

# Лекция 9

Image segmentation - k-means, mean-shift.  
BoW. SVD compression.

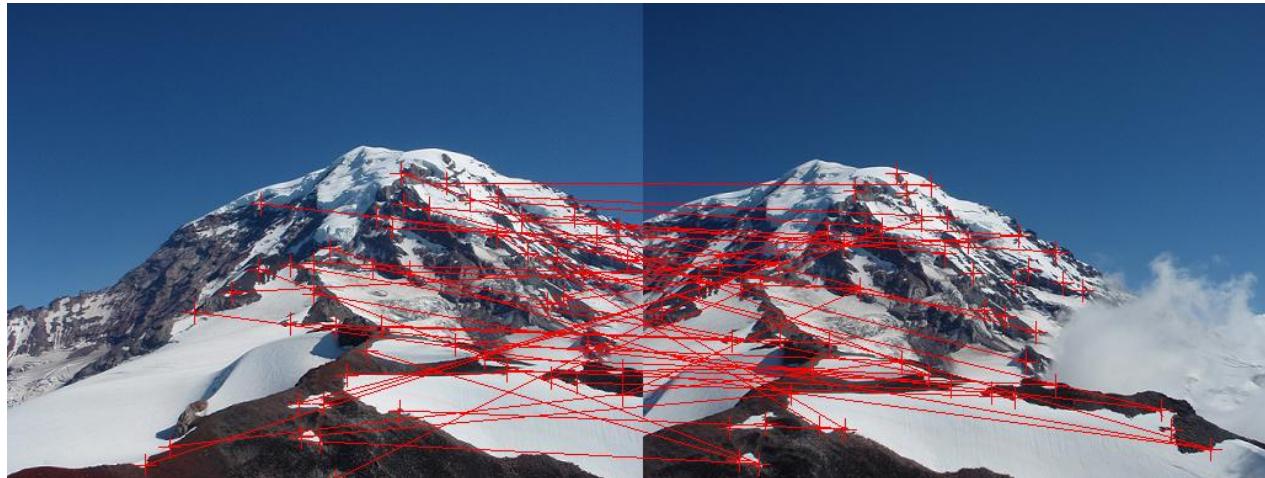
11.02.2020  
Руслан Алиев

В ПРЕДЫДУЩИХ СЕРИЯХ

# Panorama stitching

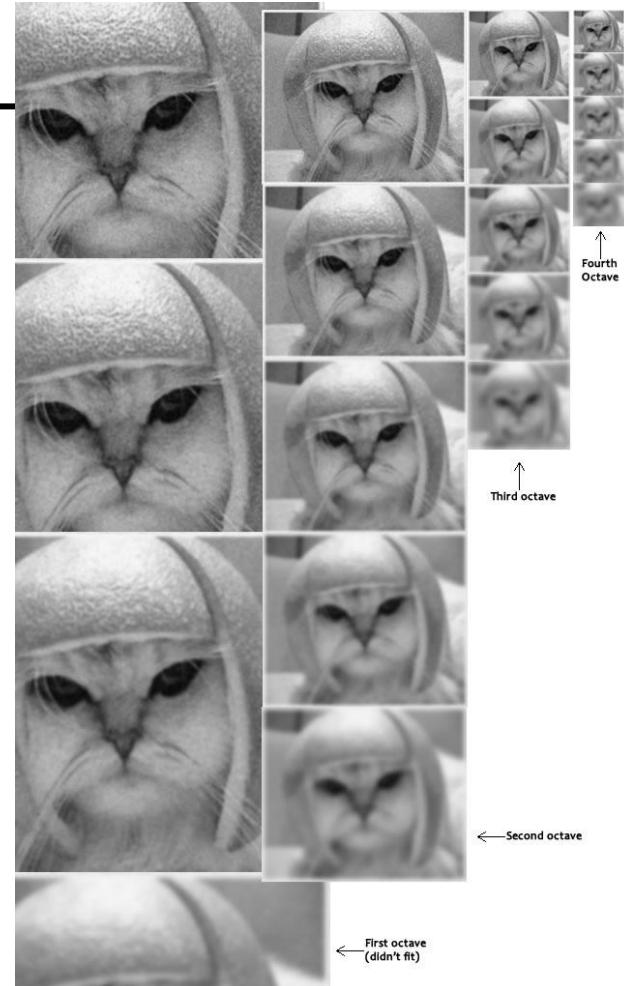
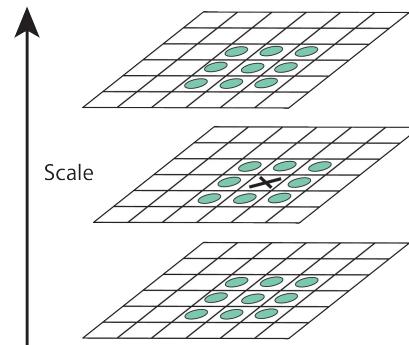
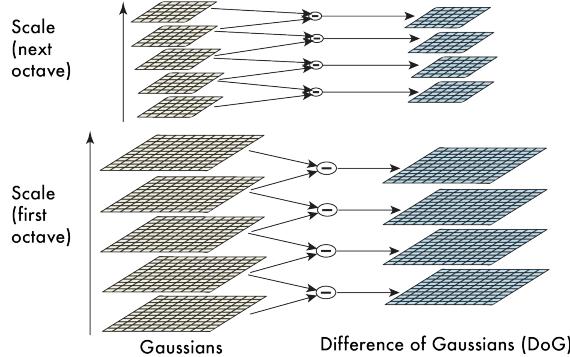
---

1. Находим ключевые точки на обоих изображениях
2. Вычисляем дескрипторы для каждой ключевой точки
3. Матчим точки друг с другом
4. Ищем проективное преобразование с помощью RANSAC (чтобы избавиться от выбросов)
5. Проецируем второе изображение на первое с помощью полученного преобразования



# SIFT

1. Строим scale space
2. Вычисляем DoG
  - a. позволяет находить характерные точки разного размера
3. Находим кандидатов в ключевые точки
4. Находим ключевые точки
5. Находим доминантное направление в каждой ключевой точке
6. Считаем дескриптор через градиенты (аналогично, HoG)

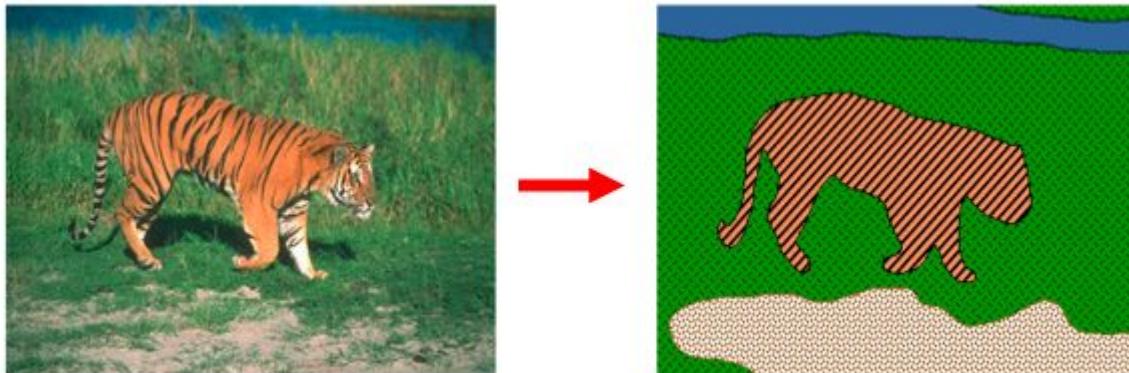


# Image segmentation

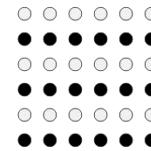
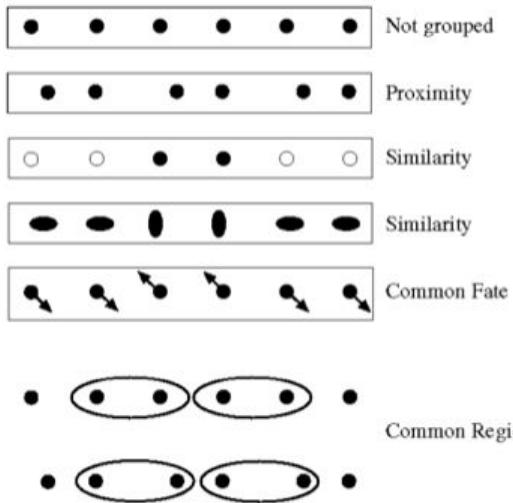
# Image segmentation

---

Задача: разбить похожие пиксели на группы



# Гештальт-психология



Принцип схожести.  
Рисунок воспринимается как строки, а не как колонки



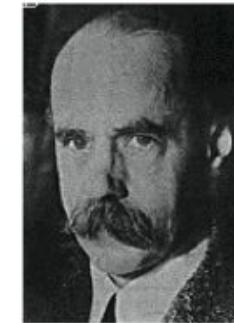
Принцип близости. Правая часть рисунка воспринимается как три столбика.



Принцип замкнутости. Рисунок воспринимается не как отдельные отрезки, а как круг и прямоугольник

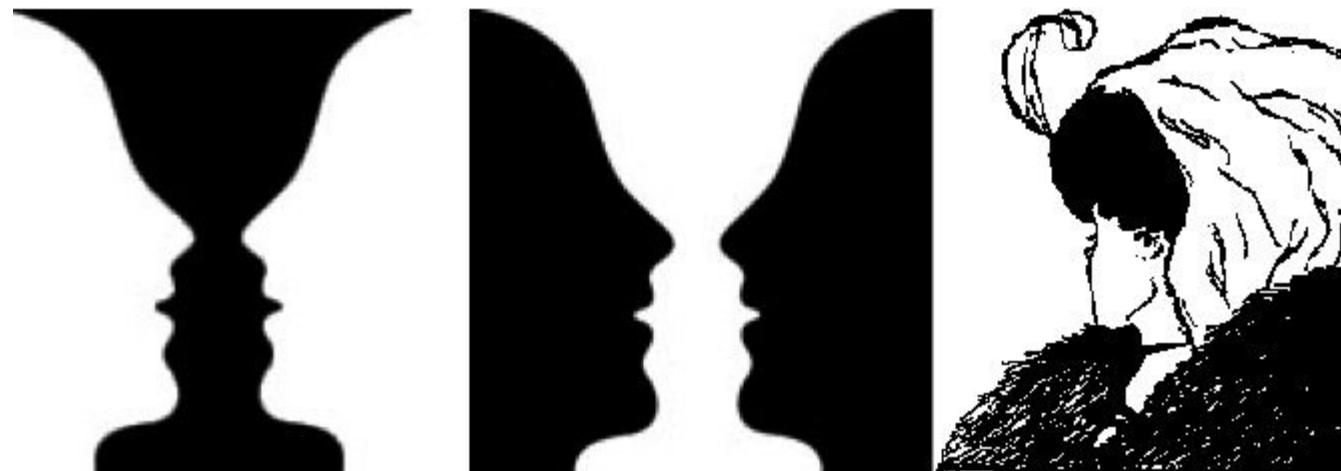
*"I stand at the window and see a house, trees, sky.  
Theoretically I might say there were 327 brightnesses and  
nuances of colour. Do I have "327"? No. I have sky, house,  
and trees."*

Max Wertheimer  
(1880-1943)



# Гештальт-психология

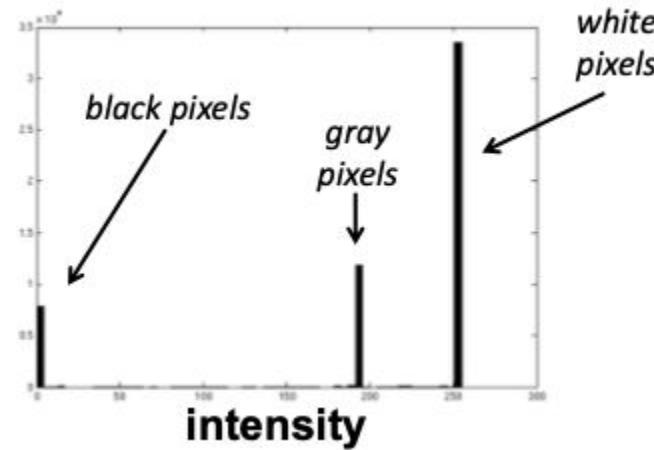
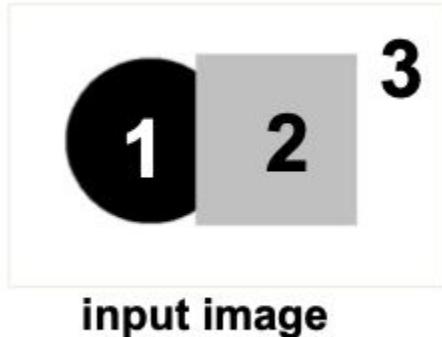
---



# Image segmentation. Пример

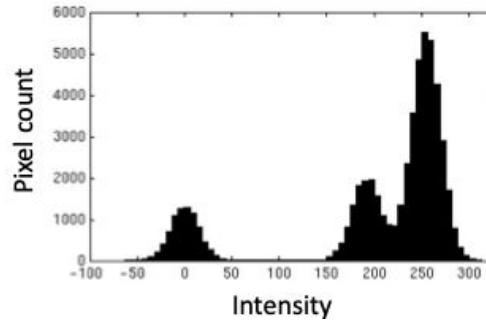
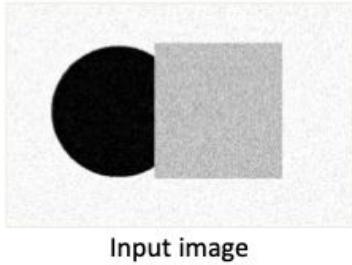
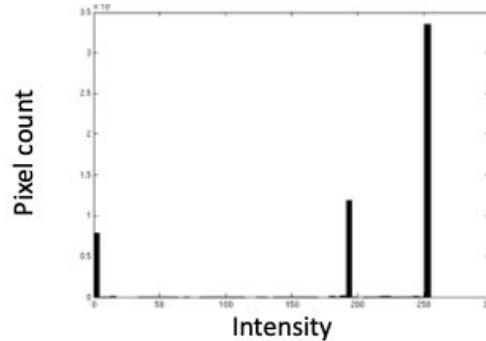
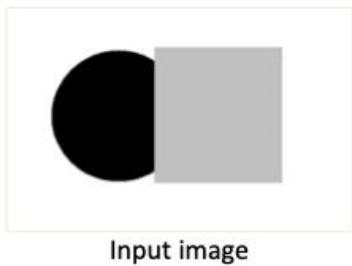
---

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



# Image segmentation. Пример

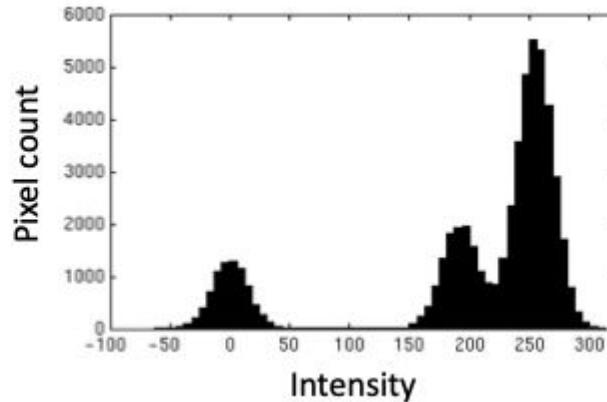
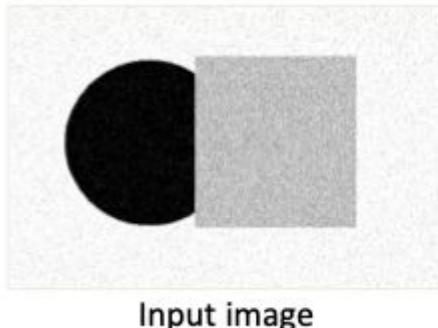
---



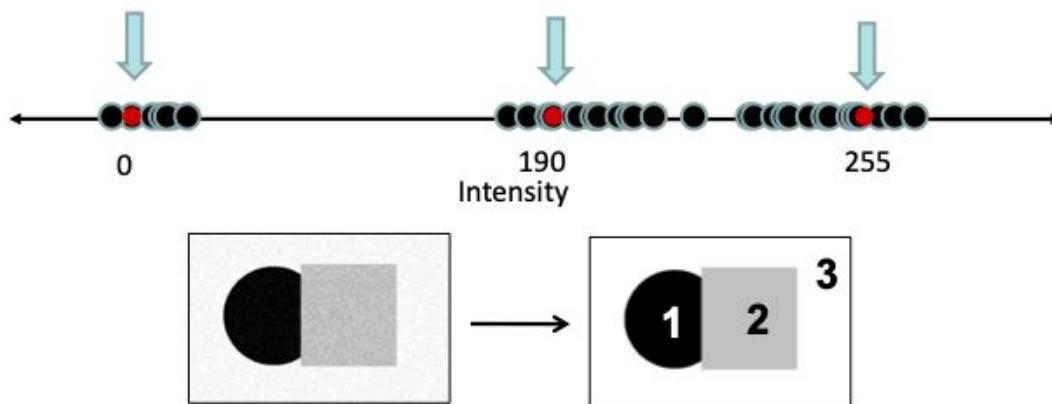
# Image segmentation. Пример

---

- 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

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

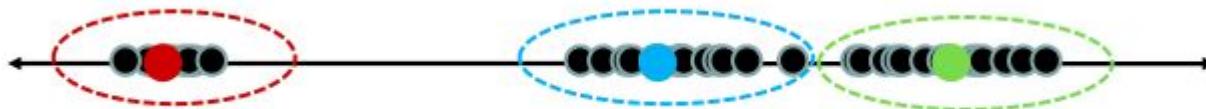
Cluster center      Data  
↓                    ↓  
Whether  $x_j$  is assigned to  $c_i$

# Кластеризация

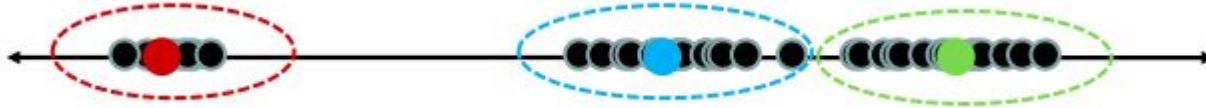
---

“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

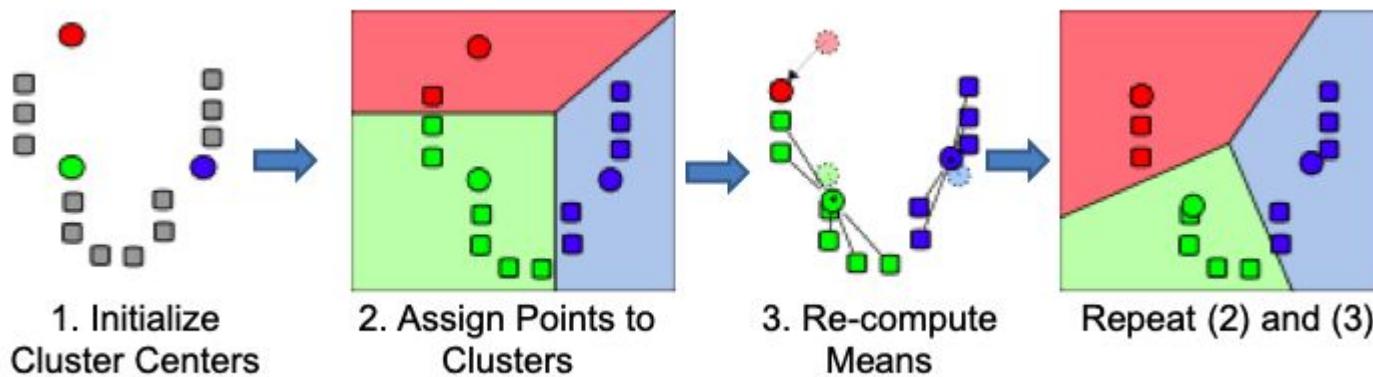
# K-means

---

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

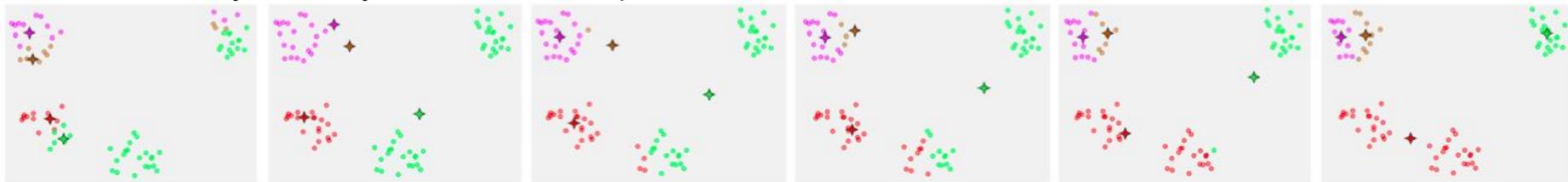
# K-means

---

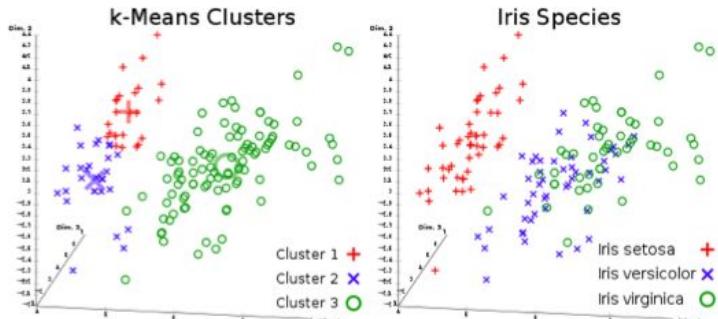


# K-means

- Сходится к локальному минимуму
  - Нужно запускать несколько раз



- Релевантен только для “шарообразных данных”
- Требует выбора К (количество кластеров)
- Требует хранить все имеющиеся данные



