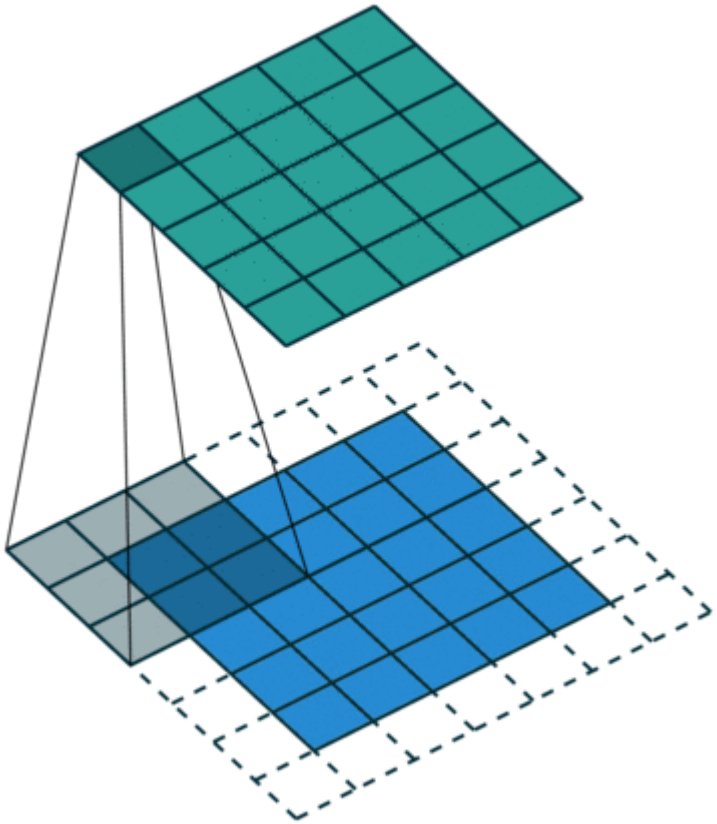
Продолжение введения в CNN

CNN – сверточные нейронные сети (convolutional neural network)

Recap: свертки



1	0	1
0	1	0
1	0	1

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

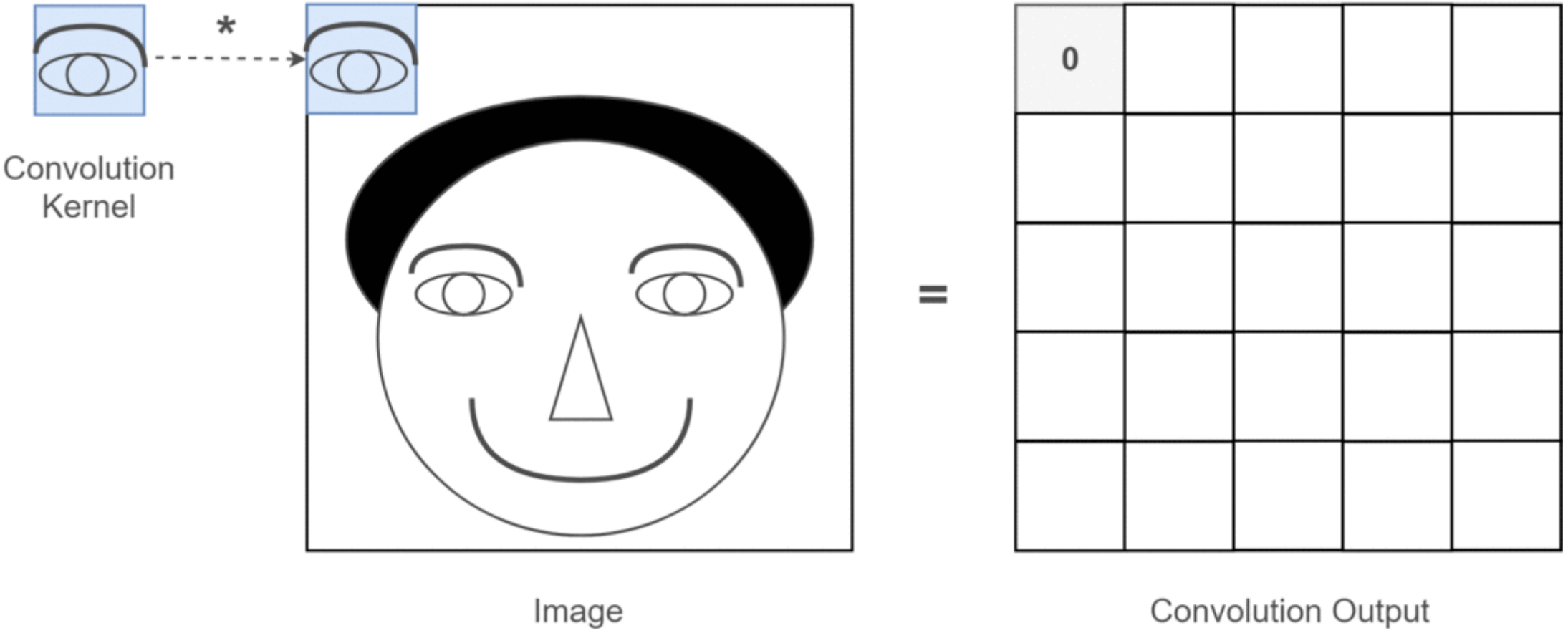
4		

Convolved Feature

Что следует помнить:

1. **Одно** ядро свертки на **один** слой. То есть, если получаем из 3 каналов 70 каналов, будет 70 сверточных слоев и суммарно 70 ядер свертки.
2. Каждый слой отвечает за один паттерн. Чем больше слоев делаем, тем больше признаков можем выделить.
3. *Ядра свертки* – **параметры** модели, т.е. они задаются **случайным образом** и впоследствии **меняются** в **процессе обучения**.

