

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO  
FAKULTETA ZA MATEMATIKO IN FIZIKO

Gregor Šraj

**Uporaba diskretne kosinusne  
transformacije pri kompresiji slik v  
formatu JPEG**

DIPLOMSKO DELO

INTERDISCIPLINARNI UNIVERZITETNI  
ŠTUDIJSKI PROGRAM PRVE STOPNJE  
RAČUNALNIŠTVO IN MATEMATIKA

MENTOR: doc. dr. Tadej Kanduč

Ljubljana, 2023

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavnine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani creativecommons.si ali na Inštitutu za intelektualno lastnino, Strelška 1, 1000 Ljubljana.



Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani <http://www.gnu.org/licenses/>.

*Besedilo je oblikovano z urejevalnikom besedil L<sup>A</sup>T<sub>E</sub>X.*

**Kandidat:** Gregor Šraj

**Naslov:** Uporaba diskretne kosinusne transformacije pri kompresiji slik v formatu JPEG

**Vrsta naloga:** Diplomska naloga na univerzitetnem programu prve stopnje Računalništvo in matematika

**Mentor:** doc. dr. Tadej Kanduč

**Opis:**

Kompresija slik je temelj učinkovitega shranjevanja in pošiljanja slik. Na tem področju prevladuje metoda diskretne kosinusne transformacije. Kandidat bo preučil njeno teoretično ozadje. Praktično implementacijo te metode bo preučil na formatu JPEG.

**Title:** The Discrete Cosine Transform for Image Compression in the JPEG format

**Description:**

Image compression is one of the key elements for efficient image storage and transfer. In this area the discrete cosine transform is one of most recognized methods. Candidate will examine its theoretical background. The practical implementation of this method will studied on JPEG format.



*Zahvalil bi se svojemu mentorju, doc. dr. Tadeju Kanduču, za vse napotke in pomoč pri pisanju te naloge. Zahvalil bi se rad tudi svoji družini in prijateljem, še posebej Marku in Viti, ki sta pregledala nalogo za pravopisne napake.*



# Kazalo

## Povzetek

## Abstract

<b>1 Uvod</b>	<b>1</b>
<b>2 Diskretna kosinusna transformacija</b>	<b>5</b>
2.1 Fourierova kosinusna transformacija . . . . .	5
2.2 Izpeljava DCT . . . . .	7
2.3 Enodimenzionalni DCT . . . . .	7
2.4 Dvodimenzionalni DCT . . . . .	15
2.5 Algoritem za hitri DCT . . . . .	17
2.6 Celoštevilski DCT . . . . .	20
<b>3 JPEG</b>	<b>25</b>
3.1 Predprocesiranje . . . . .	27
3.2 Kodiranje . . . . .	28
3.3 Kvaliteta kompresiranih slik . . . . .	36
3.4 Primer kodiranja . . . . .	37
3.5 Dekodiranje . . . . .	41
3.6 Primer dekodiranja . . . . .	44
<b>4 Uporaba diskretne kosinusne transformacije</b>	<b>49</b>
4.1 Primeri kompresije slik v JPEG-u . . . . .	49
4.2 Problematične slike za JPEG . . . . .	54

**5 Sklepne ugotovitve** **57**

**Literatura** **59**

# Seznam uporabljenih kratic

kratica	angleško	slovensko
DCT	Discrete cosine transform	Diskretna kosinusna transformacija
DFT	Discrete Fourier transform	Diskretna Fourierova transformacija
FCT	Fourier cosine transform	Fourierova kosinusna transformacija
FFT	Fast Fourier transform	Hitra Fourierova transformacija
JPEG	Joint Photographic Experts Group	
MSE	Mean squared error	Povprečna kvadratna napaka
PSNR	Peak signal-to-noise ratio	
VLI	Variable-length integer	Celo število spremenljive dolžine



# Povzetek

**Naslov:** Uporaba diskretno kosinusne transformacije pri kompresiji slik v formatu JPEG

**Avtor:** Gregor Šraj

V diplomskem delu smo definirali diskretno kosinusno transformacijo in predstavili osnovno idejo njene izpeljave. Pokazali smo, kako se ta definicija razširi na dve dimenziji in da lahko dvodimenzionalno obliko izračunamo s pomočjo enodimenzionalne. Opisali smo primer algoritma za hiter izračun DCT s pomočjo že obstoječih algoritmov FFT. Podobno smo opisali primer algoritma za celoštivilski DCT, ki deluje na podlagi enostavnejše aritmetike. Poglobili smo se v najpogostejo implementacijo JPEG-a. Opisali smo celoten postopek kodiranja originalne slike v bitni niz, ki se uporablja za shranjevanje in pošiljanje. Opisali smo potek obratnega procesa, to je dekodiranja. Nato smo na primeru dejanskega bloka ilustrirali oba procesa. Za konec smo nekaj dejanskih slik pretvorili v format JPEG z različnimi nivoji kvalitete. Izračunali smo kvaliteto teh slik in primerjali, kateri nivoji kvalitete so primerni za različne aplikacije.

**Ključne besede:** diskretna kosinusna transformacija, JPEG, aproksimacija podatkov, kompresija podatkov.



# Abstract

**Title:** The Discrete Cosine Transform for Image Compression in the JPEG format

**Author:** Gregor Šraj

In this bachelor's thesis, we will define the Discrete Cosine Transform (DCT) and explain the fundamental principles behind its derivation. Additionally, we will explore its two-dimensional extension and demonstrate how to compute the two-dimensional form using the one-dimensional definition. We will provide an example of an algorithm for fast DCT computation, utilizing existing Fast Fourier Transform (FFT) algorithms. Similarly, we will introduce an example of an Integer DCT algorithm that works on simpler arithmetic operations. Furthermore, we will delve into the most common implementation of JPEG compression. We will explain the encoding process, transforming the original image into a bit string for storage and transmission. We will also describe the inverse process, i.e., decoding. To illustrate these processes, we will use an example involving a block of an image. Lastly, we will convert several images into the JPEG format, varying the quality levels, and subsequently assess their quality to determine the most suitable qualities for specific applications.

**Keywords:** discrete cosine transform, JPEG, data approximation, data compression.



# Poglavlje 1

## Uvod

Diskretna kosinusna transformacija (DCT) in njej podobne metode so temelj današnje komunikacije preko spletja, kjer je kompresija podatkov ključna za hitro pošiljanje podatkov. Uporablja se v formatih JPEG, WebP, HEIF, BPG ... za slike, v formatih H.264, H.265, Apple ProRes, AV1 ... za video posnetke in v formatih MP3, AAC ... za zvočne posnetke.

Osnovna ideja DCT izhaja iz Fourierove transformacije, ki deluje na zveznih signalih. S to preslikavo se periodični signal razstavi na vsoto sinusnih in kosinusnih funkcij z različnimi frekvencami. Iz te transformacije izhaja zvezna oblika DCT, ki se imenuje Fourierova kosinusna transformacija (FCT). Ta je zelo podobna Fourierovi transformaciji, vendar razstavi signal samo na kosinusne funkcije. V praksi želimo imeti diskretizirane oblike teh transformacij, saj je v računalništvu vse diskretizirano. Podobno zvezo med zvezno in diskretno variacijo vidimo v primeru, ko iz Diskretne Fourierove transformacije (DFT) izpeljemo DCT, ki uporablja zgolj diskretizirane kosinusne funkcije.

Začeli bomo v 2. poglavju s teoretičnim ozadjem DCT. Najprej bomo podali osnovno idejo izpeljave DCT z uporabo Fourierove transformacije [3]. Podali bomo definicijo te metode v eni dimenziji in nato razširitev na dve dimenziji ter opisali kako lahko dvodimensionalen problem pretvorimo v enodimensionalnega [9]. Vizualno bomo predstavili, kaj pomeni transformacija

slike z DCT in kaj nam bodo predstavljeni koeficienti v novem prostoru.

Ključna lastnost pri kompresiji podatkov je korelacija med sosednjimi podatki. Korelirane podatke želimo preslikati v nekoreliran frekvenčen prostor, kjer bodo v boljši obliki za kompresijo. Te podatke bomo nato kodirali, da bomo prihranili na prostoru. Preslikava DCT sama po sebi ne povzroči izgube podatkov. S kodiranjem šele dosežemo, da bo to kompresija z izgubo (*angl. lossy compression*). Podatke, ki jih izgubimo, so tisti, ki imajo visoko frekvenco in so manj pomembni. Na ta način bomo lahko shranili slike z manj biti, kar bo za nas predstavljalo prihranek prostora.

DCT je v praksi zelo uporaben, ker je njegov izračun lahko implementiran na hiter način. Mi bomo predstavili metodo, ki za izračun uporablja algoritem za hiter DFT. Hitrost je še toliko pomembnejša, saj je na primer pri video posnetkih včasih potrebno, da kompresija poteka v realnem času. Pokazali bomo tudi metodo, ki lahko teče na enostavnejši aritmetiki, kjer je potrebnih manj operacij na številih, predstavljenih v plavajoči vejici.

Namesto DCT bi lahko uporabljali tudi DFT kot transformacijo za kompresijo slik. To v praksi vrne slabše rezultate, ker je DCT bližje transformaciji Karhunen–Loëve, ki velja za statistično optimalno metodo za zgoščevanje podatkov. Problem transformacije Karhunen–Loëve je, da ne obstaja algoritem za njen hiter izračun, kot pri DCT in DFT. Posledično se ta transformacija v praksi ne uporablja. Več informacij o teh alternativah je zapisano v [2, 3].

V 3. poglavju se bomo osredotočili na slike, saj je na njih najenostavnije predstaviti, kako deluje DCT v praksi in kakšne prihranke nam omogoča. Specifično bomo opisali format JPEG [7, 17]. Pokazali bomo, kako sliko preslikamo v prostor DCT in katere podatke odstranimo, da bo vizualno najmanjša razlika med originalno in skrčeno sliko. Opisali bomo tudi kodiranje, ki se uporablja v tem procesu. Ta celoten proces kodiranja bomo predstavili na primeru. Za konec bomo pokazali, kako dekodiramo podatke nazaj v sliko.

V zadnjem 4. poglavju bomo na nekaj primerih pokazali, kako izgledajo slike JPEG, kjer nastavimo njihovo kvaliteto na različne ravni. Izračunali

bomo njihov PSNR (*angl.* Peak signal-to-noise ratio) in pogledali, kaj to pomeni v praksi. V zadnjem razdelku bomo pogledali tudi primer slike, ki deluje zelo slabo v kombinaciji s kompresijo JPEG.

V tem diplomskem delu se bomo osredotočili na slike, vendar se ta ideja dokaj enostavno razsiri na video posnetke, saj pri njih velja še več uporabnih lastnosti. Ne samo, da so korelirani sosednji piksli, korelirani so tudi piksli na istih lokacijah v zaporednih sličicah. Podobno je ta metoda uporabna za kompresijo zvoka.



# Poglavlje 2

## Diskretna kosinusna transformacija

DCT preslika signal v prostor diskretnih baznih funkcij. To se uporablja pri kompresiji, saj so signali v tem prostoru bolj nekorelirani. Najprej bomo pokazali, kako pridemo do zvezne oblike, torej Fourierove kosinusne transformacije (FCT). Nato bomo opisali idejo diskretizacije za pridobitev DCT.

### 2.1 Fourierova kosinusna transformacija

Izpeljavo FCT bomo naredili po [3], s pomočjo Fourierove transformacije. Dano imamo funkcijo  $x(t)$  na intervalu  $-\infty < t < \infty$ . Če obstaja spodnji integral, je Fourierova transformacija funkcije  $x(t)$  podana s

$$X(\omega) \equiv F[x(t)] := \left( \frac{1}{2\pi} \right)^{1/2} \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt. \quad (2.1)$$

Tukaj sta  $i = \sqrt{-1}$  in  $\omega = 2\pi f$ , kjer je  $\omega$  krožna frekvenca in  $f$  frekvenca v Hertzih. Iz Fourierove transformacije lahko  $x(t)$  dobimo nazaj z inverzno Fourierovo transformacijo

$$x(t) \equiv F^{-1}[X(\omega)] = \left(\frac{1}{2\pi}\right)^{1/2} \int_{-\infty}^{\infty} X(\omega) e^{i\omega t} d\omega. \quad (2.2)$$

V teh definicijah smo uporabili naslednje oznake:

- $t$  ... parameter iz fizičnega prostora,
- $F$  ... Fourierova transformacija,
- $F^{-1}$  ... inverzna Fourierova transformacija,
- $x$  ... funkcija iz fizičnega prostora,
- $X$  ... funkcija iz frekvenčnega prostora.

Uporabljeni sta definiciji Fourierove transformacije in inverzne Fourierove transformacije, ki imata  $\left(\frac{1}{2\pi}\right)^{1/2}$  za vodilni faktor<sup>1</sup>.

Če je  $x(t)$  definirana samo za  $t \geq 0$ , lahko konstruiramo sledečo funkcijo  $y(t)$ :

$$y(t) = \begin{cases} x(t), & t \geq 0 \\ x(-t), & t \leq 0 \end{cases}. \quad (2.3)$$

Potem sledi:

$$\begin{aligned} F[y(t)] &= \left(\frac{1}{2\pi}\right)^{1/2} \left\{ \int_0^{\infty} x(t) e^{-i\omega t} dt + \int_{-\infty}^0 x(-t) e^{-i\omega t} dt \right\} \\ &= \left(\frac{1}{2\pi}\right)^{1/2} \int_0^{\infty} x(t) [e^{-i\omega t} + e^{i\omega t}] dt \\ &= \left(\frac{2}{\pi}\right)^{1/2} \int_0^{\infty} x(t) \cos(\omega t) dt. \end{aligned} \quad (2.4)$$

---

<sup>1</sup>V nekaterih drugih literaturah se ta definicija navaja drugače. V [2] je definirana Fourierova transformacija z enotskim vodilnim koeficientom in inverzna Fourierova transformacija z vodilnim koeficientom  $\frac{1}{2\pi}$ . Različne formulacije tako privedejo do različnih variacij FCT. Tudi lastnosti Fourierove transformacije in FCT so odvisne od uporabljenе definicije.

Zdaj lahko definiramo FCT za  $x(t)$  kot

$$X_C(\omega) \equiv F_C[x(t)] = \left(\frac{2}{\pi}\right)^{1/2} \int_0^\infty x(t) \cos(\omega t) dt. \quad (2.5)$$

Če uporabimo dejstvo, da je  $X_C(\omega)$  soda funkcija, lahko uporabimo Fourierov inverz na (2.4) in dobimo

$$y(t) = x(t) \equiv F_C^{-1}[X_C(\omega)] = \left(\frac{2}{\pi}\right)^{1/2} \int_0^\infty X_C(\omega) \cos(\omega t) d\omega, \quad t \geq 0. \quad (2.6)$$

Opazimo, da je frekvenčni prostor realen, medtem ko je frekvenčni prostor pri Fourierovi transformaciji kompleksen.

## 2.2 Izpeljava DCT

DCT ni zgolj diskretizirana verzija FCT. Idejo dobimo iz opazovanja zvezne transformacije, vendar se DCT izpelje iz rešitve diferencialne enačbe s specifičnimi robnimi pogoji. Odkrivno od robnih pogojev, ki jih vzamemo, dobimo različne variante DCT. Ta izpeljava je opisana v [3]. Varianti, ki se najbolj uporablja in katerima se v diplomskega delu posvetimo, sta DCT tipa II in inverzni DCT, ki je tipa III. Vedno, ko se v diplomski nalogi sklicujemo na DCT, mislimo na DCT tipa II.

