Сверточные нейронные сети (CNN)

Что такое сверточные нейронные сети?

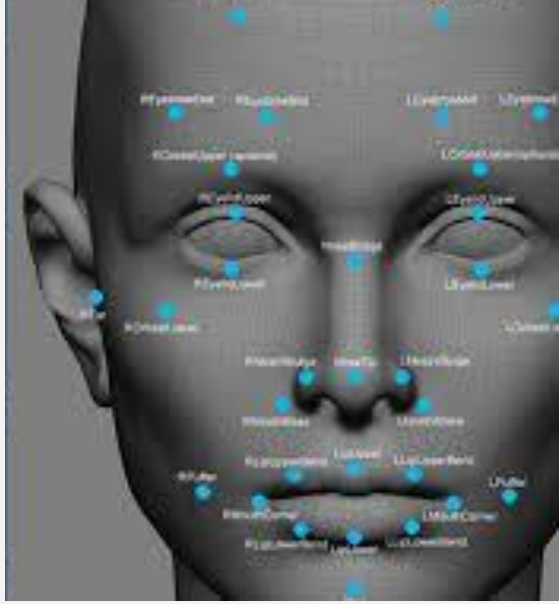
Свёрточные нейронные сети (Convolutional Neural Networks, CNN) – это тип нейросетей, предназначенный для обработки изображений и пространственных данных.

Способны автоматически извлекать признаки из изображений.

Отличаются от обычных нейросетей за счет использования свёрточных слоев и пулинга.

Применяются в задачах распознавания объектов, классификации изображений, медицинской диагностики и др.

Примеры применения



CNN используются в таких системах, как:

- Распознавание лиц (Face ID, камеры видеонаблюдения).
- Медицинская диагностика (обнаружение опухолей на снимках).
- Самоуправляемые автомобили (обнаружение объектов на дороге).

Основные компоненты CNN

Свёрточные нейронные сети состоят из нескольких ключевых компонентов:

- **Сверточные слои (Convolutional Layers)** – извлекают признаки из входных данных с помощью фильтров.
- **Фильтры (Керасы свертки, Kernels)** – маленькие матрицы весов, которые применяются к входному изображению.

Основные компоненты CNN

- **Функция активации (ReLU, Rectified Linear Unit)** – делает модель нелинейной.
- **Пулинг (Pooling, Subsampling)** – уменьшает размерность данных, сохраняя основные признаки.
- **Полносвязные слои (Fully Connected Layers, FC)** – выполняют финальную классификацию.

Механизм работы сверточного слоя

- **Фильтр (ядро, Kernel)** – небольшая матрица весов (например, 3×3), которая перемещается по изображению.
- **Свертка (Convolution Operation)** – умножает значения фильтра на соответствующие пиксели и суммирует результаты.
- **Карта признаков (Feature Map)** – итоговое изображение после применения свёртки.

Механизм работы CNN

- CNN анализирует изображение по частям, используя фильтры (ядра свёртки).
- Каждый фильтр выявляет определённые характеристики изображения (границы, текстуры, контуры).
- На выходе каждого слоя формируются новые признаки, передающиеся в следующие слои.

Как происходит свёртка?

Принцип свёртки – перемещение ядра (фильтра) по изображению

- Фильтр скользит по изображению, выполняя матричное умножение и суммирование.
- В результате формируется карта признаков (feature map).
- Чем глубже слой – тем более сложные признаки обнаруживаются.

Формула свёртки

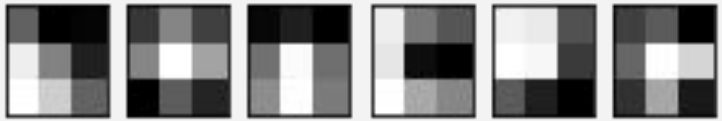
$$\text{Output}(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot K(m,n)$$

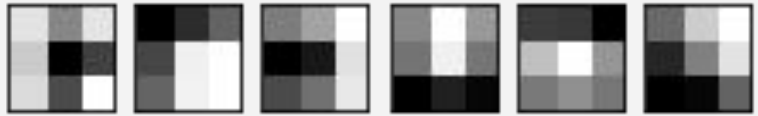
где X – входное изображение, K – ядро свёртки.

