Deep Learning Crash Course

Hui Xue

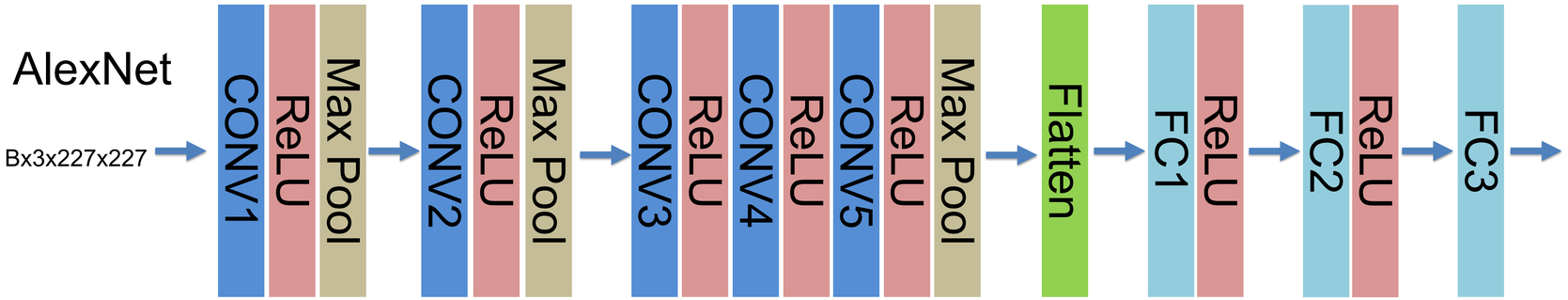
Fall 2021

www.deeplearningcrashcourse.org

Outline

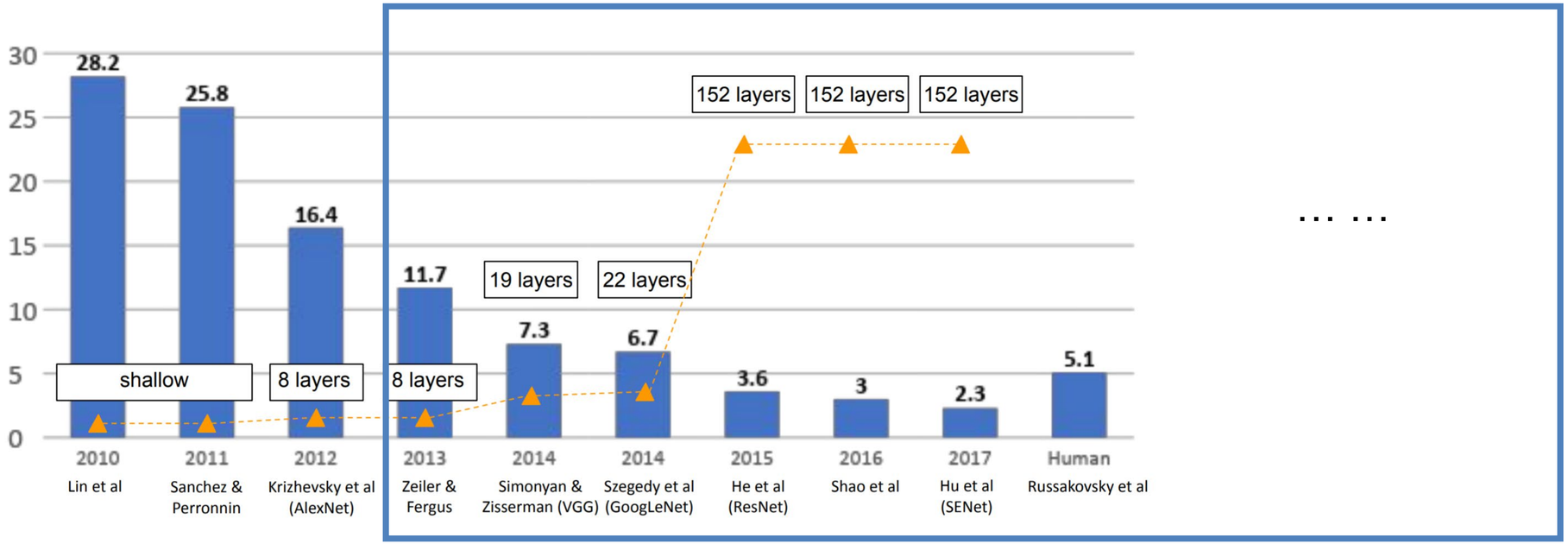
- CNN architectures
- Object detection
- Segmentation
- Denoising and super-resolution

ImageNet winners



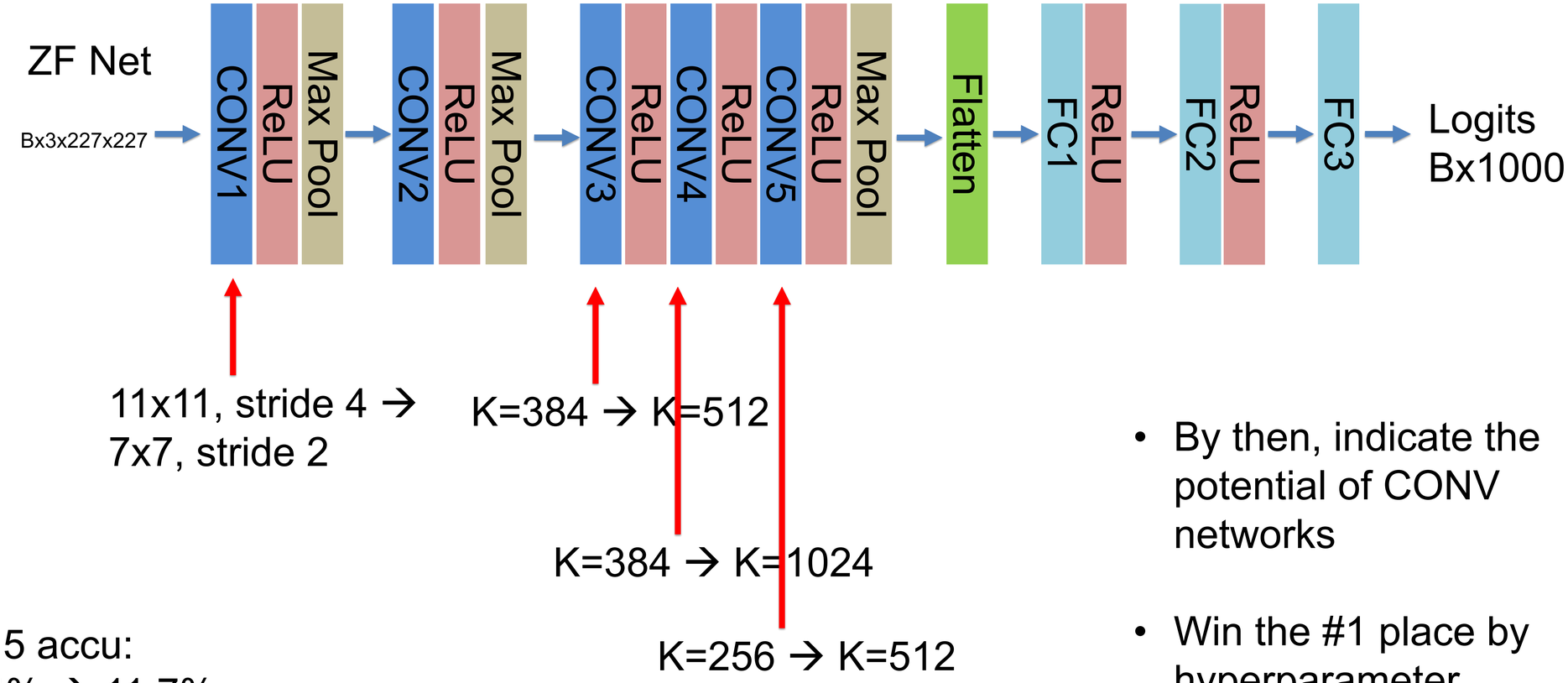
~61M parameters

- Backprop to train CONV
- Use ~~ave~~ max pooling
- Use ~~tanh~~ ReLU
- Use flatten and FC
- Use softmax and CE loss
- Increase number of filters with reduced spatial resolution
- Much bigger
- Data augmentation
- Use GPU for training
- Much larger dataset



… …

Z-F net, 2013



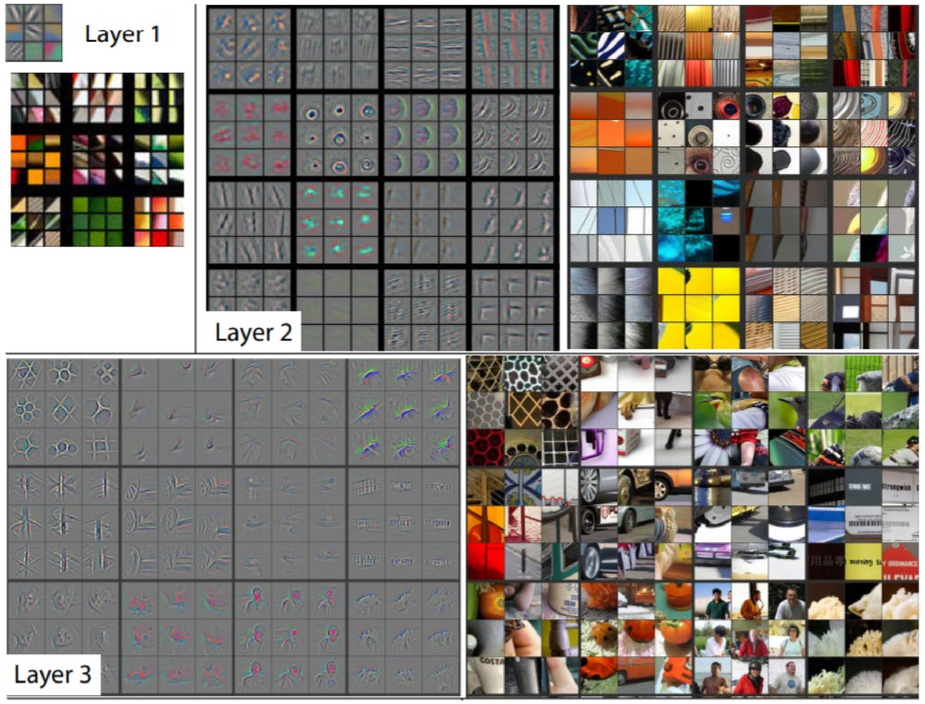
Top 5 accu:
16.4% → 11.7%

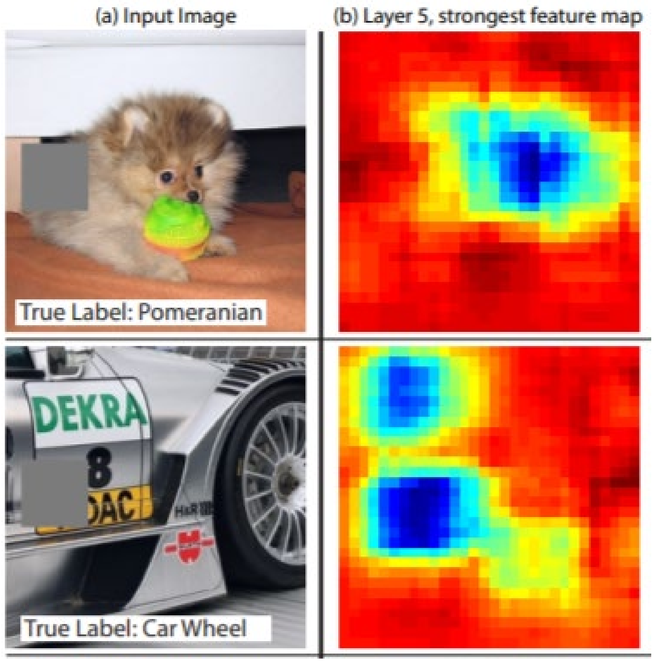
- By then, indicate the potential of CONV networks
- Win the #1 place by hyperparameter searching

Visualizing and Understanding Convolutional Networks. https://arxiv.org/abs/1311.2901

Z-F net paper has other contributions

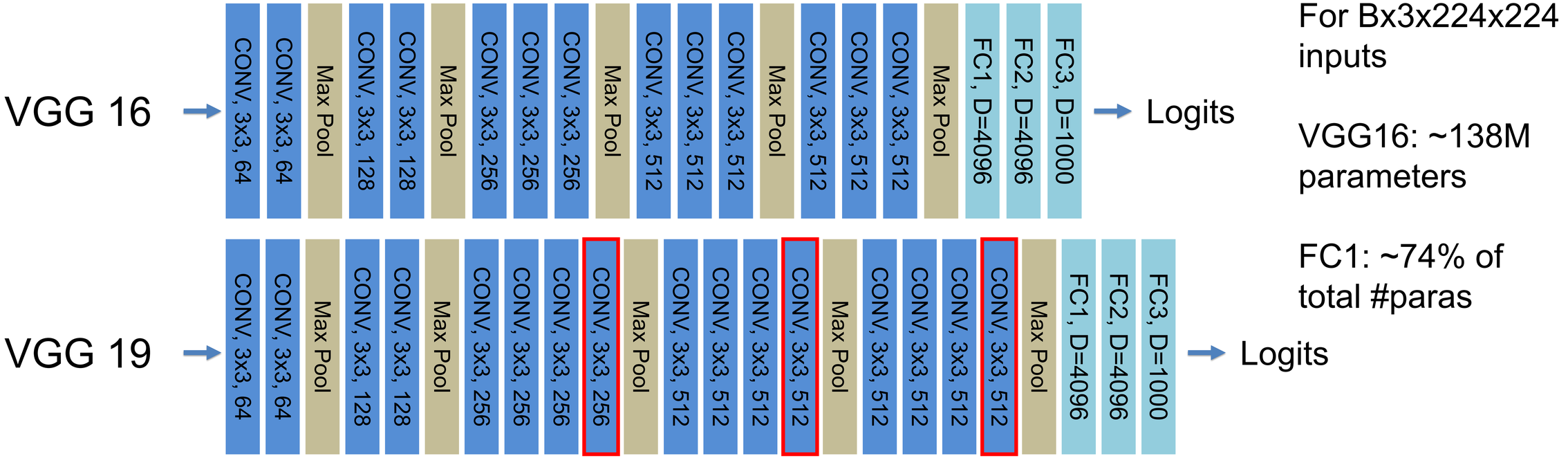
- A seminal paper for neural network visualization
- Insights to understand convnet
- More in the NN visualization later





Visualizing and Understanding Convolutional Networks. https://arxiv.org/abs/1311.2901

VGG net, 2014



For Bx3x224x224 inputs

VGG16: ~138M parameters

FC1: ~74% of total #paras

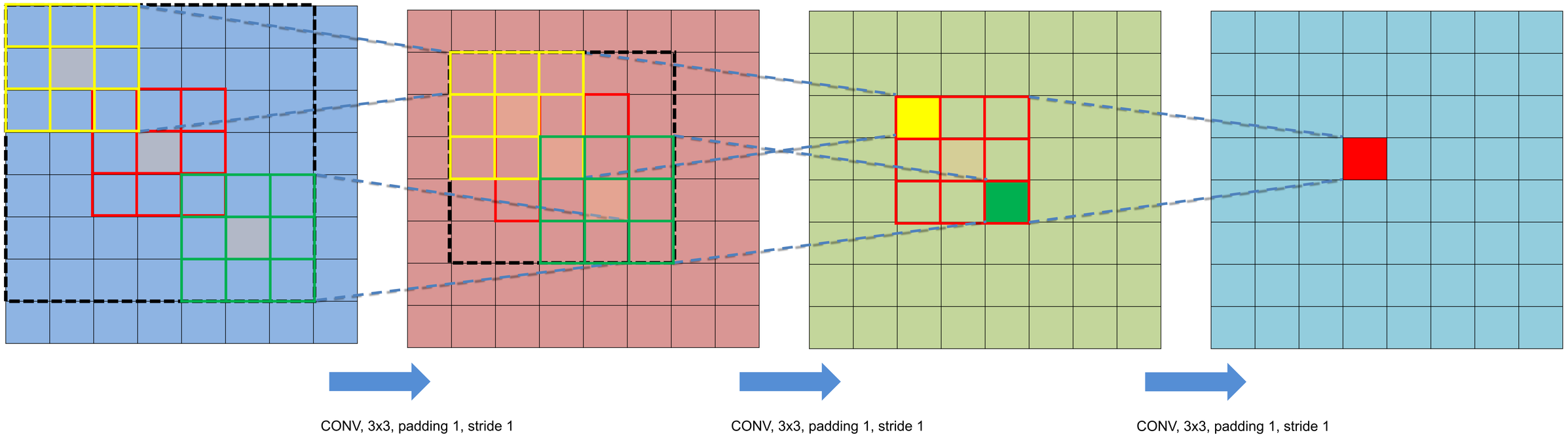
- Go deeper!
- From 8 layers with learnable weights in the Alexnet to 16 – 19 layers with weights
- Only use 3x3 conv kernel with stride 1 → do not change image size
- Solely relying on max pooling to reduce image size
- ReLU after every CONV
- CONV does not change image spatial size
- Increase #output channels with reduced image size

Top 5 accu:
11.7% → 7.3%

Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556

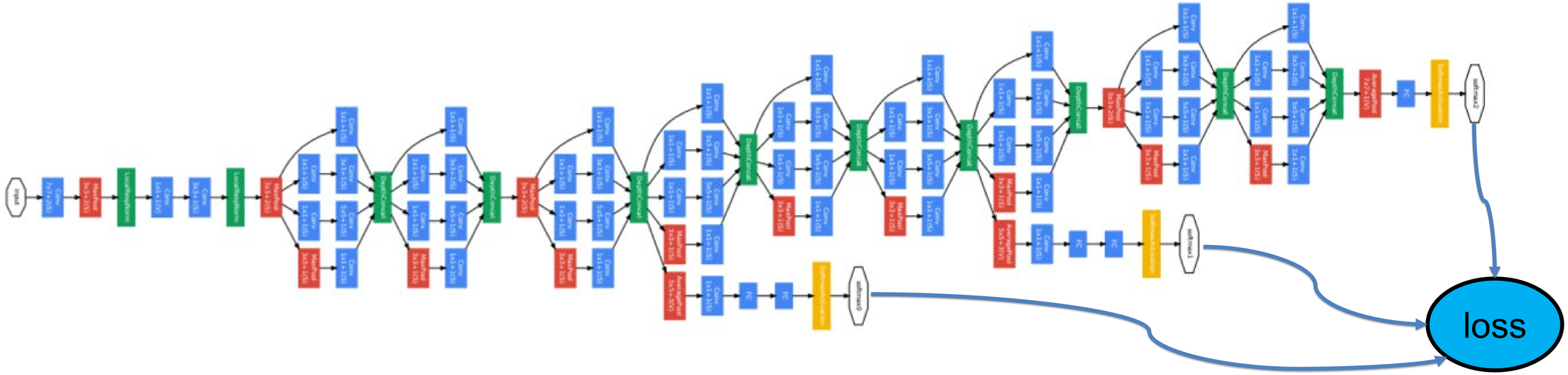
VGG net, 2014

- Stack small CONV layers to gain larger receptive fields
- Reduced computation
- Cleaner architecture design
- Still has the expensive FC layer



Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556

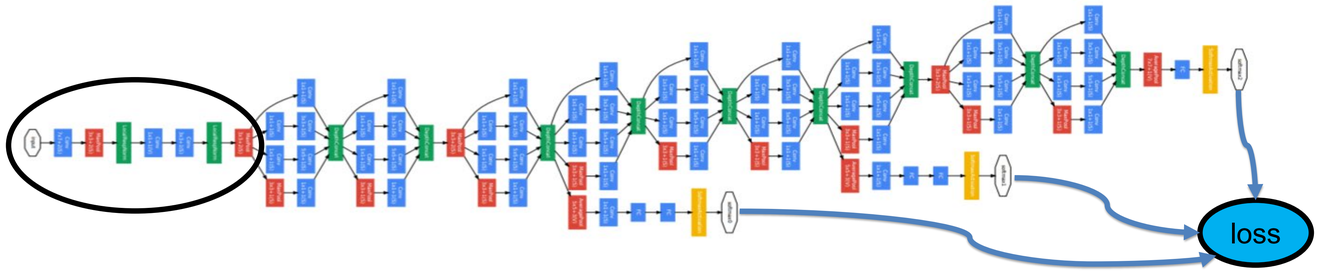
Google Inception net, 2014



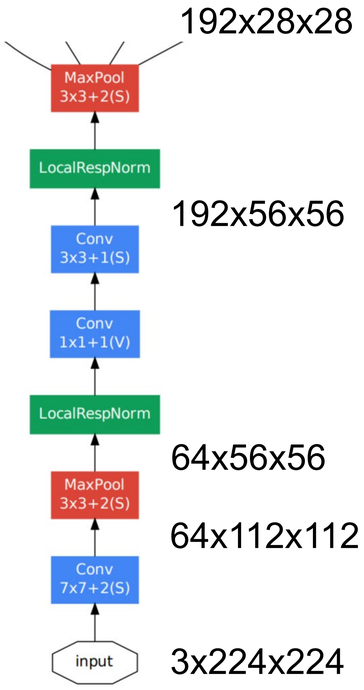
- Go deeper and **go wider** !
- 22 layers
- Inception module
- Extensively using 1x1 conv layer
- Remove the expensive FC layer
- Use intermediate feature maps for loss computation
- Only ~5M parameters

