Automatska optimizacija slika za web

Kopić, Vlado

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:792471

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

AUTOMATSKA OPTIMIZACIJA SLIKA ZA WEB

Diplomski rad

Vlado Kopić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 21.09.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Vlado Kopić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 818 R, 12.10.2015.
OIB studenta:	50072530918
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Časlav Livada
Član Povjerenstva:	Krešimir Romić
Naslov diplomskog rada:	Automatska optimizacija slika za web
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Opisati JPEG tehnologiju kompresije slike. Opisati dva najpopularnija JPEG kompresora otvorenog koda: libjpeg-turbo i mozjpeg. Teorijski obraditi popularnu SSIM metriku sličnosti slike i njenu primjenu za automatsku optimizaciju slike. U praktičnom dijelu implementirati sustav koji će omogućiti automatsku optimizaciju slika unutar nekog repozitorija. Tehnologija: Ruby on Rails Sumentor: Hrvoje Leventić. Za sve informacije u vezi rada javiti se na hrvoje.leventic@etfos.hr
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.10.2017.

Ime i prezime studenta:	Vlado Kopić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 818 R, 12.10.2015.
Ephorus podudaranje [%]:	1%

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatska optimizacija slika za web**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
2. KOMPRESIJA SLIKE 2
2.1.1. YC_bC_r konverzija 5
2.1.2. DCT i njeni koeficijenti 6
2.1.3. Kvantizacija 7
2.1.4. Entropijsko kodiranje i Huffmanovo kodiranje (algoritam) 8
2.2.1. Usporedba libjpeg i mozjpeg kompresora 11
3. IMPLEMENTACIJA SUSTAVA AUTOMATSKE OPTIMIZACIJE SLIKA ZA WEB 15
3.1. Korištene tehnologije 15
3.1.1. Ruby on Rails 15
3.1.2. Docker 16
3.2.1. Baza podataka i njeno popunjavanje slikama 17
3.2.2. Algoritam optimizacije 19
3.2.3. Aplikacija za optimizaciju 22
4. ZAKLJUČAK 26
LITERATURA 27
SAŽETAK 29
Automated Optimization of Images for the Web 29
ŽIVOTOPIS 31

1. UVOD

Optimizacija slika za uporabu na web stranicama postaje sve potrebniji postupak u doba modernog interneta. Kada se postojeći internetski promet podijeli po količini podataka koja se preuzima za pojedine elemente stranice, slike će u većini slučajeva biti na prvom mjestu po potrošnji.

Razvojem i napretkom modernih internetskih tehnologija te sve većih i pristupačnijih brzina pristupa internetu, često optimizacija slika pri izradi stranica pada u zaborav. Ovo je vrlo loša praksa jer unatoč činjenici da dobar dio korisnika ima dovoljno velike brzine i pakete potrošnje kako bi podnijeli teret koji loše optimizirane slike predstavljaju za internetsku vezu, postoje i korisnici koji se spajaju putem loših mreža, koriste male brzine ili nemaju dovoljno velik paket potrošnje. A čak i ukoliko se zanemare ovakvi korisnici i pretpostavi idealno stanje pristupa internetu, opet je dobra praksa optimizirati svaki dio web stranice radi ugodnijeg korisničkog iskustva, pa tako i slike.

Glavni zadatak ovog diplomskog rada je pokušati teorijski i na temelju konkretne realizacije predstaviti rješenje za problem optimizacije slika, te kako to rješenje primijeniti za potrebe web stranica. U pisanom dijelu rada biti će predstavljen pregled tehnologija za kompresiju digitalne slike, s posebnom pažnjom posvećenom vrstama JPEG kompresije i JPEG kompresorima. Usporedit će se dvije vrste JPEG kompresora te izložiti najpopularnije metrike za procjenu kvalitete kompresije. Na kraju će se opisati sama implementacija rješenja problematike ovog diplomskog rada.

2. KOMPRESIJA SLIKE

U podatkovnim znanostima, kompresija podataka označava postupak enkodiranja podataka predstavljajući ih na način koji smanjuje sveukupnu veličinu tih podataka. Ovakvo smanjivanje je moguće kad izvorni skup podataka sadrži određenu vrstu zalihosti (eng. *redundancy*).

Obzirom da su i digitalne slike vrsta podatkovnih skupova, moguće je ovakve metode smanjivanja veličine primjeniti i na njih. Kompresija digitalne slike (u nastavku: kompresija slike) polje je kompresije podataka koje proučava metode smanjivanja ukupnog broja bitova potrebnih za predstavljanje neke slike. Ovakav je ishod moguće postići eliminacijom raznih vrsta zalihosti koje postoje u vrijednostima piksela slike. Generalno govoreći, prema [1], sljedeće tri osnovne vrste zalihosti postoje u digitalnim slikama:

- Psiho-vizualna zalihost: Ova zalihost odgovara različitim osjetnim percepcijama svih slikovnih signala od strane ljudskih očiju. Eliminacija nekih relativno manje bitnih djelića informacije iz slike koja se obrađuje može biti prihvatljiva.
- Međupikselna zalihost: Ova zalihost odgovara statističkim međuovisnostima piksela, posebno kad se radi o susjednim pikselima.
- Zalihost u kodiranju: Nekompresirana slika obično je kodirana s fiksnom duljinom svakog piksela (npr. predstavljanje slike s 256 razina sive boje koristeći niz 8-bitnih integera). Korištenje shema koda s varijabilnim duljinama kao npr. Huffmanovo kodiranje ili aritmetičko kodiranje može dovesti do kompresije.

Postoji više metoda za nošenje s navedenim vrstama zalihosti. Zbog toga kompresor slike često koristi algoritme s više koraka (eng. *multi-step algorithm*) za smanjivanje tih zalihosti.

Sama kompresija slika može biti kompresija bez gubitaka (eng. *lossless*) ili s gubitcima (eng. *lossy*). Pri tome se misli na gubitak kvalitete u odnosu na izvornu sliku, dakle na izgubljene dijelove izvornog skupa podataka nad kojim se vršila kompresija.

Kompresija bez gubitaka obično se koristi kod umjetnih slika koje sadrže linije s oštrim rubovima kao što su tehnički crteži, tekstualna grafika, stripovi, karte ili logotipi. Razlog tomu je što kompresija s gubitcima proizvodi tzv. kompresijske artefakte zbog kojih oštre linije postaju mutne, posebno ako se koristi jači stupanj kompresije.

Kompresija s gubitcima znatno bolje primjene ima za smanjivanje prirodnih slika kao što su slike krajolika, gdje je prihvatljiv mali gubitak na oštrini kako bi se postigla manja veličina slike.

Postoji mnogo metoda za kompresiju slika, od kojih će u nastavku biti ukratko predstavljene najraširenije i najpoznatije. Primjena takvih metoda na neku sliku naziva se enkodiranje, a samim time program, uređaj ili format koji vrši postupak enkodiranja naziva se enkoder. Iz podataka koji se dobiju enkodiranjem zatim se dobiva konačna, komprimirana slika. Taj proces naziva se dekodiranje.

Neki od najpopularnijih i najpoznatijih slikovnih formata u digitalnom obliku te formati o kojima će biti više govora u ovom radu su JPEG, GIF, PNG i BMP. Ovo su ujedno i odgovarajući enkoderi za svaki taj format.

JPEG (eng. Joint Photographic Experts Group) je uz PNG najraširenij, najpoznatiji i najpodržavaniji format i enkoder digitalnih slika prema [2], no glavna mu je prednost u odnosu na PNG što je puno primjereniji za korištenje na web stranicama jer je, za razliku od PNG formata, enkodiran kompresijom s gubitcima što znači da žrtvuje nešto veći postotak kvalitete radi puno bolje iskoristivosti prostora i mrežnih resursa. Ova razlika u kvaliteti na velikoj većini web stranica neće biti vidljiva ili upadna korisniku, no omogućava bolje performanse stranice te brža vremena učitavanja. Osim toga, JPEG format pogodan je za web stranice jer korisniku omogućava definiranje skalabilnog omjera kvalitete i kompresije; prema [3], JPEG format tipično može postići stupanj kompresije 10:1 u odnosu na original, uz malen ili gotovo neprimjetan gubitak na kvaliteti. Slike enkodirane JPEG formatom obično imaju ekstenziju .jpg ili .jpeg.

PNG (eng. Portable Network Graphics) je već spomenuti izuzetno popularan i podržavan format i enkoder digitalnih slika i najraširenij format koji koristi kompresiju bez gubitaka. Ova činjenica ga čini idealnim za neke vrste stranica koje si ne smiju dozvoliti dodatno žrtvovanje kvalitete koje uz sebe povlači JPEG format, i spremne su za to platiti cijenu u vidu nešto sporijeg učitavanja i lošijeg iskorištavanja mrežnih resursa. U slučajevima gdje ovakva vrsta kompresije nije potrebna, preporučljivo je koristiti JPEG enkodiranje. PNG format izuzetno je pogodan za pohranu slika koje će se uređivati, obzirom da one ovim enkodiranjem neće izgubiti na kvaliteti. Slike enkodirane PNG formatom obično imaju ekstenziju .png.

