List of algorithms used in graphical analysis

Rafael De Smet

December 06, 2016

1 Algorithms

1.1 Edge Detection

Edge detection algorithms all use what are called convolution kernels. A kernel in image processing is a small matrix used to apply effects to an image, such as blurring, outlining. Here we will see kernels used for edge detection only. Listed below are six of the best and most used algorithms.

- Sobel
- Frei-Chen
- Prewitt
- Roberts Cross
- LoG
- Scharr

1.1.1 Convolution kernel

Since all the algorithms are based on the mathematical principle of convolution, some explanation of these convolution kernels is in order.

Convolution is the technique of multiplying together two arrays of different size but of the same dimension. One of the two arrays used in the calculation is the numerical representation of the image (pixels) on which we

want to perform the edge detection algorithm. The second array is called the kernel and is usually much smaller (but in the same dimension).

Each pixel of the image is added to its local neighbours, weighted by the kernel. This produces a new image. If the kernel is chosen wisely, we get all the edges found in the image.

Mathematically we can write the convolution as follows, with O the output image, I the input image and K the kernel:

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1) K(k,l) \tag{1}$$

1.1.2 Sobel

This algorithm performs a 2D spatial gradient measurement and finds regions of 'high spatial frequency' or edges. It uses two 3x3 kernels, one kernel is used for the vertical edges and the other for the horizontal edges in the image. These two kernels can be applied separately and afterwards combined together to find the absolute magnitude of the gradient.

The horizontal kernel: $\begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix}$ and the vertical kernel: $\begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$

1.1.3 Frei-Chen

The Frei-Chen algorithm also uses 3x3 kernels, but this time there are nine different convolution kernels. The four first matrices, G1, G2, G3, G4, are used for edges, the next four are used for lines and the last one is used to compute averages.

$$G_1 = \frac{1}{2\sqrt{2}} \begin{vmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{vmatrix} \quad G_2 = \frac{1}{2\sqrt{2}} \begin{vmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{vmatrix} \quad G_3 = \frac{1}{2\sqrt{2}} \begin{vmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{vmatrix}$$

$$G_4 = \frac{1}{2\sqrt{2}} \begin{vmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{vmatrix} \quad G_5 = \frac{1}{2} \begin{vmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{vmatrix} \quad G_6 = \frac{1}{2} \begin{vmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{vmatrix}$$

$$G_7 = \frac{1}{6}\begin{vmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{vmatrix} \qquad G_8 = \frac{1}{6}\begin{vmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{vmatrix} \qquad G_9 = \frac{1}{3}\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

1.1.4 Prewitt

This algorithm is very similar to the Sobel and Frei-Chen algorithms. Again, two kernels are used, one for the horizontal and one for the vertical edges. Afterwards, they are combined together to get all the edges in the image.

This time the kernels are considerably simpler:

$$\text{Horizontal filter} = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{vmatrix} \qquad \text{Vertical filter} = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix}$$

Because the kernels are simpler, the result is not so accurate when compared to the other algorithms.

1.1.5 Roberts Cross

This algorithm uses even simpler kernels than Prewitt does. This time we use two 2x2 kernels. These kernels correspond to the edges running at 45° to the pixel grid, one for each of the two perpendicular orientations.

$$\text{Horizontal filter} = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} \qquad \text{Vertical filter} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

1.1.6 LoG

This algorithm combines two methods, the Gaussian filtering method [1] and the Laplacian method for edge detection[2]. Hence the name "Laplacian of Gaussian" (LoG). The edge points of an image are detected by finding the zero crossings of the 2nd derivative of the image intensity. Because the 2nd derivative is very sensitive to noise, which could give us bad results, the Gaussian filter is used to clear the noise from the image.

[1] The Gaussian filter is used to blur images and remove noise and detail.

[2] This method highlights regions in the image of rapid intensity change, so it is useful for edge detection.

The R library OpenImageR[3] uses the following LoG mask.

$$\text{LoG mask} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

1.1.7 Scharr

This algorithm is an extension of the Sobel algorithm. Although the Sobel kernels are very good, they do not have perfect rotational symmetry. This is what the Scharr kernels try to optimize.

The most frequently used kernels are the following:

$$\text{Horizontal filter} = \begin{vmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{vmatrix} \qquad \text{Vertical filter} = \begin{vmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{vmatrix}$$

1.2 Color Analysis

Only edge detection algorithms are not going to be sufficient to get enough data from the image to form a musical interpretation. We will need some form of color analysis. This section proposes some algorithms and methods we can use to obtain information about the colors of the image.

