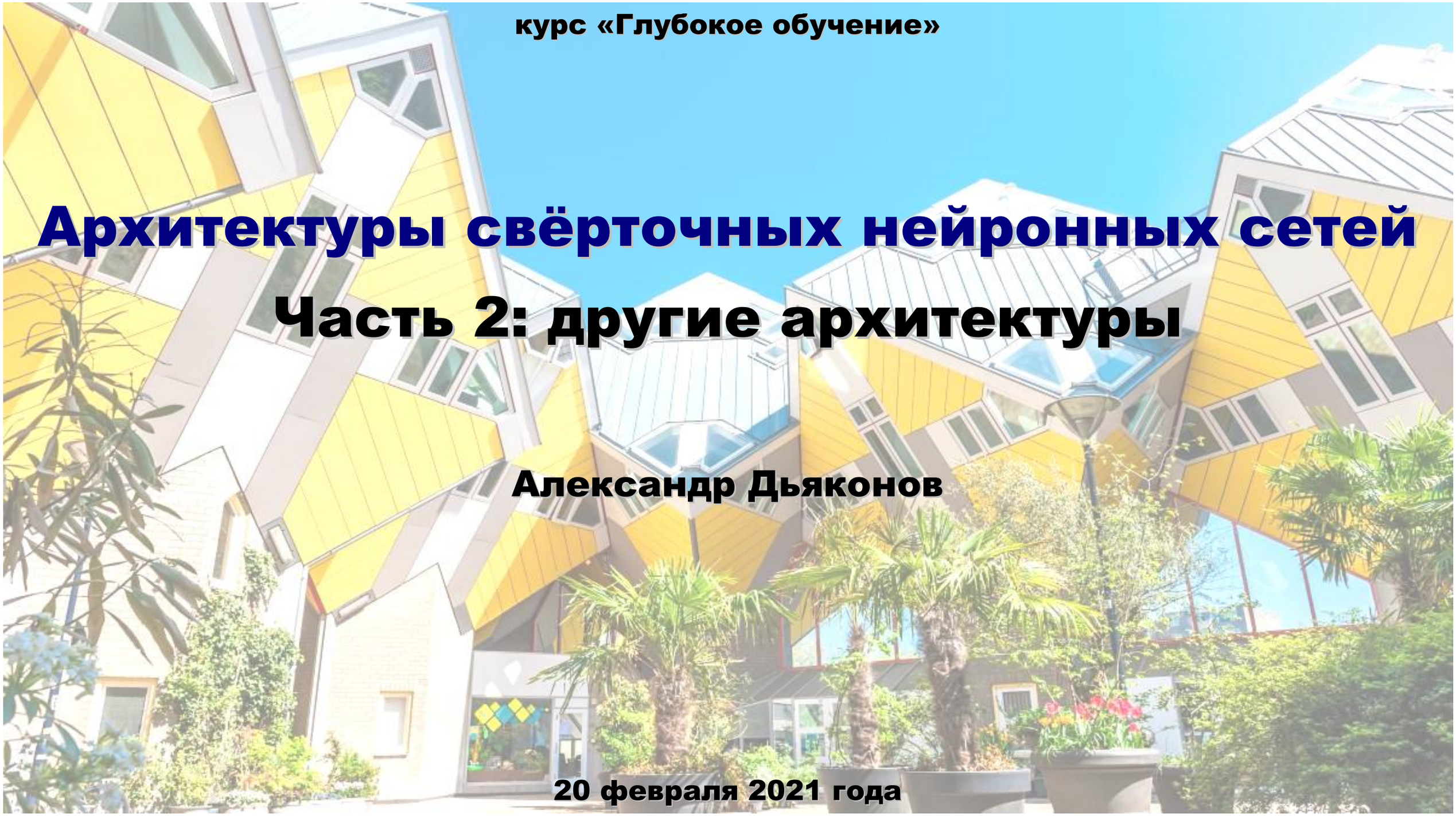


курс «Глубокое обучение»

Архитектуры свёрточных нейронных сетей

Часть 2: другие архитектуры

Александр Дьяконов

20 февраля 2021 года

План

Network in Network (NiN)

Deep Networks with Stochastic Depth

FractalNet, Fractal of FractalNet

DenseNets

ResNeXt, MultiResNet, PolyNet

HyperNets

EfficientNet, MobileNet, SqueezeNet, ShuffleNet

FBNet (+NAS)

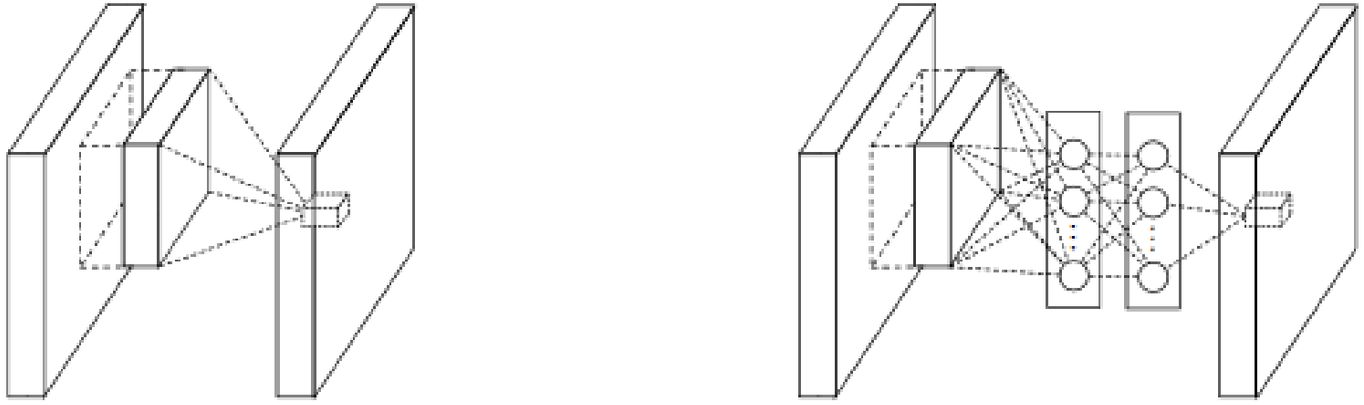
WideResNets

RevNet, iRevNet

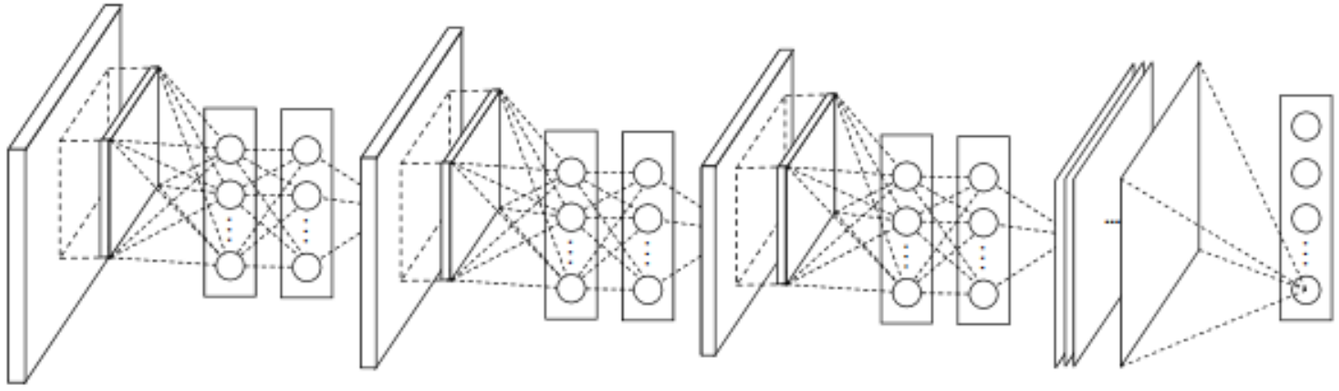
NFNets

ConvNeXt

Какие архитектуры ещё надо знать… Network in Network (NiN)



(a) Linear convolution layer (b) Mlpconv layer



полносвязность ~ свёртки 1×1 внутри свёртки

глобальный пулинг

Min Lin « Network in Network (NiN)» 2014, https://arxiv.org/pdf/1312.4400.pdf

Deep Networks with Stochastic Depth

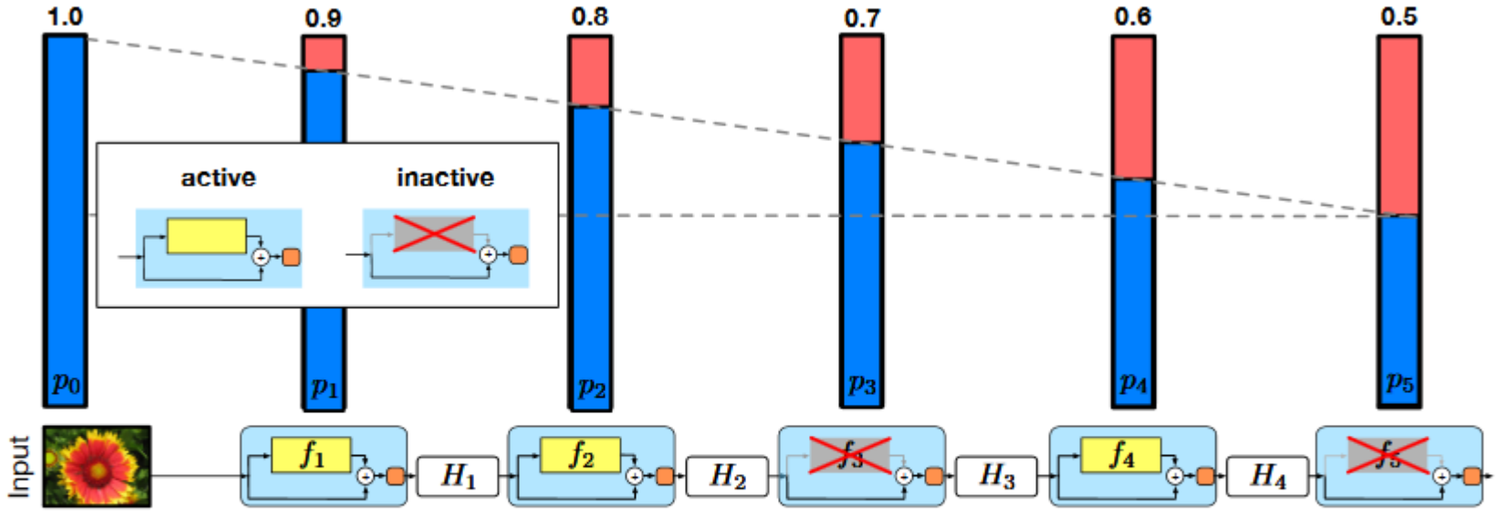


Fig. 2. The linear decay of p_ℓ illustrated on a ResNet with stochastic depth for $p_0 = 1$ and $p_L = 0.5$. Conceptually, we treat the input to the first ResBlock as H_0, which is always active.

- **Во время обучения: случайно удаляем подмножество слоёв**

(используем менее глубокую сеть во время обучения)

- **«Прокидывание» тождественной функции**

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, Kilian Q. Weinberger «Deep Networks with Stochastic Depth» https://arxiv.org/abs/1603.09382

FractalNet: Ultra-Deep Neural Networks without Residuals

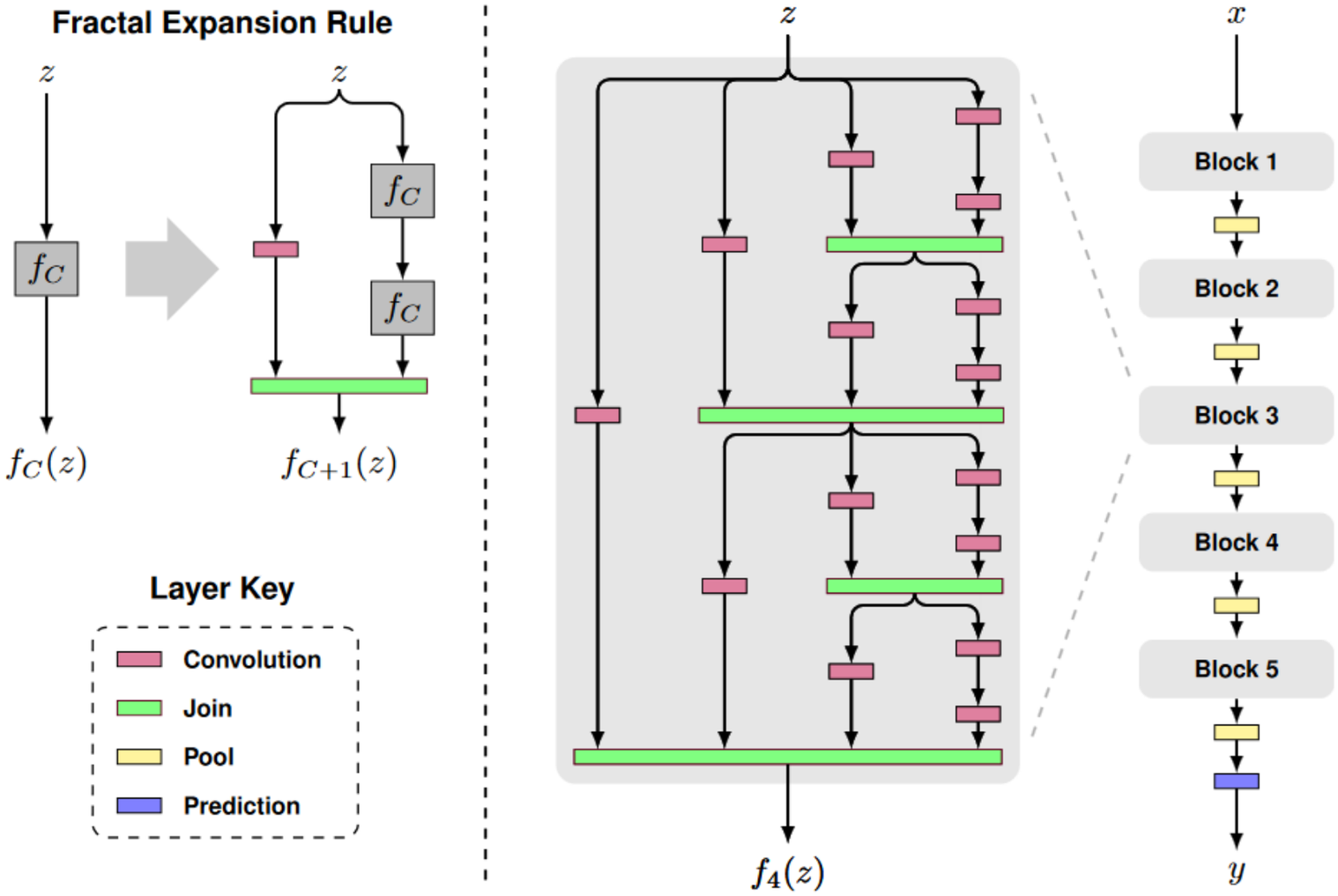


Figure 1: **Fractal architecture.** *Left:* A simple expansion rule generates a fractal architecture with C intertwined columns. The base case, $f_1(z)$, has a single layer of the chosen type (*e.g.* convolutional) between input and output. Join layers compute element-wise mean. *Right:* Deep convolutional networks periodically reduce spatial resolution via pooling. A fractal version uses f_C as a building block between pooling layers. Stacking B such blocks yields a network whose total depth, measured in terms of convolution layers, is $B \cdot 2^{C-1}$. This example has depth 40 ($B = 5$, $C = 4$).

Фрактальная архитектура с короткими и длинными связями

[Gustav Larsson 2017 https://arxiv.org/abs/1605.07648]

FractalNet: Ultra-Deep Neural Networks without Residuals

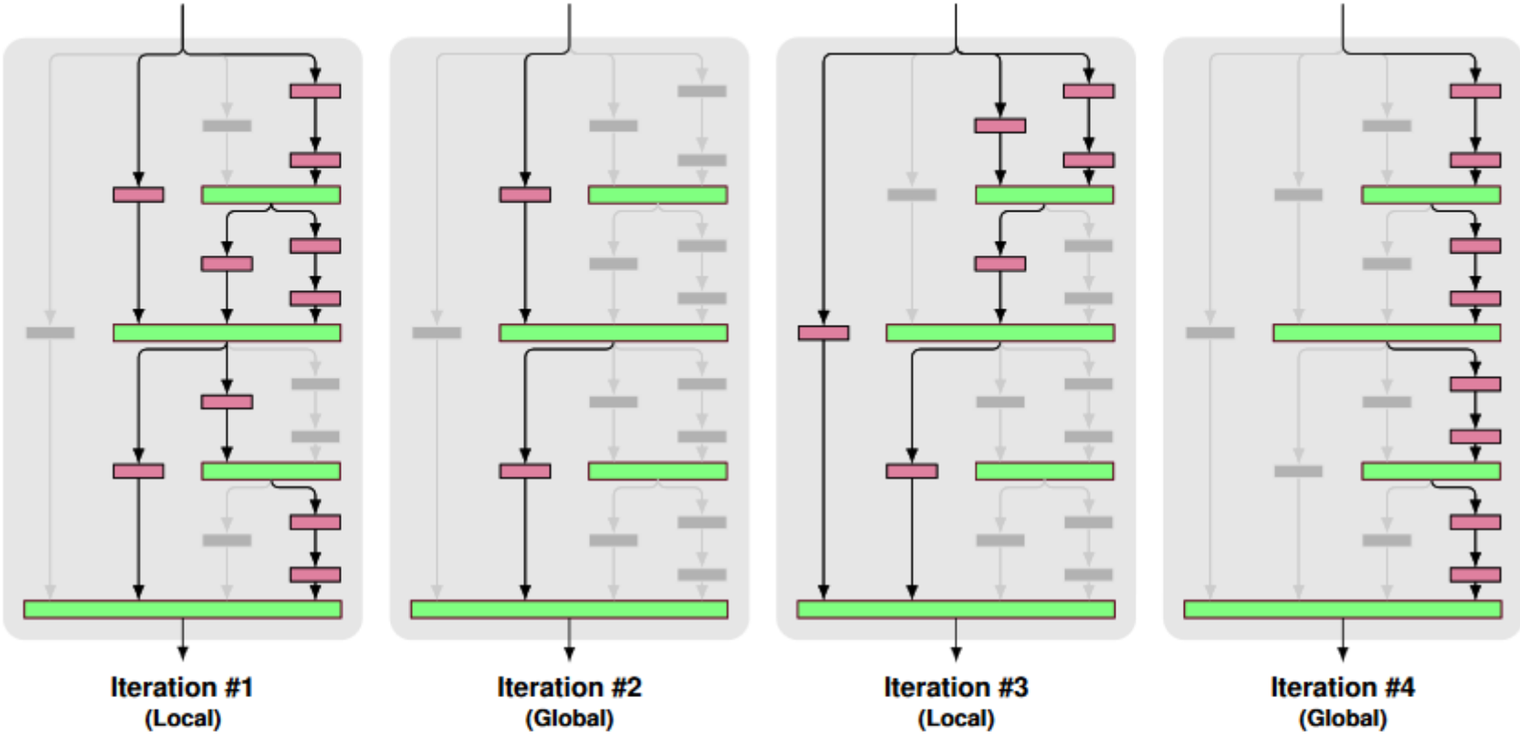
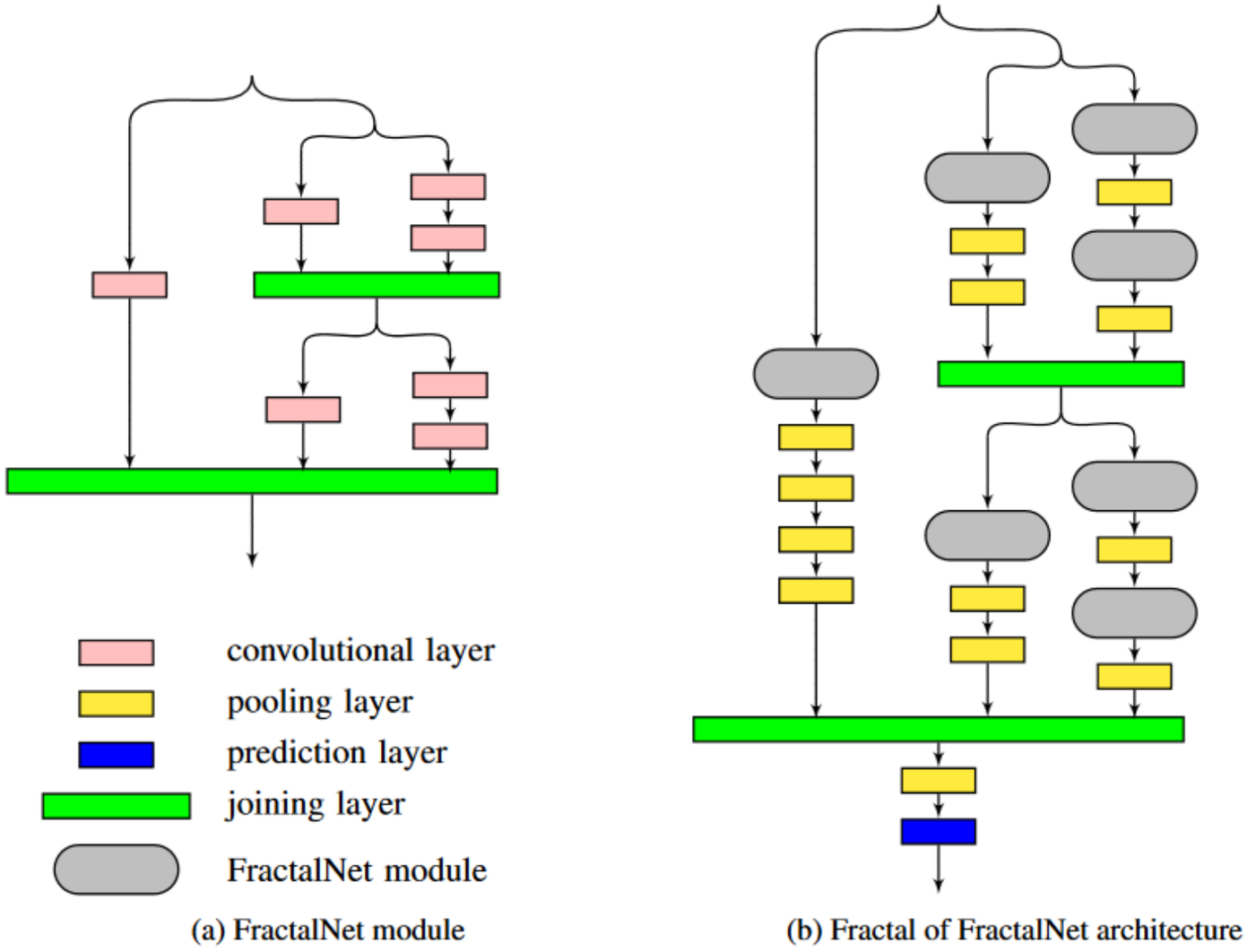


Figure 2: **Drop-path.** A fractal network block functions with some connections between layers disabled, provided some path from input to output is still available. Drop-path guarantees at least one such path, while sampling a subnetwork with many other paths disabled. During training, presenting a different active subnetwork to each mini-batch prevents co-adaptation of parallel paths. A global sampling strategy returns a single column as a subnetwork. Alternating it with local sampling encourages the development of individual columns as performant stand-alone subnetworks.

