NEURAL NETWORKS

1.1. **Биологический нейрон.**

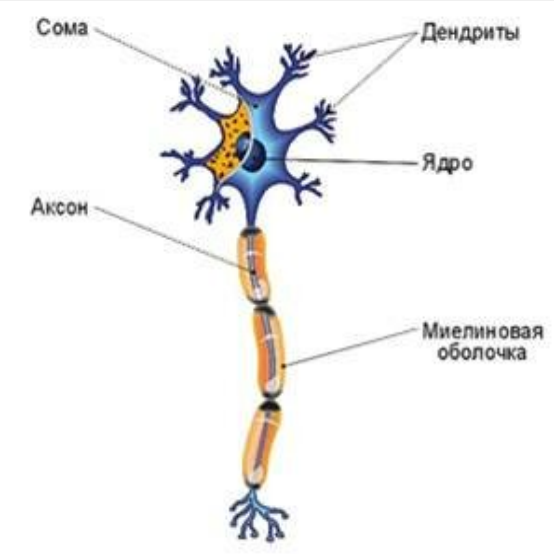


Рис. 1. Биологический нейрон

1.2. **Математическая модель нейрона.**

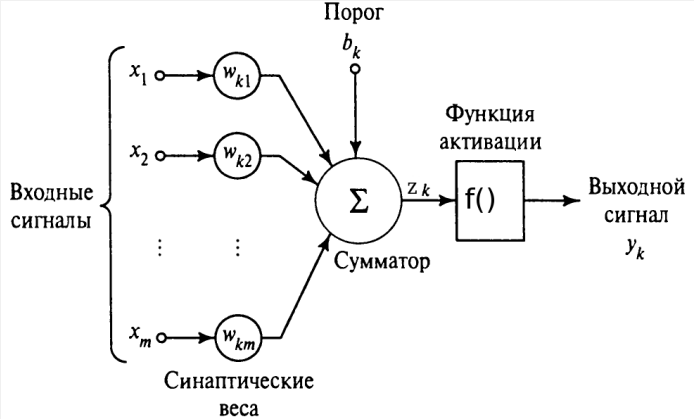


Рис. 2. Модель нейрона

$$z_k = \sum_{i=1}^{m} w_{ki} x_i + b_k,$$

$$y_k = f(z_k).$$

$b_k = -\theta, \quad \theta \geq 0$ — порог.

$$y_k = f(\sum_{i=1}^{m} w_{ki} x_i + b_k)$$

или матричном виде:

$$\boxed{\vec{y} = f(W\vec{x} + \vec{b})},$$

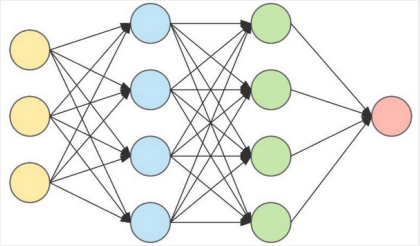
где $W = (w_{ki})$ — *матрица весов* (размера $n \times m$),

m — количество нейронов в предыдущем слое (input size),

n — количество нейронов в данном слое (output size),

$x = (x_1, ..., x_m)$ — INPUT (или выход предыдущего слоя),

$y = (y_1, ..., y_n)$ — OUTPUT (выход текущего слоя).

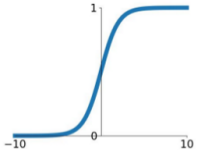


1.3. **Функция активации.**

Activation Functions

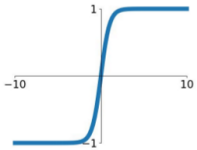
Sigmoid

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



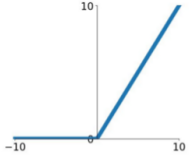
tanh

$$\tanh(x)$$



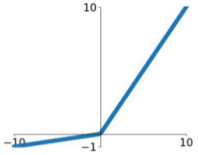
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

