

20180412 PHash 以图搜图

另外两个关于以图搜图的原理介绍：

1. http://www.ruanyifeng.com/blog/2011/07/principle_of_similar_image_search.html
2. http://www.ruanyifeng.com/blog/2011/07/principle_of_similar_image_search.html

原文：<https://blog.csdn.net/luoweifu/article/details/8220992>

这是我第一次翻译外文文章，如果翻译的不好，还望大家多包含！以下黑色部分是作者原文的翻译，红色部分是我本人自己的理解和对其的补充。

原文：[Looks Like It](#)

在google里对



的搜索结果是

外观类似的图片 - 举报图片



下面是我用pHash算法(java)实现的结果:

十张比较的图如下:



source: f0a0000030400000

1-5 2-5 3-0 4-5 5-5 6-5 7-5 8-7 9-6 10-3 11-5

f0a0000030400000是原图片的指纹数

下面的一行“a-b”型的数据，a表示序号，b表示汉明距离，b越小就越相似；汉明距离 ≤ 5 表示很相似。

实现源码下载：

<http://download.csdn.net/download/luoweifu/4807319>

-----译文-----

在过去的几个月，我不停地寻求“[TinEye](#) 如何工作”的答案，或者说它是如何搜索图片的。

结果是我仍没法知道TinEye图片搜索引擎是如何工作的，他们并没有公开他们所用使用的算法细节。然而，根据它返回的结果，呈现给我的是感知哈希算法的一个变种。

这是有感知的

感知哈希(hash)算法描述了一个有可比较的哈希函数的类。图像特征被用于生成独特的（但不是唯一的）指纹，而这些指纹是可比较的。

感知哈希与像MD5和SHA1这样的加密哈希（散列）函数是不同的概念。加密哈希的hash值是随机的，数据用于生成像随机数种子的散列行为，所以相同的数据会产生相同的结果，不同的数据会产生不同的结果。比较两个SHA1的hash值，实际上只告诉我们两个东西，如果hash值是不同的，则数据也是不同的；如果hash值是相同的，则数据是相似的。（因为可能存在hash冲突，相同的hash值会产生不同的数据）。相比之下，感知哈希是可比较的——给你一种两个数据集之间相似的感觉。

我遇到的每一个感知哈希算法都有一个共同的特征：图片可以被放大或缩小，有不同的纵横比，甚至轻微的着色差异(对比度、亮度等)，它们依然能够匹配相似的图片，TinEye也有同样的性能。（但TinEye似乎做了更多，我稍后会去了解）

美丽之道

如何创建感知哈希呢？有一些常见的算法，但没有一个是很复杂的。（我总是很惊讶，为什么如此简单却几乎所有的常见算法都能工作）。最简单的算法之一应该是基于低频的均值哈希。

一张高频率的图片可以提供详细的信息，而低频率的图片只显示一个框架；一张大的，详细的图片有很高的频率，而小图片缺乏图像细节，所以都是低频的。为了演示均值哈希算法如何工作，我将使用我妻子——Alyson Hannigan的图片。

1. 缩小尺寸

去除高频和细节的最快方法是缩小图片，将图片缩小到8x8的尺寸，总共64个像素。不要保持纵横比，只需将其变成8*8的正方形。这样就可以比较任意大小的图片，摒弃不同尺寸、比例带来的图片差异。



2. 简化色彩

将8*8的小图片转换成灰度图像，将64个像素的颜色(red,green,blue)转换成一种颜色（黑白灰度）。

3. 计算平均值


计算所有64个像素的灰度平均值。

4. 比较像素的灰度

将每个像素的灰度，与平均值进行比较。大于或等于平均值，记为1；小于平均值，记为0。

5. 计算hash值

将上一步的比较结果，组合在一起，就构成了一个64位的整数，这就是这张图片的指纹。组合的次序并不重要，只要保证所有图片都采用同样次序就行了。（我设置的是从左到右，从上到下用二进制保存）。


$$\text{hash} = 8f373714acfcf4d0$$

如果图片放大或缩小，或改变纵横比，结果值也不会改变。增加或减少亮度或对比度，或改变颜色，对hash值都不会太大的影响。最大的优点：计算速度快！

如果你想比较两张图片，为每张图片构造hash值并且计算不同位的个数。**(汉明距离)**如果这个值为0，则表示这两张图片非常相似，如果汉明距离小于5，则表示有些不同，但比较相近，如果汉明距离大于10则表明完全不同的图片。

