SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Miroić

Bruno Mataušić

Antonio Hip

Dominik Posavec

Perceptualni hash

SEMINARSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Miroić

Bruno Mataušić

Antonio Hip

Dominik Posavec

Studij: Organizacija poslovnih sustava

Perceptualni hash

SEMINARSKI RAD

Mentor/Mentorica:

Doc. dr. sc. Igor Tomičić

Doc. dr. sc. Petra Grd

Varaždin, siječanj 2021.

Sadržaj

1. Uvod

Tema ovog projektnog zadatka je perceptualno hashiranje koji se obrađuje na kolegiju Sigurnost informacijskih sustava. Pomoću programskog rješenja i pomoću 2 različita tipa perceptualnog hashiranja slika, prikazat ćemo ispitivanje sličnosti binarnih znamenaka dvije slike koje korisnik učita prema svom odabiru. Tipovi hashiranja fotografija u aplikaciji prikazat ćemo pomoću Average Hash i Perception hash algoritama.

Seminarski rad perceptualnog hashiranja podijelili smo na dva načina. Prvi način je teorijski dio rada u njemu su detaljno opisani Average i Perception hash algoritmi ispitivanja sličnosti između fotografija. Uz njihove opće definicije, također su dane i opisane uloge i primjene navedenih algoritama u informacijskom svijetu. Drugi dio ovog seminarskog rada predstavlja praktični, odnosno programski dio projektnog zadataka koji je implementiran u aplikaciju i koji prikazuje način rada prethodno navedenih algoritama.

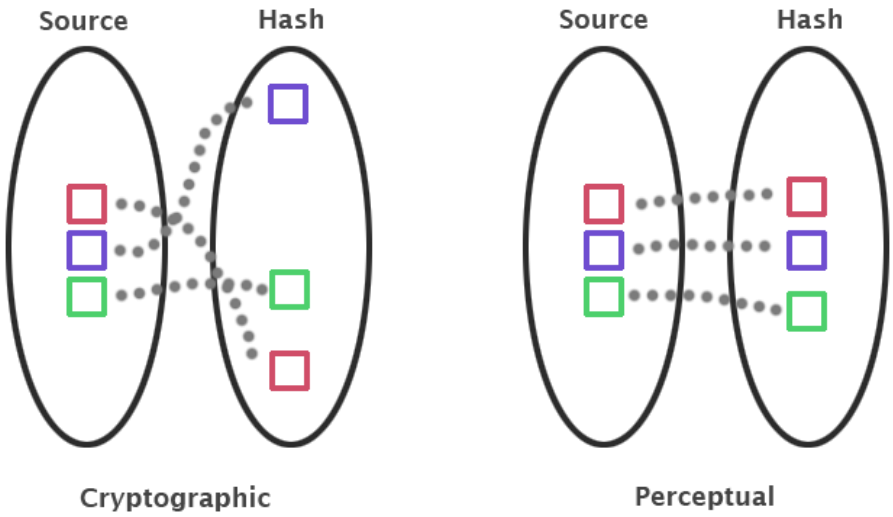
2. Vrste hashiranja

U ovome poglavlju navest ćemo nazive i detaljno opisati pojedine hash algoritme koji se koriste za ispitivanje binarnih sličnosti znamenaka između dvije fotografije.

2.1. Perceptualno hashiranje

Započinjemo sa objašnjenjem pojma perceptualnog hashiranja, i to kao algoritma koji stvara svojevrstan 'otisak prsta' različitih oblika multimedijalnih datoteka. Pomoću tog otiska se provjerava sličnost između dva medijska sadržaja (u našem slučaju slika) putem raznovrsnih svojstva medija. Tako se perceptualno hashiranje može koristiti za detekciju kršenja prava medijskog vlasništva i pronalaženje sličnih fotografija, videa, glazbe i slično.

Često se perceptualno hashiranje uspoređuje sa kriptografskim hashiranjem, upravo zato jer se dva algoritma jako razlikuju jedan od drugoga, a često se koriste. Naime, malim modifikacijama medijskog sadržaja, perceptualna hash vrijednost tog sadržaja se neće jako razlikovati od prvobitne vrijednosti, dok se kod kriptografskog hasha drastično mijenja. Prethodno navedena tvrdnja se najbolje može uočiti na slici 1.



Slika 1. Ilustracija kriptografskog i perceptualnog hashiranja (izvor: https://towardsdatascience.com/black-box-attacks-on-perceptual-image-hashes-with-gans-cc1be11f277)

Ono što moramo imati kod algoritma za perceptualno hashiranje je ekstrakcija svojstava (izvlači snažna svojstva iz inputa koja su nepokvariva umjerenim promjenama), smanjenje svojstava (pretvaranje svojstava u mjerive i zapisive vrijednosti) i nasumičnost. [1] (A Secure Perceptual Hash Algorithm for Image Content Authentication)

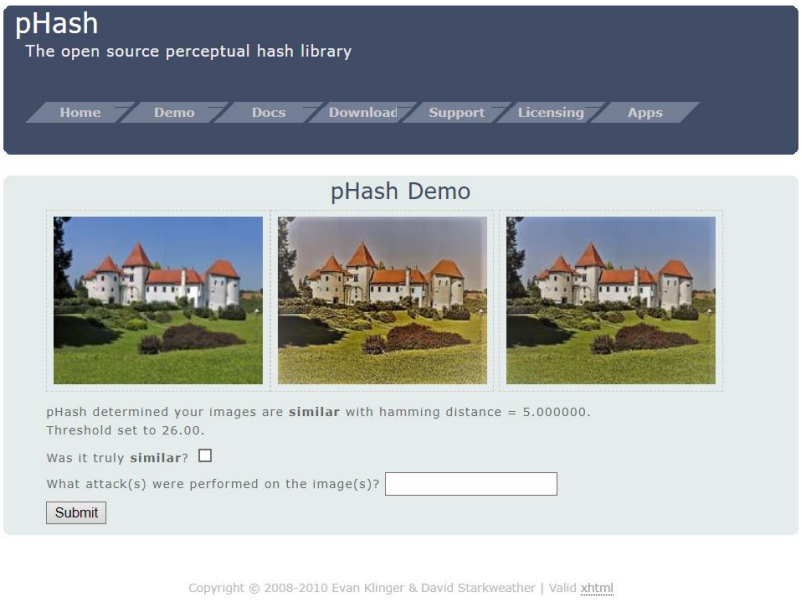
Naš rad i aplikacija će prikazivati dvije različite vrste algoritama hashiranja koje će određivati sličnosti između dvije slike.

