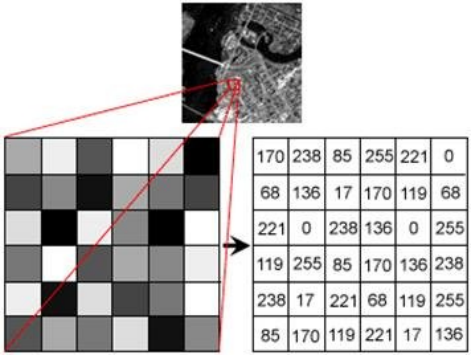
Лекция 3. Сверточные нейронные сети

Бабич Павел Васильевич

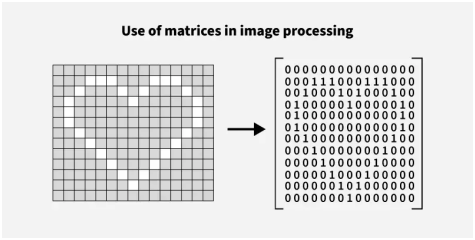
ИММиКН

2026

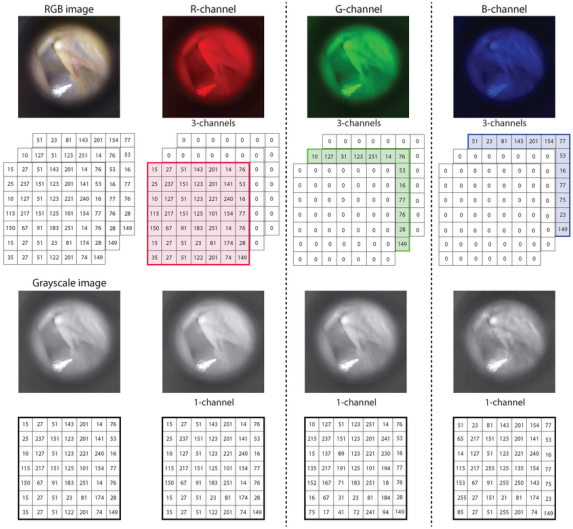
Изображение как матрица



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136



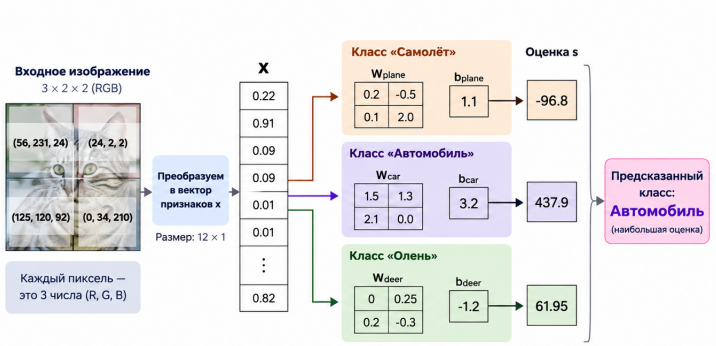
Изображение как матрица



Наивная классификация

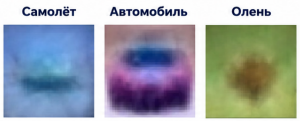


Наивная классификация



Вектора весов можно «прорисовать» в изображение

Каждая матрица весов **W** показывает, какие цвета и формы важны для класса. Если преобразовать веса обратно в изображение, мы увидим «шаблон» класса.



Это интерпретация модели: видно, «как выглядят» классы с точки зрения весов.

Линейная модель классификации: $\mathbf{s = Wx + b}$

$s_i = w_i^T x + b_i$ — линейная оценка для класса i

Минутка кода

Как будет выглядеть линейный классификатор для MNIST

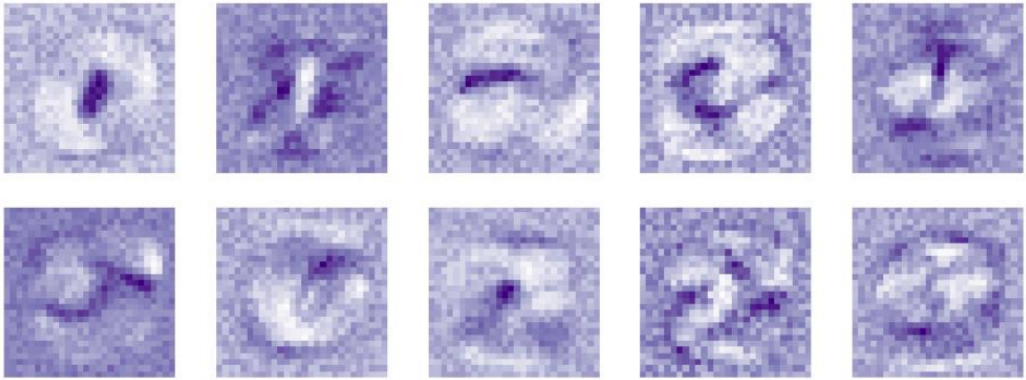
```
class NaiveImageClassifier(nn.Module):
    def __init__(self):
        super(Mlinear, self).__init__()
        self.conv = nn.Conv2d(in_channels=1, out_channels=10, kernel_size
            =28)
        # self.fc = nn.Linear(28*28, 10)

    def forward(self, x, verbose=False):
        x = self.conv(x)
        x = x.view(-1, 10)
        x = F.log_softmax(x, dim=1)
        return x
```

Есть ли тут разница между `nn.Linear(28*28, 10)` и `nn.Conv2d(in_channels=1, out_channels=10, kernel_size=28)`?

Минутка кода

Можно визуализировать матрицы весов обученных на MNIST моделей



Математические основы свёртки

Свёртка позволяет вычислять локальные признаки изображения с помощью небольшого ядра (фильтра).

