Введение в компьютерное зрение. Сверточные сети

Лектор — Троешестова Лидия

Классификация изображений

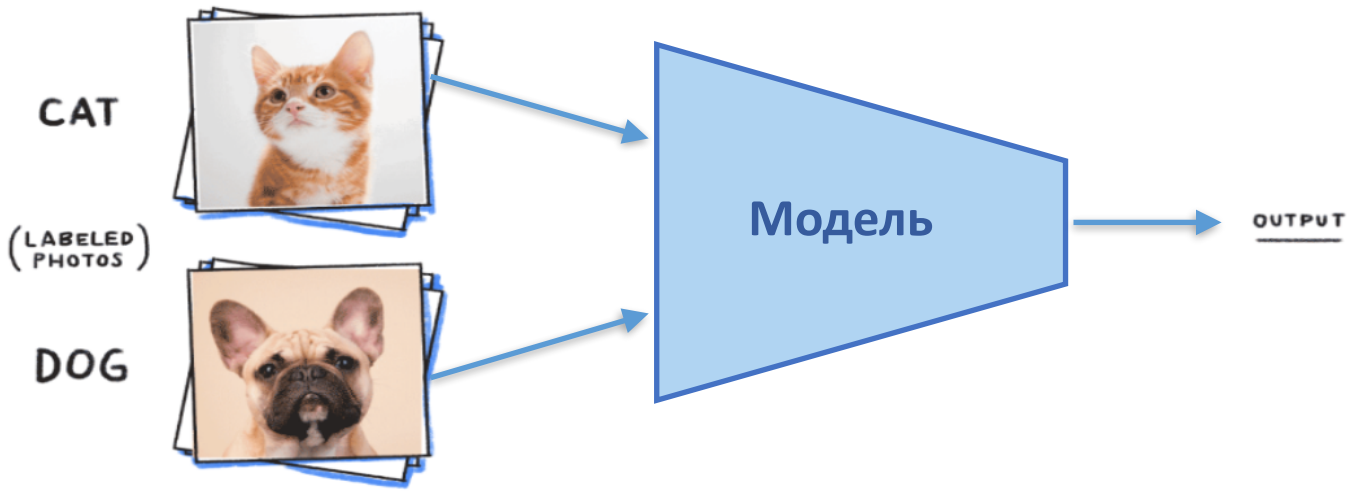
Классификация изображений

X— пространство картинок,

Y — набор классов, например {кошка, собака}.

Требуется построить модель $f: X \to Y$,

определяющую к какому классу относится объект на картинке.



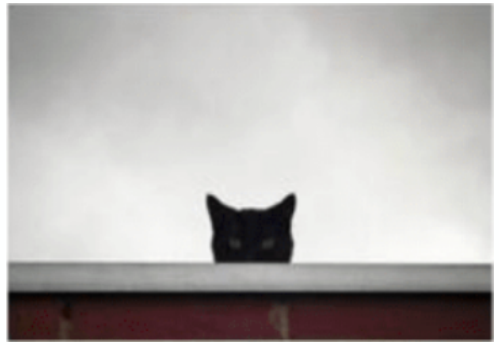
Проблемы классификации изображений

Разные углы обзора

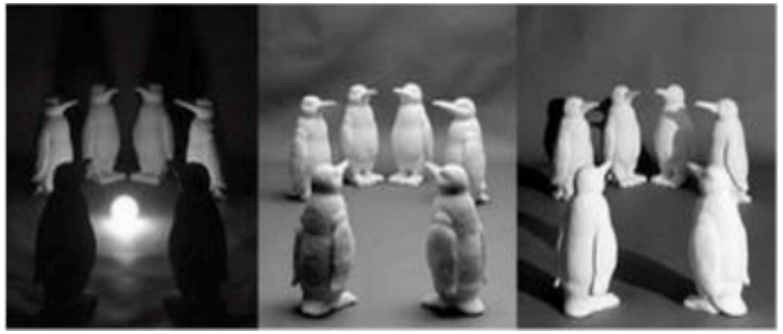
Разный размер

Деформация

Перекрытие



Разная освещенность



Фон. помехи

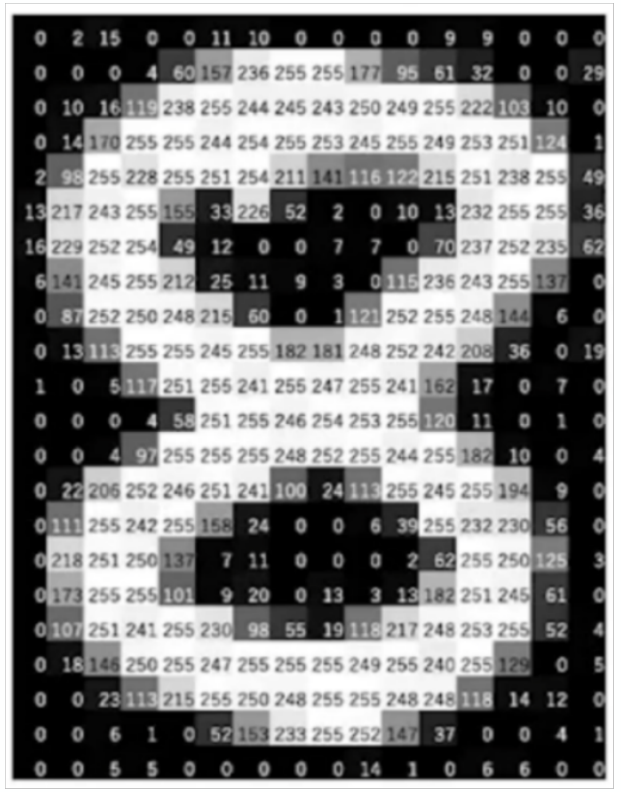
Разная форма

Стандартное представление 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.

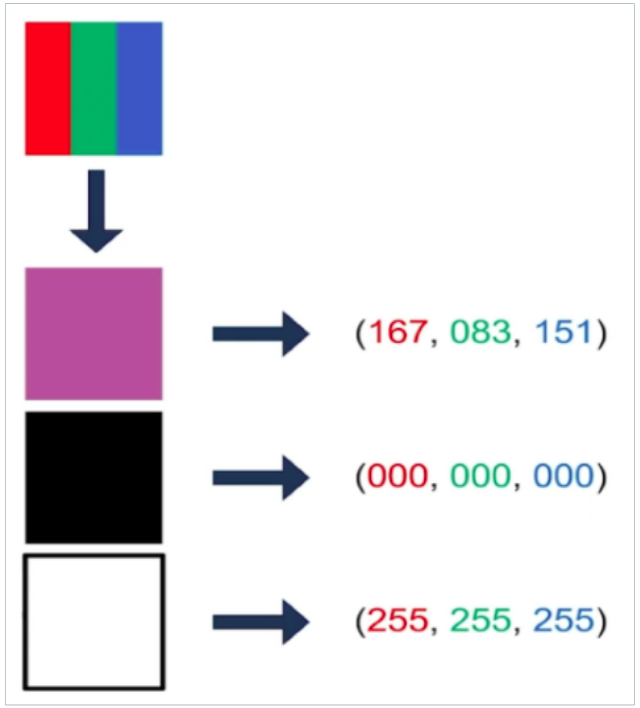
Стандартное представление изображения

- **Цветная картинка** — матрица из пикселей,

  H — высота 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).