1.2.1 RGB

This is the most common representation of color on a computer. Each pixel is described using three values, the amount of red (R), green (G) and blue (B) it has. Using these values we can count how many pixels in the image are dominantly red, green or blue. When we're going to translate the image data into musical patterns, we can match each color to another kind of music (happy, sad, etc...)

1.2.2 HSV

HSV, sometimes called HSB, is another representation of a pixel. This time we use the hue (H), the saturation (S) and the brightness or value (B or V) to

[3] https://cran.r-project.org/web/packages/OpenImageR/OpenImageR.pdf

describe the pixel. Where the RGB model consists of three values indicating the amounts of a certain color each pixel has, the HSV model consists of three independent components.

First we have the hue, which is basically the color. Here the color is represented using a circle with all the colors on it. So this means the value of this component is the degrees of the angle we have to make on the circle to get this color.

The saturation indicates the fullness of the color. This value is expressed in a percentage, with 0% a gray, flat color and 100% is a full, rich color.

The value or brightness indicates the lightness of the color and is also expressed in a percentage, 0% is black and 100% is white.

This representation will be used in combination with the RGB model in the translation into the musical patterns.

1.2.3 CMYK

CMYK is another representation of color, based on the mixing of four colors, cyan (C), magenta (M), yellow (Y) and key (K, black). This model is used frequently with printing.
When combined, the data from the three models can be very accurate to use in the translation.

1.2.4 Black And White Balance

During the image analysis we will convert the image to a gray scale image, so we can determine the amount of white and black in it. Using the RGB model, this is an easy process. Black in the RGB model is (0,0,0) and white is (255,255,255). So we can simply count the amount of pixels that are very close to those two values and we know the amount of black and white pixels in the image.
Analogous to this we can count every gray pixel in the image, by finding every pixel where the RGB values are exactly the same.

1.3 Image Hashing

It is also possible to get the hash value of an image. This is a hexadecimal number.

1.3.1 Average Hashing (aHash)

The first hash method is just the average hash (aHash) of the image. This algorithm works in four steps.

1. Convert the image to grayscale.
2. Reduce the size of the image, to simplify the number of computations.
3. Average the resulting colors. For an 8x8 image, 64 will be averaged.
4. Compute the bits of the hash value by comparing if each color value is above or below the mean.
5. Construct the hash.

1.3.2 Difference Hash (dHash)

We can also compute the dHash of an image. It works analoguosly as the average hash, but now the difference between adjacent pixels is also considered in the computation.

1.3.3 Perceptive Hash (pHash)

Similar to the dhash, there exists the pHash. This method uses the discrete cosine transform (DCT) and compares the pixels based on the frequencies, given by the DCT, instead of the color values.

DCT is very similar to the Fourier analysis, it expresses a finite sequence of points/pixels/data in terms of a sum of cosine functions.

1.3.4 Use Of The Hash

One application of these methods is the recognition of images. Search engines like Google use this technique to search similar images as any image you provide to the search engine. I could use this as a way to store information about the image my application already scanned and generated music for.

For example, if you use a picture of a tree with my application, it will produce a certain kind of music. If you use another picture of another tree that looks quite similar to the first tree, you expect to get a similar result in the music. The hash values of the images will help with this.

1.4 Entropy

Another method we can use to get information from an image is via the entropy and the information content of the image. The entropy of an image is a measure of the amount of disorder in the image. This technique is also used in other disciplines, for example to study the structures of living organisms.

In the case of an image, we consider the gray levels of the individual pixels. If all the pixels of the image have the same gray level, the entropy is zero. This means there is not a lot of information to gain from the image. When all the pixels in the image are different, the entropy of this image is maximum.

To get the right information about the pixels, we use a histogram that shows the spread of the gray levels of every pixel.

The entropy of an image H is defined as:

$$H = -\sum_{k=0}^{M-1} p_k log_2(p_k) \tag{2}$$

where M is the number of gray levels and p_k is the probabilty associated with gray level k.

When the entropy of an image is maximal, the histogram of the gray levels is uniformly distributed. If $M = 2^n$, then p_k is constant and given by

$$p_k = \frac{1}{M} = 2^{-n} \tag{3}$$

and the maximum entropy is found from

$$H_{max} = -\sum_{k=0}^{M-1} (\frac{1}{M} log_2(\frac{1}{M}) = -log(2^{-n}) = n \tag{4}$$

What is the purpose of entropy? Entropy sets a lower bound on the average number of bits per pixel required to encode an image without distortion.