# K-means

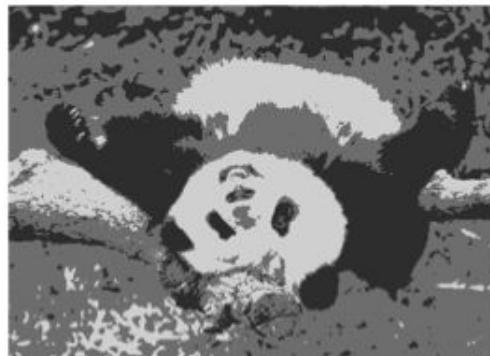
---



Original image



2 clusters



3 clusters

## K-means++

---

- Как можно улучшить инициализацию?
  1. Рандомно выбираем первый центр
  2. Выбираем новый центр с вероятностью пропорциональной  $(x - c_i)^2$ 
    - a. Чем дальше  $x$ , тем большую ошибку он вносит, и тем с большей вероятностью его выберут как новый центр кластера
  3. Повторить пока не наберем  $K$  центров

# Пространство признаков

---

- Как можно улучшить инициализацию?
1. Рандомно выбираем первый центр
  2. Выбираем новый центр с вероятностью пропорциональной  $(x - c_i)^2$ 
    - a. Чем дальше  $x$ , тем большую ошибку он вносит, и тем с большей вероятностью его выберут как новый центр кластера
  3. Повторить пока не наберем  $K$  центров

# Пространство признаков

---

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

Простейший случай: ч/б картинка группировка по интенсивности

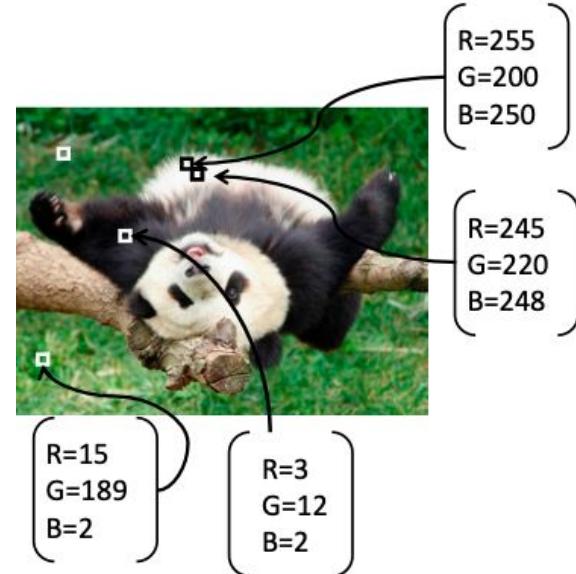
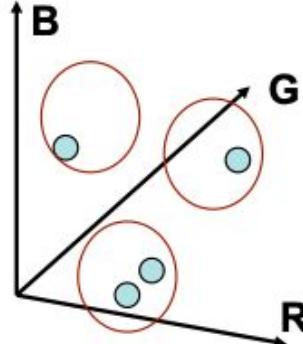


Признаки: интенсивность (1D)

# Пространство признаков

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

Группировка пикселя по схожести цвета (RGB)



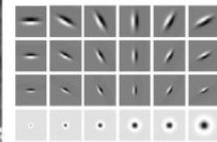
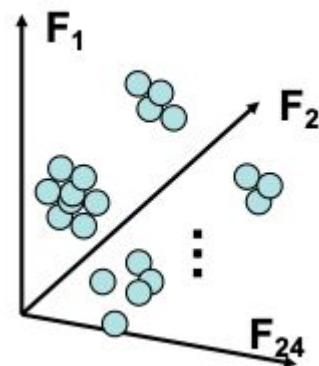
Признаки: значение цвета (3D)

# Пространство признаков

---

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

Группировка пикселя по схожести текстур



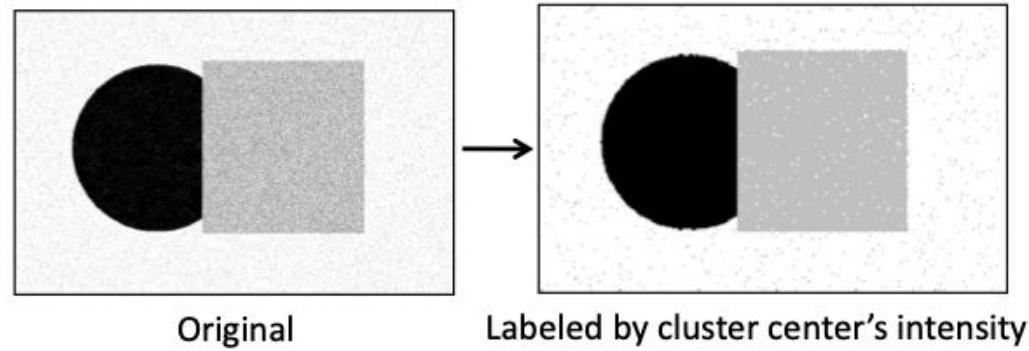
Filter bank of  
24 filters

Признаки: отклики на различные фильтры (24D)

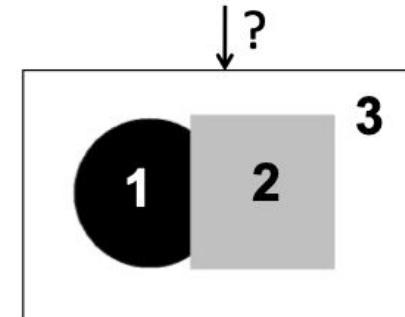
# Пространство признаков

---

Попиксельная кластеризация почти наверняка приведет к большому количеству выбросов



Как учитывать пространственную информацию при кластеризации?

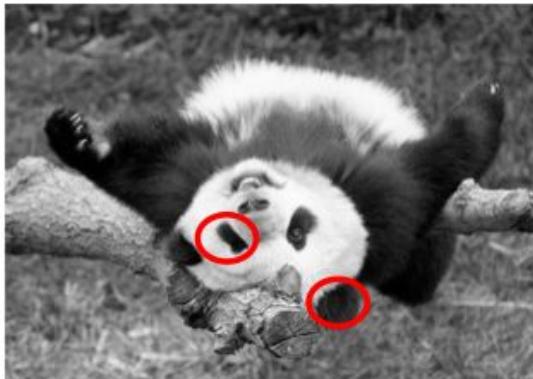
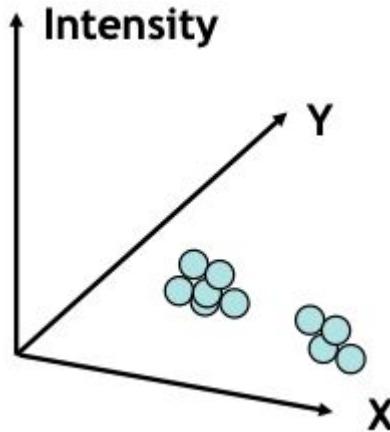


# Пространство признаков

---

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

Группировка пикселя по интенсивности+позиции



# K-means

---

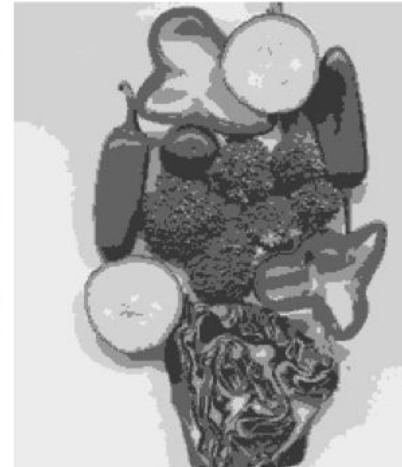
К-means на основе цвета или интенсивности - это своего рода квантование (сжатие) изображения

Такие кластеры не будут пространственно согласованными

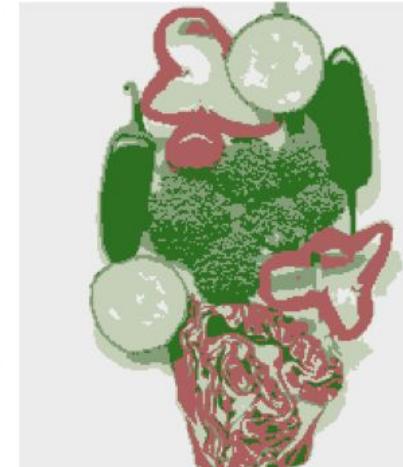
Image



Intensity-based clusters



Color-based clusters



# K-means

---

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

Такие кластеры не будут пространственно согласованными

Кластеры значений ( $r, g, b, x, y$ ) будут пространственно согласованными



# Метрики качества

---

## Как оценивать кластеры?

- **Генеративные метрики**

Насколько хорошо кластеры соотносятся с точками?

- **Дискриминативные метрики**

Насколько хорошо мы предсказали класс для каждого пикселя

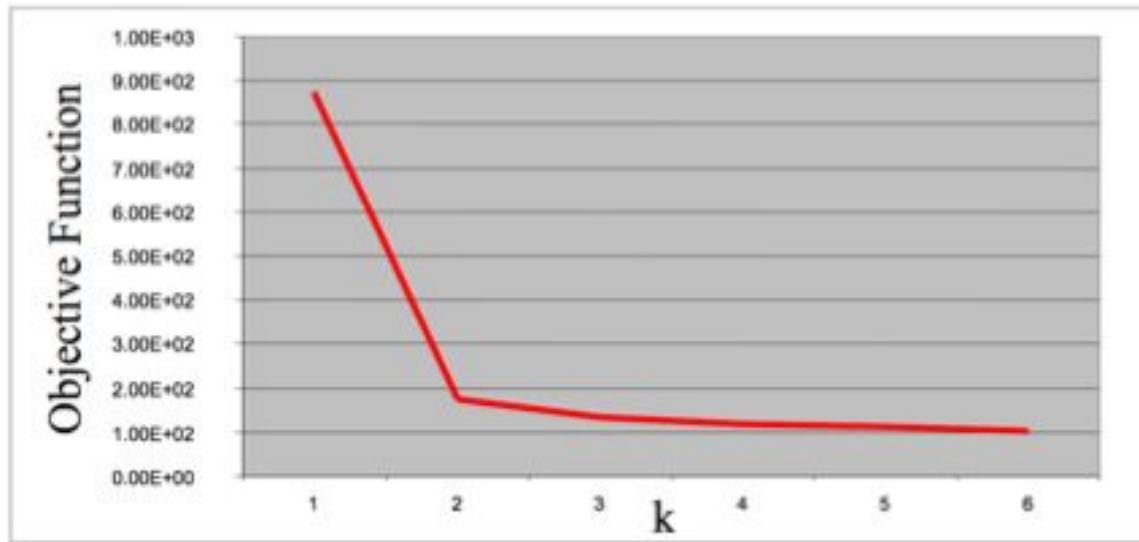
- **Обучения без учителя не подразумевает**

дискриминативных метрик, поскольку у нас нет истинных значений классов пикселей

# Как выбрать количество кластеров?

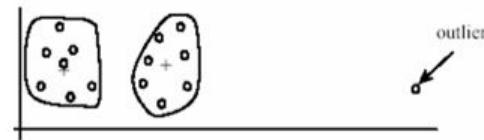
---

Попробовать различное количество кластеров, и посмотреть значение метрики качества для каждого из них

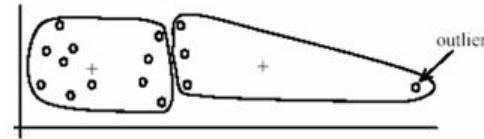


# K-means: pros and cons

- 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



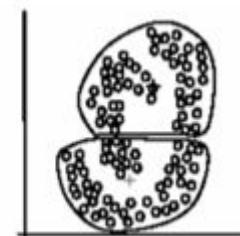
(A): Two natural clusters



(B): Ideal clusters



(A): Two natural clusters

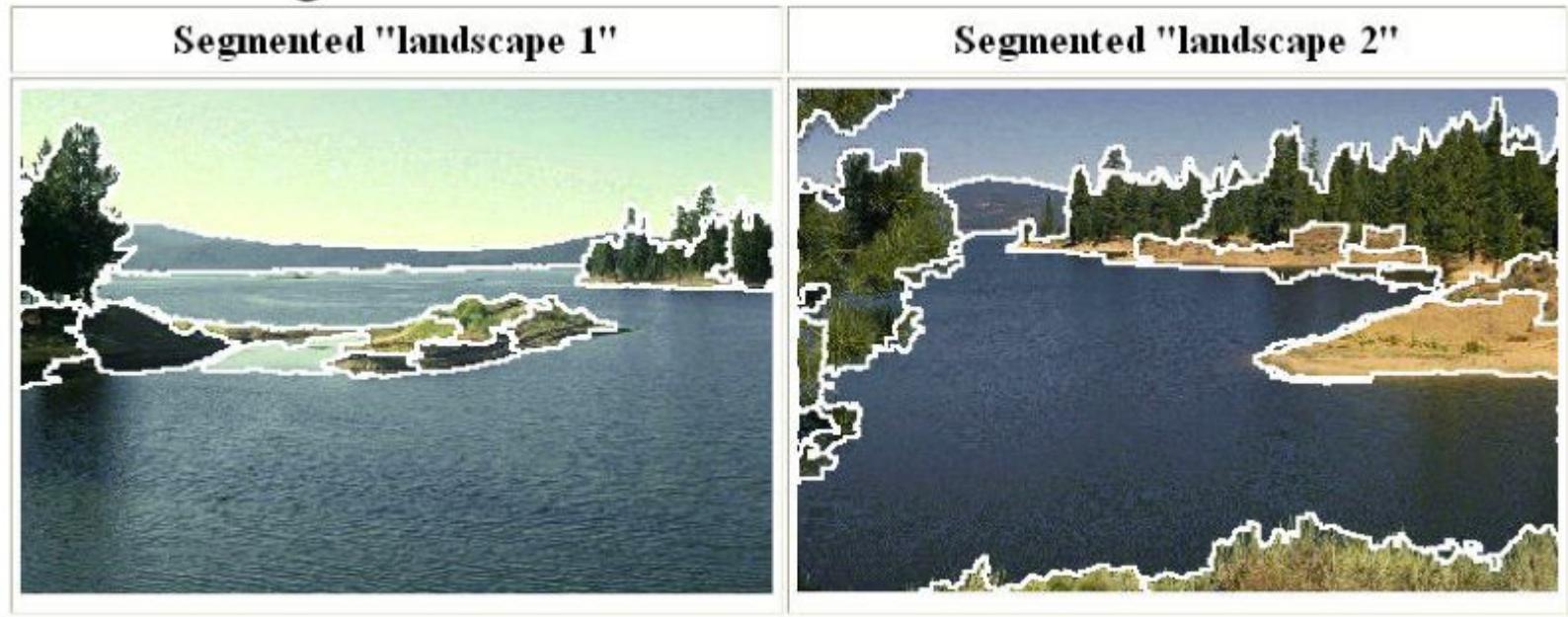


(B):  $k$ -means clusters

# Mean-shift clustering

# Mean-shift clustering

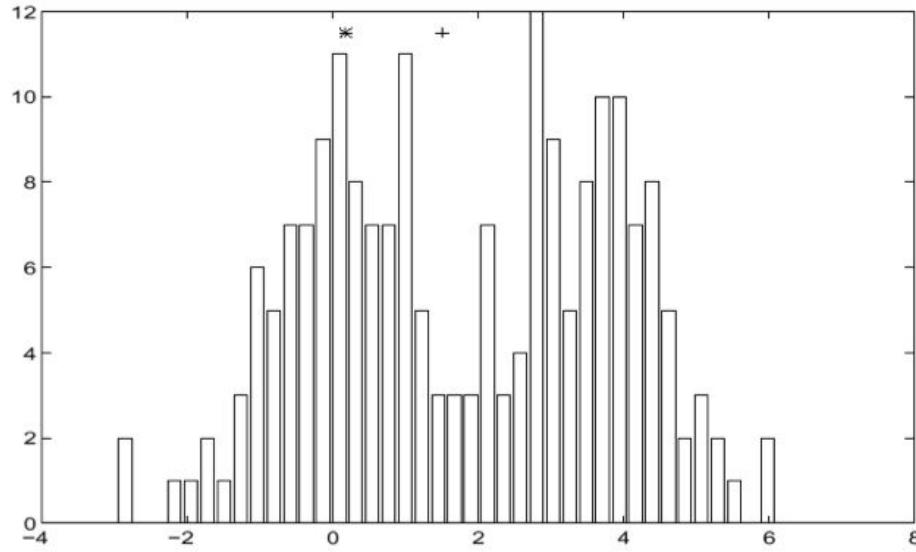
Гибкий алгоритм для cluster-based сегментации



# Mean-shift clustering

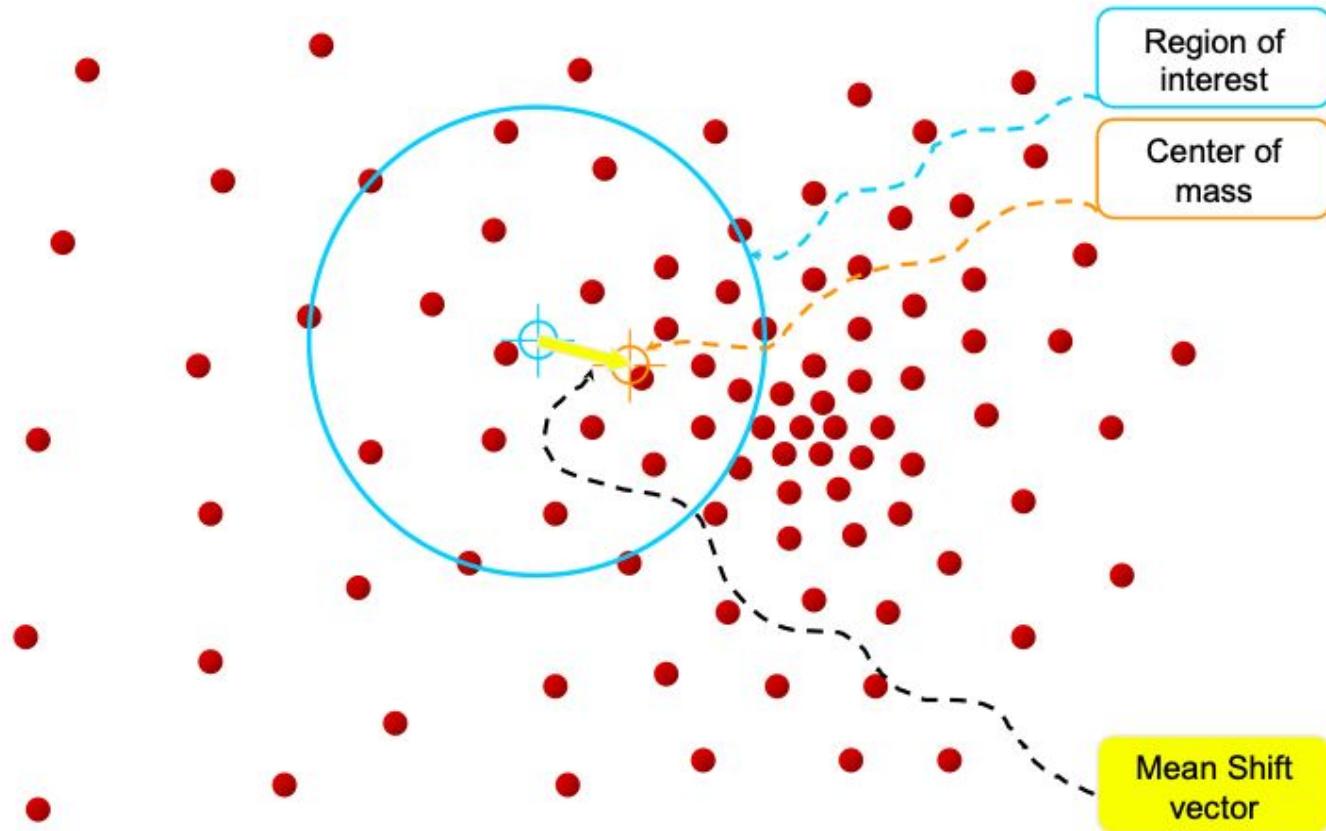
Итеративный алгоритм:

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

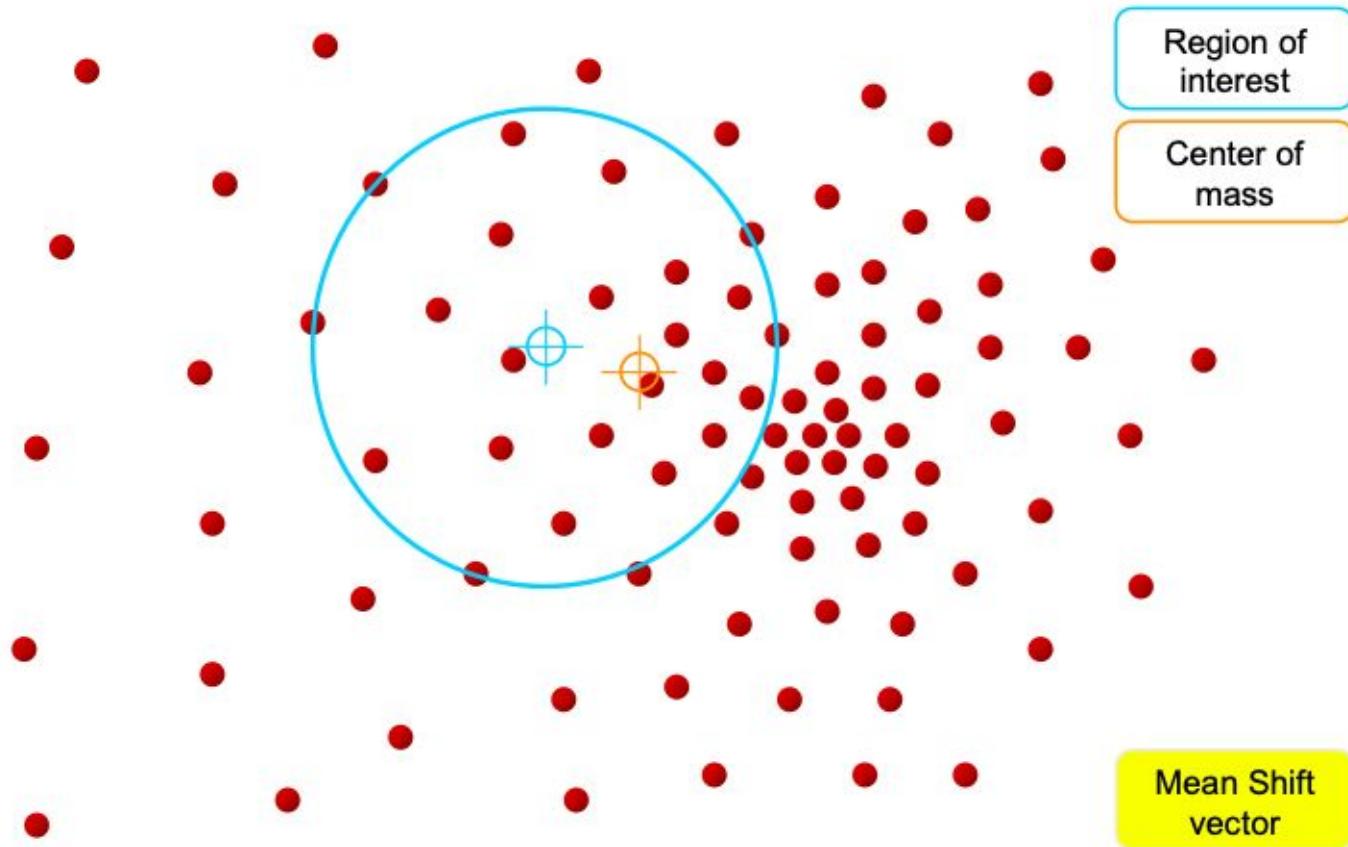


# Mean-shift clustering

---

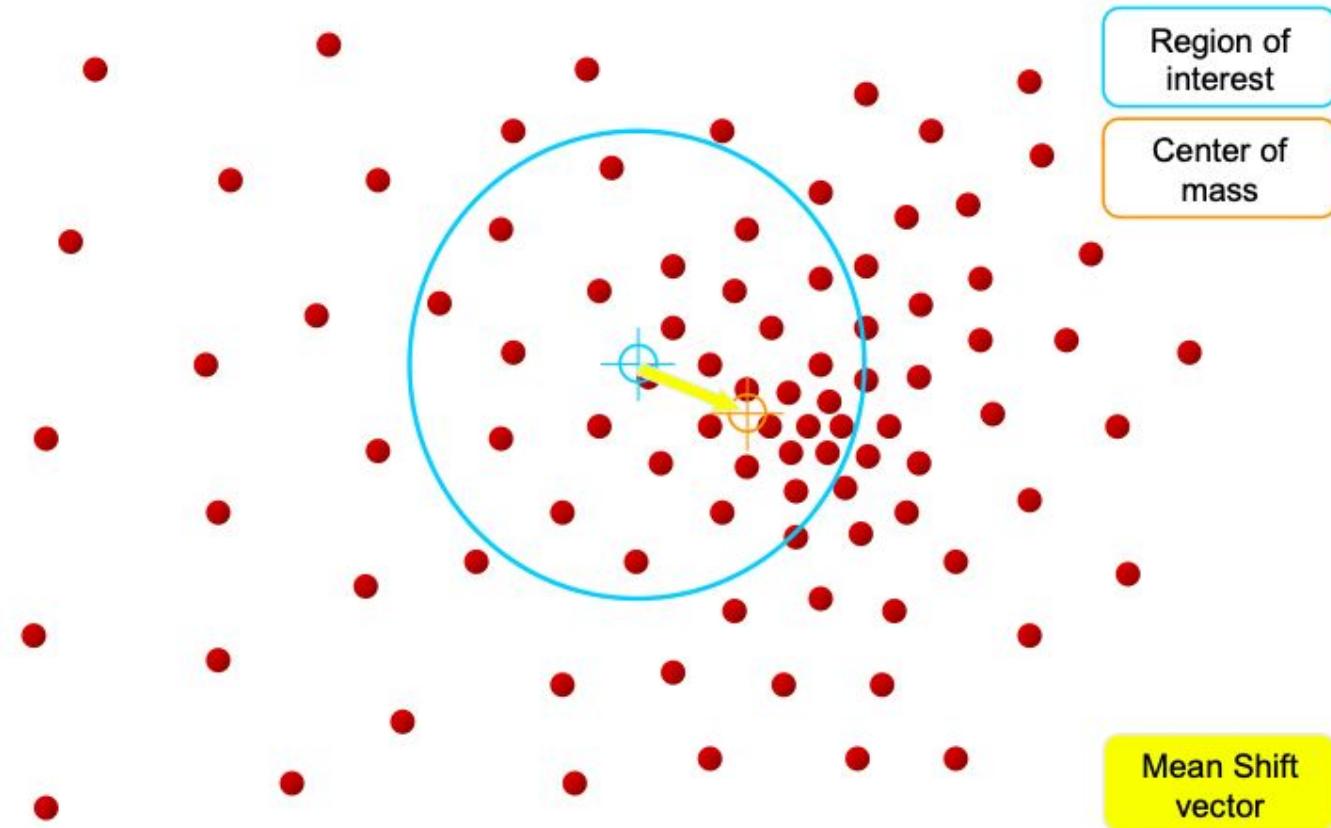


# Mean-shift clustering



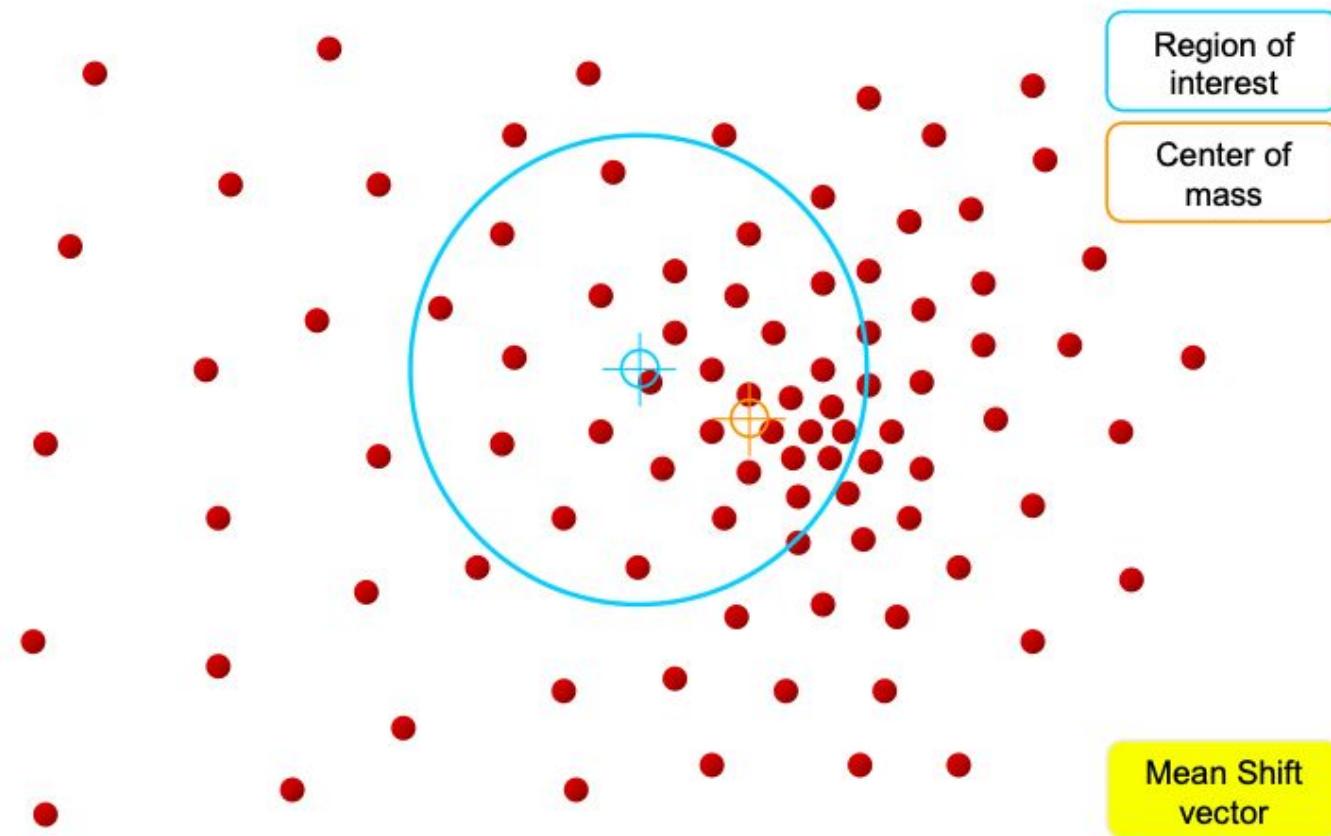
# Mean-shift clustering

---



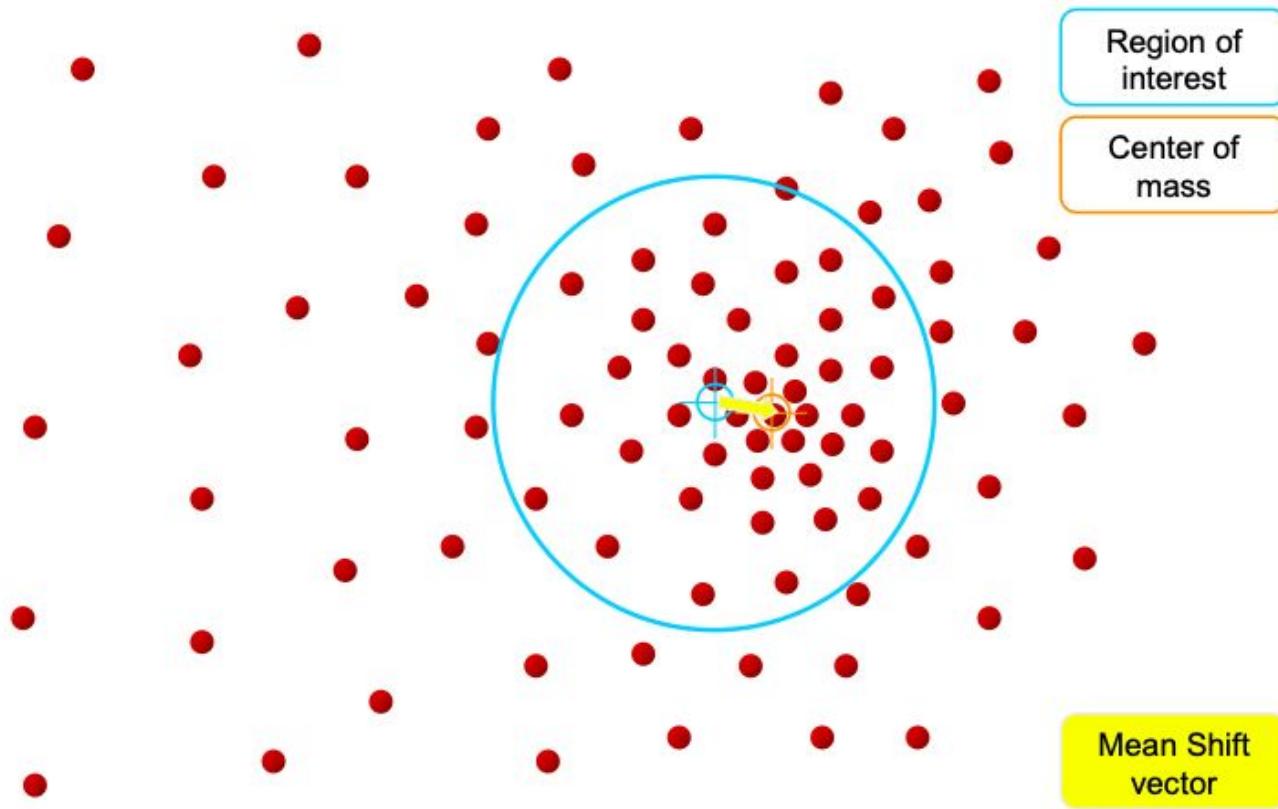
# Mean-shift clustering

---



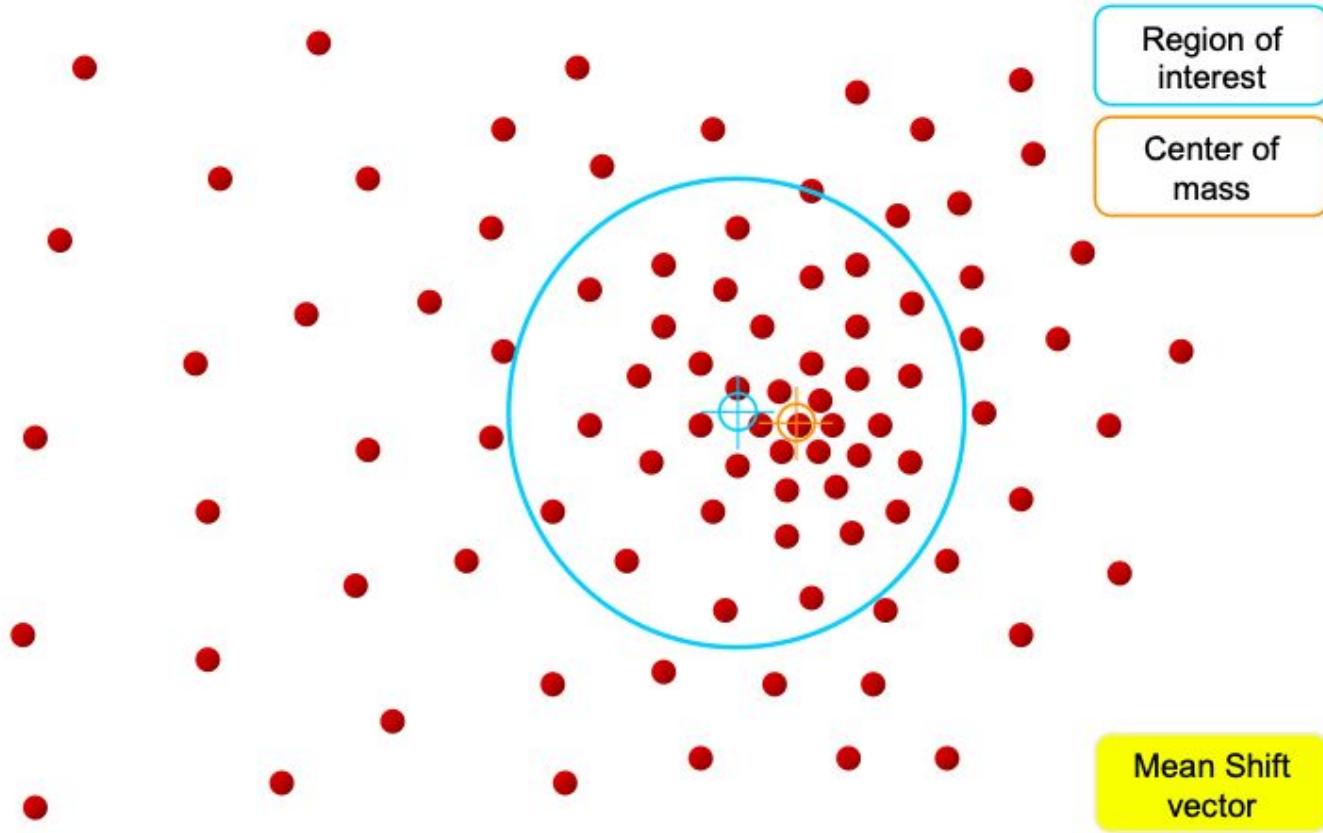
# Mean-shift clustering

---



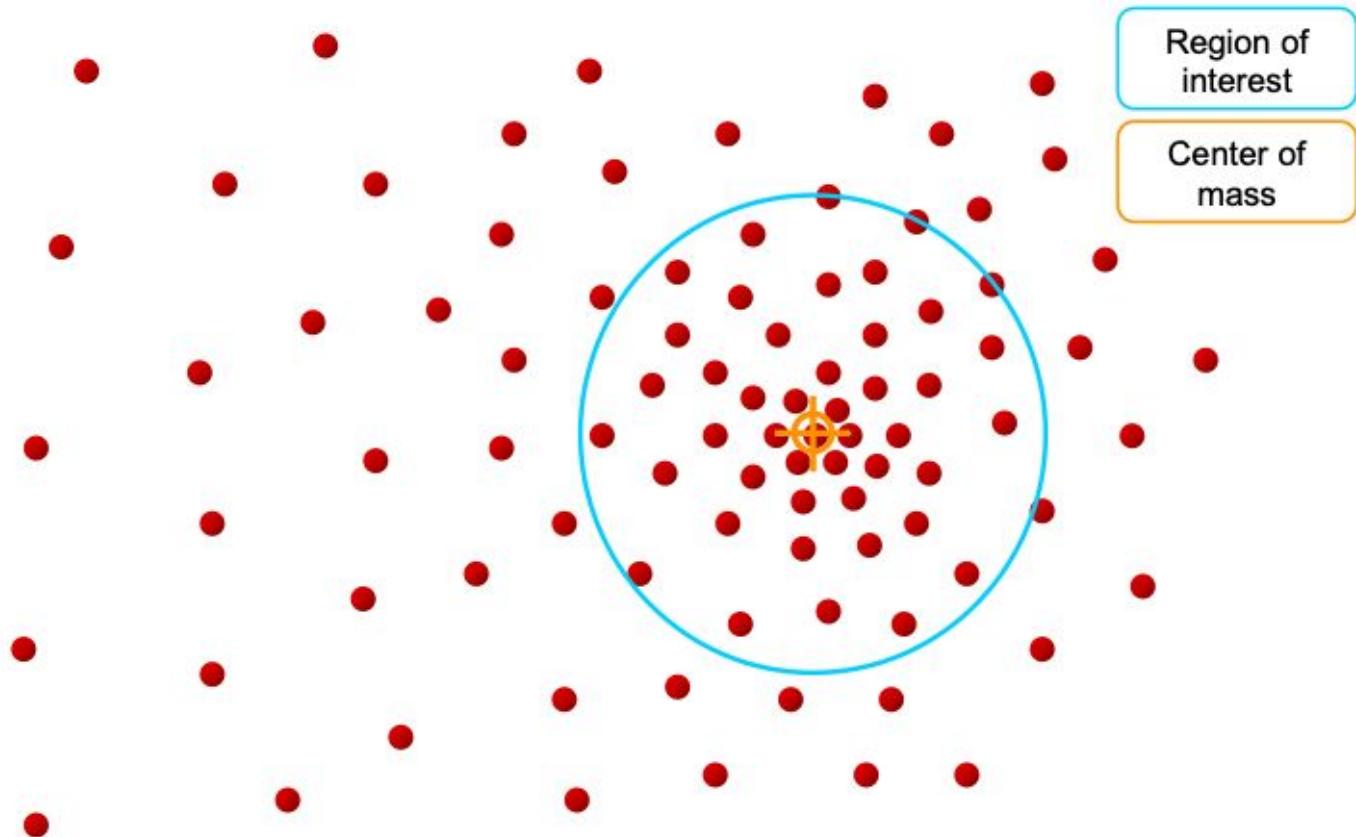
# Mean-shift clustering

---



# Mean-shift clustering

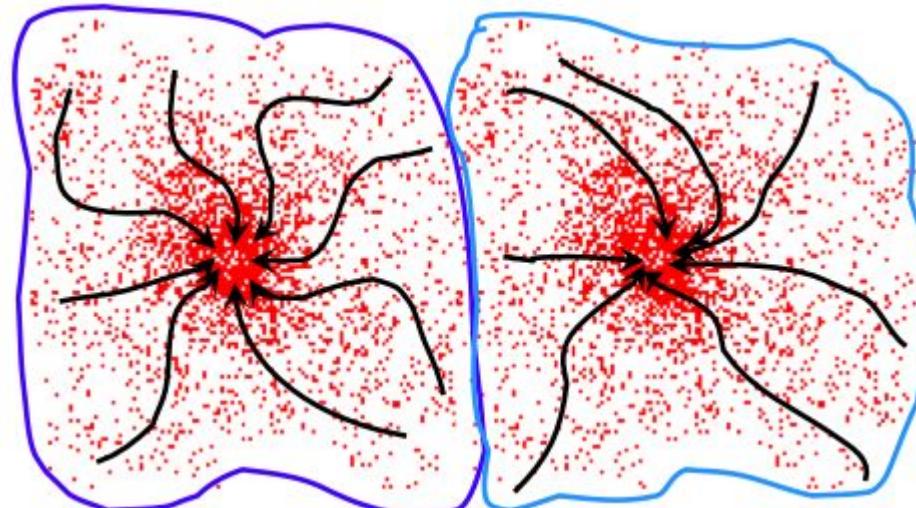
---



# Mean-shift clustering

---

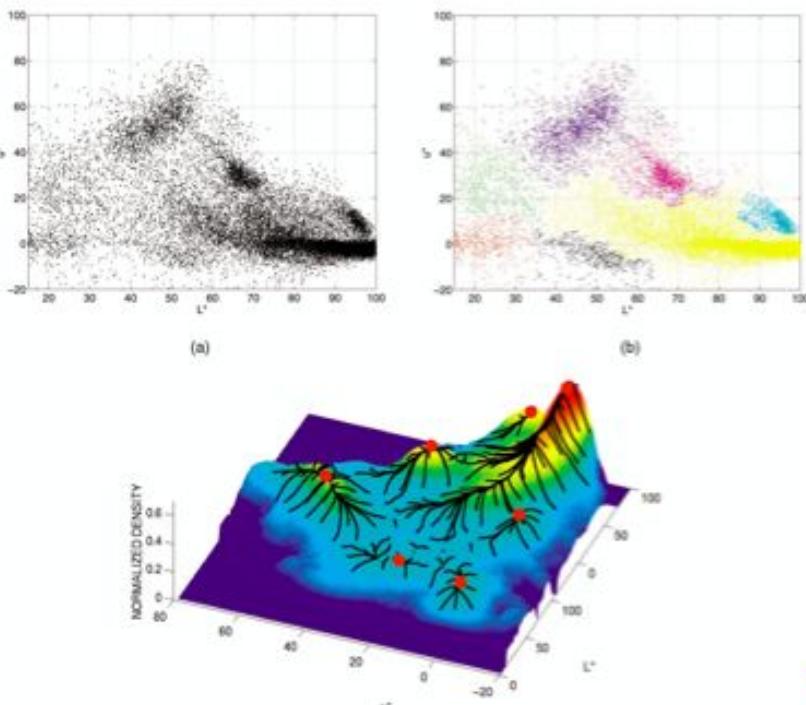
- Кластер: место куда “стекаются” окна
- Множество притяжения: область, в которой все траектории окон ведут в одну точку



# Mean-shift clustering

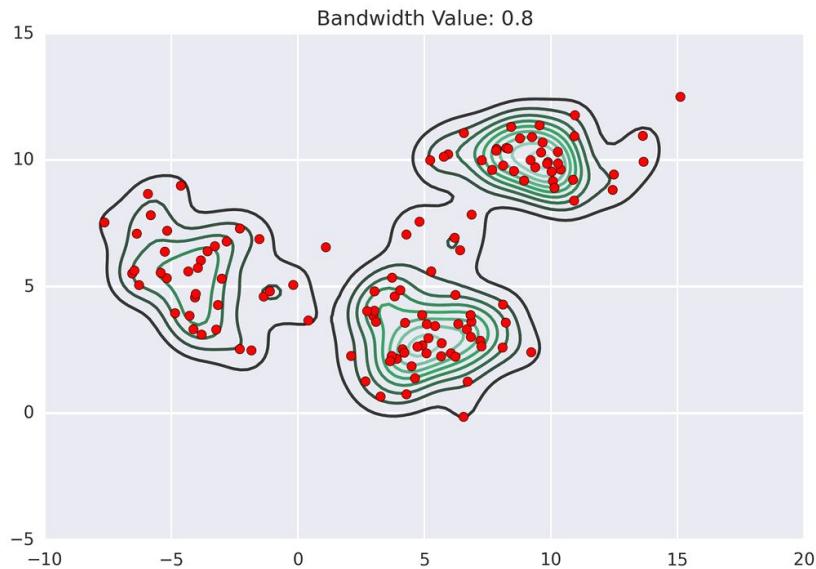
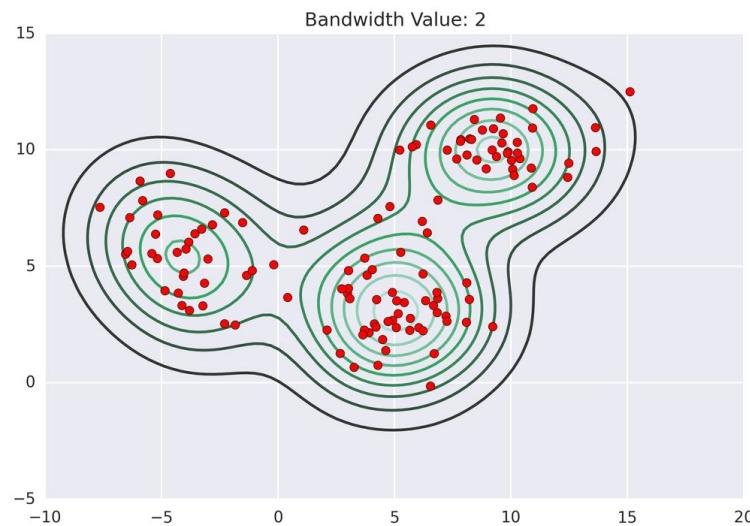
Алгоритм:

1. Определить признаки для каждого из пикселей (цвет, градиент, текстура, ...)
