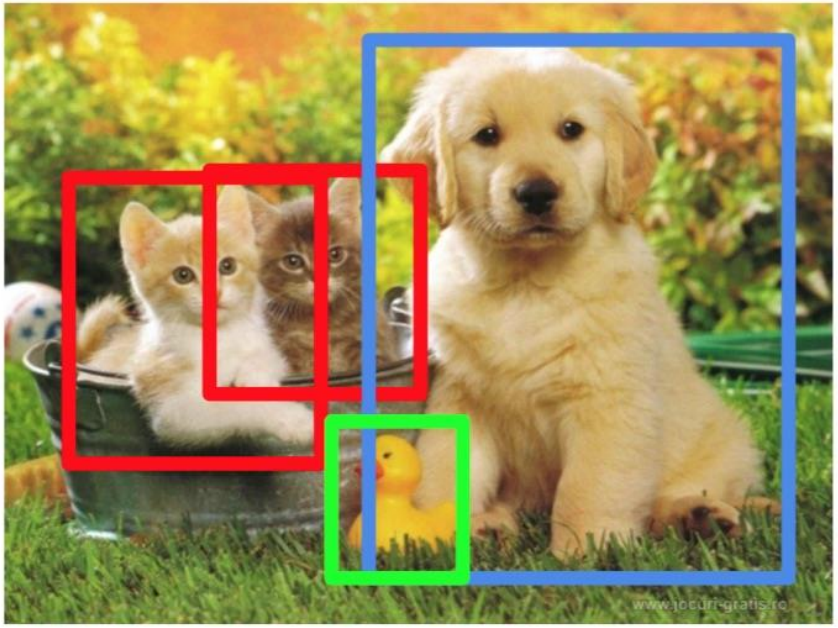
Детектирование объектов, классический подход

Вадим Писаревский,
руководитель проекта OpenCV,
Intel Corporation

Постановка задачи



Кошка

Собака

Утка

Определить положение (прямоугольник) и метку для каждого объекта (из определенного множества классов) на изображении

https://telecombcn-dl.github.io/2017-dlcv/slides/D3L4-objects.pdf

Детектирование объектов как задача классификации

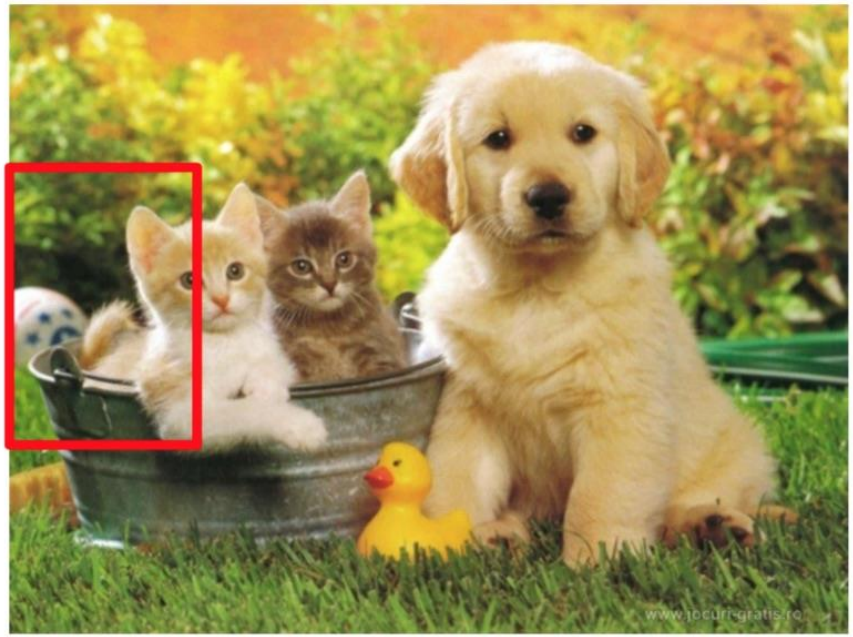
Классы = [Кошка, Собака, Утка]

Кошка (окно(0,0,w,h))? **Нет**

Собака (окно(0,0,w,h))? **Нет**

Утка(окно(0,0,w,h))? **Нет**

Детектирование объектов как задача классификации



Классы = [Кошка, Собака, Утка]

