



INTERNATIONAL INSTITUTE OF  
INFORMATION TECHNOLOGY  
HYDERABAD



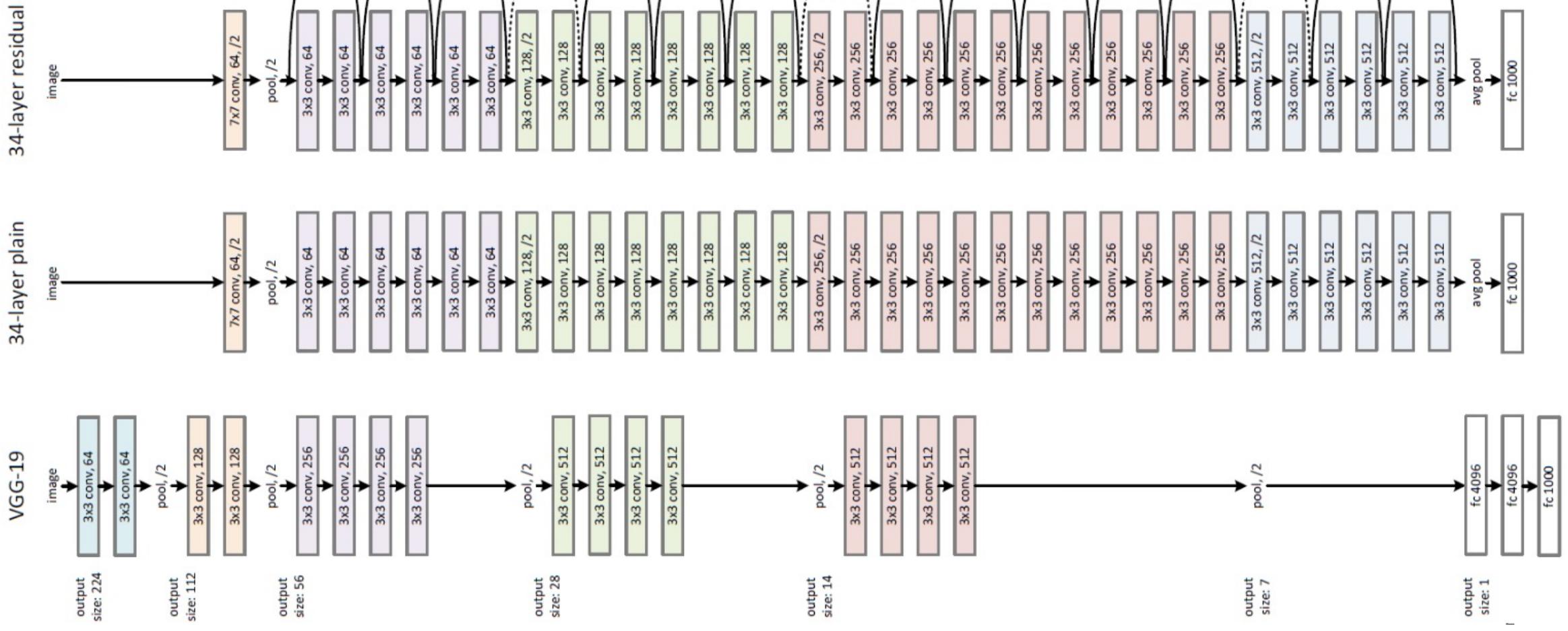
# CNN Architectures

Makarand Tapaswi

CS7.505 Spring 2024

14<sup>th</sup> February 2024

# ResNet (2015)



# Residual connections are very powerful!

- Solve the problems of vanishing gradients

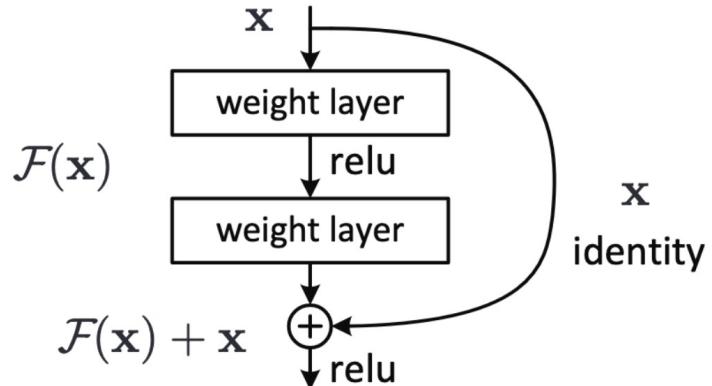


Figure 2. Residual learning: a building block.



arXiv

<https://arxiv.org> > cs

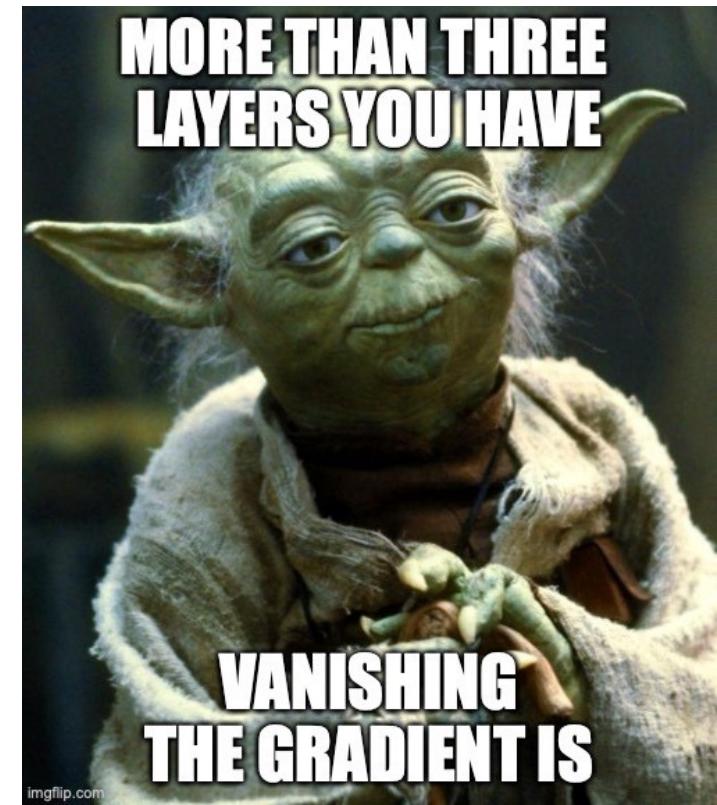


8

:

[1512.03385] Deep Residual Learning for Image Recognition

by K He · 2015 · Cited by 172159 — We present a **residual learning** framework to ease the training of **networks** that are substantially deeper than those used previously.



