

Algoritmul Niblack pentru binarizarea imaginilor.

Raport tehnic

Sinteza lucrării (Abstract)

Această lucrare prezintă în cele ce urmează particularitățile algoritmului Niblack pentru binarizarea imaginilor, precum și implementarea acestuia în limbajul de programare Python.

În această lucrare ne propunem să prezentăm eficacitatea implementării binarizării, precum și să oferim o comparație între o binarizare locală simplă și o binarizare implementată folosind Niblack.

Avantajul algoritmului Niblack este faptul că acesta poate identifica regiunile cu text din prim plan în mod corect. Pe de altă parte, o limitare a algoritmului este aceea că produce o cantitate mare de zgomot de binarizare în regiunile fără text.

Cuvinte cheie: Niblack, binarizare, thresholding

Date de intrare:

- Imaginea originală pe niveluri de gri
- Parametrii algoritmului Niblack

Date de ieșire:

- imagine binară pe niveluri de gri, obținută prin metoda Niblack

Introducere

În majoritatea aplicațiilor de prelucrare a imaginilor pixeli pe nivele de gri care aparțin obiectelor sunt complet diferiți față de cei care aparțin fundalului. Astfel binarizarea devine un simplu, dar eficient mod de separarea a obiectelor din prim plan față de fundal. Pixeli din imagine se pot împărți în două grupuri majore în funcție de nivelul lor de gri, aceste niveluri servind drept detectoare pentru distingerea între fundal și obiectul din prim plan.

Algoritmul lui Niblack oferă o binarizare mai precisă pentru imaginile pe nivele de gri, acesta identificând corect întotdeauna regiunile cu text. Acest algoritm a suferit mai multe variații pe parcursul timpului pentru a se obține rezultate mai precise.

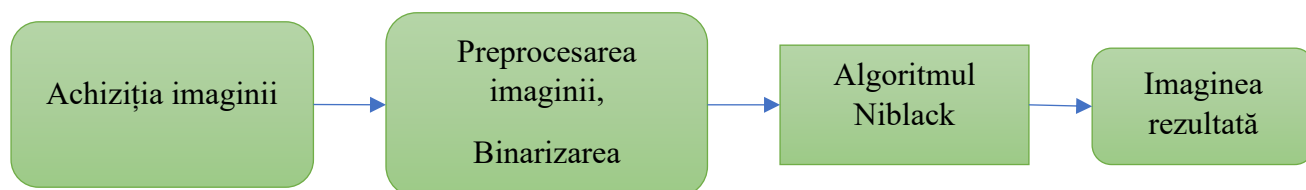
Algoritmul implementat de noi realizează o comparație între binarizarea clasică și binarizare Niblack. Diferența dintre cele două abordări a fost observată prin intermediul rezultatelor experimentale.

Binarizarea este un proces extrem de important în domenii precum cel medical, de exemplu în cazul preprocesării unei imagini de tip RMN.

Pentru implementarea la nivel de cod, am utilizat IDE-ul Spyder, folosind Python 3.8.

În urma rularii codului am obținut o comparație între imaginea originală citită, imaginea binarizată simplu și în final, imaginea binarizată prin intermediul metodei implementate.

Fundamentare teoretica



Schema 1. Schema bloc a algoritmului Niblack

Algoritmul lui Niblack determina o valoare de prag pentru fiecare pixel, prin trasarea unei ferestre dreptunghiulare peste imaginea pe niveluri de gri. Dimensiunea dreptunghiului poate sa varieze. Pragul este calculat folosind variabila locala m si deviatia standard S a tuturor pixelilor din fereastra si rezulta din derivarea urmatoare.

Tehnica Thresholding poate fi clasificată în două categorii:

- Global
- Local.

Metoda Thresholding-ului Global consideră un singur prag de intensitate pentru toti pixeli. In cazul celui Local, se calculeaza o valoare de prag pentru fiecare pixel in parte din imagine, valoare bazata pe continutul din vecinatatea fiecaruia.