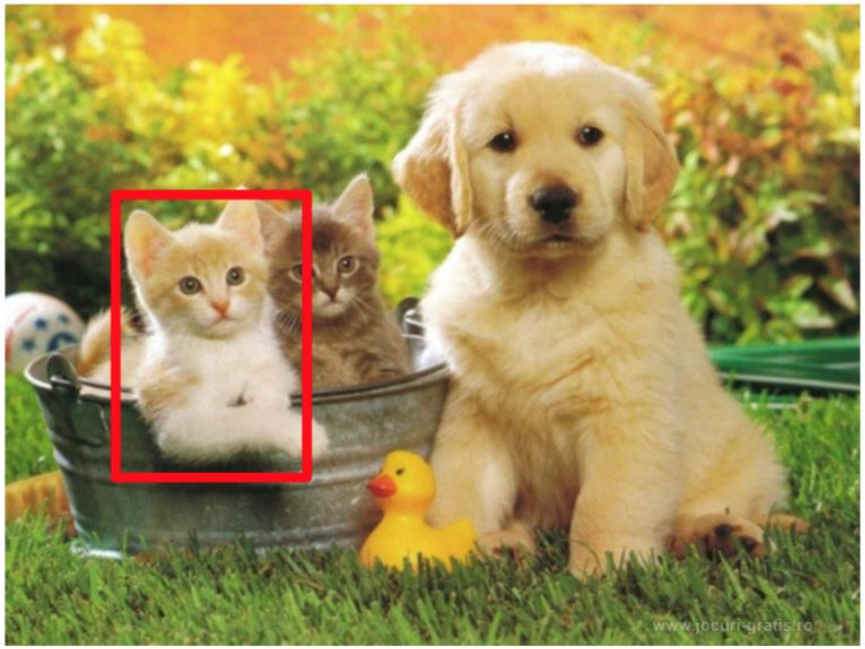
Кошка (окно(0,0,w,h))? **Нет**

Собака (окно(0,0,w,h))? **Нет**

Утка(окно(0,0,w,h))? **Нет**

Частичное пересечение не считается

Детектирование объектов как задача классификации



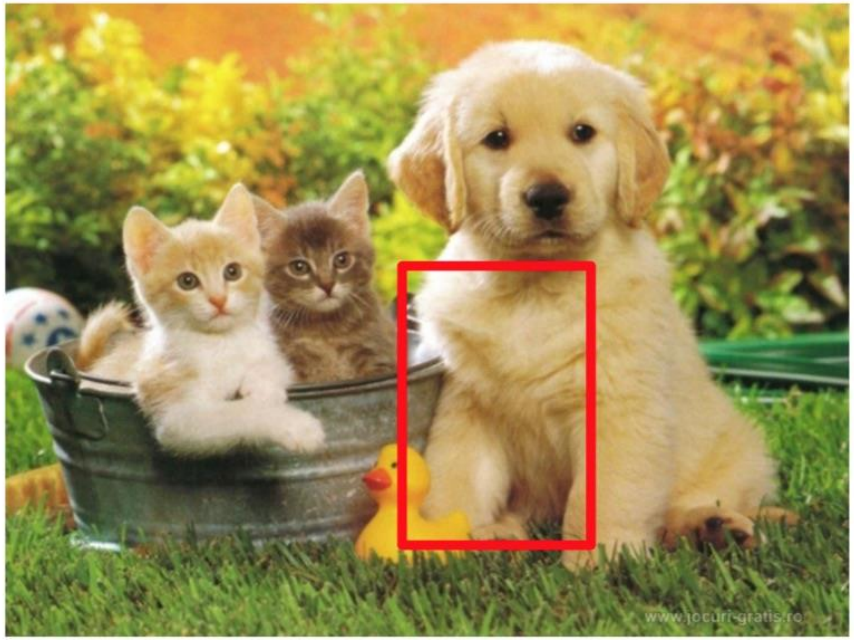
Классы = [Кошка, Собака, Утка]

Кошка (окно(0,0,w,h))? **Да**

Собака (окно(0,0,w,h))? **Нет**

Утка(окно(0,0,w,h))? **Нет**

Детектирование объектов как задача классификации



Классы = [Кошка, Собака, Утка]

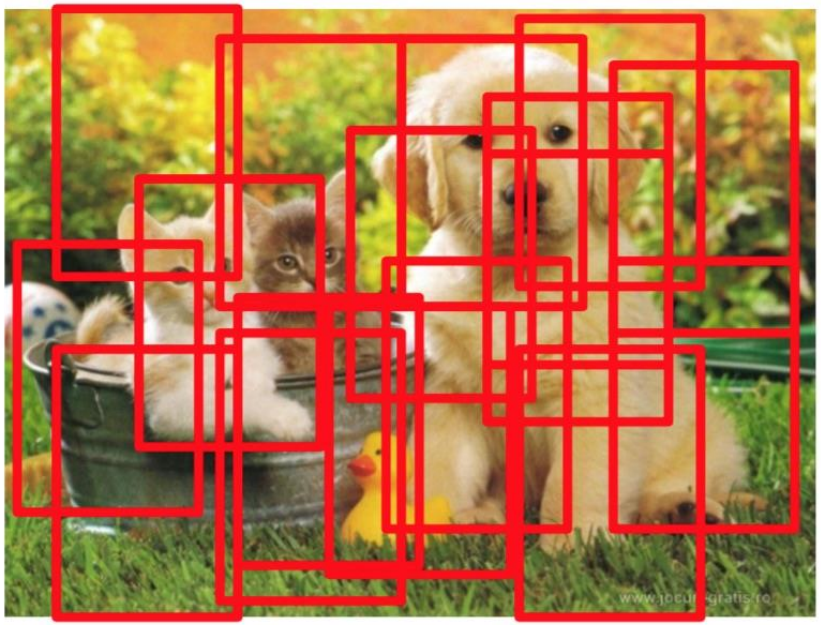
Кошка (окно(0,0,w,h))? **Нет**

Собака (окно(0,0,w,h))? **Нет**

Утка(окно(0,0,w,h))? **Нет**

Частичное покрытие не считается

Перебираем разные положения *и размеры* окон

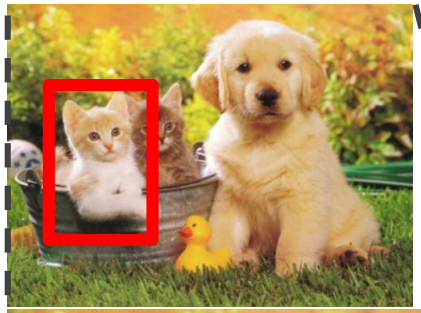


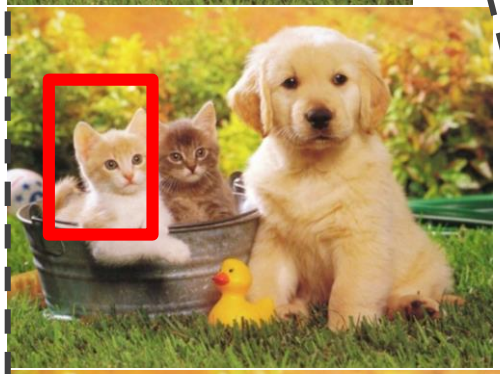
Очень трудоемкая задача – десятки или сотни тысяч положений окна (кандидатов) => классификатор должен быть очень быстрым!

Сколько классификаторов нужно иметь?

Пирамида + фиксированный размер окна

…





Т.н. “multi-scale sliding window approach”

\+ Нужно натренировать только один классификатор на каждый класс.

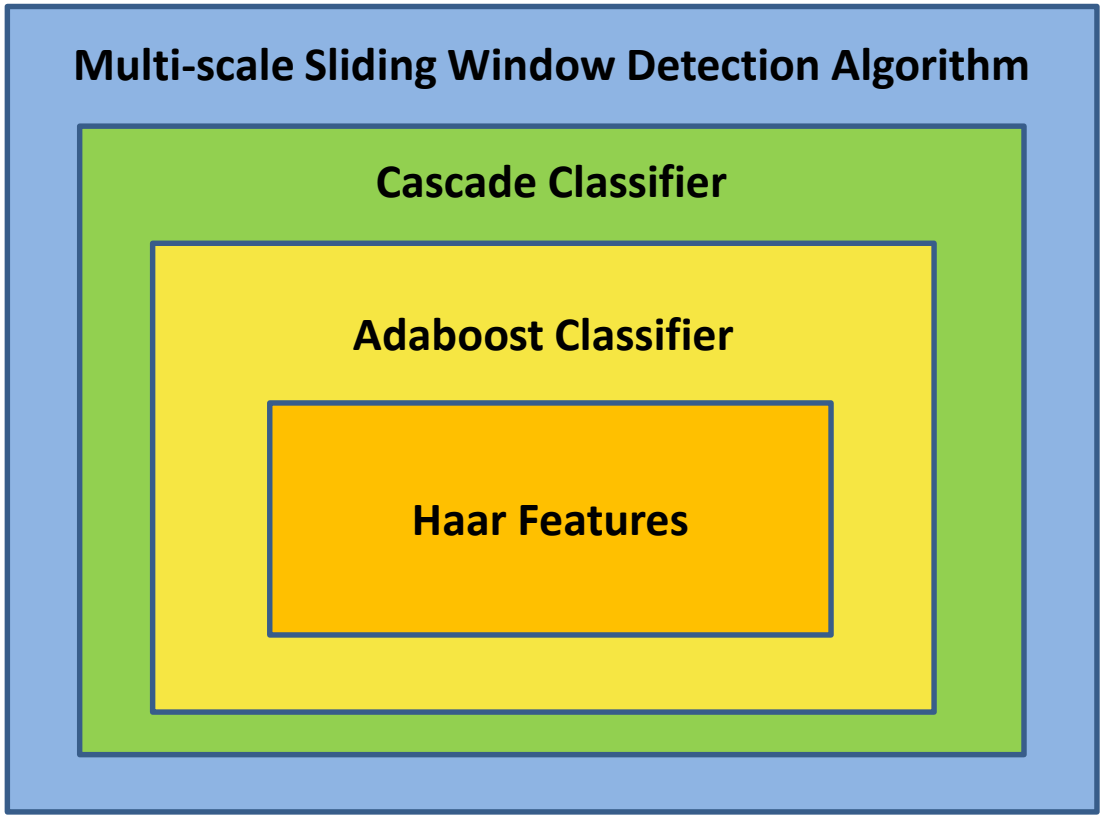
\+ На верхних слоях пирамиды меньше позиций окна – экономим вычисления

