Обзор нейронных сетей для компьютерного зрения

Общая схема обучения

Процесс можно упростить до следующей схемы:

Данные → Архитектура модели → Функция потерь → Оптимизатор → Обучение → Оценка

Типы задач и данных

- Классификация: У каждого изображения есть одна метка (например, «кошка», «собака»).
  Набор данных **ImageNet** — классический пример

- Обнаружение объектов (Object Detection): Нужно не только классифицировать, но и найти объекты на изображении, указав их координаты в bounding box.
  Примеры наборов данных: **COCO, PASCAL VOC**

- Семантическая сегментация: Присвоить каждый пиксель изображения определенному классу («дорога», «дерево», «человек»).
  Пример набор данных : **Cityscapes**

- Сегментация экземпляров (Instance Segmentation): Как семантическая сегментация, но разные объекты одного класса различаются (человек 1, человек 2, человек 3).
  Подходит набор данных: **COCO**

Предварительная обработка данных (Data Preprocessing)

- **Изменение размера** (Resizing): Все изображения приводят к одному размеру, который ожидает модель на входе (например, 224x224 пикселей)

- **Нормализация**: Значения пикселей (обычно от 0 до 255) масштабируют, например, до диапазона [0, 1] или [-1, 1]. Часто используют стандартизацию по каналам (RGB), вычитая среднее значение и деля на стандартное отклонение. Это ускоряет сходимость

- **Аугментация данных** (Data Augmentation): Критически важный прием! Чтобы модель не запоминала данные, а училась общим признакам, исходные изображения искусственно искажают:
  - Геометрические: повороты, отражения, сдвиги, масштабирование
  - Цветовые: изменение яркости, контрастности, насыщенности
  - Более сложные: вырезание частей (CutOut, MixUp), что заставляет модель смотреть на объект целиком, а не на один признак

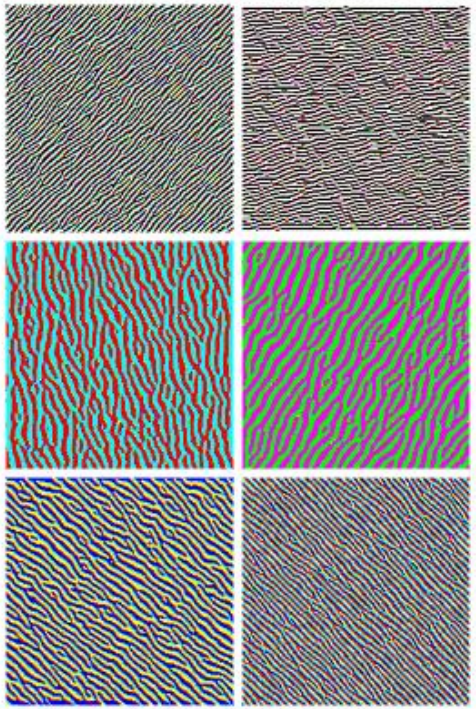
Архитектуры нейронных сетей. Ключевые идеи CNN

- **Локальные связи**: Нейроны обрабатывают не всё изображение сразу, а небольшие области (как бы скользящим окном). Это эффективно и позволяет находить локальные паттерны

- **Разделение весов** (Weight Sharing): Один и тот же фильтр (ядро свертки) применяется ко всему изображению. Это резко сокращает количество параметров и позволяет находить один и тот же признак (например, вертикальный край) в любой части картинки

- **Пространственная иерархия**: Слои идут от простого к сложному
  - Первый слой: учится обнаруживать простые края, градиенты, цвета
  - Средние слои: комбинируют края в более сложные текстуры и геометрические формы (углы, окружности)
  - Глубокие слои: комбинируют формы в части объектов (глаз, колесо, ухо) и целые объекты

Пространственная иерархия: Слои идут от простого к сложному



Края (слой conv2d0)

Текстуры (слой смешанный 3a)

Узоры (слой смешанный4a)

Части (слои смешанные4b и смешанные4c)

Объекты (слои смешанные4d и смешанные4e)

Ключевые архитектуры CNN

- **LeNet-5** (1998): Пионер для распознавания цифр
- **AlexNet** (2012): Прорыв, который показал мощь GPU и глубоких CNN на ImageNet
- **VGG** (2014): Показала важность глубины сети
- **GoogLeNet/Inception** (2014): Представила модули Inception, позволяющие эффективно считать свертки разного размера внутри одного слоя
- **ResNet** (2015): Решила проблему исчезающего градиента в очень глубоких сетях с помощью остаточных связей (skip connections). Это позволило обучать сети в сотни и даже тысячи слоев
- **EfficientNet** (2019): Метод составного масштабирования, который создает более эффективные и точные модели

Процесс обучения: Функция потерь (Loss Function)

Это экзаменатор сети. Она измеряет, насколько предсказание сети отличается от правильного ответа

- Классификация: Кросс-энтропия (Cross-Entropy Loss)
- Обнаружение объектов: Комбинированные функции, например, Focal Loss (для проблем с дисбалансом классов) или Smooth L1 Loss для регрессии bounding box
- Сегментация: Dice Loss или Cross-Entropy для каждого пикселя

Процесс обучения: Оптимизатор (Optimizer)

Это «метод обучения».
Он определяет, как обновлять веса сети, чтобы минимизировать функцию потерь

- Stochastic Gradient Descent (SGD) и его модификации (например, SGD with Momentum)
- Adam — очень популярный адаптивный оптимизатор, который хорошо работает «из коробки»

Процесс обучения: Важные приемы и технологии

- **Transfer Learning (Трансферное обучение)**: Самый важный прием на практике! Вместо обучения с нуля берут предобученную на огромном наборе данных (например, ImageNet) модель и доучивают её под свою конкретную задачу

  Пример: У вас есть 1000 фото больных и здоровых растений.
  Вы берете ResNet50, обученную на ImageNet (она уже умеет видеть края, текстуры, формы), заменяете последний полносвязный слой на новый (под 2 класса) и дообучаете модель на своих данных.
  Это работает гораздо лучше и быстрее, чем обучение с нуля.

