SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

DIGITALNA OBRADA I ANALIZA SLIKE

Laboratorijske vježbe

Maja Braović / Damir Krstinić

Split, 2018.

Sadržaj

Popis oznaka i kratica

BSDS500 – Berkeley Segmentation Data Set

FESB – Fakultet elektrotehnike, strojarstva i brodogradnje

GT – Ground Truth

HSI – Hue Saturation Intensity

HSV – Hue Saturation Value

MLID – Mediterranean Landscape Image Dataset

OpenCV – Open Source Computer Vision

RGB – Red Green Blue

Važne informacije prije početka laboratorijskih vježbi

Prof. Darko Stipaničev

- Ured: A413
- E-mail: darko.stipanicev@fesb.hr
- Broj telefona: 305-643

Izv. prof. Damir Krstinić

- Ured: B502
- E-mail: damir.krstinic@fesb.hr
- Broj telefona: 305-895

Dr. sc. Maja Braović

- Ured: B501
- E-mail: maja.braovic@fesb.hr
- Broj telefona: 305-601

Za svaku laboratorijsku vježbu potrebno je napisati izvještaj i postaviti ga na e-Learning portal. Vremensko ograničenje za predaju svih izvještaja je do kraja posljednjeg tjedna nastave.

Na svim laboratorijskim vježbama iz Digitalne obrade i analize slike koristiti će se programski jezik Python. Programi se u Pythonu pišu u datoteci s ekstenzijom .py, a pokreću se naredbom python mojProgram.py (za Python 2) ili python3 mojProgram.py (za Python 3).

Budući da su neke laboratorijske vježbe složenije i duže od drugih, njih ćemo obrađivati duže (tj. u više termina) nego one kraće.

Da bi se laboratorijske vježbe položile potrebno ih je sve odraditi. U slučaju izostanka sa jedne laboratorijske vježbe ona se može nadoknaditi pri kraju semestra za vrijeme nadoknadi laboratorijskih vježbi. Osim u iznimnim slučajevima (npr. duža bolest koju je potrebno dokazati liječničkom ispričnicom, predstavljanje Fakulteta na studentskim natjecanjima i sl.), na nadoknadama se ne može nadoknaditi više od jedne laboratorijske vježbe.

1 Uvod u obradu i analizu digitalne slike

U ovoj laboratorijskoj vježbi upoznati ćemo se s osnovnom terminologijom koja se koristi prilikom obrade i analize digitalne slike.

1.1 Osnovne operacije na slici

Primjer učitavanja i prikazivanja slike u Pythonu pomoću biblioteke OpenCV (Open Source Computer Vision):

Ime datoteke u kojoj se nalazi programski kod: **primjer1.py**.

```
import cv2

slika = cv2.imread('slika.jpg')

cv2.namedWindow('Prva slika')
cv2.imshow('Prva slika', slika)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Pokretanje programa: **python3 primjer1.py**.

Primjer spremanja slike:

```
import cv2
slika = cv2.imread('slika.jpg')
cv2.imwrite('moja_slika.jpg', slika)
```

Primjer dobivanja informacija o slici (broj stupaca, redaka i kanala):

```
import cv2

slika = cv2.imread("slika.jpg")

retci, stupci, kanali = slika.shape

print('Broj redaka je ' + str(retci) + '.')
print('Broj stupaca je ' + str(stupci) + '.')
print('Broj kanala je ' + str(kanali) + '.')
```

Zadatak 1. U Pythonu učitajte i prikažite sliku u boji, te ispišite broj redaka, stupaca i kanala te slike.

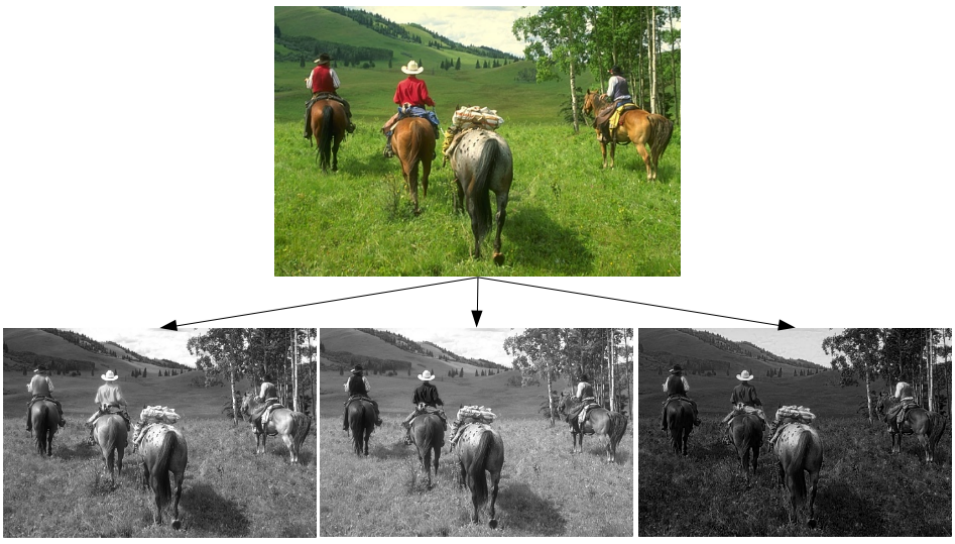
Primjer promjene dimenzija slike:

```
import cv2

slika = cv2.imread('slika.jpg')

# Slika se umanjuje preko postotaka
nova_slika_1 = cv2.resize(slika, (0, 0), fx = 0.7, fy = 0.7)

# Slika se umanjuje preko novih brojeva redaka i stupaca
nova_slika_2 = cv2.resize(slika, (100, 200))

cv2.imwrite('slika_1.jpg', nova_slika_1)
cv2.imwrite('slika_2.jpg', nova_slika_2)
```

Zadatak 2. U Pythonu učitajte sliku u boji i uvećajte njezine dimenzije na način da slika bude duplo veća no što je u originalu.

1.2 Prostori boja

Prostor boja je matematički model koji definira određeni rang boja te time određuje način na koji se slika prikazuje. Najpoznatiji prostor boja je RGB (engl. Red Green Blue), no uz njega postoje i brojni drugi, na primjer HSV (engl. Hue Saturation Value), HSI (engl. Hue Saturation Intensity), CIE L*a*b* i CIE L*u*v*.



Slika 1.1: *Slika prikazuje rastav RGB slike na tri kanala*

Različiti prostori boja se koriste u automatskoj klasifikaciji i segmentaciji digitalne slike zbog toga što različite klase vidljive na slikama (npr. oblaci, more, vatra, snijeg) nisu jednako uočljivi u svim prostorima boja, tj. u nekim prostorima boja su uočljivije nego u drugima.

Slike prikazane pomoću RGB prostora boja sastoje se od 3 kanala: crvenog, zelenog i plavog. Slika 1.1 prikazuje rastav RGB slike na 3 kanala. Za ulaznu sliku na slici 1.1 korištena je slika iz baze slika Berkeley Segmentation Data Set (BSDS500) [1].

Primjer rastavljanja slike na kanale:

```
import cv2

slika = cv2.imread('slika.jpg')

b, g, r = cv2.split(slika)

cv2.imwrite('b.jpg', b)
cv2.imwrite('g.jpg', g)
cv2.imwrite('r.jpg', r)
```

Iz gornjeg primjera može se primijetiti da su kanali RGB slike u OpenCV biblioteci obrnuti, dakle koristi se BGR struktura slike. BGR strukturu su u prošlosti često koristili proizvođači kamera i softvera, pa se danas koristi iz povijesnih razloga [2].

Zadatak 3. Učitajte sliku u boji u Pythonu i rastavite je na kanale. Po čemu se kanali razlikuju? Koji vam izgleda najprirodnije?

Primjer konverzije slike iz RGB prostora boja u HSV prostor boja:

```
import cv2

slika_bgr = cv2.imread('slika.jpg')

slika_hsv = cv2.cvtColor(slika_bgr, cv2.COLOR_BGR2HSV)

cv2.namedWindow('HSV')
cv2.imshow('HSV', slika_hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Zadatak 4. Napišite program u Pythonu koji će konvertirati sliku u boji u neki drugi prostor boja. Prikažite novu sliku i rastavite je na kanale. Po čemu se ti kanali razlikuju od kanala RGB slike?

