Primjena diskretne Fourierove transformacije u obradi signala

Fatović, Mateo

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mateo Fatović

PRIMJENA DISKRETNE FOURIEROVE TRANSFORMACIJE U OBRADI SIGNALA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Ivica Nakić

Zagreb, veljača, 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem svojim roditeljima jer su uvijek vjerovali u mene i pružali mi podršku. Također se zahvaljujem svim kolegama i prijateljima na pomoći i podršci tijekom studiranja.

Sadržaj

Uvod

Digitalna obrada slika je postupak mijenjanja svojstava digitalnih slika. Podrazumijeva korištenje matematičkih alata za izvođenje operacija na digitalnim slikama i uključuje analizu i transformaciju slika kako bi se poboljšala njihova vizualna kvaliteta ili izvukle značajne informacije. Tehnike obrade slike naširoko se koriste u raznim područjima kao što su medicina, nadzor i zabava za poboljšanje kvalitete slike, otkrivanje značajki i izdvajanje podataka. U ovom radu objasnit ćemo kako se slike reprezentiraju u računalu, te pokazati kako se njima može jednostavno manipulirati koristeći matematičke alate i programski jezik python.

U prvom poglavlju definirat ćemo diskretnu Fourierovu trasformaciju (DCT) i diskretnu kosinusnu transformaciju (DCT), te pokazati način na koji ih možemo implementirati. Spomenut ćemo i filtere, koji su jedan od važnijih pojmova u obradi signala. U drugom poglavlju ćemo pokazati kako jednostavno možemo manipulirati svojstvima digitalnih slika koristeći matematičke operacije. Konkretnije, kako možemo iz slika u boji dobiti crno-bijele slike i negativ, te kako mijenjati kontrast. Pokazat ćemo i kako možemo koristiti operacije bazirane na filterima za zaglađivanje slika i detekciju rubova. Budući da ćemo definirati DFT i DCT kao promjene koordinata u vektorskim prostorima $\mathbb{C}^N$ i $\mathbb{R}^N$ nećemo ih moći dirketno koristiti na višedimenzionalnim objektima kao što su slike. U tu svrhu pokazat će se korisnim tenzorski produkt, koji možemo koristiti kako bi proširili promjenu koordinata na višedimenzionalno okruženje, tj. kako bi implementirali višedimenzionalni FFT i DCT pomoću odgovarajućih jednodimnzionalnih operacija. Na kraju ćemo pokazati i konkretnu primjenu tako implementiranih algoritama.

Poglavlje 1

DFT, DCT i filteri

1.1 Diskretna Fourierova transformacija

U ovom dijelu rada bavit ćemo se Fourierovom trasformacijom. Fourierova transformacija u obradi signala služi nam kao prijelaz iz vremenske reprezentacije signala (signal kao funkcija vremena) u frekvencijsku reprezentaciju signala (signal kao funkcija frekvencije). Fourierova analiza proizlazi iz ideje da svaku periodičku funkciju možemo zapisati kao (ne nužno konačnu) sumu sinusa (čistih tonova) različitih amplituda, faza i frekvencija. Tu sumu nazivamo Fourierov red. Mi ćemo opisati diskretnu Fourierovu transformaciju, a više o Fourierovim redovima i Fourierovoj transformaciji može se pronaći u [4] i [3].

Definicija 1.1.1 (Diskretna Fourierova Analiza)**.** *U diskretnoj Fourierovoj analizi, vektor* $\boldsymbol{x} = (x_0, x_0, ..., x_{N-1}) \in \mathbb{C}^N$ *predstavljen je kao linearna kombinacija N vektora*

$$\phi_n = \frac{1}{\sqrt{N}}(1, e^{2\pi in/N}, e^{2\pi i 2n/N}, \ldots, e^{2\pi ikn/N}, \ldots, e^{2\pi i(N-1)n/N})$$

Kolekcija $\mathcal{F}_N = \{\phi_n\}_{n=0}^{N-1}$ *naziva se* ***Fourierova baza****. Elementi baze nazivamo normalizirane kompleksne eksponencijalne funkcije, ili čisti tonovi reda N.*

Gornja definicija zapravo kaže da vektor $x \in \mathbb{C}^N$ možemo promatrati kao linearnu kombinaciju vektora iz $\mathcal{F}_N$. Dokaz sljedeće leme možete pronaći u [3].

Lema 1.1.2. *Fourierova baza* $\mathcal{F}_N = \{\phi_n\}_{n=0}^{N-1}$ *je ortonormirana baza za* $\mathbb{C}^N$.

Glavna ideja diskretne Fourierove analize je promjeniti koordinate iz standardne baze u Fourierovu bazu, napraviti neke operacije na koordinatama, te vratiti se nazad u standardnu bazu. Za to su nam korisne matrice prijelaza iz baze u bazu.

Definicija 1.1.3. *Sa F_N ćemo označavati matricu prijelaza iz standardne baze za $\mathbb{C}^N$ u Fourierovu bazu $\mathcal{F}_N$ i zvat ćemo je* ***Fourierova matrica****.*

Matricu $\sqrt{N}F_N$ zovemo diskretna Fourierova transformacija(DFT). Ako je $\boldsymbol{x} \in \mathbb{C}^N$, onda elemente vektora $y = DFT\boldsymbol{x}$ nazivamo DFT koeficijenti od $\boldsymbol{x}$.

Razlog zašto smo različito definirali DFT i Fourierovu matricu leži u tome što je Fourierova matrica ima veću primjenu u čistoj matematici, dok DFT ima veću primjenu kod obrade signala. Pokažimo sada kako možemo pronaći izraz za F_N. Koristit ćemo neke tvrdnje iz linearne algebre koje možemo pronaći u [1].

S obzirom da je ${F_N}^{-1}$ matrica prijelaza iz Fourierove baze u standardnu bazu, vektori ϕ_n su stupci matrice ${F_N}^{-1}$. Lema 1.1.2 kaže da su ti vektori ortonormirani, takvi da je $F_N = (F_N^{-1})^{-1} = (F_N^{-1})^*$, gdje je $A^* = (\overline{A})^T$ kompleksno konjugirana i transponirana matrica matrice A. Slijedi da su retci matrice F_N konjugirani vektori Fourierove baze, te da je $(F_N)^{-1} = (F_N)^*$. To znači da je F_N unitarna matrica, pa smo dokazali smo sljedeći teorem.

Teorem 1.1.4. *Fourierova matrica F_N je unitarna matrica dimenzija $N \times N$ čiji su elementi dani sa:*

$$(F_N)_{nk} = \frac{1}{\sqrt{N}} e^{-2\pi ink/N},$$

za $0 \leq n, k \leq N$.

Primjetimo da, zbog $(F_N)^T = F_N$, vrijedi ${F_N}^{-1} = \overline{F_N}$. Sada vidimo da je lako naći inverz matrice F_N, kao i DFT.

Definicija 1.1.5 (Inverzni DFT)**.** *Matrica $\overline{F}_N/\sqrt{N}$ je inverz od DFT, tj. matrice $\sqrt{N}F_N$. Nazivamo ju inverzna diskretna Fourierova transformacija, ili IDFT.*

U obradi signala puno je češći zapis DFT-a po komponentama. Ako imamo da je $\boldsymbol{y} = DFT\boldsymbol{x}$ i $\boldsymbol{x} = IDFT\boldsymbol{y}$ to možemo zapisati po komponentama kao

$$y_n = \sum_{k=0}^{N-1} x_k e^{-2\pi ink/N} \qquad\qquad x_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{2\pi ink/N} \tag{1.1}$$

Iz ovakvog zapisa vidimo da je lako direktno implementirati algoritme za DFT i IDFT, no s obzirom da množenje matrice dimenzija $N \times N$ i vektora zahtjeva ugrubo N^2 aritmetičkih operacija, tako implementirani algoritmi će biti jako spori za veliki N. Zato ćemo pokazati puno efikasniji način za implementaciju koji nazivamo *Brza Fourierova transformacija(FFT).*

Brza Fourierova trasformacija (FFT)

Kao što smo rekli, direktna implementacija DFT zahtjeva velik broj aritmetičkih operacija pa nam neće biti od velike koristi ako radimo s jako velikim skupovima podataka, što je u obradi signala jako čest slučaj. Zato sada navodimo efikasniju implementaciju koju nazivamo *Brza Fourierova transformacija*. Pretpostavljamo da je $\boldsymbol{x} \in \mathbb{C}^N$ i da je N potencija broja 2. To možemo pretpostaviti s obzirom da u primjenama često ne radimo na cijelom ulaznom podatku nego ga dijelimo u blokove manje duljine, za koju možemo odabrati potenciju broja 2. Ideja algoritma je koristiti metodu *podijeli pa vladaj* kako bi računanje matrice DFT za vektor duljine N sveli na računanje dvije martice vektora duljine $N/2$. Sada navodimo FFT i IFFT algoritam za paran N.

Teorem 1.1.6. *Neka je $\boldsymbol{y} = DFT_N\boldsymbol{x}$, te neka je N paran i neka martica $D_{N/2}$ označava dijagonalnu $(N/2) \times (N/2)$ matricu s vrijednostima $(D_{N/2})_{n,n} = e^{-2\pi in/N}$, za $0 \leq n < N/2$. Tada vrijedi:*

$$(y_0, y_1, \ldots, y_{N/2-1}) = DFT_{N/2}\boldsymbol{x}^{(e)} + D_{N/2}DFT_{N/2}\boldsymbol{x}^{(o)} \tag{1.2}$$

$$(y_{N/2}, y_{N/2+1}, \ldots, y_{N-1}) = DFT_{N/2}\boldsymbol{x}^{(e)} - D_{N/2}DFT_{N/2}\boldsymbol{x}^{(o)} \tag{1.3}$$

Gdje se $\boldsymbol{x}^{(e)}$ i $\boldsymbol{x}^{(o)} \in \mathbb{R}^{N/2}$ sastoje od vrijednosti na parnim i neparnim indeksima u $\boldsymbol{x}$, tj.

$$\boldsymbol{x}^{(e)} = (x_0, x_2, \ldots, x_{N-2}) \qquad \boldsymbol{x}^{(o)} = (x_1, x_3, \ldots, x_{N-1})$$