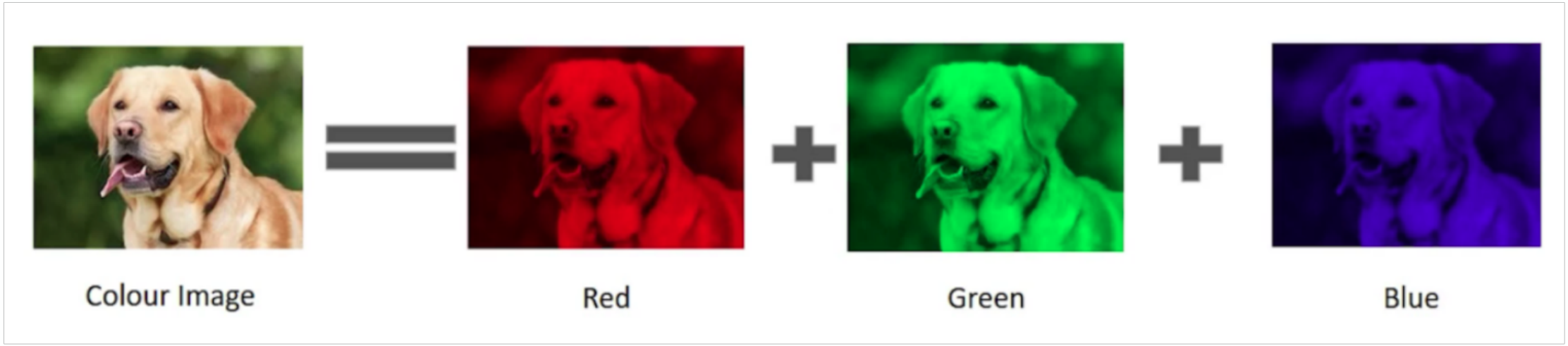


Цветная картинка — тензор размера (H, W, 3), состоящий из uint8 чисел.

Стандартное представление изображения

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

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



Классификация изображений до нейросетей

Необходима генерация признаков изображений вручную.

Генерация признаков

Полезные признаки изображения для упрощения классификации

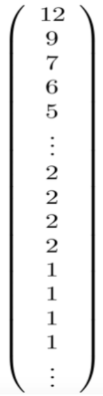
Обучение модели

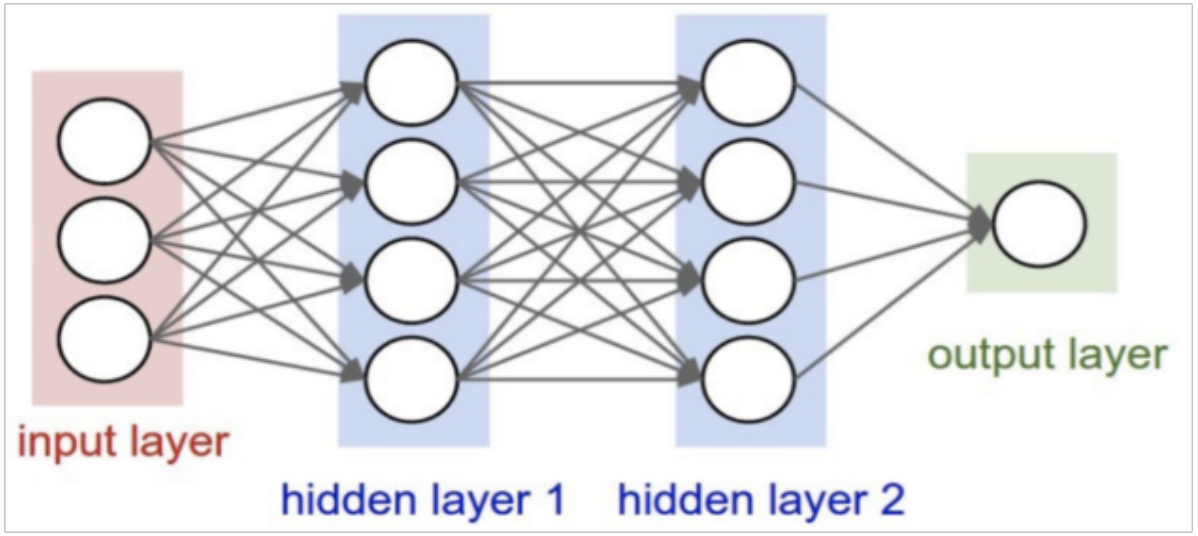
KNN, …

Предсказание

Классификация изображений с помощью нейросетей

Преобразуем картинку в вектор и подаем результат нейросети.

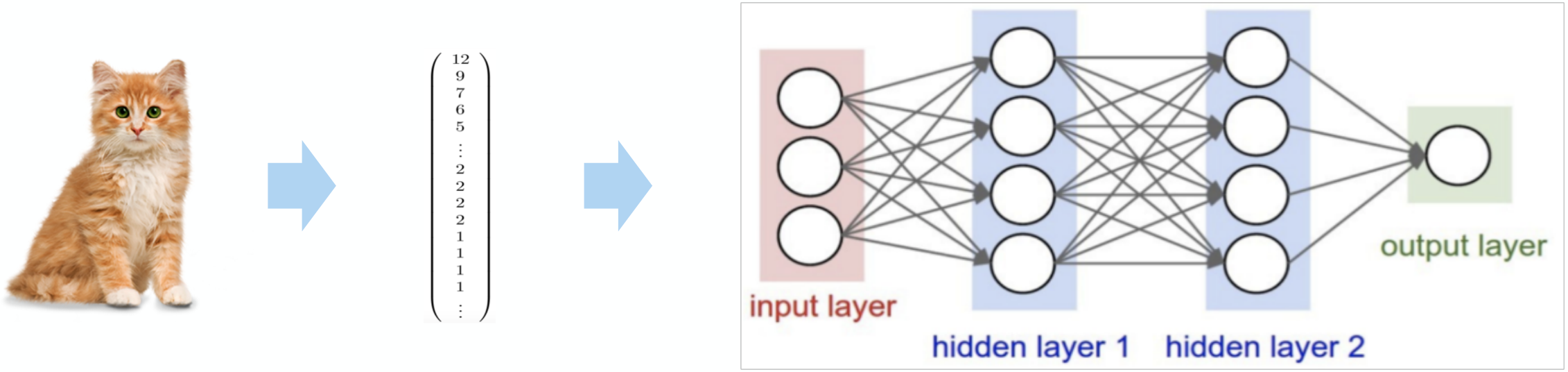




Насколько Linear слои подходят для решения задачи классификации картинок?

Классификация изображений с помощью нейросетей

Преобразуем картинку в вектор и подаем результат нейросети.



