Сверточные нейронные сети

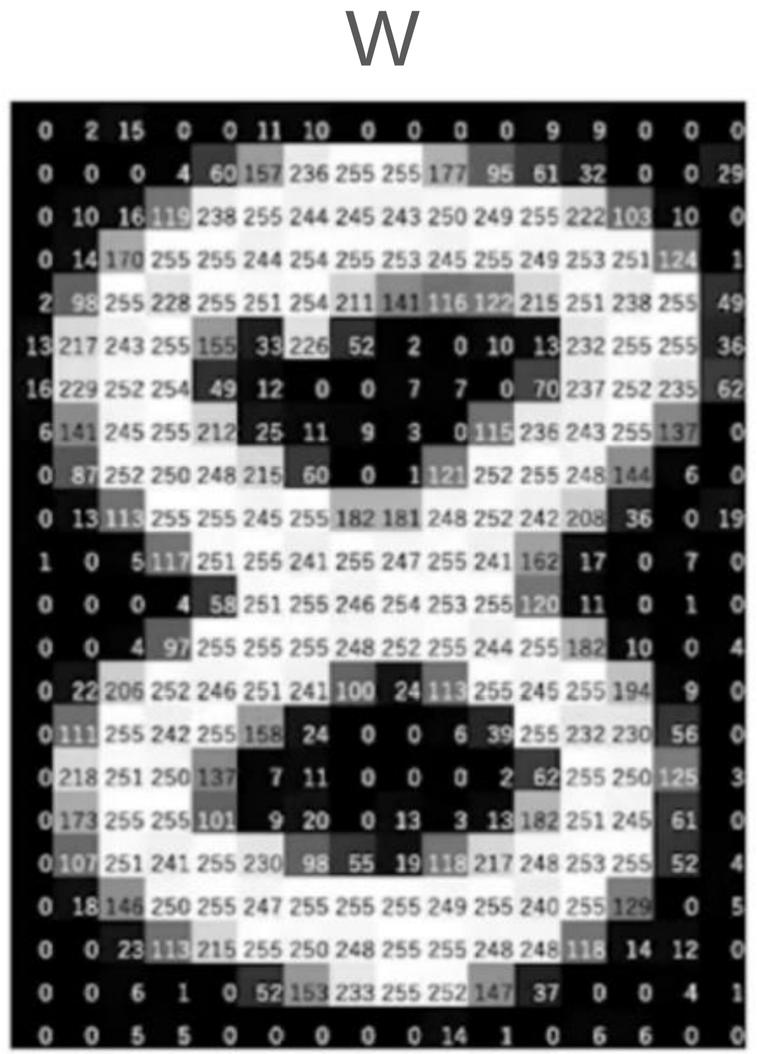
Стандартное представление изображения

Стандартное представление ч/б изображения

Черно-белая картинка — матрица из пикселей.
H — длина в пикселях,
W — ширина в пикселях.

Пиксель — число, означ. интенсивность цвета.
Интенсивность — число от 0 до 1.
Обычно ее кодируют числом от 0 до 255 (1 байт).

Черно-белая картинка —
тензор размера (H, W), состоящий из uint8 чисел.

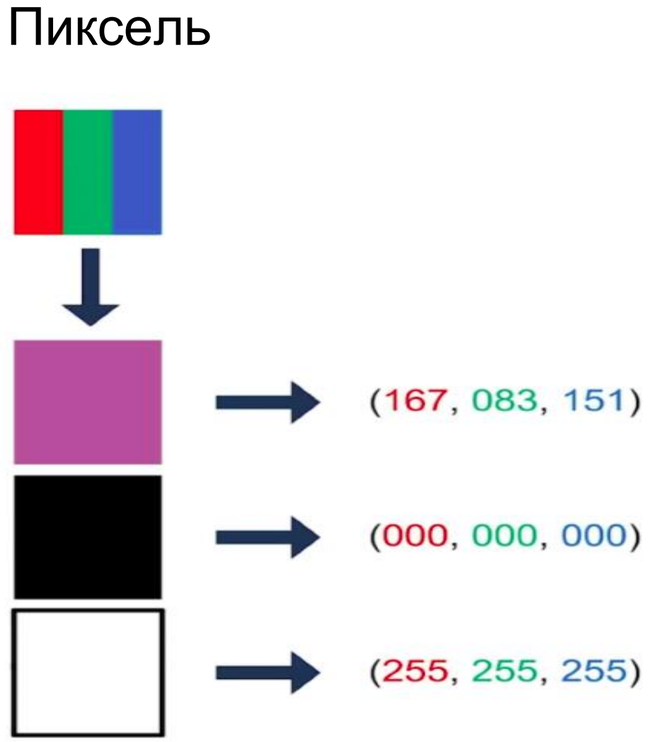


Стандартное представление цветного изображения

Цветная картинка — матрица из пикселей,
H — длина в пикселях,
W — ширина в пикселях.

Пиксель *обычно* представляют в RGB формате:
массив интенсивностей красного, зеленого и синего.
Интенсивность — число от 0 до 1.
Обычно ее кодируют числом от 0 до 255 (1 байт).

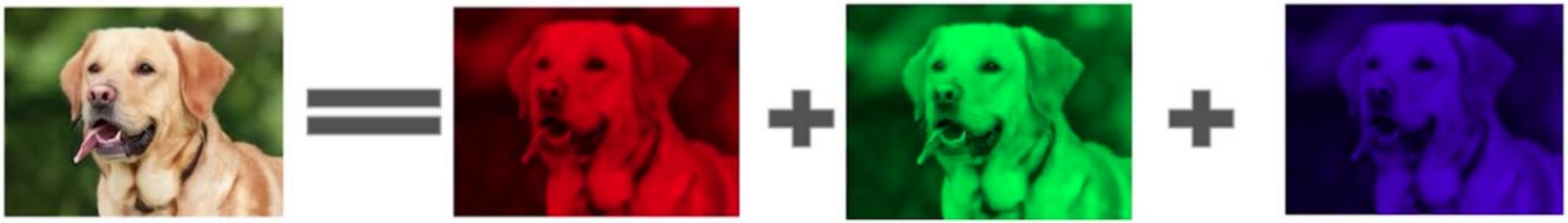
Цветная картинка —
тензор размера (H, W, 3), состоящий из uint8 чисел.



Стандартное представление изображения

Цветная картинка — тензор размера (H, W, 3), состоящий из чисел uint8.

Изображение можно разложить на *3 канала* по размерности пикселя.



Цветное изображение

Интенсивность красного

Интенсивность зеленого

Интенсивность синего

Представление изображения

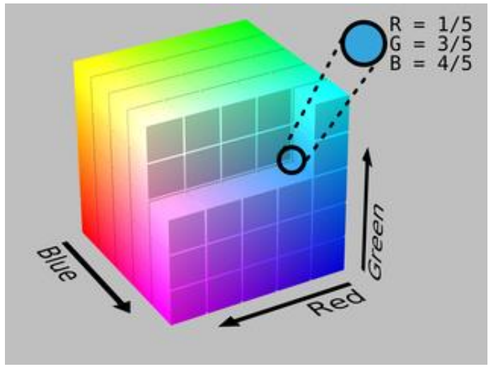
Для справки

Мы рассматривали цветовую модель RGB, но есть и другие.

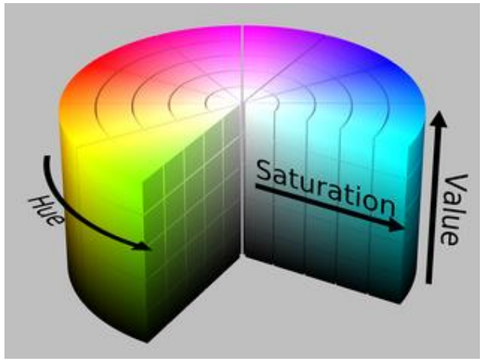
- RYB (red, yellow, blue) — красный, желтый, синий
- CMY / CMYK (cyan, magenta, yellow) — голубой, пурпурный, желтый
- HSL (hue, saturation, lightness) — оттенок, насыщенность, свет.
- HSV / HSB (hue, saturation, brightness) — оттенок, насыщенность, яркость
- ...