Ok, maybe not exactly at 3,  
but you get the point

# Efficacy of residual connections

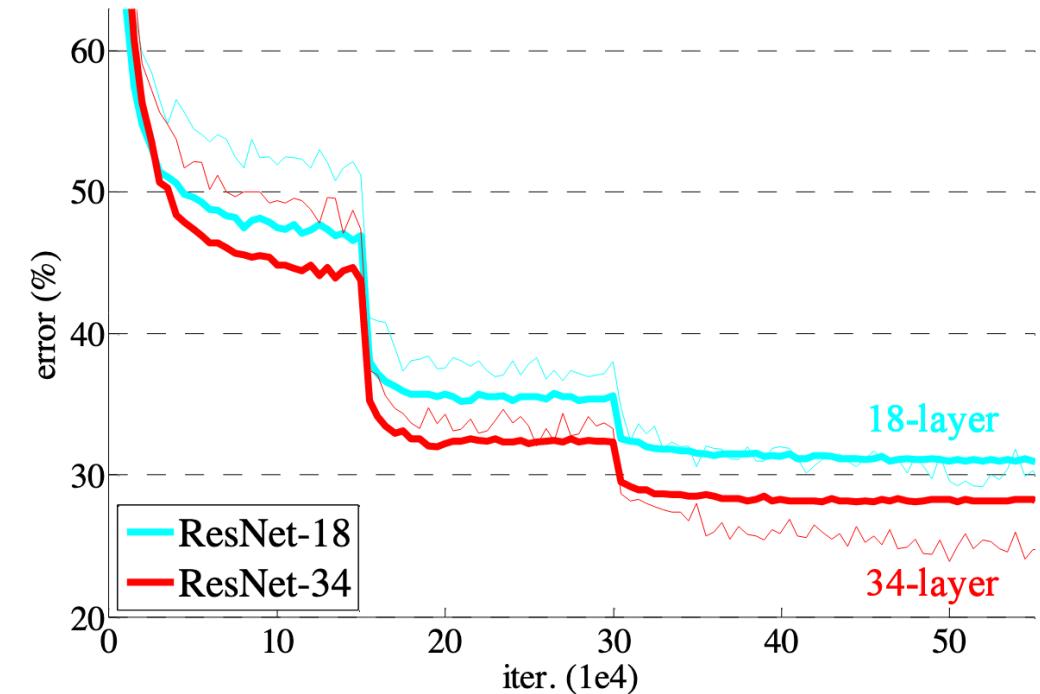
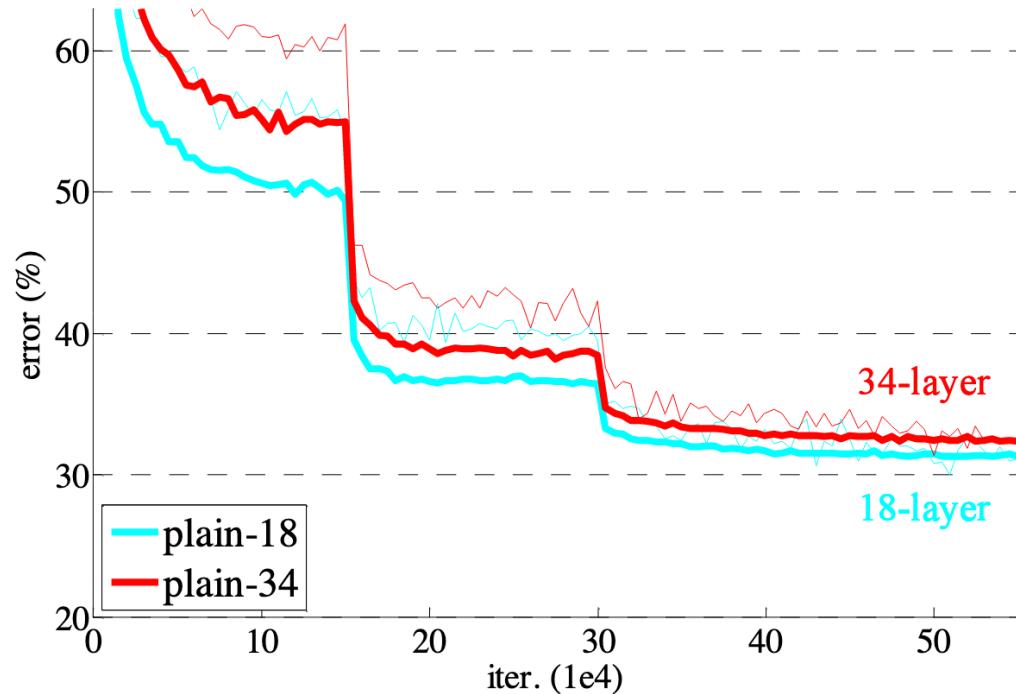


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

# Resnet modules

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

# Resnet 18

```
In [3]: model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=True)
Downloading: "https://github.com/pytorch/vision/archive/v0.10.0.zip" to /home/users/makarand/.cache/torch/hub/v0.10.0.zip
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /home/users/makarand/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:02<00:00, 17.5MB/s]

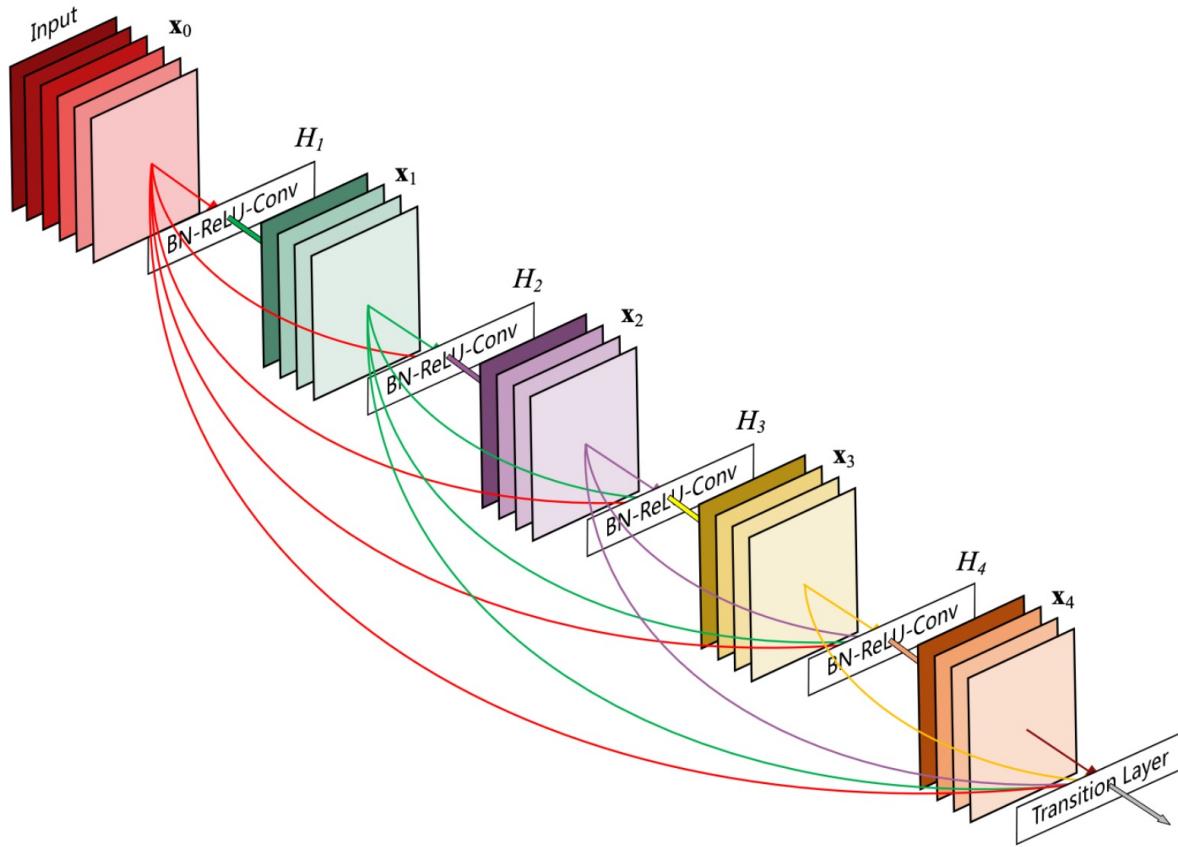
In [4]: model
Out[4]:
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
)
```

```
(layer3): Sequential(  
    (0): BasicBlock(  
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (downsample): Sequential(  
            (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)  
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
    )  
    (1): BasicBlock(  
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    )  
)  
(layer4): Sequential(  
    (0): BasicBlock(  
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)  
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (downsample): Sequential(  
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)  
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        )  
    )  
    (1): BasicBlock(  
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
        (relu): ReLU(inplace=True)  
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    )  
)  
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))  
(fc): Linear(in_features=512, out_features=1000, bias=True)
```

