Saliency 模块

显著性检测模块，提取前景区域（或目标），可以用来给目标检测、识别、跟踪等算法提供候选区、缩小检测范围、提速或提高最终效果。

类：

Saliency:

是下面所有类的基类。

StaticSaliency:

StaticSaliencySpectralResidual:

StaticSaliencySpectralResidual类，继承自StaticSaliency，StaticSaliency继承自Saliency类。

算法思想:

通过谱残差法来计算显著性区域，算法思想来源是，所有自然图像的傅里叶变换都遵循$1/f$法则，即频率 f 对应的傅里叶谱的幅度与其频率成反比，数学公式表示如下：

$$E\mathcal{A}(f) \propto 1/f$$

其中符号 $E\cdot$ 表示期望运算。

通过计算图像的傅里叶变换的Log Spetrum（对数谱），比较了单幅图像与2277幅合成图像的Log Spetrum：

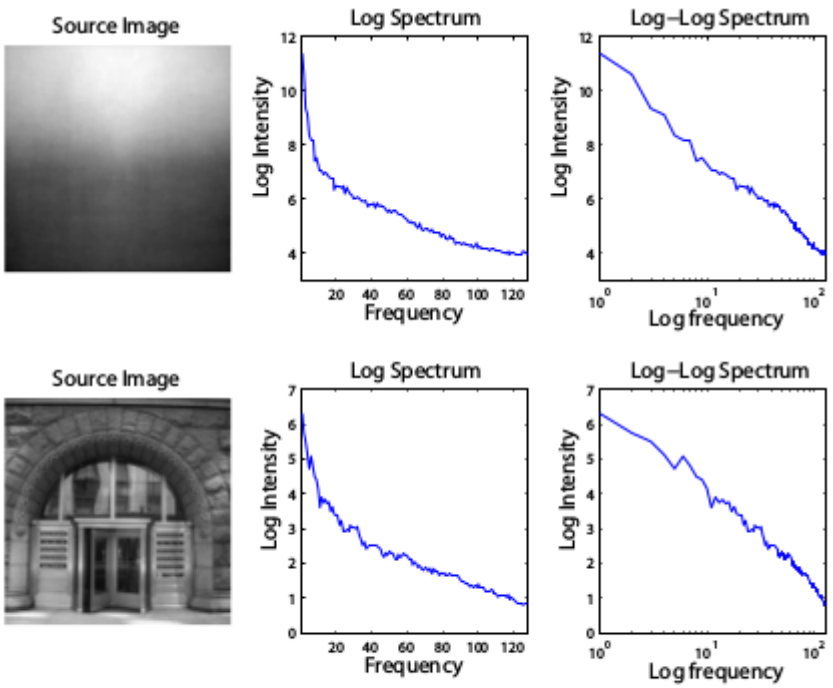


Figure 1. Examples of log spectrum and log-log spectrum. The first image is the average of 2277 natural images.

可以发现，他们具有很高的相似性。

但是也包含差异，因为合成图像更近似于背景，所以合成图像的Log Spectrum较单幅图像更加平滑。而单幅图中的Log Spetrum中的奇异点（与背景有差异的）就对应着图像与背景中不同的部分，即显著性区域。

如何通过单幅图的Log Spetrum找到奇异点呢？

原文作者提供了一种方法。对Log Spetrum通过（均值）滤波的方法可以估计背景的Log Spetrum，再将原图的Log Spetrum减去估计背景的Log Spetrum即可得到显著性区域的Log Spetrum。最后进行傅里叶反变换就能得到显著性区域的空间描述。

最后对显著性区域二值化，得到二值区域。二值化阈值计算如下：

$$threshold = E(S(x)) \times 3$$

$E\cdot$表示取期望，$S(x)$表示计算得到的显著性区域的空间描述。

算法流程:

In sum, given an image $\mathcal{I}(x)$, we have:

$$\mathcal{A}(f) = \Re\Big(\mathfrak{F}\big[\mathcal{I}(x)\big]\Big), \quad (5)$$

$$\mathcal{P}(f) = \Im\Big(\mathfrak{F}\big[\mathcal{I}(x)\big]\Big), \quad (6)$$

$$\mathcal{L}(f) = \log\big(\mathcal{A}(f)\big), \quad (7)$$

$$\mathcal{R}(f) = \mathcal{L}(f) - h_n(f) * \mathcal{L}(f), \quad (8)$$

$$\mathcal{S}(x) = g(x) * \mathfrak{F}^{-1}\Big[\exp\big(\mathcal{R}(f) + \mathcal{P}(f)\big)\Big]^2. \quad (9)$$

where $\mathfrak{F}$ and $\mathfrak{F}^{-1}$ denote the Fourier Transform and Inverse Fourier Transform, respectively. $\mathcal{P}(f)$ denotes the phase spectrum of the image, which is preserved during the process.