Top 5 accu:
11.7% → 6.7%

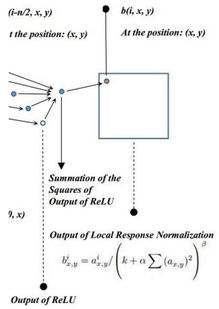
Google Inception net, 2014



Staring phase:

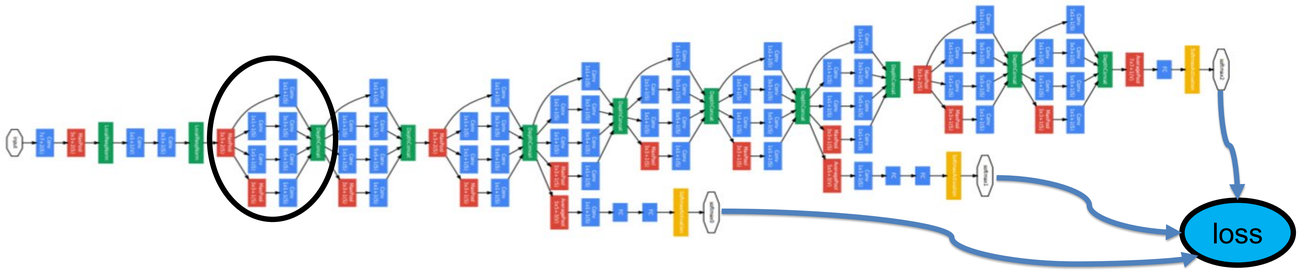


Local response normalization



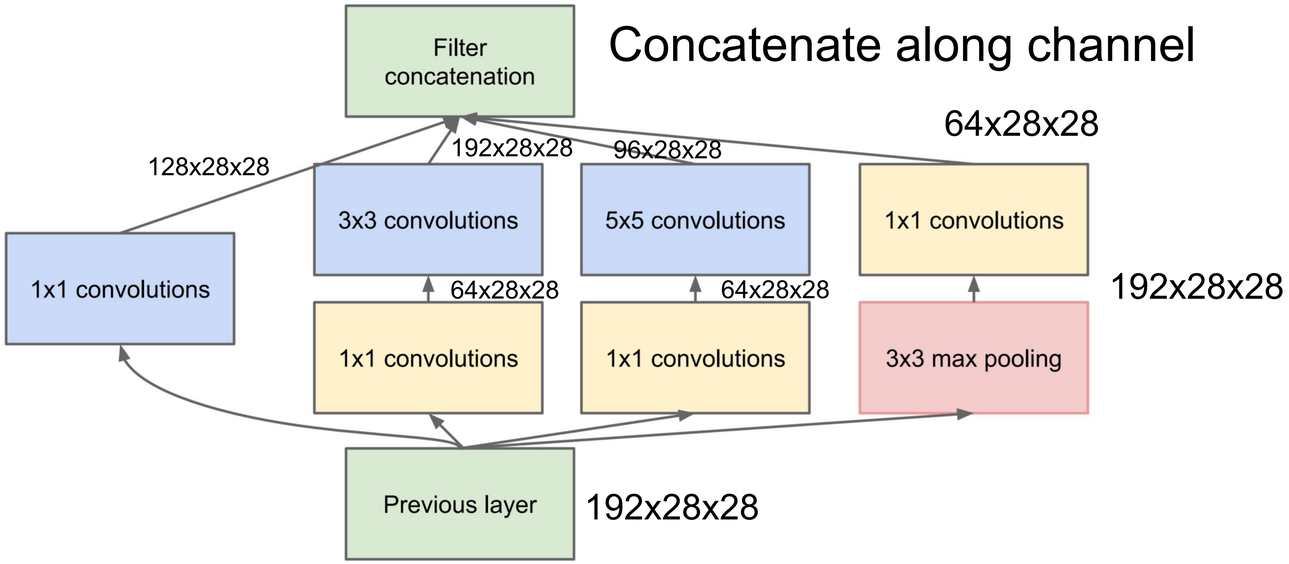
https://www.quora.com/What-is-local-response-normalization

Google Inception net, 2014



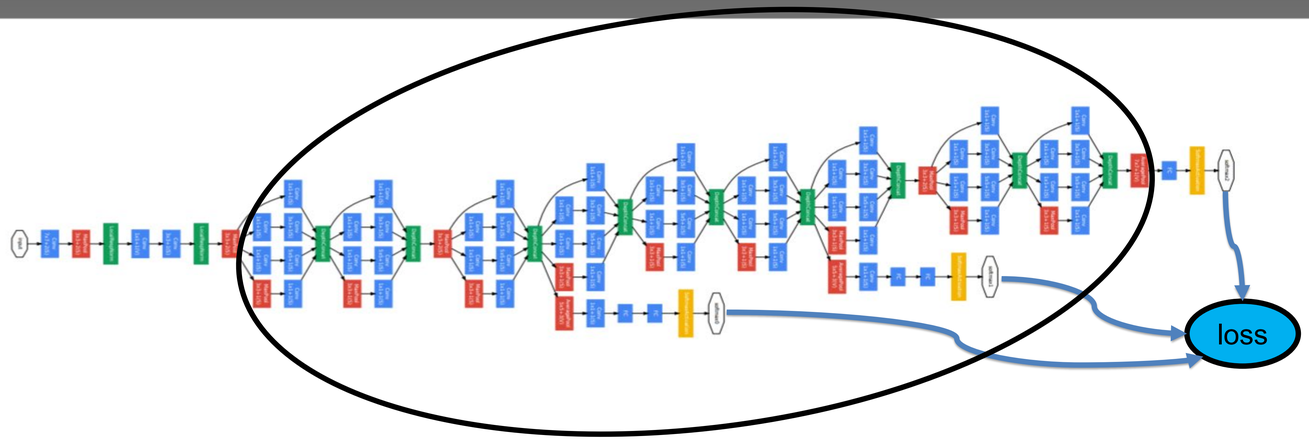
(128+192+96+64) x28x28

Inception module:

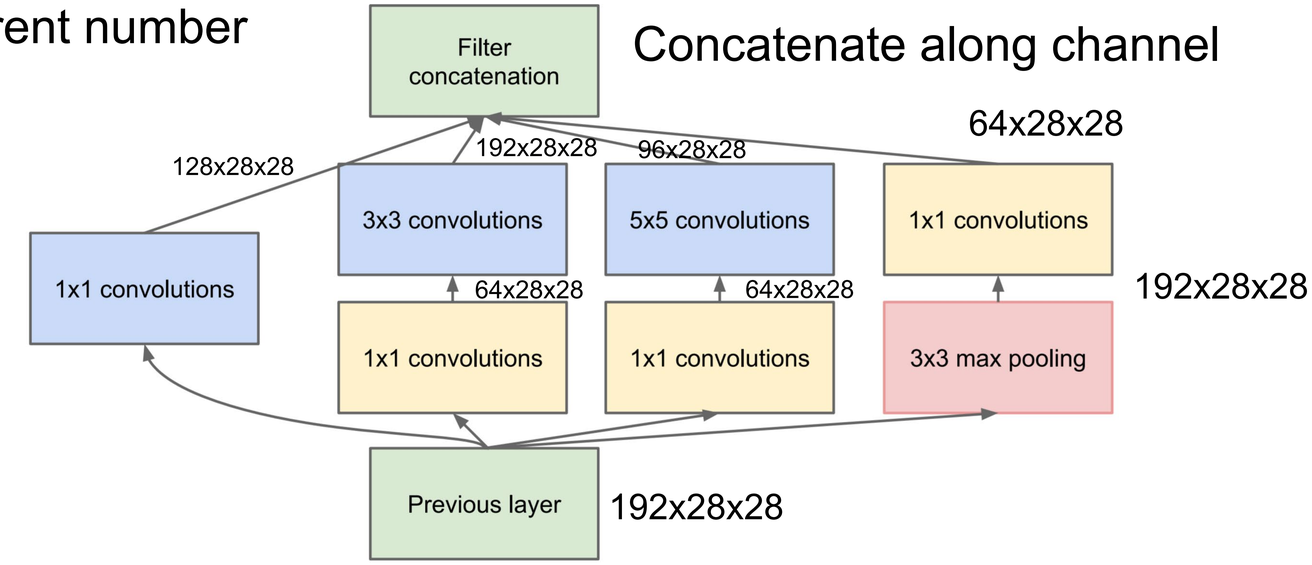


- Instead of having a single pass, having multiple passes
- Use 1x1 conv to reduce computation
- Concatenate along channels

Google Inception net, 2014

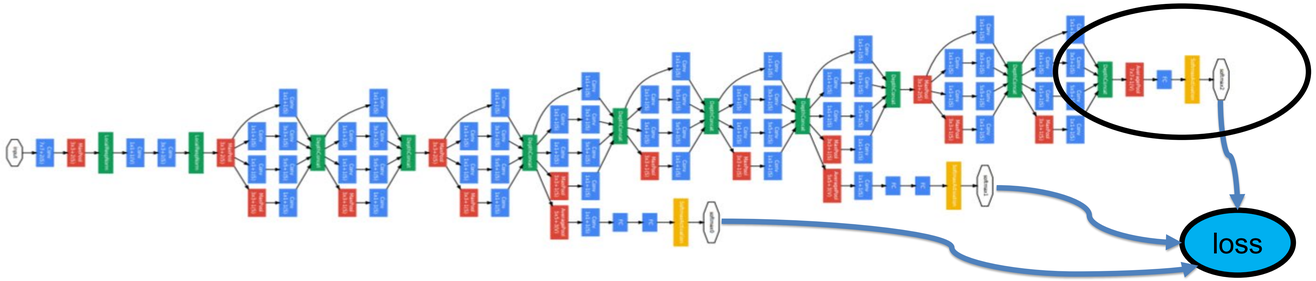


Stack several inception modules with different number of output channels

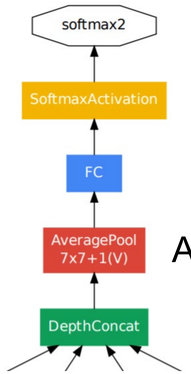


- Instead of having a single pass, having multiple passes
- Use 1x1 conv to reduce computation
- Concatenate along channels

Google Inception net, 2014



Output phase:



Averaging pooling for every 2D feature map, Cx7x7 → Cx1x1

- Avoid the expensive FC layer
- Avoid flatten layer

Resnet, 2015

- “Going deeper” improves performance
- But getting harder to train (mitigated by GoogLeNet, not an ideal solution)

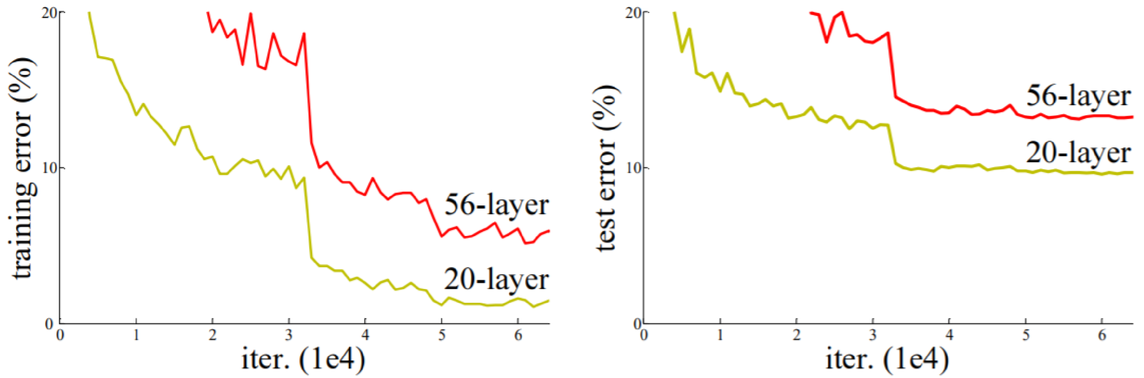
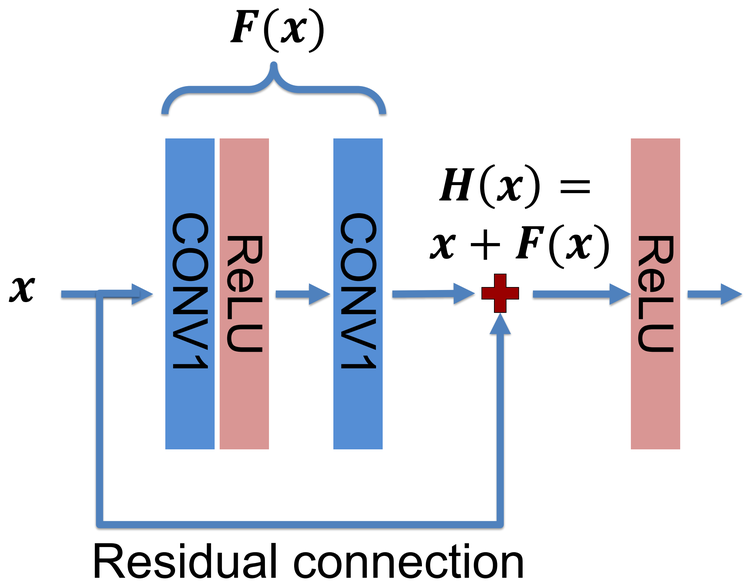


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Is this due to over-fitting?

Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385

Resnet : residual connection



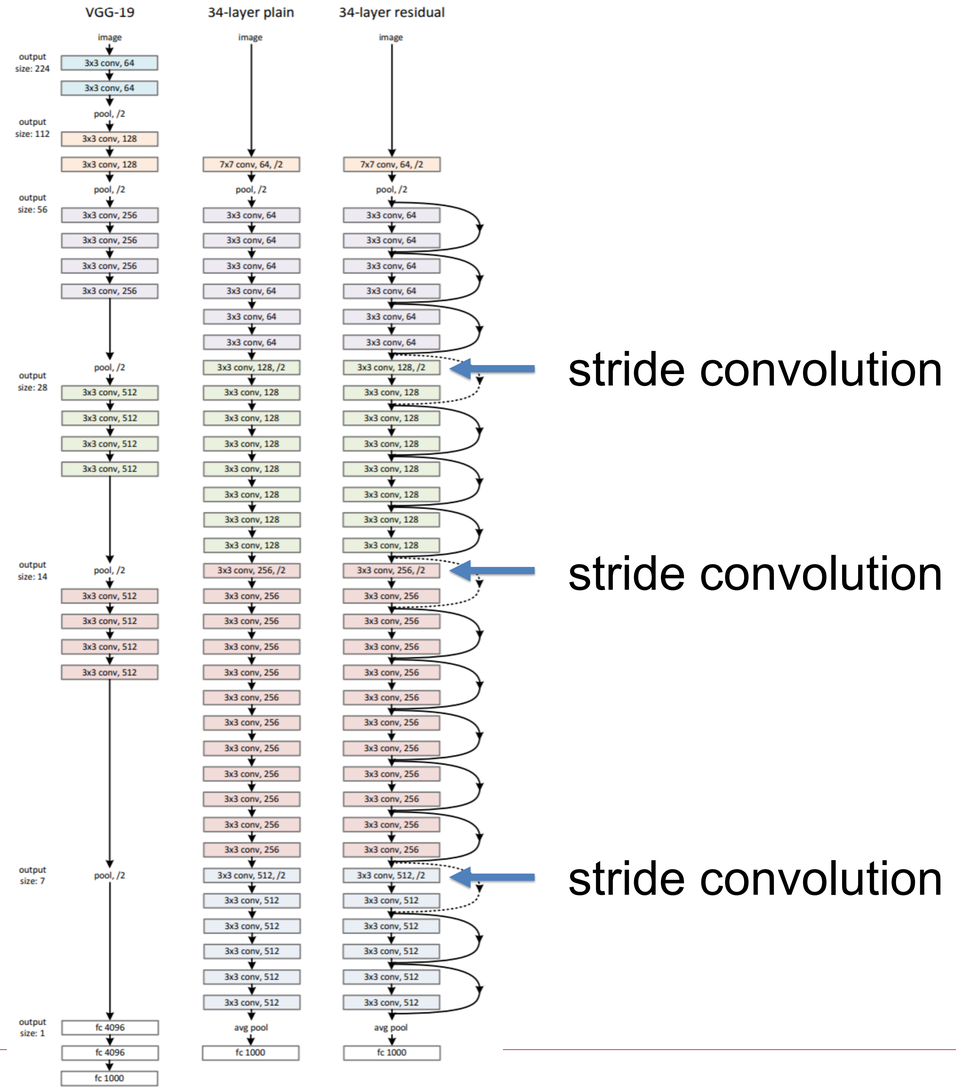
- Allow gradient flow to pass through
- Make training deeper network easier, if there are enough data
- No extra parameters added
- Constraint on network design

Training is performed to fit $F(x) = H(x) - x$
The residual function

Resnet : residual connection

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

- A lot of similarities to VGG
- 3x3 conv to keep image size
- Use stride convolution to reduce image size
- Increase number of channels while reducing image size
- Much deeper – ResNet 34/50/101/152 layers
- Avoid expensive FC – same as VGG



Resnet : residual connection

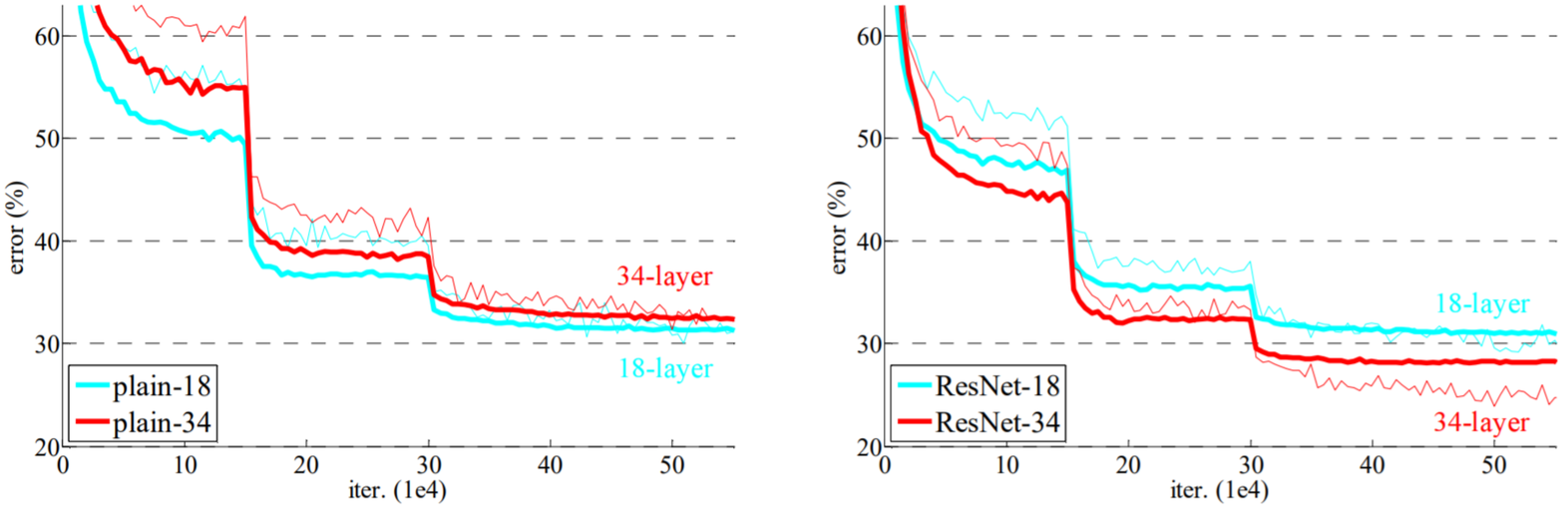


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Top 5 accu: 6.7% → 3.6%
Better than human error (5.1%)
https://arxiv.org/abs/1409.0575

Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385

Resnet : with batch normalization

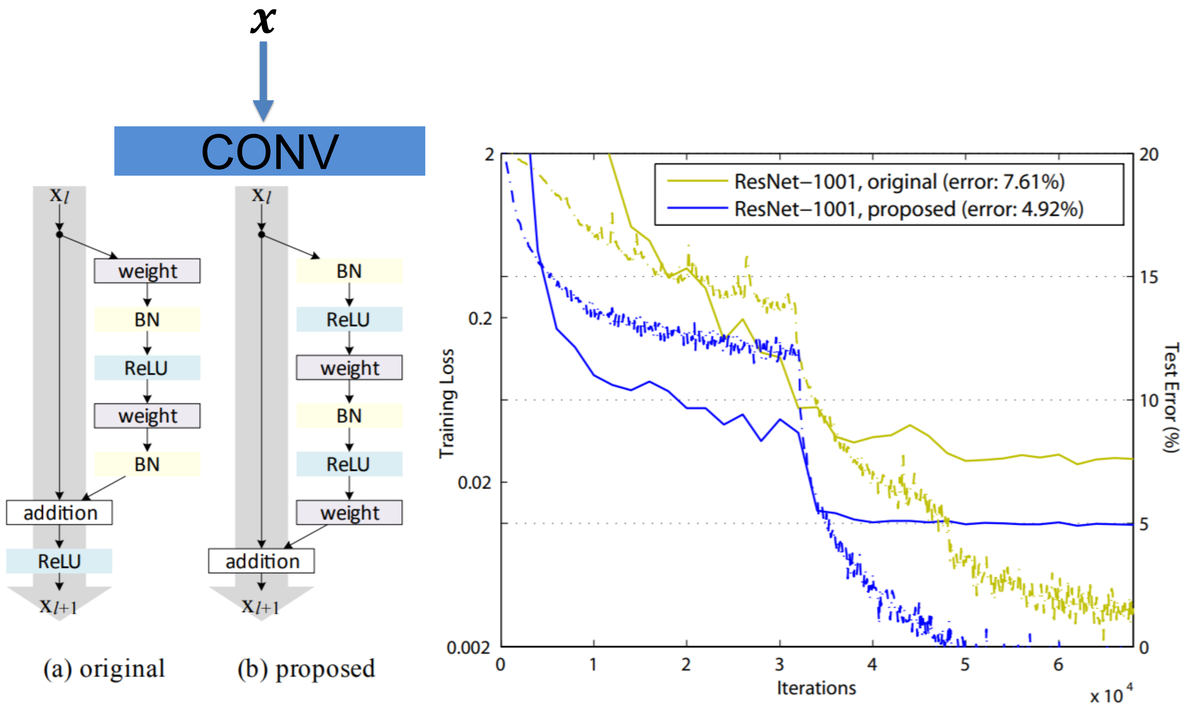


Figure 1. Left: (a) original Residual Unit in [1]; (b) proposed Residual Unit. The grey arrows indicate the easiest paths for the information to propagate, corresponding to the additive term "$\mathbf{x}_l$" in Eqn.(4) (forward propagation) and the additive term "1" in Eqn.(5) (backward propagation). **Right**: training curves on CIFAR-10 of **1001-layer** ResNets. Solid lines denote test error (y-axis on the right), and dashed lines denote training loss (y-axis on the left). The proposed unit makes ResNet-1001 easier to train.

