Introduction to OpenCV

——The OpenCV exercises in EC601 class

Simin Zhai
siminz@bu.edu

Exercise 1

1. How does a program read the cvMat object, in particular, what is the order of the pixel structure?

cvMat is the most important function in OpenCV. The program reads the cvMat object by using CvMat* cvCreateMat (int rows, int cols, int type). The type could be any pre-define type used the contruction:CV_<bit_depth> (S|U|F)C<number_of_channels>. Then, the elements of matrix could be 32 bits float or CV_8UC3, or others. One element in cvMat may not be a single number. We could use simple input to represent multiple values, then we could describle multicolor channels in a RGB image.

This kind of matrix is made of width, height, type, step and a pointer which points to the data.

Mat is used in C++. We could use:

```
Mat src;
src = imread(argv[1], CV_LOAD_IMAGE_COLOR);
```

This kind of instructions to access the image.

The structure is as follows:

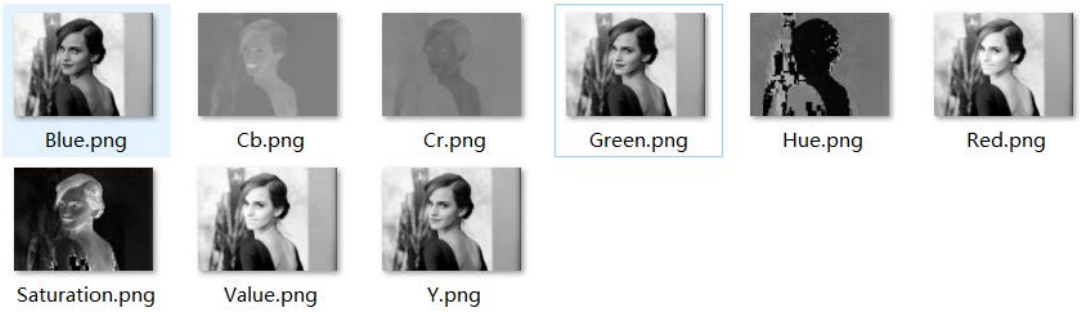
```
typedef struct CvMat

{
    int type;
    int step;
    int* refcount;/* for internal use only */
    int hdr_refcount;
    union
    {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data;
    union
    {
        int rows;
        int height;
    };
    union
    {
        int cols;
        int width;
```

};
} CvMat;

 # Exercise 2

1. ColorImage.cpp is a program that takes a look at colorspace conversions in OpenCV. Run the code in ColorImage.cpp and comment on the outputs. Implement the same thing in Python and save each image.



2. Print out the values of the pixel at (20,25) in the RGB, YCbCr and HSV versions of the image. What are the ranges of pixel values in each channel of each of the above mentioned colorspaces?

a. the values of the pixel at (20,25)

('blue pixel value', 246)
('Green pixel value', 236)
('red pixel value', 236)
('H pixel value', 120)
('S pixel value', 10)
('V pixel value', 246)
('Y pixel value', 237)
('Cb pixel value', 127)
('Cr pixel value', 133)

b. The range of R,G,B, YCbCr are all 0-255.

For HSV, Hue range is 0-179, Saturation range is 0-255 and Value range is 0-255.

 # Exercise3

1.Look at the code in Noise.cpp and implement the code in Python. Also, print the results for different noise values in the Gaussian case, mean = 0, 5, 10, 20 and sigma = 0, 20, 50, 100 and for the salt-and-pepper case, pa = 0.01, 0.03, 0.05, 0.4 and pb = 0.01, 0.03, 0.05, 0.4.

2. Change the kernel sizes for all the filters with all different values for noises and print the results for 3x3, 5x5 and 7x7 kernels. Comment on the results. Which filter seems to work "better" for images with salt-and-pepper noise and gaussian noise?

