MSPL Week 1 何柏昇

Binary Fixed & Otsu

Fixed

Let RGB image mapping to gray-level, then segmentation of each pixel by threshold(T).From 8-bit to 1-bit.

$$\mathrm{g(x,y)} = \begin{cases} 0, & f(x,y) \geq T \\ 1(or\ 255), & x < T \end{cases}$$

Otsu

Otsu's algorithm is a simple and popular thresholding method for image segmentation, which falls into the clustering category.
The algorithm divides the image histogram into two classes, by using a threshold such as the in-class variability is very small. This way, each class will be as compact as possible.The algorithm tries to minimize the weighted within-class variance $\sigma_w^2(t)$.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

The total variance can be defined, as the sum of the within class $\sigma_w^2(t)$ and the between-class variance $\sigma_b^2(t)$. The value σ^2 is constant, as it does not depend on the threshold (the variance of an image is always a constant value), meaning that the algorithm must focus on minimizing $\sigma_w^2(t)$, or maximizing $\sigma_b^2(t)$.

$$\sigma^2 = \sigma_w^2(t) + \sigma_b^2(t), \qquad where\ \sigma_b^2(t) = q_1(t)q_2(t)[\mu_1(t) - \mu_2(t)]^2.$$

```
// Binary Fixed & Otsu
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv)
{
    //Read the image
    Mat image, fixedImage, otsuImage;
    image = imread("images/lena.png", 0);
    fixedImage = image.clone();
    otsuImage = image.clone();
    long double N = image.rows * image.cols;

    //Initialize variables
    long double threshold = -1, var_max = 0,
                sum = 0, sumB = 0, q1 = 0, q2 = 0,
                u1 = 0, u2 = 0, vb =0;
    long double max_intensity = 255;
    long double histogram[255];
    for (int i = 0; i < max_intensity; i++)
        histogram[i] = 0;

    // Compute the image histogram
    for (int y = 0; y < image.rows; y++)
        for (int x = 0; x < image.cols; x++)
            histogram[(int)image.at<uchar>(y,x)]++;

    // Auxiliary value for computing µ2
    for (int i = 0; i < max_intensity; i++)
        sum += i * histogram[i];

    // Update qi(t)
    for (int t = 0; t < max_intensity; t++)
    {
        q1 += histogram[t];
```

```
        if (q1 == 0)
            continue;
        q2 = N - q1;

        // Update μi(t)
        sumB += t * histogram[t];
        u1 = sumB / q1;
        u2 = (sum - sumB) / q2;

        // Update the between-class variance
        vb = q1 * q2 * (u1 - u2) * (u1 - u2);

        // Update the threshold
        if (vb > var_max)
        {
            threshold = t;
            var_max = vb;
        }
    }

    // Build the binary-fit image
    if ( argc != 2 )
    {
        printf("usage: ./binary <binary_fit>\n");
        return -1;
    }
    int fit;
    fit =  atoi(argv[1]);

    for (int y = 0; y < fixedImage.rows; y++)
    {
        for (int x = 0; x < fixedImage.cols; x++)
        {
            if (fixedImage.at<uchar>(y,x) > fit)
                fixedImage.at<uchar>(y,x) = 255;
            else
                fixedImage.at<uchar>(y,x) = 0;
        }
```

```
    }

    // Build the binary-otsu image
    for (int y = 0; y < otsuImage.rows; y++)
    {
        for (int x = 0; x < otsuImage.cols; x++)
        {
            if (otsuImage.at<uchar>(y,x) > threshold)
                otsuImage.at<uchar>(y,x) = 255;
            else
                otsuImage.at<uchar>(y,x) = 0;
        }
    }

    // Show images
    namedWindow("Image", WINDOW_AUTOSIZE );
    imshow("Image", image);

    namedWindow("fixedImage", WINDOW_AUTOSIZE );
    imshow("fixedImage", fixedImage);

    namedWindow("Otsu", WINDOW_AUTOSIZE );
    imshow("Otsu", otsuImage);
    waitKey(0);

    // Save images
    imwrite("images/fixed.png", fixedImage);
    imwrite("images/otsu.png", otsuImage);

    return 0;
}
```

Original

Otsu

fixed_80

fixed_180

Histogram Equalization

Following is the algorithm to do histogram equalization in C language.

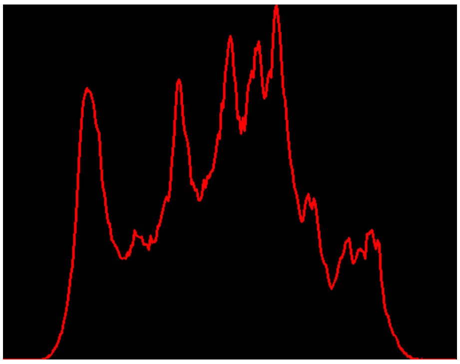
1. Convert the input image into a grayscale image
2. Find frequency of occurrence for each pixel value i.e. histogram of an image (values lie in the range [0, 255] for any grayscale image)
3. Calculate Cumulative frequency of all pixel values
4. Divide the cumulative frequencies by total number of pixels and multiply them by maximum gray count (pixel value) in the image

```cpp
// Histogram Equalization
#include <stdio.h>
#include <cmath>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv)
{
    // Read the image
    Mat image, newImage;
    image = imread("images/lena.png", 0);
    newImage = image.clone();
    // Initialize variables
    int max_intensity = 255;
    long double histogram[255];
    long double new_histogram[255];
    for (int i = 0; i < max_intensity; i++)
    {
        histogram[i] = 0;
        new_histogram[i] = 0;
    }

    // Compute the image histogram
    for (int y = 0; y < image.rows; y++)
        for (int x = 0; x < image.cols; x++)
            histogram[(int)image.at<uchar>(y,x)]++;

    // calculating total number of pixels
    long double total = image.rows * image.cols;
    long double curr =0;

    for (int i = 0; i < max_intensity; i++)
    {
        curr += histogram[i];
        new_histogram[i]=round((((float)curr) * 255) / total);
    }
```

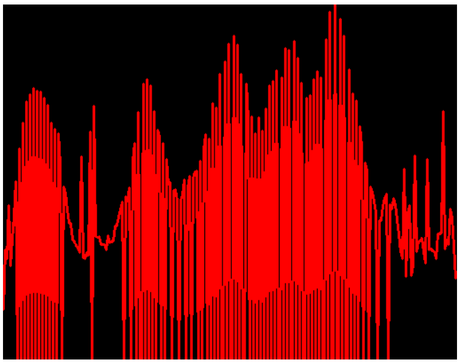
```
    // Performing histogram equalization by mapping new gray levels
    // Compute the image histogram
    for (int y = 0; y < newImage.rows; y++)
        for (int x = 0; x < newImage.cols; x++)
            newImage.at<uchar>(y,x)= new_histogram[newImage.at<uchar>(y,x)];

    // Show images
    namedWindow("Image", WINDOW_AUTOSIZE );
    imshow("Image", image);

    namedWindow("Histogram Equalization", WINDOW_AUTOSIZE );
    imshow("Histogram Equalization", newImage);
    waitKey(0);

    // Save image
    imwrite("images/hq.png", newImage);

    return 0;
}
```

Original

HQ



Original



HQ

Morphology

Erosion

Properties [edit]

- The erosion is translation invariant.
- It is increasing, that is, if $A \subseteq C$, then $A \ominus B \subseteq C \ominus B$.
- If the origin of *E* belongs to the structuring element *B*, then the erosion is *anti-extensive*, i.e., $A \ominus B \subseteq A$.
- The erosion satisfies $(A \ominus B) \ominus C = A \ominus (B \oplus C)$, where $\oplus$ denotes the morphological dilation.
- The erosion is distributive over set intersection

Dilation

Properties of binary dilation [edit]

Here are some properties of the binary dilation operator

- It is translation invariant.
- It is increasing, that is, if $A \subseteq C$, then $A \oplus B \subseteq C \oplus B$.
- It is commutative.
- If the origin of *E* belongs to the structuring element *B*, then it is extensive, i.e., $A \subseteq A \oplus B$.
- It is associative, i.e., $(A \oplus B) \oplus C = A \oplus (B \oplus C)$.
- It is distributive over set union

