面向移动设备的目标检测算法

celerychen

2017-2-26

关键要点

- ◆基于滑动窗检测的一般思路
- ◆如何快速有效的计算特征
- ◆如何加速训练算法
- ◆快速的检测算法
- ◆ACF目标检测

目标检测与监督学习

- 监督学习

  给定特征X，给定分类目标Y，寻找X到Y的概率映射关系Y = F(X)

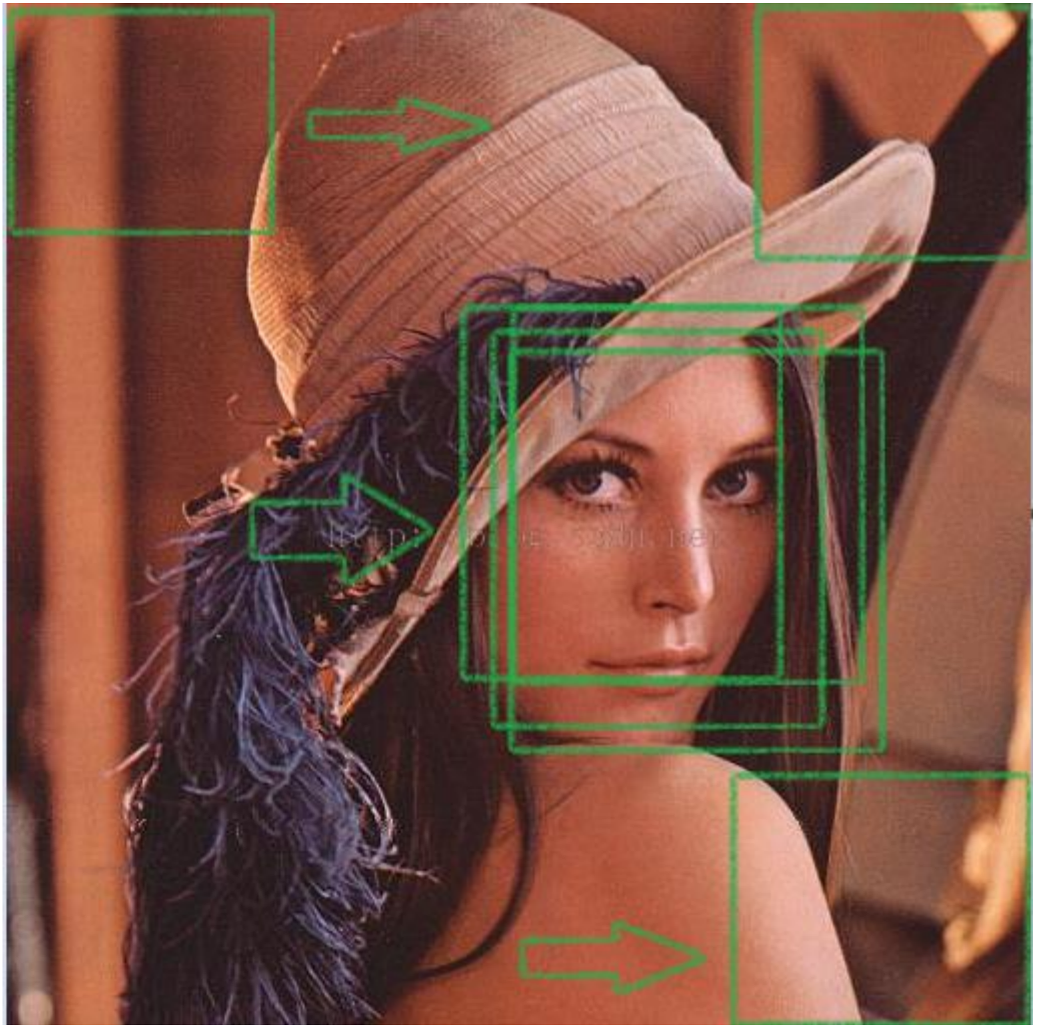
- 目标检测

  需要对图像中所有可能的目标区域提取出特征X，同时得到Y。

  采集大量的X和Y，采用监督学习的算法得到概率映射关系Y=F(X)

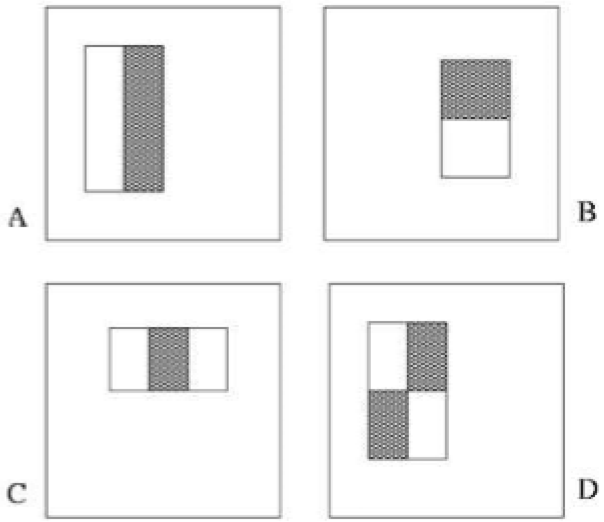
  根据已有的概率映射关系，对图像中所有可能区域，判别Y的可能性。

滑动窗遍历所有可能的区域



Haar-like特征

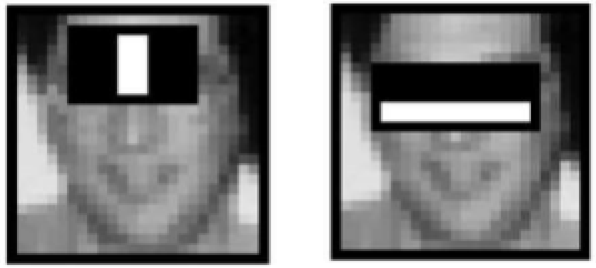
Rectangle Filters



W = $\sum(pixels\ in\ white\ area)$

B = $\sum(pixels\ in\ black\ area)$

Feature Value = W - B



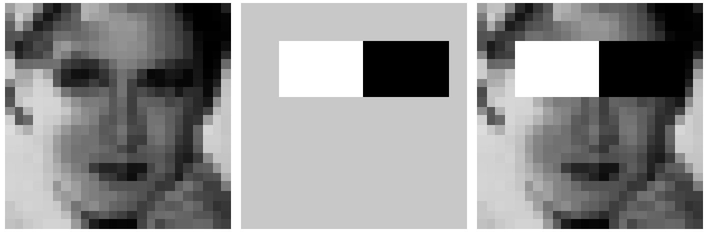
Encode information such as “eyes are darker than nose”