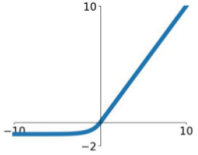


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

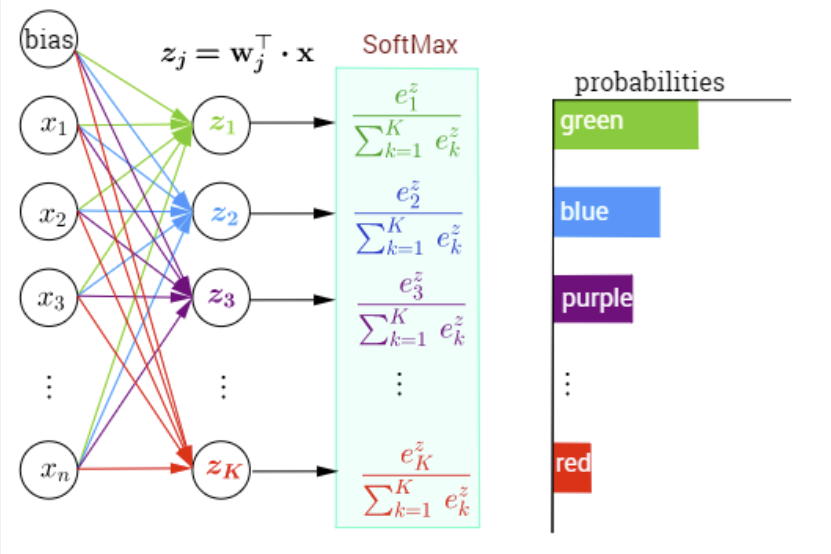
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



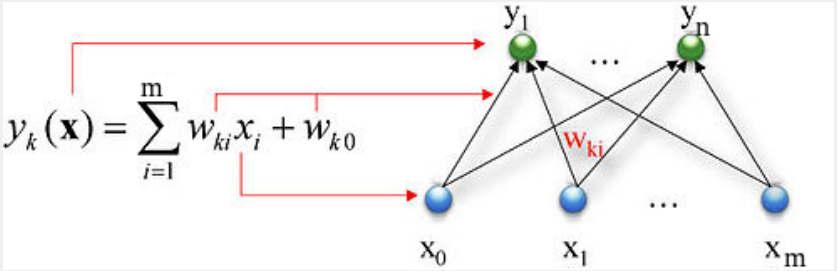
1.4. **Softmax.**

Softmax — векторная функция, применяемая к выходному слою нейронной сети для получения распределения вероятностей по классами (в задаче классификации):

$$softmax(\vec{z})_k = \frac{e^{z_k}}{\sum\limits_j e^{z_j}}, \quad \text{где } \vec{z} = W\vec{x} + \vec{b}.$$



1.5. **Однослойный персептрон.** *Однослойный персептрон* — простейшая нейронная сеть, состоящая из одного слоя нейронов.



Процесс обучения

Пусть n — номер шага (момент времени).

Сигнал ошибки: $e_k(n) = d_k(n) - y_k(n)$, где $y_k = f(\sum_{i=1}^{m} w_{ki}x_i + b_k)$.

Функция стоимости: $E = \sum_{k=1}^{n} \frac{1}{2}e_k^2, \quad E_{av} = \frac{1}{N}\sum_{s=1}^{N} E(s)$.

Дельта-правило: $\boxed{\Delta w_{ki} = \eta \cdot e_k \cdot f'(z_k) \cdot x_i}$.

$w_{ki}(t+1) = w_{ki}(t) + \Delta w_{ki} = w_{ki}(t) + \eta \cdot e_k \cdot f'(z_k) \cdot x_i$.

1.6. **Метод градиентного спуска.**

$\vec{w}(t+1) = \vec{w}(t) - \eta \nabla E$, где $\nabla E = \sum\limits_j \frac{\partial E}{\partial w_j} \vec{e}_j$.

