

Alati za modifikaciju i korekciju slika

Seminarski rad u okviru kursa
Naučno izračunavanje
Matematički fakultet

Aleksandar Anžel, 1025/2018

30. juni 2019

Sažetak

U ovom radu će biti detaljnije opisan skup odabranih alata za modifikaciju i korekciju slika, u programskom jeziku Python.

Sadržaj

1 Konvolucija	2
2 Zamućivanje (eng. <i>blur</i>)	3
2.1 Zamućivanje pomoću konvolucije	3
2.2 Zamućivanje pomoću nalaženja medijane	4
3 Izdvajanje ivica (eng. <i>edge detection</i>) pomoću konvolucije	5
4 Korekcija slika pomoću histograma inteziteta piksela (eng. <i>histogram equalization</i>)	6
5 Rezultati projekta	9
5.1 Zamućivanje	9
5.2 Izdvajanje ivica	10
5.3 Korekcija slika pomoću histograma inteziteta piskela	11
Literatura	12

1 Konvolucija

Konvolucija je matematički operator koji od dve funkcije f i g prizvodi treću koja predstavlja količinu preklapanja između funkcije f i okretnute i prevedene verzije funkcije g . Možemo je predstaviti sledećim izrazom:

$$k(t) = \int f(v) * g(t - v) dv \quad (1)$$

Opseg integracije zavisi od domena na kome su definisane funkcije. Treba napomenuti da u gore napisanom izrazu upotrebljeni simbol t ne mora da predstavlja vreme. U slučaju konačnog integracionog opsega za funkcije f i g se često smatra da se šire periodično u oba smera tako da $g(t - v)$ ne narušava opseg integracije. Ova upotreba periodičnih domena naziva se ciklična, cirkularna ili periodična konvolucija. Naravno, moguće je i proširenje nulama. Korišćenje domena proširenih nulom ili beskonačnih naziva se linearna konvolucija. [2]

Za diskretne funkcije koristi se sledeća verzija diskretne konvolucije:

$$(f * g)(m) = \sum_n (f(n) * g(m - n)) \quad (2)$$

Neke od važnih osobina konvolucije su:

- Komutativnost

$$f * g = g * f$$

- Asocijativnost

$$f * (g * h) = (f * g) * h$$

- Distributivnost

$$f * (g + h) = (f * g) + (f * h)$$

- Asocijativnost sa skalarnim množenjem

$$c \cdot (f * g) = (c \cdot f) * g = f * (c \cdot g)$$

- Pravilo diferencijacije

$$D(f * g) = Df * g = f * Dg$$

Gde Df označava izvod od f , ili u diskretnom slučaju operator razlike:

$$Df(n) = f(n + 1) - f(n)$$

U daljem radu ćemo se fokusirati na konvoluciju nad diskretnim funkcijama dve promenljive, gde će te funkcije biti predstavljene matricama dimenzija $m \times n$ i $p \times q$ respektivno. U tom slučaju, konvolucija predstavlja operaciju oblika:

$$(f * g)_{ij} = \sum_{k=0}^{p-1} \sum_{l=0}^{q-1} f_{i-k, j-l} * g_{k, l} \quad (3)$$

Matricu f ćemo nazivati ulaznom slikom ili samo ulazom, a matricu g filterom.[1] Primetimo da primena konvolucije se svodi na prevlačenje filtera preko ulaza sleva na desno i odozgo na dole. Takođe primetimo da je rezultujuća slika manjih dimenzija. Grafički prikaz:

Slika 1: Konvolucija ulazne slike sa filterom

U nastavku pretpostavljamo da su ulazne slike monohromatske, tj. sastoje se samo od jednog kanala boja što znači da se primena filtera nad slikom vrši nad samo tim, jednim kanalom.