GIF (eng. Graphics Interchange Format) je vrlo poznat i još uvijek raširen format i enkoder digitalnih slika usprkos činjenici da je predstavljen prije čak 30 godina. Često se koristi na web stranicama, u današnje vrijeme najčešće u svrhu predstavljanja pokretnih slika ili animacija. GIF format je još jedan primjer kompresije bez gubitaka, te kao takav direktno "konkurira" PNG formatu te se prirodno vuku paralele i usporedbe između ta dva formata. Prema [4], GIF podržava maksimalno 256 boja u bilo kojoj slici enkodiranoj ovim formatom, što nameće veliko ograničenje bez obzira što se pri kompresiji ne ostvaruju gubitci. Zbog ove činjenice obično se ne preporuča spremati digitalne slike u ovom formatu, već za njih koristiti JPEG (ukoliko je prihvatljiv određeni gubitak kvalitete) ili PNG format. Slike enkodirane GIF formatom obično imaju ekstenziju .gif.

BMP (eng. Bitmap) je često korišten format za spremanje digitalnih slika. Prema [5], uveden je na Windows platformi, no u današnje vrijeme ga prepoznaju i mnogi Mac i Linux programi i skripte. Glavna odlika ovog formata je da on zapravo ne primjenjuje kompresiju na piksele slike. Ova činjenica kao rezultat daje vrlo jasne i visokokvalitetne slike, no to znači da slike u ovom formatu zauzimaju puno više mjesta nego kompresirane slike. Prema [5], upravo zbog ovog razloga BMP format nije pretjerano raširen na web stranicama već se prvenstveno koristi za slike koje se ispisuju. Slike enkodirane BMP formatom obično imaju ekstenziju .bmp.

2.1. JPEG kompresija

Kao što je ranije spomenuto, JPEG format koristi kompresiju s gubitcima. Ovo u praksi znači da se pikseli slike enkodirane JPEG kompresijom prikazuju ne točno onakvima kakvi su bili, a u svrhu postizanja manje konačne veličine dobivene komprimirane slike – pod uvjetom da konačni rezultat izgleda vrlo slično originalu. Vidljive razlike pojavljuju se tek povećavanjem stupnja kompresije, odnosno smanjivanjem željene kvalitete nauštrb veličine. Prema [6], JPEG algoritam iskorištava činjenicu da ljudi ne mogu vidjeti boje na visokim frekvencijama, a upravo te frekvencije su podatkovne točke koje se eliminiraju postupkom kompresije.

Algoritam JPEG enkodiranja objašnjen je u sljedećim koracima u obliku generalnog pregleda postupka, i to prema [6, 7, 8].

1. Konverzija slike iz RGB formata u tzv. YC_bC_r format boja.
2. Smanjivanje razlučivosti C_b i C_r komponenti slike (eng. *downsampling*) u unaprijed definiranim omjerima.

3. Dijeljenje konvertirane izvorne slike u blokove od 8x8 piksela. Ukoliko je ovakva podjela nemoguća, mogu se dodati prazni pikseli oko rubova slike kako bi se ona omogućila.
4. Izvesti DCT (diskretna kosinusna transformacija, eng. *Discrete Cosine Transform*) na svakom 8x8 bloku.
5. Izvršiti proces kvantizacije (eng. *quantization*) nad amplitudama dobivenog frekvencijskog spektra.
6. Primjeniti Huffmanovo kodiranje na dobivenim komponentama slike.

Postupak dekodiranja je obrnut i njime se poništavaju sve promjene u odnosu na izvornu sliku, osim kvantizacije. Taj postupak je ireverzibilan.

2.1.1. YC_bC_r konverzija

Velika većina digitalnih slika informacije o bojama nosi u RGB formatu, koji svaki piksel iscrtava ovisno o količini crvene, zelene i plave boje sadržane u njemu. Radi lakšeg postupka kompresije i boljeg baratanja bojama, prije početka postupka same kompresije tzv. prostor boja slike (eng. *color space*) pretvara se iz RGB formata u YC_bC_r format.

Prema [10], Y označava svjetlinu slike (eng. *luminance,* ili *luma* ako se radi o Y' oznaci), dok C_b i C_r označavaju komponente boja u ovisnosti o plavoj (C_b) i crvenoj (C_r). Često se u notaciji viđaju i omjeri uz oznaku YC_bC_r, npr. 4:4:4. Prema [9], ovo samo znači da se komponente boja spremaju ili iscrtavaju na ekran istom stopom kao i svjetlina. Ukoliko se sprema blok od 8x8 piksela i koristi omjer 4:4:4, niti jedan stupac tog bloka neće biti preskočen pri spremanju kad se razmatraju komponente boja. Ukoliko je pak omjer stopa 4:2:2, kod spremanja komponenti boja preskočit će se svaki drugi stupac bloka radi uštede na potrošenoj memoriji. Ovakav omjer se smatra visokokvalitetnim, dok se 4:4:4 obično smatra pretjeranim za većinu primjena.

Sama konverzija iz RGB prostora boja u YC_bC_r prostor boja vrši se korištenjem jednadžbi koje definiraju koliko koje boje početnog piksela prelazi u komponente boja i svjetlinu YC_bC_r prostora boja. Prema [11], na primjeru 8-bitne JPEG slike u RGB prostoru boja, konverzija u 256-razinski (raspon gdje 0 označava bijelu, 255 crnu boju) YC_bC_r prostor boja odvija se sljedećim jednadžbama:

$$Y = 0.299R + 0.587G + 0.114B \quad (2\text{-}1)$$

$$C_b = -0.1687R - 0.3313G + 0.5B + 128 \quad (2\text{-}2)$$

$$C_r = 0.5R - 0.4187G - 0.0813B + 128 \quad (2\text{-}3)$$

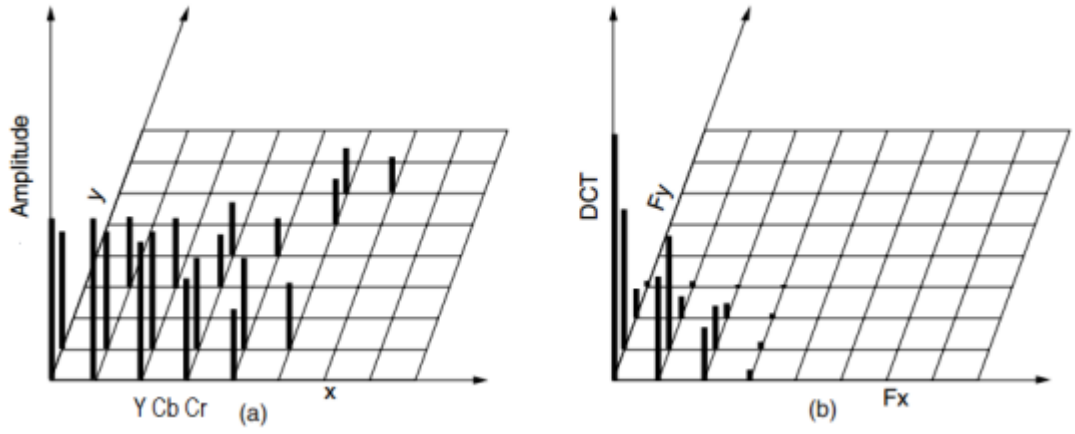
R označava količinu crvene boje u promatranom pikselu, G količinu zelene boje, a B količinu plave boje.

2.1.2. DCT i njeni koeficijenti

Nakon smanjivanja razlučivosti komponenata boje C_b i C_r te podjele dobivenog podatkovnog skupa u 8x8 blokove, na svaki od tih blokova primjenjuje se DCT. Rezultat svake od ovih primjena je 8x8 matrica DCT koeficijenata, gdje je DCT element (0, 0) prosječna vrijednost pojedinog bloka. Ovi koeficijenti računaju se na temelju DCT jednadžbi (2-4) i (2-5), a rezultat prikazan u grafičkom obliku vidljiv je u grafovima na slici 2.1. Primjena DCT zapravo se odvija u dva koraka, zbog čega su potrebne dvije jednadžbe. Prvi korak je primjenjivanje DCT prema naprijed (eng. *Forward DCT* ili FDCT), dok je drugi korak primjenjivanje inverzne DCT (eng. *Inverted DCT* ili IDCT).

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16}\right] \quad (2\text{-}4)$$

$$f(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v) * cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16}\right] \quad (2\text{-}5)$$



Slika 2. 1. Prema [7], jedan blok Y matrice (a) i odgovarajući DCT koeficijenti (b)

U jednadžbama (2-4) i (2-5), *C(u)* i *C(v)* iznose $\frac{1}{\sqrt{2}}$ za *u,v* = 0, odnosno *C(u)* i *C(v)* = 1 u bilo kojem drugom slučaju. *f(x, y)* označava 8-bitnu vrijednost slike na koordinatama *x, y*. *F(u, v)* označava novi unos u frekvencijskoj matrici, odnosno dobivenoj matrici.