- Use the (b) layout
- Network needs input CONV layers

Identity Mappings in Deep Residual Networks. https://arxiv.org/abs/1603.05027

ResNeXt 2016 : wider network with skip connection

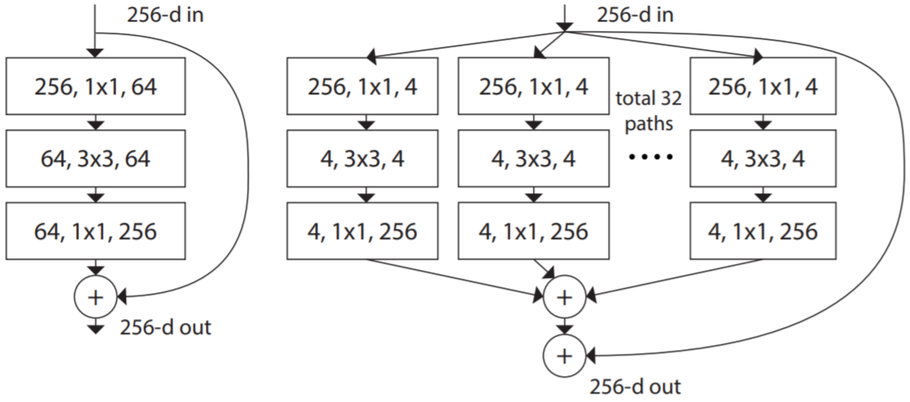


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

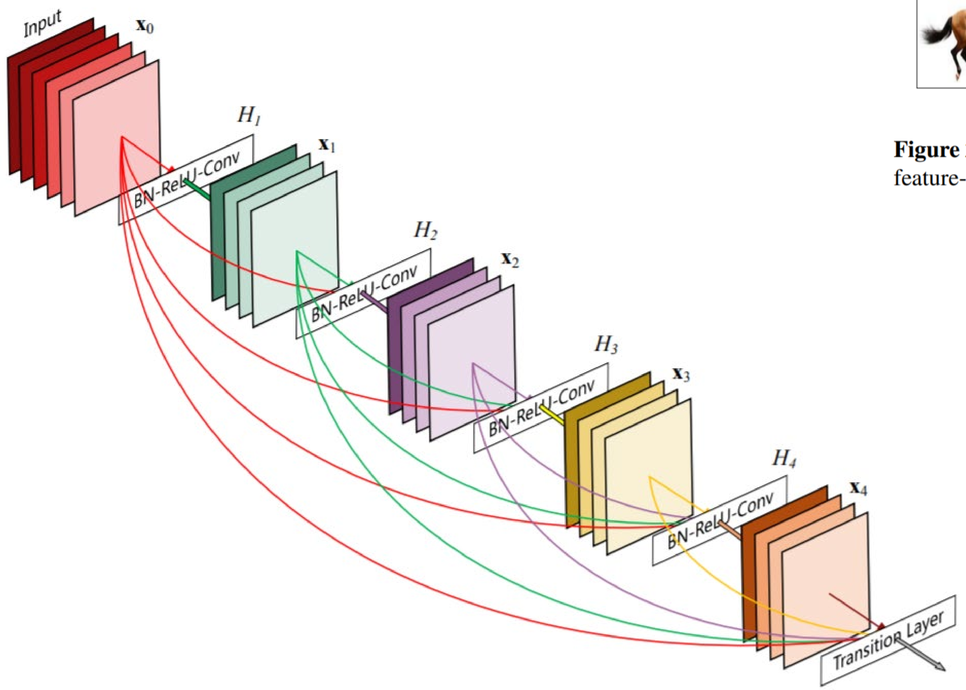
stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2 [1×1, 64 3×3, 64 1×1, 256] ×3	3×3 max pool, stride 2 [1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

- Combine the idea of inception module with residual connection
- Use 1x1 conv to reduce computation
- Multiple passes with homogenous architecture

Top 5 accu: 3.6% → 3.0%

Aggregated Residual Transformations for Deep Neural Networks. https://arxiv.org/abs/1611.05431

DenseNet : Densely Connected Convolutional Networks



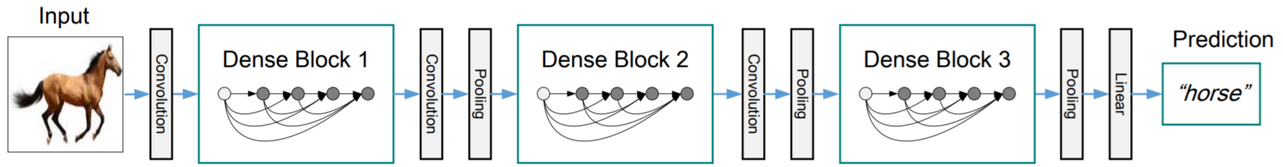
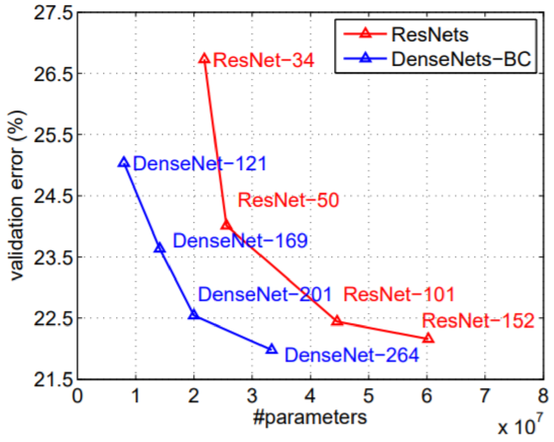


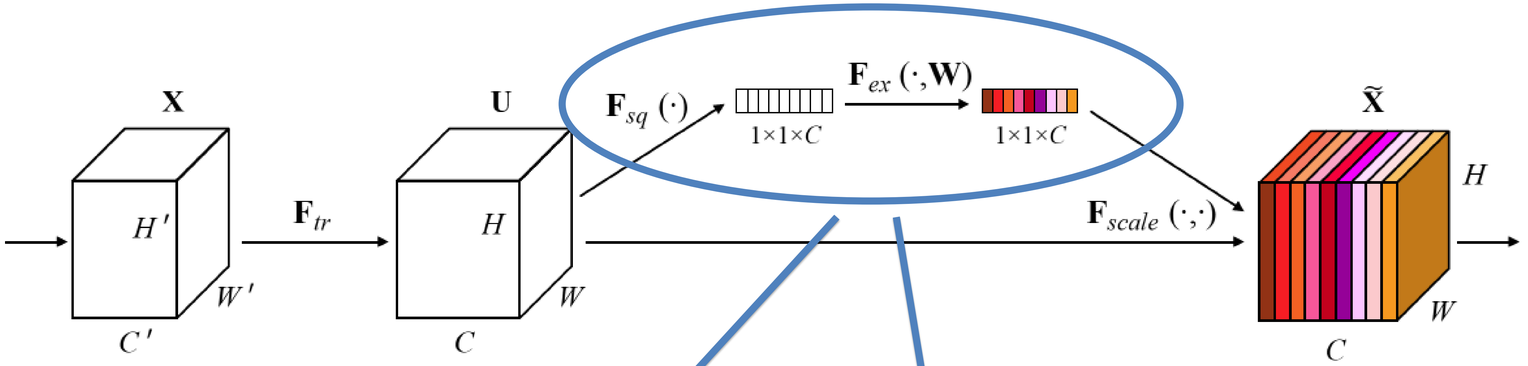
Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

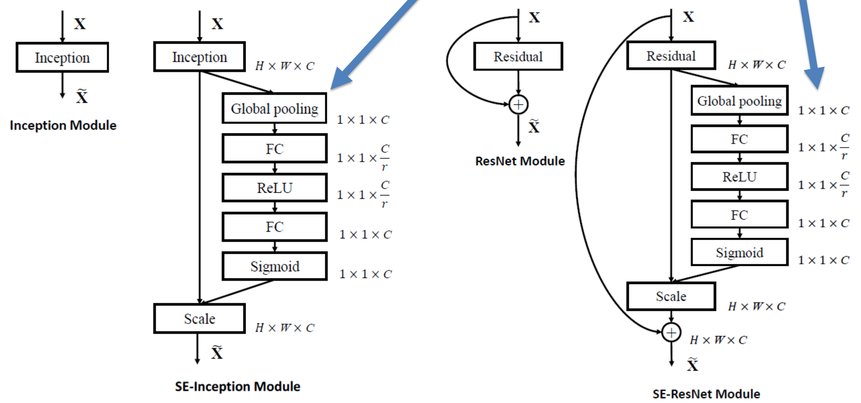


- Heavily using 3x3 and 1x1 conv
- Flexible dense block design
- Optionally, reduce number of feature maps by half between dense blocks (DenseNet-BC)

Densely Connected Convolutional Networks. https://arxiv.org/abs/1608.06993

SENet 2017 : **Squeeze-and-Excitation Network**





r can be 4, 8, 16, 32 …

Not sensitive to accuracy

- A SE module to rebalance channels

- First, global pooling along HxW
- Then squeeze with a FC layer
- Then expand it back to C channels
- After sigmoid, use the C ratio to scale every channel

- Relying on training to learn the best weights to rebalance channels
- With SE module, every channel contributes to all other channels → similar idea is popular in attention mechanism

Top 5 accu: 3.0% → 2.3%

Many other models in Pytorch model zoo

TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

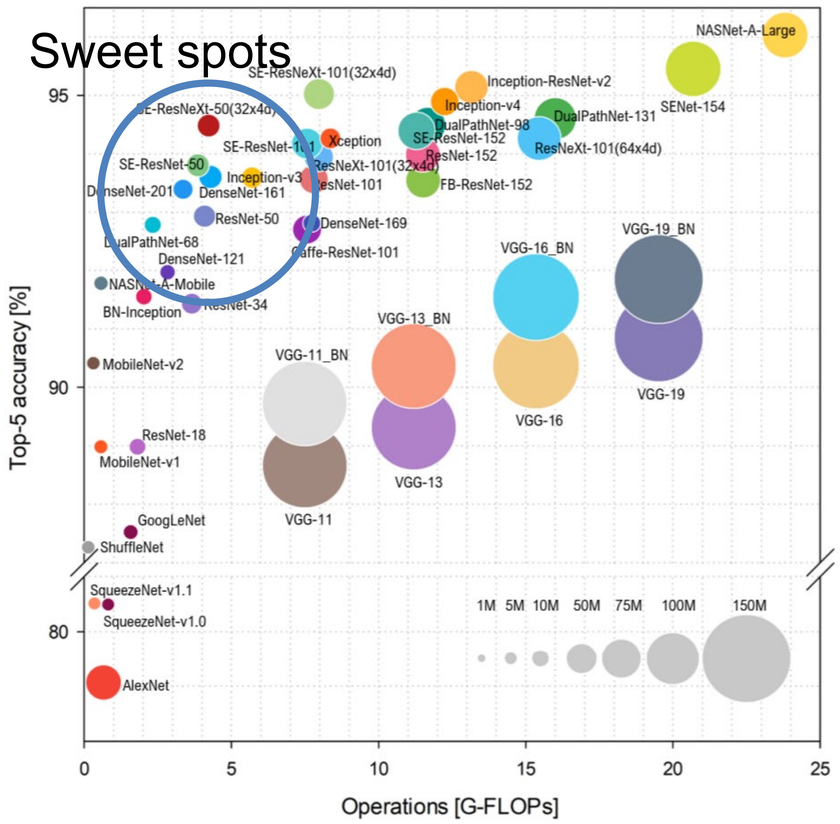
Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet

You can construct a model with random weights by calling its constructor:

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet_v2 = models.mobilenet_v2()
mobilenet_v3_large = models.mobilenet_v3_large()
mobilenet_v3_small = models.mobilenet_v3_small()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```



https://pytorch.org/vision/stable/models.html
https://pytorch.org/serve/model_zoo.html

Benchmark Analysis of Representative Deep Neural Network Architectures. https://arxiv.org/pdf/1810.00736.pdf

Another dimension: training speed

DAWNBench

An End-to-End Deep Learning Benchmark and Competition

- Image Classification (ImageNet)
- Image Classification (CIFAR10)
- Question Answering (SQuAD)

DAWNBench is a benchmark suite for end-to-end deep learning training and inference. Computation time and cost are critical resources in building deep models, yet many existing benchmarks focus solely on model accuracy. DAWNBench provides a reference set of common deep learning workloads for quantifying training time, training cost, inference latency, and inference cost across different optimization strategies, model architectures, software frameworks, clouds, and hardware.

Building on our experience with DAWNBench, we helped create **MLPerf** as an industry-standard for measuring machine learning system performance. Now that both the MLPerf **Training** and **Inference** benchmark suites have successfully launched, we **ended rolling submissions to DAWNBench on 3/27/2020** to consolidate benchmarking efforts.

The original **results before the April 20, 2018 deadline are archived** for reference. To learn more about key takeaways from DAWNBench, check out **our analysis of DAWNBench**.

Read the paper | Read the analysis | More information

Training time
Training cost
Inference latency
Inference cost

Image Classification on ImageNet

Training Time

All Submissions

Objective: Time taken to train an image classification model to a top-5 validation accuracy of 93% or greater on ImageNet.

Rank	Time to 93% Accuracy	Model	Hardware	Framework
1 Mar 2020	0:02:38	ResNet50-v1.5 *Apsara AI Acceleration(AIACC) team in Alibaba Cloud* source	16 ecs.gn6e-c12g1.24xlarge (AlibabaCloud)	AIACC-Training 1.3 + Tensorflow 2.1
2 May 2019	0:02:43	ResNet-50 *ModelArts Service of Huawei Cloud* source	16 nodes with InfiniBand (8*V100 with NVLink for each node)	Moxing v1.13.0 + TensorFlow v1.13.1
3 Dec 2018	0:09:22	ResNet-50 *ModelArts Service of Huawei Cloud* source	16 * 8 * Tesla-V100(ModelArts Service)	Huawei Optimized MXNet
4 Sep 2018	0:18:06	ResNet-50 *fast.ai/DIUx (Yaroslav Bulatov, Andrew Shaw, Jeremy Howard)* source	16 p3.16xlarge (AWS)	PyTorch 0.4.1

Sweet spots

Good balance with computation and accuracy

- ResNet, ResNext, DenseNet type network with Batch normalization
- Adam optimization, OneCycle scheduler

https://dawn.cs.stanford.edu/benchmark/#imagenet-train-time

Mobile net : depth-wise separable convolution

- Add the “soft constraint” to have depth separable convolution

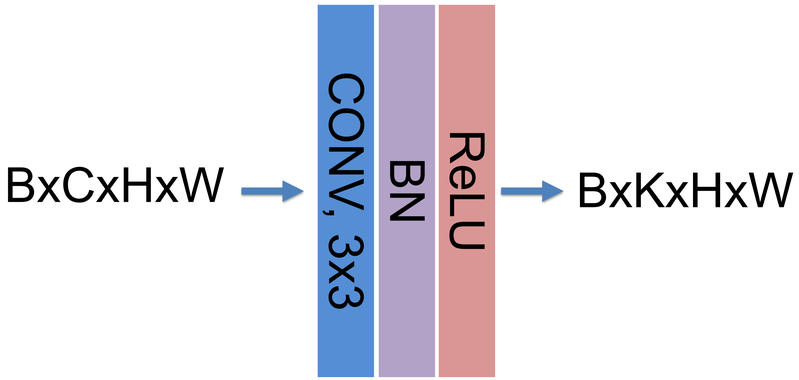
$$g(h,w,c) = gs(h,w) * gd(c)$$
$$x(h,w,c) * g(h,w,c) = [x(h,w,c) * gs(h,w)] * gd(c)$$

CONV 3x3
9xKxCxHxW multiplications

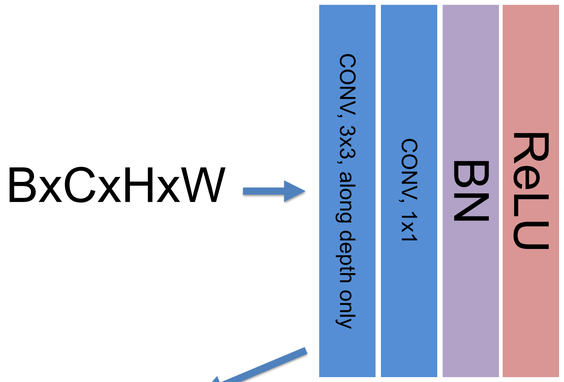
CONV 3x3, depth-wise
9xCxHxW multiplications