1. 对图像进行傅里叶变换；
2. 计算振幅谱$A(f)$；
3. 计算相位谱$P(f)$；
4. 计算振幅的Log谱$L(f)$和将Log谱滤波后的Log谱$h_n(f) \times L(f)$；
5. 计算Log谱残差$R(f)$；
6. 傅里叶反变换并经过高斯滤波后得到显著性空间区域；
7. 对显著性空间区域二值化。

补充:

> 虽然侯晓迪现已明言谱残差理论是错误的，不过还是不能否认这个方法在视觉显著性研究领域起到的重要作用。

在实际测试中，该算法表现是很好的，算法耗时相对于该模块另外三个算法更少。虽然提取的区域不是很精确，但是就像本文开头说的一样，该方法可以作为目标检测或识别的预处理手段，可以用来减少候选区域，或者用来提高检测或识别精度。

API:

C++:

```
//设置图像高度
void setImageHeight(int val);

//设置图像宽度
void setImageWidth(int val);

//获取图像高度
int getImageHeight() const;

//获取图像宽度
int getImageWidth() const;

//读取算法参数文件
void read(const FileNode &fn) CV_OVERRIDE;

//保存算法参数到文件
void write(FileStorage &fs) const CV_OVERRIDE;

//计算图像saliency（显著性）
//参数：
//  image：输入图像，输入图像可以为单通道或三通道
//  saliencyMap：计算出来的saliency，矩阵元素为浮点类型
//成功计算saliency时返回true，否则返回false
bool computeSaliency(InputArray image, OutputArray saliencyMap);

//二值化saliencyMap
//参数：
//  saliencyMap：通过computeSaliency计算得到的saliencyMap
//  binaryMap：将saliencyMap二值化之后的图像
//成功计算binaryMap时返回true，否则返回false,只能输入saliencyMap(浮点型)，否则可能引起段错误
bool computeBinaryMap(InputArray saliencyMap, OutputArray binaryMap)

//生成StaticSaliencySpectralResidual对象
static Ptr<StaticSaliencySpectralResidual> create();
```

Python:

```
#计算显著性区域wenjian
retval, saliencyMap = cv.saliency_StaticSaliencySpectralResidual.computeSaliency(image[,
saliencyMap])

#创建实例
retval = cv.saliency.StaticSaliencySpectralResidual_create()

#获取输入图像高度
retval = cv.saliency_StaticSaliencySpectralResidual.getImageHeight()

#获取输入图像宽度
```

```
retval = cv.saliency_StaticSaliencySpectralResidual.getImageWidth()

#设置输入图像高度
None = cv.saliency_StaticSaliencySpectralResidual.setImageHeight(val)

#设置输入图像宽度
None = cv.saliency_StaticSaliencySpectralResidual.setImageWidth(val)

#读取算法参数文件
None = cv.saliency_StaticSaliencySpectralResidual.read(fn)
```

demo:

注意计算binaryMap时的写法：

```
StaticSaliencySpectralResidual spec;
spec.computeBinaryMap(saliencyMap, binaryMap);
```

