

# Глубокое обучение и вообще

Кирпа Вадим

30 ноября 2022 г.

**Посиделка N:** Свёрточные сетки

# Agenda

- Как видит компьютер
- Свёртка
- Пуллинг
- Data augmentation
- ImageNet
- Transfer Learning

# Затравка (2006)



Please click on all the images that show cats:

The grid contains 16 images arranged in four rows of four. The images alternate between dogs and cats. Each image has a blue 'adopt me' link below it. The images are as follows:

- Row 1: Black and white dog, orange cat, white and brown dog, orange cat.
- Row 2: White and brown dog, tan dog, person petting a black cat, grey cat.
- Row 3: White dog, tan dog, brown and white dog, black and white dog.
- Row 4: Yellow cat, brown and white dog, brown dog, black and white dog.

# Затравка (2006)

- Капча портит вид сайтов, довольно бесполезная, в Microsoft придумывают в 2006 году новый вид капчи. Надо отличать котов от собак.
- В то время разделение собак от кошек было очень сложной задачей, лучшая точность была 0.6.
- У нас есть 12 картинок, робот нагнёт нас с вероятностью  $0.6^{12}$ .
- Пул картинок пополнялся фотографиями из приютов.

# В 2014 проект закрыли



## Dogs vs. Cats

Create an algorithm to distinguish dogs from cats  
215 teams · 6 years ago

Overview Data Notebooks Discussion Leaderboard Rules

[Public Leaderboard](#) [Private Leaderboard](#)

The private leaderboard is calculated with approximately 70% of the test data. [Refresh](#)

This competition has completed. This leaderboard reflects the final standings.

■ Gold ■ Silver ■ Bronze

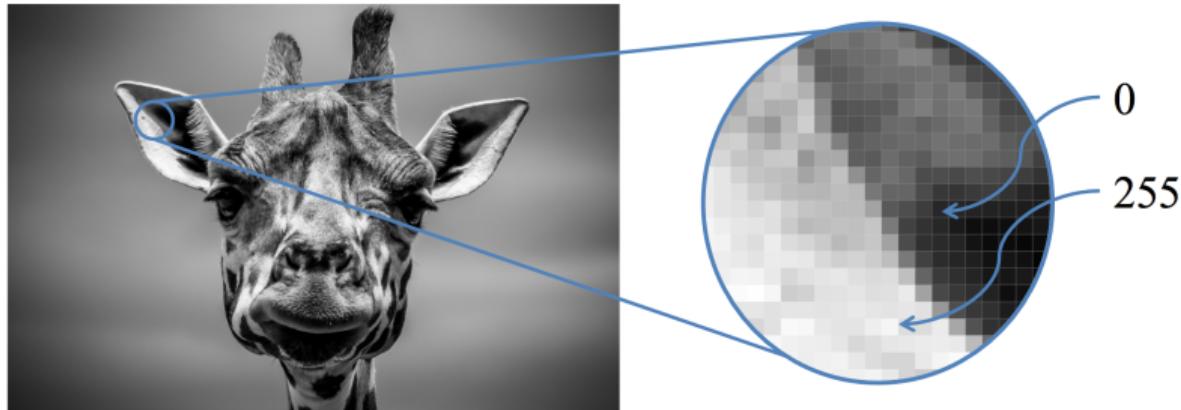
#	△pub	Team Name	Notebook	Team Members	Score ⓘ	Entries	Last
1	—	Pierre Sermanet			0.98914	5	6y
2	▲ 4	orchid			0.98308	17	6y
3	—	Owen			0.98171	15	6y
4	—	Paul Covington			0.98171	3	6y

# Как видит компьютер



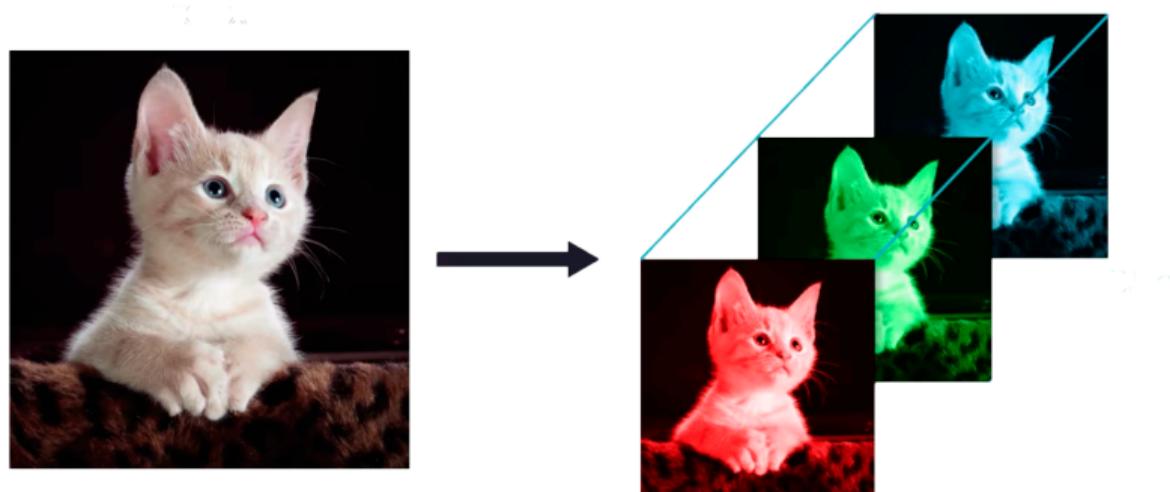
# Картина – тензор

- Каждая картинка – это матрица из пикселей
- Каждый пиксель обладает яркостью по шкале от 0 до 255

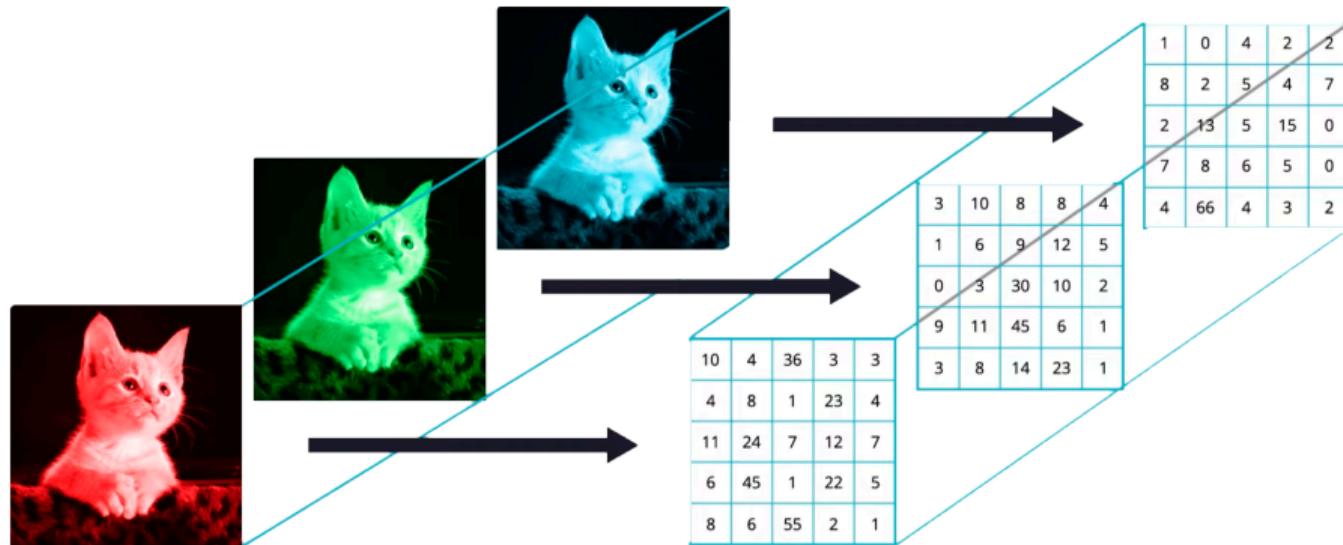


# Картина – тензор

- Цветное изображение имеет три канала пикселей: красный, зелёный и синий (rgb), размерность изображения  $5 \times 5 \times 3$



# Картина – тензор



# Картина – тензор

$5 \times 5 \times 3$



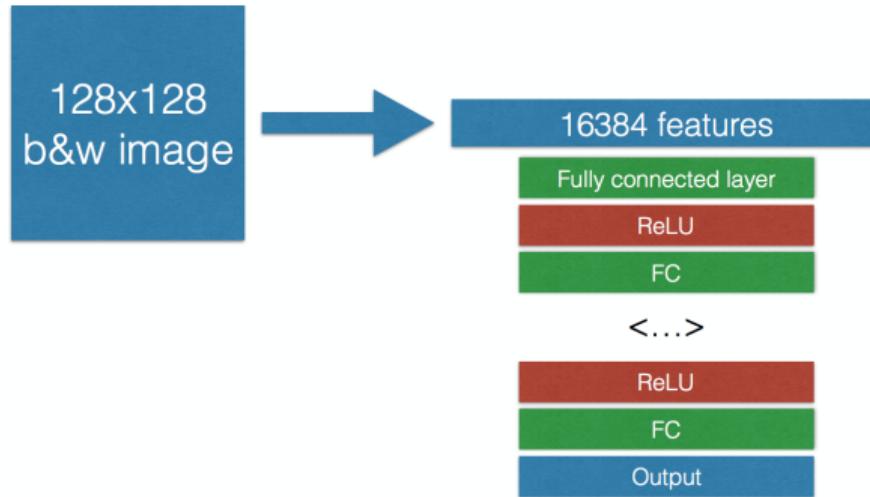
3D Array

7	0	4	2	2	2	3
3	50	3	3	3	3	3
107	2	36	3	3	3	3
4	8	1	23	4	4	4
11	24	7	12	7	7	7
6	45	1	22	5	5	5
8	6	55	2	1	1	1

5      R      G      B

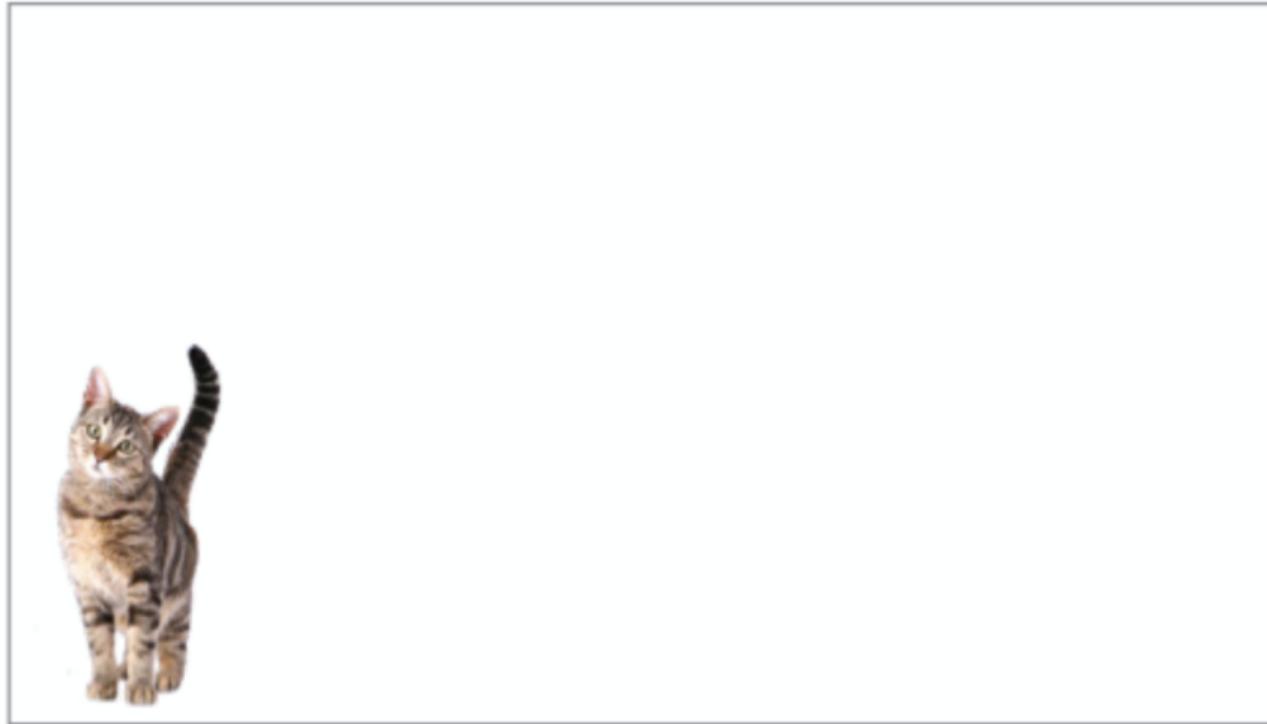
5

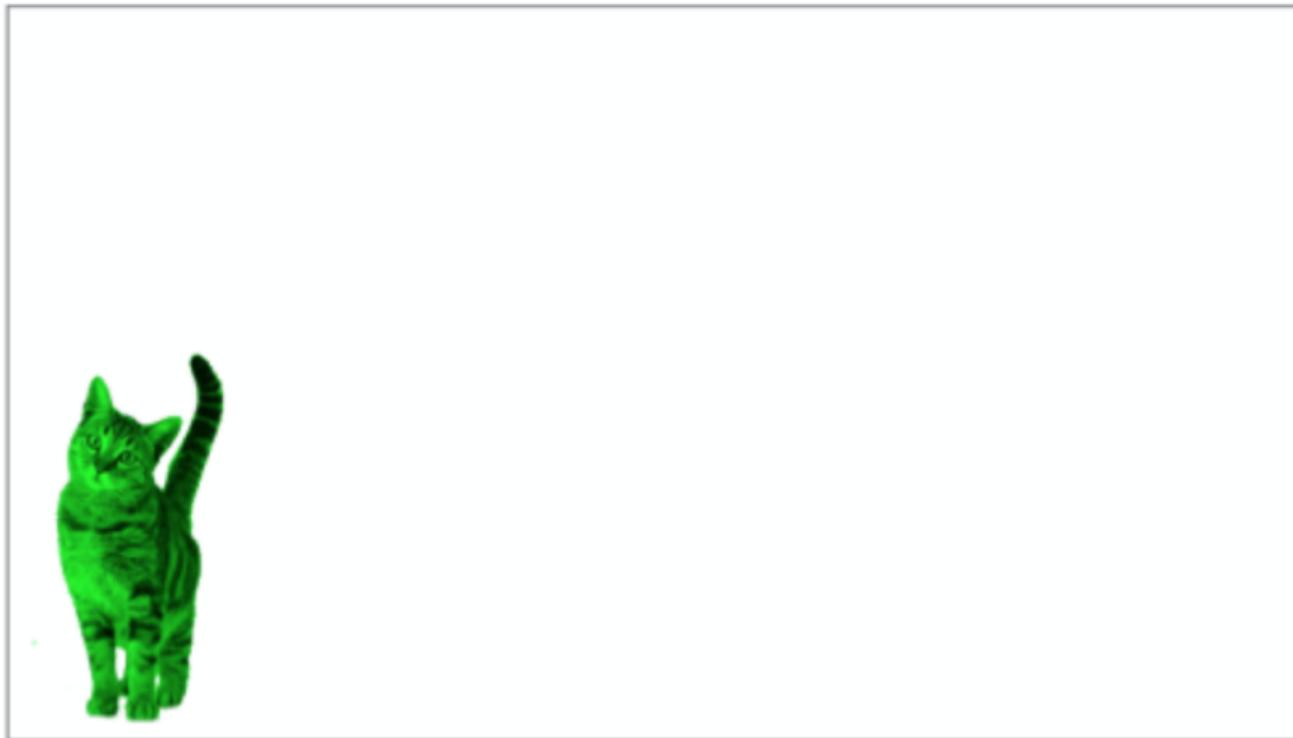
# Обычная сетка



- Очень много весов
- Теряется информация о взаимном расположении пикселей
- Изображение в разных местах картинки даёт разные веса







