Deep Neural Networks (Głębokie Sieci Neuronowe)

Module 1: Convolutional Neural Networks

Krzysztof Krawiec

Wydział Informatyki i Telekomunikacji
Politechnika Poznańska
2019/2020
http://www.cs.put.poznan.pl/kkrawiec/

Outline

1. Introduction
2. Milestone CNN architectures
   - AlexNet
   - VGG models
3. Advanced CNN architectures
   - Inception Network
   - Residual networks
   - Related architectures
4. Fully convolutional architectures
   - U-Net
   - Recurrent Neural Nets for Segmentation
   - Object instance segmentation
   - Fast R-CNN
   - Faster R-CNN
5. Deep learning for scene interpretation

Introduction

Fundamentals of Convolutional NNs

Fundamentals of Convolutional NNs have been covered in the “Pattern Recognition” course (first semester of the Master’s course). Please consult your lecture notes, and the CS231n Stanford course, which that part of my course is based on: http://cs231n.stanford.edu/

More specifically, the slides are available here:
http://cs231n.stanford.edu/slides/2017/

(permission granted by dr Justin Johnson)

Some thoughts on CNNs

Why are CNNs so powerful?

- Multiple filters working in parallel.
- Nonlinear mapping.
- Hierarchical organization.
- Helpful: initialization, training algorithms, batch normalization,

Some observations about CNNs

- Weight sharing:
  - Features are learned globally
- The dimensions of (purely) convolutional layers are not fixed.
  - Convolution is a strictly local operation, it cares only about the part of the input that falls within the dimensions of its receptive field (mask)
  - See, e.g., the documentation of Keras.
- In contemporary CNNs, the large number of parameters originate mostly in:
  - Fully-connected layers
  - Large numbers of filters/channels.
- CNNs can be used for harvesting global image features.

Some observations about CNNs

- Using pretrained networks is almost always worth considering.
  - Visual features are universal.
- Super efficient implementation, particularly on GPUs and hardware architectures that target tensor-based computing.
- Can be easily ‘uplifted’ to higher dimensions:
  - 3D
  - 2D+time
  - Many software packages provide and
- The concept of convolution can be generalized to other spaces.
  - E.g. graph convolution.
- One may extend input with image metadata to provide the learner with more information:
  - E.g., image dimensions.
  - Some nonlinear mapping necessary.

Milestone CNN architectures

AlexNet

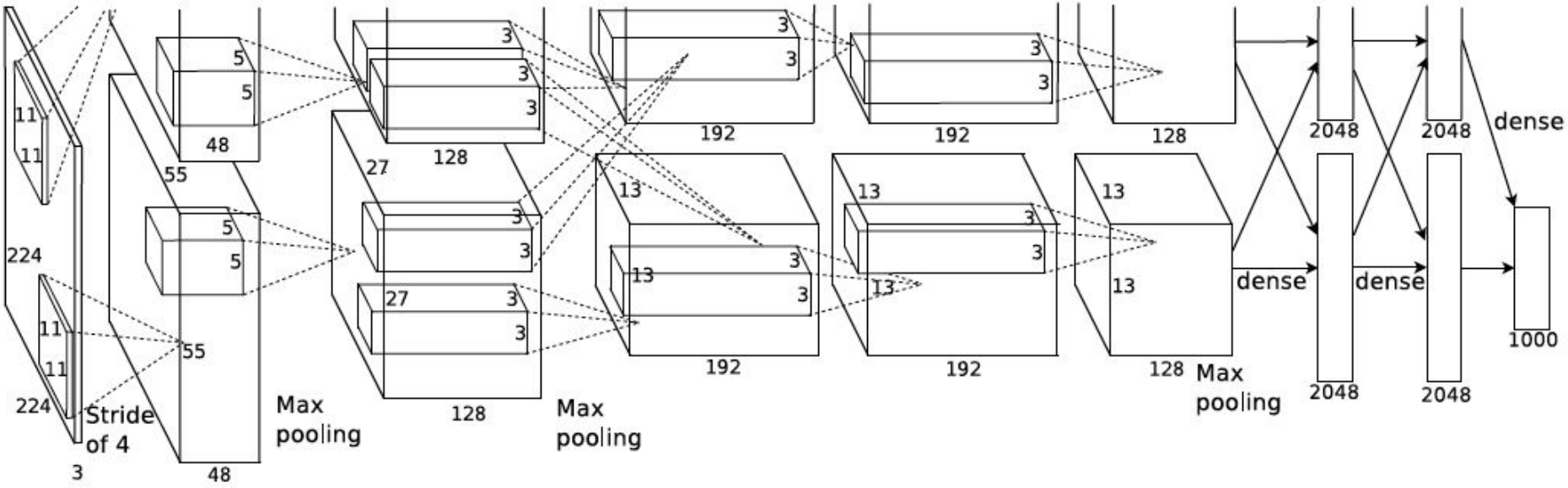
ImageNet Classification with Deep Convolutional Neural Networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton
http://arxiv.org/pdf/1409.0575

Contributions

- Large and deep (by 2012 standards)
- Trained on 1.2M images
- Bet SoTA methods of the time.
- Second place in the ILSVRC-2012 competition

Architecture:

- Five conv layers
- Some max pooling layers
- Uses dropout (recently introduce at that time)
- One of the first networks to use ReLUs
- ~60M parameters
- 1000 classes (ImageNet LSVRC contest)
- Deployed and trained on 2 GPUs (GTX 580, 3GB)


11
11
224
224
11
11
3
Stride
of 4
55
5
5
48
48
55
5
5
Max
pooling
27
3
3
3
128
27
3
3
128
Max
pooling
13
3
3
3
192
13
3
3
3
192
3
3
3
192
13
3
3
3
192
13
3
3
128
Max
pooling
dense
2048
2048
dense
dense
2048
2048
1000

Results

Effectiveness of ReLUs ——————>

Summary of results:

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

1 CNN - single CNN

5 CNN - averaging the predictions of 5 CNNs

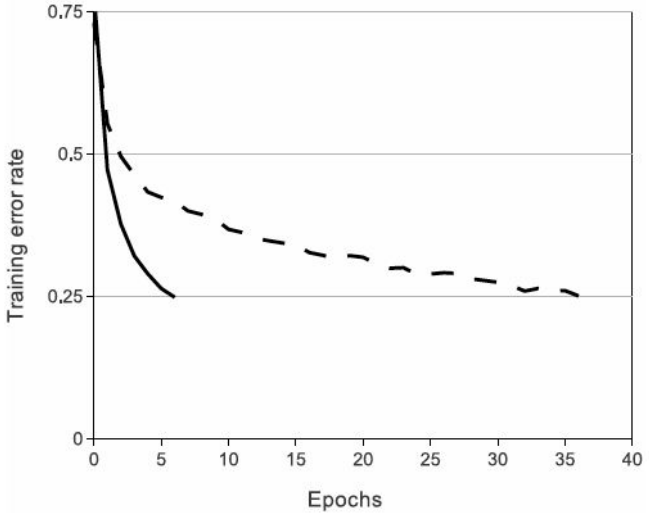
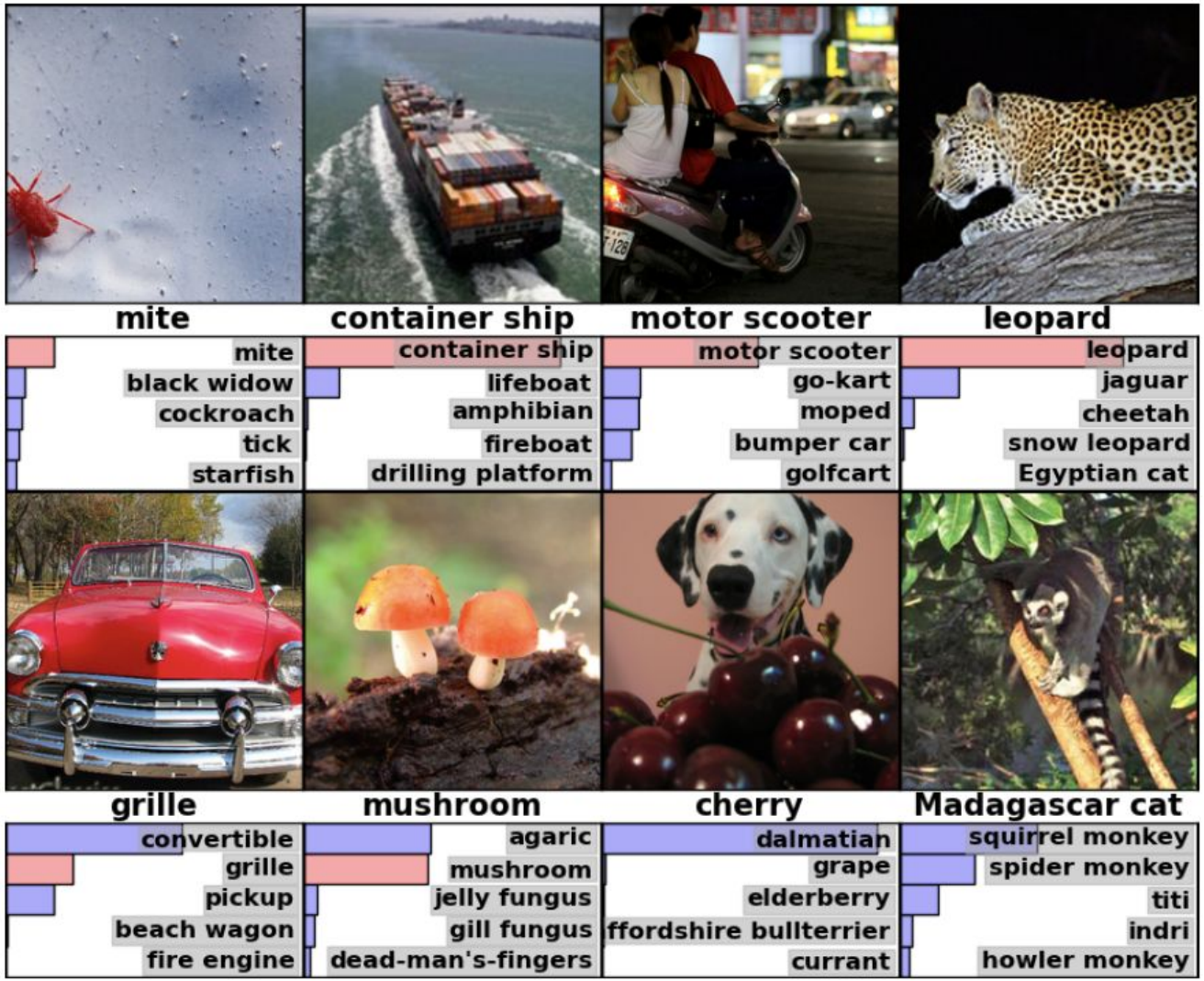


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Top-5 characteristics



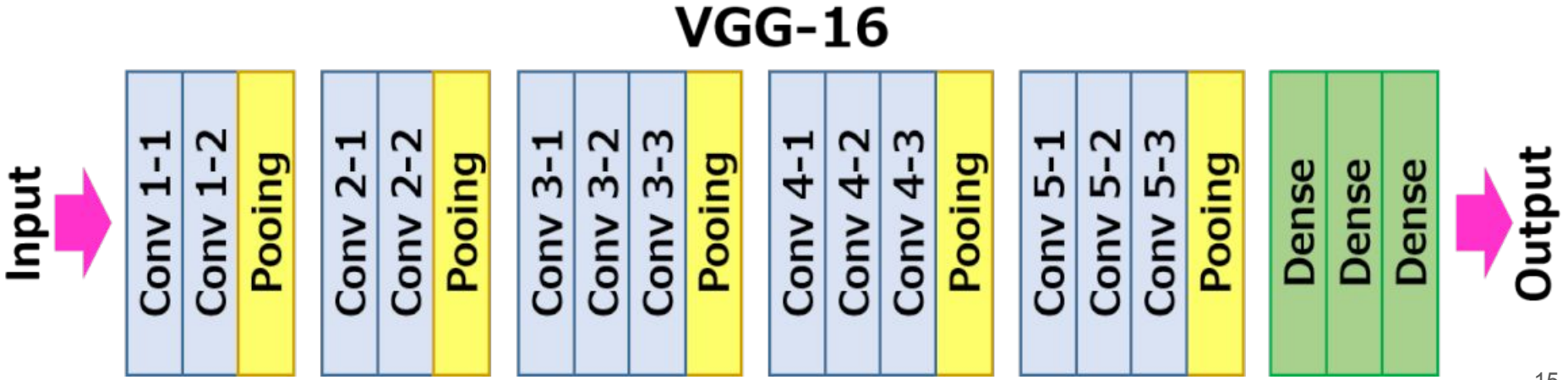
VGG models

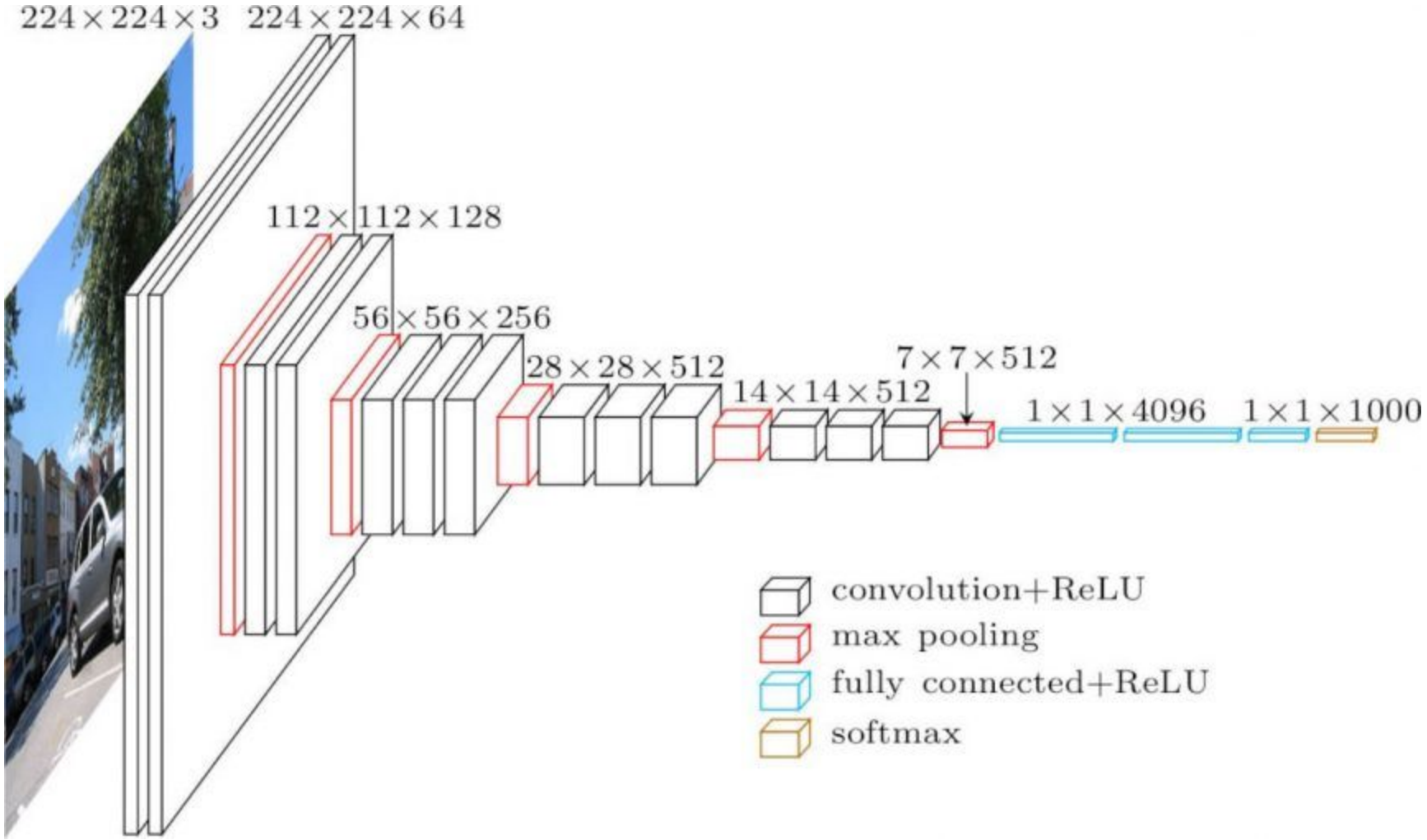
Very Deep Convolutional Networks for Large-Scale Image Recognition
Karen Simonyan, Andrew Zisserman
https://arxiv.org/abs/1409.1556

Original contribution

Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers.

The effect: lower number of parameters, more efficient training, less overfitting.




224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Advanced CNN architectures

Inception Network

Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

https://arxiv.org/abs/1409.4842

Main features

- Carefully crafted design,
- Increased the depth and width of the network while keeping the computational budget constant
- Architectural decisions based on the Hebbian principle and the intuition of multi-scale processing.
- Specific variant/instance: GoogLeNet, a 22-layer Inception network

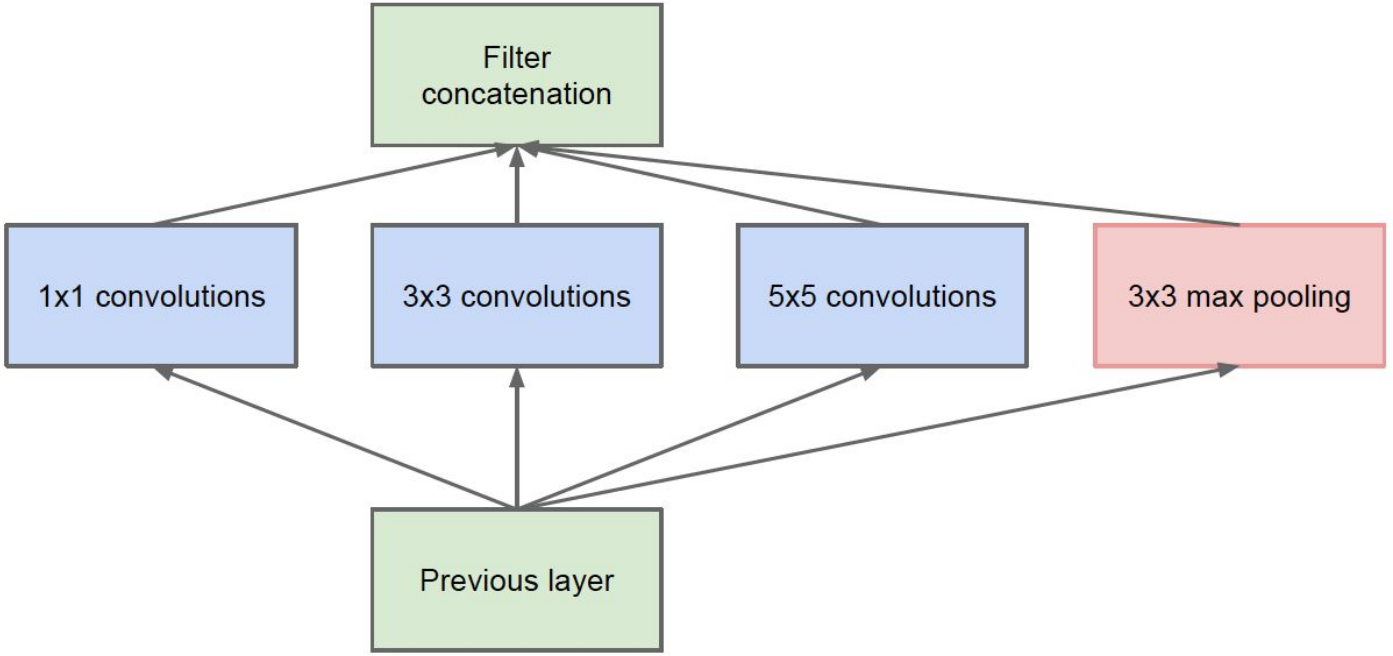
Motivations

- Arora et al. [2]:
  If the probability distribution of the dataset is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer after layer by analyzing the correlation statistics of the preceding layer activations and clustering neurons with highly correlated outputs.
  - Related to Hebbian principle – *Neurons that fire together, wire together* [Donald Hebb, 1949]
- If two convolutional layers are chained, an increase in the number of their filters results in a quadratic increase of computation.
  - If the added capacity is used inefficiently (for example, if most weights end up to be close to zero), then much of the computation is wasted.
  - This could be addressed with sparse data structures, but today's computing architectures are inefficient for such structures.

