# **Raportul tehnic sub forma unui articol – Algoritmul Niblack pentru binarizarea imaginilor**

**Membrii echipei**

**Manager**: Kiraly Natalia Iulia *(e-mail : kiraly.io.natalia@student.utcluj.ro)*

**Dezvoltator**: Asavei Florin Constantin *(e-mail:* *asavei.co.florin@student.utcluj.ro)*

**Nume îndrumător**: Florea Camelia Costina

**Data predării proiectului (zi/lună/an)**: 6/01/2022

## Sinteza lucrării (Abstract)

Algoritmul Niblack este un algoritm de binarizare a imaginii folosit pentru imagini, unde fundalul nu este uniform, in special pentru recunoasterea textului. In loc de calcularea unui prag unic global pentru intreaga imagine, algoritmul sectioneaza imaginea si calculeaza pragul pentru fiecare sectiune in parte folosind formule specifice care iau in calcul media si deviatia standard.

Programul a fost realizat in Google Colab folosind limbajul python si descrie mai multe metode de realizare a binarizarii imaginii. S-au folosit algoritmi preluati de pe skimage, dar si algoritmi proprii pentru compararea rezultatelor.

Ca date de intrare s-a folosit o imagine preluata de pe internet care s-a transformat mai apoi pe nivele de gri pentru a se putea aplica algoritmul.

Ca date de iesire avem 4 imagini binarizate prin variatiuni a algoritmului Niblack.

## Introducere

In programul realizat sunt descrise mai multe metode de binarizare a imaginii, variatiuni ale algoritmului Niblack si sunt evidentiate avantajele si dezavantajele fiecarei metode. Astfel, se pot compara rezultatele obtinute pentru a determina metoda cea mai avantajoasa in recunoasterea textului din imagine cu imperfectiuni minime. Histogramele imaginilor obtinute in final, ne arata ca binarizarea a reusit si imaginile au doar doua valori ale pixelilor (0 – negru si 255 – alb).

## Fundamentare teoretica

Fig 1. Schema Bloc

Pentru fiecare pixel din imagine se calculeaza media si dispersia nivelelor de gri ale pixelilor vecini pe o fereastra cu dimensiuni predefinite, parcurgand astfel intreaga imagine.

Pragul este calculat pe baza formulei: ,unde k este o constanta, stdev este deviatia standard, iar m este media.

Acest algoritm nu presupune un prag unic ceea ce inseamna ca se adapteaza mai bine la particularitatile locale ale imaginii care urmeaza sa fie procesata.

Pentru imagini in tonuri de gri, operatia de binarizare se scrie matematic astfel:

φ(x) =

Fig 2. Ecuatii folosite

,unde NP este numarul de pixeli [1]

S-au utilizat 4 metode pentru compararea rezultatelor:

|  |  |
| --- | --- |
| ***Metoda*** | ***Descriere*** |
| 1. Algoritm Niblack cu functia importata din skimage | Functia importata sectioneaza imaginea in ferestre (dimensiune data ca parametru), si calculeaza pragul in functie de media si deviatia standard. Algoritmul este complex si se trateaza toate erorile care pot aparea. In plus marginile sunt prelucrate si ele.[2]  Rezultatul e o imagine booleana.  *Avantaj:* Zonele de umbra sunt eliminate |
| 1. Algoritm cu prag unic | Se calculeaza media si deviatia standard a intregii imagini si se calculeaza pragul.  Se parcurge imaginea pixel cu pixel si se compara cu pragul.  Rezultatul e o imagine booleana.  *Dezavantaj:* Zonele de umbra nu sunt eliminate |
| 1. Algoritm Niblack propriu | Se creeaza o imagine alba cu dimensiunile imaginii care urmeaza sa fie prelucrata.  Algoritmul sectioneaza imaginea in ferestre (dimensiune data ca parametru) si se extrage aceasta fereastra pentru a putea calcula pragul.  Se adauga valori in imaginea obtinuta fereastra cu fereastra.  *Avantaje:* Zonele de umbra sunt eliminate  *Dezavantaj:* Marginile raman neprelucrate. |
| 1. Algoritm Niblack cu prag pe randuri | Algoritmul parcurge imaginea si calculeaza pragul pentru un numar de randuri date ca parametru.  Rezultatul e o imagine booleana.  *Dezavantaj:* Zonele de umbra nu sunt eliminate |