Для изображения $X \in \mathbb{R}^{H \times W}$ и ядра $K \in \mathbb{R}^{k \times k}$:

$$Y_{i,j} = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} K_{u,v} \, X_{i+u,j+v}.$$

Y – **скалярное произведение** ядра и локального фрагмента изображения:

$$Y_{i,j} = \langle K, \, X_{i:i+k,\, j:j+k} \rangle \, .$$

Свёртка в нейронной сети

Для цветного изображения учитываются все входные каналы:

$$X \in \mathbb{R}^{H \times W \times C_{\text{in}}}, \qquad K \in \mathbb{R}^{k \times k \times C_{\text{in}} \times C_{\text{out}}}.$$

$$Y_{i,j,m} = b_m + \sum_{c=1}^{C_{\text{in}}} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} K_{u,v,c,m} \, X_{i+u,j+v,c}.$$

После нелинейности получаем карту признаков:

$$Z_{i,j,m} = \sigma(Y_{i,j,m}), \qquad \sigma(x) = \max(0, x) \quad \text{(ReLU)}.$$

При шаге s и паддинге p размер выхода равен

$$H_{\text{out}} = \left\lfloor \frac{H + 2p - k}{s} \right\rfloor + 1, \qquad W_{\text{out}} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1.$$

Пример свёртки

Например, ядро для выделения вертикальных границ:

$$K = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}.$$

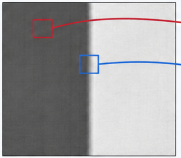
Основная идея

Одно и то же ядро K применяется во всех областях изображения: **локальность** + **разделение весов**.

Пример работы свёртки: обнаружение вертикальных границ

Входное изображение
(оттенки серого, 6 × 6)

50	50	50	200	200	200
50	50	50	200	200	200
50	50	50	200	200	200
50	50	50	200	200	200
50	50	50	200	200	200
50	50	50	200	200	200



Как выглядит изображение
(чёрный ≈ 0, белый ≈ 255)

Ядро свёртки
(вертикальные границы)

−1	0	1
−1	0	1
−1	0	1

Реагирует на вертикальные изменения интенсивности: слева тёмное – справа светлое

Случай 1: однородная область (свёртка даёт 0)

Локальный фрагмент (3 × 3) | Произведение поэлементно | Сумма

			⊙				=				Σ = 0
50	50	50		−1	0	1		−50	0	50	
50	50	50		−1	0	1		−50	0	50	
50	50	50		−1	0	1		−50	0	50	

$$\Sigma = -50 + 0 + 50 - 50 + 0 + 50 - 50 + 0 + 50 = 0$$

В однородной области интенсивность не меняется, поэтому отклик свёртки равен нулю.

Случай 2: вертикальная граница (свёртка даёт большое значение)

Локальный фрагмент (3 × 3) | Произведение поэлементно | Сумма

			⊙				=				Σ = 450
50	200	200		−1	0	1		−50	0	200	
50	200	200		−1	0	1		−50	0	200	
50	200	200		−1	0	1		−50	0	200	

$$\Sigma = -50 + 0 + 200 - 50 + 0 + 200 - 50 + 0 + 200 = 450$$

На вертикальной границе слева тёмные пиксели, справа светлые, поэтому отклик свёртки большой (**положительный**).

Свёртка вычисляет скалярное произведение ядра и локального фрагмента изображения, что позволяет выделять вертикальные границы: сильный отклик на резких изменениях, нулевой – на однородных областях.

Отступ (padding) в 2D свёртке

Отступ — это добавление дополнительных пикселей по краям изображения, которое позволяет управлять размером выходной карты признаков.

Изображение (вход)

Размер: 4 × 4 (оттенки серого)

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



Визуализация изображения

Размер: 4 × 4

Ядро свёртки (3 × 3)

1	0	−1
1	0	−1
1	0	−1

(например, выделение вертикальных границ)

Без отступа (valid)

$p = 0$

Дополнение не добавляется. Свёртка применяется только в тех положениях, где ядро полностью помещается в изображение.

Вход (без отступа)

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

Результат свёртки

−1	3
1	−2

Размер выхода: 2 × 2

Нулевой отступ (zero padding)

$p = 1$

По краям добавляются нули. Позволяет сохранить пространственный размер (при $k = 3$ и $s = 1$).

Вход с отступом ($p = 1$)

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

Результат свёртки

−3	−2	1	5
−2	−2	0	3
0	−1	−1	0
3	2	2	−4

Размер выхода: 4 × 4 (такой же, как у входа)

Константный отступ

$p = 1$, значение = 1

По краям добавляется заданная константа (например, 1).

Вход с отступом ($p = 1$, $c = 1$)

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

Результат свёртки

−2	−2	2	7
−3	−2	0	6
−1	−1	−1	3
3	2	2	−1

Размер выхода: 4 × 4

Зеркальный отступ

$p = 1$

По краям добавляются значения из изображения (отражение относительно границы).

Вход с отступом ($p = 1$, отражение)

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

Результат свёртки

0	−2	1	3
−3	−2	0	4
0	−1	−1	1
0	2	2	−3

Размер выхода: 4 × 4

Размер выходной карты признаков:

$$H_{out} = \left\lfloor \frac{H + 2p - k}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1$$

H, W — размер входного изображения
k — размер ядра (фильтра)
p — размер отступа (padding)
s — шаг (stride)

Зачем нужен отступ?

- позволяет контролировать размер выходной карты признаков;
- сохранять пространственное разрешение;
- уменьшать потерю информации на границах изображения.

Параметр stride в свертках

Stride (шаг) определяет, на сколько пикселей сдвигается окно свертки при каждом шаге.