2 Zamućivanje (eng. *blur*)

2.1 Zamućivanje pomoću konvolucije

Konvolucija nad dvodimenzionalnim signalima (slikama) se vrši u različite svrhe a jedna od njih upravo predstavlja zamućivanje ulaznih slika. Ovo znači da je rezultujuća slika rezultat primene odgovarajućih filtera nad istom pomoću operacije konvolucije. U projektu su korišćena dva filtera:

- Gausov filter (5×5)

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- Prosek filter (5×5)

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

```

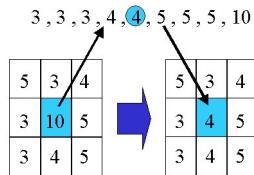
1 def primeni_filter (n_filter, putanja_slike, ime_filtera,
2                     ime_slike):
3
4     n_filter = np.array (n_filter)
5
6     slika = img.open (putanja_slike).convert ('L')
7     slika_mat = slika.load()
8     sirina = slika.size[0]
9     visina = slika.size[1]
10
11    rez = img.new ('L', (sirina, visina))
12    rez_mat = rez.load()
13
14    vel_filtera = len (n_filter)
15
16    # Krećemo se po pikselima i primenjujemo filter
17    for red in range (sirina - vel_filtera):
18        for kolona in range (visina - vel_filtera):
19
20            R = 0.0
21
22            for i in range (vel_filtera):
23                for j in range (vel_filtera):
24
25                    tmp = slika_mat [red + i, kolona + j]
26                    R += n_filter[i][j] * tmp
27
28            rez_mat[red + 1, kolona + 1] = int (R)
29
30    rez.save (os.path.join ('Rezultati', 'Glacanje', ime_slike +
31                      '_' + ime_filtera + '.jpg'))

```

Implementacija 1: Implementacija zamućivanja konvolucijom

2.2 Zamućivanje pomoću nalaženja medijane

Pored zamućivanja uz pomoć konvolucije, u projektu je obrađen i slučaj zamućivanja tehnikom nalaženja medijane. Ono što treba uzeti u obzir je da je ovo tehnika koja se koristi prilikom otklanjanja šuma iz slike uz verno čuvanje ivica. Prethodni filteri se takođe mogu koristiti za otklanjanje šuma, ali manje primena takvih filtera jesu upravo da se uklanjanjem šuma gube i ivice slike, što je u obradi signala često nepoželjno. Filter se realizuje tako što se prvo bira veličina okoline tj. filtera za koju će se tražiti medijana, zatim se vrednosti piksela sortiraju i uzme se srednji član niza kao vrednost piskela nove slike. Grafički prikaz:



Slika 2: Nalaženje medijane

Računanje medijane niza umesto proseka niza je robustniji pristup jer odudarajuće vrednosti koje se mogu naći u okolini neće značajno uticati na rezultat. Pored toga, kako medijana niza zapravo predstavlja neki od postojećih elemenata niza, to znači da se ovim metodom izbegava dobijanje nerealističnih vrednosti piksela. Veličina okoline korišćena u projektu predstavlja matricu dimenzija 5×5 .

```

1 def primeni_filter (n_filter, putanja_slike, ime_filtera,
2                     ime_slike):
3
4     n_filter = np.array (n_filter)
5
6     slika = img.open (putanja_slike).convert ('L')
7     slika_mat = slika.load()
8     sirina = slika.size[0]
9     visina = slika.size[1]
10
11    rez = img.new ('L', (sirina, visina))
12    rez_mat = rez.load()
13
14    vel_filtera = 5
15
16    # Krećemo se po pikselima i primenjujemo filter
17    for red in range (sirina - vel_filtera):
18        for kolona in range (visina - vel_filtera):
19
20            R = []
21
22            for i in range (vel_filtera):
23                for j in range (vel_filtera):
24
25                    tmp = slika_mat [red + i, kolona + j]
26                    R.append (tmp)
27
28    rez_mat [red + 1, kolona + 1] = int(np.median (np.array
(R)))

```

Implementacija 2: Implementacija zamućivanja medijanom

3 Izdvajanje ivica (eng. *edge detection*) pomoću konvolucije

Pored zamućivanja, konvolucija se može koristiti i za detektovanje ivica ulazne slike. U projektu su u ove svrhe korišćena tri filtera, pri čemu su prvo prikazani filteri za detektovanje horizontalnih pa za detektovanje vertikalnih ivica:

- Sobel filter (3×3)

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

- Prewitt filter (3×3)

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- Roberts filter (2×2)

