Regions characterisation

Image Feature Extraction Techniques

Piotr Fulmański

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE
University of Lodz

Form regions within an image

Image segmentation (pixel labelling)

In general, the process of partitioning the pixels in an image to form *regions* is known as *image segmentation* or *pixel labelling*. In our case, we illustrate the analysis of regions by considering techniques that group pixels according to their properties. Any grouping techniques essentially change the image resolution from high to very low.

I will present several techniques that grow regions by following an iterative process aggregating pixels according to measured local properties. This approach allows to create primitives that can be used to segment meaningful structures in images.

Watershed transform

Watershed transform

The watershed method derives from the morphological operations, and follows an analogy of flooding areas in hydrology. It partitions the image into multiple segments by growing regions from a set of initial seeds.

The main idea is to use labels to aggregate pixels in an iterative process. The pixels in each region have a different label. During each iteration, pixels that have not been aggregated into any region are labelled according to the labels of its neighbours. As such, connected components grow and edges that limit the regions can be delineated.

Watershed transform

The x axis defines pixels, and the y axis defines a ***distance property*** measured for all the pixels. We can visualise region growing process as a flooding of the topographic surface. In this case curve visualise an intersection of an image and represents only a one row of it. To represent the whole 2D image, the curve should be visualised as a surface.

Watershed transform

Watershed transform

According to the flooding analogy, the watershed process involves three main steps:

1. Compute a distance or property for each pixel. This represents how pixels are organised in regions and it defines a topographic surface

2. Find a local minima that represent initial regions.

3. Flood the topographic surface by labelling pixels and delineating edges.

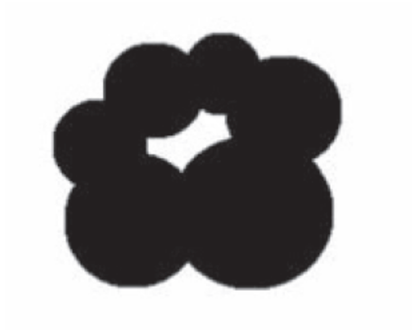
Although these three steps are fixed, there are many ways in which they can be implemented, thus there are many algorithms that segment an image following the watershed process.

Watershed transform

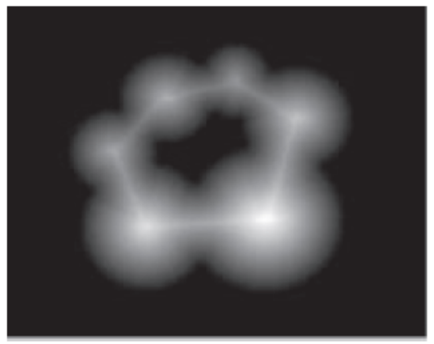
Generally, the **distance property is defined as the radial distance between pixel's positions and their closest edge**.

The main idea is to make the watersheds flood the image towards the edges. That is, pixels far away from edges are considered the centre of the regions (i.e. minima) and the flooding makes the regions grow until the region reaches the image edges (i.e. maxima). To compute the radial distances, it is necessary to identify edges and to find the closest edge to every pixel.

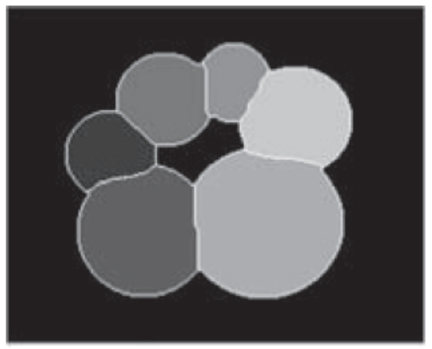
Watershed transform



Image



Distance



Watershed

Image source: [1]

Watershed transform

In the watershed process, initial regions are found by looking for extreme values. In general, these values are determined by finding minima in a local neighbourhood. This process has to deal with two practical issues.

- The minima can be extended over large regions forming a plateau.
- There may be small changes in the distance values due to noise.

Maximal Stable Extremal Region (MSER)

Maximal Stable Extremal Region (MSER)

The previous section showed how region growing can be used to find image regions according to the radial distance between pixel positions and edges; in the watershed method, regions are grown towards edges, thus the segmentation contains regions delimited by significant changes in image intensities. **Maximal Stable Extremal Region (MSER) also follows a region growing approach, but instead of using the distance to edges, it uses image intensities.**

The main motivation is to locate regions that are uniform in intensity, so they are maintained when images are captured with different illumination conditions or from different viewpoints.

