

# domaci1\_18\_018

November 15, 2021

## 1 Domaći zadatak iz predmeta Digitalna obrada signala (13E044DOS)

### 1.0.1 Danica Bandović 0018/2018

#### 1.1 Učitavanje svih potrebnih modula i biblioteka

```
[1]: from pylab import *

import os
import imageio
import time
import skimage
from skimage import color
from skimage import exposure
from skimage import filters
from skimage import io
from skimage import img_as_float
from skimage.util import img_as_ubyte

from ipywidgets import interact, interactive, fixed, interact_manual

import numpy as np

from scipy import ndimage
```

#### 1.2 1.Zadatak

Realizovana je funkcija dress\_queen\_still kojom se menja dezen kaputa i haljine engleske kraljice na slici queen\_dress.jpg. Da bismo promenili dezen haljine, potrebno je izdvojiti deo slike koji pripada haljini, odnosno napraviti masku, nakon čega bismo promenili vrednosti intenziteta onih piksela koji pripadaju maski. Ideja je da se segmentacijom boje napravi maska, pošto je haljina zelene boje. Nakon toga je izvršeno filtriranje kako bismo dobili bolju masku.

Za segmentaciju zelene boje u HSV kolor sistemu, korišćena je H (Hue) komponenta, jer ona govori o nijansi boje na slici. Vrednosti H komponente koje su izdvajile zelenu boju na slici nalaze se izmedju 0.3 i 0.5. Razlog je što nijanse zelene zauzimaju otprilike trećinu ukupnog opsega nijansi, izmedju crvene i plave. Ako bi se uzele neke druge vrednosti za granice opsega, na primer ako bi gornja granica bila 0.7, osim zelene izdvojili bismo i nijanse plave.Ukoliko bismo uzeli donju granicu

malu, na primer 0.1 ili 0, osim zelene bismo izdvojili i crvenu boju. Pošto je cilj izdvojiti samo zelenu boju, uzet je gore pomenut opseg vrednosti za H. Nije uzet ni ceo opseg zelene boje kako bi se izdvojila haljina, a u masku što manje ušli delovi iz pozadine koji su zelene boje, ali ne iste nijanse kao haljina kraljice. Takodje, kako bi se izdvojila haljina od ostalih zelenih delova na slici, posmatrana je i S (Saturation) komponenta. Pošto su delovi u pozadini manje zasićeni od kraljičine haljine, prag zasićenosti iznad kog smo uzeli masku stavljen je na 0.3. Dobijena maska predstavlja deo slike na kome je haljina, ali je pored toga obuhvatila i još neke sitne detalje u pozadini. Urađeno je zamućivanje dobijene maske kako bi se ti detalji koji ne predstavljaju haljinu i dosta su manji od nje, izgubili. Izvršeno je filtriranje usrednjavanjem. Za dimenziju matrice kojom se filtrira maska uzeta je vrednost 7, jer previše mala vrednost dimenzija, kao što je već 5 na primer, nije uspešno utopila detalje u pozadinu. Sa druge strane, box filter dimenzija iznad 10x10 je previše zamućivao bitne delove maske. Nakon što je maska filtrirana, izvršeno je poređenje sa pragom, s obzirom da su se ivice haljine zamutile filtriranjem. Poredenjem sa pragom poboljšali smo ivice, i time dobili krajnju masku koja je korišćenja za blednovanje slike kraljice i slike dezena. Za filtriranje se može koristiti i Gausov filter, ali je u ovom slučaju korišćeno filtriranje usrednjavanjem jer daje zadovoljavajući rezultat.

Kod korišćene funkcije nalazi se u nastavku:

```
[2]: def dress_queen_still(designName):
    """
    Returns queen_dress image in which the design of the dress has been changed

    Parameters:
    designName(string): name of the image that represent the design

    Returns:
    dressDesigned(numpy.ndarray): image in RGB format

    """
    #creating paths to images
    dirPath = '../sekvence//'
    #designPath = os.path.join(dirPath, designName)
    queenPath = os.path.join(dirPath, 'queen_dress.jpg')

    #loading images
    queenImg = io.imread(queenPath)
    designImg = io.imread(designName)

    #convert RGB to HSV
    queenHSV = color.rgb2hsv(queenImg)
    designHSV = color.rgb2hsv(designImg)

    #green color segmentation using H component, and S component for saturation
    imgOut = zeros(shape(queenHSV[:, :, 0]))
    mask = (queenHSV[:, :, 0] >= 0.3) & (queenHSV[:, :, 0] <= 0.5) & (queenHSV[:, :, 1] >= 0.3)
    imgOut[mask] = 255
```

```

#plotting mask before filtering
fig, ax = plt.subplots(ncols=3, figsize=(16,8), dpi=80)
ax[0].imshow(queenHSV[:, :, 0], vmin=0, vmax=1, cmap='jet'); ax[0].
→set_title('H'); ax[0].axis('off')
ax[2].imshow(imgOut, cmap='gray'); ax[2].set_title('Maska pre filtriranja'); u
→ax[2].axis('off')
ax[1].imshow(queenHSV[:, :, 1], vmin = 0, vmax = 1, cmap = 'jet'); ax[1].
→set_title('S'); ax[1].axis('off');
plt.show()

#blurring details and improving edges
imgOut[~mask] = 255
imgOut[mask] = 0
#mask after filtering
mask = filterMaskNorm==0
imgOut[mask] = 255
imgOut[~mask] = 0

#replacement of brightness and color shade in hsv color system
queenHSV[:, :, 0][mask] = designHSV[:, :, 0][mask]
queenHSV[:, :, 1][mask] = designHSV[:, :, 1][mask]
#converting HSV to RGB format
dressDesigned = color.hsv2rgb(queenHSV)

#resulting picture

fig, ax = subplots(nrows=1, ncols=2, figsize=(12,6), dpi=80);
ax[0].imshow(queenImg);
ax[0].set_title('Originalna slika kraljice');
ax[0].axis('off');
ax[1].imshow(dressDesigned);
ax[1].set_title('Izlazna slika');
ax[1].axis('off');

plt.tight_layout();
plt.show();

```

```
return dressDesigned
```

```
[3]: designed = dress_queen_still('haljina1.jpg')
#figure(figsize=(20,10))
#imshow(design)
```



Originalna slika kraljice



Izlazna slika



Drugi deo prvog zadatka jeste da se promeni dizajn kraljičinog kaputa u videu koji je dat. Koristi se ista ideja kao u prethodnom delu zadatka, s tim što se za određivanje parametara koji predstavljaju prag za H i S komponentu koriste frejmovi iz različitih scena pošto imamo relativno statične scene,