# ResNet

- Add residual connections between **basic blocks**
- Easy gradient flow for deep models
- Enable very deep models
  - from ~20 layers to 150 and even 1000 layers!
- “Fully convolutional”: No linear layers except the classifier!

# DenseNet (2016)



arXiv

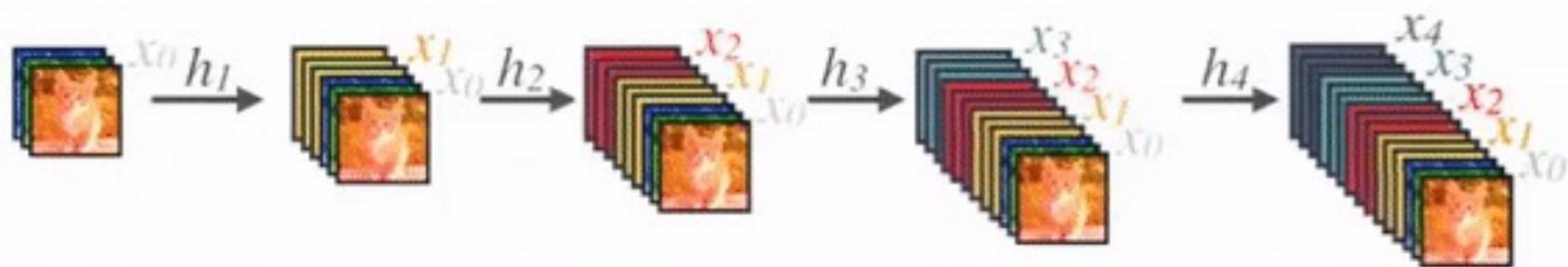
<https://arxiv.org> > cs



:

[1608.06993] Densely Connected Convolutional Networks

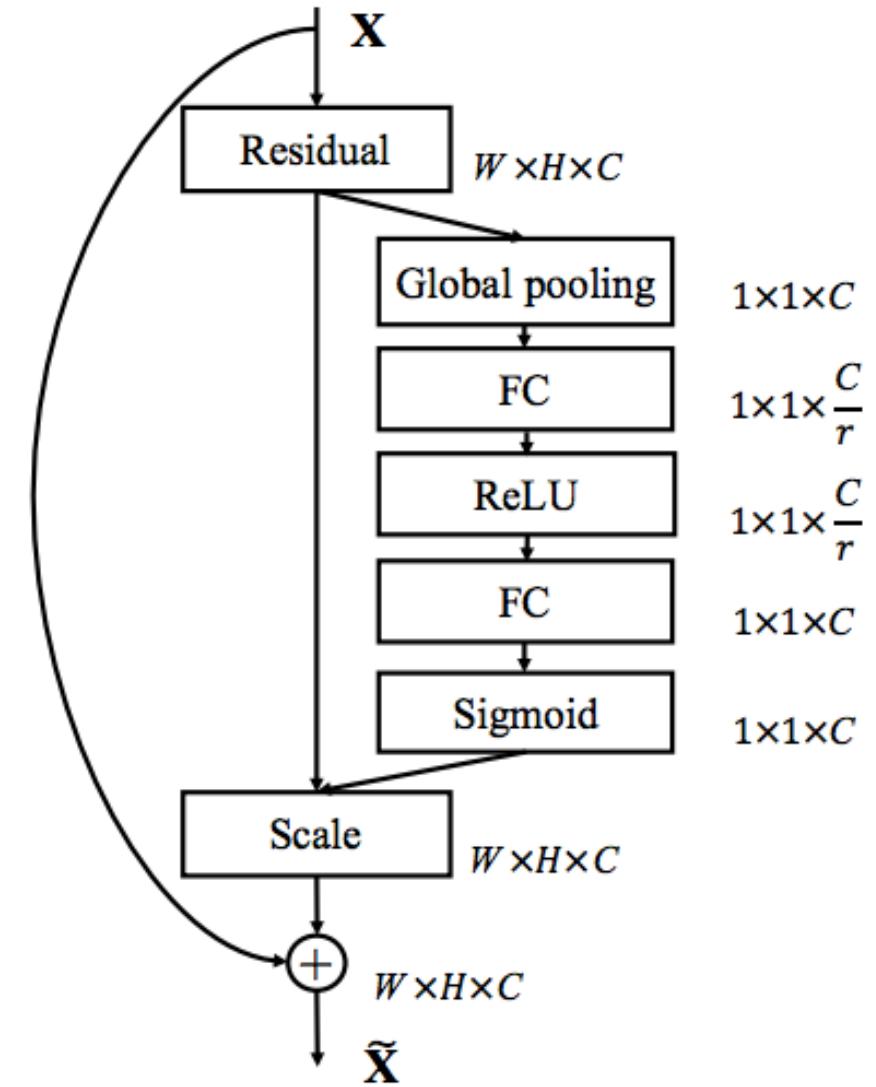
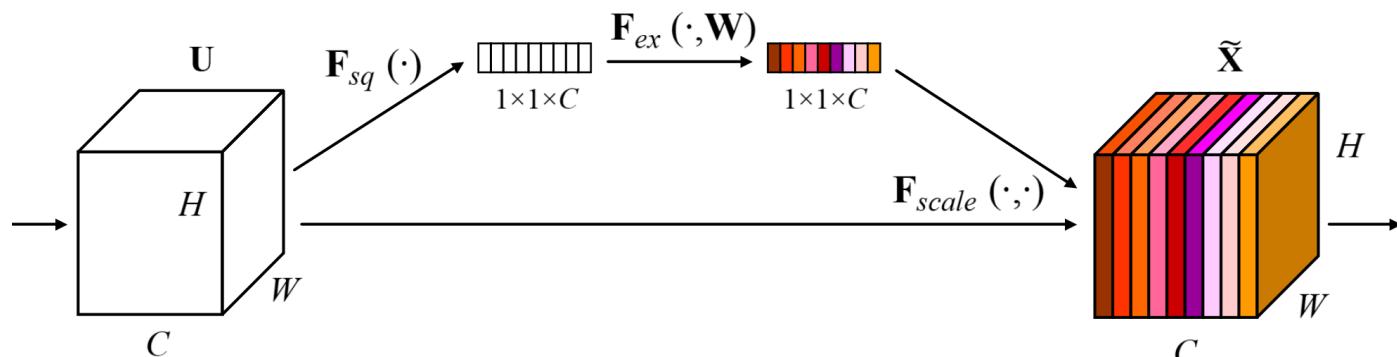
by G Huang · 2016 · Cited by 36367 — In this paper, we embrace this observation and introduce the Dense Convolutional Network (**DenseNet**), which connects each layer to every ...