- **Fine-Tuning**: Более глубокая форма трансферного обучения, когда размораживают и переобучают не только последние слои, но и более глубокие

- **Регуляризация**: Методы борьбы с переобучением (когда модель запоминает данные, а не учится общим закономерностям). К ним относятся:
  - Dropout (случайное «выключение» нейронов во время обучения)
  - L2-регуляризация (штраф за слишком большие веса)
  - аугментация

Примерный пайплайн обучения для классификации

1. Сбор и разметка своего датасета или поиск готового (Kaggle и т.д.)
2. Разделение на обучающую, валидационную и тестовую выборки
3. Выбор архитектуры (например, ResNet34) и инициализация весов (лучше всего — предобученными на ImageNet)
4. Определение функции потерь (Cross-Entropy) и оптимизатора (Adam)

Примерный пайплайн обучения для классификации

5. Цикл обучения (Эпоха):
   - Пропускаем пакет (batch) обучающих данных через сеть → считаем предсказания
   - Сравниваем предсказания с истиной через функцию потерь → считаем градиенты (обратное распространение ошибки, Backpropagation)
   - Оптимизатор обновляет веса сети на основе градиентов
   - Повторяем для всех пакетов

6. Валидация: После каждой эпохи проверяем качество модели на валидационной выборке. Это помогает отслеживать переобучение

7. Сохранение лучшей модели по результатам валидации

8. Финальное тестирование на тестовой выборке — это итоговая, объективная оценка качества модели

Что обучается в нейронных сетях для CV?

Группа слоёв 1: Простейшие признаки (Low-level features)
Что учится: Сеть учится распознавать простейшие геометрические фигуры

Группа слоёв 2: Комбинации простых признаков (Mid-level features)
Что учится: Нейроны этих слоёв комбинируют признаки из предыдущих слоев

Глубокие слои: Сложные, семантические признаки (High-level features)
Что учится: Нейроны здесь реагируют на очень сложные и специфичные комбинации

Выходной слой: Классификация
Что учится: Этот слой получает на вход высокоуровневые признаки и учится их комбинировать, чтобы принять окончательное решение

Настройка двух ключевых типов параметров внутри сети

- **Веса** (Weights) **в свёрточных ядрах** (Filters/Kernels)

Что это: **Ядра свёрток**. Каждое такое ядро отвечает за поиск одного конкретного признака

Что обучается: **Числовые значения внутри этой матрицы**. В начале обучения они устанавливаются случайно. В процессе обучения сеть меняет эти числа так, чтобы одни ядра лучше всего активировались на краях, другие — на текстурах и т.д.

- **Веса в полносвязных слоях** (Fully Connected Layers)

Что это: В конце CNN обычно идут **полносвязные слои**, где каждый нейрон связан с каждым нейроном предыдущего слоя

Что обучается: **Сила связей между нейронами**. Сеть учится, какие комбинации высокоуровневых признаков (например, «ушко А» + «глазик Б» + «носик В») с каким весом ведут к классу «кошка»ы

Резюме: Что же обучается?

- **Набор детекторов признаков**: Сеть сама создает для себя «библиотеку» визуальных шаблонов (фильтров) — от линий и углов до частей объектов
- **Иерархическая структура**: Эти шаблоны организованы в иерархию, где сложные шаблоны собираются из простых
- **Правила принятия решений**: Сеть учится, какие комбинации этих шаблонов соответствуют какому классу объектов (кошка, собака, машина)

Параметры и гиперпараметры модели

Что это?

Откуда берутся?

Где и для чего используются?

Параметры модели (Parameters)

Что это?
Параметры — это внутренние переменные модели, которые она учит сама из данных в процессе обучения. Это «знания» модели

Откуда берутся?
Их настраивает алгоритм оптимизации (например, градиентный спуск) для минимизации функции потерь. Мы не задаем их вручную, модель находит их сама

Где находятся в нейронной сети?

Веса (Weights) связей между нейронами. Они определяют, насколько сильно выход одного нейрона влияет на вход другого

Смещения (Biases) у каждого нейрона. Это порог, который нужно преодолеть нейрону, чтобы активироваться

Параметры: Ключевые особенности

Изучаются из данных

Не задаются вручную

Сохраняются в файле модели после обучения

Используются для прогнозирования на новых данных

Гиперпараметры модели (Hyperparameters)

Что это?
Гиперпараметры — это внешние конфигурации модели, которые мы задаём вручную до начала процесса обучения. Они управляют процессом обучения

Откуда берутся?
Их выбирает Data Scientist/исследователь
Их находят методом проб и ошибок (например, с помощью поиска по сетке - Grid Search) или на основе опыта

Примеры гиперпараметров

- **Скорость обучения** (Learning Rate): Насколько сильно модель корректирует свои параметры на каждом шаге. Самый важный гиперпараметр

- **Количество эпох** (Number of Epochs): Сколько раз модель увидит всю обучающую выборку

- **Размер мини-выборки** (Batch Size): Количество примеров, обрабатываемых моделью за один шаг перед обновлением параметров

- **Архитектура модели**:
  - Количество слоёв (Number of Layers)
  - Количество нейронов в каждом слое (Number of Units)

- **Функция активации** (Activation Function): ReLU, Sigmoid, Tanh и т.д.