## 2.3 Enodimenzionalni DCT

Najpogostejsa definicija DCT enodimenzionalnega zaporedja dolžine  $N$  je

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos \left[ \frac{\pi(2x+1)u}{2N} \right], \quad u = 0, 1, 2, \dots, N-1, \quad (2.7)$$

kjer je  $f(x)$  realna funkcija definirana za  $x = 0, 1, 2, \dots, N-1$ . Vrednosti  $C(u)$  nam predstavljajo utež vsake funkcije v bazi. Vrednostim  $C(u)$  običajno

rečemo kar koeficienti DCT. Podobno je inverzna transformacija definirana kot

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) C(u) \cos \left[ \frac{\pi(2x+1)u}{2N} \right], \quad x = 0, 1, 2, \dots, N-1. \quad (2.8)$$

V definicijah (2.7) in (2.8) je  $\alpha(u)$  definiran kot

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u \neq 0 \end{cases}. \quad (2.9)$$

Preko DCT smo  $f(x)$  v (2.8) zapisali v bazi kosinusnih funkcij. Torej baza DCT je  $B = \{\cos \frac{\pi(2x+1)u}{2N}, x = 0, 1, 2, \dots, N-1\}$ . To pomeni, da smo funkcijo  $f$ , ki je v fizičnem prostoru običajno podana kot nek signal, zapisali v frekvenčnem prostoru kosinusnih funkcij.

Definirajmo še DCT v matrični obliki, saj bomo to obliko kasneje potrebovali. Naj bo  $C_N$  matrika, kjer velja

$$[C_N]_{xu} = \left[ \alpha(u) \cos \frac{\pi(2x+1)u}{2N} \right], \quad x, u = 0, 1, 2, \dots, N-1. \quad (2.10)$$

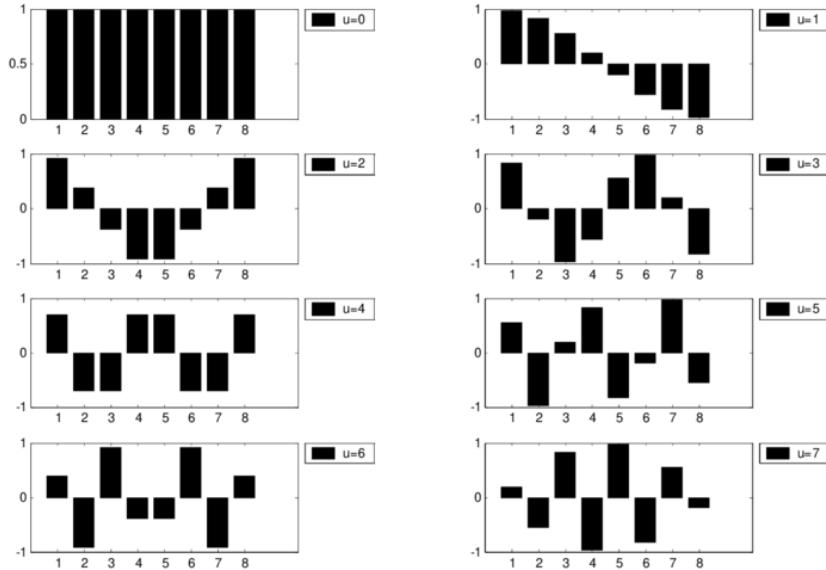
Zapišimo  $f(x)$  kot vrstični vektor  $f = [f(0) \ f(1) \ \dots \ f(N-1)]$ . Potem lahko izračunamo  $C = [C(0) \ C(1) \ \dots \ C(N-1)]$  kot  $C = fC_N$ . Iz (2.7) očitno sledi  $C(0) = \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} f(x)$ . Torej je prvi transformacijski koeficient povprečna vrednost vzorčnega zaporedja. V literaturi se ta vrednost običajno imenuje koeficient DC, vsi ostali koeficienti se imenujejo koeficienti AC<sup>2</sup>. Glavni razlog, da koeficient DC tako ločimo je pomembnost nizkih frekvenc. Od vseh ima koeficient DC najnižjo oziroma ničelno frekvenco. V splošnem bi v primeru zanemarjenja najvišjih frekvenc dobili relativno dober približek originalnega signala.

---

<sup>2</sup>Ta imena izhajajo iz zgodovinskih razlogov, ko se je DCT uporabljal za analiziranje električnih vezij z direktnimi in izmeničnimi tokovi.

Povejmo še nekaj o časovni zahtevnosti tega algoritma. Denimo, da imamo signal dolžine  $N$  in želimo izračunati njegov DCT. Če računamo po definiciji (2.7), moramo izračunati  $N$  koeficientov DCT. Za vsakega od teh koeficientov potrebujemo  $N - 1$  seštevanj. Za vsako od števil, ki jih seštejemo, potrebujemo prej nekaj seštevanj in množenj, a to za vsak posamezen koeficient pride le  $\mathcal{O}(1)$ . Torej je skupna računska zahtevnost  $\mathcal{O}(N^2)$ . Pri inverzni DCT pridemo po podobnem postopku do iste računske zahtevnosti. Kasneje bomo v poglavju 2.5 pokazali, kako se DCT izračuna hitreje.

Oglejmo si nekaj lastnosti baznih funkcij. Izris grafa  $h(x) = \cos\left[\frac{\pi(2x+1)u}{2N}\right]$  za  $N = 8$ , pri različnih vrednostih  $u$ , je prikazan na Sliki 2.1. V skladu s prejšnjimi opazkami je pri krivulji zgoraj levo ( $u = 0$ ) izrisana konstantna (DC) vrednost, medtem ko spremenljivke  $u = 1, 2, \dots, 7$  dajo krivulje z vedno višjimi frekvencami. Te krivulje se imenujejo kosinusne bazne funkcije. Pokažimo, da so bazne funkcije ortogonalne glede na običajen skalarni produkt, definiran na vektorjih. To je ena izmed ključnih lastnosti za izpeljavo hitrih in numerično stabilnih algoritmov za DCT.



Slika 2.1: Enodimensionalne kosinusne bazne funkcije pri različnih  $u$  ( $N = 8$ ). Na abscisi so vrednosti  $x$ , na ordinati pa vrednosti  $h(x)$ . Vir: [9].

**Izrek 2.1**  $C_N$  je ortogonalna matrika, torej velja  $C_N^{-1} = C_N^T$ .

*Dokaz.* Da se čim bolj izognemo trigonometriji, bomo ta dokaz naredili na indirekten način po [15]. Najprej razpišimo  $C_N$ :

$$C_N = \sqrt{\frac{2}{N}} \begin{bmatrix} u=0 & u=1 & \dots & u=N-1 \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{\pi}{N} \cdot \frac{1}{2}\right) & \dots & \cos\left(\frac{(N-1)\pi}{N} \cdot \frac{1}{2}\right) \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{\pi}{N} \cdot \frac{3}{2}\right) & \dots & \cos\left(\frac{(N-1)\pi}{N} \cdot \frac{3}{2}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{2}} & \cos\left(\frac{\pi}{N} \cdot \frac{2N-1}{2}\right) & \dots & \cos\left(\frac{(N-1)\pi}{N} \cdot \frac{2N-1}{2}\right) \end{bmatrix} \begin{array}{l} x=0 \\ x=1 \\ \vdots \\ x=N-1 \end{array} .$$

(2.11)

Naj bo  $A_N$  realna, simetrična tridiagonalna matrika, definirana kot

$$A_N = \begin{bmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \ddots & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{bmatrix} .$$

Označimo z  $v^{(u)}$  stolpec matrike  $C_N$  in z  $v_x^{(u)}$  element v  $x$ -ti vrstici in  $u$ -tem stolpcu. Pokazali bomo, da so  $v^{(u)}$  lastni vektorji matrike  $A_N$  za različne lastne vrednosti. Ker je  $A_N$  simetrična, bo iz tega sledilo, da je  $C_N$  ortogonalna.

Prvo poglejmo primer za  $u=0$ , torej je  $v^{(0)} = \frac{1}{\sqrt{N}} [1 \ 1 \ \dots \ 1]^T$ . Potem je

$$A_N v^{(0)} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \ddots & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} .$$

Iz tega sledi  $A_N v^{(0)} = 0v^{(0)}$ . Vidimo, da je  $v^{(0)}$  res lastni vektor  $A_N$ . Komponente preostalih stolpcov  $C_N$  so oblike

$$v_j^{(u)} = \sqrt{\frac{2}{N}} \cos\left(\frac{\pi u}{N}(j + \frac{1}{2})\right).$$

Pokazali bomo, da so vsi  $v^{(u)}$  lastni vektorji matrike  $A_N$ . To bomo naredili tako, da bomo pogledali, kaj dobimo, če vsako vrstico matrike  $A_N$  pomnožimo s stolpci  $v^{(u)}$ . Za enostavnejši zapis definirajmo še  $\theta = \frac{\pi u}{N}$  in  $B = \theta(x + \frac{1}{2})$ .

Za  $x = 1, \dots, n - 2$  je

$$\begin{aligned} (A_N v^{(u)})_x &= \sum_{j=0}^{n-1} (A_N)_{xj} v_j^{(u)} \\ &= -v_{x-1}^{(u)} + 2v_x^{(u)} - v_{x+1}^{(u)} \\ &= \sqrt{\frac{2}{N}} \left[ -\cos\left(\theta(x - \frac{1}{2})\right) + 2\cos\left(\theta(x + \frac{1}{2})\right) - \right. \\ &\quad \left. - \cos\left(\theta(x + \frac{3}{2})\right) \right] \\ &= \sqrt{\frac{2}{N}} \left[ -\cos\left(\theta(x + \frac{1}{2}) - \theta\right) + 2\cos\left(\theta(x + \frac{1}{2})\right) - \right. \\ &\quad \left. - \cos\left(\theta(x + \frac{1}{2}) + \theta\right) \right] \\ &= \sqrt{\frac{2}{N}} [-\cos(B - \theta) + 2\cos(B) - \cos(B + \theta)] \\ &= \sqrt{\frac{2}{N}} [-\cos B \cos \theta - \sin B \sin \theta + 2\cos B + \sin B \sin \theta] \\ &= \sqrt{\frac{2}{N}} [2 - 2\cos \theta] \cos B \\ &= \left[ 2 - 2\cos \frac{\pi u}{N} \right] v_x^{(u)}. \end{aligned}$$

Za  $x = 0$  je

$$\begin{aligned}
 (A_N v^{(u)})_0 &= v_0^{(u)} - v_1^{(u)} \\
 &= \sqrt{\frac{2}{N}} \left[ \cos\left(\frac{1}{2}\theta\right) - \cos\left(\frac{3}{2}\theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} \left[ \cos\left(\frac{1}{2}\theta\right) - \cos\left(\frac{1}{2}\theta + \theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} \left[ \cos\left(\frac{1}{2}\theta\right) - \cos\left(\frac{1}{2}\theta\right) \cos\theta + \sin\left(\frac{1}{2}\theta\right) \sin\theta \right] \\
 &= \sqrt{\frac{2}{N}} \left[ \cos\left(\frac{1}{2}\theta\right) - \cos\left(\frac{1}{2}\theta\right) \cos\theta + \right. \\
 &\quad \left. + \sin\left(\frac{1}{2}\theta\right) 2 \sin\left(\frac{1}{2}\theta\right) \cos\left(\frac{1}{2}\theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} \cos\left(\frac{1}{2}\theta\right) \left[ 1 - \cos\theta + 2 \sin^2\left(\frac{1}{2}\theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} [2 - 2 \cos\theta] \cos\left(\frac{1}{2}\theta\right) \\
 &= \left[ 2 - 2 \cos\frac{\pi u}{N} \right] v_0^{(u)}.
 \end{aligned}$$

Za  $x = N - 1$ :

$$\begin{aligned}
 (A_N v^{(u)})_{N-1} &= v_{N-1}^{(u)} - v_{N-2}^{(u)} \\
 &= \sqrt{\frac{2}{N}} \left[ \cos\left(\theta(N - \frac{1}{2})\right) - \cos\left(\theta(N - \frac{3}{2})\right) \right] \\
 &= \sqrt{\frac{2}{N}} \left[ (-1)^u \cos\left(\frac{1}{2}\theta\right) - (-1)^u \cos\left(\frac{1}{2}\theta + \theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} (-1)^u \left[ \cos\left(\frac{1}{2}\theta\right) - \cos\left(\frac{1}{2}\theta + \theta\right) \right] \\
 &= \sqrt{\frac{2}{N}} (-1)^u [2 - 2 \cos\theta] \cos\left(\frac{1}{2}\theta\right) \\
 &= \sqrt{\frac{2}{N}} (-1)^{2u} [2 - 2 \cos\theta] \cos\left(\theta(N - \frac{1}{2})\right) \\
 &= \sqrt{\frac{2}{N}} [2 - 2 \cos\theta] \cos\left(\theta(N - \frac{1}{2})\right) \\
 &= \left[ 2 - 2 \cos\frac{\pi u}{N} \right] v_{N-1}^{(u)}.
 \end{aligned}$$

Tukaj smo v 3. vrstici uporabili enačbi

$$\cos\left(\left(\frac{1}{2} + N - 1\right)\frac{u\pi}{N}\right) = (-1)^u \cos\left(\left(\frac{1}{2} + 0\right)\frac{u\pi}{N}\right)$$

in

$$\cos\left(\left(\frac{1}{2} + N - 2\right)\frac{u\pi}{N}\right) = (-1)^u \cos\left(\left(\frac{1}{2} + 1\right)\frac{u\pi}{N}\right),$$

ki izhajata iz simetrije funkcije cos. V 5. vrstici pa smo uporabili vmesni rezultat primera  $x = 0$ .