Prema [12], elementi koji ulaze u FDCT računaju se tako da se iz 8x8 ulaznog bloka uzme prosječna vrijednost svih elemenata (za 256-razinski YC_bC_r prostor boja to je 128) i oduzme od postojećih elemenata ulaznog bloka, te se na njih primjeni FDCT. Ovaj korak se odvija pri enkodiranju. Pri dekodiranju se uzimaju koeficijenti dobiveni FDCT-om i na njih se primjenjuje IDCT, čime se dobiva rekonstruirana slika.

DCT koeficijenti su u svojoj biti amplitude čije su vrijednosti relativne količine prostornih frekvencija sadržanih u ulaznim podacima, te su ovo koeficijenti dobiveni primjenom FDCT-a na ulazne blokove. Promatrajući skup rezultata u dvije dimenzije, x i y, koeficijent koji ima frekvenciju "0" u obe dimenzije naziva se DC koeficijent, dok se preostala 63 nazivaju AC koeficijentima.

DC koeficijent je prosjek svih elemenata u bloku. On definira ton boje (eng. *hue*) za cijeli blok. AC koeficijenti predstavljaju kosinusne funkcije kojima se frekvencija povećava što su dalje od DC koeficijenta (udesno za vertikalni smjer, prema dolje za horizontalni smjer). Vrijednosti viših frekvencija često su znatno manje od vrijednosti nižih frekvencija jer je njihov doprinos slici puno zanemariviji. U tipičnoj situaciji cijela donja desna polovina matrice se poništi u sljedećem koraku, kvantizaciji.

2.1.3. Kvantizacija

U ovom koraku JPEG kompresije eliminiraju se manje bitni DCT koeficijenti, dakle, kao što je prethodno definirano, oni s visokim frekvencijama. Ovaj postupak odvija se dijeljenjem svakog elementa nekog bloka DCT koeficijenata s konstantom za taj koeficijent, a koja se isčitava iz tzv. kvantizacijske matrice (eng. *quantization matrix*). Tipična kvantizacijska matrica s kriterijem kvalitete postavljenim na 50% kao što je specificirano u izvornom JPEG standardu prikazana je na slici 2.2. no prema [8] postoje i druge kvantizacijske matrice te se one mogu prilagoditi ovisno o potrebama korisnika i pružaju se kao jedan od ulaznih elemenata (eng. *input*-a) enkoderu.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

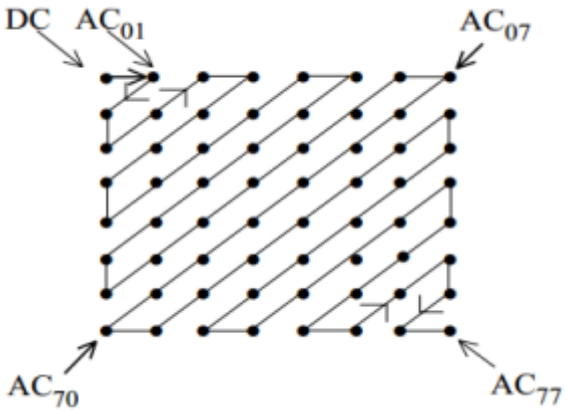
Slika 2. 2. Standardna JPEG kvantizacijska matrica za kvalitetu slike 50%

$$B_{j,k} = round\left(\frac{G_{j,k}}{Q_{j,k}}\right) \; za \; j,k = 0,1,\ldots,7 \qquad (2\text{-}6)$$

Formula za računanje elemenata novonastale matrice nakon postupka kvantizacije dana je relacijom (2-6), gdje *B* predstavlja kvantizirane DCT koeficijente, *G* predstavlja nekvantizirane DCT koeficijente a *Q* predstavlja zadanu kvantizacijsku matricu. Dakle, nekvantizirani DCT koeficijent podijeli se odgovarajućim elementom kvantizacijske matrice i uzme se najbliži integer. Taj broj postaje odgovarajući element kvantizirane matrice u tom položaju.

2.1.4. Entropijsko kodiranje i Huffmanovo kodiranje (algoritam)

Nakon kvantizacije treba poredati komponente slike u "zig-zag" obliku koristeći algoritam koji iz tog zig-zag oblika u kojem se nalaze kvantizirani koeficijenti oblikuje niz simbola, a potom se ti simboli pretvaraju u struju podataka (eng. *data stream*) u kojoj simboli više nemaju vanjski prepoznatljive granice. Ovaj korak naziva se entropijsko kodiranje, a točan oblik i definicija tih simbola ovisi o DCT načinu rada i o samoj metodi entropijskog kodiranja koja se koristi. Postoje dvije glavne metode koje se koriste: Huffmanovo kodiranje i aritmetičko kodiranje, no zbog problema s patentima i sporijeg procesa enkodiranja i dekodiranja aritmetičko kodiranje rijetko se koristi.



Slika 2. 3. Prema [7], vizualizirani postupak oblikovanja niza simbola iz kvantizirane DCT matrice

Postupak oblikovanja niza simbola iz kvantiziranog bloka prikazan na slici 2.3. naziva se sekvencijalno kodiranje. Postoji i drugi pristup, tzv. progresivno kodiranje. Osnovna razlika je u tome što sekvencijalno kodiranje oblikuje niz simbola iz jednog po jednog bloka, grupirajući njihove elemente ovisno o frekvencijama. Progresivno kodiranje pak grupira zajedno i enkodira skupove koeficijenata u sličnim položajima gledajući sve blokove odjednom (ovaj postupak naziva se skeniranje - eng. *scan*). Kad završi trenutno skeniranje, algoritam prelazi na idući skup slično pozicioniranih koeficijenata i tako dalje.

Huffmanovo kodiranje odvija se na kraju ovog koraka i zahtijeva jedan ili više skupova Huffmanovih kodnih tablica. One mogu biti predefinirane od strane aplikacije ili korisnika, ili pak izračunate za danu sliku prije samog koraka kompresije. I ovdje se kristalizira jedna od glavnih razlika između sekvencijalnog i progresivnog kodiranja – algoritam progresivnog kodiranja dopušta korištenje više Huffmanovih tablica koje su prilagođene za svaki pojedini postupak skeniranja. Upravo zbog ovog razloga je progresivno kodiranje obično nešto učinkovitije i postiže veći stupanj kompresije bez gubitka na kvaliteti, no razlike nisu prevelike.

Postupak Huffmanovog kodiranja za sekvencijalno generiran uzorak simbola je sljedeći:

1. Algoritam promatra svaki AC koeficijent različit od nule i određuje koliko nula je prethodilo prošlom AC koeficijentu. Na temelju ovog podatka generiraju se simboli za nastavak rada Huffmanovog algoritma koji kažu koliko nula prethodi svakom AC koeficijentu, te koliko bitova će biti potrebno za predstavljanje tog AC koeficijenta nakon kodiranja.
2. Poseban segment koda dodjeljuje se dijelu matrice kad su svi preostali koeficijenti do kraja 0. Ovo je tzv. *End-of-Block* ili EOB kodna riječ. Drugi poseban slučaj pojavljuje se kad između dva AC koeficijenta postoji više od 15 nula. U takvom slučaju

odgovarajući simbol za taj AC koeficijent biti će enkodiran posebnim simbolom, npr. (15, 0) ukoliko se radi o 16 nula.

3. Nakon dodjeljivanja svih simbola AC koeficijentima i određivanja posebnih slučajeva iz koraka 2, slijedi računanje na temelju frekvencija gdje algoritam zaključuje koliko često se koji simbol treba pojavljivati u slici.
4. Na temelju ovih podataka generira se kod za svaki simbol čija duljina varira ovisno o statističkoj vjerojatnosti njegovog pojavljivanja. Simbol koji se često koristi biti će nakon enkodiranja pridružen kodu čija je duljina samo nekoliko bita, dok će rjeđi simboli biti predstavljeni kodom za čije enkodiranje treba puno više bita.

2.2. JPEG kompresori

Nakon opisivanja postupka enkodiranja digitalne slike JPEG kompresijom, o ovom poglavlju biti će govora o konkretnim primjerima dostupnih JPEG kompresora, odnosno biblioteka i enkodera koje imaju mogućnost iz neke ulazne slike stvoriti JPEG verziju. U idućem potpoglavlju bit će uspoređeni libjpeg i mozjpeg kompresori zbog njihove raširenosti i relevantnosti za ovaj rad.

libjpeg je besplatna biblioteka s funkcijama za baratanje JPEG formatom. Napisana je u C-u a dostupan je i njen izvorni kod. Izvorna verzija koja se pojavila 1991. godine održavana je i nadograđivana sve do danas, a postoji i nekoliko odvojenih projekata temeljenih na libjpeg izvornom kodu (eng. *fork*). Izvorna verzija razvijena je i održavana od strane Independent JPEG Group-a (IJG).

