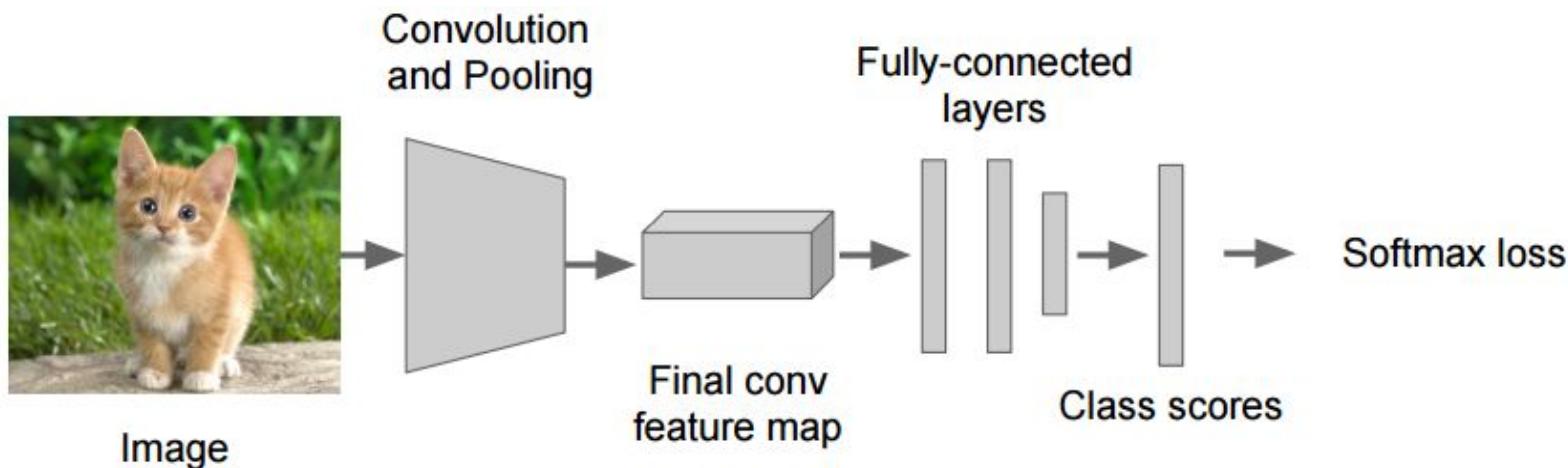


Before deep learning

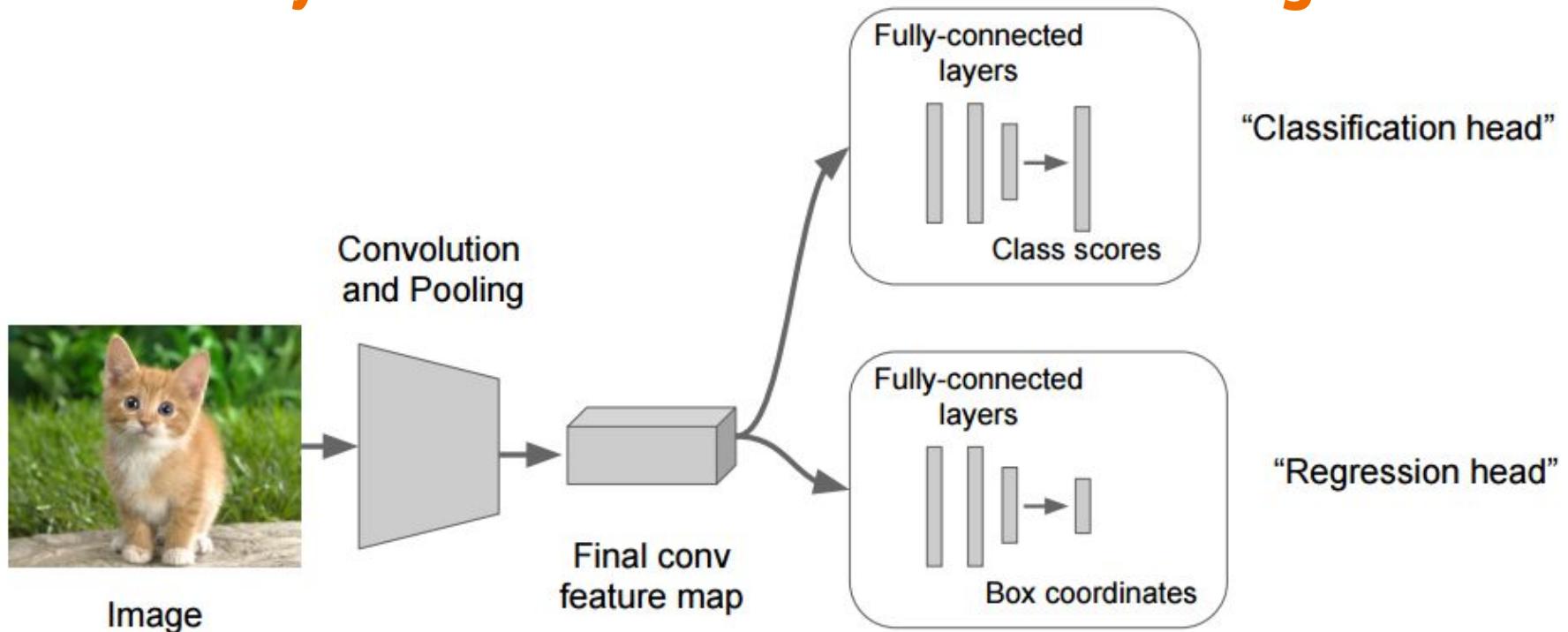
- Deformable Parts Model
 - low-level features based on HOG, SVM classifier
 - mAP: 33%
- Selective Search
 - Bag-of-words with SIFT features, merge similar segments found by segmentation
 - mAP: 35%
- Regionlets
 - HOG, LBP, Covariance features, relative position of sub-parts
 - mAP: 40%



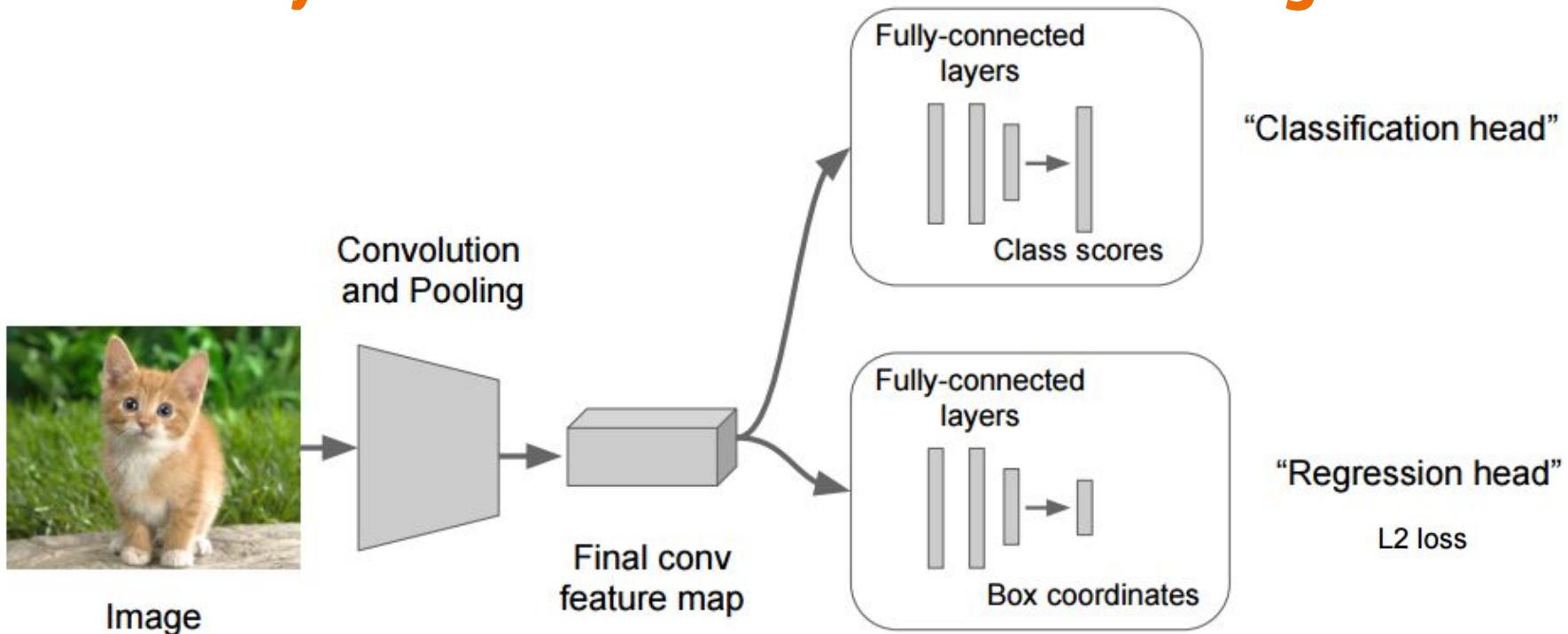
Naive way to use CNN for localization - Regression