Fig 3. Descriere metode

## Implementarea soluţiei adoptate

S-au utilizat urmatoarele librarii pentru a putea importa, prelucra si afisa imagini: cv2, matplotlib.pyplot si numpy.

import cv2

import matplotlib.pyplot as plt

import numpy as np

Conversia imaginii pe nivele de grii s-a realizat utilizand secventa de mai jos:

R, G, B = InImg[:,:,0], InImg[:,:,1], InImg[:,:,2]

ImgGray = 0.2989 \* R + 0.5870 \* G + 0.1140 \* B

1. **Descrierea primei metode (Algoritm Niblack cu functia importata din skimage)**

Functia din skimage s-a importat astfel:

from skimage.filters import (threshold\_niblack)

Pragul pragul s-a calculat apeland functia avand ca parametrii dimensiunea ferestrei, imaginea pe nivele de gri si constanta k.

thresh\_niblack = threshold\_niblack(ImgGray, window\_size=window\_size, k=0.9)

Imaginea rezultata se creeaza comparand pragul astfel:

Img\_niblack = ImgGray > thresh\_niblack

1. **Descrierea metodei 2 (Algoritm cu prag unic)**

Calcularea mediei si deviatiei standard a intregii imagini:

mean=ImgGray.mean()

stdev=ImgGray.std()

Calcularea pragului si compararea cu pixelii din imaginea originala:

T=mean+k\*stdev

Img\_T\_unic = ImgGray > T

1. **Descrierea metodei 3 (Algoritm Niblack propriu)**

Crearea imaginii noi cu dimensiunile imaginii originale si alba

newImg= np.zeros(ImgGray.shape)+255

Imaginea originala se parcurge si pe linii si pe coloane si se extrage fereastra cu dimensiunile date intr-o noua imagine pentru a se putea calcula pragul pentru fereastra respectiva. Se compara pixelii din fereastra cu valoarea calculata pentru fereastra respectiva si se completeaza in imaginea alba cu valoarea corespunzatoare a pixelilor (0 sau 1) pe pozitiile prelucrate. Procesul se repeta pana la parcurgerea intregii imaginii in ferestre.

for i in range(w+1,len(ImgGray)-w):

  for j in range(w+1,len(ImgGray[i])-w):

    formedImg=[]

    for x in range(0, len(ImgGray[i-w:i+w])): #extragerea ferestrei pentru calcularea pragului pentru fiecare fereastra

      formedImg.append(ImgGray[i-w:i+w][x][j-w:j+w]) #formare imagine noua cu dimensiunea ferestrei

    mean=np.asarray(formedImg).mean()

    stdev=np.asarray(formedImg).std()

    T=mean+k\*stdev

    if ImgGray[i][j] < T:#comparare pixeli

      newImg[i][j] = 0

    else:

      newImg[i][j] = 255

1. **Descrierea metodei 4 (Algoritm Niblack cu prag pe randuri)**

Se afla dimensiunea imaginii:

h,w=ImgGray.shape

Imaginea se parcurge imaginea in blocuri de x randuri si pentru fiecare bloc se calculeaza pragul si se creeaza imaginea noua:

while i<h:

  mean=ImgGray[i:i+win].mean()

  print(i)

  print("Media:",str(mean))

  stdev=ImgGray[i:i+win].std()

  T=mean+k\*stdev

  ImgWin[i:i+win]=ImgGray[i:i+win] > T

  i+=win

Afisarea imaginilor se realizeaza astfel:

plt.imshow(InImg)

plt.show()

Afisarea histogramelor se realizeaza astfel:

plt.hist(ImgGray.ravel(), 256,[0,255])

plt.show()

## Rezultate experimentale

Imaginea originala se transforma in nivele de gri:

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

Folosind functia din skimage, zonele de umbra dispar, iar scrisul se deosebeste de fundal. Raman cateva zone de zgomot care nu afecteaza calitatea imaginii si care se pot elimina usor folosind alte metode.