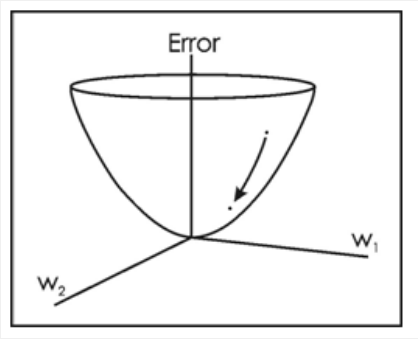


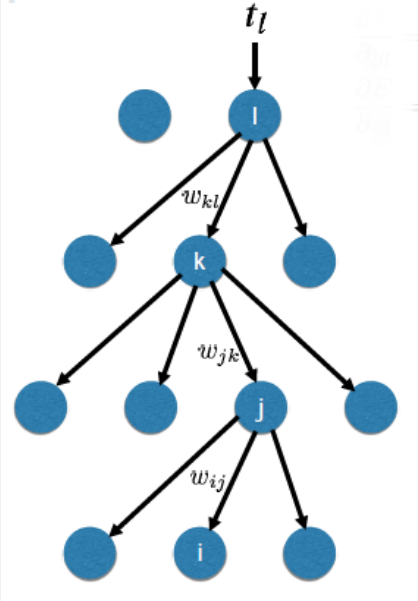
Рис. 3. Поиск минимума функции E

$$\boxed{\Delta w_{ki} = -\eta \frac{\partial E}{\partial w_{ki}}}.$$

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-T

2. BACKPROPAGATION

Метод обратного распространения ошибки (backpropagation) — метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Сигнал ошибки распространяется от выходов сети к ее входам.



3. CNN — Convolutional Neural Networks

Свёрточные нейронные сети — вид искусственных нейронных сетей, в которых используется операция свертки для выделения признаков во входном изображении.

Операция свертки двух функций: $(f * g)(t) = \int\limits_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$

Поводом к созданию CNN послужили исследования зрительного аппарата кошек, проведенные Хубелем и Вейселем в 1950–60-х гг.