Обучение со случайным выбрасыванием связей

Fractal of FractalNet (FoF)



(a) FractalNet module (b) Fractal of FractalNet architecture

Figure 1: (a) The FractalNet module and (b) the FoF architecture.

Leslie N. Smith, Nicholay Topin, Deep Convolutional Neural Network Design Patterns // https://arxiv.org/abs/1611.00847

Densely Connected Convolutional Networks (DenseNets)

Блоки, в которых слой соединён с каждым последующим

Обычная сеть: $z_i = H_i(z_{i-1})$

где z_i **выход i-го слоя.**

ResNet: $z_i = H_i(z_{i-1}) + z_{i-1}$

DensNet: $z_i = H_i([z_{i-1}, z_{i-2}, \ldots, z_0])$

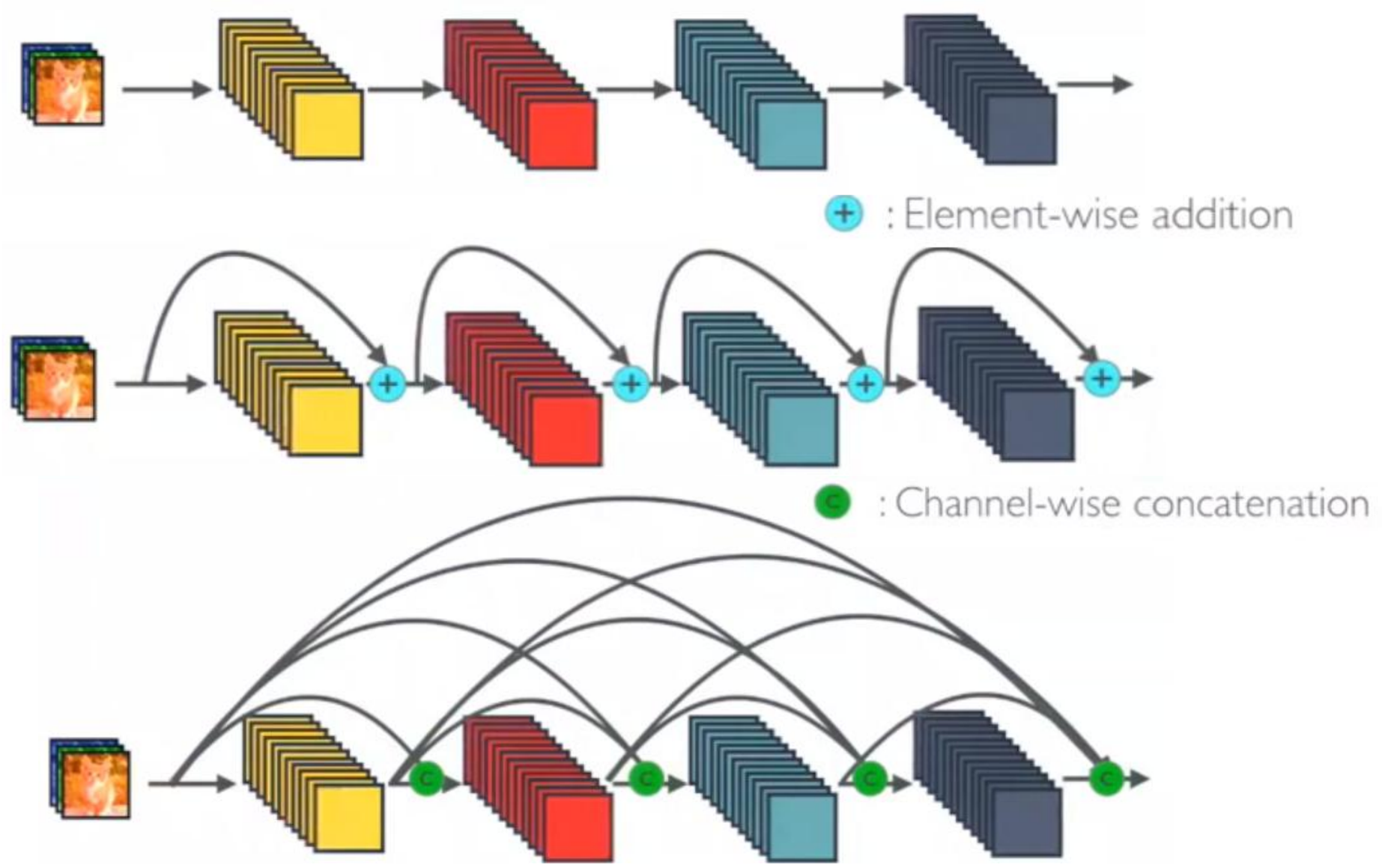
H = BN + ReLU + convolution + dropout

число признаков линейно вырастает…

Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger «Densely Connected Convolutional Networks» https://arxiv.org/abs/1608.06993

Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens van der Maaten, Kilian Q. Weinberger «Convolutional Networks with Dense Connectivity» // https://arxiv.org/abs/2001.02394

Nets → ResNets →DenseNets



Densely Connected Convolutional Networks (DenseNets)

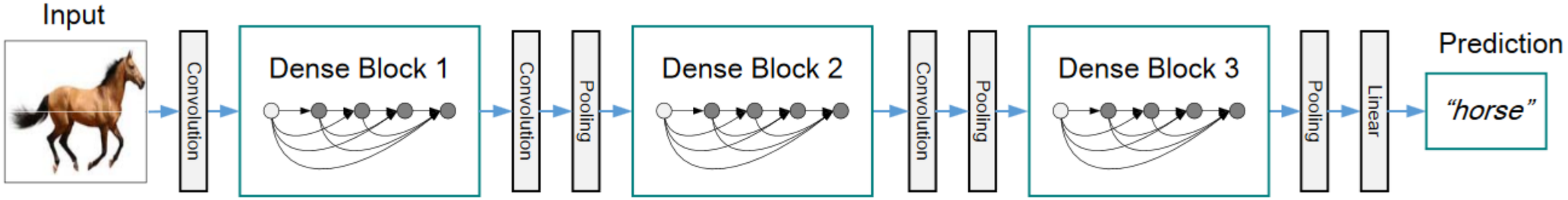


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

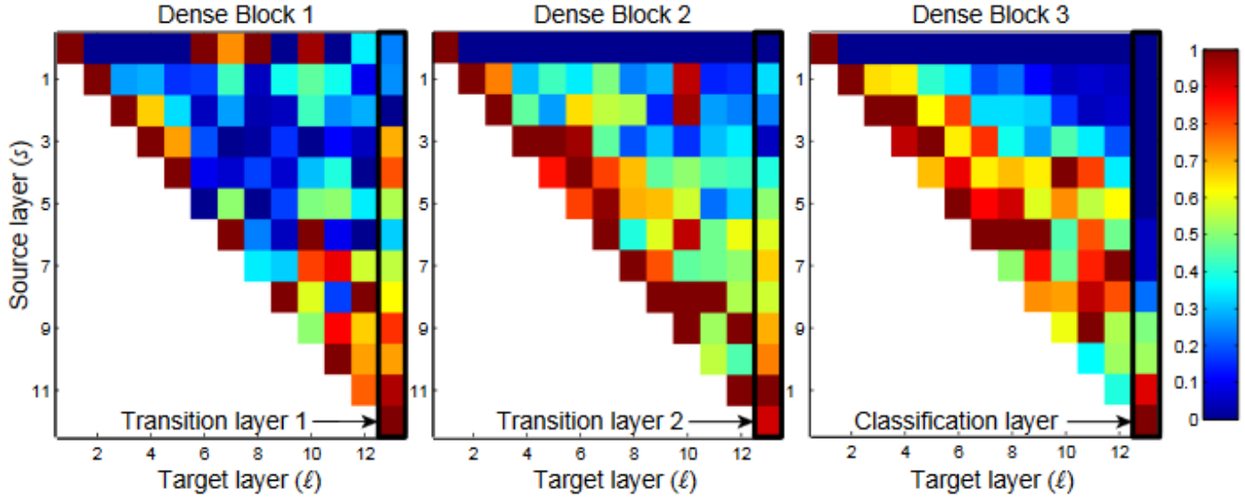


Figure 5: The average absolute filter weights of convolutional layers in a trained DenseNet. The color of pixel (s, ℓ) encodes the average $L1$ norm (normalized by number of input feature-maps) of the weights connecting convolutional layer s to ℓ within a dense block. Three columns highlighted by black rectangles correspond to two transition layers and the classification layer. The first row encodes weights connected to the input layer of the dense block.

Минутка кода: DenseNet

```
def conv_block(input_channels, num_channels):
    return nn.Sequential(nn.BatchNorm2d(input_channels),
                         nn.ReLU(),
                         nn.Conv2d(input_channels, num_channels, kernel_size=3, padding=1))

class DenseBlock(nn.Module):
    def __init__(self, num_convs, input_channels, num_channels):
        super(DenseBlock, self).__init__()
        layer = []
        for i in range(num_convs):
            layer.append(conv_block(num_channels * i + input_channels, num_channels))
        self.net = nn.Sequential(*layer)

    def forward(self, X):
        for blk in self.net:
            Y = blk(X)
            # Concatenate the input and output of each block on the channel dimension
            X = torch.cat((X, Y), dim=1)
        return X
```

https://d2l.ai/chapter_convolutional-modern/densenet.html

ResNeXt

такое же число параметров как в ResNet, но разнести их по cardinality=32 разным путям

тут блок – conv + BN + ReLU – используется bottleneck!!!

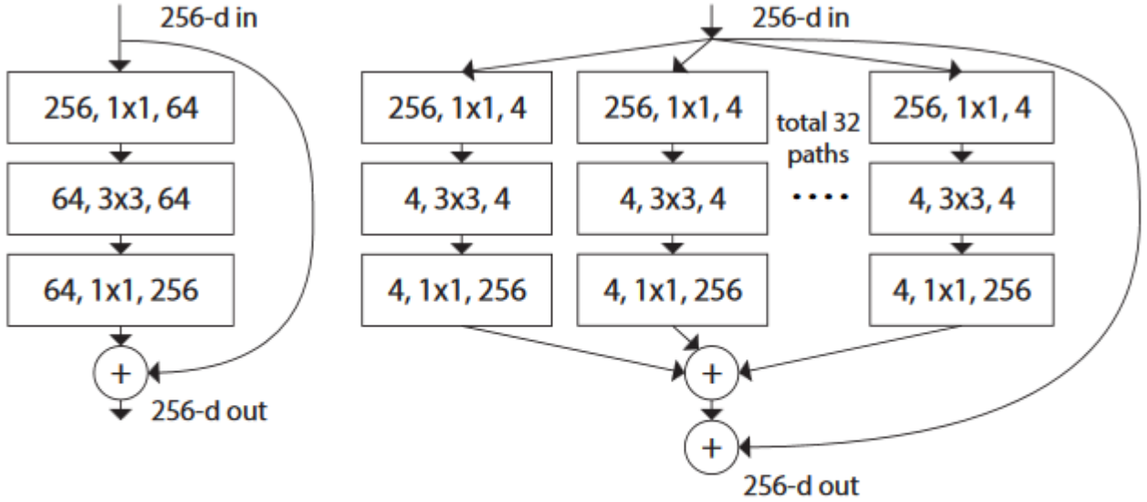


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He «Aggregated Residual Transformations for Deep Neural Networks» // https://arxiv.org/abs/1611.05431

ResNeXt

Похожие блоки: (a) ResNeXt Block, (b) Inception-ResNet Block, (c) Grouped Convolution

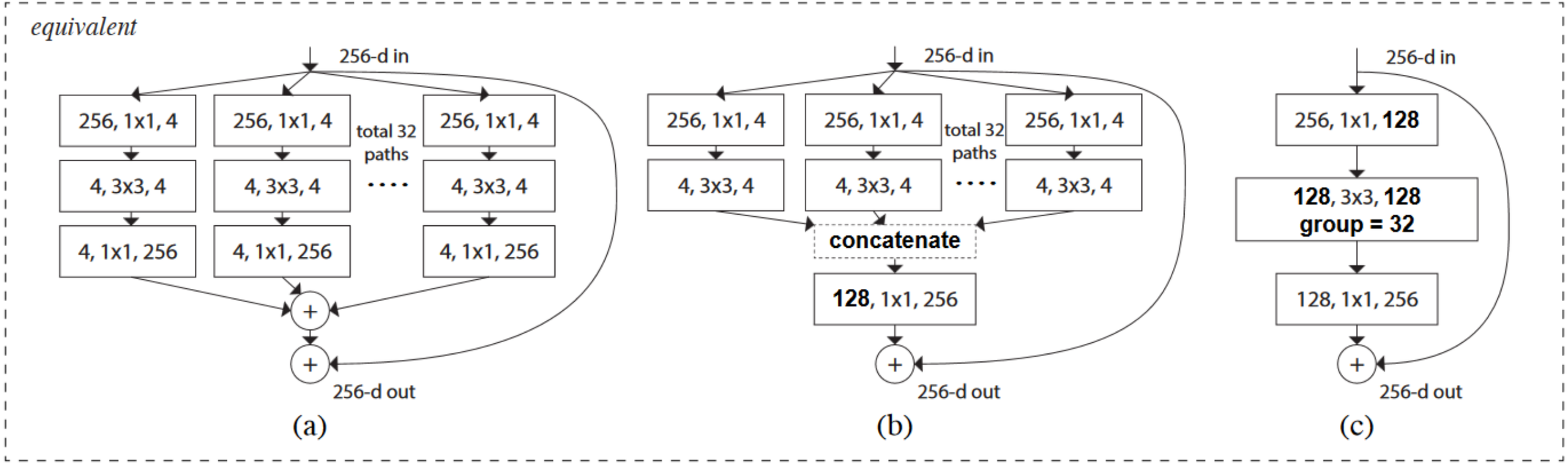


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivalent to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [24]. Notations in **bold** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

Ещё разделения на ветви: ResNeXt, MultiResNet, PolyNet

Стратегия «split-transform-aggregate»

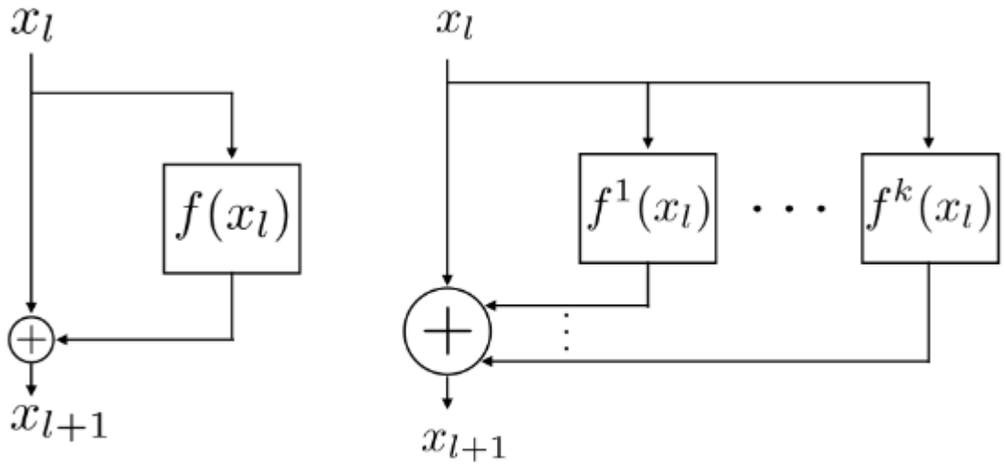


Fig. 2: A residual block (left) versus a multi-residual block (right).

Masoud Abdi, Saeid Nahavandi «Multi-Residual Networks: Improving the Speed and Accuracy of Residual Networks» // https://arxiv.org/abs/1609.05672

Ещё разделения на ветви: ResNeXt, MultiResNet, PolyNet

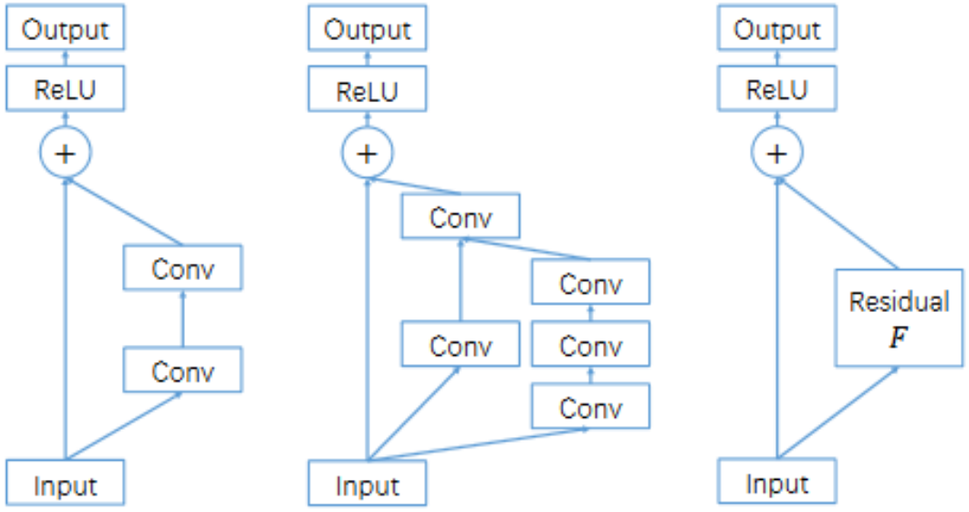
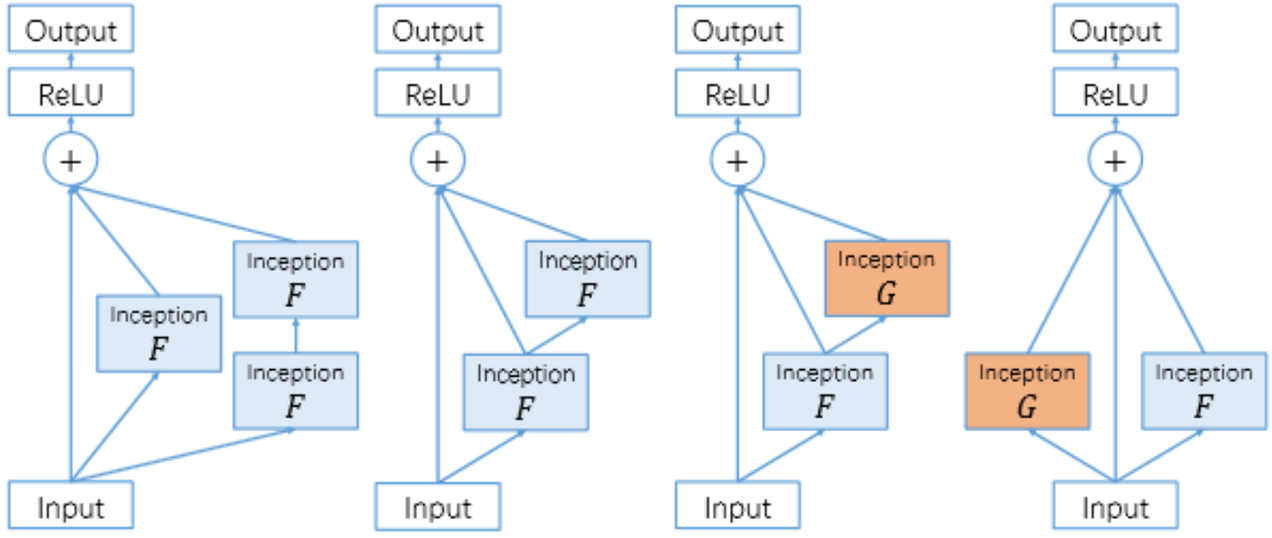


Figure 3: **Left**: residual unit of ResNet [9]. **Middle**: type-B Inception residual unit of Inception-ResNet-v2 [25]. **Right**: abstract residual unit structure where the residual block is denoted by F.