# Обычная сетка

- Хотим, чтобы информация не терялась
- Хотим, чтобы сетка была устойчива к положению картинки
- Хотим одинаковые веса  $\Rightarrow$  свёртка

# Свёртка



# Свёртка

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

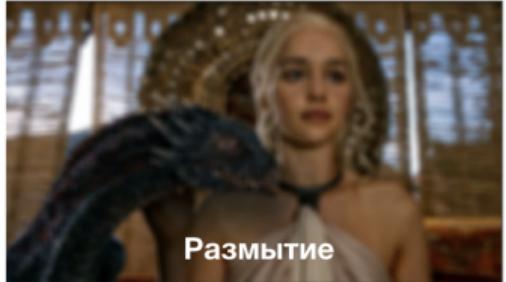
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

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

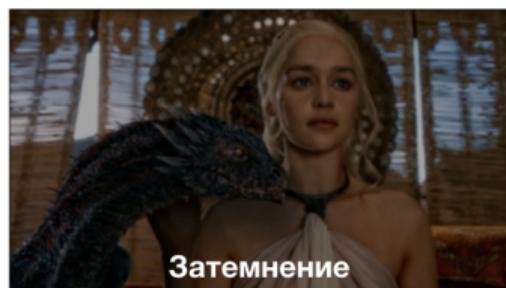
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



$$\frac{1}{9} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



$$\begin{pmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{pmatrix}$$



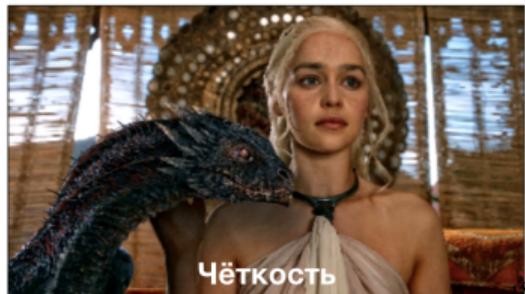
```
[[62, 36, 9], [[62, 36, 9], [[62, 36, 9],  
[62, 36, 9], [61, 35, 8], [60, 34, 7],  
[61, 35, 8], [59, 33, 6], [57, 31, 4],  
..., ..., ...,  
[58, 43, 20], [53, 41, 17], [50, 38, 14],  
[57, 45, 21], [53, 41, 17], [49, 37, 13],  
[57, 45, 21]] [52, 40, 16]] [48, 36, 12]]
```

Красный

Зеленый

Синий

$$\begin{pmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{pmatrix}$$



# Свёртка

- Разные ядра помогают накладывать на кртинку различные эффекты
- Какие-то ядра помогают искать границы
- **Идея:** возможно, с помощью некоторых ядер можно искать разные образы...



## Классификатор слэшей

The diagram illustrates a convolution operation. The input image is a 4x4 grid with values [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0], and [0, 0, 0, 1]. A 2x2 kernel with values [1, 0], [0, 1] is applied to the input. The result is an output image of size 2x2 with values [0, 0], [0, 1], and [0, 0, 2]. The highlighted regions in red and green indicate the receptive field and the kernel's movement across the input.

The diagram shows the convolution operation between an input matrix and a kernel matrix to produce an output matrix.

**Input:**

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

**Kernel:**

1	0
0	1

**Output:**

0	0	0
0	0	1
0	1	0

The result is obtained by applying the kernel to the input. The highlighted regions in red and green indicate the receptive field of each output unit. The final output value is 1 at position (2,2) because it is the sum of the products of the corresponding kernel values and input values:  $(0 \cdot 1) + (0 \cdot 0) + (0 \cdot 0) + (1 \cdot 1) = 1$ .

# Классификатор слэшей

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

Input

\*

1	0
0	1

Kernel

=

0	0	0
0	1	0
0	0	2

Output

Max = 2



Simple  
classifier



Max = 1

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

Input

\*

1	0
0	1

Kernel

=

0	0	0
0	0	1
0	1	0

Output

# Свёртка инвариантна к расположению

$$\begin{array}{|c|c|c|c|} \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 2 \\ \hline \end{array}$$

Input                              Kernel                              Output

$$\begin{array}{|c|c|c|c|} \hline 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 2 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

Input                              Kernel                              Output

# Свёртка инвариантна к расположению

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

Input

\*

1	0
0	1

=

0	0	0
0	1	0
0	0	2

Output

Max = 2

↑  
Didn't  
change  
↓

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

Input

\*

1	0
0	1

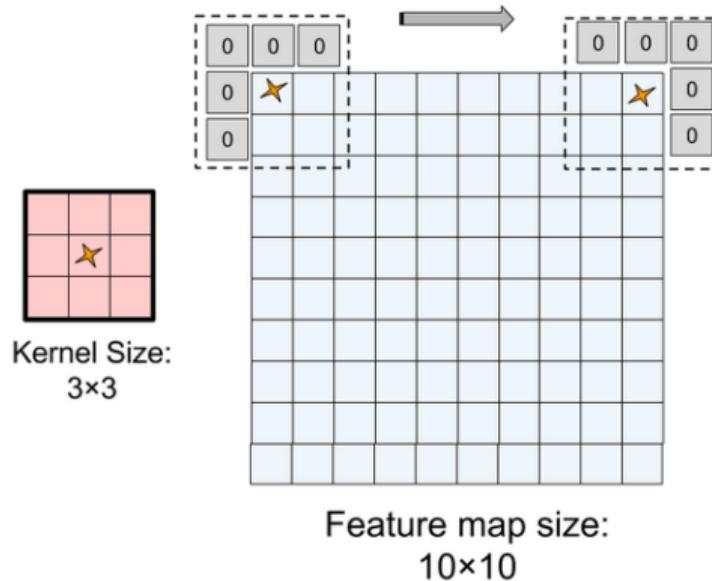
=

2	0	0
0	1	0
0	0	0

Output

Max = 2

# Дополнение (Padding)



**Padding** используют, чтобы пространственная размерность картинки не уменьшалась после применения свёртки, это помогает не терять информацию на краях и избежать проблем с размерностями

# Дополнение (Padding)

$0_2$	$0_0$	$0_1$	0	0	0	0
$0_1$	$2_0$	$2_0$	3	3	3	0
$0_0$	$0_1$	$1_1$	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

# Свёрточный слой

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

Input 3x3  
image with  
zero **padding**  
(grey area)

Shared bias:  
 $b$

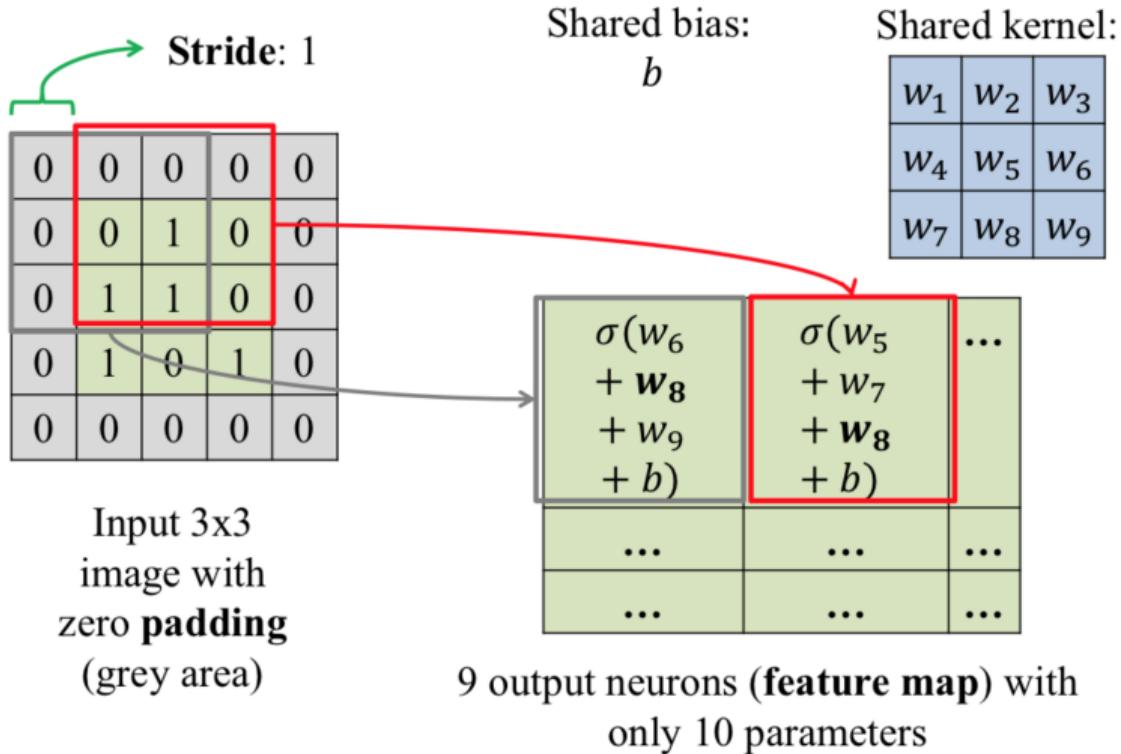
Shared kernel:

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

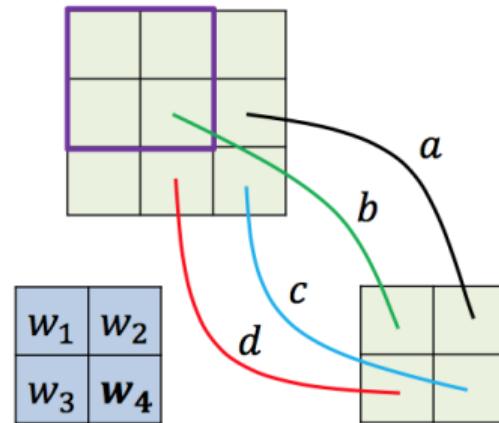
$\sigma(w_6 + w_8 + w_9 + b)$	...	...
...	...	...
...	...	...

9 output neurons (**feature map**) with  
only 10 parameters

# Свёрточный слой



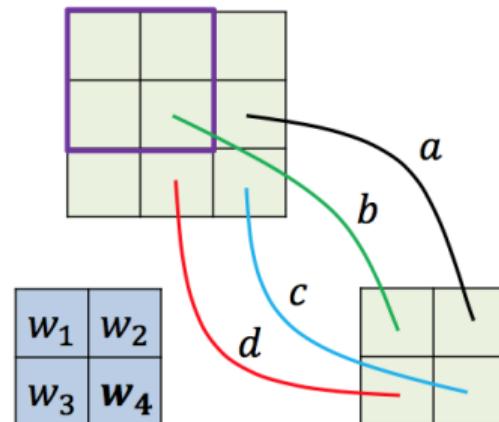
# Backpropagation для свёрточного слоя



$$b = w_1 x_{11} + w_2 x_{12} + w_3 x_{21} + w_4 x_{22}$$

$$a = w_1 x_{12} + w_2 x_{13} + w_3 x_{22} + w_4 x_{23}$$

# Backpropagation для свёрточного слоя

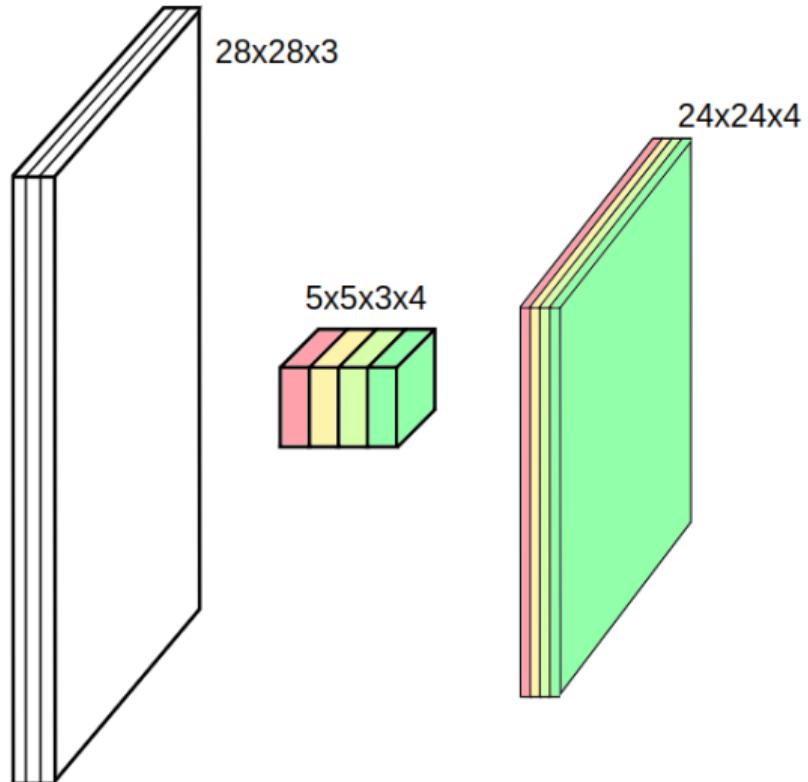


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \cdot x_{11} + \frac{\partial L}{\partial b} \cdot x_{12} + \frac{\partial L}{\partial c} \cdot x_{22} + \frac{\partial L}{\partial d} \cdot x_{12}$$