Проблемы линейных слоев:

- имеют очень много параметров
- изначально не понимают связи между близко расположенными пикселями, что делает нахождение паттернов сложнее

Свёртка (convolution)

Интуиция свёртки

Поиграем в классификацию

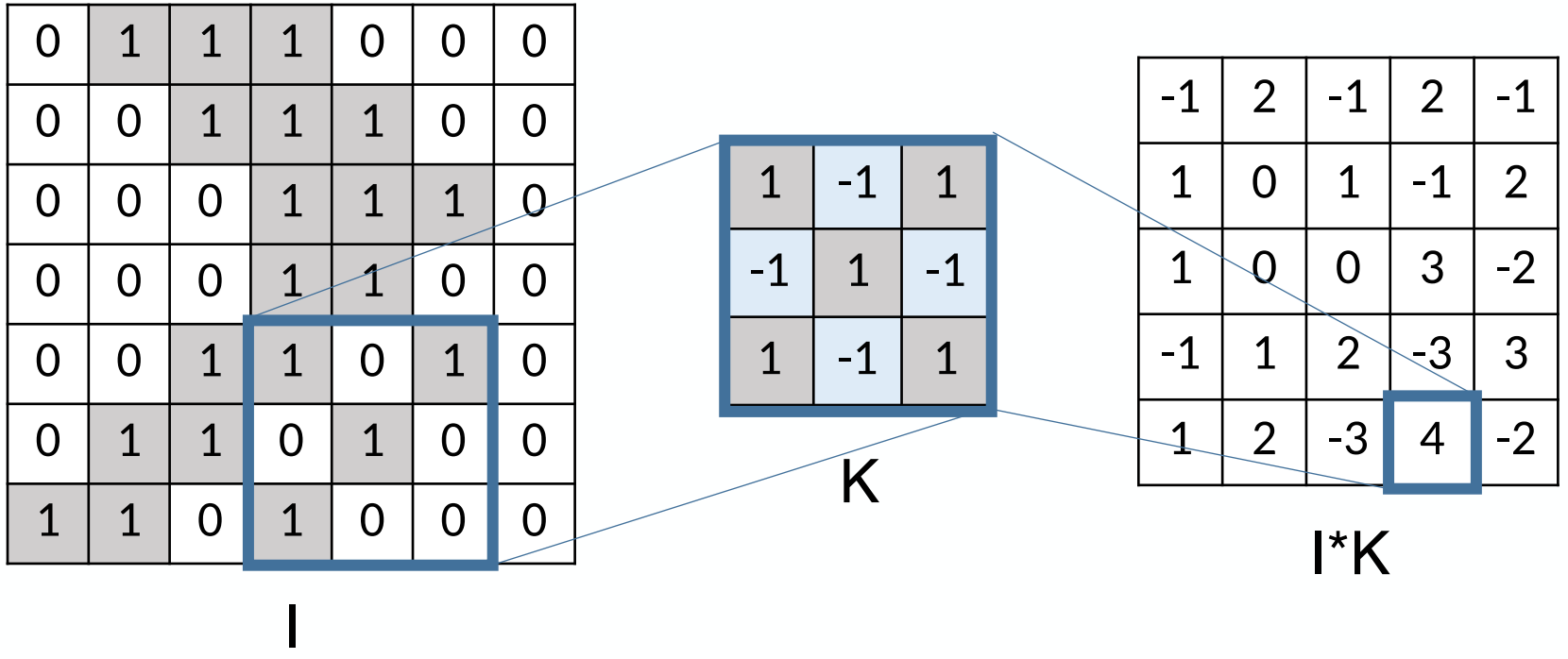
Требуется уметь находить **крестики и нолики** на картинке.
Причем объекты могут находиться в *разных местах* картинки,
быть *не в точности равны* искомым паттернам.

Интуиция свёртки

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.



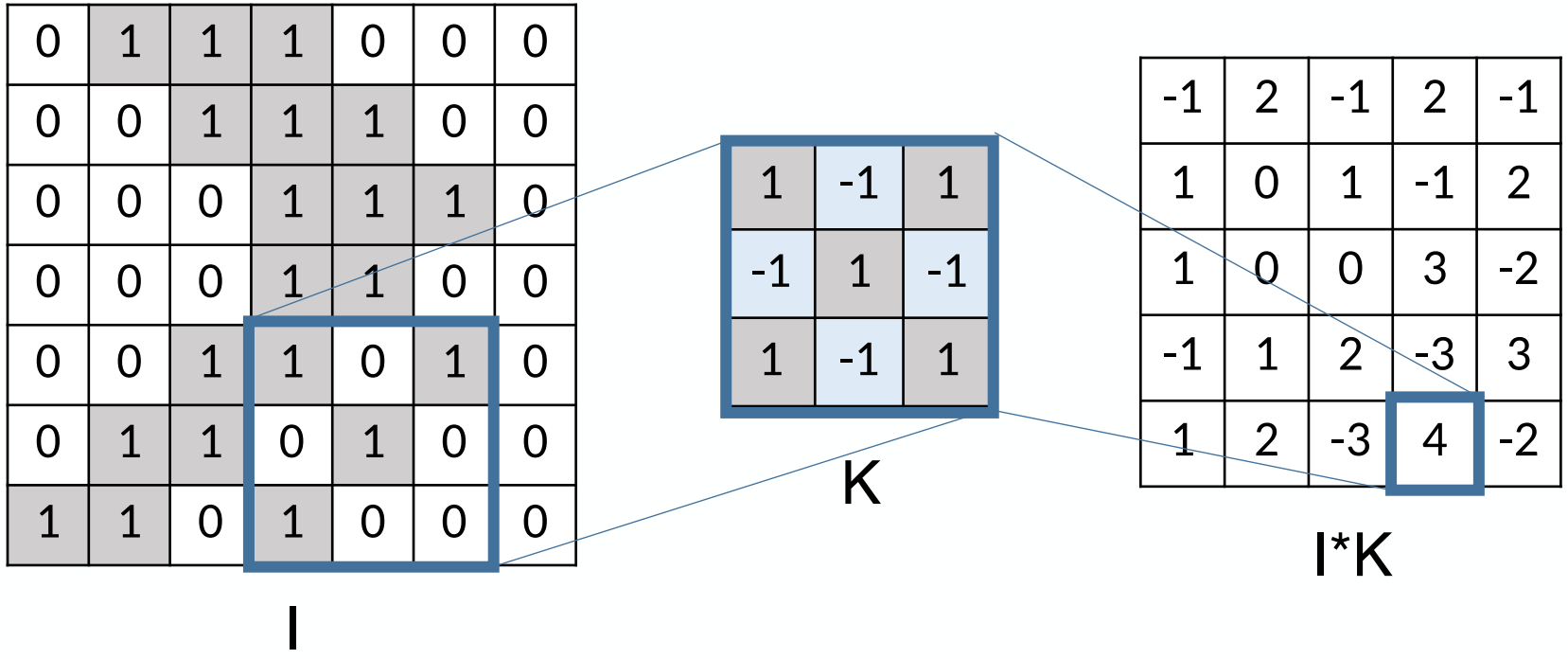
Интуиция свёртки

Как считать схожесть?

Чтобы оценить схожесть двух паттернов перемножим их скалярно:

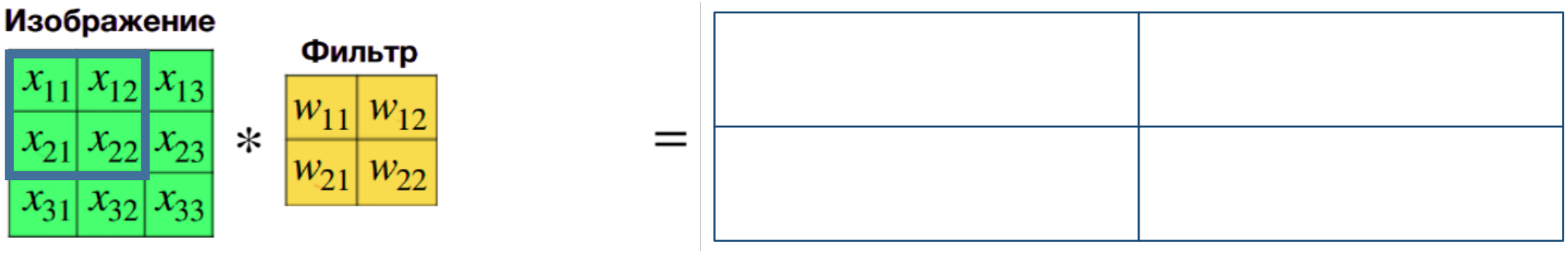
- умножим матрицы поэлементно;
- сложим числа получ. матрицы.