Визуализация работы фильтров

Разные фильтры извлекают разные признаки:

- Детектирование контуров (вертикальных/горизонтальных)
- Выявление текстур и углов
- Распознавание более сложных деталей





Как формируются признаки в разных слоях?

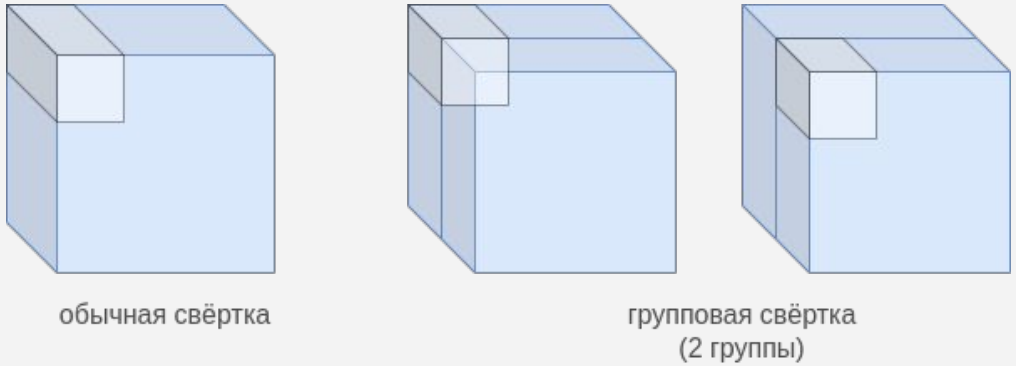
📌 **Слои CNN извлекают признаки с разной степенью абстракции**

- **Ранние слои** – выявляют простые признаки (границы, текстуры).
- **Средние слои** – обнаруживают более сложные объекты (формы, части объектов).
- **Глубокие слои** – распознают сложные структуры (лица, объекты, символы).

Разновидности операций свёртки

1. Обычная свёртка (Standard Convolution)

Это классическая операция свёртки, где фильтр (ядро свёртки) скользит по изображению и выполняет матричное произведение с фрагментами входного изображения.



1. Обычная свёртка (Standard Convolution)

- Используется фиксированное ядро (например, 3×3, 5×5).
- Каждый пиксель выходной карты признаков вычисляется как сумма произведений весов ядра на соответствующие пиксели входного изображения.
- Позволяет выделять локальные признаки, такие как границы, текстуры.

$$Y(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} X(i+m, j+n) \cdot W(m,n)$$

Разновидности операций свёртки

2. Разреженная свёртка (Sparse Convolution)

Операция свёртки выполняется не на каждом пикселе, а на выборочных позициях, что снижает количество вычислений.

- Уменьшает вычислительную сложность, обрабатывая только часть пикселей.
- Используется в задачах, где важны только определённые элементы изображения (например, обработка точечных облаков в 3D).

Разновидности операций свёртки

3. Свёртка с увеличенным шагом (Dilated Convolution)

Позволяет увеличить охват области восприятия ядра без увеличения числа параметров. В отличие от обычной свёртки, между значениями, по которым скользит фильтр, появляются "пропуски" (dilation rate).