Libjpeg-turbo je tzv. fork libjpeg-a koji ubrzava temeljno JPEG enkodiranje i dekodiranje. Mnogi korisnici su se s izvornog libjpeg-a prebacili na libjpeg-turbo upravo zbog ovih poboljšanih performansi. To uključuje projekte kao što su Fedora, Debian, openSUSE i slične Linux distribucije, Mozilla i Chrome. libjpeg-turbo hvali se brzinama enkodiranja 2-6 puta većim od izvornog libjpeg-a uz isti stupanj kvalitete i veličine komprimirane slike.

Mozjpeg je fork libjpeg-a kojeg su razvili programeri iz Mozilla Reasearch-a predvođeni Joshom Aasom. Ovaj kompresor temeljen je na jpeg-turbo biblioteci prema [13]. Potpuno je kompatibilan s libjpeg-om te se može koristiti kao zamjena za njega. Podržava progresivno kodiranje radi dodatnog smanjivanja veličine komprimiranih slika.

Guetzli je novi kompresor osmišljen i stvoren od strane Google-a. Hvali se slikama koje su u prosjeku 20-30% manje od istovjetnih slika generiranih libjpeg bibliotekama. Guetzli koristi samo sekvencijalno kodiranje jer prema [14] ono nudi veće brzine dekompresije. Ovaj kompresor zamišljen je s radom na velikim slikama u vidu, te kao ulaznu sliku preferira nekomprimirane izvorne slike. Što je manji stupanj kompresije ulazne slike, konačni rezultat komprimiranja s Guetzli kompresorom biti će bolji. Podržava i odabir željenog stupnja kvalitete na isti način kao i libjpeg.

2.2.1. Usporedba libjpeg i mozjpeg kompresora

Kao što je ranije spomenuto, mozjpeg nastao je kao fork libjpeg-turbo kompresora koji je pak temeljen na izvornom libjpeg-u. Ova tri kompresora danas su vjerojatno najpopularniji i najrašireniji enkoderi svoje vrste.

Mozjpeg je samoproglašeni „kompresor koji najbolje služi kao enkoder za web stranice“ prema [13]. Mozilla Research tim svjesno je raznim kompromisima odlučio poboljšati pogodnost mozjpeg kompresora za web slučajeve, nekad nauštrb ostalih potencijalnih primjena. Štoviše, u samom opisu mozjpeg-a na njihovoj GitHub stranici preporuča se korištenje libjpeg-a u svim slučajevima osim web enkodiranja.

Prema [13], ovaj kompresor kombinira progresivno enkodiranje s prilagođenom inačicom kvantizacijskog algoritma (tzv. Trellis kvantizacija koja smanjuje neke koeficijente a povećava druge kako bi se našla optimalna kvantizacija za svaki blok – učinkovitija kompresija no kompliciraniji postupak izvođenja) kako bi smanjio veličinu JPEG slika. libjpeg i libjpeg-turbo podržavaju progresivno enkodiranje kao opciju, no ne i mozjpeg-ovo prilagođeno kvantizacijsko rješenje.

Kao posljedica ovoga, prema [15] slike enkodirane mozjpeg kompresorom u odnosu na standardno JPEG enkodiranje (eng. *baseline*) imaju poboljšan omjer kompresije u intervalu od 15-27% s prosječnim poboljšanjem od 20%. No zauzvrat se vrijeme kompresije povećava između 34 i 59 puta. Ovo u praksi znači da mozjpeg dozvoljava puno kvalitetnije slike s istom veličinom kao da se koristi baseline enkodiranje, a prema [15], čak i kad se koristi libjpeg-turbo biblioteka stupanj kompresije je poboljšan u intervalu od 4-15% (prosječno 8.3%).

Libjpeg i njegova brža inačica libjpeg-turbo, s druge strane, imaju puno brža vremena izvođenja, pogotovo libjpeg-turbo. Prema [16], čak u odnosu na proprijetarni Intelov algoritam za kompresiju JPEG formatom, libjpeg-turbo ostvarivao je rezultate 2.1 do 6 puta brže, no u većini scenarija imao je lošije stope kompresije, pogotovo na čistim x86 sustavima. libjpeg-turbo strogo je bolji od libjpeg-a u svim situacijama.

Ukoliko se uzme u obzir činjenica da mozjpeg uvijek ima bolji stupanj kompresije od libjpeg-turbo kompresora, prirodno se nameće zaključak kako se libjpeg biblioteka usredotočuje na brzinu izvođenja i prihvatljiv stupanj kompresije, dok mozjpeg inzistira na optimiziranoj kompresiji i samim time kvaliteti nauštrb brzine izvođenja.

2.3. Metrike za evaluaciju kvalitete kompresije

Pri osmišljavanju i testiranju postupka automatske optimizacije slika izuzetno je bitno imati mjerilo koliko je slika degradirala u odnosu na izvornu. Postoji više definiranih metrika upravo za ovu svrhu, no za potrebe ovog rada dovoljne su sljedeće.

MSE (eng. *Mean Square Error*) kao metrika općenito uzima prosjeke kvadrata greške ili odstupanja između izvorne slike i slike koja se promatra i mjeri razliku među njima. Najčešće se u sklopu obrade slike koristi zajedno s PSNR metrikom, koja je opisana u nastavku. Prema [17], matematički se princip rada MSE metrike može izraziti relacijom (2-7):

$$MSE = \frac{1}{MN} \sum_{y=1}^{M} \sum_{x=1}^{N} [I(x,y) - I'(x,y)]^2 \tag{2-7}$$

gdje je *I(x, y)* izvorna slika, *I'(x, y)* je aproksimirana verzija (odnosno dekomprimirana slika) i *M, N* su dimenzije slika. Niža vrijednost MSE znači manju grešku.

PSNR (eng. *Peak Signal to Noise Ratio*) predstavlja omjer između maksimalne moguće snage signala i snage korumpirajućeg šuma koji utječe na vjernost predstavljanja tog signala. Kao što je ranije spomenuto, PSNR se najčešće definira upravo preko MSE, što je vidljivo iz relacije (2-8):

$$PSNR = 20 * log10(\frac{255}{\sqrt{MSE}}) \tag{2-8}$$

Obzirom na obrnuto proporcionalan odnos PSNR i MSE, moguće je zaključiti da će visoka vrijednost MSE dati nisku vrijednost PSNR i obrnuto te da je prema tomu visoka vrijednost PSNR

poželjna jer to znači da je omjer iskoristivog signala i štetnog šuma visok u korist signala. Konkretno na primjeru digitalne slike visok PSNR bi značio da je omjer signala (izvorne slike) i šuma (greške u rekonstrukciji i kompresiji) u korist izvorne slike, što je dobro i poželjno.

SSIM (eng. *Structural SIMilarity*) indeks metoda je mjerenja sličnosti između dviju slika. Ukoliko se jedna od slika smatra savršenim primjerkom (u slučaju kompresije to je izvorna slika) SSIM se može smatrati mjerilom kvalitete druge, komprimirane slike. Bitno je istaknuti kako je mogući rezultat SSIM metrike u intervalu [-1, 1], gdje 1 označava identičnost promatranih slika. Također, SSIM metrika zadovoljava uvjet simetričnosti, odnosno vrijedi

$$SSIM(X, Y) = SSIM(Y, X)$$

za bilo koje dvije promatrane slike X i Y.

2.3.1. Structural Similarity metrika

Proširujući prethodno postavljene definicije SSIM metrike, valja proučiti koje sve parametre SSIM procjenjuje pri donošenju zaključka o kvaliteti slika. Prema [18], postoje tri faktora koji ulaze u analizu SSIM metrikom: promjene u luminanci slike, promjene u kontrastu te strukturne promjene (preostale greške koje ne spadaju u prve dvije kategorije). Pretpostavimo da je *x* izvorni signal (slika), a *y* enkodirana slika. SSIM indeks se za njih matematički definira kao:

$$SSIM(x, y) = [l(x, y)]^{\alpha}[c(x, y)]^{\beta}[s(x, y)]^{\gamma} \quad (2\text{-}9)$$

gdje su $\alpha, \beta, \gamma > 0$ i upravljaju relativnom važnošću svakog od tri člana indeksa, a obično su sva tri grčka faktora jednaka 1. Članovi u uglatim zagradama su redom luminanca, kontrast i strukturne komponente indeksa i definiraju se kao:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x{}^2 + \mu_y{}^2 + C_1}, \quad (2\text{-}10)$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x{}^2 + \sigma_y{}^2 + C_2}, \quad (2\text{-}11)$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \quad (2\text{-}12)$$

gdje μ_x i μ_y označavaju prosjeke izvorne i enkodirane slike, σ_x i σ_y označavaju standardne devijacije svakog od signala, a $\sigma_x\sigma_y$ označava kovarijancu dviju slika. Zbog slučaja da su djeljitelji

približni nuli, uvode se konstante C_1, C_2 i C_3. Iznosi ovih konstanti variraju od slike do slike ovisno o broju razina boje i broju bitova po pikselu slike. Za 8-bitnu crno-bijelu sliku s 256 razina sive boje ove konstante se računaju na sljedeći način:

$$C_1 = (K_1 L)^2, C_2 = (K_2 L)^2, C_3 = \frac{C_2}{2} \quad (2\text{-}13)$$

gdje je L = 256, K_1 = 0.01 i K_2 = 0.03.