# DenseNet Architectures

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$		$7 \times 7$ conv, stride 2		
Pooling	$56 \times 56$		$3 \times 3$ max pool, stride 2		
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$		$1 \times 1$ conv		
	$28 \times 28$		$2 \times 2$ average pool, stride 2		
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$		$1 \times 1$ conv		
	$14 \times 14$		$2 \times 2$ average pool, stride 2		
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$		$1 \times 1$ conv		
	$7 \times 7$		$2 \times 2$ average pool, stride 2		
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$		$7 \times 7$ global average pool		
			1000D fully-connected, softmax		

# SENet (2017)



[1709.01507] Squeeze-and-Excitation Networks

by J Hu · 2017 · Cited by 22533 — We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different ...

**SE-ResNet Module**

# Stacking considerations

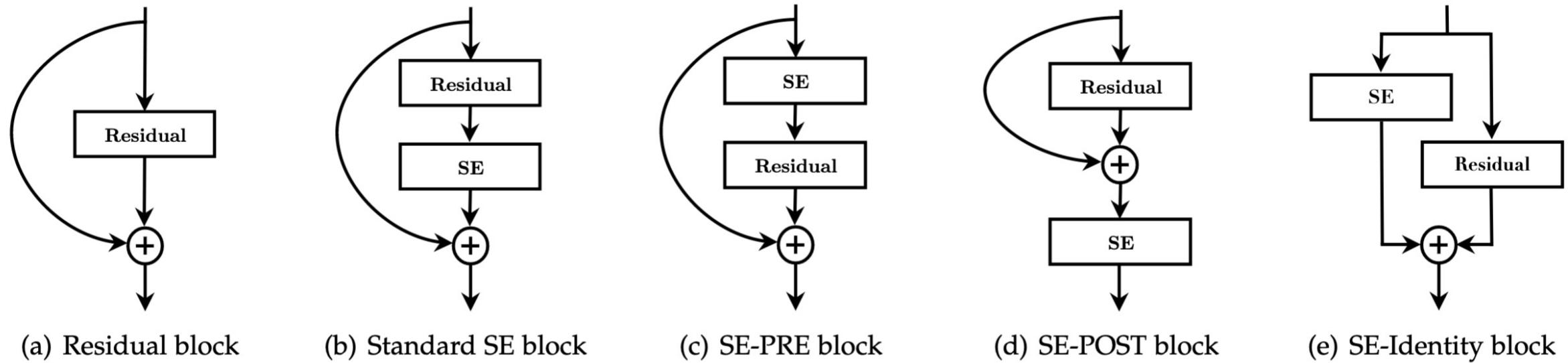
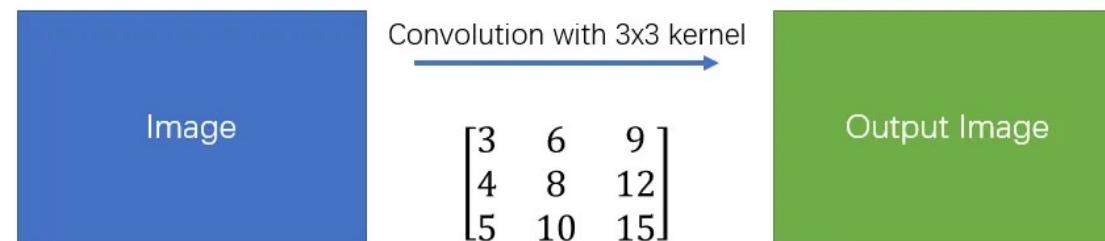


Fig. 5. SE block integration designs explored in the ablation study.

# Spatial Separable Convolutions

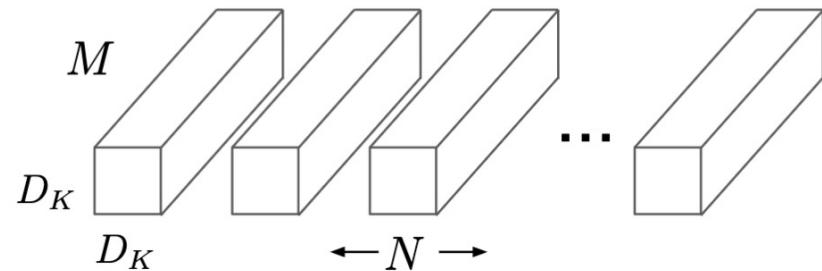
## Simple Convolution



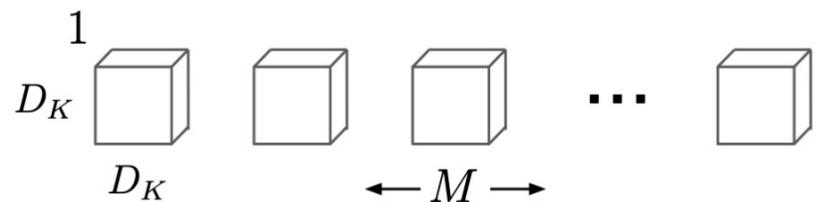
## Spatial Separable Convolution



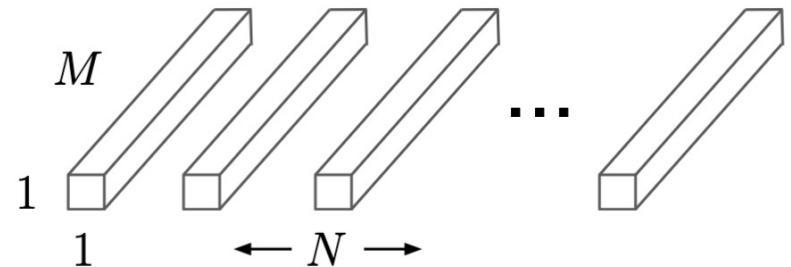
# MobileNet (2017)