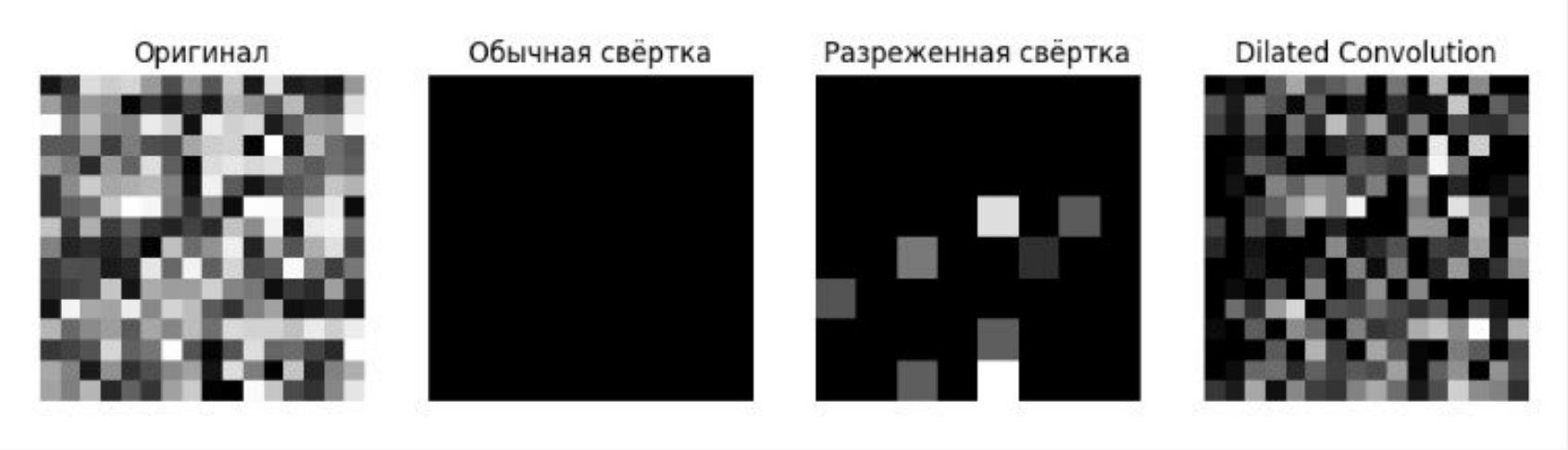
Пример:

- Dilation rate = 1 → обычная свёртка.
- Dilation rate = 2 → ядро 3×3 охватывает область 5×5, но использует 9 элементов.

3. Свёртка с увеличенным шагом (Dilated Convolution)

- Используется параметр **dilation rate** (коэффициент расширения), определяющий расстояние между соседними значениями в ядре.
- Позволяет учитывать более широкие контексты без увеличения размера ядра.
- Часто применяется в **сегментации изображений** (например, DeepLab) и **обработке последовательностей**.

Визуализации разных типов свёртки

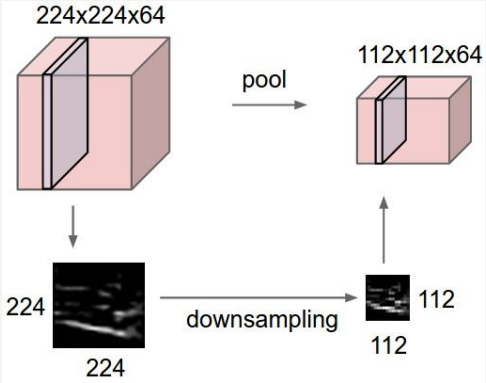


Пулинг (Pooling): уменьшение размерности

Пулинг — это операция уменьшения размерности карты признаков, которая выполняется после свёрточного слоя. Она позволяет:

✔ Уменьшить количество вычислений, сокращая размер входных данных.
✔ Сделать извлечённые признаки более устойчивыми к сдвигам и искажениям.
✔ Избежать переобучения за счёт удаления незначительной информации.

Виды пулинга



1. **MaxPooling** (макспулинг)

- Выбирает **максимальное** значение в каждой области фильтра.
- Позволяет выделить наиболее выраженные признаки.
- Часто применяется в задачах распознавания объектов.