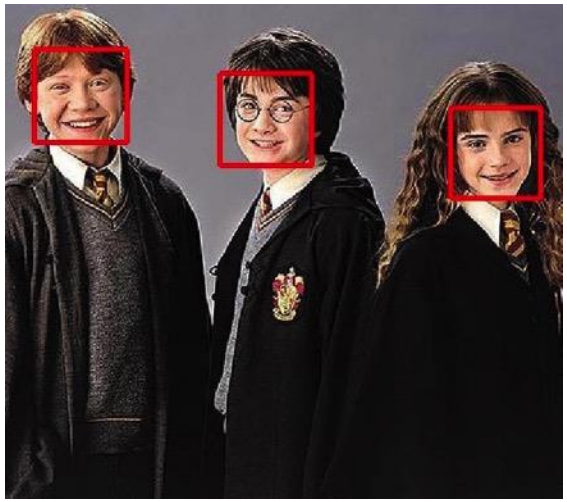
\-- Фиксированное соотношение сторон: ok для лиц, почти ok для пешеходов, не ok – для машин (Google тренировал ~2500 классификаторов для машин до появления Deep Learning)

Детектирование лиц

P. Viola, M. Jones. Rapid object detection using a boosted cascade of simple features. CVPR 2001.

Первый realtime детектор лиц: 15FPS на Pentium III



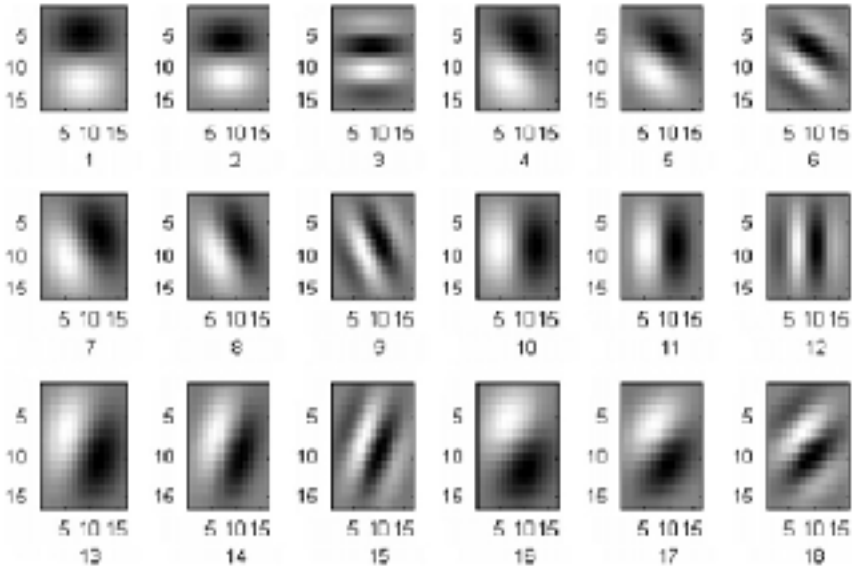


+ Тренировка

Тренировка классификатора как сжатие с потерями – выделить общее/ценное, отбросить детали/шум

Подготовим базу: 10000 лиц приведенных к размеру 20x20, примерно столько же не-лиц 20x20.