使用computeBinaryMap（需要输入浮点型矩阵）方法时，如果输入图像为普通灰度图像（整形），则可能引发段错误。

由于算法内部可能使用了傅里叶正反变换，且计算显著性区域精度不高，所以在实际测试中，考虑了将原图缩放到一个更小的尺寸，用以提高运行速度。

```
Ptr<Saliency> saliencyAlgorithm;
Mat saliencyMap;

//create StaticSaliencySpectralResidual object
saliencyAlgorithm = StaticSaliencySpectralResidual::create();
for (;;)
{
    cap >> frame;
    if (frame.empty())
    {
        return 0;
    }
    //gray frame or bgr frame
    cvtColor(frame, frame, COLOR_BGR2GRAY);

    frame.copyTo(image);

    //resize for speeding up
    Size originSize = image.size();
    resize(image, image, Size(64, 64));

    //saliencyMap's elements are float type
    if (saliencyAlgorithm->computeSaliency(image, saliencyMap))
    {
        //compute the binaryMap
        StaticSaliencySpectralResidual spec;
        spec.computeBinaryMap(saliencyMap, binaryMap);
```

```
        imshow("Saliency Map", saliencyMap);
        imshow("Original Image", frame);

        //mask
        resize(binaryMap, binaryMap, originSize);
        binaryMap &= frame;
        imshow("Binary Map", binaryMap);

        //controler
        char c = waitKey(10);
        if (c == 27)
            break;
    }
}
```

StaticSaliencyFineGrained:

StaticSaliencyFineGrained类，继承自StaticSaliency，StaticSaliency继承自Saliency类。

算法描述:

基于中心环绕差分（center-surround difference）计算显著性区域。

为了方便后续阅读，我们先定义两个基本公式：

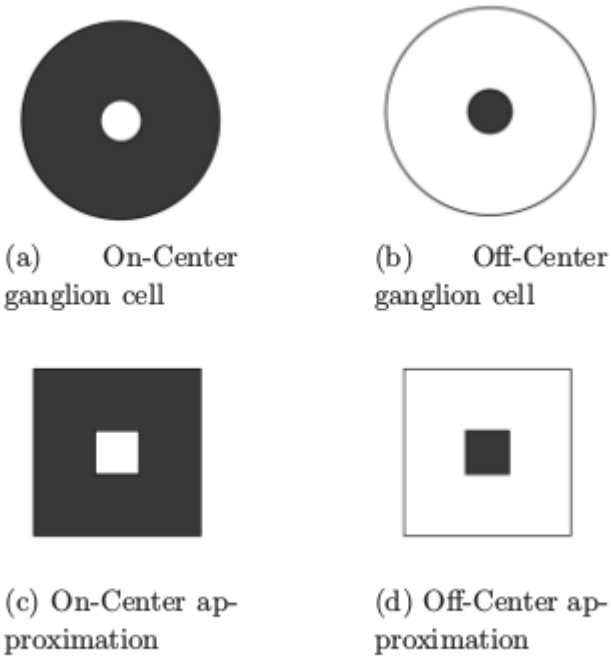
$$I(x,y)=\sum_{x^{'}\leq x,y^{'}\leq y}i(x^{'},y^{'}) \tag{1}$$

其中，$i(x^{'},y^{'})$表示点$(x^{'},y^{'})$处的像素值。

$$rectSum(x_1,y_1,x_2,y_2)=I(x_2,y_2)-I(x_1,y_2)-I(x_2,y_1)+I(x_1,y_1) \tag{2}$$

这表示的是(x_1,y_1)和(x_2,y_2)围成矩形的像素值的和，画画图就能发现了。

接下来是我们的差分模板：



(a) On-Center ganglion cell

(b) Off-Center ganglion cell

(c) On-Center approximation

(d) Off-Center approximation

Fig. 1. On-center and off-center ganglion cells and their approximation on computational models of visual saliency.

圆形差分模板为正规模板，为了简化计算使用正方形差分模板。称为中心环绕差分（center-surround difference），模板大小是需要设定的，采用如下计算公式设定模板大小：

$$\zeta = \delta \times 2^s \tag{0}$$

表示δ表示模板大小，s图像尺度，ζ最终输出的模板大小，由于该算法只采用原图计算saliencyMap，故上述式子实际上可以不试用，列出是因为其他算法会将图像进行不同尺度缩放后在计算saliencyMap，这是本算法和其他算的区别。

使用中，ζ有几个经验值$\{12, 24, 28, 48, 56, 112\}$。

设置完模板大小，现在计算差分。计算公式如下：

$$surround(x, y, \zeta) = \frac{rectSum(x-\zeta, y-\zeta, x+\zeta, y+\zeta) - i(x, y)}{(2\zeta+1)^2 - 1} \tag{3}$$

$$center(x, y) = i(x, y) \tag{4}$$

接下来计算每个模板尺度下的像素的强度：

$$Int_{On}(x, y, \zeta) = max\{center(x, y) - surround(x, y, \zeta), 0\} \tag{5}$$

$$Int_{Off}(x, y, \zeta) = max\{surround(x, y, \zeta) - center(x, y), 0\} \tag{6}$$