O imagine care conține text

Descriere generată automat

Folosind un prag unic si un k mult mai mic, imaginea binarizata are o zona de umbra care iterfereaza cu scrisul.

O imagine care conține text, chitanță, captură de ecran

Descriere generată automat

Folosind algoritmul propriu Niblack, zona de umbra dispare. Ca un dezavantaj, marginile de dimensiunea ferestrei raman neprocesate.

O imagine care conține text, chitanță, captură de ecran

Descriere generată automat

Rezultatul cel mai dezamagitor s-a obtinut cu pragurile calculate pe randuri deoarece umbrele se accentueaza.

O imagine care conține text

Descriere generată automat

**Testarea algoritmului propriu Niblack pentru alte imagini (metoda 3)**

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat O imagine care conține text

Descriere generată automat

## ~~Manual de utilizare~~

* ~~Prezentare mod de lucru/rulare aplicație~~

## Concluzii

Analizand imaginile rezultate consideram ca metode optime algoritmul Niblack folosind functia din skimage, dar si algoritmul propriu deoarece reusesc sa elimine umbrele astfel incat scrisul din imagine sa fie lizibil.

Acest algoritm are rezultate bune pentru recunoasterea textului in cazul unui fundal neuniform.

Pentru imbunatatirea rezultatelor curente se pot implementa algoritmi de filtrare care sa elimine zgomotul in totalitate.

## Bibliografie

[1] <https://link.springer.com/content/pdf/10.1007/978-3-642-21587-2_25.pdf>

[2] <https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_niblack_sauvola.html>

## Anexe

# ***Algoritmul Niblack⛄***

StudentName1 = "Kiraly Natalia"

StudentName2 = "Asavei Florin"

StudentSpecialization = "TST RO"

print("This work is realised by:")

print(StudentName1," and ",StudentName2, " - ", StudentSpecialization)

Importarea surselor

import cv2

import matplotlib.pyplot as plt

import numpy as np

Importul imaginii pe care urmeaza sa fie implementat algoritmul

InputIm\_FileName="imagine.png"

InImg = cv2.cvtColor(cv2.imread('imagine.png'), cv2.COLOR\_BGR2RGB)

Afisare imagine originala

plt.imshow(InImg)

plt.suptitle('Imaginea originala')

plt.show()

Afisare imagine pe nivele de gri

#transformare

R, G, B = InImg[:,:,0], InImg[:,:,1], InImg[:,:,2]

ImgGray = 0.2989 \* R + 0.5870 \* G + 0.1140 \* B

#afisare

plt.imshow(ImgGray, cmap='gray')

plt.suptitle('Imaginea pe nivele de gri')

plt.show()

Afisarea matricii imaginii pe nivele de gri

ImgGray

*I. Aplicarea algoritmului Niblack folosind functia importata din skimage (pentru compararea rezultatelor)*

from skimage.filters import (threshold\_niblack)

window\_size = 51 #fereastra pentru care se calculeaza pragul

thresh\_niblack = threshold\_niblack(ImgGray, window\_size=window\_size, k=0.9)

Img\_niblack = ImgGray > thresh\_niblack #se creeaza o imagine doar cu 0 si 1(boolean)

# se pune 1 daca e adevarata conditia, 0 in rest

#afisare imagini

plt.imshow(ImgGray,cmap='gray')

plt.suptitle('Imaginea pe nivele de gri')

plt.show()

plt.imshow(Img\_niblack,cmap='gray')

plt.suptitle('Imaginea formata')

plt.show()

## II. Aplicarea algoritmului cu prag unic folosind algoritm propriu (pentru compararea rezultatelor)

Calcularea mediei intregii imagini

mean=ImgGray.mean()

print("Media:",str(mean))