na kojima se pojavljuje kraljica, i one na kojima se ne pojavljuje a koji imaju ili nemaju zelenu boju. Uzet je po jedan frejm iz svake scene, i posmatrane su njegove statičke karakteristike, na osnovu čega je određeno koji opseg nijanse zelene je potrebno izbaciti, a koji ostaviti, tako da algoritam za svaki frejm detektuje zelenu boju na kraljičinom kaputu, ali da što je moguće više odbaci zelene boje koje ne predstavljaju kaput. Ovakva analiza odradlena je i za H i za S komponentu, nakon čega je uzeto da H treba da se nalazi između 0.2 i 0.5, a S treba da bude veće od 0.7, pošto je boja kaputa veoma zasićena. Gausovim filtrom izbačeni su detalji iz maske koji ne pripadaju haljini, a ivice poboljšane poređenjem sa pragom. Dobijeni video nalazi se u fajlu ‘queen\_coat\_out.mp4’. Kod funkcije nalazi se ispod:

```
[8]: def dress_queen_video(designName):
    """
    Create video 'queen_coat_out.mp4' in which the design of the queen coat has
    ↪been changed

    Parameters:
    designName(string): name of the image that represent the design

    """

    #loading video and coat pattern
    dirPath = '../sekvence//'
    #designPath = os.path.join(dirPath,designName)
    filePath = os.path.join(dirPath,'queen_coat.mp4')
    vid = imageio.get_reader(filePath, 'ffmpeg')
    design = io.imread(designName)
    designHSV = color.rgb2hsv(design)
    data = vid.get_meta_data()

    P = data['size'][0]*data['size'][1]

    video_out = imageio.get_writer('queen_coat_out.mp4', format='FFMPEG', mode=
    ↪= 'I', fps = 30, codec = 'h264')

    #print('Pocelo')
    #start = time.time()

    for cur_frame in vid:

        imgHSV = color.rgb2hsv(cur_frame)

        mask =(imgHSV[:, :, 0]<=0.50) & (imgHSV[:, :, 0]>=0.20) & (imgHSV[:, :, 1]>=0.
    ↪70)
        mask = filters.gaussian(mask, sigma=3)>0.3

        proc = 100*(np.count_nonzero(mask))/P
        if proc>4:
```

```

    imgHSV[:, :, 0:2][mask] = designHSV[:, :, 0:2][mask]
    #cur_frame[:, :, 0:3][mask] = design[:, :, 0:3][mask]
    im1=color.hsv2rgb(imgHSV);
else:
    im1 = cur_frame

video_out.append_data(img_as_ubyte(im1));

#end = time.time()
#ex_time = (end-start)

#print('Gotovo. Vreme izvrsavanja je ', + round(ex_time,3))
video_out.close()

```

[9]: `dress_queen_video('dizajn1.jpg')`

Na vreme izvršavanja funkcije `dress_queen_video` najviše utiče to što se za svaki frejm vrši konverzija u HSV kolor sistem i obratno, iz HSV u RGB. Vreme izvršavanja je između 6 i 7 minuta. Ukoliko se vrednosti piksela menjaju direktno u RGB sistemu, dok HSV služi samo za segmentaciju boje, vreme izvršavanja se smanjuje na 3 do 4 minuta.

Takođe, u kreiranom videu se primećuje da postoji nekoliko frejmova na kojima se nalazi kraljica, ali algoritam nije izdvojio njen kaput kao masku. U pitanju je prelazak između dve statične svene, i problem kod ovih frejmova je što su previše svetli. Verovatno bi se moglo postići dobro izdvajanje maske ako se izvrši neka prethodna obrada ovakvih frejmova, gde bi se prvo ceo frejm zatamneo, a potom bila pokušana da se pronade maska. Ovo je samo ideja, ali ne mora da znači da bi ovakav algoritam dao dobar rezultat.

### 1.3 2.Zadatak

U drugom zadatku traženo je da se izoštiri slika u boji. Ideja je da se prebaci slika iz RGB u YUV kolor sistem, pošto nam Y komponenta (siva slika) daje najviše informacija o detaljima, tako da ćemo na ovoj komponenti primeniti izoštrevanje. Prvo je kreirana maska za neoštire delove (unsharped masking). Pri vraćanju detalja, množimo ih sa 2 kako bismo pojačali intenzitet detalja. Mogu se pomnožiti i nekim drugim brojem u zavisnosti koliko želimo da istaknemo detalje, ali treba voditi računa o tome da ako je broj kojim se množi previše mali, slika neće biti dovoljno izoštrena, a ukoliko je ovaj broj veliki može doći do preteranog izoštrevanja, te dobijena slika može izgledati neprirodno. Pre nego što vratimo sliku iz YUV u RGB kolor sistem, potrebno je ostaviti U i V komponente iste, a Y komponentu zameniti sa dobijenom izostrenom slikom. Dobijena slika u RGB formatu sačuvana je u fajlu ‘miner\_sharp.jpg’.

Kod korišćen za ovaj zadatak:

[10]: `#loading image`  
`imgRGB = img_as_float(imread('../sekvence/miner.jpg'))`  
`#transfer from RGB to YUV and take Y component`  
`imgYUV = color.rgb2yuv(imgRGB)`  
`img = imgYUV[:, :, 0]`

```

#unsharped masking
lowpassMask = np.ones(shape=(3,3))/9
imgBlurred = ndimage.correlate(img, lowpassMask)
imgDetails = img - imgBlurred

#returning details multiplied with factor of 2
imgSharp = img + 2*imgDetails

imgYUVout = np.zeros(shape=imgYUV))
imgYUVout[:, :, 1:3] = imgYUV[:, :, 1:3]
imgYUVout[:, :, 0] = imgSharp

#transfer to RGB
imgRGBout = color.yuv2rgb(imgYUVout)
imgRGBout[imgRGBout<0] = 0
imgRGBout[imgRGBout>1] = 1

#plotting input and output image
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(22,10), dpi=60)
ax = axes.ravel()

ax[0].imshow(imgRGB, vmin=0, vmax=1); ax[0].set_axis_off(); ax[0].
    set_title('Ulazna slika', fontsize=16)
ax[1].imshow(imgRGBout, vmin=0, vmax=1); ax[1].set_axis_off(); ax[1].
    set_title('Izoštrena slika', fontsize=16)

plt.tight_layout()
plt.show()

#data_dir = '../sekvence//'
#path = os.path.join(data_dir, 'miner_sharp.jpg')
io.imsave('miner_sharp.jpg', img_as_ubyte(imgRGBout))

```



Ovo nije jedina metoda koja je mogla da se koristi kako bi se izoštrila data slika, ali je korišćena jer je jednostavna za implementaciju, ne dobija se šum na slici, a rezultati izoštravanja su sasvim zadovoljavajući.

## 1.4 3.Zadatak

U trećem zadatku implementirana je funkcija dosCLAHE koja realizuje adaptivnu ekvalizaciju histograma uz ograničenje kontrasta.

### 1.4.1 Adaptivna ekvalizacija histograma uz ograničenje kontrasta (CLAHE)

Ekvalizacija histograma služi za poboljšanje kontrasta na slici. Kod obične ekvalizacije mogu da se javi određeni problemi. Ukoliko je slika niskog kontrasta i u tamnim delovima nema detalja, kumulativna funkcija verovatnoće imaće dosta strmu karakteristiku. Kada se primeni takva funkcija u funkciji ekvalizacije, dobiće se slika sa mnogo šuma. Problem se javlja zbog strmog skoka u transformacionoj funkciji (jer onda se mali opseg vrednosti mapira na ceo intenzitet opsega slike, pa će se mali broj vrednosti razvući na veliku oblast intenziteta). Posledica strme karakteristike je izuzetno visok pik na histogramu. Zato se uvodi ograničenje kontrasta kod ekvalizacije. Ideja je da, pre određivanja funkcije preslikavanja, ograničimo maksimalni nagib te funkcije kako bi se izbeglo prenaglašavanje kontrasta u nekim regionima slike. Što je limit manji, ekvalizacija je ograničenija i kontrast je manji, čim se poveća limit previše, javiće se šum. Klip limit donekle rešava problem. Kod slika koje imaju i tamne delove, ali se na njima javljaju i previše svetli delovi, javiće se problem. Na histogramu takvih slika dominiraju delovi kojih ima najviše. Tada ekvalizacija pokuša da razvuče onaj deo histograma u kom ima najveći broj piksela. U primeru koji se posmatra, odnosno na slici train.jpg, najveći deo slike je taman, ali postoji i svetli deo. Ekvalizacija bi uzela i poboljšala samo taman deo slike, ali bi svetli deo postao prezasićen, čak i uz

ograničenje kontrasta. Ideja kod CLAHE algoritma jeste da ne posmatramo sliku kao jednu celinu, jer ukoliko je slika složena, različiti delovi slike imaju različite statističke karakteristike. Obična ekvalizacija pokušava da modifikuje ovakve delove slike, dakle, čiji su histogrami različiti, istom transformacionom funkcijom. Kod CLAHE algoritma slika se podeli na nepreklapajuće blokove i za svaki blok se izračuna funkcija ekvalizacije histograma. Određivanje izlaznih piksela vrši se pomoću bilinearne interpolacije.