2. Инициализировать окна на месте каждого пикселя
3. Пересчитать центры окон, пока алгоритм не сойдется
4. Смержить окна, которые сошлись в одну точку

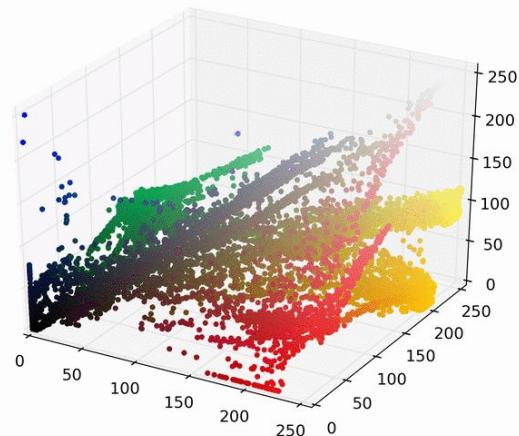
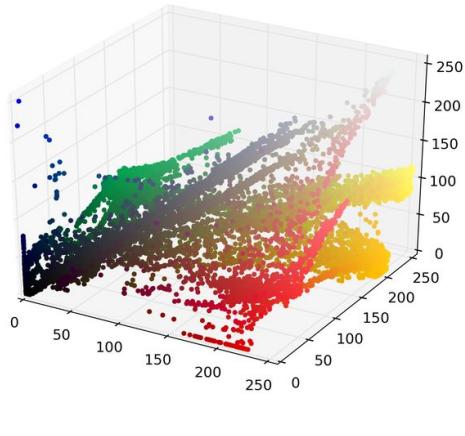
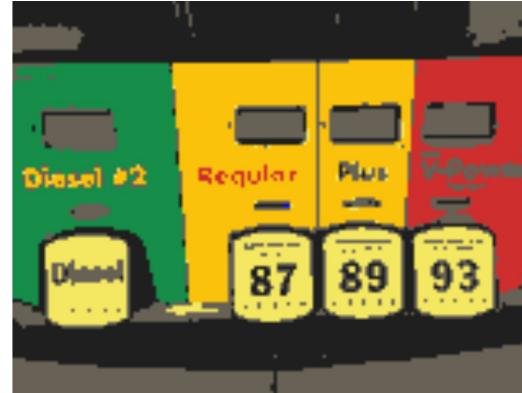


# Mean-shift algorithm

---



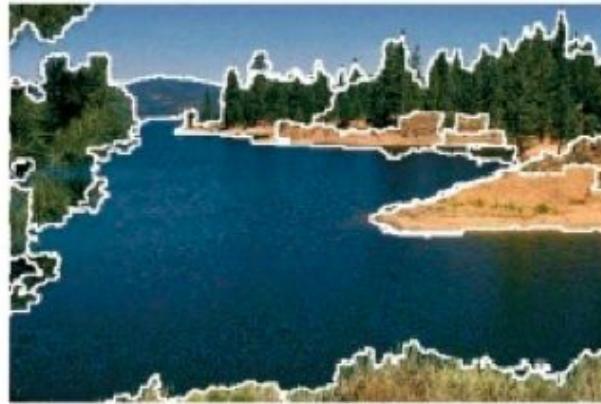
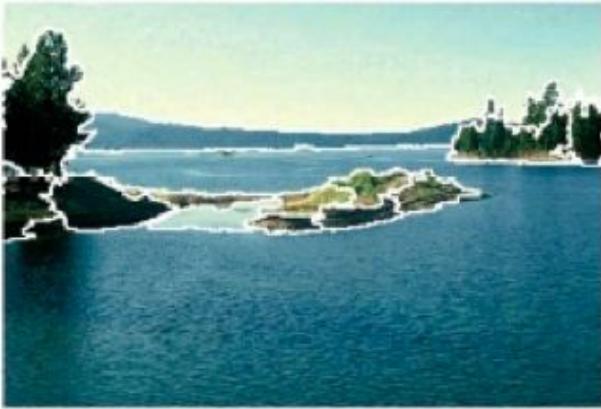
# Mean-shift algorithm



# Примеры работы



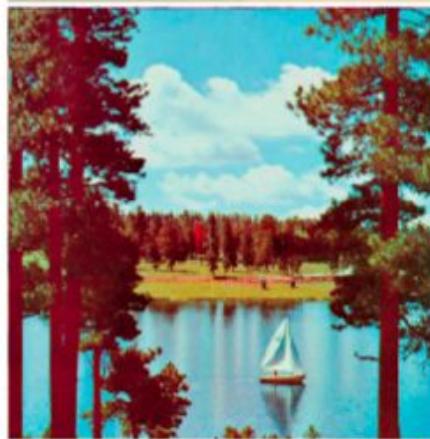
## Примеры работы



## Примеры работы



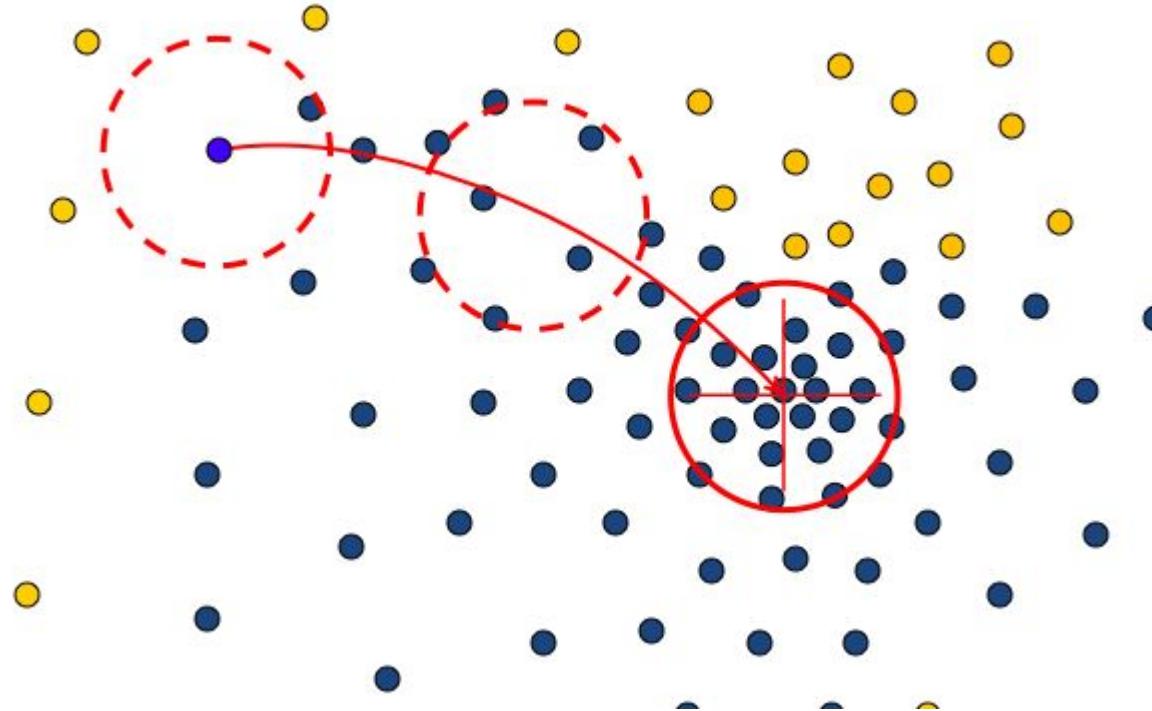
## Примеры работы



# Проблема: вычислительно затратно

---

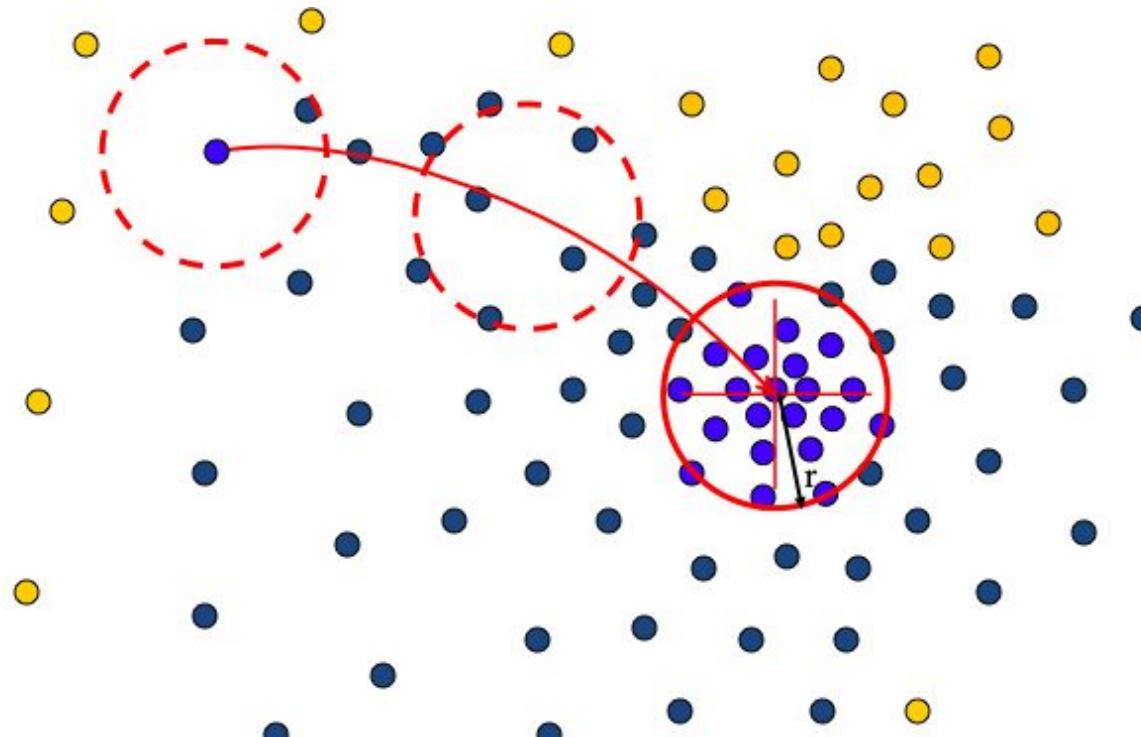
- Нужно рассматривать каждый пиксель...
- Многие вычисления будут избыточными



# Проблема: вычислительно затратно

---

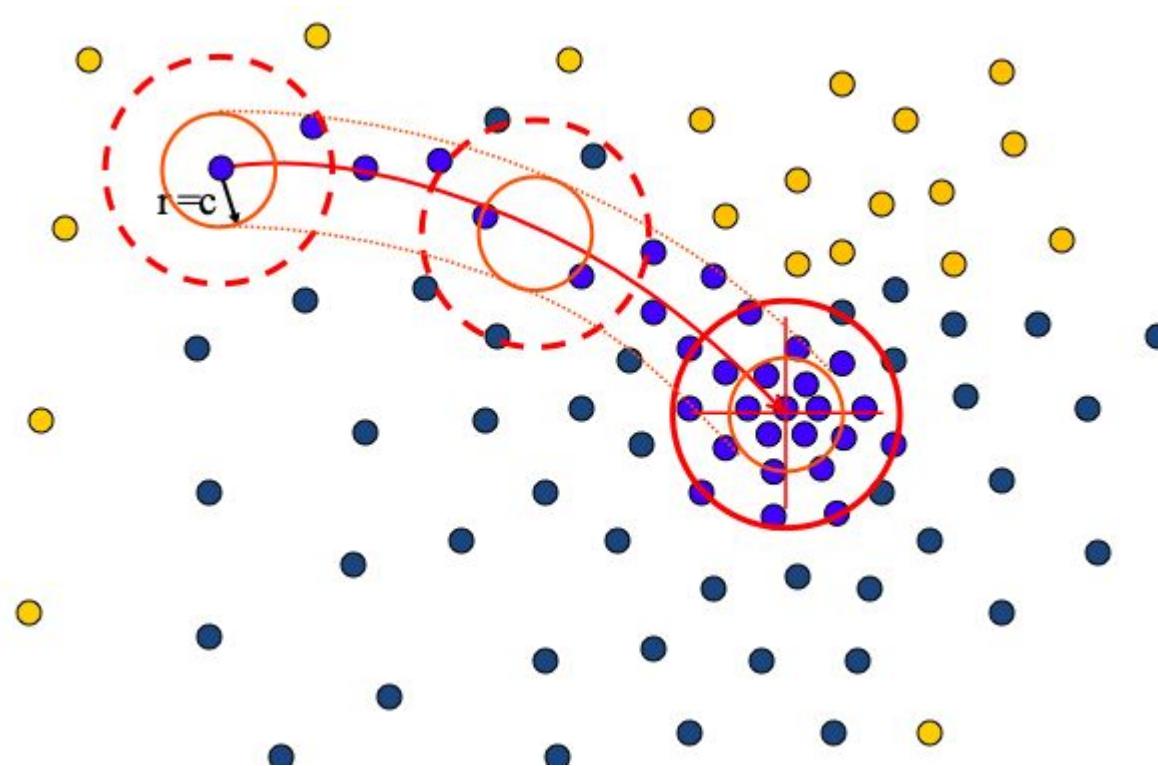
- Когда окно пришло в конечную точку, назначить все точки в радиусе  $r$  к этому же кластеру