Haar-like特征

The Viola-Jones algorithm uses Haar-like features, that is, a scalar product between the image and some Haar-like templates. More precisely, let I and P denote an image and a pattern, both of the same size $N \times N$ (see Figure 1). The feature associated with pattern P of image I is defined by

$$\sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j)\text{ is white}} - \sum_{1\le i\le N}\sum_{1\le j\le N} I(i,j)1_{P(i,j)\text{ is black}}.$$

To compensate the effect of different lighting conditions, all the images should be mean and variance normalized beforehand. Those images with variance lower than one, having little information of interest in the first place, are left out of consideration.

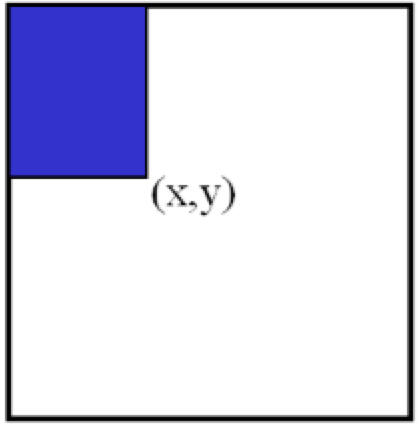


(a) (b) (c)

Haar-like features. Here as well as below, the background of a template like (b) is painted gray to highlight the pattern's support. Only those pixels marked in black or white are used when the corresponding feature is calculated.

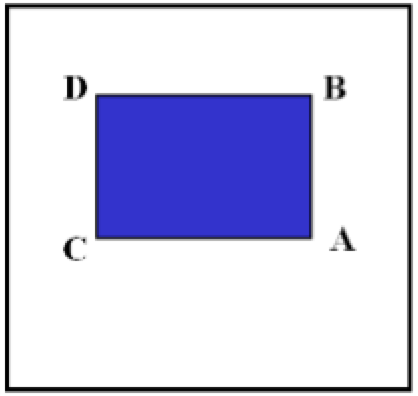
积分图快速计算矩形特征

Integral Image



- The *integral image* computes a value at each pixel (*x*,*y*) that is the sum of the pixel values above and to the left of (*x*,*y*), inclusive
- This can quickly be computed in one pass through the image [see Viola&Jones paper]

Integral Image



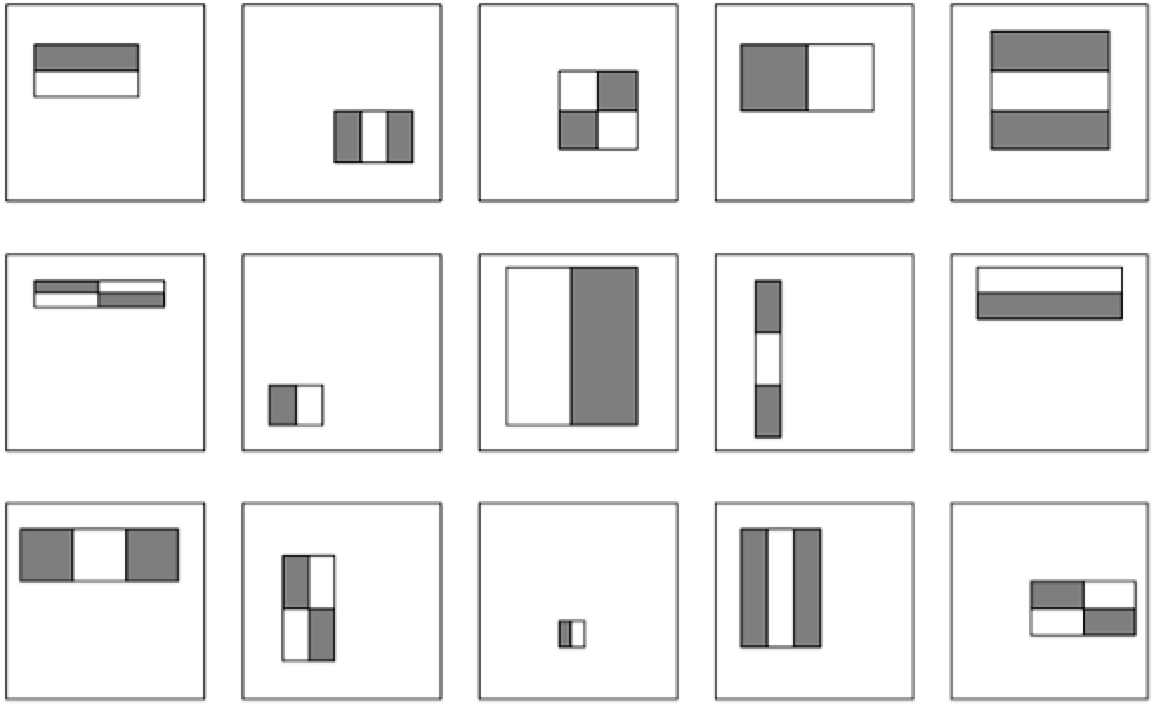
- The integral image allows fast evaluation of rectangle filters:

  sum = A – B – C + D

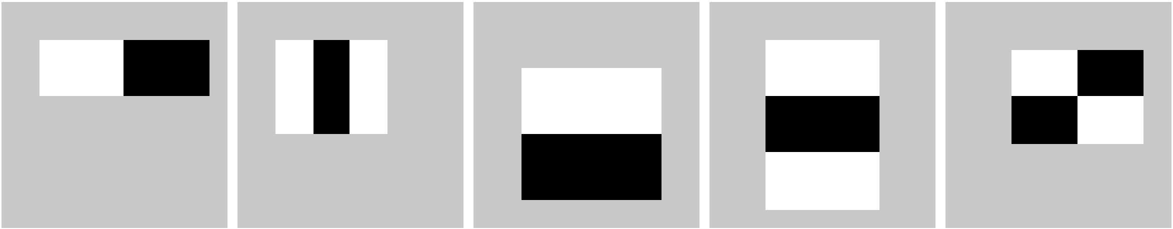
- Only 3 additions are required for any size of rectangle!