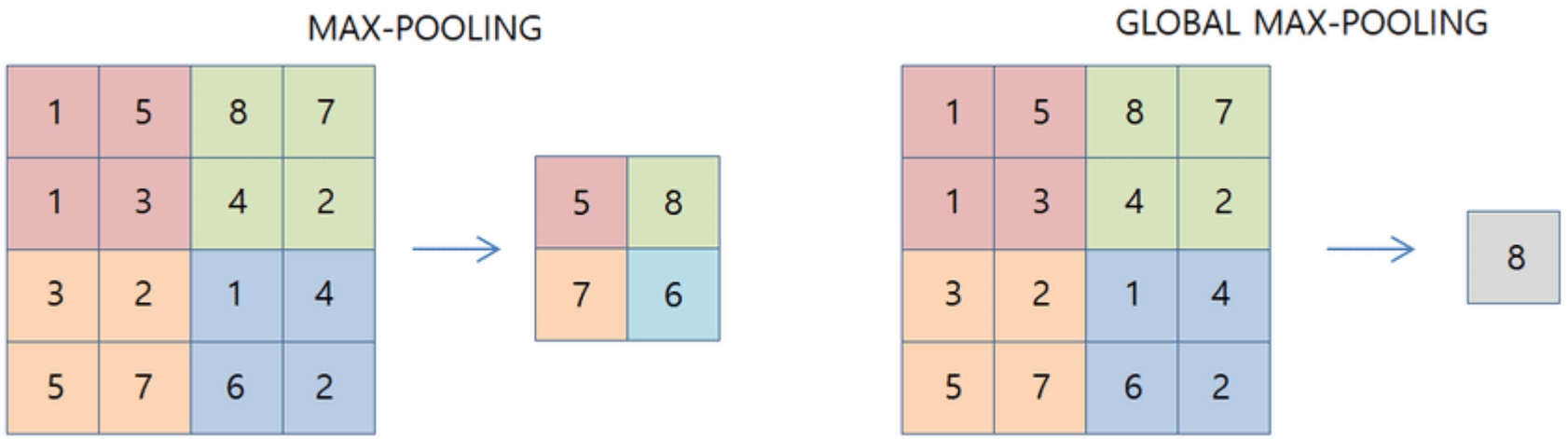
Recap: пример свертки в последних сверточных слоях