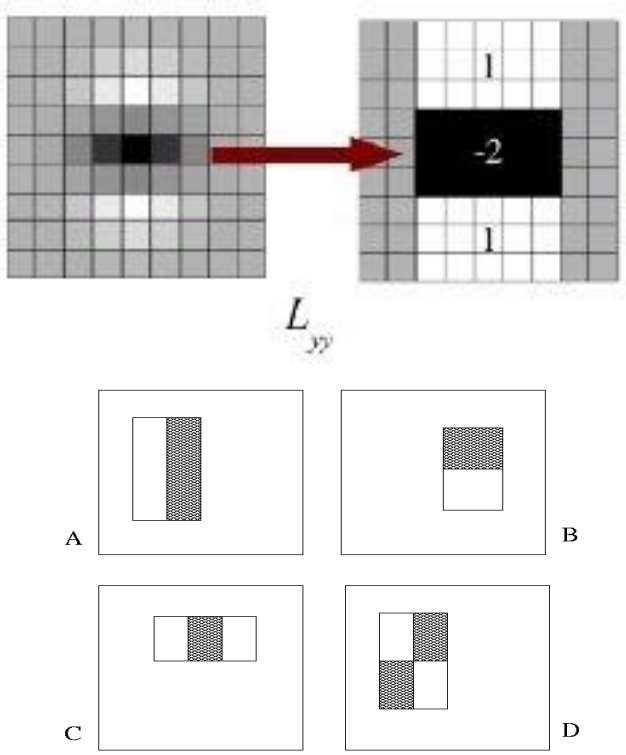
Подход “в лоб”. Фильтры Габора

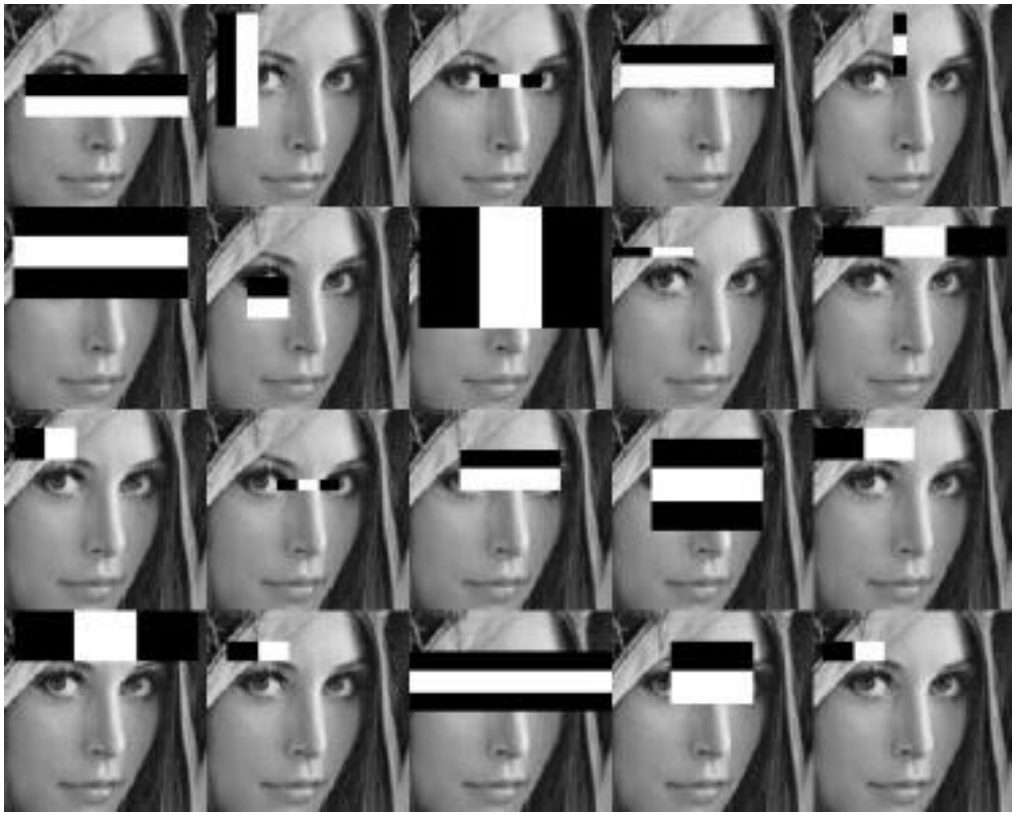


Аппроксимируем изображение 20x20 с помощью декореллированного вектора коэффициентов: PCA, ICA, DFT, DCT, Gabor filters (1980s), …. Потом применим нелинейный SVM (1993) или многослойную нейронную сеть (1980s).

- Это работает, но очень медленно!
- Затруднительно использовать большую базу для тренировки

Упрощаем и ускоряем фильтры – используем фильтры/фичи Хаара

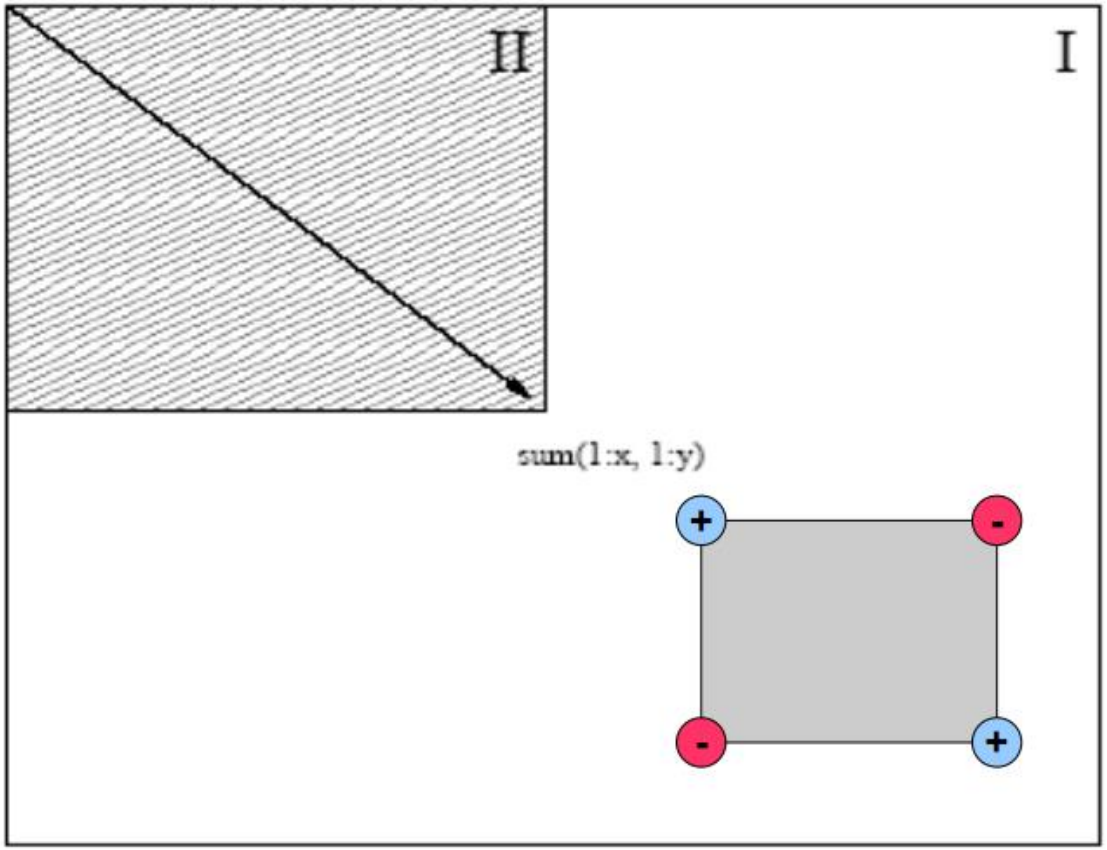




Для окна 20x20 всего порядка 40000 фич Хаара

Коэффициенты внутри каждой черной/белой области одинаковы

Считаем фичи Хаара за O(1) с помощью интегральных изображений



Имея Integral(X,Y) = $\sum_{x \le X,\ y \le Y} img(x,y)$, можно вычислить $\sum_{x1 \le x \le x2,\ y1 \le y \le y2} img(x,y)$ за несколько операций.

Заменяем классификатор на Adaboost

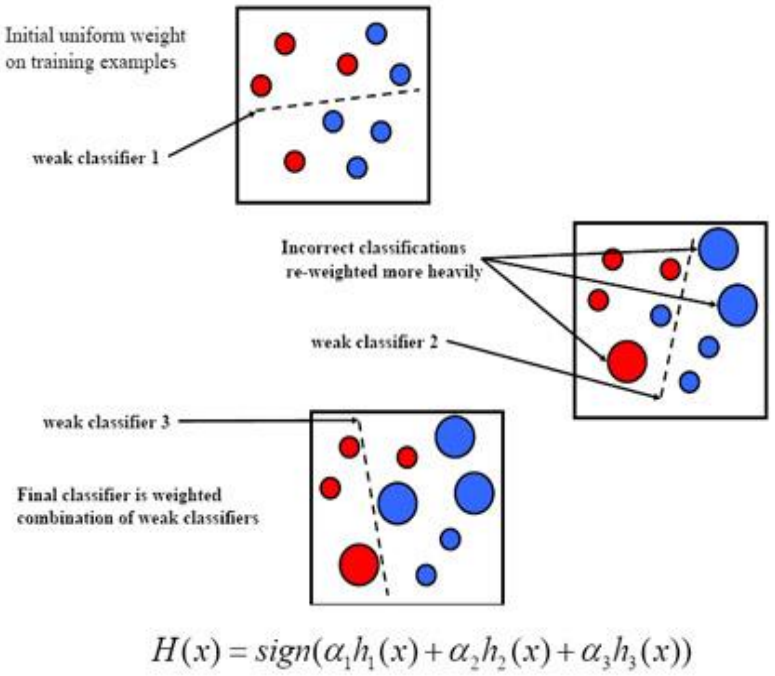
Сильный классификатор:

$f(w) = sign(\sum_t a_t h_t(w))$

Слабый классификатор – минимальное дерево решений (”пенек”)

$h_t(w) = sign(HaarFeature_t(w) - d_t)$

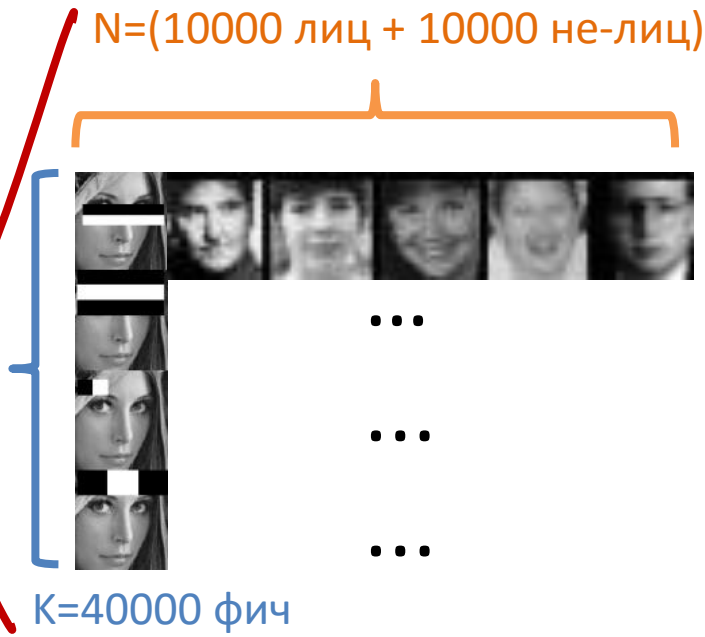
Freund, Yoav; Schapire, Robert E (1997). "A decision-theoretic generalization of on-line learning and an application to boosting”. Придумали алгоритм Discrete Adaboost для построения сильного классификатора f(w) из слабых и доказали для него следующую теорему: **“При наличии достаточного количества слагаемых, f(w) может стать сколь угодно сильным если $h_t(w)$ хотя бы чуть лучше чем случайный выбор”**



Учимся на ошибках!

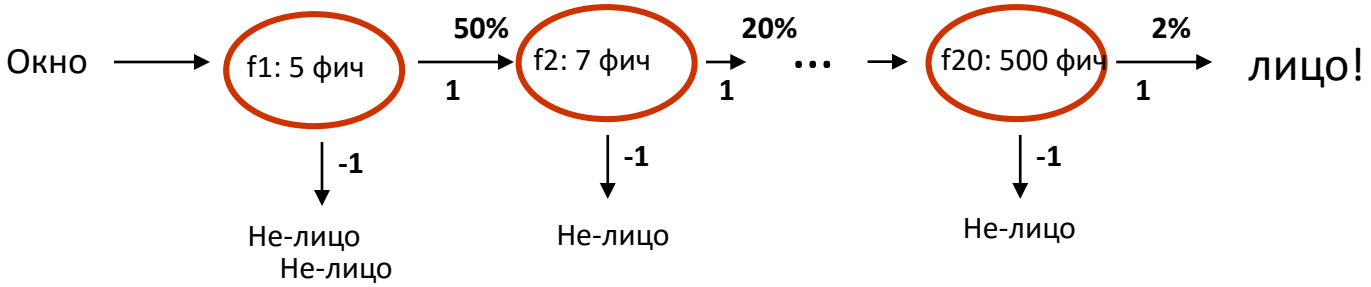
Adaboost: алгоритм

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$



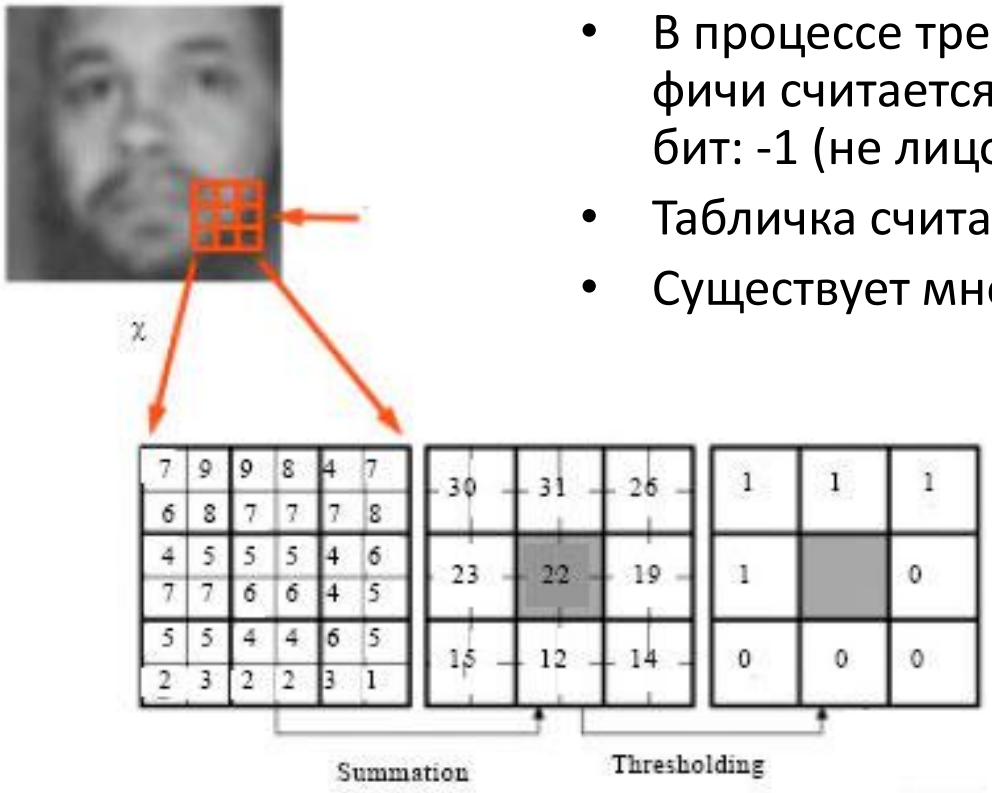
Шаг 2 выполняется за K*O(N) !!!

