

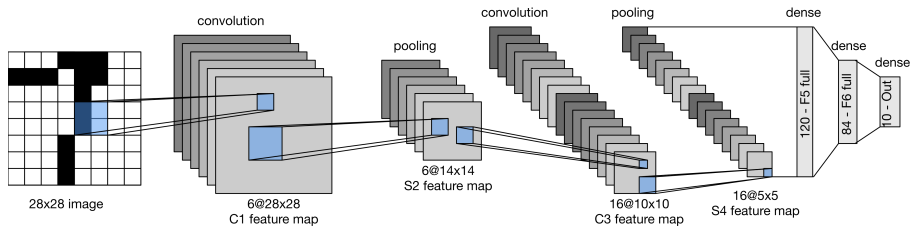
Лекция 4. Архитектуры сверточных нейронных сетей

Бабич Павел Васильевич

ИММиКН

2026

LeNet, 1998



LeNet-5 в PyTorch

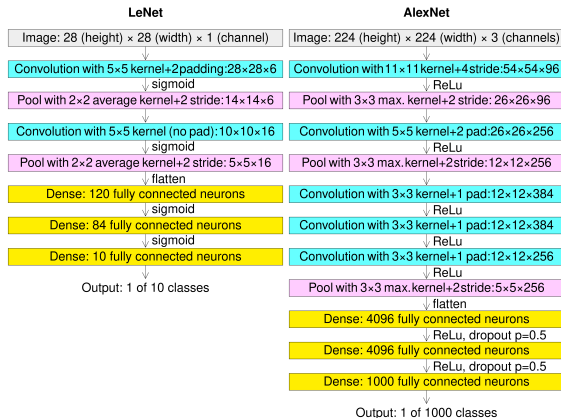
```
import torch.nn as nn

class LeNet(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),

            nn.Conv2d(6, 16, kernel_size=5),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(16 * 5 * 5, 120),
            nn.Tanh(),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        return self.classifier(x)
```

LeNet vs AlexNet



https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

AlexNet в PyTorch

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=1000):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 11, stride=4, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(3, stride=2),
            nn.Conv2d(64, 192, 5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(3, stride=2),
            nn.Conv2d(192, 384, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(3, stride=2),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 6 * 6, 4096), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096), nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        return self.classifier(self.features(x))
```

- **ImageNet-1K (ILSVRC-2012)** — один из основных датасетов для классификации изображений.
- Содержит **1000** классов объектов: животные, транспорт, предметы быта и др.
- Размер датасета:
 - **1,281,167** изображений для обучения;
 - **50,000** изображений для валидации;
 - **100,000** изображений в тестовой выборке.
- Классы организованы на основе иерархии **WordNet**.

Итого

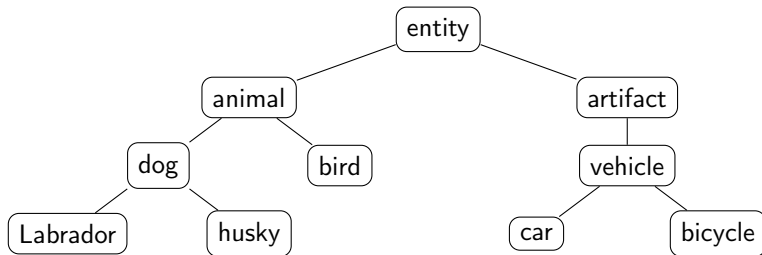
≈ 1.43 млн изображений, распределённых по **1000** классам.



<https://www.image-net.org/>

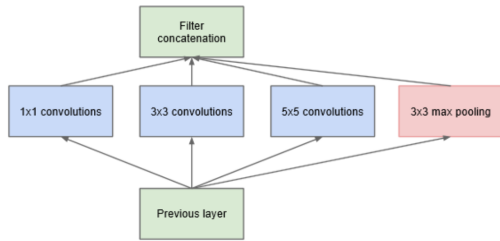
Иерархия классов ImageNet

ImageNet использует семантическую иерархию **WordNet**. Каждый класс соответствует **synset** — набору слов, обозначающих один концепт.

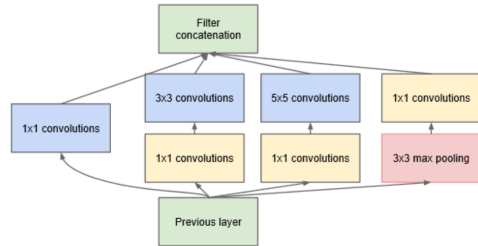


- Связи отражают отношение **общее понятие** → **частный класс**.
- ImageNet-1K содержит **1000 synset-классов**.
- Классы могут находиться на разных уровнях иерархии.

Inception Block



(a) Inception module, naïve version

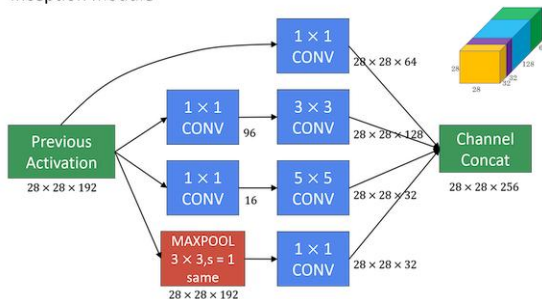


(b) Inception module with dimensionality reduction

<https://arxiv.org/pdf/1409.4842>

Inception Example

Inception module



<https://arxiv.org/pdf/1409.4842>

Inception-блок в PyTorch