Gaussian mean = 0 sigma 20 kernel = 3*3 S&P: pa = 0.01 pb = 0.01


Boxfilter.png


Boxfilter2.png


Gaussian.png


Gaussian_noise.png


Gaussian2.png


Median.png


Median2.png


S&P.png

Gaussian mean = 0 sigma 20 kernel = 5*5 S&P: pa = 0.01 pb = 0.01


Boxfilter.png


Boxfilter2.png


Gaussian.png


Gaussian_noise.png


Gaussian2.png


Median.png


Median2.png


S&P.png

Gaussian mean = 0 sigma 20 kernel = 7*7 S&P: pa = 0.01 pb = 0.01


Boxfilter.png


Boxfilter2.png


Gaussian.png


Gaussian_noise.png

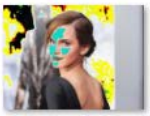

Gaussian2.png


Median.png

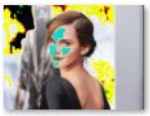

Median2.png

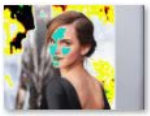

S&P.png

Gaussian mean = 20 sigma = 0 kernel = 3*3 S&P: pa = 0.01 pb = 0.03

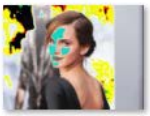

Boxfilter.png


Boxfilter2.png


Gaussian.png


Gaussian_noise.png

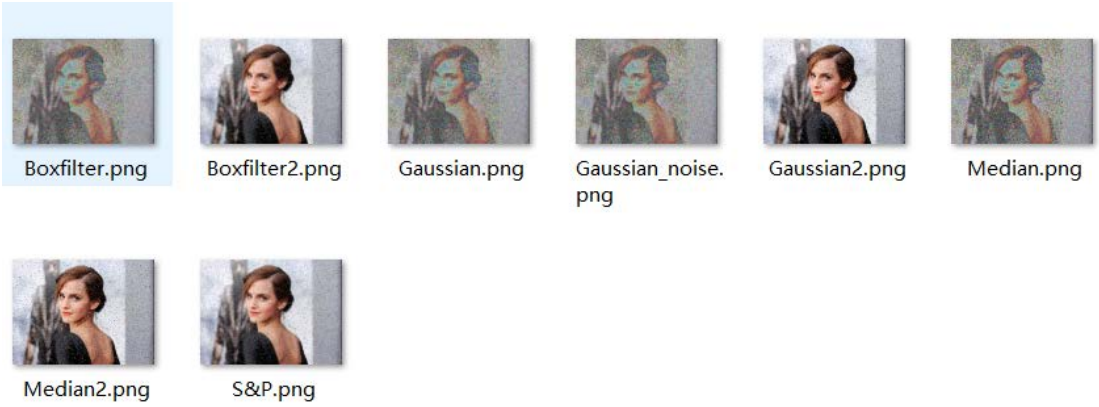

Gaussian2.png


Median.png

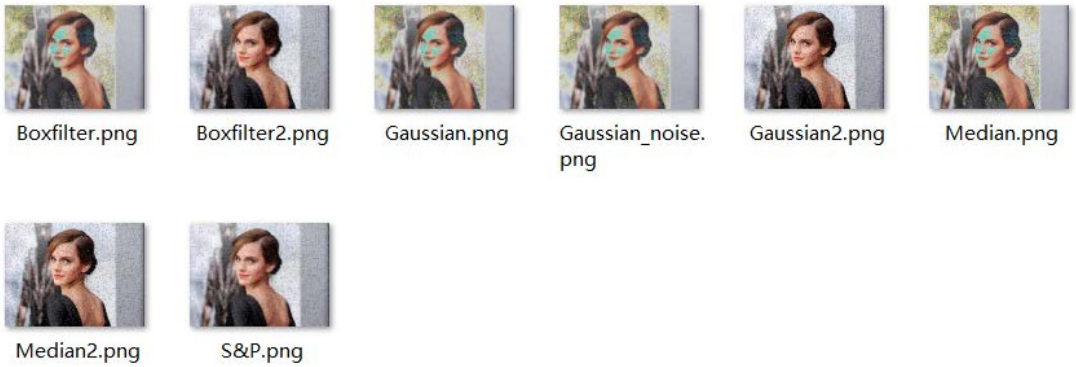

Median2.png


S&P.png

Gaussian mean = 5 sigma= 50 kernel = 3*3 S&P: pa = 0.03 pb = 0.03



Gaussian mean = 10 sigma = 20　kernel = 3*3 S&P: pa = 0.05 pb = 0.03



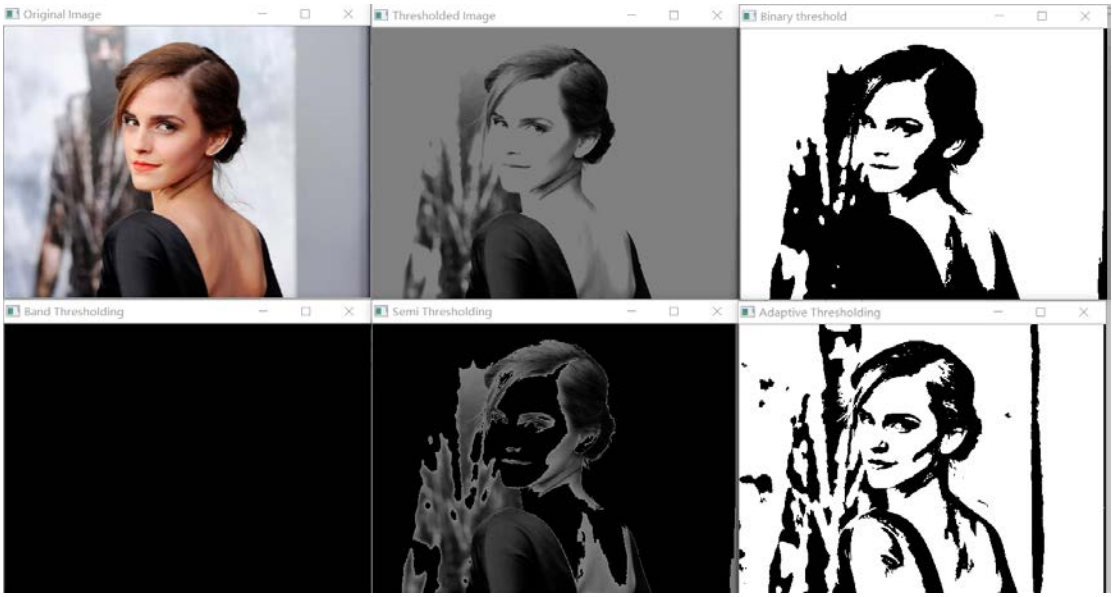
Gaussian mean = 10 sigma = 100　kernel = 3*3 S&P: pa = 0.4 pb = 0.05

Gaussian mean = 20 sigma = 100　kernel = 3*3 S&P: pa = 0.4 pb = 0.4

According to the result, median filter works better for salt and pepper images and gaussian filter works better for gaussian noise images. If the size of kernel becomes bigger, the results of the image becomes more blurred.

#Exercise4

1. **Look at Threshold.cpp and implement the code in Python, and observe the results for different threshold values. Comment on the results.**



2. **What are the disadvantages of binary threshold?**

Limited application: as the representation is only a silhouette, application is restricted to tasks where internal detail is not required as a distinguishing characteristic.

Does not extend to 3D: the 3D nature of objects can rarely be represented by silhouettes. (The 3D equivalent of binary processing uses voxels, spatial occupancy of small cubes in 3D space).

Specialised lighting is required for silhouettes: it is difficult to obtain reliable binary images without restricting the environment. The simplest example is an overhead projector or light box

The problems which occur in blindly applying thresholding are due to the nature of real images and the fact that the assumptions which underlie the method, i.e., the image is representable using only two grey levels, are not satisfied.

A common problem is the effect of gradual shading across the surface of a large image. The image is still segmentable provided nonrandom contrast which delineates objects is locally preserved.

Another problem may be that the object area is small compared with the background area and therefore the size of the mode that it contributes to the histogram is no more significant than that of random picture noise. Both of the above problems can be addressed by summing the histogram over an area which is more appropriate, i.e., in these cases a smaller area.

3. **When is Adaptive Threshold useful?**

adaptive thresholding is used to separate desirable foreground image objects from the background based on the difference in pixel intensities of each region. If conventional thresholding method is not good in all the conditions where image has different lighting

conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculates the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.