Lecture 7 - Even more about ConvNets

DD2424

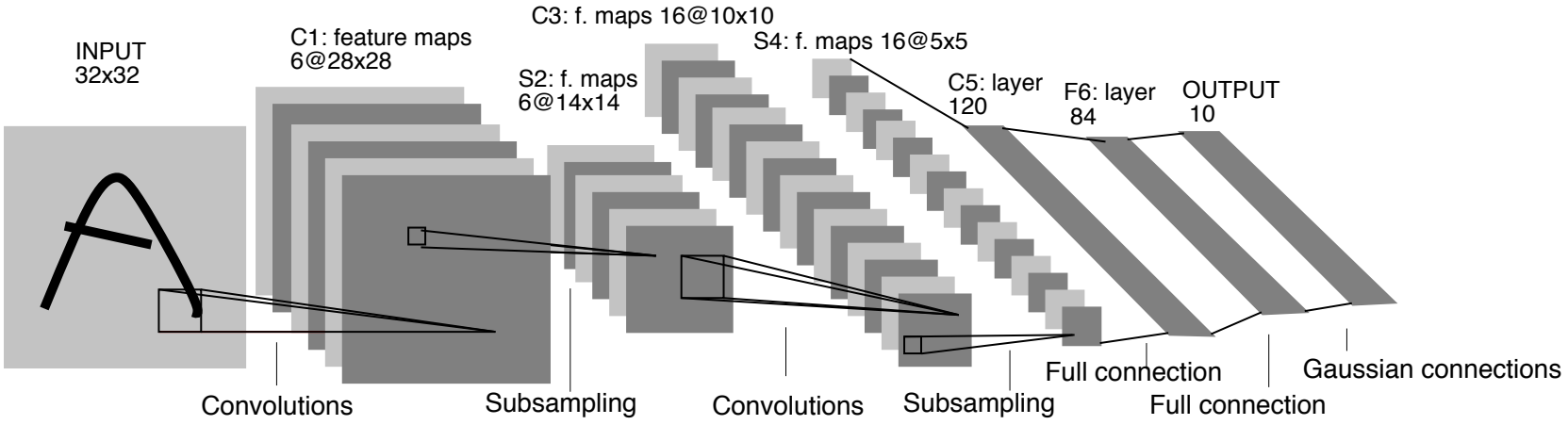
April 2, 2019

Overview of today's lecture

- More on Modern ConvNet architectures
- Practicalities of training & designing ConvNets
  - Stacking convolutional filters.
  - Data augmentation.
  - Batch Normalization for convolutional layers.
  - Transfer learning.
- ConvNets for Semantic segmentation
  - Transposed convolutions

Architecture of Modern ConvNets

LeNet-5 [LeCun et al., 1998]



- Conv filters are 5×5, applied at stride 1
- Pooling layers are 2×2, applied at stride 2
- Architecture is

  [CONV-POOL-TANH-CONV-POOL-TANH-FC-TANH-FC-TANH-FC]

AlexNet [Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

Output size	**Layer type**	**Details**
$227 \times 227 \times 3$	Input	
$55 \times 55 \times 96$	Conv	96 filters, size 11×11 at stride 4, pad 0
$27 \times 27 \times 96$	Max Pool	3×3 regions at stride 2
$27 \times 27 \times 96$	Norm	Normalization layer
$27 \times 27 \times 256$	Conv	256 filters, size 5×5 at stride 1, pad 2
$13 \times 13 \times 256$	Max Pool	3×3 regions at stride 2
$13 \times 13 \times 256$	Norm	Normalization layer
$13 \times 13 \times 384$	Conv	384 filters, size 3×3 at stride 1, pad 1
$13 \times 13 \times 384$	Conv	384 filters, size 3×3 at stride 1, pad 1
$13 \times 13 \times 256$	Conv	256 filters, size 3×3 at stride 1, pad 1
$6 \times 6 \times 256$	Max Pool	3×3 regions at stride 2
4096	Fully connected	4096 neurons
4096	Fully connected	4096 neurons
1000	Fully connected	1000 neurons (class scores)

VGGNet [Simonyan and Zisserman, 2014]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB imag)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	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	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The 6 different architecures of VGG Net. Configuration D produced the best results

- Only filters of size 3×3 in the convolutional layers, stride 1 and pad 1.
- Relatively sparse use of **Max Pooling** with region size 2×2 and stride 2.
- **Improved ILSVRC Performance** top 5 error:

  2013 best 11.2% $\rightarrow$ 7.3% (VGGNet ensemble)

VGGNet architecture [Simonyan and Zisserman, 2014]

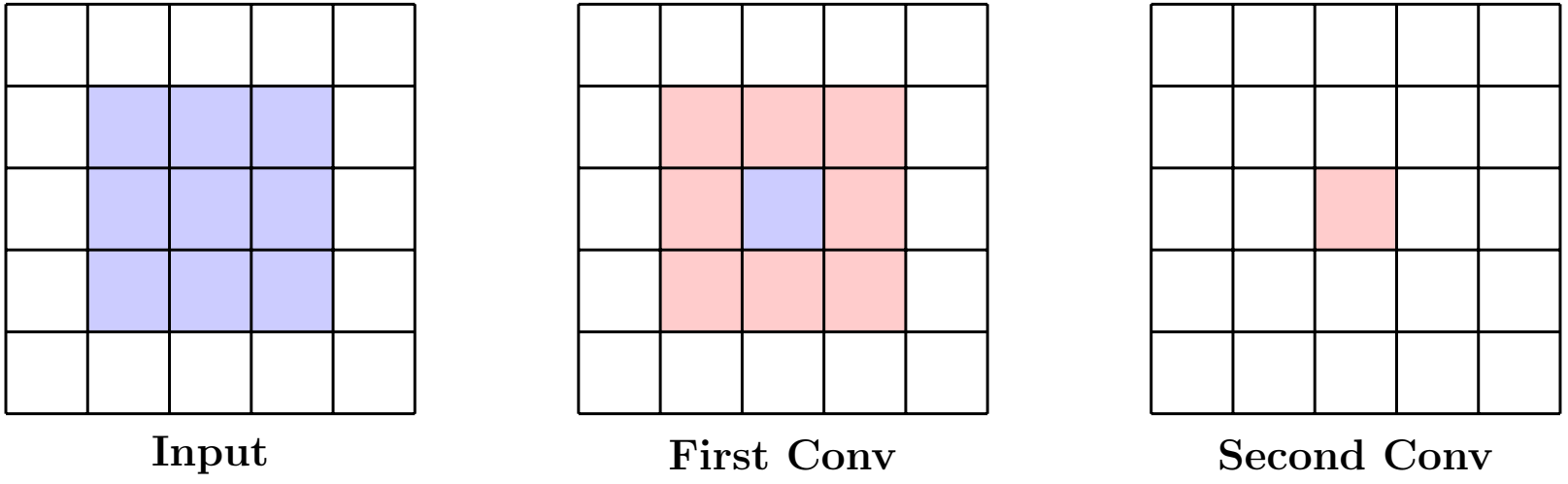
Output size	Layer type	Memory	# parameters
$224 \times 224 \times 3$	Input	224*224*3 = 150K	0
$224 \times 224 \times 64$	Conv	224*224*64 = 3.2M	1,728 = (3*3*3)*64
$224 \times 224 \times 64$	Conv	224*224*64 = 3.2M	36,864 = (3*3*64)*64
$112 \times 112 \times 64$	Max Pool	112*112*64 = 800K	0
$112 \times 112 \times 128$	Conv	112*112*128 = 1.6M	73,728 = (3*3*64)*128
$112 \times 112 \times 128$	Conv	112*112*128 = 1.6M	147,456 = (3*3*128)*128
$56 \times 56 \times 128$	Max Pool	56*56*128 = 400K	0
$56 \times 56 \times 256$	Conv	56*56*256 = 800K	294,912 = (3*3*128)*256
$56 \times 56 \times 256$	Conv	56*56*256 = 800K	589,824 = (3*3*256)*256
$56 \times 56 \times 256$	Conv	56*56*256 = 800K	589,824 = (3*3*256)*256
$28 \times 28 \times 256$	Max Pool	28*28*256 = 200K	0
$28 \times 28 \times 512$	Conv	28*28*512 = 400K	1,179,648 = (3*3*256)*512
$28 \times 28 \times 512$	Conv	28*28*512 = 400K	2,359,296 = (3*3*512)*512
$28 \times 28 \times 512$	Conv	28*28*512 = 400K	2,359,296 = (3*3*512)*512
$14 \times 14 \times 512$	Max Pool	14*14*512 = 100K	0
$14 \times 14 \times 512$	Conv	14*14*512 = 100K	2,359,296 = (3*3*512)*512
$14 \times 14 \times 512$	Conv	14*14*512 = 100K	2,359,296 = (3*3*512)*512
$14 \times 14 \times 512$	Conv	14*14*512 = 100K	2,359,296 = (3*3*512)*512
$7 \times 7 \times 512$	Max Pool	7*7*512 = 25K	0
$1 \times 1 \times 4096$	Fully connected	4096	102,760,448 = 7*7*512*4096
$1 \times 1 \times 4096$	Fully connected	4096	16,777,216 = 4096*4096
$1 \times 1 \times 1000$	Fully connected	1000	4,096,000 = 4096*1000

- Most memory is in early convolutional layers.
- Most parameters in the early fully connected layers.

Why use smaller filters?

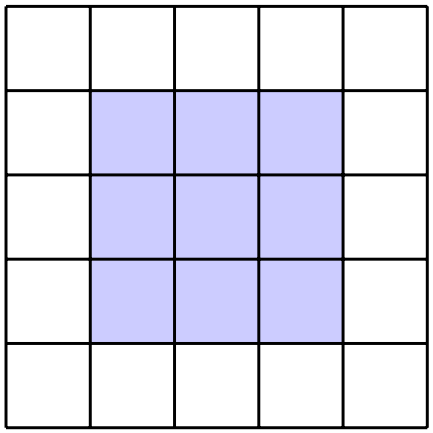
The power of small filters

- Have two consecutive 3×3 conv layers with stride 1.
- Each response *sees* 3×3 region in previous activation map.

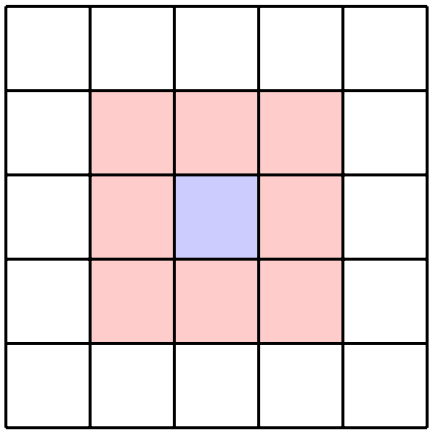


The power of small filters

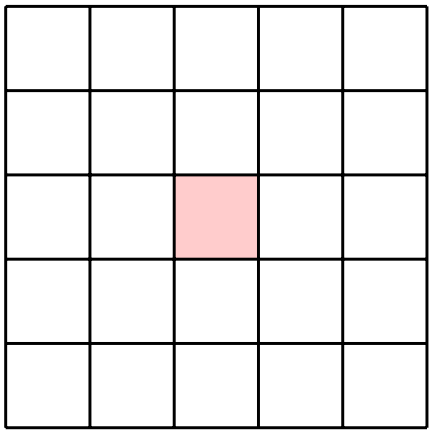
- How big of a region in the input does a response in the second conv layer see?
- **Answer**: 5×5



Input



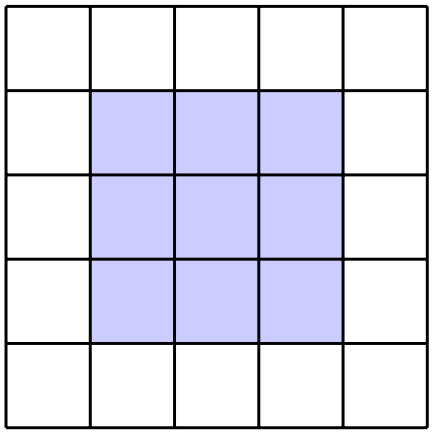
First Conv



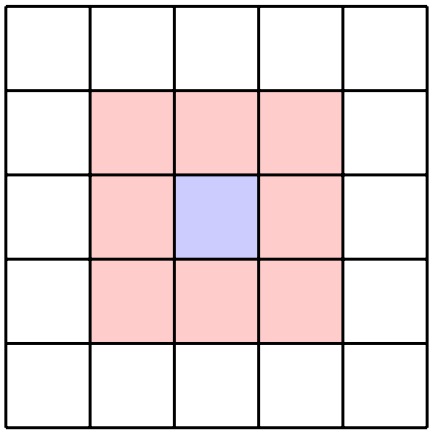
Second Conv

The power of small filters

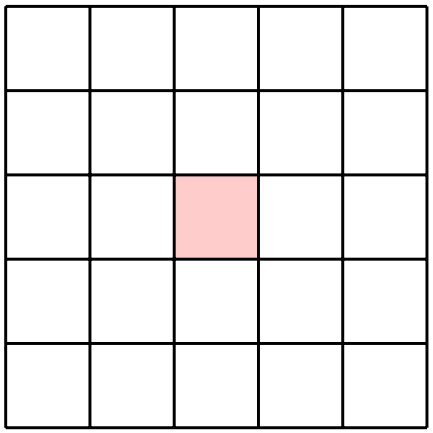
- How big of a region in the input does a response in the second conv layer see?
- **Answer**: 5×5



Input



First Conv



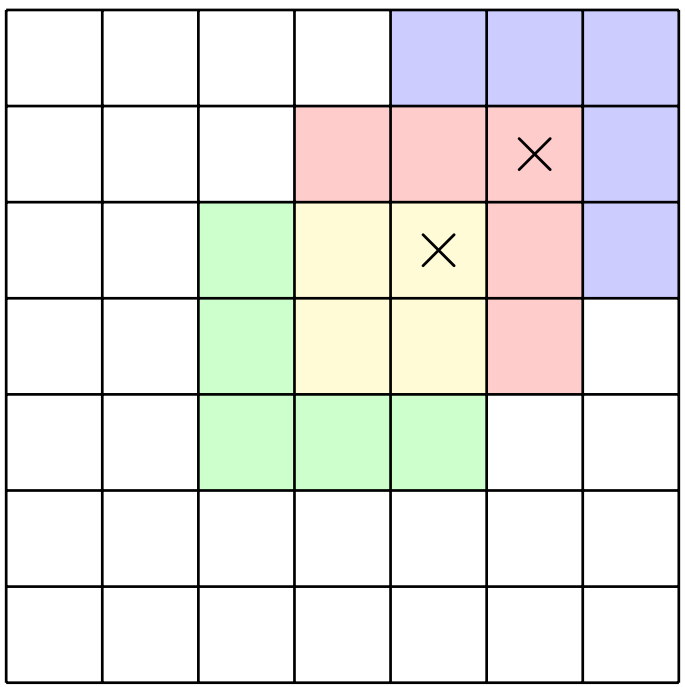
Second Conv

The power of small filters

- We stack three 3×3 conv layers.
- How big is the region in the input layer does that a response in the third layer sees?

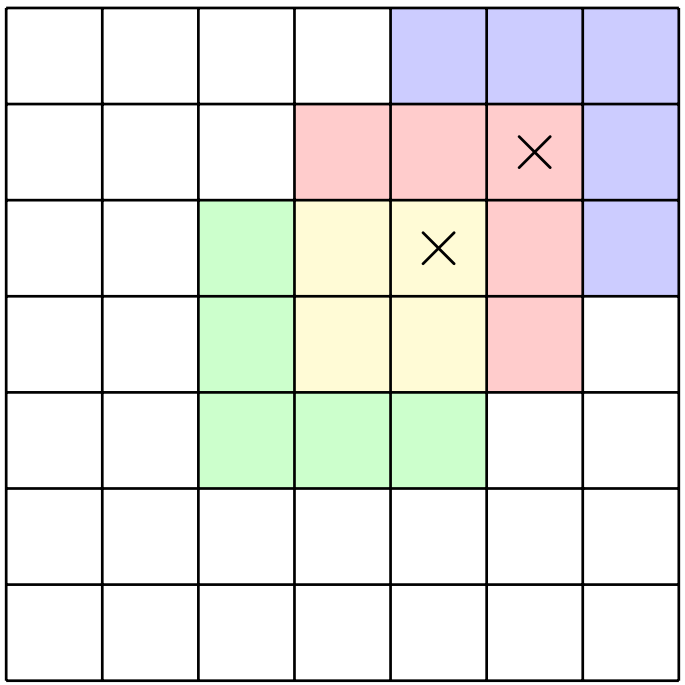
The power of small filters

- We stack three 3×3 conv layers.
- How big is the region in the input layer does that a response in the third layer sees?
- **Answer**: 7×7
- Three stacked 3×3 convolutions give similar representational powers as a single 7×7 convolution.



The power of small filters

- We stack three 3×3 conv layers.
- How big is the region in the input layer does that a response in the third layer sees?
- **Answer**: 7×7
- Three stacked 3×3 convolutions give similar representational powers as a single 7×7 convolution.



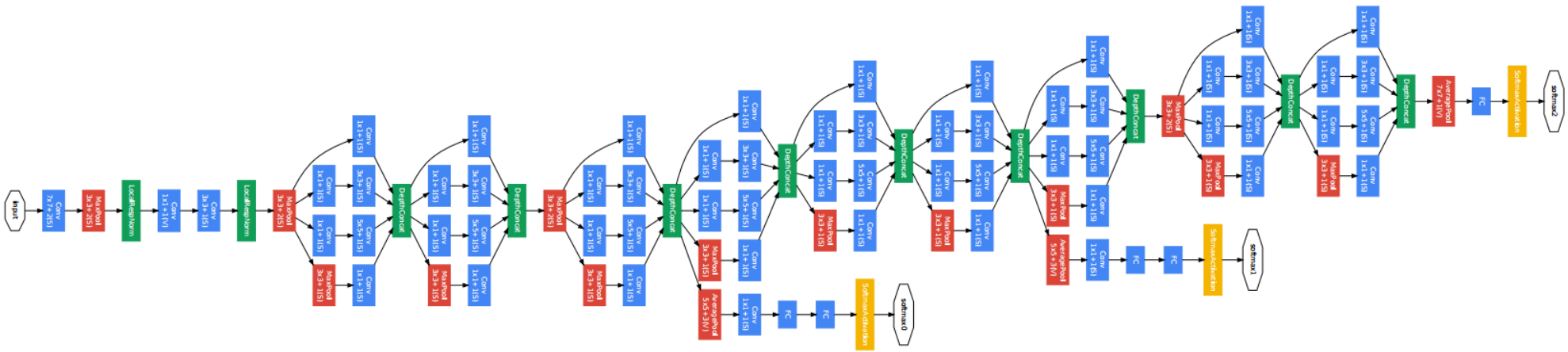
The power of small filters

- Suppose input is $W \times H \times D$ and use D filters to preserve the depth (stride $= 1$, zero-padding to preserve W, H).
- **Option 1** Apply one layer of D filters of size 7×7
- **Option 2** Apply three layers of D filters of size 3×3
- After both options the activations are calculated from the same sized regions in the input image.
- Overview of options:

Option	# of layers	size of filters	# of parameters	# of $\times$'s & $+$'s
1	1	7×7	$D(7 * 7 * D) = 49D^2$	$49WHD^2$
2	3	3×3	$3D(3 * 3 * D) = 27D^2$	$27WHD^2$

- Option 2 requires fewer parameters, less computational effort and applies more non-linearity.

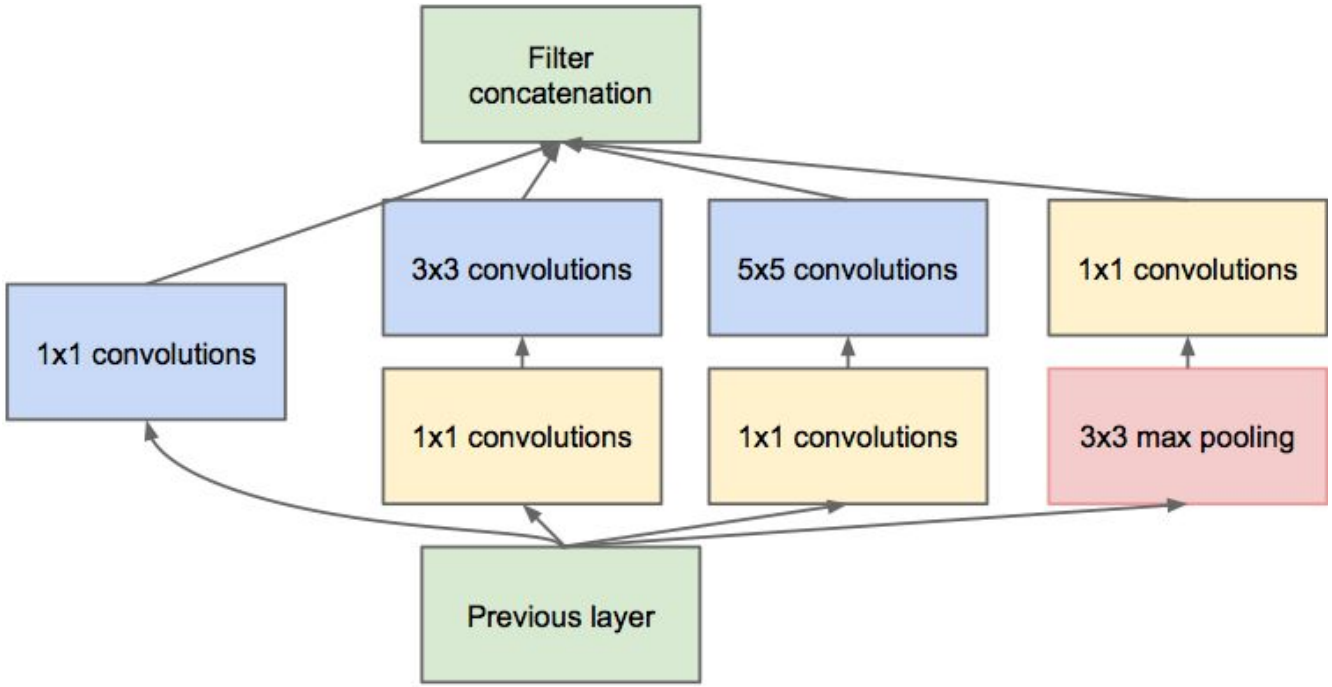
GoogLeNet [Szegedy et al., 2014]



- Convolution layer
- Pooling layer
- SoftMax layer

ILSVRC 2014 winner (6.7% top 5 error)

GoogLeNet: Inception Module



Use $1 \times 1 \times D$ filters to reduce the depth of the response volume before applying the larger more computationally expensive filters.

Back to the power of small filters

- Why stop at 3×3 filters?
- Why not try 1×1 filters?

