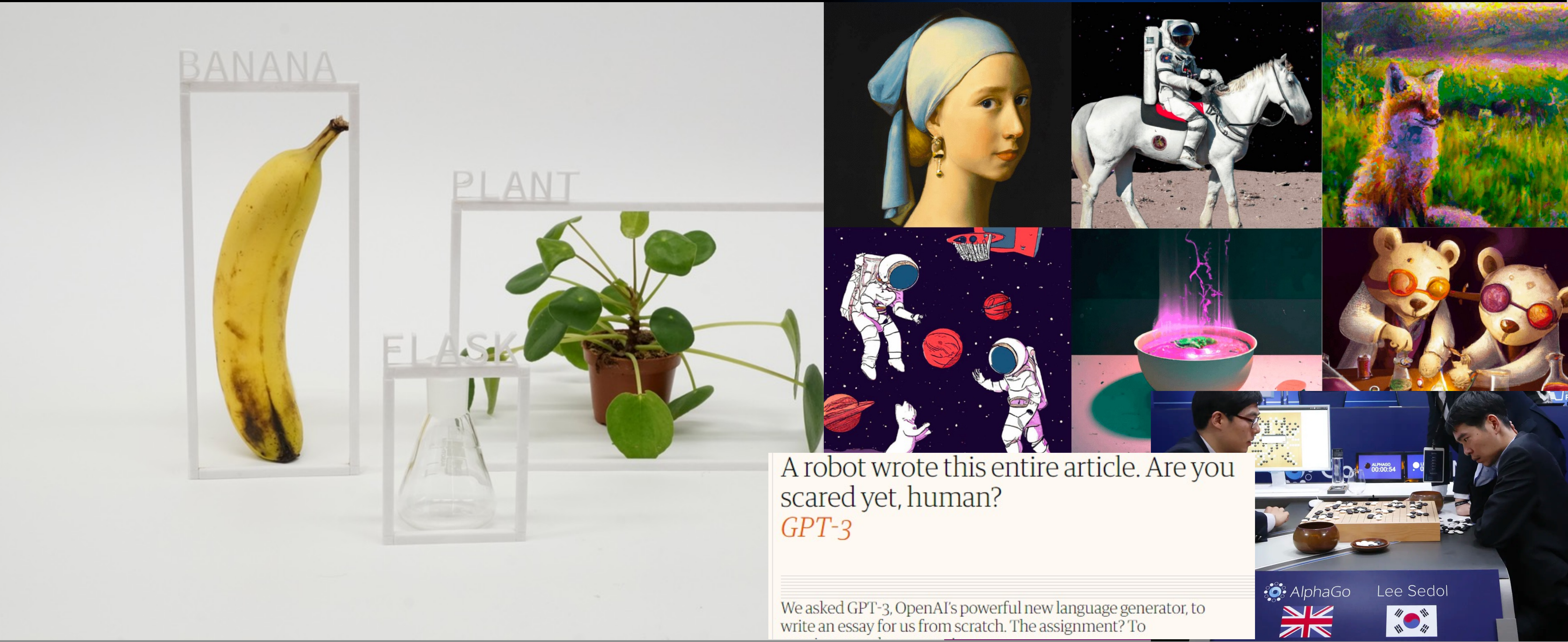


Lecture 6: Modern ConvNets

Deep Learning 1 @ UvA

Yuki M. Asano

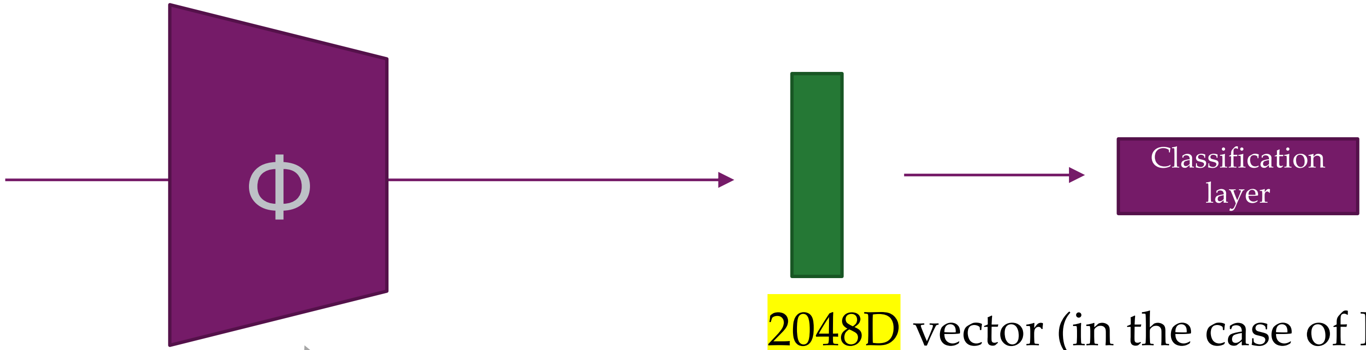
Lecture overview

- What are embeddings
- Popular Convolutional Neural Networks architectures
  - VGGNet
  - GoogLeNet
  - ResNet
  - DenseNet
  - MobileNet
  - BagNet
- R-CNN and friends
- Fully Convolutional Networks

Understanding deep embeddings

- The ”mental model”:
  - Dimensionality reduction:

3x224x224 = 150K dimensional input

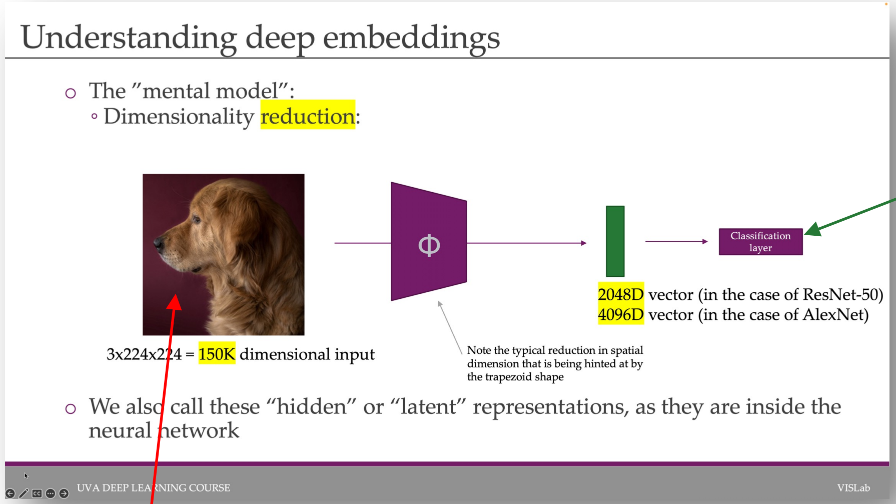


2048D vector (in the case of ResNet-50)
4096D vector (in the case of AlexNet)

Note the typical reduction in spatial dimension that is being hinted at by the trapezoid shape

- We also call these “hidden” or “latent” representations, as they are inside the neural network

The deep layers will gradually learn more abstract features.

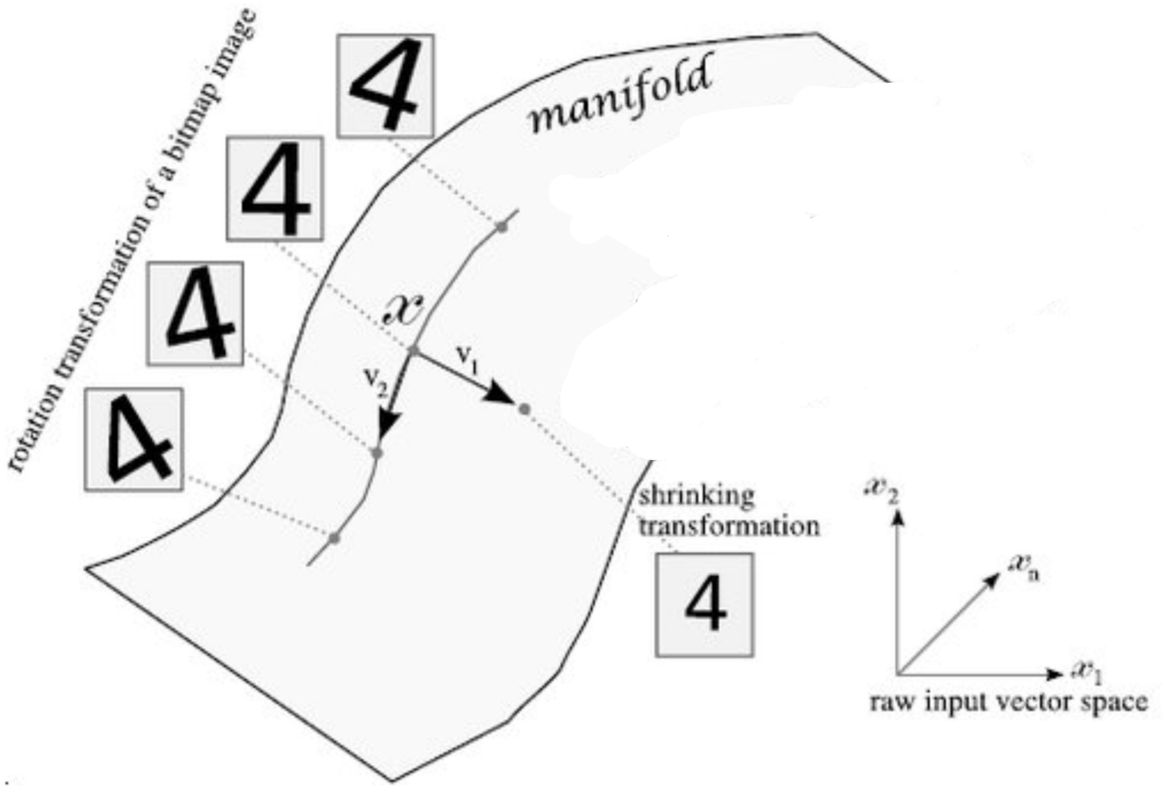


This outputs for example class probabilities

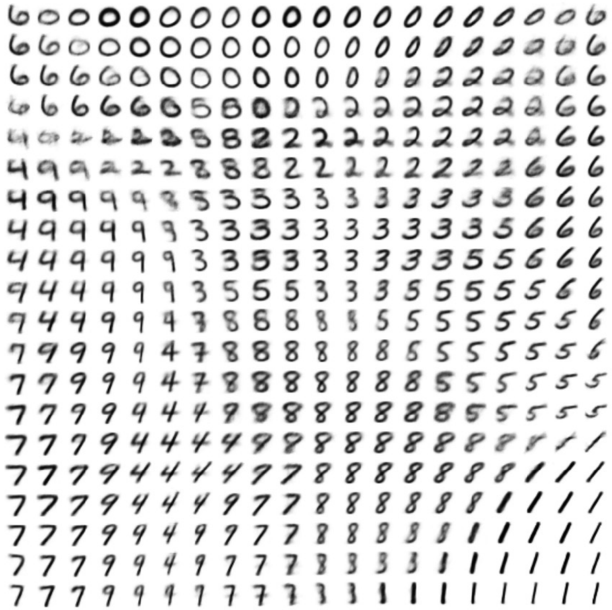
These inputs are simply raw normalised pixel values, like [[0.3,-0.02,…], [1.2,1.3,…, 0.05]…]

- What needs to happen to get from low-level information to a more abstract high-level representation?
- A process which gradually increases such information (the semantics)
- and reduces information about irrelevant features (e.g. the exact color of the background, the size of the snout…)

This deep, lower dimensional space learns meaningful structures



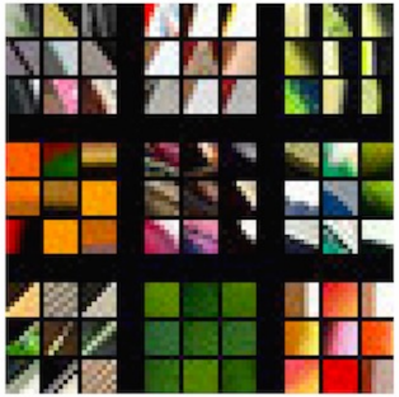
Lower dimensional space also sometimes called "manifold"
Here, rotation simply means going in one direction of this manifold, and size changes are another direction.
The RGB space does not have this structure



Empirically we indeed see this

What do the different layers in a deep neural network learn

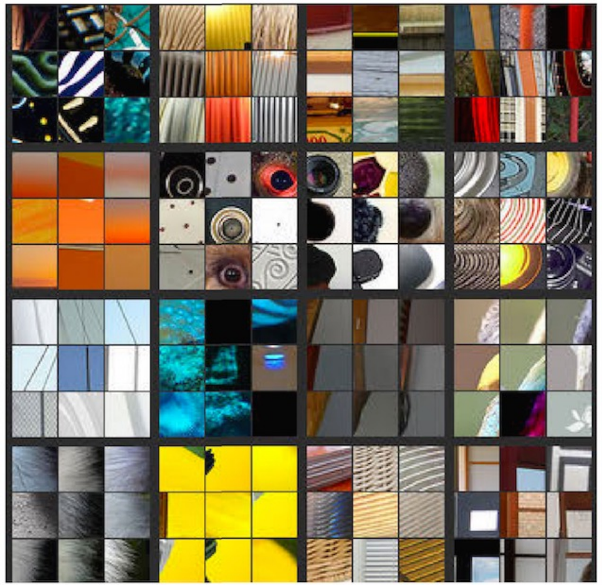
Layer 1



Filters

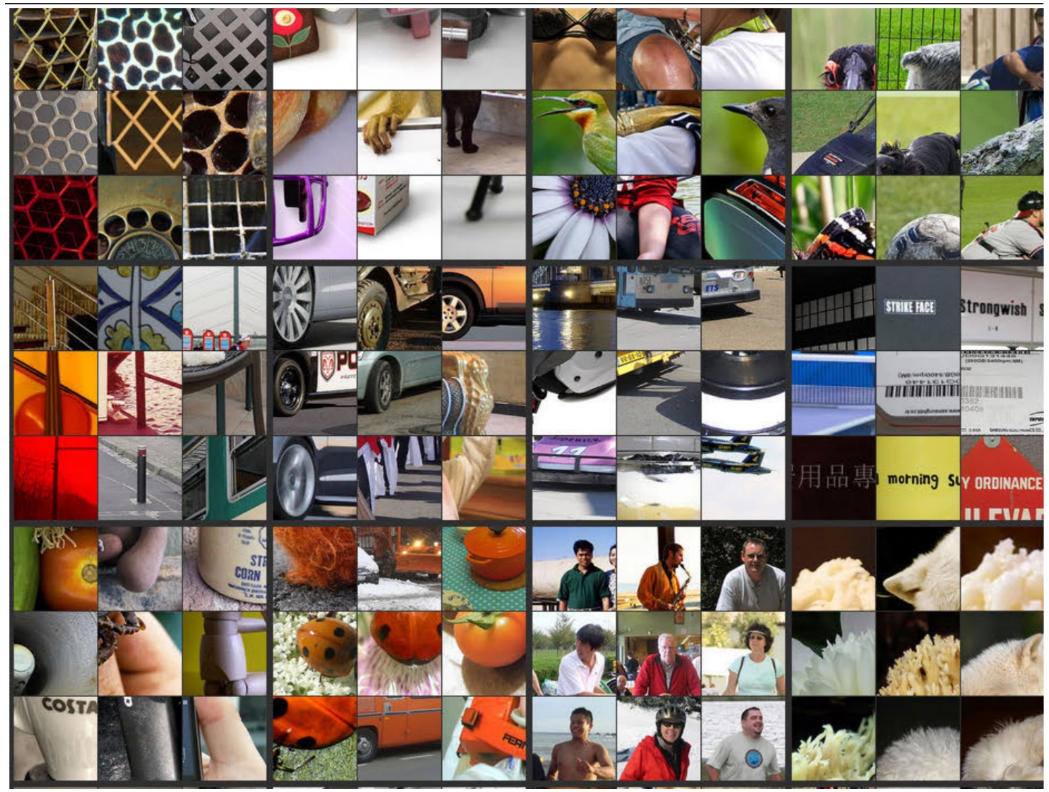
Image patches that activate this neuron maximally

Layer 2



(note that patches are bigger now, as receptive field size is larger)

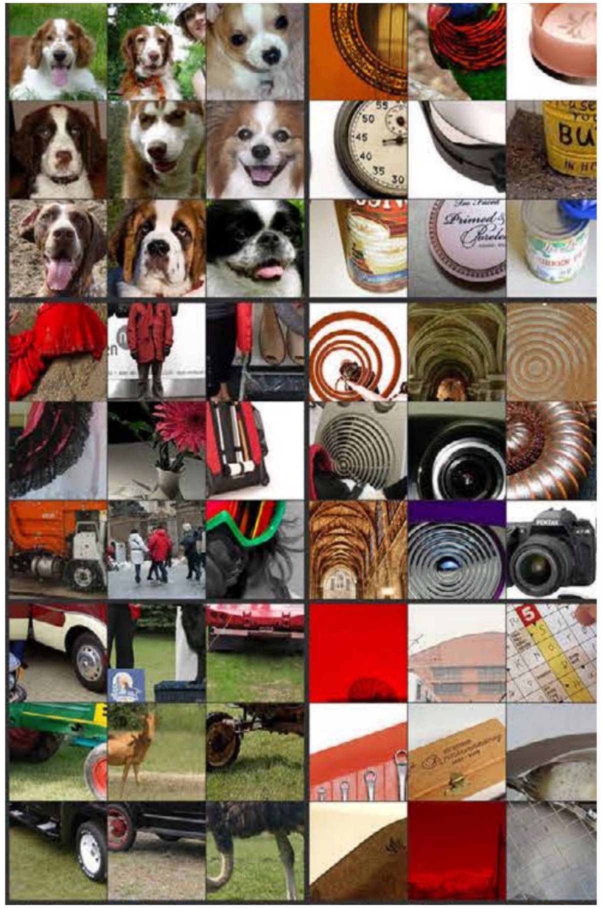
Layer 3



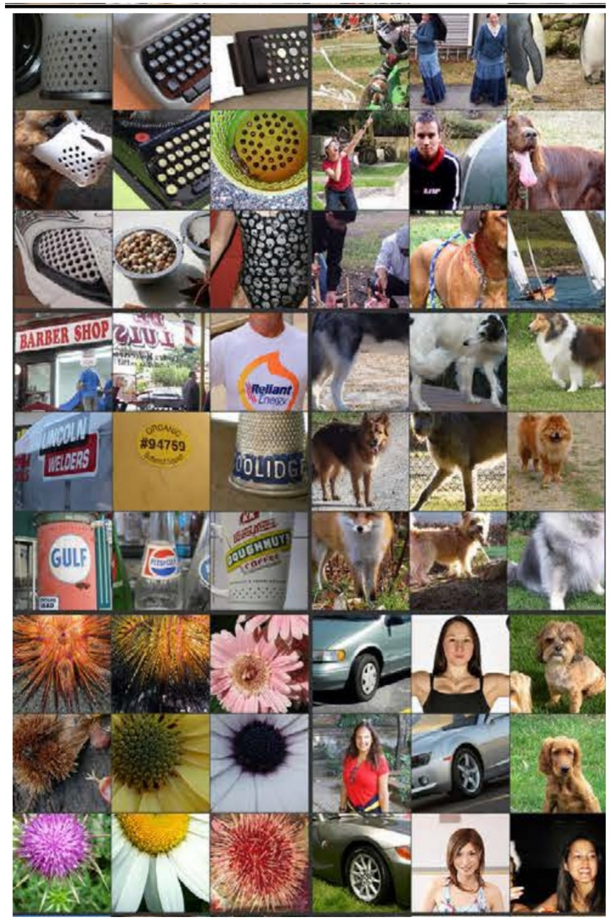
Visualizing and Understanding Convolutional Networks. Zeiler & Fergus. ECCV 2014

What do the different layers in a deep neural network learn

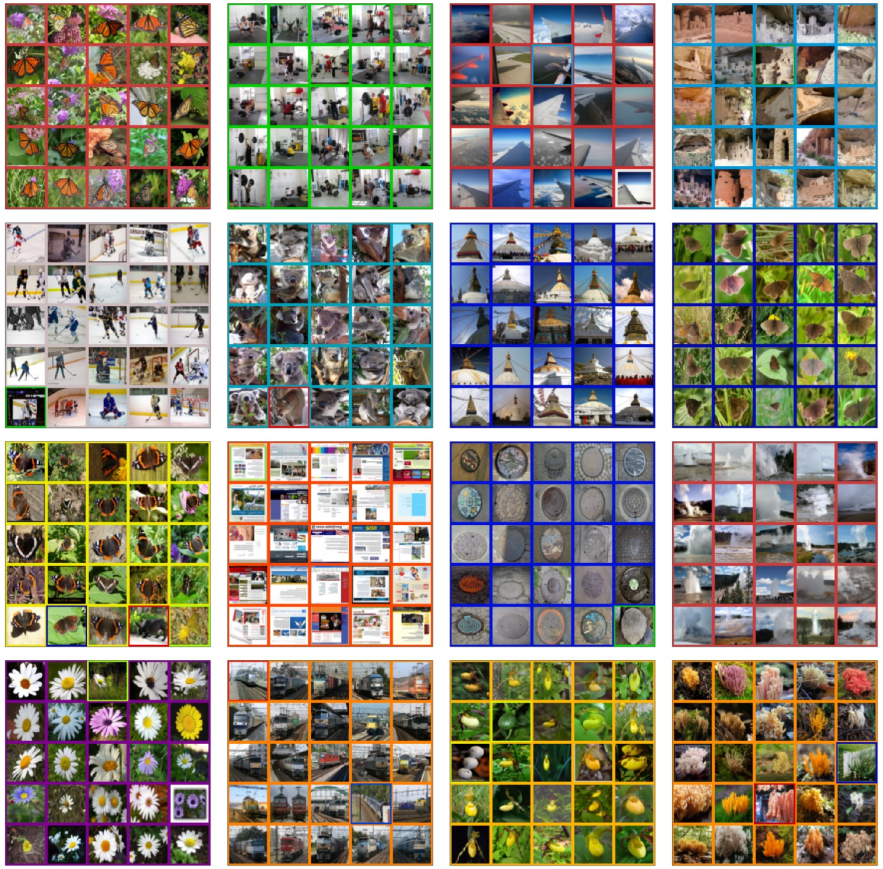
Layer 4



Layer 5



Actually same for self-supervised methods:



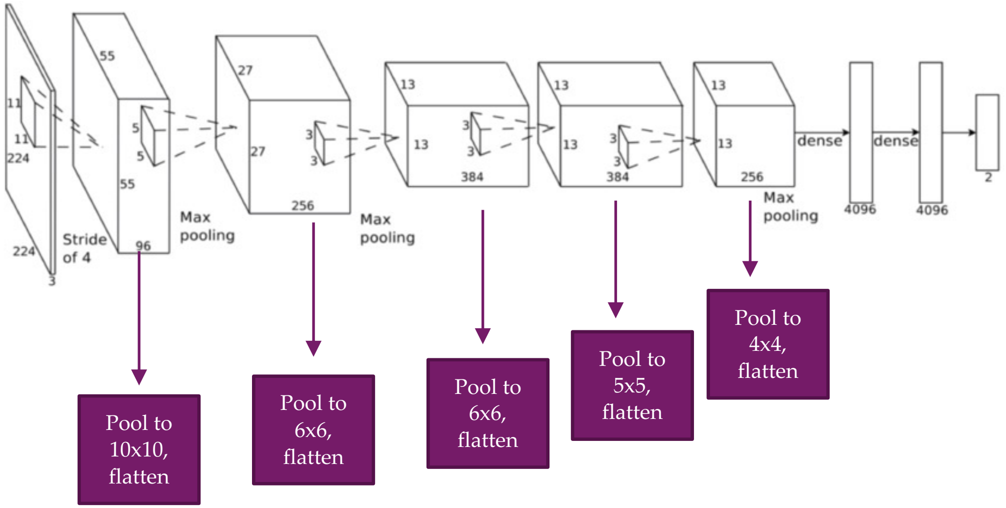
Last layer. (border color indicates true image class ID)

Visualizing and Understanding Convolutional Networks. Zeiler & Fergus. ECCV 2014

How do these layers correspond to “semantics”: numerical evaluation

Linear probing method: “exit” network at different locations & evaluate how good the model is at a given depth
How? Keep backbone (the whole AlexNet) frozen and only train a simple model, such as a linear layer

AlexNet



yields features of roughly equal sizes [9600, 9216, 9600, 9600, 9216]

now train a linear layer on top of each to predict the 1K classes.
call these classifiers [c1, c2, c3, c4, c5]

	ILSVRC-12				
Method	c1	c2	c3	c4	c5
ImageNet supervised, (Zhang et al., 2017)	19.3	36.3	44.2	48.3	50.5
Random, (Zhang et al., 2017)	11.6	17.1	16.9	16.3	14.1

Observations
- Performance gradually improves --> deeper representations are contain more semantic information of ILSVRC-12 (ImageNet)
- (Even randomly initialised networks extract some useful features)

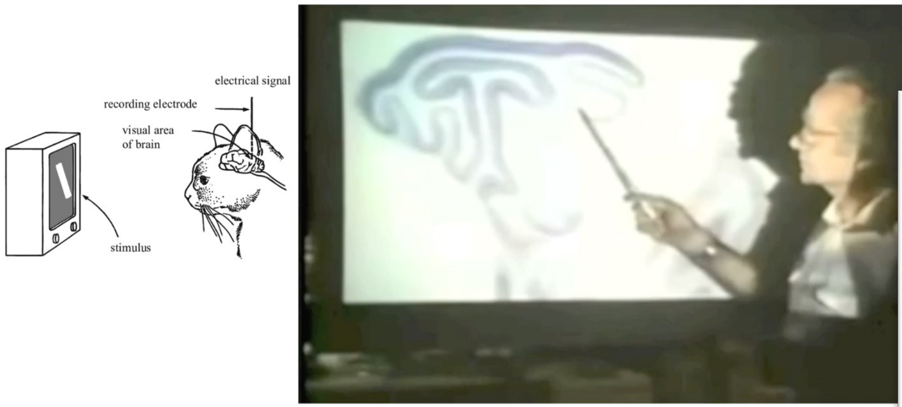
Let's wake up: Summarize the last few minutes to your neighbor.

