ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ МГУ ИМЕНИ М.В. ЛОМОНОСОВА

ФОНД ВОЛЬНОЕ ДЕЛО

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ

КОМПЬЮТЕРНОЕ ЗРЕНИЕ

КОНУШИН
АНТОН СЕРГЕЕВИЧ

ВМК МГУ

КОНСПЕКТ ПОДГОТОВЛЕН СТУДЕНТАМИ, НЕ ПРОХОДИЛ ПРОФ. РЕДАКТУРУ И МОЖЕТ СОДЕРЖАТЬ ОШИБКИ. СЛЕДИТЕ ЗА ОБНОВЛЕНИЯМИ НА VK.COM/TEACHINMSU.

ЕСЛИ ВЫ ОБНАРУЖИЛИ ОШИБКИ ИЛИ ОПЕЧАТКИ, ТО СООБЩИТЕ ОБ ЭТОМ, НАПИСАВ СООБЩЕСТВУ VK.COM/TEACHINMSU.

БЛАГОДАРИМ ЗА ПОДГОТОВКУ КОНСПЕКТА
СТУДЕНТКУ ФАКУЛЬТЕТА ВМК МГУ
ЧЕРНИКОВУ ПОЛИНУ ГЕОРГИЕВНУ

Оглавление

Лекция 1. Введение в компьютерное зрение. Свет и цвет. Модели обработки цвета.

Понятие компьютерного зрения

Задача обычного зрения – понять, что находится на изображении. Компьютерное зрение – построение компьютерной модели системы зрения, т.е. создание программы, которая может ответить на любой вопрос об изображении, на который может ответить человек.

Определение 1. **Компьютерное зрение** – часть области искусственного интеллекта (AI)

Как и для других задач, для компьютерного зрения есть своя версия теста Тьюринга, т.е. компьютер должен ответить на любой вопрос про изображение, на который может ответить человек.

Человек и компьютер могут отвечать на следующие вопросы про изображение:

1) Что и где находится на изображении? – *задача детекции*
2) Вопрос о свойстве объектов и их атрибутов – *задача классификации* в зависимости от ответа
3) Какой формы и какого размера объект? – *задача метрического зрения*, сложной версией которой является построение 3-D модели объекта

Поговорим о зрении человека. Поскольку зрение относится к области искусственного интеллекта, значит, наиболее известный эталон – зрение человека. По оценкам, 25% мозга занято решением задачи зрения. С помощью глаз мы извлекаем визуальные подсказки, какие-то характеристики реальных объектов, сопоставляем их с наши априорными знаниями об устройстве окружающего мира и делаем выводы.
Мы можем распознать объекты на изображении, используя следующие *подсказки*:

1) Цвет
2) Контур
3) Текстура
4) Тени и освещение
5) Контекст (окружение объекта)

К априорным знаниям об объекте можно отнести следующее:

1) Размеры
2) Контекст
3) Типичный диапазон распределения конкретной характеристики
4) Эталонные примеры

Нейрофизиологи много времени посвятили изучению мозга человека, одной из важнейших и этапных работ в этой области являются работы британского

нейрофизиолога Давида Марра в 1970-х, который сформулировал принципиальную структуру обработки визуальной информации в мозге человека. Он разделил процесс обработки на 3 этапа: *основной набросок, набросок 2.5D, 3D модель*.

Этап	Процесс
Основной набросок	Извлечение низкоуровневых (low-level) свойств изображения: направленных краев, отрезков и т.д.
Набросок 2.5D	Добавление бинокулярной информации (упорядочивание по глубине), учет текстуры и т.д.
3D модель	Распознавание объектов с учетом априорной информации, представление в 3D модели

Именем Давида Марра (“The Marr Prize”) была названа главная премия в области компьютерного зрения.
Компьютерное зрение на прямую зависит от развития искусственного интеллекта и машинного обучения. Первым алгоритмом, дошедшим до пользователя, стал алгоритм Viola-Jones (2001 г.) – быстрый (работа в режиме реального времени) и надежный детектор лиц, демонстрирующий силу машинного обучения. После появления этого алгоритма число работ в области компьютерного зрения стремительно начало расти.

Цифровое изображение

Определение. **Изображение оптическое** – картина, получаемая в результате прохождения через оптическую систему лучей, распространяющихся от объекта, и воспроизводящая его контуры и детали.

Одним из старейших устройств для получения изображения является **камера-обскура** (рис. 1).
Ее математической модель – перспективная проекция: пучок лучей проходит через одну точку (точечное отверстие) – «центр проекции» (focal point) и на картинной плоскости (image plane) позади фокуса формируется изображение.

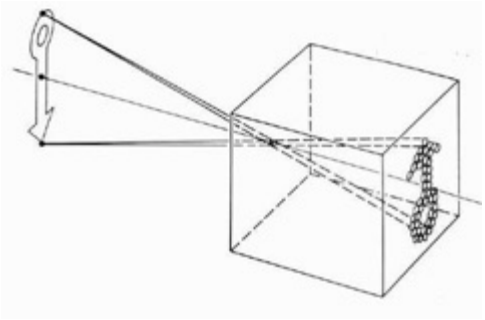


рис.1

Перспективная проекция отображает 3D объекты в 2D изображение. При этом происходят некоторые искажения реальных характеристик объектов. Например, изменение пропорций, угла наклона. Так на рис.2 боковые колонны кажутся шире, потому что колонны, которые мы рассматриваем это 3D объект, а не рисунок, и с одно точки мы видим разные части этих реальных объектов, поэтому проекции этих одинаковых объектов на плоскость будут разными.

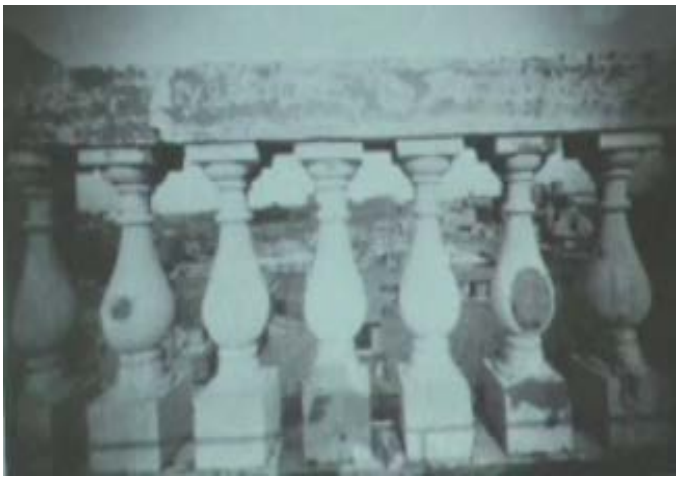


рис.2

Проецируя 3D мир в 2D изображение, мы теряем значительную часть информации: углы, расстояния и длины. Остается только свойство линейности: если какая-то линия является прямой, то после перспективной проекции она остается прямой.

Современные цифровые фотоаппараты устроены по принципу камеры-обскура, но вместо точечного отверстия – объектив, и есть цифровая матрица.

Определение. **Дискретизация** – перевод изображения в цифровой вид

Определение. **Цифровое полутоновое изображение** – матрица $I \in (b_{ij})^{n,m}$, элементами которой b_{ij} являются значения интенсивности света, измеренного на двухмерной прямоугольной сетке.

- b (интенсивность) можно описать вещественным числом. обычно ограничиваются интервалом $b_{ij} \in [0; 1]$б где 0 – нет света, 1- максимальная яркость
- Обычно b кодируется 1 байтом, т.е. b $b \in [0; 255]$
- Можно использовать большую точность (10-16 бит)

Цветные изображения, свет и цвет, модели цвета

Определение 1. **Цвет** – это психологическое свойство нашего зрения, возникающее при наблюдении объектов и света, а не физическое свойство объектов и света (S. Palmer, Vision Science: Photons to Phenomenology)

Определение 2. **Цвет** – это результат взаимодействия света, сцены и нашей зрительной системы.

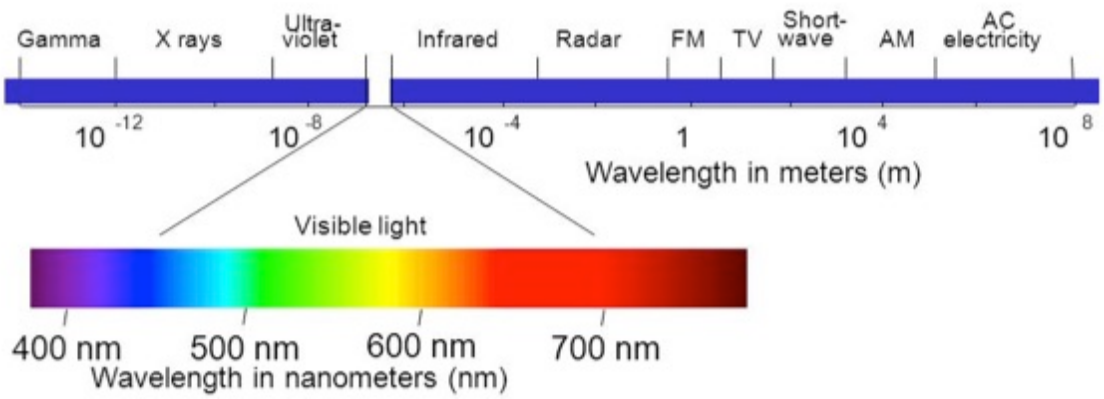


рис.3

Видимый свет – это электромагнитное излучение в диапазоне $[380nm, 780nm]$ (рис. 3) Максимум чувствительности глаза приходится на длину волны примерно 560 nm, что соответствует цвету, воспринимаемому человеком как зеленый. в ходе эволюции наш глаз приспособился воспринимать свет именно в таком диапазоне, поскольку именно на этот диапазон приходится значительная энергия ($\sim 46\%$).

Любой источник света можно полностью описать спектром6 количество излученной энергии в единицу времени для каждой длины волны в интервале $[380nm, 780nm]$

Разные источники света отличаются своим спектром. Есть простые источники света, которые излучают свет только с одной длиной волны – *когерентные* источники света или *лазеры.* Есть объекты, которые излучают свет в узком диапазоне – разные *фосфоресцирующие кристаллы.* Видимый спектр отличается распределением энергии. Так, *например*, лампочка накаливания большую часть энергии передают в красном и желтом диапазоне, а дневной свет распределен более равномерно. хотя большая часть энергии переносится более коротки волнами.

Далее, попадая на объект свет отражается от него, в зависимости от свойств поверхности разные длины волн отражаются по-разному. Для того, чтобы измерить и посмотреть, какой будет отраженный свет, нужно измерить отражающие свойства поверхности: измерить долю света, который отражается для каждой длины волны. Таким образом, мы домножаем график света на кривую отражательной способности (рис.4) и получаем график отраженного света.

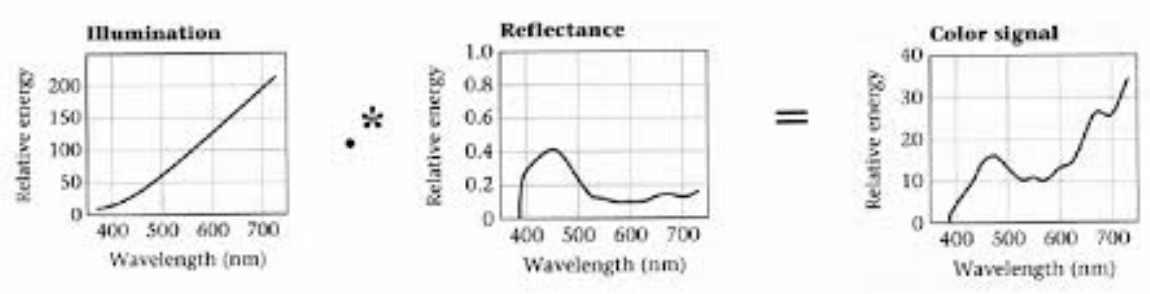


рис.4

Далее этот спектр попадает в глаз человека. Глаз человека похож на цифровой фотоаппарат и представляет собой оптическую систему, состоящую их хрусталика и светочувствительной матрицы в виде сетчатки, которая состоит из палочек и колбочек. *Палочки* (Rods) измеряют общую яркость цвета, а *колбочки* (Cones) измеряют цвет. Всего 3 типа колбочек, которые чувствительны к разным диапазонам света (рис. 5).

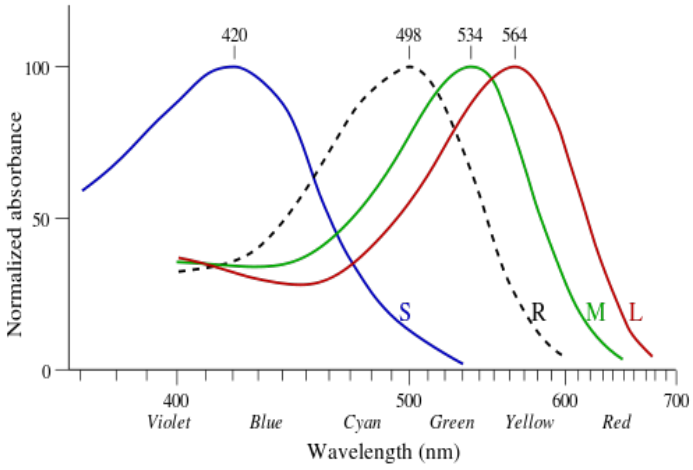


рис.5

Палочки и колбочки – фильтры света. Три числа, которые идут из синих, зеленых и красных колбочек (S, M, L) дают описание воспринимаемого света, который дальше анализируется глазом. Однако мы не можем взаимно-однозначно описать весь спектр 3 числами, так как два разных спектра - *метамеры* могут быть неотличимы и восприниматься глазом человека идентично.

Модели цвета

Модель LMS – биологически обусловленная модель

В качестве представление цвета записываются те 3 числа, которые условно формируются в сетчатке: L – long, M- medium, S – short, то есть домножаем спектр света на краевые значения колбочек и получаем 3 числа. Такая модель на практике неудобна. Во-первых, на практике невозможно достичь все комбинаций этих трех чисел. Например, не существует такого спектра, для которого $M > 0,\ S = L = 0$. Во-вторых, невозможно построить дисплей, который состоял бы из 3 излучателей и работал бы в LMS.

Трихроматическая теория – все множество цветов можно получить комбинацией трех базовых цветов.

- Выбираем базовые цвета P_1, P_2, P_3
- Остальные цвета задаются линейной комбинацией базовых цветов (primales), веса- координаты цвета.
- Смешение трех цветов: $A = u_1P_1 + u_2P_2 + u_3P_3$

На основании множества экспериментов были сформулированы несколько эмпирических законов трихроматической теории – *закон аддитивности Грассмана.*

Закон аддитивности Грассмана 1853 г.:

Сопоставление цветов линейное

Если два источника света сопоставляются одинаковыми весами базовых источников, то они воспринимаются одинаково

- Пусть $A = u_1P_1 + u_2P_2 + u_3P_3$ и B $= u_1P_1 + u_2P_2 + u_3P_3$
- Тогда A = B

Если мы смешиваем два источника, тогда смешение соответствующих им базовым источников будет восприниматься так же

- Пусть $A = u_1P_1 + u_2P_2 + u_3P_3$ и B $= v_1P_1 + v_2P_2 + v_3P_3$
- Тогда A + B = $(u_1 + v_1)P_1 + (u_2 + v_2)P_2 + (u_3 + v_3)P_3$

Если мы увеличиваем яркость источника света, тогда яркость соответствующих базовых цветов должна быть увеличена на такое же значение

- Пусть $A = u_1P_1 + u_2P_2 + u_3P_3$
- Тогда k $A = ku_1P_1 + ku_2P_2 + ku_3P_3$

Модель CIE RGB 1931

В качестве базовых источников цвета три монохроматических лазера:

p_1 = 645.2 nm

p_2 = 523.3 nm

p_3 = 444.4 nm

Все множество цветов, представимых такой моделью мы можем визуализировать в виде цветового куба (рис.6).

Описывает ли модель RGB все множество видимых человеком цветов? Для проверки проведем эксперимент и попробуем сопоставить каждой длине волны видимого цвета (когерентному источнику света) цвет модели RGB.

Функции сопоставления – веса, необходимые для сопоставления с когерентными источниками света. Будем усреднять результаты участника эксперимента. В ходе эксперимента выясняется, что существуют такие длины волны, которые с точки зрения человека невозможно сопоставить с базовыми источниками цвета. Соответствие можно добиться только, если добавить один из базовых источников цвета к тестовому сигналу. Если у функции сопоставления отрицательный вес, это означает, что для сопоставления с данным цветом нам нужно подсветить источник одним из базовых цветов (рис. 7).

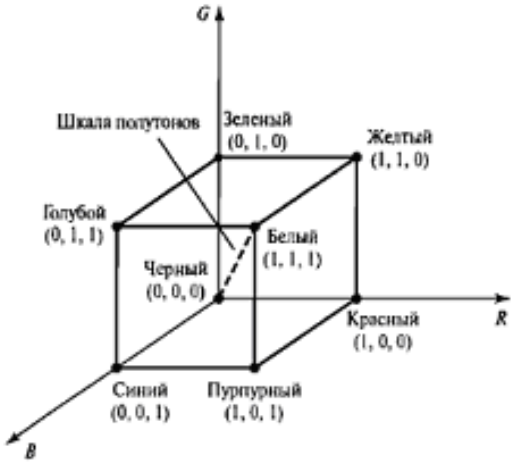


рис. 6

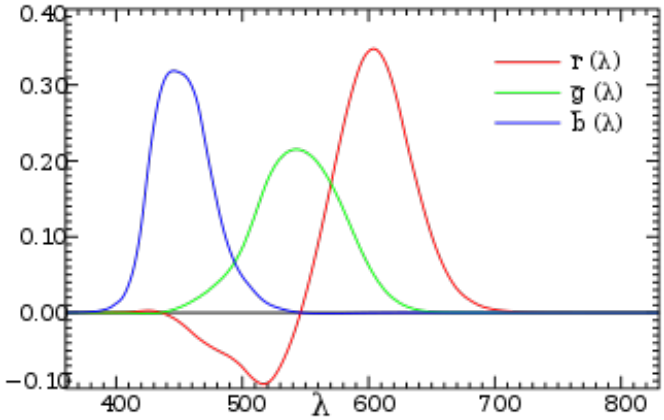


рис. 7

Таким образом, модель RGB не полностью описывает все множество видимых цветов. Кроме того, представление в виде RGB – это несколько неестественное описание для человека, так как мы воспринимаем цвет как некую единую характеристику, а не как комбинацию зеленого, красного, синего. Интуитивно можно выделить характеристики света: яркость (brightness) и цветность (chromaticity). В цветности можно выделить тон (hue) и насыщенность (saturation).

МодельYIQ

Цветовая модель YIQ используется в коммерческом цветном телевидении США. она своместима с черно-белым телевидением и используется в стандарте JPEG. I = R = C; Q = M –G. Пример модели с отдельной яркостью:

Y = .299R+ .587G+ .114B	R = 1.000 Y – 0.956 I – 0.621 Q
I = .569R- .275G- .321B	G = 1.000 Y – 0.272 I + 0.647 Q
Q = .212R- .528G+ .311B	B = 1.000 Y – 1.106 I + 1.703 Q

Модель CIE 1931 (XYZ)

Линейная аддитивная модель XYZ, которая покрывает все множество цветов, Y соответствует видимой яркости цвета, функции сопоставления неотрицательны (рис. 8).

- X, Z описывают «хроматическую» (цветовую) компоненту
- точки (1,0,0), (0,1,0), (0,0,1) – мнимые базовые цвета
- X, Y, Z изменяются от 0 до ∞

$$X = \int_{380}^{780} L_{e,\Omega,\lambda}(\lambda)\bar{x}(\lambda)d\lambda$$

$$Y = \int_{380}^{780} L_{e,\Omega,\lambda}(\lambda)\bar{y}(\lambda)d\lambda$$

$$Z = \int_{380}^{780} L_{e,\Omega,\lambda}(\lambda)\bar{z}(\lambda)d\lambda$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Перевод из XYZ в RGB

XYZ, RGB - линейные трихроматические модели, из одной модели в другую можно перейти с помощью линейного преобразования

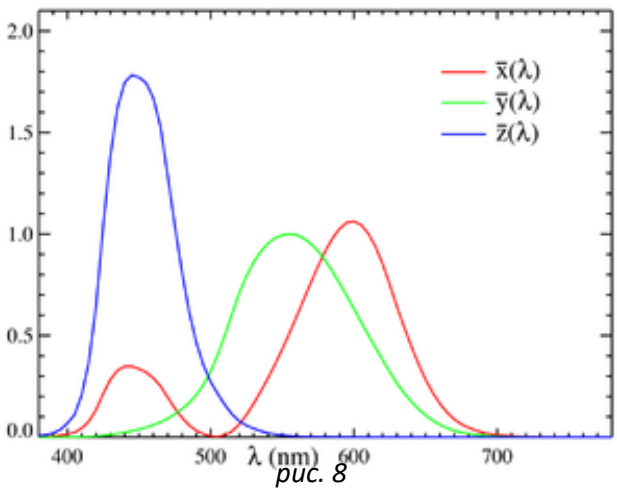


рис. 8

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & 0.1805 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

CIE XYZ и диаграмма цветов (Gamut)

Опишем цветность двумя нормированными параметрами x и y:

x = X/(X+ Y + Z), y = Y/(X + Y + Z)

Можем построить диаграмму цветов для x, y ∈ [0,1] (рис.9)

Наблюдения:

- Когерентные источники света располагаются по дуге
- Нижняя прямая «фиолетового» соответствует цветам, которые невозможно получить
- Никакими тремя реальными базовыми цветами невозможно покрыть видимый диапазон цветов
- Какие бы три точки мы не брали, всегда будет область, которая попадает вне этой трихроматической модели, следовательно, чтобы представить все возможные цвета, нужны мнимые базовые цвета.

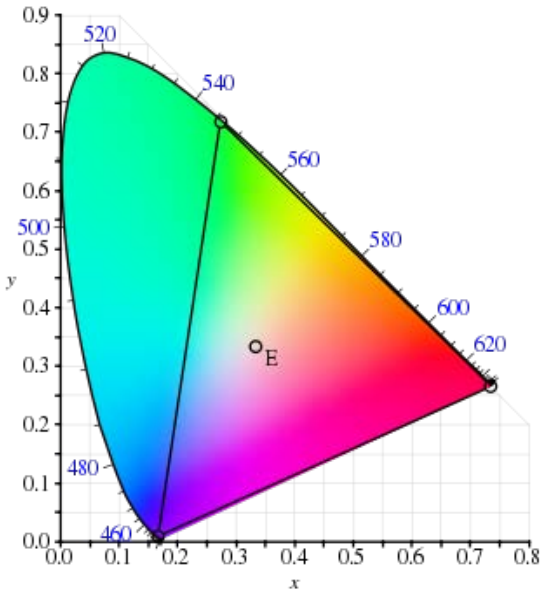


рис. 9

Нелинейность яркости

Глаз человека воспринимает яркость нелинейно, он лучше различает изменение яркости в темных областях, чем в светлых. Поэтому хранить абсолютную значению

яркости неэффективно. Вместо этого используется *гамма-преобразование* $y = c \cdot x^{\gamma}$. Линейные яркости отображаются на нелинейную шкалу и дискретизируются Маленьким изменениям в параметрах яркости соответствует большое изменение физической яркости.

Физически равномерные яркости

Физически неравномерные яркости

Модель sRGB

Стандартная RGB модель для HDTV, мониторов, цифровых камер и т.д. Отличается нелинейным представлением яркости. На практике при работе с этой моделью получаем из XYZ с помощью линейных преобразований линейное представление RGB. Затем применяем к этим линейным значениям нелинейную проекцию.

Из XYZ в sRGB

$$\begin{bmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{bmatrix} = \begin{bmatrix} 3.2406 & -1.5372 & 0.1805 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$C_{srgb} = \begin{cases} 12.92C_{linear}, & C_{\ linear} \leq 0.0031308 \\ (1+a)C_{linear}{}^{\frac{1}{24}} - a, & C_{linear} > 0.0031308, a = 0.055 \end{cases}$$

Из sRGB и XYZ $C_{linear} = \begin{cases} \frac{C_{srgb}}{12.92}, & C_{srgb} \leq 0.04045 \\ \left(\frac{C_{srgb}+a}{1+a}\right)^{24}, & C_{srgb} > 0.04045 \end{cases}$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Модель HSV (HIS)

Попытка задать цвет в интуитивно понятных для человека параметрах. Визуализируется эта модель в виде конуса (рис. 10), где углом направления задается тон (hue), конкретный цвет, высотой задается яркость (Value, Intensity), удалением от центральной оси задается насыщенность цвета (Saturation). Эта модель не является линейной, так как представляет собой конус.

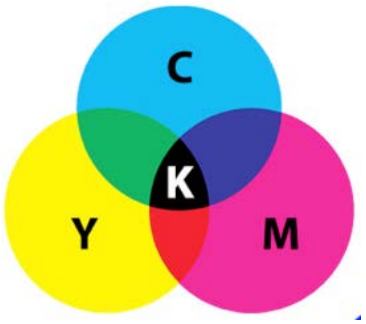


рис. 9

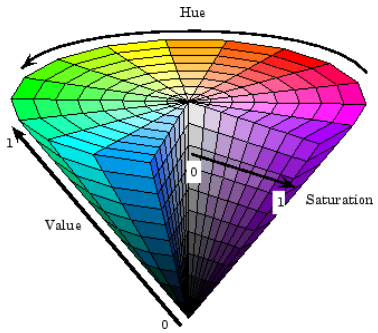


рис. 10

Субтрактивная модель CMYK

Используется для печати. Базовыми цветами являются желтый, фиолетовый, голубой и черный. Аддитивная система – RGB, субтрактивная система – CMY.

C = G + B = W - R

M = R + B = W - G

Y =R + G = W - B

Цветное цветовое изображение

Накроем каждый пиксель светофильтром, поскольку нам важна яркость изображения, а на яркость в большей степени влияет зеленая компонента, мы делаем зеленых фильтров больше, чем красных. То есть все пиксели разбиваются на группы по 4: 2 пикселя закрываются зеленым светофильтром, 2 пикселя закрываются синим и красным, что представляет собой *Байеровский шаблон* (рис. 11). Получаем картинку, где мы знаем одну компонента цвета (либо зеленую, либо красную, либо синюю). Для того, чтобы получить цветное изображение, нужно *проинтерполировать* значения компонент в пропущенных областях. Линейная интерполяция плоха тем, что в области границы реального объекта могут смешаться цвета разных объектов, что приводит к ошибке. Для того, чтобы интерполяция была корректной, сначала нужно определить границы

объекта, а затем выбирать цвета, принадлежащие только одному и тому же объекту.

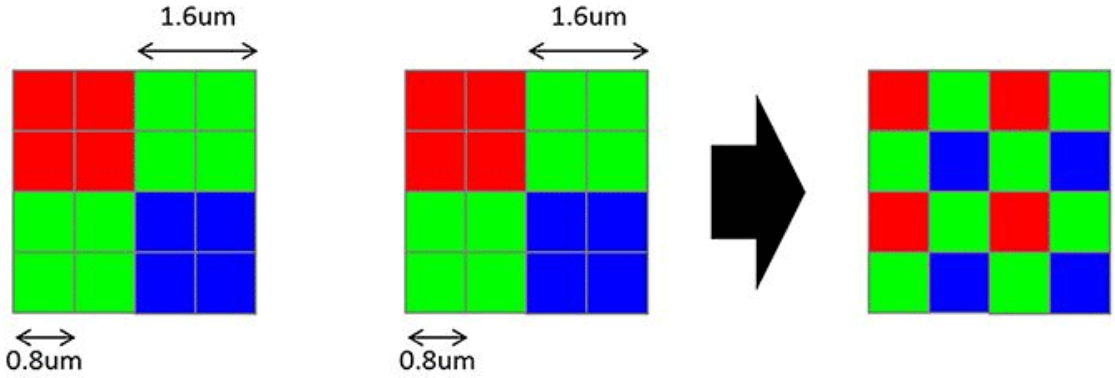


рис. 11

Цветовой баланс

Когда мы смотрим на фотографию или монитор, глаза адаптируются к освещению в комнате, а не к освещению сцены на фотографии. Если баланс белого неточен, цвета фотографии кажутся неестественными.

Коррекция яркости и цветопередачи в изображении

В чем причины неправильной передачи яркости и цвета? Рассмотрим тракт передачи изображения: реальный объект – матрица оптической системы – дискретные значения яркости и цвета – визуализация на мониторе. Основные причины некорректной передачи кроются в функции отображения яркости падающего цвета. Если функция неправильна с точки зрения человеческого восприятия, то визуализация картинки отличается от восприятия человека реального объекта в реальном мире.

Чтобы оценить качество передачи яркости в изображении строят *гистограмму.*

Определение. **Гистограмма –** это график распределения яркостей на изображении. На горизонтальной оси – шкала яркостей тонов от белого до черного, на вертикальной оси - число пикселей заданной яркости.

Определение. **Точечный оператор** - оператор, который определяет значение выходного пиксела по значению только одного входного пиксела. все пикселы обрабатываются независимо друг от друга.

$\boldsymbol{f^{-1}(y) = x}$ y – яркость пикселя на исходно изображении,

x – яркость пикселя после коррекции

Пишем $\boldsymbol{f^{-1}}$ потому что «восстанавливаем» истинное значение яркости по неправильному.

Если гистограмма занимает только часть диапазона, то изображение получается неконтрастным. Можно растянуть гистограмму на весь диапазон, воспользовавшись *линейной коррекцией*.

Суть линейной коррекции заключается в следующем: мы строим такое линейное отображение, чтобы самый яркий пиксел стал белым, а самый темный пиксель стал черным. Линейное растяжение не работает, если в изображении есть хотя бы один белый и хотя бы один черный пиксель. Чтобы улучшить контраст изображения с наличием такого шума используется *робастная (устойчивая) версия метода:* вычисляем линейную коррекцию, чтобы 5% самых темных пикселов стали черными и 5% самых светлых стали белыми.

В случае, когда не помогает линейная коррекция, можно применить нелинейную коррекцию – гамму-коррекцию $y = c \cdot x^{\gamma}$. Также возможна произвольная нелинейная коррекция.

Также в обработке изображений используются такой автоматический метод, как выравнивания гистограммы. Идея заключается в следующем: возьмем неконтрастное изображение, визуализируем гистограмму распределения пикселов по яркости и построим *коммутативную гистограмму*: для каждого значения яркости посчитаем, какая доля пикселей в изображении имеет яркость меньше или равную заданной. На неконтрастных изображениях обычно эта гистограмма имеет резкие перепады, если изображение контрастное, то коммутативное гистограмма – это прямая линия.

Цветокоррекция

Мы можем определить, что цветопередача некорректна по каким-то эталонным примерам – *коррекция по шаблону.* Разумный подход: сфотографировать объект с известным цветом (шаблон), вычислить цветовое преобразование, чтобы цвет объекта на фотографии совпал с нужным.

Оценка параметров цветокоррекции

Если нет цветовых шаблонов, тогда нужно спрогнозировать коэффициенты усиления. Например, с помощью *модели «серого мира».*

Модель серого мира:

- Средний уровень («серый») по каждому каналу должен быть одинаков для всех каналов
- Если цветовой баланс нарушен, тогда «серый» в этом канале больше «серого» других каналов
- Вычислим коэффициенты усиления так, чтобы среднее в каждом канале стало одинаковым

Лекция 2. Сопоставление изображений

Понятие сопоставления изображений

Определение. **Совмещение изображений** (image alignment) – определение совместно наблюдаемой области и совмещение изображений до совпадения общей наблюдаемой области

Определение. **Сопоставление изображений** (image matching) – определение соответствия точек и фрагментов между изображениями (пр. определение пар точек, которые соответствуют одной и той же точке сцены)

Примером совмещения изображений является построение панорамы (мозаики). Близкой задачей к «склейке» изображений является построение карт - аэрофотосъемка, космическая съемка. Например, чтобы построить карту высот, мы находим общие точки на нескольких изображениях одной сцены, по этим точкам совмещаем изображения и потому, как меняются объекты при переходе от одного изображения к другому, мы определяем высоту и строим некоторую трехмерную модель сцены.

Помимо этого, задача сопоставления изображений – это важнейшая часть трехмерной реконструкции по изображениям. Например, если мы хотим скачать из интернета фотографии какого-то объекта и по множеству этих изображений сделать трехмерную реконструкцию объекта. Другой важной областью применения является анализ медицинских изображений. Например, нужно сопоставить изображение конкретного человека с медицинским атласом для того, чтобы понять, какая структура соответствует какой-то структуре мозга, что позволяет произвести диагноз.

Рассмотрим пример - построение панорамы. Это можно сделать по шагам:

1. Найти общие части на изображении
2. Перенести проекции трехмерной сцены на общую картинную плоскость. Например, первое изображение перенести на плоскость, принадлежащую второму изображению и на этой плоскости склеить
3. Цветокоррекция изображений и места «склейки»
4. 2D преобразование изображений в зависимости от ракурсов, с которых они были получены (рис. 12). Самая простая модель - перенос, когда изображения находятся на одной плоскости, выравнены относительно друг друга, то есть перевести одно изображение в другое можно с помощью сдвига. Более сложное евклидово преобразование, когда изображения еще могут быть повернуты. Еще более сложное преобразование с подобием, то есть еще может отличаться масштаб. Далее возможно аффинное и перспективное преобразования.

Модели преобразования изображений

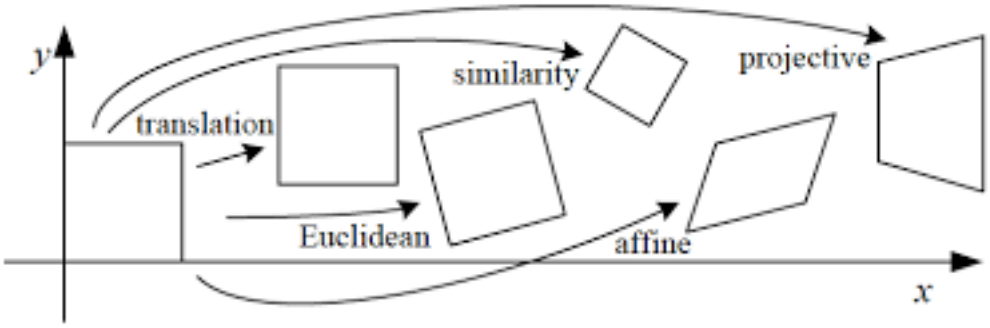


рис. 12

Для удобства преобразований в матричной форме вводятся «однородные» (homogenous) координаты. Перевод из обычных («неоднородных») в однородные:

$(x, y) \implies \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$ - однородные координаты 2D точки изображения

$(x, y, z) \implies \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$ - однородные координаты 3D точки сцены

Перевод из однородных координаты в обычные:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow \left({}^{x}/_{w}, {}^{y}/_{w}\right) \qquad \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow \left({}^{x}/_{w}, {}^{y}/_{w}, {}^{z}/_{w}\right)$$

Обычные двухмерные преобразования:

Перенос (translation) $x' = [I\ t]\ \tilde{x}$

Поворот (rotation) $x' = [R\ t]\ \tilde{x}$ $R = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$

Подобие (similarity) $x' = [sR\ t]\ \tilde{x}$

Аффинное (affine) $x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \tilde{x}$

Для того, чтобы построить перспективное преобразование, посмотрим, как оно устроено в камере: имеется камера обскура (точечное отверстие, через которое проходят все лучи от объекта, которые формируют перевернутое уменьшенное изображение на картинной плоскости, расположенной на фокусном расстоянии). Для математической модели сделаем следующие допущения:

- модель камеры-обскуры – перспективная проекция

- перенесем объект на противоположную сторону
- то же самое фокусное расстояние
- изображение нормальное, не перевернутое

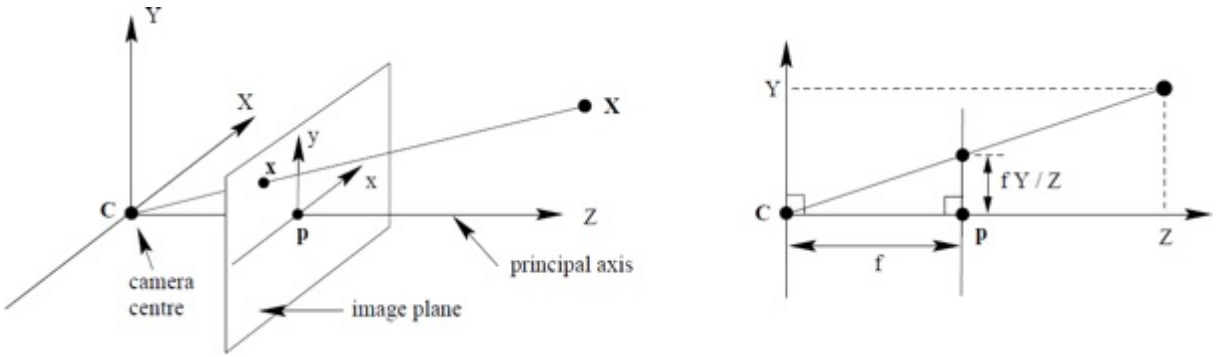


рис. 13

Запишем уравнение перспективной проекции (рис. 13). Для этого возьмем модель камеры и совместим центр проекции с камеры с мировой системой координат. Пусть центр проекции имеет координаты C = (0,0,0), камера будет смотреть в направлении оси Z, и оси мировой системы координат будут выравнены с осями на изображении. Мы хотим построить проекцию некоторой трехмерной точки X = (X, Y, Z) на картинную плоскость. Пусть проекция X на картинную плоскость будет обозначена x= (x, y, f), где f – фокусное расстояние. Точки c, x, X лежат на одной прямой. Тогда простейшее уравнение перспективной проекции:

$$x = f\frac{X}{Z} \qquad y = f\frac{Y}{Z}$$

Простейшее уравнение проекции нужно модифицировать, чтобы оно могло соответствовать произвольному положению камеры. Полное уравнение перспективной проекции в матричном виде записывается в виде x = PX, где X – точка в 3D, x - точка на цифровом изображении (в пикселях), P – матрица проекции.

$\boldsymbol{P = K \cdot [I|0] \cdot C^{-1}}$, K – внутренняя калибровка, $\boldsymbol{I|0}$ - центральная проекция, $\boldsymbol{C^{-1}}$- внешняя калибровка.

$$K = \begin{bmatrix} a_x & s & c_x \\ 0 & a_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f/_{pix} & s & c_x \\ 0 & f/_{pix} & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Внутренняя калибровка отображает точки с идеальной картинной плоскости в пиксели (фокусное расстояние в мм делится на размер пиксела, c_y - принципиальная точка, положение)

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Проекция на идеальную картинную плоскость

$$\mathrm{C}^{-1} = \left(\begin{bmatrix} R & T \\ [0,0,0] & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} R^T & -R^T \\ [0,0,0] & 1 \end{bmatrix}$$

Матрица внешней калибровки определяется положением и ориентацией камеры в мировой системе координат

Предположим, мы сняли одну и ту же сцену с 2 разных ракурсов и у нас имеются матрицы проекций двух камер. У нас есть одно изображение, полученное из точки O, мы хотим построить второе изображение из точки O', как выглядела бы примерная сцена, которая запечатлена на первом изображении. Чтобы построить вид этой сцены с другого ракурса, одного изображения недостаточно, нужна информация о «глубине».

Однако есть особы случай (г*омография*) (рис. 14), когда оба изображения получены с камер, у которых совпадает центр проекций. Если два снимка сделаны так, что центр проекций совпал, то тогда трехмерное преобразование – это просто перспективная проекция и глубину знать необязательно. в однородных координатах можем записать как x' = Hx. У этой матрицы 8 степеней свободы, то есть мы ее можем определить с точностью до масштаба.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11}\ h_{12}\ h_{13} \\ h_{21}\ h_{22}\ h_{23} \\ h_{31}\ h_{32}\ h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

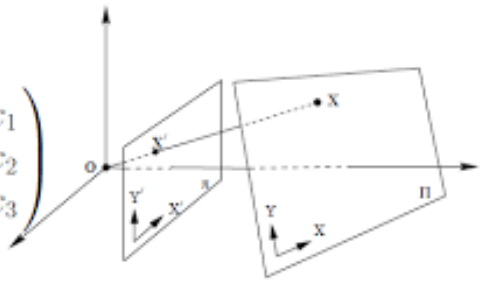


рис. 14

Вывод: если строим панораму или карту местности (считая ее «плоской»), то одно изображение преобразуется в другое с помощью *гомографии* (рис.15).

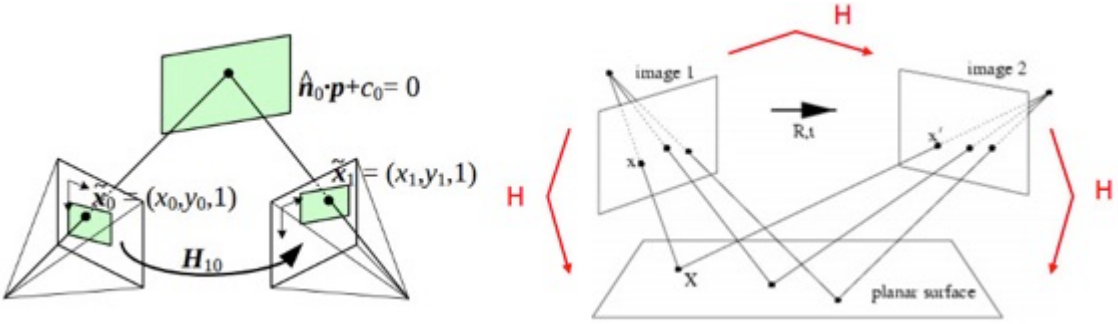


рис. 15

Основная идея использования гомографии для совмещения изображений – это проецирование снимков на общую плоскость. Однако с помощью такого подхода можно построить только панораму $< 180°$.Если мы будем проецировать изображение не на плоскость, а на поверхность цилиндра, то мы сможем получить частичную панораму (360° по одному измерению). Преобразования при этом представляют из себя параллельный перенос.

Чтобы получить полную панораму нужно все проецировать на поверхность сферы.

Координаты на цилиндре (θ, h)	Координаты на сфере (θ, φ)
$(sin\theta, h, cos\theta)\ \chi\ (x, y, f)$	$(sin\theta cos\varphi, sin\varphi, cos\theta cos\varphi)\ \chi\ (x, y, f)$
Преобразование:	Преобразование:
$x' = s\theta = s \tan \frac{x^{-1}}{f}$	$x' = s\theta = s \tan \frac{x^{-1}}{f}$
$y' = sh = s \frac{y}{\sqrt{x^2 + f^2}}$	$y' = s\varphi = s \tan \frac{y}{\sqrt{x^2 + f^2}}^{-1}$

После выбора определенной модели (гомография или параллельный перенос) нам нужно вычислить преобразования, сдвиг между получившимися фрагментами. Для этого существует два основных подхода.

Прямое выравнивание (Direct Alignment)

Подход, который заключается в поиске преобразования, минимизирующего ошибку сопоставления.

- Выберем модель преобразования. Например, параллельный перенос T при построении цилиндрической панорамы
- Зададим какие-то параметры преобразования T
- Найдем область пересечения с учетом этой модели преобразования

- Посчитаем «качество» сопоставления через попиксельное сопоставление изображений в области пересечения
- Перебираем параметры преобразования до тех пор, пока ошибка не минимизируется

Чтобы вычислить ошибку совмещения, можно использовать метрики. Простейшей метрикой является *метрика среднего квадратичного отклонения*: считаем разницу для каждого пикселя и берем сумму квадратов разности.

$E_{SSD}(u) = \sum_i [I_1(x_i + u) - I_0(x_i)]^2 = \sum_i e_i^2,\ e_i = I_1(x_i + u) - I_0(x_i)$

Рабастные метрики (SRD)

$$E_{SRD}(u) = \sum_i \rho(I_1(x_i + u) - I_0(x_i)) = \sum_i \rho(e_i)$$

$\rho(e)$ - робастная функция, которая при больших ошибках растет медленнее, чем квадрат

Сумма модулей (Sum of Absolute Distances)

$$E_{SAD}(u) = \sum_i |I_1(x_i + u) - I_0(x_i| = \sum_i |e_i|$$

Дифференцируемый вариант – *функция Geman-McClure*:

$$\rho_{GM}(x) = \frac{x^2}{1 + x^2/a^2}$$

Корреляционные метрики:

Cross-Correlation (CC)

$$E_{CC}(u) = \sum_i I_1(x_i + u) I_0(x_i)$$

Normalized Cross-Correlation (NCC)

$$E_{NCC}(u) = \frac{\sum_i (I_0(x_i) - \overline{I_0})((I_1 x_i + u) - \overline{I_1})}{\sqrt{\sum_i (I_0(x_i) - \overline{I_0})^2}\,((I_1 x_i + u) - \overline{I_1})^2}$$

$$\overline{I_0} = \frac{1}{N}\sum_i I_0(x_i),$$

$$\overline{I_1} = \frac{1}{N}\sum_i I_1(x_i + u)$$

Подходы поиска параметров

После выбора функции ошибки мы ее оптимизируем по параметрам:

- Дискретизация параметров
  - Перебор все вариантов (Grid Search)
  - Какой-то метод дискретной оптимизации
- Непрерывные параметры
  - Градиентный спуск

Если преобразования слишком больший, для прямого сопоставления изображений используется итеративный поход – *Пирамида Гауссиан* (рис. 16). Построим несколько уменьшенных копий изображения. Для этого будем сглаживать фильтром Гаусса изображение и брать каждый второй пиксель, уменьшив таким образом изображение в 2 раза на каждом этапе. С каждым уровнем пирамиды абсолютный размер смещения будет уменьшаться. После этого воспользуемся иерархическим методом: возьмем два самых маленьких изображения и совместим их, взяв это совмещение как начальное приближение для совмещения на более высоком уровне. На более высоком уровне уточним это преобразование, увеличим его в два раза и перейдем к следующему уровню. И так до тех пор, пока не дойдем до исходного изображения. Для ускорения оценки на каждом шаге мы можем делать неполной перебор, например, применить метод градиентного спуска.

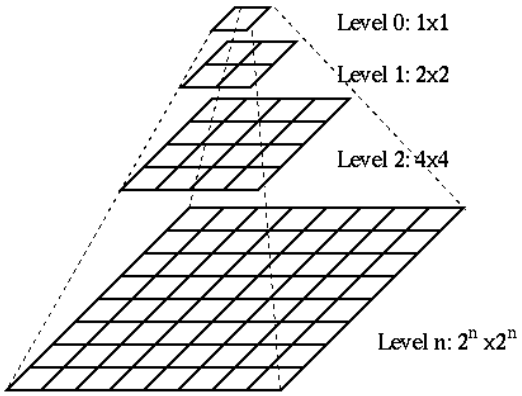


рис.16

То есть базовый подход заключается в следующем:

- Выбираем модель преобразования
- Выбираем функцию ошибки (согласования изображений)
- Применяем метод оптимизации для поиска оптимальных параметров

Однако есть и другой поход, основанный на особенностях объекта, хорошо различимых фрагментах, а именно на:

- «особенностях» (features)
- «характеристических точках» (characteristic points)
- «ключевых точках» (key points)
- «локальных особых точках» (local feature points)

Такие характерные фрагменты позволяют справиться с изменениями ракурса, масштаба и перекрытиями, что позволяет решать задачу сопоставления даже в сложных случаях.

Локальная (особая) точка p изображения должна обладать «характерной окрестностью» D, т.е. отличаться от всех точек в некоторой окрестности p. Таким образом, для определения того, что точка является «характерной» достаточно проанализировать ее локальную окрестность.

Существует два основных класса характерных особенностей: *уголки* (corners) и *пятна* (blobs).

Монотонный регион - регион, при смещении в любом направлении которого нет изменений окрестности.

Край – регион, при смещении вдоль которого изменений окрестности нет, а при смещении в другом направлении есть изменения.

Уголок – регион, в котором при смещении в любом направлении есть изменения окрестности.

Один из самых известных методов нахождения уголков в изображении является *Детектор Харриса и Стивенсона*. Он ищет такие точки (x, y), окрестность которых меняется при любом сдвиге (x+u, y+u).

Изменение окрестности точки (x, y) при сдвиге $[u, v]$:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - \ I(x,y)]^2$$

$w(x,y)$ - функция окна, $I(x,y)$ - яркость, $I(x+u, y+v)$ - сдвиг яркости

Разложение в ряд Тейлора второго порядка I (x, y) вокруг (x, y) (билинейная интерполяция вокруг при маленьких сдвигах).

$$[I(x+u, y+v) - \ I(x,y)]^2 \cong \left[I(x,y) + I_x u + I_y v - \ I(x,y)\right]^2 = \left[I_x u + I_y v\right]^2$$
$$= I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2 = (u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} (u,v)^T$$

Чем больше градиенты в рассматриваемой окрестности, тем больше отклонение. Таким образом, изменение окрестности можно свести к $E(u,v) \approx [u\ v]\ M \begin{bmatrix} u \\ v \end{bmatrix}$, где M - матрица 2x2, вычисленная по частным производным:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \qquad M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \quad I_y]$$

$$= \sum \Delta \, \mathrm{I} (\Delta \, I)^T$$

M – матрица моментов. Если градиенты выровнены по осям (вертикальные или горизонтальные), то $M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$.

Если одно из λ близко к 0, то тогда это не угол и нужно искать другие точки. В общем случае M – симметричная, поэтому ее можно привести к диагональному виду: $M = R^{-1} D R = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$, где R - ортогональная матрица из собственных векторов M, D – диагональная матрица из собственных значений M. Матрицу M можно визуализировать в виде эллипса (рис. 17), у которого длины осей определены собственными значениями, а ориентация определена матрицей R. Уравнение эллипса: $[u \; v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = const.$

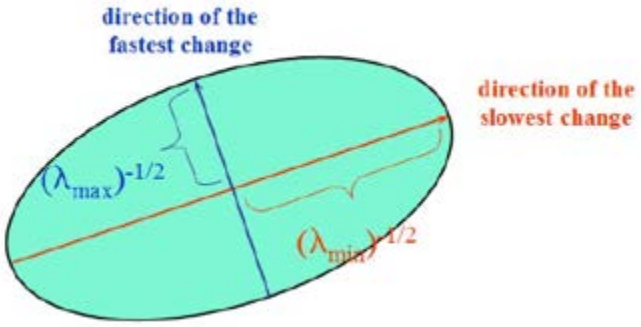


рис.17

Если эллипс сильно вытянут, значит, сильные градиенты в одном направлении и слабые в другом, то есть изображение похоже на *край.* Мы можем классифицировать все точки изображения по собственным значениям матрицы производных M (рис.18). Если оба собственных значения малы, то $E(u,v)$ не меняется по всем направлениям и является плоской областью. Если одно собственное значение сильно больше другого и больше 0, то в одном направлении градиенты сильные, в другом слабые, этой край. Если оба собственных значения велики и примерно равны, значит это потенциальный угол.

Однако вычислять собственные значения не очень удобно, поэтому в методе Харриса была предложена R функция отклика, которая оценивает, насколько точка похожа на угол.

$R = detM\ - k(traceM)^2$

$\det M = \lambda_1 \lambda_2$, $traceM = \lambda_1 + \lambda_2$, k = 0.04-0.06

Если функция отклика больше нуля, тогда собственные значения примерно равны, и точка является углом. Если значение функции маленькое, то она соответствует плоской области. Если значение меньше нуля, то точка соответствует краю.

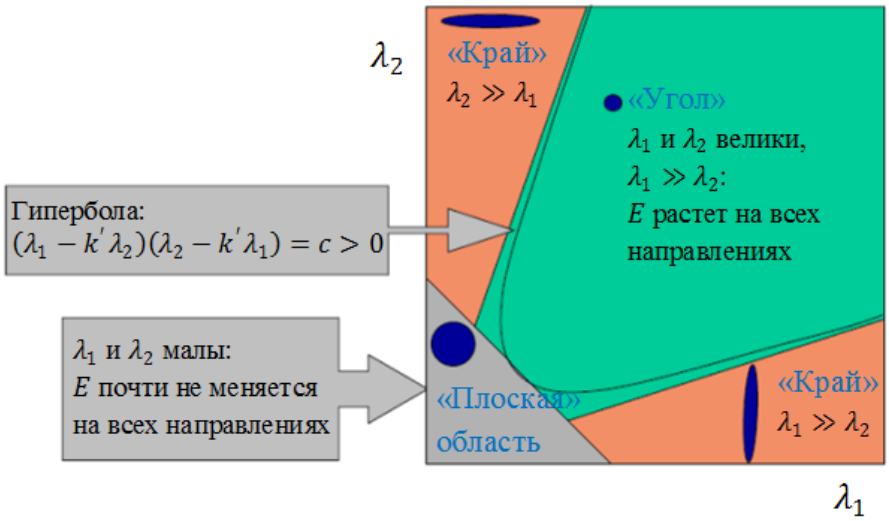


рис. 18

Алгоритм детектора Харриса

1. Вычислить градиент изображения в каждом пикселе (с использованием гауссового сглаживания)
2. Вычислить матрицу вторых моментов M по окну вокруг каждого пикселя
3. Вычисляем отклик угла R
4. Выбираем только те точки, которые больше порога R
5. Находим локальные максимумы функции отклика (non-maximum suppression) по окрестности заданного радиуса

Детектор Харриса *инвариантен* к вращению изображени (рис.19), т.к. при вращении эллипса его форма (собственные значения) остаются неизменными. То есть **отклик угла R** инвариантен **относительно вращения изображения.**

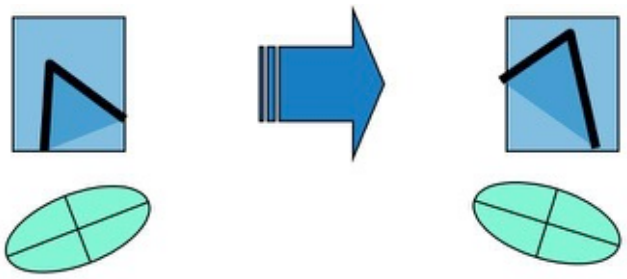


рис.19

Однако детектор Харриса не устойчив к масштабированию. Для того, чтобы находить точки с учетом их характерного масштаба был предложен *метод Лаплассиан Гауссианы* – центрально-симметричный оператор поиска блобов в 2D. Идея метода свелась к следующему: в качестве функции оценки размеров характерной окрестности использовалась свертка, веса которой рассчитываются как Лаплассиан функции Гаусса, то есть как вторая производная от Гауссианы (рис. 20).

$$\nabla^2{}_{norm} = \sigma^2\left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}\right)$$

Характеристический размер определяется как масштаб, на котором достигается максимум отклика Лаплассиана.

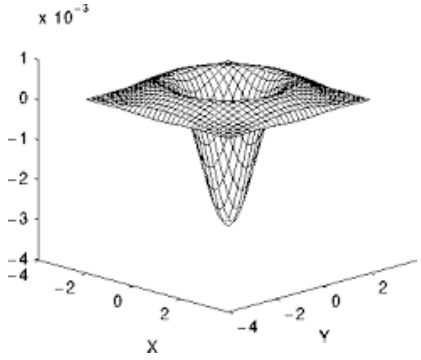


рис.20

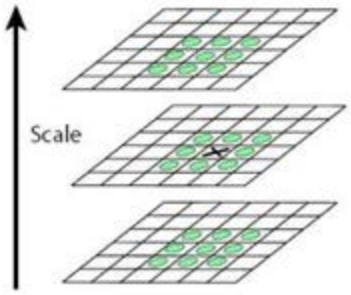


рис.21

На основе Лаплассиана можно сделать алгоритм для нахождение локальных особых точек в виде пятен, многомасштабный детектор блобов (рис. 21):

1. Свертываем изображение нормализованным фильтром Лаплассиана на разных масштабах
2. Ищем максимум отклика Лаплассиана

Метод нахождения локальных точек с помощью Лаплассиана получил название LoG (Laplassian of Gaussin)

$L = \sigma^2\big(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)\big)$ – Лаплассиан

$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$ - разница Гауссиан, более эффективная реализация (рис. 22).

Мы сворачиваем изображение до тех пор, пока σ не станет больше единицы, уменьшаем изображение в 2 раза по всем осям и снова сворачиваем с помощью фильтра Гаусса. Для того, чтобы аппроксимировать и посчитать разницу, мы берем разницу соседних слоев. За счет уменьшения изображения мы существенно ускоряем операцию расчета.

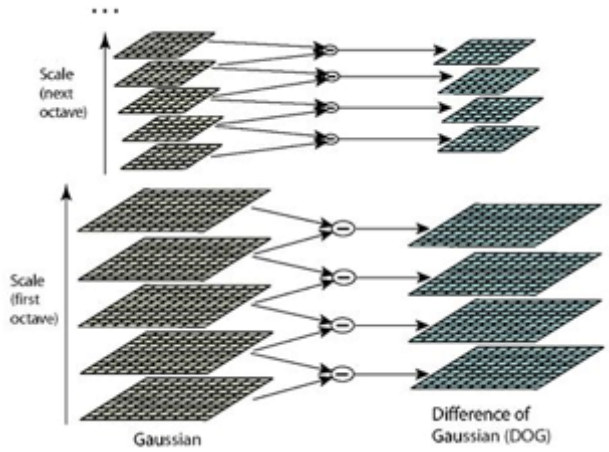


рис.22

Определение. **Дескриптор** (Descriptor) – вектор признак окрестности точки, который позволяет индифицировать точку.

Простейший подход для построения дескриптора и сопоставления изображений:

- Возьмем квадратные окрестности со сторонами параллельными к строкам и столбцам изображения
- Яркости пикселов будут признаками
- Сравнивать будем два изображения попиксельно (SAD, SDD)
- Такая окрестность инвариантна только сдвигу изображения

Однако у простой окрестности есть свои недостатки: детектор точек инвариантен повороту, а окрестность нет, небольшие сдвиги (ошибки в нахождении точки) делают невозможным попиксельное сравнение. Для решения этой проблемы в 2004 году был предложен метод SIFT (Scale-Invariant Feature Transform) устойчивый к изменениям освещенности и небольшим сдвигам:

- Определение положения и характерного масштаба особенности с помощью детектора DoG
- Определение доминантной ориентации особенности по градиентам
- Использование статистик по направлению градиентам

Для построения дескриптора в методе SIFT используется подсчет гистограммы ориентаций градиентов. Для этого

- Вычислим направление градиента в каждом пикселе
- Квантуем ориентации градиентов на 8 ячеек (направлений), пометим каждый пиксель номером ячейки
- Посчитаем гистограмму направлении градиентов, для каждой ячейки посчитаем количество пикселов

Идея ориентации фрагмента заключается в следующем: найти доминантное направление градиентов пикселей в окрестности точки. Для этого нужно выбрать в гистограмме ячейку с максимальным значением и взять ее направление за доминирующее. Далее повернем фрагмент так, чтобы доминантное направление градиента было направлено вверх. Если локальных максимумов несколько, считаем, что несколько точек с разной ориентацией.

Теперь для каждой характерной точки мы знаем характеристические масштаб и ориентацию. Выберем соответствующую прямоугольную окрестность (rotation invariant frame) и приведем ее к стандартному размеру (масштабируем).

Построение дескриптора (рис. 23):

- Для учета пространственного распределения свойств разделим окрестность на блоки сеткой, в каждом блоке посчитаем, свою гистограмму градиентов
- Сетка 4x4, в каждой гистограмма с 8ю ячейками
- Стандартная длина вектора-дескриптора – 128 (4*4*8)
- Для сравнения можно использовать SSD или другие метрики, учитывающие, что дескриптор SIFT – это гистограмма.

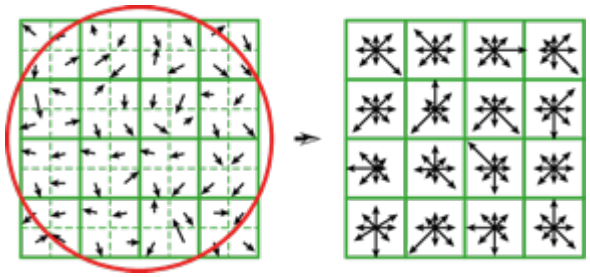


рис.23

Чтобы сделать дескриптор устойчивым к небольшим сдвигам, нужно использовать ядро при расчете гистограммы, то есть

- Взвешиваем вклад пикселей по ядру
- Веса рассчитываем в зависимости от близости к центру по Гауссиане

- Небольшие изменения в локализации (положении, масштабе и ориентации) будут приводить к небольшим изменениям дескриптора

Вывод: дескриптор SIFT специфичен, устойчив к изменениям освещения и небольшим сдвигам, схема SIFT (детектор, выбор окрестностей, дескриптор) оказалась очень эффективным инструментом для анализа изображений.

Чтобы посчитать преобразование, например, *гомографию*, предположим, что среди пар (x, x') (известные соответствия на двух изображениях) нет никаких ложных соответствий, тогда мы можем решить задачу, выбрав 4 точки и составив систему уравнений $\lambda x' = Hx$. Каждое соответствие даст 2 уравнения на параметры H, 4 соответствия дадут 8 уравнений на 8 неизвестных. Решив эти уравнения, вычислим матрицу гомографии. Но на практике есть ложные соответствия, которые могут испортить вычисление модели, поэтому нужно применить *робастные методы (методы, которые позволяют оценить параметры по данным, часть из которых является ложными)*, устойчивые к ошибкам.

Методы оценки парамтеров

Самым известным методом является рандомизированный метод, первым из которых был RANSAC. Идея метода заключается в следующем: проводим оценку не по всем данным, а по выборке, не содержащей вбросов. Выборки строим случайным образом, потому что заранее неизвестно, какие элементы данных являются выбросом. Далее по каждой выборке строим гипотезу и среди гипотез выбираем ту, которая лучше всего согласуется со всеми данными.

Базовая схема RANSAC

Повторяем N раз:

i. Построение случайной выборки $S \subset X$
ii. Построение гипотезы по выборке S
iii. Оценка качества гипотезы Θ по набору исходных данных X
iv. Если гипотеза Θ лучше предыдущих, запоминаем ее

После завершения итераций:

i. Построение чистой выборки X' путем фильтрации выбросов, не удовлетворяющих Θ_{best}
ii. Уточнение гипотезы Θ_{best} по X'

У базового метода есть проблема: мы считаем число голосов, которые удовлетворяют гипотезе, сравнивая ошибку с некоторым порогом. Если мы порог выберем неверно, например, слишком большим, то неверные гипотезы будут похожи на верные, если порог слишком маленький, то даже верные примеры не будут ему удовлетворять.

Чтобы такого не происходило, была предложена модификация метода M-SAC. Для этого в качестве целевой функции берем M-оценку:

$$R(\theta) = \sum_i p(\varepsilon_i(\theta)^2), p({\varepsilon_i}^2) = \begin{cases} {\varepsilon_i}^2, & {\varepsilon_i}^2 \le T^2 \\ T^2, & {\varepsilon_i}^2 > T^2 \end{cases}$$

M-оценка – функция, которая равна квадрату ошибки, если ошибка меньше заданного порога, и квадрату порога, если ошибка меньше заданного порога. Поэтому, если какая-то точка «плохая» (лежит очень далеко), то для оценки гипотезы мы используем не эту бесконечно далекую точку, а заменяем ее ошибку на некоторое фиксированное значение. Если же точка удовлетворяет условиям, то чем лучше она подходит, тем меньше вклад в ошибку вносит.

Теперь вычислим ошибку для гомографии – ошибку переноса. Пусть (x,x') – пара соответствующих точек, H – гомография. Тогда $\tilde{x} = Hx, \tilde{x} = H^{-1}x'$ – образы точек, полученных с помощью гомографии. Ошибка переноса (transfer error), т.е. ошибка, которая показывает, насколько плохо гомография переносит точки изображения:

$$r(x, x') = d^2(x_i', \tilde{x_i}) + d^2\left(x_i, \widetilde{x_i'}\right)$$

Затем мы задаем порог на эту ошибку. В итоге получаем **алгоритм сопоставления**

- Дано $\{(x, x')\}$ – набор пар соответствующих точек на изображениях I и I'
- Вычислим модель H гомографии между изображениями I и I' по парам соответствующих точек с помощью RANSAC
- Отфильтруем выбросы в $\{(x, x')\}$ по модели H
    - Если ошибка r(x, x') >T, тогда (x,x') –выброс (ложное соответствие)
- Уточним модель H
    - Метод наименьших квадратов по всем оставшимся точкам
    - Итеративный пересчет H и истинных/ложных соответствий

Лекция 3. Задача классификации и поиска похожих изображений

Понятие классификации

Примером задачи классификации является *бинарная классификация*, когда есть вопрос про изображение, который допускает бинарный ответ: да или нет. Например, «есть ли на этом изображении пешеход?», $y \in [0,1]$, 1 - да, 0 – нет.

Многоклассовая классификация - семейство задач, в которых мы отвечаем на вопрос про изображение, который допускает один из нескольких ответов, обычно множество ответов ограничено. Ответ выдаётся в виде вероятности правильного ответа, то есть абсолютной (эталонный) ответ $y \in [1, S]$ – дать метку класса, которому соответствует данное изображение, но поскольку алгоритм не всегда может это сделать, ответ выдается в виде вероятности $y_i \in [0,1],\ i = [1, S], \sum y_i = 1$.

С помощью задач классификации изображений, мы можем определять атрибуты изображение.

Определение. **Атрибуты** – «типичные» характеристики объекта

Например, для человека атрибутами являются пол. возраст, раса, наличие бороды, усов или улыбки и т.п.

К задаче классификации также относится определение человека по лицу, *верификация*. Самая базовая задача распознавания человека – сравнение двух изображений. Похожей задачей является *поиск изображений по содержанию* (content-based image retrieval), которая заключается в следующем: есть коллекция изображений, мы хотим найти в ней похожее изображение на изображение-запрос, изображения сравниваются по визуальному сходству. Таким образом, поиск изображения по содержанию – это задача классификации с открытым и заранее неизвестным набором классов.

Задача классификации изображений

В общем случае задачу классификации можно декомпозировать на следующие этапы (рис.24):

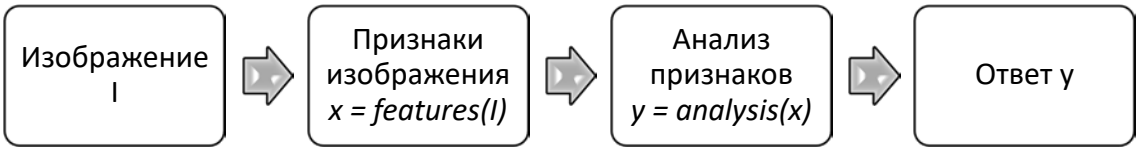


рис.24

Для того, чтобы решать задачу с помощью машинного обучения, нам потребуется:

- Тестовая (обучать классификатор) и обучающая выборка (оценивать работу классификатора)
- Критерий качества работы
- Метод построения вектор-признака изображения

Далее применим выбранный метод машинного обучения для построения классификатора. С какими выборками при классификации изображений обычно сталкиваются? Одна из первых выборок – CIFAR-10, CIFAR-100. Эти выборки состоят из очень маленьких изображения размером 32x32 пиксела.

CIFAR-10	CIFAR-100
10 классов	100 классов
60000 изображений	60000 изображений
5000 обучающих и 1000 тестовых на класс	500 обучающих и 100 тестовых на класс

Другая эталонная коллекция, которая актуальна до сих пор – коллекция ImageNet. Цель этой коллекции заключалась в том, чтобы собрать коллекцию с 1000 изображений на каждый из 117 тыс. synsets (понятий). В итоге была создана коллекция из 14,197,122 изображений, отвечающих 21841 понятиям. Некоторые (1,034,908) изображения были с рамкой, выделяющей объект. Подклассом этих коллекций подразумевался достаточно широкий класс объектов. Например, класс машины, то есть все машины вне зависимости от марки. Но мы можем сформулировать задачу классификации с выделением более четких подклассов, например, марки машины. Такие задачи называются fine-graded classification.

Остро встает вопрос: как правильно готовить эталонные коллекции? Практический ответ: разметка изображений вручную. Так с помощью 150,000 волонтеров астрономы решили задачу классификации галактик совершенно бесплатно. Волонтеры сделали более 60 млн. меток на сайте www.galazyzoo.org. По такому же принципу работают капчи при регистрации или скачивании информации из интернета. Сейчас есть несколько интернет бирж, например, amazon mechanical torg, которые сводят заказчиков с людьми, готовыми решат задачу разметки изображений за плату.

После составления большой эталонной коллекции необходимо определить критерий качества. Для этого используются такие метрики, как

- % правильно классифицированных изображений (для какого % тестовой выборки классификатор сработал верно)
- ранговая метрия Rank X
    - если классификация многоклассовая, мы ранжируем все выходы по качеству
    - если истинный ответ попадает в первые X выходов, тогда результат считается верным

Далее нужно определить признаки. Поскольку мы работаем с изображением, какие бы признаки мы не конструировали, они будут опираться на признаки, которые мы можем вычислить по пикселям изображения:

- *цвет*: пространства цветов RGB, HSV, L*a*b*
- *градиенты в каждом пикселе*
- *наличие и ориентация края в каждом пикселе*

Можно ли напрямую использовать признаки всех пикселей для построения вектора изображения? Можно, если все изображения для классификации будут одинакового размера. Однако признаков слишком много и классификаторам тяжело извлечь важную информацию, поэтому вместо использования пикселов напрямую, лучше считать разные статистики распределения характеристик. Гистограммы- стандартный способ непараметрического описания распределения признаков. Если признак дискретный, построить гистограмму легко, но, если признак изменяется непрерывно, например, направления градиента, нам потребуется дискретизация, квантование. Так для направления градиента

- Разбиваем 360 градусов на 8 секторов
- Каждый сектор нумеруем от 1 до 8
- Считаем число пикселов, градиенты которых попадают в соответствующие сектора

Эту же задачу можно решить адаптивно с помощью методов кластеризации, путем группирования данных «по похожести» в кластеры и подсчета объектов, которые входят в тот или иной кластер. Например, можно кластеризовать изображение по цветам и построить кластеры часто встречающихся цветов. Для кластеризации можно использовать любой метод, например, метод К-средних и производных.

Построение гистограммы, например, гистограммы распределения цветов имеет основной недостаток – мы игнорируем пространственную информацию. Чтобы сохранить пространственную информацию хотя бы частично, мы можем считать гистограмму не по всему изображению, а по отдельным секторам. Стандартный подход построения вектор-признаков такой:

1. Разбиваем изображение на ячейки
2. Считаем гистограммы в каждой ячейке в несколько этапов (считаем гистограмму всего изображения, потом поделенного на ячейки изображения и поделенных на ячейки, получив на последнем этапе 16 ячеек)

Гистограммы градиентов, которые мы строим по всему изображению получили свое название HOG (Histogram of Oriented Gradients), по сути тот же самый дескриптор SIFT, который мы посчитали по всему изображению.

Общая схема для признаков (рис.25)

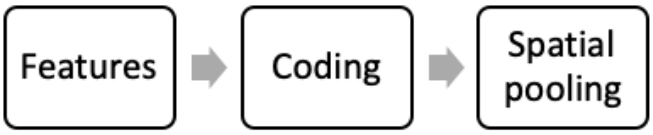


рис.25

Features – признаки, которые мы извлекаем (цвет, градиенты и.т.д.)