标准训练模板下的特征计算

For a 24x24 detection region, the number of possible rectangle features is ~160,000!



标准训练模板下的特征计算



(a) (b) (c) (d) (e)

Algorithm 1 Computing a 24×24 image's Haar-like feature vector

Input: a 24×24 image with zero mean and unit variance

Output: a $d \times 1$ scalar vector with its feature index $\mathfrak{f}$ ranging from 1 to d

Set the feature index $\mathfrak{f} \leftarrow 0$

Compute feature type (a)

for all (i, j) such that $1 \leq i \leq 24$ and $1 \leq j \leq 24$ **do**

    for all (w, h) such that $i + h - 1 \leq 24$ and $j + 2w - 1 \leq 24$ **do**

        compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$

        compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$

        record this feature parametrized by $(1, i, j, w, h)$: $S_1 - S_2$

        $\mathfrak{f} \leftarrow \mathfrak{f} + 1$

    end for

end for

Compute feature type (b)

for all (i, j) such that $1 \leq i \leq 24$ and $1 \leq j \leq 24$ **do**

    for all (w, h) such that $i + h - 1 \leq 24$ and $j + 3w - 1 \leq 24$ **do**

        compute the sum S_1 of the pixels in $[i, i + h - 1] \times [j, j + w - 1]$

        compute the sum S_2 of the pixels in $[i, i + h - 1] \times [j + w, j + 2w - 1]$

        compute the sum S_3 of the pixels in $[i, i + h - 1] \times [j + 2w, j + 3w - 1]$

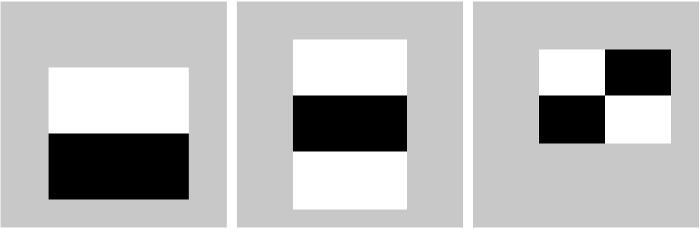
        record this feature parametrized by $(2, i, j, w, h)$: $S_1 - S_2 + S_3$

        $\mathfrak{f} \leftarrow \mathfrak{f} + 1$

    end for

end for

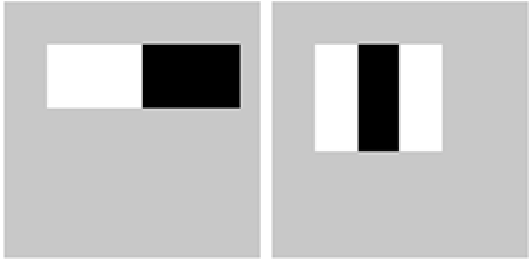
标准训练模板下的特征计算



(c) (d) (e)

Compute feature type (c)
for all (i,j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w,h) such that $i+2h-1 \le 24$ and $j+w-1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i+h-1] \times [j, j+w-1]$
    compute the sum S_2 of the pixels in $[i+h, i+2h-1] \times [j, j+w-1]$
    record this feature parametrized by $(3, i, j, w, h)$: $S_1 - S_2$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (d)
for all (i,j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w,h) such that $i+3h-1 \le 24$ and $j+w-1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i+h-1] \times [j, j+w-1]$
    compute the sum S_2 of the pixels in $[i+h, i+2h-1] \times [j, j+w-1]$
    compute the sum S_3 of the pixels in $[i+2h, i+3h-1] \times [j, j+w-1]$
    record this feature parametrized by $(4, i, j, w, h)$: $S_1 - S_2 + S_3$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for
Compute feature type (e)
for all (i,j) such that $1 \le i \le 24$ and $1 \le j \le 24$ **do**
  for all (w,h) such that $i+2h-1 \le 24$ and $j+2w-1 \le 24$ **do**
    compute the sum S_1 of the pixels in $[i, i+h-1] \times [j, j+w-1]$
    compute the sum S_2 of the pixels in $[i+h, i+2h-1] \times [j, j+w-1]$
    compute the sum S_3 of the pixels in $[i, i+h-1] \times [j+w, j+2w-1]$
    compute the sum S_4 of the pixels in $[i+h, i+2h-1] \times [j+w, j+2w-1]$
    record this feature parametrized by $(5, i, j, w, h)$: $S_1 - S_2 - S_3 + S_4$
    $\mathfrak{f} \leftarrow \mathfrak{f} + 1$
  end for
end for

