

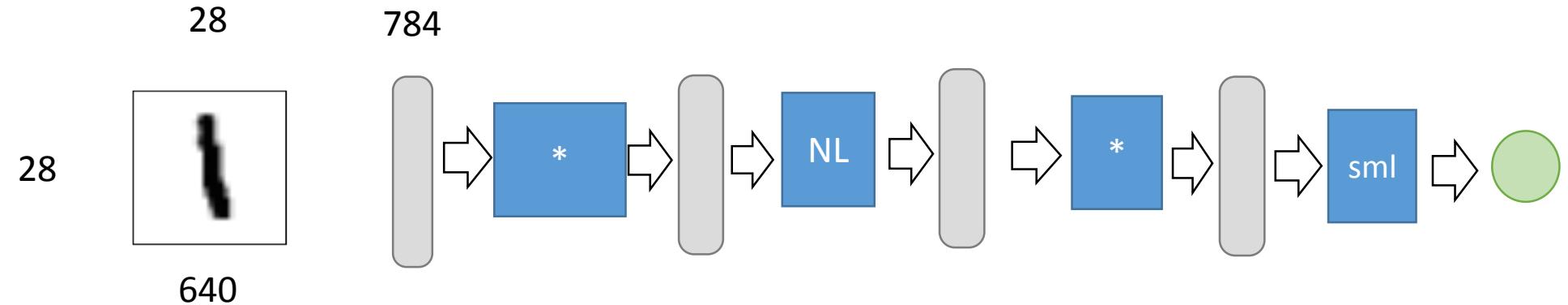
# Сверточные нейронные сети

Куликов В.А. в соавторстве с Лемпицким В.С.

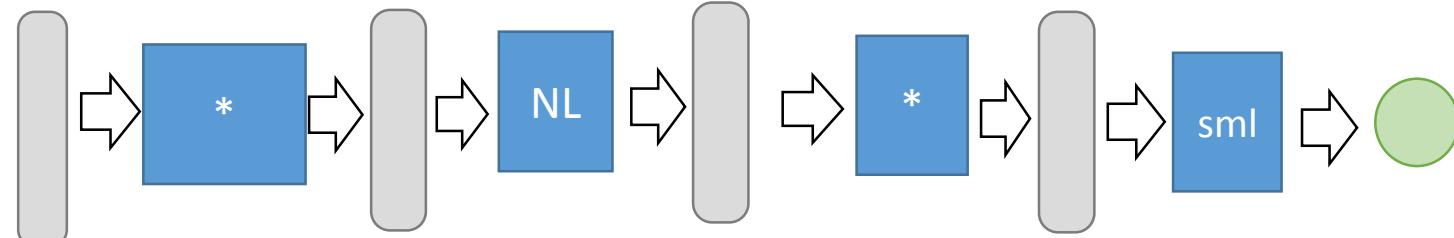


[v.kulikov@skoltech.ru](mailto:v.kulikov@skoltech.ru)

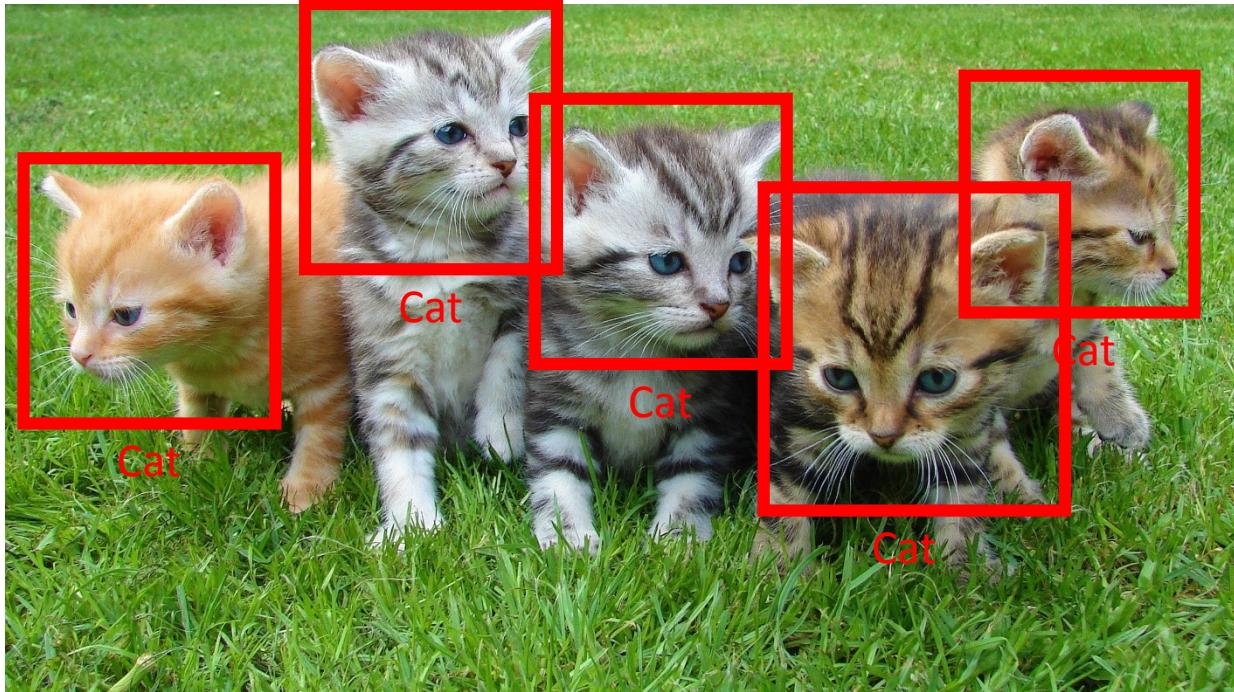
# Проблема классификации больших картинок с помощью нейронной сети



1966080 элементов во входном векторе!

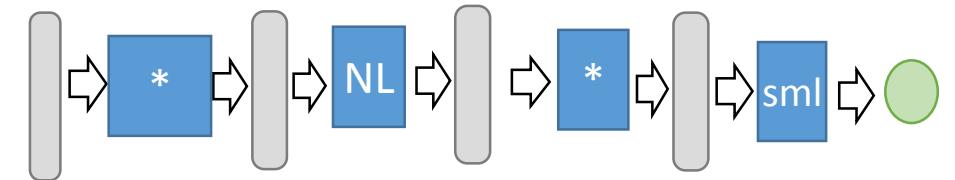


# Инвариантность к смещению и цвету



1. Положение на изображении не влияет на наличие кота
2. Зачем тренировать нейроны отдельные для каждого пикселя?
3. Возможно повторное использование параметров модели
4. Хотим использовать пространственные связи

Идея – применять нейронную сеть к маленькому фрагменту изображения



Теперь для каждого фрагмента применяется классификатор, но он способен находить только объекты определенного размера

# Двумерная свертка

$$V(x, y) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} K(i - x + \delta, j - y + \delta) * I(i, j)$$

$$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

# Двумерная свертка

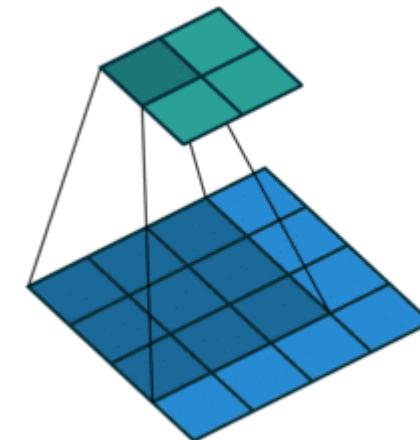
$$V(x, y) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} K(i - x + \delta, j - y + \delta) * I(i, j)$$

*I*

1	-1	0	0	1
1	1	0	1	1
3	2	0	0	0
1	1	0	1	1
1	0	1	1	0
1	2	2	0	0

*K*

1	0
0	-1



*V?*

0	-1	-1	-1
-1	1	0	1
2	2	-1	-1
1	0	-1	1
-1	-2	1	1

# Двумерная свертка в обработке изображений



Фильтр Превитт

1	0	-1
1	0	-1
1	0	-1

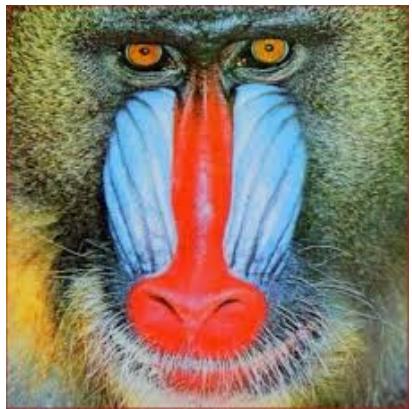


1	1	1
0	0	0
-1	-1	-1

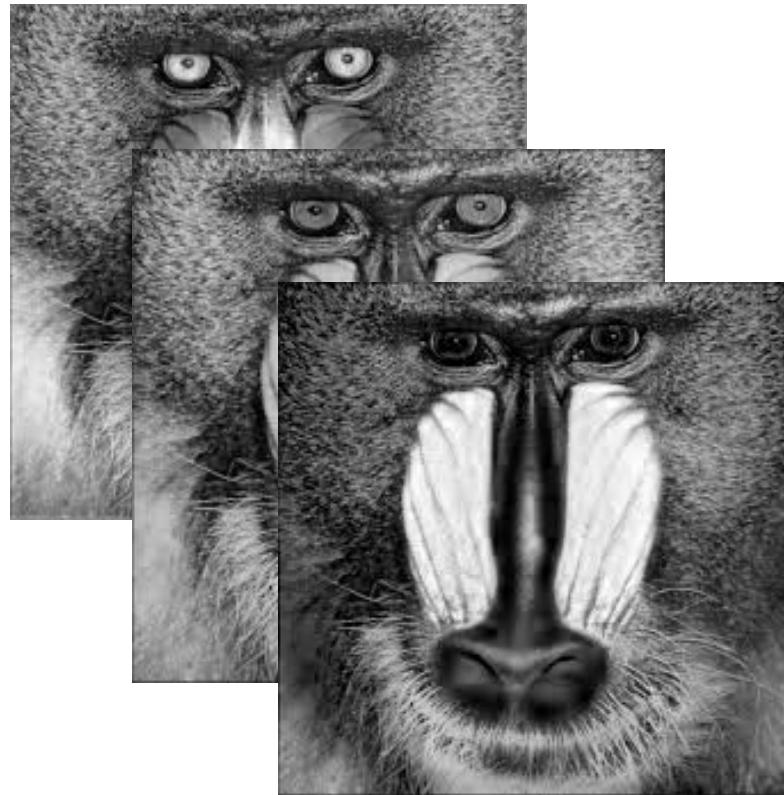


Модуль результата

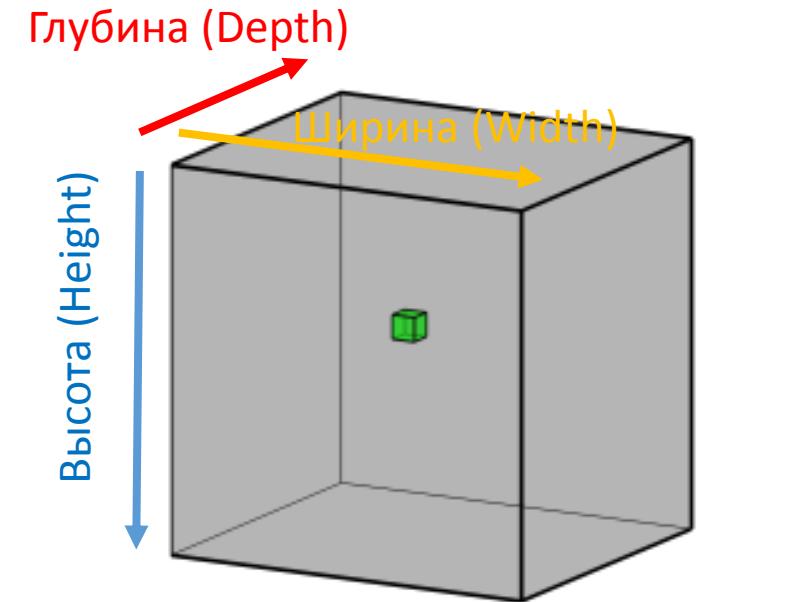
# Что если изображение цветное?



$S=3$



$$V(x, y) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^S K(s, i - x + \delta, j - y + \delta) * I(s, i, j)$$



1	1	1		
0		1	1	1
-1		0	0	1
		1	1	1
-1		0	0	0
		-1	-1	-1

# Реализация conv2D библиотеках

- `torch.nn.functional.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)`
- input 4D тензор: [количество изображений, количество каналов у изображения, ширина, высота]
- Weight (веса) – множество фильтров заданное в виде 4D тензора: [количество фильтров, количество входных каналов, ширина фильтра, высота фильтра]
- Выход 4D тензор: [количество изображений, количество фильтров, ~ширина, ~высота]

# Применение свертки в нейронной сети

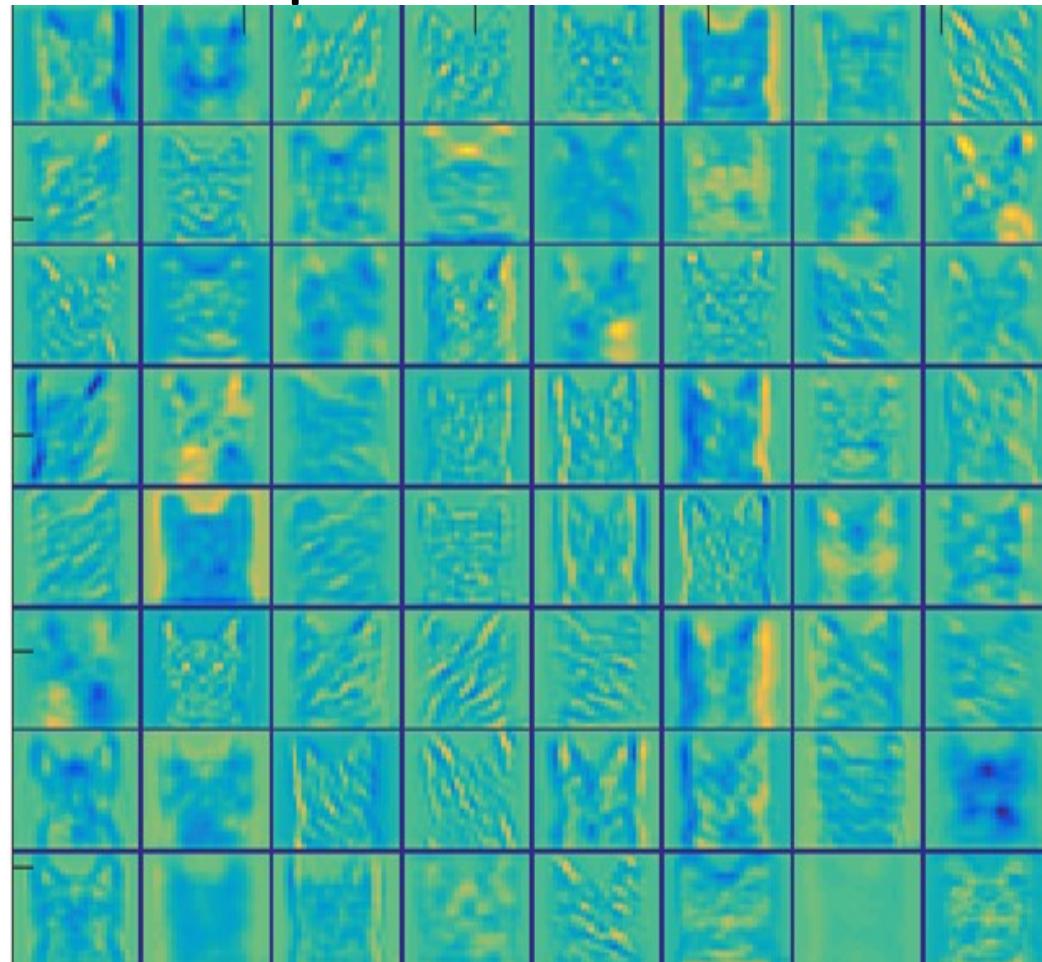


Сворачиваем с фильтрами

$64 \times 3 \times 5 \times 5$

$t \times s \times 2 \delta + 1 \times 2 \delta + 1$

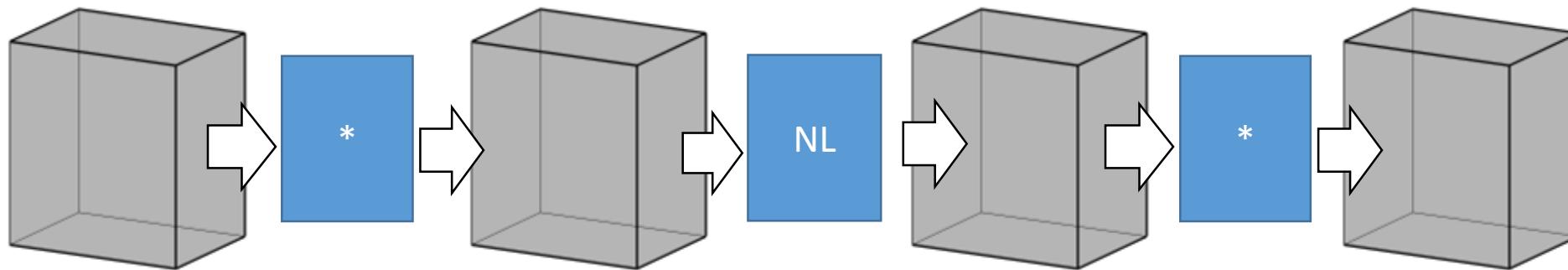
$$V(x, y, t) = \sum_{i=x-\delta}^{x+\delta} \sum_{j=y-\delta}^{y+\delta} \sum_{s=1}^S K(t, s, i - x + \delta, j - y + \delta) * I(s, i, j) + \beta(t) \leftarrow \text{bias}$$



# Свойства свертки

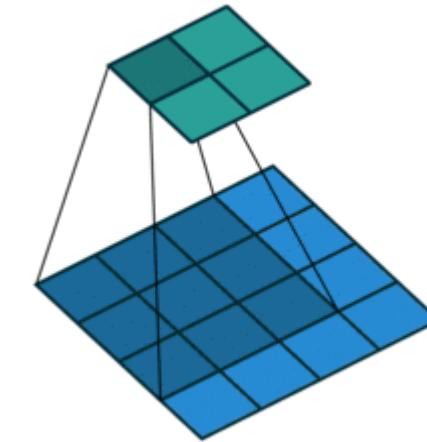
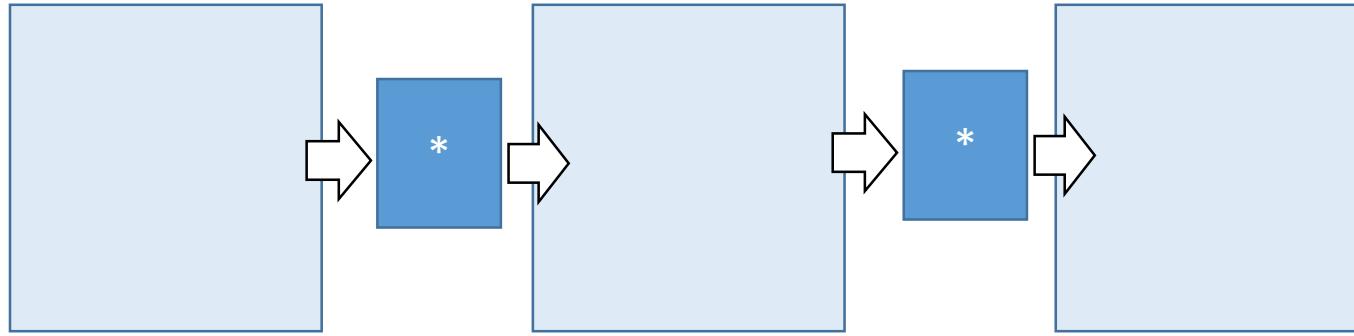
- Свертка линейна
  - Дистрибутивность по сложению  $(f + g) \otimes h = f \otimes h + g \otimes h$
  - Умножение на скаляр  $f \otimes (\alpha * h) = \alpha(f \otimes h)$
- Уменьшение числа параметров
  - Если полно связанная сеть для изображения и 256x256, чтобы получить 4 выходные карты: 262144 параметров
  - Свертка 4x1x5x5 – 100 параметров!
  - Сокращение числа параметров в 2621 раз!