# Проблема: вычислительно затратно

---

2. Все точки на расстоянии  $c < r$  от центра окон на в каждый момент прохождения траектории -> уменьшает количество рассматриваемых точек



# Summary: mean-shift

---

- Pros
  - Гибкий алгоритм, подходит для многих задач
  - Подходит для данных любой “формы” (сферические, эллиптические, ...)
  - Всего один параметр (размер окна  $h$ )
    - а еще у  $h$  есть физический смысл ( в отличие от k-means)
  - Устойчиво к выбросам
- Cons
  - Результат сильно зависит от выбора параметра
  - $h$  трудно выбрать
  - Вычислительно затратно
  - Плохо масштабируется с увеличением количества признаков

# Image compression

# Matrix factorization

---

- Есть множество алгоритмов, факторизующих матрицы - раскладывают на сомножители
- Самый популярный/крутой/полезный - SVD
- Представляет любую матрицу  $A$  в виде произведения трех матриц:  $A = U\Sigma V^T$
- Python: `np.linalg.svd(A)`

# SVD

---

- $U, V$  - ортогональные матрицы,  $\Sigma$  диагональная матрица

$$U\Sigma V^T = A$$

$$\begin{matrix} U \\ \left[ \begin{matrix} -.40 & .916 \\ .916 & .40 \end{matrix} \right] \end{matrix} \times \begin{matrix} \Sigma \\ \left[ \begin{matrix} 5.39 & 0 \\ 0 & 3.154 \end{matrix} \right] \end{matrix} \times \begin{matrix} V^T \\ \left[ \begin{matrix} -.05 & .999 \\ .999 & .05 \end{matrix} \right] \end{matrix} = \begin{matrix} A \\ \left[ \begin{matrix} 3 & -2 \\ 1 & 5 \end{matrix} \right] \end{matrix}$$

## SVD

---

В общем случае, если  $A$  -  $M \times N$  матрица, тогда  
 $U$ -  $M \times M$  матрица,  $\Sigma$  -  $M \times N$  матрица,  $V$  -  $N \times N$  матрица

$$U \begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} \times \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} \times V^T \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}$$

# SVD

---

- SVD можно рассматривать как разбиение на 3 преобразования - поворот, масштабирование, и снова поворот
- SVD для матриц изображений также может быть очень полезно
- Будем смотреть на SVD в менее геометрическом ключе

# SVD

---

$$\begin{bmatrix} U \\ -.39 & -.92 \\ -.92 & .39 \end{bmatrix} \times \begin{bmatrix} \Sigma \\ 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} A \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\begin{aligned} & \begin{bmatrix} U\Sigma \\ -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad A_{partial} \\ & + \begin{bmatrix} U\Sigma \\ -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad A_{partial} \\ & = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{aligned}$$

# SVD

---

$$\begin{aligned} & \left[ \begin{matrix} -3.67 \\ -8.8 \end{matrix} \right] \times \begin{matrix} U\Sigma \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix} \end{matrix} \\ & + \left[ \begin{matrix} -3.67 \\ -8.8 \end{matrix} \right] \times \begin{matrix} U\Sigma \\ \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \end{matrix} \times \begin{matrix} V^T \\ \begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix} \end{matrix} \\ & = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \end{aligned}$$

Каждая компонента (*i*-ый столбец  $U$ ) $^*$ (*i*-ое значение Sigma) $^*$ (*i*-ая строка  $V^T$ ) формирует финальную матрицу  $A$

# SVD

---

$$\begin{aligned} & \left[ \begin{matrix} -3.67 \\ -8.8 \end{matrix} \quad \begin{matrix} -.71 \\ .30 \end{matrix} \quad 0 \right] \times \left[ \begin{matrix} V^T \\ \cdot .42 & \cdot -.57 & \cdot -.70 \\ \cdot .81 & \cdot .11 & \cdot -.58 \\ \cdot .41 & \cdot -.82 & \cdot .41 \end{matrix} \right] \quad \left[ \begin{matrix} A_{partial} \\ 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{matrix} \right] \\ + & \left[ \begin{matrix} -3.67 \\ -8.8 \end{matrix} \quad \begin{matrix} -.71 \\ .30 \end{matrix} \quad 0 \right] \times \left[ \begin{matrix} V^T \\ \cdot -.42 & \cdot -.57 & \cdot -.70 \\ \cdot .81 & \cdot .11 & \cdot -.58 \\ \cdot .41 & \cdot -.82 & \cdot .41 \end{matrix} \right] \quad \left[ \begin{matrix} A_{partial} \\ -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{matrix} \right] \quad \left[ \begin{matrix} A \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \right] \\ = & \left[ \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \right] \end{aligned}$$

- Мы восстанавливаем  $A$  как линейную комбинацию столбцов  $U$
- Используя все столбцы  $U$ , мы полностью восстановим исходную матрицу  $A$
- Но мы можем забыть и использовать только первые несколько столбцов, и все равно получим довольно близкий результат

# SVD

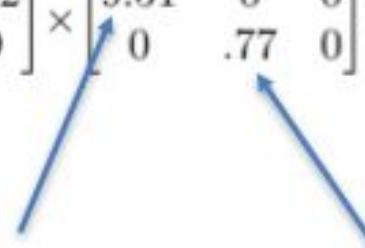
---

$$\begin{aligned} & \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad A_{partial} \\ + & \begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} V^T \\ -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} \quad A_{partial} \\ = & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A \end{aligned}$$

- Будем называть эти первые столбцы матрицы U **главными компонентами**
- Они содержат основные “паттерны” исходной матрицы A
- Строки матрицы  $V^T$  показывают, какие коэффициенты нужно ставить перед главными компонентами, чтобы восстановить исходную матрицу

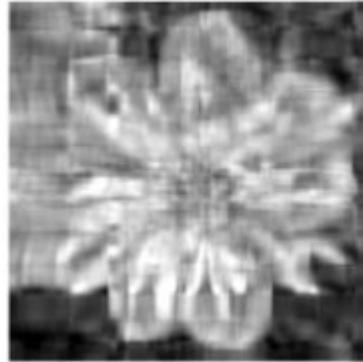
# SVD

---

$$\begin{matrix} U \\ \left[ \begin{matrix} -.39 & -.92 \\ -.92 & .39 \end{matrix} \right] \end{matrix} \times \begin{matrix} \Sigma \\ \left[ \begin{matrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{matrix} \right] \end{matrix} \times \begin{matrix} V^T \\ \left[ \begin{matrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{matrix} \right] \end{matrix} = \begin{matrix} A \\ \left[ \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \right] \end{matrix}$$


# SVD for image compression

---



- Используя только 10 из 300 главных компонент, можно добиться достаточно хорошего качества
- SVD можно использовать для сжатия изображений

# SVD

---

$$\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = A_{partial}$$

- Столбцы U - главные компоненты данных, содержат основные паттерны, их линейная комбинация позволяет восстановить столбцы исходной матрицы
- Построим матрицу, где каждый столбец - отдельная “точка”
- Факторизовать SVD, посмотреть на первые столбцы U, чтобы увидеть паттерны, общие среди столбцов
- Это называется метод главных компонент (PCA)

# SVD

---

$$\begin{bmatrix} -3.67 & -.71 & 0 \\ -.88 & .30 & 0 \end{bmatrix} \times \begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix} = \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}$$

- Зачастую, сырые данные избыточны, и содержат много общих паттернов
- РСА позволяет представить данные как веса векторов главных компонент
- С помощью такого представления, можно получить явную разницу между различными изображений
- Позволяет делать многие алгоритмы машинного обучения гораздо более эффективными

# Image compression

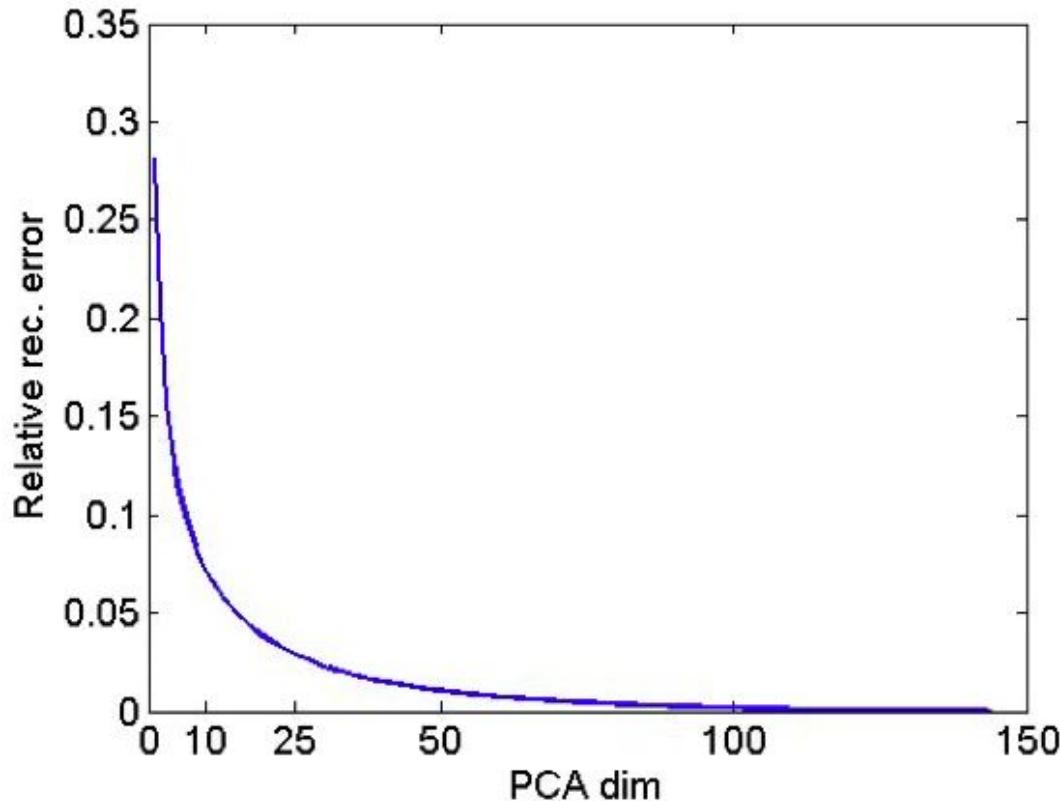
---



- Разбить оригинальное изображение 372x492 на патчи
  - Каждый патч - размера 12x12, патчи не пересекаются
- Будем рассматривать патч, как вектор размерности 144D
- Всего - около 1300 патчей

# Image compression

---



# Image compression (60/144)

---



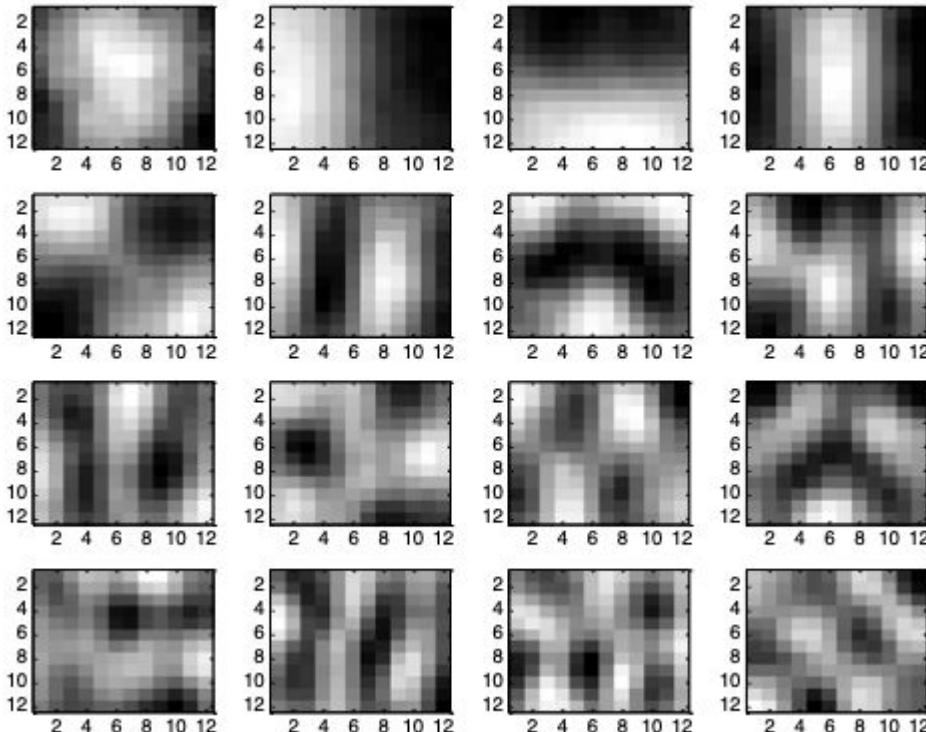
## Image compression (16/144)

---



# Image compression (16/144)

---



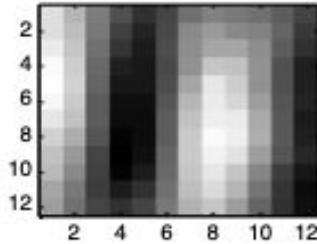
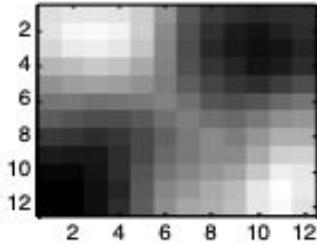
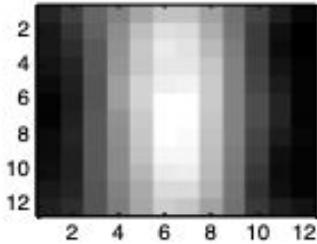
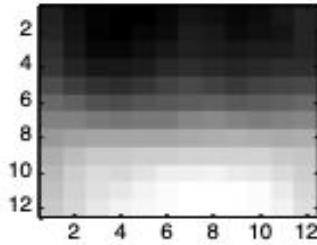
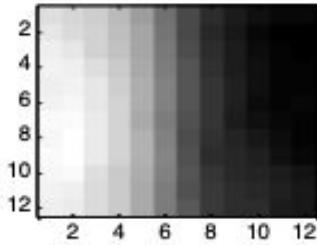
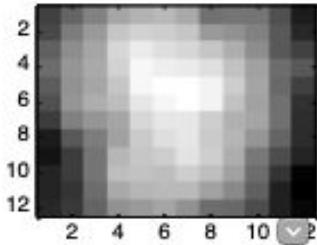
## Image compression (6/144)

---



## Image compression (6/144)

---



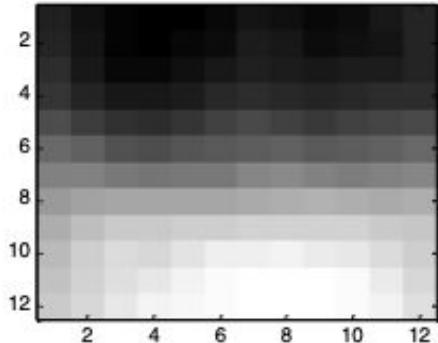
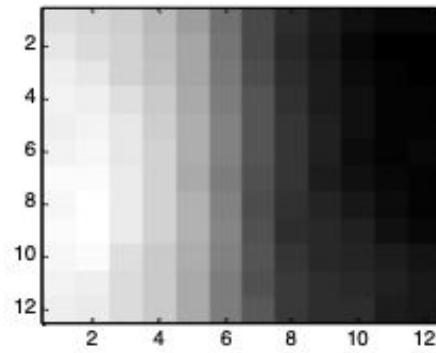
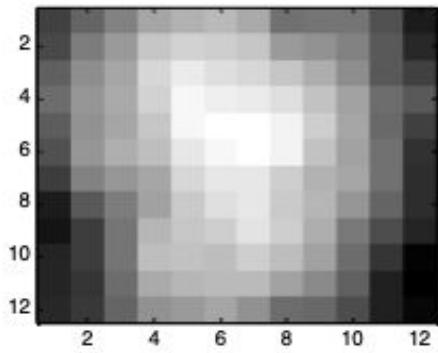
## Image compression (3/144)

---



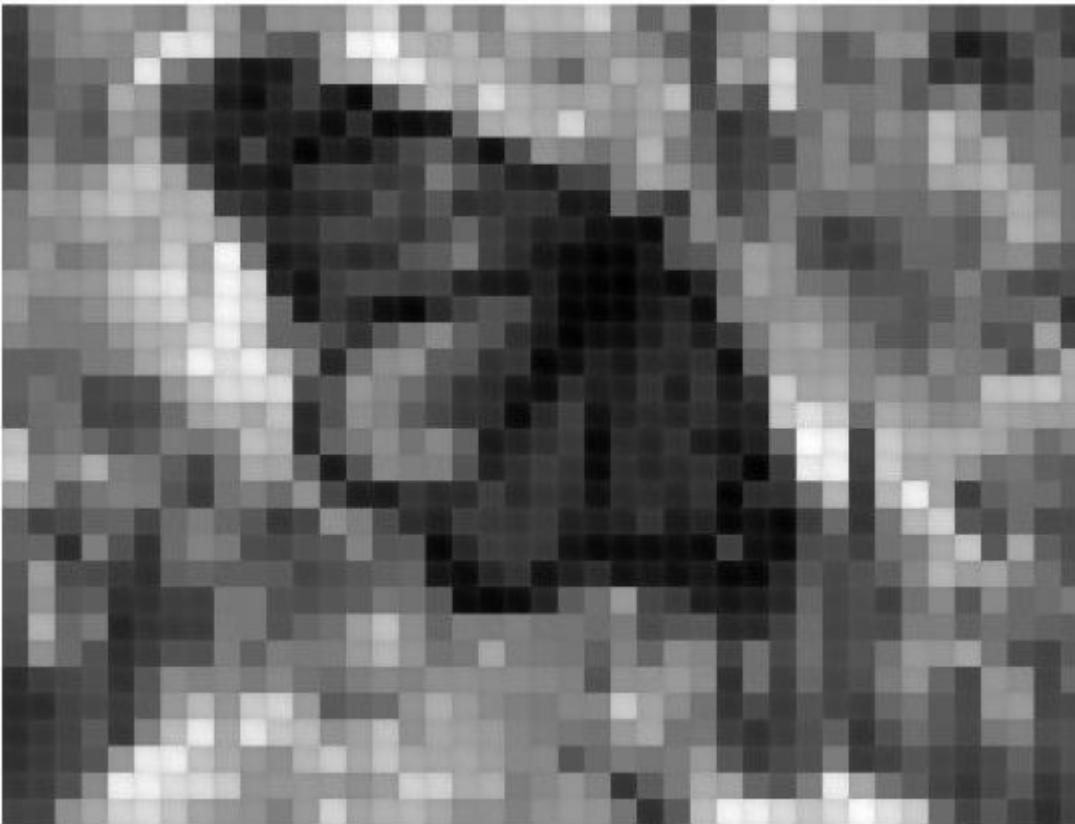
## Image compression (3/144)

---



# Image compression (1/144)

---

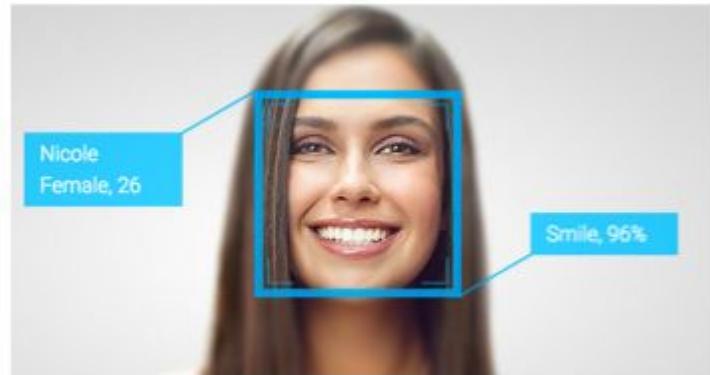
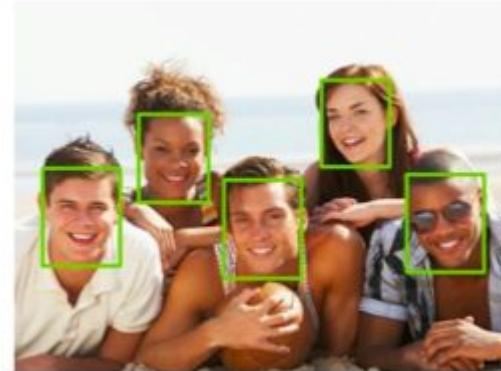


# Face recognition

# Face recognition

---

- Цифровая фотография
- Наблюдение
- Верификация
- Безопасность
- Memoji
- Организация альбомов
- ...