任意尺度下特征的计算

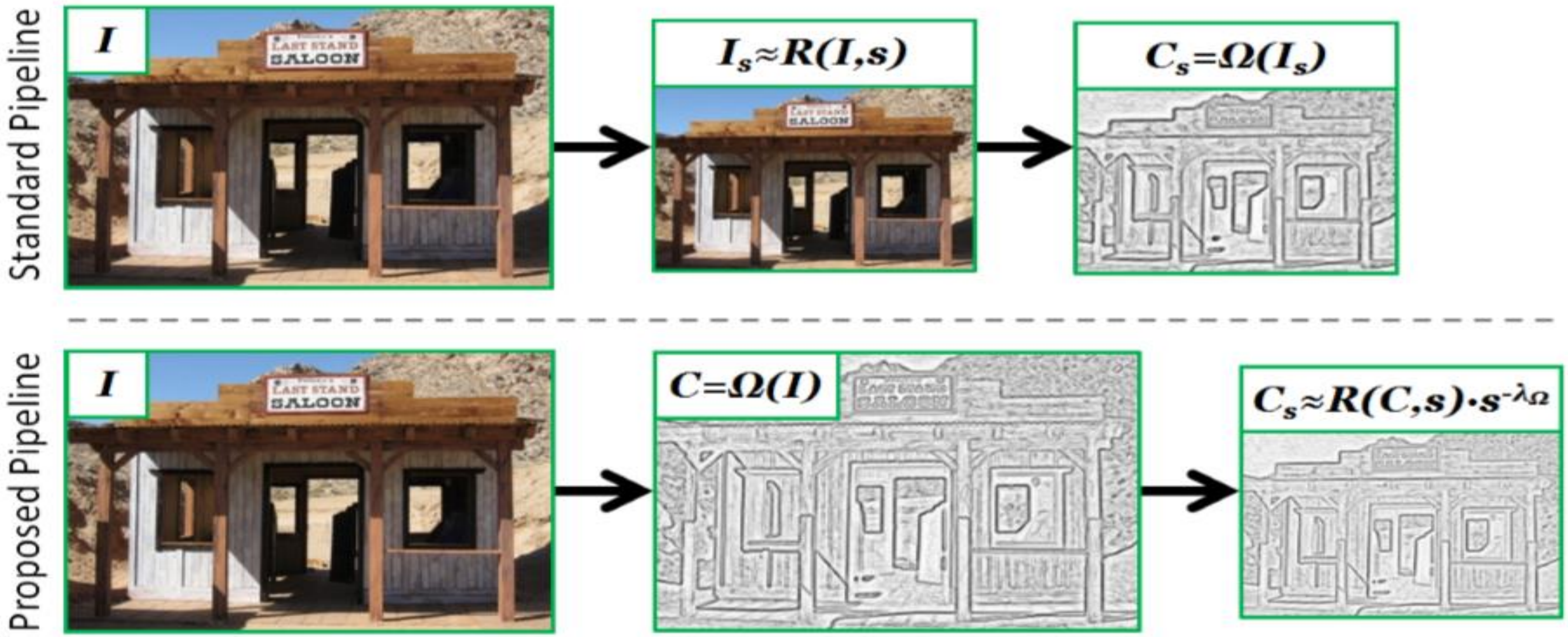


(a) (b)

Algorithm 3 Feature Scaling

Input: an $e \times e$ image with zero mean and unit variance ($e \geq 24$)
Parameter: a Haar-like feature type and its parameter (i, j, w, h) as defined in Algorithm 1
Output: the feature value
if *feature type* (a) **then**
  set the original feature support size $a \leftarrow 2wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, h \leftarrow \mathrm{J}he/24\mathrm{K}$ where $\mathrm{J}z\mathrm{K}$ defines the nearest integer to $z \in \mathbb{R}_+$
  $w \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 2we/24\mathrm{K}/2,\ 2\kappa \leq e - j + 1\}$
  compute the sum S_1 of the pixels in $[i, i+h-1] \times [j, j+w-1]$
  compute the sum S_2 of the pixels in $[i, i+h-1] \times [j+w, j+2w-1]$
  return the scaled feature $\frac{(S_1 - S_2)a}{2wh}$
end if
if *feature type* (b) **then**
  set the original feature support size $a \leftarrow 3wh$
  $i \leftarrow \mathrm{J}ie/24\mathrm{K}, j \leftarrow \mathrm{J}je/24\mathrm{K}, h \leftarrow \mathrm{J}he/24\mathrm{K}$
  $w \leftarrow \max\{\kappa \in \mathbb{N} : \kappa \leq \mathrm{J}1 + 3we/24\mathrm{K}/3,\ 3\kappa \leq e - j + 1\}$
  compute the sum S_1 of the pixels in $[i, i+h-1] \times [j, j+w-1]$
  compute the sum S_2 of the pixels in $[i, i+h-1] \times [j+w, j+2w-1]$
  compute the sum S_2 of the pixels in $[i, i+h-1] \times [j+2w, j+3w-1]$
  return the scaled feature $\frac{(S_1 - S_2 + S_3)a}{3wh}$
end if