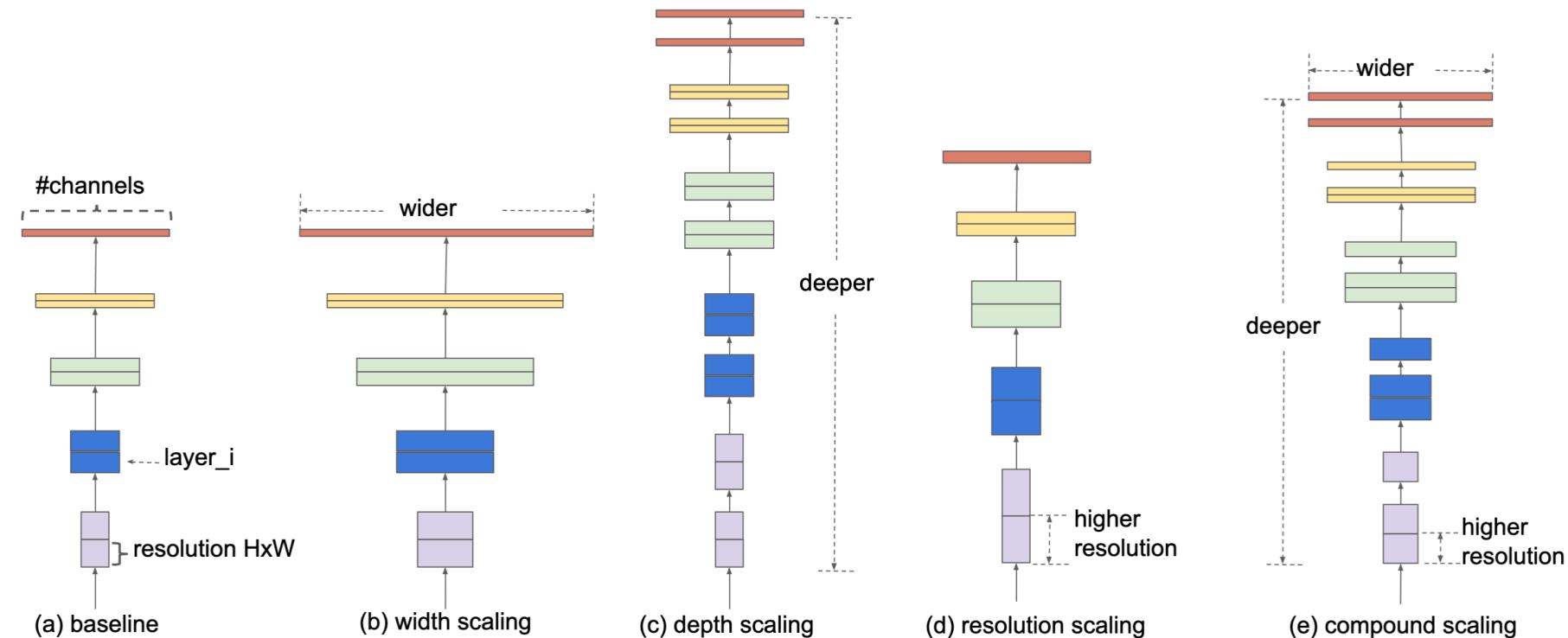
(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