2.1.1. pHash

Kao nadogradnja Average Hashu pojavljuje se algoritam pHasha. Razlika između dva algoritma je ta da je pHash otporniji na manje promjene i korekcije, zbog korištenja *diskretne kosinusne transformacije* (DCT) kako bi smanjio frekvencije. Ukratko DCT se koristi u procesiranju signala i kompresiji podataka tako što prikazuje konačan skup točaka podataka u smislu sume funkcija kosinusa koje osciliraju na različitim frekvencijama. Koriste se funkcije kosinusa umjesto sinusa iz razloga što je potrebno manje takvih funkcija da se približi željenom signalu. [2,3]
Algoritam pHasha funkcionira na slijedeći način (prema [2]):

1. Smanjivanje veličine – na veličinu slike veću od 8x8, kako bi se pojednostavilo izračunavanje DCT (32x32)
2. Smanjenje boje – slika je reducirana do sive boje ponovno kako bi se olakšalo izračunavanje
3. Računanje DCT-a – koji razdvaja sliku u skup frekvencija i skalara
4. Smanjenje DCT-a – iako DCT iznosi 32x32, gornji lijevi kut iznosi 8x8 i predstavlja najmanju frekvenciju slike
5. Računanje prosječne vrijednosti – DCT-a
6. Ponovno smanjivanje DCT-a – 64 bitova iz gornjeg lijevog kuta (koeficijenti DCT-a) se preračunavaju u 0 ili 1 ovisno o tome je li vrijednost veća ili manja od prosječne vrijednosti DCT-a → iz tog razloga hash ostaje sličan makar dođe do promjene boje u slici
7. Konstruiranje hasha – pretvaranje 64 bitova u integer

Slika 2 predstavlja primjenu algoritma pHasha kod određivanja sličnosti između originala slike i modificiranog originala. Hamming distance predstavlja broj bitova koji su različiti u hashovima slika.



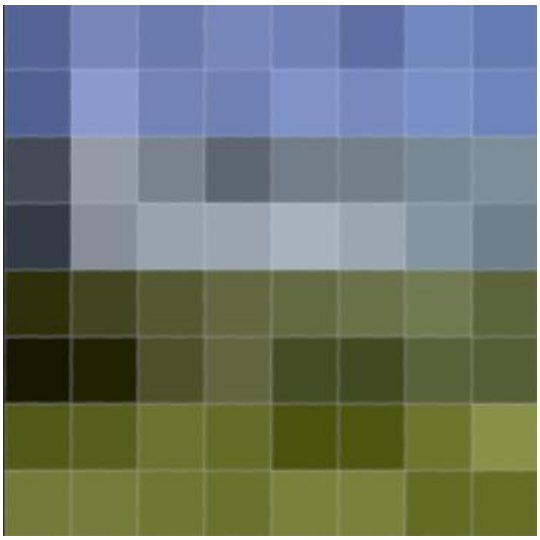
Slika 2. Uspodređivanje slika pomoću pHasha (izvor: https://www.phash.org/cgi-bin/phash-demo-new.cgi)

2.2. Average hash

Average Hash algoritam najjednostavniji je i najkorišteniji algoritam za hashiranje slika. Cilj algoritma je dobiti srednje vrijednosti boja piksela ulazne slike na temelju čega se onda izračunava hash. Average Hash algoritam ima četiri koraka, a da bi se bolje shvatio, provest ćemo ga na primjeru slike Starog grada Varaždin.

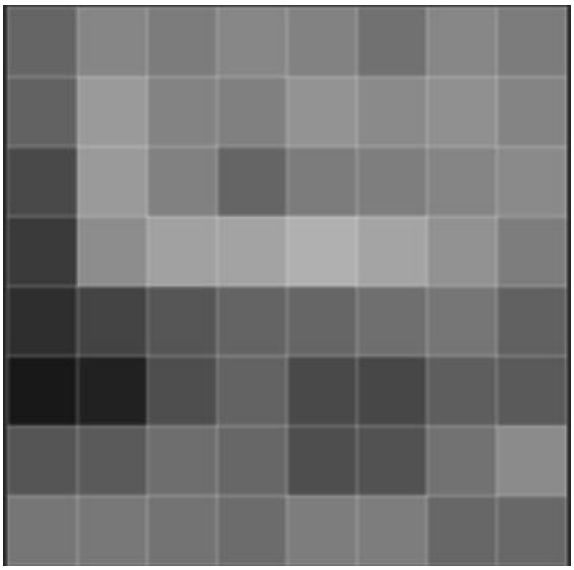
Slika 3. Average hash kroz primjer, (dostupno na: http://www.turizam-trakoscan.hr/turisticke-novinare-jako-zanima-varazdinska-zupanija/ , 8.1.2021.)

1. Smanjiti dimenziju slike na 8x8:
    - U ovom koraku dimenzije svake slike smanjuje se na 8x8 piksela neovisno o omjeru slike. Rezultat toga je slika od 64 piksela koja gubi detalje, ali zadržava najgrublji sastav boja pa je lakše primijetiti sličnosti i izračunati hash.