Back to the power of small filters

- Why not try 1×1 filters?

$$W \times H \times D$$

$$\Downarrow \text{ Apply } \frac{D}{2} \ 1 \times 1 \text{ conv filters}$$

$$W \times H \times \frac{D}{2}$$

1. Use *bottleneck* 1×1 conv filters to reduce the depth dimensionality.

Back to the power of small filters

- Why not try 1×1 filters?

$$W \times H \times D$$

$$\Downarrow \text{ Apply } \frac{D}{2} \; 1 \times 1 \text{ conv filters}$$

$$W \times H \times \frac{D}{2}$$

$$\Downarrow \text{ Apply } \frac{D}{2} \; 3 \times 3 \text{ conv filters}$$

$$W \times H \times \frac{D}{2}$$

1. Use *bottleneck* 1×1 conv filters to reduce the depth dimensionality.
2. Apply 3×3 conv filters at the reduced depth.

Back to the power of small filters

- Why not try 1×1 filters?

$$W \times H \times D$$

$$\Downarrow \text{ Apply } \frac{D}{2} \; 1 \times 1 \text{ conv filters}$$

$$W \times H \times \frac{D}{2}$$

$$\Downarrow \text{ Apply } \frac{D}{2} \; 3 \times 3 \text{ conv filters}$$

$$W \times H \times \frac{D}{2}$$

$$\Downarrow \text{ Apply } D \; 1 \times 1 \text{ conv filters}$$

$$W \times H \times D$$

1. Use *bottleneck* 1×1 conv filters to reduce the depth dimensionality.
2. Apply 3×3 conv filters at the reduced depth.
3. Restore original depth dimension with another D 1×1 conv filters.

Back to the power of small filters

- Why not try 1×1 filters?

$W \times H \times D$

$\Downarrow$ Apply $\frac{D}{2}$ 1×1 conv filters

$W \times H \times \frac{D}{2}$

$\Downarrow$ Apply $\frac{D}{2}$ 3×3 conv filters

$W \times H \times \frac{D}{2}$

$\Downarrow$ Apply D 1×1 conv filters

$W \times H \times D$

Bottleneck sandwich

$W \times H \times D$

$\Downarrow$ Apply D 3×3 conv filters

$W \times H \times D$

Single 3×3 conv filter

Back to the power of small filters

- Why not try 1×1 filters?

$W \times H \times D$

$\Downarrow$ Apply $\frac{D}{2}$ 1×1 conv filters

$W \times H \times \frac{D}{2}$

$\Downarrow$ Apply $\frac{D}{2}$ 3×3 conv filters

$W \times H \times \frac{D}{2}$

$\Downarrow$ Apply D 1×1 conv filters

$W \times H \times D$

$3.25D^2$ parameters

$W \times H \times D$

$\Downarrow$ Apply D 3×3 conv filters

$W \times H \times D$

$9D^2$ parameters

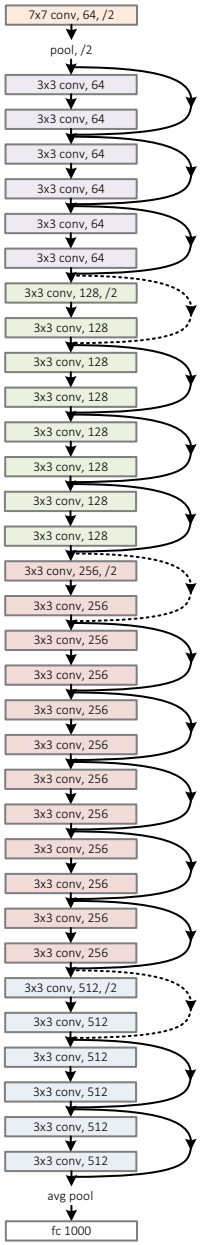
More non-linearity, fewer params, less computations

GoogLeNet: architecture

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
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

- Only 5 million params! (Only one FC layer $\Longrightarrow$ far fewer parameters)
- Compared to AlexNet:
  - 12× less params
  - 2× more compute
  - Performance on ILSVRC: 6.67% (vs. 16.4%)

ResNet [He et al., 2015]



- ILSVRC 2015 winner (3.6% top 5 error)
- 2-3 weeks of training on 8 GPU machine
- At runtime: faster than a VGGNet! (even though it has 8× more layers)

← "shallow" version of winning entry.

Next slides from:

Deep Residual Networks, Deep Learning Gets Way Deeper by Kaiming He, ICML 2016 tutorial.

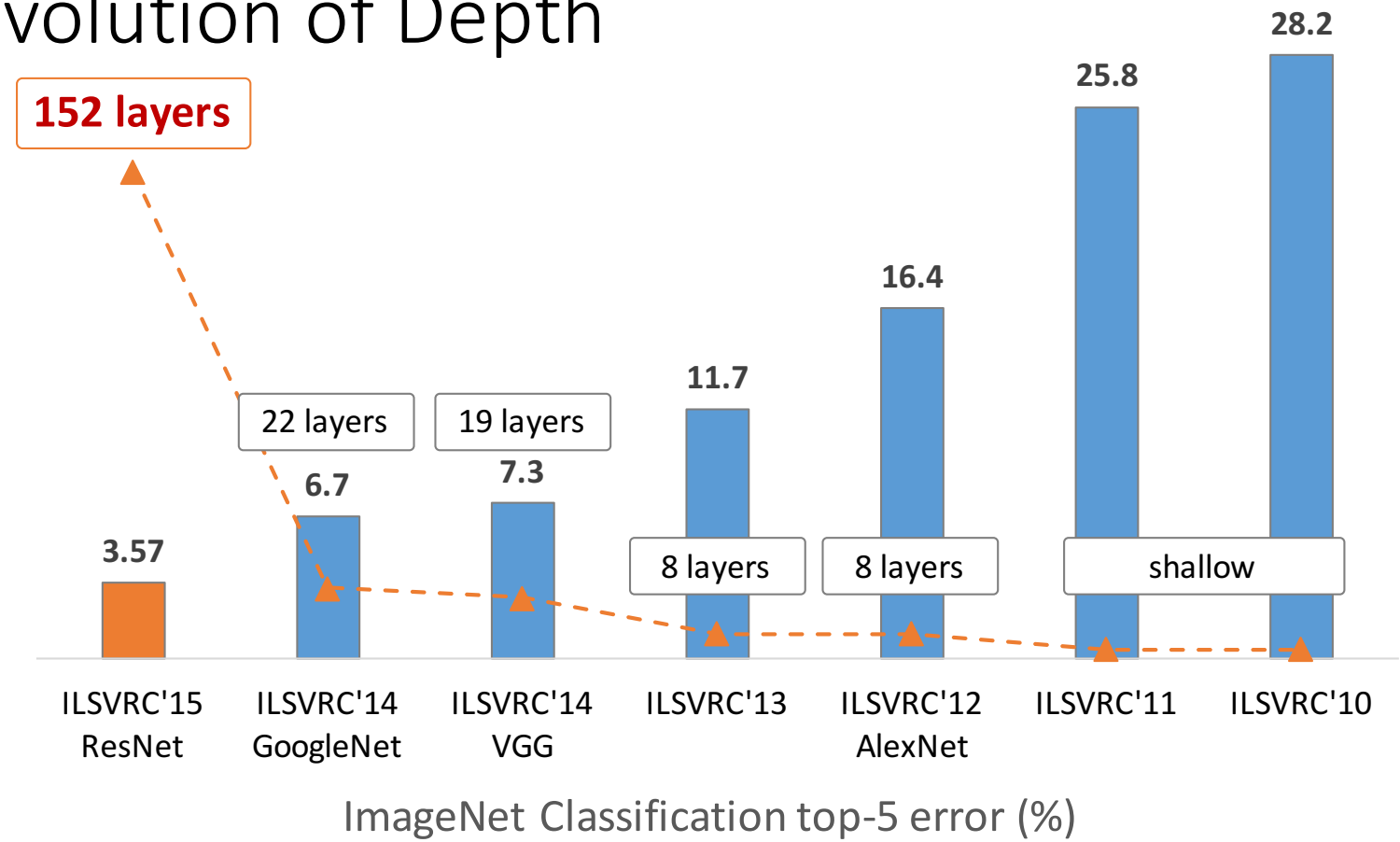
ResNets @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" 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

*improvements are relative numbers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

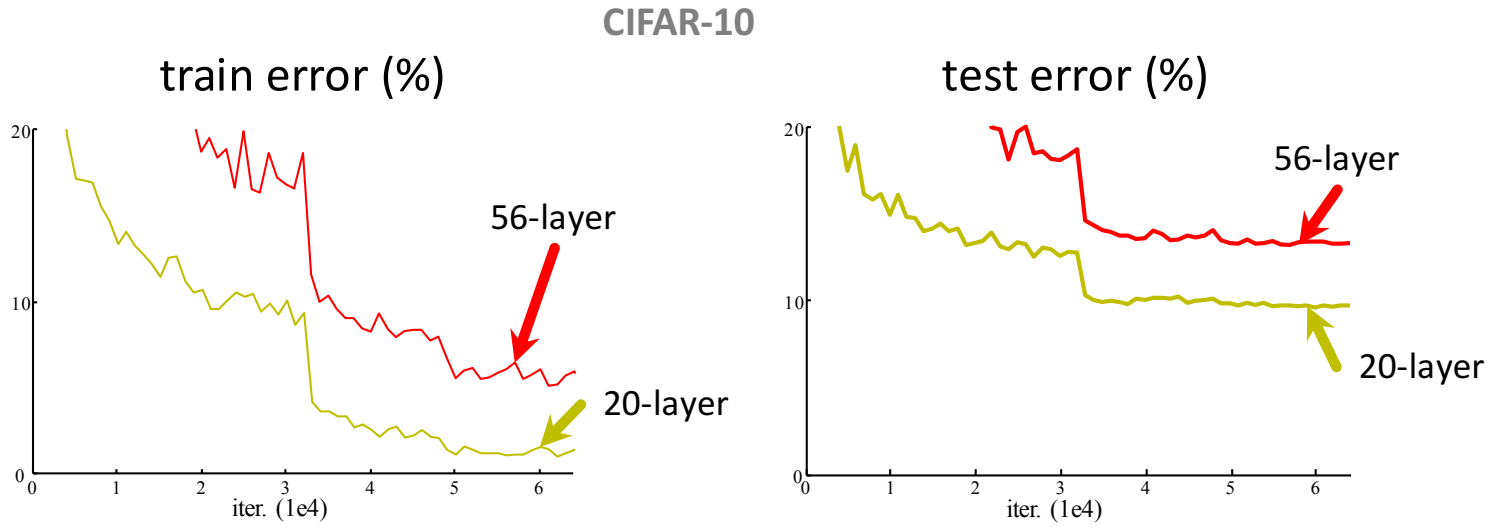
ResNet Motivation [He et al., 2015]

Going Deeper requires

- Care with initialization ✓
- Batch Normalization ✓

Is learning better networks as simple as stacking more layers?

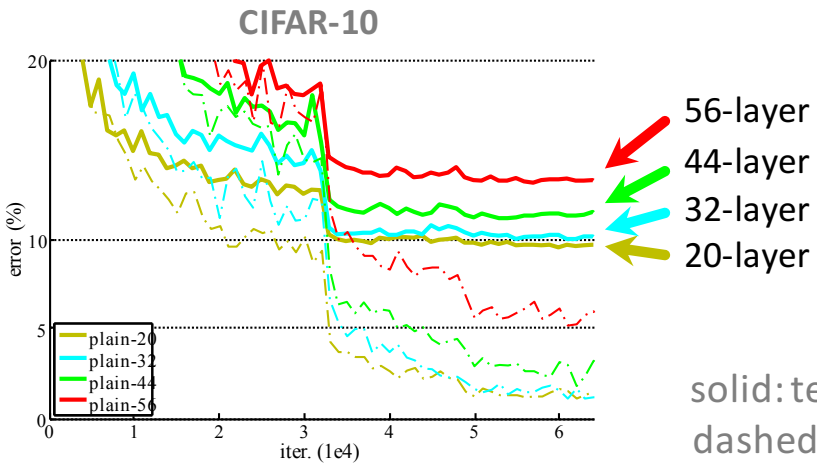
Simply stacking layers?

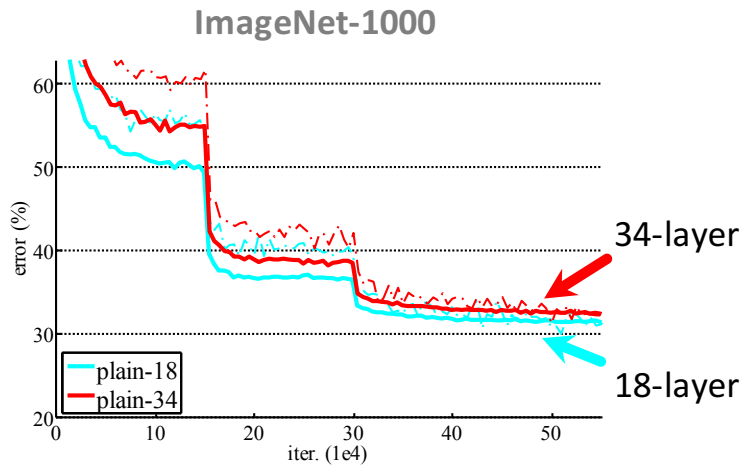


- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Simply stacking layers?

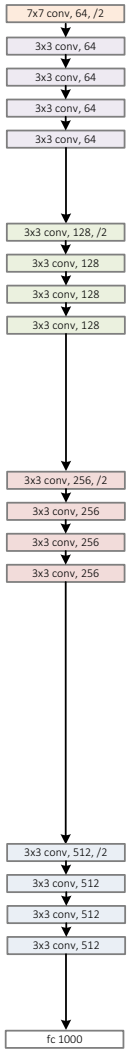


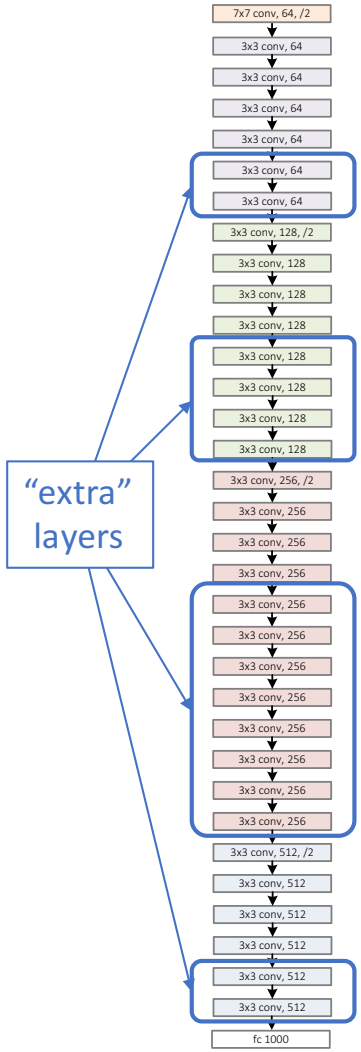


- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

a shallower model (18 layers)





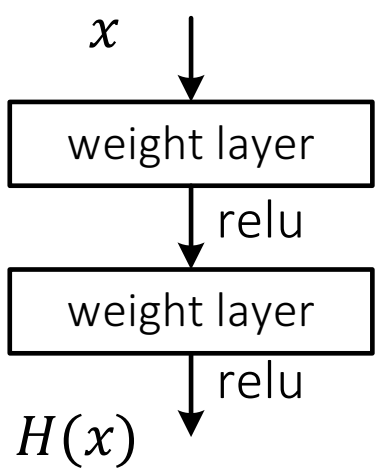
a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Deep Residual Learning

- Plaint net



any two stacked layers

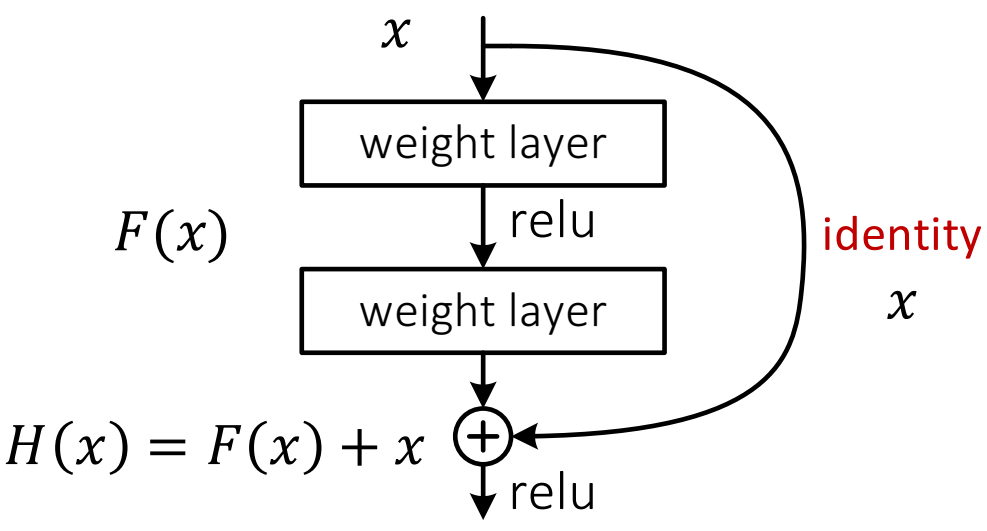
$H(x)$ is any desired mapping,

hope the 2 weight layers fit $H(x)$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Deep Residual Learning

- Residual net



$H(x)$ is any desired mapping,

~~hope the 2 weight layers fit $H(x)$~~

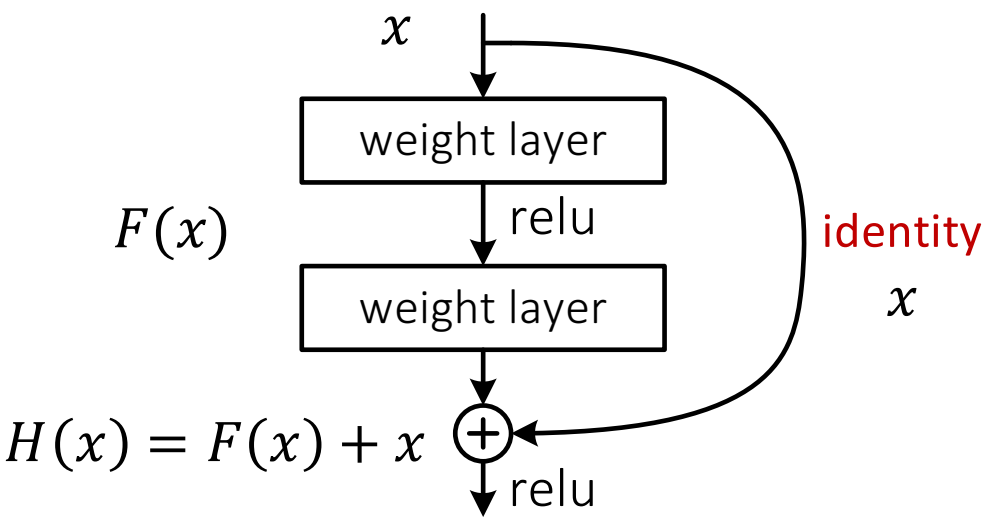
hope the 2 weight layers fit $F(x)$

let $H(x) = F(x) + x$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Deep Residual Learning

- $F(x)$ is a residual mapping w.r.t. identity



- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

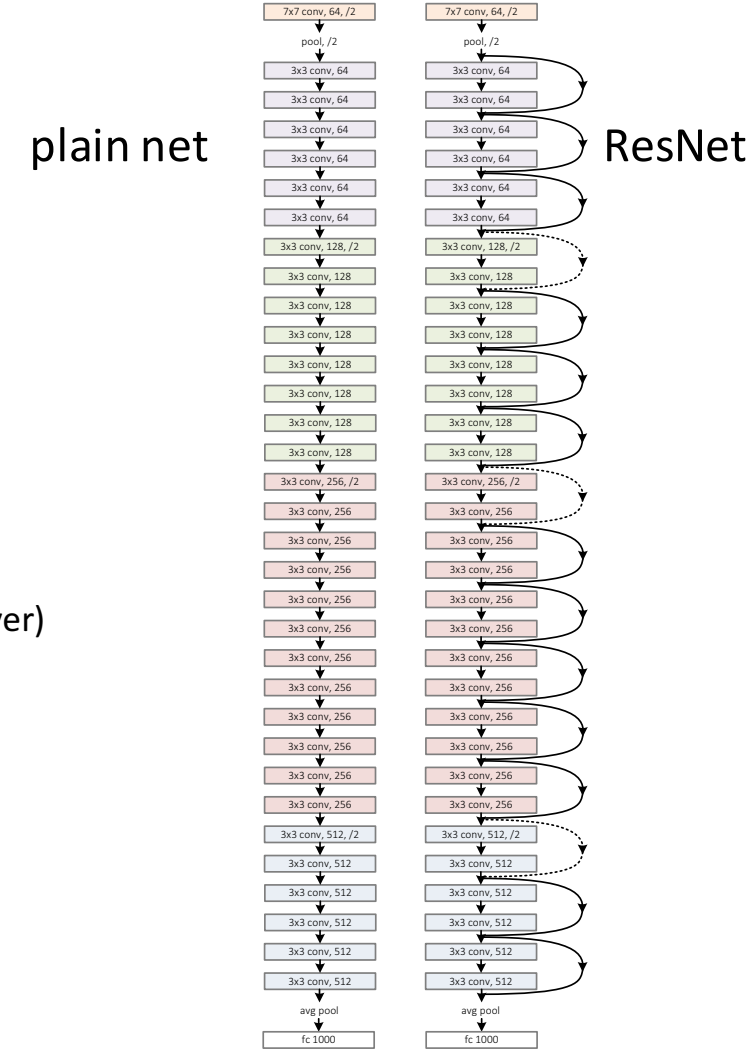
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Network “Design”

- Keep it simple

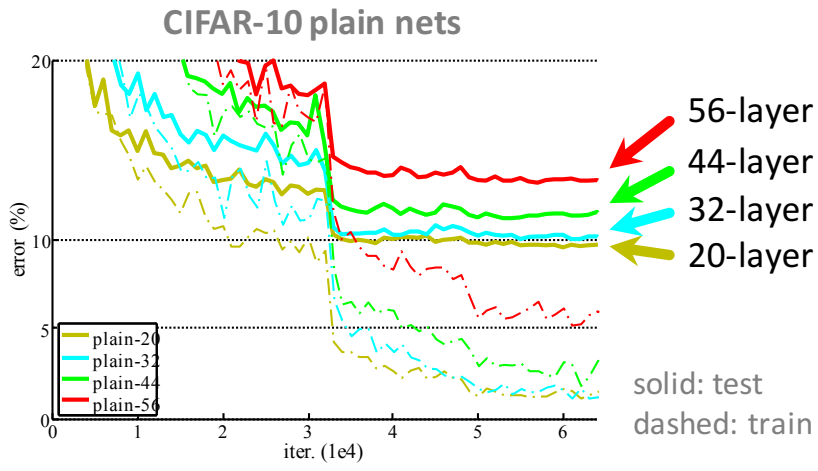
- Our basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2 (~same complexity per layer)
  - Simple design; just deep!

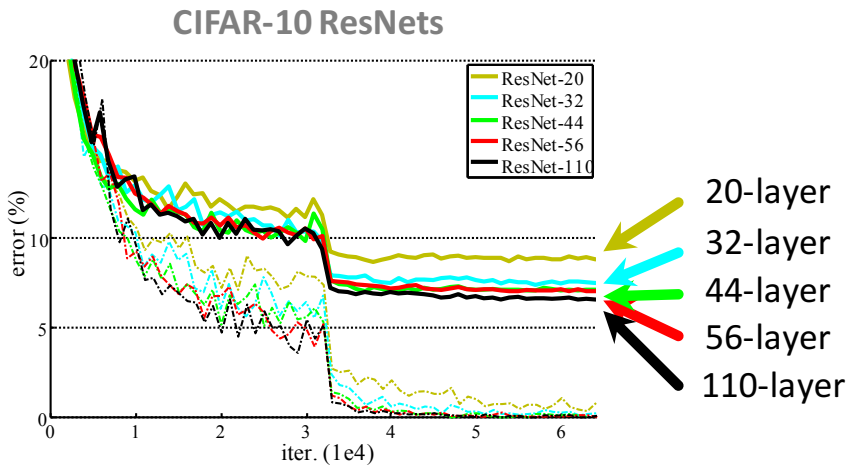
- Other remarks:
  - no hidden fc
  - no dropout



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016.

CIFAR-10 experiments



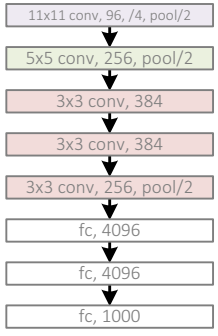


- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

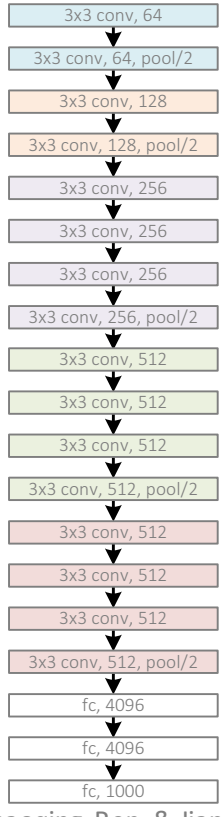
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

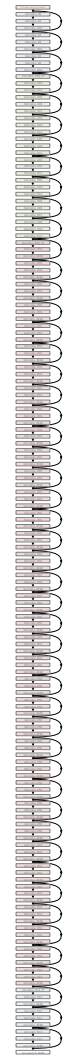
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

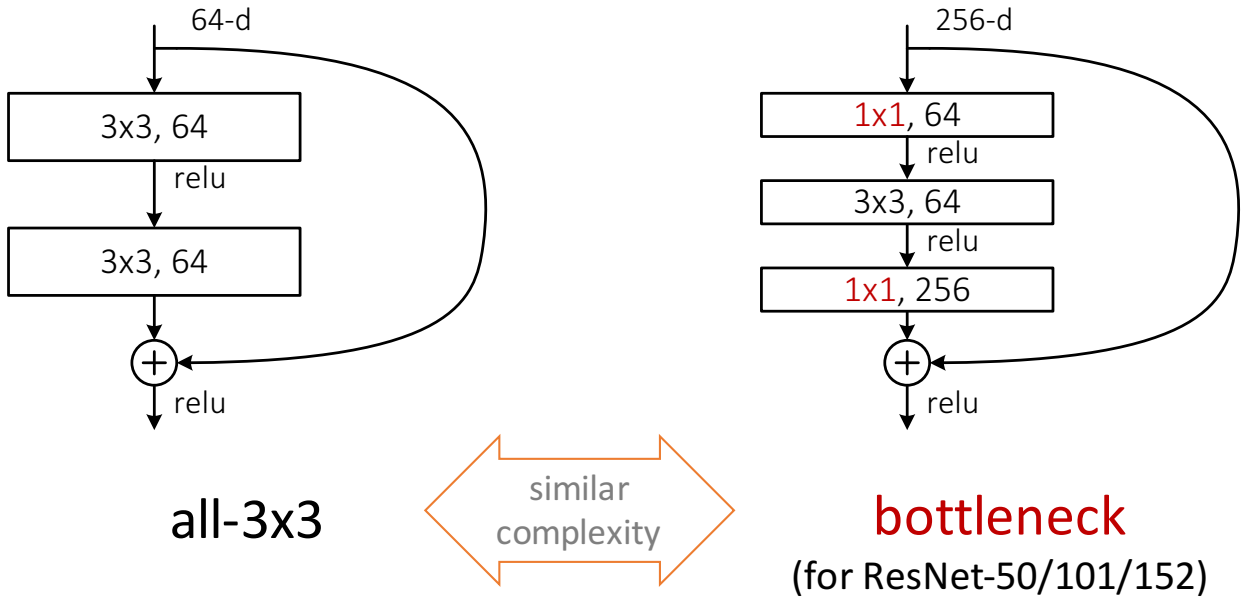


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

- Batch Normalization after every CONV layer.
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9).
- Learning rate: 0.1, divided by 10 when validation error plateaus.
- Mini-batch size 256.
- Weight decay of 1e-5.
- No dropout used.

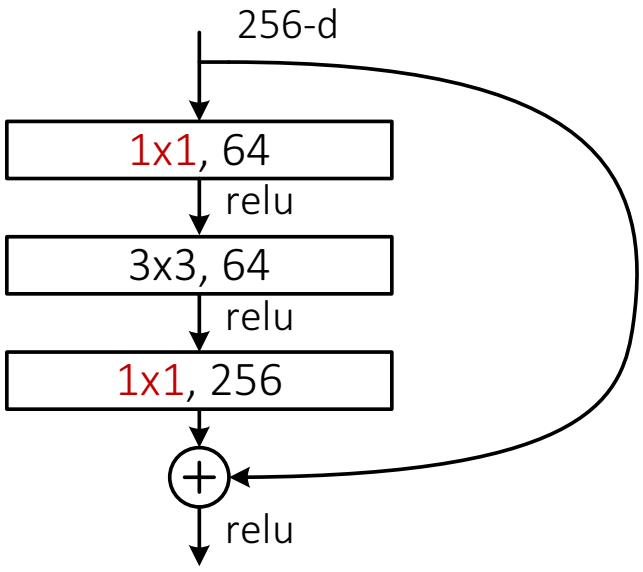
ImageNet experiments

- A practical design of going deeper

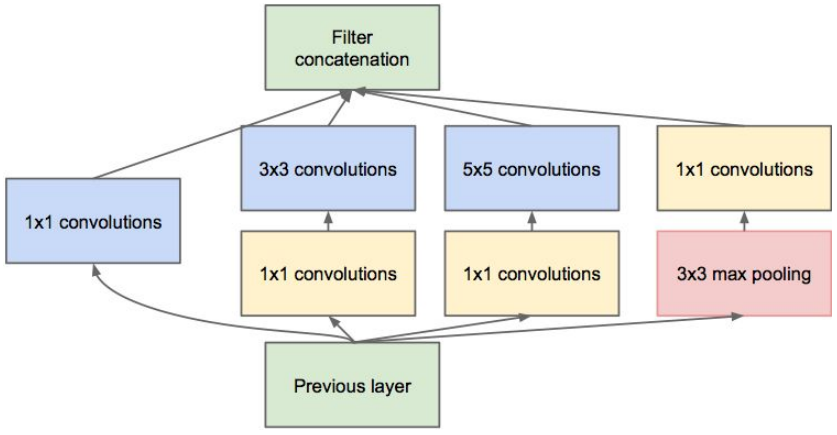


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Same trick as used in GoogLeNet

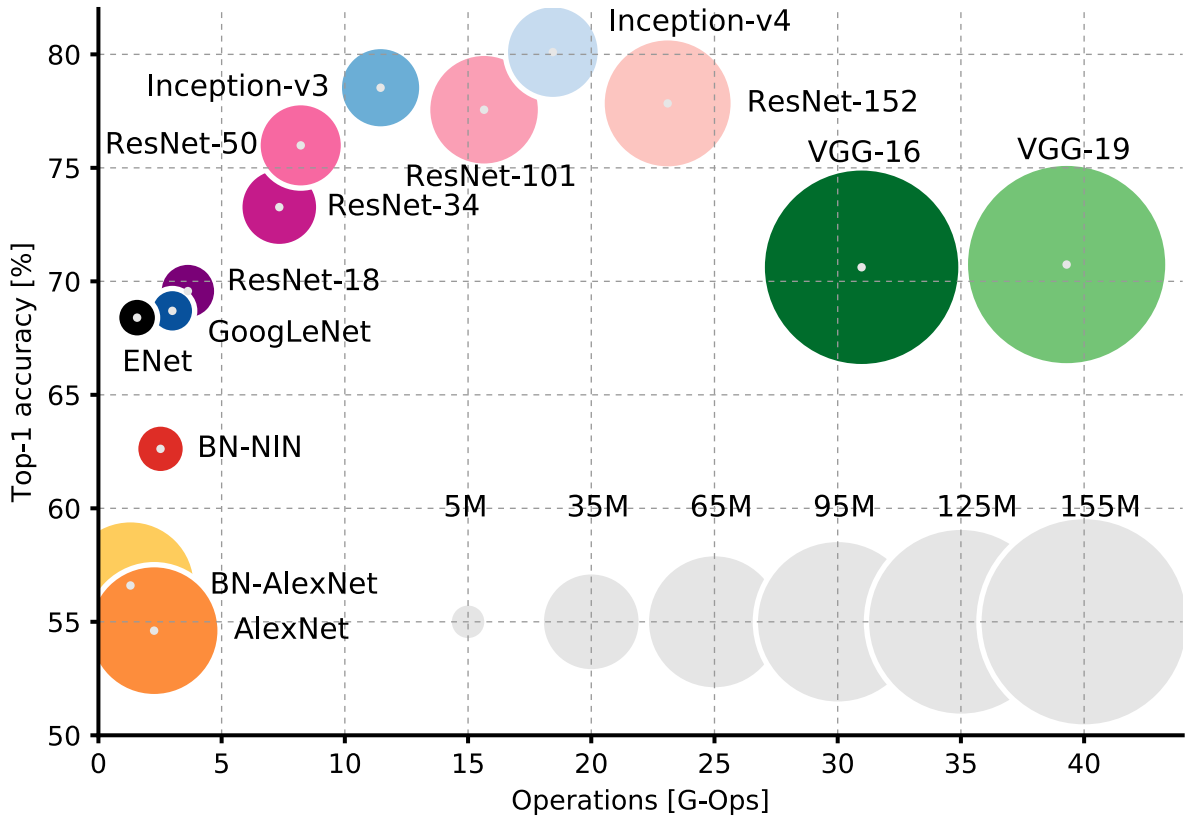


ResNet



GoogLeNet

Accuracy on ImageNet Vs network size



- Top-1 one-crop accuracy Vs amount of operations required for a single forward pass.
- The size of the blobs is proportional to the number of network parameters.

Fig: **An Analysis of Deep Neural Network Models for Practical Applications** by Canziani, Culurciello & Paszke.

Summary: ConvNet Architectures

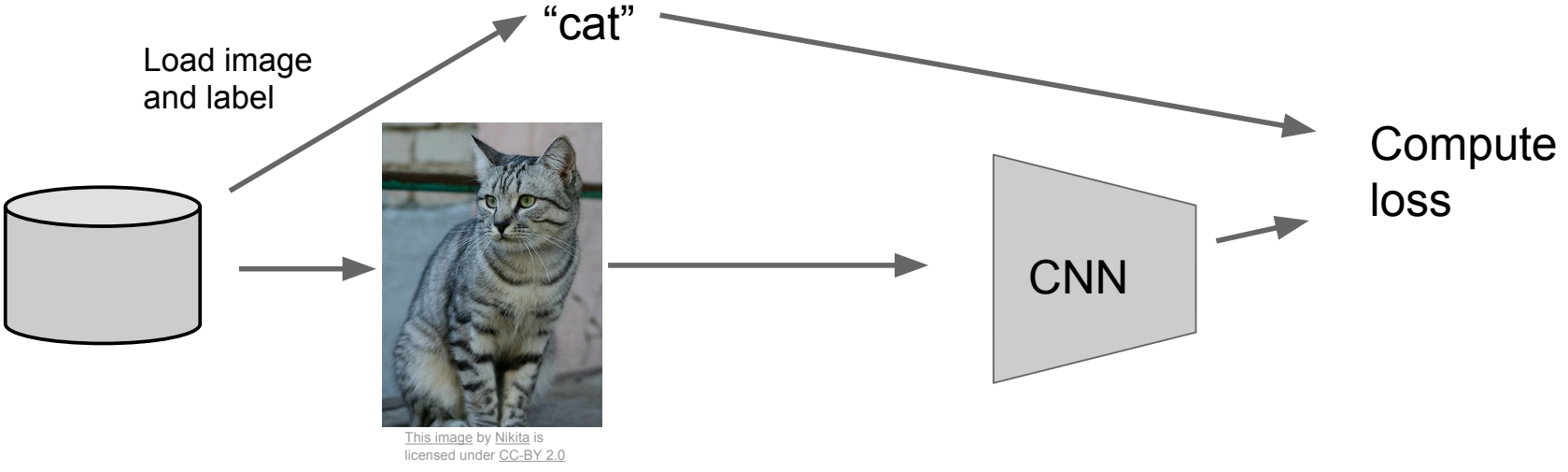
- VGG, GoogLeNet, ResNet all in wide use, available in model zoos.
- ResNet current best default.
- Research focused on design of layer / skip connections and improving gradient flow
- More recent trend towards examining necessity of depth vs. width and residual connections.

Practicalities of training ConvNets

- Data augmentation.
- Batch Normalization for convolutional layers
- Transfer learning.

Data Augmentation - regularization strategy to avoid over-fitting

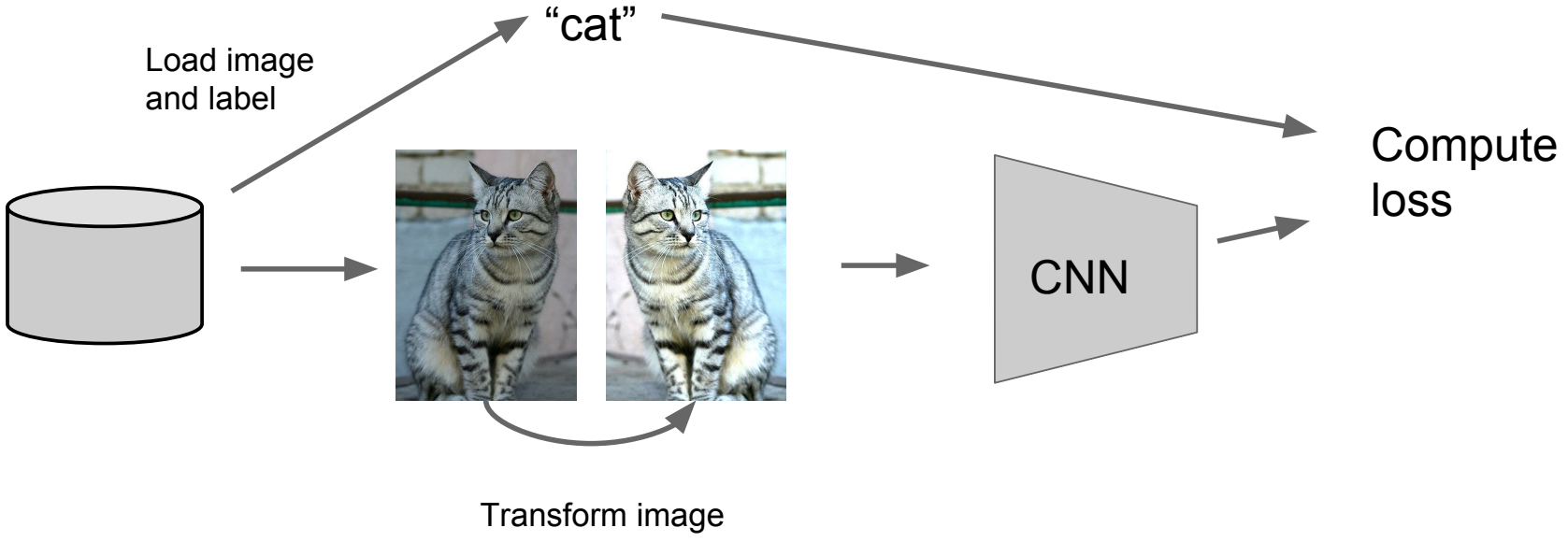
Straightforward approach to training examples



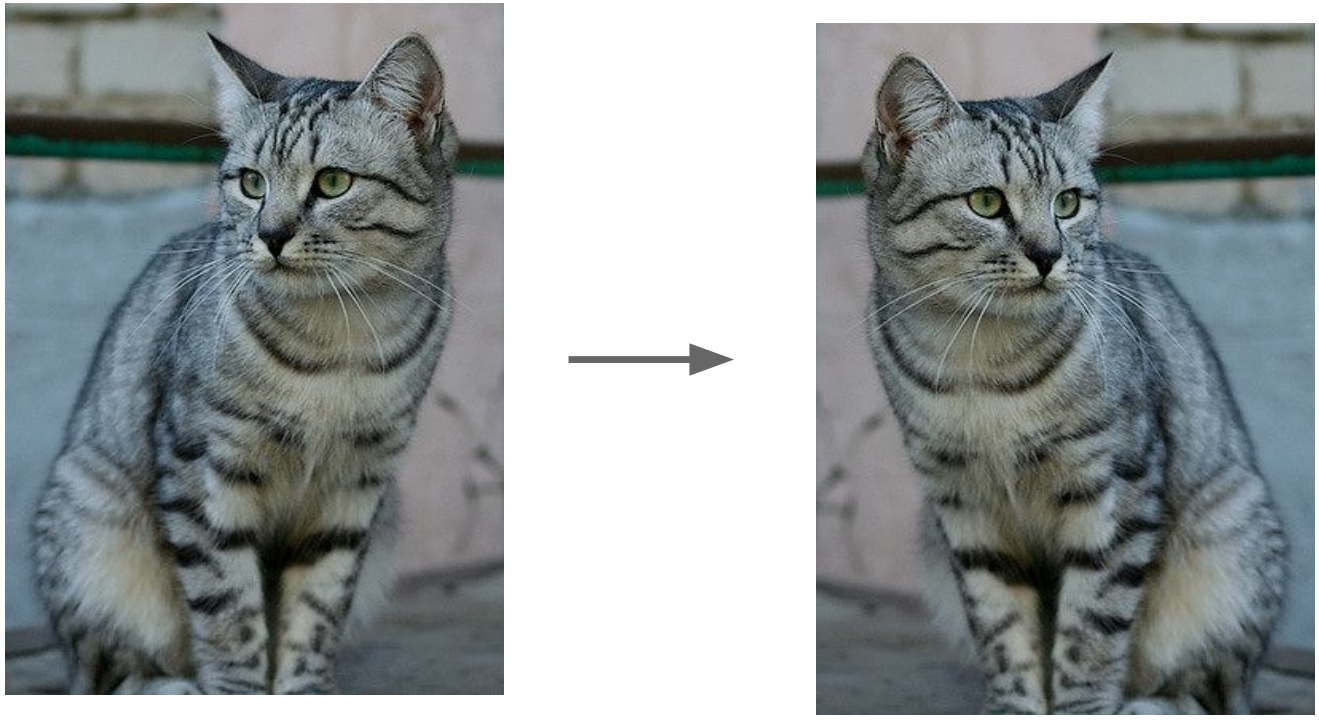
Have n labelled training examples $\implies$ have n training examples.

Data augmentation

- Encounter a training example *transform* it.
- If have n labelled training examples $\implies$ potentially infinite training examples.



Horizontal Flip

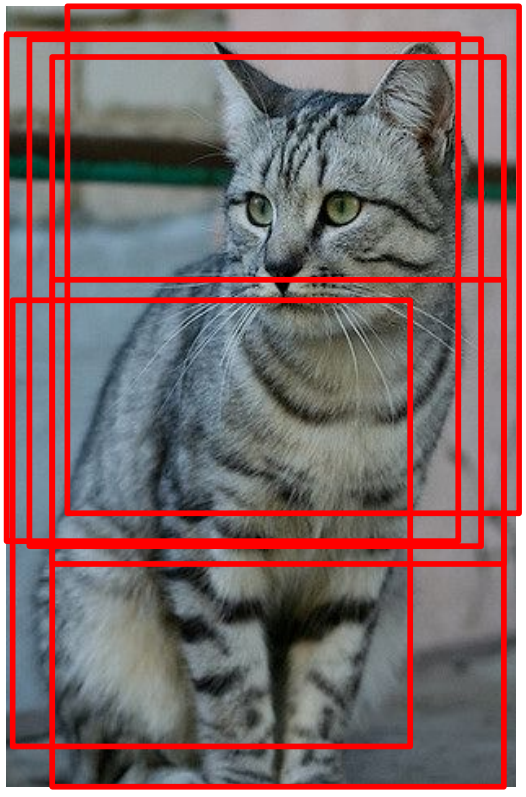


Random crops and scales

During training: Sample random crops / scales

Example for ResNet:

1. Pick random L in range $[256, 480]$
2. Resize training image, short side $= L$
3. Sample random 224×224 patch.

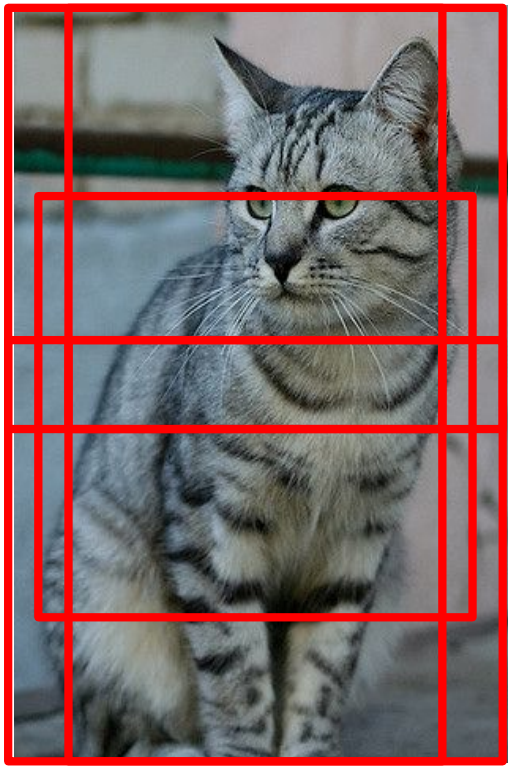


Random crops and scales

At testing time: Average a fixed set of crops

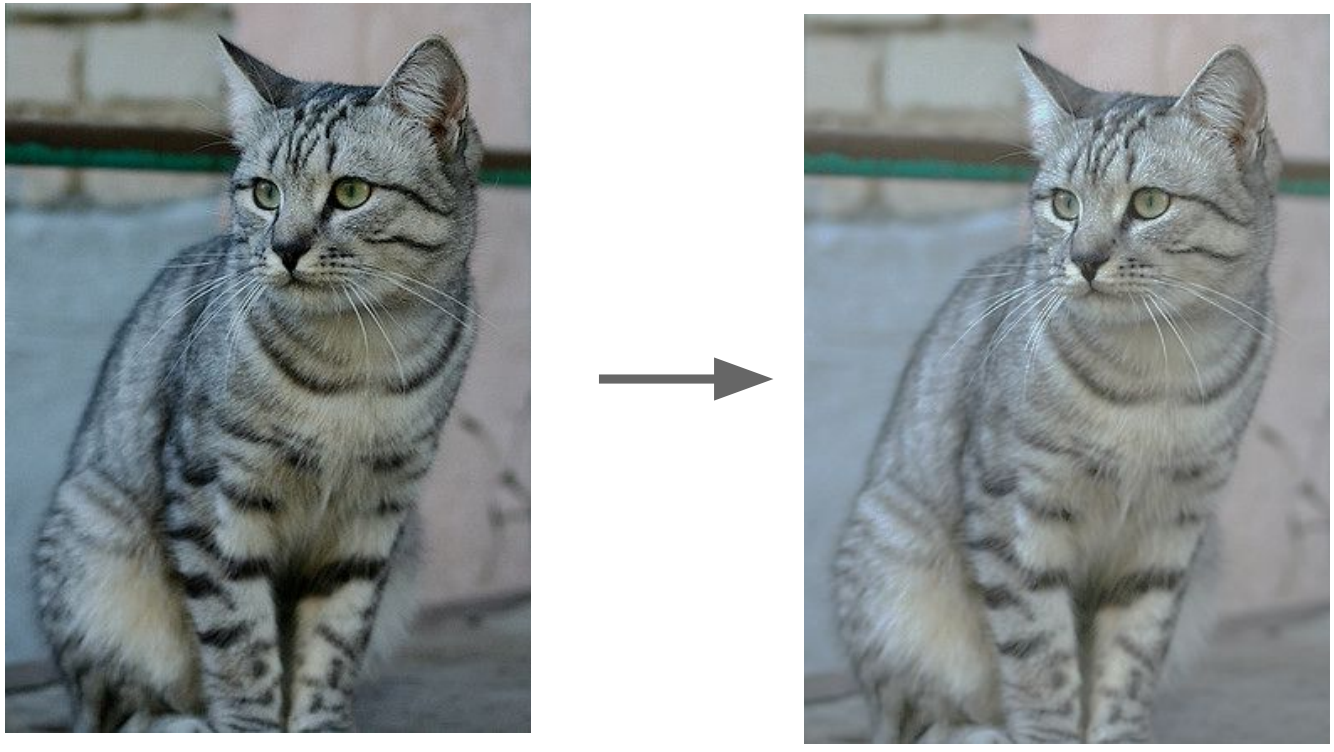
Example for ResNet:

1. Resize image at 5 scales: $\{224, 256, 384, 480, 640\}$
2. For each size, use 10 224×224 crops: (4 corners + center) + flips



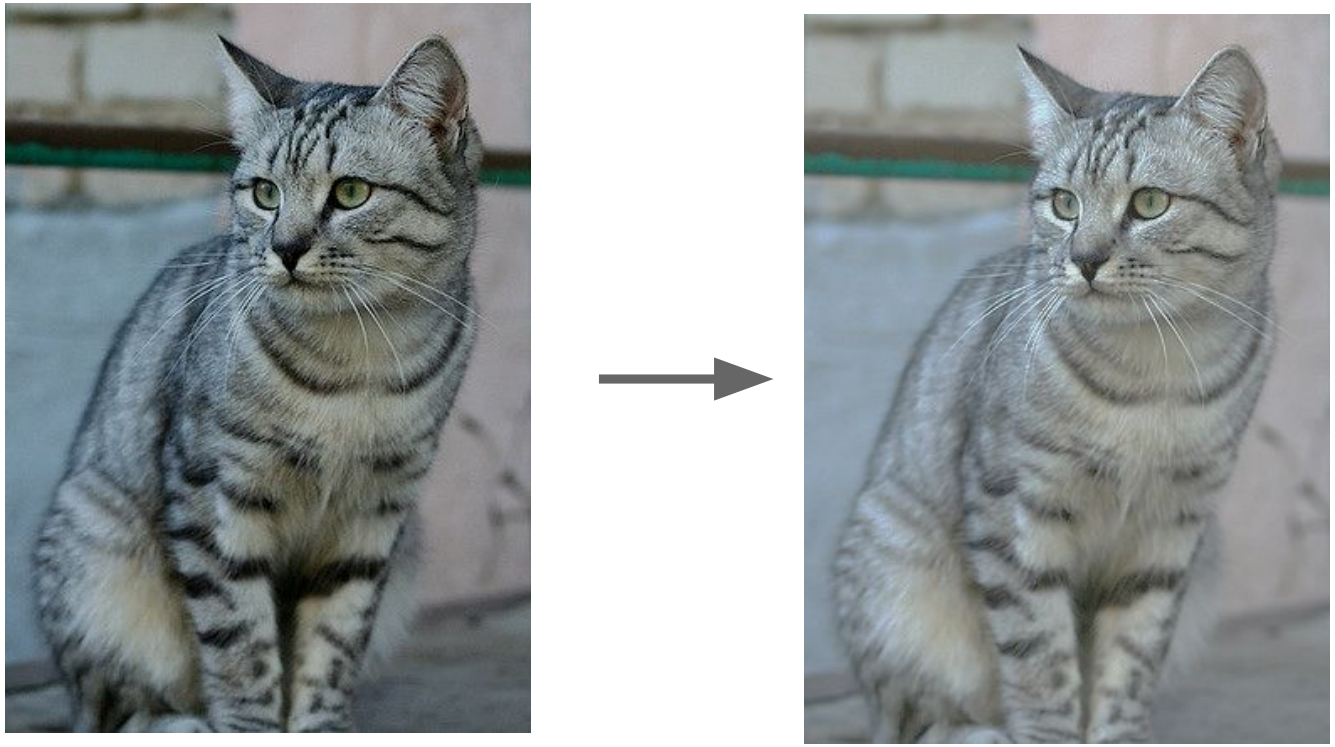
Colour Jitter - simple version

- Randomize contrast and brightness



Colour Jitter - more complex

1. Apply PCA to all (r, g, b) pixels in training set.
2. Sample a "*colour offset*" along the principal component directions.
3. Add offset to all pixels of a training image.



Types of data augmentation

Only limited by your imagination and computational resources

Random mix/combinations of

- translation
- rotation
- stretching
- shearing,
- lens distortions,
- ...

Batch Normalization: Help with gradient flow + regularization

Batch Normalization for Convolutional Layers

Option 1: Perform batch norm. as in a fully connected layer.

- Let $\mathcal{B} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ be the mini-batch of data.
- Each $\mathbf{X} \in \mathcal{B}$ has the form $\mathbf{X} = \{X_1, \ldots, X_D\}$ where X_i is $W \times H$
- Compute the mean 3D array $\mathbf{M}$ ($W \times H \times D$) for the batch:

$$\mathbf{M} = \frac{1}{n} \sum_{\mathbf{X} \in \mathcal{B}} \mathbf{X}$$

- Compute the 3D array $\mathbf{V}$ ($W \times H \times D$) of variances for the batch:

$$\mathbf{V} = \frac{1}{n} \sum_{\mathbf{X} \in \mathcal{B}} (\mathbf{X} - \mathbf{M}) \odot (\mathbf{X} - \mathbf{M}) \quad \leftarrow \odot \text{ denotes element-wise mult.}$$

- for each $\mathbf{X} \in \mathcal{B}$ set

$$\hat{X}_{k,ij} = \frac{X_{k,ij} - M_{k,ij}}{\sqrt{V_{k,ij}}} \text{ for } k = 1, \ldots, D,\ i = 1, \ldots, W,\ j = 1, \ldots, H$$

Batch Normalization for Convolutional Layers

Option 1: Perform batch norm. as in a fully connected layer.

- Let $\mathcal{B} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ be the mini-batch of data.
- Each $\mathbf{X} \in \mathcal{B}$ has the form $\mathbf{X} = \{X_1, \ldots, X_D\}$ where X_i is $W \times H$
- Compute the mean 3D array $\mathbf{M}$ $(W \times H \times D)$ for the batch:

$$\mathbf{M} = \frac{1}{n} \sum_{\mathbf{X} \in \mathcal{B}} \mathbf{X}$$

- Compute the 3D array $\mathbf{V}$ $(W \times H \times D)$ of variances for the batch:

$$\mathbf{V} = \frac{1}{n} \sum_{\mathbf{X} \in \mathcal{B}} (\mathbf{X} - \mathbf{M}) \odot (\mathbf{X} - \mathbf{M}) \quad \leftarrow \odot \text{ denotes element-wise mult.}$$

- for each $\mathbf{X} \in \mathcal{B}$ set

$$\hat{X}_{k,ij} = \frac{X_{k,ij} - M_{k,ij}}{\sqrt{V_{k,ij}}} \text{ for } k = 1, \ldots, D,\ i = 1, \ldots, W,\ j = 1, \ldots, H$$

Does this make much sense? **No**. Have a separate mean and variance for each spatial location $\Rightarrow\Leftarrow$.

Batch Normalization for Convolutional Layers

Option 2: Compute means and variances per filter.

- Let $\mathcal{B} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ be the mini-batch of data.
- Each $\mathbf{X} \in \mathcal{B}$ has the form $\mathbf{X} = \{X_1, \ldots, X_D\}$ where X_i is $W \times H$
- Compute the mean vector $\mathbf{m}$ $(D \times 1)$ for the batch:

$$m_k = \frac{1}{nWH} \sum_{\mathbf{X} \in \mathcal{B}} \sum_{i=1}^{W} \sum_{j=1}^{H} X_{k,ij} \text{ for } k = 1, \ldots, D$$

- Compute the vector $\mathbf{v}$ $(D \times 1)$ of variances for the batch:

$$v_k = \frac{1}{nWH} \sum_{\mathbf{X} \in \mathcal{B}} \sum_{i=1}^{W} \sum_{j=1}^{H} (X_{k,ij} - m_k)^2 \text{ for } k = 1, \ldots, D$$

- for each $\mathbf{X} \in \mathcal{B}$ set

$$\hat{X}_k = \frac{X_k - m_k}{\sqrt{v_k}} \text{ for } k = 1, \ldots, D$$

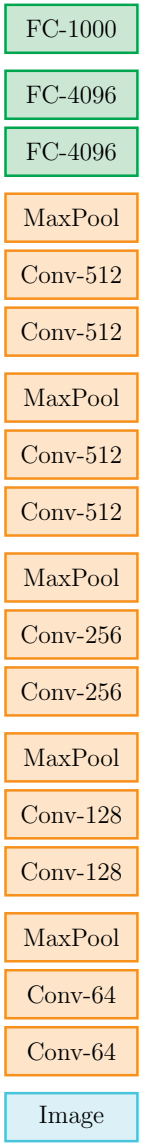
Transfer Learning

"*You need a lot of labelled data if you want to train/use ConvNets*"

Transfer Learning

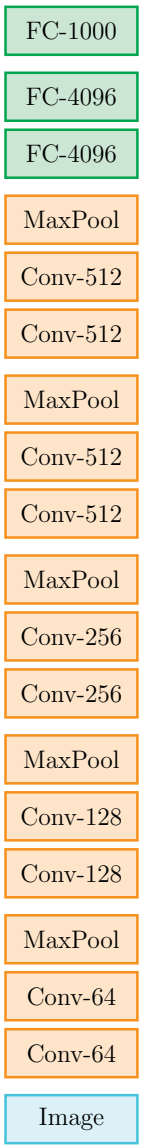
"*You need a lot of labelled data if you want to train/use ConvNets*"

1. Train on ImageNet

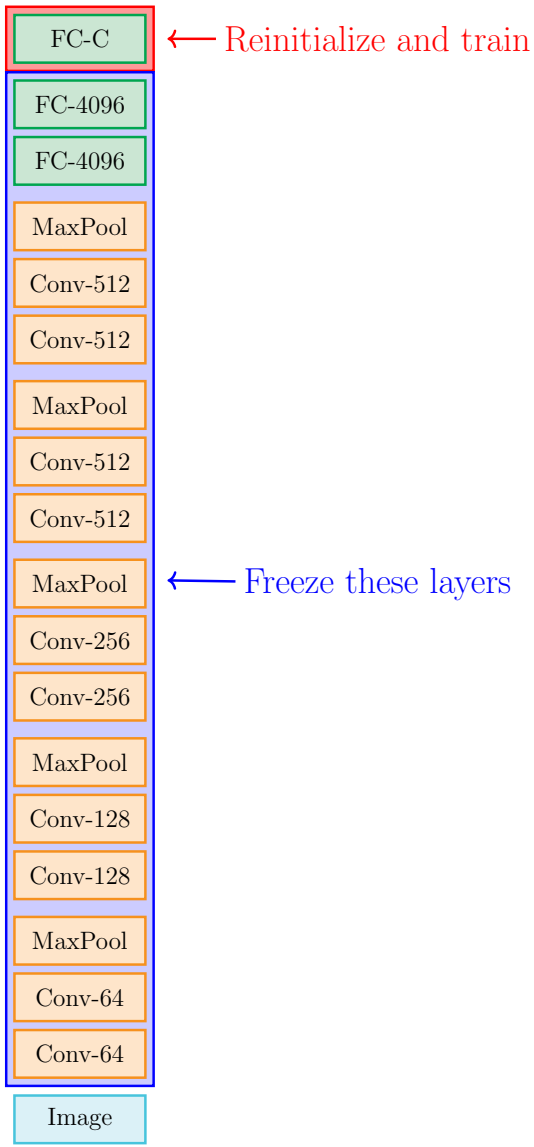


Transfer Learning with ConvNets

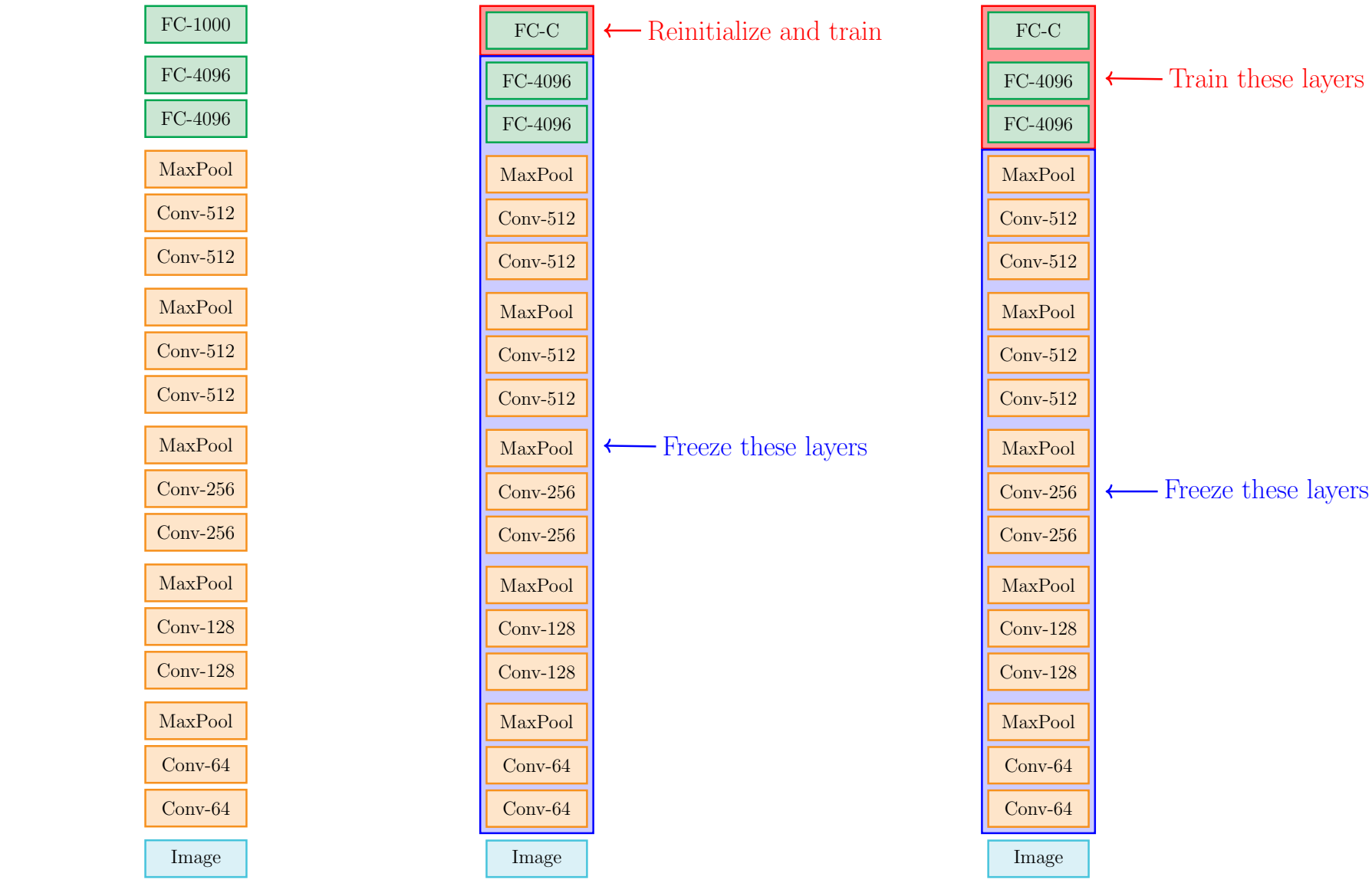
1. Train on ImageNet



2. Small dataset (C classes)

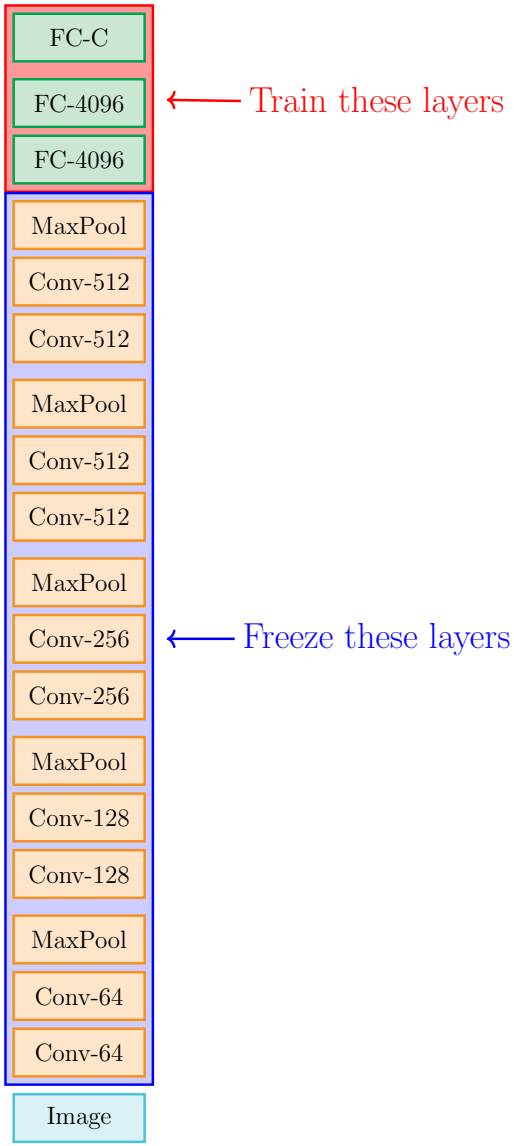


Transfer Learning with ConvNets



Transfer Learning with ConvNets

Bigger dataset



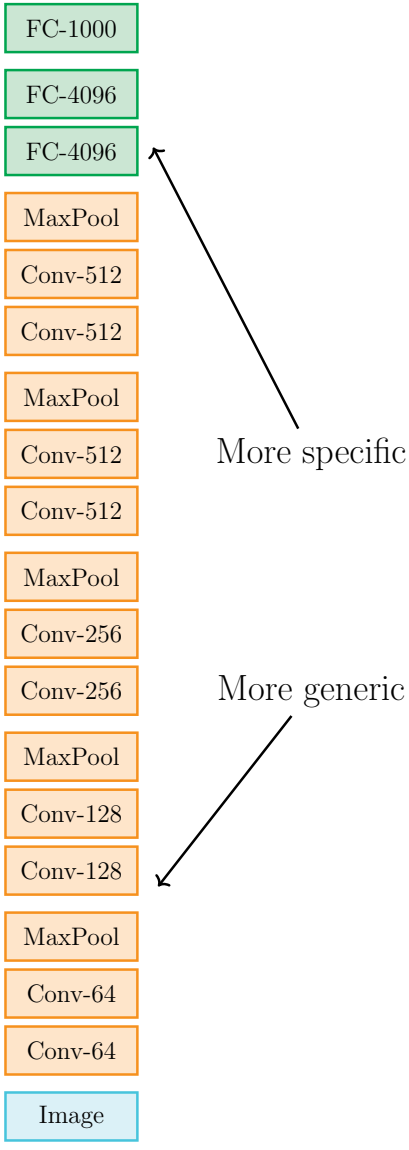
- With bigger dataset, train more layers.
- Lower learning rate when finetuning; 1/10 of original learning rate is good starting point.

Transfer Learning with ConvNets References

- Donahue et al., *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition*, ICML 2014

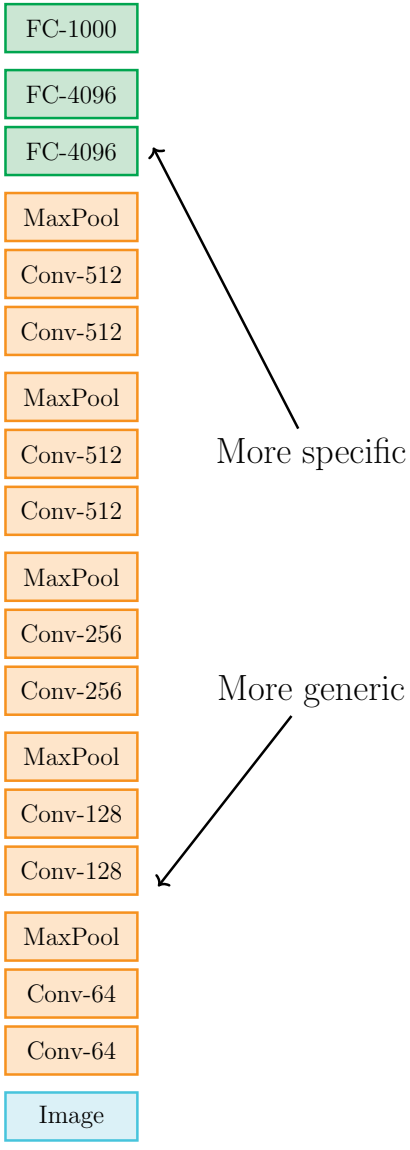
- Razavian et al., *CNN Features Off-the-Shelf: An Astounding Baseline for Recognition*, CVPR Workshops 2014

Transfer Learning with ConvNets



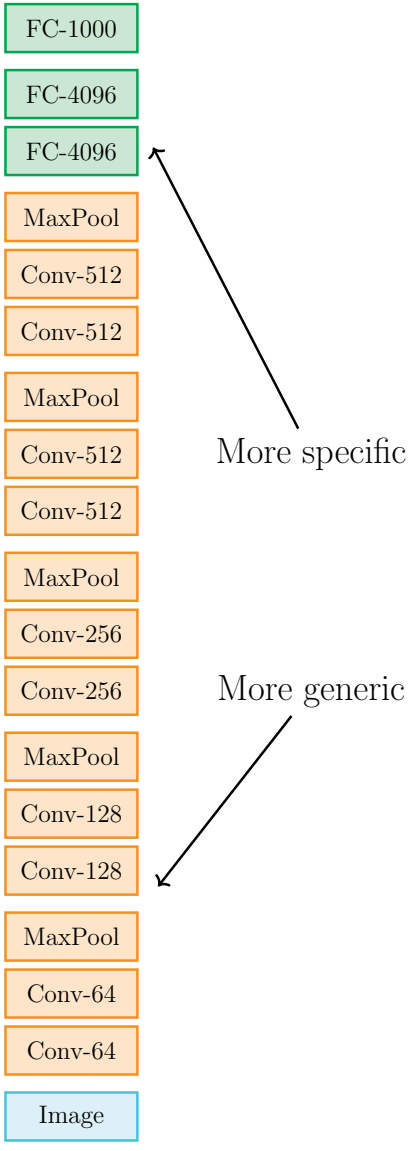
	Your new dataset	
Amount of data	**very similar**	**very different**
very little	?	?
quite a lot	?	?

Transfer Learning with ConvNets



	Your new dataset	
Amount of data	**very similar**	**very different**
very little	Use linear classifier on top layer.	?
quite a lot	Finetune a few layers	?

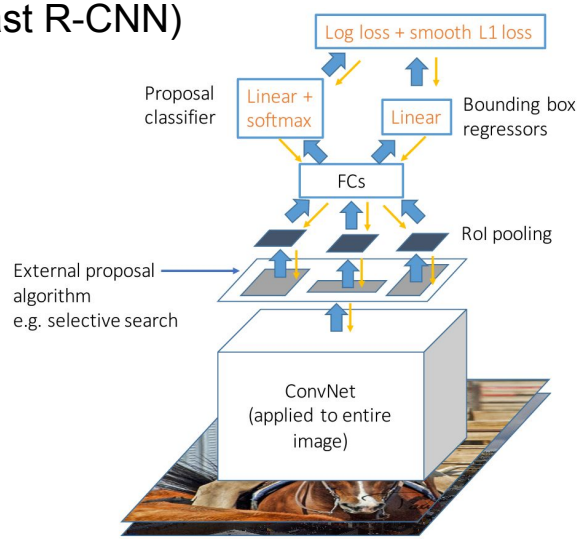
Transfer Learning with ConvNets



	Your new dataset	
Amount of data	**very similar**	**very different**
very little	Use linear classifier on top layer.	Oh dear! Try linear classifier from different stages.
quite a lot	Finetune a few layers	Finetune a larger number of layers

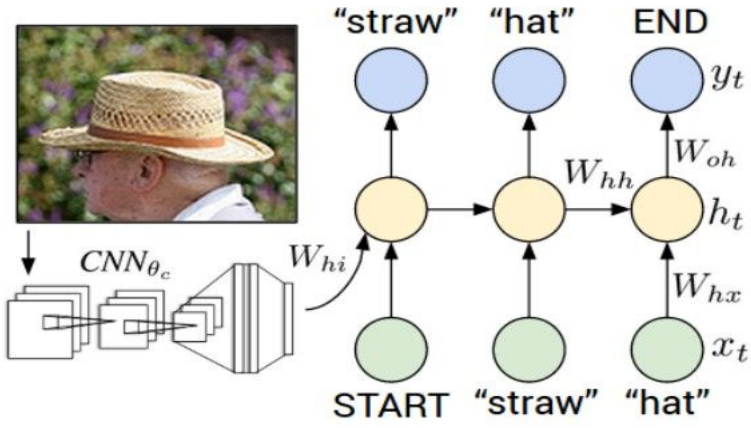
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



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

Image Captioning: CNN + RNN



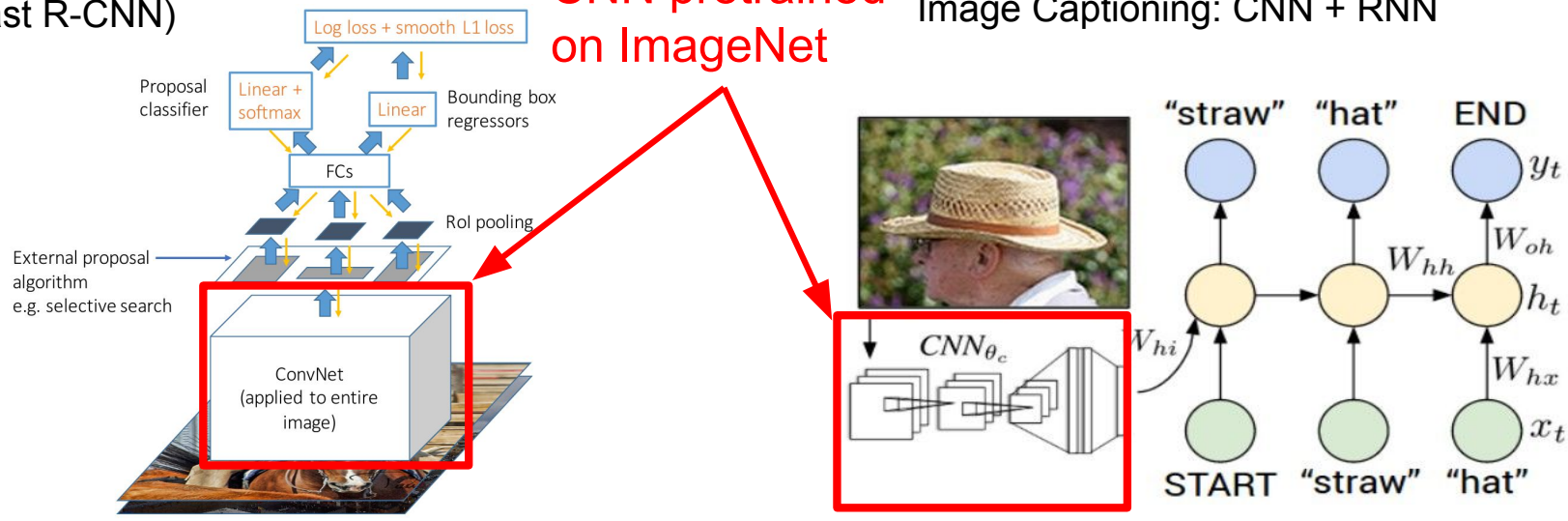
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

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

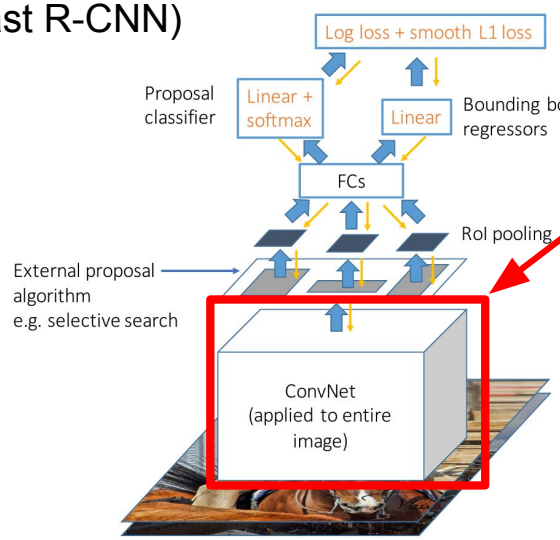


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

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

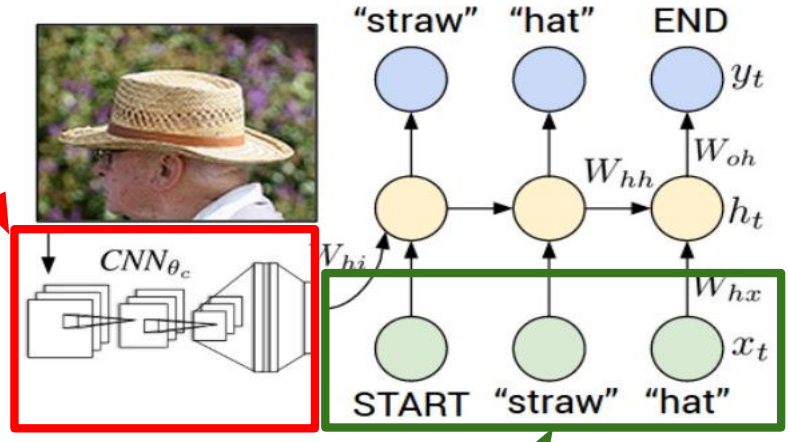
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN



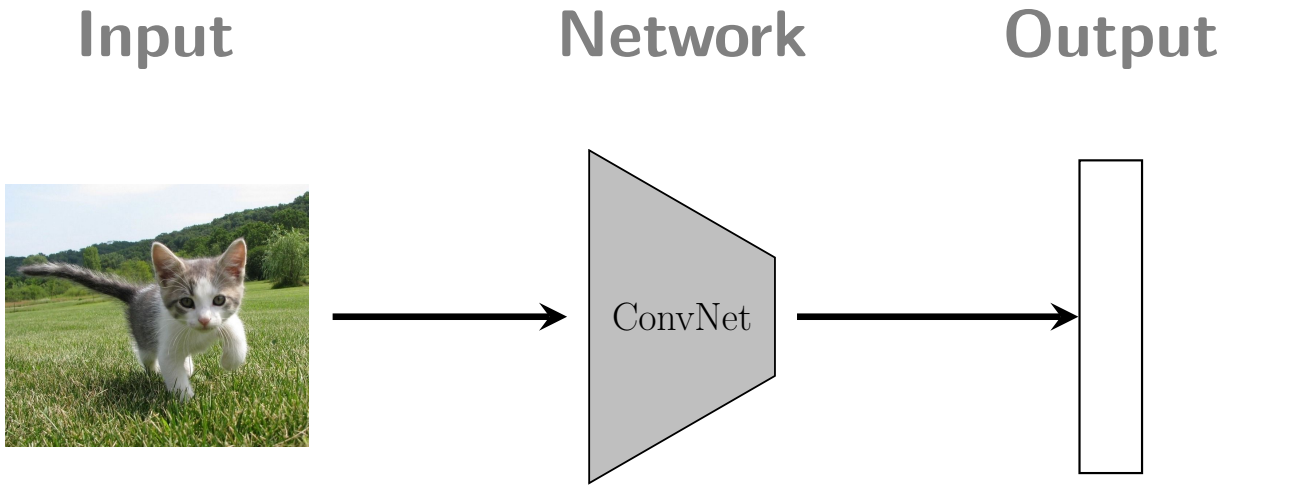
Word vectors pretrained with word2vec

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

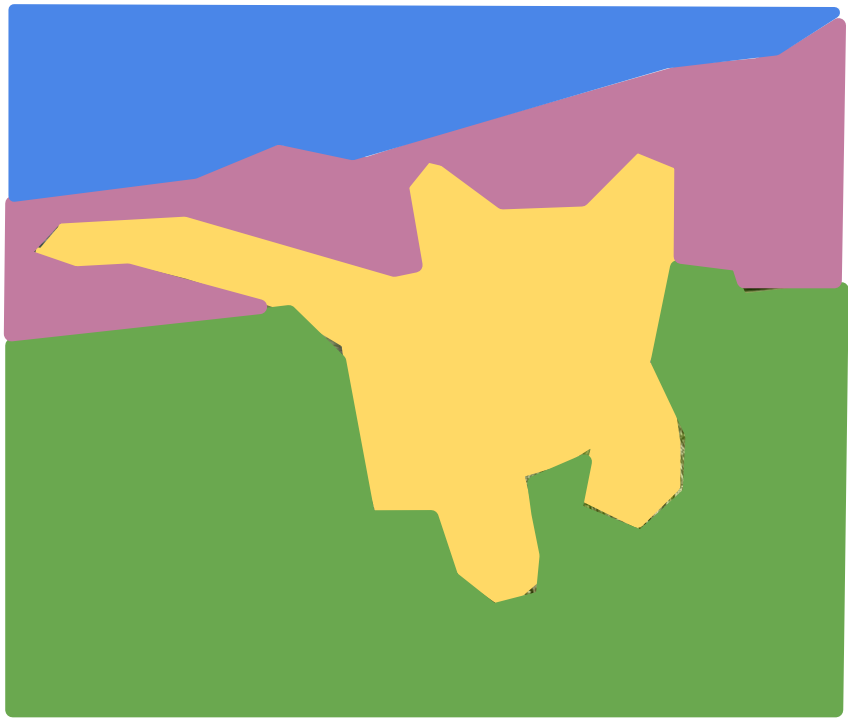
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Can we use ConvNets to solve other Computer Vision tasks??

So far: Image Classification



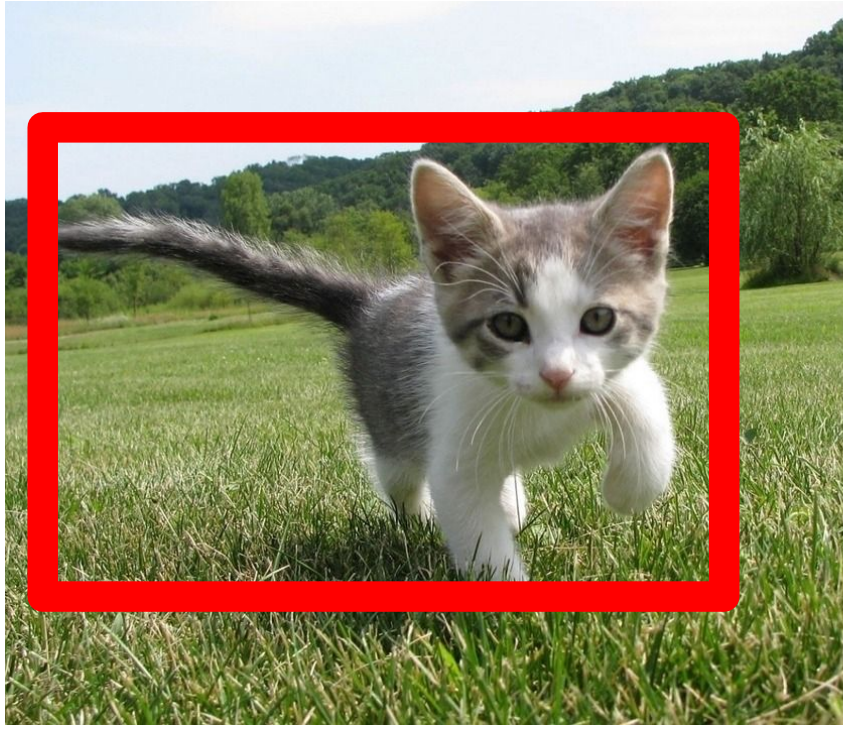
Semantic Segmentation



Grass, **Cat**, **Tree**, **Sky**

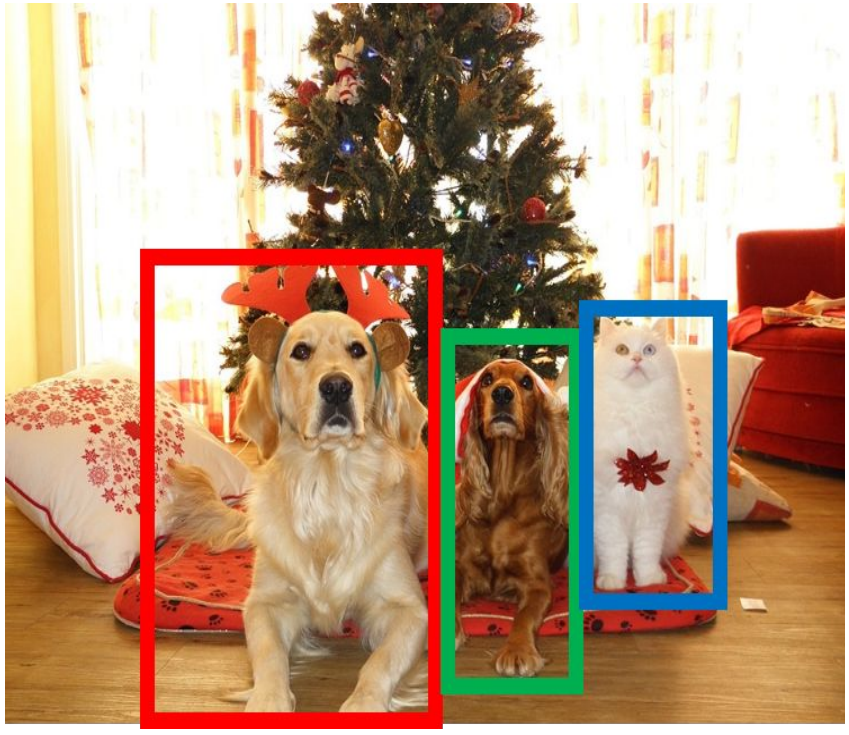
Predict a label for each pixel in the image.

Classification & Localization



Classify object in the image and where it is.

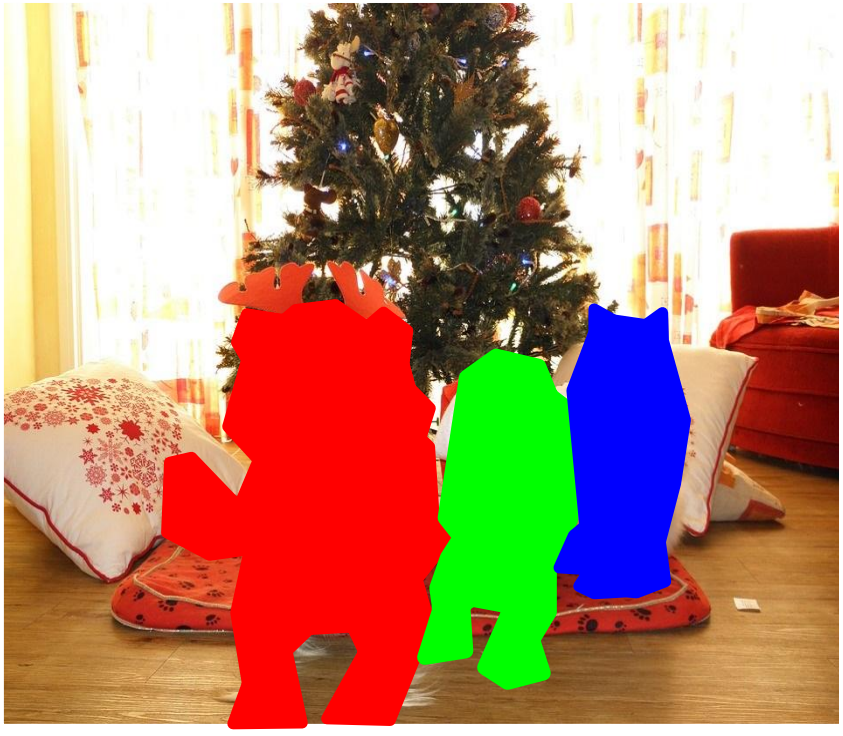
Object Detection



Dog, **Dog**, **Cat**

Classify objects and detect where they are.

Instance Segmentation



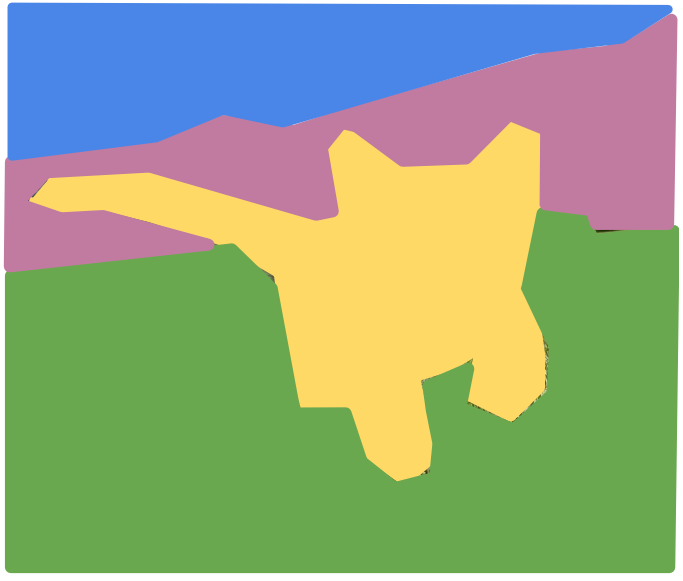
Dog, Dog, Cat

Identify the pixels belonging to each detected object.

Focus on Semantic Segmentation

Input

Output

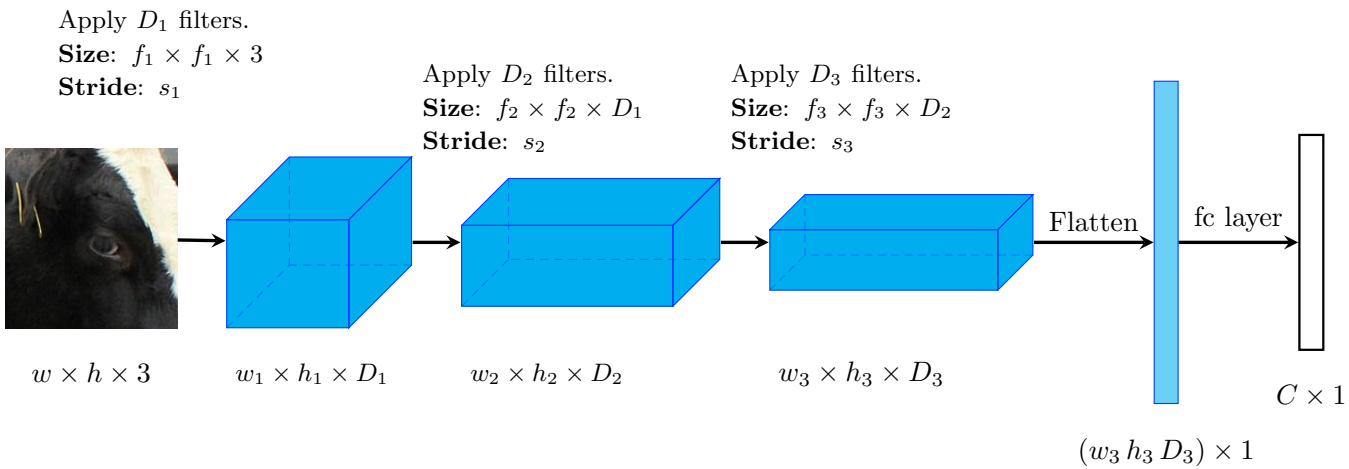


Grass, **Cat**, **Tree**, **Sky**

- Predict a label for each pixel in the image.
- Don't differentiate instances, only care about pixels.

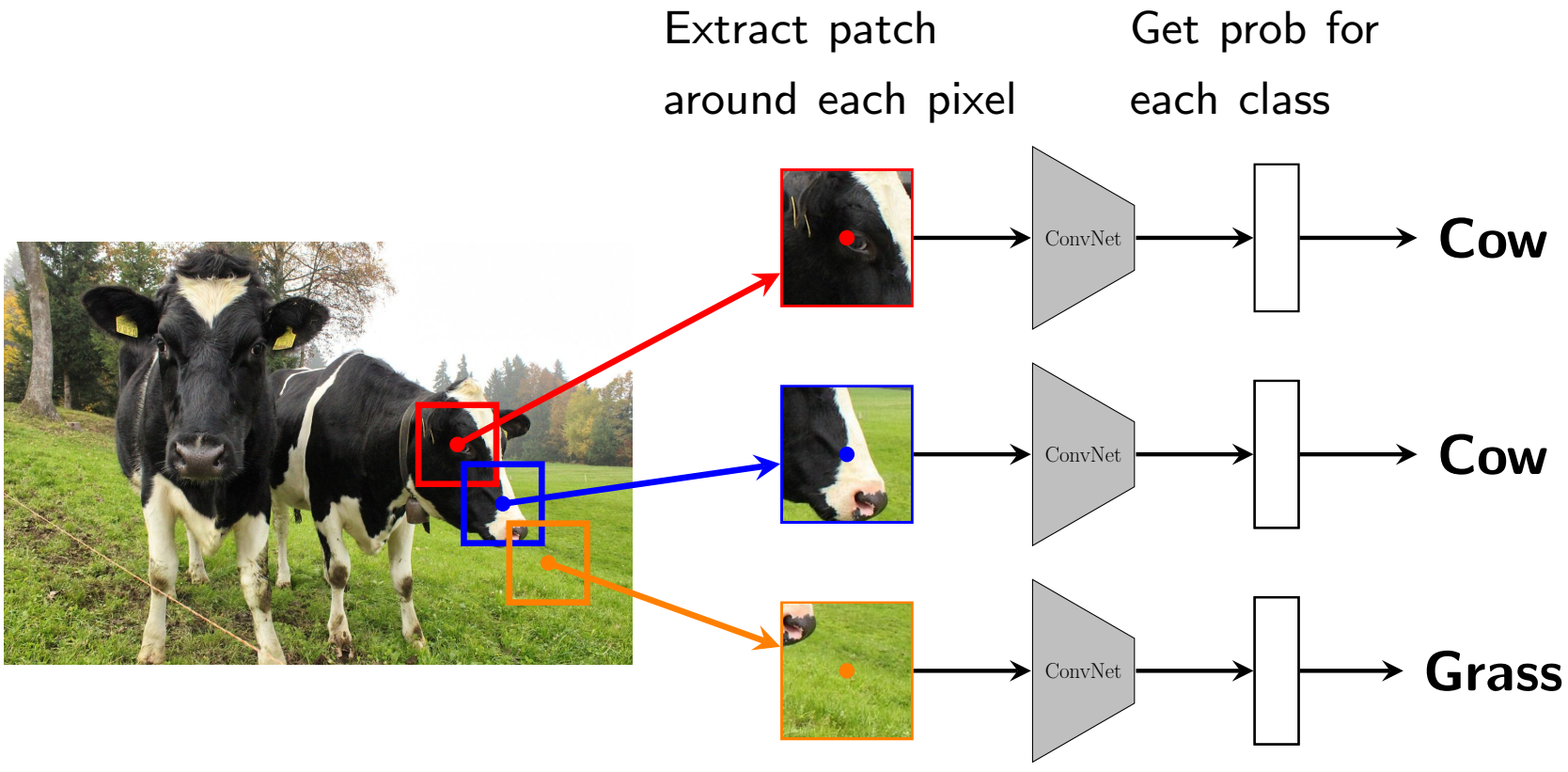
Sliding Window Approach to Semantic Segmentation

Assume have a classification network

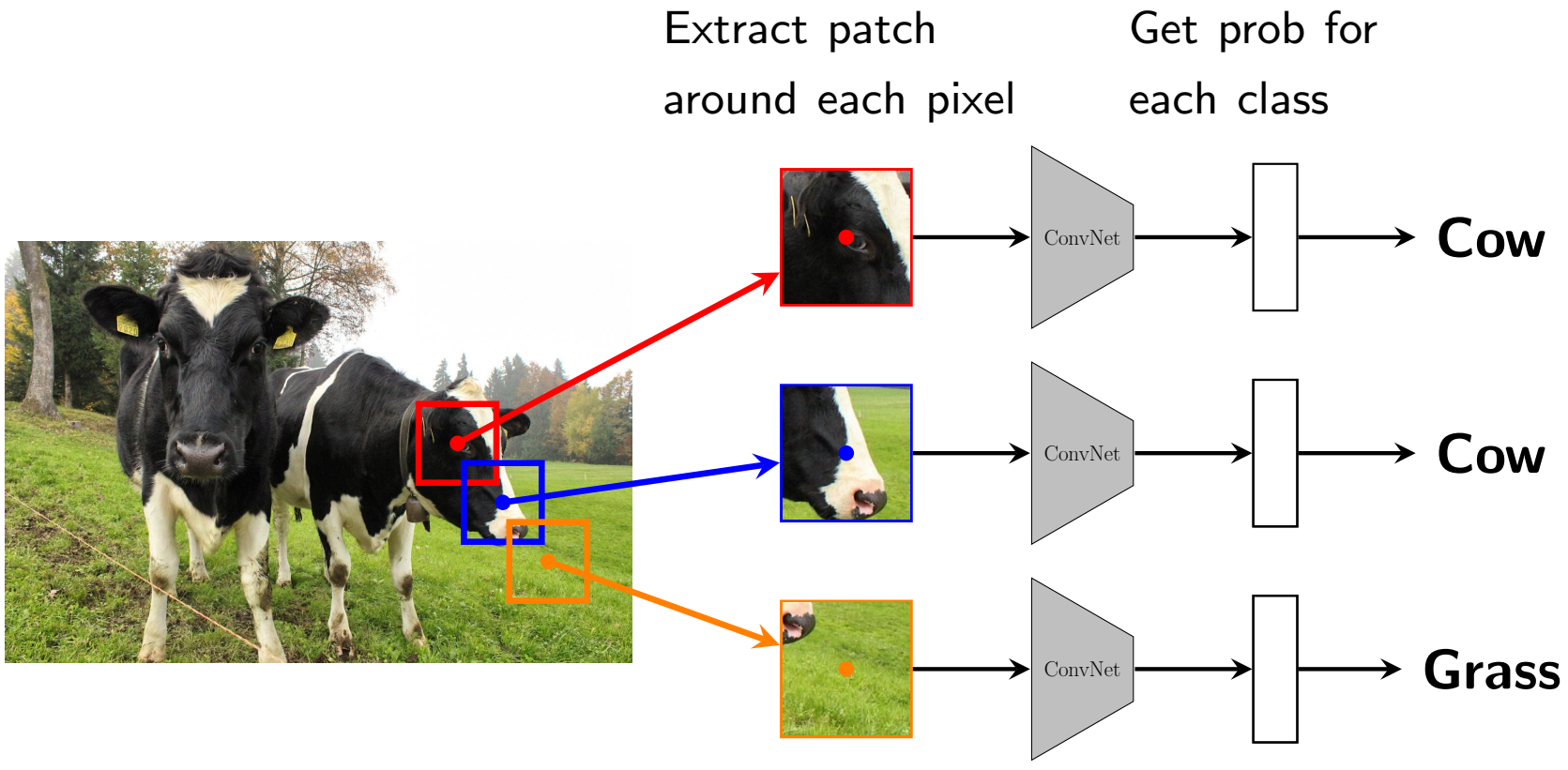


- Have a network with convolutional layers and one fully connected layer.
- Network only works for input images of size $w \times h \times 3$ because of fully-connected layer.

Sliding Window Approach to Semantic Segmentation

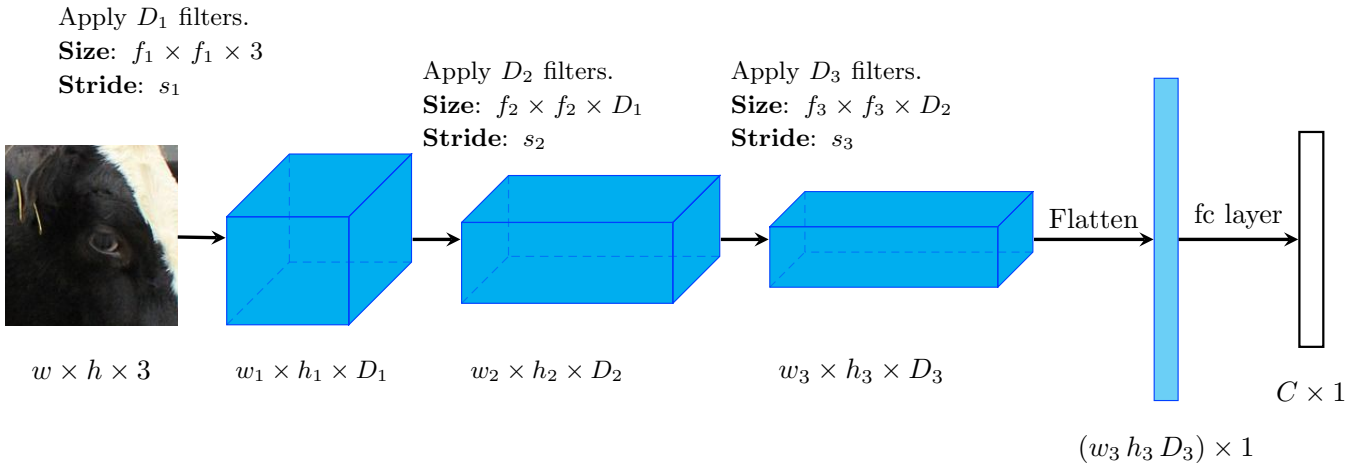


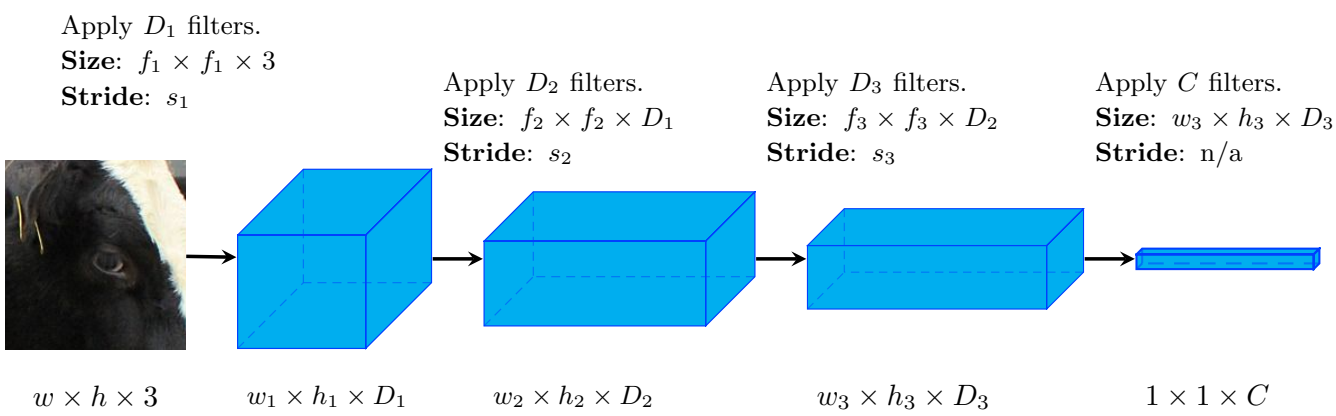
Sliding Window Approach to Semantic Segmentation



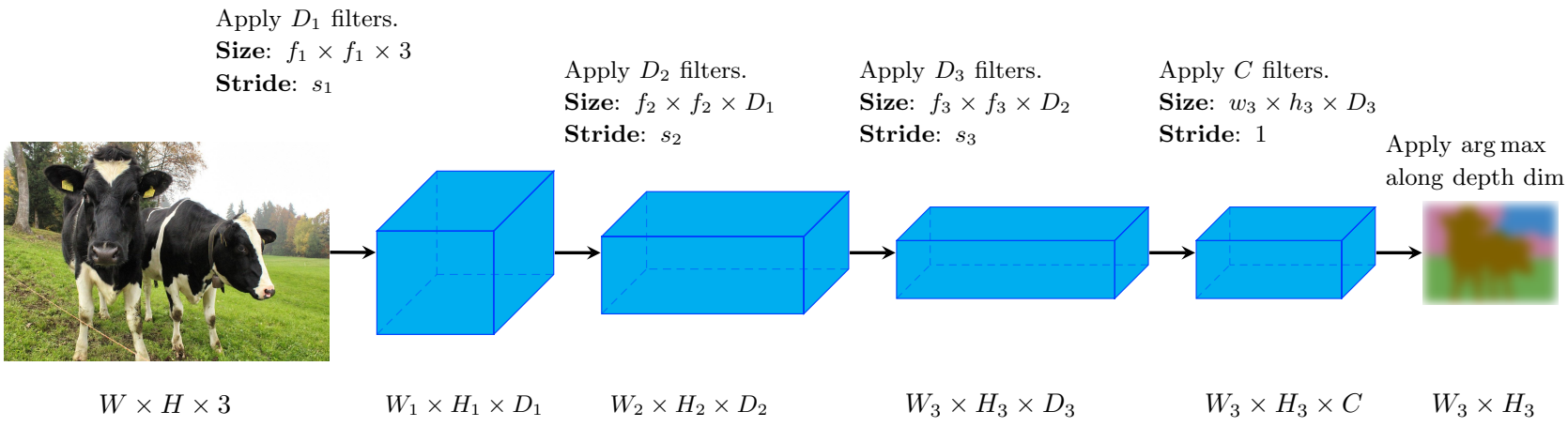
Problem: Very inefficient! Recomputing the same features multiple times between overlapping patches.

Idea: Replace fully connected layer with convolutional layer



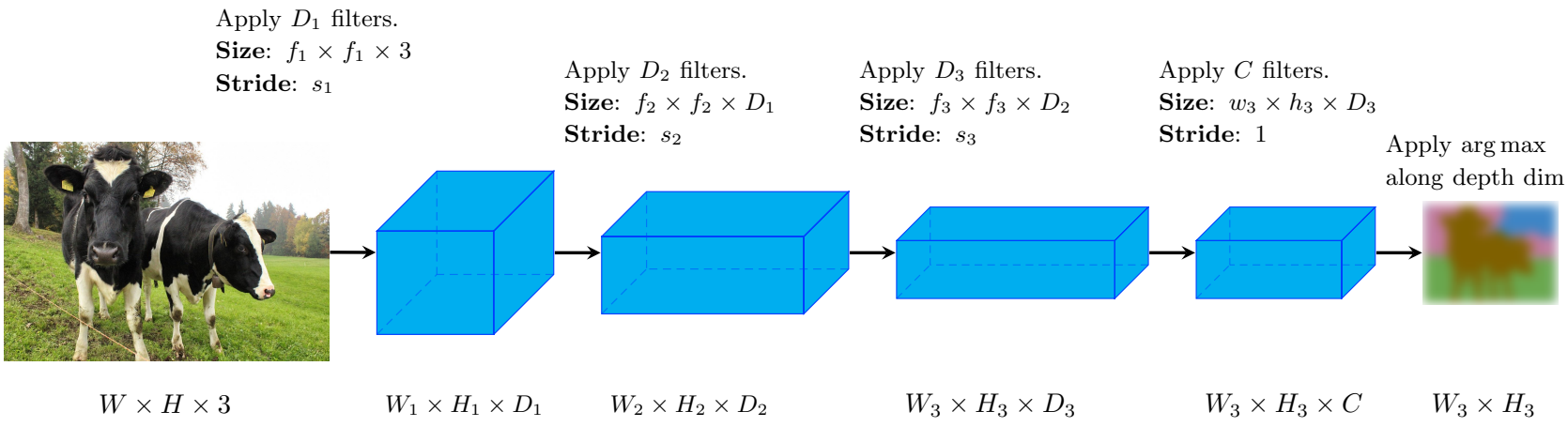


Replace fully connected layer with convolutional layer



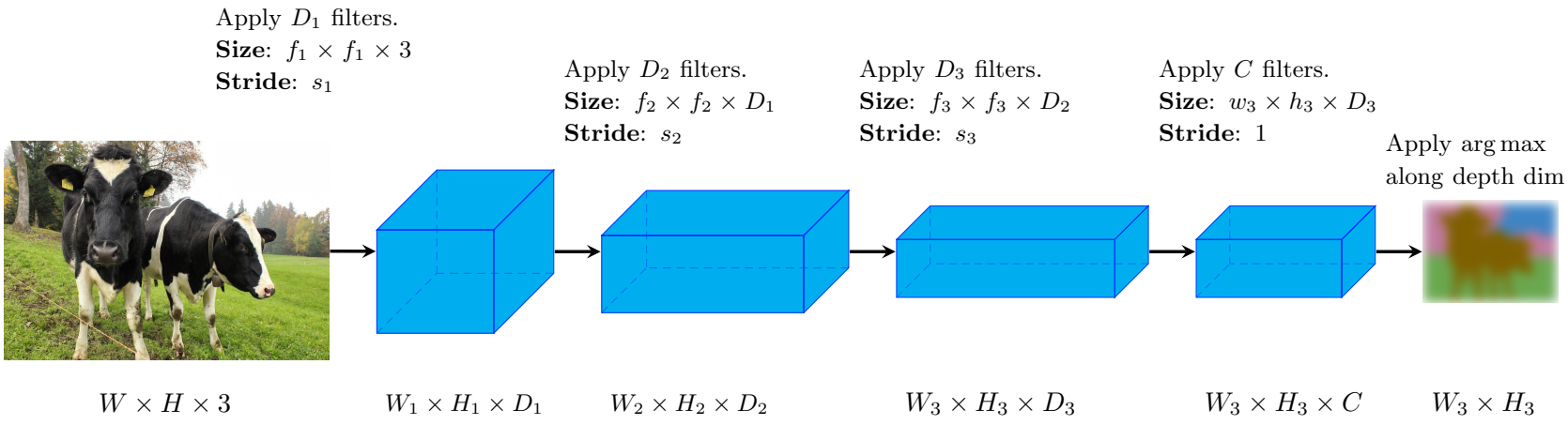
- Can now input any sized image into the network and get C predictions at multiple locations...
- **Problem**: Only have predictions for a subset of the original pixels!

Replace fully connected layer with convolutional layer

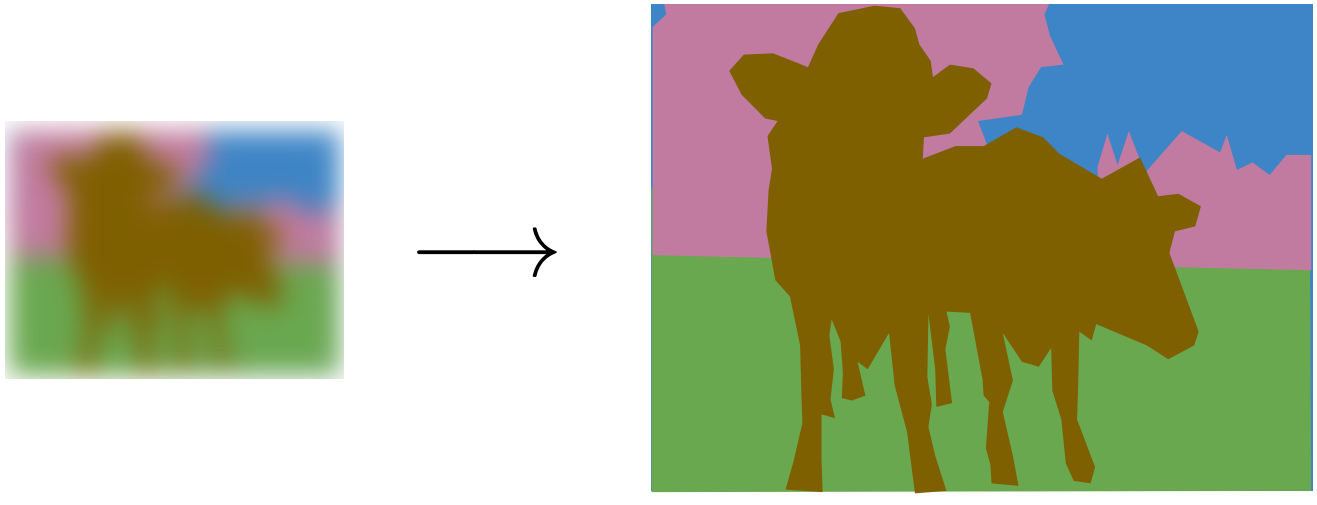


- Can now input any sized image into the network and get C predictions at multiple locations...
- **Problem**: Only have predictions for a subset of the original pixels!

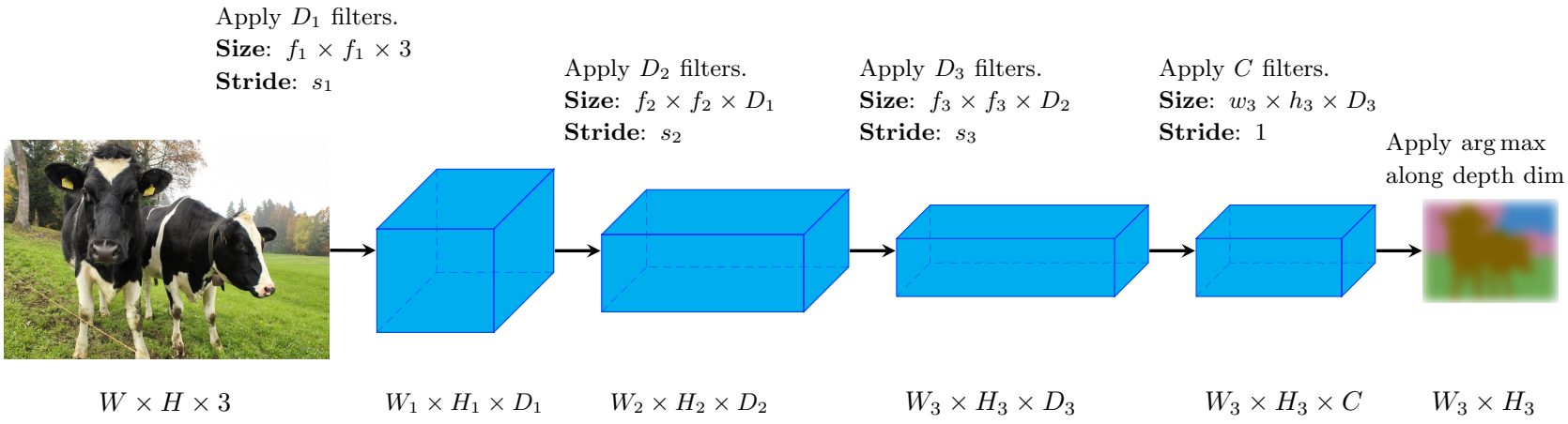
How can we get a high-resolution segmentation map?



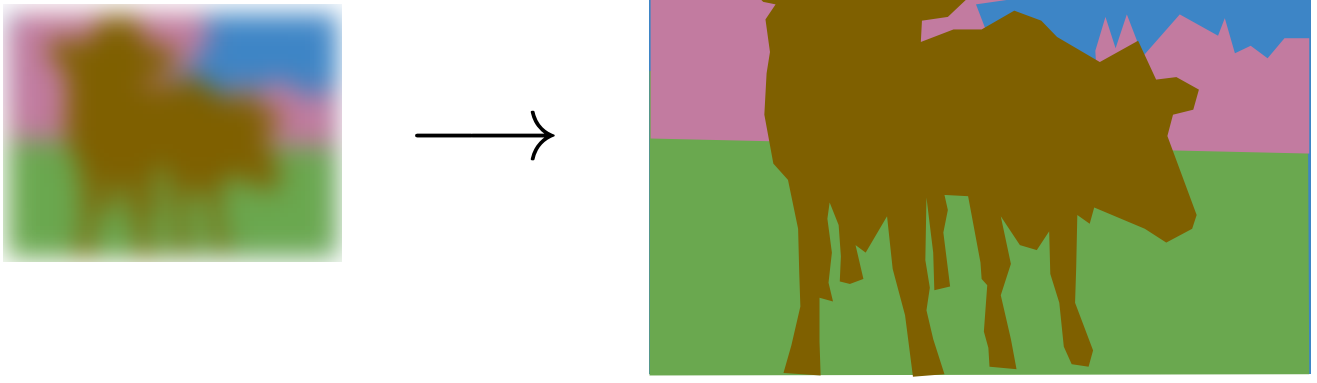
Is there some magic interpolation?



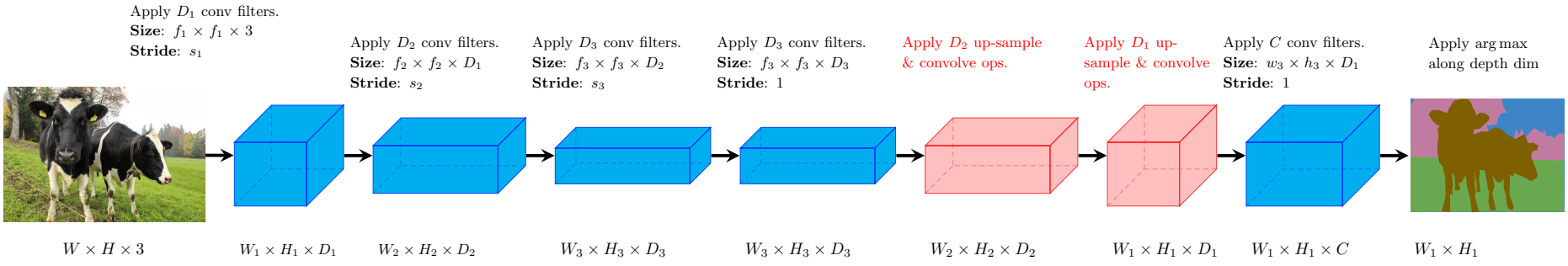
How can we get a high-resolution segmentation map?



Is there some magic interpolation? Nearest neighbour, bilinear, etc. interpolation is probably not going to do the job!

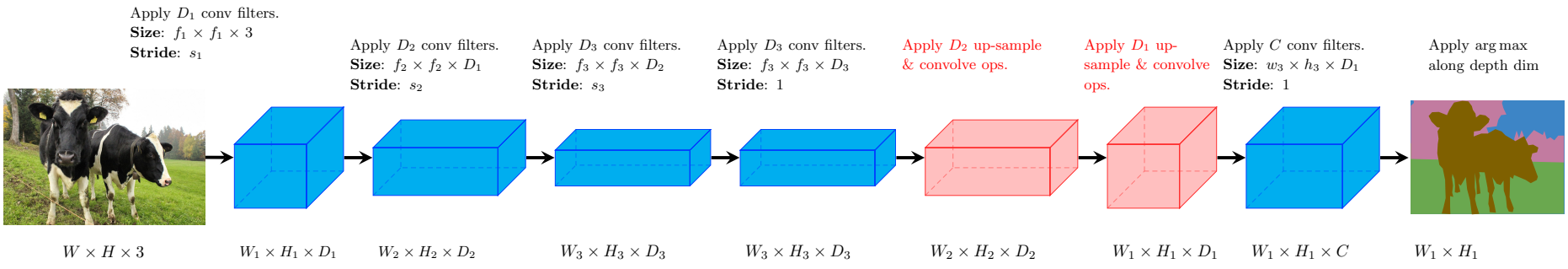


Idea: Learn how to up-sampling within the network



What is the up-sampling combined with convolution operation?

Idea: Learn how to up-sampling within the network



What is the up-sampling combined with convolution operation?

Transposed convolution

Learnable Upsampling: Transposed Convolution

Introduce this concept via our simple example

Simple Example

- Have an input image X of size 4×4.
- Have a filter F of size 3×3.
- Set the stride $s = 1$ and zero-padding $P = 0$.
- Convolve X by F gives a response map of size 2×2

$$S = X * F$$

- Each entry of S can be written as

$$S_{lm} = \sum_{i=1}^{3} \sum_{j=1}^{3} X_{l+i-1,m+j-1} F_{ij}$$

Can write convolution as a matrix multiplication

Write this convolution as a matrix multiplication involving $\text{vec}(X)$

$$S = \begin{pmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{pmatrix} * \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix}$$

Solution:

$$\begin{pmatrix} S_{11} \\ S_{12} \\ S_{21} \\ S_{22} \end{pmatrix} = \underbrace{\begin{pmatrix} \overbrace{F_{11} & F_{12} & F_{13} & 0}^{\text{row 1 of } X} & \overbrace{F_{21} & F_{22} & F_{23} & 0}^{\text{row 2 of } X} & \overbrace{F_{31} & F_{32} & F_{33} & 0}^{\text{row 3 of } X} & \overbrace{0 & 0 & 0 & 0}^{\text{row 4 of } X} \\ 0 & F_{11} & F_{12} & F_{13} & 0 & F_{21} & F_{22} & F_{23} & 0 & F_{31} & F_{32} & F_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & F_{11} & F_{12} & F_{13} & 0 & F_{21} & F_{22} & F_{23} & 0 & F_{31} & F_{32} & F_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 & F_{11} & F_{12} & F_{13} & 0 & F_{21} & F_{22} & F_{23} & 0 & F_{31} & F_{32} & F_{33} \end{pmatrix}}_{M_F^{\text{filter}}} \text{vec}(X)$$

Thus

$$\text{vec}(S) = M_F^{\text{filter}} \, \text{vec}(X)$$

where M_F^{filter} is 4×16.

Transposed Convolution defined by F, stride 1, zero-pad 0

- Given a response map, $\tilde{S}$, of size 2×2.
- **Transposed convolution** of $\tilde{S}$ by F is given by

$$(M_F^{\text{filter}})^T \ \text{vec}(\tilde{S})$$

- Let $\tilde{\mathbf{s}} = \text{vec}(\tilde{S})$ and

$$\tilde{\mathbf{x}} = (M_F^{\text{filter}})^T \ \tilde{\mathbf{s}}$$

- What is the size of $\tilde{\mathbf{x}}$?
- What does $\tilde{X}$ correspond to if $\text{vec}(\tilde{X}) = \tilde{\mathbf{x}}$?

Answers

Have:

$$\tilde{\mathbf{x}} = \left(M_F^{\text{filter}}\right)^T \tilde{\mathbf{s}}$$

- **Q1**: M_F^{filter} is $4 \times 16 \implies \left(M_F^{\text{filter}}\right)^T$ is $16 \times 4 \implies \tilde{\mathbf{x}}$ is 16×1
- **Q2**:

$$\text{vec}(\tilde{X}) = \begin{pmatrix} F_{11} & 0 & 0 & 0 \\ F_{12} & F_{11} & 0 & 0 \\ F_{13} & F_{12} & 0 & 0 \\ 0 & F_{13} & 0 & 0 \\ F_{21} & 0 & F_{11} & 0 \\ F_{22} & F_{21} & F_{12} & F_{11} \\ F_{23} & F_{22} & F_{13} & F_{12} \\ 0 & F_{23} & 0 & F_{13} \\ F_{31} & 0 & F_{21} & 0 \\ F_{32} & F_{31} & F_{22} & F_{21} \\ F_{33} & F_{32} & F_{23} & F_{22} \\ 0 & F_{33} & 0 & F_{23} \\ 0 & 0 & F_{31} & 0 \\ 0 & 0 & F_{32} & F_{31} \\ 0 & 0 & F_{33} & F_{32} \\ 0 & 0 & 0 & F_{33} \end{pmatrix} \text{vec}(\tilde{S}) \implies \tilde{X} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \tilde{S}_{11} & \tilde{S}_{12} & 0 \\ 0 & 0 & \tilde{S}_{21} & \tilde{S}_{21} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} F_{33} & F_{32} & F_{31} \\ F_{23} & F_{22} & F_{21} \\ F_{13} & F_{12} & F_{11} \end{pmatrix}$$

- Therefore **transposed convolution** of $\tilde{S}$ by a filter F applied with zero-padding 0 and stride 1 is:

$$\tilde{X} = \tilde{S}_{\text{zero-pad}} * F_{\text{rot180}}$$

where the zero-padding of $\tilde{S}$ is $3 - 1 = 2$.

Transposed Convolution defined by F, stride 2, zero-pad 0?

- Define M_F^{filter} by applying a filter F of size 2×2 with stride 2 and no zero-padding.

$$S = \begin{pmatrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{pmatrix} * \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}$$

- Then S has size 2×2 and M_F^{filter} is

$$\begin{pmatrix} F_{11} & F_{12} & 0 & 0 & F_{21} & F_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & F_{11} & F_{12} & 0 & 0 & F_{21} & F_{22} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & F_{11} & F_{12} & 0 & 0 & F_{21} & F_{22} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & F_{11} & F_{12} & 0 & 0 & F_{21} & F_{22} \end{pmatrix}$$

- Apply transposed convolution operator to $(M_F^{\text{filter}})^T$ to the 2×2 array $\tilde{S}$.

- Then do we get the same relationship as before? that is

$$\text{vec}(\tilde{X}) = \left(M_F^{\text{filter}}\right)^T \text{vec}(\tilde{S}) \implies \tilde{X} = \tilde{S}_{\text{zero-pad}} * F_{\text{rot180}}$$

Transposed Convolution defined by F, stride 2, zero-pad 0?

- Not quite instead we get:

$$\text{vec}(\tilde{X}) = \begin{pmatrix} F_{11} & 0 & 0 & 0 \\ F_{12} & 0 & 0 & 0 \\ 0 & F_{11} & 0 & 0 \\ 0 & F_{12} & 0 & 0 \\ F_{21} & 0 & 0 & 0 \\ F_{22} & 0 & 0 & 0 \\ 0 & F_{21} & 0 & 0 \\ 0 & F_{22} & 0 & 0 \\ 0 & 0 & F_{11} & 0 \\ 0 & 0 & F_{12} & 0 \\ 0 & 0 & 0 & F_{11} \\ 0 & 0 & 0 & F_{12} \\ 0 & 0 & F_{21} & 0 \\ 0 & 0 & F_{22} & 0 \\ 0 & 0 & 0 & F_{21} \\ 0 & 0 & 0 & F_{22} \end{pmatrix} \text{vec}(\tilde{S}) \implies \tilde{X} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \tilde{S}_{11} & 0 & \tilde{S}_{12} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \tilde{S}_{21} & 0 & \tilde{S}_{21} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{pmatrix}$$

- Zero-pad $\tilde{S}$ with $P = 2 - 1 = 1$ and zeros are inserted between the entries of $\tilde{S}$.

- Apply the same transformation to F as before $F \to F_{\text{rot180}}$

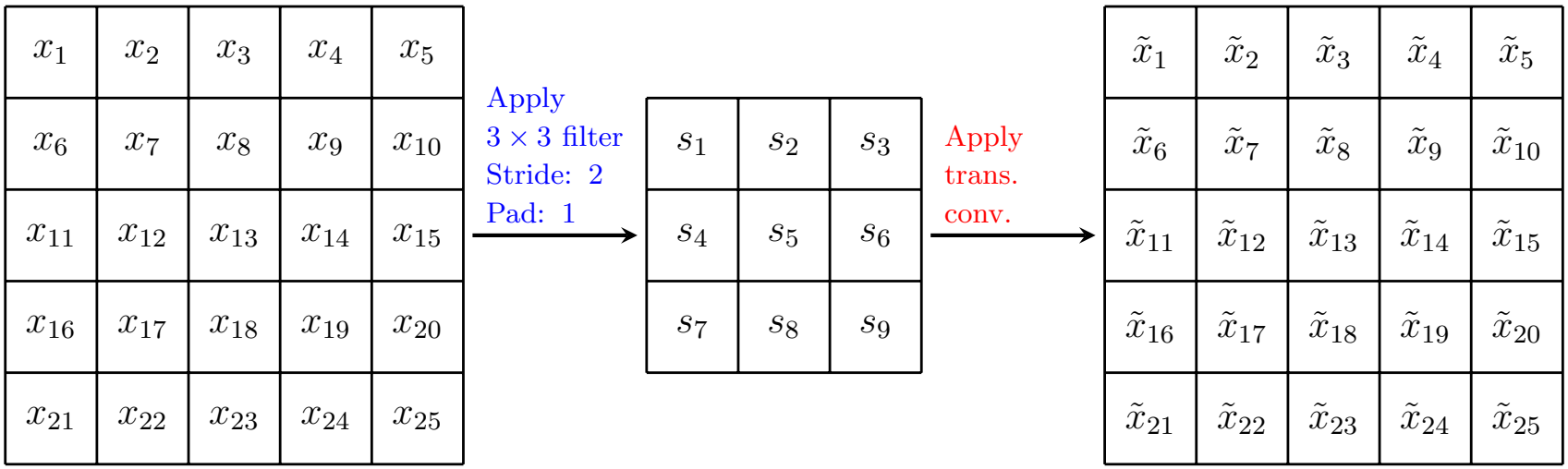
Transposed Convolution defined by F, stride s, zero-pad p?

General case:

- Create M_F^{filter} by applying
  - a convolutional filter of size $f \times f$
  - with stride s and
  - zero-padding p
  - to an input of size $w \times w$
- Then apply $(M_F^{\text{filter}})^T$ to $\tilde{S}$ of size $w' \times w'$ with $w' = (w - f - 2p)/s + 1$ to implement a transposed convolution.
- This transposed convolution is equivalent to applying a convolutional filter
  - of size $f \times f$
  - with stride $s' = 1$ and
  - zero-padding $p' = f - p - 1$

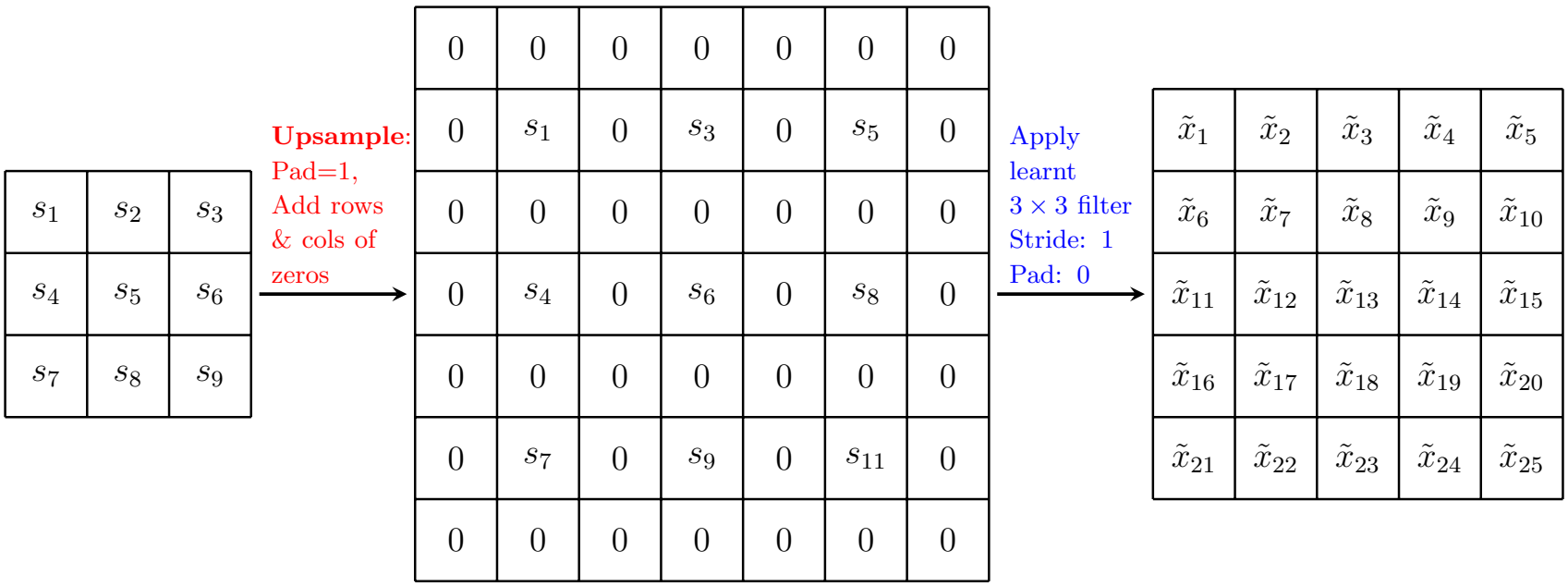
  to the new version of $\tilde{S}$ generated by adding $s - 1$ zeros between each entry in $\tilde{S}$.

Visualization of Transposed Convolution



- Convolve a 5×5 input with a 3×3 filter applied with stride 2 and zero-padding equal to 1.
- Get a 3×3 output.
- Apply a transposed convolution operation to up-sample 3×3 output to 5×5 output. How?

Visualization of Transposed Convolution

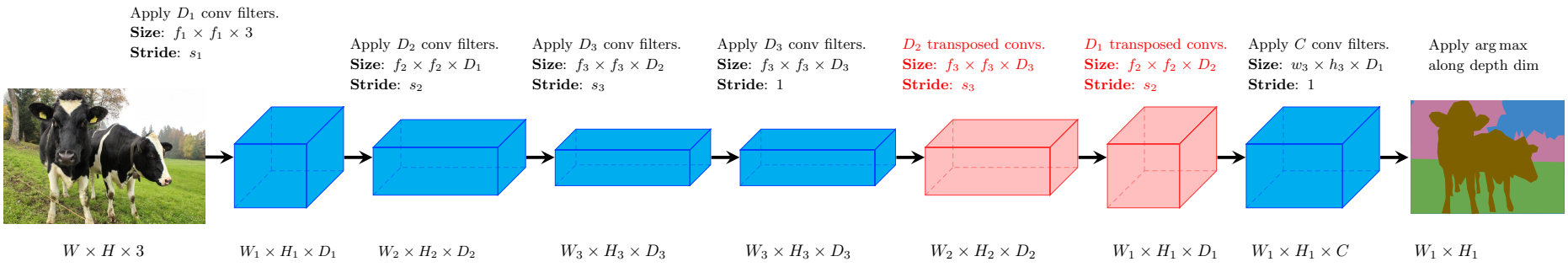


- Up-sample 3×3 output to size 7×7 by
  - Add $s - 1 = 2 - 1 = 1$ rows and columns of zeros between elements of output.
  - Zero-pad with $f - p - 1 = 3 - 1 - 1 = 1$.
- Apply a normal convolution operation to the upsampled output with a filter of size 3×3 with stride 1 and zero-pad 0.

Summary of using Transposed Convolution to up-sample

- Have a 2d array of responses X of size $w' \times h'$.
- The *transposed convolution* is a operator we can use to up-sample $\mathbf{X}$ to a larger spatial size.
- Conceptually you have a filter F of size $f \times f$
  - Insert between each entry of X $s-1$ zeros to get $\mathbf{X}_{\text{tc}}$ - (both between rows and columns).
  - $\mathbf{X}_{\text{tc}} =$ Zero-pad $\mathbf{X}_{\text{tc}}$ with $f-p-1$ zeros.
  - Convolve $\mathbf{X}_{\text{tc}}$ with F with stride 1 and zero-padding 0.
- The choice of s dictates the amount of up-sampling. Should not have $s > f$.

Include transposed convolution in segmentation network



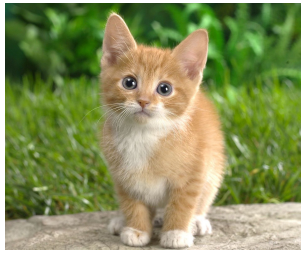
Good blog post on semantic segmentation: An overview of semantic image segmentation.

Classification + Localization: Task

Classification: C classes
Input: Image
Output: Class label
Evaluation metric: Accuracy

CAT

Localization:
Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union



(x, y, w, h)

Classification + Localization: Do both

Idea #1: Localization as Regression

Input: image

Only one object,
simpler than detection

Neural Net

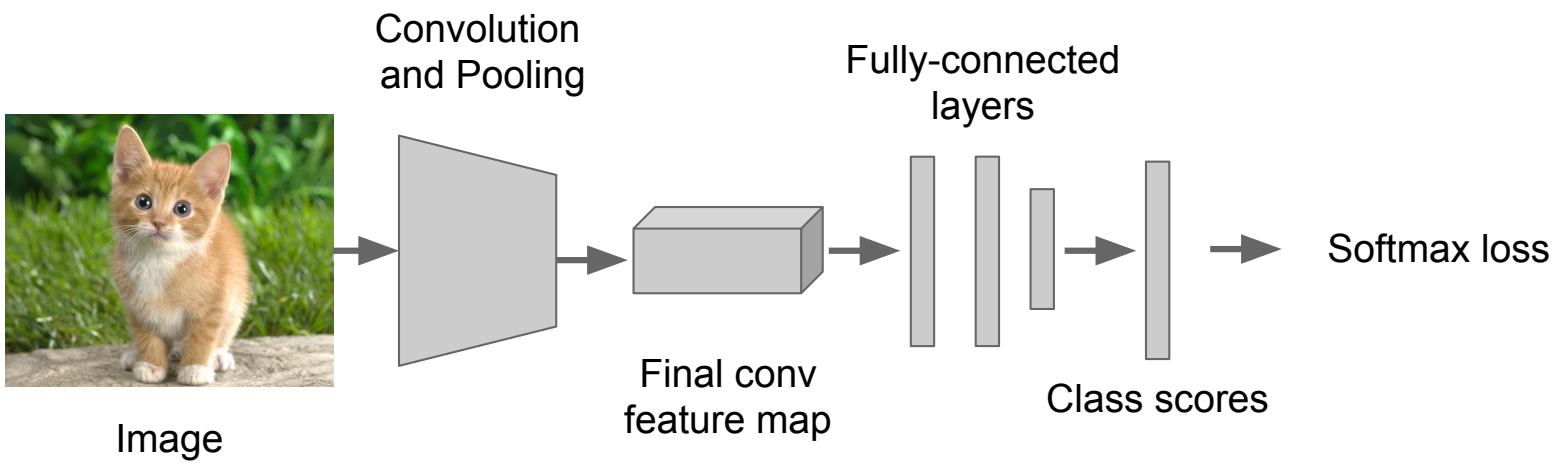
Output:
Box coordinates
(4 numbers)

Correct output:
box coordinates
(4 numbers)

Loss:
L2 distance

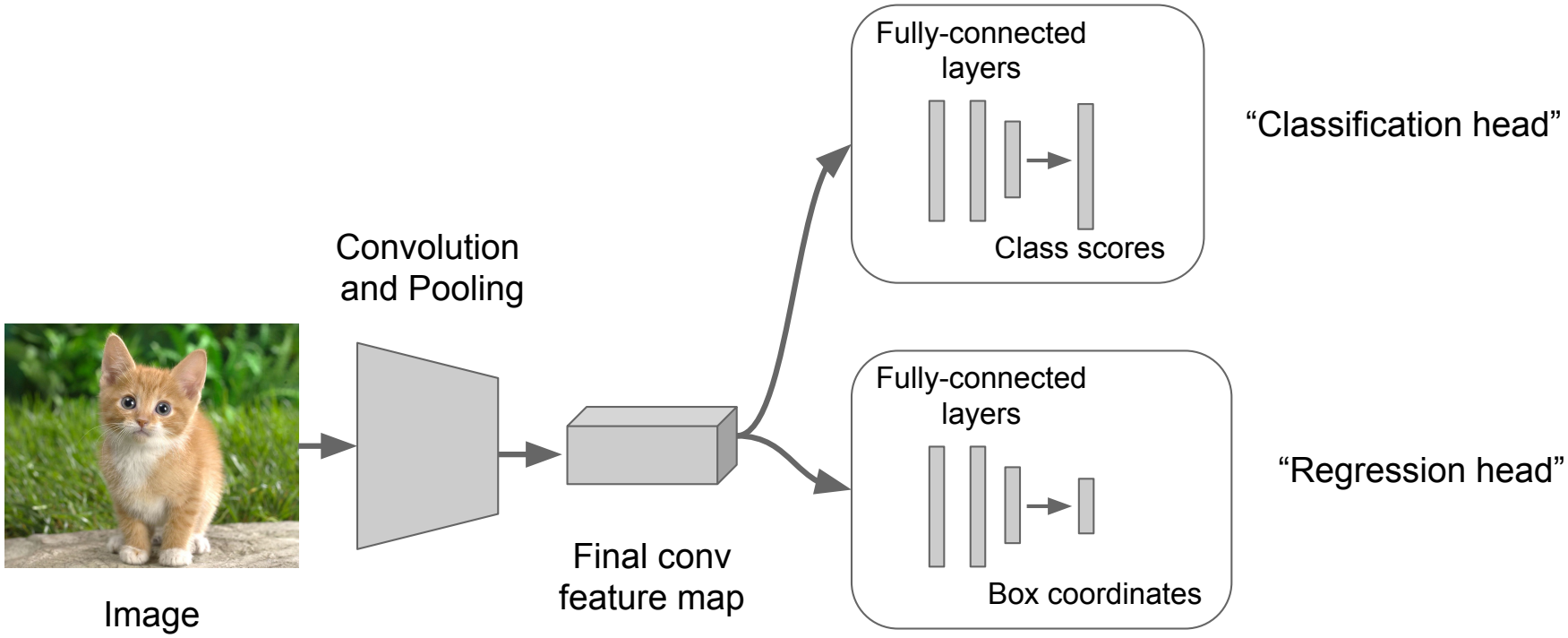
Simple Recipe for Classification + Localization

Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



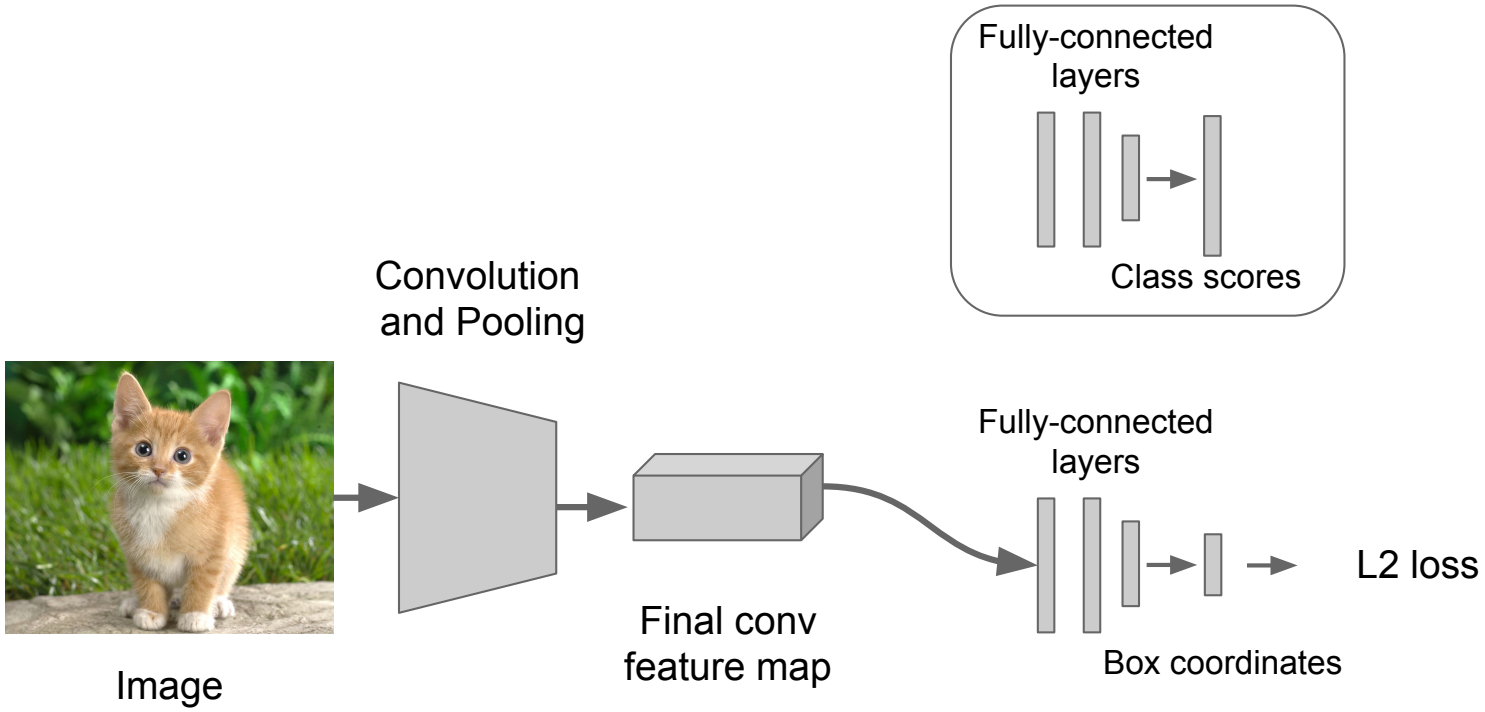
Simple Recipe for Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network



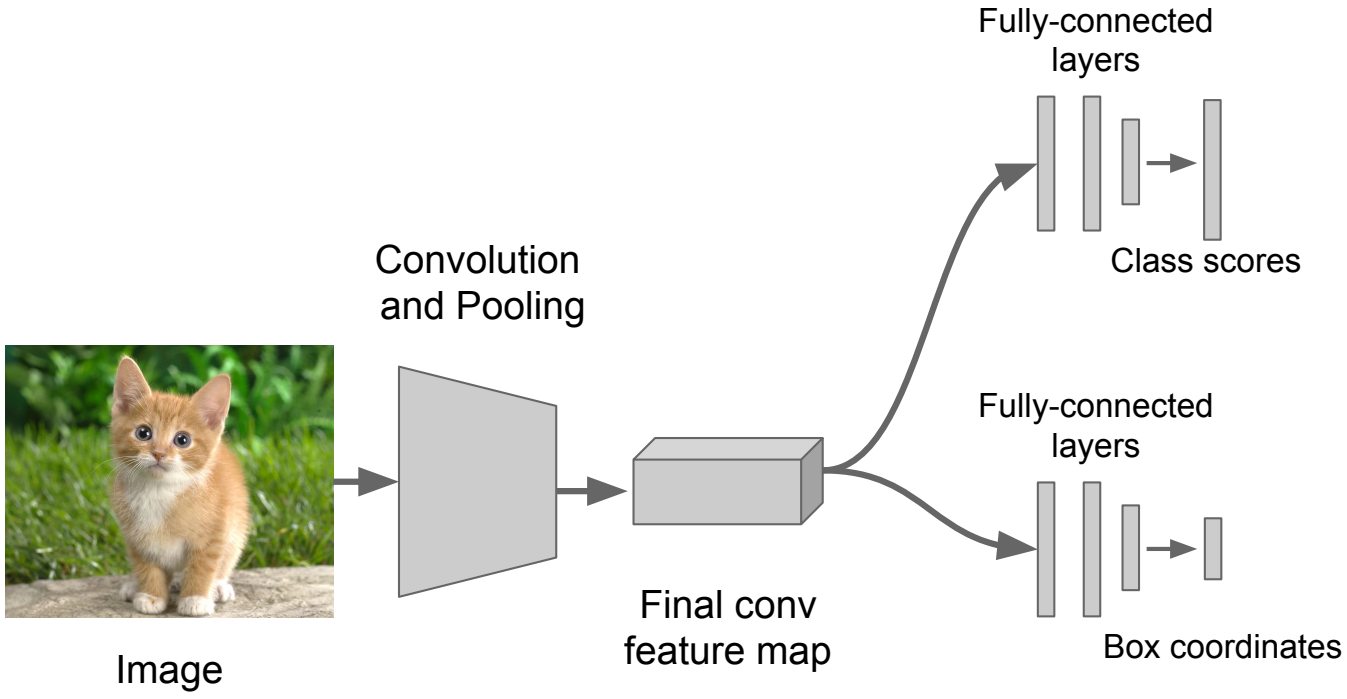
Simple Recipe for Classification + Localization

Step 3: Train the regression head only with SGD and L2 loss



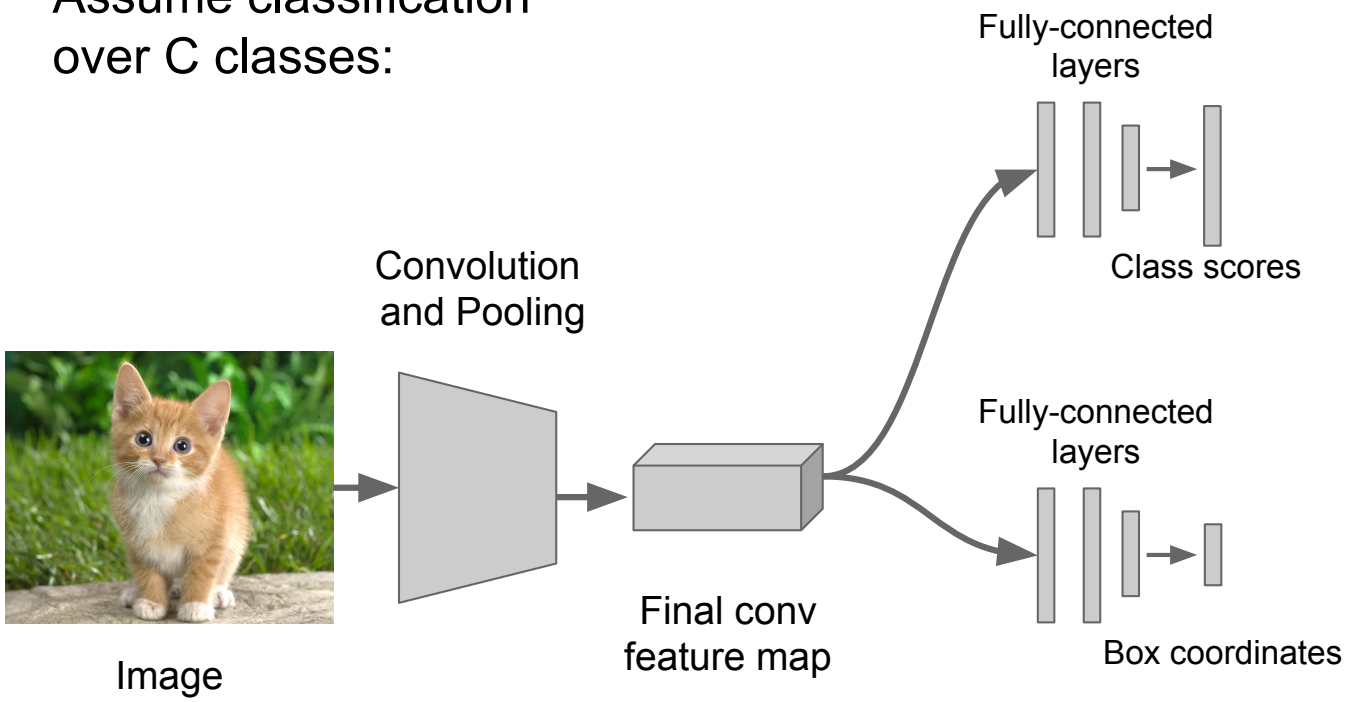
Simple Recipe for Classification + Localization

Step 4: At test time use both heads



Per-class vs class agnostic regression

Assume classification over C classes:

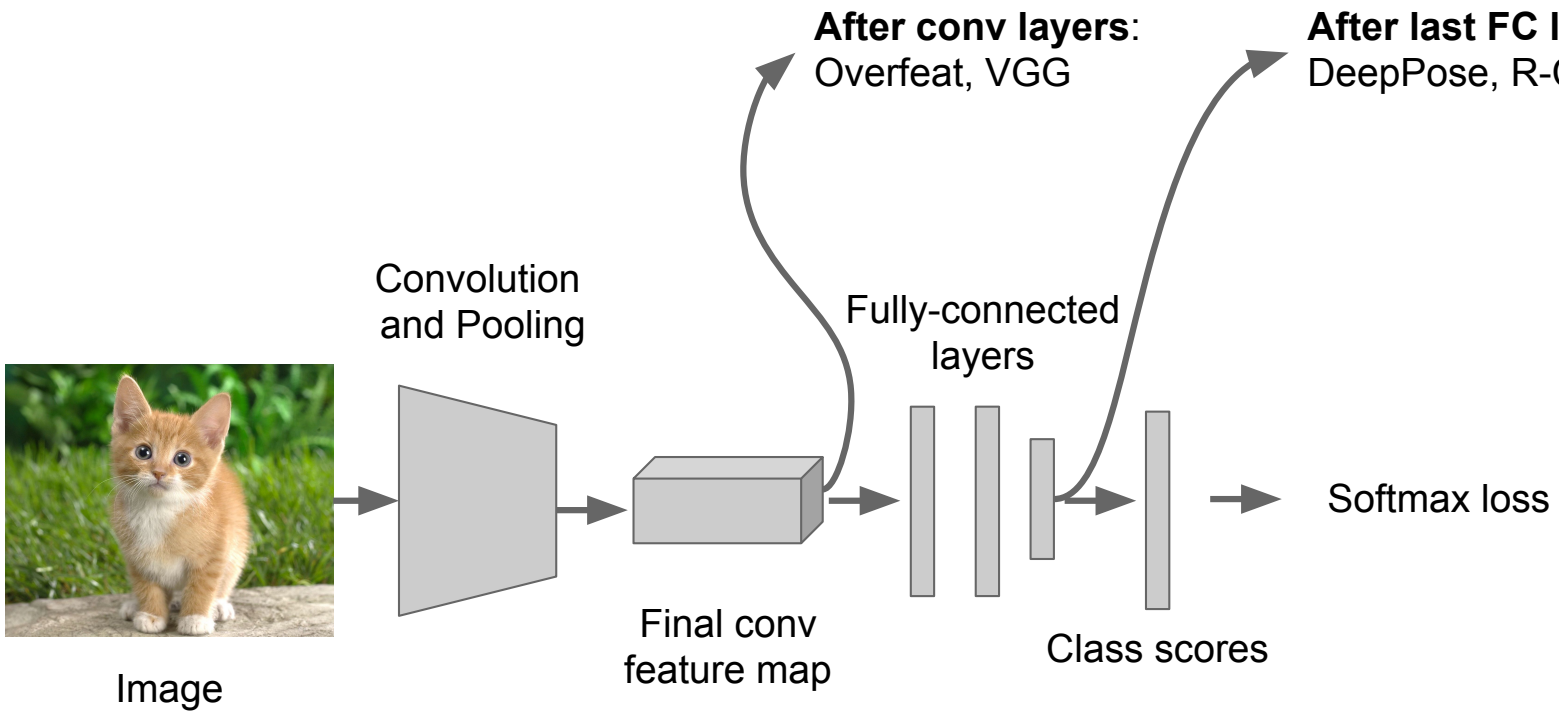


Classification head:
C numbers
(one per class)

Class agnostic:
4 numbers
(one box)

Class specific:
C x 4 numbers
(one box per class)

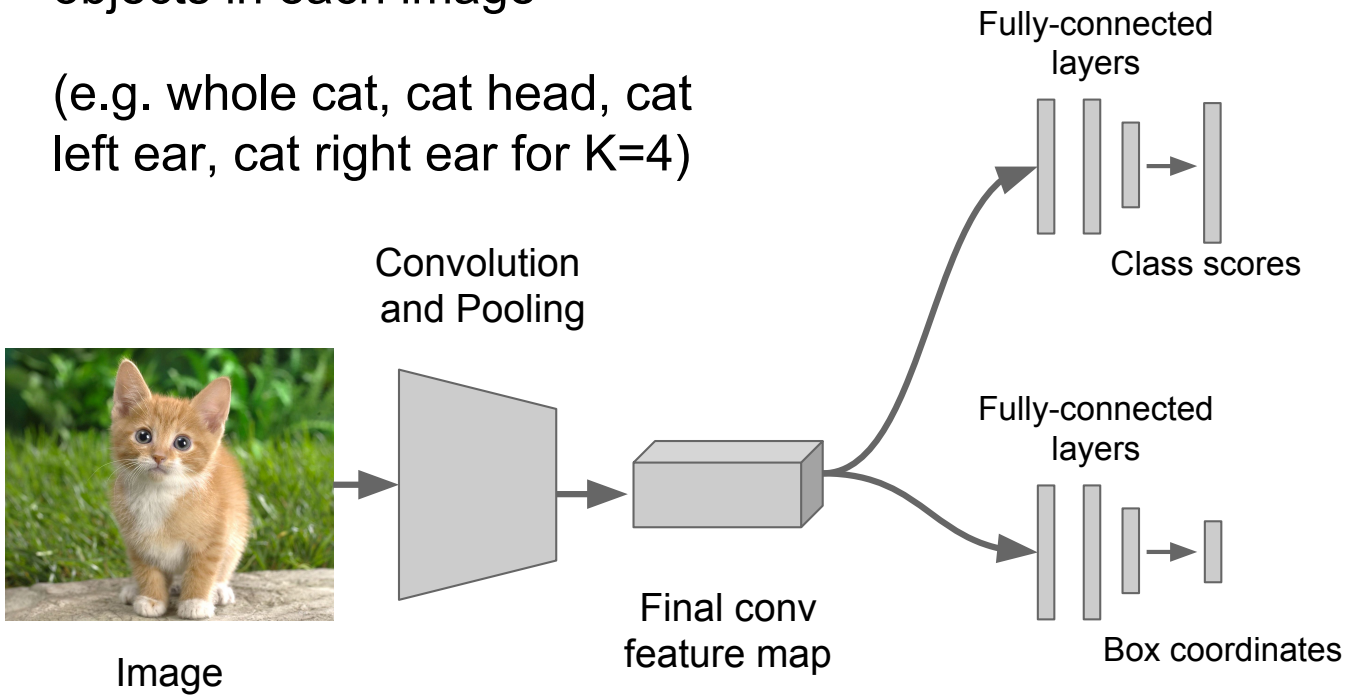
Where to attach the regression head?



Aside: Localizing multiple objects

Want to localize **exactly** K objects in each image

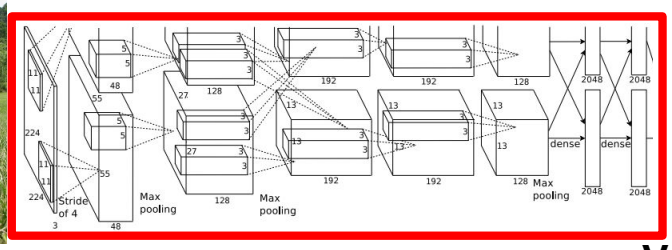
(e.g. whole cat, cat head, cat left ear, cat right ear for K=4)



K x 4 numbers
(one box per object)

Classification + Localization

This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

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

Correct label: Cat

Softmax Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

+ → **Loss**

Treat localization as a regression problem!