#### 1.4.2 Pomoćne funkcije korišćene u funkciji dosCLAHE:

Funkcija koja vraća histogram prosleđenih vrednosti:

```
[11]: def mojHistogram(arr, bins=256, bin_range_min=0, bin_range_max=255):
    """
    Returns the histogram of an array

    Parameters:
    array(1d array): input array for which the histogram is calculated
    bins(int): tells how many bins the histogram has
    bin_range_min, bin_range_max(int): specify the range of values to be observed

    Returns:
    histVal(array): the values of histogram
    bin_edges(array): returns the bin edges (length(histogram)+1)

    """
    histVal = np.zeros(bins)
    bin_edges = np.zeros(bins+1)
    bin_edges[0] = 0;
    dx = (bin_range_max-bin_range_min)/bins;
    for i in range(0,bins):
        val = (arr>=i*dx) & (arr<(i+1)*dx)
        bin_edges[i+1] = (i+1)*dx;
        histVal[i] = sum(val)
    return histVal, bin_edges
```

**Funkcija Vrati\_sk koja vraća funkciju preslikavanja T (odnosno sk) za prosleđenu sliku:** Ideja je da se nadje histogram prosleđenih elemenata(vrednosti niza, u ovom slučaju slike), potom uradi ograničenje kontrasta tako što se odseku vrednosti histograma veće od zadatog limita, ali se površina koju one zauzimaju na histogramu doda duž celog opsega vrednosti, kako bi površina raspodele ostala jedan. Za dobijeni histogram se potom izračuna kumulativna suma, odnosno to će biti transformaciona funkcija ekvalizacije za posmatrani histogram.

```
[12]: def Vrati_sk(im, nbr_bins=256, limit=0.01):
    """
    Returns the transfer function for equalization of input array

    Parameters:
```

```

array(2d array): input image as 2d array
nbr_bins(int): tells how many bins the histogram of input array has
limit(float): set a limit that the histogram values should not exceed

Returns:
s(array): normalized cumulative distribution of histogram
representing the transfer function for the equalization
of the input string
"""
hist,bins = mojHistogram(im.flatten(),nbr_bins,0,255)

#normalization of histogram
hist = hist/(im.shape[0]*im.shape[1])

#clip limit of histogram
povrs = 0
for i in range(256):
    if (hist[i]>limit):
        povrs+=hist[i]-limit
povrs/=256
hist = np.clip(hist,0,limit)
hist +=povrs*ones(256)

#cumulative sum of histogram
s = np.cumsum(hist)
#normalization of sum
s = 255 * s

return s

```

**Funkcija koja izračunava novu vrednost zadatog piksela pomoću bilinearne interpolacije:** Na osnovu pozicije piksela na slici i trenutnog bloka u kom se nalazi, bira se koji će blokovi uticati na promenu intenziteta piksela. Računanjem udaljenosti piksela po horizontali i vertikali od centara blokova koji ulaze u interpolaciju, dobiće se koliko koji blok utiče na promenu intenziteta piksela. Primenom formula za bilinearnu transformaciju dobija se novi intenzitet piksela koji se posmatra, što je i povratna vrednost ove funkcije.

```
[13]: def bil_int_piksela(slika,tren_blok, dimbloka_n,dimbloka_m,br_blokova_h,x,y,_
→dim_slike_n,dim_slike_m, sk):
"""
Returns bilinear interpolation of pixel slika[x,y]
"""

if (x<0 or y<0 or x>dim_slike_n or y>dim_slike_m):
    print("Dimension error")
    return -1

ind_pre = tren_blok - br_blokova_h
```

```

ind_posle = tren_blok + br_blokova_h

#top block
if (x<dimbloka_n):
    t_10=0
else:
    t_10=1

#left block
if (y<dimbloka_m):
    t0_1=0
else:
    t0_1=1

#bottom block
if x>=(dim_slike_n-dimbloka_n):
    t10=0
else:
    t10=1

#right block
if (y>=(dim_slike_m-dimbloka_m)):
    t01=0
else:
    t01=1

#top left block
if t_10==0 or t0_1==0:
    t_1_1=0
else:
    t_1_1=1

#top right block
if t_10==0 or t01==0:
    t_11=0
else:
    t_11=1

#bottom left block
if t0_1==0 or t10==0:
    t1_1=0
else:
    t1_1=1

#bottom right block
if t10==0 or t01==0:
    t11=0

```

```

else:
    t11=1

#calculation of coordinates of center and top left pixel
x_GL = dimbloka_n * floor(x/dimbloka_n)
y_GL = dimbloka_m * floor(y/dimbloka_m)
x_C = dimbloka_n * floor(x/dimbloka_n) + (dimbloka_n-1)/2
y_C = dimbloka_m * floor(y/dimbloka_m) + (dimbloka_m-1)/2

#calculating which blocks are used in interpolation
razX = x-x_C
razY = y-y_C

#left top quadrant
if (razX<0 and razY<0):
    a = 0.5+dimbloka_m/2 + (y % dimbloka_m)
    b = dimbloka_m/2 - (y % dimbloka_m) - 0.5
    c = 0.5 + dimbloka_n/2 + (x % dimbloka_n)
    d = dimbloka_n/2 -(x % dimbloka_n) - 0.5
    T1 = t_1_1*sk[t_1_1*(ind_pre-1)][slika[x,y]];T2 =_
    ↪t_10*sk[t_10*(ind_pre)][slika[x,y]]; T4= sk[tren_blok][slika[x,y]]; T3=_
    ↪t0_1*sk[t0_1*(tren_blok-1)][slika[x,y]];
    T1_bool = t_1_1==1
    T2_bool = t_10==1
    T4_bool = True
    T3_bool = t0_1==1

#left bottom quadrant
elif (razX>=0 and razY<0):
    a = 0.5 + dimbloka_m/2 + (y % dimbloka_m)
    b = dimbloka_m/2 - (y % dimbloka_m) - 0.5
    c = (x % dimbloka_n) - dimbloka_n/2 + 0.5
    d = dimbloka_n/2 + (dimbloka_n-1 - (x % dimbloka_n)) + 0.5
    T1 = t0_1*sk[t0_1*(tren_blok-1)][slika[x,y]];T2 =_
    ↪sk[tren_blok][slika[x,y]]; T4= t10*sk[t10*(ind_posle)][slika[x,y]]; T3=_
    ↪t1_1*sk[t1_1*(ind_posle-1)][slika[x,y]];
    T1_bool = t0_1==1
    T2_bool = True
    T4_bool = t10==1
    T3_bool = t1_1==1

#right top quadrant
elif (razX<0 and razY>=0):
    a = 0.5 - dimbloka_m/2 + (y % dimbloka_m)
    b = dimbloka_m/2 + 0.5 + (dimbloka_m-1-(y % dimbloka_m))
    c = 0.5 + dimbloka_n/2 + (x % dimbloka_n)

