

Компьютерное зрение

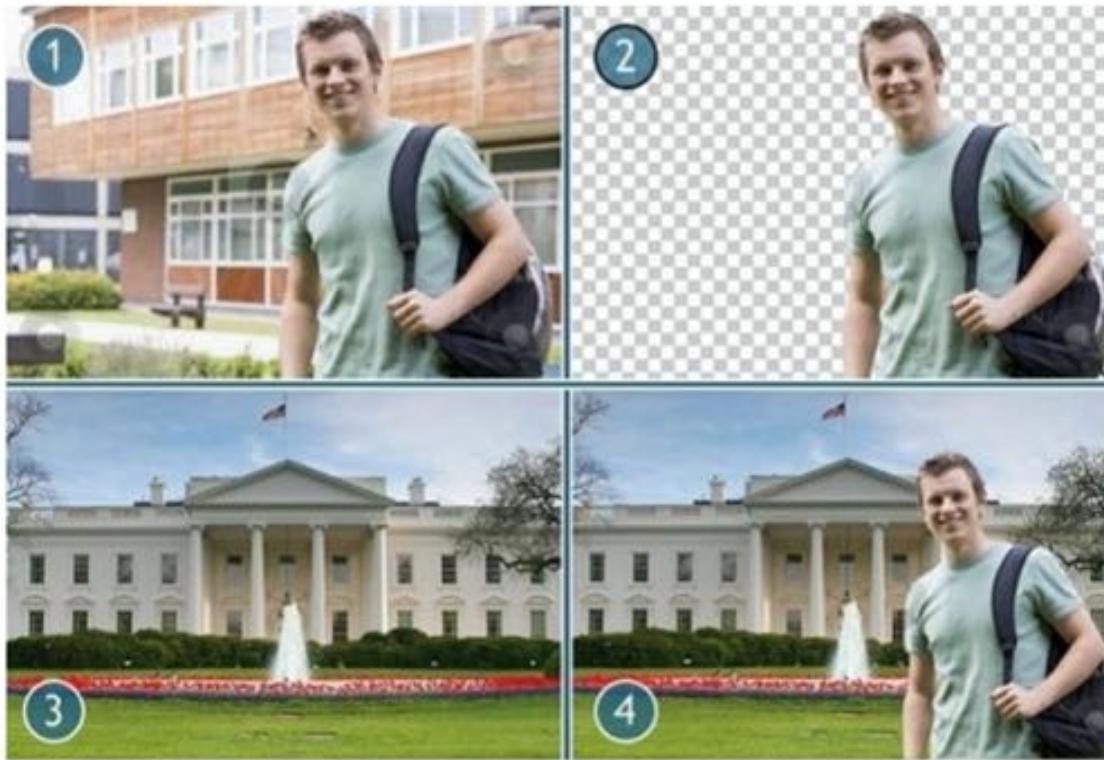
Лекция 6. Сегментация, детекция, трекинг Классические подходы

18.06.2020
Руслан Рахимов

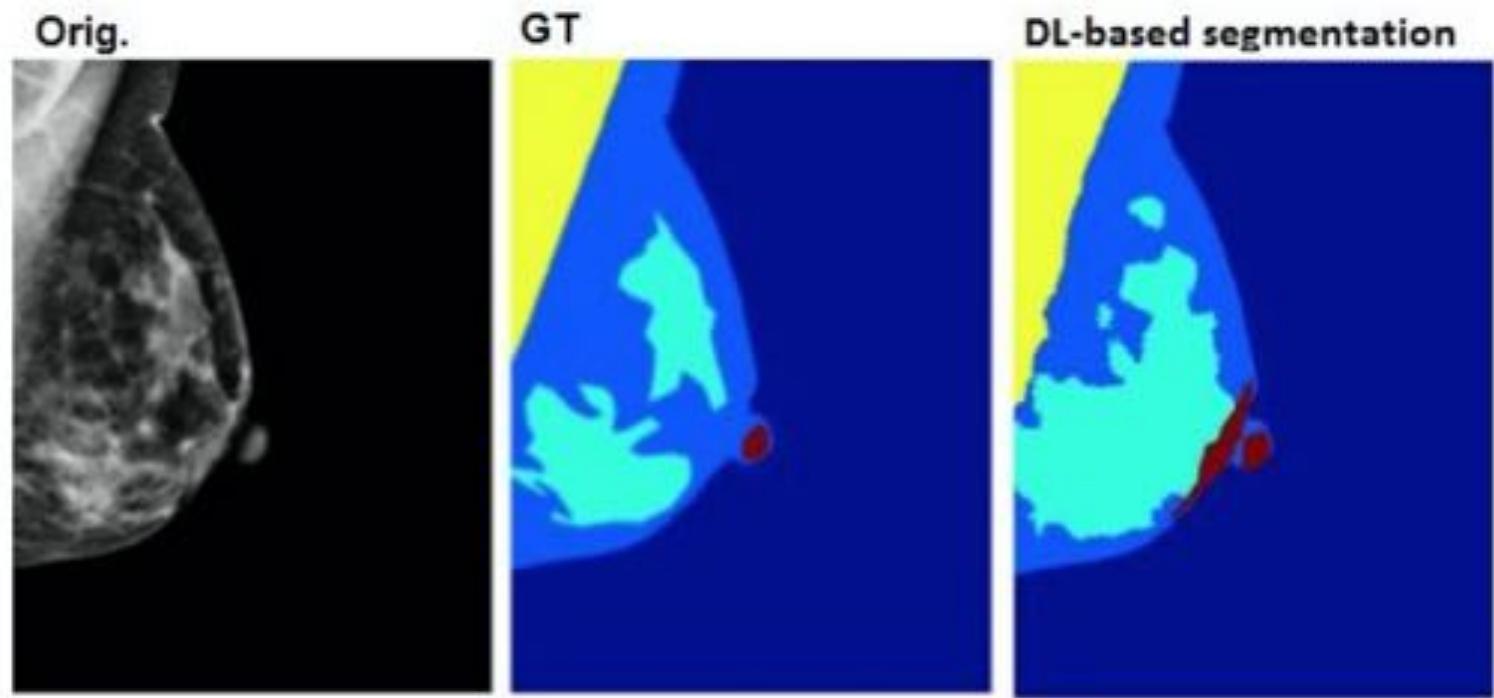
План занятия

- Сегментация
- Детекция
- Трекинг

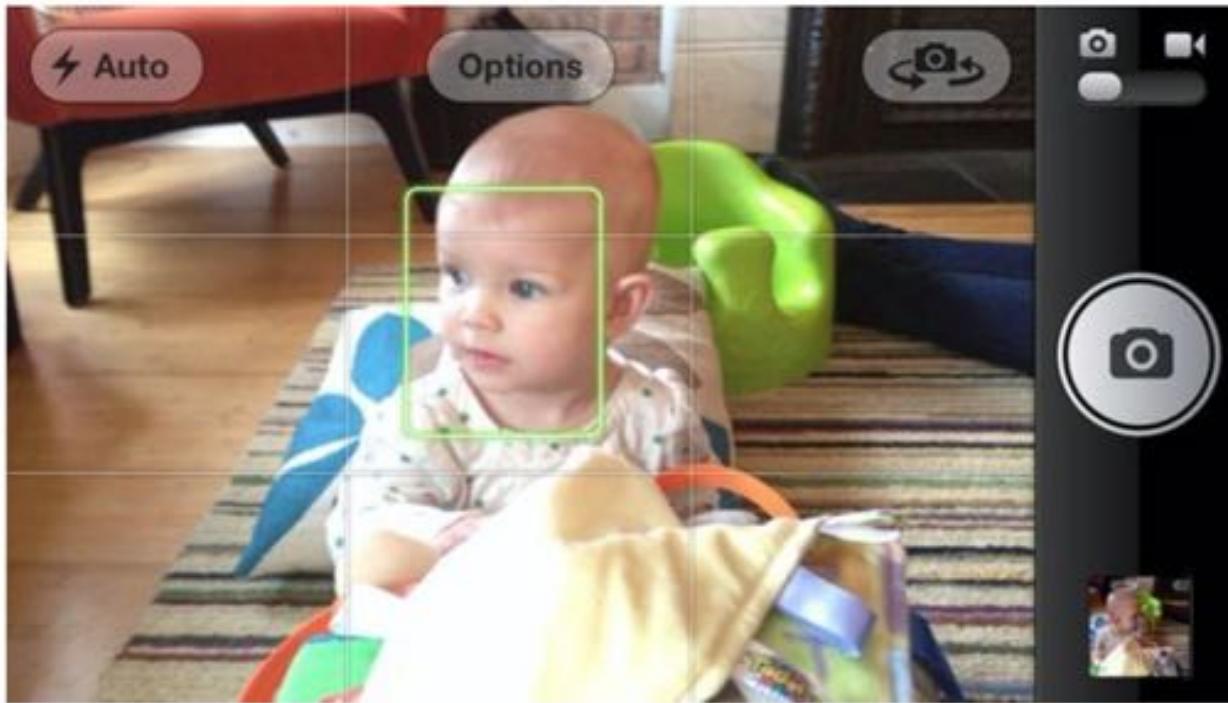
Пример. Удаление фона



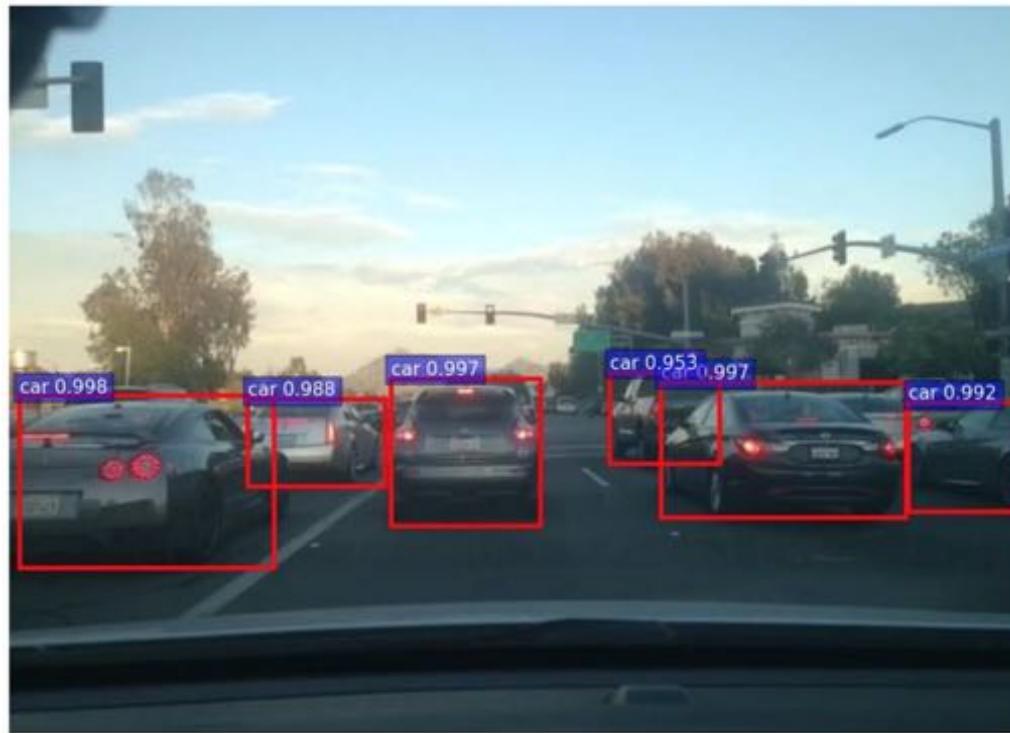
Пример. Сегментация медицинских снимков



Пример. Детекция лица для автофокусировки камеры



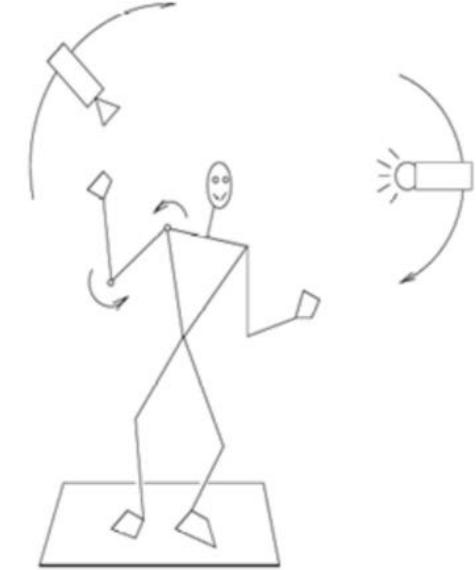
Пример. Детекция и распознавание объектов на дороге



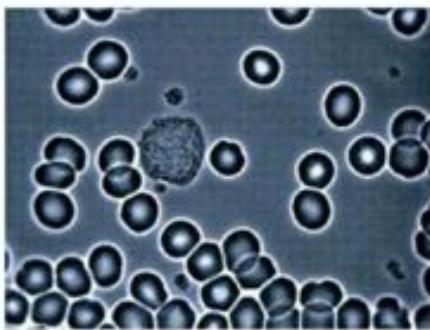
Сегментация

Изменчивость изображений

- Внешние факторы:
 - Положение камеры
 - Внутренние свойства камеры
 - Освещение
- Внутренние факторы:
 - Деформация объектов
 - Внутриклассовая изменчивость
- Пока приходится задачу упрощать, вводя ограничение на ракурс съемки, условия освещения, типы объектов
- Мы будем рассматривать простые случаи, когда все факторы варьируются незначительно



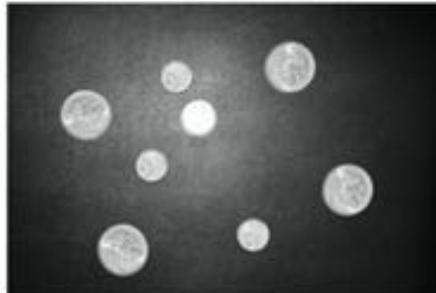
Примеры простых задач



Клетки крови



Ложки и сахар



Монеты и купюры

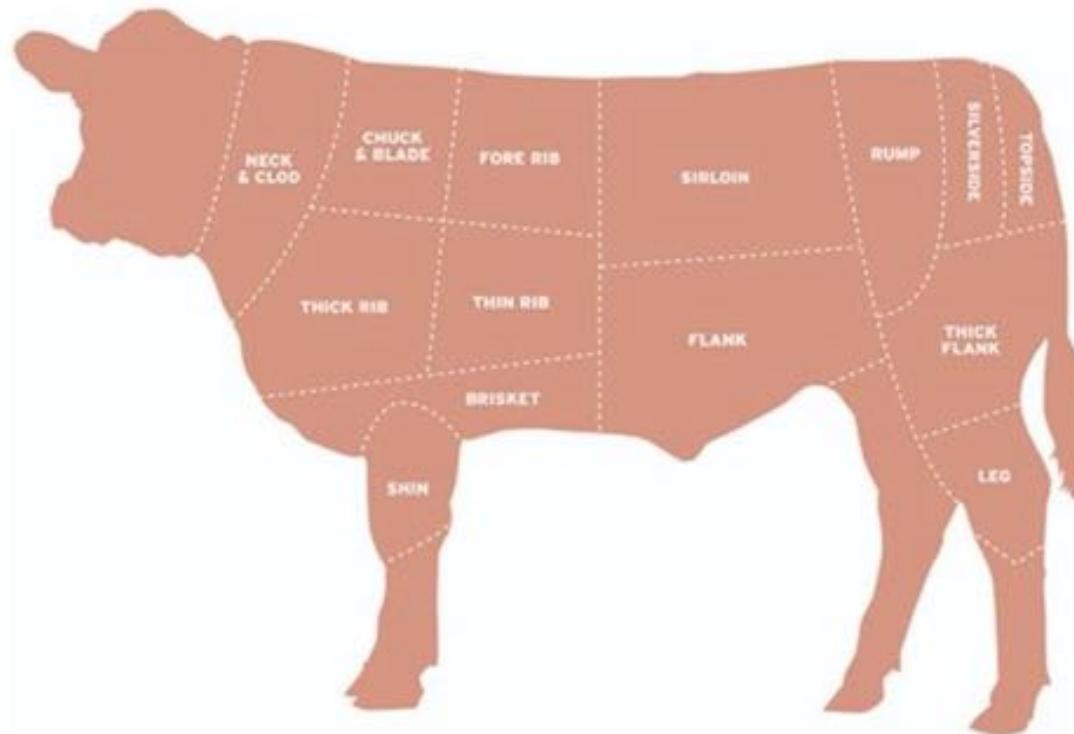


Номера

Из чего состоит изображение?

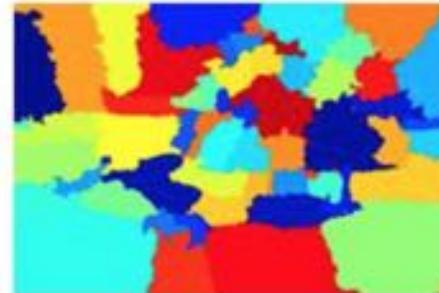


Из кусков - “отдельных объектов”



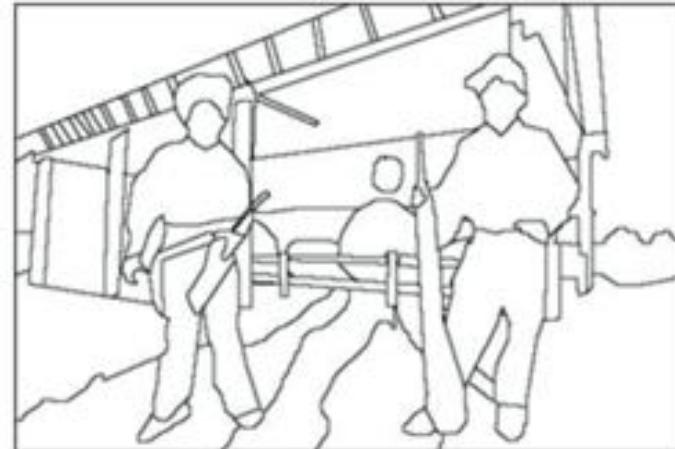
Сегментация

- Сегментация - это способ разделения сцена на “куски”, с которыми проще работать
- Тесселяция - разбиение изображения на неперекрывающиеся области, покрывающие все изображение и однородные по некоторым признакам

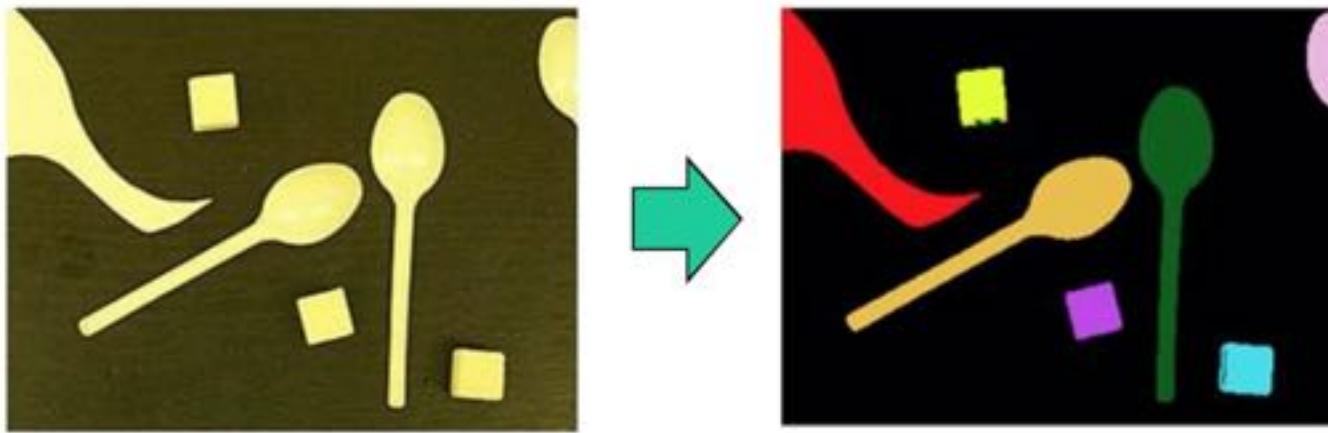


Требования к сегментации

- Сегментация - это способ разделения сцены на «куски», с которыми проще работать
- Границы сегментов должны соответствовать границам объектов



Результат сегментации



- Как мы будем записывать результат сегментации?
- Сделаем карту разметки – изображение, в каждом пикселе которого номер сегмента, которому принадлежит этот пиксель
- Визуализировать удобно каждый сегмент своим цветом

Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

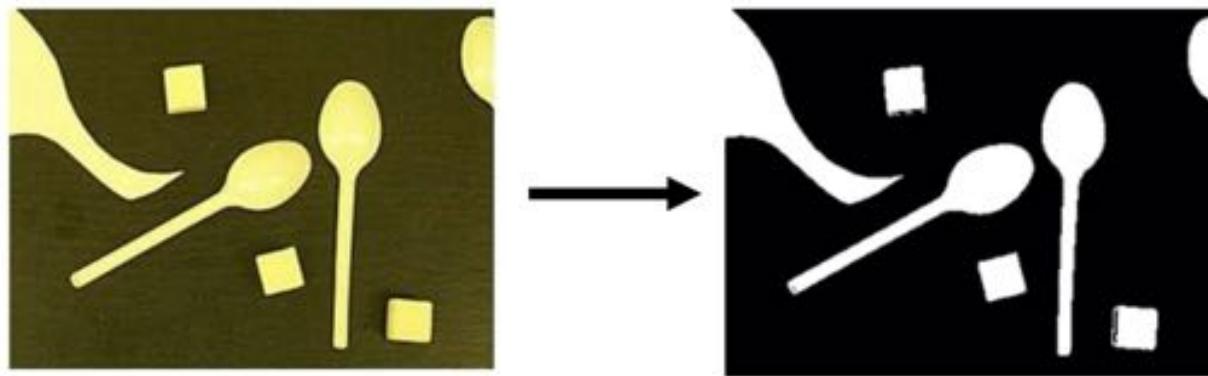
Простейшая сегментация

Чем отличаются объекты на этом изображении?