# Свёрточный слой

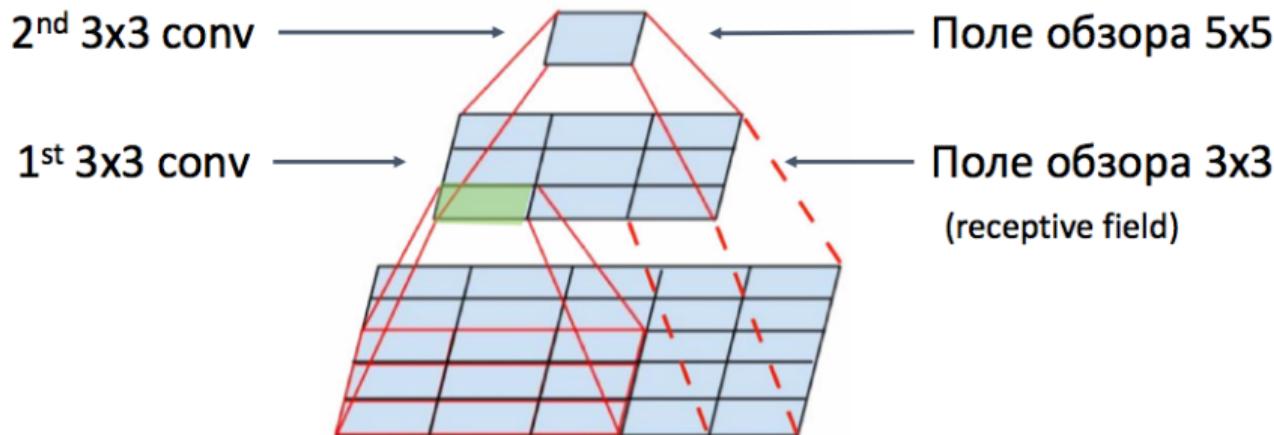
- Слой действует одинаково для каждого участка картинки, в отличие от полносвязного
- Нужно оценивать меньшее количество параметров
- Слой учитывает взаимное расположение пикселей
- Можно учить тем же самым backpropagation
- Свёрточный слой - это полносвязный слой с ограничениями, попробуйте нарисовать свёрточный слой также как мы рисовали полносвязный

# Одного ядра недостаточно



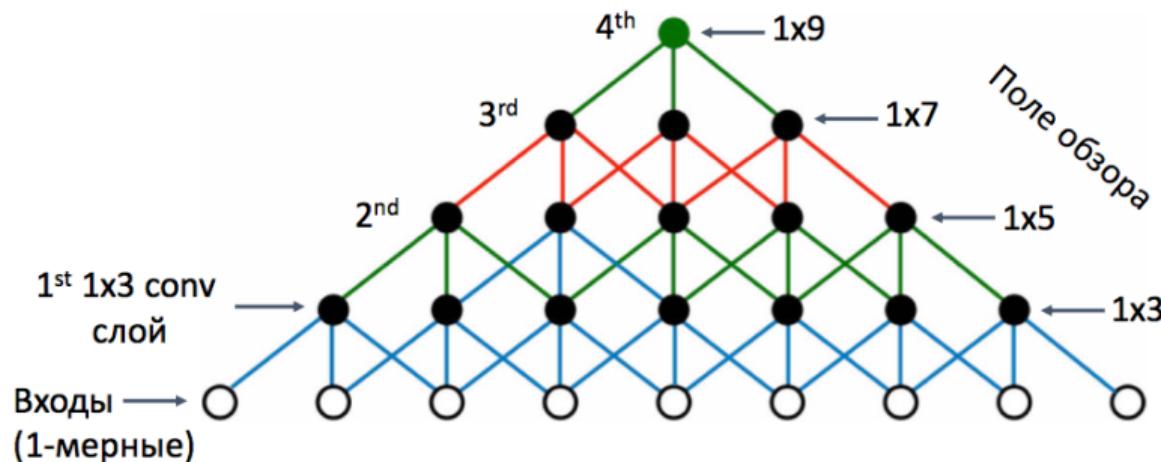
# Одного свёрточного слоя недостаточно

- Нейроны первого слоя смотрят на поле  $3 \times 3$
- Если интересующий нас объект больше, нам нужна вторая свёртка



# Одного свёрточного слоя недостаточно

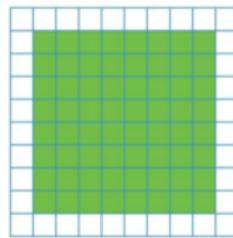
- $N$  слоёв со свёртками  $3 \times 3$
- На  $N$ -ом слое поле обзора  $(2N + 1) \times (2N + 1)$
- Если наш объект размера 300, надо 150 слоёв...  $\Rightarrow$  нужно растить поле обзора быстрее



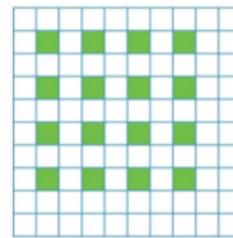
# Сдвиги и Пулинг

# Нужно растить поле обзора быстрее!

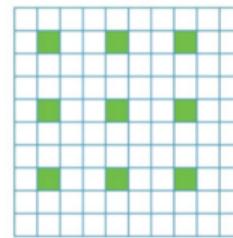
- Пиксели локально скоррелированы — соседние пиксели, как правило, не сильно отличаются друг от друга
- Если будем делать свёртку с каким-то шагом, сэкономим мощности компьютера и не потеряем в информации



Stride = 1



Stride = 2

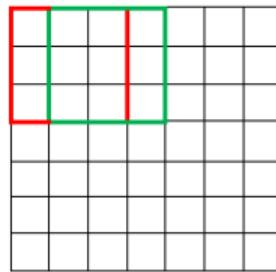


Stride = 3

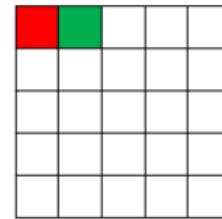
- Очень агрессивная стратегия снижения размерности картинки

# Сдвиг (Stride)

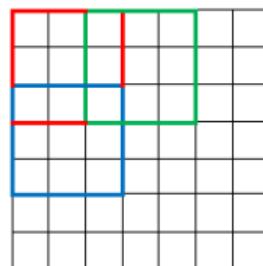
7 x 7 Input Volume



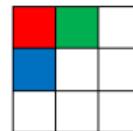
5 x 5 Output Volume



7 x 7 Input Volume

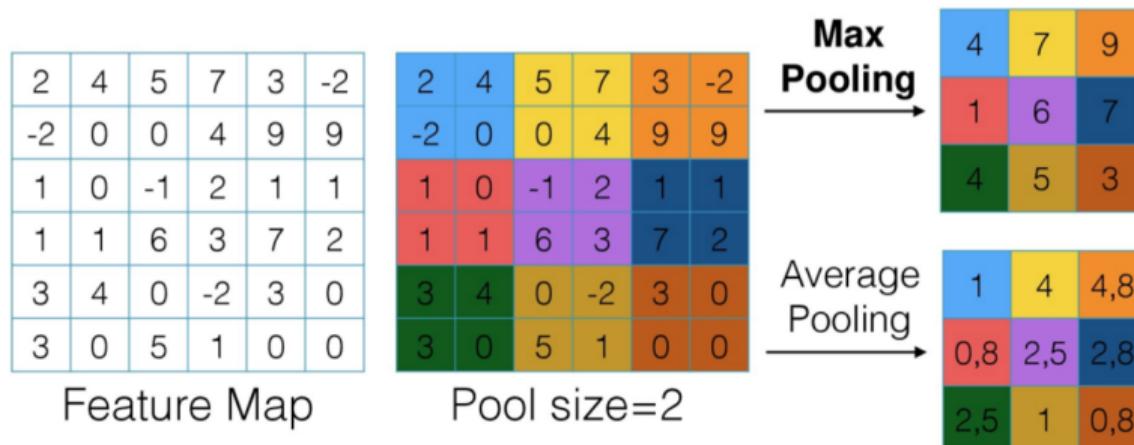


3 x 3 Output Volume



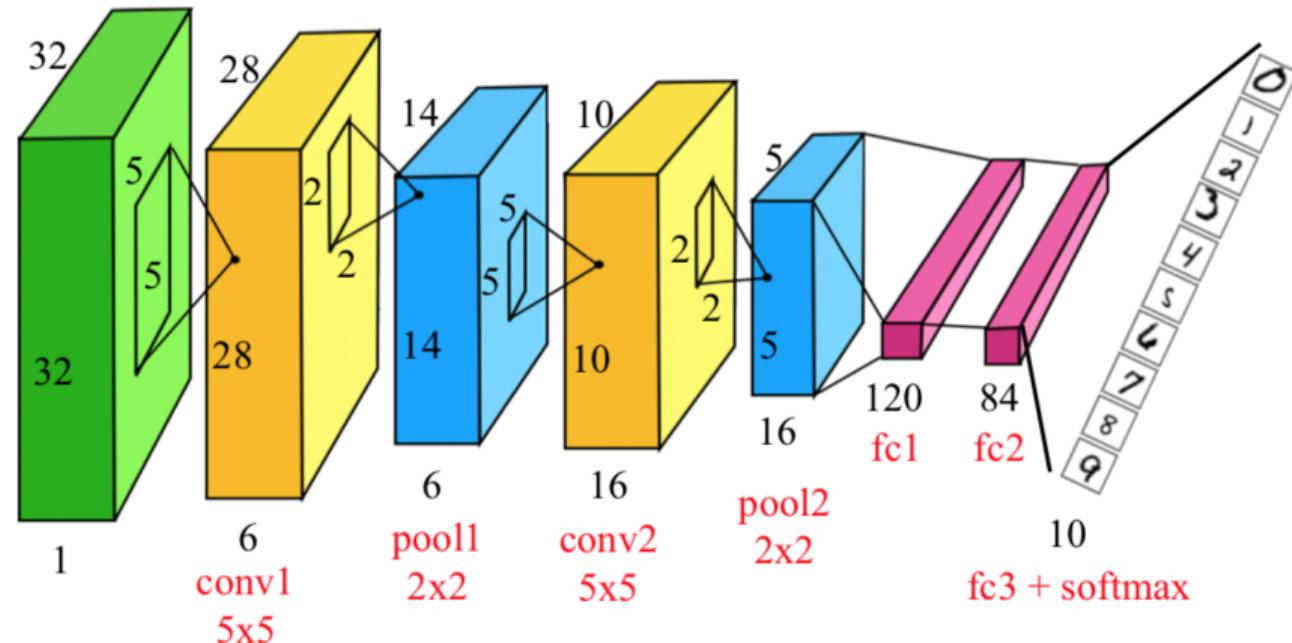
# Пулинг слой (Pooling)

- Будем считать внутри какого-то окна максимум или среднее и сворачивать размерность, пользуясь локальной коррелированностью

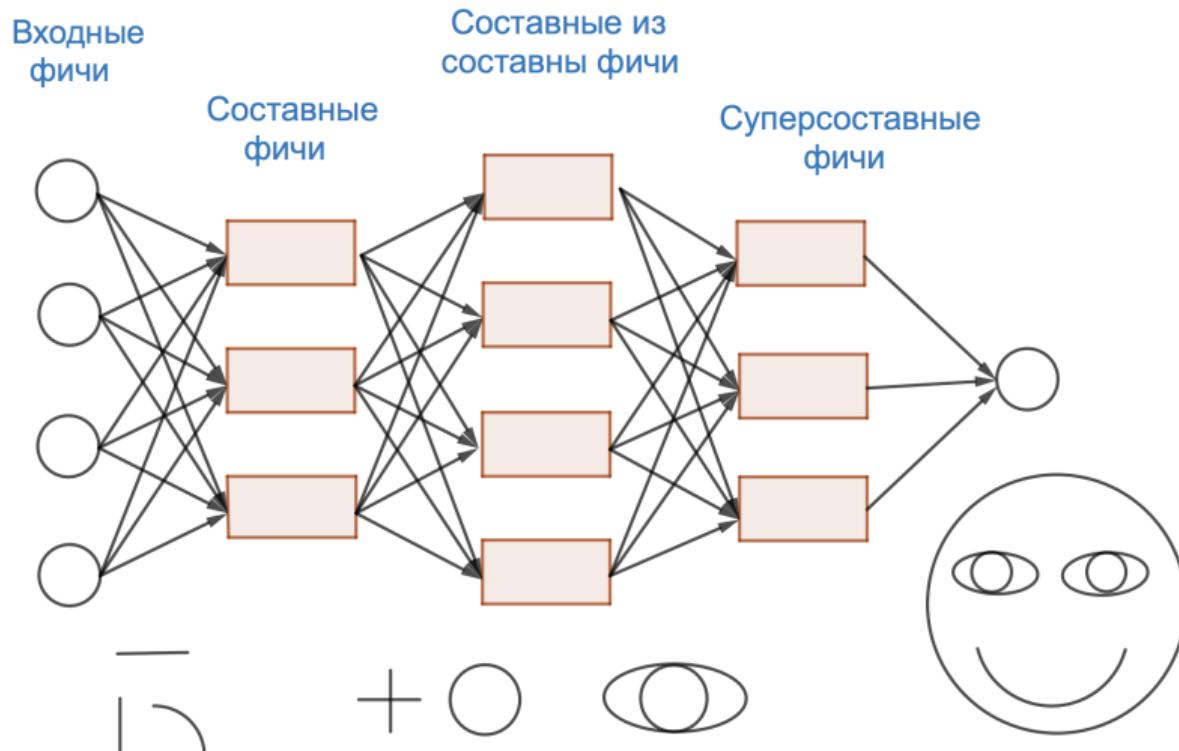


# Простейшая CNN

Нейросеть LeNet-5 (1998) для распознавания рукописных цифр



# Что выучивают нейросети



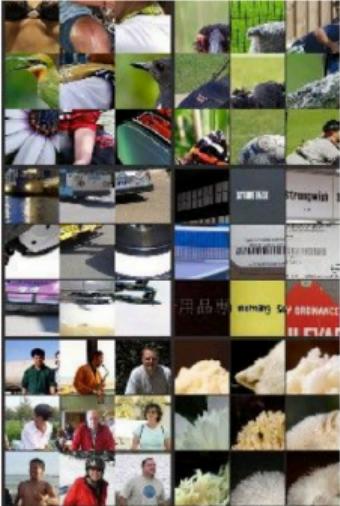
# Что выучивают нейросети



*Layer 1*



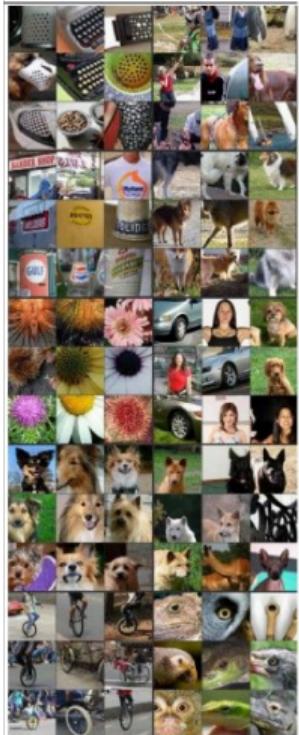
*Layer 2*



*Layer 3*



*Layer 4*



*Layer 5*

<https://arxiv.org/pdf/1311.2901.pdf>

# Data augmentation

# Data augmentation

- В сетке может быть миллионы параметров!
- Естественная регуляризация, дополнительная регуляризация, генерация новых данных (data augmentation)
- Генерируем новые цвета, сдвигаем, искажаем и тп



<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

# Data augmentation

- Сдвиги (вместо них лучше пулинг)
- Увеличение, уменьшение
- Повороты
- Искажение
- Затенение
- Смена стиля (красок)