将所有模板尺度考虑进去，得到每个像素的综合强度：

$$Int_{On}(x, y) = \sum_{\zeta} Int_{On}(x, y, \zeta) \tag{7}$$

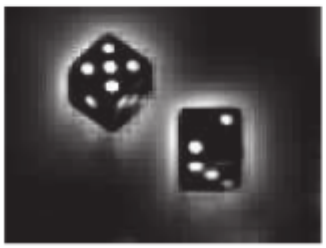
$$Int_{Off}(x, y) = \sum_{\zeta} Int_{Off}(x, y, \zeta) \tag{8}$$

式(7)、(8)即为最终使用不同模板得到的saliencyMap。

效果对比：

与论文中提到的其他算法（此处指的其他算是是在(0)式中考虑了尺度因子的，即构建了原图的尺度金字塔）对比，考虑尺度因子提取到的saliencyMap精度不如直接对原图操作精度高。

(a) Original Image



(b) VOCUS Off-Center Surround

(c) VSF Off-Center Surround

(d) VOCUS On-Center Surround

(e) VSF On-Center Surround

Fig. 6. Comparing Results of VOCUS and VSF. Notice how the detail is preserved in VSF (right) and how the VOCUS results are poorly defined (left).

可见本算法（VFS）较其他算法很大的一个特点在于本算法能够更精细地提取到显著性区域。

在人体检测部分和其他算法对比：

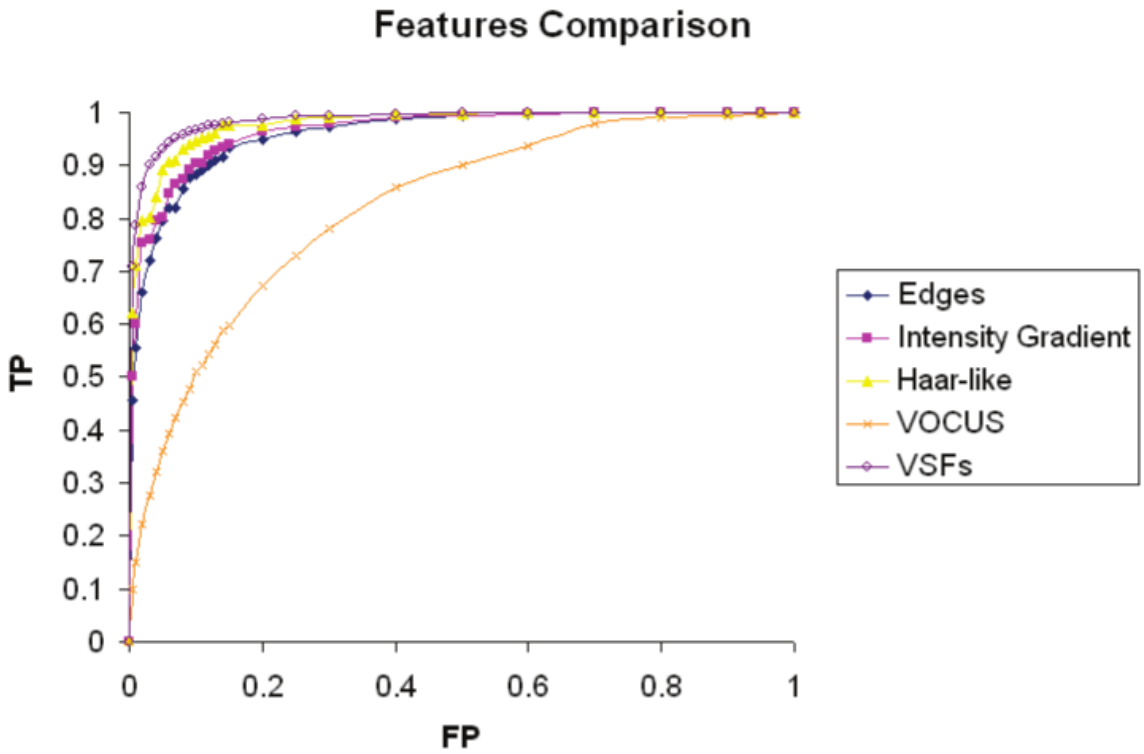


Fig. 16. ROC curves for human detection using different visual features.