Motivations

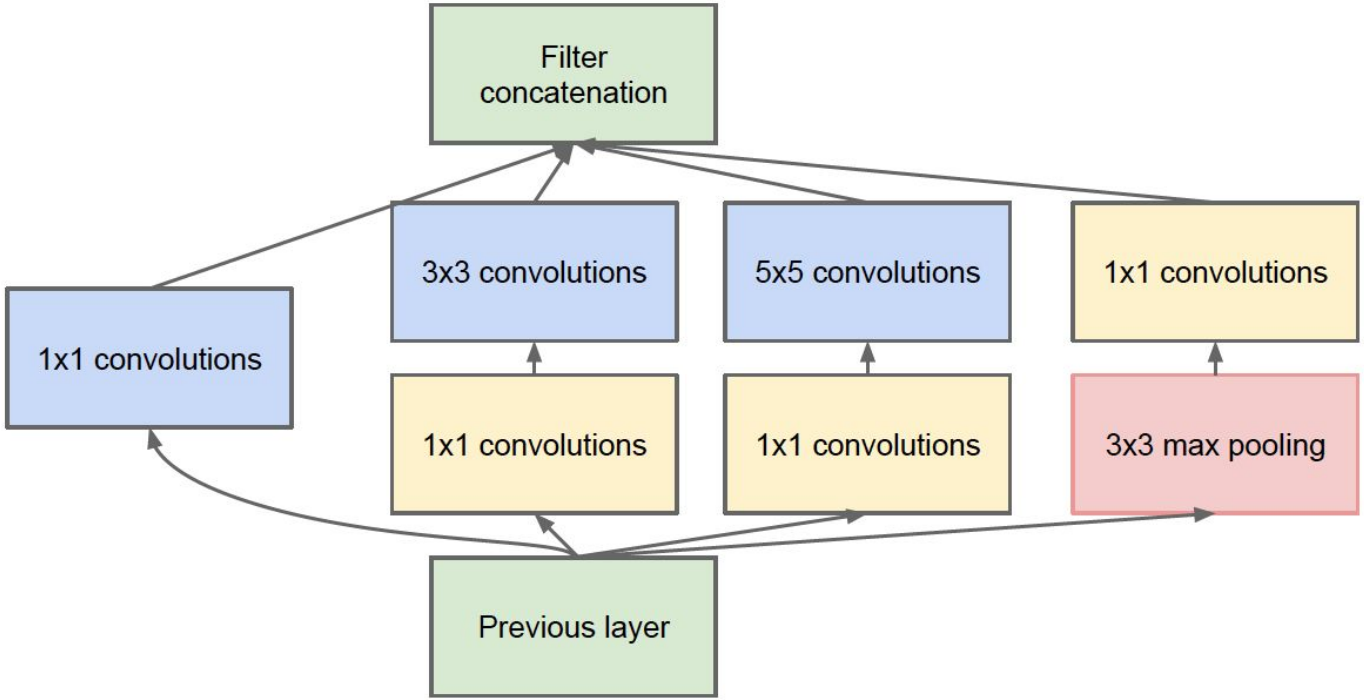
- How an optimal local sparse structure of a convolutional vision network can be approximated and covered by readily available dense components?
  - Assuming translation invariance => use convolutional building blocks.
  - All we need is to find the optimal local construction and to repeat it spatially.
- In the lower layers (the ones close to the input) correlated units would concentrate in local regions.
  - Clusters concentrated in a single region can be covered by a layer of 1x1 convolutions.
- Smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches
  - Hence also 3x3 and 5x5 convolutions.
- All those layers concatenated into a ‘filter bank’
- Additional pooling operations performed in parallel to the above.

Inception module: naive version



- The ratio of 3x3 and 5x5 convs to 1x1 should increase with consecutive layers, as features become sparser and more spatially distributed.
- The challenge: large depth of concatenated filters.

Inception module



- 1x1 convolutions embed the stimuli in a lower-dimensional space.
- Added before convolutions, to preserve spatial sparseness.
- Implement also ReLus

Complete Inception architecture

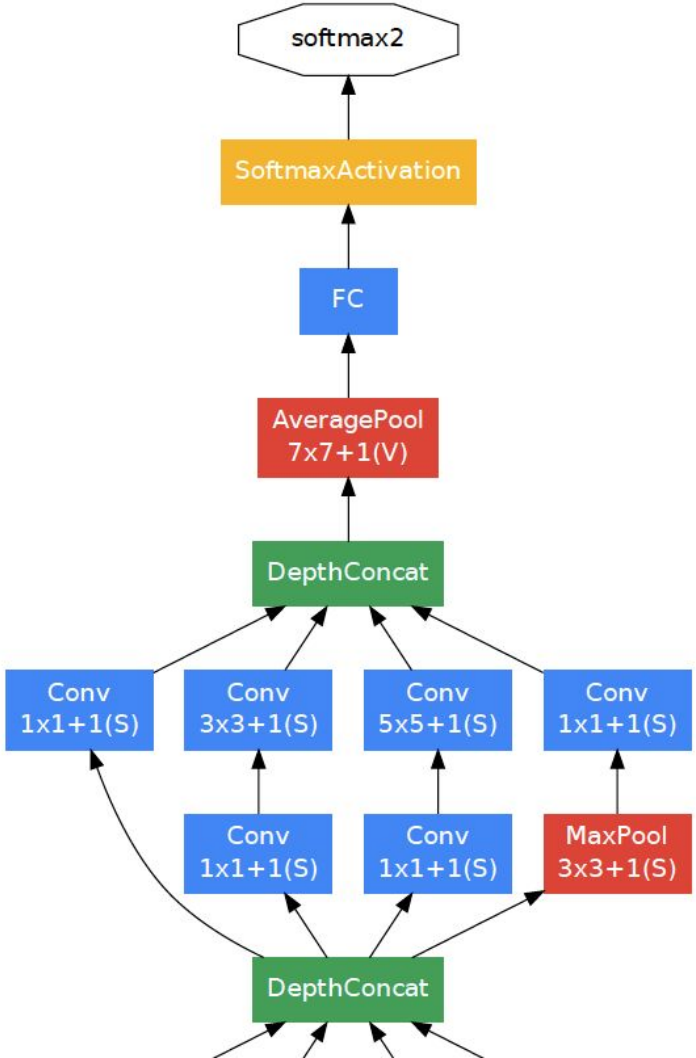
- A network consisting of modules of the above type stacked upon each other,
- Occasional max-pooling layers with stride 2 to halve the resolution of the grid.
- For memory efficiency: start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion

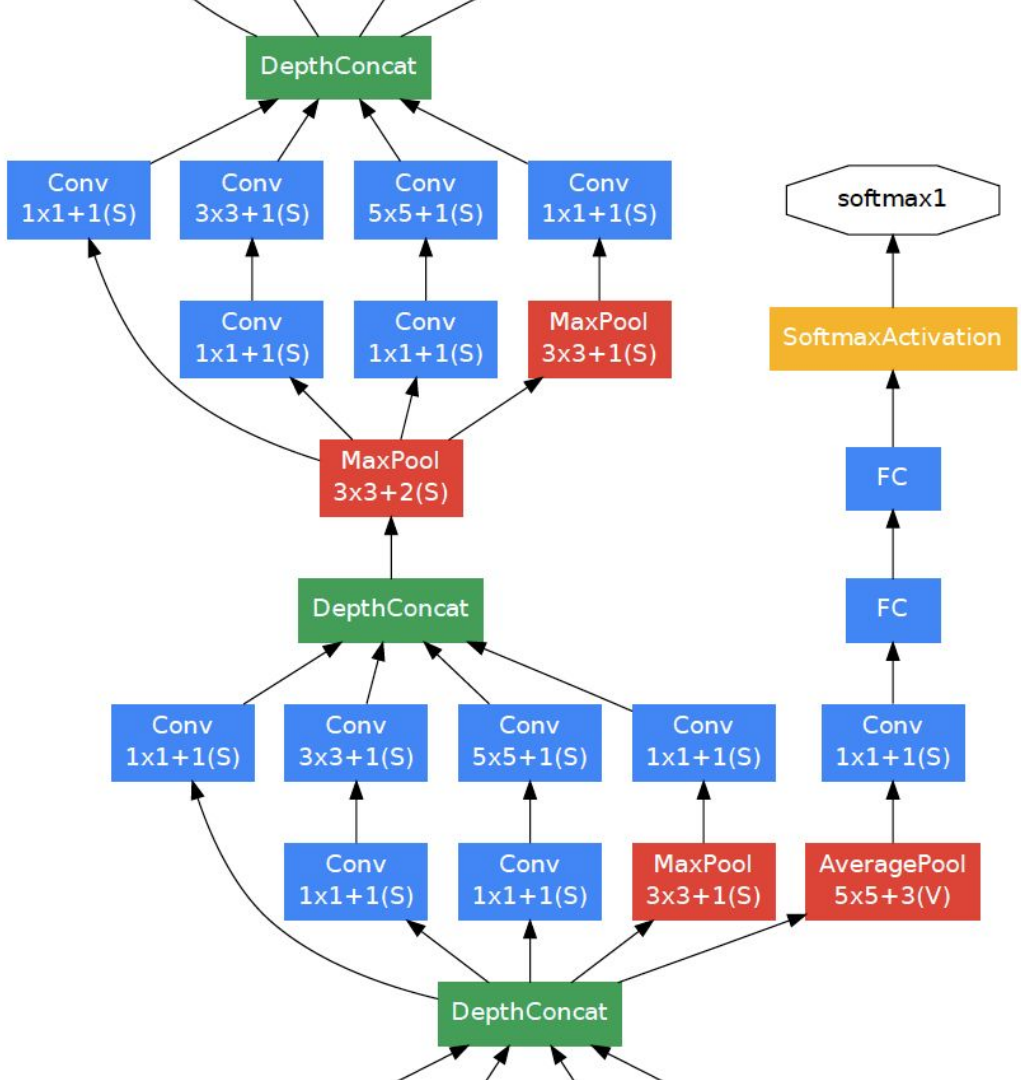
GoogLeNet

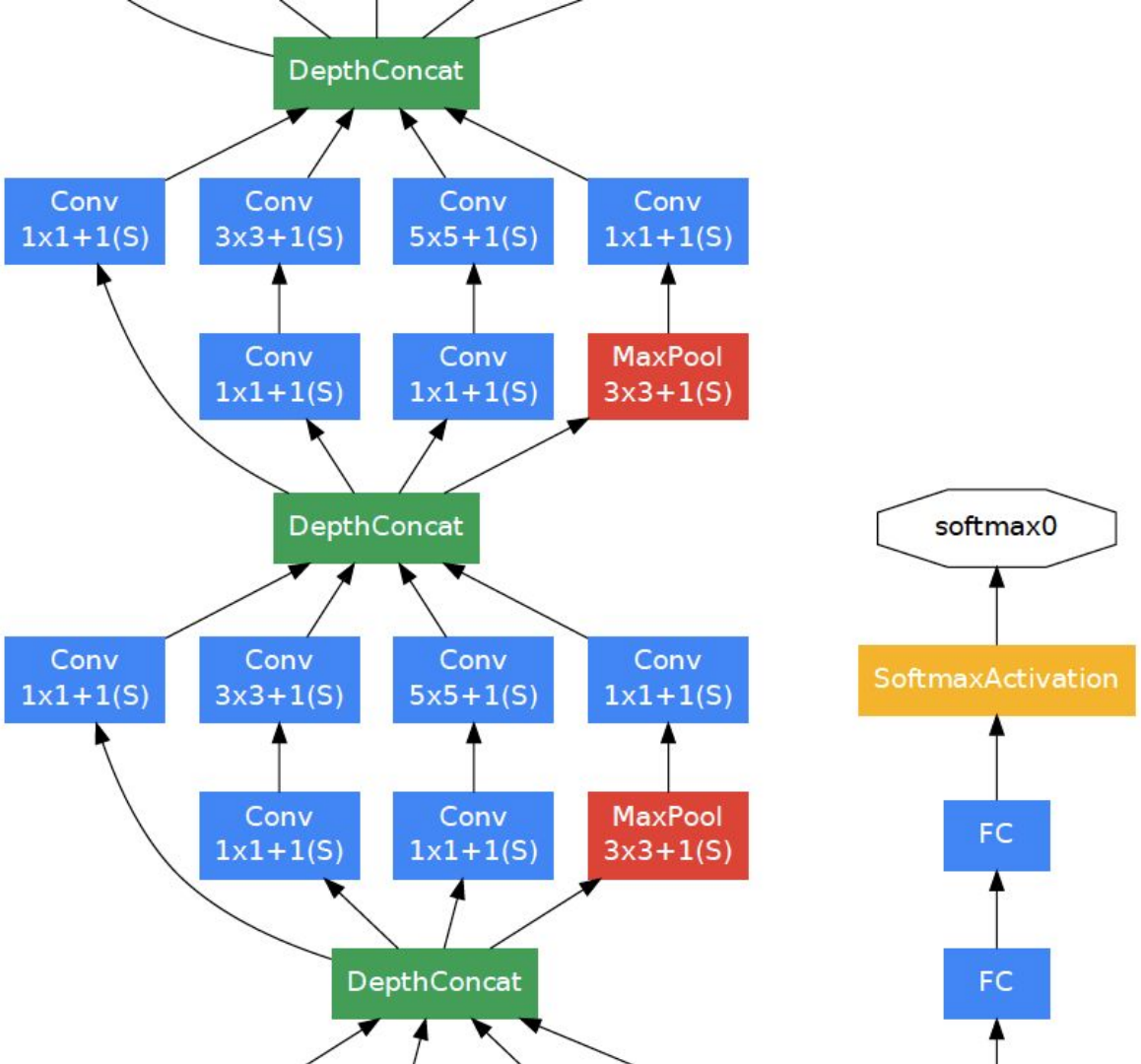
- 22 layers deep when counting only layers with parameters
  - 27 layers if also counting pooling
- The overall number of layers (feature maps): about 100.
- Uses average pooling before the classifier (with additional linear layer)

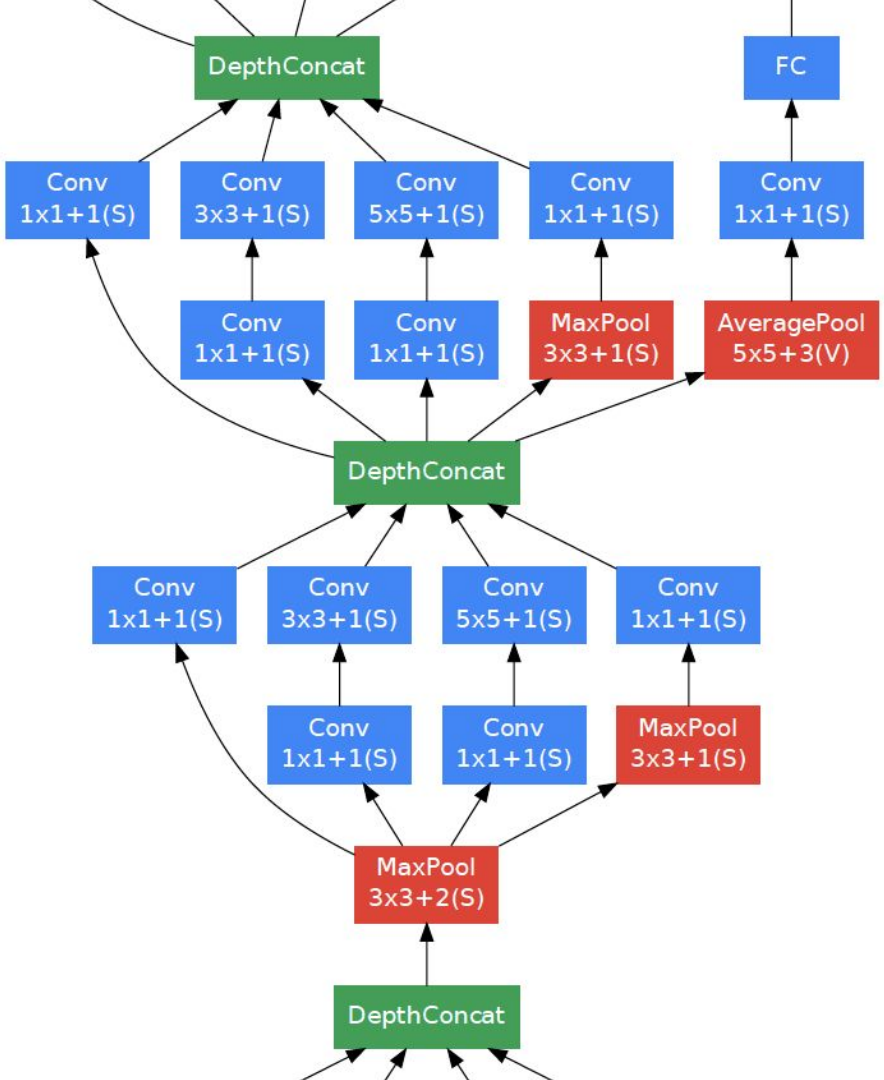
Addressing the vanishing gradient problem:

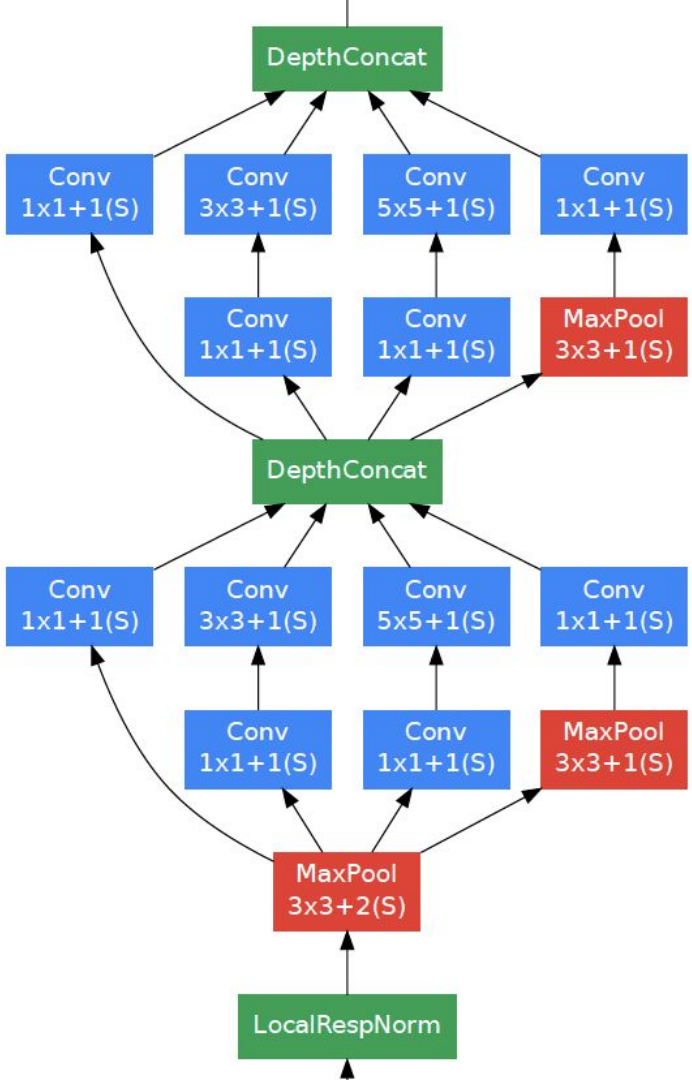
- *The strong performance of shallower networks on this task suggests that the features produced by the layers in the middle of the network should be very discriminative*.
- By adding auxiliary classifiers connected to these intermediate layers, discrimination in the lower stages in the classifier was expected

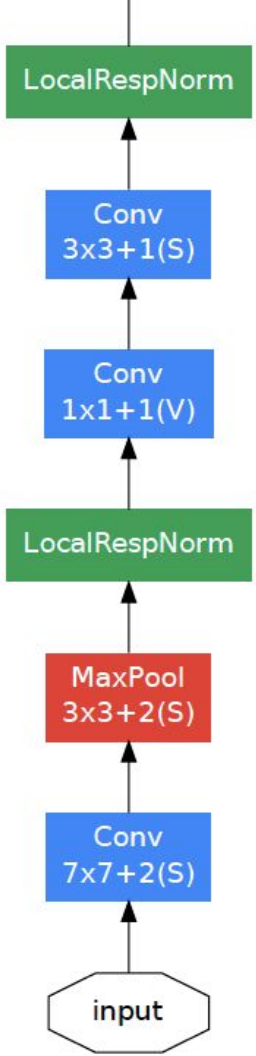

softmax2
SoftmaxActivation
FC
AveragePool
7x7+1(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
DepthConcat


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
softmax1
SoftmaxActivation
FC
FC


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
SoftmaxActivation
FC
FC


DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm


LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

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								

GoogLeNet

- Trained using asynchronous GD with 0.9 momentum.
- Quite sophisticated image cropping approach for harvesting the training data.
- Softmax probabilities are averaged
  - over multiple crops
  - and over all the individual classifiers
  - to obtain the final prediction.
- Uses an ensemble of 7 independently trained GoogLeNets:
  - Trained with the same initialization, even with the same initial weights, due to an oversight (!)
  - Same dynamic learning rate policies.
  - They differed only in sampling methodologies and the randomized input image order.
  - The ensemble included one wider version.

Residual networks

Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

https://arxiv.org/abs/1512.03385

Residual networks

- Explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions.
- The authors evaluate residual nets with a depth of up to 152 layers—8 deeper than VGG nets, but still having lower complexity.
- 1st place on the ILSVRC 2015 classification task
- 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

Motivations

- Let H(x) be the mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers.
- If a multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., H(x) - x
  - (assuming that the input and output are of the same dimensions).
- Let's **explicitly let these layers approximate a residual function** F(x) := H(x)-x.
- In other words: the original function becomes F(x)+x.
- Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different (due to 'shortcuts' in gradient flow).

Basic building block

Mapping realized by a Resnet building block:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

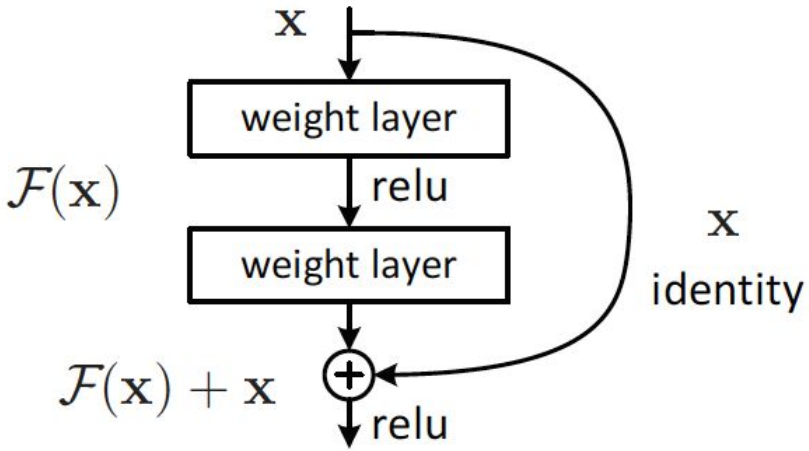
Note: requires the shape of y to be the same as x. What if, for some reason, that's impossible?

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

Nevertheless, the authors work mostly with the former case.

Basic building block

The specific building block used by the authors:



Resnets are thus *modular networks* (though there's no weight sharing between individual building blocks/modules.