Pokazali bomo še po [4], da so stolpci matrike  $C_N$  normirani. To pomeni, da velja

$$\|v^{(u)}\|_2^2 = \sum_{x=0}^{N-1} (v_x^{(u)})^2 = 1, \quad x, u = 0, 1, \dots, N-1.$$

Poglejmo prvo primer za  $u = 0$ :

$$\|v^{(0)}\|_2^2 = \sum_{x=0}^{N-1} \sqrt{\frac{1}{N}}^2 = N \cdot \frac{1}{N} = 1.$$

Preden lahko gremo na ostale primere pokažimo, da velja

$$\sum_{x=0}^{N-1} \cos\left(\frac{2u\pi}{N}(x + \frac{1}{2})\right) = 0, \quad u = 1, \dots, N-1. \quad (2.12)$$

Da bomo to pokazali, uporabimo formulo

$$2 \cos \alpha = \exp(i\alpha) + \exp(-i\alpha), \quad (2.13)$$

kjer exp pomeni eksponentno funkcijo. Vstavimo (2.13) v levi del (2.12), kjer smo 2 dodali za lažji zapis:

$$\begin{aligned} & 2 \sum_{x=0}^{N-1} \cos\left(\frac{2u\pi}{N}(x + \frac{1}{2})\right) = \\ &= \sum_{x=0}^{N-1} \left[ \exp\left(\pi i \frac{2u}{N}(x + \frac{1}{2})\right) + \exp\left(-\pi i \frac{2u}{N}(x + \frac{1}{2})\right) \right] \\ &= \exp\left(\pi i \frac{2u}{2N}\right) \sum_{x=0}^{N-1} \left[ \exp\left(\pi i \frac{2u}{N}x\right) + \exp\left(-\pi i \frac{2u}{N}(x + 1)\right) \right] \\ &= \exp\left(\pi i \frac{2u}{2N}\right) \sum_{x=-N}^{N-1} \exp\left(\pi i \frac{2u}{N}x\right) = 0. \end{aligned}$$

Zadnji enačaj velja, ker je vsota geometrijska vrsta z  $2N$  členi in količnikom  $r = \exp(\pi i \frac{2u}{N})$ . Nas bodo zanimali samo primeri, ko je  $u = 1, \dots, N - 1$ . V teh primerih  $r \neq 1$ . Za geometrijske vsote, kjer  $r \neq 1$ , velja naslednja enačba

$$\sum_{x=m}^n ar^x = \frac{a(r^m - r^{n+1})}{1 - r}.$$

V našem primeru je  $m = -N$  in  $n = N - 1$ , kjer velja  $\exp(\pi i) = -1$ , torej je

$$\begin{aligned} \sum_{x=-N}^{N-1} ar^x &= \frac{a \left( \exp \left( \pi i \frac{2u}{N} \right)^{-N} - \exp \left( \pi i \frac{2u}{N} \right)^N \right)}{1 - \exp \left( \pi i \frac{2u}{N} \right)} \\ &= \frac{a \left( (-1)^{\frac{2u}{N}(-N)} - (-1)^{\frac{2u}{N}N} \right)}{1 - \exp \left( \pi i \frac{2u}{N} \right)} \\ &= \frac{a((-1)^{-2u} - (-1)^{2u})}{1 - \exp \left( \pi i \frac{2u}{N} \right)} \\ &= \frac{a(1 - 1)}{1 - \exp \left( \pi i \frac{2u}{N} \right)} = 0. \end{aligned}$$

Sedaj lahko izračunamo normo še za  $u = 1, \dots, N - 1$ :

$$\begin{aligned} \|v^{(u)}\|_2^2 &= \frac{2}{N} \sum_{x=0}^{N-1} \left( \cos \left( \frac{u\pi}{N} \left( x + \frac{1}{2} \right) \right) \right)^2 \\ &= \frac{2}{N} \sum_{x=0}^{N-1} \left( \frac{1 + \cos \left( 2 \cdot \frac{u\pi}{N} \left( x + \frac{1}{2} \right) \right)}{2} \right) \\ &= \frac{1}{N} \sum_{x=0}^{N-1} 1 + \frac{1}{N} \sum_{x=0}^{N-1} \cos \left( 2 \cdot \frac{u\pi}{N} \left( x + \frac{1}{2} \right) \right) \\ &= \frac{1}{N} \cdot N + \frac{1}{N} \sum_{x=0}^{N-1} \cos \left( 2 \cdot \frac{u\pi}{N} \left( x + \frac{1}{2} \right) \right) \\ &= 1 + \frac{1}{N} \sum_{x=0}^{N-1} \cos \left( 2 \cdot \frac{u\pi}{N} \left( x + \frac{1}{2} \right) \right) \\ &= 1. \end{aligned}$$

Zadnji enačaj velja zaradi (2.12).  $\square$

S tem, da smo dokazali ortogonalnost matrike  $C_N$ , smo dokazali tudi medsebojno ortogonalnost kosinusnih baznih funkcij.

## 2.4 Dvodimenzionalni DCT

Idejo enodimenzionalnega DCT bomo sedaj razširili na dve dimenziji. Po želji se lahko ta metoda razširi na poljubno število dimenzij. Nas višje dimenzije ne bodo zanimale, saj se bomo kasneje osredotočili na uporabo DCT pri slikah, kjer bomo delali z dvodimenzionalnimi podatki.

Definirajmo dvodimenzionalni DCT kot

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right],$$

$$u, v = 0, 1, 2, \dots, N-1,$$
(2.14)

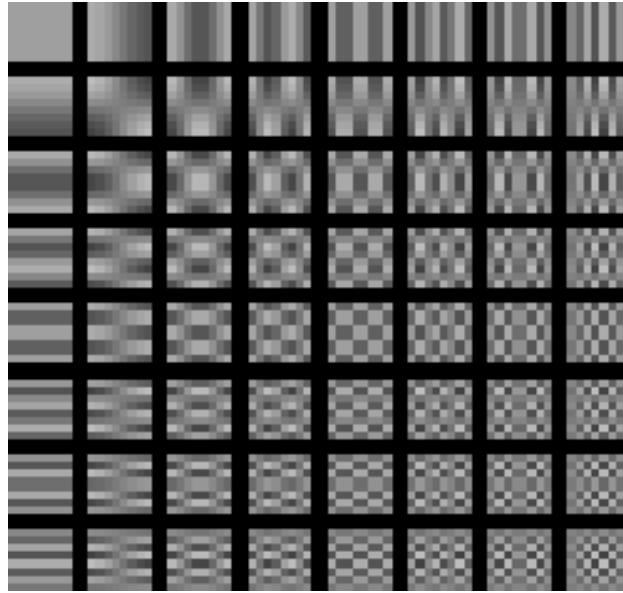
kar je direktna razširitev enodimenzionalnega primera. V tej definiciji sta  $\alpha(u)$  in  $\alpha(v)$  definirana enako kot v (2.9). Funkcija  $f(x, y)$  je realna funkcija, definirana za  $x, y = 0, 1, 2, \dots, N-1$ . Številom  $C(u, v)$  podobno kot v enodimenzionalnem problemu rečemo koeficienti DCT. V tem primeru imamo za vsako dimenzijo svojo spremenljivko, ki nam pove frekvenco baznih funkcij.

Inverzna transformacija je definirana kot

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v) C(u, v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right],$$

$$x, y = 0, 1, 2, \dots, N-1.$$
(2.15)

Bazne funkcije za  $N = 8$  so prikazane na Sliki 2.2. Opazimo, da se s pomikanjem po sliki navzdol večajo navpične frekvence, s pomikanjem desno pa vodoravne. Zgornja leva bazna funkcija je rezultat množenja komponente DC na Sliki 2.1 s svojo transponirano različico. Iz tega sledi, da je ta funkcija konstantna in ji prav tako pravimo koeficient DC. Vsem ostalim podobno rečemo koeficienti AC.



Slika 2.2: Dvodimenzionalne bazne funkcije DCT ( $N = 8$ ). Srednje siva barva predstavlja ničelno amplitudo, svetlo siva predstavlja pozitivno in temno siva negativno amplitudo. Vir: [6].

### 2.4.1 Separabilnost

Eden od mnogih razlogov za popularnost DCT je dejstvo, da lahko večdimenzionalne verzije DCT izračunamo z zaporednimi enodimenzionalnimi DCT-ji. To si bomo pogledali na primeru dveh dimenzij. Definicijo dvodimenzionalnega DCT iz (2.14) lahko zapišemo sledeče.

$$C(u, v) = \alpha(u) \sum_{x=0}^{N-1} \cos \left[ \frac{\pi(2x+1)u}{2N} \right] \left( \alpha(v) \sum_{y=0}^{N-1} f(x, y) \cos \left[ \frac{\pi(2y+1)v}{2N} \right] \right),$$

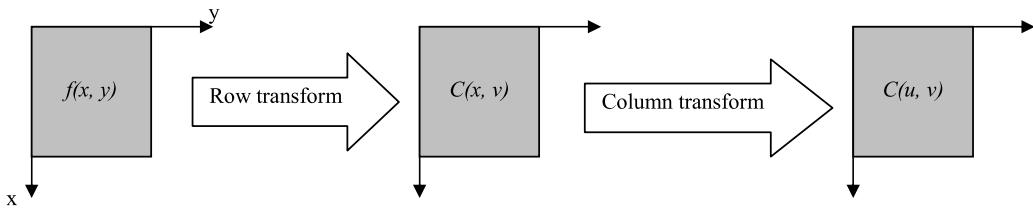
$$u, v = 0, 1, 2, \dots, N-1. \quad (2.16)$$

Definirajmo še

$$g(x, v) = \left( \alpha(v) \sum_{y=0}^{N-1} f(x, y) \cos \left[ \frac{\pi(2y+1)v}{2N} \right] \right).$$

Tukaj najprej izračunamo  $g(x, v)$ , kot da je enodimenzionalen DCT, nato pa vstavimo ta rezultat v levi del in ponovno izračunamo enodimenzionalen DCT kot  $C(u, v) = \alpha(u) \sum_{x=0}^{N-1} g(x, v) \cos \left[ \frac{\pi(2x+1)u}{2N} \right]$ . Tako smo pokazali, da lahko dvodimenzionalni DCT zapišemo kot dva enodimenzionalna DCT-ja.

Ta lastnost nam omogoča, da lahko  $C(u, v)$  izračunamo v dveh zaporednih enodimenzionalnih operacijah na vrsticah in stolpcih. Ideja je prikazana na Sliki 2.3. Podobno lahko izpeljemo enačbo (2.15) in pokažemo, da je inverzen DCT tudi separabilen.



Slika 2.3: Shema izračuna dvodimenzionalnega DCT z lastnostjo separabilnosti. Vir: [9].

## 2.5 Algoritem za hitri DCT

V tem razdelku bomo predstavili enega izmed algoritmov za hiter izračun DCT. Ta algoritem deluje na  $N$  elementih signala s časovno zahtevnostjo  $\mathcal{O}(N \log N)$ . Povzeli bomo algoritem za enodimenzionalen DCT opisan v [11]. Ta algoritem uporablja že obstoječe algoritme za hitro Fourierovo transformacijo (FFT) in njihove rezultate le prilagodi, da ustrezajo definiciji DCT. Obstajajo tudi alternativne izbire, ki ne uporabljajo FFT, temveč rekurzivno uporabljajo samo DCT-je. Pri teh algoritmih so potrebni tudi drugi tipi DCT-jev. Primer takšnega algoritma je opisan v [3]. Izbira implementacije je odvisna od strojnih zmogljivosti, saj so lahko različni algoritmi optimalni. Na primer strojne implementacije direktno na čipih uporabljajo drugačne algoritme kot pa implementacije v programski opremi na večnamenskih procesorjih.

Izpeljavo bomo naredili po [11]. Ta je rahlo prilagojena, saj je tam uporabljena nestandardna definicija DCT. Želimo izračunati DCT od  $f(x)$ , ki je zaporedje realnih števil dolžine  $N$ , torej je definirano za  $0 \leq x \leq N-1$ . DCT bomo izpeljali iz  $2N$ -točkovne diskretne Fourierove transformacije (DFT). Obstaja tudi izboljšava tega algoritma, kjer je potreben FFT na samo  $N$  točkah [11]. Definirajmo

$$g(x) = \begin{cases} f(x), & 0 \leq x \leq N-1 \\ f(2N-x-1), & N \leq x \leq 2N-1 \end{cases}. \quad (2.17)$$

Velja  $g(2N-x-1) = g(x)$ . Za  $g(x)$  je DFT definirana kot

$$G(u) = \sum_{x=0}^{2N-1} g(x) W_{2N}^{xu}, \quad (2.18)$$

kjer je  $W_{2N}$   $2N$ -ti koren enote, torej

$$W_{2N} = e^{-i2\pi/2N}. \quad (2.19)$$

Vstavimo definicijo  $g(x)$  iz (2.17) v (2.18) in dobimo

$$G(u) = \sum_{x=0}^{N-1} f(x) W_{2N}^{xu} + \sum_{x=N}^{2N-1} f(2N-x-1) W_{2N}^{xu}. \quad (2.20)$$

V [11] je izpeljano, da je to enako

$$G(u) = W_{2N}^{-u/2} 2 \sum_{x=0}^{N-1} f(x) \cos \frac{\pi(2x+1)u}{2N}, \quad 0 \leq u \leq 2N-1. \quad (2.21)$$

Vstavimo definicijo DCT iz (2.7) v (2.21) in dobimo

$$G(u) = W_{2N}^{-u/2} \frac{2}{\alpha(u)} C(u) \quad (2.22)$$

oziroma

$$C(u) = W_{2N}^{u/2} \frac{\alpha(u)}{2} G(u). \quad (2.23)$$

Za izračun DCT torej uporabimo  $2N$ -točkovni FFT od  $g(x)$ . Njegove rezultate nato vstavimo v (2.23) in dobimo DCT.

Opisan algoritmom velja za hitrega, saj za izračun DFT v (2.23) uporabimo FFT, ki ima za zaporedje dolžine  $N$  časovno zahtevnost  $\mathcal{O}(N \log N)$ . Razlaga časovne zahtevnosti FFT in njegova psevdokoda je na voljo v [10].

Tukaj smo imeli DFT na  $2N$  točkah, kar nam da časovno zahtevnost  $\mathcal{O}(2N \log 2N)$  ozziroma poenostavljeno zapisano  $\mathcal{O}(N \log N)$ . Pri vsakem koeficientu potrebujemo še nekaj množenj, deljenj in izračun  $W_{2N}^{-u/2}$ , kar skupaj za vse koeficiente naredimo v  $\mathcal{O}(N)$  korakih. Skupna časovna zahtevnost hitrega DCT je torej  $\mathcal{O}(N \log N)$  za signal dolžine  $N$ . Z uporabo tega algoritma smo torej prišli s časovne zahtevnosti  $\mathcal{O}(N^2)$  na  $\mathcal{O}(N \log N)$ , kar je velika izboljšava.

Sedaj potrebujemo še hiter inverzni DCT. Izpeljemo ga na podoben način kot DCT, le da tukaj uporabimo inverzni DFT. Inverzni DFT od  $G(u)$  je definiran kot

$$g(x) = \frac{1}{2N} \sum_{u=0}^{2N-1} G(u) W_{2N}^{-xu}. \quad (2.24)$$

Ker je  $g(x)$  realna funkcija, je posledično njen DFT hermitsko simetričen, torej

$$G(2N - u) = \overline{G}(u). \quad (2.25)$$

Oznaka  $\overline{G}(u)$  pomeni konjugirano kompleksno število. Potrebujemo tudi, da po (2.21) zaradi  $\cos\left(\frac{1}{2} + k\pi\right) = 0 \forall k \in \mathbb{N}$  velja

$$G(N) = 0. \quad (2.26)$$

Če vstavimo (2.25) in (2.26) v (2.24), dobimo

$$g(x) = \frac{1}{N} \operatorname{Re} \left[ \frac{G(0)}{2} + \sum_{u=1}^{N-1} G(u) W_{2N}^{-xu} \right], \quad 0 \leq x \leq 2N - 1. \quad (2.27)$$

Če vstavimo (2.22) v (2.27) in uporabimo (2.17), dobimo inverzni DCT, kot je zapisan v (2.8).

Torej če imamo dan  $C(u)$ , dobimo  $f(x)$  tako, da prvo izračunamo  $G(u)$  z inverznim FFT. Nato s pomočjo (2.27) dobimo  $g(x)$  in iz njega pridobimo  $f(x)$  po (2.17).

Ta algoritem ima tudi časovno zahtevnost  $\mathcal{O}(N \log N)$ , ker deluje zelo podobno kot DCT. Zraven izračuna inverznega DFT ima še nekaj množenj, deljenj in izračun  $W_{2N}^{-u/2}$ , kar skupaj za vse koeficiente naredimo v  $\mathcal{O}(N)$  korakih. Nato uporabimo inverzen FFT, ki ima časovno zahtevnost  $\mathcal{O}(N \log N)$ .

Opisali smo algoritem v eni dimenziji, kjer potrebujemo FFT na  $2N$  točkah. Ta algoritem lahko posplošimo na dve dimenziji tako, da uporabimo lastnost separabilnosti, ki smo jo pokazali v razdelku 2.4.1. Obstajajo tudi podobni algoritmi, ki računajo direktno na dveh dimenzijah. Primer takega algoritma je opisan v [11]. Pomembno je omeniti tudi, da je ta algoritem za izračun DCT numerično stabilen, kar je pokazano v [12].