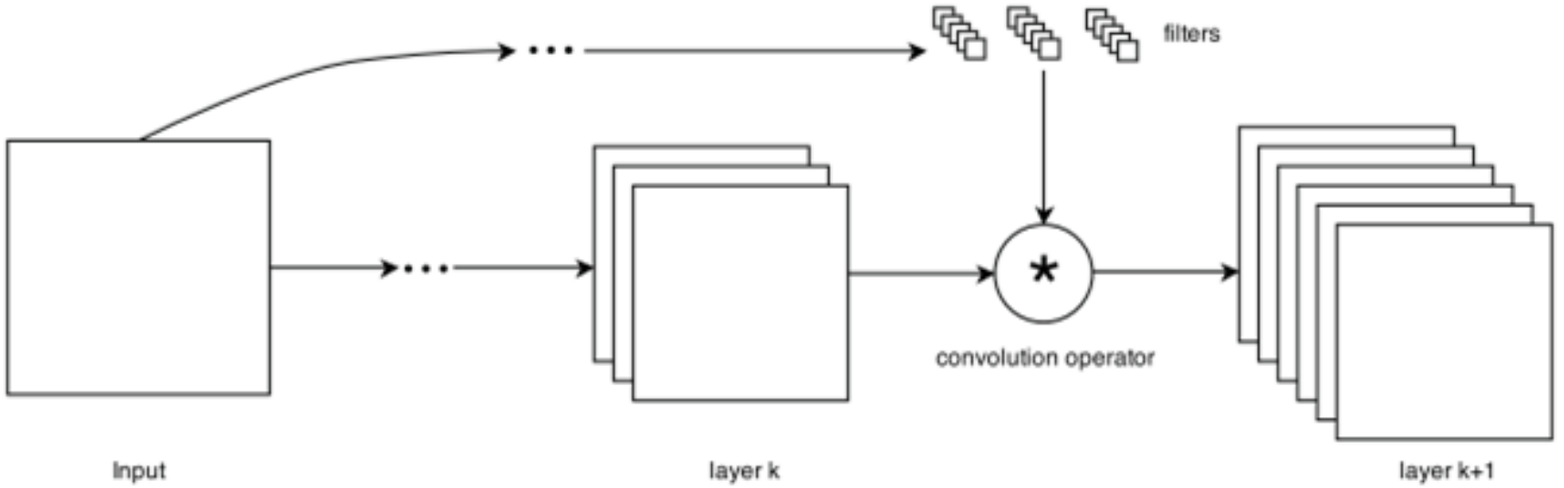


(a) *poly-2* (b) *poly-2* (c) *mpoly-2* (d) *2-way*

Figure 4: Examples of PolyInception structures.

Xingcheng Zhang, Zhizhong Li, Chen Change Loy, Dahua Lin «PolyNet: A Pursuit of Structural Diversity in Very Deep Networks» // https://arxiv.org/abs/1611.05725

Динамические свёртки: HyperNets



динамическая свёртка – результат действия мини-сети «Dynamic Convolutional Layer»
потом эта идея – SENet, Attention

Klein, B., Wolf, L., & Afek, Y. «A dynamic convolutional layer for short range weather prediction» // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

Динамические свёртки: Dynamic Filter Networks

Dynamic local filtering – свёртка может зависеть от позиции

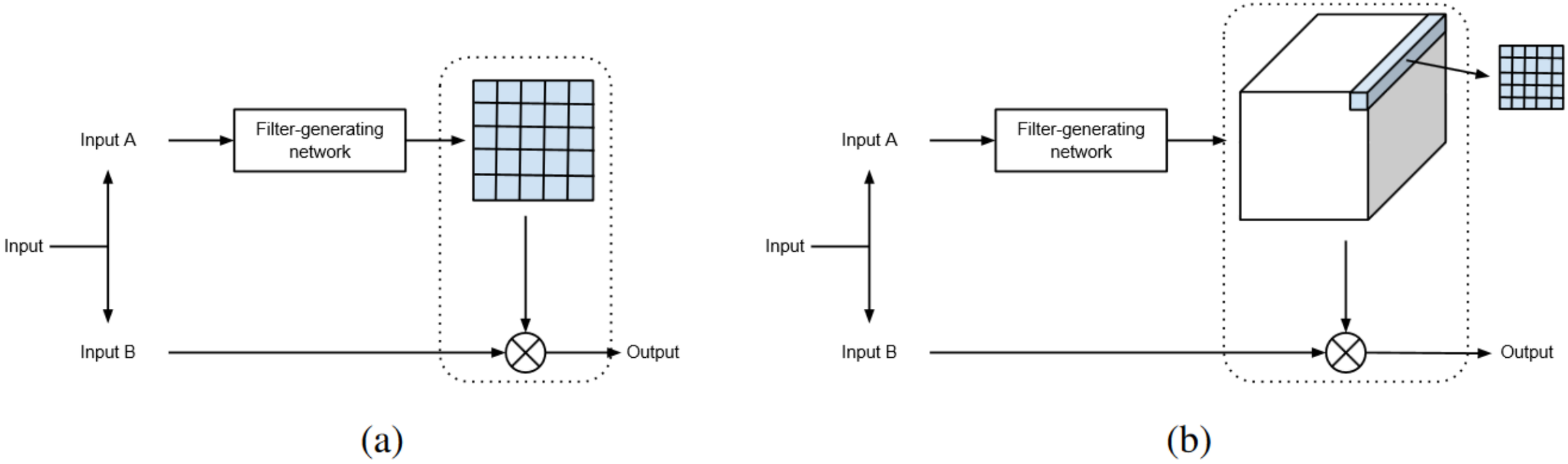


Figure 2: *Left:* Dynamic convolution: the filter-generating network produces a single filter that is applied convolutionally on I_B. *Right:* Dynamic local filtering: each location is filtered with a location-specific dynamically generated filter.

Jia, X., De Brabandere, B., Tuytelaars, T., Gool, L. V. «Dynamic filter networks» // Advances in Neural Information Processing Systems, 2016.

Динамические свёртки: HyperNets

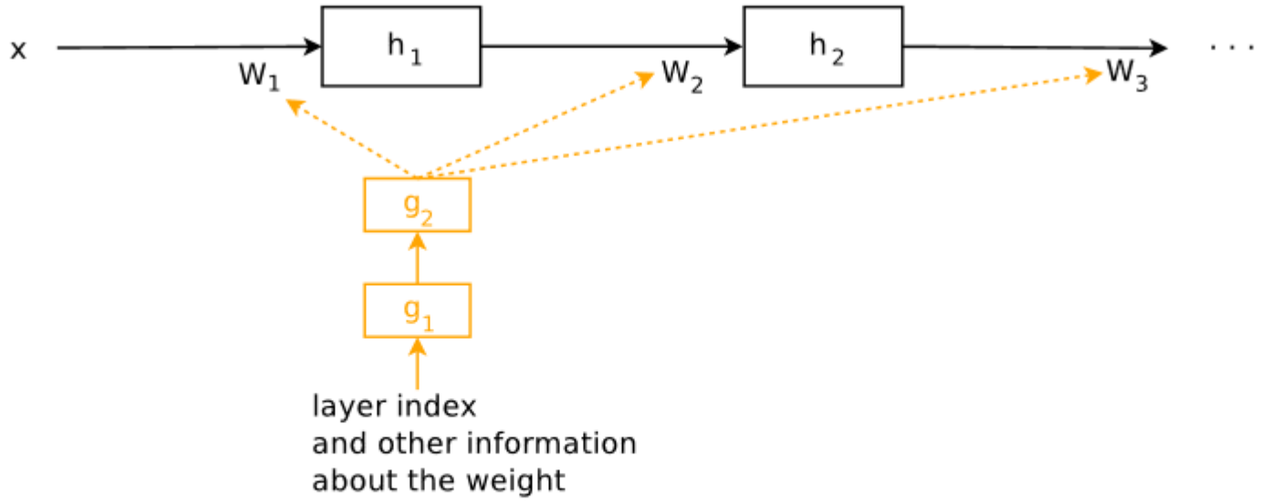


Figure 1: A hypernetwork generates the weights for a feedforward network. Black connections and parameters are associated the main network whereas orange connections and parameters are associated with the hypernetwork.

Гиперсеть – маленькая сеть, которая определяет параметры основной сети

Можно получать «non-shared weights for LSTM»

Ha, D., Dai, A., and Le, Q. V. «Hypernetworks» // https://arxiv.org/pdf/1609.09106.pdf

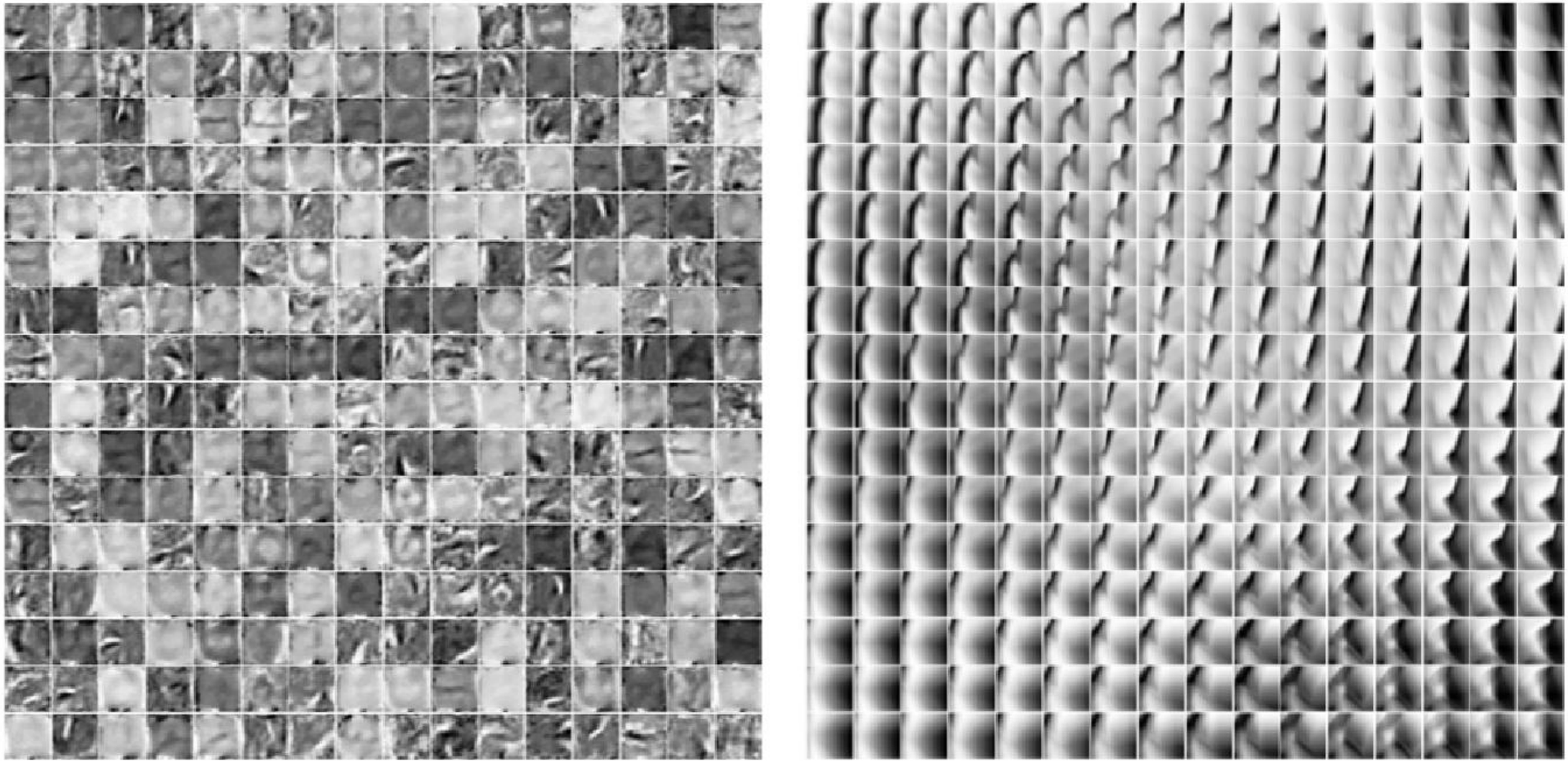
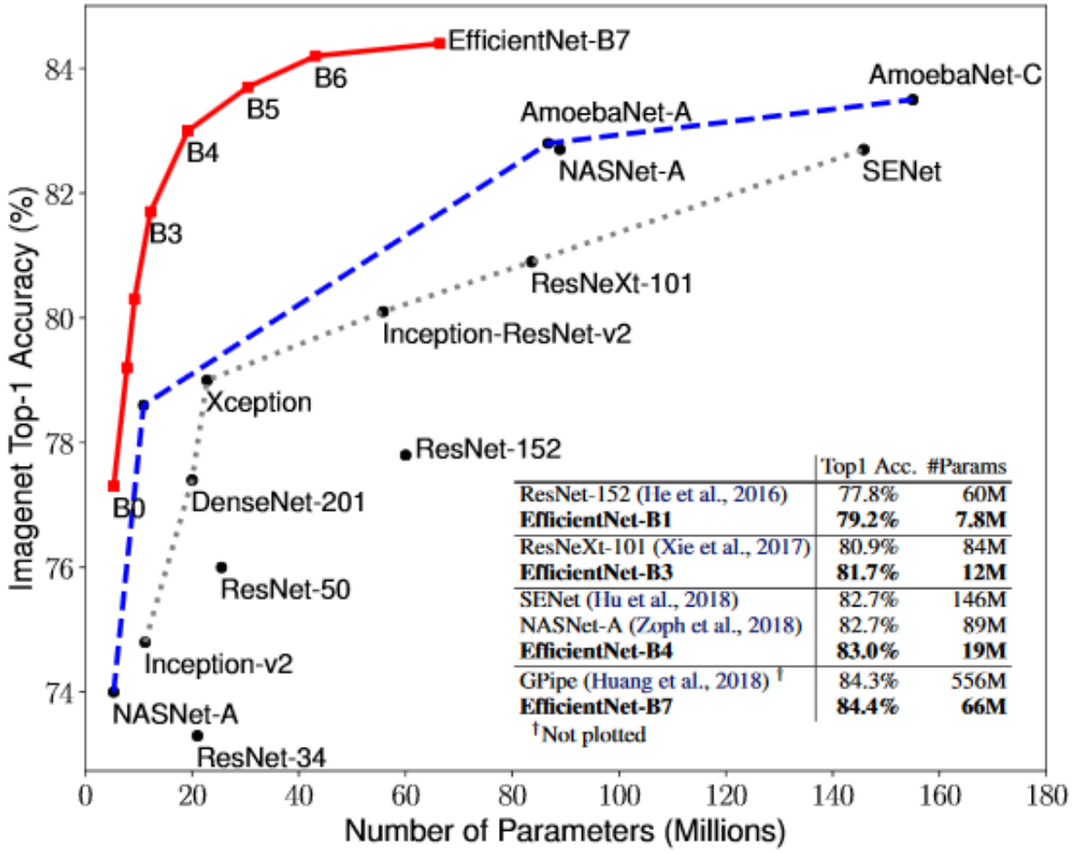


Figure 8: Filters learned to classify MNIST digits in a fully connected network (left). Filters learned by a hypernetwork (right).

но тут качество у гиперсети получилось хуже

EfficientNet: Масштабирование (scaling up) моделей



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**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 84.4% 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.

https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet

Mingxing Tan and Quoc V. Le «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks» ICML 2019. https://arxiv.org/abs/1905.11946

EfficientNet: Масштабирование (scaling up) моделей

- **увеличение глубины** (L = layers)
- **увеличение числа каналов** (C = channels)
- **увеличение разрешения** (H×W входа)
- **compound scaling method** (**увеличение всего с опред. пропорциями**)

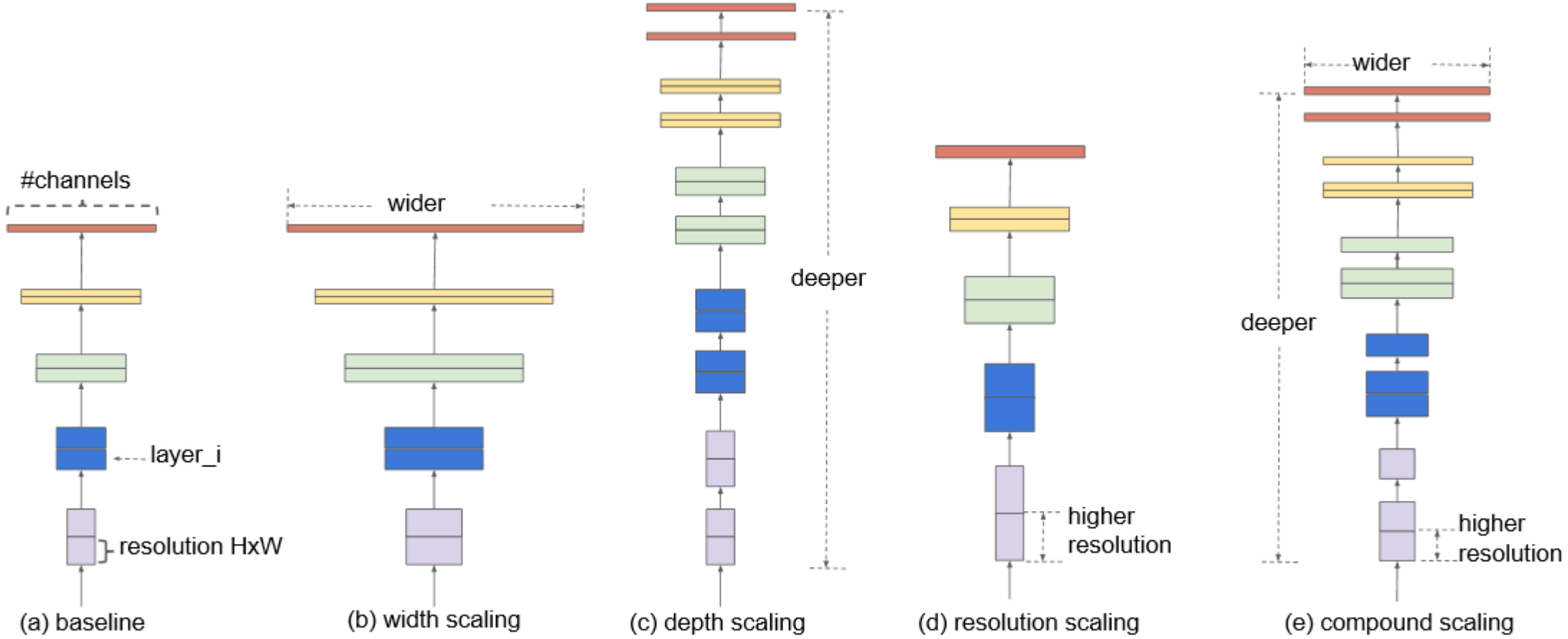


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

EfficientNet: Масштабирование (scaling up) моделей

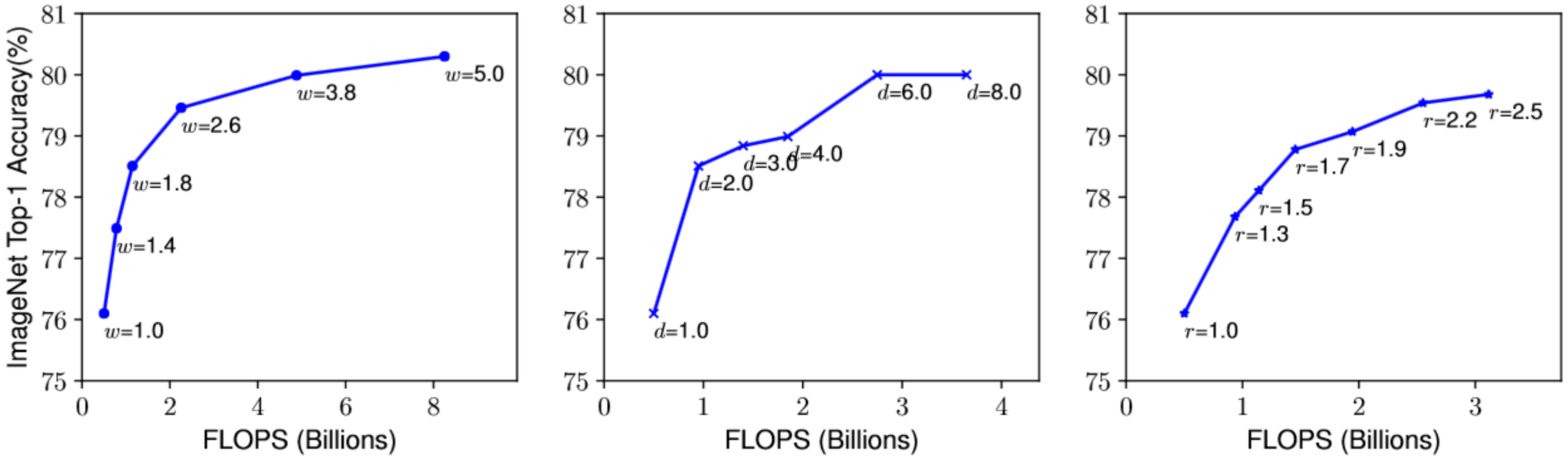


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

разные масштабирования не являются независимыми
нужно учитывать, что меняется и время обучения сети!