```
class Inception(nn.Module):
    def __init__(self, in_ch, c1, c3red, c3,
                  c5red, c5, pool):
        super().__init__()

        self.branch1 = nn.Conv2d(in_ch, c1, 1)

        self.branch3 = nn.Sequential(
            nn.Conv2d(in_ch, c3red, 1), nn.ReLU(),
            nn.Conv2d(c3red, c3, 3, padding=1),
        )

        self.branch5 = nn.Sequential(
            nn.Conv2d(in_ch, c5red, 1), nn.ReLU(),
            nn.Conv2d(c5red, c5, 5, padding=2),
        )

        self.branch_pool = nn.Sequential(
            nn.MaxPool2d(3, stride=1, padding=1),
            nn.Conv2d(in_ch, pool, 1),
        )

    def forward(self, x):
        y1 = self.branch1(x)
        y2 = self.branch3(x)
        y3 = self.branch5(x)
        y4 = self.branch_pool(x)
        return torch.cat([y1, y2, y3, y4], dim=1)
```

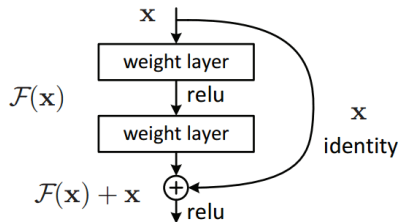
Residual Block

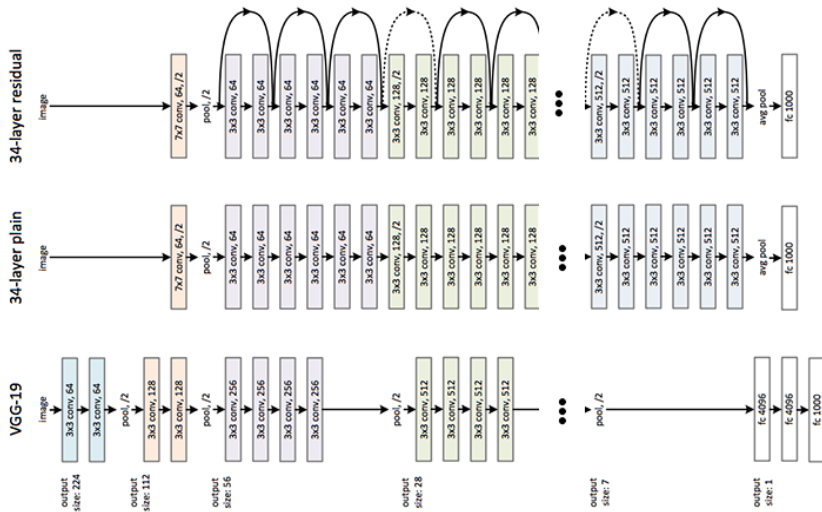
$$y = x + \mathcal{F}(x, W),$$

$$x_{l+1} = x_l + \mathcal{F}_l(x_l).$$

$$x_L = x_l + \sum_{i=l}^{L-1} \mathcal{F}_i(x_i).$$

$$\frac{\partial \mathcal{L}}{\partial x_l} = \frac{\partial \mathcal{L}}{\partial x_L} \left(1 + \frac{\partial}{\partial x_l} \sum_{i=l}^{L-1} \mathcal{F}_i(x_i) \right).$$



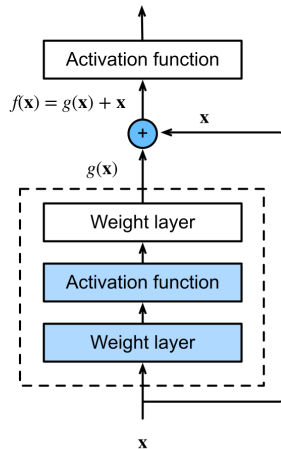
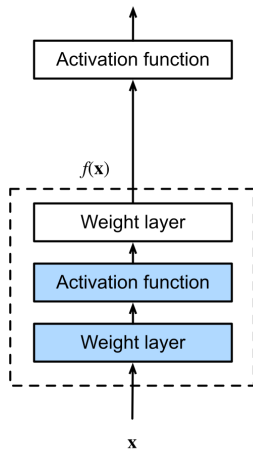


Residual Block: уменьшение размерности

$$\mathcal{P}(x) = \text{Conv}_{1 \times 1}^{s=2}(x),$$

$$\mathcal{P}(x) \in \mathbb{R}^{C_{\text{out}} \times H/2 \times W/2}.$$

$$y = \mathcal{F}(x, W) + \mathcal{P}(x)$$



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

Inception-Resnet