```

```

d = dimbloka_n/2 -(x % dimbloka_n) - 0.5
T1 = t_10*sk[t_10*(ind_pre)][slika[x,y]];T2 =_
→t_11*sk[t_11*(ind_pre+1)][slika[x,y]];T4=_
→t01*sk[t01*(tren_bloc)+1][slika[x,y]];T3= sk[tren_bloc][slika[x,y]];
T1_bool = t_10==1
T2_bool = t_11==1
T4_bool = t01==1
T3_bool = True

#right bottom quadrant
else:
    a = 0.5 - dimbloka_m/2 + (y % dimbloka_m)
    b = dimbloka_m/2 + 0.5 + (dimbloka_m-1-(y % dimbloka_m))
    c = (x % dimbloka_n) - dimbloka_n/2 + 0.5
    d = dimbloka_n/2 + (dimbloka_n-1 - (x % dimbloka_n)) + 0.5
    T1 = sk[tren_bloc][slika[x,y]];T2 =_
→t01*sk[t01*(tren_bloc+1)][slika[x,y]]; T4=_
→t11*sk[t11*(ind_posle+1)][slika[x,y]]; T3=_
→t10*sk[t10*(ind_posle)][slika[x,y]];
    T1_bool = True
    T2_bool = t01==1
    T4_bool = t11==1
    T3_bool = t10==1

    a = abs(a)
    b = abs(b)
    c = abs(c)
    d = abs(d)

#computation of sh1
sh1_bool=True
if T1_bool and T2_bool:
    sh1 = (b*T1 + a*T2)/(a+b)
elif T1_bool:
    sh1 = T1
elif T2_bool:
    sh1 = T2
else:
    sh1_bool=False

#computation of sh2
sh2_bool = True
if T3_bool and T4_bool:
    sh2 = (b*T3 + a*T4)/(a+b)
elif T3_bool:
    sh2 = T3
elif T4_bool:

```

```

    sh2 = T4
else:
    sh2_bool=False

#computation of s
if sh1_bool and sh2_bool:
    s = (d*sh1 + c*sh2)/(c+d)
elif sh1_bool:
    s=sh1
else:
    s=sh2

return s

```

1.4.3 dosCLAHE funkcija, koja vrši adaptivnu ekvalizaciju histograma slike uz ograničenje kontrasta:

```
[14]: def dosCLAHE(imgIn, numTiles=[8, 8], limit=0.01):
    """
    Contrast Limited Adaptive Histogram Equalization

    Algorithm for local contrast enhancement, that uses
    histograms computed over different tile regions of
    the image.

    Parameters:
    imgIn(ndarray): input 2D or 3D image with uint8 dtype
    numTiles(array): an array of 2 elements representing the
        number of blocks horizontally and vertically
    limit(float): maximum value per element of the histogram

    Returns:
    imgOut(ndarray): equalized 2D or 3D image with uint8 dtype
    """
    n = np.shape(imgIn)[0]
    m = np.shape(imgIn)[1]

    #validation of input parameters
    if (np.max(imgIn)>255 or (np.min(imgIn)<0)):
        print('greska')
        return -1
    if (np.max(imgIn)<1):
        print('greska')
        return -1
    if (type(numTiles[0])!=int or type(numTiles[1])!=int):
        print('greska')
```

```

    return -1
if (numTiles[0]<=0 or numTiles[1]<=0):
    print('greska')
    return -1
if (limit<0 or limit>1):
    print('greska')
    return -1

#case if input image is in RGB format, only apply CLAHE to Y component of
→YUV image
if (len(shape(imgIn))==3):
    imgYUV = color.rgb2yuv(imgIn)
    imgYUVout = np.zeros(shape(imgIn))
    imgYUVout[:, :, 1:3] = imgYUV[:, :, 1:3]
    #take Y component, and scale the range of values from 0 to 255
    Yin = (clip(imgYUV[:, :, 0]*255, 0, 255)).astype(int)
    imgYUVout[:, :, 0] = (clip(dosCLAHE(Yin,numTiles,limit)/255, 0, 1)).
→astype(float);
    imgOut1 = color.yuv2rgb(imgYUVout)
    imgOut1[imgOut1>1] = 1
    imgOut1[imgOut1<0] = 0
    return imgOut1

slika = imgIn

#image expansion
b = True
if (n % numTiles[0]!=0):
    b = False
    lastcol = slika[-1]
    slika=np.transpose(slika)
    lastcol = np.transpose([lastcol]*(numTiles[0]-n%numTiles[0]))
    slika = np.concatenate((slika,lastcol),axis=1)
if (m % numTiles[1]!=0):
    if b:
        slika=np.transpose(slika)
        b=False
        lastrow = slika[-1]
        c=np.tile(lastrow,(numTiles[1]-m%numTiles[1],1))
        slika = np.concatenate((slika,c))

if not b:
    slika = transpose(slika)

n = np.shape(slika)[0]
m = np.shape(slika)[1]

```

```

#height and width of each block in image
heightBlock = int(n/numTiles[0])
widthBlock = int(m/numTiles[1])

#calculus of transfer function T for each block
sk = []
for i in range(0,numTiles[0]):
    for j in range(0,numTiles[1]):
        sk.append(Vrati_sk(slika[i*heightBlock:
→(i+1)*heightBlock,j*widthBlock:(j+1)*widthBlock],256,limit))
sk = np.asarray(sk)

#use bilinear interpolation for pixels in image to get new values of pixels
slika2 = np.copy(slika)
for i in range(0,n):
    tren_blok = int(trunc(i/heightBlock))*numTiles[1]
    for j in range(0,m):
        slika2[i,j] = bil_int_piksela(slika,tren_blok,heightBlock,widthBlock,numTiles[1],i,j,n,m,sk)
        if ((j % widthBlock)==(widthBlock-1)):
            tren_blok=tren_blok+1

#restoring the initial dimensions of the image
imgOut = slika2[0:np.shape(imgIn)[0],:][:,0:np.shape(imgIn)[1]]

#imgOut = exposure.rescale_intensity(imgOut)
return imgOut

```

#### 1.4.4 Uticaj parametara i upoređivanje dobijenih rezultata

Funkcija je testirana za veličine blokova 1, 4, 8 i 16, kao i za ograničenje kontrasta 0, 0.01, 0.1 i 1. Izmereno je vreme izvršavanja funkcije u ovim slučajevima i upoređeni su dobijeni rezultati, kao i vreme izvršavanja, sa rezultatima i vremenom izvršavanja ugrađene funkcije exposure.equalize\_adapthist.

**Uticaj ograničenja kontrasta i broj blokova po horizontali i vertikali na rezultate** Ukoliko je ograničenje kontrasta 0, naša funkcija prvo odseče ceo histogram, a potom sve vrednosti zapravo raspodeli uniformno jer nakon odsecanja je potrebno da vrati površinu koja je odsečena ali tako da ona što manje utiče na maksimalnu vrednost histograma. Nakon izvršene adaptivne ekvalizacije, slika nije popravljena u odnosu na ulaznu sliku, i dobijena slika je ista za svaku veličinu bloka ukoliko je ograničenje kontrasta 0. Primećeno je da se ova slika dosta razlikuje u odnosu na sliku dobijenu ugrađenom funkcijom. Ako je limit 1, algoritam se ponaša kao da je u pitanju ekvalizacija bez ograničenja kontrasta ako je veličina bloka 1, te je dobijena slika sa previše naglašenih detalja u tamnim delovima, a svetli delovi slike postali su previše zasićeni. Što broj blokova veći, to je zasićenost svetlih delova manja, ali je šum i izraženost detalja više prisutna. Ako se uporede

