Cuestionario de teoría
1

Alejandro García Montoro
agarciamontoro@correo.ugr.es

28 de octubre de 2015

1. Análisis de la implementación

1.1. Función de convolución

La implementación de la función de convolución 2D se construye sobre funciones más simples; a saber:

- Cálculo de una máscara gaussiana 1D.
- Convolución 1D con una máscara general.

Sobre la convolución 2D se construye, además, la función lowPassFilter(), que implementa un filtro Gaussiano.

1.1.1. Cálculo de una máscara gaussiana 1D

La máscara gaussiana se construye con la función getGaussMask(), cuyo código se puede ver más abajo. Esta función devuelve un objeto Mat que representa un vector uni-dimensional generado a partir de la discretización de la función gaussiana en el intervalo $[-3\sigma, 3\sigma]$.

El tamaño de la máscara tiene que ser un número natural impar. Así, primero se redondea el resultado de triplicar el σ, luego se duplica lo anterior y por último se le añade uno.

Además, las máscaras de convolución tienen una restricción: la suma de sus valores tiene que ser igual a uno. Por tanto, primero se genera la máscara discretizando la función sin atender a esta restricción y luego se *normaliza*; esto es, se divide cada valor de la máscara por la suma de todos ellos.

```
1       /**
2        * Builds a gaussian mask given the parameter sigma. The
             gaussian function is
3        * sampled in the interval [-3*sigma, 3*sigma].
4        */
5       Mat Image::getGaussMask(double sigma){
```

```
6           // Gaussian function needs to be sampled between -3*sigma
                and +3*sigma
7           // and the mask has to have an odd dimension.
8           int mask_size = 2*round(3*sigma) + 1;
9
10          Mat gauss_mask = Mat(1,mask_size,CV_32FC1);
11
12          // It is necessary to normalize the mask, so the sum of
                its elements
13          // needs to be saved.
14          float values_sum = 0;
15
16          // Fills the mask with a sampled gaussian function and
                saves the sum of all elements
17          for (int i = 0; i < mask_size; i++) {
18              gauss_mask.at<float>(0,i) =
                    gaussianFunction(i-mask_size/2, sigma);
19              values_sum += gauss_mask.at<float>(0,i);
20          }
21
22          // Normalizes the gauss mask.
23          gauss_mask = gauss_mask / values_sum;
24
25          return gauss_mask;
26      }
```

Los valores de la función gaussiana se consiguen con la siguiente implementación, que da el valor de la función en un punto x con parámetro σ.

```
1       /**
2        * Samples the 1D gaussian function at point x with parameter
             sigma
3        */
4       double Image::gaussianFunction(double x, double sigma){
5           return exp(-0.5*(x*x)/(sigma*sigma));
6       }
```

1.1.2. Convolución 1D con una máscara general

El siguiente paso es la implementación de una convolución uni-dimensional con una máscara general. El código es como sigue:

```
1       /**
2        * Returns the result of convolving the uni-dimensional
             signal_vec with the
3        * mask, applying one of two types of borders: REFLECT or
             ZEROS.
```

```
4         */
5        Mat Image::convolution1D(const Mat& signal_vec, const Mat&
             mask, enum border_id border_type){
6            assert(signal_vec.rows == 1 && mask.rows == 1 && mask.cols
                 < signal_vec.cols);
7
8            int num_channels = signal_vec.channels();
9
10           // Initialization of source vector with additional borders.
11           int border_size = mask.cols/2; // Number of pixels added
                 to each side
12           Mat bordered;
13           copyMakeBorder(signal_vec,bordered,0,0,border_size,border_size,border_type,0.0);
14
15           // Splitting of the bordered vector for making a
                 per-channel processing
16           vector<Mat> bordered_channels(num_channels);
17           split(bordered, bordered_channels);
18
19           // Declaration of the result vector -with same size and
                 type as the
20           // original signal vector- and its splitted channels.
21           Mat result = Mat(signal_vec.size(), signal_vec.type());
22           vector<Mat> result_channels(num_channels);
23           split(result, result_channels);
24
25           // The mask and the source/result channels need to have
                 the same type.
26           // They are all converted to CV_32FC1 in order not to lose
                 precision.
27           Mat converted_mask;
28           mask.convertTo(converted_mask,CV_32FC1);
29
30           // Per-channel processing: we need the source channels,
                 the masked channels;
31           // i.e., the source channel focused in a ROI of the same
                 size as the mask
32           // and the result channels.
33           Mat source_channel, masked_channel, result_channel;
34
35           for (int i = 0; i < num_channels; i++) {
36               // Channel type conversion
37               bordered_channels[i].convertTo(source_channel,
                     CV_32FC1);
38               result_channels[i].convertTo(result_channel, CV_32FC1);
39
40               // Actual processing
41               for (int j = 0; j < result.cols; j++) {
42                   // We focus on the zone centered at j with mask
```

```
                    width
43                  masked_channel =
                        source_channel(Rect(j,0,mask.cols,1));
44
45                  // Scalar product between the ROI'd source and the
                        mask
46                  result_channel.at<float>(0,j) =
                        masked_channel.dot(converted_mask);
47              }
48
49              // Backwards conversion: the result should have the
                    same type as the input image
50              result_channel.convertTo(result_channels[i],result_channels[i].type());
51          }
52
53          // Merging again the processed channels
54          merge(result_channels, result);
55
56          return result;
57      }
```

2. Análisis de resultados