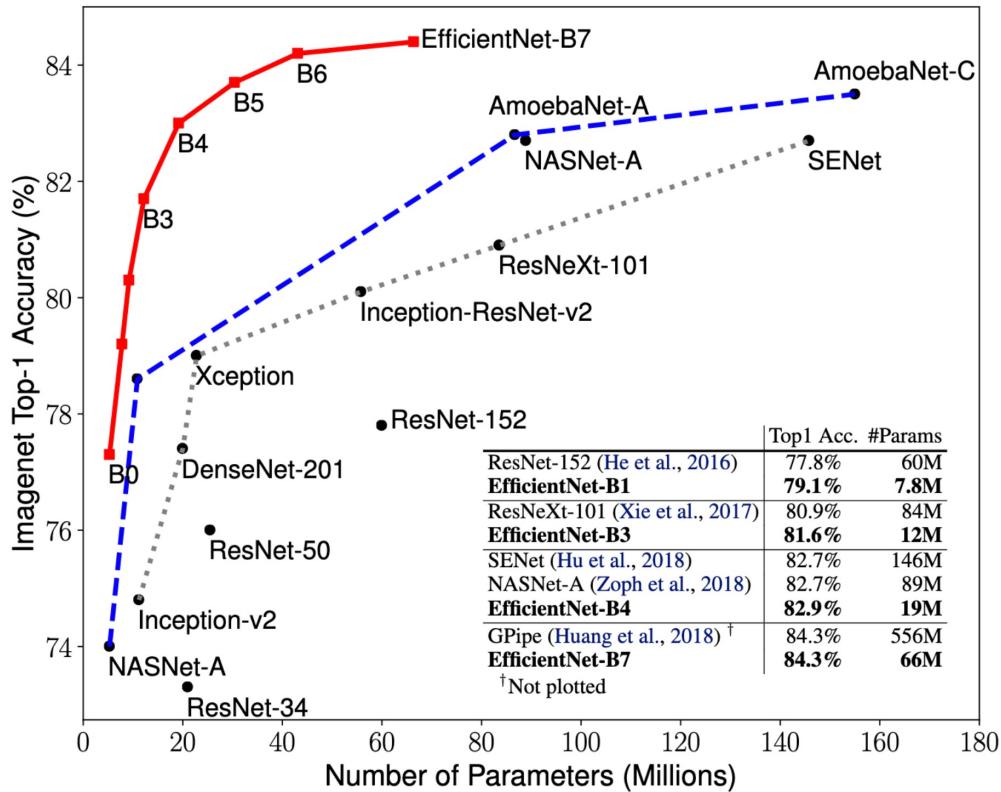


# EfficientNet (2019)



**Figure 2. Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

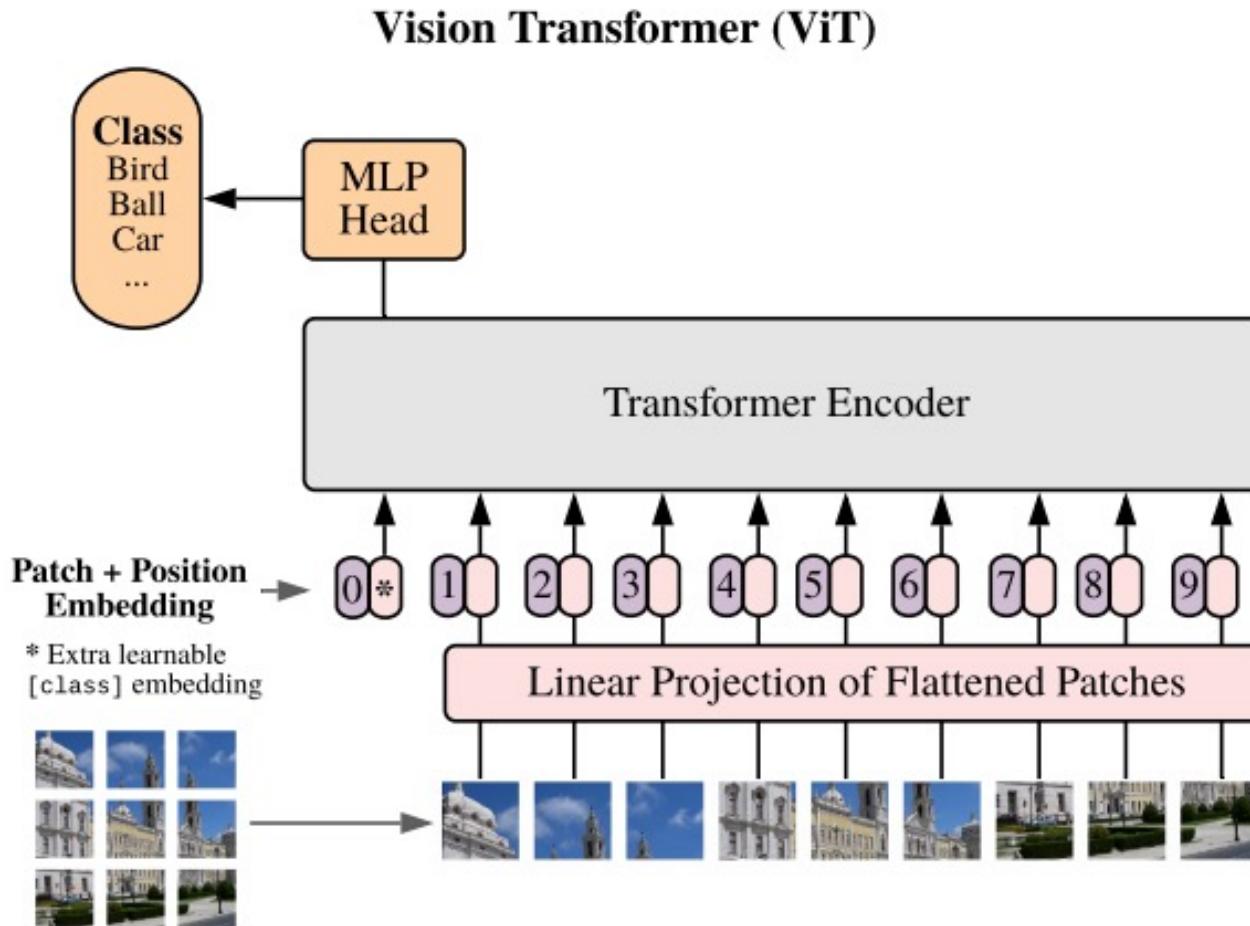
# EfficientNet



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
<b>EfficientNet-B0</b>	<b>77.1%</b>	<b>93.3%</b>	<b>5.3M</b>	<b>1x</b>	<b>0.39B</b>	<b>1x</b>
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
<b>EfficientNet-B1</b>	<b>79.1%</b>	<b>94.4%</b>	<b>7.8M</b>	<b>1x</b>	<b>0.70B</b>	<b>1x</b>
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
<b>EfficientNet-B2</b>	<b>80.1%</b>	<b>94.9%</b>	<b>9.2M</b>	<b>1x</b>	<b>1.0B</b>	<b>1x</b>
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
<b>EfficientNet-B3</b>	<b>81.6%</b>	<b>95.7%</b>	<b>12M</b>	<b>1x</b>	<b>1.8B</b>	<b>1x</b>
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
<b>EfficientNet-B4</b>	<b>82.9%</b>	<b>96.4%</b>	<b>19M</b>	<b>1x</b>	<b>4.2B</b>	<b>1x</b>
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
<b>EfficientNet-B5</b>	<b>83.6%</b>	<b>96.7%</b>	<b>30M</b>	<b>1x</b>	<b>9.9B</b>	<b>1x</b>
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
<b>EfficientNet-B6</b>	<b>84.0%</b>	<b>96.8%</b>	<b>43M</b>	<b>1x</b>	<b>19B</b>	<b>1x</b>
<b>EfficientNet-B7</b>	<b>84.3%</b>	<b>97.0%</b>	<b>66M</b>	<b>1x</b>	<b>37B</b>	<b>1x</b>
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

# Vision Transformers: 2020-21



arXiv  
https://arxiv.org > cs 8 ·

## Transformers for Image Recognition at Scale

by A Dosovitskiy · 2020 · Cited by 18051 — Title:An Image is Worth 16x16 Words:  
Transformers for Image Recognition at Scale ... Abstract: While the Transformer architectur...

arXiv  
https://arxiv.org > cs 8 ·

## Hierarchical Vision Transformer using Shifted Windows

by Z Liu · 2021 · Cited by 8390 — This paper presents a new vision Transformer, called **Swin Transformer**, that capably serves as a general-purpose backbone for computer vision.

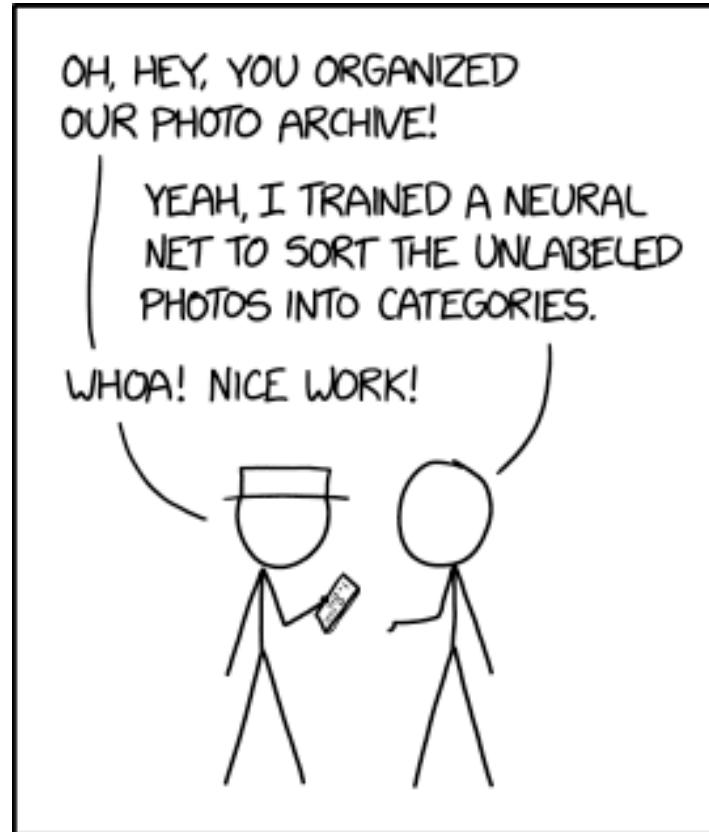
# ConvNeXt (2022)

The "Roaring 20s" of visual recognition began with the introduction of Vision Transformers (ViTs), which quickly superseded ConvNets as the state-of-the-art image classification model. A vanilla ViT, on the other hand, faces difficulties when applied to general computer vision tasks such as object detection and semantic segmentation. It is the hierarchical Transformers (e.g., Swin Transformers) that reintroduced several ConvNet priors, making Transformers practically viable as a generic vision backbone and demonstrating remarkable performance on a wide variety of vision tasks. However, the effectiveness of such hybrid approaches is still largely credited to the intrinsic superiority of Transformers, rather than the inherent inductive biases of convolutions. In this work, we reexamine the design spaces and test the limits of what a pure ConvNet can achieve. We gradually "modernize" a standard ResNet toward the design of a vision Transformer, and discover several key components that contribute to the performance difference along the way. The outcome of this exploration is a family of pure ConvNet models dubbed ConvNeXt. Constructed entirely from standard ConvNet modules, ConvNeXts compete favorably with Transformers in terms of accuracy and scalability, achieving 87.8% ImageNet top-1 accuracy and outperforming Swin Transformers on COCO detection and ADE20K segmentation, while maintaining the simplicity and efficiency of standard ConvNets.