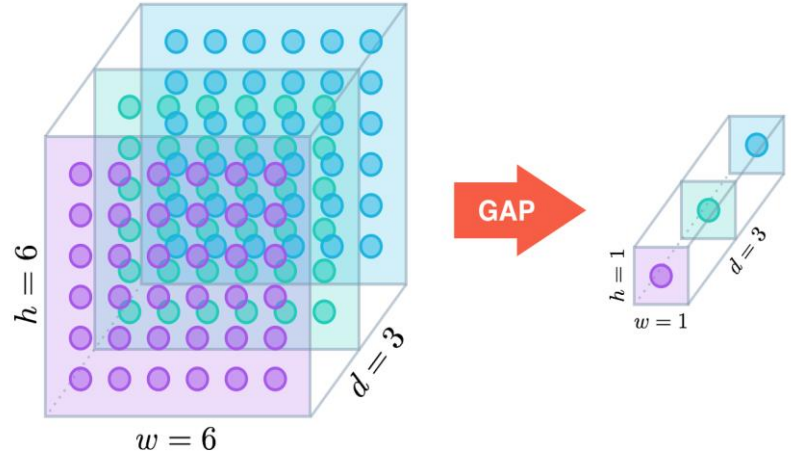


Recap: MaxPool

- Нужен для снижения размерности
- Вычленяет самые главные признаки

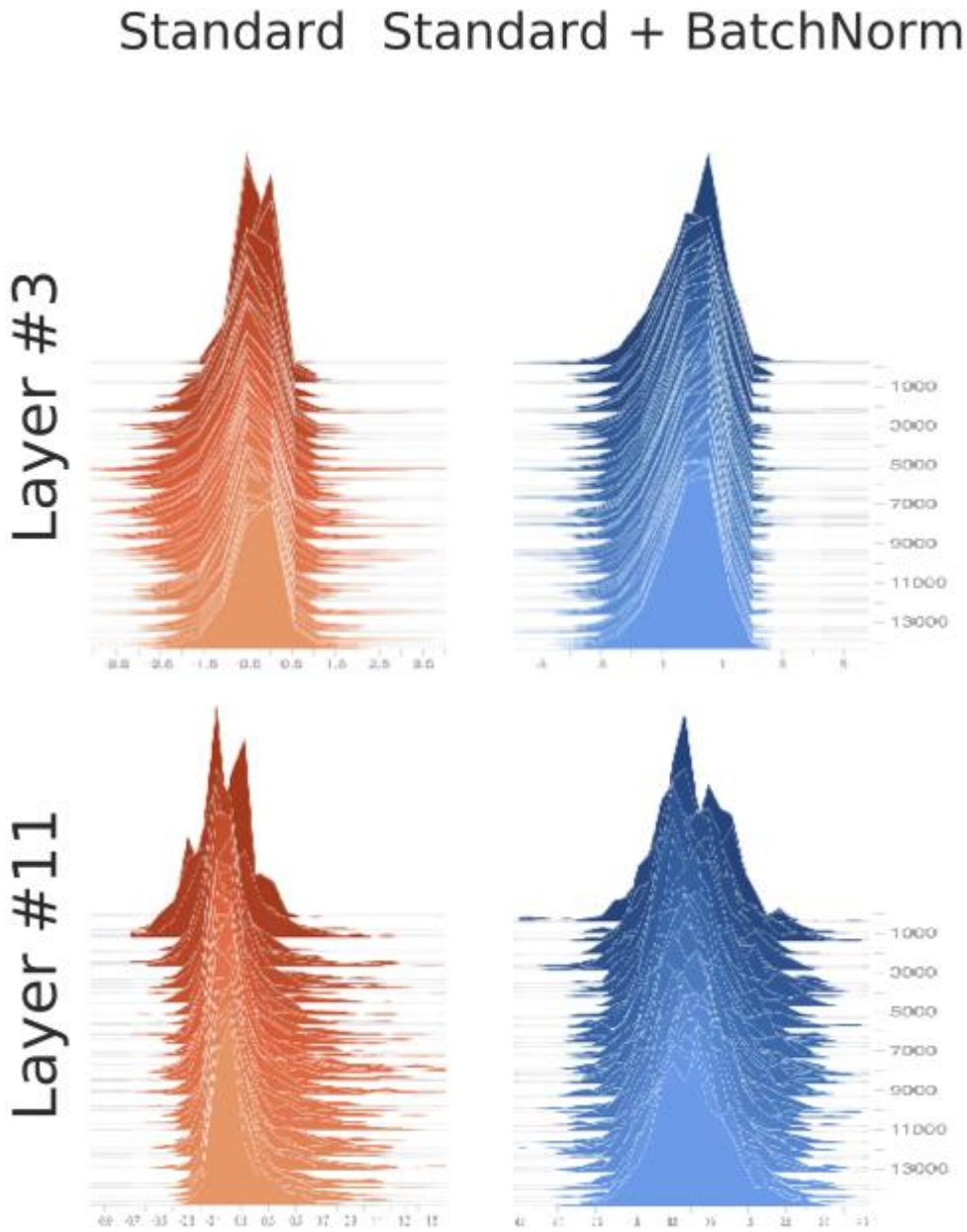
Новая информация – есть глобальный пулинг



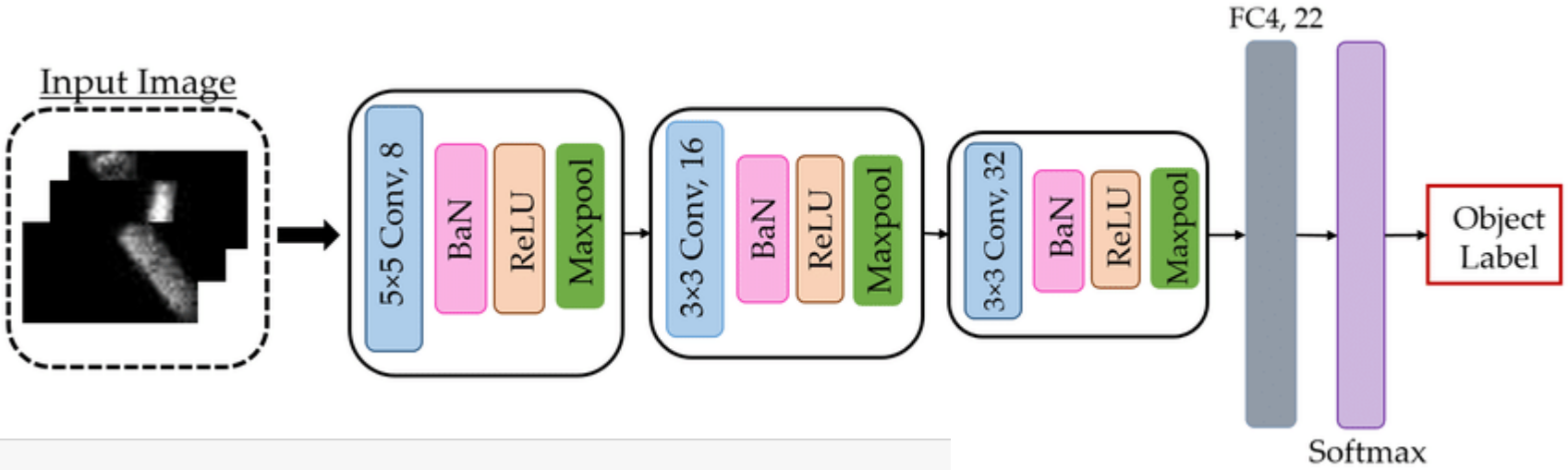


Батч нормализация

- Заключается в придании **нормального вида** распределению признаков
- Помогает быстрее сходиться алгоритму оптимизации



Батч нормализация: куда ставить?