CONV 1x1
KxCxHxW multiplications

9KC >> (9+K)C



3x3 conv, stride 1, padding 1, K channels



3x3 conv, stride 1, padding 1

BxCxHxW → BxCxHxW

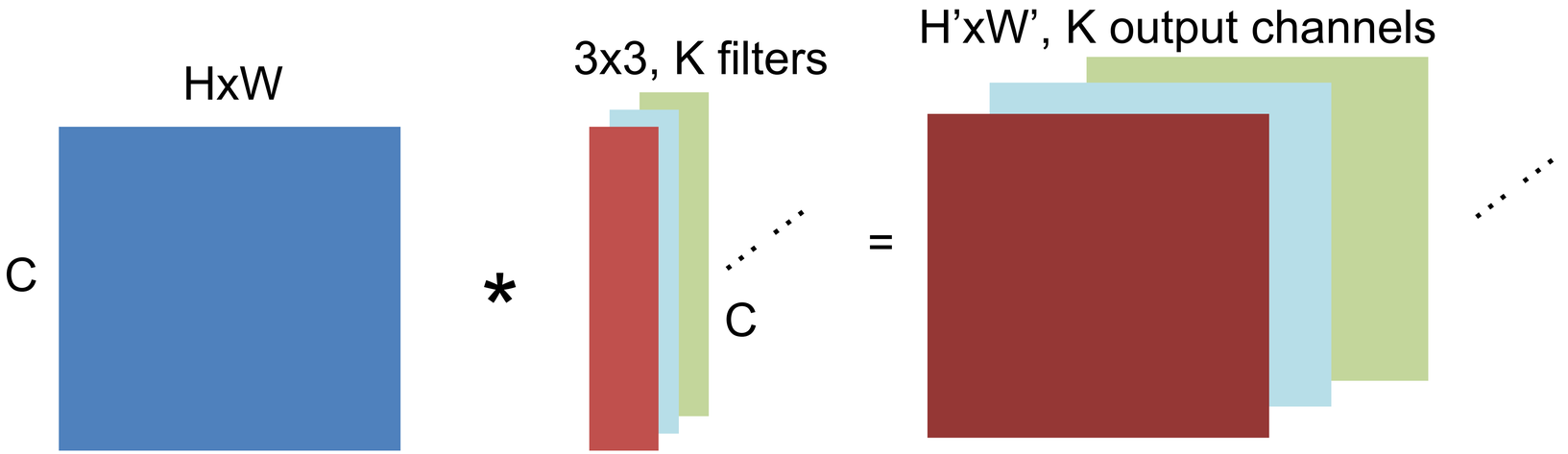
1x1 conv, stride 1, padding 0

BxCxHxW → BxKxHxW

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/abs/1704.04861

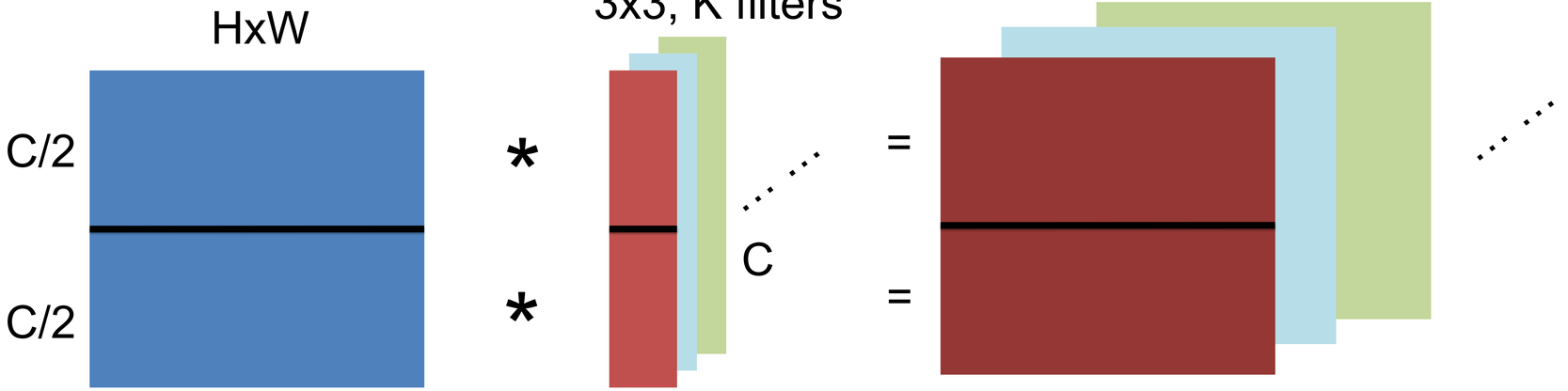
Group convolution and depth-wise convolution

Normal convolution



- Perform convolution for every output channel
- CxHxW→ KxHxW, kenrel size k_xxk_yxCxK

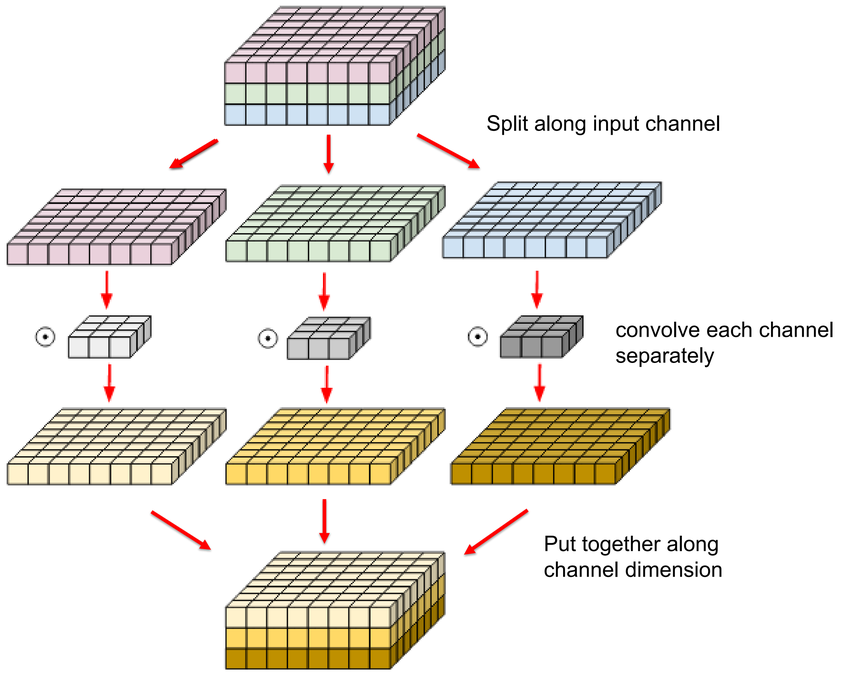
Group convolution, G=2



- Perform convolution for every group
- Split along the input channel dimension

Deep-wise convolution

- Depth-wise convolution is implemented with group convolution with G=C
- So 2D convolution for every input channel independently



CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

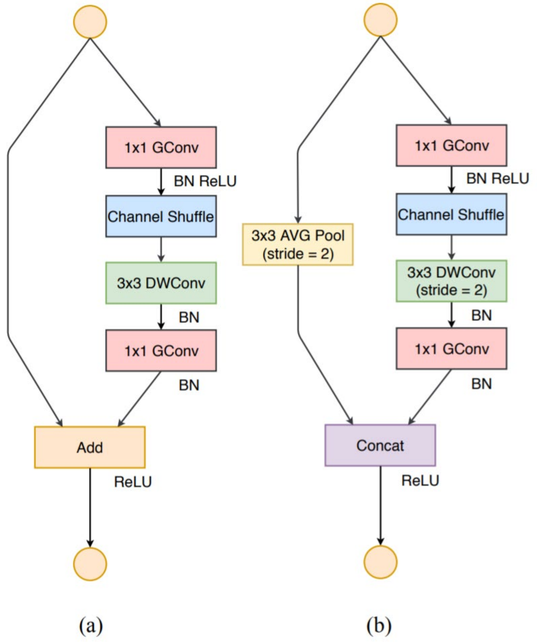
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Shuffle Net : further reduce computation for 1x1 conv

1x1 CONV, CxKxHxW multiplications

1x1 group CONV, CxKxHxW /G multiplications



Shuffle net module

Since every group is processed independently, channel shuffle was added to allow intra-group interaction

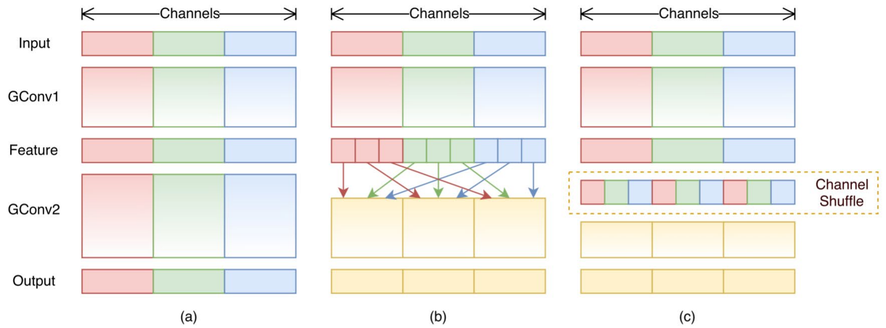


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

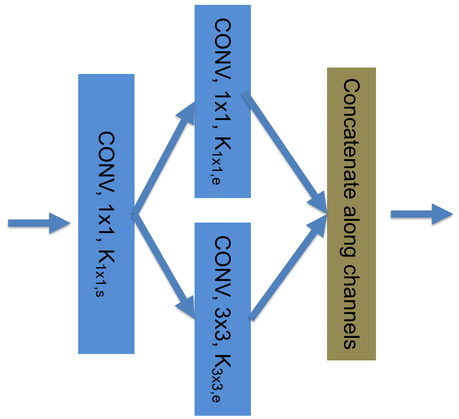
Complexity (MFLOPs)	VGG-like	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	50.7	37.3	33.6	33.3	**32.4** ($1\times$, $g = 8$)
38	-	48.8	45.1	46.0	**41.6** ($0.5\times$, $g = 4$)
13	-	63.7	57.1	65.2	**52.7** ($0.25\times$, $g = 8$)

Higher accuracy when model complexity is limited

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. https://arxiv.org/abs/1707.01083

Squeeze net : AlexNet accuracy with 50x less #paras

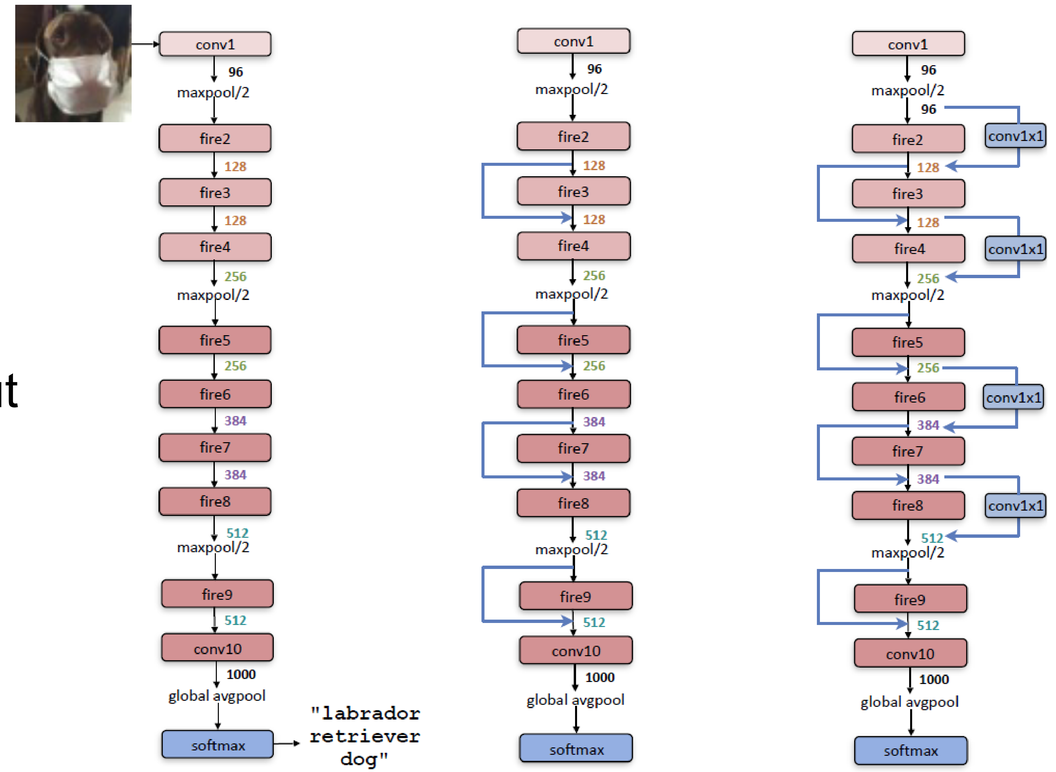
- Reduce memory size and computation while maintaining the performance
- Use 1x1 conv
- Introduce “fire” module
- Reduce the image size at a slower pace for accuracy



Fire module

Hyperparameters: number of CONV output channels

$K_{1x1,s}$, $K_{1x1,e}$, $K_{3x3,s}$

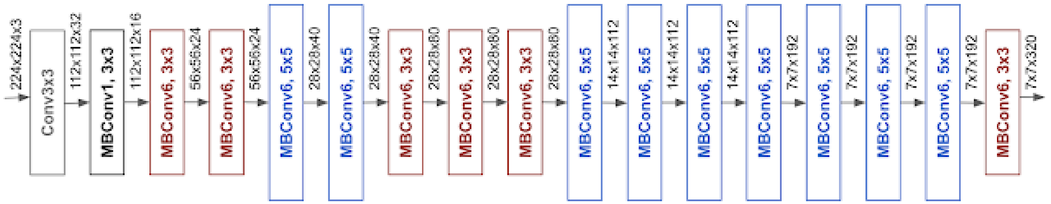


CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

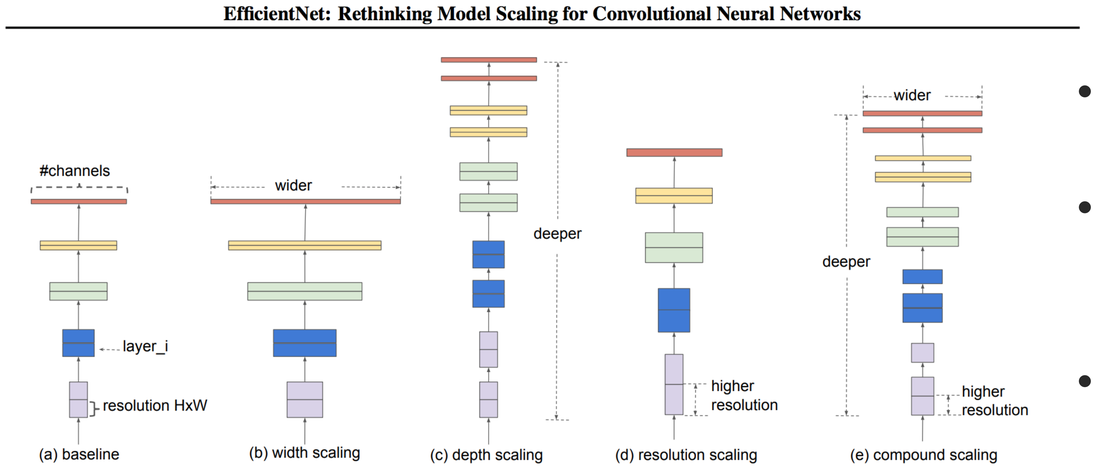
SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. https://arxiv.org/abs/1602.07360

Efficient Net

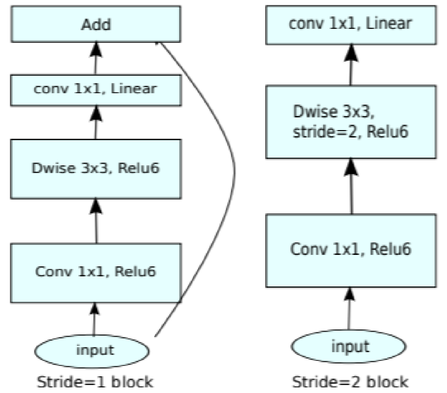
- Systematic way to expand network – depth, width, image resolution
- Starting from a base model, search optimal expansion to grow all three aspects together
- Use neural architecture search to find a good expansion



MBConv is from MobileNet V2
https://arxiv.org/pdf/1801.04381.pdf



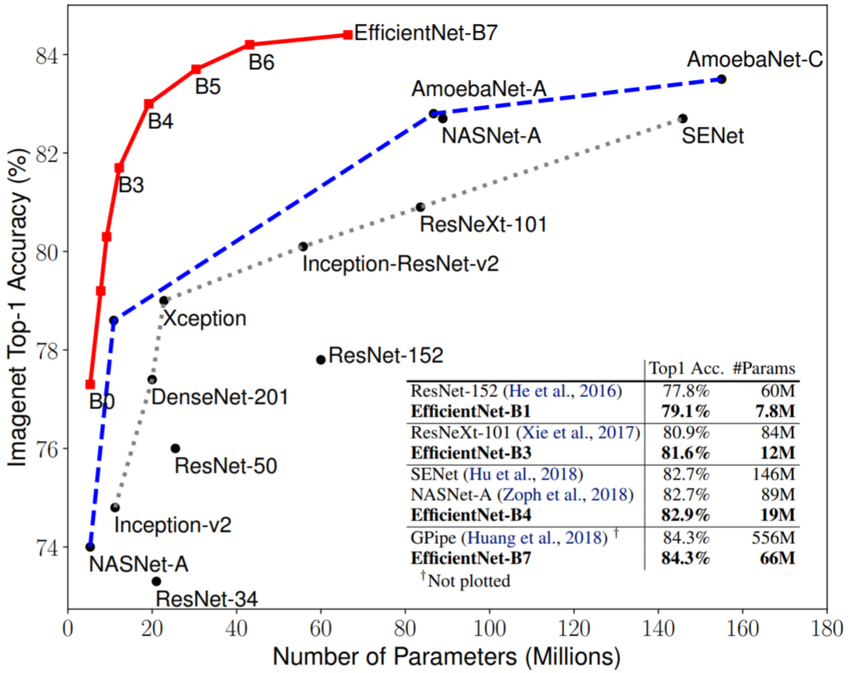
- **depth** :the number of layers in a network
- **width** is the number of neurons in a layer or the number of filters in a convolutional layer
- **resolution** is the height and width of the input tensor



(d) Mobilenet V2

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://arxiv.org/abs/1905.11946

Efficient Net



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

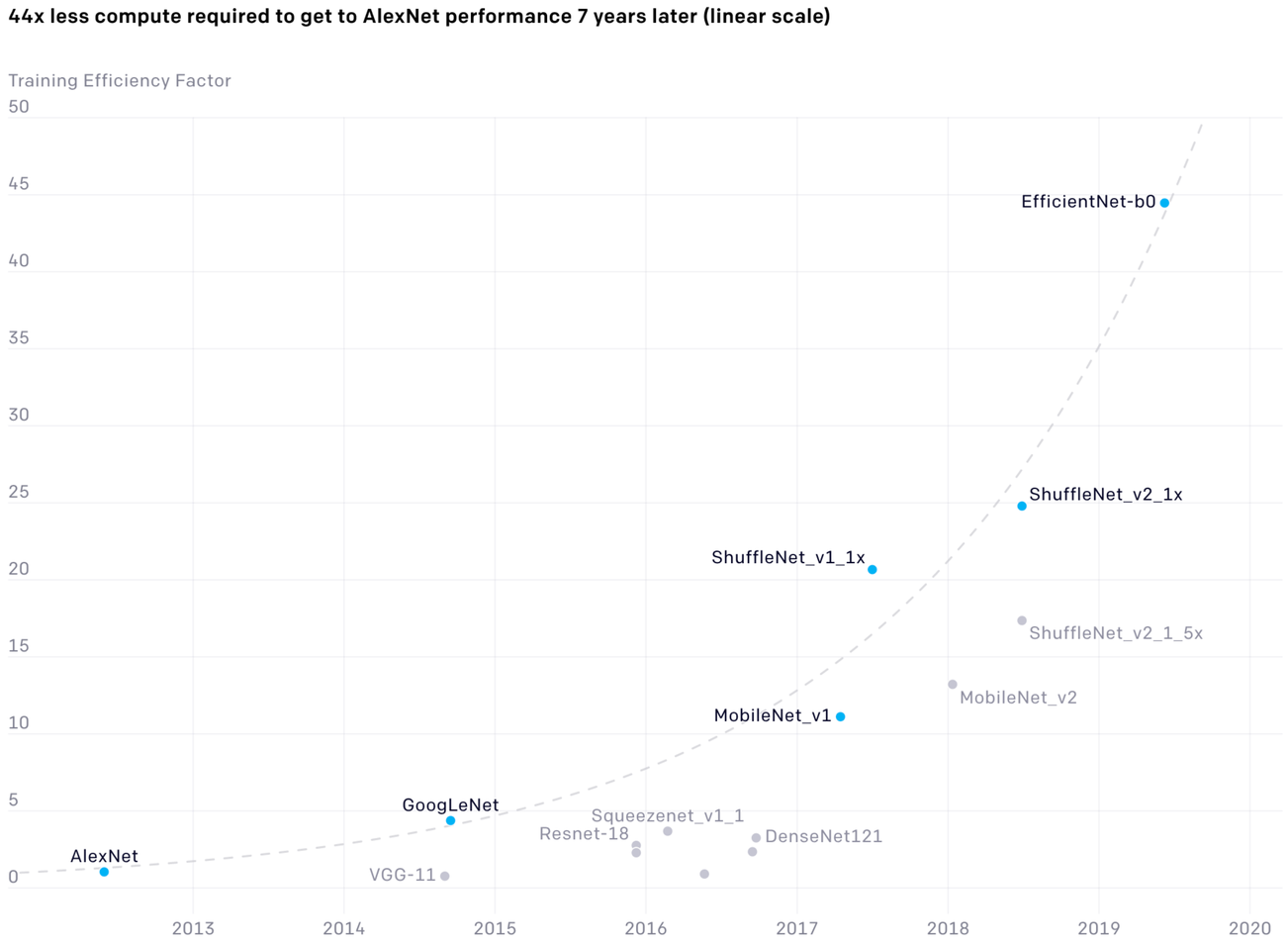
Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B3	**81.7%**	**95.6%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**83.0%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.7%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.2%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

- Scaling up a single aspect quickly saturates the accuracy
- Jointly scale up all three aspects achieved top accuracy with reduced model complexity
- A systematic approach to design CNN!