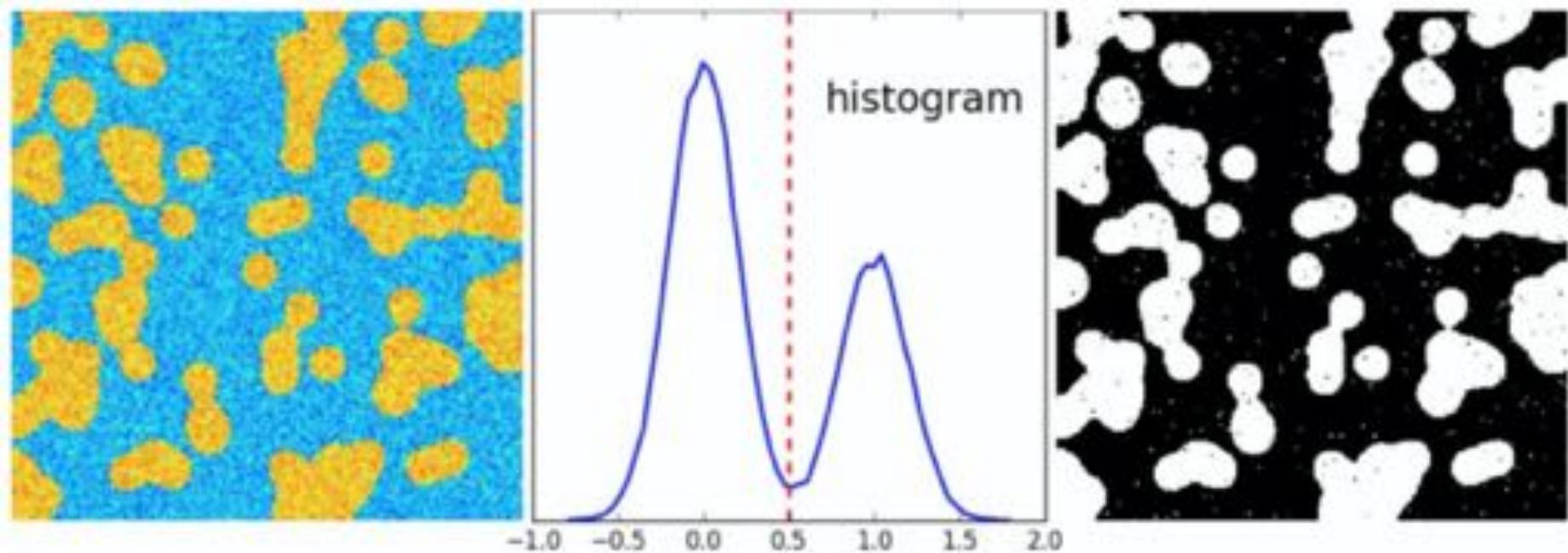
- Все объекты яркие, фон темный
- Объекты контрастны по отношению к простому (однотонному) фону
- Для сегментации такого изображения достаточно:
 - Пороговая бинаризация
 - Обработка шума
 - Выделение связанных компонент

Пороговая бинаризация



- Пороговая фильтрация (thresholding)
 - Пиксели, которых выше/ниже некоторого порога, заданного «извне», помечаются 1
 - Ниже порога помечаются 0
- Бинарное изображение – пиксели которого могут принимать только значения 0 и 1
- Бинаризация - построение бинарного изображения по полуточновому / цветному

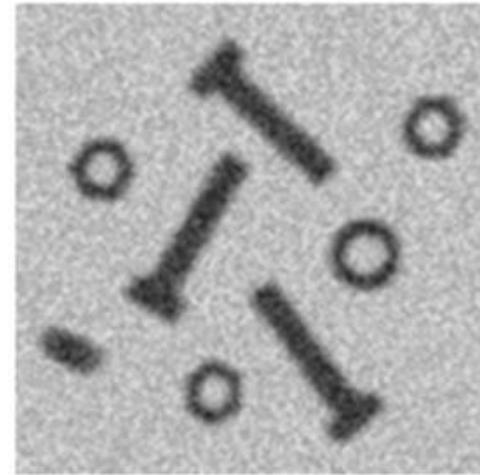
Пороговая бинаризация



Пороговая бинаризация

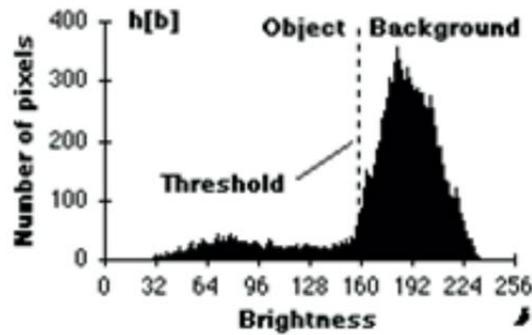
Более интересный способ – определение порога автоматически, по характеристикам изображения

- Нужно проанализировать распределение пикселей по яркости
- Анализ гистограммы яркости



Анализ гистограммы

- Анализ симметричного пика гистограммы
- Применяется когда фон изображения дает отчетливый и доминирующий пик гистограммы, симметричный относительно своего центра



1. Сгладить гистограмму;
2. Найти ячейку гистограммы h_{max} с максимальным значением;
3. На стороне гистограммы не относящейся к объекту (на примере – справа от пика фона) найти яркость hp , количество пикселей с яркостью $\geq hp$ равняется $p\%$ (например 5%) от пикселей яркости которых $\geq h_{max}$;
4. Пересчитать порог $T = h_{max} - (hp - h_{max})$;

Пороговая бинаризация

- прост в реализации
- неустойчив к шумам
- неустойчив к изменению освещенности различных частей изображения

Адаптивный порог

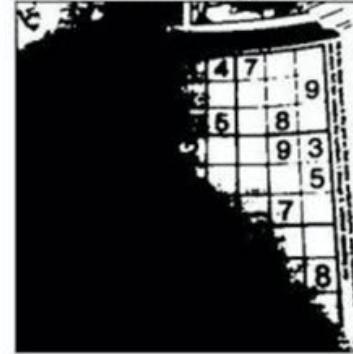
- для каждого пикселя вычисляется усредненное значение интенсивности в окрестности
- в качестве значения порога используют либо само среднее значение
- либо вычисляется взвешенное среднее с гауссовским ядром

Адаптивный порог

Original Image



Global Thresholding ($v = 127$)



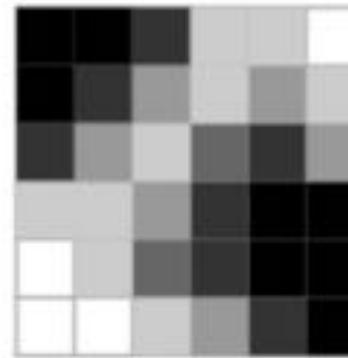
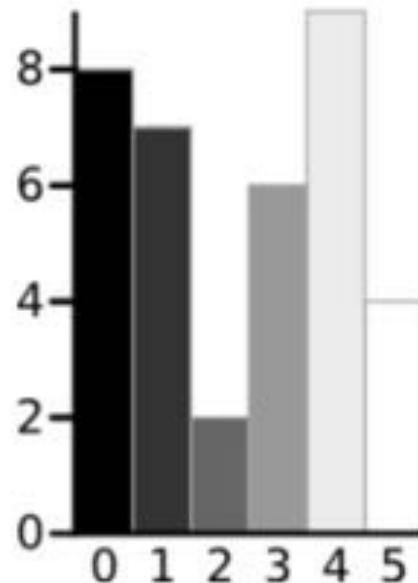
Adaptive Mean Thresholding



Adaptive Gaussian Thresholding

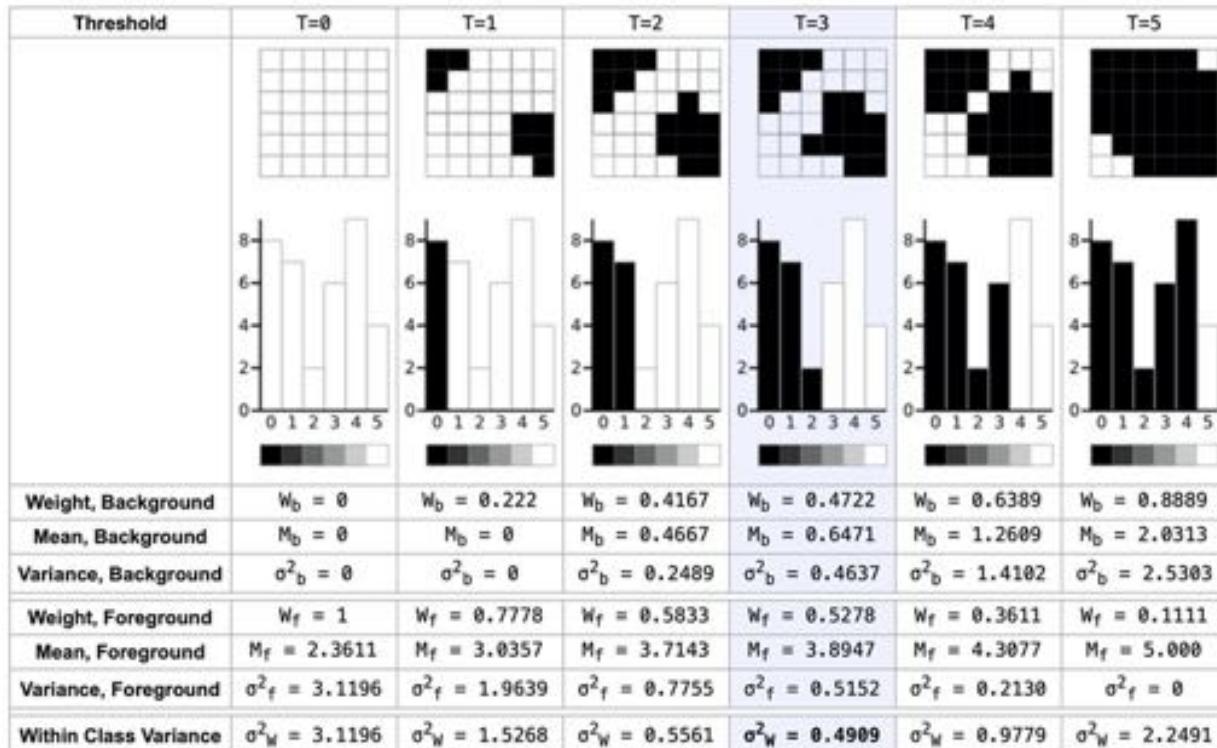


Otsu thresholding



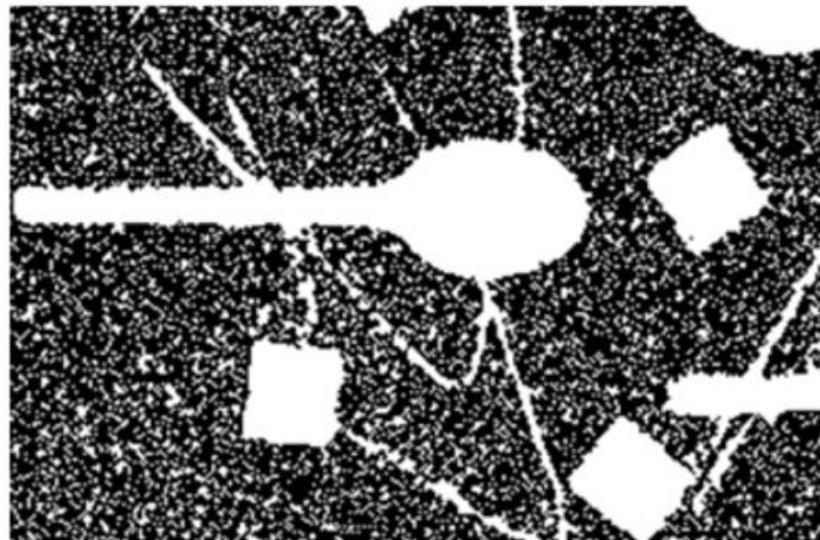
Источник: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>

Otsu thresholding

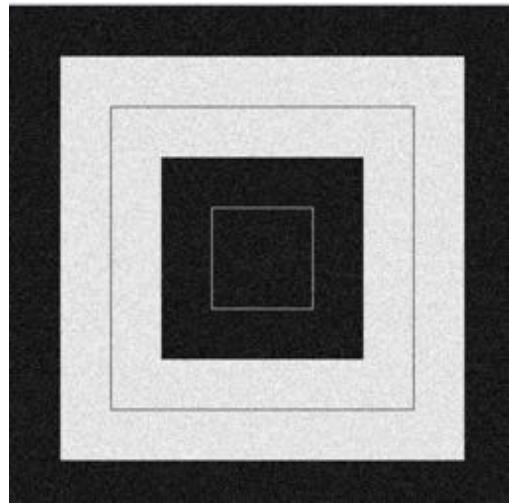


Шум в бинарных изображениях

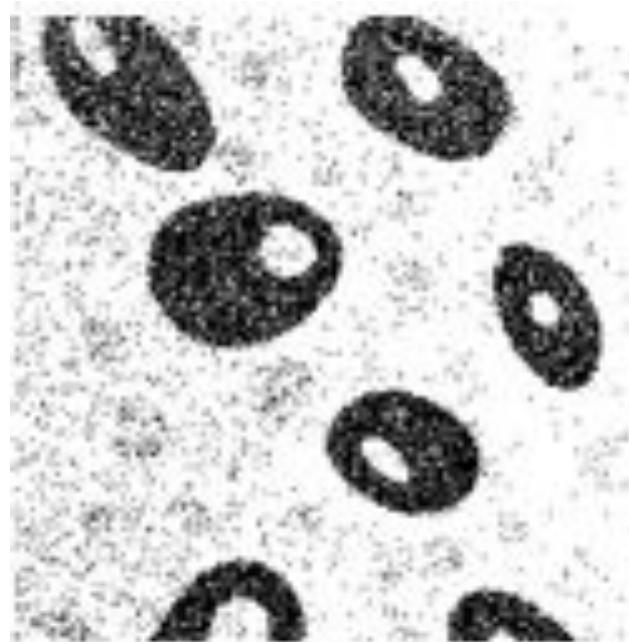
Часто возникает из-за невозможности полностью подавить шум в изображениях, недостаточной контрастности объектов и т.д.



Типы шумов



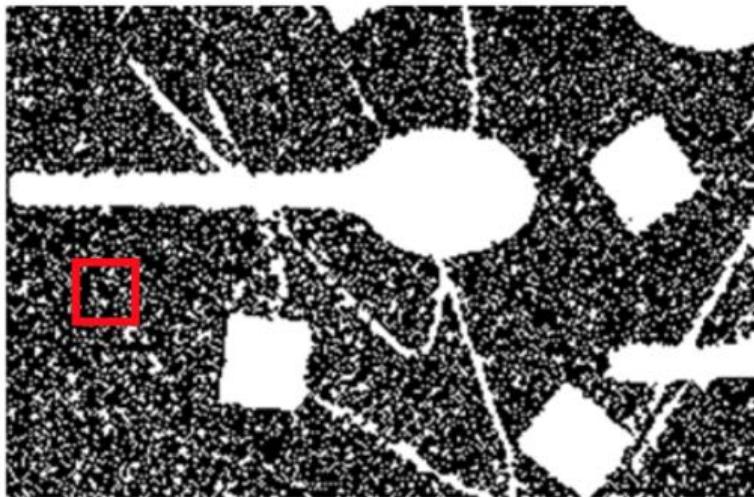
Гауссов шум



Шум “Соль и перец”

Шум в бинарных изображениях

- По одному пикслю невозможно определить – шум или объект?
- Нужно рассматривать окрестность пикселя!



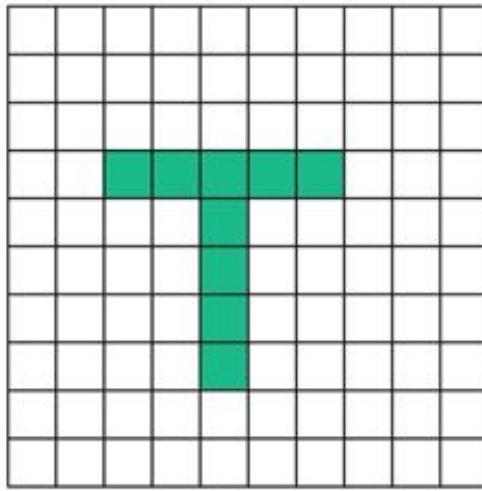
Подавление и устранение шума

Широко известный способ - устранение шума с помощью операций математической морфологии:

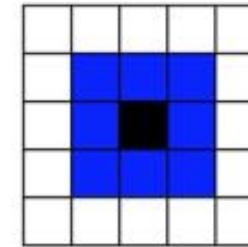
- Сужение (erosion)
- Расширение (dilation)
- Закрытие (closing)
- Раскрытие (opening)

Математическая морфология

A



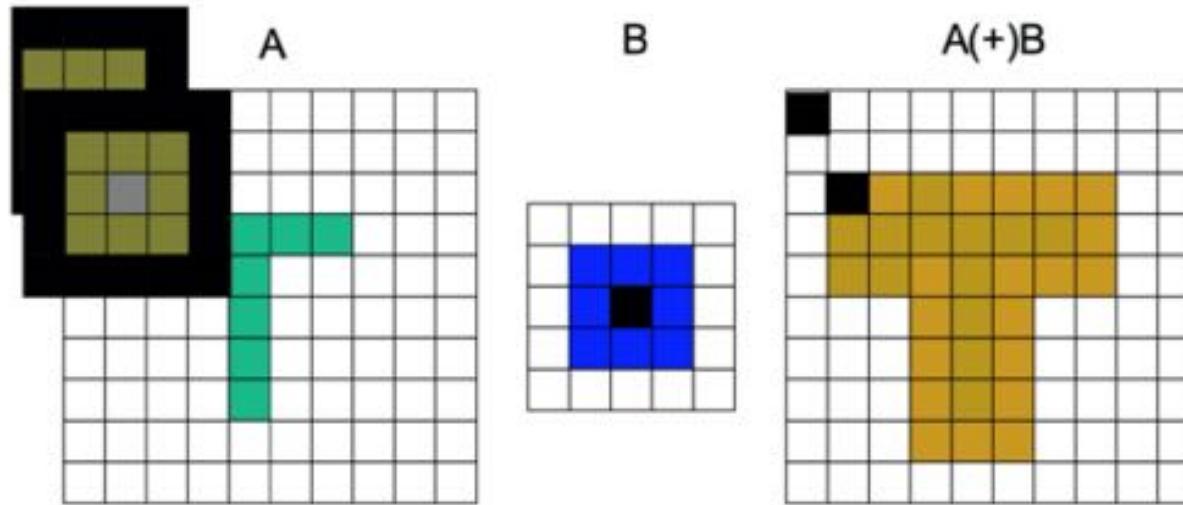
B



Множество А обычно является объектом обработки

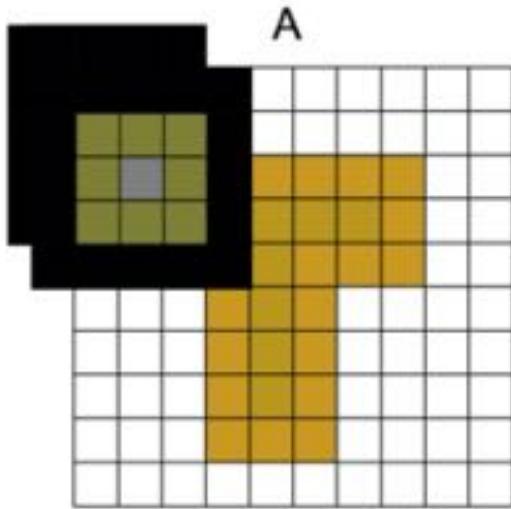
Множество В (называемое структурным элементом) – инструмент обработки

Расширение

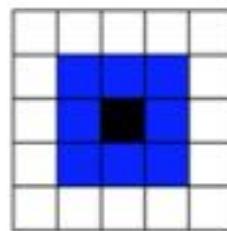


Операция «расширение» - аналог логического «ИЛИ»

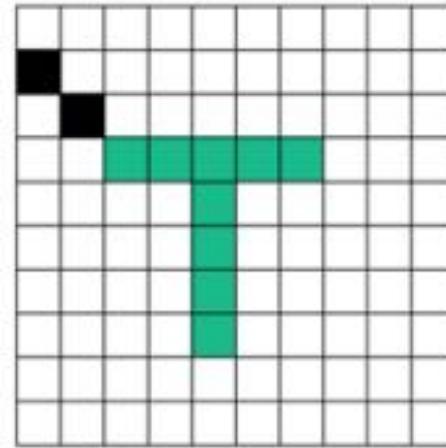
Сужение



B

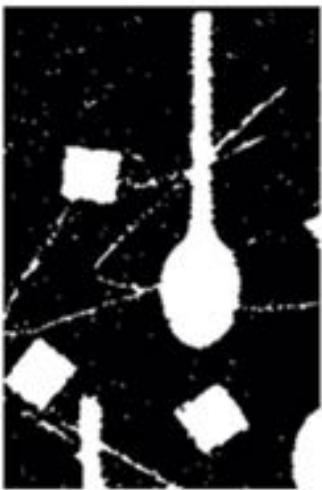


A(-)B



Операция «расширение» - аналог логического «И»

Результат операции сужения



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & [1] & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & [1] & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & [1] & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Замечание

Результат морфологических операций во многом определяется применяемым структурным элементом. Выбирая различный структурный элемент можно решать разные задачи обработки изображений:

- Шумоподавление
- Выделение границ объекта
- Выделение особенностей на медицинских снимках

Операции раскрытия и закрытия

Морфологическое раскрытие (opening) - вначале сужение, потом расширение

- $\text{open}(A,B)=(A \ (-) \ B) \ (+) \ B$

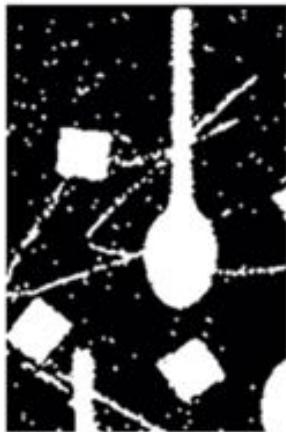
Морфологическое закрытие (closing) - вначале расширение, потом сужение

- $\text{close}(A,B)=(A \ (+) \ B) \ (-) \ B$

Попробуйте догадаться, что эти операции делают?