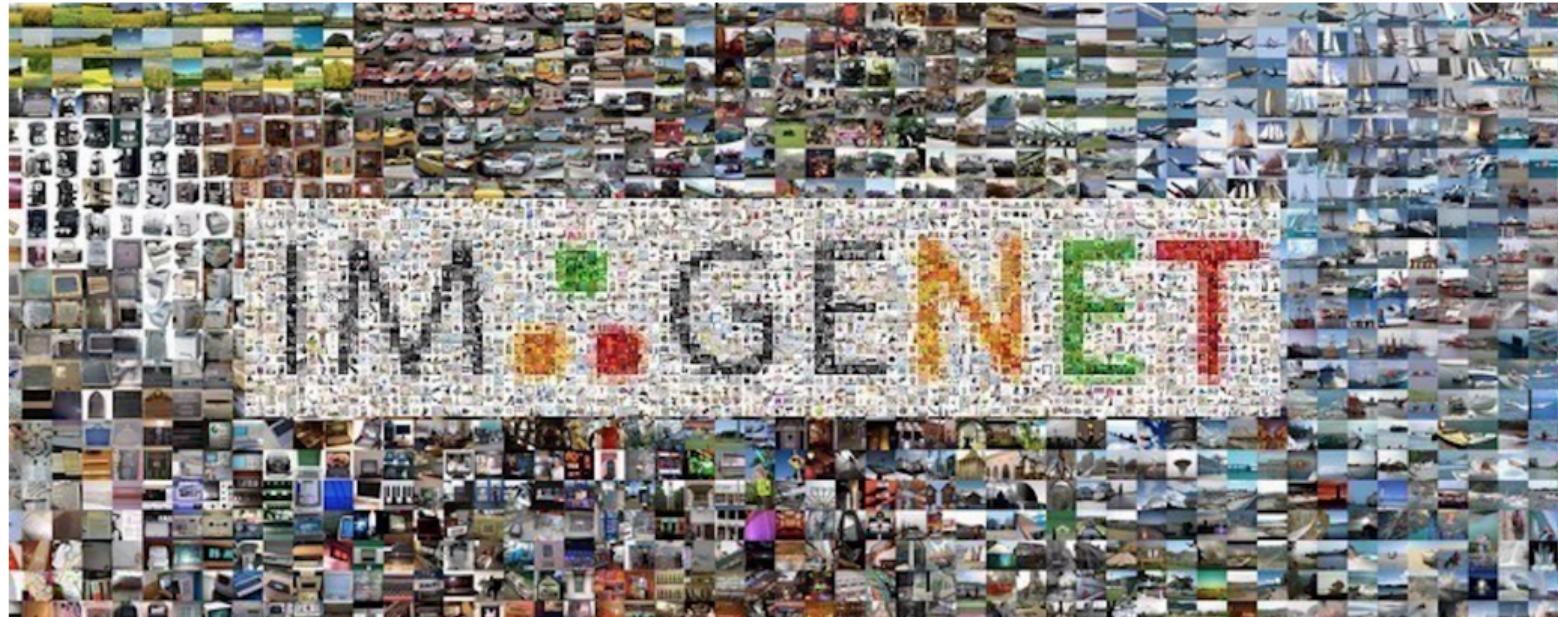


Делает модель более устойчивой, полезна при маленьких выборках. На больших датасетах также улучшает результаты.

# Сказ о том, как люди ImageNet рвали

# ImageNet

- около 10 миллионов размеченных изображений из интернета



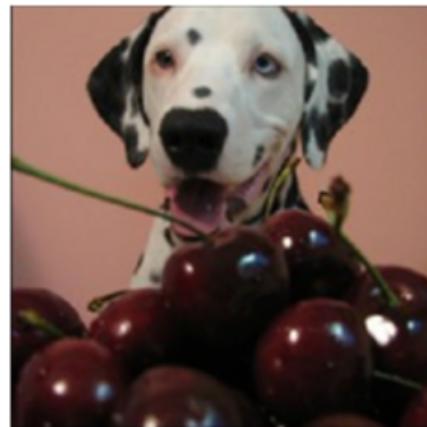
<http://www.image-net.org/>

# ImageNet

- выборка очень большая и неоднородная, постоянно пополняется, соревнования на ней проводятся каждый год
- обычно изображение требуется отнести к одному из 1000 классов, можно давать несколько ответов
- если один из пяти вариантов оказался верным, то классификация считается верной
- до 2012 года лучшие алгоритмы дают ошибку в 25%
- в 2012 году на арену выходят глубокие нейронные сети

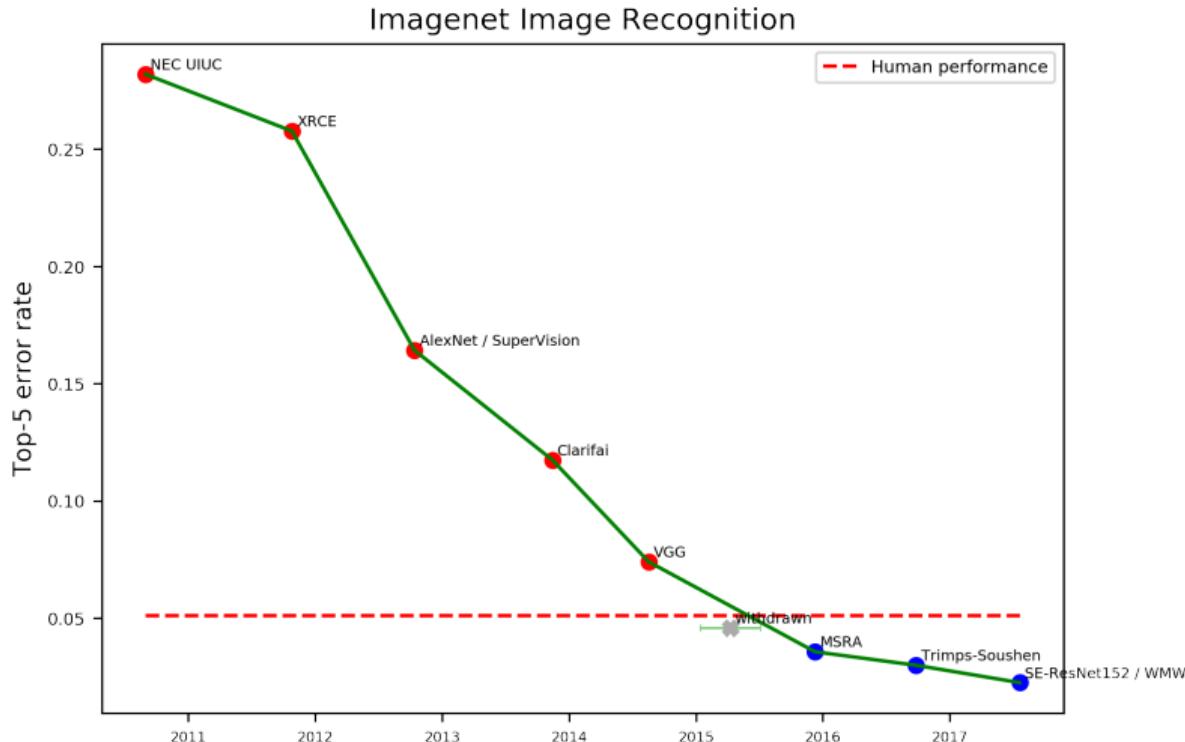
# ImageNet

- бывают спорные изображения: тут вишня, если распознать как далматинец, будет неправильно



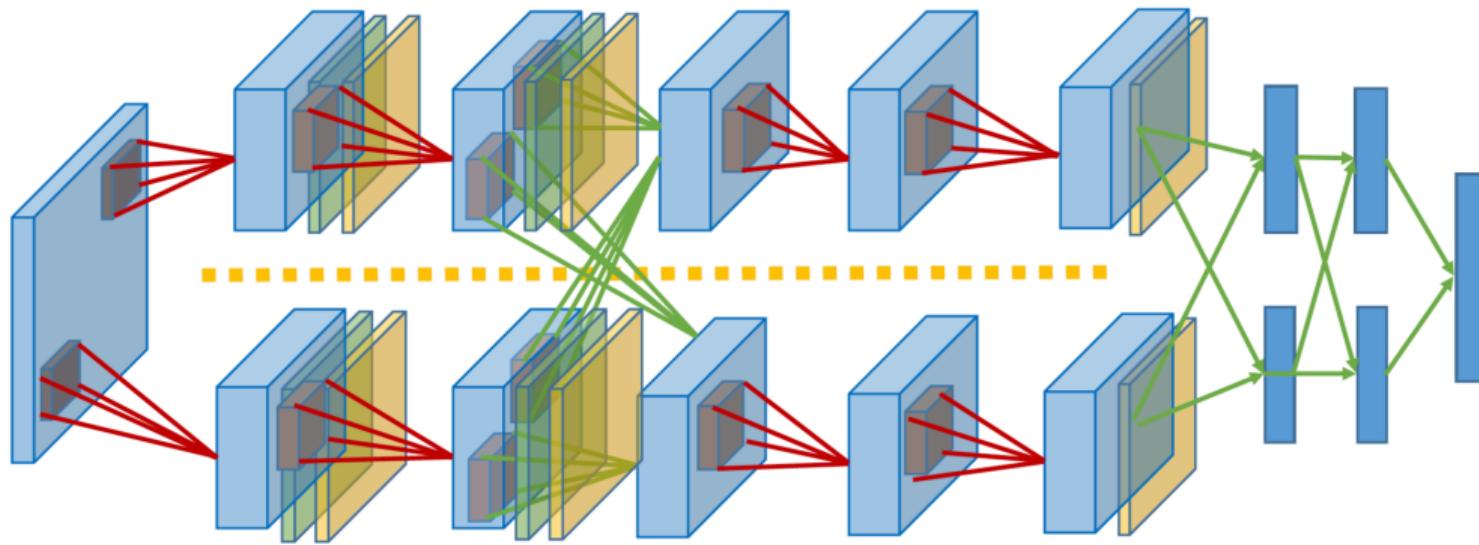
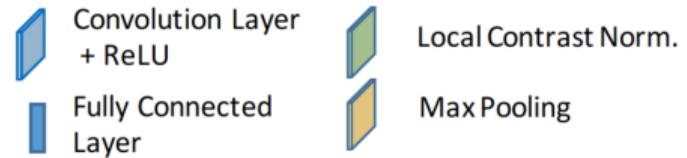
- Можно попробовать сразиться с компьютером:  
<https://cs.stanford.edu/people/karpathy/ilsvrc/>

# Точность сетей на ImageNet



<https://www.eff.org/ai/metrics>

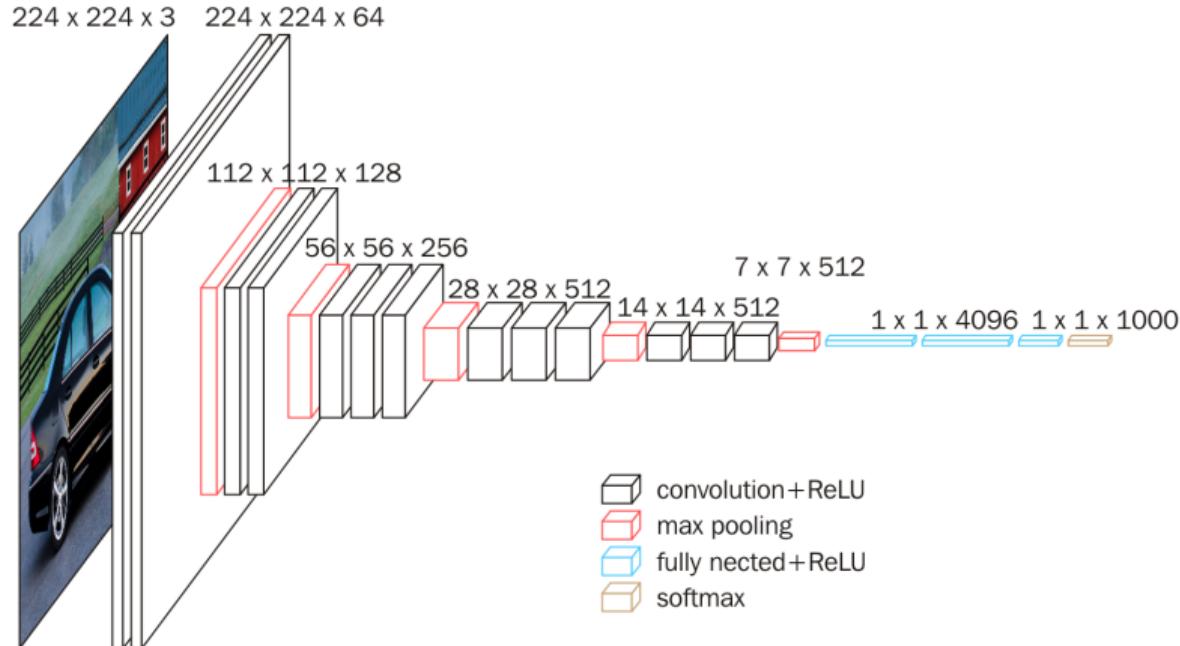
# AlexNet (2012)



## AlexNet (2012)

- Тот же LeNet из 1998г. но увеличен в 1000 раз и дополненный трюками (Dropout, ReLU, Data augmentation)
- Свёртки  $11 \times 11$ ,  $5 \times 5$ ,  $3 \times 3$
- Уронила ошибку с 25% до 15.4%
- 60 миллионов параметров
- Училась 6 дней на 2 GPU
- Попробовали сделать ансамблю из таких сеток, ошибка упала до 11.7%

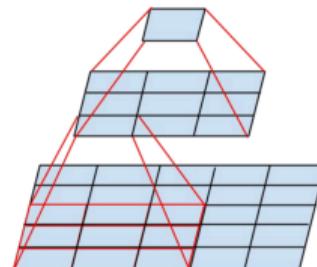
# VGG (2014)