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



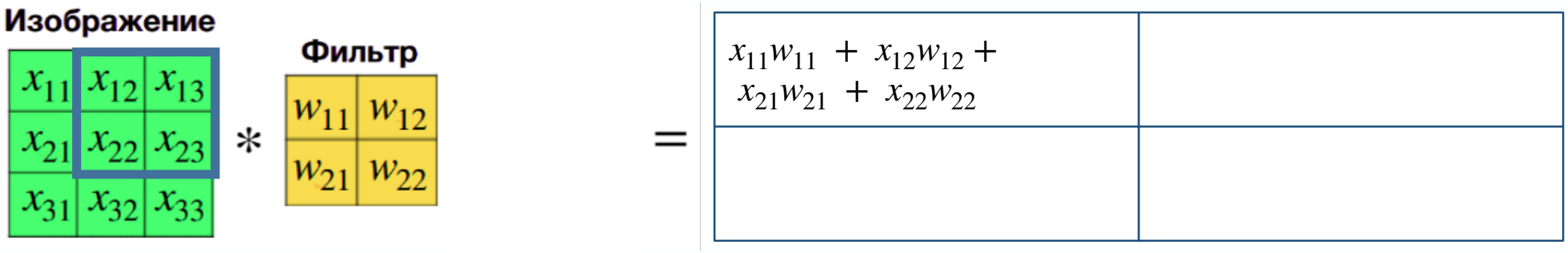
2D-свёртка

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



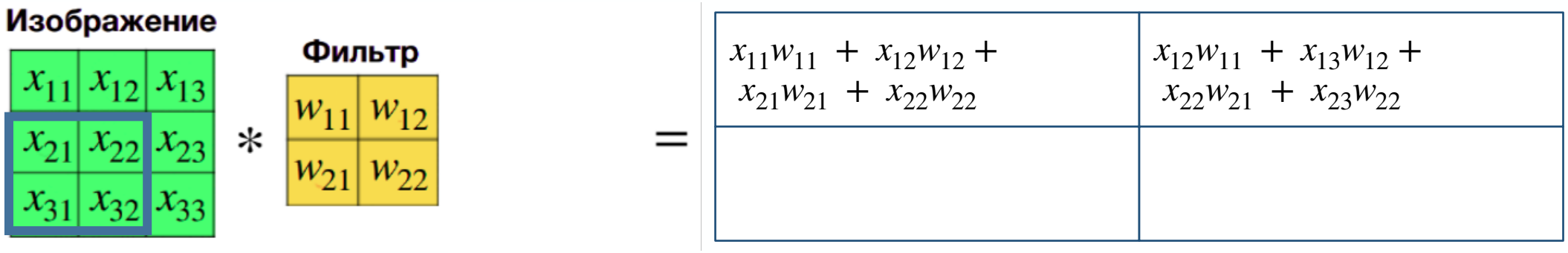
2D-свёртка

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



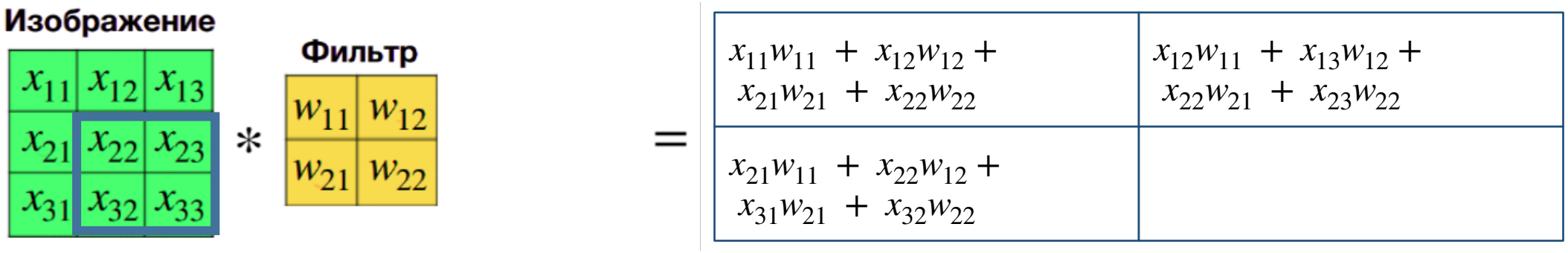
2D-свёртка

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



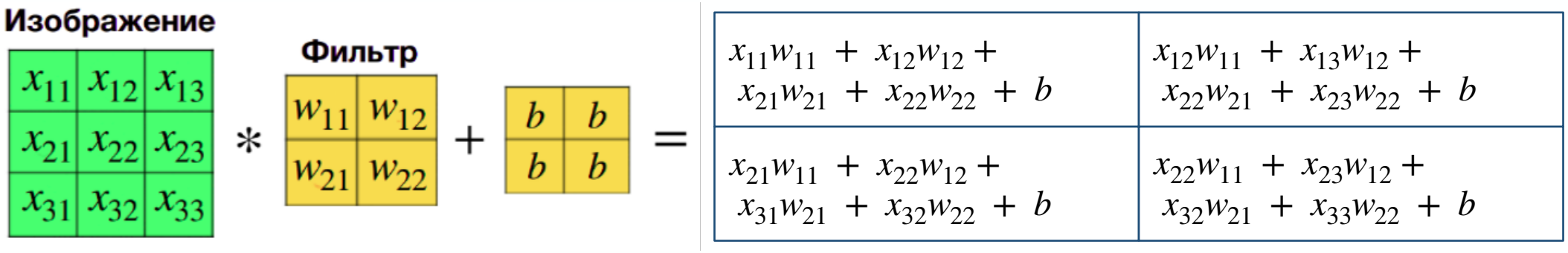
2D-свёртка

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-свёртка

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



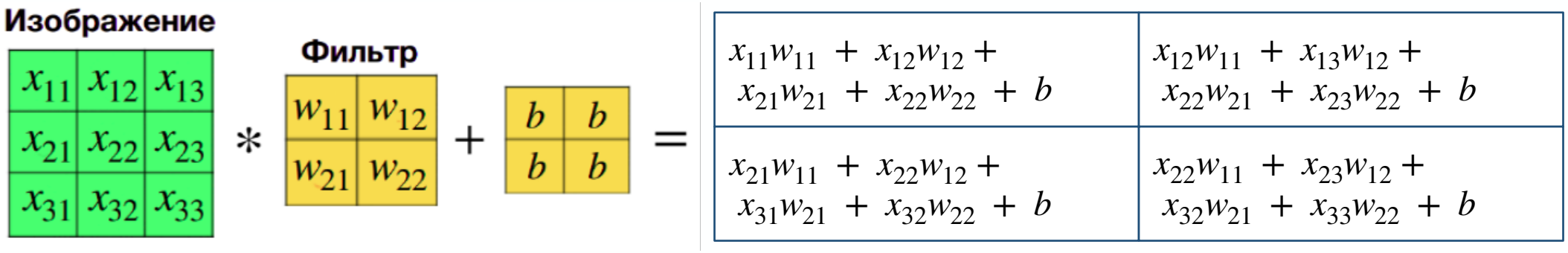
После чего к каждому элементу также добавляется смещение b.