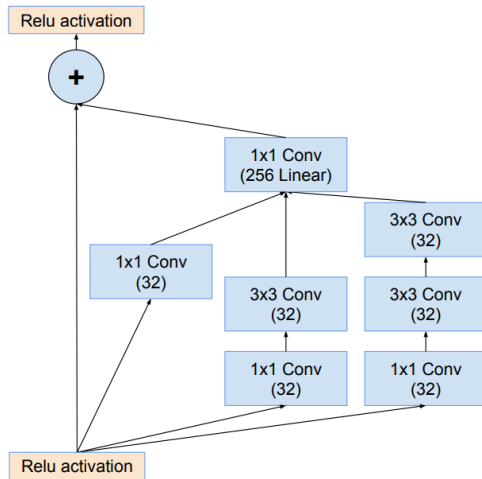
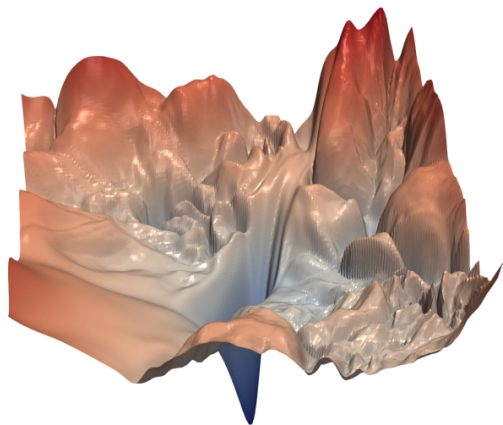
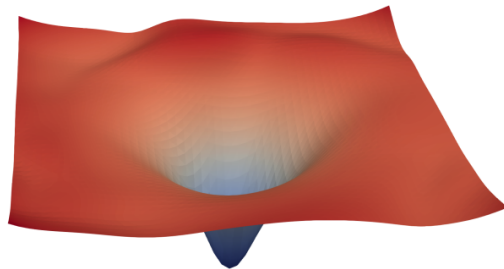


Figure 10. The schema for 35×35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.



(a) without skip connections



(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Работает ли это

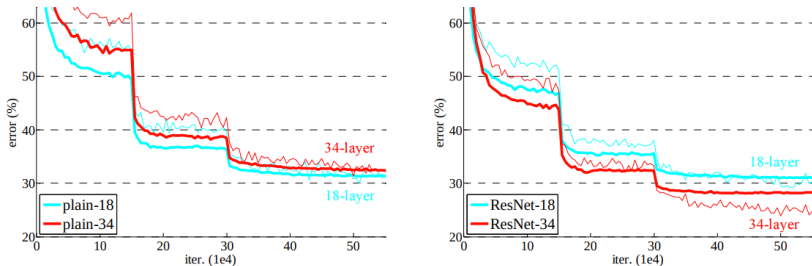


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

<https://arxiv.org/abs/1512.03385>

Работает ли это

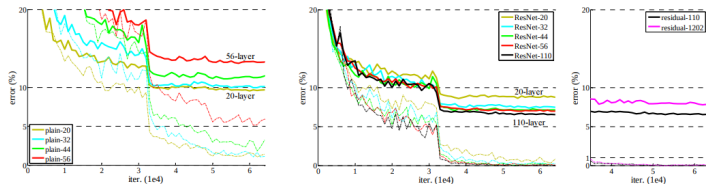


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

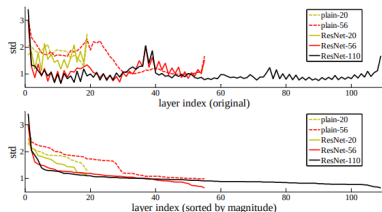


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

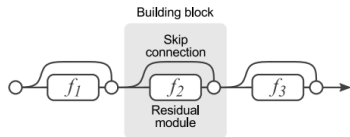
training data	07+12	07++12
test data	VOC 07 test	VOC 12 test
VGG-16	73.2	70.4
ResNet-101	76.4	73.8

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also Table 10 and 11 for better results.

metric	mAP@.5	mAP@[.5, .95]
VGG-16	41.5	21.2
ResNet-101	48.4	27.2

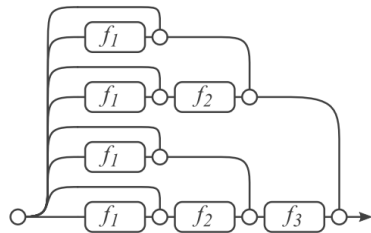
Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also Table 9 for better results.

Почему это работает?



(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

<https://arxiv.org/pdf/1605.06431>

Почему это работает?

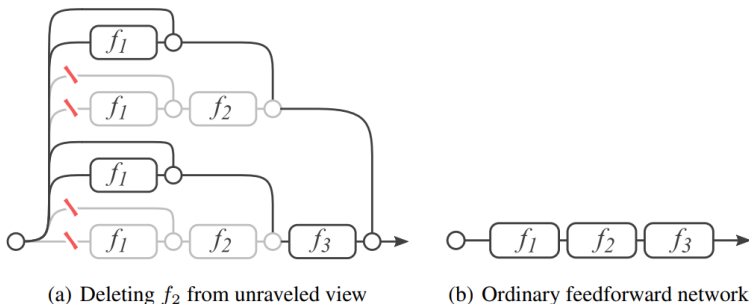


Figure 2: Deleting a layer in residual networks at test time (a) is equivalent to zeroing half of the paths. In ordinary feed-forward networks (b) such as VGG or AlexNet, deleting individual layers alters the only viable path from input to output.

<https://arxiv.org/pdf/1605.06431>

Почему это работает?

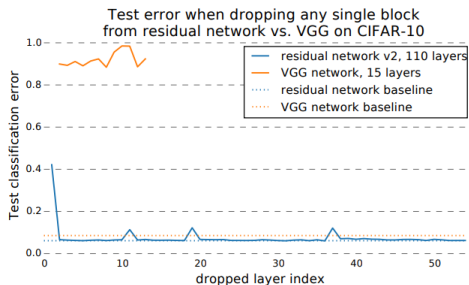