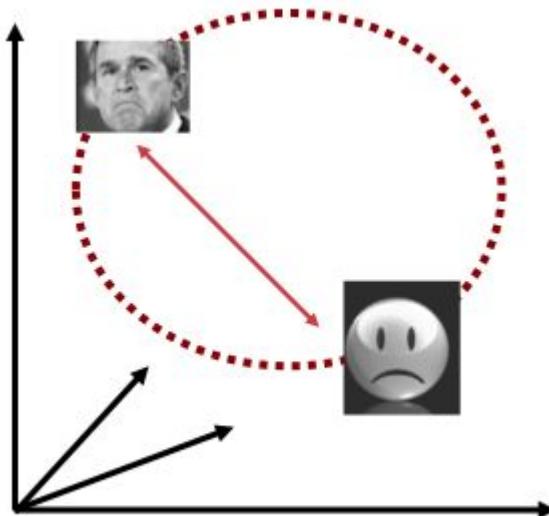


# Пространство лиц (!)

---

- Изображение - это точка в пространстве высокой размерности

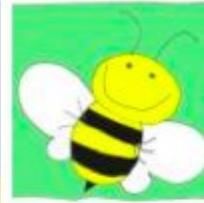
- Черно белое изображение размера NxM - это точка в пространстве  $R^{NM}$
- Например, 100x100 изображение = 10000 размерность



# Пространство лиц (!)

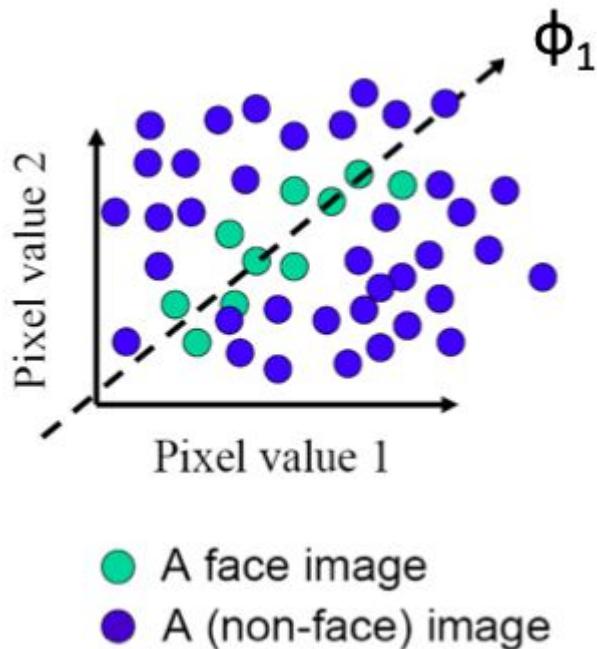
---

- Изображения 100x100 содержат не только изображения, но и много всего другого



# Пространство лиц (!)

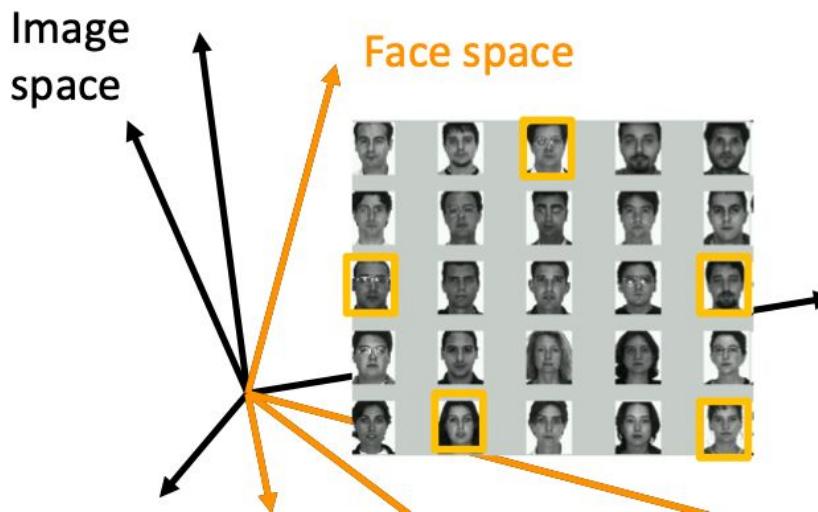
---



- Изображение - это точка в пространстве высокой размерности
  - Черно белое изображение размера  $N \times M$  - это точка в пространстве  $R^{NM}$
  - Например, 100x100 изображение = 10000 размерность
- Однако, лишь относительно небольшая часть точек в этом пространстве действительно соответствует лицам
- Мы хотим найти подпространство лиц!

# Пространство лиц (!)

---



- Найти подпространство размерности  $n$ , такое что проекция точек датасета, на это подпространство, будет иметь наибольшую дисперсию

## Пространство лиц (!)

---

- Проецируем на подпространство меньшей размерности, которое содержит ключевые характеристики
- **PCA**
- Сравнивать два изображения с лицами, проецируя их на пространство меньшей размерности и измеряя метрику расстояния между ними

# Eigenfaces

---

Идея:

- предполагаем, что большинство изображений с лицами лежат в пространстве более низкой размерности определяемом  $k$  ( $k \ll d$ ) направлениями максимальной дисперсии
- используем РСА, чтобы определить собственные вектора (“eigenfaces”) которое задают это подпространство
- Представляем все изображения с лицами из датасета как линейную комбинацию eigenfaces

# Eigenfaces

---

Training images:  $\mathbf{x}_1, \dots, \mathbf{x}_N$

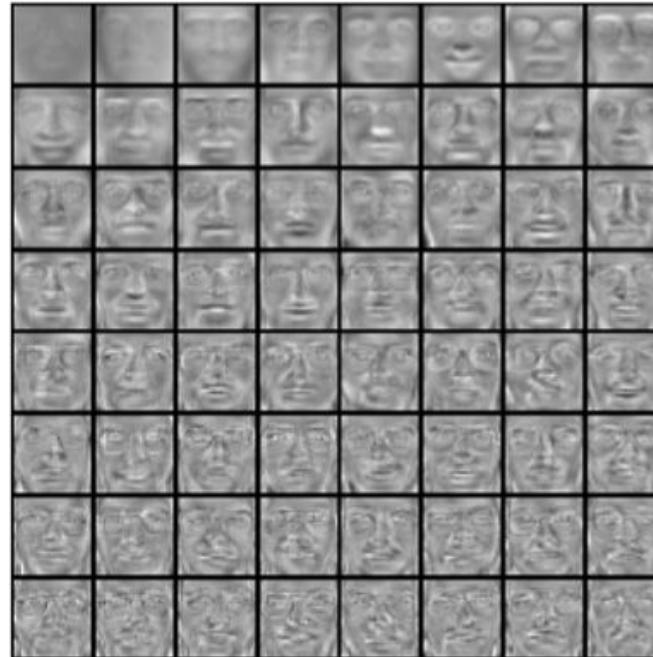


# Eigenfaces

---

Top eigenvectors:  $\phi_1, \dots, \phi_k$

Mean:  $\mu$



# Eigenface algorithm (training)

1. Преобразовать изображения (матрицы) в точки (очень длинные векторы)



2. Подсчитать “среднее” лицо

$$\mu = \frac{1}{N} \sum x_i$$

3. Центрировать данные

$$\begin{aligned} X_c &= \begin{bmatrix} | & | \\ x_1 & \dots & x_n \\ | & | \end{bmatrix} - \begin{bmatrix} | & | \\ \mu & \dots & \mu \\ | & | \end{bmatrix} \\ &= X - \mu \mathbf{l}^T = X - \frac{1}{n} X \mathbf{1} \mathbf{1}^T = X \left( I - \frac{1}{n} \mathbf{1} \mathbf{1}^T \right) \end{aligned}$$

# Eigenface algorithm

4. Вычислить матрицу ковариации

$$\Sigma = \frac{1}{n} \begin{bmatrix} | & & | \\ x_1^c & \dots & x_n^c \\ | & & | \end{bmatrix} \begin{bmatrix} - & x_1^c & - \\ \vdots & \ddots & \vdots \\ - & x_n^c & - \end{bmatrix} = \frac{1}{n} X_c X_c^T$$

5. Найти собственные векторы ковариационной матрицы (SVD)

6. Найти проекции каждого изображения из обучающей выборки на эти собственные вектора

$$x_i \rightarrow (x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \dots, x_i^c \cdot \phi_K) \equiv (a_1, a_2, \dots, a_K)$$

7. Восстановить изначальное изображение

$$x_i \approx \mu + a_1 \phi_1 + a_2 \phi_2 + \dots + a_K \phi_K$$

# Eigenface algorithm



6. Найти проекции каждого изображения из обучающей выборки на эти собственные вектора

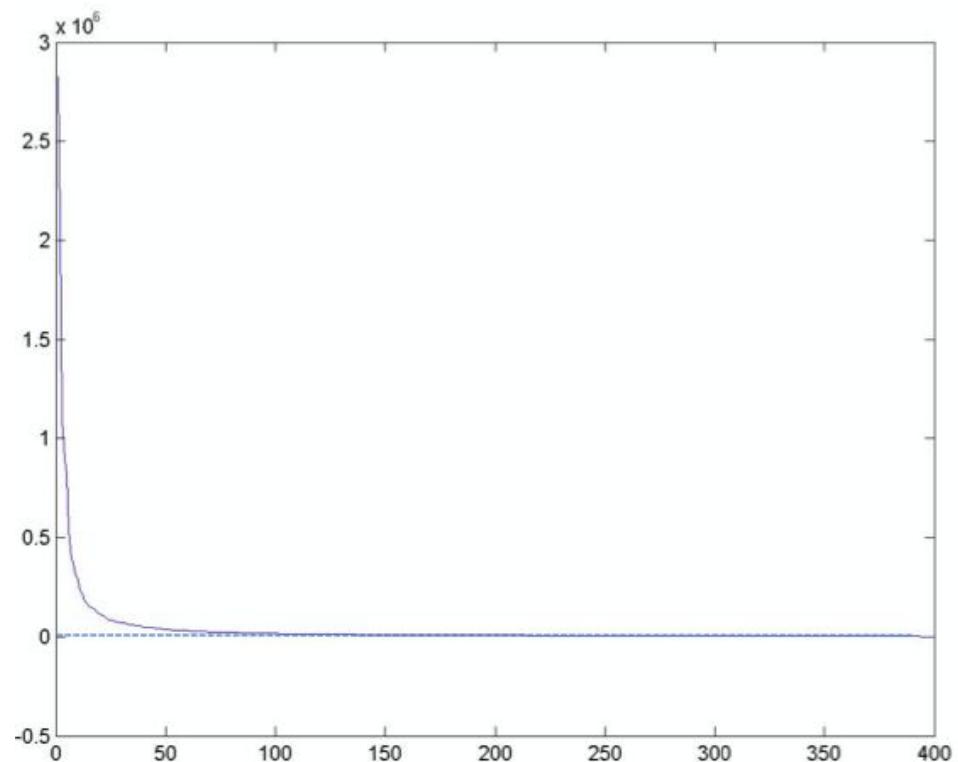
$$x_i \rightarrow (x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \dots, x_i^c \cdot \phi_K) = (a_1, a_2, \dots, a_K)$$

7. Восстановить изначальное изображение

$$x_i \approx \mu + a_1\phi_1 + a_2\phi_2 + \dots + a_K\phi_K$$

# Eigenface (variance)

---



# Eigenfaces

---



Выбирая К наилучших eigenfaces -> уменьшаем размерность

Маленькое количество eigenfaces приводит к слишком большой потери информации, а следовательно мы хуже сможем различать лица

## Eigenface algorithm (testing)

---

1. Берем тестовое изображение с лицом  $t$
2. Проецируем его в найденное подпространство и вычисляем коэффициенты проекции

$$t \rightarrow ((t - \mu) \cdot \phi_1, (t - \mu) \cdot \phi_2, \dots, (t - \mu) \cdot \phi_K) \equiv (w_1, w_2, \dots, w_K)$$

3. Сравниваем проекцию  $w$  со всеми  $N$  проекциями тренировочных изображений
  - a. Метрика сравнения: Euclidean
  - b. Классификация: k-NN

# Eigenfaces

---

- Алгоритм очень требователен к данным
  - Все лица центрированы
  - Однакового размера
  - Ограниченный угол поворота головы
    - Альтернатива: получить РСА для каждого из возможных углов
    - Использовать тот, на котором достигается наименьшая ошибка
- Методы сжатия - полностью unsupervised
- (иногда это хорошо!)
- Не учитывает, что лица - это 3D объекты
- Нет никакого механизма сохранения распределения классов

# Eigenfaces

---

- Pros:
  - Не итеративный
  - Достигает глобального оптимума
- Cons:
  - Плохо интерпретируется
  - Не гибкий
- Ограничения
  - PCA - отлично подходит для восстановления из пространства низкой размерности (сжатие, разжатие), но плохо подходит для векторного описания

# Bag of visual words

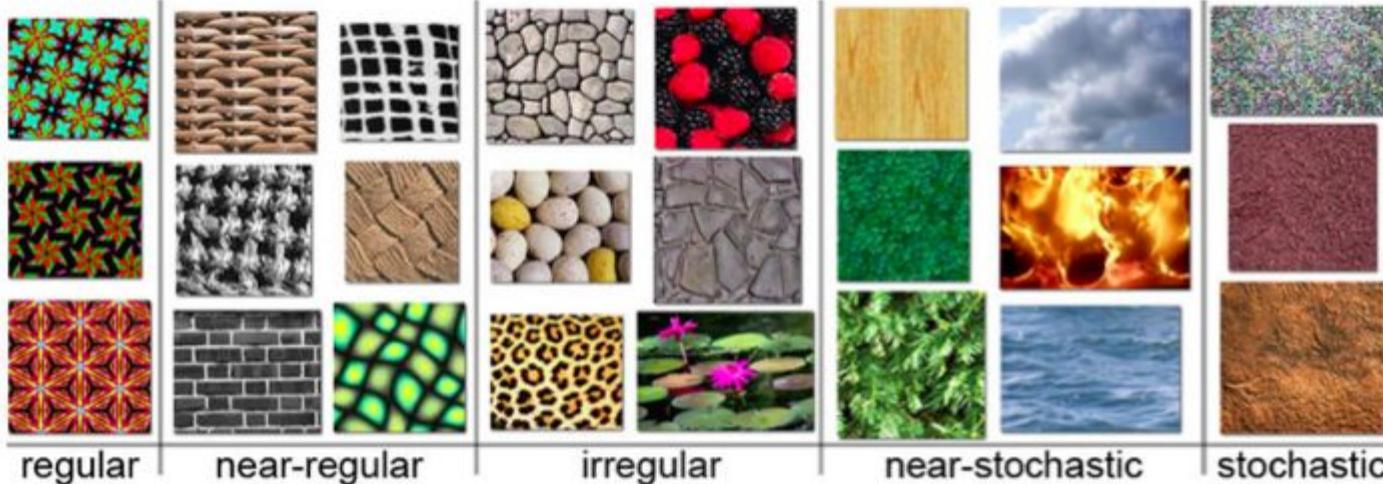
# BoW

---



# Текстуры

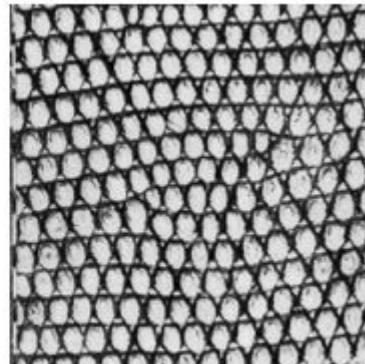
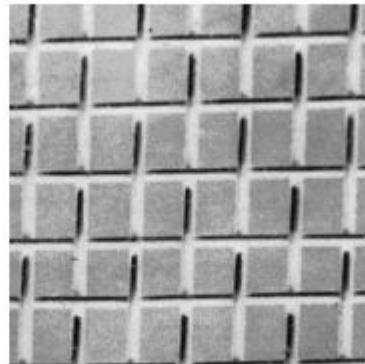
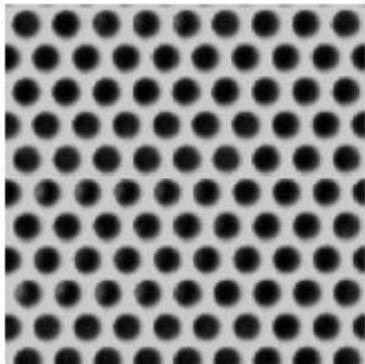
---



# Текстуры

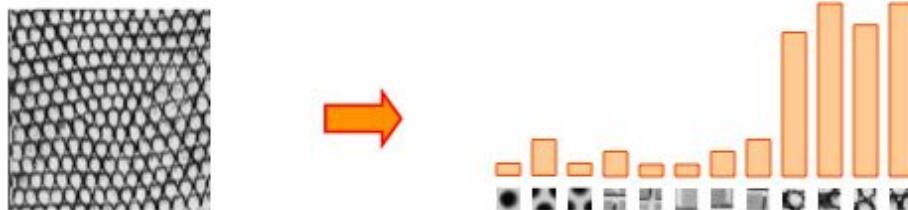
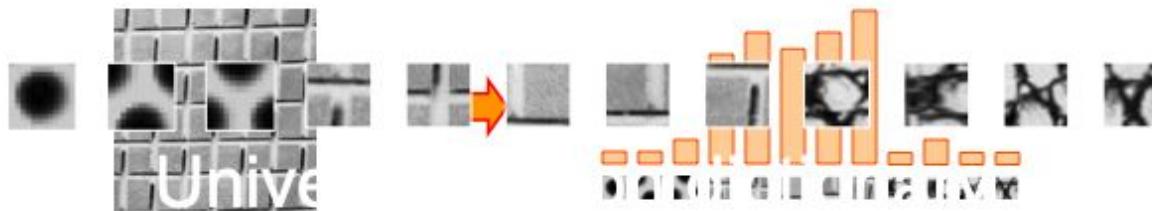
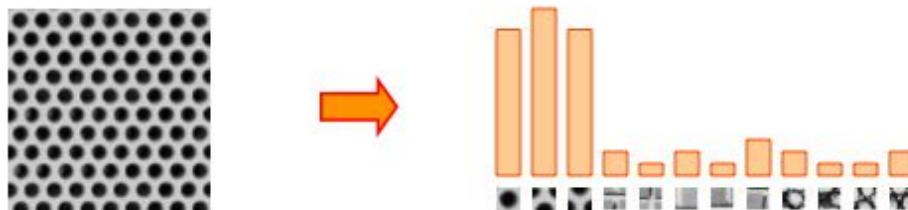
---

Текстуры характеризуются повторяющимися элементами (**текстонами**) и их частотой



# Текстуры

---



## BoW models

---

Представление текстового документа, не учитывая  
порядок слов, а только частоту слов в документе

## BoW models

Представление текстового документа, не учитывющее порядок слов, а только частоту слов в документе



# BoW models

Представление текстового документа, не учитывающее порядок слов, а только частоту слов в документе