Применение открытия

Применим операцию открытия к изображению с сильным шумом:



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Сужение vs Открытие



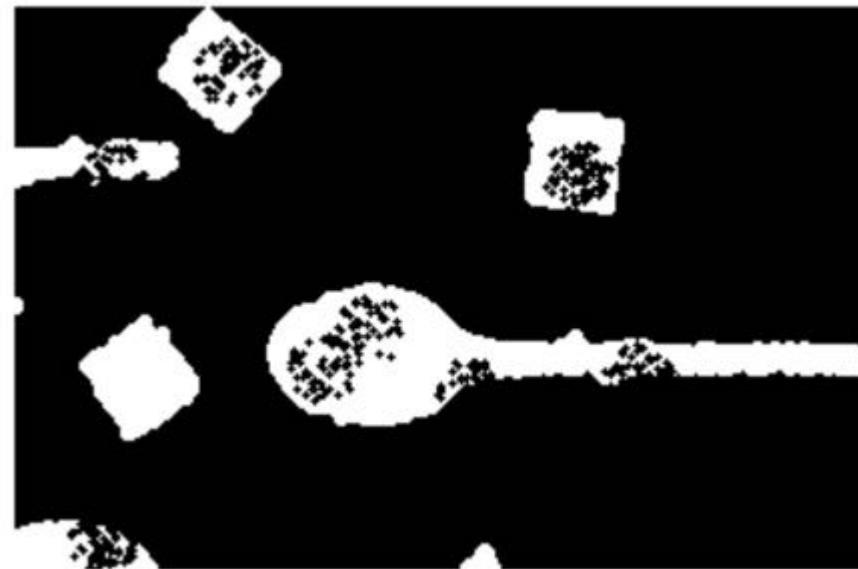
Сужение



Открытие

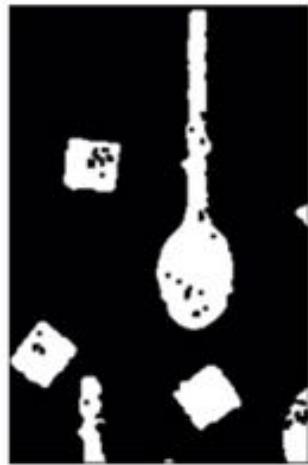
Дефекты бинаризации

Пример бинарного изображению с дефектами распознаваемых объектов



Применение закрытия

Применим операцию закрытия к изображению с дефектами объектов:



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$



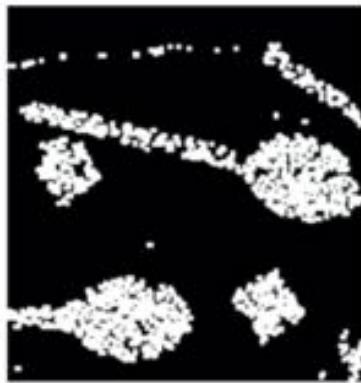
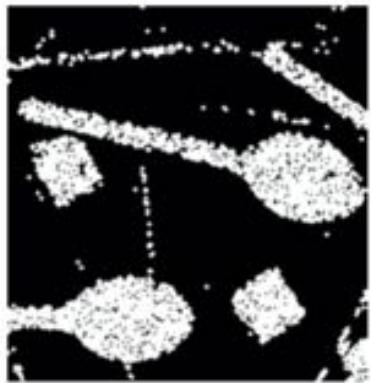
$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Не лучший пример для морфологии

Не во всех случаях математическая морфология так легко убирает дефекты, как хотелось бы...



Применение операции открытия



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Часто помогает медианная фильтрация!

Что дальше?



Получили бинарное изображение



Нужна карта разметки

Выделение связных областей

Определение связной области:

- Множество пикселей, у каждого пикселя которого есть хотя бы один сосед, принадлежащий данному множеству.
- (или) Любые два пикселя которого связаны путём, проходящим только через пиксели множества



Соседи пикселей:

	1	
2	*	3
	4	

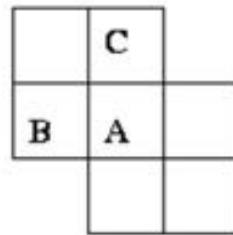
4-связность

1	2	3
4	*	5
6	7	8

8-связность

Последовательное сканирование

Последовательно, сканируем бинарное изображение сверху вниз, слева направо:



```
if A = 0
    do nothing

else if (not B labeled) and (not C labeled)
    increment label numbering and label A

else if B xor C labeled
    copy label to A

else if B and C labeled
    if B label = C label
        copy label to A
    else
        copy either B label or C label to A
        record equivalence of labels
```

За сколько операций мы разметим изображение?

Выделенные связанные компоненты



Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- **Последовательное сканирование**
- Метод K-средних
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Последовательное сканирование



- На основе последовательного сканирования можно сделать и метод сегментации изображений на однородные области
- Для простоты будем оценивать только яркость пикселов

Последовательное сканирование

Сканируем изображение сверху вниз, слева направо:

1. if $|l(A) - l_{avg}(Cl(B))| > \delta$ and $|l(A) - l_{avg}(Cl(C))| > \delta$ -
создаем новую область, присоединяем к ней пиксель A

	C	
B	A	

2. if $|l(A) - l_{avg}(Cl(B))| < \delta$ xor $|l(A) - l_{avg}(Cl(C))| < \delta$ –
добавить A к одной из областей
3. if $|l(A) - l_{avg}(Cl(B))| < \delta$ and $|l(A) - l_{avg}(Cl(C))| < \delta$:
 1. $|l_{avg}(Cl(B)) - l_{avg}(Cl(C))| < \delta$ –
сливаем области B и C.
 2. $|l_{avg}(Cl(B)) - l_{avg}(Cl(C))| > \delta$ –
добавляем пиксель A к тому классу, отклонение от
которого минимально.

$l(A)$ – яркость пикселя A

$Cl(B)$ – область к которой принадлежит пиксель B

$l_{avg}(Cl(B))$ – средняя яркость области к которой принадлежит B

Последовательное сканирование. Пример



Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- Последовательное сканирование
- **Метод K-средних**
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Image segmentation. Пример

- Интенсивности пикселей образуют 3 группы
- Мы можем отнести каждый пиксель к одной из этих групп в зависимости от интенсивности пикселя

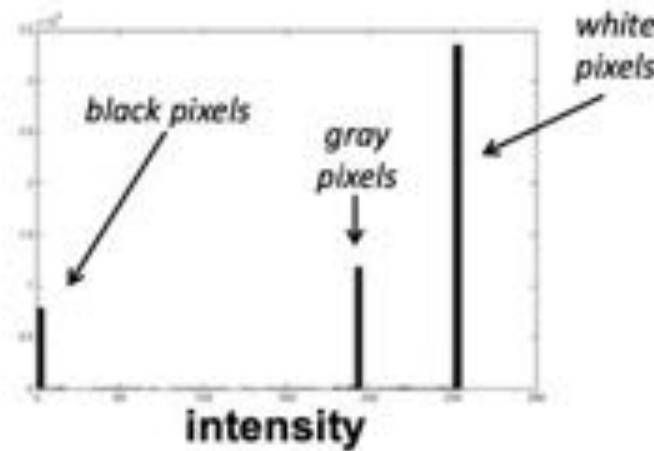
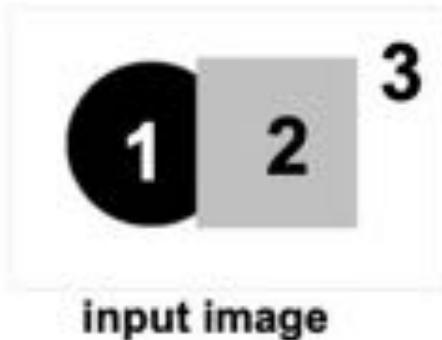


Image segmentation. Пример

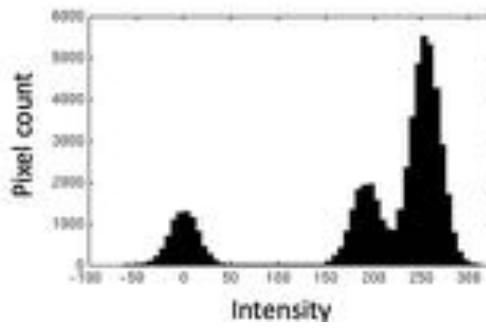
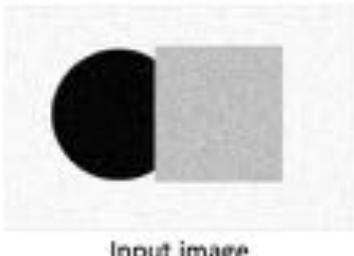
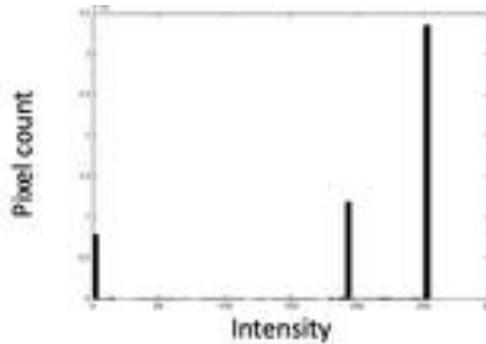
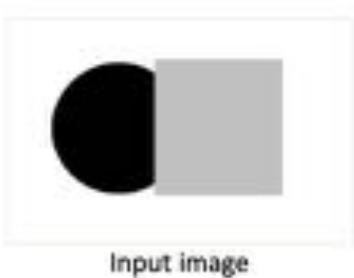


Image segmentation. Пример

- Как теперь определить к какой из групп относится каждый пиксель?
- Нужно кластеризовать!

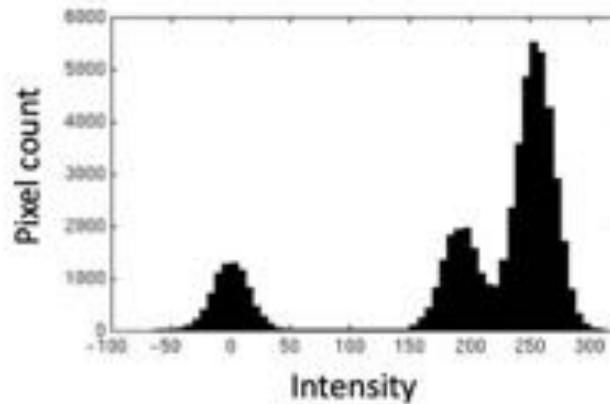
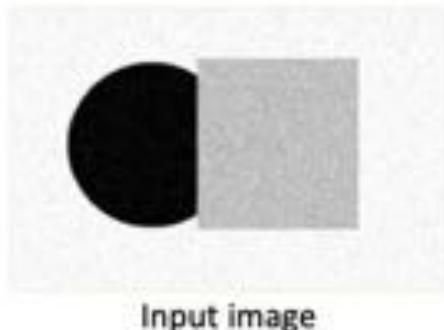
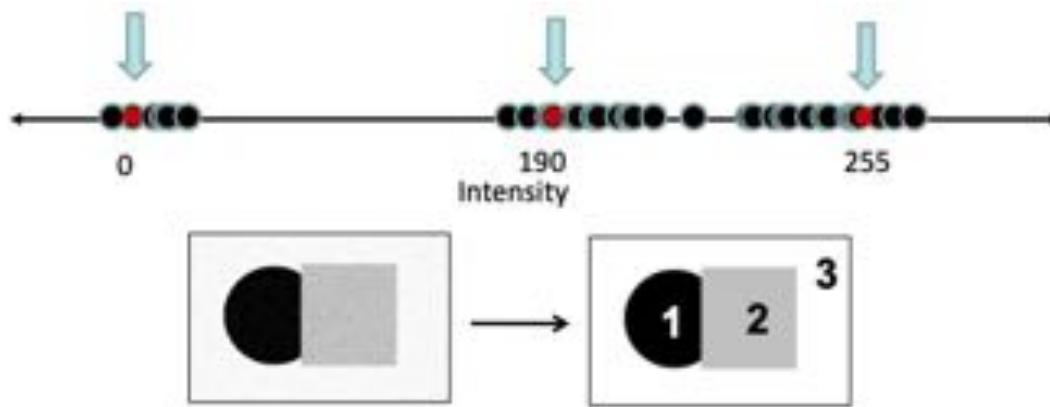


Image segmentation. Пример



Задача: выделить 3 центра кластеров, для каждого из пикселя, проверить к какому из центров он находится ближе всего

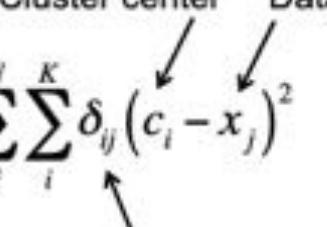
Лучшие центры кластеров - это те, которые минимизируют SSD (sum of squared distance)

$$SSD = \sum_{clusteri} \sum_{x \in clusteri} (x - c_i)^2$$

Минимизируем “разброс”

Задача: минимизировать дисперсию в получаемых кластеров

$$c^*, \delta^* = \arg \min_{c, \delta} \frac{1}{N} \sum_j^K \sum_i^K \delta_{ij} (c_i - x_j)^2$$

Cluster center Data

Whether x_j is assigned to c_i

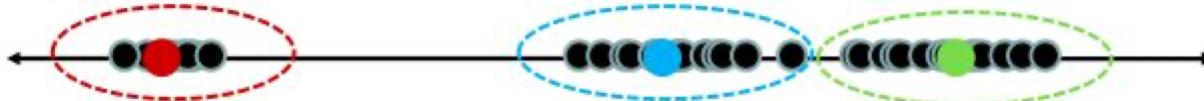
Кластеризация

“что было раньше курица или яйцо?”

- Если бы мы знали центры кластеров, мы могли бы легко разбить пиксели на группы



- Если бы мы знали, как пиксели разбиты на группы, мы могли бы легко посчитать центры кластеров



K-means

1. Инициализация ($t=0$): центры кластеров c_1, \dots, c_K
2. Группируем пиксели в соответствии с текущими центрами кластеров

$$\delta^t = \operatorname{argmin}_{\delta} \frac{1}{N} \sum_j^K \sum_i^K \delta_{ij}^{t-1} (c_i^{t-1} - x_j)^2$$

3. Пересчитываем центры кластеров

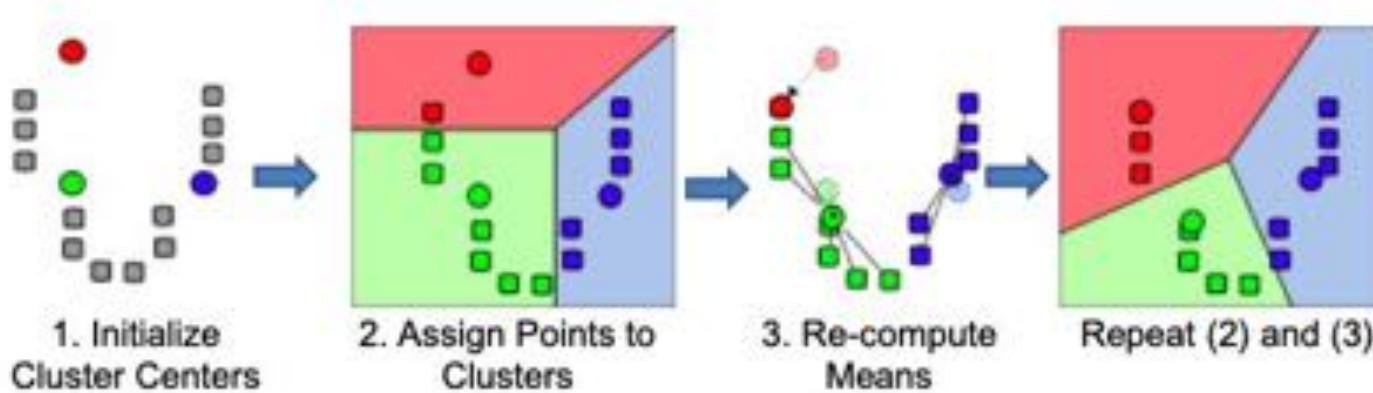
$$c^t = \operatorname{argmin}_c \frac{1}{N} \sum_j^K \sum_i^K \delta_{ij}^t (c_i^t - x_j)^2$$

4. Возвращаемся к шагу 1

K-means

1. Инициализация ($t=0$): центры кластеров c_1, \dots, c_K
 - a. Обычно: рандомная инициализация
 - b. Или жадным алгоритмом
2. Группируем пиксели в соответствии с текущими центрами кластеров
 - a. Метрика: евклидова, косинусная
$$\delta^t = \operatorname{argmin}_{\delta} \frac{1}{N} \sum_j^K \sum_i^K \delta_{ij}^{t-1} (c_i^{t-1} - x_j)^2$$
3. Пересчитываем центры кластеров
$$c^t = \operatorname{argmin}_c \frac{1}{N} \sum_j^K \sum_i^K \delta_{ij}^t (c_i^{t-1} - x_j)^2$$
4. Возвращаемся к шагу 1
 - a. Пока центры не перестают меняться

K-means

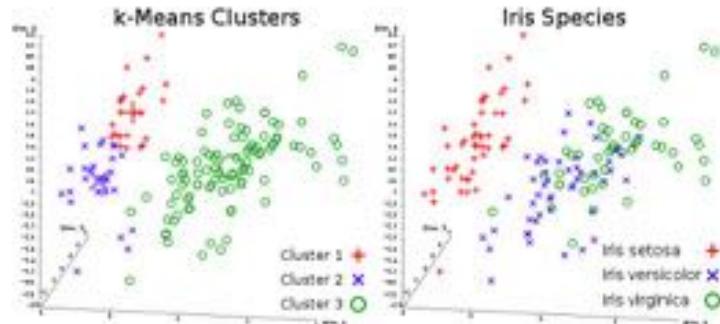


K-means

- Сходится к локальному минимуму
 - Нужно запускать несколько раз



- Релевантен только для “шарообразных данных”
- Требует выбора K (количество кластеров)
- Вычислительно затратен при большом количестве данных



K-means



Original image



2 clusters



3 clusters

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Простейший случай: ч/б картинка группировка по интенсивности

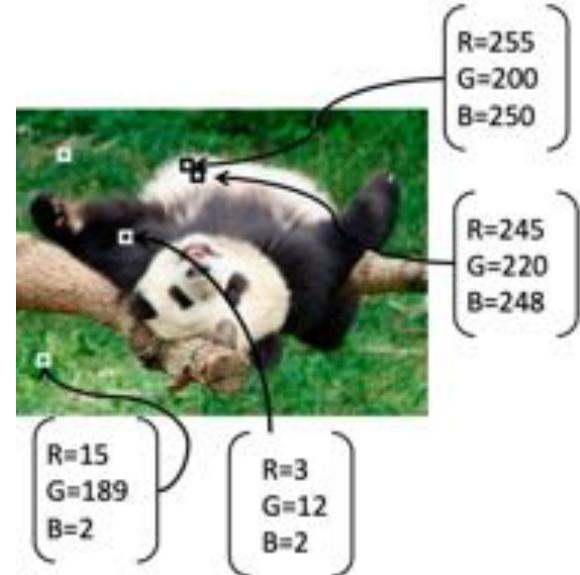
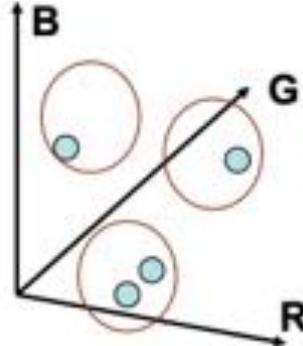