Figure 3: Deleting individual layers from VGG and a residual network on CIFAR-10. VGG performance drops to random chance when any one of its layers is deleted, but deleting individual modules from residual networks has a minimal impact on performance. Removing downsampling modules has a slightly higher impact.

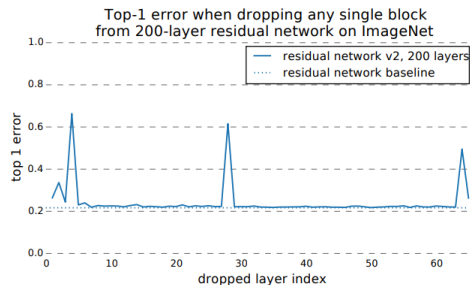


Figure 4: Results when dropping individual blocks from residual networks trained on ImageNet are similar to CIFAR results. However, downsampling layers tend to have more impact on ImageNet.

<https://arxiv.org/pdf/1605.06431>

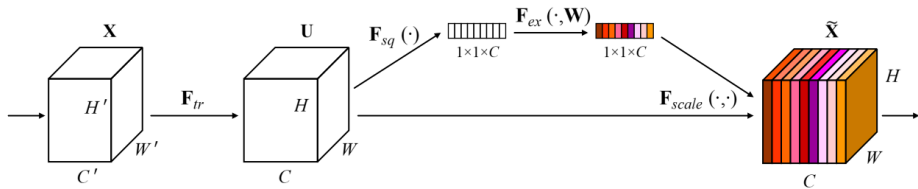


Figure 1: A Squeeze-and-Excitation block.

<https://arxiv.org/pdf/1709.01507>

SENet: внимание к каналам

Свёрточный блок выдаёт

$$\mathbf{U} = F_{\text{tr}}(\mathbf{X}) \in \mathbb{R}^{H \times W \times C}, \quad \mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_C].$$

Squeeze: каждый канал сводится к одному числу:

$$z_c = F_{\text{sq}}(\mathbf{u}_c) = \frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j), \quad \mathbf{z} \in \mathbb{R}^C.$$

Excitation: сеть вычисляет веса каналов:

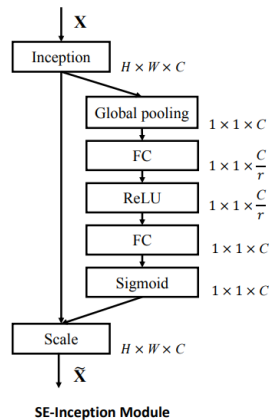
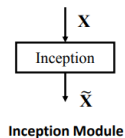
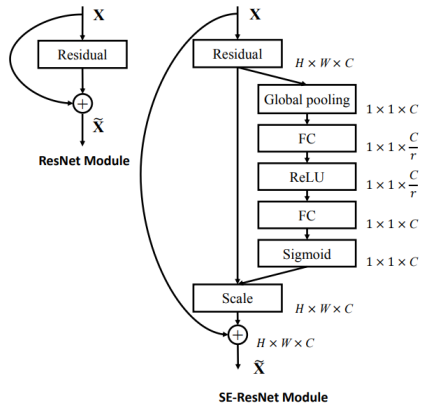
$$\mathbf{s} = F_{\text{ex}}(\mathbf{z}) = \sigma(W_2 \delta(W_1 \mathbf{z})),$$

$$W_1 \in \mathbb{R}^{\frac{C}{r} \times C}, \quad W_2 \in \mathbb{R}^{C \times \frac{C}{r}}.$$

Scale: признаки перенастраиваются:

$$\tilde{\mathbf{u}}_c = s_c \mathbf{u}_c, \quad s_c \in (0, 1).$$

SE-ResNet и SE-Inception



SE-блок: реализация на PyTorch