Один огромный классификатор => Каскад классификаторов



- Каждый f_i натренирован с ~1 (0.999) hit-rate и 0.5 false-alarm (false positive) rate.
- Для тренировки каждого f_i отбираем только сэмплы (позитивные и негативные), которые прошли все предыдущие $f_1 \ldots f_{i-1}$.
- **Результирующий классификатор из 20 стадий имеет hit-rate $0.999^{20} \approx 0.98$ и false alarm rate $0.5^{20} \approx 10^{-6}$.** Из сильных классификаторов мы собрали супер-сильный классификатор!
- Не только супер-сильный, но и супер-быстрый: откидывает ~90% кандидатов на первых 3 стадиях
- Вопрос на засыпку – как набрать достаточно негативных сэмплов для тренировки каждой стадии классификатора?

Улучшение алгоритма Viola-Jones #1: использовать другие фичи, например LBP



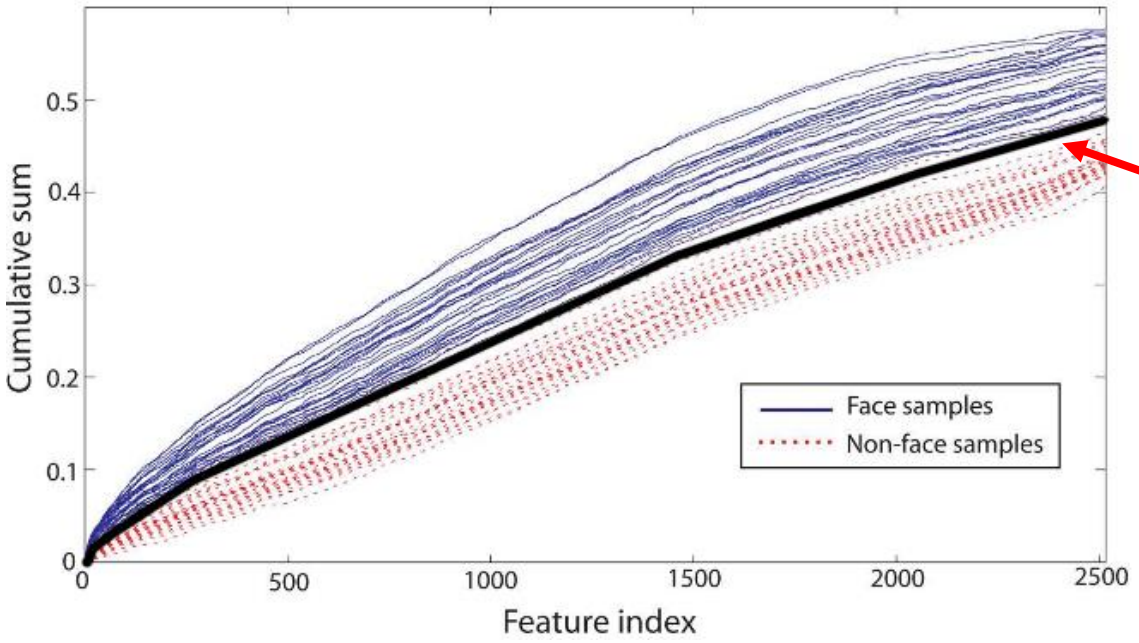
- В процессе тренировки для каждой фичи считается LUT (табличка) из 256 бит: -1 (не лицо)/+1 (лицо).
- Табличка считается за O(N) !
- Существует множество вариантов LBP

- Каждая фича вычисляется сложнее, но более дескриптивна
- Все вычисления целочисленные
- Всего для окна 20x20 имеется ≈ 4000 LBP фич вместо 40000 фич Хаара.
- На iphone каскад на LBP фичах работает в 10 раз быстрее Хаара, на ноутбуке – в 2-3 раза быстрее, занимая в 20 раз меньше памяти.

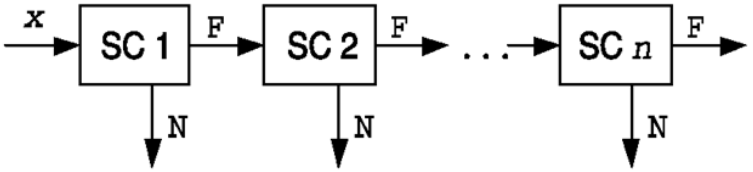
Улучшение алгоритма Viola-Jones #2: не считать каждую стадию каскада “с нуля“

- Lubomir Bourdev and Jonathan Brandt. Robust Object Detection Via Soft Cascade. IEEE CVPR, 2005.

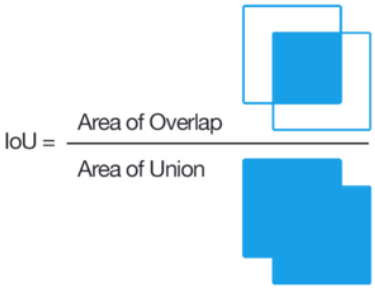
Частичные суммы при работе последней стадии каскада

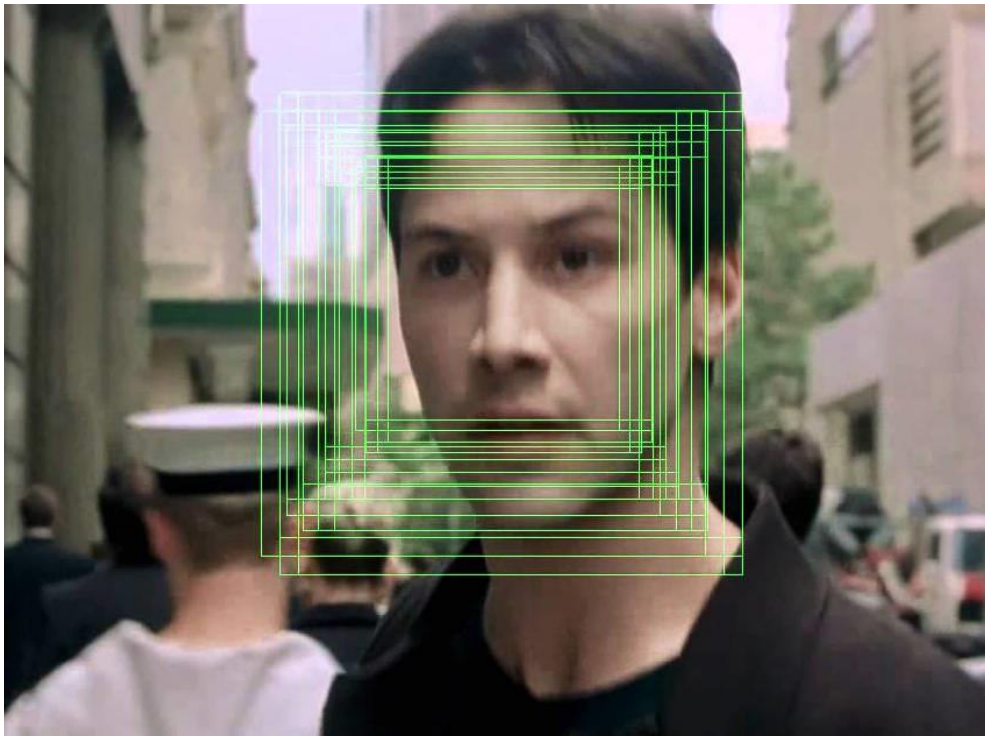


Можно развернуть каскад в строку!
Т.е. не сбрасывать сумму слабых классификаторов, но продолжать накапливать!



