Обзор нейронных сетей для компьютерного зрения

План

1. **Введение:** Задачи Computer Vision и почему именно нейронные сети?
2. **Предшественники:** От перцептрона к идее свертки
3. **Эра CNN:** LeNet, AlexNet, VGG, Inception, ResNet
4. **Архитектуры для детекции и сегментации:** R-CNN, YOLO, U-Net, Mask R-CNN
5. **Современные тренды:**
    - Transformer'ы в Vision (ViT, DETR)
    - Нейросети без учителя (SSL) и самообучение (Self-Supervised Learning)
    - Диффузионные модели для генерации изображений
6. **Заключение:** Куда движется область?

Ключевые задачи

Классификация: «Что на изображении?2 (кошка vs собака)

Детекция объектов: «Что и где находится?» (найти все машины на изображении и обвести рамкой)

Семантическая сегментация: «Каждому пикселю – класс» (все пиксели, принадлежащие дороге, выделить одним цветом)

Инстанс-сегментация: «Разделить объекты одного класса» (разделить каждого человека в толпе)

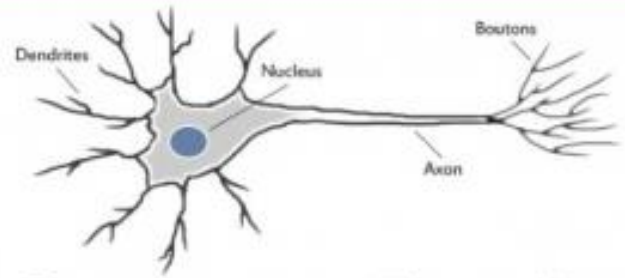
Генерация изображений: Создание новых изображений из описания или шума

Почему нейронные сети?

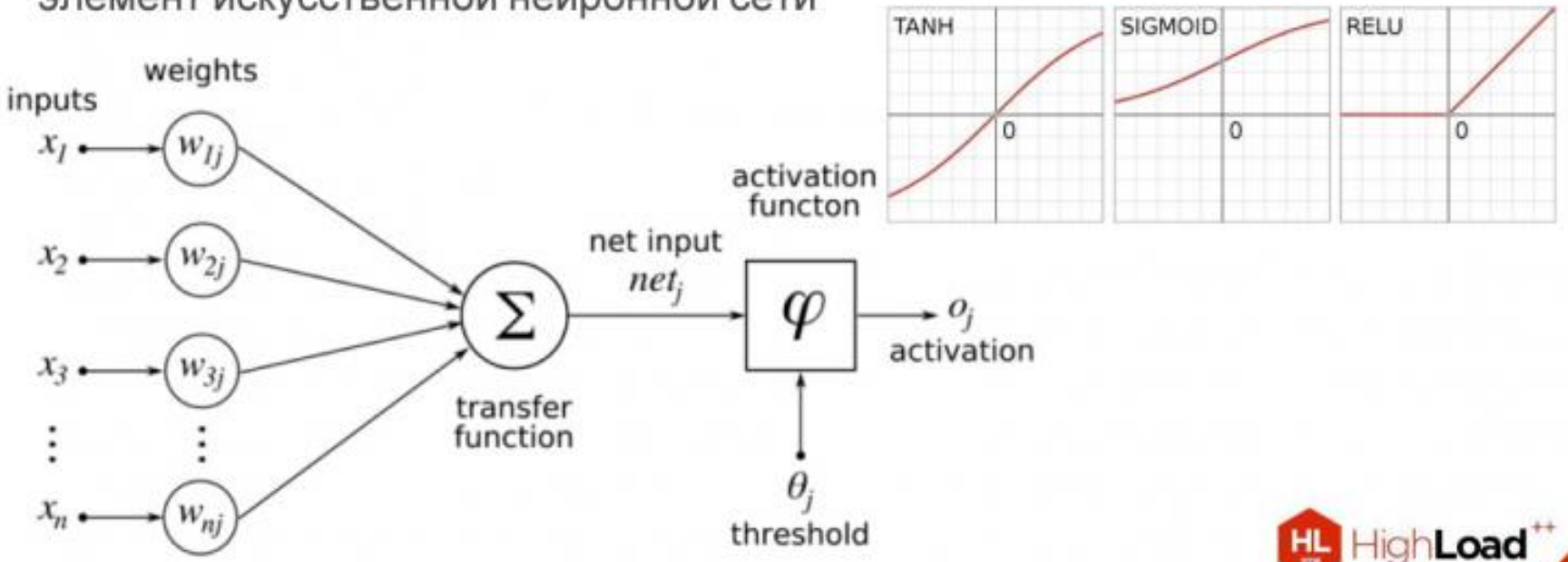
Традиционные методы (например, SIFT, HOG) требовали ручного создания признаков (feature engineering)

Нейронные сети, особенно сверточные (CNN), научились извлекать иерархические признаки автоматически: от простых (края, углы) к сложным (формы, объекты)

Искусственный нейрон



Искусственный нейрон — отдалённое подобие биологического. Базовый элемент искусственной нейронной сети



Более формальное определение функции активации

функция, которая применяется к взвешенной сумме входов нейрона (плюс смещение, bias), чтобы определить его выходной сигнал.

Процесс выглядит так:

- Нейрон получает входные данные (x1, x2, ..., xn)
- Каждый вход умножается на свой вес (w1, w2, ..., wn), отражающий его важность
- Вычисляется взвешенная сумма: z = (w1*x1 + w2*x2 + ... + wn*xn) + b (где b — смещение)
- К этой сумме применяется функция активации: output = f(z)
- Это значение output передается на вход следующим нейронам

Зачем нужна функция активации?

- **Введение нелинейности**
  Иначе вся нейронная сеть, независимо от количества слоев, сводилась бы к одной простой линейной операции (умножению на матрицу)
  Без нелинейности сеть не сможет научиться решать сложные задачи, такие как распознавание изображений, понимание речи и т.д.
  Она была бы не мощнее простой линейной регрессии

- **Ограничение и нормализация выхода**
  Функция активации преобразует неограниченную взвешенную сумму в выходное значение, которое имеет предсказуемый диапазон (например, от 0 до 1 или от -1 до 1). Это помогает стабилизировать обучение сети

- **Принятие решения (в пороговых функциях)**
  В простейших моделях функция активации буквально включает или выключает нейрон по аналогии с биологическим прототипом.

Популярные виды функций активации

1. Линейная (Linear)

Формула: f(z) = z

Проблема: Бесполезна для скрытых слоев, так как не вносит нелинейность. Используется обычно в выходном слое для задач регрессии

2. Сигмоида (Sigmoid / Logistic)

Формула: f(z) = 1 / (1 + e^(-z))

Диапазон: (0, 1)

Плюсы: Полезна для выходного слоя в задачах бинарной классификации, так как интерпретирует выход как вероятность

Минусы: Склонна к проблеме исчезающего градиента при обучении глубоких сетей

3. Гиперболический тангенс (Tanh)

Формула: f(z) = (e^z - e^(-z)) / (e^z + e^(-z))

Диапазон: (-1, 1)

Плюсы: Центрирована вокруг нуля, что часто делает обучение более эффективным, чем у сигмоиды.

Минусы: Также страдает от проблемы исчезающего градиента

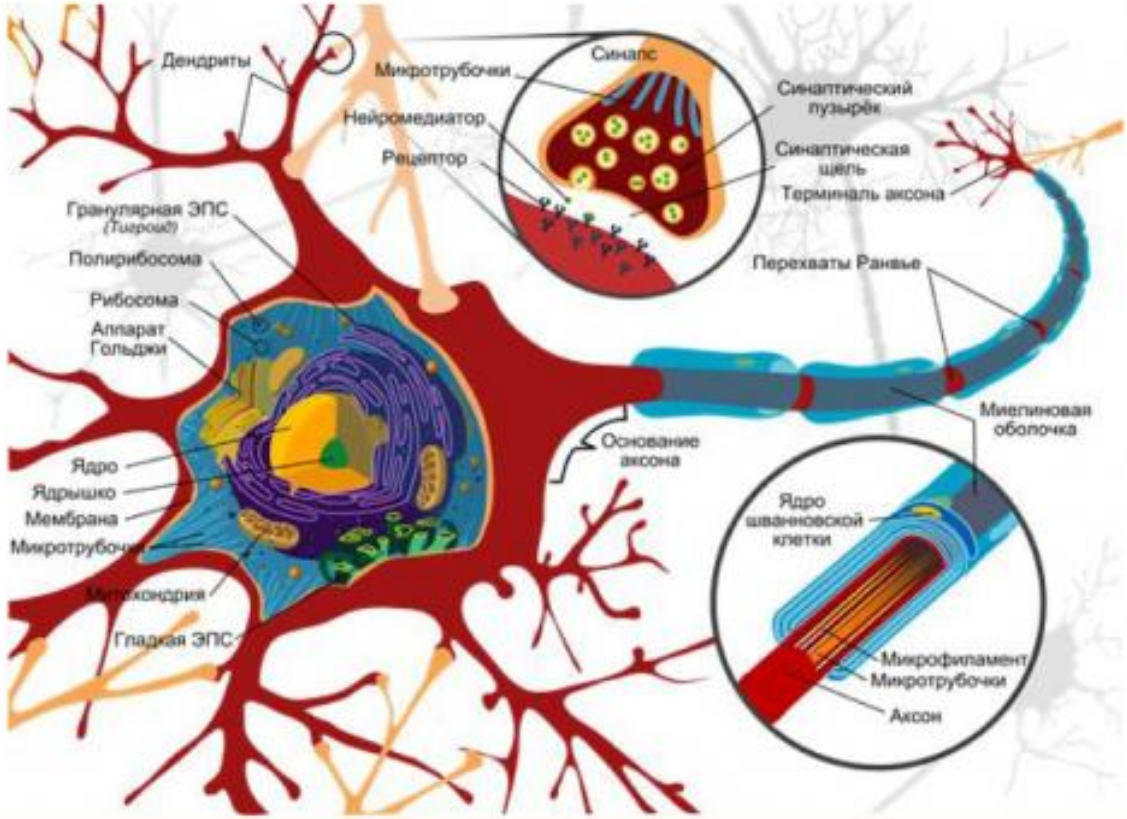
Популярные виды функций активации

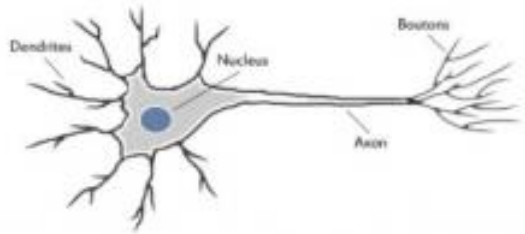
4. ReLU (Rectified Linear Unit) - Самая популярная
Формула: f(z) = max(0, z)
Диапазон: [0, +∞)
Плюсы: Очень простая для вычисления, не страдает от проблемы исчезающего градиента (в положительной области), на практике показывает отличные результаты
Минусы: Проблема умирающего ReLU — если нейрон всегда получает отрицательный z, он «умирает» и перестаёт учиться

5. Leaky ReLU (и ее варианты, как Parametric ReLU - PReLU)
Формула: f(z) = max(α * z, z) (где α — маленький параметр, например, 0.01)
Плюсы: Решает проблему умирающего ReLU, позволяя небольшой отрицательный выход. Часто работает лучше, чем обычный ReLU

6. Softmax
Формула: Преобразует вектор чисел в вектор вероятностей, где сумма всех выходов равна 1
Применение: Используется исключительно в выходном слое для задач многоклассовой классификации

В реальности всё посложнее...





HighLoad++

Перцептрон

Перцептрон — это первый в истории практический пример такого искусственного нейрона, реализованный с помощью пороговой функции. Это частный, хотя и исторически очень важный, случай искусственного нейрона

Перцептрон — это простейший тип искусственной нейронной сети, точнее, это один-единственный искусственный нейрон с определенным типом функции активации

Перцептрон был изобретен Фрэнком Розенблаттом в 1957 году и стал краеугольным камнем в области машинного обучения

Википедия:
Перцептро́н — математическая или компьютерная модель восприятия информации мозгом (кибернетическая модель мозга), впервые воплощённая в виде электронной машины «Марк-1» в 1960 году. Перцептрон стал одной из первых моделей нейросетей, а «Марк-1» — первым в мире нейрокомпьютером.

Что может перцептрон?