# Последовательно использование нескольких сверток



- Т.к. свертка линейная операция надо вставлять нелинейности

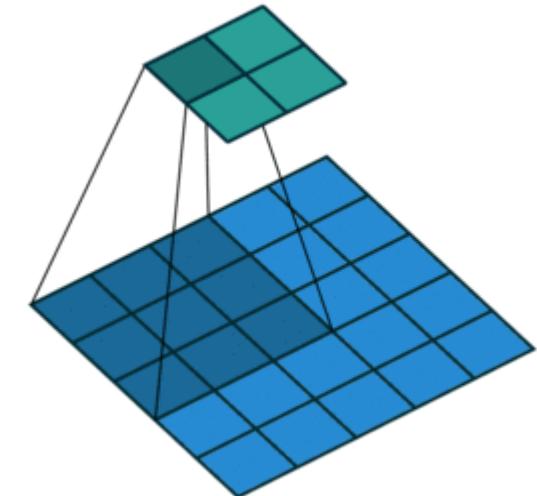
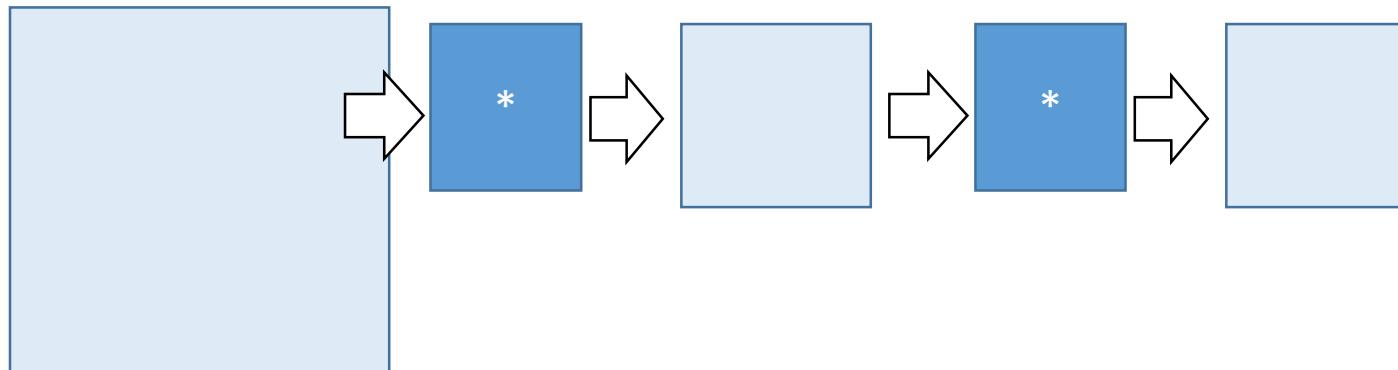
# Поле восприятия сети



- Пусть фильтр имеет размер  $7 \times 7$ 
  - Поле восприятия имеет размер  $7 \times 7$  после первой свертки
  - Какое поле восприятия сети после второй свертки?
  - Ответ:  $13 \times 13$

# Увеличение поля восприятия со stride (сдвиг)

```
torch.nn.functional.conv2d(input, weight, bias=None, stride=1,  
padding=0, dilation=1, groups=1)
```

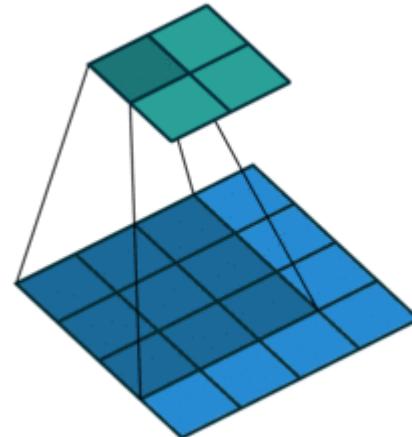


- В ранних слоях нейронных сейте  $\text{stride} > 1$
- Если  $\text{stride} > 1$  пространственное разрешение уменьшается (downscaling)
- Если  $\text{stride} > 1$  поле восприятия увеличивается
- Какое поле восприятия у сети из двух сверточ 7x7 если первая со сдвигом 2, вторая со сдвигом 1

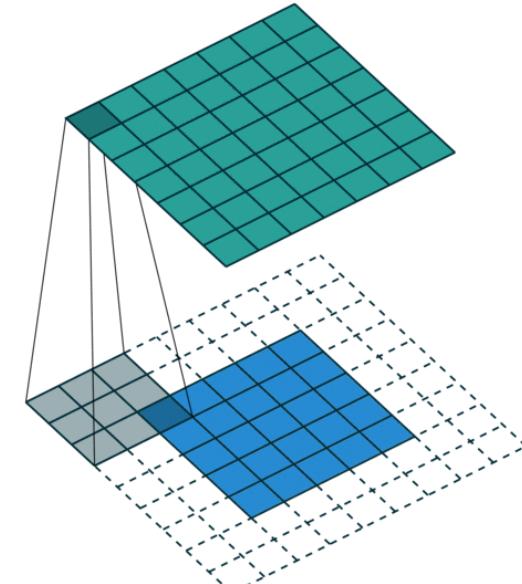
# Границные эффекты (paddings)

```
torch.nn.functional.conv2d(input, weight, bias=None, stride=1,  
padding=0, dilation=1, groups=1)
```

- Valid padding (padding=0)
- Неравный вклад элементов
- Сложно следить за размером данных в сети



- Same padding (padding>0)
- Решает проблему неравного вклада элементов
- Достраивает границу нулями



# Двумерная свертка со сдвигом и границей

0	1	2
2	2	0
0	1	2

0 <sub>2</sub>	0 <sub>0</sub>	0 <sub>1</sub>	0	0	0	0	0
0 <sub>1</sub>	2 <sub>0</sub>	2 <sub>0</sub>	3	3	3	0	0
0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>1</sub>	3	0	3	0	0
0	2	3	0	1	3	0	0
0	3	3	2	1	2	0	0
0	3	3	0	2	3	0	0
0	0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

# Другие варианты решения граничных эффектов

- Расширение краевыми значениями

1	1	1	-1	0	0	1	1	1
1	1	1	-1	0	0	1	1	1
1	1	1	-1	0	0	1	1	1
1	1	1	1	0	1	1	1	1
3	3	3	2	0	0	0	0	0
1	1	1	1	0	1	1	1	1
1	1	1	0	1	1	0	0	0
1	1	1	2	2	0	0	0	0
1	1	1	2	2	0	0	0	0
1	1	1	2	2	0	0	0	0

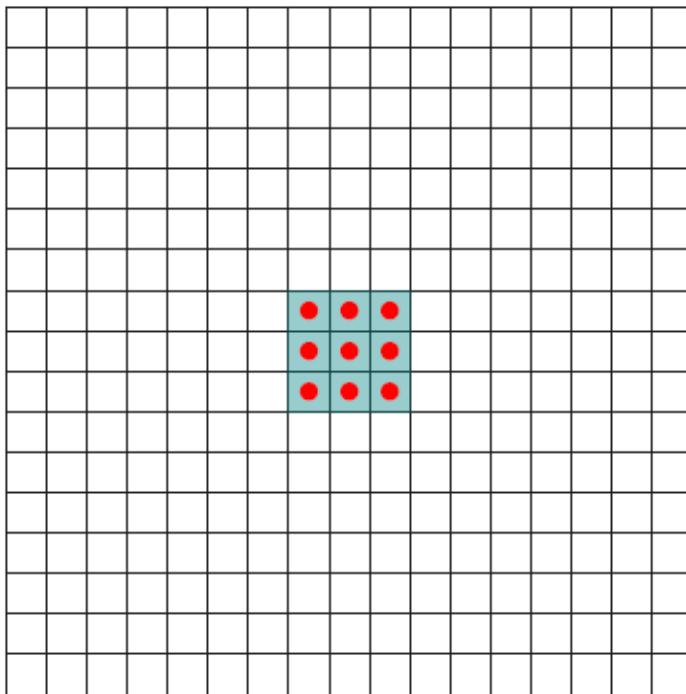
- Зеркальное отражение

1	1	1	1	0	1	1	1	1
-1	1	1	-1	0	0	1	1	0
-1	1	1	-1	0	0	1	1	0
1	1	1	1	0	1	1	1	1
2	3	3	2	0	0	0	0	0
1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	0	0	1
2	1	1	2	2	0	0	0	0
2	1	1	2	2	0	0	0	0
0	1	1	0	1	0	0	0	1

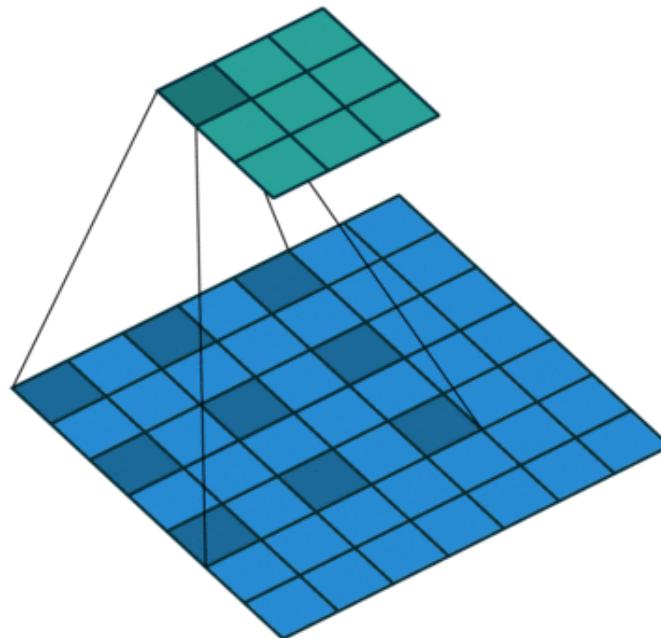
# Разреженная свертка (dilated)

```
torch.nn.functional.conv2d(input, weight, bias=None, stride=1,  
padding=0, dilation=1, groups=1)
```

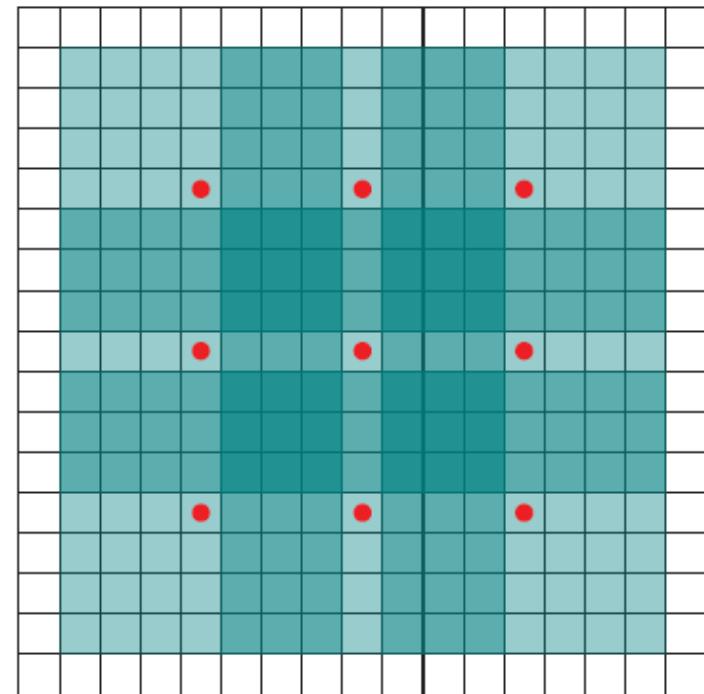
dilation=1



dilation=2



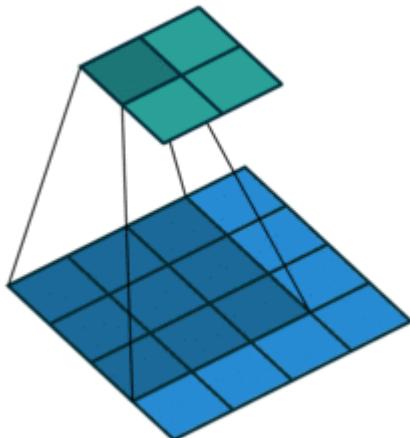
dilation=4



# Размеры в выходных карт в зависимости от параметров свертки

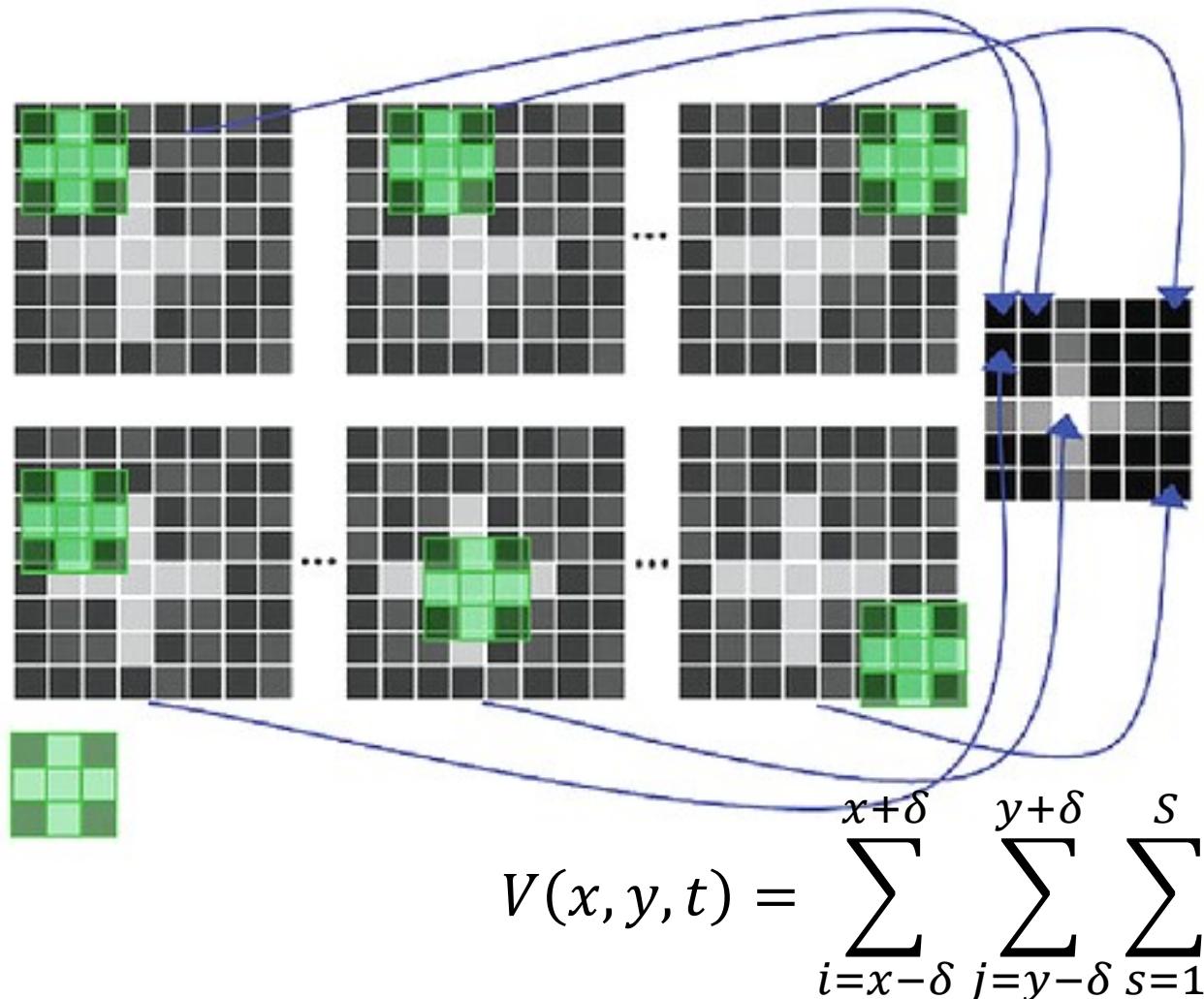
- `torch.nn.functional.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1)`
- Input = [10,10] weight=[3,3]
  
- `stride=1, padding=0, dilation=1?`
- `stride=1, padding=1, dilation=1?`
- `stride=2, padding=0, dilation=1?`
- `stride=1, padding=1, dilation=2?`