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://arxiv.org/abs/1905.11946

Training efficiency improves



- The number of floating-point operations required to train a classifier to get the AlexNet-level performance on ImageNet
- 2x reduction every 16 months and in total of 44x reduction of computation, since 2012
- Released on May, 2020

https://openai.com/blog/ai-and-efficiency/

Measuring the Algorithmic Efficiency of Neural Networks. https://arxiv.org/abs/2005.04305

Task definition

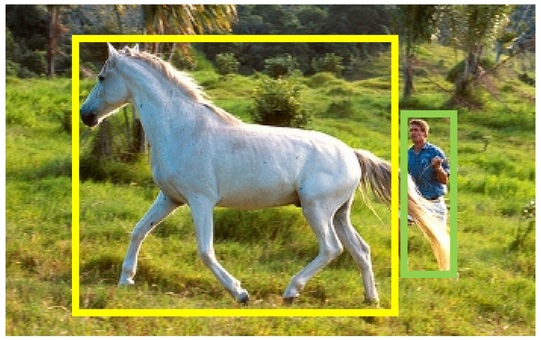
horse

Classification: given an image, output the object class

horse + bounding box (x, y, H, W)

Localization: given an image, output the object class and its bounding box

One object

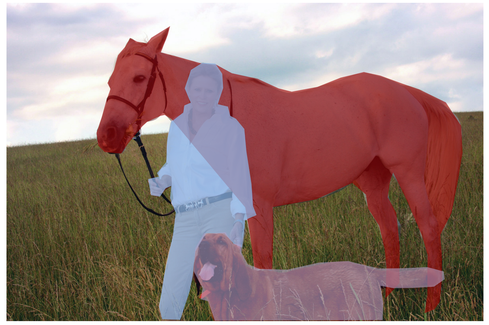


horse + its BB

human + his BB

… …

Detection: given an image, detect all objects. For each object, output its class and bounding box



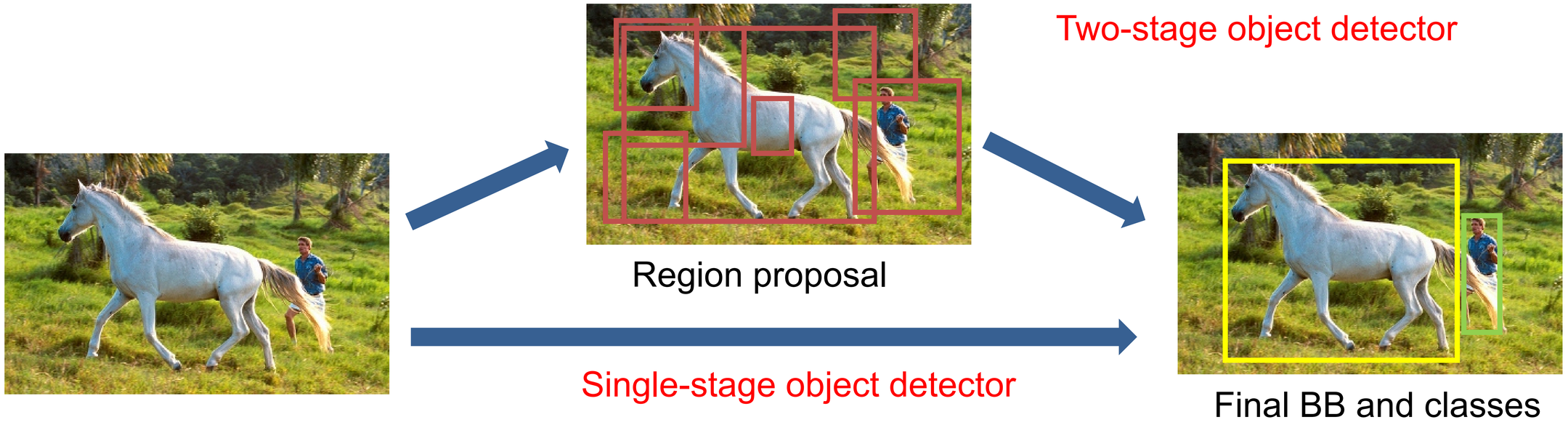
horse + its mask
human + his mask
dog + its mask

Instance segmentation: given an image, detect all objects. For each object, output its class and bounding box

multiple objects

Object detection

- Detect multiple objects
- Need to be fast and accurate
- Foundation of many consumer products



YOLO: You Only Look Once

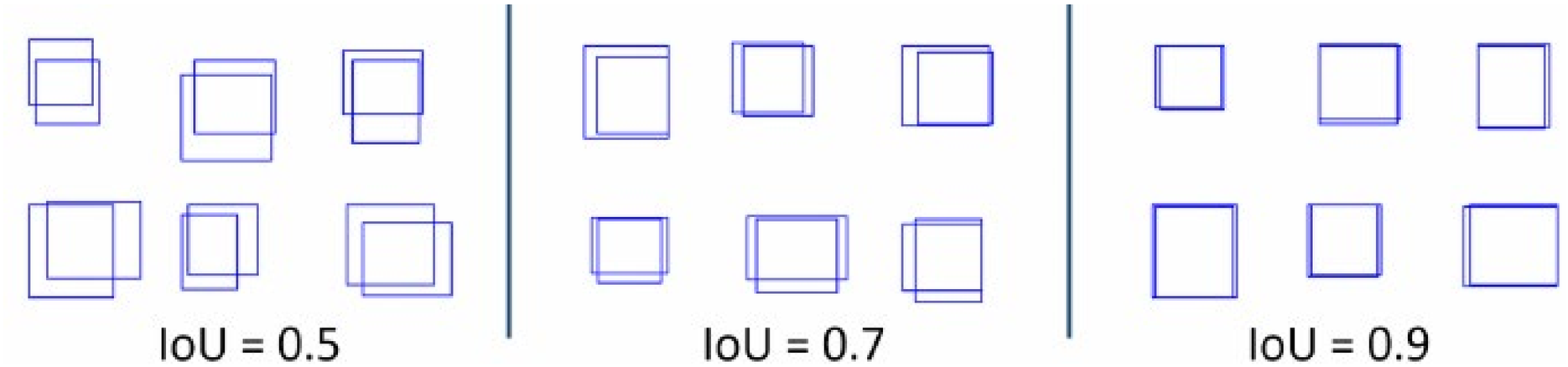
- Single stage detector
- Very fast and accurate, enable real-time detection
- Envolve from v1 to v5

Figure …nages
with YO …sizes
the inpu… …l net-
work or… …ns by
the model's confidence.

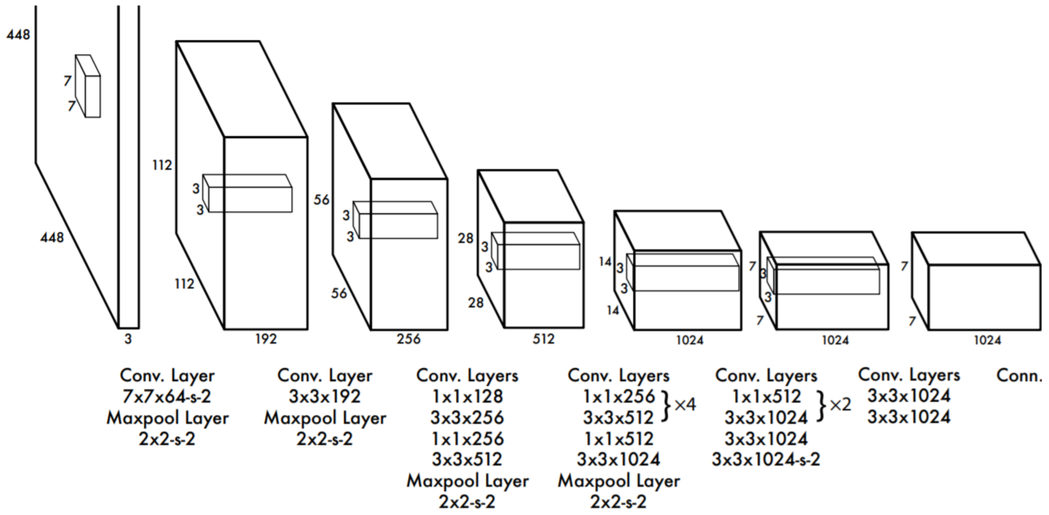
https://pjreddie.com/publications/ https://youtu.be/MPU2Histivl?t=27

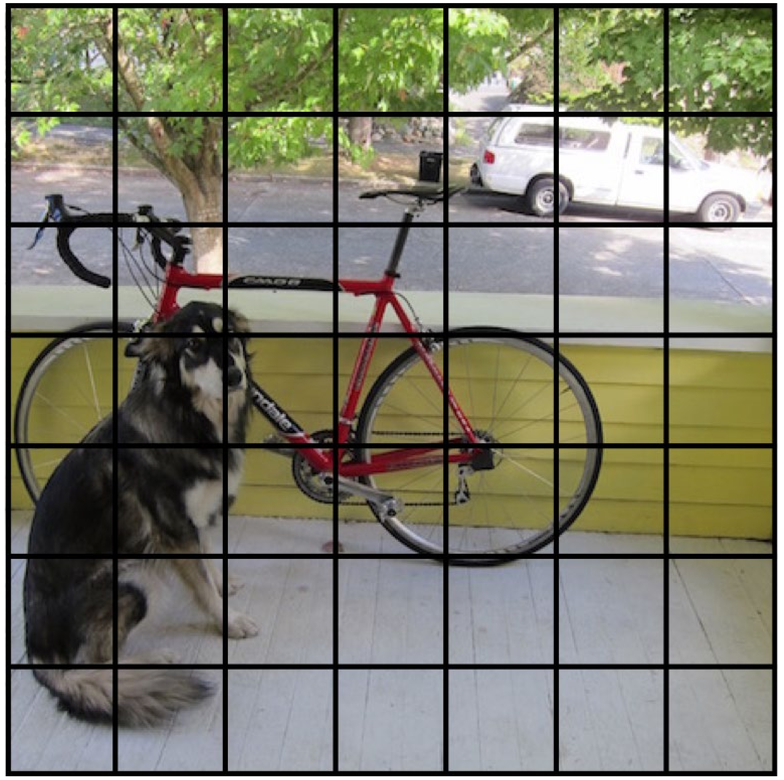
IoU: Intersection over union

$$\text{IoU} = \frac{A \cap B}{A \cup B}$$



How the YOLO works

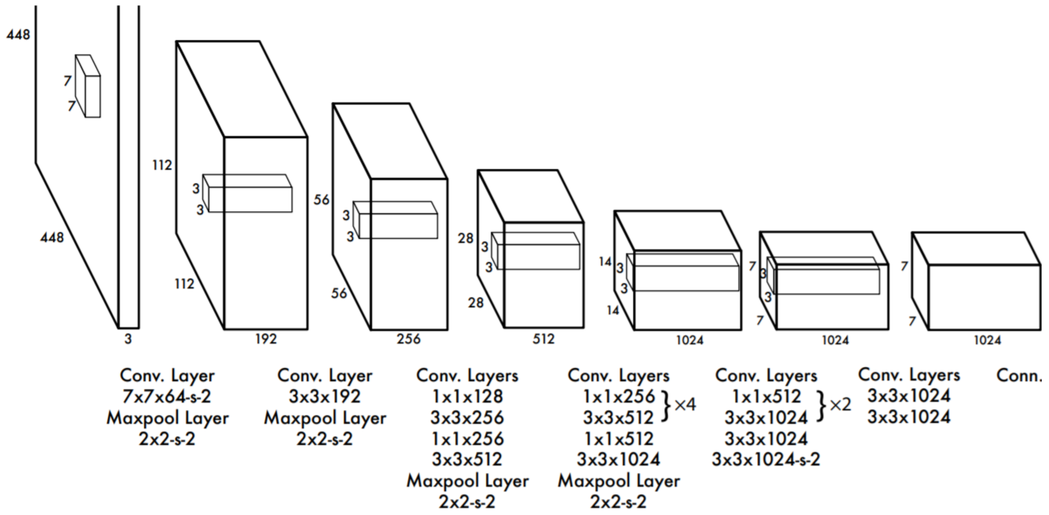


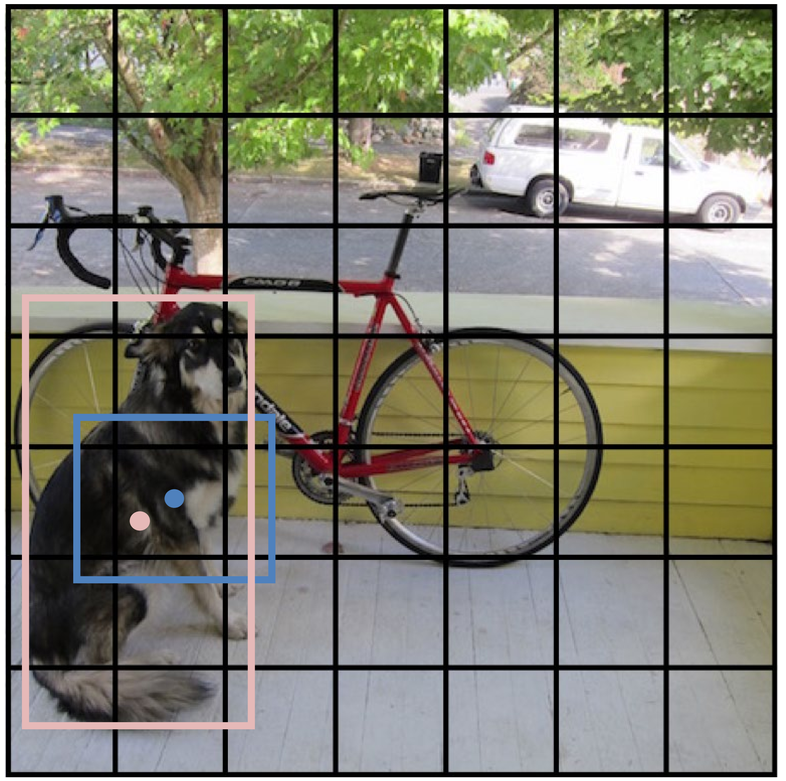


- Pick a CNN backbone network
- On the final conv layer, image is downsampled to a 7x7 feature maps
- Every feature has a large receptive field in the origin image

You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

How the YOLO works





Key idea: every cell predicts *B* bounding boxes with object classes

Every cell predicts : $p_{object}, b_x, b_y, b_w, b_h, [p_1, p_2, \dots, p_C]$

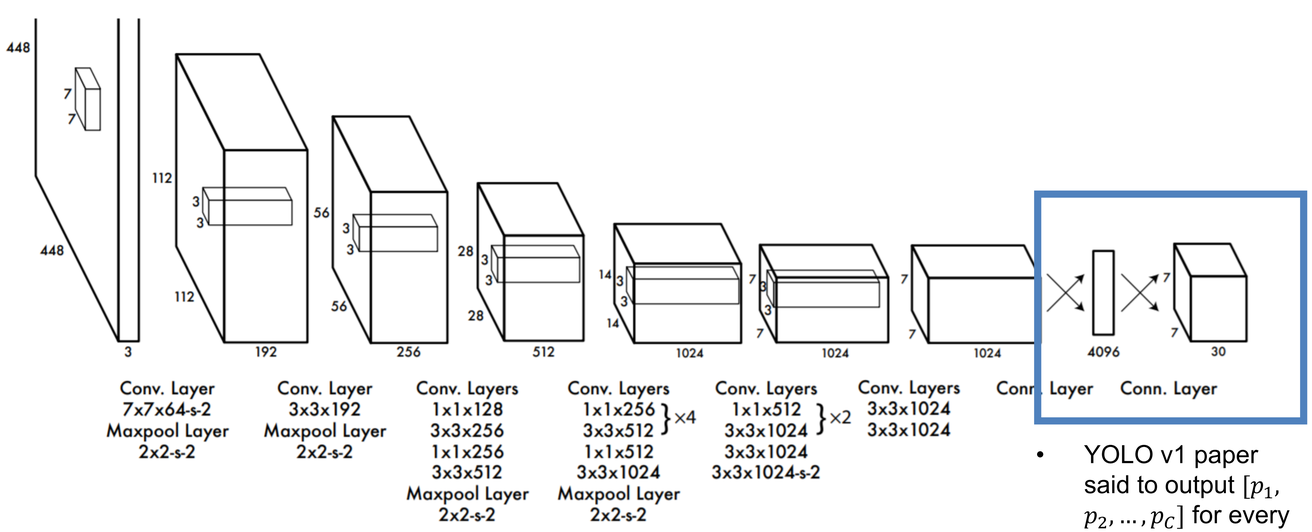
Whether this cell has an object

Bounding box center, width and height

class scores for all classes

You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

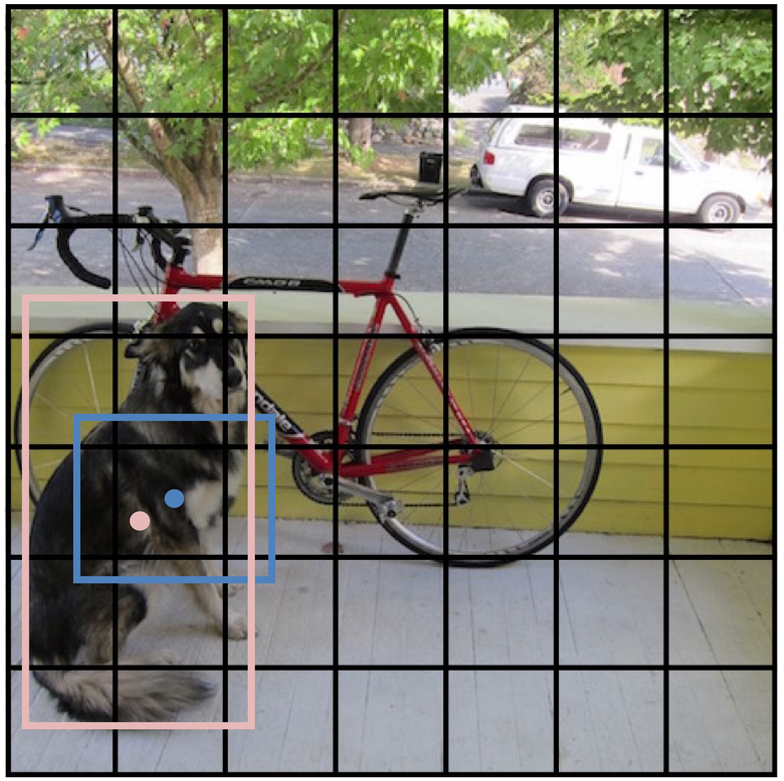
How the YOLO works



- YOLO v1 paper said to output $[p_1, p_2, \ldots, p_C]$ for every cell
- Since YOLO v2, model outputs class probabilities for every bounding box

For 7x7 cell, B=2, C=20,

YOLO outputs a tensor 7x7x(2x(5+20))=7x7x50



You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

How the YOLO works

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

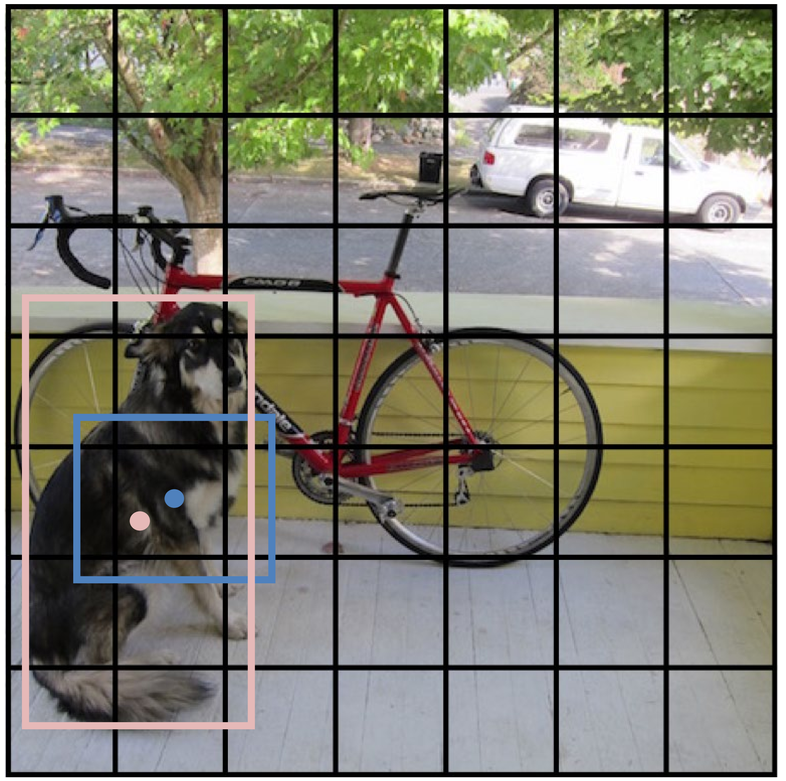
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