What's the key concept here? – Try to explain *using your own words.*
What do you find intuitive to understand?

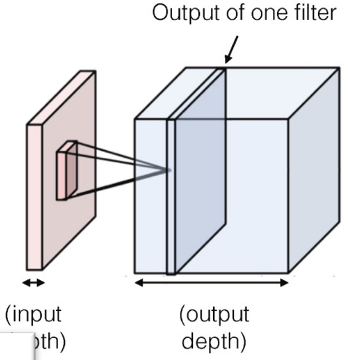
What do you find difficult to understand?

Last time

Hubel and Wiesel: Nobel Prize for Physiology or Medicine in 1981



3D Activations



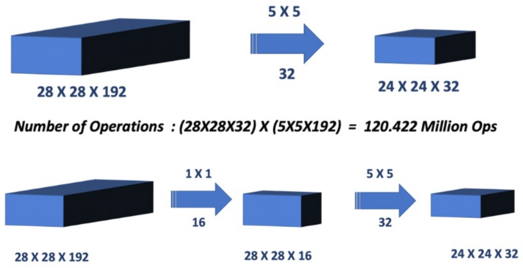
One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

1x1 Convolution: a computationally cheap method

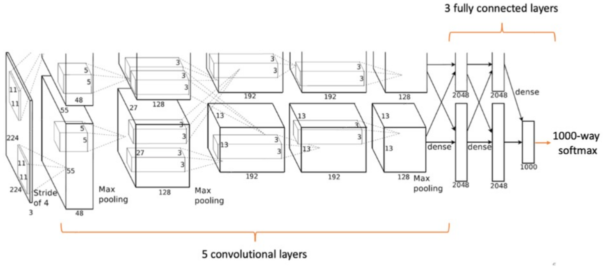
- Eg 28x28x192 -> 24x24x32



Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

AlexNet

- The architecture consists of eight layers:
  - five convolutional layers
  - three fully-connected layers



UPDATE: Transfer learning

Instagram: 5Billion+…

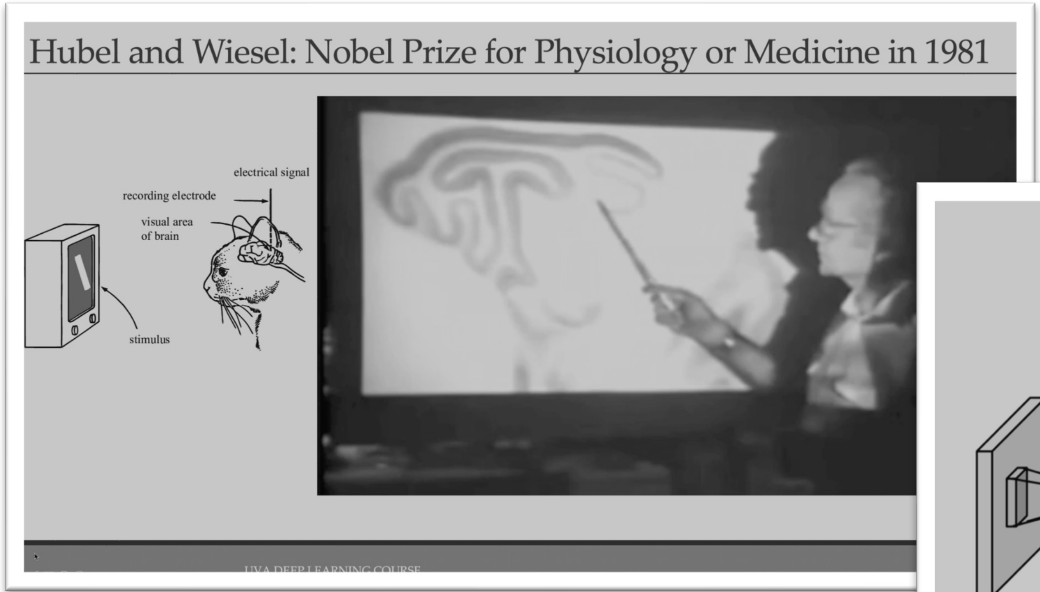
- Assume two datasets, T and S
- Dataset S is
  - ~~fully annotated, plenty of images~~
  - HUGE
  - We can build a model h_S (using self-supervised learning)
- Dataset T is
  - Not as much annotated, or much fewer images
  - The annotations of T do not need to overlap with S
- We can use the model h_S to learn a better specialised h_T
- This is called transfer learning

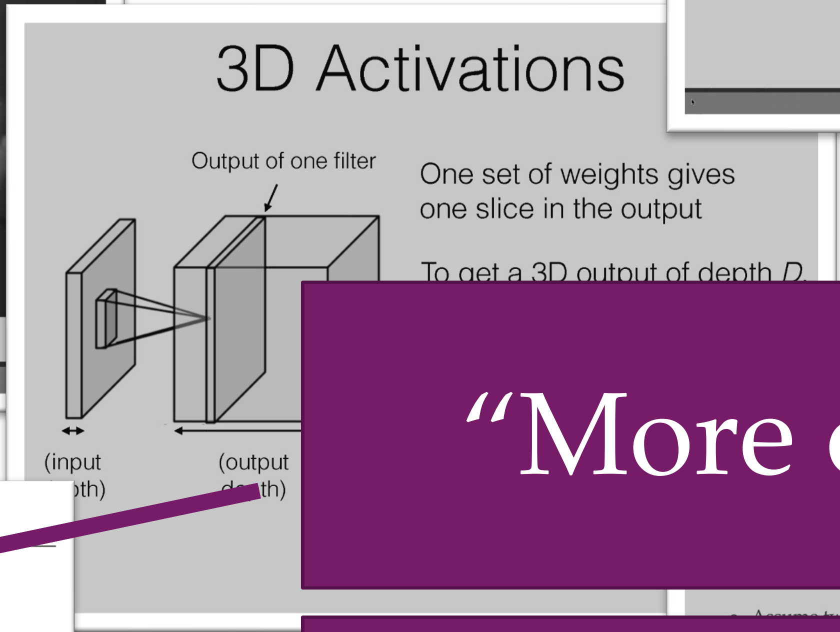
h_S

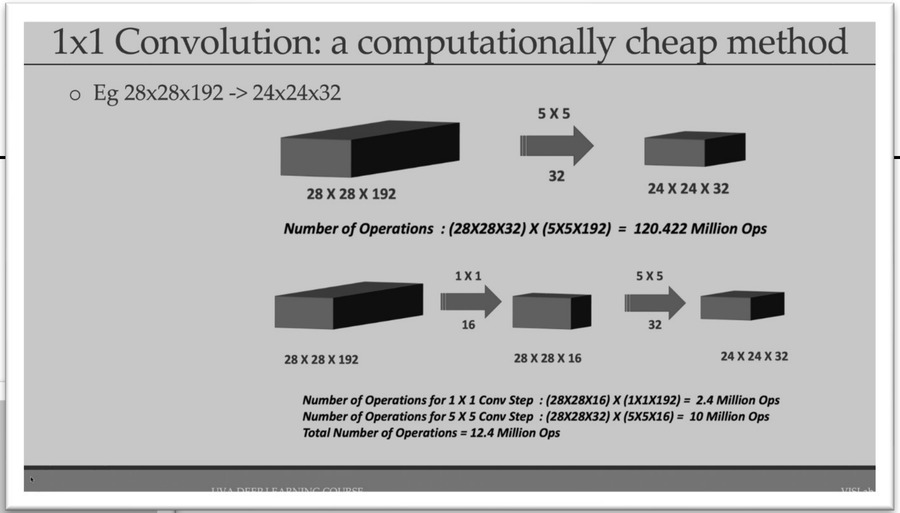
"My dataset": 1,000

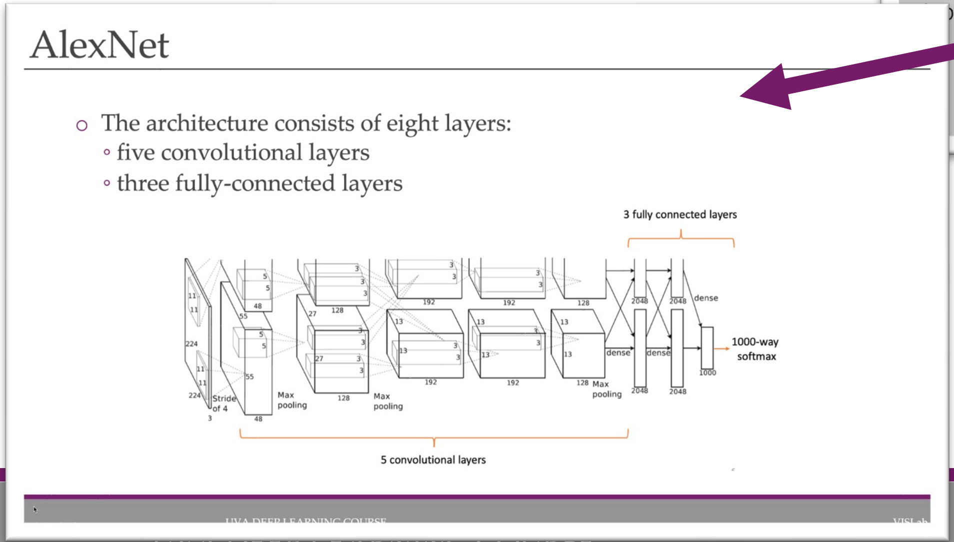
h_T

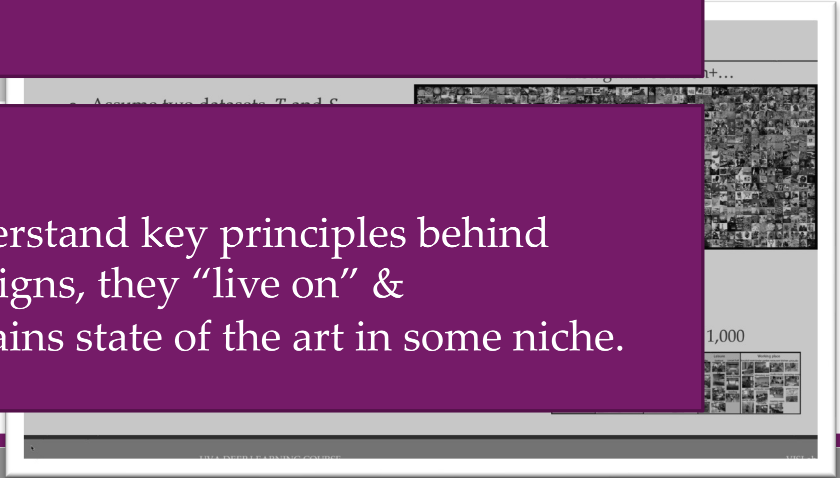
Today:











"More of this"

Why?
Important to understand key principles behind modern CNN designs, they "live on" &
most models remains state of the art in some niche.

VGG network

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 image)					
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					

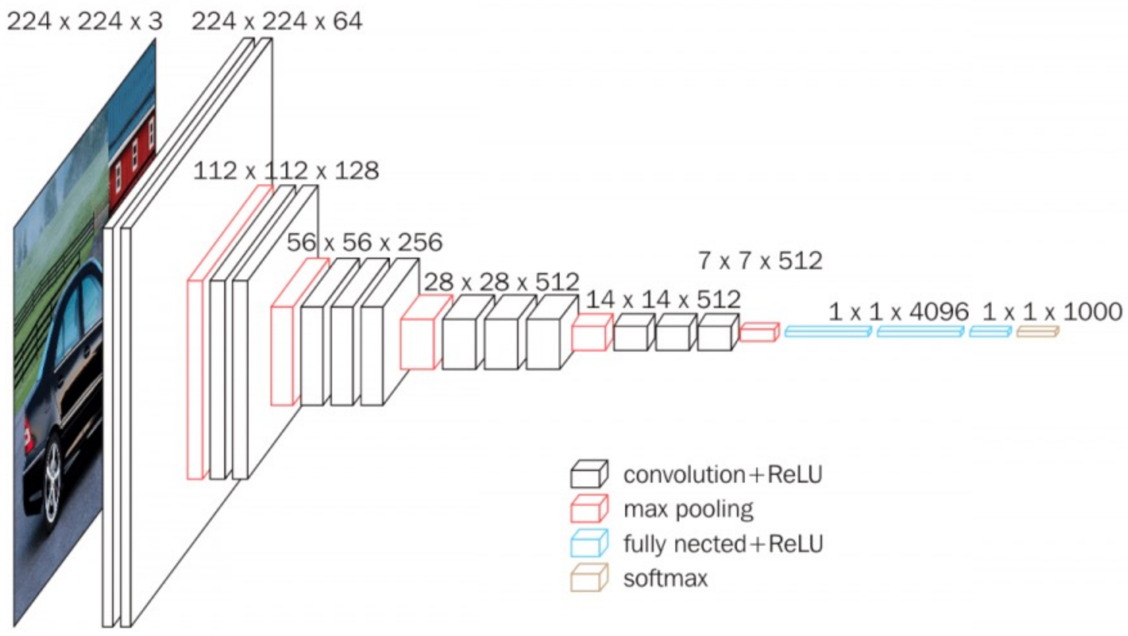
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG16

- The next step after AlexNet
- VGG addresses another very important aspect of CNNs: depth
- The runner up of the ImageNet classification challenge with 7.3% error rate

The architecture of VGG16



Characteristics

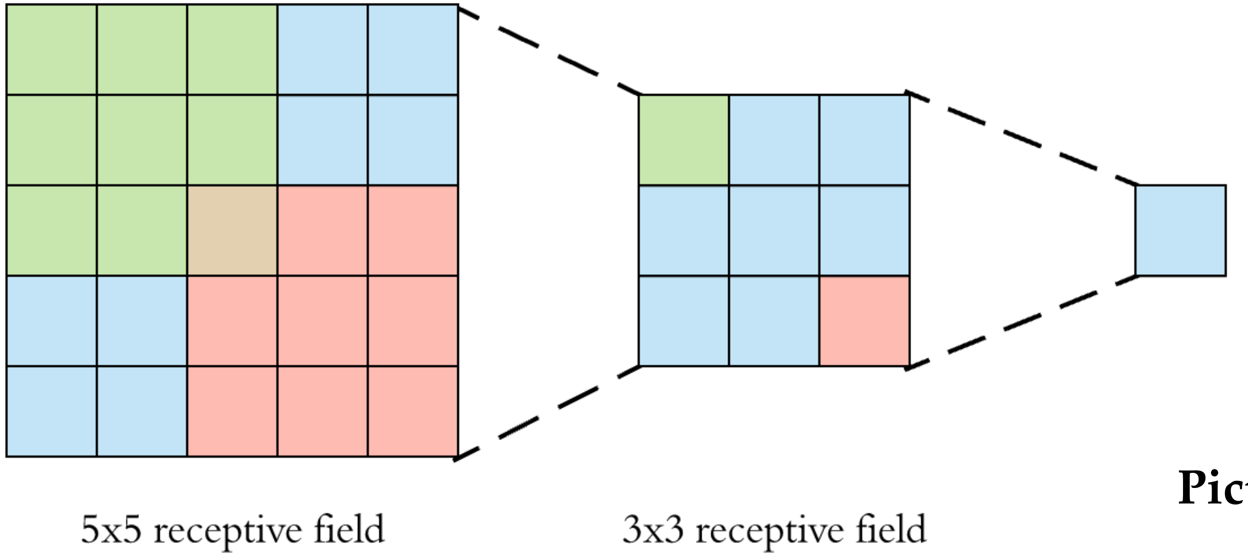
- Input size: 224×224
- Filter sizes: 3×3
- Convolution stride: 1
  - Spatial resolution preserved
- Padding: 1
- Max pooling: 2×2 with a stride of 2
- ReLU activations
- No fancy input normalizations
  - No Local Response Normalizations
- Although deeper, number of weights is not exploding

Why 3×3 filters?

Layer 1

Why 3×3 filters?

- The smallest possible filter to capture the "up", "down", "left", "right"
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of …



Picture credit: Arden Dertat

Why 3×3 filters?

- The <u>smallest</u> possible filter to captures the “up”, “down”, “left”, “right”
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?

Why 3×3 filters?

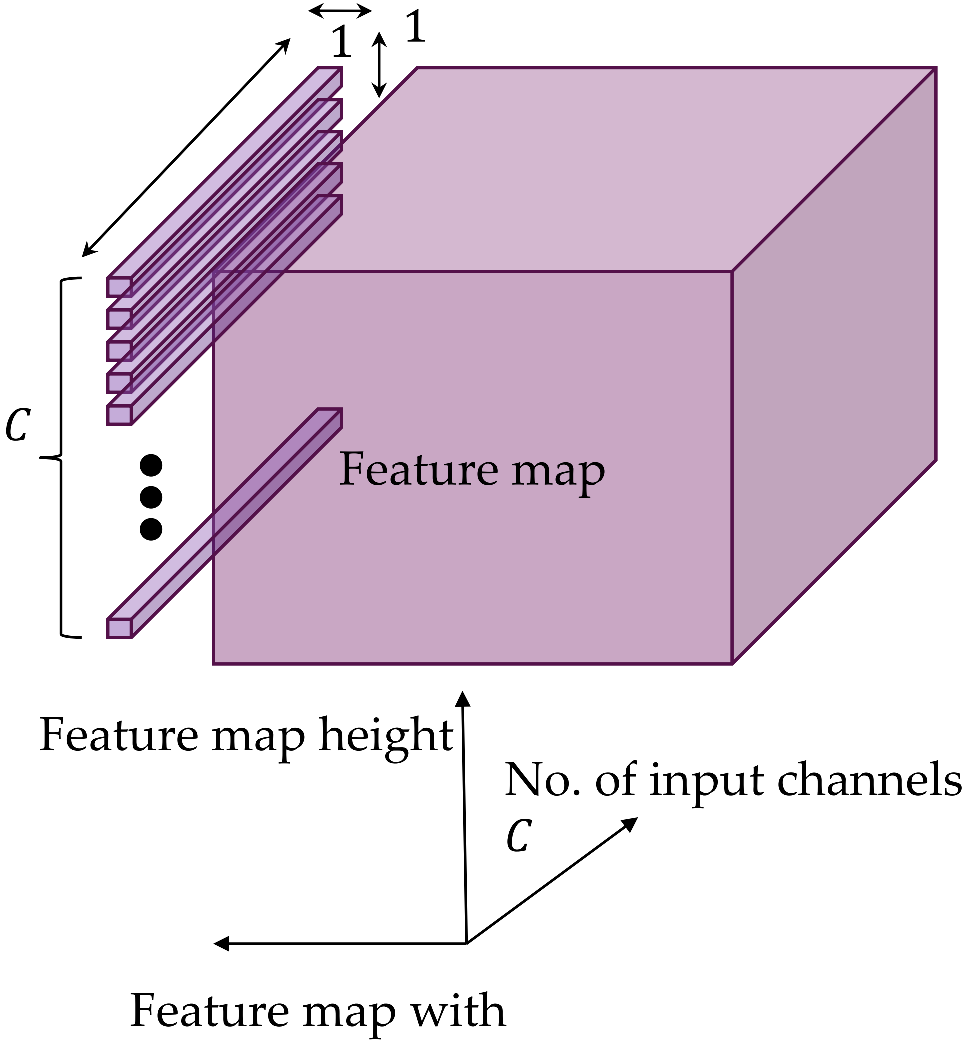
- The <u>smallest</u> possible filter to captures the "up", "down", "left", "right"
- Two 3×3 filters have the receptive field of one 5×5
- Three 3×3 filters have the receptive field of one 7×7
- 1 large filter can be replaced by a deeper stack of successive smaller filters
- Benefit?
- Three more nonlinearities for the same "size" of pattern learning
- Also fewer parameters and regularization

$$(3\times 3\times C)\times 3 = 27\cdot C,\ 7\times 7\times C\times 1 = 49\cdot C$$

- Conclusion: 1 large filter can be replaced by a deeper, potentially more powerful, stack of successive smaller filters

Even smaller filters?

- Also $1x1$ filters are used
- Followed by a nonlinearity
- **Why? (c.f. last lecture)**
- Increasing nonlinearities without affecting receptive field sizes
  - Linear transformation of the input channels



Overall shapes and sizes when inputting a 224x224 image:

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

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 image)					
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					

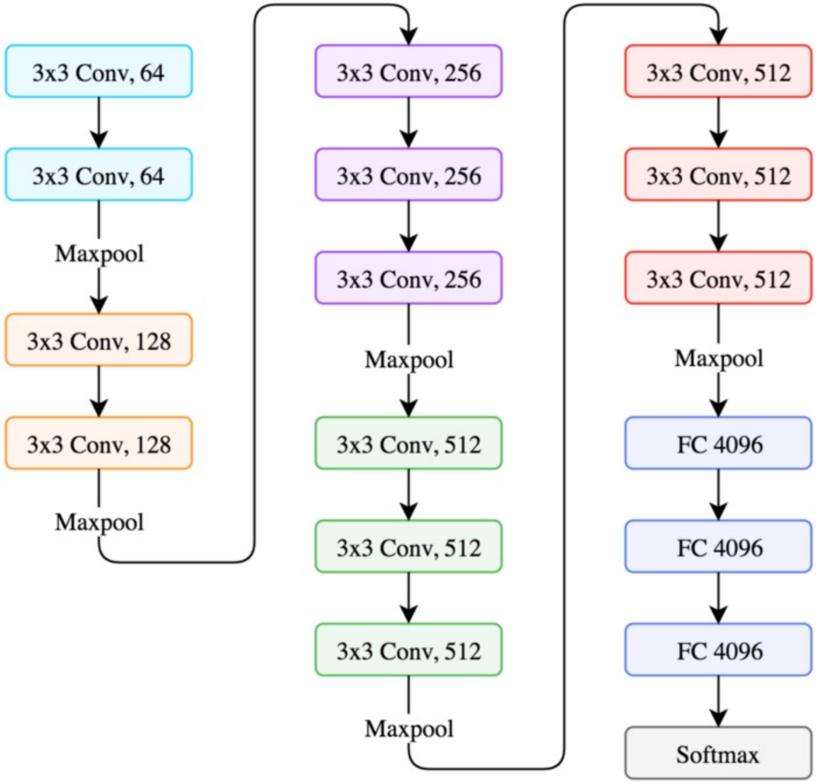
Table 2: **Number of parameters** (in millions).

