

# Компьютерное зрение

Лекция 2. Сверточные нейронные сети.  
Базовые блоки, принципы и приложения

30.05.2020  
Руслан Рахимов

# План

- Что такое операция свертка
- Основные блоки
- Общий вид архитектуры сверточных нейронных сетей
- Визуализация слоев, Deep Dream
- Style Transfer

# ImageNet Challenge



mite

container ship

motor scooter

leopard

mite	container ship	motor scooter	leopard
black widow	lifeboat	go-kart	leopard
cockroach	amphibian	moped	jaguar
tick	fireboat	bumper car	cheetah
starfish	drilling platform	golfcart	snow leopard
			Egyptian cat



grille

mushroom

cherry

Madagascar cat

convertible	agaric	dalmatian	squirrel monkey
grille	mushroom	grape	spider monkey
pickup	jelly fungus	elderberry	titi
beach wagon	gill fungus	ffordshire bullterrier	indri
fire engine	dead-man's-fingers	currant	howler monkey

# Deep Dream



# Style Transfer



PRISMA

# Сверточные нейронные сети (CNNs)

# Мотивация

- изображения представляют собой объекты с большим количеством признаков
- изображение в формате RGB размера 640x480 будет иметь ~1млн признаков
- число параметров полносвязной сети с внутренним слоем из 10 нейронов равно ~10млн
- большое число параметров модели существенно затрудняет процесс обучения

# Операция свертки

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v]$$

F - исходное изображение

H - фильтр размера kxk

G - результат на выходе свертки

i, j - координаты пикселя в районе которого применяется операция свертки

# Свертка двух матриц

Input =

0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter =

1	0	1
0	1	0
1	0	1

0	0	1
0	0	1
0	1	1

dot

1	0	1
0	1	0
1	0	1

# Свертка двух матриц

Input =

0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter =

1	0	1
0	1	0
1	0	1

0	0	1
0	0	1
0	1	1

dot

1	0	1
0	1	0
1	0	1

=

2

# Свертка двух матриц

Input =

0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter =

1	0	1
0	1	0
1	0	1

# Свертка двух матриц

Input =

0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter =

1	0	1
0	1	0
1	0	1

dot

0	1	1
0	1	1
1	1	0

1	0	1
0	1	0
1	0	1

= 3

# Как посчитать свертку

1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	0	0
0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1	0
0 <small><math>\times 1</math></small>	0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

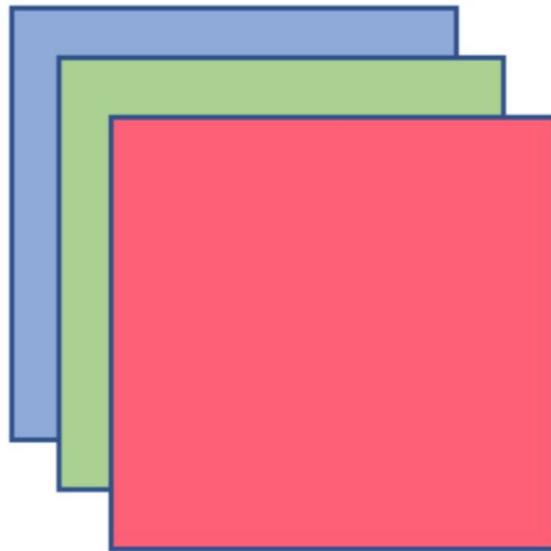
Convolved  
Feature

# Как посчитать свертку

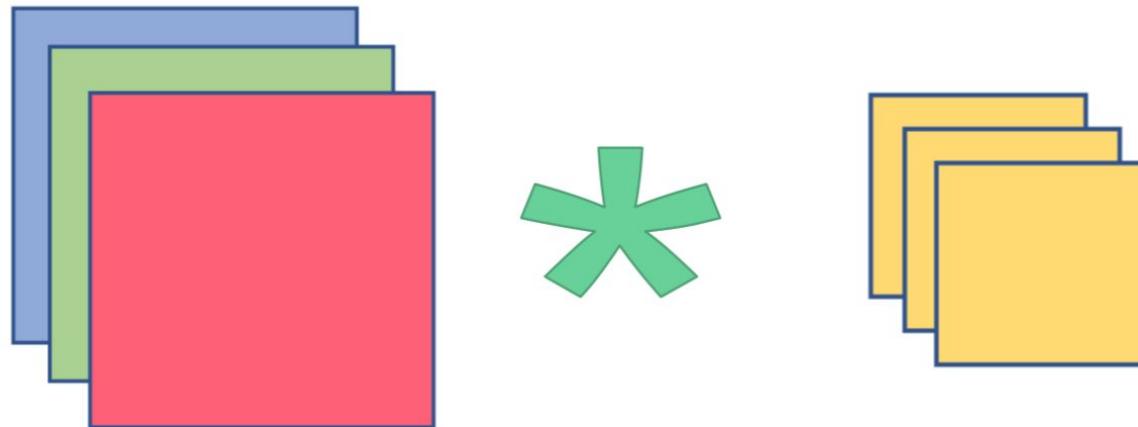
1. Имея входное изображение, наложить на него маленькое окно (ядро свертки). Найти произведение ядра и участка изображения которое накрыло ядро
2. Сдвинуть окно и повторить процедуру
3. Повторить шаги 1 и 2 для всех остальных выходных каналов уже с другими ядрами

# Объемная свертка (Volume Convolution,Conv2d)

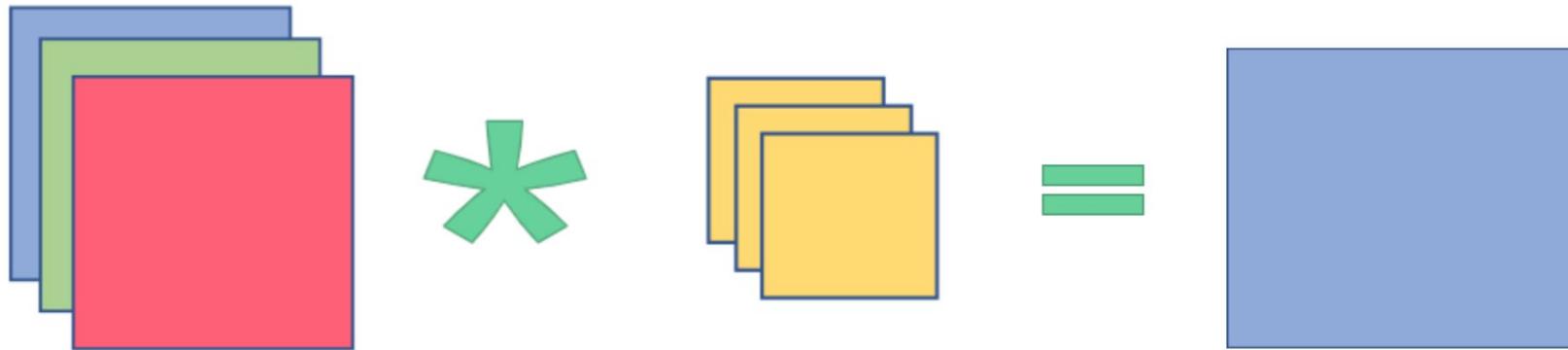
- Картинка имеет 3 канала, т.е. это 3d матрица



# Объемная свертка (Volume Convolution,Conv2d)



# Объемная свертка (Volume Convolution,Conv2d)



# Как применить свертку?

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...	...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...	...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...	...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