Виды пулинга

2️⃣ **AveragePooling** (средний пулинг)

- Вычисляет **среднее значение** в области фильтра.
- Используется, когда важно учитывать всю информацию, а не только наибольшие значения.
- Применяется в задачах, где детали важнее выделенных признаков.

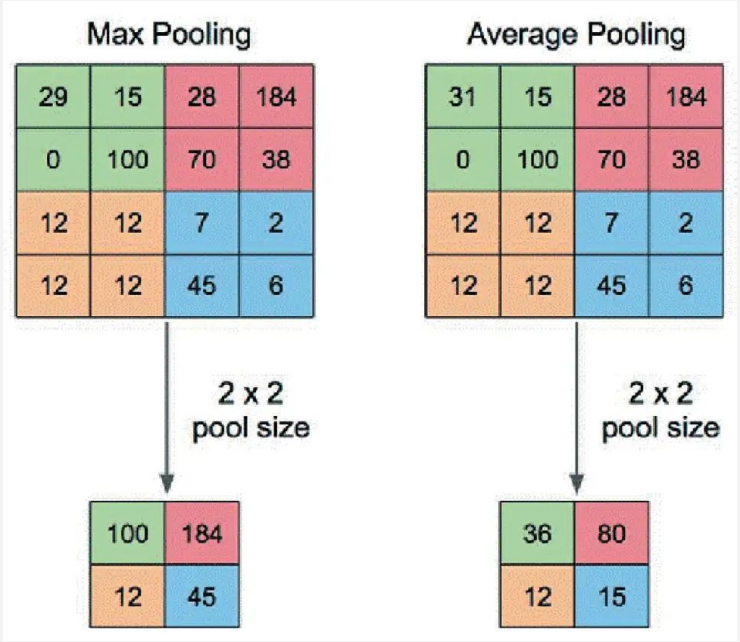
Как работает пулинг?

Входная матрица:

[[1, 3, 2, 4],
 [5, 6, 8, 9],
 [3, 2, 1, 4],
 [7, 5, 6, 8]]

После MaxPooling (2×2):

[[6, 9],
 [7, 8]]



Как выбор пулинга влияет на обучение сети?

MaxPooling:

✔ Удаляет шум, фокусируясь на наиболее значимых элементах.

✔ Делает сеть более устойчивой к изменениям положения объекта.

✔ Может потерять некоторые детали.

Как выбор пулинга влияет на обучение сети?

AveragePooling:

✔ Сохраняет больше информации, усредняя значения.

✔ Подходит для задач, где важна каждая деталь.

✔ Может размывать важные признаки.