- **Параметры оптимизатора**: Например, момент (Momentum) для SGD

- **Коэффициент регуляризации** (Lambda): Помогает бороться с переобучением

Гиперпараметры: Ключевые особенности

- Задаются вручную до обучения
- Управляют процессом обучения и тем, как модель будет учить свои параметры
- Требуют настройки и экспериментов

Сравнительная таблица

Критерий	**Параметры**	**Гиперпараметры**
Источник	Изучаются из данных	Задаются вручную
Цель	Определяют «знания» модели, делают прогнозы	Контролируют **процесс** обучения
Определение	Автоматически, через оптимизацию	Эмпирически, через эксперименты
Примеры	Веса (Weights), Смещения (Biases)	Скорость обучения, количество эпох, размер батча
Влияние	Влияют на точность модели на данных	Влияют на то, насколько хорошо и быстро модель **научится**
Сохранение	Сохраняются как часть обученной модели	Сохраняются в конфигурационном файле/коде

План

1. **Введение:** Задачи Computer Vision и почему именно нейронные сети?
2. **Предшественники:** От перцептрона к идее свертки
3. **Эра CNN:** LeNet, AlexNet, VGG, GoogLeNet/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?

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**:

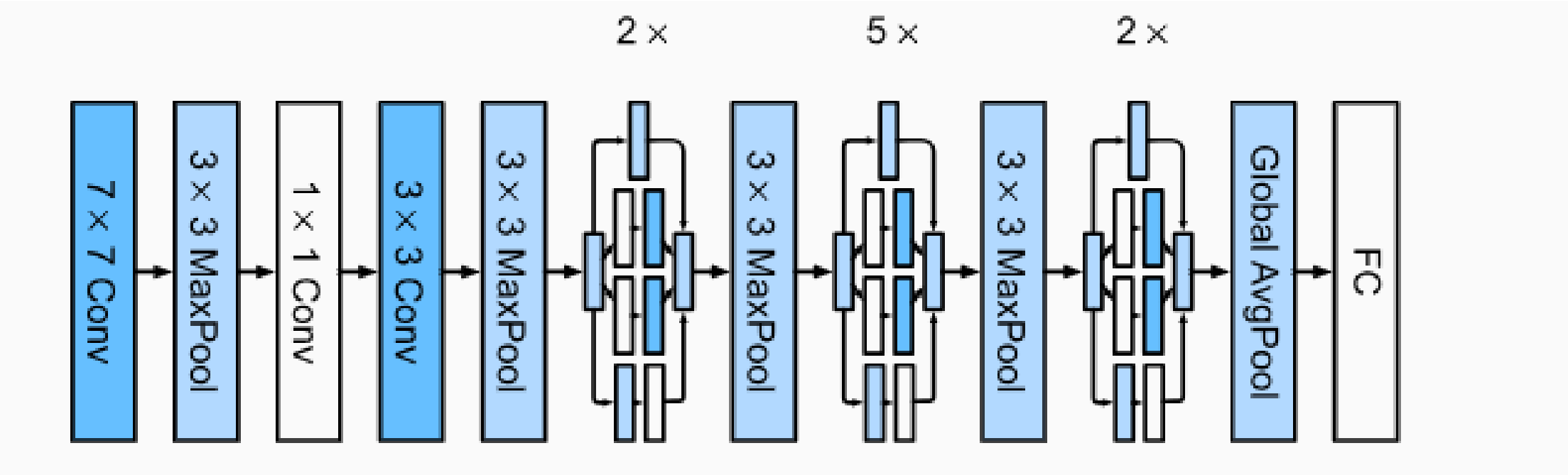
Число параметров: Огромное количество весов (например, в VGG-16 ~138 миллионов параметров), что требовало много памяти и вычислительной мощности

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

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**.

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 (ImageNet Large Scale Visual Recognition Challenge))

Базовая (наивная) идея Inception-модуля

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

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

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

Для чего это нужно?

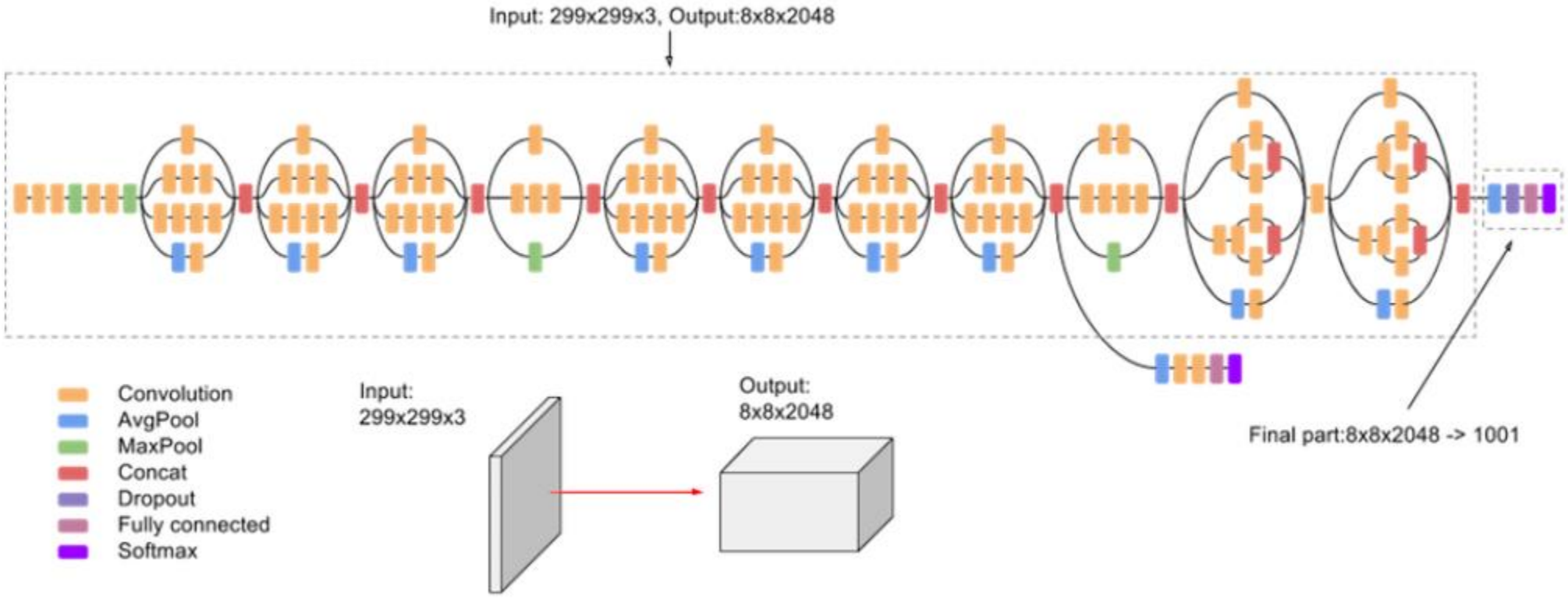
Уменьшение размерности (Reduction): Свёртка 1x1 позволяет уменьшить количество каналов (глубину тензора), прежде чем применять к нему дорогие свёртки 3x3 и 5x5