Coding – кодирование, каким образом мы используем эти признаки: явно (без кодирования), квантование, адаптивное квантование

Spatial pooling – пространственная агрегация, выбор областей для агрегации и метод агрегации

Метод К-средних

Хотим разделить изображение на 10 классов К=10. Каждый пиксель описывается параметрами RGB $o = (R, G, B)$. Составляем все множество пикселей по некоторой эталонной коллекции. Визуализируем распределение точек в трехмерном пространстве (т.к. RGB) (рис.26). В зависимости от изображения точки неравномерно занимают пространство RGB- куба, образуя группы. Такие группы называются кластерами. Теперь случайным образом возьмем 10 точек. Будем считать, что эти точки – это начальное приближение центров кластеров, то есть это гипотеза того, где располагаются кластеры. Далее для каждой точки определяем, к какому кластеру эта точка ближе всего. Теперь пересчитаем центры кластеров, взяв в качестве кластера все точки, которые оказались ближе к определенному выбранному центру, и найдем новый центр кластера, взяв среднее значение.

На базе методов кластеризации сделано много методов построения признаков, которые в совокупности можно называть «мешком слов». Идея мешков слов появилась из задачи классификации текстов: проигнорировав порядок слов в предложении и посчитав частоты, с которыми встречаются те или иные слова в предложении, мы можем классифицировать текст достаточно точно, особенно, если выборка, из которой берутся документы, специфическая.

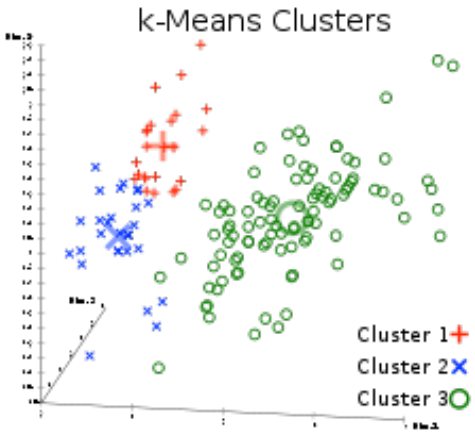


рис.26

Анализ текстуры

Первые методы распознавания изображения, которые использовали мешок слов, были методы анализа текстуры.

Определение. **Текстура** - свойство поверхности, преимущественная ориентация элементов, составляющих материал.

Для анализа текстур предложили следующий способ (рис.27):

- Выберем свертку (фильтр), чувствительный к краям определенной ориентации (например, производная гауссианы по произвольному направлению)
- Результат фильтрации сгладим, усилив сильные отклики и подавив слабые отклики
- В результате будут «подсвечены» области, содержащие текстуру с краями заданной ориентации

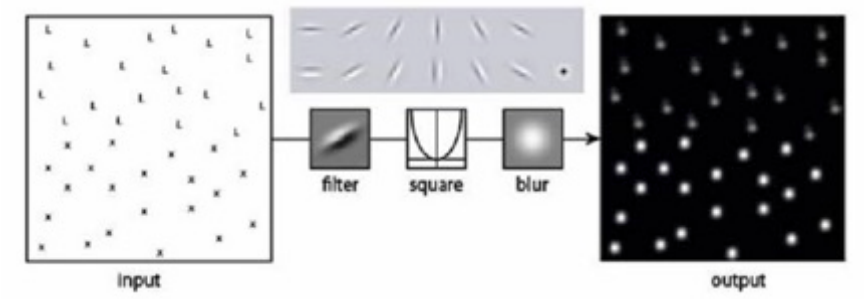


рис.27

Для того, чтобы классифицировать множество разных текстур, логично сглаживать изображение не одним фильтром, а набором фильтров, который будет чувствителен к

разным ориентациям и не только к краям. Такой набор сверток называется *банком фильтров*. Каждый пиксель изображения после обработки банком фильтров дает вектор признаков, который эффективно описывает локальную текстуру окрестности пикселя. Теперь мы можем эти векторы кластеризовать и построить словарь. Применим метод К-средних для выборки, состоящей из векторов-признаков всех пикселов, только теперь вектор-признак – это не цвет пиксела, а отклик на банк фильтров. После кластеризации получаем набор классов.

Определение. **Текстон** - характерный базовый фрагмент текстуры изображения

Теперь мы можем проанализировать изображение, сопоставив пикселу ближайший фрагмент из словаря по набору векторов-признаков. Визуализировать это можно с помощью карты цветов, сопоставим каждому текстону свой цвет. Теперь в качестве признака области изображения мы можем брать частоту встречи текстонов в области и классифицировать, какая часть изображения соответствует какой текстуре. Однако такое построение локальной окрестности слишком медленно. Поэтому в 1994 году для описания текстуры предложили такой признак, как *локальный бинарный шаблон* (local binary patterns) (рис.28). Идея локального шаблона, следующая:

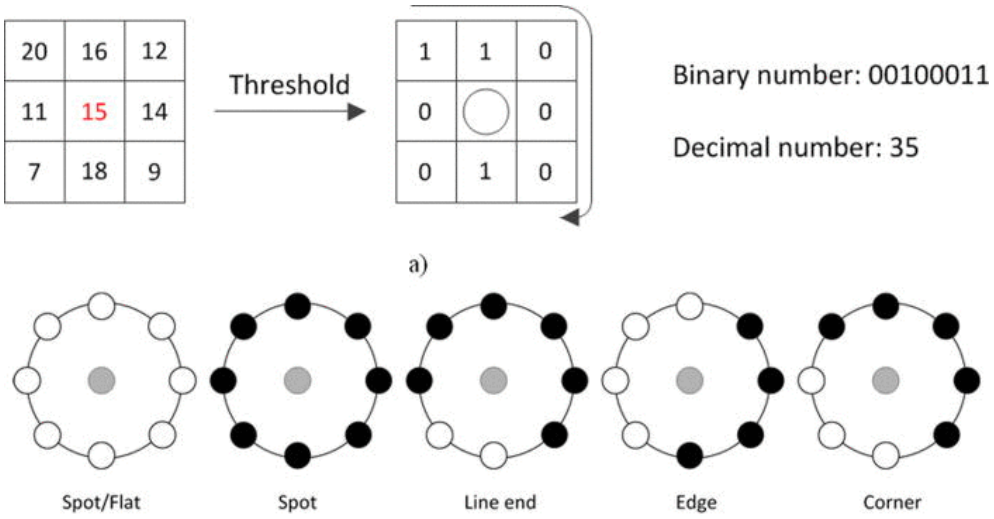


рис.28

- Возьмем 8 соседей рассматриваемого пиксела
- Каждого «соседа» сравним по яркости с центральным элементом, если сосед ярче, чем центральный элемент, сопоставим ему 1, если менее яркий – 0.
- Обойдем 8 соседей в определенном порядке и получим бинарный код длины 8, который соответствует числу от 0 до 255. Назовем его кодом локального бинарного шаблона. Таким образом, каждому пикселу мы сопоставим бинарный код, который зависит от его 8 соседей. При выборе соседей можем построить

окружность, выбрав 8 точек на равном расстоянии друг от друга, и по этим 8 точкам построим код.

Вместо того, чтобы сопоставлять каждый пиксел текстону, мы можем сразу рассчитать для него код локального бинарного шаблона.

Применение LBP:

- Изображение разбивается на области, в каждой из которых применяются LBP операторы к каждому пикселю и строится гистограмма
- Объединение гистограмм – LBP дескриптор для изображения
- Для пары изображений считается разность дескрипторов по какой-нибудь метрике (например, Хи-квадрат)

У локальных бинарных кодов есть много модификаций, одна из которых *однородные бинарные коды.* Рассиотрим бинарные коды от разных чисел и посчитаем число переходов от 0 к 1 и обратно. Однородными кодами назовем те, у которых переходов не больше, чем 2.

- 00000000 – 0 переходов
- 01110000 – 2 перехода
- 11001111 – 2 перехода
- 11001001 – 4 перехода
- 01010010 – 6 переходов

Для всех пикселов изображения однородные коды составляют от 80 до 90% всех кодов. Для упрощения все неоднородные коды объединяют в один код и считают однородные коды и все остальные. Это позволяет закодировать код эффективнее, менее, чем 8 битами.

Есть некоторая интерпретация, почему бинарные коды лучше работают: можно провести связь локальных бинарных кодов и линий уровней изображений.

Линии уровня – это линии, значения вдоль которой одинаковы. Например, можно построить линии уровней яркости изображений, то есть линии, вдоль которых яркость изображений одинакова. Соответственно бинарный код будет показывать, как линия уровня проходит вдоль изображения.

Дальнейшим развитием этого метода стал метод «визуального слова».

Определение. **«Визуальное слово»** – часто повторяющийся фрагмент изображения («visual word»).

В отличие от текстонов мы считаем не признаки одного пиксела, сравнивая его с окрестностью, а в качестве описания берем всю окрестность целиком и пытаемся охарактеризовать ее с помощью какого-то дескриптора.

Схема метода «мешок слов»

1. Построение словаря
    - Извлечение фрагментов из обучающей выборки
    - Кластеризация и построение «визуального» словаря
2. Построение дескрипторов
    - Извлечение фрагментов из изображения
    - Квантование фрагментов по словарю
    - Построение гистограммы частот визуальных слов

Для извлечения фрагментов мы можем 1) разбить все изображение на фрагменты, например, с помощью сетки или 2) взяв особые точки методом SIFT и только вокруг них рассматривать фрагменты.

Для описания каждого фрагмента вектор-признаком можно воспользоваться SIFT:

1. Строим дескриптор SIFTдля каждого фрагмента
2. Применим метод кластеризации К-средних
3. Получим заданное число слов в словаре. Словарь готов.

После построения словаря каждое изображения отображаем по этому словарю в гистограмму частот «визуальных слов». У визуальных словарей очень много особенностей. Например, если словарь маленький, слова не могут описать все особенности, если словарь большой, то возможно переобучение. Также, чем больше слов в словаре, тем медленнее все работает.

Резюме мешка слов:

- Мешок слов используется как сам по себе, так и совместно с другими признаками изображения
- Часто используются совместно и разреженная, и плотная версии: мешок слов по характеристическим точкам (bags of visual words) и плотные слова (dense words)

Резюме классификации:

- Задача классификации начинается с построения тестовой выборке, а затем обучающей
- «Классический» подход к классификации – какой-то из эвристических методов вычисления признаков, и затем построение классификатора с помощью машинного обучения по выборке
- Основные признаки - гистограммы цветов, HOG, BoW, LBP признаки, с использованием решеток и пирамид для учета пространственной информации

Поиск похожих изображений

Смежной задачей является задача поиска похожих изображений. В этом случае классификация с открытым и заранее неизвестным набором классов. Различают визуальное подобие и семантическое. Какие изображения мы считаем похожими? Самая простая формулировка – это поиск *полудубликатов (near-duplicates).*

Определение. **Полудубликаты** – слегка измененная версия изображения (ракурс, цвет)

Оценивать точность метода поиска похожих изображений достаточно сложно. Для этого мы сачала считаем среднюю точность для каждого запроса (average precision), строим кривую точности полноты, задаем количество примеров, которые мы хотим найти и считаем, сколько примеров правильных и какую долю правильных содержащихся объектов мы нашли. Далее усредняем все результаты, получаем некоторую среднюю точность работы (mAP – mean average precision).

Общая схема поиска изображений

Построение индекса коллекции (рис.29)

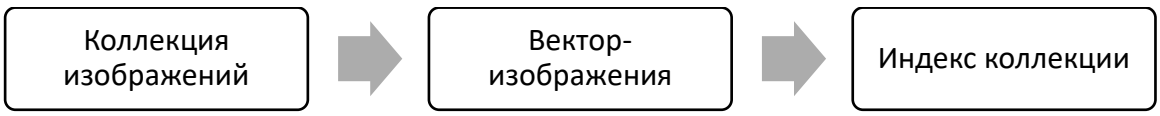


рис.29

Поиск изображения в коллекции (рис.30)

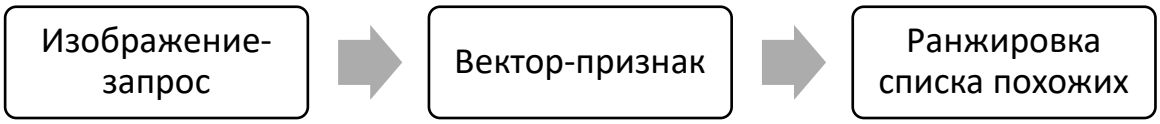


рис.30

Основная проблема для задачи поиска похожих изображений – это скорость и размер памяти. Для того, чтобы все работало быстро, нужно, чтобы векторы-признаки помещались в оперативную память. Методы поиска на протяжении истории были устроены одинаковы и отличались только признаками.

- Гистограмма цветов (1995)
- Локальные особенности и мешок слов (2003)
- Гистограмма градиентов (2003)
- Нейросетевые признаки (2013)

Для поиска похожих изображений используется многомасштабный вариант дескриптора SIFT, который называется GIST. Он устроен точно так же, как SIFT, но вместо того, чтобы находить градиенты на одном масштабе, мы находим их на

нескольких масштабах, например, на 5. Для каждого масштаба строим свой дескриптор SIFT и в итоге получаем конкатенацию нескольких дескрпиторов, которые получены на нескольких масштабах.

GIST хорошо подходит для поиска визуально похожих изображений, полудубликатов. Если коллекция достаточно большая, то отсортировав ее по близости по дескриптору GIST, мы получаем визуально похожие изображения. Однако этот дескриптор слишком массивный, так как требует 2 кбайта на одно изображение при float (8 ориентаций x 4 масштаба x 16 ячеек =512 параметров). Стандартным способом ускорения является квантование за счет уменьшения точности (vector quantization), то есть мы отображаем все вектора на меньшее дискретное число. Квантовать мы можем с помощью К-средних. Один из способов создания структуры для быстрого перебора – это *инвертированный индекс.* Мы квантуем все вектора, например, с помощью кластеризации К-средними и все дескрипторы, которые попали в свой кластер, мы записываем в виде списка, то есть получаем список кластеров, в каждом из которых записано изображение. Поскольку мы сначала пишем номер кластера, а потом изображение, то это называется «инвертированный индекс». Теперь в качестве приближенного ответа мы можем выдавать все изображения, которые попали в тот же кластер, что и наш запрос. Поскольку внутри кластера признаки могут быть близки или далеки, мы можем выдать упорядоченный ответ, сделав ранжирование результатов. Например, мы можем сделать ранжирование результатов, сравнивание дескрипторы поблизости целиком.

Однако при использовании квантования К-средним, мы сталкиваемся со следующими проблемами:

- Чем крупнее кластеры, тем меньше точность аппроксимации
- Для повышения точности требуется существенно увеличить число кластеров К
- Квантование до кодов 64 бита требует подсчета и хранения 2^{64} центроидов, что невозможно
- Сложность одного этапа O(NK) – медленно на больших выборках и при больших К
- Поэтому требуется другие способы квантования. Мы можем построить такие бинарные коды h(x) (подписи), что для близких в L2 дескрипторов x (евклидово расстояние), коды h(x) будут близки по расстоянию Хэмминга. Такой способ называется *семантическим хешированием (*рис. 31). Функция отображения изображений в такие бинарные коды назвали семантической хэш-функцией.

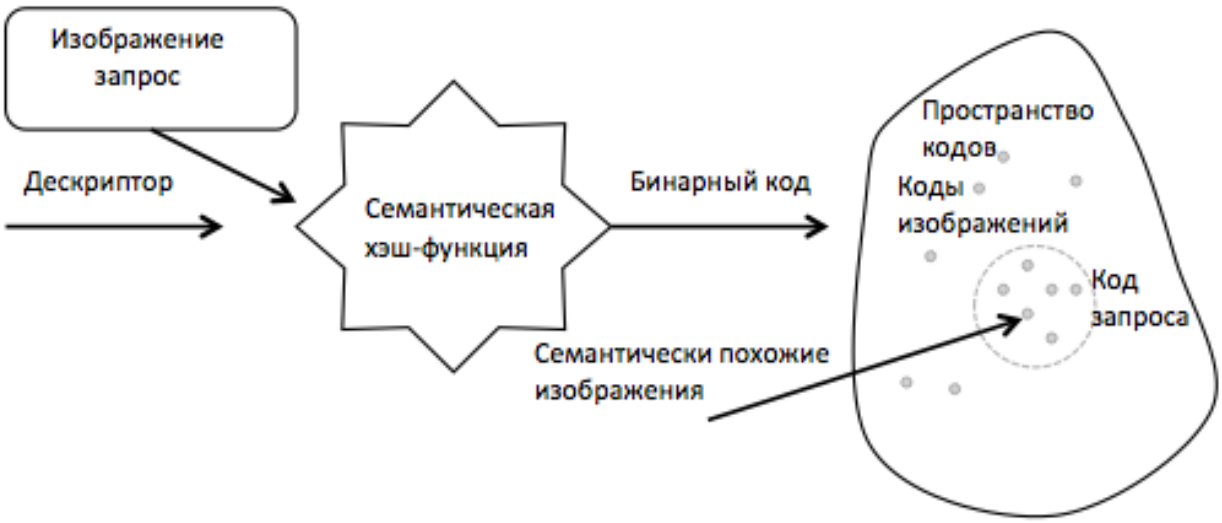


рис.31

Один из базовых алгоритмов семантического хэширования - метод локально чувствительного хэширования (Locality Sensitive Hashing) (рис.32):

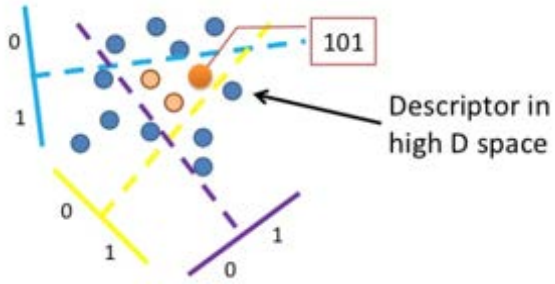


рис.32

- Возьмем случайную проекцию данных на прямую
- Выберем порог близко к медиане проекции
- Пометим проекции 0 или 1 в зависимости от расположение относительно прямой (1 бит подписи)
- С увеличением числа бит подпись приближает L2- метрику в исходных дескрипторах.

Плюсы	**Минусы**
Приближенный способ вычисления ближайшего соседа	Приближение L2 лишь асимптотическое
Быстрое квантование (быстро вычисляем бинарный код)	На практике может потребоваться слишком много бит для подписи

Выводы: использовать этот метод как замену К-средних нельзя, можно использовать в дополнение.

Идея: закодируем положение нашей точки (дескриптора) внутри ячейки диаграммы Вороного (т.е. вектор x-q(x)) с помощью семантического хэширования, построив код b(x). Получим аппроксимацию x как vec(q(x)) + vec(b(x)). Для каждой ячейки можем построить с помощью LSH свое отображение в коды.

Алгоритм GISTIS (GIST Indexing Structure) (рис.33)**:**

- Строим GIST для каждого изображения
- Кластреризуем все дескрипторы с помощью К-среднего на К=20000 кластеров
- Применяем LSH/PQ к каждому кластеру и для всех векторов в кластере считаем бинарную подпись длиной 512 бит
- Идентификатор картинки (64 бит) и бинарная подпись (512 бит) хранятся в RAM
- Остальная информация (GIST и т.д.) хранится на диске и считывается по необходимости

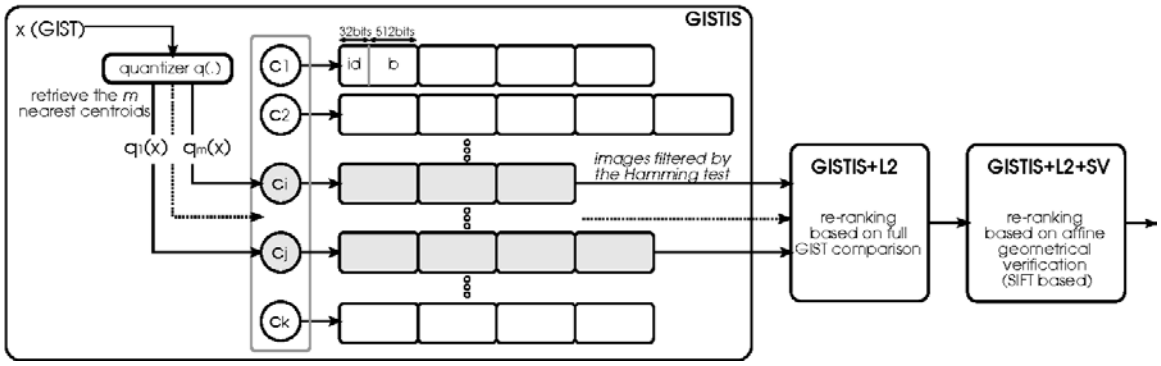


рис.33

Схема метода:

Поиск (фильтрацию) проводим в несколько стадий:

- Выбираем 200 ближайших кластеров (при идеальном распределении 1% картинок, на практике 2.3% из-за неравномерности кластеров), получаем q1(x),….,q200(x)
- Считаем бинарные коды b1(x),…,b200(x)
- Фильтрация по бинарным кодам с порогам h=220 (отбрасываем 94%)
- Итого возвращаем 0.13% картинок, отсортированных по расстоянию Хэмминга

Результаты:

Лучшие изображения мы можем дополнительно отсортировать по всему GIST (L2-метрика)

Поскольку GIST приходится читать с диска, то сортируем около 200 топовых ответов.

	Байт на изображение в RAM	**Время на построение дескриптора**	**Время поиска в базе из 110M изображений**
GIST	2840	35 мс	1.26 мс
GISTIS	68	36 мс	2 мс
GISTIS + L2	68	36 мс	6/192 мс

Если мы хотим найти изображение одного и того же объекта, дескриптор GIST плохо справится с этой задачей, лучше воспользоваться сопоставлением по особым точкам дескриптора SIFT. Но находить пары особых точек долго, поэтому нам поможет “мешок слов”:

- Построим «словарь» ключевых точек (К-среднее для выборки дескрипторов)
- Для поиска пар соответствующих точек нужно оценивать их индексы по словарю
- Количество пар соответствующих точек (putative correspondences) можно вычислить как пересечение гистограмм для двух «мешков слов»
- Получаем аппроксимацию числа соответствующих точек
- Вектор-признак изображения = мешок слов (BoW)

Построим инвертированный индекс для мешка слов /квантовых особенностей

- Таблица (слова)х(изображения)
- Список слов в словаре (терминов)
- Для каждого слова храним список изображений, в котором слово встречается

Если словарь большой, то большая часть элементов мешка слова нулевая. Храним мы только ненулевые значения, самые «употребительные слова» ставим в начало списка для ускорения поиска.

Резюме поиска похожих изображений:

- Задача классификации с открытым набором классов сводится к построению индекса, в котором хранятся вектор-признаки изображений, и поиска ближайшего соседа
- Для построения вектор-признаков можем использовать те же методы построения признаков, что и для классификации
- Для ускорения поиска по большим коллекциям применяем «умное» квантирование.

Лекция 4. Введение в сверточные нейросети

Модель нейрона

Идея сделать математическую модель нейросети появилась одновременно с идеями о цифровом компьютере. Мозг человека – эталонный образец, с которого мы списываем все наши модели. Благодаря нейрофизиологиии стало известно, что мозг человека состоит из нейронной сети, в виде сложной системы из взаимосвязанных нейронов. Каждый нейрон (рис. 34) выполняет единичную обработку информации, и он устроен следующим образом: у нейрона есть тело с большим количеством отростков, большая часть из которых короткие. Есть один специфический длинный отросток. Нейрон получает сигнал от всех своих соседей и при определенных условиях он возбуждается и генерирует электрический импульс, который двигается вдоль длинного отростка под названием аксон и затем передается на другие нейроны.

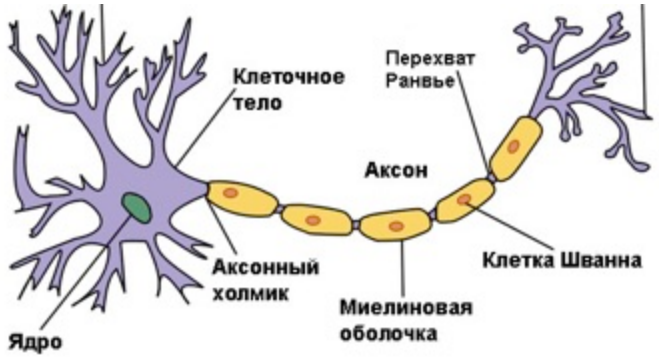


рис.34

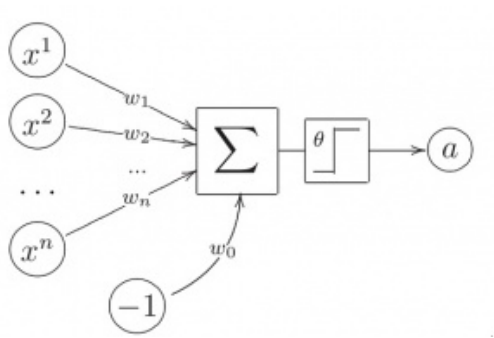


рис.35

Простейшая модель нейрона (модель МакКаллока-Питтса) (рис.35) была предложена в 1943 году, этот вариант используется и сейчас в аппарате под названием искусственные нейронные сети.

Мы знаем, что нейрон получает сигналы от соседей, пронумеруем соседей. Предположим, что они передает нейрону некоторые константные сигналы, важность этого сигнала моделируется весом, то есть некоторым коэффициентом, на который мы домножаем этот сигнал. Мы суммируем все взвешенные сигналы от соседей и добавляем к ним некоторый вес под названием *сдвиг (*bios). Если суммарный сигнал больше 0, то нейрон возбуждается и выдает сигнал 1, если суммарный сигнал меньше 0, то нейрон не возбуждается и передает 0.

Функция порога, которая решает, какой сигнал передать в зависимости от суммы входов, называется *функцией активации*. Функция активации должна быть нелинейной, простейшая функция активации – порог активации.

$$a(x.w) = \sigma(\langle w.x \rangle) = \sigma(\sum_{j=1}^{n} w_j f_j(x) - w_0)$$

$\sigma(z)$ – функция активации (например, sign)

w_j - весовые коэффициенты синаптических связей

w_0 - порог активации

$w, x \in \mathbb{R}^{n+1}$, если ввести константный признак $f_0(x) \equiv -1$

Если в качестве нелинейной функции мы используем порог, тогда нейрон - по сути, простой линейный бинарный классификатор. То есть с помощью нейрона мы создаем гиперплоскость, которая отделяет положительные и отрицательные сигналы. Под «обучением» нейрона понимают настройку весов. Настраивать веса линейного классификатора можно с помощью градиентного спуска или SVM.

Основной подход – метод градиентного спуска (рис.36):

- Есть функция стоимости от параметров w, нужно найти параметры, при которых она достигает минимума
- Считаем градиент функции в точке начального приближения и сдвигаем w в сторону уменьшения стоимости
- Повторяем до сходимости
- Попадаем в локальный минимум, которой также может быть глобальным

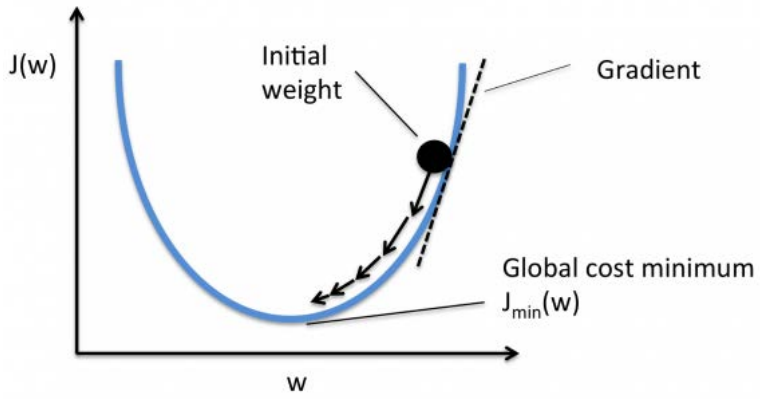


рис.36

То есть качество напрямую зависит от начального приближения. Применение градиентного метода обучения нейронов сводится к минимизации аппроксимированного эмпирического риска:

$$Q(w; X^{\ell}) = \sum_{i=1}^{\ell} \mathcal{L}(\langle w, x_i \rangle y_i) \to min$$

Численная минимизация методом градиентного спуска:

$w^{(0)} :=$ начальное приближение

$w^{(t+1)} := w^{(t)} - \eta \cdot \nabla Q(w^{(t)})\ ;\ \nabla Q(w) = (\frac{\partial Q(w)}{\partial w_j})^{n}_{j=0}$

η - градиентный шаг, называемый также темпом обучения

$w^{(t+1)} := w^{(t)} - \eta \sum_{i=1}^{\ell} \mathcal{L}'(\langle w^{(t)}. x_i \rangle y_i) x_i y_i$

Идея ускорения сходимости:

брать $(x_i\ .\ y_i)$ по одному и сразу обновлять вектор веса

То есть вместо того, чтобы посчитать градиент по всей обучающей выборке, мы будем брать один пример, оценивать ошибку на нем и менять параметры таким образом, чтобы уменьшить ошибку на этом примере. Такой метод называется *стохастический градиентный спуск* (stochatic gradient descend (SGD)). Суть в следующем: мы инициализируем веса, начальную ошибку и выбираем объекты из выборки случайным образом, вычисляем на нем ошибку и градиент. Делаем один шаг градиентного обучения, уточняем ошибку на всех примерах и повторяем до тех пор, пока не сойдемся.

Вход: выборка X^{ℓ}, темп обучения h, темп забывания λ

Выход: вектор весов w

1. инициализировать веса $w_j, j = 0 \dots n$
2. инициализировать оценку функционала $\bar{Q} := \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}_i(w)$
3. **повторять**
   - выбрать объект x_i из X^{ℓ} случайным образом
   - вычислить потерю: $\varepsilon_i := \mathcal{L}_i(w)$
   - сделать градиентный шаг: $w := w - h\nabla\mathcal{L}_i(w)$
   - оценить функционал: $\bar{Q} := (1 - \lambda)\bar{Q} + \lambda\varepsilon_i$
4. **пока** значение $\bar{Q}$ и / или веса w не сойдутся

Логические операции легко реализуются с помощью градиентного спуска. Мы можем представить логическую операцию как квадрат из четырех точек, которые обозначают 4 ответа, и нам нужно подобрать такую гиперплоскость, разделяющую прямую, которая отделит положительные ответы от отрицательных. Для операции И 1 положительный ответ, для операции ИЛИ – 3 положительных ответа. Однако если мы возьмем более сложную операцию, например, ИСКЛЮЧАЮЩЕЕ ИЛИ, то мы увидим, что гиперплоскостью невозможно отделить положительные ответы от отрицательных.

Здесь есть 2 решения: 1) добавить нелинейный признак, например, если мы на вход будем давать не 2, а 3 значения: x_1, x_2, x_1x_2, то тогда существует такая гиперплоскость, которая отличает положительные ответы от отрицательных или 2) можем воспользоваться нейросетью, состоящей из 3 нейронов.

Утверждение. Любая булева функция представима в виде ДНФ, следовательно, и в виде двухслойной сети.

Решение тринадцатой проблемы Гильберта:

Теорема Колмогорова, 1957

Любая непрерывная функция n аргументов на единичном кубе $[0,1]^n$ представима в виде суперпозиции непрерывных функций одного аргумента и операции сложения:

$f(x^1.x^2 \dots x^n) = \sum_{k=1}^{2n+1} h_k\left(\sum_{i=1}^{n} \varphi_{ik}(x^i)\right)$, где h_k, φ_{ik} – непрерывные функции и φ_{ik} не зависят от f

Результатом этой теоремы с точки зрений нейронных сетей является доказательство того, что с помощью линейных операция и одной нелинейной функции активации можно вычислить любую непрерывную функцию с любой желаемой точностью. Однако из доказательства не следует, как должна быть устроена сеть, сколько в ней должно быть нейронов и с какими весами.

Задание и обучение нейросети

Определение. **Архитектура нейросети** – взвешенный ориентированный граф, в котором вершины- нейроны, ребра – связи (рис.37)

Архитектуру обычно задает разработчик на основании опыта и “лучших примеров” , только недавно архитектуру начали “обучать”, то есть архитектура создается автоматическим подбором.

Определение. **Веса нейросети** – совокупность весов всех ребер (веса каждого нейрона)

Настройка весов – “обучение” нейросети, для чего предложено несколько подходов

Большинство сетей, которые мы будем рассматривать относятся к многослойным feed-forward нейросетям. Идея многослойных однонаправленных сетей, следующая: передача сигнала идет в одном направлении от входа к выходу, все нейроны можно разделить на группы по числу предшествующих нейронов на пути сигнала, все нейроны, расположенные за определенное число шагов от входа, это нейроны, расположенные в одном слое нейросети. Выходы нейросети зависят от поставленной задачи, например, если мы решаем задачу классификацию изображений на 1000 классов, у нейросети будет 1000 выходов, которые будут давать вероятность классов.

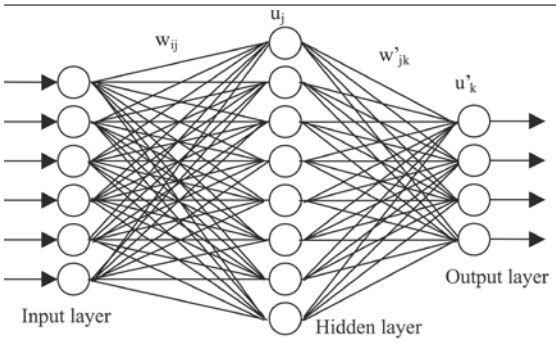


рис.37

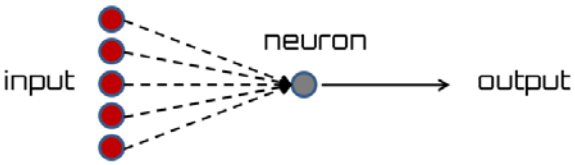


рис.38

Простейшая нейронная сеть – *линейный перцептрон (рис.38)*. Это сеть с входом x и 1 слоем, выход которого является выходом всей нейросети. Такая сеть реализует функцию:

$NN_{perceptron}(x) = xW + b, x \in \mathbb{R}^{d_{in}}, W \in \mathbb{R}^{d_{in}xd_{out}}, b \in \mathbb{R}^{d_{out}}$

Вход и выход мы видим, поэтому это видимые слои. Добавив дополнительные слои, получим *многослойный персептрон (рис.39)*. Промежуточные слои называются *скрытыми.* Если каждый нейрон слоя связан с нейроном предыдущего слоя, то такой слой называется полносвязным (fully connected). Нейроны скрытых слоев обычно имеют нелинейную функцию активации.

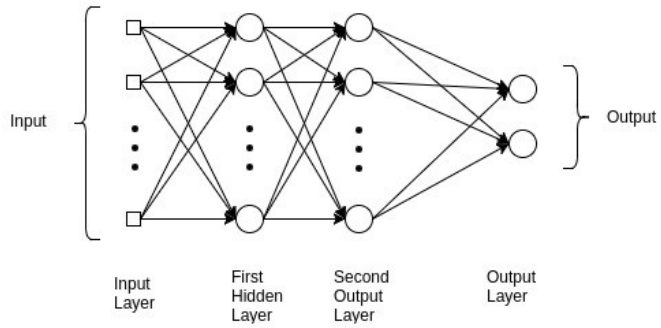


рис.39

В первых нейросетях использовались непрерывные функции активации, такие как

- $tahn(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- $sigm(x) = \frac{1}{1+e^{-x}}$ – преобразуют все значения в интервал [0,1]
- ReLU(x) = max(0,x)

Запишем послойные преобразования функции перспетрона, которые впоследствии подставляются в нелинейную функцию активации:

$NN_{MLP2}(x) = y$

$h^1 = g^1(xW^1 + b^1)$

$h^2 = g^2(xW^2 + b^2)$

$y = h^2W^3$

В итоге, скомбинировав все преобразования, можем выписать одну функцию, которая реализуется с помощью данной нейросети:

$NN_{MLP2}(x) = (g^2(g^1(xW^1 + b^1)W^2 + b^2))W^3$

Если нейросеть очень глубокая, то функцию выписать очень сложно. Этот аппарат аппроксимации очень мощный и широко распространенный.

Если мы решаем задачу многоклассовой классификации, то мы хотим интерпретировать выходы y_i как вероятности меток классов. Надо, чтобы $y_i \in [0,1], \sum_{i=1}^{N} y_i = 1$. Это можно сделать с помощью softmax преобразования:

$x = x_1, \dots, x_k$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

Сжимаем произвольный вектор длины N в вектор со значением (0,1) и суммой = 1. Эта функция взята из многомерной логистической регрессии. Softmax преобразование отображает любой вектор в вектор, который мы интерпритируем как вероятность.

С помощью персптрона (даже с 1 скрытым слоем) мы можем задавать произвольные функции. Для того, чтобы обучить нейросеть, нужно выбрать *функцию потерь*, то есть целевой функционал, который мы будем оптимизировать, и метод ее минимизации. Поскольку мы решаем задачу классификации, мы можем выбрать любую функцию, используемую в классификаторах, например, кросс-энтропию (cross-entropy). Эта функция измеряет близость истинного и оцененного распределения меток (обычно после softmax).

$L(\hat{y}, y) = -\sum_k^K y^{(k)} \log \hat{y}^{(k)}$- сумма по всем. К меткам (метки задаются в виде вероятности) произведение истинного ответа на логарифм измеренного ответа. Взяв эту функцию и применив метод стохастического градиентного спуска, мы можем обучить все параметры нейросети. Как посчитать градиент для всех параметров нейросети?

Нейросеть – это не более, чем большая дифференцируемая функция от своих входов. Соответственно, мы можем применить правила дифференцирования сложных функций, чтобы посчитать производные по каждому параметру. Этот метод получил название «обратное распространение ошибки».

Суть метода обратного распространения ошибки заключается в том, что мы можем рассчитать градиент параметра как взвешенное произведение ошибок всех связанных с ним последующих нейронов на производную функции активации нейрона. Для этого сначала посчитаем градиент нейронов выходного слоя, затем предыдущего и т.д. Для всей сети мы можем это сделать за один обратный проход, как бы запустив сеть «задом наперед».

Главная проблема стохастического спуска заключается в т ом, что поскольку мы берем один пример, в зависимости от качества примера градиент может очень сильно меняться. Компромиссом между обычным градиентным спуском и стохастическим является метод под названием minibatch.

Идея метода minibatch: формируем небольшую группу примеров под названием minibatch, вычисляем градиенты по каждому примеру, усредняем все градиенты между собой и делаем один шаг градиентного спуска с этим усредненным градиентом. Таким образом градиент получается менее шумным, и процедура сходимости работает быстрее. Вычисления более эффективны, так ка мы выбираем размер minibatch так, чтобы он поместился в оперативную память.

Перейдем к задаче инициализации градиентного спуска. Чтобы задать начально приближение всех весов, чаще всего используются инициализации Xavier (2010) и He (2015). В обоих вариантах веса задаются случайным образом, но используем разные распределения и при этом учитываем число входов и выходов.

Инициализация Xavier

$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right]$, где U[a,b] – равномерное распределение в интервале

Инициализация He

Нормальное распределение с матожиданием 0 и стандартным отклонением $\sqrt{\frac{2}{d_{in}}}$, d_{in} число входов.

Заметки по процедуре обучения:

- Темп обучения выбирают таким образом, чтобы веса не слишком колебались или сходились
- С течением времени темп обучения уменьшают, чтобы обеспечить сходимость весов
- «Эпоха» - этап обучения на всех примерах из обучающей выборки
- Обычно обучение состоит из нескольких эпох, между которыми меняют параметры обучения
- Обычно экспоненциально снижают скорость обучения
- В minibatch должны присутствовать и положительные, и отрицательные примеры, чтобы он хорошо представлял обучающую выборку

- Процедура составления minibatch- процедура *составления расписания*

Есть ряд усовершенствований стохастического градиентного спуска (SGD). Например, чаще всего используют SGD + momentum (Поляк, 1964). Суть момента: сохраняя градиенты, полученные до этого, мы усредняем текущий градиент с предыдущими и двигаемся на этот усредненный градиент. Для высчитывания момента используется *момент Нестерова* или адаптивный выбор обучения для каждого минибатча, такие как AdaGrad, AdaDelta, Adam, Ranger.

Проблемы обучения нейросетей:

1. «Затухание» (saturation) градиентов, если попадаем на область низкого градиента функции активации. (ReLU не страдает от этого в отличие от Sigmoid)
2. «Мертвые» (dead) нейроны, которые получают только отрицательные или нулевые выходы
3. Исчезающие или взрывные (vanishing or exploding gradients) градиенты в глубоких сетях

Метод обратного распространения ошибки оказался очень эффективным. Например, Rowley face detector (1998) (рис.40), представляющий из себя маленькую нейросеть, был лучшим до Viola-Jones.

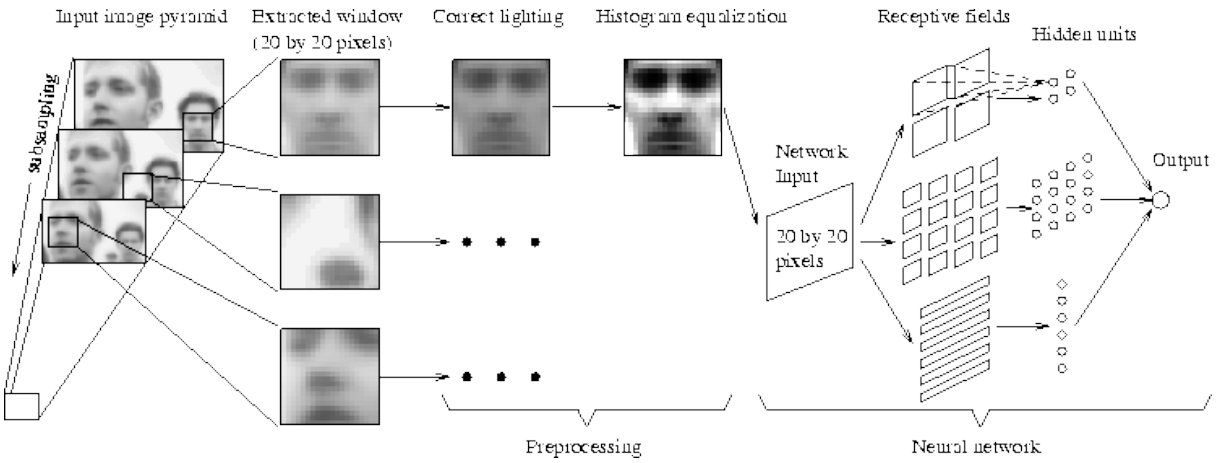


рис.40

Нейросеть для обработки изображений

Для обработки изображений необходимо построить нейросеть, которая на вход принимает изображение, заданное в виде трехмерной матрицы, и на выходе зачастую также выдает трехмерную матрицу. При конструировании этой ети мы хотим учесть знания об обработке изображения в мозге человека и эвристические методы распознавания. Обработка краев и градиентов хорошо согласуется с тем, как мозг человека и мозг животных обрабатывает визуальную информацию. В мозге человека можно выделить область под названием *первичная визуальная кора,* в которой

происходит первичная обработка визуальной информации. В визуальной коре есть клетки 2 типов: простые (simple) (рис.41) и сложные (complex)(рис.42). Простые клетки специфичны, они чувствительны к градиентам или контрастным полоскам, которые расположены в строго определенной области. Сложные клетки **инвариантны** к сдвигам в небольшой окрестности.

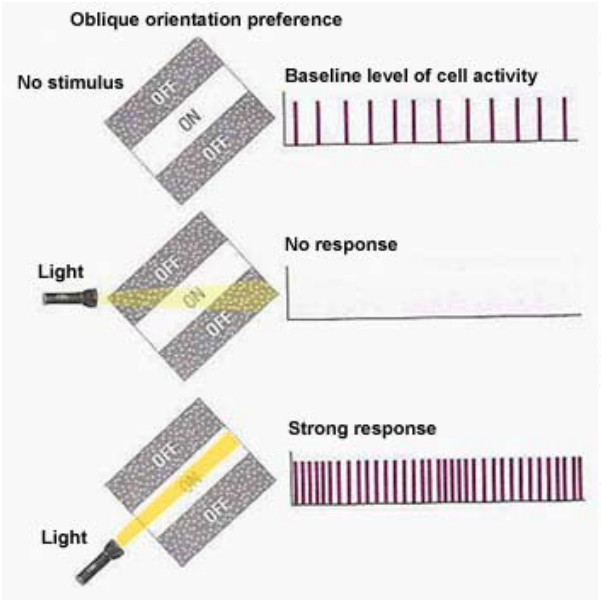


рис.41

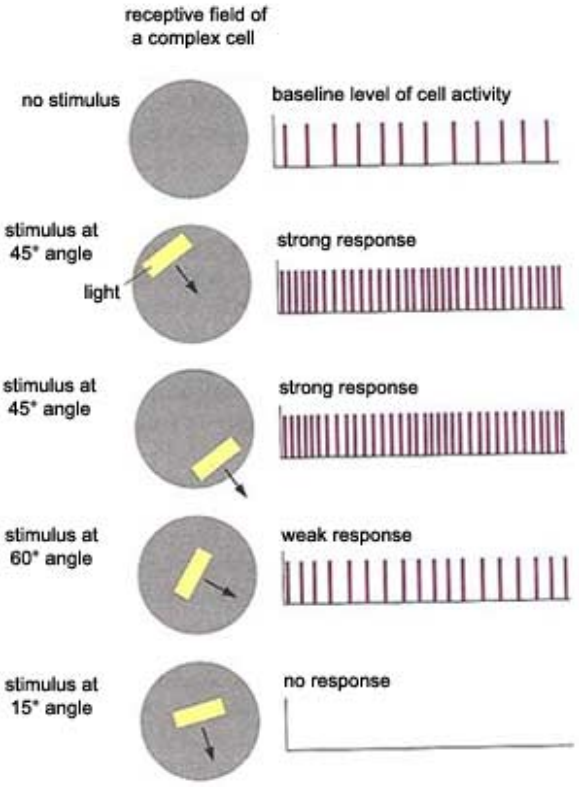


рис.42

Мы можем анализировать распределение градиентов в изображении с помощью сверток:

- Выберем набор (банк) фильтров, каждый из которых чувствителен к краю определённой ориентации и размера
- Каждый пиксель изображения после обработки банком фильтров дает вектор признаков
- Этот вектор признаков эффективно описывает локальную текстуру окрестности пикселя

Эксперименты показывают, что большинство сверток простых клеток, которые работают в мозге человека, мы можем представить одной параметрической функцией, которая называется *фильтр Габора.* Фильтр Габора – это фактически синусоида, которая домножена на Гауссиану.

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

$$x' = x\cos(\theta) + y\sin(\theta)$$

$$y' = -x\sin(\theta) + y\cos(\theta)$$

θ – ориентация

λ - длина волны

σ - сигма гауссиана

γ - соотношение размеров (aspect ratio), “эллиптичность фильтра”

ψ - сдвиг фазы

Чтобы реализовать сложную клетку мы можем воспользоваться в качестве фильтра оператором MAX. Инвариантность можно обеспечить за счет применения оператора MAX на выходах набора «простых» клеток, чувствительных к краю одной ориентации, но в разных точках одной области. Мы можем реализовать в виде нейронов свертку и фильтр максимума.

Операцию линейной фильтрации (свертки) для одного пикселя можно реализовать одним нейроном. Один нейрон свяжем не со всем изображением, а с некоторой окрестностью. Свертку изображения целиком можно реализовать как «слой» нейронов, веса которых одинаковы. Обычно под «сверточным слоем» понимают набор сверток, применяемых к одному входу («банк фильтров»). Результаты сверток объединяют в один выход (трехмерную матрицу) $X \in \mathbb{R}^{m \times n \times k}$, где k – число сверток в слое.

Фильтр MAX Poolingустроен просто: мы реализуем специальный слой, который выполняет операцию max-pooling. В этом слое мы применяем операцию max по выбранным областям (по сетке). На вход получаем трехмерную матрицу и выдаем трехмерную матрицу меньшего разрешения. При расчете градиентов ошибки пробрасываем градиент в тот нейрон предыдущего слоя, который дал max, а во все остальные подаем 0. Вместо операции максимуму мы можем применять другие операции, например, суммирование.

Впервые архитектуру нейросети, которая реализует операцию обработки изображений, похожих на мозг человека, предложили в 1980 году, и она получила название *неокогнитрон (*рис.43) (neocognitron). Она состояла из комбинации сверток и применения фильтров max-pooling. На самом последнем этапе операция max-pooling проводилась по всему изображению. На верхнем уровне обеспечивается инвариантность по положению по всему изображению. Неокогнитрон никак не обучался, все параметры задавались вручную.

Неокогнитрон + обратное распространение ошибки = сверточная сеть

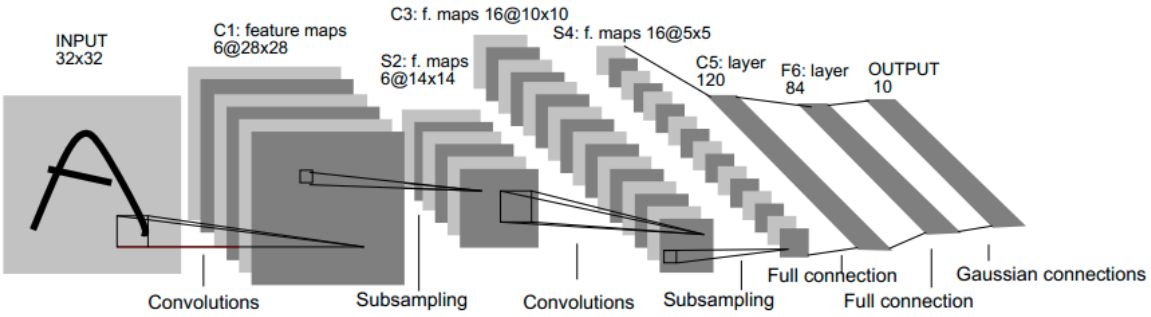


рис.43

Поскольку для сверточного слоя нужно задавать параметры только всех сверток, что число параметров заметно меньше общего числа весов слоя. Сверточные сети (convolutional neural network), состоящие только из сверточных слоев, называются полносверточными

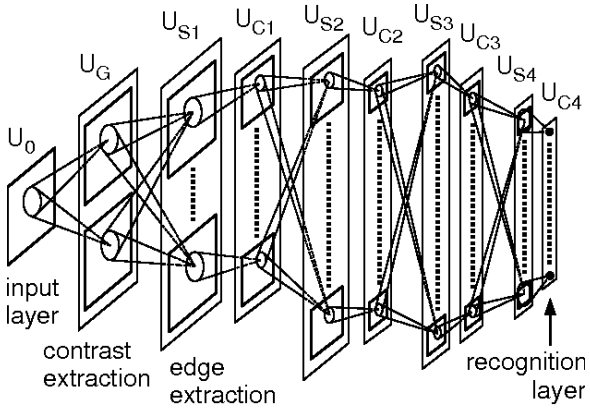


рис.44

Эталонный пример сверточной нейросети Яна Ликуна (рис.44).

На вход сеть получает картинки 32x32 пиксела, далее получаем 6 карт признаков 28x28 пикселов. Исходя из того, что после первого сверточного слоя у нас получились картинки 28x28 пикселов, мы можем сказать о размерах сверток: 6 сверток 5x5. После операции max-poolingполучаем картинки 14x14, следовательно, фильтр max-pooling применялся по областям 2x2. Далее у нас получилось 16 карт размером 10x10 пикселов. Глубина тензора на выходе сверточного слоя равна числу сверток в сверточном слое. Третье измерение свертки равно «толщине» входного тензора.

5x5x1 – размеры филтров сверток на первом слое

5x5x6 – на втором слое, также на втором слое 16 сверток, следовательно 150x16+16 =2416 параметров. Весов= (примерно) параметров x 10x 10 =240000

У сверточных нейросетей есть важное понятие, используемое для анализа, - рецептивное поле (receptive field).

Определение. **Receptive field нейрона** – область изображения, от которой зависит выход этого нейрона.

Размер и положение поля определяется глубиной нейрона, размерами сверток, размерами областей пулинга. Чем глубже мы уходим, тем больше размер рецептивной области.

По умолчанию свертка применяется к каждому пикселу изображения, но мы можем применять свертку не к каждому пикселу, а с некоторым *шагом*. При этом выход получается меньше, например, если мы делаем шаг n=2, то в результате картинка уменьшается в 2 раза.

Существуют *нормализующие слои*, которые выполняют некоторую нормализацию, например, «перевзвешивают» значение пикселов в некоторой окрестности. у этих слоев в основном нет параметров, которые обучают. Например, локальный нормализующий слой делает следующее: мы считаем локальное среднее значение откликов по окрестности, затем делим текущее значение на среднее значение и сдвигаем его. Получается повышение локального контраста.

Почти все эвристические признаки мы можем представить в виде композиции этих методов и сделать нейросеть, которая в явном виде реализует эти признаки. Например, **дескриптор SIFT:**

- находим направление градиентов, для этого достаточно сделать свертку фильтрами Габора (смотрим отклик направления градиента)
- суммируем отклики фильтров по ячейкам, таким образом считаем гистограммы распределения пикселов по ориентации
- проводим нормализацию вектора до единичной длины, получаем вектор-признак, реализующий SIFT

Добавив набор слоев нейросети к полученном дескриптору SIFT, получим операцию, похожую на **мешок слов:**

- применяем к дескрипторам свертку со словарем
- применяем операцию максимума, которая будет показывать, на какое слова из словаря больше всего был похож данный дескриптор (проводить квантование)
- суммируем отклики по окрестности, считаем сколько точек похожи на какие слова из словарей (мешок слов)

Переобучение

Переобучение – явление, когда нейросеть быстро обучается, а потом у нее резко возрастает ошибка на тесте. Из-за большого количества параметров нейросеть очень быстро выучивала всю обучающую выборку, и долгое время обучающих выборок было недостаточно.

Сейчас появилось большое количество коллекций, на которых переобучение уже не так заметно, хотя все равно текущих коллекций для обучения нейросетей недостаточно.

Один из самых популярных способов борьбы с переобучением сетей - *размножение данных* (data augmentation). Случайным образом вырезаем фрагменты из изображений и их отражения и цветовые искажения, тем самым увеличивая обучающую коллекцию. Также возможны другие изменения: сдвиги, отображения, повороты, изменения масштаба.

Резюме:

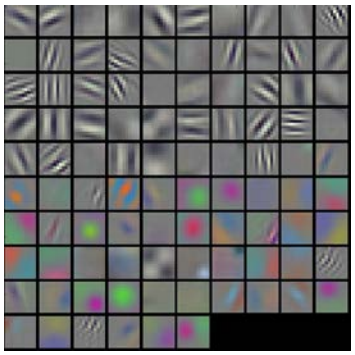
- Концептуально нейросети остались такими же, как в 1990х, но было предложено множество изменений, которые в совокупности позволили сети эффективно обучать
- Есть целый ряд библиотек и уже обученных моделей
- Возможность взять обученную модель и настроить ее на другие данные, позволяет расширять круг решаемых задач на те, где данных не так много
- Нейросетевые модели стали применяться очень широко

Лекция 5. Развитие нейросетевых методов

Визуализация работы нейросети

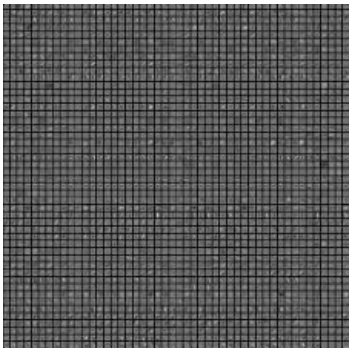
Как можно проинтерпретировать поведение нейросети и зачем вообще это нужно? Есть много приложений, которые явно этого требуют, например, медицинские приложения. Представьте, что перед вами стоит задача выявления какого-то заболевания по какому-то снимку (рентгеновскому итп), вы собираете обучающую выборку с разметкой, которую вам предоставили эксперты врачи. Вы обучаете нейросеть на этих снимках, и она ставит какой-то диагноз, причем ответ нейросети предоставляется в понятном виде для эксперта, который будет рассматривать. Но прежде, чем воспользоваться полученным диагнозом, врач должен понять, что именно находила нейросеть и не ошиблась ли она. Это один из ярких примеров, показывающий, зачем нужно интерпретировать работу нейросети.

Начнем с интерпретации фильтров первого сверточного слоя. Поскольку фильтры первого сверточного слоя, как правило, применяются к цветным изображениям, мы легко можем визуализировать эти свертки, у них будет 3 канала: у нас получатся такие цветные изображения (рис. 45)



Слой conv1

рис.45



Слой conv2

рис.46

В данном случае это свертки из нейросети AlexNet 11x11 пикселей. Изначально нейросеть обучалась на 2 видеокартах, свертки были распределены по 2 видеокартам, то есть одни свертки обучались на одной видеокарте, а другие – на другой. Суммарная визуализация представлена рис. 45. Условно эти свертки можно поделить на группы: первые свертки – свертки, похожие на градиентные фильтры, которые представлены как различные вертикальные, горизонтальные, наклонные линии; они ищут какие-то границы в изображении. Другие свертки, которые мы видим, - цветные свертки, которые анализируют локальную область изображения. Например, свертка с зеленым

пятном анализирует локальную область и смотрит, насколько там много зеленого цвета. Также есть свертки – комбинации цветного фильтра и градиентного, они представлены цветовым переходом.

Второй слой AlexNet (рис. 46) представлен 96 слоями с небольшими свертками 5x5 пикселей. То есть мы берем 96 слоев размером 5x5 пикселов и выстраиваем мозаику (рис. 46), и это только одна свертка из нейросети. Визуализировать свертки второго уровня в понятном для нас виде мы таким способом не можем, поэтому нам нужен другой способ.

Мы можем смотреть на изображение и собирать для каждой свёртки фрагменты изображения, которые вызвали у этой свертки наибольший отклик (рис. 47).

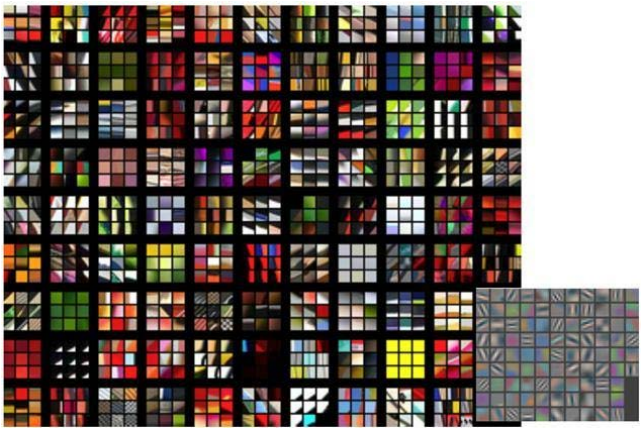


рис.47

На рис.47 визуализировано 9 фрагментов, на которых свертка первого уровня откликается сильнее всего. Здесь свертки откликаются на какие-то цветовые переходы или на равномерные цветовые области изображения. Мы получаем эти фрагменты следующим образом: нарезаем исходное изображение на фрагменты (в данном случае 11x11, потому что у нас свертка размером 11x11) и прогоняем локальные области через свертки, смотрим на мощность (модуль) отклика данного нейрона, если он достаточно большой, то мы включаем его в нашу подборку.

Фрагменты, которые активизируют наибольшим образом свертки второго слоя (рис.48) представлены уже более сложными паттернами.

рис.48

Чем выше слой, тем более абстрактные понятия находит нейрон (рис. 49, 50).

Слой 4

рис.49

Слой 5

рис.50

Фильтры слоя pool5

рис.51

Нейроны на этапе перед вытягиванием тензора в столбец (рис. 51) находят людей, цветы, надписи, различные здания, блики.

Также есть другой способ визуализации признаков: визуализация всех пикселей, которые наибольшим образом повлияли на отклик нейрона (рис. 52).

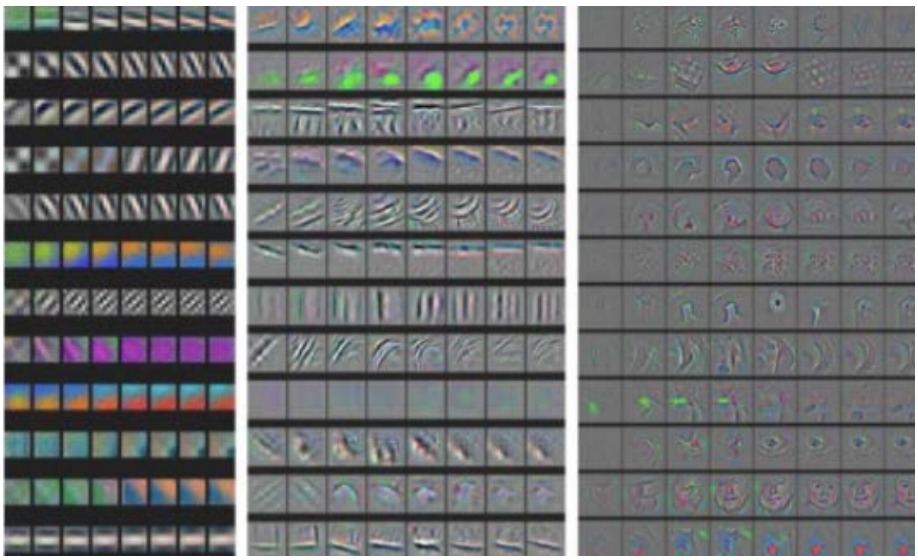


рис.52

Какие-то пиксели зануляются - они визуализированы серым цветом – , а те, которые внесли наибольший вклад, становятся яркими цветными (в зависимости от конкретной цветовой компоненты). Есть и другие способы визуализации.

После того, как мы прогоняем картинку через нейросеть, мы видим, что нейроны различных слове несут важную информацию, причем, чем выше слой, тем более абстрактная информация. Мы можем извлекать признаки с различных слоев нейросети и получать информацию различного порядка абстракции. Совокупность выходов слоя можно рассматривать как вектор-признак изображения.

Поскольку признаки, например, полносвязных слоев из AlexNet имеют размерность 4096, с которым не очень удобно работать. Поэтому мы проецируем их на двухмерное пространство, сохраняя расстояние между векторами. Далее визуализируем изображения, которые соответствуют этим признакам. Таким образом изображения разбиваются на некоторые кластеры.

UMAP визуализация

Если применить UMAP визуализацию к кластеризации цифр из мниста (рис. 52, 53), то можно увидеть, что в зависимости от класса изображения группируются, образуя явно отделимые классы. Это значит, что с помощью данной нейросети и вектор-признаков мы можем очень качественно классифицировать конкретный датасет.

рис.53

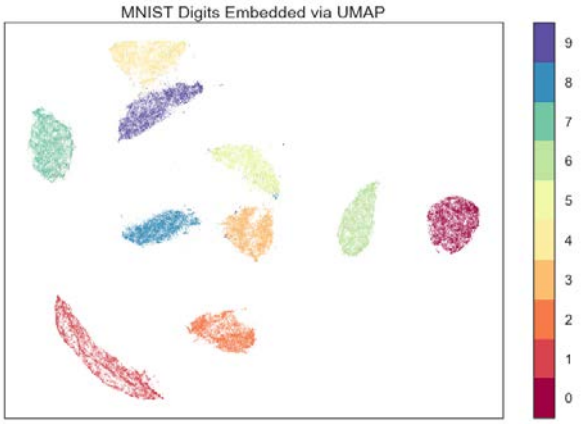


рис.54

Таким образом, нейросеть обучается отображать изображение в вектор-признаки, кодирующие семантическую «похожесть» изображений. Близкие и визуально похожие друг на друга объекты оказываются близки по нейросетевым вектор-признакам, в то время как вектор-признаки объектов разных классов далеки друг от друга.

Рассмотрим визуализацию сети AlexNet (рис. 55).

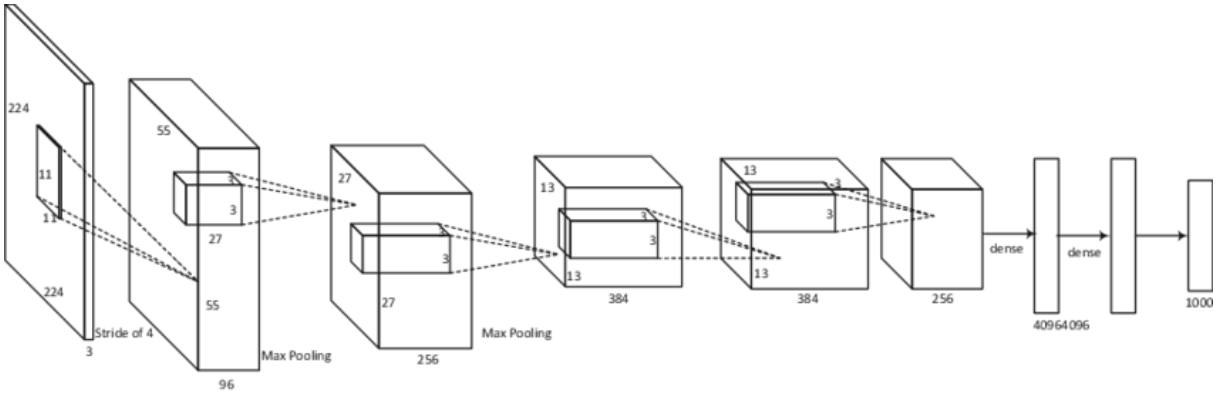


рис.55

Можно использовать выходы выбранного слоя нейросети как вектор-признак. То есть это сверточный слой перед полносвязными и два полносвязных слоя. Мы обучим всю нейросеть на ImageNet, извлечем признаки для каких-то других признаков. То есть

можно обучить нейросеть на одних данных (ImageNet) и применить ее на других для вычисления признаков, обучая поверх класификатор. Оказывается, что для того, чтобы получить высокое качество на каком-то датасете, мы можем ничего не делать с этим датасетом, но постоянно улучшать нейросети, которые мы обучаем на ImageNet. И если мы год за годом улучшаем нейросети на ImageNet, то мы автоматически получаем прирост на других задачах, для которых мы используем обученные веса. Таким образом, мы получаем универсальный фреймворк, который используется в очень многих задачах (классификация, детектирование, сегментация итп).

Классификация близких объектов

Мы можем извлекать признаки, обучать на них свн и получать хорошее качество классификации близких объектов, например, видов птиц. Но есть другой способ: мы можем инициализировать нейросеть уже обученными весами, а потом *дообучать* ее на данном наборе изображений.

Чтобы *дообучить* нейросеть, делаем следующее:

Возьмем сеть A, обученную на одной коллекции, например, ImageNet. У нее в конце стоит полносвязный слой с 1000 нейронов и с активацией softmax. Мы заменяем этот последний слой (классификатор), все кроме последнего слоя в научных статьях называется, как правило, backbone. К этой базовой сети добавляется новый полносвязный слой, и если птиц, например, 50 классов, то мы будем добавлять полносвязный слой на 50 классов. Затем мы размораживаем часть весов в этой нейросети (эти веса тоже обучаются), и начинаем дообучать нейросеть с помощью градиентного спуска. Таким образом, нейросеть получает хорошую инициализацию и обучается значительно лучше.

Плюсы дообучения:

- Вторая коллекция может быть недостаточного размера для обучения нейросети «с нуля» (from scratch), то есть мы можем обучить нейрсоеть с миллионами параметров на маленьких наборах данных
- Предобучение на большой и разнообразной коллекции может хорошо инициализировать новую сеть, и она дообучится быстрее.

Развитие базовых архитектур

1998 – LeNet

2012 – AlexNet

2014 – VGG

02.2015 – ResNet

2017 - DenseNet

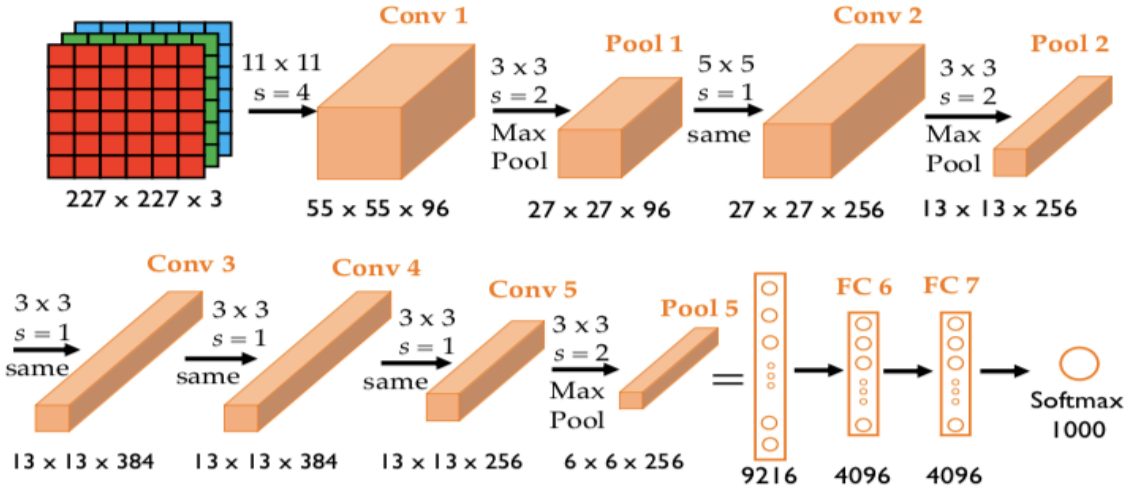


рис.56

AlexNet (рис. 56)

Первая свертка 11x11 с шагом 4, после которой идет maxpooling в 2 раза по каждой стороне, то есть изображение уменьшается по каждой стороне в 8 раз. Затем идет свертка 5x5 и снова pooling. Далее несколько сверток 3x3, pooling, пара полносвязных слоев большого размера 409, и наконец последний полносвязный слой, который осуществляет классификацию на 1000 классов.

*AlexNet это в несколько раз увеличенная архитектура LeNet

AlexNet работает на кропах из изображений размером 256x256 и 224x224. При предсказании делается 10 кропов, предсказания усредняются на этих кропах. Если изображение не 224x224, мы можем делать кроп или несколько кропов, но интересующий нас объект может не поместиться целиком ни в один из этих кропов, или в кроп может попасть слишком большая часть фона. Также мы можем брать интересующую нас область и масштабировать ее, но и это не очень хороший вариант, так как мы теряем пропорции объектов, которые несут очень важную информацию при классификации. Поэтому мы будем пользоваться другим способом, который называется *Spatial Pyramid Pooling (SPP)* (рис.57).

Spatial Pyramid Pooling (SPP)

Возьмем архитектуру AlexNet, ее сверточные слои и слои maxpooling мы можем применять к изображениям произвольных размеров. Эта базовая часть до полносвязных слоев может быть применена к изображению любого расширения, и в результате будет получен тензор произвольного размера. Для того, чтобы этот тензор подать полносвязному слою для классификации, его нужно сделать фиксированного размера. Для этого возьмем тензор и пространственно разобьем его на заранее зафиксированное количество ячеек. У нас будут ячейки 4x4, или 2x2, или вообще одна сплошная ячейка. В каждой из этих ячеек мы применим либо операцию максимума,

либо операцию усреднения, и для каждой ячейки мы получим одно число. Далее можем все это вытянуть в вектор-признак, вектора-признаки с различного количества ячеек сконкатенировать и получить представление фиксированного размера, которое дальше мы можем подать на вход полносвязному слою.