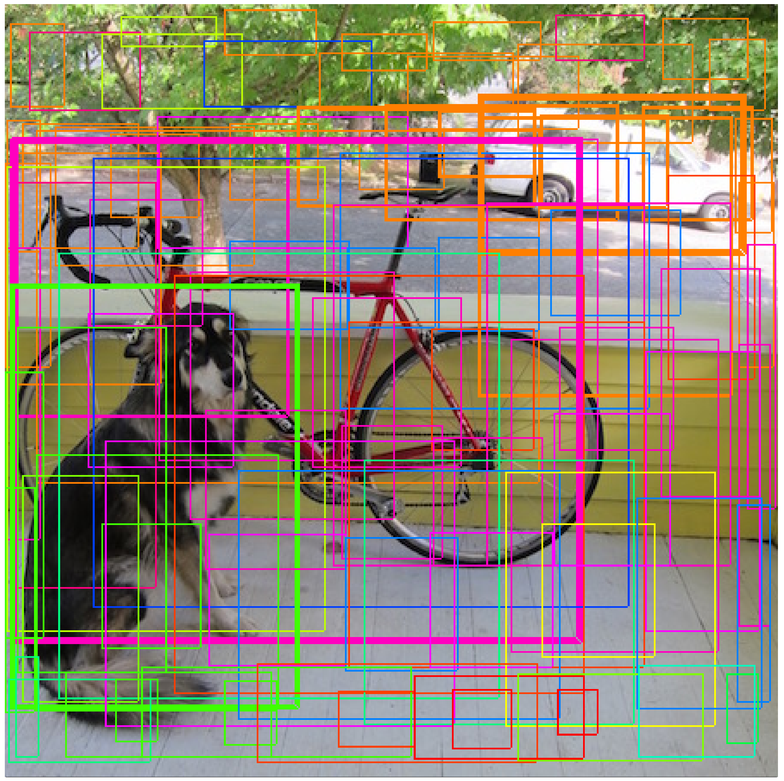


- *C* is the confidence, computed with IoU
- At training time, among the B bounding boxes for a cell, one with highest IoU with ground-truth is made "responsible"

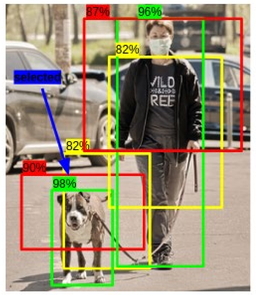
You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

How the YOLO works

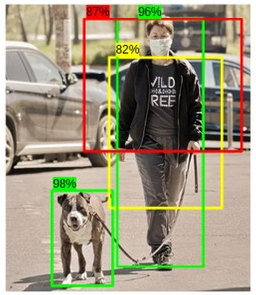
- With one forward pass, a lot of bounding boxes are estimated, together with p_{object} and class probability
- Compute confidence as $p_{object} \times IoU\left(BB_{pred,}\, BB_{truth}\right)$
- Threshold with confidence >0.3 and NMS



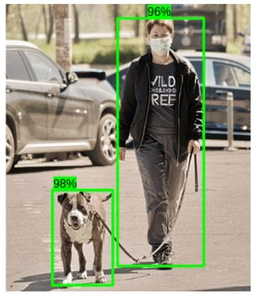
non-max suppression



Step 1: Selecting Bounding box with highest score



Step 3: Delete Bounding box with high overlap



Step 5: Final Output

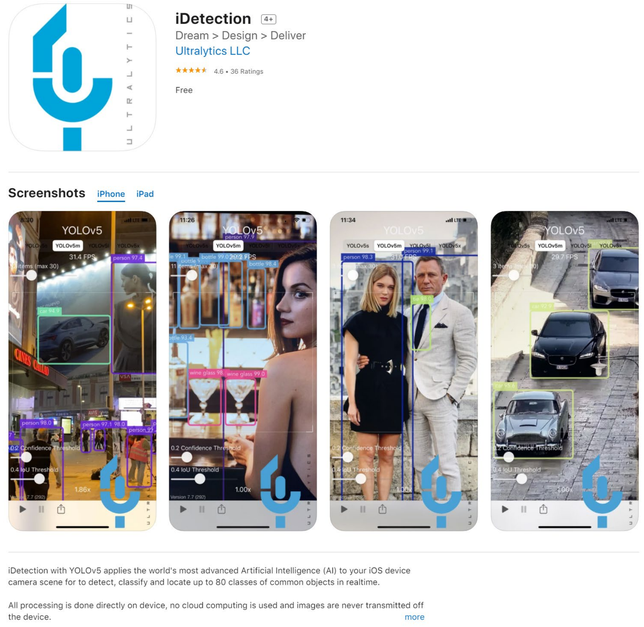
Step 1: Select the box with highest objectiveness score
Step 2: Then, compare the overlap (intersection over union) of this box with other boxes
Step 3: Remove the bounding boxes with overlap (intersection over union) >50%
Step 4: Then, move to the next highest objectiveness score
Step 5: Finally, repeat steps 2-4

https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/

You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640

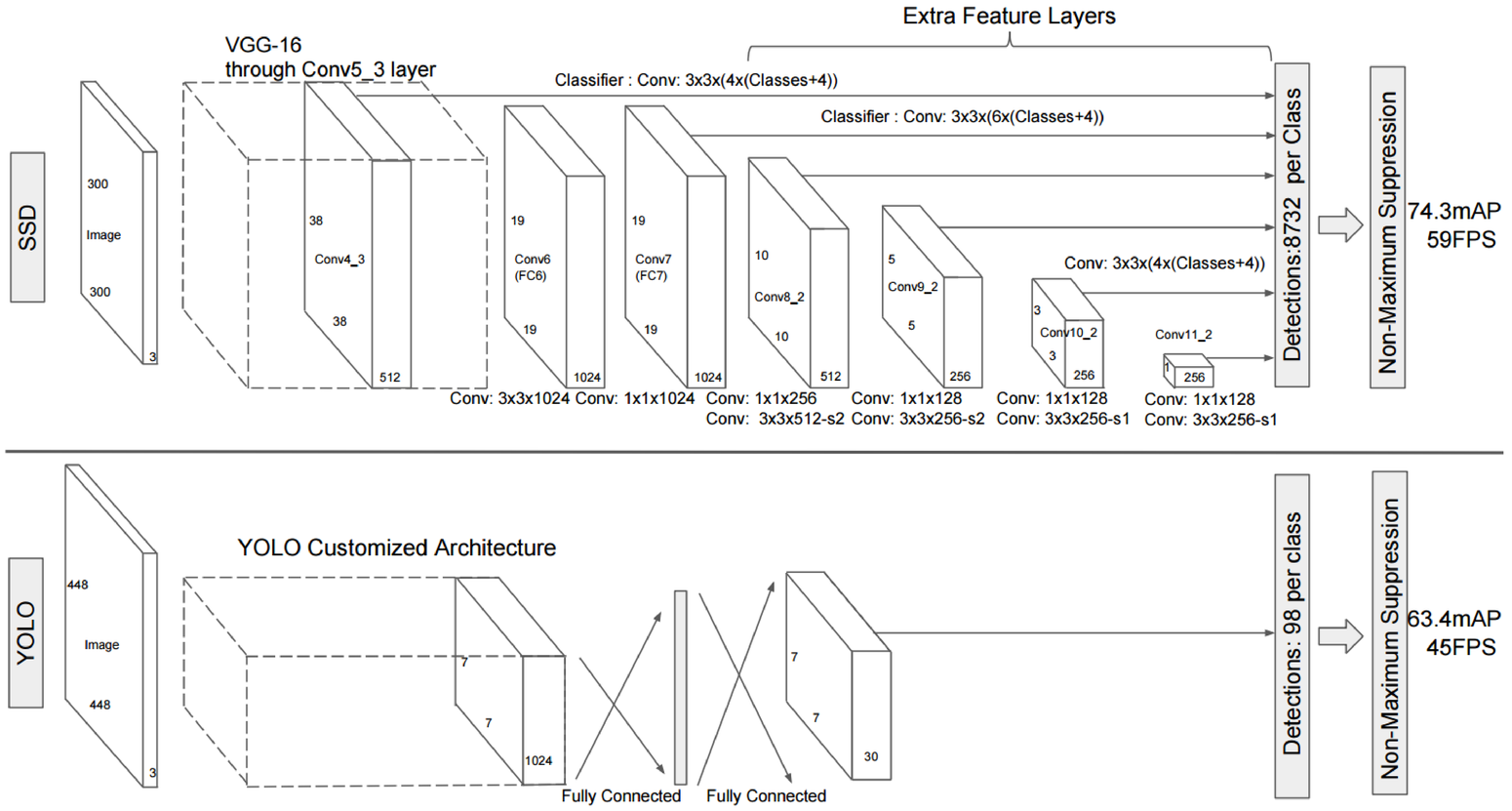
YOLO v1 to v5

- YOLO v1 : You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640
- YOLO v2 : YOLO9000: Better, Faster, Stronger. https://arxiv.org/abs/1612.08242
- YOLO v3: YOLOv3: An Incremental Improvement. https://arxiv.org/abs/1804.02767
- YOLO v4: Optimal Speed and Accuracy of Object Detection. https://arxiv.org/abs/2004.10934v1
- YOLO v5: https://github.com/ultralytics/yolov5



SSD: single shot detector

- Another one-stage detector
- Use multiple conv feature maps

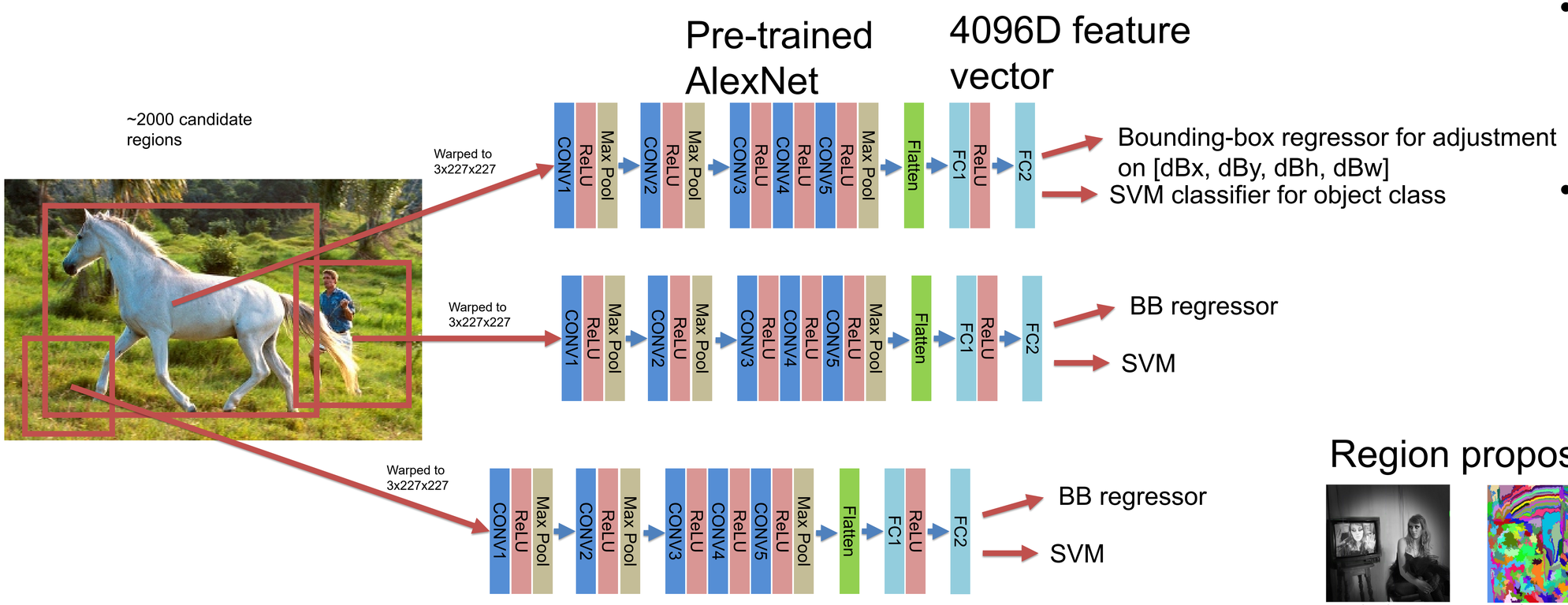


https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/single-shot-detectors/ssd

SSD: Single Shot MultiBox Detector. https://arxiv.org/abs/1512.02325

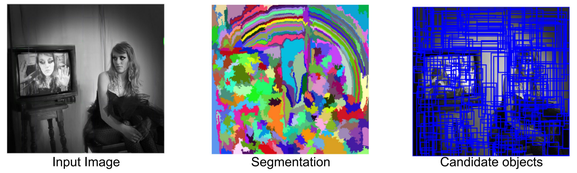
R-CNN : two stage detector

- Find regions which may contain objects → region proposal
- For every candidate region, run a CNN to predict object class and correction for the bounding box



- Very slow detection - ~50s per image
- First time show the detection can be done for ~1000 classes of objects

Region proposal algorithm

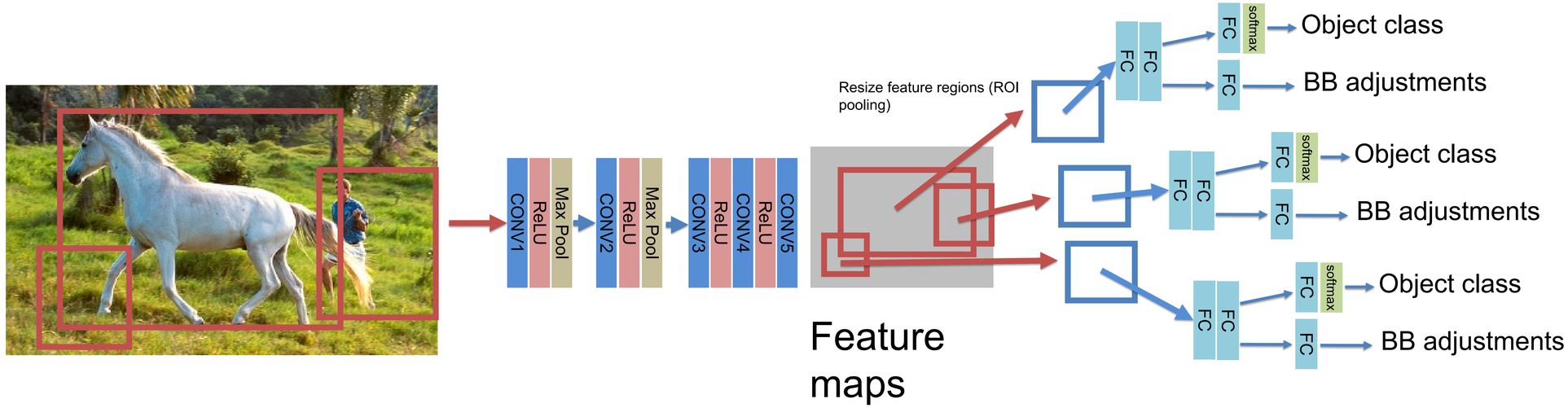


http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ssearch_schuyler.pdf

Rich feature hierarchies for accurate object detection and semantic segmentation. https://arxiv.org/abs/1311.2524

Fast R-CNN : region proposal on feature maps

- Work on the CNN feature maps
- Only need one pass for backbone network
- Much faster than R-CNN



Fast R-CNN. https://arxiv.org/abs/1504.08083

Faster R-CNN : add region proposal network

- Add region proposal network
- End-to-end training

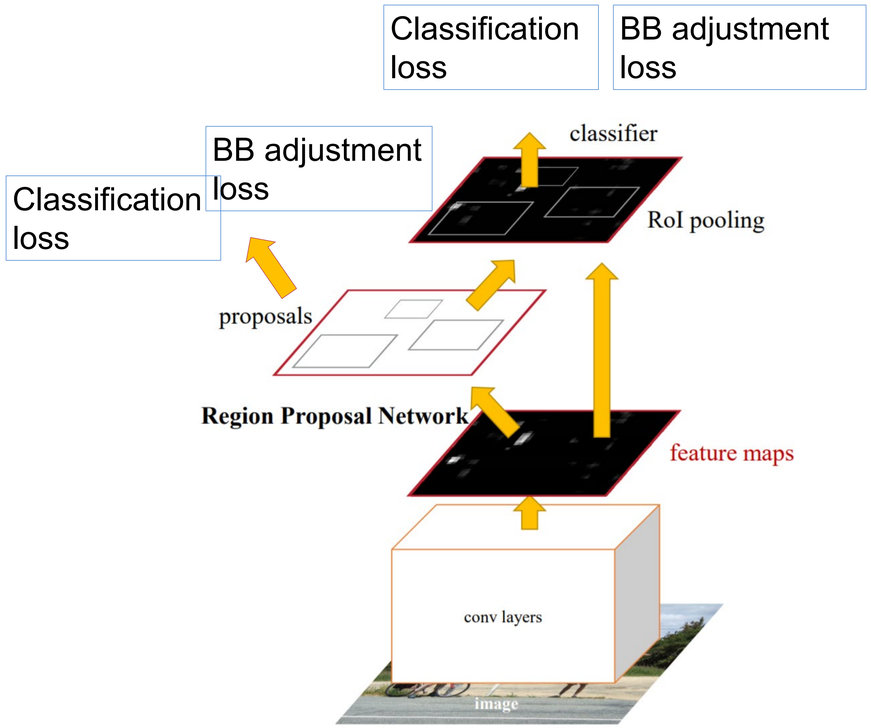
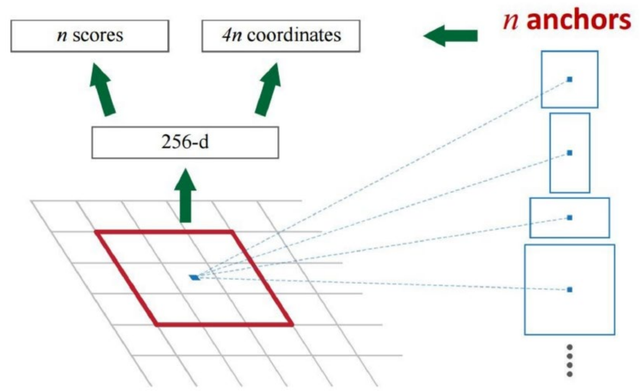


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

RPN: region proposal network



- For every location in feature maps
- Consider k anchor boxes at each location
- For every anchor box, predict p_{object}, db_x, db_y, $\mathrm{d}b_w$, db_h → *similar to YOLO, but for region proposal*
- Jointly optimize four losses

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://arxiv.org/abs/1506.01497

Mask R-CNN : add segmentation layer

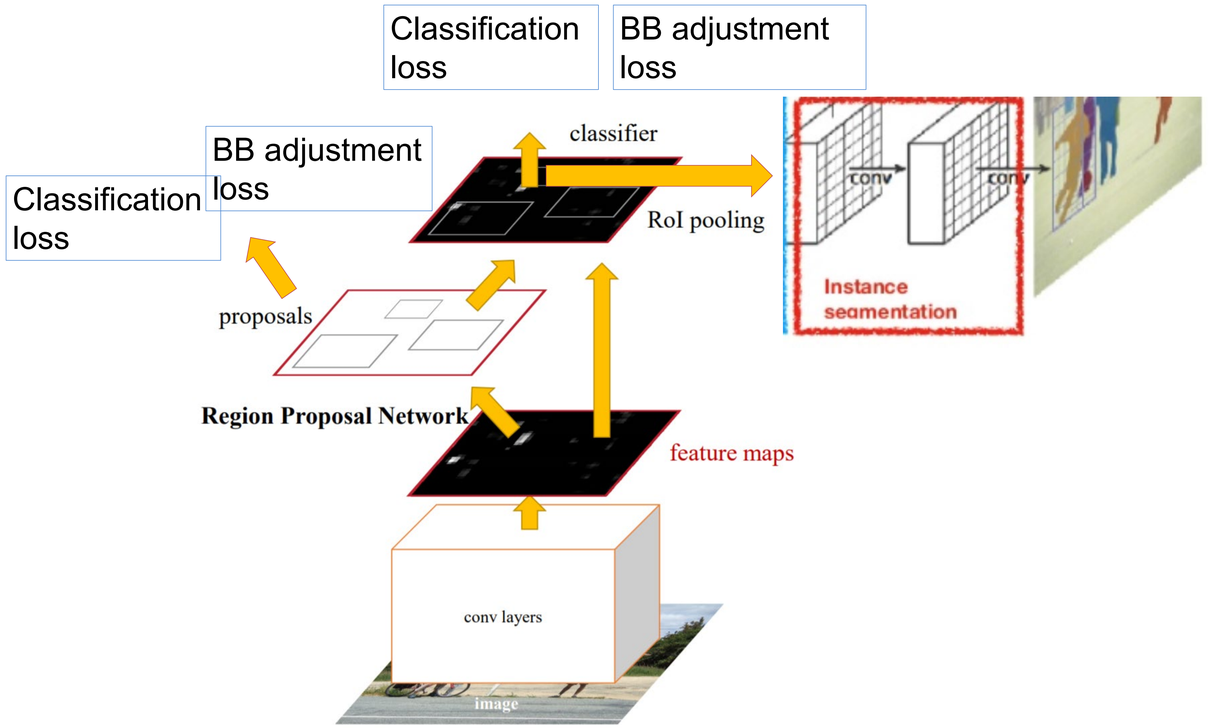
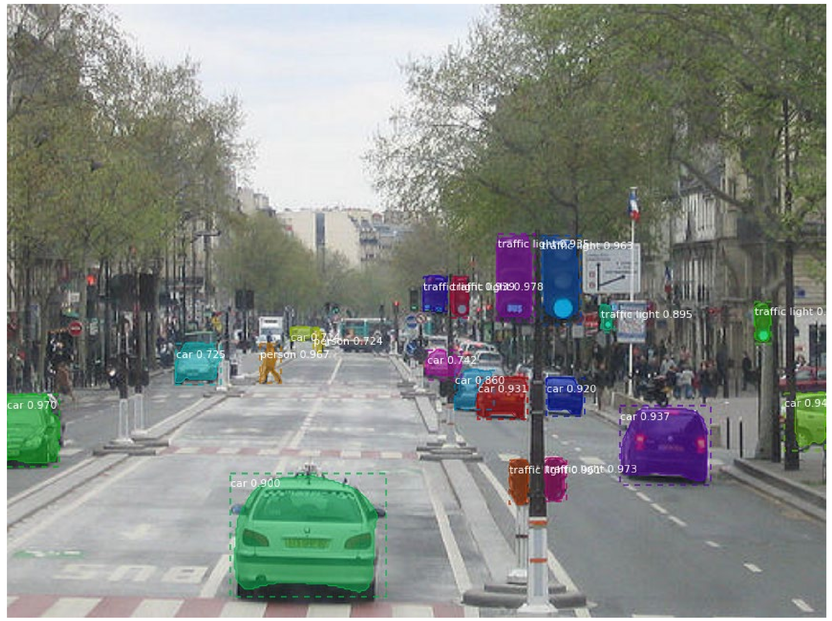


