Zhang Suen thinning / skeletonization Algorithm.

Image Processing project

184650
Akram Abuhajar.

Definition and uses.

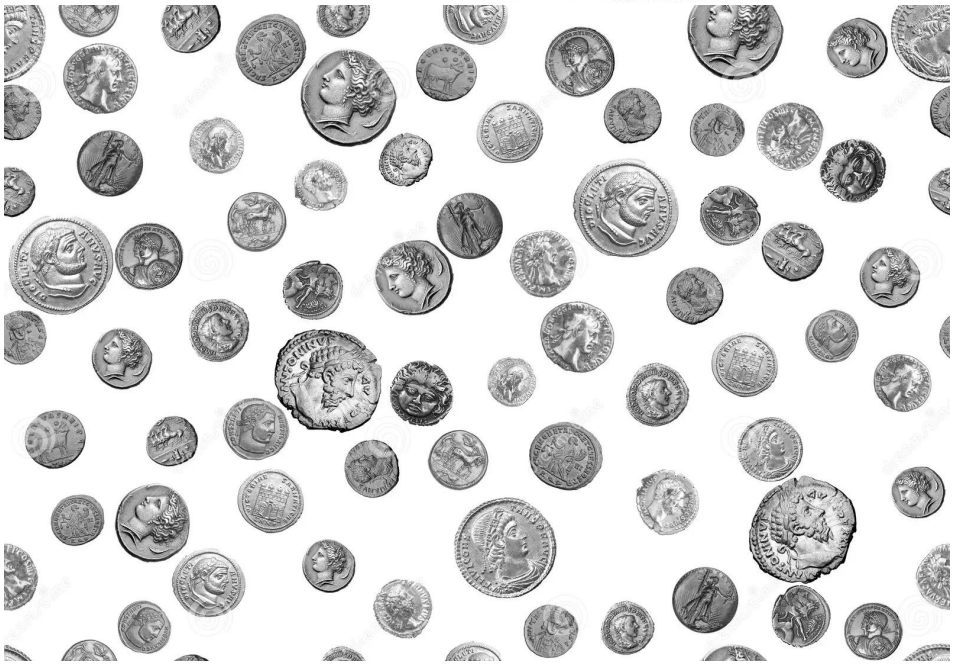
so what is the zhang suan algorithm all about, it is a thinning AKA skeletonization algorithm that takes affect on a binary picture and draws a 1 pixel wide skeleton of the same image, and it is smart enough to maintain the shape of the object in the image, but skeletonized.

The main reason for using such algorithm is shape detection and object recognition, due to the way it works and the output this algorithm produces, basically thinning is simplifying, and simplifying the image is reducing the amount of details to the comparable minimum (its most abstract version of the image), and with such an output, it is so much easier to study image patterns, how specfic object will look like skeletonized, away from color and size and such other specifications of an image.