Признаки: интенсивность (1D)

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Группировка пикселя по схожести цвета (RGB)

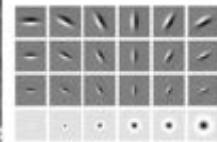
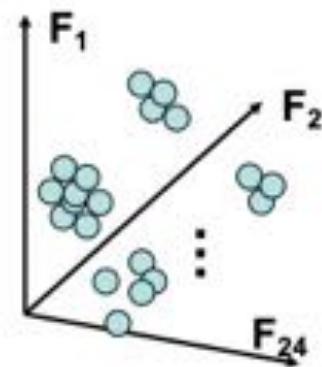


Признаки: значение цвета (3D)

Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Группировка пикселя по схожести текстур

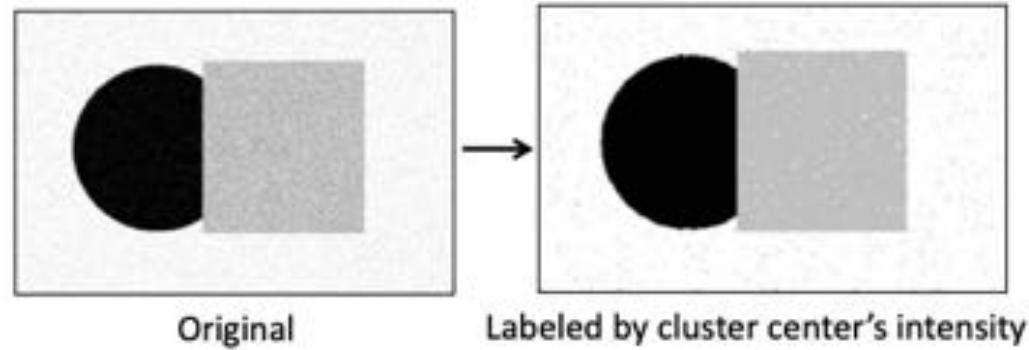


Filter bank of
24 filters

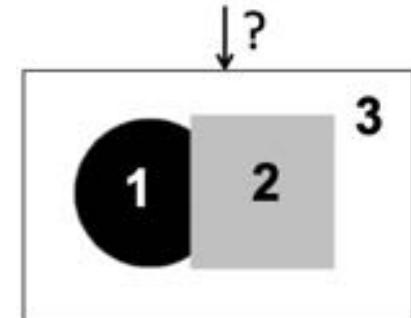
Признаки: отклики на различные фильтры (24D)

Пространство признаков

Попиксельная кластеризация почти наверняка приведет к большому количеству выбросов



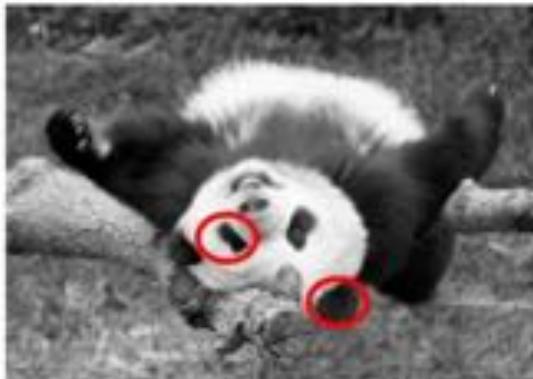
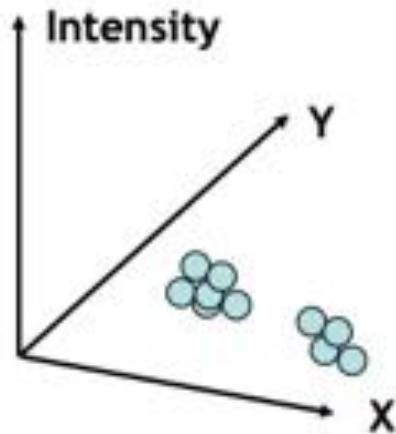
Как учитывать пространственную информацию при кластеризации?



Пространство признаков

В зависимости от того, что мы выберем как признаки, можно по-разному кластеризовать пиксели

Группировка пикселя по интенсивности+позиции



K-means

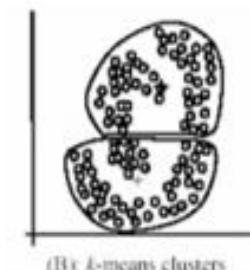
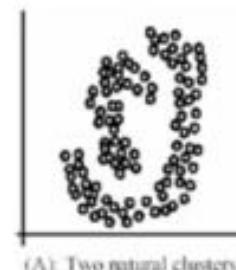
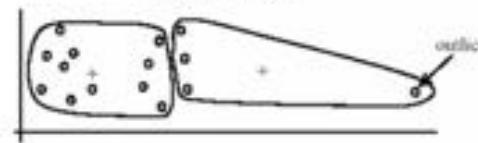
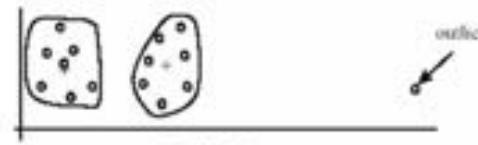
К-means на основе цвета или интенсивности - это своего рода квантование (сжатие) изображения

Такие кластеры не будут пространственно согласованными



K-means: pros and cons

- Pros:
 - Находит центры кластеров, минимизирующих эмпирическую дисперсию (хорошо описывает закономерности в данных)
 - Быстро и просто, легко заимплементить
- Cons:
 - Необходимо выбирать количество K
 - Чувствителен к выбросам
 - Сходится к локальному минимуму
 - Не универсальная метрика расстояния
 - Может быть медленным: каждая итерация стоит $O(KNd)$ для N точек размерности d
- Usage:
 - Кластеризация без учителя
 - Сегментация пикселей

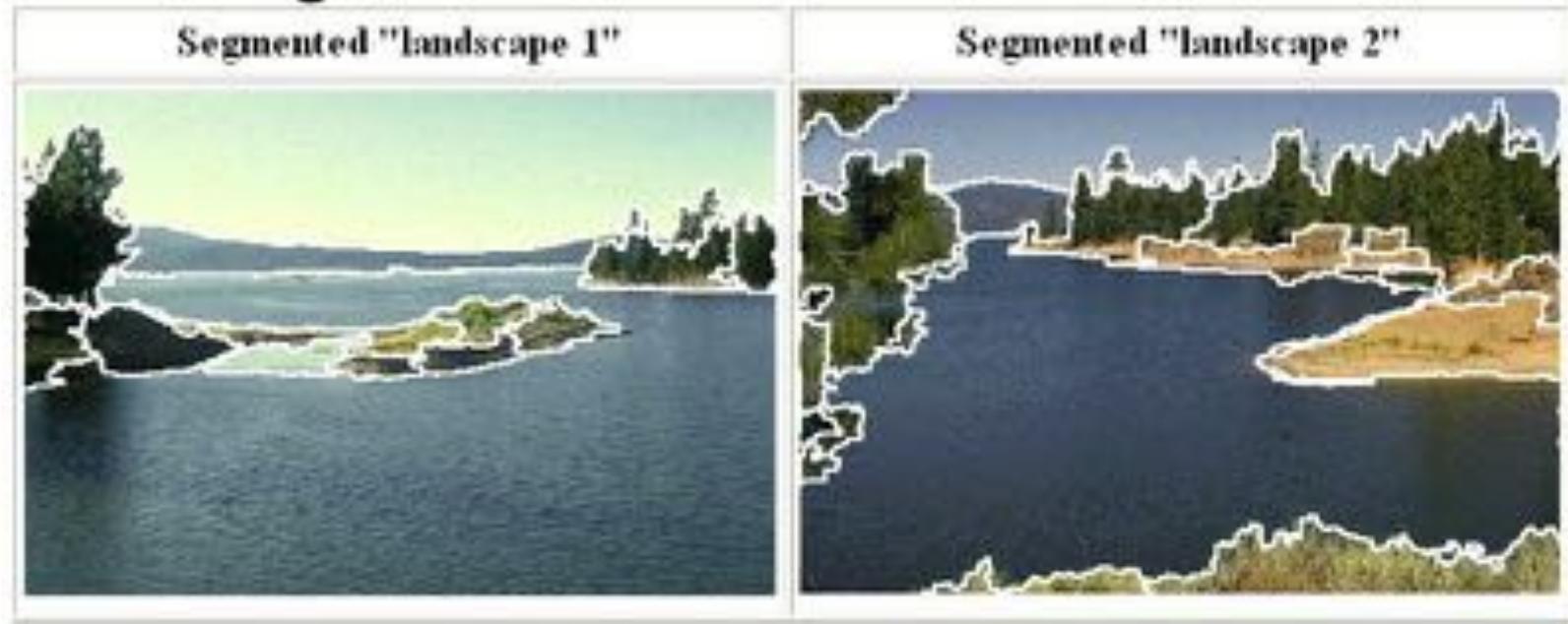


Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Mean-shift clustering

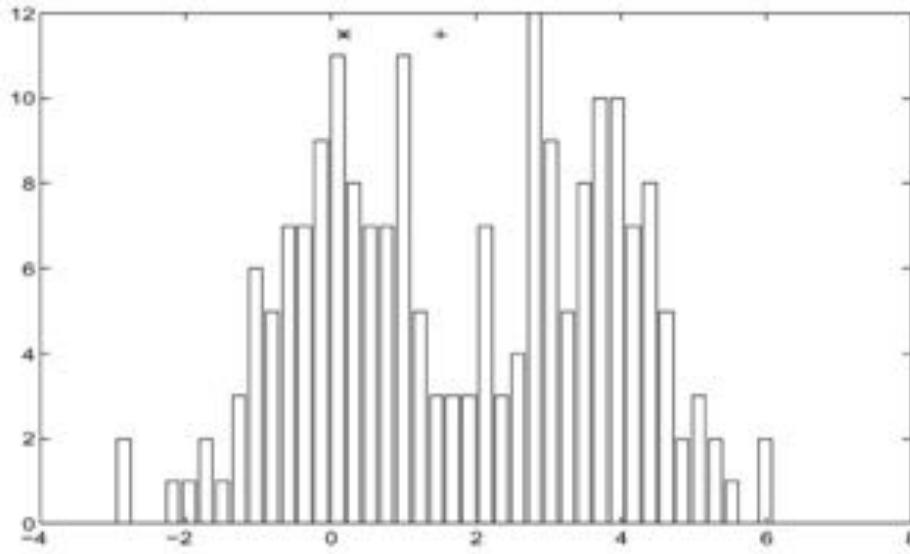
Гибкий алгоритм для cluster-based сегментации



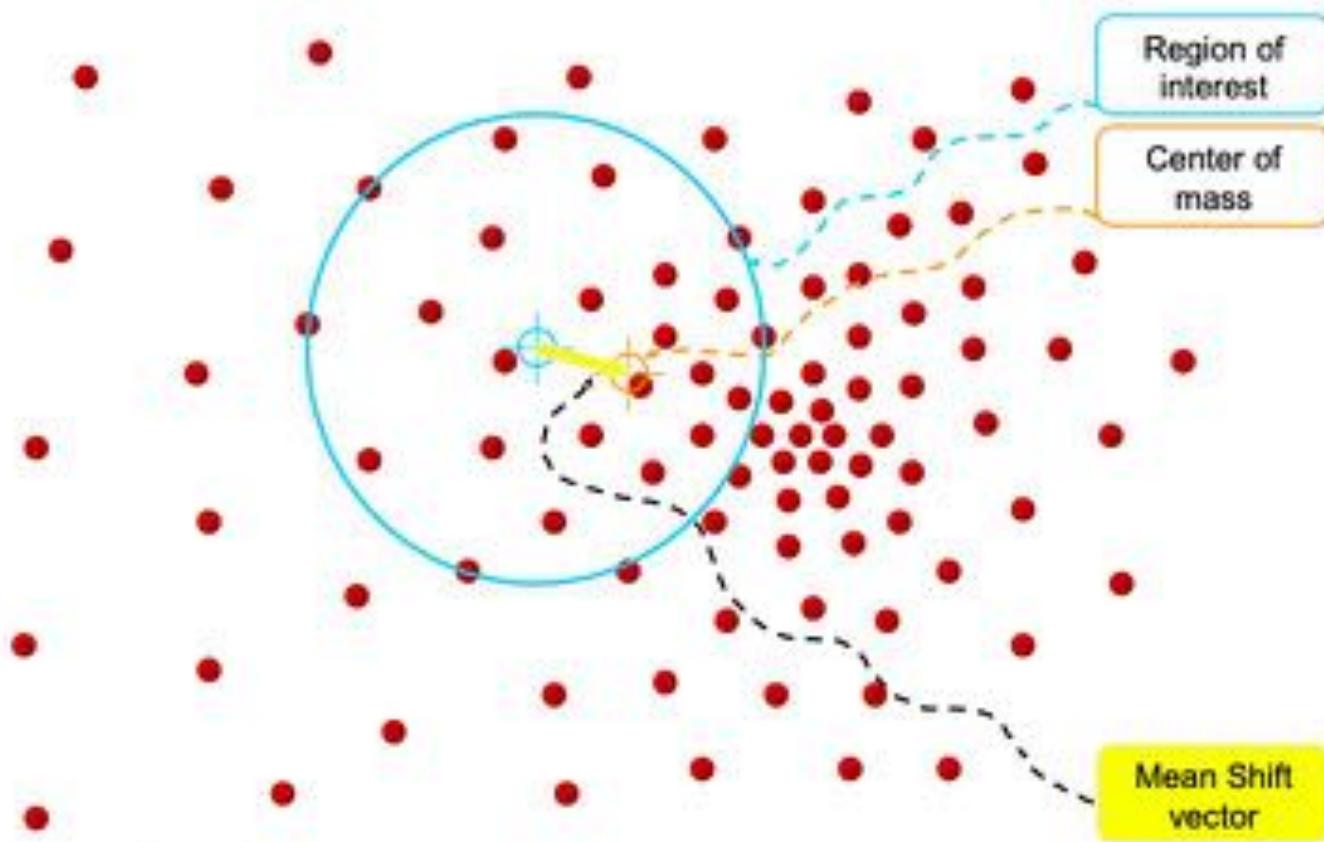
Mean-shift clustering

Итеративный алгоритм:

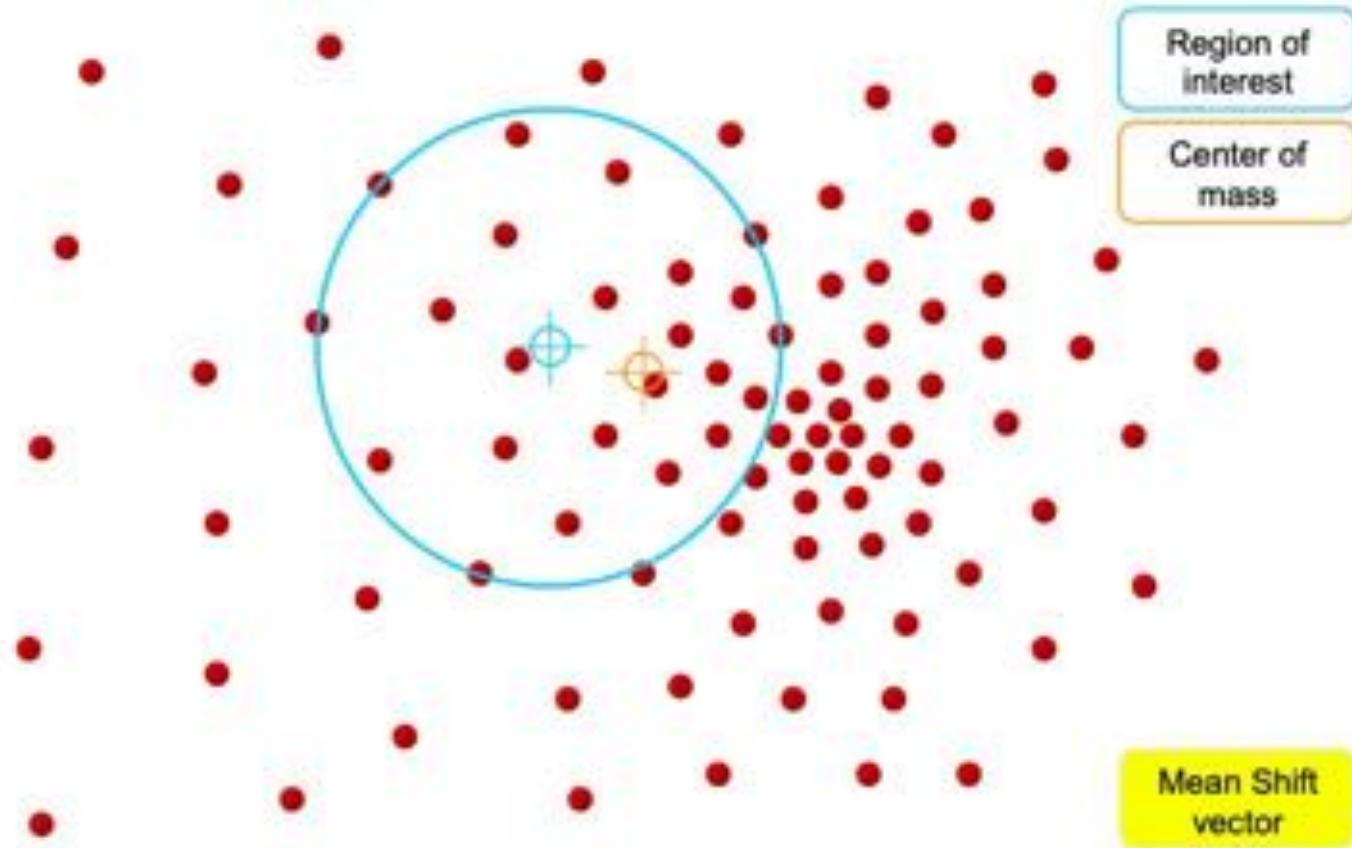
1. Инициализируем несколько точек и окно W
2. Подсчитываем центроиду окна (mean of window) $\sum_{x \in W} xH(x)$
3. Сместить окно к центроиде
4. Вернуться к шагу 2, пока не сойдется