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

```

1 def primeni_filter (x_filter, y_filter, putanja_slike, ime_filtera
2   , ime_slike):
3
4   x_filter = np.array (x_filter)
5   y_filter = np.array (y_filter)
6
7   slika = img.open (putanja_slike).convert ('L')
8   slika_mat = slika.load()
9   sirina = slika.size[0]
10  visina = slika.size[1]
11
12  rez = img.new ('L', (sirina, visina))
13  rez_mat = rez.load()
14
15  vel_x_filtera = len (x_filter)
16  vel_y_filtera = len (y_filter)
17
18  # Krecemo se po pikselima i primenjujemo filter
19  for red in range (sirina - vel_x_filtera):
20    for kolona in range (visina - vel_x_filtera):
21
22      Rx = 0
23      Ry = 0
24
25      for i in range (vel_x_filtera):
26        for j in range (vel_y_filtera):
27
28          tmp = slika_mat [red+i, kolona+j]
29          Rx += x_filter[i][j] * tmp
30          Ry += y_filter[i][j] * tmp
31
32      rez_mat[red + 1, kolona + 1] = int (math.sqrt (Rx**2 +
33      Ry**2))
34
35  rez.save (os.path.join ('Rezultati', 'Ivice', ime_slike + '_'
36  + ime_filtera + '.jpg'))

```

Implementacija 3: Implementacija detekcije ivica konvolucijom

4 Korekcija slika pomoću histograma inteziteta piksela (eng. *histogram equalization*)

Histogram slike predstavlja grafičku reprezentaciju raspodele inteziteta piksela, tj. broj piksela za svaku moguću vrednost inteziteta istih (kako posmatramo samo monohromatske slike, inteziteti su u rasponu 0-255). Korekcija slika pomoću ovakvog histograma se ogleda u poboljšanju kontrasta slike kojoj on odgovara, tako što se najfrekventnije vrednosti inteziteta raspodele po celom histogramu.[3] Ovo dovodi do toga da de-lovi slike sa malim kontrastom dobiju znatno veći kontrast.

Implementacija samog algoritma se svodi na nekoliko koraka. Pre svega, uvedimo oznake: neka je n_i broj pojavljivanja sive boje nivoa i , L ukupan broj nivoa sive boje (uobičajeno 256) i n ukupan broj piksela na slici. Koraci su:

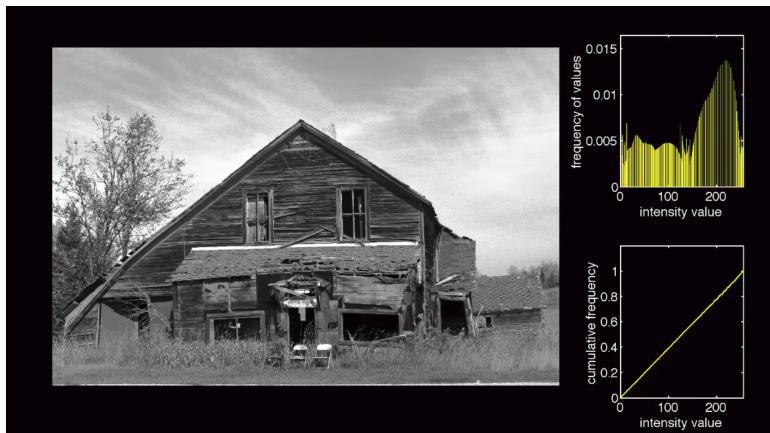
1. Računanje verovatnoće svakog piksela: $p_x(i) = p(x = i) = \frac{n_i}{n}$, $0 \leq i < L$
2. Računanje funkcije kumulativne raspodele: $cdf_x(i) = \sum_{j=0}^i p_x(j)$
3. Računanje novih vrednosti inteziteta za svaki od piksela pomoću formule: $h(v) = \text{round} \left(\frac{L-1}{n} \times cdf(v) \right)$

Ideja poslednje formule predstavlja želju da rezultujuća slika ima linearnu funkciju kumulativne raspodele, jer takvoj funkciji odgovara "poravnat" histogram slike.