[Статья на Wikipedia]

Мы будем работать
со стандартным RGB.



RGB



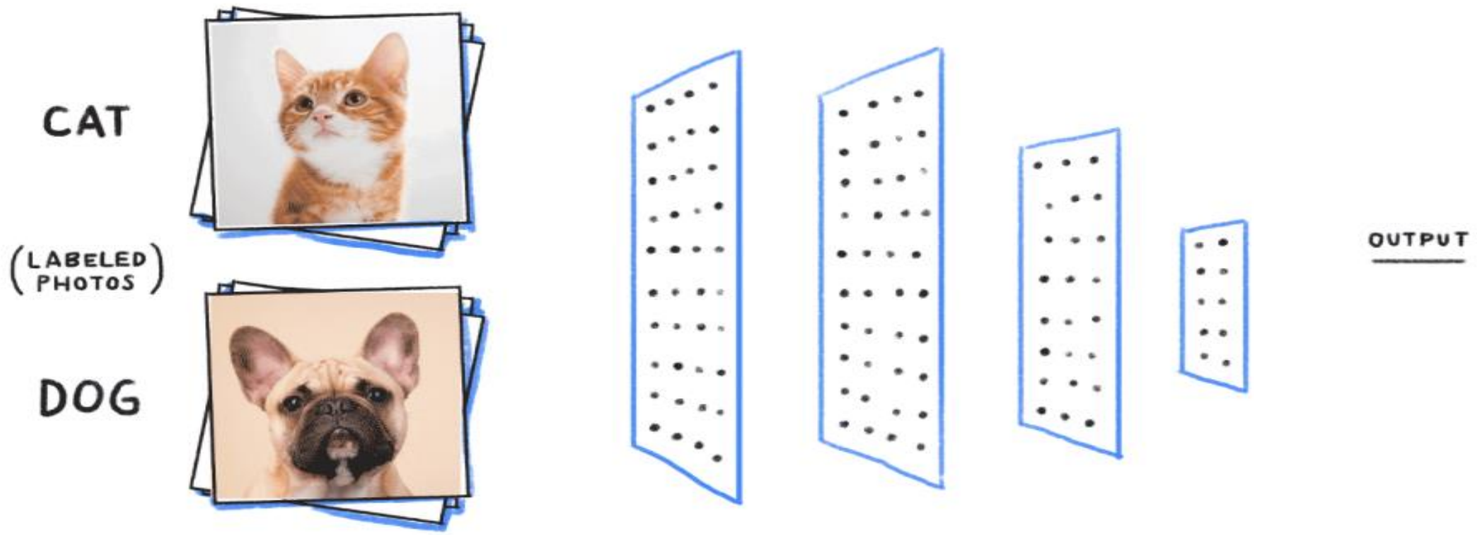
HSV

Классификация изображений

Задача классификации изображений

X— пространство картинок,
Y — набор классов, например {кошка, собака}.

Требуется построить модель $f: X \to Y$,
определяющую к какому классу относится объект на картинке.

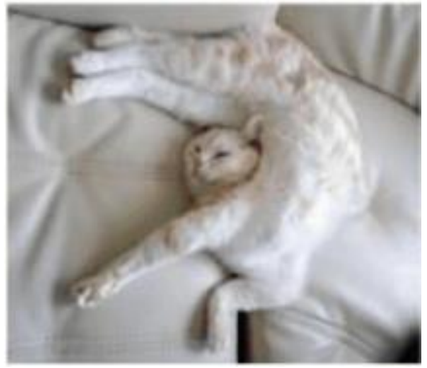


Проблемы классификации изображений

Разные углы обзора

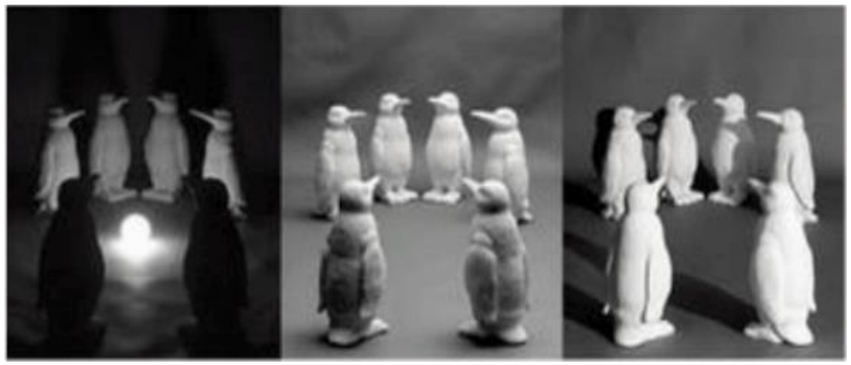
Разный размер

Деформация



Перекрытие

Разная освещенность



Фоновые помехи

Разная форма

Классификация изображений до нейросетей

Сгенерированные признаки подаем на вход модели машинного обучения.

Генерация признаков → Обучение модели → Предсказание

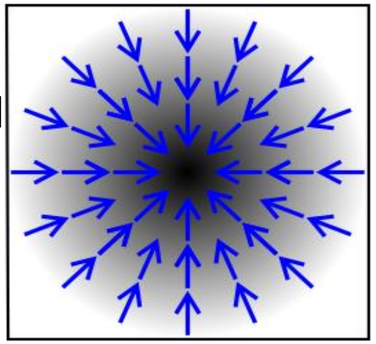
Полезные признаки изображения, для упрощения классификации

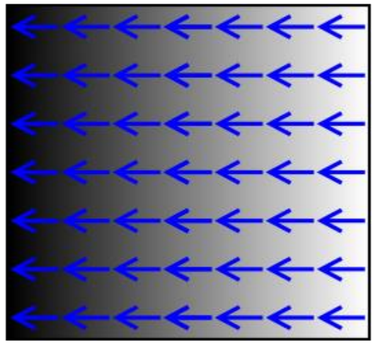
Лог. рег, деревья, бустинги, KNN, SVM и т.д.

Классификация изображений до нейросетей

Генерация признаков

- Распределение цветов на картинке.
- Распределение градиентов интенсивности (HOG фи
  Градиент в пикселе направлен в сторону наибольшего роста интенсивности в данной точке.