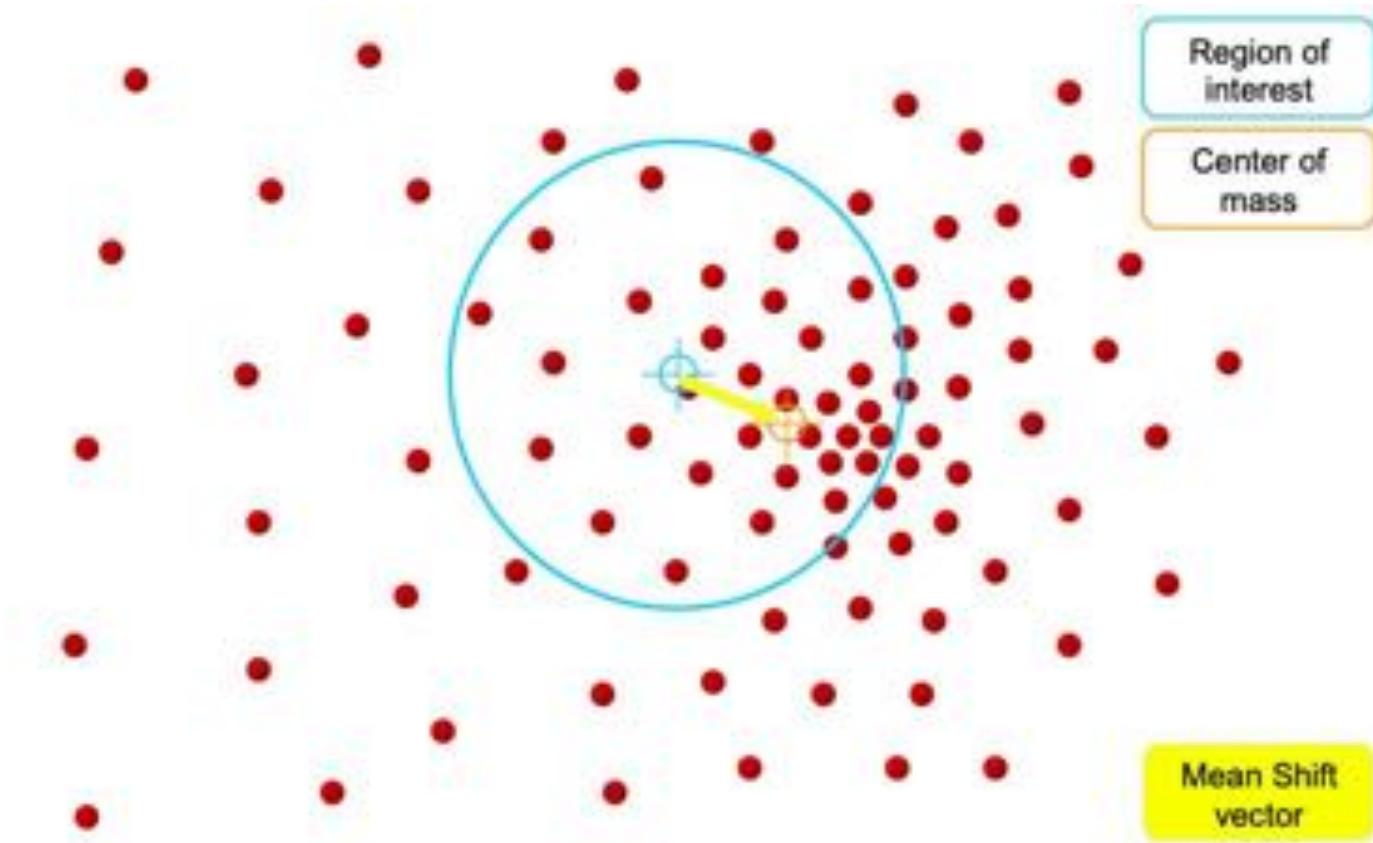
Mean-shift clustering



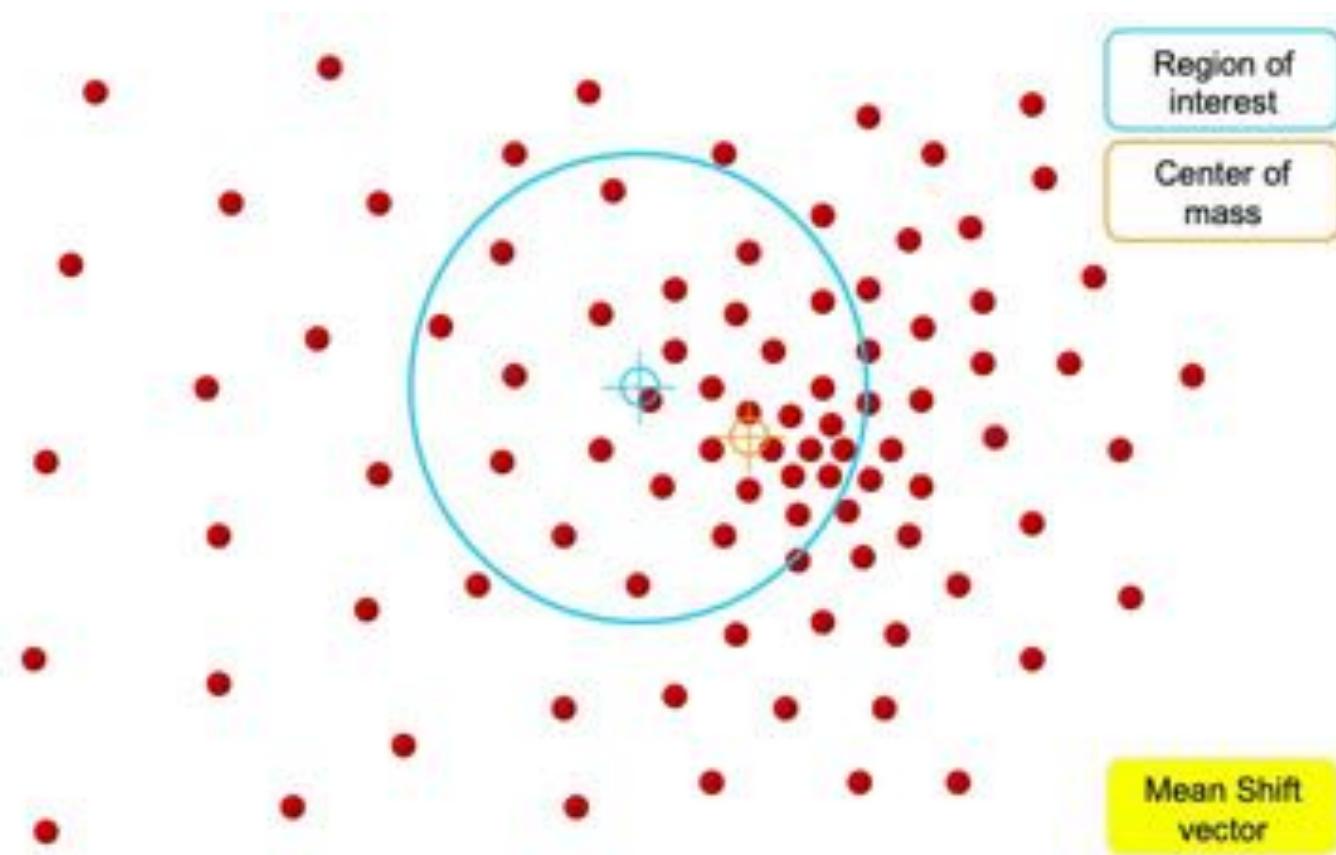
Mean-shift clustering



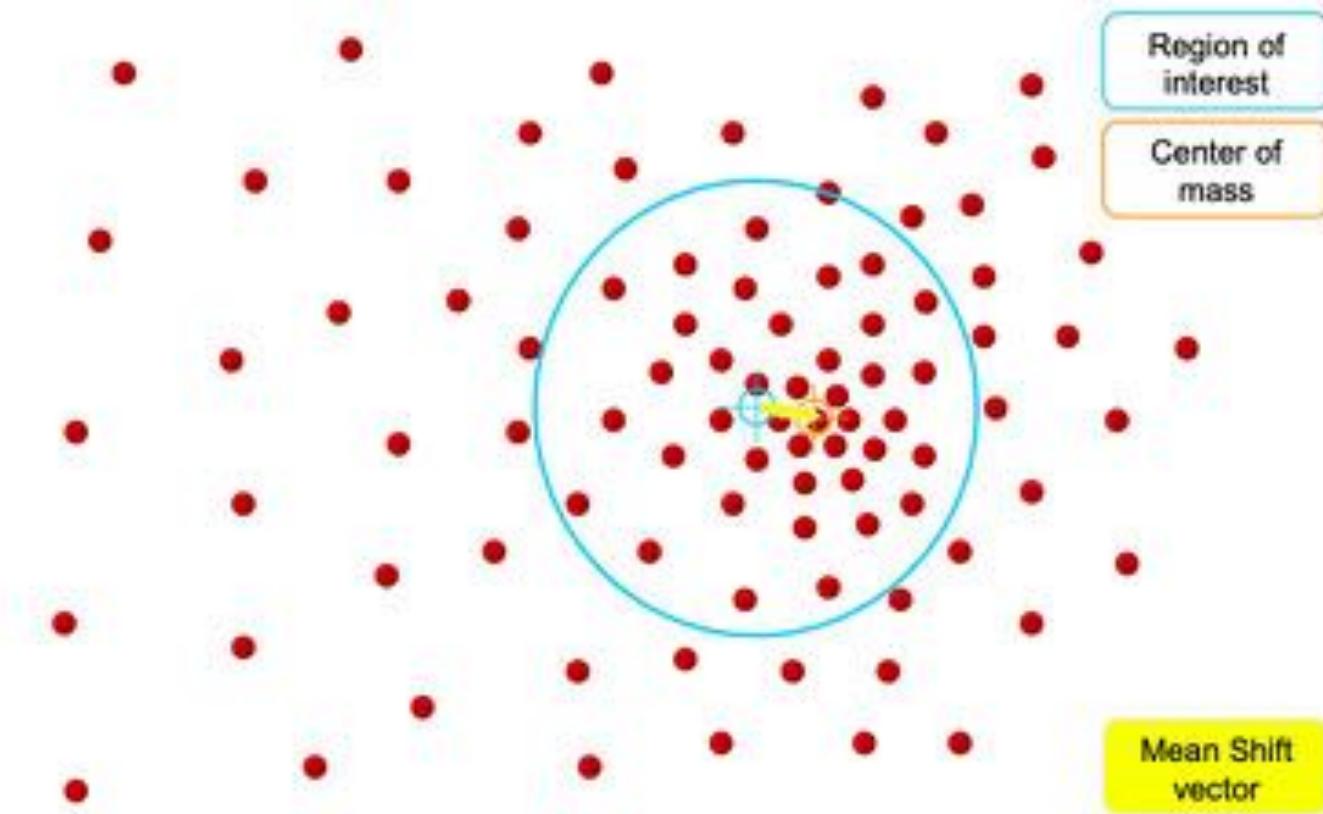
Mean-shift clustering



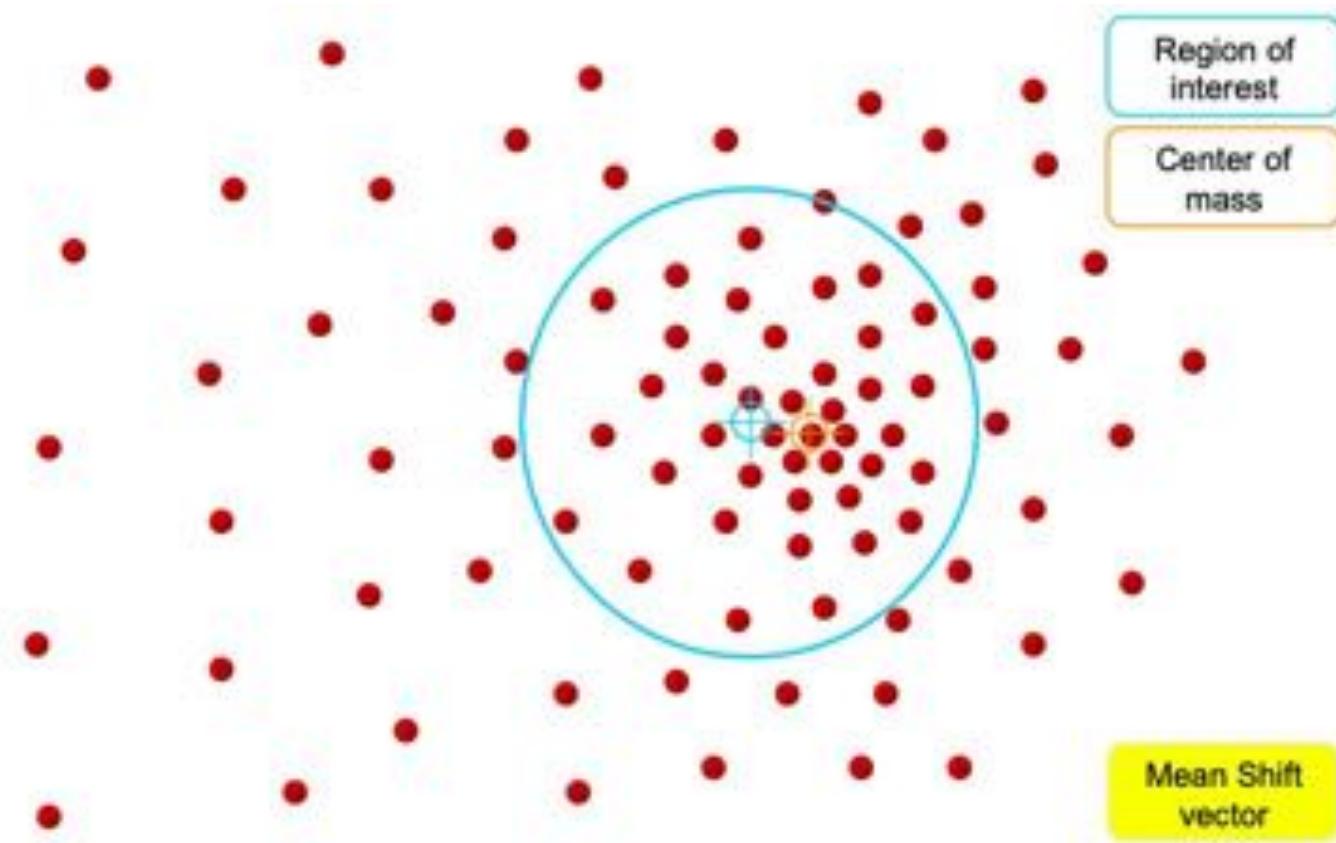
Mean-shift clustering



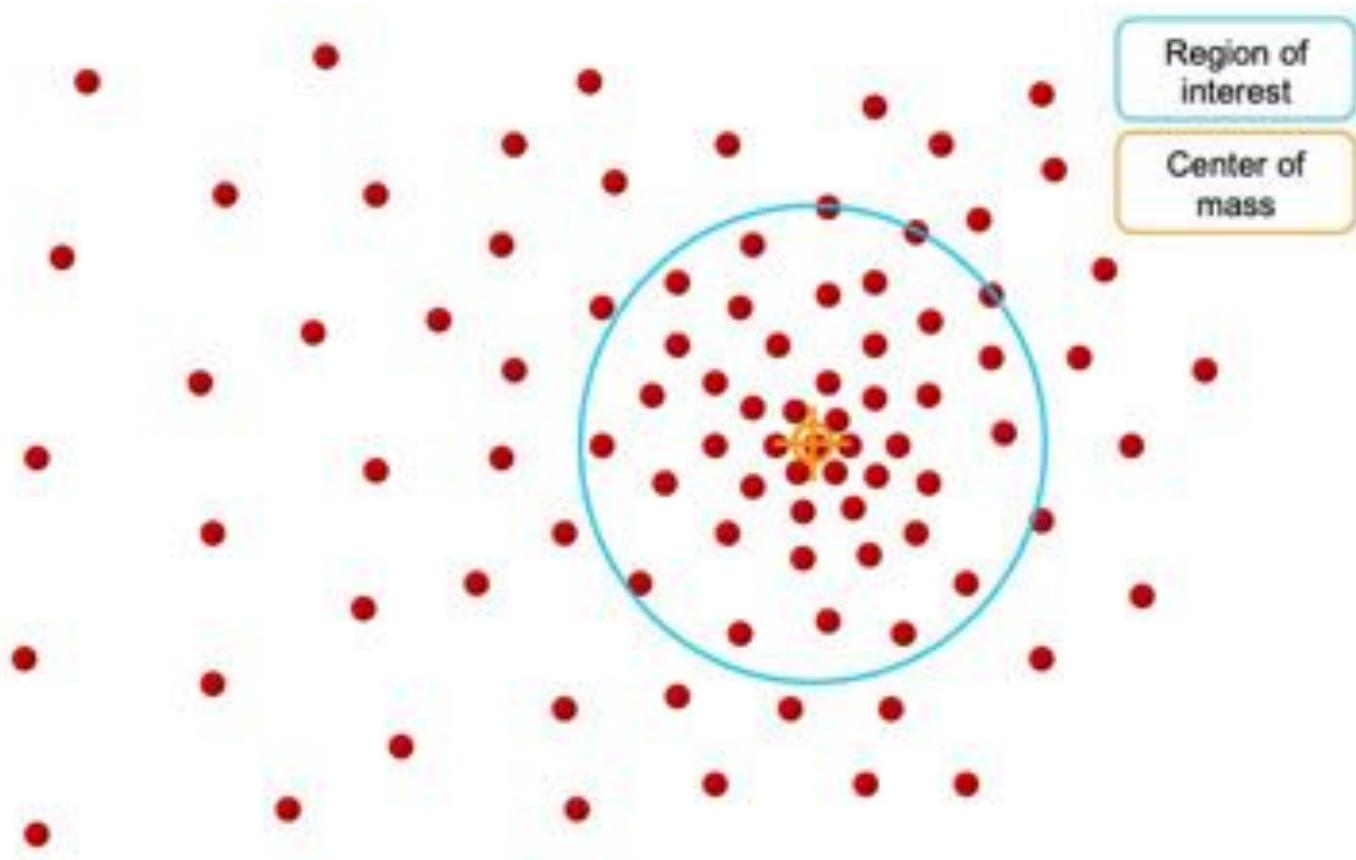
Mean-shift clustering



Mean-shift clustering

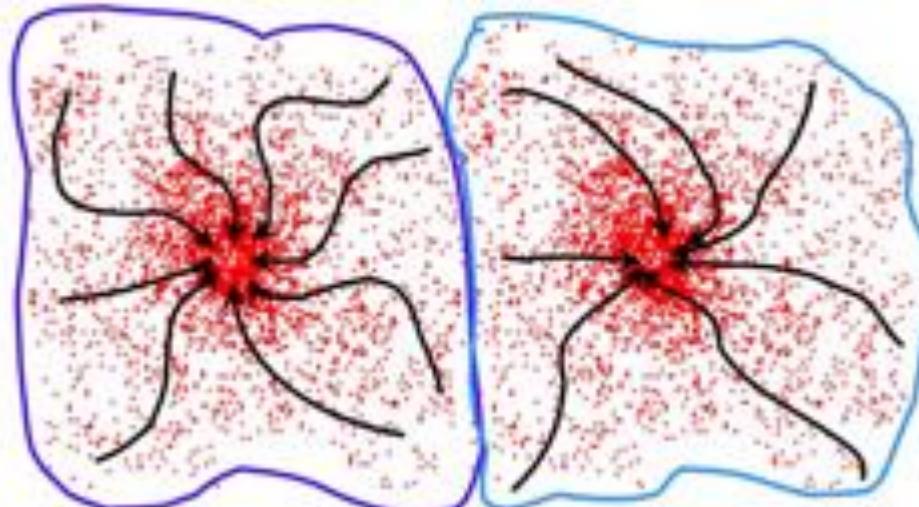


Mean-shift clustering

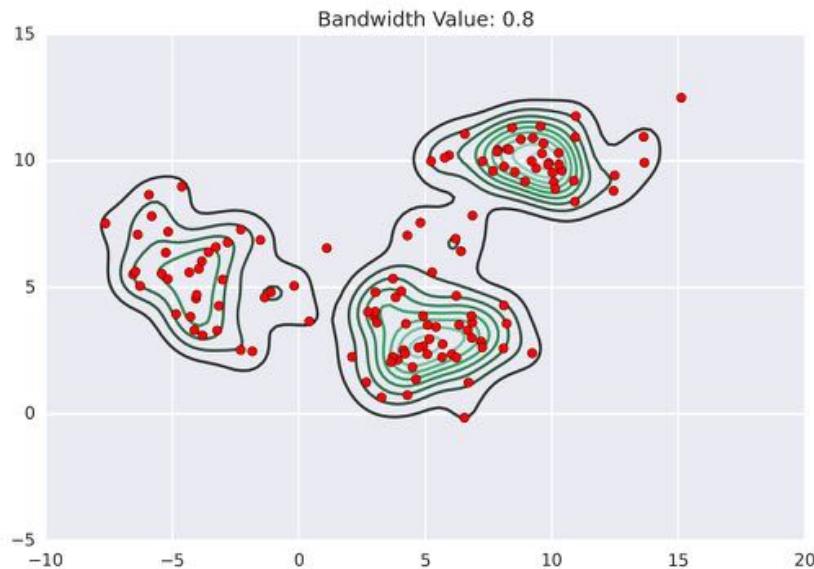
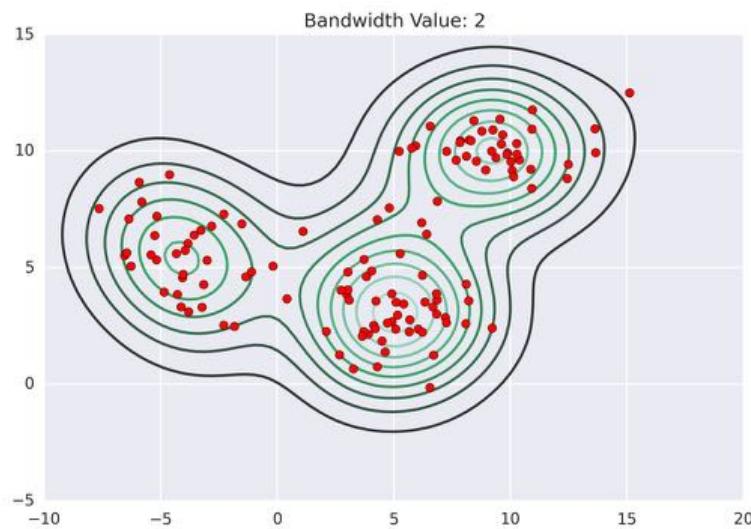


Mean-shift clustering

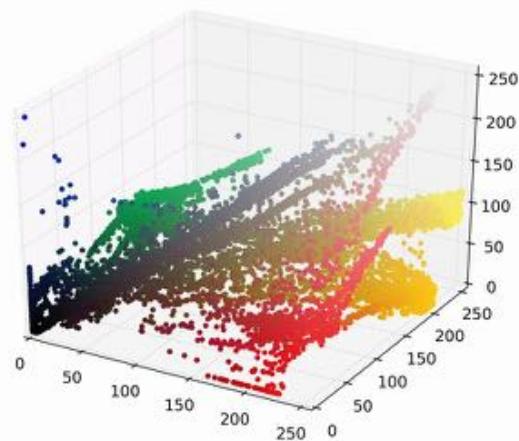
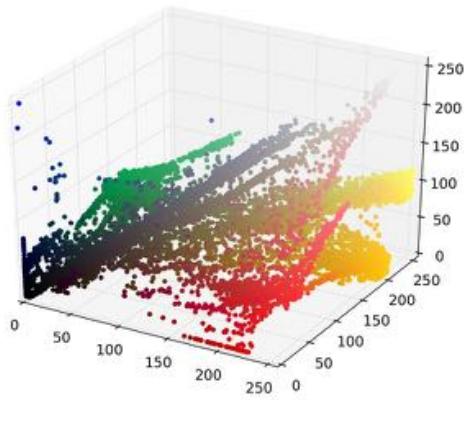
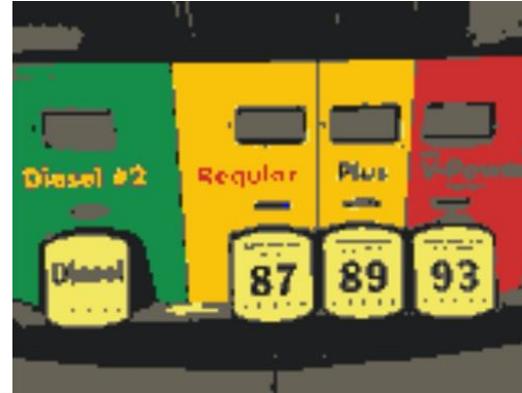
- Кластер: место куда “стекаются” окна
- Множество притяжения: область, в которой все траектории окон ведут в одну точку