Dokaz. Pretpostavimo prvo $0 \leq n \leq N/2 - 1$ i podijelimo sumu u izrazu za DFT na parne i neparne indekse,

$$\begin{aligned} y_n &= \sum_{k=0}^{N-1} x_k e^{-2\pi ink/N} = \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi in2k/N} + \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi in(2k+1)/N} \\ &= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi ink/(N/2)} + e^{-2\pi in/N} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi ink/(N/2)} \\ &= (DFT_{N/2}\mathbf{x}^{(e)})_n + e^{-2\pi in/N}(DFT_{N/2}\mathbf{x}^{(o)})_n \end{aligned}$$

gdje su $\mathbf{x}^{(e)}$ i $\mathbf{x}^{(o)}$ zadani kao i u tvrdnji teorema. Sada vidimo da smo dobili jednadžbu (1.2). Za drugu polovicu koeficijenata imamo:

$$\begin{aligned} y_{N/2+n} &= \sum_{k=0}^{N-1} x_k e^{-2\pi i(N/2+n)k/N} = \sum_{k=0}^{N-1} x_k e^{-\pi ik} e^{-2\pi in2k/N} \\ &= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi in2k/N} - \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi in(2k+1)/N} \end{aligned}$$

$$= \sum_{k=0}^{N/2-1} x_{2k} e^{-2\pi i n k/(N/2)} - e^{-2\pi i n/N} \sum_{k=0}^{N/2-1} x_{2k+1} e^{-2\pi i n k/(N/2)}$$

$$= (DFT_{N/2}\mathbf{x}^{(e)})_n - e^{-2\pi i n/N}(DFT_{N/2}\mathbf{x}^{(o)})_n$$

iz čega slijedi jednadžba (1.3)

□

Teorem 1.1.7. *Neka je N paran broj i* $\tilde{\boldsymbol{x}} = \overline{DFT_N}\boldsymbol{y}$ *i* $D_{N/2}$ *kao gore. Tada vrijedi:*

$$(\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_{N/2-1}) = \overline{DFT_{N/2}}\boldsymbol{y}^{(e)} + D_{N/2}\overline{DFT_{N/2}}\boldsymbol{y}^{(o)} \tag{1.4}$$

$$(\tilde{x}_{N/2}, \tilde{x}_{N/2+1}, \ldots, \tilde{x}_{N-1}) = \overline{DFT_{N/2}}\boldsymbol{y}^{(e)} + D_{N/2}\overline{DFT_{N/2}}\boldsymbol{y}^{(o)} \tag{1.5}$$

Gdje su $\boldsymbol{y}^{(e)}$ *i* $\boldsymbol{y}^{(o)} \in \mathbb{R}^{N/2}$ *dani sa:*

$$\boldsymbol{y}^{(e)} = (y_0, y_2, \ldots, y_{N-2}) \qquad \boldsymbol{y}^{(o)} = (y_1, y_3, \ldots, y_{N-1})$$

Štoviše, $\boldsymbol{x} = IDFT_N\boldsymbol{y}$ *se može izračunati iz* $\boldsymbol{x} = \tilde{\boldsymbol{x}}/N = \overline{DFT_N}\boldsymbol{y}/N$.

Sada možemo vidjeti jedan način implementacije gore navedenih algoritama.

```
def fft_impl(x, f, forward = True):
    if ndim(x) == 1:
        bit_reversal(x)
        f(x, forward)
    else:
        bit_reversal_arr(x)
        for s2 in range(shape(x)[1]):
            f(x[:, s2], forward)
    if not forward:
        x /= len(x)
```

```
def fft_kernel_standard(x, forward):
    N = len(x)
    sign = -1
    if not forward:
        sign = 1
    if N > 1:
        xe, xo = x[0:(int(N/2))], x[(int(N/2)):]
        fft_kernel_standard(xe, forward)
        fft_kernel_standard(xo, forward)
        D = exp(sign*2*pi*1j*arange(float(N/2))/N)
        xo *= D
        x[:] = concatenate([xe + xo, xe - xo])
```

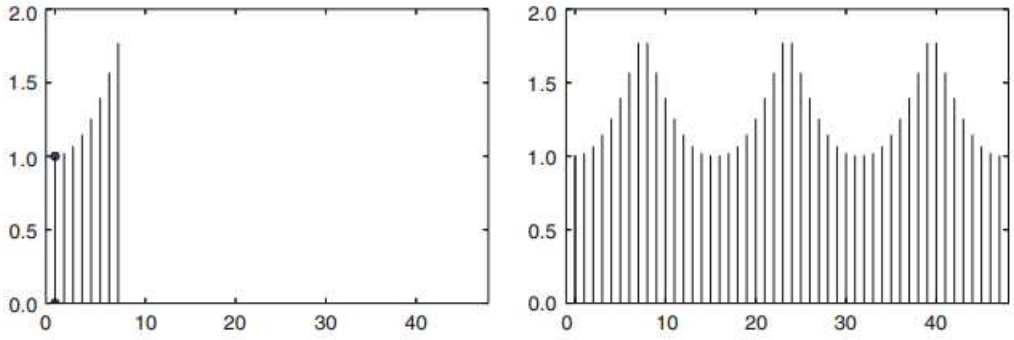
Implementacija je podijeljena na dvije funkcije tako da funkcija fft_impl provjerava dimenziju od $\boldsymbol{x}$, te ovisno o tome poziva odgovarajuću verziju funkcije bit_reversal koja se brine da ulazni podatak bude u redosljedu pogodnom za rad algoritma, to jest, da se elementi na parnim indeksima nalaze na prvih $N/2$ mjesta, a oni na neparnima na sljedećih $N/2$ mjesta. Ideja funkcije bit_reversal je okrenuti poredak bitova u indeksima orginalnog niza, kako bi parni indeksi bili na prvih $N/2$ mjesta. Funkcija fft_kernel_standard pretpostavlja da je ulazni podatak u odgovarajućem poretku pa stvara dva polja duljina $N/2$ te se poziva rekurzivno s njima. Parametar **forward** određuje hoće li se pozvati algoritam za FFT(**forward** = True) ili IFFT(**forward** = False). Množenje s $1/N$ u slučaju IFFT riješeno je u funkciji *fft_impl*. Razlog podjele na dvije funkcije je taj što je *bit_reversal* operacija korisna za razne implementacije FFT algoritma, kao na primjer, iterativni FFT ili split-radix FFT. Više o tim implementacijama i funkciji *bit_reversal* može se pronaći u [3].

Diskretna kosinusna transformacija(DCT)

Sada ćemo ukratko objasniti i diskretnu kosinusnu transformaciju koja se u digitalnoj obradi slika primjenjuje češće od diskretne Fourierove transformacije. Ideja dolazi of Fourierovih redova, gdje je pokazano da je Fourierov red simetričnog proširenja funckije f bolja aproksimacija za f na segmentu $[0, T]$ od Fourierovog reda od f. Podsjetimo, koeficijenti od DFT su bili povezani s koeficijentima Fourierovog reda periodički proširenih vektora, dok su koeficijenti od DCT povezani s koeficijentima Fourierovog reda simetričnih periodički proširenikh vektora. Tu vezu ćemo iskoristiti pa ćemo tako navesti alogoritam za DCT koji će koristiti FFT algortiam od ranije.

Definicija 1.1.8. *Neka je $\boldsymbol{x} \in \mathbb{R}^N$. Vektor $\breve{\boldsymbol{x}} \in \mathbb{R}^{2N}$ naziva se simetrično proširenje vektora $\boldsymbol{x}$ i definira kao:*

$$\breve{\boldsymbol{x}}_k = \begin{cases} x_k & , 0 \leq k < N \\ x_{2N-1-k} & , N \leq k < 2N-1 \end{cases}$$



Slika 1.1: Primjer vektora i njegovog simetričnog proširenja

Kao što vidimo na slici 1.1, podrazumjevamo da se vektor $\check{\mathbf{x}}$ periodično ponavlja, tj. prvo "zrcalimo" vektor $\mathbf{x}$ da dobijemo $\check{\mathbf{x}}$, a onda ga periodički ponavljamo. Sada navodimo nekoliko teorema i definicija vezanih za DCT, dokazi i više o DCT se može pronaći u [3].

Teorem 1.1.9.

$$\left\{ \frac{1}{\sqrt{2N}} \cos\left(\frac{2\pi\cdot 0(k+1/2)}{2N}\right), \left\{ \frac{1}{\sqrt{N}} \cos\left(\frac{2\pi n(k+1/2)}{2N}\right)\right\}_{n=1}^{N-1} \right\} \tag{1.6}$$

je ortonormirana baza za simetrična proširenja u $\mathbb{R}^{2N}$

Teorem 1.1.10.

$$\left\{ \frac{1}{\sqrt{N}} \cos\left(\frac{2\pi\cdot 0(k+1/2)}{2N}\right), \left\{ \sqrt{\frac{2}{N}} \cos\left(\frac{2\pi n(k+1/2)}{2N}\right)\right\}_{n=1}^{N-1} \right\} \tag{1.7}$$

je ortonormirana baza za $\mathbb{R}^N$*. Označavat ćemo ju sa* $\mathcal{D}_N$*, a vektore sa* $\boldsymbol{d}_n$*.*

Definicija 1.1.11 (Diskretna kosinusna transformacija)**.** *Promjena koodinata iz standardne baze za* $\mathbb{R}^N$ *u bazu* $\mathcal{D}_N$ *naziva se diskretna kosinusna transformacija (kraće, DCT). Odgovarajuća* $N \times N$ *matrica prijelaza,* DCT_N*, naziva se DCT matrica. Ako je vektor* $\boldsymbol{x} \in \mathbb{R}^N$*, tada njegove koordinate* $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1})$ *u bazi* $\mathcal{D}_N$ *zovemo DCT koeficijenti od* $\boldsymbol{x}$*(to jest, vrijedi* $\boldsymbol{y} = DCT_N\boldsymbol{x}$*). Vektor* $\boldsymbol{x} = (DCT_N)^T\boldsymbol{y}$ *zovemo inverzna dikretna kosinusna transformacija ili IDCT od* $\boldsymbol{x}$*.*

Sljedeći teoremi nam daju vezu između DCT i DFT.