## 2.6 Celoštivilski DCT

V tem razdelku si bomo ogledali aproksimacijo DCT s celimi števili. Te vrste transformacij so zaželene zaradi enostavnejše aritmetike pri računanju s celimi števili. Implementacija, ki si jo bomo tu ogledali, uporablja množenje in seštevanje celih števil. Obstajajo tudi implementacije, ki uporabljajo zgolj binarno seštevanje in operacijo bitnega premika (*angl. bit shift*). Pri slednjih se običajno dosežejo boljši rezultati v primeru, da uporabljamo števila z več biti. Posledično se sicer poveča računska zahtevnost.

Predstavili bomo eno izmed enostavnnejših implementacij celoštivilskega DCT, ki je opisana v [3]. Ista knjiga vsebuje tudi naprednejše implementacije, v katere se tukaj ne bomo poglabljali. Pri tej implementaciji bomo predstavili korake, kako pretvoriti realno štivilski DCT v celoštivilskega. Zaradi lažje razlage si bomo ogledali primer, ko je  $N = 8$ . V splošnem je ta primer tudi najpogosteje uporabljen. Želimo aproksimirati DCT matriko  $C_8$ , ki smo jo definirali v (2.10), s

$$\widetilde{C}_8 = Q_8 V_8. \quad (2.28)$$

V tem zapisu  $Q_8$  predstavlja diagonalno matriko z normalizacijskimi faktorji na glavni diagonali. Ti niso nujno celoštevilski, ampak jih potrebujemo, da bo baza v DCT ortonormirana.

V eksplisitnem zapisu  $C_8$  opazimo, da ima matrika le 7 različnih vrednosti. Zapišimo elemente z enako magnitudo z isto spremenljivko iz množice  $\{a, b, c, d, e, f, g\}$ , kjer so  $a, b, c, d, e, f, g \in \mathbb{N}$ . Definirajmo  $V_8$  tako, da ohramimo istoležne predznake kot pri  $C_8$ . Na mestih, kjer imajo števila v  $C_8$  enake magnitude, uporabimo isto spremenljivko. Izrazimo  $\widetilde{C}_8$  po tej definiciji:

$$\widetilde{C}_8 = Q_8 \begin{bmatrix} g & a & e & b & g & c & f & d \\ g & b & f & -d & -g & -a & -e & -c \\ g & c & -f & -a & -g & d & e & b \\ g & d & -e & -c & g & b & -f & -a \\ g & -d & -e & c & g & -b & -f & a \\ g & -c & -f & a & -g & -d & e & -b \\ g & -b & f & d & -g & a & -e & c \\ g & -a & e & -b & g & -c & f & -d \end{bmatrix}. \quad (2.29)$$

Želimo, da sta  $i$ -ti in  $j$ -ti bazni vektor ortogonalna. Označimo  $i$ -ti in  $j$ -ti stolpec  $V_8$  kot  $v^{(i)}$  in  $v^{(j)}$ . Ta vektorja sta ortogonalna, če je njun skalarni produkt 0, torej

$$(v^{(i)})(v^{(j)})^T = \sum_{k=0}^7 v_k^{(i)} v_k^{(j)} = 0, \quad \forall i \neq j, \quad i, j = 0, 1, \dots, 7.$$

Za ortogonalnost matrike potrebujemo še, da je norma stolpcov  $\widetilde{C}_8$  enaka 1. Definirajmo še

$$q_{ii} = \|v^{(i)}\|_2^2 = \sum_{k=0}^7 (v_k^{(i)})^2, \quad i = 0, 1, \dots, 7. \quad (2.30)$$

Sedaj lahko definiramo matriko  $Q_8$  kot

$$Q_8 = \text{diag} \left( \frac{1}{\sqrt{q_{00}}}, \dots, \frac{1}{\sqrt{q_{77}}} \right). \quad (2.31)$$

Ta definicija zagotovi, da bo  $\widetilde{C}_8$  ortogonalna. Členi  $\frac{1}{\sqrt{q_{ii}}}$  so v splošnem iracionalna števila. Temu se ne moramo izogniti, če želimo imeti ortogonalni DCT. Pogoj za ortogonalnost stolpcov lahko zapišemo tudi kot

$$V_8 V_8^T = [Q_8^{-1}]^2. \quad (2.32)$$

Iz tega pogoja pridobimo enačbe

$$a(b - c) - d(b + c) = 0 \quad (2.33)$$

in

$$\begin{aligned} q_{00} &= q_{44} = 8g^2, \\ q_{11} &= q_{33} = q_{55} = q_{77} = 2(a^2 + b^2 + c^2 + d^2), \\ q_{22} &= q_{66} = 4(e^2 + f^2). \end{aligned} \quad (2.34)$$

Iz teh pogojev vidimo, da lahko spremenljivke  $e$ ,  $f$  in  $g$  obravnavamo ločeno od  $a$ ,  $b$ ,  $c$  in  $d$ .

Določimo še nekaj neenakosti za te spremenljivke, ki veljajo za koeficiente  $C_8$ :

$$a > b > c > d > 0, \quad (2.35)$$

$$a > e > f > 0, \quad (2.36)$$

$$a > g > 0. \quad (2.37)$$

Lahko bi zapisali tudi kako neenakost več ali manj, vendar s temi dobimo kar dobre rezultate.

Želimo najti množico celih števil  $\{a, b, c, d, e, f, g\}$ , ki zadoščajo enakosti (2.33) in neenakostim (2.35)–(2.37). Za rešitev tega problema lahko iščemo števila v množici  $\{a, b, c, d, e, f, g\}$ , ki so predstavljena z  $M$  biti. V splošnem je tukaj lahko neskončno rešitev enačbe s temi omejitvami. DCT s celimi števili je določena s  $\{a, b, c, d, e, f, g\}$ , kar bomo označili kot  $IntDCT_8(a, b, c, d, e, f, g)$ . Izbrano množico  $\{a, b, c, d, e, f, g\}$  bomo vstavili v matriko  $V_8$ , kot smo definirali v (2.29). Podobno vstavimo vse  $q_{ii}$  dane v (2.34) v  $Q_8$ . Vse rešitve, ki upoštevajo enačbo (2.33) in omejitve (2.35)–(2.37), nam bodo dale matriko  $\widetilde{C}_8$ , ki je ortogonalna in blizu matriki  $C_8$ .

Efektivnost implementacije  $IntDCT_8(a, b, c, d, e, f, g)$  bo odvisna predvsem od izbire velikosti koeficientov, torej kako velik  $M$  bomo dovolili.

Tukaj bomo podali nekaj primerov za  $e = 3$ ,  $g = 1$  in  $f = 1$ . Pogledali si bomo torej nekaj primerov  $IntDCT_8(a, b, c, d, 3, 1, 1)$ , kjer bomo za  $M$  vzeli vrednosti 3, 4, 5, 6, 7 ali 8. Za mero točnosti specifičnega približka pa bomo uporabili mero povprečne kvadratne napake, ki jo označimo z MSE (*angl. Mean squared error*) in je definirana kot

$$\text{MSE}(X, \tilde{X}) = \frac{1}{mn} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (X_{ij} - \tilde{X}_{ij})^2. \quad (2.38)$$

Tukaj je  $X$  originalna matrika in  $\tilde{X}$  približek originalni matriki.

V Tabeli 2.1 bomo merili MSE različnih verzij  $IntDCT_8(a, b, c, d, 3, 1, 1)$  v primerjavi z originalnim DCT.

$IntDCT_8(a, b, c, d, 3, 1, 1)$	MSE
$DCT$	0
$IntDCT_8(5, 3, 2, 1, 3, 1, 1)$	2.721681e-003
$IntDCT_8(10, 9, 6, 2, 3, 1, 1)$	2.060647e-004
$IntDCT_8(15, 12, 8, 3, 3, 1, 1)$	2.059246e-004
$IntDCT_8(25, 21, 14, 5, 3, 1, 1)$	1.302862e-004
$IntDCT_8(45, 39, 26, 9, 3, 1, 1)$	1.380974e-004
$IntDCT_8(55, 48, 32, 11, 3, 1, 1)$	1.458331e-004
$IntDCT_8(65, 57, 38, 13, 3, 1, 1)$	1.524257e-004
$IntDCT_8(120, 105, 70, 24, 3, 1, 1)$	1.492797e-004
$IntDCT_8(175, 153, 102, 35, 3, 1, 1)$	1.481648e-004
$IntDCT_8(230, 201, 134, 46, 3, 1, 1)$	1.475946e-004

Tabela 2.1: Tabela različnih možnih  $IntDCT_8(a, b, c, d, 3, 1, 1)$  s pripadajočimi MSE od originalnega DCT.

Izpišimo še matriko  $\widetilde{C}_8$  za primer  $IntDCT_8(15, 12, 8, 3, 3, 1, 1)$

$$\widetilde{C}_8 = Q_8 \begin{bmatrix} 1 & 15 & 3 & 12 & 1 & 8 & 1 & 3 \\ 1 & 12 & 1 & -3 & -1 & -15 & -3 & -8 \\ 1 & 8 & -1 & -15 & -1 & 3 & 3 & 12 \\ 1 & 3 & -3 & -8 & 1 & 12 & -1 & -15 \\ 1 & -3 & -3 & 8 & 1 & -12 & -1 & 15 \\ 1 & -8 & -1 & 15 & -1 & -3 & 3 & -12 \\ 1 & -12 & 1 & 3 & -1 & 15 & -3 & 8 \\ 1 & -15 & 3 & -12 & 1 & -8 & 1 & -3 \end{bmatrix}, \quad (2.39)$$

kjer so  $q_{ii}$  dani kot

$$\begin{aligned} q_{00} &= q_{44} = 1, \\ q_{11} &= q_{33} = q_{55} = q_{77} = 884, \\ q_{22} &= q_{66} = 10, \end{aligned} \quad (2.40)$$

torej je

$$Q_8 = \text{diag} \left( \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{442}}, \frac{1}{\sqrt{40}}, \frac{1}{\sqrt{442}}, \frac{1}{\sqrt{8}}, \frac{1}{\sqrt{442}}, \frac{1}{\sqrt{40}}, \frac{1}{\sqrt{442}} \right). \quad (2.41)$$

Tukaj smo zapisali samo matriko, s katero bi morali množiti vektor vrednosti signala, da bi dobili DCT. V knjigi [3] so zapisane še izboljšave tega algoritma. Opisan je tudi način, kako lahko tak DCT izračunamo v podobnem številu korakov, kot smo pokazali pri hitri verziji navadnega DCT.

# Poglavlje 3

## JPEG

Format JPEG je opisan v standardu ISO/IEC 10918. Ime je dobil po skupini, Joint Photographic Experts Group, ki ga je ustvarila. Izdan je bil leta 1992 v [7].

JPEG standard določa zgolj ogrodje formata, ki poda nekatere zahteve in predloge za uporabo. Ponuja veliko število opcijskih implementacij, ki so odvisne od namenjene uporabe in od potrebe po moči kompresije. Tukaj bomo predstavili podrobnejše zgolj eno izmed popularnejših izvedb tega formata. Opis delovanja JPEG-a je povzet po [7, 17].

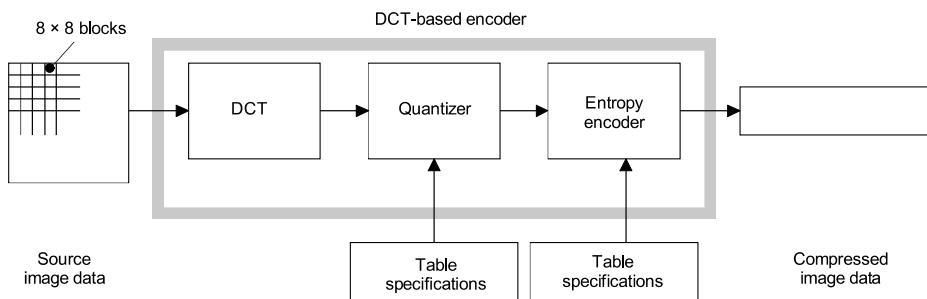
JPEG podpira štiri različne načine delovanja. Prve trije od teh uporablajo kompresijo z izgubo, zadnja pa uporablja kompresijo brez izgub.

- Zaporedno kodiranje (*angl. Sequential encoding*): slika je kodirana od leve proti desni in od vrha do dna.
- Progresivno kodiranje (*angl. Progressive encoding*): slika je kodirana v več obhodih za aplikacije, kjer je čas prenosa dolg. Tukaj ob dekodiranju takoj dobimo nizko resolucijo slike, ki z obhodi pridobiva na ločljivosti.
- Hierarhično kodiranje (*angl. Hierarchical encoding*): slika je kodirana pri več različnih resolucijah. Na ta način lahko dostopamo do nižjih resolucij, ne da bi morali dekompresirati sliko pri polni resoluciji.

- Kodiranje brez izgub (*angl. Lossless encoding*): slika je kodirana tako, da ob dekodiranju dobimo identično kopijo originala (ta način delovanja daje zelo minimalno kompresijo).

Vsak od teh načinov delovanja ima specificiranih več različnih kodekov (par kodirnika in dekodirnika). Od sedaj naprej bomo govorili zgolj o zaporednem kodiranju, saj je to najpogostejša implementacija v današnjih programih.

Na Sliki 3.1 je prikazana osnovna shema kodiranja slike v formatu JPEG. Predstavili bomo vse potrebne procese, da bomo znali preslikati sliko v



Slika 3.1: Shema kodiranja JPEG-a. Vir: [17] Copyright © 1992, IEEE.

končno bitno zaporedje, ki se shrani in pošilja med napravami. V prvem delu bomo opisali kako se obravnavajo barvni kanali. Za to se običajno uporablja barvno podvzorčenje (*angl. chroma subsampling*). O podvzorčenju ne bomo šli v podrobnosti, saj so za nas zgolj pomembni prihranki. Več informacij je na voljo v [16]. Delali bomo na 8x8 blokih slike, kot je prikazano na Sliki 3.1. Na teh blokih bomo izvedli DCT, ki ga nato kvantiziramo (*angl. quantization*), torej odstranijo se manj pomembni podatki. Pokazali bomo, kako se ti podatki pripravijo za kodiranje. Na njih bomo izvedli entropijsko kodiranje (*angl. entropy encoding*) s Huffmanovim kodiranjem. Specifike tega kodiranja niso zelo pomembne za to diplomsko delo, saj bomo uporabili vnaprej izračunano tabelo za kodiranje, ki je izračunana za povprečne bloke slike. Več informacij o Huffmanovem kodiranju je na voljo v [1]. Na koncu kodiranja bomo dobili niz bitov, ki predstavlja blok slike.