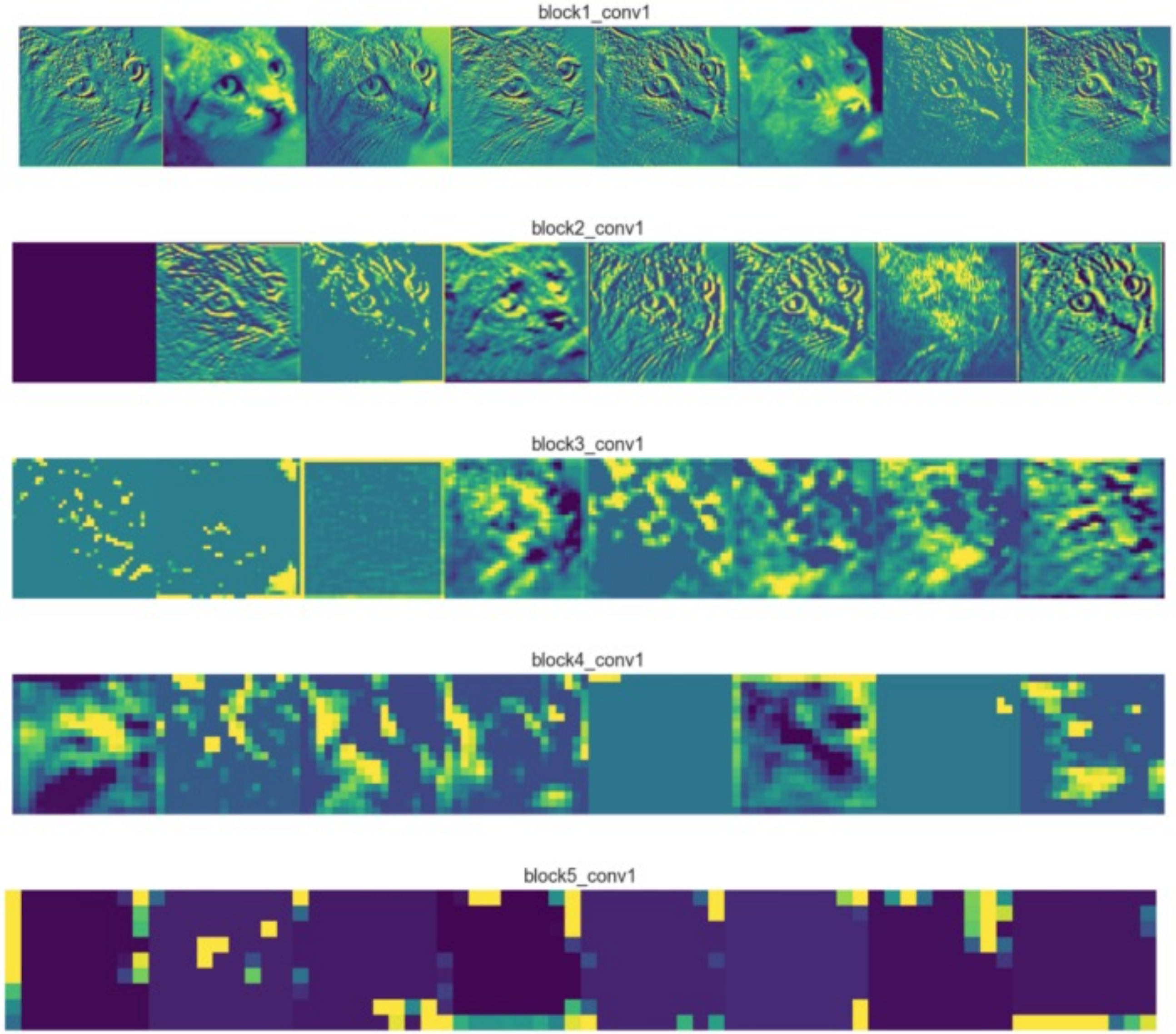
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Training

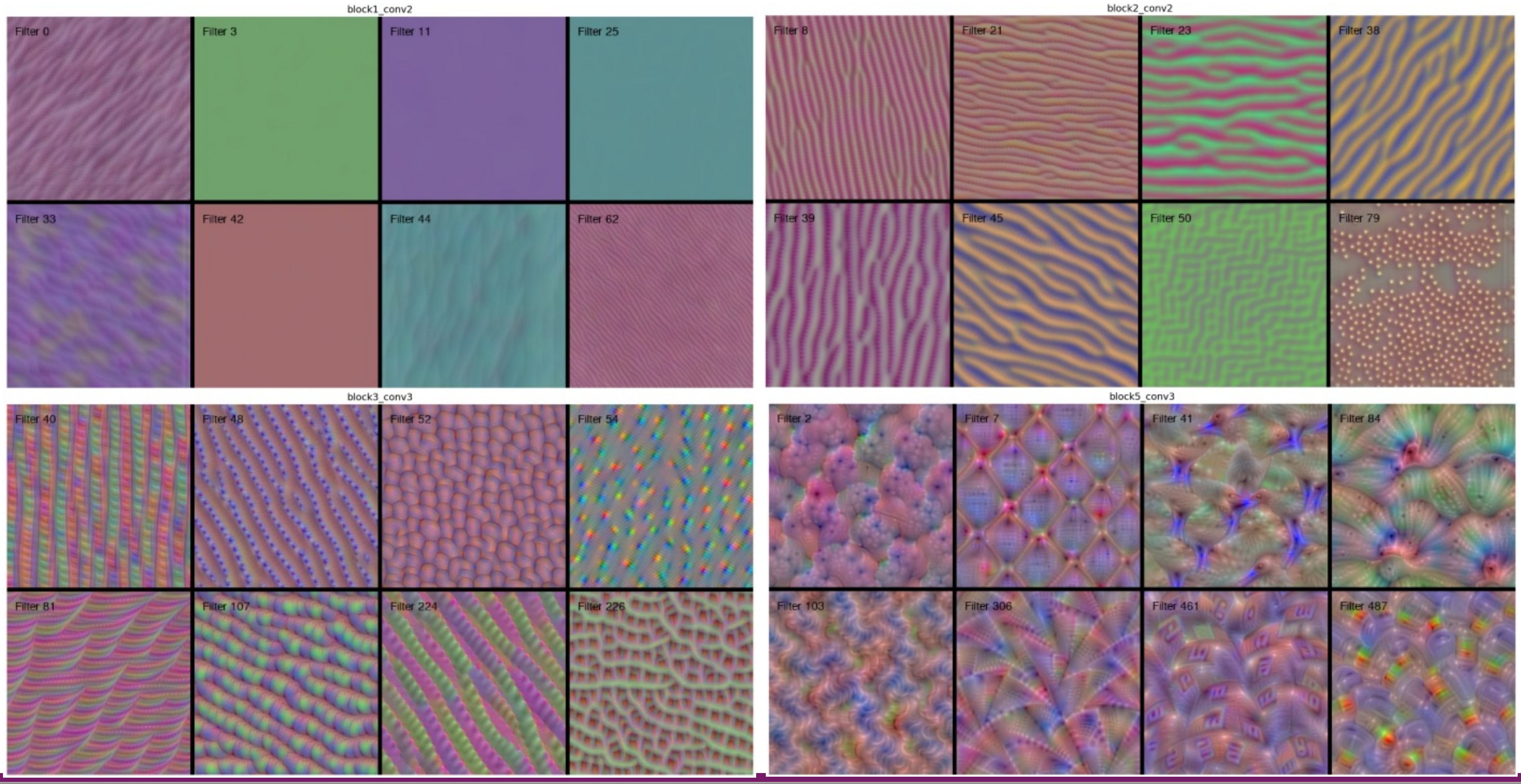
- Batch size: 256
- SGD with momentum=0.9
- Weight decay $\lambda = 5 \cdot 10^{-4}$
- Dropout on first two fully connected layers
- Learning rate $\eta_0 = 10^{-2}$, then decreased by factor of 10 when validation accuracy stopped improving
  - Three times this learning rate decrease
- Faster training
  - Smaller filters
  - Depth also serves as regularization

Feature maps

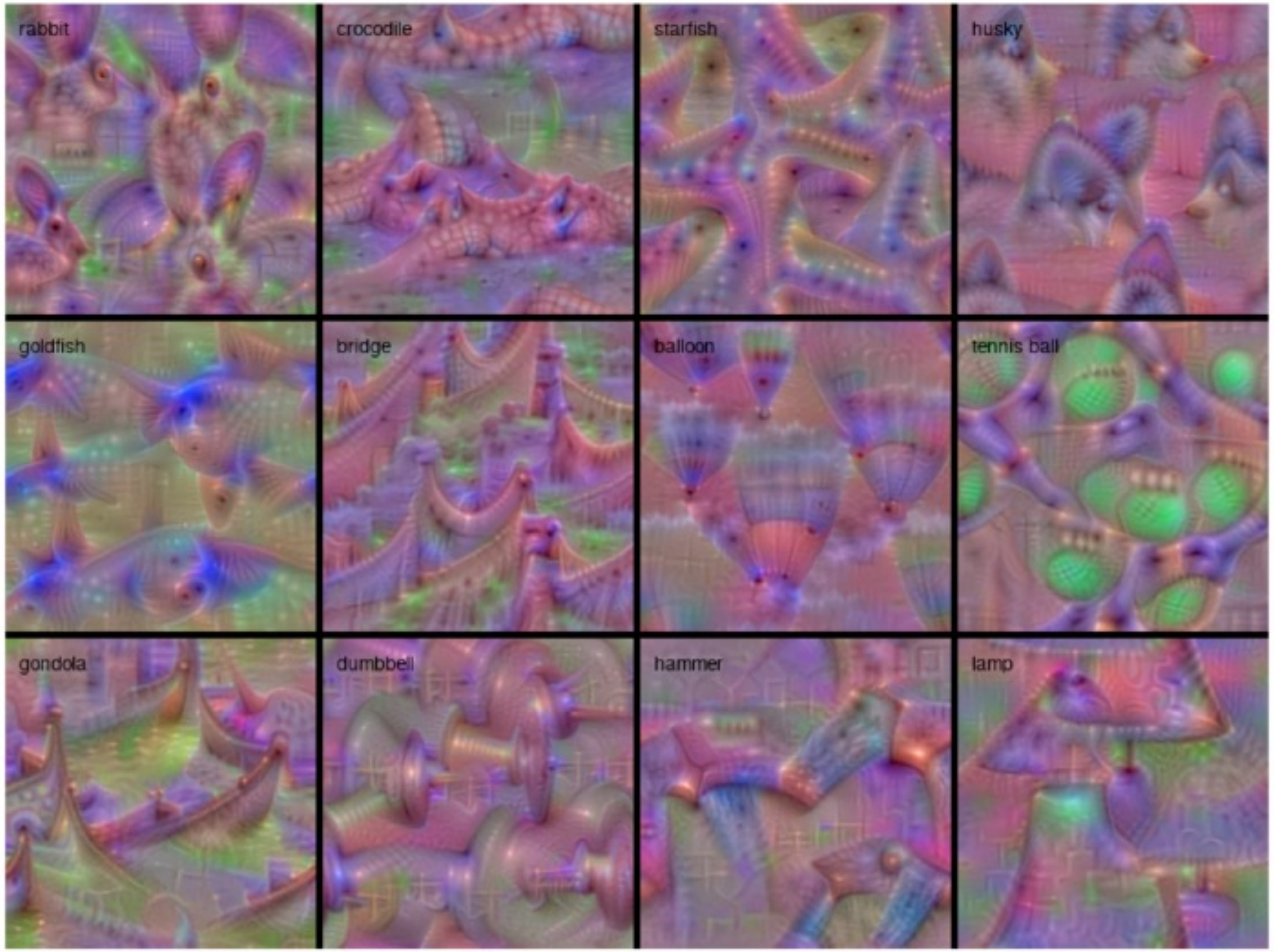




Filters


block1_conv2
Filter 0
Filter 3
Filter 11
Filter 25
Filter 33
Filter 42
Filter 44
Filter 62
block2_conv2
Filter 8
Filter 21
Filter 23
Filter 38
Filter 39
Filter 45
Filter 50
Filter 79
block3_conv3
Filter 40
Filter 48
Filter 52
Filter 54
Filter 81
Filter 107
Filter 224
Filter 226
block5_conv3
Filter 2
Filter 7
Filter 41
Filter 84
Filter 103
Filter 306
Filter 461
Filter 487

Class Outputs



- Basic process:

1. Goal: find input that maximizes the value of a convnet filter's output

2. Start from a blank input image.

3. Do *gradient ascent* in input space. Meaning modify the input values such that the filter activates even more.

4. Repeat this in a loop.

(note: now-a-days some more sophisticated visualisations exist, but basic principle stays)

GoogLeNet

type	patch size/ stride	output size	depth	$\#1{\times}1$	$\#3{\times}3$ reduce	$\#3{\times}3$	$\#5{\times}5$ reduce	$\#5{\times}5$	pool proj	params	ops
convolution	$7{\times}7/2$	$112{\times}112{\times}64$	1							2.7K	34M
max pool	$3{\times}3/2$	$56{\times}56{\times}64$	0								
convolution	$3{\times}3/1$	$56{\times}56{\times}192$	2		64	192				112K	360M
max pool	$3{\times}3/2$	$28{\times}28{\times}192$	0								
inception (3a)		$28{\times}28{\times}256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28{\times}28{\times}480$	2	128	128	192	32	96	64	380K	304M
max pool	$3{\times}3/2$	$14{\times}14{\times}480$	0								
inception (4a)		$14{\times}14{\times}512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14{\times}14{\times}512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14{\times}14{\times}512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14{\times}14{\times}528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14{\times}14{\times}832$	2	256	160	320	32	128	128	840K	170M
max pool	$3{\times}3/2$	$7{\times}7{\times}832$	0								
inception (5a)		$7{\times}7{\times}832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7{\times}7{\times}1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7{\times}7/1$	$1{\times}1{\times}1024$	0								
dropout (40%)		$1{\times}1{\times}1024$	0								
linear		$1{\times}1{\times}1000$	1							1000K	1M
softmax		$1{\times}1{\times}1000$	0								

Table 1: GoogLeNet incarnation of the Inception architecture

Basic idea

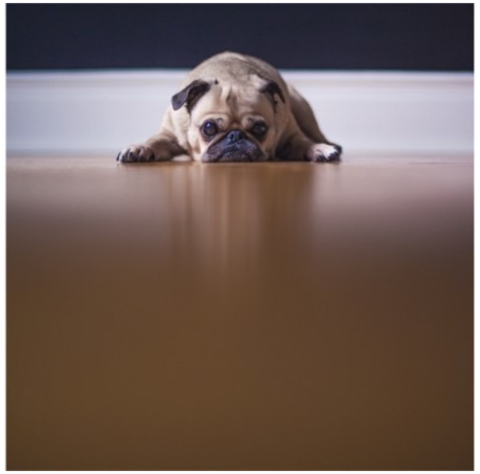
- Problem?

Picture credit: Bharath Raj

Basic idea

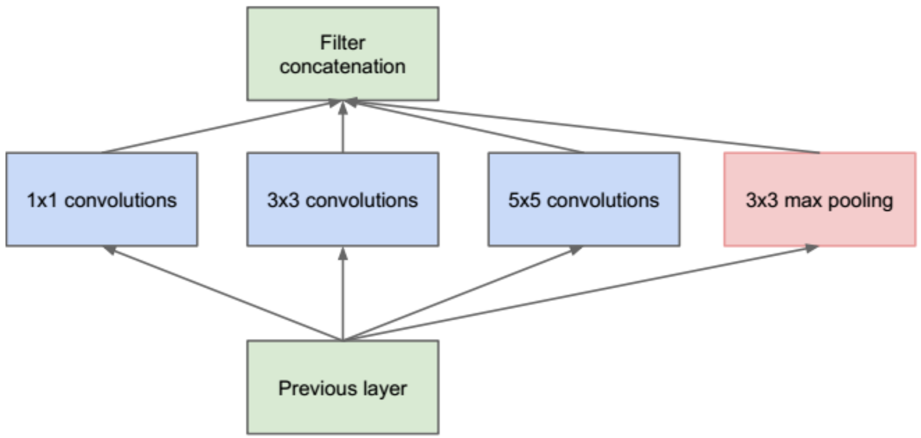
- Salient parts have great variation in sizes
- Hence, the receptive fields should vary in size accordingly
- Naively stacking convolutional operations is expensive
- Very deep nets are prone to overfitting





Inception module

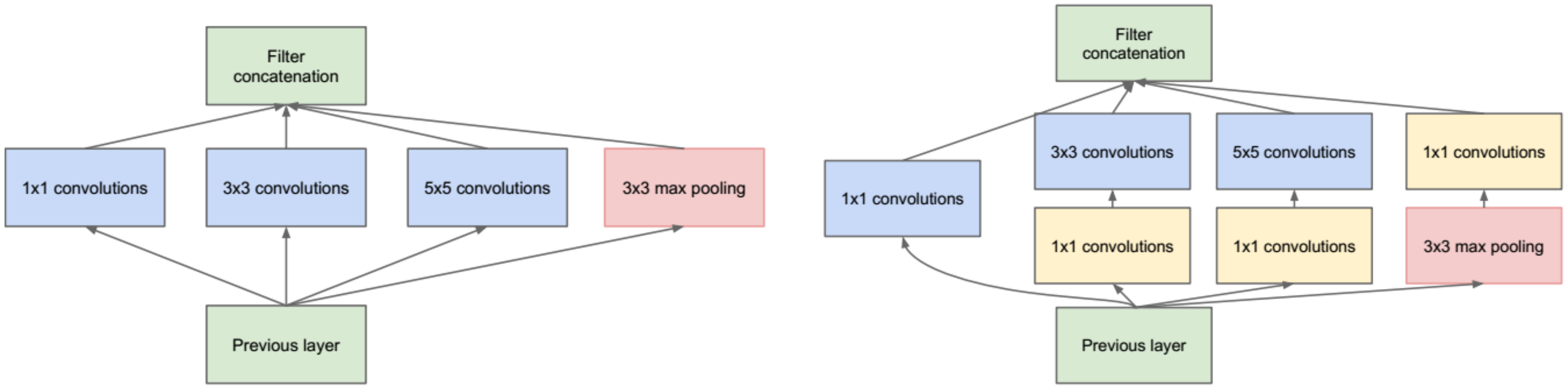
- Multiple kernel filters of different sizes $(1\times1, 3\times3, 5\times5)$
  - Naïve version
- Problem?



(a) Inception module, naïve version

Inception module

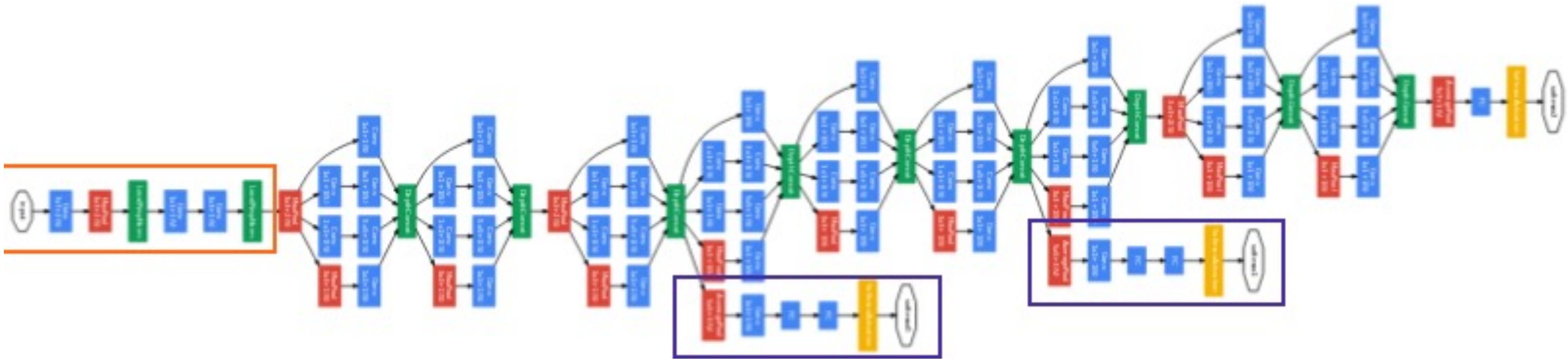
- Multiple kernel filters of different sizes $(1\times 1, 3\times 3, 5\times 5)$
  - Naïve version
- Problem?
  - Very expensive!
- Add intermediate 1×1 convolutions



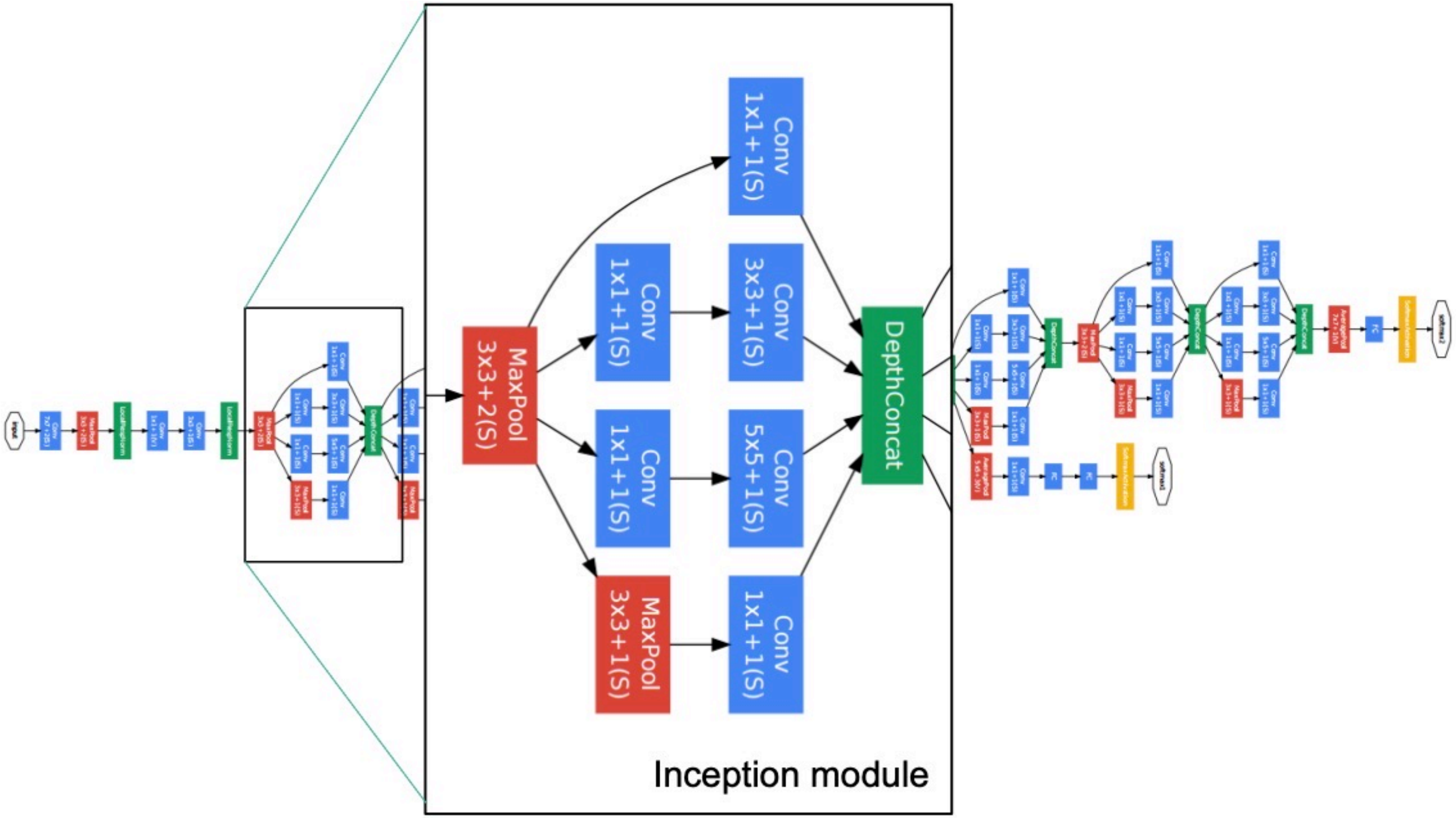
(a) Inception module, naïve version (b) Inception module with dimension reductions

Architecture

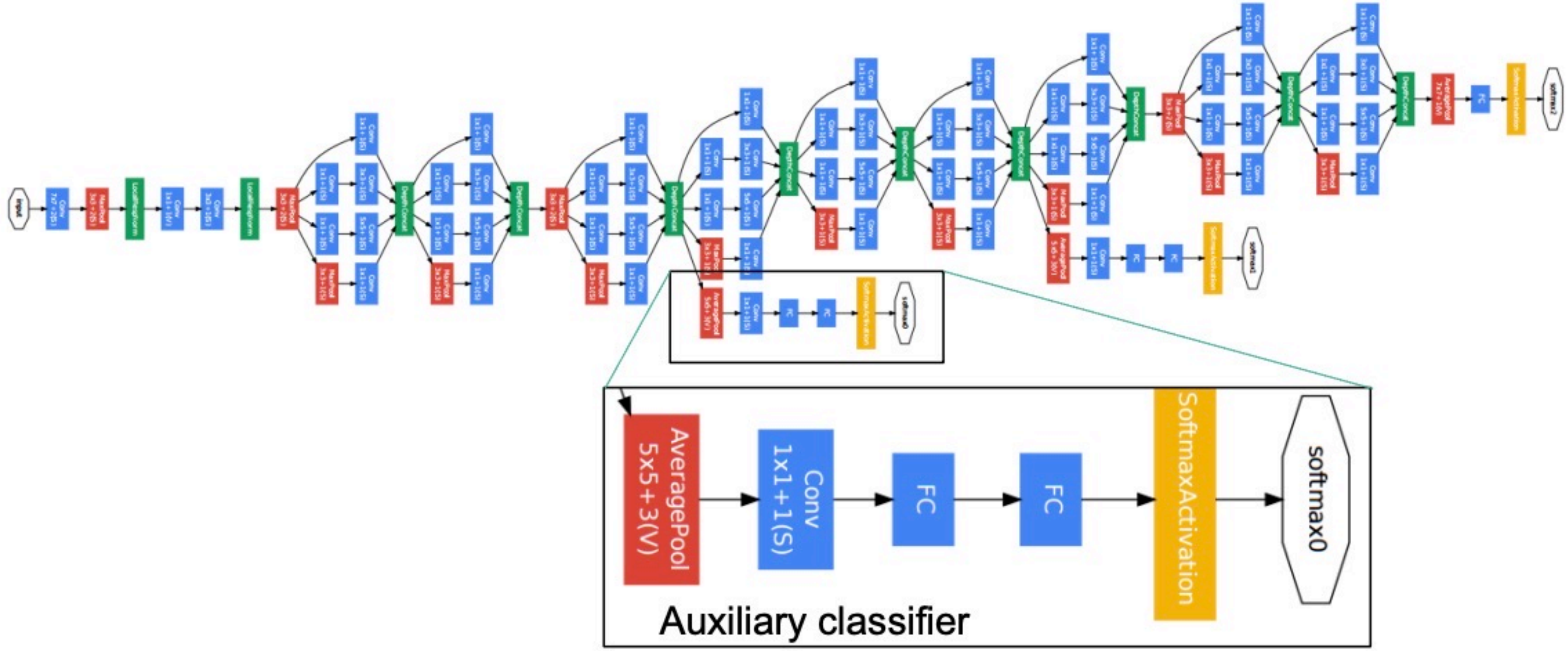
- 9 Inception Modules
- 22 layers deep (27 with the pooling layers)
- Global average pooling at the end of last Inception Module
- 6.67% Imagenet error, compared to 18.2% of Alexnet