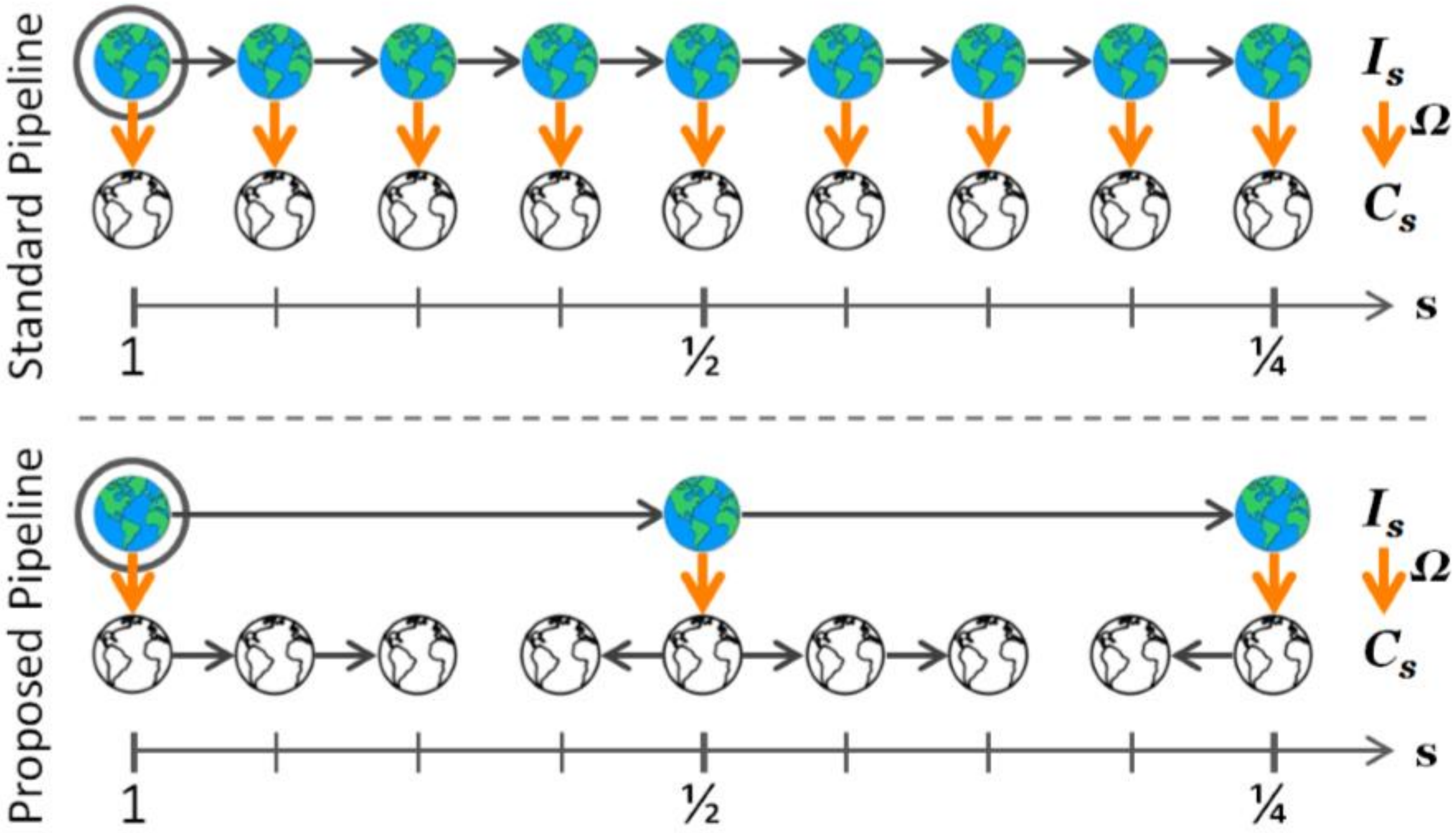
任意尺度HOG特征快速计算



The standard approach is to compute $C_s = \Omega(R(I, s))$, ignoring the information contained in $C = \Omega(I)$. Instead, we propose the following approximation:

$$\boxed{C_s \approx R(C, s) \cdot s^{-\lambda_\Omega}}$$

尺度金字塔HOG特征近似计算



HOG特征中求梯度的优化

```
static void hog_compute_gradient_gray(iv8u* outOri, iv16s* outMag, int width, int height, int stride, iv8u* img){
    int x, y;
    for (y = 1; y < height - 1; y++) {
        iv16s* ptr_outMag_t = outMag + y * width;
        iv8u*  ptr_outOri_t = outOri + y * width;
        ptr_outOri_t[0] = 0, ptr_outMag_t[0] = 0;
        for (x = 1; x < width - 1; x++) {
            int dx1 = img[(y + 0) * stride + (x + 1)] - img[(y + 0)* stride + (x - 1)];
            int dy1 = img[(y + 1) * stride + (x + 0)] - img[(y - 1)* stride + (x + 0)];
            int v1 = abs(dx1) + abs(dy1);
            {
                ivf32 best_dot = 0;
                int  best_o = 0, o;
                for (o = 0; o < 9; o++) {
                    ivf32 dot = uu[o] * dx1 + vv[o] * dy1;
                    if (dot > best_dot) {
                        best_dot = dot;
                        best_o = o;
                    }  else if (-dot > best_dot) {
                        best_dot = -dot;
                        best_o = o + 9;
                    }
                }
                ptr_outMag_t[x] = v1;
                ptr_outOri_t[x] = best_o;
            }
        }
        ptr_outMag_t[x] = 0;
        ptr_outOri_t[x] = 0;
    }
}
```

HOG特征中求梯度的优化

```
        for (y = 1; y < height - 1; y++){
            iv8u* line_i_sub = img + (y - 1) * stride;
            iv8u* line_i     = img + (y + 0) * stride;
            iv8u* line_i_add = img + (y + 1) * stride;
            iv16s* ptr_outMag_t = outMag + y * width;
            iv8u*  ptr_outOri_t = outOri + y * width;
            *ptr_outOri_t++ = 0; *ptr_outMag_t++ = 0;
            line_i_sub++; line_i++; line_i_add++;
            for (x = 1; x < width - 8; x += 8){
                iv16s FIV_DALIGNED dx1[8];
                iv16s FIV_DALIGNED dy1[8];
                __m128i m_line_f   = _mm_loadl_epi64((__m128i*)&line_i[-1]);
                __m128i m_line_b   = _mm_loadl_epi64((__m128i*)&line_i[1]);
                __m128i m_line_sub = _mm_loadl_epi64((__m128i*)line_i_sub);
                __m128i m_line_add = _mm_loadl_epi64((__m128i*)line_i_add);
                __m128i m_dx = _mm_sub_epi16(_mm_cvtepu8_epi16(m_line_b), _mm_cvtepu8_epi16(m_line_f));
                __m128i m_dy = _mm_sub_epi16(_mm_cvtepu8_epi16(m_line_add), _mm_cvtepu8_epi16(m_line_sub));
                __m128i m_v1 = _mm_add_epi16(_mm_abs_epi16(m_dx), _mm_abs_epi16(m_dy));
                _mm_storeu_si128((__m128i*)ptr_outMag_t, m_v1);
                _mm_store_si128((__m128i*)dx1, m_dx);
                _mm_store_si128((__m128i*)dy1, m_dy);
                ptr_outOri_t[0] = ATAN2_TABLE[dy1[0] + 255][dx1[0] + 255];
                ptr_outOri_t[1] = ATAN2_TABLE[dy1[1] + 255][dx1[1] + 255];
                ptr_outOri_t[2] = ATAN2_TABLE[dy1[2] + 255][dx1[2] + 255];
                ptr_outOri_t[3] = ATAN2_TABLE[dy1[3] + 255][dx1[3] + 255];
                ptr_outOri_t[4] = ATAN2_TABLE[dy1[4] + 255][dx1[4] + 255];
                ptr_outOri_t[5] = ATAN2_TABLE[dy1[5] + 255][dx1[5] + 255];
                ptr_outOri_t[6] = ATAN2_TABLE[dy1[6] + 255][dx1[6] + 255];
                ptr_outOri_t[7] = ATAN2_TABLE[dy1[7] + 255][dx1[7] + 255];
                line_i       += 8;
                line_i_sub   += 8;
                line_i_add   += 8;
                ptr_outMag_t += 8;
                ptr_outOri_t += 8;
```