Mean-shift algorithm



Mean-shift algorithm



Примеры работы



Примеры работы



Примеры работы



Резюме: mean-shift алгоритм

- Pros
 - Гибкий алгоритм, подходит для многих задач
 - Подходит для данных любой “формы” (сферические, эллиптические, ...)
 - Всего один параметр (размер окна h)
 - Понятная статистическая интерпретация
 - Устойчиво к выбросам
- Cons
 - Результат сильно зависит от выбора параметра
 - h трудно выбрать
 - Вычислительно затратно
 - Плохо масштабируется с увеличением количества признаков

Алгоритмы к рассмотрению

- Бинаризация, морфология, и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Mean-Shift
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Водораздел (Watershed)

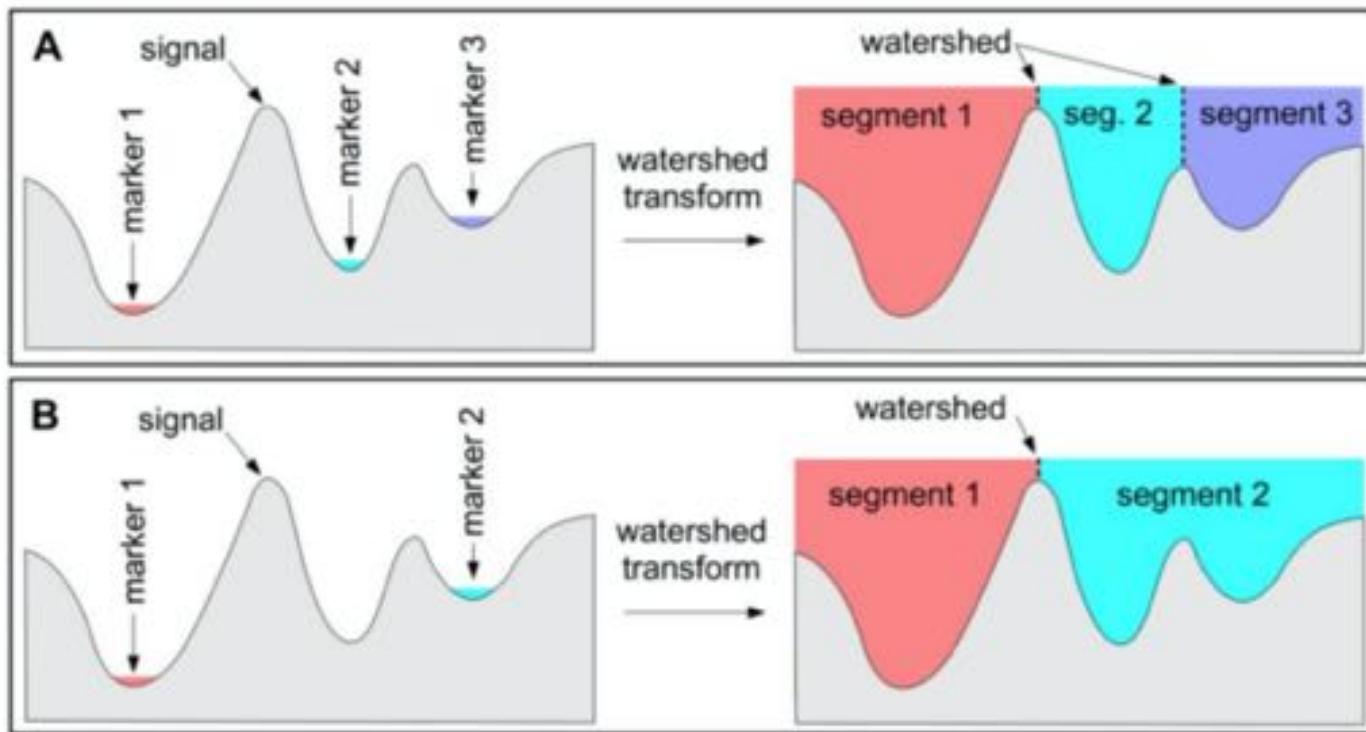
- изображение представляется в виде рельефа
- точки с высокой интенсивностью - возвышенности
- точки с низкой интенсивностью - низины
- помимо интенсивности, используют фактор расстояния пикселя до грани

Водораздел (Watershed)

Алгоритм:

- начинаем заполнять низины
- там где сталкиваются области из разных бассейнов проходит граница объекта
- заполняем до тех пор, пока все вершин не скроются под водой

Водораздел (Watershed)

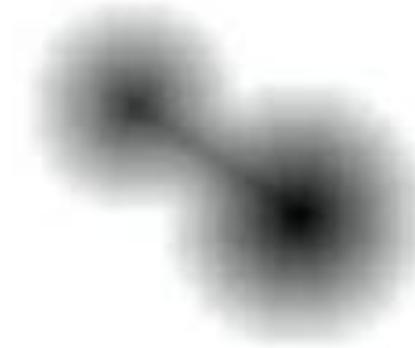


Водораздел (Watershed)

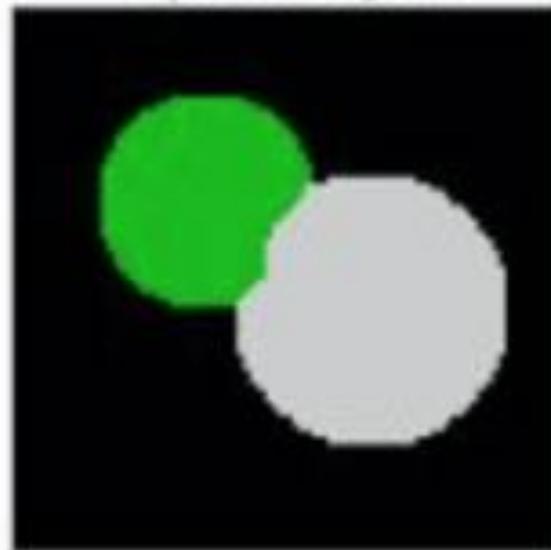
Overlapping objects



Distances



Separated objects



Водораздел (Watershed)

- из-за шума и не идеальной структуры сегменты могут объединяться в один
- необходимо указать начальное приближение (маркеры) для сегментации

Алгоритмы к рассмотрению

- Бинаризация и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

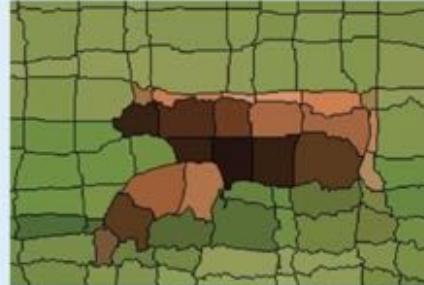
Супер-пиксель

- объединение нескольких соседних пикселей на изображении в единую область
- объединение происходит по таким признакам как цвет или текстура
- в результате объединения пикселей получается упрощенное представление изображения, которое можно использовать в задачах распознавания

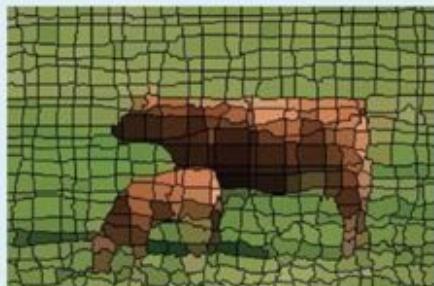
Супер-пиксель



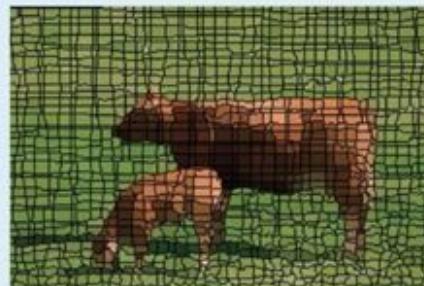
(a)



(b)

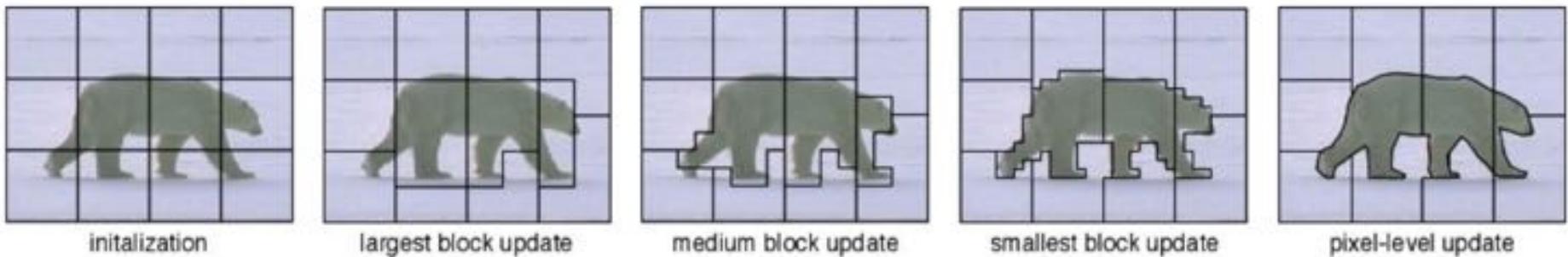


(c)



(d)

Superpixels Extracted via Energy-Driven Sampling



Superpixels Extracted via Energy-Driven Sampling

```
function SEEDS(
    I, // Color image.
    w(1) × h(1), // Initial block size.
    L, // Number of levels.
    Q // Histogram size.
)
    initialize the block hierarchy and the initial superpixel segmentation
    // Initialize histograms for all blocks and superpixels:
    for l = 1 to L
        // At level l = L these are the initial superpixels:
        for each block B(l)i
            initialize histogram hB(l)i
    // Block updates:
    for l = L - 1 to 1
        for each block B(l)i
            let Sj be the superpixel B(l)i belongs to
            if a neighboring block belongs to a different superpixel Sk
                if ∩(hB(l)i, hSk) > ∩(hB(l)i, hSj)
                    Sk := Sk ∪ B(l)i, Sj := Sj - B(l)i
    // Pixel updates:
    for n = 1 to W · H
        let Sj be the superpixel xn belongs to
        if a neighboring pixel belongs to a different superpixel Sk
            if hSk(h(xn)) > hSj(h(xn))
                Sk := Sk ∪ {xn}, Sj := Sj - {xn}
return S
```

Algorithm 1: The basic algorithm of **SEEDS**. In practice, both block updates as well as pixel updates can be iterated more than once.

Алгоритмы к рассмотрению

- Бинаризация и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- **Graph Cut**

Graph Cut

- необходимо задать начальные приближения для пикселей фона и объекта
- по этим приближениям оцениваются распределения цвета фона и картинки
- зная эти распределения, можно получить вероятность принадлежности пикселя фону или картинке

Graph Cut

- строится граф с двумя вершинами: источник (фон) и сток (объект)
- каждый пиксель изображения соединяется с источником и стоком
- вес ребра пропорционален вероятности принадлежности пикселя фону и объекту соответственно

Graph Cut

- пиксели изображения разделяются на части алгоритмом [Minimum Cut](#)
- алгоритм разрезает связный граф на две части таким образом, чтобы сумма весов ребер через которые проходит разрез была минимальна

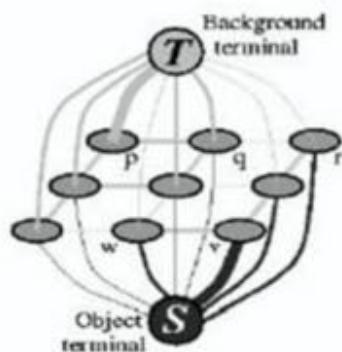
Graph Cut



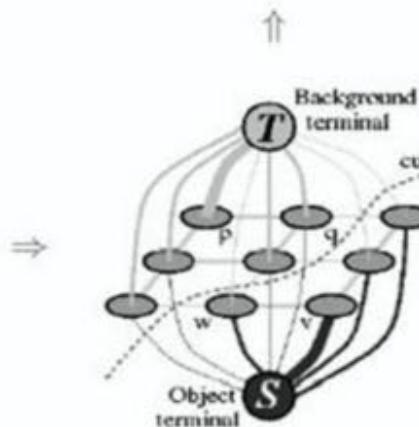
(a) Image with seeds.



(d) Segmentation results.

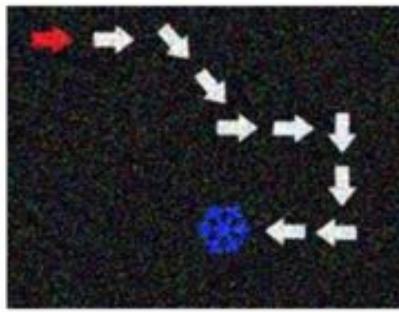


(b) Graph.

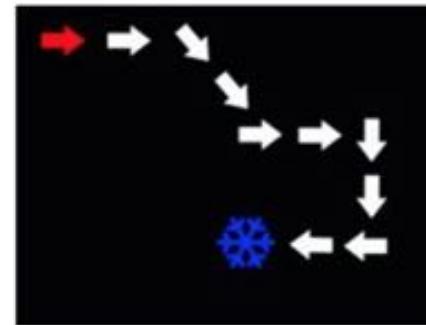


(c) Cut.

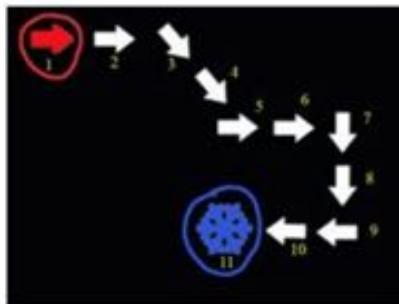
Анализ полученных регионов



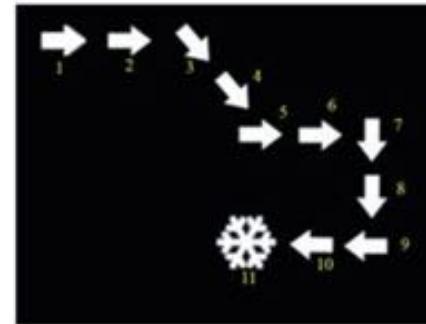
Предобработка
изображения



Сегментация
изображения



Вычисление
признаков
сегментов и
классификация



Детекция объектов

Things vs. Stuff

Объекты (Things)

- Имеют определенный размеры и формы



Найти «человека»

Материалы (Stuff)

- Имеют определенный размеры и формы
- Однородный или повторяющийся шаблон мелких деталей
- Нет определенного размера и формы



Найти «небо»

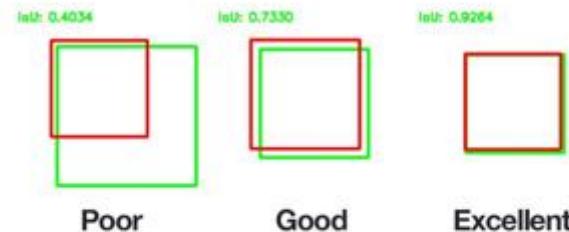
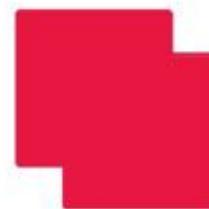
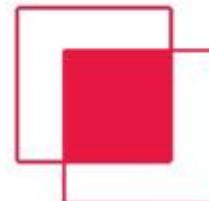
Задачи распознавания

- Категоризация изображений
 - Image categorization
 - Содержит ли изображение самолёт?
- Выделение объектов
 - Object class detection and localization
 - Есть ли на изображении самолёт, и если да, то где и сколько их?
 - Каждый объект выделяем рамкой (bounding box)
- Семантическая сегментация
 - Object class pixel-level segmentation
 - Какие пиксели изображения принадлежат самолёту?

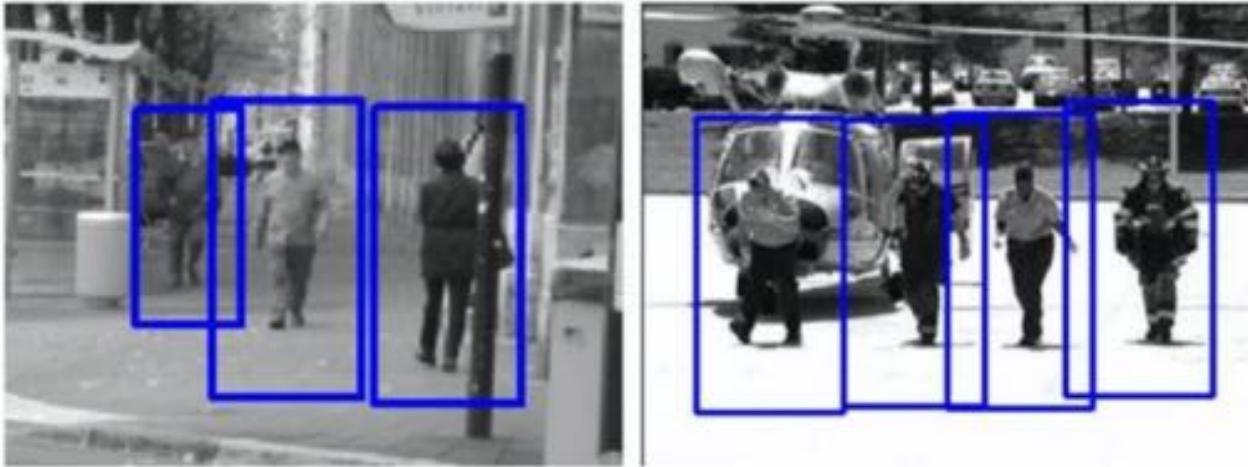


Критерий обнаружения

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Пример - поиск пешеходов



- Нужно найти «пешеходов» (если есть) и выделить их прямоугольной рамкой
- Пешеход (pedestrian) – это «категория», а не «объект»
 - Пешеходы все похожи, но все разные
 - Известным нам способом (сегментация по контрасту + анализ признаков сегмента) выделять пешеходов получается плохо

Скользящее окно (Sliding window)

- Возьмём «окно», пусть размером 64×128 пикселов
- Сканируем изображение «окном»
- Применяем классификатор «пешеход?» к каждому окну
- Скользящее окно – форма сегментации изображения!