Grafički prikaz:



Slika 3: Slika pre korekcije



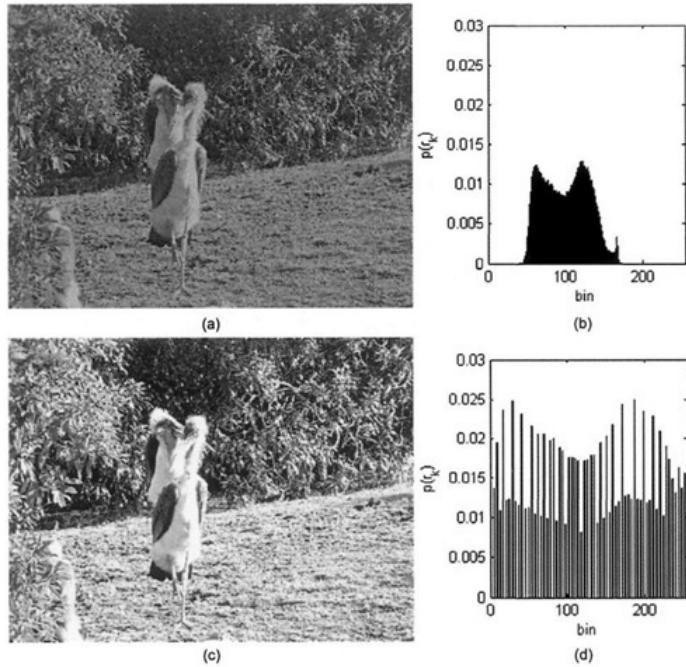
Slika 4: Slika nakon korekcije

```

1 def izracunaj_koef (slika_niz):
2
3     br_piksela = slika_niz.size
4     hist, _ = np.histogram (slika_niz, bins = maks_intezitet,
5                             range = (0, maks_intezitet))
6
7     izmenjeni = np.zeros (maks_intezitet + 1)
8
9     for i in range (len(izmenjeni)):
10         izmenjeni[i] = (maks_intezitet / br_piksela) * hist[: i
11             +1].sum()
12
13     izmenjeni = np.floor (izmenjeni)
14     return izmenjeni

```

Implementacija 4: Funkcija za računanje nove raspodele



Slika 5: Polazna slika i slika nakon korekcije

```

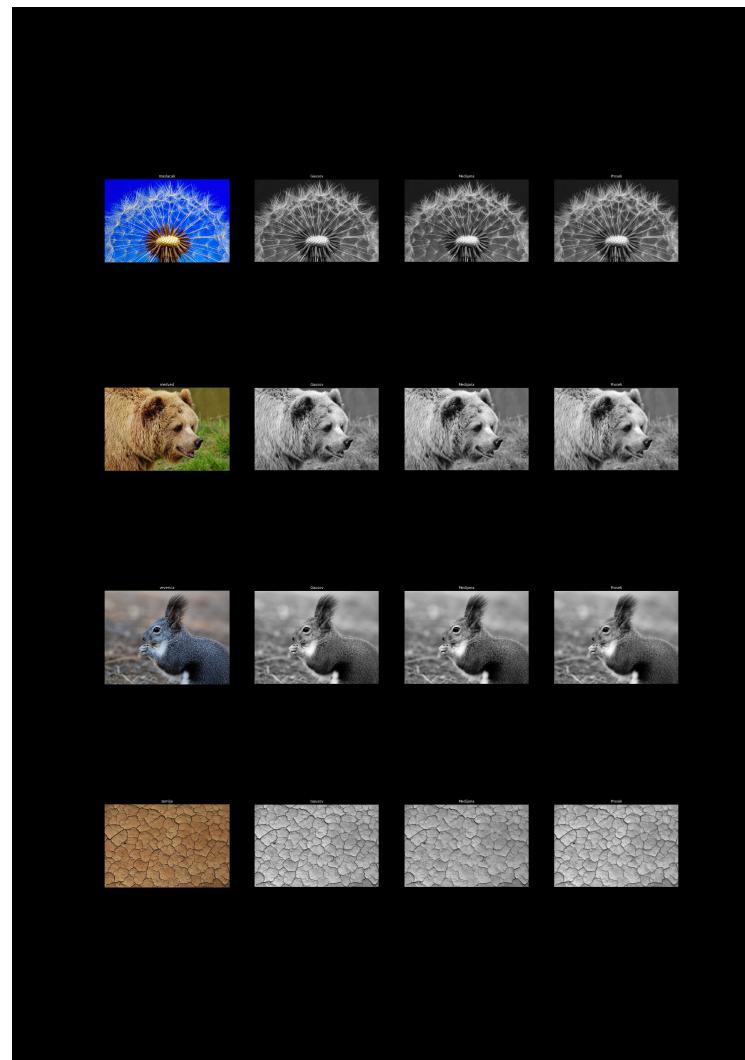
1 def Globalna (putanja_slike, ime_slike):
2
3     slika = img.open (os.path.join(putanja_slike, ime_slike + '.'
4                         jpg')).convert ('L')
5     slika_niz = np.uint8 (np.array (slika))
6
7     rez_niz = np.zeros (slika_niz.shape, np.uint8)
8
9     hist, _ = np.histogram (slika_niz, bins = maks_intezitet,
10                           range = (0, maks_intezitet))
11    napravi_Histogram (hist, ime_slike, 'Globalna_1')
12
13    tmp_niz = izracunaj_koef (slika_niz)
14    #napravi_Histogram (tmp_niz, ime_slike, 'KoefGlobalna')
15
16    for (x, y), vrednost in np.ndenumerate (slika_niz):
17        rez_niz [x, y] = tmp_niz[vrednost]
18
19    hist, _ = np.histogram (rez_niz, bins = maks_intezitet + 1,
20                           range = (0, maks_intezitet))
21    napravi_Histogram (hist, ime_slike, 'Globalna_2')
22
23    rez = img.fromarray (rez_niz)
24    rez.save (os.path.join ('Rezultati', 'Histogram', ime_slike +
25                         '_Globalna_0.jpg'))