HOG特征中求梯度的优化

```
static void init_atan2_lut_table()
{
    int dy, dx;
    for (dy = -255; dy <= 255; ++dy) {
        for (dx = -255; dx <= 255; ++dx) {
            ivf32 best_dot = 0;
            int o, best_o = 0;
            for (o = 0; o < 9; o++) {
                ivf32 dot = uu[o] * dx + vv[o] * dy;
                if (dot > best_dot) {
                    best_dot = dot;
                    best_o = o;
                }
                else if (-dot > best_dot) {
                    best_dot = -dot;
                    best_o = o + 9;
                }
            }
            ATAN2_TABLE[dy + 255][dx + 255] = best_o;
        }
    }
}
```

级联分类器

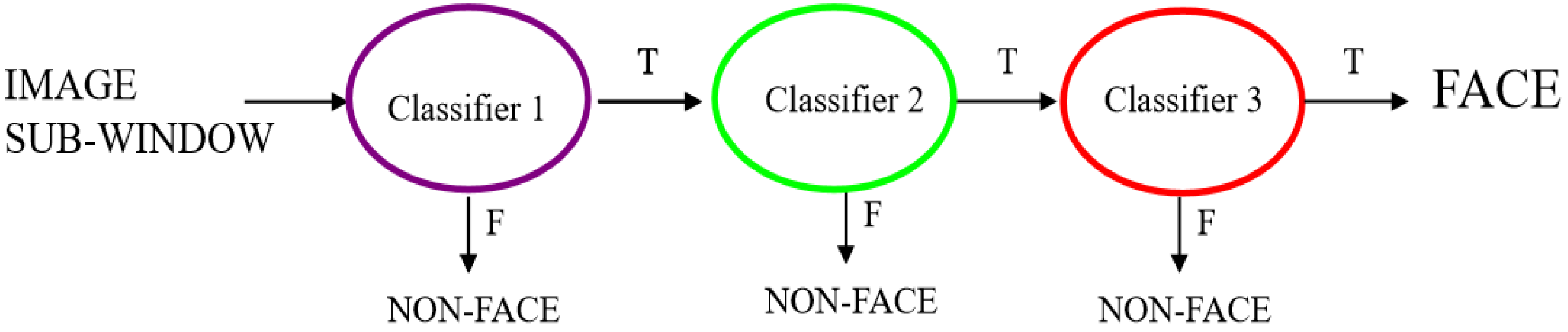
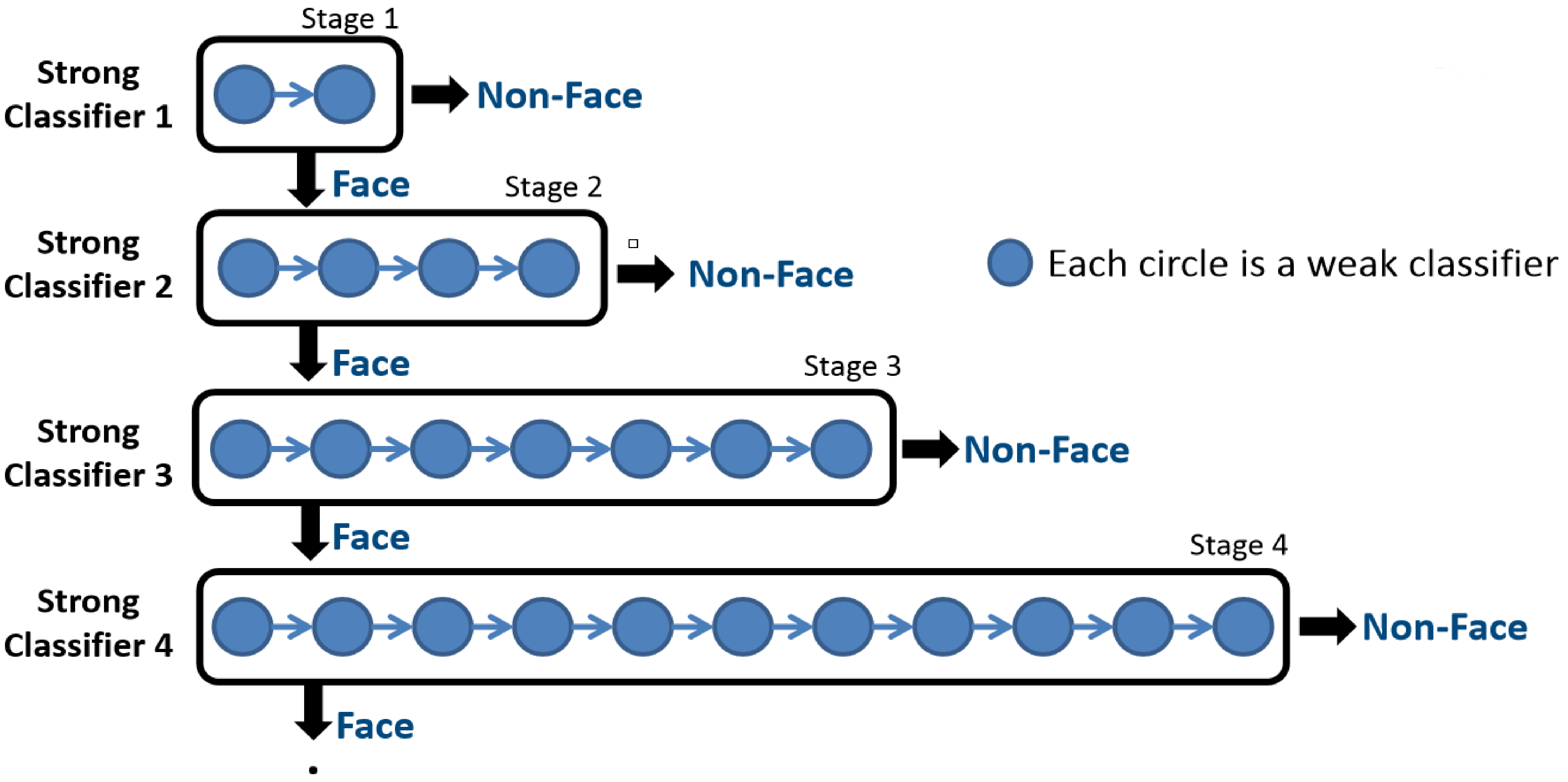