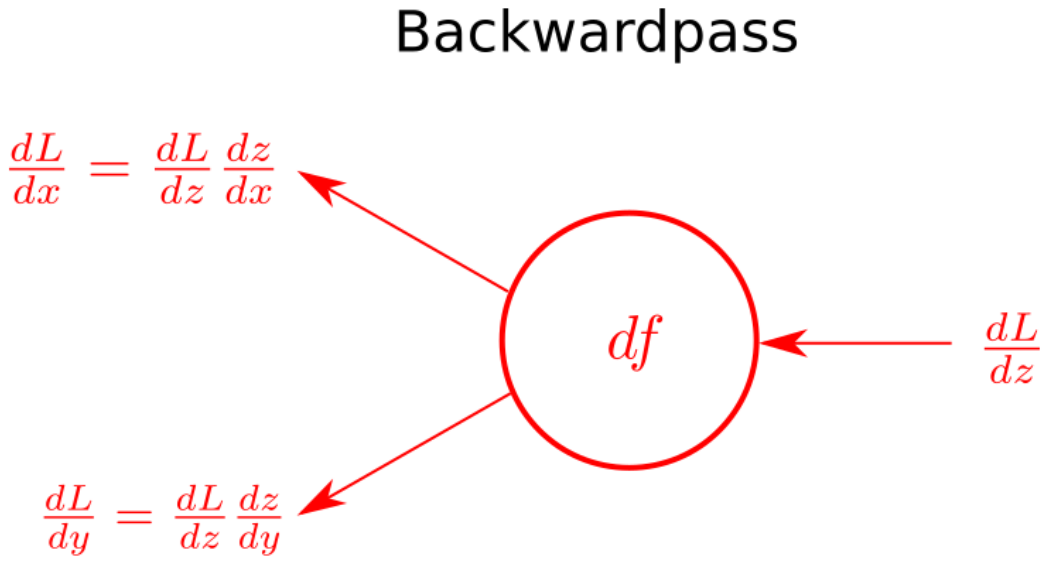

```
1 from torch import nn
```

```
1 nn.BatchNorm1d()
```

```
Init signature:
nn.BatchNorm1d(
    num_features: int,
    eps: float = 1e-05,
    momentum: float = 0.1,
    affine: bool = True,
    track_running_stats: bool = True,
    device=None,
    dtype=None,
) -> None
```

Если сети глубокие, то есть проблемы

Проблема – затухающий градиент



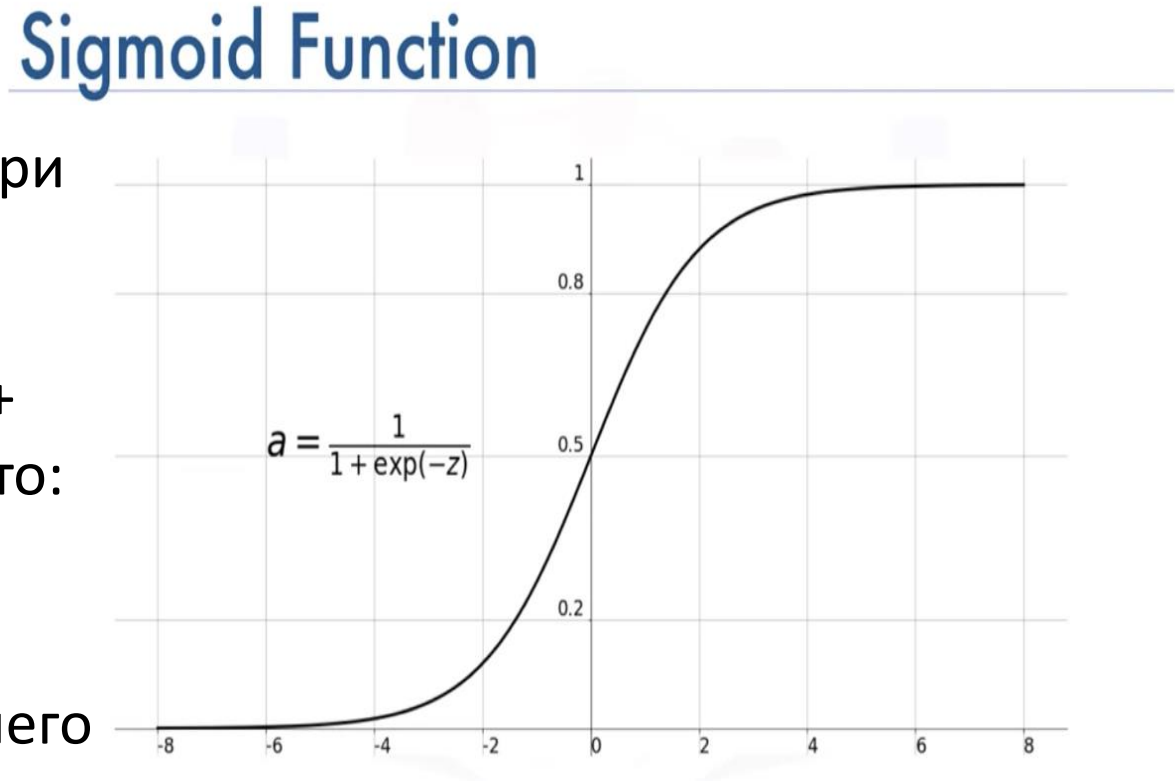
Пример: 6 слоев: A, B, C, D, E, F.
На выходе получаем лосс L.