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.

GoogLeNet/Inception (2014)



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

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

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

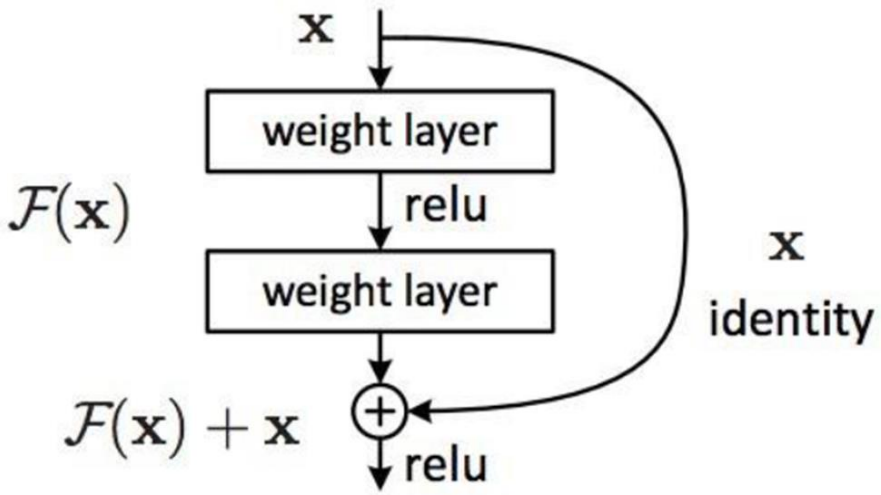
Решение: Остаточные блоки (Residual Blocks)

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

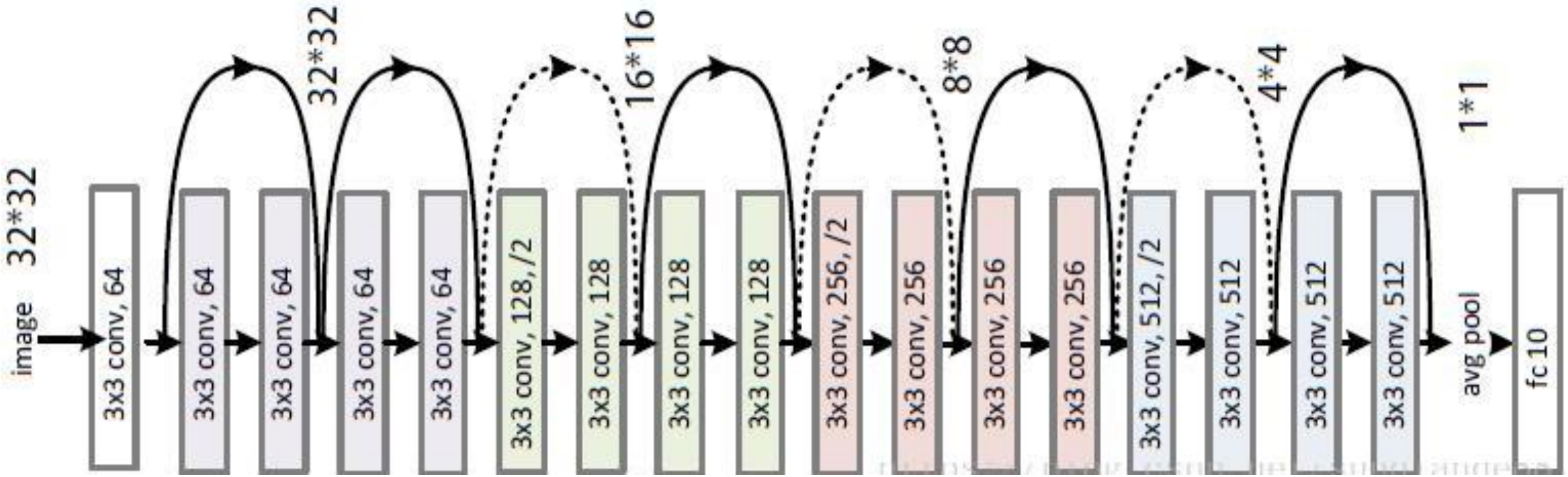
- При обратном проходе: Градиент от функции потерь теперь может напрямую течь назад по skip-connection'y. Цепное правило для блока выглядит так:

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

  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

Архитектура ResNet



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 даёт 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