Ключевая особенность перцептрона:

- В качестве функции активации используется пороговая функция (ступенчатая функция): если сумма взвешенных входов больше порога, выход равен 1, иначе выход равен 0

Он может решать только **задачи линейной классификации**

- Это означает, что он может провести только одну прямую линию (или гиперплоскость в многомерном пространстве), чтобы разделить два класса данных

Классический пример: Логическая функция И (AND)

- Есть два входа (x_1, x_2), каждый может быть 0 или 1
- Выход равен 1, только если ОБА входа равны 1
- Перцептрон может легко обучиться проводить линию, которая правильно разделяет эти два класса: (0,0), (0,1), (1,0) → класс «0», и (1,1) → класс «1»

Ограничение перцептрона (и почему была «зима ИИ»)

Перцептрон НЕ МОЖЕТ решить задачу «исключающего ИЛИ» (XOR),
где классы не являются линейно разделимыми

Это было доказано Марвином Мински и показало фундаментальное ограничение перцептронов,
что на время охладило интерес к нейронным сетям

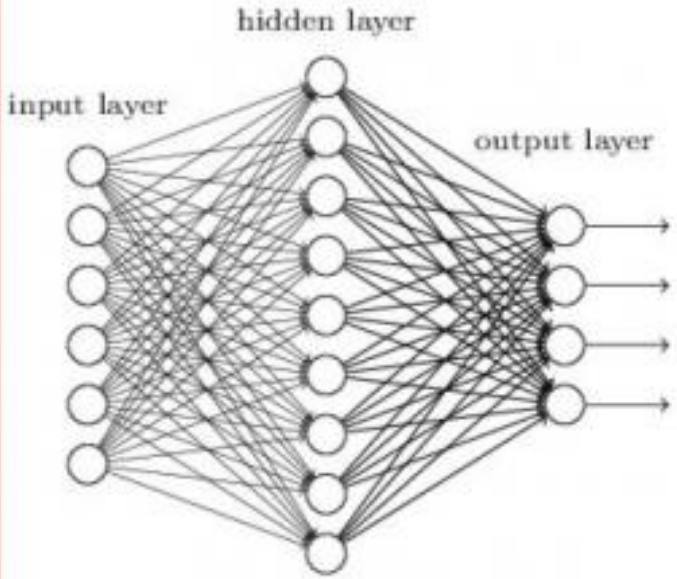
Эволюция: от Перцептрона к Современным Нейронным Сетям

Чтобы преодолеть ограничения перцептрона:

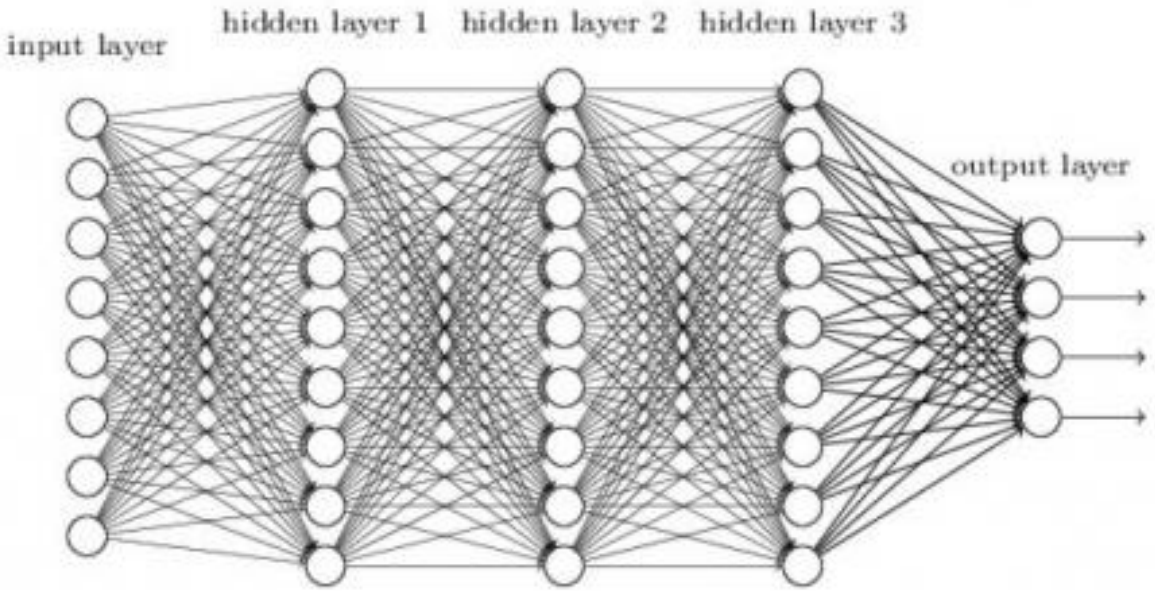
1. Стали использовать другие, более «гладкие» функции активации (например, сигмоиду)
2. Стали объединять множество нейронов в слои
3. Создали многослойные перцептроны (MLP) — это уже полноценные нейронные сети с входным, скрытыми и выходным слоями. Именно такие сети сегодня и называют «нейронными сетями»

Искусственная нейросеть

"Non-deep" feedforward neural network

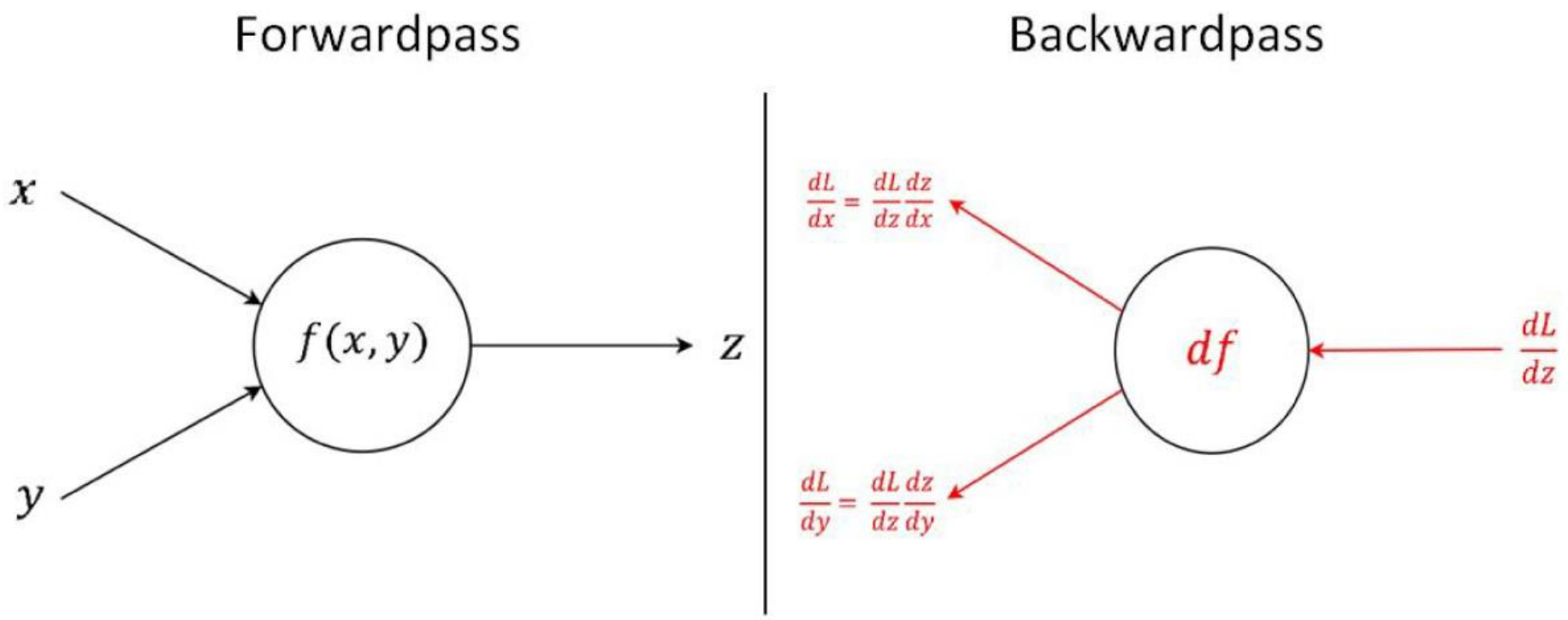


Deep neural network



HighLoad++

Обучение нейросети: backpropagation



Алгоритм обратного распространения ошибки

Полносвязная нейронная сеть (Fully Connected Network) — Многослойный перцептрон (MLP)

Это классический и самый простой тип искусственной нейронной сети

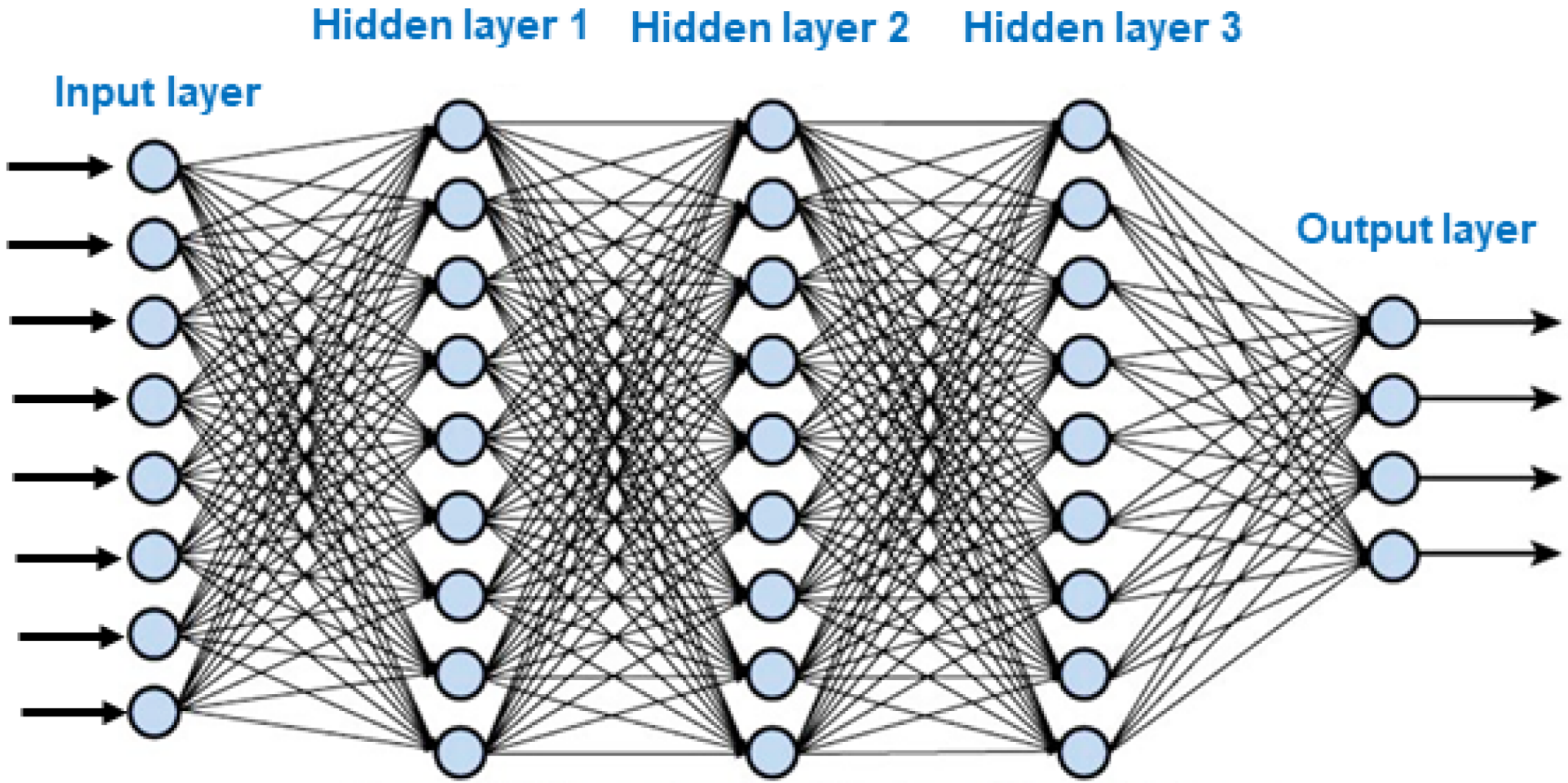
Ключевой принцип: Каждый нейрон в текущем слое соединен с каждым нейроном в следующем слое

Входные данные: Данные должны быть представлены в виде одномерного вектора (1D) Например, изображение 28x28 пикселей разворачивается в вектор из 784 чисел

Слои: Сеть состоит из входного, нескольких скрытых и выходного слоев. Между слоями существуют полные связи

Основная операция — матричное умножение входного вектора на матрицу весов и добавление смещения

Полносвязная нейронная сеть (Fully Connected Network)



Плюсы и минусы полносвязной нейронной сети

Плюсы

- **Универсальность**: Теоретически может аппроксимировать любую функцию (теорема универсальной аппроксимации)

- **Простота**: Легко понять и реализовать

Минусы

- **Огромное количество параметров**: Для изображения 28x28 пикселей между первым (784 нейрона) и вторым (скажем, 128 нейронов) слоем будет 784 * 128 = 100 352 весовых параметров только для одного соединения! Это очень неэффективно

- **Потеря пространственной структуры**: При разворачивании изображения в вектор теряется информация о том, какие пиксели находятся рядом друг с другом

- **Слабая устойчивость к трансформациям**: Сеть, обученная распознавать кошку в центре кадра, может не узнать ту же кошку, сдвинутую вправо на несколько пикселей

Где используется полносвязная нейронная сеть?