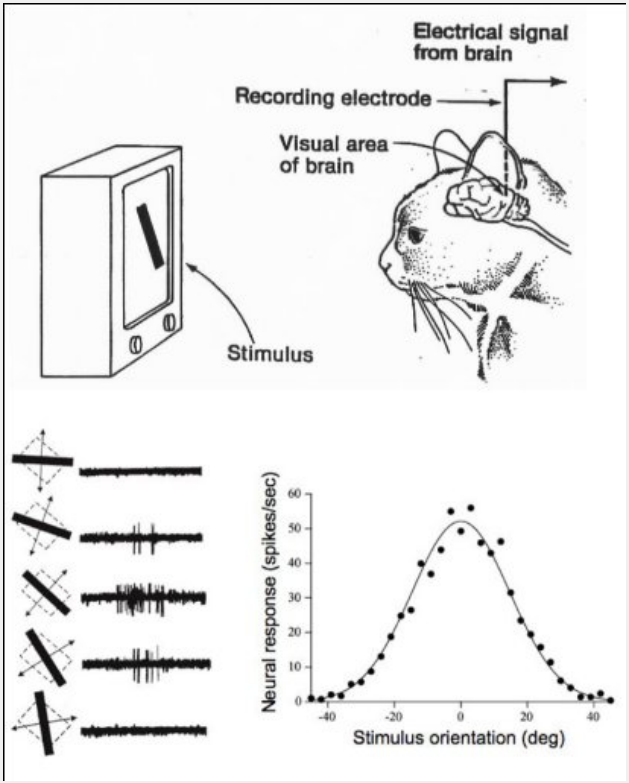


Рис. 4. Hubel and Wiesel, 1959

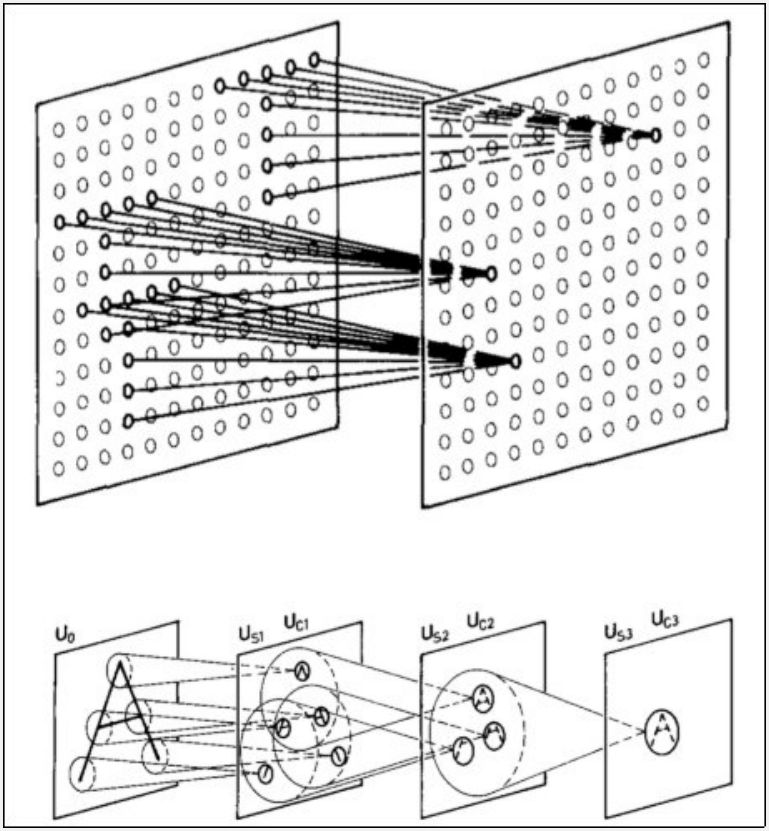


Рис. 5. Neocognitron (Fukushima, 1980)

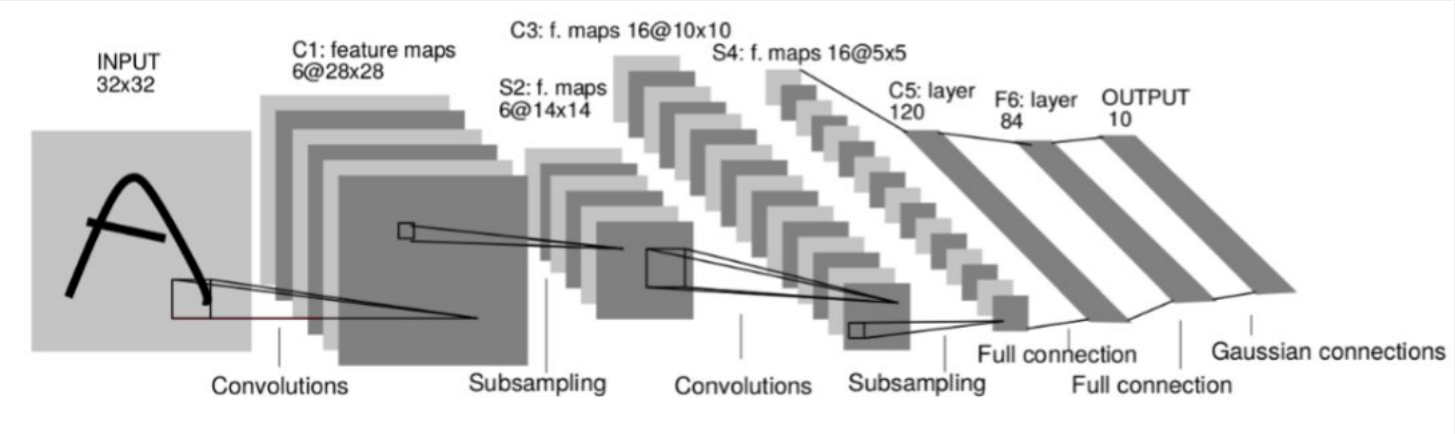


Рис. 6. Архитектура LeNet5 (Yann LeCun, 1988)

Для обучения CNN Ян Лекун использовал алгоритм обратного распространения ошибки (созданный в 1986 г.).

Предложенная архитектура состоит из чередующихся **сверточных** слоев (convolutio layers) и слоев **подвыборки** (subsampling, or pooling layers). Несколько последних слоев сети являются **полносвязными**.