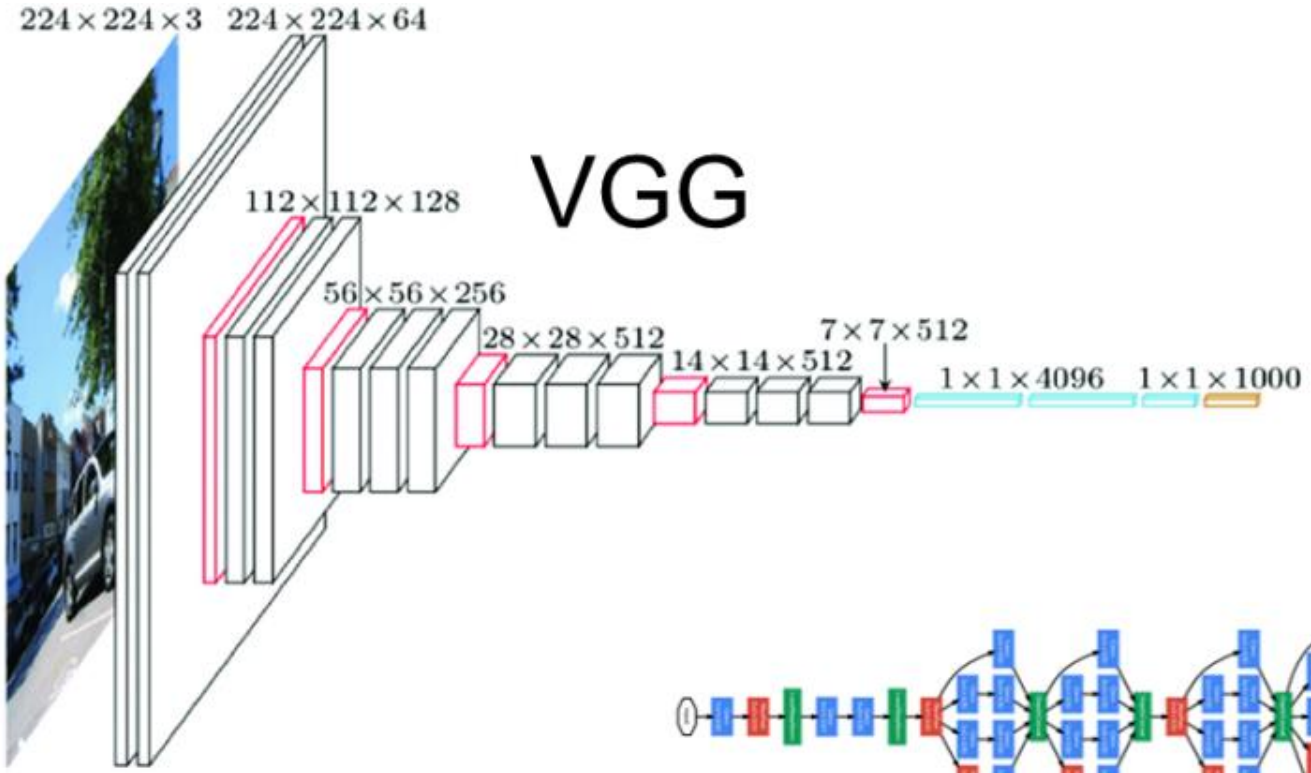
Значение и наследие ResNet

- **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)

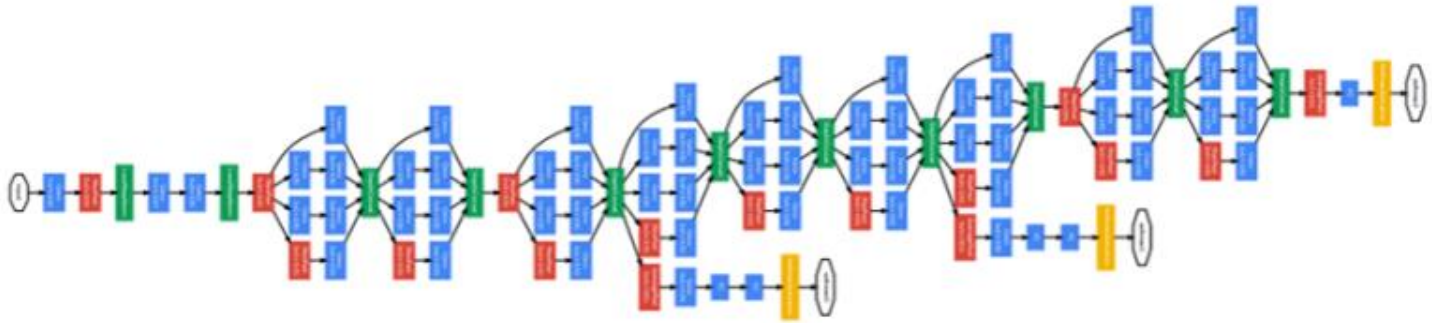
- **Интуитивное объяснение**:
  ResNet интуитивно понятен — сеть учится вносить небольшие поправки F(x) к уже имеющемуся представлению x, а не каждый раз заново изобретать его

- **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)