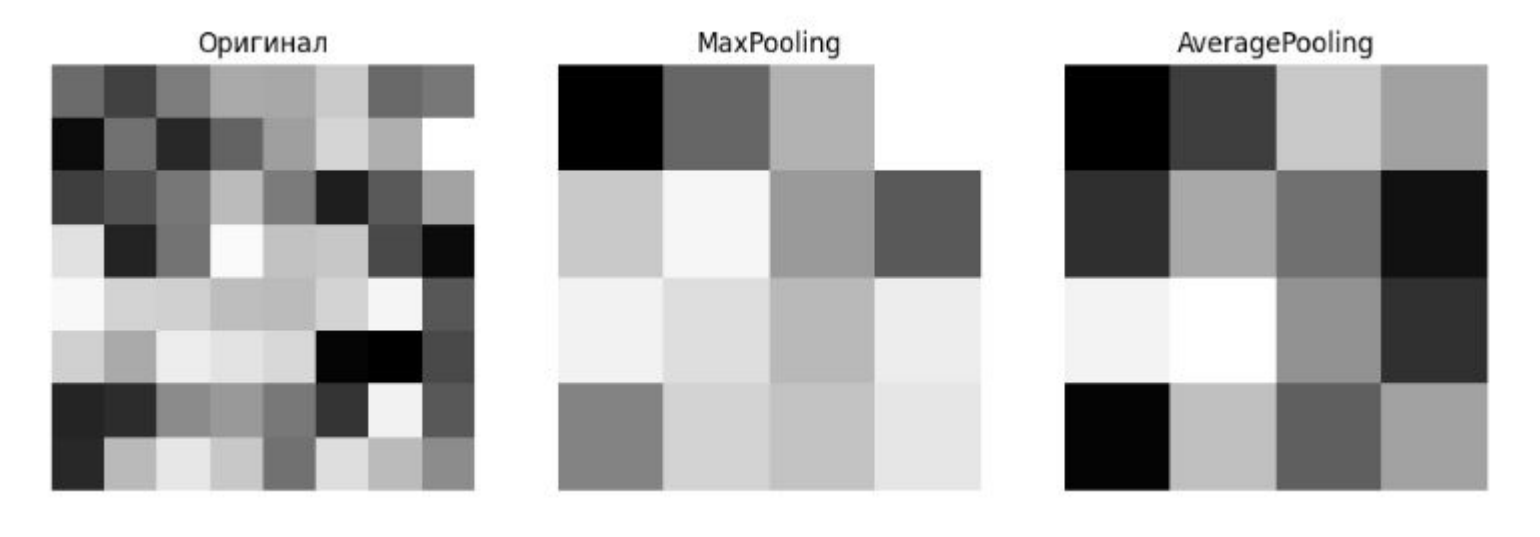
Код для демонстрации работы пулинга в Python

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from    tensorflow.keras.layers    import    MaxPooling2D,
AveragePooling2D

image = np.random.rand(1, 8, 8, 1).astype(np.float32)

max_pooled = MaxPooling2D(pool_size=(2, 2))(image)
avg_pooled = AveragePooling2D(pool_size=(2, 2))(image)
```

Код для демонстрации работы пулинга в Python

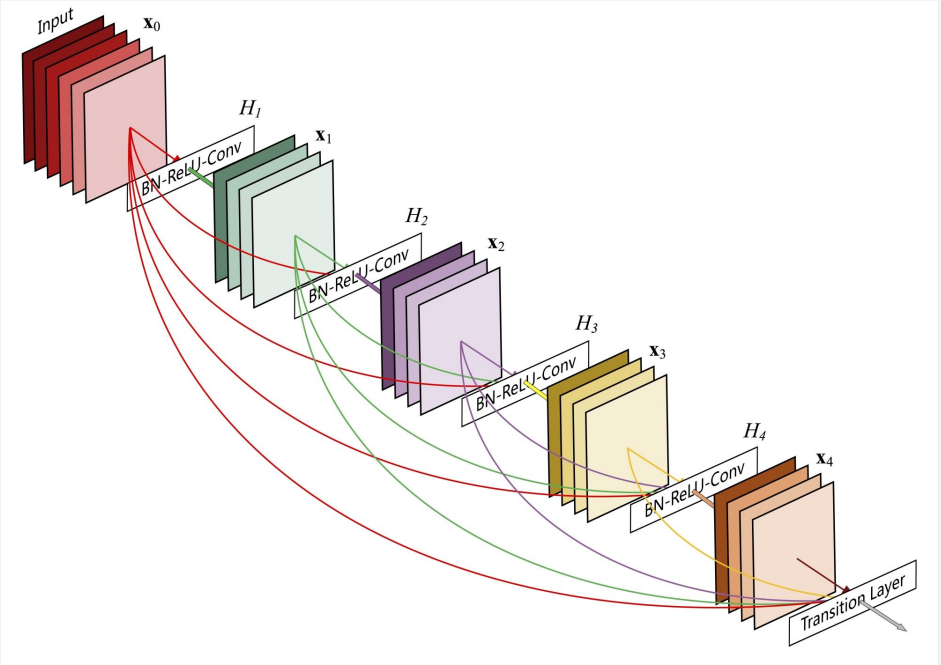
```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[0].imshow(image[0, :, :, 0], cmap="gray")
axes[0].set_title("Оригинал")
axes[1].imshow(max_pooled[0, :, :, 0], cmap="gray")
axes[1].set_title("MaxPooling")
axes[2].imshow(avg_pooled[0, :, :, 0], cmap="gray")
axes[2].set_title("AveragePooling")