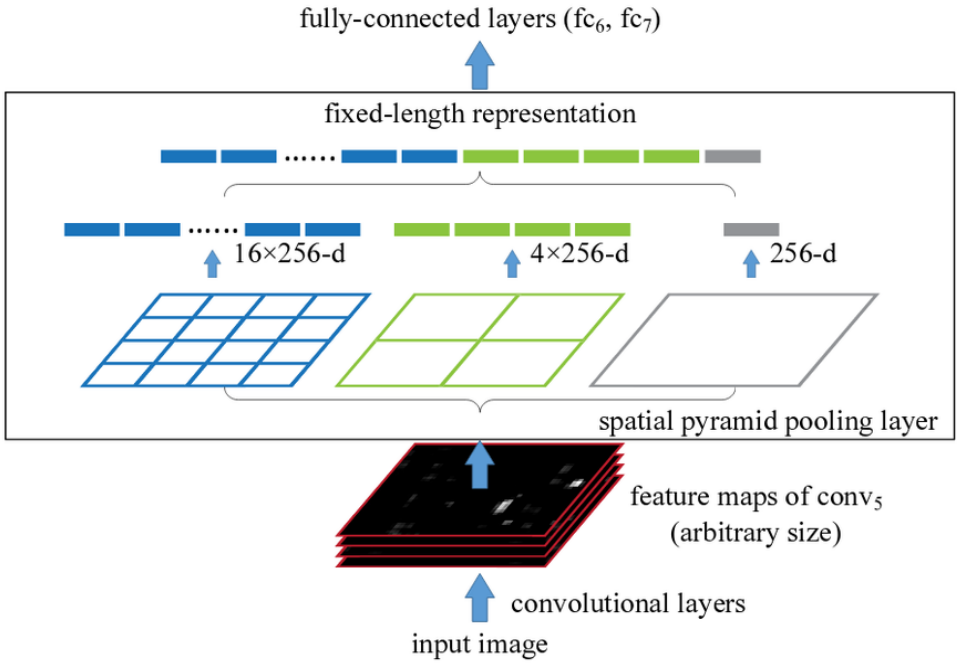


рис.57

Average Pooling

Мы могли не разбивать тензор на ячейки, а оставить одну карту, и сделать усреднение по всем картам. То есть если, например, на вход подается тензор и 256 слоев, то мы будем делать усреднение пространственно по каждой из карт, и у нас останется в каждом слое по одному значению. Мы можем вытянуть это в вектор длины 256, и затем подавать его на вход для классификации полносвязному слою. Таким образом можно получить качественный результат при дообучении.

Очень глубокие сети (VGG)

Эта нейросеть названа в честь лаборатории, в которой проходили исследования Visual Geometry Group в Оксфорде. Исследователи попробовали заменить большие (11x11, 5x5) свертки в AlexNet на более маленькие свертки, тем самым увеличить глубину, исходя из соображения: чем больше глубина нейросети, тем более качественно проводится классификация. Поэтому используются:

- только маленькие свертки 3x3

- шаг (stride) 1, чтобы не терять информацию
- ReLu активация
- нет нормализации
- уменьшение разрешения через maxpooling
- число фильтров x2 при уменьшении разрешения уменьшается в 2 раза*

*Далее некоторая эвристика, которая состоит в следующем: на вход есть тензор, к которому мы будем применять maxpooling; если maxpooling уменьшает пространственное разрешение в 2 раза, то перед этим количество слоев должно в 2 раза увеличиться, то есть у нас уменьшается количество информации пространственно, но при этом она переходит в слои.

Свертки 3x3 используются из следующих соображений (рис. 58):

Если мы посмотрим на область изображения, на которую «смотрит» свертка 5x5, то область изображения, на которую «смотрит» свертка 3x3 и за ней другая свертка 3x3, окажется такой же. То есть мы можем свертку 5x5 аппроксимировать с помощью 2 сверток 3x3, поместив между ними нелинейность, которая позволит функции, задающей нейросеть вести себя более сложным образом (она будет становиться все более и более нелинейной). Кроме того, если мы используем свертки 3x3 и с помощью них аппроксимируем свертки либо 5x5, либо 7x7 (добавив третью свертку 3x3), то мы используем меньше параметров, то есть мы можем позволить себе более глубокие сети с меньшим количеством параметров.

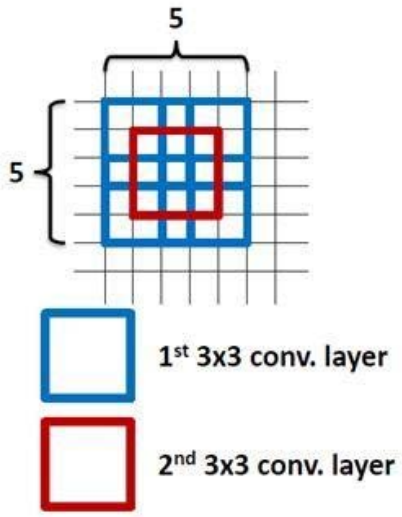


рис.58

В этом исследовании было предложено несколько архитектур VGG, самые интересные – это VGG-16 и VGG-19. Однако они оказались очень тяжелыми: если в AlexNet

использовалось порядка 60 млн параметров, то в VGG порядка 130-135 млн параметров.

Рассмотрим свертку 1x1: это свертка, у которой размер фильтра 1x1. Мы можем рассматривать такую свертку как локальный классификатор, то есть эта свертка «смотрит» на один пиксель, но при этом «смотрит» на всю глубину тензора. По совокупности значений в данном пикселе в различных слоях можно делать какой-то вывод. Вообще свертка 1x1 используется для эффективной регуляции количества слоев в нейросети, то есть если на выходе из свертки 1x1 количество слоев меньше, чем на входе, то тогда получается, что мы делаем локальное сжатие тензора (меньше слоев за счет каких-то линейных комбинаций предыдущих слоев); если количество слоев в выходном тензоре больше, чем во входном, значит, мы дополняем какими-то линейными комбинациями слоев, это может быть полезно в сложных моделях (например, графы), чтобы в разных слоях на выходе было одно и то же количество слоев в тензоре.

Архитектура Inception

Построим нейросеть из модулей более сложной структуры, чем просто набор сверточных слоев VGG. В этой архитектуре авторы отказались от линейной последовательности слоев и сделали такой базовый блок (рис. 59). То есть входной тензор прогоняется через свертку 1x1, 3x3, 5x5 и maxpooling размером 3x3. Таким образом мы одновременно считаем признаки разного порядка, так как свертки смотрят на различные окрестности разного размера (свертка 5x5 смотрит на окрестность побольше, свертка 1x1 рассматривает суженные признаки, а maxpooling – признаки другого порядка). Для того, чтобы количество параметров всех этих сверток было не очень большим, перед ними и после maxpooling стоят свертки 1x1, которые уменьшают количество слоев в данном тензоре (рис. 60). На вход подается тензор 256x256, свёрток 1x1 – 64, подаем тензор на вход первой свертке 1x1. Из 256 слоев делаем 64, подаем на вход сверткам 3x3, конкатенируем с предыдущим результатом и то же самое дальше. То есть вместо того, чтобы подавать на вход, как в базовой архитектуре (рис.59), тензор с 256 слоями и делать 64 свертки, мы сначала уменьшаем количество слоев, и свертки у нас уже размерностью не 256, а 64 (уменьшаем количество параметров в 4 раза). За счет сверток 1x1 мы можем легко комбинировать легковесные блоки и получать более эффективную архитектуру, чем VGG. Архитектура Inception выглядит следующим образом (рис. 61):

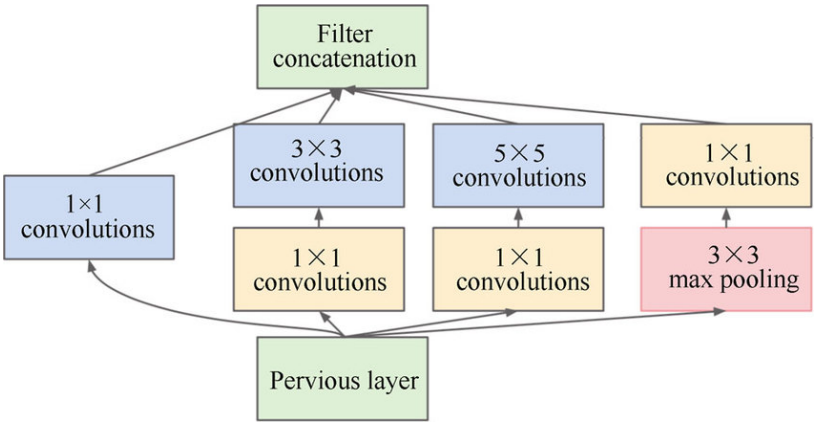


рис.59

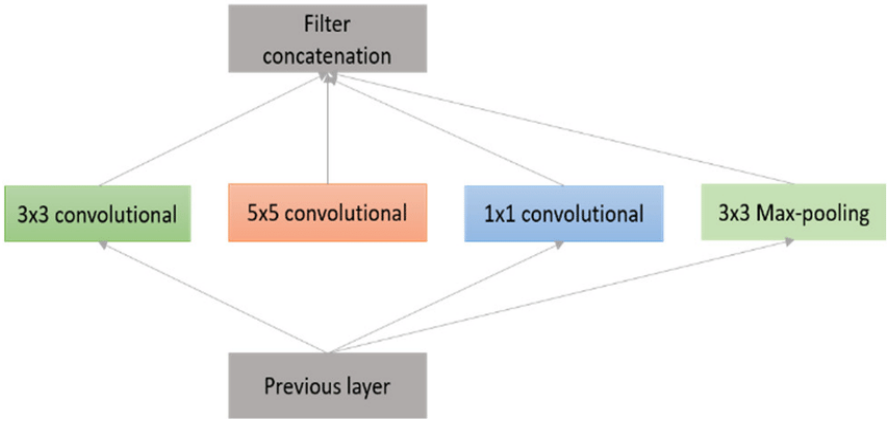


рис.60

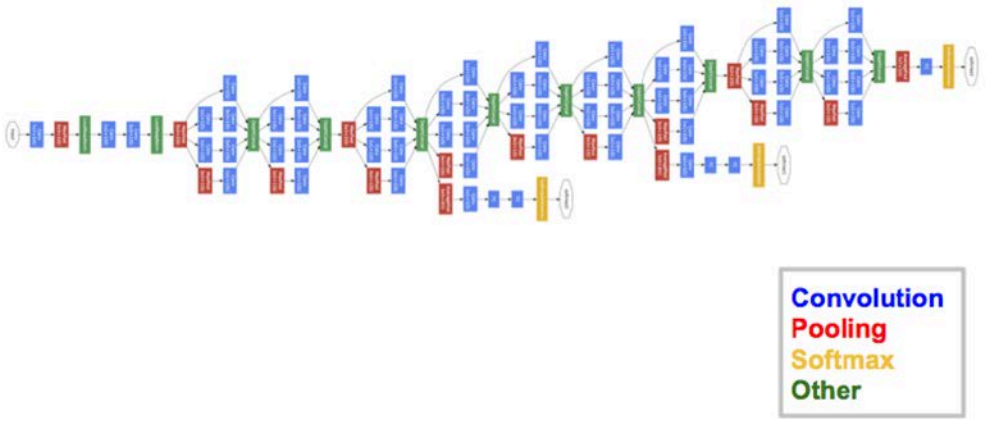


рис.61

Такая архитектура, состоящая из 9 основных блоков, достаточно капризна в обучении.

Однако простое добавление глубины может привести к падению качества итоговой модели. Можно доказать, что существует глубокая нейросеть с произвольным количеством слоев с качеством работы не хуже, чем у неглубокой нейросети, при этом она будет обучаться хуже. Поэтому авторы этой архитектуры предложили ввести специальный блок.

Рассмотрим 2 свертки:

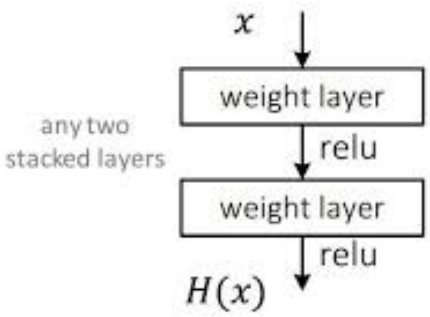


рис.62

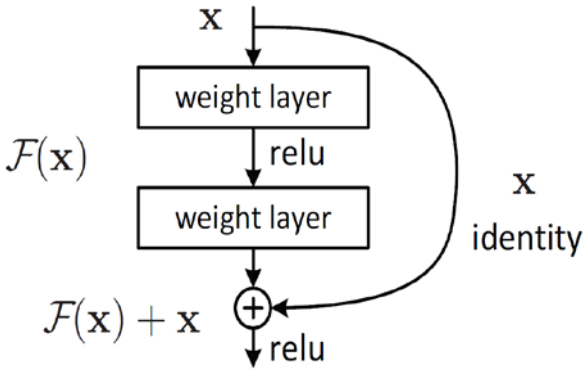


рис.63

В базовой сети между 2 свертками (рис.62) нелинейная активация. Авторы предложили улучшение этого блока (рисю.63) – Residual net (остаточная сеть). В процессе обучения сигнал может проходить по skip-connection до самого конца, при этом если нейросеть захочет, она может обучить какой-то остаток, который мы будем добавлять. Тогда будем учить не преобразование, а добавку (пертурбацию) к тождественному преобразованию.

Если такие блоки вставить в глубокую нейросеть, то мы получим ResNet (рис.64)

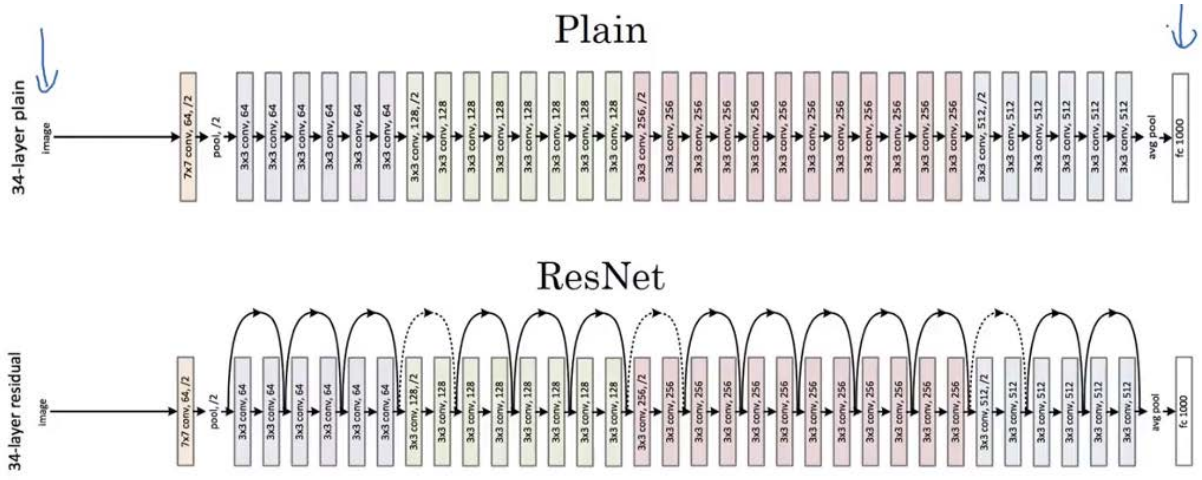


рис.64

В отличие от VGG с двумя полносвязынми слоями, в ResNet везде свертки. Базовая модель этой архитектуры состоит из сверток 3x3, если нужно уменьшить разрешение используем свертку с шагом 2, а не maxpooling.

Для того, чтобы эта архитектура была более эффективной, мы можем блок Residual net заменить на блок со сверткой 1x1 (рис.65).

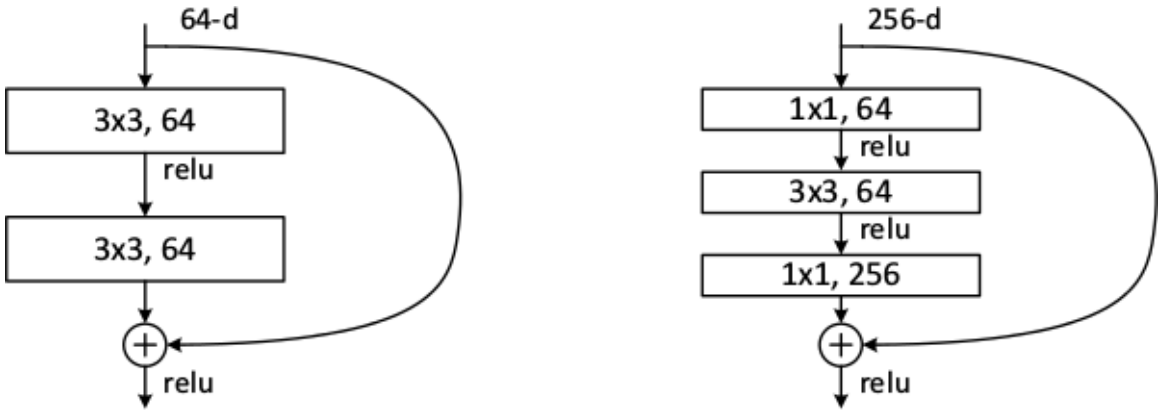


рис.65

Таким образом на вход поступает тензор из 256 слоев, мы его сжимаем до 64 слоев, применяем одну свертку 3x3, а потом обратно разжимаем, чтобы у нас все тензоры совпадали по размеру.

В результате, чем глубже ResNet (до разумного предела), тем выше качество классификации. При этом ResNet с 152 слоями будет легче, чем VGG. С развитием нейросетей количество слоев (можно сказать) экспоненциально растет, а точность линейно убывает (рис. 66).

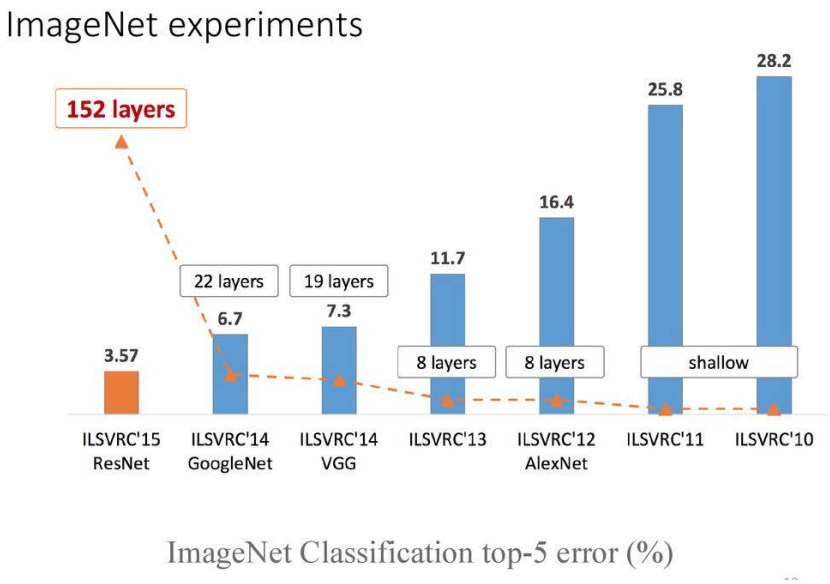


рис.66

DenseNet (плотные сети) (рис. 67)

Базовый блок DenseNet состоит из BN, ReLu и свертки, которые идут в 4 слоя, каждый из которых связан со всеми предыдущими.

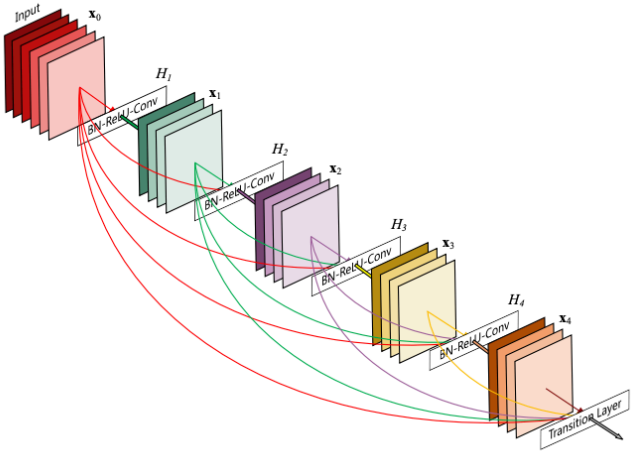


рис.67

Предобучение

Предположим, что мы можем собрать датасет больше, чем ImageNet, и обучить на нем нейросеть. Что получится? Google провел такое исследование: вместо 1 млн изображений они обучали нейросеть на 300 млн изображений, причем у каждого изображения есть несколько категорий, разметка очень шумная – около 20% ошибки, это все обучается на кластере из 50 тесл (стандартный ResNet – 101). Этот датасет использовался как базовая сеть на MS COCO (задача сегментации изображений). Далее рассматривалось 2 варианта Fine-tuning и No Fine-tuning (рис.68). Оказалось, что простое увеличение количества примеров позволяет отказываться от Fine-tuning. Также обратим внимание на график, здесь логарифмический масштаб, а точность растет линейно, значит для того, чтобы линейно увеличивать точность. нужно экспоненциально увеличивать количество данных. 300 млн изображений – это примерное количество изображений, которое видит ребенок до 5-6 лет. Это говорит о том, что больше 300 млн изображений в принципе не нужно, потому что человек хорошо обучается на таком датасете.**SqueezeNet**

Отдельная тема исследований – ускорение нейросетей. Например, с помощью сверток 1x1 можно переделать обычный AlexNet так. чтобы у него было в 50 раз меньше параметров, а точность осталась неизменной. Это говорит об избыточности AlexNet.

Дугой способ ускорения нейросетей – факторизациясверток (рис. 69) - использование сверток с меньшим количеством параметров. Для каждого слоя входного тензора возьмем одну свертку, то есть каждая свертка будет смотреть на один входной слой.

Результаты этих сверток просуммируем с помощью свертки 1x1, тем самым мы уменьшим количество параметров в **количество раз** слоев. Таким образом можно получить архитектуры, которые можно запускать на мобильных устройствах, например, MobileNet. Мобильные архитектуры имеют точность 40-44%.

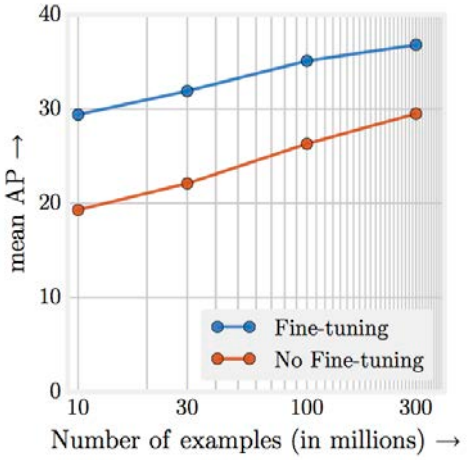


рис.68

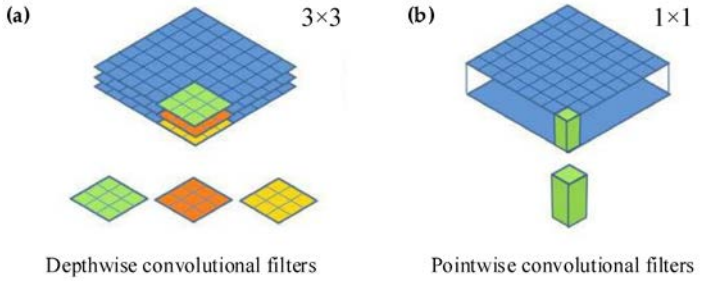


рис.69

Сложность обычного сверточного слоя: $D_k * D_k * M * N * D_f * D_f$

Сложность комбинации depthwise convolution + pointwise convolution: $D_k * D_k * M * D_f * D_f + M * N * D_f * D_f$

Резюме базовых архитектур

- Основные принципы построения архитектур
  - Строить глубокие модели из похожих блоков
  - Большие свертки можно приближать последовательностью сверток 3x3
  - Свертки 1x1 позволяют управлять «толщиной» слоя и уменьшать вычислительную сложность
  - Можно обрабатывать данные параллельно в нескольких ветвях и объединять их
  - Residual-связи позволяют снизить проблему затухания градиента и обучать очень глубокие сети
- Предобучать нейросеть лучше на как можно большей и разносторонней коллекции

Нейросетевые признаки для поиска похожих изображений

Выходы из верхних слоев базовой нейросети можно использовать как вектор-признак изображения. По этим вектор-признакам можно искать похожие изображения. Можно брать «готовую» сеть, но лучше дообучать ее на целевой коллекции.

Для поиска похожих изображений был собран специальный датасет – «достопримечательности». Нейросеть, обучающаяся на таком датасете, выдает более точный результат. Но поиск по полносвязным дескрипторам выдает большую погрешность, так вместо изображения черного кота сеть может выдать изображение черного лемура. Если мы будем использовать для поиска сверточные признаки достаточно высокого уровня, то изображение на выходе будет совпадать точнее.

Классический подход для обучения признаков - contrast loss с использованием евклидовой метрики. Подход заключается в следующем: два изображения пропускаем через одну нейросеть и извлекаем признаки. Если изображения похожи, то функция потерь должна требовать, чтобы извлеченные признаки были достаточно близки, и наоборот, если изображения разные, то признаки должны быть далеки.

$$L\big(\theta,(X_1,X_2,Y)\big)=(1-Y)L_G(\|X_1-X_2\|_2)+\ YL_I(\|X_1-X_2\|_2))$$

L_G- функция потерь для положительной пары. L_I- для отрицательной

Как правило, $L_G(d)=\ d^2, L_I(d)=\max{(0,m-d)^2}$

В качестве меры близости можно использовать и классическую меру $L(\theta,X_1,X_2,Y)=\frac{1}{2}\big(Y-\sigma(wd+b)\big)$, где близость $d=\ \frac{x_1x_2}{\|x_1\|_2\|x_2\|_2}$

Более сложная и более современная функция потерь Triplet Loss (рис.70) , которая состоит из запроса q (query), релевантного изображения и нерелевантного изображения. То есть в ней мы прогоняем через нейросеть 3 изображения: изображение-запрос, правильный ответ и неправильный ответ, сравниваем признаки с помощью triplet-loss, который будет требовать, чтобы признаки запроса и правильного ответа были достаточно близки, а признаки запроса и неправильного ответа – далеки. Эта функция потерь лучше, чем contrast loss, но имеет свои особенности. Например, при обучении с triplet-loss (рис. 71) нам нужно брать неправильный ответ, который является наиболее сложным, но при этом классифицируется правильно. Triplet loss используется для задачи детекции лиц. Например, FaceNet

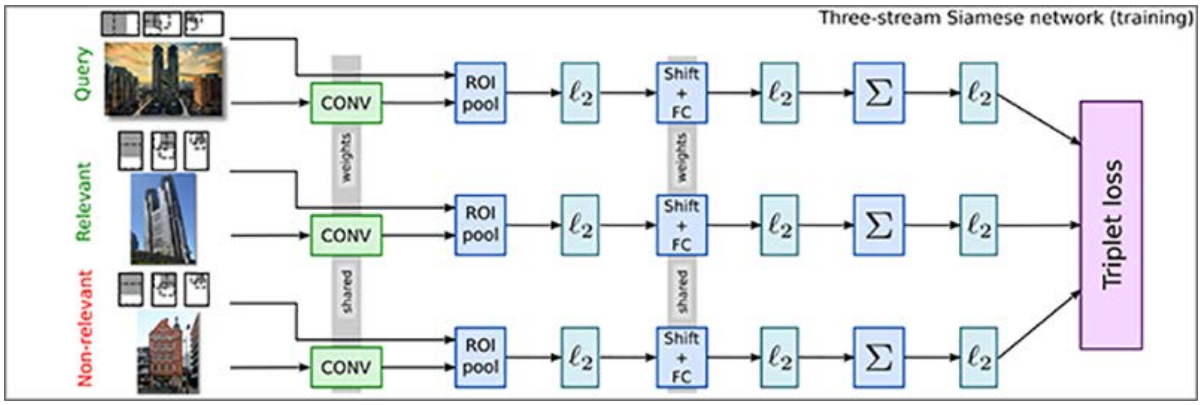


рис.70

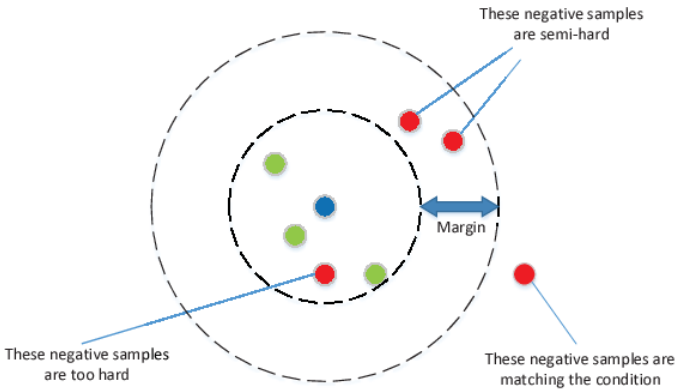


Figure 5. Triplet Selection Mechanism. Correct triplet selection is

рис.71

Лекция 6. Детектор объектов

Задача детекции

Задача детекции – определить присутствуют ли объекты заданных классов, если присутствуют, то определить их конкретное положение.

Формализуем задачу:

- Пусть задан набор из s интересующих нас классов объектов, пронумерованных от 1 до s
- *Вход:* цветные (необязательно) изображения $I \in R^{N \times M \times C}$
- *Выход*: набор найденных объектов $Obj = \{Obj_i | i = 1, p\}$, где Obj_i - структура, описывающая локализацию объекта и его класс $c_i \in [1, s]$

Можно придумать несколько вариантов для описания локализации объекта. Например, ограничивающий прямоугольник, который допускает разную параметризацию (угол, ширина, высота итп). Помимо прямоугольника можно использовать эллипс или окружность в зависимости от формы объекта. Чтобы увеличить точность локализации объектов, нужно добавить поворот объекта. Чтобы наиболее точно описать положение объекта, мы может описать границу объекта, либо область объекта.

Определение. Детекторы (object detectors) – алгоритмы, выделения объектов на изображении

Какого рода объекты мы можем выделять? Условно мы можем разделить объекты на 2 большие категории: things – объекты, которые можно локализовать с помощью ограничивающего прямоугольника (машина, человек, корова), и Stuff – материал, который не имеет определенного формы и размера и описывается в основном текстурой (небо, трава, дорога).

Чтобы решить задачу детекции (как и любую другую задачу компьютерного зрения) нужно разобраться с датасетом. Есть огромное количество эталонных коллекций с разметкой объектов для задачи детектирования, большая часть из которых содержит разметку только одного класса объектов, например, лица, головы или фигуры человека. Сейчас в статьях рассматриваются более сложные задачи, в которых присутствует не один, а множество классов объектов. Например, коллекция Microsoft COCO, которая состоит из 200 тыс. изображений. 80 классов и 500 тыс. выделенных объектов с маской. Сейчас в ряде конкурсов отказались от детекции объектов с помощью ограничивающей рамки (через bbox) и рассматривают только маски (instance segmentation).

Допустим, мы разработали метод, который выдают ограничивающий прямоугольник. Как оценить точность локализации? Стандартным критерием обнаружения ошибки является критерий IoU (Intersection over Union) или вычисление значения области пересечения эталонного прямоугольника и выданного прямоугольника к области их

объединения, если это соотношение больше заданного порога, то мы правильно выделили объект, если меньше – значит, объект не найден. Чем выше порог, тем более точно мы требуем локализовать объект. На практике очень многие методы работают с IoU = 0,5, это означает, что область пересечения может быть в 2 раза меньше, чем область объединения. Для оценки качества многих методов часто используется усреднение: мы оцениваем характеристики работы метода для нескольких значений, потом их- усредняем. Критерий IoU мы используем для того, чтобы для каждого конкретного прямоугольника сказать, можем ли мы его сопоставить с эталонным прямоугольников. Для оценки качества детектора нам нужно сопоставить все множество объектов со всем множеством объектов, размеченных на изображении, поэтому мы можем классифицировать все выданные прямоугольники. Если существует эталонный прямоугольник такой, что по критерию IoU он считается верным обнаружением, то это истинно положительное обнаружение. Если не существует такого эталонного прямоугольника, для которого выполнено это неравенство, тогда выданный прямоугольник – это ложное срабатывание false positive. Также есть примеры в коллекции, для которых не существует выданных прямоугольников, значит это false negative, пропущенный пример.

- $IoU > p \rightarrow true\ positive$
- $IoU < p \rightarrow false\ positive$
- Пропущенный пример $\rightarrow false\ negative$

Для того, чтобы посчитать интегральные характеристики, мы считаем такие параметры, как *точность* и *полнота.*

Определение. **Точность** (precision) – доля истинных объектов основного класса среди все классифицированных как основной класс

Определение. **Полнота** (recall) – доля правильно распознанных объектов основного класса среди всех объектов основного класса и тестовой выборки.

$$Presicion = \frac{number\ of\ true\ detections}{number\ of\ detections}$$

$$Recall = \frac{number\ of\ true\ detections}{number\ of\ gt\ objects}$$

В большинстве случаев есть возможность регулировать некоторый параметр. Например, это может быть порог на score классификатора, то есть классификатор на найденные объекты дает степень своей уверенности, мы сравниваем эту степень уверенности с порогом, и те обнаружения, в которых классификатор мало уверен мы отбрасываем и проверяем только те, в которых он уверен. Варьируя этот порог, например, понижая его, мы в качестве ответов будем получат все больше и больше малоуверенных обнаружений. Скорее всего таким образом мы повысим полноту, но среди этих обнаружений с малой степенью уверенности будут и ложные срабатывания,

поэтому одновременно будет падать точность. Поэтому кривая точности и полноты имеет следующую форму (рис. 72) :

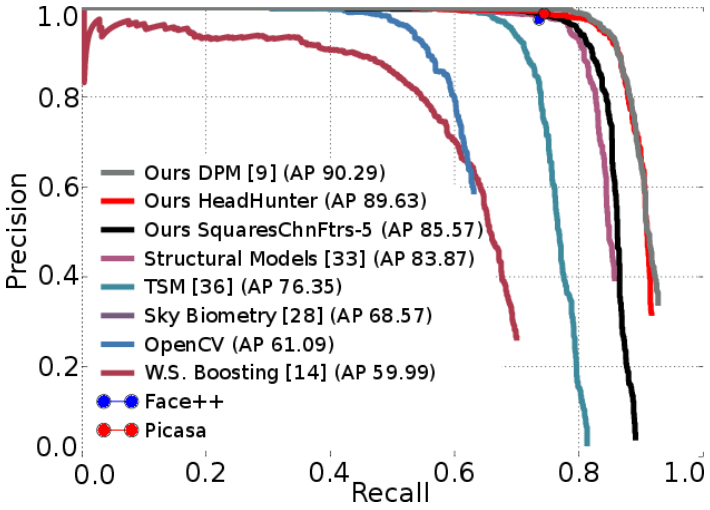


рис.72

- Зависимость между точностью и полнотой в зависимости от параметра
- Average Precision (AP): $AP = \frac{1}{11}\sum_{r\in(0,0.1,\ldots,1)} p_{interp}(r)$
- Mean Average Precision (mAP): average across classes
- По кривой точности-полноты мы можем выбрать параметр с оптимальным для наших целей соотношением ошибок)

Для того, чтобы посчитать интегральную характеристику алгоритма иногда делают следующее: считают среднюю точность- берут несколько значений полноты и усредняют значения точности между этими значениями.

Иногда используются другие метрики: доля ложных срабатываний на одно изображение против доли пропущенных объектов (1- recall). Такая метрика важна, когда мы работаем с видеопотоком с большим количеством кадров. Если доля ложных срабатываний 10^0, то есть одно ложное срабатывание на кадр, то мы в каждый момент времени видим какой-то ложный объект, что на практике неприемлемо. Поэтому когда используются эти критерии в качестве интегральной характеристики, обычно усредняют долю пропущенных объектов для доли ложных срабатываний 10^{-3}, 10^{-2}, 10^{-1}, то есть от одного ложного срабатывания на 10 кадров, до одного ложного срабатывания на 1000 кадров. Понятно, что, когда мы говорим об одном ложном срабатывании на 100 кадров, доля пропущенных изображений у нас очень большая. Например, на графике детектирования пешеходов (рис.73), в том числе и нейросетевыми методами, если мы требуем долю ложных срабатываний 1 на 100 кадров, то наилучший метод при этом пропускает 40% пешеходов на изображении. Для беспилотного автомобиля такая точность была бы недостаточна.

Если мы хотим получить практически применимый алгоритм, нам нужен баланс характеристик: никакая характеристика не может опускаться ниже заданного значения. Если метод не может обеспечить данный баланс, то для этой задачи практически применимых методов нет. Важно помнить, что все эти характеристики всегда измеряются на фиксированном наборе данных, поэтому важно, чтобы этот набор хорошо описывал реальность.

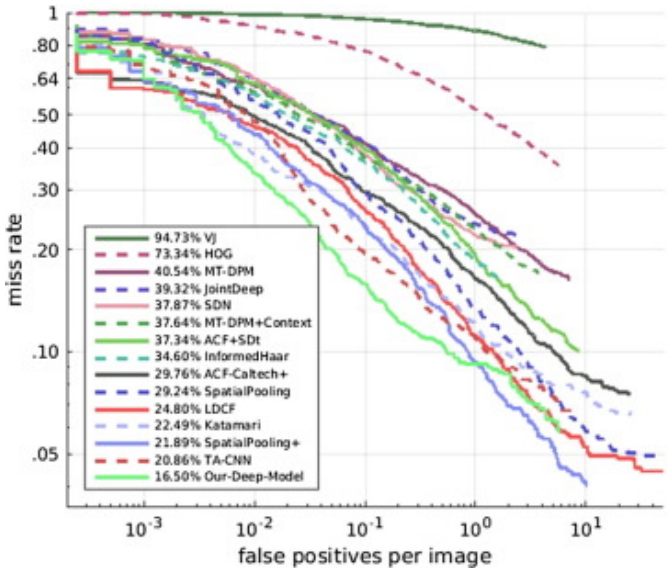


рис.74

Проблема с разметкой

Подход к разметке в различных наборах данных и у различных людей очень сильно отличается. Одна работа 2014 года показала, что за последние несколько лет перед этой работой разница характеристик алгоритмов полностью нивелируется правильной разметкой данных. То есть если мы правильно переразметим данные, то старый метод покажет ту же самую точность, что и новый.

Таким образом можно выделить следующие проблемы:

- Разметка в разных датасетах и от разных ассесоров существенно различается
- Есть объекты, которые лучше игнорировать
- Почти всегда есть проблемы с маленькими и большими объектами

Поэтому очень важен строгий протокол разметки. В одной из коллекций по детектированию людей (рис.74) протокол был следующим: требуется указать самую верхнюю точку на фигуре человека и соответствующую ей точку на земле, после чего ограничивающий треугольник рассчитывался автоматически по маске человека, человек сегментировался и находился наиболее точный прямоугольник.

рис.74

Выделить ограничивающий прямоугольник можно разными способами: мы можем выделить ограничивающий верхний левый угол, потянуть мышкой и указать правый нижний угол, но это требует сопоставления с верхней границей, и правильно выделить левый верхний угол в пустоте оказывается очень сложно. Недавняя работа показала, что если мы будем кликать не на границы прямоугольника, а на границы фигуры человека, укажем самые верхние, нижние, правые, левые точки, то такая операция для человека гораздо более естественная для человека, кроме того она повзоляет сэкономить в 3 раза время. Такой подход использовал Google для своей коллекции open images.

Метод скользящего окна

Определение. **Скользящее окно**- просмотр всех фрагментов изображения и применение к ним классификатора.

Мы можем свести задачу локализации объектов к задаче бинарной классификации объектов методом скользящего окна. Будем выбирать фрагменты изображения и применять к ним классификатор. Идея простая, однако ку не есть несколько фундаментальных ограничений:

- Разные масштабы
- Разные пропорции
- Множественные отклики
- Перекрытие

Чтобы решить проблему объектов разного размера, мы можем построить пирамиду масштабов и сканировать окном одного размера. В таком подходе в зависимости от качества признаков и классификатора может потребоваться разное число масштабов. Чем мощнее классификатор, тем меньше масштабов нужно просмотреть. До появления нейросетевых методов для достижения высокой точности работы масштаб приходилось

брать с шагом 5%, то есть приходилось брать 50 масштабированных копий изображений, которые нужно просмотреть для детектирования объекта.

Чтобы решить проблемы разных пропорций, будем рать окна с разными пропорциями. Но теперь для каждой пропорции окна нужно просмотреть еще все масштабы, то есть

$$Number\ or\ images\ scanned = \ Number\ of\ scales\ \times Number\ of\ aspect\ rations$$

Поэтому чтобы справиться с такой задачей, требуется очень быстрый классификатор.

Что делать с множественными откликами? Если мы используем критерий IoU, существует множество правильных прямоугольников, которые содержат объект. Чтобы выбрать наилучшие на практике чаще всего начинают с простого **жадного алгоритма**:

- На вход список окон, отсортированный по убыванию score (выхода классификатора)
- Выбираем окно с наибольшим score
- Находим и убираем все окна, для который $\text{IoU} > 0.5$
- Повторяем до сходимости

Вместо того, чтобы подавлять окна, мы можем взять все окна, которые сопоставились с самым сильным и усреднить их. Когда на изображении много близкорасположенных объектов, такой метод оказывается не очень точным.

Воспользовавшись схемой скользящего окна, построим простой детектор объектов. Простейшим методом будет сопоставление шаблонов. В качестве классификатора будем использовать просто пороговый классификатор, а в качестве вектор-признака объекта будем брать все пиксели объекта. Берем изображение объекта в качестве шаблона, сканируем этим шаблоном все изображение, в каждом положении скользящего окна мы сравниваем фрагмент изображения с шаблоном и если по нашей метрике он оказался похож больше, чем на порог, то объект обнаружен.

Далее, с развитием методов усложнялись классификаторы и используемые для классификации признаки. Например, мы можем воспользоваться SIFT: в изображении будем вычислять дескриптор SIFT для скользящего окна и его подавать на вход классификатору.

HOG + SVM

- Скользящее окно
- Признаки HOG/SIFT
    - Окно 64x128 пикселей
    - Разобьем его на блоки 8x8 пикселей
    - Всего будет 8*16=128 блоков
    - В каждом блоке посчитаем гистограмму ориентаций градиентов с 8 ячейкамиВс (парметрами)
    - Всего получится 128*8=1024 признаков

- Обучим линейный SVM признак

Для обучения классификатора нам нужны как положительные, так и отрицательные примеры, однако не все отрицательные примеры подходят для обучения: некоторые простые, некоторые более сложные. Поэтому очень часто применяют процедуру итеративного обучения, на каждом из этапов которого происходит процедура нахождения *трудных* примеров. Мы считаем пример *трудным* для нашего метода, если метод на нем ошибается. Чтобы найти *трудные* примеры воспользуемся процедурой negative mining:

- Выбираем отрицательные примеры случайным образом
- Обучаем классификатор
- Применяем классификатор к данным
- Добавляем ложные примеры к обучающей выборке
- Переобучаем
- Повторяем несколько раз процедуру

Смысл: ложные обнаружения для первого детектора – сложные.

Можно сравнить, насколько меняется интегральная характеристика алгоритма после нескольких этапов переобучения на трудных примерах (рис.75). Метод существенно улучшил свою точность, не поменяв выборку.

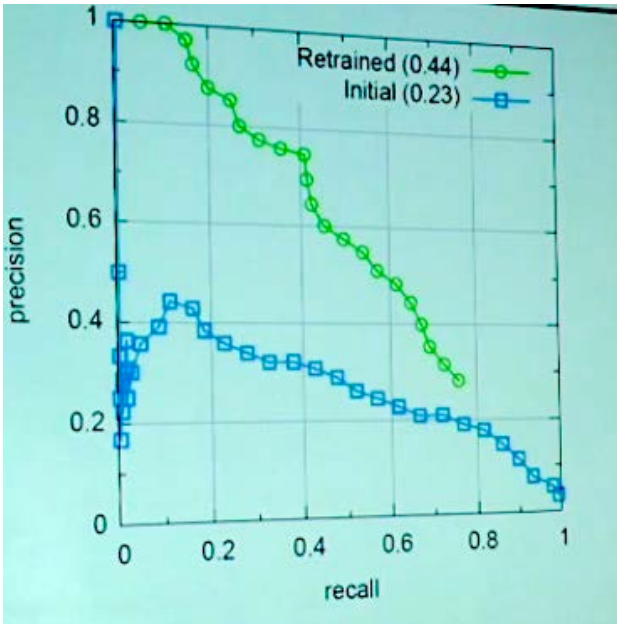


рис.75

Самый простой способ использования мощного классификатора для решения задачи детекции – просто применить в режиме скользящего окна нейросеть. Однако у этого

подхода есть существенный недостаток – это медленно. Здесь нам может помочь идея, которая появилась еще до нейросетевых методов – *каскад* (attentional cascade) (рис.76).

Идея каскада очень простая: вместо того, чтобы применять один классификатор, построим цепочку усложняющихся классификаторов. Нам нужно вначале применить очень быстрый метод, который отбросит часть отрицательных примеров, то есть у него может быть очень низкая точность, но должна быть высокая полнота. Если у него будет полнота близка к 100%, а точность 50%, значит он сэкономит нам огромное количество ресурсов на следующем этапе.

- Цепочка классификаторов
- Положительный отклик первого классификатора запускает вычисление второго, более сложного классификатора и т.д.
- Если она срабатывает отрицательно, дальнейшая обработка прерывается
- С каждым уровнем детектор становится более сложным, ошибка второго рода постоянно снижается

Самые быстрые методы, которые были до нейросетевых и которые до сих пор используются для детекции основаны на каскаде классификаторов.

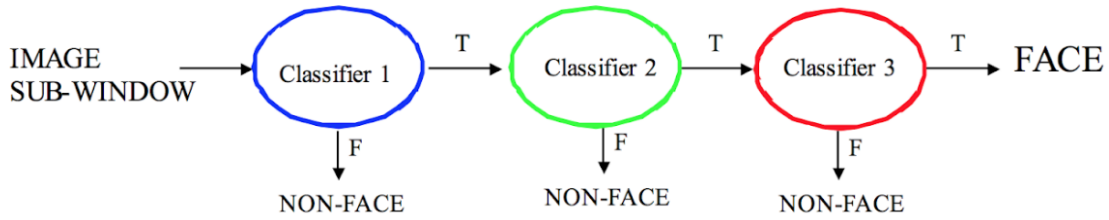


рис.76

Максимально сложный вариант: на каждом этапе оценивается только один признак, то классификатор превращается в цепочку максимально простых пороговых классификаторов.

Детекторы R-CNN

Первая нейросеть на основе каскада – Region-based convolution networks (R-CNN), по сути двухэтапный каскад (рис.77). Мы используем какой-то внешний алгоритм как генератор гипотезы объектов, с помощью него генерируем заданное количество гипотез, чтобы обеспечить достаточно высокую полноту. Затем каждую из этих гипотез мы вырезаем из изображения и подаем на вход классификатору.

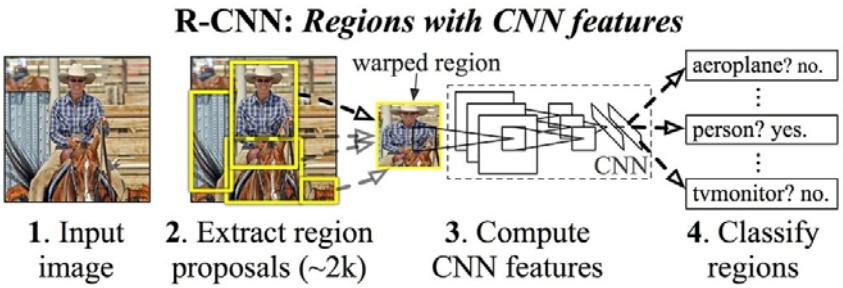


рис.77

Метод генерации гипотез, который использовался в R-CNN (регионально сверточных сетях), - один из методов иерархической сегментации, то есть этот метод последовательно разбивает изображение на однородные сегменты.

Обучение R-CNN (рис.78)

Поскольку нейросети на тот момент принимали на вход только квадратные изображения фиксированного разрешения, а сегменты, которые находил метод были произвольных пропорций, то нам нужно было их преобразовывать. При этом в тот момент эталонные коллекции для обучения были не очень большими по сравнению с теми коллекциями, ан которых обучались для задачи классификации. Поэтому примеров в этих коллекциях было недостаточно для того, чтобы обучить нейросеть с нуля. Поэтому нейросеть использовали только как метод извлечения признаков, брали выход одного из слоев, обычно полносвязного, и подавали его на вход методу опорных векторов. Фактически взяли метод HOG+CVN, но вместо того, чтобы использовать гистограмму ориентированных градиентов как признаки, брались нейросетевые признаки с сети классификатора. Это было достаточно сложно и медленно, но точность высокая.

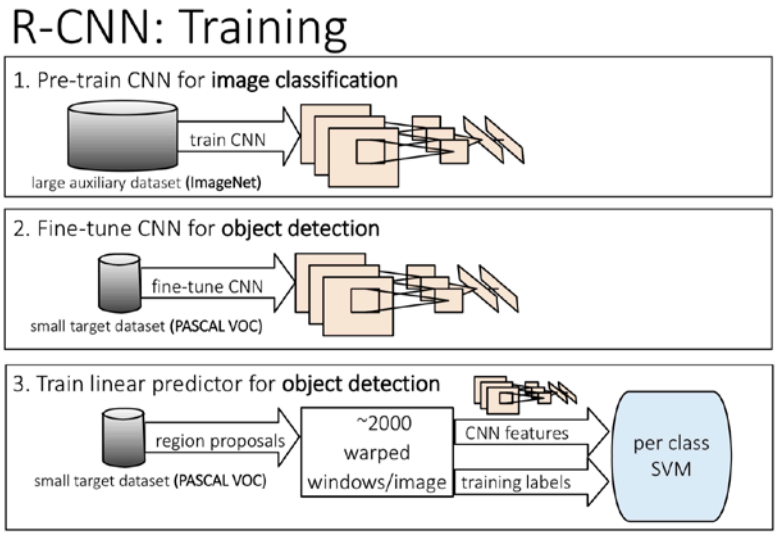


рис.78

При этом очень сильно повысить точность помогла еще одна идея – уточнение ограничивающего прямоугольника или *регрессия bbox* (рис.79). Суть идеи заключается в следующем:

- Исходные гипотезы bbox могут быть очень неточными, поскольку мы получаем их от внешнего метода и не знаем насколько точно они локализованы. За счет того, что нейросетевой метод устойчив к положению объектов внутри этого ограничивающего прямоугольника, даже если локализация не очень точная, он все равно выдаст правильный ответ: есть объект или нет.
- Чтобы повысить точность локализации, вместо классификатора на тех же самых признаках обучим линейную регрессию, то есть мы будем находить сдвиг ограничивающего прямоугольника относительно исходного, где именно должен располагаться объект.
- Такую регрессию можно легко обучить, и она существенно улучшает точность классификации.

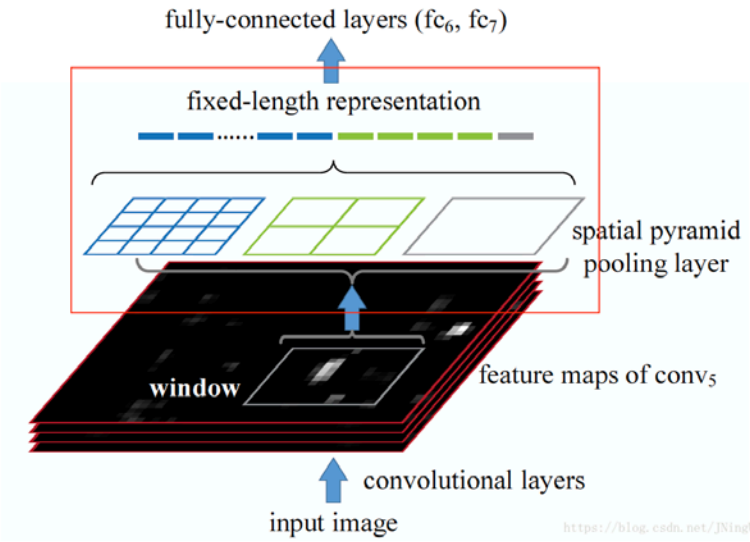


рис.79

Таким образом, метод регрессии bbox позволяет взять неточную гипотезу и уточнить ее. Такого не было до нейросетевых методов, до этого мы полагались на ограничивающий прямоугольник и для него только предсказывали метку: он или не он. Теперь мы можем не только предсказывать метку, но и уточнять ее, за счет этого мы можем просматривать гораздо меньше прямоугольников, чем раньше. Теперь мы можем брать прямоугольники, фрагменты с достаточно большим шагом на меньшем числе масштабов за счет того, что мы будем уточнять наши изначальные гипотезы значительно (рис. 80)

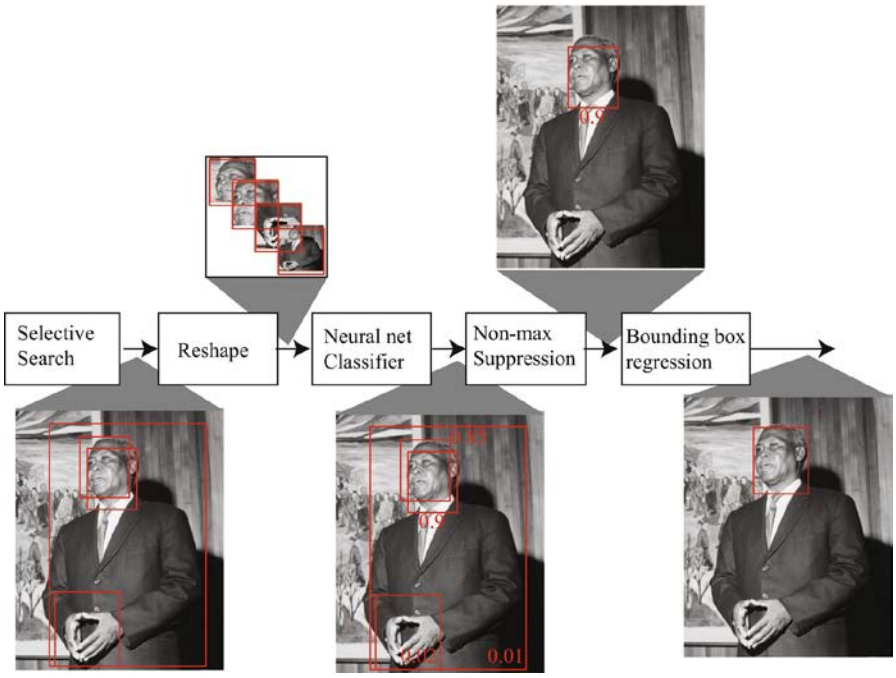


рис.80

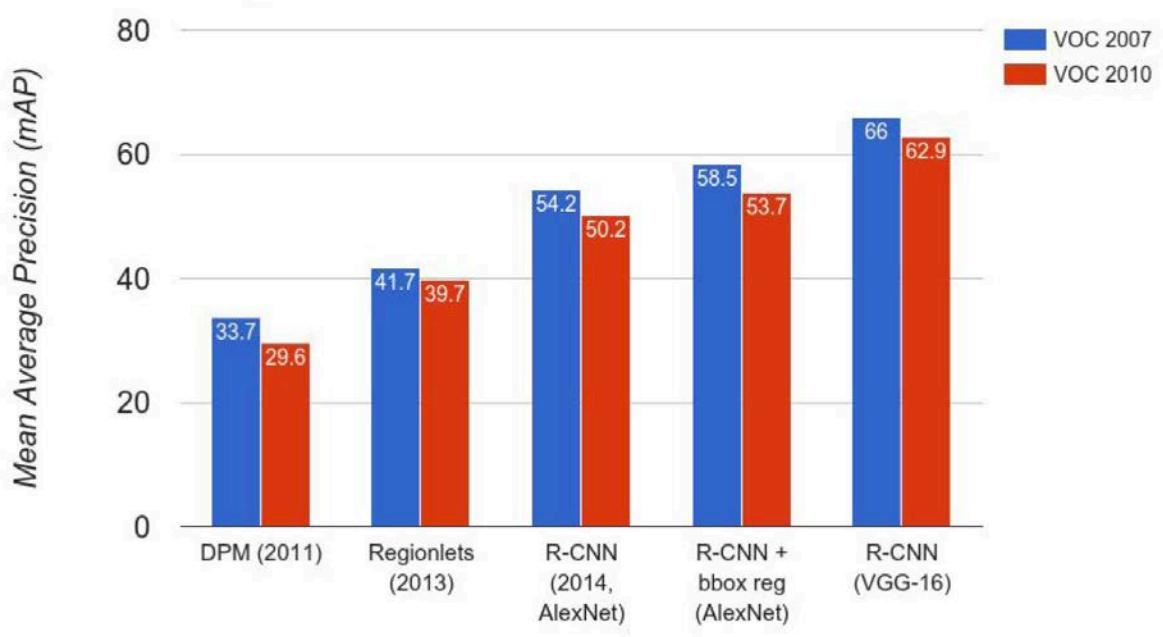


рис.81

Улучшение классификатора (AlexNet -> VGG16) повышает качество детектора.

У R-CNN было много недостатков:

1. Вычисляем нейросетевые признаки независимо для каждого окна-гипотезы. Поскольку они пересекаются, это ведет к избыточным вычислениям
2. Нужно масштабировать фрагменты-гипотез до нужного разрешения
3. Сложная процедура обучения
4. Зависимость от внешнего алгоритма генерации гипотез

Решить первую проблему помогла идея ROI-pooling (рис.82). ROI-pooling – это модификация pooling слоя, в котором операция pooling применяется не для всего изображения, а для региона интереса (ROI- Region of Interest). ROI-pooling позволяет нам делать следующее: вычислим сверточные признаки один раз для всего изображения, получим гипотезы объектов из вне (их ограничивающие прямоугольники). Эти гипотезы задаются на исходном изображении, мы посмотрим, каким регионам на последней карте сверточных признаков соответствуют эти гипотезы, отметим эти регионы и применим операцию pooling для получения признаков к конкретным регионам изображения.

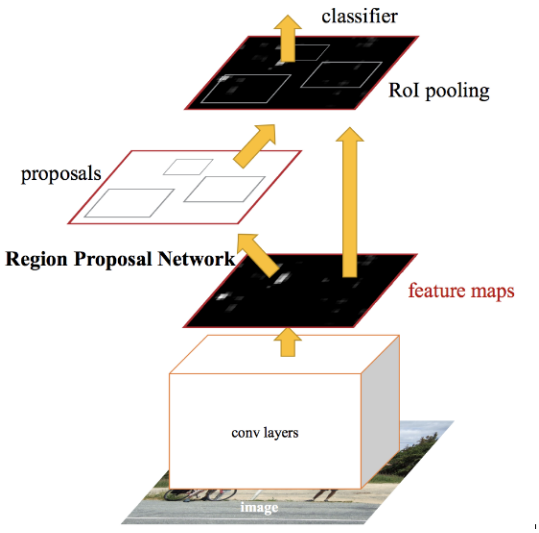


рис.82

Ранее мы рассматривали метод пространственного пирамидального пулинга (SPP), который применялся ко всему изображению, ко всем сверточным признакам. Но ничто не мешает нам применить этот метод не ко всем признакам, а к какой-то части, к какому-то региону интересов (рис.82). Но мы можем сделать еще проще, и вместо того, чтобы считать признаки по пирамиде, как мы делали в обычном методе SIFT, мы посчитаем признаки по сетке.

Разобьем наш регион сеткой какого-то разрешения, применим операцию pooling или max pooling к каждому региону, получим признак. Этот вектор-признак уже будет фиксированной длины и не будет зависеть от размера гипотезы и размера изображения, значит мы можем его спокойно подавать на последующий классификатор. Если мы вытянем все эти признаки в вектор, а не будем делать трехмерную матрицу, получим двухмерный вектор, то тогда мы можем подавать его на вход полносвязному слою. То есть мы как бы разбиваем классификатор на 2 части: одна часть (сверточные признаки) считается по всему изображению, а вторая часть (полносвязные слои и классификатор) считается для каждой гипотезы в отдельности.

Пример: предположим, что ROI-pooling считает по сетке 2x2, а не 4x4. Для того, чтобы посчитать по сетке, нам нужно нашу гипотезу отобразить к ближайшим значениям координат на нашем сверточном слое. Чем глубже сверточный слой, тем меньше разрешение этого сверточного слоя за счет pooling слоев. Операцию ROI- pooling мы обычно применяем к последнему сверточному слою. Современные нейросети так устроены, что обычно последний сверточный слой имеет разрешение в 32 раза меньше, чем исходное изображение. Соответственно один пиксель в этой трехмерной матрице соответствует 32 пикселям в исходном изображении. Гипотеза необязательно будет генерироваться шагом в 32 пикселя, она может генерироваться с точностью до пикселя. Нам нужно выбрать регион с точностью до пикселя, то есть с шагом 32 пикселя, поэтому мы отображаем это значение до ближайшего целого. Например, получаем вот такую матрицу (рис. 83) 7x5. Теперь ее нужно разбить сеткой 2x2. Получаем 4 области, в каждой из которой применяем выбранную операцию pooling, например, max pooling. Получим результат (рис.84). Полученную карту вытягиваем в вектор и получаем вектор-признак с данного региона интересов.

Операция ROI-pooling эффективная, но включает в себя округление до ближайшего целого для каждого региона, что вносит некоторую систематическую погрешность, поэтому иногда ее заменяет на другую операцию ROI-Align.

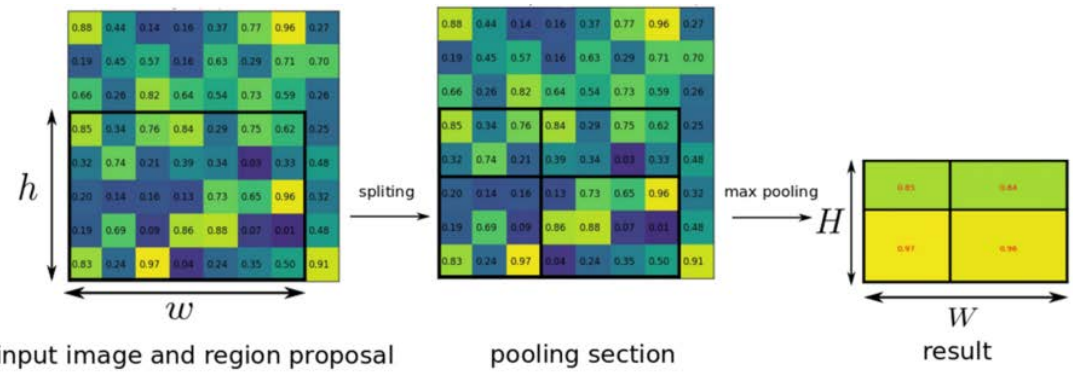


рис.83

рис.84

ROI-Align – это фактически передискретизация изображения. То есть мы отображаем наш регион интереса на карту признаков с пиксельной точностью и получаем дробные координаты. Затем разбиваем эту область сеткой и считаем значение в каждом элементе сетки уже с помощью линейной интерполяции по карте признаков, то есть берем не ближайшее значение, а линейно интерполируемое по соседям. Получившиеся признаки либо вытягиваем в вектор, либо используем как картинку. Оказывается, что такой метод за счет более точной аппроксимации дает более высокую точность для последующей классификации, чем ROI-pooling.

На основании ROI-pooling сделали модификацию метода R-CNN – Fast R-CNN (рис.85). Основные идеи этого метода:

Вычислять сверточные признаки по всему изображению, извлекать из конкретных гипотез вектор-признаки с помощью слоя ROI-pooling. И за тем вместо того, чтобы подавать их на вход SVM, подавать их на вход обычным полносвязным слоям нейросетевого классификатора и одновременно по этим признакам делать и классификацию с помощью soft max, и регрессию bbox. При этом мы можем обучать нашу сеть в режиме многоцелевого обучения. Таким образом, мы можем и дообучить и обучить сверточные признаки, которые мы вычисляем по всему изображению. Для того, чтобы использовать особенность этого метода, которая заключается в том, что он не требует повторного вычисления признаков, нам нужно с каждого изображения брать много гипотез. Гипотезой в нашем случае будет какое-то окно интересов, потому что мы вычислим для этого окна интересов ошибку, затем применим обратное распространение ошибки для получения градиента по всей нейросети. Соответственно одним элементом у нас является именно регион конкретного изображения (регион интересов).

Градиенты от всей нейросети занимают много места в памяти, поэтому обычно берут малое количество изображений в одном mini-batch, но с одного изображения собирают много гипотез. Из-за огромного размера современных нейросетей сейчас mini-batch может состоять из одной картинки и нескольких десятков гипотез.

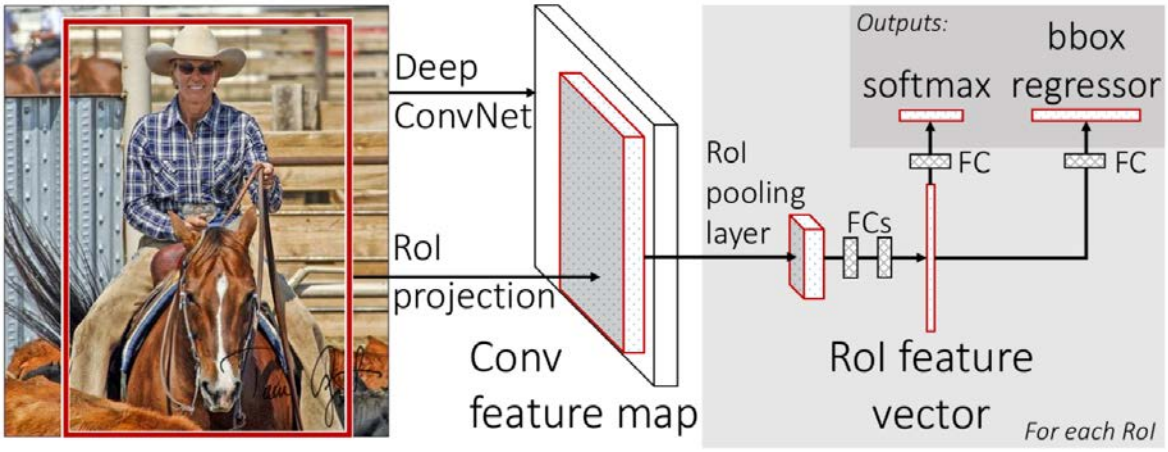


рис.85

Чтобы обеспечить сбалансированность выборки, нужно брать примерно одинаковое количество положительных и отрицательных гипотез внутри изображения. Если, например, мы можем в каждом изображении взять 64 гипотезы, нам нужно, чтобы 32 гипотезы из этой выборки были положительными и 32- отрицательными. Отрицательные гипотезы могут быть разными по качеству, поэтому на этом этапе можем применить процедуру, похожую на процедуру поиска сложных примеров для обучения нейросетей.

Метод Fast R-CNN по всем параметрам существенно превосходил базовый метод R-CNN (рис.86)

Следующий шаг, который нужно было сделать для улучшения сети – избавиться от внешнего генератора гипотез. Так появилась Faster R-CNN (рис.87).
Faster R-CNN – пример нейросетевой архитектуры, которая используется до сих пор.
По сути Faster R-CNN = Fast R-CNN + RPN (нейросетевого генератора гипотез).
Нейросетевой генератор гипотез – маленькая нейросеть, которая по тем же самым сверточным признакам генерирует гипотезы. Наиболее вероятные гипотезы подаются на вход ROI-pooling слою и затем классифицируются более мощным классификатором.

		R-CNN	Fast R-CNN
Faster!	Training Time:	84 hours	**9.5 hours**
	(Speedup)	1x	**8.8x**
FASTER!	Test time per image	47 seconds	**0.32 seconds**
	(Speedup)	1x	**146x**

рис.86

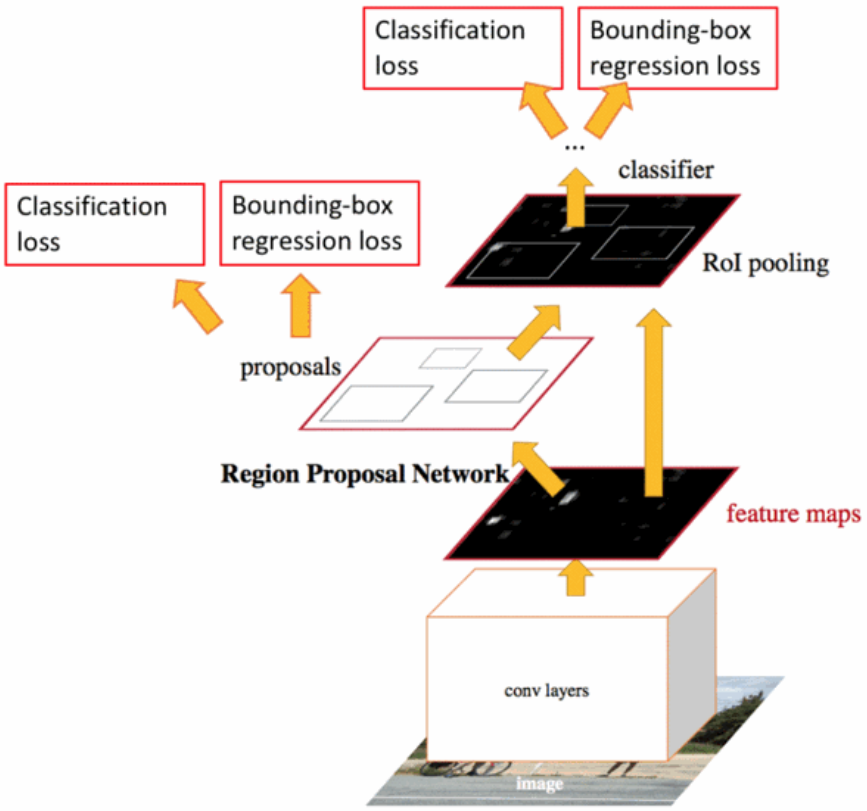


рис.87

RPN сети (сети генератора гипотез регионов) были устроены следующим образом (рис. 88):

- Маленькое скользящее окно по карте признаков (feature map)
- Маленькая нейросеть для
  - Классификации объект/не объект
  - Регрессии bbox
- Позиция окна показывает локализацию объекта относительно изображения
- Регрессия bbox показывает положение bbox относительно положения скользящего окна

Скользящее окно маленького размера, например, 3x3. То есть мы каждому положению скользящего окна применяем сверточный слой со свертками 3x3 с числом сверток 256. Так мы отображаем каждый фрагмент изображения размером 3x3 пикселя в вектор-признак длиной 256, который подаем на вход и классификатора, и регрессора. Задача

классификатора сказать, есть ли в соответствующей области объект, задача регрессора – уточнить положение этого объекта.

Поскольку мы применяем в режиме скользящего окна, то есть как свертку, мы можем реализовать генерацию гипотез всего изображения как еще один сверточный слой поверх последних сверточных признаков со свертками 3x3. Далее мы реализуем классификаторы с помощью сверток 1x1.