```
import torch
from torch import nn

class SEBlock(nn.Module):
    def __init__(self, channels, reduction=16):
        super().__init__()
        hidden = max(1, channels // reduction)
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(channels, hidden),
            nn.ReLU(),
            nn.Linear(hidden, channels),
            nn.Sigmoid()
        )

    def forward(self, x):
        b, c, _, _ = x.shape
        weights = self.fc(self.pool(x))
        return x * weights.view(b, c, 1, 1)
```

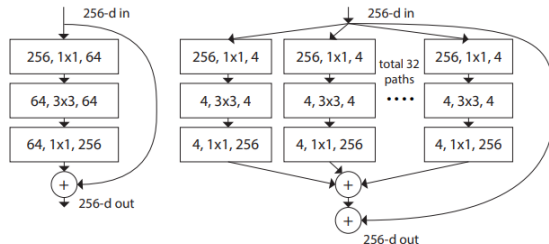


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

<https://arxiv.org/pdf/1611.05431>

ResNeXt: число параллельных преобразований

В ResNet: $\mathbf{y} = \mathbf{x} + F(\mathbf{x})$.

В ResNeXt: $\mathbf{y} = \mathbf{x} + \sum_{k=1}^C F_k(\mathbf{x})$

Cardinality C — число параллельных ветвей в остаточном блоке. Это отдельный параметр архитектуры наряду с глубиной сети и шириной слоёв.

На практике параллельные ветви удобно реализовать одной **групповой свёрткой**:

$$\text{Conv}_{3 \times 3} \longrightarrow \text{GroupConv}_{3 \times 3}(C).$$

Например, обозначение **ResNeXt-50 (32×4d)** означает сеть из 50 слоёв с $C = 32$ группами; каждая группа обрабатывает 4 канала внутри групповой свёртки.

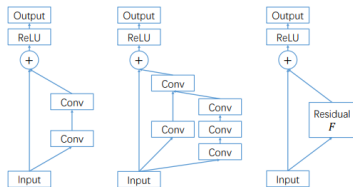


