HW 7 Thinning Operator

Thinning Operator

Source code

Please refer to the file "`hw7_ver2.py`" within the same folder as this report document.

Step #1: Binarizing the image

```
"""
    This function sets pixels to 0 if they're less than $threshold$ in $src$.
    @param src should be an image
    @param threshold should be an integer (default 128 if not given)
    @return a binarized image instance
"""
def binarize(src, threshold=128):
```

— binarize() is implemented for banalizing any given image with a given threshold (default 128).

Step #2: Down-sampling the image

```
"""
    This function does the downsampling.
    @param src should be an image
    @param block a 2-tuple representing a block, and we'll take upmost-left point of this block
    @return the new image downsampled
"""
def downSample(src, block=(8, 8)):
```

— downSample() is implemented for down-sampling any given image by taking the upmost-left pixel to represent a given block size.

Step #3: Mark interior/border pixels

```
"""
    @param src should be an image
    @param connectedType either 4 or 8.
    @return a new image with border pixels in white, interior pixels in grey(128), and background pixels in black.
"""
def getBorderPixels(src, connectedType):
```

— getBorderPixels() is responsible for detecting interior pixels and border pixels. It will return a new image with interior pixels in grey, border pixels in white, and background pixels in black.

Step #4: Pair relationship operator

```
"""
    @param src should be an image
    @param connectedType either 4 or 8.
    @param borderImage a image returned by getBorderPixels() function.
    @return a new image with marked pixels in white, foregroundpixels in grey(128), and background pixels in black.
"""
def doPairRelationship(src, connectedType, borderImage):
```

— The function doPairRelationship() helps mark those border pixels which neighbors at least one interior pixels, and therefore it takes an image returned by the getBorderPixels() function as one of the parameters.

Step #5: Connected-shrink operator

```
# the h function for 4-connected connected shrink
def h4(b, c, d, e):
    if b == c and (d != b or e != b):
        return 1
    return 0

# the h function for 8-connected connected shrink
def h8(b, c, d, e):
    if b != c and (d == b or e == b):
        return 1
    return 0

def f(a1, a2, a3, a4, x):
    s = a1 + a2 + a3 + a4
    return 'g' if s == 1 else x
```

I didn't implement a function for doing connected-shrink operator.
However, the two primitive functions of this operator are implemented: h() and f().

Result