Разделили изображение на фрагменты, которые классифицируем независимо друг от друга

Простейший классификатор



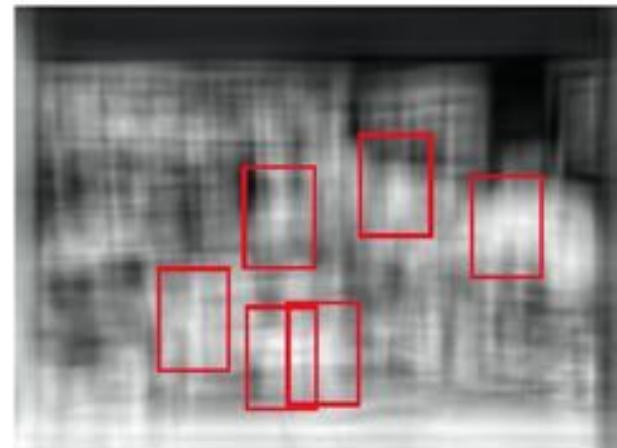
Сопоставление шаблонов («Pattern matching»):

- Фиксируем изображение объекта («шаблон»)
- Классификатор – сравнение шаблона и текущего окна (фрагмента)
- Итогом скользящего окна будет карта откликов (score)

Ограничения подхода



Найдём стул в этом изображении



Мусор

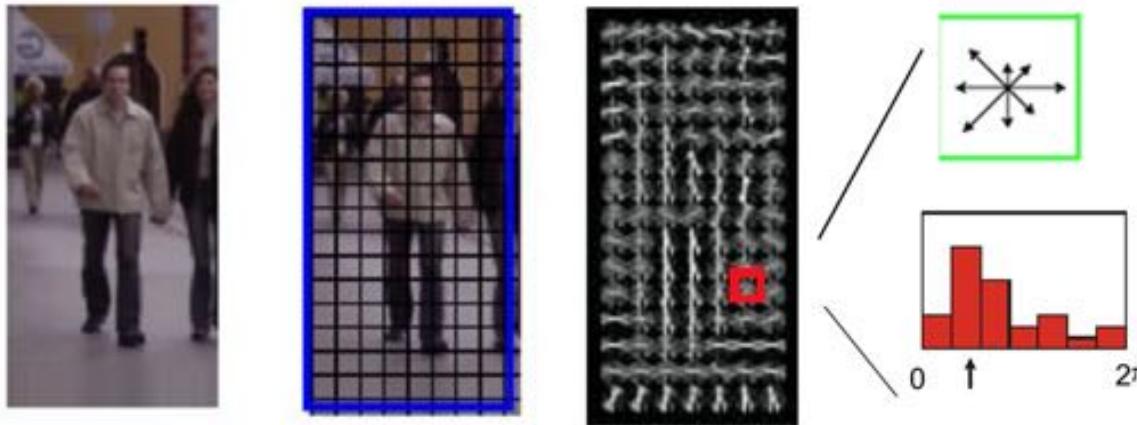
«Классификатор» в виде сравнения с шаблоном не может учесть всё
множество факторов изменчивости

Построение классификатора



- Нужно выбрать признаки и классификатор
- Мы рассмотрим схему HOG + линейный SVM

Вычисление признаков



- Возьмём окно 64×128 пикселей
- Разобьём его на блоки 8×8 пикселей
- Всего будет $8 * 16 = 128$ блоков
- В каждом блоке посчитаем гистограмму ориентаций градиентов с 8 ячейками (8 параметров)
- Всего у нас получится $128 \times 8 = 1024$ признака

Обучение классификатора



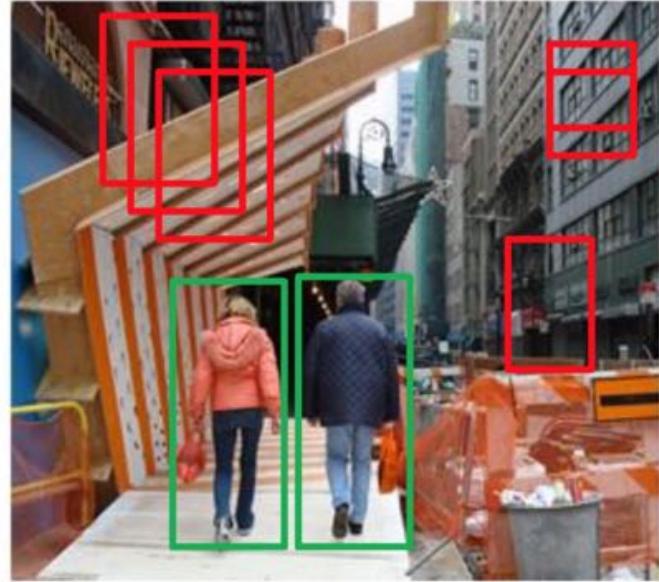
- Соберём обучающую выборку фрагментом изображения с пешеходами и без
- Для каждого фрагмента посчитаем вектор признак x и метку y
- На полученной обучающей выборке обучим линейный классификатор SVM

Алгоритм поиска пешеходов

- Получили работающий «детектор пешеходов»
- Базовый алгоритм:
 - Скользящее окно поиска
 - Вычисление признаков (гистограмм ориентаций градиентов)
 - Классификация с помощью SVM
- Хотя схема HOG + SVM изначально была предложена для пешеходов, она успешно применялась в дальнейшем к разным категориями объектов

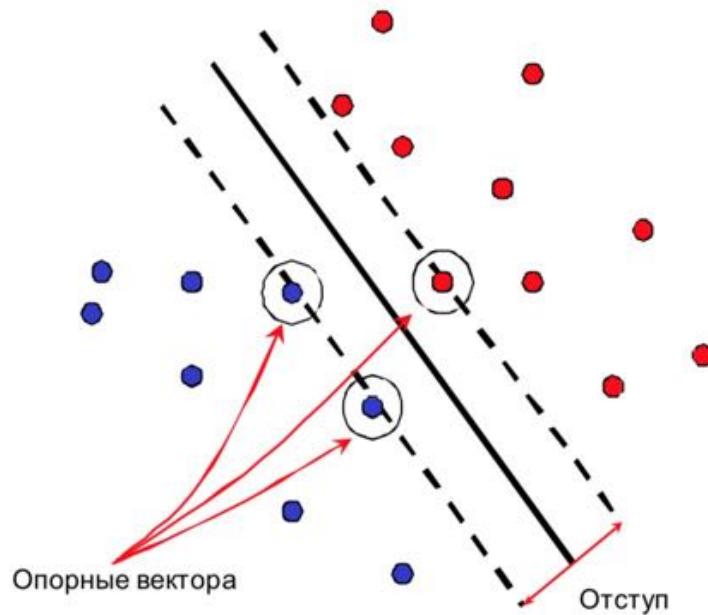
Обучение детектора

- Сколько на изображении объектов «пешеход» и сколько фрагментов фона?
- Выделение объектов ассиметричная задача: объектов гораздо меньше, чем «не-объектов»
- Вдобавок, класс «не объект» очень сложный – нужно много разных данных для обучения
- Для SVM желательно одинаковое количество и фона, и объекта



Обучение детектора

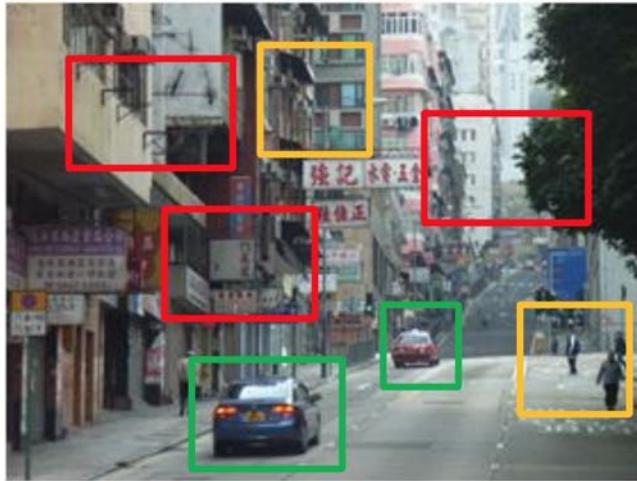
Какие нам нужны примеры фона для получения наилучшего детектора?



Самые трудные!

Как их найти?

Бутстрэппинг (Bootstrapping)



- Выбираем отрицательные примеры
Случайным образом
 - Обучаем классификатор
 - Применяем к данным
 - Добавляем ложные обнаружение к выборке
 - Повторяем
-
- Смысл:
 - Ложные обнаружения для первого детектора – сложные (hard negative)
 - Пусть наша выборка фона будет маленькой, но сложной и представительной

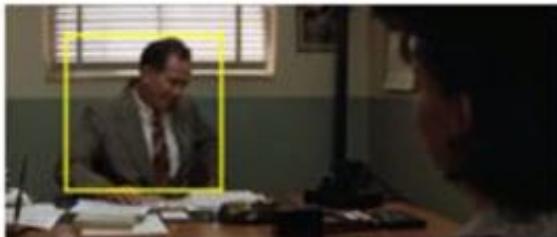
Трудный пример

Трудный
отрицательный
пример



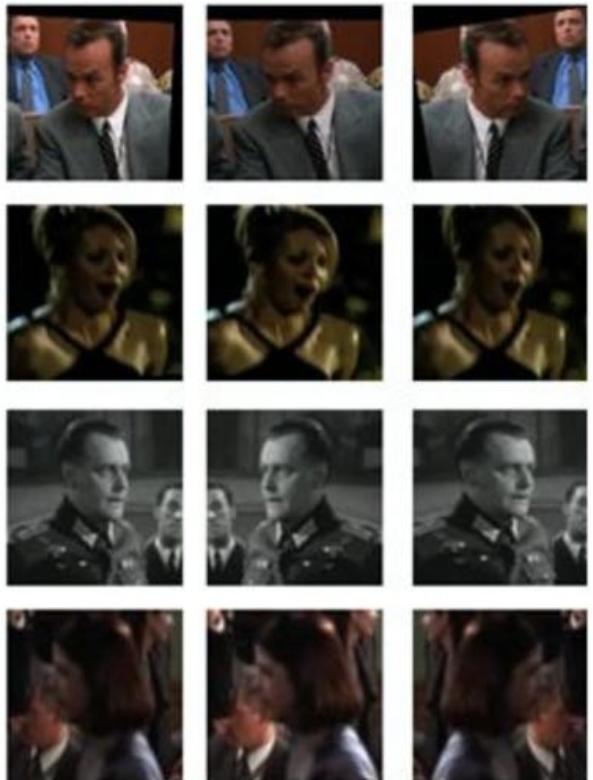
Пример - поиск “торса”

- Хотим построить детектор «верхней части тела и головы»
- Воспользуемся схемой HOG + линейный SVM с бустраппингом
- Данные
 - 33 фрагмента фильмов из базы Hollywood2
 - 1122 кадров с размеченным объектами
- На каждом кадре отмечены 1-3 человека, всего 1607 людей, это маловато



Аугментируем данные

- Из 1607 эталонных примеров получили ~32000 искаженных (jittered) примеров
- Сколько отрицательных примеров можно набрать из 1100 кадров?
 - Гораздо больше 32k.
- Воспользуемся схемой бутстрэппинга для поиска «трудных примеров» фона



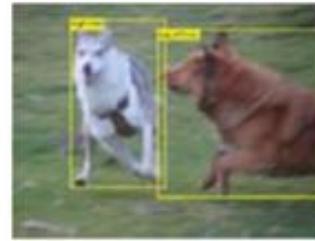
Резюме алгоритма HOG+SVM

- Используем скользящее окно
- Вычисляем вектор-признак на основе HOG
 - разбиваляем окно на ячейки
 - в каждой ячейке считаем гистограмму ориентации градиентов
- Обучаемый линейный SVM
- Для обучения:
 - Размножаем (шевелим) эталонные примеры объектов
 - Используем схему bootstrapping для выбора примеров фона
 - На первой стадии берём случайные окна для фона
 - На следующих стадиях выбираем ложные срабатывания детектора как «трудные» примеры

Проблемы скользящего окна



Разный размер
объектов



Соотношение
размеров окна



Перекрытие и плохое соответствие
формы объекта окну



Множественные отклики

Разный размер объектов



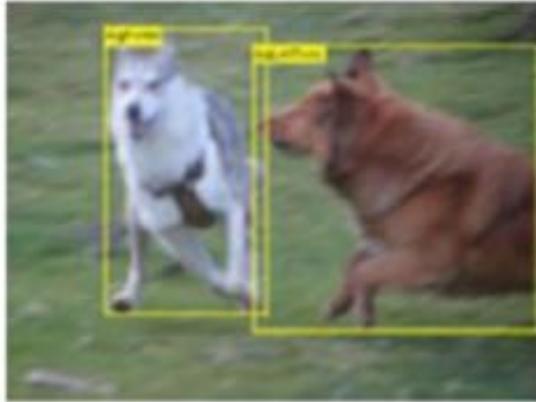
Сканируем изображение рамками разного размера и разных пропорций

Разный размер объектов



- Вместо изменения размеров рамки можем построить «пирамиду» изображений разных размеров и сканировать одной рамкой
- Сколько потребуется масштабов?
 - На практике до 50-70

Пропорции объектов

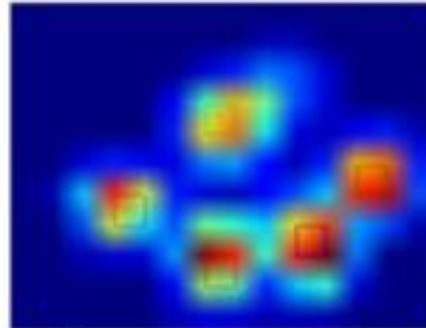


- Будем сканировать рамками разных пропорций
- С учетом масштабов нам придётся сканировать
рамок $\times N$ масштабов раз
- Поэтому общее количество сканирований может
достигать 150-200

Множественные отклики



Множественные отклики



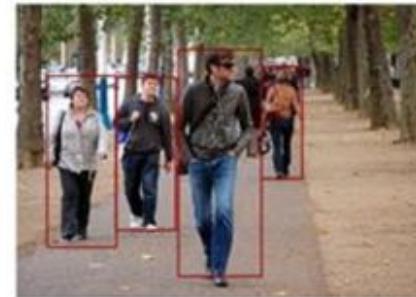
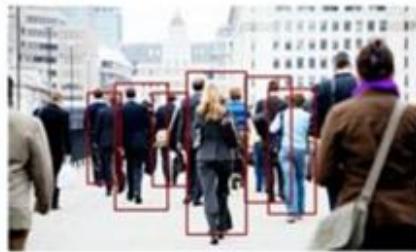
Карта откликов



Пирамида изображений

- В одном масштабе может быть множество откликов разной силы
- Мы выберем из них точки, в которых значение достигает локального максимума («Подавление не-максимумов»)
- При наличии нескольких масштабов нужно выбирать максимальные значения по 3х мерной области
- Параметры выделения максимумов могут значительно влиять на результат!

Перекрытия



Самая сложная проблема!

Требования к детектору

- Для изображения в 1МП нужно просмотреть порядка 1M окон (например, для случая лиц)
- На одном изображении обычно 0-10 лиц



- Чтобы избежать ложных обнаружений (false positives) ошибка 2го рода должна быть ниже 10^{-6}
- Нужно быстро отбрасывать ложные окна

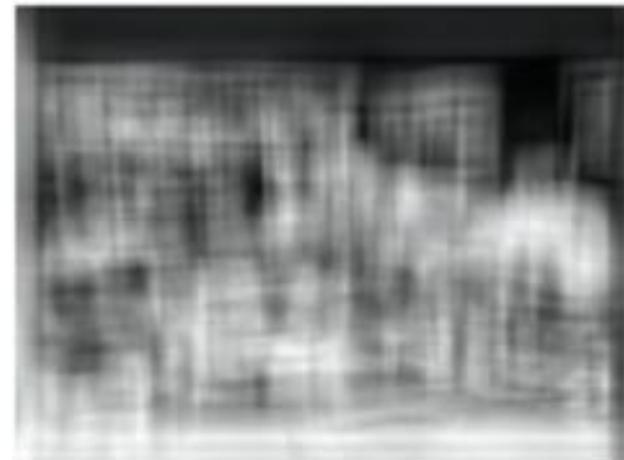
Детектор Viola-Jones

- Основополагающий метод для поиска объектов на изображении в реальном времени
- Обучение очень медленное, но поиск очень быстр
- Основные идеи:
 - Признаки Хаара в качестве слабых классификаторов
 - Интегральные изображения для быстрого вычисления признаков
 - Бустинг для выбора признаков
 - Каскад (Attentional cascade) для быстрой отбраковки окон без лица

Вспомним сопоставление шаблонов

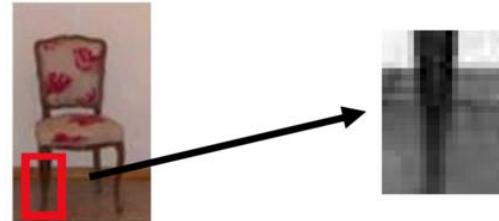


Найти стул в изображении



Что если воспользоваться маленькими фрагментами?
Искать не стул, а часть стула?

Части объектов



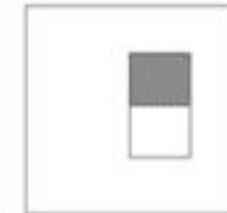
Не так плохо, срабатывает на ножках

Признаки Хаара

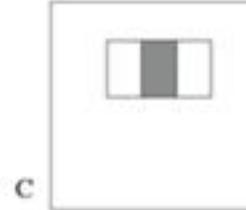
“Прямоугольные
фильтры”



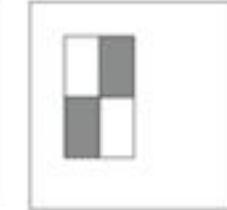
A



B



C



D

Value =

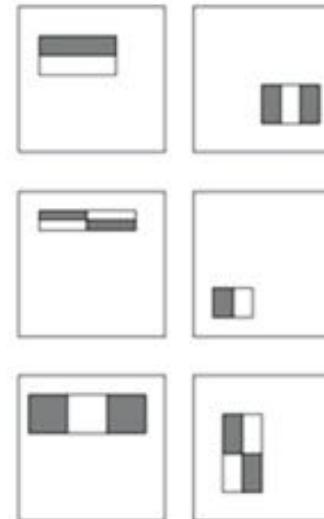
$$\sum (\text{pixels in white area}) - \sum (\text{pixels in black area})$$

Слабые детекторы / классификаторы

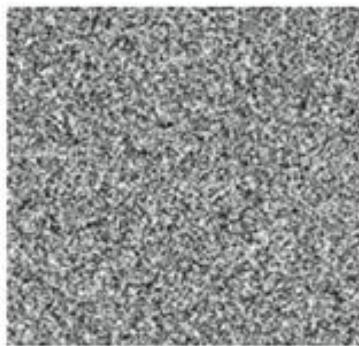
- Определяем слабые классификаторы на основе прямоугольных признаков

$$h_t(x) = \begin{cases} 1 & \text{if } f_t(x) > \theta_t \\ 0 & \text{otherwise} \end{cases}$$

Значение признака
окно
порог



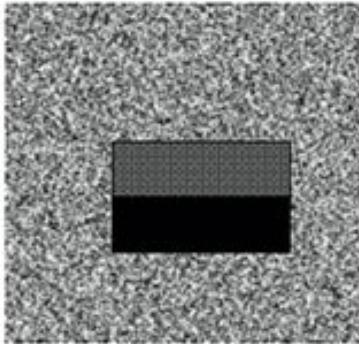
Пример



Source

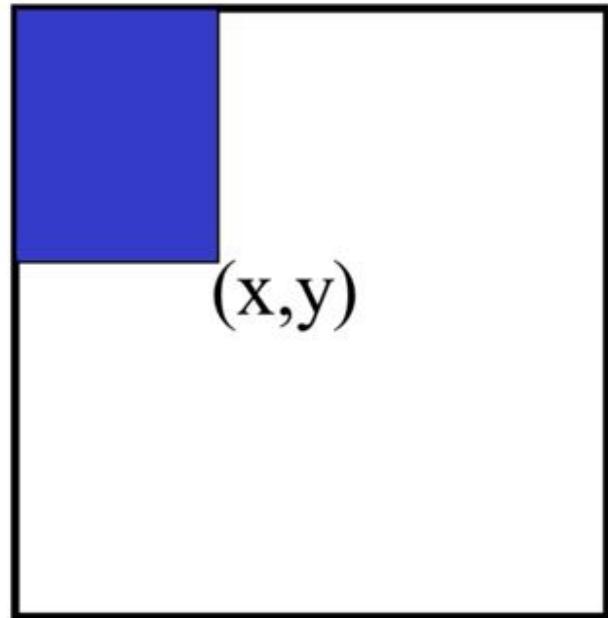


Result

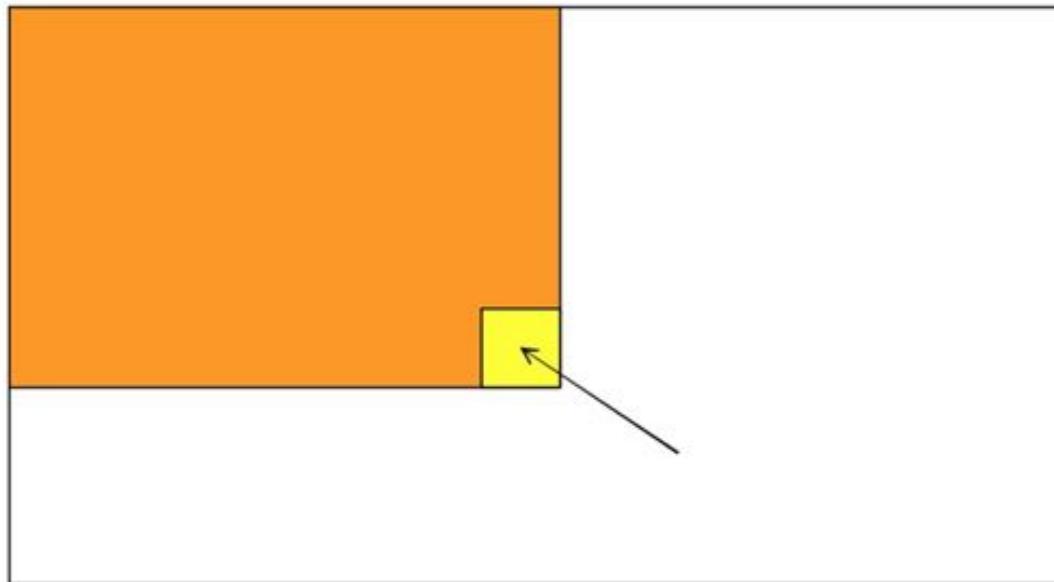


Интегральные изображения

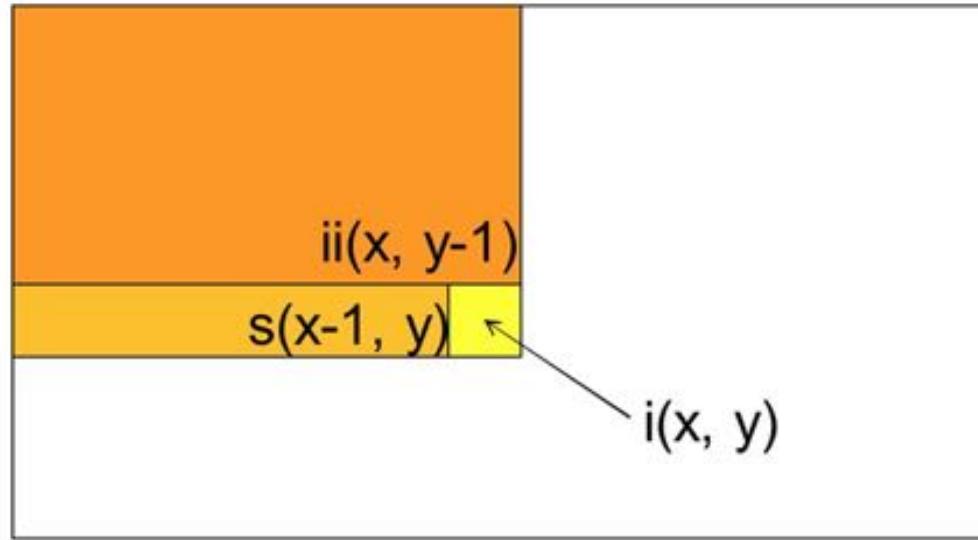
- Значение каждого пикселя (x,y) равно сумме значений всех пикселов левее и выше пикселя (x,y) включительно
- Интегральное изображение рассчитывается за один проход