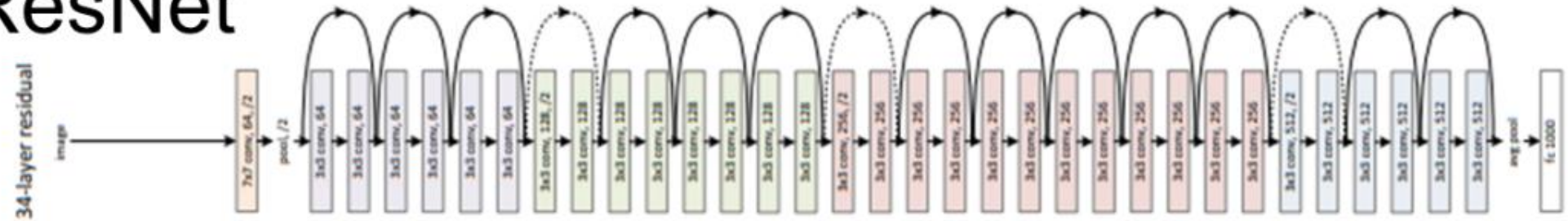


VGG

GoogLeNet



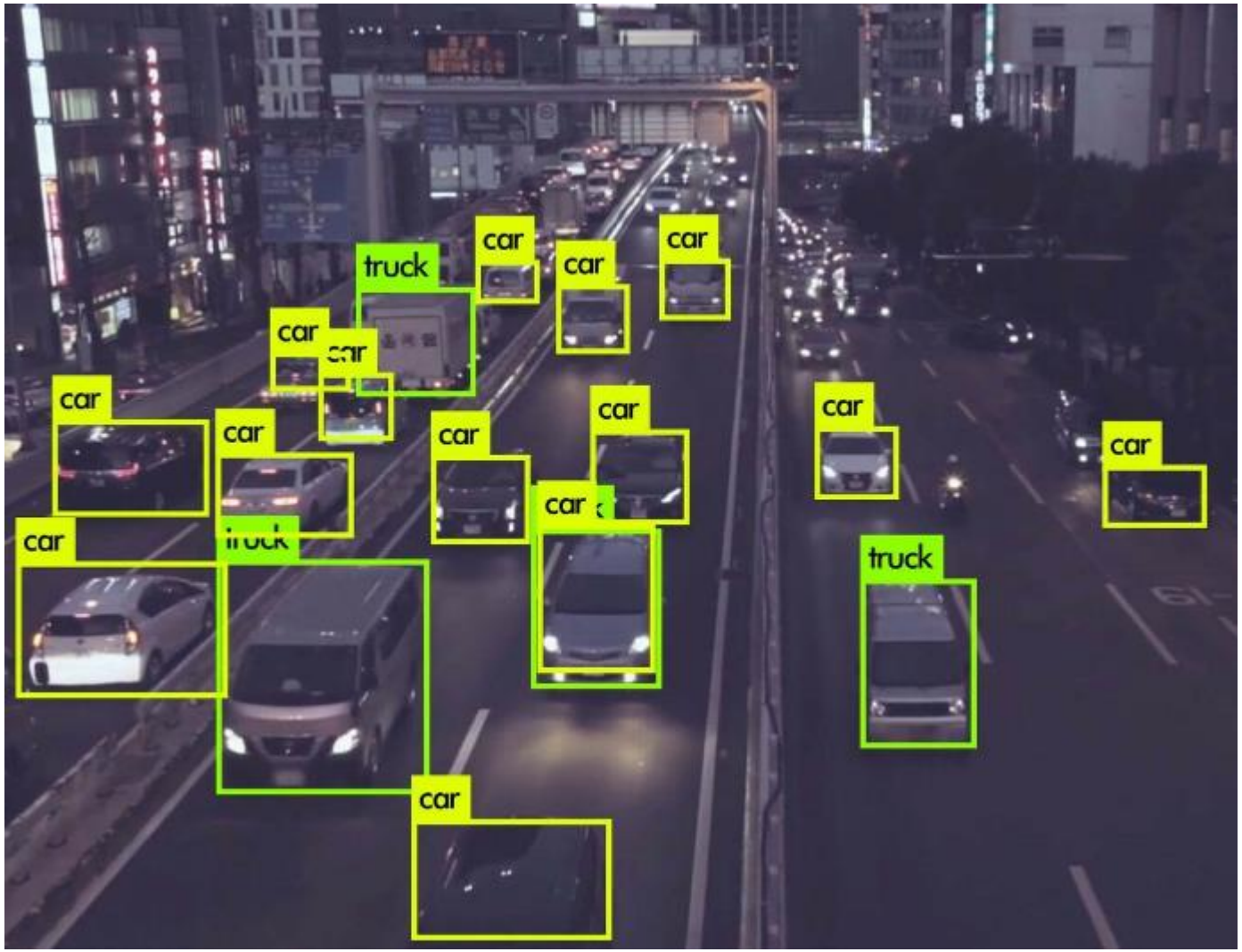
ResNet



План

1. **Введение:** Задачи Computer Vision и почему именно нейронные сети?
2. **Предшественники:** От перцептрона к идее свертки
3. **Эра CNN:** LeNet, AlexNet, VGG, GoogLeNet/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?

Детекция объектов



Основные аспекты задачи детекции объектов

Идентификация объектов:

- объекты в изображении или видео могут быть классифицированы на основе заданных категорий, таких как люди, автомобили, животные и т. д.
- Идентификация предполагает, что модель может определить, какие классы объектов присутствуют в сцене

Локализация объектов:

- определение местоположения каждого обнаруженного объекта в виде ограничительной рамки.
- Рамка указывает координаты верхнего левого и нижнего правого угла объекта на изображении.

Детекция объектов: От Двухстадийных к Одностадийным и Современное Состояние

Двухстадийные детекторы объектов (такие как R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN) сначала выделяют области-кандидаты, а затем уточняют их классификацию и границы, обеспечивая высокую точность, но меньшую скорость.

Одностадийные детекторы (такие как SSD, RetinaNet, EfficientDet, YOLO) выполняют задачу за один проход, одновременно предсказывая классы и границы объектов, что делает их быстрыми, но менее точными для мелких или плотно расположенных объектов.