### 3.1 Predprocesiranje

Osnovni format JPEG sam po sebi ne določi veliko o barvnih kanalih. Lahko ga uporabljam za slike v sivinah (*angl. grayscale images*), kar je najenostavenjši primer, saj imajo zgolj en barvni kanal. Ta barvni kanal se lahko takoj pošlje na naslednji korak. Pogosteje se uporablja barvne slike, ki imajo tri barvne kanale. Najpogostejši od teh zapisov je oblike RGB, kjer imamo po en kanal za rdečo, zeleno in modro intenziteto. Te bi lahko obravnavali kot neodvisne in jih ločeno poslali dalje ter jih šele ob dekodiranju nazaj združili. Uspešnejši pristop za višjo kompresijo je, da jih preslikamo v barvni prostor, kjer se loči svetilnost (*angl. luminance*) in barvo (*angl. chrominance*). Tukaj se slika preslika iz prostora RGB v prostor  $Y'C_B C_R$ , kjer  $Y'$  predstavlja svetilnost, medtem ko  $C_B$  in  $C_R$  predstavlja barvo. To naredimo, ker je človeško oko občutljivejše na svetilnost kot na barvo. Na tem prostoru se izvede barvno podvzorčenje, preden pošljemo sliko dalje. Najpogosteje se uporablja barvno podvzorčenje 4:2:2 ali 4:2:0. Prvi iz barvnih kanalov odstrani 1/2 podatkov, torej, če gledamo celotne podatke, tu odstranimo 1/3. V drugem primeru odstranimo 3/4 barvnih podatkov, torej v celoti 1/2 podatkov. Tu se zmanjša resolucija drugih dveh kanalov, ko jih pošljemo dalje.

Preden razdelimo sliko na bloke, je potrebno dodati dovolj pikslov, da bo število pikslov v vodoravni in navpični smeri deljivo z 8. V standardu ni specificirano, katero metodo moramo uporabiti. Ena od pogostejših implementacij je, da zrcalimo najbolj robne piksle (*angl. Reflection padding*). To sliko nato razdelimo na bloke velikosti 8x8. Označimo vrednosti v teh blokih s  $f(x, y)$ , kjer je  $x, y = 0, 1, 2, \dots, 7$ . Izbira velikosti bloka je 8x8 zaradi hitre izračunljivosti in zelo dobrih rezultatov pri kompresiji ter vizualni ločljivosti slik. V teh blokih nato vrednosti zamaknemo iz nepredznačenih celih števil na intervalu  $[0, 2^P - 1]$  v predznačena cela števila na intervalu  $[-2^{P-1}, 2^{P-1} - 1]$ . Tukaj  $P$  označuje število bitov, ki predstavljajo intenziteto piksla v tem barvnem kanalu. Običajno je  $P = 8$ . To pomeni, če označimo z  $z$  vrednost piksla, izračunamo zamaknjen  $z$  kot  $z' = z - 2^{P-1}$ .

## 3.2 Kodiranje

Pri kodiranju gredo 8x8 bloki skozi DCT, torej se uporabi formula (2.14), kjer je  $N = 8$ . Poenostavljeno zapisano kot

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right],$$

$$u, v = 0, 1, 2, \dots, 7, \quad (3.1)$$

kjer je

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}} & \text{za } u = 0 \\ \sqrt{\frac{1}{4}} & \text{za } u \neq 0 \end{cases}. \quad (3.2)$$

V praksi se uporablajo hitrejše metode za izračun DCT, kot na primer algoritem, ki smo ga predstavili v poglavju 2.5. V formatu JPEG ni predpisani specifičen način izračuna DCT, saj so pri različnih aplikacijah različne implementacije optimalne in lahko privedejo do različnih rezultatov. Iz tega razloga predpisi zahtevajo le, da je algoritem znotraj meja njihovega testa natančnosti.

Drugi večji korak je kvantizacija izhoda DCT, to je zmanjševanje vpliva manj pomembnih koeficientov tako, da te koeficiente delimo z večimi števili. Elementi so kvantizirani s pomočjo kvantizacijske tabele velikosti 8x8, ki je specificirana od uporabnika ali aplikacije. Elemente te tabele označimo s  $Q(u, v)$ . To so cela števila med 1 in 255, ki določajo, koliko se kvantizira pripadajoč DCT koeficient. Kvantizirani koeficienti so izračunani s pomočjo naslednje formule:

$$C^Q(u, v) = \text{Integer round}\left(\frac{C(u, v)}{Q(u, v)}\right). \quad (3.3)$$

To storimo za vse  $u, v = 0, \dots, 7$ .

Ta korak je glavni vir izgube znotraj JPEG-a. Iz tega razloga je komprezija končne slike odvisna predvsem od izbire kvantizacijskih tabel. Primer takšnih tabel je podan v [7]. Tam so kot izbiro podali Tabelo 3.1 za svetilnost

in Tabelo 3.2 za barvo. Opazimo, da sta sicer tabeli različni, vendar obe temeljita na principu, da se števila, ko gremo desno in navzdol, višajo. Tabele so tako zgrajene, ker nam podatki desno in spodaj predstavljajo bazne funkcije z višjimi frekvencami. Te so manj pomembne za vizualno ločljivost in s tem njihov pomen zmanjšamo. Uporabljene kvantizacijske tabele se shranijo v glavi JPEG-a, saj se bodo kasneje potrebovale za dekodiranje.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	12
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

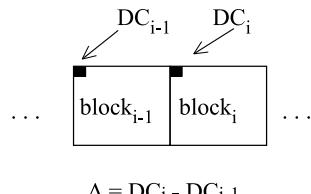
Tabela 3.1: Kvantizacijska tabela za svetilnost.

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tabela 3.2: Kvantizacijska tabela za barvo.

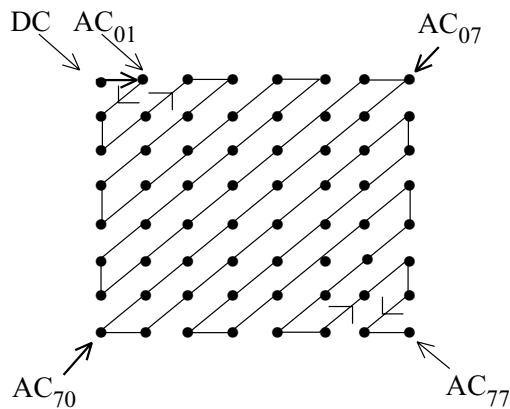
Po kvantizaciji se koeficient DC kodira ločeno od 63 koeficientov AC. To se stori, ker so pogosto koeficienti DC sosednjih blokov korelirani in ker vsebujejo večino informacij slike. Kvantizirani koeficienti DC so kodirani kot razlika prejšnjega in trenutnega koeficiente DC, kot je prikazano na Sliki 3.2.

Pri prvem bloku se prejšnji koeficient šteje, kot da je 0, kar je povprečna vrednost, saj smo vhodne vrednosti centrirali okoli 0.



Slika 3.2: Diferencialno DC kodiranje. Vir: [17] Copyright © 1992, IEEE.

Preostanejo koeficienti AC, ki jih najprej preuredimo v enovrstično zaporedje po zaporedju, prikazanem na Sliki 3.3. S to razporeditvijo skupaj združimo koeficiente, ki predstavljajo nižje frekvence. Koeficienti na začetku zaporedja imajo večjo verjetnost, da so neničelni. To je koristno za entropijsko kodiranje (*angl. entropy coding*), ki ga bomo izvedli na teh zaporedjih.



Slika 3.3: Zaporedje kodiranja koeficientov AC. Vir: [17] Copyright © 1992, IEEE.

Sedaj, ko imamo zaporedje 63 koeficientov AC, jih lahko nadalje preoblikujemo za entropijsko kodiranje. Ta postopek izvede kompresijo podatkov brez izgub. Uporabijo se statistične lastnosti kvantiziranih koeficientov DCT. V JPEG standardu sta predloženi dve metodi entropijskega kodiranja, to sta

Huffmanovo kodiranje in aritmetično kodiranje. Mi bomo opisali zgolj Huffmanovo kodiranje, saj se to uporablja v privzetem kodeku.

V tem diplomskem delu bomo zgolj opisali uporabo že obstoječih tabel za kodiranje. Te tabele niso specificirane v standardu in je zato pri aplikacijah več možnih izbir. Lahko se uporablja ista tabela za vse slike ali pa se te tabele izračunajo dinamično pred kodiranjem slike. Kot možna izbira je v JPEG standardu podana ena tabela za koeficiente DC in ena tabela za koeficiente AC. JPEG omogoča tudi uporabo različnih tabel za različne barvne kanale, vendar dovoljeni sta največ dve DC tabeli in dve AC tabeli. Vse tabele, ki se uporabijo, je potrebno shraniti v glavo JPEG-a, da se lahko kasneje uporabijo tudi pri dekodiranju.

Preden lahko uporabimo Huffmanovo kodiranje, moramo pripraviti podatke v primerno obliko za kodiranje, ki jo imenujmo vmesna predstavitev. Najprej jo pripravimo za koeficiente AC. Zapisali jih bomo kot dva simbola, ki ju poimenujemo  $S_1$  in  $S_2$ . V  $S_1$  bomo zapisali dve števili. Prvo od teh števil bo predstavljal število zaporednih ničel v našem zaporedju od prejšnjega neničelnega koeficiente. Drugo nam bo povedalo najmanjše število potrebnih bitov za predstavitev trenutnega koeficiente AC.  $S_2$  bo vseboval vrednosti neničelnega koeficiente AC. Zapisali ju bomo kot  $S_1 = (R, S)$  in  $S_2 = (V)$ .

Povedati moramo še nekaj omejitev teh koeficientov.  $R$ , ki predstavlja število zaporednih ničel, lahko zavzame vrednosti od 0 do 15. V primeru, da je zaporednih ničel več, zapišemo  $S_1 = (15, 0)$ , s čimer povemo, da imamo 16 zaporednih ničel. Dalje zapisujemo enako, kot da bi bil 16. koeficient neničelno število. V najbolj ekstremnem primeru so lahko zapored trije zapisi  $(15, 0)$ , če imamo zapored 48 ali več ničel. Vsakemu  $S_1$  sledi v končnem zapisu en  $S_2$ . Edina izjema temu je, ko je zadnji 63. koeficient enak 0, kar se zgodi zelo pogosto. V tem primeru se zapiše za zadnji neničelen koeficient le  $S_1$  z vsebino  $(0, 0)$ . V tem primeru se  $S_2$  spusti. Torej  $S_1 = (0, 0)$  je oznaka za EOB (*angl. end of block*), kar pomeni konec bloka.

Omejimo se na najobičajnejše 8-bitne barvne kanale, torej koeficiente  $V$  lahko predstavimo z 8 biti. V [17] je pokazano, da se lahko DCT koefici-

enti povečajo kvečjemu za 3 bite. Kvantizacija tega ne poveča, saj lahko delimo zgolj s števili večjimi ali enakimi ena. Naš originalni interval števil je  $[-2^7, 2^7 - 1]$ , torej so naši kvantizirani koeficienti AC kvečjemu znotraj  $[-2^{10}, 2^{10} - 1]$ . Iz tega vidimo, da bomo potrebovali za zapis  $V$  v  $S_2$  kvečjemu 10 bitov. Uporabi se zapis celih števil spremenljive dolžine (*angl. variable-length integer*), kar bomo označili kot VLI. V tej obliki se lahko vsa števila iz zgornjega intervala zapišejo binarno z 1 do 10 biti. Iz tega sledi, da bo  $S$  v  $S_1$  vsebovala števila od 1 do 10. Spremenljivko  $S$  izračunamo s pomočjo Tabele 3.3. Tukaj vrednost 0 ni mogoča, ker se vse ničle zapišejo z zaporedji ničel.

$S$	koeficienti AC
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1023,...,-512,512,...,1023

Tabela 3.3: Tabela za zapis velikosti koeficientov AC.

Sedaj naredimo še vmesno predstavitev koeficientov DC. Pozorni moramo biti, da tukaj uporabljam  $\Delta = DC_i - DC_{i-1}$  in ne le trenutni koeficient DC. Ta predstavitev je zgrajena podobno kot pri koeficientih AC, ponovno imamo dva simbola. V prvem je tokrat le  $S_1 = (S)$ , ker verjetnost, da dobimo 0 za razliko sosednjih koeficientov DC, ni dovolj visoka, da bi se splačalo dodati vrednost za zaporedno število ničel.  $S_2$  je podoben kot v prvem primeru. Razlika je, da so vrednosti razlike dveh koeficientov DC lahko na intervalu  $[-2^{11}, 2^{11} - 1]$ , saj lahko pri razliki dveh 10-bitnih števil dobimo 11-bitno

število. Za zapis  $S$  uporabimo Tabelo 3.4.

$S$	vrednosti $\Delta$
0	0
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1 023,...,-512,512,...,1 023
11	-2 047,...,-1 024,1 024,...,2 047

Tabela 3.4: Tabela za zapis velikosti koeficientov DC.

Ko smo izračunali koeficiente AC in DC v vmesno predstavitev, lahko začnemo kodirati podatke v obliko za pošiljanje.  $S_1$  se kodira s Huffmanovimi kodi. Običajno so tabele za Huffmanove kode vnaprej določene, lahko pa se računajo za vsako sliko posebej. V Tabeli 3.5 je prikazan izsek tabele, ki je podana v [7] kot predlog za kodiranje svetilnosti koeficientov AC. Podali smo zgolj primera, ko je  $R$  enak 0 ali 1, saj je celotna tabela 8-krat daljša. S pomočjo Huffmanovih tabel preslikamo  $S_1$  v neko binarno kodno besedo, ki je lahko različnih dolžin.

$S_2$  bomo zapisali v obliki VLI. VLI je fiksno določen v JPEG standardu in deluje na sledeč način. Recimo, da želimo zakodirati število  $n$ . Če je  $n \geq 0$ , izračunamo njegovo binarno reprezentacijo. Iz te reprezentacije vzamemo  $S$  najmanj pomembnih bitov (*angl. least significant bits*), torej zadnjih  $S$  bitov. Če je  $n \leq 0$ , izračunamo binarno reprezentacijo nepredznačenega  $n$  in vzamemo njen eniški komplement. Ponovno vzamemo  $S$  najmanj pomembnih bitov od izračunanega eniškega komplementa. Ta proces naredimo,

$R/S$	Dolžina koda	Kodna beseda
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	111111110000010
0/A	16	111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000

Tabela 3.5: Izsek Huffmanove kodirne tabele za svetilnost koeficientov AC.

da učinkovito predstavimo koeficiente, ki vemo, da so manjši od  $2^S$ . Lahko zapišemo tudi tabelo za izračun VLI, ki je prikazana v Tabeli 3.6. Kasneje bomo na primeru pokazali, kako se kodiranje več koeficientov združi zaporedno.

$S$	$V$	Predstavitev VLI
1	-1, 1	0, 1
2	-3, -2, 2, 3	00, 01, 10, 11
3	-7, ..., -4, 4, ..., 7	000, ..., 011, 100, ..., 111
4	-15, ..., -8, 8, ..., 15	0000, ..., 0111, 1000, ..., 1111
5	-31, ..., -16, 16, ..., 31	00000, ..., 01111, 10000, ..., 11111
6	-63, ..., -32, 32, ..., 63	000000, ..., 011111, 100000, ..., 111111
7	-127, ..., -64, 64, ..., 127	0000000, ..., 0111111, 1000000, ..., 1111111
8	-255, ..., -128, 128, ..., 255	00000000, ..., 01111111, 10000000, ..., 11111111
9	-511, ..., -256, 256, ..., 511	000000000, ..., 011111111, 100000000, ..., 111111111
10	-1023, ..., -512, 512, ..., 1023	0000000000, ..., 0111111111, 1000000000, ..., 1111111111
11	-2047, ..., -1024, 1024, ..., 2047	00000000000, ..., 01111111111, 10000000000, ..., 11111111111

Tabela 3.6: Reprezentacija celih števil v VLI.