# Представление свертки в виде матричного умножения



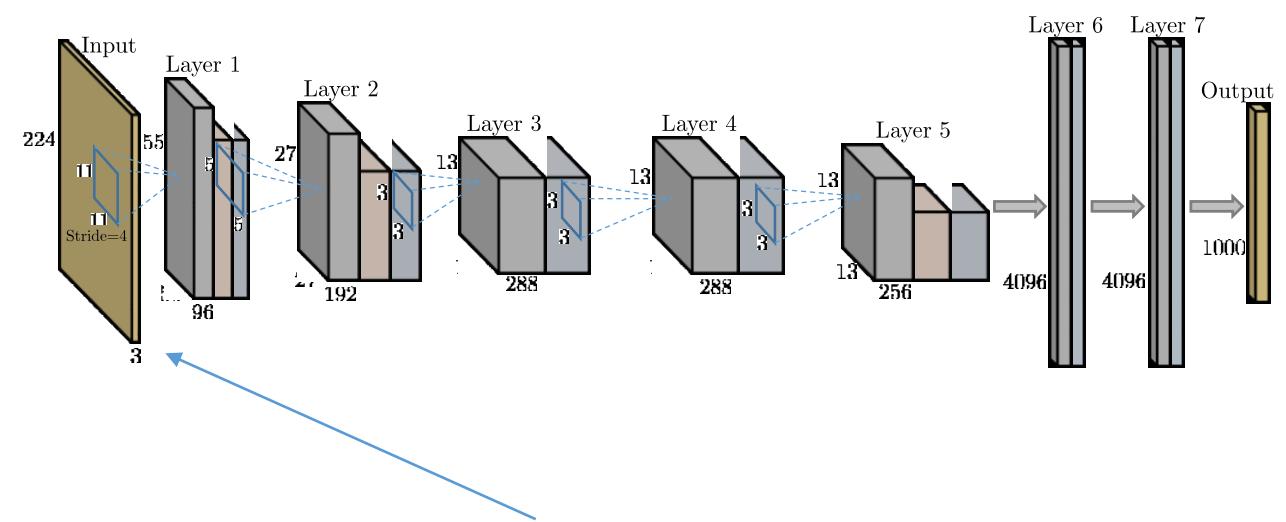
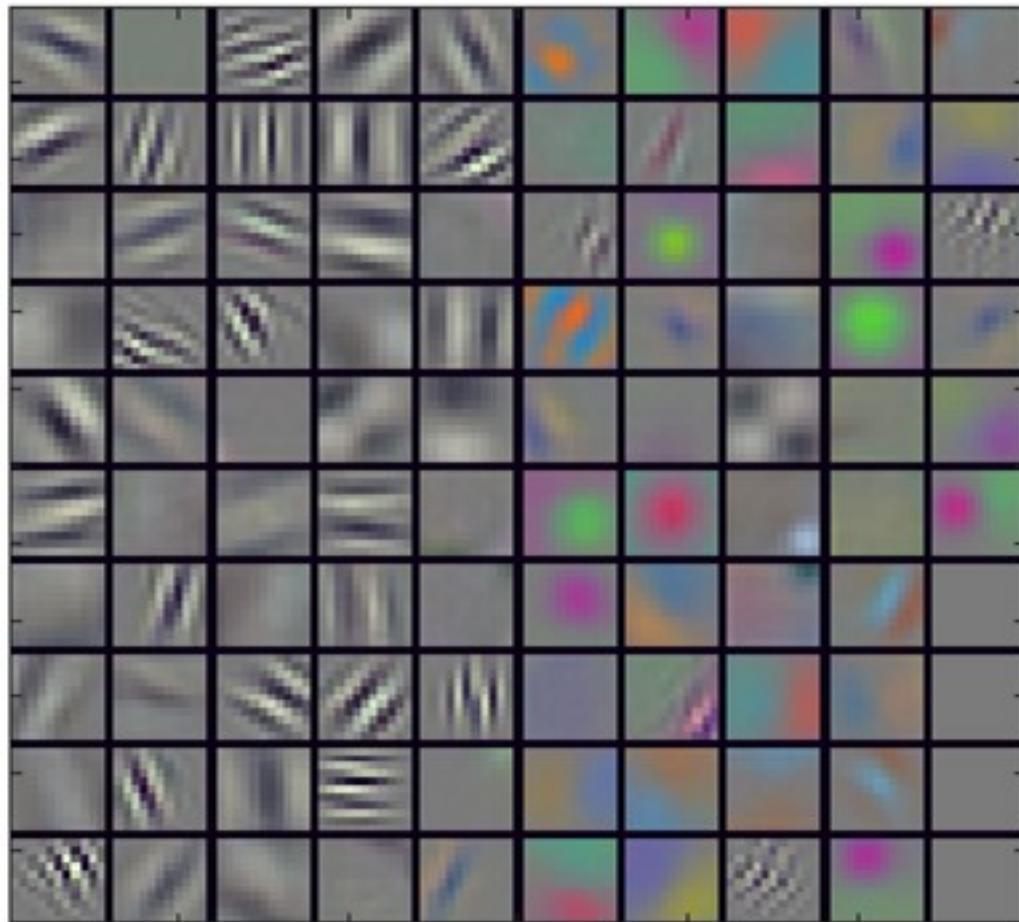
$$\begin{pmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ 0 & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{pmatrix}^T$$

# Что делает свертка в нейронной сети?

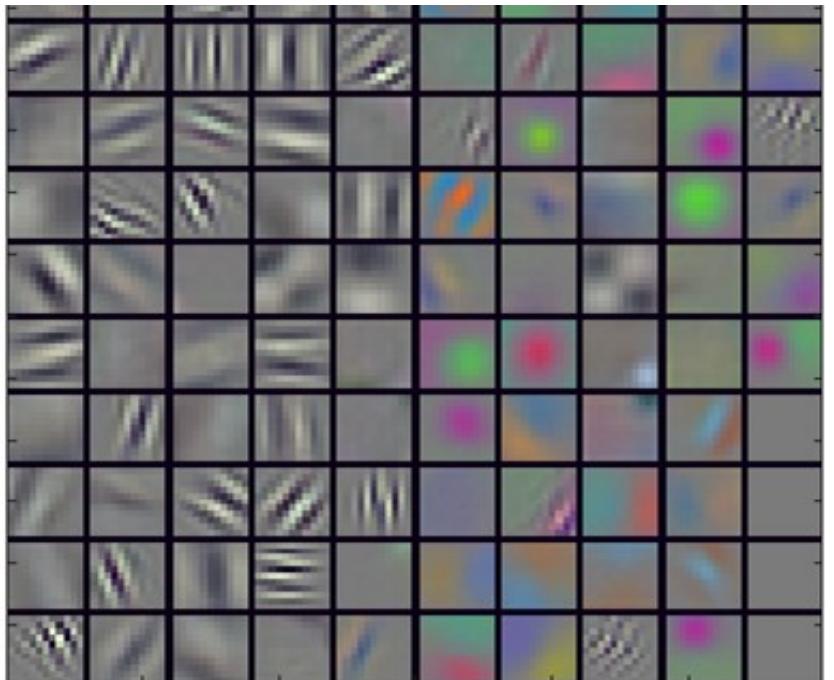


- Это операция корреляции
- Отклик максимальный на похожий объект на изображении

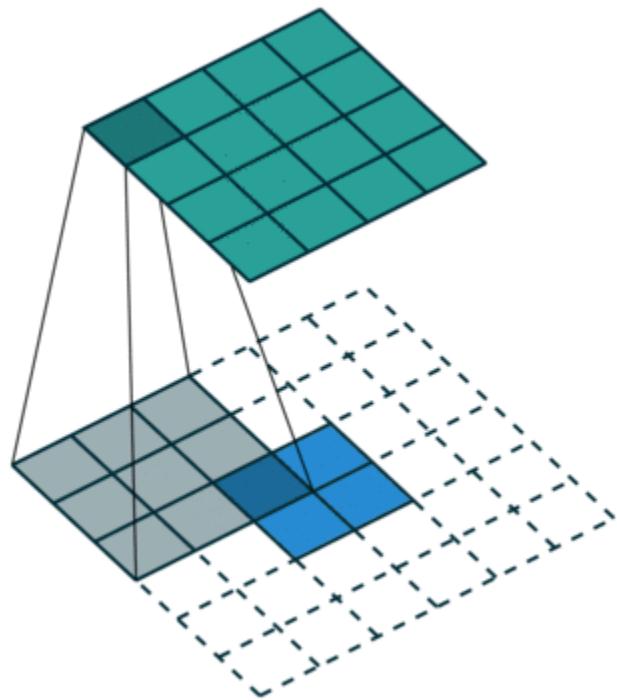
# Поиск шаблонов



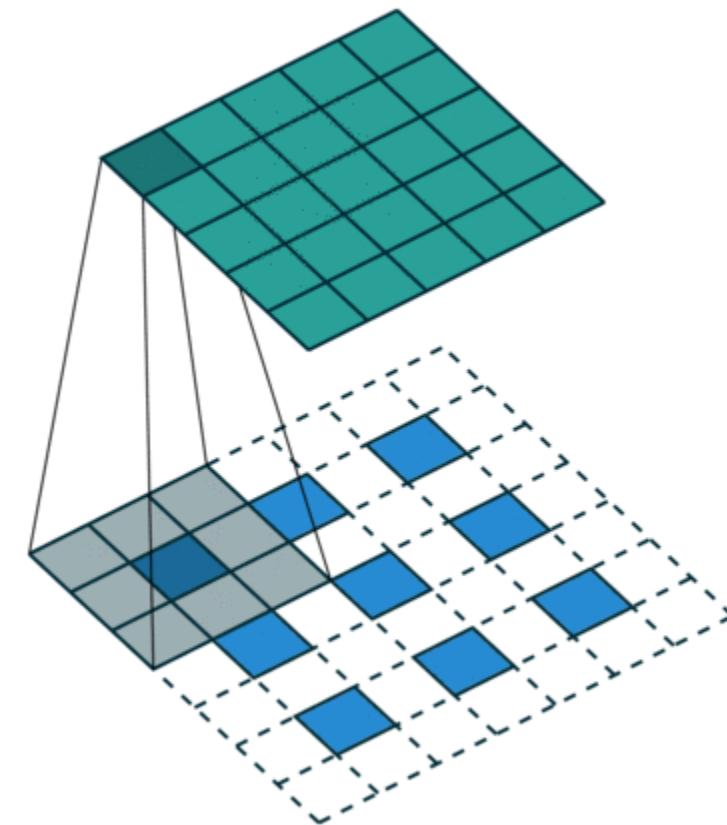
Первый уровень сети AlexNet свертка 11x11



# Обратная свертка (Transposed convolution)



Без сдвига



Со сдвигом 2

# Как работает обращенная свертка с шагом 2

Фильтр

1	1	1
1	1	1
1	1	1

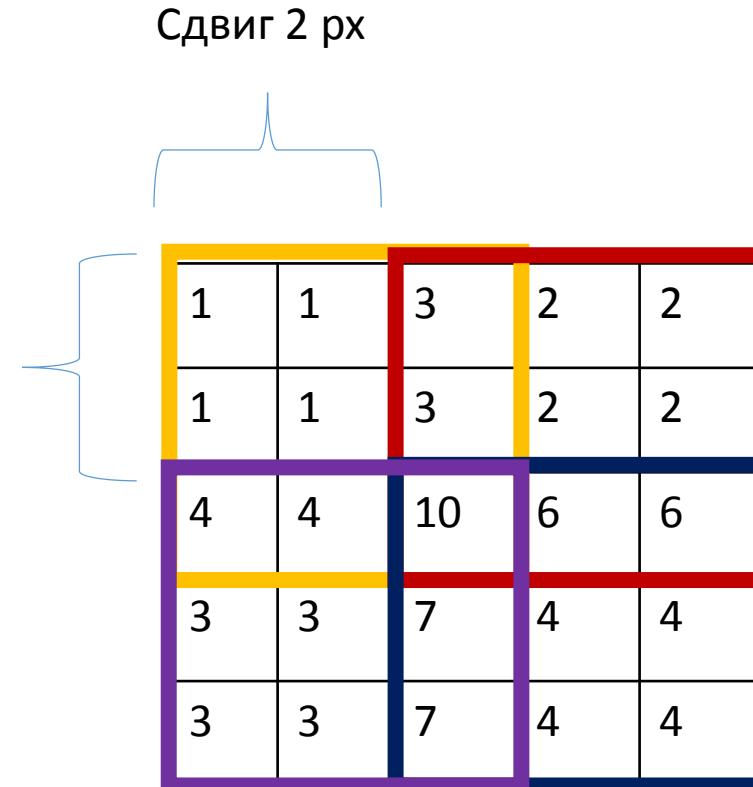
Изображение

1	2
3	4

1	1	1
1	1	1
1	1	1

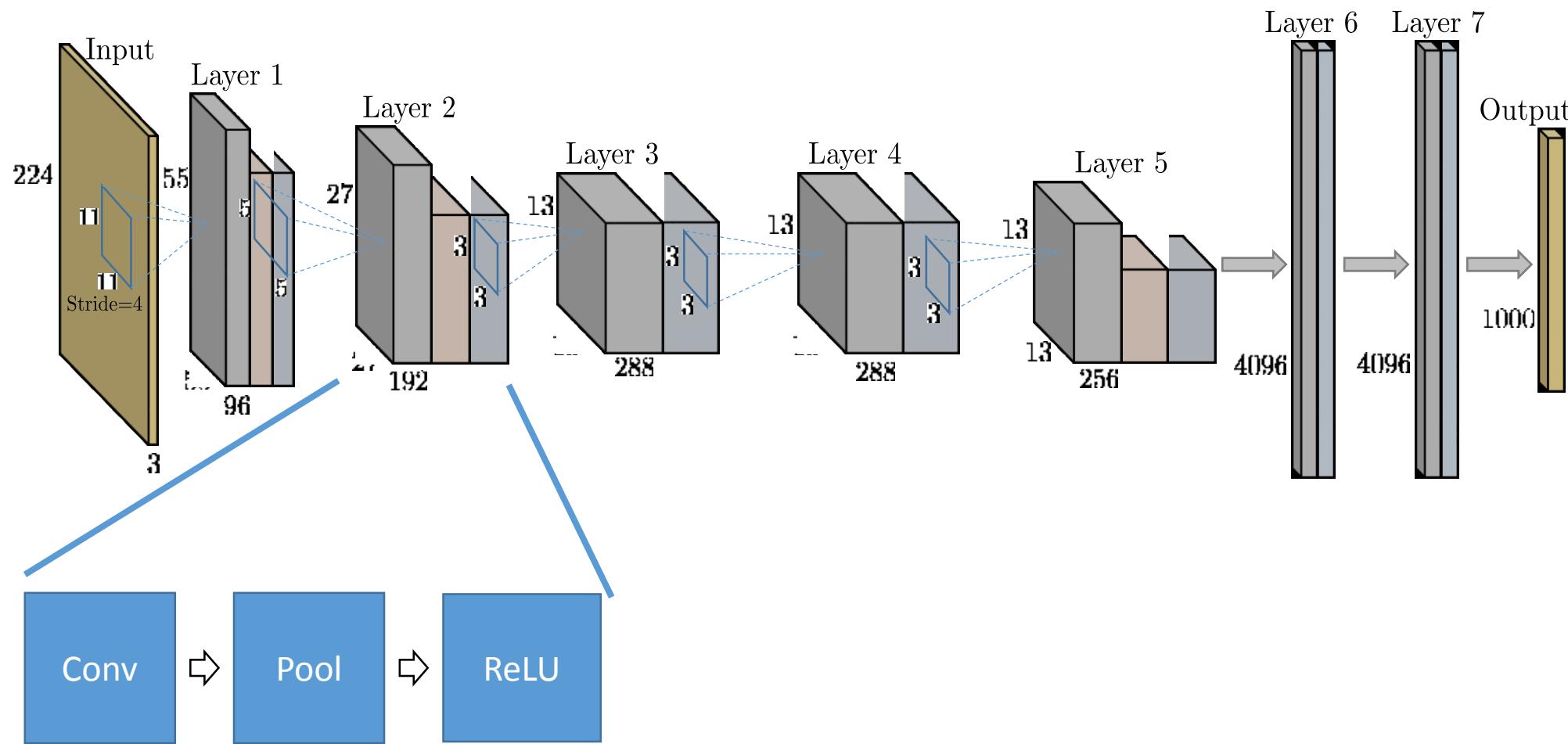
  

3	3	3
3	3	3
3	3	3

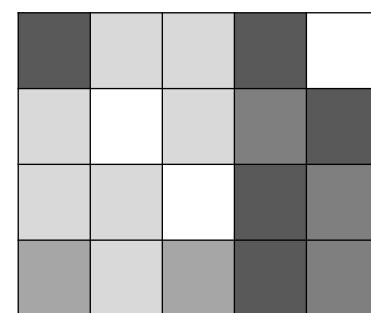
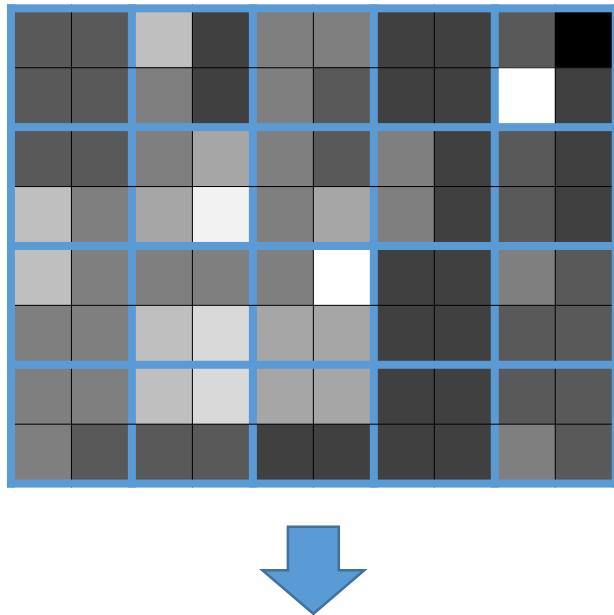


1	1	3	2
1	1	3	2
4	4	10	6
3	3	7	4

# Дизайн глубокой сверточной сети

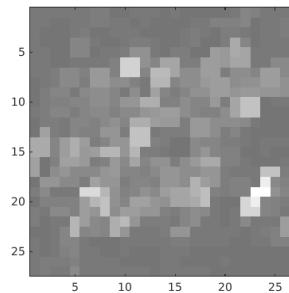
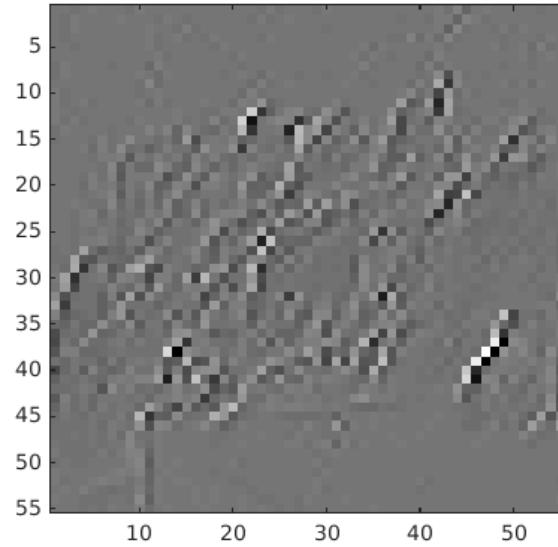


# Pooling (объединение элементов)



- Наиболее часто уменьшения пространственного разрешения – max pooling
- Альтернативы:
  - Усреднение (average pooling)
  - Локальные суммы (sum pooling)
- Быстрое уменьшение пространственного разрешения карт признаков
- Не имеет параметров