Algoritmul Niblack se bazează pe următoarele ecuații:

$$T_{Niblack} = m + k * s$$

$$T_{Niblack} = m + k \sqrt{\frac{1}{NP} \sum (p_i - m)^2} = m + k \sqrt{\frac{\sum p_i^2}{NP} - m^2} = m + k \sqrt{B}$$

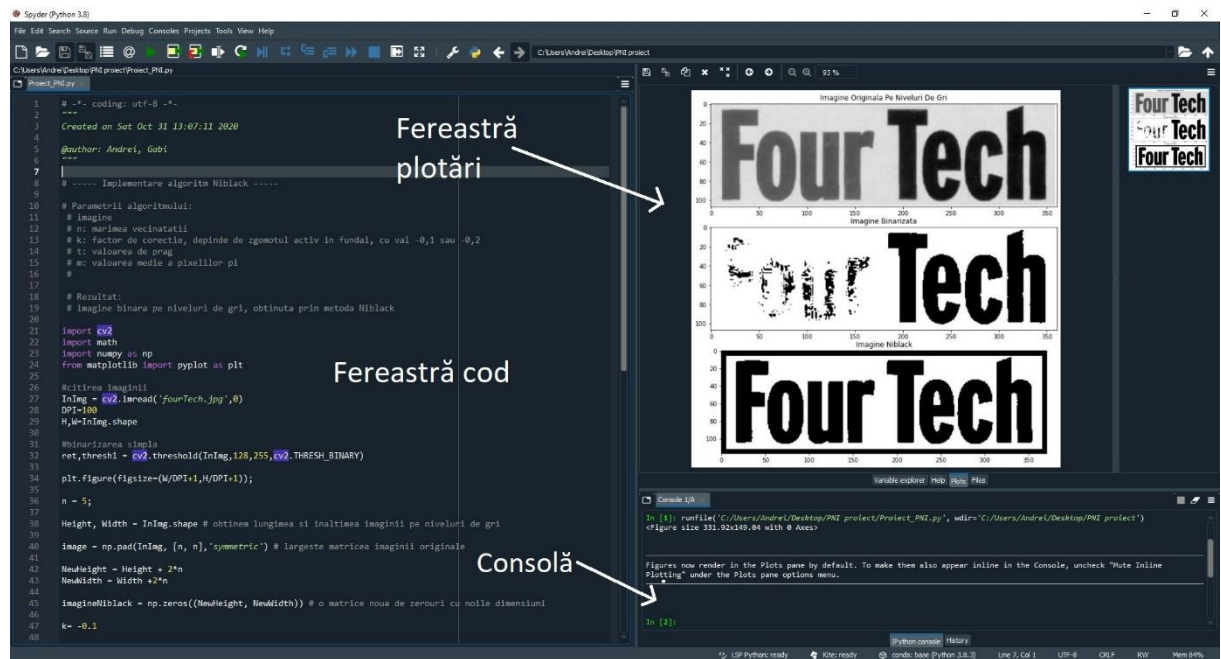
Ecuatie 1. Ecuatia de funcționare a algorimului Niblack

Unde:

- NP reprezinta numarul total de pixeli prezenti in imaginea gri
- T reprezinta valoarea de prag
- m este valoarea medie a pixelilor p_i
- k este o valoare fixa care depinde de zgomotul activ in fundal si poate lua valoarea -0,1 sau -0,2.

Implementarea soluției adoptate

Pentru implementarea acestui subiect, am ales limbajul de proiectare Python(3.8.3), rulat în cadrul mediului de programare Spyder(Anaconda3).



În cadrul implementării am inclus librăriile cv2, math, numpy, pyplot din matplotlib

```
import cv2
import math
import numpy as np
from matplotlib import pyplot as plt
```

În continuare, lucrarea va prezenta implementarea algoritmului la nivel de cod, urmărind schema bloc prezentată anterior precum și .

Achiziția imaginii

Pentru etapa de achiziție a imaginii, s-a folosit metoda “imread” din cadrul bibliotecii importate “cv2”.

```
#citirea imaginii
InImg = cv2.imread('fourTech.jpg',0)
```

Preprocesarea
imaginii,
Binarizarea

Preprocesarea imaginii s-a realizat prin pregătirea unor parametrii folosiți ulterior în cod, și anume DPI, Height, Width.

```
DPI=100  
H,W=InImg.shape
```

Pentru realizarea binarizării simple, am ales să folosim metoda “threshold”, de asemenea accesibilă din cadrul bibliotecii cv2.

```
#binarizarea simpla  
ret,thresh1 = cv2.threshold(InImg,128,255,cv2.THRESH_BINARY)
```

În continuare, se pregătesc parametrii necesari binarizării Niblack:

- obținerea înălțimii, lățimii imaginii originale pe niveluri de gri

```
Height, Width = InImg.shape # obținem lungimea și înălțimea imaginii pe niveluri de gri
```

- pregătirea unei noi matrici, bazate pe înălțimea și lungimea imaginii originale

```
image = np.pad(InImg, [n, n], 'symmetric') # largeste matricea imaginii originale  
NewHeight = Height + 2*n  
NewWidth = Width + 2*n  
imageNiblack = np.zeros((NewHeight, NewWidth)) # o matrice nouă de zerouri cu noile dimensiuni
```

Unde n, reprezintă mărimea vecinătății pixelului, aleasă în acest exemplu ca fiind 5

```
n = 5 #mărimea vecinătății
```

După preprocesare și binarizare locală, implementarea continuă cu aplicarea algoritmului Niblack asupra aceleiași imagini.