Пример: stride = 2

Фильтр 3 × 3, шаг 2. Окно сдвигается на 2 пикселя.

Входное изображение (7 × 7)

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

Шаг 2 вправо

Шаг 2 вниз

Выходная карта признаков (3 × 3)

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

Размер выходной карты признаков:

$$H_{out} = \left\lfloor \frac{H_{in} - K}{S} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - K}{S} \right\rfloor + 1$$

Для $H_{in} = W_{in} = 7$, $K = 3$, $S = 2$:

$$H_{out} = W_{out} = \left\lfloor \frac{7 - 3}{2} \right\rfloor + 1 = 3$$

Пример: stride = 3

Фильтр 3 × 3, шаг 3. Окно сдвигается на 3 пикселя.

Входное изображение (7 × 7)

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

Шаг 3 вправо

Шаг 3 вниз

Выходная карта признаков (2 × 2)

y_{11}	y_{12}
y_{21}	y_{22}

Размер выходной карты признаков:

$$H_{out} = \left\lfloor \frac{H_{in} - K}{S} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - K}{S} \right\rfloor + 1$$

Для $H_{in} = W_{in} = 7$, $K = 3$, $S = 3$:

$$H_{out} = W_{out} = \left\lfloor \frac{7 - 3}{3} \right\rfloor + 1 = 2$$

Вывод: чем больше stride, тем сильнее уменьшается пространственный размер выходной карты признаков, так как окно свертки движется быстрее и пропускает больше позиций.

1

Расширение (dilation) в свертках

Dilation (расширение) задаёт расстояние между элементами фильтра, не увеличивая его размер.
При dilation = D соседние элементы фильтра применяются к элементам входа, отстоящим на D позиций.

Пример: dilation = 2

Фильтр 3 × 3, dilation = 2. Элементы фильтра берутся с шагом 2 по обеим осям, покрывая область 5 × 5 на входе.

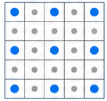
Входное изображение (7 × 7)

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

Фильтр 3 × 3 (dilation = 2)

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

Покрываемая область на входе (5 × 5)



Эффективный размер фильтра при dilation:

$$K_{eff} = K + (K-1)(D-1)$$

Для $K = 3$, $D = 2$: $K_{eff} = 3 + (3-1)(2-1) = 5$

Пример: dilation = 3

Фильтр 3 × 3, dilation = 3. Элементы фильтра берутся с шагом 3 по обеим осям, покрывая область 7 × 7 на входе.

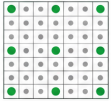
Входное изображение (7 × 7)

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48

Фильтр 3 × 3 (dilation = 3)

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

Покрываемая область на входе (7 × 7)



Эффективный размер фильтра при dilation:

$$K_{eff} = K + (K-1)(D-1)$$

Для $K = 3$, $D = 3$: $K_{eff} = 3 + (3-1)(3-1) = 7$

Вывод: при dilation = D элементы фильтра применяются к входу с шагом D по каждой оси, поэтому покрываемая область (receptive field) увеличивается до $K_{eff} = K + (K-1)(D-1)$.

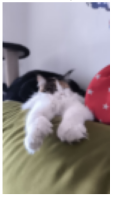
Свертка в PyTorch

```python
import torch
import torch.nn as nn

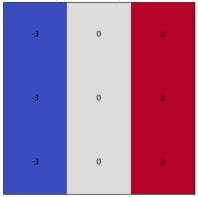
layer = nn.Conv2d(
    in_channels=3,
    out_channels=10,
    kernel_size=3,
    stride=2,
    padding=1,
    dilation=1,
    bias=True
)

x = torch.randn(1, 3, 256, 256)
y = layer(x)
print(y.shape)  # torch.Size([?, ?, ?, ?])
```