```

Implementacija 5: Implementacija korektovanja slike

5 Rezultati projekta

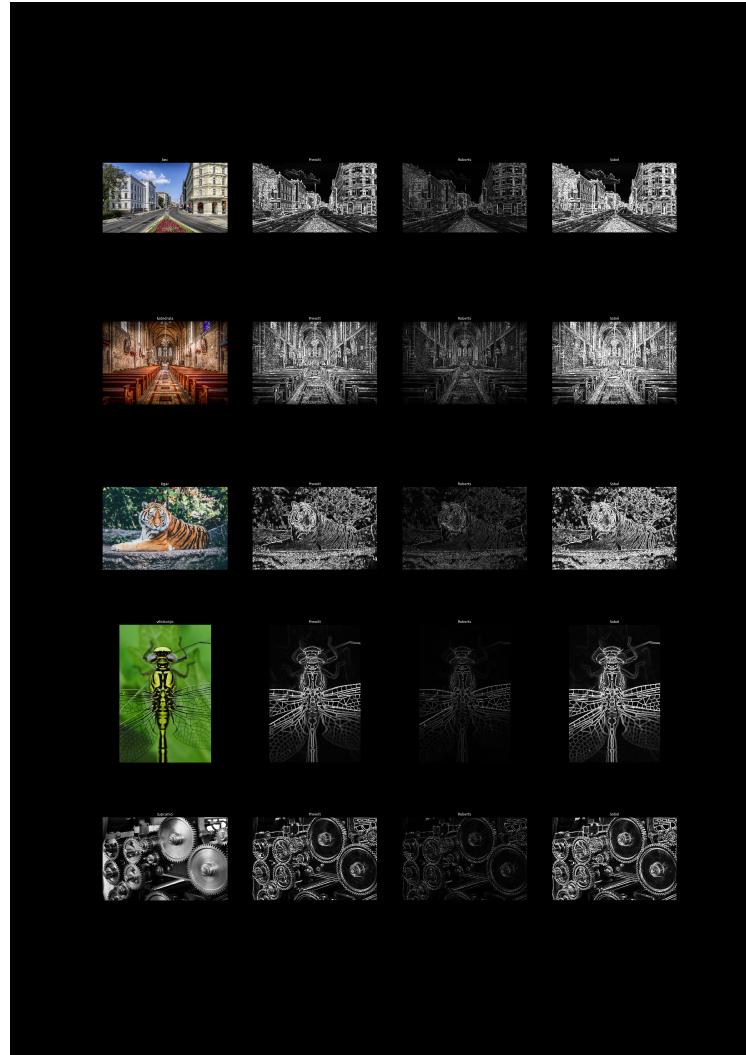
5.1 Zamućivanje



Slika 6: Zamućivanje

Link ka slici pune rezolucije: [LINK](#)