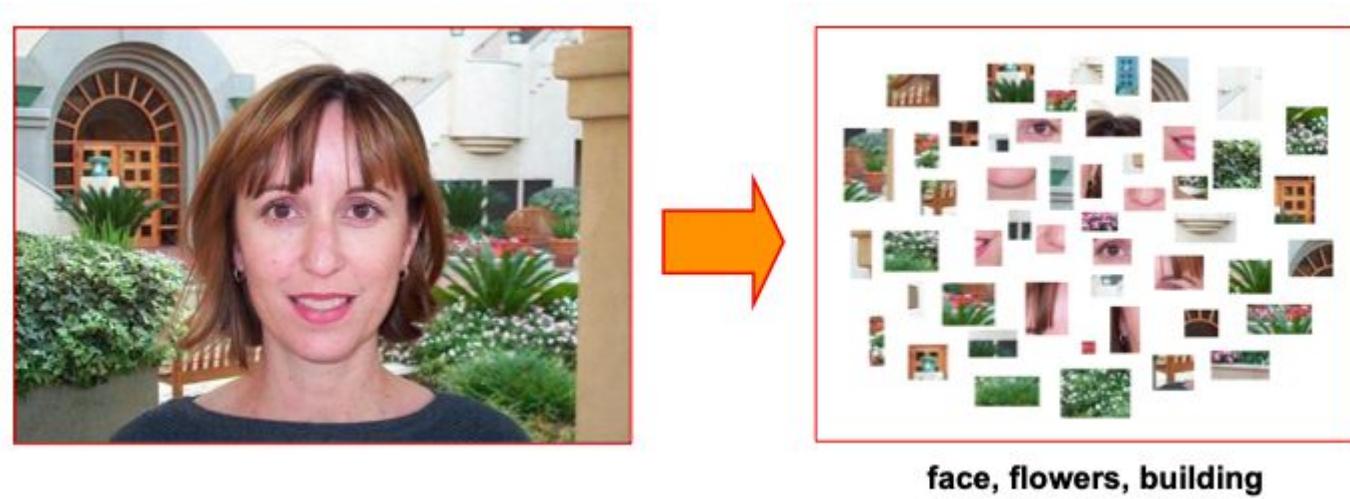
# BoW models

Представление текстового документа, не учитывающее  
порядок слов, а только частоту слов в документе



# Bag of features

---



Работает довольно хорошо для задач классификации, определения, что вообще изображено на картинке

# Bag of features

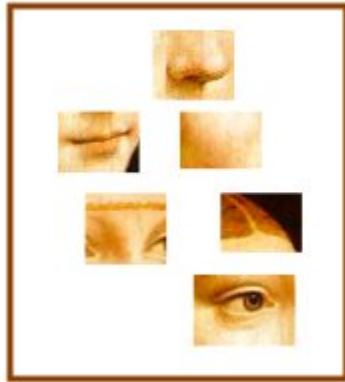
---

- Берем много изображений, извлекаем из них кучу признаков (например, SIFTом), и строим “визуальный словарь” - список общих фич
- С новым изображением - извлекаем фичи, строим гистограмму частот - для каждой фичи, находим наиближайшее слово в словаре

# Bag of features

---

## 1. Извлекаем фичи



# Bag of features

---

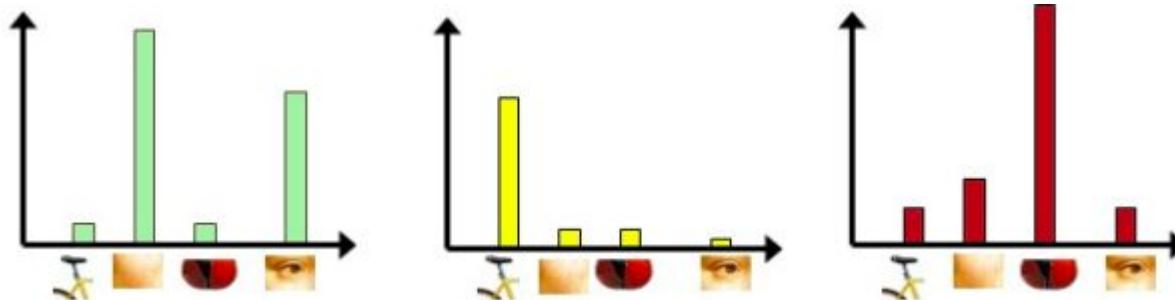
1. Извлекаем фичи
2. Строим “визуальный словарь”



# Bag of features

---

1. Извлекаем фичи
2. Строим “визуальный словарь”
3. Для каждой фичи находим ближайшее “слово” из словаря
4. Строим гистограмму частот “слов” для словаря



# Bag of features

---

## 1. Извлекаем фичи

- Делаем сетку на изображении
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005

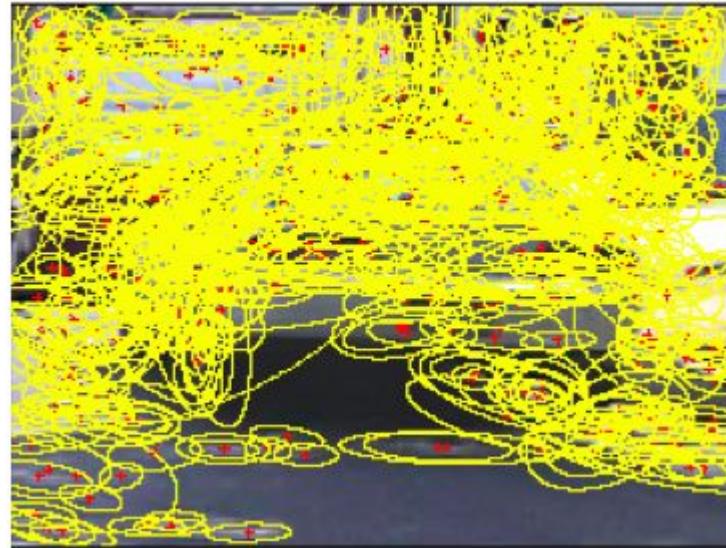


# Bag of features

---

## 1. Извлекаем фичи

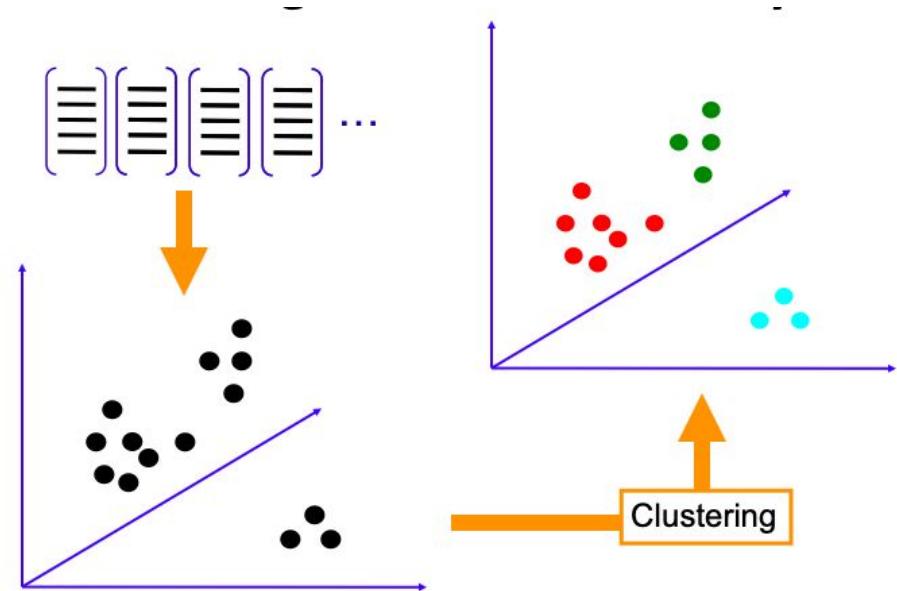
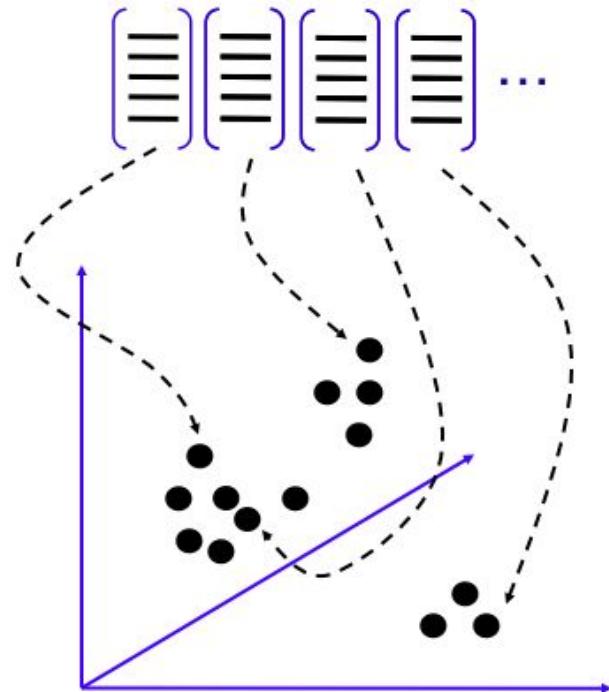
- Делаем сетку на изображении
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Детекторы ключевых точек
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005



# Bag of features

---

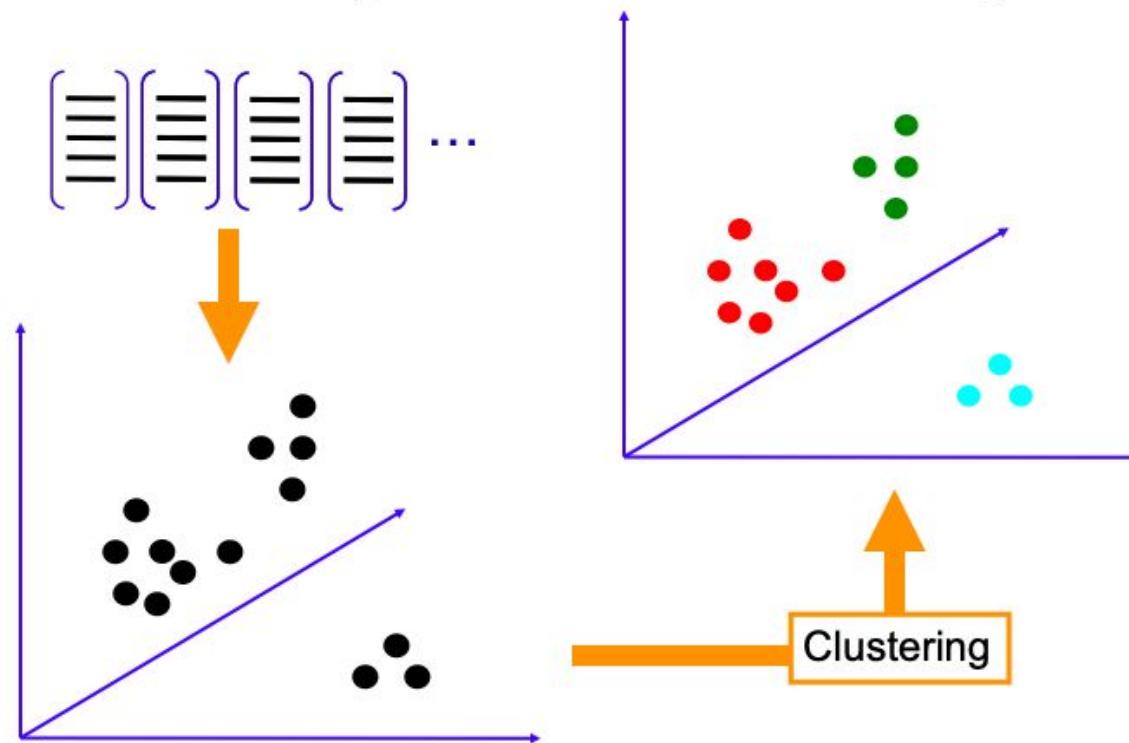
## 2. Строим визуальный словарь



# Bag of features

---

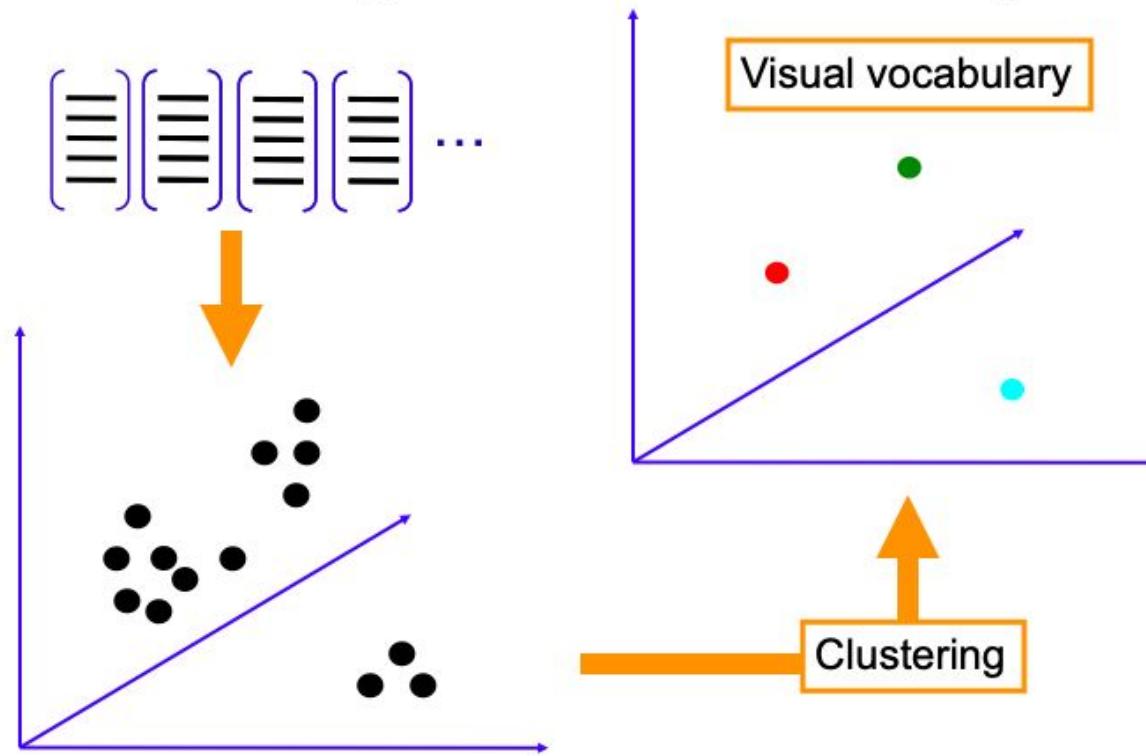
## 2. Строим визуальный словарь



# Bag of features

---

## 2. Строим визуальный словарь



# K-means

---

- Хотим минимизировать сумму квадратов расстояний между точками  $x_i$  и ближайшими к ним центров кластеров  $m_k$

$$D(X, M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (x_i - m_k)^2$$

- Алгоритм:
- Рандомно инициализируем К центров кластеров
- Итерироваться пока не сойдется
  - Назначаем каждую точку к ближайшему центру
  - Пересчитываем каждый центр кластера, как среднее его точек

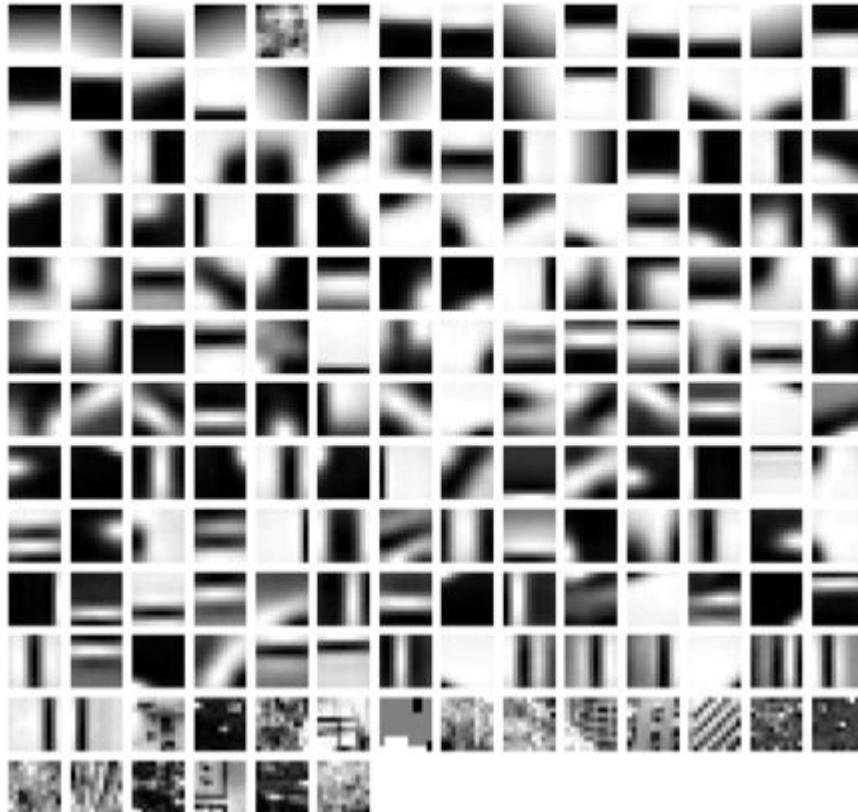
# K-means

---

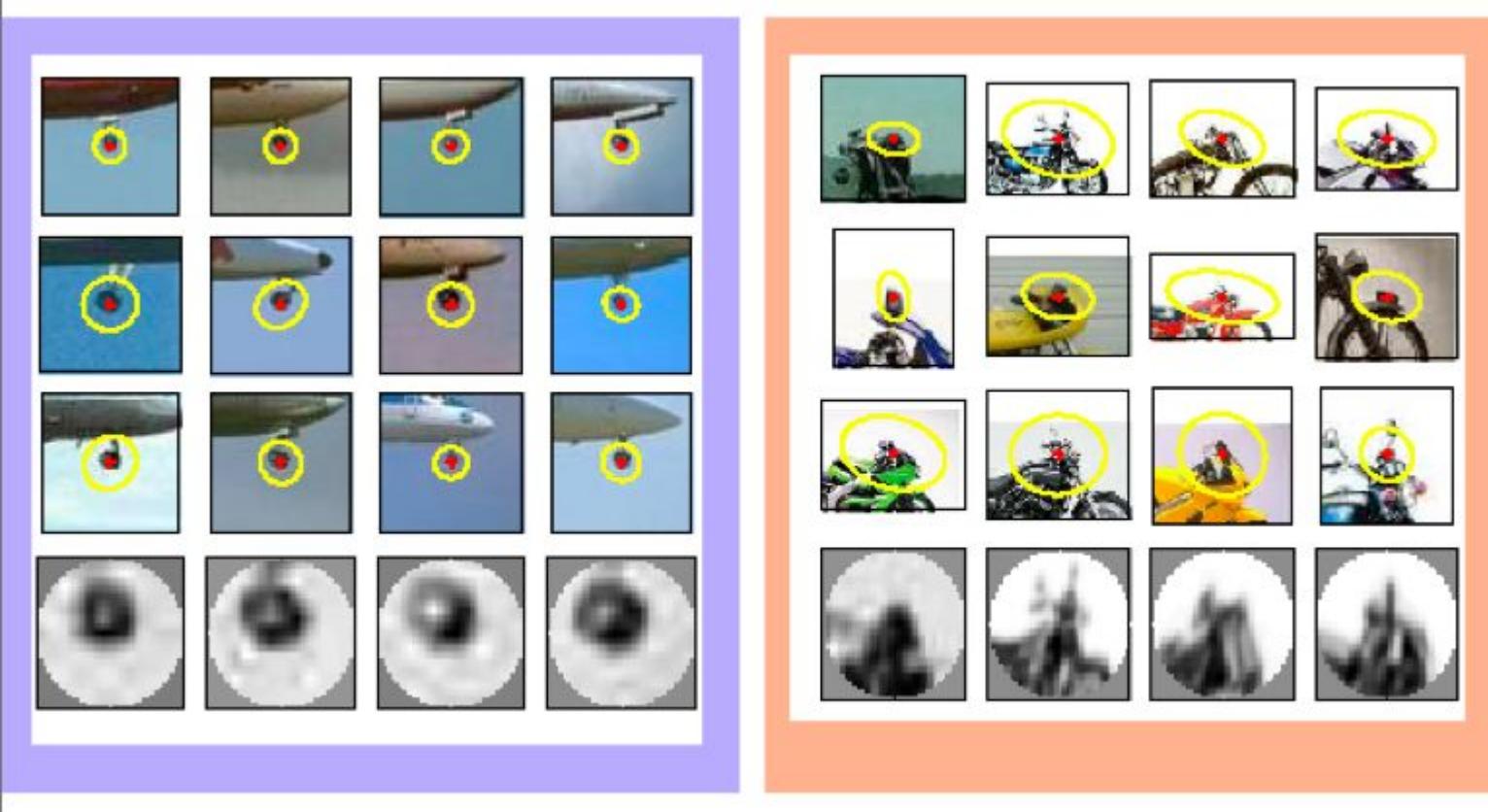
- Кластеризация - хороший метод для построения визуального словаря:
  - Unsupervised
  - Каждый центр кластера, полученный через k-means, становится “визуальным словом” в словаре
  - Словарь строится на обучающей выборке
  - Если обучающая выборка, довольно репрезентативна, то словарь становится универсальным
- Словарь используется для описания каждой фичи
  - Берет feature vector, смотрит на ближайшее к нему слово из словаря, теперь feature vector - просто индекс этого слова