IMAGE
SUB-WINDOW
Classifier 1
T
Classifier 2
T
Classifier 3
T
FACE
F
NON-FACE
F
NON-FACE
F
NON-FACE

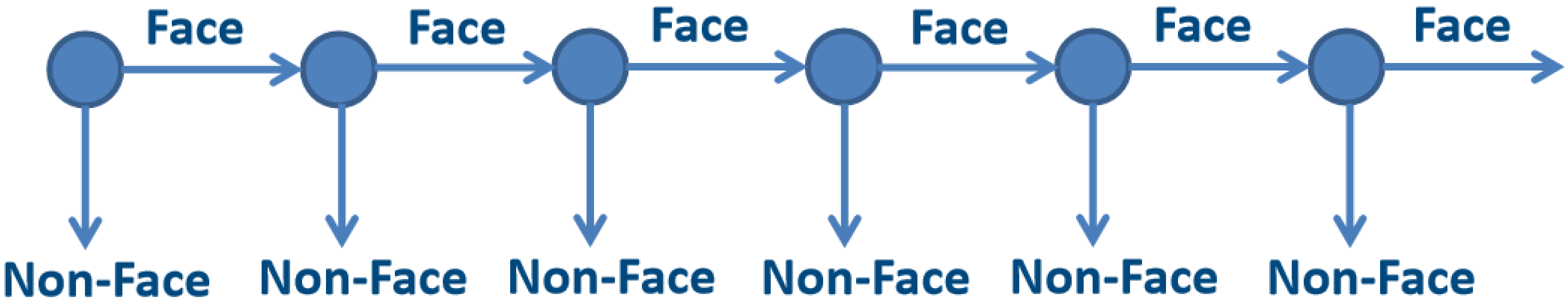
Hard Cascaded



Hard Cascaded Training

- ◆假定有1w张人脸，20w张非人脸样本
- ◆训练过程中正样本保持不变，负样本假定为5W
- ◆训练使用的特征的维度为1000，即只有1000个特征。
- ◆假定采用某种算法挑选了10个特征，用这10个特征训练一个强分类器
- ◆用这个强分类器在全部负样本上进行检测，把得到的误检收集到一起
- ◆训练第二级强分类器，假定采用100个特征。其中只计算90个特征，有10个特征在前一级分类器中已经计算过。
- ◆继续收集负样本，采用更多的特征训练新的强分类器，直到满足退出条件
- ◆经常出现的问题是负样本不足，导致后面很难训练。
- ◆训练强分类器可以用任何机器学习算法，例如可以用神经网络

soft Cascaded




Each circle is a weak classifier

Soft Cascaded Bootstrap Training

- ◆假定有1w张人脸，20w张非人脸样本
- ◆假设每一次训练参与的负样本为10W
- ◆每一次参与训练的正样本始终保持不变
- ◆训练第一个强分类器；训练完成之后在所有的负样本上 检测FP，根据FP的数量确定收集下一次训练需要的负样本
  - 如果FP 的数量大于10W，则根据得分的高低收集那些最容易出错的FP样本
  - 如果FP的数量等于10W，则把这10W的FP收集到一起
  - 如果FP的数量小于10W，大于0，则首先把FP收集起来，在从本轮负样本集合里面随机采样，直到满足10W的负样本
  - 如果FP的数量为0，则结束训练
- ◆ 训练新的强分类器。
- ◆直到达到总共训练的T个强分类器。
- ◆把最后一次训练的强分类器用于检测。

Const Soft cascaded detector

```
bool sampleIsFace(x)
    d ← 0
    for t = 1...T
        d ← d + c_t(x)
        if d < r_t return false
    return true
```

Weak Classifier

Rejection threshold for each weak classifier

Const Soft cascaded detector

bool sampleIsFace(x)

