obligatorisk innlevering 3

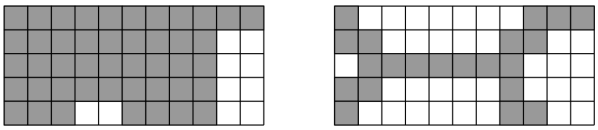
Sivert M. Skarning

April 2019

1 Oppgave 1

1.1 Algoritme

I denne oppgaven skal det implementeres morfologisk tynning på figur 1.



Figur 1: Morfologisk Tynning

Fremgangsmåte på hvordan denne skulle løses var gitt i Digital Image Processing s.661 [1]. En løsning på denne oppgaven kan bli beskrevet slik.

1. Lese inn binært bilde fra en tekstfil
2. Zero padder bilde for å støtte størrelsen på structuring element (heretter SE)
3. kjører SE gjennom bilde
4. Roterer SE 45 grader og gjør det samme helt til SE har rotert 360 grader
5. Sjekker om bilde har endret seg, hvis ikke blir bildet skrevet ut. Hvis det har endret seg, blir algoritmen kjørt om igjen

Denne løsningen er implementert i kode som du finner vedlagt i innleveringsmappen.

1.2 Implementasjon

For å konstruere et SE bruker jeg verdiene 1 for True, 0 for False og -1 for DDont Care"verdier. Min første implementasjon var med å bruke Boolean verdier, dette viser seg å ikke fungere fordi det blir vanskelig å implementere Dont care verdier".

Roteringen av SE er gjort av egen funksjon, som kun fungerer på 3x3 SE. Prøvde flere rotasjonsfunksjoner for dette men ingen av de fungerte riktig med 45 graders rotering.

Algoritmen implementert her er ikke effektiv og kjører gjennom bilde flere ganger. Bilde som er brukt for testing er lite og algoritmen fulfører dette relativt raskt.

Funksjonen som implementerer gjennomgangen av SE går gjennom alle pikslene i bilde som ikke er piksler fra paddingen. For hver av disse pikslene går den gjennom de nærliggende pikslene og piklsene i SE. Pikslene blir sammeliknet, hvis de er like eller at de er like bortsett fra der SE har dont care"verdier blir senterpikslen 0 og hvis de ikke er det forblir senterpikslen uendret.

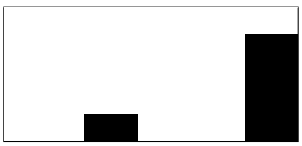
Listing 1: Morfologisk Tynning

```
import numpy as np
from skimage import io, util
from scipy import ndimage


# Reads morphological image from a txt-file
def read_binary_image_from_file():
    img = []
    lst_buff = []

    with open('txt_img_morph.txt') as fl:
        for ln in fl:
            for char in ln:
                if(char == '\n'):
                    img.append(lst_buff)
                    lst_buff = []
                elif(char == '1'):
                    lst_buff.append(1)
                elif(char == '0'):
                    lst_buff.append(0)
                else:
                    lst_buff.append(-1)
    return np.asarray(img)


def rot_mat_45(mat):
    mat_buff = np.zeros((3, 3))
    mat_buff[0, 0] = mat[1, 0]
    mat_buff[0, 1] = mat[0, 0]
    mat_buff[0, 2] = mat[0, 1]
    mat_buff[1, 0] = mat[2, 0]
    mat_buff[1, 1] = mat[1, 1]
    mat_buff[1, 2] = mat[0, 2]
```