Architecture: the “Inception” module



Architecture: the auxiliary classifier idea



Why aux classifiers? Vanishing gradients

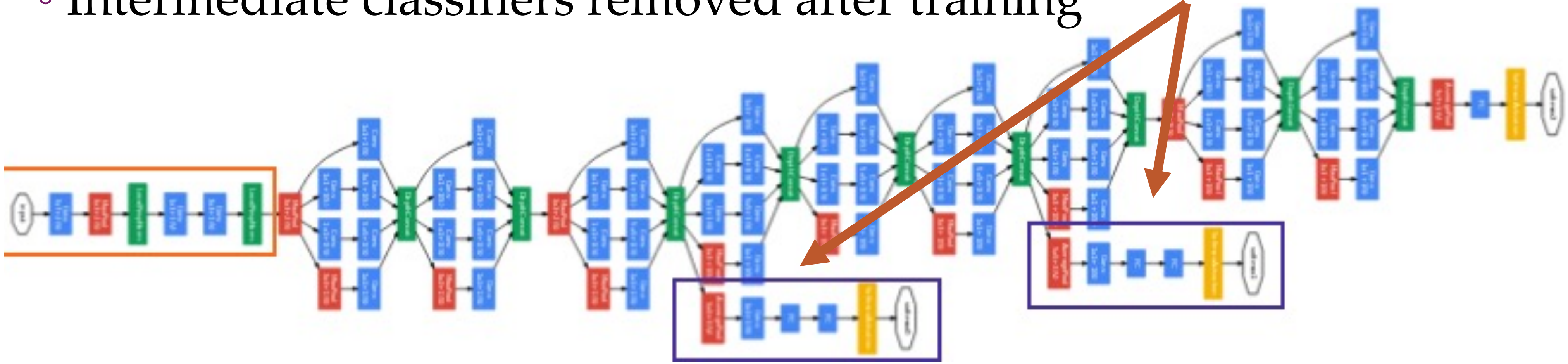
- The network was too deep (at the time)
- Roughly speaking, backprop is lots of matrix multiplications

$$\frac{\partial \mathcal{L}}{\partial w^l} = \frac{\partial \mathcal{L}}{\partial a^L} \cdot \frac{\partial a^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial a^{L-2}} \cdot \ \dots \cdot \frac{\partial a^l}{\partial w^l}$$

- Many of intermediate terms < 1 → the final $\frac{\partial \mathcal{L}}{\partial w^l}$ gets extremely small
- Extremely small gradient → Extremely slow learning

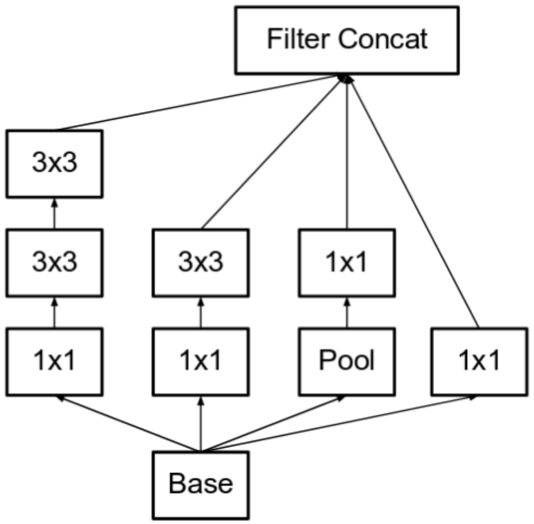
Architecture

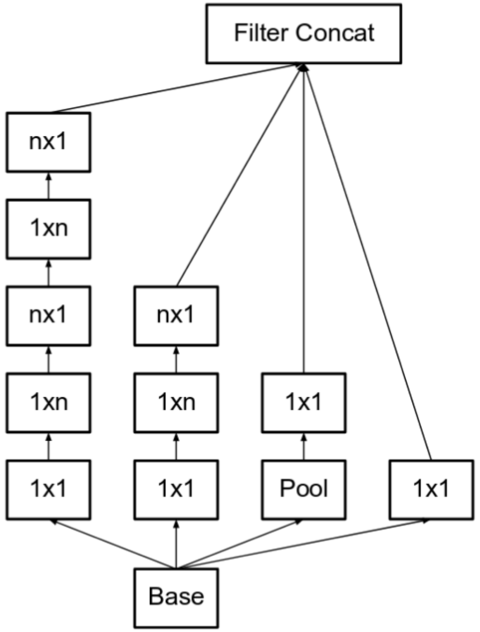
- 9 Inception Modules
- 22 layers deep (27 with the pooling layers)
- Global average pooling at the end of last Inception Module
- Because of the increased depth → **Vanishing gradients**
- Inception solution to vanishing gradients: **intermediate classifiers**
  - Intermediate classifiers removed after training

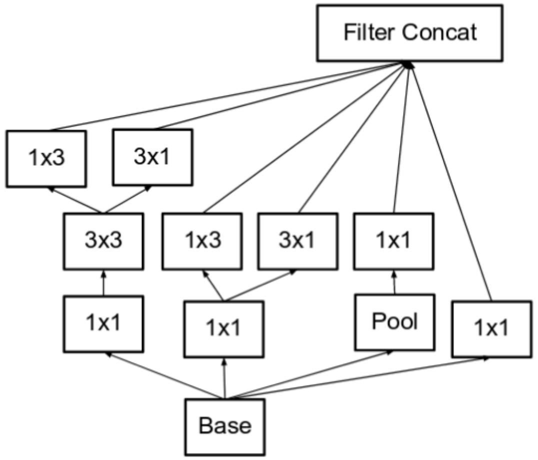


Inceptions v2, v3, v4, ….

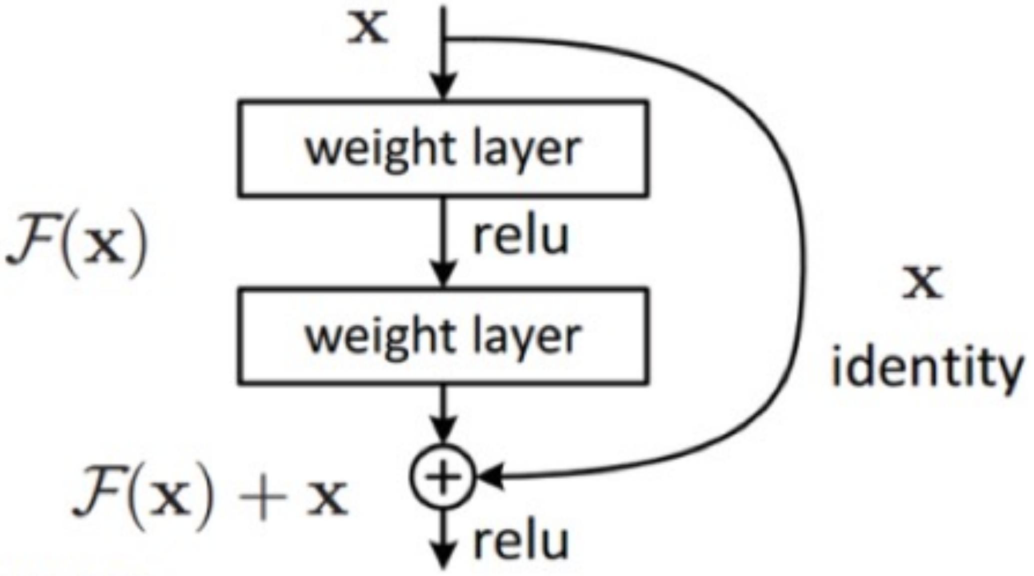
- Factorize 5×5 in two 3×3 filters
- Factorize $n \times n$ in two $n \times 1$ and $1 \times n$ filters (quite a lot cheaper)
- Make nets wider
- BatchNorms, …







ResNets
DenseNets
HighwayNets



A residual block

AlexNet (2012)

5 convolutional layers

3 fully-connected layers

Image credit: Andrea Vedaldi

Evolution

AlexNet (2012)

VGG-M (2013)

VGG-VD-16 (2014)

Image credit: Andrea Vedaldi


WE NEED TO GO
DEEPER

Evolution

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)

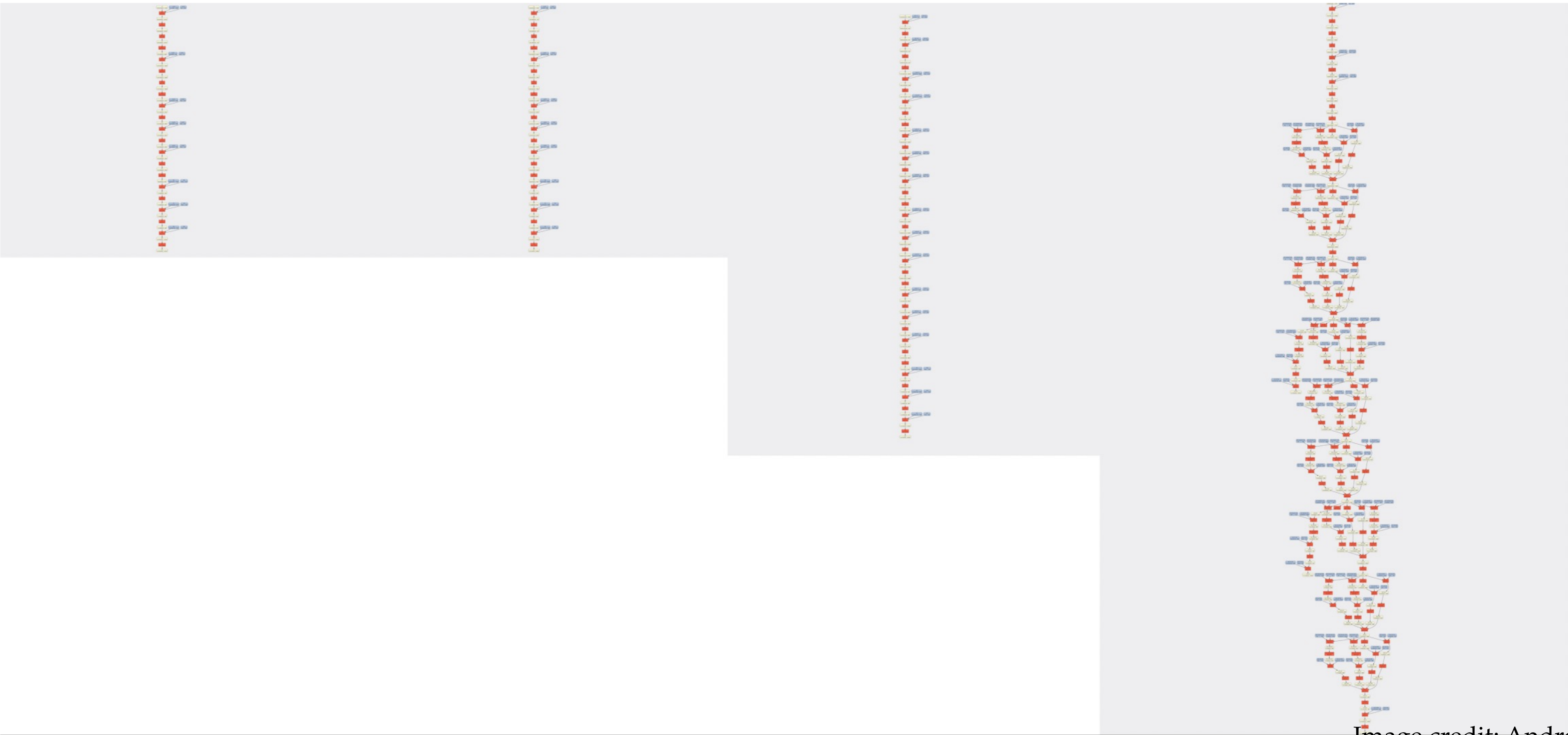
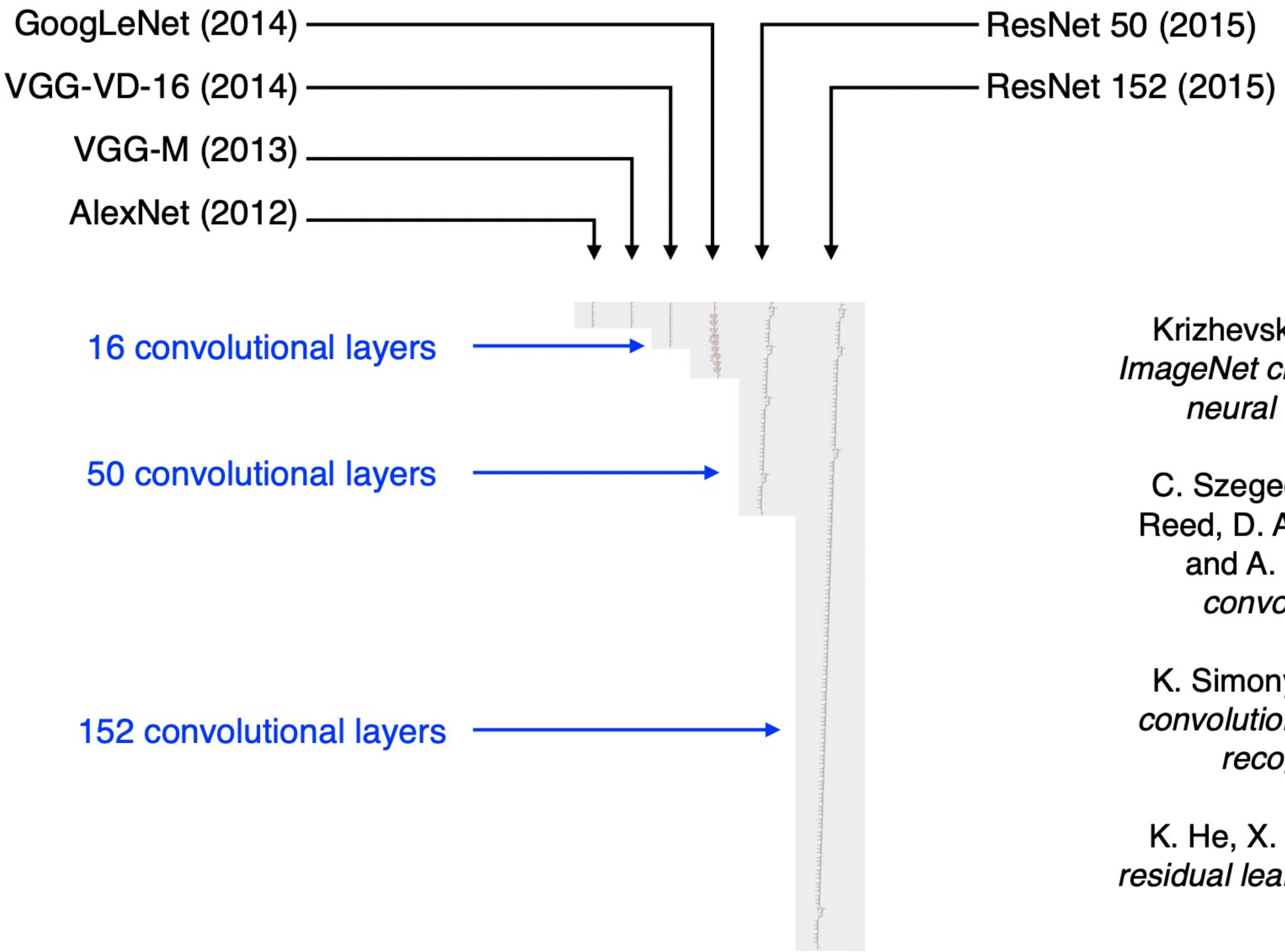


Image credit: Andrea Vedaldi

Evolution



Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

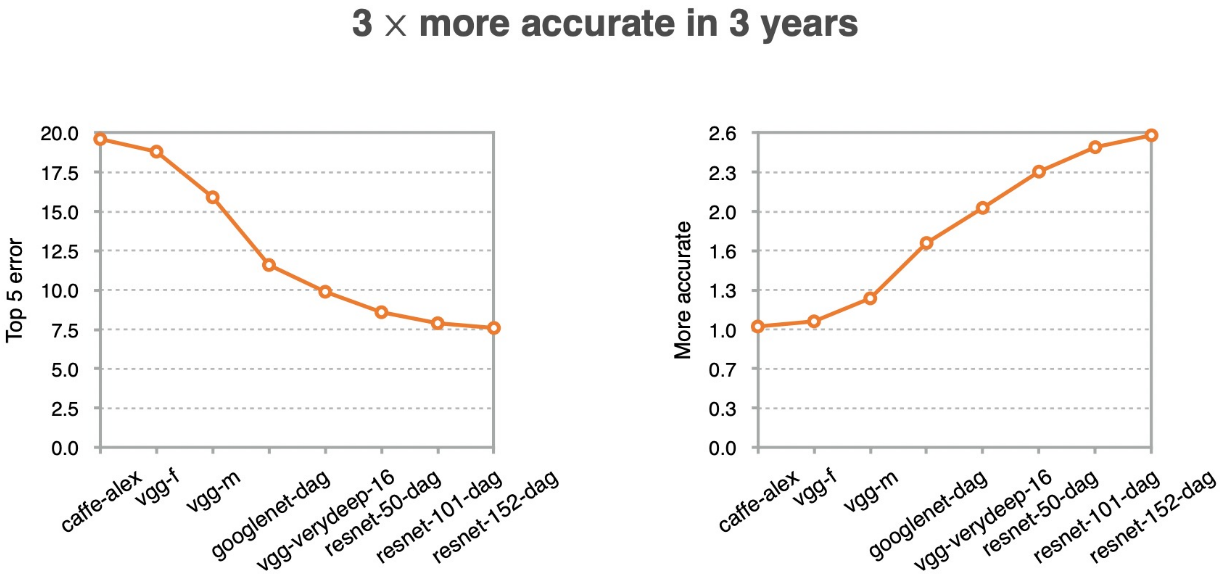
C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

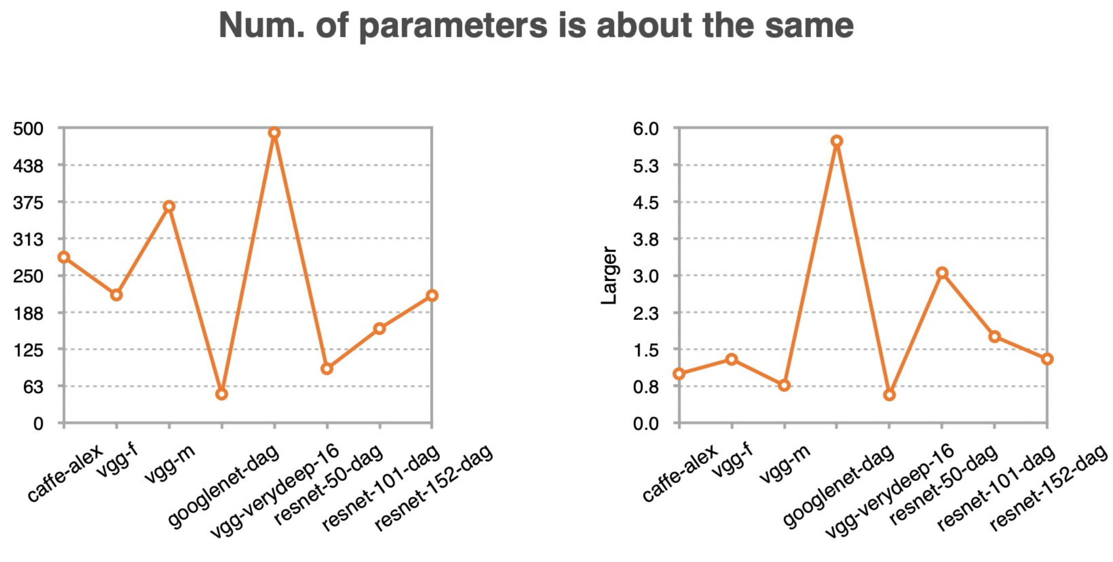
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

Image credit: Andrea Vedaldi

Why care about architectures… here's why:





Optimising the architecture is *smart:* better efficiency, performance etc.
Superior architectures (for a problem) exist: eg a CNN vs MLP for images)

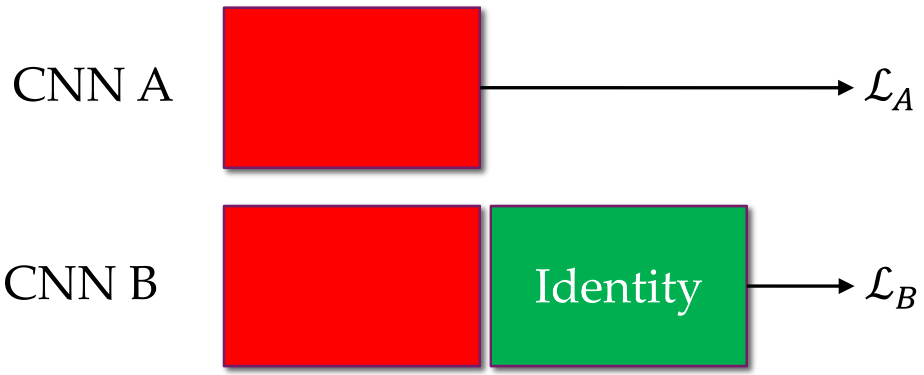
Image credit: Andrea Vedaldi

Some facts about ResNets

- The first truly Deep Network, going deeper to 152 and also 1000 layers
- More importantly, the first Deep Architecture that proposed a novel concept on how to gracefully go deeper than a few dozen layers
  - Not simply getting more GPUs, more training time, etc
- Further decreased Imagenet error, with a 3.57% Top-5 error (in ensembles)
- Won all object classification, detection, segmentation, etc. challenges

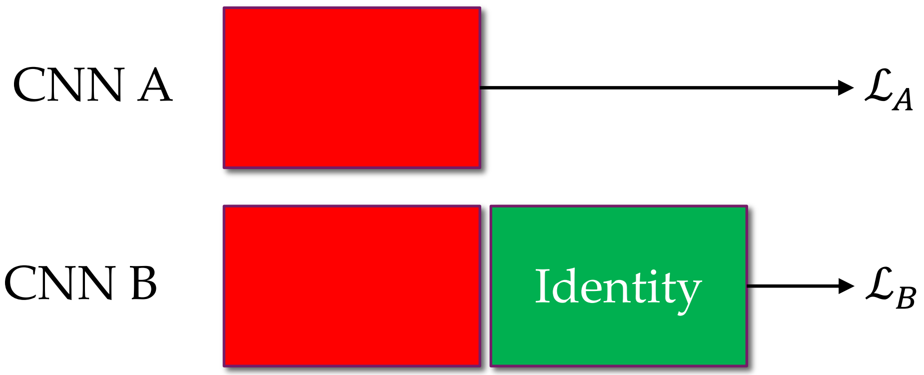
Hypothesis

- **Hypothesis:** Is it possible to have a very deep network at least as accurate as averagely deep networks?
- **Thought experiment:** Let's assume two Convnets A, B. They are almost identical, in that B is the same as A, with extra "identity" layers. Since identity layers pass the information unchanged, the errors of the two networks should …



Hypothesis

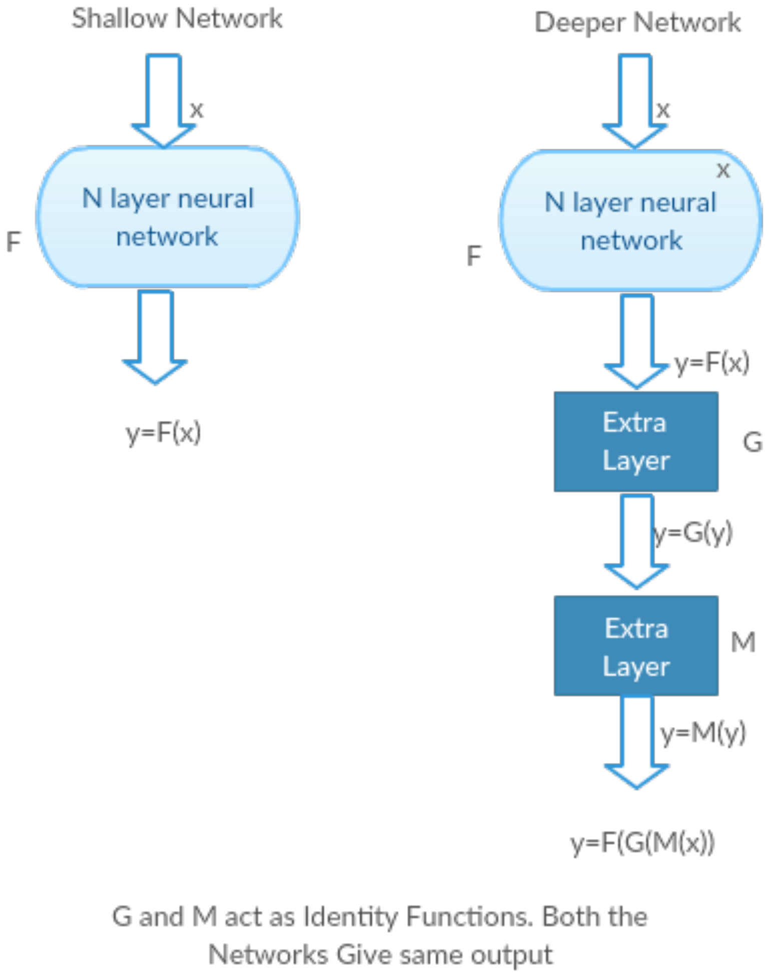
- **Hypothesis:** Is it possible to have a very deep network at least as accurate as averagely deep networks?
- **Thought experiment:** Let's assume two Convnets A, B. They are almost identical, in that B is the same as A, with extra "identity" layers. Since identity layers pass the information unchanged, the errors of the two networks should be similar. Thus, there is a Convnet B, which is at least as good as Convnet A w.r.t. training error



Hypothesis

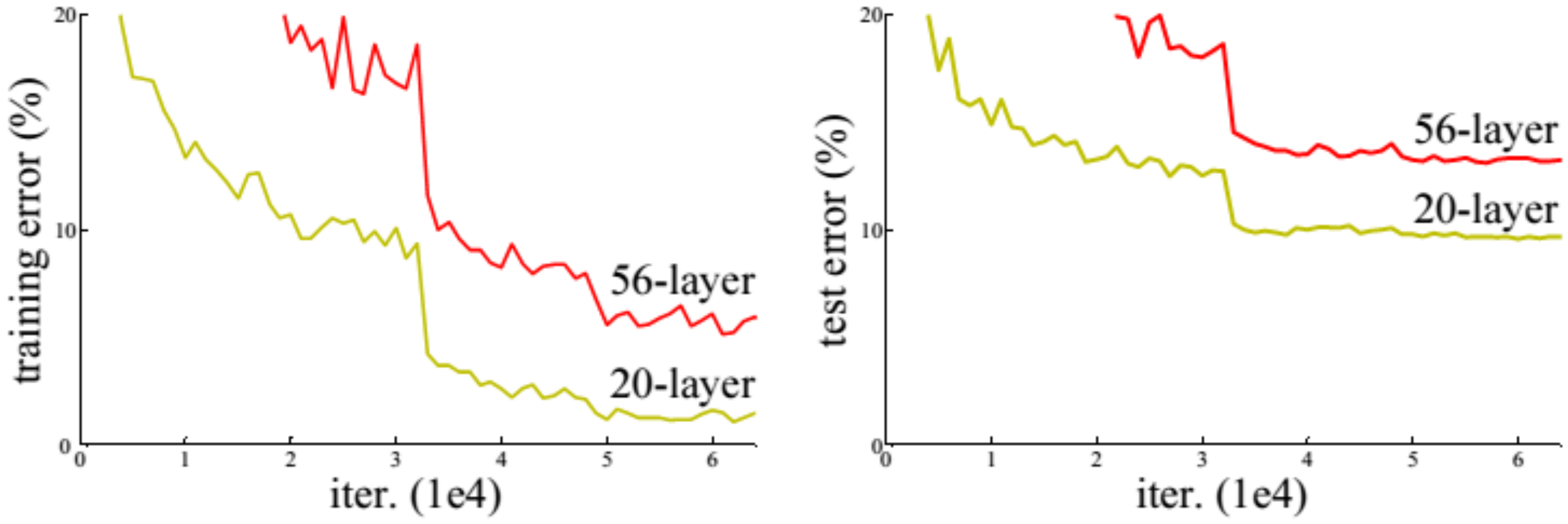
- We add a few extra layers on top of the N layers
- These few extra layers *can* learn identity mappings

$$y = x$$



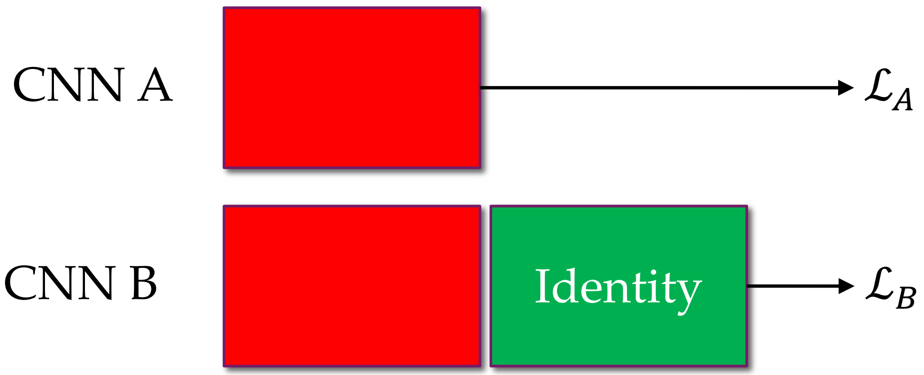
However, when trained the deeper network has higher training error

🤔 🤔 🤔 🤔



Testing the hypothesis

- Adding identity layers increases **training error**!
- **Performance degradation** not caused by overfitting
  - Just the optimization task is harder
- Assuming optimizers are doing their job fine, it appears that not all networks are similarly easy to optimize



$\mathcal{L}_A^{train} < \mathcal{L}_B^{train}$!!!