2D-свёртка

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

где I - изображение, F - фильтр размера (K, K).

Веса w_{ij} и смещение b — обучаемые параметры. K— гиперпараметр.



2D-свёртка: а как обучать?

Несложно увидеть, что эта операция линейна по отношению к w_{ij} и b. А это значит, что градиенты операции свертки выражается аналитически, то есть Back Propagation работает!

Изображение

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}

$*$

Фильтр

w_{11}	w_{12}
w_{21}	w_{22}

$+$

b	b
b	b

$=$

$x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22} + b$	$x_{12}w_{11} + x_{13}w_{12} + x_{22}w_{21} + x_{23}w_{22} + b$
$x_{21}w_{11} + x_{22}w_{12} + x_{31}w_{21} + x_{32}w_{22} + b$	$x_{22}w_{11} + x_{23}w_{12} + x_{32}w_{21} + x_{33}w_{22} + b$

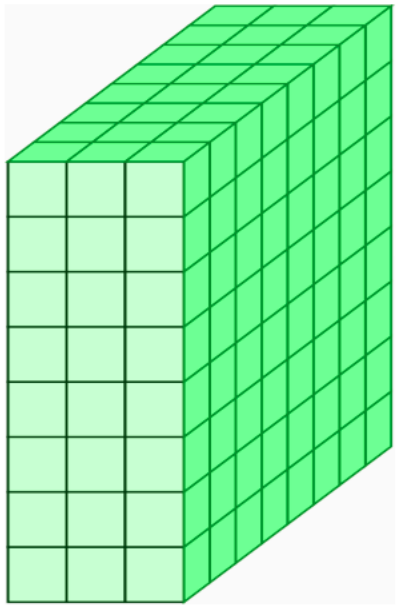
$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial X} =$$

Fun fact: градиент свертки – тоже своего рода свертка.

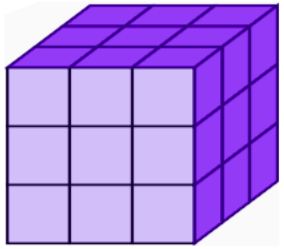
2D-свёртка. Многоканальный вход

Вход — трехмерный тензор размера $H \times W \times C$.

Ядро — трехмерная матрица размера $K \times K \times C$.

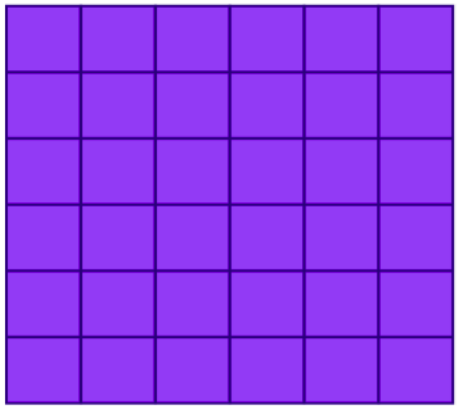


изображение 8 x 8 x 3



фильтр 3 x 3 x 3

смещение



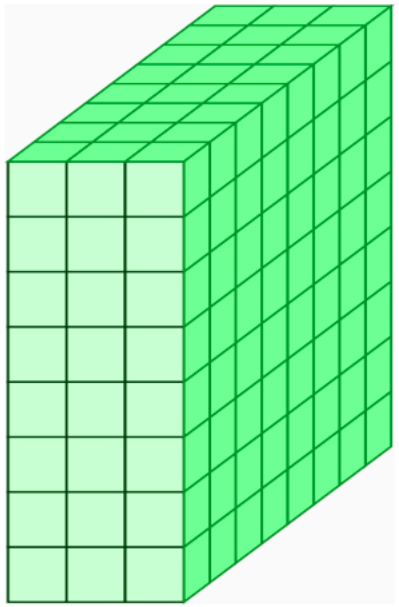
результат — карта 6 x 6

2D-свёртка. Многоканальный вход

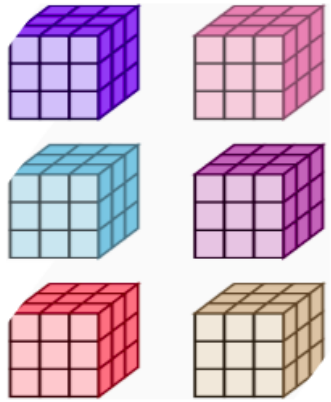
Один фильтр — одна карта, соответствующая одному паттерну.

Возьмем K разных фильтров и применим свертку к ним.

Получим K карт на выходе.

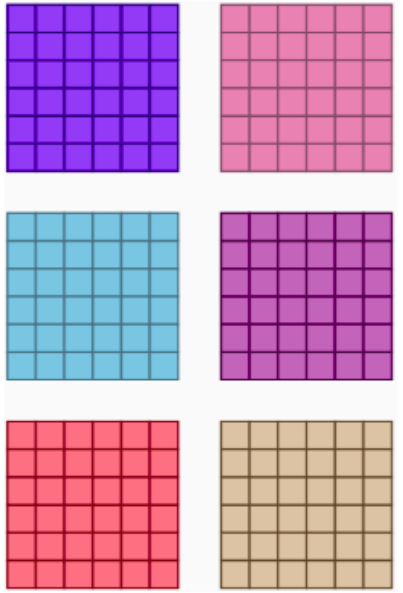


изображение 8 x 8 x 3



6 фильтров 3 x 3 x 3

смещения



6 карт 6 x 6

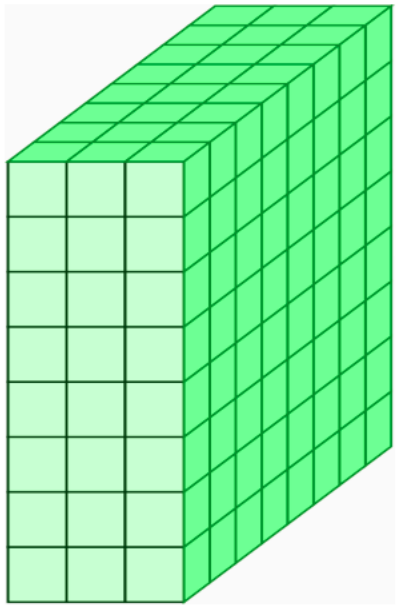
2D-свёртка. Многоканальный вход

Один фильтр — одна карта, соответствующая одному паттерну.

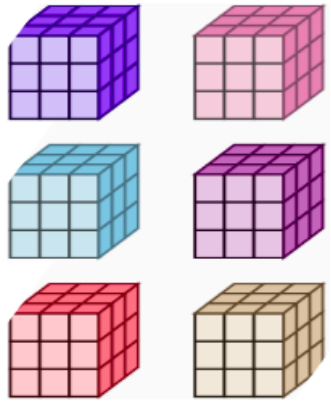
Возьмем K разных фильтров и применим свертку к ним.

Получим K карт на выходе.

Выходную матрицу можно использовать как вход следующего слоя.