308

+



-498

+

164

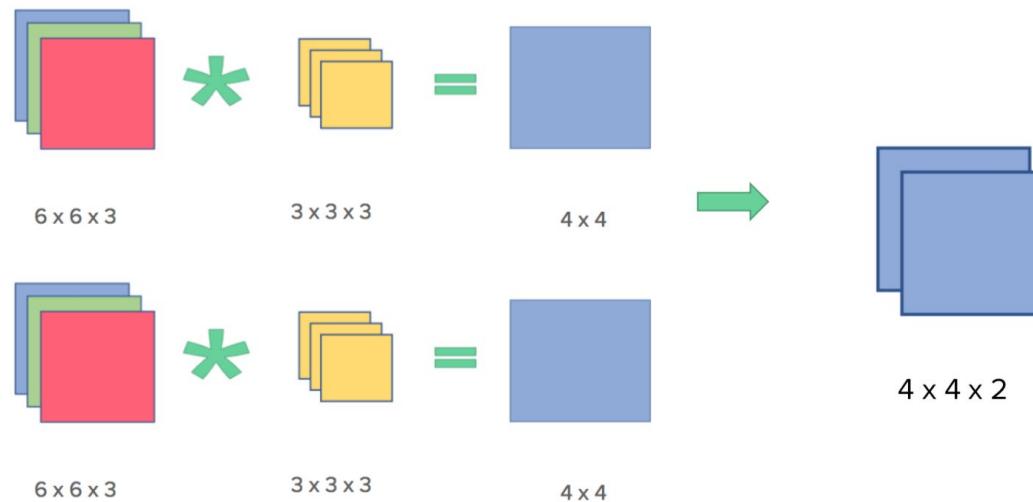
+ 1 = -25

Bias = 1

-25				...
				...
				...
				...
...	...	...	...	...

Output

# Сверточный слой



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

# Что делает свертка

Двигать окно по изображению - что это нам дает?

1. Local connectivity

свертка как способ извлекать локальные признаки - это дает

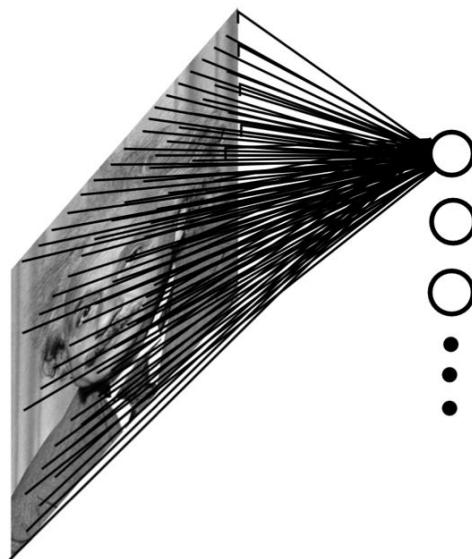
2. Weight (Parameter) sharing

свертка как способ уменьшить сложность модели

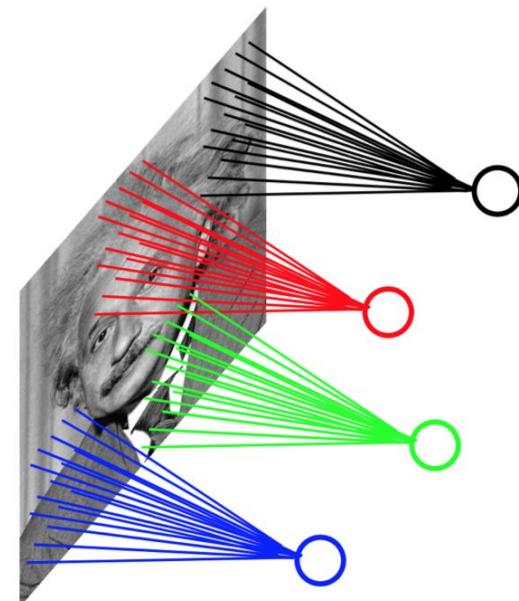
# Свертка как детектор признаков



# Local connectivity

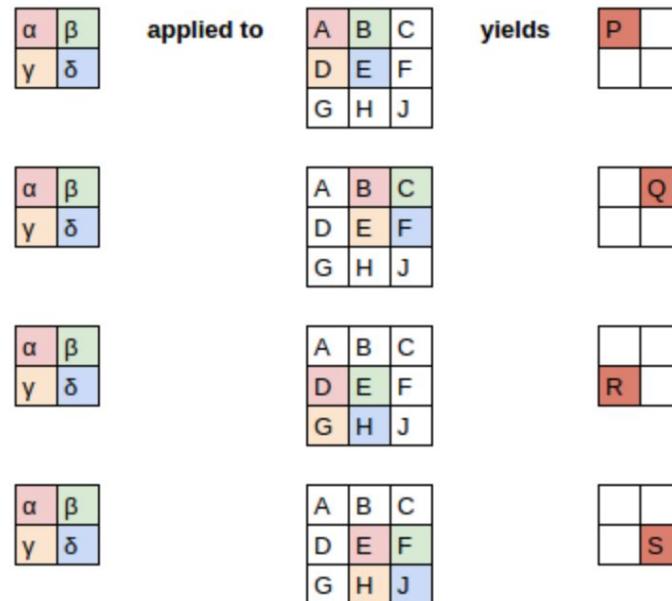


Fully Connected: 200x200 image, 40K hidden  
units, **~2B parameters**



Locally Connected: 200x200 image, filter size  
10x10, **4M parameters!**

# Weight sharing



# Weight sharing

$\begin{matrix} \alpha & \beta \\ \gamma & \delta \end{matrix}$	applied to	$\begin{matrix} A & B & C \\ D & E & F \\ G & H & J \end{matrix}$	yields	$\begin{matrix} P \\ \square \\ \square \end{matrix}$	$\alpha * A + \beta * C + \gamma * D + \delta * F + b = P$
$\begin{matrix} \alpha & \beta \\ \gamma & \delta \end{matrix}$		$\begin{matrix} A & B & C \\ D & E & F \\ G & H & J \end{matrix}$		$\begin{matrix} Q \\ \square \\ \square \end{matrix}$	$\alpha * B + \beta * E + \gamma * F + \delta * H + b = Q$
$\begin{matrix} \alpha & \beta \\ \gamma & \delta \end{matrix}$		$\begin{matrix} A & B & C \\ D & E & F \\ G & H & J \end{matrix}$		$\begin{matrix} R \\ \square \\ \square \end{matrix}$	$\alpha * D + \beta * E + \gamma * G + \delta * H + b = R$
$\begin{matrix} \alpha & \beta \\ \gamma & \delta \end{matrix}$		$\begin{matrix} A & B & C \\ D & E & F \\ G & H & J \end{matrix}$		$\begin{matrix} S \\ \square \\ \square \end{matrix}$	$\alpha * E + \beta * F + \gamma * H + \delta * J + b = S$