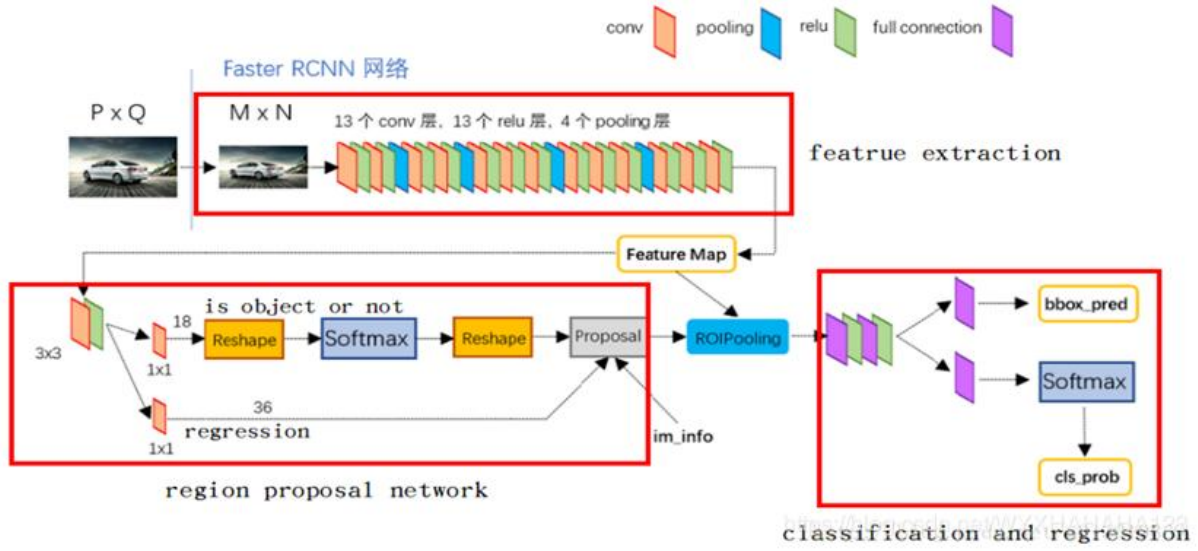


рис.88

У такого метода применения скользящего кона все еще остаются проблемы с масштабами и пропорциями, хотя нет проблем со скоростью. Для того, чтобы находить объекты разных размеров и разных масштабов, в методе Faster R-CNN придумали метод якорей.

Гипотеза является *якорем*, если

- Определяем набор из k гипотез
- У каждой гипотезы свой размер и пропорция

Якоря позволяют улучшить обнаружение объектов разного размера и пропорций (как обычно в методе скользящего окна). Если объекты разного размера, то классификатору тяжело научиться извлекать из маленького вектора длины 256 информацию о том, какого размера и формы объект. Получается, что мы делаем несколько классификаторов, каждый из которых заточен на объект определенных размеров и пропорций. То есть для каждого положения скользящего окна один классификатор будет пытаться угадать, есть ли в области с центром в данном окне поиска объект маленького размера и т.п. все это образует Faster R-CNN, который устроен следующим образом (рис.87):

- Есть какие-то сверточные признаки, по которым мы строим карту признаков
- Поверх карты признаков работает быстрая RPN сеть, которая генерирует гипотезы
- Выбираем заданное количество наиболее уверенных гипотез и передаем их на ROI-pooling слой
- ROI-pooling слой из этих гипотез извлекает вектор-признаки, которые подаются на вторую стадию каскада
- Полносверточный классификатор определяет, какие объекты располагаются и уточняет их положение

Скорость этого метода мы можем регулировать за счет числа гипотез, которые будем проверять.

Заменив базовую архитектуру в Faster R-CNN на ResNet, получим классический метод выделения объектов, который можно использовать как базовый.

Метод Faster R-CNN достаточно мощный, но зачастую нам нужны методы побыстрее. Самое медленное в этом методе – классификация гипотез с помощью полносвязного классификатора (второй этап).

Поэтому родилась следующая идея: выкинуть второй этап и сделать одностадийный метод. Для этого нужно усилить RPN. Так появились одностадийные детекторы, которые работают быстрее, но менее точно, чем Faster R-CNN. Одним из таких детекторов является YOLO (сокращение от названия статьи You Only Look Once). Как устроен этот детектор?

Разобьем изображение сеткой и сделаем такую нейросеть, которая будет для каждой ячейки предсказывать, есть ли объект, ограничивающий прямоугольник которого, - центр данной ячейки. Фактически, разбив изображение сеткой 7x7, мы сделаем 49 гипотез. Поскольку объекты бывают разных размеров и предсказывать их одним классификатором неудобно, мы воспользуемся похожей на якори идеей и будем в каждой ячейки выдвигать 2 гипотезы: одну маленькую, а другую большую. В сумме наши 49 ячеек сделают 98 гипотез. Параллельно каждую ячейку заставим предсказывать класс объекта. Для каждой гипотезы у нас есть вероятность классификатора. Применим метод подавления немаксимумов жадного подавления гипотез и оставшиеся гипотезы обрезаем по порогу по степени уверенности.

Если говорить о параметризации ответа (что мы хотим получить на выходе сети), то на выходе мы хотим получить трехмерную матрицу размера 7 x 7 x (2 x 5 + 20) = 7 x 7 x 30, так как

Сетка 7 x 7, 2 гипотезы объектов, 20 классов. То есть сеть на вход берет изображение. А на выходе предсказывает 7 x 7 параметров. Эта сеть называется DarkNet (рис. 89).

DarkNet устроен следующим образом: он состоит из сверточных слоев таким образом, что в конце у нас появляется матрица размером 7 x 7 x 1024. Далее эту матрицу

преобразуют в один полносвязный слой длиной 4096, этот слой содержит всю информацию про изображение. Из этого полносвязного слоя генерируется ответ 7 x 7 x 30.

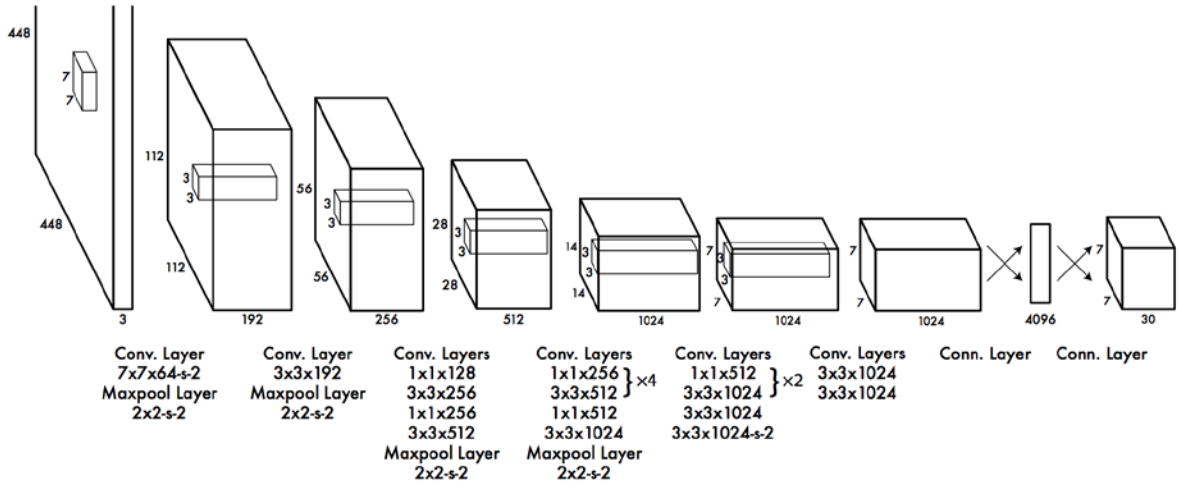


рис.89

Есть другой вариант, который был предложен в Google – Single Shot Detector (SSD) (рис. 90).

В отличие от детектора YOLO в нем нет этапа преобразования в полносвязный слой, в нем генерируются гипотезы для объектов по признакам, собранным со сверточных слоев. И в отличие от детектора YOLO, который делал одну сетку для всего изображения, в детекторе SSD генерировалось несколько масштабов, несколько разбиений изображений.

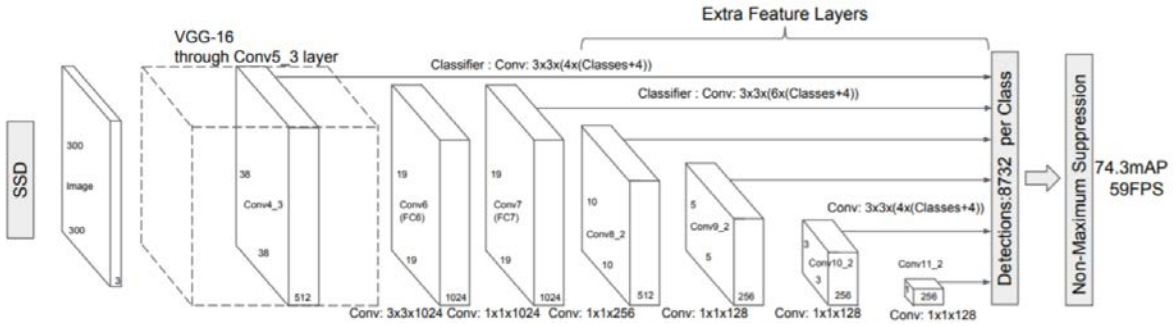


рис.90

Пирамида признаков (Feature Pyramid Network, FPN) (рис.91)

Будем предсказывать крупные объекты по очень глубокому сверточному слою, объекты меньших размеров – по менее глубокому сверточному слою. То есть в этом

слое один пиксель соответствует 16 пикселям, что позволяет предсказывать объекты точнее с меньшим шагом.

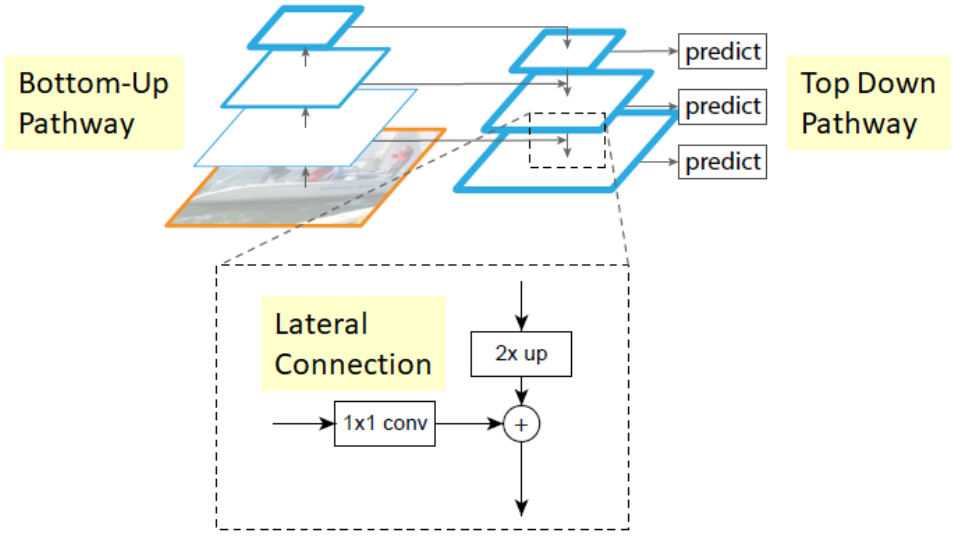


рис.91

Минус заключается в том, что, когда мы предсказываем маленькие объекты, мы не учитываем контекст, то есть чем глубже сверточные слои, тем с большей области изображения мы собираем признаки, то есть получаем большее рецептивное поле и больше информации о контексте, который важен для распознавания объектов.

Чтобы совместить эти признаки, мы можем сделать дополнительную подсеть внутри нашей сети специально для построения признаков. Эта сеть и называется сетью признаков изображения, которая устроена следующим образом:

- По самому глубокому сверточному слою базовой сети строим карту признаков того же самого разрешения с шагом 32 x 32
- Затем строим карту признаков с более высоким разрешением, объединив информацию с карты признаков низкого разрешения и соответствующего слоя базовое нейросети
- Повторяем несколько раз, строя таким образом пирамиду
- Далее к каждым слоям будем применять свои RPN сети

Построение FPN (рис. 92):

- Пусть глубина каждой карты признаков 256
- На каждом уровне складываем увеличенную в 2 раза карту меньшего разрешения с преобразованными признаками из feature extractor

- После объединения можем использовать фильтры 3 x 3
- Никаких нелинейностей при построении FPN

Объединив идеи одностадийного алгоритма детектирования объектов, построения пирамиды признаков и применение RPN сети по этим признакам, состоящей из 4 слоев 3 x 3 пикселов, сделали архитектуру Retina Net (рис. 93), которая является одной из наиболее точных архитектур для детектирования объектов в настоящее время.

RetinaNet = SSD + FPN + FocalLoss

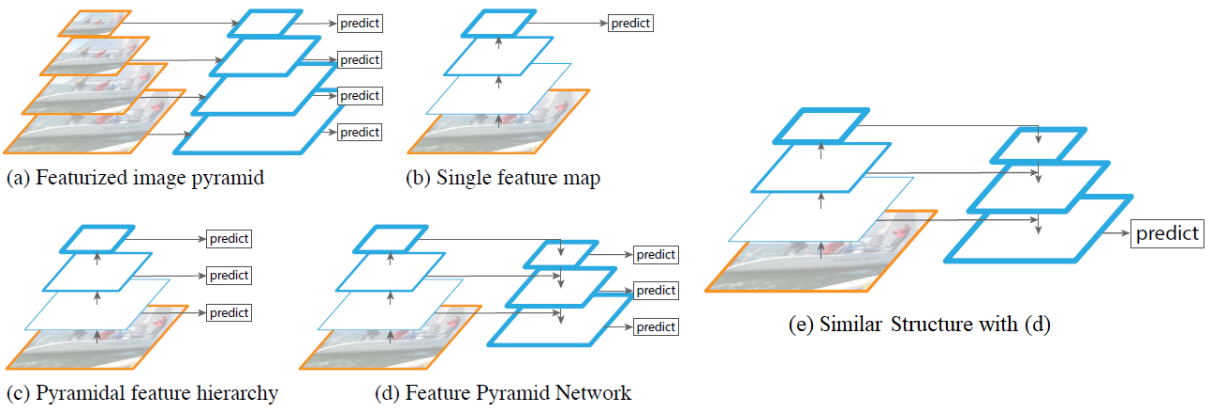


рис.92

Сейчас основной упор в развитии детекторов объектов идет в двух направлениях: усовершенствовании функции потерь и аугментации.

Одним из вариантов улучшения функции потерь является FocalLoss. Это попытка реализовать в сетях типа SSD и YOLO учет сложных примеров. В этой архитектуре при подсчете функции потерь классификатора вместо обычной кросс энтропии домножим логарифм на коэффициент (рис. 94). Этот коэффициент регулирует функцию потерь, придавая больший вес трудным примерам, что положительно сказывается на качестве классификации.

Другое направление развитие методов – улучшение аугментации. Мы можем существенно повысить качество работы, если будем обучать политику аугментации, то ест выбирать, какие способы аугментации картинок нам нужно использовать.

И третье направление развития – переход от предсказания bbox к попиксельному выделению объектов, наиболее общая задача – паноптическая сегментация.

Резюме детекции объектов

- Скользящее окно позволяет свести задачу детекции к задаче классификации
- Для работы с разными масштабами/пропорциями рассматриваем гипотезы (окна) разных размеров и масштабов
- На один объект получаем множество откликов, поэтому приходится подавлять «слабые» (NMS)
- ROI-pooling и варианты позволяют считать скользящие окна по картам признаков
- Для работы с разными размерами объектов лучше собирать признаки с разных слоев нейросети
- Ключевые методы – SSD, Faster R-CNN

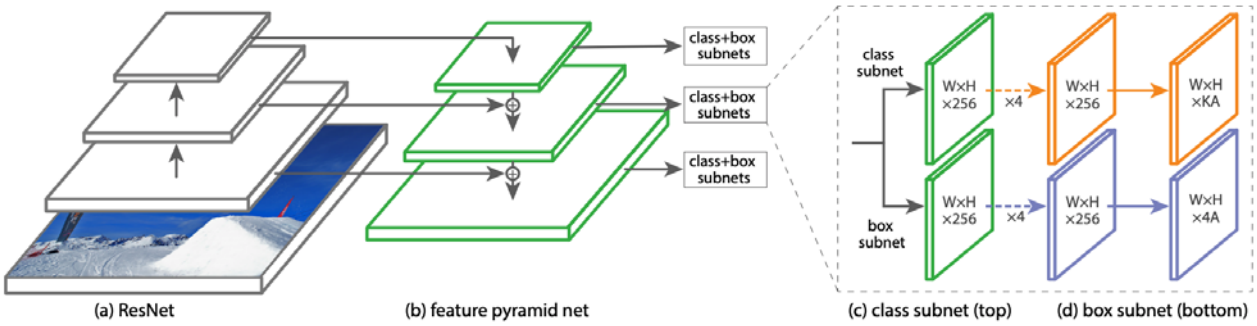


рис.93

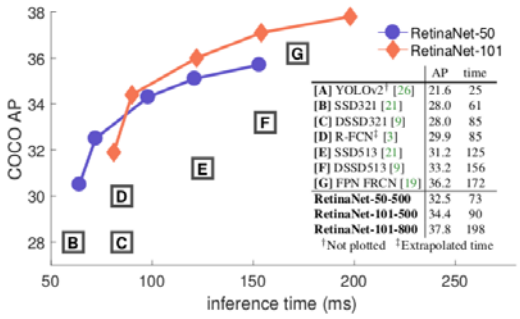


Figure 2. Speed (ms) versus accuracy (AP) on COCO `test-dev`. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [27] system from [19]. We show variants of RetinaNet with ResNet-50-FPN (blue circles) and ResNet-101-FPN (orange diamonds) at five scales (400-800 pixels). Ignoring the low-accuracy regime (AP<25), RetinaNet forms an upper envelope of all current detectors, and a variant trained for longer (not shown) achieves 39.1 AP. Details are given in §5.

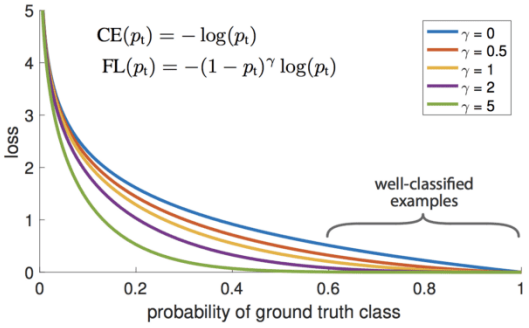


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

рис.94

Лекция 7. Задача сегментации

Задача сегментации

В общем случае *задача сегментации* – разделить изображение на фрагменты (группы пикселов) по некоторому общему критерию. В зависимости от критерия получаются разные задачи сегментации изображения.

1. **Извлечение объекта** – выделение конкретного произвольного объекта, указанного пользователем или по-другому заданного. Например, пользователь может выделить объект ограничивающим прямоугольником или нарисовать контур объекта.
2. **Сегментация без учителя** – разделение изображения на регионы, однородные по своим визуальным характеристикам и отличающиеся от соседних регионов. Например, нужно выделить области, соответствующие разным фрагментам на одежде человека (рис. 95)

рис.95

Поскольку такие методы разбивают объекты обычно на несколько сегментов, то их еще называют *методами пересегментации*, потому что мы сегментируем чрезмерно по сравнению с тем, как хотелось бы. Другой вариант – **семантическая сегментация.**

Семантическая сегментация – попиксельная разметка изображения, где каждая метка соответствует определенному объекту (рис. 96). При этом:

- Разные пиксели одного объекта существенно отличаются друг от друга по признакам (яркости, цвету, текстуре окрестности)
- Единственное, что у них общее – «семантика»
- Поэтому задача сегментации тесно связана с задачей распознавания.

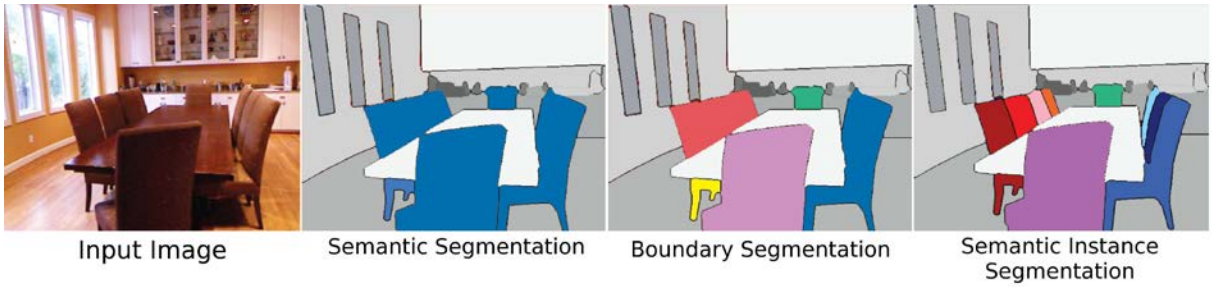


рис.96

На рис.96 мы проводим семантическую сегментацию на 3 класса: стол, стулья и фон.

Следующая постановка задачи – **сегментация экземпляров (Instance segmentation)**. Мы фиксируем классы объектов, которые хотим выделить и каждый экземпляр заданных классов мы помечаем своей меткой. Задача сегментации экземпляров – это дальнейшее развитие задачи детектирования объектов. Если в обычной задаче детектирования мы локализовывали объекты, например, типа «автомобили» и показывали, где они расположены с помощью прямоугольной рамки, то здесь нужно указать точную область, которую занимает каждый конкретный экземпляр.

В задаче сегментации экземпляров мы выделяем объекты типа things, которые хорошо локализованы в пространстве. В задаче семантической сегментации мы можем пометить области, которые соответствуют другой категории объектов, таких как небо и дорога. Разумеется, возникает задача, в которой и семантическая сегментация, и сегментация экземпляров являются подзадачами, такая задача получила название **паноптическая сегментация (panoptic segmentation)** (рис. 97).

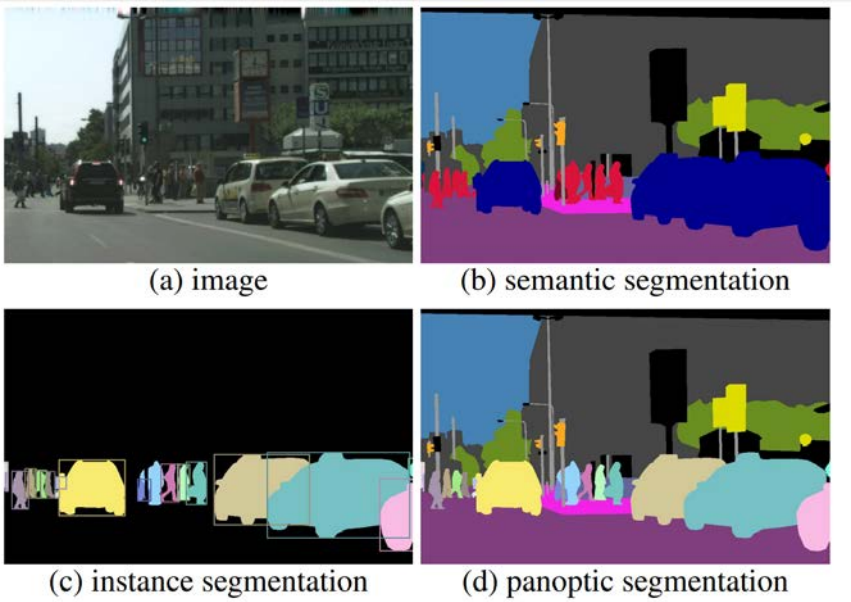


рис.97

Название «паноптическая сегментация» недавнее, оно было введено в работе 2018 года. Суть паноптической сегментации заключается в том, что каждому пикселю изображения мы придумываем 2 метки: одна метка – это метка классов объектов, что дает нам выход в карту семантической сегментации, вторая карта — это карта сегментации экземпляров, где каждому экземпляру объектов присваивается своя метка. Методы семантической сегментации оцениваются по совокупности точности распознавания объектов и точности разметки меток, и придумана своя метрика паноптической сегментации. Это означает, что каждому региону, каждому объекту мы можем назначить свою метку и свой цвет и визуализировать разбиение изображения и на объекты, и на классы.

Оценка точности сегментации

Как оценивается точность сегментации? Во-первых, есть попиксельные метрики, то есть мы оцениваем долю правильно отсегментированных пикселей на изображении независимо от того, какую задачу мы рассматриваем. Если метка пиксела верна – это хорошо, и рассматриваем долю верных меток пикселов по отношению ко всем пикселам.

Второй вариант оценки примерно такой же, как мы использовали в детектировании объектов, то есть у нас имеется множество сегментов, мы сопоставляем сегменты друг другу и считаем критерий IOU (intersection over union) (рис. 98). Мы для всех сегментов считаем долю правильно классифицированных пикселов по отношению к комбинации правильно классифицированных, ложно позитивных срабатываний и ложноотрицательных срабатываний для каждого рассматриваемого класса объектов.

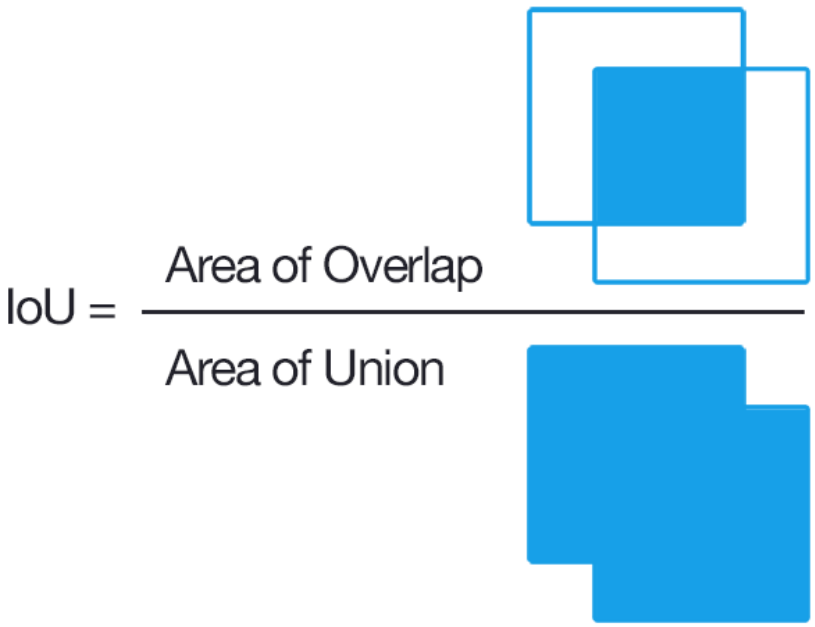


рис.98

$$segmentation\ accuracy = \frac{true\ positives}{true\ positives + false\ positives + false\ negatives}$$

Интерактивная сегментация

Интерактивная сегментация – выделения объектов, где указания поступают от пользователя.

Эта задача достаточно старая и возникла с появлением первых графических редакторов. Первыми двумя методами для сегментации изображения были *магическая кисточка* (magic wand) и *умные ножницы* (intelligent scissors), которые были реализованы в Photoshop (рис. 99).

Magic Wand – цветовая сегментация объектов по меткам пользователя. От пользователя требовалось нанести мазки кисточкой, по всем пикселам, которые попали в мазки, считалась цветовая модель пикселов объекта и по всем пикселам изображения считалась вероятность принадлежности их объекту по цвету. Добавлялась некоторая информация о связности, то есть мы присоединяли пикселы начиная от семян, если пиксел связан, то мы его присоединяем.

Intelligent scissors – проведение пути по области с высоким градиентом на основе динамического программирования через отмечаемые пользователем точки на границе. Пользователя просят указать точки на границах объекта, после чего между этими границами строилась методом динамического программирования путь минимальной стоимости. Идея стоимости бралась из градиентов, то есть, чем сильнее градиент, тем меньше стоимость проведения пути. Таким образом, мы ищем траекторию, которая проходит по наиболее сильным краям.

Понятно, что эти 2 метода учитывали разную информацию, один метод учитывал информацию похожести по цвету на объект, а второй требовал, чтобы граница проходила по сильным градиентам. Поэтому дальше все пытались найти способ, который бы позволил объединить оба этих фактора, влияющих на сегментацию объекта.

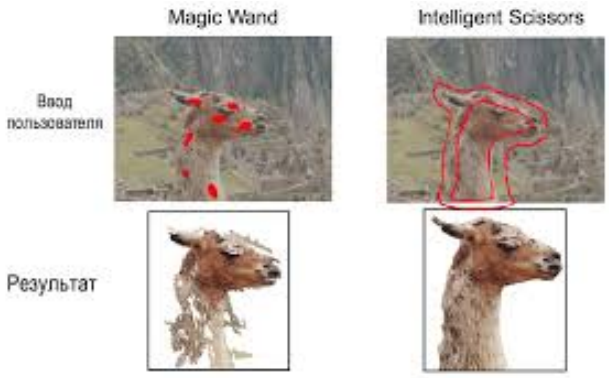


рис.99

Это удалось сделать в работе 2001 года Interactive GraphCuts методе интерактивных разрезов графов. Это основополагающая работа, поскольку она ввела аппарат из области многомерной статистики, применив его к изображениям. Дальше этот аппарат графических моделей активно использовался при анализе изображений. С его помощью большинство задач решалось наиболее точно до появления нейросетевых методов.

В методе интерактивных разрезов графов (рис. 100) в качестве интерфейса был использован метод, похожий на магическую кисточку: от пользователя просили нанести мазки, но в этот раз кисточек было 2 типа: одна кисточка помечала пиксели объекта, а вторая – пиксели фона для того, чтобы можно было построить цветовую модель пикселей объекта и некоторую цветовую модель пикселов фона. При этом также учитывалась информация о границах. Для того, чтобы учесть и то, и другое, нужно сформулировать задачу таким образом, чтобы не оценивать каждую метку по отдельности (независимо от других), а высчитывать совокупность всех меток для всего изображения в целом. То есть нам нужно выписать какой-то целевой функционал, который описывает качество сегментации с учетом всех факторов, которые на нее влияют, и найти минимум этого функционала, то есть какая разметка соответствует минимуму этого функционала.

Для формализации задачи используется аппарат графических моделей. Обозначим:

$x_i -$ множемство всех пикселей (наблюдаемые переменные)

$y_i -$ метки для всех пикселей x_i

$y_i \in [0; 1], i \in [1; N]$

Сформулируем задачу, если мы представляем ее в вероятностном виде, как поиск аргумента по всему множеству меток у такому, что достигается максимум условной вероятности всех переменных y_i по всем наблюдаемым переменным x_i. Эта некоторая совместная вероятность распределения известных переменных у предполагает, что между переменными у существуют какие-то зависимости, которые нужно учитывать, оставив задачу вычислимой на практике.

$$\boldsymbol{Y = \arg\max P(Y|X)}$$

Но между какими переменными есть зависимость? Для того, чтобы задать структуру зависимости, используется граф такой, что каждой вершине графа соответствуют скрытые переменные, которые нужно оценить, а ребро в графе присутствует в том случае, если мы говорим, что между данными 2 переменными есть непосредственная вероятностная зависимость. Есть 2 типичные структуры: полносвязный граф и решетка.

Полносвязный граф: мы предполагаем, что метка каждого пикселя зависит от меток всех остальных пикселов непосредственно.

Решетка: мы считаем, что метка каждого пиксела зависит только от меток ее соседей. То есть если мы знаем метки соседей, то знать метки всех остальных пикселов нет необходимости.

Формально мы строим граф G = (V,E), V- вершины, E – ребра такой, что:

- Все переменные из Y проиндексированы вершинами V
- Ребро $E = (V_i;\ V_j)$ присутствует, если между y_i и y_j есть непосредственная вероятностная зависимость.
- Отсутствие ребра между y_i и y_j означает, что они условно независимы, то есть: $P(y_i, y_j|Y_{\{i,j\}}) = P(y_i|Y_{\{i,j\}})\ P(y_j|Y_{\{i,j\}})$ – марковское свойство

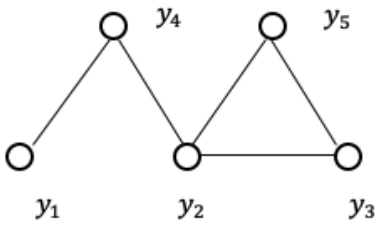


рис.100

Пару (Y, G) называют графической моделью, и поскольку для нее выполняется свойство Марковости, то такую модель называют «Марковским случайным полем» (МСП).

Теорема Хаммерсли-Клиффорда

Совместное распределение P(Y) для Марковского случайного поля (МСП) можно факторизовать в произведение неотрицательных функций φ_i (называемых «факторами») определенных на максимальных кликах D_i графа G.

$$P(y) = \frac{1}{Z}\prod_i \Phi_i(D_i)$$

$Z = \prod_i \Phi_i(D_i)$ – нормализующая константа (partition function)

$\Phi_i(D_i) > 0$

$P(Y) = \frac{1}{Z}\Phi(y_1, y_4)\ \Phi(y_2, y_4)\Phi(y_2, y_3, y_5)$ - (рис.100)

Мы можем каждый фактор, поскольку он является неотрицательной функцией, представить в виде экспоненты со степень «минус некоторая функция ε набора переменных D».

$\Phi(D) = \exp(-\varepsilon(D))$

$\varepsilon(D) = -ln\ \Phi(D)$ – функция энергии (energy function) или потенциал (potential)

Это название пришло из статистической физики, в которой обычно вероятность состояния системы обратно пропорциональна энергии (чем меньше значение энергии, тем более вероятно данное состояние, так как каждое состояние стремится к минимуму своей потенциальной энергии).

Теперь можем записать P(Y) в логарифмическом виде:

$$P(Y) \propto \exp\left(-\sum_{i=1}^{m} \varepsilon_i(D_i)\right)$$

Поскольку мы хотим найти наиболее вероятную конфигурацию системы, то есть такое распределение меток, при котором совместная вероятность наибольшая, то это эквивалентно нахождению такого значения функции энергии, которое минимально. Поиск максимума совместного распределения эквивалентен поиску минимума суммы потенциальных функций.

$$\arg\max P(Y) = \arg\max \exp\left(-\sum_{i=1}^{m} \varepsilon_i(D_i)\right) = \arg\min \sum_{i=1}^{m} \varepsilon_i(D_i)$$

В обычных марковских случайных полях рассматриваются только ненаблюдаемые переменные Y, а мы хотим найти наиболее вероятную конфигурацию наших меток скрытых переменных при условии наблюдаемых пикселов. Для того, чтобы это сделать формализм графических моделей применяется не для расписывания безусловной вероятности, а для того, чтобы расписать совместную вероятность: распределение всех переменных Y, обусловленных всеми наблюдаемыми переменными X. В этом случае такой формализм называют *условным случайным полем.*

Условным случайным полем (conditional random field, CRF) назовем (Y, X, G), в котором вершины G соответствуют Y, граф аннотирован факторами $\Phi_i(D_i)$, обусловленных наблюдаемыми переменными X

$$P(Y|X) = \frac{1}{Z}\prod_{i} \Phi_i\,(Y_i|X)$$

Применив этот формализм, получим, что совместное распределение всех скрытых переменных при условии наблюдаемых переменных представляется в виде произведения факторов, каждый из которых обусловлен всеми наблюдаемыми переменными.

Плюсом модели условного случайного поля является то, что нам нужно моделировать зависимости только между скрытыми переменными. Мы можем использовать большой набор наблюдаемых переменных, зависимости между которыми сложны или вообще непонятны. Поэтому в большинстве случаев используется или подразумевается именно CRF.

Каждому пикселу изображения задаем метку y, которая зависит только от своих соседей.

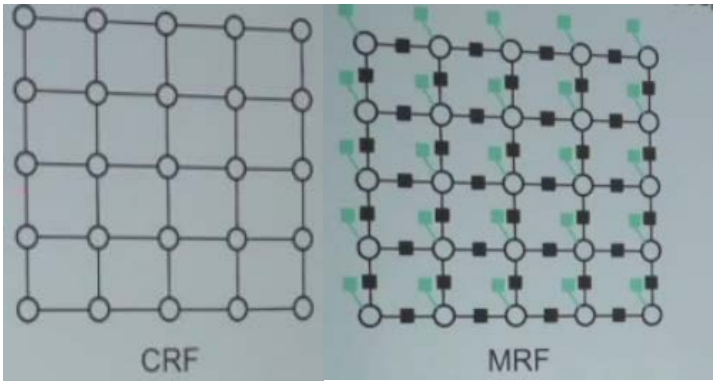


рис.101

Связи мы задали между соседними пикселами: их можно задать просто между соседними по горизонтали и вертикали, а можно задать по диагонали. Благодаря формализму мы можем переписать условную совместную вероятность всех переменных y, обусловленных x, через произведение попарных факторов, то есть факторов, зависящих от двух переменных по каждому ребру от всех остальных переменных.

$$P(Y,X) = \frac{1}{Z}\prod_i \Phi_i(y_i,x_i)\prod_{(i,j)\in N}\Psi_{i,j}(y_i,y_j) - \text{MRF}$$

$$P(Y|X) = \prod_{(i,j)\in N} A_{i,j}(y_i,y_j|X) = \frac{1}{Z}\prod_i \Phi_i(y_i|X)\prod_{(i,j)\in N}\Psi_{i,j}(y_i,y_j|X) \text{ - CRF}$$

Поскольку нам зачастую удобнее выделить отдельно зависимость каждой метки пиксела от свойств изображения, то выделим из попарных факторов факторы, задающие зависимость только отдельной переменной. То есть представим, что совместная вероятность всех тех переменных задана как произведение факторов (или штрафов), которые задают распределение отдельных переменных при изображении и факторов, которые влияют на ребра.

Факторы, которые задают зависимость только одной переменной при всем изображении называются **унарными**.

Факторы, которые связывают две переменные, называются **парными**.

Как мы можем задать эти факторы? Например, если мы решаем задачу разметки или сегментации изображений, то *унарный* фактор – это зависимость метки от окружения (изображения), мы ее можем проинтерпретировать как попиксельный классификатор, то есть классификатор, который задает метку пиксела в зависимости от окружения. Парные факторы – это штраф за сходство или различие соседних пикселов в зависимости от свойств изображения.

Предположим, что мы решаем задачу сегментации объекта (хотим выделить объект). В каких случаях штраф за сходство или различие соседних переменных должен быть маленьким, а в каких большим? Во-первых, если мы решаем задачу выделения объекта, нам нужно посчитать парный фактор для 4 конфигураций: 00, 01, 10, 11, то есть либо метки одинаковые, либо метки разные. Предположим, что у нас разные метки. Если метки одинаковые по цвету (попиксельная разница), значит, штраф должен быть большим, потому что мы проводим границу между визуально одинаковыми областями. Если по цветам эти пиксели отличаются, значит, штраф должен быть меньше.

Мы можем задать зависимости более высокого порядка, то есть взять группу пикселов и задать штраф на эту группу в совокупности (кривизну, контура, площадь фигуры итп).

Обычно используют унарные и парные потенциалы, так как для потенциалов более высоких порядков оптимизация сложнее.

$$P(Y|X) = \frac{1}{Z}\prod_i \Phi_i(y_i|X) \prod_{(i,j)\in N} \Psi_{i,j}\left(y_i, y_j|X\right)$$

$$E(Y|X) = -\sum_i \log\left(\Phi_i(y_i|X)\right) - \sum_{(i,j)\in N} \log\left(\Psi_{i,j}\left(y_i, y_j|X\right)\right)$$

Бинарная сегментация

В случае задачи бинарной сегментации мы используем обычную решетку, унарные и парные факторы. Унарный фактор задает вероятность или штрафы за принадлежность пиксела к фону.

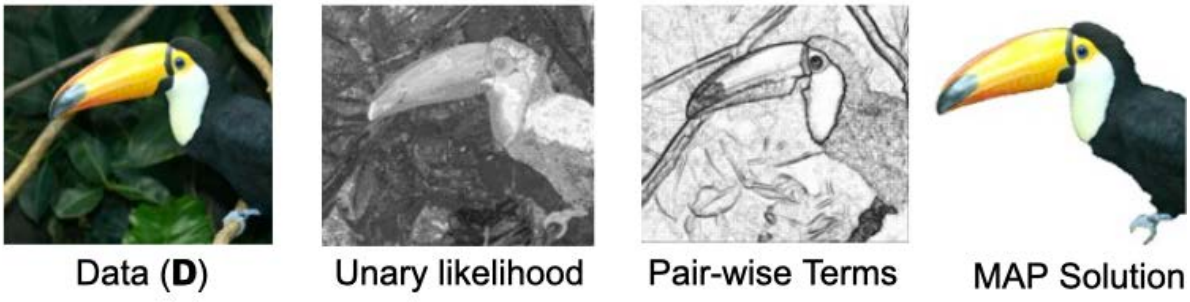


рис.102

$$\text{Energy}_{\text{MRF}} = E(\mathbf{x}) = \sum_{i \in S} \left(\phi(\mathbf{D}|\mathbf{x}_i) + \sum_{j \in N_i} (\phi(\mathbf{D}|\mathbf{x}_i, \mathbf{x}_j) + \psi(\mathbf{x}_i, \mathbf{x}_j)) \right) + \text{const}$$

Unary likelihood — Contrast Term — Uniform Prior (Potts Model)

$$\text{Maximum-a-posteriori (MAP) solution } \mathbf{x}^* = \arg\min_{\mathbf{x}} E(\mathbf{x})$$

В данном случае штраф визуализируется (рис. 102) за то, что данный пиксель принадлежит фону. Видно, что сама птица имеет достаточно большие значения, то есть если мы какой-то пиксель птицы пометим как фон, то мы какой-то пиксель птицы пометим как фон, то у нас будет достаточно большой штраф, но не везде. То есть мы посчитали цветовую модель птицы и для каждого пикселя оцениваем схожесть с ней.

Бинарный член дает штраф за градиент. Видно, что на резких границах в изображении этот штраф маленький, то есть, когда мы будем находить сегментацию, наш метод будет стремиться, чтобы граница проходила по областям, где штрафы низкие за изменением меток.

Если мы найдем минимум этой функции, то получим изображение птицы (MAP solution).

Как мы можем задавать эти штрафы?

Унарные факторы

- По похожести по цветовой модели (из попиксельных классфикаторов)
- Модели цвета считаются по отмеченным областям

$$\varphi(I|"obj") = -ln\, P(I_p|O)$$

$$\varphi(I|"bkg") = -ln\, P(I_p|B)$$

Мы считаем штраф как «минус логарифм вероятности того, что данный пиксель принадлежит цветовой модели объекта или цветовой модели фона. Цветовые модели мы можем посчитать разным способом, например, сделать гистограмму распределения цветов или посчитать нормальное распределение.

Парные факторы

- Пропорционально контрасту и удаленности

$$P^{pq} \propto \exp\left(-\frac{\|I_p - I_q\|^2}{2\sigma^2}\right)\frac{1}{\|p-q\|^2}$$

Штрафы за разные метки мы будем вычислять пропорционально похожести пикселов по цвету, то есть чем они более похожи, тем штраф больше, и пропорционально расстоянию, то есть штрафы на диагональных связях будут меньше, чем на горизонтальных.

Чтобы найти минимум энергии, воспользуемся *аппаратом поиска минимального разреза в сети* (рассматривается в курсе теории игр).

Можно задать графы специального вида, которые называются сетью. Сетью они называются, потому что у них выделены 2 специальных вершины: исток и сток. Предполагается, что по сети мы прокачиваем некоторый набор, например, жидкости, то сеть можно проинтерпретировать как набор трубопроводов. Можно доказать, что максимальный поток (максимум жидкости, которую мы можем прокачивать через граф) равен минимальному *разрезу*.

Разрез – разбиение графа на два подграфа так, что все переменные попадают либо в один подграф, либо в другой.

Фактически, мы находим самое узкое место в сети.

Для бинарной задачи мы можем задать сеть специального вида, в которой максимальный поток эквивалентен значению энергии. Для этого сделаем следующее:

- Все парные штрафы припишем как пропускные способности ребер
- Минимальный разрез должен пройти через ребра минимальной стоимости
- Унарные факторы припишем ребрам, которые будут связывать исток со стоком
- Искомый разрез обязательно должен для каждой вершины разрезать либо ребро, которое связывает его со стоком, либо ребро, которое связывает его с истоком
- Считая разрез, мы суммируем веса всех ребер, по которым проходит разрез, то есть мы учитываем либо одно значение унарного потенциала, либо альтернативное значение унарного потенциала

Поскольку есть методы, которые позволяют находить точное значение минимального разреза, то это точное значение минимального разреза соответствует точному значению минимума энергии.

$$E(y) = \sum_{p \in \mathcal{V}} F^p y_p + \sum_{p \in \mathcal{V}} B^p (1 - y_p) + \sum_{p,q \in \mathcal{N}} |y_p - y_q|$$

GrabCut

Одной из модификаций этого метода является GrabCut, который реализован в Microsoft Office как «удаление фона» (рис.103).

рис.103

Для того, чтобы сделать этот метод практически применимым необходимо было заменить интерфейс с мазков на выделение объекта рамкой. Кроме того, необходимо было сделать метод более точным, чтобы он смог выделить автоматически контрастные объекты на фоне.

Это удалось сделать за счет более сложной модели цвета и итеративного уточнения этой модели.

Предположим, что пользователь выделил в рамку объект (рис. 103), а то, что попало внутрь рамки — это частично фон, частично объект. Если мы попробуем представить распределение цветов объекта в виде, например, одного нормального распределения, у нас ничего не получится, так как цвета фона и объекта очень отличаются. Значит, нам нужно представить это в виде смеси, то есть считать, что цвет пикселя может принадлежать одной из нескольких гауссиан (5-8). То есть мы строим более сложные цветовые модели фона и объектов переднего плана. Для этого составим выборку из всех цветов пикселов, применим к ним метод кластеризации К-средних, который нам все разобьет на несколько кластеров по цветам.

Будем использовать кластеры как цветовую модель, для которой применим метод разрезов графов и уточним контур. Уточнив контуры, пересчитаем цветовую модель объекта и цветовую модель фона и снова применим метод разрезов. Так, итеративно сойдемся.

Метод показывает хорошие результаты для контрастных объектов, однако если объекты похожи по цвету на фон (камуфляж, низкий контраст, тонкие структуры), то у нас могут быть дефекты.

Требования к сегментации

Каким условиям должны удовлетворять сегменты?

- Границы сегментов должны соответствовать границам объектов
- Сегмент должен содержаться целиком внутри объекта

- Небольшие объекты должны не быть частью сегмента, а описываться своим сегментом
- Сегмент должен быть достаточно большим, чтобы быть «информативным»
- Сегменты должны быть компактными, примерно одного размера
- Равномерно распределены по изображению
- Алгоритм должен работать быстро

Если метод удовлетворяет всем этим требованиям, то говорят, что это не просто пересегментация, а *суперпиксельная сегментация*. Эти крупные компактные фрагменты можно назвать *суперпикселями (рис. 104).*

рис.104

Задача суперпиксельной сегментации очень похожа на задачу кластеризации, поскольку мы тоже хотим разбить все множество пикселов на группы похожих по своим характеристикам пикселам так, чтобы внутри группы все пикселы были похожи, а между разными группами пикселы отличались.

Проблема применения методов кластеризации, например, метода К-средних к изображению будет заключаться в том, что нам нужно обеспечить локальность кластеров, чтобы пиксели не были разбросаны по изображению, а компактно лежали. Кроме того, применить метод К-средних ко всему изображению будет очень медленно.

Почему метод кластеризации медленный? Предположим, мы хотим выделить сегмент размером SxS (рис. 105 а). Применяя метод кластеризации, мы толкуем все пиксели одинаково, сравниваем по похожести кластер с пикселями, которые расположены на всем изображении, где угодно. Но это неверный подход, потому что пиксель, расположенный в противоположном углу, никак не может попасть в этот сегмент, потому что лежит слишком далеко. В этот сегмент могут попасть пиксели, которые лежат максимум на расстоянии s от нашего пикселя. Значит, нам нужно сравнивать

только с ближайшими кластерами размером 2Sx2S (рис. 105 b), и это будет гораздо быстрее.

Для того, чтобы суперпиксели были равномерно распределены по изображению, нужно отказаться от случайной инициализации центров классов в методе К-средних и инициализировать их, например, по решётке.

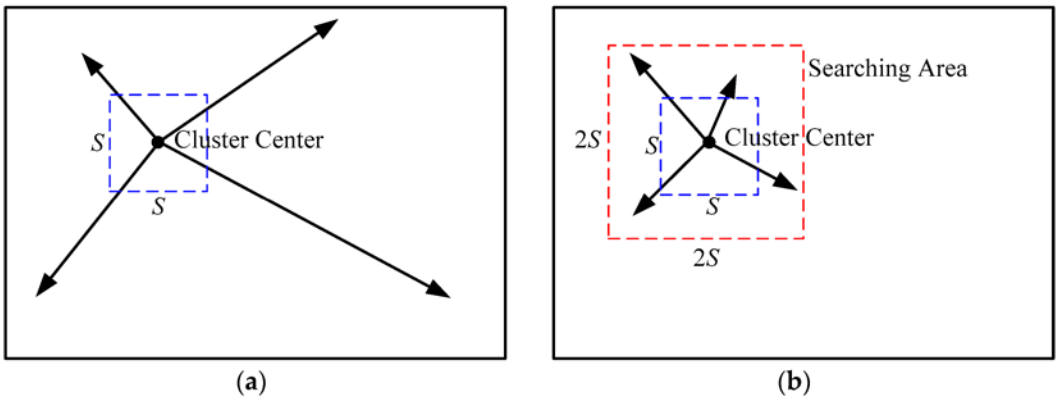


рис.105

Таким образом, получили простой линейный метод итеративной кластеризации Simple Linear Iterative Clustering (SLIC).

Устройство метода:

- Инициализируем центры кластеров C_k по сетке с шагом S
- Инициализируем в каждом шаге метку L(i) = -1 и расстояние D(i) до ближайшего кластера = -∞
- Проходим по кластерам C_k

Для каждого пиксела в области 2Sx2S считаем расстояние до C_k. Если расстояние меньше D(i), тогда ставим метку L(i) = k и записываем в D(i) новое значение.

- Повторяем до тех пор, пока суммарное изменение кластеров не будет меньше порога

Мы будем считать расстояние по цвету + пространству, то есть

- Берем какую-то цветовую модель, считаем в ней расстояние (работаем в LABCIE + (x,y) пространстве)
- Суммируем расстояние по цвету и положению и нормализуем на максимальное расстояние внутри кластера

$$d_c = \sqrt{(l_j - l_i)^2 + (a_j - a_i)^2 + (b_j - b_i)^2}$$

$$d_s = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

$$D' = \sqrt{\left(\frac{d_c}{N_c}\right)^2 + \left(\frac{d_s}{N_s}\right)^2}$$

Поскольку расстояние по цвету неизвестно, можем зафиксировать его как m

$$D' = \sqrt{\left(\frac{d_c}{m}\right)^2 + \left(\frac{d_s}{S}\right)^2} \rightarrow D = \sqrt{d_c^2 + \left(\frac{d_s}{S}\right)^2 m^2}$$

Сложность $\left(\frac{n}{s}\right)^2 S^2 t = n^2 t$ = Nt то есть время работы метода не зависит от числа кластеров, а зависит только от размера картинки.

Семантическая сегментация

До появления нейросетей наилучшие результаты в решении задачи семантической сегментации показывали обучаемые попиксельные классификаторы и аппараты условных случайных полей (CRF). То есть мы рассматриваем задачу семантической сегментации как задачу разметки графа.

Каждому пикселю задаем вероятностную переменную, метку, поверх этих вероятностных переменных мы задаем графическую модель. Эта графическая модель сочетает в себе унарные факторы и члены контраста (те же самые факторы, которые мы использовали при выделении объектов). То есть мы штрафуем за разность соседних меток, и штраф пропорционален степени схожести соседних пикселов. Унарные факторы- попиксельные классификаторы. Мы обучаем какой-то классификатор для каждого пиксела по его окрестности. Например, берем метод опорный векторов, на вход подаем SIFT признаки из небольшой окрестности, и он классифицирует пикселы. Затем все получившиеся признаки мы отдаем в графическую модель, применяем метод оптимизации и получаем решение.

Вклад CRF - сглаживание демонстрируется на рис. 106.

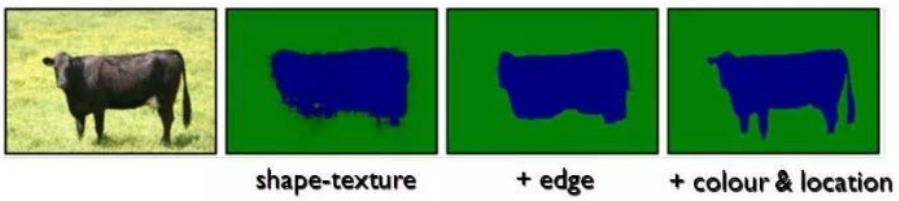


рис.106

Видно, что основной вклад в точность делают попиксельные классификаторы, а добавление случайного поля позволяет немного повысить качество и гладкость границ.

Для упрощения работы и повышения ее точности мы можем сделать следующий шаг – применить *суперпиксельную сегментацию*.

То есть разделим изображение на сегменты, зададим графическую модель поверх сегментов и обучим классификаторы, которые на вход принимают признаки сегментов. Такие методы долгое время были наиболее точными.

Точность предыдущих методов ограничивалась точностью суперпиксельной сегментации и не позволяла находить очень тонкие границы.

Для того, чтобы находить сложные границы, нам нужно учитывать вклад не только непосредственно соседей, а всех соседей по картинке или вообще всей картинки. То есть лучше сделать графическую модель не разреженной, а полносвязной, в которой каждый пиксел зависит от всех остальных пикселов. Но если все эти зависимости будут серьезными, найти минимум такой функции будет затруднительно. На помощь приходит **билатеральная фильтрация Bilateral Pairwise Term** (рис. 107).

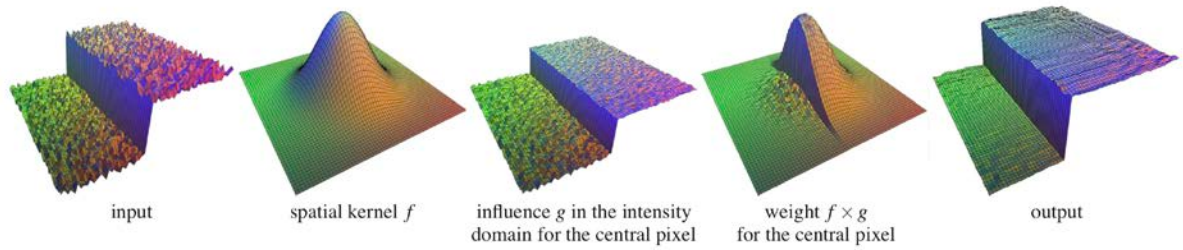


рис.107

$$\psi_{ij}(x_i, x_j) = \mu(x_i, x_j) exp\left(-\frac{|s_i - s_j|^2}{2\sigma_\alpha^2} - \frac{|c_i - c|^2}{2\sigma_\beta^2}\right)$$

Идея заключается в следующем:

Делаем парные факторы между всеми пикселами к друг другу зависимыми от *расстояния* между пикселами (для этого возьмем гауссово ядро и будем считать, что близко расположенные пикселы должны быть более связаны, чем далеко расположенные, то есть связанность уменьшается с увеличением расстояния) и *схожести по цветам* (если имеется два пиксела на одном и том же расстоянии, но один из них похож по цвету, а другой нет, то менее похожий пиксел будет давать меньший штраф).

За счет использования множества связей методы вывода в таких полносвязных графических моделях достигают достаточно высокой точности в семантической сегментации (рис.108).

рис.108

Нейросетевые модели для сегментации

- Применим сверточную архитектуру для классификации к изображению, VGG16
- Получим матрицу признаков (карту признаков) в 32 раза меньше исходного изображения
- Для каждого пикселя признаков применим классификатор для оценки классов (примерно, как RPN в Faster R-CNN)
- Получим карту разметки низкого разрешения
- Нужно повысить разрешение до исходного

Первая идея исключительно проста DeepLab (рис.109): воспользуемся билинейной интерполяцией и Dense CRF для повышения разрешения. Для этого сделаем следующее:

- С помощью нейросети построим карту сегментации низкого разрешения
- Сделаем линейную интерполяцию в 32 раза
- Получим размытую карту сегментации высокого разрешения
- Подадим на вход эту карту сегментации как попиксельные классификаторы для полносвязных графических моделей
- Получим точную границу (результаты рис. 110)

Первым способом, который придумали, чтобы на выходе получить карту признаков высокого разрешения, стала *транспонированная свертка Transposed Convolution.*

Обычно, делая свертку, мы уменьшаем разрешение изображения, но мы можем и увеличить разрешение картинки, расставив пиксели редко и затем интерполируя значения в промежутках с помощью свертки.

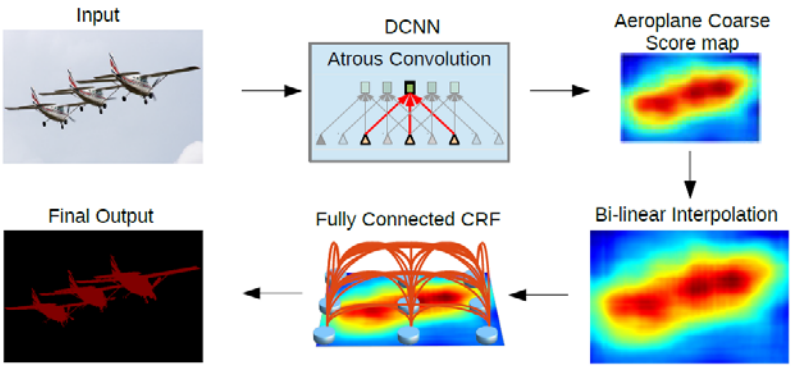


рис.109

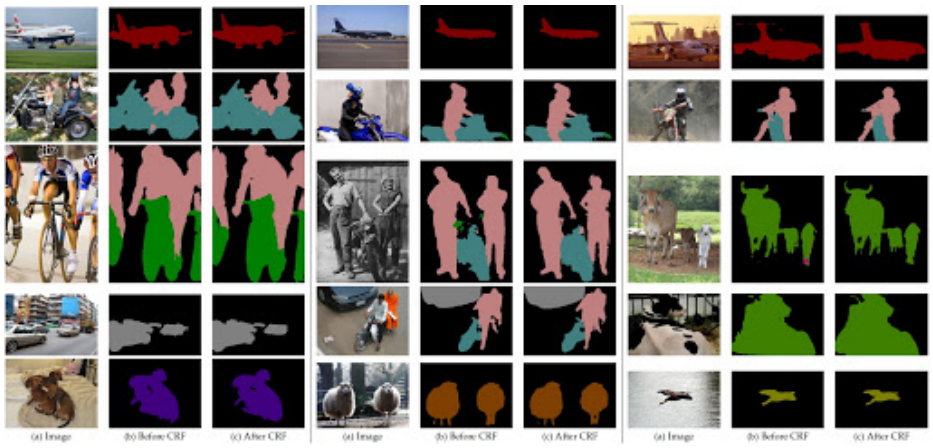


рис.110

Воспользовавшись таким методом повышения разрешения, сделали базовую архитектуру для сегментации изображений, получившею название *полносверточная сеть fully convolutional network* (рис. 111).

В этой модели повышение разрешения признаков устроено достаточно хитро. Во-первых, мы будем несколько раз предсказывать карты сегментации, после первого раза мы предскажем карту размеров в 32 раза меньше, чем исходное изображение, второй раз – в 16 раз меньше, в третий раз – в 8 раз меньше.

Чтобы предсказать карту более высокого разрешения, мы возьмем признаки более низкого уровня и применим к ним транспонированную свертку, получим карту признаков с разрешением в 2 раза больше. Затем возьмем признаки с backbone архитектуры предыдущего слоя (тоже в 16 раз меньше исходного разрешения) и просуммируем их. По этим суммированным признакам предскажем разметку в 16 раз меньше и так сделаем несколько раз. В этой базовой архитектуре было 3 этапа повышения, то есть на выходе получалась карта признаков в 8 раз меньше, чем исходное разрешение, которое потом линейно интерполировалось еще в 4 раза, получалась карта разметки в 2 раза меньше, чем исходное изображение и по ней уже сравнивалось качество работы.

Архитектура достаточно хорошая, потому что полностью состоит из сверток, а значит, очень эффективно реализовывается на процессорах, в ней нет полносвязных слоев, все работает быстро.

Как архитектуры развивались дальше?

Одним из направлений было исследование новых способов повышения разрешения карт признаков с использованием какой-то дополнительной информацией. Так получился слой max *unpooling (*рис. 112).

Вспомним, что при понижении разрешения изображения мы обычно используем max pooling, например, с шагом 2x2, то есть из каждого блока 2x2 пиксела мы сохраняем только одно максимальное значение. Запомним индекс, из которого было получено максимальное значение, поскольку нам важно знать, где именно в картинке было это максимальное значение. Если мы запомним этот индекс, тогда сможем выполнить операцию повышения разрешения, которая будет не просто делать пиксели с равномерным шагом, а заносить целевой пиксель именно туда, откуда мы взяли изначально максимальное значение. Уже после этого будем проходиться по этой карте набором сверток, которые будут интерполировать промежуточные значения.

- Сохраняем индексы каждого max pooling слоя
- При повышении разрешения копируем значения из выхода max pooling слоя с учетом запомненных индексов, применяем обученные свертки для сглаживания

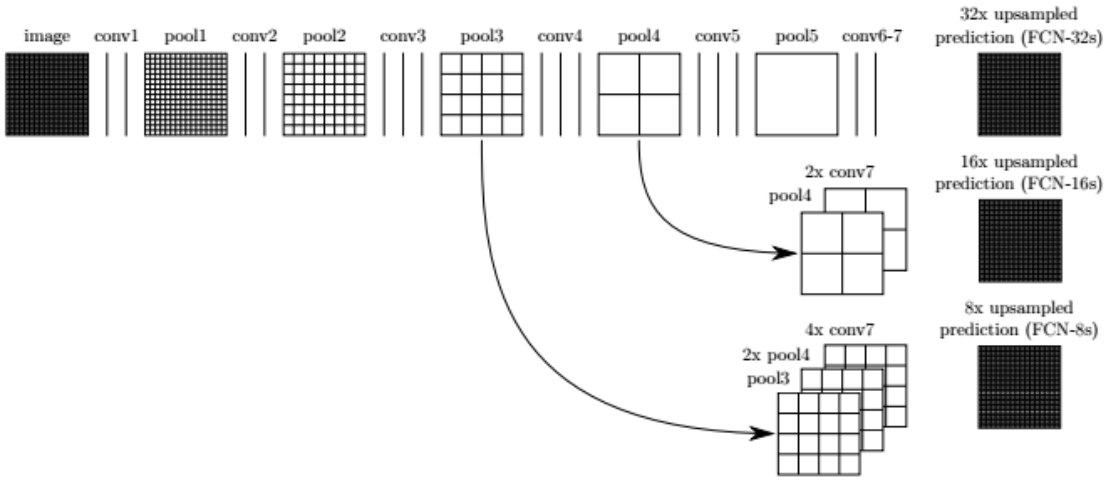


рис.111

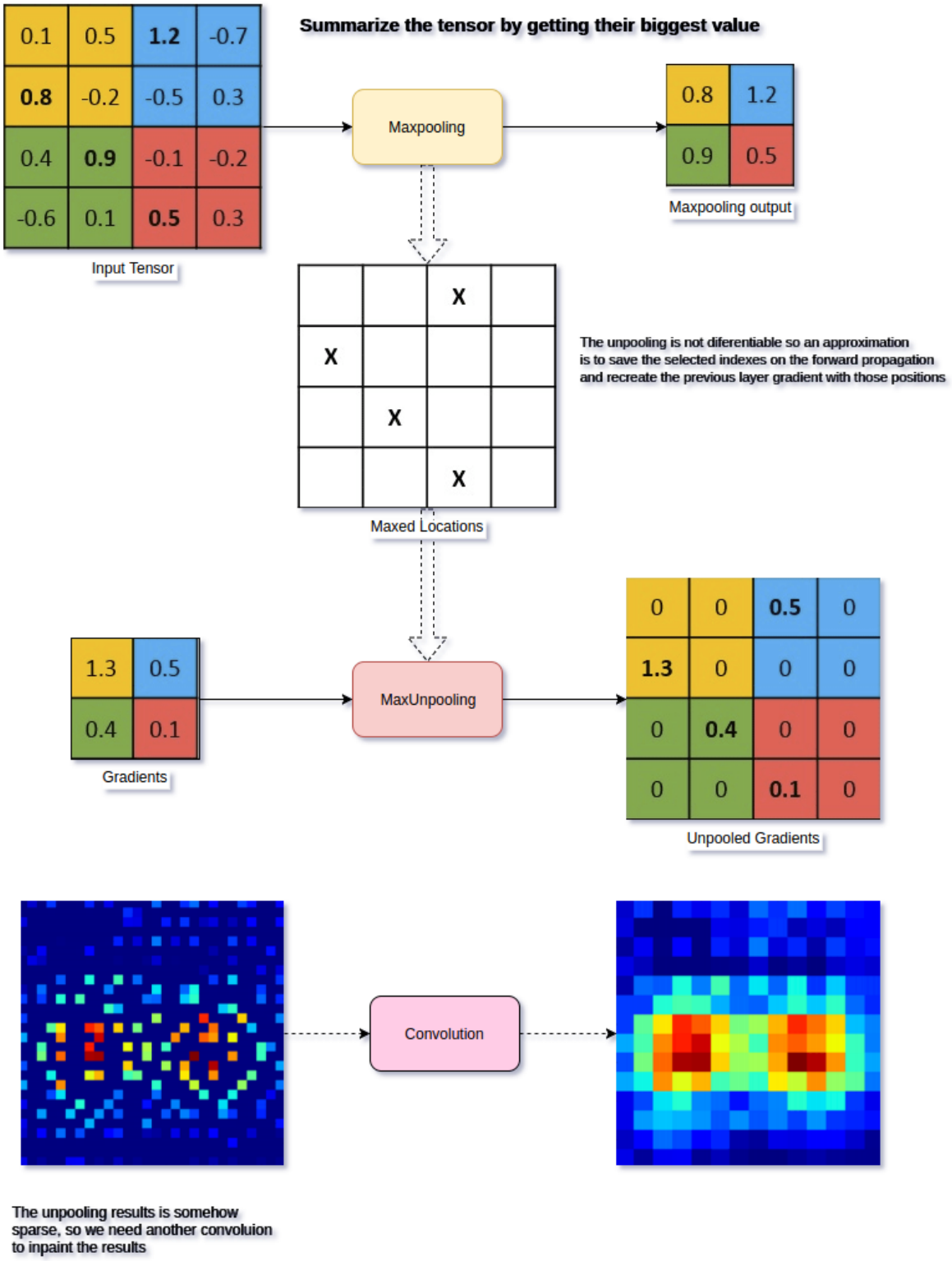


рис.112

С помощью операции unpooling появилась архитектура для сегментации изображения, фактически для декодирования признаков, *Encoder-Decoder with Max Unpooling* (рис. 113).

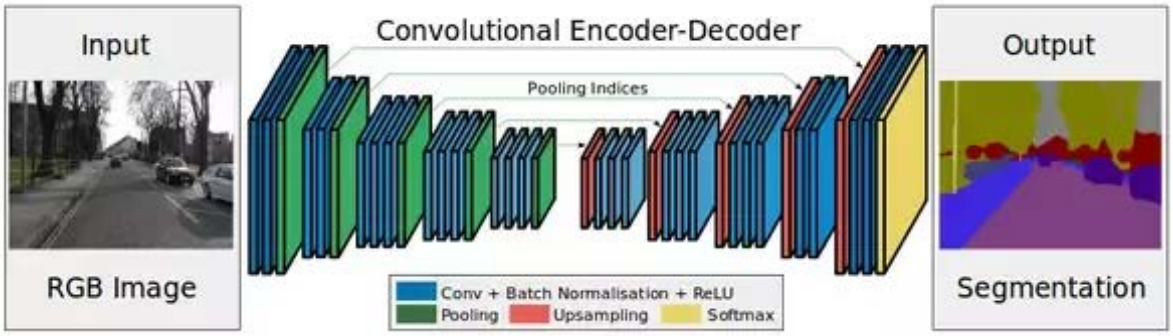


рис.113

Здесь между соответствующими словами кодировщика и декодировщика передаётся информация в виде индексов операции max-pooling, то есть не вся информация со слоев высокого разрешения теряется, она сохраняется и передается на декодировщик. Эта архитектура оказалась точнее предыдущей.

В одной из работ эту архитектуру довели до абсолюта, то есть построили такой кодировщик, который превращает картинку в вектор 1x1 пиксел (из картинки сохраняется только контекст). Используя сохраненные max-pooling индексы, повышаем разрешение до исходного, предсказывая карту разметки. Как показали эксперименты, контекст вообще можно проигнорировать и заменить вектор на константный, так как в индексах операции max-pooling хранится достаточно информации для того, чтобы предсказать карту сегментации для исходного изображения. Поскольку свертки связаны с визуальными образами, информации о том, где свертка дала максимальное значение, достаточно, чтобы детектировать объект.

Концептуальное ограничение этой архитектуры заключается в следующем: между одинаковыми слоями кодировщика и декодировщика передается только очень ограниченная информация в виде индексов операции max-pooling. Вместо этого мы можем передавать всю информацию, все нейросетевые признаки соответствующего уровня. Таким образом, мы получим *архитектуру U-Net* (рис. 114), которая сейчас является стандартной архитектурой для сегментации изображений.

Архитектура U-Net похожа на кодировщик-декодировщик, только на каждом этапе повышения разрешения происходит следующее: мы повышаем разрешение признаков предыдущего уровня путем конкатенации признаков повышенного разрешения соответствующего уровня кодировщика и затем прогоняем на них несколько сверток. При этом для повышения разрешения достаточно сделать билинейную интерполяцию в 2 раза, то есть просто растянуть карту признаков в 2 раза, конкатенировать ее с картой признаков более высокого разрешения и преобразовать.

Первой модификацией U-Net стали «песочные часы» Hourglass. Авторы этой модификации (рис. 115).