3.1. **conv2d.**

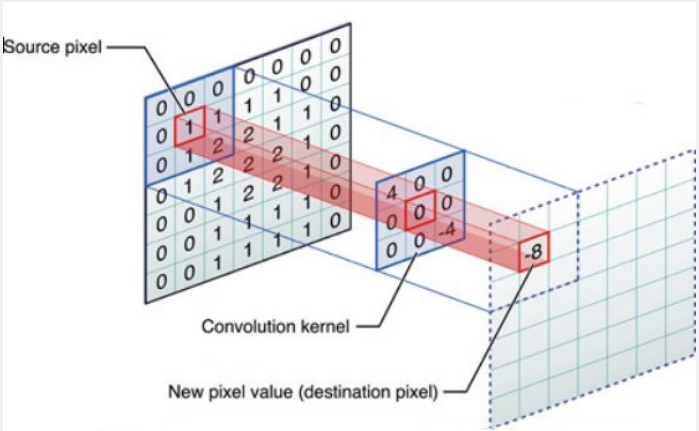
Двумерная дискретная свертка (conv2d) $Z = K * X$:

$$Z_{ij} = (K * X)_{ij} = \sum_{\alpha=0}^{k-1} \sum_{\beta=0}^{k-1} K_{\alpha\beta} \cdot X_{i+\alpha, j+\beta}, \quad 0 \le i, j < m - k + 1$$

X — входное изображение размера $m \times m$;

K — фильтр (или ядро свертки) размера $k \times k$;

Z — выход размера $n \times n$, где $n = m - k + 1$.

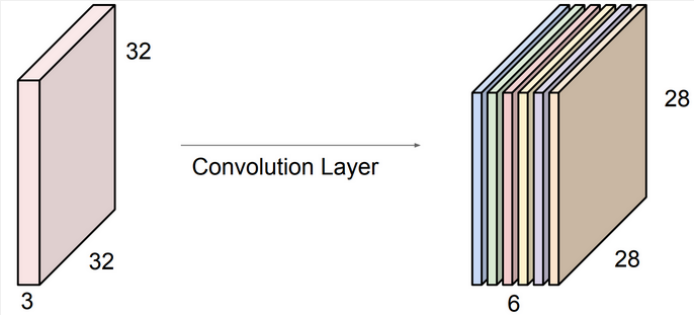


Реализация в PyTorch:

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size,
   stride=1, padding=0, bias=True)
```

Пример:

```
torch.nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5,
   stride=1, padding=0)
```



3.2. **Stride and Padding.**

Stride — шаг перемещения фильтра.

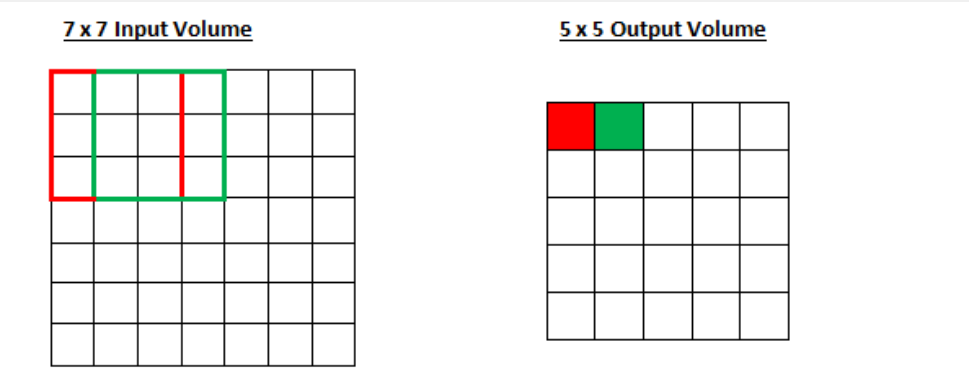


Рис. 7. kernel size = 3, stride = 1

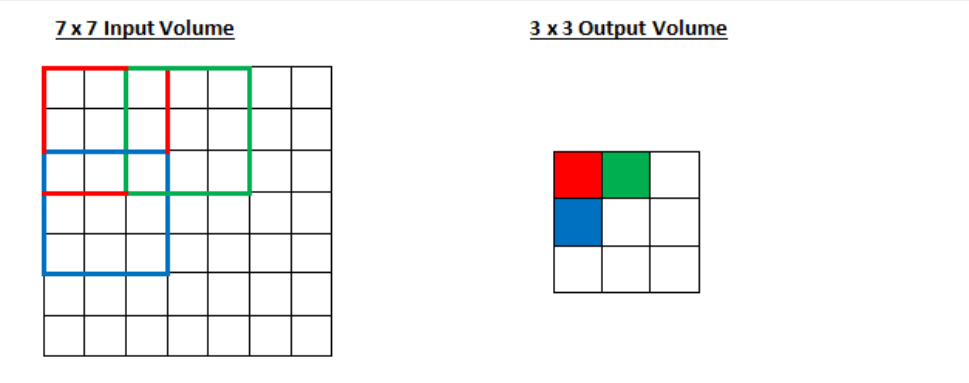
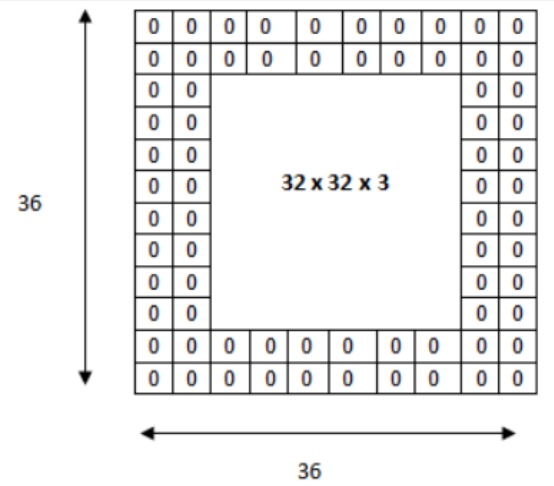


Рис. 8. kernel size = 3, stride = 2

Padding (набивка) — дополнение границ входного изображения нулями (или другими значениями).



Позволяет получать выходного изображение того же размера, что и входное. Для этого размер padding'а должен быть $\boxed{p = \frac{k-1}{2}}$ (если stride=1).

В общем случае размер выходного изображения рассчитывается по формуле:

$$\boxed{out = \frac{in - k + 2p}{s} + 1}$$

in — ширина входного изображения,

k — размер ядра (фильтра),

p — padding,

s — stride.

Ref.:

`https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Conv`

3.3. **Параметры padding в TensorFlow.**

- `SAME` (HALF) — к входному изображению добавляются нули так, чтобы размер изображения после конволюции не уменьшался: $out = in$.
- `VALID` — нет паддинга, то есть размер n изображения уменьшается после конволюции и становится равным $out = in - k + 1$, где k — размер ядра.

3.4. **Слой подвыборки.**

Слой подвыборки (subsample) — слой, который осуществляет локальное усреднение и понимажет разрешение изображения. Обычно следует за сверточным слоем.

- `avg_pool2d` — среднее значение,
- `max_pool2d` — максимальное значение (чаще используется):

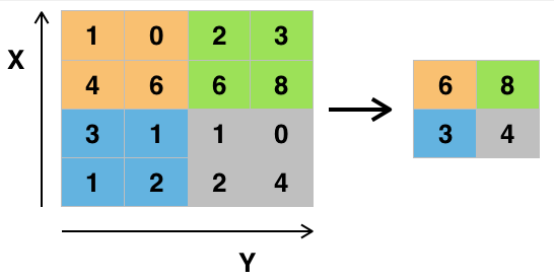


Рис. 9. maxpool2d with kernelsize=2 and stride=2

Реализация в PyTorch:

```
nn.MaxPool2d(kernel_size, stride=None, padding=0, ...)
```

или

```
nn.functional.max_pool2d(input, kernel_size, stride=None, padding=0)
```

Пример:

```
self.pool2d = nn.MaxPool2d(2, 2)
###
x = F.max_pool2d(x, 2, 2)
```

3.5. **MNIST.** :

MNIST database — Modified National Institute of Standards and Technology database.

Рис. 10. Примеры изображений из MNIST

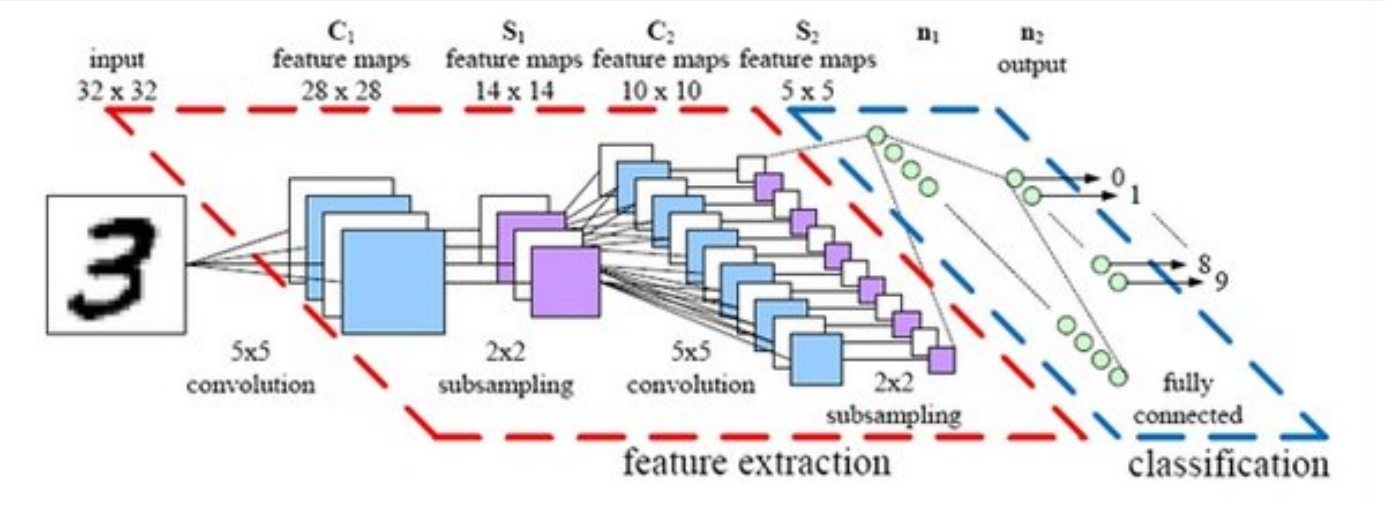