EfficientNet: compound scaling method

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

ограничение = 2 для ограничения числа операций (конкретно в статье)

изменение в φ приводят к увеличению FLOPS на 2^φ

α, β, γ находятся перебором

Table 3. **Scaling Up MobileNets and ResNet.**

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

EfficientNet: compound scaling method

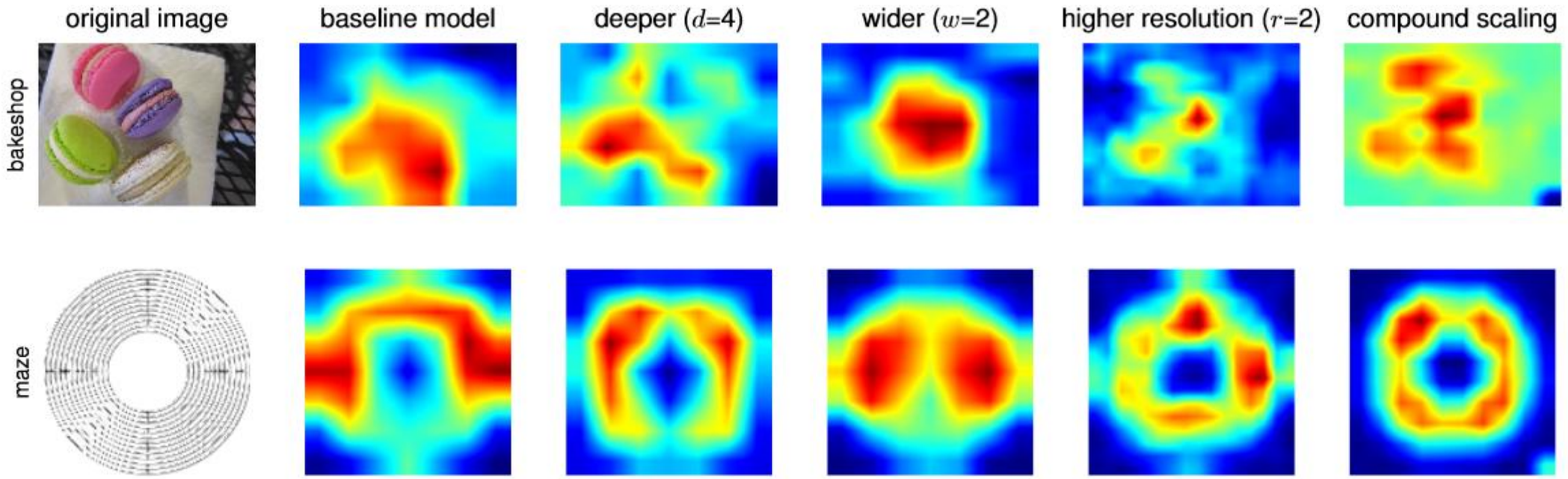


Figure 7. **Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods-** Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details. Model details are in Table 7.

Мобильные архитектуры: MobileNet

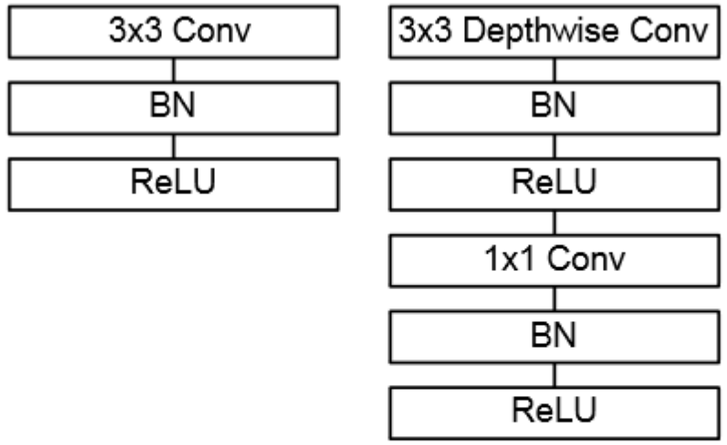


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

использование Depthwise separable convolution (уже было)

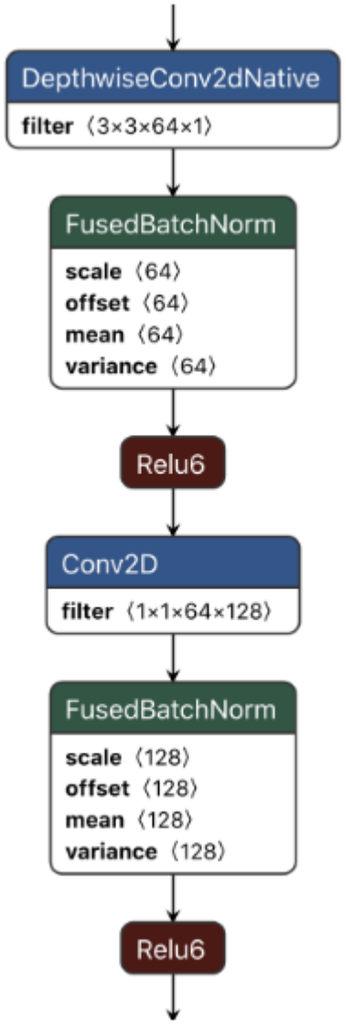
нет MaxPool, вместо этого в свёртках stride=2

Andrew G. Howard et. al. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications» https://arxiv.org/pdf/1704.04861.pdf

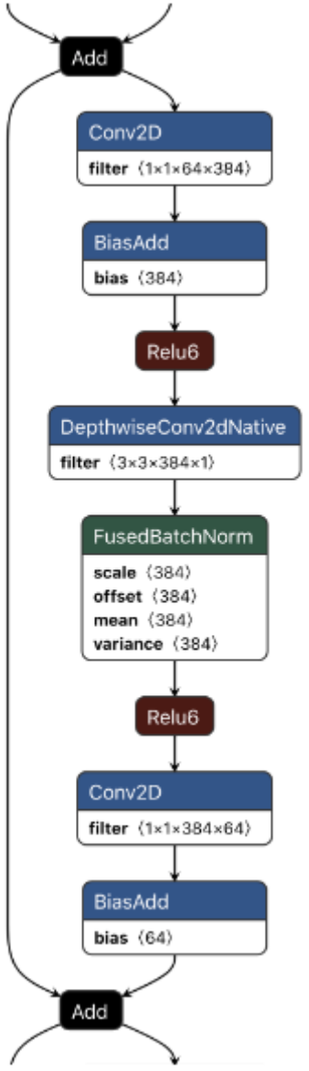
https://machinethink.net/blog/mobile-architectures/

Мобильные архитектуры: MobileNet v2

v1



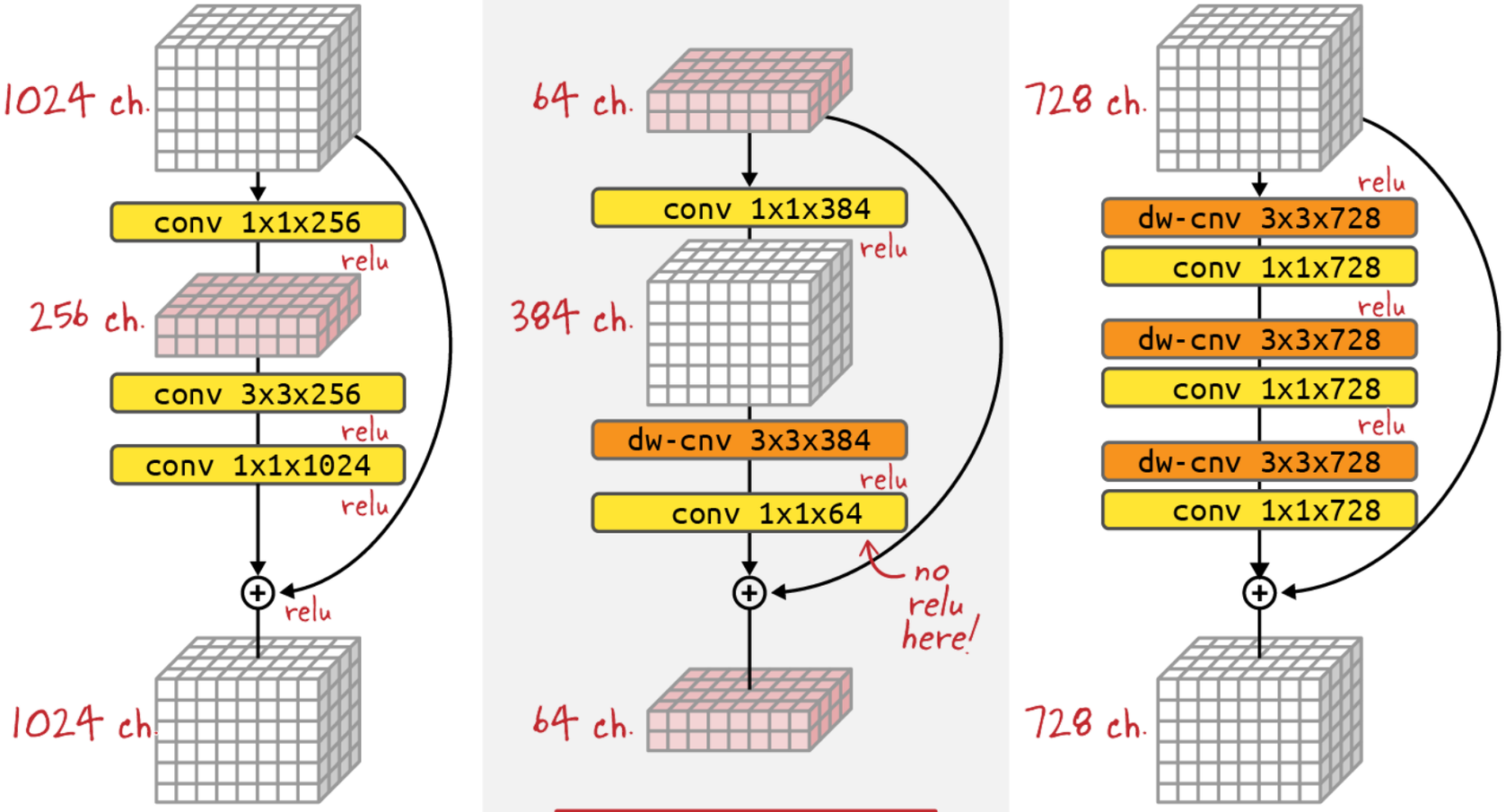
v2



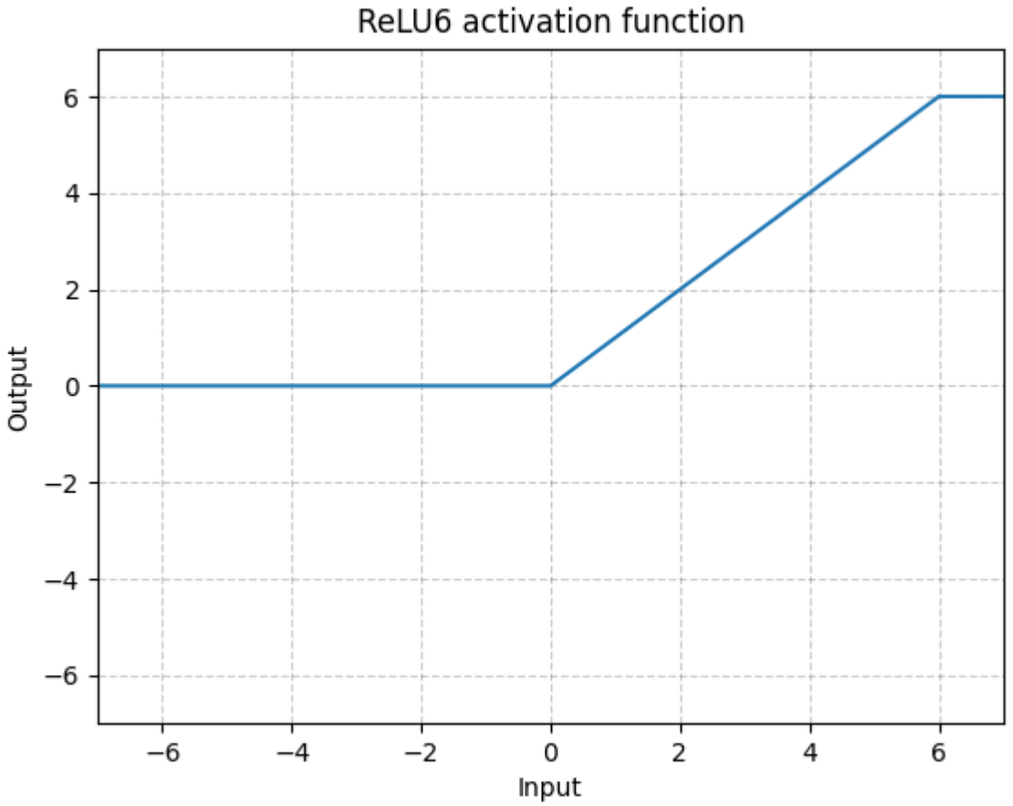
depthwise convolution в середине + прокидывание

тут увеличивается число признаков, потом уменьшается (в SqueezeNet наоборот)

Мобильные архитектуры: MobileNet v2 – Inverted residual bottlenecks



ReLU6



https://paperswithcode.com/method/relu6

Мобильные архитектуры: MobileNet v3

переделаны дорогие слои

ReLU6 $\rightarrow$ Hard Swish (на глубоких слоях)

```
h_swish(x) = x * ReLU6(x + 3) / 6
```

squeeze-and-excitation modules (SE)

Мобильные архитектуры: MobileNet v3

Изменена голова сети:

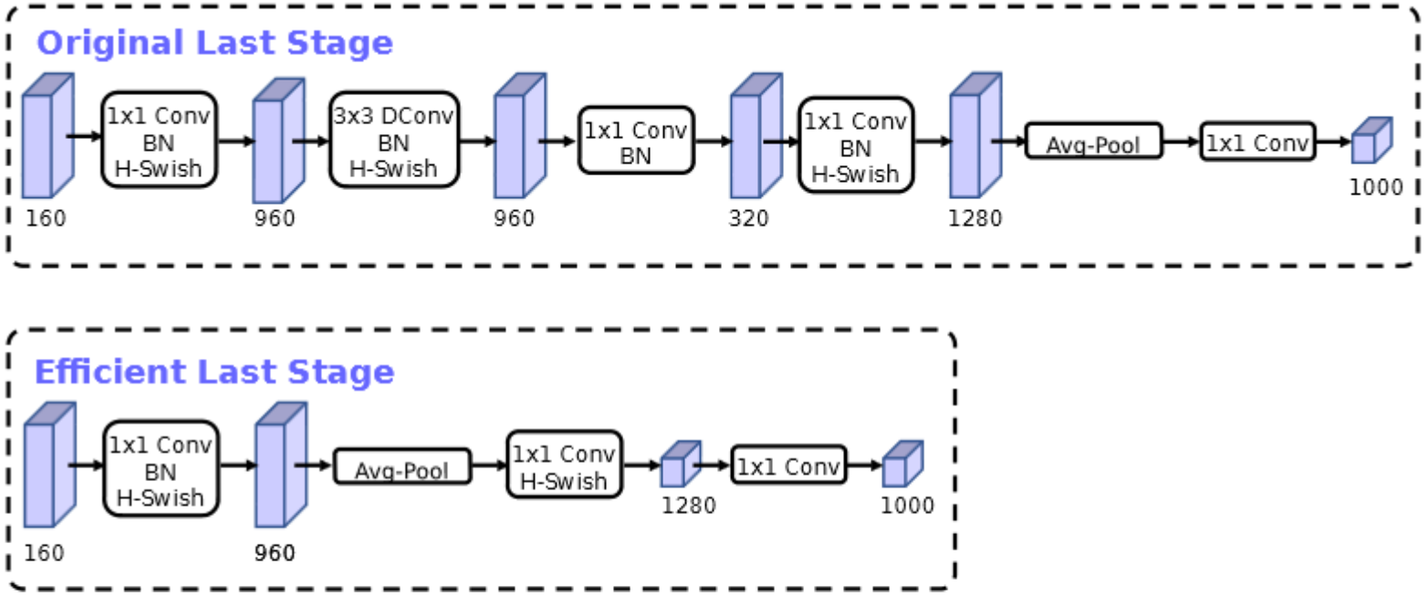
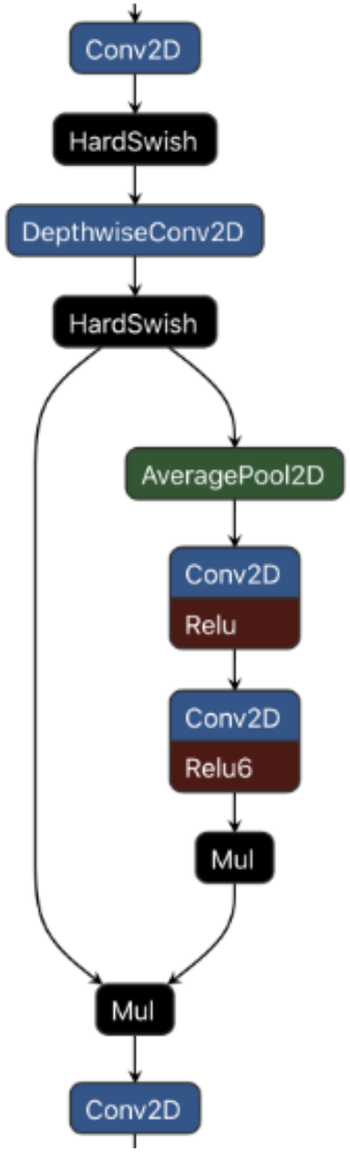


Figure 5. Comparison of original last stage and efficient last stage. This more efficient last stage is able to drop three expensive layers at the end of the network at no loss of accuracy.