# Visual vocabulary

---



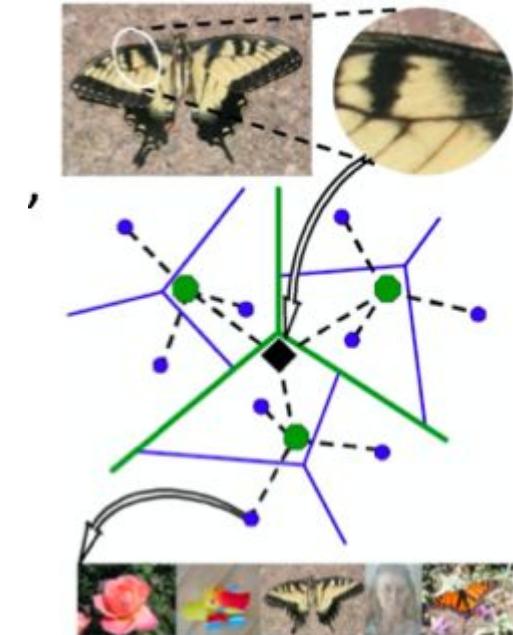
# Пример визуальных слов



# BoW: issues

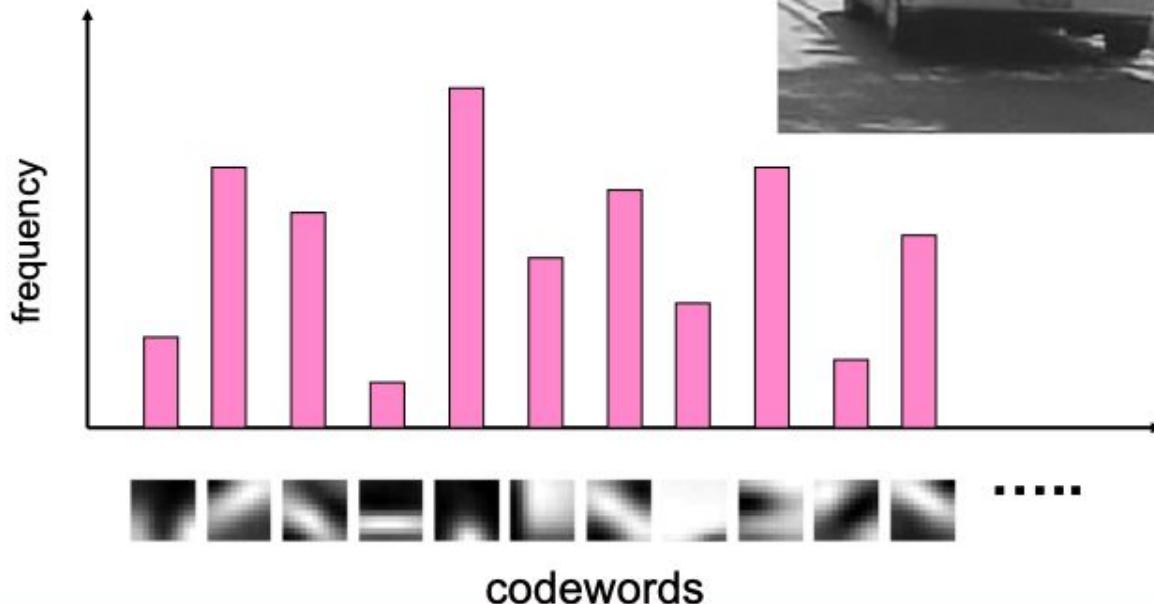
---

- Как выбрать размер словаря?
  - Слишком маленький: недостаточно репрезентативен
  - Слишком большой: переобучение
- Вычислительная сложность
  - Трудно найти “ближайшего” соседа, если соседей 100\_000



# Image representation

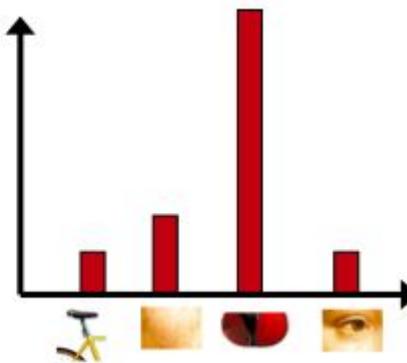
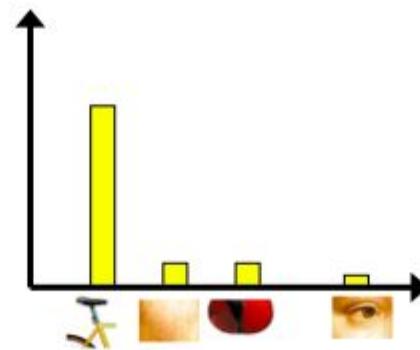
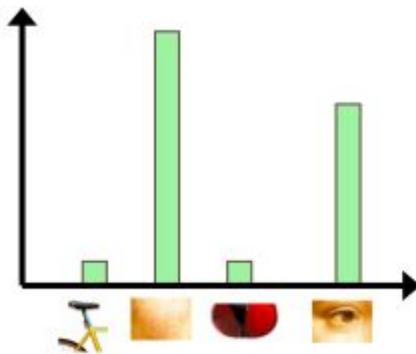
---



# Image representation

---

- Теперь у нас есть векторное описание каждого изображения (частота слов из словаря). Что с ними делать дальше?



# BoW representation

---

- Использовать как векторное описание для стандартных классификаторов (например, k-NN, SVM)
- Кластеризовать BoW векторы
  - Находим визуальные темы в изображении!

# Large-scale image search

---



BoW модели полезны в задаче поиска изображений.



Как найти это изображение в датасете?

# Large-scale image search

---



11,400 images of game covers  
(Caltech games dataset)



Строим базу данных изображений

- Извлекаем фичи из изображений
- Строим словарь с помощью k-means (обычно, k порядка 100\_000)
- Считаем вес для каждого слова

## Weighting words

---

- Как и в случае с текстом, некоторые визуальные слова уникальнее, чем другие

*и, а, когда*                  **vs.**                  *корова, хоккей, банк*

- Чем больше есть документов, в которых встречается слово, тем оно менее, репрезентативно
  - Например, если слово встречается во всех документах, то оно бесполезно

# TF-IDF

---

- Вместо того, чтобы строить гистограмму частот слов, мы будем учитывать какое у слова inverse document frequency (IDF)
- IDF слова  $j = \log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}}$

# TF-IDF

---

- Значения столбца  $j$  в изображении  $i$ :

$term\ frequency\ of\ j\ in\ i \times inverse\ document\ frequency\ of\ j$

- $j$  - столбец/визуальное слово/центр кластера
- $i$  - изображение/документ

# Inverted file

---

- В каждом изображении около 1000 признаков
- В словаре около 100\_000 визуальных слов
  - Гистограммы получаются разряженными (sparse), почти везде нули
- Inverted file:
  - отображение из слов в документы

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

# Inverted file

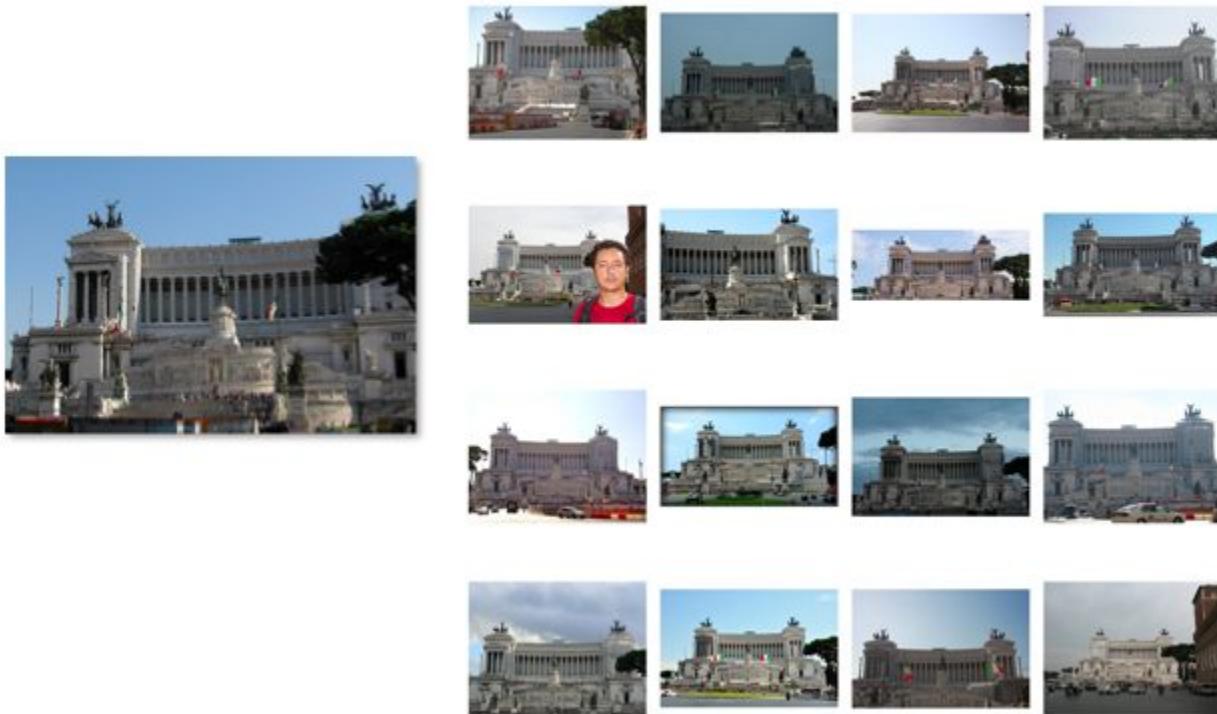
---

- Приходит новое изображение
  - Сравниваем его только с теми изображениями из бд, у которых хоть как-нибудь пересекаются гистограммы

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

# Пример, BoW search

---



# Пример, BoW search

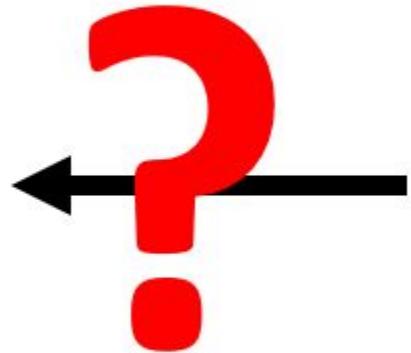
---



# Spatial pyramid matching

# Spatial info

---



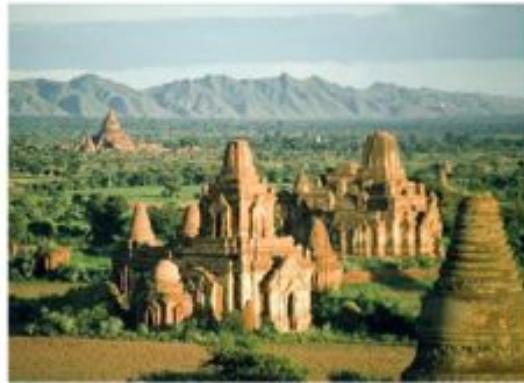
## Pyramid matching

---

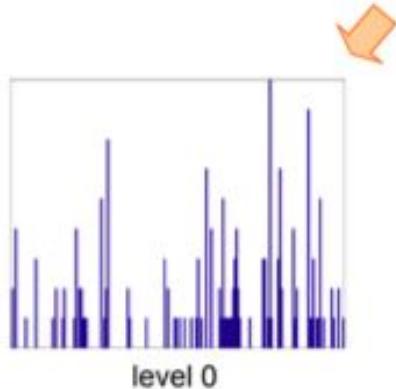
- Очень полезное представление изображений
- Пирамид строятся на уменьшенных копиях изображения
- Каждый уровень пирамиды -  $1/4$  размера предыдущего уровня
- Самый низкий уровень - самое высокое разрешение
- Самый высокий уровень - самое низкое разрешение

# Pyramide matching + BoW

---

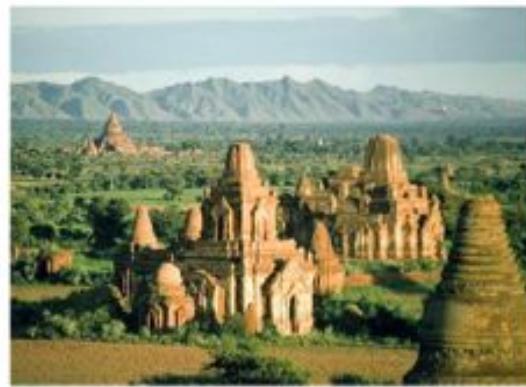


Покалъно неупорядоченное  
представление изображения

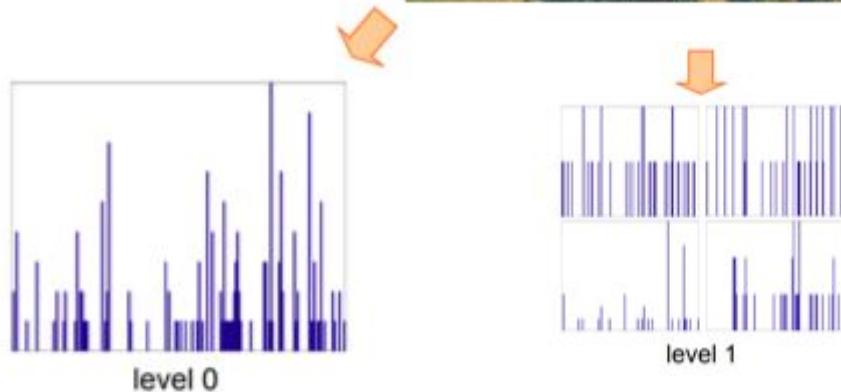


# Pyramide matching + BoW

---

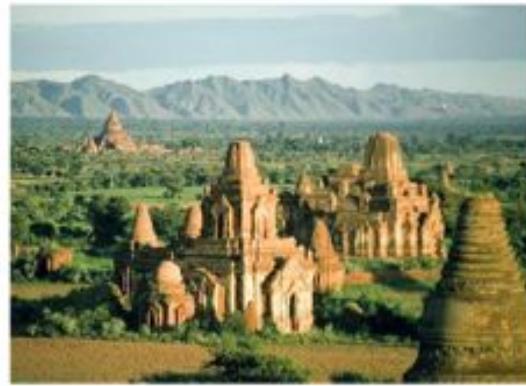


Покалъно неупорядоченое  
представление изображения

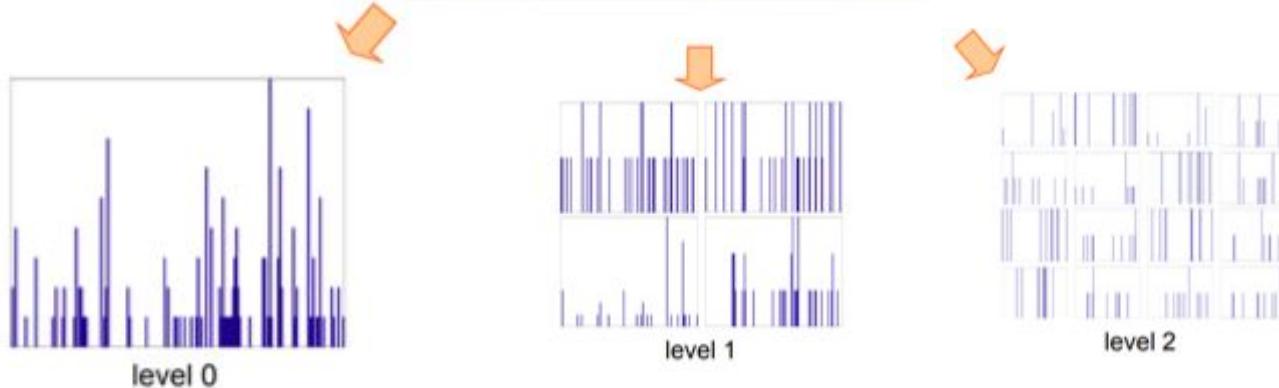


# Pyramide matching + BoW

---

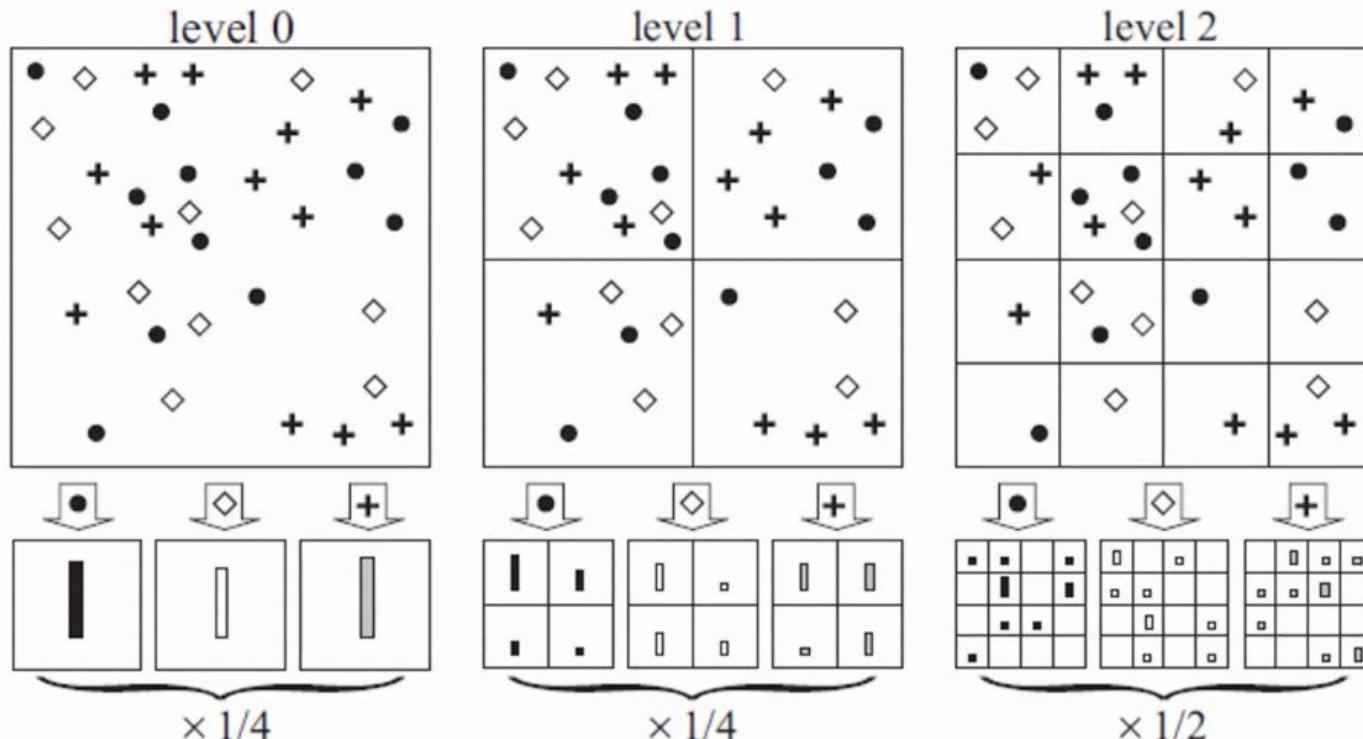


Покально неупорядоченное представление изображения



# Pyramide matching + BoW

---



Let's code!