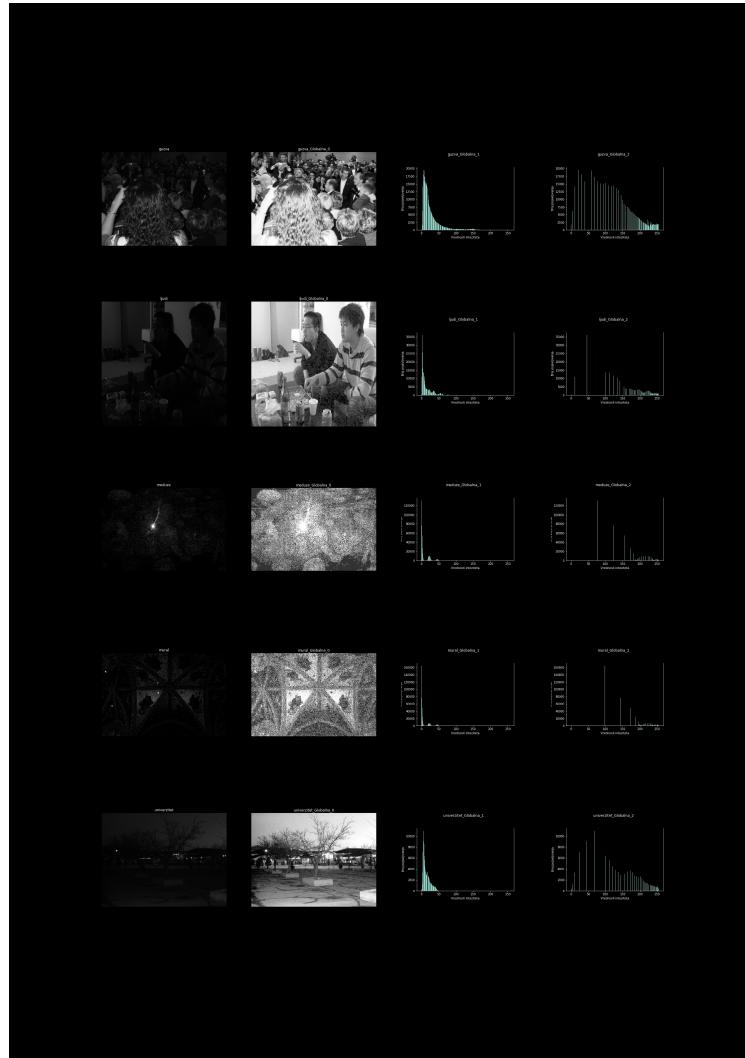
5.2 Izdvajanje ivica



Slika 7: Detektovanje ivica

Link ka slici pune rezolucije: [LINK](#)

5.3 Korekcija slika pomoću histograma inteziteta piskela



Slika 8: Korekcija slika

Link ka slici pune rezolucije: [LINK](#)

Literatura

- [1] Mladen Nikolić Andelka Zečević. *Mašinsko učenje*. Matematički fakultet, Univerzitet u Beogradu, 2019.
- [2] Studenti matematičkog fakulteta. Konvolucija. <http://alas.matf.bg.ac.rs/~af00195/link15.html>, 2000. [Online; accessed 24-jun-2019].
- [3] Wikipedia contributors. Histogram equalization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Histogram_equalization&oldid=897686591, 2019. [Online; accessed 24-June-2019].

Implementacije

1	Implementacija zamućivanja konvolucijom	4
2	Implementacija zamućivanja medijanom	5
3	Implementacija detekcije ivica konvolucijom	6
4	Funkcija za računanje nove raspodele	7
5	Implementacija korektovanja slike	8