1.3 Pristup pikselima slike

Primjer pristupa pikselima digitalne slike:

```
import cv2

slika = cv2.imread('slika.jpg')

cv2.namedWindow('slika')
cv2.imshow('slika', slika)
cv2.waitKey(0)
cv2.destroyAllWindows()

retci = slika.shape[0]
stupci = slika.shape[1]
kanali = slika.shape[2]

for k in range(0, kanali):
    for i in range(0, retci):
        for j in range(0, stupci):
            slika[i, j, k] = 255

cv2.namedWindow('slika')
cv2.imshow('slika', slika)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Zadatak 5. Učitajte sliku u boji u Pythonu i sve bijele piksele zamijenite crnima, tj. sve piksele čija je vrijednost u R, G i B kanalima 255 postavite na nulu.

1.4 Pretvaranje slike u boji u sliku u razinama sive boje

Za pretvaranje slike u boji u sliku u razinama sive boje (engl. *grayscale image*) najčešće se koristi slijedeća formula:

$$siva\_slika = 0.299 \cdot crveni\_kanal + 0.587 \cdot zeleni\_kanal + 0.114 \cdot plavi\_kanal \tag{1.1}$$

Parametri gornje formule određeni su eksperimentalno, tj. znanstvenici su utvrdili da većini ljudi zeleni kanal slike izgleda najprirodnije, pa je zato njemu dodijeljenja najveća težina. Analogno su određeni parametri za crveni i plavi kanal.

Primjer pretvaranja slike u boji u sliku u razinama sive boje:

```
import cv2

slika_bgr = cv2.imread('slika.jpg')

slika_gray = cv2.cvtColor(slika_bgr, cv2.COLOR_BGR2GRAY)

cv2.namedWindow('Grayscale')
cv2.imshow('Grayscale', slika_gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Zadatak 6. Napišite program u Pythonu koji će pretvoriti sliku u boji u sliku u razinama sive boje. Nemojte koristiti već postojeću funkciju iz OpenCV biblioteke, već napišite sami svoju funkciju. Funkciju iz OpenCV biblioteke možete koristiti za usporedbu rezultata.

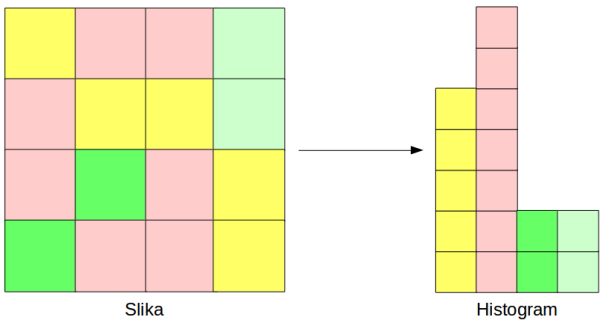
1.5 Prikaz većeg broja slika u istom prozoru

Ako u Pythonu želite usporediti dvije slike možete ih obje prikazati u istome prozoru na slijedeći način:

```
import cv2
from matplotlib import pyplot as plt

prva_slika = cv2.imread('prva_slika.jpg'
druga_slika = cv2.imread('druga_slika.jpg'

plt.subplot(121)
plt.imshow(prva_slika)
plt.title('Prva slika')
plt.xticks([])
plt.yticks([])

plt.subplot(122)
plt.imshow(druga_slika)
plt.title('Druga slika')
plt.xticks([])
plt.yticks([])

plt.show()
```

2 Histogrami

Histogram slike prikazuje broj piksela na slici po razinama intenziteta. Na primjer, ako se na slici nalazi 1000 piksela koji imaju intenzitet 32, onda će pripadajući stupac histograma imati visinu 1000. Shematski prikaz histograma dan je na slici 2.1.

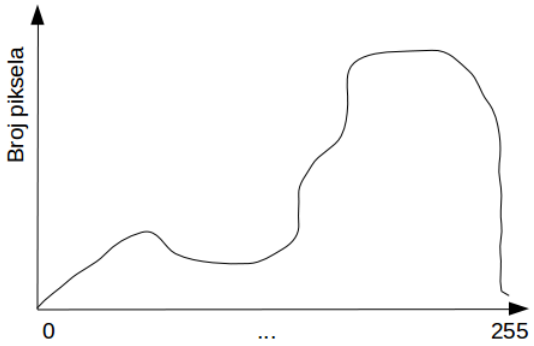


Slika 2.1: *Shematski prikaz histograma*

Zadatak 1. Nacrtajte histogram za sliku prikazanu na slici 2.2. Histogram možete skicirati u proizvoljnom programu.

8	8	4	2	5	0
8	8	1	2	0	0
6	5	3	2	0	6
4	3	2	2	0	7
1	0	4	5	4	4
1	0	4	0	6	6

Slika 2.2: *Primjer digitalne slike*



Slika 2.3: *Primjer histograma*

Zadatak 2. Imajući na umu da 0 označava crnu boju a 255 bijelu boju, što možete zaključiti o slici čiji je histogram prikazan na slici 2.3?

Primjer prikaza histograma za sliku u razinama sive boje u Pythonu:

```
import cv2
from matplotlib import pyplot as plt

slika_bgr = cv2.imread('slika.jpg')
slika_gray = cv2.cvtColor(slika_bgr, cv2.COLOR_BGR2GRAY)

histogram = cv2.calcHist([slika_gray], [0], None, [256], [0,256])

plt.hist(slika_gray.ravel(), 256, [0,256])
plt.title('Histogram slike')
plt.xlabel('Intenziteti sive boje')
plt.ylabel('Broj piksela')
plt.show(block = False)
plt.pause(3)
plt.close()
```

U gornjem primjeru funkcija calcHist() prima 5 argumenata:

1. slika,
2. broj određenog kanala slike za koji se računa histogram,
3. maska koja označava dio slike za koji se računa histogram, a postavlja se na 'None' ako se histogram računa za cijelu sliku,
4. broj stupaca u histogramu
5. opseg boja koje se koriste.

Zadatak 3. U Pythonu učitajte sliku u boji i prikažite histogram njezinog zelenog kanala.

Zadatak 4. U Pythonu sami napišite funkciju za računanje histograma slike. Usporedite dobiveni rezultat s onim koji se dobije pomoću Pythonove ugrađene funkcije.

2.1 Rastezanje histograma

Rastezanje histograma (engl. *histogram stretching*, *histogram normalization*) je operacija prilikom koje se histogram slike razvuče od 0 do 255, odnosno po svim mogućim intenzitetima. Rezultat ove operacije je poboljšanje kontrasta slike kojoj histogram pripada.

Primjer rastezanja histograma u Pythonu:

```
from PIL import Image, ImageOps

slika = Image.open("slika.jpg")
slika.show()

nova_slika = ImageOps.autocontrast(slika, cutoff = 0)
nova_slika.show()
```

Zadatak 5. U Pythonu učitajte sliku u razinama sive boje te na nju primijenite operaciju rastezanja histograma. Usporedite dobivenu sliku i njezin histogram s originalnom slikom i njenim histogramom. Što se dogodi s slikom ako promijenite *cutoff* vrijednost?