Andrew Howard, et al. «Searching for MobileNetV3» https://arxiv.org/abs/1905.02244

Мобильные архитектуры: MobileNet v3



Мобильные архитектуры: SqueezeNet – маленькая сеть с качеством AlexNet

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

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%

в 3 раза быстрее

Fully Convolutional Network (FCN) – нет полносвязных слоёв

Forrest N. Iandola «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size»2017 https://arxiv.org/abs/1602.07360

Мобильные архитектуры: SqueezeNet – маленькая сеть с качеством AlexNet

продолжает идею «модульной архитектуры» ~ Inception

тут тоже будет путь 1×1-свёрток и 3×3-свёрток

но модули упростили

замена 3×3-свёрток на 1×1 – squeeze – см. дальше

уменьшаем в 9 раз число параметров

уменьшаем число каналов перед 3×3-свёртками

опять же – меньше параметров

делать побольше stride ближе к концу сети

есть гипотеза, что повышает качество

https://machinethink.net/blog/mobile-architectures/

Мобильные архитектуры: SqueezeNet – маленькая сеть с качеством AlexNet

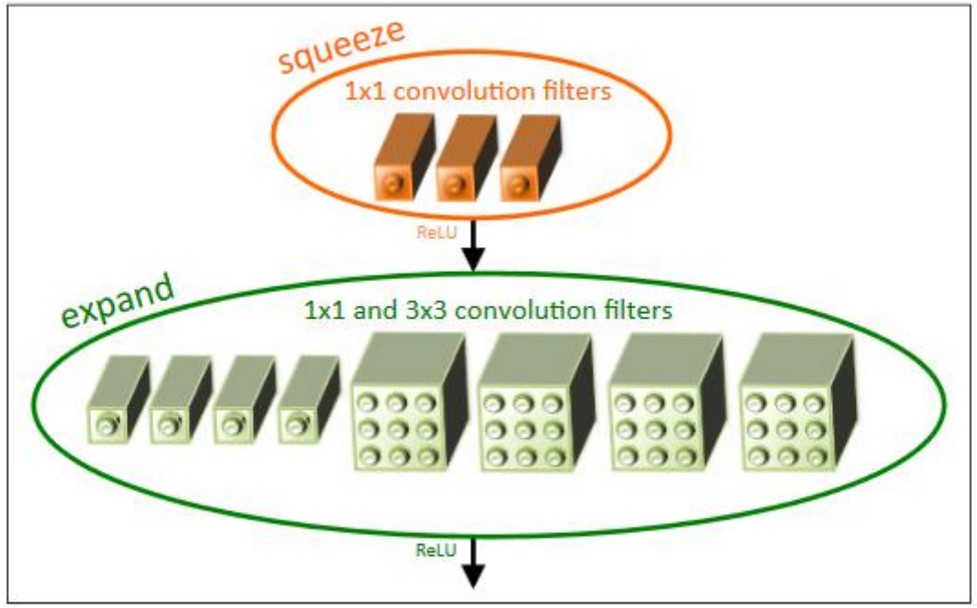


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

см. дальше понятнее иллюстрация

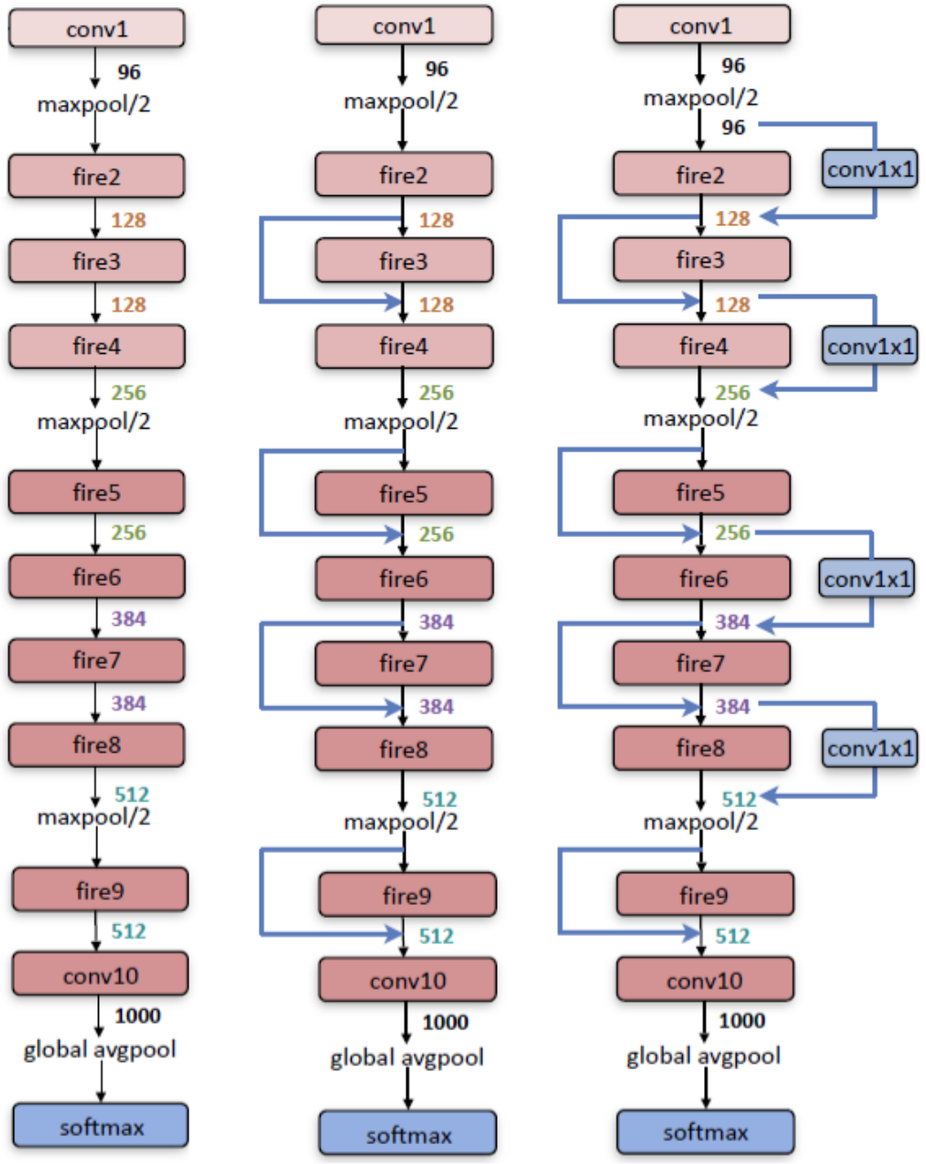
«Fire-модуль», его параметры (здесь – слева):

s1x1 = 3 – число 1×1-свёрток в сжимающей части

e1x1 = 4 – число 1×1-свёрток в расширяющей части

e3x3 = 4 – число 3×3-свёрток в расширяющей части

SqueezeNet – три варианта – SN, SN + simple bypass, SN + complex bypass



Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	**60.4%**	**82.5%**	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

max-pooling stride = 2
после conv1, fire4, fire8, conv10

нет полносвязной части,
вместо этого – AvgPool

Мобильные архитектуры: SqueezeNext

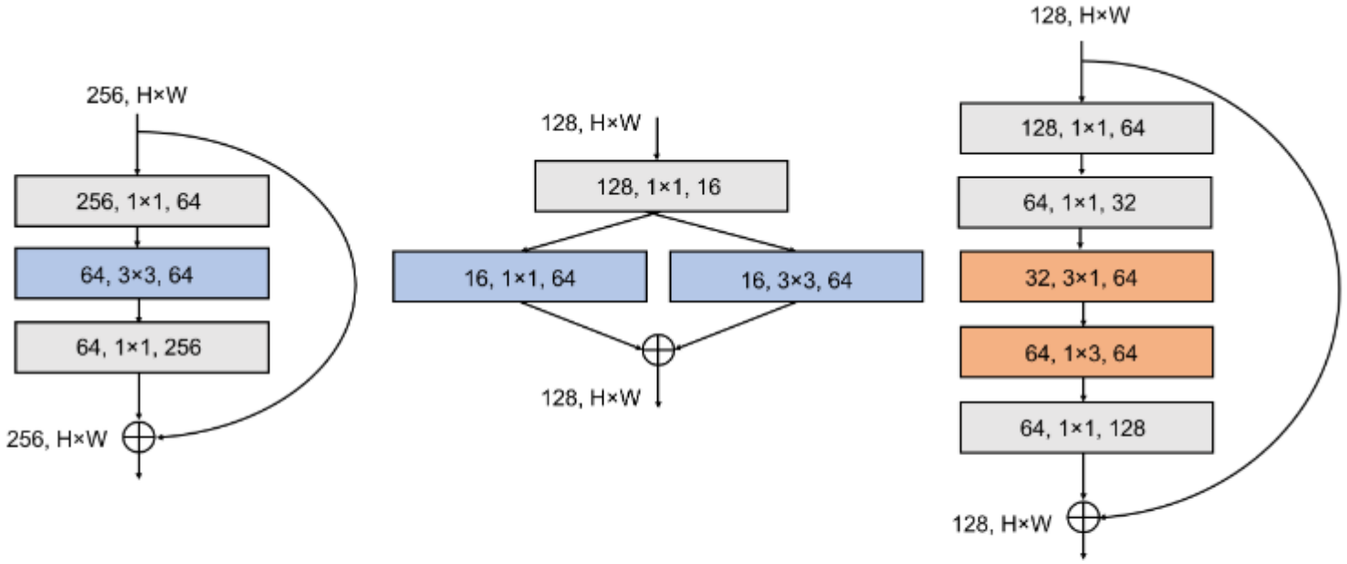


Figure 1: *Illustration of a ResNet block on the left, a SqueezeNet block in the middle, and a SqueezeNext (SqNxt) block on the right. SqueezeNext uses a two-stage bottleneck module to reduce the number of input channels to the 3×3 convolution. The latter is further decomposed into separable convolutions to further reduce the number of parameters (orange parts), followed by a 1×1 expansion module.*

5 свёрточных блоков, каждый с BN и ReLU

блок с 3×1 + 1×3 свёртками чередуется с блоком 1×3 + 3×1

размеры понижаются свёртками со stride=2

сначала стоит 64×7×7-свёртка со stride=2 + MaxPool

и изображение сразу перегоняется в 55×55

Amir Gholami, et al. «SqueezeNext: Hardware-Aware Neural Network Design»

https://arxiv.org/abs/1803.10615

Мобильные архитектуры: SqueezeNext

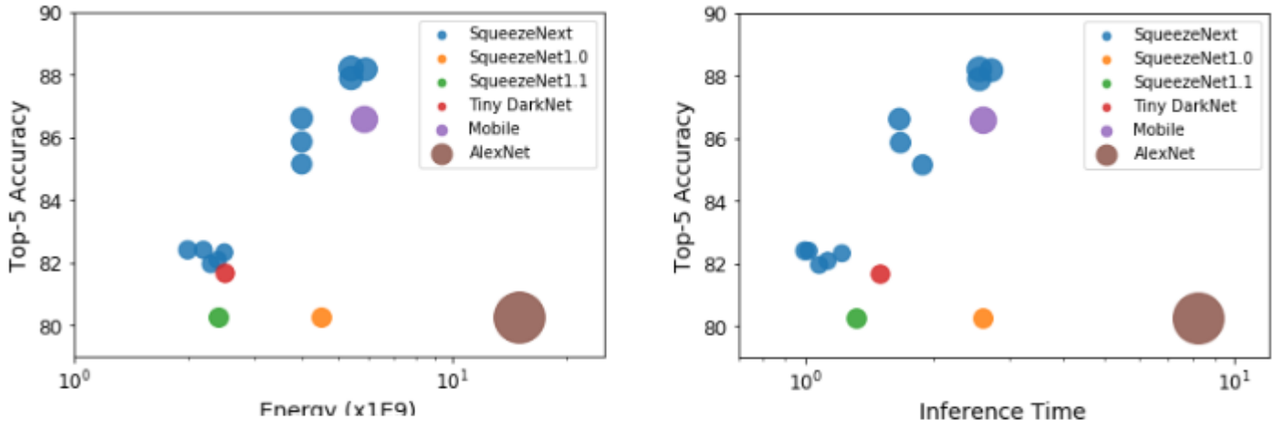


Figure 7: *The spectrum of accuracy versus energy and inference speed for SqueezeNext, SqueezeNet (v1.0 and v1.1), Tiny DarkNet, and MobileNet. SqueezeNext shows superior performance (in both plots higher and to the left is better). The circle areas are proportional to square root of model size for each network.*

Мобильные архитектуры: ShuffleNet – перемешивание каналов

для мобильных устройств (низкая вычислительная мощность)

идея перемешивание каналов (в групповых свёртках)

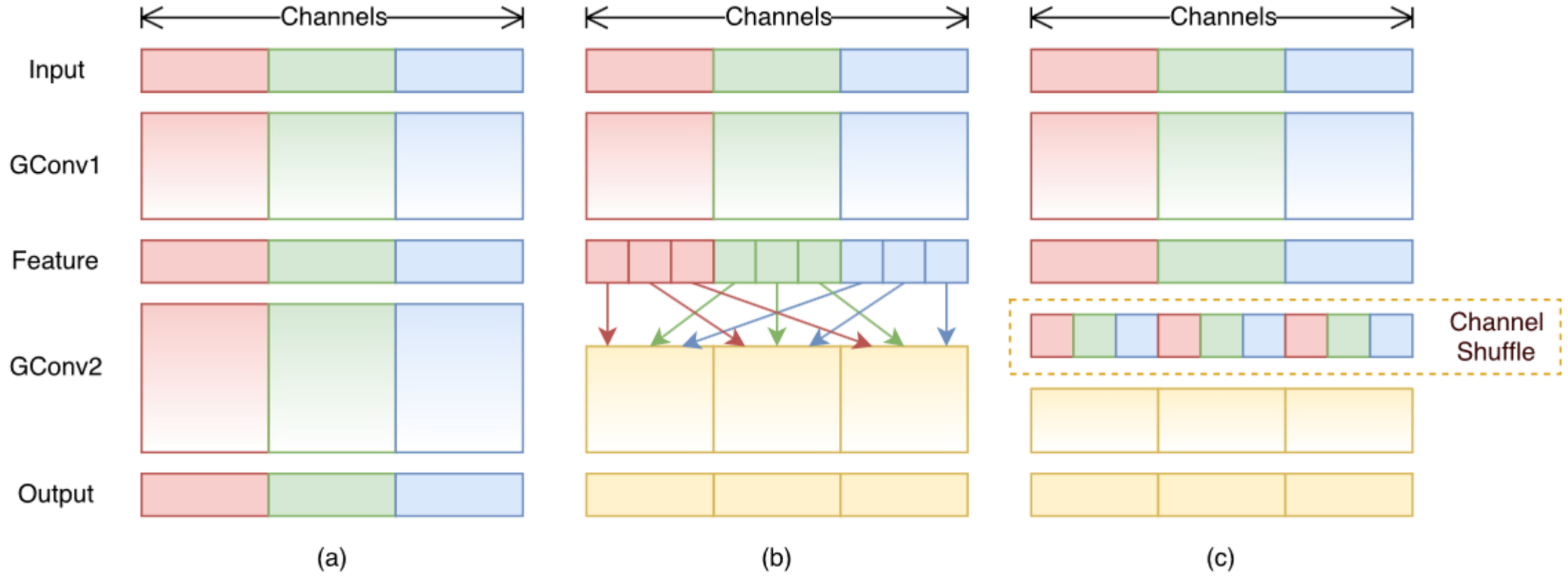


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

Xiangyu Zhang «ShuffleNet: An Extremely Efficient Convolutional Neural Network for MobileDevices»

https://arxiv.org/pdf/1707.01083.pdf

Мобильные архитектуры: ShuffleNet – перемешивание каналов

группы нужны, чтобы свёртка работала только на каналах группы (быстрее)

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** ($0.25\times$, $g = 8$)

Table 4. Classification error vs. various structures (%, *smaller number represents better performance*). We do not report VGG-like structure on smaller networks because the accuracy is significantly worse.

чтобы не было такого, что зависимость локальна в группах
каналы «перемешивают»

N×C×H×W → N×K×G×H×W → N×G×K×H×W → N×C×H×W

принцип: resize([A, A, B, B], [2, 2]).T.flattern = [A, B, A, B]

основной блок ~ ResNeXt

Мобильные архитектуры: ShuffleNet – перемешивание каналов

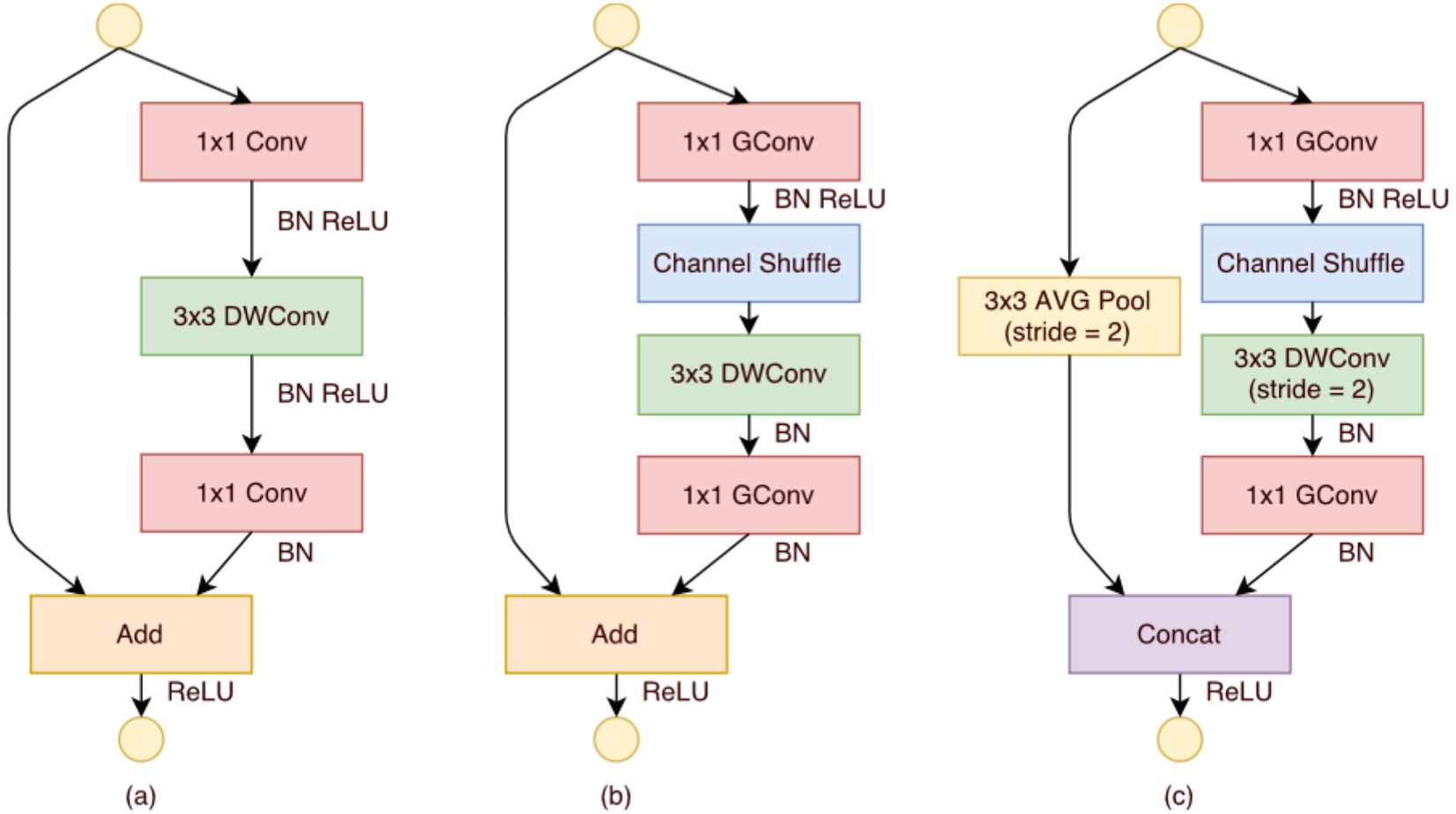


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

последние 1×1-свёртки для получения нужного числа каналов (для прокидывания связи)

Мобильные архитектуры: ShuffleNet v2

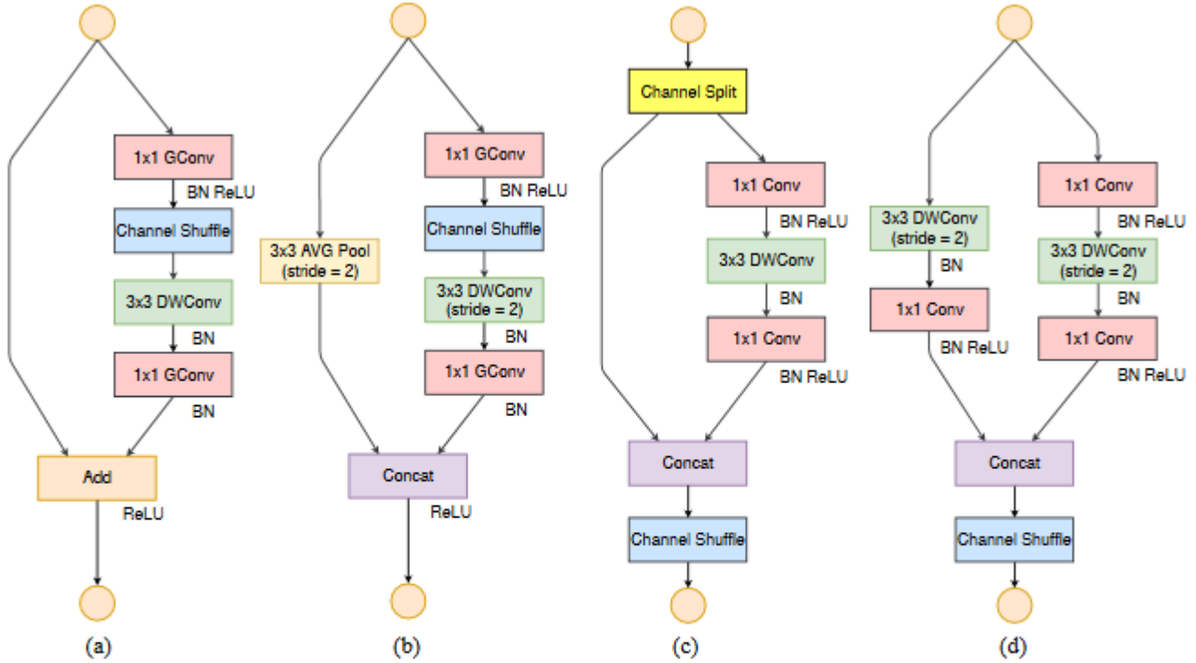


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

channel split – по разным веткам пускаем разные каналы

(прокидывание половины с конкатенацией)

