Web aplikacija za obradu digitalne slike

Ćosić, Hrvoje

Undergraduate thesis / Završni rad

2024

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:200:270411

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-30**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I
INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni prijediplomski studij Računarstvo

WEB APLIKACIJA ZA OBRADU DIGITALNE SLIKE

Završni rad

Hrvoje Ćosić

Osijek, 2024. godina.

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

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	Hrvoje Ćosić
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	A 4608, 08.10.2021.
JMBAG:	0165084583
Mentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 1:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 2:	izv. prof. dr. sc. Tomislav Keser
Naslov završnog rada:	%naziv_rada%
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	[Rezervirano: Hrvoje Ćosić] Potrebno je napraviti aplikaciju koja će od krajnjeg korisnika primati početni oblik digitalne slike u jednom od podržanih formata te parametre potrebne za njezinu obradu. U pozadini na serveru izvršavati će se pozivi na poseban korisnički program u kojemu se nalaze implementacije različitih algoritama za obradu digitalne slike. Nakon obrade, korisniku se prikazuje i nudi za preuzimanje modificirani oblik slike ili primjena nekog drugog filtera.

Datum ocjene pismenog dijela završnog rada od strane mentora:	02.07.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Izvrstan (5)
Datum obrane završnog rada:	18.7.2024
Ocjena usmenog dijela završnog rada (obrane):	Vrlo dobar (4)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	18.07.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 18.07.2024.

Ime i prezime Pristupnika:	Hrvoje Ćosić
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	A 4608, 08.10.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za obradu digitalne slike**

izrađen pod vodstvom mentora izv. prof. dr. sc. Alfonzo Baumgartner

i sumentora

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 pristupnika:

Sadržaj

1. UVOD 2

1.1. Zadatak završnog rada 2

2. PREGLED PODRUČJA TEME 4

3. PRIKAZ SLIKE U RAČUNALU 6

3.1. Rezolucija 6
3.2. Piksel 6
3.3. Kanal 7

4. OBRADA DIGITALNE SLIKE 9

4.1. Skaliranje slike 9
4.2. Isticanje rubova objekata sa slike 11
4.3. Niskopropusno filtriranje 12
4.3.1. Median filter 13
4.3.2. Gaussov filter 13
4.3.3. Kutijasti filter 13
4.4. Grayscale filter 14
4.5. Binarni filter 15
4.6. Okretanje slike 18
4.7. Razvrstavanje piksela 21

5. IZRADA APLIKACIJE ZA OBRADU DIGITALNE SLIKE 24

5.1. Poslužiteljski dio aplikacije 24
5.2. Klijentski dio aplikacije 27

6. ZAKLJUČAK 32

7. SAŽETAK 34

8. ABSTRACT 35

1. UVOD

U ovom završnom radu rješava se problem razvoja web aplikacije za obradu digitalnih slika, čiji je temeljni zadatak omogućiti krajnjem korisniku jednostavno slanje digitalnih slika u podržanim formatima te parametara potrebnih za njihovu obradu. Na temelju tih parametara, aplikacija šalje zahtjeve na poslužiteljsku stranu gdje se koriste specijalizirani korisnički programi s implementiranim algoritmima za obradu slika. Osnovna svrha aplikacije je omogućiti korisnicima da primjene različite operacije obrade slika poput filtriranja, poboljšanja kvalitete, ili primjene umjetničkih efekata. Korištenjem web sučelja, korisnici mogu vidjeti rezultate obrade u stvarnom vremenu i preuzeti modificirane slike ili primijeniti dodatne promjene. Specifično, aplikacija se fokusira na integraciju klijentskog i poslužiteljskog dijela kako bi pružila intuitivno korisničko iskustvo uz visoku razinu računalne učinkovitosti i pouzdanosti. Kroz interakciju s korisničkim sučeljem, korisnici mogu odabrati željene opcije obrade slika, a aplikacija će dinamički komunicirati s poslužiteljem radi izvršavanja odabranih operacija. U radu se detaljno istražuju osnovni koncepti obrade digitalnih slika, tehnike manipulacije, kao i tehnologije potrebne za razvoj stabilne i efikasne web aplikacije.

S razvojem tehnologije i sve većom dostupnošću moćnih osobnih računala i mobilnih uređaja, obrada slika postala je pristupačna široj populaciji, od profesionalnih fotografa do običnih korisnika. Cilj obrade slika može biti raznolik, od poboljšanja vizualne kvalitete ili izdvajanja značajki, sve do primjene umjetničkih efekata. U modernom dobu, digitalne slike igraju ključnu ulogu u komunikaciji, umjetnosti, znanosti i mnogim drugim područjima. Razvoj tehnologije omogućio je ne samo snimanje i pohranu slika u visokoj rezoluciji, već i njihovu obradu na način koji je prije bio nezamisliv. Obrada slika nije važna samo za estetske ili umjetničke svrhe, već ima primjenu u medicini, forenzici, robotici, te mnogim drugim industrijama. Web aplikacije korisnicima pružaju pristup funkcionalnostima za specijaliziranu upotrebu, kao što je obrada digitalne slike, bez potrebe za instalacijom specijaliziranog softvera na njihove uređaje. U današnje vrijeme, s obzirom na sveprisutnost interneta, web aplikacije postaju sve popularnije zbog svoje dostupnosti, jednostavnosti korištenja i fleksibilnosti.

Web aplikacija za obradu digitalnih slika iz ovog rada implementirana je s naglaskom na jednostavno korisničko sučelje koje omogućuje brzu navigaciju i interakciju. Kroz intuitivni dizajn, korisnici mogu lako manipulirati parametrima obrade slika i odabrati željene operacije, što olakšava proces prilagođavanja slika njihovim specifičnim zahtjevima.

U razvoju klijentskog dijela ove aplikacije koristi se suvremeni razvojni okvir Vue.js u kombinaciji s efikasnim rješenjima na strani poslužitelja također implementiranim u JavaScript tehnologijama. Klijentski dio aplikacije osmišljen je kako bi korisnicima pružio intuitivan način korištenja aplikacije, dok se poslužiteljski dio bavi učinkovitim izvršavanjem zahtjeva za obradu slika na serveru. Ovakav pristup omogućuje jasnu razgraničenost između korisničkog sučelja i serverske logike, čime se osigurava skalabilnost i fleksibilnost aplikacije.

Uz pregled područja teme u kojem će se razmotriti postojeća rješenja na području ovog problema, ovaj rad predstavit će osnovne pojmove koji stoje iza svake digitalne slike, istražiti različite tehnike manipulacije istih, te opisati proces izrade web aplikacije za obradu digitalnih slika, uključujući razvoj korisničkog sučelja i poslužiteljskog dijela aplikacije.

1.1. Zadatak završnog rada

Potrebno je napraviti aplikaciju koja će od krajnjeg korisnika primati početni oblik digitalne slike u jednom od podržanih formata te parametre potrebne za njezinu obradu. U pozadini na serveru izvršavati će se pozivi na poseban korisnički program u kojemu se nalaze implementacije različitih algoritama za obradu digitalne slike. Nakon obrade, korisniku se prikazuje i nudi za

preuzimanje modificirani oblik slike ili primjena nekog drugog filtera.

2. PREGLED PODRUČJA TEME

U pregledu različitih platformi za obradu digitalne slike, vrlo je jednostano pronaći poveći broj alata koji nude raznovrsne mogućnosti za profesionalce i amatere u području fotografije, dizajna i vizualnih umjetnosti [1]. Adobe Photoshop [2] je neupitni lider u obradi digitalnih slika, često nazivan standardom u industriji. Razvijen od strane Adobe-a, ovaj softver nudi neusporedivu dubinu i širinu funkcionalnosti za profesionalce i entuzijaste diljem svijeta. Svojom sposobnošću da uređuje i komponira rasterizirane slike u slojevima, podržava maske, alfa kompoziciju i različite modele boja, Photoshop se ističe kao ključni alat za stvaranje visokokvalitetnih digitalnih umjetničkih djela. Osim što omogućuje složene manipulacije slikama, Photoshop također ima ograničene mogućnosti za rad s tekstom, vektorskom grafikom i 3D elementima, čime zadovoljava širok spektar kreativnih potreba. Korisnici mogu proširiti mogućnosti programa kroz razne dodatke koji nude dodatne funkcionalnosti i alate. Od svojih početaka, nazivni sustav Photoshopa evoluirao je kroz različite generacije, od verzija označenih brojevima do uvođenja modela pretplate. Danas, Ono što Photoshop čini posebnim među alatima za obradu digitalne slike je njegova integracija s drugim Adobe aplikacijama kao što su Illustrator, InDesign i Lightroom, čineći ga ključnim dijelom svog ekosustava. Kao dio "Adobe Photoshop obitelji", program je dostupan i u mobilnoj verziji za iPad, pružajući korisnicima mogućnost da rade na svojim projektima dok su u pokretu. Sa svojom stalnom evolucijom i nadogradnjama, Photoshop ostaje jedan od najboljih izbora za sve koji traže vrhunske rezultate u digitalnoj obradi slika.

GNU Image Manipulation Program [3], poznat pod akronimom GIMP, potpuno je besplatan izbor za digitalnu obradu slika koji nudi širok spektar naprednih alata za retuširanje, stvaranje grafika i drugih funkcionalnosti. GIMP se ističe po svojoj prilagodljivosti i podršci za različite operativne sustave, što ga čini popularnim među korisnicima koji traže alternativu komercijalnim softverima. Predstavlja besplatan i alat otvorenog tipa za obradu raster grafike koji se koristi za širok spektar manipulacija i uređivanja slika. Umjesto da bude samo alat za crtanje, GIMP pruža napredne mogućnosti za retuširanje fotografija, slobodno crtanje, konverziju između formata slika te podršku za različite specijalizirane zadatke. Razvija se kroz zajedničke napore volontera povezanih s GNU i GNOME projektima, s otvorenim pristupom preko Git repozitorija i javnih komunikacijskih kanala. Inovacije u GIMP-u prolaze kroz rigorozan proces prije nego što postanu dio glavne verzije, što osigurava da nove funkcionalnosti ne naruše stabilnost programa. Ovo dovodi do toga da neke značajke mogu čekati godinama prije nego što postanu dostupne za širu upotrebu. Izvorni kod aplikacije je javno dostupan, što znači da se različiti distributeri mogu brinuti za stvaranje instalacijskih paketa za različite operativne sustave. Tim GIMP-a redovito se susreće na različitim konferencijama, gdje se raspravlja o budućem smjeru razvoja.