Samo računanje SSIM-a odvija se u koracima, gdje svaki korak obuhvaća dio slike odnosno prozor dimenzija 8x8 piksela koji se pomiče preko cijele slike piksel po piksel. Na kraju svakog koraka (ovakvog pomaka) računa se tzv. lokalni SSIM rezultat. Konačni rezultat cijele slike naziva se MSSIM (eng. *Mean SSIM*) ili prosječni SSIM i računa se kao obična aritmetička sredina svakog od lokalnih rezultata.

3. IMPLEMENTACIJA SUSTAVA AUTOMATSKE OPTIMIZACIJE SLIKA ZA WEB

Nakon izložene teorijske osnove potrebne za razumijevanje implementacije ovakvog sustava slijedi i opis samog rada i tehnologija potrebnih za njegovu realizaciju.

3.1. Korištene tehnologije

Dvije glavne tehnologije korištene za realizaciju ovog diplomskog rada u njegovom praktičnom dijelu su Ruby on Rails i Docker. U nastavku slijedi pregled svake od njih i opis njihove važnosti za rad.

3.1.1. Ruby on Rails

Ruby on Rails je framework za izradu web aplikacija koje su pogonjene Ruby programskim jezikom. Omogućuje fleksibilnu i dinamičnu izradu kompletnih web stranica zajedno s bazama podataka. Izuzetno je proširiv i fleksibilan zahvaljujući ogromnom broju dostupnih programskih proširenja (eng. *gems*) kreiranih od strane drugih korisnika, koji se otvoreno dijele na internetu. Koristeći ove gemove, Ruby on Rails može proširiti svoju osnovnu funkcionalnost na gotovo svaki zamisliv način.

Osnova funkcioniranja Ruby on Rails frameworka je MVC struktura. MVC (eng. *Model – View – Controller*) označava način komunikacije stranice same sa sobom te njenu reakciju na zahtjeve web preglednika. Ukratko, stranica putem akcija definiranih u kontroleru šalje zahtjeve i razmjenjuje informacije s modelima koji služe kao sučelja za komunikaciju s bazom podataka. Te zahtjeve zatim prosljeđuje natrag kontroleru koji ih upućuje odgovarajućem pogledu (view, u svojoj srži dio stranice koji korisnik vidi). Ovakav model rada stranice izuzetno je fleksibilan i prilagodljiv, te omogućava veliku slobodu pri realizaciji stranice i definiranju mogućih akcija.

Za potrebe ovog rada Ruby on Rails framework korišten je za izradu korisničkog sučelja, baze podataka te metoda za identifikaciju slika koje će se komprimirati. Osim toga, Ruby on Rails

zadužen je za pokretanje skripte koja će vršiti samu zadaću komprimiranja te za spremanje dobivenih komprimiranih slika.

3.1.2. Docker

Docker je sučelje za gradnju i održavanje prijenosnih virtualnih softverskih okruženja. Oslanja se na Go jezik za svoje funkcioniranje jer je u njemu napisan, no može funkcionirati u gotovo svim jezičnim okruženjima.

Docker postavlja još jedan sloj apstrakcije između sustava i korisnika za upravljanje razvojnim okruženjima. Za razliku od nekih drugih rješenja slične namjene, Docker ne zahtijeva instalaciju i pokretanja neke Linux distribucije u virtualnom okruženju, već za realizaciju svoje funkcionalnosti koristi samo i isključivo svojstva Linux kernela.

Princip funkcioniranja Dockera je sljedeći: on pri pokretanju implementira API visoke razine kako bi omogućio pokretanje pojedinih procesa aplikacije u izolaciji (u obliku tzv. *containera)* i pomaže programeru pri komunikaciji s pojedinim modulima programskog rješenja. Automatizira konfiguraciju virtualnih okruženja tako da korisnik ne mora izravno koristiti niti jedan drugi softver za virtualizaciju. Svu konfiguraciju i hardverske/softverske zahtjeve Docker sprema u takozvani Dockerfile dokument kako bi se kopiranjem na neki drugi uređaj mogli savršeno replicirati uvjeti virtualnog okruženja. U praksi ovo znači kako je sustav realiziran Dockerom potpuno prenosiv i u većini slučajeva neće ovisiti o operacijskom sustavu, programskom jeziku ili nekom drugom faktoru.

3.2. Realizacija projektnog dijela diplomskog rada

U ovom potpoglavlju biti će opisani najbitniji dijelovi same aplikacije, uz pozivanje na programski kod gdje to bude potrebno. Posebna pažnja biti će pridodijeljena bazi podataka i postupku njenog popunjavanja slikama, samoj skripti za optimizaciju slika, te dizajnu aplikacije i prolasku kroz proces odlučivanja o izgledu sučelja i njegovoj funkcionalnosti.

3.2.1. Baza podataka i njeno popunjavanje slikama

Za potrebe ovog diplomskog rada korištena je SQLite3 relacijska baza podataka. "Relacijska" znači da su između pojedinih objekata baze uspostavljene veze kako bi se izrazila njihova povezanost. SQLite3 baza dolazi već postavljena za korištenje s Rails framework-om, te je za nekomercijalnu uporabu više nego dostatna.

Najveći problem pri realizaciji rada što se interakcije s bazom tiče bilo je ostvariti dostupnost obje verzije slika u bilo kojem trenutku – kako komprimirane, tako izvorne. Ovom problemu se doskočilo tako što je u bazu spremljena putanja izvorne slike, a pri optimizaciji slike se ta izvorna putanja prepisala novom, komprimiranom slikom. Za ovaj postupak bilo je potrebno prvo identificirati i u bazu spremiti putanju do izvorne slike, što radi isječak koda na slici 3.1.

```
files.sort.each do | path |
unless File.directory ? (path)
filetype = `file #{path}`
if filetype.split(" ")[1].eql ? "JPEG"
image = path.sub(/ \._original$ / , "")
images[image] = {} if images[image].nil ?

if image.eql ? path
images[image][:short] = path.to_s
else
images[image][:long] = path.to_s
end

end
end
end
```

Slika 3. 1. Dio koda koji prolazi kroz sve slike i dohvaća njihove putanje

Nakon dohvaćanja putanja, slijedi učitavanje svake slike iz direktorija pojedinačno. Ovo je odrađeno na način da se za svaku sliku provjeri je li već optimizirana i ako je, spremaju se putanje izvorne i optimizirane slike u posebni stupac u bazi. Ako nije, sprema se samo putanja izvorne slike. Ovo osigurava da se već optimizirane slike neće više puta optimizirati, te osigurava dostupnost izvorne i optimizirane slike u svakom trenutku, što je vidljivo u isječku koda na slici 3.2.

```
puts "Loading images"
images.each do | key, image |
if image[:long].nil ?
original = image[:short]
optimized = nil
else
original = image[:long]
optimized = image[:short]
end

imgdata = {}
imgdata[original] = `identify - format "%[fx:w] %[fx:h] %b" #{original}`.split(" ")
imgdata[optimized] = `identify - format "%[fx:w] %[fx:h] %b" #{optimized}`.split("
") unless optimized.nil ?

newimg = Image.new
newimg.optimized = optimized.nil ? ? 0 : 1
```

Slika 3. 2. Dio koda koji učitava svaku sliku i provjerava stanje optimizacije

Za kraj, objekt kojim se barata se populira isčitanim podacima ovisno o stanju optimizacije učitanih slika, te se ovako nastali objekt sprema u samu bazu. Pritom se u bazu spremaju putanje, izvorne dimenzije i optimizirane dimenzije slika, ovisno o tome je li slika optimizirana ili ne. Ovaj postupak prikazan je kodom sa slike 3.3.

```
newimg.original_path = original[6.. - 1]
newimg.original_width = imgdata[original][0].to_i
newimg.original_height = imgdata[original][1].to_i
newimg.original_size = imgdata[original][2].to_i

newimg.optimized_path = optimized[6.. - 1] unless optimized.nil ?
newimg.optimized_width = imgdata[optimized][0].to_i unless optimized.nil ?
newimg.optimized_height = imgdata[optimized][1].to_i unless optimized.nil ?
newimg.optimized_size = imgdata[optimized][2].to_i unless optimized.nil ?

newimg.save

putc "."
end
```

Slika 3. 3. Dio koda koji učitane vrijednosti dodjeljuje „newimg“ objektu i sprema ga u bazu

3.2.2. Algoritam optimizacije

Najvažniji dio ove aplikacije zasigurno je skripta koja obavlja sam postupak komprimiranja slika. Ovaj program napisan je u Ruby programskom jeziku potpomognut dijelovima napisanim u bash interpreteru koji se izvršava na sustavima temeljenim na Unixu, no obzirom da je cijela aplikacija pogonjena Dockerom koji omogućava višeplatformsko izvršavanje na bilo kojem sustavu, ova ovisnost je izbjegnuta.