2.2 Ujednačavanje histograma

Ujednačavanje histograma (engl. *histogram equalization*) je operacija prilikom koje se histogram slike prepravi na način da svi njegovi stupci imaju otprilike jednaku visinu. Rezultat ove operacije je poboljšanje kontrasta slike kojoj histogram pripada.

Primjer ujednačavanja histograma u Pythonu:

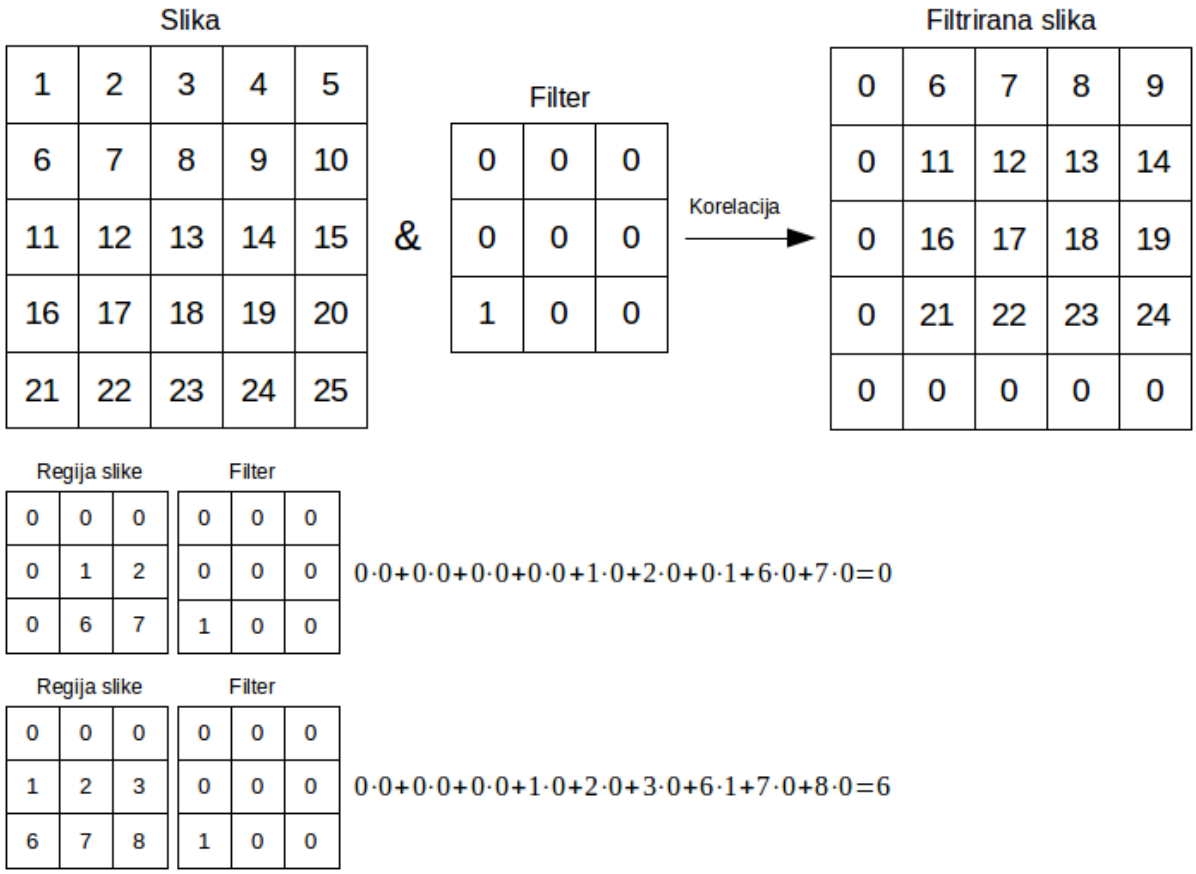
```
from PIL import Image, ImageOps

slika = Image.open("slika.jpg")
slika.show()

nova_slika = ImageOps.equalize(slika, mask = None))
nova_slika.show()
```

Zadatak 6. U Pythonu učitajte sliku u razinama sive boje te na nju primijenite operaciju ujednačavanja histograma. Usporedite dobivenu sliku i njezin histogram s originalnom slikom i njenim histogramom.

3 Korelacija i konvolucija

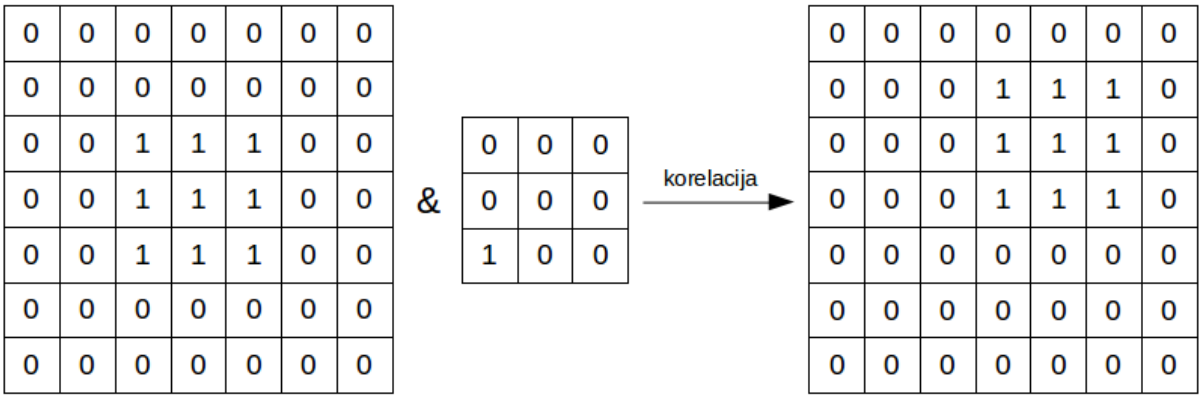
Korelacija (engl. *correlation*) i konvolucija (engl. *convolution*) spadaju među najvažnije operacije u digitalnoj obradi i analizi slike. Obje se koriste za filtriranje digitalnih slika pomoću različitih filtera, a jedina razlika je u tome što se kod konvolucije filter rotira za 180°, a kod korelacije se ne rotira. Ako se izričito ne navede da se negdje koristi konvolucija, uvijek se pretpostavlja da se koristi korelacija.

Kod filtriranja digitalnih slika potrebno je najprije odabrati ili konstruirati potrebni filter ili kernel. Filter je matrica koja najčešće ima neparne dimenzije kao što su na primjer 3x3, 5x5, 7x7 i 9x9 piksela. Korelacija se obavlja na način da se filter postavlja preko svakog piksela slike te se zatim obavlja operacija prikazana na slici 3.1. Na slici 3.1 je pretpostavljeno da se središnji element filtera postavlja na trenutni piksel slike, ali u praksi se može bilo koji element filtera postaviti na trenutni piksel slike. Važno je napomenuti da se najčešće pretpostavlja da se uokolo slike nalaze nule ako se dogodi da filter izleti izvan okvira slike.