Slika 4. Average hash - prvi korak, [autorski rad]

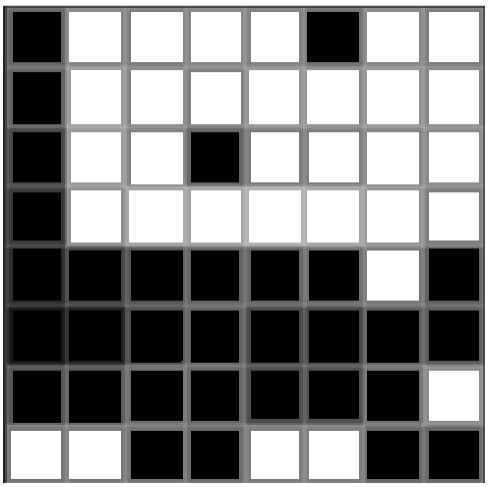
2. Smanjenje broja boja ili bojanje u sivo:
    - U ovom koraku smanjuju se vrijednosti crvenih, zelenih i plavih tonova u jedan raspon sive boje za svaki od 64 piksela. Rezultat je 8x8 slika sa pikselima vrijednosti sive boje u rasponu 0-255.



Slika 5. Average hash – drugi korak, [autorski rad]

3. Izračun prosječne vrijednosti boje:
    - U ovom koraku računa se prosječna vrijednost sive boje na način da se izračuna suma boja svih 64 piksela i podijeli sa 64. Pritom redoslijed zbrajanja i čitanja nije bitan. Dobivena vrijednost koristi se u sljedećem koraku.

4. Stvaranje hasha:
    - Hash se stvara na temelju dobivene prosječne vrijednosti sive boje. Za svaki piksel uspoređuje se vrijednost sive boje piksela sa prosječnom vrijednosti dobivenom u prijašnjem koraku. Ako je vrijednost sive boje piksela veća ili jednaka prosječnoj vrijednosti sive, taj piksel predstavlja 1. Ako je vrijednost sivog piksela manja od prosječne vrijednosti, taj piksel predstavlja 0. Nakon usporedbe svih 64 piksela, dobiva se 64-bitna vrijednost hasha.



Slika 6. Average hash – korak pet, [autorski rad]

2.3. Usporedba perceptualnog i average hash algoritma

Kada su slike u pitanju, visoke frekvencije daju slici detalje dok niske frekvencije prikazuju strukturu slike. Iz toga možemo zaključiti da „velike“ slike sa mnogo detalja sadrže mnogo visokih frekvencija, dok vrlo „male“ slike ne sadrže mnogo detalja pa tako sadrže samo niske frekvencije.

Kod average hash algoritma vidimo da prvo što radimo jest smanjivanje odabrane slike na 8x8, odnosno sveukupno 64 piksela upravo da bi svakoj slici maknuli

visoke frekvencije i gledali samo niske, pa tako eliminiramo mogućnost odstupanja hash vrijednosti zbog veličine slike i formata prikaza. Kod average hash algoritma sama hash vrijednost zadane slike se neće mnogo promijeniti ukoliko se promijeni svjetlina, kontrast ili čak i boje, a najbolje od svega što je taj algoritam vrlo jednostavan i brz. Iz toga svega možemo zaključiti da ako po ovom algoritmu dobijemo 0 različitih bitova, najvjerojatnije su slike vrlo slične, ali ako je 10 ili više različitih bitova to najvjerojatnije znači da su dvije slike vrlo različite. Iako je average hash algoritam vrlo jednostavan i brz, za neku detaljniju usporedbu slika nije dovoljno detaljan zbog toga što postoji mogućnost lažnih negativnih podudaranja. Ukoliko se na sliku primjeni gama korekcija automatski se mijenja prosječna vrijednost boje određenog piksela pa se samim tim i mijenja koji je bit iznad ili ispod prosjeka. Zbog toga u igru ulazi perceptual hash algoritam koji koristi DCT funkciju za smanjivanje frekvencija slike.

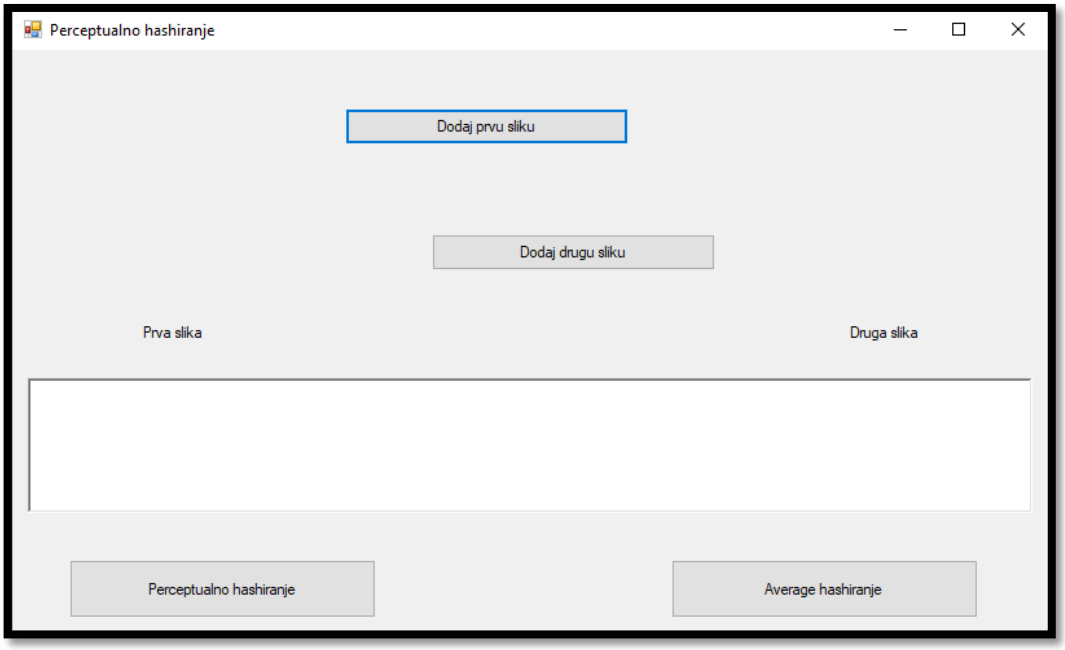
Perceptual hash isto tako počinje sa smanjivanjem slike, kao i kod average hash algoritma, međutim on ju smanjuje na 32x32 da bi se olakšalo računanje DCT, a ne smanjivanje visokih frekvencija. Nakon računanja DCT se opet računa prosjek DCT vrijednosti, s tim da se uzimaju samo DCT vrijednosti 8x8 niskih frekvencija i to bez prvog elementa zbog toga što se DC koeficijent može značajno razlikovati od ostalih vrijednosti i tako poremetiti vrijednost prosjeka. Zatim se kod perceptualnog hash algoritma isto kao i kod average hash algoritma gleda prosjek, samo što se kod perceptualnog hash algoritma gleda prosjek DCT i dali su vrijednosti pojedinih piksela veće ili manje od prosjeka pa zbog toga perceptulani hash algoritam može zanemariti i promjene na slici kao što su gama korekcije slike i sl.