Naive way to use CNN for localization - Regression

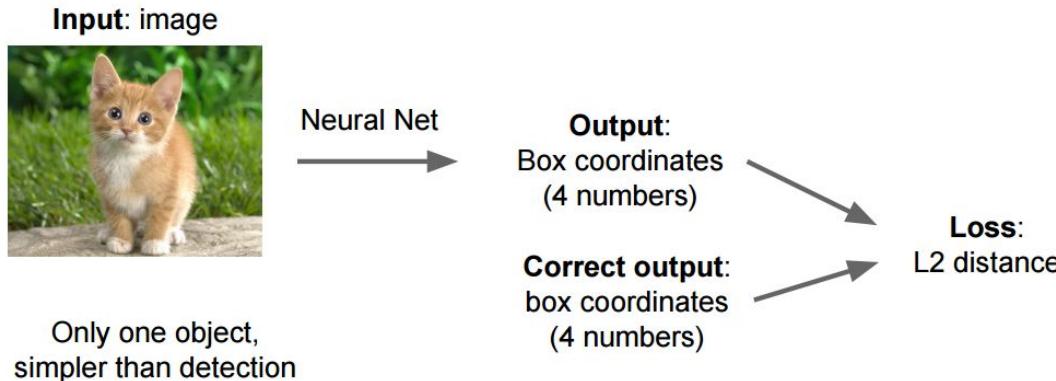


Naive way to use CNN for localization - Regression

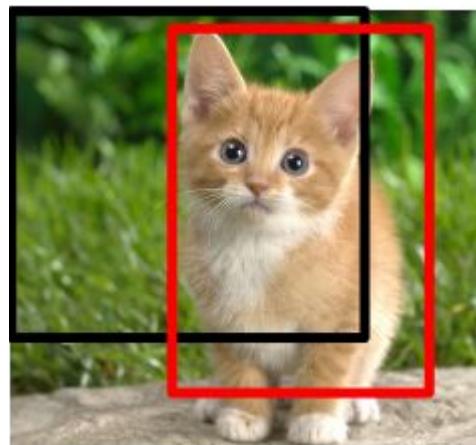


Naive way to use CNN for localization - Regression

- Limited to 'one' class per image
- Limited to 'one' instance of object per image
- Cannot separate object parts - like ear, head, eyes
- Limited to 'rectangle' object boundaries

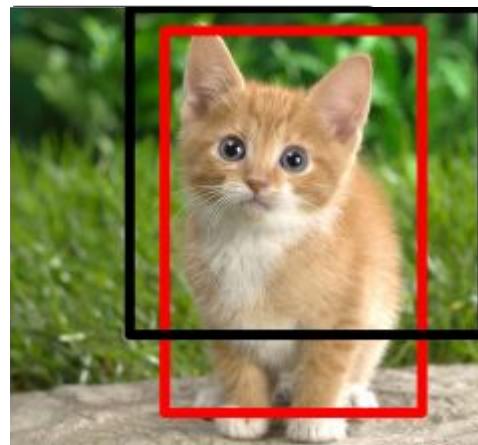


Better way - Sliding window



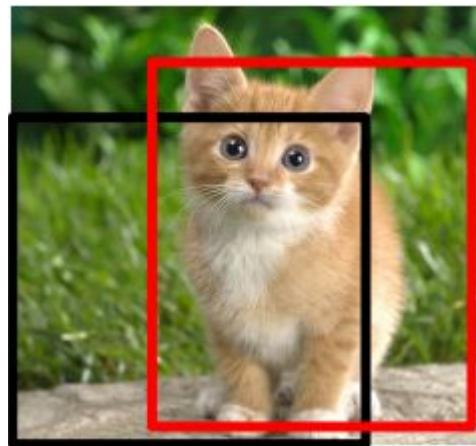
0.5	

Better way - Sliding window



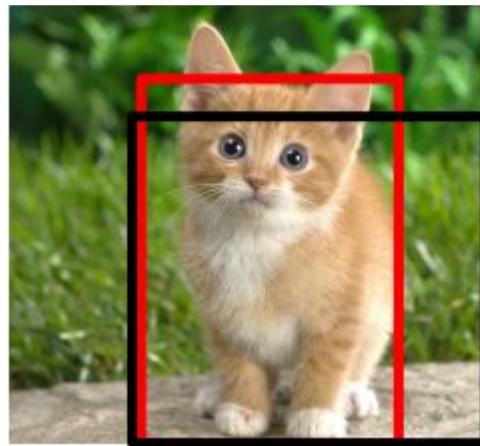
0.5	0.75

Better way - Sliding window



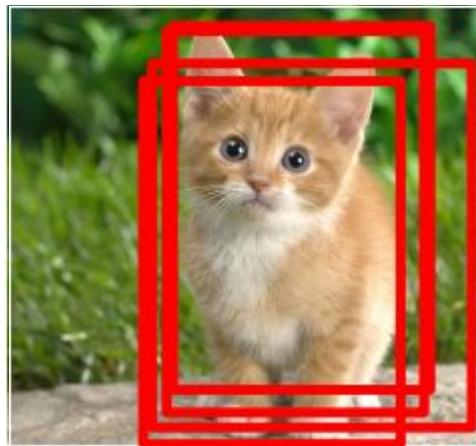
0.5	0.75
0.6	

Better way - Sliding window



0.5	0.75
0.6	0.8

Better way - Sliding window



0.5	0.75
0.6	0.8

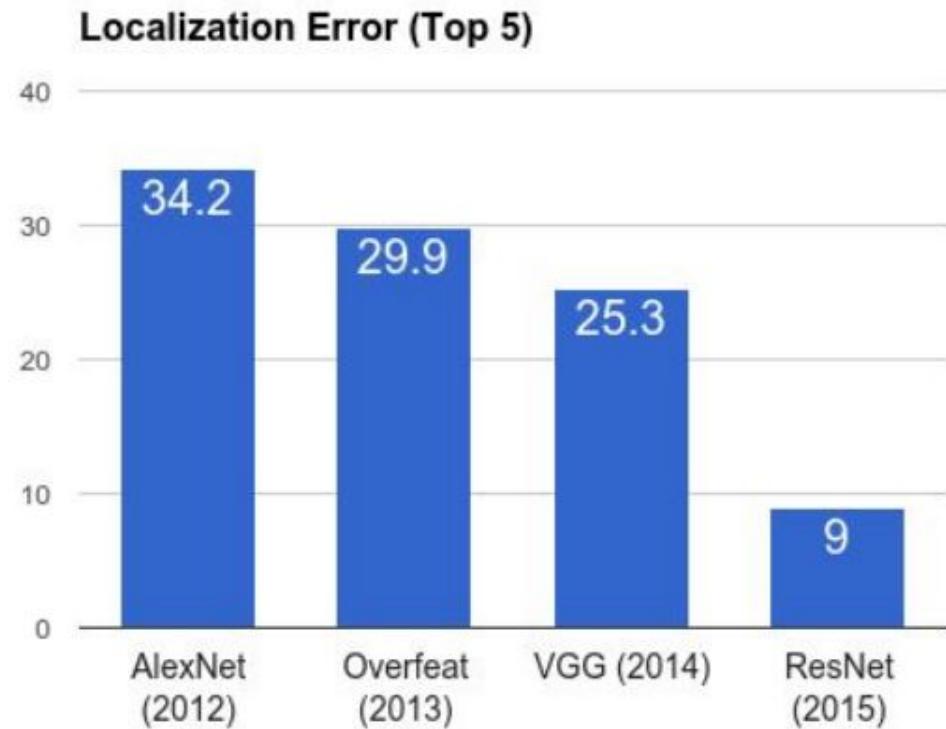
Imagenet results (Localization)

AlexNet: Localization method not published

Overfeat: Multiscale convolutional regression with box merging

VGG: Same as Overfeat, but fewer scales and locations; simpler method, gains all due to deeper features

ResNet: Different localization method (RPN) and much deeper features



Can we do detection as regression?

- Four corners for every object, many numbers to regress.
- No idea on how many objects to expect.



Can we do detection as classification?

- How many boxes should we check? All of them?
- At what scale? Some dogs are smaller (or far away)



Can we do detection as regression?

- Four corners for every object, many numbers to regress.
 - No idea on how many objects to expect.



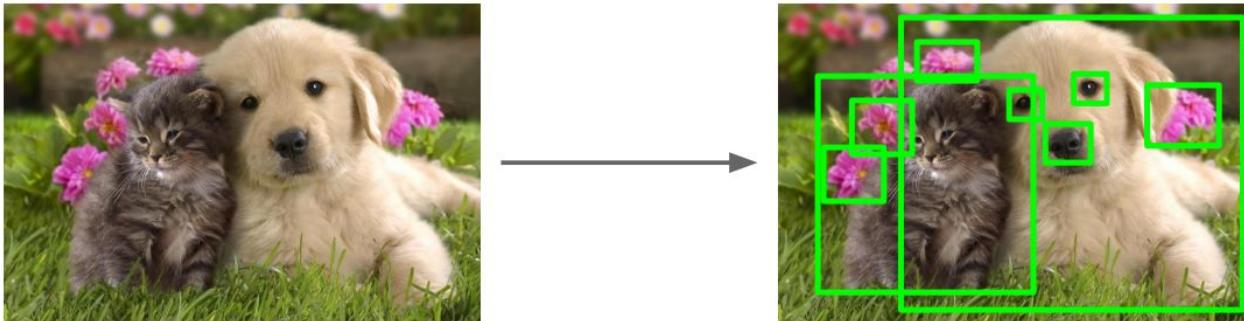
Can we do detection only as classification?

- Solution:**

 - How many boxes should we check? All of them?
At what scale? Some dogs are smaller (or far away)



Better solution - Region proposals

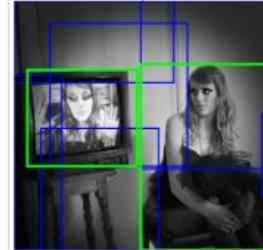
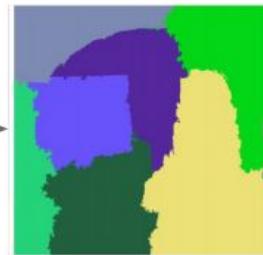
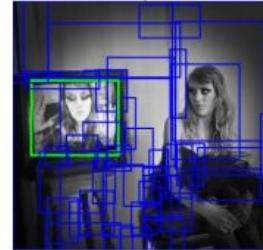
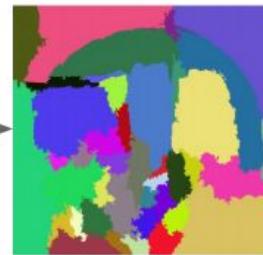
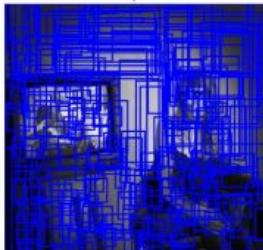


- Find 'regions-of-interest' that are likely to contain some objects
- A simple detector which just says there is something of interest and not necessarily about the class
- These region proposals could be just blob of pixels or 'super-pixels'.
- There are lot of techniques, like BING, CPMC, EdgeBoxes, Geodesic, MCG etc