Similar to watershed process, the image can be considered as a topographic surface, but in this case the vertical axis corresponds to pixel intensity.

Since, the surface is defined from pixel's intensities, then the flood level is actually equivalent to an image threshold, and so the MSER can be seen as a region growing process or as a multi-thresholding method.

In any case, the aim of the flooding (or thresholding) is to obtain a description of how the size of connected components changes as a function of the flooding level (or threshold).

Maximal Stable Extremal Region (MSER)

Stable regions

A general implementation requires four main steps:

1. Grow existing regions and store their size increment, this can be implemented as a flooding process similar to the one discussed in the previous part.

2. Merge touching regions and to update the change in sizes accordingly, which requires deletion of all the regions but one wherein all the pixels are aggregated.

3. Find new regions when the flooding reaches a minimum.

4. Detect stable regions by recording the number of times the regions have grown under the growing criteria.

Superpixels
Simple Linear Iterative Clustering (SLIC)

Simple Linear Iterative Clustering (SLIC)

Superpixel techniques segment an image into regions by considering similarity measures defined using perceptual features. That is, different from watersheds and MSER, **superpixel techniques create groups of pixels that look similar**. The motivation is to obtain regions that represent meaningful descriptions with far less data than is the case when using all the pixels in an image.

Simple Linear Iterative Clustering (SLIC)

In the SLIC technique, each pixel is characterised by a five-dimensional vector:

- Three components of the vector represent the colour using the CIELAB colour space as it gives a good similarity measure for colour perception.
- Two components of its position.

Similarity between two pixels $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ indexed by i and j is defined as:

$$D(i,j) = d_c(i,j) + m\frac{d_s(i,j)}{s}$$

where colour distance d_c and is a spatial difference d_s are defined as:

$$d_s(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

$$d_c(i,j) = \sqrt{(l_i - l_j)^2 + (a_i - a_j)^2 + (b_i - b_j)^2}.$$

The constant s is used to normalise the distance d_s. In SLIC the initial centre of a region is defined in a regular grid with sides s. Thus the normalisation $\frac{d_s}{s}$ defines the distance in terms of number of interval lengths. If we decrease the grid size, then the distance contributes more to the similarity increasing the compactness.

The constant m controls the compactness independent of the grid size. The greater the value of m the more special proximity is emphasised.

Simple Linear Iterative Clustering (SLIC)

The SLIC algorithm creates regions by three main steps:

1. Creates initial regions according to a parameter that defines the desired number of superpixels.

2. Performs region clustering to aggregate pixels to the regions according to the similarity criteria.

3. Reinforces connectivity (through iterative process).

Simple Linear Iterative Clustering (SLIC)

The process of creating initial regions

- Initially define superpixels by a regular grid of the size:

  $$s = \sqrt{\frac{N}{K}},$$

  where:

  - N is the number of pixels in the input image,
  - K is the number of superpixels.

  Initial location for each superpixel (its *reference point*) is set as a center pixel of corresponding grid cell.

  To improve this choice, you can shift center pixel according to some criteria. For example, you can shift it to the lowest gradient in a small pixel's neighbourhood.

- Compute the average colour of each region.

Simple Linear Iterative Clustering (SLIC)

The iterative region update process

Simply speaking, this is a well known k-means algorithm applying to the image data.

```
# For each pixel [x,y] find region [minX, minY] such that
# similarity D between this pixel and region's reference point
# is minimal.

newRegionColour[minX, minY] += colourOfPixel[x, y]
newRegionPos[minX, minY] += coordinatesOfPixel[x, y]
newRegionSize[minX, minY] += 1
regionsIDOfPixel[x, y] = [minX, minY]

# Update regions
regionPos[x,y] = newRegionPos[x,y] / newRegionSize[x,y]
regionColour[x,y] = newRegionColour[x,y] / newRegionSize[x,y]
```

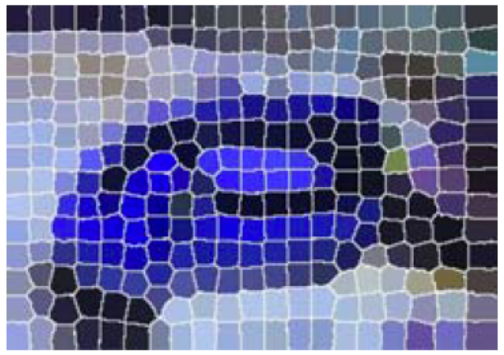


Image source: [1]

Textures and texture-based description

Statistical approach with co-occurrence matrix

Texture

There is no unique definition of texture and there are many ways to describe and extract it.

Co-occurrence matrix

The co-occurrence matrix contains elements that are counts of the number of pixel pairs for specific brightness levels, when separated by some distance and at some relative inclination. For brightness levels b_1 and b_2 the co-occurrence matrix C is

$$C(b_1, b_2) = \sum_{x=1}^{W} \sum_{y=1}^{H} ((I(x,y) == b_1) \text{ and } (I(x',y') == b_1))?1:0)$$

where W is an image width, H is an image height, and:

$$x' = x + r\cos(\theta),$$

$$y' = y + r\cos(\theta).$$

When above formula is applied to an image, you obtain a square, symmetric, matrix whose dimensions equal the number of grey levels in the picture.

Co-occurrence matrix

```
func textureCoocurrence(image, dist, dirs):
  #Get dimensions
  [rows,cols]=size(image)

  #Create Array[256][256]
  coocMatrix = [[0 for x in range(256)] for y in range(256)]

  for x = 1:cols:
    for y = 1:rows:
      for r = 1:dist:
        for theta = 0:2*pi/dirs:2*pi*(1-1/dirs)
          xc=round(x+r*cos(theta));
          yc=round(y+r*sin(theta));
          coocc[image(y,x)][image(yc,xc)] += 1

  return coocMatrix
```

Co-occurrence matrix

Co-occurence matric can be treated as a discriminator or fingerprint of an image.

We may need measurements that describe these matrices.

Local Binary Pattern (LBP)

Local Binary Pattern (LBP)

The goal is to create code P encodes the local intensity structure called *the local binary pattern*.

In the most basic form, LBP is derived by comparing the point $p = (x_p, y_p)$ with its neighbours $q_{p,x}$ from 3 x 3 region, to derive a code which is stored at the point p. For points p and $q_{p,x}$ of intensity I_p and $I_{p,x}$ respectively, the process depends on thresholding, which is given by the following function:

$$s(p, x) = \begin{cases} 1 & \text{if } I_{p,x} > I_p \\ 0 & \text{otherwise} \end{cases}.$$

The code is derived from binary weighting applied to result of thresholding for a point P with eight neighbours x_i for $i = 1,\ldots,8$:

$$\text{LBP}(p) = \sum_{i=1,\ldots,8} s(p, x_i) \cdot 2^{i-1}.$$

118	190	6
69	106	110
42	31	106

1	1	0
0		1
0	0	0

1	128	64
2		32
4	8	16

1	128	0
0	**161**	32
0	0	0

Note: The thresholding process, the unwrapping and the weighting can be achieved in different ways.

Local Binary Pattern (LBP)

The code P now encodes the local intensity structure.

It is complemented by two local measures: contrast and variance.

The **contrast** is computed from the difference between points encoded as a ‘1’ and those encoded as a ‘0’.

The **variance** is computed from the four neighbour pixels aiming to reflect pattern correlation as well as contrast.

Of these two complementary measures, **contrast was found to add most to discriminatory capability**.

Local Binary Pattern (LBP)

Invariance — translation

The approach is inherently **translation invariant** by its formulation: a texture which is shifted should achieve the same histogram of LBP codes.

Local Binary Pattern (LBP)

Invariancy — scale

The **scale invariance** requires consideration of points at a greater distance. If the space is sampled in a circular manner, and P points are derived at radius r, then the co-ordinate equations for $i \in [1,P]$ are:

$$x_R(i) = [x_0 + r(\cos(\frac{2\pi}{P})i), y_0 + r(\sin(\frac{2\pi}{P})i)].$$

Using Bresenham's algorithm you can precompute circle.

```
111
101
111
```

```
00100
01010
10001
01010
00100
```

```
0011100
0100010
1000001
1000001
1000001
0100010
0011100
```

Now the LBP formula takes the form:

$$\text{LBP}(p, r) = \sum_{i=1,\ldots,8} s(p, x_i^r) \cdot 2^{i-1}$$

Local Binary Pattern (LBP)

Invariance — rotation

The **rotation invariant** arrangement then shifts the derived code so as to achieve a minimum integer.

Local Binary Pattern (LBP)

Invariance — rotation

The LBP approach determines a histogram of the codes derived for an entire image and this histogram describes the texture.

Bibliography

Bibliography

1. Mark S. Nixon, Alberto S. Aguado, *Feature Extraction and Image Processing for Computer Vision*, Academic Press (Elsevier), 2020 (4th edition)