# Weight sharing

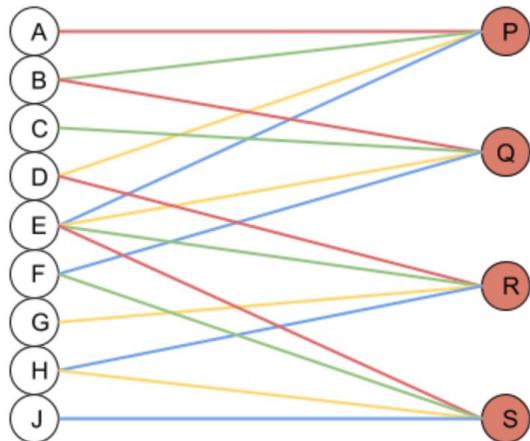
$$\begin{array}{c} \begin{array}{|c|c|} \hline \alpha & \beta \\ \hline Y & \delta \\ \hline \end{array} \end{array} \text{ applied to } \begin{array}{c} \begin{array}{|c|c|c|} \hline A & B & C \\ \hline D & E & F \\ \hline G & H & J \\ \hline \end{array} \end{array} \text{ yields } \begin{array}{c} \begin{array}{|c|} \hline P \\ \hline \end{array} \end{array}$$
  

$$\begin{array}{c} \begin{array}{|c|c|} \hline \alpha & \beta \\ \hline Y & \delta \\ \hline \end{array} \end{array} \text{ applied to } \begin{array}{c} \begin{array}{|c|c|c|} \hline A & B & C \\ \hline D & E & F \\ \hline G & H & J \\ \hline \end{array} \end{array} \text{ yields } \begin{array}{c} \begin{array}{|c|} \hline Q \\ \hline \end{array} \end{array}$$
  

$$\begin{array}{c} \begin{array}{|c|c|} \hline \alpha & \beta \\ \hline Y & \delta \\ \hline \end{array} \end{array} \text{ applied to } \begin{array}{c} \begin{array}{|c|c|c|} \hline A & B & C \\ \hline D & E & F \\ \hline G & H & J \\ \hline \end{array} \end{array} \text{ yields } \begin{array}{c} \begin{array}{|c|} \hline R \\ \hline \end{array} \end{array}$$
  

$$\begin{array}{c} \begin{array}{|c|c|} \hline \alpha & \beta \\ \hline Y & \delta \\ \hline \end{array} \end{array} \text{ applied to } \begin{array}{c} \begin{array}{|c|c|c|} \hline A & B & C \\ \hline D & E & F \\ \hline G & H & J \\ \hline \end{array} \end{array} \text{ yields } \begin{array}{c} \begin{array}{|c|} \hline S \\ \hline \end{array} \end{array}$$

$$\begin{array}{l} \alpha * A + \beta * B + \gamma * C + \delta * D + \epsilon * E + b = P \\ \alpha * B + \beta * C + \gamma * D + \delta * E + \epsilon * F + b = Q \\ \alpha * D + \beta * E + \gamma * F + \delta * G + \epsilon * H + b = R \\ \alpha * E + \beta * F + \gamma * G + \delta * H + \epsilon * I + b = S \end{array}$$



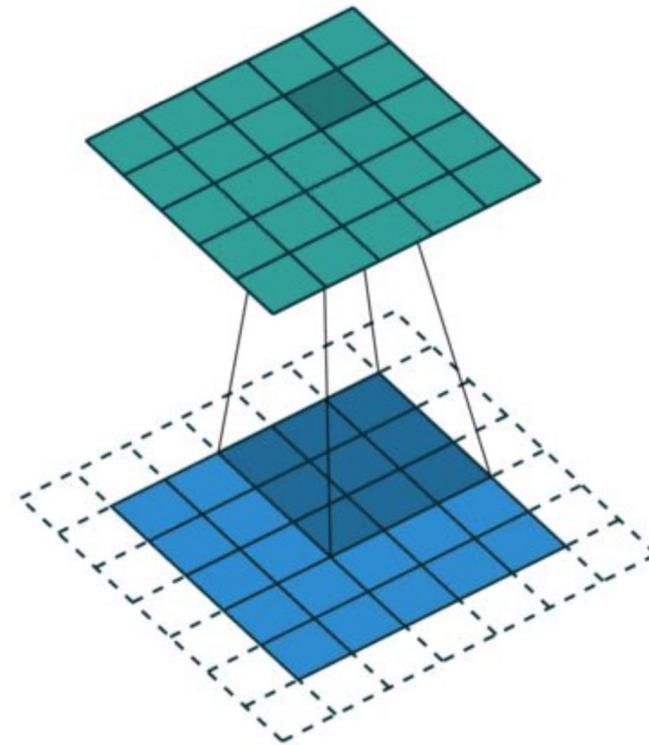
$$\begin{array}{c} \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline \alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta & 0 \\ \hline \end{array} \end{array} * \begin{array}{c} \begin{array}{|c|} \hline A \\ \hline B \\ \hline C \\ \hline D \\ \hline E \\ \hline F \\ \hline G \\ \hline H \\ \hline J \\ \hline \end{array} \end{array} + \begin{array}{c} \begin{array}{|c|} \hline b \\ \hline \end{array} \end{array} = \begin{array}{l} \alpha A + \beta B + 0C + \gamma D + \delta E + 0F + 0G + 0H + 0J + b \\ 0A + \alpha B + \beta C + 0D + \gamma E + \delta F + 0G + 0H + 0J + b \\ 0A + 0B + 0C + \alpha D + \beta E + 0F + \gamma G + \delta H + 0J + b \\ 0A + 0B + 0C + 0D + \alpha E + \beta F + 0G + \gamma H + \delta J + b \end{array} = \begin{array}{c} \begin{array}{|c|} \hline \alpha A + \beta B + \gamma D + \delta E + b \\ \hline \alpha B + \beta C + \gamma E + \delta F + b \\ \hline \alpha D + \beta E + \gamma G + \delta H + b \\ \hline \alpha E + \beta F + \gamma H + \delta J + b \\ \hline \end{array} \end{array} = \begin{array}{c} \begin{array}{|c|} \hline P \\ \hline Q \\ \hline R \\ \hline S \\ \hline \end{array} \end{array}$$

# Как применить свертку

Вопросы на которые мы еще не ответили включают:

1. Размер изображения на выходе меньше чем на входе и мы не знаем как хорошо обрабатывать границы
2. Накладываем ядро свертки на каждый участок изображения или можем двигаться с интервалом?

# Отступы (padding)

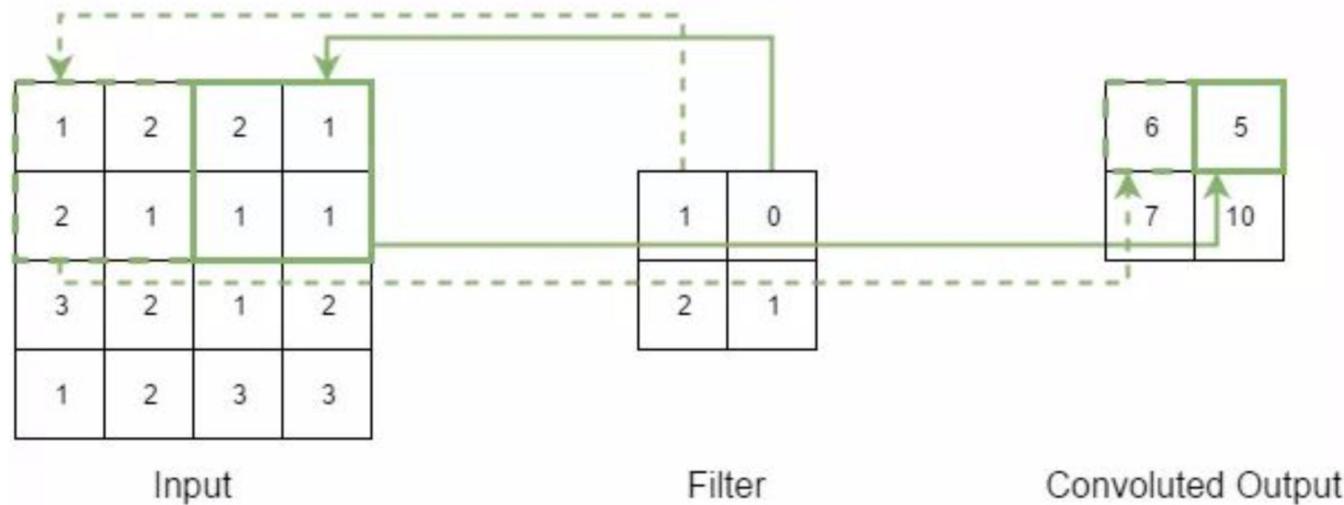


# Отступы (padding)

- чтобы размер на выходе совпадал с размером на входе, размер исходного изображения увеличивают, заполняя пространство по периметру например определенными значениями
- zero padding
- replication padding
- reflection padding