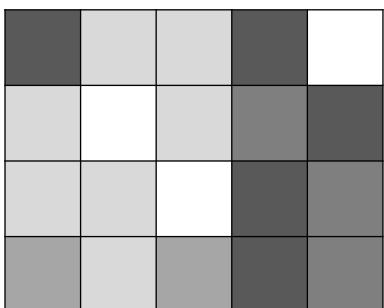
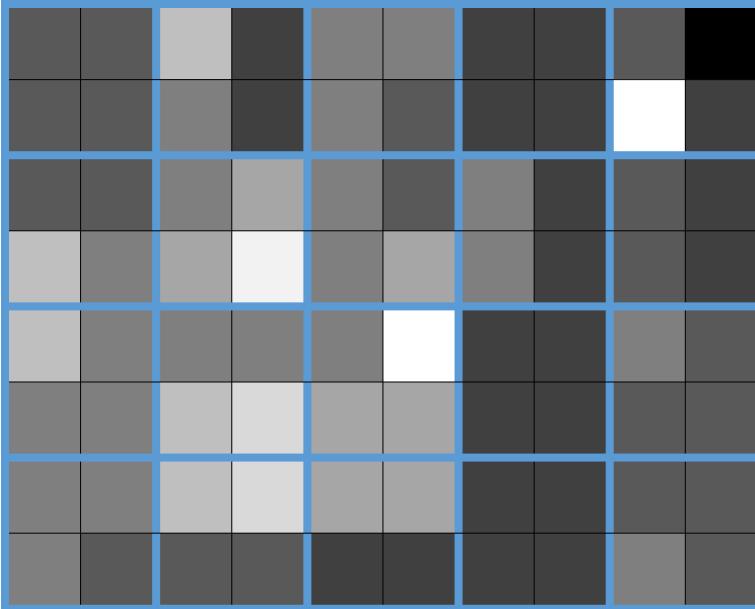
# Max Pooling и устойчивость к сдвигу



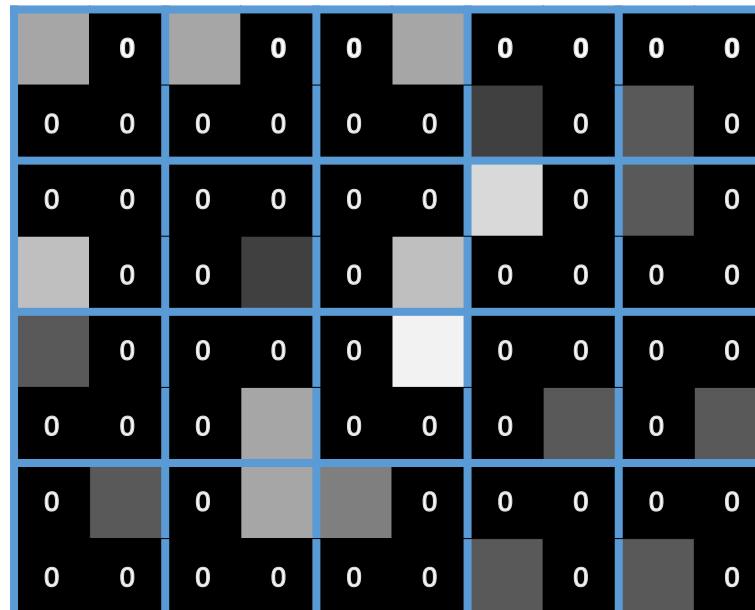
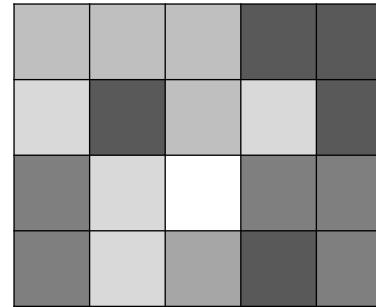
- Часто используется для понижения чувствительности сети к небольшим сдвигам
- Несколько max pooling в сети позволяет получить устойчивость к большим сдвигам

# Обратное распространение ошибки для Max Pooling

Прямой проход

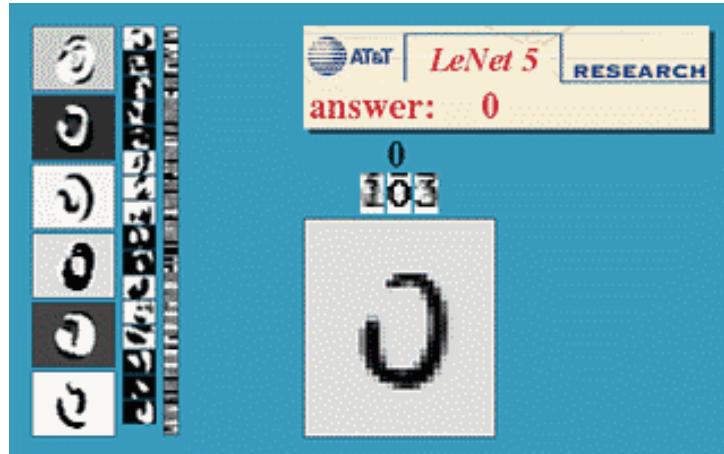


Обратный проход

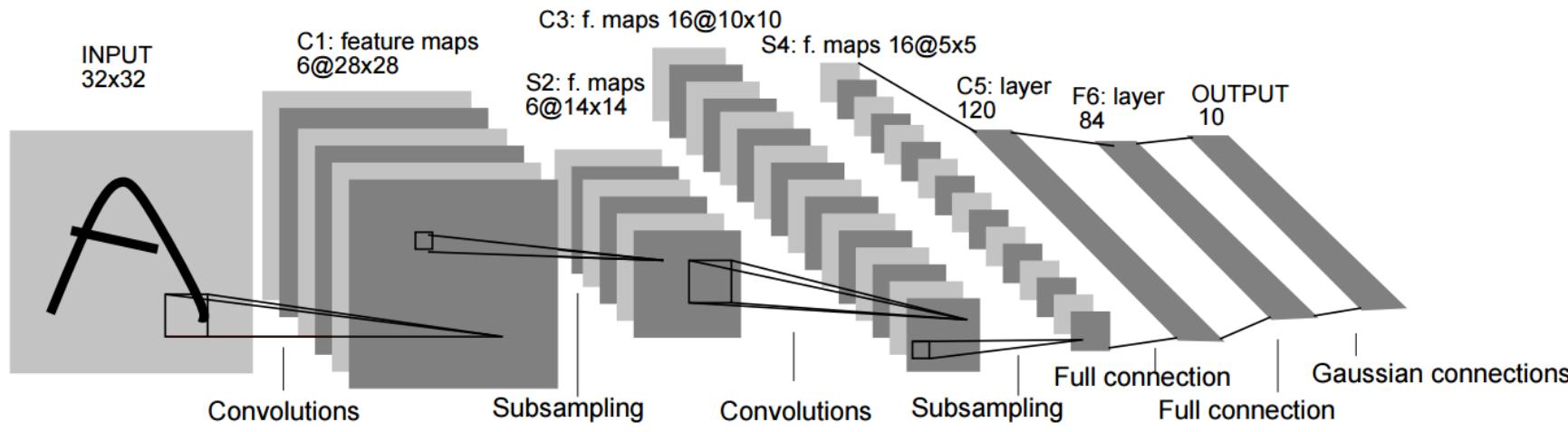


$$\frac{dz}{dx} = \frac{dy^T}{dx} \frac{dz}{dy}$$

# LeNet



3	4	2	1	9	5	6	2	1	8
8	9	1	2	5	0	0	6	6	4
6	7	0	1	6	3	6	3	7	0
3	7	7	9	4	6	6	1	8	2
2	9	3	4	3	9	8	7	2	5
1	5	9	8	3	6	5	7	2	3
9	3	1	9	1	5	8	0	8	4
5	6	2	6	8	5	8	8	9	9
3	7	7	0	9	4	8	5	4	3
7	9	6	4	7	0	6	9	2	3

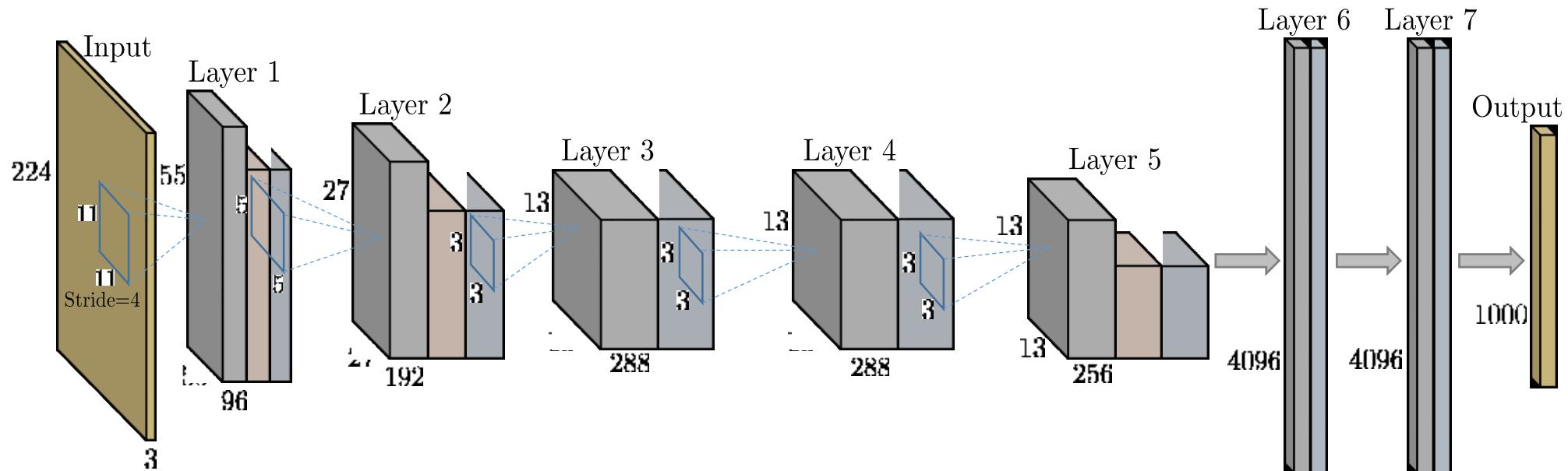


[LeCun 89, 98]

# Практика сверточная сеть

- Взять полносвязанную сеть для классификации изображений чисел
- Заменить узлы на свертки со вставкой ReLU+MaxPooling
- Добавить Dropout

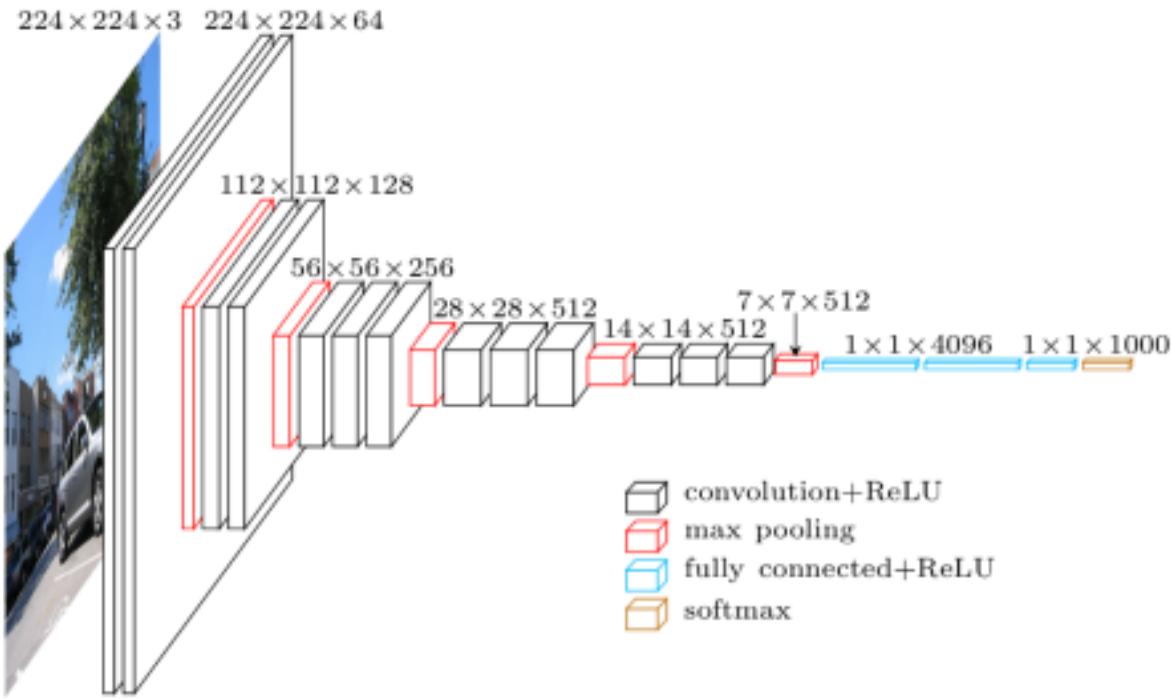
# AlexNet



- 5 сверточных слоев (11x11, 5x5, 3x3, 3x3, 3x3)
- 60М параметров
- Обучается 3-5 дня на GPU

[Krizhevsky et al. 2012]

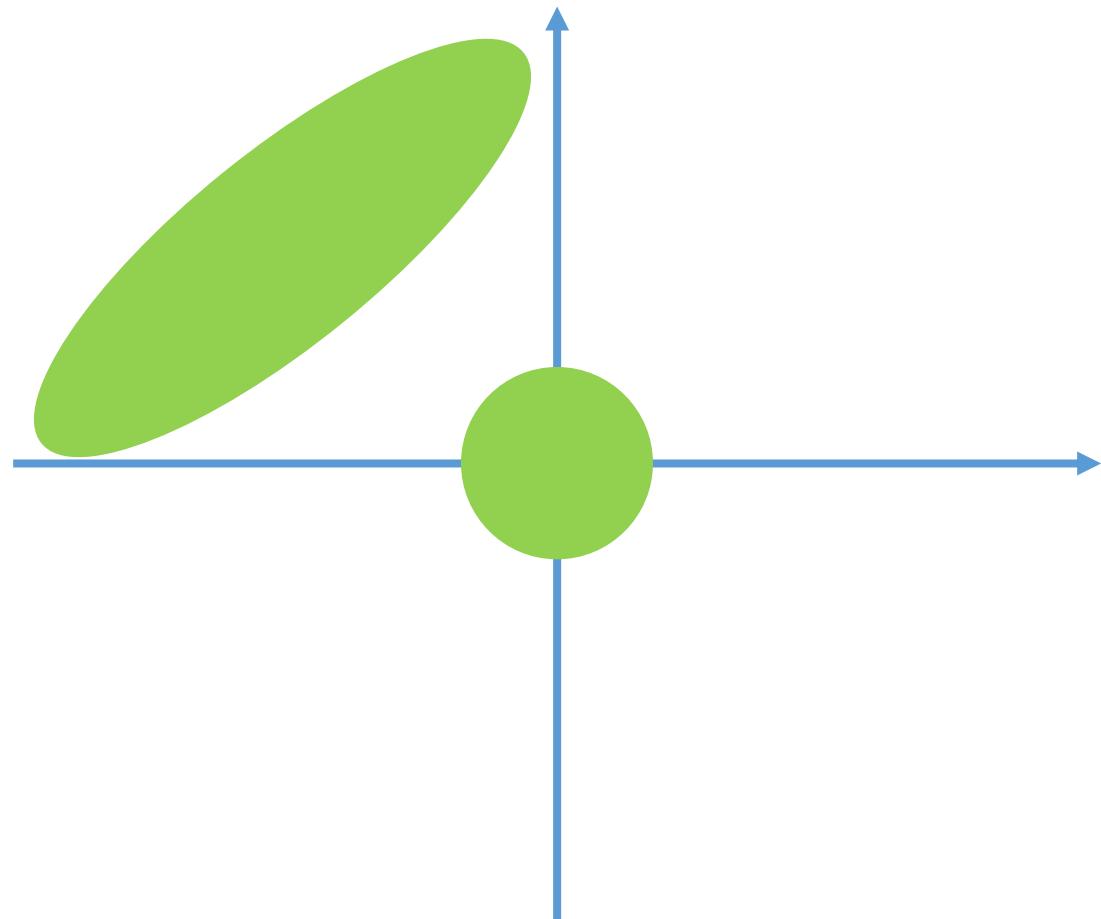
# VGG16



- 16 сверточных слоев
- Все фильтры 3x3
- Распределенная нагрузка между слоями
- ~140M параметров
- Наилучшее быстродействие в классе прямых сетей

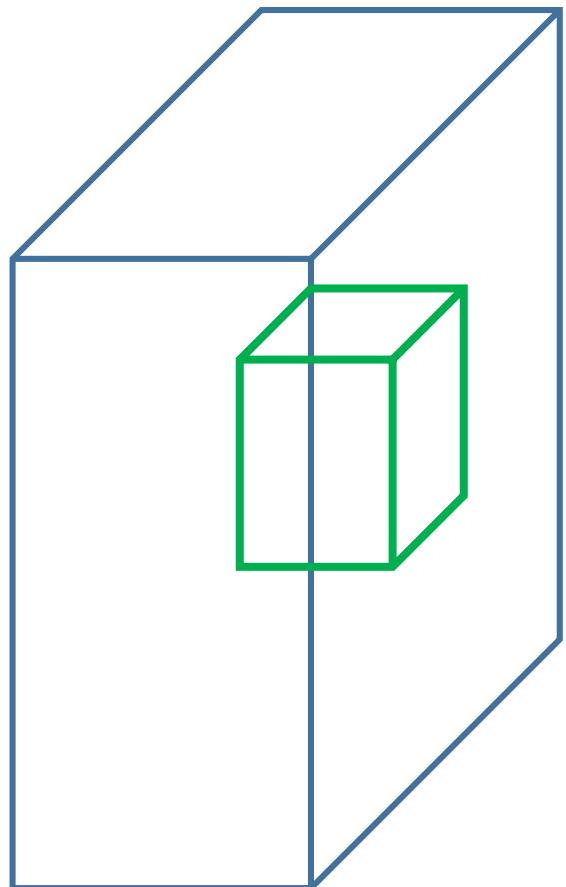
[Simonyan & Zisserman, 2014]

# Batchnorm (Нормализация батча)



- Есть батч  $B = \{x^{(i)}\}, i = 1..m$
- Параметры  $\gamma, \beta$
- $\mu_B = \frac{1}{m} \sum_i x^{(i)}$
- $\sigma_B^2 = \frac{1}{m} \sum_i (x^{(i)} - \mu_B)^2$
- $\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
- $BN_{\gamma, \beta}(x^{(i)}) = \gamma \hat{x}^{(i)} + \beta$