rezultati sa ovim ograničenjem sa rezultatima ugrađene funkcije, primećuje se da se dobija isti rezultat. Za ograničenje 0.1 već se dobija loš rezultat, slika ne izgleda prirodno i javljaju se isti problemi kao kada je ograničenje kontrasta 1, osim što su za nijansu blaži, što se uočava pri većem broju blokova. Za ograničenje 0.01 dobija se najbolja slika nakon primene dosCLAHE funkcije, jer se tamni delovi razvuku da veći opseg vrednosti, te postanu svetlij, a svetli delovi ne postanu previše zasićeni, i kontrast je bolji u odnosu na ulaznu sliku. Primećeno je da što je ograničenje kontrasta manje, to broj bloka ima manji uticaj na rezultujuću sliku. Drugim rečima, broj blokova nije uticao na rezultujuću sliku kada su limiti bili 0 i 0.01, dok je sa 0.1 i 1, kako se menja broj blokova po horizontali i vertikali, dobijena drugačija slika na izlazu funkcije dosCLAHE. Takođe, što je veći broj blokova po horizontali i vertikali, kontrast postaje izraženiji na celoj slici.

```
[29]: #loading image
train = imread('../sekvence/train.jpg').astype(np.uint8)

#numTiles=[1,1]
RGBout1 = dosCLAHE(train,[1,1],0)
RGBout2 = dosCLAHE(train,[1,1],0.01)
RGBout3 = dosCLAHE(train,[1,1],0.1)
RGBout4 = dosCLAHE(train,[1,1],1)

#plotting
fig, ax = plt.subplots(ncols=4, figsize=(16,8), dpi=80)
ax[0].imshow(RGBout1); ax[0].set_title('limit=0'); ax[0].axis('off');
ax[1].imshow(RGBout2); ax[1].set_title('limit=0.01'); ax[1].axis('off')
ax[2].imshow(RGBout3); ax[2].set_title('limit=0.1'); ax[2].axis('off');
ax[3].imshow(RGBout4); ax[3].set_title('limit=1'); ax[3].axis('off');
plt.tight_layout()
plt.show()

#numTiles=[4,4]
RGBout5 = dosCLAHE(train,[4,4],0)
RGBout6 = dosCLAHE(train,[4,4],0.01)
RGBout7 = dosCLAHE(train,[4,4],0.1)
RGBout8 = dosCLAHE(train,[4,4],1)

#plotting
fig, ax = plt.subplots(ncols=4, figsize=(16,8), dpi=80)
ax[0].imshow(RGBout5); ax[0].set_title('limit=0'); ax[0].axis('off')
ax[1].imshow(RGBout6); ax[1].set_title('limit=0.01'); ax[1].axis('off')
ax[2].imshow(RGBout7); ax[2].set_title('limit=0.1'); ax[2].axis('off');
ax[3].imshow(RGBout8); ax[3].set_title('limit=1'); ax[3].axis('off');
plt.tight_layout()
plt.show()

#numTiles=[8,8]
RGBout9 = dosCLAHE(train,[8,8],0)
```

```

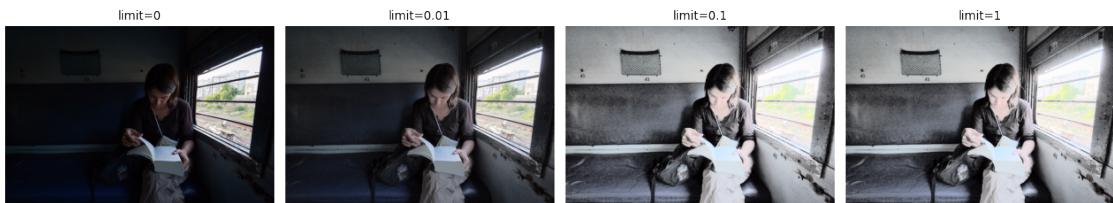
RGBout10 = dosCLAHE(train,[8,8],0.01)
RGBout11= dosCLAHE(train,[8,8],0.1)
RGBout12= dosCLAHE(train,[8,8],1)

#plotting
fig, ax = plt.subplots(ncols=4, figsize=(16,8), dpi=80)
ax[0].imshow(RGBout9); ax[0].set_title('limit=0'); ax[0].axis('off')
ax[1].imshow(RGBout10); ax[1].set_title('limit=0.01'); ax[1].axis('off')
ax[2].imshow(RGBout11); ax[2].set_title('limit=0.1'); ax[2].axis('off');
ax[3].imshow(RGBout12); ax[3].set_title('limit=1'); ax[3].axis('off');
plt.tight_layout()
plt.show()

#numTiles=[16, 16]
RGBout13 = dosCLAHE(train,[16,16],0)
RGBout14 = dosCLAHE(train,[16,16],0.01)
RGBout15= dosCLAHE(train,[16,16],0.1)
RGBout16= dosCLAHE(train,[16,16],1)

#plotting
fig, ax = plt.subplots(ncols=4, figsize=(16,8), dpi=80)
ax[0].imshow(RGBout13); ax[0].set_title('limit=0'); ax[0].axis('off')
ax[1].imshow(RGBout14); ax[1].set_title('limit=0.01'); ax[1].axis('off')
ax[2].imshow(RGBout15); ax[2].set_title('limit=0.1'); ax[2].axis('off');
ax[3].imshow(RGBout16); ax[3].set_title('limit=1'); ax[3].axis('off');
plt.tight_layout()
plt.show()

```





**Upoređivanje rezultata sa dobijenim rezultatima iz ugrađene funkcije exposure.equalize\_adapthist** Upoređivanjem dobijenih rezultata, može se zaključiti da je ove dve funkcije daju različit izlaz za ograničenje kontrasta 0. Objasnjenje za to možda je u tome na koji način se vrši ograničenje kontrasta u ugrađenoj funkciji i onoj koja je implementirana u ovom domaćem. Naime, u implementaciji ograničavanja histograma, kada vraćamo odsečenu površinu u histogram ali je raspoređujemo tako što samo dodamo jedan pravougaonik, iako smo odsekli deo histograma da vrednosti ne bi prelazile ograničenje, nakon dodavanja ovih vrednosti, postojaće delovi histograma koji će da premašuju ograničenje. Ugrađena funkcija verovatno ovo rešava tako da drugačije raspodeli odsečenu površinu, kako bi vrednosti histograma zaista bile ograničene limitom. Maksimalno ograničenje je to od kog zavisi koliko će ova pojava da utiče na rezultat algoritma. Naša slika se veoma malo razlikuje u odnosu na sliku dobijenu iz ugrađene funkcije, a to se najbolje može videti ukoliko se nadje razlika ove dve slike (oduzme se jedna matrica od druge i dobijena matrica predstavlja sliku razlike), ali se primećuje i ukoliko se posmatraju tamniji delovi ovih slika.