# Шаг (Stride)

Stride (2,2) Convolution



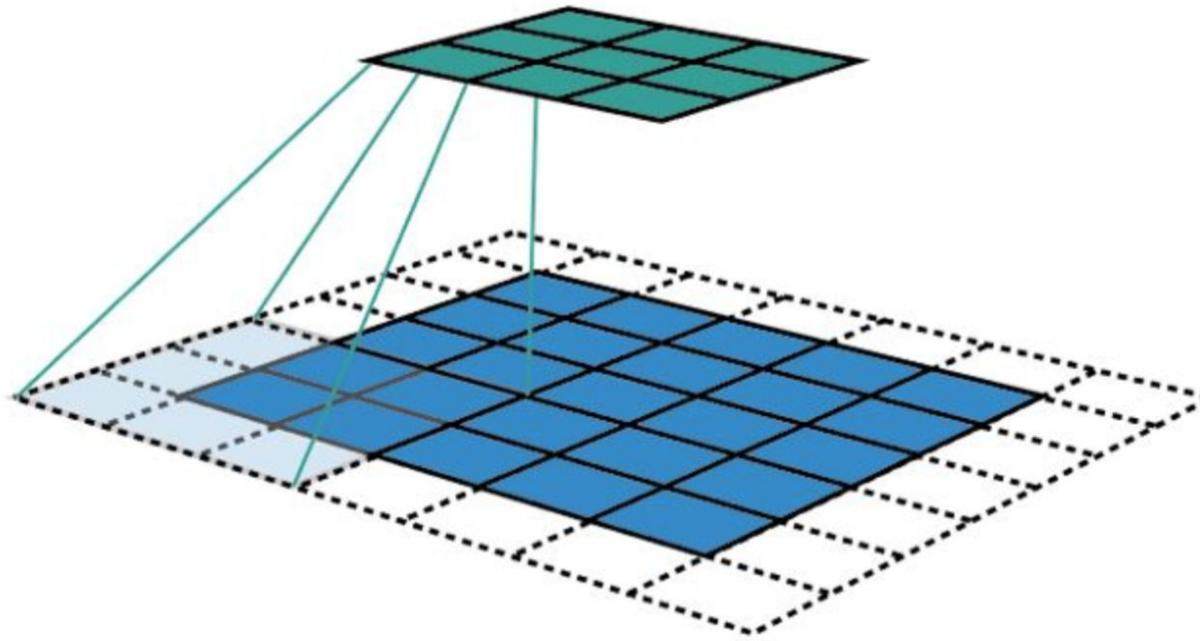
# Шаг (Stride)

- шаг может быть разным по осям x и y, но в основном используют одинаковый
- определяет смещение фильтра на очередной итерации
- при увеличении шага уменьшается размер выхода
- шаг  $> 1$  в начальных слоях
- шаг  $> 1$  увеличивает receptive field (далее)

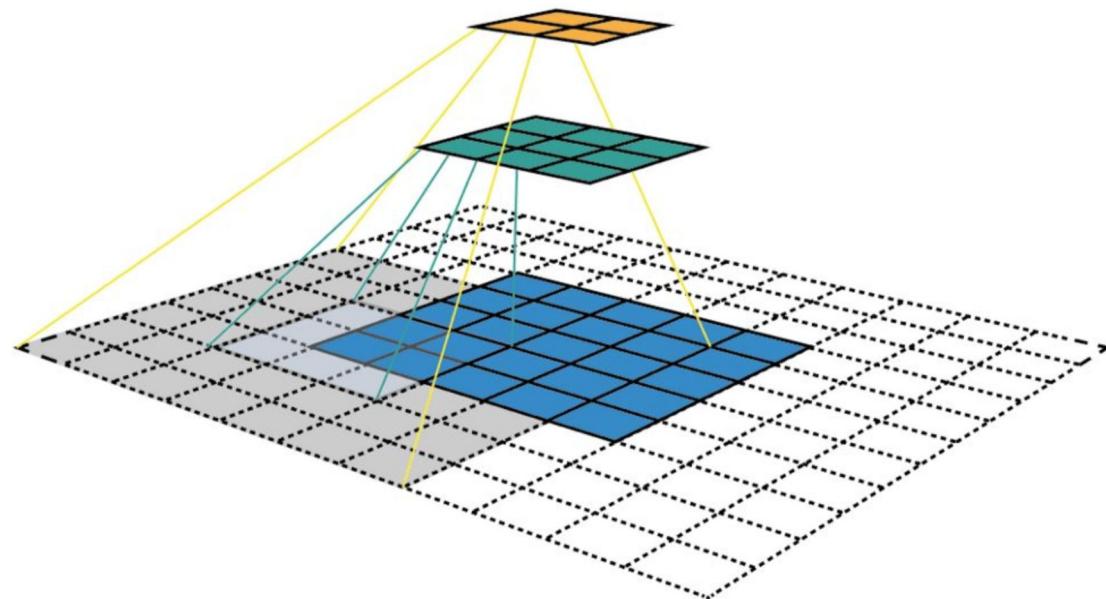
# Область видимости фильтра (reception field)

- область видимости - область исходного изображения от которой зависит значение пикселя на выходе сверточного слоя
- область видимости отдельного пикселя на выходе сверточного слоя возрастает с увеличением глубины сети

# Область видимости фильтра (reception field)



# Область видимости фильтра (reception field)



# Pooling Layer

# Pooling Layer

Тензор признаков на выходе свертки говорит нам о:

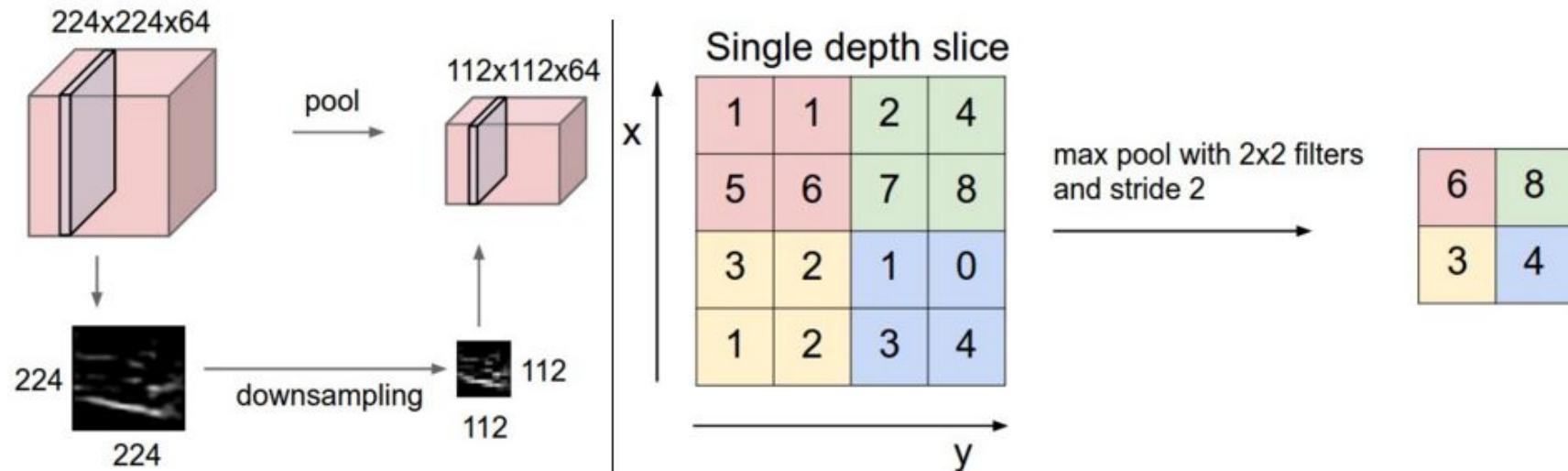
1. присутствует ли признак на изображении (большая активация)
2. если да, то где примерно был найден на изображении