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

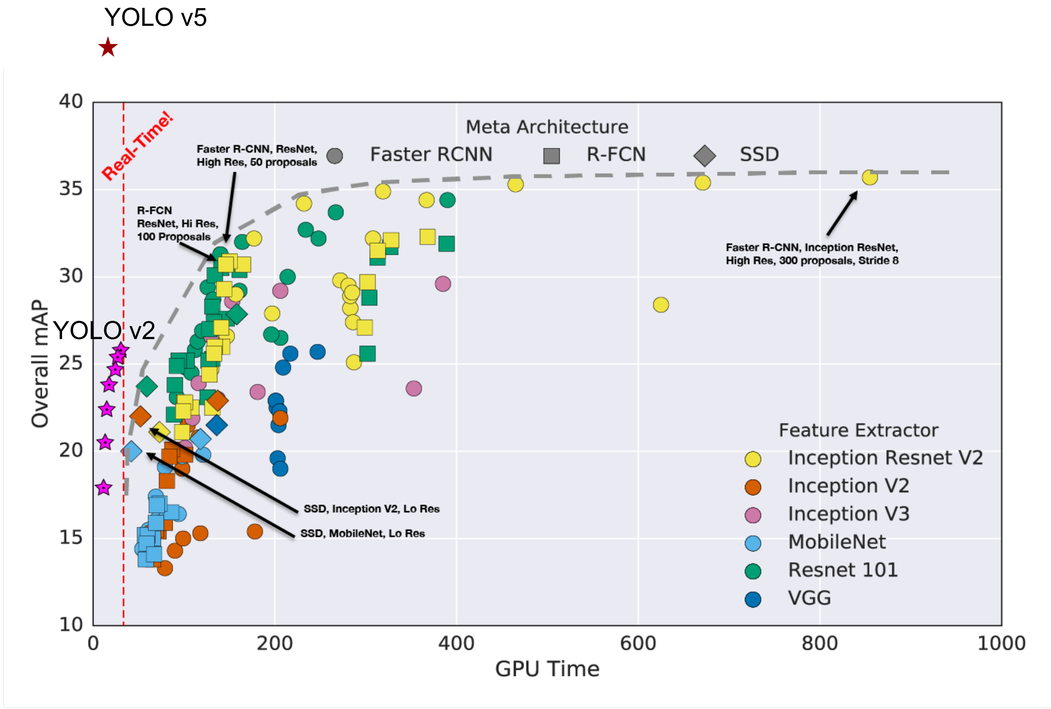
- For every bounding box, segment the object
- One object per bounding box
- Instance segmentation



Mask R-CNN. https://arxiv.org/abs/1703.06870

https://github.com/matterport/Mask_RCNN

Put them together



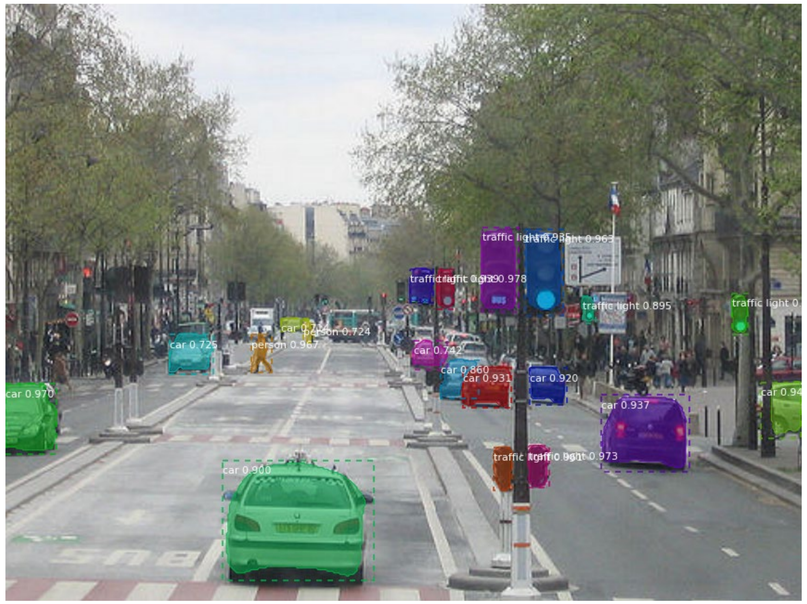
Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{test} 0.5:0.95	mAP^{val} 0.5	Speed V100 (ms)	params (M)	FLOPs 640 (B)
YOLOv5s	640	36.7	36.7	55.4	**2.0**	7.3	17.0
YOLOv5m	640	44.5	44.5	63.1	2.7	21.4	51.3
YOLOv5l	640	48.2	48.2	66.9	3.8	47.0	115.4
YOLOv5x	640	**50.4**	**50.4**	**68.8**	6.1	87.7	218.8

https://github.com/facebookresearch/detectron2

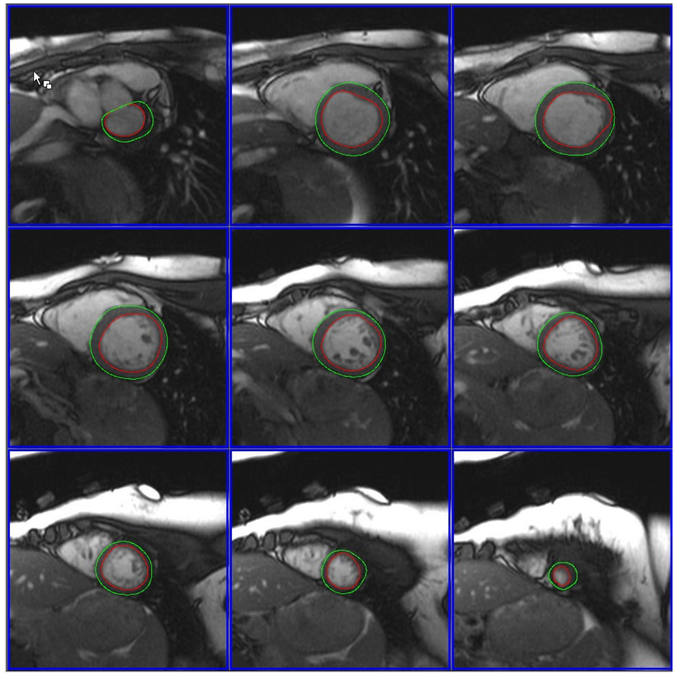
Speed/accuracy trade-offs for modern convolutional object detectors. https://arxiv.org/abs/1611.10012

Object Detection in 20 Years: A Survey. https://arxiv.org/abs/1905.05055

Usecase discrepancy: Object detection vs. Imaging



- Don’t know what is in the image
- Large number of object classes
- Less accurate segmentation is ok
- Require real-time processing
- Segmentation is limited by the low resolution of conv feature maps



- Know what is imaged, from protocols
- Small number of object classes
- Very accurate segmentation is needed
- Often do not require real-time processing, e.g. <500ms for a model call is ok in most cases

Segmentation : output the full resolution

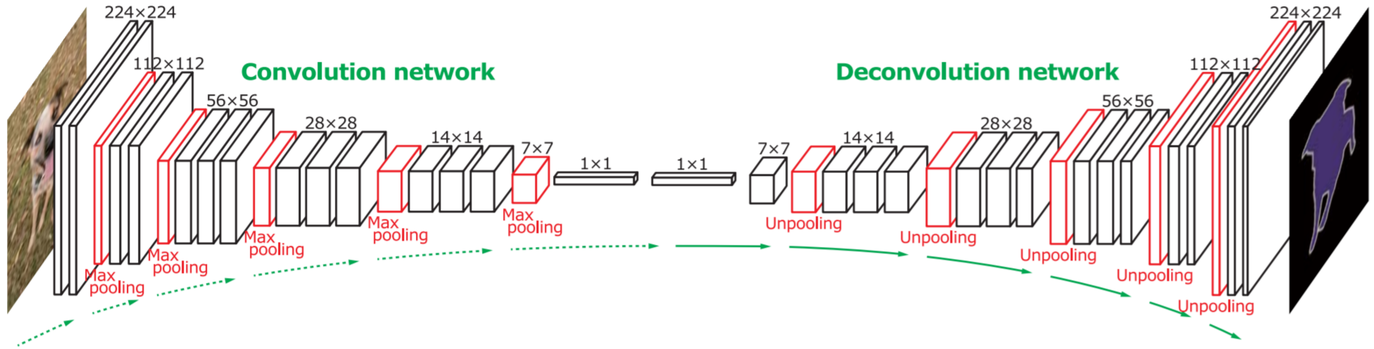


Figure 2. Overall architecture of the proposed network. On top of the convolution network based on VGG 16-layer net, we put a multi-layer deconvolution network to generate the accurate segmentation map of an input proposal. Given a feature representation obtained from the convolution network, dense pixel-wise class prediction map is constructed through multiple series of unpooling, deconvolution and rectification operations.

- Encoder-decoder architecture
- Gradually reduce image size, increase number of filters
- Bottle-neck at the very low-resolution layers

Learning Deconvolution Network for Semantic Segmentation. https://arxiv.org/abs/1505.04366

Fully Convolutional Networks for Semantic Segmentation. https://arxiv.org/abs/1411.4038

Segmentation : U-net

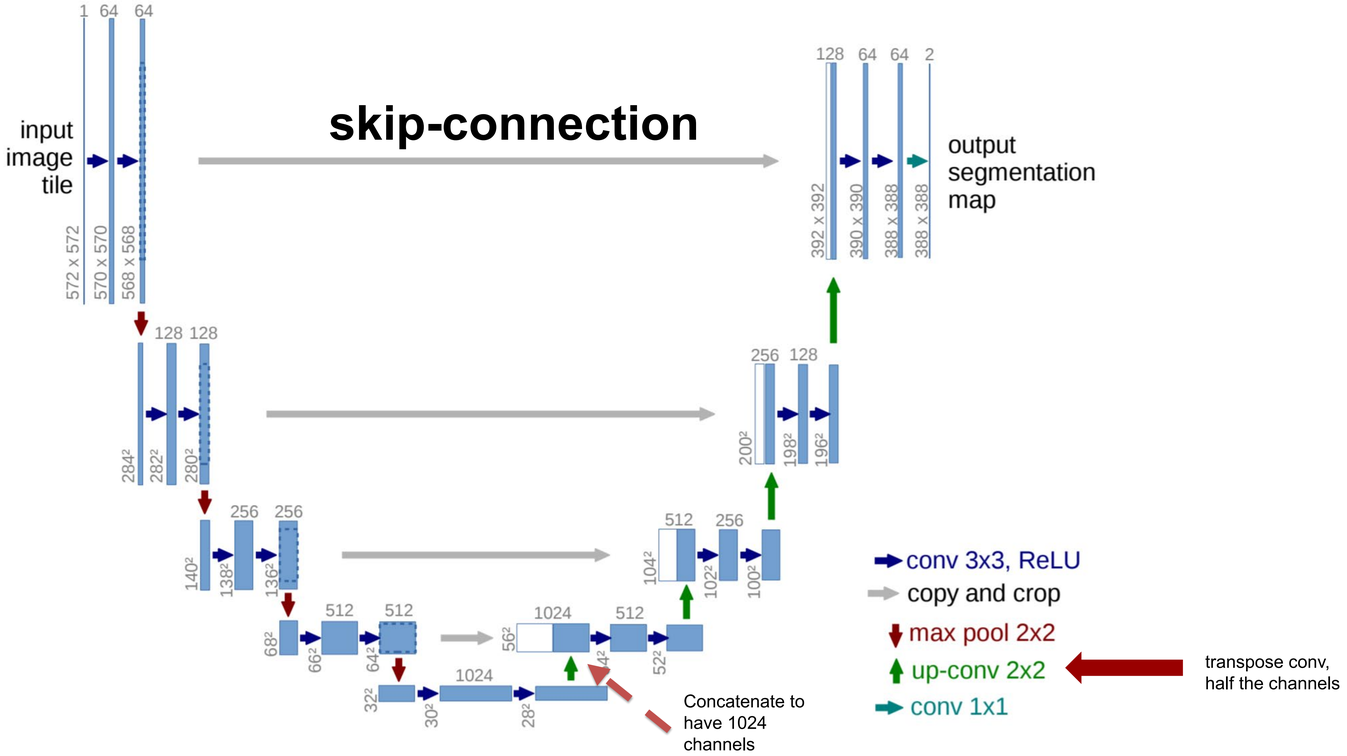
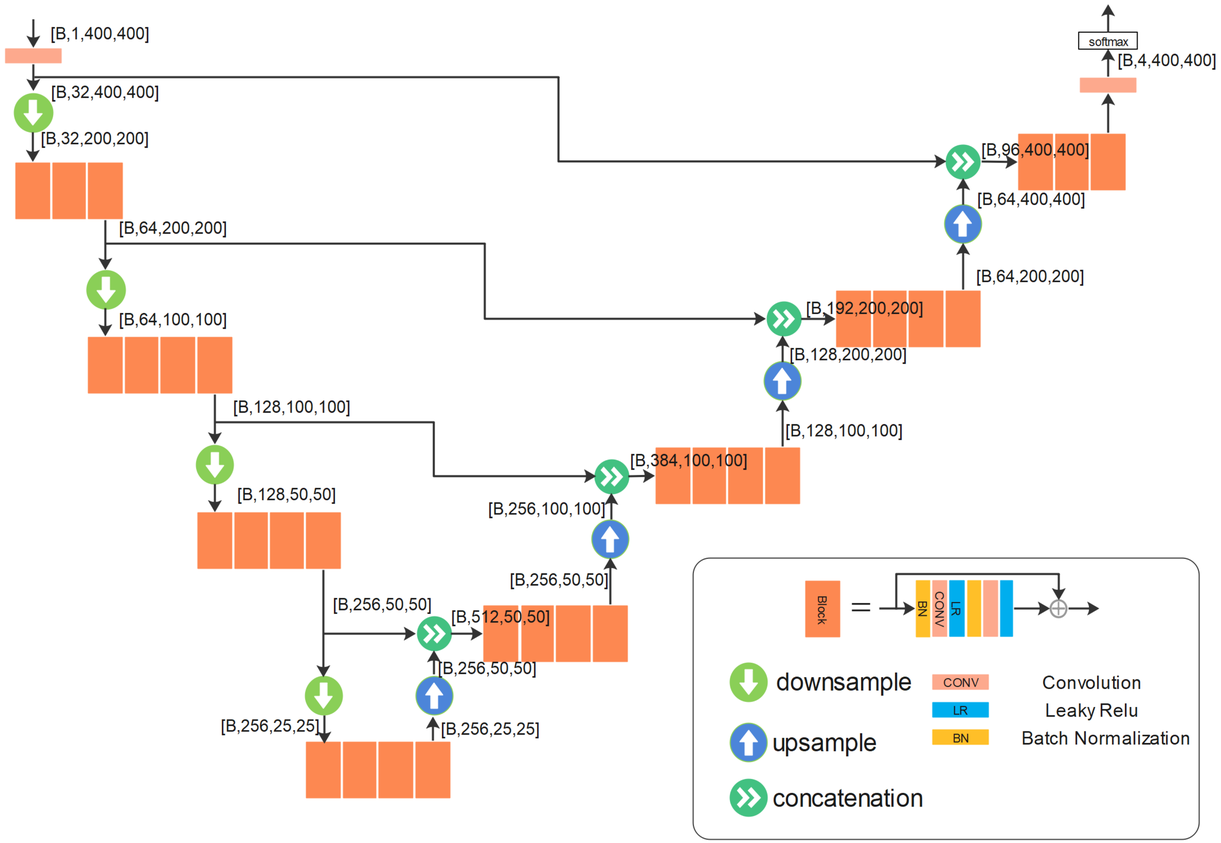


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

- Encoder-decoder architecture
- Gradually reduce image size, increase number of filters
- Remove the bottle-neck at the very low-resolution layers by adding the **long-range skip-connections**
- **Downsample/upsample layers**
- Concatenate the filters, not add → compare to the residual connection
- Allow network to use information from different resolution levels

U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/abs/1505.04597

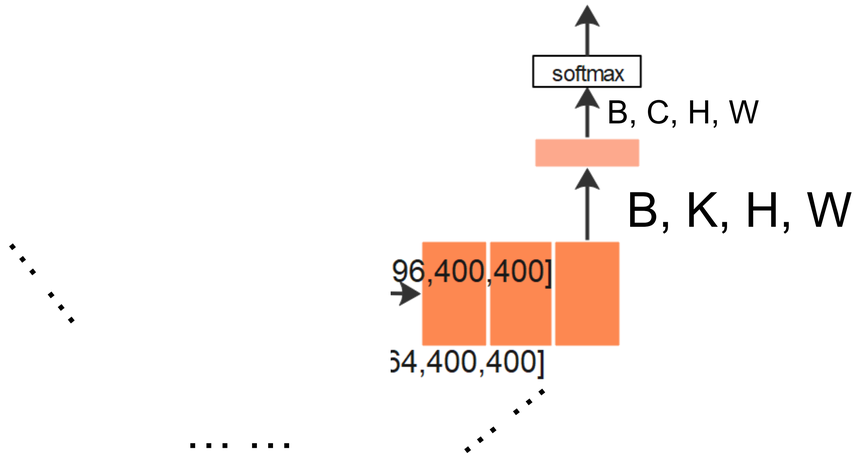
Segmentation : ResU-net



- Add short-range **residual connections** in each block
- Use interpolation for down/upsample
- Add batch normalization
- Downsample branch increases number of channels
- Upsample branch decreases number of channels

Landmark detection in Cardiac Magnetic Resonance Imaging Using A Convolutional Neural Network. https://pubs.rsna.org/doi/10.1148/ryai.2021200197

Segmentation : Output layer and loss

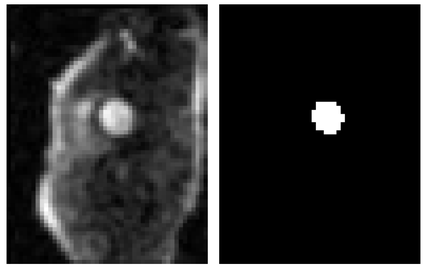


$$\ell = \frac{1}{HW}\sum_{i=0}^{HW-1} BCE\_loss(i)$$ C=1

$$\ell = \frac{1}{HW}\sum_{i=0}^{HW-1} CE\_loss(i)$$ C>1

- Single-class segmentation

  One target object in the label

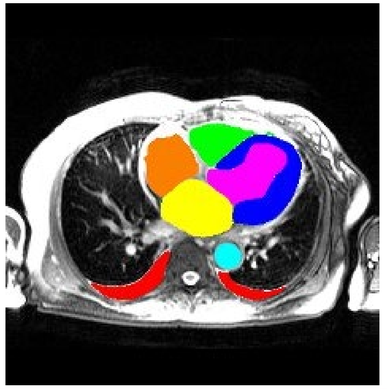


C=1

No need to count background

- Multi-class segmentation

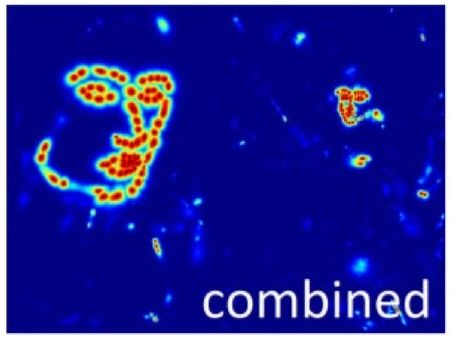
  More than one target objects in the label