Observation

- Very deep networks stop learning after a bit
  - An accuracy is reached, then the network saturates and starts unlearning
- Signal gets lost through so many layers
- Models start failing

The “residual idea”, intuitively

- Let’s say we have the neural network nonlinearity a $= F(x)$
- Perhaps easier to learn a function a $= F(x)$ to model differences a~δy than to model absolutes a~y
  - Otherwise, you may need to model both the magnitude as well as the direction of activations
  - Think of it like in input normalization → you normalize around 0
  - Think of it like in regression → you model differences around the mean value
- “Residual idea”: Let neural networks explicitly model difference mappings

$$F(x) = H(x) - x \Rightarrow H(x) = F(x) + x$$

- $F(x)$ are the stacked nonlinearities
- x is the input to the nonlinear layer

The residual block

- $H(x) = F(x) + x$
- If dimensions don't match
  - Either zero padding
  - Or a projection layer to match dimensions

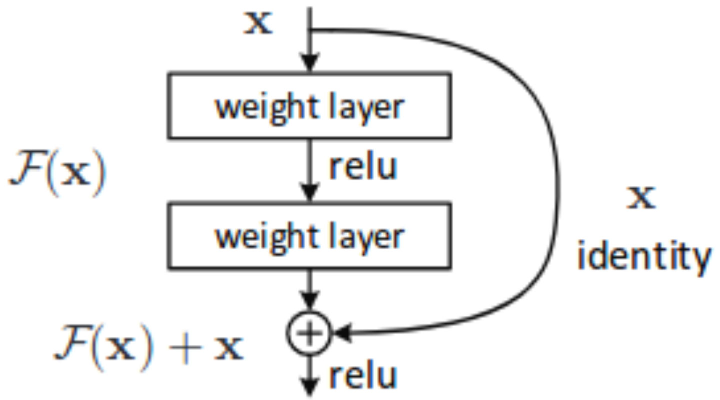
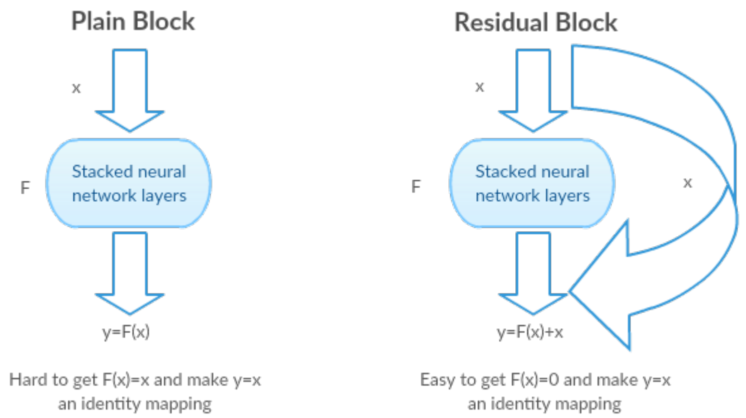
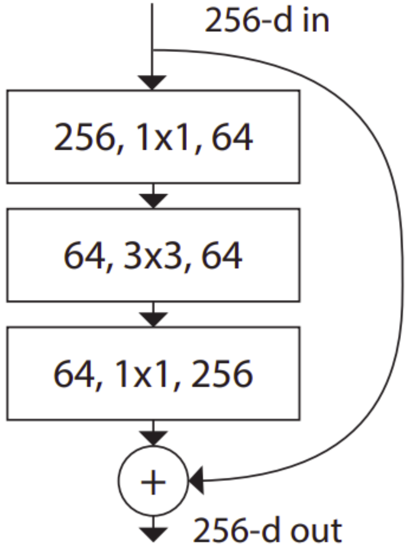


Figure 2. Residual learning: a building block.





Example

No degradation anymore

- Now there's a benefit (right plot)

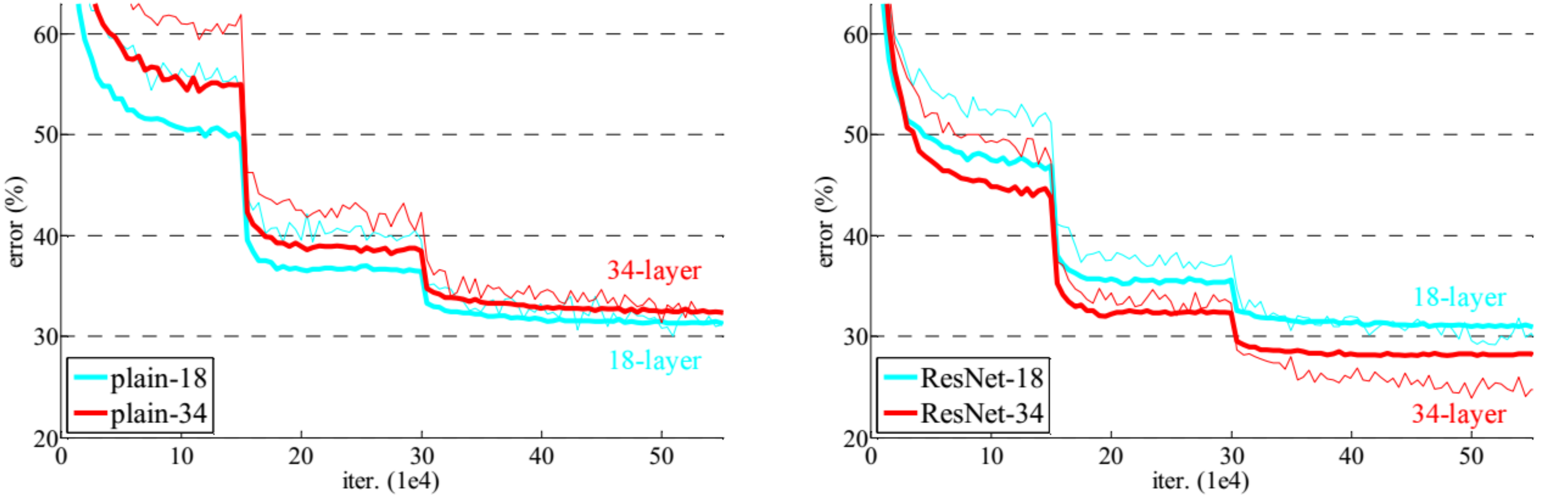


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

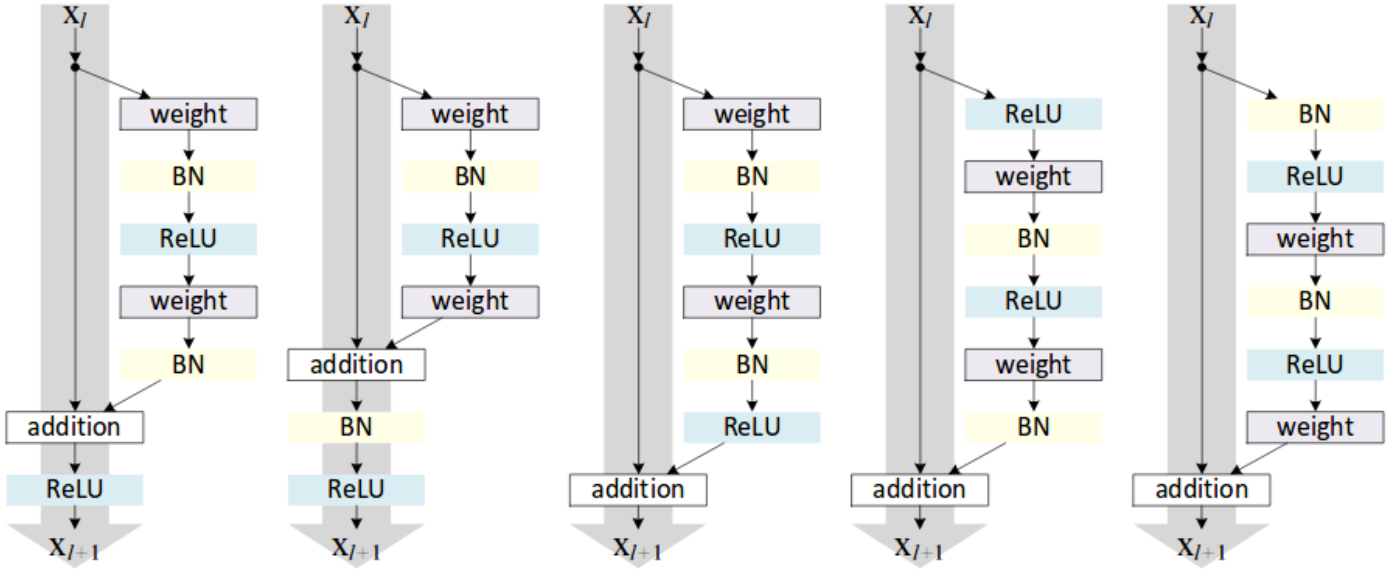
ResNet breaks records

- Very low error in ImageNet
- Up to 1000 layers ResNets trained
  - Previous deepest network ~30-40 layers on simple datasets

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

ResNet variants & ResNeXt



(a) original

(b) BN after addition

(c) ReLU before addition

(d) ReLU-only pre-activation

(e) **full pre-activation**

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

ResNeXt

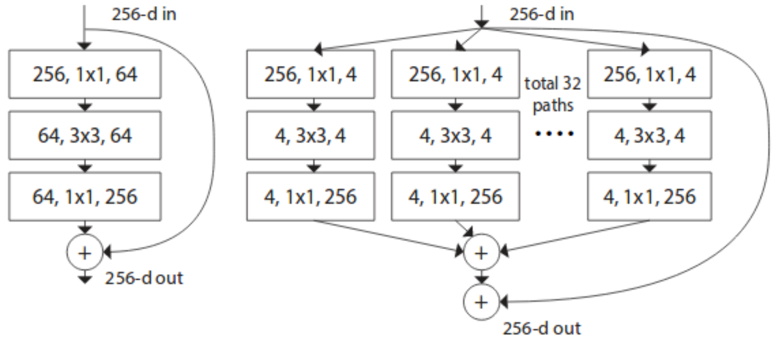


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

[Aggregated Residual Transformations for Deep Neural Networks, Xie et al., 2016]

Some observations

- Normalisations (BatchNorms) absolutely necessary because of vanishing gradients
- Networks with skip connections (like ResNets) converge faster than the same network without skip connections
- Identity shortcuts cheaper and almost equal to project shortcuts
- Most current architectures have skip connections (eg Transformers)

Quiz: On the right you see the forward function of a generic ResNet Block. What is True?

1) Only ~¼ of the values of out are non-zero after line 97

2) Writing line 102 as out= out+ identity will not propagate the right gradients

3) self.downsample calls a point-wise convolution

```
89      def forward(self, x: Tensor) -> Tensor:
90          identity = x
91
92          out = self.conv1(x)
93          out = self.bn1(out)
94          out = self.relu(out)
95
96          out = self.conv2(out)
97          out = self.bn2(out)
98
99          if self.downsample is not None:
100             identity = self.downsample(x)
101
102         out += identity
103         out = self.relu(out)
104
105         return out
```

https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

HighwayNet (slightly earlier than ResNets in 2015)

- Similar to ResNets, only introducing a gate with learnable parameters on the importance of each skip connection

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

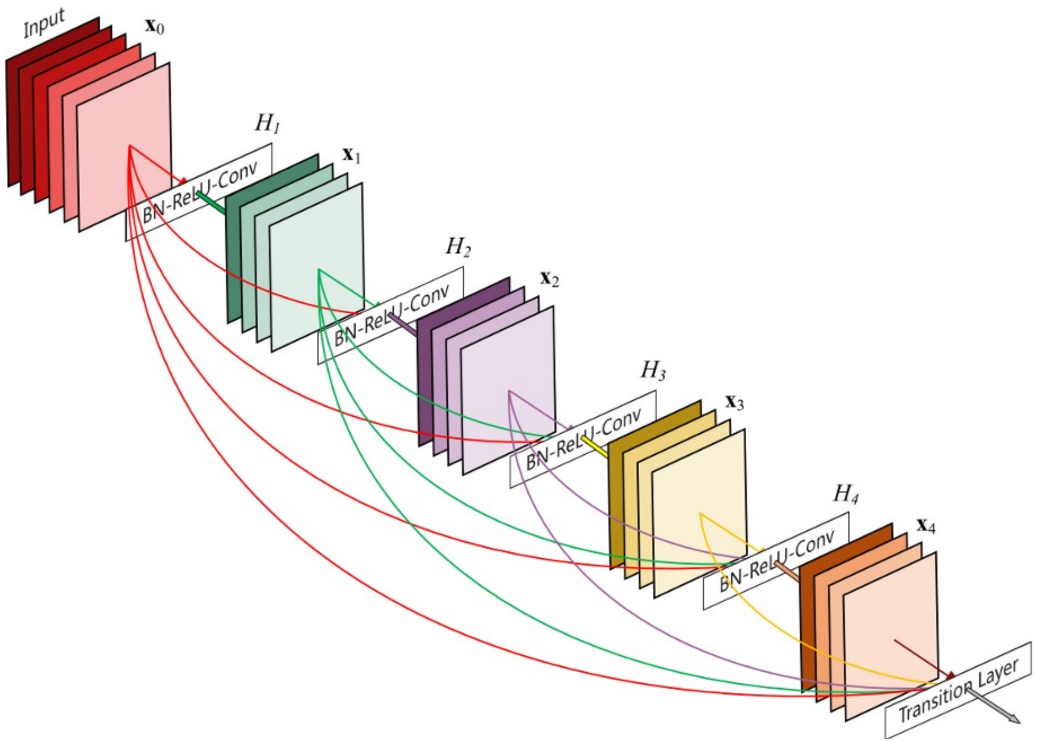
- Similar to LSTM

DenseNet

- Add skip connections to multiple forward layers

$$y = h(x_l, x_{l-1}, \dots, x_{l-n})$$

- Why?

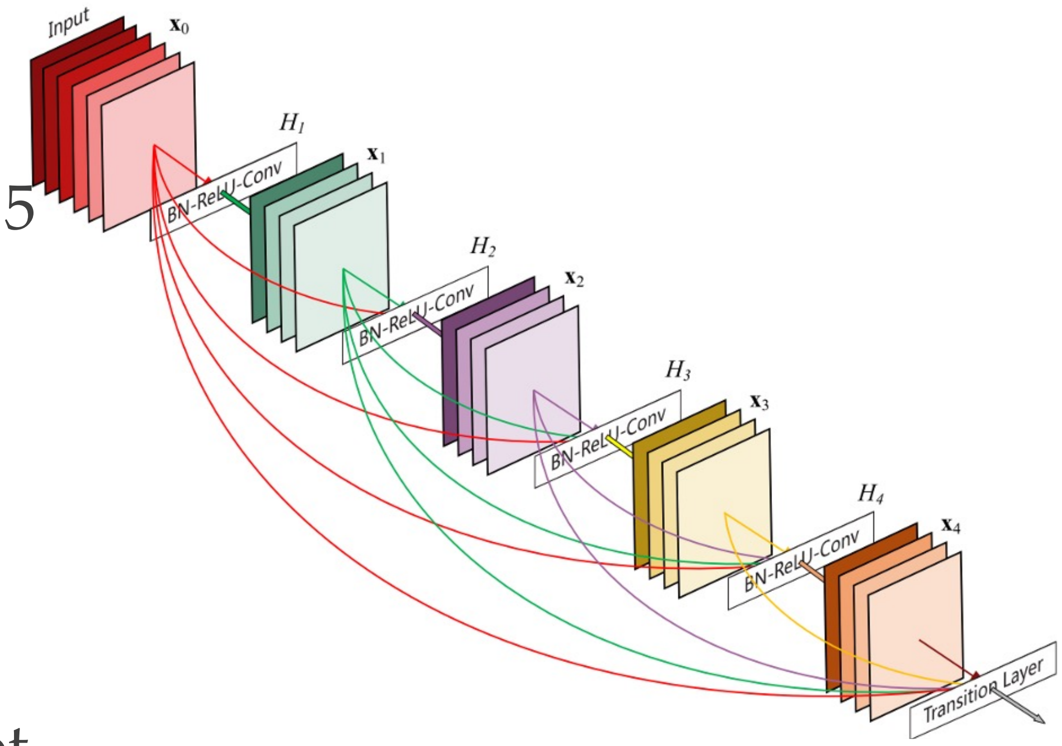


DenseNet

- Add skip connections to multiple forward layers

$$y = h(x_l, x_{l-1}, \dots, x_{l-n})$$

- Assume layer 1 captures edges, while layer 5 captures faces (and other stuff)
- Why not have a layer that combines both faces and edges (e.g. to model "a scarred face")
- Standard ConvNets do not allow for this
    - Layer 6 combines only layer 5 patterns, not lower

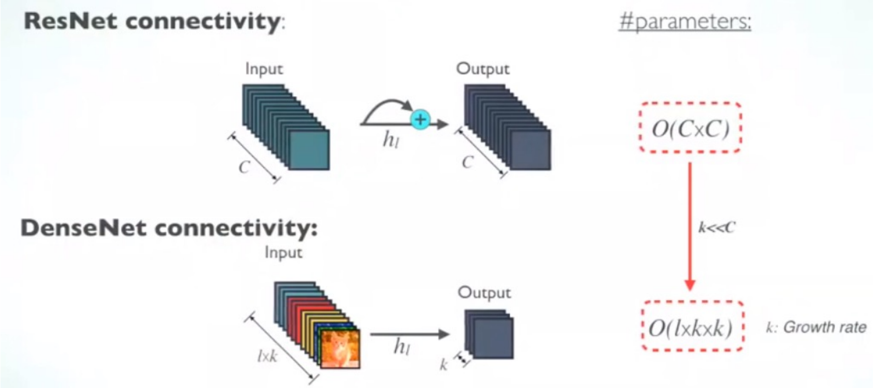


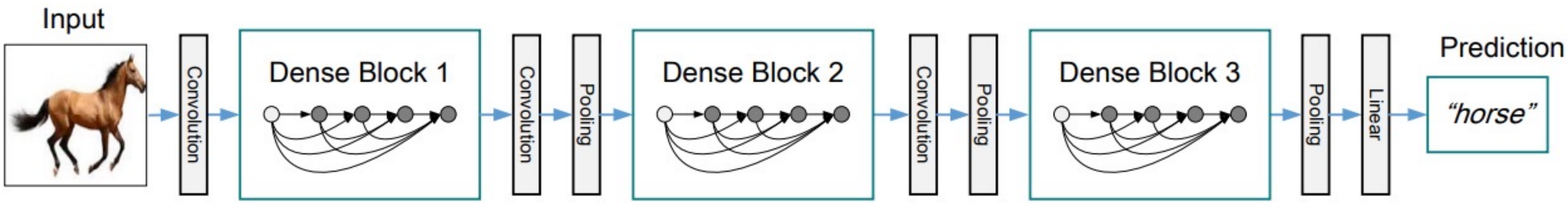
DenseNet

- Each layer is receiving a “collective knowledge” from all preceding layers.