Figure 3: **Left:** residual unit of ResNet [9]. **Middle:** type-B Inception residual unit of Inception-ResNet-v2 [25]. **Right:** abstract residual unit structure where the residual block is denoted by F .

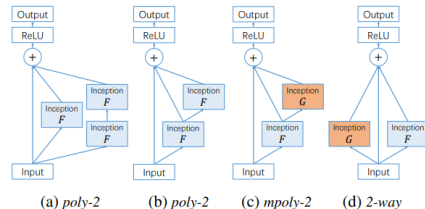


Figure 4: Examples of PolyInception structures.

<https://arxiv.org/pdf/1611.05725>

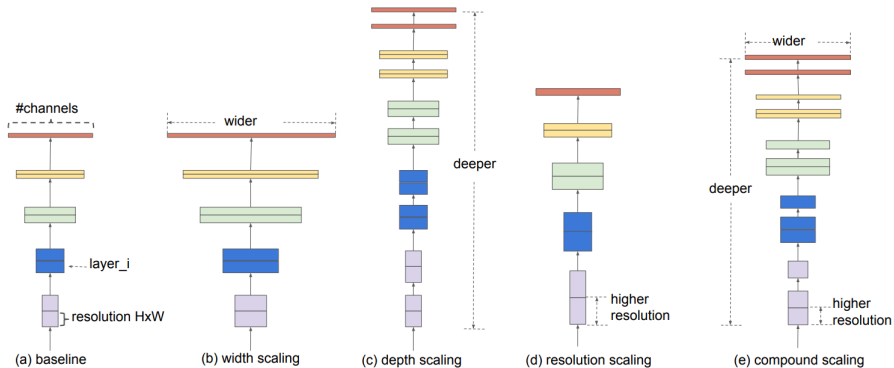


Figure 2. Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

<https://arxiv.org/abs/1905.11946>

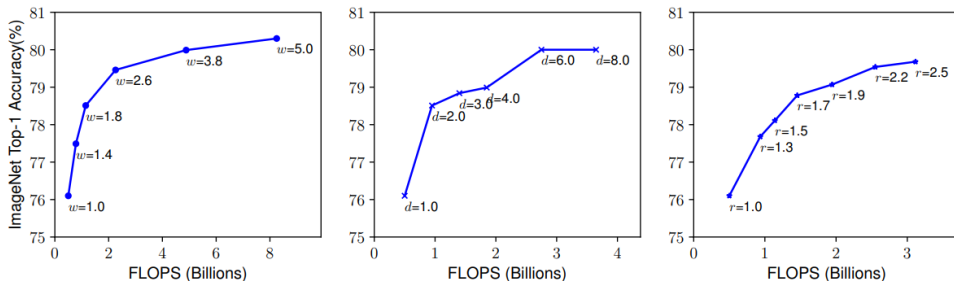


Figure 3. Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients. Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single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

<https://arxiv.org/abs/1905.11946>

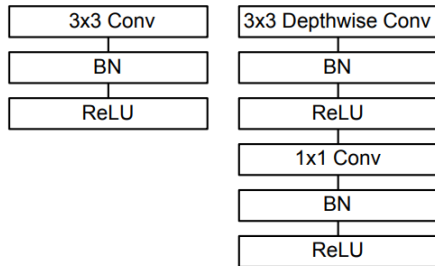


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

<https://arxiv.org/pdf/1704.04861>

Depthwise separable convolution

- 1 **Depthwise:** отдельный фильтр $K \times K$ для каждого из C_{in} каналов.
- 2 **Pointwise:** свёртка 1×1 объединяет каналы и создаёт C_{out} выходных каналов.

Обычная свёртка: $HWK^2 C_{\text{in}} C_{\text{out}}$,

Depthwise + pointwise: $HW(K^2 C_{\text{in}} + C_{\text{in}} C_{\text{out}})$.

$$\frac{\text{Depthwise + pointwise}}{\text{Обычная свёртка}} = \frac{1}{C_{\text{out}}} + \frac{1}{K^2}$$

Например, при $K = 3$ и $C_{\text{out}} = 64$: $\frac{1}{64} + \frac{1}{9} \approx 0,127$.