Pokazali bomo še, kako se kodira koeficiente DC. Najprej kodiramo  $S_1$ . To je enostavnejše kot v AC primeru, saj nimamo števila zaporednih ničel  $R$  in je kodirna tabela zato veliko manjša. Primer tabele za svetilnost, ki je podana v [7], je v Tabeli 3.7. Za  $S_2$  uporabimo enak način kodiranja z VLI kot pri koeficientih AC. Pri koeficientih DC lahko pride do specifičnega primera, ko je koeficient, ki ga kodiramo, enak 0. V tem primeru je  $S = (0)$  in se  $S_2$  praktično ne kodira oziroma kodiranje nič ne vrne. Dovolj informacij smo že zapisali s tem, da je  $S = (0)$ .

$S$	Dolžina koda	Kodna beseda
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Tabela 3.7: Huffmanova kodirna tabela za svetilnost koeficientov DC.

### 3.3 Kvaliteta kompresiranih slik

Opisali bomo nekaj okvirnih pravil za kompresijo pri določeni kvaliteti. Če imamo barvno sliko s srednje zahtevnimi oblikami, velja:

- 0,25–0,5 bitov na piksel nam prinese srednjo do dobro kvaliteto, kar je primerno za nekatere aplikacije.
- 0,5–0,75 bitov na piksel nam prinese dobro do zelo dobro kvaliteto, kar je primerno za veliko aplikacij.
- 0,75–1,5 bitov na piksel nam prinese odlično kvaliteto, kar je primerno za večino aplikacij.
- 1,5–2 bita na piksel nam prinese skoraj neločljivo kvaliteto od originala, kar je primerno tudi za najzahtevnejše aplikacije.

Te mere večinoma veljajo, vendar se nanje ne moremo vedno zanašati. JPEG ni format, kjer bi pri kompresiji določili količino prostora za shranjevanje

slike. Namesto tega navedemo želeno kakovost in sistem nam vrne sliko s potrebno količino podatkov, da to dosežemo.

### 3.4 Primer kodiranja

S pomočjo programa v [14] smo izračunali primer kodiranja bloka  $I$  izmišljene slike, kjer je

$$I = \begin{bmatrix} 182 & 161 & 152 & 164 & 165 & 171 & 188 & 191 \\ 160 & 157 & 169 & 176 & 179 & 177 & 181 & 189 \\ 173 & 176 & 179 & 180 & 187 & 182 & 169 & 175 \\ 190 & 186 & 183 & 176 & 182 & 184 & 169 & 164 \\ 185 & 195 & 189 & 178 & 176 & 183 & 180 & 167 \\ 185 & 190 & 189 & 185 & 177 & 175 & 177 & 170 \\ 184 & 171 & 182 & 185 & 180 & 174 & 172 & 174 \\ 175 & 186 & 179 & 178 & 180 & 182 & 181 & 184 \end{bmatrix}.$$

Zamaknimo ta blok za  $2^7$  na interval  $[-2^7, 2^7 - 1]$ . Označimo z  $I'$  tak zamaknjen blok

$$I' = I - 128 = \begin{bmatrix} 54 & 33 & 24 & 36 & 37 & 43 & 60 & 63 \\ 32 & 29 & 41 & 48 & 51 & 49 & 53 & 61 \\ 45 & 48 & 51 & 52 & 59 & 54 & 41 & 47 \\ 62 & 58 & 55 & 48 & 54 & 56 & 41 & 36 \\ 57 & 67 & 61 & 50 & 48 & 55 & 52 & 39 \\ 57 & 62 & 61 & 57 & 49 & 47 & 49 & 42 \\ 56 & 43 & 54 & 57 & 52 & 46 & 44 & 46 \\ 47 & 58 & 51 & 50 & 52 & 54 & 53 & 56 \end{bmatrix}.$$

Izračunajmo DCT tega bloka

$$C(I') = \begin{bmatrix} 399,1 & 3,5 & -0,7 & 4,2 & 0,9 & 0,3 & 1,8 & 0 \\ -20,6 & -27,3 & 9,3 & 6,1 & 9,8 & 2,3 & 1,1 & 1 \\ -15,1 & -35 & 15,4 & -4,2 & 14,7 & 5,6 & -3,1 & -0,5 \\ -2,7 & 3,8 & 15,3 & -0,6 & -0,9 & 12 & -5,2 & 0,3 \\ 3,4 & 4,7 & 16,6 & 10 & -8,9 & -1,9 & -4 & -2,8 \\ -4,8 & 10,9 & 3,4 & 9,1 & 10,1 & 4,1 & 4,1 & 1,1 \\ 5 & 5,7 & 3,9 & 0,3 & -3,3 & -4,4 & -4,6 & -2,5 \\ 0,2 & -0,4 & 0 & 0,3 & -0,2 & 0 & 0 & -0,3 \end{bmatrix}.$$

Sedaj kvantiziramo ta blok s kvantizacijsko tabelo za svetilnost prikazano v Tabeli 3.1 in dobimo

$$C^Q(I') = \begin{bmatrix} 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

To zapišemo v enovrstično zaporedje, kot je prikazano na Sliki 3.3. Torej ločeno zapišemo koeficient DC in 63 koeficientov AC. V tem primeru velja, da je DC = 25 in

$$\text{AC} = \begin{bmatrix} 0, & -2, & -1, & -2, & 0, & 0, & 1, & -3, & 0, & 0, & 0, & 1, & 0, & 0, & 0, & 0, \\ 0, & 1, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}.$$

Sedaj, ko imamo koeficiente v pravilnem zaporedju, jih zapišemo v vmesni predstavitev. V tem primeru bomo računali, kot da je to prvi blok slike,

kar pomeni, da je prejšnji koeficient DC enak 0. Najprej izračunamo  $\Delta = 25 - 0 = 25$ . Iz Tabele 3.4 razberemo, da je  $S = 5$ , kar pomeni, da bo vmesna predstavitev koeficiente DC  $(5)(25)$ .

Pretvorimo še koeficiente AC v vmesno predstavitev. Prvi člen je 0, zato bo  $R = 1$ . Drugi člen je  $-2$ , kar preslikamo v  $S = 2$  glede na Tabelo 3.3. Tako dobimo zapis  $(1, 2)(-2)$ . Nadaljujemo na podoben način in dobimo celotno zaporedje simbolov za ta blok kot

$$(5)(25), (1, 2)(-2), (0, 1)(-1), (0, 2)(-2), \\ (2, 1)(1), (0, 2)(-3), (3, 1)(1), (5, 1)(1), (0, 0).$$

Sedaj začnemo dejansko kodirati v zapis za pošiljanje. Začnemo s koeficientom DC, ki ga po Tabeli 3.7 preslikamo iz  $(5)$  v  $110$ . Nato po VLI preslikamo vrednost  $(25)$ . Najprej zapišemo binarno reprezentacijo, ki je  $00011001$ . Vzamemo  $S$  najmanj pomembnih bitov, torej v tem primeru 5 bitov. Ostane nam  $11001$ . Pri tem uporabljamo 8-bitno reprezentacijo, v splošnem je priporočljivo uporabljati 16-bitno reprezentacijo, da pokrijemo vse možne vrednosti. Ta dva zapisa združimo in dobimo kodiranje  $11011001$  za koeficient DC.

Nadaljujemo s kodiranjem koeficientov AC. Najprej bomo poiskali vsa kodiranja za  $S_1$  v razširjeni Tabeli 3.5 v [7]. To so

$(1, 2)$	11011
$(0, 1)$	00
$(0, 2)$	01
$(2, 1)$	11100
$(3, 1)$	111010
$(5, 1)$	1111010
$(0, 0)$	1010.

Pretvorimo še vse  $S_2$  po VLI. Najprej pretvorimo  $-2$ . Tukaj zapišemo 2 v binarno reprezentacijo, ki je  $00000010$ . Naredimo eniški komplement tega števila, ki je  $11111101$ . Vzamemo njenih  $S$  najmanj pomembnih bitov, torej

2 bita in dobimo 01. Podobno naredimo za ostala števila

(-2)	01
(-1)	0
(1)	1
(-3)	00.

Sedaj po vrsti uporabimo izračunano in dobimo

(5)(25),	110 11001
(1, 2)(-2)	11011 01
(0, 1)(-1)	00 0
(0, 2)(-2)	01 01
(2, 1)(1)	11100 1
(0, 2)(-3)	01 00
(3, 1)(1)	111010 1
(5, 1)(1)	1111010 1
(0, 0)	1010.

To zapored skupaj napišemo v en sam binaren niz (presledki so dodani za lažje branje)

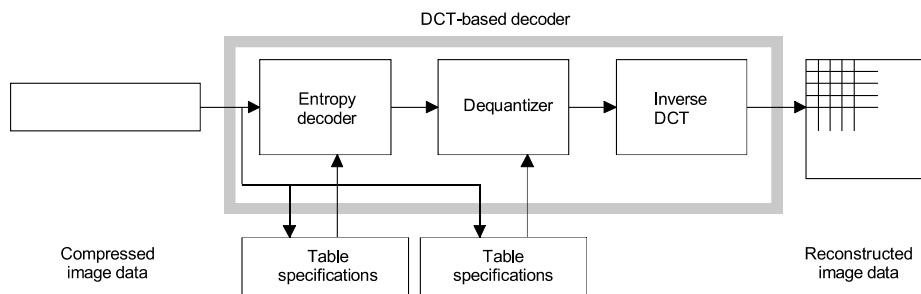
11011001 11011010 00010111 10010100 11101011 11101011 010.

Tako smo uspeli 8x8 blok zakodirati v 51 bitov. Blok vsebuje 64 pikslov, pri čemer vsak od teh pikslov vsebuje 8 bitov informacije. Zmanjšali smo torej podatke iz 512 bitov na 51 bitov. V tem primeru smo podatke stisnili za 10-krat. O dejanskem prihranku na sliki tukaj ne moramo povedati veliko, saj potrebujemo vzorec več blokov. Če gledamo le sliko v sivinah, smo sicer dosegli kompresijo na 0,8 bitov na piksel za ta blok.

### 3.5 Dekodiranje

V tem razdelku bomo pokazali, kako pridemo iz zakodiranega zaporedja bitov nazaj do slike, ki je blizu originalu.

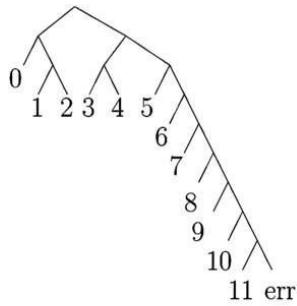
Shema, kako poteka dekodiranje, je prikazana na Sliki 3.4. Prvi korak je entropijsko dekodiranje, kjer tabele za dekodiranje prebremo iz glave JPEG-a. Izhod dekodiranja preoblikujemo nazaj v 8x8 blok. Ta gre skozi dekvantizacijo s pomočjo tabel, ki so zapisane v glavi JPEG-a. Na tej tabeli izvedemo inverzni DCT, ki nam da blok, ki ga še samo za  $2^7$  zamaknemo. V primeru več barvnih kanalov moramo istoležne bloke združiti nazaj v naš približek originalne slike.



Slika 3.4: Shema dekodiranja JPEG-a. Vir: [17] Copyright © 1992, IEEE

Podan imamo binaren niz bloka. V tem nizu ne vemo, kako dolga je kodirna beseda za  $S_1$ . Tukaj uporabimo lastnost Huffmanovega koda, da lahko beremo bite po vrsti, da dobimo zakodiran simbol. Za uporabo te lastnosti moramo iz tabele, ki smo jo prebrali v glavi JPEG-a, zgraditi dvojiško drevo. To drevo izgleda tako, da če po vrsti beremo bite, vsaka 0 pomeni, da gremo levo po drevesu, vsaka 1 pa pomeni, da gremo desno po drevesu. To deluje, ker so Huffmanovi kodi zgrajeni tako, da nobena kodirna beseda ne mora biti prefiks drugi daljši kodirni besedi.

Prvi simbol se nanaša na koeficient DC, zato prvo uporabimo zgrajeno drevo za tabelo koeficientov DC prikazano na Sliki 3.5. V našem primeru je to za Tabelo 3.7. Začnemo brati binarni niz in ga porabimo toliko, da



Slika 3.5: Binarno drevo za Huffmanovo dekodiranje koeficientov DC po Tabeli 3.7. Vir: [8] Copyright © 2011, IEEE.

pridemo po drevesu do lista. Dobili smo  $S_1 = (S)$  in s tem informacijo, da naslednjih  $S$  bitov predstavlja razliko dveh koeficientov DC. Sedaj vzamemo naslednjih  $S$  bitov, ki so števila zakodirana po VLI. Uporabiti moramo naslednjo proceduro, da pridemo iz VLI do celih števil.

Označimo bitno zaporedje z  $b$ , kjer je  $b_i$  i-ti člen tega zaporedja. Definirajmo binToInt kot funkcijo, ki preslika binarno število v celo število.

- Če je  $b_0 = 0$ , potem vemo, da gre za negativno število. V tem primeru moramo izračunati eniški komplement  $b$ , ki ga označimo z  $b'$ . Dobimo  $V = -\text{binToInt}(b')$ .
- Če je  $b_0 = 1$ , potem vemo, da gre za pozitivno število. V tem primeru je  $V = \text{binToInt}(b)$ .

Da dobimo originalen koeficient DC, moramo izračunati še  $DC_i = \Delta + DC_{i-1}$ , torej prištejemo prejšnji koeficient DC. Če je ta blok prvi, potem odštejemo 0. Število  $DC_i$  damo na svoje mesto levo zgoraj v 8x8 bloku.

Sledijo zakodirani vsi koeficienti AC. Te bomo zaporedoma zapisovali v seznam, dokler jih ne bo 63. Podobno kot v DC primeru, začnemo brati binarni niz. Zgradimo drevo za razširjeno Tabelo 3.5. Sledimo drevesu kot piše v binarnem nizu, dokler ne pridemo do lista v drevesu. Ko pridemo do lista, dobimo zakodirano  $S_1 = (R, S)$ . Sedaj moramo obravnavati nekaj primerov:

- Če je  $R = 0$  in  $S = 0$ , potem smo prišli do EOB in moramo naš seznam koeficientov AC napolniti s toliko ničlami, da bo vseboval 63 števil.
- Če je  $R = 15$  in  $S = 0$ , moramo v seznam koeficientov AC zapisati 16 ničel.
- V vseh ostalih primerih zapišemo v seznam koeficientov AC  $R$  ničel. Vzamemo naslednjih  $S$  bitov, ki predstavljajo zakodirano število po VLI. To število dekodiramo na isti način, kot je bilo opisano pri koeficientu DC, in ga dodamo na seznam.

Do konca bloka pridemo v primeru, ko je  $R = 0$  in  $S = 0$  ali ko imamo v seznamu zapisanih 63 koeficientov. Če še nismo na koncu bloka, ponovimo postopek za naslednji koeficient.

Na koncu tega postopka smo dobili seznam 63 koeficientov AC, te moramo razporediti nazaj v blok  $8 \times 8$ . To storimo na enak način kot pri kodiranju, le da sedaj dodajamo koeficiente po vrstnem redu na Sliki 3.3.

Naslednji korak je dekvantizacija. Ta poteka tako, da preberemo tabelo za kvantizacijo iz glave JPEG-a. Približke originalnim DCT koeficientom izračunamo po naslednji formuli:

$$C'(u, v) = C^Q(u, v) \cdot Q(u, v). \quad (3.4)$$

To storimo za vse  $u, v = 0, \dots, 7$ .

Na tej tabeli se sedaj izvede inverzni DCT. Koeficientom, ki smo jih dobili, moramo še prišteti zamik, da bodo ponovno na intervalu  $[0, 2^P - 1]$ . Označimo z  $\tilde{z}'$  zamaknjen približek piksla. Potem je naš končen približek piksla enak  $\tilde{z} = \tilde{z}' + 2^{P-1}$ .