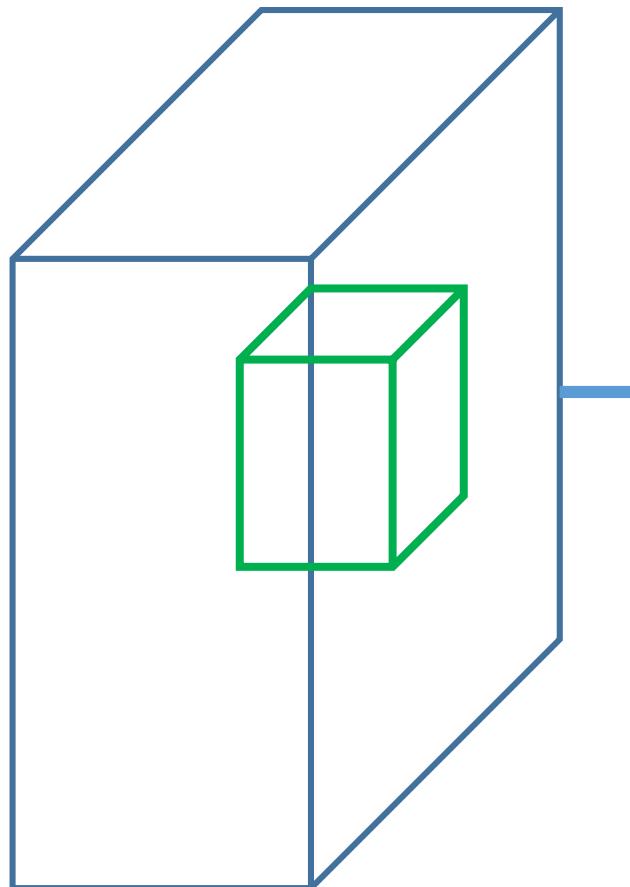
# Свертка 1x1



Свертка – линейный классификатор для выбранного патча



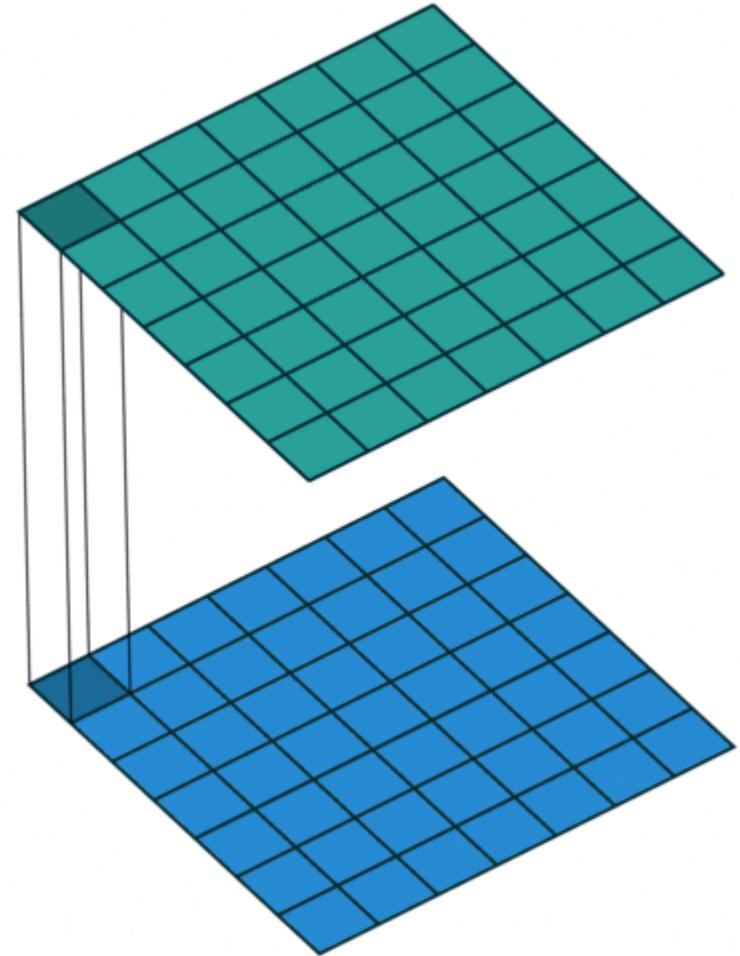
# Свертка 1x1



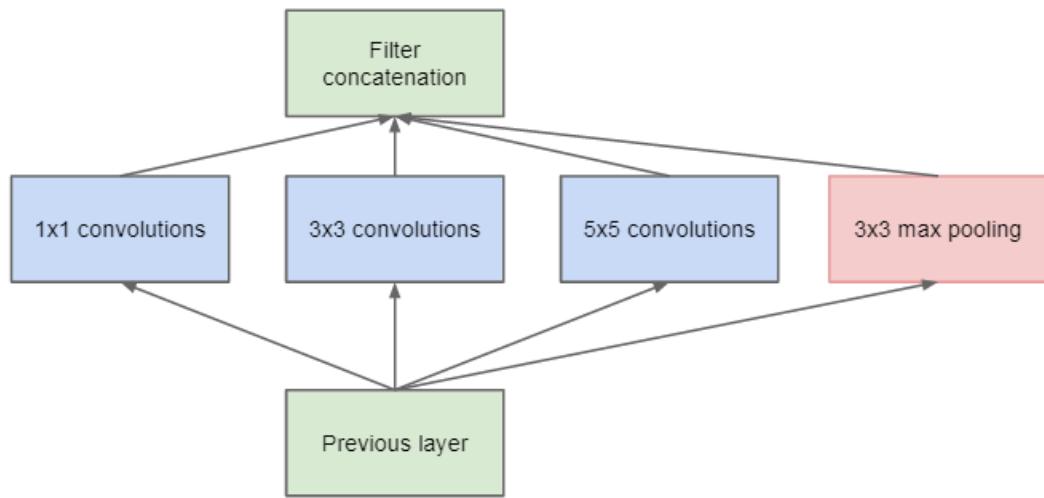
Свертка KxK

Свертка 1x1

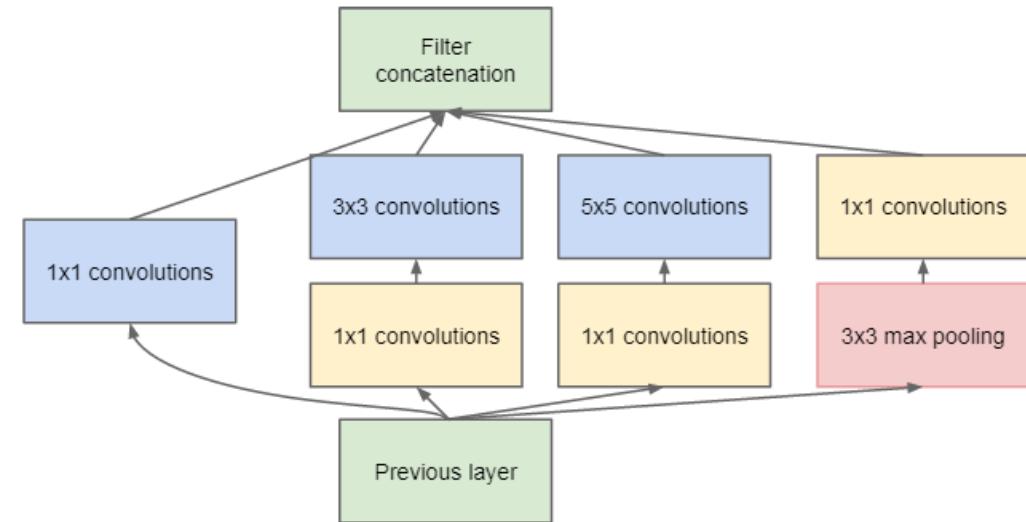
Мини нейронная сеть с очень  
дешевой операцией умножения



# Inception (Вложение)

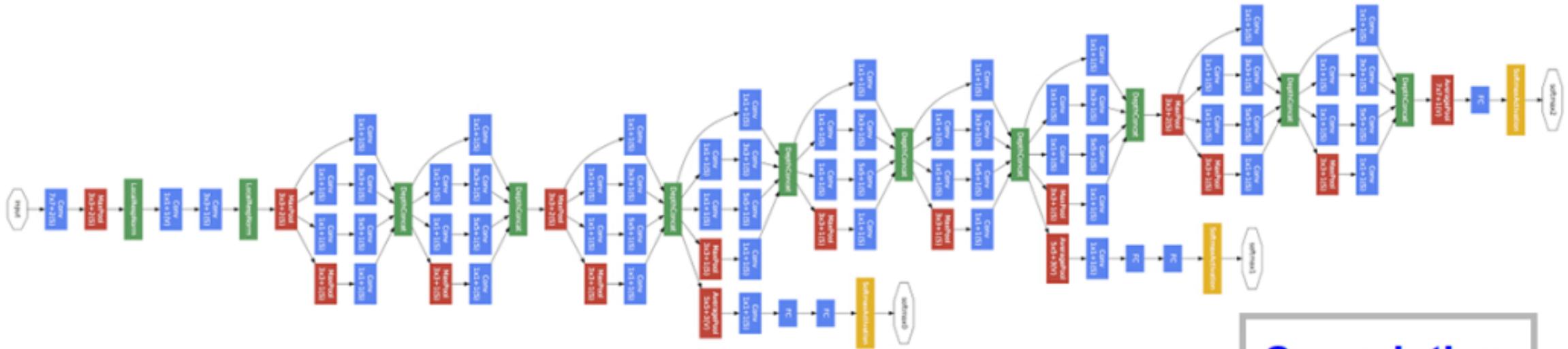


Исходный модуль вложения



1x1 свертка используется для уменьшения  
пространства признаков и экономии памяти

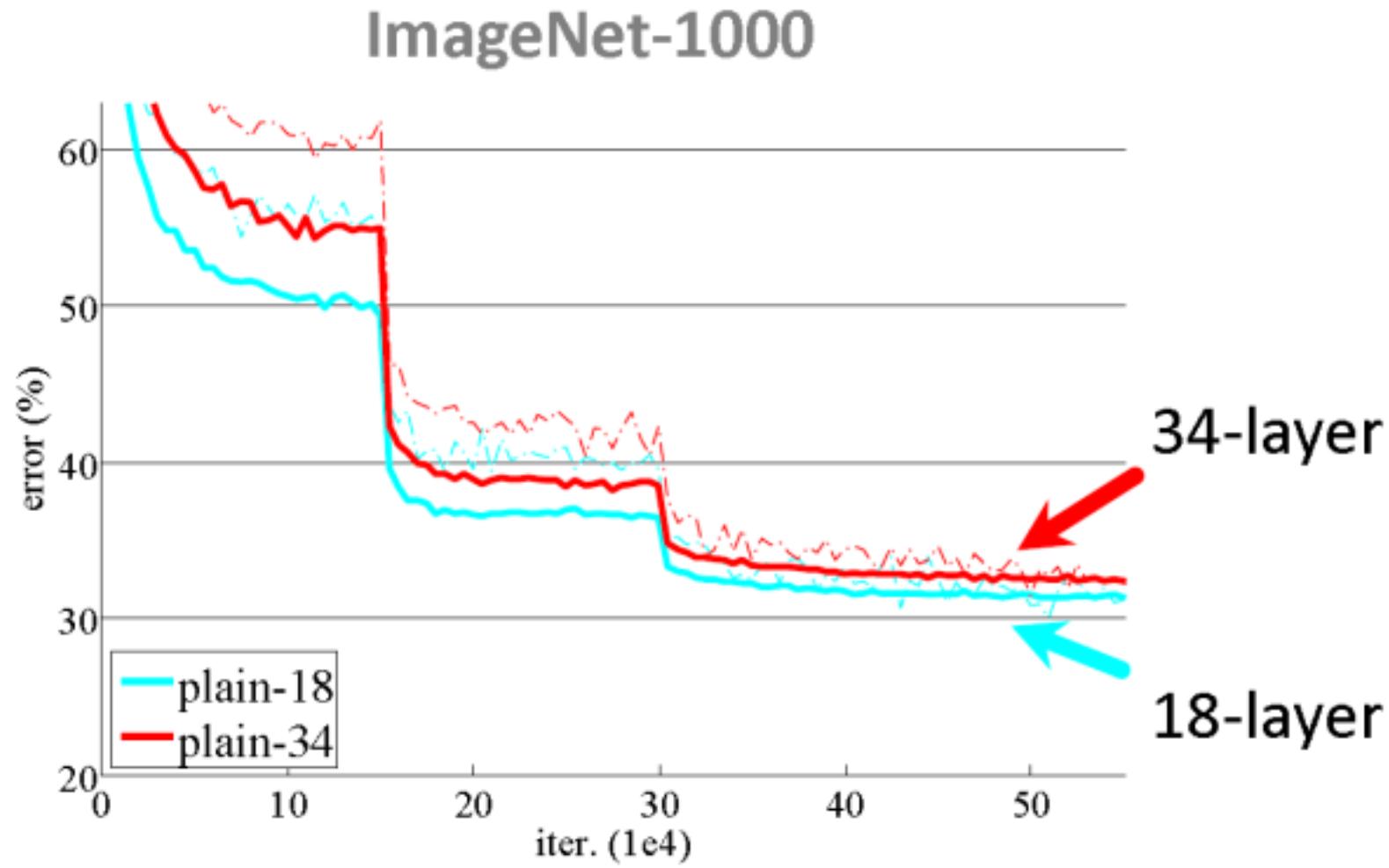
# GoogleNet



Convolution  
Pooling  
Softmax  
Other

[Szegedy et al. 2014]

# Предел глубины прямых сетей



# ResNet

ResNet, **152 layers**  
(ILSVRC 2015)



AlexNet, 8 layers  
(ILSVRC 2012)



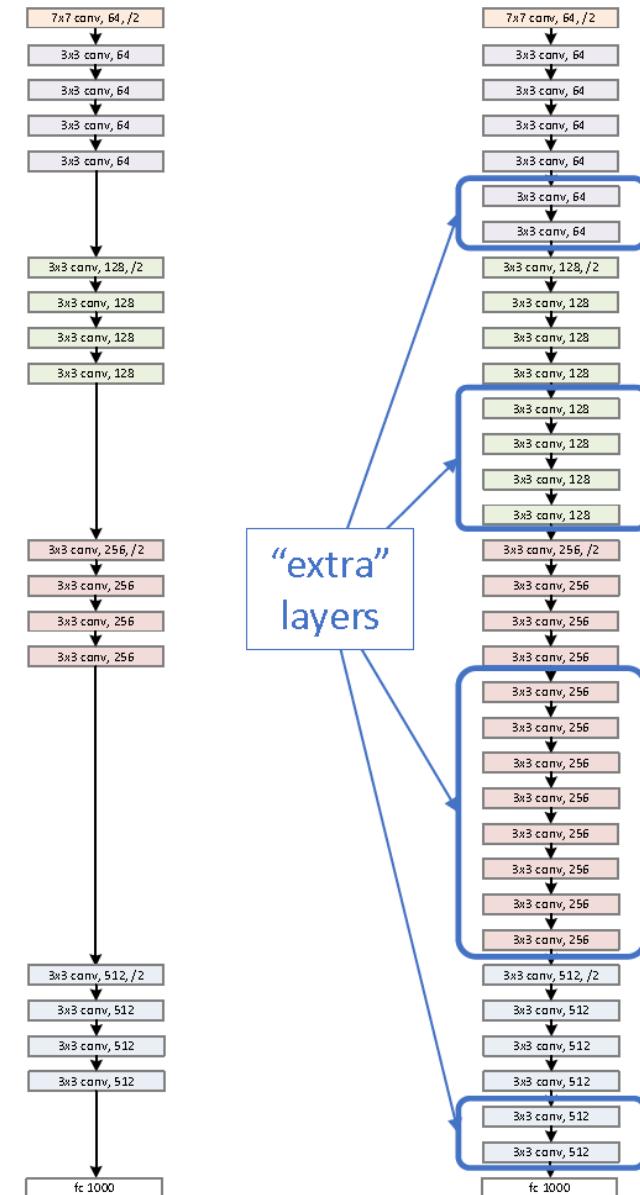
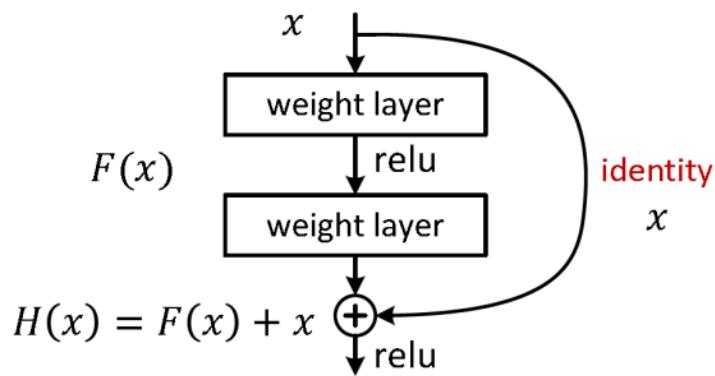
VGG, 19 layers  
(ILSVRC 2014)



[He et al. 2015]

# ResNet

- Обучаем сначала упрощенную версию
- Потом вставляем модули коррекции, которые делают «добавку» исходной сети



[He et al. 2015]

# ImageNet

## Soccer, association football

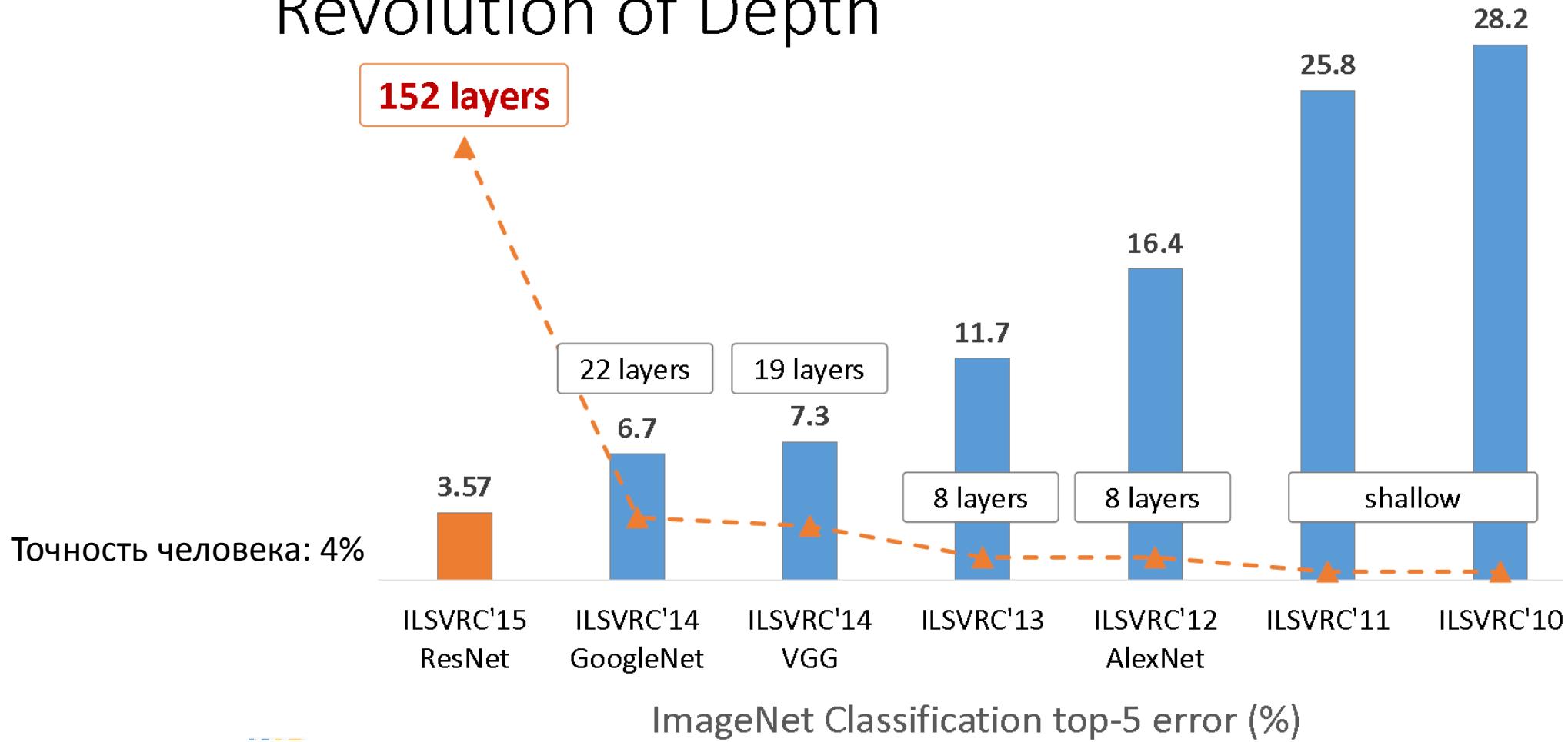
A football game in which two teams of 11 players try to kick or head a ball into the opponents' goal

The screenshot shows the ImageNet search interface for the term "Soccer, association football". The sidebar on the left displays a hierarchical tree of categories, with "soccer, association foot" selected. The main area shows a grid of 40 thumbnail images of soccer games. At the top right, there are statistics: 1395 pictures, 90.76% Popularity Percentile, and Wordnet IDs. Below the grid, a note states: "Images of children synsets are not included. All images shown are thumbnails. Images may be subject to copyright." Navigation links "Prev" and "Next" are at the bottom.