- Финальные классифицирующие слои в сверточных сетях
  (после того как CNN извлекла признаки)
- Простые задачи на табличных данных (например, прогнозирование цен)
- Не для обработки изображений, звука или других данных с ярко выраженной пространственной/временной структурой

Свёрточная нейронная сеть (Convolutional Neural Network)

Это специализированный тип нейронных сетей, созданный для работы с данными, имеющими пространственную структуру (изображения, видео, карты)

Ключевой принцип: Использование свёрточных слоев, которые применяют небольшие фильтры (ядра) ко всем областям входного изображения, чтобы извлечь локальные признаки

Аналогия: Теперь вы смотрите на пазл не по одной детали, а небольшими областями (например, по 2x2 детали). Вы ищете локальные паттерны: уголок синего неба, линию горизонта, часть уха животного. Затем вы комбинируете эти паттерны, чтобы понять общую картину.

Почему свёртка?

Проблема полносвязных сетей для изображений:

- Изображение 1000x1000 пикселей = 3 млн входных нейронов (для RGB).
- Слишком много параметров, переобучение, вычислительная сложность.

Решение — Идея свёртки:

- Локальные связи: Нейрон связан не со всеми пикселями, а только с локальной областью (например, 3x3).
- Разделение весов (Weight Sharing): Один и тот же фильтр (ядро) «скользит» по всему изображению.
- Пространственная иерархия: Сначала сети учатся простым признакам (края, углы), затем сложным (текстуры, части объектов, объекты).

Ключевые слои CNN

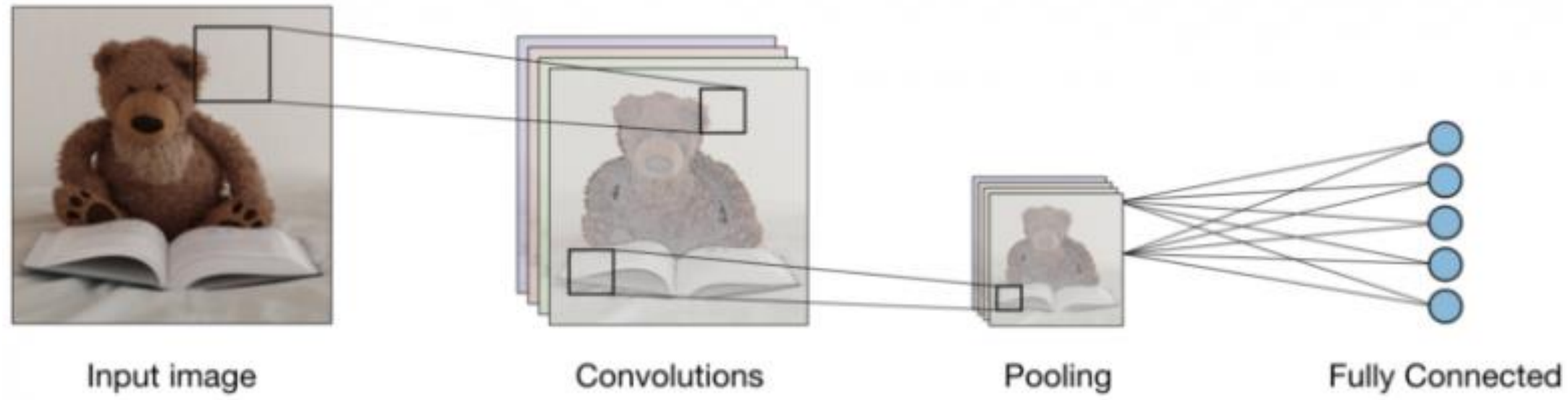
Сверточный слой (Convolutional Layer): Применяет фильтры для извлечения признаков

Слой пулинга (Pooling Layer): Уменьшает размерность карты признаков, сохраняя важную информацию (Max Pooling)

Слой активации (Activation Layer): Добавляет нелинейность (ReLU)

Полносвязный слой (Fully Connected Layer): В конце сети для классификации

Пример. Слои, используемые в CHC для классификации

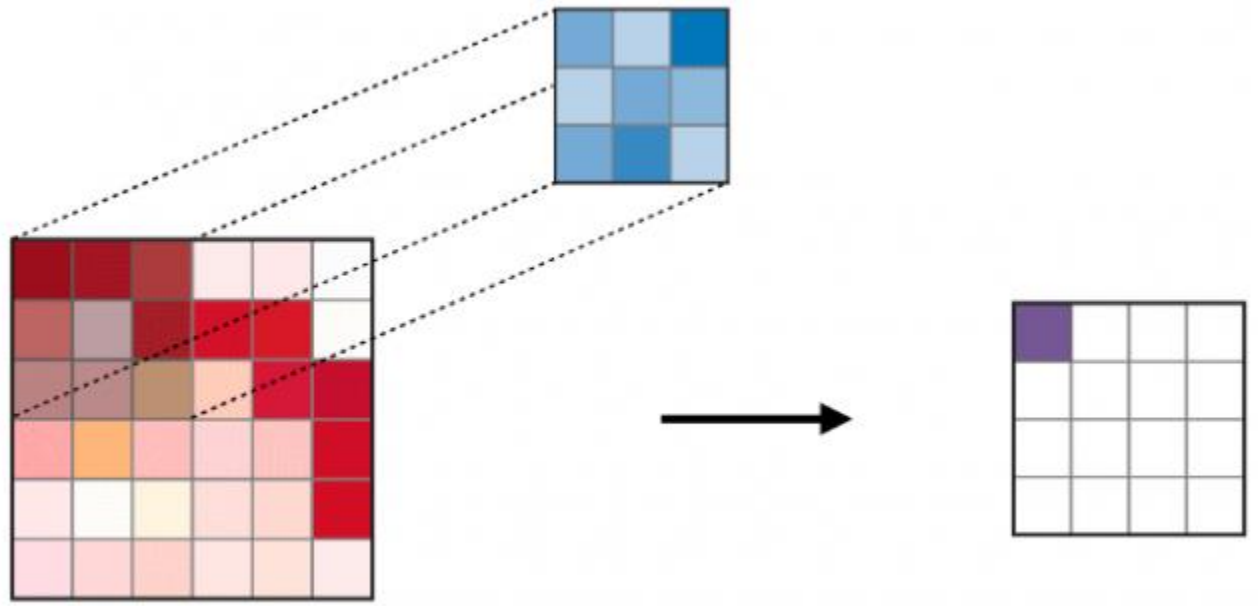


Свёрточный слой (CONV)

Использует фильтры, которые выполняют операции свёртки, сканируя входное изображение

Его гиперпараметры включают в себя размер фильтра, который может быть 2x2, 3x3, 4x4, 5x5 (и т.п.) и шаг S

Каждое ядро обучается распознавать определенный простой признак: границы, углы, пятна цвета и т.д

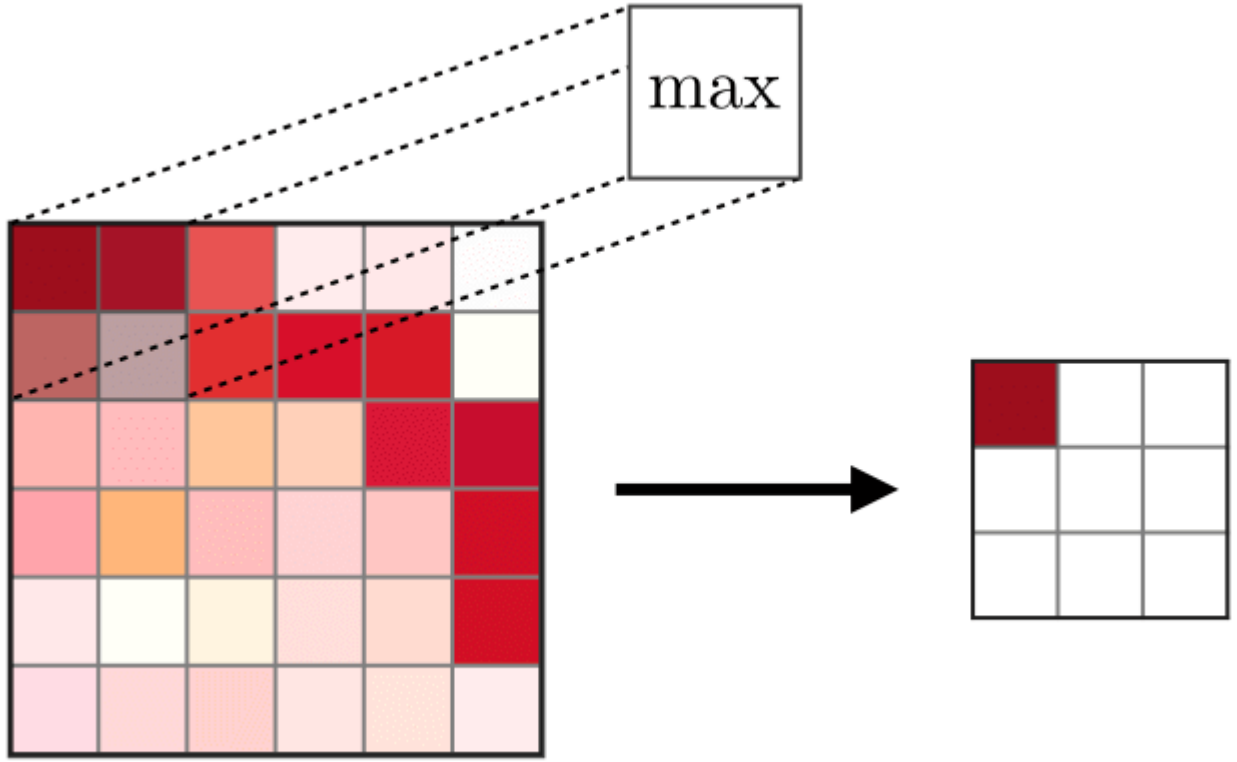


На выходе получается **карта признаков (feature map)**, которая подсвечивает места, где был найден этот признак
Полученный результат называется картой признаков или картой активации, в которой все признаки рассчитаны с использованием входных слоев и фильтров.