$d \leftarrow 0$

for $t = 1 \ldots T$

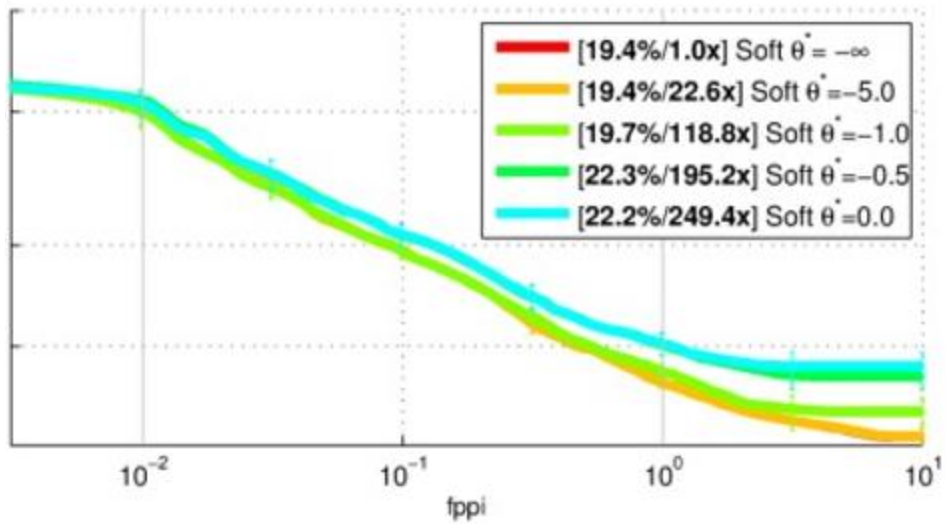
$d \leftarrow d + c_t(x)$

if $d < r_t$ return false

return true

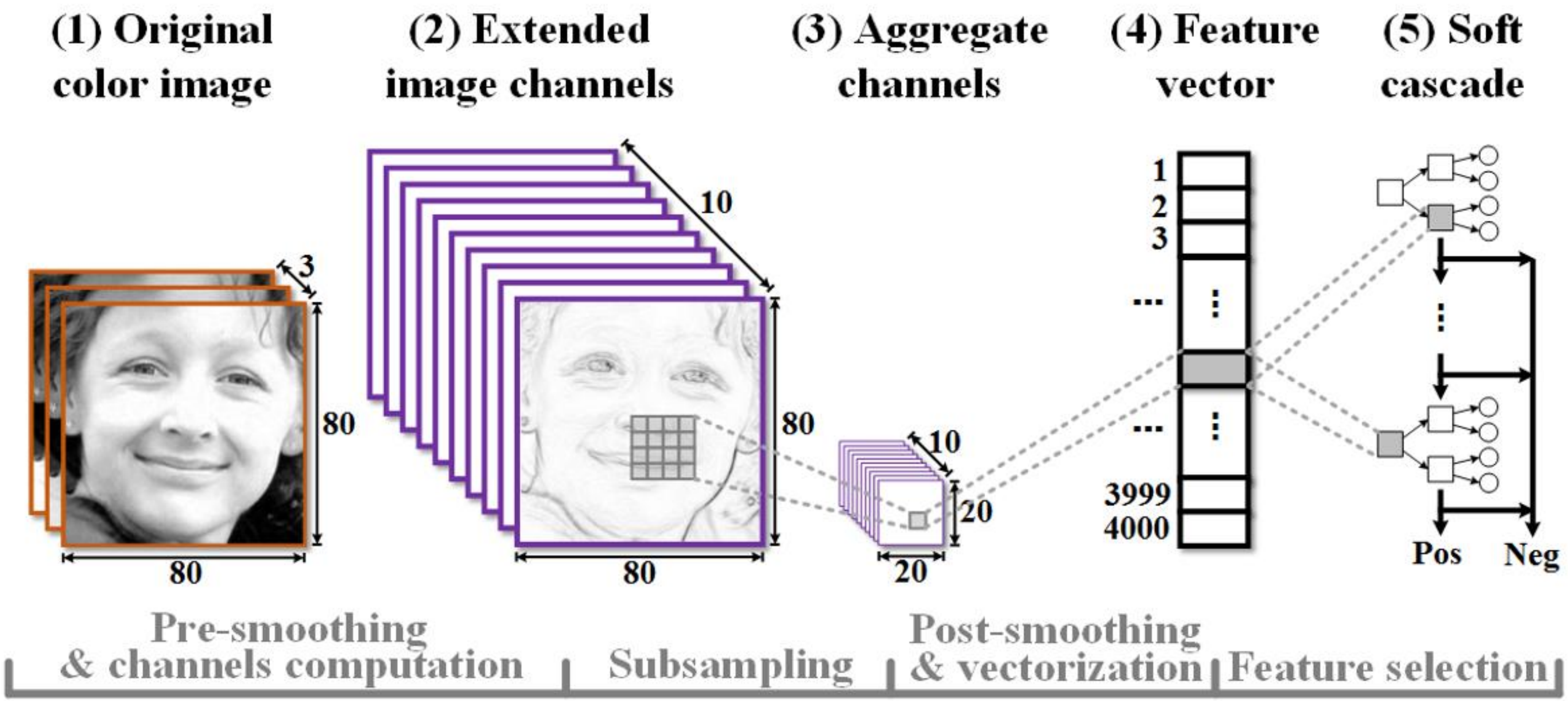
Weak Classifier

Rejection threshold for each weak classifier



调整不同固定阈值的Soft-Cascade对准确率和速率的影响

Aggregate Channel Features for Multi-view Face Detection



算法部署到移动平台的挑战

- ◆在普通I5机器上CPU端单帧处理时间越小越小，通常需要小于10ms。
- ◆模型大小越小越小，通常情况需要小于5M。
- ◆算法运行占用内存越小越好，通常不能大于100M。

Ours method

- ◆特征采用FHOG+LAB+grad，强分类器采用adaboost算法
- ◆5w正样本，20w负样本，在I7 8核台式机上训练不到3分钟的时间
- ◆在640x480的图片上检测一张人脸，尚未采用金字塔特征快速算法，平均检测时间为8ms

参考文献

- Bin Yang. Junjie Yann. Aggregate Channel Features for Multi-view FaceDetection
- PAMI2014 piotr Dollar. Fast Feature Pyramids for Object Detection
- http://vision.ucsd.edu/~pdollar/toolbox/
- http://www.ipol.im/pub/art/2014/104/

陈老师博客

http://blog.csdn.net/celerychen2009