Funkcionalnost aplikacije realizirana je korištenjem tzv. *Active Job* komponenti koje su dio Ruby on Rails framework-a. Active Job-ovi su pozadinski zadaci koji se izvršavaju na principu reda izvođenja (eng. *queue*) te se izvode asinkrono i neovisno o funkcioniranju ostatka aplikacije. U praksi ovo znači da će se zadatak pokrenuti kad korisnik to zatraži, dok ostatak aplikacije zbog izvođenja zadatka neće biti blokiran ili zakinut. Bash komponente ovog programa barataju samim slikama, čitajući ih i zapisujući na disk ovisno o operacijama koje se nad njima izvršavaju, te pozivajući algoritme za optimizaciju kada je to potrebno.

Sam Active Job razdvojen je na tri glavne komponente, odnosno metode. Prva je *perform* metoda koja igra ulogu glavne metode; učitava sliku te poziva ostale dvije metode nad njom. Ova metoda prikazana je na slici 3.4.

```
def perform(*args)
image = args[0]
origpath = get_origpath image
optipath = get_optipath image
if optipath.nil ?
workpath, quality = optimize(origpath)
save_optimized(image, workpath, quality)
       end

end
```

Slika 3. 4. Perform metoda koja dohvaća putanje slike, poziva „optimize“ metodu te „save_optimized“ metodu nakon izvršenja optimizacije

Druga važna metoda je već spomenuti *optimize* u kojoj je sadržan glavni dio koda za pozivanje algoritama optimizacije i kompresije. Kao argument prima putanju izvorne slike, sprema ju kao *naziv_izvorne_slike.original* u privremeni spremnik (*workpath*) kako se izvorna slika

kompresijom ne bi izgubila te određuje stupanj kompresije. Ovaj postupak odvija se određivanjem idealnog faktora kvalitete *q* koji se dobiva iz DSSIM metrike kvalitete slike.

DSSIM je u svojoj biti mjera strukturne različitosti dobivene slike od izvorne slike, i obrnuto je proporcionalna SSIM metrici o kojoj je bilo riječi u teorijskom dijelu rada. Ova metoda prolazi kroz algoritam kompresije nekoliko puta sve dok se ne dobije idealni stupanj kompresije, nakon čega se dobivena komprimirana slika sprema u privremeni spremnik i metoda završava s izvođenjem i vraća *workpath* prethodnoj metodi. Metoda *optimize* prikazana je kodom na slici 3.5.

```
def optimize(origpath)

puts "Copy orig file"
digest = Digest::SHA1.hexdigest(File.read(origpath))
workpath = "/ramdisk/#{digest}"
`mkdir "#{workpath}"`
`cp "#{origpath}" "#{workpath}/original.jpg"`

puts "Convert to png"
`convert - quality 1 "#{workpath}/original.jpg" "#{workpath}/original.png"`

iter = 1
qlow = 10
qhigh = 90
dssim_low = 0.0038
dssim_high = 0.0042
dssim_avg = (dssim_high - dssim_low) / 2.0

dssim = 1
while (dssim < dssim_low or dssim > dssim_high) and iter<7
	q = (qhigh + qlow) / 2
	puts "Iteration #{iter}"
	puts "High #{qhigh}, Low #{qlow}, Q #{q}"
	`~ / mozjpeg / build / cjpeg - quality #{q} "#{workpath}/original.jpg" >
"#{workpath}/#{q}.jpg"`
	`convert - quality 1 "#{workpath}/#{q}.jpg" "#{workpath}/#{q}.png"`
	dssim = `~ / dssim / bin / dssim "#{workpath}/original.png"
"#{workpath}/#{q}.png"`
	dssim = dssim.split(" ")[0].to_f
	puts "DSSIM: #{dssim}"
	if dssim > dssim_high
		qlow = q + 1
```

```
            #q += 10
            elsif dssim < dssim_low
                  qhigh = q - 1
                  #q -= 10
            end

            iter += 1
      end

      `mv "#{workpath}/#{q}.jpg" "#{workpath}/optimized.jpg"`

      return[workpath, q]
      end
```

Slika 3. 5. Optimize metoda koja prima putanju izvorne slike, pretvara ju u .png format, traži DSSIM i faktor kvalitete q te na temelju njih poziva algoritam za kompresiju koliko god puta to bilo potrebno

Konačno, poziva se *save_optimized* metoda koja ima vrlo jednostavnu zadaću spremanja dobivenih putanja izvorne (sada preimenovane u *naziv.original*) slike te optimizirane slike iz *workpath* privremenog spremnika na sam disk. Pritom se izvorna slika prepisuje dobivenom optimiziranom slikom kako bi se aplikacija mogla primijeniti na postojeće web stranice bez potrebe za refaktoriranjem i mijenjanjem izvornog koda – sve slike ostaju učitane na stranici, samo što su to sad nove, komprimirane slike a ne izvorne. Metoda *save_optimized* prikazana je kodom sa slike 3.6.

```
def save_optimized(image, workpath, quality)

abs_old_path = get_origpath image
abs_new_path = abs_old_path.to_s + "._original"
abs_opt_path = abs_old_path

rel_orig = image.original_path.to_s + "._original"
rel_opti = image.original_path.to_s

`cp "#{abs_old_path}" "#{abs_new_path}"`
`cp "#{workpath}/optimized.jpg" "#{abs_opt_path}"`
`rm - rf "#{workpath}"`

optimized_data = `identify - format "%[fx:w] %[fx:h] %b" #{abs_opt_path}`.split("
")
image.optimized_width = optimized_data[0].to_i
```

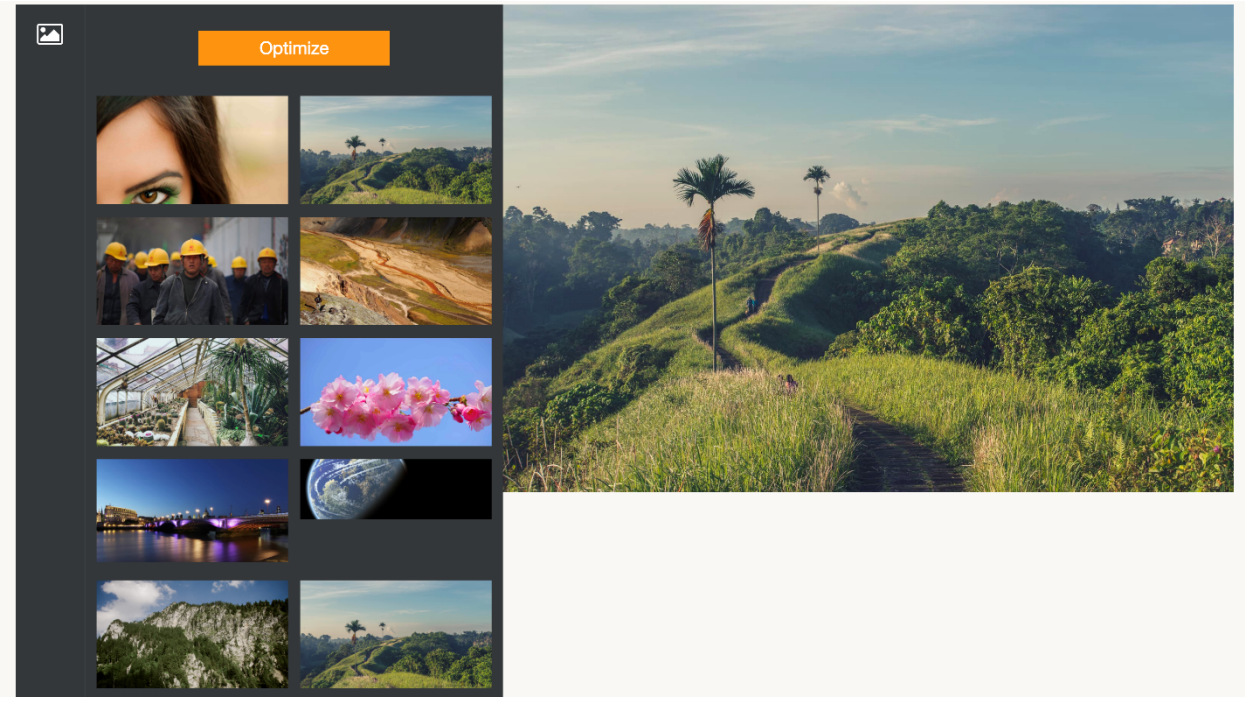
```
image.optimized_height = optimized_data[1].to_i
image.optimized_size = optimized_data[2].to_i
image.original_path = rel_orig
image.optimized_path = rel_opti
image.optimized = 1
image.quality = quality
image.encoder = "mozjpeg"
image.save
end
```