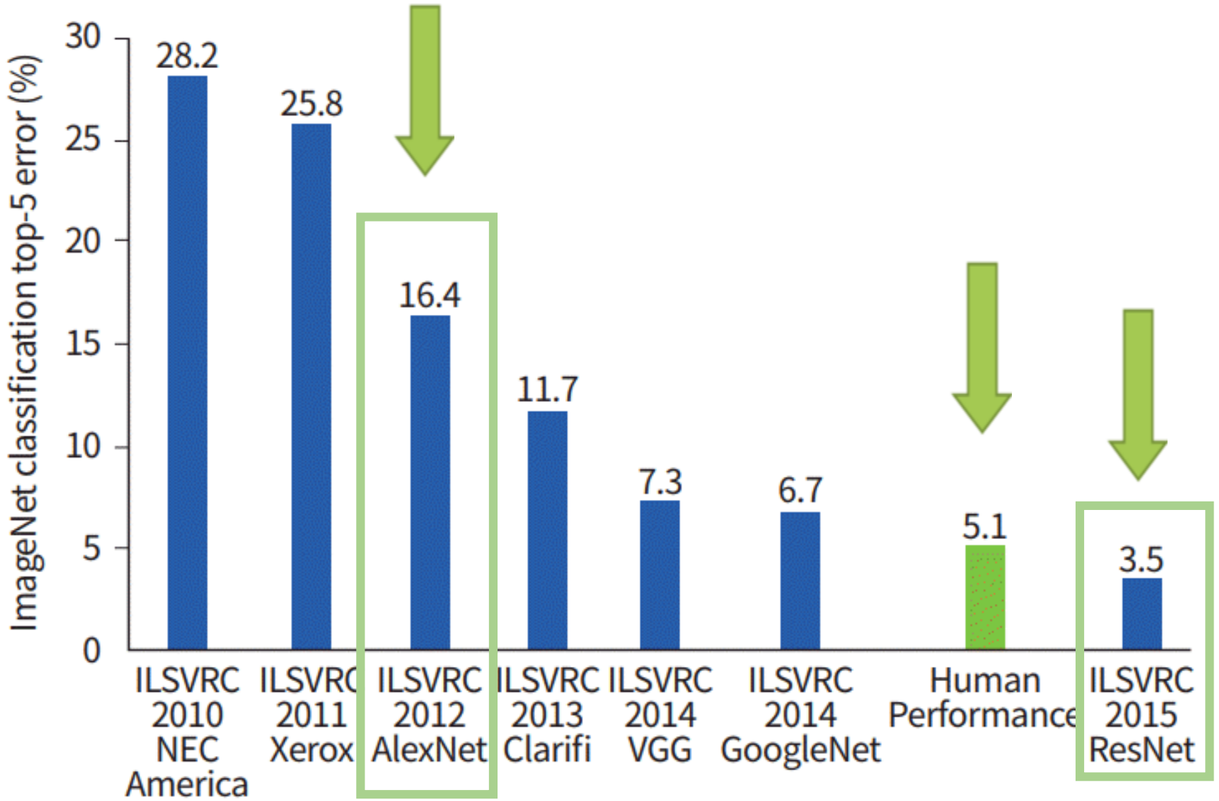


- Наличие и ориентация края в пикселе.
  Есть различные алгоритмы для поиска краев, например детектор Canny.
- Наличие характерных фрагментов текстуры.
  Задается большой словарь из характерных фрагментов разных текстур и осуществляется поиск похожих структур на изображении.

- Другие признаки. BoW, LBP признаки и прочее...

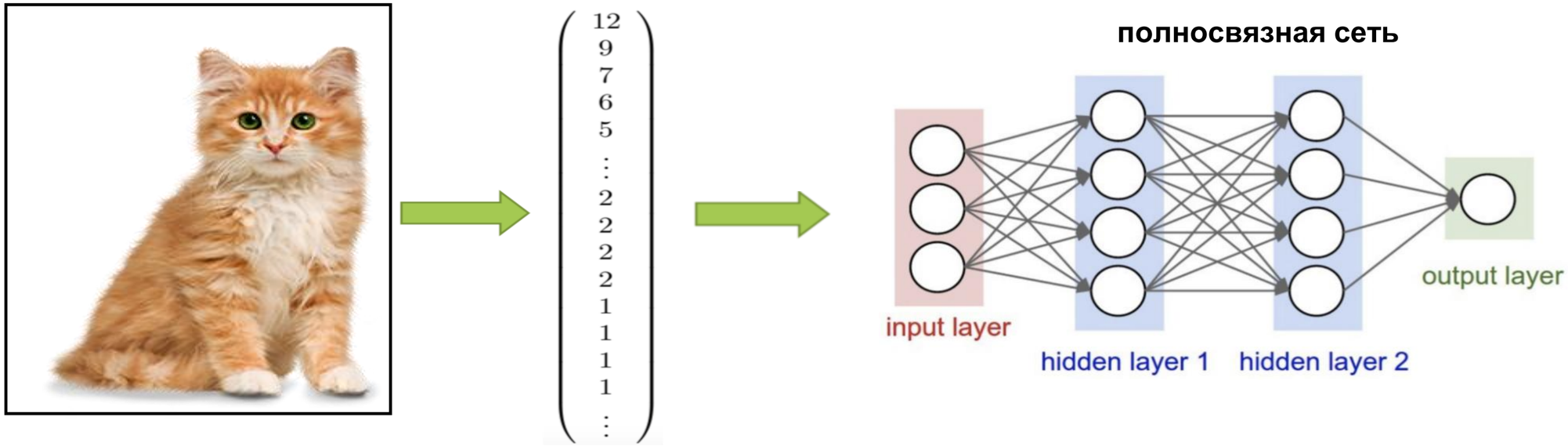
Историческая справка

- 1998 год — Ян ЛеКун предложил архитектуру первой CNN - **LeNet**.
- ~2010 год — появилась имплементация **LeNet**[5]
- 2012 год — свёрточная нейронная сеть **AlexNet** победила в конкурсе ILSVRC.
- 2015 год — свёрточная нейронная сеть **ResNet** обогнала по качеству человека в конкурсе ILSVRC.



Классификация изображений с помощью нейросетей

Как классифицировать картинки?



Почему такой подход плох?

1. Не учитывает явно взаимное расположение пикселей.
2. Расположение объекта на картинке не должно иметь значения для классификации.
3. Сеть нельзя сделать очень глубокой, она будет иметь слишком много параметров.

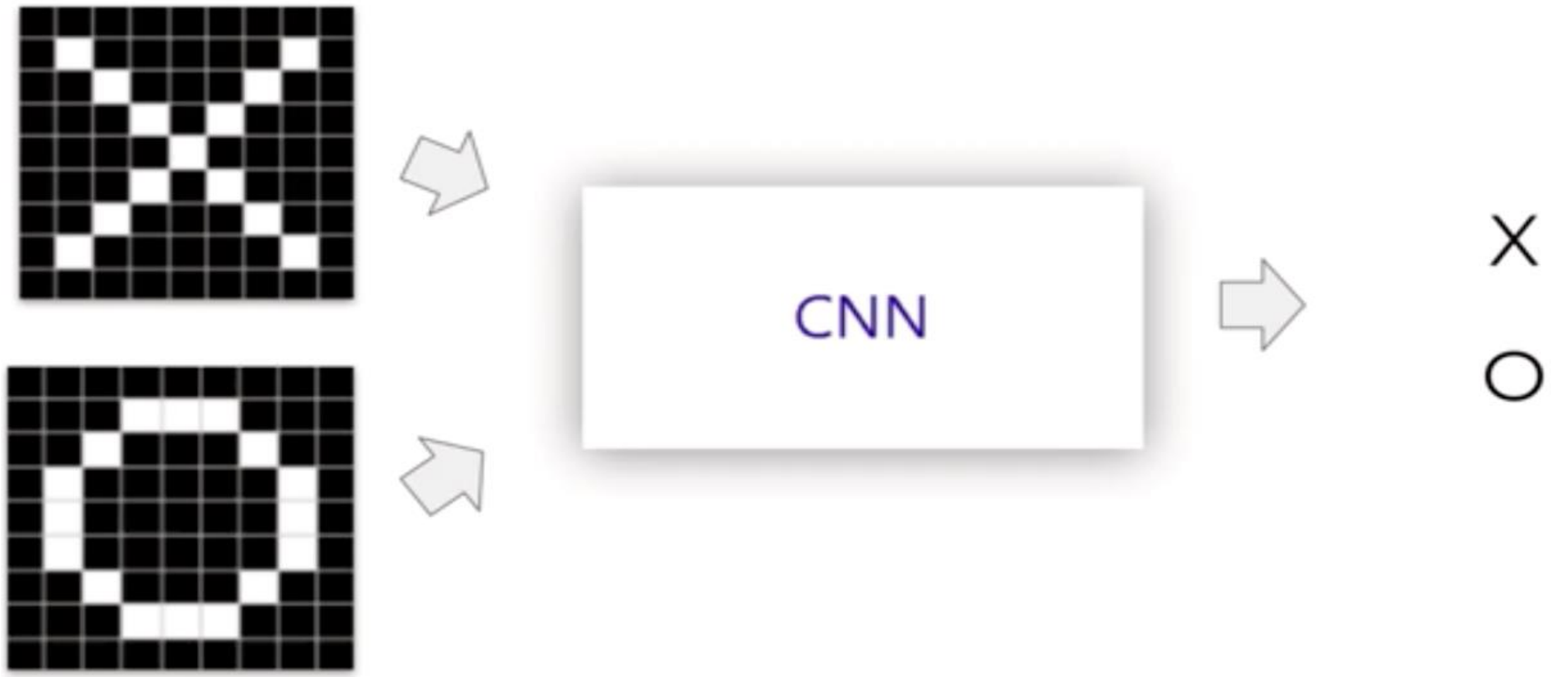
Интуиция к CNN

Поиграем в классификацию

Требуется уметь находить **крестики и нолики** на картинке.