for ax in axes:
    ax.axis("off")

plt.show()
```

Демонстрация на реальном примере



Популярные архитектуры CNN

1. LeNet-5
2. AlexNet
3. VGGNet
4. ResNet



1. LeNet-5 (1998)

Применение: Распознавание рукописных цифр (MNIST)

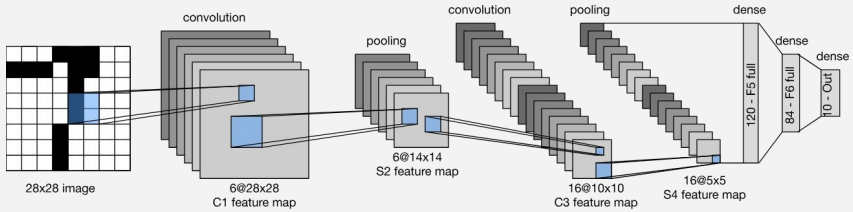
🛠️ **Особенности:**

✔ Одна из первых CNN

✔ Содержит **7 слоёв** (свёрточные, субдискретизация (пулинг), полносвязные)

✔ Использует **активацию tanh** вместо ReLU

1. LeNet-5 (1998)



📊 Структура:

- **Conv1** (5×5) → **Pooling** →
- **Conv2** (5×5) → **Pooling** →
- **FC1** → **FC2** → **Softmax (Выход 10 классов)**

Плюсы: Простая, быстрая

Минусы: Слабая для сложных изображений

2. AlexNet (2012)

Применение: Победитель ImageNet Challenge 2012

🛠️ **Особенности:**

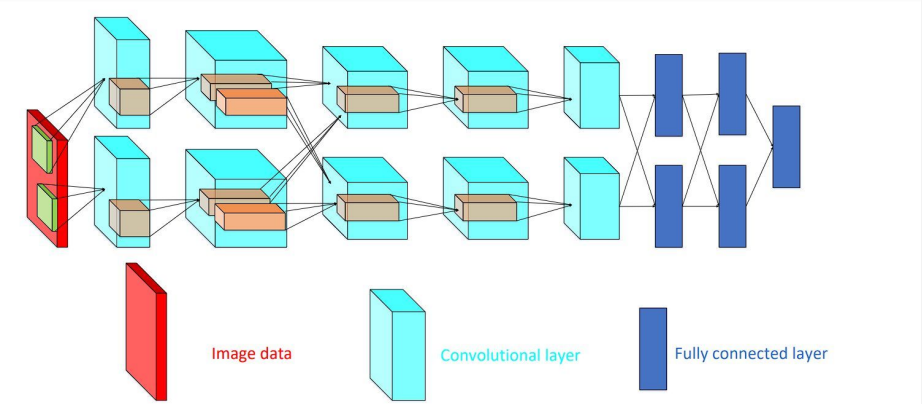
✔ Использует **ReLU** вместо tanh

✔ Применяет **Dropout** для борьбы с переобучением

✔ Вводит **свёртки с перекрытием (overlapping pooling)**

✔ **Обучена на двух GPU** для ускорения

2. AlexNet (2012)



📊 **Структура:**

- **Conv1** (11×11) → **Pooling** →
- **Conv2** (5×5) → **Pooling** →
- **Conv3,4,5** (3×3) →
- **FC1** → **FC2** → **FC3 (Softmax, 1000 классов)**

Плюсы: Глубже и мощнее LeNet

Минусы: 60 млн параметров → требует мощного оборудования

3. VGGNet (2014)

Применение: Улучшенный CNN для ImageNet

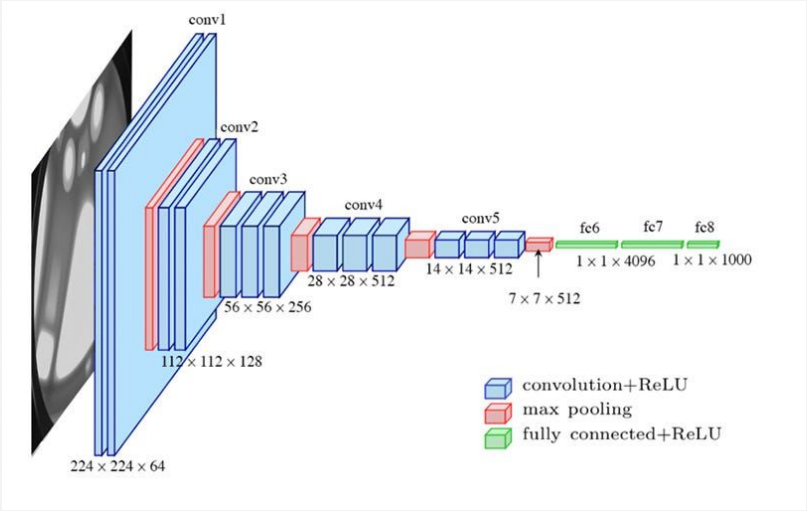
🛠️ **Особенности:**