Slika 3. 6. Save_optimized metoda koja sprema slike na disk nakon optimizacije

3.2.3. Aplikacija za optimizaciju

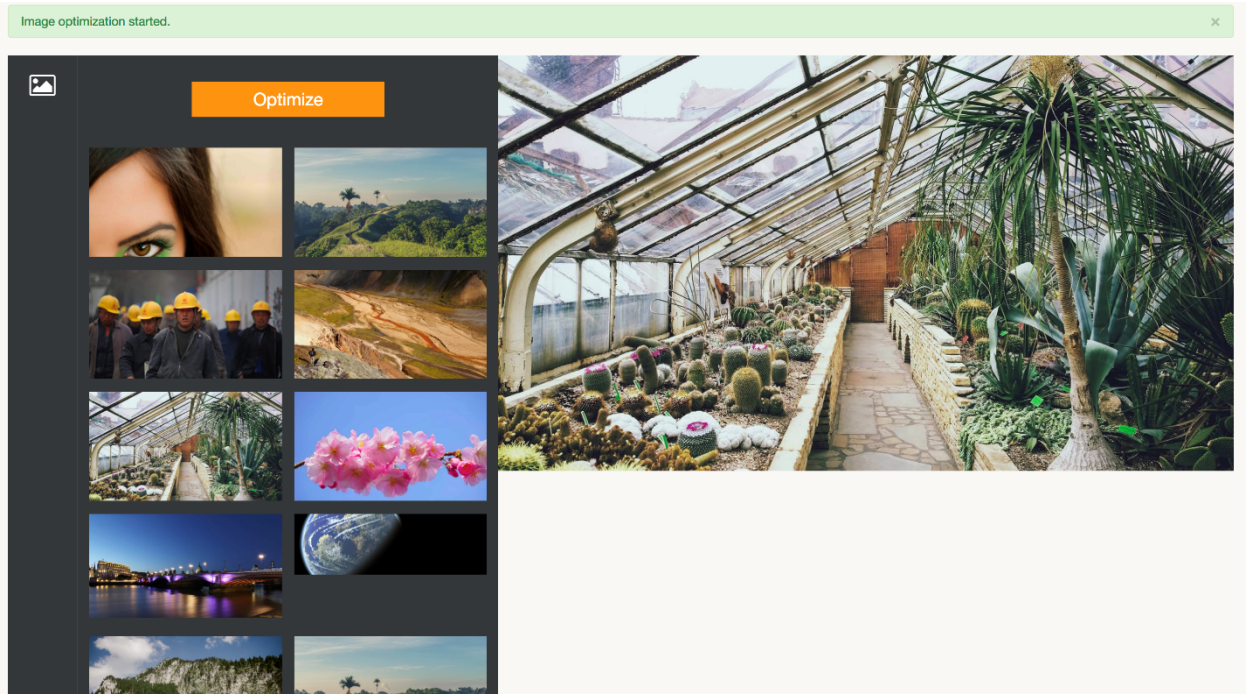
Sama aplikacija pogonjena je Ruby on Rails framework-om koristeći Ruby-em obogaćeni HTML kod, uz asinkrone pozive za prikazivanje pojedine slike bez ponovnog učitavanja čitave stranice. Ovaj dio realiziran je AJAX-om i JavaScript-om. Pokretanje aplikacije je trivijalno: potrebno je samo nakon instaliranja Docker-a pokrenuti *imageclinic run* naredbu u direktoriju unutar kojeg se nalaze slike koje korisnik želi optimizirati. U pozadini se digne Ruby on Rails server na adresi *localhost:3000* te se pristupanjem toj adresi otvara samo korisničko sučelje aplikacije.

Aplikacija se sastoji od dva glavna dijela: lijevog preglednika svih učitanih slika u obliku pločica kroz koje se može *scrollati*, i desnog pogleda za trenutno odabranu sliku. Dizajn je minimalistički, uz namjerni odabir prikaza u obliku samo jedne stranice radi jednostavnosti korištenja i intuitivnosti rada. Osim ovih elemenata, vjerojatno najvažniji dio stranice je gumb „Optimize“ na gornjoj lijevoj strani prikaza koji pokreće optimizaciju svih učitanih slika. Izgled aplikacije pri pokretanju može se vidjeti na slici 3.7.



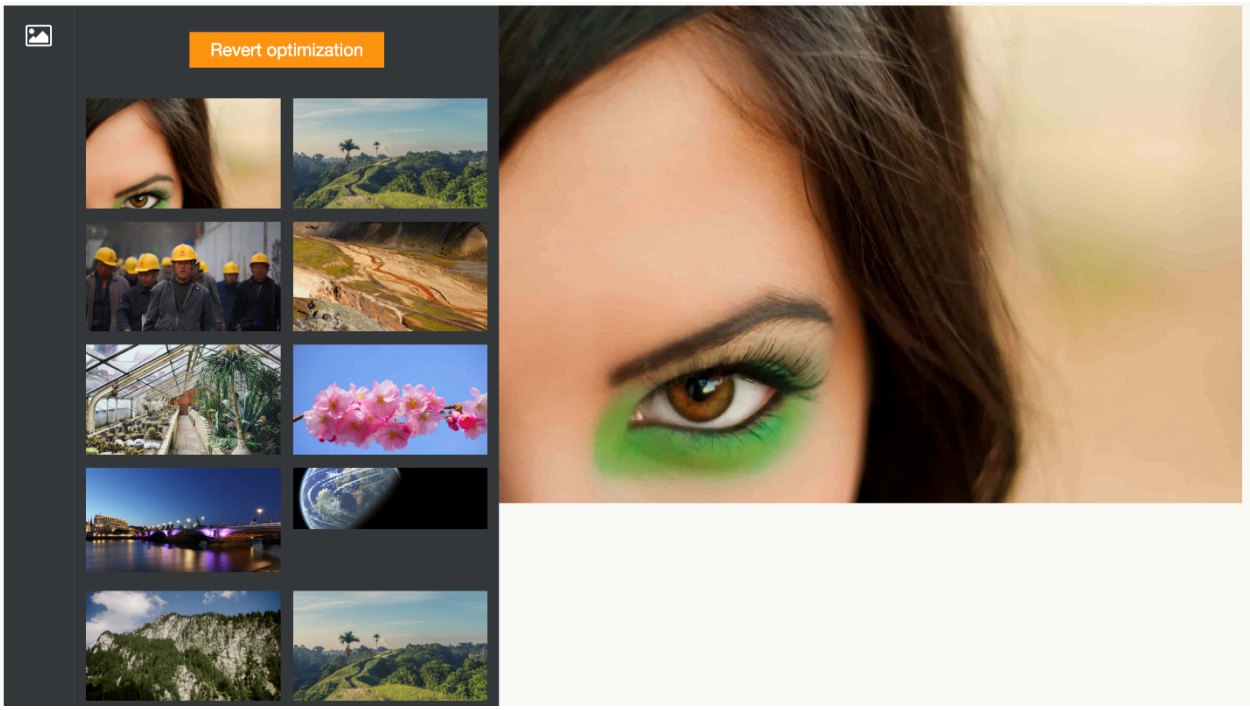
Slika 3. 7. Aplikacija nakon pokretanja servera i učitavanja adrese localhost:3000

Klikom na gumb „Optimize“ pokreće se automatska optimizacija svih učitanih slika iz direktorija. Nakon pozadinskog pokretanja skripte, moguće je promatrati napredak postupka pretvorbe slika u konzoli. Ovaj proces varira u trajanju od jedne do 10ak sekundi, ovisno o veličini učitane slike i broju potrebnih ponavljanja algoritma radi određivanja idealne stope kompresije. Korisnik je o pokretanju postupka optimizacije obaviješten notifikacijom na vrhu ekrana, što je vidljivo na slici 3.8.



Slika 3. 8. Obavijest o pokretanju optimizacije nakon pritiska na gumb „Optimize“

Nakon odrađene kompresije i optimizacije, korisnik može osvježiti stranicu te se gumb „Optimize“ pretvara u gumb „Reverse optimization“, što je dodatna mogućnost aplikacije. Ukoliko korisnik nije zadovoljan rezultatima, pritiskom na ovaj gumb pokreće se automatsko dekompresiranje slika te se one vraćaju u svoje prvotno stanje, a optimizirane slike se brišu. Cijeli postupak traje znatno kraće od postupka kompresije. Promjena stanja gumba može se uočiti na slici 3.9.