Overall architecture

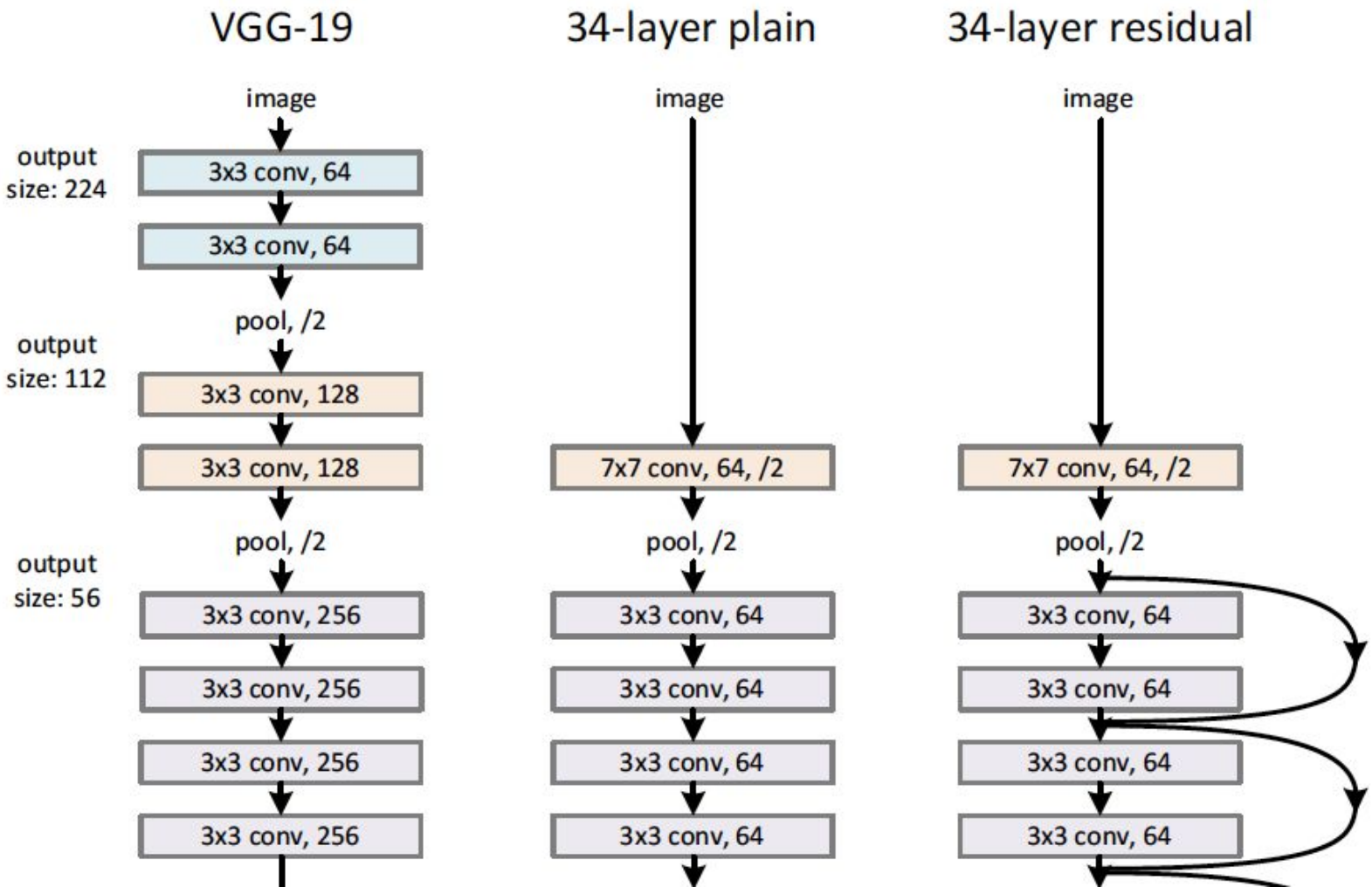
Design follows VGG networks:

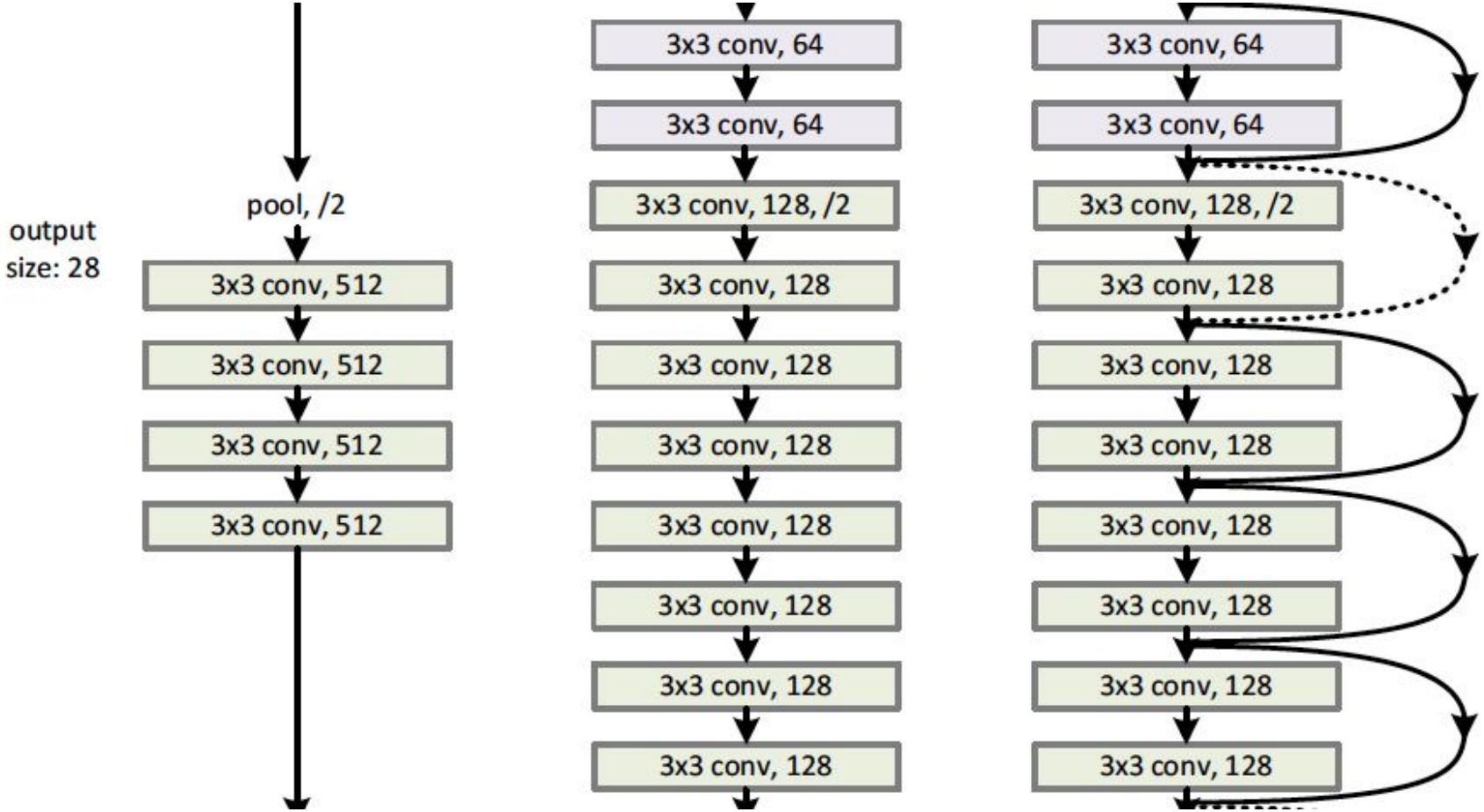
- Mostly 3x3 filters
- For the same output feature map size, the layers have the same number of filters (what did I mean by that?)
- **If the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer.**
- Downsampling using convolution with stride 2 (“/2”).

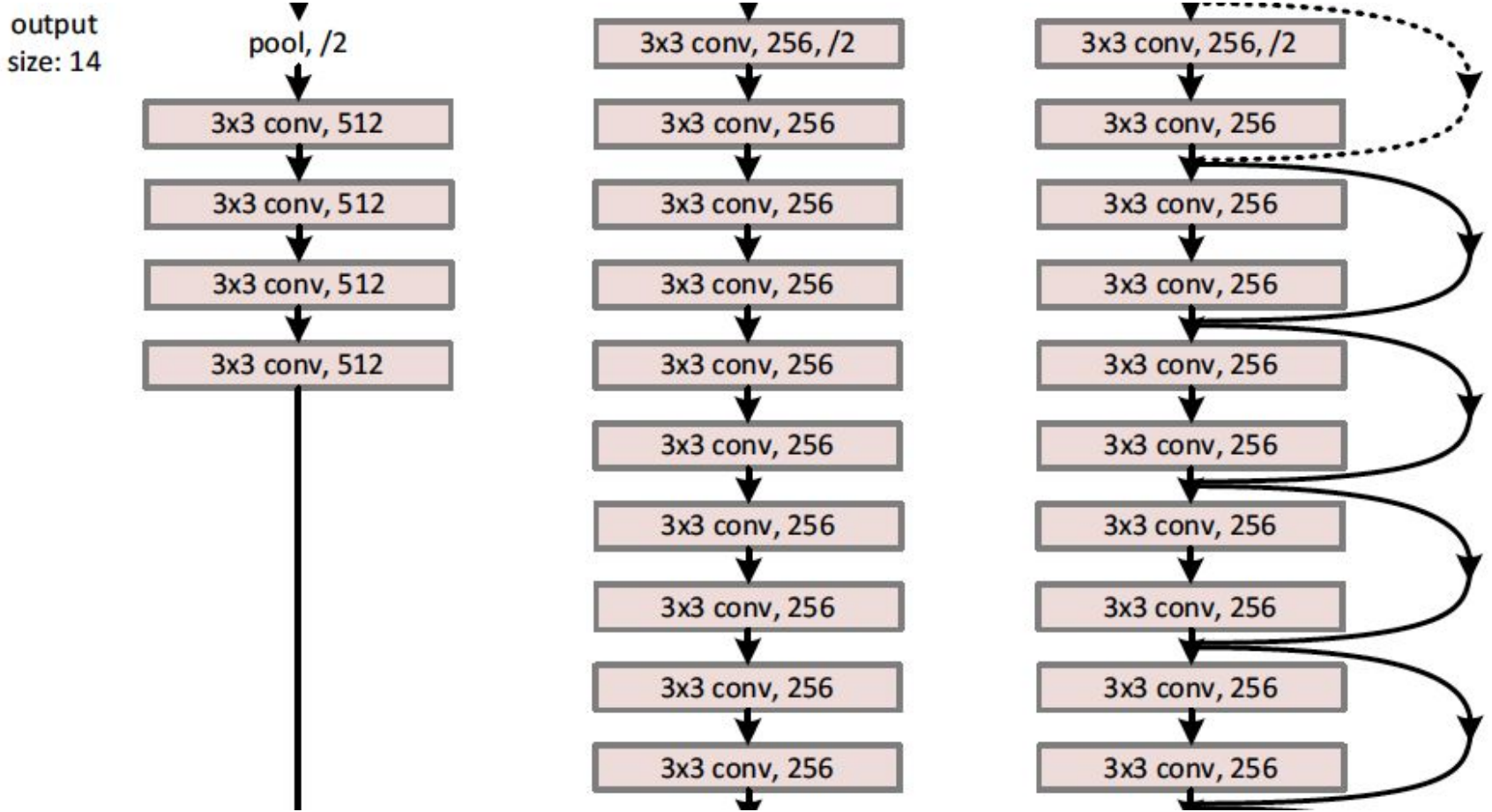
The figure that follows:

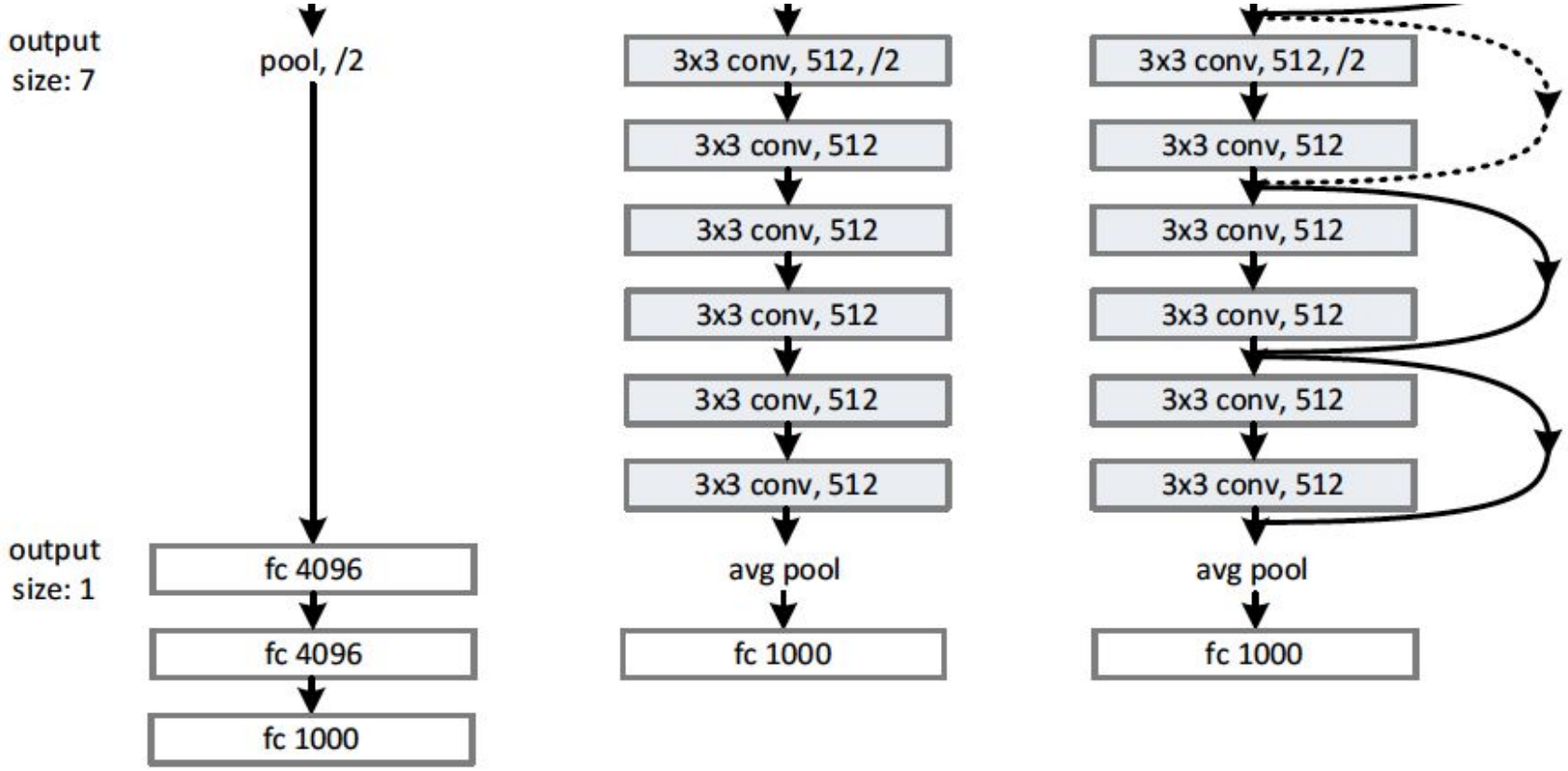
- Left: VGG-19 model: 19.6 billion FLOPs
- Center: A plain network with 34 parameter layers: 3.6 billion FLOPs
- Right: A residual network with 34 parameter layers: 3.6 billion FLOPs

The dotted shortcuts increase dimensions.


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64


output
size: 28
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128


output
size: 14
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256


output
size: 7
pool, /2
output
size: 1
fc 4096
fc 4096
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

Other elements

Batch normalization after each convolution (before activation)

Two options for the dotted connections:

- A: The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions (no extra parameters)
- B: The projection shortcut in is used to match dimensions, using done by 1x1 convolutions:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

Considered configurations

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Some results

(A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter free
(B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity;
(C) all shortcuts are projections

Trained with ordinary SGD with momentum.

Experimented also with ensambles;
networks up to 1000 layers (found it to be worse than 110-layer net)

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

Why do ResNets work?

- They address the vanishing/exploding gradient problem.
  - Recall what happens with gradient in computation graphs/flows.
- This implicitly allows maintaining relatively low depth (number of channels/dimensions) along the network
  - Notice the not too frequent dotted connections.
- Does modularity help?
  - An open question.
  - Eases automation of architecture optimization (see, e.g., neuroevolution).

Related architectures

Wide Residual Networks (Wide-Nets)

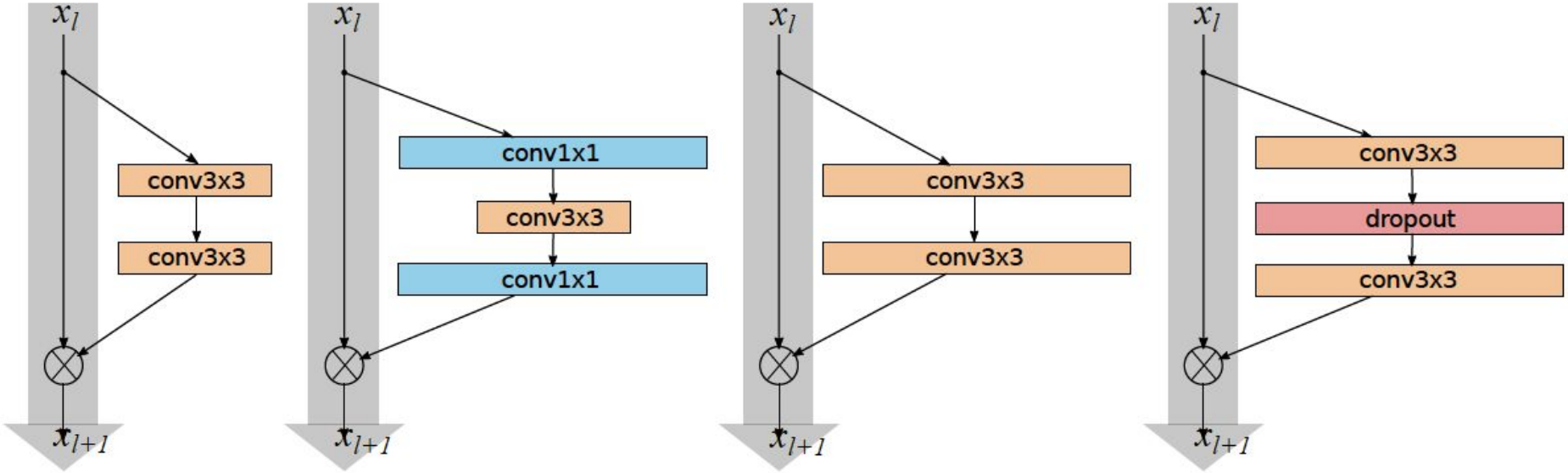
Wide Residual Networks, Sergey Zagoruyko, Nikos Komodakis,
http://arxiv.org/abs/1605.07146

Motivations

- The study of residual networks has focused mainly on the order of activations inside a ResNet block and the depth of residual networks.
- Goal: To explore a much richer set of network architectures of ResNet blocks and thoroughly examine how several other different aspects besides the order of activations affect performance
- Width vs depth in residual networks.
- shallow circuits can require exponentially more components than deeper circuits

Motivations

- **Diminishing feature reuse** [28]: As gradient flows through the network there is nothing to force it to go through residual block weights and it can avoid learning anything during training, so it is possible that there is either only a few blocks that learn useful representations, or many blocks share very little information with small contribution to the final goal.
  - Some earlier works tried to address this by randomly disabling residual blocks during training.
- Hypothesis: show that the widening of ResNet blocks (if done properly) provides a much more effective way of improving performance of residual networks compared to increasing their depth.
- In particular, we present wider deep residual networks that significantly improve over [13], having 50 times less layers and being more than 2 times faster.
-



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

The architecture

- k multiplies the number of features in convolutional layers
- k=1 => ResNet
- N: the number of blocks in a group

group name	output size	block type = $B(3,3)$
conv1	32×32	$[3\times3,\ 16]$
conv2	32×32	$\begin{bmatrix} 3\times3,\ 16\times k \\ 3\times3,\ 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3,\ 32\times k \\ 3\times3,\ 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3,\ 64\times k \\ 3\times3,\ 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Some results (no dropout)

	depth-k	# params	CIFAR-10	CIFAR-100
NIN [20]			8.81	35.67
DSN [19]			8.22	34.57
FitNet [24]			8.39	35.04
Highway [28]			7.72	32.39
ELU [5]			6.55	24.28
original-ResNet[11]	110	1.7M	6.43	25.16
	1202	10.2M	7.93	27.82
stoc-depth[14]	110	1.7M	5.23	24.58
	1202	10.2M	4.91	-
pre-act-ResNet[13]	110	1.7M	6.37	-
	164	1.7M	5.46	24.33
	1001	10.2M	4.92(4.64)	22.71
WRN (ours)	40-4	8.9M	4.53	21.18
	16-8	11.0M	4.27	20.43
	28-10	36.5M	**4.00**	**19.25**

Related architectures

- Highway networks: Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber, https://arxiv.org/abs/1505.00387
  - Shortcut connections with gating functions (rather than identities): *transform gate* T and *carry gate* C:
    $$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$
  - Original paper: T is a dense sigmoid layer and:
    $$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$
  - T's bias initialized to implement the carry behavior.
  - Allow unimpeded information flow across several layers on "information highways"
- Fractal networks, FractalNet [Larsson et al. 2017]: parallel multi-scale, trained by dropping out entire modules/paths
- Deep networks with stochastic depth [Huang et al. 2016]: a bit like dropout on the level of entire layers

Fully convolutional architectures

Motivations

Any usage scenario that requires image -> image mapping, i.e. image processing.
Examples:

- Image segmentation
  - Including semantic segmentation
- Image denoising
  - A CNN serves as a denoising filter.
  - Can adapt in training to the characteristics (distributions) of a given class of images.
- Image superresolution
- Style transfer
- Virtually any image processing.