Причем, объекты могут

- находиться в *разных местах* картинки,
- быть *не в точности равны* искомым паттернам.



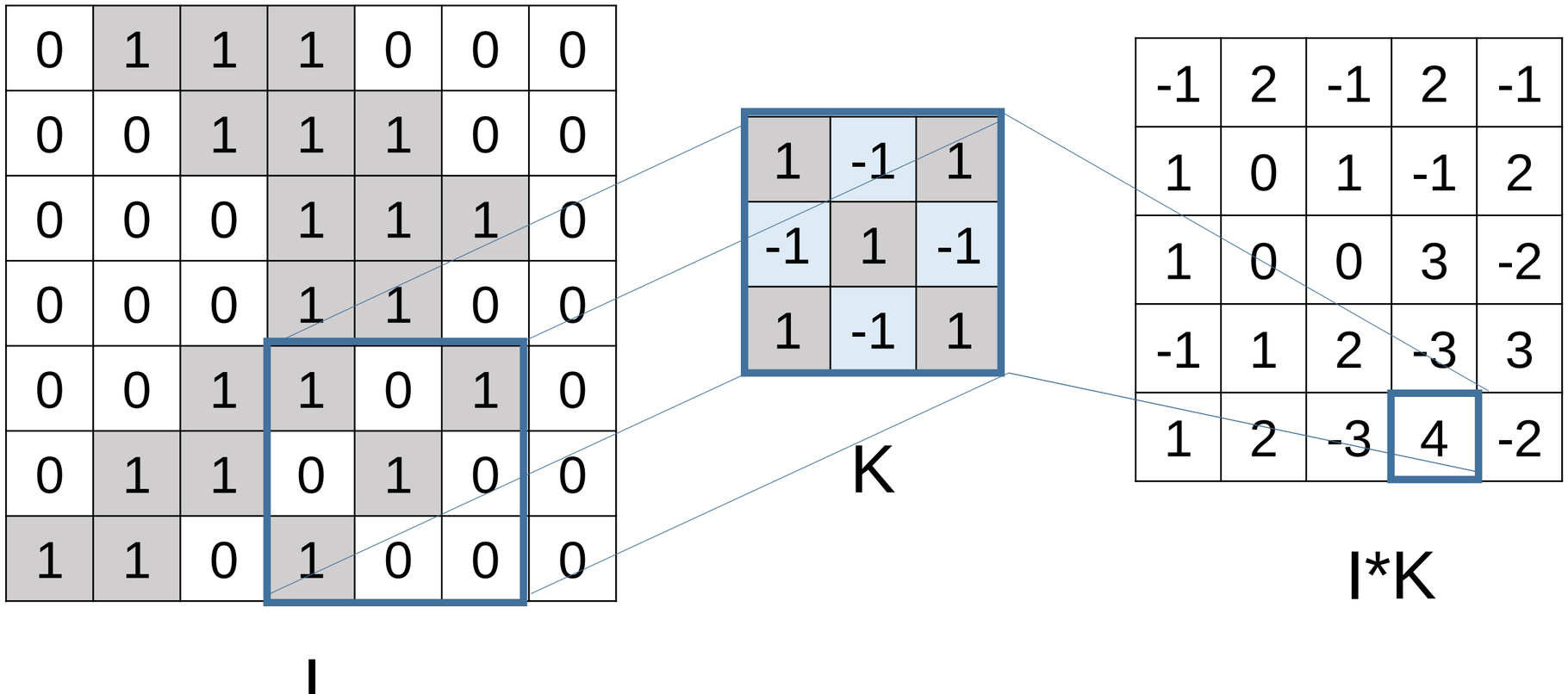
Классификация изображений с помощью нейросетей

Идея: сделаем локальный поиск паттернов

Берем паттерн крестика и ищем похожий паттерн на картинке.

- Проходимся **окнами** по картинке
- Считаем **схожесть** этой части картинки и паттерна.
- Записываем результат в соотв. место в **матрице**.

Чем больше значение в матрице, тем больше похожа область картинки на паттерн.

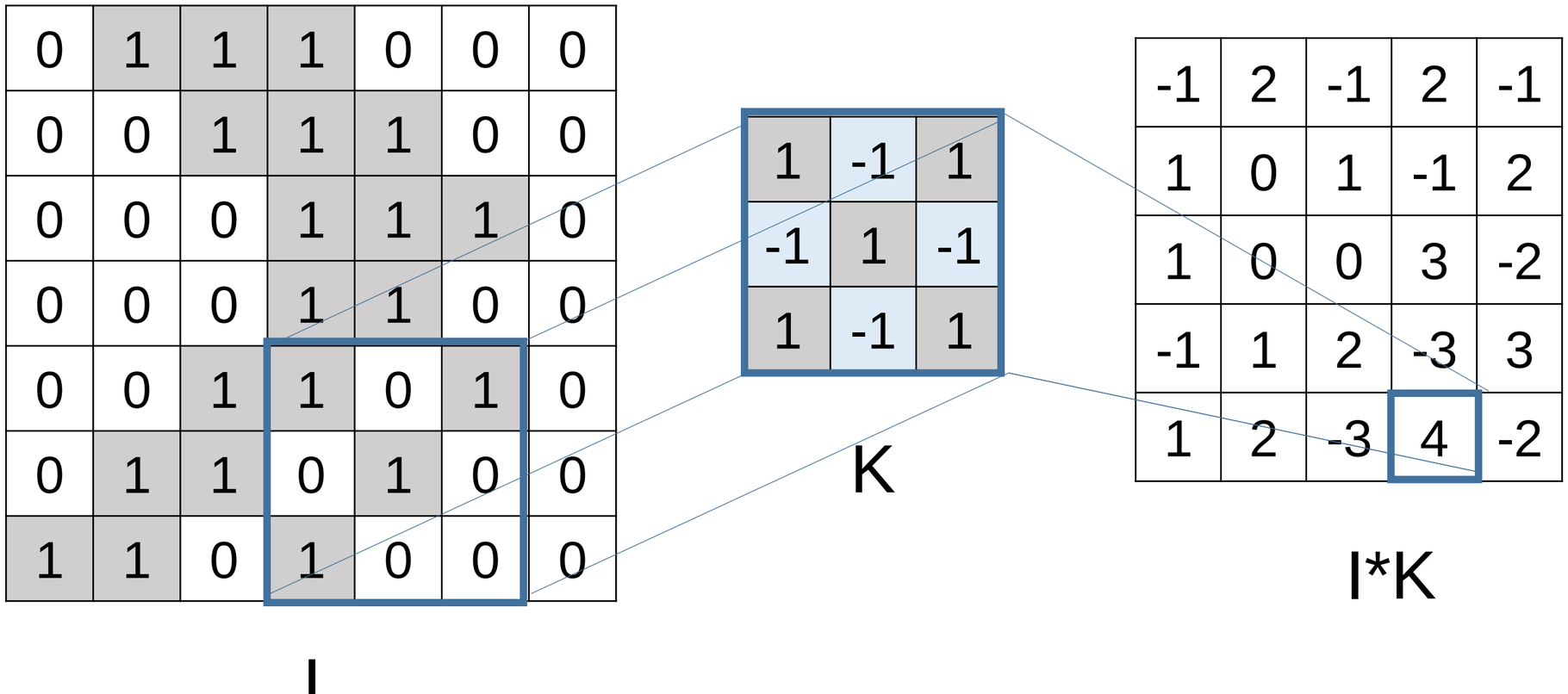


Классификация изображений с помощью нейросетей

Как считать схожесть?