Opening

$$dst = open(src, element) = dilate(erode(src, element))$$

Closing

$$dst = close(src, element) = erode(dilate(src, element))$$

```
// Morphology : Erosion, Dilation, Opening, Closing
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char** argv)
{
    // Read the image
    Mat image;
    image = imread("images/lena.png", IMREAD_GRAYSCALE);

        // Build the binary-fit image
    if ( argc != 2 || (atoi(argv[1])%2) == 0 || atoi(argv[1]) > image.rows
|| atoi(argv[1]) > image.cols)
    {
        printf("usage: ./morphology <kernel size(3 ,5 ,7, ..., 2N+1)>\n");
        return -1;
    }
    int kernel;
    kernel =  atoi(argv[1])/2;

    // Dilated and Eroded
    Mat dilatedImage, erodedImage;
    dilatedImage = image.clone();
    erodedImage = image.clone();
    // find the maximum and minimum pixel intensity of image
    for (int i = 0; i < image.rows; i++)
    {
        for (int j = 0; j < image.cols; j++)
        {
            uchar maxV = 0;
            uchar minV = 255;
            for (int y = i-kernel; y <= i+kernel; y++)
            {
                for (int x = j-kernel; x <= j+kernel; x++)
                {
```

```
                if (x < 0 || x >= image.cols || y < 0 || y >= image.rows)
                    continue;
                // Dilated
                maxV = max<uchar>(maxV, image.at<uchar>(y,x));
                // Eroded
                minV = min<uchar>(minV, image.at<uchar>(y,x));
            }
        }
        dilatedImage.at<uchar>(i,j) = maxV;
        erodedImage.at<uchar>(i,j) = minV;
    }
}

// Opening and Closing
Mat openImage, closeImage;
openImage = image.clone();
closeImage = image.clone();
// find the maximum and minimum pixel intensity of image
for (int i = 0; i < image.rows; i++)
{
    for (int j = 0; j < image.cols; j++)
    {
        uchar maxV = 0;
        uchar minV = 255;

        for (int y = i-kernel; y <= i+kernel; y++)
        {
            for (int x = j-kernel; x <= j+kernel; x++)
            {
                if (x < 0 || x >= image.cols || y < 0 || y >= image.rows)
                    continue;
                // Opening = Dilated(Eroded())
                maxV = max<uchar>(maxV, erodedImage.at<uchar>(y,x));
                // Close = Eroded(Dilated())
                minV = min<uchar>(minV, dilatedImage.at<uchar>(y,x));
            }
        }
        openImage.at<uchar>(i,j) = maxV;
```

```
            closeImage.at<uchar>(i,j) = minV;
        }
    }

    // Show image
    namedWindow("Image", WINDOW_AUTOSIZE );
    imshow("Image", image);

    namedWindow("Dilated", WINDOW_AUTOSIZE );
    imshow("Dilated", dilatedImage);

    namedWindow("Eroded", WINDOW_AUTOSIZE );
    imshow("Eroded", erodedImage);

    namedWindow("Opening", WINDOW_AUTOSIZE );
    imshow("Opening", openImage);

    namedWindow("Closing", WINDOW_AUTOSIZE );
    imshow("Closing", closeImage);
    waitKey(0);

    // Save image
    imwrite("images/dilatedImage.png", dilatedImage);
    imwrite("images/erodedImage.png", erodedImage);
    imwrite("images/openImage.png", openImage);
    imwrite("images/closeImage.png", closeImage);

    return 0;

}
```

Erosion

Dilation

Opening

Closing

Reference

1. https://www.ipol.im/pub/art/2016/158/article_lr.pdf
2. https://hbyacademic.medium.com/otsu-thresholding-4337710dc519
3. https://learnopencv.com/otsu-thresholding-with-opencv
4. https://jason-chen-1992.weebly.com/home/-histogram-equalization
5. https://www.geeksforgeeks.org/histogram-equalisation-in-c-image-processing/
6. https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
7. https://docs.opencv.org/3.4/d3/dbe/tutorial_opening_closing_hats.html
8. https://blog.51cto.com/u_15414551/4399694
9. http://www.scu.edu.tw/math/Chieping/n.5-dimension/image_processing.html
10. https://homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf
11. https://docs.opencv.org/3.4.0/db/df5/tutorial_linux_gcc_cmake.html

Github

MSPL/ week1