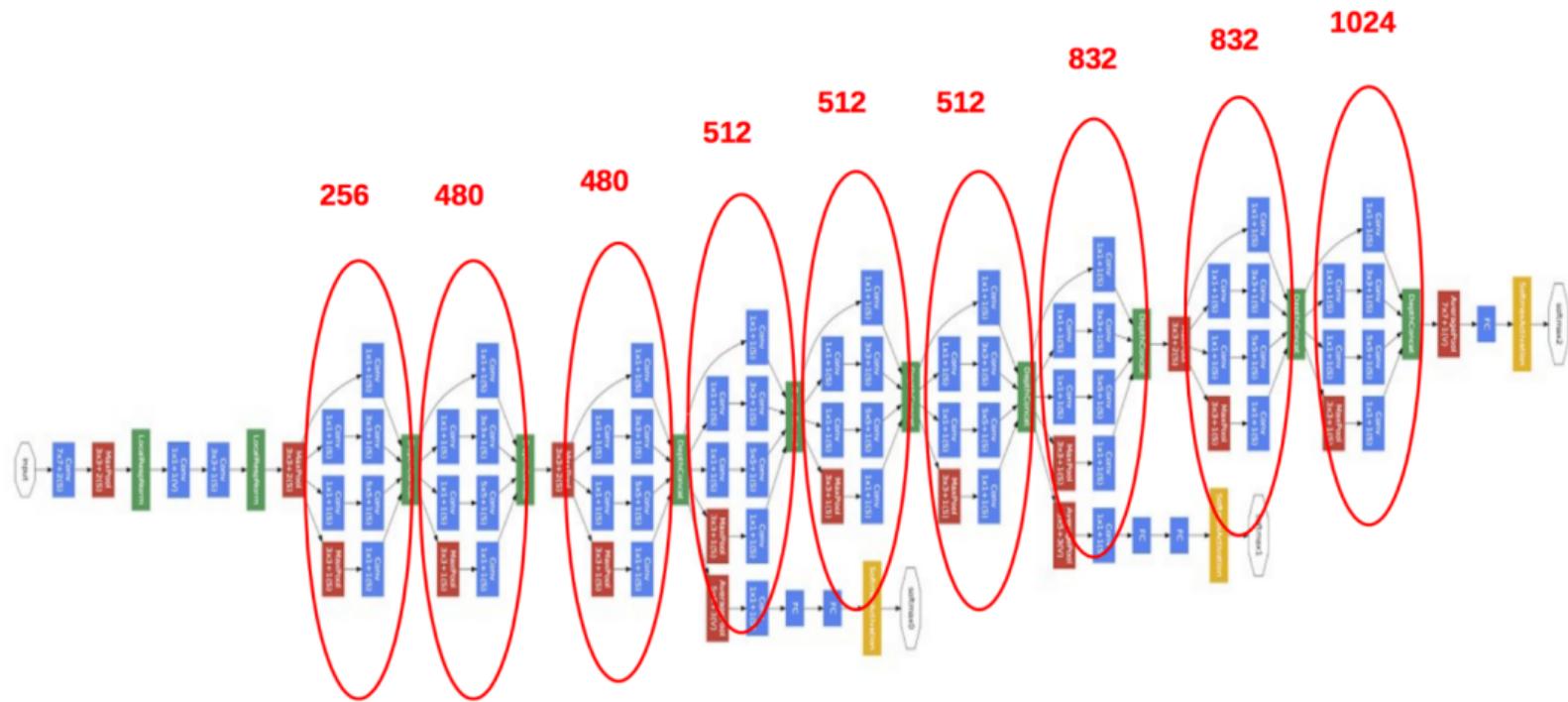
[https://www.datalearner.com/paper\\_note/content/300035](https://www.datalearner.com/paper_note/content/300035)

# VGG (2014)

- 138 миллионов параметров
- Училась 2 – 3 недели на 4 GPU
- Ошибка упала до 6.8%
- Свёртки только  $3 \times 3$ , но намного больше фильтров для экономии параметров (две свёртки  $3 \times 3$  покрывают поле  $5 \times 5$  и требуют 18 параметров, а свёртка  $5 \times 5$  требует 25 параметров)

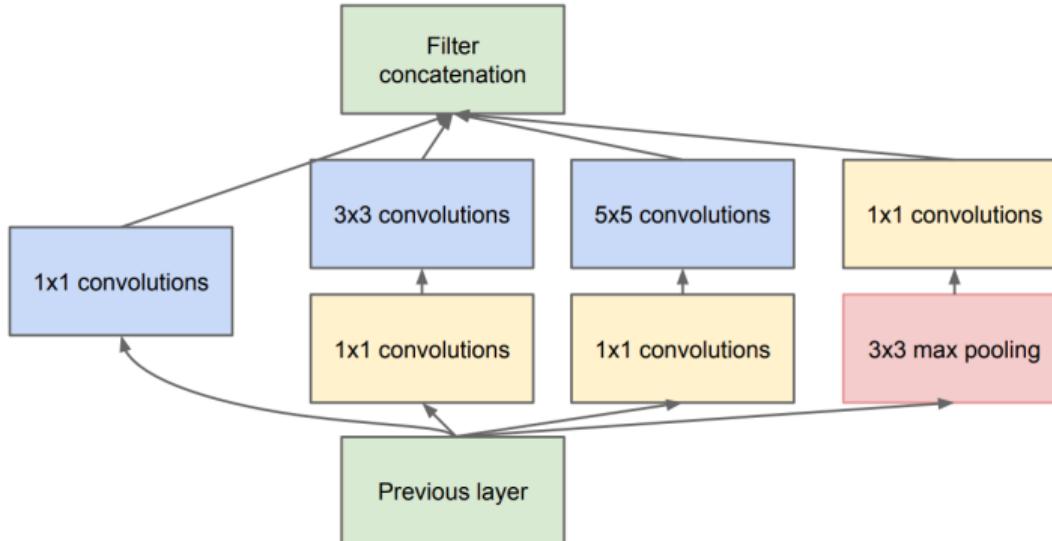


# GoogleLeNet aka Inception V1 (2014)



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

# GoogleLeNet aka Inception V1



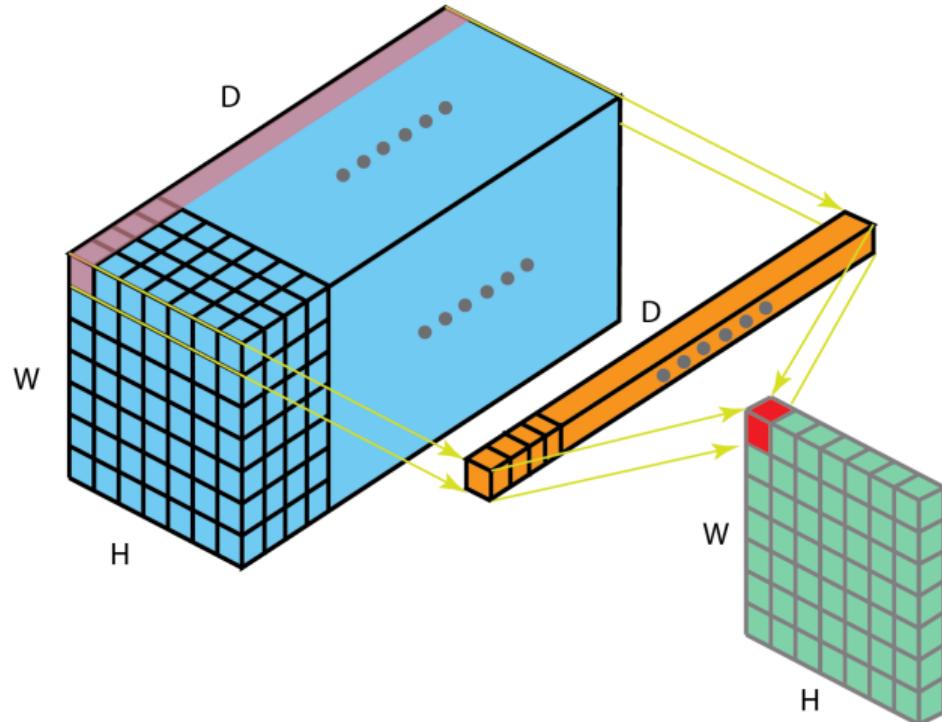
(b) Inception module with dimension reductions

# GoogleLeNet aka Inception V1 (2014)

- Парни из Google решили усовершенствовать AlexNet.
- Уменьшаем свёртки, саму **сетку собираем из компонент** (всего 9 блоков).
- На каждом слое используется не одна свёртка, а несколько разных, что помогает реагировать на сигналы разного масштаба и улучшает работу, используются **свёртки  $1 \times 1$** .
- **Несколько дополнительных классификаторов на разных уровнях.** Идея в том, что такие классификаторы позволяют «протолкнуть» градиенты к ранним слоям и тем самым уменьшить эффект затухания градиента.
- Параметров в 10 раз меньше, чем в AlexNet, работает лучше, 6.7% ошибок

# Свёртка $1 \times 1$

- Позволяет сократить размерность по числу каналов
- Представляет из себя полносвязный слой по фильтрам
- В Inception мы делаем много разных свёрток, а потом свёрткой  $1 \times 1$  выбираем из них лучшее, агрессивно понижая размерность



# GoogleLeNet aka Inception V1

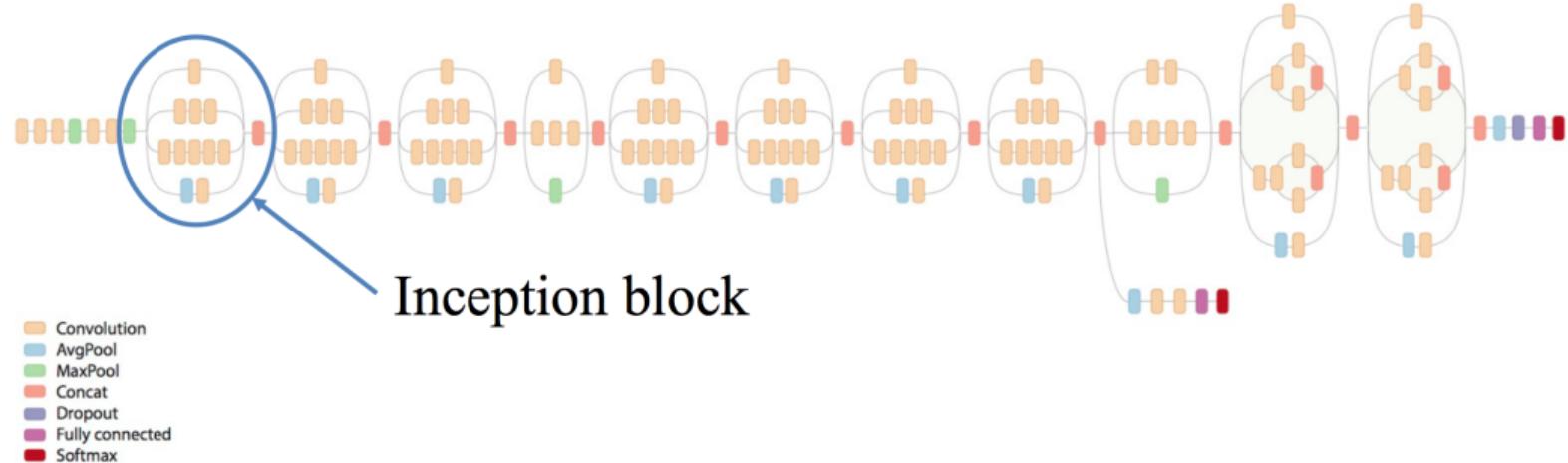
## 10 Acknowledgements

We would like to thank Sanjeev Arora and Aditya Bhaskara for fruitful discussions on [2]. Also we are indebted to the DistBelief [4] team for their support especially to Rajat Monga, Jon Shlens, Alex Krizhevsky, Jeff Dean, Ilya Sutskever and Andrea Frome. We would also like to thank to Tom Duerig and Ning Ye for their help on photometric distortions. Also our work would not have been possible without the support of Chuck Rosenberg and Hartwig Adam.

## References

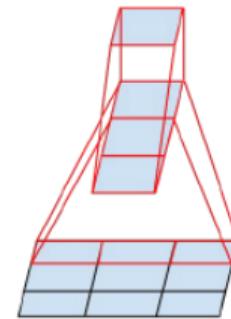
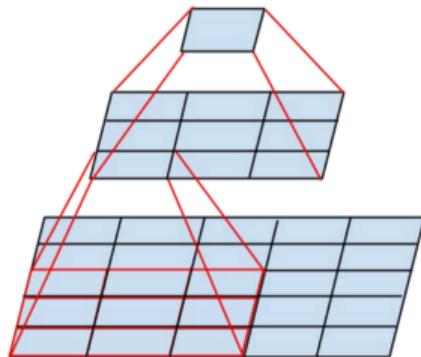
- [1] Know your meme: We need to go deeper. <http://knowyourmeme.com/memes/we-need-to-go-deeper>. Accessed: 2014-09-15.
- [2] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. *CoRR*, abs/1310.6343, 2013.
- [3] Ümit V. Çatalyürek, Cevdet Aykanat, and Bora Uçar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM J. Sci. Comput.*, 32(2):656–683, February 2010.

# Inception V3 (2015)



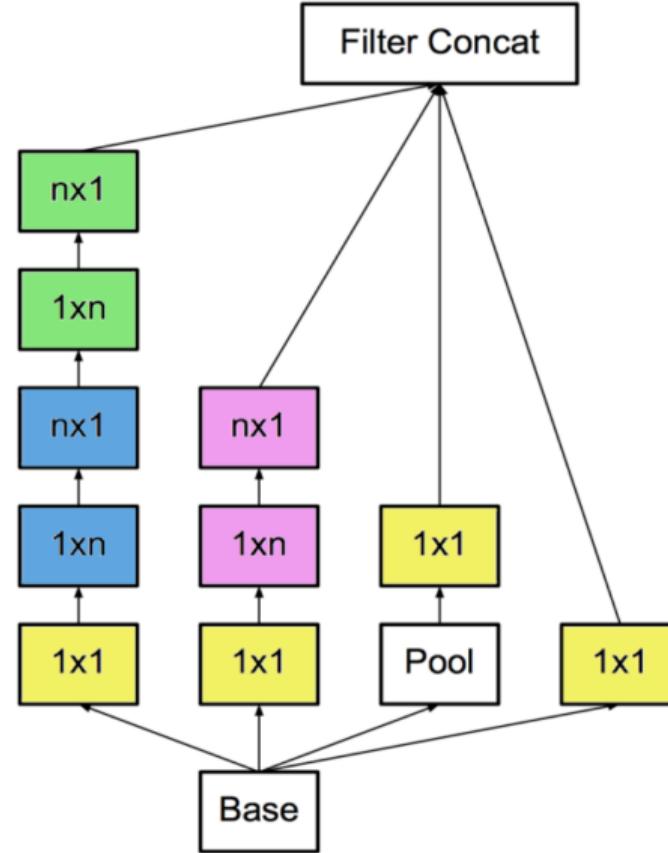
<https://arxiv.org/abs/1512.00567>

## Свёртка $n \times 1$ и $1 \times n$



- В VGG заменяли большие свёртки на последовательные  $3 \times 3$  для экономии
- Пойдём дальше и заменим их на последовательные  $3 \times 1$  и  $1 \times 3$
- Экономим ещё больше параметров, для каждой свёртки 6 вместо 9

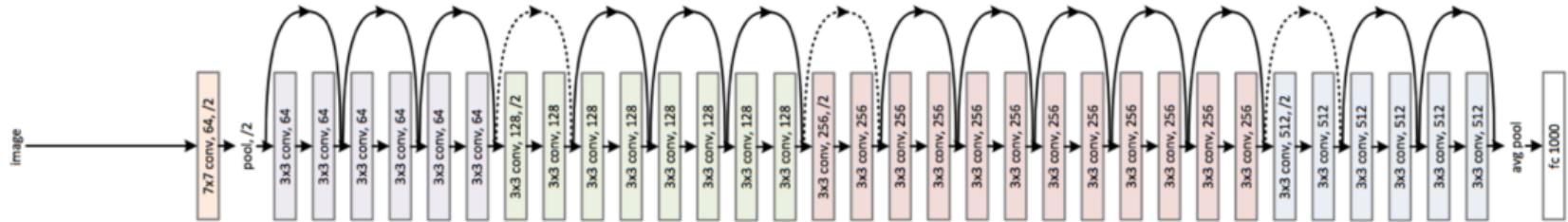
# Inception V3 (2015)



# Inception V3 (2015)

- Сетка собирается из 11 inception layers, добавили BatchNorm (то же самое без него Inception V2)
- В результате исследований сформулировали принципы обучения глубоких свёрточных сетей:
  1. Избегайте representation bottlenecks, нельзя резко снижать размерность слоя, это надо делать плавно: от начала к концу
  2. Свёртку нужно разбивать на более мелкие части для экономии ресурсов и увеличения размера сетки
  3. Нельзя резко увеличивать глубину, забивая на ширину, надо растить сбалансированно
- Итоговое качество 4.2% ошибок, ансамбль из 4-х моделей дал 3.8%.

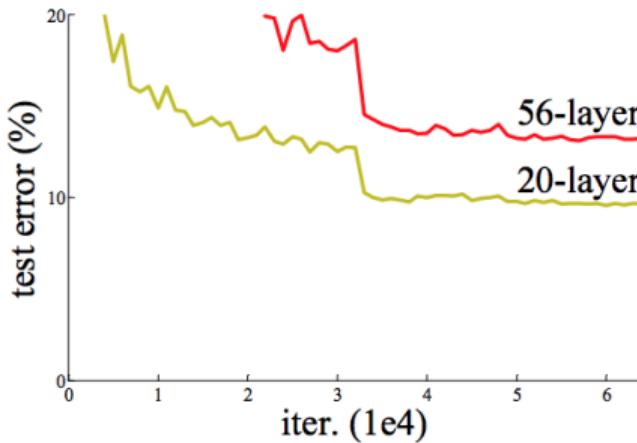
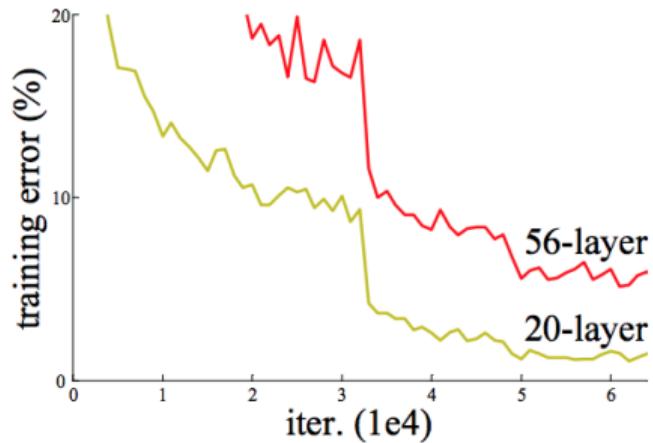
# ResNet (Microsoft) (2015)



- 152 слоя, ошибка составила 3.75%, ансабль из сеток дал 3.57%

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

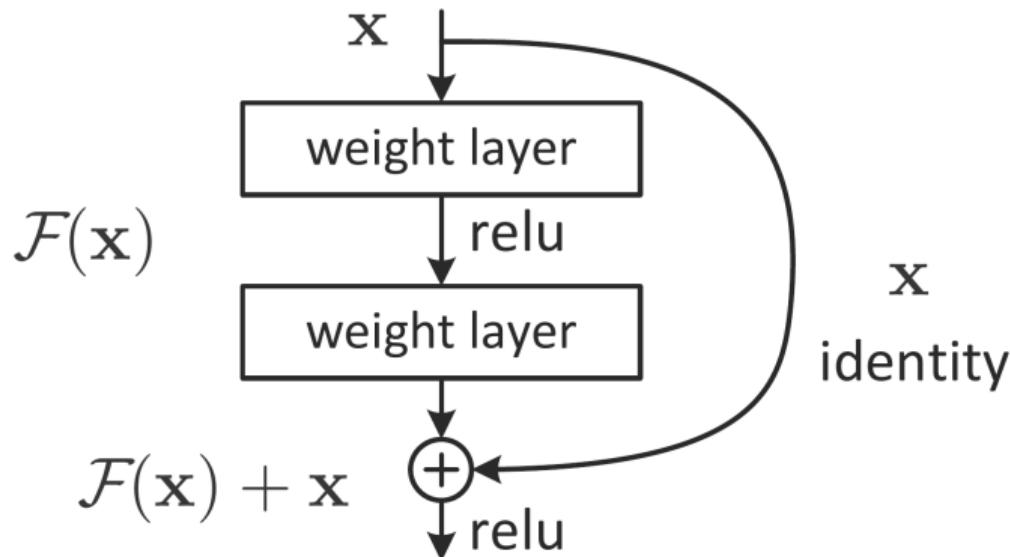
# Завязочка



## Завязочка

- VGG из 20 слоёв и VGG из 56 слоёв, большая сеть обучается хуже
- **Проблема:** слои инициализированы шумом, если какой-то один слой не натренирован, он убивает работу сети, через него не проходит полезный сигнал
- Чем больше слоёв, тем более ярко выражен этот эффект
- **Решение:** Будем посыпать вход на выход и давать слою возможность немного его подправить (residual слой)
- Идея чем-то похожа на бустинг, сеть сама решает когда заканчивать подправлять выходы (грубо говоря, сама выбирает глубину)

# ResNet (Microsoft) (2015)



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

# ResNet (Microsoft) (2015)

- **Идея:** более глубокие уровни должны улавливать разницу между новым и тем, что было раньше
- Ключевым элементом архитектуры является связь, которая пропускает несколько слоёв, передавая результат предыдущего слоя
- Такое изменение позволило полностью отказаться от таких техник регуляризации, как DropOut
- Градиенты не взрываются, свойства ResNet активно пытаются сейчас изучать

# Что было дальше?

- В 2016 году Google попробовал Inception-Resnet и поставил ансамблем новый рекорд, 3.08%

<https://arxiv.org/abs/1602.07261>

- После Google задумались о переносе своих огромных архитектур в мобильные телефоны и стали работать над их компактностью

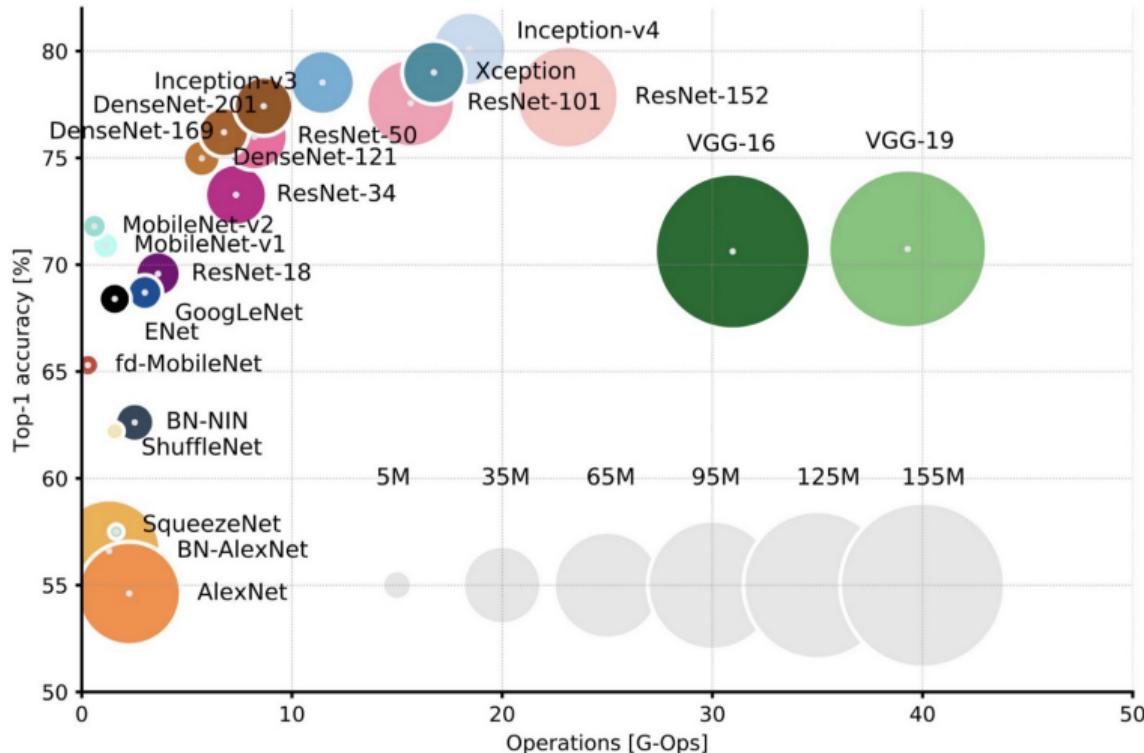
<https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>

- В целом область развивается очень динамично и всплывает куча идей, на хабре иногда есть рубрика "Читаем статьи за вас", там можно посмотреть сколько идей возникло только за осень 2017

<https://habr.com/ru/company/ods/blog/343822/>

Слайд из первой презентации :)

## # 2. Сложность сетей растёт



<https://towardsdatascience.com/neural-network-architectures-156e5bad51ba>

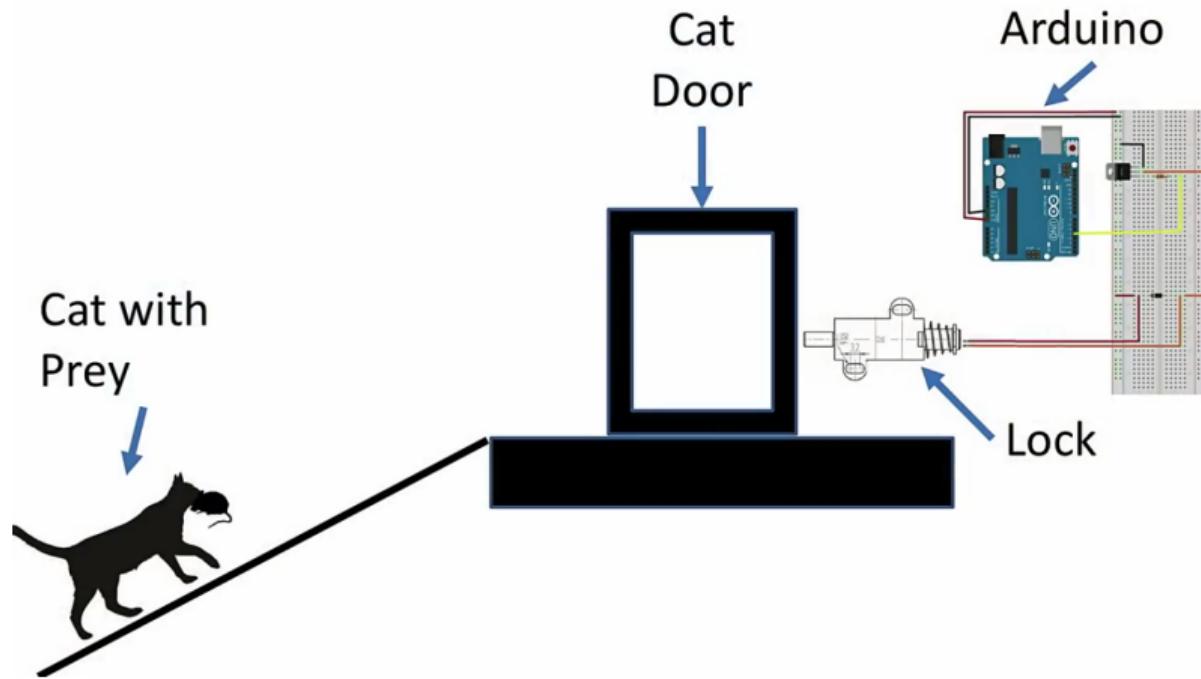
# История про Метрику

# Метрика это кот



<https://www.youtube.com/watch?v=1A-Nf3QIJjM>





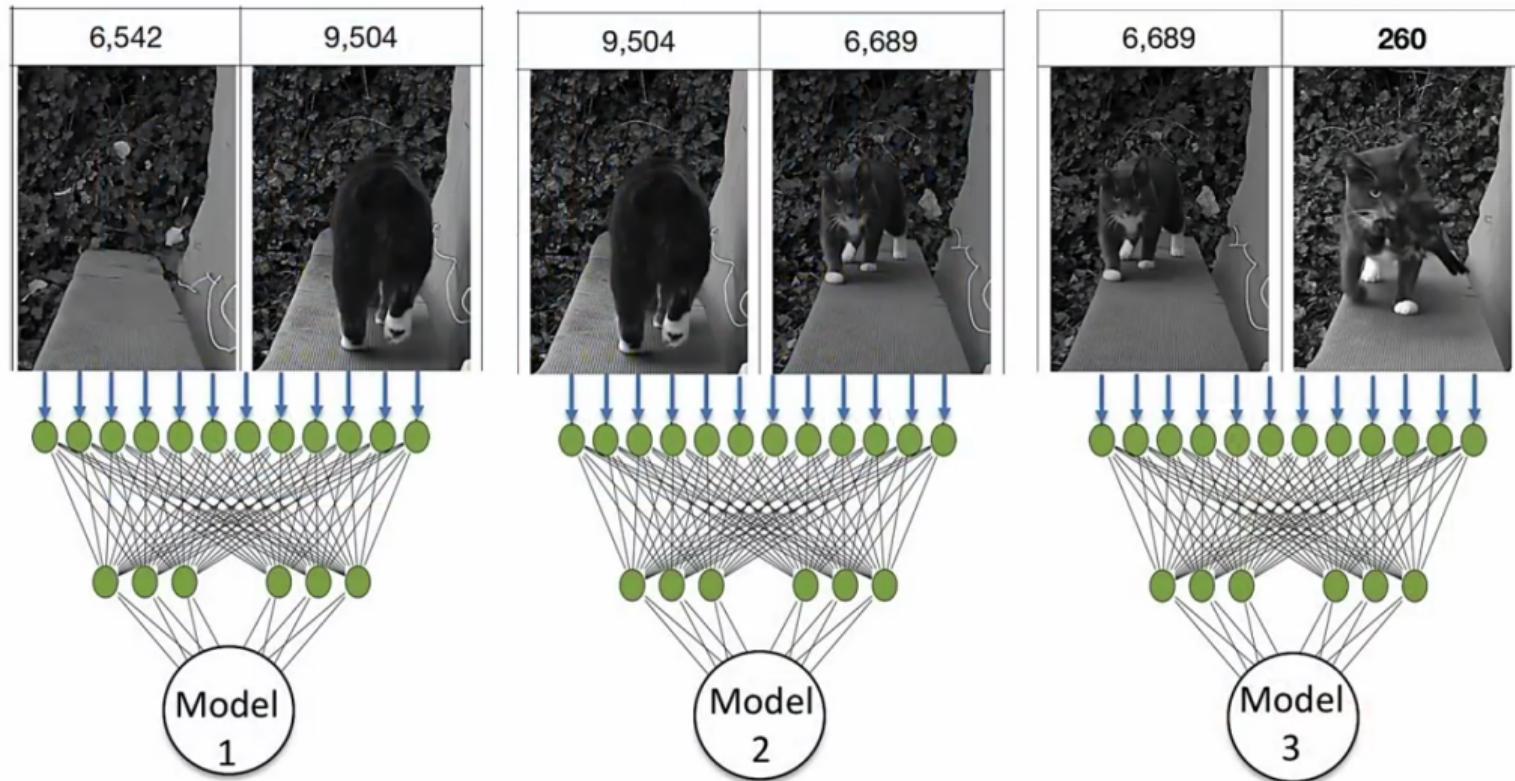




VS.



Image Type	No Cat	Cat not on approach	Cat on approach	Cat with prey
Count of Images	6,542	9,504	6,689	260
Example				



# CRITTER DETECTED!

Step 1:  
Lock the door!



Step 2:  
Text me pics

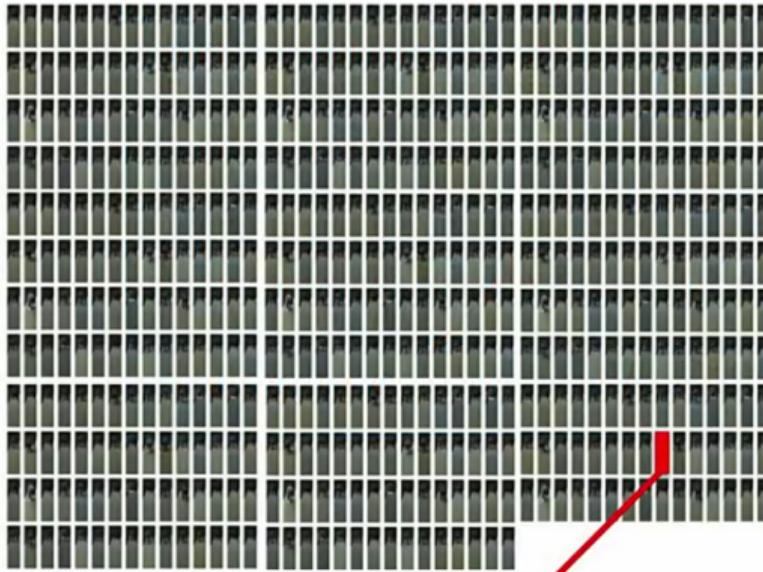


Step 3:  
Donate blood  
money



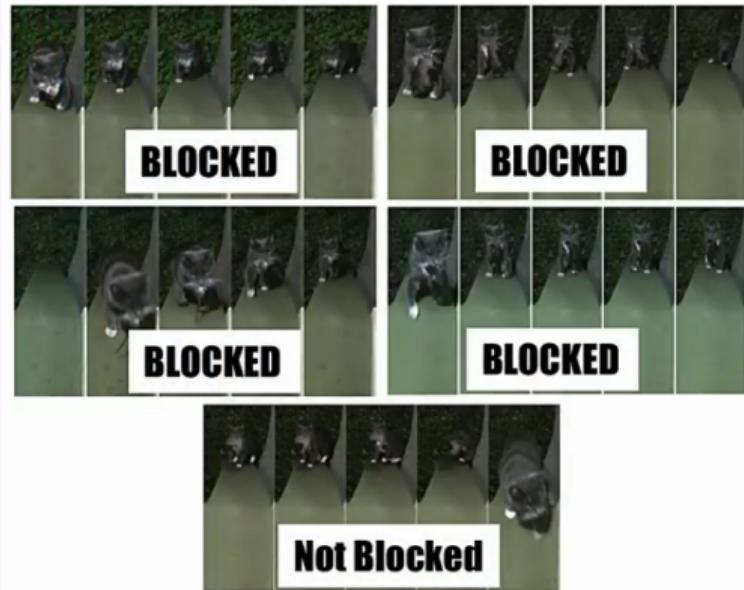
Audubon

## Innocent Entries



1 Unfair Lockout

## Entries with Prey



4/5 Critters Blocked

# Как он это сделал?

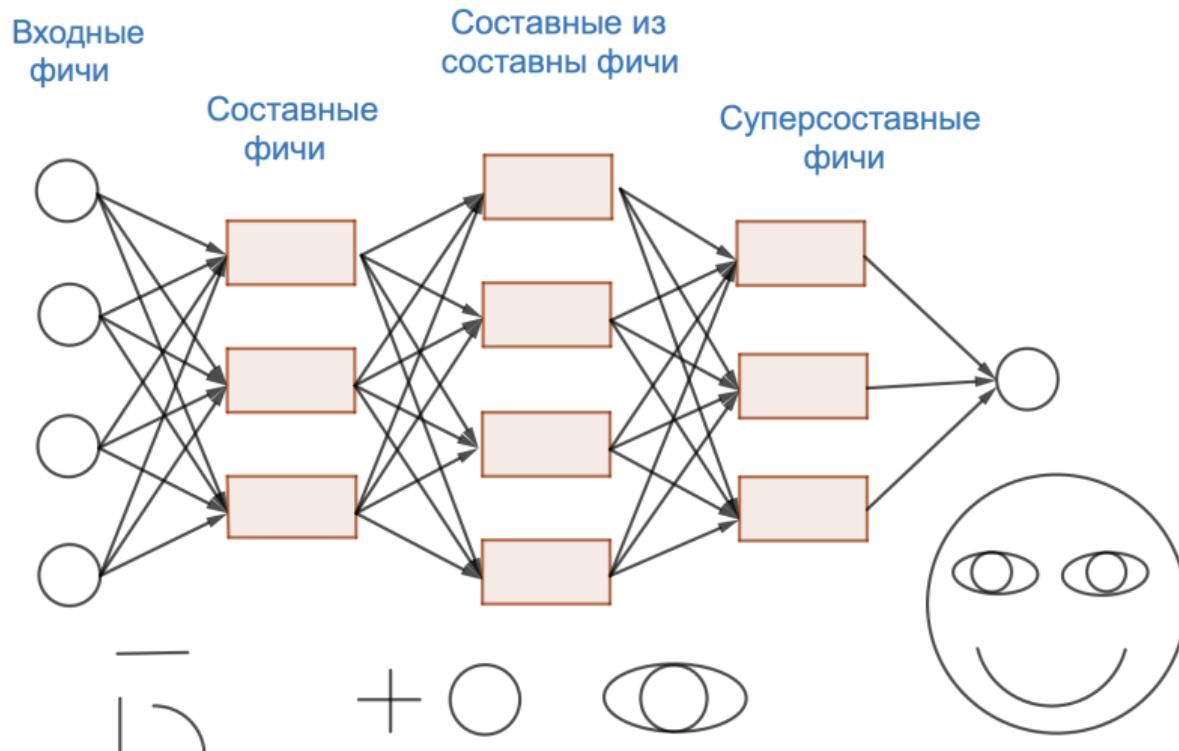
- В выборке было всего лишь 260 фотографий кота с добычей, неужели этого хватило для обучения сетки?
- На самом деле сетку с нуля никто не учил, делался transfer learning

# Transfer learning

# Как он это сделал?

- На практике свёрточные сети с нуля обучаются только огромные компании
- Это происходит из-за ограниченности ресурсов
- Уже обученные архитектуры пытаются адаптировать под новые задачи, это называется **transfer learning**

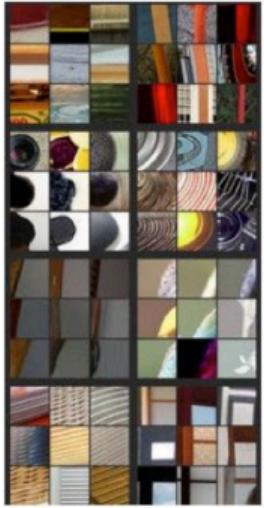
# Что выучивают нейросети



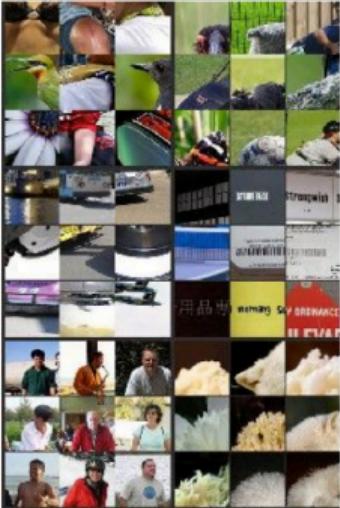
# Что выучивают нейросети



*Layer 1*



*Layer 2*



*Layer 3*



*Layer 4*

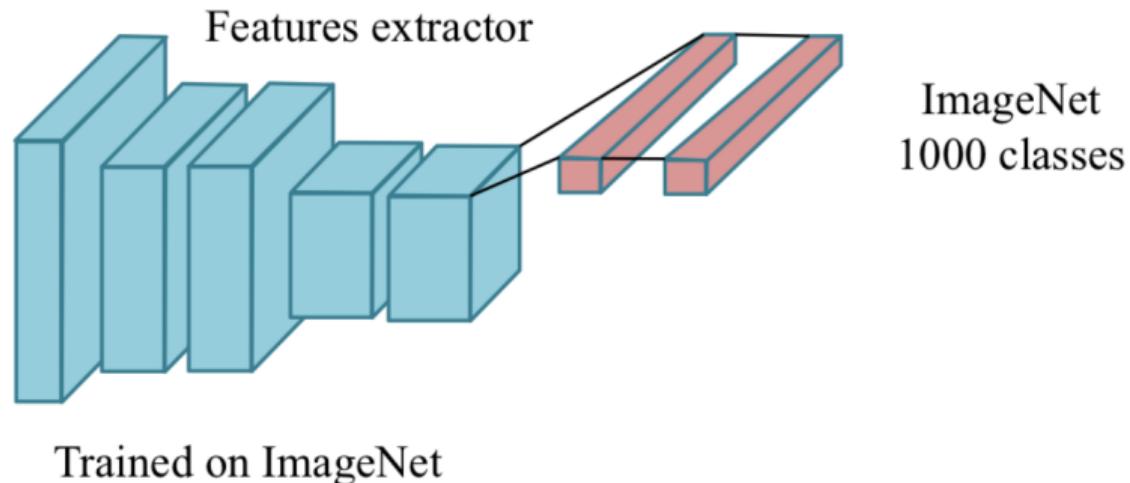


*Layer 5*

<https://arxiv.org/pdf/1311.2901.pdf>

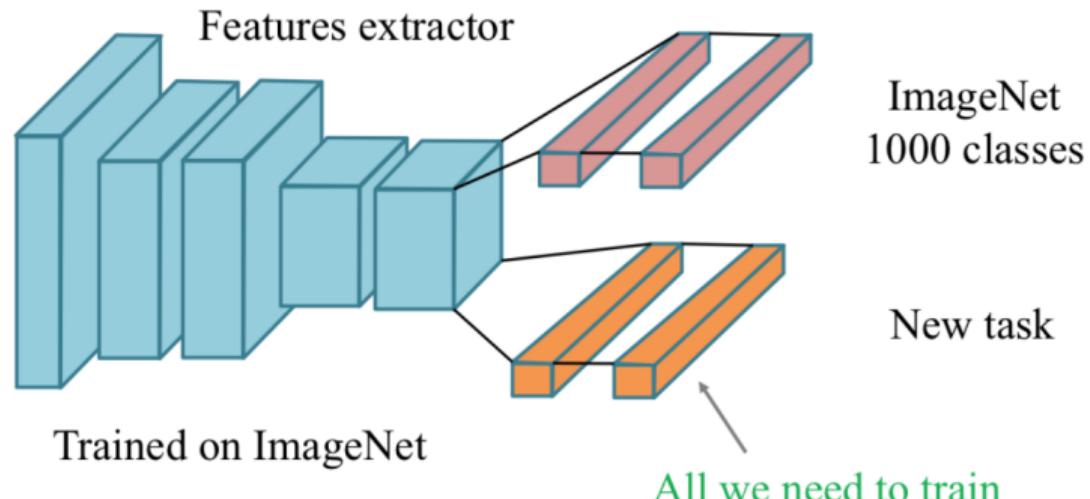
# Transfer learning

- Глубокие сети извлекают из изображений сложные фичи, но для их обучения нужно много данных...



# Transfer learning

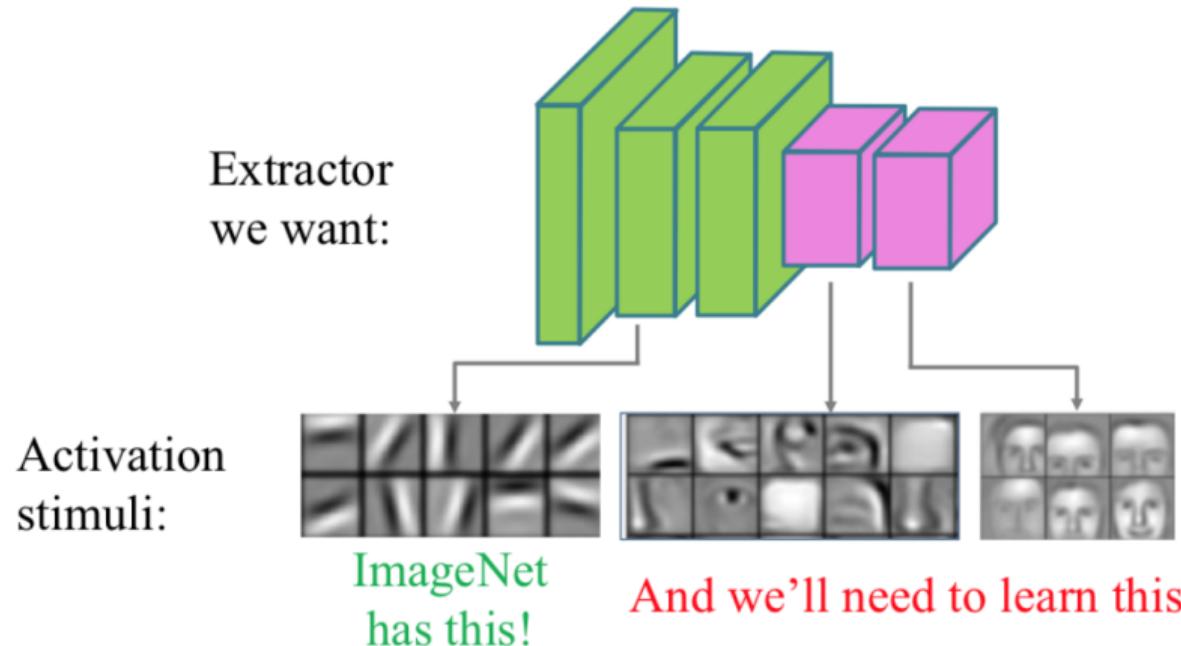
- Глубокие сети извлекают из изображений сложные фичи, но для их обучения нужно много данных...
- Давайте повторно использовать уже предобученную сеть!