Для последнего слоя $\frac{\mathrm{dL}}{dF} = \frac{dL}{dF}$ все просто. Для E слоя $\frac{dL}{dE} = \frac{dL}{dF} \cdot \frac{dF}{dE}$.

Для A слоя $\frac{dL}{dA} = \frac{dL}{dF} \cdot \frac{dF}{dE} \cdot \frac{dE}{dD} \cdot \frac{dD}{dC} \cdot \frac{dC}{dB} \cdot \frac{dB}{dA}$

Затухающий градиент

Для A слоя $\frac{dL}{dA} = \frac{dL}{dF} \cdot \frac{dF}{dE} \cdot \frac{dE}{dD} \cdot \frac{dD}{dC} \cdot \frac{dC}{dB} \cdot \frac{dB}{dA}$

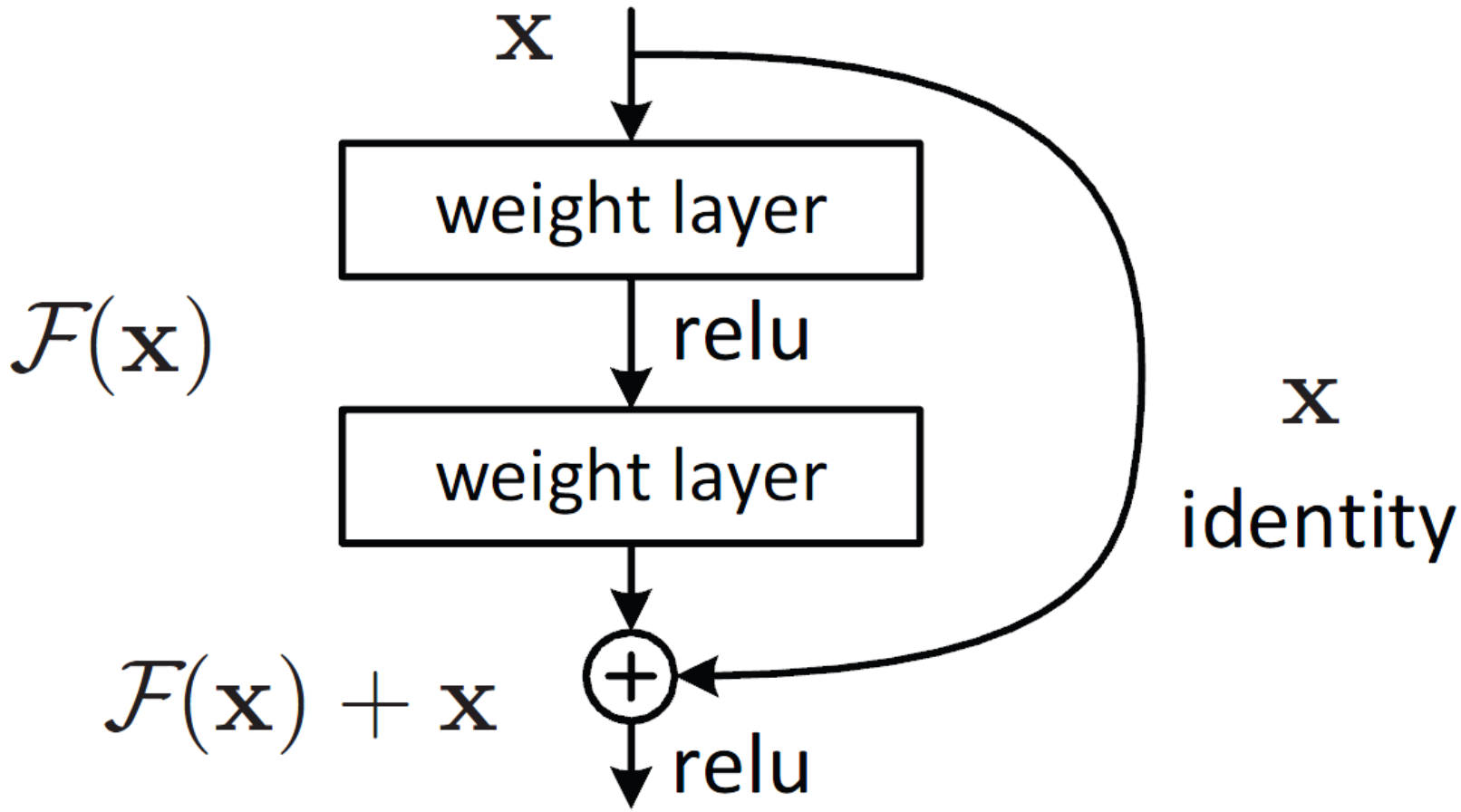


Если функция активации – сигмоида, то при $|x|>4$ градиент стремится к 0.