**Vreme izvršavanja** Ugrađena funkcija brža je od implementirane funkcije u ovom domaćem iz razloga što su u računima korištene operacije koje jezik Python podržava a ubrzavaju rad algoritma u odnosu na iterativni prolazak kroz matricu ili niz. Vreme izvršavanja funkcije dosCLAHE je između 10 i 16 sekundi, dok je vreme izvršavanja ugrađene funkcije reda veličine 0.1 sekunda. Naša funkcija bi mogla da se ubrza tako što se u histogramu, ali i u ostalim funkcijama, for petlje zamene naredbama za nizove čija je vremenska složenost mnogo manja od prolaska kroz niz iterativno. To bi se naročito moglo uraditi kod računanja bilinearne transformacije (da se ne računa u jednoj iteraciji za jedan piksel, već jednim prolaskom za ceo red piksela na primer ili sve piksele iz jednog bloka).

```
[31]: train1 = imread('../sekvence/train.jpg').astype(np.uint8)
imgYUV1 = color.rgb2yuv(train1)

Yin1 = (clip(imgYUV1[:, :, 0]*255, 0, 255)).astype(int)
```

```

print('dosCLAHE funkcija:')
start = time.time()
B1 = dosCLAHE(Yin1,[8,8],0.01)
end = time.time()
ex_time = (end-start)
print('Vreme izvrsavanja u sekundama je ', + round(ex_time,3))
execution_time_norm = ex_time/np.size(train1[:, :, 0])
print('Vreme izvrsavanja:' + str(round(ex_time*1e6,3))+ 'us/pix')

imgYUVout1 = np.zeros(shape(train1))
imgYUVout1[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout1[:, :, 0] = (clip(B1/255,0,1)).astype(float);
imgOut11 = color.yuv2rgb(imgYUVout1)
imgOut11[imgOut11>1] = 1
imgOut11[imgOut11<0] = 0

#ugradjena funkcija
print('Funkcija exposure.equalize_adapthist:')
start = time.time()
Yout = exposure.equalize_adapthist(imgYUV1[:, :, 0], [np.shape(Yin1)[0]/8, np.
    ↪shape(Yin1)[1]/8], clip_limit=0.01, nbins=256)
end = time.time()
ex_time = (end-start)
print('Vreme izvrsavanja u sekundama je ', + round(ex_time,3))
execution_time_norm = ex_time/np.size(train1[:, :, 0])
print('Vreme izvrsavanja:' + str(round(ex_time*1e6,3))+ 'us/pix')
imgYUVout2 = np.zeros(shape(train1))
imgYUVout2[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout2[:, :, 0] = Yout;
imgOut12 = color.yuv2rgb(imgYUVout2)
imgOut12[imgOut12>1] = 1
imgOut12[imgOut12<0] = 0

difference = abs(imgOut11-imgOut12)

fig, ax = plt.subplots(ncols=3, figsize=(14,10), dpi=120)
ax[0].imshow(imgOut11); ax[0].set_title('dosCLAHE,n=8,limit=0.01'); ax[0].
    ↪axis('off')
ax[1].imshow(imgOut12); ax[1].set_title('exposure.
    ↪equalize_adapthist,n=8,limit=0.01'); ax[1].axis('off')
ax[2].imshow(difference, cmap = 'gray'); ax[2].set_title('razlika'); ax[2].
    ↪axis('off')
plt.tight_layout()
plt.show()

```

```

dosCLAHE funkcija:
Vreme izvrsavanja u sekundama je 11.027
Vreme izvrsavanja:11026845.932us/pix
Funkcija exposure.equalize_adapthist:
Vreme izvrsavanja u sekundama je 0.27
Vreme izvrsavanja:269706.011us/pix

```



Kao što je razmatrano, slika je skoro identična slici koja se dobija ugrađenom funkcijom. U nastavku će biti prikazani rezultati kada se menja broj blokova, i kada se menja maksimalno ograničenje kontrasta.

```

[32]: train1 = imread('../sekvence/train.jpg').astype(np.uint8)
imgYUV1 = color.rgb2yuv(train1)

Yin1 = (clip(imgYUV1[:, :, 0]*255, 0, 255)).astype(int)

B1 = dosCLAHE(Yin1, [8,8], 0.1)
imgYUVout1 = np.zeros(shape(train1))
imgYUVout1[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout1[:, :, 0] = (clip(B1/255, 0, 1)).astype(float);
imgOut11 = color.yuv2rgb(imgYUVout1)
imgOut11[imgOut11>1] = 1
imgOut11[imgOut11<0] = 0

#ugradjena funkcija
Yout = exposure.equalize_adapthist(imgYUV1[:, :, 0], [np.shape(Yin1)[0]/8, np.
    ↪shape(Yin1)[1]/8], clip_limit=0.1, nbins=256)
imgYUVout2 = np.zeros(shape(train1))
imgYUVout2[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout2[:, :, 0] = Yout;
imgOut12 = color.yuv2rgb(imgYUVout2)
imgOut12[imgOut12>1] = 1
imgOut12[imgOut12<0] = 0

difference = abs(imgOut11-imgOut12)

```

```

fig, ax = plt.subplots(ncols=3, figsize=(14,10), dpi=120)
ax[0].imshow(imgOut11); ax[0].set_title('dosCLAHE,n=8,limit=0.1'); ax[0].
    ~axis('off')
ax[1].imshow(imgOut12); ax[1].set_title('exposure.
    ~equalize_adapthist,n=8,limit=0.1'); ax[1].axis('off')
ax[2].imshow(difference, cmap='jet', vmin=0, vmax=255); ax[2] .
    ~set_title('razlika'); ax[2].axis('off')
plt.tight_layout()
plt.show()

```

```
Yin1 = (clip(imgYUV1[:, :, 0]*255, 0, 255)).astype(int)
```

```

B1 = dosCLAHE(Yin1,[8,8],1)
imgYUVout1 = np.zeros(shape(train1))
imgYUVout1[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout1[:, :, 0] = (clip(B1/255,0,1)).astype(float);
imgOut11 = color.yuv2rgb(imgYUVout1)
imgOut11[imgOut11>1] = 1
imgOut11[imgOut11<0] = 0

```

#### *#ugradjena funkcija*

```

Yout = exposure.equalize_adapthist(imgYUV1[:, :, 0], [np.shape(Yin1)[0]/8, np.
    ~shape(Yin1)[1]/8], clip_limit=1, nbins=256)
imgYUVout2 = np.zeros(shape(train1))
imgYUVout2[:, :, 1:3] = imgYUV1[:, :, 1:3]
imgYUVout2[:, :, 0] = Yout;
imgOut12 = color.yuv2rgb(imgYUVout2)
imgOut12[imgOut12>1] = 1
imgOut12[imgOut12<0] = 0

```

```
difference = abs(imgOut11-imgOut12)
```

```

fig, ax = plt.subplots(ncols=3, figsize=(14,10), dpi=120)
ax[0].imshow(imgOut11); ax[0].set_title('dosCLAHE,n=8,limit=1'); ax[0].
    ~axis('off')
ax[1].imshow(imgOut12); ax[1].set_title('exposure.
    ~equalize_adapthist,n=8,limit=1'); ax[1].axis('off')
ax[2].imshow(difference, cmap='jet', vmin=0, vmax=255); ax[2] .
    ~set_title('razlika'); ax[2].axis('off')
plt.tight_layout()
plt.show()

```

```
B1 = dosCLAHE(Yin1,[4,4],0.01)
```

```



```

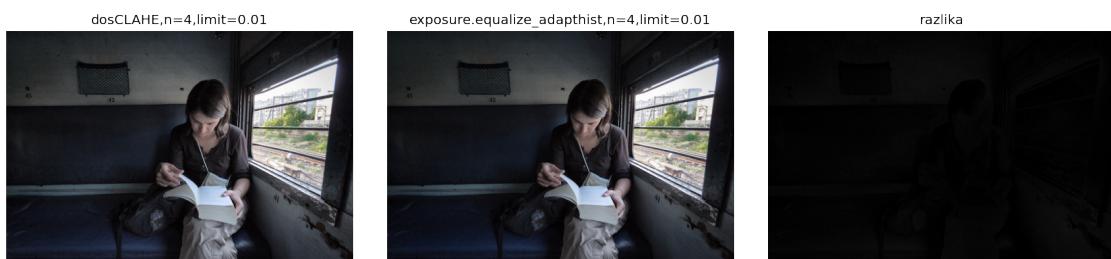
```

imgYUVout2[:, :, 0] = Yout;
imgOut12 = color.yuv2rgb(imgYUVout2)
imgOut12[imgOut12>1] = 1
imgOut12[imgOut12<0] = 0

difference = abs(imgOut11-imgOut12)

fig, ax = plt.subplots(ncols=3, figsize=(14,10), dpi=120)
ax[0].imshow(imgOut11); ax[0].set_title('dosCLAHE,n=16,limit=0.01'); ax[0].
    ~axis('off')
ax[1].imshow(imgOut12); ax[1].set_title('exposure.
    ~equalize_adapthist,n=16,limit=0.01'); ax[1].axis('off')
ax[2].imshow(difference, cmap='jet', vmin=0, vmax=255); ax[2] .
    ~set_title('razlika'); ax[2].axis('off')
plt.tight_layout()
plt.show()