Note: All these tasks offer much stronger training signal than regular classification or regression.

Image segmentation has many faces ...

- ‘Ordinary’ segmentation
  - We’re happy if the algorithms manages to delineate perceptually distinct regions.
- Semantic segmentation
  - The algorithm is expected to work with semantics-rich labels, like sidewalk, car, person, road sign, ...
- Object instance segmentation
  - Multiple instances of the same type of entity should be delineated.
- ...

DNN-based approaches to segmentation

Main categories of representatives:

- Patch-based - a ‘naive’ approach
  - Replaces the segmentation task with classification of individual pixels based of image patches
- Fully convolutional (FCNN), ‘holistic’
  - Performs segmentation of all image patches in parallel
- Recurrent
  - Uses recurrent layers/cell/networks to ‘sweep’ the input image.

Patch-based segmentation

Segmenting Retinal Blood Vessels with Deep Neural Networks

Paweł Liskowski, Krzysztof Krawiec

https://ieeexplore.ieee.org/document/7440871?arnumber=7440871

A naive approach to segmentation using DL

- A network in a sliding-window setup predicts class label of each pixel by analyzing a local region (patch) around that pixel (e.g., Ciresan et al. [1])
- Advantage: each training image gives rise to thousands or more of training examples (patches)

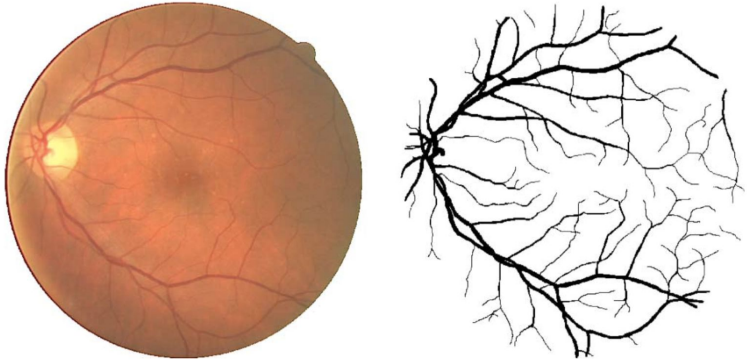


Fig. 1. A training image from the DRIVE database (left) and the corresponding manual segmentation (right).

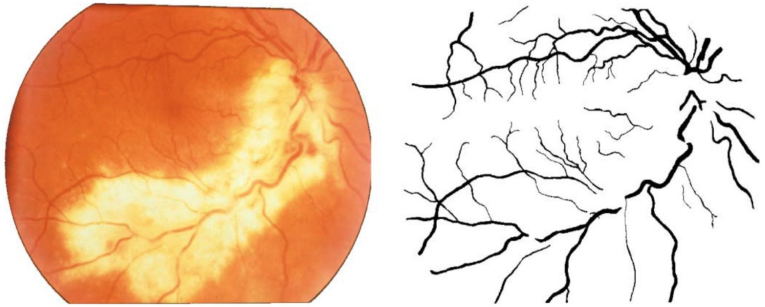


Fig. 2. A pathological image from the STARE database (left) and the corresponding manual segmentation (right).

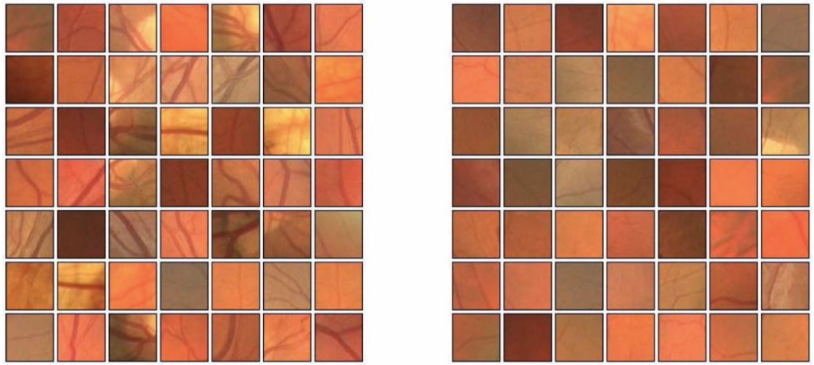


Fig. 4. Examples of positive (left) and negative (right) 27×27 training patches extracted from the DRIVE images.

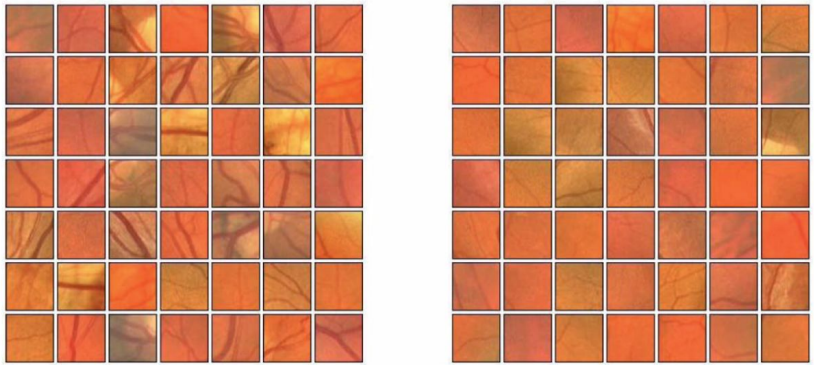


Fig. 5. Examples of positive (left) and negative (right) training patches after applying GCN transformation.

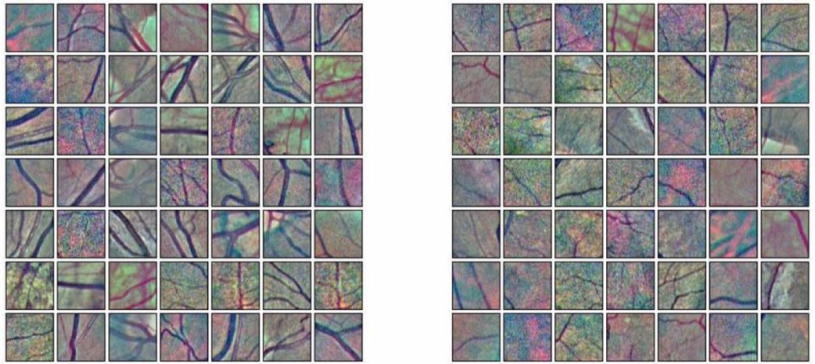


Fig. 6. Positive (left) and negative (right) training patches after applying ZCA whitening transformation.

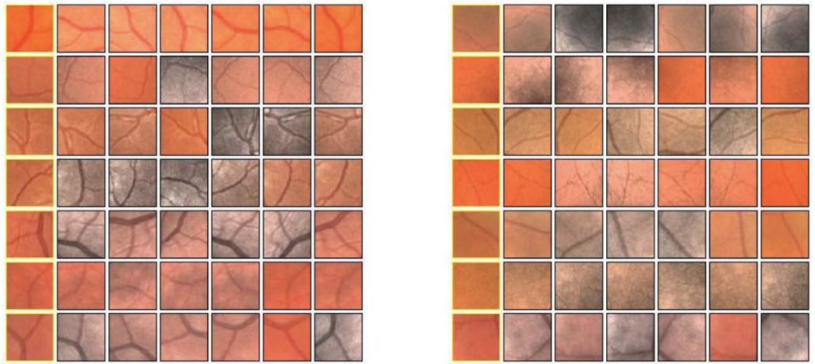


Fig. 7. Augmentations of positive (left) and negative (right) training patches. Each row shows 6 random augmentations of the leftmost patch.

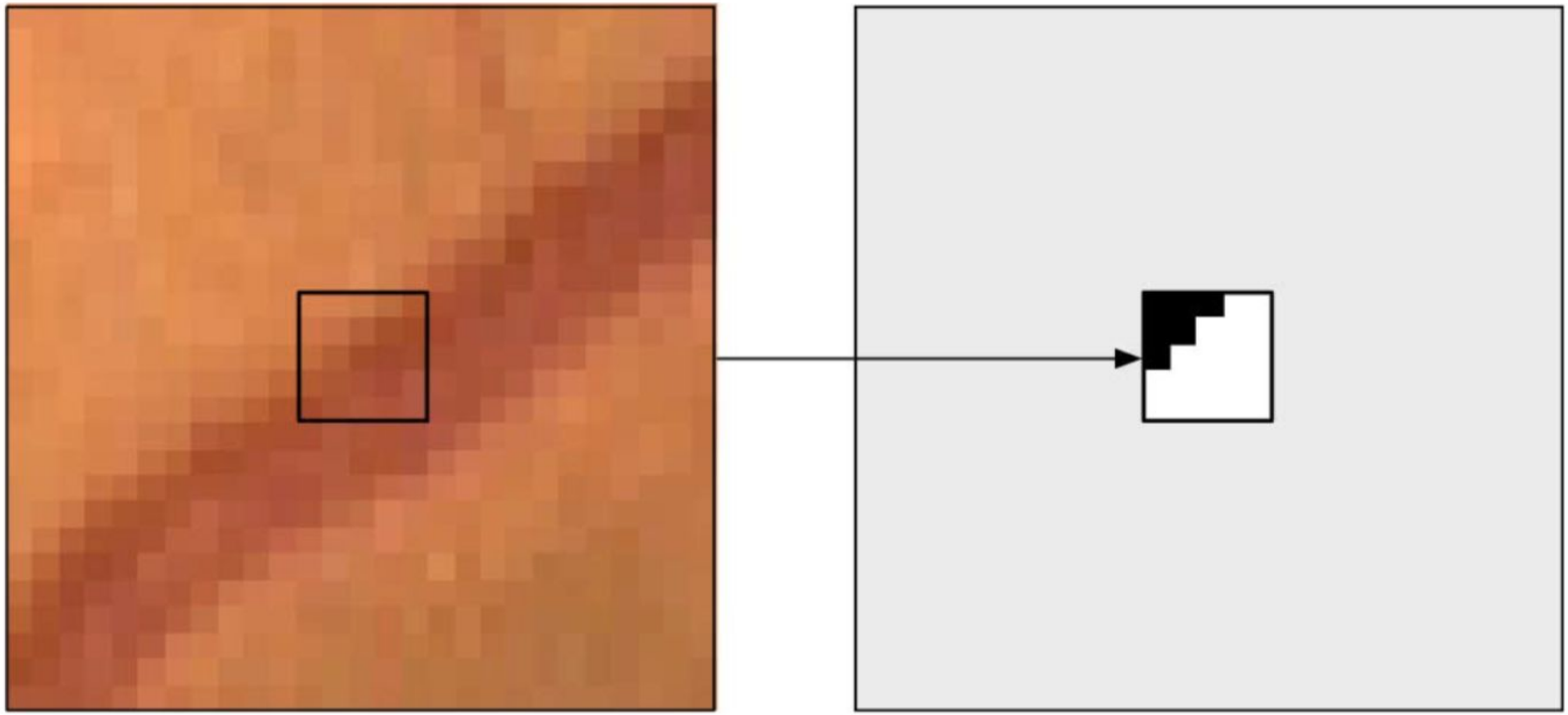


Fig. 8. A training example for the SP approach: the 27×27 patch with the $s \times s = 5 \times 5$ output window (left) and the corresponding desired output (right).

Patch-based segmentation: Summary

Disadvantages:

- Slow, in particular at querying, because of explicit scanning of the image.
- Trade-off between the localization accuracy and the use of context:
  - Larger patches require typically more max-pooling layers, which reduces localization accuracy (spatial ‘aliasing’)

U-Net

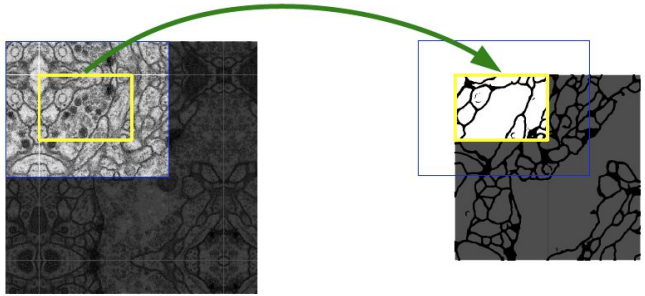
U-Net: Convolutional Networks for Biomedical Image Segmentation

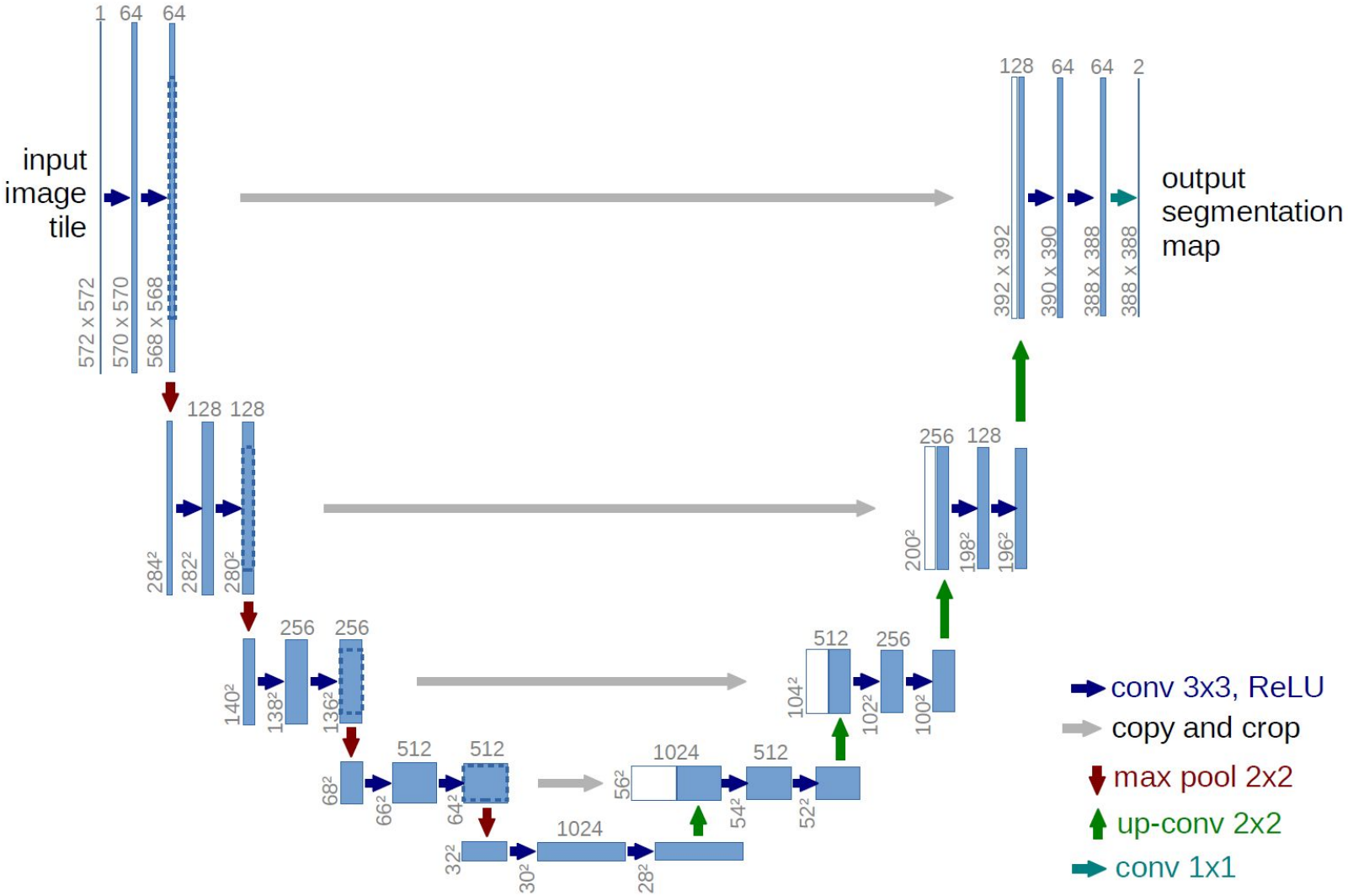
Olaf Ronneberger, Philipp Fischer, Thomas Brox

https://arxiv.org/abs/1505.04597

The architecture

- Uses FCNN to supplement a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators.
  - Increase the resolution of the output.
- Two main building blocks:
  - a contracting path to capture context
  - a symmetric expanding path that enables precise localization
- The network does not have any fully connected layers and only uses the valid part of each convolution.
  - Padding via mirroring at the edge of the image.




input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Some comments on U-net

- Advantages: Trains effectively from small samples.
- Q: Is this an autoencoder?
- The ‘copy and crop’ connections are not equivalent to residual connections.
  - However, they definitely allow part of gradient flow along a shorter path.
- Even though the numbers in the figure (previous slide) refer to specific image dimensions, this is still a FCNN:
  - All operations (convolutions, max pooling, up-conv) rely on local receptive fields.
  - Therefore, U-nets are scalable/applicable to images of arbitrary dimensions.

Other features

- Uses weighted target: separating background labels between touching objects (e.g., cells) obtain relatively large weight.
- Separation border computed using morphological operations.

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \cdot \exp\left(-\frac{(d_1(\mathbf{x}) + d_2(\mathbf{x}))^2}{2\sigma^2}\right)$$

- Where:
  - wc : balances class frequencies;
  - d1: distance to the border of the nearest object (cell in the original paper),
  - d2: distance to the border of the second nearest cell.
  - The authors used w0=10

Henrietta Lacks (1920-1951)

HeLa

- HeLa is an immortal cell line used in scientific research.
  - The first human cells grown in a lab that were naturally "immortal", meaning that they do not die after a set number of cell divisions
- It is the oldest and most commonly used human cell line (1951).
- The first human cells grown in a lab that were naturally "immortal", meaning that they do not die after a set number of cell divisions
- Scientists have grown an estimated 50 million metric tons of HeLa cells
- Almost 11,000 patents based on the HeLa line.

U-Net's weighted target on HeLa cells image

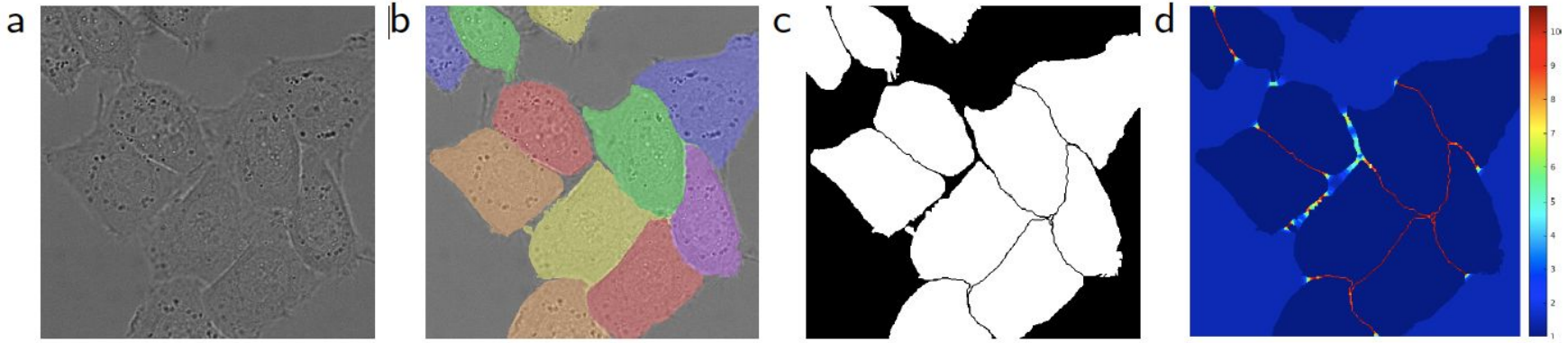


Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**d**) map with a pixel-wise loss weight to force the network to learn the border pixels.

Other features

- Loss function: cross-entropy on output (softmax) and target labels

$$p_k(\mathbf{x}) = \exp(a_k(\mathbf{x}))/\left(\sum_{k'=1}^{K}\exp(a_{k'}(\mathbf{x}))\right)$$

$$E = \sum_{\mathbf{x}\in\Omega} w(\mathbf{x})\log(p_{\ell(\mathbf{x})}(\mathbf{x}))$$

- Notice: ‘soft labels’
  - Effectively, the classification tasks turned into a regression task.
  - Disputable, as regression tasks are usually more difficult (must approach the target value as closely as possible).
- Makes extensive use of augmentations.