在论文的对比测试中，在经过本算法（VSF）预处理后，比其他算法（包括Haar检测器）检测效果好。

API:

C++:

```
//计算图像saliency（显著性）
//参数：
//  image：输入图像，输入图像可以为单通道或三通道
//  saliencyMap：计算出来的saliency，灰度图
//成功计算saliency时返回true，否则返回false
bool computeSaliency(const InputArray image, OutputArray saliencyMap);

//生成StaticSaliencyFineGrained对象
static Ptr<StaticSaliencyFineGrained> create();
```

Python:

```
#计算显著性区域
retval, saliencyMap = cv.saliency_StaticSaliencyFineGrained.computeSaliency(image[,
saliencyMap])

#创建实例
retval = cv.saliency.StaticSaliencyFineGrained_create()
```

demo:

输入图像可以为单通道或三通道。

```
Ptr<Saliency> saliencyAlgorithm;
Mat saliencyMap;

//create StaticSaliencyFineGrained object
saliencyAlgorithm = StaticSaliencyFineGrained::create();
for (;;)
{
    //get a new frame
    cap >> frame;
    if (frame.empty())
    {
        return 0;
    }
    // cvtColor(frame, frame, COLOR_BGR2GRAY);
    frame.copyTo(image);

    //compute saliency map
    if (saliencyAlgorithm->computeSaliency(image, saliencyMap))
    {
        imshow("Saliency Map", saliencyMap);
        imshow("Original Image", image);
     }
}
```

Objectness:

ObjectnessBING:

ObjectnessBING类，继承自Objectness，Objectness继承自Saliency类。

算法描述:

训练:

该方法依赖于训练好的模型，模型通过两个线性SVM分类器得到。

首先，对训练集上的每张图像的所有可能的标注区域，生成不同尺度（一定经验范围内）的样本，计算这些样本与原始样本的重叠率，保留重叠率大于50%的样本，并归一化为8×8大小，记这些样本为X。对样本X进行直方图统计，统计不同尺度下的正样本数，剔除正样本数小于N的尺度。

固定100次随机产生100个备选负样本窗口，将与图片中所有目标重叠率小于50%的备选窗口作为负样本窗口。

计算归一化样本X的L_2范数梯度（NG特征），实现时采用$[-1, 0, 1]$模板计算梯度，用$|gradient(x)| + |gradient(y)|$计算近似$L_2$范数梯度。物体和背景的梯度模式有明显区别，即物体梯度模式比较杂乱，而背景梯度模式比较单一和清楚。归一化到$[0, 255]$后即得到的样本X梯度为G_{NG}。

接下来，对G_{NG}用线性SVM训练，得到参数模型W。通过参数模型W可以为每一个候选区域G'打分，分数越S_l高则越接近目标。数学形式为：

$$S_l = function(W, G)$$

对W及G_{NG}二值化（G_{NG}二值化得到G_{BING}），可将卷积操作转换成位运算操作，从而可加速上式计算过程。

通过非极大抑制（NMS）的方法，可以选择一些建议的尺寸，比如100×100比10×500更可能包含目标区域。训练第二级SVM分类器，计算在每种尺度下的分值（一维）。数学形式为：

$$O_l = V_i \times S_l + T_i$$

第二级SVM分类器用于训练得到模型参数V_i和T_i，V_i和T_i是与尺度相关的，即针对每个有效尺度都需要训练一次。第二级分类器决定了不同尺度下最终分数的权值和偏置。

测试:

测试图像时，对图像进行不同尺度的缩放，取候选区域，计算图像候选区域的NG特征G'_{NG}，二值化后得到二值化赋范特征（BING）G'_{BING}，经过一、二级SVM分类器后通过最终得分判定候选区域是否为目标区域。

说明:

> we scan over a predefined quantized window sizes (scales and aspect ratios).

提取正样本和测试时依然采用滑动窗口的方式，不过候选区域的大小和横纵比是固定的几个（即，一定经验范围内）。

> using non-maximal suppression (NMS), we select a small set of proposals from each size i. Some sizes (e.g. 10 × 500) are less likely than others to contain an object instance (e.g. 100 × 100).