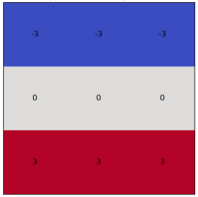
Исходное изображение



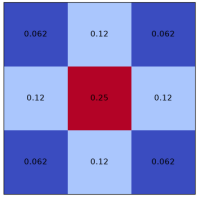
Вертикальные границы



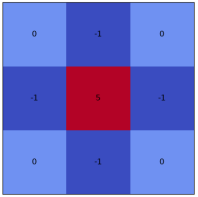
Горизонтальные границы

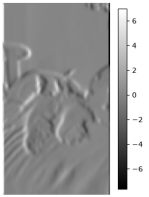


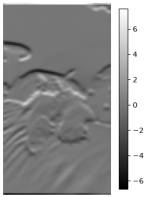
Размытие

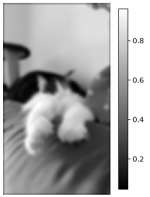


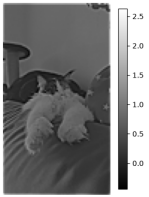
Резкость



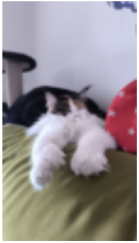




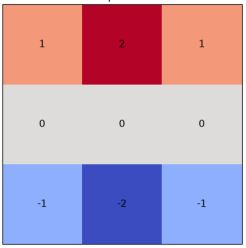




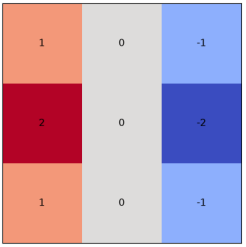
Исходное изображение



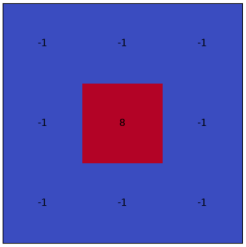
top sobel

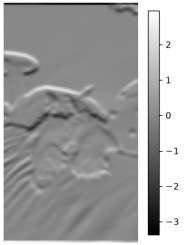


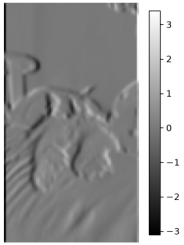
left sobel

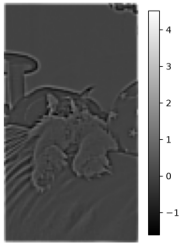


outline

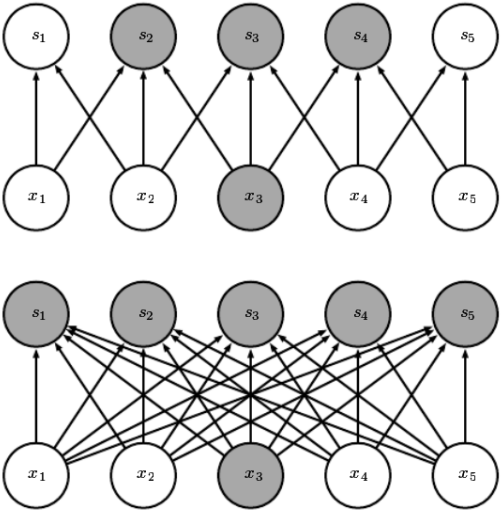




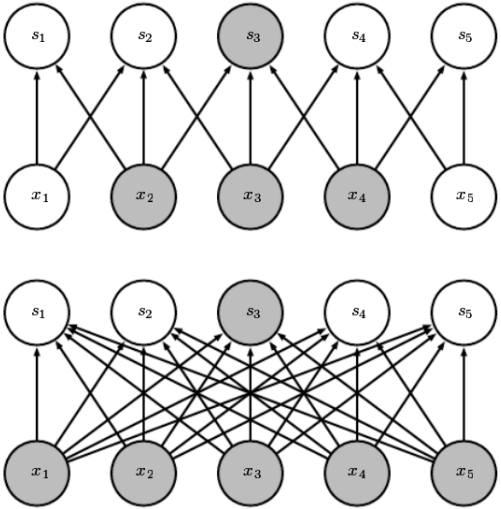




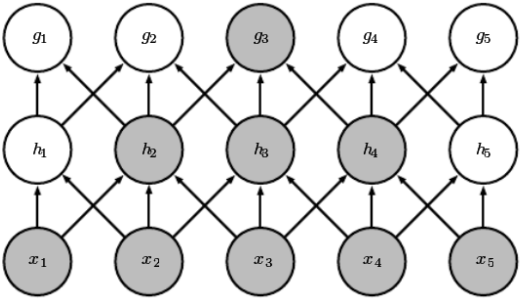
Сравнение сверточных и полносвязных сетей



Сравнение сверточных и полносвязных сетей



Сравнение сверточных и полносвязных сетей



https://www.deeplearningbook.org/contents/convnets.html

Рецептивное поле (receptive field)

Рецептивное поле (RF) — это область во входе, которая влияет на одно значение выходной карты признаков.
Оно характеризует, сколько информации из исходного изображения «видит» нейрон.

Определение и формулы

Рецептивное поле (RF)

Рецептивное поле R_l — это размер области входного изображения, влияющей на одно значение выхода слоя l.

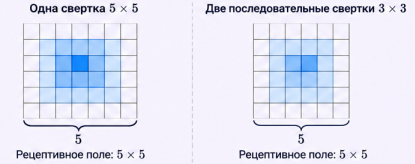
Для последовательности сверток ($\text{stride} = 1$):

$$R_l = R_{l-1} + (k_l - 1) \prod_{i=1}^{l-1} s_i,$$

где k_l — размер ядра l-го слоя, s_i — его шаг.

При $s_i = 1$ для всех слоев: $R_l = 1 + \sum_{i=1}^{l} (k_i - 1)$.

Интуитивная иллюстрация



Вариант 1: одна свертка 5 × 5



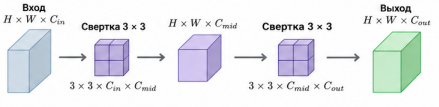
Рецептивное поле

$$R_1 = 1 + (5 - 1) = 5 \quad \Rightarrow \quad 5 \times 5$$

Число параметров

$$P_{5\times5} = 5 \cdot 5 \cdot C_{in} \cdot C_{out} = 25 \cdot C_{in} \cdot C_{out}$$

Вариант 2: две последовательные свертки 3 × 3



Рецептивное поле

$$R_1 = 1 + (3 - 1) = 3$$
$$R_2 = R_1 + (3 - 1) = 5 \quad \Rightarrow \quad 5 \times 5$$

Число параметров

$$P_{3\times3,3\times3} = 3 \cdot 3 \cdot C_{in} \cdot C_{mid} + 3 \cdot 3 \cdot C_{mid} \cdot C_{out} = 9 \cdot (C_{in} \cdot C_{mid} + C_{mid} \cdot C_{out})$$

Сравнение и вывод

- Две последовательные свертки 3×3 и одна свертка 5×5 имеют **одинаковое receptive field:** 5×5.
- **Число параметров** во втором случае **меньше** при любом $C_{in} > 0, C_{out} > 0$, если $C_{mid} < \frac{25}{18} \frac{C_{in} C_{out}}{C_{in} + C_{out}}$.
- Чаще всего берут $C_{mid} = C_{out} = C$, тогда $P_{3\times3,3\times3} = 18 \cdot C^2 < 25 \cdot C^2 = P_{5\times5}$.
- Последовательные небольшие свертки позволяют увеличивать receptive field эффективнее по параметрам и вносят дополнительные нелинейности.