Napomena 1.1.12. *Matricu C_N danu s $(C_N)_{n,k} = \cos\left(2\pi\frac{n}{2N}\left(k+\frac{1}{2}\right)\right)$ nazivamo kosinusna matrica(to je zapravo DCT matrica gdje retci nisu skalirani). Sljedeći teoremi će biti iskazani u terminima matrice C_N jer će tako biti praktičnije definirati algoritme, a kada računamo DCT treba samo skalirati na kraju. Dokazi sljedećih teorema se nalaze u [3].*

Teorem 1.1.13. *Neka je $\boldsymbol{y} = C_N\boldsymbol{x}$. Tada vrijedi:*

$$y_n = \left(\cos\left(\pi\frac{n}{2N}\right)\Re\left(\left(DFT_N\boldsymbol{x}^{(1)}\right)_n\right) + \sin\left(\pi\frac{n}{2N}\right)\Im\left(\left(DFT_N\boldsymbol{x}^{(1)}\right)_n\right)\right), \tag{1.8}$$

gdje je $\boldsymbol{x}^{(1)} \in \mathbb{R}^N$ definiran s:

$$(\boldsymbol{x}^{(1)})_k = x_{2k},\ za\ 0 \le k \le N/2-1$$

$$(\boldsymbol{x}^{(1)})_{N-k-1} = x_{2k+1},\ za\ 0 \le k \le N/2-1$$

Napomena 1.1.14. *Sa Q_N ćemo označititi dijagonalnu $N \times N$ matricu, definiranu s $Q_{n,n} = e^{-\pi in/(2N)}$.*

Teorem 1.1.15 (IDCT algoritam)**.** *Neka je $\boldsymbol{x} = (C_N)^{-1}\boldsymbol{y}$, te neka je $\boldsymbol{z}$ vektor takav da mu je na mjestu s indeksom 0 element $y_0/Q_{0,0}$, a na ostali elementi su dani s $(y_n - iy_{N-n})/Q_{n,n}$. Tada vrijedi :*

$$\boldsymbol{x}^{(1)} = IDFT_N\boldsymbol{z},$$

gdje je $\boldsymbol{x}^{(1)}$ definiran kao i u teoremu 1.1.13.

Sada navodimo implemenacije za DCT i IDCT [3].

```
def dct_impl(x):
    N = len(x)
    if N > 1:
        x1 = concatenate([x[0::2], x[-1:0:-2]]).astype(complex)
        fft_impl(x1, fft_kernel_standard)
        cosvec = cos(pi*arange(float(N))/(2*N))
        sinvec = sin(pi*arange(float(N))/(2*N))
        if ndim(x) == 1:
            x[:] = cosvec*real(x1) + sinvec*imag(x1)
        else:
            for s2 in range(shape(x)[1]):
                x[:, s2] = cosvec*real(x1[:, s2]) \
                + sinvec*imag(x1[:, s2])
        x[0] *= sqrt(1/float(N))
        x[1:] *= sqrt(2/float(N))
```

```
def idct_impl(y):
    N = len(y)
    if N > 1:
        y[0] /= sqrt(1/float(N))
        y[1:] /= sqrt(2/float(N))
        Q = exp(-pi*1j*arange(float(N))/(2*N))
        y1 = zeros_like(y).astype(complex)
        y1[0] = y[0]/Q[0]
        if ndim(y) == 1:
            y1[1:] = (y[1:] - 1j*y[-1:0:-1])/Q[1:]
        else:
            for s2 in range(shape(y)[1]):
                y1[1:, s2] = (y[1:, s2] - 1j*y[-1:0:-1, s2])/Q[1:]
        fft_impl(y1, fft_kernel_standard, 0)
        y[0::2] = real(y1[0:(int(N/2))])
        y[1::2] = real(y1[-1:(int(N/2)-1):-1])
```

1.2 Filteri

U ovom dijelu rada objasnit ćemo ukratko pojam filtera, te ćemo navesti implementaciju funkcije `filter_impl` koja će nam biti korisna u primjerima. Filteri su korisni jer pomoću njih možemo razdvajati i obnavljati signale, npr. možemo ukloniti nepotrebnu buku ili obnoviti signal snimljen lošijom opremom. Možemo reći da su filteri linearne transformacije koje „čuvaju" frekvencije.

Definicija 1.2.1. *Linearna transformacija s se naziva filter, ako za bilo koju frekvenciju $e^{2\pi ivt}$ vrijedi*

$$s\left(e^{2\pi ivt}\right) = \lambda_s(v)e^{2\pi ivt}$$

za neku funkciju λ_s. λ_s se naziva frekvencijski odziv od s, a $\lambda_s(v)$ opisuje kako filter s djeluje na frekvenciju v.

U kontekstu frekvencijske domene, imamo četiri osnovna tipa filtera: niskopropusni, visokopropusni, pojasnopropusni i pojasnonepropusni filteri. Raspon frekvencija koje filter propušta naziva se propusni pojas. Mi ćemo filtere promatrati u kontekstu vremenske domene. Dva najbitnija pojma vezana uz filtere u vremenskoj domeni su impulsni odziv i konvolucija. Filtere možemo konstruirati konvolucijom nekog ulazog signala s nekim određenim impulsnim odzivom.

Definicija 1.2.2. *Konvolucija vektora $\boldsymbol{t}$ i $\boldsymbol{x}$, u oznaci $\boldsymbol{t} * \boldsymbol{x}$, označava vektor $\boldsymbol{z}$, dan po komponentama s:*

$$z_n = \sum_k t_k x_{n-k}. \tag{1.9}$$

t_k se nazivaju koeficijenti filtera.

Svaki filter se može zapisati kao konvolucija, a svaku konvoluciju možemo promatrati kao beskonačnu Toeplitz matricu. Matricu nazivamo Toeplitz matrica ako su joj elementi na svim dijagonalama jednaki. Filter (1.9) možemo predstaviti matricom

$$S = \begin{pmatrix} \ddots & \cdots & \cdots & \cdots & \ddots \\ \vdots & t_0 & t_{-1} & t_{-2} & \vdots \\ \vdots & t_{-1} & \underline{t_0} & t_1 & \vdots \\ \vdots & t_{-2} & t_{-1} & t_0 & \vdots \\ \ddots & \cdots & \cdots & \cdots & \ddots \end{pmatrix}$$

Za filter (1.9) koristit ćemo kompaktnu notaciju

$$S = \{\ldots, t_{-2}, t_{-1}, \underline{t_0}, t_1, t_2, \ldots\},$$

gdje naglašavamo element t_0. Broj koeficijenata t_k koji nisu nula može biti i beskonačan, no u našim primjerima ćemo imati samo konačan broj koeficijenata t_k koji nisu nula. Filteri s tim svojstvom nazivaju se FIR filteri(eng. Finite Impulse Response filters). U obradi signala vektor $\mathbf{t}$ naziva se *impulsni odziv*. Ako je S FIR filter i k_{min} i k_{max} najmanji i najveći k tako da je $t_k \neq 0$, onda pišemo i

$$S = \{t_{k_{min}}, \ldots, t_{-1}, \underline{t_0}, t_1, \ldots, t_{k_{max}}\}.$$

Filtere je puno jednostavnije računati pomoću konvolucija nego da prvo konstruiramo matricu pa obavimo matrično množenje.

Sada navodimo implmentaciju funkcije `filter_impl` koju ćemo koristiti kasnije za implementaciju potrebnih filtera.

```
def filter_impl(t, x, bd_mode):
    szx = shape(x)
    N = szx[0]
    n = int(prod(szx[1:]))
    y =  reshape(x, (N, n))
    tlen = len(t); N0 = int((tlen - 1)/2)
    w = 0

    if bd_mode.lower() == 'symm':
        w = concatenate([ y[N0:0:(-1)], y, y[(N-2):(N - N0 - 2):(-1)] ])
    elif bd_mode.lower() == 'per':
        w = concatenate([ y[(N - N0):], y, y[:N0]])
    elif bd_mode.lower() == 'none' or bd_mode.lower() == 'bd':
```

```
        w = concatenate( (zeros(N0, n), y, zeros(N0, n)) )
    for k in range(n):
        z = convolve(t, w[:, k])
        y[:, k] = z[(2*N0):(len(z)-2*N0)]
    x[:] = reshape(y, szx)
```

Pretpostavljamo da je duljina vektora koeficijenata filtera `t` neparna i da se koeficijent t_0 nalazi na sredini vektora. Parametar `bd_mode` nam govori kako će se filter ponašati na granicama vektora x. Ako je parametar jednak 'symm', proširujemo vektor x simetrično, a ako je parametar jednak 'per' proširujemo ga periodično. Ako parametar nije naveden, proširujemo vektor nulama. Nakon toga koristimo funkciju `convolve` kako bi izračunali konvoluciju. Funkcija `reshape` nam koristi u slučaju da x ima više od dvije dimenzije, što će biti slučaj kod slika u RGB formatu. U tom slučaju prvo 'spljoštimo' x na dvodimenzionalan objekt, izračunamo konvoluciju, te vratimo x u orginalni oblik.

Poglavlje 2

Digitalne slike

O ovom poglavlju ćemo se baviti digitalnim slikama te ćemo pokazati kako njima možemo manipulirati, prvo koristeći neke jednostavne operacije na matricama, a nakon toga i koristeći dvodimenzonalne analogone za filtere, DFT i DCT.

2.1 Uvod i primjeri

Digitalna slika P je matematički opisana kao matrica vrijednosti intenziteta $\left\{p_{i,j}\right\}_{i,j=1}^{M,N}$. Vrijednost $p_{i,j}$ daje informacije o boji na tom mjestu. Kod crno-bijelih i dvobojnih slika $p_{i,j}$ je jedan broj, dok je kod slika u boji $p_{i,j}$ vektor s tri ili više elemenata. Za slike u boji postoji mnogo modela a mi ćemo koristiti rgb-model. Konkretno, u rgb-modelu $p_{i,j}$ je vektor s tri elementa:

$$p_{i,j} = \left(r_{i,j}, g_{i,j}, b_{i,j}\right),$$

gdje su $r_{i,j}, g_{i,j}, b_{i,j}$ redom količine crvene, zelene i plave na mjestu (i, j) te poprimaju vrijednosti između 0 i 255. Koordinate (i, j) imaju isto značenje kao i kod matrica, tj. i će označavati indeks retka, dok će j označavati indeks stupca. Dimenzija ovako definirane slike je $M \times N$. Slike u boji možemo promatrati i kao tri matrice, gdje jedna matrica predstaljvlja intenzitete crvene, druga intenzitete zelene i treća intenzitete plave boje. U primjerima će se implementirane operacije istovremeno zasebno izvšavati za svaku komponentu boje, tj. zasebno na svakoj od tri matrice.

Slika 2.1: Slika koju ćemo koristiti u primjerima

Sada ćemo pokazati kako dobiti crno-bijelu sliku, negativ te kako možemo namještati kontrast kod slika. Za učitavanje, spremanje i prikazivanje slika koristit ćemo sljedeći kod:

```
X = double(imread('filename.fmt', 'fmt'))
imwrite(uint8(X), 'filename.fmt', 'fmt')
imshow(uint8(X))
```

Umjesto 'filename.fmt' unosimo putanju do slike, a 'fmt' je opcionalni parametar koji daje format slike ('jpg','png','gif', itd.). Funkcija `imread` učitava vrijednosti pixela u matricu $\mathbf{X}$. Te vrijednosti su cijelobrojne (`uint8`), a budući da operacije s matricama očekuju tip `double`, potrebna je pretvorba tipova. Sa `imwrite` spremamo sliku reprezentiranu matricom $\mathbf{X}$. Ponovo je potrebna pretvorba tipova. Funkcija `imshow` prikazuje sliku, te također očekuje tip `uint8`.

Rekli smo da su inteziteti cijelobrojne vrijednosti između 0 i 255, no kada budemo izvršavali operacije na slikama, pretpostavljat ćemo da su vrijednosti realni brojevi iz $[0, 1]$. Zato ćemo trebati preslikavati između $[0, 255]$ i $[0, 1]$, dijeljenjem i množenjem s 255. Također, ponekad ćemo kao rezultat operacija dobiti rezultate izvan segmenta $[0, 1]$ pa ćemo trebati nekako normalizirati vrijednosti da budu u $[0, 1]$. Za to ćemo koristiti funkciju

$$g(x) = \frac{x - a}{b - a}, \qquad a < b,$$

koja preslikava segment $[a, b]$ u $[0, 1]$. Za a i b možemo uzeti minimalnu i maksimalnu vrijednost inteziteta. Sljedeća funkciju ćemo koristi u daljnim primjerima kako bi preslikali vrijednosti nazad u $[0, 1]$.

```
def map_to_01(X):
    minval, maxval = X.min(), X.max()
    X -= minval
    X /= (maxval-minval)
```

Kada učitamo sliku u boji, vraćeni objekt $\mathbf{X}$ ima tri dimenzije, prve dvije predstavljaju retke i stupce dok treća dimenzija daje informacije o boji na tom mjestu. Ako sada uzmemo samo određene elemente iz treće dimenzije možemo dobiti različite komponente boje.

```
img = double(imread('images/lena.png'))

X1 = zeros_like(img)
X1[:, :, 0] = img[:, :, 0]
X2 = zeros_like(img)
X2[:, :, 1] = img[:, :, 1]
X3 = zeros_like(img)
X3[:, :, 2] = img[:, :, 2]
```

Slika 2.2: Crvena, zelena i plava komponenta slike 2.1

Za dobiti sliku u sivim nijansama iz slike u boji trebamo vektor s tri elementa (r, g, b) zamijeniti s jednom vrijednosti q. To možemo na više načina, npr uzeti najveću od tri vrijednosti, njihovu aritmatičku sredinu, ili normu vektora boje (r, g, b), tj. uzmemo jedno od sljedećeg:

- $q_{i,j} = \max(r_{i,j}, g_{i,j}, b_{i,j})$
- $q_{i,j} = \left(r_{i,j} + g_{i,j} + b_{i,j}\right)/3$
- $q_{i,j} = \sqrt{r_{i,j}^2 + g_{i,j}^2 + b_{i,j}^2}$

```
img = double(imread('images/lena.png'))

mx = maximum(img[:, :, 0], img[:, :, 1])
X1 = maximum(img[:, :, 2], mx)

X2 = (img[:, :, 0] + img[ :, :, 1] + img[ :, :, 2])/3.

X3 = sqrt(img[:,:,0]**2 + img[:, :, 1]**2 + img[:, :, 2]**2)
map_to_01(X3)
X3 *= 255
```

Kod trećeg slučaja treba paziti jer dobivene vrijednosti mogu biti izvan dozvoljenog intervala pa treba normalizirati vrijednosti. Slike dobivene ovim operacijama su prikazane dolje.

Slika 2.3: Različiti načini za dobiti sliku 2.1 u sivim nijansama

Sada smo dobili slike u sivim nijansama reprezentirane matricama $X1, X2, X3$. Da bi dobili negativ potrebno je samo zamijeniti vrijednost inteziteta p s "zrcalnom" vrijednosti $255 - p$, tj. matrice $X1, X2, X3$ matricama $255 - X1, 255 - X2, 255 - X3$ Tako dobivamo sljedeće slike.

Slika 2.4: Negativi slika sa slike 2.3

Za dobiti crno bijele slike iz onih na slici 2.3 potrebno je samo zamijeniti vrijednosti u matrici s 0 ili 255. To možemo tako da postavimo vrijednost elementa na bližu od te dvije, tj. sa sljedećim kodom:

```
X1cb = 255*(X1 >= 128)
X2cb = 255*(X2 >= 128)
X3cb = 255*(X3 >= 128)
```

Tako dobivamo sljedeće slike.

Slika 2.5: Crno bijele verzije slika iz 2.3

Još jedan jednostavan primjer manipulacije slikom je mijenjanje kontrasta. Kontrast najčešće nije dobar zbog toga što se veliki udio sivih vrijednosti nalazi u malom podintervalu od [0, 1], pa te vrijednosti treba nekako raspršiti. Možemo koristiti monotonu funkciju f koja ima veliku derivaciju na mjestima gdje se nalazi velika koncentracija vrijednosti. Mi

ćemo koristiti familije funkcija f_n i g_ϵ definiranih s:

$$f_n(x) = \frac{\operatorname{arctg}(n(x-1/2))}{2\operatorname{arctg}(n/2)} + \frac{1}{2}$$

$$g_\epsilon(x) = \frac{\ln(x+\epsilon) - \ln\epsilon}{\ln(1+\epsilon) - \ln\epsilon}$$

Funkcije f_n imaju velike derivacije oko $x = 0.5$, pa će povećavati kontrast kod slika koje imaju veliku koncentraciju intenziteta oko 0.5, dok funkcije g_ϵ imaju velike derivacije oko $x = 0$ pa povećavaju kontrast kod slika koje imaju veliku koncentraciju malih vrijednosti intenziteta, npr jako tamne slike. Sad možemo vidjeti implementacije tih funkcija i primjenu na slici 2.1.

```
def contrast_adjust1(X,n):
    X /= 255.
    X -= 0.5
    X *= n
    arctan(X,X)
    X /= 2*arctan(n/2.)
    X += 0.5
    X*=225
    return X

def contrast_adjust2(X,epsilon):
    X /= 255.
    X += epsilon
    log(X, X)
    X -= log(epsilon)
    X /= (log(1+epsilon)-log(epsilon))
    X *= 255
    return X


img = double(imread('images/lena.png'))

X1 = contrast_adjust1(img, 10)
X2 = contrast_adjust2(img, 0.01)
```


Slika 2.6: Slike dobivene primjenom f_{10} i $g_{0.01}$ na sliku 2.1

S obzirom da su vrijednosti inteziteta na slici 2.1 dobro izbalansirane, funkcija f_{10} je dodatno potamnila tamnije dijelove, a posvijetlila svijetle, dok je funkcija $g_{0.01}$ posvijetlila cijelu sliku.

Pogledajmo sada još jedan primjer namještanja kontrasta. Koristit ćemo familiju funkcija h_n definiranih s:

$$h_n(x) = x^n.$$

Funkcija h_n preslikava segment $[0, 1]$ u $[0, 1]$ i vrijedi $h_n'(1) \to \infty$, kad $n \to \infty$. Iskoristit ćemo funkciju h_{100} da bi prikazali „skrivenu poruku" na slici 2.7.

Slika 2.7: Slika prije mijenjanja kontrasta

Korstimo sljedeći kod i dobivamo sliku 2.8, gdje je poruka vidljiva.

```
X = double(imread('images/secret.jpg'))
X = (X[:,:,0] + X[:,:,1]+ X[:,:,2])/3. #pretvaramo u sivu
map_to_01(X) #preslikvamo u segment [0,1]
X **= 100
X *= 255 #preslikvamo nazad u segment [0,255]
```

Slika 2.8: Slika nakon primjene funkcije h_{100}

2.2 Operacije bazirane na filterima

Sada ćemo pokazati kako možemo primjeniti neke operacije bazirane na filterima na obradu slika. Prvo ćemo definirati kako primjeniti filter na višedimenzionalne podatke, zatim ćemo objasniti kako možemo prikazati filtere koristeći tenzorski produkt te na kraju pokazati primjere zaglađivanja slike i detekcije rubova.

Definicija 2.2.1. *Kažemo da je operacija S na slici X dana računalnom molekulom (eng. computational molecule)*

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & a_{-1,-1} & a_{-1,0} & a_{-1,1} & \cdots \\ \cdots & a_{0,-1} & \underline{a_{0,0}} & a_{0,1} & \cdots \\ \cdots & a_{1,-1} & a_{1,0} & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

ako vrijedi:

$$(SX)_{i,j} = \sum_{k_1,k_2} a_{k_1,k_2} X_{i-k_1,j-k_2} \tag{2.1}$$

Indeksi u molekuli mogu biti i pozitivni i negativni. Podcrtani element s indeksom $(0,0)$ *naziva se centar molekule, a pretpostavljamo i da su svi ostali elementi koji nisu navedeni u molekuli jednaki 0.*

Može se dogoditi da indeksi $i - k_1$ i $j - k_2$ izađu izvan dozvoljenih indeksa u matrici X. Zato ćemo pretpostavljati da je matrica X nekako proširena u oba smjera, npr. simetrično ili periodično. Primjena molekula se svodi na to da postavimo centar molekule na piksel, pomnožimo piksel i susjedne elemente s odgovarajućim težinama $a_{i,j}$ u obrnutom poretku, te na kraju zbrojimo da bi dobili rezultat. Ovakav tip operacija će biti koristan pri obradi slika. Sljedeći teorem nam daje vezu između filtera i računalnih molekula.

Teorem 2.2.2. *Neka su S_1 i S_2 filteri s kompaktnim zapisima $\boldsymbol{t}_1$ i $\boldsymbol{t}_2$ i neka je S operacija koja prvo primjenjuje S_1 na stupce slike, a nakon toga primjenjuje S_2 na dobivene retke. Tada S možemo izraziti pomoću računalne molekule $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$*

Dokaz. Neka su $X_{i,j}$ pikseli slike. Kada primjenimo S_1 na stupce od X dobivamo sliku Y definiranu s:

$$Y_{i,j} = \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j}$$

Ako sada primjenimo S_2 na retke u Y, dobvamo sliku Z definiranu s:

$$Z_{i,j} = \sum_{k_2} (t_2)_{k_2} Y_{i,j-k_1} = \sum_{k_2} (t_2)_{k_2} \sum_{k_1} (t_1)_{k_1} X_{i-k_1,j-k_2}$$
$$= \sum_{k_1} \sum_{k_2} (t_1)_{k_1} (t_2)_{k_2} X_{i-k_1,j-k_2}$$

Ako sada suporedimo dobiveno s (2.1), vidimo da je operacija S dana računalnom molekulom s elementima $a_{i,j} = (\mathbf{t}_1)_i (\mathbf{t}_2)_j$.

□

Primjetimo da redosljed kojim filtriramo sa S_1 i S_2 nije bitan. Primjeniti S_1 na stupce u X je isto kao da računamo $S_1 X$, a primjeniti S_2 na retke u Y je isto kao i izračunati $Y(S_2)^T$. Operacija S tada ima oblik:

$$S(X) = S_1 X (S_2)^T$$

pa sada iz asocijativnosti matričnog množenja slijedi da redosljed filtriranja nije bitan. Primjenu S_1 na stupce od X nazivamo *verikalno filtriranje*, a primjenu S_2 na retke od X nazivamo *horizontalno filtriranje*.

Pogledajmo sada kako jednostavno implementirati transformaciju $S(X) = S_1 X (S_2)^T$. Prva tri parametra su `X`, S_1 i S_2, a parametar `bd_mode` kaže kako će se S_1 i S_2 ponašati oko granica. Korisitmo funkciju `transpose` kako bi zamijenili prve dvije dimenzije slike `x`, `fx` i `fy` su implementacije filtera S_1 i S_2, a pomoćno polje `sz` osigurava da kod radi i za slike u boji, tj. ako `x` ima i treću dimenziju na koju treba obratiti pažnju kod trasnponiranja.

```
def tensor2_impl(x, fx, fy, bd_mode):
    sz = arange(len(shape(x)))
    #racunamo S1(X)
    fx(x, bd_mode)
    #transponiranjem dobivamo (S1X)^T = (X^T)(S1^T)
    y = transpose(x, concatenate( ([1, 0], sz[2:]) ))
    #racunamo S2((S1X)^T)
    fy(y, bd_mode)
    # transponiranjem dobivsmo S1X(S2)^T
    x[:] = transpose(y, concatenate( ([1, 0], sz[2:]) ))
```

Ako je molekula dobivena pomoću filtera S_1 i S_2 s koeficijentima $\mathbf{t}_1$ i $\mathbf{t}_2$ onda možemo izračunati $S(X)$ sa sljedećim kodom.

```
def S1(x, bd_mode):
    filter_impl(t1, x, bd_mode)

def S2(x, bd_mode):
    filter_impl(t2, x, bd_mode)

tensor2_impl(X, S1, S2, 'symm')
```

Tu vidimo da smo pozvali funkciju `filter_impl` s parametrom "**symm**", tj. očekujemo simetrične filtere. Ako filter nije simetričan, zadnji parametar treba promijeniti u "**per**". Također, pretpostavljamo da su duljine filtera neparne i da srednji element ima indeks 0 jer to pretpostavlja i funkcija `filter_impl`.

Operacije bazirane na filterima možemo i kompaktije zapisati koristeći tenzorski produkt.

Definicija 2.2.3 (Tenzorski produkt vektora)**.** *Neka su $\boldsymbol{x}$ i $\boldsymbol{y}$ vektori duljina M i N. Njihov tenzorski produkt $\boldsymbol{x} \otimes \boldsymbol{y}$ je definiran kao $M \times N$-matrica s elementima $(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j} = x_i y_j$. Drugim riječima, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.*

Napomena 2.2.4 (Standardna baza za $M_{M,N}(\mathbb{R})$)**.** *Neka su $\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1}$ i $\mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$ redom standardne baze za $\mathbb{R}^M$ i $\mathbb{R}^N$. Tada je*

$$\boldsymbol{\mathcal{E}}_{M,N} = \left\{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\right\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

baza za $M_{M,N}(\mathbb{R})$, vektorski prostor matrica dimenzija $M \times N$. Naziva se još i standardna baza za $M_{M,N}(\mathbb{R})$. U matrici $\boldsymbol{e}_i \otimes \boldsymbol{e}_j$ element na mjestu (i, j) je 1, dok su ostali elementi jednaki 0.

Sad vidimo da crno-bijele i slike u sivim nijansama možemo promatrati kao matrice unutar $M_{M,N}(\mathbb{R})$, a računalne molekule kao specijalni tip linearnih transformacija s $M_{M,N}(\mathbb{R})$ u $M_{M,N}(\mathbb{R})$.

Definicija 2.2.5 (Tenzorski produkt matrica)**.** *Neka su $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ i $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ matrice linearnih transformacija. Tada definiramo $S_1 \otimes S_2$ kao jedinstveno preslikavanje s $M_{M,N}(\mathbb{R})$ u $M_{M,N}(\mathbb{R})$ koje zadovoljava:*

$$(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j),$$

za svaki i i j. $S_1 \otimes S_2$ se naziva tenzorski produkt matrica S_1 i S_2

Iz linearne algebre znamo da ako je S linearno preslikavanje na V i $S(v_i)$ je poznato za bazu $\{v_i\}_i$ za V, tada je S jedinstveno određeno za sve $x \in V$. Iz toga i činjenice da je $\{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{i,j}$ baza, slijedi da gore navedeno preslikavanje postoji i jedinstveno je. Očito iz linearnosti vrijedi $(S_1 \otimes S_2)(\mathbf{x} \otimes \mathbf{y}) = (S_1\mathbf{x}) \otimes (S_2\mathbf{y})$, za bilo koje $\mathbf{x}$ i $\mathbf{y}$.

Sada možemo pokazati vezu između kompaktnog zapisa filtera i računalnih molekula. Nismo formalno definirali tenzorski produkt između kompaktnih zapisa filtera, no taj produkt je izravno proširenje tenzorskog produkta vektora, gdje dodatno označavamo element na mjestu $(0, 0)$.

Teorem 2.2.6. *Ako su $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ i $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ matrice linearnih transformacija, onda je $(S_1 \otimes S_2)X = S_1X(S_2)^T$, za $X \in M_{M,N}(\mathbb{R})$. Točnije, $S_1 \otimes S_2$ je operacija koja primjenjuje S_1 na stupce od X, i S_2 na dobvene retke. Ako su S_1 i S_2 filteri s kompaktnim zapisima $\boldsymbol{t}_1$ i $\boldsymbol{t}_2$, onda $S_1 \otimes S_2$ ima računalnu molekulu $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2$.*

Dokaz. Vrijedi

$$(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S_1\boldsymbol{e}_i) \otimes (S_2\boldsymbol{e}_j) = S_1\boldsymbol{e}_i((S_2\boldsymbol{e}_j))^T$$

$$= S_1\boldsymbol{e}_i(\boldsymbol{e}_j)^T(S_2)^T = S_1(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)(S_2)^T$$

Iz ovoga slijedi $(S_1 \otimes S_2)X = S_1X(S_2)^T$, za $X \in M_{M,N}(\mathbb{R})$ zato što tvrdnja vrijedi za vektore baze. S obzirom na to da matricu A sa vrijednostima $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$ možemo zapisati i kao $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2$, slijedi tvdrnja teorema. □

Zaglađivanje slike

Pogledajmo sada kako možemo iskoristiti operacije bazirane na filetrima na nekim primjerima. Kod zvučnih signala spomenuli smo tzv. *niskopropusne (eng. low-pass)* filtere, koji prigušuju visoke frekvencije. Sada ćemo vidjeti kako možemo izgladiti slike primjenjući niskopropusne filtere na retke i stupce u slici. U daljnim primjerima ćemo koristiti sliku 2.9 koja je približena na lice kako bi efekt zaglađivanja bi vidljiviji.

Slika 2.9: Slika na kojoj radimo zaglađivanje

Koristit ćemo niskopropusne filtere s koeficijentima iz Pascalovog trokuta. Ako uzmemo filter $S_1 = \frac{1}{4}\{1, \underline{2}, 1\}$ (drugi red Pascalovog trokuta), iz teorema 2.2.2 dobivamo računalnu molekulu

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Ako sada uzmemo filter $S_2 = \frac{1}{64}\{1, 6, 15, \underline{20}, 15, 6, 1\}$ (šesti red Pascalovog trokuta) dobivamo molekulu

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}$$

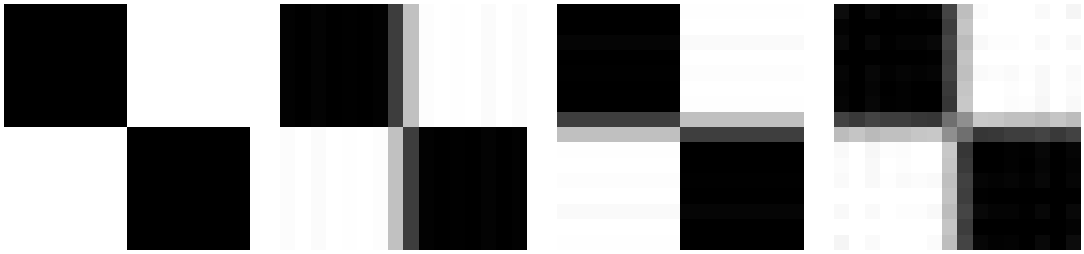
Pogledajmo sada implementaciju i dobivene slike. Vidimo da druga molekula (slika 2.10 desno) zaglađuje više nego prva, što je i očekivano.

```
def S1(x, bd_mode):  # implementacija filetra S1
    filter_impl(array([1, 2, 1])/4., x, bd_mode)

def S2(x, bd_mode): # implementacija filetra S2
    filter_impl(array([1, 6, 15, 20, 15, 6, 1])/64., x, bd_mode)

X1 = X.copy() # X sadrzi zumiranu sliku
tensor2_impl(X1, S1, S1, 'symm')

X2 = X.copy()
tensor2_impl(X2, S2, S2, 'symm')
```

Slika 2.10: Slike dobivene zaglađivanjem slike 2.9

Ako želimo, možemo zaglađivati sliku i u samo jednom smjeru. Pogledajmo to na sljedećem primjeru.

```
# prvo "stvorimo" sliku
N=16
img=255*ones((N, N))
img[0:(int(N/2)),0:(int(N/2))] = 0
img[(int(N/2)):N,(int(N/2)):N] = 0

#defiramo filter koji nista ne radi
def nista(x, bd_mode):
    a=1

def S1(x, bd_mode):
    filter_impl(array([1, 2, 1])/4., x, bd_mode)

X1 = img.copy()
tensor2_impl(X1, nista, S1, 'symm') # zagladujemo samo horizontalno

X2 = img.copy()
tensor2_impl(X2, S1, nista, 'symm') # zagladujemo samo vertikalno

X3 = img.copy()
tensor2_impl(X3, S1, S1, 'symm') # zagladujemo u oba smjera
```



Slika 2.11: Primjeri izglađivanja lijeve slike na tri različita načina

U ovom primjeru smo opet koristili filter $S_1 = \frac{1}{4}\{1, \underline{2}, 1\}$. Na drugoj i trećoj slici možemo vidjeti rezultat dobiven zaglađivanjem prve slike u samo jednom smjeru, a na četvrtoj vidimo rezultat zaglađivanje u oba smjera.

Detekcija rubova

Sada ćemo pokazati i drugi važan primjer operacija koje možemo prikazati pomoću računalnih molekula - detekciju rubova. Rubovi na slikama su karakterizirani velikim promjenama vrijednosti inteziteta na malom području slike. Kod neprekidnih funkcija to bi odgovaralo velikim derivacijama. Ne možemo izračunati derivaciju za slike jer su one definirane u izoliranim točkama, ali postoje tehinike iz numeričke matematike koje nam mogu pomoći.

Prvo ćemo računati parcijalnu derivaciju $\partial P/\partial x$. Primjetimo da se kod slika druga koordinata odnosi na smjer x na koji smo navikli kod crtanja funcija. Parcijalnu derivaciju možemo aproksimirati simetričnom razlikom [2]:

$$\frac{\partial P}{\partial x}(i, j) \approx \frac{p_{i,j+1} - p_{i,j-1}}{2},$$

gdje smo uzeli $h = 1$, pa derivacija mjeri 'intezitet po pikselu'. To odgovara primjeni visokopropusnog filtera $S = \frac{1}{2}\{1, \underline{0}, -1\}$ na sve retke slike, ili primjeni tenzorskog produkta $I \otimes S$ na sliku, tj. molekule

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{0} & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Pogledajmo sada što dobijemo kada to primjenimo na sliku 2.9. Koristimo sljedeći kod:

```
def diffxmolecule(x, bd_mode):
    filter_impl(array([1, 0, -1])/2., x, bd_mode)

def nista(x, bd_mode):
```

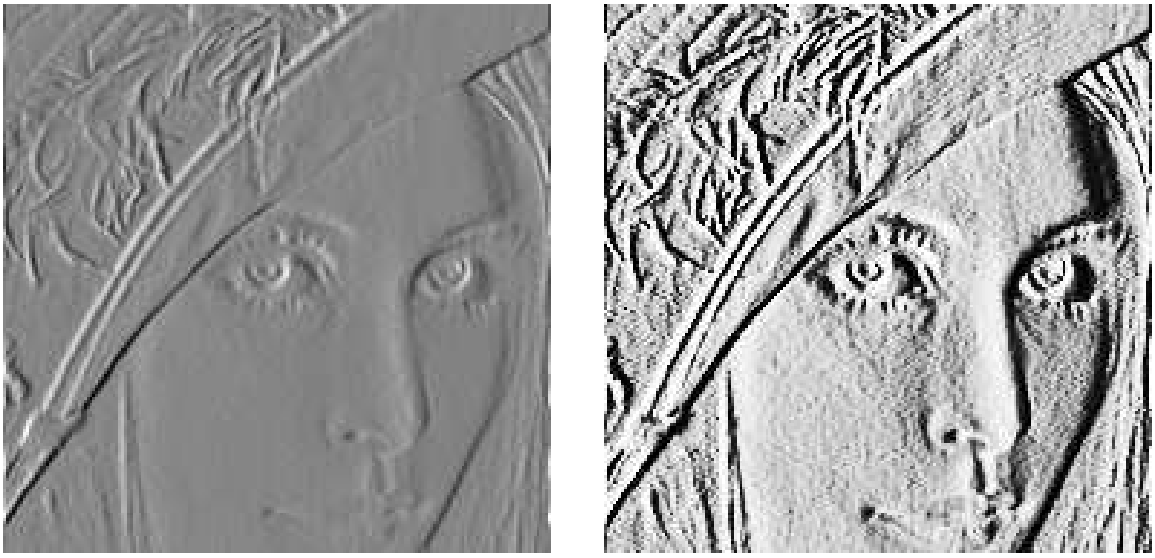
```
    a=1

X1 = X.copy()
tensor2_impl(X1, nista, diffxmolecule, 'per')
```

Slika 2.12: Slika dobivena pracijalnom defivacijom po x

Vidimo da se na slici pojavljuju mnogi artefakti jer velik broj vrijednosti inteziteta nije u dozvoljenom intervalu, točnije dosta vrijednosti je negativno. Sljedećim kodom možemo normalizirati vrijednosti. Tako dobivamo lijevu sliku sa slike 2.13. Na kraju još povećavamo kontrast funkcijom f_{50} kako bi slika bila detaljnija te dobivamo desnu sliku sa slike 2.13.

```
#preslikavanje u inteval [0,1]
X2 = X.copy()
tensor2_impl(X2, nista, diffxmolecule, 'per')
map_to_01(X2)
X2 *= 255
imshow(uint8(X2))

#preslikavanje u inteval [0,1] i povecanje kontrasta
```

```
X3 = X.copy()
tensor2_impl(X3, nista, diffxmolecule, 'per')
map_to_01(X3)
X3 *= 255
contrast_adjust1(X3,50)
imshow(uint8(X3))
```



Slika 2.13: Slike nakon normalizacije i povećanja kontrasta

Objasnimo sada što možemo zaključiti iz ovih slika. Izračunali smo derivaciju u x smjeru, te za ovu sliku ispada da su najmanja i najveća vrijednost derivacije otprilike jednake apsolutne vrijednosti, ali suprotnih predznaka. Vrijednosti 0 sa slike 2.12 nam kažu da nema promjene inteziteta između dva piksela i one će biti preslikane približno oko 0.5. To možemo vidjeti na lijevoj slici u 2.13, gdje su rubovi (područja gdje su najveće vrijednosti derivacije) preslikani u crnu i bijelu boju, a točke s malim ili nikakvim promjenama intenziteta su preslikane u sivu boju. Iz toga možemo zaključiti da na većini slike nema velikih promjena intenziteta.

Analogno možemo izračunati i parcijanlu derivaciju $\partial P/\partial y$, primjenjujući filter $-S = \frac{1}{2}\{-1, \underline{0}, 1\}$ na sve retke slike P (ili primjenjujući tenzorski produkt $-S \otimes I$ na sliku). Tako dobivamo molekulu

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{0} & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

Sljedećim kodom dobivamo desnu sliku sa slike 2.14. Ponovno normaliziramo vrijedosti i povećavamo kontrast funkcijom f_{50}. Lijeva slika je parcijalna derivacija po x, normalizirana i s povećanim kontrastom, kao i gore. Sa slike 2.14 možemo vidjeti da parcijalna derivacija po x naglašava vertikalne rubove, dok parcijalna derivacija po y naglašava horizontalne rubove.

```
def diffymolecule(x, bd_mode):
    filter_impl(array([-1, 0, 1])/2., x, bd_mode)

X4 = X.copy()
tensor2_impl(X4, diffymolecule, nista, 'per')
map_to_01(X4)
X4 *= 255
contrast_adjust1(X4,50)
```

Slika 2.14: Parcijalne derivacije prvog reda po x i y. Na obje slike su vrijednosti normalizirane i kontrast povećan

Kada smo izračunali derivacije prvog reda, lako je izračunati i gradijent i njegovu duljinu. Podsjetimo, gradijent funkcije dvije varijabe je definiran kao vektor

$$\nabla P = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right),$$

a njegova duljina je

$$|\nabla P| = \sqrt{\left(\frac{\partial P}{\partial x} \right)^2 + \left(\frac{\partial P}{\partial y} \right)^2}$$

Kao kod derivacija prvog reda, moguće je da duljina gradijenta bude izvan dozvoljenog intervala, pa ponovno treba normalizirati. Na slici 2.15 mozemo vidjeti sliku gradijenta (lijevo), sliku nakon preslikavanja u dozvoljeni interval (sredina) i sliku nakon namještanja kontrasta funkcijom f_{100} (desno).

Slika 2.15: Slike gradijenta

Vidimo da se slika gradijenta razlikuje od slika parcijalnih derivacija. Razlog je taj što je duljina gradijenta korijen sume kvadrata vrijednosti parcijalnh derivacija. To znači da će sada područja slike gdje nema velike promjene intenziteta (parcijalne derivacije blizu 0) biti obojeni u crno, za razliku od slika parcijanlih derivacija gdje bi te vrijednosti bile preslikane oko 0.5, tj. ti dijelovi slike bili bi obojani sivom bojom. Za pojačati kontrast na ovoj slici bi zato bilo bolje koristiti neku od funkcija g_n. Rezultat povećanja kontrasta funkcijom $g_{0.01}$ možemo vidjeti na slici 2.16. Očito gradijent sadrži informacije o obje parcijalne derivacije pa naglašava rubove u svim smjerovima.

Slika 2.16: Slika gradijenta nakon normaliziranja i povećanja kontrasta funkcijom $g_{0.01}$

Ako sada želimo računati derivcije drugog reda možemo koristiti molekule opisane i korištene kod derivacija prvog reda. Za

- $\frac{\partial^2 P}{\partial x^2}$ koristimo $(I \otimes S)(I \otimes S) = I \otimes S^2$
- $\frac{\partial^2 P}{\partial y \partial x}$ koristimo $(I \otimes S)((-S) \otimes I) = -S \otimes S$
- $\frac{\partial^2 P}{\partial y^2}$ koristimo $((-S) \otimes I)((-S) \otimes I) = S^2 \otimes I$

Derivacije drugog reda slike 2.9 dobivene su sljedećim kodom i prikazane na slici 2.17. Ponovno normaliziramo vrijednosti i povećavamo kontrast funkcijom f_{100}.

```
X1 = X.copy()

# xx
tensor2_impl(X1, nista, diffxmolecule, 'per')
tensor2_impl(X1, nista, diffxmolecule, 'per')
map_to_01(X1)
X1 *= 255
contrast_adjust1(X1, 100)
imshow(uint8(X1))

# xy
X2 = X.copy()
tensor2_impl(X2, diffymolecule, diffxmolecule, 'per')
```

```
map_to_01(X2)
X2 *= 255
contrast_adjust1(X2, 100)
imshow(uint8(X2))

#yy
X3 = X.copy()
tensor2_impl(X3, diffymolecule, nista, 'per')
tensor2_impl(X3, diffymolecule, nista, 'per')
map_to_01(X3)
X3 *= 255
contrast_adjust1(X3, 100)
```

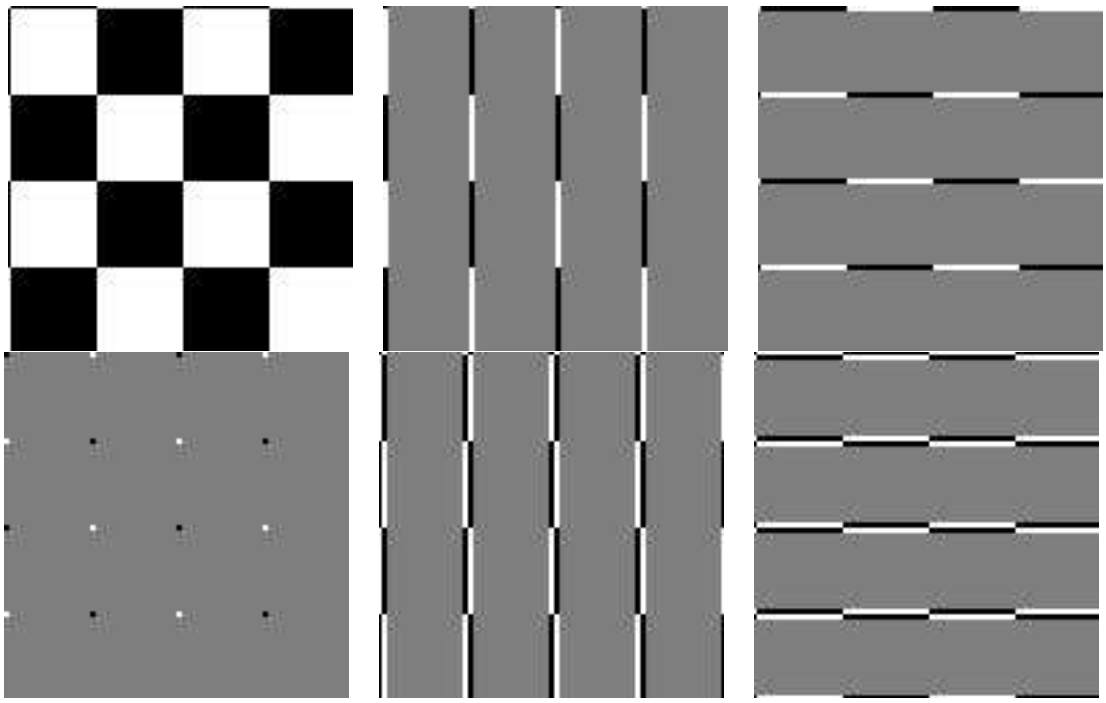
Slika 2.17: Parcijalne derivacije drugog reda

Kao i kod derivacija prvog reda, $(I \otimes S^2)$ naglašava vertikalne rubove, a $(S^2 \otimes I)$ horizontalne. No vidimo da su derivacije drugog reda osjetljivije na šum na slici jer su područja sive boje manje ujednačena. $S \otimes S$ će naglasiti točke u kojima dolazi do naglih promjena vrijednosti u oba smjera.

Ovaj proces možemo generalizirati na derivacije višeg reda. Da bi primjenili $\frac{\partial^{k+l}}{\partial x^k \partial y^l}$ na sliku možemo izračunati $S_l \otimes S_k$, gdje S_r odgovara bilo kojoj metodi za izračunavanje derivacije r-og reda. Također možemo izračunati $(S^l \otimes S^k)$ iterirajući filer $S = \frac{1}{2}\{1, \underline{0}, -1\}$ za derivaciju prvog reda.

Na slici 2.18 možemo vidjeti redom primjenu $I \otimes S$, $(-S) \otimes I$, $(-S) \otimes S$, $I \otimes S^2$ i $S^2 \otimes I$ na sliku dobivenu sljedećim kodom i prikazanu u gornjem ljevom kutu.

```
N = 128;
img=zeros((N, N))
for x in range(N):
    for y in range(N):
        img[x,y] = 255*( (mod(x-1,64)>=32) == (mod(y-1,64)>=32) )
```



Slika 2.18: Različiti tenzorski produkti primjenjeni na sliku u gornjem lijevom kutu

Iz gornje slike još je vidljivije da $I \otimes S$ otkriva vertikalne rubove, $(-S) \otimes I$ horizontalne, a $(-S) \otimes S$ otkriva točke gdje dolazi do nagle promjene vrijednosti u oba smjera. Također, vidimo da parcijalne dervacije drugog reda otkrivaju iste rubove kao i parcijalne derivacije prvog reda, ali i da su rubovi otkriveni s $I \otimes S^2$ širi od onih otkrivenih s $I \otimes S$. Razlog za to je činjenica da filter S^2 ima više koeficijenata od filtera S.

2.3 Promjena koordinata u tenzorskom produktu

U prvom poglavlju smo promatrali smo DCT i DFT koji su bili definirani kao promjene koordinata vektorskim prostorima a sada ćemo definirati slične operacije za promjenu koordinata za slike. Ponovno će nam koristan biti tenzorski produkt.

Teorem 2.3.1 (Baza $\mathcal{B}_1 \otimes \mathcal{B}_2$)**.** *Ako je* $\mathcal{B}_1 = \{\boldsymbol{v}_i\}_{i=0}^{M-1}$ *baza za* $\mathbb{R}^M$ *i* $\mathcal{B}_2 = \{\boldsymbol{w}_i\}_{i=0}^{N-1}$ *baza za* $\mathbb{R}^N$*, onda je* $\left\{\boldsymbol{v}_i \otimes \boldsymbol{w}_j\right\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ *baza za* $M_{M,N}(\mathbb{R})$ *i označavamo je sa* $\mathcal{B}_1 \otimes \mathcal{B}_2$.

Dokaz. Pretpostavimo da je $\sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = 0$. Stavimo $\boldsymbol{h}_i = \sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j$ i dobivamo

$$\sum_{j=0}^{N-1} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \boldsymbol{v}_i \otimes (\sum_{j=0}^{N-1} \alpha_{i,j}\boldsymbol{w}_j) = \boldsymbol{v}_i \otimes \boldsymbol{h}_i$$

gdje smo koristili bilinearnost tenzorskog produkta. To znači

$$\mathbf{0} = \sum_{(i,j)=(0,0)}^{(M-1,N-1)} \alpha_{i,j}(\boldsymbol{v}_i \otimes \boldsymbol{w}_j) = \sum_{i=0}^{M-1} \boldsymbol{v}_i \otimes \boldsymbol{h}_i = \sum_{i=0}^{M-1} \boldsymbol{v}_i \boldsymbol{h}_i^T.$$

k-ti stupac u matričnoj jednadžbi kaže $\mathbf{0} = \sum_{i=0}^{M-1} h_{i,k}\boldsymbol{v}_i$, gdje su $h_{i,k}$ komponente od $\boldsymbol{h}_i$. Iz linearne nezavisnosti baze $\{\boldsymbol{v}_i\}$ slijedi da mora biti $h_{0,k} = h_{1,k} = \cdots = h_{M-1,k} = 0$. Slijedi da mora biti $\boldsymbol{h}_i = 0$. To znači da je $\sum_{j=0}^{N-1} \alpha_{i,j} = 0$, za svaki i, iz čega iz linearne nezavisnosti baze $\{\boldsymbol{w}_j\}$ slijedi da su $\alpha_{i,j} = 0$, za sve i i j. To znači da je $\mathcal{B}_1 \otimes \mathcal{B}_2$ baza. □

Definicija 2.3.2 (Koordinatna matrica)**.** *Neka je $\mathcal{B} = \{\boldsymbol{b}_i\}_{i=0}^{M-1}$ baza za $\mathbb{R}^M$, $C = \{\boldsymbol{c}_i\}_{i=0}^{N-1}$ baza za $\mathbb{R}^N$ i $A \in M_{M,N}(\mathbb{R})$. Koordinatna matrica od A u bazi $\mathcal{B} \otimes C$ je $M \times N$-matrica X (s komponentama $X_{k,l}$), takva da je $A = \sum_{k,l} X_{k,l}(\boldsymbol{b}_k \otimes \boldsymbol{c}_l)$.*

Teorem 2.3.3 (Promjena koordinata u tenzorskom produktu)**.** *Pretpostavimo sljedeće:*

- *$\mathcal{B}_1$ i C_1 su baze za $\mathbb{R}^M$ i S_1 matrica prijelaza iz baze $\mathcal{B}_1$ u bazu C_1*
- *$\mathcal{B}_2$ i C_2 su baze za $\mathbb{R}^N$ i S_2 matrica prijelaza iz baze $\mathcal{B}_2$ u bazu C_2*

Tada su $\mathcal{B}_1 \otimes \mathcal{B}_2$ i $C_1 \otimes C_2$ baze za $M_{M,N}(\mathbb{R})$ i ako je X koordinatna matrica u bazi $\mathcal{B}_1 \otimes \mathcal{B}_2$, i Y koordinatna matrica u bazi $C_1 \otimes C_2$, tada se promjena koordinata iz $\mathcal{B}_1 \otimes \mathcal{B}_2$ u $C_1 \otimes C_2$ može izračunati sa:

$$Y = S_1 X (S_2)^T \tag{2.2}$$

Vidimo da kao i kod filtriranja moramo izračunati preslikavanje $X \to S_1 X (S_2)^T$, što se svodi na operacije na stupcima i retcima.

Napomena 2.3.4 (Promjena koordinata u tenzorskom produktu)**.** *Promjenu koordinata iz baze $\mathcal{B}_1 \otimes \mathcal{B}_2$ u bazu $C_1 \otimes C_2$ možemo implementirati na sljedeći način:*

- *Za svaki stupac u koordinatnoj matrici u $\mathcal{B}_1 \otimes \mathcal{B}_2$, obavimo zamjenu koordinata iz $\mathcal{B}_1$ u C_1.*
- *Za svaki redak u dobivenoj matrici, obavimo zamjenu koordinata iz $\mathcal{B}_2$ u C_2.*

Slika 2.19: Slika korištena u primjerima

DFT i DCT smo definirali kao promjenu koordinata iz standardne baze za $\mathbb{C}^N$ i $\mathbb{R}^N$ u Fourierovu bazu $\mathcal{F}_N$ i DCT bazu $\mathcal{D}_N$. Sada želimo implementirati analogne operacije za dvodimenzionalne objekte. Podsjetimo se da DFT vrijednosti izražavaju frekvencijske komponente, a isto vrijedi i za 2D DFT, pa i za slike, ali frekvencije su sada predstavljene u dva različita smjera. Zamjenu koordinata obavljamo na sljedeći način: pretpostavljamo da su vrijednosti piksela slike koordinate u standarnoj bazi, te prvo obavljamo zamjenu koordinata u bazu $\mathcal{F}_N$ ili $\mathcal{D}_N$ za svaki stupac, a nakon toga obavimo zamjenu koordinata u $\mathcal{F}_N$ ili $\mathcal{D}_N$ na svakom retku dobivene matrice. U obradi slika običaj je da se zamjena koordinata ne vrši na cijeloj slici nego se slika podijeli na manje kvadrate, koje nazivamo blokovi, te se zamjena koordinata izvršava na svakom bloku zasebno. U primjerima ćemo podijeliti sliku na blokove veličine 8×8, jer znamo da DFT i DCT imaju efikasne implementacije kada je N potencija broja 2. Pokažimo sada kako možemo implementirati funkcije `dft_impl8`, `idft_impl8`, `dct_impl8` i `idct_impl8` koje će primjenjivati DFT, IDFT, DCT i IDCT na uzastopne blokove duljine 8. Koristimo funkcije definirane u biblioteci `numpy`. Parametar `axis` određuje os po kojoj će se izvršavati FFT i DCT, a parametar `norm` određuje na koji način i kojim faktorom će transformacije biti skalirane.

```
def dft_impl8(x, bd_mode):
    N = shape(x)[0]
    for n in range(0, N, 8):
        x[n:(n+8), :] = fft.fft(x[n:(n+8), :], axis=0)
def idft_impl8(x, bd_mode):
    N = shape(x)[0]
    for n in range(0, N, 8):
        x[n:(n+8), :] = fft.ifft(x[n:(n+8), :], axis=0)
```

```
def dct_impl8(x, bd_mode):
    N = shape(x)[0]
    for n in range(0, N, 8):
        x[n:(n+8), :] = dct(x[n:(n+8), :], norm='ortho', axis=0)
def idct_impl8(x, bd_mode):
    N = shape(x)[0]
    for n in range(0, N, 8):
        x[n:(n+8), :] = idct(x[n:(n+8), :], norm='ortho', axis=0)
```

Sada možemo implementirati višedimenzionalni DFT, IDFT, DCT i IDCT za slike. S obzirom da trebamo izračunati preslikavanje $X \to S_1 X (S_2)^T$, ponovno koristimo funkciju `tensor2_impl`, a umjesto S_1 i S_2 koristit ćemo odgovarajuće funkcije za zamjenu koordinata.

```
def dft_impl(x, bd_mode):
    x[:] = fft.fft(x, axis=0)

tensor2_impl(X, dft_impl_full, dft_impl_full, 'symm')

def idft_impl_full(x, bd_mode):
    x[:] = fft.ifft(x, axis=0)

tensor2_impl(X, idft_impl_full, idft_impl_full, 'symm')

def dct_impl_full(x, bd_mode):
    x[:] = dct(x, norm='ortho', axis=0)

tensor2_impl(X, dct_impl_full, dct_impl_full, 'symm')

def idct_impl_full(x, bd_mode):
    x[:] = idct(x, norm='ortho', axis=0)

tensor2_impl(X, idct_impl_full, idct_impl_full, 'symm')
```

Ovako definirani, DFT, IDFT, DCT i IDCT primjenjuju se na cijelu sliku a ne na blokove. Sljedeći kod koristimo ako želimo primjenu na blokovima.

```
tensor2_impl(X dft_impl8, dft_impl8, 'symm')

tensor2_impl(X, idft_impl8, idft_impl8, 'symm')

tensor2_impl(X, dct_impl8, dct_impl8, 'symm')

tensor2_impl(X, idct_impl8, idct_impl8, 'symm')
```

Pokažimo sada kako možemo koristiti ove funkcije na primjerima. U primjerima koristimo funkciju `forw_comp_rev_2d` iz biblioteke dostupne uz knjigu [3]. Ideja je primjeniti

zamjenu koordinata slike, pa odbaciti, tj. postaviti na 0 one koeficijenate koji su manji od određenog praga zadanog parametrom `threshold` i na kraju se ponovno vratiti u koordinate u standardnoj bazi. Funkcija `forw_comp_rev_2d` ispisuje koliki postotak koeficijenata je odbačen (tj. manji od `threshold`). Ako dobro odaberemo prag, moguće je da razlike između tako dobivene i orginalne slike budu jedva vidljive ljudskom oku.

```
def forw_comp_rev_2d(X, f, invf, threshold):
    M, N = shape(X)[0:2]

    tensor2_impl(X, f, f, 'symm')
    tot = prod(shape(X))

    thresholdmatr = (abs(X[:,:,:]) >= threshold)
    zeroedout = tot - sum(thresholdmatr)
    X[:,:,:] *= thresholdmatr
    tensor2_impl(X[:,:,:], invf, invf, 'symm');
    X[:] = abs(X)
    map_to_01(X)
    X *= 255
    print('%f percent of samples zeroed out' % (100*zeroedout/float(tot)
                                                ))
```

Na slici 2.20 vidimo slike dobivene primjenom ove funkcije s `dft_impl8` i `idft_impl8` i parametrom `threshold` jednakim 100, 200 i 400 redom. Koristimo sljedeći kod.

```
X1 = X.copy() # X sadrzi orginalnu sliku
forw_comp_rev_2d(X1, dft_impl8, idft_impl8, 100)

X2 = X.copy()
forw_comp_rev_2d(X2, dft_impl8, idft_impl8, 200)

X3 = X.copy()
forw_comp_rev_2d(X3, dft_impl8, idft_impl8, 400)
```

Slika 2.20: Slike dobivene primjenom DFT na blokove, i odbacivanjem DFT koeficijenata ispod zadanog praga (redom 100, 200, 400)

Na prvoj slici vidimo da je razlika skoro nezamjetna, a odbačeno je 76,6% koeficijenata. Kod druge slike odbačeno je 89,3% koeficijenata, i granice između blokova su vidljivije. Kod posljednje slike je odbačeno 95,3% koeficijenata, slika je još raspoznatljiva ali se i očito može vidjeti podjela na blokove.

Kad smo definirali DCT rekli smo da se u obradi slika koristi više nego DFT. Na primjer JPEG standard koristi DCT, a ne DFT. Također se koristi i podijela na blokove veličine 8×8, kao i gore. Pokažimo sada slike dobivene koristeći `forw_comp_rev_2d` sa `dct_impl8` i `idct_impl8`. Parametar `threshold` je redom 30, 50 i 100.

```
X1 = X.copy()
forw_comp_rev_2d(X1, dct_impl8, idct_impl8, 30)

X2 = X.copy()
forw_comp_rev_2d(X2, dct_impl8, idct_impl8, 50)

X3 = X.copy()
forw_comp_rev_2d(X3, dct_impl8, idct_impl8, 100)
```

Slika 2.21: Slike dobivene primjenom DCT na blokove, i odbacivanjem DCT koeficijenata ispod zadanog praga (redom 30, 50, 100)

Na slikama u 2.21 odbačeno je redom 93,2%, 95,8% i 97,7% koeficijenata, no iako je odbačen veći postotak koeficijenata slike su dosta slične onima iz 2.20.

Još je zanimljivo vidjeti što dobijemo ako sliku ne podijelimo u blokove, tj primjenimo DCT na cijelu sliku. Na 2.22 su pokazane tako dobivene slike s istim vrijednostima parametra `threshold` kao i na slici 2.21.

Slika 2.22: Slike dobivene primjenom DCT, i odbacivanjem DCT koeficijenata ispod zadanog praga (redom 30, 50, 100). Sliku ne dijelimo na blokove

Redom je odbačeno 93,2%, 96,6% i 98,8% koeficijenata. Kao što bi i očekivali nakon odbacivanja tolikog broja koeficijenata, na slikama se pojavljuju artefakti, no za razliku od slika 2.20 i 2.21 artefakti se ne pojavljuju na granicama blokova.

Bibliografija

[1] Leonid Mirsky, *An introduction to linear algebra*, Courier Corporation, 2012.

[2] Knut Mørken, *Numerical Algorithms and Digital Representation*, 2013.

[3] Øyvind Ryan, *Linear Algebra, Signal Processing, and Wavelets–a Unified Approach*, Springer, 2019.

[4] Georgi P. Tolstov, *Fourier series*, Courier Corporation, 2012.

Sažetak

U ovom radu pokazano je kako primjeniti matematičke alate u obradi digitalnih slika. Pri implementaciji je korišten programski jezik python. U prvom poglavlju definirani su algoritmi za DFT i DCT, kao i njihove implementacije, te je naveden i kratki opis i implementacija filtera. U drugom poglavlju objašnjeno je kako se slike reprezentiraju u računalu i kako im jednostavno mijenjati neka svojstva. Pokazana je primjena jednostavnih operacija kao npr. mijenjanje kontrasta, te primjena operacija baziranih na filterima(detekcija rubova i zaglađivanje slike). Dana je i definicija tenzorskog produkta koji je koristan kako bi se algoritmi definirani u prvom poglavlju jednostavno proširili na višedimenzionalno okruženje. Na kraju je pokazana implementacija višedimenzionalnih algoritama za FFT i DCT, te primjena tih algoritama na konkretnim primjerima.

Summary

In this thesis is shown how to apply mathematical tools in image processing. The Python programming language was used during the implementation. In the first chapter, the algorithms for DFT and DCT are defined, as well as their implementations, and a brief description and implementation of the filter is provided. In the second chapter, it is explained how images are represented in the computer and how to easily change some of their properties. The application of simple operations, such as changing the contrast, and the application of filter based operations (edge detection and image smoothing) are shown. A definition of the tensor product is also given, which is useful in order to easily extend the algorithms defined in the first chapter to a multidimensional environment. At the end, the implementation of multidimensional algorithms for FFT and DCT is shown, as well as the application of these algorithms on concrete examples.

Životopis

Rođen sam 25. lipnja 1995. u Zadru. Školovanje sam započeo 2002. godine u Osnovnoj školi Valentina Klarina u Preku te nastavio 2010. godine upisom prirodoslovno-matematičkog smjera u Gimnaziji Jurja Barakovića u Zadru. Nakon toga upisujem preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu, gdje sam 2019. godine stekao titulu sveučilišnog prvostupnika matematike, nakon čega upisujem diplomski studij Računarstvo i matematika.