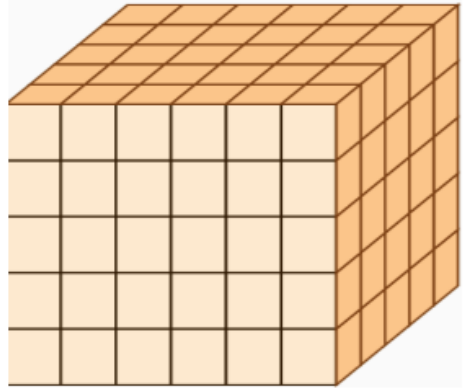


изображение 8 x 8 x 3



6 фильтров 3 x 3 x 3

смещения



6 карт 6 x 6

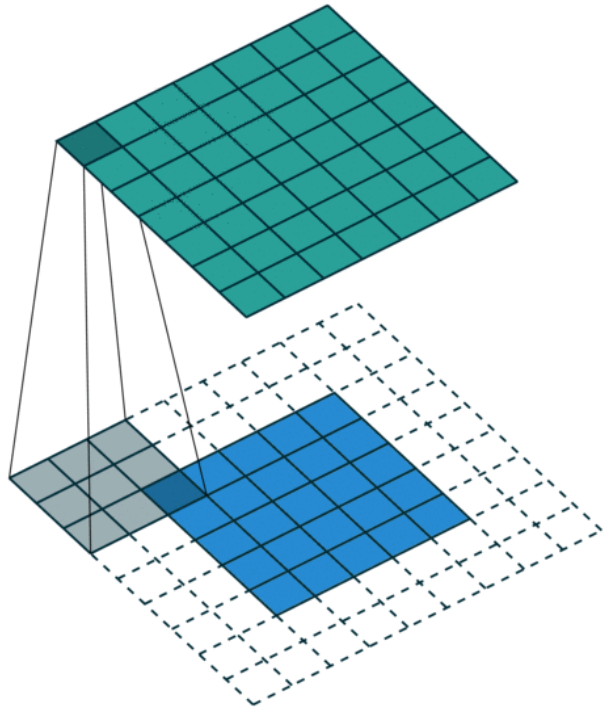
2D-свёртка: padding

Проблема:

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

Padding добавляет по краям фейковые пиксели.

Тогда все пиксели поучаствуют во всех позициях ядра :)



Какими значениями заполнять?

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

https://github.com/vdumoulin/conv_arithmetic

2D-свёртка: stride

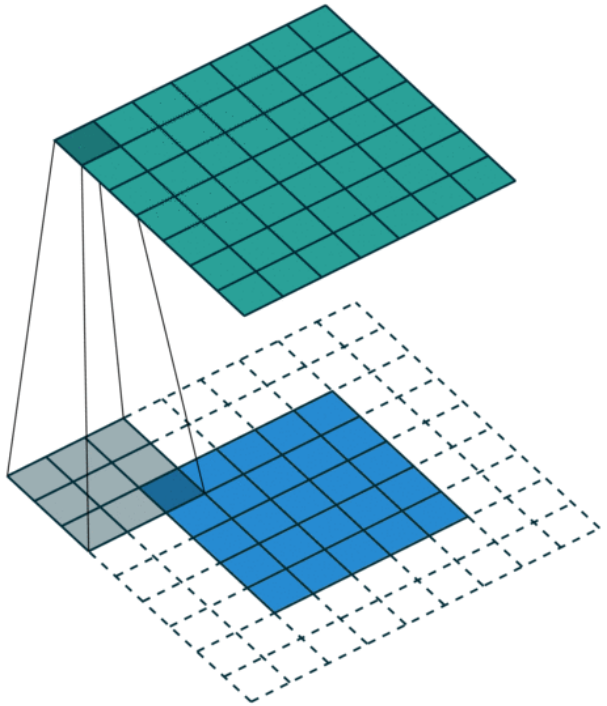
Идея: перемещаем ядро с некоторым дискретным шагом.

- stride = 1 (default) - ядро сдвигается на 1 пиксель.
- stride = 2 - ядро сдвигается на 2 пикселя.

Таким образом, выходной размер уменьшается.

Искомые паттерны всё равно должны найтись,

если исходное изображение достаточно большое.



Больше визуализаций:

https://ezyang.github.io/convolution-visualizer/

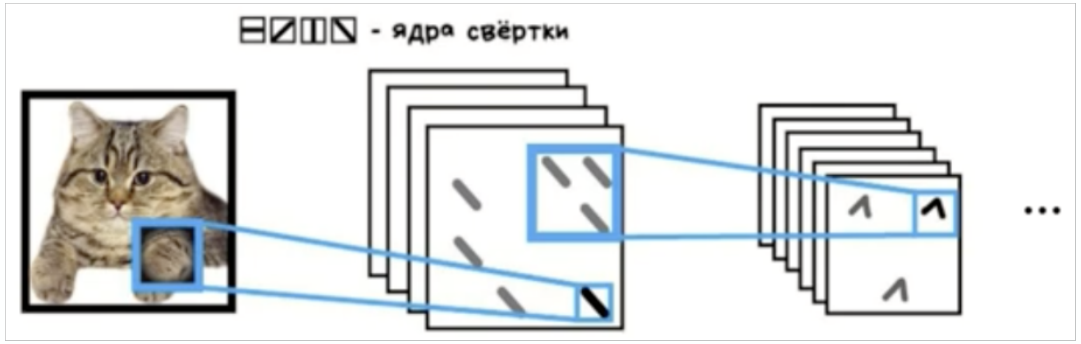
https://github.com/vdumoulin/conv_arithmetic

Больше свёрток!

- 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.
- …



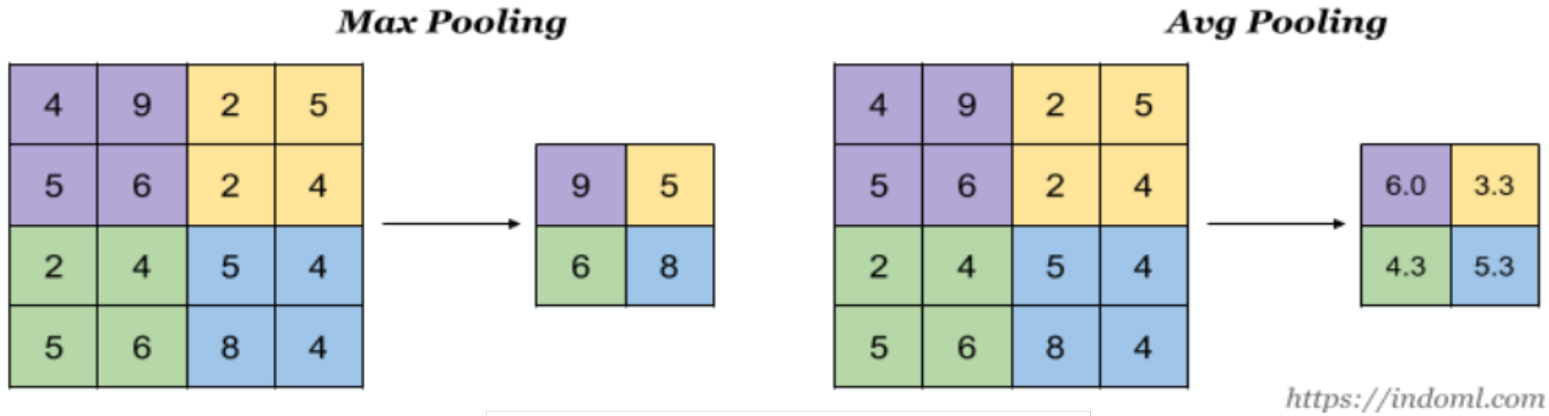
Pooling

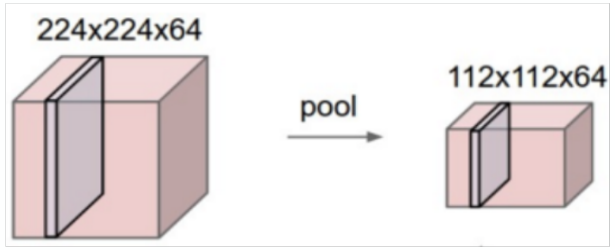
2D Pooling

Скользим окном по входным данным и вычисляем нек. функцию от его элементов.

Цель — уменьшение размерности. Применяется обычно после свёрточного слоя.

После свёртки имеем большое количество признаков, для дальнейшей работы настолько подробные признаки уже не нужны. Уменьшим количество признаков, оставив только самое важное.





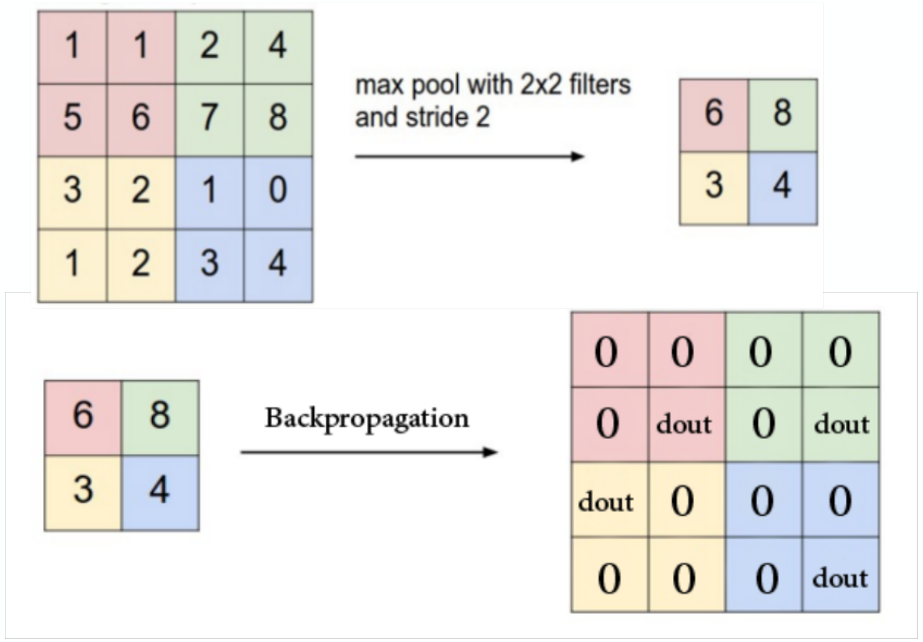
2D Pooling

Гиперпараметры

- kernel_size: часто берут 2
- stride: шаг, с которым будем перемещать окно, часто берут 2
- Функция, которую применяем к элементам в окне:
  - Max
  - Average
  - …

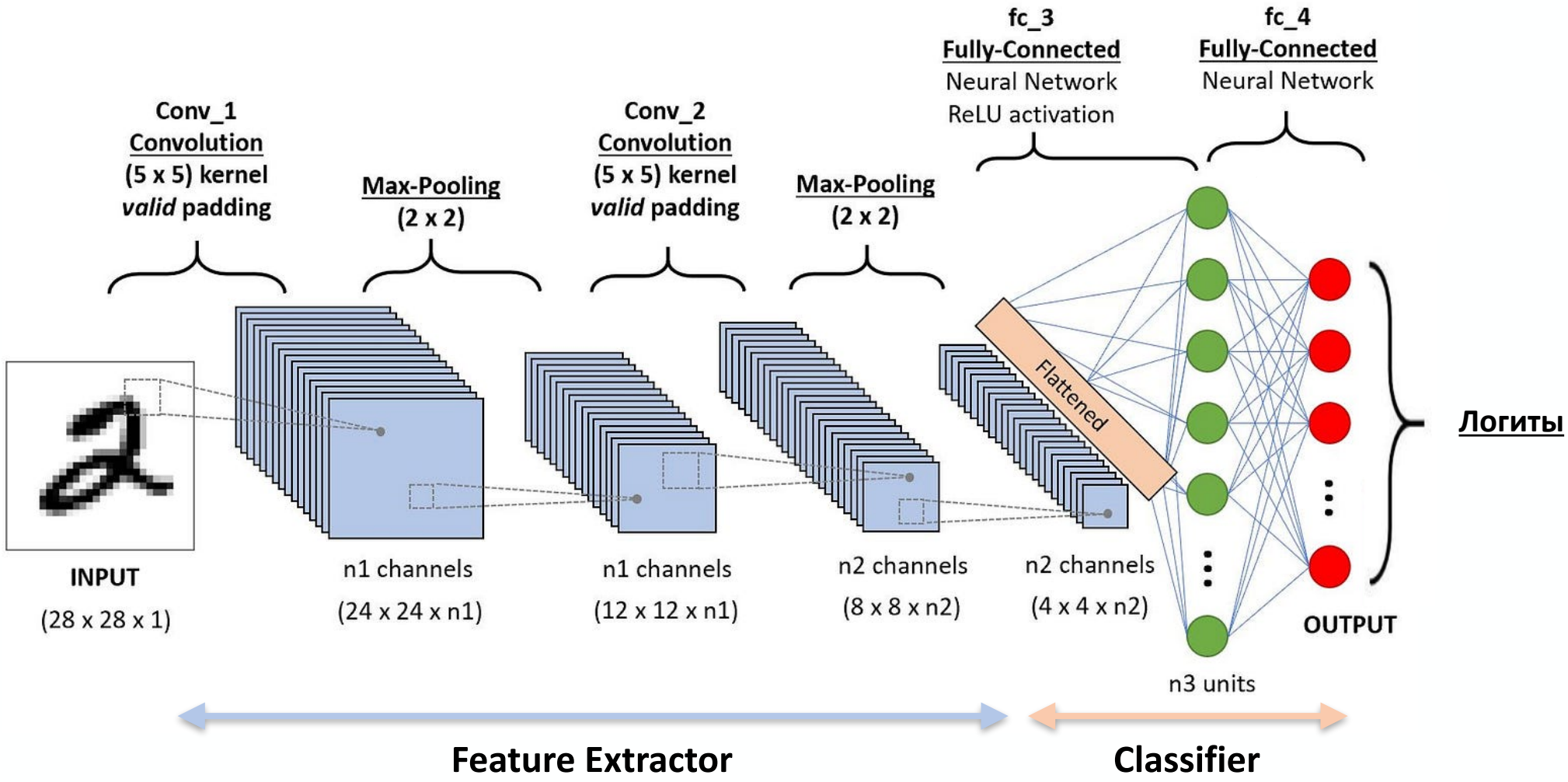
Back propagation (MaxPooling):

Градиент потечет назад только через значения, выбранные в качестве максимумов.