Чтобы оценить схожесть двух паттернов перемножим их **скалярно**:

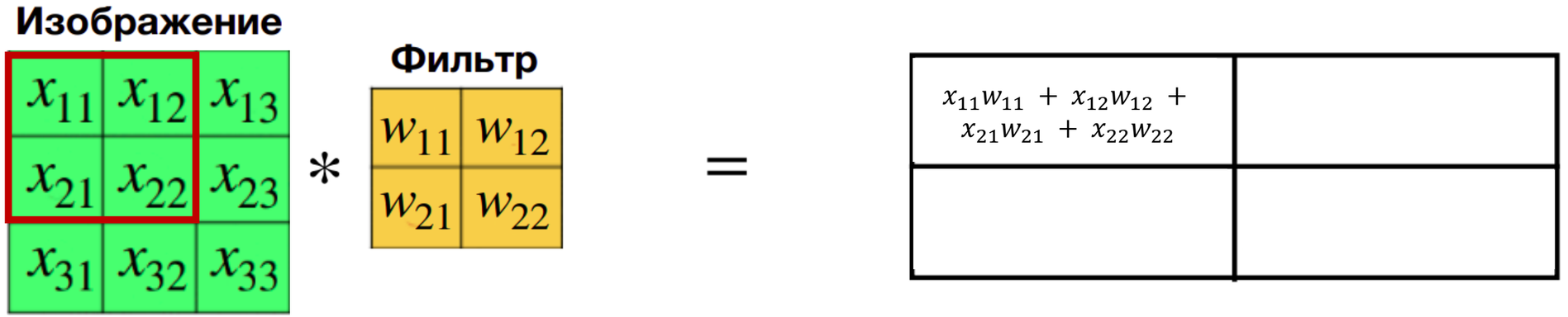
1. Умножим матрицы поэлементно.
2. Сложим числа полученной матрицы.



2D свертка / 2D convolution

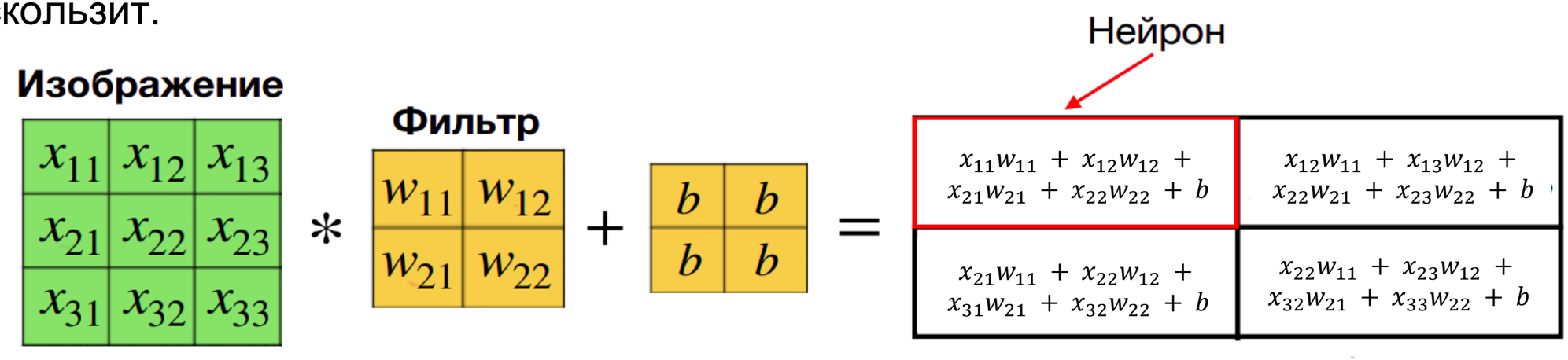
2D свертка / 2D convolution

Фильтр / ядро, представляющий из себя матрицу весов,
пробегает по исходным данным и вычисляет скалярные произведения
с той частью изображения, над которой он сейчас находится.



2D свертка / 2D convolution

Фильтр / ядро, представляющий из себя матрицу весов,
пробегает по исходным данным и вычисляет скалярные произведения
с той частью изображения, над которой он сейчас находится.
Ядро повторяет эту процедуру с каждой локацией, над которой оно
скользит.



После чего к каждому элементу также добавляется смещение *b*.

Каждый отдельный элемент в полученной матрице — нейрон.

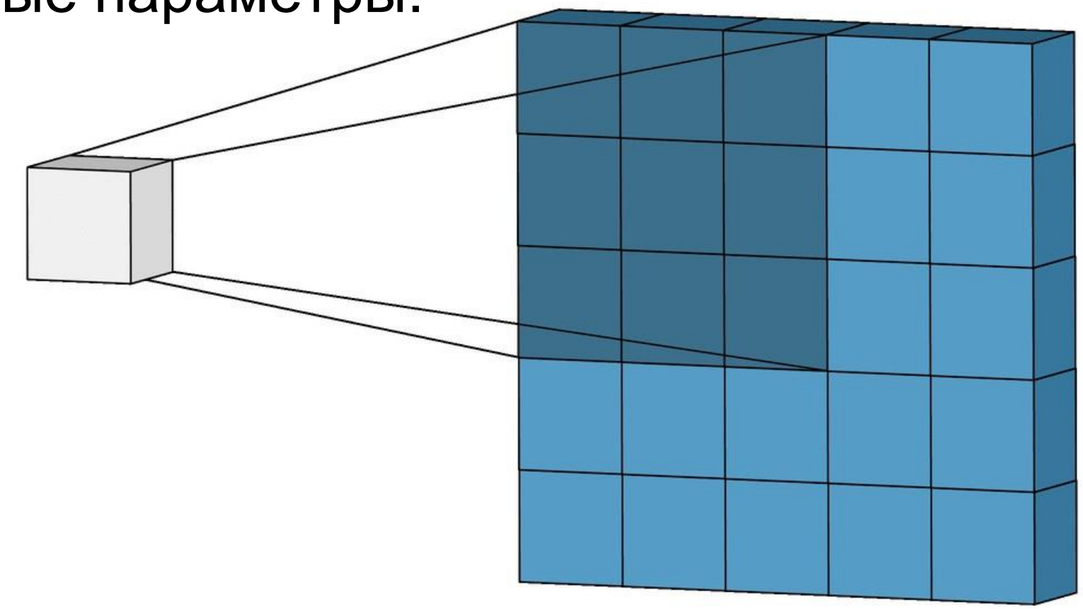
Формула свёртки: $(I * F)_{m,n} = \sum_{i=1}^{M} \sum_{j=1}^{M} w_{ij} \cdot x_{m+i-1,n+j-1} + b$

где I *— изображение,* F *— фильтр размера* (M, M).

2D свертка / 2D convolution

Веса в фильтре и смещение — обучаемые параметры.

Размер фильтра — гиперпараметр.



Исходные данные

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

Ядро

Выход

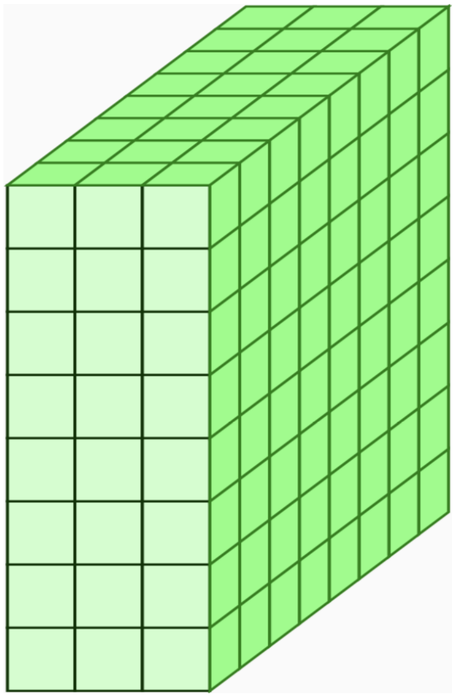
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Многоканальный вход

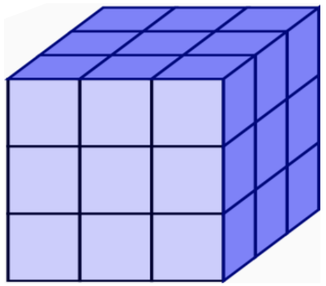
Вход — трехмерные тензоры размера $H \times W \times C$ (высота, ширина, каналы).

Ядро — трехмерная матрица размера $k_x \times k_y \times C$.
Обычно $k_x = k_y$, где k_x, k_y — гиперпараметры.