1×1-свёртки теперь не групповые, число каналов не меняется

Ningning Ma, et al. «ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design» https://arxiv.org/abs/1807.11164

Мобильные архитектуры + NAS: FBNet

Дифференцируемый NAS
похожа на MobileNet v2

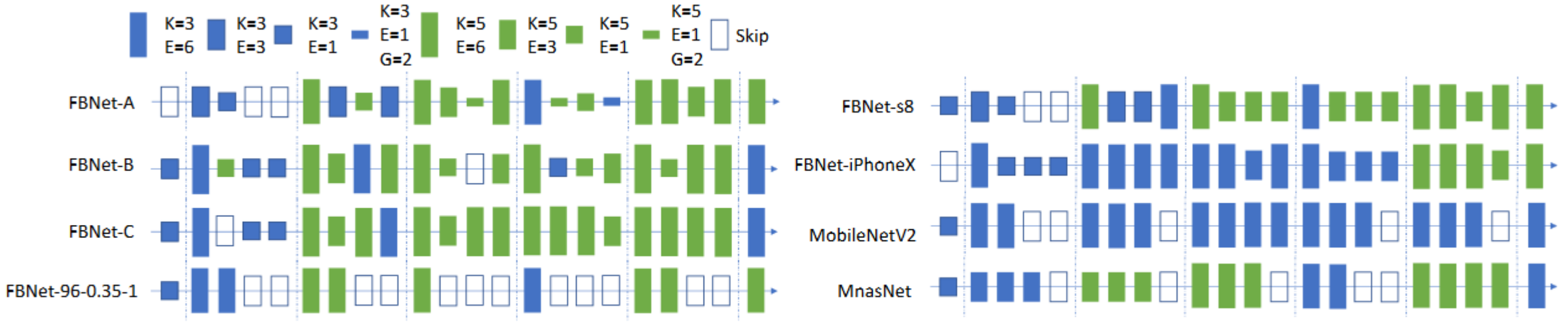


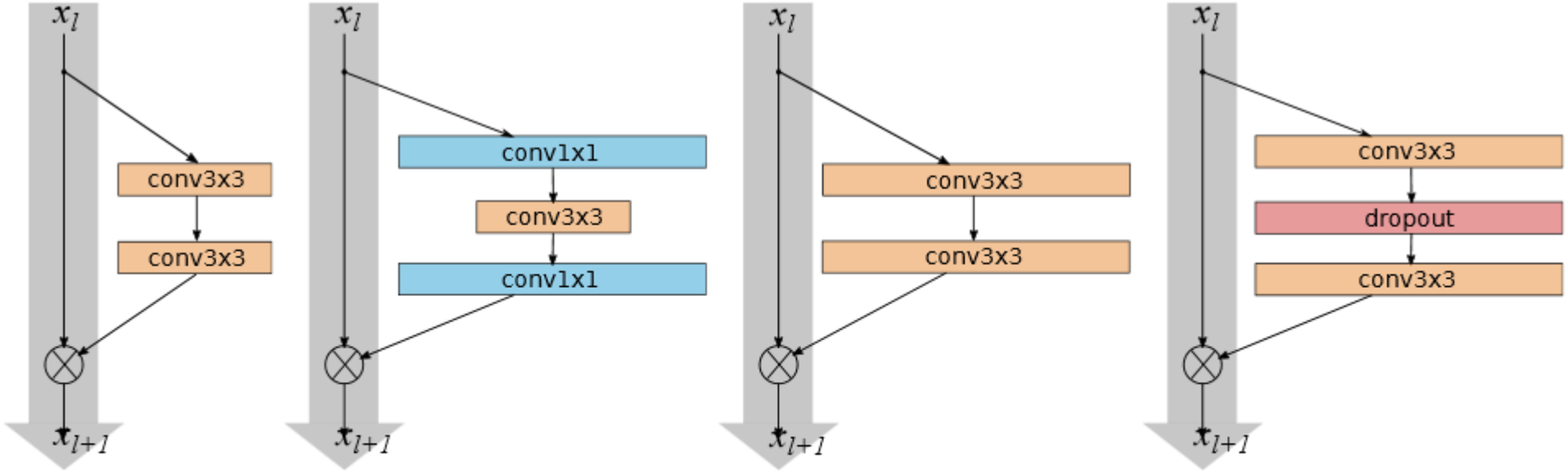
Figure 4. Visualization of some of the searched architectures. We use rectangle boxes to denote blocks for each layer. We use different colors to denote the kernel size of the depthwise convolution, blue for kernel size of 3, green for kernel size of 5, and empty for skipping. We use height to denote the expansion rate of the block: 6, 3, 1, and 1 with group-2 convolution.

разные архитектуры под разные устройства

Bichen Wu, et al. «FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search» // https://arxiv.org/abs/1812.03443

WideResNets – «широкие» сети

можно увеличивать глубину, а что если увеличивать ширину



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

Figure 1: Various residual blocks used in the paper. Batch normalization and ReLU precede each convolution (omitted for clarity)

Sergey Zagoruyko, Nikos Komodakis «Wide Residual Networks» // https://arxiv.org/abs/1605.07146

WideResNets – «широкие» сети

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 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 × 8]

Table 1: Structure of wide residual networks. Network width is determined by factor k. Original architecture [13] is equivalent to $k = 1$. Groups of convolutions are shown in brackets where N is a number of blocks in group, downsampling performed by the first layers in groups conv3 and conv4. Final classification layer is omitted for clearance. In the particular example shown, the network uses a ResNet block of type $B(3,3)$.

параметр k – ширина – и оптимизация по нему

WideResNets – «широкие» сети

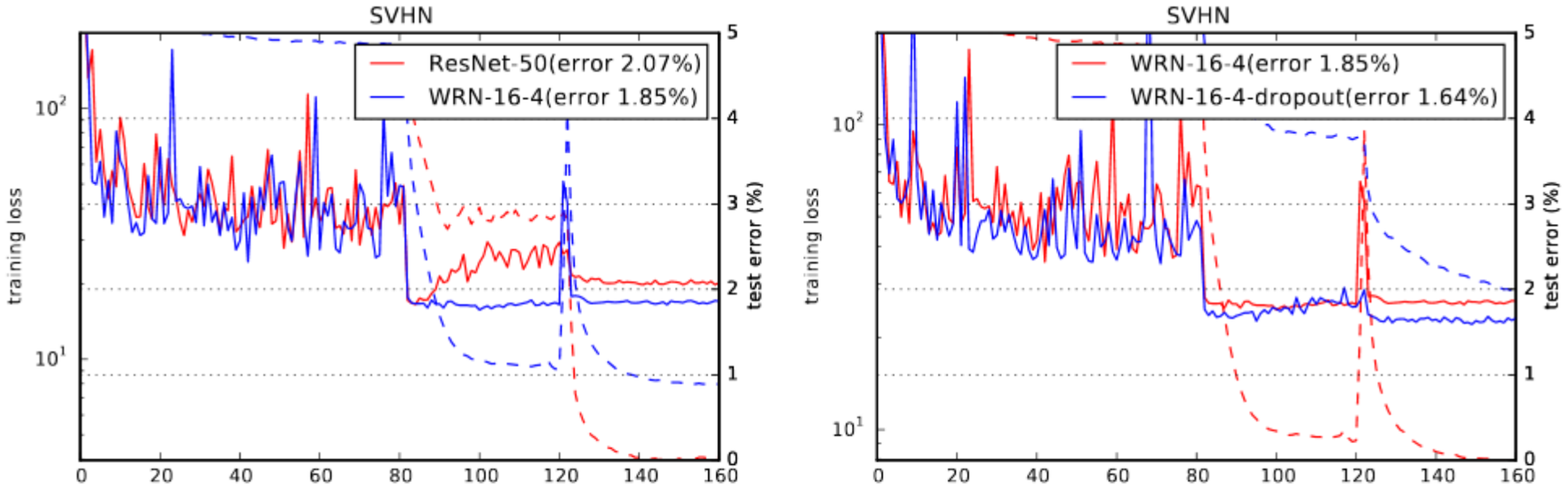


Figure 3: Training curves for SVHN. On the left: thin and wide networks, on the right: effect of dropout. Solid lines denote test error (y-axis on the right), dashed lines denote training loss (y-axis on the left).

RevNet

При backprop надо хранить значения активаций – занимает много памяти

Выход: получаем их автоматически при обратном проходе (из предыдущего слоя)

Идея:

$$y_1 = x_1 + \mathcal{F}(x_2) \qquad x_2 = y_2 - \mathcal{G}(y_1)$$
$$y_2 = x_2 + \mathcal{G}(y_1) \qquad x_1 = y_1 - \mathcal{F}(x_2)$$

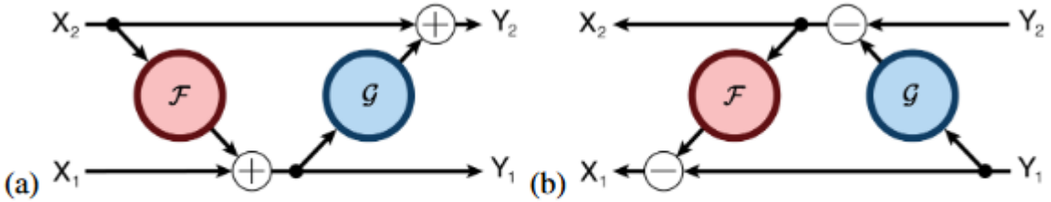


Figure 2: (**a**) the forward, and (**b**) the reverse computations of a residual block, as in Equation 8.

Если в сети всё-таки есть необратимые части (пулинг, свёртки с шагом), то придётся хранить активации

Aidan N. Gomez «The Reversible Residual Network: Backpropagation Without Storing Activations» // https://arxiv.org/abs/1707.04585

RevNet

Table 3: Classification error on CIFAR

Architecture	CIFAR-10 [15]		CIFAR-100 [15]	
	ResNet	RevNet	ResNet	RevNet
32 (38)	**7.14%**	7.24%	29.95%	**28.96%**
110	**5.74%**	5.76%	26.44%	**25.40%**
164	5.24%	**5.17%**	**23.37%**	23.69%

Table 4: Top-1 classification error on ImageNet (single crop)

ResNet-101	RevNet-104
23.01%	23.10%

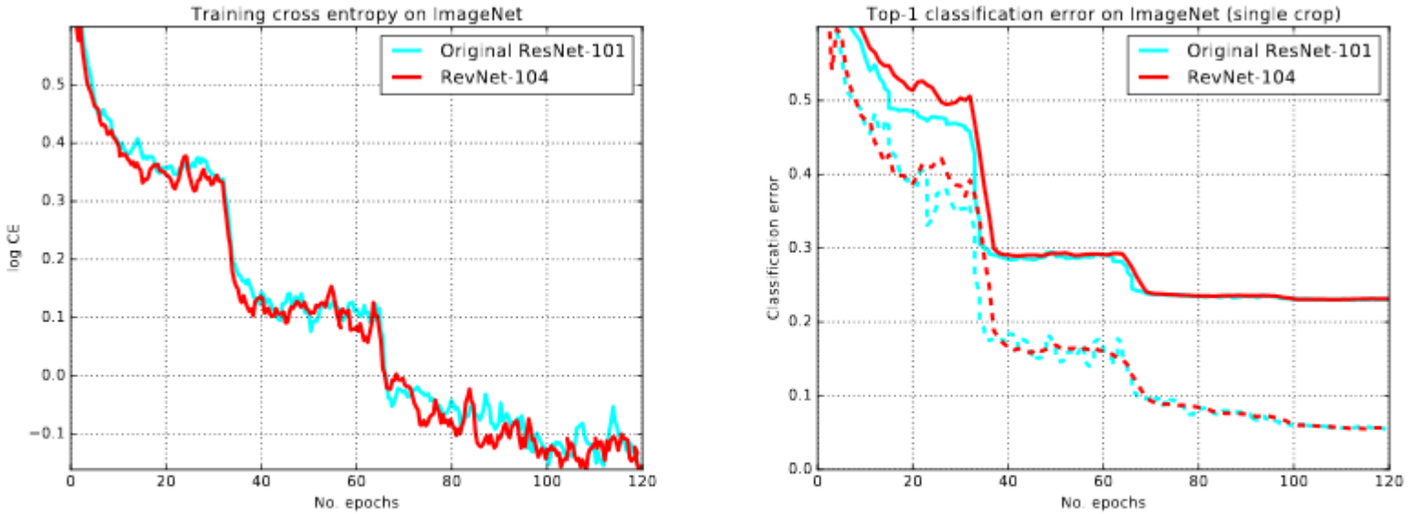


Figure 3: Training curves for ResNet-101 vs. RevNet-104 on ImageNet, with both networks having approximately the same depth and number of free parameters. **Left:** training cross entropy; **Right:** classification error, where dotted lines indicate training, and solid lines validation.

iRevNet

полностью обратимое преобразование

~ решение задачи без сжатия информации

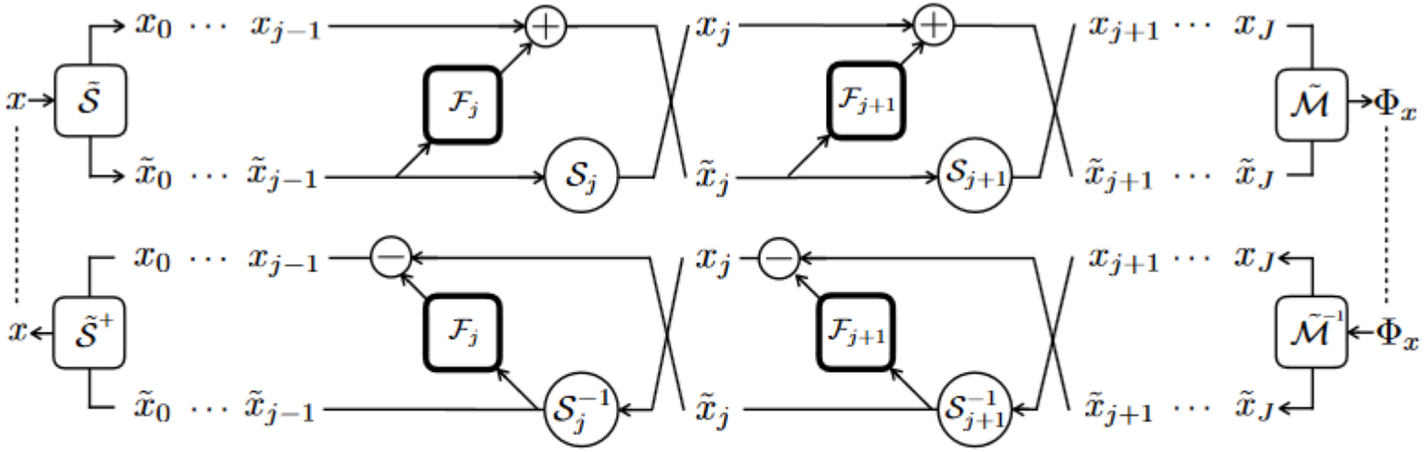


Figure 1: The main component of the i-RevNet and its inverse. RevNet blocks are interleaved with convolutional bottlenecks $\mathcal{F}_j$ and reshuffling operations $\mathcal{S}_j$ to ensure invertibility of the architecture and computational efficiency. The input is processed through a splitting operator $\tilde{\mathcal{S}}$, and output is merged through $\tilde{\mathcal{M}}$. Observe that the inverse network is obtained with minimal adaptations.

J ̈orn-Henrik Jacobse «i-RevNet: Deep Invertible Networks» //

https://arxiv.org/pdf/1802.07088.pdf

iRevNet

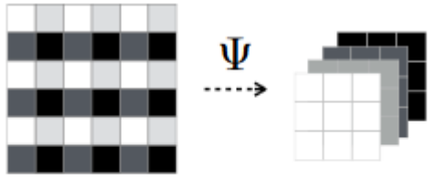


Figure 2: Illustration of the invertible down-sampling

Architecture	Injective	Bijective	Top-1 error	Parameters
ResNet	-	-	24.7	26M
RevNet	-	-	25.2	28M
i-RevNet (a)	yes	-	24.7	181M
i-RevNet (b)	yes	yes	26.7	29M

Table 1: Comparison of different architectures trained on ILSVRC-2012, in terms of classification accuracy and number of parameters

NFNets = Normalization Free Networks

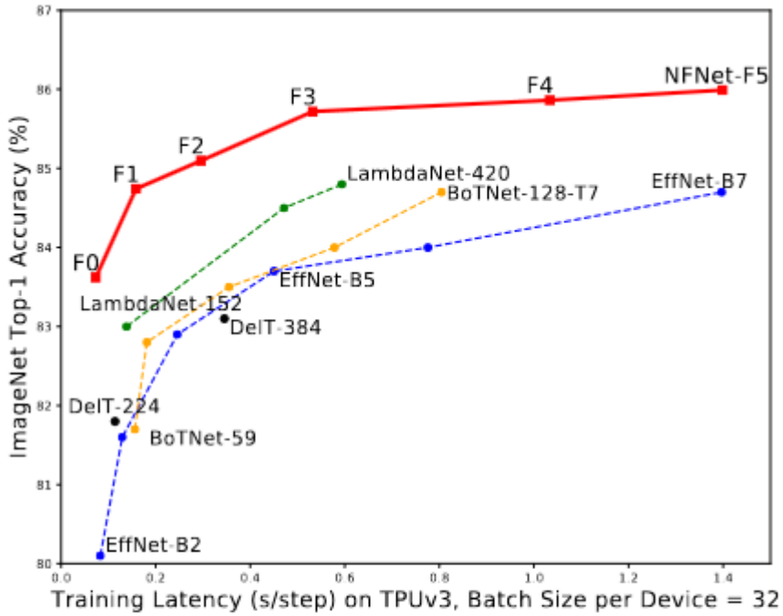
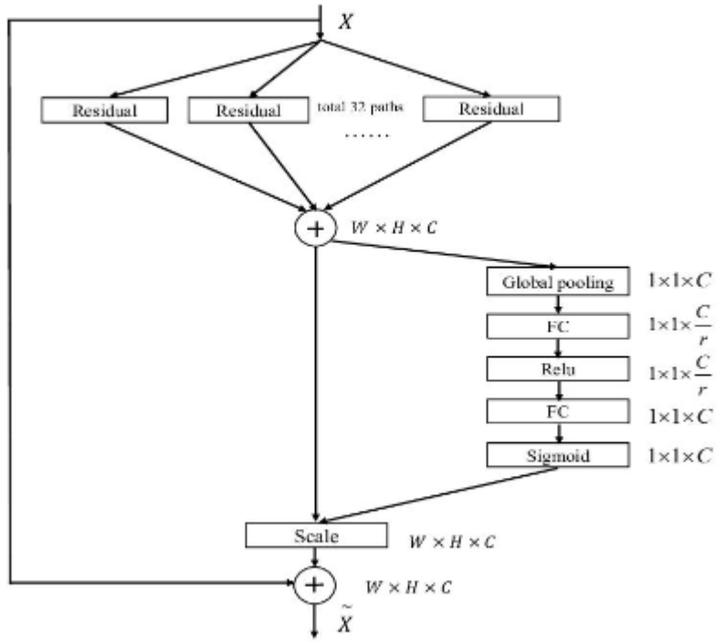


Figure 1. **ImageNet Validation Accuracy vs Training Latency.** All numbers are single-model, single crop. Our NFNet-F1 model achieves comparable accuracy to an EffNet-B7 while being 8.7× faster to train. Our NFNet-F5 model has similar training latency to EffNet-B7, but achieves a state-of-the-art 86.0% top-1 accuracy on ImageNet. We further improve on this using Sharpness Aware Minimization (Foret et al., 2021) to achieve 86.5% top-1 accuracy.

Базовая архитектура NFNet – SE-ResNext-D



главное – техника отказа от BN!