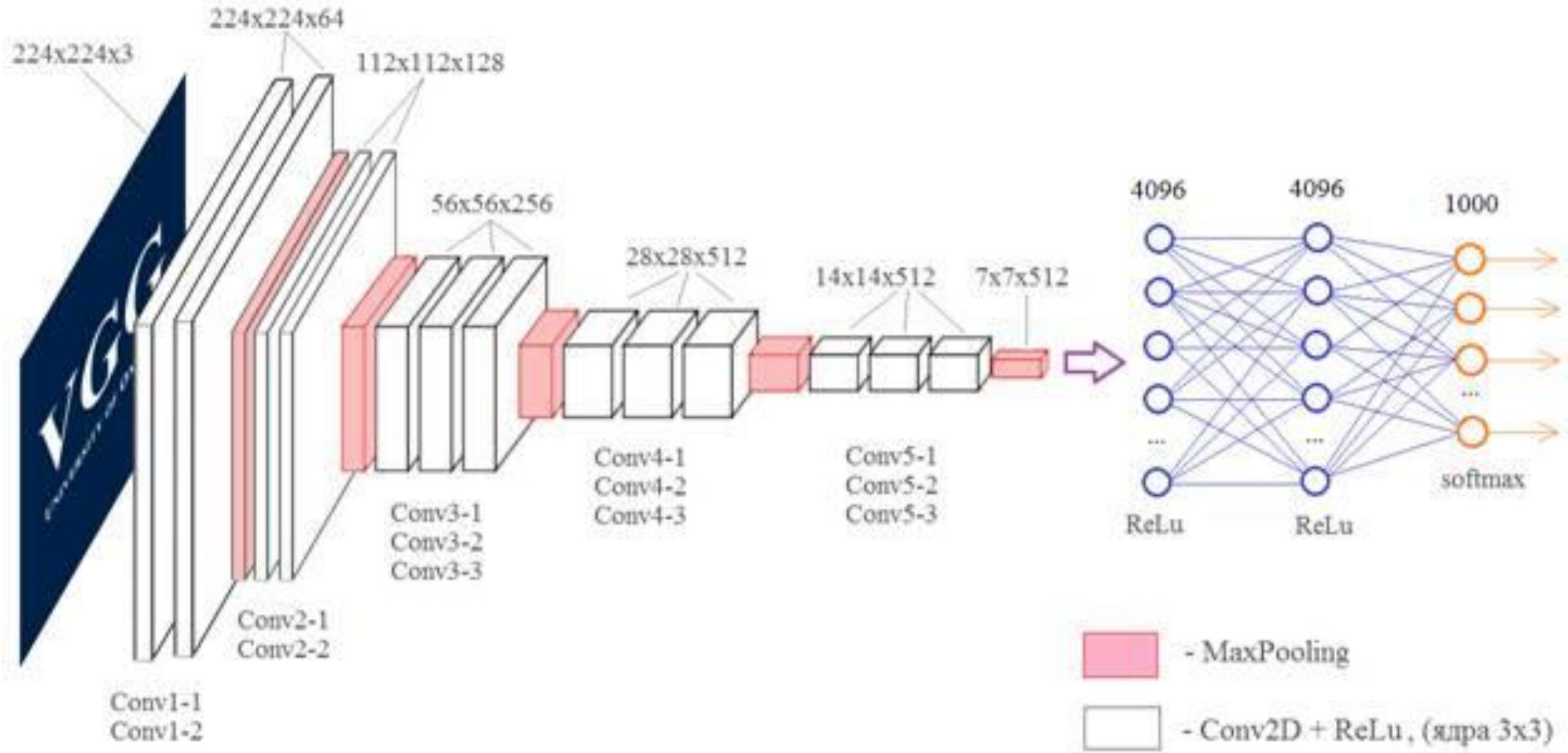
Если каждый слой будет линейный слой + сигмоида, то возможна такая ситуация, что:

Для A слоя $\frac{dL}{dA} = 0.1 \cdot 0.1 \cdot 0.1 \cdot 0.1 \cdot 0.1 \cdot 0.1 = 10^{-6}$ значение градиента, дошедшего до слоя A. То есть, фактически первые слои почти не обучаются.