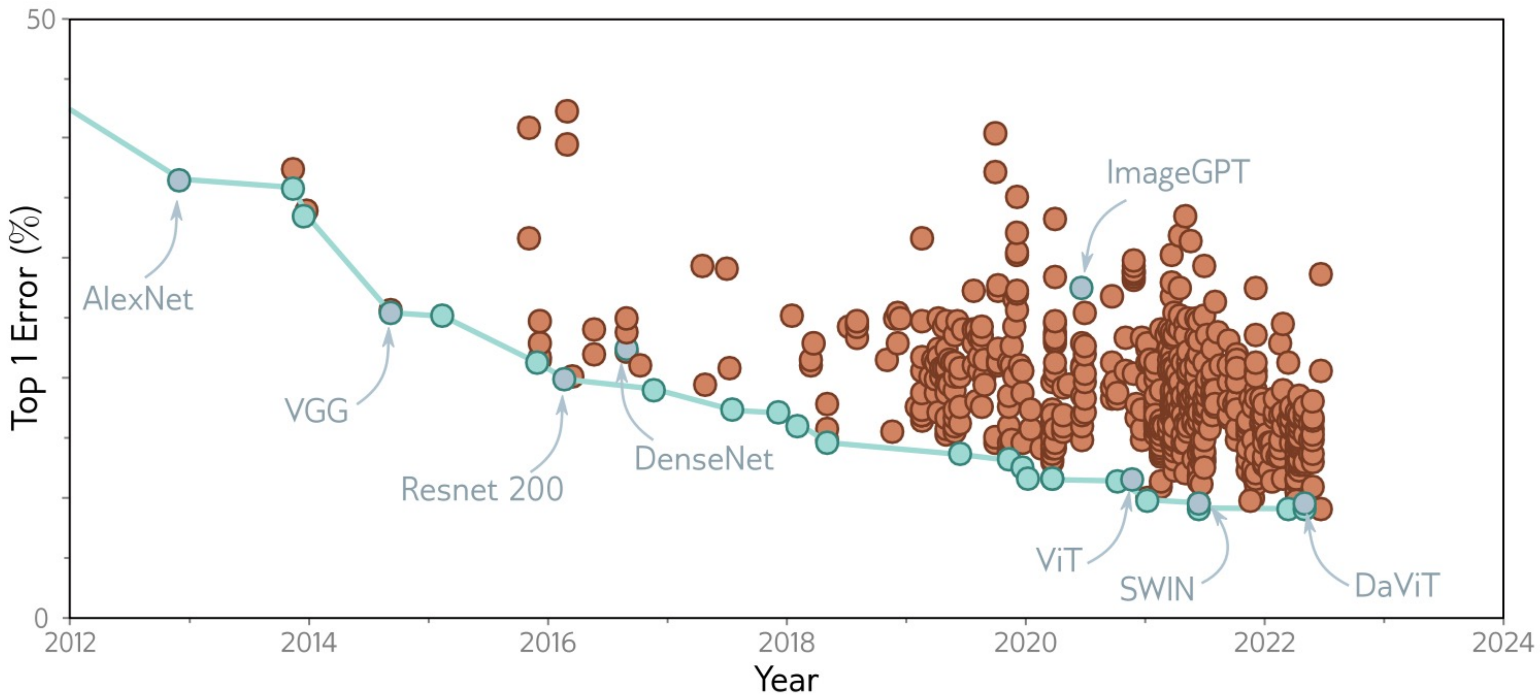
DenseNets

- DenseNets have several compelling advantages:
  - they alleviate the vanishing-gradient problem,
  - strengthen feature propagation,
  - encourage feature reuse,
  - and reduce the number of parameters -->



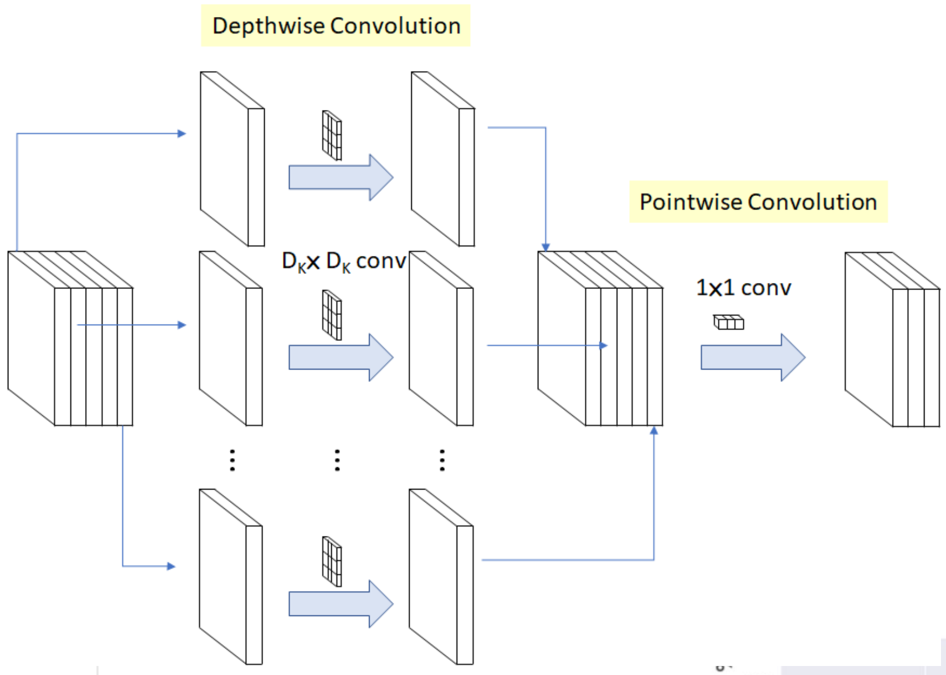


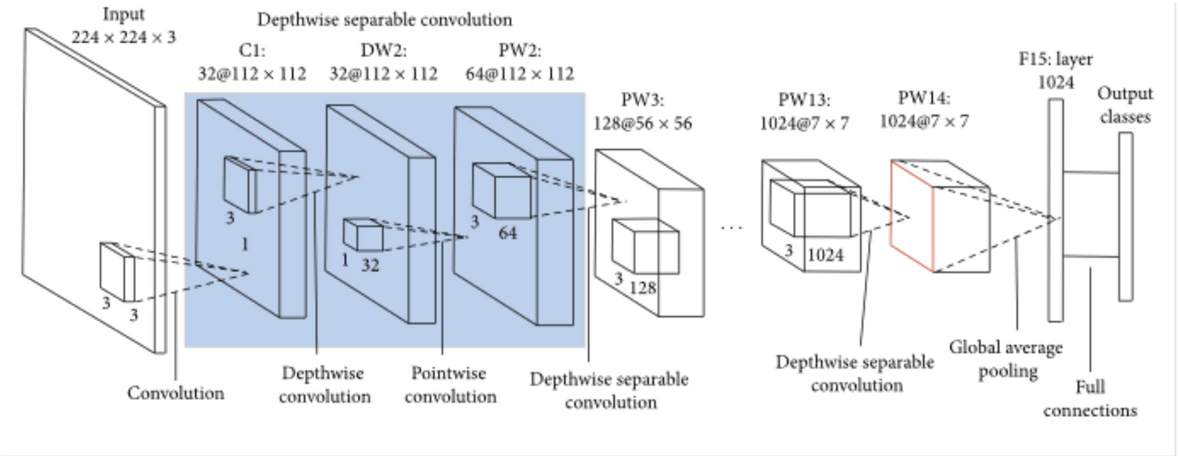
Trend has not stopped with DenseNet

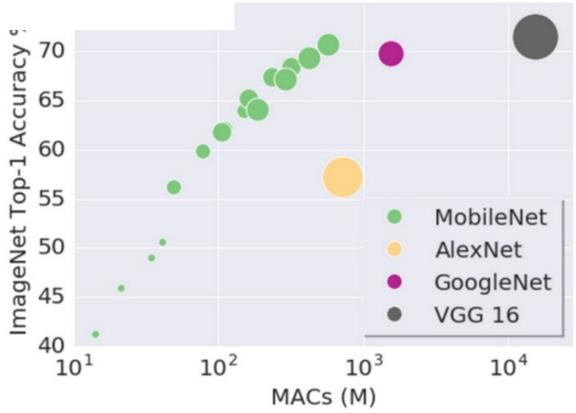


MobileNets: Depthwise convolutions for high latency

- Depthwise convolutions replace normal convolutions
- Less parameters
- Increased throughput/latency



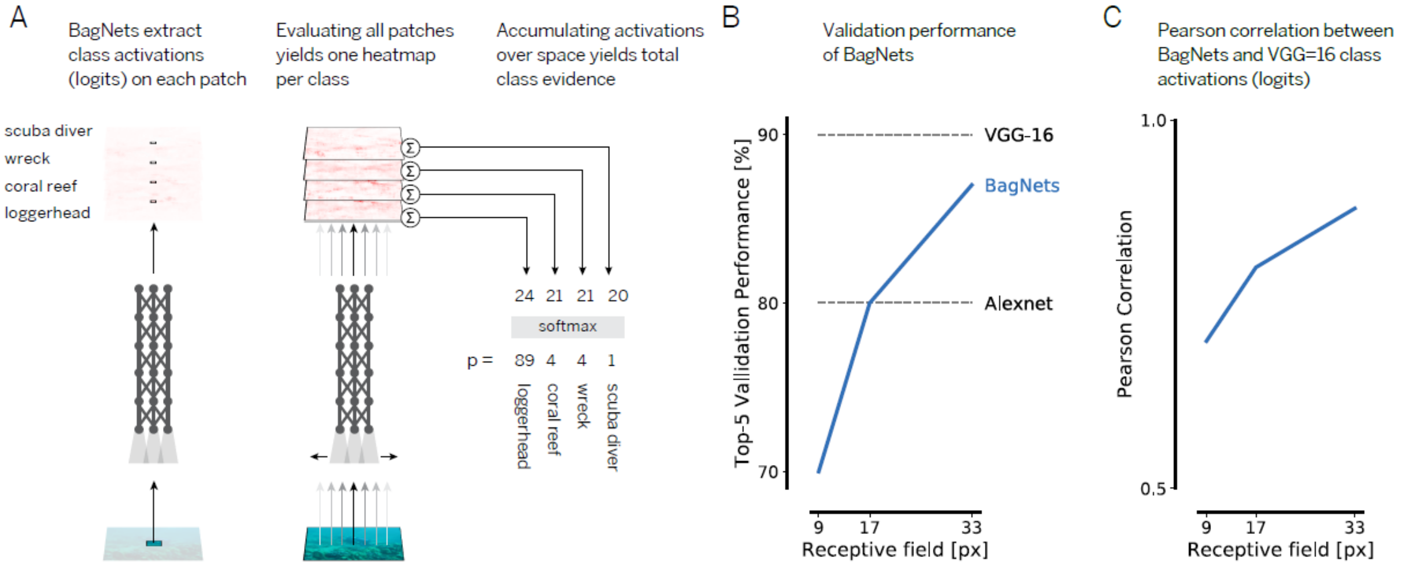




MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Howard et al. 2017

BagNet: Solving ImageNet with tiny 9x9 sized puzzle pieces?

- Model is an adaptation of ResNet: instead of 3x3, strictly 1x1 convs

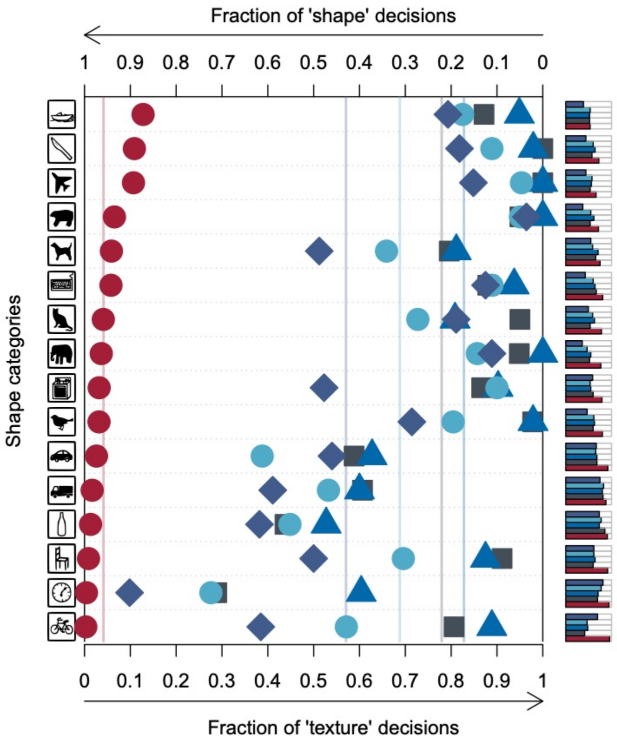


- High performance demonstrates that ImageNet can be solved (fairly well) with just *local patterns*!

Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet. Brendel & Bethge, ICLR 2019

ImageNet: mostly textures?

Figure 4: Classification results for human observers (red circles) and ImageNet-trained networks AlexNet (purple diamonds), VGG-16 (blue triangles), GoogLeNet (turquoise circles) and ResNet-50 (grey squares). Shape vs. texture biases for stimuli with cue conflict (sorted by human shape bias). Within the responses that corresponded to either the correct texture or correct shape category, the fractions of texture and shape decisions are depicted in the main plot (averages visualised by vertical lines). On the right side, small barplots display the proportion of correct decisions (either texture or shape correctly recognised) as a fraction of all trials. Similar results for ResNet-152, DenseNet-121 and Squeezenet1_1 are reported in the Appendix, Figure 13.



(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. Geirhos et al. ICLR 2018

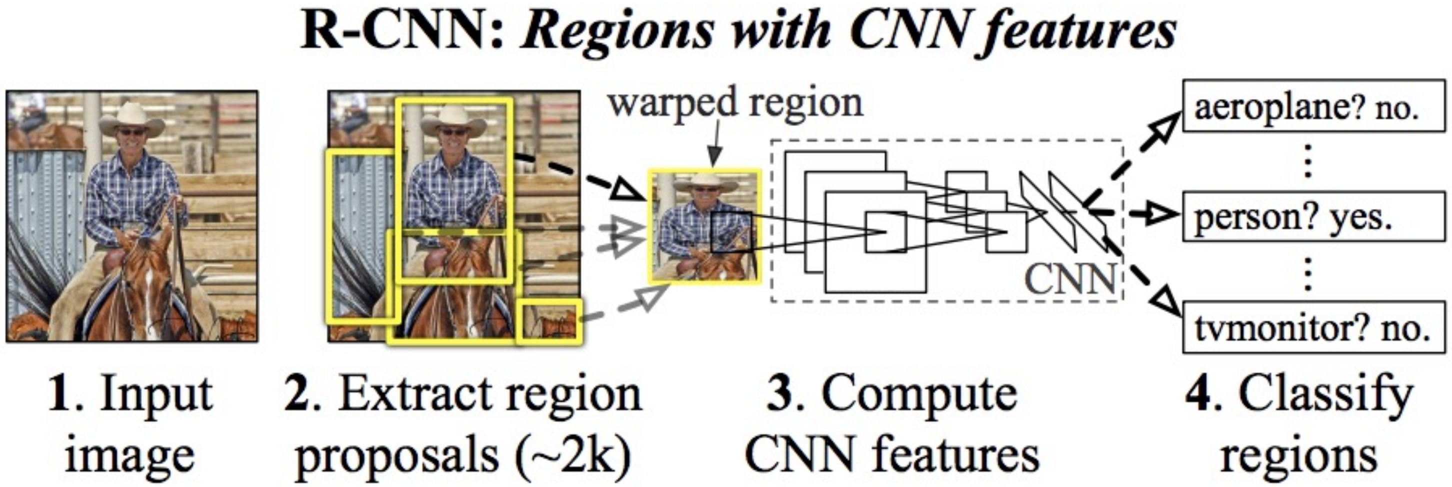
How research gets done part 5

Previous parts:
[fundamental understanding, read papers, how-to-read-papers, implement & tinker with code, realise and seek *funny* moments]

Today:

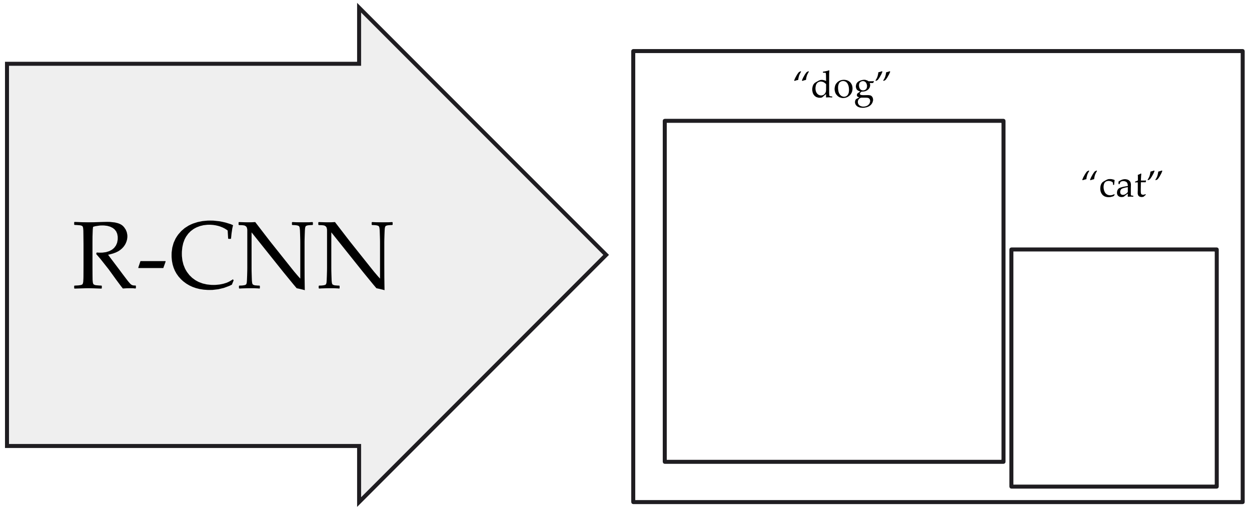
- We found something funny (or we have an inkling that there's something interesting here)
- How do we analyse this further?
  1. Establish benchmarks (we cannot know what we cannot measure)
  2. Establish baselines (is 58% good or bad?)
  3. Figure out what the *minimum viable proof of principle* is: if this works, it shows our idea is right
  4. Compare and contrast this to existing ideas:
     1. Why might it work?
     2. Why not?
     3. What are the *principles* at play here?
  5. Next times: when to give up and when not, how to design ablations

R-CNNs & Fully Convolutional Networks



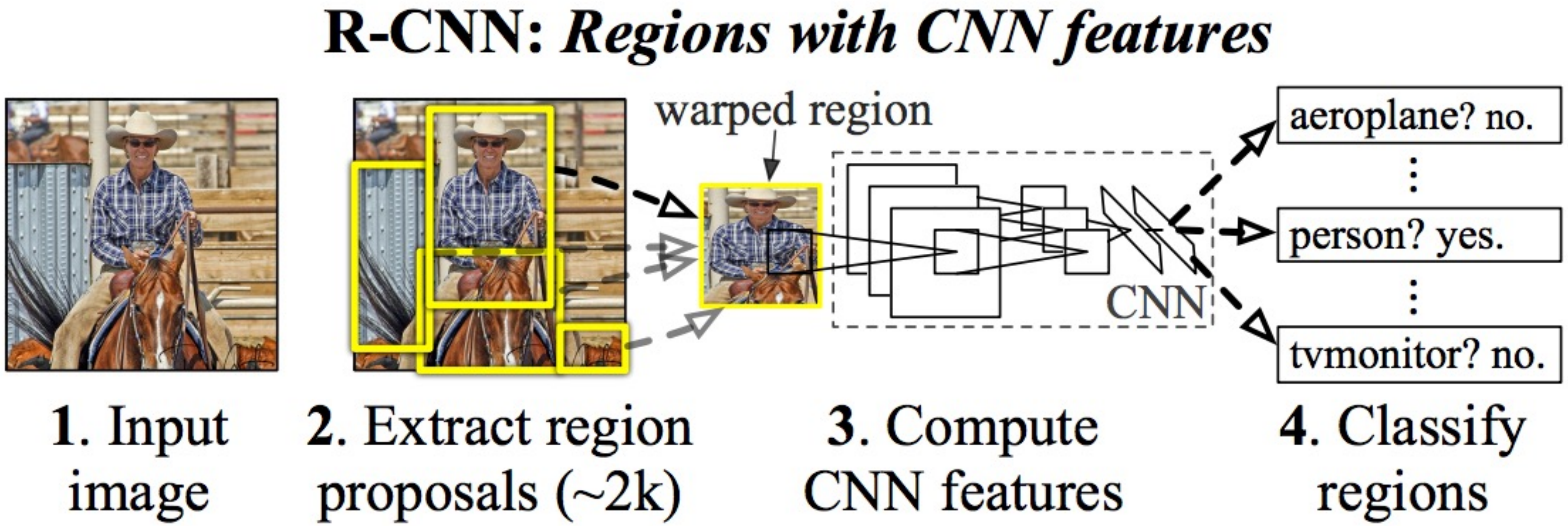
Region-based Convolutional Neural Network (R-CNN)

- The goal of R-CNN is to take in an image, and correctly identify where the main objects (via a bounding box) in the image.
- This task is called *object detection*
- R-CNN creates these bounding boxes, or region proposals, using a process called Selective Search.
- At a high level, Selective Search looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects.



R-CNN

- Once the proposals (~2K) are created, R-CNN warps the region to a standard square size and passes it through to a modified version of AlexNet.
- On the final layer of the CNN, R-CNN adds a Support Vector Machine (SVM) that simply classifies whether this is an object, and if so what object.



Improving the Bounding Boxes

- Now, having found the object in the box, can we tighten the box to fit the true dimensions of the object?
- We can, and this is the final step of R-CNN.
- R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates to get the final result.
  - Inputs: sub-regions of the image corresponding to objects.
  - Outputs: New bounding box coordinates for the object in the sub-region.

To summarize

R-CNN is just the following steps

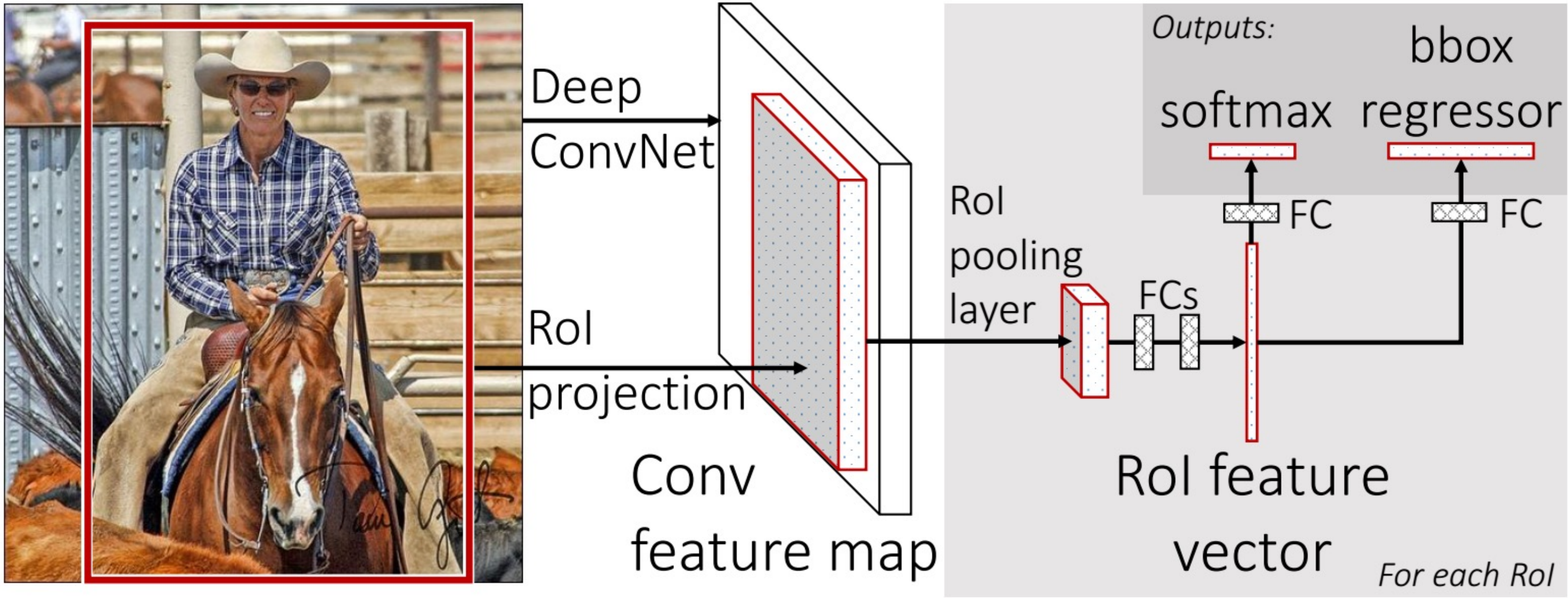
- Generate a set of proposals for bounding boxes.
- Run the images in the bounding boxes through a pre-trained AlexNet and finally an SVM to see what object the image in the box is.
- Run the box through a linear regression model to output tighter coordinates for the box once the object has been classified.

R-CNN is really quite slow for a few simple reasons:

- It requires a forward pass of the CNN (AlexNet) for every single region proposal for every single image
  - (that’s around 2000 forward passes per image!).
- It has to train three different models separately
  - the CNN to generate image features,
  - the classifier that predicts the class,
  - and the regression model to tighten the bounding boxes.
  - This makes the pipeline extremely hard to train.

Fast R-CNN

- Fast R-CNN [Girshick2015]



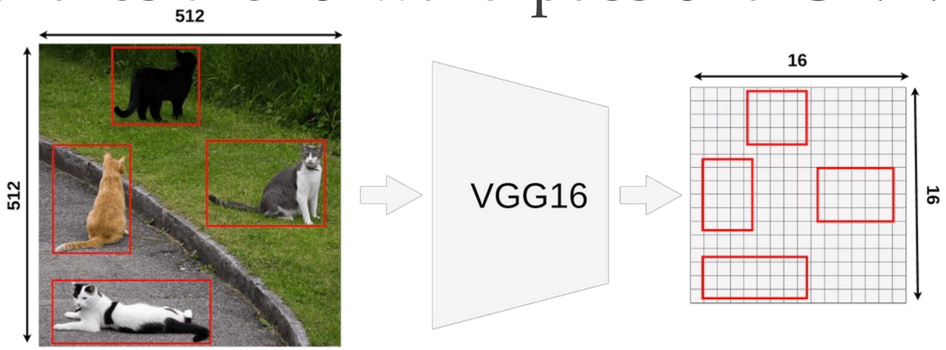
Fast R-CNN Insight 1: Region of Interest Pooling (ROIPool)

- For the forward pass of the CNN, a lot of proposed regions for the image invariably overlapped …
  - … causing us to run the same CNN computation again and again (~2000 times!).

- Why not run the CNN just once per image and then find a way to share that computation across the ~2000 proposals?

Region of Interest Pooling (ROIPool)

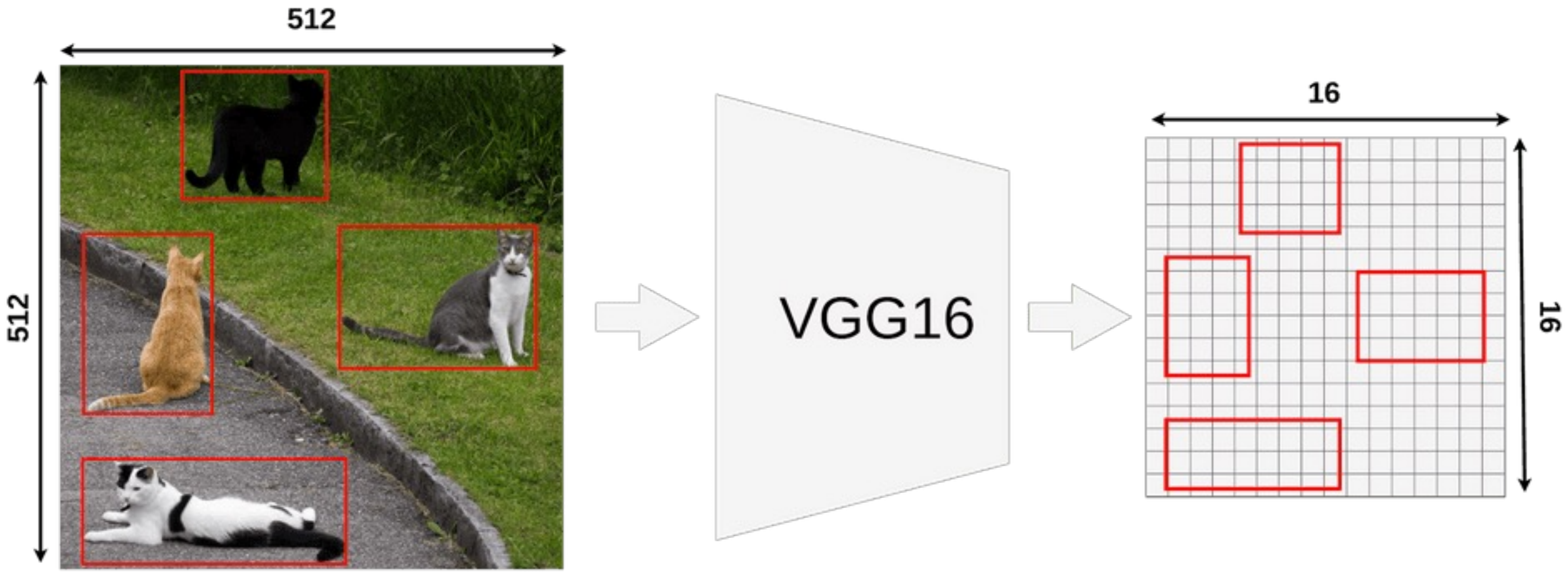
- At its core, RoIPool shares the forward pass of a CNN for an image across its subregions.