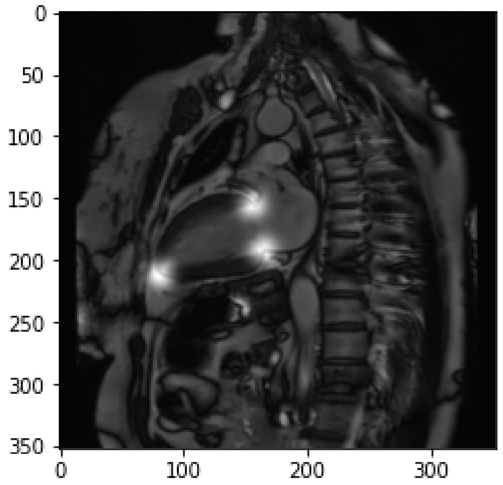


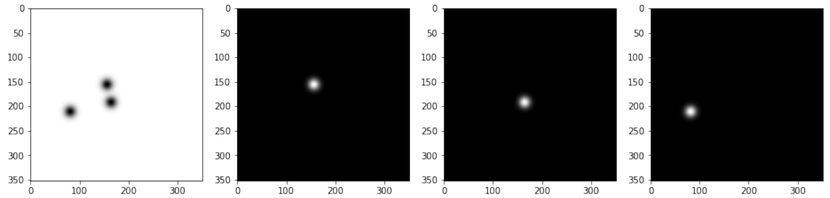
C=7+1

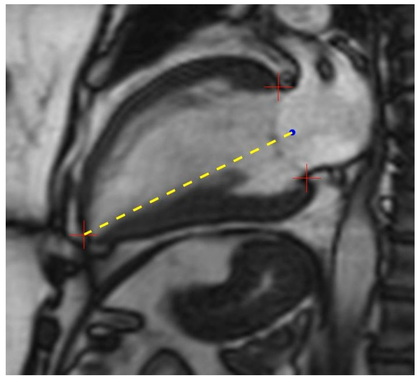
Need to count background

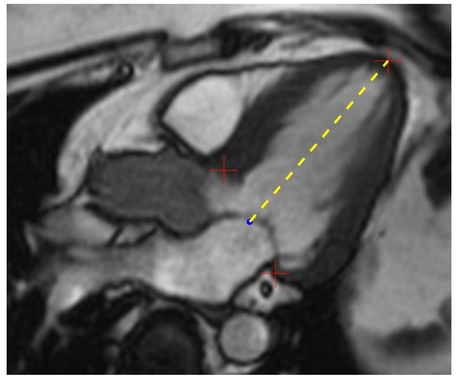
Segmentation : applied to landmark detection

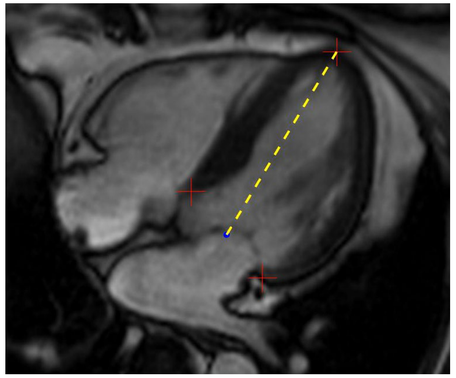








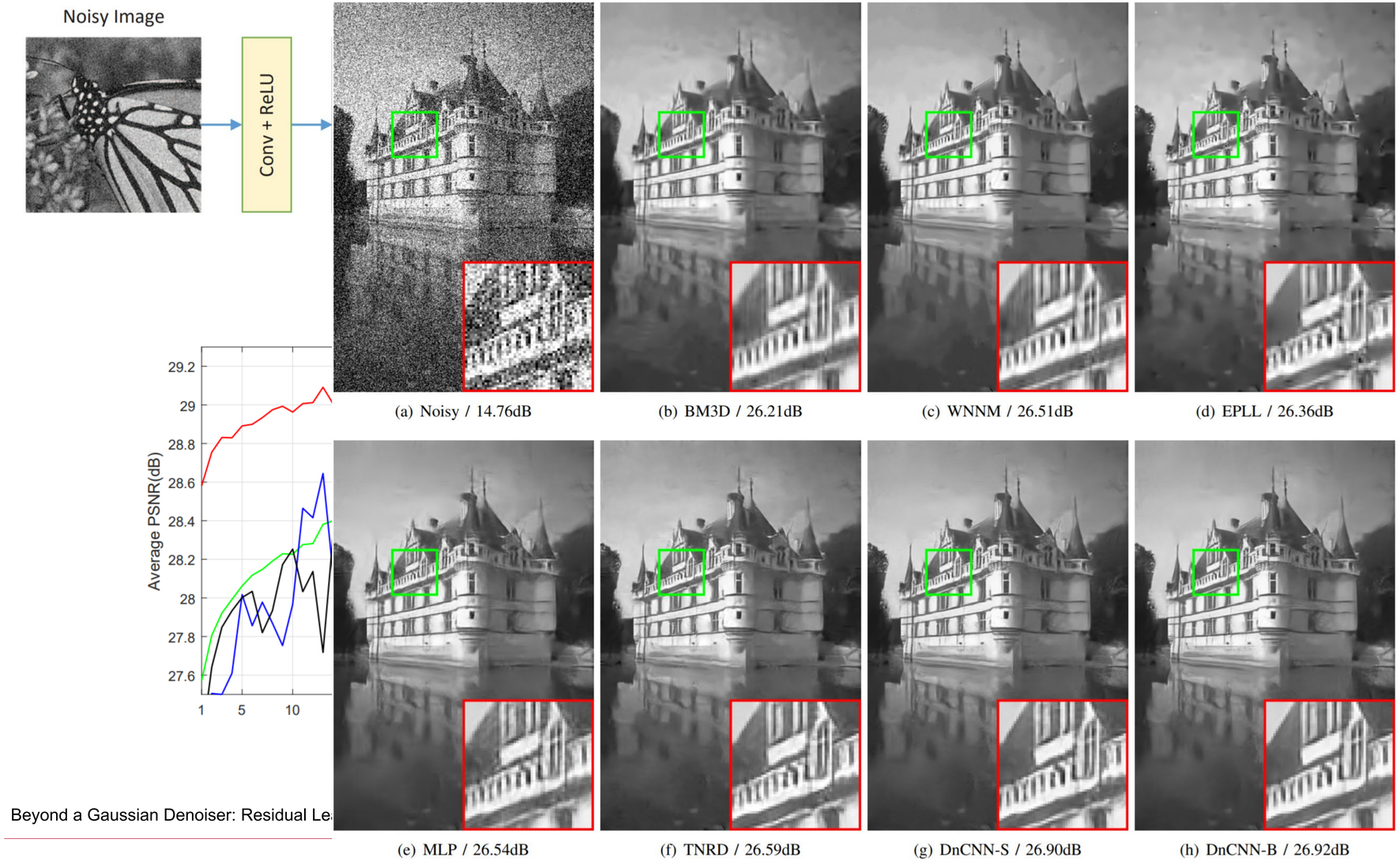




- More robust than L2 regression
- Consider surrounding context to detect landmarks
- Handle the occlusion nicely

Robust Facial Landmark Detection via a Fully-Convolutional Local-Global Context Network.CVPR 2018.
Landmark detection in Cardiac Magnetic Resonance Imaging Using A Convolutional Neural Network. https://arxiv.org/abs/2008.06142

Denoising CNN: DnCNN



(a) Noisy / 14.76dB (b) BM3D / 26.21dB (c) WNNM / 26.51dB (d) EPLL / 26.36dB

(e) MLP / 26.54dB (f) TNRD / 26.59dB (g) DnCNN-S / 26.90dB (h) DnCNN-B / 26.92dB

*white noise as input

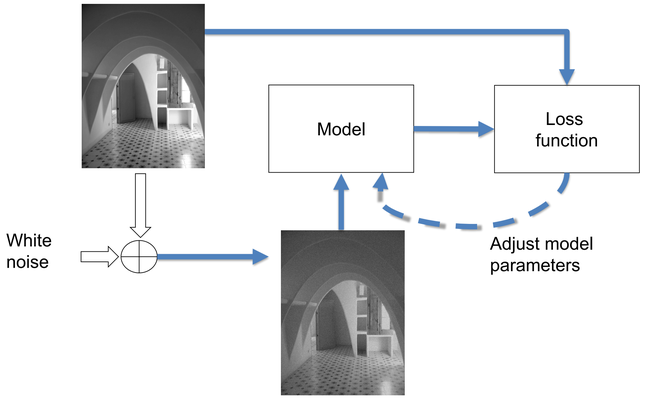
output → residual
BL)

cropped to random
and these patches are
raining → patch training

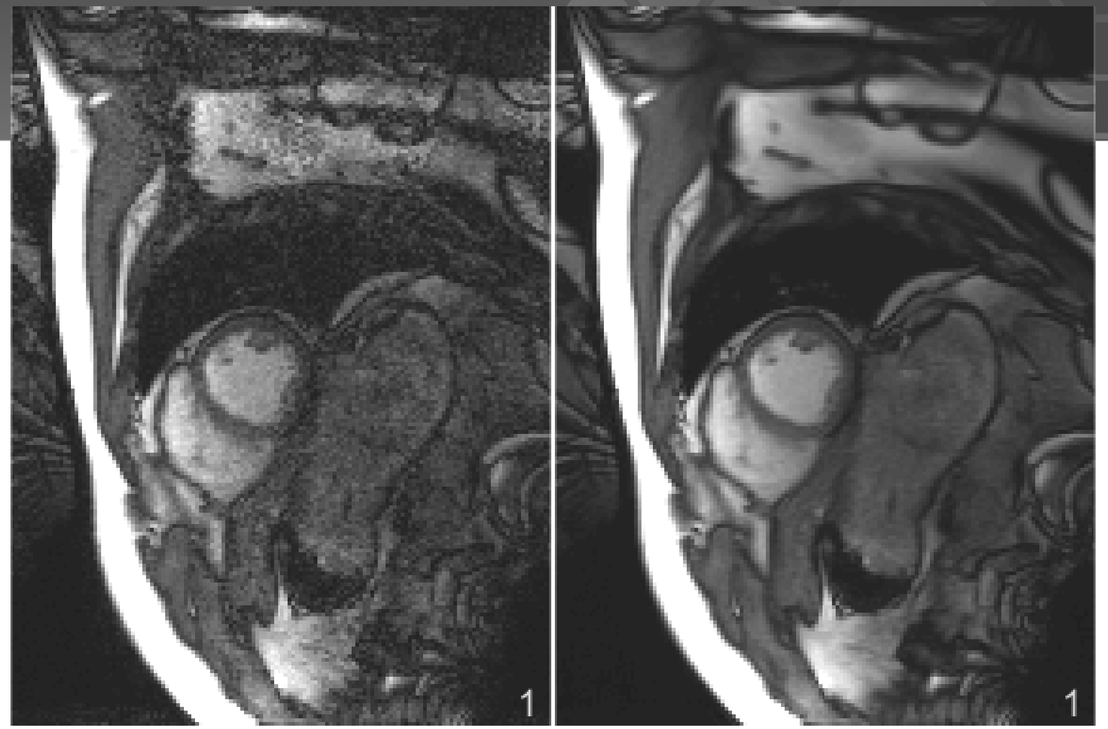
her development on top
rk

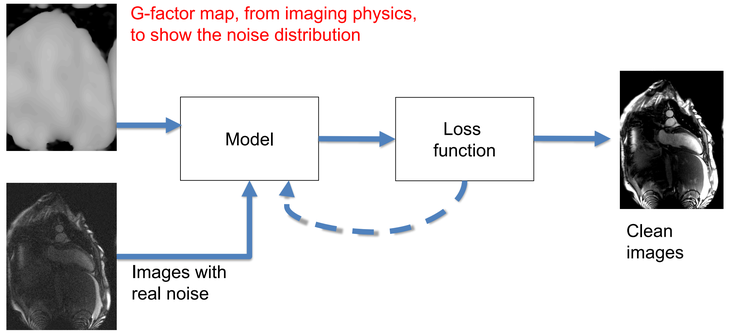
Beyond a Gaussian Denoiser: Residual Le

Denoising with imaging physics

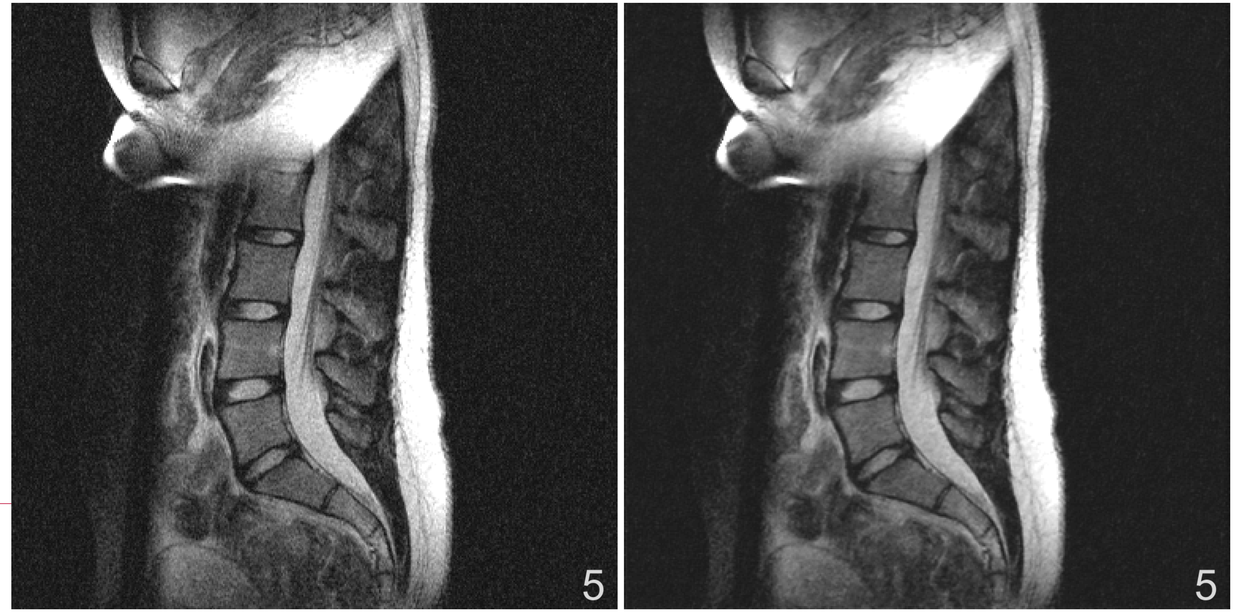


- Develop for consumer electronics
- Not optimal for medical imaging



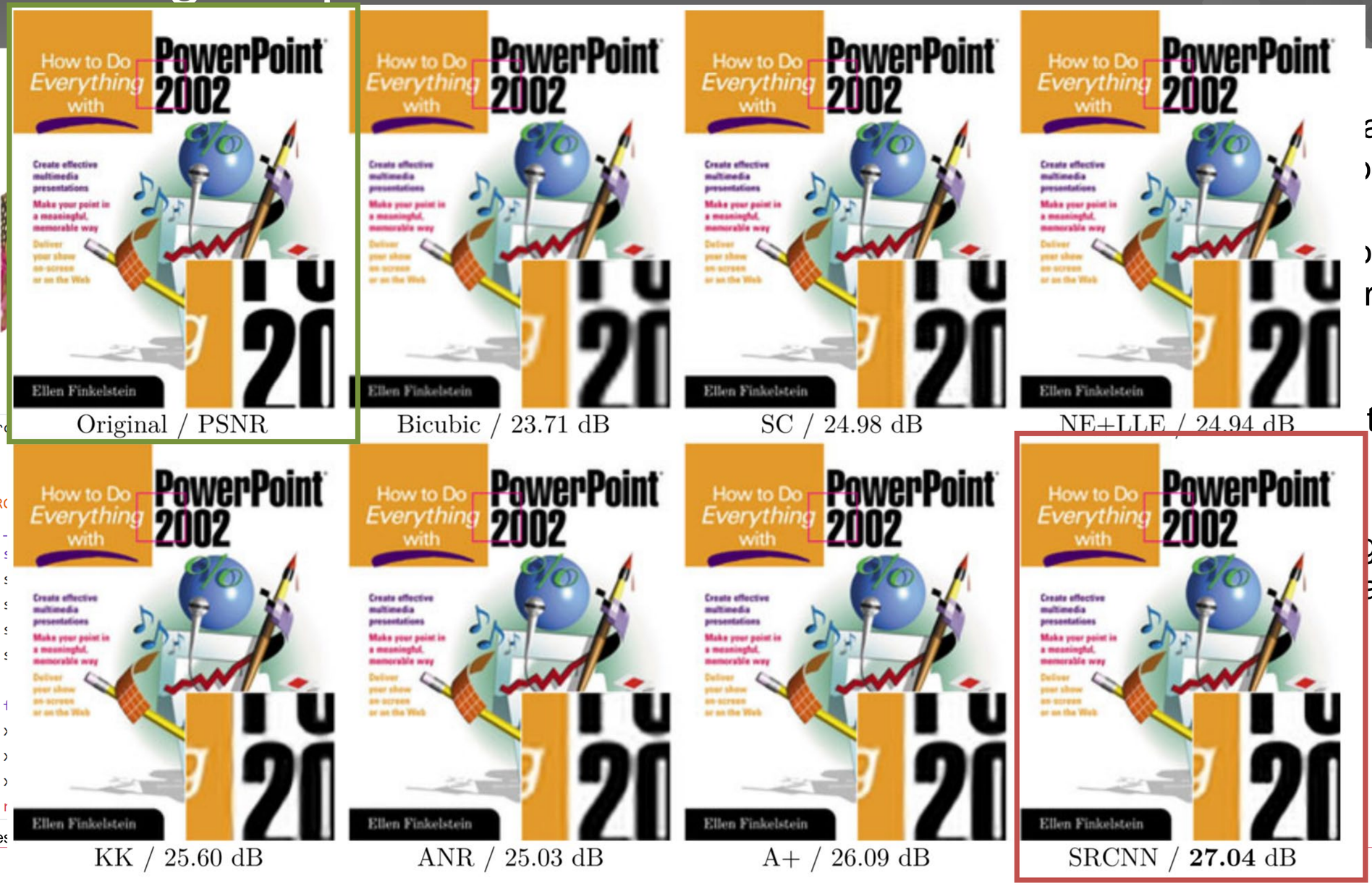


- Integrate imaging physics knowledge



General-purpose Deep Learning Image Denoising based on Magnetic Resonance Imaging Physics. US Patent application.

Single-image super-resolution CNN



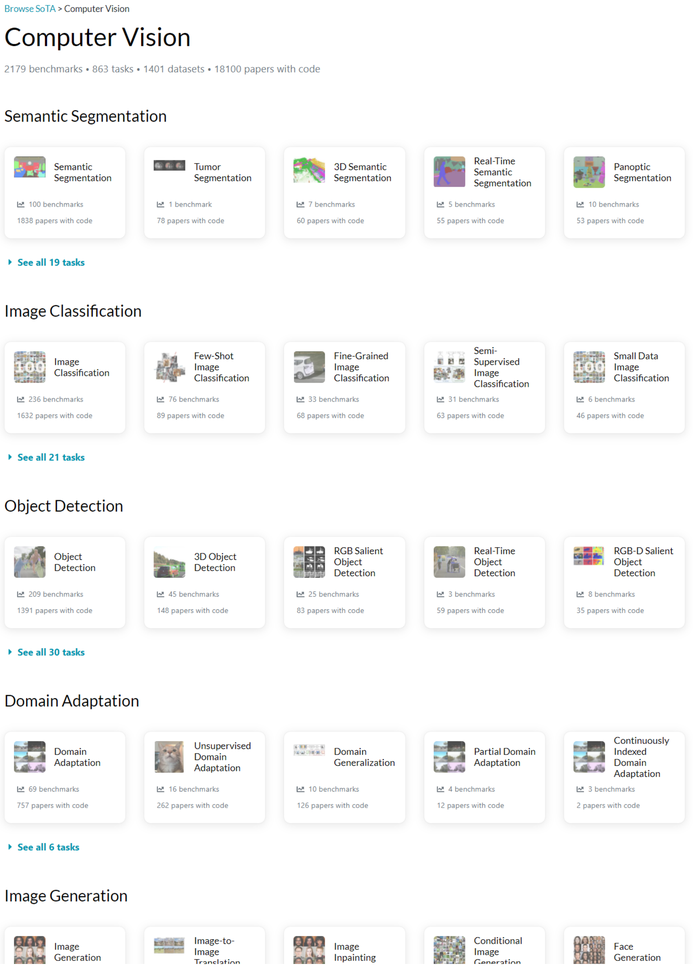
ale to
olation

om low-
res

ture

gh-res
age

A lot more is out there …



https://github.com/aiff22/PyNET

https://arxiv.org/pdf/2002.05509.pdf

https://www.paperswithcode.com/area/computer-vision