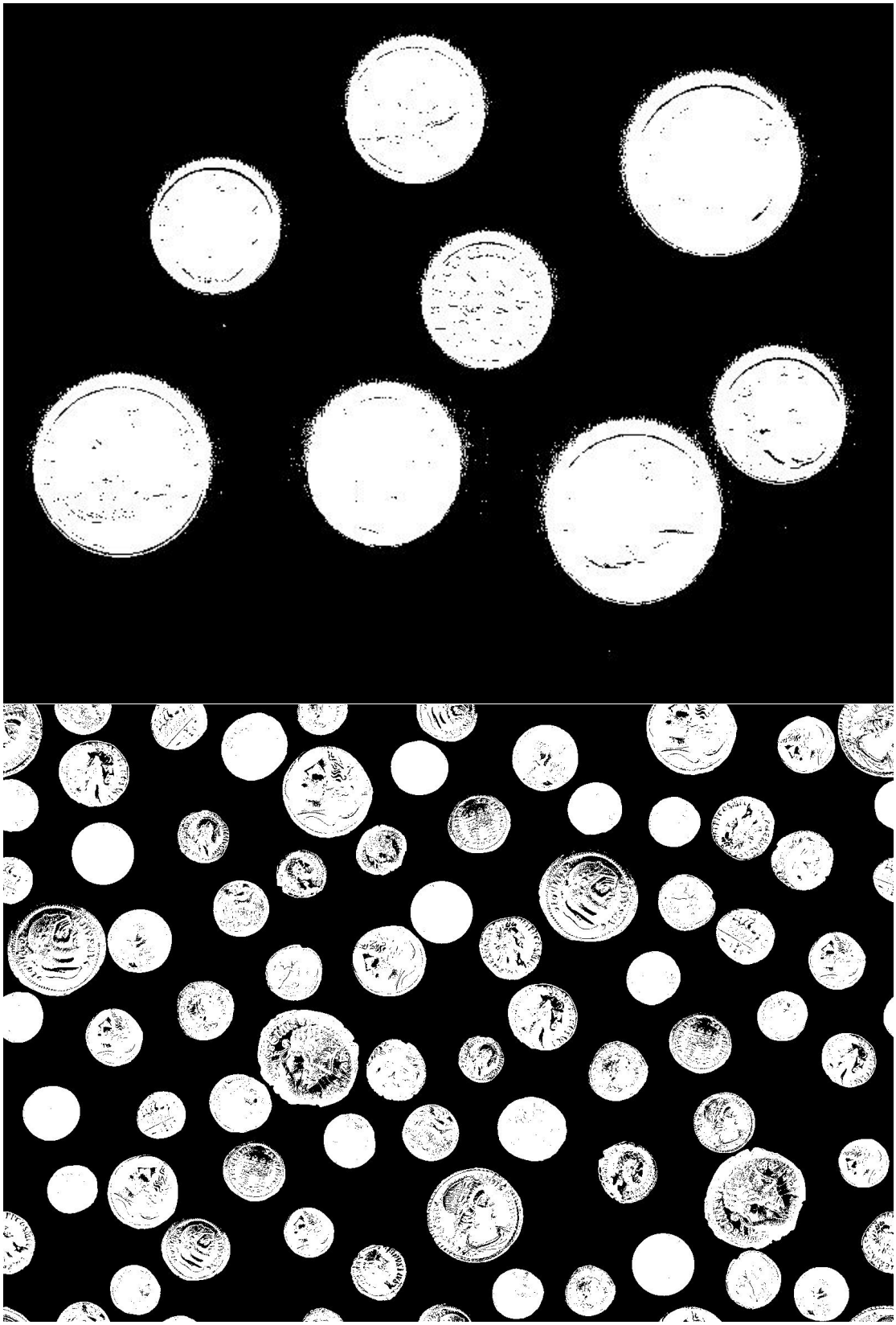
Getting a skeletonized version of the picture will help a lot for comparison that will help with all sorts of fields that require differentiating between a group of images.

Example and code explanation.