Слой объединения (POOL)

Цель: Уменьшить размерность карты признаков, оставив самую важную информацию. Это повышает устойчивость к малым сдвигам и искажения

Как работает:
Чаще всего используется **Max Pooling** - он берет область (например, 2x2) и оставляет только максимальное значение, отбрасывая остальные



Используется для уплотнения признаков, обычно применяемых после слоя свёртки

Существует два типа операций объединения — это максимальное и среднее объединение, где берется максимальное и среднее значение признаков, соответственно

Основные цели пуллинга

Уменьшение вычислительной сложности: Меньшие размеры данных означают меньше параметров и вычислений на последующих слоях сети

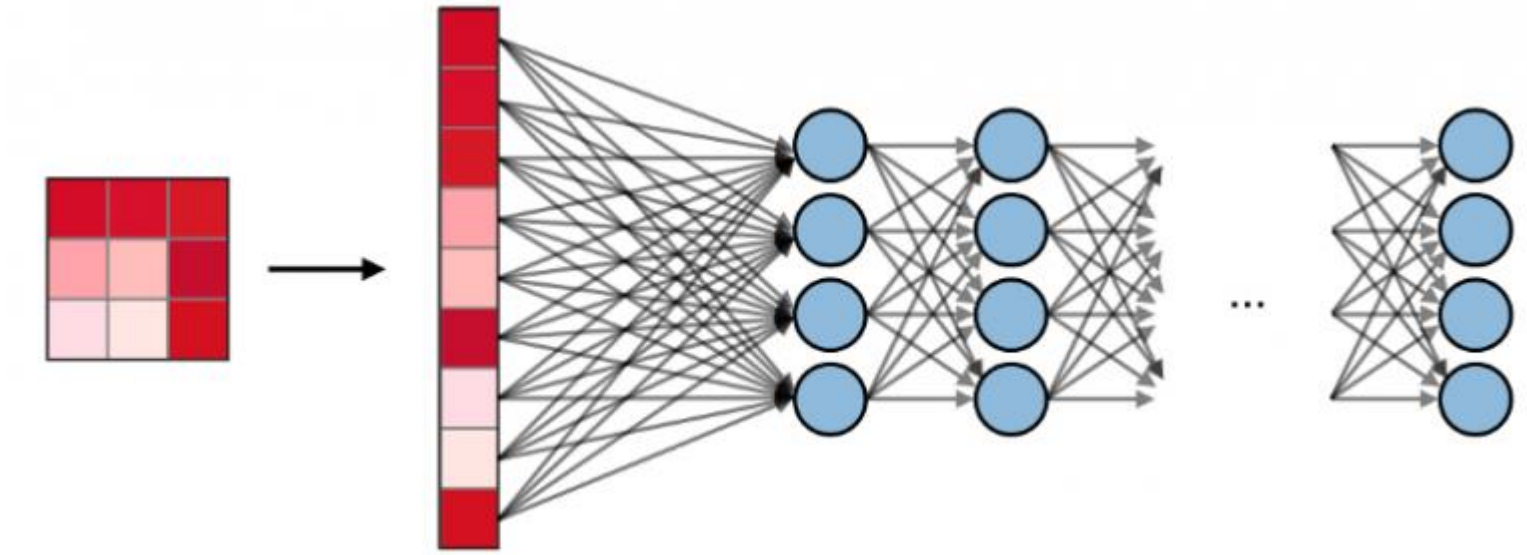
Борьба с переобучением (Regularization): Уменьшая размерность, пуллинг делает представление данных более абстрактным и менее чувствительным к мелким изменениям и шуму в исходных данных. Это помогает сети обобщать лучше и не запоминать тренировочные данные

Инвариантность к небольшим смещениям и искажениям: Пуллинг делает сеть более устойчивой к тому, что объект на изображении немного сдвинут, повернут или искажён. Сеть учится распознавать объект по его ключевым признакам, а не по их точному местоположению

Выделение доминирующих признаков: Пуллинг позволяет сети сосредоточиться на самом главном в определённой области, игнорируя менее значимые детали

Полносвязный слой (FC)

После нескольких свёрточных и пулинговых слоёв высокоуровневые признаки разворачиваются в вектор и подаются на один или несколько полносвязных слоёв для финальной классификации

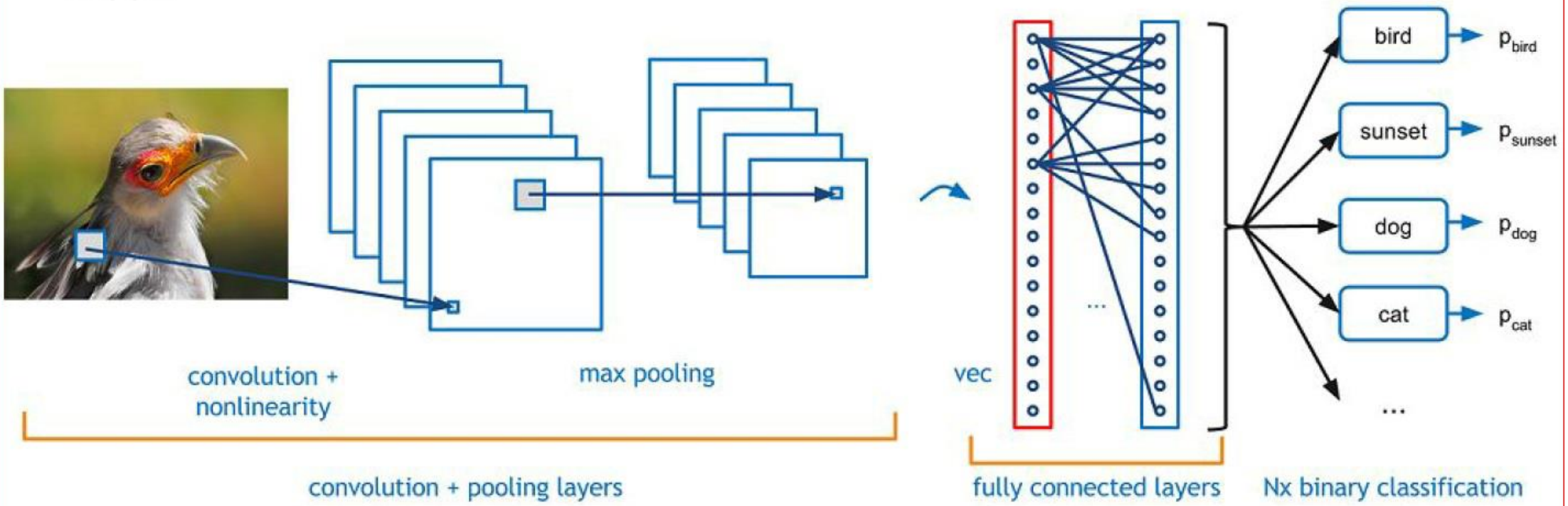


Работает с плоским входом, где каждый вход связан со всеми нейронами

Обычно они используются в конце сети для подключения скрытых слоёв к выходному слою, что помогает оптимизировать оценки классов

Свёрточная нейросеть: общий вид

Свёрточная нейросеть (CNN) — это Feed-Forward сеть специального вида:



Плюсы и минусы свёрточной нейронной сети

Плюсы

- Экономия параметров (Parameter Sharing): Одно и то же ядро используется для всех участков изображения. Не нужно учить отдельный вес для каждой пары пикселей
- Учет пространственной структуры: Не разрушает двумерную структуру изображения
- Устойчивость к трансформациям: Способность распознавать объекты независимо от их положения в кадре (инвариантность к сдвигу)

Минусы

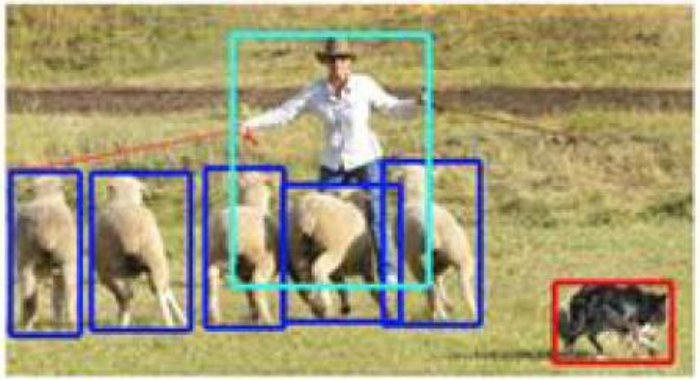
- Более сложная архитектура для понимания
- Специализированы в первую очередь для данных с пространственной структурой

Где используется CNN?

- **Компьютерное зрение**: Распознавание изображений, обнаружение объектов, сегментация
- **Обработка естественного языка**: Можно применять к тексту, представленному как последовательность слов/символов
- **Обработка звуковых сигналов**

Классические задачи для CNN в CV

(a) classification

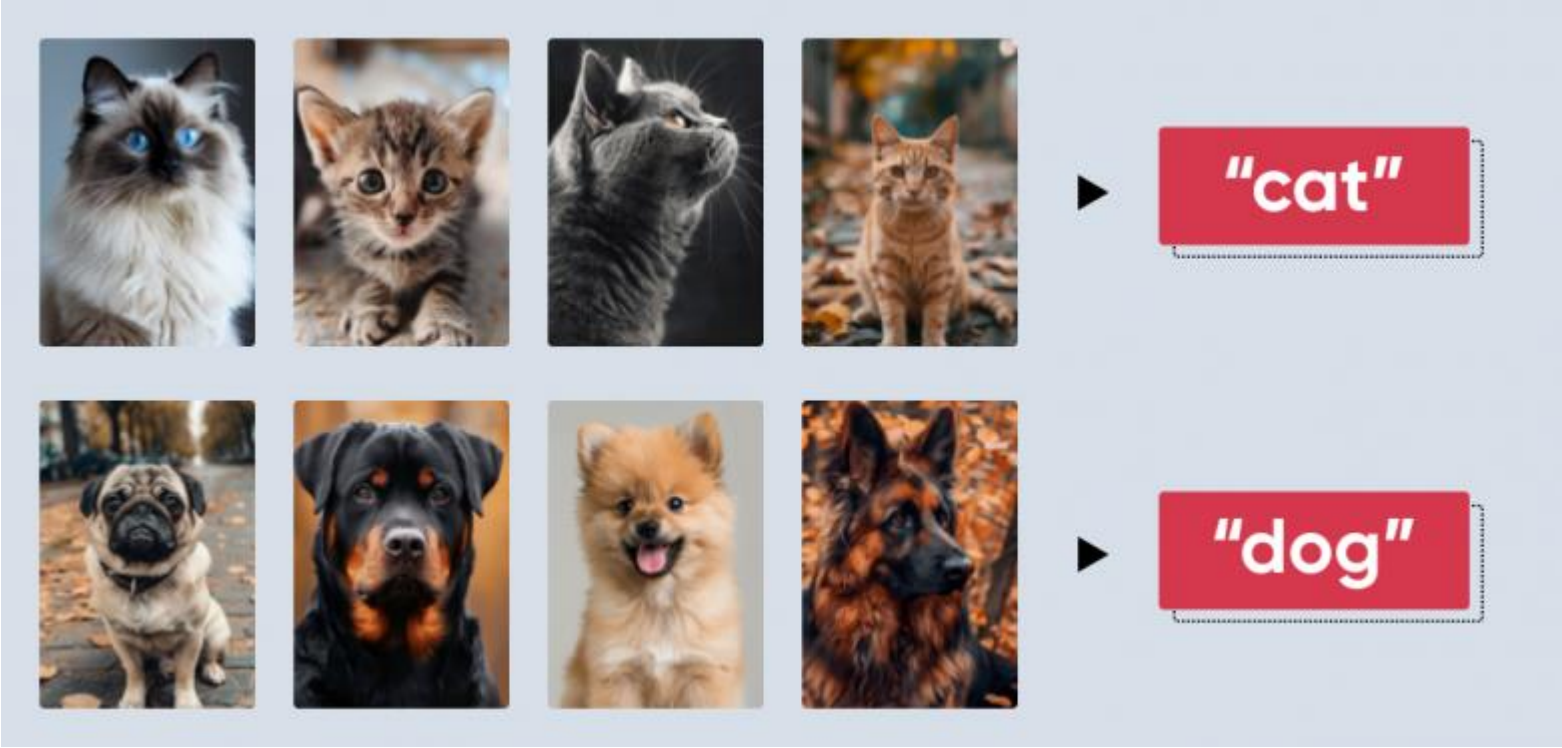


(b) detection

(c) segmentation

Эволюция архитектур для классификации — эра CNN

1. LeNet-5 (1998) — Прародитель
2. AlexNet (2012) — «Большой взрыв» Deep Learning
3. VGGNet (2014) — Сила глубины и простоты
4. GoogLeNet / Inception (2014) — «Ширина и эффективность»
5. ResNet (2015) — Проблема исчезающего градиента и ее решение



Основные сферы деятельности — машинное обучение, компьютерное зрение, мобильная робототехника и вычислительная нейробиология.

Лауреат премии Тьюринга (2018, совместно с Бенжио и Хинтоном за формирование направления глубокого обучения).

Получил докторскую степень по информатике в Университете Пьера и Марии Кюри в 1987 году.

Я. Лекун

8 июля 1960 (65 лет)

Известен работами по применению нейросетей к задачам оптического распознавания символов и машинного зрения.

Разработал серию методов машинного обучения, в том числе свёрточные нейронные сети.

Один из основных создателей технологии сжатия изображений DjVu.

Вместе с Леоном Боту создал язык программирования Lush.

Для реализации механизма автоматическое обучение признаков Я. Лекуном были придуманы свёрточные слои, которые основаны на идее функции свёртки, которая, в свою очередь, очень часто используется в классических алгоритмах компьютерного зрения (БПФ, Гауссово размытие и т.п.).

В последующем они были частично заменены на блоки внимания (self-attention).