Мы можем сохранить оба свойства достаточно хорошо с помощью операции pooling (например max или average)

# Отступы (padding)

- при последовательном применении фильтров размер выхода будет уменьшаться с каждым шагом на размер фильтра
- при этом теряется информация о краях изображения

# Pooling Layer



# Pooling Layer

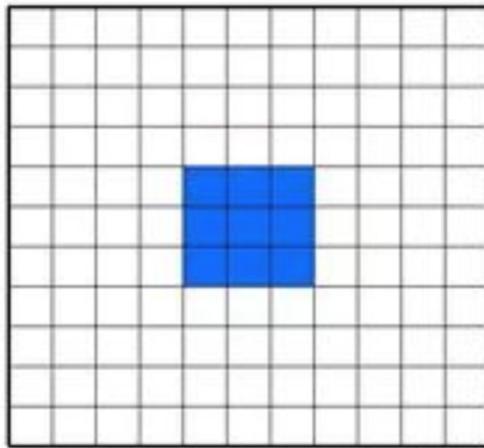
- pooling слой вставляется после активаций сверточного слоя
- уменьшает пространственный размер данных
- как результат уменьшается число параметров сети
- необучаемый слой - не содержит параметров

# Расширение (dilation)

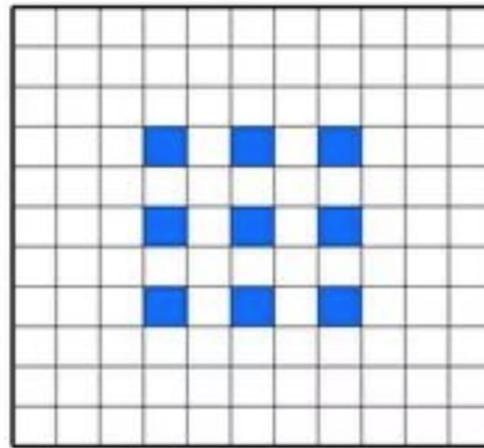
- увеличение размера фильтра (области видимости) без увеличения числа параметров (весов)
- веса фильтра “раздвигаются” в пространстве
- фильтр имеет разреженную структуру
- свободные позиции фильтра заполняются нулями
- число свободных позиций является гипер-параметром
- свертки с расширением именуются atrous (dilated) convolutions

# Расширение (dilation)

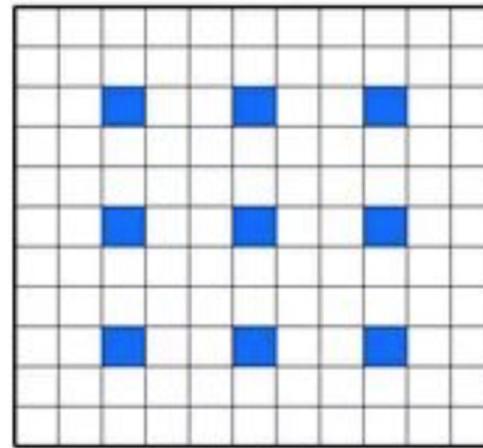
D = 1



D = 2



D = 3



# Математика

- Размер картинки (cin, h, w)
- Размер фильтра (cout, cin, k1, k2)
  - Обычно  $k_1=k_2=f$ =нечетному числу, например 3, 5, 7
- Допустим  $h = n$ ,  $w = n$  тогда размер картинки на выходе (cout, m, m)

$$m = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1$$

- где  $S$  - размер шага,  $p$  значение паддинга с одной стороны

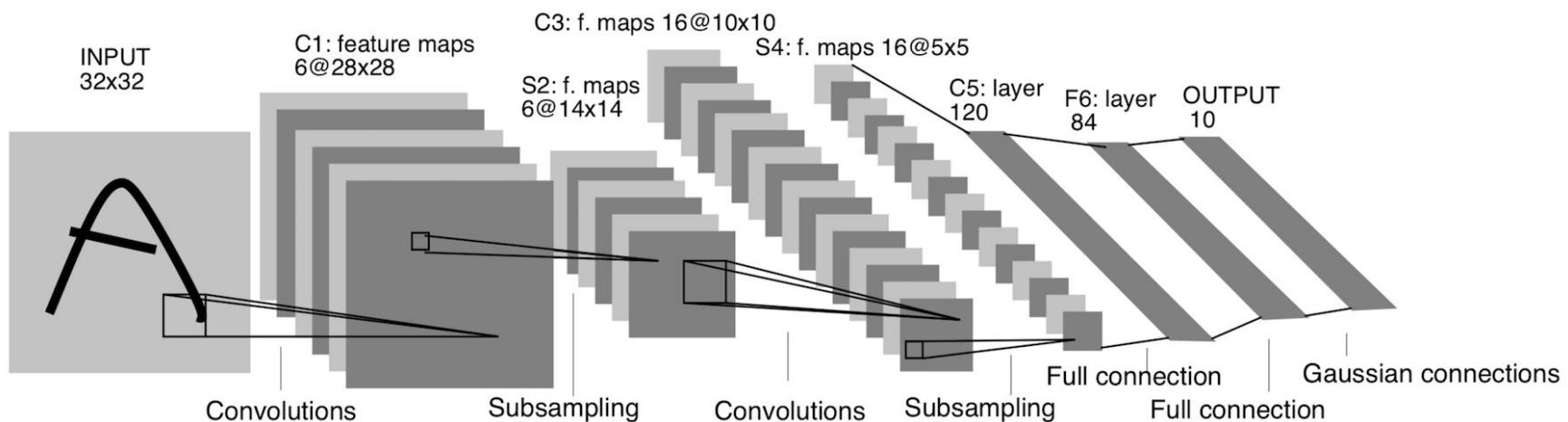
# Сверточный слой (Convolution layer)

- Содержит набор из cout фильтров одинакового размера (cin, f, f)
- Размер входа (b, cin, h, w)
- Размер выхода (b, cout, h~, w~)

где:

- b - размер батча
  - cin - число каналов на входе
  - cout - число каналов на выходе
- 
- На выходе применяется нелинейная активация

# LeNet (1998)



# Сверточные нейронные сети

- в нейронных сетях для обработки изображений полносвязные слои заменяют на сверточные
- полносвязные слои (+dropout) могут использоваться в выходном слое
- сверточный блок обычно имеет следующий вид:  
Conv2d -> нормализация -> нелинейная активация (н: ReLU) -> Pooling

# Invariance

- CNNs обладают интересным свойством translation (shift) invariance, т.е. если мы сдвинем объект на фото, сеть все равно можно распознать его, так как свертка - операция локальная
- Pooling layers тоже добавляют свойство инвариантности, но это invariance to small perturbations, т.е. если появляется какой-то шум после активаций, pooling позволяет снизить влияние шума