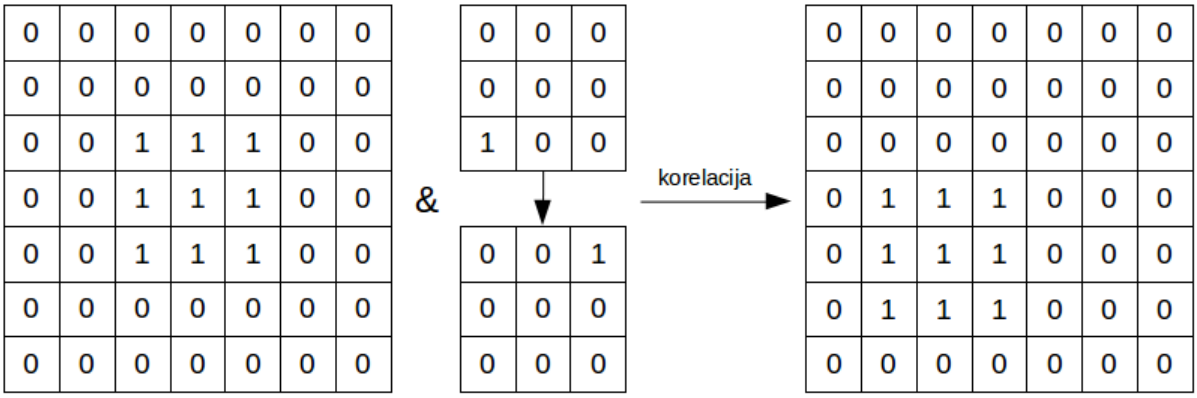


Slika 3.1: *Primjer računanja korelacije*

Primjer korelacije na digitalnoj slici u Pythonu:



Slika 3.2: *Primjer korelacije na slici*



Slika 3.3: *Primjer konvolucije na slici*

```
import cv2
import numpy as np

slika = cv2.imread('slika.jpg', cv2.IMREAD_GRAYSCALE)

# Kreiranje i ispis filtera
moj_filter = np.zeros((50,50), np.float32)
print(moj_filter)
moj_filter[49,0] = 1
print()
print(moj_filter)

# Primjena filtera na sliku
filtrirana_slika = cv2.filter2D(slika, -1, moj_filter,
anchor=(-1,-1), borderType=cv2.BORDER_CONSTANT)

cv2.namedWindow("Originalna slika")
cv2.namedWindow("Filtrirana slika")
cv2.imshow("Originalna slika", slika)
cv2.imshow("Filtrirana slika", filtrirana_slika)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Zadatak 1. Ručno primijenite filter $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ na prvi redak matrice prikazane na slici 3.4. Ovaj filter pripada skupini filtera za izoštravanje slike o kojima će biti više riječi u slijedećoj laboratorijskoj vježbi. Pretpostavite da se oko matrice prikazane na slici 3.4 nalaze nule.

8	8	4	2	5	0
8	8	1	2	0	0
6	5	3	2	0	6
4	3	2	2	0	7
1	0	4	5	4	4
1	0	4	0	6	6

Slika 3.4: *Slika na koju je potrebno primijeniti filter iz prvog zadatka*

Zadatak 2. U Pythonu učitajte sliku u razinama sive boje. Konstruirajte filter koji će sliku pomaknuti za 100 piksela desno i 30 piksela gore te ga primijenite na učitanu sliku.

4 Filteri

U prethodnoj laboratorijskoj vježbi vidjeli smo da su filteri samo matrice koje se korelacijom primjenjuju na digitalnu sliku. U ovisnosti o tome koji se brojevi nalaze u filteru i kolike su njegove dimenzije, učinak filtera na sliku će biti drugačiji. U digitalnoj obradi i analizi slike postoje već definirani filteri koji se koriste za zamućivanje slike, detekciju rubova na slici, detekciju raznoraznih značajki, i sl. U ovoj laboratorijskoj vježbi vidjeti ćemo koji se filteri najčešće koriste i kakvi su njihovi učinci na digitalnu sliku.

4.1 Zamućivanje slike

Zamućivanje slike (engl. *image smoothing, image blurring*) se u obradi i analizi digitalne slike najčešće koristi za uklanjanje šuma sa slike. Na primjer, ako skenirate neke stare slike u računalo one će na sebi najčešće imati čestice prašine ili manja oštećenja. Te se nesavršenosti mogu ukloniti zamućivanjem slike. Nadalje, ako na sliku želite primijeniti neki složeniji algoritam za obradu i analizu digitalne slike, često je dobro sliku najprije zamutiti filterom za zamućivanje malih dimenzija koji će ukloniti sitni šum na slici i time često poboljšati rezultate složenijeg algoritma.

Budući da filteri za zamućivanje slike ublažavaju nagle promjene na slici (tj. ublažavaju visoke frekvencije), oni spadaju u tzv. **niskopropusne filtere** (engl. *low-pass filters*).

4.1.1 Gaussov filter zamućivanja

Gaussov filter zamućivanja ili ujednačavanja (engl. *Gaussian filter*, *Gaussian smoothing filter*, *Gaussian averaging filter*) jedan je od najčešće korištenih filtera u digitalnoj obradi i analizi slike. Sastoji se od matrice ispunjene jedinicama koja se množi sa razlomkom u čijem se nazivniku nalazi ukupan broj jedinica u matrici.

Ono što Gaussov filter zapravo radi je računa aritmetičku sredinu piksela u određenom susjedstvu.

Gaussov filter dimenzija 3x3:

$$\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.1}$$

Gaussov filter dimenzija 5x5:

$$\frac{1}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.2}$$

Gaussov filter dimenzija 7x7:

$$\frac{1}{49} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

Zadatak 1. U Pythonu učitajte sliku u boji i na nju primijenite Gaussov filter zamućivanja dimenzija 9x9 piksela. Što bi se dogodilo sa slikom da smo na nju primijenili Gaussov filter manjih dimenzija?

4.1.2 Medijan filter

Vidjeli smo da Gaussov filter zamućivanja računa aritmetičku sredinu piksela u određenom susjedstvu, no to nije jedina statistička sredina koja postoji. Još jedna statistička sredina se često koristi u obradi i analizi digitalne slike - medijan. Medijan određenog niza brojeva računa se na način da se ti brojevi poredaju od najmanjeg prema najvećem, te se kao rezultat uzme onaj koji se nalazi u sredini niza. Ako niz ima paran broj elemenata, onda se računa aritmetička sredina središnja dva elementa.

Zadatak 1. Izračunajte medijan slijedećeg niza: [13, 25, 60, 1, 0].

Zadatak 2. U Pythonu učitajte sliku u boji i na nju primijenite medijan filter dimenzija 3x3 piksela. Nemojte koristiti već postojeće funkcije u Pythonu za računanje medijana, već napišite sami svoju funkciju. Napomena: medijan filter se posebno primjenjuje na svaki kanal RGB slike.

4.2 Izoštravanje slike

Izoštravanje slike (engl. *image sharpening*) se u obradi i analizi digitalne slike najčešće koristi kada je na slici potrebno istaknuti sitne detalje koji su inače slabo vidljivi.

Neki od najpoznatijih filtera za izoštravanje digitalnih slika su:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 5 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.4}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 16 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.5}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.6}$$

Budući da filteri za izoštravanje slike naglašavaju nagle promjene na slici (tj. ističu ili pojačavaju visoke frekvencije), oni spadaju u tzv. **visokopropusne filtere** (engl. *high-pass filters*).

Zadatak 3. U Pythonu učitajte sliku u boji i na nju primijenite proizvoljni filter izoštravanja.

4.3 Detekcija rubova na slici

Detekcija rubova (engl. *edge detection*) se u obradi i analizi digitalne slike najčešće koristi kao prvi korak u složenijim programima za analizu slike. Na primjer, ako se na digitalnoj slici žele pronaći krvne žile u svrhu analize medicinskih slika, bilo bi dobro detektirati rubove na slici da znamo gdje otprilike tražiti krvne žile.

4.3.1 Laplaceov filter

Laplaceov filter je jedan od najpoznatijih filtera za detekciju rubova na slici, i spada u visokopropusne filtere.

Laplaceov filter dimenzija 3x3:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.7}$$

Zadatak 4. U Pythonu učitajte sliku u boji i na nju primijenite Laplaceov filter za detekciju rubova.

4.3.2 Magnituda gradijenta

Magnituda gradijenta se odnosi na detekciju rubova na slici u dva smjera - horizontalnom i vertikalnom. Ova se detekcija obavlja pomoću dva Sobelova filtera:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.8}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{4.9}$$

U literaturi se digitalna slika na koju je primjenjen prvi Sobelov filter naziva G_x, a ona na koju je primijenjen drugi Sobelov filter G_y.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{4.10}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{4.11}$$

U formulama 4.10 i 4.11 A označava digitalnu sliku, a $*$ operaciju korelacije. Magnituda gradijenta G se zatim dobije pomoću formule 4.12, ili aproksimacijom pomoću formule 4.13.

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{4.12}$$

$$G = |G_x + G_y| \tag{4.13}$$

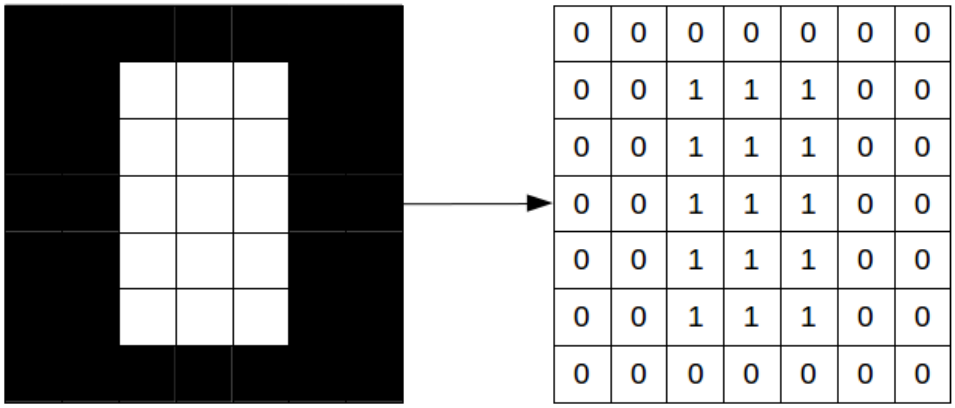
Zadatak 5. U Pythonu učitajte sliku u boji i za nju izračunajte i prikažite magnitudu gradijenta.

5 Morfološke operacije na digitalnoj slici

Morfološke operacije se često koriste u obradi i analizi digitalne slike. U ovoj laboratorijskoj vježbi naučiti ćemo koje su od morfoloških operacija najpoznatije i za što se točno koriste.

5.1 Binarne slike

Binarne slike (engl. *binary images*) su slike koje se sastoje samo od nula i jedinica. Često se koriste u digitalnoj obradi i analizi slike prilikom upotrebe morfoloških operacija. Slika 5.1 prikazuje primjer binarne slike gdje jedinica označava bijelu boju, a nula crnu boju.



Slika 5.1: *Primjer binarne slike*

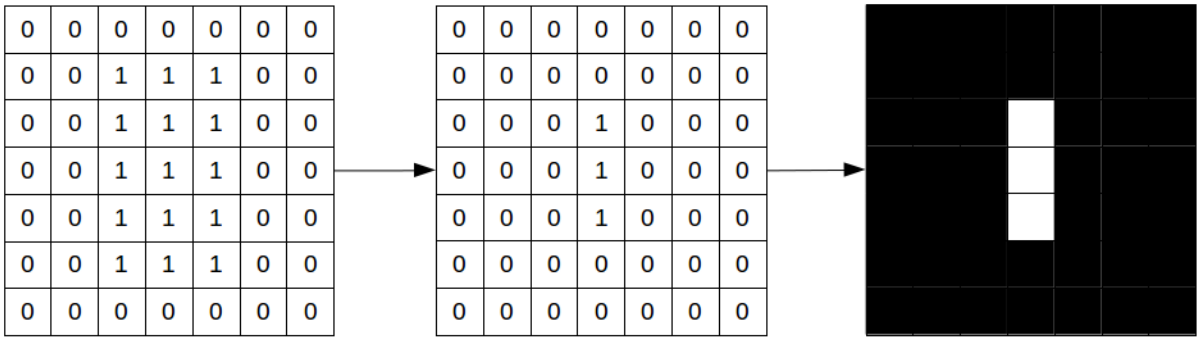
5.2 Erozija i dilatacija

Erozija (engl. *erosion*) i dilatacija (engl. *dilation*) su morfološke operacije koje se često koriste u obradi i analizi digitalne slike. Kod ovih operacija na slike se primjenjuju određeni filteri, a filter koji će se koristiti u ovoj laboratorijskoj vježbi prikazan je na slici 5.2).

1	1	1
1	1	1
1	1	1

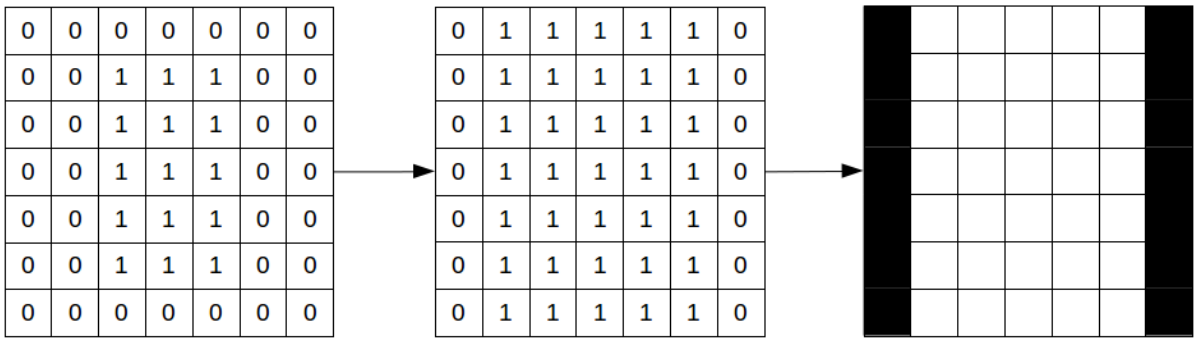
Slika 5.2: *Primjer filtera za eroziju i dilataciju*

Erozija se koristi za odvajanje spojenih regija na slikama koje ne bi trebale biti spojene. Na primjer, kod detekcije proteina na mikrobiološkim slikama ponekad se dogodi da se dva proteina dodiruju i bez primjene erozije računalni program bi ih vjerojatno pogrešno detektirao kao jedan protein, a ne dva. Slika 5.3 prikazuje primjer erozije na digitalnoj slici.



Slika 5.3: *Primjer erozije binarne slike*

Dilatacija se koristi za spajanje odvojenih regija na slikama koje ne bi trebale biti odvojene. Na primjer, kod detekcije krvnih žila na digitalnim slikama retine oka ponekad se dogodi da se ne detektira manji dio neke krvne žile. Primjenom dilatacije taj se nedetektirani dio može spojiti sa dijelom koji je detektiran. Slika 5.4 prikazuje primjer dilatacije na digitalnoj slici.



Slika 5.4: *Primjer dilatacije binarne slike*

Pojednostavljeno, erozija se koristi za smanjivanje regija na slici a dilatacija za proširivanje.

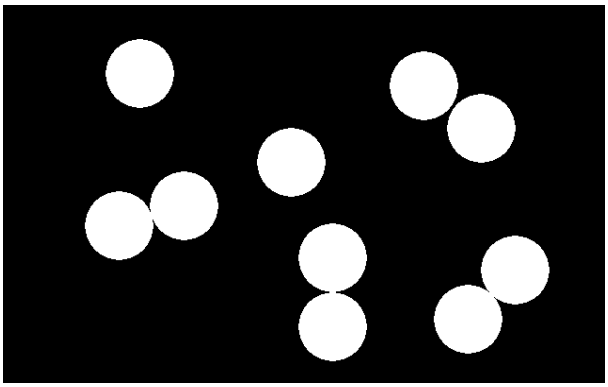
Primjer erozije i dilatacije u Pythonu:

```
import cv2
import numpy as np

slika_siva = cv2.imread('slika.jpg', cv2.IMREAD_GRAYSCALE)

(prag, slika_binarna) = cv2.threshold(slika_siva, 127, 255,
cv2.THRESH_BINARY)

moj_filter = np.ones((5,5), np.uint8)

slika_erozija = cv2.erode(slika_binarna, moj_filter, iterations = 1)
slika_dilatacija = cv2.dilate(slika_binarna, moj_filter, iterations
= 1)
```

```
cv2.imshow('Siva slika', slika_siva)
cv2.imshow('Binarna slika', slika_binarna)
cv2.imshow('Erozija', slika_erozija)
cv2.imshow('Dilatacija', slika_dilatacija)
cv2.waitKey(0)
```

Zadatak 1. U Pythonu učitajte sliku prikazanu na slici 5.5.

- Učitanu sliku konvertirajte u binarnu sliku.
- Pomoću funkcije cv2.SimpleBlobDetector() odredite koliko se pojedinačnih regija nalazi na učitanoj slici.
- Na učitanu sliku primijenite operaciju erozije i odredite koliko se regija nalazi na novoj slici. Pokušajte koristiti različite veličine filtera za operaciju erozije da vidite koja će vam najviše odgovarati.



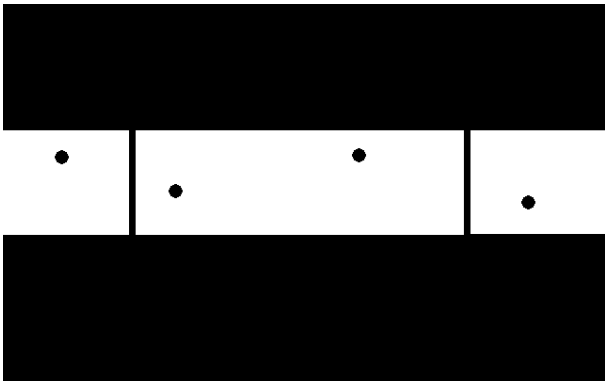
Slika 5.5: *Slika vezana za zadatak 1*

Zadatak 2. U Pythonu učitajte sliku prikazanu na slici 5.6.

- Učitanu sliku konvertirajte u binarnu sliku.
- Pomoću funkcije cv2.SimpleBlobDetector() odredite koliko se pojedinačnih regija nalazi na učitanoj slici.
- Na učitanu sliku primijenite operaciju dilatacije i odredite koliko se regija nalazi na novoj slici. Pokušajte koristiti različite veličine filtera za operaciju dilatacije da vidite koja će vam najviše odgovarati.

5.3 Morfološko otvaranje i zatvaranje

Morfološko otvaranje (engl. *morphological opening*) uključuje eroziju koju slijedi dilatacija, a morfološko zatvaranje (engl. *morphological closing*) uključuje dilataciju koju slijedi erozija.



Slika 5.6: *Slika vezana za zadatak 2*

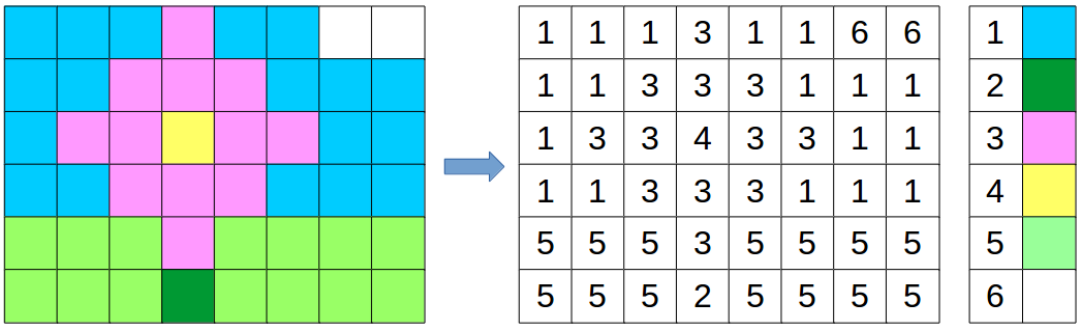
Zadatak 3. U Pythonu učitajte sliku prikazanu na slici 5.5 te na nju primijenite morfološku operaciju otvaranja. Po čemu se dobiveni rezultati razlikuju od onih dobivenih u 1. zadatku?

Zadatak 4. U Pythonu učitajte sliku prikazanu na slici 5.6 te na nju primijenite morfološku operaciju zatvaranja. Po čemu se dobiveni rezultati razlikuju od onih dobivenih u 2. zadatku?

6 Segmentacija digitalne slike

Segmentacija slike se može definirati kao proces u kojem se slika dijeli na homogene segmente (cjeline ili regije). Ovi se homogeni segmenti ponekad nazivaju i superpikseli. U okviru ove laboratorijske vježbe mi ćemo ove homogene segmente nazivati klase, s naglaskom na tome da računalo ne zna što točno koja klasa predstavlja (npr. nebo ili more). Ako se piksele želi ciljano svrstati u određenu klasu koju računalo prepoznaje, potrebno je izvršiti klasifikaciju slike, a ne samo segmentaciju.

Dakle, segmentacija digitalne slike podrazumijeva svrstavanje njezinih piksela u određeni broj klasa. Na primjer, ako na slici imamo sunce i nebo, žuti pikseli mogli bi se svrstati u prvu klasu, a plavi u drugu. Potrebno je naglasiti da računalo ne zna što koja klasa predstavlja, već piksele raspodjeljuje na temelju nekih značajki (npr. boje ili teksture). Primjer segmentacije slike u 6 klasa dan je na slici 6.1. Na toj slici brojevi od 1 do 6 predstavljaju oznake klasa, tzv. labele (engl. *labels*).



Slika 6.1: *Primjer segmentacije digitalne slike*

Zadatak 1. U Pythonu učitajte sliku u razinama sive boje i napišite funkciju koja će je segmentirati na slijedeći način:

- klasa 1 - svi pikseli u rangu $[0, 85]$
- klasa 2 - svi pikseli u rangu $[86, 171]$
- klasa 3 - svi pikseli u rangu $[172, 255]$

Postoji jako puno različitih algoritama za automatsku segmentaciju digitalne slike, a najpoznatiji su algoritam k-sredina i algoritam razvodnjavanja.

6.1 Algoritam *k*-sredina

Algoritam *k*-sredina (engl. *k-means algorithm*) je jedan od najjednostavnijih i najčešće korištenih algoritama za segmentaciju podataka. Njime se pikseli slike dijele na k regija, a k se unaprijed zadaje.

Algoritam *k*-sredina je prvi put predložio Lloyd u [3], pa se u računarstvu još naziva i Lloydov algoritam. Riječ je o iterativnom algoritmu koji skup od n ulaznih podataka dijeli na k regija u konačnom broju iteracija. Algoritam slučajnim odabirom odabire k početnih središta regija, uspoređuje koordinate svakog ulaznog podatka sa tim središtima, te podatak dodjeljuje regiji čijem je središtu najbliži. Podaci u i-toj regiji su sličniji jedni drugima nego podacima u bilo kojoj drugoj regiji. Sličnost između različitih piksela na slici je najčešće definirana kao Euklidska udaljenost između njihovih koordinata.

Ako se algoritmu *k*-sredina kao ulaz zada slika u razinama sive boje, onda pikseli te slike imaju samo jednu koordinatu - razinu sive boje. Udaljenost između 1-dimenzionalnih točaka $X = (x)$ i $Y = (y)$ dana je kao $d(X,Y) = \sqrt{(x-y)^2} = |x-y|$.

