**[Session 2] OpenCV Supplement**

**Interesting functions**

We will briefly introduce some of interesting image processing functions. Not only the function use, we will briefly look at the background.

**(Binarization)**

As the name suggests, it bifurcates a given color. Note that each channel can have the integer value 0 to 255. We can specify a certain "threshold" and apply this:

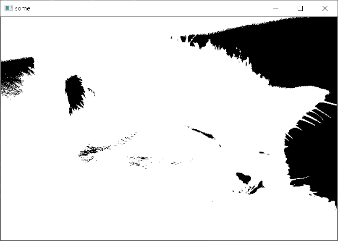
The function can be used like this:

* **cv2.threshold(image, threshold, max, type, dst)**
* **image**: image to apply binarization.
* **threshold**: the threshold value (T)
* **max**: the max value
* **type**: Binarization type flag
  + **cv2.THRESH\_BINARY**: Usual binarization, maxval if x > T, and 0 otherwise
  + **cv2.THRESH\_INVERSE**: 0 if x > T, and maxval otherwise
  + **cv2.THRESH\_TRUNC**: T if x > T, x otherwise
  + **cv2.THRESH\_TOZERO**: x if x > T, 0 otherwise
  + **cv2.THRESH\_TOZERO\_INV**: 0 if x > T, x otherwise
* **dst:** the destination

And it returns 2-tuple (thresh, image)

* thresh: the threshold that was used
* image: modified image

This is useful because it can be used for distinguishing something useful / or not clearly and simplifying the given image. Note that the threshold should be set carefully:



**(Resize)**

Resizing seems straightforward, but it needs **"interpolation"** mechanism.



When you zoom the above image, then you can see an aliasing effect, so the edge seems is too rough. The remedy for this is called anti-aliasing. We can interpolate the aliasing, by inserting some color between two space (in this example, between white and black). Suppose that we want to make some image larger.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

In the above example, the shaded area may need some interpolation. Thus, we specify the interpolation method when we call resize function.

* **cv2.resize(src, dsize, dst, fx, fy, interpolation)**
* dst, fx, fy, interpolation is keyword-argument (with a default value(None))
* For interpolation, We usually use cv2.INTER\_AREA when zoom out scaling, and cv2.INTER\_CUBIC / cv2.INTER\_LINEAR when zoom in scaling

**(Affine Transformation)**

**Affine transformation** is a kind of transformation mechanism. This is useful because some "nice" property is conserved even after the transformation is applied. We can do (parallel) transition, scaling, rotation, shearing, etc. Because the image is a kind of matrix, you can apply affine transformation, if you are aware of some backgrounds of linear algebra.

* **cv2.getAffineTransform(src, M)**
* **src**: the source image
* **M**:affine transformation matrix

To get some affine transformation matrix, we can use some backgrounds for matrix and linear algebra. Also, we can get an affine matrix for 2D rotation by using the function.

* **cv2.getRotationMatrix2D(center, angle, scale)**

Use the return value as a transformation matrix.

**Encode / Decode, and Codec**

**Suggestion**

1. It is good to know how to deal with matrix or array. So learning NumPy will be helpful for you.