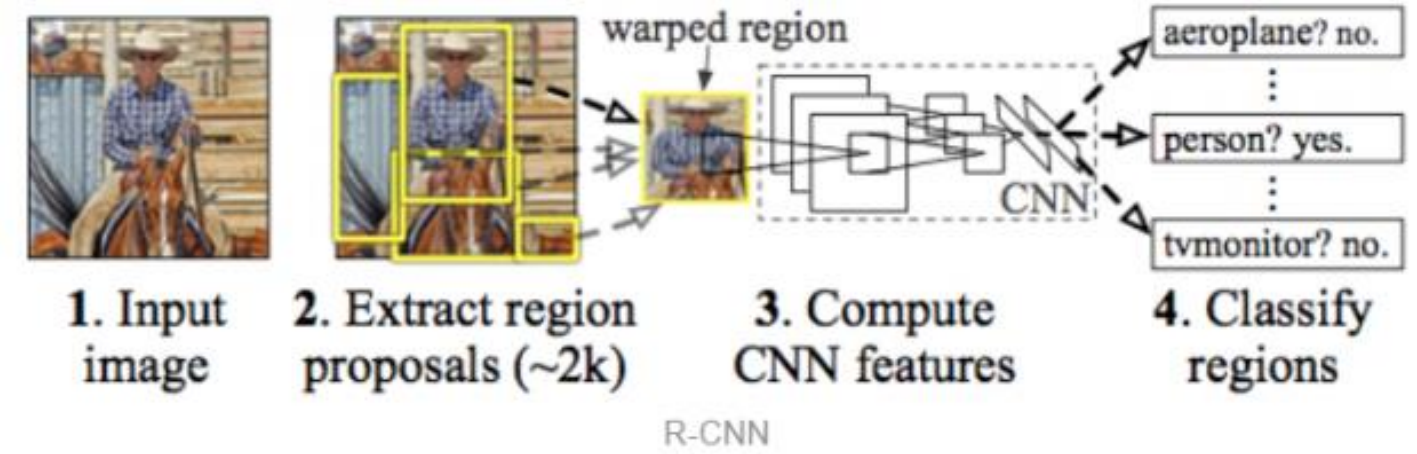
Выбор подхода зависит от приоритета: точность или скорость.

Двухстадийные детекторы (Region-Based Detectors)

Ключевой принцип:
Что такое объект? ➡ Что это за объект?

Сначала ищутся регионы интереса (Region Proposals),
а затем эти регионы классифицируются

R-CNN (2014)



R-CNN (Regions with CNN features) — это архитектура нейронной сети, предложенная в 2014 году, которая произвела революцию в задачах детекции объектов.

R-CNN **объединяет** методы извлечения **признаков** на основе свёрточных нейронных сетей (**CNN**) с **традиционными** подходами к **обнаружению** объектов, создавая более точные и эффективные системы.

Основные компоненты R-CNN

Выбор кандидатов на объекты (Region Proposals):

- для начала R-CNN использует алгоритм Selective Search, который генерирует набор предложений (region proposals) — ограничительных рамок, которые могут содержать объекты
- Этот шаг позволяет значительно сократить количество областей, которые нужно анализировать, вместо того чтобы проверять каждую позицию и масштаб изображения

Извлечение признаков с помощью CNN:

- каждая область-кандидат (region proposal) масштабируется до фиксированного размера и передаётся через свёрточную нейронную сеть (CNN), чтобы извлечь высокоуровневые признаки.
- Обычно для этой цели используют предобученные модели, такие как AlexNet, VGG16 и другие

Основные компоненты R-CNN

Классификация объектов:

1. извлечённые признаки передаются в полносвязные слои, которые обучаются на определённых классах объектов
2. Модель предсказывает вероятность принадлежности каждой области-кандидата к различным классам

Регрессия ограничительных рамок:

- для каждой ограничительной рамки также выполняется задача регрессии, чтобы уточнить местоположение объекта
- Это позволяет улучшить точность ограничительных рамок, предсказывая координаты с учетом ошибок первоначального выбора

R-CNN (2014): Итоги

Как работает:
- Использует внешние алгоритмы (вроде Selective Search) для генерации ~2000 регионов.
- Каждый регион масштабируется, пропускается через CNN (например, AlexNet) для извлечения признаков,
- и затем классифицируется SVM-ом.

Недостатки:
- Чрезвычайно медленно,
- огромное потребление памяти,
- многоэтапный пайплайн.

Fast R-CNN (2015)

Улучшенная версия оригинальной R-CNN (Region-based Convolutional Neural Network), предложенная Россом Гиршем в 2015 году для задачи детекции объектов.

Fast R-CNN была разработана для увеличения скорости и эффективности модели, сохраняя при этом высокую точность обнаружения объектов.

Основная задача Fast R-CNN — локализация объектов на изображениях и классификация их в один из предопределённых классов.

Основные этапы работы Fast R-CNN:

Использование свёрточной сети:

- в отличие от оригинальной R-CNN, Fast R-CNN обрабатывает всё изображение с помощью свёрточной нейронной сети (например, VGG16 или ResNet), которая извлекает признаки из изображения.
- Вместо обработки каждого возможного региона отдельно, изображение целиком проходит через сеть только один раз (что сильно ускоряет процесс обработки).

Предсказание регионов (ROI, Region of Interest):

- после обработки изображения сеть генерирует региональные предложения (Region Proposals) с помощью алгоритма Selective Search.
- Вместо вырезания этих регионов и повторного использования для классификации, как в R-CNN, Fast R-CNN использует ROI pooling для извлечения этих предложенных регионов из карты признаков.

Основные этапы работы Fast R-CNN:

ROI Pooling:

- уменьшает размер предложенных регионов до фиксированного, независимо от их оригинальных размеров.
- Это упрощает обработку, позволяя унифицировать входы для последующих слоёв, сохраняя ключевые признаки регионов — пропорции и размер.

Классификация и регрессия:

- затем, с использованием извлечённых признаков каждого региона, модель предсказывает класс объекта, находящегося в предложенном регионе,
- и уточняет его координаты с помощью регрессии, чтобы получить более точные рамки для объекта.