Če osnovna resolucija slike ni bila deljiva z 8, moramo na tej točki odstraniti še piksle, ki smo jih v tistem koraku dodali. V primeru, da smo imeli sliko z več barvnimi kanali in smo uporabili barvno podvzorčenje, moramo še uporabiti metode, da kanale, ki predstavljajo barvo, razširimo nazaj na originalno velikost in v prostor RGB.

### 3.6 Primer dekodiranja

Sedaj bomo dekodirali blok, ki smo ga v poglavju 3.4 zakodirali. Imamo dano bitno zaporedje

110110011101101000010111100101001110101111101011010.

Izrišimo si drevo za Tabelo 3.7, kot je prikazano na Sliki 3.5. Sedaj gremo po drevesu skladno z bitnim zaporedjem (za vsako 0 gremo levo, za vsako 1 gremo desno). Do lista pridemo pri 110, ki pripada  $S_1 = (5)$ . Iz tega vidimo, da moramo vzeti naslednjih 5 bitov za koeficient DC. To je  $b = 11001$ , ki je v obliki VLI in ga pretvorimo nazaj v celo število. Najprej pogledamo, da je  $b_0 = 1$ , torej je pozitivno število, in izračunamo  $\text{binToInt}(11001) = 25$ . Sedaj moramo še prišteti prejšnji koeficient DC, ki je bil 0, ker smo to vzeli za prvi blok. Torej je  $\text{DC} = 25 + 0 = 25$ .

Nadaljujemo s koeficienti AC. Izrišemo si graf za celo razširjeno Tabelo 3.5 iz [7]. Dobimo 11011, ki pripada  $S_1 = (1, 2)$ . Prvo zapišemo eno ničlo v zaporedje. Za koeficient vzamemo naslednja dva bita  $b = 01$ . Vidimo, da je  $b_0 = 0$ , torej je to negativno število, in izračunamo eniški komplement  $b' = 10$ . Dobimo koeficient  $-\text{binToInt}(10) = -2$ . Podobno nadaljujemo in dekodiramo v naslednje koeficiente.

Huffmanove besede	$(R, S)$	VLI	$V$	Dodani koeficienti
11011	$(1, 2)$	01	- 2	0, - 2
00	$(0, 1)$	0	- 1	- 1
01	$(0, 2)$	01	- 2	- 2
11100	$(2, 1)$	1	1	0, 0, 1
01	$(0, 2)$	00	- 3	- 3
111010	$(3, 1)$	1	1	0, 0, 0, 1
1111010	$(5, 1)$	1	1	0, 0, 0, 0, 0, 1
1010	$(0, 0)$			46-krat 0

Prvi stolpec predstavlja uporabljena binarna zaporedja, da smo dobili  $(R, S)$ , ki je predstavljen v drugem stolpcu. V tretjem stolpcu je število bitov, ki jih zaseda število v formatu VLI. To število v četrtem stolpcu pretvorimo v koeficient, ki ga dodamo na konec seznama. V petem stolpcu so zapisani vsi koeficienti, ki so prišli iz dekodiranja para simbolov. Zapišimo sedaj vse koeficiente po vrsti.

Dobimo DC = 25 in

$$\text{AC} = \begin{bmatrix} 0, & -2, & -1, & -2, & 0, & 0, & 1, & -3, & 0, & 0, & 0, & 1, & 0, & 0, & 0, & 0, \\ 0, & 1, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix},$$

kar je enako, kot smo zakodirali.

Sedaj, ko imamo to zaporedje, ga zapišemo nazaj v 8x8 blok po Sliki 3.3.  
Dobimo nazaj naslednji 8x8 blok

$$C^Q(\tilde{I}') = \begin{bmatrix} 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Ta blok moramo dekvantizirati po (3.4) s Tabelo 3.1, saj smo to tabelo upo-

rabili za kvantizacijo. Dobimo

$$C(\tilde{I}') = \begin{bmatrix} 400 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -24 & -24 & 14 & 0 & 0 & 0 & 0 & 0 \\ -14 & -39 & 16 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 22 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Na tej matriki izvedemo inverzni DCT, kjer rezultat zaokrožimo na najblžje celo število. Dobimo

$$\tilde{I}' = \begin{bmatrix} 40 & 36 & 31 & 30 & 36 & 47 & 60 & 69 \\ 40 & 40 & 39 & 41 & 44 & 49 & 54 & 57 \\ 44 & 47 & 51 & 53 & 53 & 50 & 46 & 44 \\ 54 & 56 & 58 & 59 & 56 & 49 & 43 & 38 \\ 62 & 61 & 59 & 56 & 52 & 48 & 44 & 42 \\ 62 & 60 & 56 & 53 & 50 & 49 & 48 & 48 \\ 54 & 54 & 53 & 53 & 53 & 52 & 51 & 51 \\ 45 & 48 & 52 & 55 & 56 & 55 & 53 & 51 \end{bmatrix}.$$

Ta približek je še zamknjen za  $2^7$ , zato mu dodamo še  $2^7$  in dobimo končni približek originalnemu bloku

$$\tilde{I} = \begin{bmatrix} 168 & 164 & 159 & 158 & 164 & 175 & 188 & 197 \\ 168 & 168 & 167 & 169 & 172 & 177 & 182 & 185 \\ 172 & 175 & 179 & 181 & 181 & 178 & 174 & 172 \\ 182 & 184 & 186 & 187 & 184 & 177 & 171 & 166 \\ 190 & 189 & 187 & 184 & 180 & 176 & 172 & 170 \\ 190 & 188 & 184 & 181 & 178 & 177 & 176 & 176 \\ 182 & 182 & 181 & 181 & 181 & 180 & 179 & 179 \\ 173 & 176 & 180 & 183 & 184 & 183 & 181 & 179 \end{bmatrix}.$$

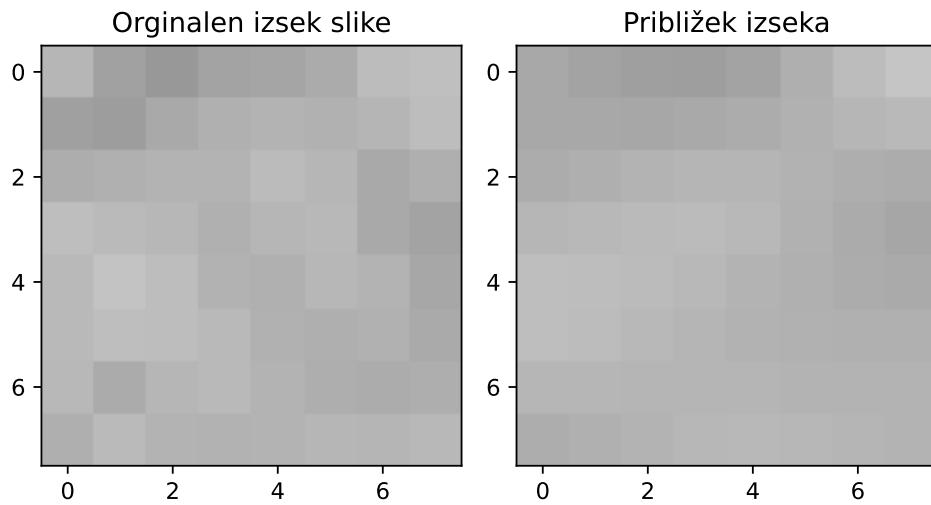
Originalen blok zgleda sledeče

$$I = \begin{bmatrix} 182 & 161 & 152 & 164 & 165 & 171 & 188 & 191 \\ 160 & 157 & 169 & 176 & 179 & 177 & 181 & 189 \\ 173 & 176 & 179 & 180 & 187 & 182 & 169 & 175 \\ 190 & 186 & 183 & 176 & 182 & 184 & 169 & 164 \\ 185 & 195 & 189 & 178 & 176 & 183 & 180 & 167 \\ 185 & 190 & 189 & 185 & 177 & 175 & 177 & 170 \\ 184 & 171 & 182 & 185 & 180 & 174 & 172 & 174 \\ 175 & 186 & 179 & 178 & 180 & 182 & 181 & 184 \end{bmatrix}.$$

Če izračunamo razliko, vidimo, koliko napake je nastalo:

$$I - \tilde{I} = \begin{bmatrix} 14 & -3 & -7 & 6 & 1 & -4 & 0 & -6 \\ -8 & -11 & 2 & 7 & 7 & 0 & -1 & 4 \\ 1 & 1 & 0 & -1 & 6 & 4 & -5 & 3 \\ 8 & 2 & -3 & -11 & -2 & 7 & -2 & -2 \\ -5 & 6 & 2 & -6 & -4 & 7 & 8 & -3 \\ -5 & 2 & 5 & 4 & -1 & -2 & 1 & -6 \\ 2 & -11 & 1 & 4 & -1 & -6 & -7 & -5 \\ 2 & 10 & -1 & -5 & -4 & -1 & 0 & 5 \end{bmatrix}.$$

Največja razlika med originalom in približkom je 14, kar ni veliko. Oba bloka sta prikazana na Sliki 3.6. Opazimo, da sta bloka precej podobna. Na desnem vidimo, da je zglajena verzija levega, torej ni zelo hitrih sprememb v svetlosti. Ta razlika je zares vidna le takrat, ko je ta del slike zelo povečan. V praksi je takšna razlika skoraj neopazna za človeško oko.



Slika 3.6: Prikaz blokov pred in po kodiranju z JPEG-om. Levo je prikazan originalen blok in na desni je blok po rekonstrukciji. Vir: generirano s kodo v [14].

## Poglavlje 4

# Uporaba diskretne kosinusne transformacije

V tem poglavju si bomo pogledali dejansko uporabo DCT v kombinaciji z JPEG standardom. Ovrednotili bomo kvaliteto slik s pomočjo PSNR (*angl. Peak signal-to-noise ratio*).

### 4.1 Primeri kompresije slik v JPEG-u

Ogledali si bomo tri slike, ki jih bomo nato pretvorili v JPEG s 5 različnimi nivoji kvalitete. Pretvarjali jih bomo s knjižnico PIL v programskejem jeziku Python. Za različne kvalitete bomo nastavili spremenljivko *quality* v funkciji *save* na 5 različnih vrednosti. Izbrali smo vrednosti 5, 25, 50, 75 in 95.

Za mero kakovosti slik bomo uporabili mero PSNR, ki meri razmerje med največjo možno močjo signala in močjo nezaželenega šuma. PSNR je definirana v [5] kot

$$\text{PSNR}(X, \tilde{X}) = 10 \log_{10} \frac{255^2}{\text{MSE}(X, \tilde{X})}. \quad (4.1)$$

Tukaj je z  $X$  označena originalna slika in z  $\tilde{X}$  kompresiran približek  $X$ . Za funkcijo MSE je uporabljena ista definicija kot v (2.38). Enota PSNR je *dB*. Ta definicija je zapisana za primer, ko imamo 8-bitno sliko, torej je možnih

255 pozitivnih vrednosti piksla. V drugih primerih je potrebno prilagoditi definicijo številu bitov. Pri slikah v sivinah je ta definicija enolična. Ko gledamo PSNR na barvnih slikah, sta dve možni definiciji. Lahko vzamemo povprečje MSE na treh barvnih kanalih ali izračunamo MSE na svetilnosti, torej na Y komponenti, če imamo sliko zapisano v  $Y'C_BC_R$ . Mi bomo uporabili definicijo, kjer vzamemo povprečje treh barvnih kanalov.

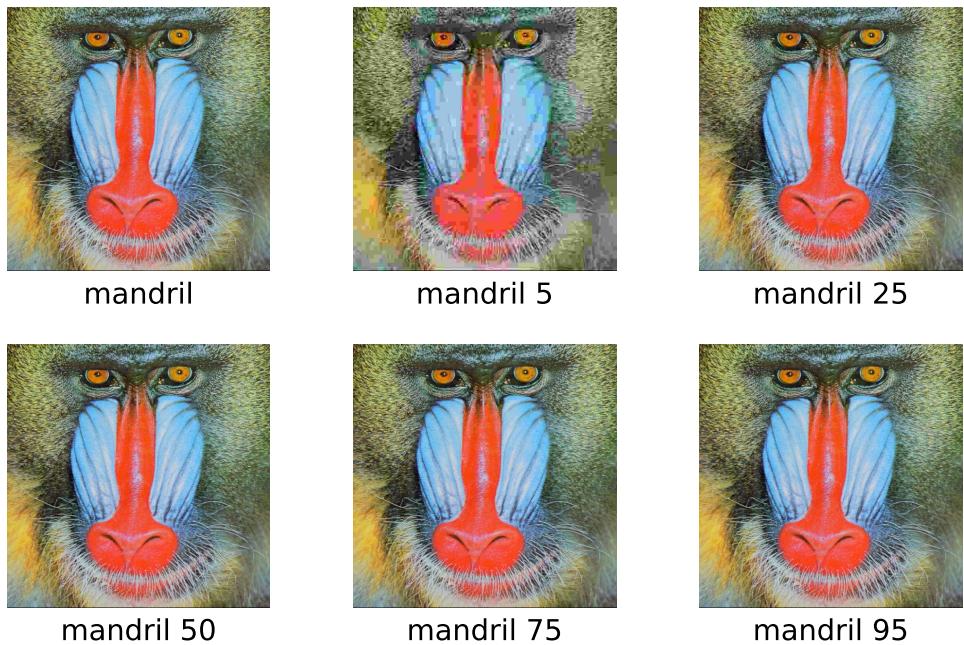
Ko gledamo vrednosti PSNR, nam višje vrednosti predstavljajo boljšo kvaliteto. V splošnem velja, da za 8-bitne slike je PSNR nad 25 že običajno soliden, nad 30 pa zelo dober.

Izbrali smo tri slike, ki se pogosto uporabljajo v procesiranju slik. Imenujejo se mandril, peppers in splash. Te slike imajo resolucijo 512x512 in tri barvne kanale. Njihovi originali so v formatu TIFF. Obstajajo sicer formati, ki hranijo sliko brez izgube kvalitete, ki zavzamejo manj prostora kot TIFF, vendar je TIFF široko uporabljan in zelo konsistenten.

Slike smo pretvorili v JPEG s petimi različnimi nivoji kvalitete in izračunali njihov PSNR. Za sliko mandril so rezultati prikazani na Sliki 4.1, PSNR ter velikosti datotek v Tabeli 4.1. Podobno imamo za sliko peppers na Sliki 4.2 in v Tabeli 4.2 ter za sliko splash na Sliki 4.3 in v Tabeli 4.3.

Opazimo, da imajo originali  $\text{PSNR} = \infty$ , kar se zgodi, ker je MSE slike same s sabo enak 0. Velikost datotek originalov je pri treh slikah enaka, ker so bile generirane z enakimi nastavtvami v formatu TIFF.

V splošnem velja pravilo, da je pri izbiri kvalitete 95 skoraj neopazna izguba kvalitete slike. Privzeta izbira v tej implementaciji je sicer 75, kar je tudi zelo težko opazno. Kvalitete manjše od 50 imajo običajno že vidne artefakte kompresije. Te bi se morale uporabljati zgolj za aplikacije, kjer je majhna velikost slik nujna. Pri teh ocenah moramo biti sicer previdni, saj JPEG dobro deluje le na naravnih oblikah, ki jih dobimo npr. s fotografiranjem okolice. Kasneje bomo pokazali tudi primer, ko JPEG ni primeren format za shranjevanje slike.