Andrew Brock et al. «High-Performance Large-Scale Image Recognition Without Normalization» // https://arxiv.org/pdf/2102.06171.pdf

NFNets

Плюсы и минусы BatchNorm

+ сглаживание поверхности функции ошибки: более стабильное обучение при увеличении lr (при достаточном увеличении размера батча)
+ регуляризация
+ возможность обучения более глубоких сетей
+ возможность использования больших батчей

– дополнительные расходы по времени и памяти
– разное поведение во время обучения и тестирования
– сильное влияние размера батча, сложности с распределённым обучением
– в батче объекты зависимы

NFNets

до этой статьи авторы применяли идею констант для блоков, которые определялись заранее (с учётом того, как блок меняет данные)

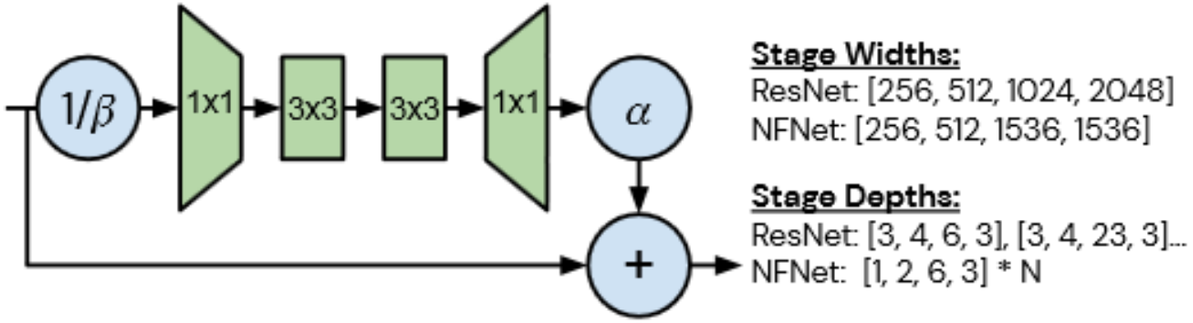


Figure 3. Summary of NFNet bottleneck block design and architectural differences. See Figure 5 in Appendix C for more details.

$$h_{i+1} = h_i + \alpha f_i(h_i/\beta_i)$$

но до EfficientNet качество не дотягивало

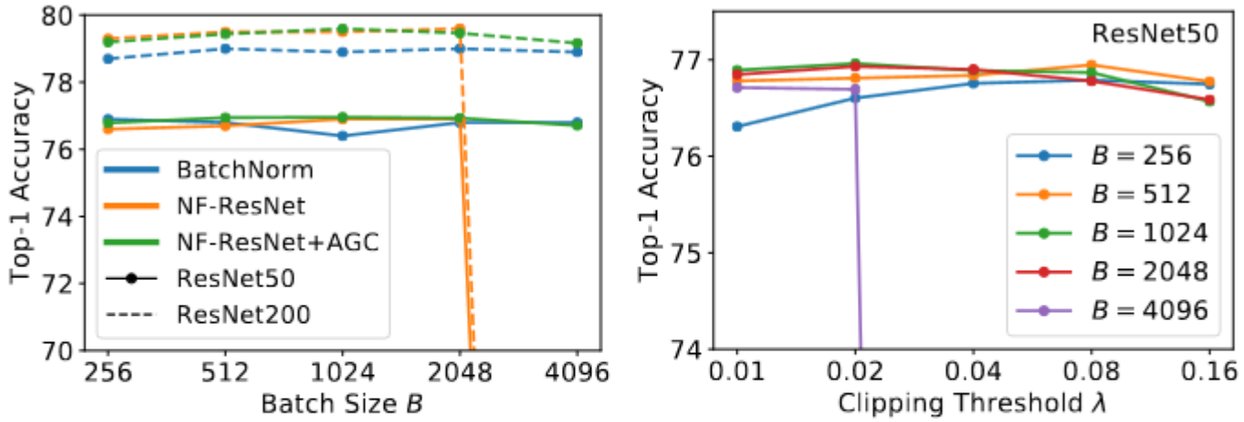
NFNets: Adaptive Gradient Clipping

если отношение норм градиента и веса большое…

$$G_i^\ell \to \begin{cases} \lambda \frac{\|W_i^\ell\|_F^\star}{\|G_i^\ell\|_F} G_i^\ell & \text{if } \frac{\|G_i^\ell\|_F}{\|W_i^\ell\|_F^\star} > \lambda, \\ G_i^\ell & \text{otherwise.} \end{cases}$$

Table 2. The effect of architectural modifications and data augmentation on ImageNet Top-1 accuracy (averaged over 3 seeds).

	F0	F1	F2	F3
Baseline	80.4	81.7	82.0	82.3
+ Modified Width	80.9	81.8	82.0	82.3
+ Second Conv	81.3	82.2	82.4	82.7
+ MixUp	82.2	82.9	83.1	83.5
+ RandAugment	83.2	84.6	84.8	85.0
+ CutMix	**83.6**	**84.7**	**85.1**	**85.7**
Default Width + Augs	83.1	84.5	85.0	85.5



(a) (b)

Figure 2. (a) AGC efficiently scales NF-ResNets to larger batch sizes. (b) The performance across different clipping thresholds λ.

– ключевая техника, которая помогла сети без нормализации (раньше не получалось достигнуть хорошего качества)

позволяет более эффективно использовать некоторые аугментации

Table 3. ImageNet Accuracy comparison for NFNets and a representative set of models, including SENet (Hu et al., 2018), LambdaNet, (Bello, 2021), BoTNet (Srinivas et al., 2021), and DeIT (Touvron et al., 2020). Except for results using SAM, our results are averaged over three random seeds. Latencies are given as the time in milliseconds required to perform a single full training step on TPU or GPU (V100).

Model	#FLOPs	#Params	Top-1	Top-5	TPUv3 Train	GPU Train
ResNet-50	4.10B	26.0M	78.6	94.3	41.6ms	35.3ms
EffNet-B0	0.39B	5.3M	77.1	93.3	51.1ms	44.8ms
SENet-50	4.09B	28.0M	79.4	94.6	64.3ms	59.4ms
NFNet-F0	**12.38B**	**71.5M**	**83.6**	**96.8**	**73.3ms**	**56.7ms**
EffNet-B3	1.80B	12.0M	81.6	95.7	129.5ms	116.6ms
LambdaNet-152	–	51.5M	83.0	96.3	138.3ms	135.2ms
SENet-152	19.04B	66.6M	83.1	96.4	149.9ms	151.2ms
BoTNet-110	10.90B	54.7M	82.8	96.3	181.3ms	–
NFNet-F1	**35.54B**	**132.6M**	**84.7**	**97.1**	**158.5ms**	**133.9ms**
EffNet-B4	4.20B	19.0M	82.9	96.4	245.9ms	221.6ms
BoTNet-128-T5	19.30B	75.1M	83.5	96.5	355.2ms	–
NFNet-F2	**62.59B**	**193.8M**	**85.1**	**97.3**	**295.8ms**	**226.3ms**
SENet-350	52.90B	115.2M	83.8	96.6	593.6ms	–
EffNet-B5	9.90B	30.0M	83.7	96.7	450.5ms	458.9ms
LambdaNet-350	–	105.8M	84.5	97.0	471.4ms	–
BoTNet-77-T6	23.30B	53.9M	84.0	96.7	578.1ms	–
NFNet-F3	**114.76B**	**254.9M**	**85.7**	**97.5**	**532.2ms**	**524.5ms**
LambdaNet-420	–	124.8M	84.8	97.0	593.9ms	–
EffNet-B6	19.00B	43.0M	84.0	96.8	775.7ms	868.2ms
BoTNet-128-T7	45.80B	75.1M	84.7	97.0	804.5ms	–
NFNet-F4	**215.24B**	**316.1M**	**85.9**	**97.6**	**1033.3ms**	**1190.6ms**
EffNet-B7	37.00B	66.0M	84.7	97.0	1397.0ms	1753.3ms
DeIT 1000 epochs	–	87.0M	85.2	–	–	–
EffNet-B8+MaxUp	62.50B	87.4M	85.8	–	–	–
NFNet-F5	**289.76B**	**377.2M**	**86.0**	**97.6**	**1398.5ms**	**2177.1ms**
NFNet-F5+SAM	289.76B	377.2M	86.3	97.9	1958.0ms	–
NFNet-F6+SAM	**377.28B**	**438.4M**	**86.5**	**97.9**	**2774.1ms**	–

NFNets

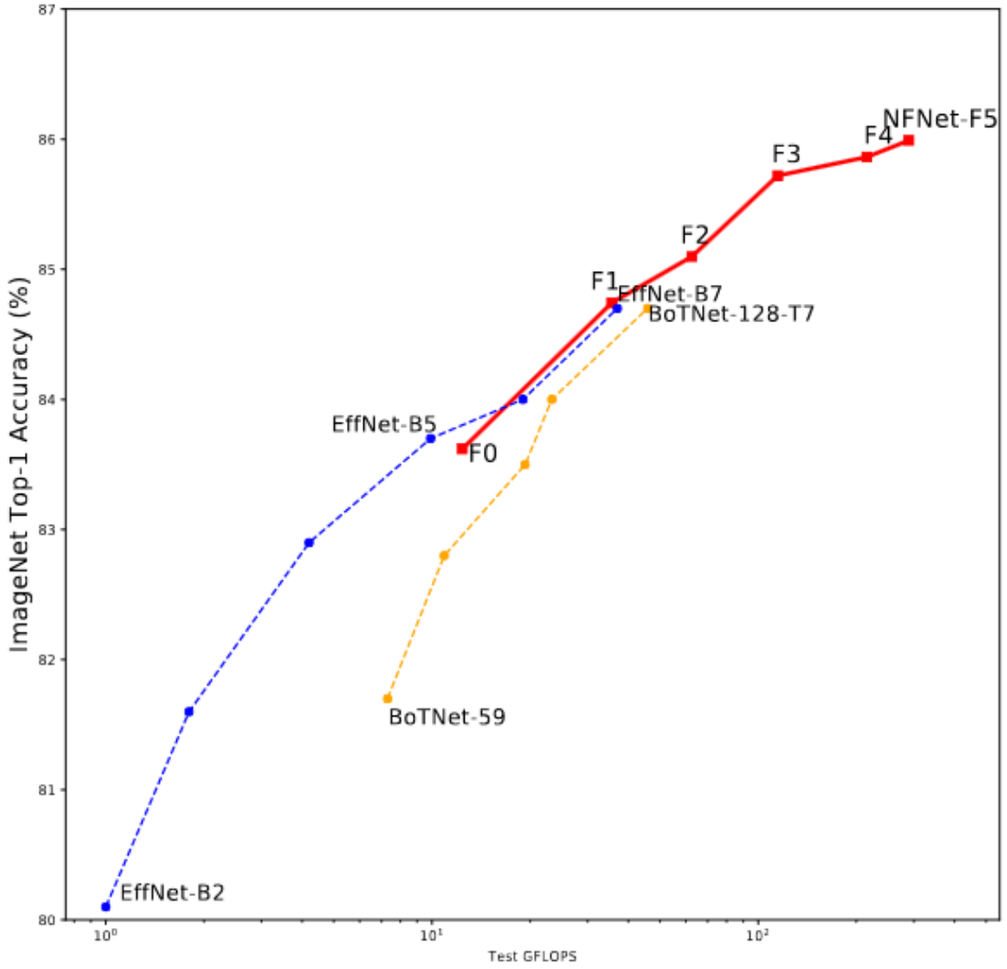


Figure 4. **ImageNet Validation Accuracy vs. Test GFLOPs.** All numbers are single-model, single crop. Our NFNet models are competitive with large EfficientNet variants for a given FLOPs budget, despite being optimized for training latency.

лучше себя показывают после пред-тренировки (для трансфера)

– при сравнении по GFLOPS-ам не тесте уже не выглядят лучшими

У наименьшей модели качество как и у EfficientNet-B7, а обучается в 8.7 раз быстрее

У лучшей модели SOTA top-1 accuracy = 86.5%

ConvNeXt

ResNet + последние (трансформерные) новинки
– чисто свёрточная архитектура, не хуже трансформеров

300 эпох
AdamW
Mixup, Cutmix, RandAugment, Random Erasing
регуляризация(stochastic depth, label smoothing, …)

количество блоков в ResNet: (3,4,6,3) → (3,3,9,3)
4×4-свёртки с шагом 4, есть 7×7-свёртки
depthwise свёртки
inverted bottleneck (вместо узкое горло, наоборот – широкое)
ReLU →GELU (Gaussian Error Linear Unit)
LayerNorm
меньше активаций и нормализаций

Zhuang Liu et. al.«A ConvNet for the 2020s» // https://arxiv.org/abs/2201.03545

ConvNeXt

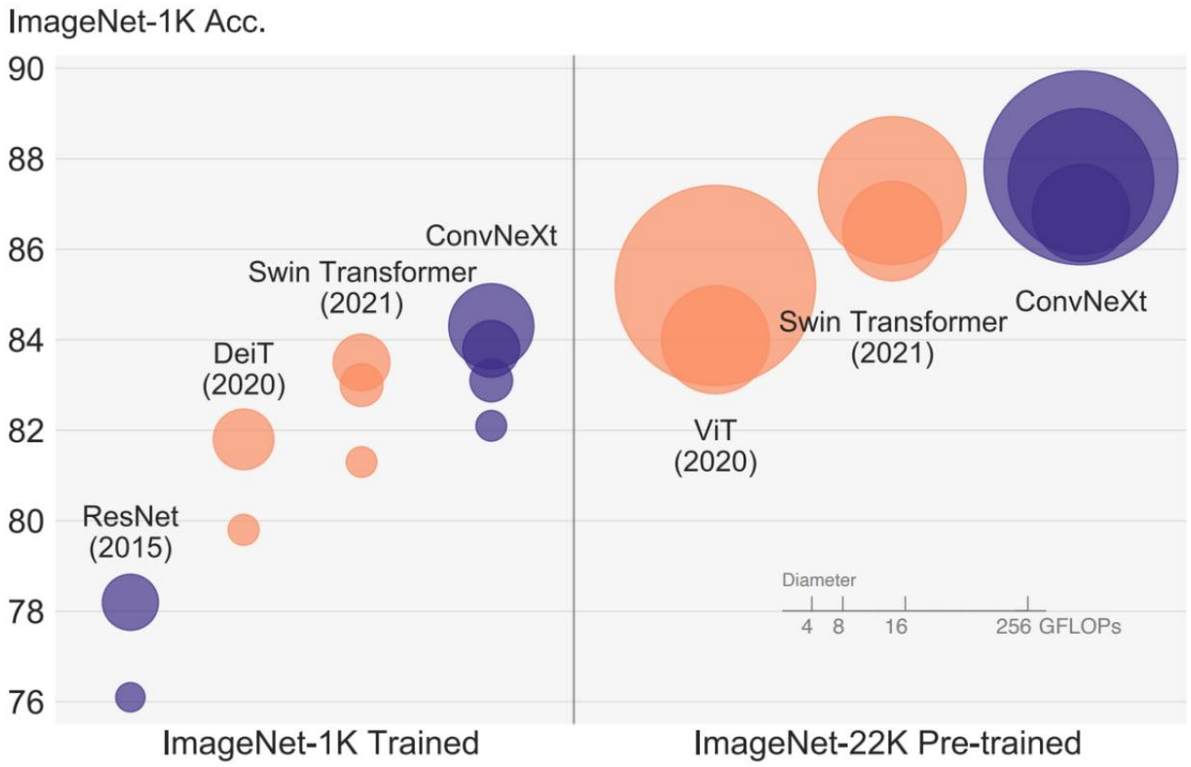


Figure 1. **ImageNet-1K classification** results for • ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

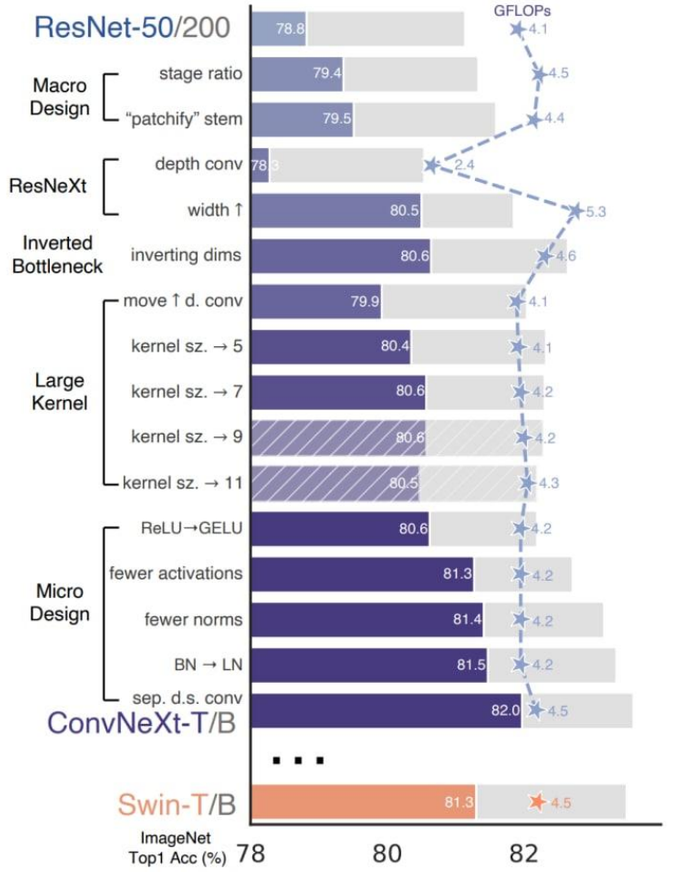


Figure 2. We modernize a standard ConvNet (ResNet) towards the design of a hierarchical vision Transformer (Swin), without introducing any attention-based modules. The foreground bars are model accuracies in the ResNet-50/Swin-T FLOP regime; results for the ResNet-200/Swin-B regime are shown with the gray bars. A hatched bar means the modification is not adopted. Detailed results for both regimes are in the appendix. Many Transformer architectural choices can be incorporated in a ConvNet, and they lead to increasingly better performance. In the end, our pure ConvNet model, named ConvNeXt, can outperform the Swin Transformer.

ConvNeXt

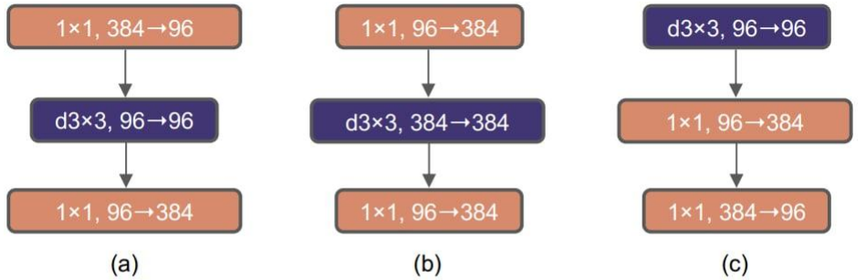


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

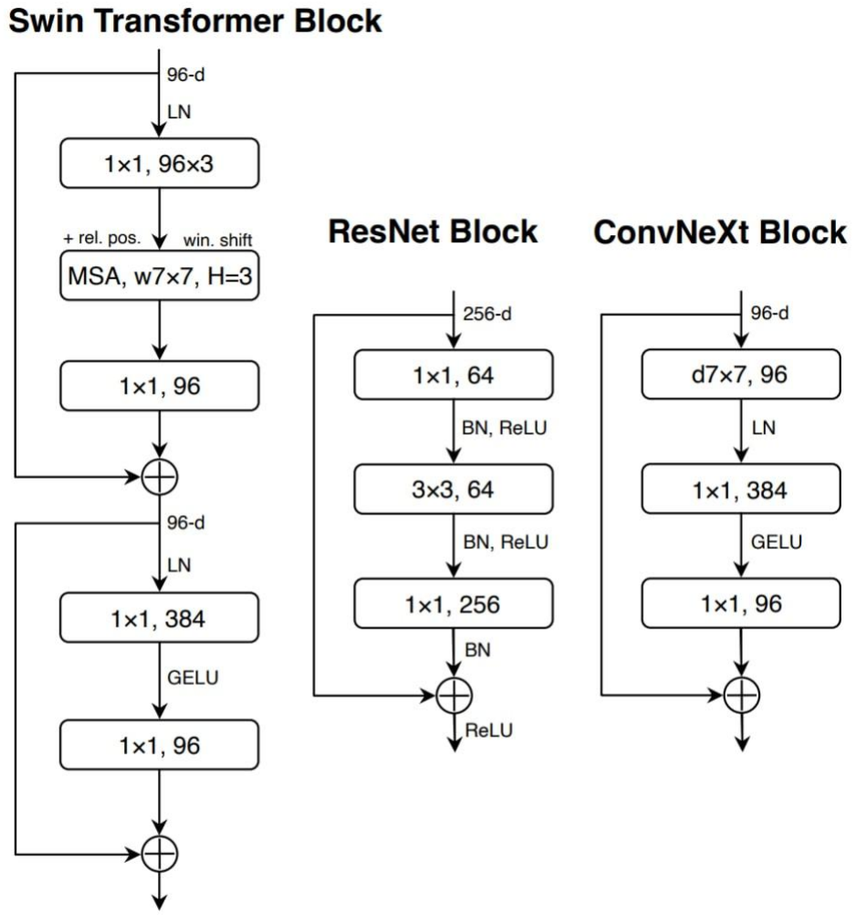


Figure 4. **Block designs** for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

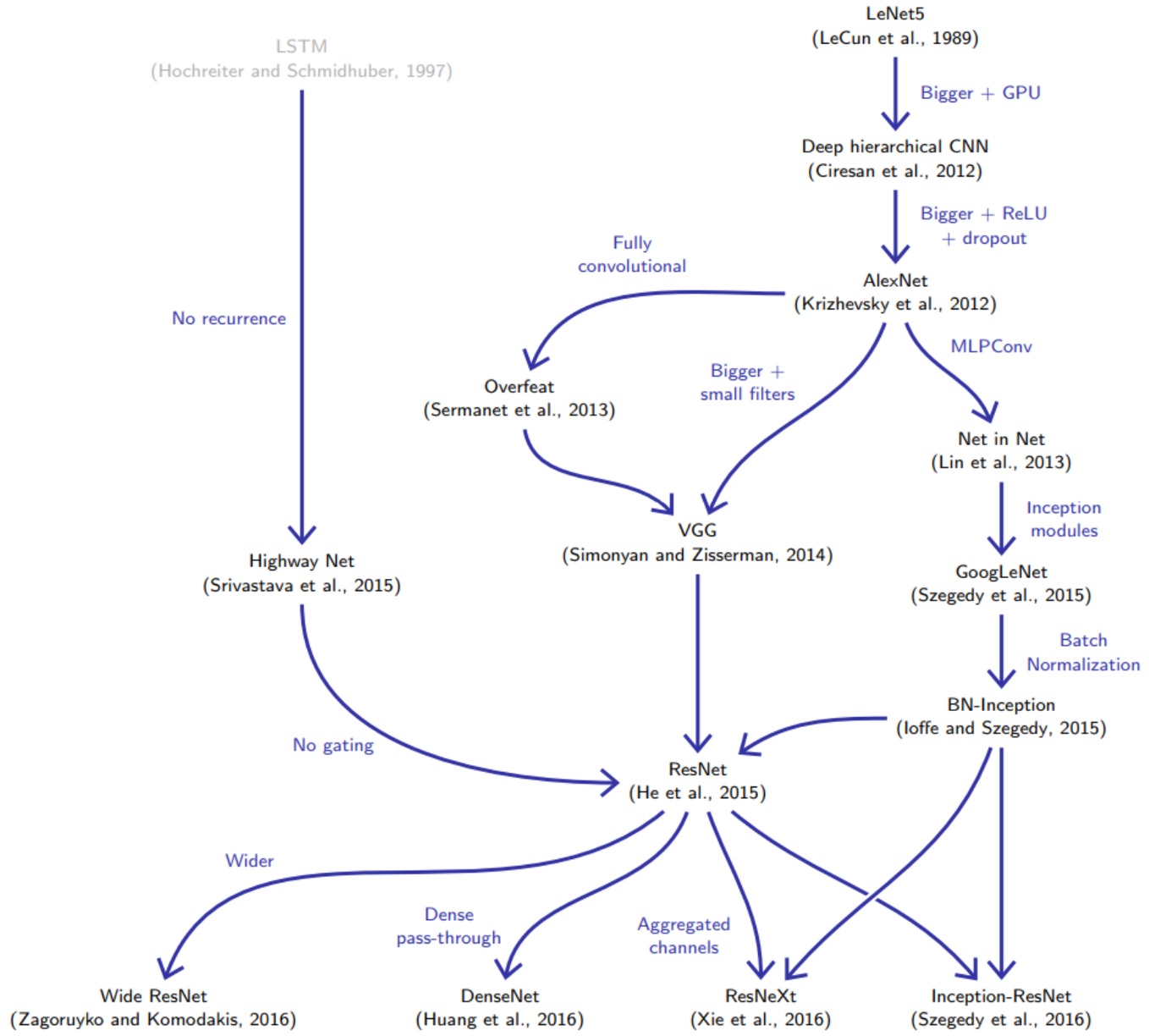
ConvNeXt

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
• RegNetY-4G [51]	224^2	21M	4.0G	1156.7	80.0
• RegNetY-8G [51]	224^2	39M	8.0G	591.6	81.7
• RegNetY-16G [51]	224^2	84M	16.0G	334.7	82.9
• EffNet-B3 [67]	300^2	12M	1.8G	732.1	81.6
• EffNet-B4 [67]	380^2	19M	4.2G	349.4	82.9
• EffNet-B5 [67]	456^2	30M	9.9G	169.1	83.6
• EffNet-B6 [67]	528^2	43M	19.0G	96.9	84.0
• EffNet-B7 [67]	600^2	66M	37.0G	55.1	84.3
○ DeiT-S [68]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [68]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
• ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
• ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
• ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
• ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
• ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
• ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**
ImageNet-22K pre-trained models					
• R-101x3 [36]	384^2	388M	204.6G	-	84.4
• R-152x4 [36]	480^2	937M	840.5G	-	85.4
○ ViT-B/16 [18]	384^2	87M	55.5G	93.1	84.0
○ ViT-L/16 [18]	384^2	305M	191.1G	28.5	85.2
○ Swin-B	224^2	88M	15.4G	286.6	85.2
• ConvNeXt-B	224^2	89M	15.4G	292.1	**85.8**
○ Swin-B	384^2	88M	47.0G	85.1	86.4
• ConvNeXt-B	384^2	89M	45.1G	95.7	**86.8**
○ Swin-L	224^2	197M	34.5G	145.0	86.3
• ConvNeXt-L	224^2	198M	34.4G	146.8	**86.6**
○ Swin-L	384^2	197M	103.9G	46.0	87.3
• ConvNeXt-L	384^2	198M	101.0G	50.4	**87.5**
• ConvNeXt-XL	224^2	350M	60.9G	89.3	**87.0**
• ConvNeXt-XL	384^2	350M	179.0G	30.2	**87.8**

Table 1. **Classification accuracy on ImageNet-1K.** Similar to Transformers, ConvNeXt also shows promising scaling behavior with higher-capacity models and a larger (pre-training) dataset. Inference throughput is measured on a V100 GPU, following [42]. On an A100 GPU, ConvNeXt can have a much higher throughput than Swin Transformer. See Appendix E.

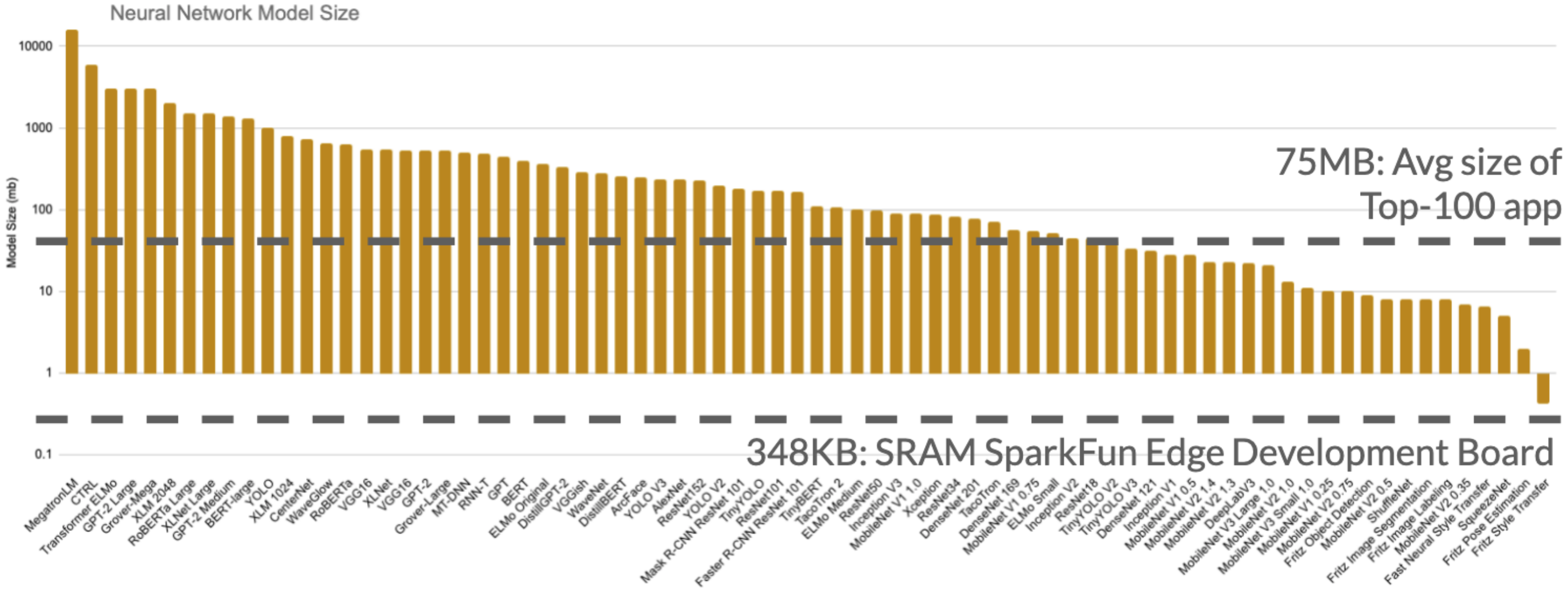
	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

Table 9. **Detailed architecture specifications** for ResNet-50, ConvNeXt-T and Swin-T.



https://fleuret.org/ee559/ee559-slides-7-2-image-classification.pdf

Сложность моделей



Сравнение архитектур https://arxiv.org/pdf/1810.00736.pdf

TABLE 1: Inference time *vs.* batch size. Inference time per image is estimated across different batch sizes for the Titan Xp (left), and Jetson TX1 (right). Missing data are due to the lack of enough system memory required to process the larger batches.

DNN	1	2	4	8	16	32	64
AlexNet	1.28	0.70	0.48	0.27	0.18	0.14	0.15
BN-Inception	5.79	3.00	1.64	1.10	0.87	0.77	0.71
CaffeResNet-101	8.20	4.82	3.32	2.54	2.27	2.16	2.08
DenseNet-121 (k=32)	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-169 (k=32)	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201 (k=32)	17.15	9.25	5.36	3.66	2.84	2.41	2.27
DenseNet-161 (k=48)	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DPN-68	10.68	5.36	3.24	2.47	1.80	1.59	1.52
DPN-98	22.31	13.84	8.97	6.77	5.59	4.96	4.72
DPN-131	29.70	18.29	11.96	9.12	7.57	6.72	6.37
FBResNet-152	14.55	7.79	5.15	4.31	3.96	3.76	3.65
GoogLeNet	4.54	2.44	1.65	1.06	0.86	0.76	0.72
Inception-ResNet-v2	25.94	14.36	8.82	6.43	5.19	4.88	4.59
Inception-v3	10.10	5.70	3.65	2.54	2.05	1.89	1.80
Inception-v4	18.96	10.61	6.53	4.85	4.10	3.77	3.61
MobileNet-v1	2.45	0.89	0.68	0.60	0.55	0.53	0.53
MobileNet-v2	3.34	1.63	0.95	0.78	0.72	0.63	0.61
NASNet-A-Large	32.30	23.00	19.75	18.49	18.11	17.73	17.77
NASNet-A-Mobile	22.36	11.44	5.60	2.81	1.61	1.75	1.51
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNeXt-101 (32x4d)	17.05	9.02	6.27	4.62	3.71	3.25	3.11
ResNeXt-101 (64x4d)	21.05	15.54	10.39	7.80	6.39	5.62	5.29
SE-ResNet-101	15.10	9.26	6.17	4.72	4.03	3.62	3.42
SE-ResNet-152	23.43	13.08	8.74	6.55	5.51	5.06	4.85
SE-ResNet-50	8.32	5.16	3.36	2.62	2.22	2.01	2.06
SE-ResNeXt-101 (32x4d)	24.96	13.86	9.16	6.55	5.29	4.53	4.29
SE-ResNeXt-50 (32x4d)	12.06	7.41	5.12	3.64	2.97	3.01	2.56
SENet-154	53.80	30.30	19.32	13.27	10.45	9.41	8.91
ShuffleNet	5.40	2.67	1.37	0.82	0.66	0.59	0.56
SqueezeNet-v1.0	1.53	0.84	0.66	0.59	0.54	0.52	0.53
SqueezeNet-v1.1	1.60	0.77	0.44	0.37	0.32	0.31	0.30
VGG-11	3.57	4.40	2.89	1.56	1.19	1.10	1.13
VGG-11_BN	3.49	4.60	2.99	1.71	1.33	1.24	1.27
VGG-13	3.88	5.03	3.44	2.25	1.83	1.75	1.79
VGG-13_BN	4.40	5.37	3.71	2.42	2.05	1.97	2.00
VGG-16	5.17	5.91	4.01	2.84	2.20	2.12	2.15
VGG-16_BN	5.04	5.95	4.27	3.06	2.45	2.36	2.41
VGG-19	5.50	6.26	4.71	3.29	2.59	2.52	2.50
VGG-19_BN	6.17	6.67	4.86	3.56	2.88	2.74	2.76
Xception	6.44	5.35	4.90	4.47	4.41	4.41	4.36

DNN	1	2	4	8	16	32	64
AlexNet	28.88	13.00	8.58	6.56	5.39	4.77	
BN-Inception	35.52	26.48	25.10	23.89	21.21	20.47	
CaffeResNet-101	84.47	91.37	70.33	63.53	56.38	53.73	
DenseNet-121 (k=32)	66.43	50.87	50.20	43.89	40.41	38.22	
DenseNet-169 (k=32)	137.96	130.27	110.82	100.56	92.97	88.94	
DenseNet-201 (k=32)	84.57	61.71	62.62	53.73	49.28	46.26	
DenseNet-161 (k=48)	103.20	76.11	77.10	68.32	62.73	59.14	
DPN-68	113.08	52.73	42.85	43.32	38.18	36.40	36.22
DPN-98	243.51	148.51	135.30	125.92	123.34	118.68	117.27
DPN-131	330.15	204.69	184.89	172.25	165.59	162.67	160.66
FBResNet-152	133.68	147.75	113.48	105.78	94.26	97.47	
GoogLeNet	32.11	27.19	23.29	21.66	19.77	19.96	
Inception-ResNet-v2	198.95	141.29	127.97	130.25	117.99	116.47	
Inception-v3	79.39	59.04	56.46	51.79	47.60	46.85	
Inception-v4	158.00	120.23	106.77	102.21	95.51	95.40	
MobileNet-v1	15.06	11.94	11.34	11.03	10.82	10.58	10.55
MobileNet-v2	20.51	14.58	13.67	13.56	13.18	13.10	12.72
NASNet-A-Large	437.20	399.99	385.75	383.55	389.67		
NASNet-A-Mobile	133.87	62.91	33.72	30.62	29.72	28.92	28.55
ResNet-101	84.52	77.90	71.23	67.14	58.11		
ResNet-152	124.67	113.65	101.41	96.76	82.35		
ResNet-18	21.16	15.30	14.35	13.82	11.99	10.73	12.45
ResNet-34	39.88	28.82	27.51	24.97	20.41	18.48	17.97
ResNet-50	53.09	44.84	41.20	38.79	35.72		
ResNeXt-101 (32x4d)	115.37	90.93	84.64	79.66	77.63		
ResNeXt-101 (64x4d)	177.40	155.77	144.82	137.43	134.07		
SE-ResNet-101	118.13	105.11	96.71	91.68	80.99		
SE-ResNet-152	169.73	155.08	139.72	133.59	116.97		
SE-ResNet-50	69.65	61.37	55.33	51.87	47.80		
SE-ResNeXt-101 (32x4d)	139.62	122.01	112.05	105.34	102.39		
SE-ResNeXt-50 (32x4d)	80.08	69.86	67.20	62.66	61.19		
SENet-154	309.48	240.80	221.84	211.00	207.06	201.49	201.66
ShuffleNet	36.58	22.61	13.80	13.36	12.91	12.66	12.50
SqueezeNet-v1.0	17.00	16.47	15.03	13.97	13.25	12.89	12.70
SqueezeNet-v1.1	11.05	9.88	8.80	7.90	7.38	7.20	7.04
VGG-11	106.44	125.84	85.84	60.10	32.56	30.51	32.27
VGG-11_BN	101.58	122.82	86.26	54.50	47.81	47.31	41.26
VGG-13	122.59	148.80	108.28	75.99	70.57	64.88	62.79
VGG-13_BN	129.69	153.68	113.90	81.19	76.39	70.59	67.38
VGG-16	151.52	169.92	129.89	96.81	91.72		
VGG-16_BN	163.37	176.35	136.85	103.45	98.11		
VGG-19	178.04	192.86	152.28	117.92	112.39		
VGG-19_BN	185.18	198.66	159.20	124.88	119.15		
Xception	98.96	93.40	90.49	87.65	86.89		

Сравнение архитектур

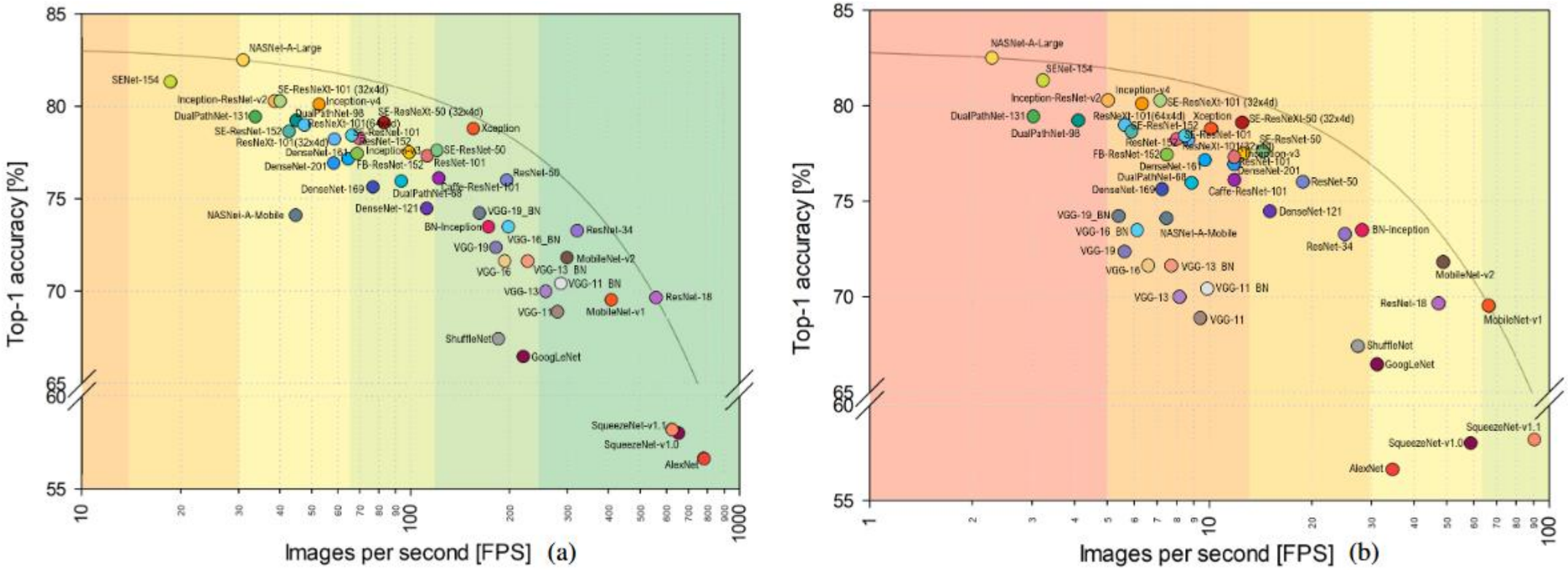


FIGURE 3: Top-1 accuracy *vs.* number of images processed per second (with batch size 1) using the Titan Xp (a) and JetsonTX1 (b).

Итоги

- Много интересных концепций (например, стохастическая глубина)
- Гиперсети – аналогичные идеи потом в механизме внимания
- Умное масштабирование (scaling up)
- Свёртки вместо пулинга
- depthwise/grouped-свёртки
- inverted bottleneck (вместо узкого – широкое горло)
- Перемешивание каналов
- Эксперименты с функциями активации
- Клиппирование помогает убрать нормализацию!
- Важный параметр – «inference time»
- Новые приёмы меняют старые сети!

Ссылки

Хороший обзор по мобильным архитектурам

https://machinethink.net/blog/mobile-architectures/