Some results

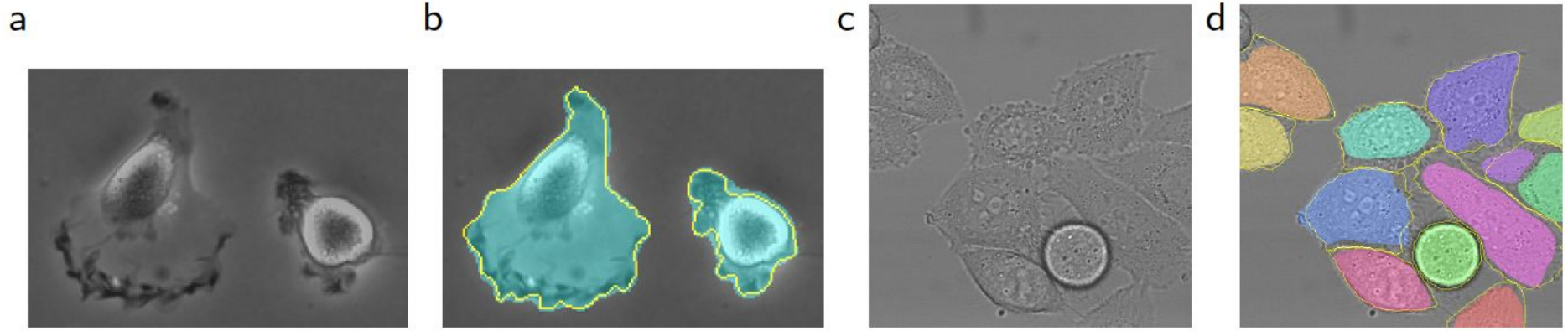


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

Some results

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

IOU = Intersection over Union

Recurrent Neural Nets for Segmentation

RNNs for segmentation

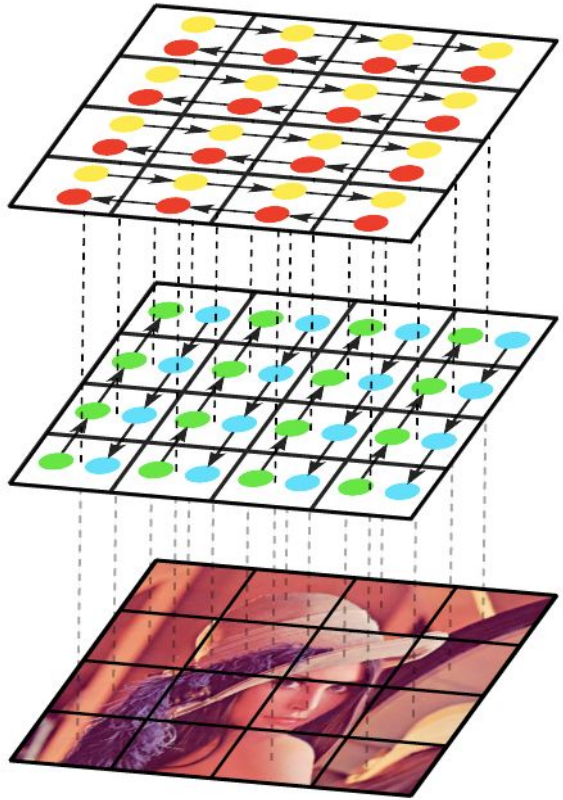
ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks
Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, Yoshua Bengio
https://arxiv.org/abs/1505.00393

Motivations

- Not explicitly stated
- “Apply a recurrent NN to an image and see what happens”?
- A nice example of versatility of contemporary DNN architectures.

Architecture

- Replace each convolutional+pooling layer in the CNN with four RNNs that sweep over lower-layer features in different directions:
  - (1) bottom to top,
  - (2) top to bottom,
  - (3) left to right and
  - (4) right to left.
- Each feature activation in its output is an activation at the specific location
- Use conventional sequential (one-dimensional RNNs)



Some details

Formally, for the vertically-scanning layer:

$$v_{i,j}^{F} = f_{\text{VFWD}}(z_{i,j-1}^{F}, p_{i,j}), \text{ for } j = 1, \cdots, J$$
$$v_{i,j}^{R} = f_{\text{VREV}}(z_{i,j+1}^{R}, p_{i,j}), \text{ for } j = J, \cdots, 1$$

- f_{VFWD} and f_{VREV} can be
  - LSTM cells (Hochreiter & Schmidhuber 1997), or
  - GRU cells (Cho et al. 2014)
- The outputs of the hidden steps are concatenated and so create a composite feature map.
- Many such bi-layers (V+H) can be stack atop each other.

Comparison with conventional CNNs

- ReNets propagate information through *‘lateral’* connections,
  - (not to be confused with *latent*)
  - The layer has access to “global information”.
- Lateral connections help remove/resolve redundant features at different locations in the image.
- ReNet does not use any pooling: *The lateral connection in ReNet can emulate the local competition among features induced by the max-pooling in LeNet.*
  - Max pooling causes irreversible loss of information about the exact location of the feature
    - (not mentioning the loss of information about the presence of features at non-maxima)
  - Avoiding max pooling makes the mapping easier to invert, and thus better applicable in autoencoder-like architectures.
- Disadvantage of ReNet: Not easily parallelizable.

Tested on

- MNIST
- CIFAR-10
- Street View House Numbers (SVHN)

Used GRU units:

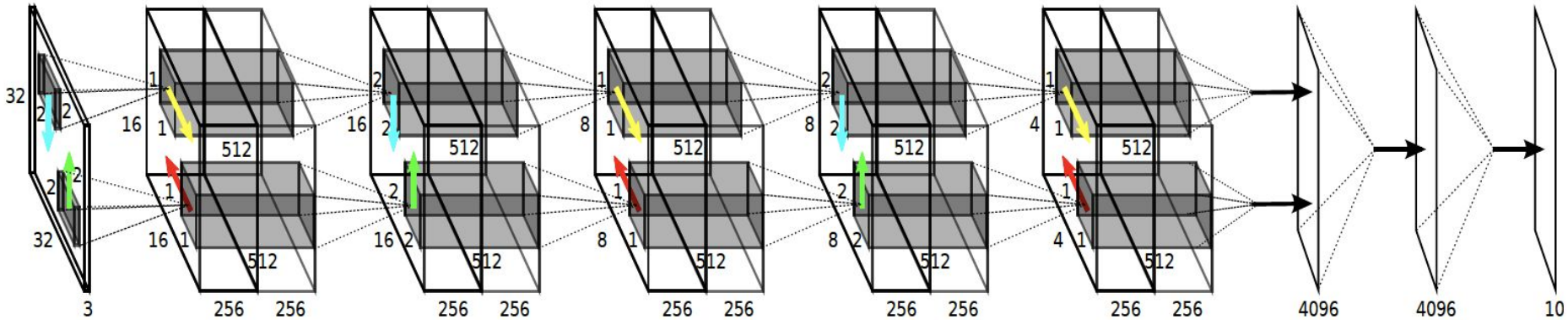
$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t,$$

$$\tilde{h}_t = \tanh\left(W x_t + U(r_t \odot h_{t-1}) + b\right)$$

$$[u_t; r_t] = \sigma\left(W_g x_t + U_g h_{t-1} + b_g\right).$$

Where: x - input, h - hidden state, b - bias, u - gating signal

Architecture used for SVHN



SoTA accuracy, but not better than conventional CNNs (next slide)

Test Error	Model
0.28%	[Wan et al., 2013]⋆
0.31%	[Graham, 2014a]⋆
0.35%	[Ciresan et al., 2010]
0.39%	[Mairal et al., 2014]⋆
0.39%	[Lee et al., 2014]⋆
0.4%	[Simard et al., 2003]⋆
0.44%	[Graham, 2014b]⋆
0.45%	[Goodfellow et al., 2013]⋆
0.45%	**ReNet**
0.47%	[Lin et al., 2014]⋆
0.52%	[Azzopardi and Petkov, 2013]

(a) MNIST

Test Error	Model
4.5%	[Graham, 2014b]⋆
6.28%	[Graham, 2014a]⋆
8.8%	[Lin et al., 2014]⋆
9.35%	[Goodfellow et al., 2013]⋆
9.39%	[Springenberg and Riedmiller, 2013]⋆
9.5%	[Snoek et al., 2012]⋆
11%	[Krizhevsky et al., 2012]⋆
11.10%	[Wan et al., 2013]⋆
12.35%	**ReNet**
15.13%	[Zeiler and Fergus, 2013]⋆
15.6%	[Hinton et al., 2012]⋆

(b) CIFAR-10

Test Error	Model
1.92%	[Lee et al., 2014]⋆
2.23%	[Wan et al., 2013]⋆
2.35%	[Lin et al., 2014]⋆
2.38%	**ReNet**
2.47%	[Goodfellow et al., 2013]⋆
2.8%	[Zeiler and Fergus, 2013]⋆

(c) SVHN

Table 2: Generalization errors obtained by the proposed ReNet along with those reported by previous works on each of the three datasets. ⋆ denotes a convolutional neural network. We only list the results reported by a single model, i.e., no ensembling of multiple models. In the case of SVHN, we report results from models trained on the Format 2 (cropped digit) dataset only.

Comparative study of segmentation methods

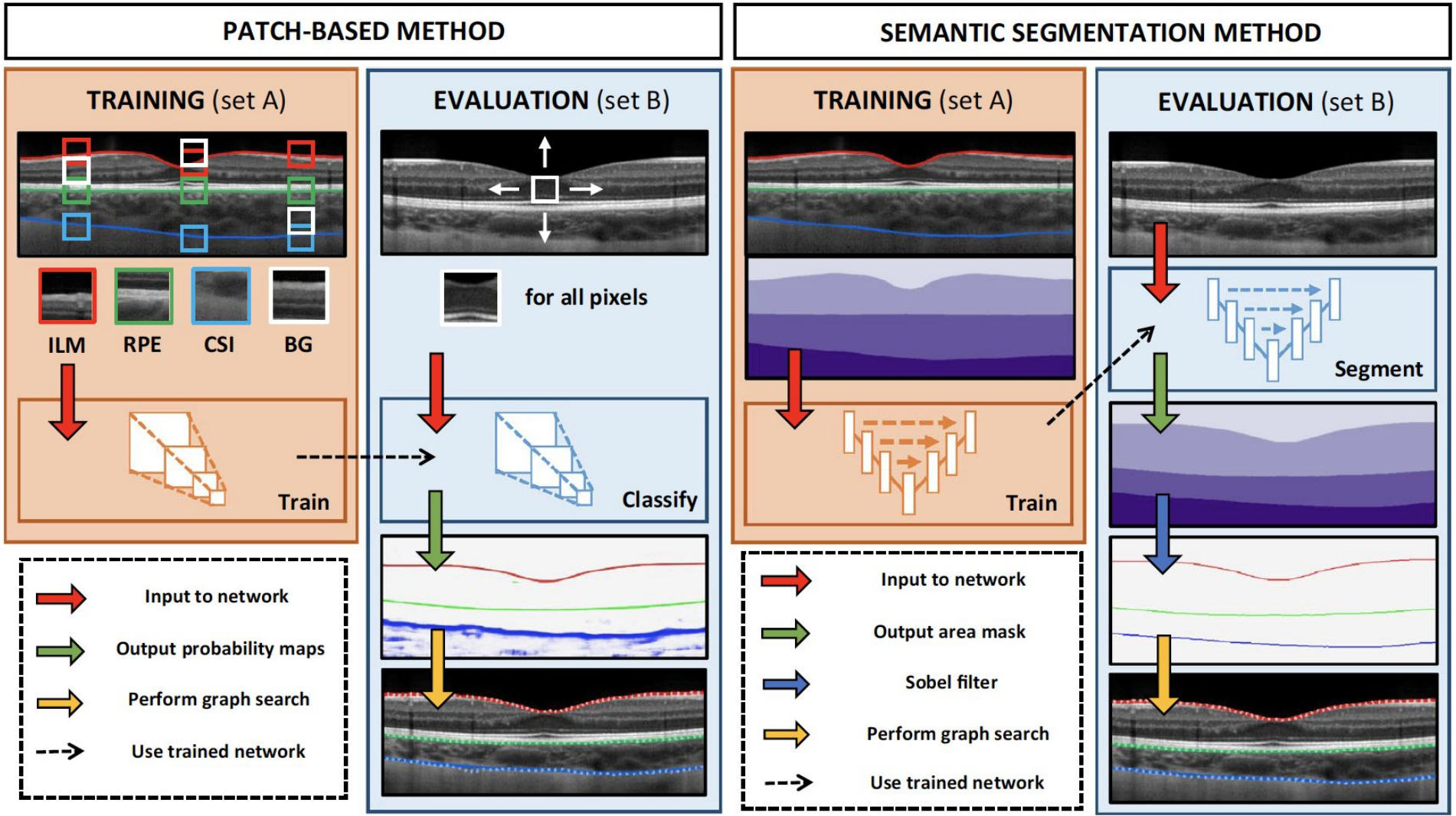
Automatic choroidal segmentation in OCT images using supervised deep learning methods

Jason Kugelman, David Alonso-Caneiro, Scott A. Read, Jared Hamwood, Stephen J. Vincent, Fred K. Chen, Michael J. Collins

https://www.nature.com/articles/s41598-019-49816-4

Outline

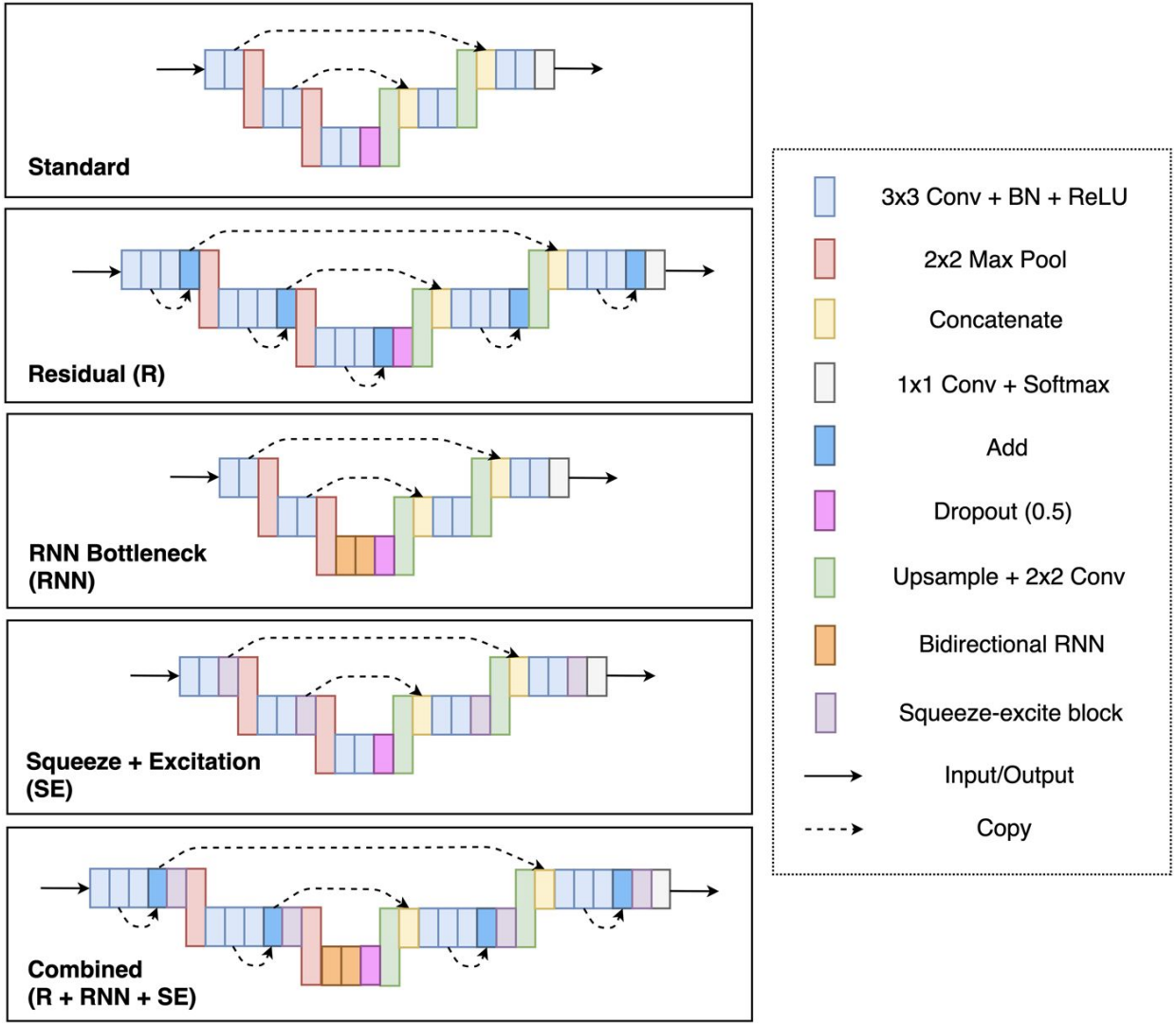
- Compare patch-based methods with semantic segmentation methods
- Use
  - Conventional conv layers
  - Recurrent layers based on ReNets
  - Residual connections
- Domain specific aspect: the goal is to segment a boundary between regions, not the regions themselves.
  - Notice: in general, this is not equivalent to the ‘generic’ segmentation task.
- Next slide shows the experimental protocol for particular types of architectures.
  - Warning: The term ‘semantic segmentation’ used not entirely correctly.


PATCH-BASED METHOD
SEMANTIC SEGMENTATION METHOD
TRAINING (set A)
EVALUATION (set B)
TRAINING (set A)
EVALUATION (set B)
ILM
RPE
CSI
BG
for all pixels
Train
Classify
Train
Segment
Input to network
Output probability maps
Perform graph search
Use trained network
Input to network
Output area mask
Sobel filter
Perform graph search
Use trained network

Compared architectures

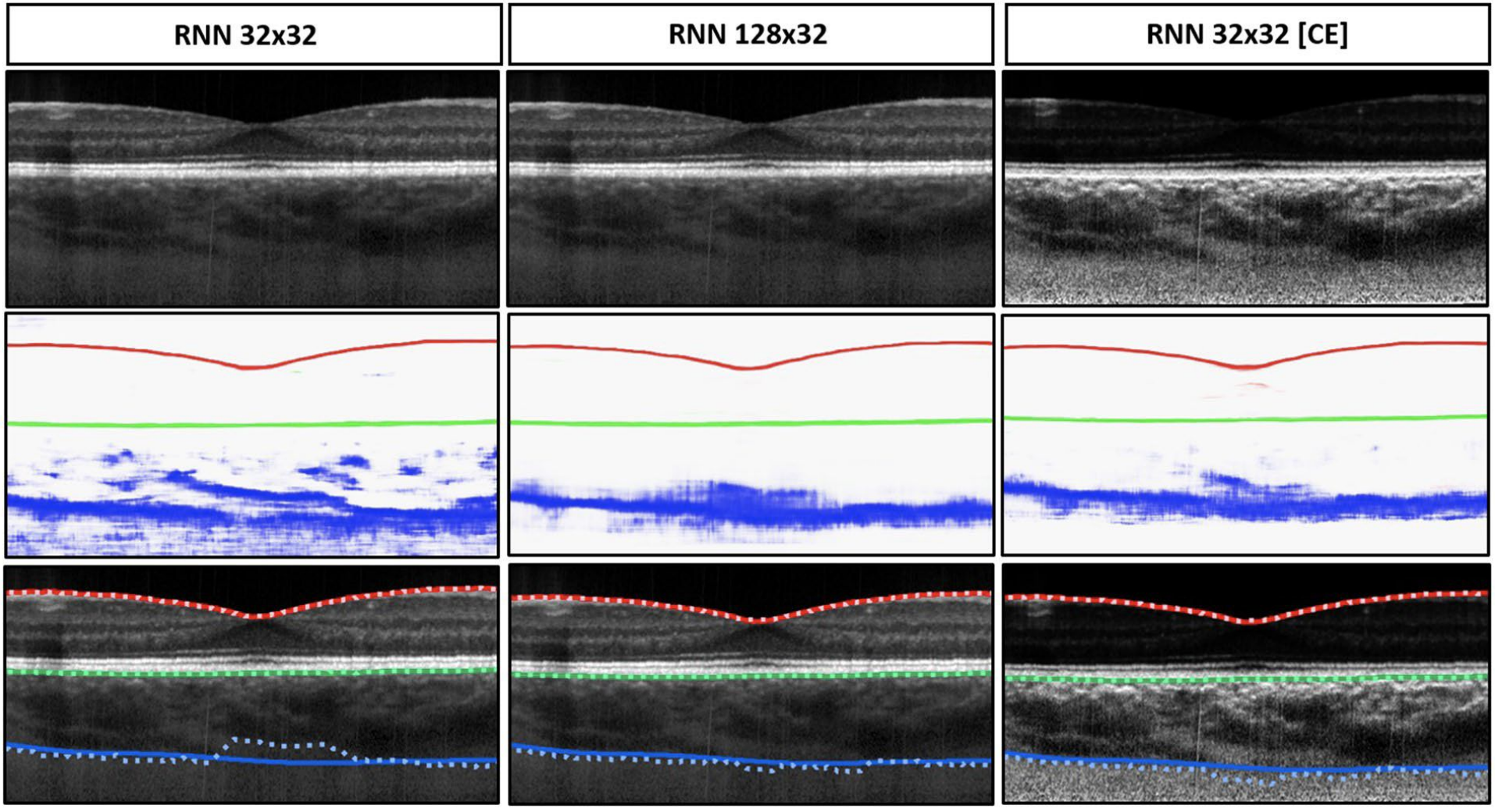
Softmax:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



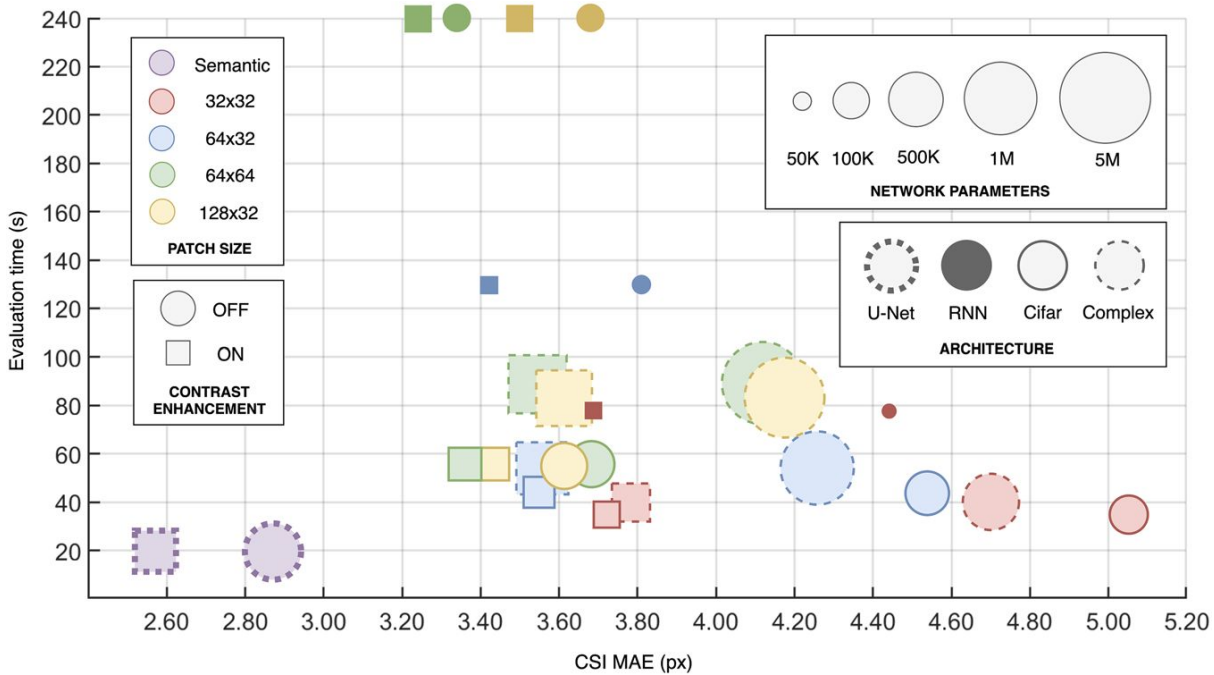
Postprocessing

- Networks produce probability maps (last layer is always SoftMax)
  - These need to be converted into definitive layer locations
- Uses Dijkstra’s shortest path algorithm
- Next slide shows
  - input image
  - probability maps (midlle row);
  - segmented boundary
  - CE = Contrast Enhancement