# Neural Architecture Search

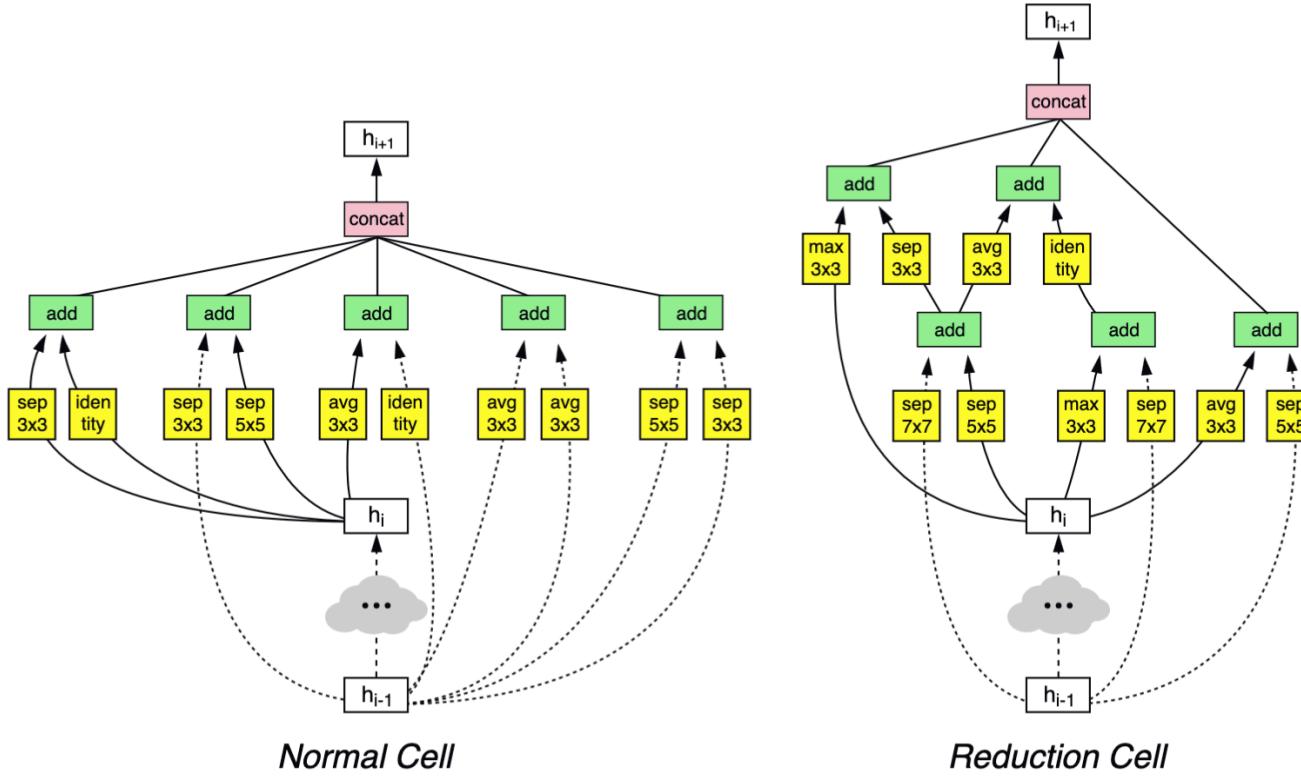


# They said feature engineering would go away!



ENGINEERING TIP:  
WHEN YOU DO A TASK BY HAND,  
YOU CAN TECHNICALLY SAY YOU  
TRAINED A NEURAL NET TO DO IT.

# NASNet (2017)



arXiv

<https://arxiv.org> > cs

8

:

Learning Transferable Architectures for Scalable Image ...

by B Zoph · 2017 · Cited by 5675 — In this paper, we study a method to learn the model architectures directly on the dataset of interest.