Algoritmul Niblack

Implementarea implică trasarea unei ferestre dreptunghiulare peste imaginea pe niveluri de gri. Dimensiunea dreptunghiului poate să varieze.

În implementare am ales să parcurgem matricea imaginii pe niveluri de gri, să pregătim și să trasăm mereu un chenar, în funcție de pixelul la care ne aflăm.

```
for i in range(1+n, Height+n) :
    for j in range(1+n, Width+n) :

        #trasarea chenarului
        randSus = i - math.floor(n/2 - 1/2); # Randul de sus, randul de jos
        randJos = i + math.floor(n/2);
        randSus_Jos = range(randSus, randJos+1)

        coloanaStanga = j - math.floor(n/2 - 1/2); # Coloana din stanga, dreapta
        coloanaDreapta = j + math.floor(n/2);
        coloanaStanga_Dreapta = range(coloanaStanga, coloanaDreapta+1)

        #pregatim chenarul
        chenar = [randSus_Jos, coloanaStanga_Dreapta]
```

A doua parte a secvenței de cod care implementează algoritmul Niblack urmărește formula caracteristică, calculând astfel:

-Valoarea medie a tuturor pixelilor

```
# calculam valoarea medie a tuturor pixelilor
mean_all = np.mean(np.mean((image)))
```

-Deviația standard

```
# Calculam deviatia standard
deviatiaPatrata = chenar - mean_all
deviatiaPatrata = np.mean(np.mean(deviatiaPatrata * deviatiaPatrata))
deviatia = math.sqrt(deviatiaPatrata)
```

-Calcularea valorii de prag conform formulei prezentate ulterior în fundamentarea teoretică

```
#setam o valoare de prag conform Niblack
T = mean_all + k * deviatia # T=k*s(x,y)+m(x,y) valoare prag
```

Unde parametrul k, reprezintă factorul de corecție între -0,2 și -0,1. În situația curentă, acesta a fost ales ca fiind -0.1

```
k= -0.1 #factor de corectie
```

Ultima parte a implemetării, urmărește construirea unei noi imagini cu pixelii în funcție de pragul ales,

```
#construim noua imagine, cu pixelii stabiliți în funcție de pragul ales
if (image[i][j] > T) : imagineNiblack[i][j] = 1 #daca e mai mare decat pragul, pixelul e negru
else : imagineNiblack[i][j] = 0 #daca e mai mic decat pragul, pixelul e alb
```

Precum și afișarea rezultatului obținut, folosind metoda plt (pyplot din libraria matplotlib).

```
# pregatirea imaginilor, a titlurilor, si afisarea rezultatelor :
titles = ['Imagine Originala Pe Niveluri De Gri', 'Imagine Binarizata', 'Imagine Niblack']
images = [InImg, thresh1, imagineNiblack]

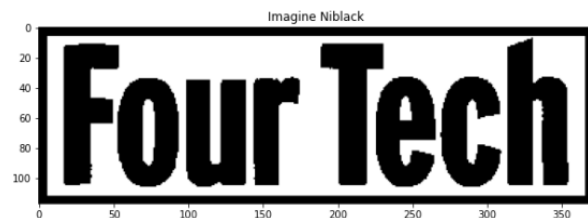
plt.figure(figsize=(1000/DPI+1,1000/DPI+1));

for i in range(0,3) :
    plt.subplot(3,1, i+1), plt.imshow(images[i], cmap= 'gray' )
    plt.title(titles[i])



plt.show()
```

După aplicarea tuturor pașilor, se obține imaginea rezultată finală.

Imaginea
rezultată



Rezultate experimentale

Nr. Crt.	Imagine Originală	Imagine Binarizată Simplu	Imagine Binarizată Niblack
1			
2			
3			
4			
5			

Tabel 1. Rezultate experimentale, esantion de 5 imagini cu text

O mai bună acuratețe a detecției zonelor cu text se poate observa în cazul binarizării folosind algoritmul Niblack. Astfel, binarizarea simplă locală tinde să omită zonele mai puțin luminate (șterse) precum cele din exemplul numărul 3. Un alt avantaj vizibil al algoritmului Niblack este diferențierea mai bună între zonele de text și fundal, în cazul în care fundalul are zone cu ton mai închis, fapt vizibil în exemplul numărul 2.