效果更佳的pHash

虽然均值哈希更简单且更快速，但是在比较上更死板、僵硬。它可能产生错误的漏洞，如果有一个伽马校正或颜色直方图被用于到图像。这是因为颜色沿着一个非线性标尺 - 改变其中“平均值”的位置，并因此改变哪些高于/低于平均值的比特数。

一个更健壮的算法叫pHash，(我使用的是自己改进后的算法，但概念是一样的) pHash的做法是将均值的方法发挥到极致。使用离散余弦变换(DCT)降低频率。

1.缩小尺寸

pHash以小图片开始，但图片大于88，3232是最好的。这样做的目的是简化了DCT的计算，而不是减小频率。

2.简化色彩

将图片转化成灰度图像，进一步简化计算量。

3.计算DCT

DCT是把图片分解频率聚集和梯形状，虽然JPEG使用88的DCT变换，在这里使用3232的DCT变换。

4.缩小DCT

虽然DCT的结果是3232大小的矩阵，但我们只要保留左上角的88的矩阵，这部分呈现了图片中的最低频率。

5.计算平均值

如同均值哈希一样，计算DCT的均值，

6.进一步减小DCT

这是最主要的一步，根据8*8的DCT矩阵，设置0或1的64位的hash值，大于等于DCT均值的设为“1”，小于DCT均值的设为“0”。结果并不能告诉我们真实性的低频率，只能粗略地告诉我们相对于平均值频率的相对比例。只要图片的整体结构保持不变，hash结果值就不变。能够避免伽马校正或颜色直方图被调整带来的影响。

7.构造hash值

将64bit设置成64位的长整型，组合的次序并不重要，只要保证所有图片都采用同样次序就行了。将3232的DCT转换成3232的图像。



= 8a0303f6df3ec8cd

与均值哈希一样，pHash同样可以用汉明距离来进行比较。(只需要比较每一位对应的位置并计算不同的位的个数)

同类中的最佳算法？

自从我做了大量关于数码照片取证和巨幅图片的收集工作之后，我需要一种方法来搜索图片，所以，我用了一些不同的感知哈希算法做一个图片搜索工具，根据我并不很科学但长期使用的经验来看，我发现均值哈希比pHash显著地要快。如果你找一些明确的东西，均值Hash是一个极好的算法，例如，我有一张图片的小缩略图，并且我知道它的大图存在于一个容器的某个地方，均值哈希能算法快速地找到它。然而，如果图片有些修改，如过都添加了一些内容或头部叠加在一起，均值哈希就无法处理，虽然pHash比较慢，但它能很好地容忍一些小的变型(变型度小于25%的图片)。

其次，如果，你运行的服务器像TinEye这样，你就可以不用每次都计算pHash值，我确信它们肯定之前就把pHash值保存在数据库中，核心的比较系统非常快，所以只需花费一次计算的时间，并且几秒之内能进行成千上百次的比较，非常有实用价值。

改进

有许多感知哈希算法的变形能改进它的识别率，例如，在减小尺寸之前可以被剪裁，通过这种方法，主体部分周围额外的空白区域不会产生不同。也可以对图片进行分割，例如，你有一个人脸识别算法，然后你需要计算每张脸的hash值，

可以跟踪一般性的着色(例如，她的头发比蓝色或绿色更红，而背景比黑色更接近白色)或线的相对位置。

如果你能比较图片，那么你就可以做一些很酷的事情。例如，你可以在GazoPa搜索引擎拖动图片，和TinEye一样，我并不知道GazoPa工作的细节，然而它似乎用的是感知哈希算法的变形，由于哈希把所有东西降低到最低频率，我三个人物线条画的素描可以和和其它的图片进行比较——如匹配含有三个人的照片。

算法实现

关于均值哈希算法的实现，请参考：[Google 以图搜图 - 相似图片搜索原理 - Java实现](http://blog.csdn.net/luoweifu/article/details/7733030)
<http://blog.csdn.net/luoweifu/article/details/7733030>