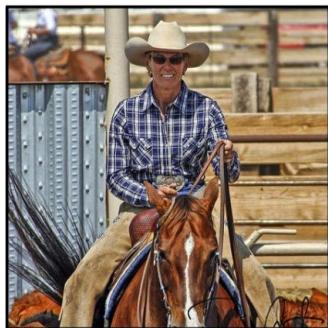
Region Proposals - Selective Search

Bottom-up segmentation, merging regions at multiple scales

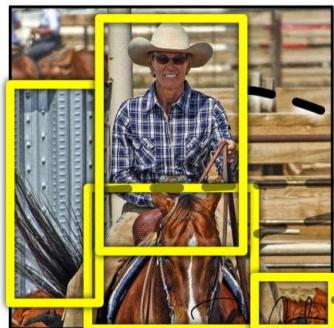
Convert
regions
to boxes



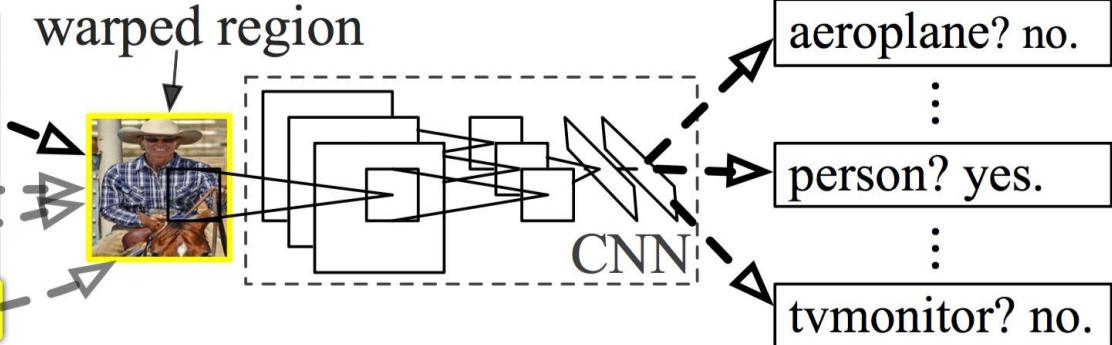
Region based CNN (R-CNN)



1. Input image



2. Extract region proposals (~2k)

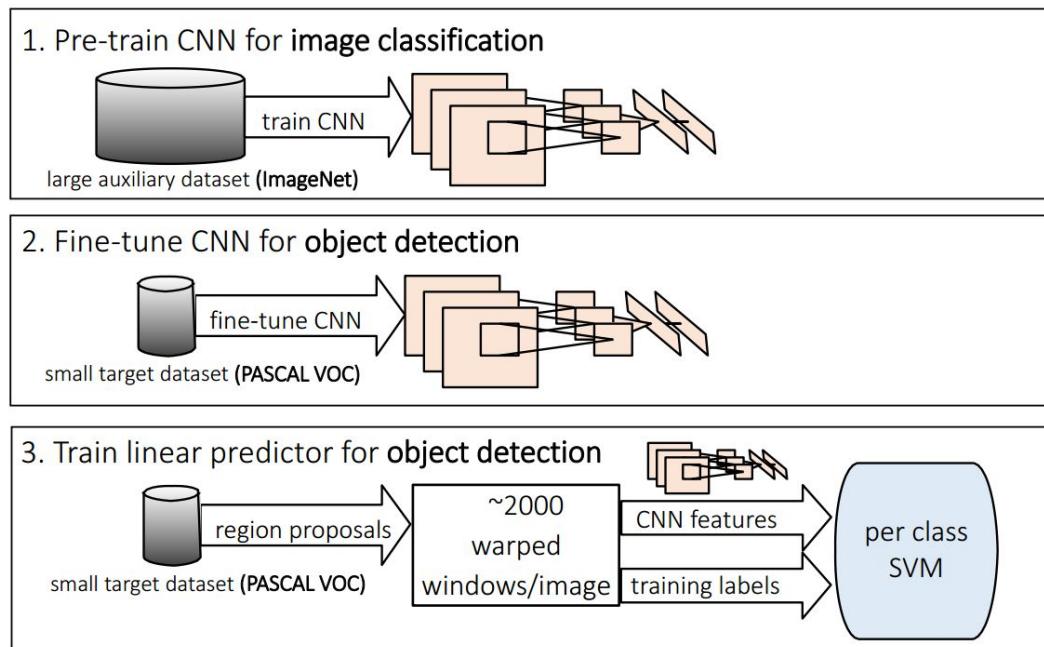


3. Compute CNN features

4. Classify regions

R-CNN Training

1. [offline] ↗ Pre-train a ConvNet for ImageNet classification
2. ↗ Fine-tune M' for object detection (softmax classifier + log loss) [add background class]
3. ← Cache feature vectors to disk using M'
4. Train *post hoc* linear SVMs on F (hinge loss) [binary classifier]
5. Train *post hoc* linear bounding-box regressors on F (squared loss)



R-CNN Results/Analysis

mAP - mean Average Precision

- Average of Average Precision for each class.
- True positive if IoU with ground-truth box > 0.5.
- AP is area under the Precision-Recall curve.

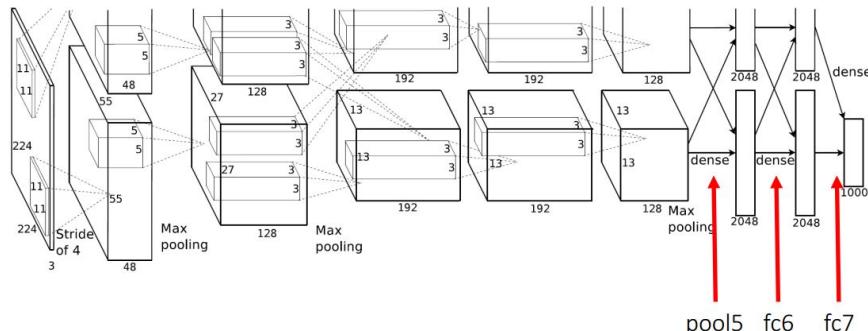
1. Very slow (upto 20 seconds per image on K40 GPU).
2. Needs to run full CNN pass on each proposed region.
3. Not end-to-end training.
4. CNN features are not updated with results of training SVM
5. Better results than any other system

R-CNN Results/Analysis

mAP - mean Average Precision

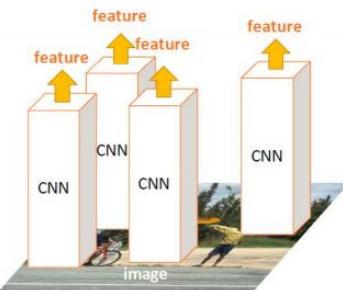
- Average of Average Precision for each class.
 - True positive if IoU with ground-truth box > 0.5.
 - AP is area under the Precision-Recall curve.
1. Very slow (upto 20 seconds per image on K40 GPU).
 2. Needs to run full CNN pass on each proposed region.
 3. Not end-to-end training.
 4. CNN features are not updated with results of training SVM
 5. Better results than any other system

		VOC 2007	VOC 2010
reference	DPM v5 (Girshick et al. 2011)	33.7%	29.6%
	UVA sel. search (Uijlings et al. 2012)		35.1%
	Regionlets (Wang et al. 2013)	41.7%	39.7%
pre-trained only	R-CNN pool ₅	44.2%	
	R-CNN fc ₆	46.2%	
	R-CNN fc ₇	44.7%	
fine-tuned	R-CNN pool ₅	47.3%	
	R-CNN fc ₆	53.1%	
	R-CNN fc ₇	54.2%	50.2%
	R-CNN fc ₇ (Bounding Box regression)	58.5%	53.7%

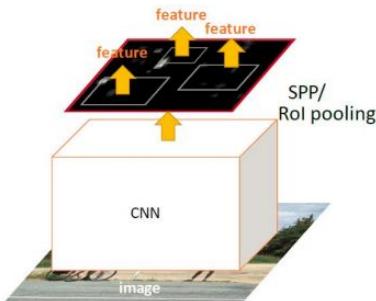


Fast R-CNN

“Slow” R-CNN

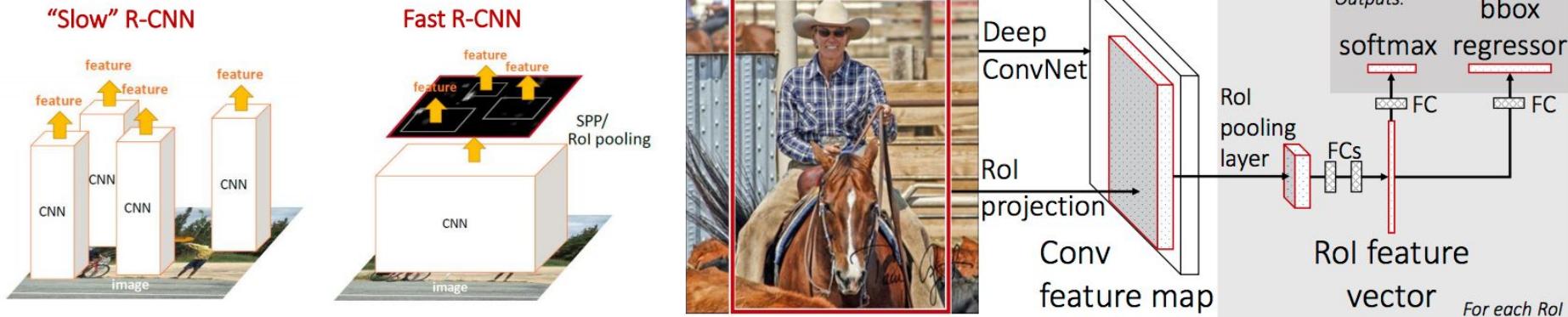


Fast R-CNN



- Share computation of convolutional layers between ‘regions of proposal’