Изображение $8 \times 8 \times 3$

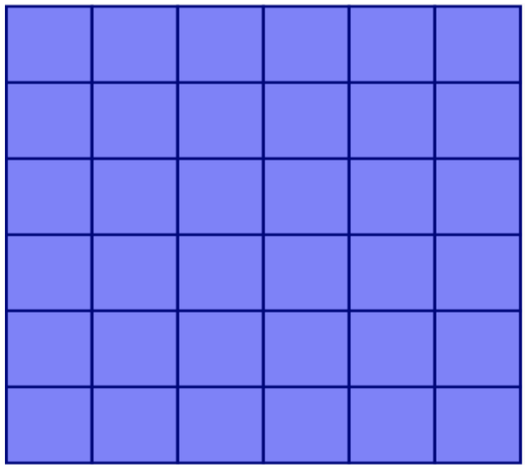


Фильтр $3 \times 3 \times 3$



Смещение

Карта 6×6



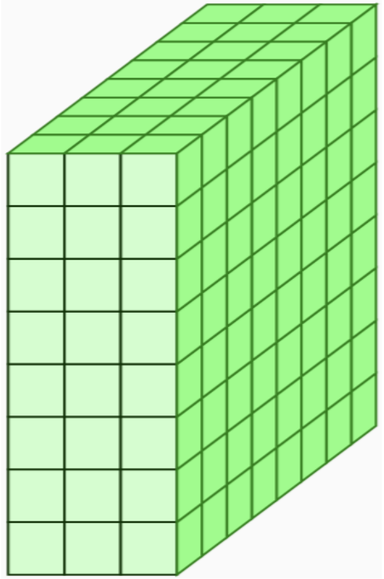
Много фильтров

Один фильтр → одна карта, соответствующая одному паттерну.

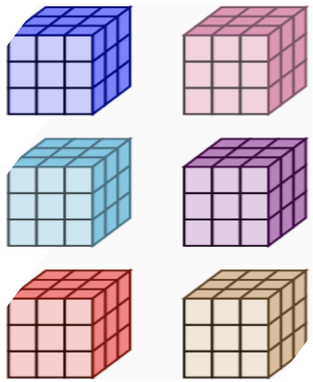
Возьмем K разных фильтров и применим свёртку с ними.

Получим K карт на выходе.

Изображение 8 × 8 × 3

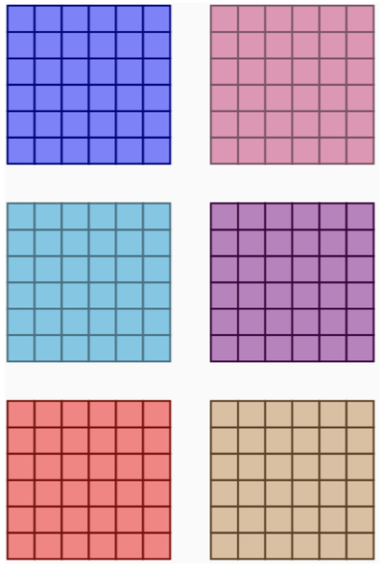


6 фильтров 3 × 3 × 3



Смещения

6 карт 6 × 6



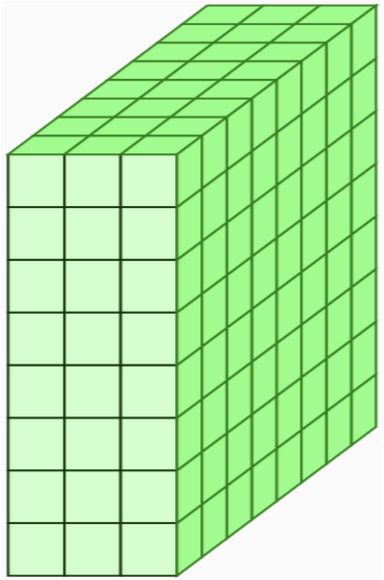
Много фильтров

Один фильтр → одна карта, соответствующая одному паттерну.

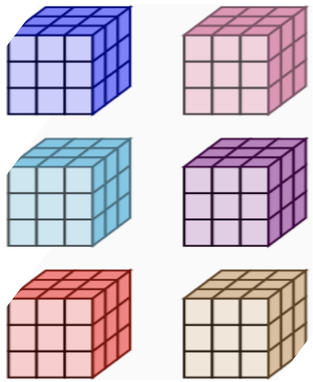
Возьмем K разных фильтров и применим свёртку с ними.

Получим K карт на выходе.

Изображение 8 × 8 × 3

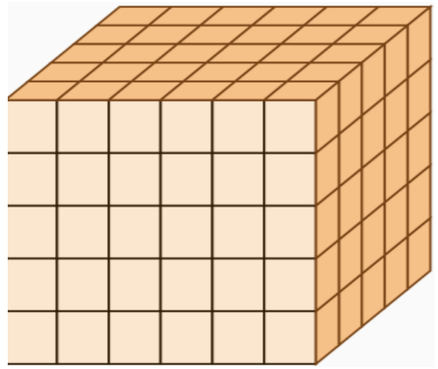


6 фильтров 3 × 3 × 3



Смещения

6 карт 6 × 6



Выходную матрицу 6 × 6 × 6 можно использовать
как вход следующего свёрточного слоя.

Отступ / Padding

Проблема

1.Крайние пиксели никогда не оказываются в центре ядра.
2.Выходной размер получается меньше входного.

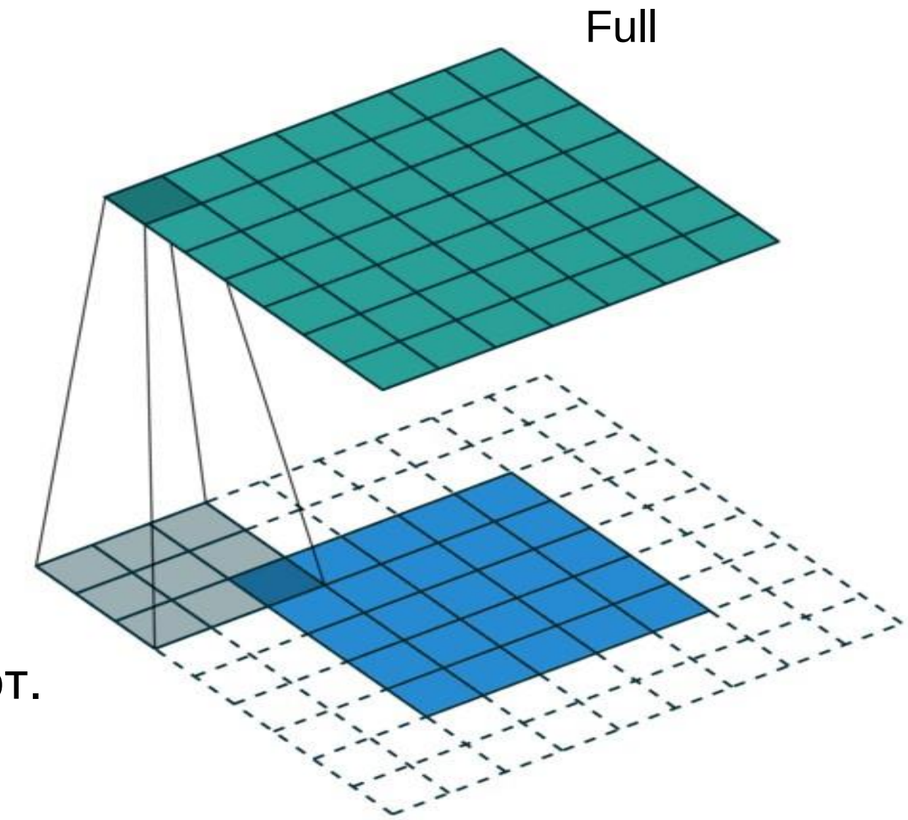
Padding добавляет по краям поддельные пиксели.
Тогда позволяем краевым неподдельным пикселям оказываться в центре ядра.

Какими значениями заполнять?

- Нулями (zero-padding)
  Самый популярный вариант.
  Сеть учиться понимать, что окно находится на границе картинки.
- Значением ближайшего пикселя

Виды

- Same: высота и ширина выходн. и входн.изобр. совпадают.
- Valid: отсутствие паддинга.
- Full: высота и ширина выходн. изобр. больше входн.