Max Pooling (максимальное подвыборка)

Max pooling — операция понижения пространственного разрешения, которая разбивает вход на непересекающиеся (или частично перекрывающиеся) окна и для каждого окна выбирает максимальное значение.

Определение

Для входа $X \in \mathbb{R}^{H \times W \times C}$, окна размера $k \times k$ и шага s, max pooling вычисляет

$$Y_{i,j,c} = \max_{(u,v) \in \mathcal{W}_{i,j}} X_{u,v,c},$$

где $\mathcal{W}_{i,j} = \{(u,v) \mid u \in [is, is+k-1], \; v \in [js, js+k-1]\}$.

Размер выходной карты признаков:

$$H_{out} = \left\lfloor \frac{H-k}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W-k}{s} \right\rfloor + 1.$$

Операция применяется независимо к каждому каналу (глубине).

Пример 1: окно 2 × 2, шаг 2

Вход: $X \in \mathbb{R}^{4 \times 4}$ (один канал)

1	3	2	0
0	5	1	4
2	1	3	1
0	2	1	6

Выход: $Y \in \mathbb{R}^{2 \times 2}$

5	4
2	6

$$H_{out} = \left\lfloor \frac{4-2}{2} \right\rfloor + 1 = 2$$

$$W_{out} = \left\lfloor \frac{4-2}{2} \right\rfloor + 1 = 2$$

$$Y_{0,0} = \max(1,3,0,5) = 5$$
$$Y_{0,1} = \max(2,0,1,4) = 4$$
$$Y_{1,0} = \max(2,1,0,2) = 2$$
$$Y_{1,1} = \max(3,1,1,6) = 6$$

Пример 2: окно 3 × 3, шаг 2

Вход: $X \in \mathbb{R}^{5 \times 5}$ (один канал)

1	3	2	4	0
0	5	1	2	1
2	1	3	0	2
4	0	1	6	3
1	2	0	1	4

Выход: $Y \in \mathbb{R}^{2 \times 2}$

5	4
4	6

$$H_{out} = \left\lfloor \frac{5-3}{2} \right\rfloor + 1 = 2$$

$$W_{out} = \left\lfloor \frac{5-3}{2} \right\rfloor + 1 = 2$$

$$Y_{0,0} = \max(X_{0:2,0:2}) = 5$$
$$Y_{0,1} = \max(X_{0:2,2:4}) = 4$$
$$Y_{1,0} = \max(X_{2:4,0:2}) = 4$$
$$Y_{1,1} = \max(X_{2:4,2:4}) = 6$$

Свойства max pooling

- Уменьшает пространственный размер: $H_{out}, W_{out} < H, W$
- Выбирает наиболее активные признаки (инвариантность к небольшим сдвигам).
- Не имеет обучаемых параметров.
- Применяется независимо по каналам:

$$Y_{i,j,c} = \max_{(u,v) \in \mathcal{W}_{i,j}} X_{u,v,c},$$

- Обратное распространение градиента проходит только к элементу, который был максимальным (остальные получают 0).

Обобщение на прозвольный размер и шаг

Для входа $X \in \mathbb{R}^{H \times W \times C}$, окна $k \times k$ и шага s:

$$Y_{i,j,c} = \max_{0 \le u < k, 0 \le v < k} X_{is+u, js+v, c}.$$

Размер выходной карты признаков:

$$H_{out} = \left\lfloor \frac{H-k}{s} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W-k}{s} \right\rfloor + 1, \quad C_{out} = C.$$

Частный случай. Если $s = k$, окна не перекрываются.
Если $s < k$, окна перекрываются, что даёт более плотное представление.

Mean Pooling (среднее подвыборка)

Mean pooling — операция понижения пространственного разрешения, при которой для каждого окна вычисляется среднее значение элементов.

Определение

Для входа $X \in \mathbb{R}^{H\times W\times C}$, окна размера $k \times k$ и шага s, mean pooling вычисляет

$$Y_{i,j,c} = \frac{1}{k^2} \sum_{(u,v)\in\mathcal{W}_{i,j}} X_{u,v,c},$$

где $\mathcal{W}_{i,j} = \{(u,v) \mid u \in [is, is+k-1], \; v \in [js, js+k-1]\}$.

Размер выходной карты признаков:

$$H_{out} = \left[\frac{H-k}{s}\right] + 1, \quad W_{out} = \left[\frac{W-k}{s}\right] + 1.$$

Операция применяется независимо к каждому каналу (глубине).

Пример 1: окно 2 × 2, шаг 2

Вход: $X \in \mathbb{R}^{4\times 4}$ (один канал)

1	3	2	0
0	5	1	4
2	1	3	1
0	2	1	6

Выход: $Y \in \mathbb{R}^{2\times 2}$

2.25	1.75
1.25	2.75

$$H_{out} = \left[\frac{4-2}{2}\right] + 1 = 2$$

$$W_{out} = \left[\frac{4-2}{2}\right] + 1 = 2$$

$$Y_{0,0} = \frac{1+3+0+5}{4} = 2.25$$

$$Y_{0,1} = \frac{2+0+1+4}{4} = 1.75$$

$$Y_{1,0} = \frac{2+1+0+2}{4} = 1.25$$

$$Y_{1,1} = \frac{3+1+1+6}{4} = 2.75$$

Пример 2: окно 3 × 3, шаг 2

Вход: $X \in \mathbb{R}^{5\times 5}$ (один канал)