Slika 3. 9. Dinamičko mijenjanje gumba „Optimize“ ovisno o stanju optimiziranosti slika

4. ZAKLJUČAK

Ovaj diplomski rad uhvatio se u koštac s problemom koji je sve prominentniji i zastupljeniji u svijetu punom web sadržaja, od čega su većina slike. Loša optimizacija slika na web stranicama dovodi do loše iskorištenog internetskog prometa i nepotrebnog preuzimanja suvišne količine podataka, kao i do sporijeg učitavanja stranica. Rješenje predloženo u ovom radu imalo je za cilj omogućiti brz i bezbolan način masovne i automatizirane kompresije slika odjednom, bez potrebe za ručnim smanjivanjem svake slike pojedinačno.

Kroz ovaj pisani dio rada opisani su postupci i mehanizmi na kojima se temelji projektna realizacija rada. Krenuvši od samih temelja i definicije kompresije, preko specifičnih algoritama i načina enkodiranja JPEG formata i zaključno s tehnologijama i alatima korištenim za realizaciju projektnog zadatka, ovaj rad se trudio što je jasnije i preciznije izložiti svu potrebnu problematiku ali i rješenja za razumijevanje iste.

Kompresija slika vrlo je složeno i široko područje, te na kraju dana pravog odgovora i konačnog rješenja nema. Makar ovo predloženo programsko rješenje može funkcionirati u mnogo slučajeva i olakšati rad s bilo kojom količinom slika nad kojima je potrebno izvršiti kompresiju, ono nikako nije savršeno niti je idealno u apsolutno svakoj situaciji. No, bitno je napomenuti da je cilj ovog rada kao takav ispunjen do kraja. Realizirano je potpuno, samostojeće rješenje koje može odrađivati definirani posao na više platformi i neovisno o vanjskim okolnostima.

LITERATURA

[1] Digital Image Compression, Utah State University, Utah, SAD, dostupno na: http://digital.cs.usu.edu/~xqi/Proposal/Chapter2.pdf [15. 6. 2017.]

[2] Most Commonly Used Image Formats on the Internet, W3Techs, dostupno na: https://w3techs.com/technologies/overview/image_format/all [15. 6. 2017.]

[3] R. F. Haines, S. L. Chuang, The Effects of Video Compression on Acceptability of Images for Monitoring Life Sciences Experiments, NASA, SAD, 1992., dostupno na: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19920024689.pdf [15. 6. 2017.]

[4] GIF Definition, TechTerms, dostupno na: https://techterms.com/definition/gif [15. 6. 2017.]

[5] Microsoft Windows Bitmap - Summary, Fileformat, dostupno na: http://www.fileformat.info/format/bmp/egff.htm [15. 6. 2017.]

[6] M. Marcus, JPEG Image Compression, Dartmouth College, Hanover, New Hampshire, SAD, 2014, dostupno na: https://math.dartmouth.edu/~m56s14/proj/Marcus_proj.pdf [15. 6. 2017.]

[7] M. S. Al-Ani, F. H. Awad, The JPEG Image Compression Algorithm, College of Computer, Anbar University, Anbar, Irak, 2013., dostupno na: http://www.e-ijaet.org/media/2I15-IJAET0715522_v6_iss3_1055to1062.pdf [15. 6. 2017.]

[8] G. K. Wallace, The JPG Compression Standard, Digital Equipment Corporation, Maynard, Massachusetts, SAD, 1991., dostupno na: http://www4.ncsu.edu/~rhee/export/papers/TheJPEGStillPictureCompressionStandard.pdf [16. 6. 2017.]

[9] Chroma Subsampling, YourDictionary, dostupno na: http://computer.yourdictionary.com/chroma-subsampling [16. 6. 2017.]

[10] Color Formats, EquaSYS Systementwicklung, dostupno na: http://www.equasys.de/colorformat.html [16. 6. 2017.]

[11] JPEG Conversion, W3Schools, dostupno na: https://www.w3.org/Graphics/JPEG/jfif3.pdf [16. 6. 2017.]

[12] N. Kashyap, J. Modi, The JPEG Image Code Format, Massey University of New Zealand, New Zealand, dostupno na: http://www.massey.ac.nz/~mjjohnso/notes/59731/presentations/jpeg.pdf [16. 6. 2017.]

[13] Mozilla Research Team, mozjpeg, github, dostupno na: https://github.com/mozilla/mozjpeg [16. 6. 2017.]

[14] Google Team, Guetzli, github, dostupno na: https://github.com/google/guetzli [16. 6. 2017.]

[15] mozjpeg Comparison to libjpeg-turbo, libjpeg-turbo, dostupno na: http://www.libjpeg-turbo.org/About/Mozjpeg [16. 6. 2017.]

[16] libjpeg-turbo Performance, libjpeg-turbo, dostupno na: http://www.libjpeg-turbo.org/About/Performance [24. 6. 2017.]

[17] An introduction to Image Compression, DebugMode, dostupno na: http://www.debugmode.com/imagecmp/ [10. 6. 2017.]

[18] R. Dosselmann, X. Dong Yang, A Formal Assessment of the Structural Similarity Index, University of Regina, Regina, Saskatchewan, Canada, 2008., dostupno na: http://www.cs.uregina.ca/Research/Techreports/2008-02.pdf [10. 6. 2017.]

SAŽETAK

U radu je obrađena tema automatske optimizacije slika za Web te je predloženo i realizirano konkretno programsko rješenje. Postavljanje okvira za uspješno predstavljanje teme započeto je definiranjem pojmova kompresije i specifično, kompresije slika.

Nakon uvođenja pojma JPEG kompresije slika i predstavljanja osnovnog oblika algoritma za JPEG enkodiranje, definirani su ključni dijelovi tog algoritma. Opisan je postupak prelaska iz RGB u YC_bC_r prostor boja, diskretna kosinusna transformacija, kvantizacija te postupak entropijskog kodiranja koji završava primjenom Huffmanovog algoritma.

U nastavku se predstavljaju konkretni primjeri JPEG kompresora te pregled i usporedba najpopularnijih i najraširenijih među njima. Usporedbom libjpeg i mozjpeg kompresora i isticanjem njihovih prednosti i mana stvara se podloga za praktično razumijevanje primjene ovakvih alata ne samo na ovom konkretnom radu, nego i generalno.

Konačno, uvodi se pojam metrike kvalitete slike te se definiraju osnovne matematičke relacije za svaku od njih, kao i njihova značenja. Na samom kraju se opisuje implementacija sustava automatske implementacije slika za Web realizirana u ovom radu, te se u kratkim crtama predstavljaju osnovne tehnologije potrebne za njegovo stvaranje.

Ključne riječi: Kompresija, JPEG enkodiranje, prostor boja, YC_bC_r, DCT, kvantizacija, Huffmanovo kodiranje, libjpeg, mozjpeg, metrika kvalitete slike, MSE, PSNR, SSIM, MSSIM, Ruby on Rails, Docker

Automated Optimization of Images for the Web

The topic of discussion in this thesis was the automated optimization of images for the Web, including the suggestion and implementation of a concrete programmed solution. Setting the framework for the successful presentation of this topic was initiated by first defining the terms *compression* and more specifically, *image compression*.

After introducing the term *JPEG compression* and presenting the basic form of the JPEG encoding algorithm, key parts of that algorithm were defined. The process of transitioning from the RGB *color space* to the YC_bC_r space was detailed, as well as the terms *discrete cosine transform, quantization* and *entropy coding*, concluding with the application of the *Huffman algorithm*.

By presenting concrete examples of JPEG compressors and comparing two of them, *libjpeg* and *mozjpeg*, the groundwork was set for understanding the application of such tools not only in the context of this thesis, but also in general.

Finally, the term *image quality metric* is introduced and the basic mathematical relations for each of them, as well as their respective relevance, are defined and explained. The paper concludes by describing the most important technologies used in the implementation of an automated image optimization solution.

Key words: Compression, JPEG encoding, color space, YC_bC_r, DCT, quantization, Huffman coding, libjpeg, mozjpeg, image quality metric, MSE, PSNR, SSIM, MSSIM, Ruby on Rails, Docker

ŽIVOTOPIS

Vlado Kopić rođen je 1992. godine u Osijeku. Pohađao je Osnovnu školu Milko Cepelić u Vuki, selu gdje i trenutno prebiva. Osnovnu je školu završio s odličnim uspjehom, a u osmom razredu predstavljao je školu na državnom natjecanju iz njemačkog jezika gdje je postigao sedmo mjesto.

Nakon osnovne škole upisuje III. Gimnaziju u Osijeku (matematičku gimnaziju) u kojoj također polučuje odličan uspjeh kroz sve četiri godine školovanja. U četvrtom razredu gimnazije predstavljao je školu na državnom natjecanju iz filozofije gdje je postigao drugo mjesto.

Ostvaruje izravan upis na Elektrotehnički fakultet u Osijeku, gdje i dalje studira kao redovan student druge godine diplomskog studija računarstva.