Concluzii

Lucrarea prezinta coparatiia dintre algoritmul Niblack si binarizarea simpla.

Algoritmului Niblack identifica regiunile cu text din prim plan în mod correct.

Acesta determina o valoare de prag pentru fiecare pixel.

Se poate implementa practic in domeniul medical.

Perspective de viitor pentru continuarea temei/ imbunatatirea rezultatelor curente.

O implementare mai eficientă a algoritmului în ceea ce priveşte timpul de execuţie în momentul aplicarii algoritmului asupra unei imagini de dimensiuni crescute.

Bibliografie

Senthilkumaran N, et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014, 2173-2176

Dana Dabiri and Charles Pecora,*Particle image identification*

Wan Azani Mustafa and Mohamed Mydin M. Abdul Kader,*Binarization of Document Image Using Optimum Threshold Modification*

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.439.509&rep=rep1&type=pdf>

Anexe

coding: utf-8 -*-

"""

Created on Sat Oct 31 13:07:11 2020

@author:Gabi

"""

----- Implementare algoritm Niblack -----

Parametrii algoritmului:

imagine

n: marimea vecinatatii

k: factor de corectie, depinde de zgomotul activ in fundal, cu val -0,1 sau -0,2

t: valoarea de prag

m: valoarea medie a pixelilor pi

#

Rezultat:

imagine binara pe niveluri de gri, obtinuta prin metoda Niblack

import cv2

import math

import numpy as np

from matplotlib import pyplot as plt

#citirea imaginii

InImg = cv2.imread('handwriting.jpg',0)

DPI=100

H,W=InImg.shape

#binarizarea simpla

ret,thresh1 = cv2.threshold(InImg,128,255,cv2.THRESH_BINARY)

plt.figure(figsize=(W/DPI+1,H/DPI+1));

n = 2 #mărima vecinătății

k= -0.2 #factor de corectie

Height, Width = InImg.shape # obtinem lungimea si inaltimea imaginii pe niveluri de gri

image = np.pad(InImg, [n, n], 'symmetric') # largeste matricea imaginii originale

```

NewHeight = Height + 2*n
NewWidth = Width + 2*n

imageNiblack = np.zeros((NewHeight, NewWidth)) # o matrice noua de zerouri cu noile dimensiuni

for i in range(1+n, Height+n) :
    for j in range(1+n, Width+n) :
        #trasarea chenarului

        randSus = i - math.floor(n/2 - 1/2); # Randul de sus, randul de jos
        randJos = i + math.floor(n/2);
        randSus_Jos = range(randSus, randJos+1)

        coloanaStanga = j - math.floor(n/2 - 1/2); # Coloana din stanga, dreapta
        coloanaDreapta = j + math.floor(n/2);
        coloanaStanga_Dreapta = range(coloanaStanga, coloanaDreapta+1)

        #pregatim chenarul
        chenar = [randSus_Jos, coloanaStanga_Dreapta]

        # calculam valoarea medie a tuturor pixelilor
        mean_all = np.mean(np.mean((image)))

        # Calculam deviatia standard
        deviatiaPatrata = chenar - mean_all

        deviatiaPatrata = np.mean(np.mean(deviatiaPatrata * deviatiaPatrata))

        deviatia = math.sqrt(deviatiaPatrata)

        #setam o valoare de prag conform Niblack
        T = mean_all + k * deviatia # T=k*s(x,y)+m(x,y) valoare prag

        #construim noua imagine, cu pixelii stabiliti in functie de pragul ales
        if (image[i][j] > T) : imageNiblack[i][j] = 1 #daca e mai mare decat pragul, pixelul e
negru

        else : imageNiblack[i][j] = 0 #daca e mai mic decat pragul, pixelul e alb

# pregatirea imaginilor, a titlurilor, si afisarea rezultatelor :
titles = ['Imagine Originala Pe Niveluri De Gri', 'Imagine Binarizata', 'Imagine Niblack']

images = [InImg, thresh1, imageNiblack]

```

```
plt.figure(figsize=(1000/DPI+1,1000/DPI+1));  
for i in range(0,3) :  
    plt.subplot(3,1, i+1), plt.imshow(images[i], cmap= 'gray' )  
    plt.title(titles[i])  
plt.show()
```