# Invariance vs Equivariance

- Часто это эти понятия путают



**Shift-equivariance and invariance** A function  $\tilde{\mathcal{F}}$  is shift-equivariant if shifting the input equally shifts the output, meaning shifting and feature extraction are commutable.

$$\text{Shift}_{\Delta h, \Delta w}(\tilde{\mathcal{F}}(X)) = \tilde{\mathcal{F}}(\text{Shift}_{\Delta h, \Delta w}(X)) \quad \forall (\Delta h, \Delta w) \quad (1)$$

A representation is shift-invariant if shifting the input results in an *identical* representation.

$$\tilde{\mathcal{F}}(X) = \tilde{\mathcal{F}}(\text{Shift}_{\Delta h, \Delta w}(X)) \quad \forall (\Delta h, \Delta w) \quad (2)$$

# Нормализация

# Нормализация

- чаще всего дает прирост как в качестве, так и в скорости сходимости процесса обучения сети
- повышает обобщающую способность сети

# Batch normalization

- приводит значения активаций на выходе слоя к нулевому среднему и единичной дисперсии
- во время обучения среднее и дисперсия оцениваются для каждого батча
- далее применяется аффинное преобразование (optional)
- самый популярный вариант

# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Batch normalization (режимы работы)

- train or eval mode
  - во время обучения используются текущие статистики батча
  - во время тестирования (inference) используются бегущие средние статистик, собранные во время обучения
- affine = True (default) or affine = False
  - применять ли scale и shift после нормализации или нет

# Batch normalization (минусы)

- в случае маленького размера батча, могут появиться коррелированные по контенту изображения или зависимости между картинками, что плохо
- требуется большой размер матча, обычно  $\geq 16$
- такой размер батча иногда сложно использовать, так как потребление памяти очень сильно растет с увеличением модели
- возможное решение synchronous batch norm
  - если у вас много gpu
  - статистики батча собираются со всех видеокарт
- еще решение - убрать зависимость от размера батча, использовать другую нормализацию

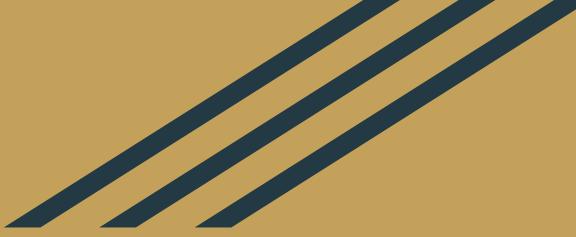
# Другие известные нормализации

В этих нормализациях зависимость от размера батча исчезает, делаем для каждой картинки по-отдельности

- Layer normalization
  - среднее и дисперсия считаются по всем каналам по всем пикселям Local response normalization
- Local response normalization
  - среднее и дисперсия считаются по группам каналов для каждого пикселя
  - практически потеряла популярность
- Group normalization
  - среднее и дисперсия считаются по группам каналов по всем пикселям

# Другие виды нормализаций (no details yet)

- Weight normalization
- Filter response normalization
- Spectral Normalization (в основном для GANs)
- Instance normalization (в основном для style transfer)
- Adaptive instance normalization (в основном для style transfer)
- ...

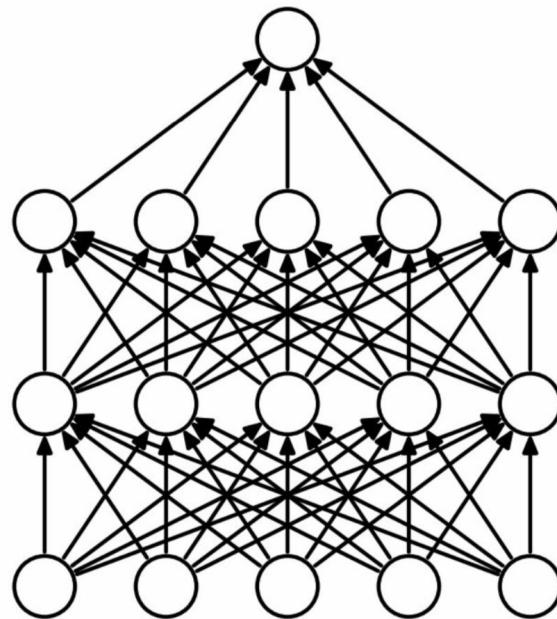


# Dropout

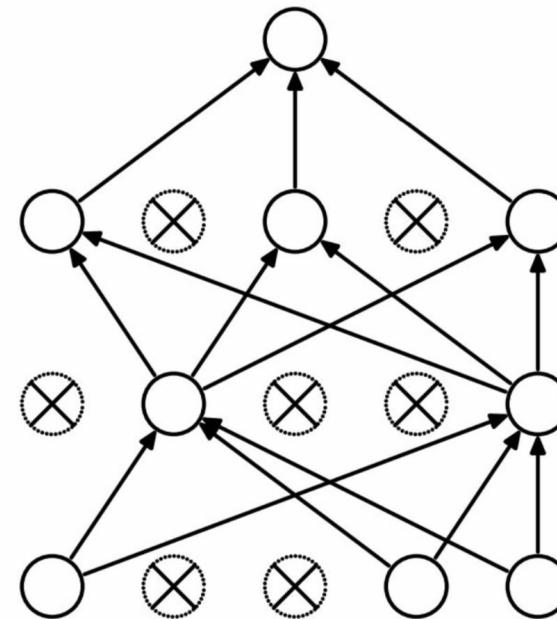
# Dropout

- в процессе обучения случайным образом выбранная часть нейронов (каналов в случае CNN) слоя зануляется
- доля зануляемых входов является гипер-параметром  $p$
- на этапе обучения значения выходов делится на  $p$
- на этапе тестирования dropout не используется
- является способом регуляризации, т.е. позволяет модели быть более устойчивой к переобучению