```





```
[33]: #loading image
train2 = imread('..../sekvence/train.jpg').astype(np.uint8)

ex_time1 = []
execution_time_norm1 = []
ex_time2 = []
execution_time_norm2 = []

#posmatra se vreme ako se menja broj blokova, limit=0.01
start = time.time()
RGBo1 = dosCLAHE(train2,[1,1],0.01)
end = time.time()
ex_time1.append(end-start)
execution_time_norm1.append(ex_time1[0]/np.size(train2[:, :, 0]))

start = time.time()
RGBo2 = dosCLAHE(train2,[4,4],0.01)
end = time.time()
ex_time1.append(end-start)
execution_time_norm1.append(ex_time1[1]/np.size(train2[:, :, 0]))

start = time.time()
RGBo3 = dosCLAHE(train2,[8,8],0.01)
end = time.time()
ex_time1.append(end-start)
execution_time_norm1.append(ex_time1[2]/np.size(train2[:, :, 0]))

start = time.time()
RGBo4 = dosCLAHE(train2,[16,16],0.01)
end = time.time()
ex_time1.append(end-start)
execution_time_norm1.append(ex_time1[3]/np.size(train2[:, :, 0]))
```

```

#posmatra se vreme ako se menja limit, broj blokova je 4
start = time.time()
RGBo5 = dosCLAHE(train2,[4,4],0)
end = time.time()
ex_time2.append(end-start)
execution_time_norm2.append(ex_time2[0]/np.size(train2[:, :, 0]))

start = time.time()
RGBo6 = dosCLAHE(train2,[4,4],0.01)
end = time.time()
ex_time2.append(end-start)
execution_time_norm2.append(ex_time2[1]/np.size(train2[:, :, 0]))

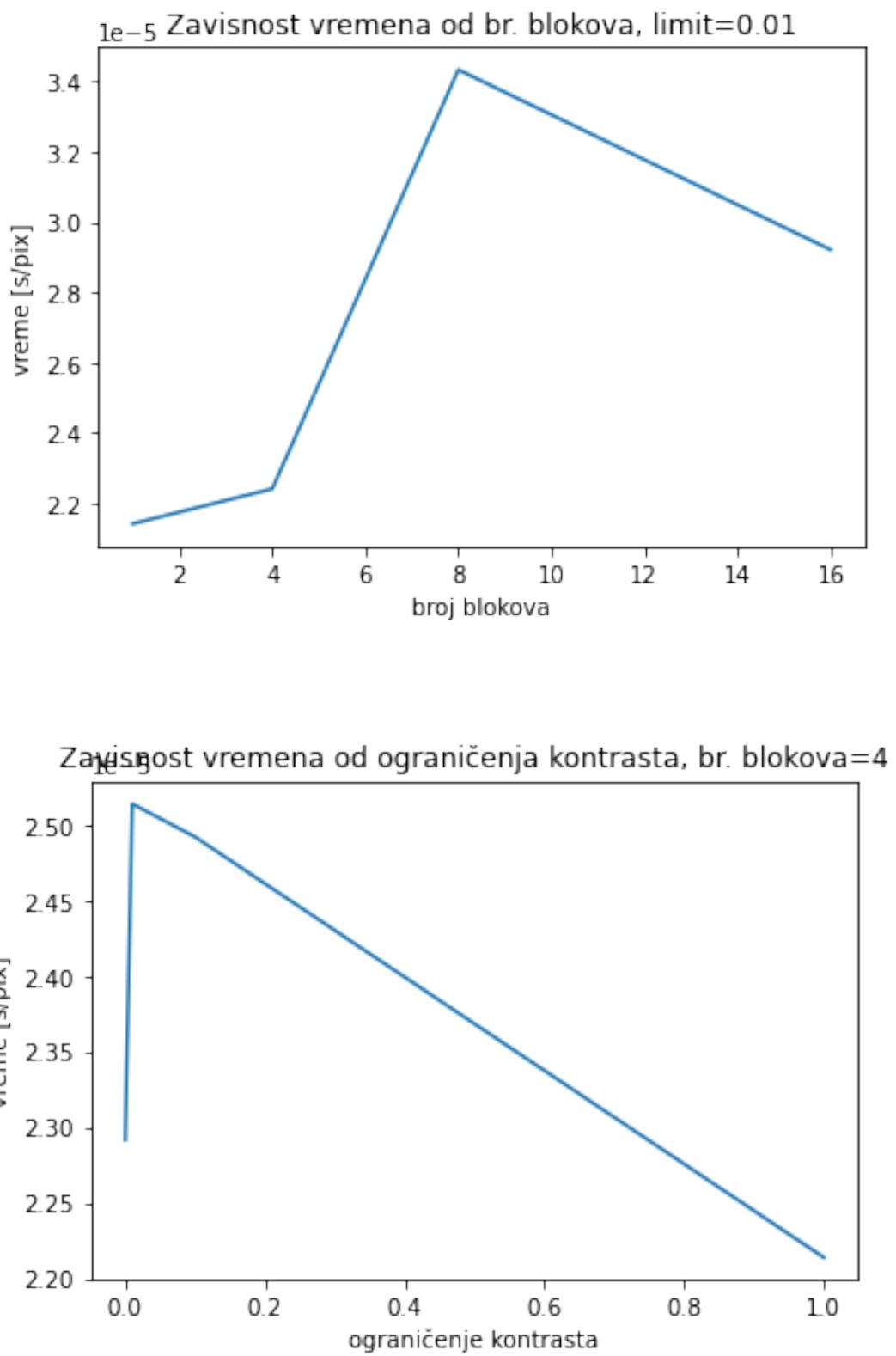
start = time.time()
RGBo7 = dosCLAHE(train2,[4,4],0.1)
end = time.time()
ex_time2.append(end-start)
execution_time_norm2.append(ex_time2[2]/np.size(train2[:, :, 0]))

start = time.time()
RGBo8 = dosCLAHE(train2,[4,4],1)
end = time.time()
ex_time2.append(end-start)
execution_time_norm2.append(ex_time2[3]/np.size(train2[:, :, 0]))


figure();
plt.plot([1,4,8,16],np.asarray(execution_time_norm1))
plt.xlabel('broj blokova')
plt.ylabel('vreme [s/pix]')
plt.title('Zavisnost vremena od br. blokova, limit=0.01')
plt.show()

figure();
plt.plot([0,0.01,0.1,1],np.asarray(execution_time_norm2))
plt.xlabel('ograničenje kontrasta')
plt.ylabel('vreme [s/pix]')
plt.title('Zavisnost vremena od ograničenja kontrasta, br. blokova=4')
plt.show()