Ako se algoritmu *k*-sredina kao ulaz zada slika u boji, onda pikseli te slike imaju tri koordinate – po jednu razinu sive boje za svaki kanal od kojeg se sastoje. Udaljenost između 3-dimenzionalnih točaka $X = (x_1,x_2,x_3)$ i $Y = (y_1,y_2,y_3)$ dana je kao $d(X,Y) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+(x_3-y_3)^2}$.

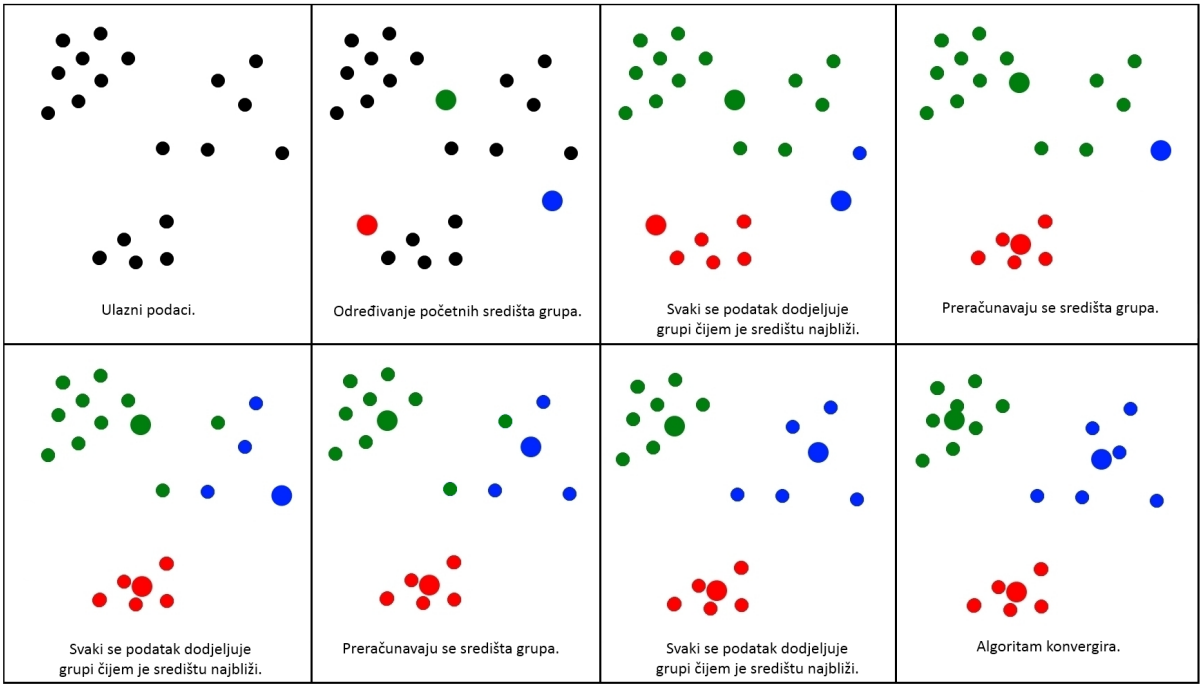
Sličnost između točaka se povećava sa smanjivanjem udaljenosti između njih.

Pseudokod algoritma *k*-sredina:

1. korisnik zadaje broj regija za segmentaciju k;
2. algoritam slučajnim odabirom odabire k početnih središta regija koji se nalaze na maksimalnoj međusobnoj udaljenosti;
3. računa se udaljenost svakog podatka (npr. točke ili piksela) od svih mogućih središta regija i podatak se dodjeljuje regiji čijem je središtu najbliži;
4. središta regija se preračunavaju na način da se smještaju u središta podataka koji su im u prethodnoj iteraciji dodijeljeni;
5. algoritam se ponavlja sve dok ne konvergira ili dok se ne postigne tražena točnost.

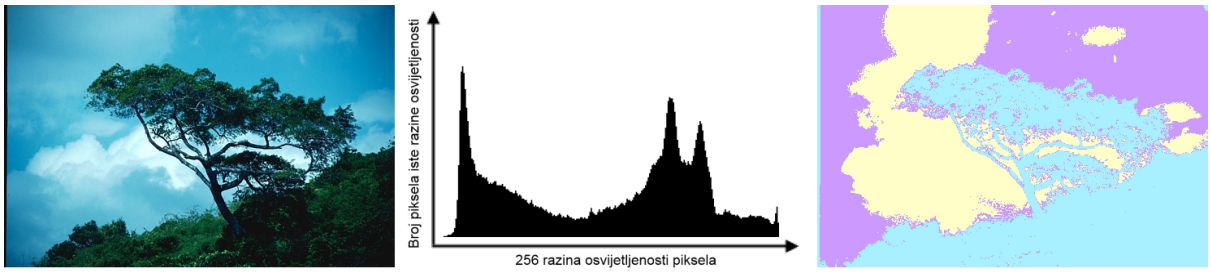
Shematski prikaz algoritma *k*-sredina dan je na slici 6.2.

Početne koordinate središta regija se odabiru slučajnim odabirom, a u svakoj sljedećoj iteraciji se računaju kao aritmetička sredina koordinata podataka koji su im u prethodnoj iteraciji dodijeljeni. Na primjer, ako se regiji A u iteraciji i dodijele 3-dimenzionalni podaci $X = (x_1,x_2,x_3)$ i $Y = (y_1,y_2,y_3)$, onda će koordinate središta grupe A u iteraciji $i+1$ biti $C_A = (c_1,c_2,c_3)$, gdje je $c_1 = \frac{x_1+y_1}{2}$, $c_2 = \frac{x_2+y_2}{2}$ i $c_3 = \frac{x_3+y_3}{2}$.

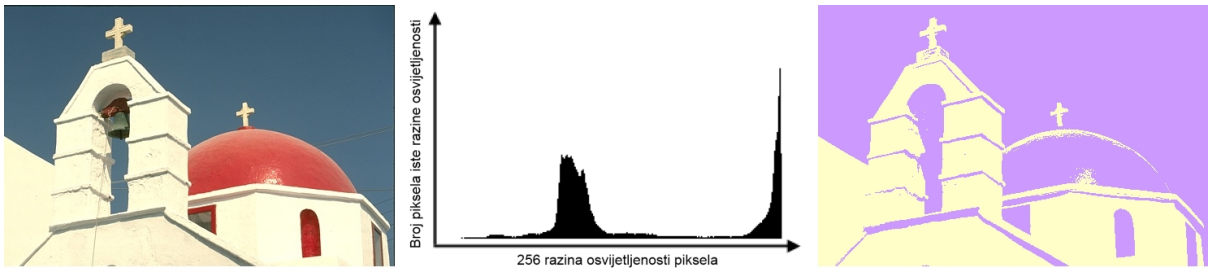


Slika 6.2: *Grafički prikaz rada algoritma k-sredina. U danom primjeru korištene su tri grupe za segmentaciju. Manje točke predstavljaju ulazne podatke, a veće točke predstavljaju središta grupa na koje se ti podaci dijele*

Primjeri segmentacije slike pomoću algoritma *k*-sredina dani su na slikama 6.3 i 6.4. Ulazne slike sa slika 6.3 i 6.4 preuzete su iz Berkeley Image Dataset-a [4].



Slika 6.3: *Primjer segmentacije slike pomoću algoritma k-sredina*



Slika 6.4: *Primjer segmentacije slike pomoću algoritma k-sredina*

Zadatak 2. U Pythonu učitajte sliku u boji i na nju primijenite algoritam k-sredina (ne trebate ga sami implementirati, možete koristiti postojeću funkciju). Kako se segmentirana slika i brzina algoritma mijenjaju kada se mijenja k?

6.2 Algoritam razvodnjavanja

Algoritam razvodnjavanja (engl. *watershed algorithm*) često se koristi prilikom segmentacije digitalnih slika. Osnovna ideja ovog algoritma je da se na sliku gleda kao na reljef - pikseli sa visokim intenzitetima su planine, a oni sa niskim intenzitetima su doline. U dolinama se stvaraju jezera različitih boja koja se šire kako se razina vode na reljefu podiže. Kada se dvije različite boje vode dotaknu, tu se postavlja granica ili brana. Granice predstavljaju rubove različitih regija segmentirane slike. Konačna segmentirana slika se dobije kada se čitav reljef nalazi pod vodom. Novije verzije algoritma razvodnjavanja uključuju različite preinake i poboljšanja, te su kao takve implementirane u različitim bibliotekama za obradu i analizu digitalnih slika.

Zadatak 3. U Pythonu učitajte sliku u boji i na nju primijenite algoritam razvodnjavanja (ne trebate ga sami implementirati, možete koristiti postojeću funkciju). Usporedite dobivenu segmentiranu sliku sa rezultatima iz prethodnog zadatka.

7 Detekcija objekata na slici

Automatska detekcija objekata na digitalnoj slici jedna je od najčešće korištenih metoda u obradi i analizi digitalne slike.

Primjeri korištenja automatske detekcije objekata na slici:

- detekcija registarskih tablica na automobilima,
- detekcija dima i vatre sa slika nadzornih kamera postavljenih u svrhu zaštitu i sprječavanja požara,
- detekcija kotača prilikom prolaska vozila kroz naplatne kućice na autocestama u svrhu klasificiranja broja i vrsta automobila koji tu prođu u određenom vremenskom periodu,
- detekcija znakova bolesti na medicinskim slikama,
- detekcija sumnjivih objekata na aerodromima,
- detekcija lica u većini današnjih fotoaparata i kamera.

Detekcija objekata na slici sastoji se od 3 koraka:

1. prikupljanje ulaznih podataka te ručno označavanje traženog objekta na njima, te podjela prikupljenih slika na skup za treniranje algoritma i skup za testiranje algoritma,
2. odabir i određivanje značajki objekta kojega je potrebno klasificirati,
3. testiranje odabranih značajki na skupu slika za testiranje klasifikacijskog algoritma.