Fast R-CNN

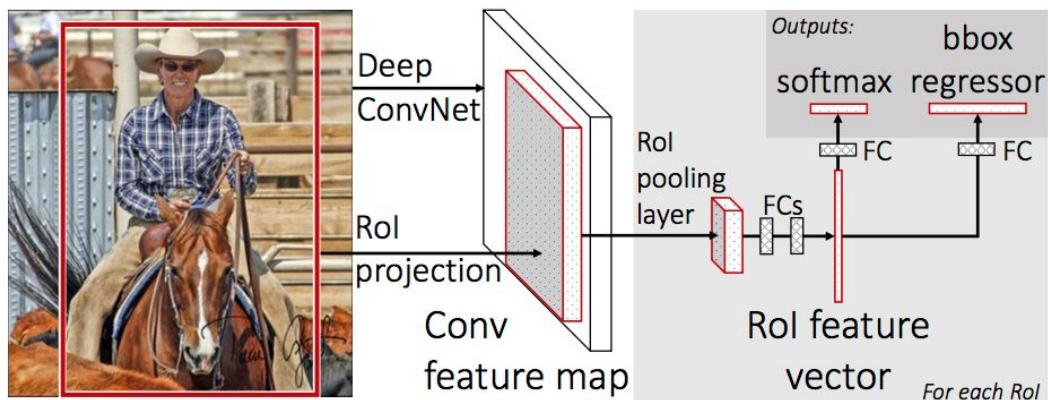
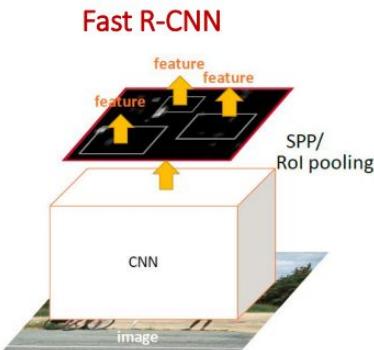
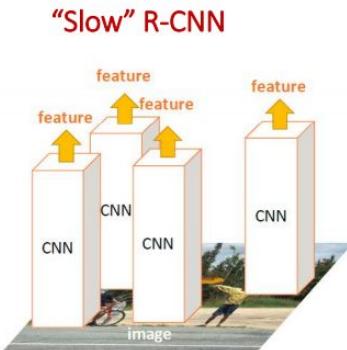


- Share computation of convolutional layers between 'regions of proposal'
- Train the whole system end-to-end
 - no more post-hoc classifiers
 - No caching features to disk

Fast R-CNN

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

True box coordinates
 Predicted box coordinates
 ↑
 True class scores
 Predicted class scores
 ↑ Log loss
 ↑ Smooth L1 loss



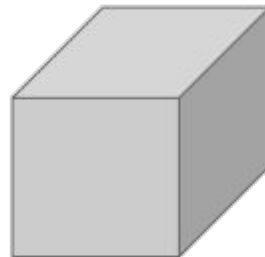
- Share computation of convolutional layers between ‘regions of proposal’
- Train the whole system end-to-end
 - no more post-hoc classifiers
 - No caching features to disk
- Positive samples are those with IoU overlap with ground-truth bounding box > 0.5.
- Negative samples are in interval [0.1,0.5]
- 25%/75% ratio of Positive to Negative samples

Fast R-CNN: Region of Interest Pooling



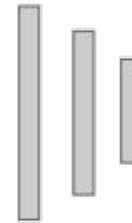
Hi-res input image:
 $3 \times 800 \times 600$
with region
proposal

Convolution
and Pooling



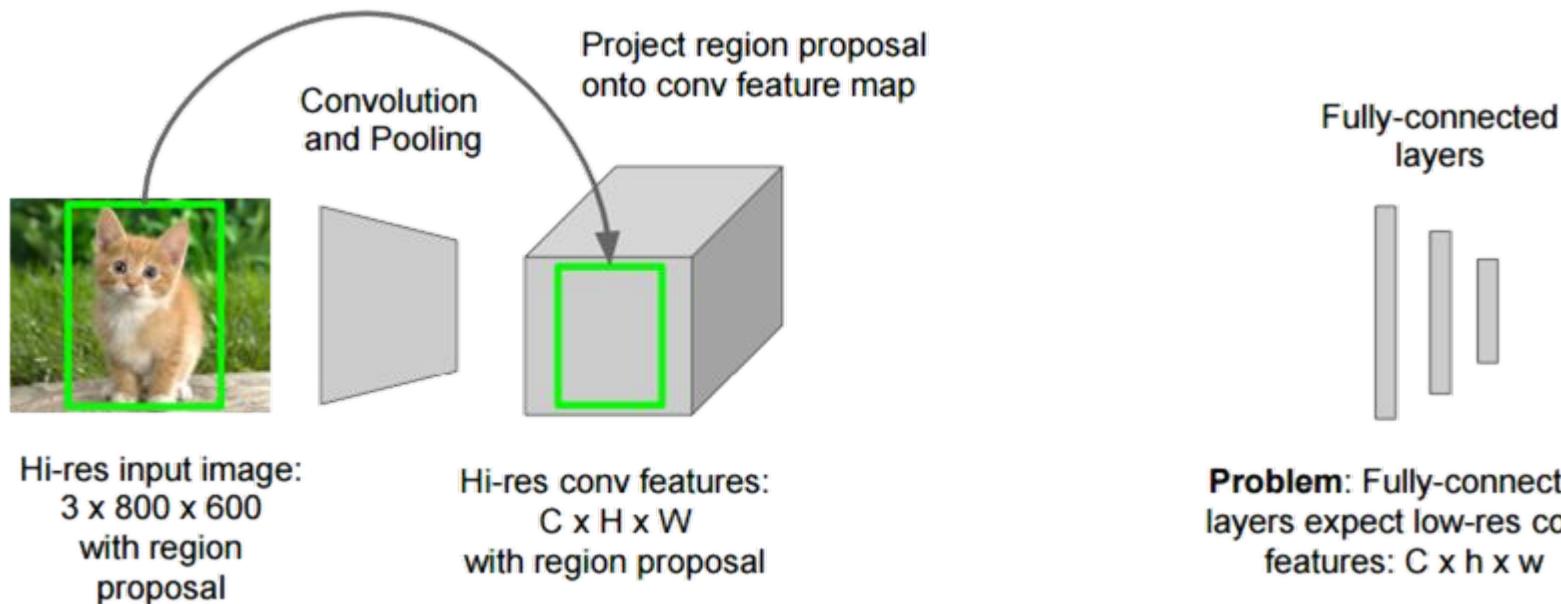
Hi-res conv features:
 $C \times H \times W$
with region proposal

Fully-connected
layers

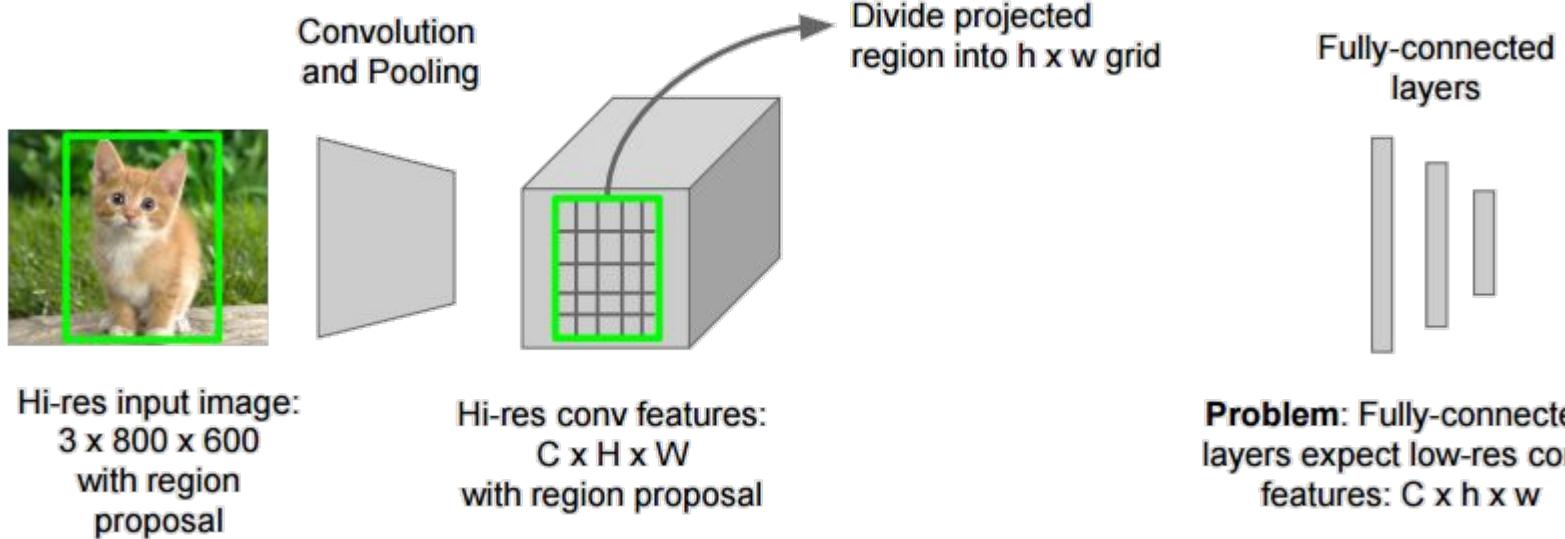


Problem: Fully-connected
layers expect low-res conv
features: $C \times h \times w$