Slika 4.1: Mandril prikazan v različnih kvalitetah. Levo zgoraj je original v formatu TIFF. Ostali so v formatu JPEG, kjer številka zraven imena predstavlja nivo kvalitete nastavljen v knjižnici PIL za generiranje JPEG-a. Vir: original [13] in ostale generirane s [14].

Ime slike in kvaliteta	PSNR	velikost datoteke
mandril original	$\infty$	769 kB
mandril 5	24,77	10 kB
mandril 25	30,50	32 kB
mandril 50	31,97	50 kB
mandril 75	33,23	76 kB
mandril 95	34,91	186 kB

Tabela 4.1: Tabela z vrednostmi PSNR in velikostmi datotek za različne kakovosti slike mandril. Prva vrstica tabele predstavlja original v formatu TIFF, ostale vrstice predstavljajo kompresirane slike v JPEG s kvaliteto, ki je zapisana v prvem stolpcu.



Slika 4.2: Peppers prikazan v različnih kvalitetah. Levo zgoraj je original v formatu TIFF. Ostali so v formatu JPEG, kjer številka zraven imena predstavlja nivo kvalitete nastavljen v knjižnici PIL za generiranje JPEG-a. Vir: original [13] in ostale generirane s [14].

Ime slike in kvaliteta	PSNR	velikost datoteke
peppers original	$\infty$	769 kB
peppers 5	19,90	8 kB
peppers 25	23,50	17 kB
peppers 50	24,85	26 kB
peppers 75	26,21	41 kB
peppers 95	28,85	122 kB

Tabela 4.2: Tabela z vrednostmi PSNR in velikostmi datotek za različne kakovosti slike peppers. Prva vrstica tabele predstavlja original v formatu TIFF, ostale vrstice predstavljajo kompresirane slike v JPEG s kvaliteto, ki je zapisana v prvem stolpcu.



Slika 4.3: Splash prikazan v različnih kvalitetah. Levo zgoraj je original v formatu TIFF. Ostali so v formatu JPEG, kjer številka zraven imena predstavlja nivo kvalitete nastavljen v knjižnici PIL za generiranje JPEG-a. Vir: original [13] in ostale generirane s [14].

Ime slike in kvaliteta	PSNR	velikost datoteke
splash original	$\infty$	769 kB
splash 5	23,14	7 kB
splash 25	28,04	13 kB
splash 50	29,25	20 kB
splash 75	30,29	32 kB
splash 95	32,07	88 kB

Tabela 4.3: Tabela z vrednostmi PSNR in velikostmi datotek za različne kakovosti slike splash. Prva vrstica tabele predstavlja original v formatu TIFF, ostale vrstice predstavljajo kompresirane slike v JPEG s kvaliteto, ki je zapisana v prvem stolpcu.

Iz Tabel 4.1–4.3 opazimo, da pri enakih vrednostih kvalitete, pri različnih slikah z isto originalno velikostjo, dobimo različno velike datoteke. To je ena izmed pomembnih lastnosti JPEG-a, da nima vnaprej določene velikosti slike. Opazimo, da za sliko mandril 95 porabimo 186 kB, medtem ko za sliko splash 95 zgolj 88 kB, kar je več kot dvakrat manj. Razlog za to je, da za izris mandrila potrebujemo višje frekvence, saj ima veliko hitrih sprememb intenzitete. Na drugi strani je splash v večji meri enobarvna slika.

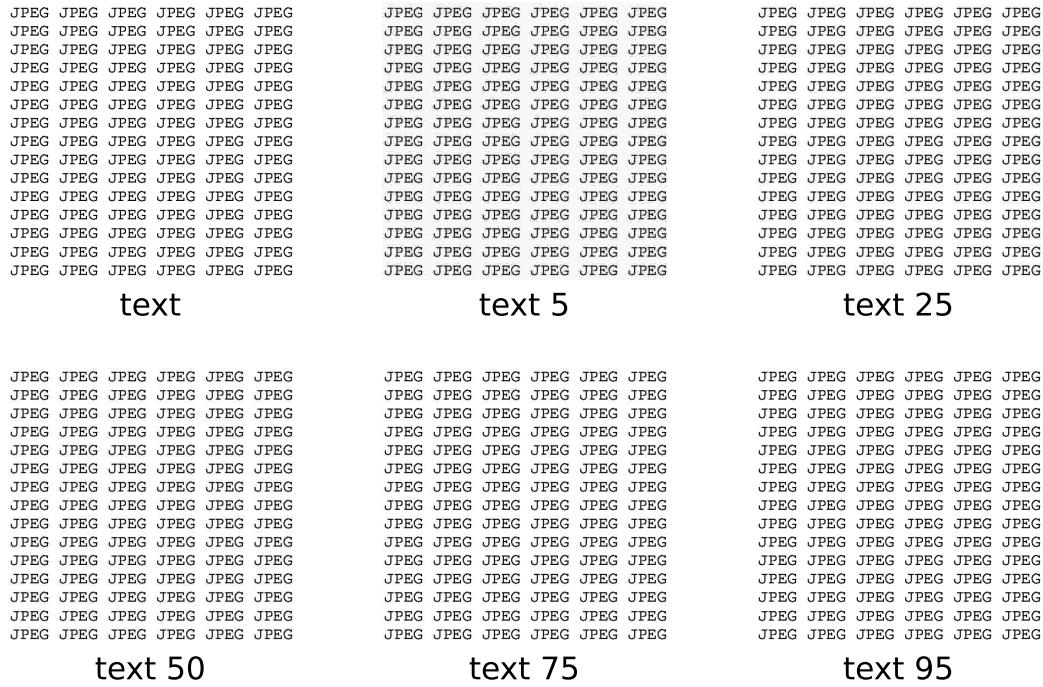
Če pogledamo sliko peppers, opazimo, da je porabila manj prostora kot mandril, vendar je tudi PSNR slik na posameznih kvalitetah nižji. Tukaj vidimo, da JPEG ne izbere vedno optimalne velikosti datoteke.

Opazimo, da je nivo kompresije JPEG-a že pri izbiri kvalitete 95 zelo dober. Na opisanih primerih nam to prinese 5- do 10-kratno kompresijo. Na tem nivoju ni skoraj nobene izgube kvalitete. Pri izbiri kvalitete 75 se to dvigne na 10- do 25-kratno kompresijo.

## 4.2 Problematične slike za JPEG

Pokazali bomo še primer slike, ko kompresija JPEG-a ne deluje dobro. To se zgodi, ko slike ne predstavljajo naravnih objektov. Najpreprostejši primer je prikaz besedila s sliko. Primer tega je prikazan na Sliki 4.4. Originalna slika je bila shranjena v formatu PNG pri resoluciji 512x512. V tem primeru je ta format veliko primernejši, ker ozadje označi kot prosojno, kar zasede veliko manj prostora. V Tabeli 4.4 so prikazane velikosti datotek za shranjevanje te slike.

V Tabeli 4.4 opazimo, da že pri največji kompresiji JPEG-a ta slika zasede več prostora, kot da jo shranimo v formatu PNG. Dovolj je, da pogledamo samo velikosti datotek v JPEG-u, da opazimo, kako neuporaben je za ta namen, vendar to ni njegov edini problem. Na Sliki 4.5 je prikazana približana verzija Slike 4.4, kjer je prikazana le ena ponovitev besede JPEG. Na tej sliki vidimo, da sta original in JPEG s kvaliteto 95 skoraj enaka. Problemi nastanejo pri kvaliteti 75 in nižje. V teh primerih opazimo, da so se pojavili novi



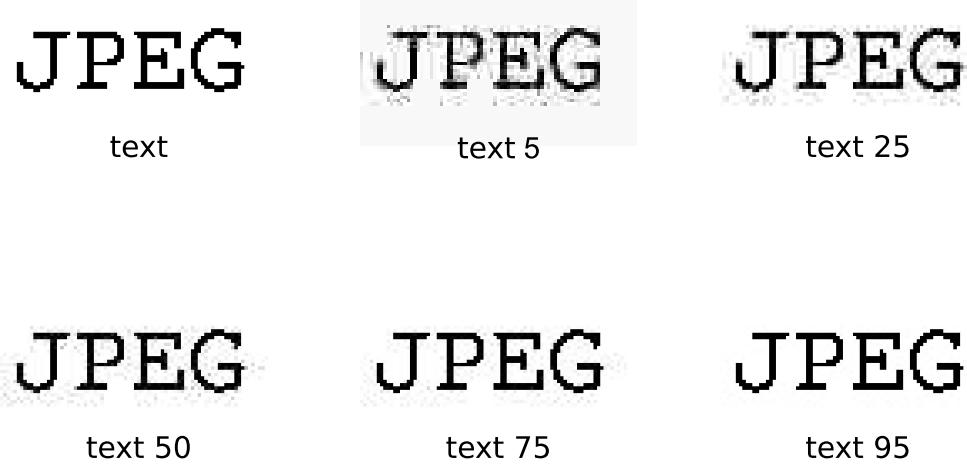
Slika 4.4: Besedilo prikazano v različnih kvalitetah. Levo zgoraj je original v formatu PNG. Ostali so v formatu JPEG, kjer številka zraven imena predstavlja nivo kvalitete nastavljen v knjižnici PIL za generiranje JPEG-a. Vir: generirano s [14].

temni piksli na mestih, kjer je original bel. Nižja kot je kvaliteta, opaznejše je to.

Iz te slike vidimo, da če želimo besedilo shraniti s sliko formata JPEG brez vidnih artefaktov, potrebujemo skoraj 10-krat več prostora, kot če bi to sliko shranili v formatu PNG. Za take primere slik torej JPEG ni ustrezen format.

Ime slike in kvaliteta	velikost datoteke
text original	18 kB
text 5	22 kB
text 25	44 kB
text 50	63 kB
text 75	85 kB
text 95	151 kB

Tabela 4.4: Tabela z velikostmi datotek za različne kakovosti slike text. Prva vrstica tabele predstavlja original v formatu PNG, ostale vrstice predstavljajo kompresirane slike v JPEG s kvaliteto, ki je zapisana v prvem stolpcu.



Slika 4.5: Približana beseda iz Slike 4.4.

# Poglavlje 5

## Sklepne ugotovitve

Pokazali smo, kako poteka kodiranje JPEG-a s pomočjo DCT. V tem procesu smo spoznali še veliko metod, ki so potrebne, da se to združi v celoto za kompresijo. Opisali smo sicer zgolj najpogosteje uporabljano verzijo JPEG-a. V samem standardu je še veliko specifik in pametnih idej, da ta format deluje tako dobro, kot deluje. Sam format je bil zelo dobro zasnovan, saj se uporablja že 30 let. Kljub temu da so najmočnejši računalniki devetdesetih let enakovredni današnjim pametnim telefonom, je to še vedno najpopularnejši format za kompresijo slik. Obstajajo sicer že boljši formati, a zaradi svoje enostavnosti in efektivnosti ga ni nadomestil še nobeden. Vsak dan nastane več milijard slik v formatu JPEG in ta številka se le veča.

V poglavju 4 smo na nekaj primerih slik preučili učinkovitost kompresije z mero PSNR. Zanimivo bi bilo dodati še mero SSIM (*angl. structural similarity index measure*), ki bolje upošteva zaznavanje človeškega očesa kot PSNR. PSNR bolj matematično ocenjuje razlike med dvema slikama, kar v praksi ni nujno tako primerno.

Pokazali smo primer slike, kjer dobimo manj zadovoljiv rezultat. Podobne težave nastopajo v manjši meri na vseh slikah. Lahko bi dalje preučevali, na kakšen način lahko odpravimo te napake.



# Literatura

- [1] K. Ashok Babu in V. Satish Kumar. “Implementation of data compression using Huffman coding”. V: *2010 International Conference on Methods and Models in Computer Science (ICM2CS-2010)*. 2010, str. 70–75. DOI: [10.1109/ICM2CS.2010.5706721](https://doi.org/10.1109/ICM2CS.2010.5706721).
- [2] R.N. Bracewell. *The Fourier Transform and Its Applications*. Circuits and systems. McGraw Hill, 2000. ISBN: 9780073039381. URL: <https://books.google.si/books?id=ZNQQAQAAIAAJ>.
- [3] V. Britanak, P.C. Yip in K.R. Rao. *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*. Elsevier Science, 2010. ISBN: 9780080464640.
- [4] E. Hernandez in G. Weiss. *A First Course on Wavelets*. Studies in Advanced Mathematics. CRC Press, 1996. ISBN: 9781420049985. URL: <https://books.google.si/books?id=Dq95ngIhy0oC>.
- [5] Alain Horé in Djemel Ziou. “Image Quality Metrics: PSNR vs. SSIM”. V: *2010 20th International Conference on Pattern Recognition*. 2010, str. 2366–2369. DOI: [10.1109/ICPR.2010.579](https://doi.org/10.1109/ICPR.2010.579).
- [6] The MathWorks Inc. *Discrete Cosine Transform*. Natick, Massachusetts, United States, 2022. URL: <https://www.mathworks.com/help/images/discrete-cosine-transform.html> (pridobljeno 30. 8. 2023).
- [7] ITU. *ISO/IEC 10918-1 : 1993(E) CCIT Recommendation T.81*. 1993. URL: <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>.

- [8] Goce Jakimoski in K. P. Subbalakshmi. “Cryptanalysis of Some Multimedia Encryption Schemes”. V: *IEEE Transactions on Multimedia* 10.3 (2008), str. 330–338. DOI: 10.1109/TMM.2008.917355.
- [9] Syed Ali Khayam. “The Discrete Cosine Transform (DCT): Theory and Application”. V: *Course Notes, Department of Electrical & Computer Engineering* (jan. 2003).
- [10] Jernej Kozak. *Podatkovne strukture in algoritmi*. 2. natis. Zv. 27. 300 izv. Društvo matematikov, fizikov in astronomov Slovenije, 1997, str. 198–204. ISBN: 961-212-072-2. 198–204.
- [11] J. Makhoul. “A fast cosine transform in one and two dimensions”. V: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.1 (1980), str. 27–34. DOI: 10.1109/TASSP.1980.1163351. URL: <https://ieeexplore.ieee.org/document/1163351>.
- [12] Daniel Potts in Manfred Tasche. “Numerical stability of nonequispaced fast Fourier transforms”. V: *Journal of Computational and Applied Mathematics* 222 (dec. 2008), str. 655–674. DOI: 10.1016/j.cam.2007.12.025.
- [13] *Sipi Image Database - MISC*. 1977. URL: <https://sipi.usc.edu/database/database.php?volume=misc> (pridobljeno 30.8.2023).
- [14] Gregor Šraj. *Izvorna koda*. [https://github.com/GregorSraj/Diplomska\\_naloga](https://github.com/GregorSraj/Diplomska_naloga). 2023. (Pridobljeno 13.9.2023).
- [15] Gilbert Strang. “The Discrete Cosine Transform”. V: *SIAM Rev.* 41 (1999), str. 135–147. URL: <https://api.semanticscholar.org/CorpusID:6693738>.
- [16] Miroslav Uhrina, Juraj Bienik in Tomas Mizdos. “Chroma Subsampling Influence on the Perceived Video Quality for Compressed Sequences in High Resolutions”. V: *Advances in Electrical and Electronic Engineering* 15 (nov. 2017). DOI: 10.15598/aeee.v15i4.2414.

- [17] Gregory K Wallace. “The JPEG still picture compression standard”. V: *IEEE transactions on consumer electronics* 38.1 (1992), str. xviii–xxxiv. URL: <https://www.ijg.org/files/Wallace.JPEG.pdf>.