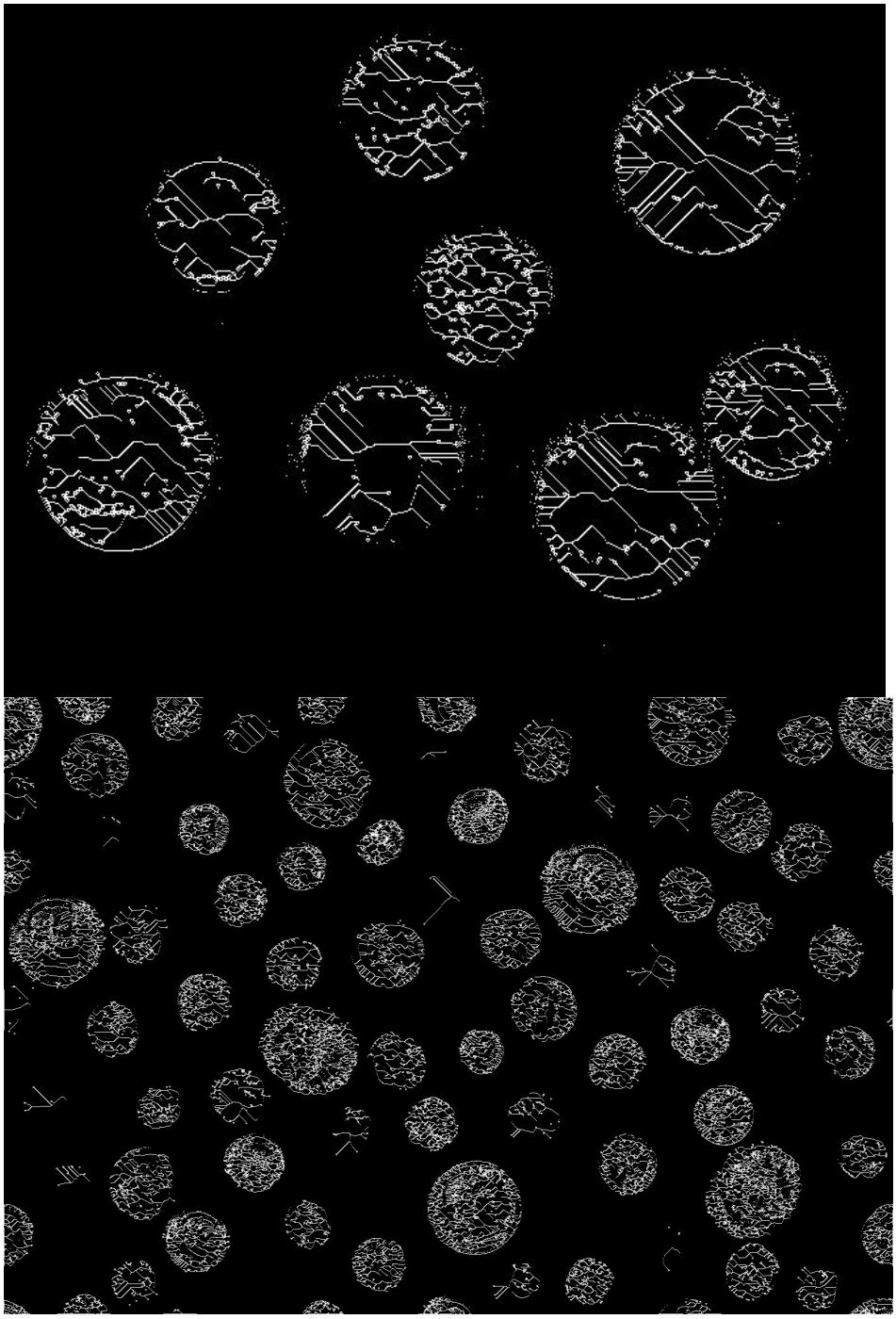
Here is some colored images that I used in testing the algorithm output.



In these images you can see two different pictures of coins, with different colors, shapes and drawings.



And over here the binary image of each one of the input pictures.



These are the output pictures of the algorithm, you can see after thinning, you can actually recongnize the pattern such shapes make, and how easily it will be to know that both a coins without the requirement of making patterns for recognition from the original pictures.

explanation.

The way it works, is it reads a 3x3 kernal, where the pixel that the kernal is used on will be placed in the middle of the kernal.

Here is a good way to explain it :

Kernal and its calculations.

(X-1, Y-1)	(X, Y - 1)	(X + 1, Y - 1)
(X - 1, Y)	(X, Y)	(X + 1, Y)
(X - 1, Y + 1)	(X , Y + 1)	(X + 1, Y + 1)

So basically each pixel has (x, y) coordinates, and since we are dealing with a binary image, values of each pixel is a 1, or a 0, and using this kernal, you make 9 intergers, and save each index of the kernel in a separate variable for further calculations.
Create a vector that will save coordinates of the pixels that will be deleted later on.
(quick info, I flipped between white and black for me to easier understand it, but it worked the same, so white is 1, and black is 0)
You go through conditions.
1- Check if the pixel has 8 neighbors, and by this, you add all the pixels of the frame to the deleted pixel immediately.
2- Check if the pixel is whitewhich is == 1, and the number of black pixels surrounding it is >= 2 and <= 6
3- Check if in the kernel there is 2 neighbors, that one of them is 1, and the other is 0.
4- Check if there is 2 neighbors that are 1 and next to each other in a clock wise direction.

If these conditions are met, then the pixel gets saved in the deletion vector for later

Now after iterating through the whole image, another check happens on it with slightly different conditions, and the conditions are :

1- Check if the center is whitewhich is equal to 1
2- Check if there is 2 <= black pixels <= 6.
3- Check if there is 2 neighbors, that one of them is 1 and the other is 0.
4- Check if there is 2 neighbors that are 1 and next to each other in an anti-clock wise direction

If these conditions are met, then the pixel gets saved in the deletion vector for later.

Using this method you keep on repeating until all pixels of the image does achieve the conditons.