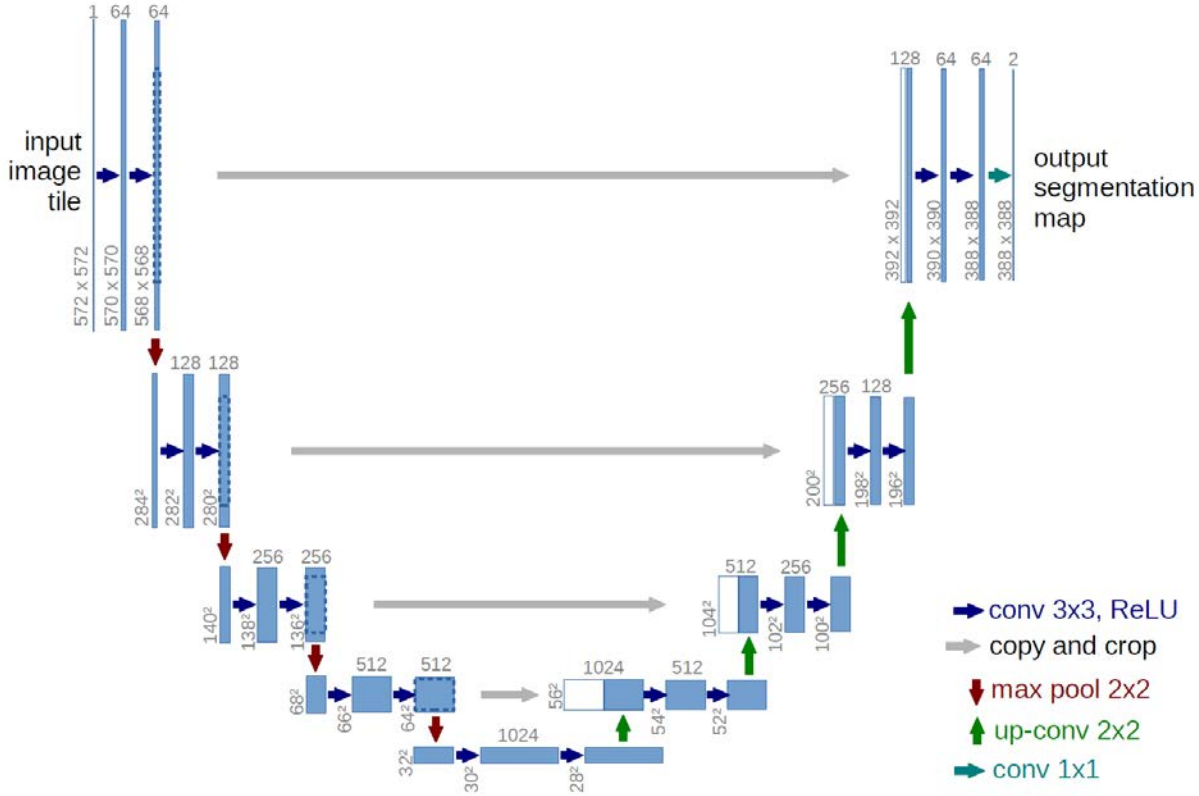


рис.114

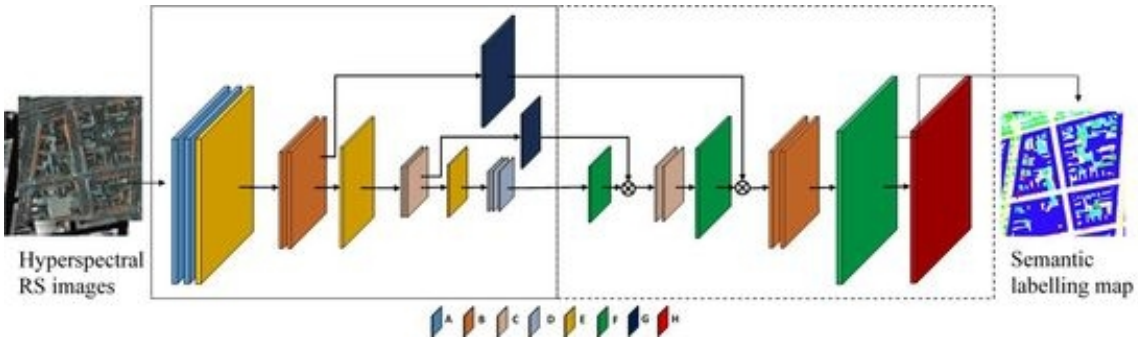


рис.115

Что изменилось в этой архитектуре:

- Появились модули преобразования, residual модули на skip-connections

- Объединение модулей в цепочку (каскад) (рис. 116): первый модуль строит карту сегментации первого уровня, второй модуль будет учить добавку (поправку) к первой сегментации. Далее суммируем их выходы и получаем результат (стек песочных часов).

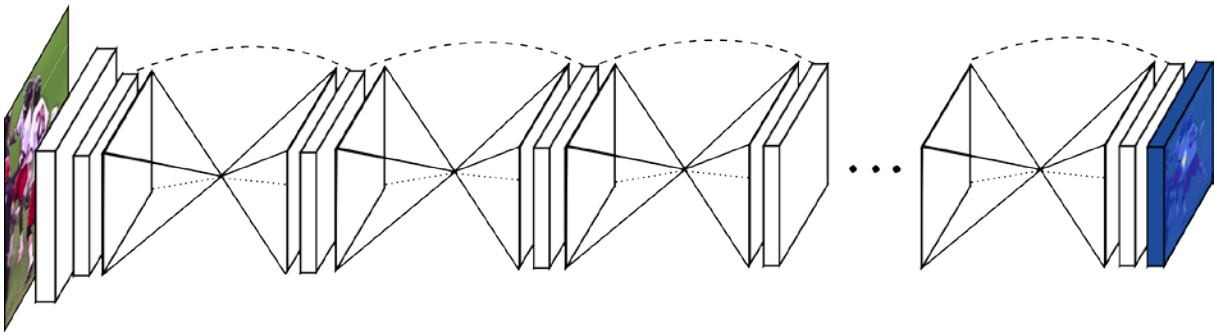


рис.116

RefineNet (рис. 117)

Архитектура похожая на U-Net, но с некоторыми модификациями. Как в архитектуре *песочные часы*, при повышении разрешения преобразуем признаки через несколько блоков, но каждый их этих блоков устроен более сложно. Здесь мы не конкатенируем, а суммируем, как в предыдущих архитектурах.

RefineNet достаточно хороша по точности, но медленная из-за сложности и многомодульности архитектуры.Вскоре появилась новая версия Light RefineNet (рис. 118), в которой авторы провели эксперимент и заменили свертки 3x3 на свертки 1x1, тем самым увеличив скорость работы, не потеряв точность.

Кое-что еще про сегментацию

Как нейросети можно применить для интерактивной сегментации?

Нужно добавить каким-то образом информацию от пользователя на вход нейросети. Одной из таких идей стал Deep GrabCut. (рис. 118). На вход помимо изображения подается дополнительно карта, на которой учтена информация об объекте, выделенного пользователем. Об объекте можно учесть информацию в виде *карты дистантного преобразования*.

Карта дистантного преобразования – карта, в каждом пикселе которой записано расстояние до некоторого контура. В данном случае под контуром понимается ограничивающий прямоугольник объекта, соответственно в каждый пиксель мы записываем расстояние до этого контура. Далее прогоняем это через обычную

архитектуру кодировщик-декодировщик сети и получаем маску. Обучать такую архитектуру мы можем практически на любых данных с объектами. Таким образом, получился метод, который по точности обогнал GrabCut.

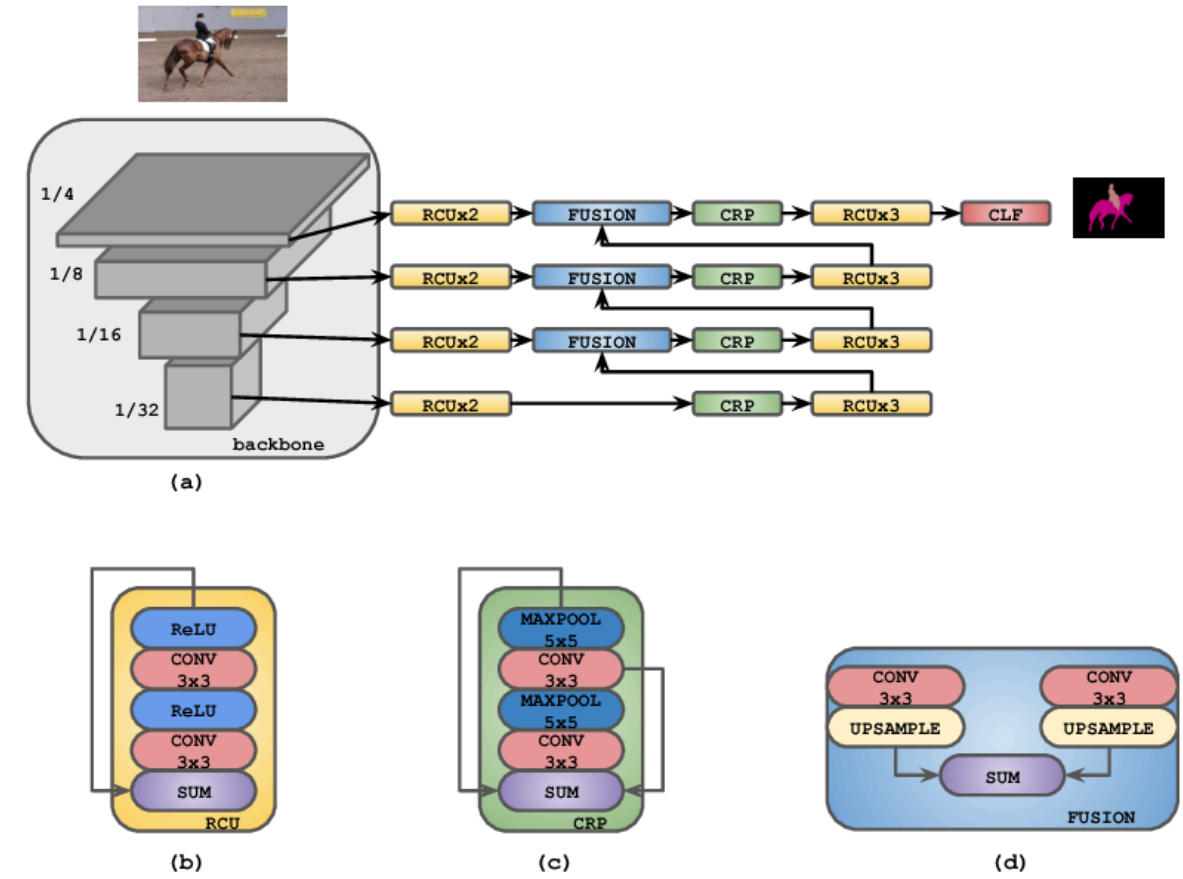


рис.117

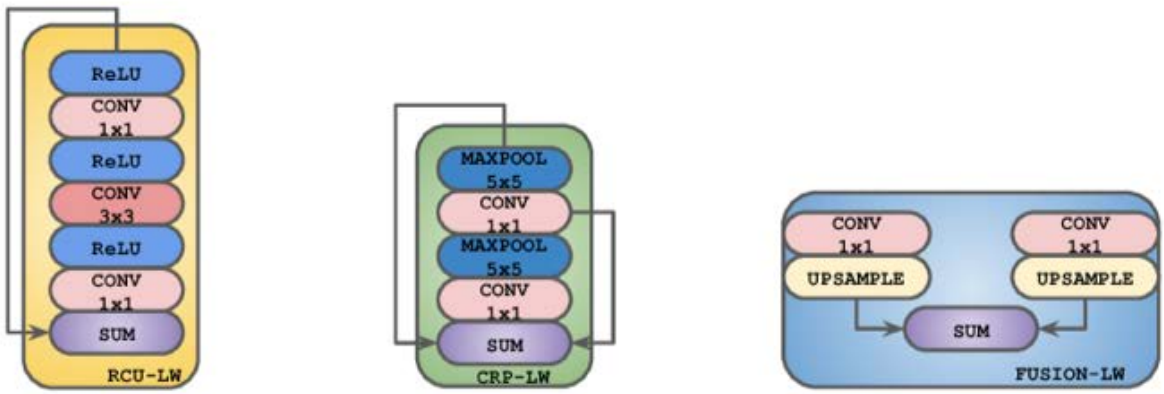


рис.118

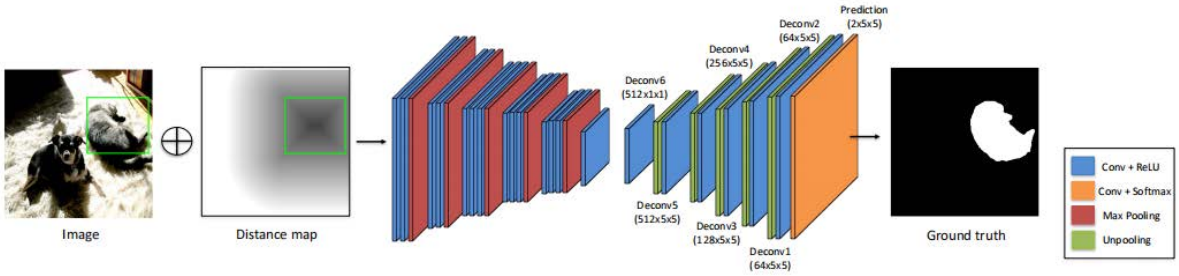


рис.119

С помощью полносверточных сетей для сегментации изображений мы можем также решить задачу *instance segmentation*. Для этого добавим декодировщики для вычисления mask-объектов к обычной архитектуре Faster R-CNN для детектирования объектов. Нам нужно получить маску объекта, а для этого сначала нужно найти объект и примениить сегментационную сеть точно к той области изображения, которая соответсвует объекту. Для этого воспользуемся слоями ROI-pooling и ROI-Align. То есть мы

- Применяем Faster R-CNN для детекции объектов
- Для найденных объектов с помощью ROI-pooling или ROI-Align извлекаем карту признаков низкого разрешения
- Подаем на вход декодировщику эту карту признаков, который строит карту сегментации более высокого разрешения

Такая архитектура получила название Mask R-CNN (рис. 120) и сейчас является основной архитектурой для *instance segmentation.*

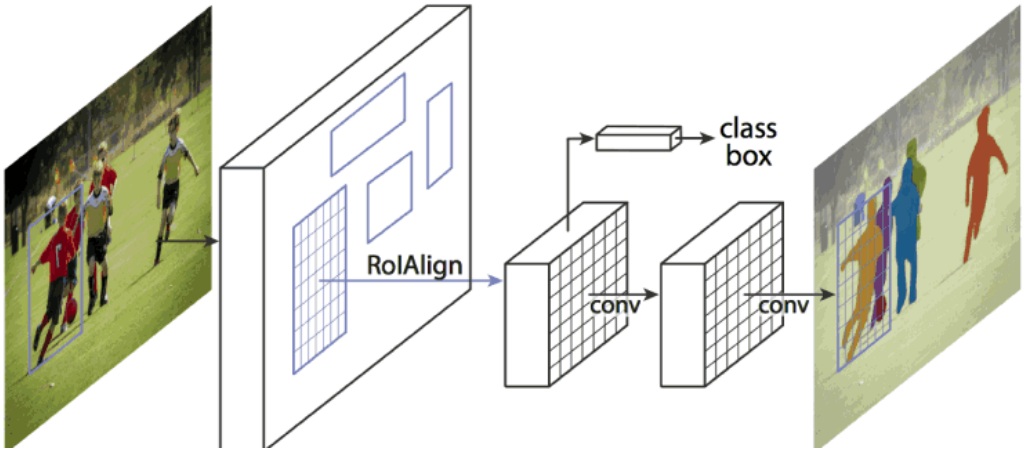


рис.120

С помощью задачи сегментации мы можем решать задачу нахождения ключевых точек объекта, например, задачу нахождения ключевых точек на скелете человека, определение позы. Для этого нам нужно локализовать точки. Предположим, что окрестность каждой точки — это определенный класс объектов. Все пикселы, которые находятся на небольшом расстоянии от целевой точки, считаются как правильно классифицированные. Сделаем разметку изображения: вокруг каждой точки зададим окрестность (кружок) (рис. 121), например, в 3 пиксела и обучим метод многоклассовой сегментации, который будет находить эти кружки на изображении. На выходе будет выдаваться карта вероятности, в которой мы найдем локальные максимумы и возьмем их как центры ключевых точек на объекте.

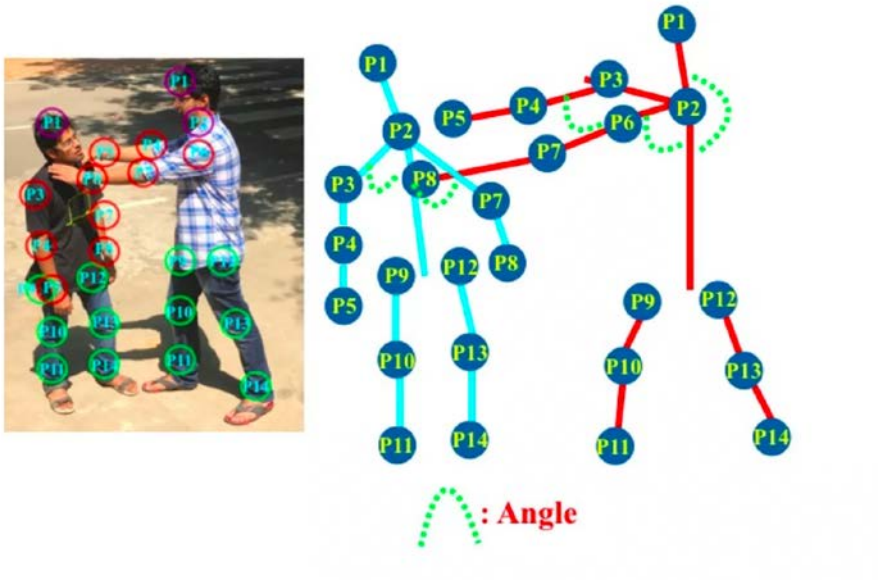


рис.121

Сейчас в основном сегментацию рассматривают как одну из задач для решения множества других задач.

Лекция 8. Перенос стиля и синтез изображений

Модификация изображений

Мы хотим сделать некоторое преобразование изображения, связанное с изменением некоторых характеристик изображения, но семантических, например, изменить возраст человека (рис. 122). Подобные задачи сейчас решаются с помощью нейросетей вида кодировщик-декодировщик. То есть модифицируем сети, которые мы рассматривали для семантической сегментации, чтобы на выходе они давали не карту сегментации, а RGB изображение и меняли некоторые признаки исходного изображения.

Другой вариант модификации изображений – это их *стилизаци*я (рис.123). В этом случае изменяется не семантическое свойство изображения, а стиль отрисовки.

рис.122

рис.123

Третий класс задач, которые мы рассматриваем, — это *синтез изображений.* Задача заключается в создании с нуля определенного класса объектов, например, лица человека. Задачи синтеза изображений сейчас решаются сетями (например, DCGAN) (рис. 124), которые выглядят как половина сети кодировщик-декодировщик. Фактически это декодировщик, который на вход берет некое сжатое представление (некий случайный вектор). По этому случайному вектору через цепочки преобразований мы получаем RGB изображение.

Начнем рассмотрение методов с визуализации вектор-признаков. Давайте попробуем синтезировать изображения, которые дают тот же самый вектор-признак, что и заданное изображение. То есть посмотрим, какие еще изображения с точки зрения нейросети похожи на данное.

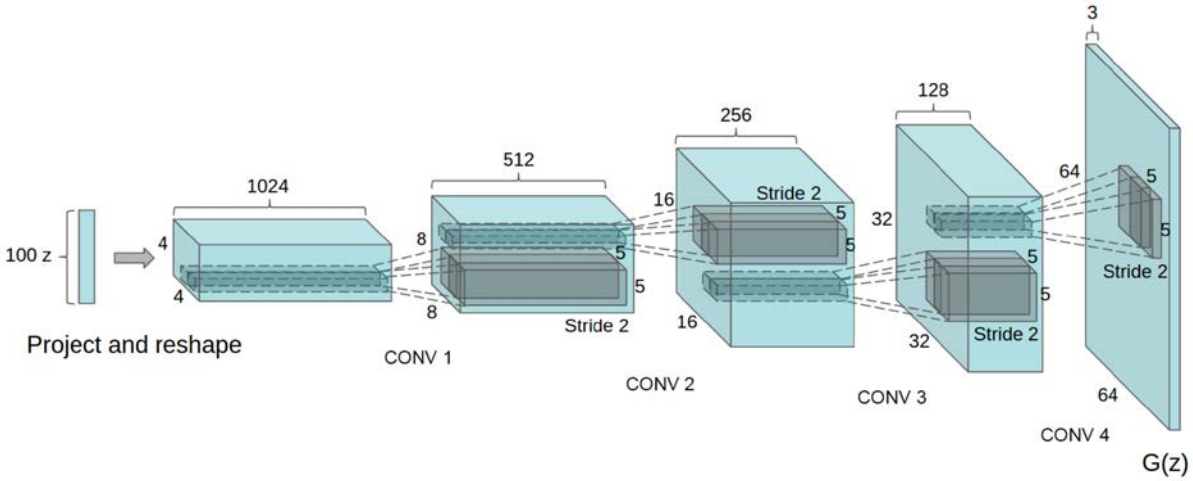


рис.124

Найдем такое изображение x, которое дает такой же вектор-признак $\Phi_0 = \Phi(x_0)$ от изображения x_0.

Задачу будем решать следующим образом:

- Инициализируем изображение x белым шумом
- Будем оптимизировать по x следующий функционал l:

  $x^* = argmin\, l(\Phi(x), \Phi_0) + \lambda\mathcal{R}(x), \quad x \in \mathbb{R}^{H\times W\times C}$

  $l(\Phi(x), \Phi_0) = \|\Phi(x) - \Phi_0\|^2$

  $\mathcal{R}(x)$ – регуляризатор, например, Total Variation (TV):

$$\mathcal{R}(x) = \sum_{i,j}((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j-}x_{i,j})^2)$$

- Оптимизируем функционал градиентным спуском

Мы минимизируем сумму двух величин: первая величина – это расстояние между соответсвующими нейросетевыми признаками (то есть прогоняем изображение через нейросеть и получаем вектор-признак $\Phi(x)$, сравниваем его с вектор признаком исходно зданного изображения Φ_0, например, по среднеквадратичному расстоянию) , вторая – регуляризатор.

Мы можем в рамках эксперимента ркеконструировать изображение, опираясь на разные слои изображения. С неглубоких слоев картинка реконструируется очень точно, и чем глубже сверточный слой, тем менее четким становится изображение, но общая пространственная конфигурация сохраняется. Когда мы переходим к полносвязным слоям, пространственное распределение нарушается, но все равно сохраняется образ, который отдаленно напоминает исходный объект.

Таким образом, если мы пытаемся реконструировать изображение, опираясь на сверточные признаки, в которых сохраняетяс пространственная информация, то получившиеся картинки похожи друг на друга. А если мы начинаем реконструкцию с полносвязных слоев, то пространственная конфигурация получается разная, н6о образы объектов все равно сохраняются.

Такимс способом была решена задача стилизация. Базовые методы переноса цвета работают со свойствами все картинки в целом. Например, приравнивание среднего и дисперсии по каждому каналу Lab.

Основная идея стилизации – модификация изображения таким образом, чтобы по чатси признаков оно было похоже (содержание) на сиходное, а по стилю на целевое. Для того, чтобы это сделать нужно определиться, как передавать содержание изображения и как передавать стиль.

Выходы нейросетевых признаков нейросети являются хорошим описателем содержания изображения. То есть если мы синтезируем изображение, которое будет похоже по своим нейросетвым признакам на целевое изображение, значит,содержание сохранилось.

Несколько лет назад экспериментальным путем было выявлено, что метрика, которая задается в виде близости нейросетевых признаков очень хорошо соответсвует близости с точки зрения человеческого восприятия. Это гораздо точнее, чем в случае попиксельных метрик.

Работа Perception Loss (ошибкая восприятия) основана как раз на сравнение нейросетевых признаков.В ней людям показывалось множество троек изображений, где было некоторое центральное изображение и 2 модифицированных. Их просили оценить, какое из изображений больше всего похоже на центральное. К изображениям применялись разные преобразования: разымытие, смещение цветовых каналов. Оказалось, что попиксельные метрикик считали более похожими размытые изображения с разнесением каналов, а человек и нейросеть говорили, что если к изображению применены просто некоторые деформации или цветовые преобразования, но содержание осталось неизменным, то это меньшее изменение, чем применение простых фильтров.

Сейчас схожесть изображений стандартно оценивают через схожесть нейросетевыхз признаков.

Как описать стиль изображения?

- За описание стиля можно взять корреляцию откликов разных фильтров по всему изображению
- Можно вычислить по признакам первых слоев стиль и попробовать реконструировать изображение с тем же стилем

Стиль можно описать корреляцией откликов фильтров, записав матрицу Грама $G^I \in \mathcal{R}^{N_I \times N_I}$

G^I_{ij} вычисляется как скалярное произведение откликов i-го и j-го фильтров:

$G^I_{ij} = \sum_k F^I_{ik} F^I_{jk}$

Генерируем изображения со стилем исходного изображения, минимизируя среднеквадратичную разницу между матрицами Грама исходного изображения G и сгенерированного изображения A (или суммами по L первых слоев).

$$E_l = \frac{1}{Norm} \sum_{i,j} (G^l_{i,j} - A^l_{i,j})^2$$

Другими словами, возьмем выход какого-то слоя нейросети – это трехмерная матрица, где по третьему измерению каждый канал – это карта отклика определенного фильтра. Мы можем каждый канал вытянуть в вектор. Соответственно корреляция откликов друг с другом будет матрицей, которая называется матрица Грама. Значение матрицы Грама можно считать как признак стиля изображения. При этом для сравнения схоести дух изображений по стилю мы можем сравнивать матрицы Грама двух изображений не по одному слою, а по нескольким слоям. То есть мы

- Берем изображение и прогоняем его через нейросеть
- Вычисляем на каждом слое матрицу Грама
- То же самое делаем для другого изображения
- Считаем близости матриц Грама, например, как сумму квадратов разницы соответсвующих элементов матрицы
- Суммируем по всем слоям

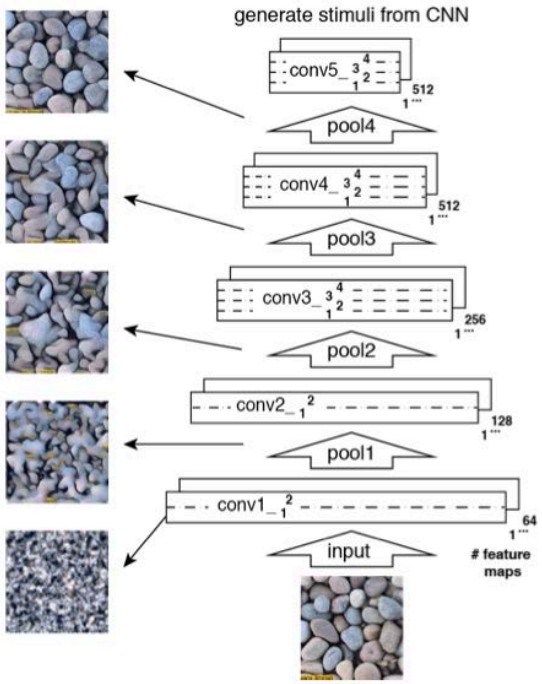


рис.125

У нас есть функция сравнения двух изображений по стилю и метод реконструкции изображения с заданными характеристиками, то есть мы можем взять шумное изображение и методом градиентного спуска модифицировать его до тех пор, пока оно по своим признакам не станет похожим на другое изображение. То же самое мы можем применить для похожести изображения по стилю.

Возьмем изображение, посчитаем для него матрицы Грама, начнем реконструировать изображение, которое по матрицам Грама будет похоже на заданное изображение. Таким способом мы можем генерировать текстуры (рис. 125).

Возьмем изображение, которое целиком является текстурой (состоит из хаотически повторяющихся элементов, например, камушков). Далее берем выходы первого сверточного слоя, считаем матрицу Грама и реконструируем изображение, которое по выходам первого сверточного слоя было похоже на целевое. Получили текстуру. Затем сравниваем по сумме матриц Грама, полученных после двух слоев: после выхода первого сверточного слоя и после выхода max-pooling. Когда мы пройдем по изображению вглубь до 4 сверточного слоя и реконструируем изображение, у нас реконструируется изображение текстуры очень похожее на исходное, но отличающееся пространственным распределением элементов текстуры. Таким образом мы можем генерировать произвольное количество текстур, если на вход подаем изображение с хаотическими элементами.

В 2015 году была опубликована работа, ключевым моментом которой было наблюдение: в оптимизационном процессе мы можем смешать похожесть по содержанию и похожесть по стилю, то есть содержание и стиль с точки зрения нейросетевых признаков оказываются разделимыми. Верхние слои больше описывают содержание изображения, корреляция нижних слоев – стиль изображения.

Мы будем генерировать изображение, начиная со случайного приближения, оптимизируя взвешенную комбинацию похожести по содержанию на одно изображение, то есть прямое сравнение вектор-признаков, и похожести по стилю на второе изображение (близость матриц Грама). Оказалось, что через 100 итераций градиентного спуска мы получим изображение хорошего качества, которое действительно сочетает в себе содержание одного изображения и стиль другого.

Поскольку у нас целевой функционал – взвешенная линейная комбинация, регулируя относительные веса стиля по отношению к содержанию, мы можем сказать алгоритму, чего в итоговом изображении должно быть больше: содержания или стиля. Видно (рис. 126), что, увеличивая вес стиля, мы все больше стилизуем изображение, пока наконец не начнем генерировать текстуру.

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

$$Decrease\ {}^{\alpha}/_{\beta}$$

С помощью таких оптимизационных процессов мы можем сделать необычные методы модификации изображения, что было опубликовано в работе по интерполяции глубоких признаков для модификации содержания изображения.

Авторы отталкивались от следующего предположения:

рис.126

- Глубокие нейросети показывают отличные результаты на image classification, используя простые линейные классификаторы
- Следовательно признаковое пространство получается очень хорошее
- Может быть, даже евклидово, тогда мы бы могли применить аппарат линейной алгебры к нейросетевым признакам?
- Можно работать напрямую в признаковом пространстве

Так и сделаем. Предположим, что у нас есть большая коллекция аннотированных изображений одного и того же класса, например, лица людей (рис. 127). Мы проаннотировали эти лица разными атрибутами: личность людей, наличие усов, бороды, этническое происхождение и т.п. Предположим, у нас есть фотография человека, атрибут которого мы хотим изменить, например, из фотографии человека без усов мы хотим синтезировать фотографию человека с усами. Для этого найдем несколько похожих на него людей. Похожесть будем оценивать, сравнивая нейросетевые признаки, но среди людей, у которых все множество атрибутов то же самое, то есть если мы берем фото Роберата Дауни Младшего, значит, нам нужны фотографии белых мужчин, без очков, без усов и т.д. Для того, чтобы определить, что такое «усатость» найдём несколько людей, которые тоже похожи на Роберта, но отличаются одним атрибутом – наличием усов. Тогда разница между в среднем безусыми и в среднем усатыми будет вектор усатости. Соответственно сдвинем вектор-

признак Роберта в направлении усатых людей и реконструируем изображение, которое соответствует этой модифицированной версии. Так Роберт Дауни Младший станет усатым.

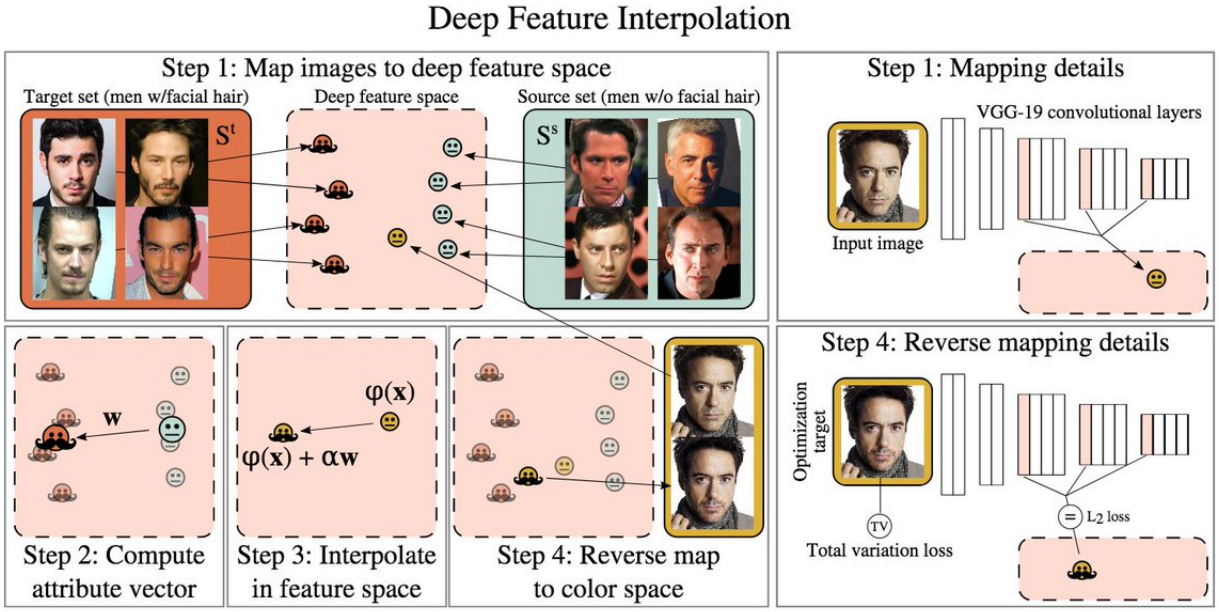


Figure 2. A schematic outline of the four high-level DFI steps.

рис.127

Поскольку мы сдвигаем человека в направлении атрибута, то в зависимости от того, насколько мы его сдвинем, мы можем промоделировать степень изменения атрибута.

Разумеется, для того, чтобы так можно было манипулировать изображениями пришлось устранить другие факторы изменчивости, например, перед преобразование все изображения выравниваются, то есть мы совмещаем их друг с другом таким образом, чтобы глаза были в одних и тех же местах итп, потому что мы хотим, чтобы разница в признаках соответствовала не разнице в пространственной информации, а только в разнице атрибутов внешности.

Для сравнения двух изображений в этой работе мы использовали сеть, которая обучена в режиме классификации не людей, а всевозможных изображений, а потом применена для сравнения людей. Если мы заменим в данном алгоритме сеть, обученную на базе ImageNet, на сеть, обученную для распознавания людей, то результаты станут гораздо хуже, потому что сеть, которая учится распознавать людей, учится игнорировать различные небольшие изменения внешности, сохраняя только личность.

Алгоритм стилизации изображений на основе градиентного спуска привлек массу внимания, поскольку это был первый метод, который позволил стилизовать

произвольные изображения под картины художников. Но у него был один серьезный недостаток – он оптимизационный, а оптимизация с белого шума работает медленно. Чтобы получить приемлемый результат, нужно было сделать 100 итераций градиентного спуска, 200 проходов по нейросети, что достаточно долго.

Нам нужна нейросеть, которая за один проход по изображению делает стилизацию. Так взяли сеть типа кодировщик-декодировщик и обучили ее таким образом, чтобы она преобразовывала изображения, сохраняя их содержание, чтобы нейросетевой вектор-признак не менялся, но по стилю это изображение было похоже на другое изображение. Для обучения такой нейросети зафиксируем изображения стиля. Обучим ее следующим способом (рис.128)

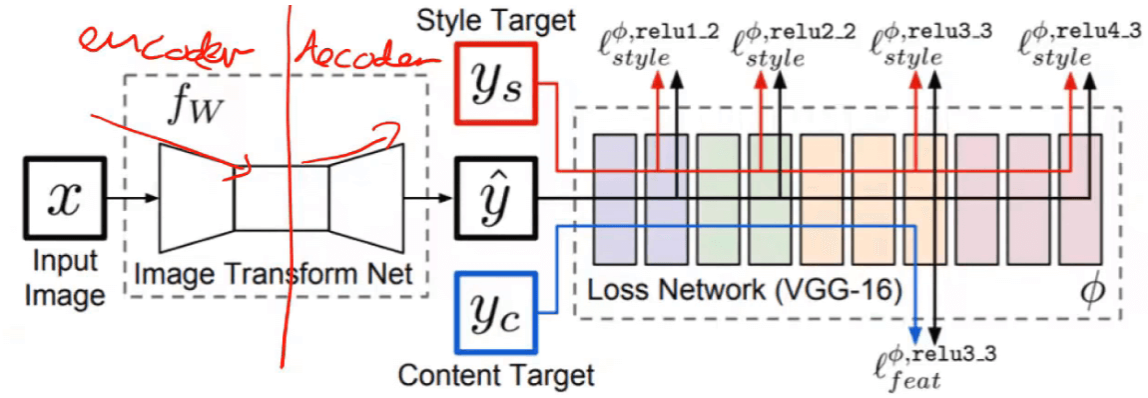


рис.128

- Воспользуемся предобученной на задаче классификации на ImageNet сетью VGG-16
- Будем использовать ее для извлечения признаков, она будет зафиксирована при обучении трансформационной сети
- Через нее прогоняем изображения y_s(стиля) и $y_c = x$ (содержание)

$\hat{y}$ - преобразованное нейросетью изображение, которое мы подаем на вход другой сети, преобученной на задаче классификации, например, VGG-16. Получаем нейросетевые признаки. Один из нейросетевых признаков мы запоминаем и сравниваем с исходным нейросетвым признаком y_c. Берем исходное изображение, прогоняем через эту нейросеть и получаем признаки. Мы хотим, чтобы эти признаки были одинаковыми, чтобы содержание изображения не изменилось, но стиль изображения $\hat{y}$ поменялся. Для этого берем изображение со стилем y_s, прогоняем его через ту же самую нейросеть, берем признаки с нескольких слоев, считаем матрицы Грама и в качестве функции потерь считаем разницу между матрицами Грама изображения y_s и $\hat{y}$. То есть наш целевой функционал – это взвешенная линейная комбинация похожести по содержанию и стилю. Получилась ошибка, делаем обратное распространение

ошибки, модифицируем нашу сеть преобразователь так, чтобы она лучше стилизовала изображение.

В итоге обучения мы получаем нейросеть, которая стилизует изображение за один проход.

Недостатком этого подхода к стилизации является факт того, что на каждое изображение стиля нам нужно обучать свою нейросеть. Хотелось бы сделать универсальный алгоритм, чтобы, с одной стороны, мы могли стилизовать изображение под любое заданное, а с другой стороны, чтобы мы могли это сделать за один проход по нейросети. Это смогли реализовать в работе Universal Style Transfer. Идея заключается в следующем:

- Обучим «автокодировщик» - сеть типа encoder-decoder, которая должна «сжимать» изображение в скрытое (latent) векторное представление F и декодировать его в исходное без потерь
- Будем манипулировать скрытым векторным представлением изображения F
- Обозначим признаки двух картинок F_c (содержание) и F_s (стиль). Пусть это матрицы размером CxA, где C – количество прзнаков.

Соответственно мы обучили автокодировщик, получили 2 вектор-признака в скрытых представлениях F_c (изображение, которое хотим преобразовать, из которого берем содержание) и F_s (изображение, из которого берем стиль).

Далее мы хотим найти такое преобразование, чтобы новый вектор F, был похож на исходный вектор F_c, но в то же время его матрица Грама совпадала с матрицей Грама изображения стиля. Мы можем решить эту задачу, посчитав специальное линейное преобразование, зависящее от F_s. Чтобы посчитать это, нудно разбить задачу на 2 этапа:

1. Уберем корреляцию исходных признаков линейным преобразованием $F_s F_s^T = I$ (изображение без стиля)
2. Приравняем нужные матрицы $FF^T = F_s F_s^T$

В итоге мы получаем сеть, которая делает необходимую стилизацию за 1,5 расчёта сети.

Вспомним, что мы генерировали структуры, начиная со случайного вектора. Давайте теперь скрытый вектор-признак изображения выкинем и заменим на случайный вектор шума, а затем случайный вектор шума стилизуем под изображение текстуры. В этом случае мы получим генератор текстур. Мы будем случайный вектор шума после стилизации раскрывать, декодировать в изображение текстуры.

Этот метод в дальнейшем развивался в метод модификации с пост-обработкой. После преобразования стиля применим к изображению фильтр, пришедший из матирования изображений, который делает градиенты итогового изображения похожими на градиенты исходного и тоже реализуемого в виде линейного преобразования.

Видно (рис. 129), что в результате стилизации изображение несколько разрушается: появляются достаточно сильные переходы, мелкая текстура также изменилась. Это происходит из-за того, что мы берем вектор-признак на одном уровне и стилизацию делаем на одном уровне. Для того, чтобы уменьшить эти искажения, авторы придумали метод пост-обработки следующего типа: ищем изображение, попиксельно похожее на полученное, но градиенты этого изображения будут похожи на градиенты исходного изображения.

Решить данную задачу можно линейным преобразованием и итог получится как на рис. 129 d.

Синтез изображений

Мы хотим научиться синтезировать целиком новые изображения, похожие на обучающую выборку. Задачу синтеза изображения мы тоже можем решить с помощью нейросети, то есть построить нейросеть, которая отображает вектор-признак в изображение.

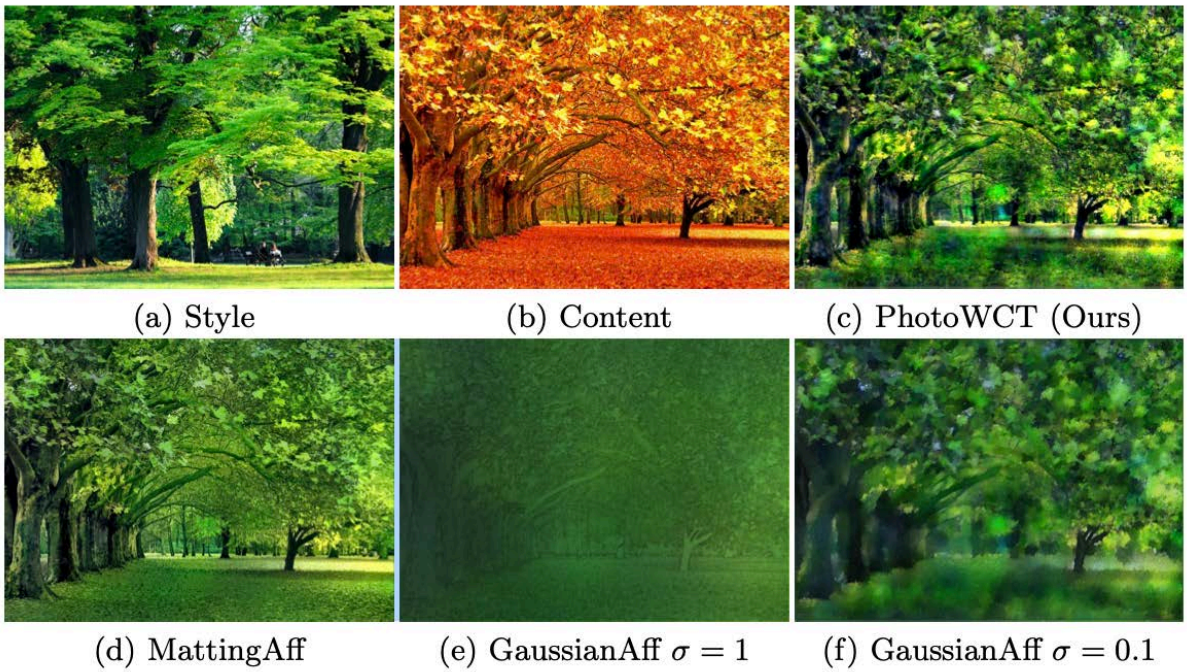


рис.129

Одна из главных проблем в компьютерной графике – оценка реалистичности изображений. В компьютерной графике использовалось 2 способа: посмотреть глазами или сравнить с эталонным изображением. Эталоны получить крайне тяжело. Кроме того, разница изображения с эталоном ничего не говорит о том, как нужно поменять

алгоритм, чтобы картинка стала более реалистичной. В нейросети GAN мы можем обучить метод классификации, который будет отличать синтезированные изображения от реальных. Поскольку классификатор – тоже нейросеть, мы можем рассмотреть генератор и классификатор как одну большую нейросеть, считать ошибку классификатора и распространять ее через генератор, но с изменением знака. Задача генератора обмануть дискриминатор, задача дискриминатора научиться классифицировать сделанное генератором.

Однако, мы не можем с самого начала хорошо обучить дискриминатор. Так как не знаем, что такое «подделки». Поэтому обучение должно проходить поэтапно:

- Инициализируем генератор и дискриминатор
- Фиксируем генератор (его веса)
- Берем примеры реальных и генерированных изображений
- Учим дискриминатор их различать (рис. 130)
- Фиксируем веса дискриминатора
- Синтезируем примеры генератором
- Распространяем ошибку от дискриминатора

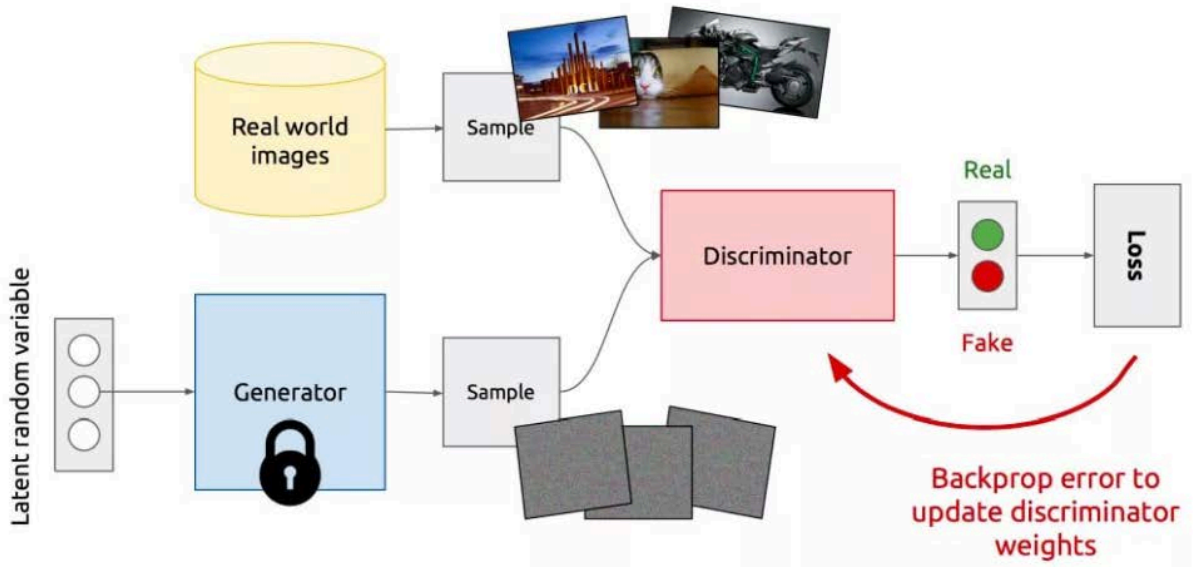


рис.130

Чередование обучения дискриминатора повторяется до сходимости, которая может и не состояться. В конце мы надеемся, что генератор научится синтезировать такие изображения, что дискриминатор не сможет их различить (рис. 131).

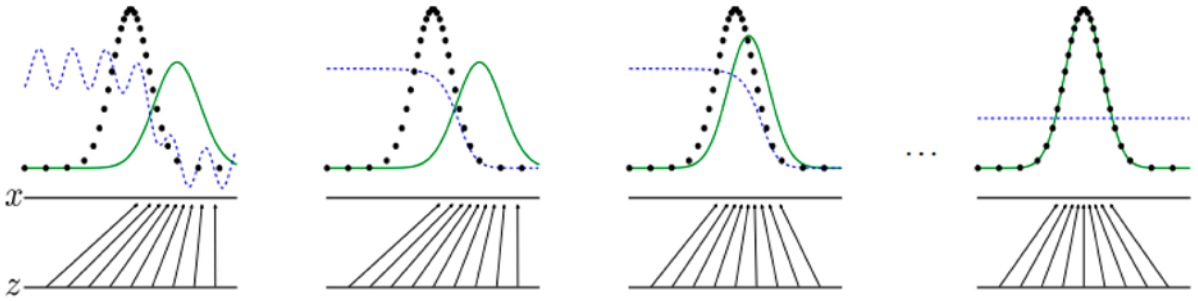


рис.131

Когда картинки, генерируемые нейросетями, стали более реалистичными, задались вопросом: как вообще связаны элементы генератора с получившимися изображениями.

Есть ли связь между конкретными свертками и классами объектов, которые генерируются?

Для этого на сгенерированных изображениях выделяем область, например, окна и ищем корреляцию между наличием окна в заданном месте и активациями сверток. Найдя соответствующие свертки, корреляция которых наиболее сильна, мы можем манипулировать их значения, например, увеличивать или занулять. Если мы занулим свертки, у которых корреляция наиболее сильна, то окна из изображений пропадут или деградируют. Значит, действительно есть некоторая корреляция, и какие-то определенные фильтры генерируют определенный визуальный образ.

Также можно попробовать применить векторную арифметику (рис. 132).

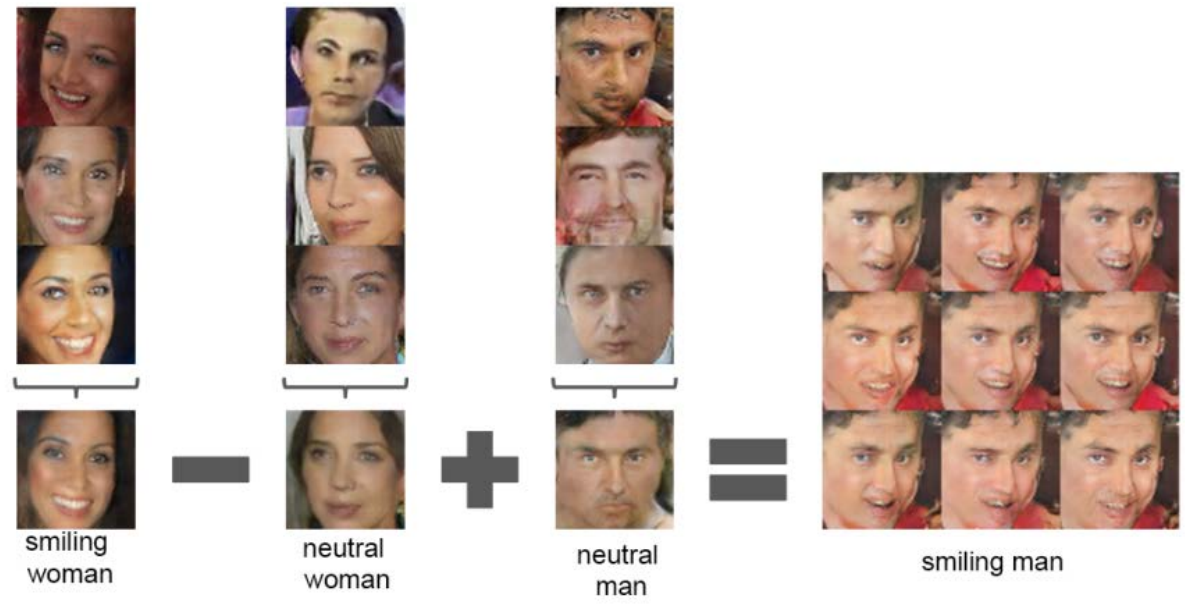


рис.132

Запомним изображения, которые нам нравится и те вектора шума, которые использовались для их генерации (вспомним, что на вход нейросети мы подаем некоторый случайный вектор, который генерирует картинку). Как связаны этот случайный вектор картинка мы заранее не знаем, поэтому отбираем картинки, которые нам понравились. Возьмем несколько картинок, которые мы классифицировали как «улыбающаяся женщина», усредним их вектор шума, проверим его: он действительно генерирует улыбающуюся женщину. Берем женщину с нейтральным выражением лица и мужчину с нейтральным выражением лица. Вычитаем из улыбающейся женщины нейтральную, получаем вектор шума, ответственны за улыбку. Добавляем к этому вектору мужчину с нейтральным лицом и получаем улыбающегося мужчину.

Wasserstein GAN (рис. 133) - это пример направления обучения генераторов, связанного с улучшением дискриминатора. Один из концептуальных недостатков исходных данных заключался в следующем: GAN просто определяет, является ли изображение подделкой, причем он обычно достаточно легко определяет подделки, особенно на начальных этапах обучения нейросети, для обучения генератора информация о том, что генератор делает подделки не очень полезна. Было бы лучше, если бы дискриминатор был бы не просто дискриминатором, а «критиком», который разбирался бы в «сортах» подделок, например, через сравнение выборки сгенерированных изображений с выборкой реальных изображений, то есть считал вероятностное расстояние между выборками. Один из вариантов сделать это с помощью расстояния Вассерштайна. Если мы заменим дискриминатор на метрику на основе расстояния Вассерштайна, то визуальное качество генерируемых изображений улучшится.

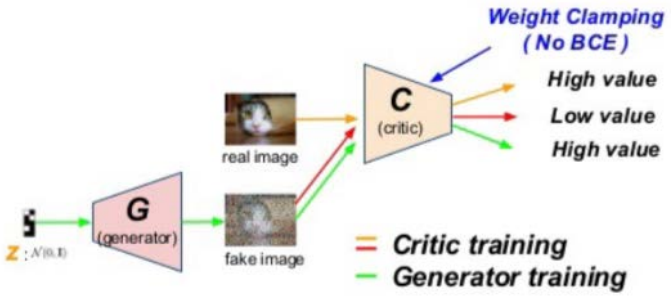


рис.133

Условные генераторы

Еще одно большое направление исследований – это условные генераторы. Основная проблема по всех генераторах – это абсолютная случайность. Мы даем какой-то вектор, получаем на выходе изображение, а нам бы хотелось контролировать генератор, как минимум, определять класс объектов, который мы хотим сгенерировать. Это можно промоделировать следующим способом: делаем условными и генератор, и дискриминатор. То есть генератор будет получать на вход помимо вектора шума еще некоторый вектор параметров, например, это будет метка класса объекта, который

нужно сгенерировать, а дискриминатор будет помимо сгенерированного изображения получать дополнительно эту метку (рис. 134).

Но условные GAN — это очень сложная архитектуру, которую не всегда удается обучить.

Идея дискриминатора как метода, который позволит оценить качество сгенерированного изображения, очень хороша не только для сетей, синтезирующих изображение, но и для сетей, делающих преобразования изображений. Одна из пионерских работ по соединению GAN и сетей преобразования работа PIX2PIX. Идея заключается в следующем: сделать преобразование изображения между двумя доменами, в которых изначально есть строгое соответствие, например, научиться генерировать изображение обуви по контурному рисунку обуви. Называем такую сеть преобразования изображений *генератором (рис. 135)*. На вход изображения краев, на выходе -изображение ботинка. Затем дискриминатор сравнивает изображение ботинка с картой краев и выдает решение: правильный ботинок сгенерировался или нет.

Что будет происходить, если в качестве функции потерь мы будем использовать попиксельную похожесть, дискриминатор, или сумму компонент в методе PIX2PIX?

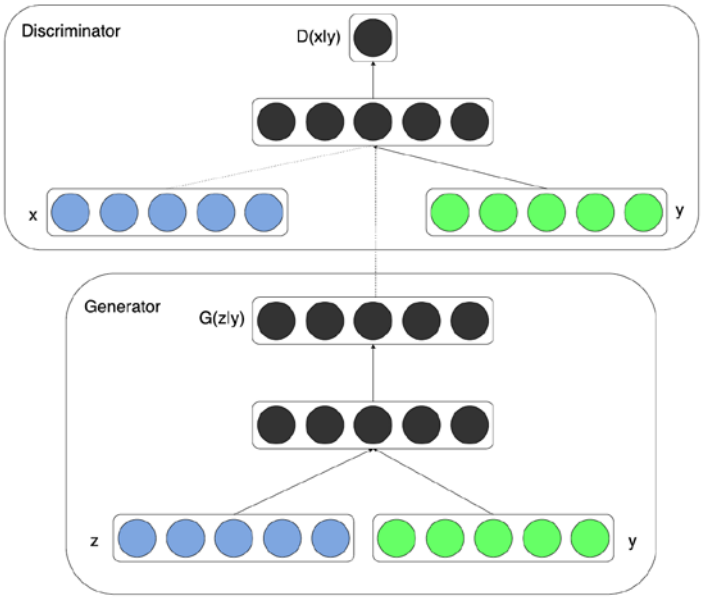


рис.134

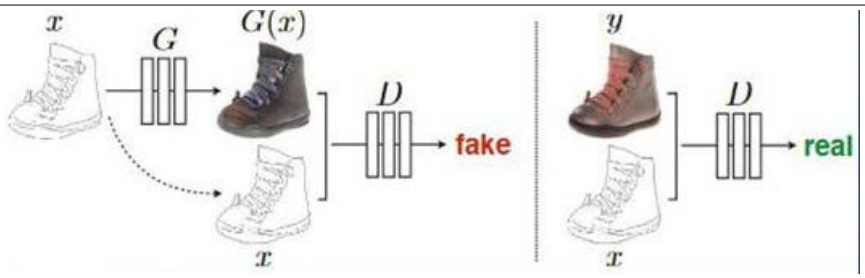


рис.135

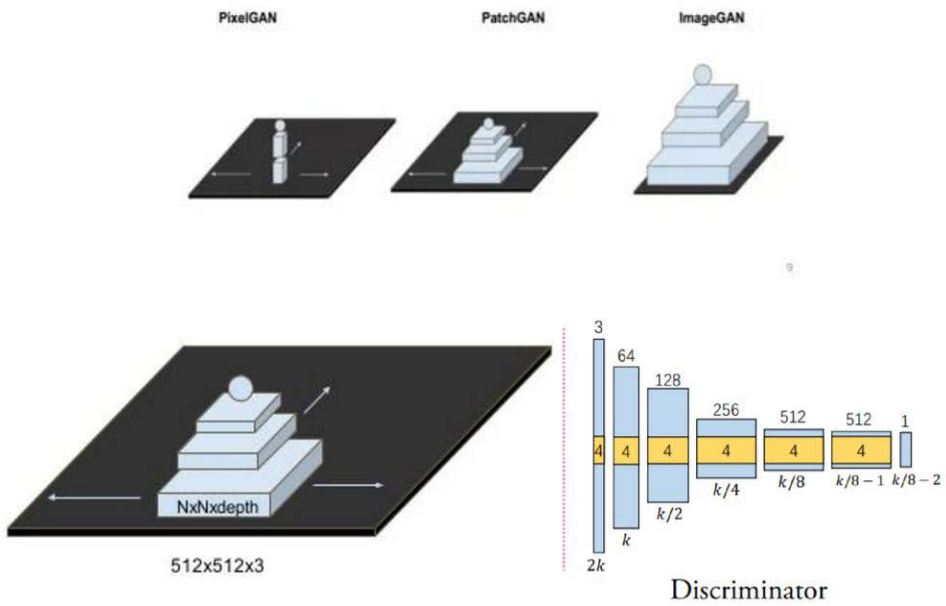
Есть обучающая пара (рис. 136 input и ground truth) карта разметки и изображение, по которому эта карта разметки была сгенерирована. Если мы будем обучать сеть-преобразователь, сравнивая целевое изображение со сгенерированным по попиксельной метрике L1, у нас получатся изображения, на который в целом пространственное расположение верно, но картинки очень размытые. Если мы используем GAN, то картинки получаются четкими, но в них могут быть отличия от исходных изображений. А если мы в качестве цели используем комбинацию попиксельной метрики и GAN, то мы получаем оптимальный результат.

рис.136

У этого метода есть одна интересная деталь: мы работаем с картинками высокого разрешения. Картинки высокого разрешения тяжело обучать, и не очень понятно, какой дискриминатор по ним делать. Количество пар изображений исходных и целевых ограничено. А значит, вместо того чтобы обучать дискриминатор целиком по всему изображению, мы можем обучить дискриминатор по фрагментам изображения, то есть он будет брать фрагменты целевого и исходного изображений и сравнивать их (правильно ли мы сгенерировали объект?). Затем, если картинка высокого разрешения, мы проходим дискриминатором по всему изображению, как скользящим коном и усредняем результат. Такой вариант дискриминатора получил название PatchGAN (рис. 137), потому что мы применяем дискриминатор к патчам изображений.

Обучать преобразователь мы тоже можем на патчах. То есть вместо того, чтобы обучать генератор и дискриминатор на всех изображениях, мы будем рать какие-то фрагменты изображений, случайные кропы, и обучать преобразование на них. За счет этого мы существенно размножим обучающую выборку и метод будет лучше учиться и стабильнее работать. Затем мы просто увеличим сеть под нужное нам целевое разрешение.

Метод PIX2PIX достаточно хорош, но имеет существенный концептуальный недостаток: он требует обучающей выборки в виде пар соответствующих друг другу изображений. У нас часто бывает ситуация, когда есть 2 домена, например, фотографии и картины и нет попиксельного сопоставления между этими картинами. Как обучать? Ответом стал Cycle GAN (рис. 138)



Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In Proc. CVPR 2017.

рис.137

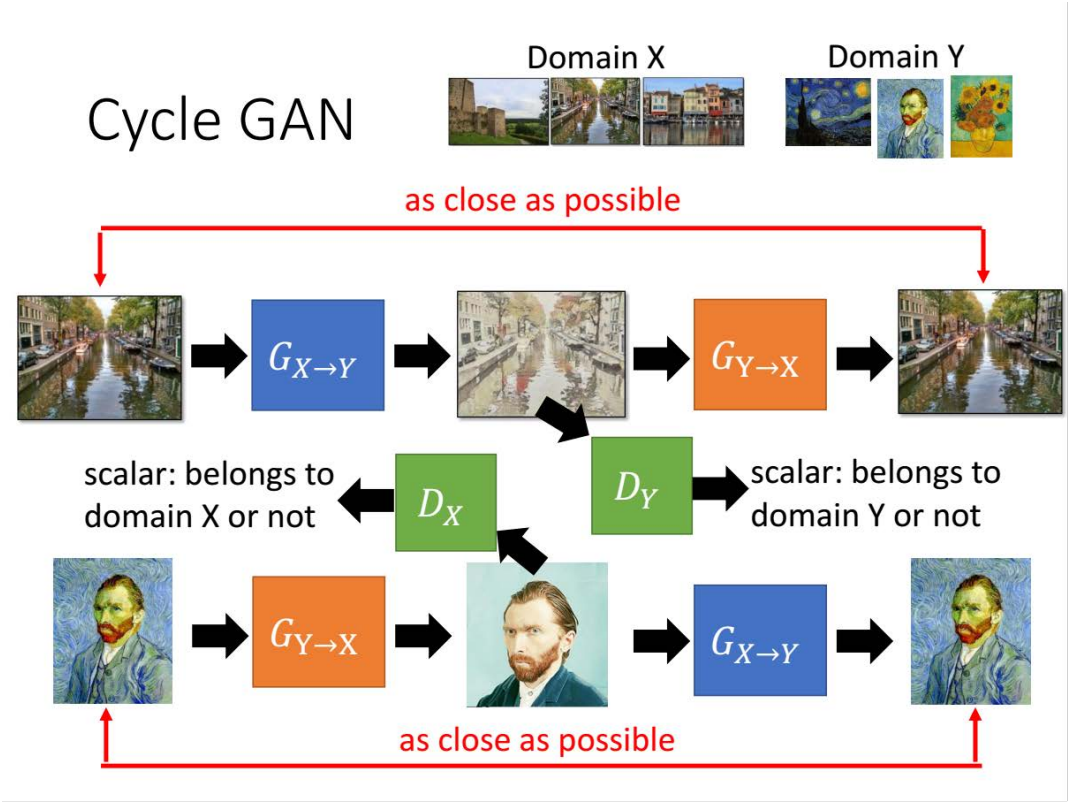


рис.138

Идея Cycle GAN заключалась в следующем: обучать одновременно 2 генератора: преобразование из первого домена во второй, из второго домена в первый. Допустим, у нас есть 2 домена X и Y. Отображаем X в Y, проверяем с помощью дискриминатора, что сгенерированная картинка у действительно принадлежит домену Y. Затем применяем обратный генератор из Y в X, получаем обратно преобразованное изображение, сопоставляем обратно преобразованное с исходным: нужно, чтобы после двойного преобразования изображения не поменялись.

Обучать Cycle GAN мы будем поэтапно: вначале обучаем один генератор в одну сторону, потом обучаем второй генератор, проверяем, что второй генератор теперь генерирует правильные картинки из домена X и что после обратного преобразования картинки не портятся. Все обучается совместно, качество преобразования получается хуже, чем у PIX2PIX, но зато мы можем работать с произвольными доменами.

Обучение сетей-генератор – одна из самых трудоемки и вычислительно тяжелых задач на данный момент. Обучит сеть, которая генерирует картинки высокого разрешения, с нуля ни у кого не получалось, пока NVIDIA не реализовала Progressive GAN (рис. 139).

Они показали, что при наличии вычислительных ресурсов можно обучить нейросеть генерации картинок высокого разрешения, например, мегапиксель, обучая последовательно:

- Обучим генератор и дискриминатор картинок 4x4 пиксела
- Дополним их слоями, чтобы теперь генерировать и классифицировать картинки размеров 8x8 пикселов
- Сделаем еще один этап обучения
- И так далее достроим сеть до генерации мегапиксельных картинок

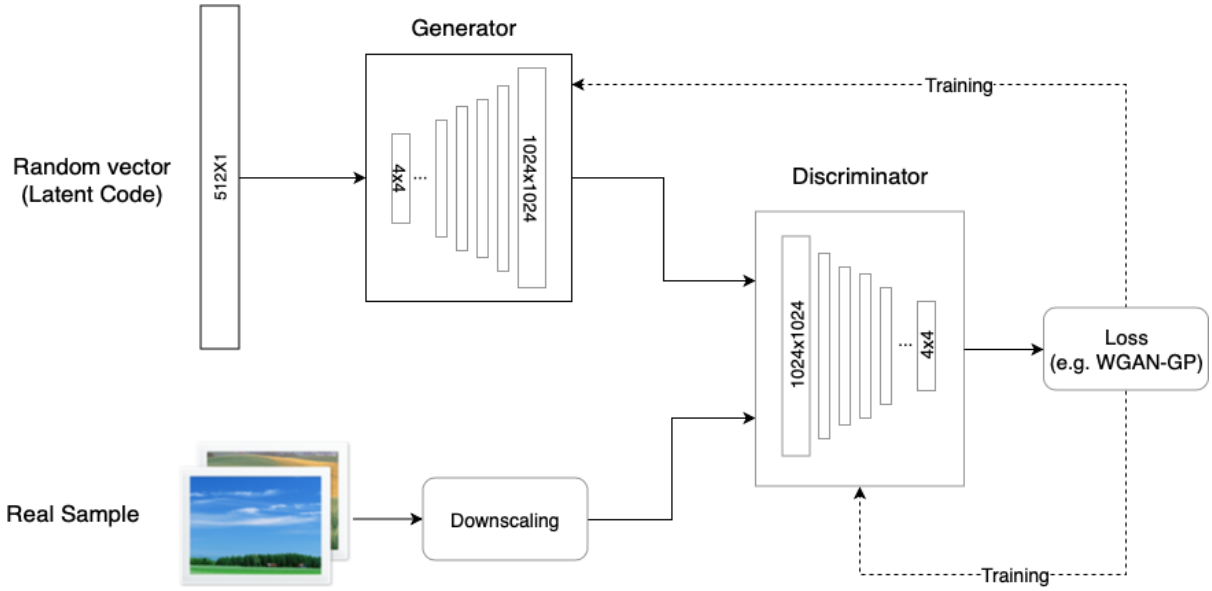


рис.139

До финального изображения высокого качества на обучение ушло 14 дней.

Картинки, которые получил Progressive GAN, были очень хорошие, но люди все равно их легко отличали от реальных изображений. В декабре 2018 года вышла работа StyleGAN (рис. 140).

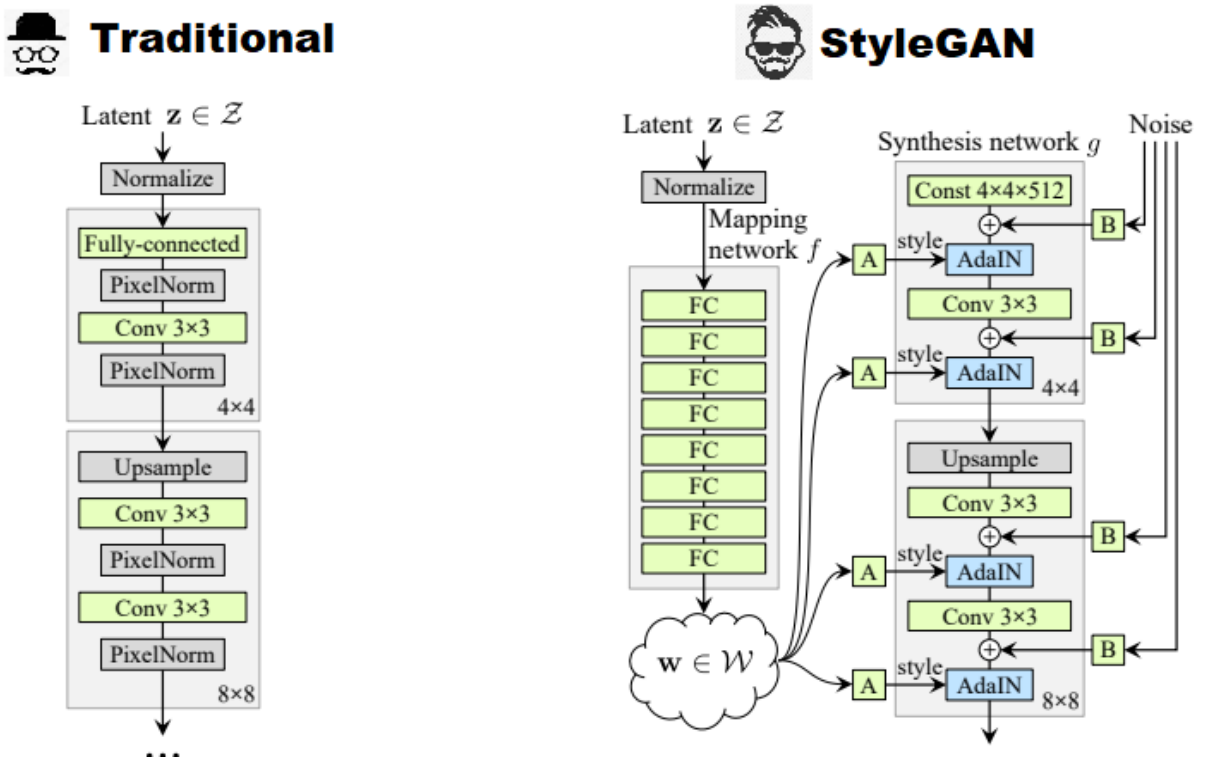


рис.140

Вспомним, что на каждом уровне свертки отвечают за определенные визуальные образы иклассы объектов. Хотелось бы воспринимать нейросеть как некоторый управляемый алгоритм генерации изображений.ю чтобы мы могли отдельными элементами этого алгоритма управлять. В полной мере это не удается, но мы можем построить нейросеть, которая будет сама управлять элементами генерации изображений. Как это сделать: мы хотим иметь возможность усиливать вклад одних сверток и уменьшать вклад других сверток. Это можно сделать с помощью *адаптивного instance-нормализационный слой (AdaIN).*

Вообще нейросети требуют хорошей нормализации, и в них встраивют разные нормализационные слои. В одной из работ (кстати, российских атворов) было показано, что в сетях-гненераторах хорошо работает instance- нормализационный (нормализация по данному экземпляру)слой. Идея instance- нормализационного слоя в следующем: мы считаем для каждого конкретного текущего примера среднее значение отклика фичи по каналу и дисперсию отклика фичи по каналу и делаем нормализацию, то есть вычитаем среднее и делим на дисперсию. Если добавлять в нейросеть такие слои, то выходы каналов приводятся к интервалу от -1 до 1 с нулевым автозаданием, и сеть учится лучше.