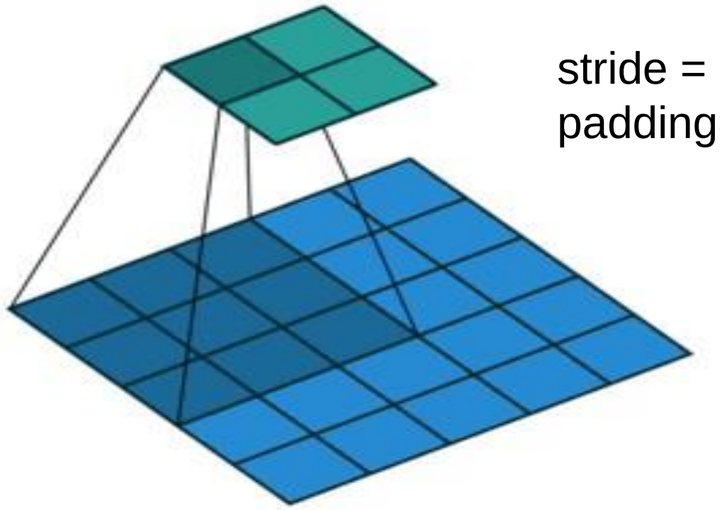


Шаг / Striding

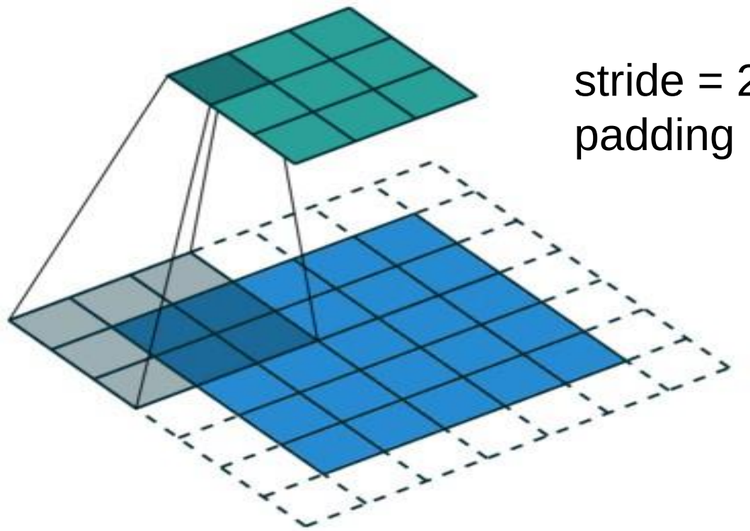
Идея.
Пропустим некоторые области, над которыми скользит ядро.

Stride = 1 - окна сдвигаются на 1 пиксель.
Stride = 2 - окна сдвигаются на 2 пикселя.

Таким образом, размер выходной матрицы уменьшается.
Искомые паттерны всё равно должны найтись,
если исходное изображение достаточно большое.



stride = 2
padding=0



stride = 2
padding = 1

Подсчет размеров

Пусть входная картинка имеет размер $H_{in} \times W_{in} \times C_{in}$.
Размер применяемого фильтра $M \times M \times C_{in}$.

Пример 1
Применяем операцию свертки с **stride=1, padding=0**.
Какой размер будет у выхода после свертки?

Размер карты:

$$H_{out} = H_{in} - M + 1$$
$$W_{out} = W_{in} - M + 1$$

Применив C_{out} фильтров, получим размер

$$H_{out} \times W_{out} \times C_{out}$$

Подсчет размеров

Пусть входная картинка имеет размер $H_{in} \times W_{in} \times C_{in}$.
Размер применяемого фильтра $M \times M \times C_{in}$.

Пример 2
Применяем операцию свертки с **stride=S, padding=0**.
Какой размер будет у выхода после свертки?

Окна, с которыми берем скаляр. произвед. имеют координаты по горизонтали
$[1, M], [1 + S, S + M], [1 + 2S, 2S + M], \ldots, [1 + kS, kS + M]$,
где k — такое целое число, при котром $[1 + kS, kS + M]$ — последнее окно.
Тогда должно быть выполнено: $kS + M \leq H_{in}$, значит

$$H_{out} = k + 1 = \lfloor (H_{in} - M) / S \rfloor + 1$$

Аналогично

$$W_{out} = k + 1 = \lfloor (W_{in} - M) / S \rfloor + 1$$

Применив C_{out} фильтров, получим размер

$$H_{out} \times W_{out} \times C_{out}$$

Подсчет размеров

Пусть входная картинка имеет размер $H_{in} \times W_{in} \times C_{in}$.
Размер применяемого фильтра $M \times M \times C_{in}$.

Пример 3
Применяем операцию свертки с **stride=S, padding=P**.
Какой размер будет у выхода после свертки?

Добавляем P пикселей с каждой стороны входного изображения

$$H_{out} = \lfloor (H_{in} - M + 2P) / S \rfloor + 1$$

Аналогично

$$W_{out} = k + 1 = \lfloor (W_{in} - M + 2P) / S \rfloor + 1$$

Применив C_{out} фильтров, получим размер

$$H_{out} \times W_{out} \times C_{out}$$

Больше сверток

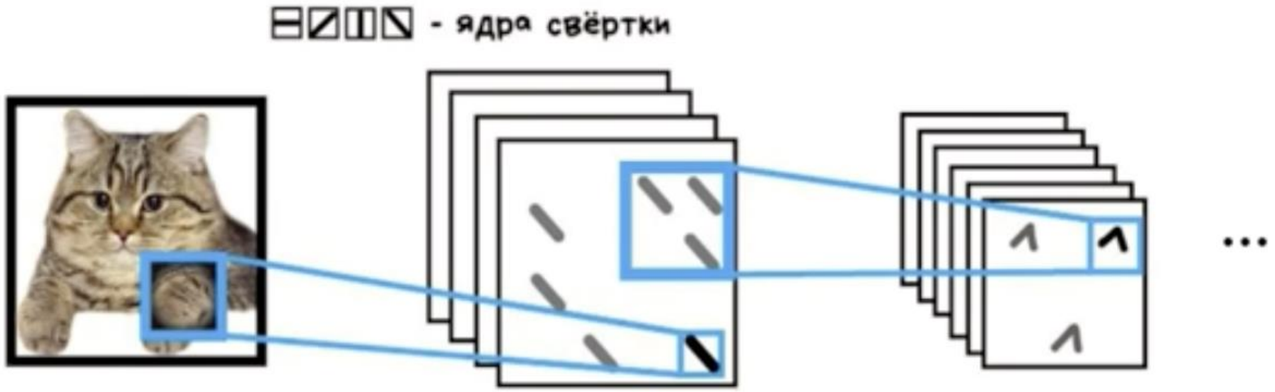
- **Одной свёрткой** можем узнать только наличие **простых паттернов** на картинке.
- Одной свёрткой **не можем найти сложные паттерны**.

  Если фильтр будет изображать лицо кота, то мы вряд ли найдем что-то похожее на картинке с котом, ведь коты бывают разные.

Идея

Сделаем несколько свёрток подряд.

- Первая свёртка будет искать простые паттерны на исходной картинке.
- Вторая будет искать простые паттерны уже на картах после первой свёртки.
  Простые паттерны из простых паттернов —уже более сложные паттерны.
- Третья …

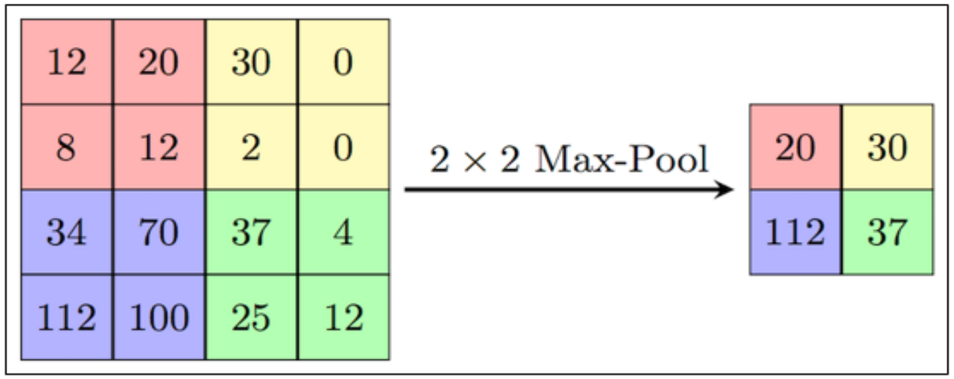


2D Pooling

Pooling

Скользим окном по входным данным
и вычисляем некоторую функцию
от элементов в окне.