```



Sa grafika zavisnosti vremena vidi se da vreme izvršavanja funkcije dosCLAHE raste sa povećanjem broja blokova, što ima smisla iz razloga što je potrebno da se izračuna više funkcija ekvalizacije Ti, za svaki blok i. U nekoliko realizacija primećeno je da vreme ne zavisi mnogo od ograničenja konstrasta. Prikazaćemo ispod grafike zavisnosti vremena ugrađene funkcije, za iste parametre kao iz prethodnog primera.

```
[34]: #loading image
train3 = imread('../sekvence/train.jpg').astype(np.uint8)
train4 = color.rgb2yuv(train3)

ex_time3 = []
execution_time_norm3 = []
ex_time4 = []
execution_time_norm4 = []

#posmatra se vreme ako se menja broj blokova, limit=0.01
start = time.time()
RGBBo11 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪1, np.shape(train4[:, :, 0])[1]/1], clip_limit=0.01, nbins=256)
end = time.time()
ex_time3.append(end-start)
execution_time_norm3.append(ex_time3[0]/np.size(train3[:, :, 0]))

start = time.time()
RGBBo21 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪4, np.shape(train4[:, :, 0])[1]/4], clip_limit=0.01, nbins=256)
end = time.time()
ex_time3.append(end-start)
execution_time_norm3.append(ex_time3[1]/np.size(train3[:, :, 0]))

start = time.time()
RGBBo31 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪8, np.shape(train4[:, :, 0])[1]/8], clip_limit=0.01, nbins=256)
end = time.time()
ex_time3.append(end-start)
execution_time_norm3.append(ex_time3[2]/np.size(train3[:, :, 0]))

start = time.time()
RGBBo41 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪16, np.shape(train4[:, :, 0])[1]/16], clip_limit=0.01, nbins=256)
end = time.time()
ex_time3.append(end-start)
execution_time_norm3.append(ex_time3[3]/np.size(train3[:, :, 0]))


#posmatra se vreme ako se menja limit, broj blokova je 4
```

```

start = time.time()
RGBBo51 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪4, np.shape(train4[:, :, 0])[1]/4], clip_limit=0, nbins=256)
end = time.time()
ex_time4.append(end-start)
execution_time_norm4.append(ex_time4[0]/np.size(train3[:, :, 0]))

start = time.time()
RGBBo61 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪4, np.shape(train4[:, :, 0])[1]/4], clip_limit=0.01, nbins=256)
end = time.time()
ex_time4.append(end-start)
execution_time_norm4.append(ex_time4[1]/np.size(train3[:, :, 0]))

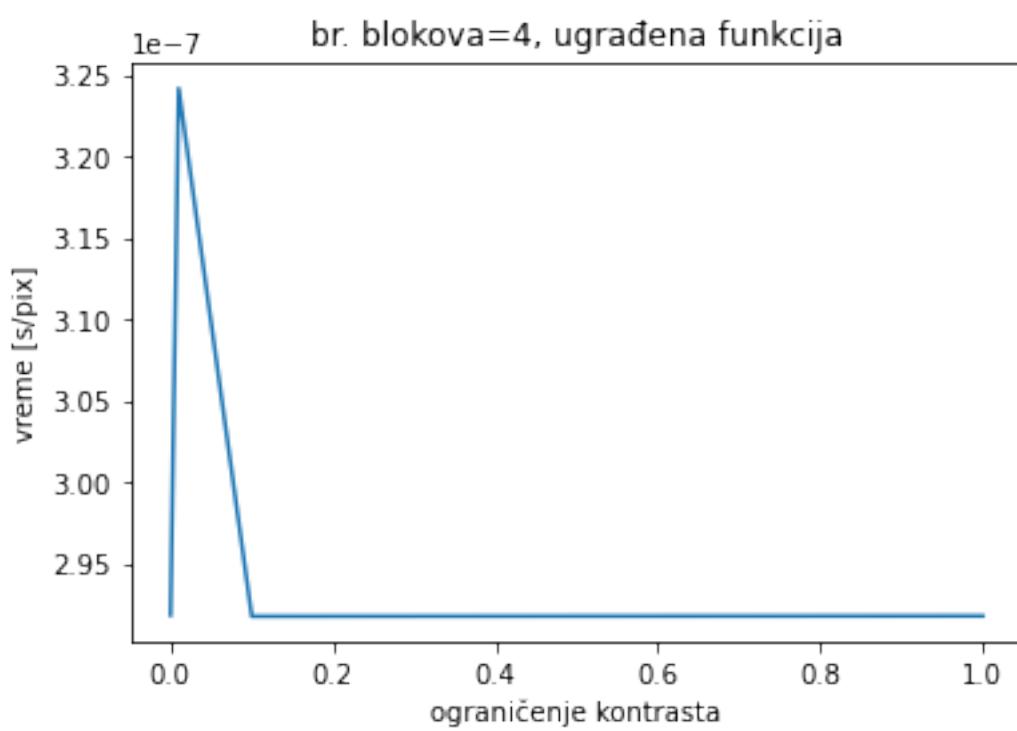
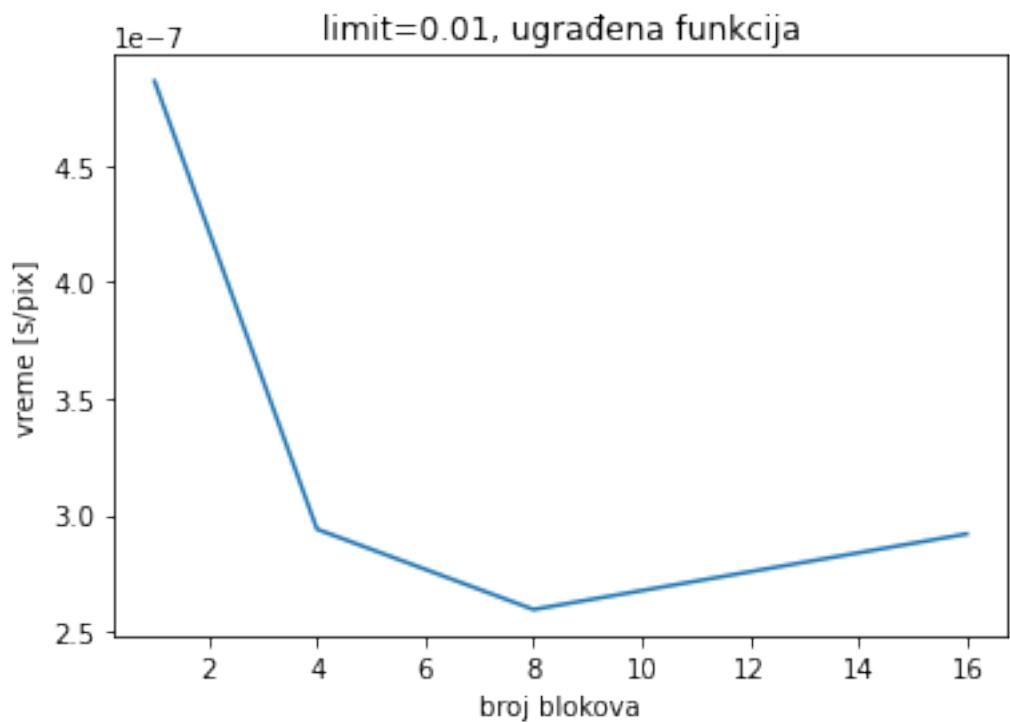
start = time.time()
RGBBo71 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪4, np.shape(train4[:, :, 0])[1]/4], clip_limit=0.1, nbins=256)
end = time.time()
ex_time4.append(end-start)
execution_time_norm4.append(ex_time4[2]/np.size(train3[:, :, 0]))

start = time.time()
RGBBo81 = exposure.equalize_adapthist(train4[:, :, 0], [np.shape(train4[:, :, 0])[0]/
    ↪4, np.shape(train4[:, :, 0])[1]/4], clip_limit=1, nbins=256)
end = time.time()
ex_time4.append(end-start)
execution_time_norm4.append(ex_time4[3]/np.size(train3[:, :, 0]))


figure();
plt.plot([1,4,8,16],np.asarray(execution_time_norm3))
plt.xlabel('broj blokova')
plt.ylabel('vreme [s/pix]')
plt.title('limit=0.01, ugrađena funkcija')
plt.show()

figure();
plt.plot([0,0.01,0.1,1],np.asarray(execution_time_norm4))
plt.xlabel('ograničenje kontrasta')
plt.ylabel('vreme [s/pix]')
plt.title('br. blokova=4, ugrađena funkcija')
plt.show()

```



Vreme izvršavanja ugrađene funkcije ne raste sa povećanjem broja blokova jer je njena složenost bar reda  $n$  manja od vremenske složenosti algoritma dosCLAHE. Kao i kod algoritma dosCLAHE, vreme izvršavanja ne zavisi od ograničenja kontrasta.