论文中没有提及该非极大抑制该如何使用，个人观点是，对第一级SVM算出来的分数可以做以下合乎常理的判断：对于一个尺度i，如果它是一个好的尺度，那么在尺度i的周围尺度（大小、横纵比相近）中，它评分最高，如果满足上述条件则保留该尺度（或者设定为建议尺度）。

API:

未找到生成训练模型的相关接口。

C++:

```
//下列方法官方demo未涉及，官方tutorial document未做说明
double getBase() const;
void setBase(double val);
int getNSS() const;
void setNSS(int val);
int getW() const;
void setW(int val);
void read();
void write() const;

//设置保存算法运行结果的路径
//内容为：
//   第一行为目标矩形数目
//   其他每一行代表矩形四个角点
//可能因为权限问题导致无法写入，请使用chmod设置必要的权限
```

```
void setBBRessDir(const String& resultsDir);

//返回所有objectness矩形的分数，vector形式
std::vector<float> getobjectnessValues();

//设置已经训练好的模型文件路径
//参数：
//  trainingPath：已经训练好的模型路径
void setTrainingPath(const String& trainingPath);

//计算图像saliency（显著性）
//参数：
//  image：输入图像，输入图像可以为单通道或三通道
//  saliencyMap：计算出来的saliency
//成功计算saliency时返回true，否则返回false
bool computeSaliency(const InputArray image, OutputArray saliencyMap);

//生成ObjectnessBING对象
static Ptr<ObjectnessBING> create();
```

Python:

```
retval = cv.saliency_ObjectnessBING.getBase()
None = cv.saliency_ObjectnessBING.setBase(val)
retval = cv.saliency_ObjectnessBING.getNSS()
None = cv.saliency_ObjectnessBING.setNSS(val)
retval = cv.saliency_ObjectnessBING.getW()
None = cv.saliency_ObjectnessBING.setW(val)
None = cv.saliency_ObjectnessBING.read()
None = cv.saliency_ObjectnessBING.write()

#计算显著性区域
retval, saliencyMap = cv.saliency_ObjectnessBING.computeSaliency(image[, saliencyMap])

#创建实例
retval = cv.saliency.ObjectnessBING_create()

#获取各区域分数
retval = cv.saliency_ObjectnessBING.getobjectnessValues()

#设置模型路径
None = cv.saliency_ObjectnessBING.setTrainingPath(trainingPath)

#设置保存算法运行结果的路径
None = cv.saliency_ObjectnessBING.setBBResDir(resultsDir)
```

demo:

```
Ptr<Saliency> saliencyAlgorithm;

//create ObjectnessBING object
```

```
saliencyAlgorithm = ObjectnessBING::create();
vector<Vec4i> saliencyMap;

//set the trained model path
saliencyAlgorithm.dynamicCast<ObjectnessBING>()->setTrainingPath(training_path);
//set the output path
saliencyAlgorithm.dynamicCast<ObjectnessBING>()->setBBResDir("Results");
// saliencyAlgorithm.dynamicCast<ObjectnessBING>()->write();
for (;;)
{
    cap >> frame;
    if (frame.empty())
    {
        return 0;
    }
    frame.copyTo(image);

    //compute candidate bunding boxes
    if (saliencyAlgorithm->computeSaliency(image, saliencyMap))
    {
        int ndet = int(saliencyMap.size());
        std::cout << "Objectness done " << ndet << std::endl;
        // The result are sorted by objectness. We only use the first maxd boxes here.
        int maxd = 7, step = 255 / maxd, jitter = 9; // jitter to seperate single rects
        Mat draw = image.clone();
        for (int i = 0; i < std::min(maxd, ndet); i++)
        {
            Vec4i bb = saliencyMap[i];
            Scalar col = Scalar(((i * step) % 255), 50, 255 - ((i * step) % 255));
            Point off(theRNG().uniform(-jitter, jitter), theRNG().uniform(-jitter,
jitter));
            rectangle(draw, Point(bb[0] + off.x, bb[1] + off.y), Point(bb[2] + off.x,
bb[3] + off.y), col, 2);
            rectangle(draw, Rect(20, 20 + i * 10, 10, 10), col, -1); // mini temperature
scale
        }
        imshow("BING", draw);
        // imshow("saliencyMap", saliencyMap);
        char c = waitKey(10);
        if (c == 27)
            break;
    }
    else
    {
        std::cout << "No saliency found for " << video_name << std::endl;
    }
}
```

MotionSaliency:

MotionSaliencyBinWangApr2014:

MotionSaliencyBinWangApr2014类，继承自MotionSaliency，MotionSaliency继承自Saliency类。

算法描述：

用于提取运动图像（帧流）的前景。

需要迭代，即在运动图像（帧流）中，需要较长时间（768x576大小需要1000ms以上）迭代才有显著效果。

像素分类：

算法使用$K+1$个背景模板$T_0 \dots T_K$，每个模板包含背景值$B_N(i,j)$和权值$C_N(i,j)$，其中$C_N(i,j)$从T_0到T_K递减。即使得T_0包含了存在最久的背景信息，$T_1 \dots T_k$包含了存在时间较短的背景信息，并依次递减，所以$T_N(N \neq 0)$的$C_N(i,j)$在某个时刻的值可能为0。

对于输入图像，所有像素默认为前景，如果某个像素P和模板T_N匹配时，则将该像素判定为背景像素，否则为候选前景像素。该模板的$B_N(i,j)$将被像素P的像素值通过某种算法（running average function）更新，并且与学习率α有关。所有模板中，第一个将像素P判定为背景像素的模板的$C_N(i,j) = C_N(i,j) + 1$，其他模板$C_N(i,j) = C_N(i,j) - 1$，当某个模板的所有$C_N(i,j) = 0$时，该模板不再参与后续处理，等待被新的模板替换。

为了增强算法对于环境中微小的高频信号改变的适应性，将输入图像和模板T_0、T_1降采样为一个$M \times M$的方阵，对像素的前景或背景的判定方法和上述相同，不过不再更新$C_N(i,j)(N=0,1)$和$B_N(i,j)(N=0,1)$。完成低分辨率下的所有像素的判断之后，将低分辨率的判定结果重新设定（resize，论文中未指定方法）为初始分辨率。

只有上述两种方法都将同一个像素判定为前景时，才最终判定该像素为前景。

背景模型：

为了适应背景突变的情况，需要一种算法来将新的背景值加入背景模板，并且能够移除无效的背景模板。

更新$C_N(i,j)$后需要对$T_1 \dots T_K$排序，使得$C_N(i,j)$值从模板$T_1 \dots T_k$顺序满足递减条件。

对T_0和T_1，如果T_1的$C_1(i,j)$值长期存在，可以通过$C_1(i,j) > \theta_L$且$C_0(i,j) < \theta_L$判断，则交换T_0和T_1对应部分，C_0变为$\gamma \times \theta_L$。（即T_1中的$C_1(i,j)$比T_0中$C_0(i,j)$的更有作用）后续T_0与T_1的更新将与γ有关（openCV中未提供修改接口）。

对于帧流中的每一个像素会设定一个隐藏的$C_A(i,j)$和$B_A(i,j)$。如果该像素被判定为前景，就会计算该像素与之对应的$B_A(i,j)$之间的距离$L_A(i,j)$（被判定为前景的像素与背景之间的距离），如果$L_A(i,j)$小于阈值ϵ则$C_A(i,j) = C_A(i,j) + 1$，否则$C_A(i,j) = C_A(i,j) - 1$。如果$C_A(i,j) = 0$并且当前像素被判定为前景像素，则将$B_A(i,j)$置为当前像素像素值，令$C_A(i,j) = 1$；否则，如果$C_A(i,j) > \theta_A$，则将$B_A(i,j)$置为像素分类的判定值（$B_N(i,j)$）。伪代码描述如下：

```
#计算距离
Dist = distance(P(i, j), Ba(i, j))

#判定距离，计算Ca
if(Dist <= La):
    Ca += 1
else:
    Ca -= 1

#Ca为0则置Ba为对应像素像素值，且初始化Ca=1
if(Ca == 0):
    Ba(i, j) = P(i , j)
```

```
    Ca = 1
elif(Ca > Theta):
#Ca大于设定阈值，则将Ba置为像素判定值（Bk）
    B(i ,j) = Ba(i, j)
```

该部分实际应对了背景变化的情况，比如在背景中放置一个物体，初始时算法可以将该物体判定为前景，但是如果物体不动，随着时间的推移，可以认为该物体成为了背景一部分，此时上述方法即可做到将一段时间内未移动的前景判定为背景。

API:

C++:

```
//设置图像大小
//参数：
//  W:  输入图像宽度
//  H:  输入图像高度
//当输入图像大小与设置大小不匹配时，会引发段错误。
void setImageSize(int W, int H);

//设置图像宽度
//和setImageSize(int W , int H)效果一样，可以直接使用setImageSize(int W , int H)设置。
void setImageWidth(int val);

//设置图像高度
//和setImageSize(int W , int H)效果一样，可以直接使用setImageSize(int W , int H)设置。
void setImageHeight(int val);

//初始化设置
//在设置图像大小之后调用
bool init();

//计算图像saliency（显著性）
//参数：
//  image：输入图像，需要为灰度图
//  saliencyMap：计算出来的saliency，矩阵元素值为0或1
//成功计算saliency时返回true，否则返回false
bool computeSaliency(const InputArray image, OutputArray saliencyMap);

//生成MotionSaliencyBinWangApr2014对象
static Ptr<MotionSaliencyBinWangApr2014> create();
```

Python:

```
#计算显著性区域
retval, saliencyMap = cv.saliency_MotionSaliencyBinWangApr2014.computeSaliency( image[,
saliencyMap])

#创建实例
retval = cv.saliency.MotionSaliencyBinWangApr2014_create()
```

```
#获取输入图像高度
retval = cv.saliency_MotionSaliencyBinWangApr2014.getImageHeight()

#获取输入图像宽度
retval = cv.saliency_MotionSaliencyBinWangApr2014.getImageWidth()

#初始化
retval = cv.saliency_MotionSaliencyBinWangApr2014.init()

#设置输入图像高度
None = cv.saliency_MotionSaliencyBinWangApr2014.setImageHeight(val)

#设置输入图像宽度
None = cv.saliency_MotionSaliencyBinWangApr2014.setImageWidth(val)

#设置输入图像长宽
None = cv.saliency_MotionSaliencyBinWangApr2014.setImagesize(W, H)
```

demo:

输入图像只能为单通道图像，否则会引发错误。

```
Ptr<Saliency> saliencyAlgorithm;
Mat saliencyMap;

//create MotionSaliencyBinWangApr2014 object
saliencyAlgorithm = MotionSaliencyBinWangApr2014::create();
//set image size
saliencyAlgorithm.dynamicCast<MotionSaliencyBinWangApr2014>()->setImagesize(image.cols,
image.rows);
// saliencyAlgorithm.dynamicCast<MotionSaliencyBinWangApr2014>()-
>setImageHeight(image.rows);
// saliencyAlgorithm.dynamicCast<MotionSaliencyBinWangApr2014>()-
>setImageWidth(image.cols);
//initialize the settings
saliencyAlgorithm.dynamicCast<MotionSaliencyBinWangApr2014>()->init();

for(;;)
{
    //get a frame
    cap >> frame;
    //while the frame is empty, exit
    if (frame.empty())
    {
        return 0;
    }
    //convert bgr frame to gray
    cvtColor(frame, frame, COLOR_BGR2GRAY);

    //compute sliency map(mask)
    saliencyAlgorithm->computeSaliency(frame, saliencyMap);
```

```
    imshow("image", frame);
    imshow("saliencyMap", saliencyMap * 255);
}
```

参考资料：

StaticSaliencySpectralResidual:

1. OpenCV中显著性检测算法的使用
2. OpenCV实现显著性检测中的谱残差法（Spectral Residual Method）涉及到了傅立叶正反变换
3. 图像显著性论文（二）—Saliency Detection: A Spectral Residual Approach
4. Spectral Residual 小记
5. Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

StaticSaliencyFineGrained:

1. Sebastian Montabone and Alvaro Soto. Human detection using a mobile platform and novel features derived from a visual saliency mechanism. In *Image and Vision Computing, Vol. 28 Issue 3*, pages 391–402. Elsevier, 2010.

ObjectnessBING:

1. Ming-Ming Cheng, Ziming Zhang, Wen-Yan Lin, and Philip Torr. Bing: Binarized normed gradients for objectness estimation at 300fps. In *IEEE CVPR*, 2014.
2. CVPR 2014 ObjectnessBING 原文翻译
3. 目标检测（一）--Objectness算法总体理解，整理及总结

MotionSaliencyBinWangApr2014:

1. Bin Wang and Piotr Dudek. A fast self-tuning background subtraction algorithm. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 401–404. IEEE, 2014.