Рис. 11. Пример архитектуры сверточной сети

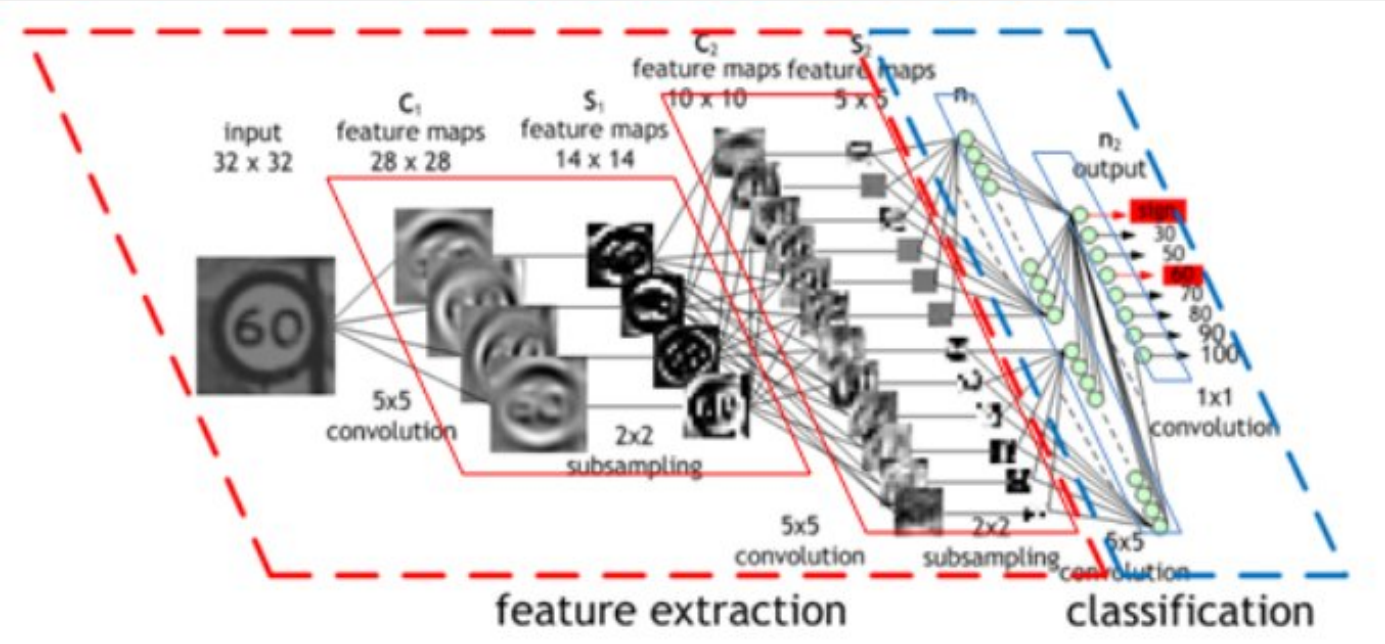


Рис. 12. Извлечение признаков

3.6. **AlexNet.** :

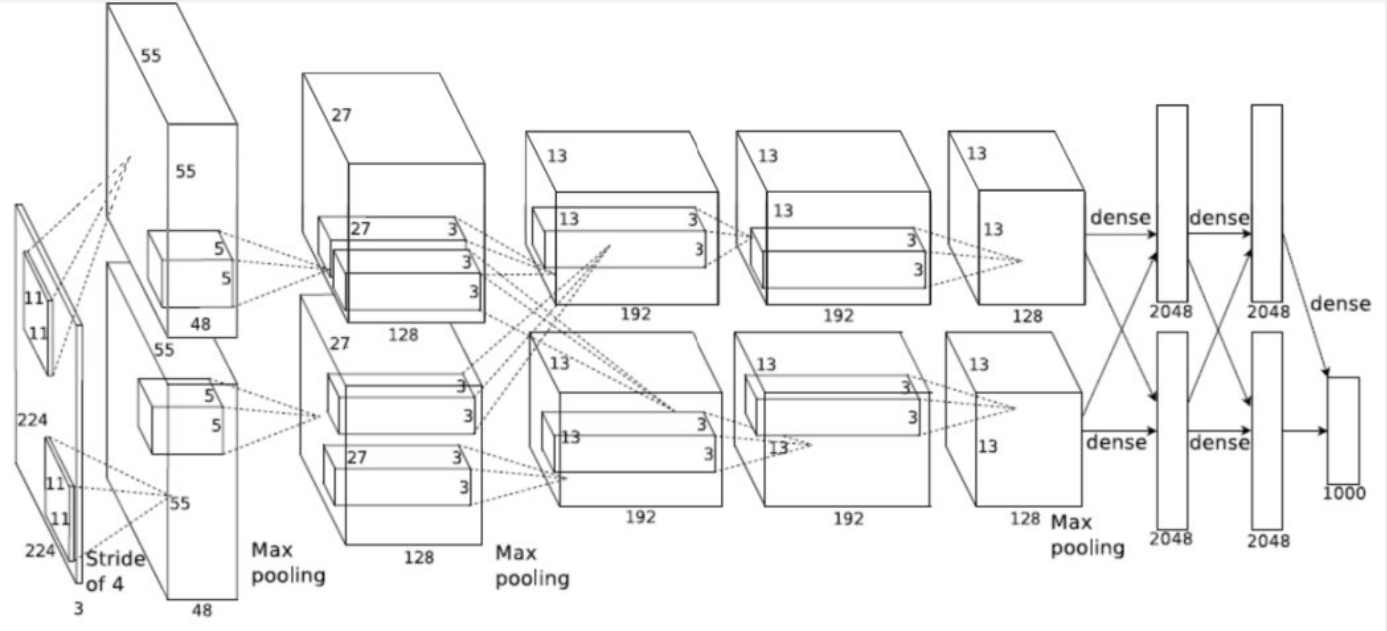


Рис. 13. AlexNet (2012) — точность >99.7% на MNIST