- 14,197,122 изображений, 21841 классов
- Соревнование проходит на 1000 классах
- Задача: чтобы истинный класс был в 5 наиболее вероятных предсказаниях

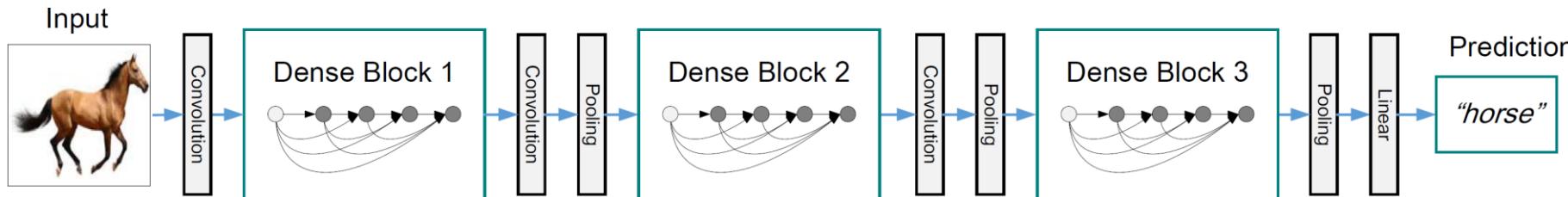
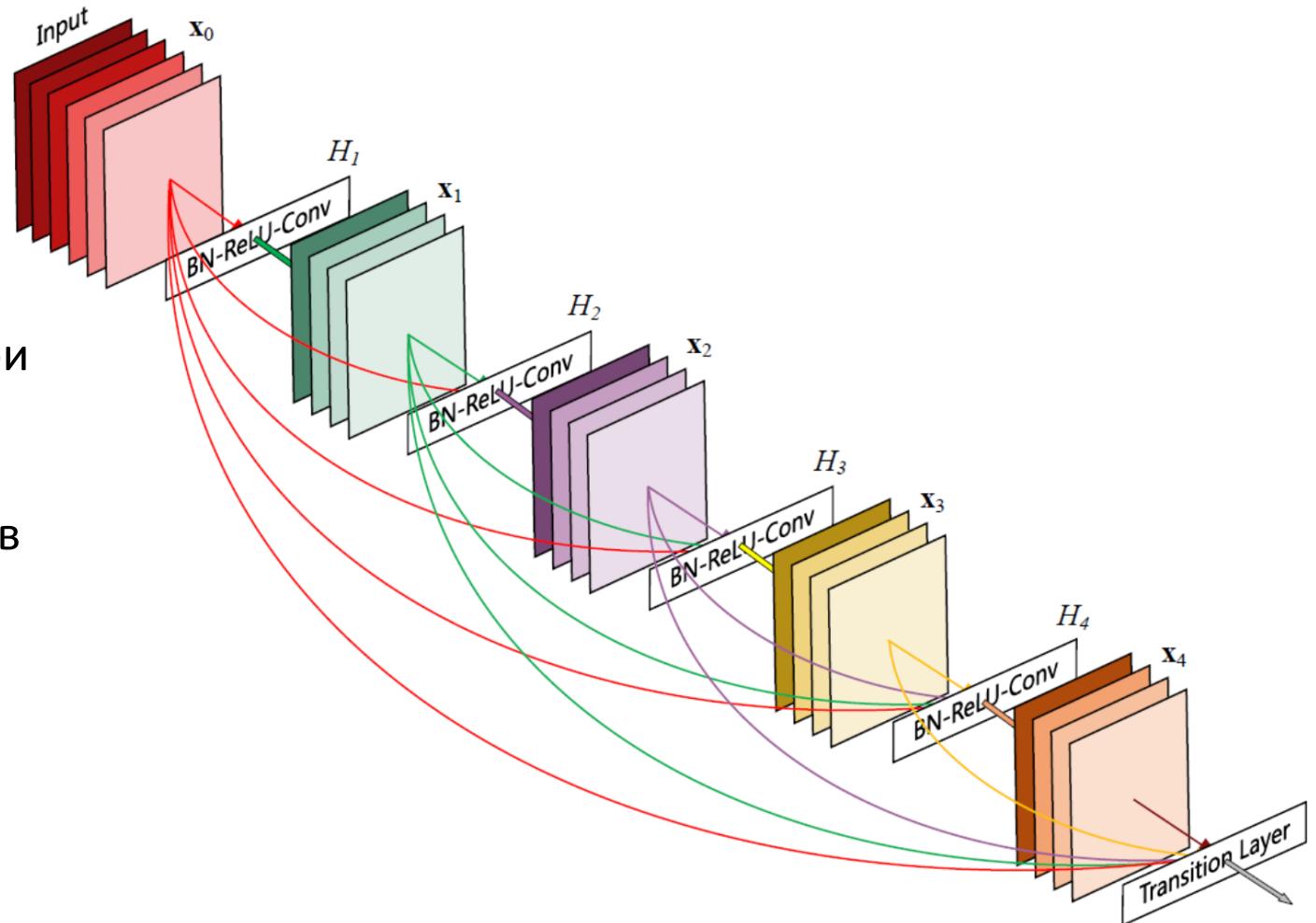
# Углубление нейронной сети

## Revolution of Depth



# DenseNet

- Точность аналогична ResNet при меньшем числе параметров/операций
- Позволяет сети выбирать пути в вычислительном графе

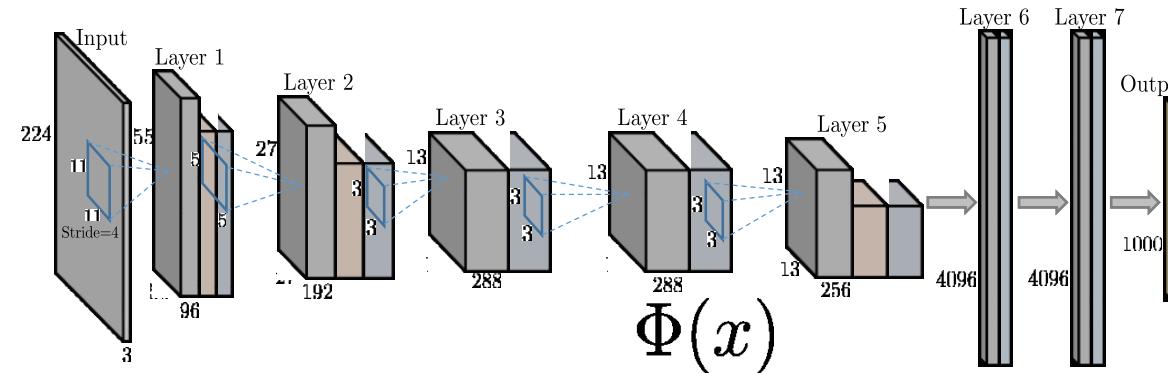


[Huang et al CVPR17]

# ИТОГО

- Сверточные сети – наиболее популярный и влиятельный инструмент в современном глубинном обучении
- Свертки позволяют сильно сократить число параметров модели за счет повторного использования параметров
- Прямые нейронные сети уменьшают пространственное разрешение, параллельно увеличивая глубину. Глубина позволяет кодировать семантическую информацию.

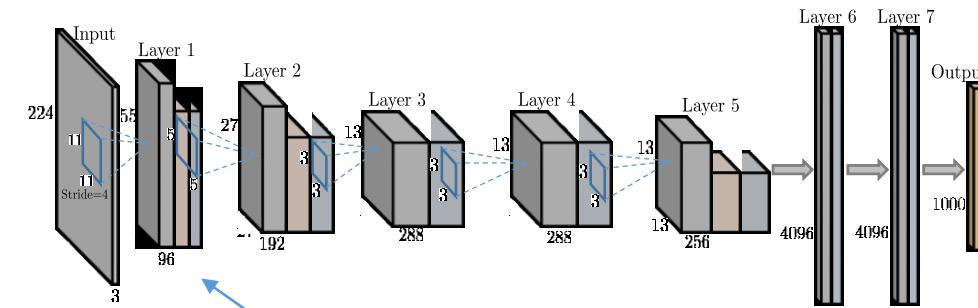
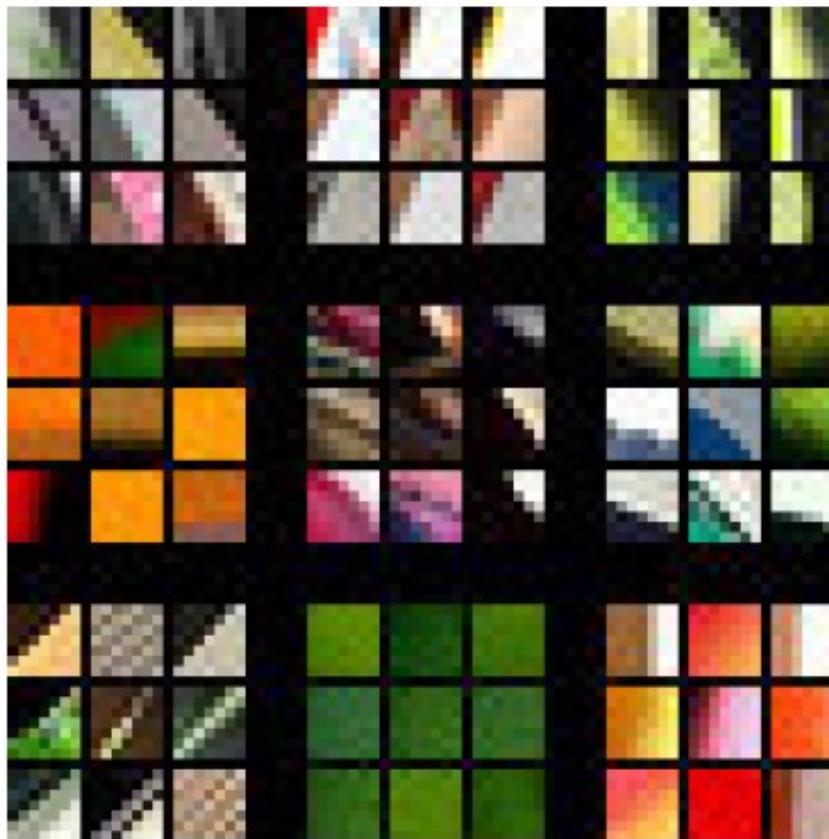
# Представления внутри нейронной сети



Вопросы:

- Каковы их свойства?
- Являются ли они избыточными?
- Являются ли они обратимыми?
- Как мы их можем использовать?

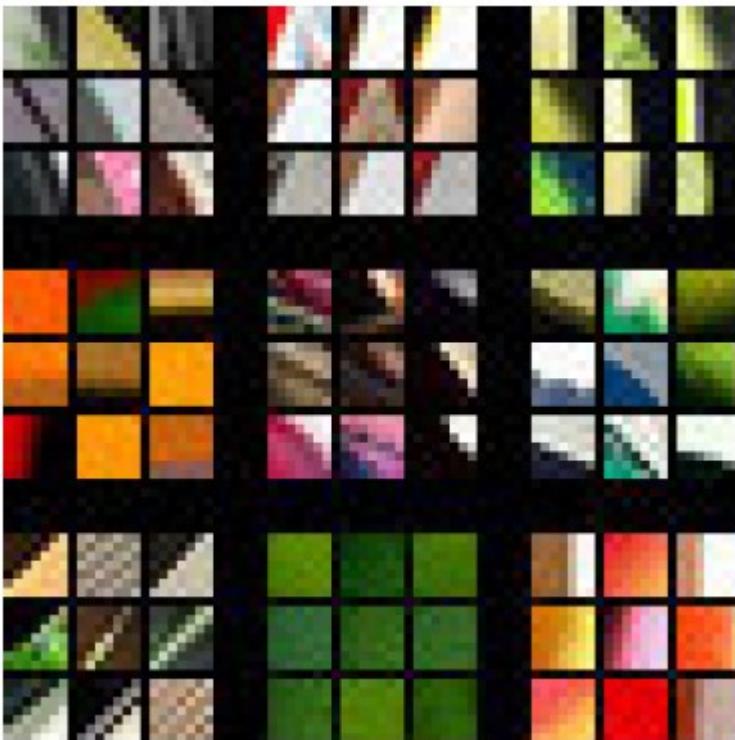
# Чувствительность к шаблонам



$$\Phi_1(x)$$

$$\operatorname{argmax}_{x \in S} \Phi_1^t(x) - ?$$

# Чувствительность к шаблонам



$\Phi_1(x)$



$\Phi_2(x)$

[Zeiler Fergus 14]

# Чувствительность к шаблонам



$$\Phi_3(x)$$

[Zeiler Fergus 14]

# Пространство признаков

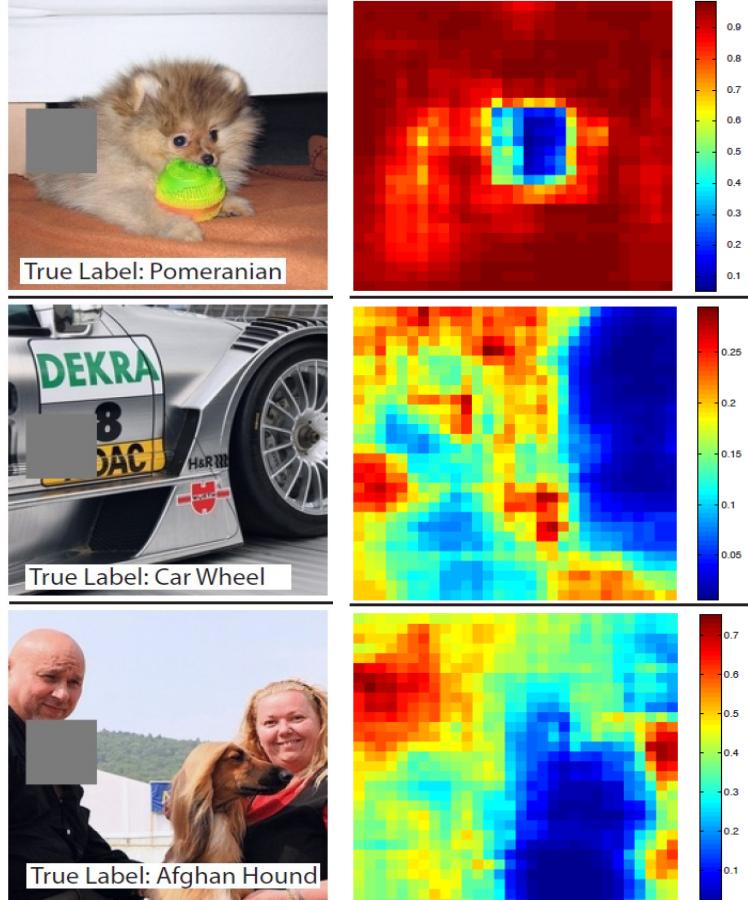
- Максимальные активации случайного нейрона
  - $\text{argmax}_{x \in S} \Phi_5^t(x)$
- $\Phi_k(x)$  - образует N мерное
  - Линейная комбинация векторов  
$$\text{argmax}_{x \in S} \sum_{t \in T} w_t \Phi_k^t(x)$$



# Практика представление в сверточных сетях

- Загрузить архитектуру AlexNet с весами из torchvision
- Визуализировать фильтры на первом уровне
- Визуализировать карты
- Визуализировать градиенты для заданного класса

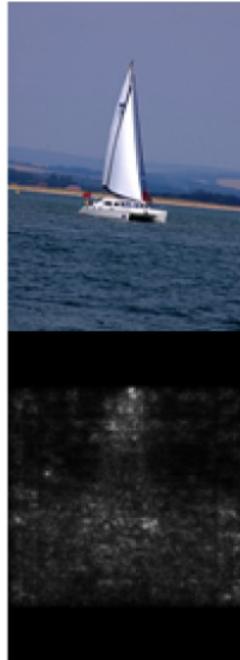
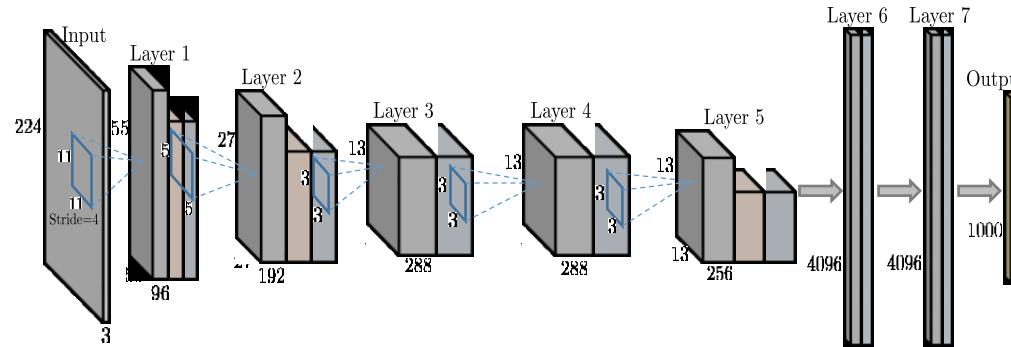
# Какие части изображения влияют больше всего на решение классификатора?