# Dropout

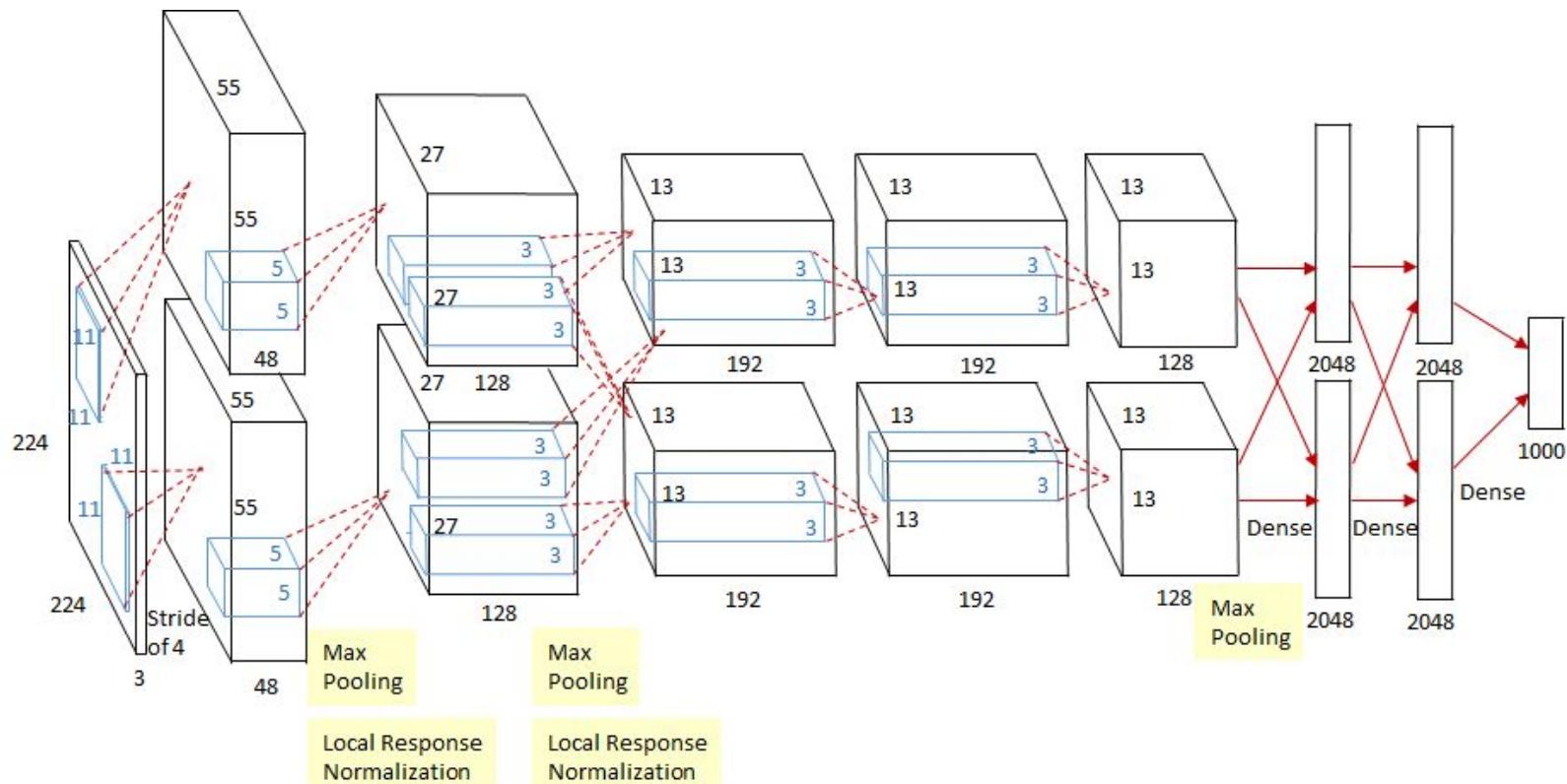


(a) Standard Neural Net



(b) After applying dropout.

# AlexNet (2012)

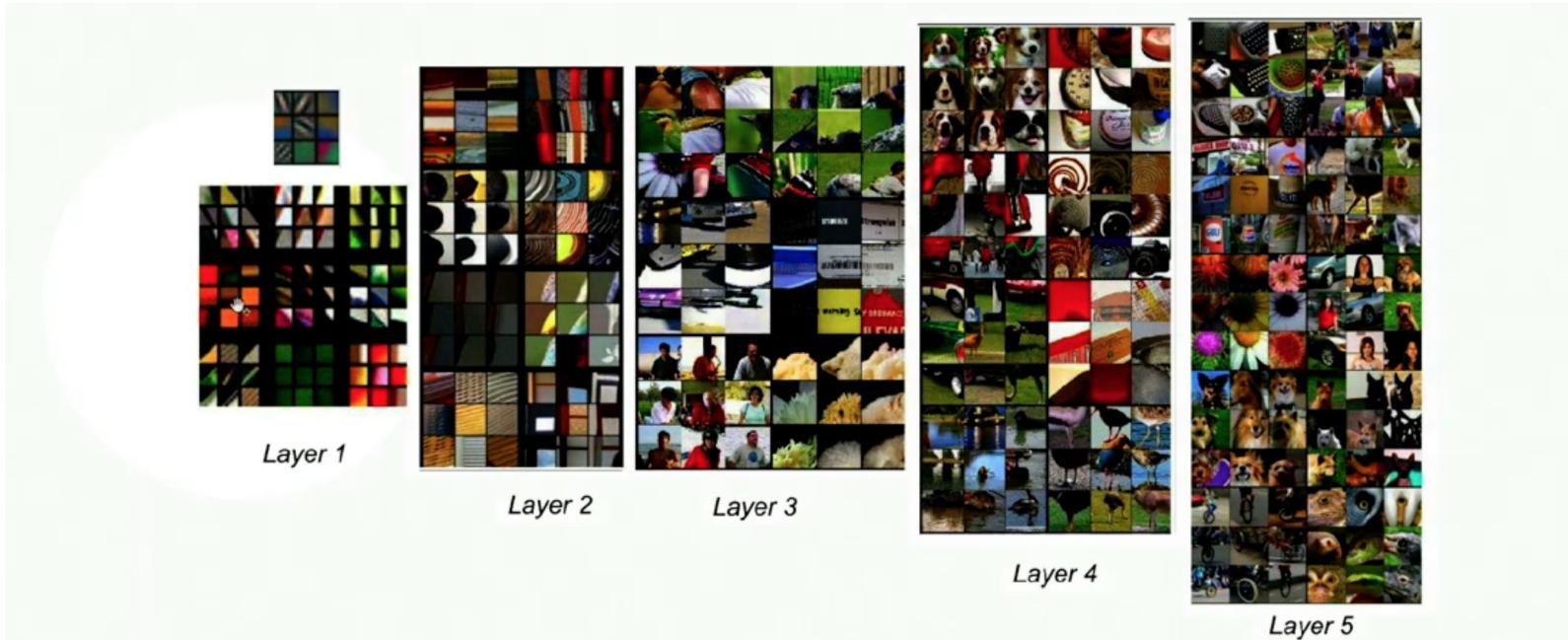


# Визуализация сверточных сетей

# Layers interpretability (Zeiler et al.)

- выбирается слой сверточной сети (в статье AlexNet) и ищется картинка, которая максимизирует активацию нейронов этого слоя
- первые слои реагируют больше всего на простые геометрические формы
- с увеличением глубины фильтры выучивают более сложные паттерны, которые визуально становятся похожи на части объектов

# Layers interpretability (Zeiler et al.)



**Figure:** layers responsibilities (from Matthew D Zeiler et al. (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833)

# Deep Dream

- выбираем предобученную сеть
- фиксируем интересующий нас сверточный слой
- подаем картинку на вход и вычисляем активации на выбранном слое
- распространяем обратно значения активаций как значения градиента
- обновляем изображение с помощью полученных градиентов и повторяем итерацию