1	3	2	4	0
0	5	1	2	1
2	1	3	0	2
4	0	1	6	3
1	2	0	1	4

Выход: $Y \in \mathbb{R}^{2\times 2}$

1.67	2.00
1.67	2.11

$$H_{out} = \left[\frac{5-3}{2}\right] + 1 = 2$$

$$W_{out} = \left[\frac{5-3}{2}\right] + 1 = 2$$

$$Y_{0,0} = \frac{1+3+2+0+5+1+2+1+3}{9} = 1.67$$

$$Y_{0,1} = \frac{2+4+0+1+2+1+3+0+2}{9} = 2.00$$

$$Y_{1,0} = \frac{2+1+3+4+0+1+1+2+0}{9} = 1.67$$

$$Y_{1,1} = \frac{0+2+2+1+6+3+0+1+4}{9} = 2.11$$

Свойства mean pooling

- Уменьшает пространственный размер: $H_{out}, W_{out} < H, W$.
- Усредняет признаки, что делает представление более устойчивым к локальным вариациям и шуму.
- Не имеет обучаемых параметров.
- Применяется независимо по каналам:

$$Y_{i,j,c} = \frac{1}{k^2} \sum_{(u,v)\in\mathcal{W}_{i,j}} X_{u,v,c}.$$

- Обратное распространение градиента:

если $\frac{\partial \mathcal{L}}{\partial Y_{i,j,c}} = g$, то каждый элемент окна получает

$$\frac{\partial \mathcal{L}}{\partial X_{u,v,c}} = \frac{g}{k^2}, \quad (u,v) \in \mathcal{W}_{i,j}.$$

Обобщение на произвольный размер, шаг и число каналов

Для $X \in \mathbb{R}^{H\times W\times C}$, окна $k_h \times k_w$ и шага s_h, s_w:

$$Y_{i,j,c} = \frac{1}{k_h k_w} \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} X_{is_h+u, js_w+v, c}.$$

Размер выходной карты признаков:

$$H_{out} = \left[\frac{H-k_h}{s_h}\right] + 1, \quad W_{out} = \left[\frac{W-k_w}{s_w}\right] + 1, \quad C_{out} = C.$$

Частные случаи:

- Если $s_h = k_h, s_w = k_w$, окна не перекрываются.
- Если $s_h < k_h$ или $s_w < k_w$, окна перекрываются, что даёт более плотное представление.

Разновидности операций пулинга (pooling)

Пулинг — это параметрическая или беспараметрическая операция агрегации, которая понижает пространственное разрешение, сохраняя наиболее важную информацию. Существуют различные виды пулинга, отличающиеся способом агрегации.

Max Pooling (максимальный)

Выбирает максимальное значение в окне.

$$y = \max_{(i,j)\in\mathcal{W}} x_{i,j}$$

Пример (окно 2×2):

1	3	2	0
0	5	1	4
2	1	3	1
0	2	1	6

Выход:

5	4
2	6

Mean Pooling (средний)

Вычисляет среднее значение в окне.

$$y = \frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} x_{i,j}$$

Пример (окно 2×2):

1	3	2	0
0	5	1	4
2	1	3	1
0	2	1	6

Выход:

2.25	1.75
1.25	2.75

Stochastic Pooling (стохастический)

Выбирает элемент из окна случайным образом с вероятностью, пропорциональной его значению.

$$P(x_{i,j}) = \frac{x_{i,j}}{\sum_{(u,v)\in\mathcal{W}} x_{u,v}}, \quad y = x_{i,j} \sim P$$

Пример (окно 2×2):

1	3
0	5

Вероятности выбора:

$P(1) = 1/9$

$P(3) = 3/9$

$P(0) = 0/9$

$P(5) = 5/9$

Выход: случайный элемент (например, 5) с вероятностью $5/9$.

L_p Pooling (обобщённый)

Частный случай, который включает $\max\ (p \to \infty)$ и $\text{mean}\ (p = 1)$.

$$y = \left(\frac{1}{|\mathcal{W}|} \sum_{(i,j)\in\mathcal{W}} |x_{i,j}|^p \right)^{1/p}$$

Пример ($p = 3$, окно 2×2):

1	3
0	5

$$y = \left(\frac{1^3 + 3^3 + 0^3 + 5^3}{4} \right)^{1/3} = (38)^{1/3} \approx 3.36$$

При $p \to \infty$ получаем max pooling, при $p = 1$ — mean pooling.

Min Pooling (минимальный)

Выбирает минимальное значение в окне.

$$y = \min_{(i,j)\in\mathcal{W}} x_{i,j}$$

Пример (окно 2×2):

1	3
0	5

Выход:

0

Median Pooling (медианный)

Выбирает медиану значений в окне.

$$y = \text{median}\left(\{ x_{i,j} \mid (i,j) \in \mathcal{W} \} \right)$$

Пример (окно 2×2):

1	3
0	5

Выход:

2

Global Pooling (глобальный)

Агрегирует по всему пространству признаков.

$$y = \max_{i,j} x_{i,j} \quad \text{(Global Max)}$$

$$y = \frac{1}{HW} \sum_{i,j} x_{i,j} \quad \text{(Global Avg)}$$

Пример (Global Avg):

1	2	3
4	5	6

$$y = \frac{1+2+3+4+5+6}{6} = 3.5$$

Adaptive Pooling (адаптивный)

Автоматически подбирает размер окна для получения заданного размера выхода $H_{\text{out}} \times W_{\text{out}}$.

Пример: Adaptive Avg Pooling к 1×1 (глобальное среднее):

1	2	3
4	5	6

$\longrightarrow$ 3.5

Используется, например, перед полносвязными слоями.

Fractional Max Pooling (дробный)

Использует непостоянный (дробный) шаг, задавая вероятностное разбиение на окна.

Шаги s_h, s_w могут быть нецелыми.

Окна перекрываются нерегулярно, что повышает разнообразие представлений.