Группировка выходных прямоугольников



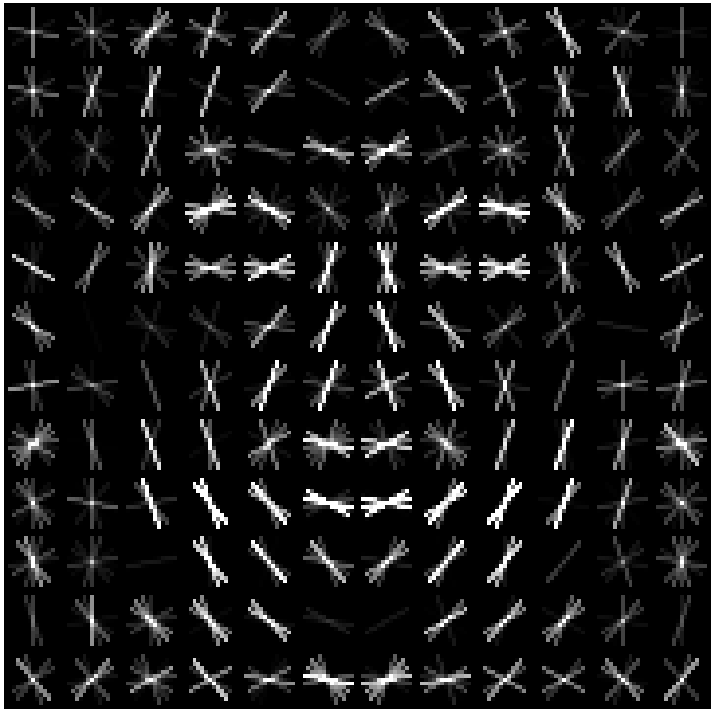


Jaccard Distance (A,B) = area(A∩B)/area(A∪B)= area(A∩B)/(area(A) + area(B) - area(A∩B))

- A и B считаются близкими если JD(A, B) ≤ eps (eps ≈ 0.6..0.9)
- Можно построить классы эквивалентности (связные компоненты в графе близких кандидатов). Из каждого класса сформировать один прямоугольник (median, avg, …)
- Можно использовать *non-maxima suppression*: рассмотреть все пары близких прямоугольников (A,B), выбрать прямоугольник с б**о**льшим значением confidence

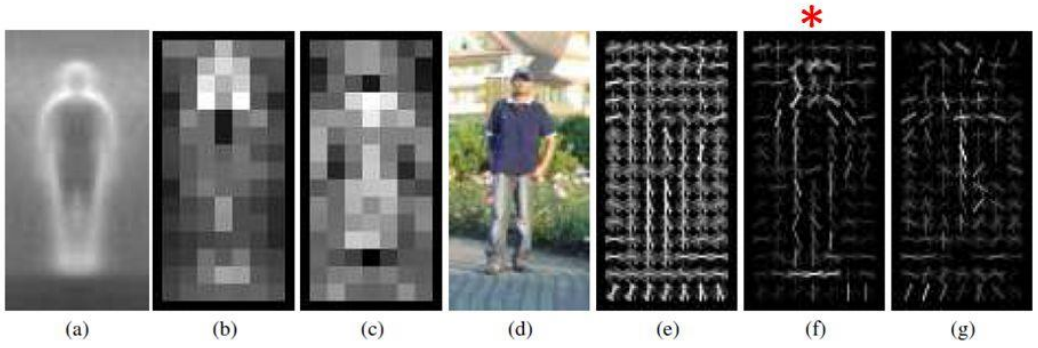
Что насчет других классов? Пешеходы, машины и т.д.

- Ни LBP, ни Haar фичи не годятся для сложных, разноцветных, разнотекстурных объектов.
- Вместо этого часто применяются вариации HOG (histograms of oriented gradients)



Впрочем, HOG работают и для лиц

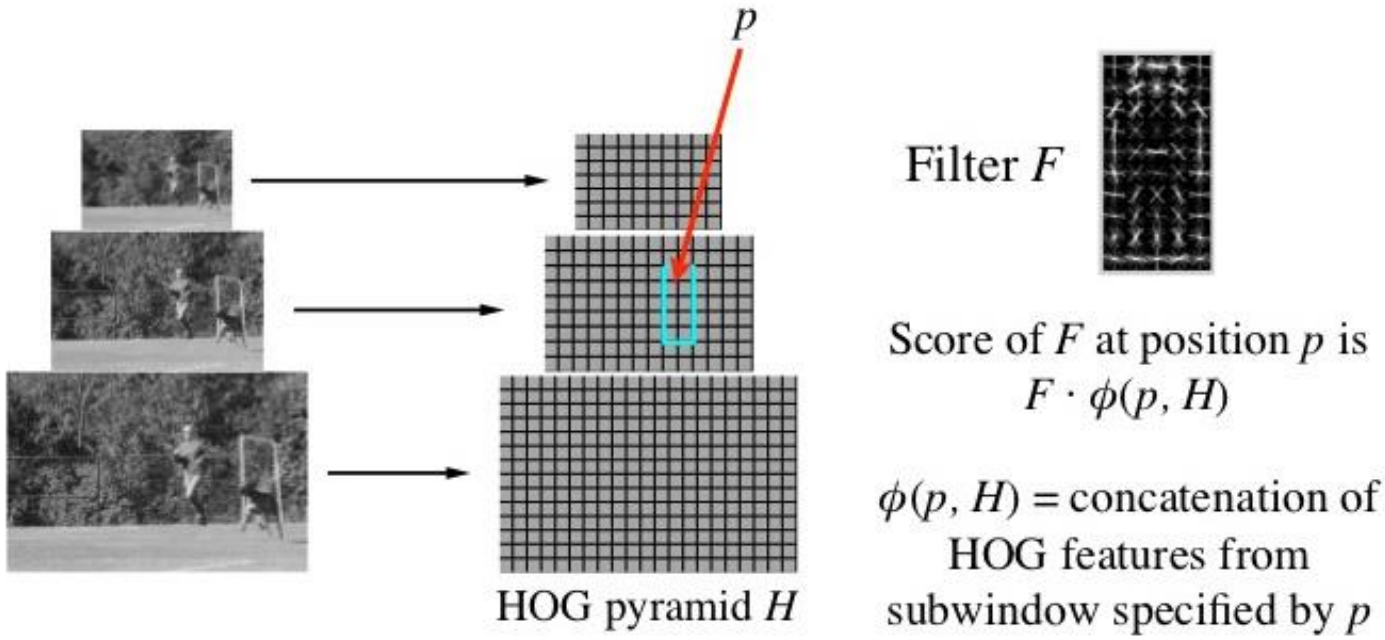
Pictorial Example of HOG for Human Detection