Ключевые улучшения Fast R-CNN

Обработка изображения одним проходом через сверточную сеть:
в оригинальной R-CNN для каждого регионального предложения проводился отдельный прогон через свёрточную сеть, что делало процесс очень медленным.
Fast R-CNN устраняет эту проблему, проводя свёртку всего изображения только один раз, что значительно ускоряет процесс.

ROI Pooling:
вырезание регионов интереса (ROI) на уровне карты признаков делает процесс более эффективным, поскольку позволяет работать с областями разного размера и формы без необходимости прямого вырезания на исходном изображении.

Ключевые улучшения Fast R-CNN

Интеграция обучения:
в Fast R-CNN происходит end-to-end обучение, то есть все параметры сети, включая классификатор и регрессионные веса для точных координат объектов, обучаются одновременно в рамках одной модели.
Это упрощает процесс обучения по сравнению с R-CNN.

Быстрота:
благодаря тому, что Fast R-CNN обрабатывает изображение только один раз и использует более оптимизированные подходы для предсказания регионов, она значительно быстрее оригинальной R-CNN.

Fast R-CNN (2015): Итоги

Улучшение:

- Вместо прогона каждого региона через CNN, всё изображение один раз пропускается через CNN
- Регионы проектируются на итоговую карту признаков, и для каждого извлекается фиксированный вектор (RoI Pooling)
- Далее идёт классификация и уточнение bounding box

Результат:

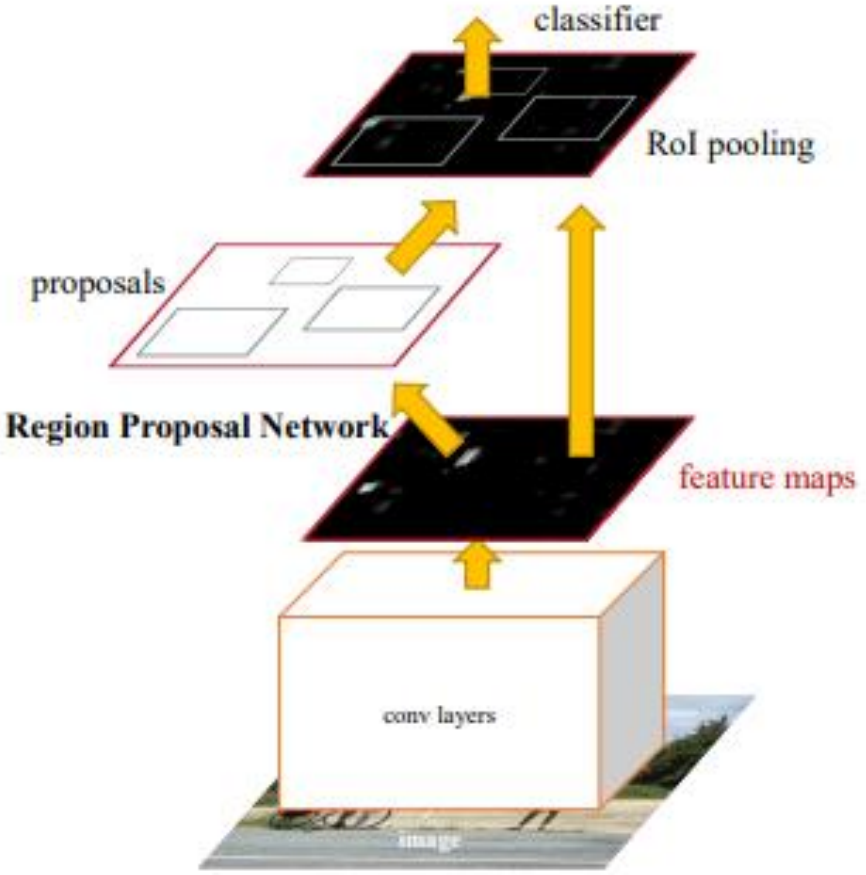
- Значительно быстрее и точнее R-CNN

Faster R-CNN (2015) — Прорыв:

Faster R-CNN — это улучшенная версия R-CNN (Regions with CNN features), представленная в 2015 году.

Эта архитектура решает проблемы оригинальной R-CNN, связанные с медленной скоростью работы и высоким потреблением ресурсов, благодаря внедрению Region Proposal Network (RPN).

Faster R-CNN обеспечивает высокую точность в детекции объектов, сохраняя при этом скорость обработки.



Основные компоненты Faster R-CNN

- **Region Proposal Network (RPN)** — это отдельная сеть, которая генерирует кандидаты на объекты (region proposals) непосредственно из карты признаков, созданной свёрточной нейронной сетью.
  Она значительно ускоряет процесс, так как RPN и классификатор могут быть обучены совместно. Это позволяет избежать использования внешних методов, таких как Selective Search.

- **Сверточная нейронная сеть (CNN)**: как и в оригинальной R-CNN, Faster R-CNN использует предобученные CNN (например, VGG16 или ResNet) для извлечения высокоуровневых признаков из входного изображения.

- **Классификация и регрессия**: после того как RPN генерирует кандидаты на объекты, они передаются в классификатор, который предсказывает классы объектов и уточняет координаты ограничительных рамок через регрессию.

Faster R-CNN (2015): Итог

Ключевое нововведение: Region Proposal Network (RPN).

- Нейросеть сама учится генерировать регионы интереса, а не использует внешние алгоритмы.
- RPN и детектор (классификатор) разделяют общие свёрточные слои, что делает пайплайн полностью end-to-end.

Итог:

- Высокая точность, ставшая золотым стандартом на многие годы.

Плюсы и минусы двухстадийных детекторов

Плюсы:

- Исторически высочайшая точность, особенно для маленьких объектов.
- Более точное локализация bounding box.

Минусы:

- Относительно низкая скорость (2-5 FPS на R-CNN, до ~10 FPS на Faster R-CNN).
- Сложный пайплайн.