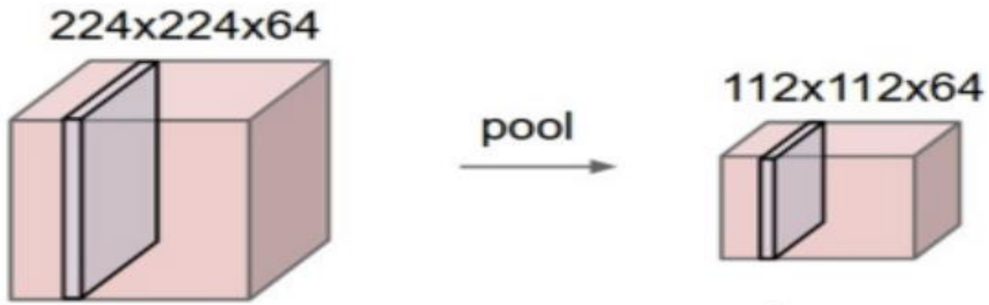
Пример



Цель: уменьшение размерности.

Применяется обычно *после свёрточного слоя*.

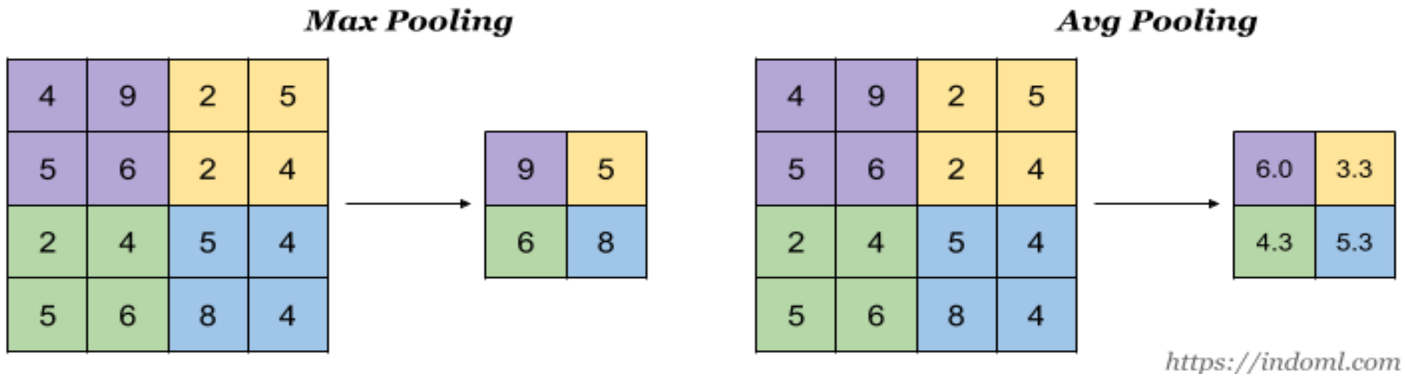
После свёртки имеем большое количество признак
для дальнейшей работы
настолько подробные признаки уже не нужны.
уменьшим количество признаков,
оставляя важную информацию.



Pooling

Гиперпараметры:

- **Размер окна**. Обычно 2
- **Stride.** Шаг, с которым будем перемещать окно. Обычно 2.
- **Функция**, которую будем применять к элементам в окне.
  1. Max (самый популярный вариант)
  2. Average
  3. Sum



Pooling. Цели

- **Увеличиваем receptive field** следующих слоев.

- **Уменьшаем изображение**, чтобы следующие свёрточные слои оперировали над большей областью исходного изображения.

- **Ускорение вычислений** за счет уменьшения количества признаков.
  Такая фильтрация помогает не переобучаться, т.к. выкидываем часть информации.

- **Помогаем модели понять был ли найден паттерн**.

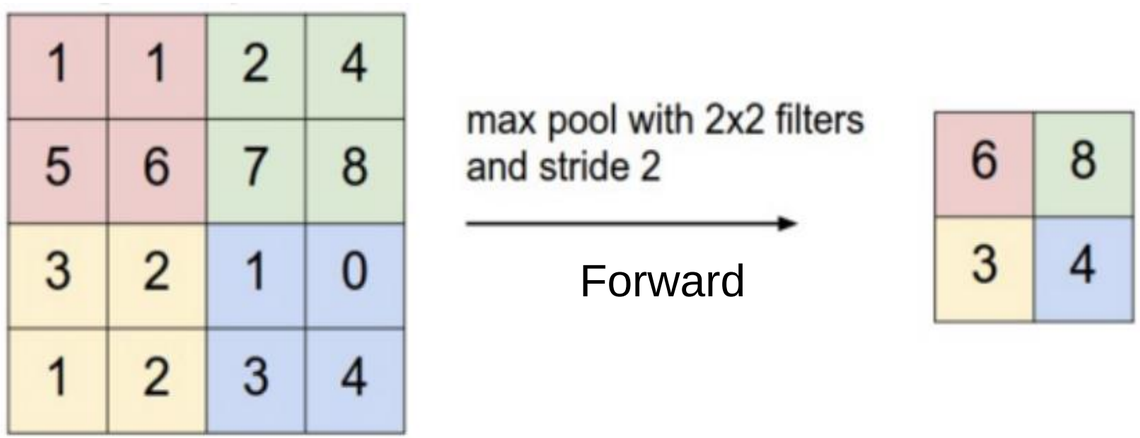
Значения в картах сверки отвечают за схожесть с паттерном.
При взятии максимального значения в окне, оставляем максимальную схожесть, встреченную в этом окне. Нам не важны остальные значения,
по максимуму можно понять был ли встречен паттерн в данном окне.

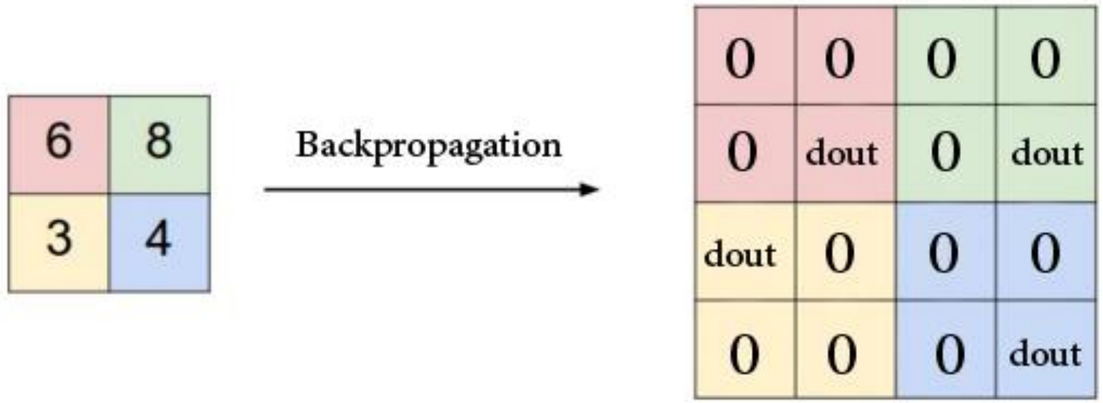
Помогли сети, выкинув ненужную информацию.
Применение максимума в данной ситуации более логично,
именно поэтому это самый популярный вариант.

Pooling. Backpropagation

Функция потерь не зависит от значений,
не выбранных в качестве максимума.

Градиент по ним будет равен 0.
Т.е. градиент потечет назад
только через значения,
выбранные в качестве максимумов.





ВСЁ!