Классификация с помощью CNN

Классификация с помощью CNN

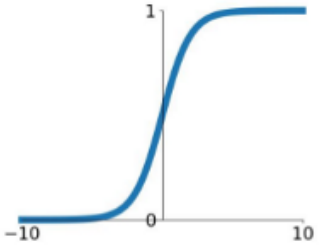


Предсказание вероятности

Бинарная классификация

Нужно нормализовать выход сети на отрезок $[0, 1]$, для этого используется **сигмоида.**

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Мультиклассовая классификация

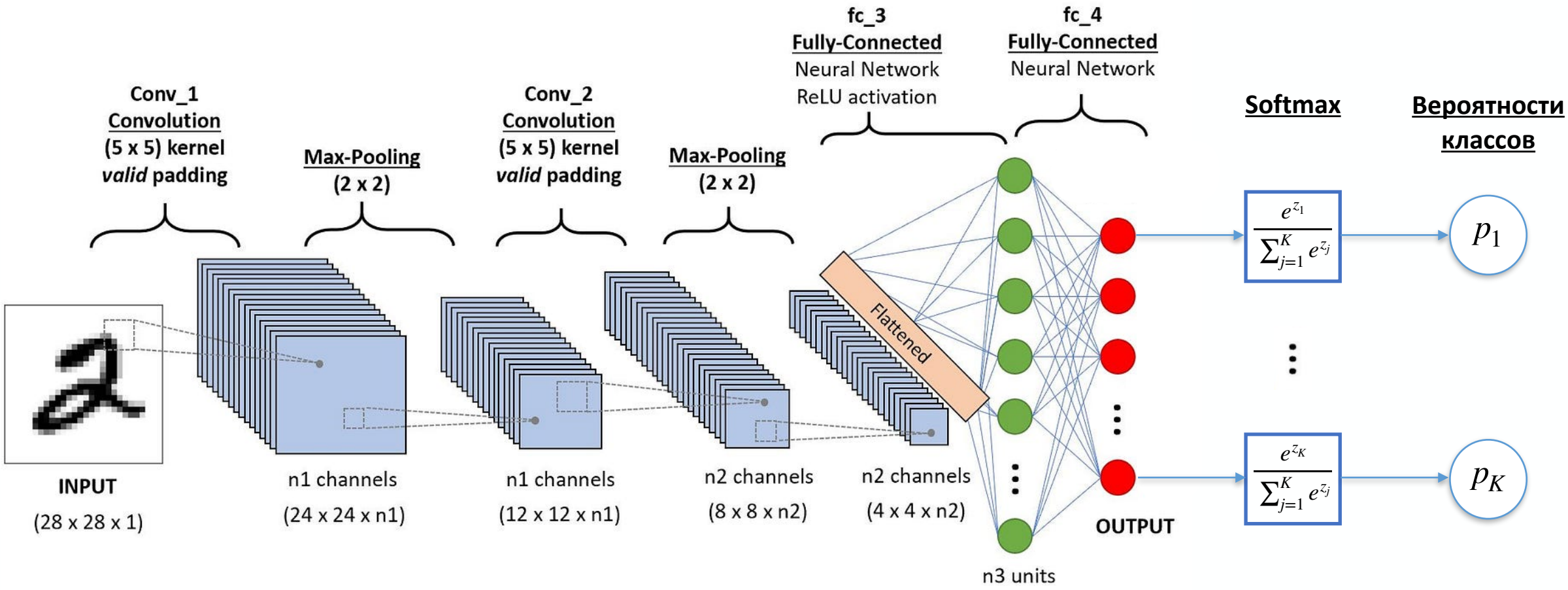
Классы не зависят друг от друга, по сути имеем бинарную классификацию для каждого класса в отдельности. Чтобы отнормировать значения выходов модели на отрезок применяется функция softmax.

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}},\ k \in \{1, \ldots, K\},\ K -$$ число классов

Почему бы просто не взять argmax из значений как метку класса?

Градиент будет нулевым практически везде, что не позволит сети обучаться.

Вероятности классов из логитов



Алгоритм обучения

1. Forward Pass:
   - вход X – батч картинок (n, H, W, C) – подается в сеть с параметрами θ;
   - выход $\hat{y}_\theta(X)$ – вероятности (n, K);
   - Y – one-hot представление истинных меток класса (n, K).

2. Loss: для многоклассовой классификации используется Cross Entropy Loss:

$$\text{CE}(Y, \hat{y}_\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} Y_{ij}\log(\hat{y}_\theta(X_i))_j$$

3. Backward Pass: считаем градиенты $\nabla_\theta \text{CE}(Y, \hat{y}_\theta)$ в порядке от последних к первым слоям.

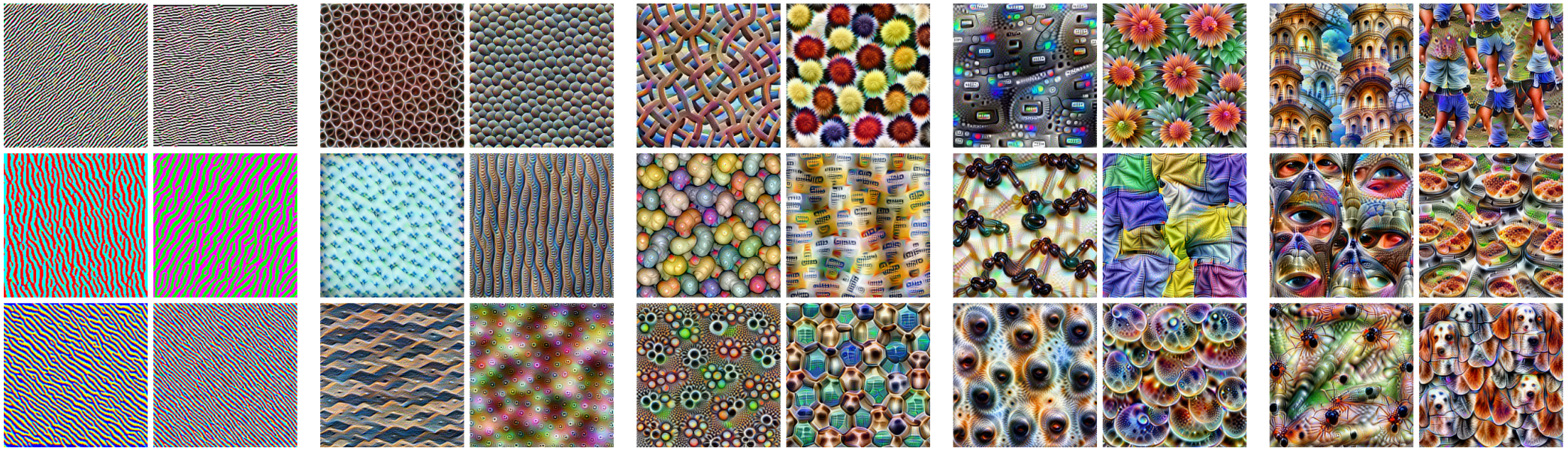
4. Шаг оптимизации: $\theta_t = \theta_{t-1} - \eta \nabla_{\theta_{t-1}} \text{CE}(Y, \hat{y}_{\theta_{t-1}})$

Чему учатся CNN?

Пусть X – вход, Z_{kl} – выход нейрона l на слое k после функции активации.

Так как CNN дифференцируема по своим входам, можно решать задачу $Z_{kl} \longrightarrow \max_X$.

Результат – картинка, больше всего активирующая данный нейрон.



Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

https://distill.pub/2017/feature-visualization/

ВСЁ!