✔ Использует **только 3×3 свёртки**, но **очень глубокая сеть**

✔ Применяет **двойной MaxPooling**

✔ Два варианта: **VGG-16 и VGG-19**

3. VGGNet (2014)



📊 Структура (VGG-16):

- **Conv1-Conv2** (2×3×3) → Pooling
- **Conv3-Conv4** (3×3×3) → Pooling
- **Conv5** (3×3×3) → Pooling
- **FC1 → FC2 → Softmax (1000 классов)**

Плюсы: Отличное качество, простая архитектура

Минусы: Очень медленная, требует **много памяти**

4. ResNet (2015)

Применение: Победитель ImageNet 2015

🛠️ **Особенности:**

✔ Вводит **residual connections** (пропускные соединения)

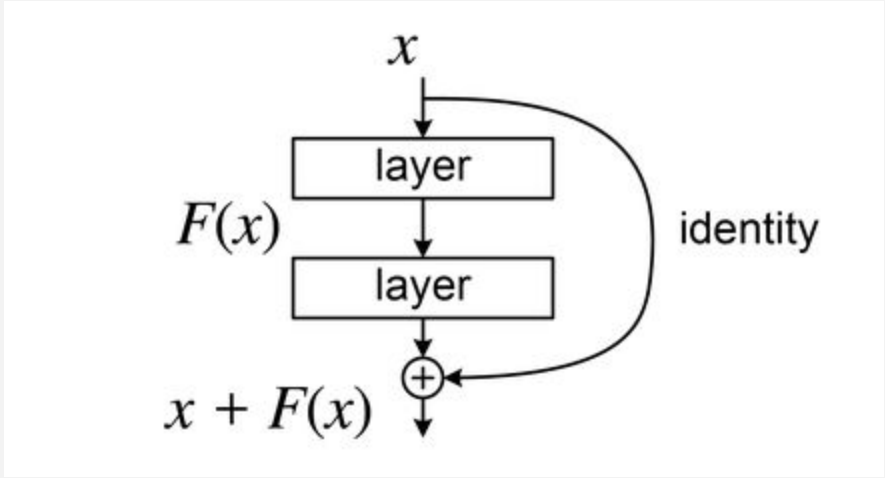
✔ Позволяет строить **глубокие сети (ResNet-50, ResNet-101, ResNet-152)**

✔ Обходит **проблему затухающих градиентов**

4. ResNet (2015)

📊 **Структура:**

- **Conv1** → Pooling
- **Residual Block (3×3, 3×3)** →
- **Много residual-блоков**
- **FC** → **Softmax**



Плюсы: Глубокая, точная

Минусы: Сложная, требует GPU

Пример работы CNN в классификации изображений

CNN отлично подходит для задач классификации изображений. Один из стандартных датасетов для тестирования – **CIFAR-10**, который содержит 60 000 цветных изображений (10 классов).

Пример кода для обучения CNN на CIFAR-10 (Python, TensorFlow/Keras)

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  #
Нормализация
y_train, y_test = to_categorical(y_train),
to_categorical(y_test)
```