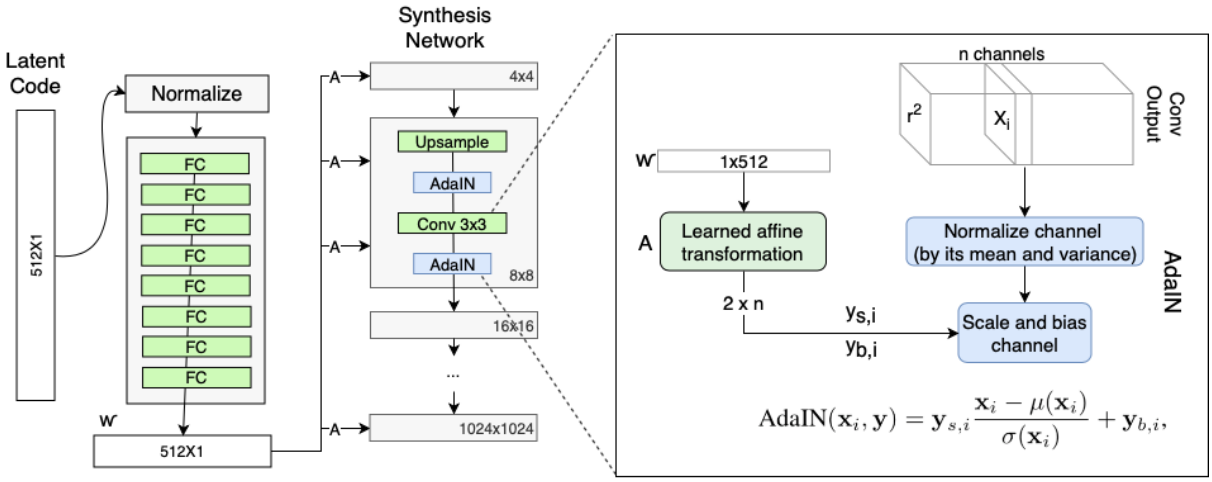


рис.141

Чтобы превратить сеть в управляемую, добавим масштабирующий коэффициент y_s и сдвиг y_b (рис. 141), параметры y_s и y_b мы будем брать откуда-то еще, это и будут управляющие параметры. Сеть-генератор будет устроена как обычно: мы берем какой-то случайный вектор, поадем на вход последовательным блокам, которые генерируют картинку заданного разрешения, поэтапно повышая его. В отлтче от предыдущего блока, где было просто повышение разрешения свертки, между свертками будут адаптивные instance-нормализационные слои. Откуда взять управляющие сигналы для этого блока?

Сделаем специальную управляющую нейросеть, которая будет получать на вход случайный вектор и из этого случайного вектора генерировать управляющие сигналы для адаптивных слоев.

Оказывается, что на вход генератору можно подавать константное мини-изображение, которое будет преобразовываться через управляющие сигналы в целевое.

Для того, чтобы изображения получались более рандомизированными, мы добавим второй источник шума, то есть не только будем получать управляющие сигналы от сети-контролера, но будем подмешивать еще шум. Будем гненерировать шумные картинки, суммировать их с нейросетевыми признаками генератора и затем подавать их на управляющий *AdaIN,* повышая разрешение.

Для того, чтобы получить картинки высокого разрешения, придтся учить эту сеть в прогрессивном режиме, как и Progressive GAN.

Кроме того, авторы внедрили в StyleGAN интересно свойство: у нас есть несколько уровней генерации изображений, мы генерируем картинки нескольких разрешений, соотвтсвенно управляющие сигналы поступают в нейросеть на разных уровнях. Мы хотим приучить нейрпосеть к тому, что управляющий сигнал, который подан на первом уровне, никак не связан с управляющим сигналом на втором уровне. Это нужно для

того, чтобы нейросеть научилась трактовать все сигналы независимо, и тогда, возможно, все эти сигналы будут отвечать за свои свойства изображения. Чтобы добиться такого при обучении, возьмем нескоько случайных векторов , пустим их через несколько сетей контролеров , каждая сеть сгенерирует свои управляющие параметры. Управляющие парметры для первого этапа возьмем из первой сети, для второго этапа – со второй сети, для третьего этапа с третьей сети. Поскольку сигнала сгенерированы разными сетями, а итоговый результат должен быть реалистичным лицом, сети генератора придется приучиться к тому, что эти параметры разные.

К примеру, у нас есть два источника: A и B (рис. 142). Когда мы генерируем картинку на пересечении двух источников, мы все парметры с целевой картинки из ресурса A за исключением параметров на определенном слое, которые берутся из картинки ресурса B. То есть разделим «стиль» на компоненты. Для этого будем учить сеть так, что на разные уровни поадется управляющий сигнал, сгенерированный от разных кодов. Можем управлять генерацией, выбирая интересные нам изображения и подмешивая их в коды. В Примере стиль от B на соответсвующем уровне, а все остальное от A.

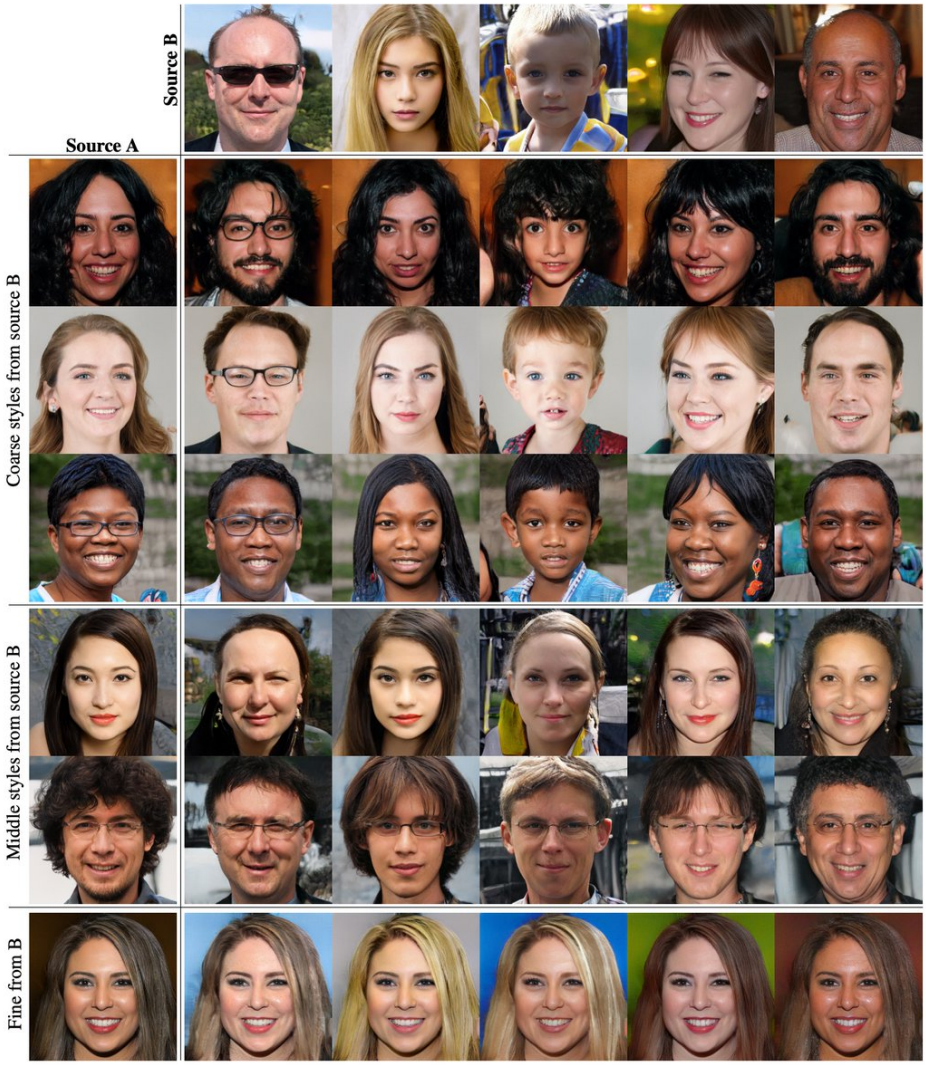


рис.142

На основе управляемой генерациии сделалаи генератор фотографий SPADE с семантической разметкой, задаваемой пользователем. Для того, чтоб учитывать пространственное распределение картинок, потребовалось сделать прстранственно зависимую модификацию слоя *AdaIN*. В этом слое параметры свертки задавались целиком для всего канала.

Авторы SPADE (рис. 143) сдлеали генерирующую сеть, которая генерирует не поканальные параметры, а поканальные парметры для каждого пиксела изображения независимо. Множество таких слоев, на каждом уровне подаем свои управляющие сигналы и в конце дискриминатор, оценивающий картинку, на вход получает не только картинку, но и карту разметки. Соответственно он требует, чтобы в нужных местах, где сгенерировалась кратинка, сгенерировался объект заданного класса.

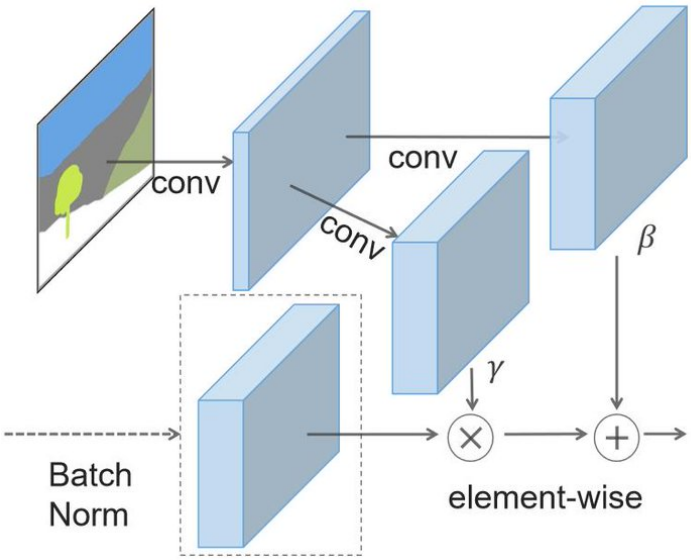


рис.143

При этом в отличие от StyleGAN мы сохраняем подачу некоторого случайного вектора, для того, чтоб этот случайный вектор имел смысл, делаем следующее (рис. 144):

- Обучаем архитектура кодировщик-декодировщик с помощью PIX2PIX
- Берем выход промежуточного слоя из этой сети (PIX2PIX)
- Берем изображение, из которого нужно сгенерировать стиль, подаем на вход. Теперь сеть будет генерировать все изображения в заданном стиле
- Пространственное расположение будет задаваться картой семантической разметки. Через сеть-контролер эта информация будет подаваться на генератор и управлять им таким образом, что в нужном месте изображения генерировался объект

То есть на вход подается случайный вектор, вместо случайного вектора можно подавать «сжатое» представление стиля, полученное из кодировщик-декодировщик сети.

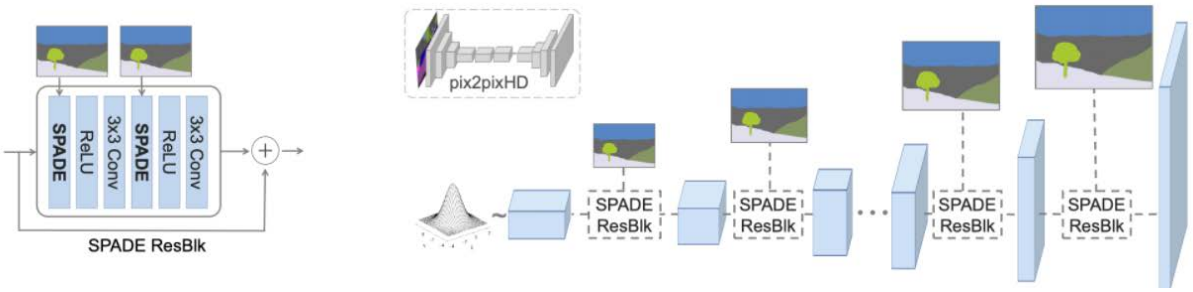


рис.144

К подобного рода задачам преобразования изображений можно отнести очень много задач:

- Задача суперразрешения: генерация картинки более высокого разрешения с помощью сетей с ошибками на основе дискриминаторов
- Предсказание кадров путем интерполяции промежуточных кадров с добавлением дискриминатора
- Domain Adaptation (многие сети работают хорошо только на данных, на которых они обучились) задача состоит в том, чтобы научиться обучать алгоритмы на синтетических данных, чтобы он хорошо работал на реальных данных

Резюме

- Реалистичность изображений можно оценить с помощью сети-дискриминатора
- Похожесть изображений хорошо оценивается с помощью Perception Loss - сравнения значений нейросетевых признаков (Identity Loss, если признак кодирует «личность»)
- Style Loss - функция от признаков, описывающая статистики распределения признаков
- Можем проводить линейные преобразования в пространстве признаков изображения
- Cycle Loss позволяет работать с произвольными выборками, а не с парами соответствующих изображений
- AdaIN и SPADE слои позволяют управлять синтезом, а параметры для них генерируем управляющими сетями

Лекция 9. Основы анализа видеоданных

Оптический поток

Видеоданные могут быть разного типа. Принципиально мы можем разделить их на 2 класса: *видеопоток* и *видеопоследовательность.*

Видеопоток – упорядоченная последовательность изображений, полученных с одной камеры через небольшие промежутки времени. Видеопоток подразумевает обработку на лету.

Видеопоследовательность конечна, ее можно обрабатывать целиком.

Если рассматривать временное окно, то можно свести обработку видеопотока к видеопоследовательности. Мы можем обрабатывать видеопотоки с временным окном, принимая решение с задержкой равной длине этого окна.

Также в рамках обработки видео нам важен его размер:

- Пользовательское видео от 3-5 кадров/сек до 30-50 кадров/сек
- Разрешение от 320x240 до 1920x1080 (HD) (сейчас еще 4K)
- В градациях серого (одноканальное) или цветное (3-х канальное)
- Поток данных 2Мб (один канал HD) x 3 (RGB) x 50 кадр/с = 300 Мб/с, что больше, чем пропускная способность 1 гигабитной Ethernet сетки

Это означает, что, когда бы мы с видео не работали, почти всегда оно будет в сжатом виде. Соответственно всегда пропадает часть информации. Кроме того, нам всегда требуется время на сжатие и разжатие видео, поэтому сжатие и разжатие – это операции, которые обычно реализуются аппаратно.

Что мы хотим от анализа видео?

Мы хотим извлечь следующую информацию из видео:

- Найти все объекты
- Отследить их движение
- Определить свойства (позы, атрибуты, личность)
- Распознать действия людей и события

К особенностям видео можно добавить следующее: собирать в качестве данных отдельные кадры гораздо проще, чем собирать видео, потому что это долго и, например, точки обзора у видеокамеры ограничены. И если мы обучим детекторы автомобилей по данным с камеры видеонаблюдения, то ракурсов у нас будет немного, в лучшем случае 100, а 100 разных видов фонов для обучения алгоритма

детектирования объектов – это слишком мало, поэтому приходится использовать дополнительную информацию в виде отдельных кадров.

Кроме того, из-за того, что ракурсы съемки очень разные, пока не получается сделать алгоритм, который бы хорошо работал со всеми ракурсами.

Оптический поток (Optic flow)

Самая первая задача в анализе видео – это оценка движения или оптического потока. Движение – главное отличие видео от изображений. Оно само по себе является мощной визуальной подсказкой. Есть множество психофизиологических экспериментов, в которых пользователям показывают движение отдельных точечных источников света (нет никаких контуров, фигур и прочее). Человек может распознать происходящее в видео, наблюдая за движением отдельных точек. Например, если мы закрепим точечные источники света на теле человека, то мы сможем распознать с высокой вероятностью его пол, настроение, занятие.

Возникает вопрос: как описать движение, если сцена трехмерная, а изображение двухмерное?

Поле движения (motion field)– это векторное поле 2D проекций на изображение 3D точек объектов сцены. Именно его мы хотим формализовать и измерить. Однако движение трехмерных точек можно увидеть далеко не всегда. Это зависит от текстуры, контрастности объекта и т.д. Экстремальный пример – это серый матовый шар, который освещен с одной стороны и вращается вокруг своей оси. Если нет шумов в камере, то мы не увидим никакого изменения несмотря на то, что шар движется. Под оптическим потоком подразумевают векторное поле видимого движения пикселов между кадрами. Нахождение оптического потока – одна из базовых задач анализа видео.

Optical flow (оптический поток) – векторное поле видимого (u_{ij}, v_{ij}) движения между кадрами

Мы хотим оценить плотное поле, то есть для каждого кадра посчитать вектор оптического потока. Еще иногда рассматривают задачу разреженного оптического поля, когда мы находим движение только для отдельных точек в кадре.

Поскольку задача плотная, визуализировать результат работы просто так не получится. Традиционно есть 2 варианта:

1. Вектора движения для отдельных точек

   То есть мы показываем движение не всех точек, а части точек, например, берем точки по некоторой сетке, визуализируем текущее положение точки точкой, смещение к текущему положению от предыдущего кадра визуализируем отрезком.

2. Цветовое кодирование вектора движения

Каждому направлению и амплитуде задается свой цвет и своя яркость.

Нахождение оптического потока – это пример задачи, для которой максимально трудно подготовить обучающие данные. Один из наборов данных, придуманный для этой задачи, Middleburry optical flow dataset, предлагал несколько способов задания оптического потока, один из которых предполагал съемку в двух диапазонах. То есть все не очень сильно текстурированные объекты покрывались флюоресцирующей краской так, чтобы краска при увеличении была видна в виде отдельных пятнышек. Съемка ведется со специальной камеры: либо с двух близко установленных камер, либо лучше с одного объектива, но на 2 матрицы соответственно в видимом диапазоне и в ультрафиолетовом диапазоне. В ультрафиолетовом диапазоне съемка ведется с высоким разрешением так, что каждое пятнышко видно в отдельности. Значит, можно эти пятнышки принять за особые точки, сопоставить их между соседними кадрами, посчитать движение отдельных точек. Если теперь уменьшить изображение, то получится, что в каждом пикселе есть точка из другого диапазона, соответственно мы сможем оценить движение оптического потока в каждом пикселе. В итоге датасет получается маленький, но сложный (порядка 50 последовательностей).

Другой вариант, более широко используемый, — это синтетические данные. Однако этот метод тоже сильно ограничен: реалистичность сцен целиком зависит от доступа к контенту. По-настоящему качественная графика применяется для спецэффектов кино, это крайне дорогостояще, поэтому создавать эталонные последовательности из таких хороших синтетических данных для исследователей невозможно.

Одним из практических датасетов, которые сейчас используются, является датасет KITTY (рис. 145) для беспилотных автомобилей. В нем камеры были установлены на автомобиле и для того, чтобы построить плотный оптический поток делалось следующее: для каждого кадра имеется разреженное множество трехмерных точек, в кадре есть статичные объекты (окружение) и движущиеся объекты. Особенно сложно с движущимися объектами, поскольку они движутся сами по времени, в результате их движение от кадра к кадру очень сложное. Что мы делаем:

1. 3D сканирование
   Восстанавливаем 3D форму сцены (неподвижной ее части, отдельно выделяя движущиеся объекты)

2. Сопоставляем точки неподвижной сцены от кадра к кадру между разными ракурсами
3. Для движущихся объектов (автомобилей) авторы делали так: находили похожую 3D модель автомобиля, вписывали ее вручную в трехмерное облако точек в каждом кадре. Получалась одна и та же 3D модель автомобиля в каждом кадре, значит, было известно движение каждой точки от кадра к кадру и считался оптический поток.

Это очень трудоемко и в конечном датасете было не более 200 обработанных пар кадров.

рис.145

Методы оптического потока исследуются очень давно, первые работы появились в начале 80-х годов. Схема оценки алгоритмов оптического потока эволюционировала. Классическая схема последних алгоритмов выглядит так (рис. 146):

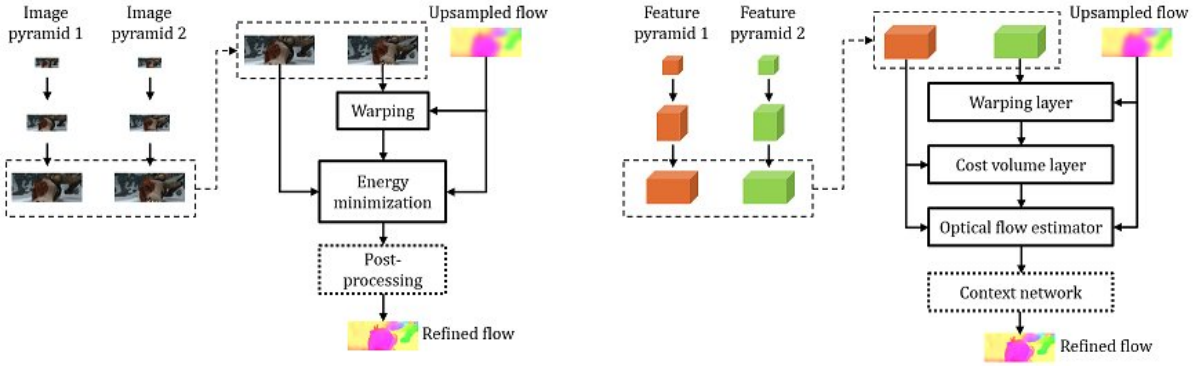


рис.146

Во-первых, мы работаем с пирамидами разрешений, то есть строим уменьшенные копии всех изображений и оцениваем оптический поток последовательно: сначала плотно сопоставляем самые маленькие ракурсы низкого разрешения, получаем какую-то оценку оптического потока, затем переходим на следующих масштаб, берм 2 изображения более высокого разрешения, увеличиваем разрешение карт оптического потока, полученного для предыдущего кадра. Применяем эту карту оптическогоп потока для того, чтобы трансформировать один из кадров в паре, приблизив его к первому кадру (мы убрали существенное движение, и разница между 2 кадрами теперь должна быть незначительной, тем не менее она есть из-за разрешения). Мы получили 2

приближенные карты и теперь считаем оптический поток между текущим и трансформированным предыдущим кадром. Для расчета оптического потока задаем некоторый целевой функционал энергии, который обычно зависит от оптического потока изображения в целом. Мы задаем функционал для всего изображения, поскольку на него можно наложить разные дополнительные требования, например, гладкость оптического потока, разрывы в некоторых краях итп. Этот функционал мы оптимизируем с помощью функции оптимизации и получаем уточненный результат. Так мы делаем несколько раз до максимального разрешения, после чего запускаем еще один этап пост-процессинга. На этапе пост-процессинга оптический поток дополнительно уточняется с использованием некоторых дополнительной информации.

Эта схема достаточно медленная, поскольку нужно работать с многими разрешениями, есть функционал энергии, который нужно оптимизировтаь (причем параметры, по которым оптимизируем – это двухкратное число пикселов в изображении, то есть множество векторов оптического потока во всем изобажении). Тем не менее результаты могут быть весьма точными.

Когда появились нейросети, первое, что пытались сделать – упростить этот процесс, заменив оценку оптического потока просто на сверточную нейросеть. Какой метод может быть самым простым?

Возьмем обысную сверточную архитектуру нейросети, например, ту, что использовалась для сегментации изображений. Объединяем 2 изображения в биканальное изображение, конкатенируя их вектор, подаем на вход нейросети, реконструируем карту оптического потока.

Можно сделать сложнее, объединив признаки с 2х картинок с помощью специального слоя сравнения патчей. Получим архитектуру FlowNet (рис.147) Как обучать такую нейросеть?

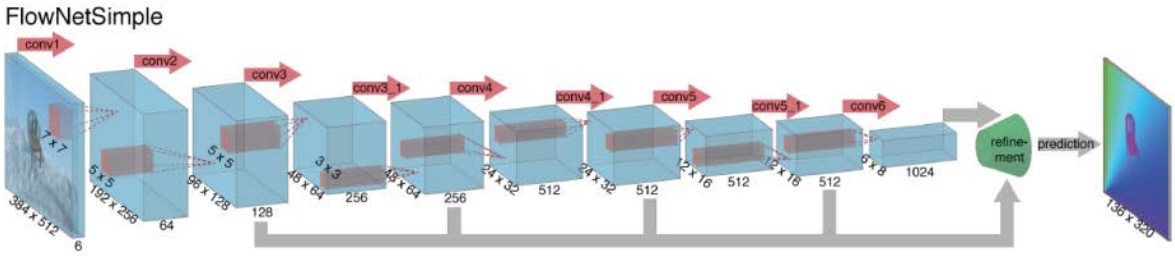


рис.147

Если данных нет, нужно их синтезировать. Поскольку авторы этой сети не были большими доками в компьютерной графике, они сделали достаточно простой эталонный набор: летающие стулья (рис. 148).

рис.148

Для этого берутся какие-то кадры, и на х фоне начинают летать стулья с разным освещением и в разных направлениях. Таких данных можно сгенерировать большое количество. Оказалось, что этих данных достаточно для того, чтобы обученная на них нейросеть показывала весьма неплохие результаты. Важны трансформации точек, чтобы нейросеть научилась определять кадры движения и сопоставлять точки. Разумеется, если применить к эталонным данным модель, обученную на этих данных, результат будет не ахти, но мы можем использовать это как предобучение и затем дообучить нейросеть на целевом датасете, например, на KITTY.

Следующим шагом стала PWC-NET (Pyramid, Warping, Cost volume) (рис. 149) – реализация классического подхода через нейросетевые признаки и нейросетевой вывод. Мы можем не строить пирамиду разрешений, поскольку признаки вычисляются с помощью сверточной нейросети, где и так есть разные уровни пространственного разрешения.

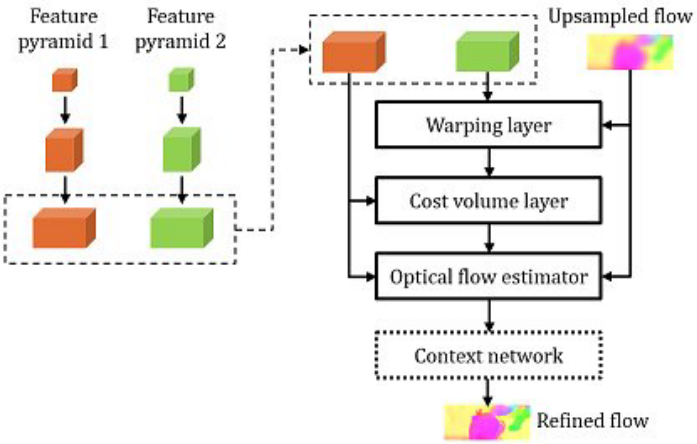


рис.149

Предположим, мы вычислили признаки и карту оптического потока на каком-то уровне и теперь хотим посчитать оптически поток для большего в 2 раза разрешения. Для этого повысим разрешение карты оптического потока с предыдущего разрешения в 2 раза с помощью билинейной интерполяции. Применим карту билинейной интерполяции для трансформации признаков предыдущего кадра. Поскольку признаки на предыдущем кадре – это тоже изображение низкого разрешения, но большой толщины, ничто не мешает рассмотреть его как изображение и к нему применить движение по оптическому потоку. Далее нужно посчитать стоимость сопоставления всех пикселов из некоторой окрестности *Cost Volume*. Для этого мы трактуем все признаки как признаки соответствующих пикселов. Можем сравнивать их друг с другом, например, через среднеквадратичное отклонение или через скалярное произведение. Мы считаем всевозможные смещения, то есть сравниваем пикеслы по похожести в некоторой окрестности, то есть говорим, что пиксел может сместиться на данном уровне не более, чем на 2 пиксела. Значит текущий пиксел нужно сопоставить с пикселами второго изображения в окрестности 5x5 пикселов, то есть посчитать 25 разных значений похожести. Далее составим из них новую трехмерную матрицу Cost volume. Эта трехмерная матрица будет того же пространственного разрешения, что и текущий уровень признаков, в ней будет 25 слоев, для каждого слоя будет оценка похожести пикселов при заданном смещении. Этот набор стоимостей затем нужно передать в оптимизатор, который должен оптимизировать целевой функционал. Оптимизатор заменяем на нейросеть, которая получает на вход трехмерный объем стоимости, признаки одного изображения и карту оптического потока. На выходе она выдает уточненную карту оптического потока.

Чтобы посчитать Cost Volume, считаем корреляцию (произведение) признаков для сдвигов не более d. Размер Cost Volume равен $d^2 \times Width \times Height$

Оказалось, что такой метод и быстрый, и точный. Если сравнить этот метод с существующими аналогами, мы увидим, что он существенно превосходит все более точные методы по скорости (рис.150).

	Method	Sintel Clean		Sintel Final		KITTI 2012			KITTI 2015	
		train	test	train	test	train	test	test(Fl)	train	test(Fl)
Unsupervised	BackToBasic+ft [20]	–	–	–	–	11.3	9.9	–	–	–
	DSTFlow+ft [37]	(6.16)	10.41	(6.81)	11.27	10.43	12.4	–	16.79	39%
	UnFlow-CSS [29]	–	–	(7.91)	10.22	3.29	–	–	8.10	23.30%
	OccAwareFlow+ft [46]	(4.03)	7.95	(5.95)	9.15	3.55	4.2	–	8.88	31.2%
	MultiFrameOccFlow-None+ft [18]	(6.05)	–	(7.09)	–	–	–	–	6.65	–
	MultiFrameOccFlow-Soft+ft [18]	(3.89)	7.23	(5.52)	8.81	–	–	–	6.59	22.94%
	DDFlow+ft [26]	(2.92)	**6.18**	3.98	7.40	2.35	3.0	8.86%	5.72	14.29%
	Ours	**(2.88)**	6.56	**(3.87)**	**6.57**	**1.69**	**2.2**	**7.68%**	**4.84**	**14.19%**
Supervised	FlowNetS+ft [10]	(3.66)	6.96	(4.44)	7.76	7.52	9.1	44.49%	–	–
	FlowNetC+ft [10]	(3.78)	6.85	(5.28)	8.51	8.79	–	–	–	–
	SpyNet+ft [35]	(3.17)	6.64	(4.32)	8.36	8.25	10.1	20.97%	–	35.07%
	FlowFieldsCNN+ft [2]	–	3.78	–	5.36	–	3.0	13.01%	–	18.68 %
	DCFlow+ft [49]	–	3.54	–	5.12	–	–	–	–	14.83%
	FlowNet2+ft [15]	(1.45)	4.16	(2.01)	5.74	(1.28)	1.8	8.8%	(2.3)	11.48%
	UnFlow-CSS+ft [29]	–	–	–	–	(1.14)	1.7	8.42%	(1.86)	11.11%
	LiteFlowNet+ft-CVPR [14]	(1.64)	4.86	(2.23)	6.09	(1.26)	1.7	–	(2.16)	10.24%
	LiteFlowNet+ft-axXiv [14]	**(1.35)**	4.54	(1.78)	5.38	(1.05)	1.6	7.27%	(1.62)	9.38%
	PWC-Net+ft-CVPR [43]	(2.02)	4.39	(2.08)	5.04	(1.45)	1.7	8.10%	(2.16)	9.60%
	PWC-Net+ft-axXiv [42]	(1.71)	3.45	(2.34)	4.60	(1.08)	**1.5**	6.82%	(1.45)	7.90%
	ProFlow+ft [27]	(1.78)	**2.82**	–	5.02	(1.89)	2.1	7.88%	(5.22)	15.04%
	ContinualFlow+ft [31]	–	3.34	–	4.52	–	–	–	–	10.03%
	MFF+ft [36]	–	3.42	–	4.57	–	1.7	7.87%	–	**7.17%**
	Ours+ft	(1.68)	3.74	**(1.77)**	**4.26**	**(0.76)**	**1.5**	**6.19%**	**(1.18)**	8.42%

рис.150

Визуальное сопровождение объекта

Visual Object Tracking (VOT)

Мы хотим отследить движение объектов по всем кадрам и определить траектории их движения. Выходом алгоритмов сопровождения объектов является траектория движения объектов для каждого найденного объекта нужно определить последовательность его положений в каждом кадре видео.

Есть несколько задач сопровождения, и визуальное сопровождение объектов – одна из этих задач. Мы рассматриваем задачу слежения за одним объектом, этот объект уже задан на первом кадре, но при этом мы ничего заранее не знаем, про объект, который нам нужно отслеживать. Это означает, что у нас нет детектора этого объекта. И если мы потеряем объект, то снова найти его не сможем. Поэтому эти методы не очень надежны, и реально мы можем сопровождать объекты только на коротких интервалах времени. Кроме того, мы принципиально не используем следующие кадры, то есть мы работаем с видеопотоком и считаем, что отслеживаем движение объекта налету.

Сложность задачи заключается в следующем:

- Вычислительная нагрузка

Нужно обрабатывать N кадров в секунду

- Изменение во времени

Вид объекта меняется от кадра к кадру из-за изменения ракурса, освещения, внутренних изменений

- Взаимодействие объектов

Перекрытие объектов, визуальное сходство и т.д.

Сейчас развитие методов визуального сопровождение объектов вращается вокруг открытых ежегодных конкурсов по методам отслеживания движений. Самый крупный из них VOT Challenge, на нем подводят итоги годовых разработок. Основные требования конкурса:

- Открытая реализация
- Matlab-toolkit для оценки
- Оценка качества и скорости
- Небольшой, но разнообразный набор роликов в качестве тестового датасета
- Короткие ролики (порядка 100 кадров)

Сейчас появились конкурсы для отслеживания в течение длительного времени и отслеживания с учетом дополнительной информации (RGB + Depth, RGB + IN).

Поскольку в методе сопровождения нет детектора объектов, сопровождение проводится итеративно. В каждый момент времени мы рассматриваем 2 кадра: новый кадр и предыдущий кадр (рис. 151). Строим некоторое описание области изображения. Которую занимает объект на предыдущем кадре. Затем в новом кадре ищем такую область изображения в окрестности предыдущего положения, которая по своим визуальным характеристикам больше всего похоже на область объекта из предыдущего кадра. Повторяем это для следующих кадров.

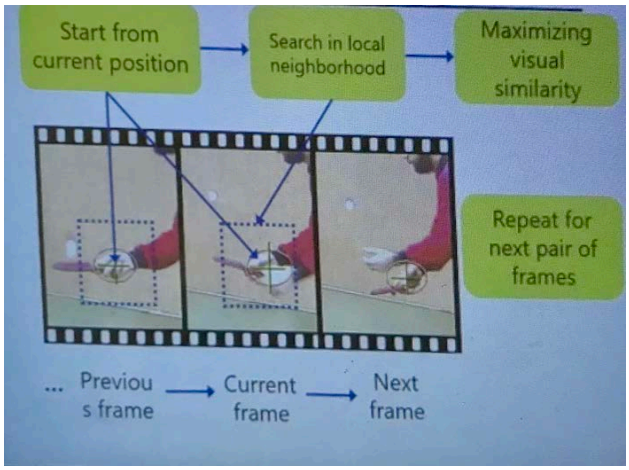


рис.151

Поскольку мы ищем окрестность максимально похожую, простейший метод – сопоставление шаблона. Берем фрагмент изображения как маленькую матрицу

попиксельно и ищем на следующем кадре прямоугольную область, которая наиболее как изображение похожа на ту же самую область на предыдущем кадре. Поскольку видео достаточно быстрое, деформации объектов небольшие, этот метод работает достаточно хорошо и на коротких последовательностях он позволяет отследить объекты.

Если мы рассматриваем какую-то практическую систему, например, систему видеонаблюдения с идентификацией человека по лицу, такое простейшее сопоставление используется для того, чтобы применять детектор лиц не на каждом кадре, а, например, каждые 5 кадров. Каждые 5 кадров применяется детектор, а в промежуточных кадрах отслеживается лицо как маленький фрагмент через метод сопоставления шаблонов.

GOTURN

Для этой задачи мы можем сделать достаточно простой нейросетевой метод. Сделаем нейросеть, которая на вход будет получать 2 кадра: предыдущий кадр, в котором искомый объект располагается строго в центре кадра, и новый кадр, вырезанный из второго изображения с центром в предыдущем положении объекта, где объект сместился. Задача такой нейросети найти bounding box объекта на новом кадре. Метод, реализовавший это схему, получил название GOTURN (Generic Object Tracking Using Regression Networks, обобщенный трекинг объекта с использованием регрессионных сетей, регрессионных поскольку мы находим bbox объектов) (рис. 152).

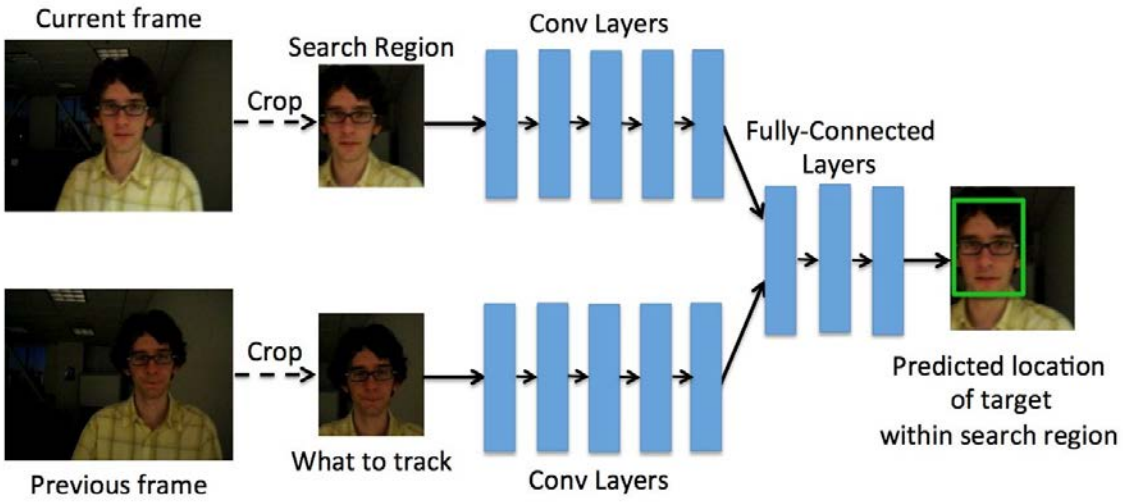


рис.152

Метод простой, работает на паре кадров и за счет этого имеет высокую скорость 100 кадров/сек.

Архитектура сети может быть разной, в работе (рис. 153) была предложена следующая:

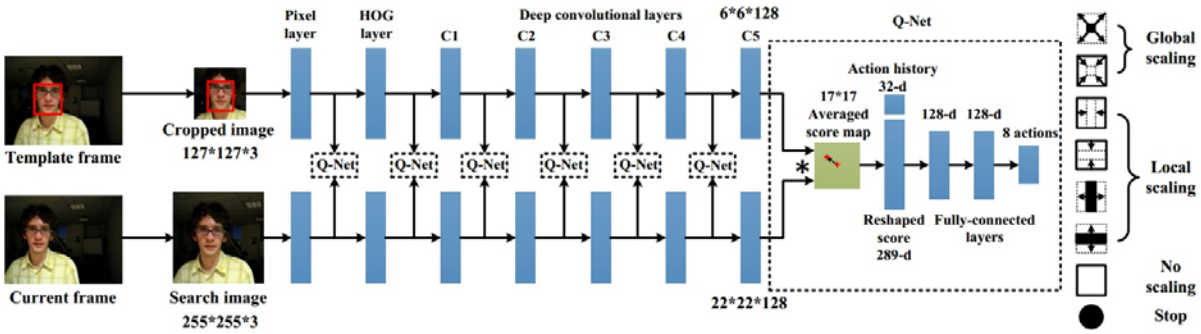


рис.153

Строим признаки изображения обоих кадров, проводя их через набор сверточных слоев, объединяем эти признаки и подаем на вход полносвязной сети, которая предсказывает bbox объекта.

В чем плюс для метода обучаться только на паре кадров? В том, что нам не нужно видео для обучения трекинга объектов. Достаточно брать кадры с известными bbox объектами, то есть любую выборку с размеченными объектами. Дальше мы можем к этим кадрам применять любые преобразования (двигать, поворачивать, добавлять гомографию). Мы знаем расположение нового объекта, смещение и на этой паре учим отслеживать движение объектов. Поскольку обычно смещения маленькие, то мы генерируем смещения неравномерно: маленькие сдвиги генерируются чаще, чем большие сдвиги. В итоге такая сеть работает достаточно неплохо.

Сопровождение множества объектов
Multiple Object Tracking (MOT)

Сопровождение множества объектов более практически применимая задача, которая описывает сценарий видеонаблюдения. У нас имеется множество объектов из нескольких или одного класса заданных объектов, например, люди, которых мы хотим отслеживать на длительных промежутках времени (камера видеонаблюдения может работать сутками). Есть два типа методов, решающих такую задачу:

1. Detection Based Tracking (DBT) – методы на основе детекторов
2. Detection Free Tracking (DFT) – методы без детекторов

Мы будем рассматривать методы только на основе детекции, посокльку они сейчас реально применяются на практике.

На выходе MOT мы хотим получть набор траекторий объектов для всех объектов, анблюдаемых в видео. В классичесом множественном сопровождении положение каждого объекта описывается с помощью bbox, но сейчас наблюдается постепенный переход от детектирования объектов с помощью bbox к выделению найденных оюъектов с помощью масок.

В принципе задачу MOT можно рассматривать как обобщение задачи детектирования объектов, но на случай видео. То есть нам нужно из видео извлечь все экземпляры объектов, которые в нем втсречались, и для каждого экземпляра показать его траекторию. Если обычно выделение – это множество bbox, то теперь выделение – это множество траекторий bbox. Поскольку это траектории, появляются дополнительные ошибки в детектировании: *переключение идентификатора* и *фрагментация*.

Переключение индентификатора (ID switches) – для одной эталонной траектории выдается две или более траекторий

Фрагментация (fragmentations) – для одной эталонной траектории выдаеся две траектории с пропусками между ними, то есть объект не виден на ряде кадров

Поэтому метрики, которые оценивают надежность отслеживания объекта должны учитывать все типы ошибок. Стандартной метрикой являеется метрика MOTA Multiple Object Accuracy (надежность построения траектории).

MOTA рассчитывается как 1 минус сумма всех ошибок к сумме эталонных bbox.

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t}$$

Поскольку число ошибок в каждом кадре может быть произвольным, это отношение может быть больше 1, и метрика Мота еняется от 1 до -∞. Самые плохие методы уходят в отрицательную область, идальный метод без ошибок будет выдавать 1.

Были придуманы и другие метрики, которые оценивают качество сопровождения:

Multiple Object Tracking Precision (MOTP) – точность локализации объектов

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_y}$$

Например, все траеткории можно распределить на классы по качеству их сопровождения:

- Mostly tracked (MT)

Объект успешной сопровождался > 80% длины эталонно траектории

- Mostly lost (ML)

Объект успешно сопровождался < 20% длины эталонной траектории

- Partially tracked (PT)

Все остальные

Наша задача создавать такие методы, у которых большинство траекторий было в целом отслежено, а доля в целом потерянных была как можно меньше

Оценка качества MOT тоже крутится вокруг конкурсов, самый крупный из которых MOTChallenge. В этом конкурсе объединяют данные, полученные из разных датасетов, и предоставляют исследователям помимо видео эталонное обнаружение. Есть коункрс по трекингу и конкурс из детекции. В 2015-2017 году было 15948 кадров (645 секунд видео) и 112297 bbox.

Самой масшттабой коллекцией для MOT была коллекция Duke multi-target, multi-camera.

В этом наборе данных исследователи проаннотировали 1,5 часа видео с 8 статичных камер видеонаблюдения, установленны в университете Дьюка.Они разметили 2 000 000 кадров и обнаружи 2000 уникальных людей. Этот датасет по числу людей превосходил все датасеты для MOT вместевзятые. Однако в 2019 году этот датасет изъяли из общего доступа, потому что никто не получил разрешение у этих 2000 людей.

Очень много датасетов собираются из фотографий публичных лиц, выложенных в общий доступ, и из фотграфий обычных людей, которые выкладывают свои фотографии под лицензией creative commands, которая допускает свободное использование фотографий в целях создания искусства. Вопрос, который решается в сенате США: считается ли разработка биометрических систем искусством? – остается открытым. Microsoft одну из своих больших эталонных коллекций в миллион изображений снял по этим же причинам.

Друг пример эталонной коллекции – UA-DETRAC Benchmark, собранная в Китая и отслеживающая автомобили. В ней 10 часов видео, 8 000 отслеживаемых автомобилейб 1,2 M bbox.

Как методы в принципе утсроены? Поскольку у нас есть детектор, мы можем решать задачу MOT чересзадачи ассоциации. Мы применяем детектор либо к каждому кадру, либо к набору ключевых кадров, затем нам нужно сопоставить обнаружение на текущем кадре с уже имеющимся набором траекторий, то есть объединить траектории в треки.

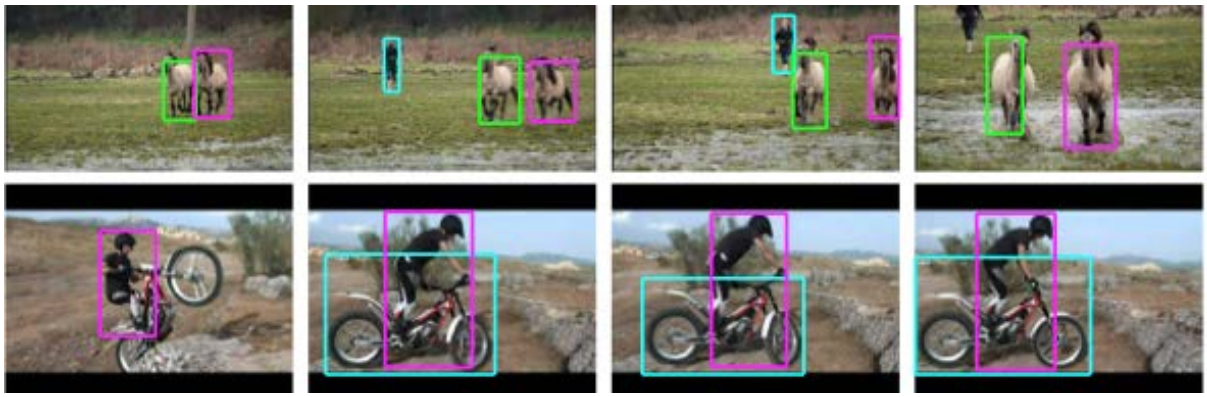


рис.154

Например, есть 2 кадра с лошадьми и нужна определить, какая лошадь соответсвует какой лошади на предыдущем кадре. При этом метод сопровождения использует метод детектирования объектов в основном как источник внешних данных, и у детектора могут быть ошибки: ложные срабатывания и пропуски ообнаружений. Хороший детектор должен устранять пропуски в детекции и фильтровать ложные срабатывания. Помимо сопоставляения обнаружения могут быть разные сложные ситуации, например, ситуации появления нового объекта (появление человека на кадре с лошадьми рис. 154). В таких ситуациях нужно инициализировать новую траекторию для объекта. Также может быть ситуация пропадания объекта из кадра, в этом случае он либо вышел из кадра, либо зашел в портал (услов дверь), либо перекрылся другим объектом и вреенно пропал. Поэтому в каждый момент времени нам нужно принимать решение: мы можем счесть,что это либо лжное обнаружение, либо инициализировать новую траекторию, либо присоединить это обнаружение к какому-то другому обнаружению. Также нам нуждно решать, что делать с теми траекториями, которые не были сопоставлены с текущим кадром: либо продолжить их. Сказав, что объект временно пропал, либо завершить их.

Пару лет назад на конкурсах было сделано важное открытие о том, что конкурсы не совсем честные и ключевым элементом в работе алгоритмов являеятся детектор. Детектор, который авторы конкурса представляли, не видел 18% траекторий и 37% траекторий были покрыты только обнаружениями с низким скором (ненадежные). Значит, по умолчанию трекеру не удается добавить новое обнаружение.

Tracker	Avg Rank	↑MOTA	IDF1	MT	ML	FP	FN	ID Sw.	Frag	Hz	Detector
DH_TRK	15.7	**54.1** ±13.0	49.2	21.6%	**28.4%**	38,196	**216,670**	5,918 (96.1)	7,760 (126.0)	1,775.7	Public
1.											Anonymous submission
HIK_MOT17	11.3	53.9 ±13.7	54.3	**23.7%**	32.0%	27,656	230,042	2,366 (40.3)	4,192 (70.8)	5.4	Public
2.											
NOTBD	17.8	53.9 ±12.7	51.2	21.5%	35.6%	28,912	228,356	2,964 (49.8)	3,600 (60.5)	0.3	Public
3.											Anonymous submission
yt_face	14.8	52.6 ±13.1	51.5	23.0%	35.9%	23,894	241,489	2,047 (35.8)	2,827 (49.4)	2.2	Public
4.											Anonymous submission
SST	21.8	52.4 ±15.9	49.5	21.4%	30.7%	25,423	234,592	8,431 (144.3)	14,797 (253.3)	6.3	Public
5.											ShiJie Sun, Naveed Akhtar, Ajmal Mian (UWA and ShiYuan Research Institute)
CNN_search	12.9	52.2 ±13.8	50.7	21.4%	33.3%	23,565	243,232	3,059 (53.8)	4,702 (82.6)	3.7	Public
6.											Anonymous submission
eHAF17	14.1	51.8 ±13.2	54.7	23.4%	37.9%	33,212	238,772	1,834 (31.6)	2,739 (47.2)	0.7	Public
7.											TCSVT-02141-2018
AFN17	15.0	51.5 ±13.0	46.9	20.6%	35.5%	22,391	248,420	2,593 (46.3)	4,308 (77.0)	1.8	Public
8.											Anonymous submission
FWT	17.3	51.3 ±13.1	47.6	21.4%	35.2%	24,101	247,921	2,648 (47.2)	4,279 (76.3)	0.2	Public
9.											R. Henschel, L. Leal-Taixé, D. Cremers, B. Rosenhahn. Fusion of Head and Full-Body Detectors for Multi-Object Tracking. In Trajnet CVPRW, 2018.
jCC	15.3	51.2 ±14.5	54.5	20.9%	37.0%	25,937	247,822	1,802 (32.1)	2,984 (53.2)	1.8	Public
10.											M. Keuper, S. Tang, Y. Zhongjie, B. Andres, T. Brox, B. Schiele. A multi-cut formulation for joint segmentation and tracking of multiple objects. In arXiv preprint arXiv:1607.06317, 2016.

рис.155

Если сравнить сложные методы трекинга, которые используют плохой детектор (рис. 155) спростыми методами трекинга, которые используют сложный детектор, то методы с хорошим детектор явно обгоняют методы с простым детектором. Поэтому честно сравнивать только те методы, которые работают на одном детекторе.

Методы трекинга можно разделить на online и offline методы.

Online метод	**Offline метод**
Для текущего кадра принимают решение, как продолжить траекторию	Пытаются найти оптимальный набор траекторий, который описывает все видео в целом
Жадный алгоритм	

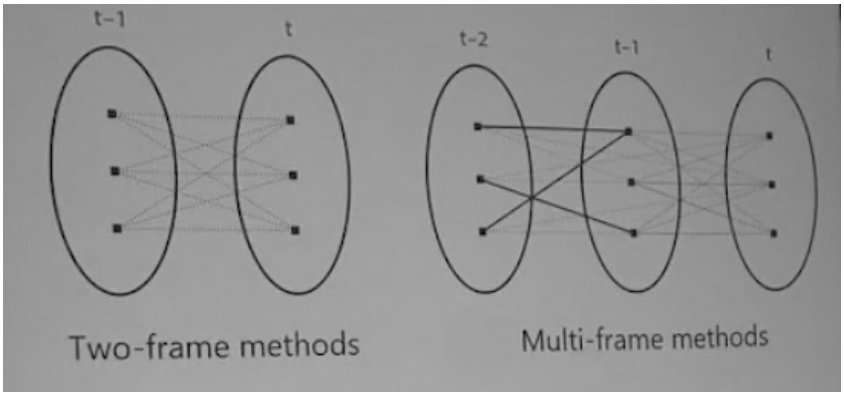


рис.156

Чаще всего используются двухкадровые методы (рис. 156), когда есть 2 ключевых кадра и обнаружение в текущем кадре соединяем с траекториями из этого предыдущего кадра. Могут быть и многокадровые методы, когда мы даем себе время подумать, отслеживаем несколько кадров и проводим траектории уже через это временное окно. Мы срабатываем с задержкой, но зато можем разрешить некоторые сложные случаи.

Компоненты функции сходства

Affinity

Важным компонентом методов отслеживания является функция сходства. Нам нужно обнаружение на текущем кадре проассоциировать с обнаружением на предыдущем кадре, оценить степень их похожести. Для этого использовали всевозможные факторы: динамические модели (насколько хорошо по траектории движения новые обнаружения соответствуют), внешность объекта (насколько визуально текущий объект похож на объект, отслеживаемый ранее), некоторое взаимодействие и перекрытие, например, мы не можем один объект проассоциировать двум предыдущим траекториям (рис. 157)

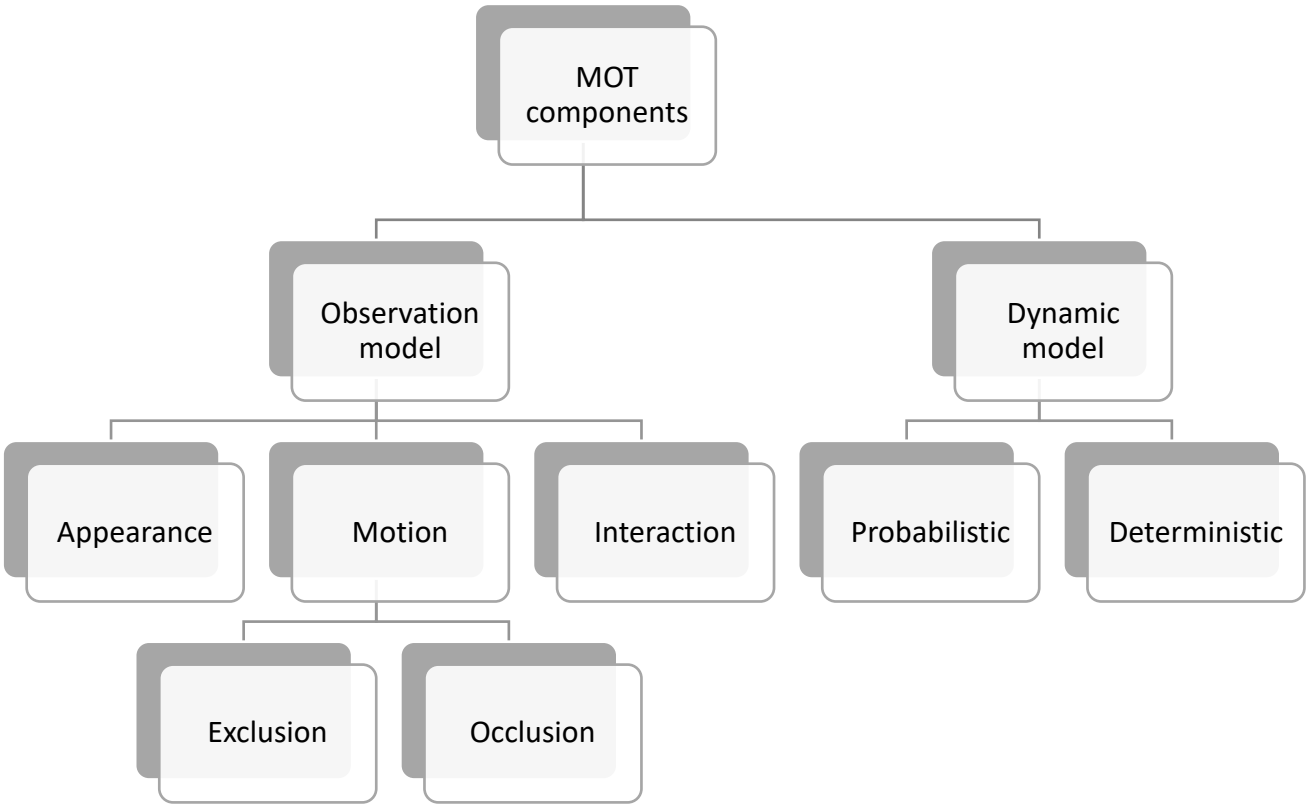


рис.157

Достаточно высокие результаты показывает максимально простой метод, в котором в качестве критерия похожести объектов берется площадь пересечения bbox объектов, то есть мы на текущем кадре сопоставляем box с предыдущем кадром и сопоставляем объект с той траекторией, с которой пересечение по bbox максимальное (рис. 158).

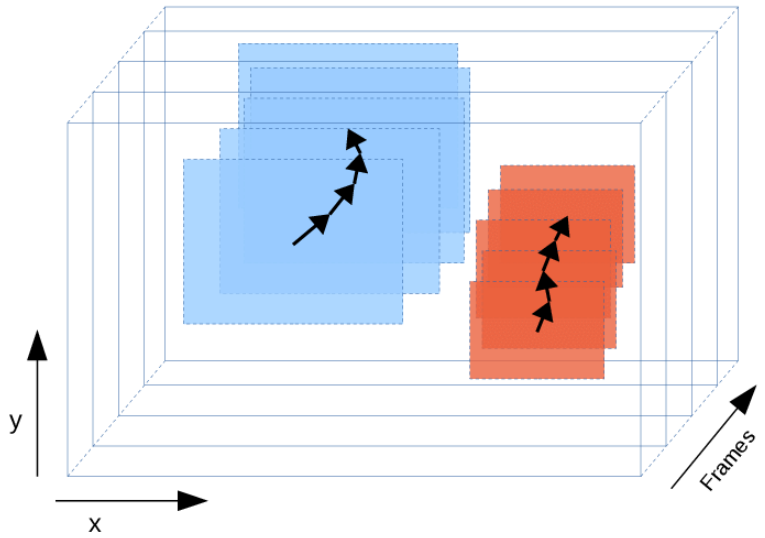


рис.158

Этот метод был открыт только в 2017 году. Оказалось, что при использовании мощного детектора, качество работы этого метода очень высокое. Ниже представлен год всего алгоритма (рис. 159). По точности этот метод легко конкурирует со сложными методами (рис. 160). Он показал эталонные результаты на коллекции DETRACK для отслеживания автомобилей, при этом он показал феноменальную скорость (самые медленные в то время давали скорость 0.71 fps, а он показывал 100 840 fps.

Simple Online and Realtime Tracking (SORT)

- CNN-based detector
- Фильтр Кальмана для предсказаний новых положений bbox в кадре
- Венгерский алгоритм для сопоставления текущего положения объекта с предсказанным
- IoU в качестве компоненты похожести

Такой простой метод мы можем усовершенствовать, например, используя не жадный алгоритм для поиска сопоставлений, а более сложный, например, венгерский алгоритм для паросочетаний назначений. Объекты будем сопоставлять по bounding box, но не с предыдущим положением объекта, а с предсказанным по динамической модели, то есть для каждой траектории мы будем обучать *фильтр Кальмана* (Kalman filter), линейный фильтр, чтобы предсказывать новое положение bounding box в кадре. В качестве меры похожести будем использовать пересечение предсказанного положения с текущим положением. Это повышает точность при большой скорости кадра.

Этот метод работает медленнее предыдущего метода (IoU), но он повышает интегральные качества работы метода и увеличивает долю в целом отслеженный траекторий. Увеличение доли отслеженных траекторий происходит по большей части из-за предсказания. То есть если у нас есть близко расположенные объекты, при смещении к следующему кадру мы можем эти траектории перепутать. Например, в кадре близко идут 2 человека, и bbox второго человека оказался на предыдущем месте первого человека. Но если мы будем предсказывать их траектории, то мы будем видеть, что они идут параллельно и будем ассоциировать каждого со своим bbox.

Algorithm 1 IOU Tracker

Inputs:
$D = \{D_0, D_1, ..., D_{F-1}\} =$
$\{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
Initialize:
$T_a = \emptyset, T_f = \emptyset$
$D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} | D_j \in D\}$
for f = 0 to F :
  for $t_i \in T_a$:
    $d_{best} = d_j$ where $max(IOU(d_j, t_i)), d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$:
      add d_{best} to t_i
      remove d_{best} from D_f
    else
      if $highest\_score(t_i) \geq \sigma_h$
        and $len(t_i) \geq t_{min}$:
          add t_i to T_f
      remove t_i from T_a
  for $d_j \in D_t$:
    start new track t with d_j and insert into T_a
for $t_j \in T_A$:
  if $highest\_score(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
    add t_i to T_f
return T_f

рис.159

Tracker	Detector	PR-MOTA	PR-MOTP	PR-MT	PR-ML	PR-IDs	PR-FM	PR-FP	PR-FN	Speed
Overall (Easy + Medium + Hard)										
CEM [1]	CompACT	5.1%	35.2%	3.0%	35.3%	**267.9**	**352.3**	**12341.2**	260390.4	4.62 fps
CMOT [3]	CompACT	12.6%	36.1%	16.1%	18.6%	285.3	1516.8	57885.9	**167110.8**	3.79 fps
GOG [12]	CompACT	14.2%	37.0%	13.9%	19.9%	3334.6	3172.4	32092.9	180183.8	390 fps
DCT [2]	R-CNN	11.7%	38.0%	10.1%	22.8%	758.7	742.9	336561.2	210855.6	0.71 fps
H^2T [18]	CompACT	12.4%	35.7%	14.8%	19.4%	852.2	1117.2	51765.7	173899.8	3.02 fps
IHTLS [6]	CompACT	11.1%	36.8%	13.8%	19.9%	953.6	3556.9	53922.3	180422.3	19.79 fps
IOU	R-CNN	16.0%	**38.3%**	13.8%	20.7%	5029.4	5795.7	22535.1	193041.9	**100,840 fps**
IOU	EB	**19.4%**	28.9%	**17.7%**	**18.4%**	2311.3	2445.9	14796.5	171806.8	6,902 fps
Easy (AVSS17 Beginner Challenge)										
CEM [1]	Average	7.2%	42.0%	3.4%	42.3%	96.4	119.6	4253.3	63296.3	-
CMOT [3]	Average	16.6%	43.8%	20.0%	23.0%	**68.3**	327.9	16282.2	39467.6	-
GOG [12]	Average	20.1%	44.6%	17.5%	24.0%	1019.0	981.2	8423.5	42065.9	-
DCT [2]	Average	16.3%	44.1%	9.6%	34.3%	73.5	**69.4**	4754.6	51393.3	-
H^2T [18]	Average	17.1%	42.9%	17.8%	24.7%	298.8	305.5	12866.0	42086.5	-
IHTLS [6]	Average	14.8%	43.0%	15.5%	25.3%	299.5	1102.3	13839.9	43954.4	-
IOU	R-CNN	29.3%	**47.2%**	25.0%	**17.3%**	1112.5	1261.0	3457.6	**33394.1**	**117,340 fps**
IOU	EB	**34.0%**	37.8%	**27.9%**	20.4%	573.6	603.7	**1617.0**	33760.8	9,002 fps
Medium + Hard (AVSS17 Experienced Challenge)										
IOU	R-CNN	11.8%	**36.5%**	8.9%	25.0%	3693.1	4228.3	16634.7	168527.2	**87,906 fps**
IOU	EB	**16.4%**	26.7%	**14.8%**	**18.2%**	**1743.2**	**1846.3**	**12627.0**	**136077.8**	6,069 fps

рис.160

При этом оба этих метода никак не учитывают внешность объекта. Поэтому логичным шагом будет добавить в функцию похожести сопоставление по внешнему виду. Для этого мы можем обучить метод сравнений изображений или отобразим вектор

изображения человека в вектор-признак и будем сравнивать по этим вектор-признакам изображения людей. Методы сравнения людей по внешности, когда мы рассматриваем всего человека в целом, для задачи сопровождения и поиска людей в видео получили собственное название Re-identification, реиндентификация (рис. 161).

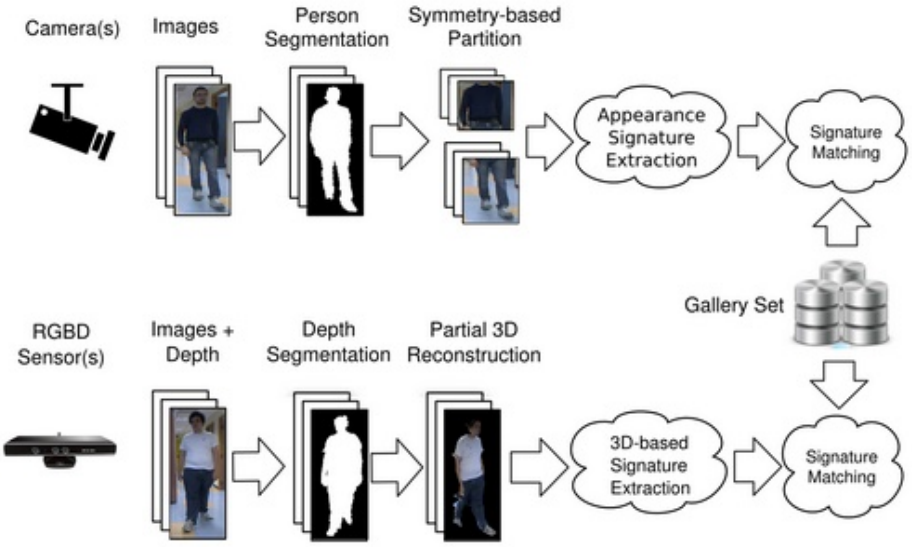


рис.161

Добавление сопоставления людей по похожести дало название методу SORT+DA (data association). За счет этого мы еще на пару процентов повышаем качество работы, увеличиваем долю в основном отслеженных траекторий, но уваливается и число ложных срабатываний. Сильно меняется параметр переключения идентификатора, поскольку теперь такие срабатывания, когда bbox пересеклись, мы сразу отсекаем, потому что дескриптору внешности оказываются непохожи. Ошибки в сопоставлении по внешности приводят к фрагментации траектории. В целом интегральная характеристика качества улучшается.

Распознавание событий

Событие – нечто комплексное, происходящее в видео. У распознавания событий множество практических применений, например, индексация и поиск видео по коллекциям, умное видеонаблюдение и умная видеоперемотка. Что подразумевается под действиями?

Вначале под действием подразумевали базовые атомарные движения человека, например, бег, бег трусцой, ходьба, боксирование и т.д. Это может быть более сложное событие, например, пожать руку или ответить на телефон. Наконец множество маленьких действий превращается в одно длинное действие с определенной цель, например, сделать бутерброд. Множество действий, в которых участвует множество людей и которые могут занимать длительный промежуток времени, превращаются в *событие,* например, отмечание дня рождения, парад итп.

Для распознавания событий можно сформулировать задачу, похожую на задачу классификации и детекции. Задача классификации – определить, присутствует ли в видеособытие заданной категории, то есть задать метку «присутсвует /отсутсвует». Задача детекции – пространственно-временная локализация события, то есть, где именно происходит действие как по времени, так и в пространстве.

События и действия тесно связаны с движением, их можно описать движением. Например, событие типа хлопок в руки или бег – это события, которые определяются действиями вне зависимости от контекста. Другие события существенно опираются на контекст, например, действие поворачивания чего-то рукой в зависимости от контекста может быть открыть дверь, завести машину или открутить болт. То есть действие с точки зрения движения одинаково, если мы будем рассматривать оптический поток, мы увидим одно и то же действие, но смысловая нагрузка и интерпретация будут разными.

В отличие от задачи оптического потока и задачи трекинга составлять эталонные коллекции для распознавания событий очень легко, поскольку эталонная коллекция — это видео и метка, нам нужно лишь сопоставить одну метку одному видео.

Первая эталонная коллекция для распознавания действий, которая собиралась, — это коллекция KTH Actions (2004) из 2,5 тысяч видео, которые содержат базовые атомарные движения человека.

Затем достаточно быстро перешли на составление коллекций из роликов YouTube. Например, коллекция UCF 101 (2012) содержит 13 тысяч видео 101 класса разного типа:

1. Взаимодействие человека с объектами
2. Движение
3. Взаимодействие людей
4. Игра на музыкальных инструментах
5. Спортивные события

Это типичный пример коллекции, которая с одной стороны стимулирует развитие методов, а с другой – быстро насыщается. Ее опубликовали в 2013 году, точность классификации была 43,9%, через год точность стала 87,9%.

В качестве тяжелой коллекции можно привести коллекцию TrecVid MED’13 – большой европейский конкурс, посвящённый аннотации видео. В нем 100 положительных примеров на каждую из 20 категорий и 5000 отрицательных примеров на каждую категорию. Тестирование проводилось на 4000 часов видео, то есть тестировать алгоритм на датасете – само по себе тяжелая задача. Зато в конкурсе можно использовать любую информацию, в частности можно распознавать аудио, речь и т.д.

Вернемся к задаче классификации действий – отображения видео в метки. Как мы будем распознавать действия? Оказывается, что решить задачу классификации действия мы зачастую можем решить по одному кадру (рис.162). Красить глаза, брить бороду, заниматься серфингом – нам не нужно видео, чтобы определить эти действия,

достаточно взять один кадр, применить к нему мощный классификатор и обучить. С появлением нейросетей такой метод распознавания событий оказалось сделать очень просто. Мы берем центральный кадр и его классифицируем, поверх этого мы пытаемся усилить базовый классификатор за счет учета временной информации видео. Но некоторые события выучить по одному кадру нельзя. Это, кстати, является качественным показателем датасета. Если базовое качество распознавания по центральным кадрам не очень высокое, значит, датасет для классификации действий сложный.

рис.162

Зачастую в датасеты вкрадывается какая-то систематическая ошибка, систематические данные, которые можно использовать. Например, датасеты для распознавания лица иногда тестируют следующим образом: закрывают всю область лица и обучают классификатор. Если ошибка классификатора не очень высокая, значит что-то кроется в фонах, что может учитываться.

Как подходят к распознаванию? Сначала были эвристические методы, которые просто обобщали эвристический подход к построению признаков к пространственно-временному объему. Изображение делилось на области, в которых считалась гистограмма распределения признаков, например гистограмма распределения градиентов. Теперь у нас пространственно-временной объем, который мы будем делить на кубики и в каждом кубике будем считать гистограмму распределения градиентов. Поскольку теперь есть движение, мы можем считать не только гистограмму

распределения градиентов, но и гистограмму распределения пикселов по направлению вектора оптического потока, то и другое конкатенировать и использовать совместно.

Минусом такого подхода является то, что мы разбиваем видео на кубики (рис. 163). У нас объекты двигаются, соответственно сначала часть объекта была в одном кубике, потом она перешла в другой кубик, и это затрудняет распознавание. Одно из направлений развития — это более детальный учет движения объектов в сцене. Например, вместо того чтобы делить на кубики мы можем сделать следующее: найдем множество ключевых точек, будем отслеживать их движение и строить кубики вдоль траектории движения ключевых точек. То есть поскольку точка соответствует скорее всего одной и той же части сцены, например, какой-то точке на теле человека, когда мы будем собирать характеристики траектории движения и характеристики движения в окрестности точки, мы будем учитывать только эту конкретную область объекта, и она будет более показательна для распознавания. Этот метод получил название Плотная траектория Dense trajectories (рис. 164).

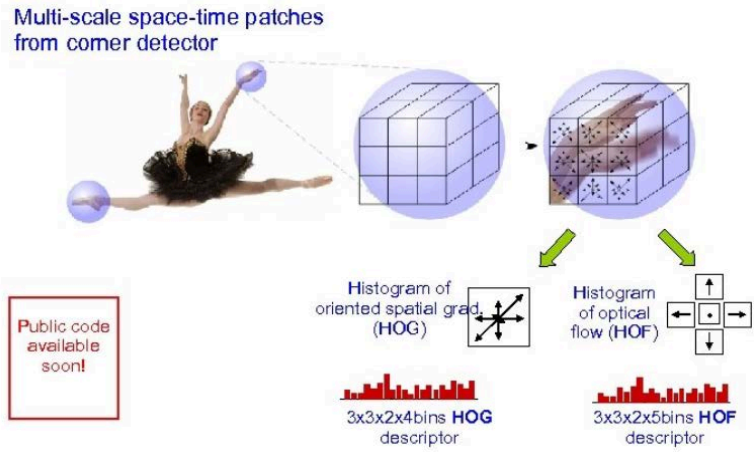


рис.163

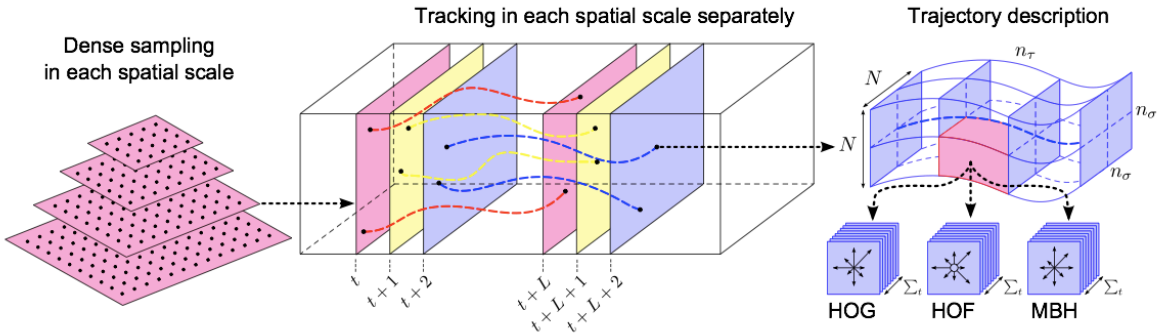


рис.164

Потом появились нейросети. Некоторым базовым методом распознавания событий с использованием нейросети можно считать двухпоточный метод (Two-stream CNN-model) (рис. 165), который объединил в себе распознание по одному кадру и распознавание движения.