# Deep Dream



Horizon



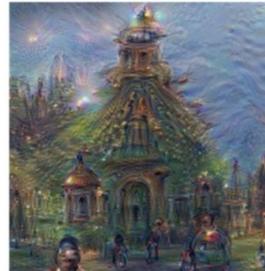
Trees



Leaves



Towers & Pagodas



Buildings



Birds & Insects

# Deep Dream

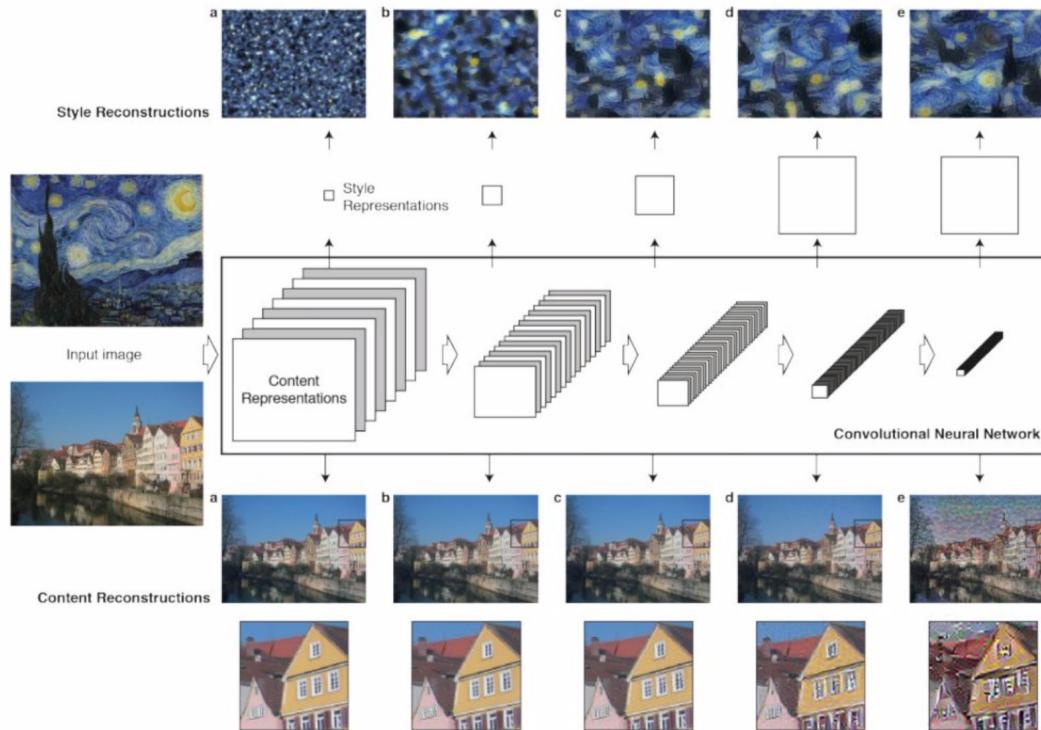
- результат будет зависеть от того, на каких данных обучалась нейронная сеть
- детали рисунка будут зависеть от глубины слоя с которого распространяются активации

# Style Transfer

# Style Transfer

- сгенерируем такое изображение, которое максимально похоже на исходное
- при этом, стиль сгенерированного изображения должен быть похож на стиль референсного изображения
- активации сгенерированного изображения должны быть близки к активациям исходного
- ковариационная матрица фильтров сгенерированного изображения должна быть близка к матрице референсного изображения

# Style Transfer (Gatys et al. 2015)



# Style Transfer - Content Loss

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

# Style Transfer - Style Loss

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

i,j - индексы фильтров

k - номер активации

Fk - значение активации k-го пикселя

G, A - матрица ковариации фильтров для заданного слоя

N - число фильтров в слое

M - число активаций (размер изображения на выходе сверточного слоя)

E - мера расхождения в стиле

w - вес ошибки для заданного слоя

L - число слоев

# Style Transfer - Total Loss

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

р - исходное изображение

а - изображение с референсным стилем

х - сгенерированное изображение

alpha, beta - параметры обучения

# Style Transfer

1

## Upload photo

The first picture defines the scene you would like to have painted.



2

## Choose style

Choose among predefined styles or upload your own style image.



3

## Submit

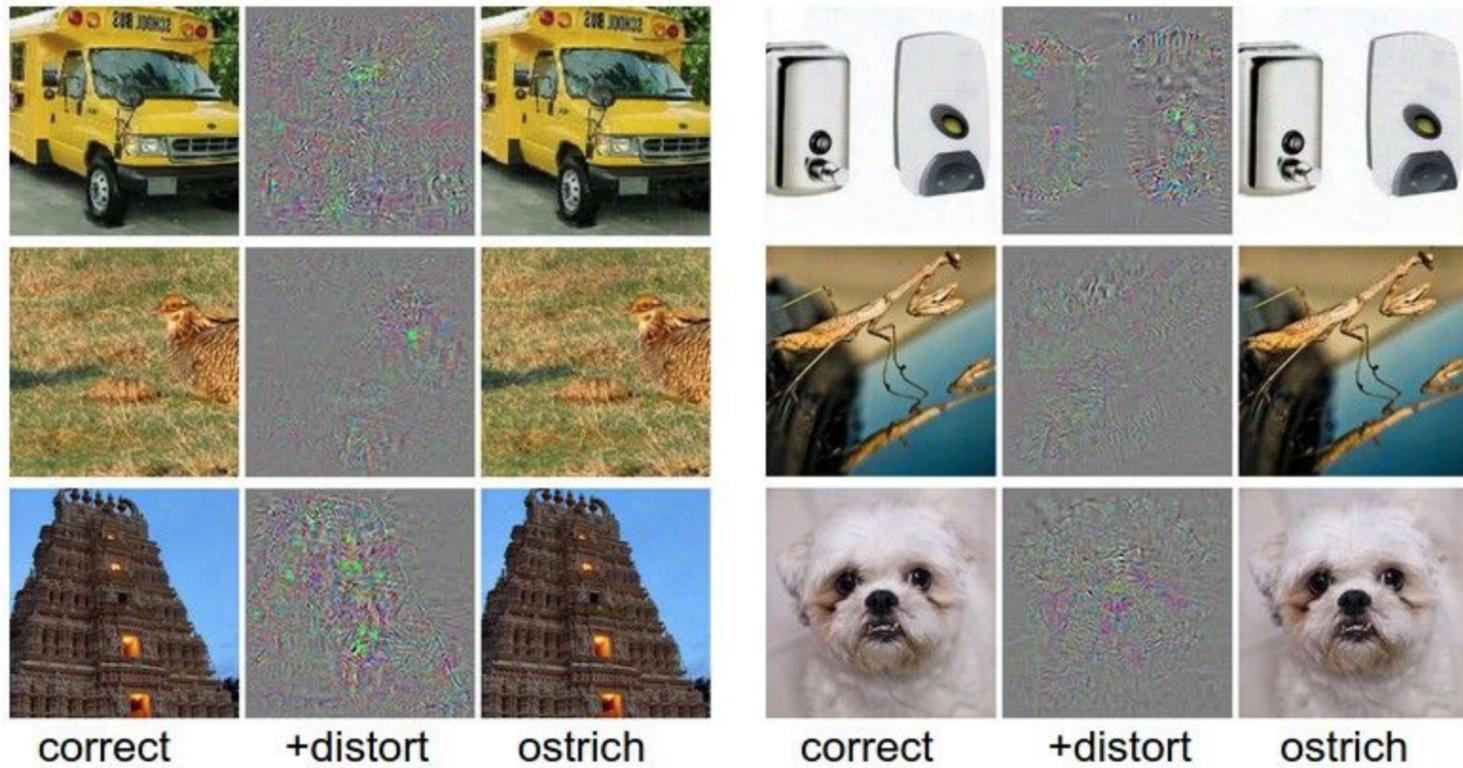
Our servers paint the image for you. You get an email when it's done.



<https://deepart.io>

# Обманываем сверточную сеть (Adversarial attacks)

# Обманываем сверточную сеть



# Recap

- в отличие от полносвязных, сверточные сети содержат меньше параметров, как результат менее склонны к переобучению
- основные параметры свертки: размер фильтра, receptive field
- pooling layer и stride  $> 1$  помогает увеличить receptive field
- основной операцией в процессе работы сверточной нейронной сети является перемножение матриц

# Recap

- с увеличением глубины слоя признаки становятся более общими (увеличивается область видимости изображения)
- для оценки стиля изображения можно использовать ковариационную матрицу фильтров для каждого слоя
- добавление специального шума может заставить нейронную сеть выдавать неверные ответы

# Полезные материалы

- [A guide to convolution arithmetic for deep learning](#)
- [Making Convolutional Networks Shift-Invariant Again](#)
- [Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization](#)