Вычисление интегрального изображения



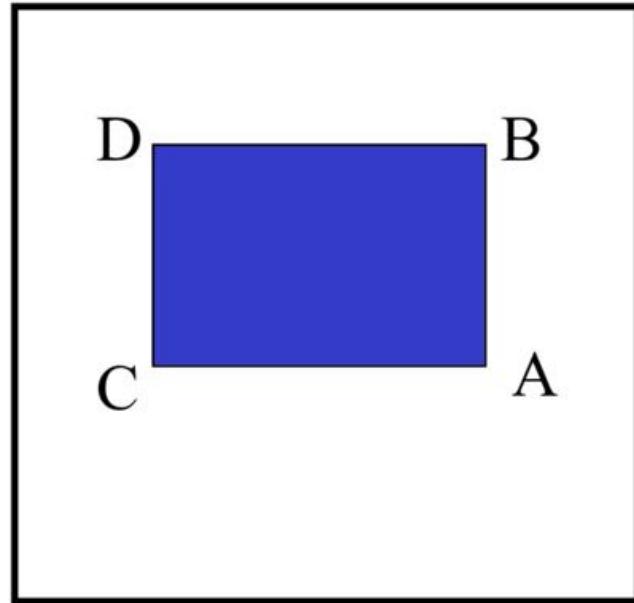
Вычисление интегрального изображения



- Сумма по строке: $s(x, y) = s(x-1, y) + i(x, y)$
- Интегральное изображение: $ii(x, y) = ii(x, y-1) + s(x, y)$

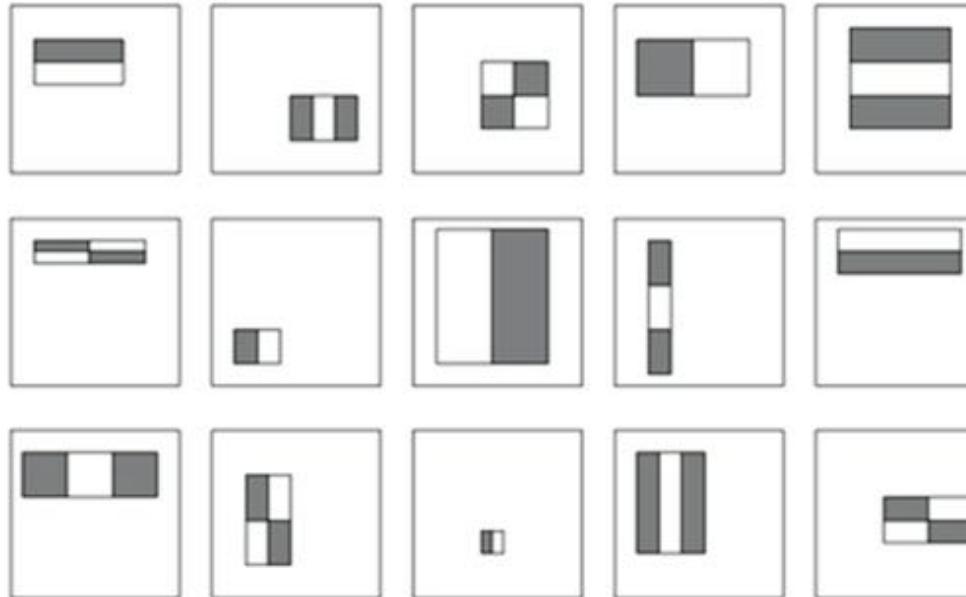
Вычисление суммы в прямоугольнике

- Пусть A,B,C,D – значения интегрального изображения в углах прямоугольника
- Тогда сумма значений пикселов в исходном изображении вычисляется по формуле:
$$\text{sum} = A - B - C + D$$
- 3 операции сложения для любого прямоугольника



Количество признаков

Для окна поиска 24x24 пикселя, число возможных прямоугольных признаков достигает ~160,000!



Выбор признаков

- Для окна поиска 24x24 пикселя, число возможных прямоугольных признаков (слабых классификаторов) достигает ~160,000!
- Каждый по отдельности очень «слабый»
- В процессе поиска вычислять все признаки нереально
- Хороший классификатор должен использовать небольшое подмножество всевозможных признаков
 - «Комитетные классификаторы»
- Будем выбирать нужные признаки и строить из них линейные комбинации с помощью бустинга

БУСТИНГ

- Бустинг – схема построения сильного классификатора (комитета) за счёт комбинирования слабых классификаторов
 - Слабый классификатор должен быть лучше монетки
- Обучение состоит из нескольких этапов усиления (boosting rounds)
 - На каждом этапе выбираем слабый классификатор, который лучше всех сработал на примерах, оказавшихся трудными для предыдущих классификаторов
 - «Трудность» записывается с помощью весов, приписанных примерам из обучающей выборки
 - Составляем общий классификатор как взвешенную линейную комбинацию слабых классификаторов

Бустинг (Viola-Jones)

$$h(\mathbf{x}) = \text{sign} \left[\sum_{j=0}^{m-1} \alpha_j h_j(\mathbf{x}) \right]$$

$h(\mathbf{x})$ - финальный классификатор

m - число классификаторов в ансамбле

$h_j(\mathbf{x})$ - базовый классификатор

α_j - вес базового классификатора (зависит от ошибки)

Бустинг (Viola-Jones)

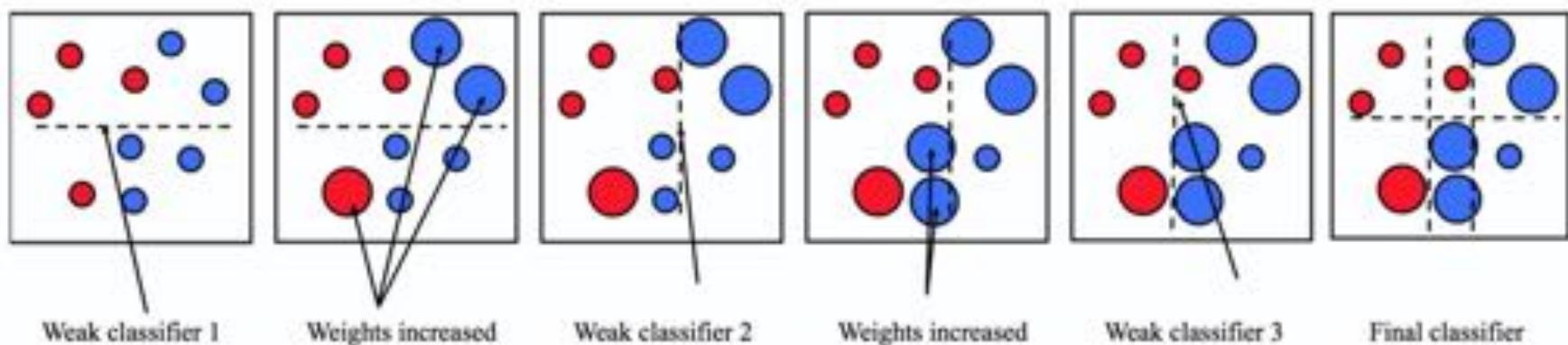
$$h_j(\mathbf{x}) = a_j[f_j < \theta_j] + b_j[f_j \geq \theta_j] = \begin{cases} a_j & \text{if } f_j < \theta_j \\ b_j & \text{otherwise,} \end{cases}$$

a_j - решение принимается в случае если значение фактора f меньше порога theta
b_j - принимается в противном случае

Бустинг (Viola-Jones)

- при инициализации присваиваем всем объектам в выборке одинаковый вес
- на очередном шаге j среди множества простых классификаторов выбираем тот, который дает наибольший прирост качества
- вычисляем коэффициент α_j пропорциональный приросту качества с учетом весов объектов
- увеличиваем вес объектов, которые остаются классифицированы неверно

Бустинг (Viola-Jones)



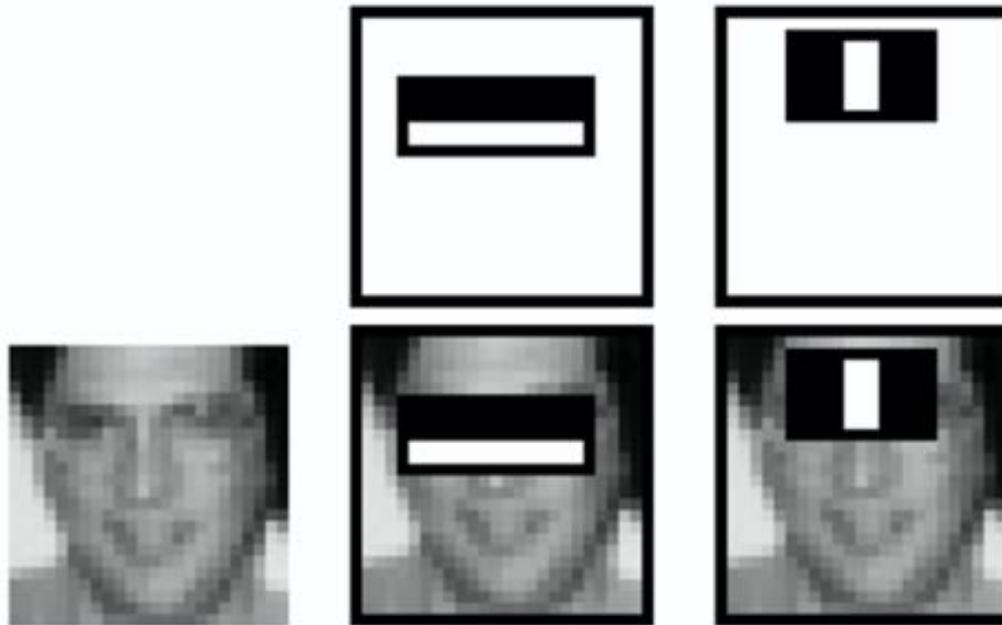
Бустинг (Viola-Jones)

- Плюсы
 - +Универсальность
 - +Высокая скорость сходимости
 - +Высокая обобщающая способность
 - +Возможность очень эффективно и программной реализации и распараллеливания
 - +Простота метода и отсутствия параметров
- Минусы
 - Трудность определения нужного числа итераций обучения (зачастую, ошибка на контроле продолжает падать и после нуля эмпирического риска)

Бустинг (Viola-Jones)

- Определяем слабые классификаторы на основе прямоугольных признаков
- Для каждого этапа бустинга:
 - Вычисляем каждый прямоугольный признак на каждом примере
 - Выбираем наилучший порог для каждого признака (обучаем слабый классификатор)
 - Выбираем наилучший слабый классификатор и добавляем его в линейную комбинацию
 - Перевзвешиваем выборку
- Вычислительная сложность обучения: $O(MNK)$
 - M этапов, N примеров, K признаков

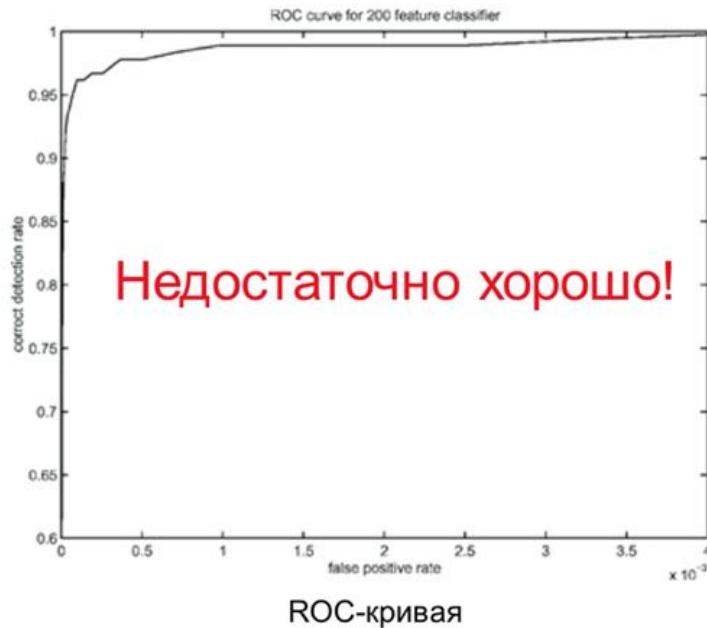
Бустинг для поиска лиц



Первые два классификатора, выбранные бустингом дают 100% detection rate и 50% false positive rate

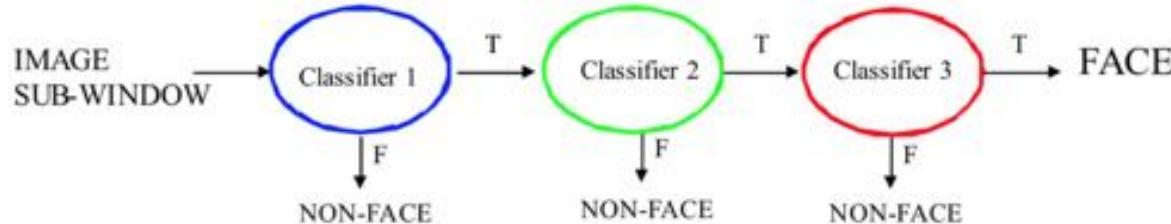
Бустинг для поиска лиц

Классификатор из 200 признаков дает 95% точность и долю ложноположительных срабатываний 1 из 14084



Каскад (Attentional Cascade)

- Начинаем с простых классификаторов, которые отбрасывают часть отрицательных окон, при этом принимая почти все положительные окна
- Положительный отклик первого классификатора запускает вычисление второго, более сложного, классификатора, и т.д.
- Отрицательный отклик на любом этапе приводит к немедленной отбраковке окна



Каскад (Attentional Cascade)



Обучение каскада

- Задаем требуемые значения уровня обнаружения и доли ложноположительных срабатываний для каждого этапа
- Добавляем признаки до тех пор, пока параметры текущего этапа не достигнут заданного уровня
 - Приходится понижать порог AdaBoost для максимизации обнаружения (в противоположность минимизации общей ошибки классификации)
 - Тестирование на отдельном наборе (validation set)
- Если общий уровень ложноположительных срабатываний недостаточно низок, добавляем очередной этап
- Ложные обнаружения на текущем этапе используются как отрицательные примеры на следующем этапе

Резюме: детектор Viola-Jones

- Прямоугольные признаки как слабые классификаторы
- Интегральные изображения для быстрого вычисления признаков
- Бустинг для выбора признаков
- Каскад классификаторов для быстрого выбраковки отрицательных окон
- Ориентированное число простых моделей в ансамбле ~3000
- Поиска скользящим окном по изображению большим ансамблем может занимать слишком много времени
- На практике строят каскады ансамблей меньшего размера
- Область классифицируется очередным ансамблем в каскаде, только если предыдущий ансамбль классифицировал область как положительную

Недостатки Viola-Jones

- Малое количество используемых каналов изображения и признаков
 - Серые изображения, фильтры Хаара
- Недостаточная точность
 - 10^{-6} – слишком много ложных срабатываний
- Множественные отклики у правильных обнаружений
- «Дискретность» - возможность завершения только в конце этапа
- Очень долгое время обучения

ТРЕКИНГ

Трекинг



Постановка задачи

- отслеживание перемещения объекта в кадре
- предсказание направления перемещения
- сообщение в случае потери объекта

Сложности

- несколько объектов на кадре
- частичное или полное перекрытие другими объектами
- реидентификация после временной потери из зоны видимости
- объект может изменять внешний вид во время наблюдения (повороты, масштаб)

Детекция и трекинг

- как правило, детекция сложнее чем трекинг
- трекинг использует информацию из предыдущих кадров
- трекер может накапливать ошибку
- детектор ищет объект на изображении без учета предыдущих кадров
- на практике алгоритм детекции запускается реже чем трекинг

Обзор подходов

- трекинг отдельного объекта
 - детектируем объект на первом кадре, затем отслеживаем его перемещение с помощью трекера
- трекинг нескольких объектов
 - детектор для каждого кадра детектор выдает предсказания, задача трекера - поставить в соответствие детекции между кадрами

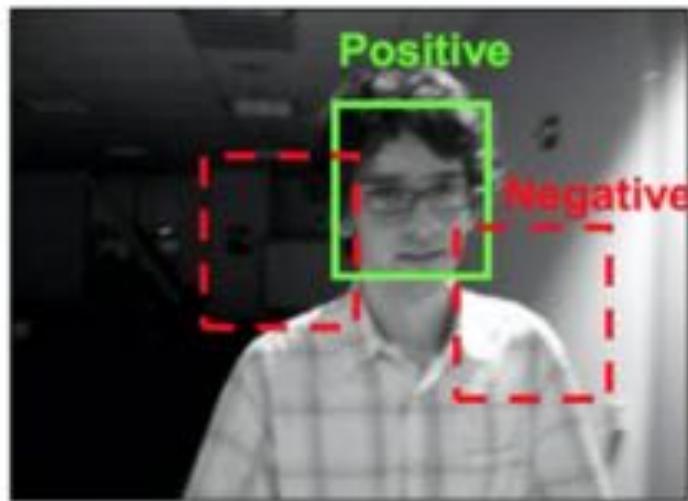
Алгоритмы трекинга в OpenCV

- для использования трекеров в opencv необходимо установить пакет [opencv-contrib-python](#)
- трекеру необходимо хранить состояние между вызовами
- состояние хранится в объекте трекера
- создать объект трекера можно с помощью функции
cv2.Tracker<алгоритм_трекинга>_create()

Boosting Tracker

- основан на онлайн версии алгоритма AdaBoost
- детекция является положительным примером для обучения
- область вокруг детекции - отрицательные примеры
- для определения объекта на очередном кадре запускается поиск в окрестности предыдущей детекции
- качество работы среднее

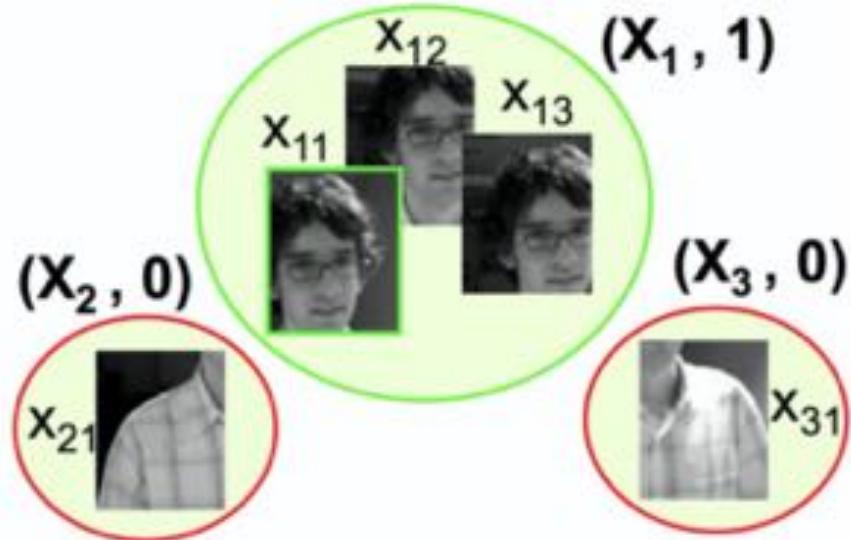
Boosting Tracker



MIL - Multiple Instance Learning

- решение аналогично Boosting Tracker
- в отличие от Boosting Tracker семплируется несколько примеров вокруг детекции
- полученные семплы объединяются в группу
- задача классификатора предсказать класс группы
- качество детекции выше, чем у Boosting Tracker
- устойчив к частичным перекрытием объектов
- не восстанавливается после полной потери объекта из зоны видимости

MIL - Multiple Instance Learning



Резюме

- Сегментация изображения позволяет работать не со всем изображением в целом, а с отдельными значимыми фрагментами
- Сегменты могут выбираться по критериям однородности по яркости, цвету, текстуре и по комбинации этих признаков
- Бинаризация и выделение связных компонент
- Последовательное сканирование
- Метод K-средних
- Водораздел (WaterShed)
- Super pixel. Energy driven sampling.
- Graph Cut

Резюме

- Скользящее окно – основной способ сведения задачи выделения объектов к задаче категоризации изображений
- Каскад классификаторов – основной способ повышения скорости и точности работы
- В качестве признаков для базовых классификаторов используются разностные свертки
- Viola-Jones работает real-time, но требует сравнительно много времени и большого числа примеров для обучения
- Нужно уделить большое внимание построению хорошей выборки для обучения (jittering, bootstrapping)

Резюме

- один из способов решения задачи трекинга - трекинг через детекцию (tracking by detection)
- на кадре с детекцией обучаем модель
- обученную модель применяем на следующем шаге