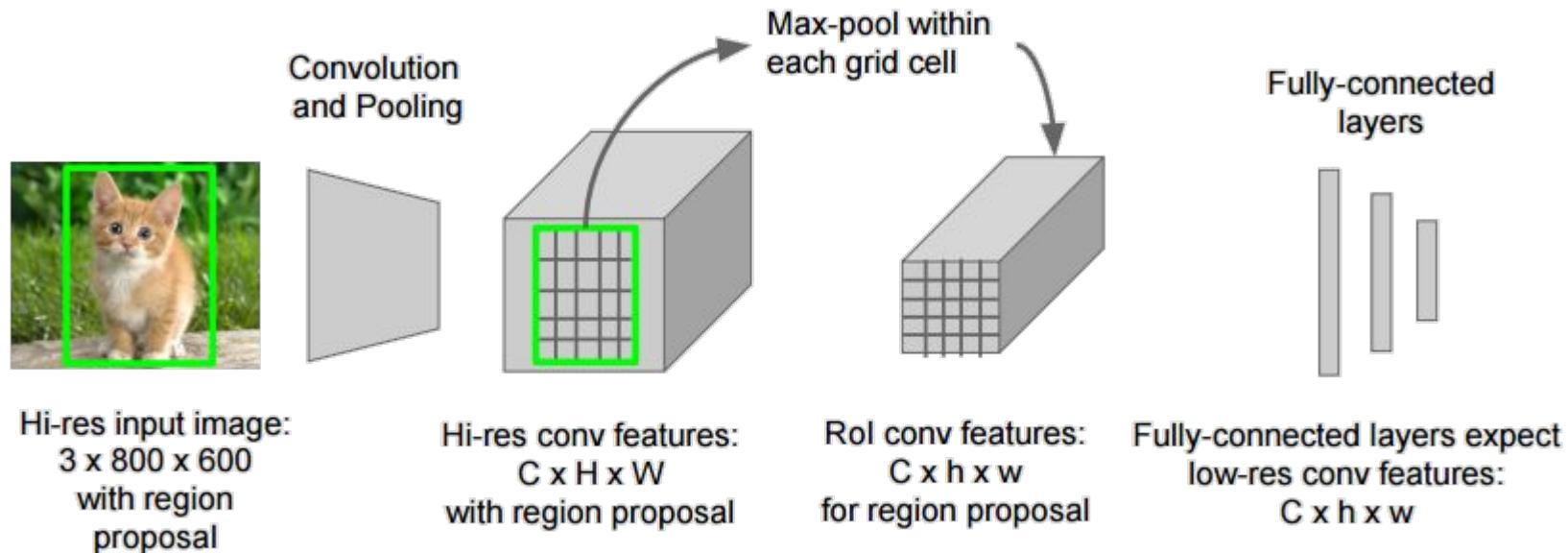
Fast R-CNN: Region of Interest Pooling



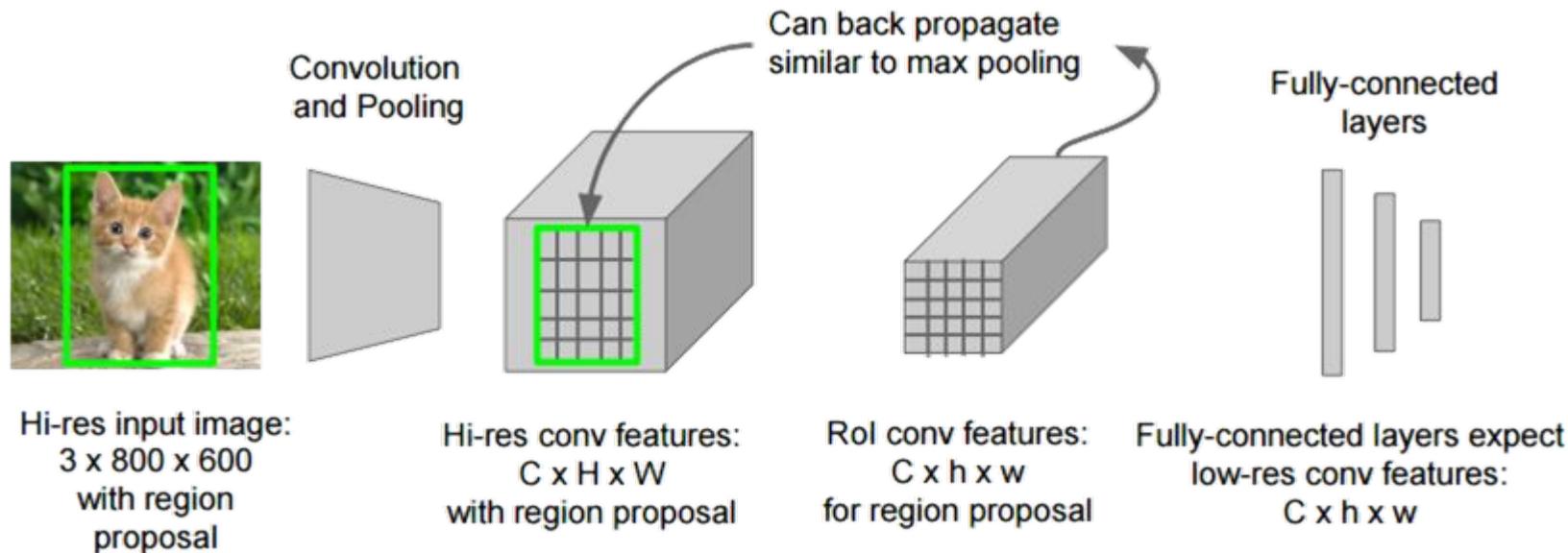
Fast R-CNN: Region of Interest Pooling



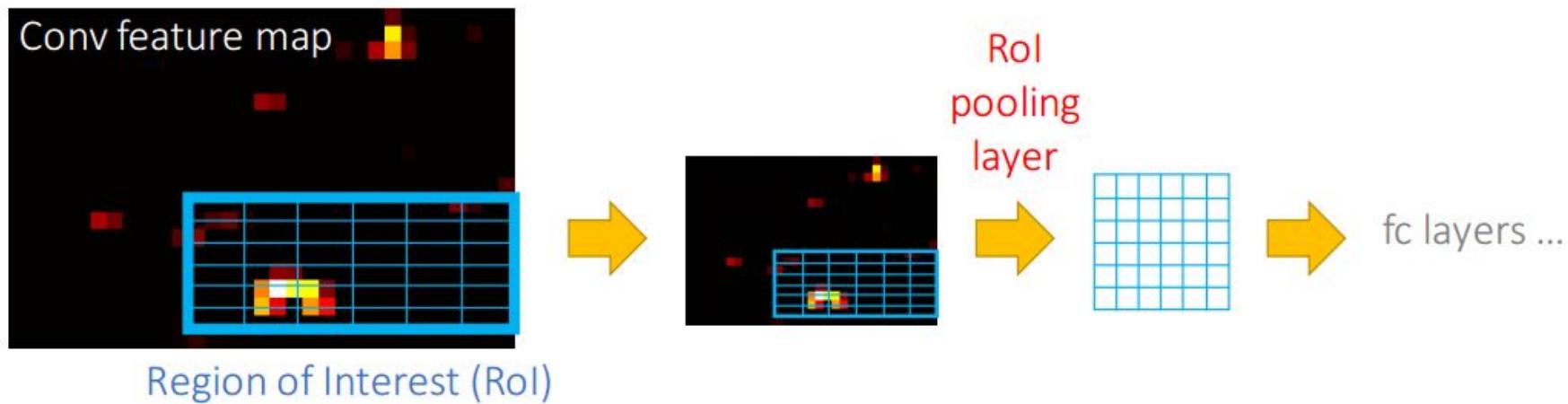
Fast R-CNN: Region of Interest Pooling



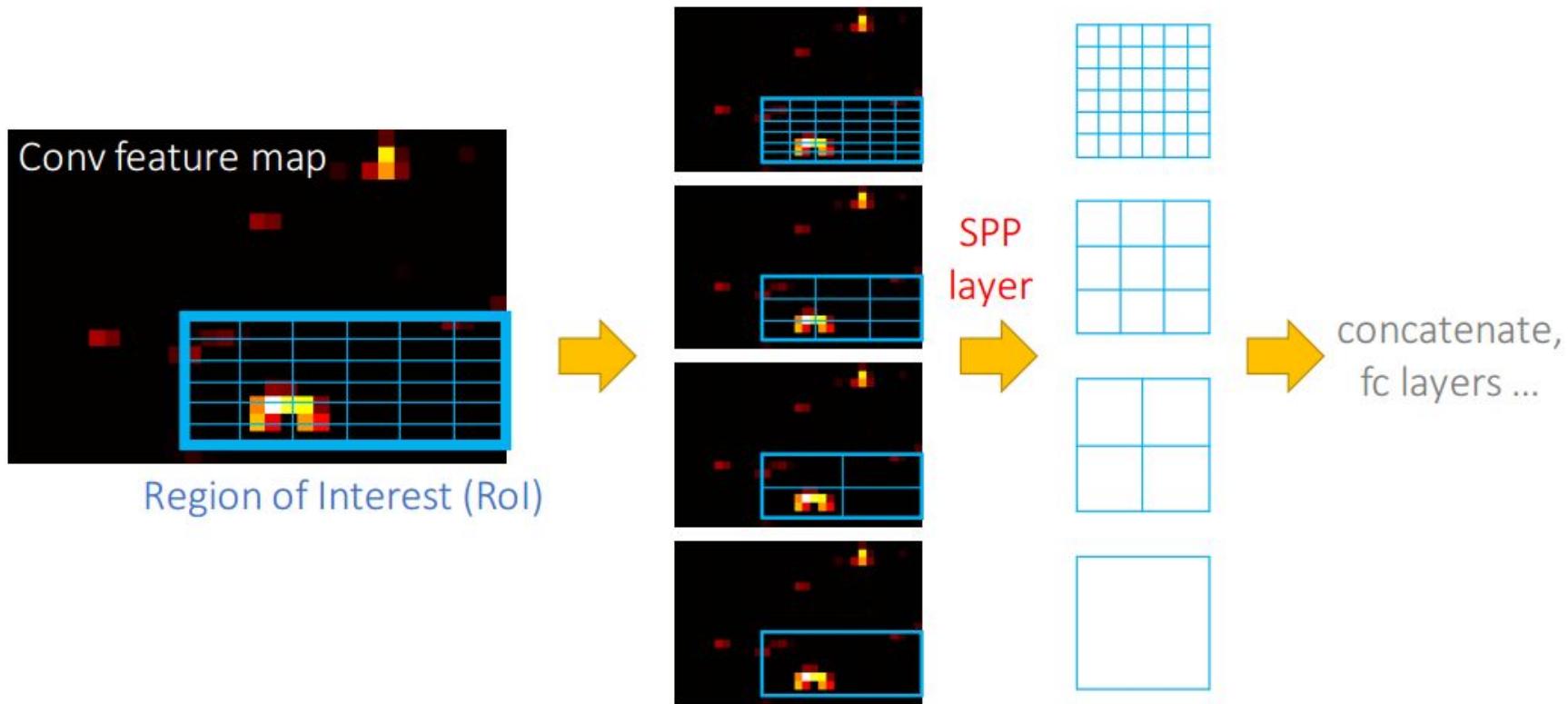
Fast R-CNN: Region of Interest Pooling



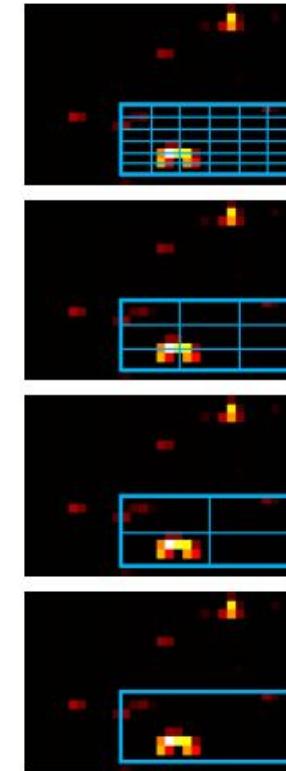
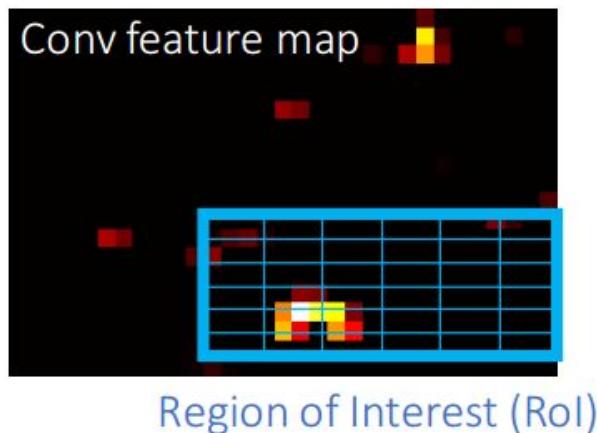
Fast R-CNN: Region of Interest Pooling



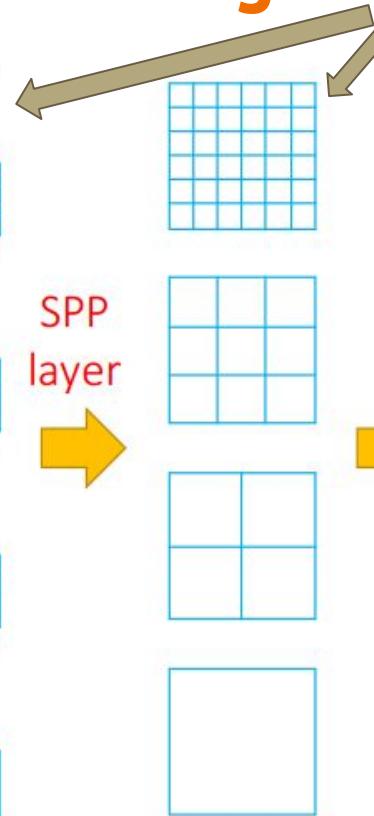
Fast R-CNN: Spatial Pyramid Pooling



Fast R-CNN: Spatial Pyramid Pooling



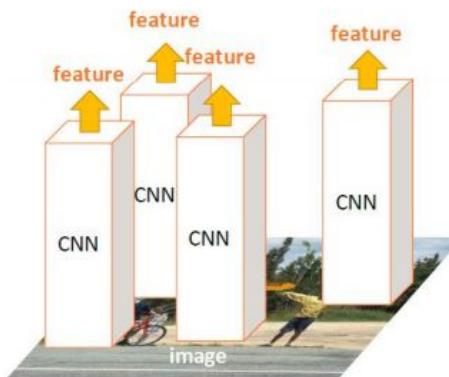
SPP
layer



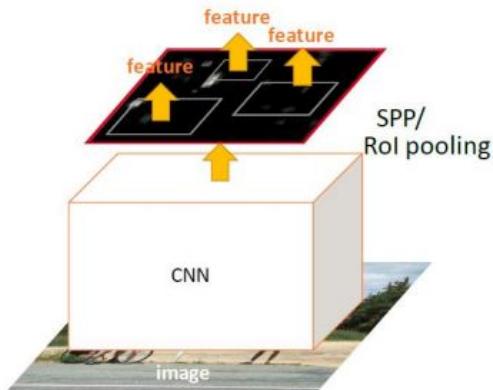
ROI Pooling is a special case
of SPP pooling

Faster R-CNN

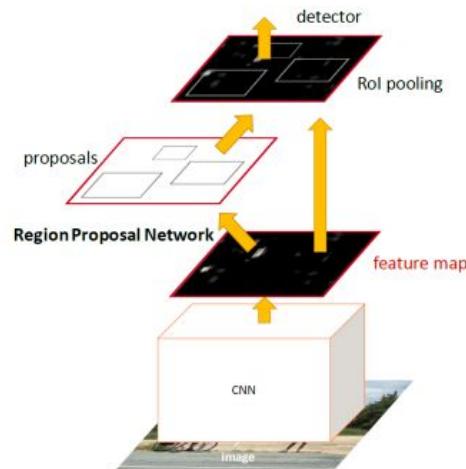
“Slow” R-CNN



Fast R-CNN



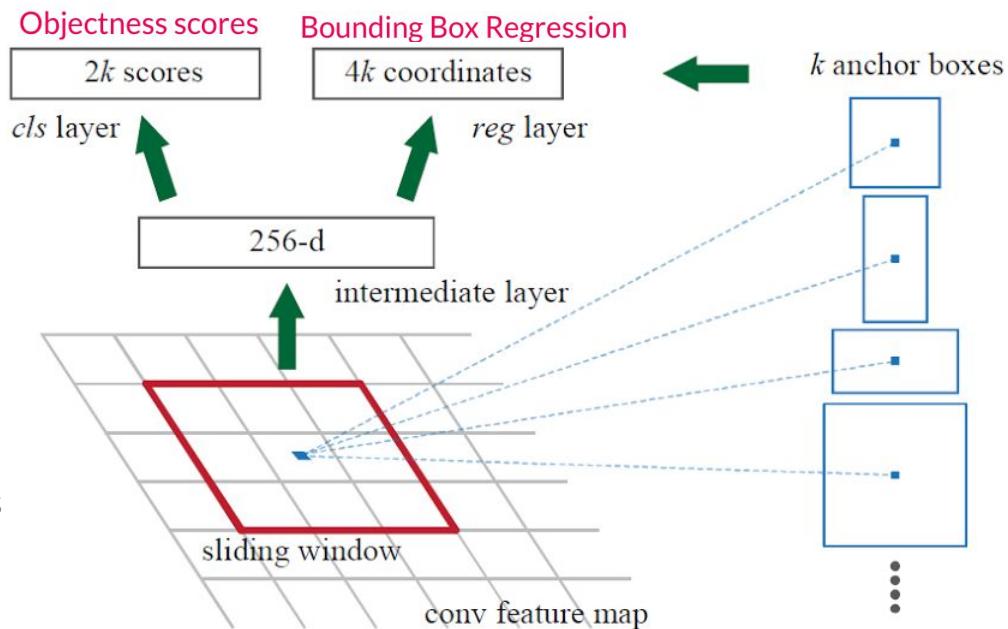
Faster R-CNN



- In R-CNN and Fast R-CNN, bottleneck is ‘region proposal’ (selective search, CPMC, MCG).
- Replace external proposals with Region Proposal Network (RPN), trained with model directly.
- After RPN use ROI Pooling like Fast R-CNN

Region Proposal Network

- Slide a small window (anchor boxes) on the feature map.
- Have multiple such (k) anchor boxes.
- Build small network for
 - binary classification of object in an anchor. (笑脸)
 - regression of bounding box (anchor->proposal)
- Position of sliding window provides localization information with reference to the image.
- Binary classification gives probability that each anchor shows an object.
- Regression score gives offset from anchor boxes
- Anchors are selected for different scales and different aspect ratio, generally 3 each, total 9.



Faster R-CNN: RPN

i = anchor index in minibatch

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Annotations:

- Upward blue arrows point to:
 - Predicted probability of being an object for anchor i
 - Coordinates of the predicted bounding box for anchor i
- Purple arrow labeled "Log loss" points to $L_{cls}(p_i, p_i^*)$.
- Red arrow labeled "Ground truth objectness label" points to p_i^* .
- Red circle labeled λ is placed near the term $\lambda \frac{1}{N_{reg}}$.
- Upward red arrow labeled "True box coordinates" points to t_i .
- Purple arrow labeled "Smooth L1 loss" points to $L_{reg}(t_i, t_i^*)$.

N_{cls} = Number of anchors in minibatch (~ 256)

N_{reg} = Number of anchor locations (~ 2400)

In practice $\lambda = 10$, so that both terms are roughly equally balanced

Anchor is labelled positive if:

- anchor is the one with highest IoU overlap with ground truth box
- anchor has IoU overlap with a ground-truth box higher than 0.7

Anchor is labelled negative if IoU lower than 0.3 for all ground-truth boxes.

50%/50% ratio of positive to negative samples

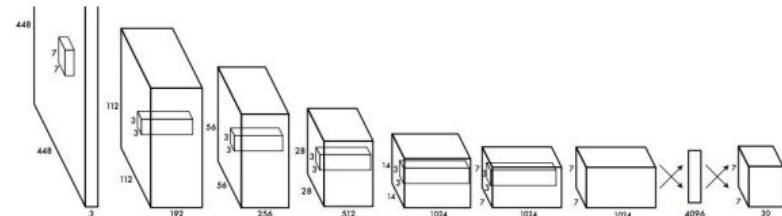
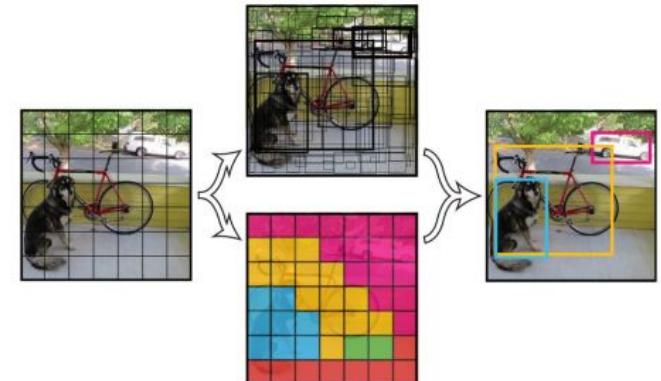
Faster R-CNN: Comparison accuracy/speed

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speedup (+proposal)	1x	25x	250x
mAP (VOC 2007)	66.0	66.9	66.9

You Only Look Once (YOLO)

- Divide image into $S \times S$ grid
- Within each grid cell predict:
 - B Boxes (4 coordinates + confidence)
 - Class scores: C scores
- Regression from image to $7 \times 7 \times (5 * B + C)$ tensor
- Direct prediction using a CNN
- Faster than Faster R-CNN but not as good
- Struggles with small objects and different aspect ratio

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

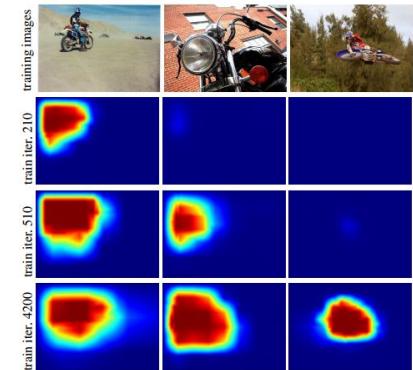


"I remember, when I was reading this paper for the first time, I had a really hard time figuring out what was going on." - Andrew Ng

Can you do this without spatial ground-truth?

Weakly supervised object localization

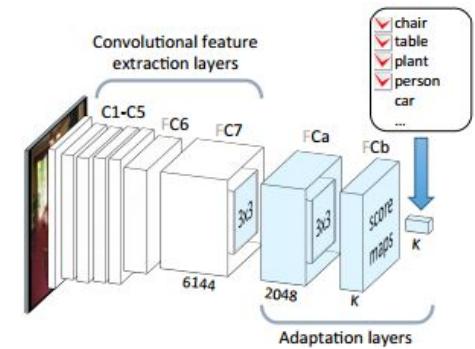
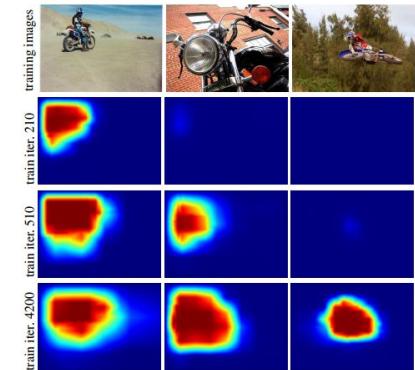
- If you look at the pixels which give maximum activation for a given class, it is no surprise that pixels corresponding to actual object have higher activation.
- But how do you control of actual object pixels? Remember the ‘tank problem’?
- Also this is not reliable in a cluttered scene containing many objects, and partially occluded objects or cropped objects.
- Oquab et al,
 - removed the Fully Connected layer at the end of a CNN model to prevent losing any more localization information and,
 - Added global max-pooling (GMP) to output layer to find highest scoring object position in the image.



Can you do this without spatial ground-truth?

Weakly supervised object localization

- If you look at the pixels which give maximum activation for a given class, it is no surprise that pixels corresponding to actual object have higher activation.
- But how do you control of actual object pixels? Remember the ‘tank problem’?
- Also this is not reliable in a cluttered scene containing many objects, and partially occluded objects or cropped objects.
- Oquab et al,
 - removed the Fully Connected layer at the end of a CNN model to prevent losing any more localization information and,
 - Added global max-pooling (GMP) to output layer to find highest scoring object position in the image.
- GMP enforces correspondence between feature maps and categories, and thus they can be interpreted as ‘confidence maps’.
- Best part - it is parameter free.



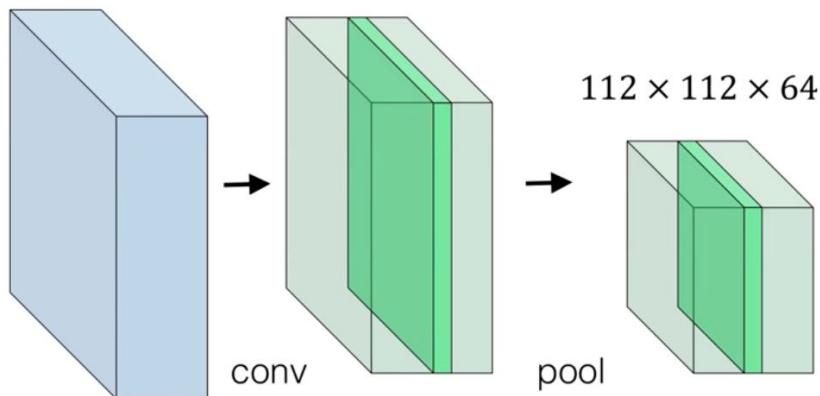
Max Pooling

vs

Global Max Pooling

$224 \times 224 \times 3$

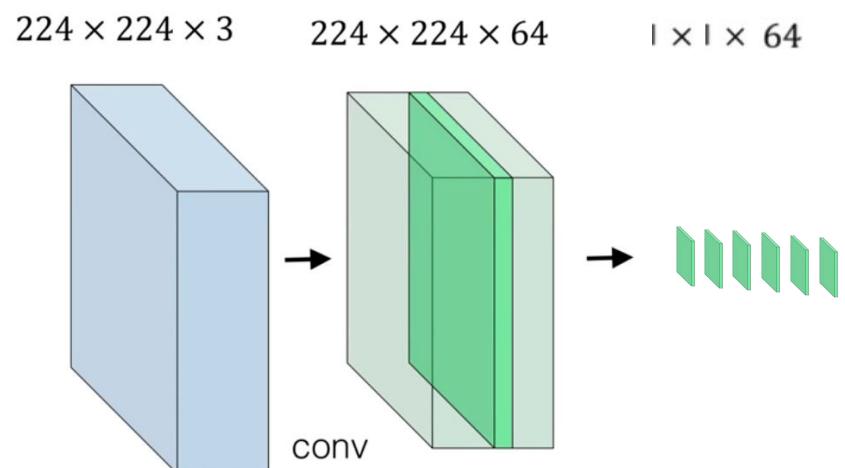
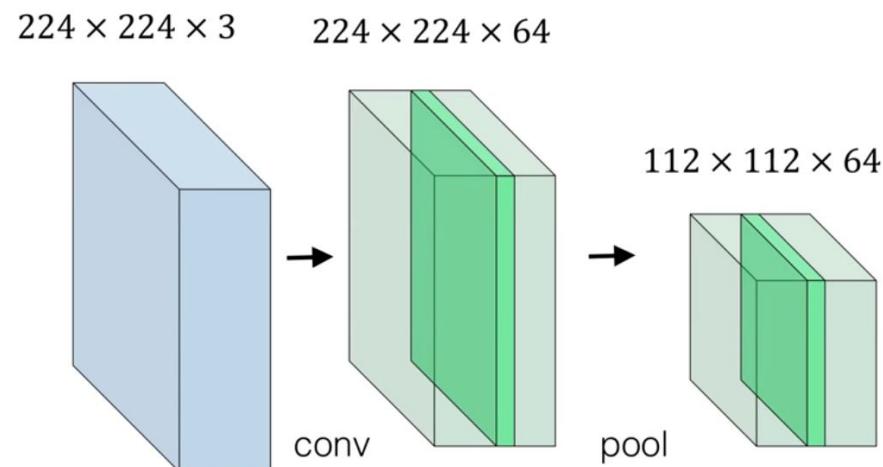
$224 \times 224 \times 64$



Max Pooling

vs

Global Max Pooling



Global Max Pooling - Results



Object-level sup.	mAP
A.NUS-SCM [51]	82.2
B.OQUAB [37]	82.8

Image-level sup.	mAP
C.Z&F [60]	79.0
D.CHATFIELD [6]	83.2
E.NUS-HCP [56]	84.2
F.FULL IMAGES	78.7
G.WEAK SUP	86.3

- Localization is only about finding 'center' point, cannot draw bounding box
- Helps improve the classification score
- Masked pooling is where you minimize the object score outside the masked area (requires location GT)

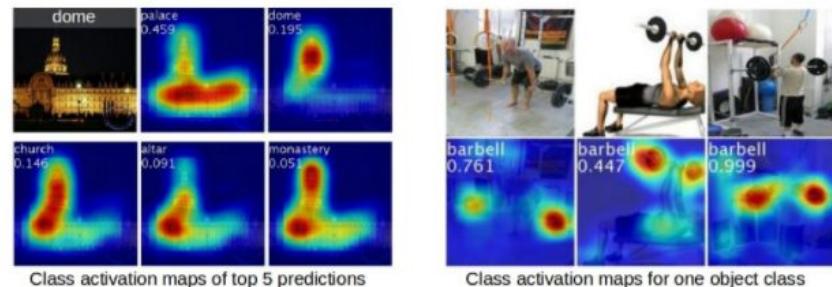
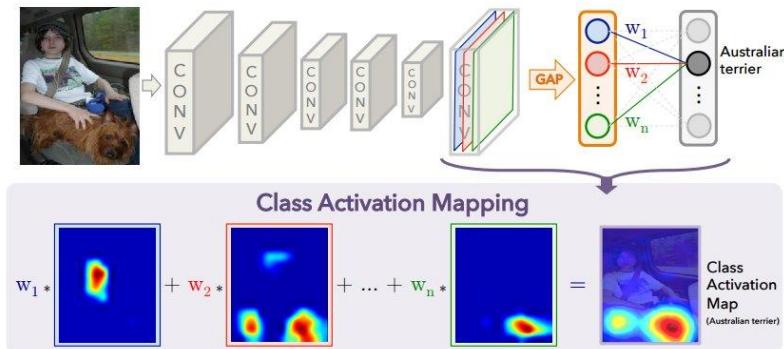
Setup	Classification		Location Prediction	
	VOC	COCO	VOC	COCO
H.FULL IMAGES	76.0	51.0	-	-
I.MASKED POOL	82.3	62.1	72.3	42.9
J.WEAK SUP	81.8	62.8	74.5	41.2
K.CENTER PRED.	-	-	50.9	19.1
L.RCNN*	79.2	-	74.8	-

Class activation map

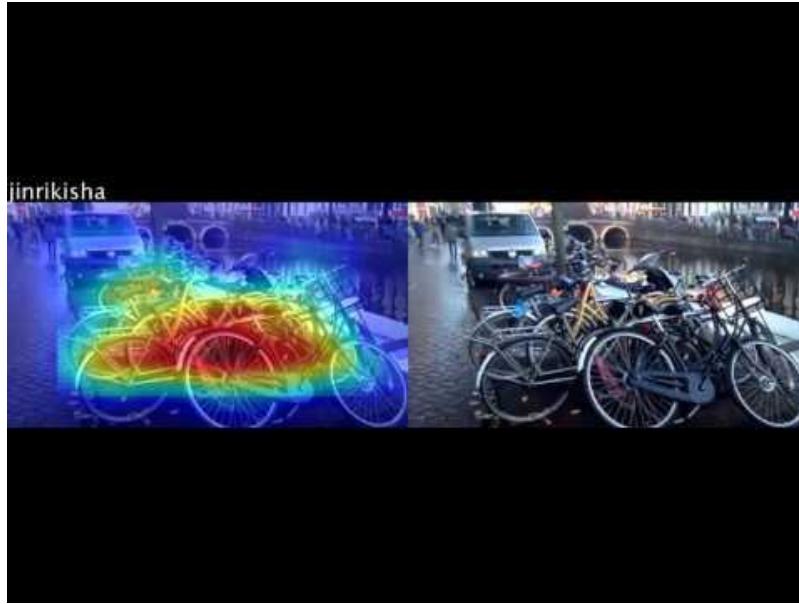
- Replace Global Max Pooling (GMP) with Global Average Pooling (GAP).
- The use of average pooling encourages the network to identify the complete extent of the object
- Intuition behind this is that the loss for average pooling benefits when the network identifies all discriminative regions of an object as compared to max pooling
- Class activation map is when you take the output of GAP and add a ‘fully connected layer’ (without nonlinearity). Use the weights of this layer to weight the activation at unit k , to generate a spatial map.

$$M_c(x, y) = \sum_{d \in D} w_d^c f_d(x, y) \quad \text{where } w_d^c \text{ is the learned weight of class } c \text{ for feature map } d$$

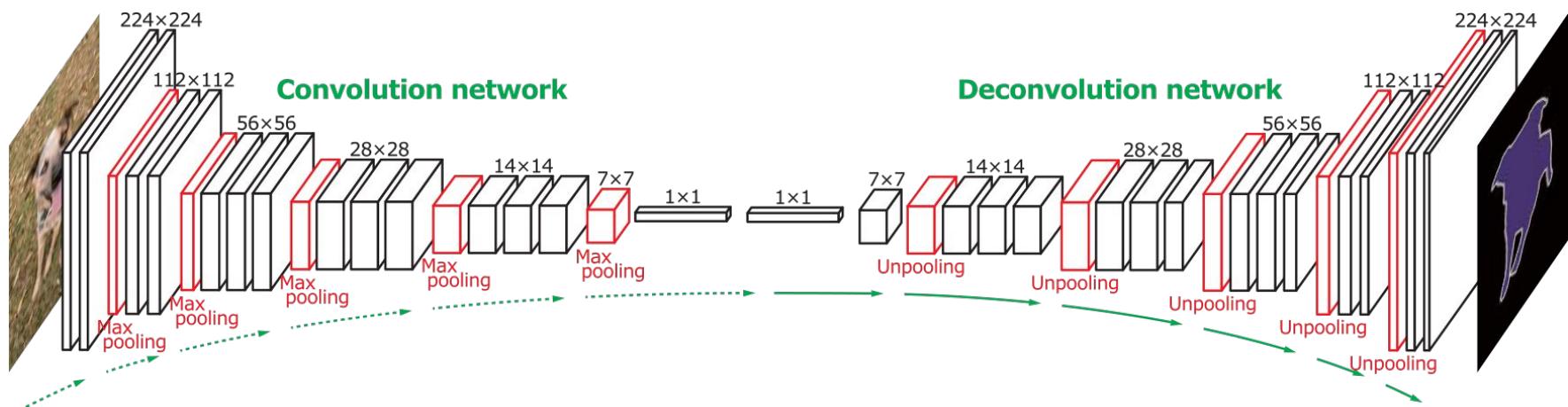
- For training, you maximize the cross entropy between predicted class and true distribution $P(c) = \frac{\exp(\sum_{xy} M_c(x, y))}{\sum_c \exp(\sum_{xy} M_c(x, y))}$
- This means it can “highlight” only one salient class in a given image. Gives more than center but not bounding box.



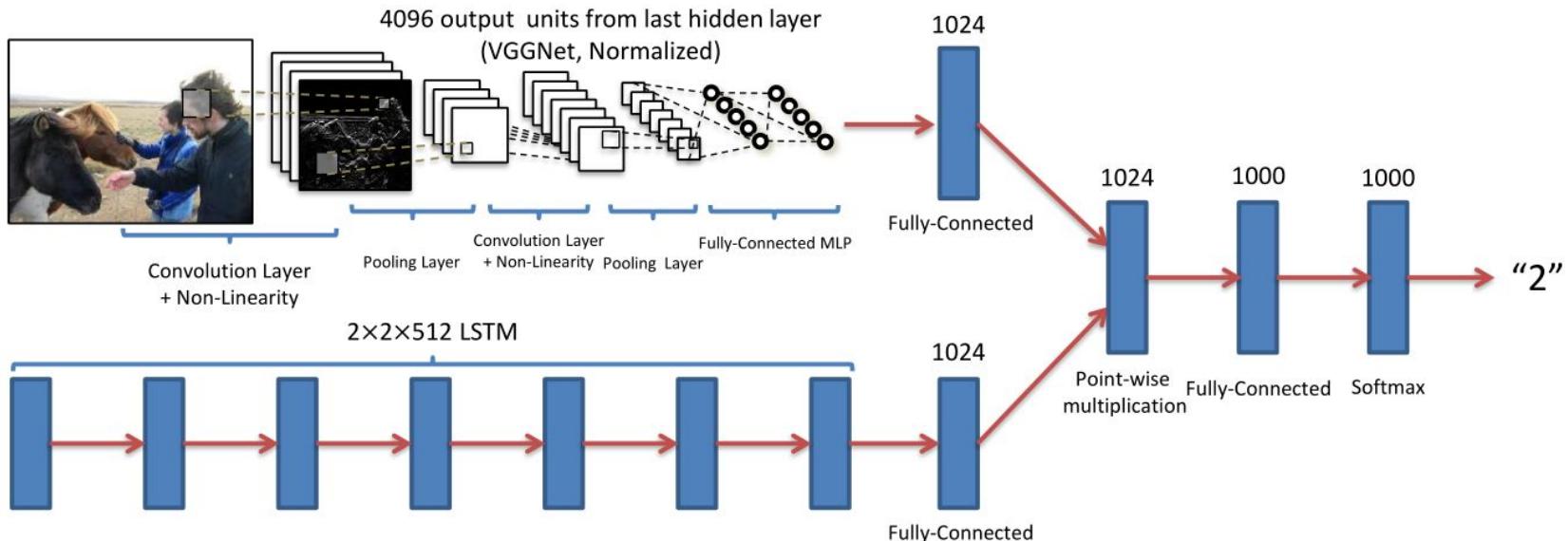
Class activation map



Semantic Segmentation



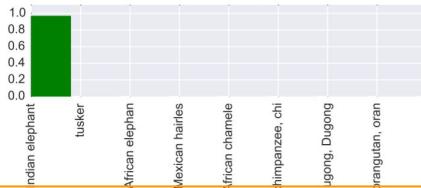
Visual Question Answering



"How many horses are in this image?"

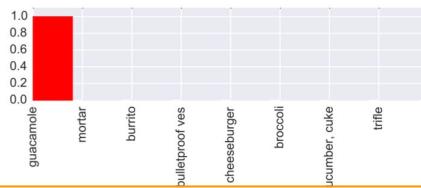
Adversarial Attack

Original Image



Predicted: Indian Elephant (99.7%)

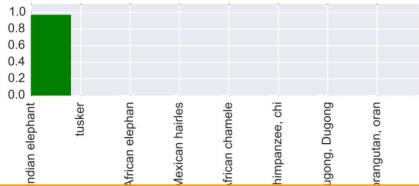
Adversarial Image



Predicted: Guacamole (99.9%)

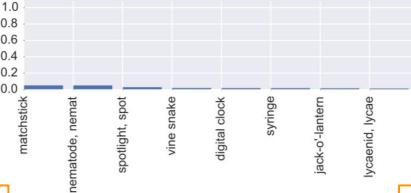
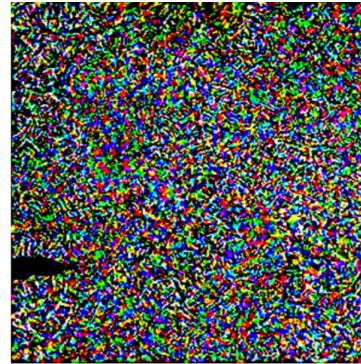
Adversarial Attack

Original Image



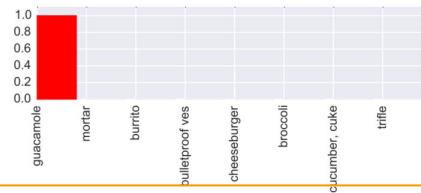
$+ \epsilon$

Perturbations



=

Adversarial Image



Predicted: Indian Elephant (99.7%)

Predicted: Guacamole (99.9%)

- With cleverly designed additive noise the classification of the image can be changed
- It is easy to find noise which makes the classifier predict any given class - Targeted attack

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
- but recent advances such as ResNet/GoogLeNet challenge this paradigm

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
- but recent advances such as ResNet/GoogLeNet challenge this paradigm

Credits

1. [CS231n](#) Convnets
2. [Chris Colah](#)'s awesome blog
3. [Chris Burger](#) - Style transfer images
4. [Convolutional Neural Networks](#) - Nervana Systems
5. [DeepVis](#) - Jason Yosinski
6. [Fooling CNNs](#) - Anh Nguyen
7. [More demos](#) - Yann LeCun

Any questions?

You can find me at
iamaaditya.github.io
aprakash@brandeis.edu