- In the image above, notice how the CNN features for each region are obtained by selecting a corresponding region from the CNN's feature map.

- Then, the features in each region are pooled (usually using max pooling). So all it takes us is one pass of the original image as opposed to ~2000!

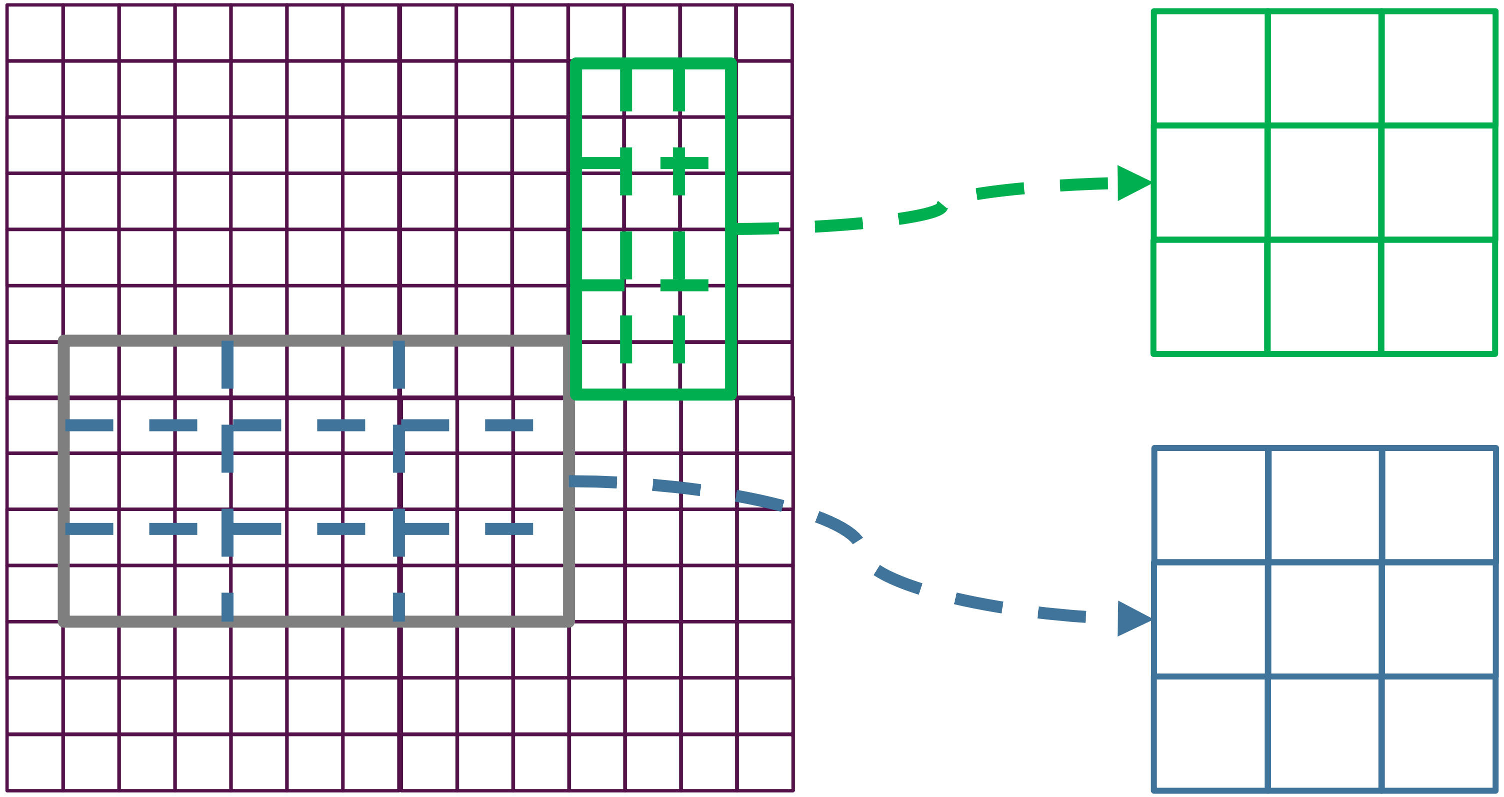
Region of Interest Pooling (ROIPool)



Mapping RoIs onto the output of VGG16

Region of Interest Pooling (ROIPool)

- Divide feature map in TxT cells
  - The cell size will change depending on the size of the candidate location



Always 3x3 no matter the size of candidate location

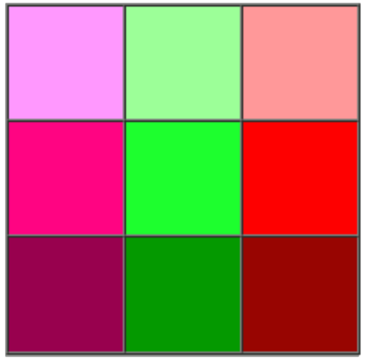
Region of Interest Pooling (ROIPool)

- Max pooling

4x6 RoI

0.1	0.2	0.3	0.4	0.5	0.6
1	0.7	0.2	0.6	0.1	0.9
0.9	0.8	0.7	0.3	0.5	0.2

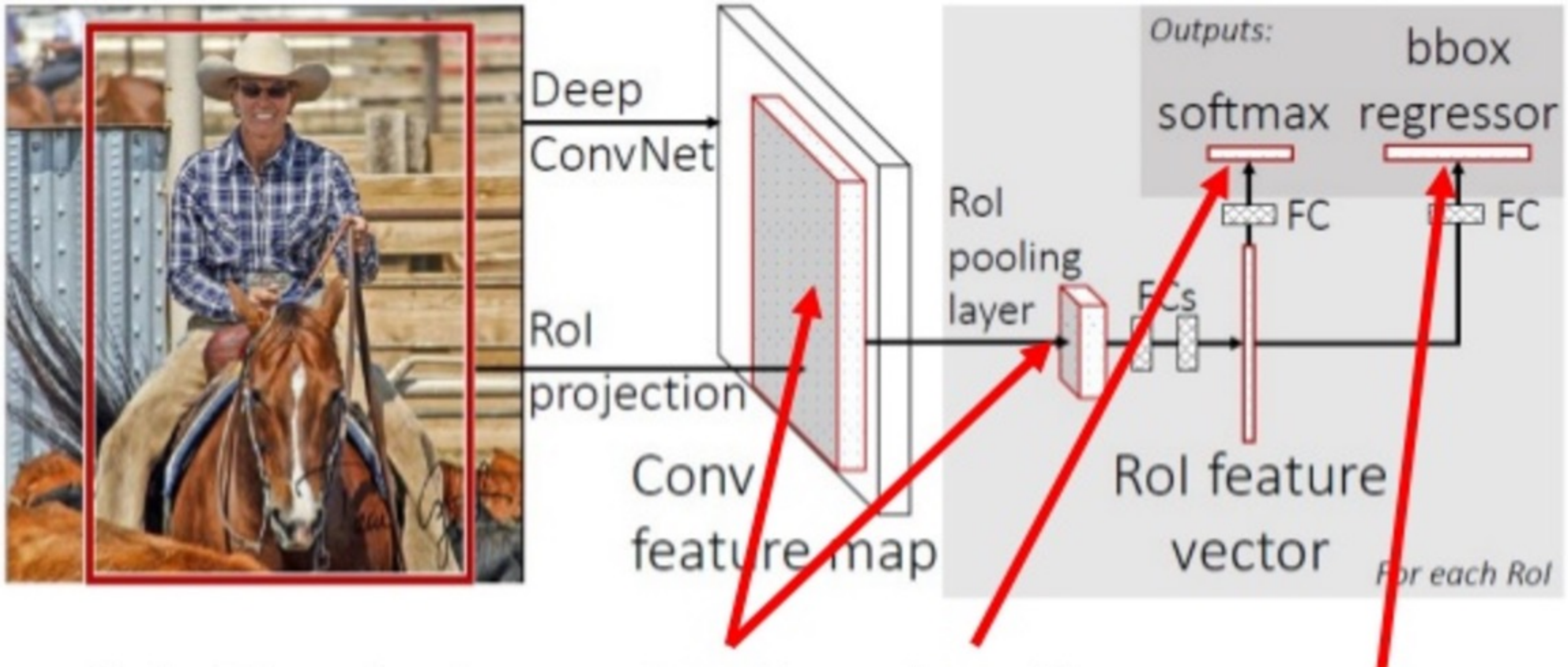
3x3 RoI Pooling



Fast R-CNN Insight 2: Combine All Models into One Network

- The second insight of Fast R-CNN is to jointly train the CNN, classifier, and bounding box regressor in a single model.
- Where earlier we had different models to extract image features (CNN), classify (SVM), and tighten bounding boxes (regressor),
- Fast R-CNN instead used a single network to compute all three.

Fast R-CNN: Joint training framework



Joint the feature extractor, classifier, regressor together in a unified framework

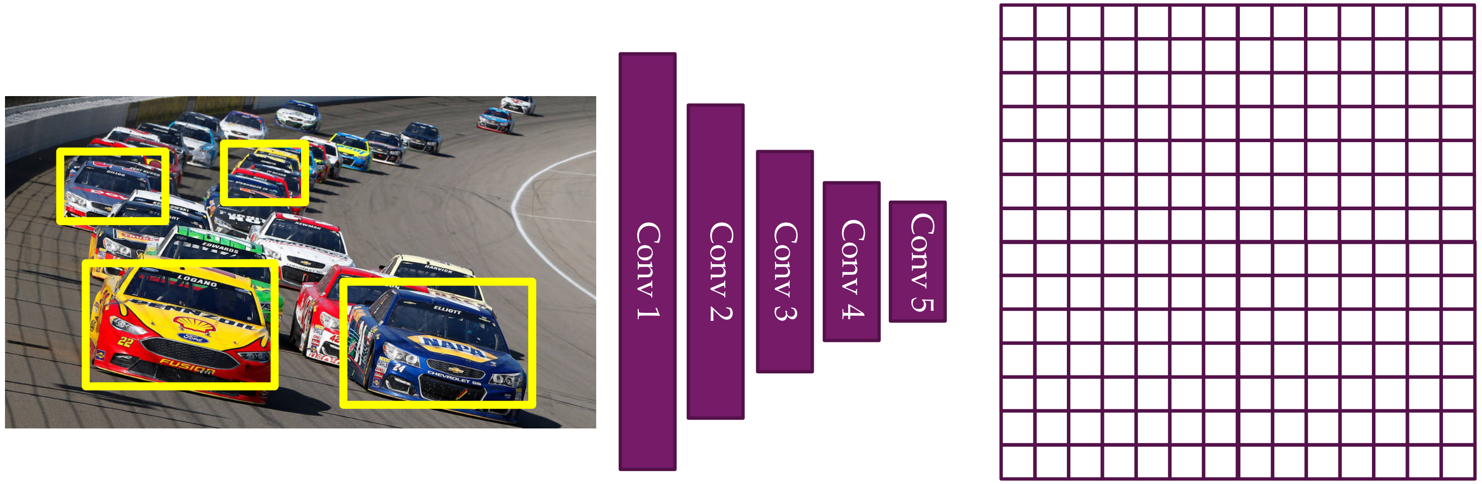
Fast R-CNN: Steps

- Process the whole image up to conv5

Conv 5 feature map

Fast R-CNN: Steps

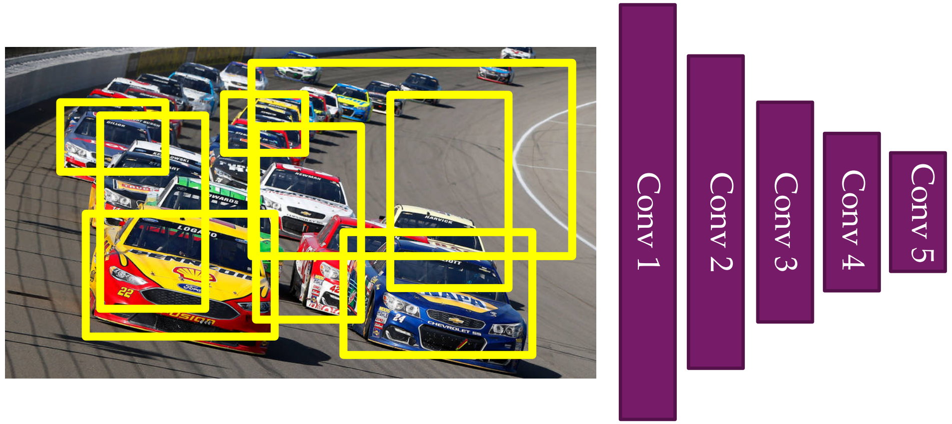
- Process the whole image up to conv5
- Compute possible locations for objects

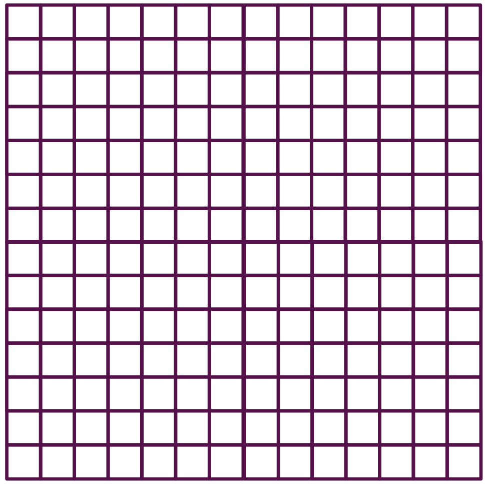


Conv 5 feature map

Fast R-CNN: Steps

- Process the whole image up to conv5
- Compute possible locations for objects (some correct, most wrong)

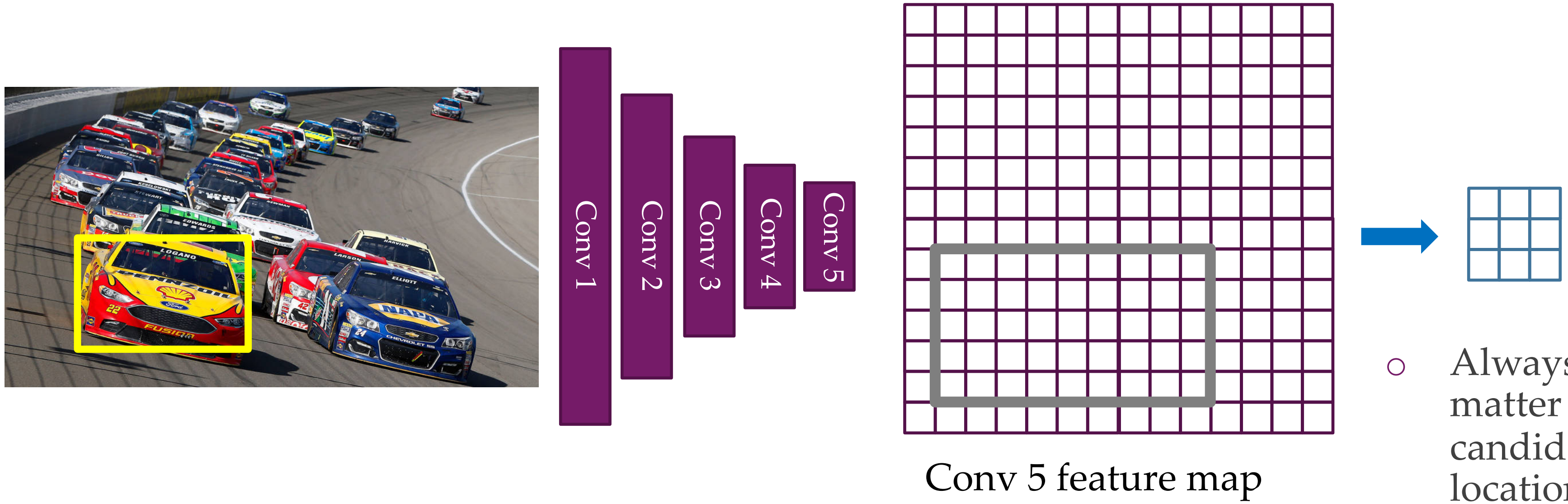




Conv 5 feature map

Fast R-CNN: Steps

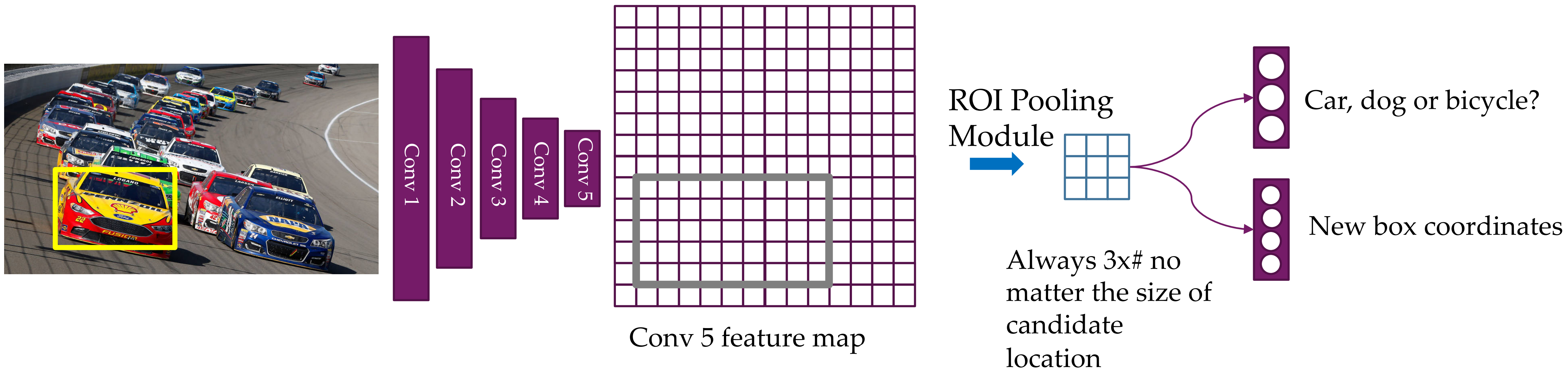
- Process the whole image up to conv5
- Compute possible locations for objects
- Given single location → ROI pooling module extracts fixed length feature



- Always 3x3 no matter the size of candidate location

Fast R-CNN: Steps

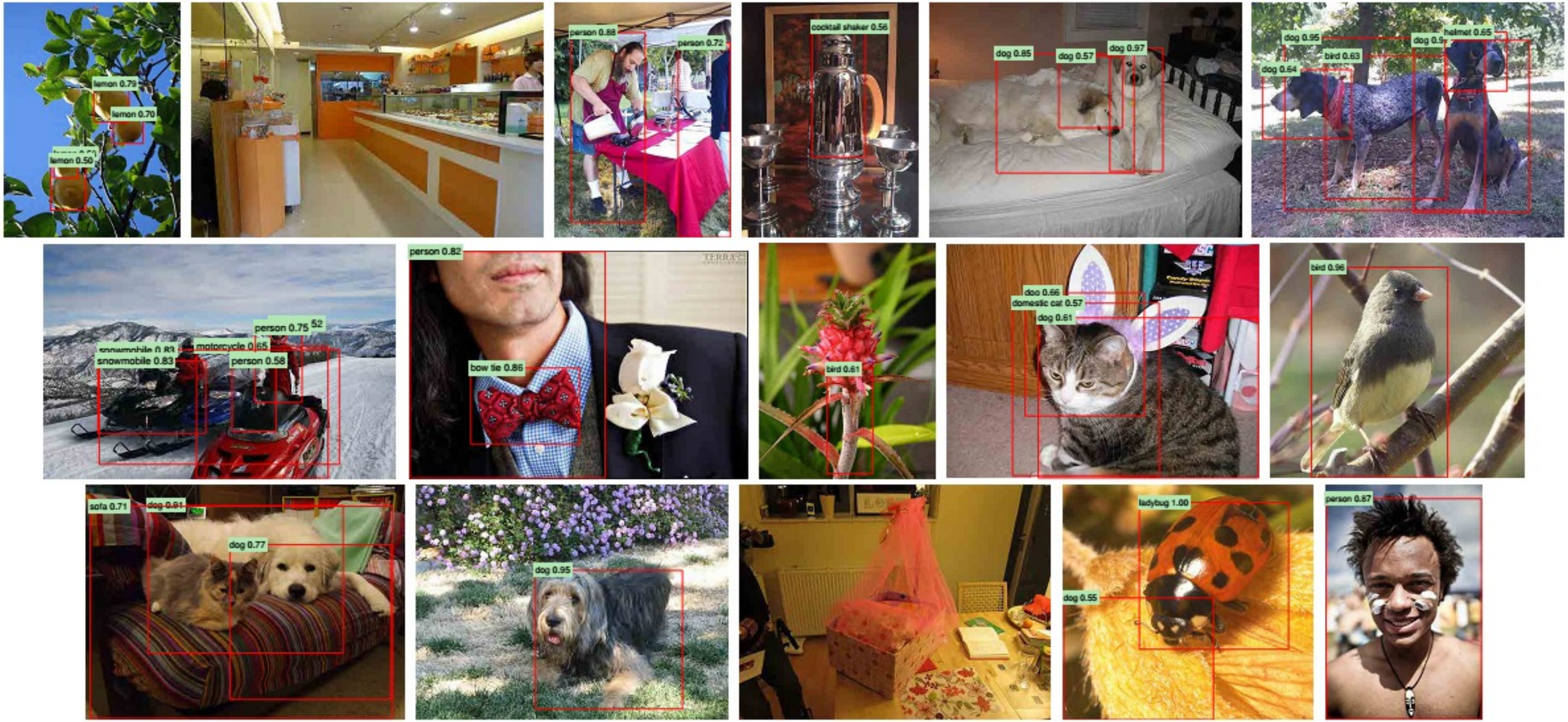
- Process the whole image up to conv5
- Compute possible locations for objects
- Given single location → ROI pooling module extracts fixed length feature
- Connect to two final layers, 1 for classification, 1 for box refinement



Smart training

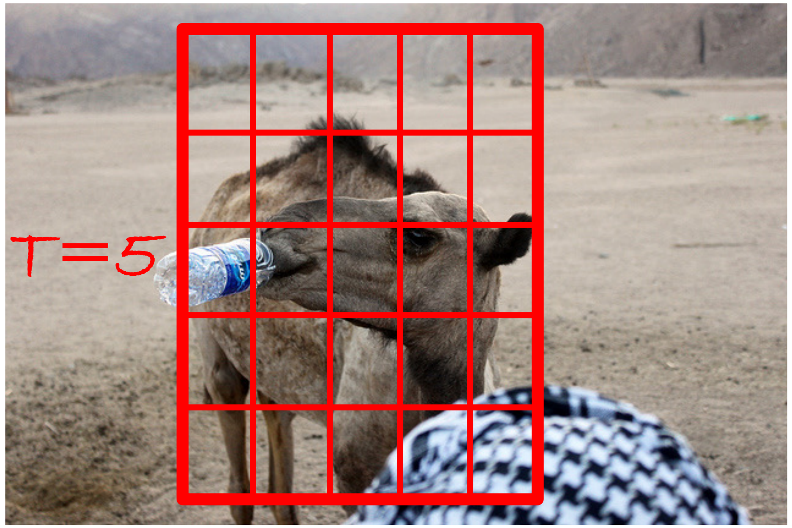
- Normally samples in a mini-batch completely random
- Instead, organize mini-batches by ROIs
- 1 mini-batch = N (images) $\times \frac{R}{N}$ (candidate locations)
- Feature maps shared → training speed-up by a factor of $\frac{R}{N}$

Some results



Fast-RCNN

- Reuse convolutions for different candidate boxes
  - Compute feature maps only once
- Region-of-Interest pooling
  - Define stride relatively → box width divided by predefined number of "poolings" T
  - Fixed length vector
- "End-to-end" training!
- (Very) Accurate object detection
- (Very) Faster
  - Less than a second per image
- External box proposals needed



Faster R-CNN - Speeding Up Region Proposal

- Even with all these advancements, there was still one remaining bottleneck in the Fast R-CNN process — the region proposer.
- As we saw, the very first step to detecting the locations of objects is generating a bunch of potential bounding boxes or regions of interest to test.
- In Fast R-CNN, these proposals were created using Selective Search, a fairly slow process that was found to be the bottleneck of the overall process.