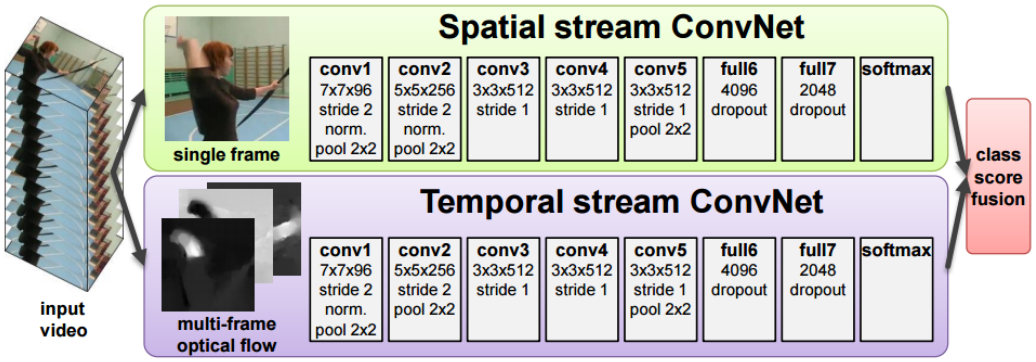


Figure 1: **Two-stream architecture for video classification.**

рис.165

На вход классификатору подаются карты оптического потока, которые кодируют только движение, не зависят от цветов объектов и тп). Для того, чтобы сеть обучилась, мы выбираем некоторое небольшое количество кадров из видео (5-10 кадров), считаем карты оптического потока, все это объединяем вместе в матрицу глубины 20 ($w \times h \times 2L$) и подаем на вход сверточной нейросети. Результаты мы объединяем и получаем итоговый результат.

Дальнейшее усовершенствование этого метода возможно за счет объединения с плотными траекториями (Dense trajectories + Deep features) (рис. 166). Будем отдельно учитывать признаки внешности и движения и собирать эти признаки не просто по всему изображению, а вдоль траектории движения точек. То есть извлечем из изображения набор плотных траекторий, для каждого кадра посчитаем признаки оптического потока и признаки нейросети по RGB изображениям и будем усреднять эти признаки вдоль траектории движения особой точки. Получим набор признаков, который затем будем классифицировать. За счет такого многоступенчатого учета движения отдельно через оптический поток, отдельно за счет движения ключевых точек, мы получим лучшее качество распознавания.

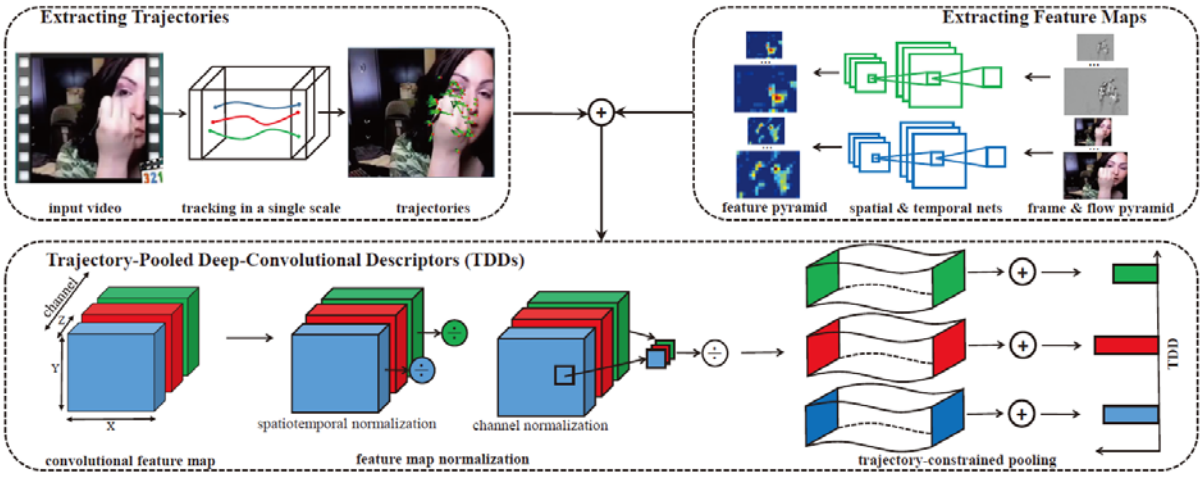


рис.166

Сейчас одно из направлений развития методов распознавания событий – обучение self-supervised, когда нет эталонной обучающей выборки и мы заставляем метод самому выбирать из данных то, что он хочет предсказывать. Эта идея пришла из области анализа текстов, в которой сейчас происходит большая революция, связанная с сетями-трансформерами и самообучающимися методами. Идея такая: текст связанный, в нем есть некоторая последовательность слов, значит мы можем сформулировать задачу обучения предсказывания новых слов (какое слово или предложение пойдет в тексте следующем). Если мы научимся решать эту задачу хорошо, значит нейросеть научилась извлекать из текста смысл. И тогда мы сможем использовать эти признаки, которые сеть извлекает из сети для того, чтобы на другой новой размеченной коллекции быстренько обучить сеть. Например, мы обучили сеть, которая предсказывает предложения, обученные на википедии, затем берем маленькую выборку новостных заметок или отзывов на кинофильмы положительные, отрицательные и нейтральные, собираем с помощью этой сети признаки и обучаем простой классификатор, который предсказывает отношение к заметке: положительное оно или отрицательное. Эти сети хорошо учатся, поскольку датасетов из текстов огромное количество.

Что-то похожее мы можем сделать и для видео, ведь видео – это тоже упорядоченная последовательность. Можно делать сети, которые будут предсказывать новые кадры, фрагменты. Если сеть научилась предсказывать новые фрагменты, значит она научилась извлекать содержание и знает все продвижение. После этого мы можем применить эту сеть к новому набору видео и использовать извлечённые ей признаки как признаки для классификации. Эксперименты показывают, что это сильно улучшает качество распознавания.

Обученную сеть для извлечения признаков можно использовать для обучения классификаторов с учителем или как признак для поиска похожих видео.

Резюме

- Основные задачи обработки видео – оценка оптического потока, сопровождение объектов, распознавание событий
- Сейчас они все решаются нейросетевыми моделями
- Движение нужно учитывать отдельно и желательно детально (вроде dense trajectories) учитывать оптический поток
- Сейчас активно развиваются self-supervised методы

Лекция 10. Разреженная трехмерная реконструкция

3D реконструкция по изображением

У нас имеется множество изображений, описывающих одну и ту же трехмерную сцену, мы хотим получить трехмерную модель, воплощающую эту сцену со всеми измерениями и с возможностью визуализировать эту модель так, чтобы итоговые картинки совпадали с исходными изображениями. Например, можно построить трёхмерную модель Колизея по набору пользовательских фотографий из интернета.

Вообще задача трехмерной реконструкции считается одной из наиболее проработанных задач в области компьютерного зрения, но это не совсем так: хорошо проработанной до недавнего времени считалась задача *разреженной трехмерной реконструкции*. Многие методы сейчас применяются на практике, однако трехмерные модели, которые получаются, из-за своего визуального качества не могут использоваться на практике.

Разреженная трехмерная реконструкция означает, что мы будем работать не со всем изображением в целом, а с отдельными точками изображения, например, найденных с помощью детектора особых точек. Для особых точек мы можем построить уникальное описание (дескрипторы) и уверенно сопоставлять эти точки между разными изображениями.

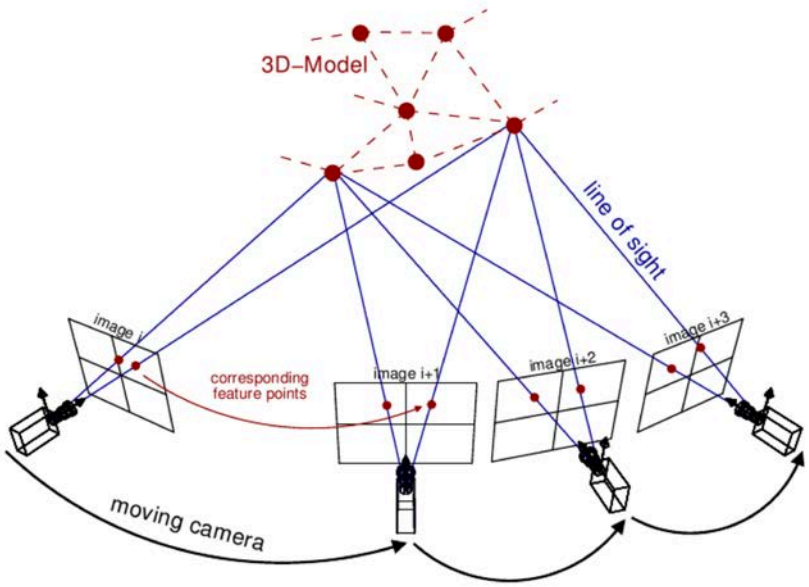


рис.168

Разреженная реконструкция проводится в форме облака точек (рис. 167), которые соответствуют найденным ключевым точкам на изображении. Мы будем решать совместно задачу оценки трехмерных координат точек и задачу определения положения камер в пространстве относительно этих ключевых точек.

В целом задача реконструкции – обратная задача к задаче компьютерной графики. В компьютерной графике мы строим изображения, соответствующие трехмерной модели, а в задаче реконструкции наоборот – строим трехмерную модель по изображению.

Начнем рассматривать модель формирования изображения с модели одной камеры. Сначала происходит преобразование из мировой системы координат в систему координат камеры. Это преобразование будет определяться положением и ориентацией камеры в пространстве. Далее происходит перспективная проекция, параметром которой является фокусное расстояние, то есть фактически расстояние между центром проекции и плоскостью, на которой формируется изображение. После этого происходит дискретизация, и изображение с картинной плоскости дискретизируется, и мы получаем цифровое изображение.

Сначала построим модель перспективной проекции (говорили о ней ранее) с учетом того, что теперь наша мировая система координат соединена с системой координат камеры. Центр системы координат камеры совпадает с центром проекции. Мы строим образы трехмерных точек X, получая их образы x на изображении. Система ориентирована таким образом, что оптическая ось совпадает с осью Z, оси X и Y ориентированы по строкам и столбцам изображения (рис. 168).

$x = f\frac{X}{Z}$ $\quad y = f\frac{Y}{Z}$ - простейшее уравнение перспективной проекции (нелинейное преобразование)

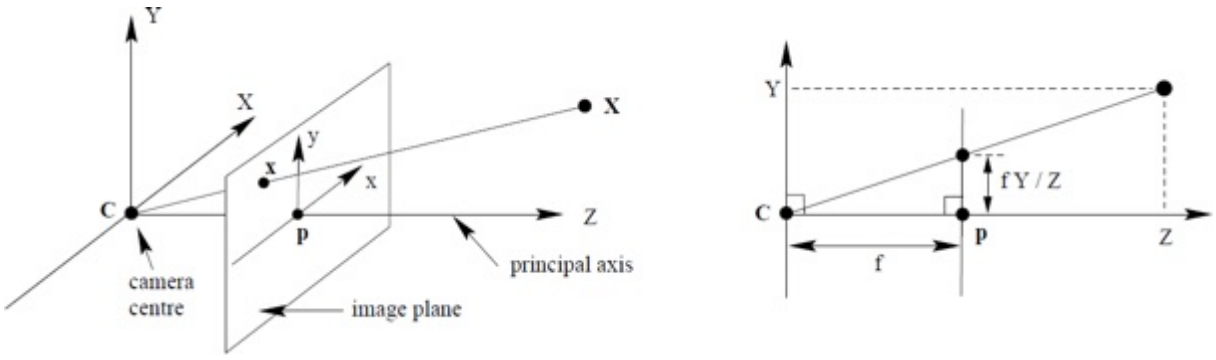


рис.167

C = (0,0,0) – центра камеры (проекции)

X = (X,Y,Z) – точка в трехмерном пространстве

x = (x,y,f) – проекция X на картинную плоскость

C, x, X лежат на одной прямой

Мы можем записать уравнение перспективной проекции в матричном виде, если перейдем к однородным координатам. Для того, чтобы перейти к однородным координатам нужно домножить все 3 компоненты на некоторый коэффициент α, и этот коэффициент будет четвертой координатой. То есть трехмерно точке XYZ будет соответствовать точка в однородных координатах XYZ_1.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cong \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$P = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

K $\qquad\qquad$ $\langle I|0\rangle$

Для удобства мы представляем эту матрицу перспективной проекции представляем в виде произведения двух матриц: матрицы К и единичной матрицы перспективной проекции. Единичную матрицу перспективной проекции можно представить как матрицу перспективной проекции с фокусным расстоянием = 1, то есть это некая идеальная картинная плоскость, которая располагается в системе координат камеры на расстоянии 1 от центра проекции.

Матрица К – матрица внутренней калибровки, поскольку это параметры, которые связаны непосредственно с камерой и не зависят от положения ориентации камеры в пространстве.

Если мы объединим все преобразования вместе, то есть переход из мировой системы координат в систему координат камеры, центральную проекцию, внутреннею калибровку, включая отображение в пикселы, получится полное уравнение перспективной проекции (рис.169).

рис.169

$$C^{-1} = \left(\begin{bmatrix} R & T \\ [0,0,0] & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} R^T & -R^T T \\ [0,0,0] & 1 \end{bmatrix}$$

• Внешняя калибровка определяется положением и ориентацией камеры в пространстве

$$K = \begin{bmatrix} f/pix & 0 & c_x \\ 0 & f/pix & c_y \\ 0 & 0 & 1 \end{bmatrix} \leftarrow$$ Внутренняя калибровка!

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cong K * I * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Текущее уравнение центральной проекции

Матрица 3x4 будет считаться матрицей перспективной проекции, если ее можно представить в виде композиции 3 матриц: матрицы внутренне калибровки (К), центральной проекции ($\langle I|0\rangle$) и внешней калибровки (C^{-1}).

С – ортогональная матрица, т.к. преобразование между евклидовыми системами координат

К – верхнетреугольная матрица

Полная версия матрицы внутренней калибровки (рис. 169) – это матрица с 5 параметрами, где первые два диагональных элемента – это фокусное расстояние, деленное на размер пиксела. При этом происходит отображение из системы координат камеры в номер пиксела в изображении. c_x, c_y – положение принципиальной точки. Принципиальная точка – точка, в которой оптическая ось, вдоль которой мы наблюдаем, пересекает матрицу проекции.

После того, как мы отобразили точку на картинную плоскость, нам нужно посчитать ее значение в пикселах в системе координат изображения. Для этого нужно посчитать смещение и произвести перевод из единиц длины в системе координат камеры в пикселы изображения.

Иногда матрица внутренней калибровки выглядит сложно, потому что пикселы могут быть не квадратными, а прямоугольными, в этом случае первые два диагональных элемента будут разными: первый – фокусное расстояние, измеренное в ширинах пиксела, второй- фокусное расстояние, измеренное в высотах пиксела. Еще пикселы бывают скошенными. Но обычно мы считаем пиксель приближённо квадратным, и

тогда первые два диагональных элемента будут равны между собой и равны фокусному расстоянию в мм, деленному на размер пиксела.

Матрица С – это ортогональная матрица, задающая преобразование из мировой системы координат в систему координат камеры, она задана в однородных координатах. Преобразование положения в однородных координатах задается матрицей поворота 4x4 и вектором сдвига. Верхний левый минор 3x3 этой матрицы – это матрица поворота, Т– вектор смещения, а последняя строка – это 0001.

Матрицу внешней калибровки обычно записывается в виде обратной матрицы к матрице, задающей положение ориентации камеры в пространстве. То есть если Т – это координаты камеры в мировой системе координат, вектор поворота, то для того, чтобы перейти от мировой системы координат в систему координат камеры, нужно взять обратное преобразование. Из-за того, что R – ортогональная матрица, мы можем расписать матрицу внешней калибровки в виде $R^T - R^T$.

Внутренняя калибровка (К) задает преобразование из идеальной картинной плоскости (плоскости на расстоянии 1 в системе координат камеры) в пиксели изображения. Соответственно обратная калибровка ($К^{-1}$) отображает точки изображения на картинную плоскость. То есть мы знаем, где у нас располагается каждый пиксель в системе координат камеры и значит можем провести через них лучи и знать, на каком луче в пространстве лежит трехмерная точка, которую мы хотим восстановить. Если внутренней калибровки нет, то мы не знаем конкретного расположения точки, соответствующей данному пикселу, соответственно мы не знаем размера объекта итп.

Такое уравнение перспективной проекции верно для идеальной камеры. На практике из-за особенности оптических систем наблюдаются радиальные искажения: прямые линии, которые должны сохраняться при перспективной проекции, на самом деле превращаются в кривые линии. Причем степень их искривления пропорционально их удаленности от принципиальной точки, то есть чем дальше точка от принципиальной, тем существеннее искривление. Сейчас стараются сделать такую оптику, чтобы радиальные искажения были минимальны. Но чем шире угол обзора камеры, тем сложнее это сделать. Из-за высокого разрешения современных камер и хорошей оптики, радиальные искажения на глаз видны слабо, однако реальные смещения точки могут быть достаточно существенны. Если мы хотим обеспечить точную трёхмерную реконструкцию, нам нужно обязательно учесть радиальную дисторсию.

Есть несколько моделей, которые задают радиальную дисторсию. Чаще всего используется модель дисторсии Tsai, в которой измерены координаты точки $\hat{x}$ рассчитываются как истинные координаты x плюс некоторая поправка, зависящая от радиуса, то есть расстояния от точки до принципиальной точки, причем эта поправка моделируется полнимом четвертой степени в зависимости от радиуса.

$\hat{x} = x + L(r)x \qquad r^2 = x^2 + y^2$

$\hat{y} = y + L(r)y \qquad L(r) \approx k_1 r^2 k_2 r^4$

Обычно оценка радиальной дисторсии для камеры делается в лабораторных условиях. Специально снимается калибровочный шаблон с прямыми линиями, про который все известно, и подбираются так его параметры. Чтобы радиальная дисторсия устранилась.

В принципе радиальную дисторсию можно устранить, если сфотографировать реальный объект, в котором можно определить, какие кривые линии на самом деле являются прямыми. Чаще всего это здания и всякие искусственные объекты, поскольку сцена обычно создается из прямых перпендикулярных друг другу линий.

Разреженная 3D реконструкция

Структура из движения

Мы рассматриваем некоторую статическую или динамическую сцену, наблюдая одновременно из нескольких точек, и по движению точек от наблюдателя к наблюдателю мы можем оценить расстояние от наблюдателя до этих точек и соответственно положение этих трехмерных точек в пространстве (рис. 170).

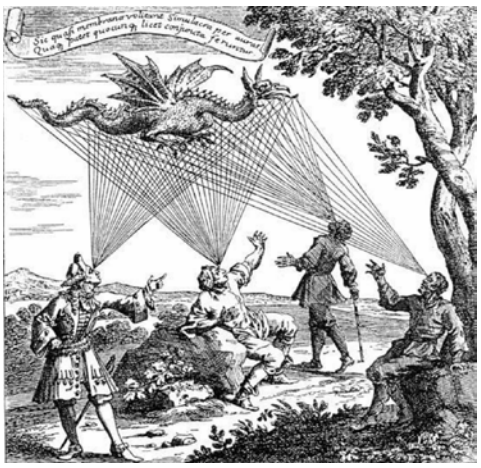


рис.170

Как видно из гравюры 1702 года (рис. 170) понятие задачи *структуры из движения* очень старое. Заметно, что на гравюре рассматривается именно разреженная реконструкция, потому что лучи направлены на отдельные точки.

Формально задачу можно описать следующим образом:

Дано m изображений n фиксированных 3D точек

$x_{ij} = P_i X_j, i = 1, \dots, m, j = 1, \dots, n$

Мы знаем проекцию каждой трехмерной точки на каждое изображение, то есть знаем множество проекций x_{ij}, где i – номер изображения, j – номер трехмерной точки. Эта

проекция получена путем проецирования неизвестной точки X_j с помощью неизвестной матрицы проекции P_i. Задача *структуры из движения* заключается в том, что нам нужно оценить m матриц P_i и n точек X_j, зная их проекции на исходные изображения (рис. 171).

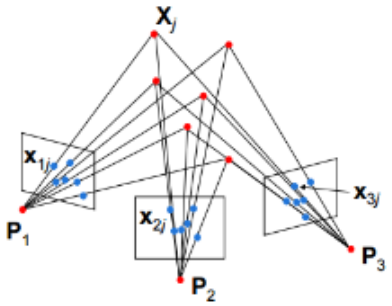


рис.171

Поскольку мы знаем только проекцию, то если у нас есть какая-то оценка матриц проекций и трехмерных точек X_j, то оценить качество реконструкции мы можем по суммарной ошибке репроекции. То есть мы строим проекции оцененных точек X_j оцененными матрицами P_i и вычисляем расстояние получившихся проекций до ранее измеренных нами точек x_{ij} и суммируем все отклонения по всем трехмерным точкам по всем изображениям.

$$E(P,X) = \sum_{i=1}^{m}\sum_{j=1}^{n} D(x_{ij,}P_{i,}x_j)^2$$

Получается нелинейная целевая функция с большим количеством неизвестных параметров: 3n параметров координат трехмерных точек, 6m параметров для матриц проекций для калиброванного случая и 11m для некалиброванного случая.

Поскольку это нелинейная целевая функция, ее можно решить оптимизационным методом, например, градиентным спуском. Но у нас много параметров, а особенности целевой функции таковы, что у не огромное количество локальных минимумов. Поэтому если мы будем начинать с произвольной точки, скорее всего попадем в локальный минимум, который будет очень далек от реального глобального оптимума. Поэтому нам нужно получить хорошее начальное приближение (именно в этом и заключалась основная проблема решения задачи структуры из движения).

До специалистов компьютерной графики задача решалась следующим образом: какие-то трехмерные точки фиксировались, например, если мы фотографируем здание, то мы можем измерить некоторые точки здания (высоту, ширину) и, зная эти реперные точки, остальные точки можем инициализировать случайным образом и далее решать задачу

градиентным спуском, надеясь, что наличие опорных точек позволит сойтись в хороший оптимум.

Что мы можем знать про изображения? По умолчанию мы знаем только соответствие особых точек. Иногда бывает ситуация, когда нам известна геометрия объекта и на поверхности этого объекта есть хорошо различимые точки, которые мы можем идентифицировать, то есть получить на вход некоторое соответствие двухмерных точек известным трехмерным точкам. Такая ситуация возникает искусственно, когда мы фотографируем какой-то известный объект. Такие объекты называются *калибровочными* и используются для калибровки камеры. Например, с их помощью мы можем оценить внутренние параметры камеры и затем при реконструкции рассматривать только калибровочный случай.

Геометрия двух камер

Для того, чтобы решить задачу, нужно посмотреть, какие геометрические особенности возникают, когда одна и та же сцена наблюдается с двух разных точек, то есть какая геометрия двух камер возникает. Геометрическая конструкция, возникающая при наблюдении одной сцены с двух точек, описывается *моделью эпиполярной геометрии* (рис. 172).

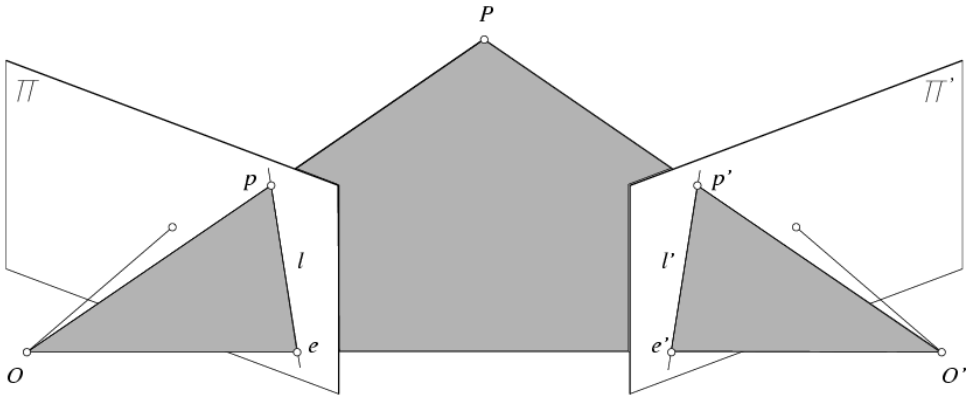


рис.172

Мы наблюдаем некоторую трехмерную точку P. Проекция P – это p на первой камере и p’ на второй камере.

Базовая линия (Baseline) – линия OO’, соединяющая центры камер (центры проекций стереопары)

Эпиполярная плоскость (Epipolar plane) – плоскость (пучок плоскостей), проходящая через базовую линию.

Базовая линия пересекается с плоскостями изображений в точках, которые называются **эпиполи** или **эпиполяры**.

Эпиполи (эпиполяры) – пересечение базовой линии с картинными плоскостями или проекция центра второй камеры на первую камеру и наоборот

Эпиполярные линии (Epipolar Lines) – линии пересечения эпиполярной плоскости с картиной (дают соответствующие пары)

Если мы возьмем одну эпиполярную плоскость, она пересчёт оба изображения по линиям, которые соответствуют друг другу. Зафиксировав эпиполярную линию на одном изображении, мы фиксируем эпиполярную плоскость и фиксируем эпиполярную линию на втором изображении. Фактически мы переходим от некоторой трехмерной или двухмерной задачи поиска соответствий к одномерной задаче соответствий.

Эпиполярные ограничения

Пусть известны 2 камеры, на одном изображении зафиксируем точку x, которая является проекцией точки X (трехмерные координаты которой мы не знаем). Что мы можем сказать о x’ – проекции точки X на второе изображение? Она лежит на соответствующей эпиполярной линии. То есть на каком бы расстоянии точка X не была от первой камеры, она будет лежать на соответствующей эпиполярной линии. Если же мы зафиксируем на втором изображении точку x’, тогда проекция неизвестной точки X на втором изображении лежит на эпиполярной линии.

Под *эпиполярным ограничением* подразумевают следующее ограничение: проекции x и x’ одной и той же трехмерной точки X (неизвестной) должны лежать на соответствующих эпиполярных линиях l и l’.

В зависимости от взаимного расположения камер ориентация пучков эпиполярных линий разная.

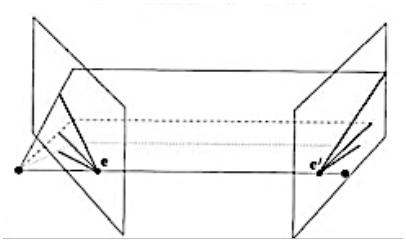


рис.173

Случай на рис. 173 – 2 камеры смотрят на одну и ту же сцену с разных ракурсов, сближающихся друг с другом. Получаются пучки наклонных эпиполярных линий. Если выровнять стереопару таким образом, чтобы картинные плоскости были параллельны базовой линии, тогда эпиполярные линии выравниваются и становятся параллельными к строкам изображений. Если камеры выравнены, то пары соответствующих эпиполярных линий будут лежать в одних и тех же строках изображений, что

существенно упрощает задачу трехмерной реконструкции, поскольку соответствующие точки должны лежать на одних и тех же строках.

Самый сложный случай для трехмерной реконструкции, когда камера движется вперед или назад. В этом случае проекция второй камеры на первую попадает внутрь изображения и эпиполярыне линии превращаются в эпиполярные лучи, то есть точка располагается где-то на луче, исходящем из эпиполи. Если камера движется вперед, проекции точек движутся к краям изображений, если назад, то их краев они сходятся в центр.

Из эпиполярного ограничения можно сделать следующие выводы (рис. 174):

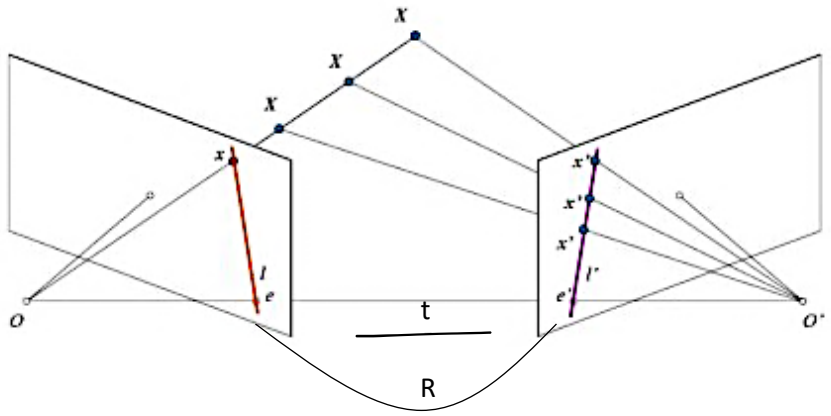


рис.174

- Линия l’ не зависит от X, если x и калибровка камер известны
- Линия l не зависит от X, если x’ и калибровка камер известна
- Между x, x’ и калибровкой камер существует зависимость, которую можно вывести

Для этого сделаем следующее:

- Совместим глобальную систему координат с координатной системой первой камеры
- Допустим, известны внутренние калибровки K, K’ и внешняя калибровка второй камеры R и T
  Умножим матрицу проекции каждой камеры на обратную матрицу калибровки K^{-1}, K'^{-1} и отобразим точки x, x’ с изображения на картинную плоскость

Поскольку мы совместили мировую систему координат с системой координат первой камеры и домножили ее на обратную матрицу калибровки, то матрица проекции первой камеры стала *единичной*, простейшая центральная проекция. В этой системе есть трехмерная точка X и ее проекция x. Запомним вектор Ox, который фактически является вектором из начала системы координат на точку на трехмерной плоскости, то есть некоторый вектор с параметрами x, y, 1. Вектор OO’ – это вектор t, поскольку t - это положение второй камеры относительно первой камеры. Вектор O’x’будет в системе координат первой камеры записываться как Rx’, потому что вектор O’x’ равен x’ в системе координат второй камеры, а в систему координат первой камеры мы его переведем, домножив на матрицу ориентации R. Соответственно получается три вектора x, Rx’ и t, которые лежат в одной и той же плоскости, то есть компланарны и мы можем вывести уравнение компланарности, записав его в следующем виде: скалярное произведение x на векторное произведение векторов t и Rx’ равно 0.

$$(x, [t, Rx']) = 0$$

- Это и сеть математическая запись эпиполярного ограничения. Мы видим, что у нас есть взаимосвязь между проекциями одной и той же точки на два изображения x и x’ и ориентацией и сдвигом второй камеры относительно первой камеры. Это условие можно переписать в виде билинейной формы следующего вида: $x^T E\, x' = 0, \quad E = [t_*]R$
- Матрица E - существенная матрица (essential matrix) была выведена в 1981 году.
- Если мы знаем существенную матрицу и зададим точку на одном изображении x’, то тогда мы выведем эпиполярную линию. То есть эпиполярная линия l задается как Ex’, эпиполярная линия l’ задается как $E^T x$. С помощью существенно матрицы мы можем вычислить эпиполярное уравнение линии соответствующей точки на втором изображении
- Домножение эпиполяра на существенную матрицу равно 0: $Ee = o, Ee' = 0$
- Существенная матрица вырождена (ранг 2) и у нее 5 степеней свободы

Если мы рассмотрим некалиброванный случай, когда матрицы K и K’ неизвестны, тогда мы сможем записать эпиполярное ограничение через неизвестные нормализованные координаты. То есть мы знаем, что существует связь между точками на картинных плоскостях, выражаемых через существенную матрицу, не знаем, как отобразить точки из пикселов изображения в координаты, но знаем, что они записываются через матрицу внутренней калибровки:

$\widehat{x^T} E \widehat{x'} = 0 \quad x = K\hat{x},\ x' = K'\widehat{x'}$

$x^T F x' = 0$, где $F = K^{-T} F K'^{-1}$ – фундаментальная матрица (1992 год)

Фундаментальная матрица обладает такими же свойствами, что и существенная матрица, только у нее 7 степеней свободы:

- l =Fx - эпиполярная линия, соответствующая x’

- l'= $F^T x$ – эпиполярная линия, соответствующая x
- $Fe = o, Fe' = 0$
- F вырождена (ранг 2)
- F имеет 7 степеней свободы

Существенные и фундаментальные матрицы описывают зависимости между калибровкой камер и парами соответствующих точек на изображениях.

Когда мы наблюдаем плоскую сцену (рис. 175), возможен вырожденный случай. Тогда перспективное преобразование задается с помощью гомографии. То есть мы можем работать не с трехмерными точками, а с двухмерными точками на плоскости и пикселами изображений. Если центры проекций камер совпадают или мы наблюдаем плоскую сцену с двух камер, тогда соответствие между пикселами задается с помощью гомографии. Фундаментальная матрица будет по-прежнему верной, но существует множество матриц гомографии, которые смогут описывать заданную конфигурацию, то есть будет некоторая неоднозначность в нахождении фундаментальной матрицы и как следствие неоднозначность в нахождении параметров сдвига и ориентаций камер.

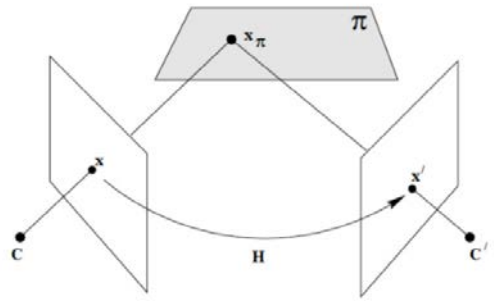


рис.175

Подзадачи структуры из движения

- Калибровка камеры (PnP -проблема)

Дан набор 2D-3D соответствий. Определить параметры камеры

- Оценка движения камеры

Дан набор соответствующих точек в 2х и более изображениях. Определить матрицы калибровки камер для этих изображений (видов)

- Геометрия сцены (структура)

Пусть даны соответствующие точки (проекции) в 2D на 2х и более изображениях. Определить 3D координаты точки

Калибровка камеры

Perspective-n-point problem (PnP) – даны n точек с известными 3D координатами X_i и известными проекциями x_i, оценить параметры камеры.

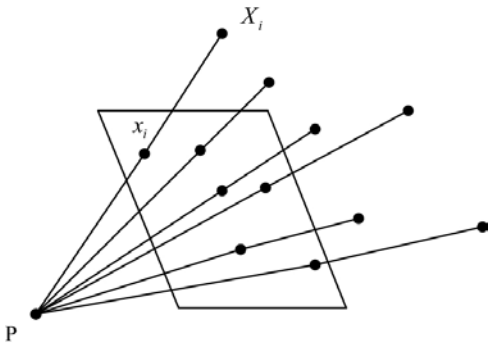


рис.176

Простейший вариант решения задачи – линейный DLT – метод.

Direct Linear Transformation – прямое линейное преобразование. Для каждого известного 2D-3D соответствия можем расписать уравнение перспективной проекции и получить два линейно независимых уравнения на параметры матрицы проекции:

$$\lambda x_i = PX_i$$

$$\lambda \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix} X_i$$

$$x_i \times PX_i = 0$$

$$\begin{bmatrix} 0 & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0 & -x_i X_i^T \\ -y_i X_i^T & x_i X_i^T & 0 \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0$$

Поскольку у нас матрицы 3х4, то проекции 6 точек нам достаточно для того, чтобы составить систему линейных уравнений, которые позволят нам определить все параметры камеры. Максимальная задача p6p - определить максимальные параметры камеры по 6 известным соответствиям 2D и 3D точек. Если есть 6 точек, можем составить систему уравнений, которая просто так не решается, поскольку все измерения с ошибками. Мы можем решить ее в виде задачи однородных наименьших квадратов.

После нахождения матрицы проекции необходимо извлечь из нее внутренние и внешние параметры калибровки. Представим матрицу проецирования в следующем виде:

$$P = KR^T[I \ |-T] = [M \ |-MT]$$

Для разложения M в KR^T воспользуемся RQ-факторизацией (Q -ортогональная матрица, K- верхнетруегольная). Потом найдем T.

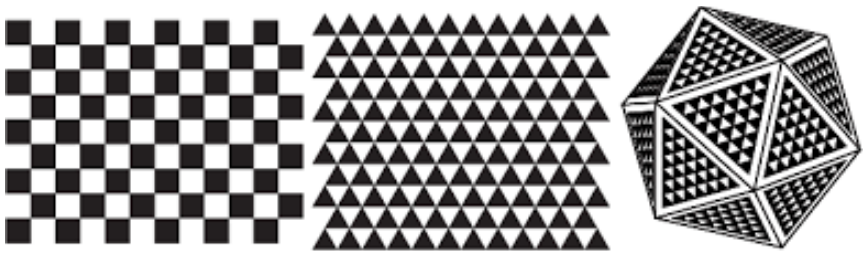
Это классический метод, у которого есть один недостаток: когда мы составляем систему линейных уравнений и решаем ее, мы оптимизируем не целевую ошибку, а некую алгебраическую, которая на самом деле не имеет физической интерпретации.

В области трехмерной реконструкции есть группа методов, которая называется Gold standardили методы золотого стандарта. Это методы нахождения параметров геометрического преобразования, которые оптимизируют ошибку, имеющую геометрический смысл. В нашем случае это ошибка репроекции. Мы хотим найти такую матрицу P, которая минимизирует ошибку репроекции. Из-за нелинейности нам нужно применить метод градиентного спуска. Просто так мы его применить не можем, так как попадем в плохое начальное приближение, поэтому стандартная схема калибровки камеры такова:

- DLT- метод для получения матрицы проецирования P, при этом оптимизируя некоторую алгебраическую ошибку
- RQ-факторизация для извлечения матрицы внутренней калибровки K и внешней калибровки R и T
- Уточнение параметров калибровки с помощью нелинейной оптимизации ошибки репроекции методом градиентго спуска

  $$min_p \sum_i d^2(x_i, PX_i)$$

Калибровочные шаблоны для решения задачи калибровки камеры могут быть крайне различными. Долгое время чаще всего использовался шаблон в виде шахматной доски (рис. 177).



(a) Checkerboard (b) Deltille Grid (c) Icosahedron

рис.177

Мы можем найти уголки клеточек с пиксельной точностью. Поскольку мы знаем размер шаблона, мы можем посчитать для каждой точки изображения, к какому уголку шаблона она соответствует. Для этого нужно, чтобы шаблон был несимметричным, как, например, на рис.177 b. Такие калибровочные шаблоны применялись много лет, и в 2017 году в Facebook, когда они калибровали камеры для систем дополненной реальности, они придумали новый калибровочный шаблон в виде набора треугольников (рис. 177 c).

Оказалось, что для такого шаблона можно придумать метод, который позволяет находить вершины, то есть трехмерные точки с более высокой точностью (попиксельно), чем для прямоугольной сетки, что позволило в несколько раз повысить качество калибровки.

Оценка движения камеры

Дан набор соответствующих точек в 2x и более изображениях, определить матрицы калибровки камер для этих изображений (видов).

Мы можем решить задачу движения через существенную матрицу или через фундаментальную матрицу. Если известна фундаментальная матрица ($x^T F\, x' = 0\,, F = K^{-T} E K'^{-1}$) и матрицы внутренней калибровки, тогда можно вычислить существенную матрицу ($x^T E\, x' = 0, \quad E = [t_*]R$) E = $K^T F K'$ и из нее извлечь внешнюю калибровку камеры.

Эпиполярное ограничение с помощью фундаментальной матрицы задаётся в виде билинейной формы. Домножаем фундаментальную матрицу 3x3 на вектора, соответствующие парам соответствующих точек, заданных в однородных координатах. Одна пара соответствующих точек задает уравнение на параметры фундаментальной матрицы. Если известно множество пар соответствующих точек, то мы можем свести к задаче на однородные наименьшие квадраты.

Поскольку у фундаментальной матрицы неизвестных параметров меньше, чем 9, то мы можем один из параметров зафиксировать, сказав, что норма матрицы равна 1. И нам будет достаточно 8 пар соответствующих точек. Поэтому первый метод, который был предложен для оценки фундаментальной матрицы – 8-и точечный алгоритм (рис. 178).

$$\mathbf{x}^T F \mathbf{x}' = 0$$

$$uu'f_{11} + uv'f_{12} + uf_{13} + vu'f_{21} + vv'f_{22} + vf_{23} + u'f_{31} + v'f_{32} + f_{33} = 0$$

$$\mathrm{A}\boldsymbol{f} = \begin{bmatrix} u_1u_1' & u_1v_1' & u_1 & v_1u_1' & v_1v_1' & v_1 & u_1' & v_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_v' & u_nv_n' & u_n & v_nu_n' & v_nv_n' & v_n & u_n' & v_n' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

Solve f from **Af = 0** using SVD

Matlab:

```
[U, S, V] = svd(A);
f = V(:, end);
F = reshape(f, [3 3])';
```

Resolve det(F) = 0 constraint using SVD

Matlab:

```
[U, S, V] = svd(F);
S(3,3) = 0;
F = U*S*V';
```

рис.178

8 пар соответствующих точек, составляем систему уравнений и решаем задачу. Найденная оценка фундаментальной матрицы не будет являться фундаментальной матрицей, потому что найденная матрица почти всегда не вырождена. У фундаментальной матрицы ранг должен быть равен 2, она должна быть вырожденной, а у найденной матрицы получается ранг = 3. Это происходит из-за того, что все параметры мы измеряем с ошибками из-за влияния шума. Поэтому нам нужно привести матрицу к рангу 2. Для этого нужно найти фундаментальную вырожденную матрицу, которая будет ближе всего к оцененной матрице по норме Фробениуса:

$$\|F - F'\|, \qquad rang(F') = 2$$

Оказывается, найти такую матрицу можно с помощью SVD разложения. Любую матрицу мы можем представить в виде UDV^T. $F = UDV^T$ или $F = Udiag(r,s,t)V^T, r \geq s \geq t$.

U, V – ортогональные матрицы, D -диагональная матрица, у которой на диагонали стоят сингулярные числа, упорядоченные по убыванию. Если мы возьмем матрицу,

записанную следующим образом: $F = Udiag(r, s, 0)V^T$, то есть заменим наименьшее сингулярное число на 0, то полученная матрица F будет удовлетворять нашим условиям (вырожденная, минимизирует расстояние).

Такой метод оптимизирует некую алгебраическую ошибку, не имеющую смысла. Для фундаментальной матрицы можно сформулировать оптимальную ошибку - ошибку расстояния. Оптимальная ошибка – это сумма расстояний от измеренных пар соответствующих точек x и x’ до такой пары точек $\hat{x}$, $\widehat{x'}$, которые удовлетворяют фундаментальной матрице и ближе всего располагаются к исходным точкам x, x’.

$$\sum_{i=1}^{N}\left[d^2(x_i', \widehat{x'}_i) + d^2(x_i, \hat{x}_i)\right],\ \widehat{x'}_i{}^T F \hat{x}_i = 0$$

Найти такие точки $\hat{x}$,которые минимизируют расстояние и удовлетворяют фундаментальной матрице тоже не так просто, требует градиентного спуска, нелинейной оптимизации, поэтому мы можем уточнит фундаментальную матрицу следующим способом: получаем начальное приближение фундаментальной матрицы, для каждой пары соответствующих точек на каждом шаге находим ближайшие точки, удовлетворяющие фундаментальной матрице, считаем для нее общую ошибку, применяем шаг градиентного спуска для уточнения фундаментальной матрицы и повторяем все заново: считаем ошибку с помощью градиентного спуска, уточняем фундаментальную матрицу с помощью шага градиентного спуска и так до сходимости.

Если в калибровочном шаблоне можно было положиться на то, что в исходные данные надежные и соответствия точек верно, то, когда мы работаем с парами соответствующих точек, у нас нет никакой уверенности, поэтому нужно использовать робастные методы, например, метод RANSAC. То есть у нас имеется множество пар соответствующих точек, из них будут случайным образом выбираться по 8 точек, оценивая фундаментальные матрицы. Выберем наилучшую фундаментальную матрицу, по ней определим, какие пары точек удовлетворяют, и по ним уточним фундаментальную матрицу.

Геометрия сцены

Даны проекции x_1, x_2 точки X=(X,Y,Z) на 2 или более изображений (с известными матрицами калибровки), найти координаты точки X (рис. 179).

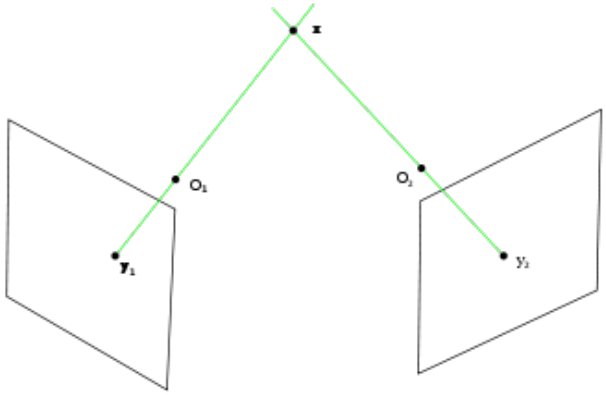


рис.179

Оценка структуры сцены еще известна как задача *триангуляции*.

Можем решить эту задачу с помощью линейного подхода. Проекция точки на каждую камеру дает 2 уравнения на параметры трехмерной точки X. Соответственно 3 неизвестных параметра, проекция трехмерной точки на 2 изображения дает 4 уравнения на параметры, которые мы можем решить и найти координаты (рис. 180).

Linear triangulation

$$x = MX \quad x' = M'X$$
$$x \times MX = 0$$
$$x' \times M'X = 0$$

$$\longrightarrow \quad AX = 0$$

Linear combination of 2 other equations

$$x(m_3^T X) - (m_1^T X) = 0$$
$$y(m_3^T X) - (m_2^T X) = 0$$
$$x(m_2^T X) - y(m_1^T X) = 0$$

$$A = \begin{bmatrix} xm_3^T - m_1^T \\ ym_3^T - m_2^T \\ x'm_3'^T - m_1'^T \\ y'm_3'^T - m_2'^T \end{bmatrix}$$

homogeneous $\quad \|X\| = 1$

$AX = 0$ Homogenous system:

X is last column of V in the SVD of A= $U\Sigma V^T$

рис.180

Опять же это будет алгебраическая ошибка, мы можем запустить градиентный спуск и уточнить трехмерные координаты, минимизируя ошибку репроекции.

$$d^2(x_1, P_1X) + d^2(x_3, P_2X)$$

Используем результаты линейного метода как начальное приближение и затем уточняем с помощью градиентного спуска.

С учетом всех рассмотренных подзадач мы можем решить общую задачу структуры и движения. У нас есть целевая ошибка -ошибка репроекции. Рассмотрим случай, когда известны внутренние калибровки всех камер. Тогда мы можем вычислять существенную матрицу между двумя видами по соответствиям, то есть оценивать напрямую внешнюю калибровку камеры относительно первой камеры.

Классический подход к решению задачи структуры и движения – это *последовательный подход:*

- Инициализируем движение через существенную матрицу двух камер (находим ориентацию и положение второй камеры относительно первой)
- Применяем триангуляцию и для всех пар точек находим соответствующие им точки, то есть инициализируем структуру
- Последовательно берем новое изображение, и для каждого нового изображения оцениваем матрицу проекции новой камеры по известным 3D точкам, видимым на этом изображении, применяем калибровку, определяем параметры новой камеры. Теперь появились точки, для которых мы можем найти новые трехмерные координаты. Соответственно мы уточняем и дополняем структуру, вычисляем новые 3D точки, уточняем существенные точки, видимые на камере, применяем триангуляцию и находим новую точку.
- Последовательно проходим по всем видам, уточняем структуру и в движение методом связок (методом градиентного спуска)

Применение метода градиентного спуска для уточнения параметров как трехмерных точек, так и матриц проекции в трехмерной реконструкции получил свое собственное название - *метод связок* (Bundle adjustment), потому что мы берем все вместе и уточняем все вместе.

Реализация метода связок -достаточно сложная задача, поскольку параметров может быть очень много параметров. Решить эту задачу градиентным спуском напрямую вычислительно затруднительно, но мы можем воспользоваться тем, что не все параметры зависят от других параметров, поэтому задачу метода связок можно факторизовать в множество простых задач.

Также важно упомянуть неоднозначность решения задачи трёхмерной реконструкции. Если мы умножим всю сцену на некоторый коэффициент k и в то же время умножим матрицы камер на 1/k, проекции точек сцены на изображения не изменятся:

$$x = PX = \left(\frac{1}{k}P\right)(kX)$$

На практике это означает, что при отсутствии какой-то дополнительной информации трехмерную реконструкцию можно решить с точностью до масштаба. Когда у нас есть множество фотографий, мы не знаем, был ли сфотографирован реальный город или его маленький макет, трёхмерную модель мы построим, но масштаб будет неизвестен. Если мы хоти далее решать какие-то метрические задачи с этой моделью, необходим эталон, например, расстояние между какими-то двумя точками. Сейчас есть отдельная задача определения масштаба для задачи структуры и движения.

Разреженная трехмерная реконструкция на практике

Теперь у нас есть теоретические основы, чтобы рассмотреть, как на практике реализуется метод разреженной трехмерной реконструкции по коллекции изображений из интернета, что было впервые представлено 11 лет назад в работе «Рим за день» (рис. 181).

1. Скачиваем несколько миллионов изображений по тегу Rome. Фотографии получены разными пользователями в разное время года, суток с разных устройств. Порядок совершенно случайный, поэтому придется воспользоваться сложным алгоритмом матчинга

Почти все фотографии сейчас хранятся в формате jpeg, в котором вместе с фотографиями передается служебная информация.

рис.181

В эту служебную информацию входит модель камеры и фокусное расстояние камеры, на которое было сделано данная фотография (рис. 182). Для того, чтобы оценить внутреннюю калибровку, необходимо оценить фокусное расстояние, измеренное в числе пикселов, то есть фокусное расстояние в мм, деленнное на размер пикселей в мм, и положение принципиальной точки. Мы можем по умолчанию считать, что принципиальная точка располагается ровно в центре изображения, хотя на практике это не всегда так.

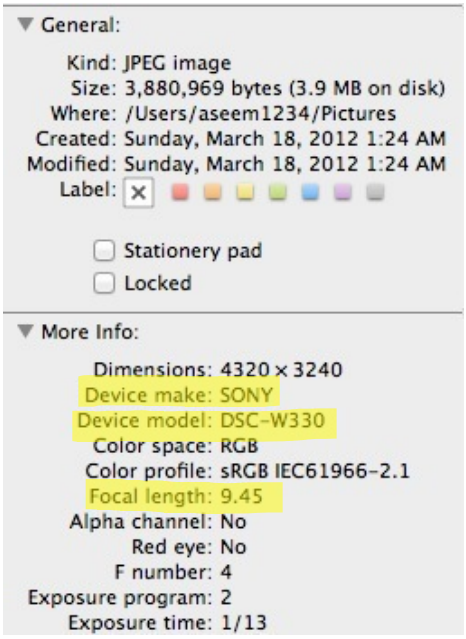


рис.182

Из информации в EXIF (рис. 182) посчитать фокусное расстояние в пикселах мы не можем, потому что физический размер пиксела нам неизвестен. Однако для каждой модели камеры по ее параметрам известен размер матрицы в мм, зная который и разрешение, мы можем посчитать размер пиксела в мм и далее посчитать фокусное расстояние в пикселах. Следовательно, внутреннюю калибровку камеры в простом варианте можем считать априори известной. Поскольку мы работаем с большим количеством пользовательских фотографий, неподходящие файлы мы можем не рассматривать.

2. Найдем особые точки и соответствия между ними. Для этого воспользуемся мощным методом, например, SIFT. Найденные точки сопоставляем по дескрипторам SIFT. Фильтруем ложные соответствия по эпиполярной геометрии. Для этого с помощью RANSAC вычисляем фундаментальную матрицу и просеивать точки. Пары, между которыми много соответствий, будем считать связанными.
3. Строим граф связанности изображений. Изображения, которые в него не попали, отбрасываем

Одна проблема: 250K изображений -> 31M пар изображений. Метод SIFT недостаточно быстрый для этого. Чтобы сопоставить все изображения друг с другом, из расчета 2 пары в секунду нам понадобится 1 год и кластер из 500 машин, что вычислительно нереально.

4. Вспомним про метод «Мешок слов». Мы можем запустить метод поиска похожих изображений, который найдет набор гипотез, например, по 40 штук, и мы будем находить пары особых точек только между этими 40 гипотезами. Таким образом резко уменьшим объем пар изображений для сопоставления и сделаем задачу вычислительно пригодной

Получим граф связности изображений (рис. 183)

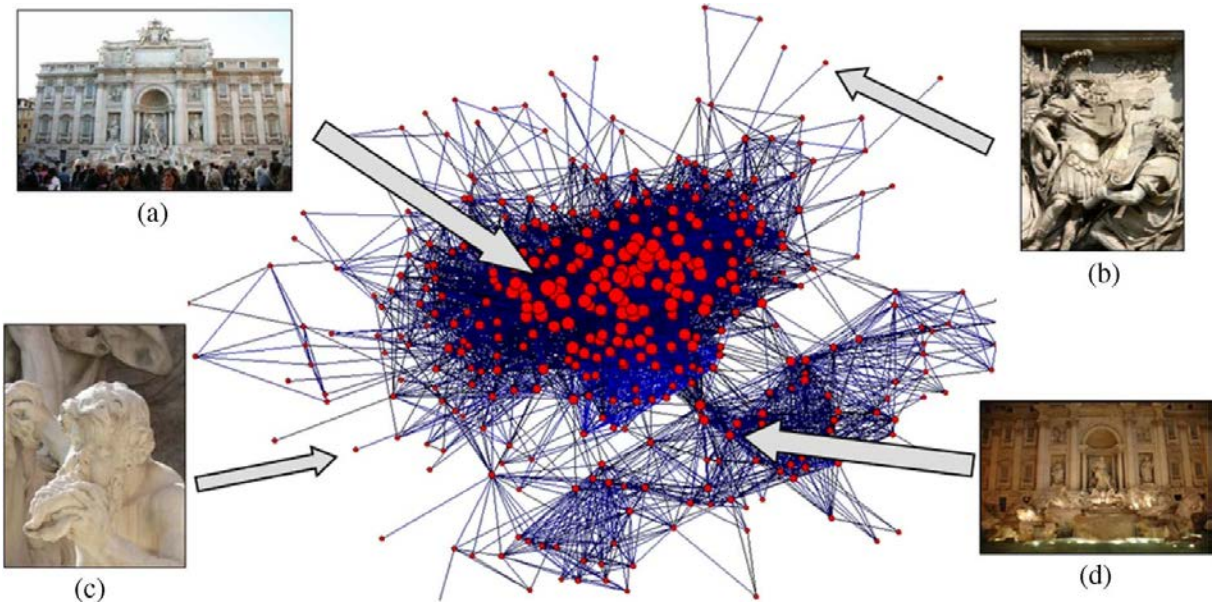


рис.183

Дополнительно улучшим его. Мы пока рассматривали только пары изображений, однако поскольку у нас много изображений, которые видят одну часть сцены, могут быть ситуации сопоставления точек между тремя и более изображениями. В правильном случае должны образовываться циклы: точка a соответствует точке b, точка b соответствует точке c, точка c ссоответствует точке a. Если такой цикл есть, значит все хорошо, но может произойти размыкание цикла: точка a соответствует точке b, точка b соответствует точке c, точка c соответствует точке d, a d и a – это 2 разные точки в изображении. Значит, где-то здесь произошла ошибка. Где именно мы не знаем, но мы не доверяем всем эти соответствиям и выбрасываем весь цикл. Объединяем соответствия в «следы», отбрасываем незамкнутые следы как ошибки. При таком прореживании отбрасываются случайные сопоставления.

Поскольку пользовательские фотографии обычно снимаются неравномерно (с некоторых ракурсов есть очень много фотографий, а с других их почти нет), то использовать множество фотографий с одного и того же ракурса смысла нет. Поэтому для ускорения упрощают конфигурацию, то есть выбирают набор опорных кадров (рис. 184).

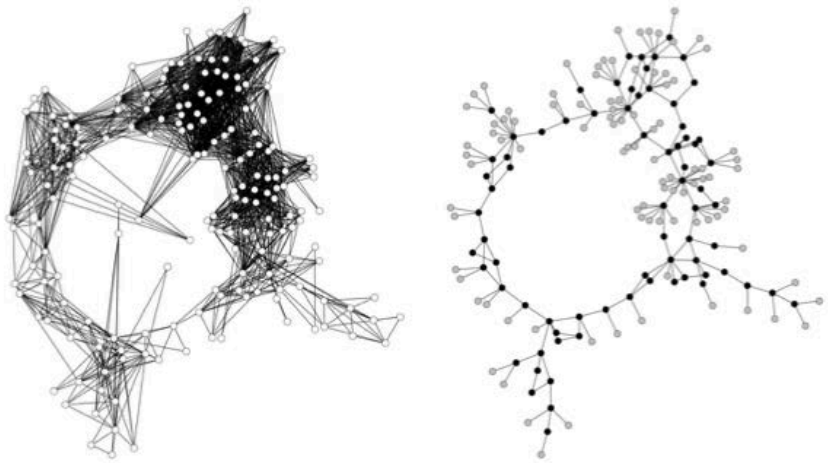


рис.184

То есть мы выбираем кадры, которые связаны друг с другом, чтобы они равномерно были распределены по пространству. Для каждого такого кадра находим ему пары, по которым будем делать базовые трехмерные реконструкции.

Оказалось, что последовательный подход реконструкции хуже иерархического. Вместо того, чтобы дополнять частичную реконструкцию, лучше разбить все множество кадров на пары, каждой парой сделать трехмерную реконструкцию и начать последовательно склеивать пары друг с другом. То есть объединить пару в четверку, уточнить, затем объединить четверки и так далее, пока не объединим все в единую реконструкцию. Такой скелет (рис. 184) существенно упрощает эту задачу.

5. Выбираем опорные пары изображений. Вычисляем структуры по паре. Для этого строим матрицы внутренней калибровки, вычисляем фундаментальные и существенные матрицы, триангулируем точки и уточняем получившуюся реконструкцию методом связок. Далее последовательно добавляем новые изображения, калибрируем камеры по уже известным 2D/3D соответствиям, триангулируем новые точки и снова уточняем методом связок.

Пары опорных кадров нужно выбирать с умом. Нужно выбирать кадры с разных ракурсов с большим количеством соответствующих точек. Для того, чтобы подобрать опорные пары. Воспользуемся гомографией. Гомография плохо описывает сцену, когда сцена неплоская, и есть существенный параллакс, то есть большая стереобаза. Соответственно, если движение в сцене маленькое или сцена плоская, тогда гомография хорошо описывает сцену. Мы можем вычислить для каждой пары кадров и гомографию, и фундаментальную матрицу и оценивать степень трехмерности сцены по отношению числа точек, которые удовлетворяют гомографии, к числу точек, которые

удовлетворяют трехмерной матрице. Чем меньшую долю точек мы можем приблизить плоскостью, тем сцена более трехмерная.

В итоге получается алгоритм, который позволяет определить положение камеры на сцене и разреженную трехмерную реконструкцию для больших сцен. Например, для трехмерной реконструкции города Дубровник было скачено 57845 изображения, процесс занял 18 часов, в итоговой модели появилось 2 М трехмерных точек, которые наблюдались с 11.3 М ракурсов, то есть в среднем каждая точка наблюдалась 5.5 раз. Некоторые примеры реконструкций (рис. 185)

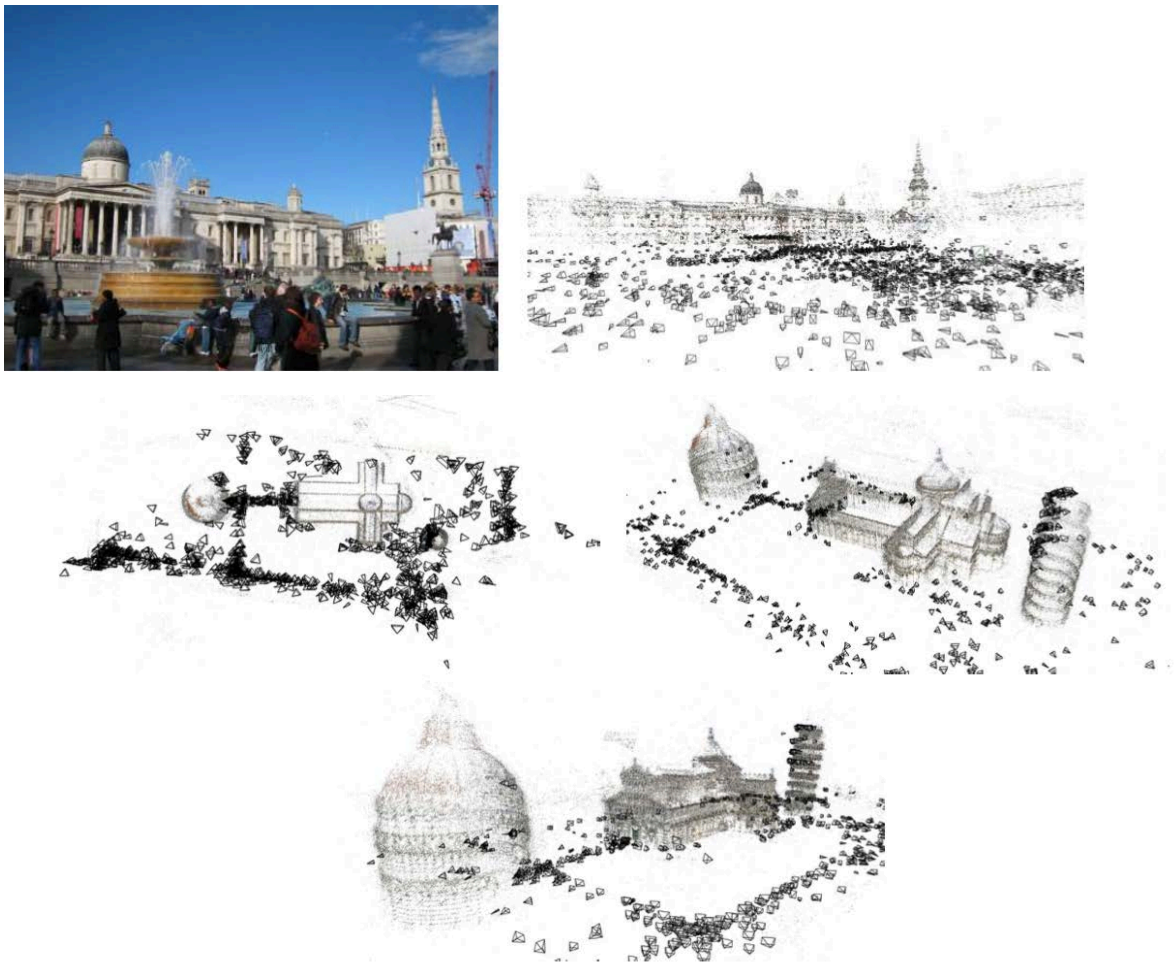


рис.185

Понятно, что для пользователя такой уровень реконструкции не очень интересен, но важно, что получилось позиционировать камеру конкретно в месте сцены, где именно относительно локации эта фотография была получена.

Исследователи, работавшие над этим проектом, позднее объединись в кооперацию BigSFM. BigSFM – это большой проект, нацеленный на реконструирование всего мира по интернет фотографиям. Несколько результатов этой работы, в частности библиотекуBundler, которая позволяет сделать разреженную реконструкцию по фотографиям, можно скачать и воспользоваться.

Производной этой работы стала *система глобального позиционирования*, которая определять из какой конкретно точки мира была сделана загруженная в систему фотография, под каким ракурсом.

Другим применением трехмерной реконструкции является *хронология сцены,* где к трехмерной реконструкции добавили временное изменение. Мы не просто можем спозиционировать фотографию, но и отобразить ее на реальную сцену и смотреть как со временем меняются те или иные объекты сцены. Например, по пользовательским фотография можно сделать хронологию рекламы в известных туристических местах. Одну из работ по такой трехмерной реконструкции сделал Microsoft. На базе этой системы он сделала прототип некоторой коммерческой системы под названием Фотосинт. Этот продукт реализует трехмерную регистрацию фотографий, по которым пользователь может двигаться. Это приложение оказалось популярным, и его показывали в разных детективах.

Резюме

- Разреженная трехмерная реконструкция – один из наиболее проработанных разделов компьютерного зрения
- Мы научились: калибровать камеру, определять взаимное расположение камер, триангулировать точки, определять положение камеры относительно объекта по 2D-3D соответствиям, сопоставлять изображения на больших коллекциях
- Дальнейшей развитие: использование нейросетевых методов для решения отдельных подзадач и задачи в целом

Лекция 11. Плотная трехмерная реконструкция

Бинокулярное стерео

Вся трехмерная реконструкция базируется на явлении *параллакса.*

Параллакс- видимое смещение объекта в зависимости от точки обзора

Если мы наблюдаем объекты в сцене, которые располагаются на разном от наблюдателя расстоянии, то смещение объекта меняется в зависимости от удаленности объекта: чем ближе объект, тем больше смещение. На основе параллакса работает психофизиологическое явление *стереопсис*.

Стереопсис – сенсорный процесс, возникающий при бинокулярном зрении как психофизическая реакция на сетчатую горизонтальную диспаратность. В результате субъект переживает специфическое ощущение глубины, то есть расстояния до объектов.

Задача плотного стерео или плотной многовидовой реконструкции – восстановить в 3D все видимые на изображении точки. Если 2 изображения, то это бинокулярное стерео (рис. 186), если более 2 изображений, то многовидовое стерео. Во всех случаях калибровка камер считается известной. В случае бинокулярного стерео камеры заранее откалиброваны и фиксированы. В случае многовидового стерео чаще всего положение камер изначально неизвестны, их нужно оценить, поэтому сначала решается задача разреженного стерео для уточнения расположения камер и затем построить трехмерную модель.

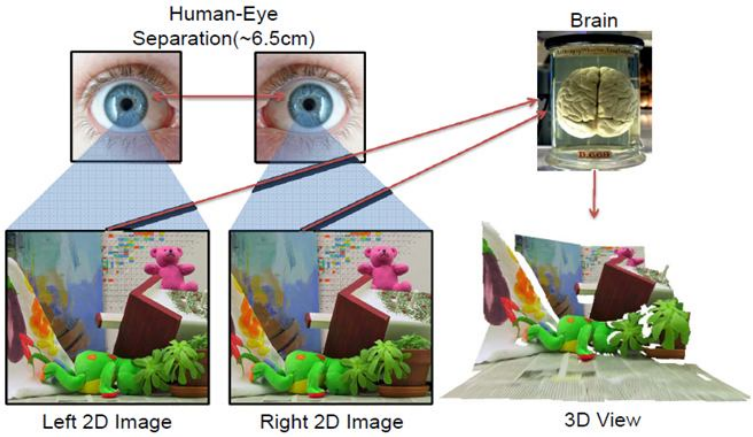


рис.186

Общая схема бинокулярного зрения следующая (рис. 187):

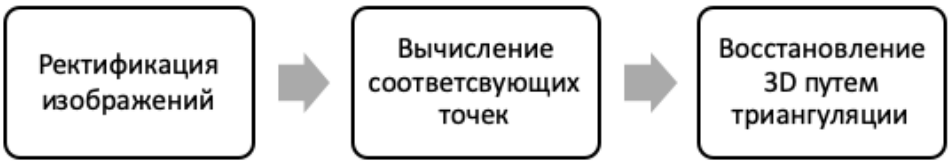


рис.187

Ректификация – преобразование стереопары в изображения, в которых соответствующие эпиполряные линии лежат на одно и той же горизонтальной строке.

Эпиполярное ограничение позволяет упростить задачу стереосопоставления, так как соответствующие точки должные лежать в соответствующих эпиполярных линиях. Вычислительно эту задачу было бы удобнее решать, если бы эпиполярные линии совпадали со строками изображения. Поэтому мы сначала упрощаем задачу, применяя ректификацию. Основной способ ректификации – проецирование на общую плоскость. То есть у нас имеется стереопара, мы берем плоскость параллельную базовой линии и проецируем на нее оба изображения. Поскольку мы проецируем изображения, то проекция каждого изображения может быть решена с помощью гомографии. Для этого нужно вычислить гомографию, которая переводит изображение из исходного в общую плоскость. На рис. 188 изображен пример ректификации через гомографию.

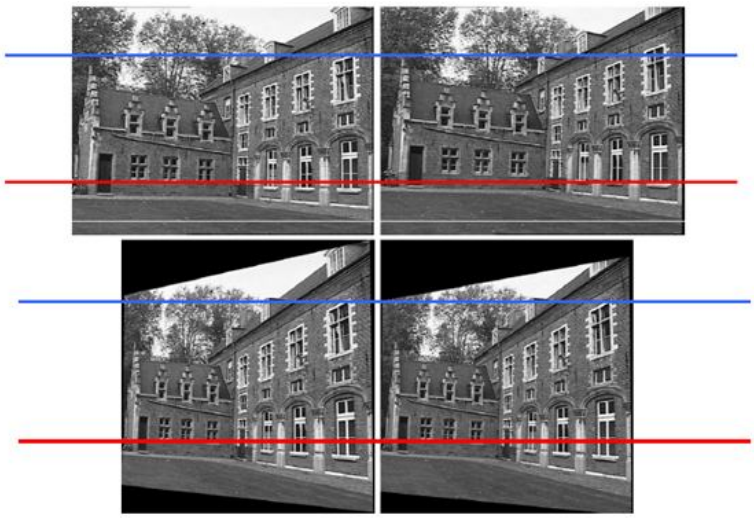


рис.188

Наверху 2 изображения. Если мы посмотрим в строки изображения, то увидим, что точки, которые лежат на одной строке для одного изображения, лежат на других строках для другого изображения. После ректификации с помощью гомографии положения преобразовались: они немного исказились и теперь при проецировании область, занимаемая изображениями, стала трапецевидной. При этом одни и те же

точки располагаются на одних и тех же строках. Например, верхний левый угол окна здания лежит на одной и той же строке в обоих изображениях. Этот метод хороший, реализован во многих библиотеках, но у него есть ограничение: он неприменим, если камера движется вперед или назад. В этих случаях эпиполярные точки попадают в центр изображения, базовая линия идет практически перпендикулярно изображениям, а проецировать изображение поперек мы не можем, поэтому нужно использовать другой способ ректификации.

Другой способ ректификации – *радиальная развертка* (polar rectification).

Идея заключается в следующем: индексируем эпиполярные линии углом поворота относительно эпиполи, копируем соответствующие пары эпиполярных линий последовательно в соответствующие горизонтали ректифицированных изображений.

То есть мы как бы разворачиваем изображение, берем эпиполярную линию, исходящую из эпиполяра, копируем ее в новое изображение, берем следующую строку и так постепенно разворачиваем все изображение.