Canva [4] je popularna platforma za uređivanje grafika i fotografija, poznata po dobro razvijenoj ravnoteži između jednostavnosti upotrebe i sveobuhvatnosti pruženih funkcionalnosti. Canva je značajno unaprijedila svoje mogućnosti lansiranjem Magic Studio [5] paketa koji koristi umjetnu inteligenciju za napredno uređivanje slika, pa čak i videozapisa. Ovaj dodatak čini Canvu različitom od drugih alata, te jednom od najnaprednijih dostupnih platformi za uređivanje, s naglaskom na generativnu umjetnu inteligenciju. Canva nudi najjednostavnije i najkorisnije alate za uređivanje fotografija koje koriste umjetnu inteligenciju na jednom mjestu, što je čini izuzetno korisnom za početnike i amaterske dizajnere. Magic Studio uključuje neke od značajki osposobljenih umjetnom inteligencijom koje bi mogle potpuno revolucionirati stvaranje sadržaja na društvenim mrežama. Na primjer, korisnici mogu jednostavno upisati ono što žele vidjeti, a Canva će im pružiti kompletno generiranu digitalnu sliku bez potrebe za ručnim obrađivanjem. Jedna od jedinstvenih značajki Magic Studia je također alat koji omogućuje

izdvajanje elemenata iz fotografija, poput osoba ili objekata, kao transparentne PNG datoteke. Ovo je korisno za daljnju upotrebu tih elemenata unutar drugih projekata na Canvi ili za izvoz i upotrebu u drugim aplikacijama. Platforma nudi širok raspon unaprijed pripremljenih predložaka za društvene mreže, dokumente i druge vrste sadržaja, kao i ilustracije te fotografije. Uređivanje u Canvi je intuitivno i omogućava korisnicima da jednostavno povlače elemente, slažući ih jedan ispred ili iza drugog. Prava snaga Canve leži u funkcionalnostima vezanim za suradnju, gdje korisnici mogu kreirati timove i zajednički raditi na dizajnima, koristeći zajedničke resurse i ostavljajući povratne informacije. Plaćeni plan sadrži funkciju koja omogućava pohranu logotipa, boja i fontova brenda, čime se osigurava dosljednost u svim marketinškim materijalima. Iako je Canva usmjerena prema početnicima i amaterskim dizajnerima, nudi većinu značajki koje koriste profesionalci, što je čini svestranim alatom za grafički dizajn. U usporedbi s drugim alatima poput Adobe Expressa, Canva se ističe po naprednijim AI alatima i integraciji u jedinstveni paket koji je pristupačan i jednostavan za korištenje. Njena sposobnost podrške timskom radu i održavanja dosljednosti brenda čini je idealnim alatom za male tvrtke i timove koji rade na daljinu ili angažiraju vanjske suradnike.

Sva navedena rješenja su razvijana na vrlo visokoj razini kroz niz godina u afirmiranim softverskim tvrtkama te sadrže mnoštvo naprednih funkcionalnosti. Iako, u usporedbi s navedenim rješenjima, manjkav brojem i razvijenošću nekih od spomenutih funkcionalnosti, rješenje predstavljeno u ovom radu pruža jednostavniji pristup funkcionalnostima obrade digialne slike jer korisnik nije primoran registrirati se na platformu (kao što je slučaj kod rješenja kao što su Adobe Photoshop i Canva) niti preuzeti softver na svoje računalno prije mogućnosti korištenja samog softvera (kao što je slučaj kod rješenja kao što je GIMP). Manji broj kombinacija mogućnosti vezanih uz obradu slike te jednostavnije korisničko sučelje predstavlja rješenje iz ovog rada kao bolju opciju iz stajališta korisničkog iskustva za vrlo jednostavne potrebe vezane uz obradu digitalne slike. Bez obzira na trenutnu razinu razvijenosti ovog rješenja, sama struktura i modularnost u arhitekturi implementacijskog koda omogućava laka buduća proširenja u slučaju potrebe za dodatnim funkcionalnostima.

3. PRIKAZ SLIKE U RAČUNALU

Prikaz digitalne slike u računalu temelji se na kompleksnom nizu procesa koji omogućuju pretvorbu stvarnih vizualnih podataka u digitalni oblik koji računalo može obraditi i prikazati. Ovaj proces uključuje različite čimbenike, poput piksela, rezolucije i kanala boja, koji zajedno određuju razinu detalja i subjektivni doživljaj kvalitete digitalnih slika. Razumijevanje tih osnovnih pojmova ključno je za svladavanje tehnika obrade slike i razvoj učinkovitih alata za manipulaciju slika. U ovom poglavlju, istražit ćemo osnovne komponente digitalne slike, počevši od rezolucije, preko piksela, do kanala boja, kako bismo u daljnjim poglavljima mogli ući u dubinu tema koje neizbježno uključuju ove pojmove.

3.1. Rezolucija

Rezolucija je jedan od najvažnijih parametara digitalne slike i odnosi se na količinu detalja koju slika može prikazati. U kontekstu digitalne slike, rezolucija se obično mjeri u pikselima i predstavlja broj piksela po jedinici duljine, najčešće po inču (ppi - pixels per inch) ili po cijeloj slici (ukupan broj piksela).

Visoka rezolucija znači da slika sadrži više piksela i može prikazati finije detalje, dok niska rezolucija rezultira slikom s manje detalja i većom vidljivošću pojedinačnih piksela. Rezolucija slike utječe na kvalitetu ispisa, prikaz na ekranima različitih veličina te na vrijeme obrade i pohrane slike.
Primjeri rezolucije:

- HD (High Definition): 1280 x 720 piksela
- Full HD: 1920 x 1080 piksela
- 4K Ultra HD: 3840 x 2160 piksela

Povećanje rezolucije slike često zahtijeva veći kapacitet memorije za pohranu i veće računalne resurse za obradu. U web aplikacijama za obradu slika, važno je optimizirati slike kako bi se postigla ravnoteža između kvalitete i performansi. Razumijevanje rezolucije ključno je za pravilnu manipulaciju i prikaz slika u digitalnim sustavima.

Postoje različite tehnike za manipuliranje rezolucijom, kao što su bikubična interpolacija, bilinearna interpolacija ili interpolacija tehnikom najbližeg susjeda. Ove tehnike obraditi ćemo detaljnije u nekima od narednih poglavlja.

3.2. Piksel

Piksel, skraćenica za pojam elementa slike (engl. picture element), osnovna je jedinica digitalne slike. Svaka digitalna slika sastoji se od skupa piksela koji se prikazuju u redovima i stupcima. Pikseli su najmanji dijelovi slike koji se mogu pojedinačno obraditi i prikazati.

Svaki piksel određen je elementom boje. Kod prikaza boja u pikselima koristi se koncept kvantizacije boja. Dubina boje određuje koliko različitih nijansi svaka boja može imati u pikselu. Mjeri se u bitima, pri čemu je najčešća 8-bitna dubina boje koja omogućava prikazivanje 256 različitih nijansi za svaku od RGB komponenti. Korištenjem više bitova za prikaz boje omogućava prikaz više nijansi i precizniju boje. Veće dubine boje su posebno važne u profesionalnoj fotografiji i grafičkom dizajnu, gdje je vjernost boja ključna. S druge strane, kako bi uštedili na memorijskim zahtjevama slike, koriste se takozvane indeksirane boje. Indeksirane

boje predstavljaju poseban način kvantizacije boja koji se koristi u nekim slikovnim formatima. Umjesto da svaki piksel pohranjuje RGB vrijednosti, on referencira boju iz unaprijed definirane palete boja. Ovaj pristup zaista omogućava smanjenje veličine datoteke, ali je ograničen brojem dostupnih boja.

Neki od načina korištenja piksela kao primarni element manipulacije u obradi slike su:

- Manipulacija bojom: Promjenom vrijednosti komponenti piksela, možemo mijenjati boje unutar slike. Na primjer, pretvaranjem svih piksela iz slike u boji u nijanse sive boje (engl. grayscale), možemo stvoriti crno-bijelu verziju takve slike.
- Filtri i efekti: Primjenom različitih algoritama na piksele slike, možemo stvoriti različite vizualne efekte kao što su zamućenje, zaoštrenje, sepija, toniranje itd.
- Detekcija rubova: Analizom razlika u vrijednostima piksela, algoritmi mogu prepoznati rubove i obrise objekata unutar slike, što je korisno u mnogim primjenama računalnog vida ili jednostavno naglašavanju takvih rubova na slici, primjerice povećanjem intenziteta istih.

Kvaliteta i detalji slike ovise o broju piksela koje sadrži. Slika visoke rezolucije ima više piksela i može prikazati finije detalje, dok slika niske rezolucije sadrži manje piksela, što rezultira gubitkom razine detalja slike i povećanom vidljivošću pojedinačnih piksela na slici.

3.3. Kanal

Kanal u kontekstu digitalne slike predstavlja jednu komponentu boje slike. Svaka digitalna slika može biti razložena na nekoliko kanala, pri čemu svaki kanal sadrži informacije o intenzitetu određene boje ili svjetlosti. Razumijevanje kanala ključno je za manipulaciju bojama i primjenu različitih tehnika obrade slike.

Najčešći modeli boja i njihovi kanali uključuju:

- RGB model: Sastoji se od tri kanala - crvenog (R), zelenog (G) i plavog (B). Svaki kanal sadrži intenzitet odgovarajuće boje, što zajedno tvori punu boju svakog piksela na slici. Kombinacijom različitih intenziteta ovih triju kanala moguće je prikazati širok spektar boja. Na primjer, piksel s vrijednostima R=255, G=0, B=0 bit će potpuno crvene boje.
- Grayscale: Sadrži samo jedan kanal koji predstavlja intenzitet svjetlosti od crne (vrijednost 0) do bijele (vrijednost 255). Grayscale slike koriste ovaj kanal za prikaz svih nijansi sive boje, omogućujući jednostavniju obradu i analizu slike. Ovaj model je koristan u situacijama gdje sama boja piksela nije bitna, poput analize teksture ili oblika.
- CMYK model: Koristi se prvenstveno u tiskarskoj industriji i sastoji se od četiri kanala - cijan (C), magenta (M), žuta (Y) i ključna crna (K). Ovaj model boja temelji se na suptraktivnom miješanju boja, gdje se boje oduzimaju od bijele svjetlosti kako bi se dobile željene nijanse. Na primjer, kombinacija 100% cijan, 100% magenta, 0% žuta i 0% crna rezultirat će ljubičastom bojom.
- Binarni model: Sadrži jedan kanal s dvije moguće vrijednosti - crnu (vrijednost 0) i bijelu (vrijednost 1). Ovaj model koristi se za vrlo jednostavne slike, kao što su skice, grafovi ili tekst. Ovaj model je posebno koristan za OCR (engl. optical character recognition - OCR) i druge oblike analize jednostavnih uzoraka.

Manipulacija pojedinačnim kanalima omogućuje razne tehnike obrade slike, kao što su:

- Ekstrakcija kanala: Izvlačenjem pojedinačnih kanala iz slike, moguće je analizirati ili manipulirati samo određene komponente boje. Na primjer, izdvajanje crvenog kanala iz RGB slike omogućava analizu intenziteta crvene boje na slici, što može biti korisno u medicinskoj analizi slika za detekciju krvi ili drugih crvenih objekata.

- Poboljšanje kontrasta: Promjenom intenziteta unutar određenih kanala, moguće je poboljšati kontrast ili istaknuti određene dijelove slike. Na primjer, povećanjem kontrasta u crvenom kanalu može se pojačati vidljivost crvenih objekata na slici, što je korisno u prometnim sustavima za detekciju semafora.

- Kombinacija kanala: Kombiniranjem informacija iz različitih kanala, moguće je stvoriti nove slike ili primijeniti specifične efekte. Na primjer, zamjenom crvenog i zelenog kanala u RGB slici, može se postići zanimljiv vizualni efekt koji može biti koristan u umjetničkim aplikacijama ili u analizi slika za istraživanje.

- Filtriranje kanala: Primjenom različitih filtara na pojedinačne kanale, moguće je postići specifične efekte, poput zamućenja ili zaoštrenja određenih boja. Na primjer, zamućenje plavog kanala u RGB slici može se koristiti za smanjenje šuma u slici snimljenoj u uvjetima slabog osvjetljenja.

4. OBRADA DIGITALNE SLIKE

Web aplikacija za obradu digitalne slike sadržavati će implementacije različitih algoritama za obradu slike kako bi omogućila korisnicima transformaciju i prilagodbu slika prema njihovim potrebama, radilo se o specijalnim efektima, poboljšanju kvalitete slike, ili kodiranju podataka.

Općenito govoreći, postupci za obradu slike mogu se provoditi u prostornoj domeni ili domeni prostornih frekvencija. Prostorna domena odnosi se na samu ravninu slike, a metode obrade slike u ovoj domeni zasnovane su na izravnoj manipulaciji elementima slike, odnosno pikselima. Dvije najvažnije kategorije obrade slike u prostornoj domeni su promjene intenziteta na skali sive boje i prostorno filtriranje. U frekvencijskoj domeni, s druge strane, obrada slike se vrši manipulacijom prostornih frekvencija slike. Ova domena uključuje analizu i modifikaciju frekvencijskih komponenti slike, umjesto direktnog rada na pikselima kao što je slučaj u prostornoj domeni. Metode obrade u frekvencijskoj domeni često koriste transformacije, kao što je primjerice Fourierova transformacija, za prebacivanje slike iz prostorne u frekvencijsku domenu. Nakon primjene odgovarajućih filtriranja ili drugih operacija u frekvencijskoj domeni, slika se vraća u prostornu domenu pomoću inverzne transformacije. Obrada u frekvencijskoj domeni omogućava efikasno uklanjanje šuma, poboljšanje slike i ekstrakciju bitnih značajki.

Algoritmi implementirani u samoj web aplikaciji za obradu slike većinski uključuju postupke iz prostorne domene obrade, kao što su postupci za skaliranje, isticanje rubova, zamućenje, primjenu različitih filtara i druge transformacije koje poboljšavaju vizualni dojam ili prilagođavaju unešenu sliku za specifične primjene.

4.1. Skaliranje slike

Skaliranje slike je proces promjene veličine slike, bilo povećavanjem ili smanjivanjem njezinih dimenzija. Skaliranje može biti potrebno iz različitih razloga, kao što su prilagodba slika različitim uređajima, optimizacija za web stranice ili priprema za ispis. Skaliranje slike u bilo kojem smjeru možemo prikazati izrazom:

$$g(x, y) = f(\frac{x}{d}, \frac{y}{d}) \tag{4-1}$$

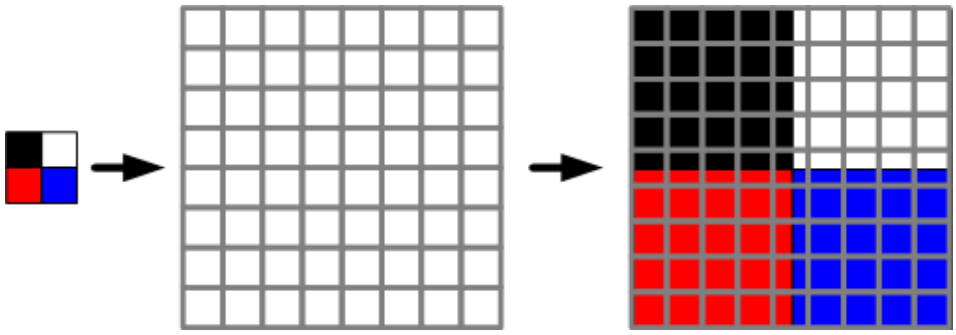
gdje ćemo uz koeficijent d1≥1 uvećati sliku, a uz d≤1 smanjiti sliku

Korištenjem procesa skaliranja slike, također moramo iskoristiti proces interpolacije piksela kako bi novonastala slika nadomjestila problem nastao nedostatkom ili viškom piksela. Najčešće korištene tehnike interpolacije su interpolacija najbližim susjednom, bilinearna interpolacija te bikubična interpolacija. Web aplikacija za obradu digitalne slike iz ovog rada implementira tehniku interpolacije najbližim susjedom. Interpolacija koristi postojeće podatke kako bi procijenila vrijednosti na nepoznatim mjestima [6] [1].

Interpolacija najbližim susjednom je najjednostavnija metoda interpolacije koja se koristi prilikom povećanja ili smanjenja slike. Ova metoda odabire najbliži piksel u izvornom koordinatnom sustavu i kopira njegovu vrijednost na novonastalu koordinatnu lokaciju. Računalno je učinkovit, a rezultirajući efekti mogu biti blokasti i manje glatki u usporedbi s drugim metodama interpolacije. Ovaj proces možemo izraziti formulom:

$$g(x, y) = f(INT(x + 0.5), INT(y + 0.5)) \tag{4-2}$$

gdje je INT funkcija za zaokruživanje ulazne vrijednosti na najbliži cijeli broj. Interpolacija najbližim susjedom za više piksela u izlaznoj slici koristi vrijednost jednog piksela ulazne slike, što doprinosi dojmu smanjenje kvalitete, odnosno manjoj rezoluciji izlazne slike.



SL. 4.1: *Slikoviti prikaz interpolacije piksela najbližim susjedom*

Procedura za skaliranje slike koristeći interpolaciju najbližim susjedom koja se koristi u servisnom dijelu web aplikacije za obradu digitalne slike prije samog algoritma za interpolaciju poziva pomoćnu proceduru za izjednačenje formata prikaza (engl. aspect ratio). Ona prilagođava širinu ili visinu slike kako bi zadržala originalni omjer stranica. Ova funkcija je korisna kako bi se izbjegla distorzija slike prilikom neprilagođene promjene jedne od dimenzija. Ova metoda prima sam objekt koji predstavlja pojedinačnosti slike, te traženu širinu i visinu izlazne slike. Omjer stranica izračunava se kao omjer širine i visine originalne slike. Ovaj omjer koristi se za prilagođavanje dimenzija. Koristi se samo jedna predana dimenzija, dok se druga dobiva iz prethodno izračunate vrijednosti omjera stranica.

```
1  void fixAspectRatio(IMAGE image, int* wProp, int* hProp) {
2      float aspectRatio = (float)image.width / image.height;
3      if (*wProp != 0) {
4          *hProp = (*wProp) / aspectRatio;
5      } else if (*hProp != 0) {
6          *wProp = (*hProp) * aspectRatio;
7      }
8  }
```

Primjer koda 1: *Procedura za ispravljanje željenih vrijednosti visine/širine slike*

Sam algoritam za skaliranje slike metodom najbližeg susjeda temelji se na jednostavnom postupku pronalaženja najbližeg piksela iz originalne slike za svaki piksel u novoj slici prilikom prolaska kroz svaki redak (vanjska petlja) te svaki stupac (unutarnja petlja) piksela slike. Algoritam koristi matematički izraz za interpolaciju, no također u obzir uzima broj kanala koji sama slika ima kako bi precizno oslikao svaki dio piksela.

```
1      double xRatio = originalImg.width / (double)newProps.width;
2      double yRatio = originalImg.height / (double)newProps.height;
3      int ch = originalImg.channelNumber;
4      for (int y=0; y<newProps.height; y++) {
5          for (int x=0; x<newProps.width; x++) {
6              int newPos = (x + y * newProps.width) * ch;
7              int orgPos = ( floor(x * xRatio) + (originalImg.width * floor(y * yRatio)) )
                   ↪ * ch;
8              for (int i=0; i<ch; i++)
9                  newProps.data[newPos + i] = originalImg.data[orgPos + i];
10         }
11     }
```

Primjer koda 2: *Implementacija algoritma za skaliranje slike metodom najbližeg susjeda*

4.2. Isticanje rubova objekata sa slike

Rubovi predstavljaju nagle promjene u intenzitetu, tj. diskontinuitete u svjetlini ili kontrastu slike, te se obično pojavljuju na granicama između različitih područja slike. Detekcija rubova je tehnika obrade slike za pronalaženje granica objekata unutar slika. Radi na način da nastoji pronaći dio slike gdje se nalazi diskontinuitet svjetline. Detekcija rubova često se koristi za segmentaciju slike i ekstrakciju podataka u područjima kao što su obrada slike i računalni vid. Rubovi su osobiti po velikim varijacijama u intenzitetu piksela. Promatranjem prve i druge derivacije intenziteta dvodimenzionalne slike, možemo učinkovito istaknuti ove promjene. Prva derivacija, ili gradijent, detektira brzinu promjene intenziteta i možemo ju definirati na sljedeći način [7]:

$$\nabla f = \frac{f(x+1,y) - f(x,y)}{f(x,y+1) - f(x,y)} \tag{4-3}$$

S druge strane, druga derivacija identificira nultočke, koje odgovaraju pozicijama rubova na digitalnoj slici. Nju možemo prikazati na sljedeći način:

$$\Delta f = \frac{f(x+1,y) - 2f(x,y) + f(x-1,1)}{f(x,y+1) - 2f(x,y) + f(x,y-1)} \tag{4-4}$$

Ove derivacije se mogu aproksimirati korištenjem konvolucijskih operacija sa specifičnim jezgrama (engl. kernel). U implementaciji web aplikacije za obradu slika, implementirana je metoda isticanja rubova koja koristi tzv. Sobelov operator (nekada označen kao Sobel-Feldmanov operator). Predstavlja derivativna masku koja se koristi za otkrivanje rubova u horizontalnom i vertikalnom smjeru. Ime je dobio po Irwinu Sobelu i Garyju M. Feldmanu, znastvenicima iz Stanfordskog laboratorija za umjetnu inteligenciju (SAIL). Sobel i Feldman predstavili su ideju o ovoj 3x3 konvolucijskoj jezgri na predavanju na SAIL-u 1968 [8]. Ovaj operator načelno koristi dvije jezgre dimenzije 3x3 koje se koriste za maskiranje originalne slike kako bi aproksimirali prethodno spomenute derivacije. Ove dvije jezgre, Hm za horizontalnu, a Hv za vertikalnu aproksimaciju derivacije, možemo prikazati na sljedeći način:

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

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

Implementacija algoritma za isticanje rubova objekata u web aplikaciji za obradu digitalne slike sastoji se od nekoliko koraka. Prije primjene Sobelovih operatora, slika se pretvara u grayscale format kako bi se istaknula luminancija svakog piksela pozivajući proceduru koja koristi postojeće podatke piksela za izračunavanje prosječne vrijednosti intenziteta boje i postavlja tu vrijednost kao novu vrijednost piksela, bez obzira na broj kanala u ulaznoj slici.

```
1 for (int i=0; i<img.size; i+=img.channelNumber) {
2         int ch = img.channelNumber;
3         int grayLvl = 0;
4         while(ch != 0) grayLvl += *(img.data + i + (--ch));
5         grayLvl /= img.channelNumber;
6
7         ch = img.channelNumber;
8         while(ch != 0) *(img.data + i + (--ch)) = grayLvl;
```

```
9
10 }
```

Primjer koda 3: *Petlja za izračun vrijednosti sive boje svakog piksela na slici*

Za svaki piksel slike, izračunavaju se horizontalni i vertikalni gradijenti koristeći Sobelove jezgre. Gradijenti predstavljaju promjenu intenziteta boje u odgovarajućem smjeru. Ukupni gradijent se izračunava kao Euklidska norma horizontalnog i vertikalnog gradijenta. Pikseli s ukupnim gradijentom većim od polovice maksimalne vrijednosti intenziteta postavljaju se na čisto bijelu boju, dok se ostali pikseli postavljaju na čisto crnu boju kako bi rubovi objekta poprimili boju "suprotnu" od ostatka slike.

```
1  int kernelX[3][3] = {
2          {1, 0, −1},
3          {2, 0, −2},
4          {1, 0, −1}
5  };
6  int kernelY[3][3] = {
7          {1, 2, 1},
8          {0, 0, 0},
9          {−1, −2, −1}
10 };
11
12 for (int y=img.channelNumber; y<img.height * img.channelNumber; y++) {
13         for (int x=0; x<img.width; x+=img.channelNumber) {
14                 int gradientX = getHorizontalGradient(img, y, x, kernelX);
15                 int gradientY = getVerticalGradient(img, y, x, kernelY);
16                 int totalGradient = sqrt( pow(gradientX, 2) + pow(gradientY, 2) );
17
18                 int ch = img.channelNumber;
19                 if (totalGradient > (int)255 / 2) {
20                         while(ch != 0) *(imgCopy.data + (img.width * y) + x + (−−
                               ↪ ch)) = 255;
21                 } else {
22                         while(ch != 0) *(imgCopy.data + (img.width * y) + x + (−−
                               ↪ ch)) = 0;
23                 }
24         }
25 }
```

Primjer koda 4: *Implementacija odvajanja piksela rubova objekta od ostalih piksela slike*

4.3. Niskopropusno filtriranje

Niskopropusno filtriranje, odnosno zamućenje slike je jedna od tehnika u obradi slika koja se koristi za smanjenje šuma i detalja unutar slike, odnosno omekšavanje oštrih rubova. Ova tehnika pomaže pri smanjenju artefakata na slici te pripremi slike za daljnju analizu, kao što je detekcija rubova ili segmentacija. Zamućenje slike postiže se filtriranjem slike, gdje se svaki piksel u slici zamjenjuje prosjekom ili težinskim prosjekom njegovih susjednih piksela. To rezultira "omekšavanjem" slike, gdje se oštri prijelazi u intenzitetu smanjuju. Neki od najčešće korištenih filtara za zamućenje su median filtar, kutijasti (engl. box) filtar i Gaussov filtar.

4.3.1 Median filter

Median filter je nelinearni filtar koji zamjenjuje vrijednost svakog piksela s medianom vrijednosti piksela u njegovom okruženju. Ovaj filter je posebno učinkovit u uklanjanju tzv. "sol i papar" šuma bez značajnog zamućivanja rubova.

Median filter koristi jezgru koja se kreće preko slike. Veličina prozora obično je 3x3, 5x5 ili 7x7 piksela. Za svaki piksel unutar prozora, median filter određuje median vrijednost svih piksela unutar tog prozora i zamjenjuje središnji piksel izračunatom vrijednošću.

Vrlo je učinkovit u uklanjanju sol-i-papar šuma jer zamjenjuje ekstremne vrijednosti s vrijednostima koje su bliže okolnim pikselima. Za razliku od linearnih filtara, median filter bolje zadržava rubove na slici jer ne koristi srednje aritmetičke vrijednosti koje mogu suviše zamutiti rubove.

S druge strane, zahtijeva sortiranje vrijednosti unutar prozora za svaki piksel, što može biti računalno intenzivno, posebno za veće prozore. Median filter također možda neće biti najbolji izbor za druge vrste šuma koji se mogu pojaviti na slici, kao što je Gaussov šum.

4.3.2 Gaussov filter

Gaussov filter je linearni filter koji se koristi u obradi slika koristeći Gaussovu funkciju za stvaranje težinskih prosječnih vrijednosti piksela u slici, pri čemu se bližim pikselima daje veća težina. Za izračun težina koje se primjenjuju na piksele unutar konvolucijske jezgre koristi se gausova funkcija koja se generira pomoću funkcije:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{4-5}$$

gdje je σ standardna devijacija Gaussove funkcije, a x i y udaljenosti od središta jezgre.

Standardna devijacija određuje širinu Gaussove funkcije i utječe na razinu zamućenja. Veća σ rezultira jačim zamućenjem. Jezgra se generira korištenjem Gaussove funkcije za izračun težina za svaki pojedinačni piksel unutar jezgre. Jezgra se pomiče preko slike, a svaka vrijednost piksela zamjenjuje se težinskom prosječnom vrijednošću piksela unutar jezgre.

Gaussov filter učinkovito uklanja Gaussov šum iz slike, stvarajući glatku sliku, a širina kernela i standardna devijacija omogućuju kontrolu razine zamućenja, što je korisno za mnoge primjene.

Ipak, Gaussov filter može zamutiti rubove slike jer koristi težinski prosjek, što može rezultirati gubitkom detalja na rubovima. Kao što je slučaj i kod median filtra, primjena Gaussovog filtra može biti računalno intenzivna, posebno za veće jezgre i slike visoke rezolucije.

4.3.3 Kutijasti filter

Kutijasti (engl. Box) filter je linearni filter koji koristi prosjek vrijednosti piksela unutar definirane konvolucijske jezgre kako bi stvorio zamućeni efekt na slici. Kutijasti filtar je računalno najmanje zahtjevan oblik zamućenja. Svaki piksel se zamjenjuje jednostavnom aritmetičkom sredinom vrijednosti piksela u njegovom okruženju. Konvolucijska jezgra korištena za ovaj filtar je obično jedinična matrica podijeljena s brojem elemenata u jezgri, to jest njenoj veličini. Zbog svojstva korištenja jednakih težina, može se implementirati korištenjem mnogo jednostavnijeg algoritma akumulacije, koji je znatno brži od korištenja algoritma kliznog prozora koji se koristi u standardnoj implementaciji [9].

Koristi pravokutnu jezgru u kojoj se sve težine postavljaju na jednaku vrijednost. Kada se primijeni na sliku, svaki piksel unutar jezgre dobiva istu težinu, a nova vrijednost piksela se izračunava kao aritmetički prosjek tih vrijednosti.

Veličina jezgre određuje područje piksela koje će biti uključeno u izračun prosjeka. Na primjer, jezgra veličine 3x3 uključuje 9 piksela. Kernel se pomiče preko slike, a za svaki položaj kernela, nova vrijednost središnjeg piksela izračunava se kao prosjek svih vrijednosti piksela unutar kernela.

Kutijasti filter je jednostavan za implementaciju, te je zbog svoje jednostavne strukture računalno učinkovit i brz za primjenu na sliku većih dimenzija. Ipak, kutijasti filter može značajno zamutiti rubove i fine detalje na slici jer koristi jednostavan prosjek bez težinskih vrijednosti.

Web aplikacija za obradu digitalne slike u ovom radu sadrži implementaciju kutijastog filtera u programskom jeziku C koristeći osnovni pristup zamućenja slike izračunavanjem prosječne vrijednosti susjednih piksela. Daljina susjednih piksela koji se uzimaju u obzir određeni su varijablom "offset" koja je određena korisnikovim unosom u programsko sučelje. Iteracija se vrši kroz sve piksele slike, te se za svaki piksel izračunava prosječna vrijednost susjednih piksela unutar jezgre, uzevši u obzir taj intenzitet zamućenja. Nakon što se obiđu svi susjedni pikseli, nova vrijednost izlaznog piksela se postavlja kao prosječna vrijednost zbrojenih susjednih piksela sa ulazne slike. Sama implementacija glavne petlje vidljiva je na slici 3.6.

```
1  for (int y=0; y<img.height * img.channelNumber; y++) {
2          for (int x=0; x<img.width; x++) {
3                  int neighborsSum = 0;
4
5                  int offset = img.channelNumber * intensity;
6                  neighborsSum += *(img.data + (img.width * y) + x);
7                  neighborsSum += *(img.data + (img.width * y) + x + offset);
8                  neighborsSum += *(img.data + (img.width * y) + x - offset);
9                  neighborsSum += *(img.data + (img.width * y + offset) + x);
10                 neighborsSum += *(img.data + (img.width * y - offset) + x);
11                 neighborsSum += *(img.data + (img.width * y + offset) + x - offset);
12                 neighborsSum += *(img.data + (img.width * y - offset) + x + offset);
13                 neighborsSum += *(img.data + (img.width * y - offset) + x - offset);
14                 neighborsSum += *(img.data + (img.width * y + offset) + x + offset);
15
16                 newImg.data[y*img.width + x] = neighborsSum / 9;
17         }
18 }
```

Primjer koda 5: *Glavna petlja u implementaciji kutijastog filtera*

4.4. Grayscale filter

Grayscale slika je vrsta slike u kojoj su sve boje uklonjene, ostavljajući samo nijanse sive. Drugim riječima, jedina vrijednost koja se može iščitati iz određenog piksela jest njegova luminancija. Ova tehnika se često koristi u raznim područjima obrade slike zbog svoje jednostavnosti i smanjene količine podataka koje je potrebno obraditi i pohraniti. Neke od primjena uključuju prepoznavanje objekata, detekciju rubova, i analizu tekstura.

Grayscale filtar pretvara obojene slike (npr. one u RGB formatu) u slike koje sadrže samo nijanse sive. Ovaj proces uključuje izračunavanje jedne vrijednosti intenziteta za svaki piksel,

SL. 4.2: *Usporedba RGB i grayscale verzije iste digitalne slike [10]*

koja predstavlja njegovu svjetlinu. Svaka nijansa sive može se smatrati različitom razinom svjetline između crne (nula) i bijele (maksimalna vrijednost). Najčešća metoda za pretvaranje slike u grayscale koristi linearnu luminanciju, koja predstavlja percipiranu svjetlinu boje.[Linearna luminancija izračunava se kao težinska suma crvene, zelene i plave komponente piksela. Formulu za linearnu luminancu možemo prikazati na sljedeći način:

$$Y = (0.299 * R) + (0.587 * G) + (0.114 * B) \tag{4-6}$$

Gdje je R komponenta crvene, G komponenta zelene, a B komponenta plave boje ciljanog piksela.

Ove težine određene su zaključkom da postoji različita osjetljivost ljudskog oka na različite boje; ono je najosjetljivije na zelenu svjetlost, zatim crvenu, a najmanje na plavu.

Primjena grayscale filtra na digitalnu sliku uključuje iteraciju kroz sve piksele slike, izračunavanje nove vrijednosti svjetline za svaki piksel, i postavljanje te vrijednosti kao nove vrijednosti piksela.

```
1  for (int i=0; i<img.size; i+=img.channelNumber) {
2          int ch = img.channelNumber;
3          int grayLvl = 0;
4          while(ch != 0) grayLvl += *(img.data + i + (--ch));
5          grayLvl /= img.channelNumber;
6
7          ch = img.channelNumber;
8          while(ch != 0) *(img.data + i + (--ch)) = grayLvl;
9
10 }
```

Primjer koda 6: *Implementacije glavne petlje grayscale filtera*

U nekim slučajevima, kao što je detekcija rubova ili prepoznavanje uzoraka, grayscale slike mogu pružiti jasnije informacije. Zato je korištenje grayscale filtera često prvi korak u mnogim algoritmima obrade slike, što omogućava fokus na strukturalne karakteristike slike bez utjecaja boje.

4.5. Binarni filter

Binarni filter pretvara sliku u crno-bijelu, pri čemu svaki piksel može imati samo jednu od dvije moguće vrijednosti: crnu (0) ili bijelu (255). Ovaj proces je ključan u različitim područjima

računalne obrade slike, uključujući prepoznavanje obrazaca, OCR (optičko prepoznavanje znakova) i analizu oblika. Binarizacija slike često olakšava daljnju analizu i izdvajanje ciljanih značajki.

Binarni filter zahtjeva određivanje takozvanog praga. Ovaj dio procesa bi odradio posao postavljanja praga koji dijeli piksele slike u dvije kategorije: one ispod praga i one iznad praga. Piksle ispod praga obično se postavljaju na crnu (0), a piksle iznad praga na bijelu (255). Određivanja praga može biti jednostavno, gdje se koristi fiksni prag, ili adaptivno, gdje se prag određuje na temelju karakteristika slike. Operacija određivanja praga koristi se za promjenu ili identifikaciju vrijednosti piksela na temelju specificiranja jedne ili više vrijednosti (naziva se vrijednost praga).

Jedna od automatiziranih tehnika za određivanje optimalnog praga koji dijeli piksele slike u dvije kategorije (pozadinu i objekt) je Otsuova metoda, nazvana po japanskom znanstveniku imena Nobuyuki Otsu koji je predstavio ovu metodu određivanja praga u svojem radu 1979. godine [11]. Ova metoda je posebno korisna za segmentaciju slike kada je potrebno odvojiti različite dijelove slike temeljem razlika u intenzitetu sivih tonova.

Otsuova metoda traži optimalni prag tako da minimizira varijancu unutar klase, tj. varijabilnost vrijednosti između pozadine i objekta. Varijanca unutar klase je mjera za to koliko su pikseli unutar svake klase (pozadina i objekt) slični jedan drugome. Manja varijanca znači da su pikseli unutar iste klase međusobno sličniji. Otsuov algoritam traži prag koji minimizira varijancu unutar klase, definiranu kao težinski zbroj varijanci dviju klasa:

$$\sigma^2\omega(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{4-7}$$

Gdje su težine ω_0 i ω_1 vjerojatnosti dvaju klasa odvojenih pragom t, a ω_0^2 i ω_1^2 njihove varijance.

Klase su podijeljene prema histogramu koji se računa za određenu sliku. Histogram slike je osnovni alat za analizu distribucije intenziteta piksela unutar slike. Predstavlja učestalost svakog intenziteta piksela, od najtamnijeg do najsvjetlijeg. U slučaju sivih slika, histogram se sastoji od 256 stupaca, gdje svaki stupac predstavlja broj piksela sa specifičnim intenzitetom, od 0 (crna) do 255 (bijela). Za slike u boji, histogram se može računati za svaki kanal zasebno. Histogram možemo zamisliti kao koordinatni sustav gdje x os predstavlja moguće vrijednosti piksela, koje za 8-bitne grayscale slike variraju od 0 do 255, dok y os predstavlja broj piksela u slici koji imaju određenu vrijednost. Osim za odvajanje objekta od pozadine, također se koriste za normalizaciju slike kako bi se ravnomjernije rasporedile vrijednosti piksela

Vjerojatnoj klasa računa se iz L odjeljaka histograma:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \tag{4-8}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \tag{4-9}$$

Prema Otsuovoj metodi, optimalni prag je onaj koji minimizira varijancu unutar klase:

$$t^* = \arg\max_t \sigma_\omega^2(t) \tag{4-10}$$

Prilikom implementacije ove varijante binarnog filtera, nakon pretvaranja slike u njezin grayscale ekvivalent (u slučaju da to već nije), koristi se izračunati prag kako bi se odlučilo, za svaki piksel pojedinačno, hoće li piksel biti vrijednosti 0 ili 255.

int threshold = computeThreshold(img);

```
1 for (int i=0; i<img.size; i+=img.channelNumber) {
2         int greyLvl = *(img.data+i);
```

```
3           int finalPixel = greyLvl < threshold ? 0 : 255;
4
5           for (int j=i; j < i+img.channelNumber; j++)
6                   *(img.data+j) = finalPixel;
7   }
```

Primjer koda 7: *Binarizacija svakog piksela nakon izračunate vrijednosti praga*

Kako bi izračunali spomenuti prag, potreban nam je histogram slike koji se određuje pomoćnom metodom koja inicijalizira polje s 256 elemenata, kako bi pokrila sve moguće vrijednosti piksela, te prolazi kroz sve piksele slike. Petlja se inkrementira za broj kanala po iteraciji, budući da svaki piksel u slici može imati više kanala. Za svaki piksel, funkcija izračunava prosječnu vrijednost kanala, koja predstavlja grayscale razinu tog piksela, a tako izračunata vrijednost koristi se za popunjavanje histograma. Drugim riječima, brojanje piksela s određenom grayscale razinom povećava se svaki put kad se takav piksel pronađe u slici. Tako dobijemo potpuno popunjeni histogram u obliku polja cjelobrojnih vrijednosti koji predstavlja distribuciju grayscale razina piksela u slici.

```
1   int* getPixelHistogram (IMAGE img) {
2       static int histogram[256];
3       const int maxPixelIntensity = 255;
4       for (int i=0; i<=maxPixelIntensity; i++)
5           histogram[i] = 0;
6
7       for (int i=0; i<img.size; i+=img.channelNumber) {
8           int channelAvg = 0;
9           int ch = img.channelNumber;
10          while(ch != 0) channelAvg += *(img.data + i + (--ch));
11          channelAvg /= img.channelNumber;
12
13          histogram[channelAvg]++;
14      }
15
16      return histogram;
17  }
```

Primjer koda 8: *Implemetacija metode za stvaranje histograma iz ulazne slike*

Tako dobivamo sve potrebne predispozicije za implementaciju algoritma za određivanje praga. Inicijalnim prolaskom kroz histogram, izračunavamo ukupnu učestalost svih piksela, a popratnim prolaskom kroz histogram, izračunavamo učestalosti piksela u prvoj i drugoj klasi, kao i sume vjerojatnosti za obje klase. Cijelo vrijeme pratimo maksimalnu među-klasnu varijancu i po potrebi ažuriramo vrijednost praga koji će na kraju algoritma sadržavati optimalnu vrijednost

```
1   int computeThreshold(IMAGE img) {
2       int* histogram = getPixelHistogram(img);
3       int threshold = 0;
4       const int maxPixelIntensity = 255;
5       float globalPixelOccurance=0, c1PixelOccurance=0, c2PixelOccurance=0,
            ↪ c1ProbSum=0, c2ProbSum=0, varMax=0;
6
7       for (int i=0; i<=maxPixelIntensity; i++)
```

```
8           globalPixelOccurance += (i * histogram[i]);
9
10      for (int i=0 ; i<=maxPixelIntensity; i++) {
11          c1ProbSum += histogram[i];
12          c2ProbSum = (img.width * img.height) - c1ProbSum;
13
14          if (c1ProbSum == 0 || c2ProbSum == 0) continue;
15
16          c1PixelOccurance += i * histogram[i];
17          c2PixelOccurance = globalPixelOccurance - c1PixelOccurance;
18          float c1Mean = c1PixelOccurance / c1ProbSum;
19          float c2Mean = c2PixelOccurance / c2ProbSum;
20
21          float betweenClassVariance = c1ProbSum * c2ProbSum * pow((c1Mean -
                ↪ c2Mean), 2);
22          if (betweenClassVariance > varMax) {
23              threshold = i;
24              varMax = betweenClassVariance;
25          }
26      }
27      return threshold;
28  }
```

Primjer koda 9: *Implemetacija metode za izračunavanje praga Otsuovom metodom*

4.6. Okretanje slike

Okretanje slike je često korištena operacija u obradi slike koja omogućava promjenu orijentacije slike za 90 stupnjeva udesno ili ulijevo, ili pak refleksiju slike horizontalno ili vertikalno. Ove operacije ne mijenjaju vrijednosti intenziteta piksela, već samo njihovu poziciju u matrici digitalne slike. Drugim riječima, korištenjem ovih operacija, ulazna slika I transformira se u novu sliku I' mijenjanjem koordinata svakog piksela sa slike.

$$I(x, y) \rightarrow I'(x', y') \tag{4-11}$$

Horizontalno okretanje slike mijenja redoslijed piksela tako da lijeva strana postaje desna i obratno. Ovo se postiže zamjenom piksela u svakom retku slike. Za svaki redak, pikseli na lijevoj strani zamjenjuju se s odgovarajućim pikselima na desnoj strani. Tu operaciju možemo jednostavno prikazati kao:

$$I'(x, y) = I(W - x, y) \tag{4-12}$$

Gdje je W širina slike u pikselima.

Sa strane implementacije u programskom jeziku C, efekt ove operacije postiže se prolaskom kroz sve retke učitane slike te zamijenom svih piksela na lijevoj strani retka s njima odgovarajućim pikselima s desne strane retka. Također moramo pripaziti na kanale piksela te ih sve premjestiti u redoslijedu u kojemu se nalaze u izvornoj slici.

```
1  unsigned right = img.width;
2  for (int y=0; y<img.height; y++) {
3          while (left < right) {
```

```
4                   int px1Idx = (y * img.width + left) * img.channelNumber;
5                   int px2Idx = (y * img.width + right) * img.channelNumber;
6
7                   int ch = 0;
8                   while (ch != img.channelNumber) {
9                           uint8_t temp = img.data[px1Idx + ch];
10                          img.data[px1Idx + ch] = img.data[px2Idx + ch];
11                          img.data[px2Idx + ch] = temp;
12                          ch++;
13                  }
14                  left++;
15                  right--;
16          }
17          left = 0;
18          right = img.width;
19  }
```

Primjer koda 10: *Petlja za horizontalno okretanje slike*

Analogno tome, vertikalno okretanje slike mijenja redoslijed piksela tako da gornja strana postaje donja i obratno. Ovo se postiže zamjenom piksela između gornjih i donjih redaka slike.

$$I'(x, y) = I(x, W - y) \tag{4-13}$$

Ovdje se, sa strane implementacije, za svaki redak piksela koristi dvostruka petlja koja zamjenjuje piksele između gornjih i donjih redaka. Indeksi za gornje i donje redove se inicijaliziraju, a zatim se pikseli zamjenjuju dok se ne dođe do same sredine matrice koja predstavlja sliku. Prilikom ove implementacije, nemoramo paziti na pravilno premještanje kanala piksela jer su oni postavljeni istim redoslijedom kojim prolazimo kroz matricu.

```
1  unsigned top = 0;
2      unsigned chWidth = img.width * img.channelNumber;
3      unsigned bottom = (img.height - 1) * chWidth;
4      while (top < bottom) {
5
6          for (int x=0; x<chWidth; x++) {
7              int px1Idx = top + x;
8              int px2Idx = bottom + x;
9
10             uint8_t temp = img.data[px1Idx];
11             img.data[px1Idx] = img.data[px2Idx];
12             img.data[px2Idx] = temp;
13         }
14         top += chWidth;
15         bottom -= chWidth;
16
17     }
```

Primjer koda 11: *Petlja za vertikalno okretanje slike*

Rotacija slike je geometrijska transformacija koja premješta poziciju svakog piksela slike prema zadanom kutu rotacije oko odabranog središta. Ova operacija se često koristi za po-

boljšanje vizualnog izgleda slike ili kao preprocesorski korak u složenijim primjenama obrade slike kao što su prepoznavanje uzoraka, detekcija rubova i morfološke operacije.

Transponiranje slike je operacija koja mijenja redove slike u stupce i obrnuto. Ako je slika predstavljena kao matrica $I(x, y)$, transponirana matrica $I^T(x, y)$ dobiva se zamjenom indeksa redaka i stupaca. Ovo je osnovni korak u mnogim rotacijskim transformacijama slike

Rotacija slike izvodi transformaciju koju možemo opisati sljedećim formulama:

$$x' = x - x_0 \cos\theta - y - y_0 \sin\theta + x_0 \tag{4-14}$$

$$y' = x - x_0 \sin\theta - y - y_0 \cos\theta + y_0 \tag{4-15}$$

gdje su x i y koordinate piksela na ulaznoj slici, x' i y' koordinate piksela u izlaznoj slici, x_0 i y_0 koordinate središta rotacije, a θ kut rotacije. Pozitivne vrijednosti kutova θ definiraju rotaciju u smjeru kazaljke na satu.

U praksi, rotacija slike može rezultirati pozicijama piksela koje ne odgovaraju cijelim brojevima. Da bi se ovaj problem riješio i da bi se odredila vrijednost intenziteta piksela na svakoj cjelobrojnoj poziciji piksela u izlaznoj slici, koriste se različite tehnike ponovnog uzorkovanja [12]. No u slučaju rotacije slike za 90 stupnjeva, ovaj problem nije prisutan.

Prilikom implementacije rotacije slike u lijevo, nakon učitavanja slike, jedino što se mora učiniti je pozvati pomoćnu proceduru za transponiranje matrice slike, koja će premjestiti piksele iz izvorne slike na koordinate gdje će se nalaziti u rezultirajućoj slici. Prilikom transponiranja, indeksi su podešeni tako da uzmu u obzir širinu i visinu slike te broj kanala.

```
1  void transposeImage(IMAGE* originalImg, IMAGE* newImg) {
2      for (int y=0; y<originalImg->height * originalImg->channelNumber; y+=
           ↪ originalImg->channelNumber) {
3          for (int x=0; x<originalImg->width * originalImg->channelNumber; x+=
               ↪ originalImg->channelNumber) {
4              int ch = originalImg->channelNumber;
5              while (ch != 0) {
6                  newImg->data[x*originalImg->height + y + ch] = originalImg->
                       ↪ data[y*originalImg->width + x + ch];
7                  ch--;
8              }
9          }
10     }
11 }
```

Primjer koda 12: *Procedura za transponiranje matrice vrijednosti intenziteta slike*

Rotacija slike u desno, osim pozivanja procedure za transponiranje matrice slike, mora dodatno okrenuti piksele prema vertikalnoj osi kako bi se dobila rotacija udesno za 90 stupnjeva.

```
1  for (int y=0; y<newImg.height*newImg.channelNumber; y+=newImg.channelNumber)
       ↪ {
2          int leftIdx = newImg.channelNumber - 1;
3          int rightIdx = newImg.width * newImg.channelNumber - 1;
4
5          while (leftIdx < rightIdx) {
6
7                  int ch = newImg.channelNumber;
8                  uint8_t temp;
```

```
9                   while (ch != 0) {
10                      ch--;
11                      temp = newImg.data[y*newImg.width + leftIdx - ch];
12                      newImg.data[y*newImg.width + leftIdx - ch] = newImg.data[y
                            ↪ *newImg.width + rightIdx - ch];
13                      newImg.data[y*newImg.width + rightIdx - ch] = temp;
14                  }
15
16                  leftIdx += newImg.channelNumber;
17                  rightIdx -= newImg.channelNumber;
18              }
19  }
```

Primjer koda 13: *Petlja za okretanje piksela prema vertikalnoj osi*

4.7. Razvrstavanje piksela

Razvrstavanje piksela (engl. pixel sort) je tehnika obrade slika koja uključuje razvrstavanje piksela unutar slike (ili određenih dijelova slike) prema određenom kriteriju, kao što je intenzitet svjetlosti ili vrijednost boje. Ova tehnika često se koristi za stvaranje umjetničkih i apstraktnih efekata na slici.

Neke od često korištenih kriterija za razvrstavanje piksela su:

- Razvrstavanje piksela na temelju njihove razine svjetline, što rezultira slikama s jasnim trakastim područjima svijetlih i tamnih područja.
- Razvrstavanje piksela na temelju njihovih vrijednosti boja, što stvara slike sa živim i nadrealnim uzorcima boja.
- Razvrstavanje piksela na temelju njihove razine zasićenosti, stvarajući slike intenzivnih i preuveličanih intenziteta boja.
- Nasumično sortiranje piksela za stvaranje kaotičnih i nepredvidivih vizualnih efekata [13].

U ovom radu implementacija razvrstavanja piksela provest će se koristeći quicksort algoritam, prilagođen soritranju piksela sa slike. Quicksort je učinkoviti algoritam za sortiranje opće namjene, objavljen od strane britanskog računalnog znanstvenika Tony Hoarea 1962. godine. [14]. Ovaj algoritam koristi princip podjele i osvajanja (engl. divide and conquer). Odabire takozvani "pivot" element iz niza i dijeljenjem preostalih elemenata u dva dodatna niza, ovisno o tome jesu li manji ili veći od istog.

U ovom radu, kod sortiranja piksela primjenom quicksort algoritma, algoritam nakon učitavanja slike sortira piksele horizontalno na temelju prosječne vrijednosti intenziteta boje svakog piksela. Ako je slika prevelika, provodi se smanjenje veličine slike kako bi se izbjeglo preopterećenje memorije i prelazak granica stoga za pozive (engl. call stack).

Procedura sa slike 3.15 prima niz piksela, lijevi indeks, desni indeks i broj kanala slike. Ako je lijevi indeks veći ili jednak desnom indeksu, funkcija se vraća jer je segment već sortiran. Ovo znači da ako u ovu proceduru za vrijednost lijevog indeksa proslijedimo vrijednost 0, a za vrijednost desnog indeksa proslijedimo vrijednost zadnjeg indeksa u matrici ulazne slike, slika će biti u potpunosti sortirana. U svakoj iteraciji glavne petlje izračunava se prosječna vrijednost intenziteta boje za trenutni piksel i takozvani pivot piksel koji se nalazi na kraju segmenta.

Ako je prosječna vrijednost trenutnog piksela manja od pivot prosječne vrijednosti, pikseli se zamjenjuju pomoću privremene varijable. Na taj način, pikseli s nižom prosječnom vrijednosti intenziteta boje premještaju se na lijevu stranu segmenta, dok se pikseli s višom prosječnom vrijednosti premještaju na desnu stranu. Nakon što su svi pikseli u segmentu pregledani, pivot piksel se premješta na odgovarajuću poziciju tako da svi pikseli s nižom prosječnom vrijednosti budu s lijeve strane, a svi pikseli s višom vrijednosti s desne strane. Funkcija se zatim rekurzivno poziva za lijevi i desni podsegment, dijeleći segment u dva dodatna dijela na temelju novog indeksa pivota.

```
1  void quickSortPixels(uint8_t* arr, int l, int r, int chN) {
2      if (l >= r) return;
3      int i=l-chN, j=0, chTemp;
4      for (j=l; j<r; j+=chN) {
5          float currChAvg=0, pivotChAvg=0;
6          chTemp = chN;
7          while (chTemp != 0) {
8              chTemp--;
9              currChAvg += arr[j+chTemp];
10             pivotChAvg += arr[r-chTemp];
11         }
12         currChAvg /= chN; pivotChAvg /= chN;
13
14         if (currChAvg < pivotChAvg) {
15             i+=chN;
16
17             chTemp = chN;
18             while (chTemp != 0) {
19                 chTemp--;
20                 int temp = arr[i+chTemp];
21                 arr[i+chTemp] = arr[j+chTemp];
22                 arr[j+chTemp] = temp;
23             }
24         }
25     }
26     int pIdx = i+chN;
27
28     chTemp = chN;
29     while (chTemp != 0) {
30         chTemp--;
31         int temp = arr[pIdx+chTemp];
32         arr[pIdx+chTemp] = arr[r+chTemp];
33         arr[r+chTemp] = temp;
34     }
35
36     quickSortPixels(arr, l, pIdx-chN, chN);
37     quickSortPixels(arr, pIdx+chN, r, chN);
38 }
```

Primjer koda 14: *Implementacija quick sort algoritma za razvrstavanje piksela sa slike*

Kao što je slučaj i u quicksort implementaciji opisanoj u Hoarevom radu, prosječnu vre-

mensku složenost procedure za sortiranje piksela na slici možemo izraziti kao $O(n \log n)$, gdje je n broj piksela u slici. To je zato što se niz piksela u prosjeku podijeli na pola u svakoj rekurzivnoj iteraciji, a svaka podjela zahtijeva linearno vrijeme za usporedbu i razmjenu elemenata. U najgorem slučaju, složenost može doseći $O(n^2)$, no u praksi možemo računati na prosječnu složenost ovog algoritma.

5. IZRADA APLIKACIJE ZA OBRADU DIGITALNE SLIKE

Aplikacija za obradu digitalne slike koristi arhitekturu klijent-poslužitelj. Poslužiteljski dio zadužen je za pozadinsku i poslovnu logiku potrebnu za obavljanje različitih operacija obrade slike, dok je klijentski dio odgovoran za prikaz korisničkog sučelja i interakciju s korisnikom.

Poslužiteljski dio aplikacije razvijen je u programskom jeziku JavaScript [15] koristeći Node.js [16] okruženje, te biblioteku za manipuiranje slikama napisanu u programskom jeziku C. Poslužiteljski dio implementiran je kao REST API (engl. Representational State Transfer Application Programming Interface), što omogućava jednostavnu i efikasnu komunikaciju između klijentskog i poslužiteljskog dijela aplikacije. API pruža funkcionalnosti za unos i obradu digitalne slike, uz opcije za obradu slike podržane od strane pozadinske biblioteke. Ovaj sloj je također zadužen za pohranu privremenih slika i različitih konfiguracijskih datoteka.

Klijentski dio aplikacije izgrađen je koristeći Vue.js [17], popularnu biblioteku za izradu korisničkih sučelja. Korištenje biblioteke za izradu korisničkih sučelja omogućava stvaranje dinamičnih i responzivnih sučelja koja korisnicima pružaju intuitivno i ugodno iskustvo. Struktura klijentskog dijela organizirana je u nekoliko direktorija, uključujući direktorij za komunikaciju s poslužiteljem, direktorij za statičke datoteke i direktorij za različite Vue.js komponente koje čine sučelje aplikacije.

Strukturu aplikacije možemo prikazati na sljedeći način:

```
[root]
├── client
│   ├── apiRepository
│   ├── public
│   └── src
│       └── components
└── server
    ├── config
    ├── modules
    │   └── image
    └── temp
        ├── include
        └── src
            ├── dimension_tools
            └── filters
```

5.1. Poslužiteljski dio aplikacije

Prilikom pokretanja poslužiteljskog okruženja, odvija se konfiguracija baze podataka koja osigurava pouzdanu vezu s PostgreSQL [18] bazom podataka. Ova baza podataka pohranjuje informacije o procesima obrade slika i dostupnim opcijama obrade. Prilikom pokretanja apli-

kacije, provjerava se postojanje potrebnih tablica i unose se osnovni podaci ako tablice ne sadrže informacije. Ovaj inicijalizacijski postupak osigurava da aplikacija uvijek ima potrebne elemente za pohranu podataka, čime se izbjegavaju greške prilikom izvršavanja operacija.

```
module.exports.createMissingTables = async () => {
        await db.query(‘
    CREATE TABLE IF NOT EXISTS public.image_process (
        process_id SERIAL PRIMARY KEY,
        image_name VARCHAR(255)
    );

    CREATE TABLE IF NOT EXISTS public.image_processing_option (
        name VARCHAR(32) PRIMARY KEY,
        parameters VARCHAR(32)[],
        possible_values VARCHAR(32)[]
    );
‘);
};


\begin{lstlisting}[language=C, caption={Programatsko popunjavanje tablica zadanim
    ↪ podacima}, captionpos=b]
module.exports.createDefaultTableData = async () => {
        await db.query(‘
        INSERT INTO public.image_processing_option (name, parameters,
            ↪ possible_values)
        VALUES
        (’binary’, ARRAY[’’], ARRAY[’’]),
        (’grayscale’, ARRAY[’’], ARRAY[’’]),
        (’blur’, ARRAY[’blur-radius’], ARRAY[’number’]),
        (’flip’, ARRAY[’plane-direction’], ARRAY[’horizontal’, ’vertical’]),
        (’rotate’, ARRAY[’direction’], ARRAY[’left’, ’right’]),
        (’pixel-sort’, ARRAY[’’], ARRAY[’’])
        ON CONFLICT DO NOTHING
    ‘);
};
```

Primjer koda 15: *Programatsko stvaranje tablica neophodnih za rad aplikacije*

Sustav za upravljanje prijenosom slika koristi middleware treće strane imena multer za prijenos datoteka na poslužiteljsku stranu. Slike koje korisnici prenose pohranjuju se s jedinstvenim nazivima spajajući originalni naziv datoteke i trenutne vremenske oznake kako bi se izbjeglo prepisivanje postojećih datoteka. Osim toga, middleware omogućava filtriranje datoteka prema tipu, osiguravajući takvu konfiguraciju da se isključivo slikovne datoteke mogu prenijeti na poslužiteljsku stranu.

```
const multer = require(’multer’);
const imageUpload = multer({
        limits: { fileSize: ’2MB’ },
        fileFilter: (req, file, cb) => {
                if (file.mimetype.includes(’image/’)) {
                        cb(null, true);
```

```
7                   } else {
8                           cb(null, false);
9                           return cb(new Error('Only images are a valid upload.'));
10                  }
11          },
12          storage: multer.diskStorage({
13                  destination: async (req, file, cb) => {
14                          const dirPath = './';
15
16                          fs.mkdir(dirPath, {recursive: true}, (err) => {
17                                  if (err) {
18                                          cb(err, dirPath);
19                                  }
20                          });
21
22                          cb(null, dirPath);
23                  },
24                  filename: (req, file, cb) => {
25                          let ext = '';
26                          if (file.originalname.split('.').length > 1){
27                                  ext = file.originalname.substring(file.originalname.
                                        ↪ lastIndexOf('.'));
28                          }
29                          cb(null, `${file.originalname.replace(/ /g, '').substring(0, file.
                                ↪ originalname.lastIndexOf('.'))}-${Date.now()}${ext}`);
30                  }
31          })
32
33  });
```

Primjer koda 16: *Middleware konfiguracija za prijenos slika na poslužiteljsku stranu*

Prilikom prijenosa slike, aplikacija kreira novi zapis u bazi podataka koji identificira proces obrade te pohranjuje naziv prenesene datoteke. Ovaj korak je važan jer omogućava praćenje stanja i rezultata svakog pojedinog procesa obrade slike te svojevrsnu paralelizaciju procesiranja korisničkih interakcija s poslužiteljem. Nakon što je slika uspješno prenesena i zabilježena u bazi podataka, korisniku se vraća jedinstveni identifikator procesa obrade. Korisnik je u mogućnosti odabrati različite opcije obrade slike od kojih su sve opisane u prethodnim poglavljima. Svaka od ovih opcija ima specifične parametre koji se mogu podesiti. Korisnici putem API-ja šalju zahtjeve s odabranim opcijama obrade i vrijednostima parametara. Aplikacija zatim izvršava odgovarajuće naredbe za obradu slike, koristeći pomoćne alate i biblioteke koje su definirane u aplikaciji.

```
1  module.exports.submitImageProcessingOptions = async (req, res) => {
2          const { processId, processingOption, processingValue} = req.body;
3          const img = await imageServices.submitImageProcessingOptions(processId,
               ↪ processingOption, processingValue);
4          res.writeHead(200, { 'Content-Type': 'image/jpeg' }).end(img);
5  };
```

Primjer koda 17: *Metoda za interakciju s klijentom za odabir opcije obrade slike*

Tijekom procesa obrade, slika se preimenuje i premješta u privremeni direktorij, gdje se izvršava stvarna obrada. Obrada slike se pokreće putem naredbi koje izvršava sustav te poziva metodu iz C biblioteke preko komandnog sučelja. Rezultirajuća, obrađena slika pohranjuje se u privremeni direktorij. Ako obrada slike uspije, korisniku se vraća obrađena slika, omogućujući mu preuzimanje i korištenje obrađene verzije.

```
1  module.exports.submitImageProcessingOptions = async (processId, processingOption,
       ↪ processingValue) => {
2          const { image_name } = await repository.getProcess(processId);
3          const tempDir = `${process.env.PWD}/temp`;
4          const command = `make run OG_FNAME="${image_name}" PROC_OPT="$
               ↪ {processingOption}" PROC_VAL="${processingValue}"`
5
6          try {
7                  await exec(command, {cwd: 'temp'});
8                  return await fs.readFile(`${tempDir}/processed-` + image_name);
9          } catch (err) {
10                 console.error('Image processing error:', err);
11         }
12 };
```

Primjer koda 18: *Proces pozivanja odgovarajuće metode za obradu slike prema zahtjevima korisnika*

Cijeli proces od prijenosa slike do njenog preuzimanja nakon obrade osmišljen je tako da bude siguran, brz i pouzdan. Poslužiteljski dio aplikacije osigurava da svi korisnički zahtjevi budu ispravno obrađeni, a rezultati pohranjeni i vraćeni korisnicima bez mogućnosti manipulacije podacima od strane korisnika. Ova arhitektura omogućava fleksibilnost i proširivost aplikacije, čime se osigurava podrška za buduće funkcionalnosti i operacije obrade slike.

5.2. Klijentski dio aplikacije

Struktura klijentskog dijela organizirana je tako da je što preglednija, modularnija i jednostavnija za održavanje i korištenje. Sadrži isključivo osnovne funkcionalnosti za komunikaciju s poslužiteljskim dijelom aplikacije te prikaz unešene i rezultantne digitalne slike prije i nakon obrade.

Komponenta zaslužna za spajanje klijentskog i poslužiteljskog sloja aplikacije koristi axios, popularnu biblioteku za HTTP zahtjeve. Ona sadrži dvije osnovne funkcije - jednu za prijenos slike na poslužitelj i jednu za slanje odabranih opcija obrade slike. Prva koristi objekt za slanje slikovnih datoteka, dok potonja funkcija omogućava korisnicima da šalju odabrane opcije obrade zajedno s odgovarajućim parametrima.

```
1  const imageRepository = {
2    uploadImage: (image) => {
3      const formData = new FormData();
4      formData.append("file", image, image.name);
5      return axios.post("http://localhost:4000/image", formData);
6    },
7
8    submitImageProcessingOptions: (processId, option, value) => {
9      return axios.post(
```

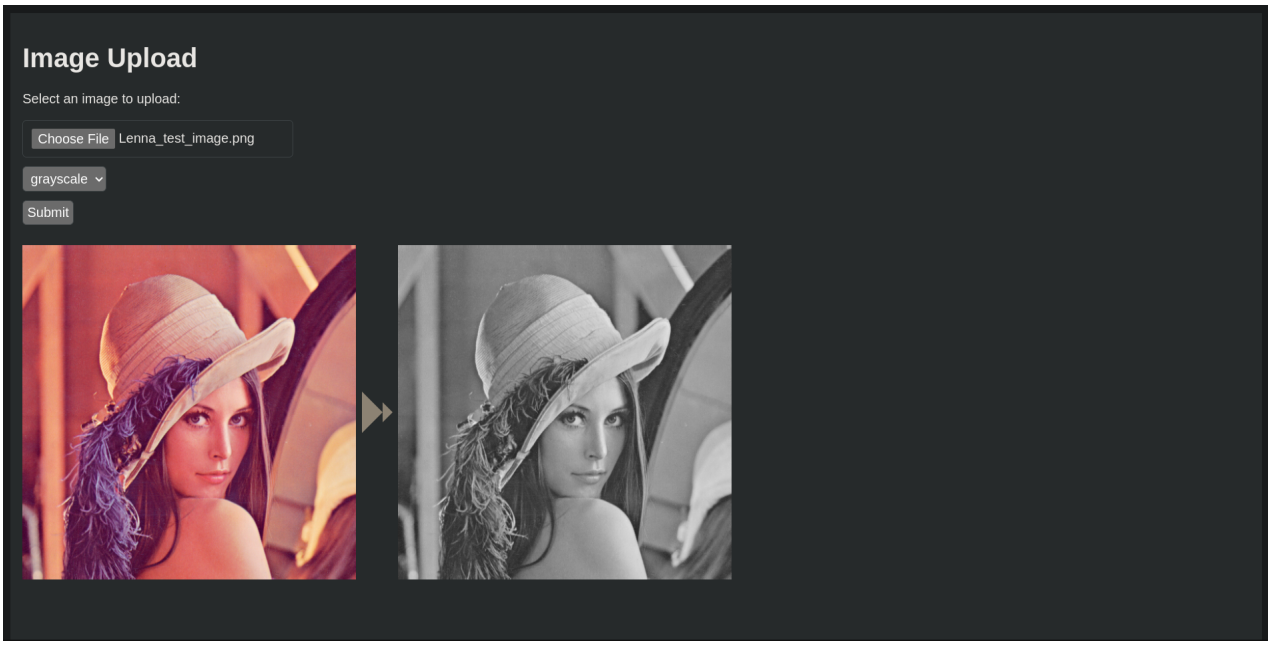
```
10          "http://localhost:4000/image/processingOptions",
11          {
12            processId,
13            processingOption: option,
14            processingValue: value,
15          },
16          { responseType: "blob" }
17        );
18    },
19  };
```

Primjer koda 19: *Objekt s metodama za slanje HTTP zahtjeva na poslužiteljski sloj*

Klijentski dio aplikacije sadrži samo jednu stranicu s vrlo jednostavnim korisničkim sučeljem. Nakon otvaranja web aplikacije, korisniku je ponuđena opcija prenošenja proizvoljne slike u sustav. Nakon odabira, prikazuje se padajući izbornik s opcijama obrade slike te gumb za potvrdu odabira iste. Ispod toga, nakon učitavanja, prikazuje se i sama odabrana slika.

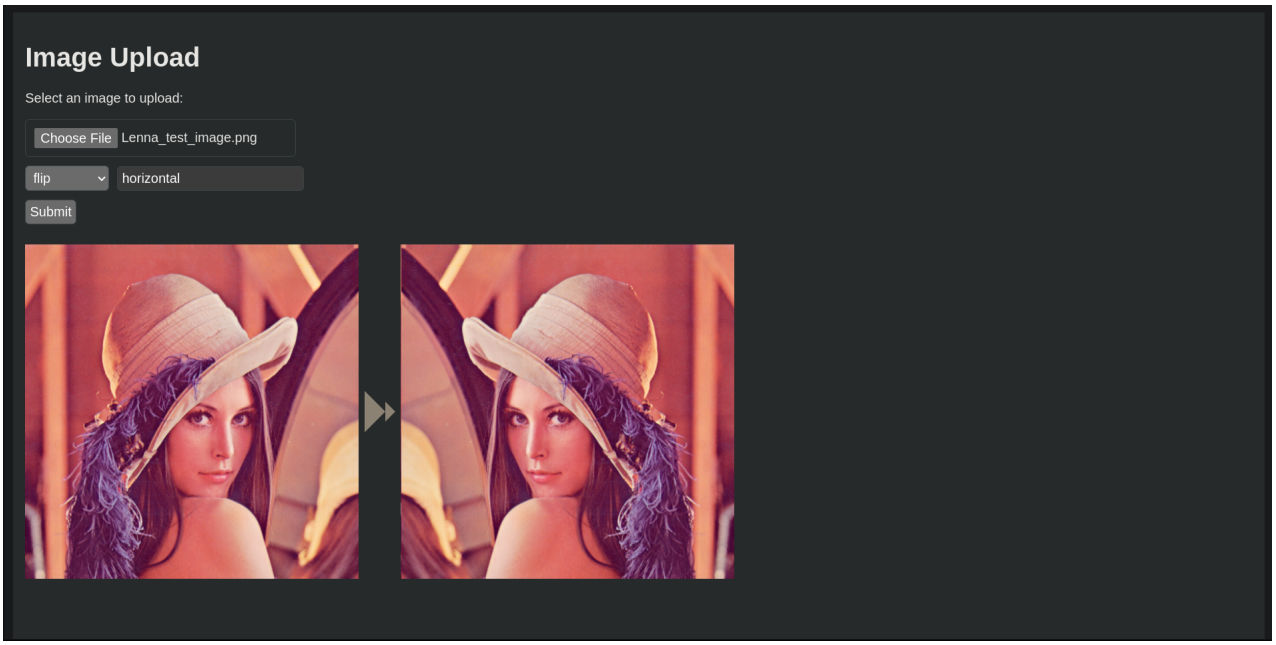
SL. 5.3: *Prikaz korisničkog sučelja nakon prijenosa proizvoljne slike*

Nakon što korisnik odabere metodu obrade slike iz padajućeg izbonika te pritisne gumb za potvrdu, slika se šalje na poslužiteljski sloj aplikacije te se od tamo obrađena slika vraća natrag na klijentski dio i prikazuje pored originalne slike.



SL. 5.4: *Prikaz korisničkog sučelja nakon potvrde oko odabira metode za obradu slike*

U slučaju da korisnik izabere neku od metoda za koju su potrebni dodatni parametri, s desne strane prethodno spomenutog padajućeg izbornika prikazat će se polje za unos tog parametra uz upute kakav parametar se očekuje. Korisnik upisuje parametar i potvrđuje unos pritiskom na prethodno spomenuti gumb. Jedina razlika u cijelokupnom toku procesa je ta što se sada taj unos validira prije slanja na poslužiteljsku stranu.



SL. 5.5: *Prikaz korisničkog sučelja nakon potvrde oko odabira metode za obradu slike uz unos dodatnog parametra*

Glavno korisničko sučelje za prijenos i obradu slika implementirano je u komponenti koja omogućava korisnicima odabir i prijenos slika putem web preglednika. Nakon što korisnik odabere sliku, ona se prikazuje kao pregled pomoću ugrađenih alata za čitanje datoteka. Komponenta također omogućava korisnicima odabir opcija obrade slike putem padajućeg izbornika

te unos parametara za odabrane opcije. Prilikom prijenosa slike, sučelje prvo ažurira pregled slike i potom šalje sliku na poslužitelj. Poslužitelj vraća jedinstveni identifikator procesa obrade i dostupne opcije obrade slike, koje se zatim prikazuju korisniku. Korisnici zatim mogu odabrati željene opcije obrade i unijeti odgovarajuće parametre.

```
async handleImageInput(input) {
        const image = input.target.files[0];
        this.updateImagePreview(image);

        try {
          const { data } = await $axios.imageRepository.uploadImage(image);
          this.processId = data.processId;
          console.log(data);
          this.imageProcessingOptions = data.imageProcessingOptions || [];
        } catch (error) {
          console.error(error);
          alert("Image uploading failed");
        }
  },
```

Primjer koda 20: *Metoda za slanje ulazne slike na poslužiteljski sloj*

Odabir opcija obrade slike omogućava korisnicima prilagodbu parametara obrade prema vlastitim potrebama. Nakon što su odabrane željene opcije, uneseni i validirani odgovarajući parametri, sučelje šalje te podatke na poslužitelj. Poslužitelj zatim izvršava odgovarajuće operacije obrade slike i vraća obrađenu sliku klijentu kao binarnu datoteku. Klijentski sloj će upravo nju prikazati korisniku.

```
async submitImageProcessingOptions() {
        const option = this.chosenImageProcessingOptionIdx
          ? this.imageProcessingOptions[this.chosenImageProcessingOptionIdx]
          : this.imageProcessingOptions[0];

        if (!this.validateInputtedParamValue(option)) {
          alert("Invalid inputted value");
          this.inputtedImageProcessingValue = "";
          return;
        }

        try {
          const res = await $axios.imageRepository.submitImageProcessingOptions(
                this.processId,
                option.name,
                this.inputtedImageProcessingValue
          );
          const url = URL.createObjectURL(res.data);
          this.processedImage = url;
        } catch (error) {
          console.error(error);
          alert("Sending image processing parameters failed");
        }
```

```
24      },
```

Primjer koda 21: *Metoda za slanje ulazne slike na poslužiteljski sloj*

Dodatna komponenta koristi se za prikaz prijelaza između originalne i obrađene slike. Ova komponenta omogućava vizualno uspoređivanje rezultata obrade, pružajući korisnicima jasan pregled promjena nastalih primjenom odabranih opcija obrade. Komponenta jednostavno prihvaća URL-ove originalne i obrađene slike kao ulazne podatke te ih prikazuje jednu pored druge s vizualnim pokazateljem prijelaza.

```
1  export default {
2    props: {
3      previewImage: {
4        type: String,
5        required: false,
6      },
7      processedImage: {
8        type: String,
9        required: false,
10     },
11   },
12 };
```

Primjer koda 22: *Stanje Vue komponente za dinamičko ažuriranje prikaza ulazne i izlazne slike*

6. ZAKLJUČAK

Web aplikacija razvijena u sklopu ovog završnog rada razvijena je s ciljem omogućenja jednostavnog i učinkovitog upravljanja obradom digitalnih slika putem intuitivnog korisničkog sučelja. S obzirom na rastuću potrebu za dostupnim i lako primjenjivim alatima za obradu slika, ova aplikacija pruža korisnicima priliku da na brz i jednostavan način učitaju svoje slike, primijene željene algoritme za obradu te preuzmu obrađene slike.

Jedna od ključnih prednosti ove aplikacije je njezina jednostavnost i pristupačnost. Korisničko sučelje je dizajnirano tako da minimalizira potrebu za tehničkim znanjem, omogućujući korisnicima svih razina iskustva da koriste aplikaciju bez poteškoća. Nakon prijenosa slike, korisnicima se pruža izbor između nekoliko popularnih algoritama za obradu slike, koji se mogu jednostavno primijeniti putem ponuđenog sučelja.

Poslužiteljski dio aplikacije, koji se temelji na učinkovitim i fleksibilnim algoritmima za obradu slika, nastoji održati brzi odaziv na korisničke zahtjeve. Korištenjem programskog jezika C za implementaciju ovih algoritama omogućava tu učnikovitost izvedbe programa, a kotištenjem popularnih alata za razvoj web servisa kao što su Node.js i Express, aplikacija pruža stabilnu i pouzdanu platformu za pozivanje tih procedura u stvarnom vremenu.

Iako je aplikacija u ovoj fazi razvoja fokusirana na osnovne funkcionalnosti, osigurane su mogućnosti za njezino daljnje proširenje. Moguće nadogradnje su dodavanje novih algoritama za obradu slike te novih opcija za prilagodbu parametara obrade ili pak integraciju s vanjskim servisima za pohranu i dijeljenje slika kao zamjena za preuzimanje rezultirajućih slika. U budućnosti, aplikacija bi mogla uključivati naprednije funkcionalnosti kao što su automatsko prepoznavanje objekata na slikama, mogućnost primjene umjetničkih filtera i efekata, ili integraciju s alatima za strojno učenje koji bi omogućili detaljniju personalizaciju obrađenih slika na temelju korisničkih preferencija.

LITERATURA

[1] "The best photo editing software," https://www.creativebloq.com/features/photo-editing-software, pristupljeno 14. lipanj 2024.

[2] "Adobe photoshop," https://www.adobe.com/products/photoshop/features.html, pristupljeno 14. lipanj 2024.

[3] "Gimp," https://www.gimp.org/, pristupljeno 14. lipanj 2024.

[4] "Canva," https://www.canva.com/hr_hr/, pristupljeno 14. lipanj 2024.

[5] "Magic studio," https://www.canva.com/magic/, pristupljeno 14. lipanj 2024.

[6] "Digital image interpolation," https://www.cambridgeincolour.com/tutorials/image-interpolation.htm, pristupljeno 30. svibanj 2024.

[7] "Edge detection," https://vincmazet.github.io/bip/detection/edges.html#f-detection-marr-hildreth-canny, pristupljeno 2. lipanj 2024.

[8] I. Sobel and G. Feldman, *A 3×3 isotropic gradient operator for image processing*, 1973, pp. 271–272.

[9] "Fast image convolutions," https://web.archive.org/web/20060718054020/http://www.acm.uiuc.edu/siggraph/workshops/wjarosz_convolution_2001.pdf, pristupljeno 3. lipanj 2024.

[10] S. Jindal and Pooja, "A review on multiscale texture features using steerable pyramids," *INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY*, vol. 15, pp. 7374–7378, 2016.

[11] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[12] "Edge detection," https://homepages.inf.ed.ac.uk/rbf/HIPR2/rotate.htm, pristupljeno 7. lipanj 2024.

[13] "Pixel sorting – definition, examples, history & more – digital art and technology glossary," https://web.archive.org/web/20060718054020/http://www.acm.uiuc.edu/siggraph/workshops/wjarosz_convolution_2001.pdf, pristupljeno 8. lipanj 2024.

[14] C. A. R. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[15] "Javascript," https://developer.mozilla.org/en-US/docs/Web/JavaScript, pristupljeno 9. lipanj 2024.

[16] "About node.js®," https://nodejs.org/en/about, pristupljeno 9. lipanj 2024.

[17] "The progressive javascript framework," https://vueframework.com/, pristupljeno 9. lipanj 2024.

[18] "What is postgresql?" https://www.postgresql.org/about/, pristupljeno 9. lipanj 2024.

7. SAŽETAK

Cilj ove web aplikacije je omogućiti korisnicima jednostavnu obradu digitalnih slika putem intuitivnog korisničkog sučelja. Aplikacija omogućuje korisnicima da prenesu sliku u jednom od podržanih formata te odaberu različite opcije obrade putem padajućeg izbornika. Nakon što korisnik potvrdi odabrane parametre, aplikacija šalje podatke na poslužitelj gdje se izvršava obrada slike koristeći specijalizirane algoritme. Rezultirajuća slika se prikazuje korisniku te je dostupna za preuzimanje ili daljnju obradu. Aplikacija je dizajnirana s naglaskom na jednostavnost i preglednost, omogućujući brz i učinkovit rad. Implementacija same biblioteke koja sadrži algoritme za obardu slike provedena je u programskom jeziku C koji pruža visoku učinkovitost izvedbe, a sama web aplikacija realizirana je u JavaScript ekosustavu koji omogućava web komunikaciju te nudi veliki raspon biblioteka što pospješuje jednostavnost implementacije.

Ključne riječi: algoritmi, digitalne slike, učinkovitost izvedbe, jednostavno korisničko sučelje, JavaScript

8. ABSTRACT

The aim of this web application is to provide users with simple digital image processing through an intuitive user interface. The application allows users to upload an image in one of the supported formats and select various processing options via a dropdown menu. Once the user confirms the selected parameters, the application sends the data to the server where image processing takes place using specialized algorithms. The resulting image is displayed to the user and is available for download or further processing. The application is designed with emphasis on simplicity and clarity, enabling fast and efficient execution. The implementation of the image processing algorithms is carried out in the C programming language, chosen for its high performance capabilities, while the web application itself is developed within the JavaScript ecosystem, facilitating web communication and offering a wide range of libraries to enhance ease of implementation.

Keywords: algorithms, digital images, performance efficiency, simple user interface, JavaScript