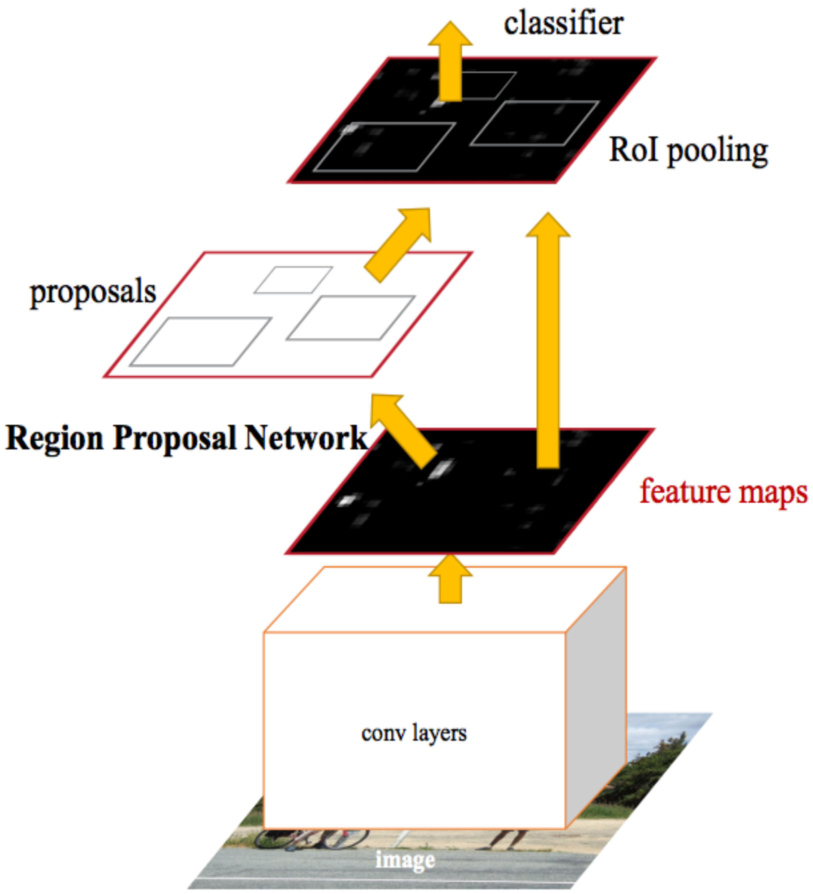
Faster R-CNN

- The insight of Faster R-CNN was that region proposals depended on features of the image that were already calculated with the forward pass of the CNN (first step of classification).

- So why not reuse those same CNN results for region proposals instead of running a separate selective search algorithm?

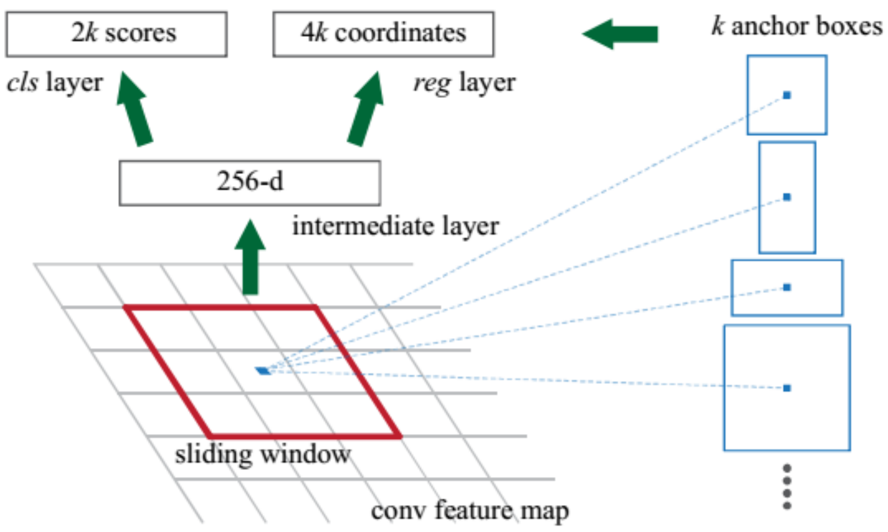
Faster R-CNN

- Faster R-CNN adds a Fully Convolutional Network on top of the features of the CNN creating what's known as the Region Proposal Network.
- The Region Proposal Network works by passing a sliding window over the CNN feature map and at each window, outputting *k* potential bounding boxes and scores for how good each of those boxes is expected to be.
- In such a way, we create *k* such common aspect ratios we call anchor boxes. For each such anchor box, we output one bounding box and score per position in the image.
- We then pass each such bounding box that is likely to be an object into Fast R-CNN to generate a classification and tightened bounding boxes.



Faster R-CNN [Girshick2016]

- Fast R-CNN → external candidate locations
- Faster R-CNN → deep network proposes candidate locations
- Slide the feature map → k anchor boxes per slide



Region Proposal Network

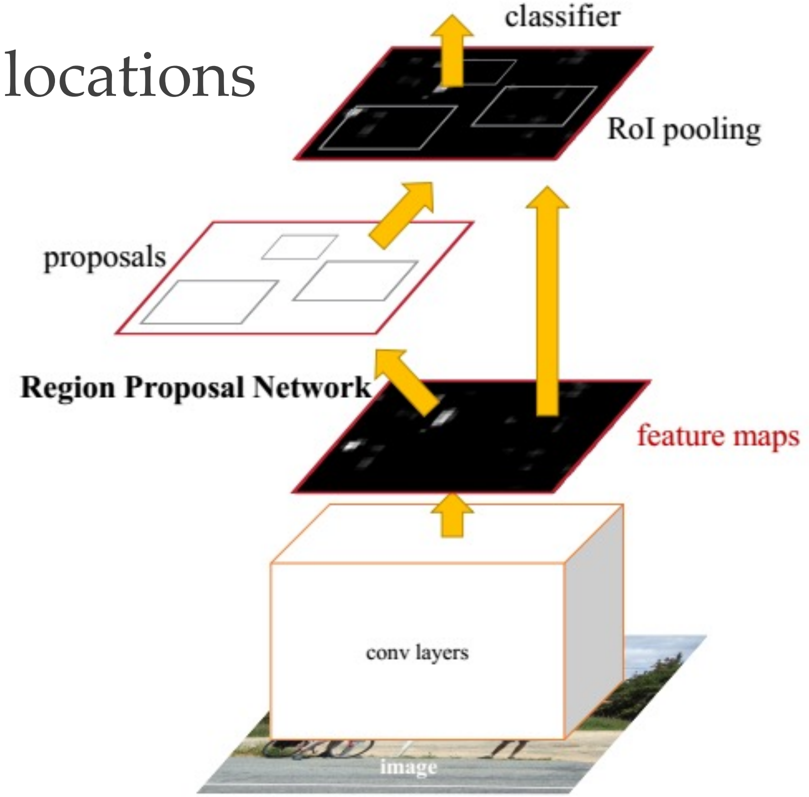


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Mask R-CNN

- Extending Faster R-CNN for Pixel Level Segmentation

- So far, we've seen how we've been able to use CNN features in many interesting ways to effectively locate different objects in an image with bounding boxes.

- Can we extend such techniques to go one step further and locate exact pixels of each object instead of just bounding boxes?

- This problem is known as *image segmentation*

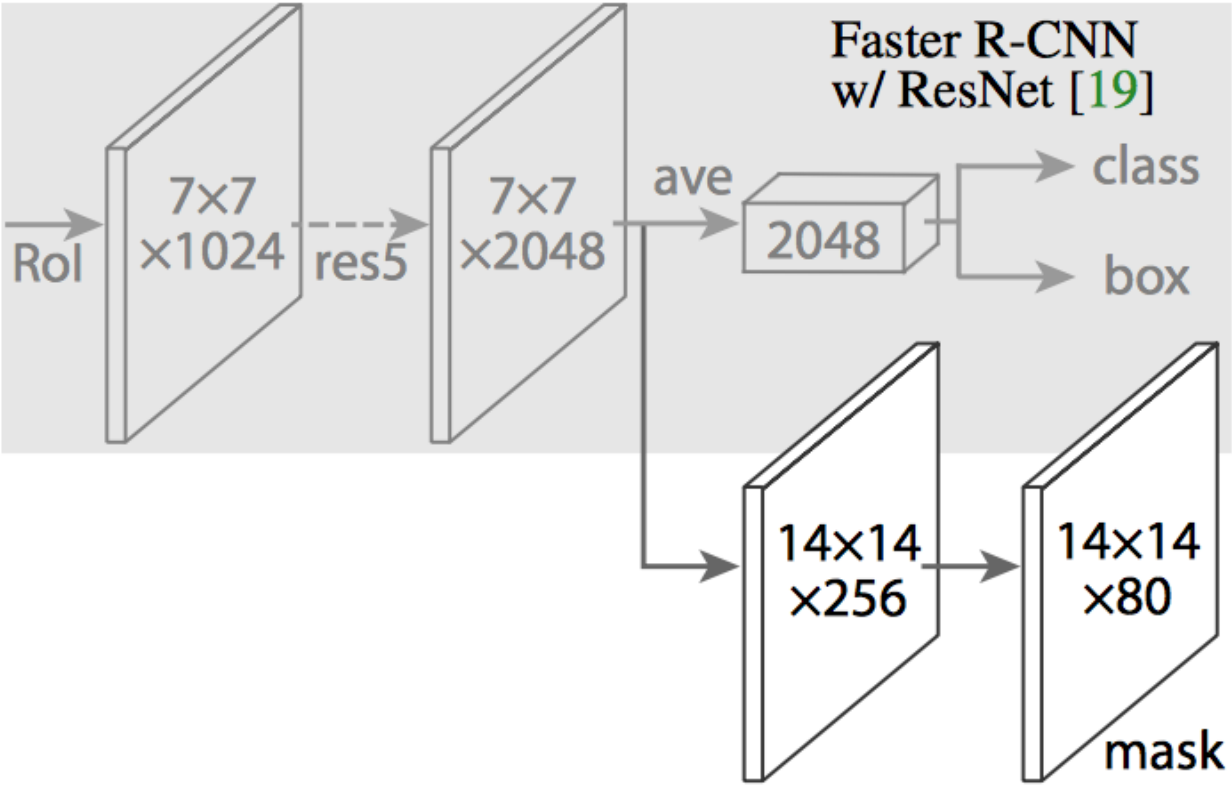
Mask R-CNN. He et al. ICCV 2017

Mask R-CNN

- Given that Faster R-CNN works so well for object detection, could we extend it to also carry out pixel level segmentation?
- Mask R-CNN does this by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object.
- Here are its inputs and outputs:
  - Inputs: CNN Feature Map.
  - Outputs: Matrix with 1s on all locations where the pixel belongs to the object and 0s elsewhere (this is known as a binary mask).

Mask R-CNN

- The branch (in white in the above image), as before, is just a Fully Convolutional Network on top of a CNN based feature map.



RoIAlign - Realigning RoIPool to be More Accurate

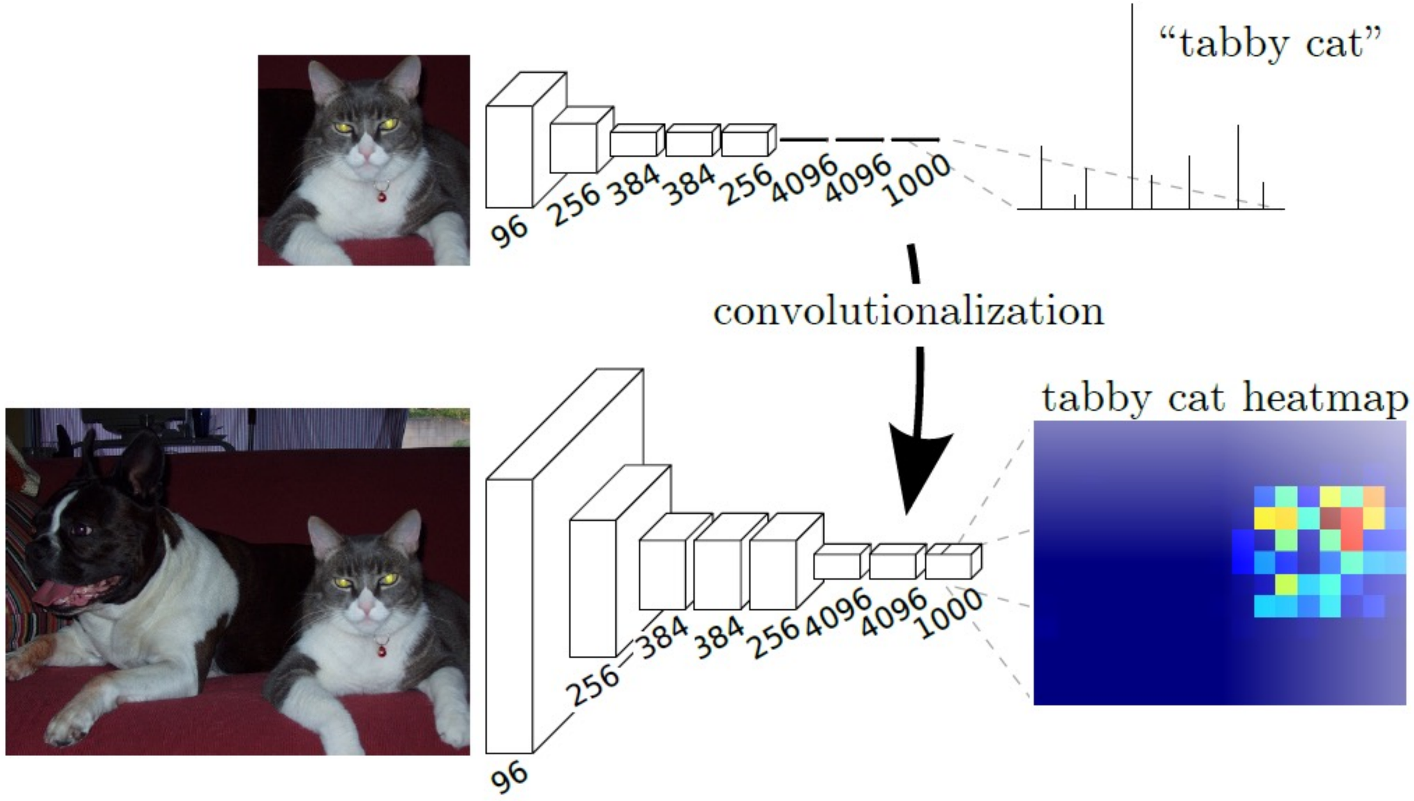
- When run without modifications on the original Faster R-CNN architecture, the Mask R-CNN authors realized that the regions of the feature map selected by RoIPool were slightly misaligned from the regions of the original image.

- Since image segmentation requires pixel level specificity, unlike bounding boxes, this naturally led to inaccuracies.

- The authors were able to solve this problem by cleverly adjusting RoIPool to be more precisely aligned using a method known as RoIAlign.

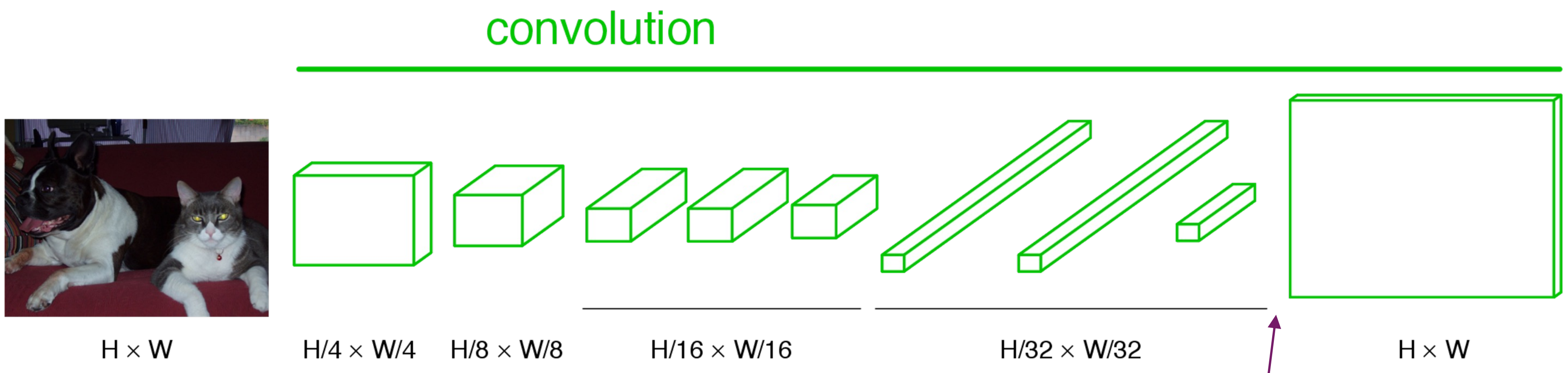
Becoming fully convolutional

- Transforming fully connected layers into convolution layers enables classification net to output a heatmap



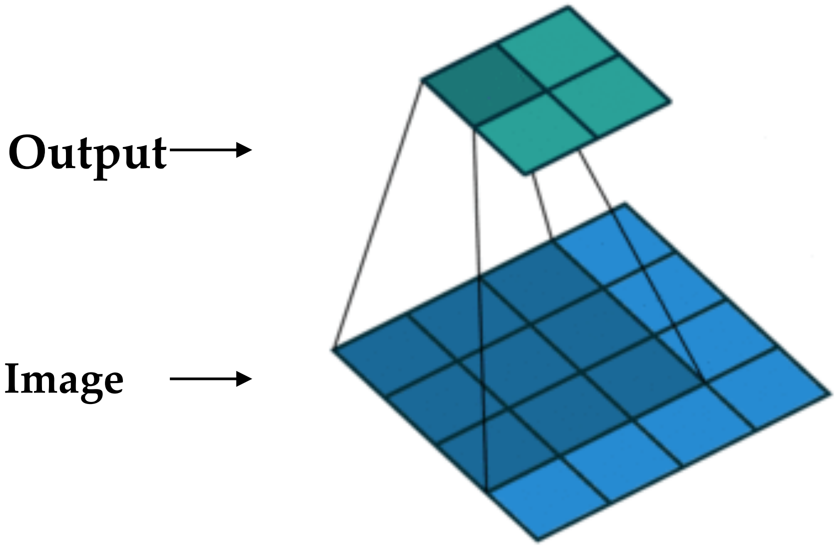
[LongCVPR2014]

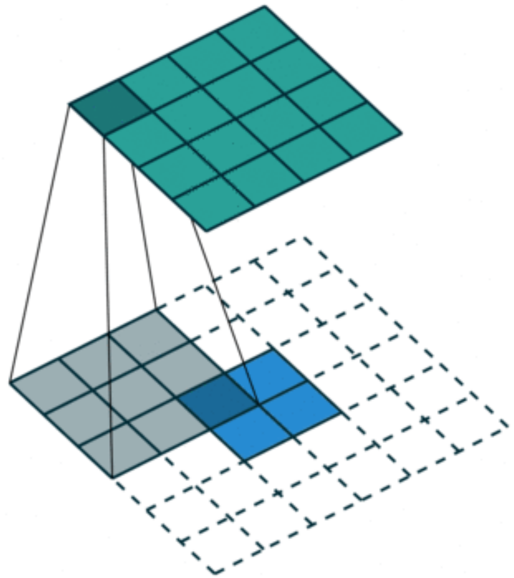
Upsampling the output

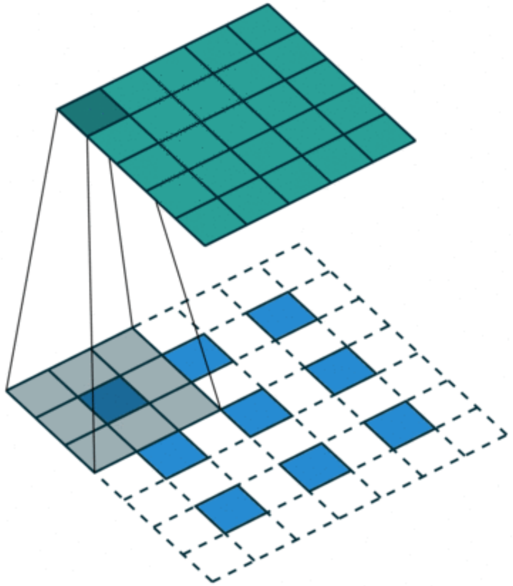


Some upsampling algorithm to return us to H x W

"Deconvolution"







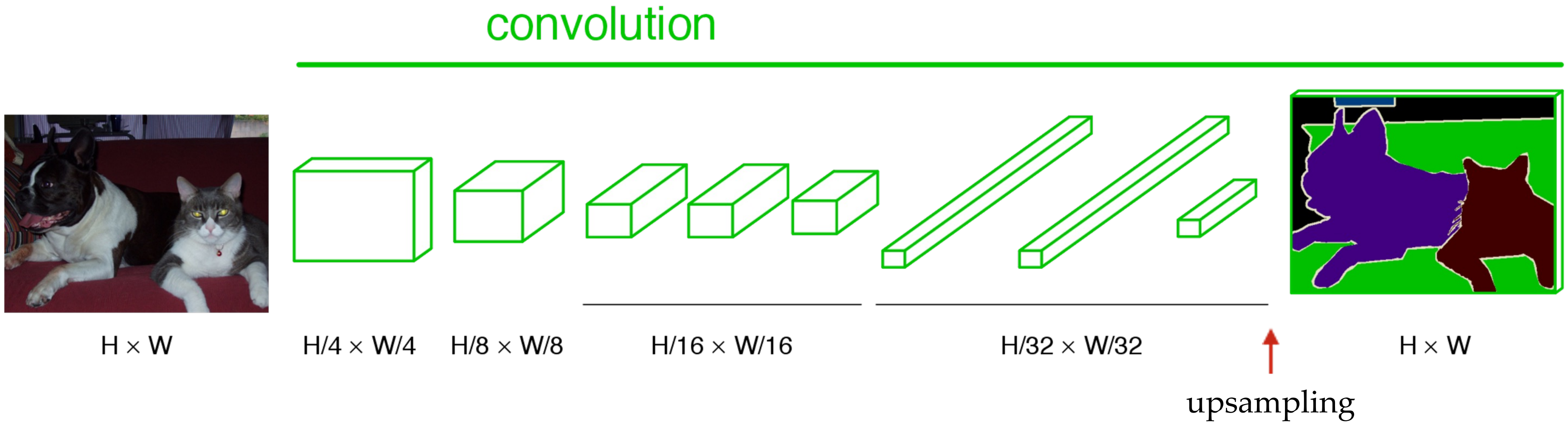
<u>Convolution</u>
No padding, no strides

<u>Upconvolution</u>
Padding, no strides

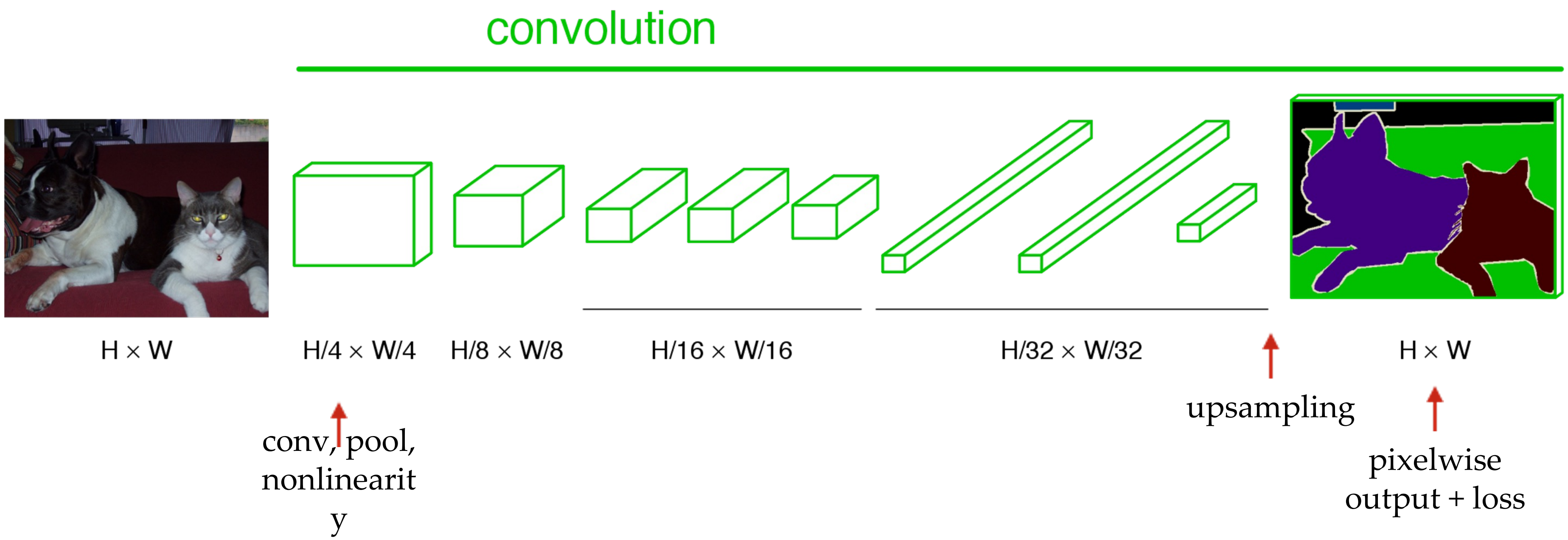
<u>Upconvolution</u>
Padding, strides

More visualizations: https://github.com/vdumoulin/conv_arithmetic

End-to-end, pixels-to-pixels network



End-to-end, pixels-to-pixels network



References

- Today & Last time:
  - Deep Learning. Chapter 9
  - UDL book. Chapter 10, 11

Announcement regarding first assignment. See Canvas.