7.1 Ulazni podaci

Ako želimo konstruirati računalni program koji će automatski detektirati određenu vrstu objekta na slici najprije moramo prikupiti i označiti što veći broj slika koje prikazuju taj objekt. Ovo označavanje se često naziva i laberiranje (engl. *labeling*) ili tzv. *ground truth (GT)*. Primjer ručnog označavanja različitih klasa na slici prirodnog krajolika dan je na slici 7.1 koja je sastavni dio FESB MLID (*Mediterranean Landscape Image Dataset*) [5] baze slika. FESB MLID baza slika je podijeljena na 200 slika za treniranje algoritma i 200 slika za testiranje algoritma.

Iako je FESB MLID baza slika označena sa 12 različitih klasa, mi ćemo se u ovoj laboratorijskoj vježbi usredotočiti samo na jednu - vegetaciju. U FESB MLID bazi slika vegetacija je na GT slikama označena intenzitetima piksela 184 (udaljena vegetacija) i 207 (bliska vegetacija). Mi nećemo raditi razliku između te dvije klase, nego ćemo samo pretpostaviti da svi pikseli s intenzitetima 184 ili 207 na GT slici predstavljaju vegetaciju.

Slika 7.1: *Primjer označavanja slike - svaka boja na desnoj slici odgovara određenoj klasi*

Zadatak 1. Sa e-Learning portala preuzmite skraćenu verziju FESB MLID baze slika. (Potpunu FESB MLID bazu slika možete pronaći na http://wildfire.fesb.hr/index.php.)

Zadatak 2. U Pythonu konstruirajte funkciju koja će učitavati i prikazivati slike iz skraćene FESB MLID baze slika. Funkciju možete realizirati na način da čita sve slikovne datoteke iz određenog direktorija. Pazite samo da morate istovremeno učitati i originalnu sliku i GT sliku jer će vam obje istovremeno trebati za treniranje algoritma.

7.2 Odabir značajki

Da bismo mogli prepoznati vegetaciju na slici prirodnog krajolika, najprije trebamo odrediti koje značajke slike izdvajaju vegetaciju od ostalih klasa. Prva pomisao je najčešće ta da trebamo izdvojiti piksele zelene boje, no u tome slučaju algoritam ne bi detektirao raslinje koje mijenja boju sa promjenom godišnjih doba. Potrebno je, dakle, istrenirati ili naučiti algoritam da na slici traži određeni rang boja, a to učenje se izvršava na skupu slika za treniranje algoritma. Sve što vaš algoritam nauči o objektu kojeg tražite nazivamo **bazom znanja**. Baze znanja su osnova ekspertnih sustava koji se često koriste u praksi.

Zadatak 3. Napišite funkciju pomoću koje ćete odrediti koje sve boje poprimaju pikseli vegetacije na slikama koje ste učitali u prethodnom zadatku.

7.3 Testiranje algoritma

Prilikom testiranja algoritma određuje se njegova uspješnost da detektira traženi objekt na novim slikama, tj. na onim slikama na kojima nije treniran.

Zadatak 4. Napišite novi program pomoću kojega ćete učitati slike za testiranje algoritma. Izdvojite piksele koji odgovaraju rangu boja kojega ste dobili u prethodnom zadatku. Detektiraju li se samo pikseli koji pripadaju vegetaciji?

Za određivanje točnosti detekcijskog algoritma mogu se koristiti različite formule, no jedna od onih koje se najčešće koriste je slijedeća:

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{7.1}$$

gdje je, u našem slučaju:

- TP (engl. *True Positives*) - broj piksela vegetacije koje je vaš algoritam ispravno prepoznao kao vegetaciju,
- TN (engl. *True Negatives*) - broj piksela pozadine (tj. onih piksela koji ne predstavljaju vegetaciju) koje je vaš algoritam ispravno prepoznao kao pozadinu,
- FP (engl. *False Positives*) - broj piksela pozadine koje je vaš algoritam prepoznao kao vegetaciju,
- FN (engl. *False Negatives*) - broj piksela vegetacije koje je vaš algoritam prepoznao kao pozadinu.

Zadatak 5. Odredite točnost konstruiranog algoritma za detekciju vegetacije.

7.4 Poboljšavanje detekcijskog algoritma

Zadatak 6. Pokušajte poboljšati točnost detekcije objekta kojega tražite koristeći se nekom od metoda iz prethodnih laboratorijskih vježbi. Primjeri mogućih metoda:

- korištenje različitih prostora boja,

- zamućivanje ili izoštravanje slike,
- segmentacija slike na veći broj klasa (cilj je dobiti superpiksele).

Literatura

[1] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE TPAMI*, 33:898–916, May 2011.

[2] Satya Mallick. Why does OpenCV use BGR color format? `https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/`, September 2015. Accessed: 2018-07-18.

[3] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, March 1982.

[4] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proc. 8th Int'l Conf. Computer Vision (2)*, pages 416–423, July 2001.

[5] M. Braović, D. Stipaničev, and D. Krstinić. Cogent confabulation based expert system for segmentation and classification of natural landscape images. *Advances in Electrical and Computer Engineering*, 17:85–94, 2017.