1. Закрываем различные области изображения – смотрим на значение активации советующего класса
2. Пикслю выходной карты присваиваем значение активации соответствующего нейрона
3. Хорошая визуализация, но медленная
4. Требуется выбор размера окна

# Использовать градиенты по входному изображению

$$\left\| \frac{\partial \Phi_{\text{Last}}[y_0]}{\partial x} \right\|$$



[Simonyan et al. 2013]

# Как обмануть классификатор?

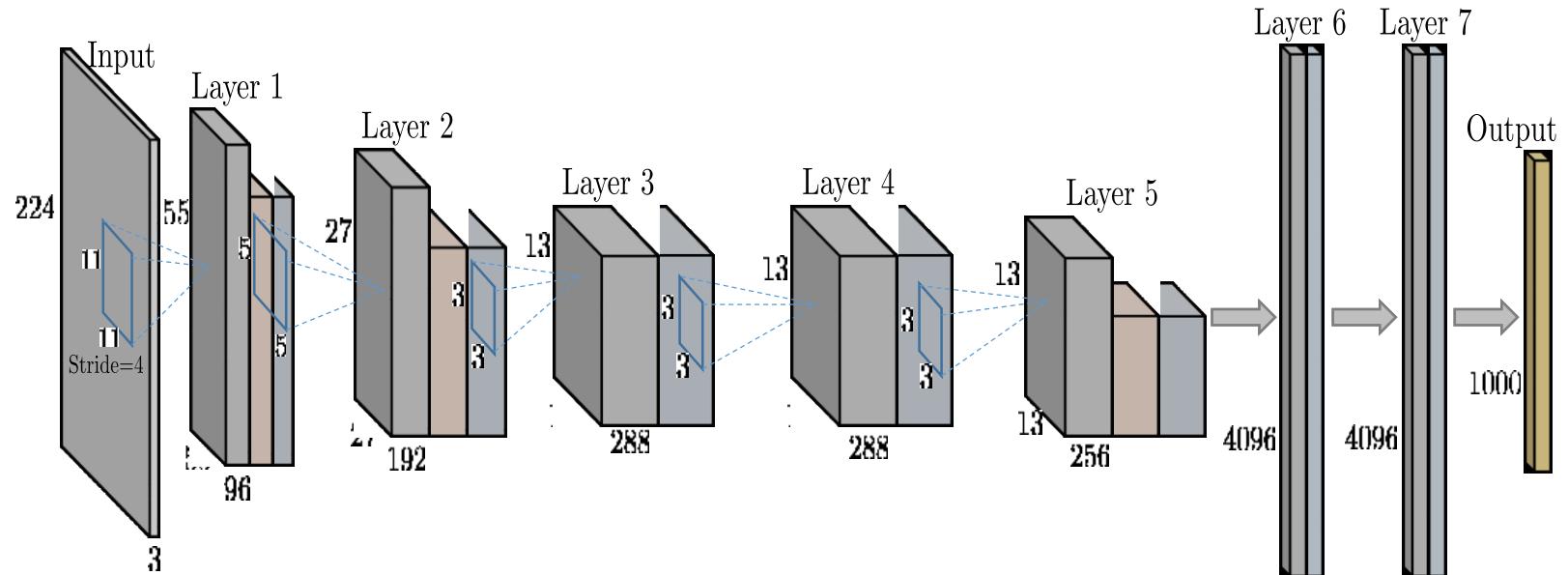


$$\hat{x} = \arg \max_x (\Phi_{\text{Last}}(x)[y_0] - \lambda R(x))$$

[Simonyan et al. 2013]

# Генерация шума, обманывающего классификатор

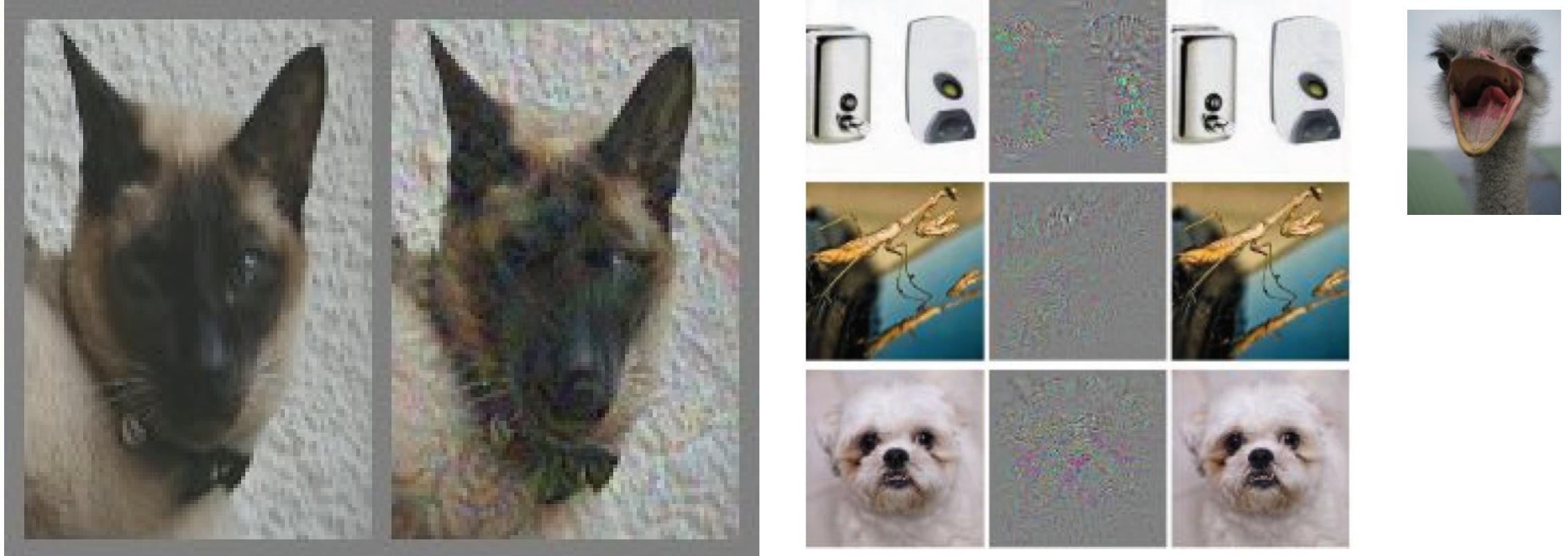
$$\Phi_{Last}(x) = y$$



$$\operatorname{argmax}_r \Phi_{Last}(x + r)[y'] - \lambda ||r||$$

[Szegedy et al 2014]

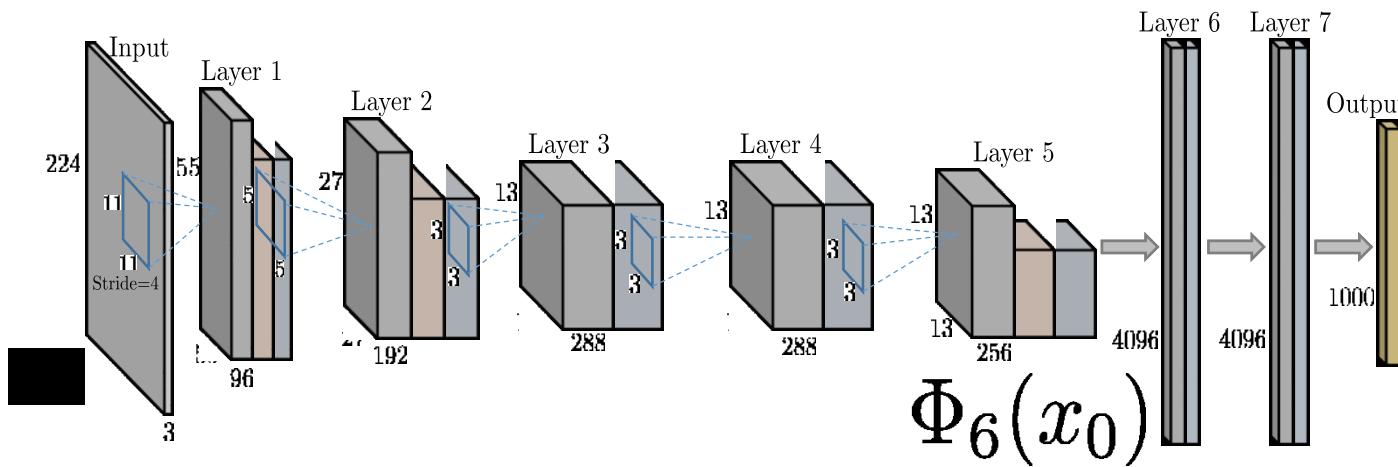
# Генерация шума, обманывающего классификатор



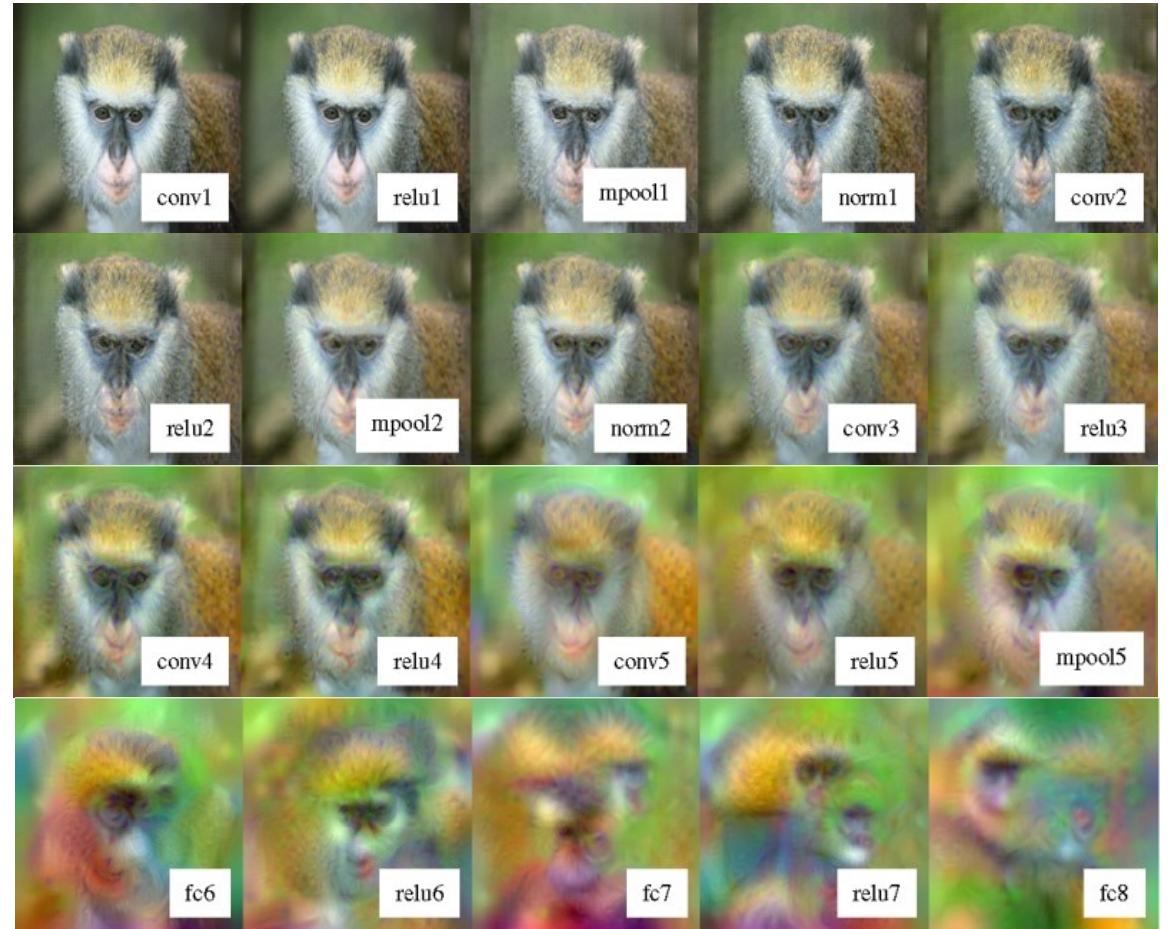
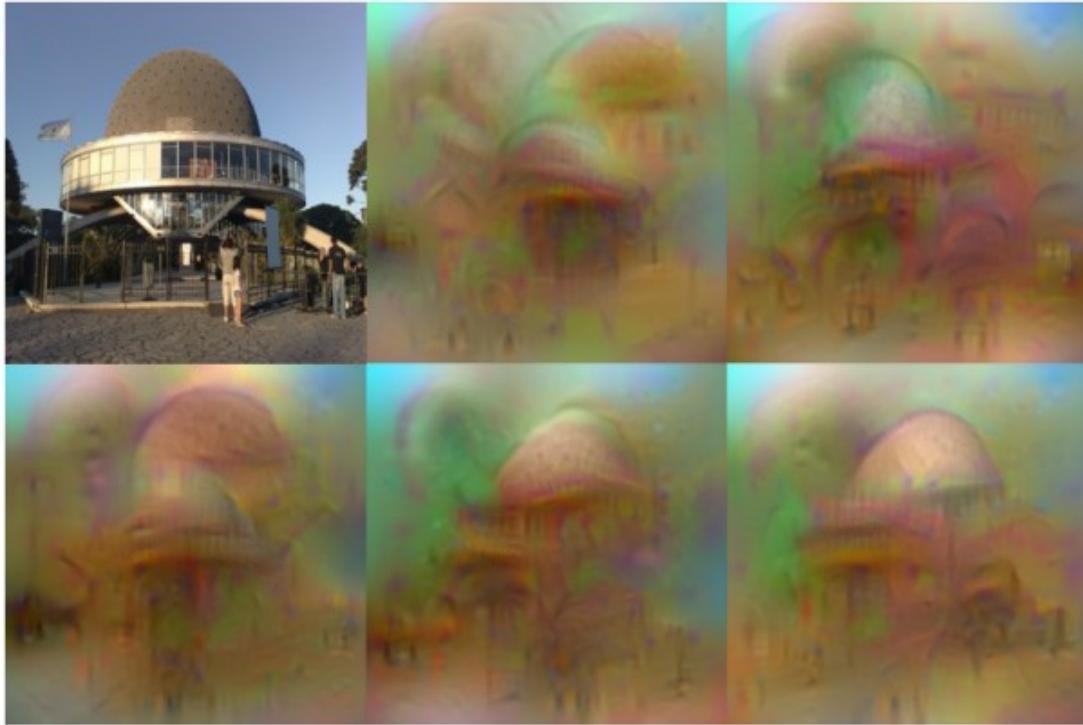
[Szegedy et al 2014, Elsayed et al. ArXiV18]

# Можно ли получить изображение из его глубокого представления?

- $\hat{x} = \operatorname{argmin}_x \|\Phi_k(x) - \Phi_k(x_0)\|^2 + \lambda R(x)$
- $R(x) = \sum_{p,q} \sqrt{(x_{p,q} - x_{p-1,q})^2 + (x_{p,q} - x_{p,q-1})^2}$



# Получение изображения путем обращения сверточной сети

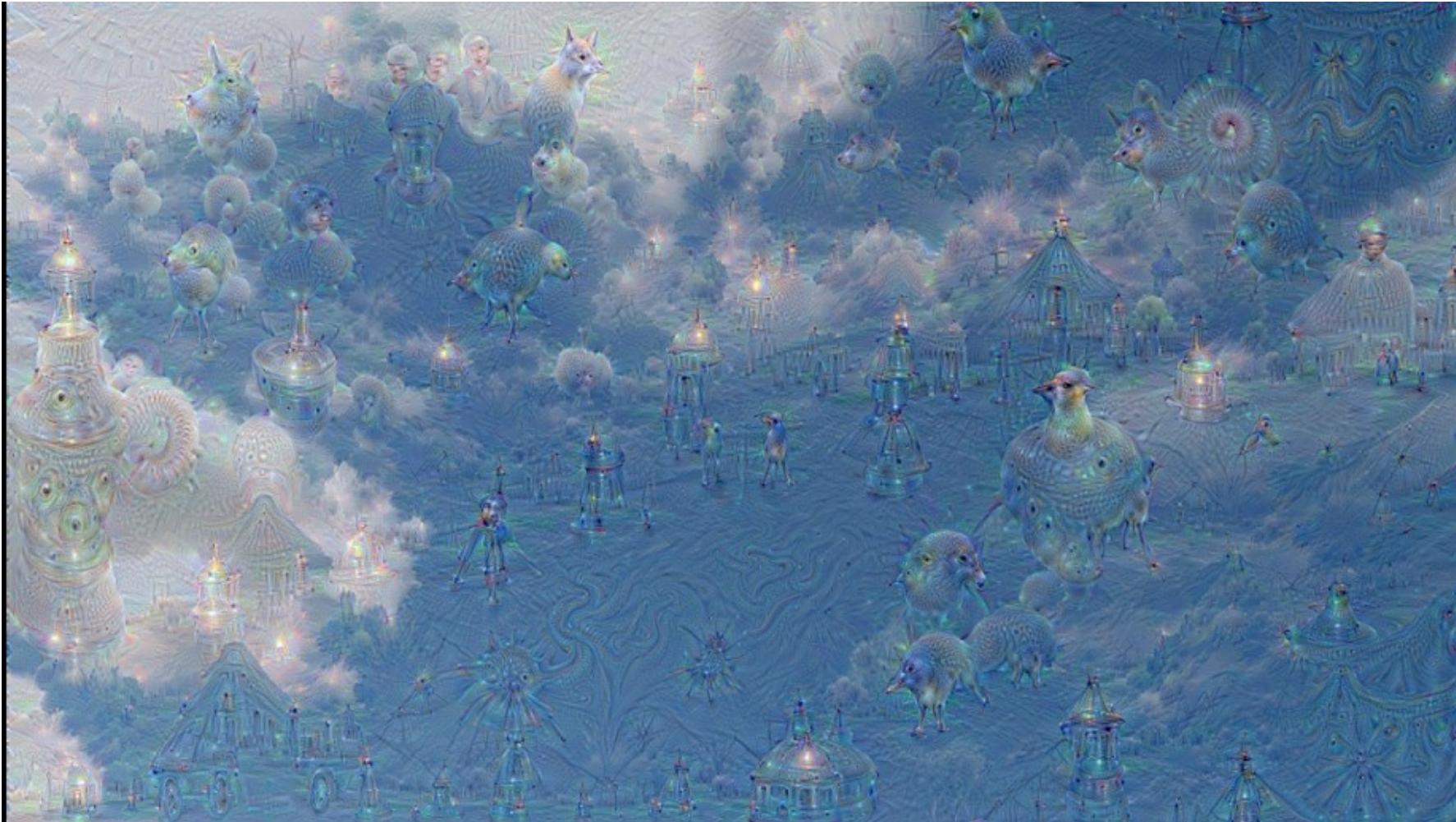


$$\hat{x} = \operatorname{argmin}_x \left\| \Phi_k(x) - \Phi_k(x_0) \right\|^2 + \lambda R(x)$$