Пример кода для обучения CNN на CIFAR-10 (Python, TensorFlow/Keras)

```
model = models.Sequential([
            layers.Conv2D(32,   (3,3),   activation='relu',
input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Пример кода для обучения CNN на CIFAR-10 (Python, TensorFlow/Keras)

```
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train,  y_train,  epochs=10,  batch_size=64,
validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Точность на тестовых данных: {test_acc:.4f}')
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.3499 - loss: 1.7619 - val_accuracy: 0.4942 - val_loss: 1.3845
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.5658 - loss: 1.2181 - val_accuracy: 0.6161 - val_loss: 1.0788
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.6328 - loss: 1.0417 - val_accuracy: 0.6347 - val_loss: 1.0299
Epoch 4/10
```

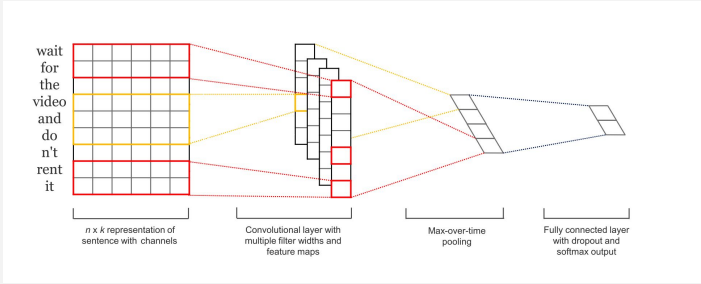
Применение CNN в различных областях

✅ Компьютерное зрение:

- Распознавание лиц (FaceID, камеры наблюдения)
- Обнаружение объектов (YOLO, SSD, Faster R-CNN)

✅ Автономные автомобили:

- Распознавание дорожных знаков
- Обнаружение препятствий
- Навигация по дороге



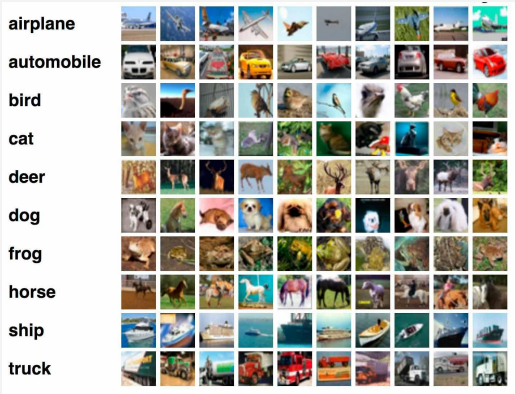
Применение CNN в различных областях

✅ Обнаружение аномалий:

- Поиск дефектов на производстве
- Фрод-детекция в банковской сфере

✅ Генерация изображений:

- GANs (Generative Adversarial Networks)
- Автоэнкодеры



Ограничения CNN

❌ **Требование больших объемов данных**

CNN нуждается в миллионах размеченных изображений для качественного обучения.

❌ **Высокая вычислительная сложность**

Глубокие сети требуют GPU/TPU, так как обучаются неделями.

❌ **Уязвимость к атакам с малым изменением входных данных**

Небольшие модификации изображения (Adversarial Attacks) могут обмануть сеть.