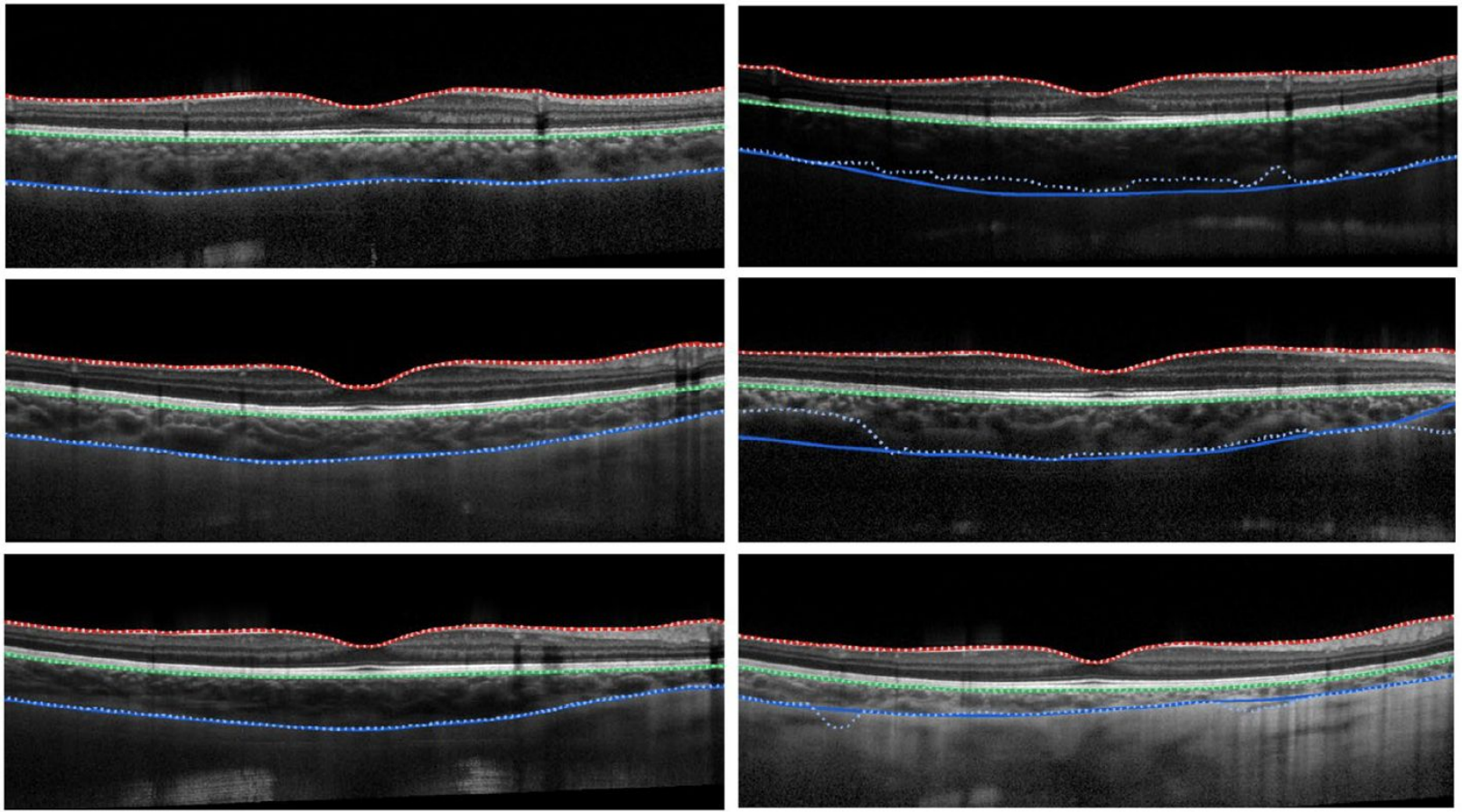

RNN 32x32
RNN 128x32
RNN 32x32 [CE]

Results

- Semantic segmentation (U-nets) significantly better than patch-based methods on some anatomical layers
- No significant differences within the above groups.



Some results for the standard U-Net architecture



Object instance segmentation

Fast R-CNN

Fast R-CNN

Ross Girshick

https://arxiv.org/abs/1504.08083

Core idea

- Object detection
- Based on earlier work: R-CNN, Region proposal CNN (R. Girshick et al., CVPR'14). Addresses the limitations of R-CNN, which:
  - Used SVMs trained on CNN features => training was multi-stage, and not end-to-end.
  - Had high cost of training.
  - Was slow at detection (required a separate CNN forward pass for each object proposal).
- Contributions of Fast R-CNN:
  - End-to-end training
  - Single-stage training
  - Better accuracy

Fast R-CNN

A first (?) NN-based attempt to solve these problems at the same time:

1. object localization,
2. object detection (object vs. non-object),
3. object classification.

However, #1 is not being solved ‘from scratch’ - RoIs must be given. Fast-CNN refines them.

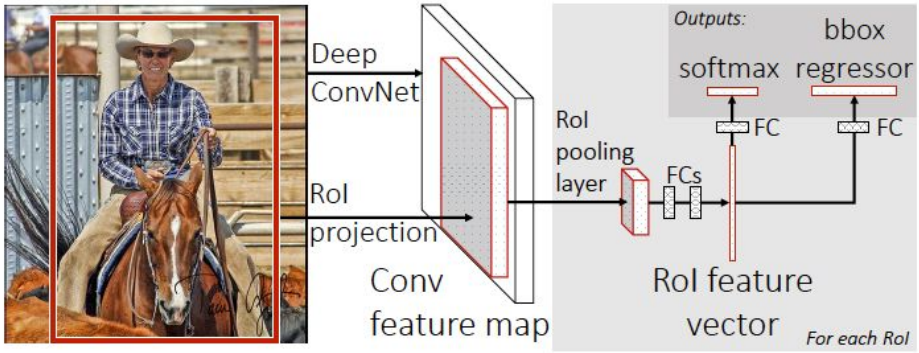


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

Method’s pipeline

- Takes as input an entire image and a set of object proposals.
  - Assumes that region proposals are given.
- Processes the whole image with several convolutional and max pooling layers to produce a feature map.

Then, for each object proposal:

- A region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map (*RoI pooling*).
- Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers:
  - classification softmax: produces softmax probability estimates over K object classes plus a catch-all “background” class, and
  - bbox regressor: another layer that outputs four real-valued numbers for each of the K object classes (refined bounding box positions).

RoI max pooling

Uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7),

Works by:

1. Dividing the hxw RoI window into an HxW grid of sub-windows of approximate size h/H x w/W
2. Max-pooling the values in each sub-window into the corresponding output grid cell.

In other words: crude ‘max-downsampling’ of any RoI to HxW.

Operates independently on each input channel.

Characteristics

- RoIs from the same image share computation and memory in the forward and backward passes
  - With additional extensions (approximating large fc layers with truncated SVD) works almost real-time, when ignoring the time required to produce region proposals.
- Downsides:
  - Needs an external RoI proposal generator.
  - Explicitly iterates over region proposals.
- This issue has been addressed by Faster R-CNN.

Approximating large FC layers with truncated SVD

Replace the (trained) u×v weight matrix W with:

$$W \approx U\Sigma_t V^T$$

Where:

- U: u×t matrix of first t left-singular vectors of W
- Σ_t: t×t diagonal matrix of the top t singular values of W
- V: v×t matrix comprising first t right-singular vectors of W.

Reduces the number of parameters from uv to t(u+v).

Technical realization: two fully connected layers (no nonlinearity between them!):

- The first layer uses Σ_tVT as weight matrix (no biases)
- The second layer used U as weight matrix (and biases taken from W).

Faster R-CNN

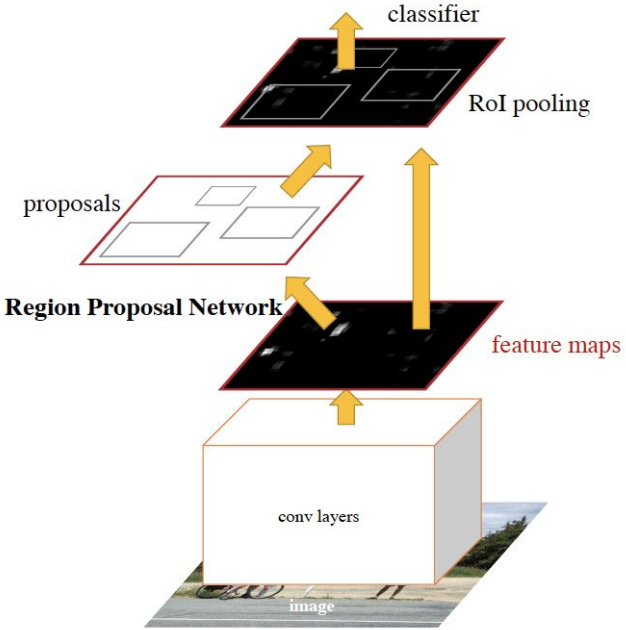
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

https://arxiv.org/abs/1506.01497

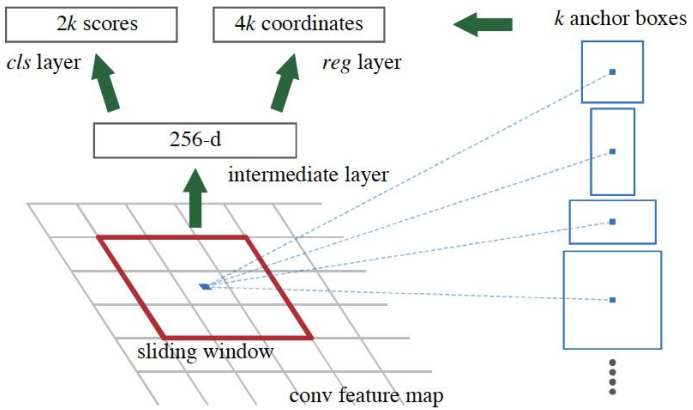
Birds-eye view

- Faster R-CNN is an example of a region proposal method.
  - Originally very expensive computationally.
  - Became more feasible via parallelization with CNNs.
- Proposal computation is nearly cost-free in the context of detection network computation.
- Core components:
  - Region Proposal Network (RPN), deep CNN that proposes regions
  - Fast R-CNN detector (and location refinement)
- These subnetworks share the convolutional layers.
  - In one variant, these are taken from VGG-16.



Region Proposal Network

1. Aggregates a small (3x3) window of the last convolutional layer into a lower-dimensional representation (+ ReLU).
   a. 3x3 may sound not much, but the effective receptive field of VGG is 228x228
2. Separately estimates the position and class using (cf. Fast R-CNN):
   a. Box-regression layer (reg)
   b. Box-classification layer (cls)
3. Important: there is no explicit sliding!
   a. The fully-connected layers are shared!
   b. Technical realization: 1x1 convolutions.
   c. High efficiency.
   d. Exercise: draw this using “box convention”.



Anchors

- Anchor = detection proposal
- Rather than predicting just one region at one location, the *reg* network predicts k of them, for different scales and ratios.
  - Default configuration: 3 scales and 3 ratios => k=9 anchors for each position.
  - Therefore, for, e.g. WxH locations in the convolutional feature map, WHk anchors (e.g. 2400x9 = 21900 anchors for the entire scene/image)
- In parallel to that, the *cls* network predicts 2k *scores*
  - Object vs. non object (hence 2 outputs per class)
  - Two-class softmax layer
  - (The authors admit they could have used one output and logistic regression).
- The upside: the actual number of parameters of reg+cls is surprisingly small:
  - The combined number of outputs: (4+2)x9
  - Total number of parameters (for VGG): 512x(4+2)x9 = 2.8x10^4

Loss function (1): Classification

- Consists in confronting anchors with ground truth boxes.
- Positive label assigned to:
  - The anchor that has the highest intersection over union (IoU) with the GT box, or
  - An anchor that has an IoU overlap higher than 0.7 with *any* GT box.
  - Implication: possibility of multiple positive labels coming from the same GT box.
- Negative labels assigned when an anchor has IoU lower than 0.3 for all GT boxes.
- Notice: no loss/penalty for ‘undecided’ anchors.

Loss function (2): Regression

- L_{reg} defined using robust loss function (smooth L1)

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

- Parameterization/standarization of coordinates for reg

$$\begin{aligned} t_{\mathrm{x}} &= (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}}, \\ t_{\mathrm{w}} &= \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}), \\ t^*_{\mathrm{x}} &= (x^* - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t^*_{\mathrm{y}} = (y^* - y_{\mathrm{a}})/h_{\mathrm{a}}, \\ t^*_{\mathrm{w}} &= \log(w^*/w_{\mathrm{a}}), \quad t^*_{\mathrm{h}} = \log(h^*/h_{\mathrm{a}}), \end{aligned}$$

Loss function (3): Combined

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

where:

- pi - cls output (object/non-object), pi* = 0/1 targets as defined above,
- ti, ti* - predicted and actual coordinates, respectively; notice pi* next to Lreg

More details

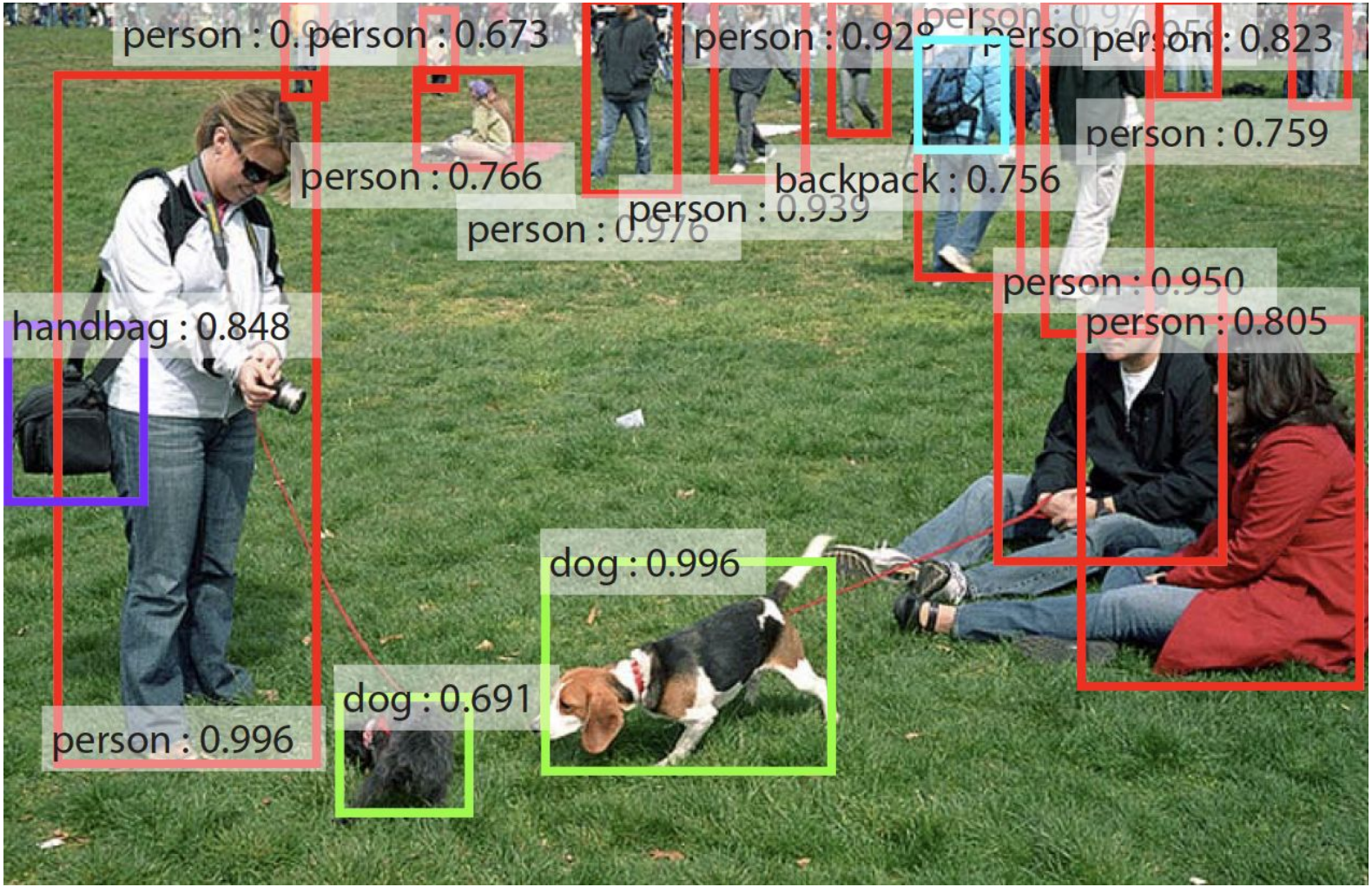
- In training, each mini-batch works with the GT boxes extracted from the same image (‘image-centric approach’)
- Not all generated anchors are taken into account (because of high imbalance: almost all anchors are negative):
  - “Stratified” sampling 256 anchors so that the positive/negative class ratio is close to 1:1 (unless that’s impossible)

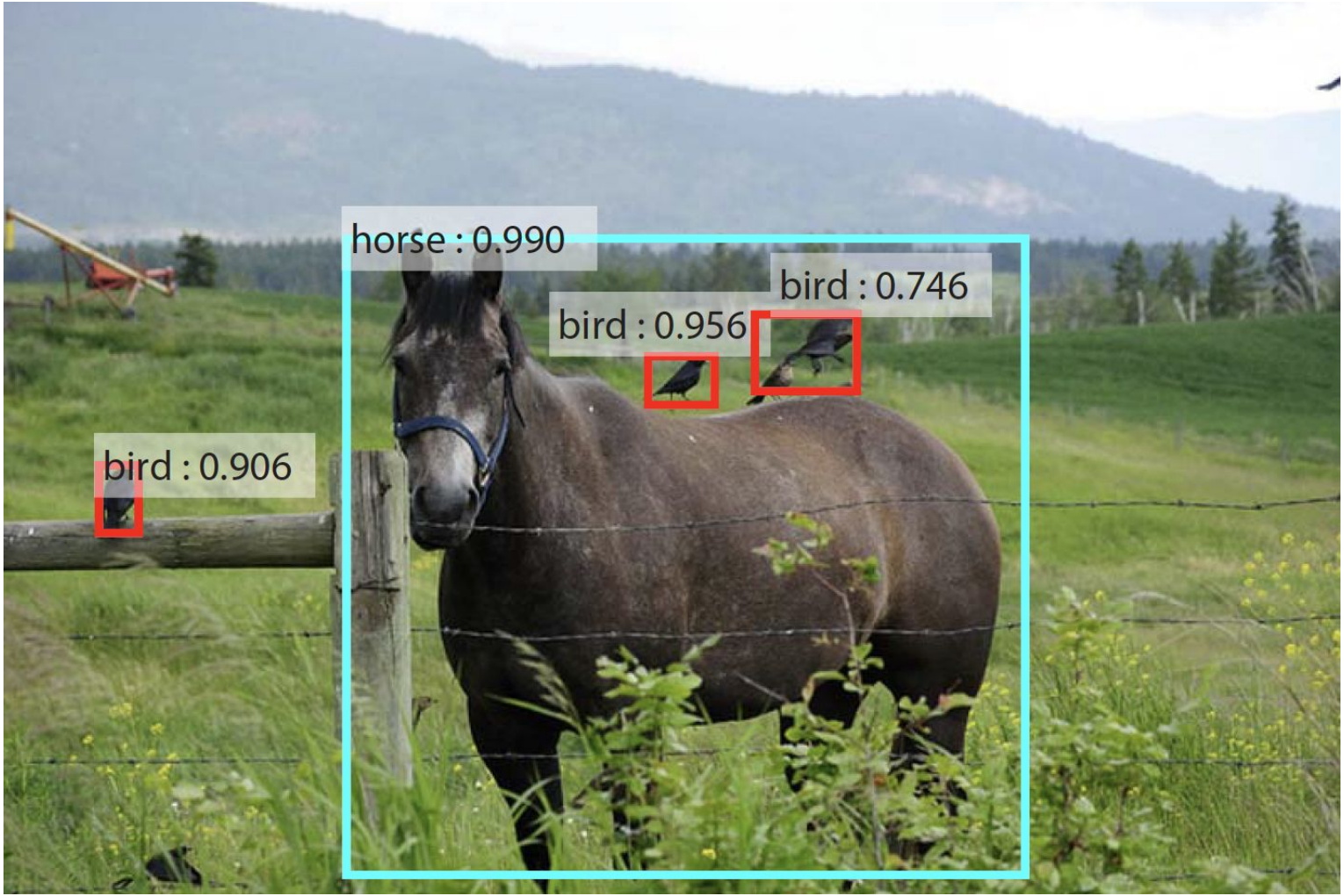
Training

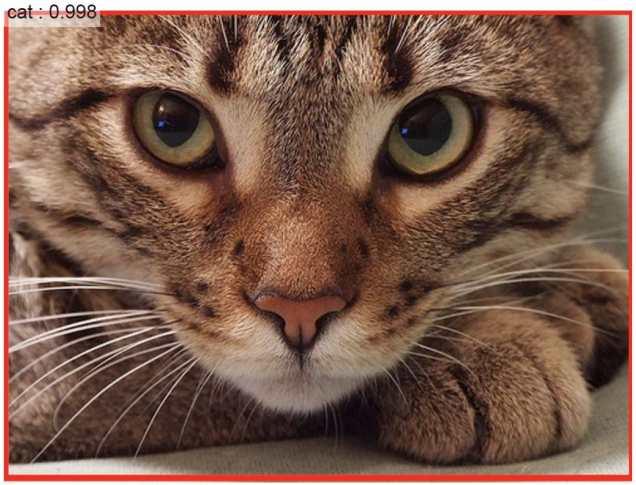
The authors consider a bunch of training regimes for cls and reg:

- Alternating (used as default):
  - First train the RPN
  - Use the region proposals to train Fast R-CNN
  - Repeat
- Approximate joint training: RPN and Fast R-CNN are merged into one network during training.
  - Has some deficiencies (ignores the derivatives w.r.t. proposal coordinates)
- Non-approximate joint training
  - Advanced, handled in another paper.