# Transfer learning

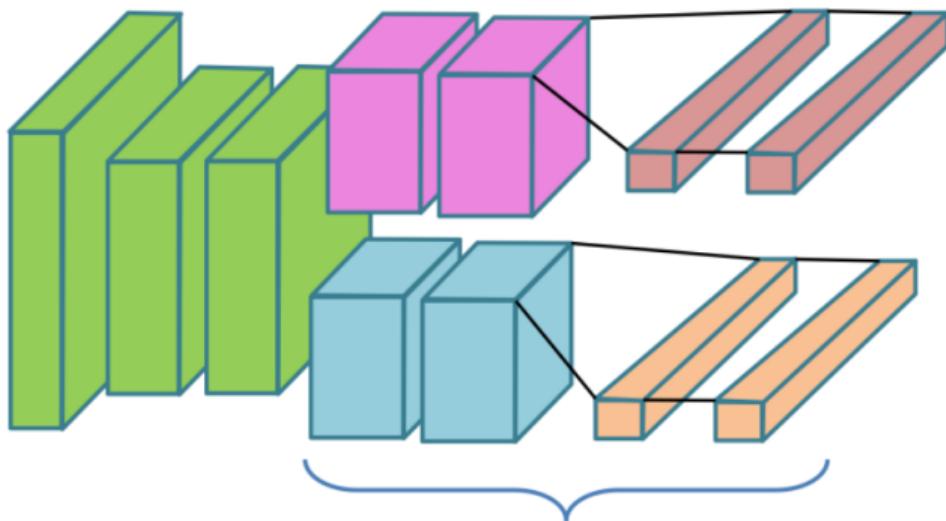
- Нужно меньше данных для обучения, так как нас интересуют лишь последние слои
- Это работает если наша задача похожа на ту, для которой обучалась используемая сетка
- Например, если мы хотим распознавать эмоции, в датасете для нашей сетки должны были быть человеческие лица

# Transfer learning



# Transfer learning

ImageNet features extractor



ImageNet  
1000 classes

New task

All we need to train

# Крокодил learning

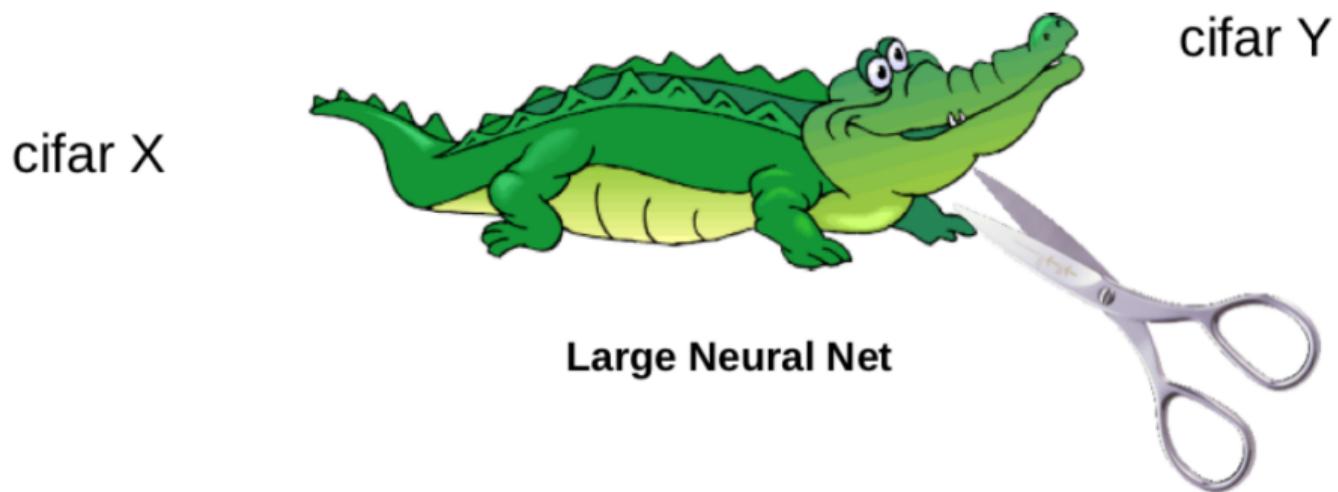
cifar X



cifar Y

Large Neural Net

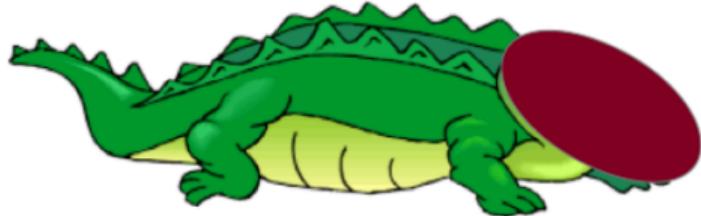
# Крокодил learning



# Крокодил learning

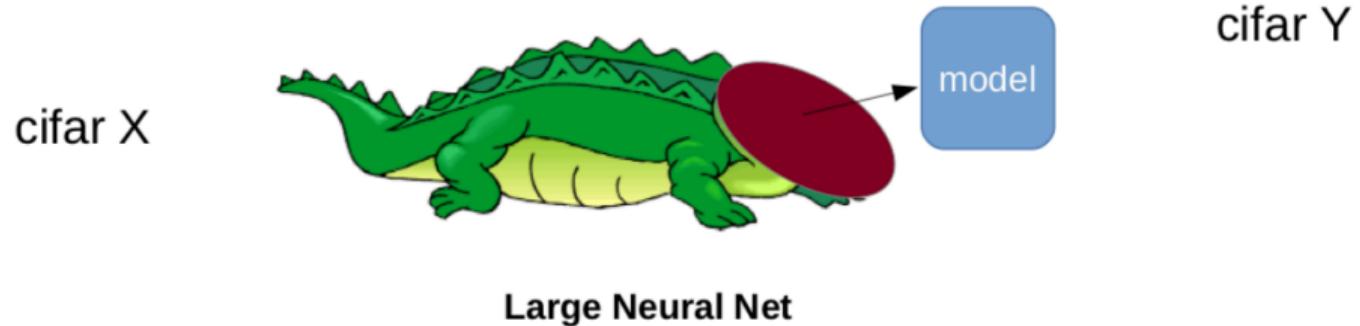
cifar X

cifar Y

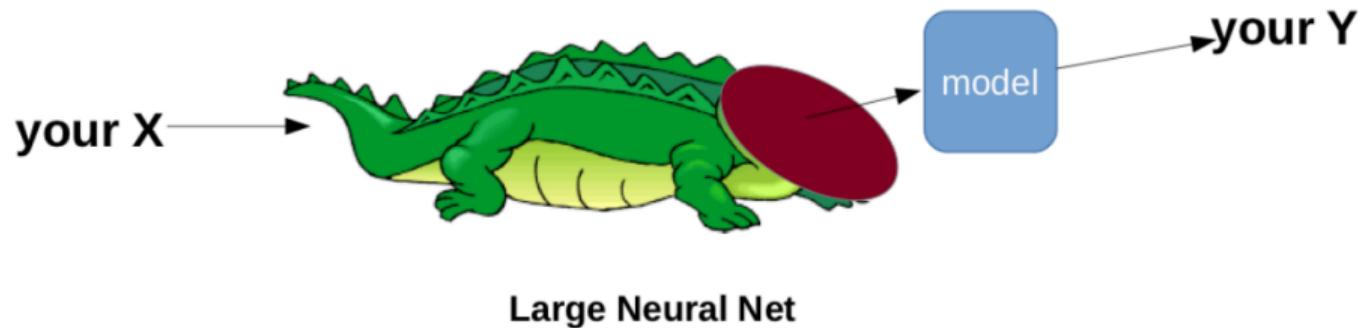


**Large Neural Net**

# Крокодил learning



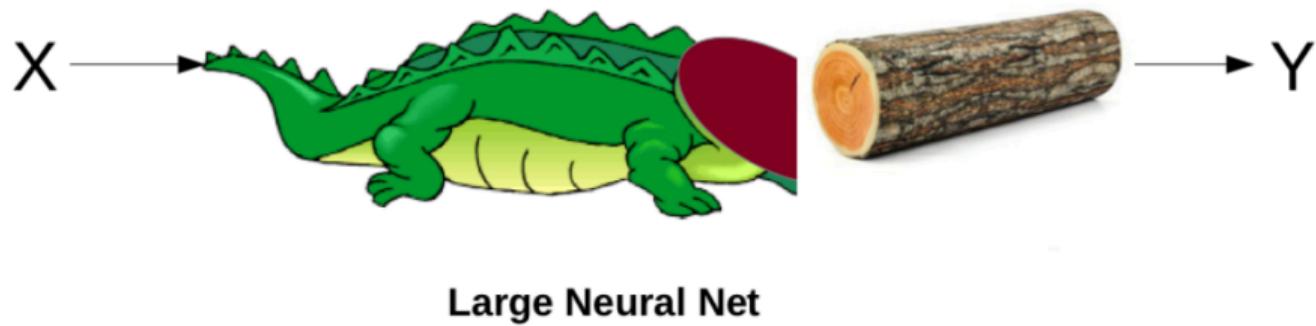
# Крокодил learning



# Крокодил learning

- Отрезали у крокодила голову
- Используем тело крокодила как экстрактор фичей
- Вместо головы крокодила можно прикрепить что угодно
- Даже случайный лес и бустинг
- В экстракторе фичей веса модели обычно не дообучают, дообучение касается только новой головы крокодила

# Крокодил learning



# Зоопарки моделей

- Перед тем как решать задачу с нуля, убедитесь, что готового решения  
ещё нет
- Зоопарк моделей внутри Tensorflow
- Другой большой зоопарк: <https://modelzoo.co/>

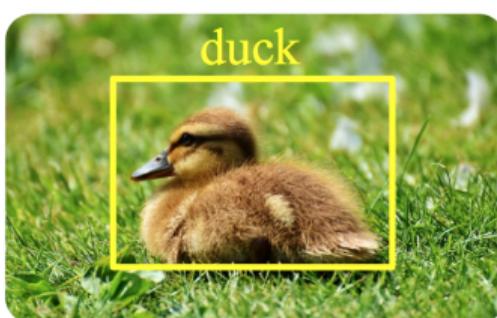
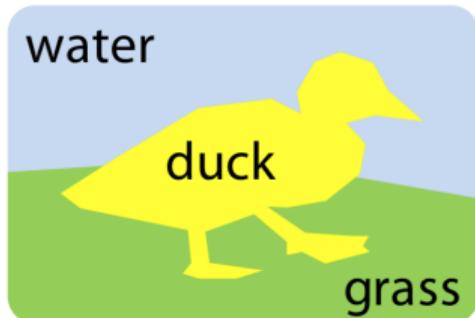
# Сегментация и локализация изображений

# Сегментация и локализация

Semantic segmentation:



Object classification  
+ localization:

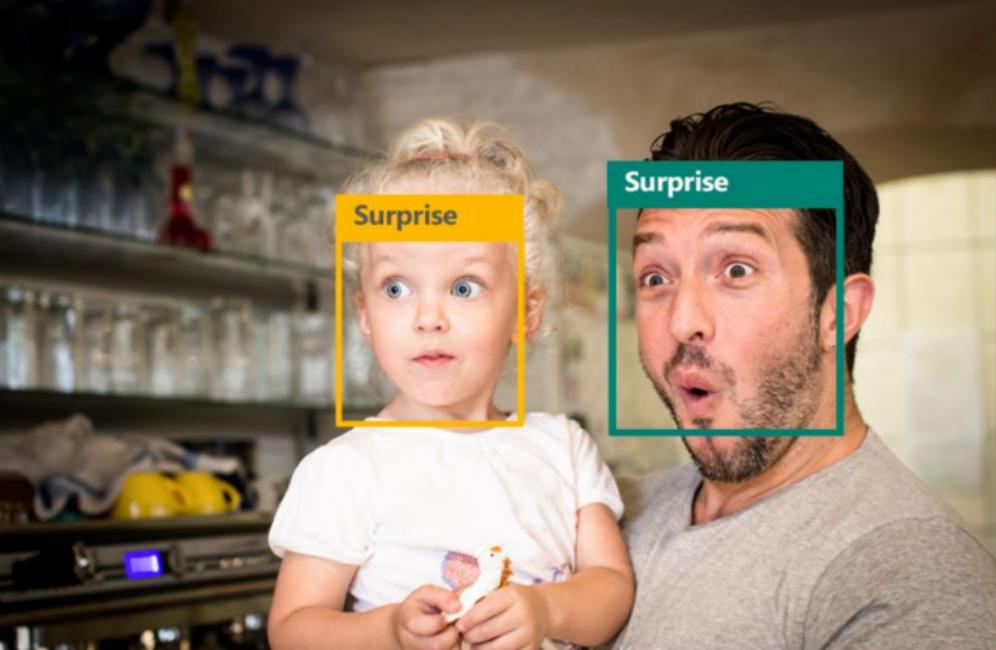


# Примеры



<https://www.youtube.com/watch?v=ZJMtDRbqH4o>

# Локализация Изображения



A photograph of a man holding a young child. Both individuals have yellow rectangular boxes overlaid on their faces, indicating detected surprise. The man is on the right, and the child is on the left.

Surprise

Surprise

Neutral:  

Happiness:  

Surprise:  

Sadness:  

Anger:  

Disgust:  

Fear:  

Contempt:  



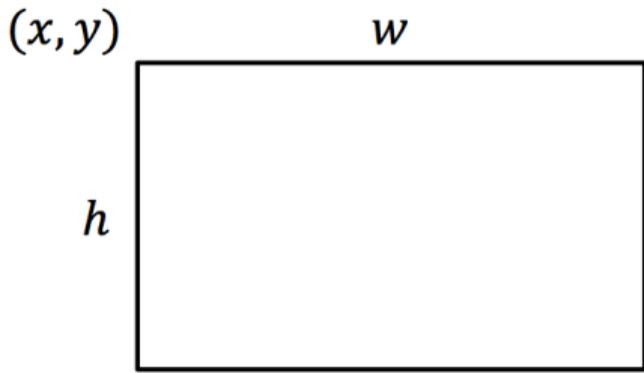
 Microsoft

Get started for free at [projectoxford.ai](http://projectoxford.ai)

# Примеры



# Локализация



- для локализации объекта нужно нащупать рамочку, в котором он находится
- рамочка описывается параметрами  $(x, y, w, h)$

# Локализация