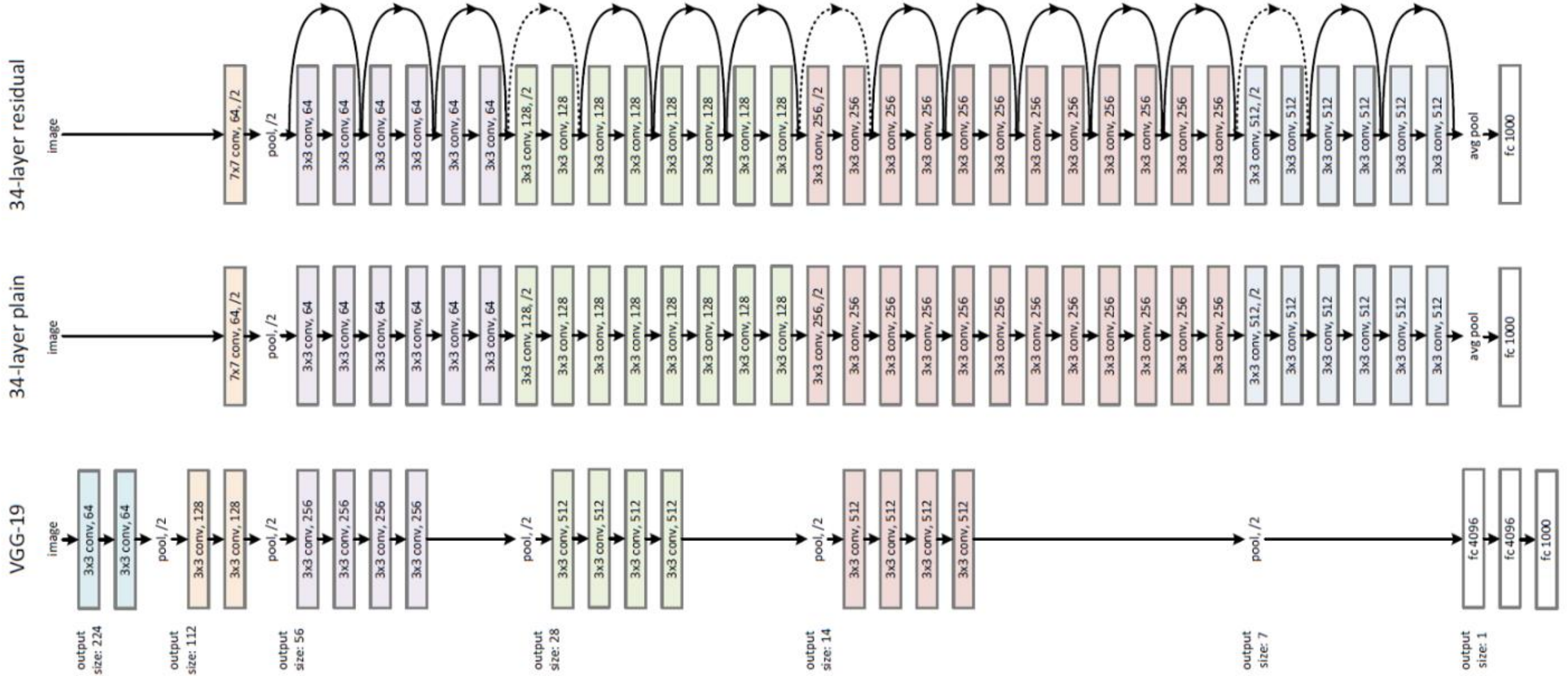
Как бороться с затухающим градиентом?



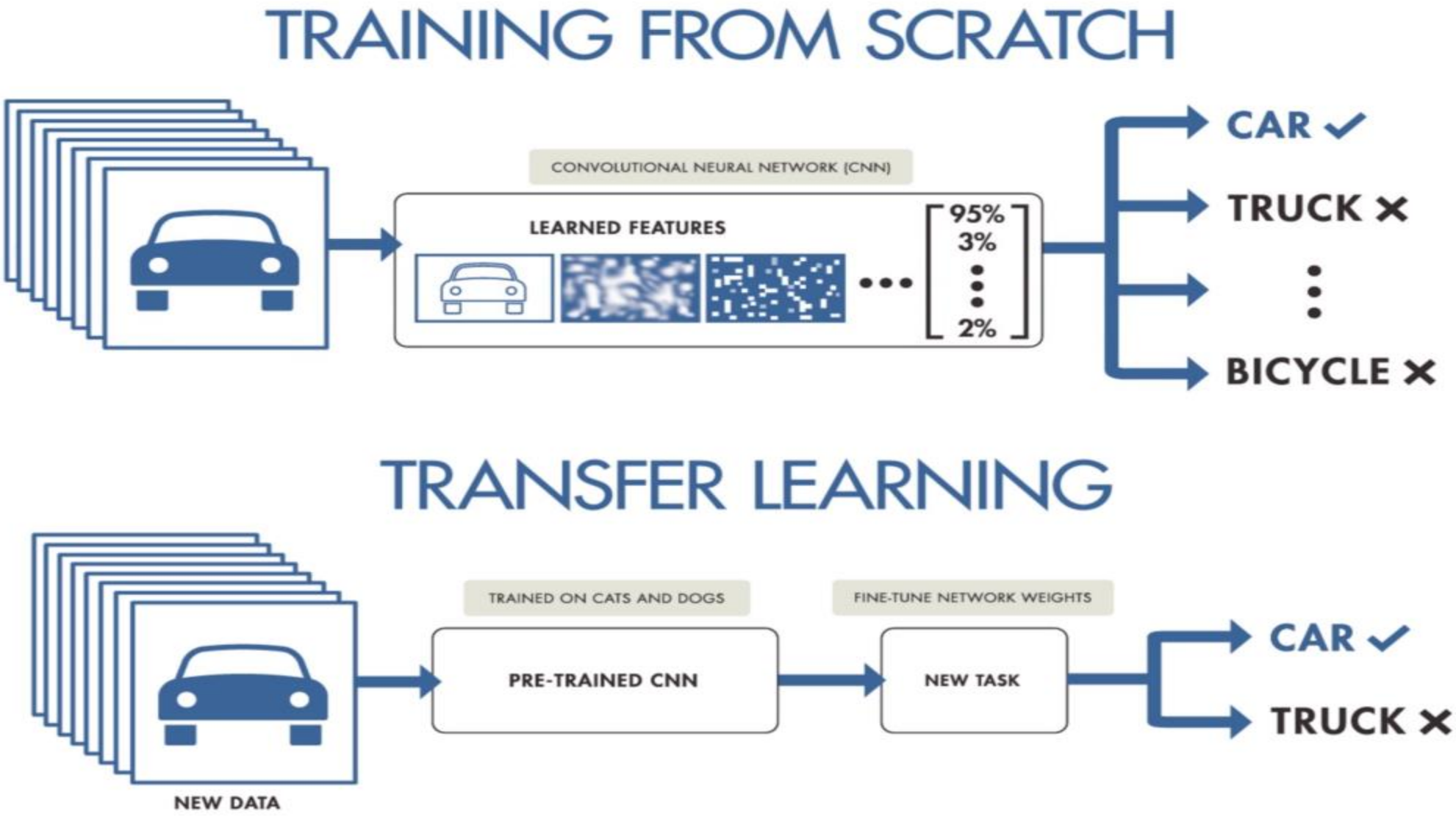
Архитектуры нейросетей (VGG)