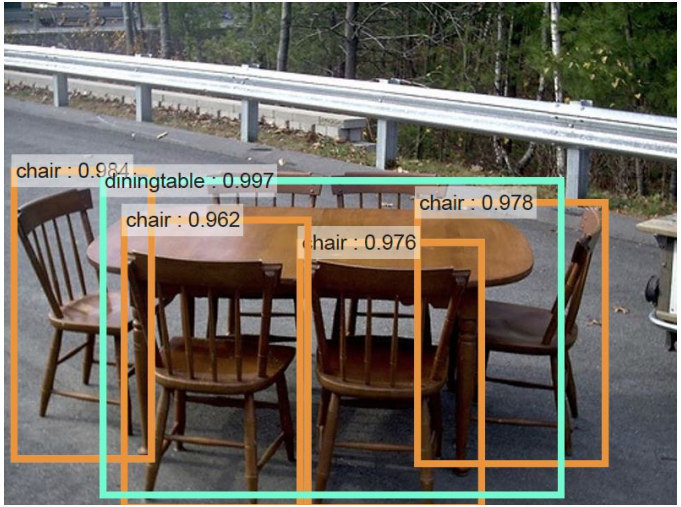
Ultimately the paper uses a hybrid of the above.

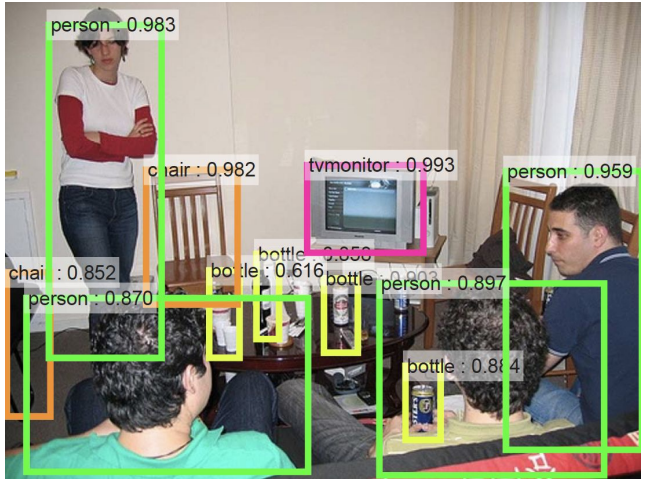

person : 0.673
person : 0.766
person : 0.939
backpack : 0.756
person : 0.759
person : 0.823
person : 0.950
person : 0.805
handbag : 0.848
dog : 0.996
dog : 0.691
person : 0.996

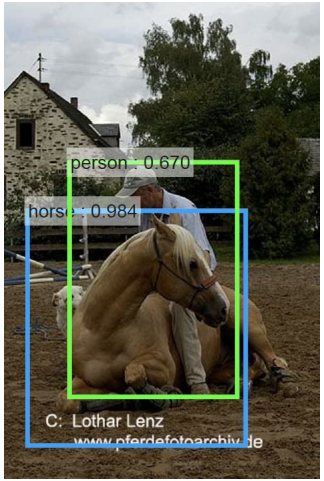

horse : 0.990
bird : 0.746
bird : 0.956
bird : 0.906


cat : 0.998


aeroplane : 0.992
aeroplane : 0.986


chair : 0.984
diningtable : 0.997
chair : 0.962
chair : 0.978
chair : 0.976


person : 0.983
tvmonitor : 0.993
person : 0.959
person : 0.897
person : 0.870


person : 0.670
C: Lothar Lenz

Some comments

- Recall that the convolutional nature of CNNs addresses only the translation invariance.
- Faster R-CNN addresses also scale invariance (to an extent).
- Rotation- and pose-invariance remains a challenge.

Object detection as a regression problem: YOLO

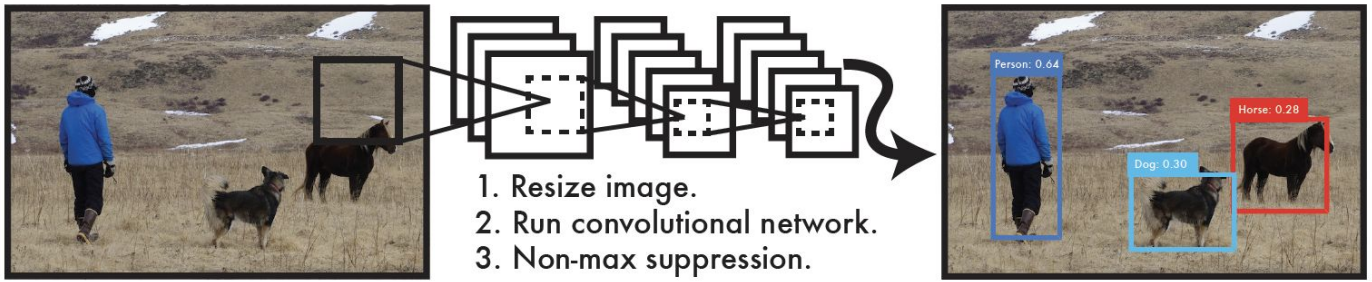
You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

http://arxiv.org/abs/1506.02640

Key contributions

- Frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities.
- A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.
- No sliding window.

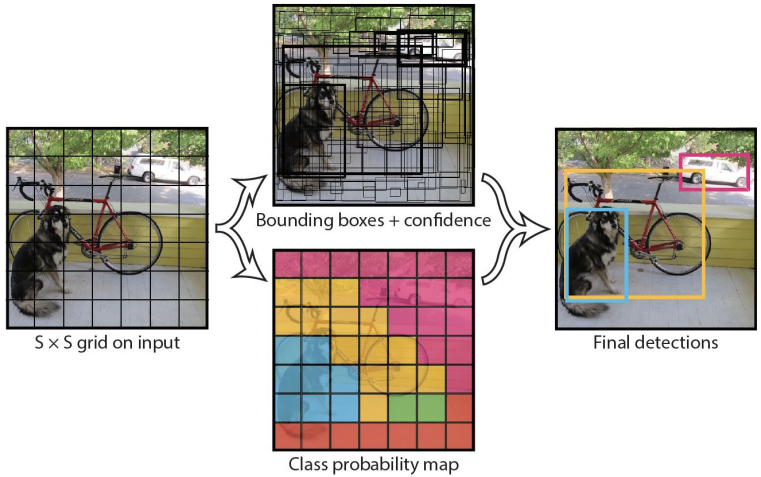


Implications:

- More than twice the mean average precision of other real-time systems.
- Fast: base version 45 fps on Titan X GPU, fast version >150 fps.

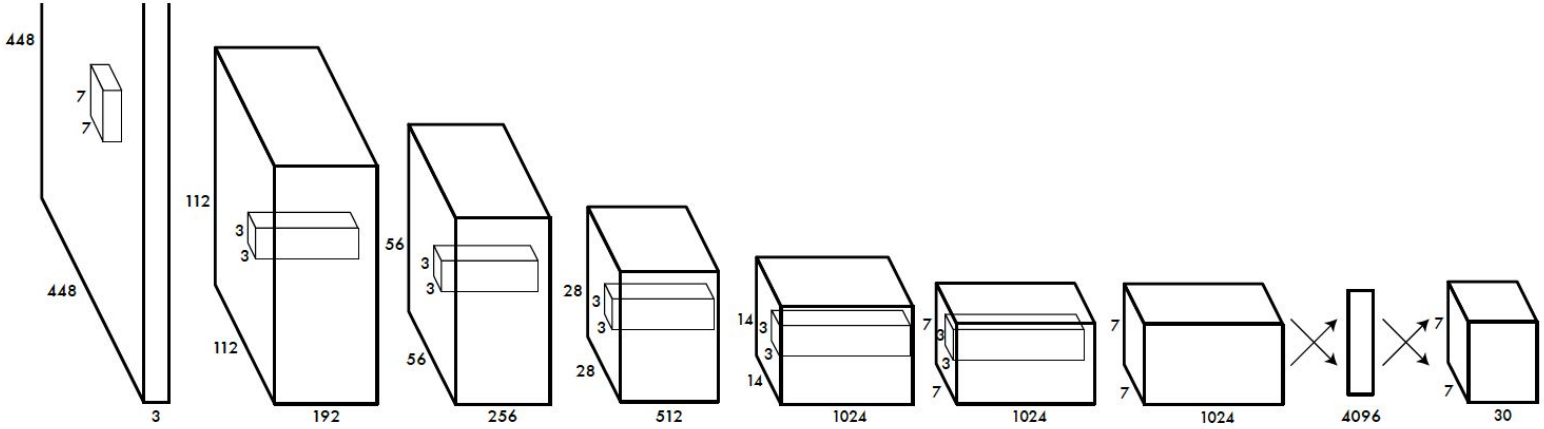
Operation

- Divide image into SxS grid (S=7)
- If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
- Each grid cell predicts
  - B=2 bounding boxes and confidence scores for those boxes.
    - Confidence scores defined as Pr(Object) * IoU(predicted, actual)
  - C conditional class probabilities $Pr(Class_i|Object)$
- Each BB prediction comprises:
  - Center and dimensions of the box: x, y, w, h,
  - Confidence
- Querying: multiplying conditional class probabilities and box-level confidence predictions.
- Output tensor size: SxSx(5B+C)



Architecture

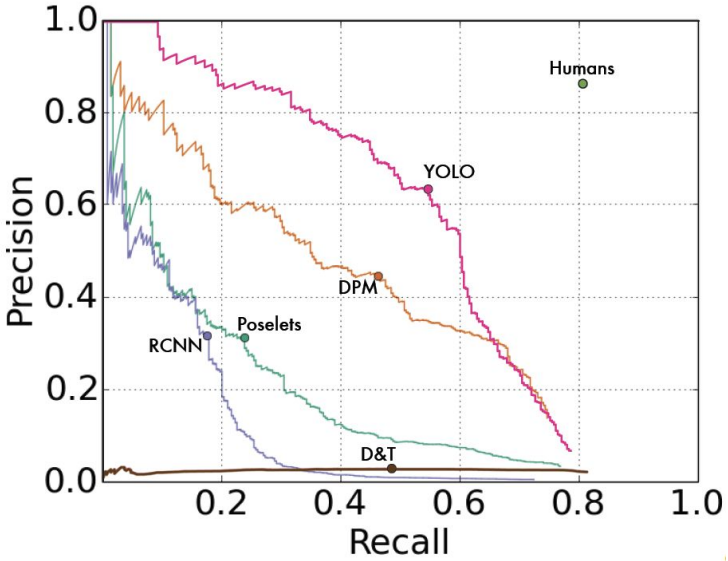
- Inspired by GoogLeNet: 24 conv layers, 2fc layers.
- Pretrained on 1000-class ImageNet



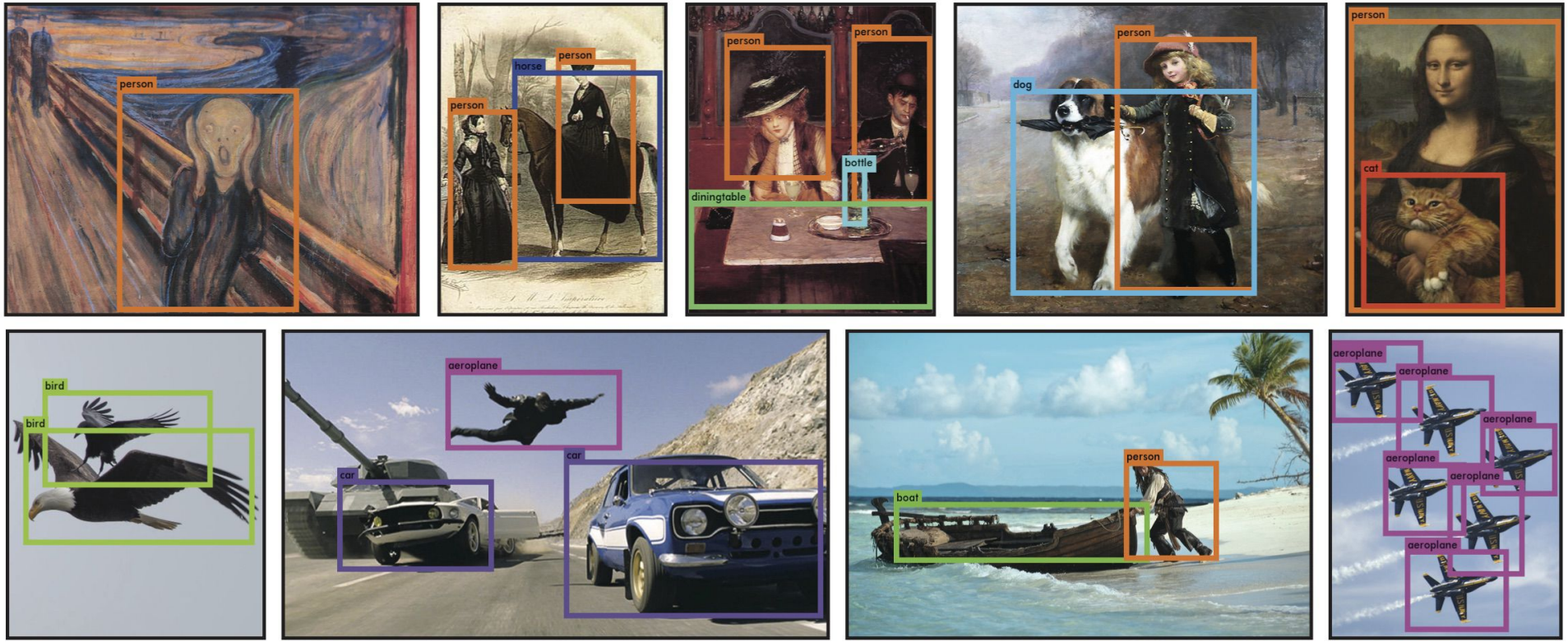
Results

- Fast YOLO: only 9 conv layers

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21



Results: Picasso and People-Art datasets



Other architectures

- Maxout
- Highway nets
- Mask R-CNN, Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick
  https://arxiv.org/abs/1703.06870

Learning higher-level concepts for scene interpretation

Summary

All those impressive results notwithstanding, all that research just scratches the surface of ‘visual intelligence’.

- Focus on mappings: classification (image-level, object-level, pixel-level), segmentation, regression, bounding box regression, …
- Can we design DL agents that really *reason* about scenes?
- More precisely, reason about:
  - Objects and object properties
  - Relationships between objects
  - Spatial arrangement of objects in the scene.
  - ...

Deep learning for scene interpretation

The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, Jiajun Wu

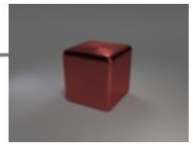
http://arxiv.org/abs/1904.12584

The role of language in scene interpretation

Motivation: humans are capable of learning visual concepts by jointly understanding vision and language.

- The cues: ‘Correlations’ between the content of the scene and the associated statement (see examples below).
- Authors call this *natural supervision*: learning from pairs (image, QA).
- Also known as Visual Query Answering task (VQA)

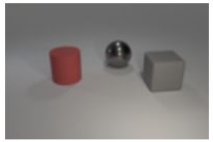
I. Learning basic, object-based concepts.



Q: What’s the color of the object?
A: Red.
Q: Is there any cube?
A: Yes.

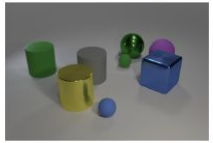
Q: What’s the color of the object?
A: Green.
Q: Is there any cube?
A: Yes.

II. Learning relational concepts based on referential expressions.



Q: How many objects are right of the red object?
A: 2.
Q: How many objects have the same material as the cube?
A: 2

III. Interpret complex questions from visual cues.



Q: How many objects are both right of the green cylinder and have the same material as the small blue ball?
A: 3

CLEVR dataset

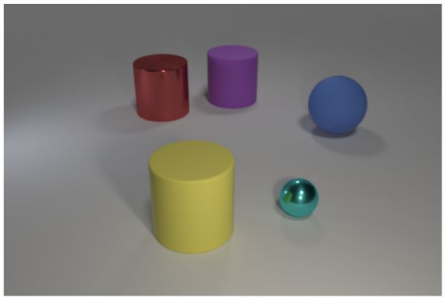
CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, Ross Girshick

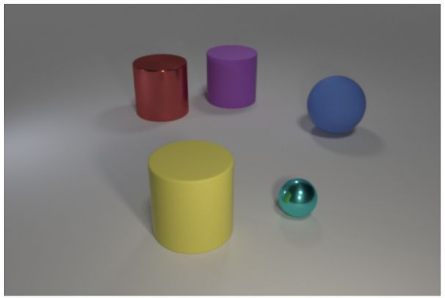
http://arxiv.org/abs/1612.06890

CLEVR dataset

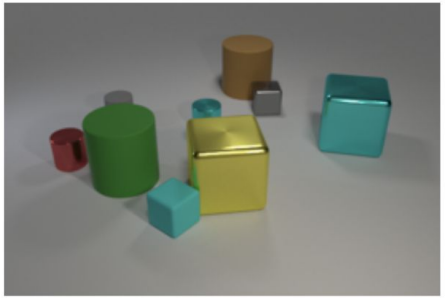
- Represents the class of task of Visual Query Answering (VQA)
- 70k training scenes, rendered in Blender
- Uses symbolic description of scenes
- Scene description is used for scene rendering
- Multiple NL questions can be generated for the same scene



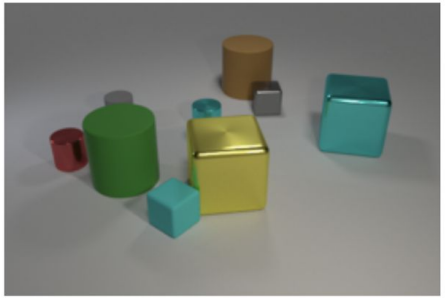
Q: What's the shape of the big yellow thing?



Q: What size is the cylinder that is left of the cyan thing that is in front of the big sphere?



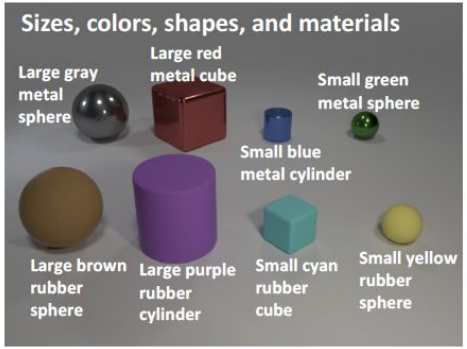
Q: What's the shape of the big yellow thing?

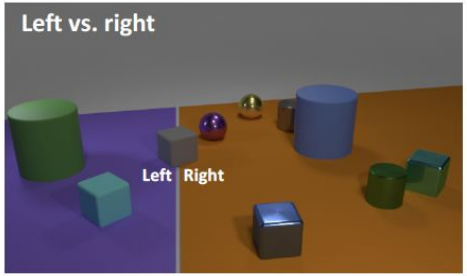


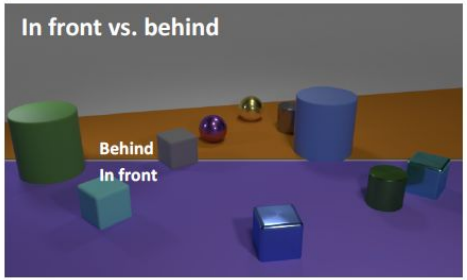
Q: What size is the cylinder that is left of the cyan thing that is in front of the gray cube?

CLEVR dataset

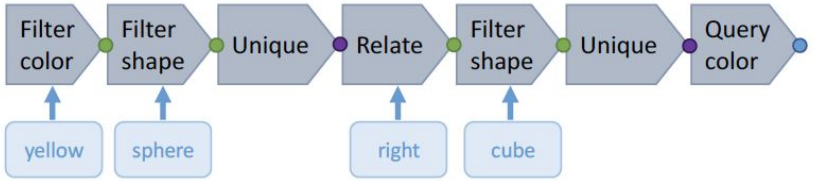
- Objects: three object shapes (cube, sphere, and cylinder), two absolute sizes (small and large), two materials (shiny “metal” and matte “rubber”), and eight colors.
- Relationships: “left”, “right”, “behind”, and “in front”.
- Scene representation: a scene graph, where nodes are objects annotated with attributes and edges connect spatially related objects.
    - Contains all ground-truth information for an image and could be used to replace the vision component of a VQA system with perfect sight.
- Image generation: random sampling a scene graph and rendering it using Blender.
- Question representation: a functional program that can be executed on an image’s scene graph, yielding the answer to the question. Built from simple basic functions that correspond to elementary operations like querying object attributes, counting sets of objects, or comparing values.