下面详细讲一下pHash算法的实现

基体的步骤已经在“效果更佳的pHash”中讲了，下面对应地给出java代码的实现：

1.缩小尺寸

```
/**
 * 局部均值的图像缩小
 * @param pix 图像的像素矩阵
 * @param w 原图像的宽
 * @param h 原图像的高
 * @param m 缩小后图像的宽
 * @param n 缩小后图像的高
 * @return
 */
public static int[] shrink(int[] pix, int w, int h, int m, int n) {

    float k1 = (float) m / w;
```

```

float k2 = (float) n / h;
int ii = (int)(1 / k1); // 采样的行间距
int jj = (int)(1 / k2); // 采样的列间距
int dd = ii * jj;
// int m=0 , n=0;
// int imgType = img.getType();
int[] newpix = new int[m * n];

for (int j = 0; j < n; j++) {
    for (int i = 0; i < m; i++) {
        int r = 0, g = 0, b = 0;
        ColorModel cm = ColorModel.getRGBdefault();
        for (int k = 0; k < jj; k++) {
            for (int l = 0; l < ii; l++) {
                r = r
                    + cm.getRed(pix[(jj * j + k) * w
                        + (ii * i + l)]);
                g = g
                    + cm.getGreen(pix[(jj * j + k) * w
                        + (ii * i + l)]);
                b = b
                    + cm.getBlue(pix[(jj * j + k) * w
                        + (ii * i + l)]);
            }
        }
        r = r / dd;
        g = g / dd;
        b = b / dd;
        newpix[j * m + i] = 255 << 24 | r << 16 | g << 8 | b;
        // 255<<24 | r<<16 | g<<8 | b 这个公式解释一下，颜色的RGB在内存中是
        // 以二进制的形式保存的，从右到左1-8位表示blue，9-16表示green，17-24表示red
        // 所以"<<24" "<<16" "<<8"分别表示左移24,16,8位

        // newpix[j*m + i] = new Color(r,g,b).getRGB();
    }
}
return newpix;
}

```

2.简化色彩

```

/**
 * 将图片转化成黑白灰度图片
 * @param pix 保存图片像素
 * @param iw 二维像素矩阵的宽
 * @param ih 二维像素矩阵的高
 * @return 灰度图像矩阵
 */
public static int[] grayImage(int pix[], int w, int h) {
    //int[] newPix = new int[w*h];
    ColorModel cm = ColorModel.getRGBdefault();

    for(int i=0; i<h; i++) {

```



```

        for(int j=0; j<w; j++) {
            //0.3 * c.getRed() + 0.58 * c.getGreen() + 0.12 * c.getBlue()
            pix[i*w + j] = (int) (0.3*cm.getRed(pix[i*w + j]) + 0.58*cm.getGreen(pix[i*w +
j]) + 0.12*cm.getBlue(pix[i*w + j]));
        }
    }
    return pix;
}

```

3.计算DCT

这一部分请参考我的上一篇博客：[离散余弦变换\(含源码\)](#)

4.缩小DCT

DCT的结果是3232大小的矩阵，但我们只要保留左上角的88的矩阵，所以只需要设置两层的for循环是从0到7就可以了。

5.计算平均值

```

/**
 * 求灰度图像的均值
 * @param pix 图像的像素矩阵
 * @param w 图像的宽
 * @param h 图像的高
 * @return 灰度均值
 */
private static int averageGray(int[] pix, int w, int h) {
    int sum = 0;
    for(int i=0; i<h; i++) {
        for(int j=0; j<w; j++) {
            sum = sum+pix[i*w + j];
        }
    }
    return (int)(sum/(w*h));
}

```

6.构造hash值

```

StringBuilder sb = new StringBuilder();
for(int i=0; i<FHEIGHT; i++) {
    for(int j=0; j<FWIDTH; j++) {
        if(dctPix[i*FWIDTH + j] >= avrPix) {
            sb.append("1");
        } else {
            sb.append("0");
        }
    }
}

```

```
}  
//System.out.println(sb.toString());  
long result = 0;  
if(sb.charAt(0) == '0') {  
    result = Long.parseLong(sb.toString(), 2);  
} else {  
    //如果第一个字符是1, 则表示负数, 不能直接转换成long,  
    result = 0x8000000000000001 ^ Long.parseLong(sb.substring(1), 2);  
}  
  
sb = new StringBuilder(Long.toHexString(result));  
if(sb.length() < 16) {  
    int n = 16-sb.length();  
    for(int i=0; i<n; i++) {  
        sb.insert(0, "0");  
    }  
}
```