Архитектуры нейросетей (ResNet)

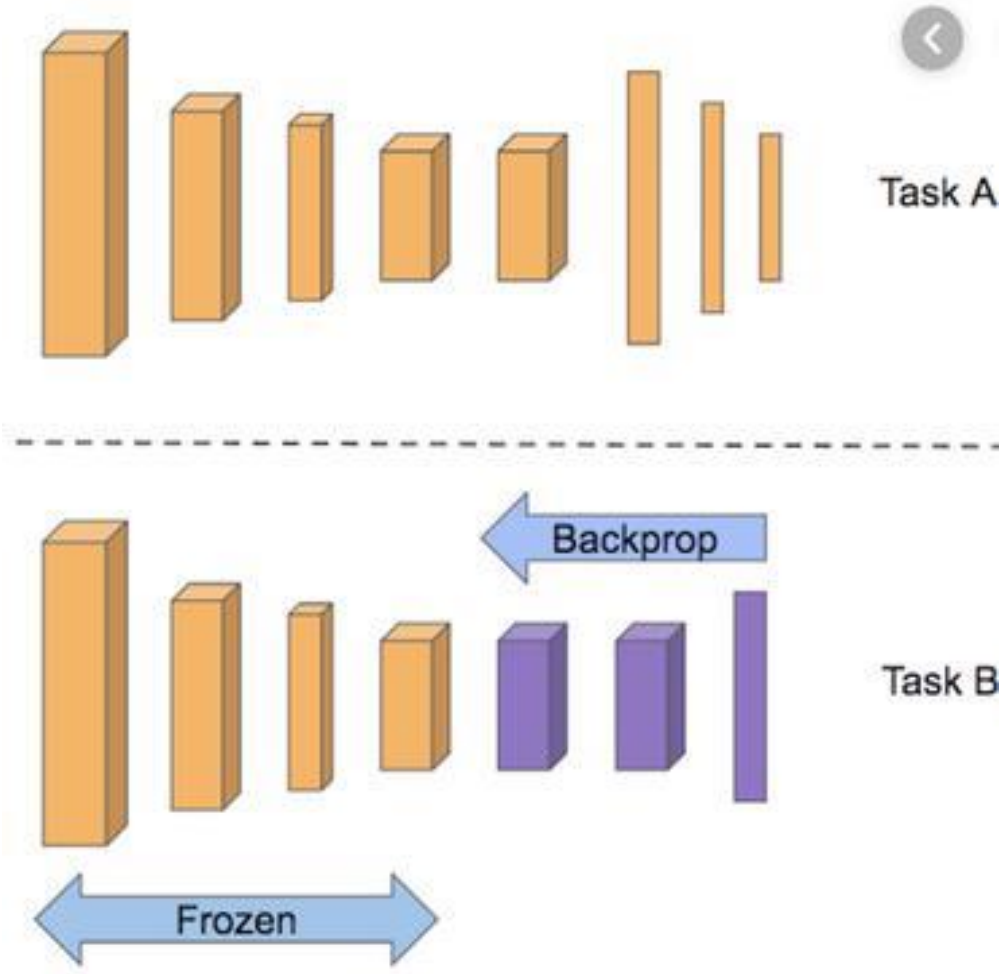


Transfer Learning



Как использовать Transfer Learning

1. Замораживаем слои все слои исходной нейросети.
2. Удаляем последние слои
3. Добавляем нужные для нашей задачи слои (например, в исходной задаче было 1000 классов, а хотим предсказывать 2)
4. Тренируем только последние слои модифицированной нейросети.



Пример кода из документации

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

Пишем код