[Mahendran & Vedaldi CVPR15]

Google DeepDream

$$\operatorname{argmax}_x \|\Phi_k(x)\| - \lambda R(x)$$



Google DeepDream

$$\operatorname{argmax}_x \left| \left| \Phi_k(x) \right| \right| - \lambda R(x)$$



# Стилизация изображений

A



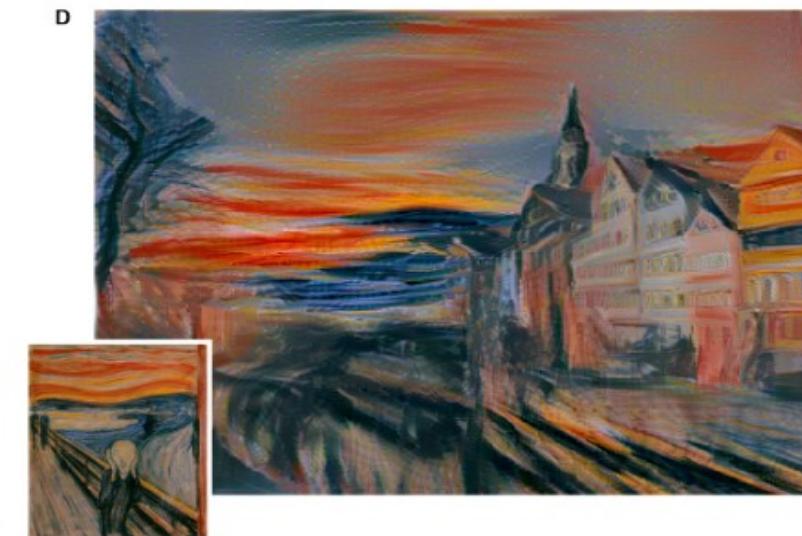
B



C



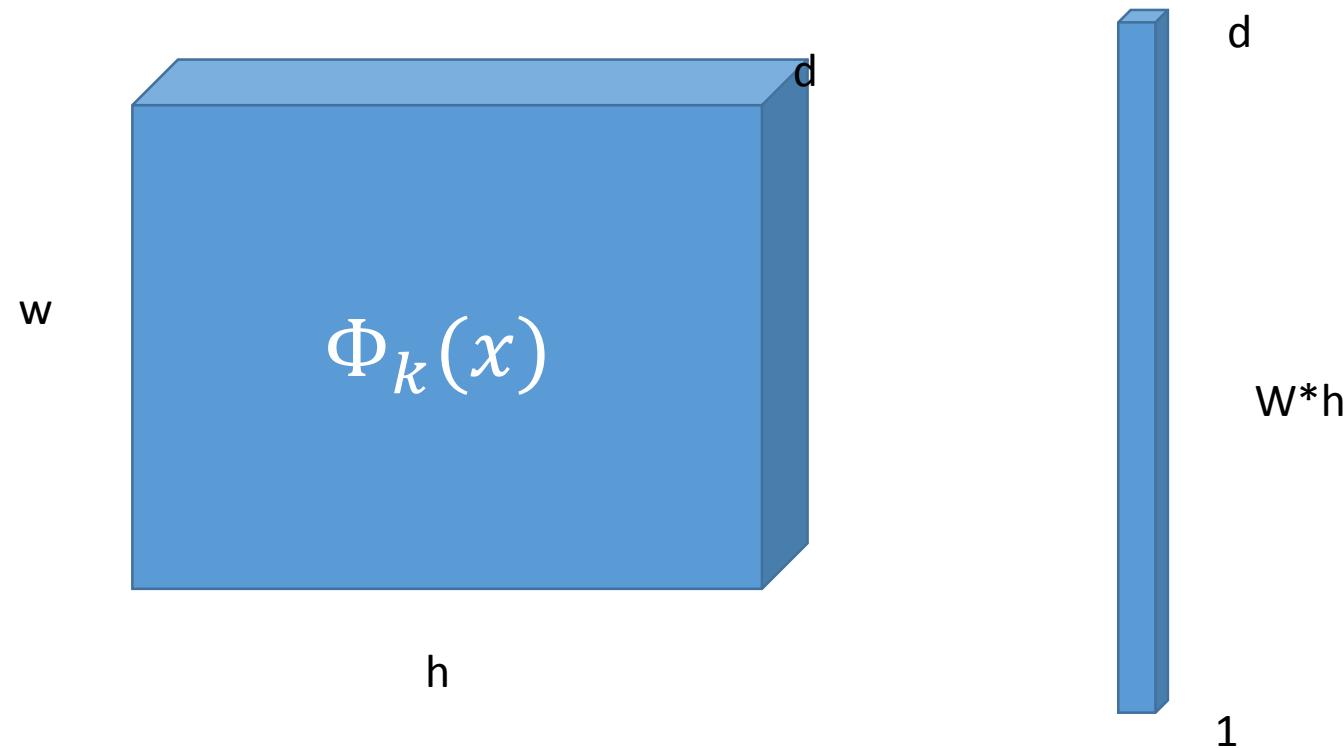
D

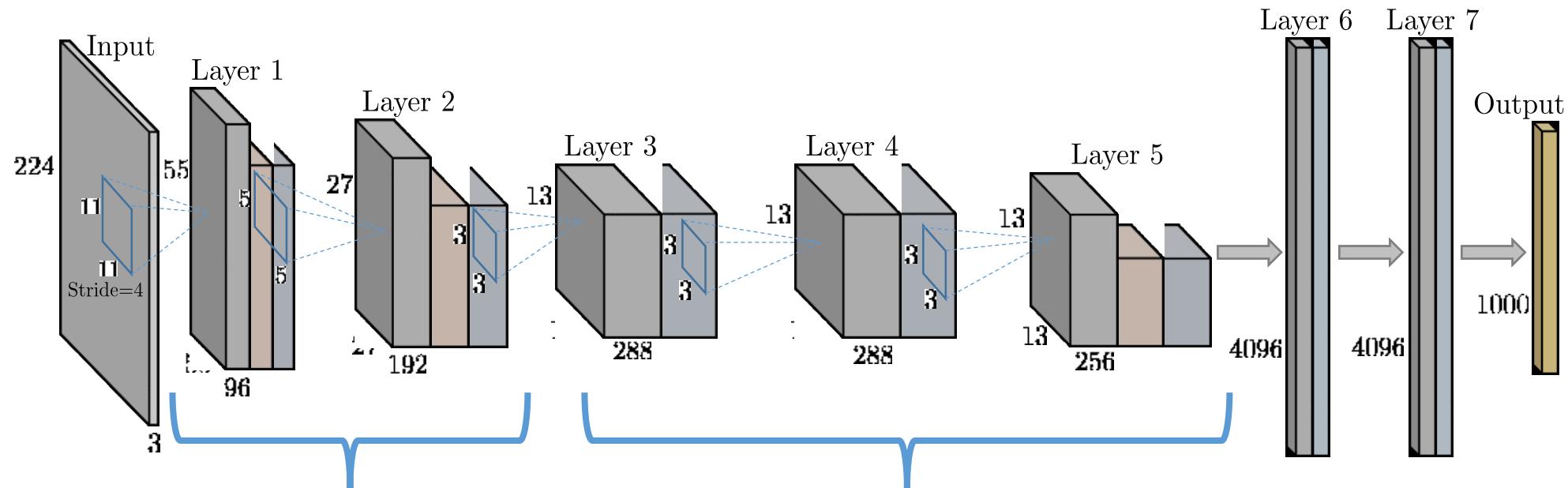


[Gatys et al. 2015]

# Матрица Грамма

$$\bullet G(x_1, \dots, x_n) = \begin{pmatrix} < x_1, x_1 > & \cdots & < x_n, x_1 > \\ \vdots & \ddots & \vdots \\ < x_n, x_1 > & \cdots & < x_n, x_n > \end{pmatrix}$$



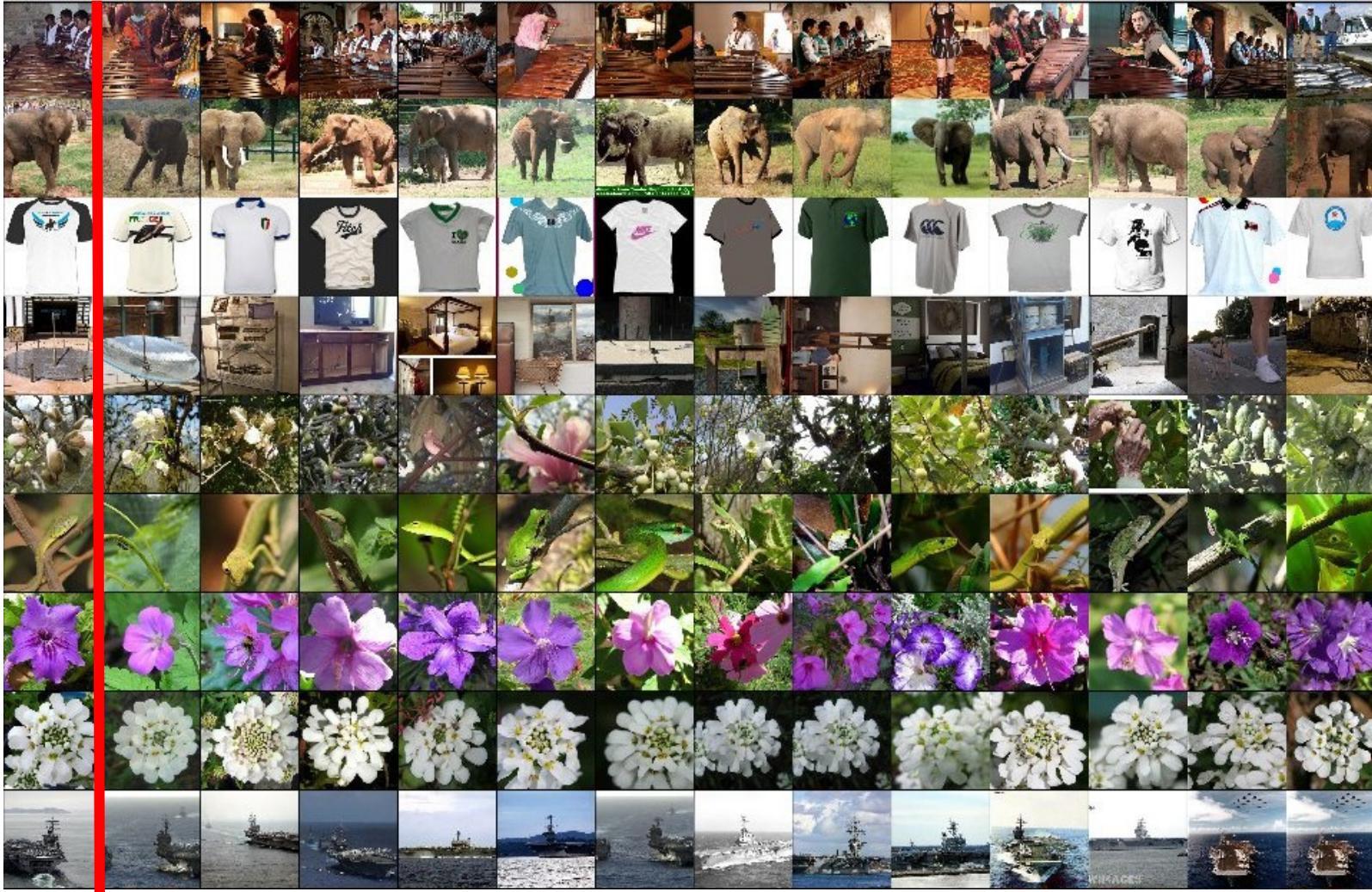


$$L_{Style}(x, x_s) = \sum_{i \in Style} \|G(\Phi_i(x)) - G(\Phi_i(x_s))\|$$

$$L_{Content}(x, x_c) = \sum_{i \in Content} \|\Phi_i(x) - \Phi_i(x_c)\|$$

$$\hat{x} = \operatorname{argmin}_x L_{Style}(x, x_s) + \lambda L_{Content}(x, x_c)$$

# Поиск изображений по пространству признаков



[Krizhevsky et al. NIPS12]

# Поиск изображений с использованием признаков генерируемых нейронной сетью



$$\Phi_5(x)$$



$$\Phi_6(x)$$



$$\Phi_7(x)$$

[Babenko et al. 2014]

# Поиск изображений с использованием признаков генерируемых нейронной сетью



$$\Phi_5(x)$$



$$\Phi_6(x)$$



$$\Phi_7(x)$$

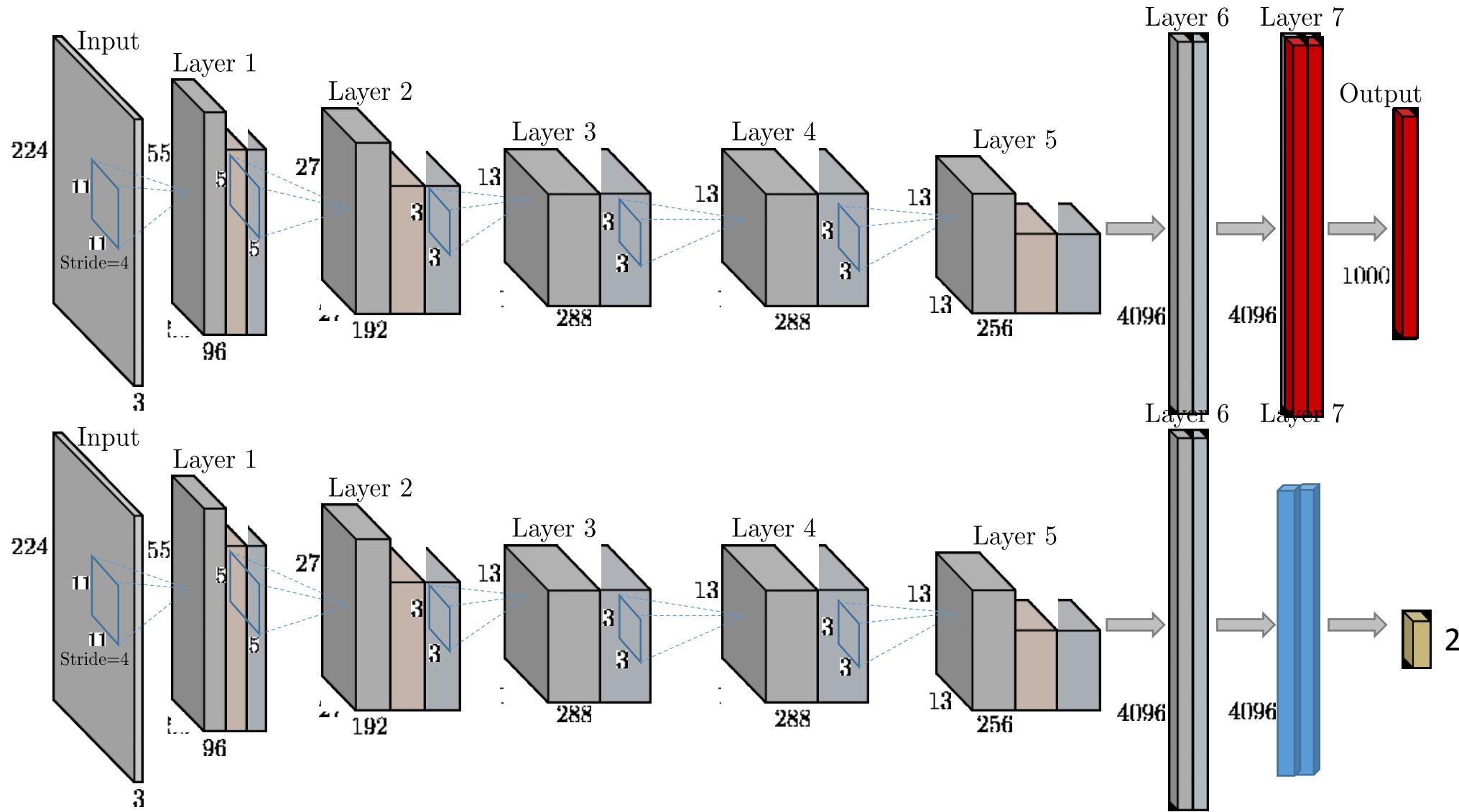
[Babenko et al. 2014]

# Создание компактных индексов поиска

- $\Psi(x) = \sum_{p,q} \Phi_k(x)[p, q]$
- $\Psi'(x) = M_{PCA}(\Phi_k(x))$
- $\Psi''(x) = \Psi'(x) / \|\Psi'(x)\|$

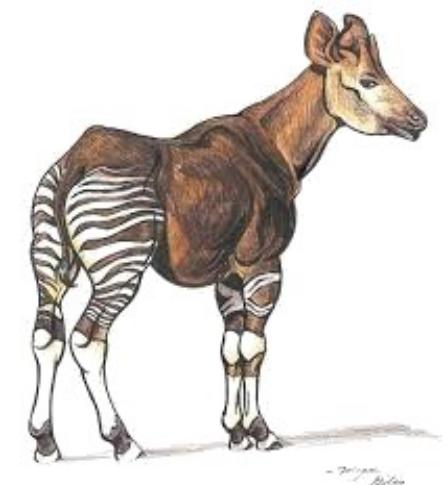
[Babenko et al. ICCV2015]

# Повторное использование глубоких слоев



# До обучения на практике

Новая задача



PyTorch torchvision

# Практика transfer learning

- Загрузить архитектуру AlexNet с весами из torchvision
- Создать свою сеть на основе AlexNet с выходом ожидаемой размерности
- Зафиксировать веса всех слоев, кроме новых
- До обучить сеть решать текущую задачу