# At a glance

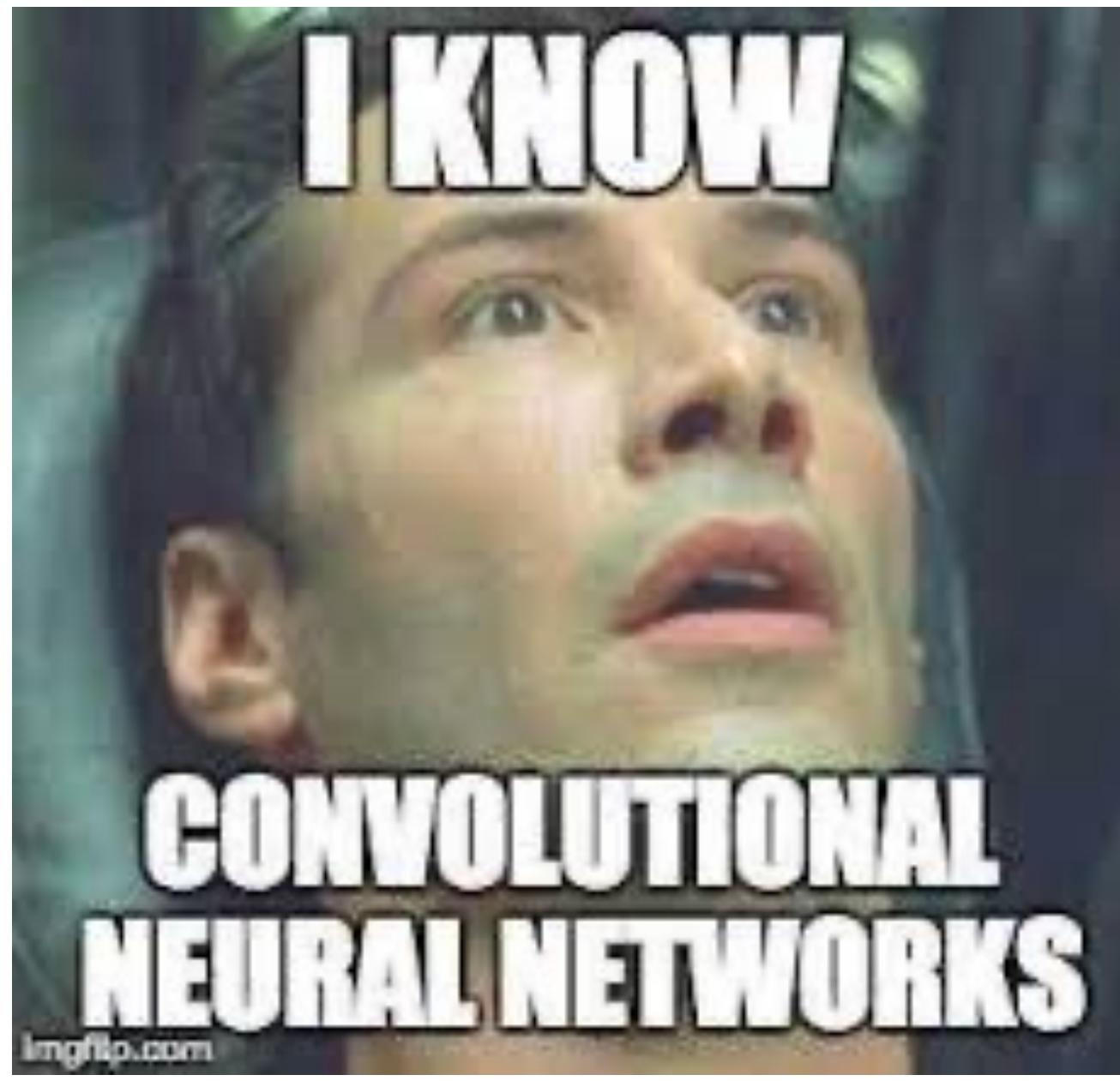
ImageNet-1k trained models

Architecture	Year	Details	Acc1	Acc5	Params	GFLOPs
AlexNet	2012	-	56.522	79.066	61.1M	0.71
VGGNet	2014	VGG-19 BN	74.218	91.842	143.7M	19.63
Inception	2014	Inception-v3	77.294	93.45	27.2M	5.71
ResNet	2015	ResNet-50	76.13	92.862	25.6M	4.09
DenseNet	2016	DenseNet-161	77.138	93.56	28.7M	7.73
EfficientNet	2019	EffNet-B0	77.692	93.532	5.3M	0.39
Vision Transformer	2020	ViT-B/16	81.072	95.318	86.6M	17.56
Swin Transformer	2021	SwinT-B/16	83.582	96.64	87.8M	15.43
ConvNeXt	2022	ConvNeXt-B	84.062	96.87	88.6M	15.36

Sources:

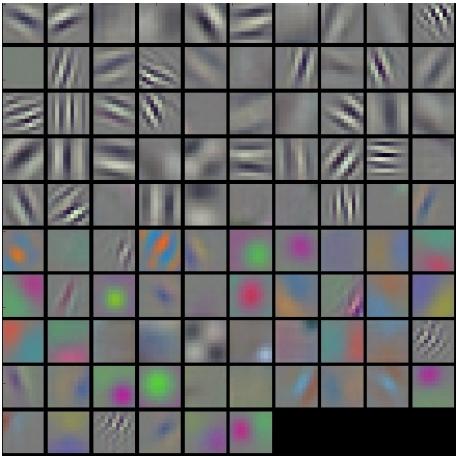
[1] Torchvision model zoo: <https://pytorch.org/vision/stable/models.html>

[2] SENet <https://arxiv.org/abs/1709.01507>

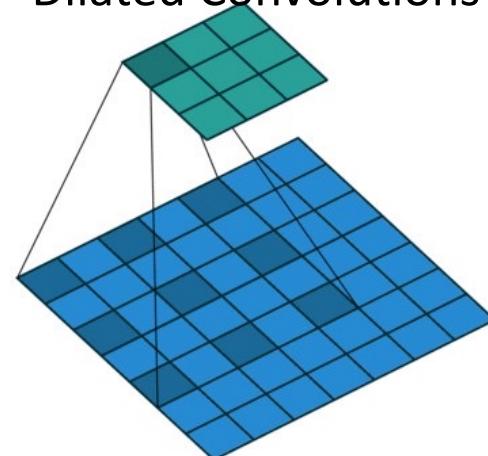


# Dig deeper!

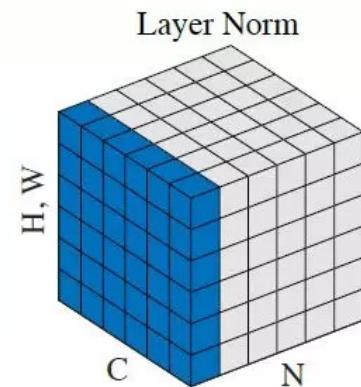
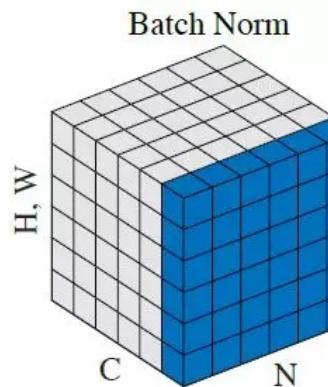
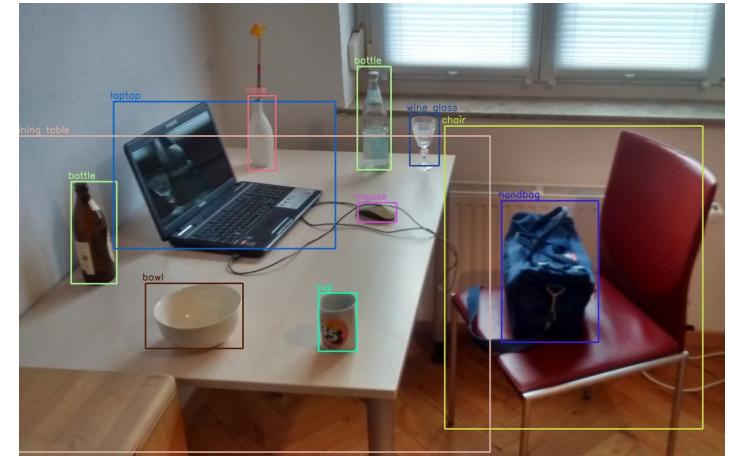
Filter visualization



Dilated Convolutions

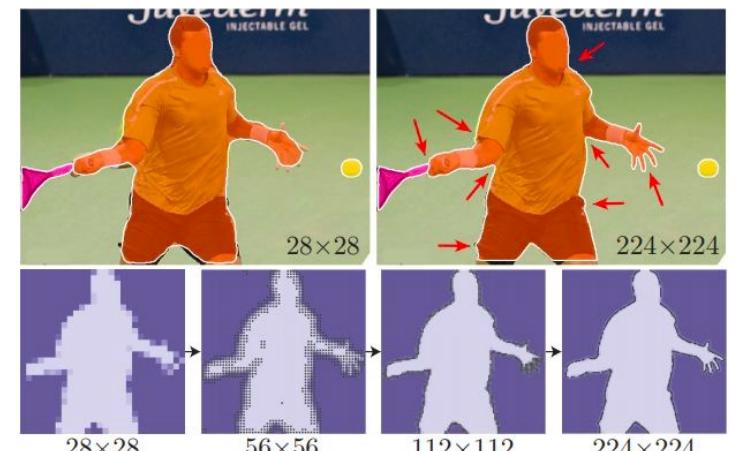
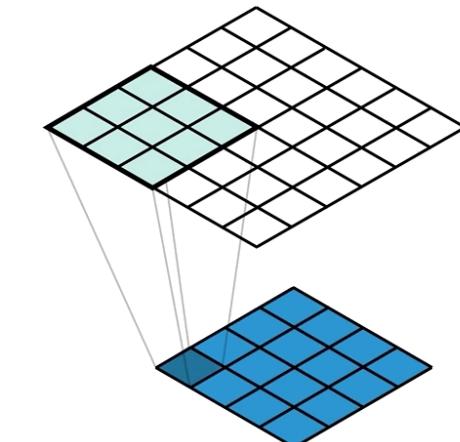


Object Detection



Normalization

Transposed Convolutions



Object Segmentation