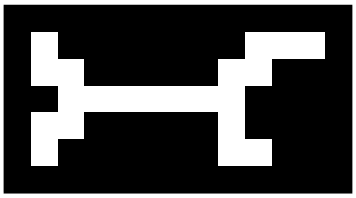
```
    mat_buff[2, 0] = mat[2, 1]
    mat_buff[2, 1] = mat[2, 2]
    mat_buff[2, 2] = mat[1, 2]
    return mat_buff


# Runs the apply SE for each pixel in image, returns result image
def thinning_iteration(img, struct_elem, n):
    for i in range(n):
        struct_elem = rot_mat_45(struct_elem)
    result_img = np.asarray((np.zeros(
        (img.shape[0], img.shape[1]))), dtype=int)
    for i in range(1, img.shape[0] - 1):
        for j in range(1, img.shape[1] - 1):
            result_img[i, j] = apply_strct_elem(j, i, img, struct_elem)
    return result_img


# Applying structuring element
def apply_strct_elem(j, i, img, struct_elem):
    curr_pix = img[i, j]
    j -= 1
    i -= 1
    org_j = j
    eq_count = 0
    for k in range(3):
        for v in range(3):
            if img[i, j] == struct_elem[k, v] or struct_elem[k, v] == -1:
                eq_count += 1
            j += 1
        i += 1
        j = org_j
    if eq_count == 9:
        return 0
    else:
        return curr_pix


def iterate(img):
    n = 0
    SE = np.asarray([[0, 0, 0], [-1, 1, -1], [1, 1, 1]])
    while True:
        if n > 7:
            return img
        img = thinning_iteration(img, SE, n)
        n += 1
```

```
# Thinn a morphological image
def thinn(img):
    img_buffer = img
    while True:
        img = iterate(img)
        if is_imgs_equal(img, img_buffer):
            return img
        img_buffer = img


def is_imgs_equal(img1, img2):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img1[i, j] != img2[i, j]:
                return False
    return True


def convert_to_ubyte(img_in):
    width, length = img_in.shape
    for i in range(width):
        for j in range(length):
            if img_in[i, j] == 1:
                img_in[i, j] = 255
    return img_in


img = read_binary_image_from_file()
img = np.pad(img, 1, mode='constant')
img = thinn(img)
print(img)
img = convert_to_ubyte(img)
io.imsave('binary_image_thinned.pbm', util.img_as_ubyte(img))
```

1.3 Resultat



Figur 2: Binært bilde lest inn fra txt-fil



Figur 3: Binært bilde etter morfologisk tynning og invertering av 0 og 1

Som man ser i figur 2 og figur 3 stemmer resultatet med figur 1.

```
Iterasjon: 3
SE:
[[ 1.  1. -1.]
 [ 1.  1.  0.]
 [-1.  0.  0.]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 0 0 0 1 1 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
Iterasjon: 4
SE:
[[ 1.  1.  1.]
 [-1.  1. -1.]
 [ 0.  0.  0.]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 0 1 1 1 1 1 0 0 0]
 [0 1 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
Iterasjon: 5
SE:
[[-1.  1.  1.]
 [ 0.  1.  1.]
 [ 0.  0. -1.]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 0 0 1 1 1 1 0 0 0]
 [0 1 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
Iterasjon: 6
SE:
[[ 0. -1.  1.]
 [ 0.  1.  1.]
 [ 0. -1.  1.]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 0 0]
 [0 0 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 0 0 1 1 1 1 0 0 0]
 [0 1 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Figur 4: Utdrag av første iterasjon med SE

I bilde ser vi at SE tynner bilde litt og litt. Totalt kjørte den gjennom 8 iterasjoner 3 ganger. Den siste gangen kjører programmet i gjennom kun for å sjekke om det er noen endringer i bilde.

Som dere ser på figur 5 og figur 6 er SE rotert 315 grader. De binære bildene stemmer også med figurene på Digital Image Processing s.661 [1].

```
Iterasjon: 0
SE:
[[ 0  0  0]
 [-1  1 -1]
 [ 1  1  1]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 1 1 1 0 0 1 1 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Figur 5: Første gjennomgang av SE i den første iterasjonen

```
Iterasjon: 7
SE:
[[ 0.  0. -1.]
 [ 0.  1.  1.]
 [-1.  1.  1.]]
Bilde
[[0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 0 0 0 0 0 1 1 0 0 0]
 [0 0 1 1 1 1 1 1 1 0 0 0 0]
 [0 1 1 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Figur 6: Siste gjennomgang av SE i den siste iterasjonen

Referanser

[1] Richard E. Woods Rafael C. Gonzalez. *Digital Image Processing.* Pearson, 2018.