3. Aplikacija za percepcijsko hashiranje

Ovo poglavlje predstavlja praktični dio izrade projektnog zadataka i u sljedećim poglavljima prikazat ćemo početnu formu aplikacije, kao i korake rješavanja same aplikacije tijekom određivanja sličnosti između dvije fotografije. Kako bi mogli pristupiti programskom rješenju projektnog zadatka, ono je dostupno na sljedećoj web adresi: https://github.com/lmiroic/Perceptualni-hash.

3.1. Perpcepcijski hash – opis aplikacije

Kao što smo prethodno spomenuli sama namjena i svrha kreirane aplikacije ovog projektnog zadatka je određivanje postotka sličnosti između dvije fotografije koje korisnik unese u aplikaciju, što se u konačnici rješava pomoću implementiranih hash algoritama. Hash algoritmi koji su implementirani unutar same aplikacije su pHash i Average hash. Programsko okruženje koje je korišteno za izrađivanje aplikacije je Visual Studio, preciznije pomoću C# programskog jezika. Forma aplikacije kreirana je na način da korisnik proizvoljno unese dvije fotografije u aplikaciju i odabirom na određeni algoritam koji je implementiran izračunava se postotak sličnosti između dvije fotografije. Izgled početne forme vidi se na idućoj fotografiji.



Slika 7. Prikaz početne forma aplikacije, [autorski rad]

3.2. Implementirani algoritmi u aplikaciji

U idućim potpoglavljima opisat ćemo odabrane algoritme kroz kodove i primjere slika na koji način oni rade i daju nam konačan rezultat.

3.2.1. pHash algoritam

Prilikom odabira i unosa željenih fotografija u aplikaciju, potrebno je pritisnuti na gumb „Perceptualno hashiranje“ kako bi algoritam počeo izračunavati postotak sličnosti između dvije unesene fotografije.Programsko rješenje u koda navedenim algoritmom izleda na sljedeći način:

```
namespace Sloj_poslovne_logike.Hash_funkcije
{
    9 references
    public class PerceptualnoHashiranje : Hashiranje
    {
        private Digest prvobitniHashSlike;
        2 references
        public PerceptualnoHashiranje(Bitmap bitmap)
        {
            this.prvobitniHashSlike = ImagePhash.ComputeDigest(bitmap.ToLuminanceImage());
            byte[] hashSlikeBytes = Encoding.ASCII.GetBytes(this.prvobitniHashSlike.ToString());
            this.hashiranaVrijednost = VratiBitove(new BitArray(hashSlikeBytes));
        }
        2 references
        public Digest GetDigest()
        {
            return this.prvobitniHashSlike;
        }
        1 reference
        public static double RačunanjeSlicnostiPercepcijskogHasha(PerceptualnoHashiranje hashPrveSlike, PerceptualnoHashiranje hashDrugeSlike)
        {
            return ImagePhash.GetCrossCorrelation(hashPrveSlike.GetDigest(), hashDrugeSlike.GetDigest()) * 100;
        }
    }
}
```

Slika 8. Prikaz programskog rješenja za pHash algoritam, [autorski rad]

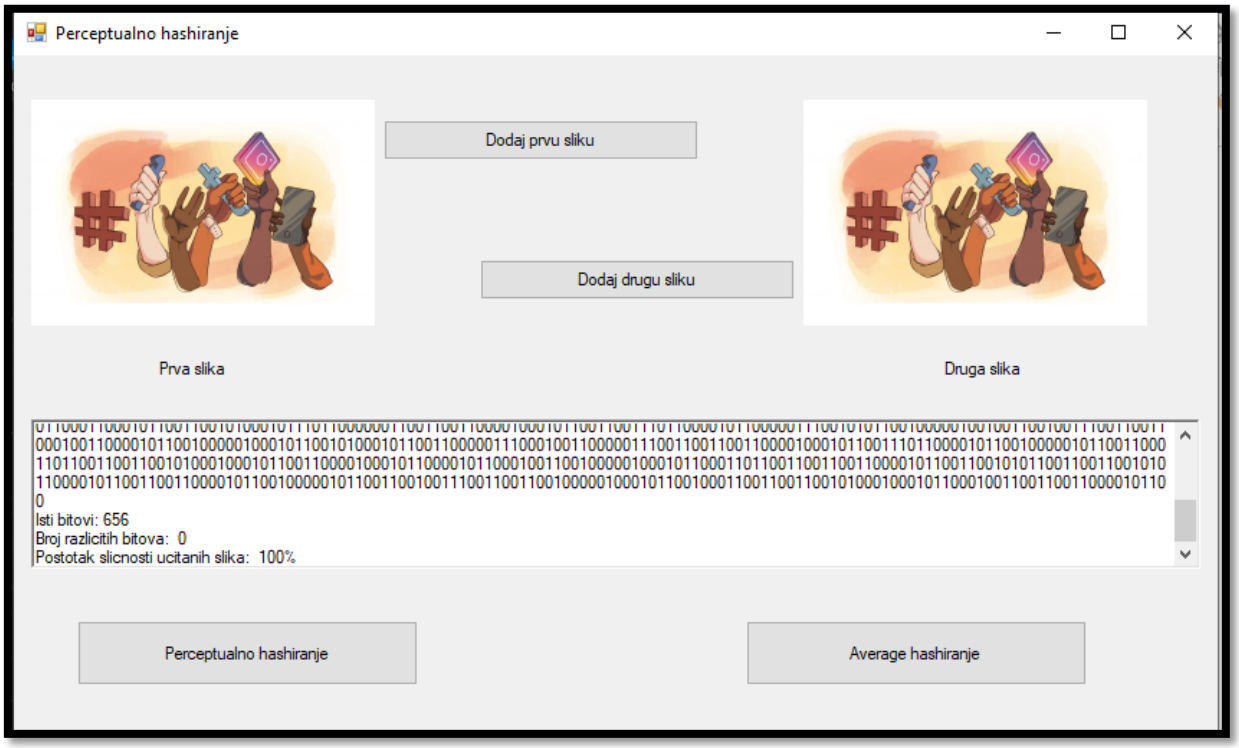
Sljedeća fotografija prikazuje programski kod koji se odnosi na gumb „Perceptualno hashiranje“:

```
1 reference
private void btnPerceptualniHash_Click(object sender, EventArgs e)
{
    try
    {
        rTxtIspis.Clear();
        Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje pHash1 = new Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje(bitmapPicture1);
        Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje pHash2 = new Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje(bitmapPicture2);
        rTxtIspis.AppendText("Hash prve slike:  " + pHash1.GetHash());
        rTxtIspis.AppendText(Environment.NewLine);
        rTxtIspis.AppendText("Hash druge slike:  " + pHash2.GetHash());
        rTxtIspis.AppendText(Environment.NewLine);
        int brojIstihBitova = Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje.VratiBrojIstihBitova(pHash1.GetHash(), pHash2.GetHash());
        rTxtIspis.AppendText("Isti bitovi:" + " " + brojIstihBitova.ToString());
        rTxtIspis.AppendText(Environment.NewLine);
        int brojRazlicitihBitova = Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje.VratiBrojRazlicitihBitova(pHash1.GetHash(), pHash2.GetHash());
        rTxtIspis.AppendText("Broj razlicitih bitova:  "+brojRazlicitihBitova.ToString());
        rTxtIspis.AppendText(Environment.NewLine);
        double prosjek = Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje.VratiPostotakSlicnosti(brojIstihBitova, brojRazlicitihBitova, pHash1.GetHash().Length);
        rTxtIspis.AppendText("Postotak slicnosti ucitanih slika:  "+(int)Sloj_poslovne_logike.Hash_funkcije.PerceptualnoHashiranje.RačunanjeSlicnostiPercepcijskogHasha(pHash1, pHash2) + "%");
    }
    catch (Exception ex)
    {

        MessageBox.Show(ex.Message, "Slika nije uploadana!", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

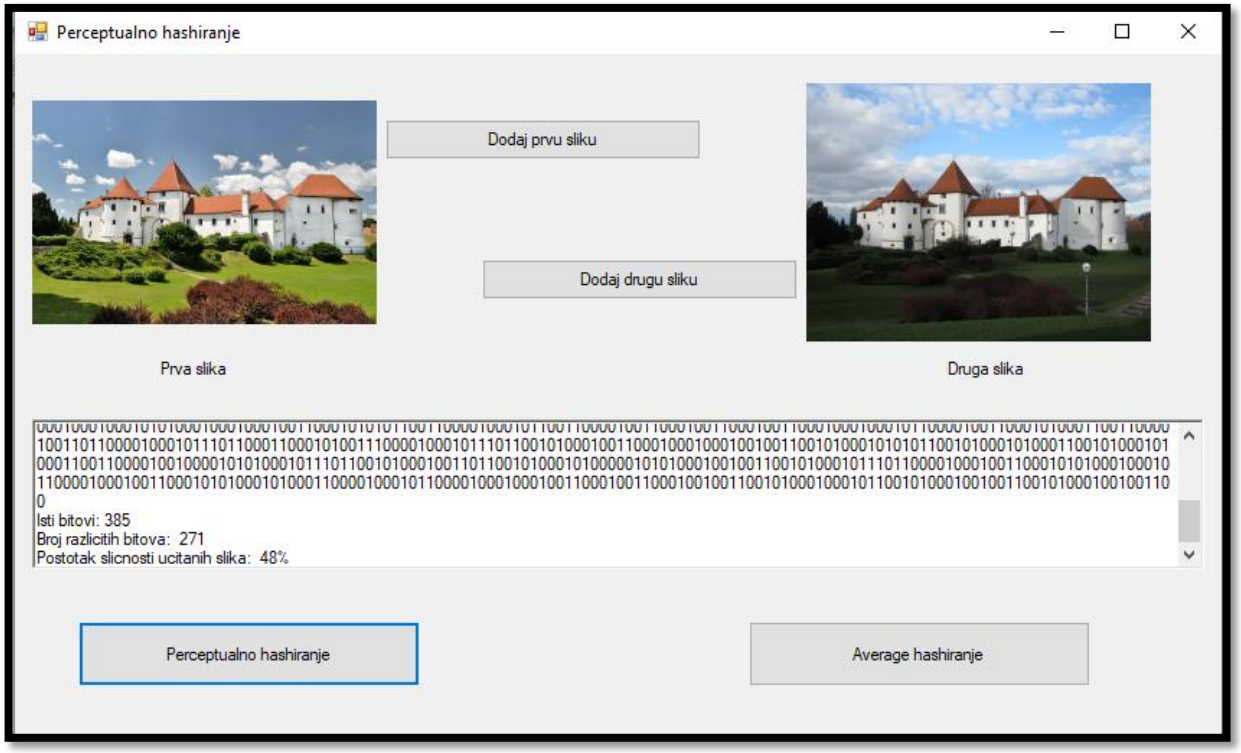
Slika 9. Prikaz programskog koda – percepsijsko hashiranje gumb, [autorski rad]

U idućoj fotografiji unesene su dvije potpuno iste slike za izračunavanje postotka sličnosti između dvije fotografije. Samim time, aplikacija prikazuje hashirane vrijednosti koje se odnose na svaku fotografiju pojedinačno pa samim time na fotografiji se može vidjeti i postotak sličnosti koji iznosi 100%.



Slika 10. Primjer pHash algoritma, [autorski rad]

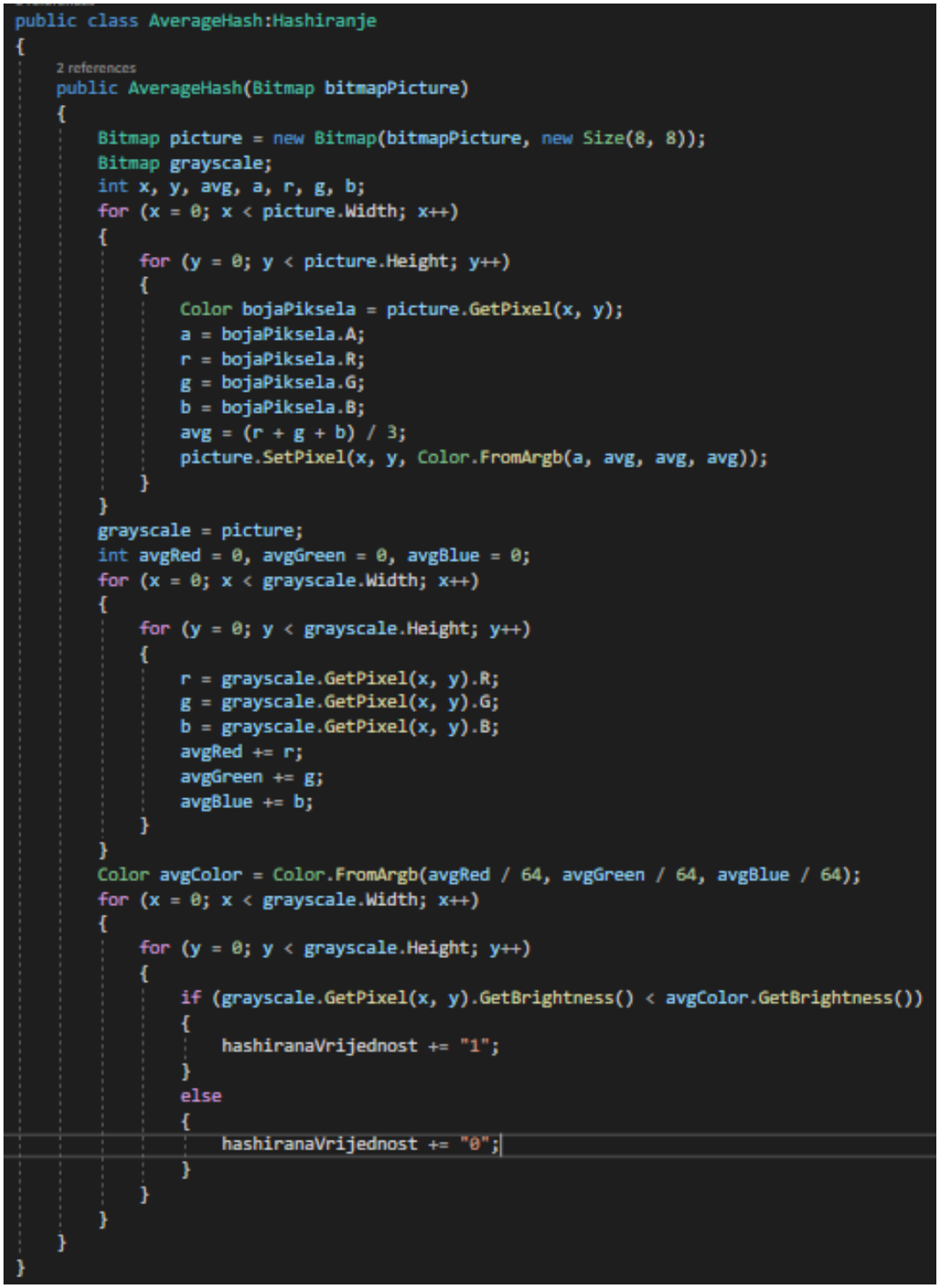
Sljedeći primjer koji je prikaza na fotografiji prikazuje dvije različite fotografije. Broj različitih bitova između njih iznosi 271, dok postotak sličnosti između njih iznosi 48%.



Slika 11. Primjer različitih fotografija pHash algoritma, [autorski rad]

3.2.2. Average algoritam

Sljedeći algoritam je naziva „Average“ algoritam koji također služi za izračunavanje sličnosti između dvije fotografije. Postupak pokretanja izračuna je identičan kao i kod prethodnog algoritma, potrebno je odabrati i unjeti dvije fotografije po izboru u aplikaciju i kliknuti na gumb „Average hashiranje“ kako bi se aplikacija pokrenula. Sami programski kod navedenog algoritma izgleda na sljedeći način:



```
public class AverageHash:Hashiranje
{
    2 references
    public AverageHash(Bitmap bitmapPicture)
    {
        Bitmap picture = new Bitmap(bitmapPicture, new Size(8, 8));
        Bitmap grayscale;
        int x, y, avg, a, r, g, b;
        for (x = 0; x < picture.Width; x++)
        {
            for (y = 0; y < picture.Height; y++)
            {
                Color bojaPiksela = picture.GetPixel(x, y);
                a = bojaPiksela.A;
                r = bojaPiksela.R;
                g = bojaPiksela.G;
                b = bojaPiksela.B;
                avg = (r + g + b) / 3;
                picture.SetPixel(x, y, Color.FromArgb(a, avg, avg, avg));
            }
        }
        grayscale = picture;
        int avgRed = 0, avgGreen = 0, avgBlue = 0;
        for (x = 0; x < grayscale.Width; x++)
        {
            for (y = 0; y < grayscale.Height; y++)
            {
                r = grayscale.GetPixel(x, y).R;
                g = grayscale.GetPixel(x, y).G;
                b = grayscale.GetPixel(x, y).B;
                avgRed += r;
                avgGreen += g;
                avgBlue += b;
            }
        }
        Color avgColor = Color.FromArgb(avgRed / 64, avgGreen / 64, avgBlue / 64);
        for (x = 0; x < grayscale.Width; x++)
        {
            for (y = 0; y < grayscale.Height; y++)
            {
                if (grayscale.GetPixel(x, y).GetBrightness() < avgColor.GetBrightness())
                {
                    hashiranaVrijednost += "1";
                }
                else
                {
                    hashiranaVrijednost += "0";
                }
            }
        }
    }
}
```

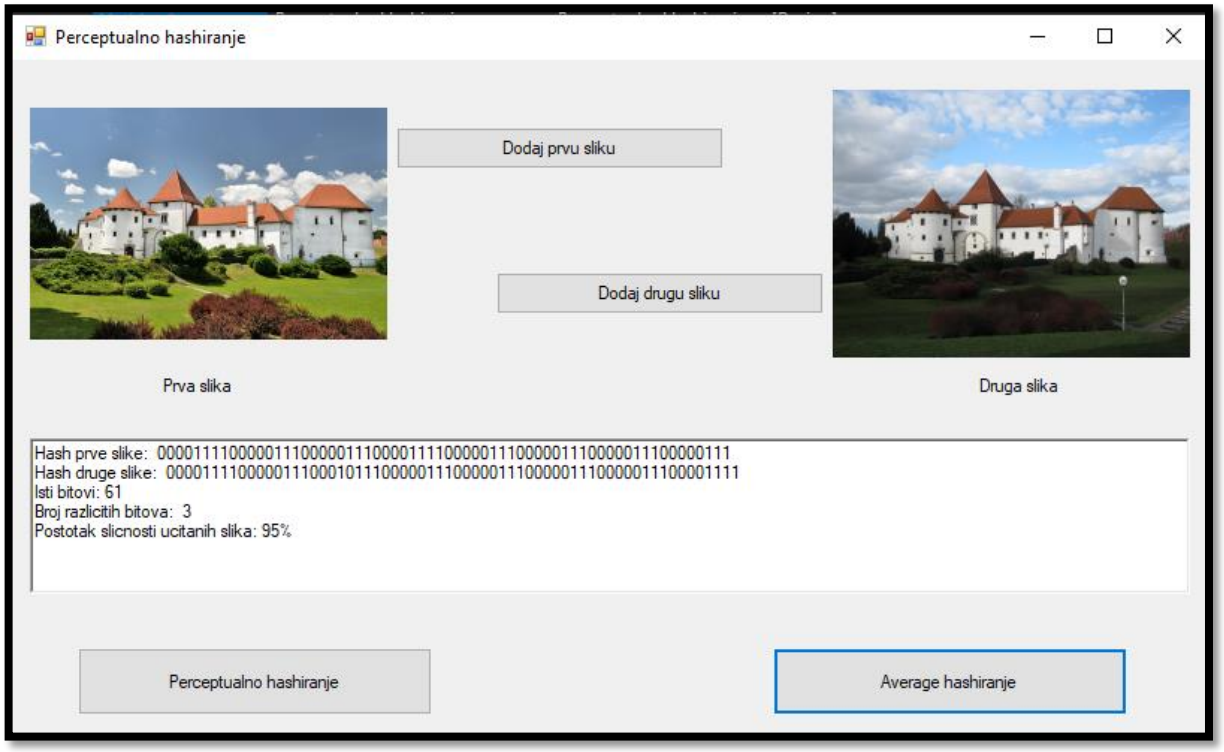
Slika 12. Prikaz programskog koda Average hash algoritma, [autorski rad]

Na sljedećoj fotografiji nalazi se programski kod koji se odnosi na gumb „Average hashiranje“:

```
private void btnAverageHash_Click(object sender, EventArgs e)
{
    try
    {
        rTxtIspis.Clear();
        AverageHash aHash1 = new AverageHash(bitmapPicture1);
        AverageHash aHash2 = new AverageHash(bitmapPicture2);
        rTxtIspis.AppendText("Hash prve slike:  " + aHash1.GetHash());
        rTxtIspis.AppendText(Environment.NewLine);
        rTxtIspis.AppendText("Hash druge slike:  " + aHash2.GetHash());
        rTxtIspis.AppendText(Environment.NewLine);
        int brojIstihBitova = AverageHash.VratiBrojIstihBitova(aHash1.GetHash(), aHash2.GetHash());
        rTxtIspis.AppendText("Isti bitovi:" + " " + brojIstihBitova.ToString());
        rTxtIspis.AppendText(Environment.NewLine);
        int brojRazlicitihBitova = AverageHash.VratiBrojRazlicitihBitova(aHash1.GetHash(), aHash2.GetHash());
        rTxtIspis.AppendText("Broj razlicitih bitova:  " + brojRazlicitihBitova.ToString());
        rTxtIspis.AppendText(Environment.NewLine);
        double prosjek = AverageHash.VratiPostotakSlicnosti(brojIstihBitova, brojRazlicitihBitova, aHash1.GetHash().Length);
        rTxtIspis.AppendText("Postotak slicnosti ucitanih slika: "+prosjek.ToString() + "%");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Slika nije uploadana!", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

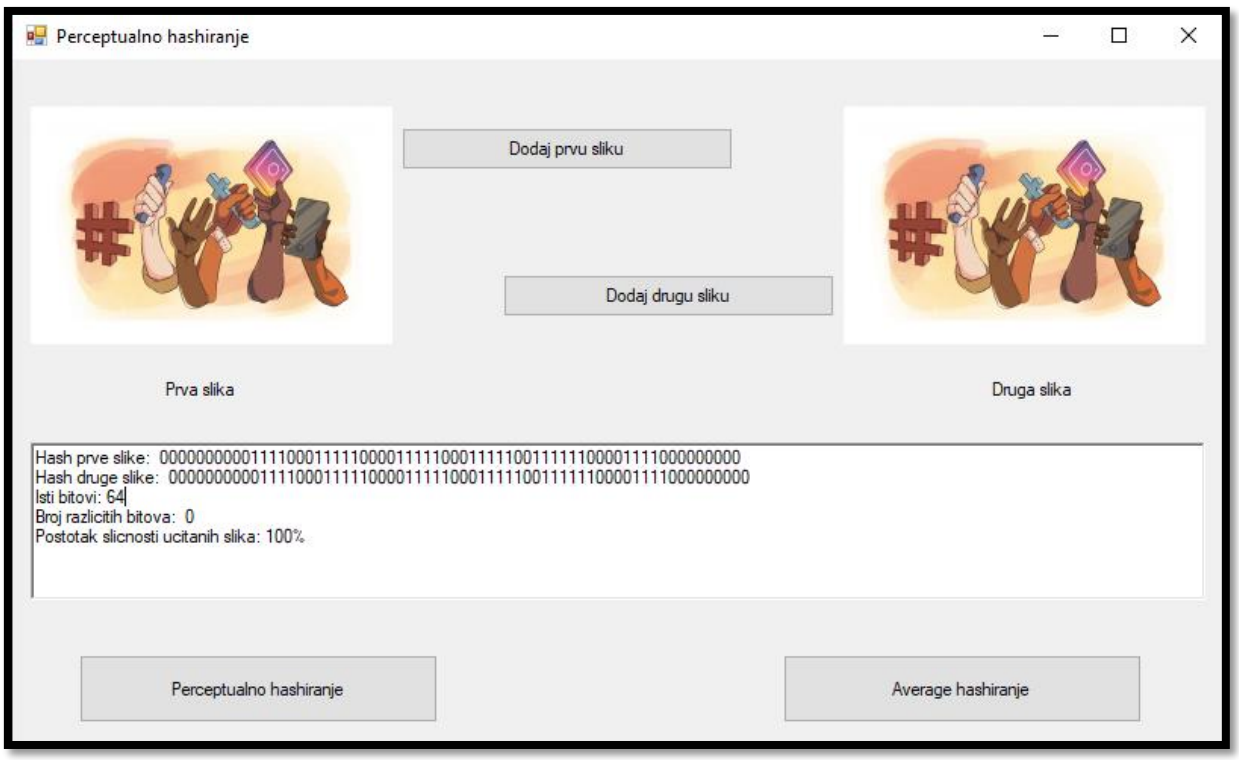
Slika 13. Average hash – programski kod, [autorski rad]

U sljedećim fotografijama prikaz ćemo dva primjera kako „Average hash“ algoritam izračunava postotak sličnost između dvije potpuno iste fotografije i 2 različite. Primjer različite fotografije nalazi se u nastavku, broj istih bitova iznosi 61, broj različitih bitova iznosi 3, a dok postotak sličnosti između fotografija iznosi 95% :



Slika 14. Primjer različith fotografija, [autorski rad]

Sljedeća fotografija prikazuje „Average hash“ sa identičnim fotografijama, koje zajedno imaju 64 bitova i 100% učinak sličnosti međusobom.



Slika 15. Primjer istih fotografija, [autorski rad]

4. Zaključak

Perceptualno hashiranje stvara 'otisak prsta' slike u obliku bitova. Nadalje ti bitovi se provjeravaju sa otiskom druge slike čime bitovi na različitim mjestima pokazuju koliko su slike različite. Kod manjih promjena slike u average hash algoritmu dobije se otisak koji se drastično razlikuje od originala, dok pHash tolerira promjene boje i smanjenje/distorzije slike do određene mjere te daje hasheve koji se relativno slični originalu. Zato se pHash algoritam koristi kod već spomenute forenzike, usporedbe slika i pronalaženje povrede autorskih prava. Napravljena aplikacija dokazuje legitimnost pHasha koji daje hasheve visokog postotka sličnosti između originalne slike i filtrirane slike. Mogao bi se prilagoditi za spajanje na bazu i provjera slika u bazi održavajući autorska prava ili samo za pronalaženje željenih fotografija.

Popis literature

1) Media.neliti.com (2016): *Average Hashing for Perceputal Image Similarity in mobile Phone Application*, dostupno na: https://media.neliti.com/media/publications/92774-EN-average-hashing-for-perceptual-image-sim.pdf, 8.1.2021.
2) Hackerfactor.com(2011): *The Hacker Factor Blog,* dostupno na: http://hackerfactor.com/blog/index.php%3F/archives/432-Looks-Like-It.html, 8.1.2021.
3) Sciencedirect.com (2021): *Discrete Cosine Transform*, dostupno na: https://www.sciencedirect.com/topics/computer-science/discrete-cosine-transform , 8.1.2021.
4) Core.ac.uk (2013): *A Robust Image Hashing Algorithm Resistant Against Geometrical Attacks,* dostupno na: https://core.ac.uk/download/pdf/30312067.pdf , 10.1.2021.
5) Phash.org (2010): *What is a perceptual hash?* ,dostupno na: https://www.phash.org/ , 10.1.2021.

Popis slika