(a) average gradient image over training examples
(b) each "pixel" shows max positive SVM weight in the block centered on that pixel
(c) same as (b) for negative SVM weights
(d) test image
(e) its R-HOG descriptor
(f) R-HOG descriptor weighted by positive SVM weights
(g) R-HOG descriptor weighted by negative SVM weights

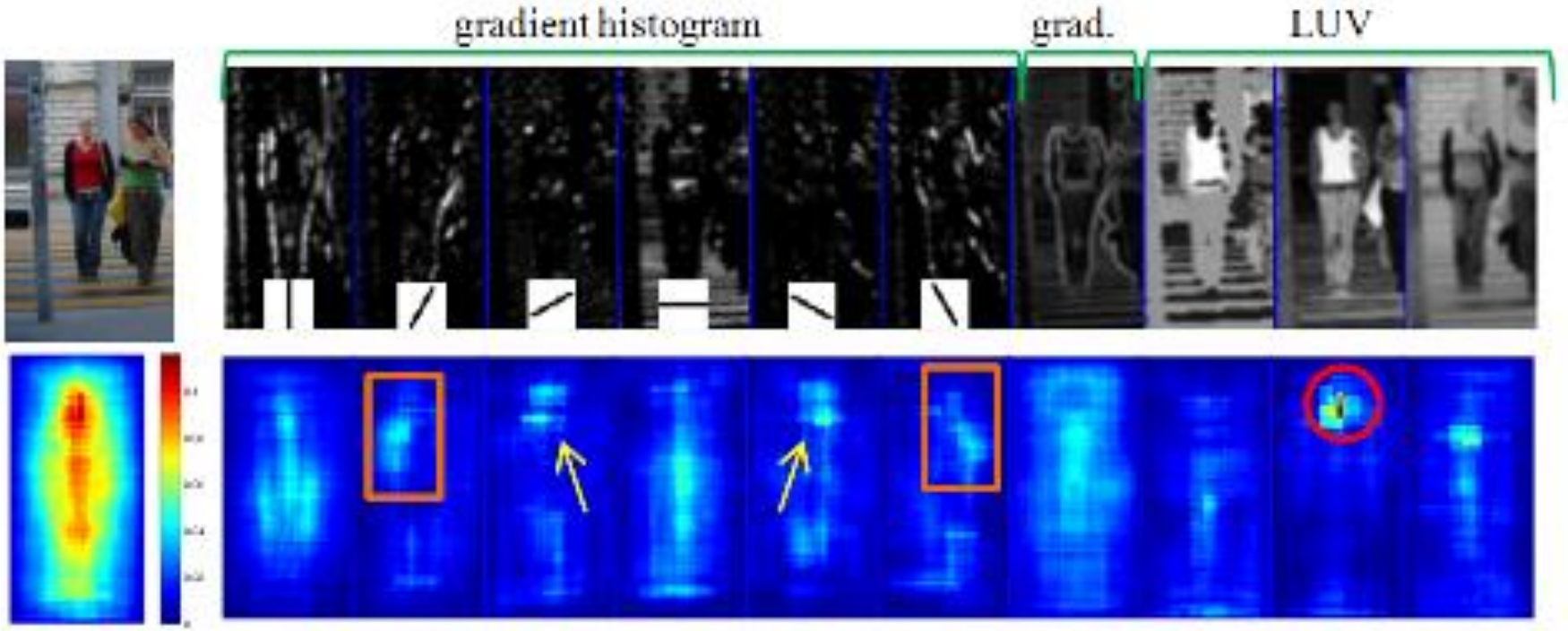
HOG алгоритмы. Начало

- N. Dalal, B. Triggs. Histograms of oriented gradients for human detection (CVPR 2005):
- Используются пирамиды и скользящее окно. Окно разбивается на блоки, шаг сдвига окна кратен блоку. Для окна считается дескриптор HOG из 3780 элементов (собирается из дескрипторов блоков). Для получения более адекватных гистограмм используется трилинейная интерполяция
- Дескриптор классифицируется с помощью линейного SVM



HOG алгоритмы. ICF

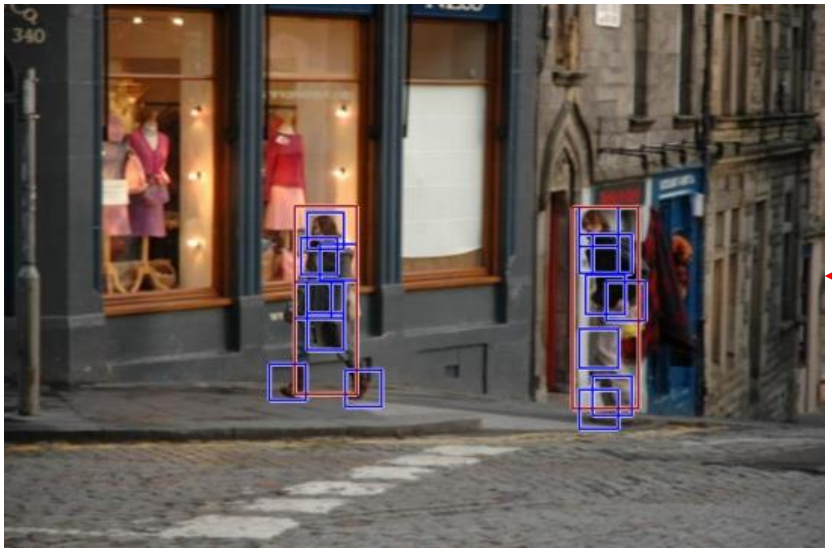
- P. Dollar et al. Integral Channel Features. BMVC 2009.
- R. Benenson. Pedestrian detection at 100 frames per second. CVPR 2012.



- Берем алгоритм Viola Jones за основу: integral images, rectangular features, Adaboost, cascade classifier, multi-scale sliding window.
- 10 типов фич: 6 направлений градиентов (HOG) + LUV + gradient magnitude.
- Статья R. Benenson’a. Необязательно считать пирамиду фич, можно посчитать на исходном разрешении, потом масштабировать окно.

Преимущества/недостатки классического подхода. Другие алгоритмы.

- Классические алгоритмы отлично работают на лицах, удовлетворительно на пешеходах, плохо на машинах и номерах машин.
- Высокая скорость работы (15-100fps).
- Плохо обрабатываются некомпактные и/или “гибкие” объекты, малотекстурные объекты, частично загороженные объекты, классы объектов с большой вариативностью (кружки, кресла и т.д.). Нетривиально сделать детектор сразу для нескольких классов. Такие задачи как детектирование и распознавание продуктов питания, растений разных видов или собак разных пород кажутся нерешаемыми => только deep learning!
- Не было рассмотрено популярное семейство алгоритмов “deformable part models“ – ICF + Boosting работает лучше и быстрее!



Вопросы?