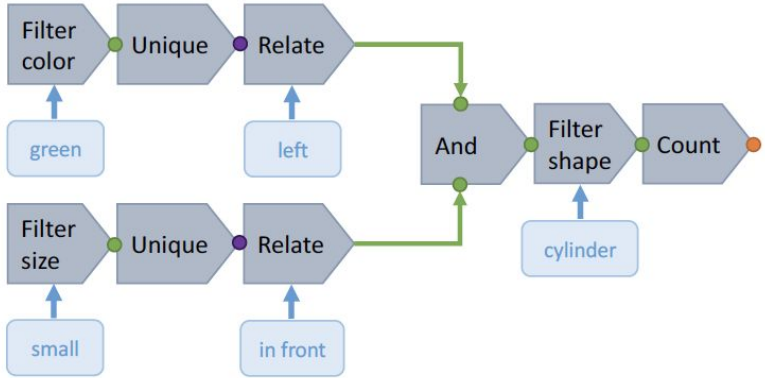


Sample chain-structured question:



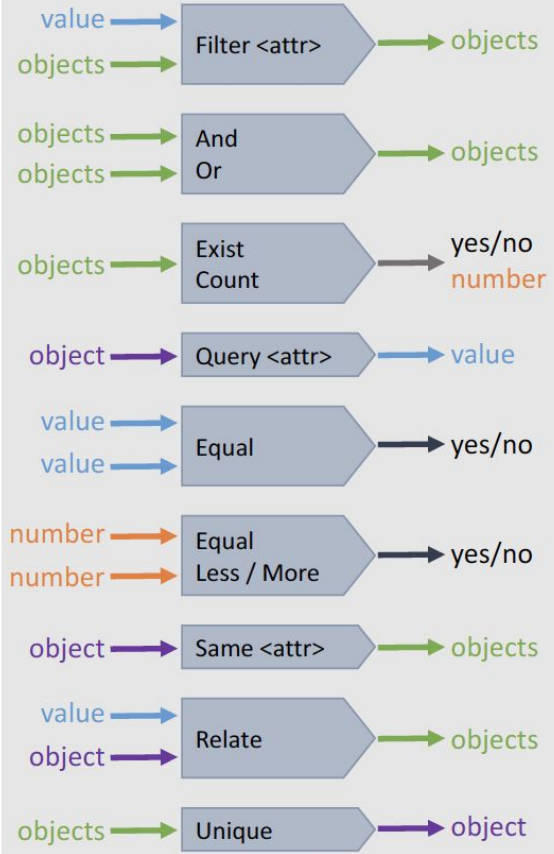
What color is the cube to the right of the yellow sphere?

Sample tree-structured question:



How many cylinders are in front of the small thing and on the left side of the green object?

CLEVR function catalog



Back to NS-CL: Neural symbolic reasoning (?)

We need to define some symbolic framework to reason in.

- Concepts: Cube, Red, … (‘constants’)
- Attributes: Shape, Color, …
  - Concepts are ‘values’ of attributes.
- Relational concepts: Left, Front, …
- Objects
- Set of objects
- Integers
- Booleans

Data Types. Our basic functional building blocks operate on values of the following types:

- `Object`: A single object in the scene.
- `ObjectSet`: A set of zero or more objects in the scene.
- `Integer`: An integer between 0 and 10 (inclusive).
- `Boolean`: Either `yes` or `no`.
- Value types:
  - `Size`: One of `large` or `small`.
  - `Color`: One of `gray`, `red`, `blue`, `green`, `brown`, `purple`, `cyan`, or `yellow`.
  - `Shape`: One of `cube`, `sphere`, or `cylinder`.
  - `Material`: One of `rubber` or `metal`.
- `Relation`: One of `left`, `right`, `in front`, or `behind`.

Note:

- This is actually the type system of CLEVR (see inset on the right).
- Scene interpretation as program synthesis and programmatic inference.

Domain-Specific Language (DSL) for CLEVR

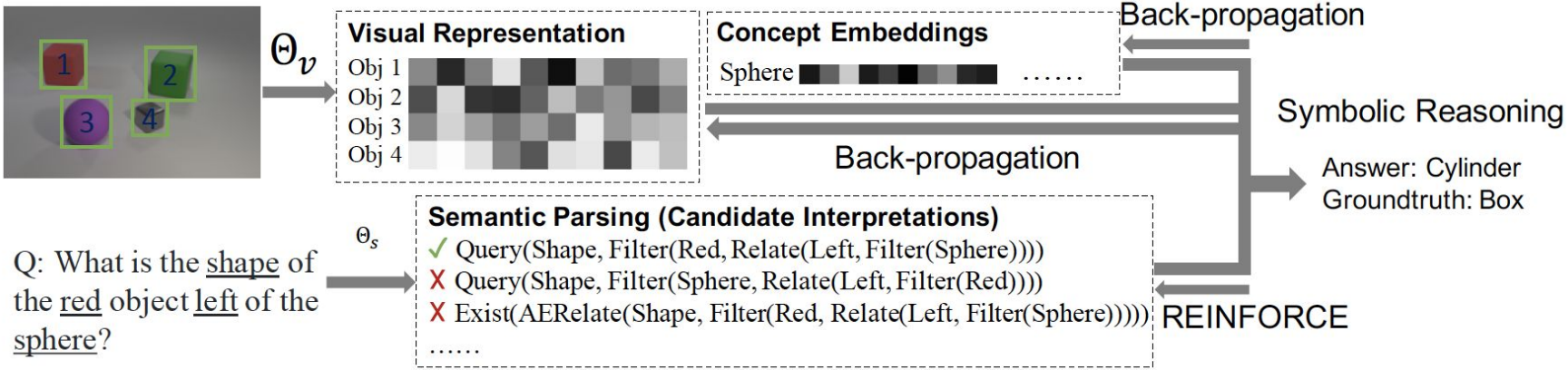
Operation	Signature	Semantics
`Scene`	() $\longrightarrow$ ObjectSet	Return all objects in the scene.
`Filter`	(ObjectSet, ObjConcept) $\longrightarrow$ ObjectSet	Filter out a set of objects having the object-level concept (e.g., red) from the input object set.
`Relate`	(Object, RelConcept) $\longrightarrow$ ObjectSet	Filter out a set of objects that have the relational concept (e.g., left) with the input object.
`AERelate`	(Object, Attribute) $\longrightarrow$ ObjectSet	(Attribute-Equality Relate) Filter out a set of objects that have the same attribute value (e.g., same color) as the input object.
`Intersection`	(ObjectSet, ObjectSet) $\longrightarrow$ ObjectSet	Return the intersection of two object sets.
`Union`	(ObjectSet, ObjectSet) $\longrightarrow$ ObjectSet	Return the union of two object sets.
`Query`	(Object, Attribute) $\longrightarrow$ ObjConcept	Query the attribute (e.g., color) of the input object.

Domain-Specific Language (DSL) for CLEVR

`AEQuery`	(Object, Object, Attribute) $\longrightarrow$ Bool	(Attribute-Equality Query) Query if two input objects have the same attribute value (e.g., same color).
`Exist`	(ObjectSet) $\longrightarrow$ Bool	Query if the set is empty.
`Count`	(ObjectSet) $\longrightarrow$ Integer	Query the number of objects in the input set.
`CLessThan`	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting LessThan) Query if the number of objects in the first input set is less than the one of the second set.
`CGreaterThan`	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting GreaterThan) Query if the number of objects in the first input set is greater than the one of the second set.
`CEqual`	(ObjectSet, ObjectSet) $\longrightarrow$ Bool	(Counting Equal) Query if the number of objects in the first input set is the same as the one of the second set.

NS-CL: Overall architecture

Key feature: use neural symbolic reasoning as a bridge to jointly learn visual concepts, words, and semantic parsing of sentences.

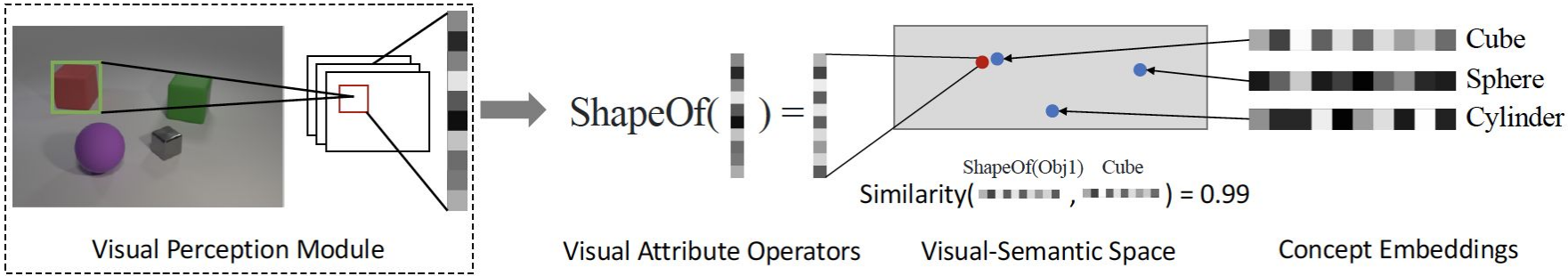


Operation: Three key components

1. Visual perception module (VPM)
   a. Detects objects in the scene
   b. Creates deep latent representation for each object
2. Semantic parsing module (SPM)
   a. Translates an input question in natural language into an executable program in the DSL
3. Program executor
   a. Executes the program upon the derived scene representation and answers the question.
   b. The execution is quasi-symbolic, because the instructions in the DSL, though symbolic, have neural underlying implementation.

Visual Perception Module (VPM)

- Pretrained Mask R-CNN (He et al., 2017) to generate object proposals
- Bounding box for each single object paired with the original image sent to a ResNet-34 (He et al., 2015) to extract respectively the features:
  - Region-based features (RoI, more specifically: RoIAlign ->)
  - Image-based features (to convey contextual information)
  - Concatenated to represent each object.
- Attributes (e.g. Shape and Color) are implemented as neural operators.
- The operators map object representations into a visual-semantic space.



Interlude: Mask R-CNN

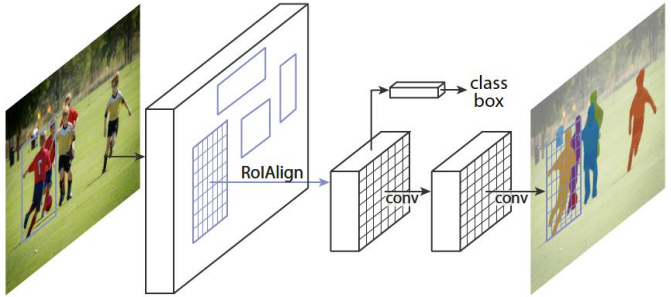
Mask R-CNN

Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick

http://arxiv.org/abs/1703.06870

Extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition.

- The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner.
- The authors propose a simple, quantization-free layer, called RoIAlign, that faithfully preserves exact spatial locations.



Mask R-CNN: ROIAlign

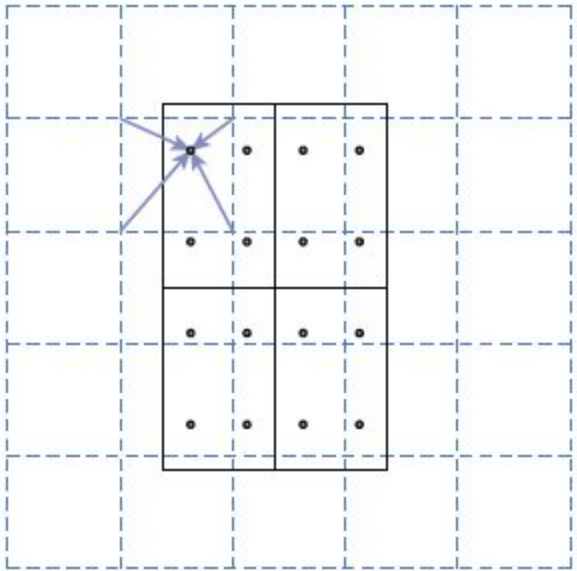
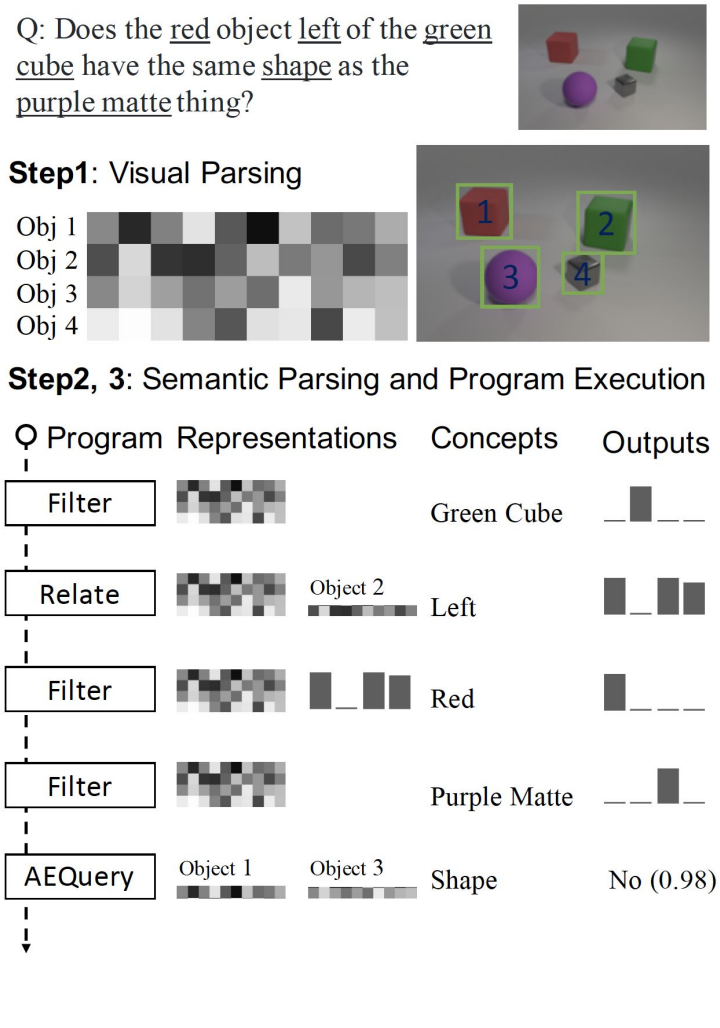


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

Semantic Parser Module (SPM)

- Bidirectional GRU cell (a type of recurrent NN architecture, (Cho et al. 2014))
- Outputs a fixed-length embedding of the entire query
- A decoder based on GRU cells is applied to the embedding, and recovers the hierarchy of operations as the latent program.

Exemplary execution



Training

Recall: the program, though symbolic has neural underlying implementation.

- Each program is thus fully differentiable.
- The outcome of program P is confronted with the correct answer to the visual question and produces reward.
- The reward resulting from program execution is back-propagated through program structure to train VPM and SPM.

Training objective:

$$\Theta_v, \Theta_s \leftarrow \arg\max_{\Theta_v, \Theta_s} \mathbb{E}_P[\Pr[A = \text{Executor}(\text{Perception}(S; \Theta_v), P)]]$$

where S is the scene, A is the answer, P is drawn from SemanticParse(Q;Θ_s),Θ_s are parameters of the parser, Θ_v are parameters of the visual perception module.

Training:

For the visual perception:

$$\Theta_v \text{ as } \nabla_{\Theta_v} \mathbb{E}_P[D_{\mathrm{KL}}(\mathrm{Executor}(\mathrm{Perception}(S;\Theta_v),P)\|A)]$$

For the semantic parser: REINFORCE (Williams, 1992)

$$\nabla_{\Theta_s} = \mathbb{E}_P[r \cdot \log \Pr[P = \mathrm{SemanticParse}(Q;\Theta_s)]]$$

reward r = 1 if the answer is correct and 0 otherwise

Quasi-symbolic execution

ObjectSet is a vector of probabilities corresponding to objects in the scene (probability that the i-th object of the scene belongs to the set).

Signature	Implementation
`Scene()` → *out*: ObjectSet	$out_i := 1$
`Filter`(*in*: ObjectSet, *oc*: ObjConcept) → *out*: ObjectSet	$out_i := \min(in_i, \text{ObjClassify}(oc)_i)$
`Relate`(*in*: Object, *rc*: RelConcept) → *out*: ObjectSet	$out_i := \sum_j (in_j \cdot \text{RelClassify}(rc)_{j,i}))$
`AERelate`(*in*: Object, *a*: Attribute) → *out*: ObjectSet	$out_i := \sum_j (in_j \cdot \text{AEClassify}(a)_{j,i}))$
`Intersection`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *out*: ObjectSet	$out_i := \min(in_i^{(1)}, in_i^{(2)})$
`Union`($in^{(1)}$: ObjectSet, $in^{(2)}$: ObjectSet) → *out*: ObjectSet	$out_i := \max(in_i^{(1)}, in_i^{(2)})$
`Query`(*in*: Object, *a*: Attribute) → *out*: ObjConcept	$\Pr[out = oc] := \sum_i in_i \cdot \frac{\text{ObjClassify}(oc)_i \cdot b_a^{oc}}{\sum_{oc'} \text{ObjClassify}(oc')_i \cdot b_a^{oc'}}$
`AEQuery`($in^{(1)}$: Object, $in^{(2)}$: Object, *a*: Attribute) → *b*: Bool	$b := \sum_i \sum_j (in_i^{(1)} \cdot in_j^{(2)} \cdot \text{AEClassify}(a)_{j,i}))$

Comparison to related work

Two approaches in related work on semantic sentence parsing for visual reasoning:

- implicit programs as conditioned neural operations,
- explicit programs as sequences of symbolic tokens
  - Problem: to learn, require extra supervision, e.g. tround-truth program annotations
  - NSCL: uses visual grounding as distant supervision to parse questions in natural languages into explicit programs, with zero program annotations.

Results

Questions of the type
“How many red objects are there?”

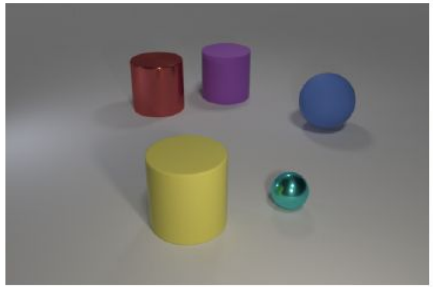
	Visual	Mean	Color	Mat.	Shape	Size
IEP	Conv.	90.6	91.0	90.0	89.9	90.6
MAC	Attn.	95.9	98.0	91.4	94.4	94.2
TbD (hres.)	Attn.	96.5	96.6	92.2	95.4	92.6
NS-CL	Obj.	**98.7**	**99.0**	**98.7**	**98.1**	**99.1**

Performance on different types of questions

Model	Prog. Anno.	Overall	Count	Cmp. Num.	Exist	Query Attr.	Cmp. Attr.
Human	N/A	92.6	86.7	86.4	96.6	95.0	96.0
NMN	700K	72.1	52.5	72.7	79.3	79.0	78.0
N2NMN	700K	88.8	68.5	84.9	85.7	90.0	88.8
IEP	700K	96.9	92.7	98.7	97.1	98.1	98.9
DDRprog	700K	98.3	96.5	98.4	98.8	99.1	99.0
TbD	700K	**99.1**	**97.6**	**99.4**	**99.2**	**99.5**	**99.6**
RN	0	95.5	90.1	93.6	97.8	97.1	97.9
FiLM	0	97.6	94.5	93.8	99.2	99.2	99.0
MAC	0	98.9	97.2	**99.4**	**99.5**	99.3	**99.5**
NS-CL	0	**98.9**	**98.2**	99.0	98.8	**99.3**	99.1

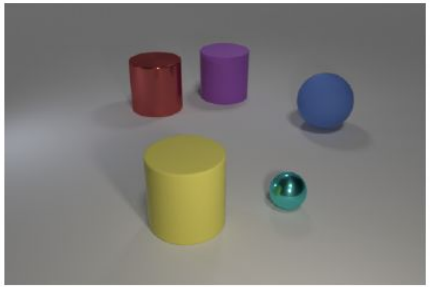
Generalizing to new visual compositions

Split A



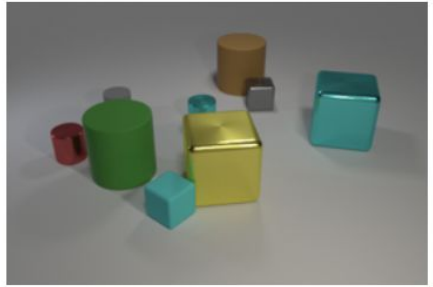
Q: What's the shape of the big yellow thing?

Split B



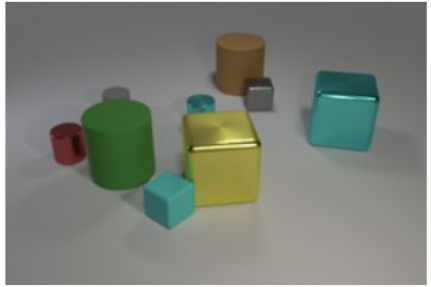
Q: What size is the cylinder that is left of the cyan thing that is in front of the big sphere?

Split C



Q: What's the shape of the big yellow thing?

Split D



Q: What size is the cylinder that is left of the cyan thing that is in front of the gray cube?

Results: Data efficiency

Can we learn efficiently from small numbers of examples?

Model	Visual	Accuracy (100% Data)	Accuracy (10% Data)
TbD	Attn.	99.1	54.2
TbD-Object	Obj.	84.1	52.6
TbD-Mask	Attn.	99.0	55.0
MAC	Attn.	98.9	67.3
MAC-Object	Obj.	79.5	51.2
MAC-Mask	Attn.	98.7	68.4
NS-CL	Obj.	**99.2**	**98.9**

Summary

Neuro-Symbolic Concept Learner:

- Achieves state-of-the-art performance on the CLEVR dataset
- Naturally learns disentangled visual and language concepts
- Capable of combinatorial generalization w.r.t. both visual scenes and semantic programs, more specifically generalization to:
  - scenes with more objects and longer semantic programs
  - new visual attribute compositions
  - novel visual concepts, such as learning a new color.
  - new tasks, such as image-caption retrieval, without any extra fine-tuning (all shown in the paper).
- Explicit program semantics enjoys compositionality, interpretability, and generalizability.