Calculare deviatiei standard a intregii imagini

stdev=ImgGray.std()

print("Deviatia standard:",str(stdev))

Calcularea pragului unic pentru constanta k=-0.9 (determinata experimental)

k=-0.9

T=mean+k\*stdev

print("Pragul unic calculat:",str(T))

Img\_T\_unic = ImgGray > T#se creeaza o imagine doar cu 0 si 1(boolean)

# se pune 1 daca e adevarata conditia, 0 in rest

#afisare imagini

plt.imshow(ImgGray,cmap='gray')

plt.suptitle('Imaginea pe nivele de gri')

plt.show()

plt.imshow(Img\_T\_unic,cmap='gray')

plt.suptitle('Imaginea formata')

plt.show()

## III. Aplicarea algoritmului Niblack folosind algoritm propriu

from PIL import Image

k=-0.9

#se creeaza o imagine noua cu caracteristicile imaginii originale(inaltime+latime), alba

newImg= np.zeros(ImgGray.shape)+255

w = 25 #dimensiunea fereastrei care urmeaza sa fie folosita

for i in range(w+1,len(ImgGray)-w):

  for j in range(w+1,len(ImgGray[i])-w):

    formedImg=[]

    for x in range(0, len(ImgGray[i-w:i+w])): #extragerea ferestrei pentru calcularea pragului pentru fiecare fereastra

      formedImg.append(ImgGray[i-w:i+w][x][j-w:j+w]) #formare imagine noua cu dimensiunea ferestrei

    mean=np.asarray(formedImg).mean()

    stdev=np.asarray(formedImg).std()

    T=mean+k\*stdev

    if ImgGray[i][j] < T:#comparare pixeli

      newImg[i][j] = 0

    else:

      newImg[i][j] = 255

#afisare imagini

plt.imshow(ImgGray,cmap='gray')

plt.suptitle('Imaginea pe nivele de gri')

plt.show()

plt.imshow(newImg,cmap='gray')

plt.suptitle('Imaginea formata')

plt.show()

## IV. Aplicarea algoritmului cu prag pe randuri folosind un algoritm propriu(pentru compararea rezultatelor)

k=-0.9

h,w=ImgGray.shape

print('width:  ', w) #latime in pixeli

print('height: ', h) #inaltime in pixeli

i=0

ImgWin=ImgGray.copy()

#numar de randuri pentru fiecare fereastra (dimensiunea feresrei [latime x numar randuri])

win=25

while i<h:

  mean=ImgGray[i:i+win].mean()

  print(i)

  print("Media:",str(mean))

  stdev=ImgGray[i:i+win].std()

  T=mean+k\*stdev

  ImgWin[i:i+win]=ImgGray[i:i+win] > T

  i+=win

#afisare imagini

plt.imshow(ImgGray,cmap='gray')

plt.suptitle('Imaginea formata cu prag pe randuri')

plt.show()

plt.imshow(ImgWin,cmap='gray')

plt.suptitle('Imaginea formata cu prag pe randuri')

plt.show()

## V. Afisarea histogramelor pentru imaginea pe nivele de gri si pentru cele 4 metode

plt.hist(ImgGray.ravel(), 256,[0,255])

plt.suptitle('Imaginea pe nivele de gri')

plt.show()

plt.hist(255\*Img\_niblack.ravel(), 256,[0,255])

plt.suptitle('Imaginea folosind Alg. Niblack (skimage)')

plt.show()

plt.hist(255\*Img\_T\_unic.ravel(), 256,[0,255])

plt.suptitle('Imaginea folosind Alg. cu prag unic')

plt.show()

plt.hist(newImg.ravel(), 256,[0,255])

plt.suptitle('Imaginea folosind Alg. Niblack propriu')

plt.show()

plt.hist(255\*ImgWin.ravel(), 256,[0,255])

plt.suptitle('Imaginea folosind Alg. cu prag pe randuri')

plt.show()