Такая полярная ректификация в отличие от проецирования с помощью гомографии применима всегда. Однако в случае полярной ректификации многие прямые линии превращаются в кривые, то есть изображение искажается сильнее, чем в случае гомографии, которая сохраняет линейность. Полярная ректификация может сильно деформировать изображение, что сильно снижает его качество и последующее стереосопоставление, но мы сводим к одной и той же упрощенной задачи плотного стерео, в которой соответствующие пикселы лежат на соответствующих строках. Таким образом, можно считать, что первый этап ректификации технический и для него существуют готовые методы, которые реализованы в библиотеке.

Третий этап – восстановление 3D путем триангуляции тоже технический. Каждую пару соответствующих точек можем обработать независимо. В случае ректифицированной стереопары сделать оценку глубины точек еще проще. Для этого возьмем срез проекции одной точки на изображение (рис. 189), получаются 2 подобных треугольника, которые даю пропорции.

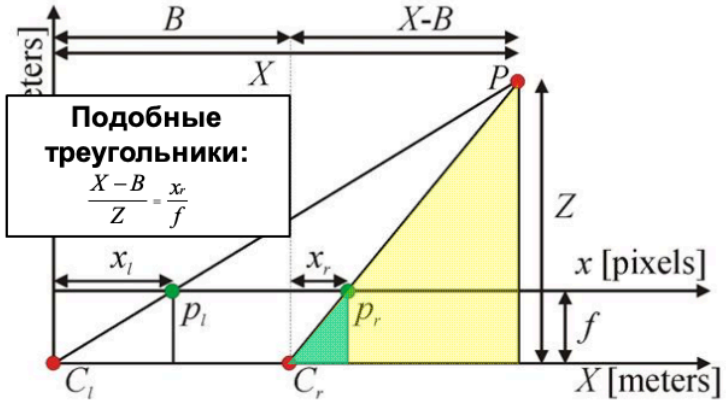


рис.189

$\frac{X}{Z} = \frac{x_l}{f}$ – отношение X координаты к удаленности точки равняется отношению проекции точки на картинную плоскость к фокусному расстоянию. На втором изображении координаты той же самой точки в системе координат второй камеры теперь $\frac{X-B}{Z} = \frac{x_r}{f}$

Из этих двух соотношений мы можем выразить глубину, исключив координаты точки x.

Из подобия треугольников:

$$\frac{X}{Z} = \frac{x_l}{f}, \frac{X-B}{Z} = \frac{x_r}{f}$$

Исключаем X и выражаем Z:

$Z = \frac{Bf}{x_l - x_r} = \frac{Bf}{d}$, $d = x_l - x_r$ – диспаритет, который задает степень параллакса

Итоговые координаты 3D точки:

$X = \frac{Zx}{f}, Y = \frac{Zv}{f}$, Z=$\frac{Bf}{d}$

Для того, чтобы решить задачу плотного стерео, для каждой точки остается найти диспаритет и для каждой точки установить соответствие на втором изображении.

Вычисление карты диспаритета – самый сложный элемент в задаче плотного стерео.

Мы свели задачу плотного стерео, то ест построения карты глубины к задаче стереосопоставления и поиска диспаритета для каждого пиксела изображения. После ректификации соответствующие точки лежат на соответствующих эпиполярных линиях и для каждой точки надо найти соответствующую точку на втором изображении. У этой задачи есть некоторые сложности:

1. Отсутствие текстуры – многие точки похожи друг на друга

Если мы рассматриваем однородную область, в которой цвета и окрестности всех точек не отличаются друг от друга, то мы не можем принять уверенное решение, какая точка какой соответсвует.

2. Текстура неизменна в горизонтальном направлении

Все точки, лежащие на одной строке, не отличаются друг от друга.

3. Повторяющаяся текстура

Визуально похожих точек на втором изображении столько же, сколько повторяющихся элементов текстуры.

Дополнительно усложняет задачу факт, что цвет точки и ее окрестность могут претерпеть изменения между ракурсами. В первую очередь влияет шум камеры, который независимо распределён. Из-за него яркости и цвета пикселов будут

отличаться друг от друга с погрешностью в шум. Также все могут испортить освещение и погрешности сэмплирования.

Поскольку границы на изображениях с плавным переходом, в зависимости от расположения изображения относительно дискретной матрицы границы (переходы) могут немного отличаться друг от друга, поэтому идеального сопоставления добиться не удастся.

Еще одна большая сложность – перекрытие и полупрозрачные объекты. Некоторые пиксели могут быть не видны (перекрыты) на другом изображении. Невидимые области называются областями перекрытия, они при бинокулярном стерео обычно не очень большие по размеру и располагаются рядом с границами объектов, расположенных на переднем плане: чем ближе располагается объект к наблюдателю, чем больше у него диспаритет, тем сильнее он смещается и тем больше область открытия, которая видна только на одном из двух изображений (рис. 190). Из-за областей открытия в некоторых точках решить задачу стереосопоставления невозможно. Приходится указывать, что в некоторых точках значение глубины неизвестно, либо потом как-то обрабатывать получившеюся карту глубины, чтобы потом находить области открытия.

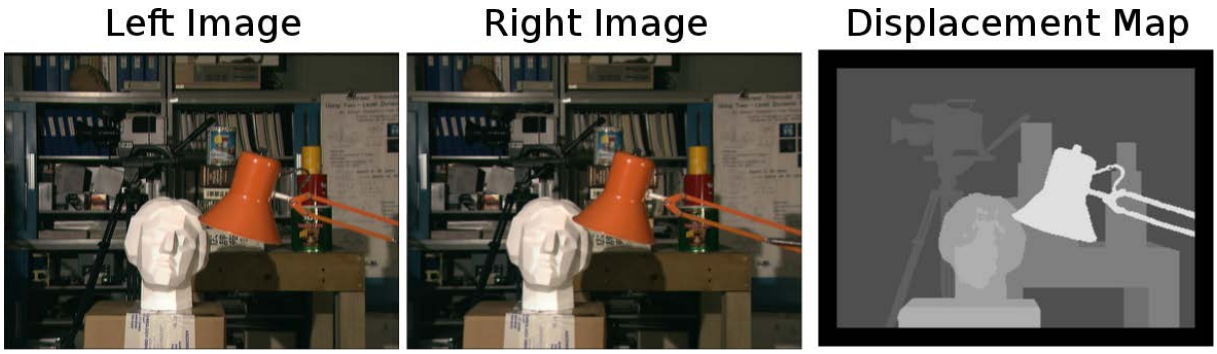


рис.190

Были предложены разные подходы к решению задачи стереосопоставления. Простейший, но тем не менее хороший подход –локальный метод с использованием стратегии WTA (Winner-Take-All), в котором в каждой точке решаем задачу стереосопоставления независимо, то есть решение зависит только от локальной окрестности точек и не зависит от всех ее соседей, побеждает соответствие с наименьшей стоимостью. Самый простой вариант -выбрать на втором изображении точку более близкую по яркости. В итоге на большинстве изображений, особенно в которых есть текстура, как на рис. 191, результат получается не самый плохой, несмотря на то что мы ошибаемся в огромном количестве точек, контуры объектов переднего плана мы видим четко и определяем диспаритет достаточно точно.

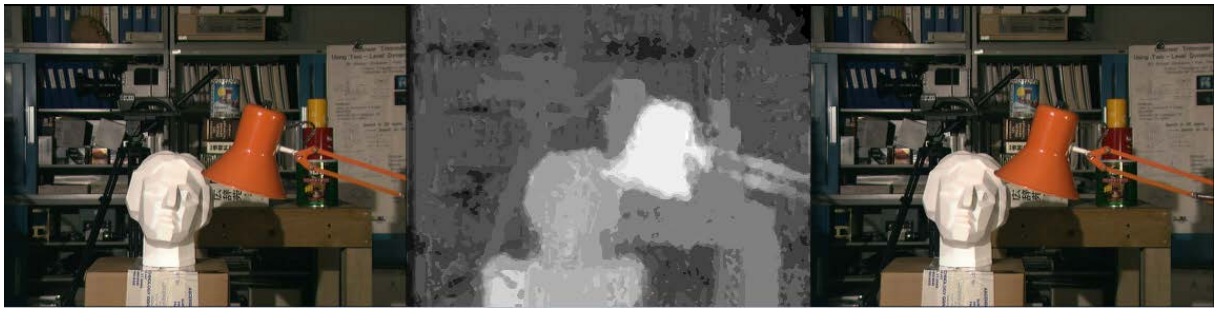


рис.191

Проблема этого наивного алгоритма заключается в слишком высокой неоднозначности. Чтобы уменьшить неоднозначность, нужно учитывать окрестности. Соответственно будем описывать каждую точку некоторой окрестностью и искать на втором изображении пиксел с более похожей окрестностью. Стоимость соответствия – SAD, SSD, NCC по окну вокруг пикселя.

$d_p = \ argmin_{0 \le d \le d_{max}} \sum_{q \in W_p} c(q, q-d)$

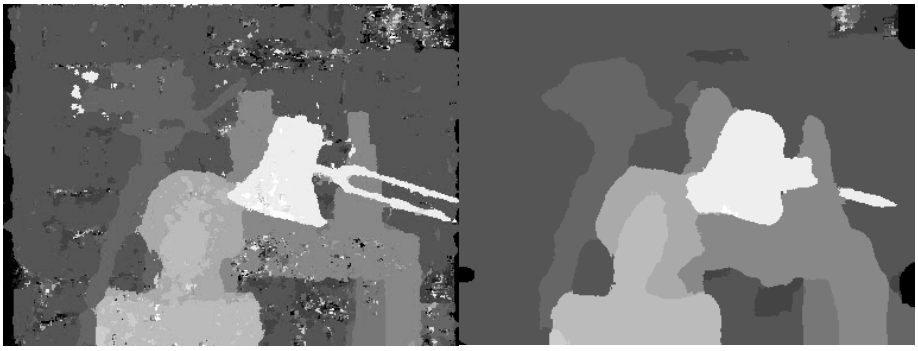
d_p – искомый диспаритет в пикселе p

$c(\ p, q)$ – функция стоимости

W_p – окно вогруг пикселя p

d_{max} – максимально возможный диспаритет

Применение жадного сопоставления с окрестностью качественно повышает результат



Окно 3x3 Окно 21x21

рис.192

При этом возникает большая зависимость от размера окна (рис. 192): чем оно больше, тем картинка получается глаже и наблюдается эффект раздувания объектов переднего плана (foreground fattening). Поскольку рядом с объектами, располагающимися на переднем плане, возникают области открытия, часть фона пропадает, при сопоставлении соседних точек с точками объектов переднего плана неизменной остается часть окрестности объекта переднего плана. Мы считаем, что эта точка, расположенная на самом деле рядом с объектом переднего плана относится к объекту переднего плана, поэтому маски передних объектов раздуваются. На рис. 192 видно, что при использовании окна 3x3 мы сможем находить диспаритеты для достаточно тонких деталей, но во многих случаях у нас ошибки из-за шума. При окне 21x21 деталей нет, ошибок меньше, но объекты переднего плана больше по своим размерам.

Идея жадных алгоритмов может быть хорошо реализована на современном уровне с использованием нейросетей. Вместо того, чтобы использовать какую-то эвристическую метрику для сравнения пикселов изображения по окрестностям, мы можем эту метрику обучить. Возьмем нейросеть, которая на вход будет получать маленькие окрестности, например, области 9x9 пикселов и сравнивать их друг с другом. Задача нейросети – ответить, являются ли точки окрестностей, которые мы подали на вход, соответствующими друг другу.

Мы можем воспользоваться разными архитектурами. Но в первой работе использовали следующую архитектуру (рис. 193):

Буквально один сверточный слов 5x5 пикселов, поскольку у нас изображение очень маленького размера и набор полносвязных слоев. Вначале строим вектор-признаки каждого фрагмента в отдельности, конкатенируем их. Дальше идет набор полносвязных слоев, который можно проинтерпретировать как классификатор меток. Поскольку на вход подается пара соответствующих или несоответствующих пикселов, то из одного изображения мы можем насэмплировать большое количество примеров для обучения, то есть обучающую выборку сделать очень легко.

Вопрос в том, какие пары пикселов считать соответствующими. Часто используют следующую метрику: если диспаритет найден с точностью выше, чем заданный порог, то есть ошибка меньше, чем заданный порог, то пара пикселов сопоставлена верно. Этот порог берется достаточно грубым, например, 5 пикселов. То есть, если мы оценили диспаритет с точностью до 5 пикселов, то в целом мы считаем, что диспаритет отметили верно. У пары истинных соответствий мы можем взять окрестность пикселов и все их считать положительными. Если пара диспаритетов меньше заданного порога, о пара ложна, будем ее использовать для обучения ложных примеров.

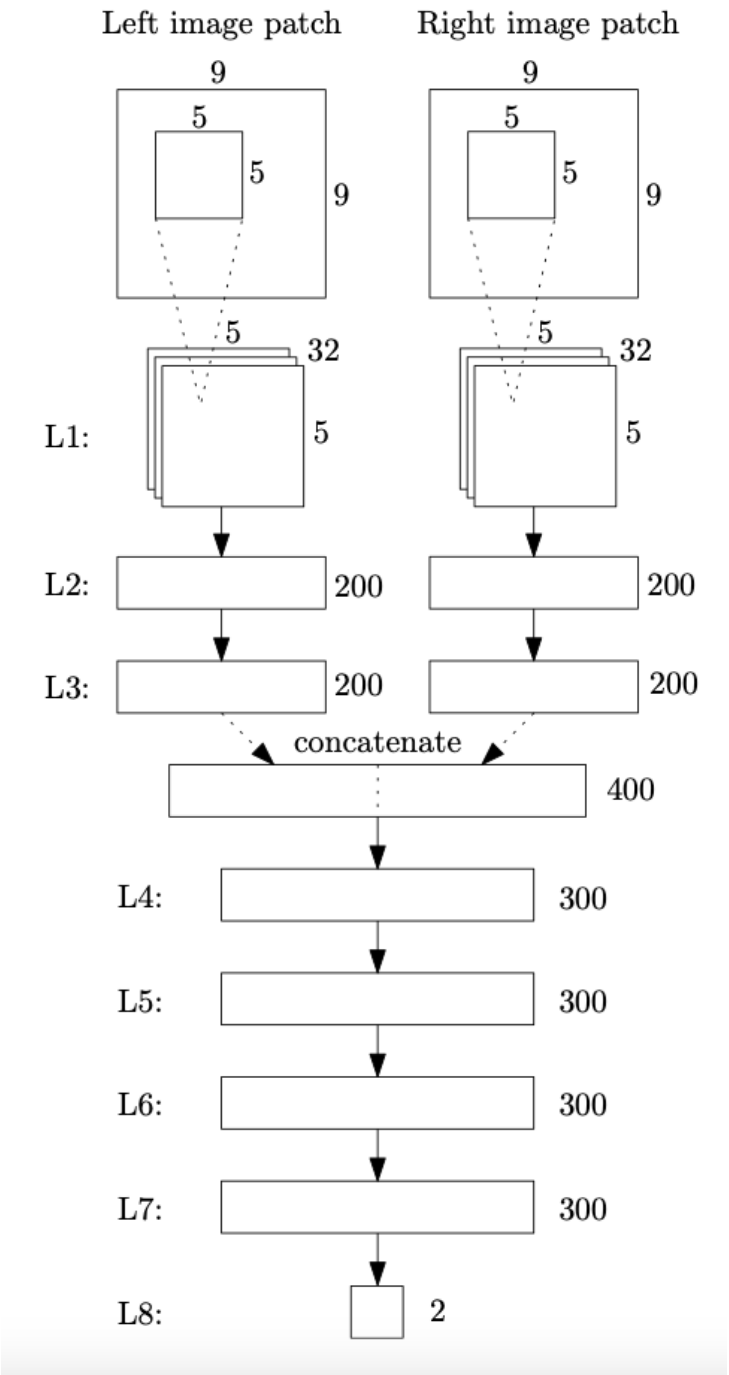


рис.193

Когда эту идею применили для обучения жадных методов нахождения диспаритета, оказалось, что на многих современных датасетах для большей части мы находим диспаритет с достаточно высокой точностью. Мы можем результат жадного алгоритма подать на вход алгоритму, который принимает решение в совокупности по всему

множеству пикселов, некоторые шумы устраняются, и получается весьма точная карта, в которой лишь 2.61% пикселов оценены неверно.

Альтернативными подходами являются глобальные методы, в которых диспаритет будет находиться по группе пикселов или по всей картинке. Если мы хотим решать задачу для всего изображения, то формулируем задачу в терминах разметки графа и минимизации энергии. В глобальных методах мы рассматриваем какую-то структуру зависимости между пикселами, например, считая, что я диспаритет в каждом пикселе должен зависеть от диспаритета во всех соседних пикселах. Соответственно целевой функционал мы формулируем в виде суммы двух компонент. Компоненты, оценивающие каждый пиксел независимо, то есть фактически мы задаем штраф, с которым заданному пикселу даем диспаритет D. Обычно этот штраф задается из жадного алгоритма, мы используем попиксельную метрику для оценки соответствия пикселов (то есть насколько хорошо дать данному пикселу этот диспаритет). Второй компонент задает гладкость и пенализирует за изменение диспаритета между пикселами.

$$E(D) = E_{data}(D) + E_{smooth}(D)$$

E_{data} – соответствие цветов

E_{smooth} - гладкость

При задании штрафа есть несколько вариантов, в зависимости от которых глобальный метод будет по-разному обрабатывать разрывы диспаритетов, то есть границы объектов переднего плана. Зачастую объект переднего плана в изображениях существенно удален от объектов заднего плана, то есть наблюдается резкое изменение диспаритета. В качестве штрафа можно использовать линейную модель, в которой штраф пропорционален изменению диспаритета. В этом случае для минимизации энергии неважно, как именно мы зададим переход от одного значения диспаритета к другому, то есть как будет выглядеть переход от объектов переднего плана к объекту заднего плана. Если мы растянем границу в виде набора единичных переходов, то суммарный штраф для данного примера (рис. 194) будет равен 9. Для второго примера, если один резкий переход, то штраф будет равен значению перехода = 9. На практике это приводит к тому, что если мы используем такую линейную модель, то у нас получаются слаженные переходы.

Если мы посмотрим на границы объектов (рис. 195), то увидим, что у них есть некоторые промежуточные значения. То есть модель пытается сгладить резкие переходы, получается, что объект переднего плана плавно переходит в объекты фона. С одной стороны, это плохо, что у нас нет резкой границы объекта, но с другой стороны, если мы будем рассматривать не плоские объекты, а объекты существенно трехмерные, у которых диспаритет меняется плавно, например, стену, которая идет под углом к наблюдателю, метод с линейным штрафом сможет корректно обработать эту стену.

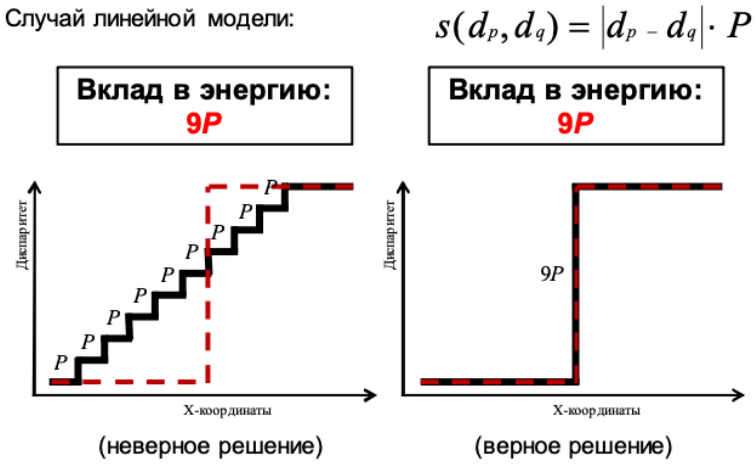
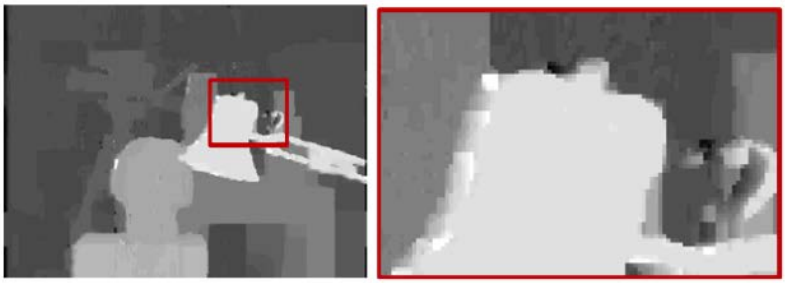


рис.194



Пример результатов для линейной модели. Справа показан увеличенный фрагмент

рис.195

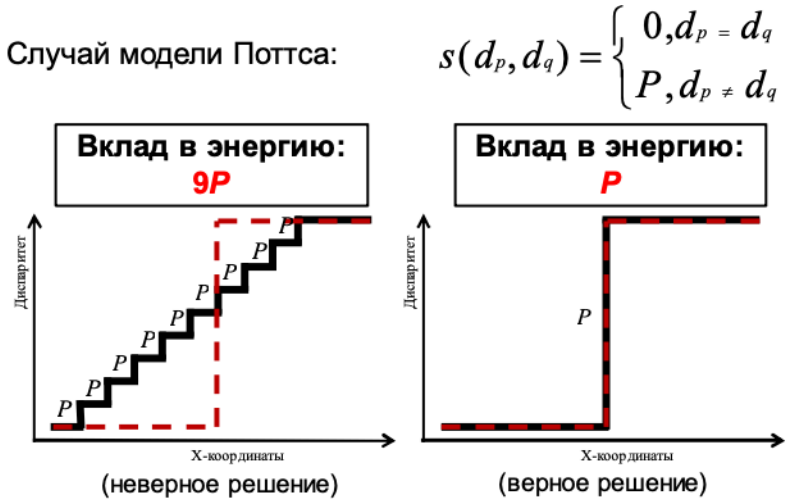
Если мы посмотрим на границы объектов (рис. 195), то увидим, что у них есть некоторые промежуточные значения. То есть модель пытается сгладить резкие переходы, получается, что объект переднего плана плавно переходит в объекты фона. С одной стороны, это плохо, что у нас нет резкой границы объекта, но с другой стороны, если мы будем рассматривать не плоские объекты, а объекты существенно трехмерные, у которых диспаритет меняется плавно, например, стену, которая идет под углом к наблюдателю, метод с линейным штрафом сможет корректно обработать эту стену.

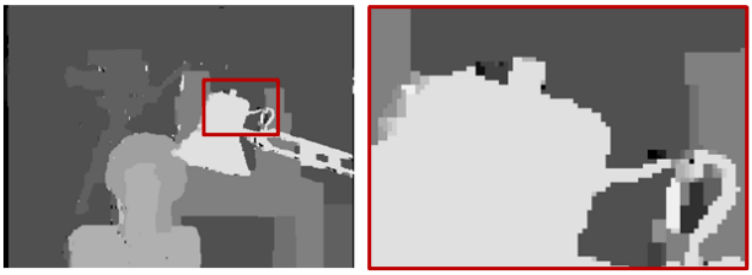
Во втором подходе, в *модели Поттса*, мы штрафуем не линейно, а за каждый подход добавляем некоторый фиксированный штраф. В этом случае мы минимизируем число разрывов диспаритета и вместо плавных переходов на границах объектов делаем один разрыв (рис. 196).

Для некоторых примеров за счет единого штрафа, мы «оторвем» объект переднего плана от фона, и граница будет резкой. Если мы будем наблюдать стену, которая от нас «уходит», между каждой парой соседних пикселов будет меняться диспаритет. С точки

зрения штрафа, это неэффективно, поэтому метод постарается уменьшить число переходов, и гладкая стена превратится в набор параллельных фрагментов ступенчатых плоскостей, что не очень хорошо.

Если наша задача – отделить объекты, расположенные на разном расстоянии от наблюдателя, друг от друга, например, для автоматической сегментации, то тогда лучше штрафовать резкие переходы по модели Поттса. Если задача – в целом оценить карту глубины, когда в сцене объект одновременно расположен на разном удалении, тогда больше подойдет линейная модель.





Пример результатов для модели Поттса. Справа показан увеличенный фрагмент

рис.196

Еще одна проблем глобальных методов – это неэффективность задачи оптимизации для произвольных потенциалов. Плотное стерео – задача многоклассовой разметки. Эффективное решение на графе общего вида существует лишь для выпуклых относительно $|d_p - d_q|$ парных потенциалов, но необходимо использовать модели, сохраняющие границы, а они невыпуклы относительно $d_p - d_q$. Задача становится NP-полной. Поэтому необходимы приближенные алгоритмы (Fusion move, Loopy belief

propagation, TRW). Альтернативный вариант -уход от графов общего вида к деревьям (рис. 197).

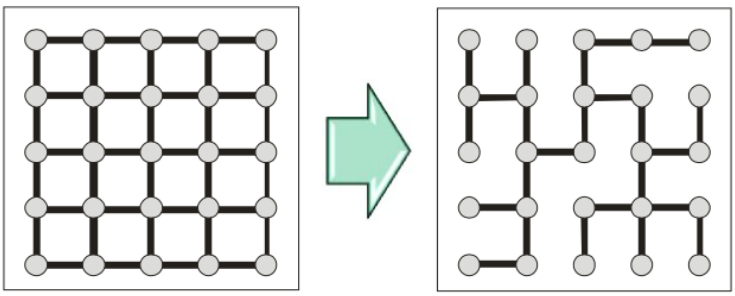


рис.197

Мы можем решить задачу оптимизации функционала энергии с помощью динамического программирования. Динамическое программирование позволяет найти глобальный минимум, произвольную энергию и работает достаточно быстро. Главный вопрос: какие ребра убирать? Самый исторически первый метод – оптимизация вдоль строк изображения Scanline Optimization (рис. 198)

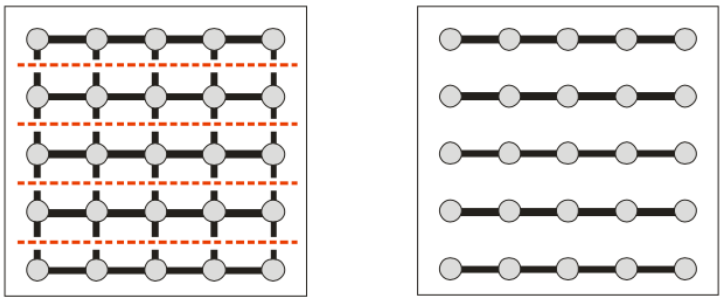


рис.198

Удаляются все вертикальные ребра. Мы можем решать задачу для каждой пары строк, так как знаем, что соответствующие пикселы располагаются в соответствующих строках изображения. Метод работает достаточно быстро, но в строках могут быть ошибки, то есть существенное рассогласование строк между собой (horizontal streaking), так как вертикальные связи не учитываются (рис. 199).

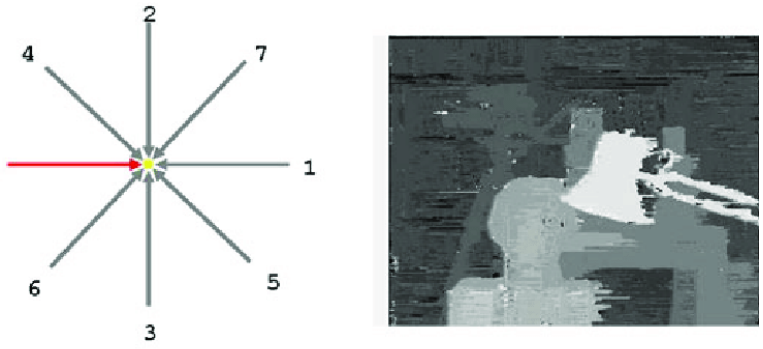


рис.199

Другой вариант – алгоритм на основе MST. Идея заключается в том, чтобы не форсировать гладкость между пикселями разного цвета, то есть если пиксели похожи друг на друга по цветам, то мы будем стремиться к тому, чтобы диспаритет между ними отличался несильно. Соответственно каждому ребру (паре пикселей p и q) присваиваем вес пропорционально похожести пикселам по цветам и строим минимальное покрывающее дерево (minimum spanning tree, MST) для всего графа:

$$w(p,q) = |l(p) - l(q)|$$

Получается метод, который работает тоже достаточно быстро. За счет того, что мы учитываем как горизонтальные, так и вертикальные ребра, рассогласования становится намного меньше (рис. 200 b).

Другой алгоритм Semi-Global Matching. В каждом пикселе строится свое дерево, и оптимизация производится вдоль лучей исходящих их пикселя. Построим такое дерево, как на рис. 201, звездочку с центром в рассматриваемом пикселе. Примерим для этой звездочки метод, который находит диспаритет для всех пикселов, но запомним решение только в центральном пикселе. Получится полуглобальный-полулокальный метод, у которого нет рассогласованности, но есть «изолированные» пиксели (рис. 200 c).

Также есть алгоритм Simple Tree (рис. 200 d), идея которого заключается в следующем: в каждом пикселе строятся два дерева, совместно покрывающих все изображение.

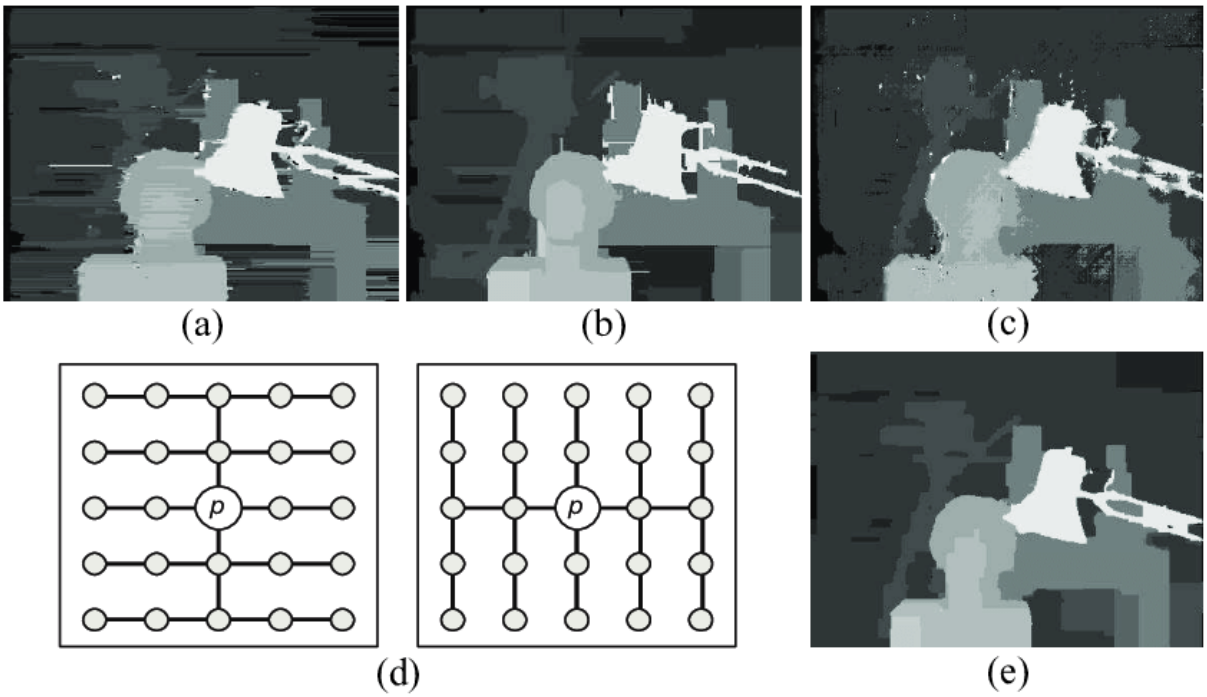


рис.201

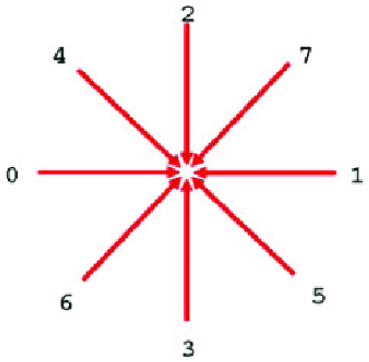


рис.200

В таком дереве учитываются как горизонтальные, так и вертикальные ребра независимо друг от друга. Получается метод, который обеспечивает наибольшее сглаживание (рис. 200 е). Его результат близок к глобальному методу, но в то же время обгоняет его по скорости работы.

Использование сегментации

До появления нейросетевых методов наилучший результат обеспечивался методами на основе сегментации изображения. Если мы знаем, что область однородная по своим визуальным характеристикам скорее всего принадлежит одному и тому же объекту, то

мы можем решать задачу не в пространстве пикселов, а в пространстве этих областей. Для этого применим метод пересегментации и независимо применим попиксельный жадный алгоритм для оценки диспаритета. После этого в каждом сегменте будем форсировать гладкость. Эта гладкая поверхность может быть плоскостью, каким-то сплайном, и мы можем подгонять ее разными способами: например, если мы считаем, что какие-то пикселы ошибочно возникают в этой области, то мы можем применить рандомизированный метод RANSAC, в котором будем выбирать несколько точек, проводить через них плоскость и смотреть, насколько хорошо она соответствует. Таким образом в каждом сегменте получаем свою плоскость и меняем параметры этих плоскостей совокупно по всему изображению, чтобы обеспечить похожесть по всему изображению.

Базовый алгоритм на основе сегментации

- Пересегментация
- Инициализация решения

Любой локальный алгоритм на пикселях

- Аппроксимация сегментов гладкими поверхностями

Модель плоскость, B-сплайн

Метод RANSAC, голосование и т.п.

- Уточнение разметки сегментов

Iterated Conditional Modes (ICM), Cooperative Optimization

Методы с сегментацией позволяют повысить надежность в областях без текстуры и снизить размерность задачи (оптимизация на уровне сегментов). Но такие методы не защищают от нарушения предположения сегментации, не решают проблему перекрытий (ее все равно надо решать на пиксельном уровне) и достаоочно сложны в плане выбора модели, описывающей изменения диспаритета внутри сегмента.

Резюме бинокулярного стерео

- Бинокулярное стерео разбивается на 3 шага: ректификация, стереосопоставление и триангуляция
- Сейчас для стереосопоставления используют комбинацию нейросети для вычисления похожести пикселов и оптимизации по изображению
- Можно использовать карты сегментации и локальную оценку параметров поверхностей
- Методы продолжают активно развиваться

Многовидовая реконструкция

На вход подается N изображений (в один момент времени) объекта с известной калибровкой. На выходе нужно получить 3D модель объекта -геометрическую модель и текстуру изображения. Такая задача многовидовой трехмерной реконструкции в отличие от задачи стерео плоха с точки зрения данных. Мы можем получить много сырых данных, для которых очень тяжело получить эталонную разметку. Поэтому долгое время не было хороших датасетов для трехмерной реконструкции (да и сейчас можно сказать, что их нет).

Benchmark	Setting	Resolution	Online Eval.	6DoF Motion	MVS	Stereo	Video	Varying FOV
Middlebury MVS [38]	Laboratory	0.3 Mpx	✓		✓			
Middlebury [32, 33]	Laboratory	6 Mpx	✓			✓		
DTU [17]	Laboratory	2 Mpx			✓			
MPI Sintel [1]	Synthetic	0.4 Mpx	✓	✓		✓	✓	
KITTI [6, 24]	Street scenes	0.5 Mpx	✓		✓	✓	✓	
Strecha [40]	Buildings	6 Mpx		✓	✓			
ETH3D (Proposed)	Varied	0.4 / 24 Mpx	✓	✓	✓	✓	✓	✓

Table 1. Comparison of existing state-of-the-art benchmarks with our new dataset. Among other factors, we differentiate between different scene types (*e.g.*, staged scenes captured in a laboratory vs. synthetic scenes), whether the camera undergoes a restricted or a full 6 degrees-of-freedom (DoF) motion, or whether cameras with different fields-of-view (FOV) are used.

рис.202

Самое лучшее, что есть на данный момент – набор ETH3D (рис. 202), который был предложен в 2017 году. В нем есть небольшое количество сцен, снятых камерами разного вида как высокого разрешения, так и низкого. Для того, чтобы снять датасет, авторы работы сделали специальную тележку, на которой стоял набор камер, GPS датчик, лидар трехмерный. Они выполнили реконструкций нескольких помещений, очень точно сопоставили результаты лидара с местами, с которых были получены снимки, и получили эталонные карты. Каждая сцена описывалась от 15 до 40 изображениями.

Основный принцип решения задачи многовидового стерео – принцип *фотосогласования.* Предположим, что мы наблюдаем трехмерный объект с двух ракурсов. Возьмем точку на поверхности этого объекта и спроецируем его на изображения, с которых эта точка действительно видна, в этом случае окрестности этой точки на обоих изображениях будут визуально похожи (рис. 203).Если они визуальны похожи, то мы говорим, что это точка *фотосогласованна.*

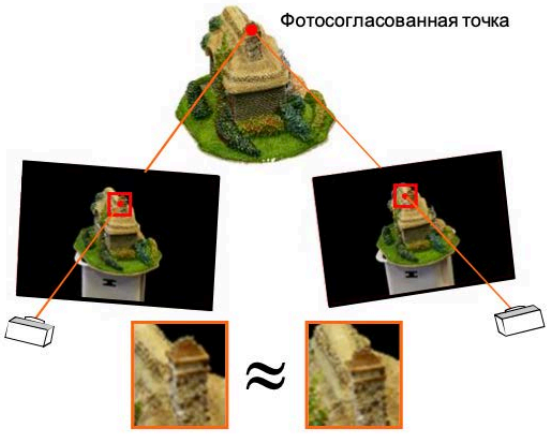


рис.203

Если точка не располагается на поверхности объекта (рис. 204), тогда окрестности, в которые она проецируется, будут непохожи друг на друга. Это не фотосогласованная точка.

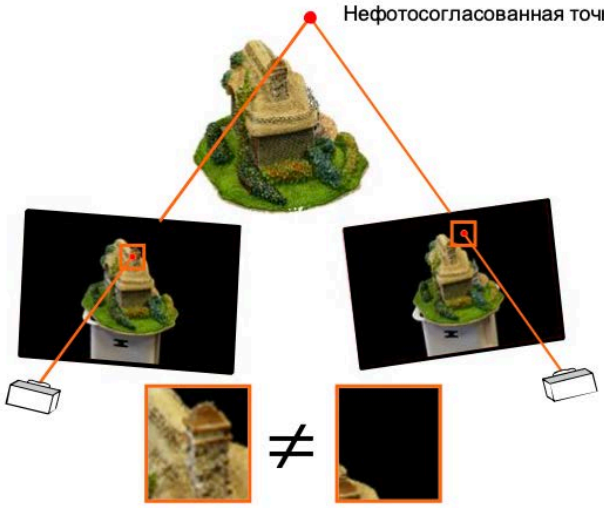


рис.204

Основной принцип - построить такую трехмерную модель, в которой каждая точка будет фотосогласованна по отношению к исходным камерам. Однако остаются те же проблемы метрики, что и в бинокулярном методе: повтор текстуры, отсутствие текстуры, блики. Возникает еще отдельная собственная проблема – видимость. С самого начала мы не знаем, какие точки сцены на каких камерах видны. То есть нам нужно не только обеспечивать фотосогласование точек, но и одновременно определять,

на каких камерах точка видна. Например, несколько камер рассматривают круглый объект (рис. 204). Рассматриваемая зеленая точка видна с трех камер, и именно с этих камер она должна быть фотосогласованна.

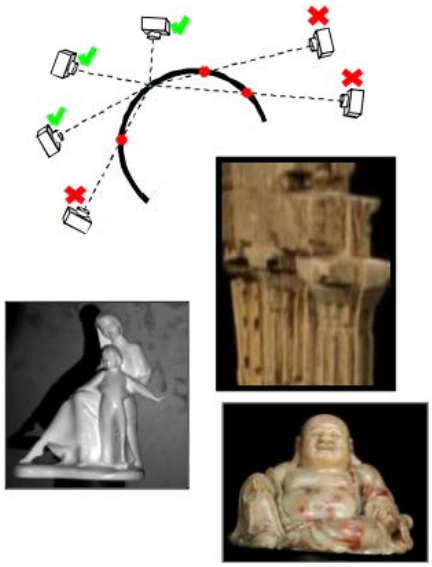


рис.204

Если мы попробуем добиться фотосогласования с другой камерой, то поскольку эта точка не видна, то фотосогласования мы добиться не сможем, а значит мы ошибемся с трёхмерной реконструкцией.

К этой задаче есть несколько подходов. Первый подход опирается на вычисление карты глубины и облаков трехмерных точек (рис. 205):

1. Вычисляем карту глубины

Для каждой точки будем находить близкие к ней точки, решать задачу бинокулярного стерео и получать карту

2. Объединяем карты глубины в объеме в облако трехмерных точек, лежащих на поверхности
3. Извлекаем 3D поверхность
4. Для каждой точки находим соседей, связываем их, получаем сеточную модель

рис.205

Извлечение поверхности – сама по себе очень сложная задача. Для этого объединяем облака точек от карт плотного стерео во всех опорных изображениях (рис. 206), триангулируем сетку и фильтруем выбросу, сглаживая шумы.

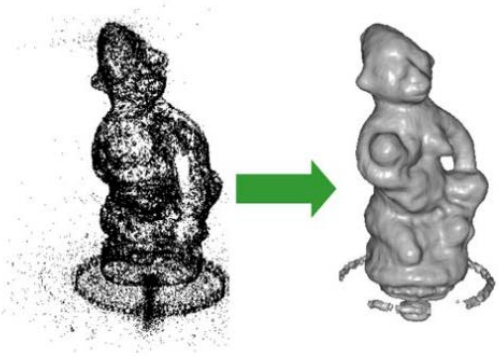


рис.206

Далее оптимизируем сетку, добиваясь фотосогласованности поверхности. Используем, например, градиентный спуск или его модификацию, чтобы модифицировать вершины сетки для увеличения степени фотосогласованности. На этом этапе можно е строить текстуру, поскольку каждому пикселу соответствует своя точка, для которой мы можем вычислить цвет. То есть у нас будет множество точек, для которых вычислен цвет, и в треугольнике, соединяющем точки, этот цвет будет плавно меняться между точками. Варьируя координаты точек, мы можем увеличивать фотосогласование. Оптимизация сетки позволяет удалить ошибки из облака точек и получить гладкие и точные поверхности.

Другой подход классического компьютерного зрения (без использования нейросетевых технологий), который позволил добиться наилучших результатов – методы на основе фрагментов PMVS (Patch-based Multi-View Stereo). Первый метод был предложен в 2005 году и затем совершенствовался в отдельных его деталях. Основа этого метода – фрагмент (patch) – модель небольшого участка поверхности объекта. Поскольку участок небольшой, то его можно описать плоской поверхностью (квадратом). Мы

представляем, что объект состоит из маленьких плоских квадратных фрагментов. Каждый фрагмент p определяется (рис. 207)

- Положением c(p)
- Нормалью n(p)
- Видимыми изображениями V(p)

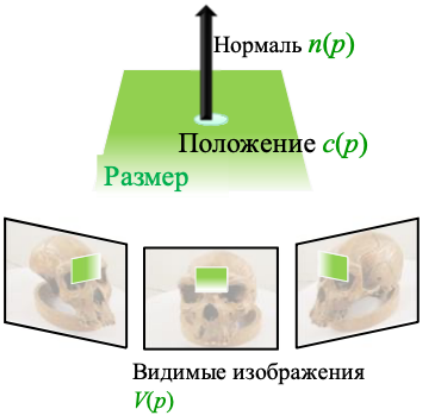


рис.207

Размер фрагмента выбирается так, чтобы p был примерно 9x9 пикселей в V(p). То есть размер фрагмента в трехмерном пространстве выбирается динамически в зависимости от разрешений исходных изображений и удаления (удаленный фрагмент должен быть большим, чтобы проецироваться в 9x9 пикселов, а поверхность точки, расположенной близко к изображениям, будет моделироваться более детально, поскольку в ней для проецирования в 9x9 пикселов нужно выбрать совсем маленькую окрестность. За счет того, что этот фрагмент хорошо различим на всех изображениях, на которых он виден, можно достаточно надежно оценить его фотосогласованность. Вычислим фотосогласованность $N(I, J, p)$ фрагмента p между изображениями I и J (рис. 208). Для этого разобьем фрагмент сеткой 9x9 точек, оценим цвет I_{xy} каждой точки с точки зрения первого изображения. Это мы можем сделать, спроецировав все точки на соответствующие изображения и измерив цвет каждой точки, например, с помощью билинейной интерполяции.

Затем точно так же оценим цвет каждой точки фрагмента с точки зрения второго изображения. Получится, что есть два изображения размером 9x9 пикселов, то есть две оценки цветов данного фрагмента с точки зрения двух изображений. Посчитаем похожесть этих двух изображений, например, с помощью нормализованной кросс-корреляции (рис. 209).

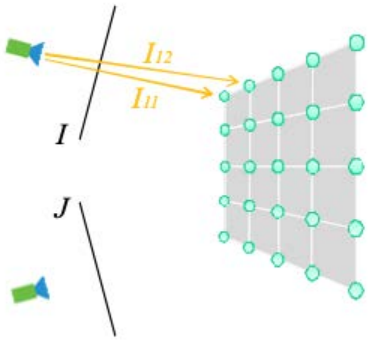


рис.208

$$N(I,J,p)=\frac{\sum(I_{xy}-\overline{I_{xy}})\cdot(J_{xy}-\overline{J_{xy}})}{\sqrt{(I_{xy}-\overline{I_{xy}})^2}\sqrt{(J_{xy}-\overline{J_{xy}})^2}}$$

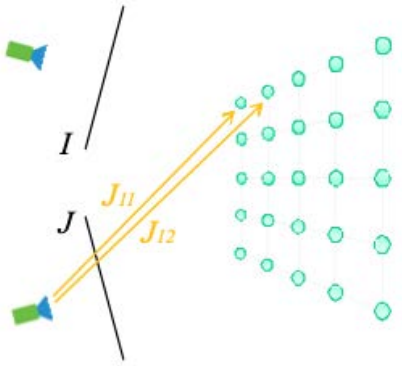


рис.209

Таким образом, оценим, насколько этот фрагмент согласован с точки зрения этих двух изображений. Если у нас более, чем 2 изображений, тогда для оценки фотосогласования нужно посчитать фотосогласованность фрагмента по всем парам изображений и усреднить получившееся значение.

$$N(p)=\frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n}N(I_i,I_j,p)}{(n+1)n/2}$$

Таким образом, мы можем оценить фотосогласованность объекта по произвольному количеству изображений. Метод на основе фрагментов тоже опирается на поиск особых точек.

Алгоритм на основе фрагментов:

1. Поиск особых точек

Работаем с теми точками, которые хорошо можем сопоставить друг с другом. Применяем детектор особых точек, например, SIFT и находим множество особых точек на всех изображениях.

2. Инициализация фрагментов

Для этого перейдем к изображениям и получим следующую ситуацию (рис. 210). Есть особая точка на первом изображении, найдем соответствующую ей особую точку на втором изображении.

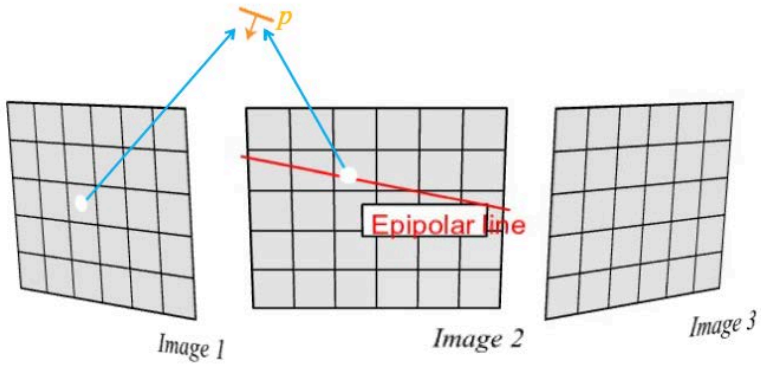


рис.210

Будем просматривать точки, лежащие близко на эпиполярной линии на втором изображении. Найдем особую точку наиболее похожую по дискриптору, получим пару соответствующих точек, для которой инициализируем фрагмент. Поскольку мы можем триангулировать положение трехмерной точки, соответствующей данной паре, то мы можем определить центр фрагмента. Инициализируем нормаль этого фрагмента таким образом, чтобы сам фрагмент располагался параллельно первому изображению. Поскольку мы триангулировали его по первым двум изображениям, мы добавим в множество видимых изображений первые два изображения (рис. 211). Далее будем искать, каким точкам в новом изображении он может соответствовать. Спроецируем фрагмент в третье изображение и проверим степень фотосогласованности по всем трем изображениям. Если этот фрагмент окажется фотосогласованным достаточно неплохо по всем трем изображениям, значит, на третьем изображении он виден, и мы добавим третье изображение в множество изображений, с которых этот фрагмент виден. Далее уточним фрагмент: $\{c(p), n(p)\} = argmax_{\{c(p),n(p)\}} N(p)$. Для этого будем подбирать

такие параметры положения фрагмента c(p) и нормали объекта n(p), чтобы максимизировать меру согласования по трем изображениям

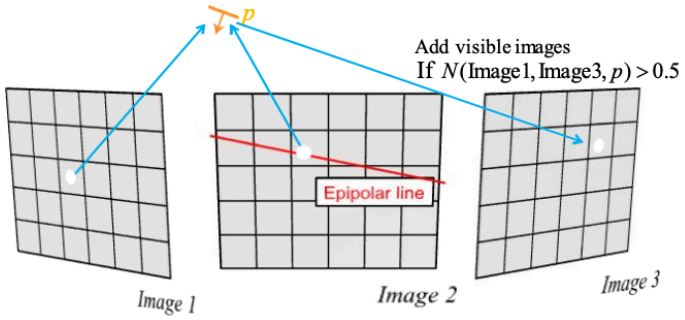


рис.211

После оптимизации убедимся, что фрагмент действительно фотосогласован и виден на всех трех изображениях: обновим V(p) и проверим $|V(p)| \geq 3$. Теперь появилась начальная реконструкция все кадров (рис. 212). Для тех пикселов, в которых были особые точки, мы вычислили соответствующий этим точкам фрагмент и на всех изображениях, на которых он виден, соответствующие пикселы запишем как реконструированные. И так пройдемся по всем особым точкам.

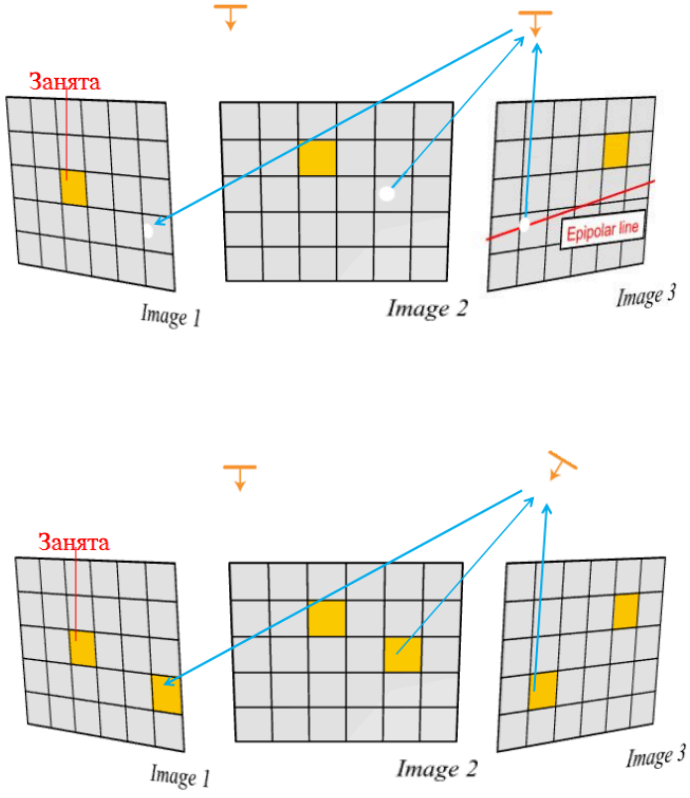


рис.212

3. Расширение фрагментов и фильтрация

В итоге часть пикселов получит свои фрагменты, а часть останется пустыми (рис. 213). Возможно, из-за того, что в этих пикселах мы не нашли особые точки, так как это были однородные поверхности или из-за того, что особую точку нашли, но не нашли ей пару и т.д. Начинаем достраивать поверхность объекта множеством фрагментов в пустых соседях. Для этого сначала находим пустого соседа, потом строим плоскость, проходящую через соседний фрагмент, находим линию пересечения с этой плоскостью, пропущенную через соседний пиксель и инициализируем в данной пустой точке новый фрагмент, у которого будет та же самая нормаль, то же множество видимых изображений, что и у соседа, но центр будет лежать на месте пересечения плоскости (рис. 214).

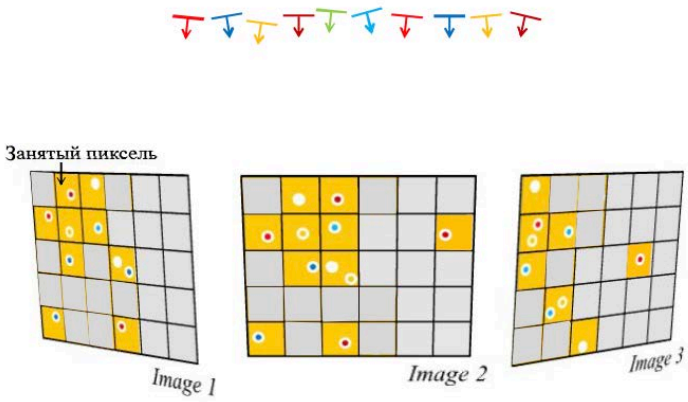


рис.213

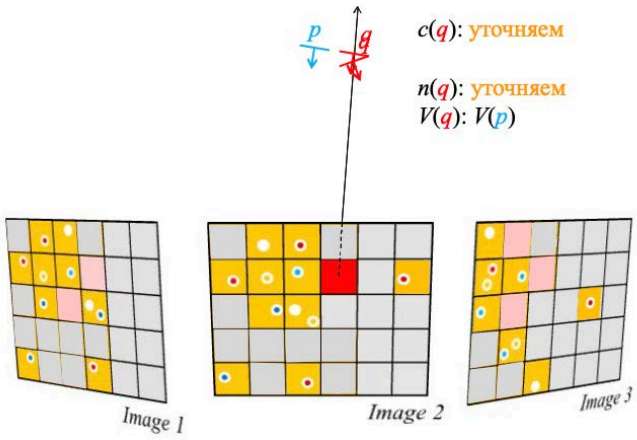


рис.214

Далее будем уточнять этот фрагмент, максимизируя фотосогласование, и продолжать процедуру. Пока остаются пустые пикселы. В итоге каждому пикселу будет соответствовать свой фрагмент. Из-за того, что этот процесс итеративный, могут возникнуть ситуации, когда по одним изображениям был создан фрагмент, который на других изображениях перекрывает какой-то фрагмент, то есть возникают *коллизии*. То есть, когда мы обрабатывали отдельно фрагменты, они были видны, а потом мы создали какой-то новый фрагмент, который их перекрыл (рис. 215). В этом случае мы выбираем, что лучше: оставить перекрывающий фрагмент или множество фрагментов, которое он перекрыл.

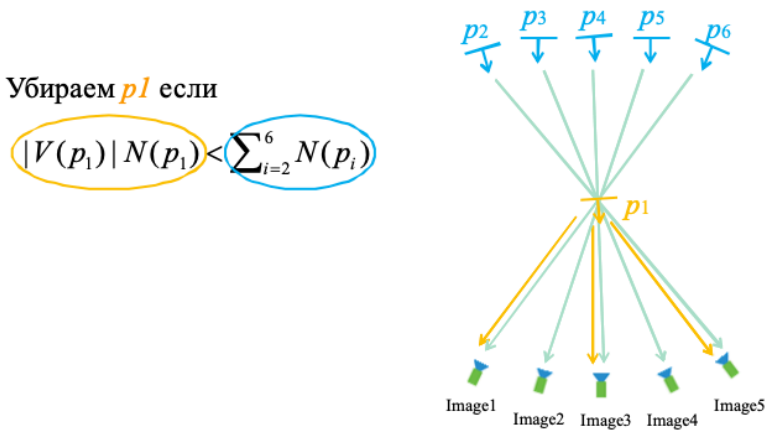


рис.215

Такой долгий итеративный подход позволяет реконструировать сложные трехмерные сцены с высокой точностью. По оценке на ETH3D методы на основе подгона фрагментов MVS обеспечивают наибольшую точность трехмерной реконструкции (рис. 216).

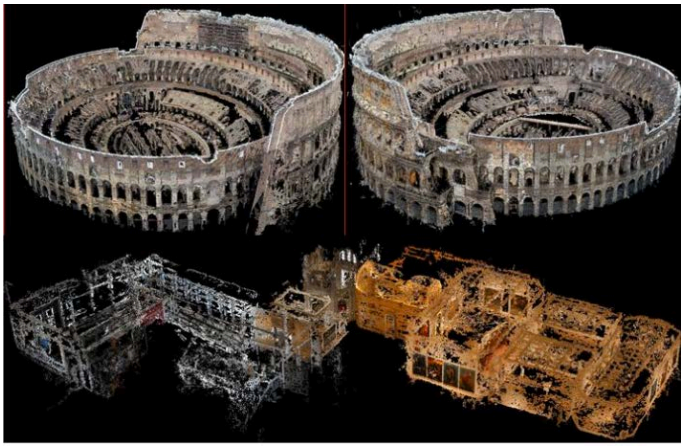


рис.216

Далее развитие плотных методов реконструкции шло в направлении ускорения реконструкции. Когда мы решали задачу построения трехмерных моделей по множеству изображений, скаченных из интернета, возникала типичная ситуация: огромное количество изображений, один объект виден на многих ракурсах. Как видно из предыдущей процедуры, время работы существенно зависит от количества изображений, которые мы используем. Чем больше изображений, тем больше вариантов и по большему числу изображений нужно делать фотосогласование. Поэтому для трехмерной реконструкции мы не можем позволить себе использовать все изображения (рис. 217). И поэтому мы кластеризируем изображения и в каждой группе выполняем свою трехмерную реконструкцию из фрагментов. Затем близкие кластеры объединяем друг с другом, то есть, когда мы их кластеризуем мы делаем так, чтобы были какие-то изображения, которые одновременно входят в два кластера. Получив две реконструкции, объединяем их, применяем уточнение фотосогласования для всех полученных фрагментов, получаем уточненную реконструкцию.

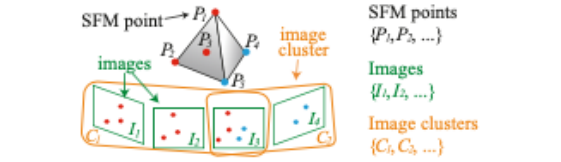


Figure 2. Our view clustering algorithm takes images $\{I_i\}$, SFM points $\{P_j\}$, and their associated visibility information $\{V_j\}$, then produces overlapping image clusters $\{C_k\}$.

рис.217

Для трехмерной реконструкции сформулировали свой вариант *теста Тьюринга*. В рамках этого теста сопоставляем рендеринг трёхмерной реконструкции с фото реальной модели. Поскольку внешний вид трехмерной модели существенно зависит от условий освещения, трехмерная модель с одним фиксированным освещением, а фотография с другим, то для того, чтобы корректно учесть модель, необходимо включить этап подбора освещения, когда мы пытаемся подобрать освещение, которое минимизирует разницу между визуализированной моделью и фотографией. Ту разницу, до которой получилось минимизировать, берем как реальную разницу между трехмерной реконструкции и реальным изображением.

Еще один вектор улучшения метода – интерполяция, то есть детектирование областей, на которых не нашлось фрагментов и интерполирование объектов между этими областями для того, чтобы в реконструкции было меньше дырок. Все эти алгоритмы вместе с разреженной реконструкцией сейчас имплементированы в мощной библиотеке COLMAP, которая выложен в opensource и используется во многих работах.

Сейчас, когда много задач пытаются решить с помощью нейросетей, сталкиваются с отсутствием хороших эталонных данных. Единственный реальный вариант – попробовать применить существующие методы трехмерной реконструкции к каким-то

данным, получить карты глубины (пуская и с ошибками) классическими методами, а затем использовать эти данные для обучения нейросетей.

Реконструкция человека

Поскольку человек – интересный объект, то многие методы бинокулярного и многовидового стерео улучшают для того, чтобы получать качественную реконструкцию человека. Пример (рис. 218) несколько работ из Диснеевской лаборатории в Цюрихе, которая исследует разные способы стереореконструкции тела человека. Общая идея – иерархическое бинокулярное стерео с итеративным уточнением. Строим стереореконструкцию для восстановления модели, делая это в видеорежиме, чтобы получить движущуюся модель. Оказывается, что если мы хотим сделать трехмерную реконструкцию как на рис. 219, то лучше всего полагаться на жадные локальные методы и делать реконструкцию итеративно.

рис.218

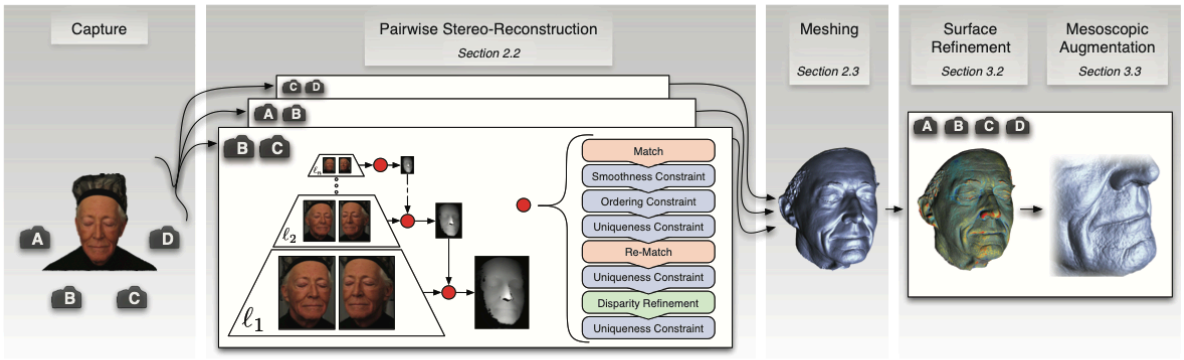


Figure 2: The proposed system - *The subject is captured with multiple cameras. This figure shows a four-camera setup, but the system can incorporate an arbitrary number of cameras.*

рис.219

Для того, чтобы не было дырок, сначала уменьшим изображение, построим карту глубины на уменьшенной копии изображения, увеличим эту копию в 2 раза. Будем

использовать полученную карту как начальное приближение, уточним стереосопоставление на повышенном разрешении и так далее до реконструкции на исходном разрешении. Ход постепенного уточнения трехмерной реконструкции модели с увеличением исходного разрешения можно наблюдать на рис. 220.

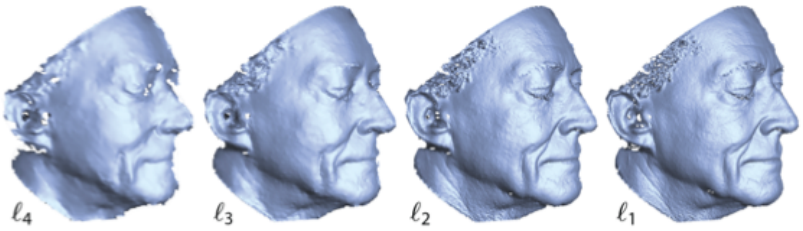


Figure 4: *Reconstructions of a stereo-pair at 4 different layers of the pyramid starting from the coarsest layer ℓ_4 (160px × 160px). The dimensions double at each layer up to the highest resolution layer ℓ_1 (1280px × 1280px).*

рис.220

При этом в этой модели не получится реконструировать мелкие детали, такие как морщины, поскольку они слишком маленькие. Но это можно сделать с помощью искажения геометрии по освещенности. Высокие частоты на фотографиях кожи скорее всего соответствуют морщинкам, значит, мы можем исказить геометрию таким образом, чтобы при освещении возникали такого же рода эффекты (рис. 221).

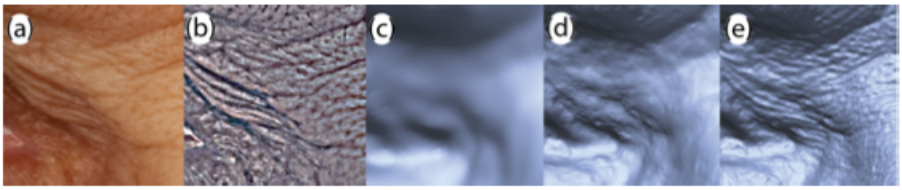


Figure 6: *This figure demonstrates the effect of the mesoscopic-consistency term. The captured image (a) is filtered to extract the mesoscopic detail (b). In (c), the Poisson-reconstructed surface is shown. The refinement described in Section 3.2 already enhances the coarse geometry (d), but only the mesoscopic formulation is capable of reproducing the fine-scale details (e).*

рис.221

Рассмотрим пример системы, которую можно применять на практике для 3D захвата лица. Это пассивная многокамерная система, состоящая из 14 камер. Для того, чтобы обеспечить однородность освещения, используется мощная подвеска с большим количеством светодиодов с разных сторон. Для детализации модели используются 7

стереопар разрешения Sony HDR-SR7. Каждая из стереопар нацелена на свою зону лица человека (рис. 222). За счет такого высокого разрешения и равномерной подсветки поры на коже дают достаточно текстуры для вычисления пассивного стерео (рис. 223).

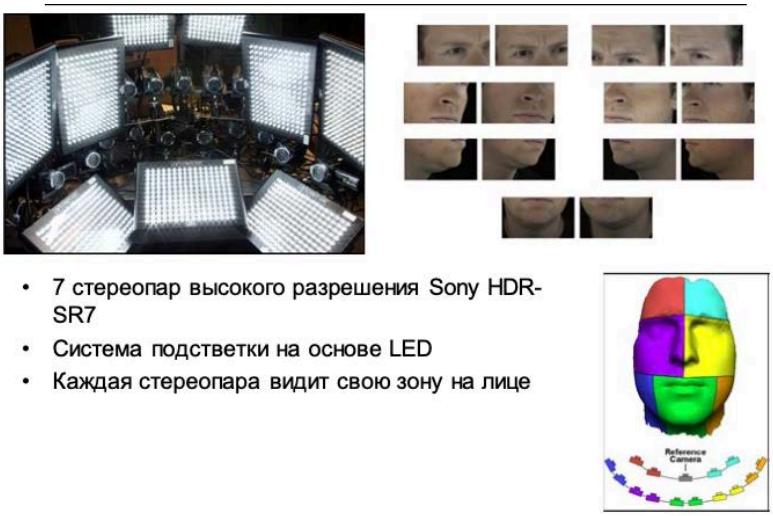


рис.222

рис.223

Если мы обеспечим снимки в таком высоком разрешении, мы сможем получить достаточно детальную реконструкцию, так как каждая стереопара дает карту глубины и облако точек (шумное). Затем все 7 стереореконструкций объединяем в одну модель и получаем сетку для каждого кадра видеопоследовательности отдельно. Но если мы хотим, чтобы наша видеомодель использовалась на практике, нужно получить не реконструкции каждого кадра независимые друг от друга, а одну деформированную модель, которая будет деформироваться от кадра к кадру. То есть нам нужно посчитать деформацию первой модели, первого кадра таким образом, чтобы получить трехмерную реконструкцию, соответствующую модели, полученной на втором кадре. Это можно посчитать как оптимизационную задачу, то есть насколько надо подвинуть вершины полигональной модели на первом кадре так, чтобы совпасть с картой глубины на втором кадре. Для того, чтобы ускорить эту процедуру, можно посчитать оптический поток между парами кадров, который задаст начальное приближение для смещения всех точек (рис. 224).

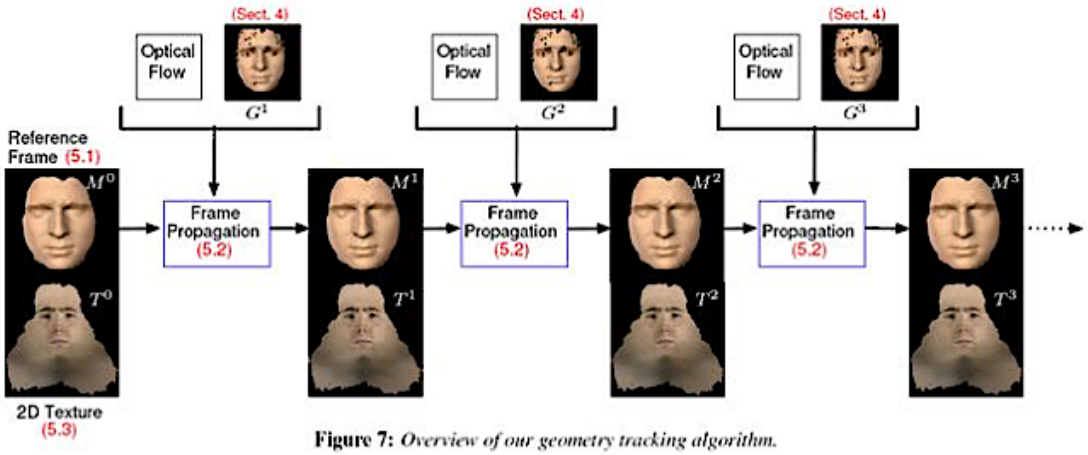


Figure 7: *Overview of our geometry tracking algorithm.*

рис.224

Несмотря на хорошее освещение, с разных ракурсов кожа будет видна немного по-разному, будет бликовать и поэтому придется сделать композицию так, чтобы цветовые переходы в результате были плавными, на стыках текстуры между разными ракурсами не было видно смешений. Участки текстуры копируются со своих камер. Поскольку камеры без цветовой калибровки, при прямом копировании текстуры видны резкие цветовые переходы (артефакты), для обработки которых используется композиция по Пуассону (рис. 225).

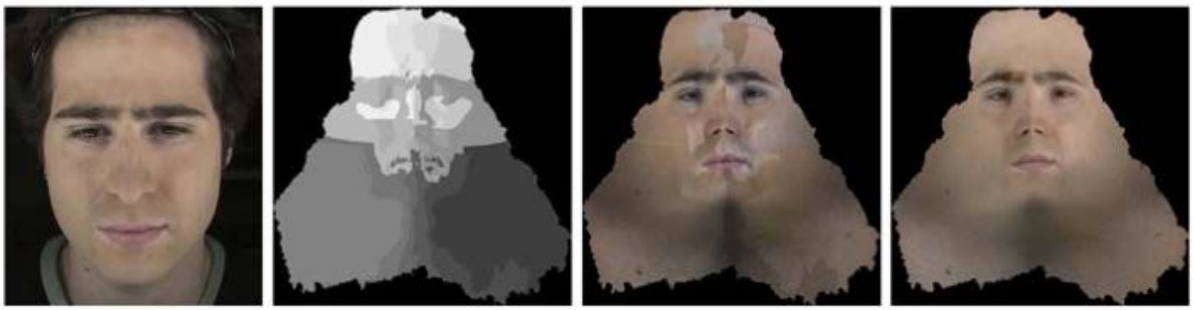


рис.225

Подобного рода технологии в 2011 году начали использоваться на практике, например, в игре L.A. Noire.

Резюме

- Все виды многовидового пассивного стерео опираются на принцип фотосогласования

- Два классических подхода: объединение сеток и распространение фрагментов
- Поверх можно накрутить пост-обработку
- На визуальное качество сильно влияет рендеринг и качество материалов

ФАКУЛЬТЕТ
ВЫЧИСЛИТЕЛЬНОЙ
МАТЕМАТИКИ И
КИБЕРНЕТИКИ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА