Elektronski fakultet  
Univerzitet u Nišu

Seminarski rad

**Sistem za pretragu slika po teksturnom sadržaju i distribuciji boja uz korišćenje *Redis* baze podataka za indeksiranje**

*Student:*

Petar Trifunović 16955

*Mentor:*

Prof. dr Suzana R. Stojković

Sadržaj

[Kratak osvrt na razvoj sistema za pretraživanje slika 1](#_Toc71663035)

[Pretraživači slika zasnovani na metapodacima 1](#_Toc71663036)

[Pretraživači slika zasnovani na vizuelnom sadržaju (*CBIR sistemi*) 1](#_Toc71663037)

[Pojašnjenje pojma *sadržaj* (eng. *content*) 2](#_Toc71663038)

[Opis problema i tema rada 3](#_Toc71663039)

[Distribucija boja 3](#_Toc71663040)

[Dodatna obrada histograma 3](#_Toc71663041)

[Teksturne karakteristike slika 4](#_Toc71663042)

[Matrice susedstva i pojam *GLCM* 4](#_Toc71663043)

[Haralikove teksturne karakteristike (eng. *Haralick’s texture features*) 5](#_Toc71663044)

[Baza podataka i indeksna struktura za rad sa deskriptorima slika 7](#_Toc71663045)

[Kratak osvrt na princip rada *Redis* baze podataka 8](#_Toc71663046)

[Generisanje jedinstvenog naziva za slike 8](#_Toc71663047)

[Čuvanje deskriptora slika 8](#_Toc71663048)

[Indeksna struktura 8](#_Toc71663049)

[Arhitektura sistema i rezultati rada 10](#_Toc71663050)

[Klasa za predstavljanje slike kao objekta 10](#_Toc71663051)

[Klasa za pristup bazi podataka 11](#_Toc71663052)

[Klasa za procesiranje slika 12](#_Toc71663053)

[Klase za sortiranje rezultata 12](#_Toc71663054)

[Klasa servisa za manipulaciju slikama 13](#_Toc71663055)

[Računanje distance između sličnih slika 13](#_Toc71663056)

[Konfiguracioni fajlovi 14](#_Toc71663057)

[Spisak korišćenih biblioteka 14](#_Toc71663058)

[Rezultati rada 14](#_Toc71663059)

[Literatura 16](#_Toc71663060)

# Kratak osvrt na razvoj sistema za pretraživanje slika

Sa napredovanjem i razvojem računarstva, izgled sadržaja koji je računarima pružan na obradu, kao i izgled rezultata obrade, menjao se i postajao kompleksniji. Prvobitno, sadržaj je bio pretežno numeričkog tipa. Sa pojavom personalnih računara i početkom upotrebe istih u svakodnevnom životu, prešlo se na sadržaj tekstualnog oblika, razumljiv širem krugu korisnika. Vremenom, a kao rezultat ekspanzije interneta i *World Wide Web*-a, tekst je obogaćivan vizuelnim i audio sadržajem, tako da su danas multimedijalni tipovi sadržaja neizbežni u interakciji čoveka i računara, a pogotovo vizuelni sadržaj, odnosno slike.

## Pretraživači slika zasnovani na metapodacima

Prvobitni sistemi za pretragu slika bili su mnogo sličniji sistemima projektovanim za pretraživanje teksta, nego onome što se danas podrazumeva pod pojmom pretrage slika. Njihov rad zasnivao se na posmatranju i skladištenju slike ne kao sadržaja vizuelnog tipa, već kao skupa tekstualnih metapodataka čija je svrha bila detaljniji opis slike. Primer ovakvog sistema razvijan je devedesetih godina na *MIT*-u, a njegov rad dokumentovan je nešto ranije, 1986. godine (Prasad, Gupta, Toong, & Madnick, 1986). Ovaj sistem je koristio princip pretrage koji je sadržao listu tekstualnih pojmova kojima su slike opisivane, i svaki pojam pokazivao je na listu identifikatora slika koje ga sadrže. Ovakav način rada bio je gotovo identičan radu sistema za pretragu tekstualnog sadržaja. Rezultati rada i performanse sistema bile su na visokom nivou, s obzirom na to da su sve optimizacije primenljive na pretraživačima tekstualnog sadržaja mogle da budu primenjene i ovde. Glavna mana ovog, kao i drugih rešenja zasnovanih na metapodacima, bila je činjenica da slike same za sebe ne nose nikakav tekstualni sadržaj iz kog bi se radom nekog parsera mogli izvući pojmovi kojima je slika opisana. Ovo je značilo da bi se tekstualni opis slika morao dodavati naknadno, ručno, bilo od strane korisnika sistema, pri dodavanju slika, bilo od strane eksperata čije bi zaduženje bilo razmatranje slika i pridruživanje metapodataka svakoj od njih.

## Pretraživači slika zasnovani na vizuelnom sadržaju (*CBIR sistemi*)

Napredak u oblasti računarskog vida uveo je mogućnost pribavljanja i analiziranja karakteristika vizuelnog sadržaja koje nisu vidljive ljudskom oku, kao što su razmatranje distribucije boja iz statističkog ugla, definisanje odnosa piksela u pogledu intenziteta, posmatranje slika iz ugla frekventnog domena, nalaženje karakterističnih tačaka na slikama i slično. Ovo su takozvane karakteristike *niskog nivoa*, čijom se daljom analizom, često korišćenjem tehnika mašinskog učenja, može doći do karakteristika *višeg nivoa* i time se isključivo računarskom obradom, bez direktnog učešća čoveka, mogu pribaviti značajne informacije smislene i golom oku, kao što su prisustvo ili odsustvo određene klase objekata na slici, ili informacije o teksturi određene površine.

Bez obzira na to da li se karakteristike niskog nivoa koriste direktno, ili se dižu na viši nivo, sama činjenica da ih je moguće pribaviti bez posredstva ljudskog faktora čini ove karakteristike pogodnim za kreiranje autonomnih sistema pretraživanja za koje je moguće kreirati pogodnu softversku i hardversku arhitekturu i programski definisati način obrade slika kojim se omogućuje pretraživanje i odgovaranje na korisnički upit. Sistemi zasnovani na korišćenju karakteristika slika pribavljenih tehnikama računarskog vida nazivaju se *CBIR sistemi* (eng. *Content Based Image Retrieval Systems*).

CBIR sistemi koriste karakteristike slika (boje, teksture, oblike, prostorne karakteristike). Pomoću njih, kreira se višedimenzioni deskriptivni vektor za svaku od slika, i ovi vektori se čuvaju u bazi podataka. Od korisnika se zahteva da unesu primer slike, ili ručno kreiranu skicu, i takav unos predstavlja korisnički upit. I sama ulazna slika pretvara se u vektor, po istom principu po kom su kreirani vektori slika koji se čuvaju u bazi podataka. Cilj CBIR sistema jeste da kao rezultat vrati slike koje po vizuelnim karakteristikama najviše odgovaraju upitu. Ovakve slike nalaze se upoređivanjem i računanjem distance između vektora upita i vektora iz baze, korišćenjem određene metrike. (Ovaj opis predstavlja slobodnu interpretaciju opisa *CBIR* sistema koji su dali (Long, Zhang, & Feng, 2003)).

## Pojašnjenje pojma *sadržaj* (eng. *content*)

Radi izbegavanja eventualne konfuzije, važno je bliže pojasniti pojam sadržaja, odnosno *content*, pri korišćenju termina *CBIR*.

Gledano iz ugla korisnika, i sistemi bazirani na metapodacima vrše pretragu po sadržaju. U ovom kontekstu, pojam *sadržaj* odnosi se na ono što je vidljivo golim okom, i ukoliko korisnik zadaje tekstualni upit u kome na primer navodi jedan ili više objekata, očekuje da kao rezultat dobije slike koje na kojima se upravo ovi objekti nalaze. Korisnici u ovakvim sistemima, ukoliko samo pretražuju, a ne dodaju slike, nisu svesni načina na koji se sadržaj slike definiše, te se sa apstraktnog nivoa oslanjaju na sam sistem i očekuju zadovoljavajući rezultat.

U skraćenici *CBIR*, pojam *content* (odnosno *sadržaj*) ne odnosi se na ono što je vidljivo korisniku sistema. U ovom kontekstu, pojam se odnosi na one karakteristike slike koje je moguće izvući direktno iz načina na koji je ona predstavljena u memoriji računara (nizom piksela), i za koje nisu potrebne dodatne anotacije i metapodaci za čije bi generisanje bilo neophodno ljudsko posredstvo.

# Opis problema i tema rada

Ovaj rad opisuje implementaciju CBIR sistema, pisanu u programskom jeziku *Python*. Implementirani sistem kao ulaz očekuje primer slike, a kao rezultat vraća slike najsličnije ulaznoj. Za upoređivanje i određivanje sličnosti između slika koriste se distribucija boja i teksturne karakteristike. Radi ubrzanja rada sistema, koristi se *Redis* baza podataka u svrhu kreiranja strukture nalik na invertovani indeks.

Ideja rada sistema u pogledu kombinacije boja i teksturnih karakteristika slika predložena je u radovima (Sharma, 2011) i (Islam, Ahmed, & Kundu, 2014). Prvopomenuti predlaže korišćenje histograma boja za prvobitnu pretragu, a nakon toga dodatno filtriranje rezultata korišćenjem teksturnih karakteristika zasnovanih na takozvanim *wavelet* transformacijama (ove transformacije nisu korišćenje u implementaciji sistema, pa o njima neće biti dodatnih detalja u daljem tekstu). Drugonavedeni rad ne bavi se *wavelet* transformacijama, već distribuciju boja kombinuje isključivo sa teksturnim karakteristikama pribavljenim korišćenjem takozvane *matrice susedstva*, konkretno *GLCM* (*Gray Level Co-Occurrence Matrix*).

## Distribucija boja

Za posmatranje karakteristika slika u pogledu distribucije boja sistem koristi *histograme boja*. Grafički, histogram se predstavlja pomoću 2D grafa. Po x osi nalaze se vrednosti u koje se pikseli mogu svrstati, dok y osa predstavlja broj piksela koji uzimaju konkretnu vrednost. Vrednosti po x osi u engleskoj literaturi nazivaju se *bin*-ovima. Bin-ova može biti onoliko koliko mogućih vrednosti piksel može uzeti; na primer, ukoliko se vrednost piksela kreće od 0 do 255, za svaku od 256 vrednosti može postojati po jedan bin. Bin-ova može biti i manje, pa tako, za primer gde se vrednosti piksela kreću od 0 do 255, a histogram ima 8 bin-ova, pikseli koji uzimaju neku od prve 32 vrednosti pripale bi prvom binu, naredne 32 vrednosti drugom i tako dalje.

Programski, za predstavljanje histograma dovoljan je samo jedan niz **H**, gde bi vrednost **H[i]** označavala broj piksela koji pripadaju i-tom bin-u.

Implementacija sistema za predstavljanje boja koristi standardni *RGB* (*Red, Green, Blue*) format, pa se vrednost svakog piksela predstavlja pomoću tri osmobitne vrednosti. Moguće je kreirati jedan histogram kojim bi se predstavila distribucija piksela po svakom od tri kanal (R, G i B), ali je, zarad nezavisnog posmatranja karakteristika koje se odnose na svaku od boja, korišćen princip kreiranja po tri histograma za svaku sliku, odnosno po jedan histogram po kanalu. Svaki histogram sadrži maksimalan moguć broj bin-ova, odnosno 256.

### Dodatna obrada histograma

Iako postoje metode za upoređivanje histograma, koje su uz to dostupne u vidu gotovih implementacija kroz *OpenCV* biblioteku u *Python*-u, samo upoređivanje histograma slika usporava rad sistema, pogotovo ukoliko sistem radi sa velikom bazom slika. Naime, puko upoređivanje histograma ulaznih slika sa histogramima slika iz baze bi dalo preciznu informaciju o sličnosti distribucije boja između para slika, ali bi bilo neefikasno ukoliko bi se zahtevalo upoređivanje ulazne slike sa čitavim skupom slika iz baze. Zbog toga, potrebno je uvesti metod indeksiranja kojim bi se količina slika sa kojom treba uporediti ulaznu suzila na određen podskup iz baze. Histogram, predstavljen nizom na način opisan u prethodnom pasusu, ne bi bio pogodan za kreiranje indeksne strukture. Dva razloga čine ovakvu reprezentaciju histograma nepogodnom – prvo, histogram jedne slike je isuviše specifičan da bi se očekivalo da više slika ima identičan histogram, što znači da bi u indeksnoj strukturi pod jednim histogramom najčešće bila čuvana samo jedna slika; drugo, metode upoređivanja histograma daju meru sličnosti, ali ne nose nikakvu relacionu informaciju o histogramima, u smislu da je jedan histogram *veći* ili *manji* od drugog, što dovodi do nemogućnosti sortiranja histograma, a indeksne strukture se uveliko oslanjaju na sortiranje indeksiranih podataka radi brže pretrage.

Ovaj problem dovodi do potrebe predstavljanja histograma pomoću manjeg broja karakterističnih vrednosti. Implementacija sistema koristi činjenicu da histogram u osnovi daje statističke informacije o bojama u slici, pa se, kako je predloženo u (Roy & Mukherjee, 2013), računaju srednja vrednost i standardna devijacija histograma svakog od tri kanala kojima se predstavlja boja piksela. Ove vrednosti se računaju za normalizovani histogram, odnosno histogram kod koga je svaka vrednost podeljena sa brojem piksela slike za koju se pravi histogram. Na osnovu ovih vrednosti, kreira se vektor oblika:

***[Bmean, Bstd, Gmean, Gstd, Rmean, Rstd]****,*

gde ***Xmean*** predstavlja srednju vrednost histograma odgovarajućeg kanala, a ***Xstd***vrednost standardne devijacije. Redosled kanala u vektoru je *BGR* (umesto standardnog RGB) zbog činjenice da *OpenCV* biblioteka u *Python*-u kanale posmatra u ovakvom redosledu.

Vrednosti kojima je predstavljen ovaj vektor se dalje koriste za indeksiranje i kreiranje indeksne strukture korišćenjem *Redis* baze podataka, o čemu je više detalja dato u daljem tekstu.

## Teksturne karakteristike slika

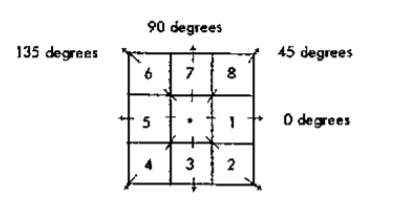
Analiza teksturnih karakteristika implementirana je korišćenjem matrice susedstva, konkretno *GLCM*.

### Matrice susedstva i pojam *GLCM*

Matrica susedstva (eng. *co-occurrence matrix*) je matrica kojom je moguće prikazati raspored piksela različitih intenziteta unutar slike. Ukoliko pri kreiranju ove matrice boja piksela nije od interesa, odnosno ukoliko se posmatra reprezentacija slike pomoću samo jednog kanala, dobija se matrica susedstva u pogledu nijansi sive boje, to jest *GLCM* (eng. *Gray Level Co-occurrence Matrix*). GLCM jeste najčešći tip matrice susedstva korišćen kada je u pitanju obrada slika (matrice susedstva ne moraju nužno da se odnose na slike), tako da se najčešće u literaturi ni ne nailazi izolovano na pojam *matrice susedstva* (eng. *co-occurrence matrix*), već se podrazumevano priča odnosi na GLCM. Ukoliko se, na primer, pri kreiranju matrice posmatraju sva tri standardna kanala (R, G, B), radi se o *CCM* (eng. *Color Co-occurrence Matrix*).

Kako navodi (Hall-Beyer, 2007), „*GLCM jeste tabelarni prikaz toga koliko često se različite kombinacije jačine piksela (u pogledu nijansi sive boje) javljaju na slici.“* Takođe, navodi se i da je GLCM matrica *drugog reda* (eng. *second order*), s obzirom na to da posmatra odnose između **grupa od po dva piksela**.

GLCM matrica ima podjednak broj vrsta i kolona, i to onoliko koliko različitih vrednosti pikseli mogu uzeti. Vrednost polja sa indeksima **i, j** označava koliko puta se piksel sa vrednošću na koju se odnosi kolona **j** javlja kao sused pikselu sa vrednošću na koju se odnosi vrsta **i**. Dva važna parametra ove matrice jesu smer u kom se posmatra susedstvo, kao i rastojanje između piksela koji se smatraju susedima. Što se rastojanja tiče, često se posmatra rastojanje 1, što znači da se susedima smatraju pikseli koji se nalaze direktno jedan uz drugog, ali rastojanje može biti i veće. Što se smera tiče, moguće je posmatrati susedstvo po horizontali, vertikali, po jednoj, ili drugoj dijagonali. Smer susedstva definisan je u stepenima, kao što je prikazano na slici:



Definisanje smera susedstva u stepenima. Slika je preuzeta iz (Haralick, Shanmugam, & Dinstein, 1973)

Matrica se kreira tako da bude simetrična, pa posmatranje ugla od 0 stepeni podrazumeva i posmatranje ugla od 180 stepeni, posmatranje ugla od 45 stepeni podrazumeva i 225 stepeni i td. Zbog toga, kao što se može videti na ilustraciji, postoji samo 4 moguća ugla posmatranja susedstva.

Opisani parametri susedstva definisani su u (Haralick, Shanmugam, & Dinstein, 1973).

Trenutna implementacija sistema posmatra susedstvo sa rastojanjem 1 i uglom od 0 stepeni (odnosno horizontalno susedtstvo), ali bi buduća modifikacija ovih parametara bila jednostavna, s obzirom na to da se njihove vrednosti nalaze samo u jednom konfiguracionom fajlu, odakle ih čita ostatak sistema.

### Haralikove teksturne karakteristike (eng. *Haralick’s texture features*)

GLCM sama za sebe ne može biti iskorišćena na odgovarajući način za analizu teksturnih karakteristika, već se zahteva njena dalja obrada zarad izvlačenja dodatnih informacija. (Haralick, Shanmugam, & Dinstein, 1973) navode u svom radu 14 teksturnih karakteristika koje je moguće izračunati na osnovu GLCM.

(Hall-Beyer, 2007) navodi da je najbolje, kada je to moguće, koristiti kombinaciju 3 karakteristike – kontrast, korelaciju i entropiju. U radu (Islam, Ahmed, & Kundu, 2014) za analizu tekstura predložene su 4 karakteristike – energija, kontrast, entropija i inverzna razlika (eng. *Inverse Difference*).

*Python* biblioteke korišćene u implementaciji ne pružaju gotove funkcije za sve karakteristike predložene ovim radovima. Zbog toga, karakteristike koje su korišćene jesu energija, korelacija i homogenost, što (Hall-Beyer, 2007) naziva još i *normalizovanim momentom inverzne razlike* (eng. *Inverse Difference Moment Normalized*).

Svih 14 karakteristika koje definišu (Haralick, Shanmugam, & Dinstein, 1973) vrše računicu nad normalizovanim oblikom GLCM, odnosno oblikom gde je zbir svih vrednosti u matrici jednak jedinici. Normalizacija se vrši deljenjem svake vrednosti nenormalizovane matrice sa zbirom svih takvih vrednosti:

,

gde je *Pi, j* normalizovana vrednost polja na koordinatama **i, j**, a *Vi, j* nenormalizovana vrednost na istim koordinatama.

Formule karakteristika korišćenih u implementaciji, uz detaljnije objašnjenje njihovih značenja, su:

1. Energija:

Kod energije, vrednost na koordinatama **i, j** u GLCM predstavlja težinu sama za sebe, odnosno što je veća vrednost na nekim koordinatama u matrici, to će ta vrednost više da doprinese konačnom rezultatu. Visok nivo energije imaće slike koje vizuelno odaju utisak velike *urednosti* (eng. *orderliness)*, odnosno slike kod kojih nema prevelikih i previše čestih variranja u intenzitetu piksela. Konkretnije, što je više sličnih piksela češće će se javljati iste kombinacije susednih piksela i vrednosti u matrici će biti veće, pa će samim tim i energija rasti s obzirom na to da suma po kojoj se ona računa raste proporcionalno kvadratu vrednosti matrice.

1. Korelacija:

Korelacija koristi još dve specifične vrednosti, srednju vrednost (µ) i standardnu devijaciju (σ). Korelacija kazuje da između dva piksela postoji predvidvljiva linearna veza. To znači da je vrednost susednog piksela moguće predstaviti linearnim izrazom zavisnim od referentnog piksela (piksela čiji se sused računa). Što se veća vrednost dobije računanjem korelacije po gorenavadenoj formuli, to je veća verovatnoća da će vrednost dobijena za susedni piksel računanjem pomoću izraza zavisnog od referentnog piksela biti tačna. Dakle, velika vrednost korelacije ukazuje na dobru mogućnost pretpostavke vrednosti susednog piksela, ukoliko se zna vrednost referentnog piksela. Tako, korelacija između piksela koji pripadaju nekoj jedinstvenoj površini na slici (nebo, trava...) biće veća od korelacije između piksela koji pripadaju različitim objektima na slici.

1. Homogenost:

Homogenost se može razumeti kao veličina suprotna kontrastu. Kod homogenosti, akcenat je na sličnosti intenziteta piksela, tako da se vrednost iz normalizovane matrice (*Pi, j*) množi težinom obrnuto proporcionalnom rastu razlike između koordinata vrste i kolone (**i** i **j**). Tako, vrednosti daleko od glavne dijagonale, koje predstavljaju odnos piksela sa velikim razlikama u intenzitetu, će biti množene malim težinama, i težine će se eksponencijalno smanjivati sa udaljavanjem od glavne dijagonale, a vrednosti na glavnoj dijagonali će imati težinu 1.

Objašnjenja ovih karakteristika su slobodna interpretacija objašnjenja datih u (Hall-Beyer, 2007). Formule su takođe odatle preuzete.

U implementaciji sistema, vrednosti dobijene računanjem ovih karakteristika koriste se da prošire vektor kreiran na osnovu distribucije boja, kako je opisano u odeljku *Distribucija boja*. Na već postojeći šestodimenzioni vektor, dodaju se tri vrednosti dobijene računanjem kontrasta, korelacije i homogenosti, čime se dobija vektor od 9 članova koji se kasnije koristi pri računanju distance između ulazne slike i slika iz baze.

## Baza podataka i indeksna struktura za rad sa deskriptorima slika

Kreiranje indeksne strukture od velikog je značaja za brzinu rada sistema za pretraživanje informacija, kako tekstualnih tako i multimedijalnih.

Što se tiče slika, nakon kreiranja vektora kao deskriptora svake slike, princip po kome bi se vektor, nastao od ulazne slike, upoređivao sekvencijalno sa vektorima slika iz baze podataka oduzimao bi previše vremena i memorijskih resursa, s obzirom na to da bi se zahtevalo upoređivanje sa svakim vektorom iz baze i pamćenje u svakom trenutku određenog broja najboljih rezultata, kako bi se moglo utvrditi koje slike su najsličnije sa ulaznom. Zbog toga, cilj indeksne strukture jeste da suzi skup vektora sa kojima treba vršiti poređenje.

U implementaciji sistema, kao baza podataka za čuvanje vektora (deskriptora) slika koristi se *Redis* baza. Pomoću struktura dostupnih u *Redis*-u implementirana je i logika indeksiranja.

### Kratak osvrt na princip rada *Redis* baze podataka

*Redis* jeste takozvana *key-value* baza podataka. Ovo znači da je cela baza organizovana kao velika heš struktura, i podaci su čuvani pod jedinstvenim ključevima. Ključevi su *binary-safe*, što znači da string kojim se ključ predstavlja ne poznaje nikakve specijalne znakove, odnosno ključ može biti predstavljen bilo kojom sekvencom karaktera. Pod *Redis* ključem mogu se čuvati različite strukture podataka, a o pojedinim će biti dato više detalja u nastavku kada će biti reči o konkretnim primenama.

### Generisanje jedinstvenog naziva za slike

Implementirani sistem radi tako što se pri dodavanju novih slika navede putanja do direktorijuma koji ih sadrži. Nakon toga, između ostalog, slike se kopiraju u poseban direktorijum koji predstavlja bazu slika po kojima se pretraživanje vrši. Pri dodavanju novih slika iz više različitih direktorijuma, može se desiti da nekoliko slika nosi isti naziv. Zato se sistem stara da u direktorijumu koji predstavlja bazu ne dođe do konflikata sa imenima, tako što se u *Redis* *string* strukturi, pod ključem *naming.counter* pamti vrednost jedinstvenog brojača. Pri dodavanju svake slike, formira se novo ime slike tako što se na vrednost brojača nadoveže ekstenzija preuzeta iz originalnog imena slike, brojač se inkrementira i slika se sa novim imenom smešta u bazu (operacija inkrementiranja u *Redis*-u moguća je nad *string* strukturom ukoliko se vrednost može parsovati u numeričku; inkrementiranje u *Redis*-u vraća novu vrednost brojača, tako da se i pribavljanje i inkrementiranje brojača može izvršiti jednom atomičnom operacijom).

### Čuvanje deskriptora slika

Za pamćenje vektora kojima se opisuju slike takođe je korišćena *string* struktura dostupna u *Redis*-u. Vektor konkretne slike pamti se pod ključem kreiranim po šablonu: *vector:****jedinstveno\_ime\_slike***. Pod ovakvim ključem pamti se vektor slike tako što se elementi vektora redom konkateniraju jedan na drugi, razdvojeni blanko znakom.

### Indeksna struktura

Indeksna struktura sistema kreirana je na osnovu ideje invertovanog indeksa, korišćenog u sistemima za pretragu tekstualnog sadržaja. Početna ideja jeste da se pod određenim „terminom“ čuva lista slika koje taj termin sadrže. Ovakav princip savršeno odgovara tekstualnim pretraživačima, s obzirom na to da se tekstualni termin javlja u podskupu korpusa dokumenata. U pretraživačima slika, *terminima* se mogu smatrati specifične vrednosti dobijene na osnovu analize vizuelnih odlika slika. Za primer sistema kojim se bavi ovaj rad, kao termini posmatrane su vrednosti dobijene analizom distribucije boja. Ipak, posmatrati svaku dobijenu vrednost kao poseban termin nije najpogodnije, jer se radi o brojevima koji nisu celi i variraju od slike do slike, pa bi se pod jednom specifičnom vrednošću gotovo uvek čuvala lista sa samo jednom slikom (na primer, izuzetno su male šanse da dve slike imaju identičnu srednju vrednost histograma crvene boje). Kreiranje termina odsecanjem decimala od vrednosti takođe ne bi imalo smisla, jer, ukoliko svedemo sve vrednosti na jednu decimalu, pod terminom *crvena\_srednja\_vrednost\_0.2* bile bi sve slike čija srednja vrednost histograma crvena boje pripada opsegu **[0.2, 0.3)**, ali vrednost 0.2 je podjednako daleko od 0.21 koliko i od 0.19, a slike sa vrednošću 0.19 ne bi bile uključene u razmatranje jer je vrednost van navedenog opsega.

Radi izbegavanja opisanih problema, korišćen je sledeći princip: neka svaki termin pripada svakoj od slika, ali neka se **vrednost** termina ralikuje od slike do slike. Tako, u implementaciji sistema korišćeno je šest različitih termina – tri termina oblika ***X:mean***, koji se odnose na srednju vrednost histograma, i tri termina oblika ***X:std.deviation***, koji se odnose na standardnu devijaciju histograma. U oba oblika, ***X*** uzima vrednost ***R, G,*** ili ***B***, tako da se dobije po jedan termin od oba oblika za svaki od tri kanala preko kojih se posmatraju boje u slici.

Svaki termin predstavlja ključ u *Redis* bazi. Pod svakim ovakvim ključem zapamćena je struktura tipa *sortiranog skupa* (eng. *sorted set*) dostupna u *Redis*-u. Ova struktura omogućava uređeno pamćenje podataka. U ovoj strukturi se ne mogu naći duplikati (ne može se dodati jedan isti element više od jednom), a uz svaki podatak pamti se i skor (eng. *score*) koji predstavlja vrednost po kome se podaci sortiraju. U implementaciji sistema, kao skor se koriste „diskretizovani“ oblici srednjih vrednosti i vrednosti standardnih devijacija pribavljenih iz histograma. Pre diskretizacije, ove vrednosti su u decimalnom obliku, jer se računaju na osnovu normalizovanog histograma. Diskretizacija se vrši (u trenutnoj implementaciji) množenjem sa hiljadu, i zatim celobrojnim deljenjem sa sto, čime se dobija efekat zaokruživanja na dve decimale i izdvajanja te dve decimale kao celog broja. Ovime se postiže skraćivanje ključa koji se pamti u *Redis* bazi i povećanje njegove čitljivosti.

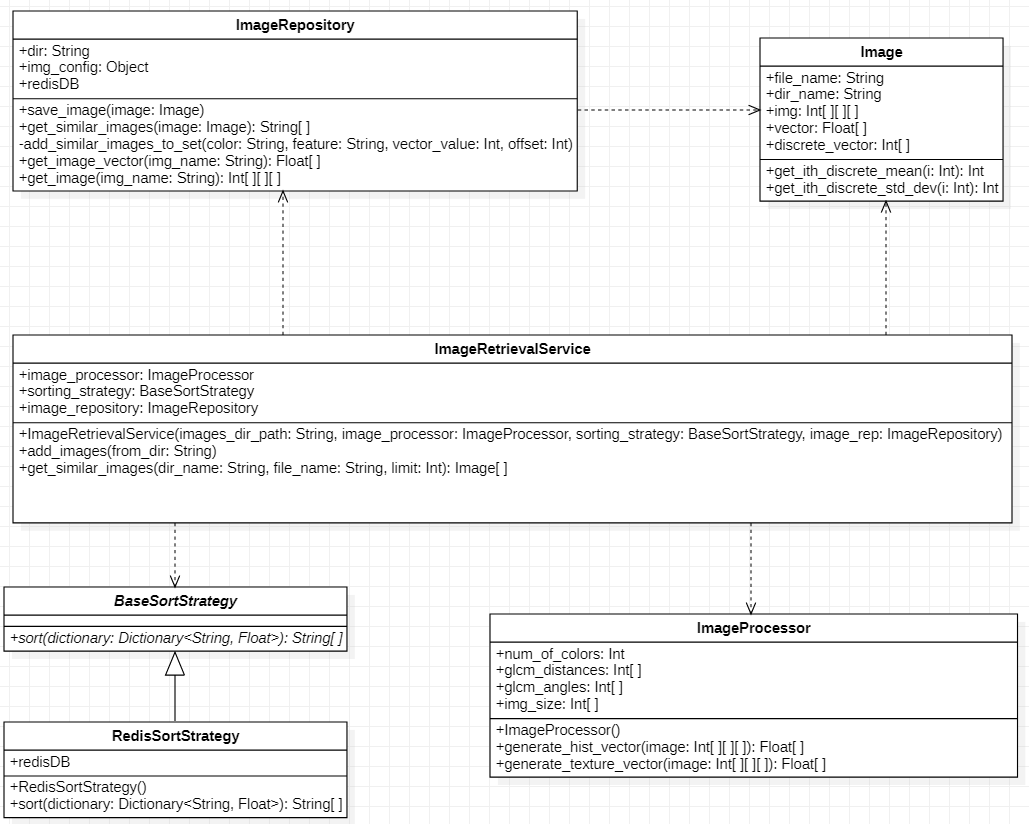
Opisana indeksna struktura omogućava pribavljanje slika koje su po srednjim vrednostima i standardnoj devijaciji histograma najbliže slici koja predstavlja upit. Ovakvo pribavljanje može se, implementaciono gledano, relativno lako obaviti korišćenjem operacija definisanih nad *Redis* strukturom sortiranog skupa. Na osnovu vektora (deskriptora) slike iz upita, računaju se skorovi za tu sliku. Zatim se za svaki od pomenutih šest „termina“, odnosno za njihove sortirane skupove, nalazi iz baze po ***N*** slika koje su po skoru ispred i iza slike iz upita, gde je ***N*** konstanta (ukoliko je slika iz upita previše blizu „vrhu“ ili „dnu“ sortiranog skupa, broj pribavljenih sličnih slika biće manji od 2 \* ***N***).

Pošto se za svaki od šest „termina“ pribavi određen broj slika, kao rezultat pribavljanja vraća se unija podskupova pribavljenih iz svakog od šest skupova, kako ne bi bilo dupliranja vraćenih rezultata. Ovo se postiže tako što se, nakon pribavljanja iz strukture sortiranog skupa, slike smeštaju u strukturu običnog skupa (ne sortiranog, eng. *set*) dostupnu u *Redis*-u. I ova struktura odlikuje se time što ne dozvoljava pojavu duplikata, tako da se praktično dobija efekat kreiranja unije.

**Napomena**: U poslednja dva pasusa, svuda gde je pomenut pojam *slika* kao nešto što se pribavlja iz ili smešta u odgovarajuće *Redis* sturkture, zapravo se govorilo o identifikatoru slike, odnosno njenom imenu koje je oblika opisanog u odeljku *Generisanje jedinstvenog naziva za slike*.

# Arhitektura sistema i rezultati rada

Arhitektura sistema organizovana je u nekoliko klasa, korišćenjem podrške koju jezik *Python* pruža za objektno orijentisanu paradigmu. Organizacija klasa prikazana je na sledećoj slici:



## Klasa za predstavljanje slike kao objekta

Klasa *Image* je pomoćna klasa koja kao atribute sadrži sve informacije od važnosti za jednu sliku u sistemu:

* *file\_name* i *dir\_name* – Redom, ovo su ime fajla koji predstavlja sliku, i ime direktorijuma u kojem je ona smeštena.
* *img* – Trodimenzionalno polje *integer* vrednosti kojim se u *Python­*-u i *OpenCV* biblioteci predstavljaju pikseli slike. Prvi i drugi indeks se odnose na poziciju piksela u matrici piksela, a treći indeks se odnosi na tri kanala boja (R, G, B).
* *vector* i *discrete\_vector* – Redom, ovo su vektor deskriptor slike, i njegova diskretizovana verzija.

Metode dostupne u ovoj klasi služe kao *getter* funkcije kojima se pribavljaju i-ta po redu srednja vrednost, ili vrednost standardne devijacije, iz vektora deskriptora slike.

## Klasa za pristup bazi podataka

Klasa za pristup bazi podataka implementirana je po ugledu na *Repository* obrazac, često korišćen u *ASP.NET* *Core* framework-u pri izradi Web API-ja (Spasojević, 2020). Kako *Python* nije objektno orijentisani jezik, ovaj obrazac nije implementiran u potpunosti, ali je iskorišćen važan princip – funkcije koje vrše direktan pristup bazi podataka radi manipulacije konkretnim entitetom grupisane su u posebnu, takozvanu *Repository* klasu, koja unutar sebe enkapsulira informacije o tome kojoj bazi i na koji način pristupa. U ovom sistemu, jedini entitet kojim se manipuliše jeste slika, i jedina *Repository* klasa jeste klasa *ImageRepository*. Ova klasa pristupa dvema bazama podataka, u zavisnosti od potrebe – *Redis* bazi, kao i bazi slika za koju se koristi *file system*.

Metode klase *ImageRepository* su:

* *save\_image* – Metoda koja izvršava sve neophodne akcije pri dodavanju nove slike u sistem. Konkretno, u ovoj metodi se u direktorijum koji predstavlja bazu slika (čija se putanja čuva kao atribut klase) smešta slika, a u *Redis* bazu se dodaje vektor deskriptor slike. Takođe, slika se, na način opisan u odeljku *Indeksna struktura*, dodaje u *Redis* indeksnu strukturu. Sve informacije o slici neophodne za obavljanje ovih akcija izvlače se iz objekta klase *Image* koji funkcije prima kao parametar.
* *get\_similar\_images* – Metoda koja pribavlja određen broj (hardkodiran unutar metode) slika iz baze koje su u indeksnoj strukturi „najbliže“ slici koja se prosleđuje kao parametar funkcije. Logika pribavljanja najbližih slika detaljnijo je objašnjena u odeljku *Indeksna struktura*.
* *add\_similar\_images\_to\_set* – Pomoćna metoda (zamišljena kao privatna) koja za konkretnu boju i konkretnu specifičnu karakteristiku histograma (srednju vrednost ili standardnu devijaciju) pribavlja određen broj (prosleđen metodi kao parametar) slika koje su u indeksnoj strukturi najbliže konkretnoj vrednosti vektora deskriptora (prosleđenoj funkciji kao parametar). Poziva se iz funkcije *get\_similar\_images* za sve kombinacije boja i karakteristika histograma.
* *get\_image\_vector* – Pribavlja iz *Redis* baze vrednost vektora za konkretan identifikator slike (prosleđen kao parametar metode), i vraća taj vektor kao povratnu vrednost.
* *get\_image* – Za konkretan identifikator slike (prosleđen kao parametar metode) vraća trodimenzionalan vektor kojim se slika predstavlja u *Python*-u i *OpenCV* biblioteci. Ova povratna vrednost je u obliku koji je moguće vizuelno prikazati na ekranu.

Viši moduli koji koriste ovu klasu nemaju nikakvo znanje o načinu na koji ona prstupa bazi podataka, niti kakav je tip baze kojoj se pristupa, što je prednost korišćenja *Repository* obrasca.

## Klasa za procesiranje slika

Klasa *ImageProcessor* zadužena je za procesiranje slika i sadrži sledeće metode:

* *generate\_hist\_vector* – Metoda zadužena za generisanje dela vektora deskriptora slike (prosleđene kao parametar metode) koji se odnosi na informacije o distribuciji boja, odnosno na informacije pribavljene iz histograma. U ovoj funkciji kreira se histogram za svaki od tri kanala boja (R, G, B), normalizuje se, a zatim se računaju srednja vrednost i standardna devijacija i smeštaju u vektor, koji predstavlja povratnu vrednost funkcije.
* *generate\_texture\_vector* – Metoda zadužena za generisanje dela vektora deskriptora slike (prosleđene metodi kao parametar) koji se odnosi na teksturne karakteristike. U ovoj funkciji računa se matrica susedstva (GLCM), iz nje se izvlače vrednosti tri karakteristike – energije, korelacije i homogenosti, i smeštaju u vektor koji predstavlja povratnu vrednost funkcije.
* *resize\_image* – Metoda koja menja veličinu slike (prosleđene metodi kao parametar) na veličinu definisanu u konfiguracionom fajlu. Koristi se kako bi se sve slike, i one koje se dodaju u sistem, i one na osnovu kojih se vrši pretraga, prebacile na istu veličinu, radi preciznijih rezultata.

## Klase za sortiranje rezultata

Klase za sortiranje rezultata organizovane su po *Strategy* projektnom obrascu, pa im se modul koji ih koristi (odnosno klasa *ImageRetrievalService*) obraća preko apstraktne klase *BaseSortStretegy*, čija je apstraktnost simulirana time što njena *sort* metoda ne sadrži implementaciju, već samo *pass* ključnu reč, koja u *Python*-u označava nepostojanje operacije (*NOP* instrukcija). Metoda *sort* prima parametar koji određuje smer sortiranja, i heš strukturu, koja pod *string* ključevima čuva vrednosti po kojima treba izvršiti sortiranje elemenata. Ključevi takođe predstavljaju elemente koje treba sortirati, tako da je povratna vrednost niz stringova sortiranih po vrednosti koja im je u heš strukturi bila pridružena. Konkretne klase izvedene iz ove apstraktne klase implementiraju *sort* metodu na odgovarajuć način.

U trenutnoj implementaciji sistema, jedina konkretna klasa za sortiranje jeste klasa *RedisSortStrategy*. Metoda *sort* ove klase koristi *Redis* strukturu sortiranog skupa zarad uređenja elemenata (sortirani skup je detaljnije objašnjen u odeljku *Indeksna struktura*). Ključevi heš strukture prosleđene metodi smeštaju se u sortirani skup, a kao skor dodeljuje im se vrednost koja im je pridružena u heš strukturi. Nakon smeštanja u skup, u zavisnosti od smera sortiranja, odlučuje se da li će se sortirane vrednosti pribaviti iz skupa *Redis* instrukcijom za pribavljanje u rastućem, ili u opadajućem smeru. Metoda vraća pribavljene sortirane vrednosti u vidu niza.

Klase za sortiranje nemaju nikakvo znanje o kontekstu podataka koje uređuju, i modul za sortiranje nikako ne zavisi od namene sistema. Heš struktura koju prima metoda za sortiranje ne daje nikakve naznake da sadrži slike koje treba urediti po sličnosti. O tome šta *sort* metoda prima pri korišćenju u sistemu koji je tema ovog rada više detalja je dato u odeljku *Klasa servisa za manipulaciju slikama*.

Sistem se može proširiti izvođenjem dodatnih klasa za sortiranje iz osnovne *BaseSortStrategy* klase. Po potrebi, klasa *ImageRetrievalService* mogla bi da dinamički u toku rada menja strategiju sortiranja koju koristi.

## Klasa servisa za manipulaciju slikama

Klasa *ImageRetrievalService* je „vršna“ klasa sistema. Servis kao atribute sadrži referencu na instancu klase *ImageProcessor*, na instancu neke od klasa izvedenih iz *BaseSortStrategy* (konkretno *RedisSortStrategy*), i na instancu *ImageRepository* klase.

Ova klasa sadrži sledeće metode:

* *add\_images* – Metoda koja kao parametar prima direktorijum iz koga se slike dodaju u sistem. Nakon učitavanja i promene veličine slike, generiše se vektor slike pomoću instance *ImageProcessor* klase. Pozivom *make\_vector\_discrete* funkcije iz pomoćnog *math\_helper* modula, kreira se diskretna verzija vektora. Zatim se sve važne informacije grupišu u objekat tipa *Image*, koji se šalje kao parametar *save\_image* metodi klase *ImageRepository*.
* *get\_similar\_images* – Metoda koja prima naziv direktorijuma i fajla slike kojoj treba naći slične slike iz baze sistema. Takođe, parametar metode je ***limit***, odnosno maksimalan broj sličnih slika koje treba vratiti. U metodi se od prosleđene slike, korišćenjem instance *ImageProcessor* klase, kreira deskriptor, kao i njegov diskretizovan oblik pomoću funkcije *make\_vector\_discrete* iz pomoćnog *math\_helper* modula. Zatim se metodom *get\_similar\_images* klase *ImageRepository* iz indeksne strukture pribave indentifikatori slika koje su ulaznoj slici dovoljno slične da bi bile uzete u razmatranje. Na osnovu indentifikatora pribavljaju se deskriptori svake slične slike pozivom *get\_image\_vector* metode *ImageRepository* klase. Pomoćnom funkcijom iz *math\_helper* modula računa se distanca između vektora ulazne slike i vektora svake od sličnih slika. Distanca se pod identifikatorom slike smešta u heš strukturu, koja se zatim prosleđuje *sort* metodi klase za sortiranje. Nakon sortiranja, uzima se prvih ***limit*** identifikatora sortiranog niza, za svaki se pribavi oblik slike pogodan za prikaz na ekranu metodom *get\_image* klase *ImageRepository*, i niz takvih slika vraća se kao rezultat poziva metode. Više detalja o računanju distance dato je u odeljku *Računanje distance između sličnih slika*.

Metode ove klase pozivaju se (u trenutnoj implementaciji) iz *main\_test.py* skripte.

## Računanje distance između sličnih slika

Računanje distance između slika svodi se na matematičku računicu, pa se funkcija koja ovo obavlja nalazi u *math\_helper* modulu. Kako predlažu (Roy & Mukherjee, 2013), kao dobra, iako ne najbolja, mera distance, korišćena je Menhetn metrika (eng. *Manhattan distance metric*). Metrika se svodi na korišćenje sledeće formule za računanje distance između N-todimenzionalnih vektora A i B:

## Konfiguracioni fajlovi

Sistem sadrži dva konfiguraciona fajla, preko kojih se mogu direktno podešavati važni parametri sistema.

Fajl *img\_config.json* sadrži informacije važne za obradu slika – spisak kanala preko kojih se posmatraju pikseli slika, veličina na koju se podešavaju sve slike pri obradi, distanca i ugao koji se koriste pri kreiranju GLCM.

Fajl *redis\_config.json* sadrži informaciju o *host*-u i *port­*-u koji se koriste pri povezivanju sa *Redis* bazom podataka.

## Spisak korišćenih biblioteka

|  |  |
| --- | --- |
| **Biblioteka** | **Verzija** |
| cycler | 0.10.0 |
| decorator | 4.4.2 |
| imageio | 2.9.0 |
| kiwisolver | 1.3.1 |
| numpy | 1.20.1 |
| opencv-contrib-python | 4.5.1.48 |
| opencv-python | 4.5.1.48 |
| Pillow | 8.1.2 |
| pip | 21.0.1 |
| pyparsing | 2.4.7 |
| python-dateutil | 2.8.1 |
| PyWavelets | 1.1.1 |
| redis | 3.5.3 |
| scikit-image | 0.18.1 |
| scipy | 1.6.2 |
| setuptools | 49.2.1 |
| six | 1.15.0 |
| tifffile | 2021.3.31 |

Podmoduli *greycomatrix* i *greycoprops* iz *feature* modula biblioteke *scikit-image* koriste se za kreiranje GLCM i računanje njenih karakteristika.

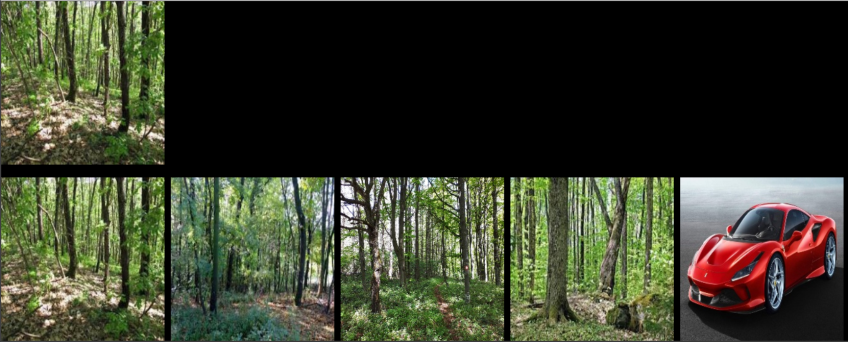
## Rezultati rada

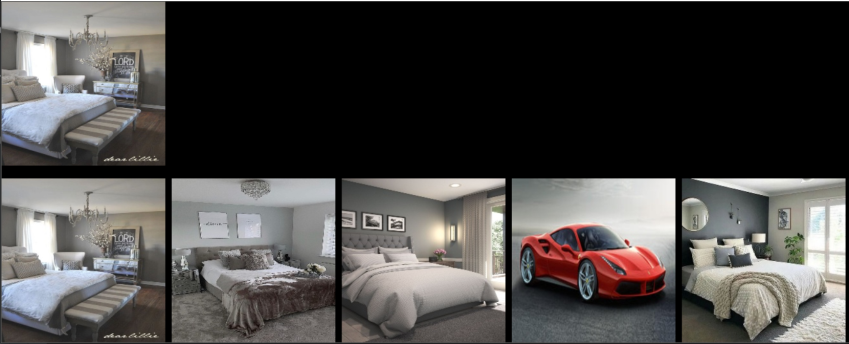
Slučajevi na kojima je testiran sistem su takvi da vraćaju 5 najboljih rezultata poklapanja. U nastavku je dato nekoliko primera upita i rezultata. Može se primetiti da kvalitet rezultata nije uvek isti, ali da se najčešće u prvih pet poklapanja nalazi barem jedna slika koju bi čovek, posmatranjem golim okom, okarakterisao kao sliku sličnu upitu. Takođe, iz rezultata se vidi da sistem najbolje radi kada se zadaju slike kod kojih je istaknutiji ambijent scene (primeri slike mora, šume, spavaće sobe...), nego slike kod kojih je najistaknutiji objekat (primer sa automobilom). Kod slike gde je kao upit zadata slika dinosaurusa, očigledno je da je pozadina odigrala veliku ulogu pri nalaženju najboljih rezultata.

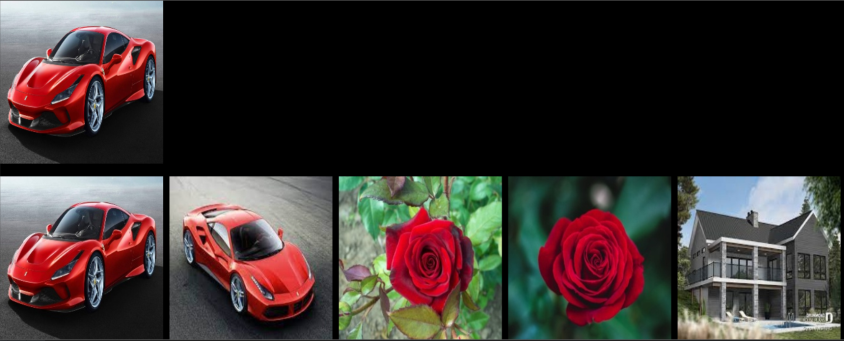


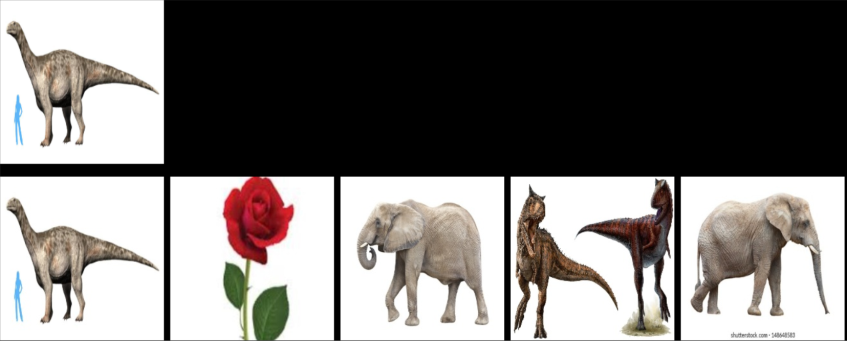
Rezultati poređani po sličnosti koja opada sleva na desno

Upit









# Literatura

Hall-Beyer, M. (2007). *GLCM Texture: A Tutorial v. 1.0 through 2.7.*

Haralick, R. M., Shanmugam, K., & Dinstein, I. (1973). Textural Features for Image Classification. *IEEE Transactions on Systems, Man and Cybernetics*.

Islam, M. B., Ahmed, A., & Kundu, K. (2014). Texture Feature based Image Retrieval Algorithms. *International Journal of Engineering and Technical Research*.

Long, F., Zhang, H., & Feng, D. D. (2003). Fundamentals of Content-Based Image Retrieval. U D. D. Feng, W. Siu, & H. Zhang, *Multimedia Information Retrieval and Management. Signals and Communication Technology.* Springer, Berlin, Heidelberg.

Prasad, B. E., Gupta, A., Toong, H.-m. D., & Madnick, S. E. (1986, February). A microcomputer-based image database management system. *IEEE Transactions on Industrial Electronics*.

Roy, K., & Mukherjee, J. (2013). Image Similarity Measure using Color Histogram, Color Coherence Vector, and Sobel Method. *International Journal of Science and Research*.

Sharma, S. (2011). Color Texture based Image Retrieval System. *International Journal of Computer Applications*.

Spasojević, M. (2020, December 24). *ASP.NET Core Web API – Repository Pattern*. Preuzeto sa CodeMaze: https://code-maze.com/net-core-web-development-part4/

<https://redis.io/>

<https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_glcm.html>

<https://docs.opencv.org/master/d1/db7/tutorial_py_histogram_begins.html>