LeNet-5 (1998) — Прародитель

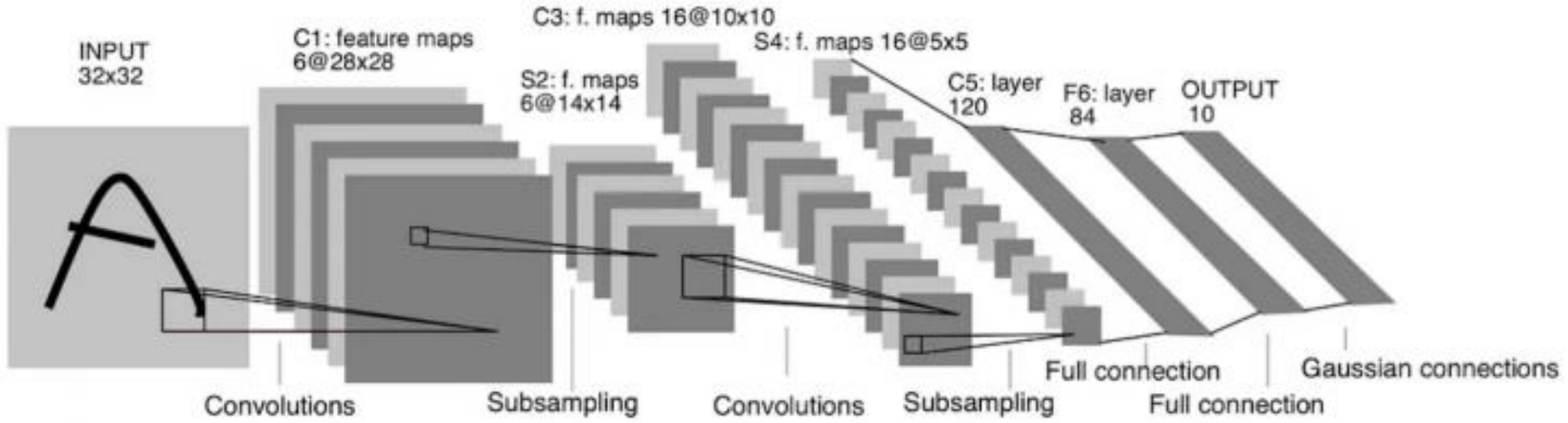
- **Авторы**: Я. Лекун и др.
- **Архитектура**: Conv -> Pool -> Conv -> Pool -> FC -> FC -> Output
- **Значение**: Успешно применялась для распознавания рукописных цифр (MNIST). Доказала работоспособность идеи свёртки. В дальнейшем ее архитектурные принципы были применены и к другим задачам компьютерного зрения, включая распознавание объектов и лиц

Первый прототип (LeNet-1) появился в 1989 году, разработан Яном Лекуном и его коллегами для задачи распознавания рукописных цифр на наборе данных MNIST.

К 1998 году появилась версия LeNet-5, ставшая одним из первых успешных применений нейронных сетей для практических задач

Сверточные и пуллинговые слои могут работать вместе для извлечения признаков и уменьшения размерности изображений, чтобы эффективно распознавать объекты

LeNet-5 (1998)



Input (32x32x1): Изображения приводились к размеру 32x32 пикселя в оттенках серого
Conv1 (C1): 6 ядер свёртки 5x5. Результат: 28x28x6
Avg Pool1 (S2): Усредняющий пулинг 2x2. Результат: 14x14x6
Conv2 (C3): 16 ядер свёртки 5x5. Результат: 10x10x16
Avg Pool2 (S4): Усредняющий пулинг 2x2. Результат: 5x5x16
FC1 (C5): Полносвязный слой на 120 нейронов
FC2 (F6): Полносвязный слой на 84 нейрона
Output: 10 нейронов (по одному на цифру от 0 до 9) с функцией активации tanh вместо современных функций

LeNet-5 доказала работоспособность идеи свёртки

Иерархическое представление признаков:
Первые свертки учатся простым паттернам (края, углы), а последующие — комбинируют их в более сложные (окружности, части цифр).

Эффективное использование параметров:
Веса свертки разделяются (weight sharing), что резко сокращает количество параметров по сравнению с полносвязными сетями и позволяет сети обобщать.

Устойчивость к малым смещениям:
Благодаря пулингу, сеть становится нечувствительной к небольшим сдвигам и искажениям изображения.

Недостатки LeNet-5

Недостатки LeNet-5 связаны с ограниченными вычислительными мощностями:

- сеть работает только с простыми объектами — небольшими изображениями с низким разрешением,
- нет методов предварительной обработки изображений,
- затухающий градиент,
- высокая вероятность переобучения,
- большое потребление памяти.

Почему она была забыта на время (около 15 лет)?

Недостаток вычислительных мощностей:
В 1998 году не было мощных GPU. Обучение даже такой небольшой сети было очень медленным

Недостаток данных:
Не было больших и разнообразных датасетов, подобных ImageNet. MNIST был хорош для проверки концепции, но недостаточен для обучения по-настоящему мощных и устойчивых моделей

Зима искусственного интеллекта:
В 2000-х годах интерес и финансирование нейронных сетей временно пошли на спад. Более простые и эффективные методы, такие как Support Vector Machines (SVM), показывали на тех данных лучшие результаты.

Воскрешение и наследие LeNet-5

LeNet-5 не была забыта навсегда. Ее воскресили в середине 2010-х годов, когда сошлись три фактора:

- Появление больших датасетов (например, ImageNet, 2009 г.)
- Массовое использование GPU для вычислений (пионером был Алекс Крижевский в 2012 г. с AlexNet)
- Развитие алгоритмов (например, использование функции ReLU вместо tanh для борьбы с затуханием градиента)

Прямое наследие LeNet-5 в современных сетях

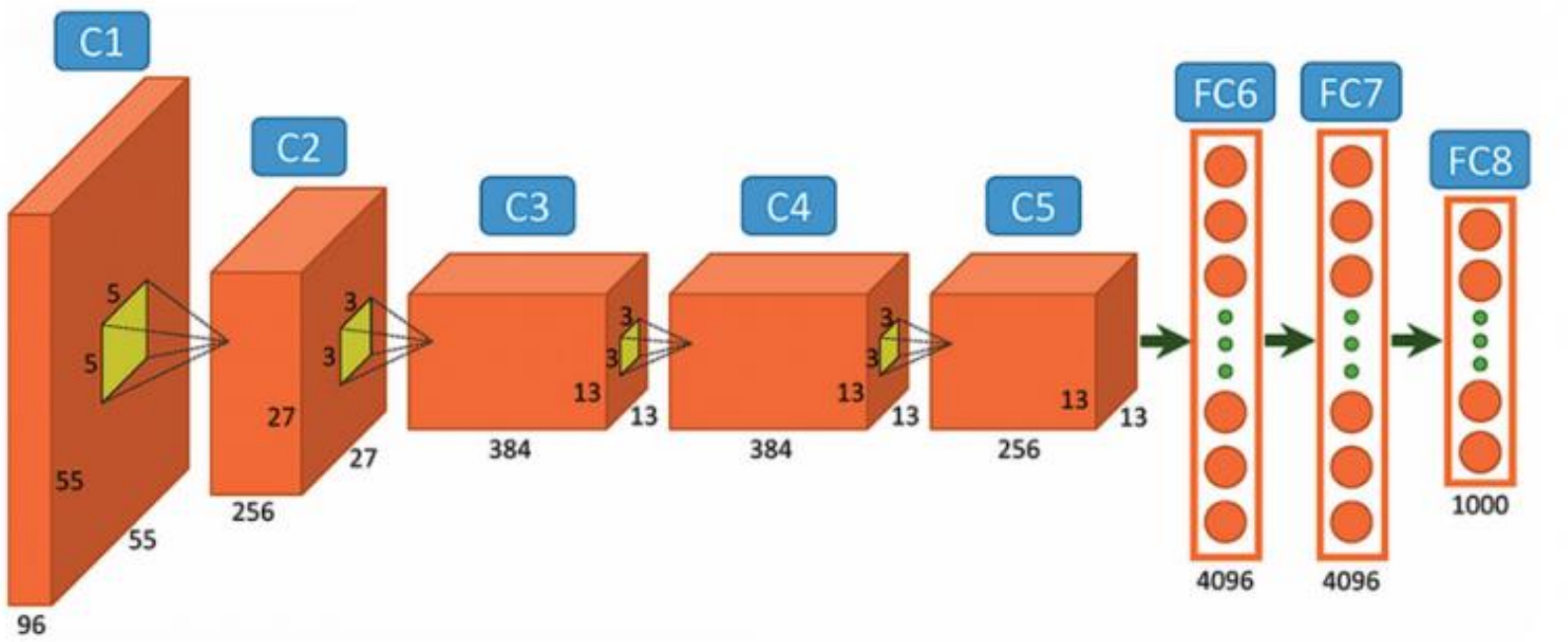
- AlexNet (2012), которая начала революцию глубокого обучения, по сути, является усиленной LeNet: больше слоёв, используется ReLU, Dropout, обучение на GPU
- VGG, GoogLeNet, ResNet и все последующие свёрточные архитектуры — это прямые потомки идей, заложенных в LeNet-5

AlexNet (2012) — «Большой взрыв» Deep Learning

- AlexNet (SOTA 2012) выиграла ImageNet Challenge 2012 (ILSVRC-2012)
- Для конкурса была использована выборка из 1,2 млн изображений с 1000 различными классами, среди которых были как обобщенные категории (например, «кошка», «собака»), так и более узкие (например, конкретные породы кошек или собак)
- Благодаря победе AlexNet в ILSVRC-2012 началась эра глубокого обучения
- Эта архитектура была глубже и сложнее, чем предшествующая LeNet-5

AlexNet

AlexNet использует комбинацию свёрточных слоёв и полносвязных слоёв: 5 свёрточных и 3 полносвязных слоя, с общим количеством 60 млн параметров, пулинг и ReLU-активацию вместо tanh



Для настройки архитектуры разработчики столкнулись с множеством проблем, связанных с подбором параметров, таких как размеры пуллинга, кернелов, страйдов и других гиперпараметров

AlexNet (2012) — Почему прорыв?

1. **Масштаб и Глубина (8 слоёв)**

Контекст: До этого большинство успешных моделей были значительно мельче (как LeNet-5 с 5 слоями, 1998 г.). Попытки обучить глубокие сети часто проваливались

Прорыв: AlexNet не просто была глубокой. Она показала, что глубокие иерархические представления данных могут быть эффективно обучены.
Низкие слои учились распознавать простые края и текстуры, средние — части объектов (глаза, носы), а высокие — целые объекты. Эта иерархия оказалась невероятно мощной.

AlexNet (2012) — Почему прорыв?

2. **ReLU Activation — Ключ к Обучению**

Проблема: Используемые до этого функции активации (**tanh, sigmoid**) являются насыщающимися. При больших входных значениях их градиент стремится к нулю. Во время обратного распространения ошибки градиенты на глубоких слоях становились ничтожно малыми, и обучение останавливалось — это проблема **затухающих градиентов** (vanishing gradient)

Прорыв: **ReLU** (Rectified Linear Unit) — **f(x) = max(0, x)** — является ненасыщающейся функцией. Что позволяет градиентам беспрепятственно течь через многие слои во время обратного распространения

Результат: Обучение стало в 6 раз быстрее по сравнению с использованием tanh (как было указано в оригинальной статье). Это был не просто трюк, а фундаментальное изменение, позволившее тренировать действительно глубокие сети

AlexNet (2012) — Почему прорыв?

3. **Dropout — Борьба с Переобучением**

Проблема: Сложная модель с миллионами параметров легко запоминает обучающие данные вместо того, чтобы учиться обобщать. Это **переобучение** (overfitting)

Прорыв: **Dropout** — техника, при которой во время обучения случайно выключается (обнуляется) определённый процент нейронов в каждом предъявлении примера
Это предотвращает ситуацию, когда нейроны становятся слишком зависимыми друг от друга и коадаптируются
Каждый нейрон вынужден учиться полезным и независимым признакам

Результат: Модель стала гораздо лучше обобщать и показывать **высокую точность на новых**, невиданных данных (тестовой выборке ImageNet)

AlexNet (2012) — Почему прорыв?

4. **Использование GPU (NVIDIA GTX 580)**

Контекст: Обучение такой сети на CPU заняло бы недели или месяцы, делая эксперименты непрактичными

Прорыв: Авторы (Алекс Крижевский, Илья Суцкевер и Джеффри Хинтон) использовали **вычисления на GPU**. GPU идеально подходят для параллельных матричных и свёрточных операций, которые составляют основу CNN

Результат: Они смогли обучить огромную модель на огромном датасете (ImageNet) за разумное время (5-6 дней)
Это доказало, что аппаратное ускорение — не опция, а необходимость для современного deep learning, и запустило золотую лихорадку в индустрии, связанную с GPU

Дополнительные критические факторы

- **Датасет ImageNet**: Без большого и сложного датасета (1.2 млн изображений, 1000 классов) не было бы и прорыва

- **Аугментация**: Использует разные методы аугментации данных (случайные обрезки, повороты и зеркальные отражения изображений), что увеличивает количество обучающих данных и улучшает обобщающую способность модели

- **Max-pooling:** В отличие от LeNet-5, которая использует Average-Pooling, AlexNet использует max-pooling, что позволяет лучше захватывать важные признаки, такие как границы и текстуры, и делает сеть более инвариантной к сдвигам и деформациям объектов

- **Перекрывающийся пулинг** (Overlapping Pooling): Сделало извлечение признаков более устойчивым к малым смещениям и искажениям изображения

Дополнительные критические факторы

Локальная нормализация ответа (Local Response Normalization):

- В отличие от Batch Normalization, LRN нормализует активность нейронов на основе активности соседних нейронов в пределах одного слоя, чтобы улучшить контрастность и стабилизировать обучение, особенно в глубоких сетях
  Помогала нейронам конкурировать за активацию, усиливая признаки, которые сильнее активируются в своём окружении

- Увеличивает скорость сходимости алгоритма

- Сейчас, в основном, применяется BatchNormalization и LayerNormalization, так как эти способы способны более стабилизировать сходимость алгоритма

Динамическое управление коэффициентом скорости обучения:
Коэффициент делится на 10, когда качество перестает улучшаться на валидационной выборке

Недостатки AlexNet

- Большое число параметров (60 млн) требует значительных вычислительных ресурсов и памяти
- Высокий риск переобучения
- Сохраняется проблема затухающих градиентов
- Большое потребление ресурсов
- Ограниченная глубина сети, что ограничивает способность извлекать сложные иерархические признаки из изображений
- Фиксированный размер изображения на входе, что может требовать дополнительных операций по изменению размера изображения, иногда приводя к потере информации

Итог: Почему именно «Большой Взрыв»?

AlexNet не предложила ничего принципиально нового с теоретической точки зрения (свёртки, обратное распространение были известны десятилетиями)

Её гений был в инженерной интеграции и масштабировании известных идей:

- Взяли известную архитектуру (CNN) -> сделали ее гораздо глубже и больше
- Взяли известную функцию активации (ReLU) -> применили ее для ускорения глубоких сетей
- Столкнулись с переобучением -> применили эффективный метод регуляризации (Dropout)
- Столкнулись с вычислительной сложностью -> использовали мощь GPU

Их победа в ImageNet 2012 года с ошибкой в 15.3% (второе место имело ошибку 26.2%) была ошеломляющей и неоспоримой.

VGG NET (2014)

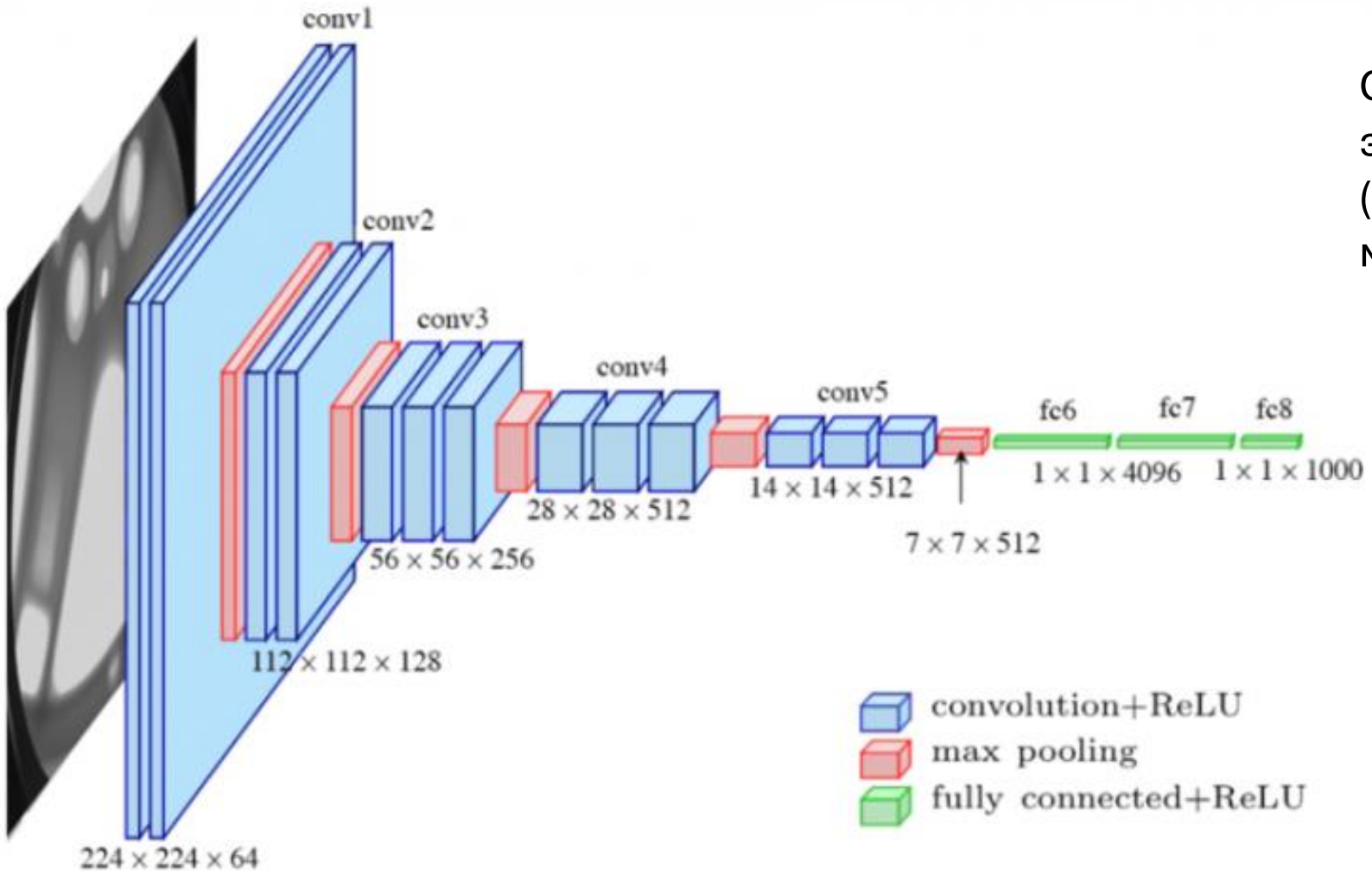
Архитектура, предложенная исследовательской группой Visual Geometry Group Оксфордского университета

Участвовала в ILSVRC 2014 года и заняла второе место, уступив лишь GoogleNet

VGG NET состоит из свёрточных фильтров 3×3, имеет несколько версий, которые отличаются количеством слоёв. Наиболее известные:
VGG-16 (13 свёрток и 3 полносвязных слоя) и VGG-19 (16 свёрток и 3 полносвязных слоя)

Каскадная архитектура свёрток в VGG подразумевает использование **нескольких последовательных свёрточных слоёв**, которые работают вместе для извлечения **более сложных признаков** на разных уровнях абстракции

VGG NET (2014)



Основная философия VGG — это замена крупных фильтров (5x5, 7x7) на стеки из маленьких свёрток 3x3

VGG NET — Почему прорыв?

То же рецептивное поле:

Один свёрточный слой 5x5 имеет рецептивное поле 5x5

Два слоя 3x3 имеют рецептивное поле 5x5. (Первый слой смотрит на 3x3, второй слой смотрит уже на эту область 3x3, но с учётом соседних ячеек, что в сумме дает 5x5)

Три слоя 3x3 заменяют рецептивное поле 7x7

VGG использует небольшие фильтры размером 3×3 во всех свёрточных слоях, что позволяет захватывать детали изображения более точно

Кроме того, последовательное применение нескольких таких слоев с небольшими фильтрами эквивалентно применению большего фильтра (например, 5×5 или 7×7), но с меньшим количеством параметров

Это отличается от AlexNet, где использовались более крупные фильтры, такие как 11×11 в первом сверточном слое

VGG NET — Почему прорыв?

Меньше параметров (Вычислительная эффективность):

Количество параметров для свёртки рассчитывается как
(input_channels * kernel_width * kernel_height * output_channels) + bias

Один слой 5x5 с C каналами: $C * 5 * 5 * C = 25C^2$ параметров

Два слоя 3x3 с C каналами: $(C * 3 * 3 * C) + (C * 3 * 3 * C) = 18C^2$ параметров

Экономия: (25-18)/25 * 100% = 28% меньше параметров для той же зоны охвата!
Это делает сеть более эффективной

VGG NET — Почему прорыв?

Больше нелинейностей:

Один слой 5x5 имеет только одну функцию активации (ReLU)

Два слоя 3x3 имеют две функции активации ReLU

Это делает модель более мощной и выразительной, так как она может обучаться более сложным нелинейным функциям

Архитектура: Однородность как принцип

Архитектура VGG невероятно проста и единообразна, что является ее второй сильной стороной

Блоки: Сеть состоит из повторяющихся блоков

Свёртки: В каждом блоке несколько свёрточных слоев 3x3 с шагом 1 и дополнением (padding) 1 (что сохраняет размерность feature map)

Пулинг: После блока свёрток следует один слой Max-Pooling 2x2 с шагом 2 (уменьшает размер feature map в 2 раза). Этот подход схож с AlexNet, но VGG применяет pooling более регулярно

Классификатор: В конце сети идут три полностью связанных (FC) слоя, первые два с 4096 нейронами каждый, и последний выходной с 1000 нейронами (по числу классов ImageNet)

Наиболее известные конфигурации — VGG16 и VGG19 (цифра означает количество весовых слоёв: свёрточных и полносвязных)

Другие особенности VGG NET

Отсутствие нормализации LRN
В отличие от AlexNet, VGG не использует Local Response Normalization (LRN), что упрощает архитектуру и уменьшает вычислительные затраты без значительных потерь в производительности

Объём данных и вычислительные ресурсы
VGG была обучена на наборе данных ImageNet, что требовало значительных вычислительных ресурсов, особенно из-за большого числа слоёв и параметров

Количество параметров
VGG16 имеет примерно 138 миллионов параметров (VGG19 имеет 144 миллиона параметров), что делает её очень требовательной к памяти и вычислительным ресурсам. Для сравнения, AlexNet имеет около 60 миллионов параметров

Недостатки VGG NET

Высокие вычислительные затраты и использование памяти
VGG имеет большое количество параметров из-за использования многих свёрточных слоев с 3×3 фильтрами. Это приводит к высокому потреблению ресурсов и долгому времени обучения, особенно на крупных наборах данных

Долгое время предсказания
Из-за глубины и количества параметров предсказания на VGG моделях могут быть медленными, что делает их неидеальными для приложений, требующих реального времени

Переобучение
Большое количество параметров может привести к переобучению на небольших датасетах, если не применять регуляризацию

Недостатки VGG NET

Большой размер модели
VGG модели занимают много места в памяти. Например, VGG16 весит более 500 МБ, что неудобно для интеграции в мобильные устройства или приложения с ограниченными ресурсами

Устаревшие методы оптимизации
Новые модели, такие как ResNet, использующие остаточные соединения, имеют более высокую точность при меньших вычислительных затратах и меньшем количестве параметров, что делает их более предпочтительными в современных приложениях

Значение VGG NET

Доказательство важности глубины
VGG наглядно показала, что увеличение глубины сети (до 16-19 слоев) при правильной конструкции приводит к значительному улучшению точности. Она стала прямым предшественником еще более глубоких сетей

Элегантность и простота
Её однородная структура сделала архитектуру очень простой для понимания, реализации и модификации

Золотой стандарт для Transfer Learning
Несмотря на то, что современные архитектуры (например, ResNet, EfficientNet) превосходят VGG по точности и эффективности, предобученные модели VGG до сих пор чрезвычайно популярны

Почему предобученные модели VGG NET популярны?

Причины

Простая структура: Ее легко взять и адаптировать

Хорошо изученные фичи: Особенности (feature maps), извлеченные VGG, оказались очень универсальными и полезными для других задач компьютерного зрения

Широкая доступность: Она встроена во все основные фреймворки глубокого обучения

VGG часто используется как базовая сеть для задач сегментации, детекции объектов, стилизации изображений и т.д.

GoogLeNet: Проблема, которую решали авторы

До 2014 года **доминирующей тенденцией в глубоком обучении** для компьютерного зрения было **увеличение глубины** сетей (количества слоёв), как в VGG, **или ширины** (количества фильтров в слое)

Глубина: Более глубокие сети могут изучать более сложные иерархические особенности

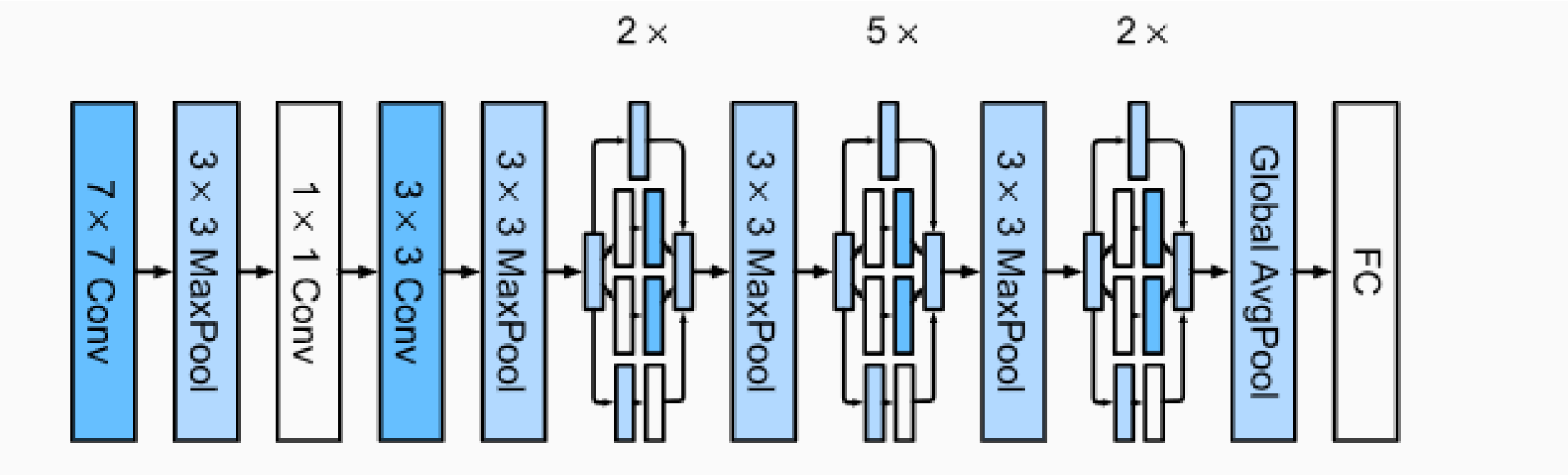
Ширина: Более широкие сети могут изучать больше разнообразных признаков на одном уровне

Однако это вело к **двум ключевым проблемам**:

Число параметров: Огромное количество весов (например, в VGG-16 ~138 миллионов параметров), что требовало много памяти и вычислительной мощности

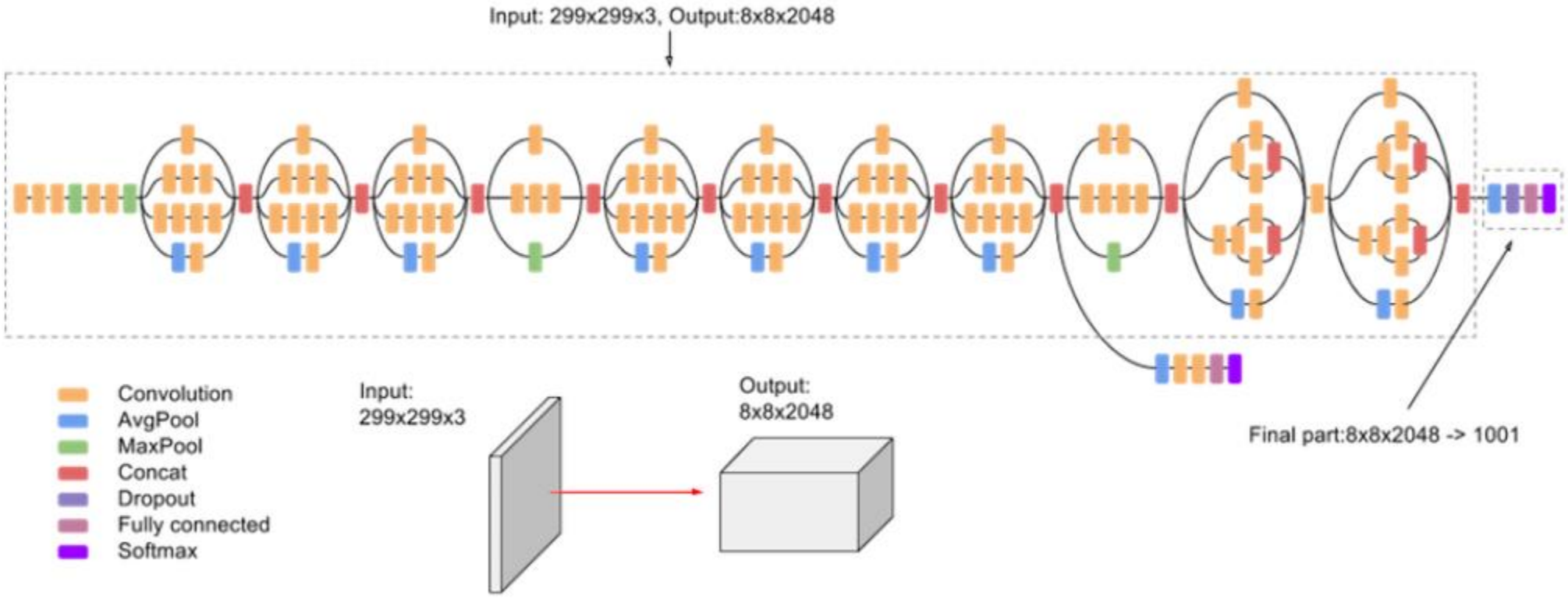
Переобучение: Сеть с большим количеством параметров легко запоминает тренировочные данные вместо того, чтобы учиться обобщать, особенно при ограниченном объеме данных

GoogLeNet/Inception (2014)



- Архитектура, разработанная Google
- Использует модуль Inception, основная идея которого заключается в комбинировании различных типов свёрток с разными размерами фильтров и пуллинга для более эффективного извлечения признаков
- Победитель ILSVRC 2014

GoogLeNet/Inception (2014)



Решение: Inception-модуль

Идея, предложенная Кристианом Сегеди и его командой, была гениальной в своей простоте: вместо того чтобы выбирать один размер фильтра для свёрточного слоя, используй их все одновременно.

Базовая (наивная) идея Inception-модуля:

Что если мы подадим входной тензор параллельно на свёртки 1x1, 3x3, 5x5 и операцию пулинга (усредняющего или максимального), а затем просто объединим (конкатенируем) все полученные карты признаков вдоль оси каналов?

Решение: Inception-модуль

Это позволяет сети на одном уровне изучать:

- Локальные особенности с помощью 1x1 сверток
- Особенности среднего масштаба с помощью 3x3 сверток
- Более глобальные особенности с помощью 5x5 сверток
- Инвариантные к малым сдвигам особенности с помощью пулинга

Сеть сама в процессе обучения решает, каким комбинациям фильтров доверять больше

Ключевое усовершенствование: Бутылочное горлышко (Bottleneck)

Наивная реализация была бы вычислительно неэффективной
Например, если на вход приходит 256 каналов, то применение 128 сверток 5x5 к ним приведёт к огромному числу операций

Гениальное решение:
Использовать свёртки 1x1 перед свертками 3x3 и 5x5 и после пуллинга

Для чего это нужно?

Уменьшение размерности (Reduction): Свёртка 1x1 позволяет уменьшить количество каналов (глубину тензора), прежде чем применять к нему дорогие свёртки 3x3 и 5x5

Пример: Было 256 каналов. Применяем 64 свёртки 1x1 -> получаем 64 канала. Теперь применяем к этим 64 каналам 128 свёрток 3x3. Число вычислений сократилось в разы по сравнению с применением 128 свёрток 3x3 сразу к 256 каналам.

Увеличение выразительности: Несмотря на уменьшение размерности, свёртки 1x1 сами по себе являются линейными преобразованиями, за которыми следует нелинейность (ReLU), что добавляет выразительную силу сети

Таким образом, **Inception-модуль** становится не просто параллельным, а умным, эффективным блоком, который сначала сжимает данные, затем извлекает признаки разного масштаба и объединяет их.

Другие важные инновации GoogLeNet

Вспомогательные классификаторы (Auxiliary Classifiers):
Для борьбы с проблемой исчезающего градиента в очень глубокой сети, в промежуточные слои были добавлены две дополнительные ветки классификации.
Их ошибка (loss) добавлялась к основной ошибке на выходе, что помогало проталкивать градиент в начальные слои во время обучения.
(Позже выяснилось, что их основная польза — в роли регуляризатора)

Полносвязные слои:
Вместо гигантских полносвязных слоёв в конце (как в AlexNet/VGG), GoogLeNet использует простой глобальный средний пулинг (Global Average Pooling), что радикально сокращает количество параметров

Глубина сети:
Первая версия Inception v1 (GoogleNet) содержит 22 слоя, что делает её глубже, чем AlexNet и даже VGG, при этом структура отличается благодаря инновационному использованию Inception-блоков

Недостатки GoogLeNet

- Модульная структура и разнообразие фильтров делает модель сложной для реализации, настройки и отладки
- Сложности с распараллеливанием операций
- Затухающие градиенты
- Сложности с масштабированием сети. Из-за сложной структуры блоков Inception модификация архитектуры, например, для адаптации к специфическим задачам, может быть затруднена
- Хотя GoogleNet оптимизировала использование ресурсов по сравнению с VGG, она всё равно требует значительных вычислительных ресурсов, особенно для обучения. Каждый блок Inception включает множество параллельных операций, что приводит к дополнительной нагрузке на память и процессор
- Из-за сложной архитектуры сети обучение может быть сложным, и для достижения наилучших результатов требуется тонкая настройка гиперпараметров

Значение и Итоги

Эффективность:
GoogLeNet достигла нового state-of-the-art на ImageNet с в 12 раз меньшим количеством параметров, чем у AlexNet (~6.8 млн против ~60 млн)

Влияние:
Inception-архитектура стала классикой. Её идеи легли в основу многих последующих моделей, а сама архитектура эволюционировала в серию (Inception v2, v3, v4, Inception-ResNet)

Принцип:
GoogLeNet доказала, что не только глубина, но и продуманная внутренняя структура (архитектурные инновации) являются ключом к созданию более мощных и эффективных моделей

GoogLeNet заменила парадигму «глубже и шире» на парадигму «умнее и эффективнее»

ResNet (2015)

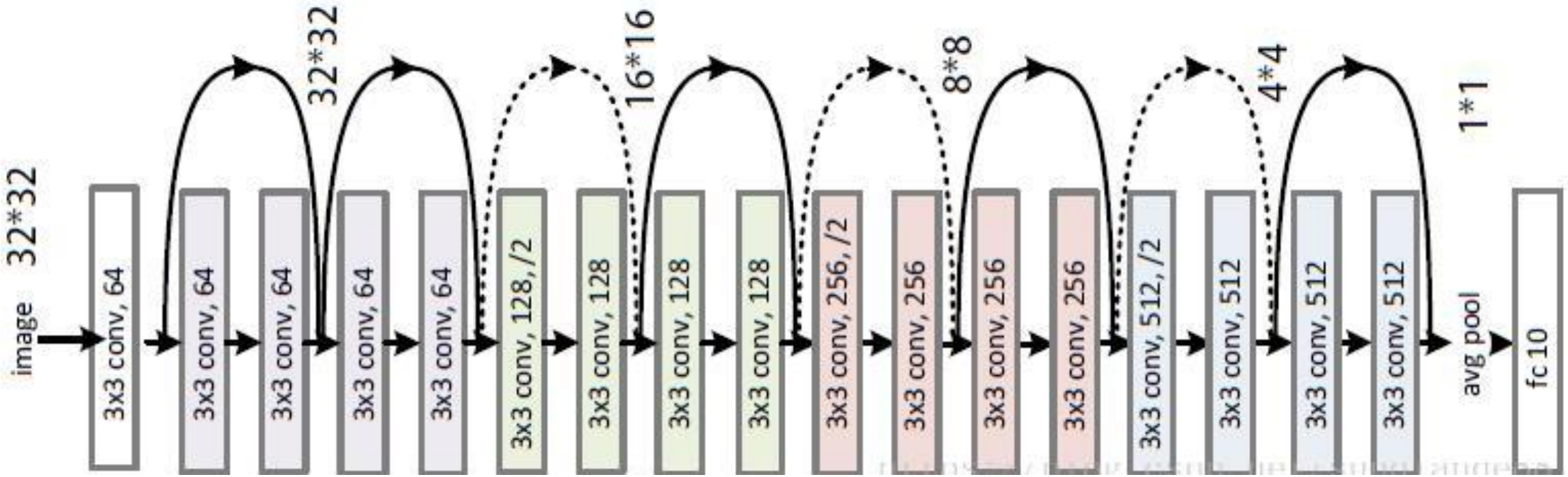
ResNet (Residual Network) — это архитектура свёрточных нейронных сетей, разработанная исследовательской группой Microsoft под руководством К. Хэ (Kaiming He), Х. Жаня (Xiangyu Zhang), С. Жэня (Shaoqing Ren) и Д. Суна (Jian Sun) в 2015 году.

Использует остаточные блоки (Residual Blocks), которые позволяют строить сверхглубокие сети (до 152 слоев) без проблемы затухающих градиентов. Завоевала первое место на конкурсе ILSVRC 2015

ResNet стала одной из самых значимых архитектур в области глубокого обучения, поскольку она решает проблему обучения очень глубоких сетей

Стала основой для многих других архитектур, таких как ResNeXt, DenseNet и EfficientNet, которые также используют остаточные блоки и прямые переходы для достижения лучшей производительности.

Архитектура ResNet



Проблема «Исчезающего градиента» (Vanishing Gradient)

Что происходит?
В очень глубоких сетях при обратном распространении ошибки (backpropagation) градиент (производная функции потерь по весам) вычисляется с помощью цепного правила. Это означает **многократное перемножение производных каждого слоя**

Почему это плохо?
Если эти производные (например, от функций активации вроде сигмоиды) меньше 1, их произведение для слоёв в начале сети становится экспоненциально малым.
В результате **веса начальных слоёв практически не обновляются**, и сеть их не обучает. Фактически, только последние слои получают существенные обновления

Это **не переобучение** (проблема обобщения), а **проблема оптимизации** — сеть просто не может научиться использовать все свои слои

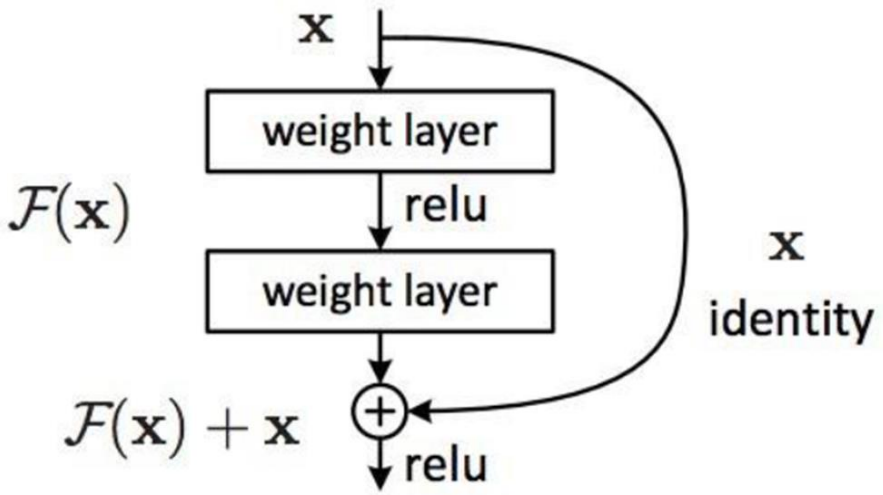
Решение: Остаточные блоки (Residual Blocks)

Гипотеза разработчиков:
Предположим, у нас есть оптимальное отображение для блока слоёв, которое мы хотим получить — H(x)
Вместо того чтобы заставлять несколько нелинейных слоёв напрямую аппроксимировать H(x), мы заставляем их **аппроксимировать остаток** (residual) — F(x) = H(x) - x

Итоговое отображение: Таким образом, исходное отображение превращается в H(x) = F(x) + x

Ключевой момент:
Сеть теперь **учится** не полной функции, а лишь **отклонению** от тождественного отображения (identity mapping)
Если оптимальная функция для блока — это просто пропустить вход дальше H(x) = x, то сети очень легко обнулить веса и смещения в остаточном блоке, чтобы получить F(x) = 0, и тогда H(x) = x
Это значительно проще, чем аппроксимировать тождественную функцию с помощью нескольких нелинейных слоёв

Реализация: Skip-Connection (Пропуск соединения)



- **Прямой путь** (Plain Path):
  Вход x проходит через свёрточные слои, функции активации (ReLU) и т.д., превращаясь в F(x)

- **Шорткат** (Shortcut/Skip-Connection): Вход x перепрыгивает через все эти слои и просто прибавляется к результату F(x)

- **Функция активации**: Обычно применяется после сложения: y = ReLU(F(x) + x)

Следствие: Прямое распространение градиента

Skip-connection создает альтернативный, беспрепятственный путь для градиента.

- При прямом проходе: Информация может течь как через свёрточные слои, так и по шорткату. Это помогает сохранить информацию от исходного входа

- При обратном проходе: Градиент от функции потерь теперь может напрямую течь назад по skip-connection'y. Цепное правило для блока выглядит так:

  $$dL/dx = dL/dy * (dF/dx + 1)$$

  Даже если производная dF/dx станет очень маленькой (исчезающий градиент в свёрточных слоях), у нас всегда есть +1. Это гарантирует, что градиент никогда не затухнет полностью — он всегда сможет пройти назад через шорткат и обновить веса более ранних слоёв.

Именно это позволило успешно обучить сети невиданной ранее глубины: ResNet-34, ResNet-50, ResNet-101, ResNet-152 и даже более глубокие

Недостатки ResNet

Требуются значительные вычислительные ресурсы и память, особенно для ResNet-101 и ResNet-152. При этом увеличение числа слоёв не всегда улучшает производительность, а может даже снижать точность модели или не давать значительных улучшений. Например, переход от ResNet-101 к ResNet-152 дает лишь незначительный прирост точности при большом увеличении вычислительных затрат

Переобучение на глубоких уровнях

Требуется сложное проектирование

С увеличением глубины сети сложнее подобрать оптимальные гиперпараметры для обучения, такие как скорость обучения и стратегия регуляризации. Параметры, подходящие для более мелких сетей, могут не работать эффективно для очень глубоких

Усложнение архитектуры с использованием пропускных соединений делает анализ внутренней работы сети и интерпретацию её решений сложнее по сравнению с более простыми архитектурами

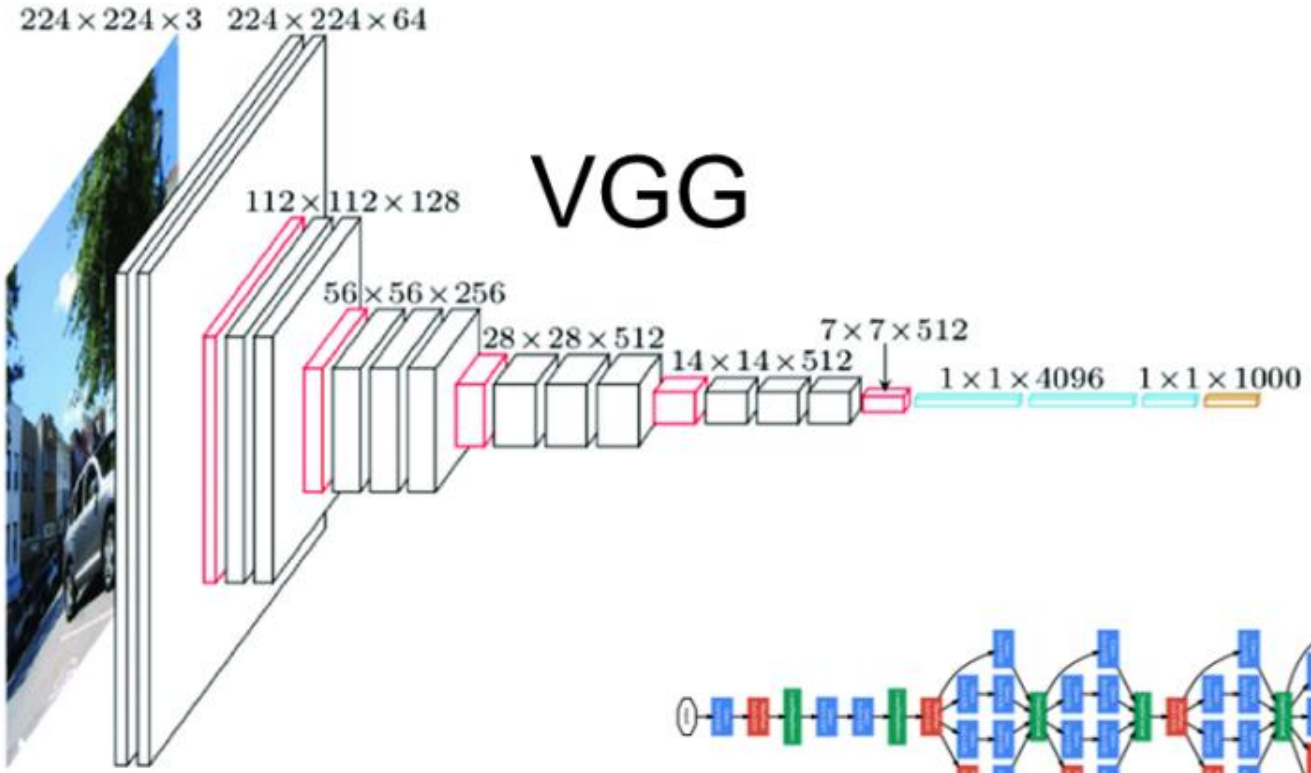
Значение и наследие ResNet

- **Слом барьера глубины**:
  ResNet доказал, что глубокие сети (100, 1000+ слоев) можно эффективно обучать

- **Новый стандарт архитектуры**:
  Остаточные связи стали неотъемлемым компонентом почти всех последующих современных архитектур (DenseNet, Inception-ResNet, EfficientNet и многие другие)

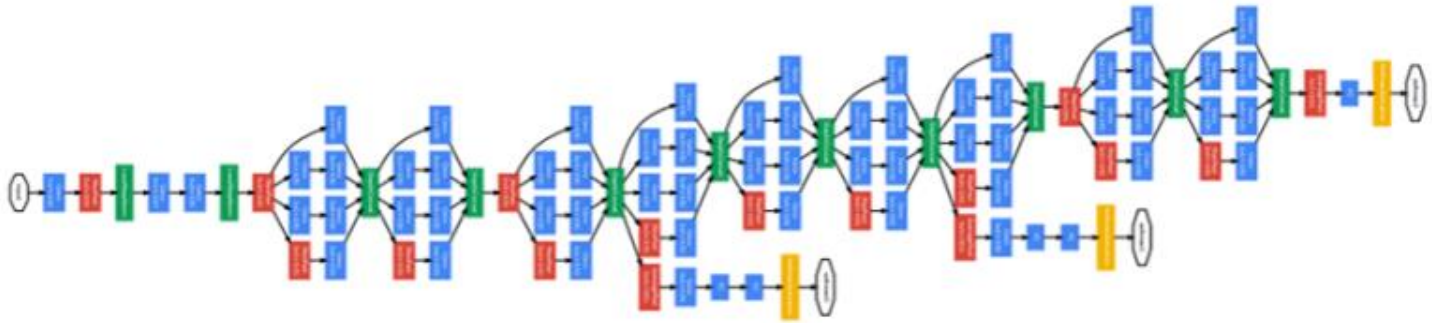
- **Интуитивное объяснение**:
  ResNet интуитивно понятен — сеть учится вносить небольшие поправки F(x) к уже имеющемуся представлению x, а не каждый раз заново изобретать его

- **Практическое доминирование**:
  Модели на основе ResNet долгое время доминировали в соревнованиях по компьютерному зрению (например, ImageNet) и до сих пор используются как мощные основы для многих задач (детекция объектов, сегментация)



VGG

GoogLeNet



ResNet