Применялся в работе:
Graham, B. (2014). Fractional Max-Pooling. arXiv:1412.6071.

Сравнение методов

Метод	Агрегация	Параметры	Свойства / Особенности
Max	max	нет	Наиболее инвариантен, сохраняет сильные ответы
Mean	среднее	нет	Сглаживает шум, учитывает все значения
Stochastic	случайная выборка	нет	Вводит стохастичность, может улучшать обобщение
L_p	ℓ_p-норма	параметр p	Интерполирует между $\text{mean}\ (p = 1)$ и $\max\ (p \to \infty)$
Min	min	нет	Реже используется, чувствителен к выбросам
Median	медиана	нет	Устойчив к выбросам
Global	агрегация по всему входу	нет	Делает представление неизменным к сдвигам (глобально)
Adaptive	автоматический размер окна	нет	Позволяет задать размер выхода
Fractional	нерегулярные окна	параметры шага	Больше разнообразия, снижает переобучение

Вывод

- Разные виды пулинга реализуют различные функции агрегации и обладают разными инвариантностями.
- Max и mean pooling — наиболее распространённые базовые методы.
- Стохастический, L_p, медианный и дробный пулинг могут улучшать обобщение в определённых задачах.
- Выбор метода зависит от задачи, архитектуры и желаемых свойств представления (устойчивость, инвариантность, степень сжатия).

Max Pooling в PyTorch

```python
import torch.nn as nn

pool = nn.MaxPool2d(kernel_size=2, stride=2)

x = torch.randn(32, 64, 28, 28)
y = pool(x)
# y.shape = (32, 64, 14, 14)
```

```python
model = nn.Sequential(
    nn.Conv2d(3, 32, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
)
```

Max Pooling в PyTorch

```
import torch.nn.functional as F

y = F.max_pool2d(x, kernel_size=2, stride=2)
```

Важно

Для входа (N, C, H, W) Max Pooling применяется независимо к каждому каналу. При $k = 2$, $s = 2$:

$$(N, C, H, W) \longrightarrow (N, C, H/2, W/2).$$

Число каналов C **не изменяется**.

Backpropagation через сверточный слой

Forward pass:

$$Y_{o,i,j} = \sum_{c=1}^{C_{\text{in}}} \sum_{u,v} W_{o,c,u,v}\, X_{c,i+u,j+v} + b_o$$

где X — вход, W — ядра свертки, Y — выход.

Пусть известен градиент следующего слоя:

$$G_{o,i,j} \equiv \frac{\partial \mathcal{L}}{\partial Y_{o,i,j}}.$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial W_{o,c,u,v}} = \sum_{i,j} G_{o,i,j}\, X_{c,i+u,j+v}}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial b_o} = \sum_{i,j} G_{o,i,j}}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial X_{c,p,q}} = \sum_{o} \sum_{u,v} W_{o,c,u,v}\, G_{o,p-u,q-v}}$$

То есть градиент по входу получается сверткой G с пространственно перевернутым ядром W.

Backpropagation через Max Pooling

Для каждого pooling-окна $\mathcal{P}_{i,j}$:

$$Y_{c,i,j} = \max_{(p,q)\in\mathcal{P}_{i,j}} X_{c,p,q}.$$

$$(p^*, q^*) = \arg\max_{(p,q)\in\mathcal{P}_{i,j}} X_{c,p,q}.$$

$$G_{c,i,j} \equiv \frac{\partial \mathcal{L}}{\partial Y_{c,i,j}}.$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial X_{c,p,q}} = \sum_{i,j} G_{c,i,j}\, \mathbf{1}[(p,q) = (p^*, q^*)_{c,i,j}]}$$

Градиент проходит **только через элемент, который был максимальным** во время forward pass. Все остальные элементы pooling-окна получают нулевой градиент.

Backpropagation через Mean Pooling

Для pooling-окна $\mathcal{P}_{i,j}$ из K элементов:

$$Y_{c,i,j} = \frac{1}{K} \sum_{(p,q)\in\mathcal{P}_{i,j}} X_{c,p,q}.$$

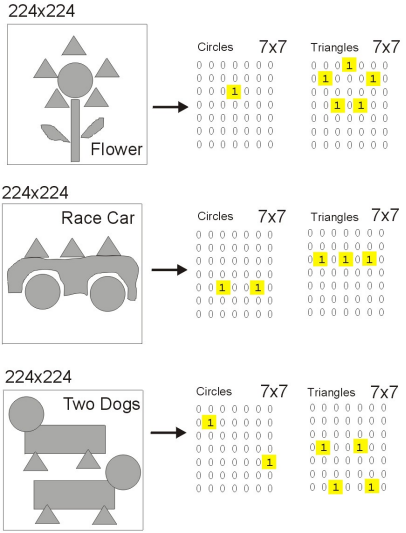
$$G_{c,i,j} \equiv \frac{\partial \mathcal{L}}{\partial Y_{c,i,j}}.$$

$$\frac{\partial Y_{c,i,j}}{\partial X_{c,p,q}} = \frac{1}{K}, \qquad (p,q) \in \mathcal{P}_{i,j},$$

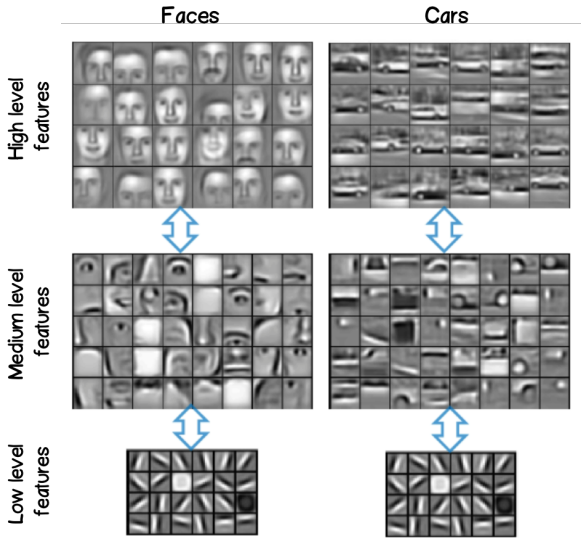
$$\boxed{\frac{\partial \mathcal{L}}{\partial X_{c,p,q}} = \sum_{(i,j):\,(p,q)\in\mathcal{P}_{i,j}} \frac{G_{c,i,j}}{K}}$$

В отличие от Max Pooling, градиент распределяется **равномерно между всеми элементами** pooling-окна.

Пирамида фичей



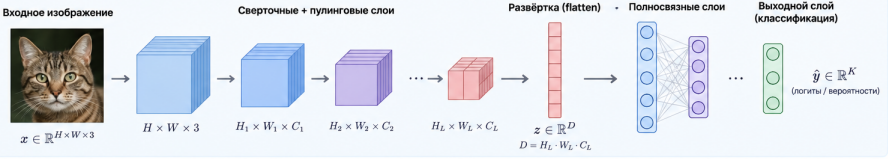
Пирамида фичей



Полносвязные слои на последних слоях сверточной сети

Сверточные и пулинговые слои извлекают пространственные признаки из изображения.
На последних слоях эти признаки преобразуются в вектор и подаются в полносвязные (fully connected, FC) слои, которые выполняют нелинейное преобразование и выдают логиты классов.

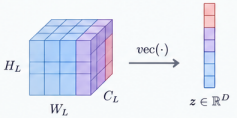
Общая схема



Развёртка признаков

Тензор признаков на выходе сверточной части преобразуется в вектор:

$$z = \mathrm{vec}(\boldsymbol{F}) \in \mathbb{R}^D, \quad D = H_L \cdot W_L \cdot C_L.$$



Пример.

Если $H_L = 7, W_L = 7, C_L = 512$, то

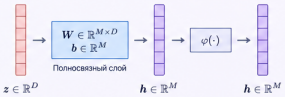
$$D = 7 \cdot 7 \cdot 512 = 25\,088.$$

Полносвязный слой

Полносвязный слой выполняет аффинное преобразование с последующей нелинейностью:

$$\boldsymbol{h} = \varphi(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}),$$

где $W \in \mathbb{R}^{M \times D}$, $b \in \mathbb{R}^M$, $h \in \mathbb{R}^M$,
$\varphi(\cdot)$ — функция активации (ReLU, GELU и др.).



Пример.

Если $D = 25\,088$ и $M = 4096$, то число параметров:

$$25\,088 \cdot 4\,096 + 4\,096 \approx 102.8 \text{ млн.}$$

Выходной слой и классификация

Последний полносвязный слой отображает вектор признаков в логиты для K классов:

$$\boldsymbol{o} = W_{cls}\boldsymbol{h} + b_{cls}, \qquad \boldsymbol{o} \in \mathbb{R}^K,$$

где $W_{cls} \in \mathbb{R}^{K \times M}$, $b_{cls} \in \mathbb{R}^K$.

Вероятности классов получаются с помощью softmax:

$$p_k = \frac{\exp(o_k)}{\sum_{j=1}^{K} \exp(o_j)}.$$

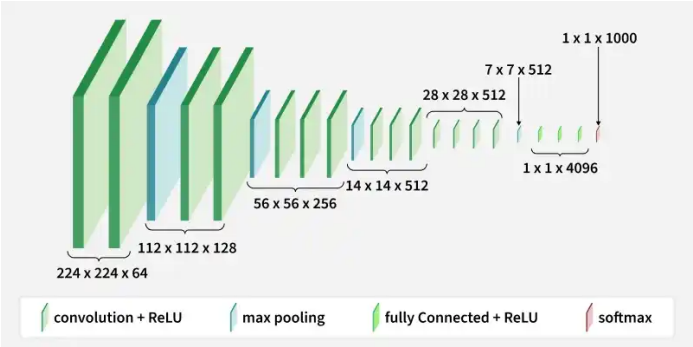
Класс с наибольшей вероятностью:

$$\hat{y} = \arg\max_k p_k.$$

Пример.

Для $K = 1000$ (ImageNet) и $M = 4096$:
число параметров выходного слоя: $1\,000 \cdot 4\,096 + 1\,000 \approx 4.1$ млн.

Пример CNN – VGG16





VGG в PyTorch

1. Определение сети вручную

```
import torch.nn as nn

model = nn.Sequential(
    nn.Conv2d(3, 64, 3, padding=1), nn.ReLU(),
    nn.Conv2d(64, 64, 3, padding=1), nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(64, 128, 3, padding=1), nn.ReLU(),
    nn.Conv2d(128, 128, 3, padding=1), nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Flatten(),
    nn.Linear(128 * 56 * 56, 4096), nn.ReLU(),
    nn.Linear(4096, 1000)
)
```

VGG в PyTorch

2. Готовая VGG из torchvision

```
from torchvision.models import vgg16, VGG16_Weights

# VGG-16
model = vgg16(weights=None)

# VGG-16,                                    ImageNet
model = vgg16(weights=VGG16_Weights.DEFAULT)
```

Архитектура VGG

Основные строительные блоки:

$$\texttt{Conv2d} \rightarrow \texttt{ReLU} \rightarrow \texttt{MaxPool2d}.$$

По мере углубления сети пространственный размер уменьшается, а число каналов увеличивается.