SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jurica Migač

David Lazar

Percepcijsko hashiranje

SEMINARSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jurica Migač

David Lazar

Studij: Informacijsko i programsko inženjerstvo

Percepcijsko hashiranje

SEMINARSKI RAD

Mentori:

Doc. dr. sc. Igor Tomičić

Doc. dr. sc. Petra Grd

Varaždin, siječanj 2020.

Sadržaj

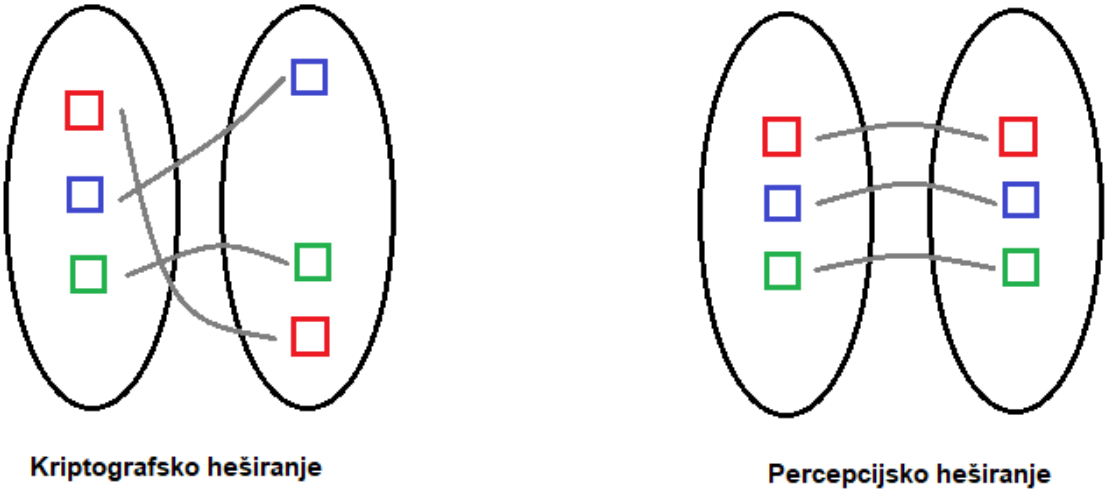
1. Uvod

U ovom projektnom zadatku iz kolegija Sigurnost informacijskih sustava na temu Percepcijskog hashiranja (eng. perceptual hashing) će se unutar programskog rješenja implementirati unos dvije slike korisnika te ispitivanje sličnosti binarnih znamenaka dviju slika kroz četiri različita tipa hashiranja fotografije. Četiri različita tipa hashiranja su Average Hash, Difference Hash, Perception Hash te Wavelet Hash.

Ovaj seminarski rad je podijeljen na dva dijela. Prvi dio je teorijski te je u njemu rečeno što je to percepcijsko hashiranje te su detaljno opisani i objašnjeni gore navedeni tipovi hashiranja. Dok je drugi dio praktični dio projekta u kojem je opisan način rada aplikacije te je prikazano kako izgleda sama aplikacija.

2. Percepcijsko hashiranje

Za razumijevanje ove teme bitno je znati što znači percepcijsko hashiranje i kako ono funkcionira, zbog toga što ima velike razlike između percepcijskog hashiranja i kriptografskog hashiranja. Bitna razlika je u tome da kod kriptografskog hashiranja ako se mijenja ulazna vrijednost koja se treba hashirati, izlazna vrijednost hasha se drastično mijenja. Dok je kod percepcijskog hashiranja suprotno, odnosno ako se malo promijeni ulazna vrijednost, izlazna vrijednost hasha je jako slična kao i kod prve slike.



Slika 1. Razlika između kriptografskog i percepcijskog hashiranja [1]

Kao što možemo primijetiti na slici (Slika 1.), kod kriptografskog hashiranja izlazna vrijednost hasha je nasumična kod male promijene ulaznih vrijednosti. Dok kod percepcijskog hashiranja razlika između ulaznih i izlaznih vrijednosti hasha je gotovo identična. Zbog te svoje specifičnosti perceptualne se hash funkcije koriste u pronalaženju slučajeva kršenja autorskih prava, odnosno plagijata, te digitalne forenzike zbog mogućnosti povezanosti hasheva, pa se mogu naći slični podaci koji koriste različiti vodeni žig.

Percepcijsko hashiranje se svodi na to da se multimedijskom sadržaju daje jedinstveni hash, odnosno „*otisak prsta*“ te ako su dva hasha jako blizu znači da je njihov sadržaj jako sličan. U našem slučaju aplikacija će uspoređivati sličnosti između dvije slike koristeći četiri različita algoritma.

2.1. Average Hash

Jedna od najjednostavnijih perceptualnih hash funkcija za hashiranje slika jest Average Hash. Glavna ideja ove funkcije je izračunavanje prosječne vrijednosti boje piksela slike, koja se upotrebljava za izračunavanje hasha. Postoji metoda koja se temelji na izračunavanju prosječne vrijednosti blokova, ali za razliku od te metode Average Hash funkcija ne dijeli sliku na blokove, već koristi cijelu vrijednost piksela za izračunavanje hasha. Ova metoda se prema [2] dijeli na četiri glavna koraka:

1. **Promjena dimenzije slike**
   U ovom koraku slika se smanjuje na dimenziju 8x8 piksela iz bilo koje dimenzije slike bez obzira na omjer slike. Ovaj korak iz slike uklanja visoke frekvencije slike i detalje. Rezultat ovog koraka je slika u 64 piksela koja predstavlja sastav boja i raspodjelu izvorne slike. Smanjenjem dimenzije slike olakšava se rad sa hashiranjem.

2. **Smanjenje boja slike ili bojanje u sivo**
   U ovom koraku se pokušava smanjiti vrijednost crvene, zelene, plave (RGB) boje u jednu sivu vrijednost boje piksela. Rezultat ovog koraka je slika od 64 piksela, gdje svaki piksel ima vrijednost između 0 i 255.

3. **Izračunavanje prosječne vrijednosti boje**
   Prosječna vrijednost boje se izračunava tako da se zbroje vrijednosti svih piksela te se taj iznos podijeli sa 64. Redoslijed čitanja i zbrajanja vrijednosti nije bitan. Rezultat ovog koraka je prosječna vrijednost boje koja se koristi u sljedećem koraku.

4. **Kreiranje hasha**
   Hash se kreira tako da se usporede vrijednosti svakog piksela slike iz drugog koraka sa prosječnom vrijednošću boje izračunatom u trećem koraku. Ako je vrijednost piksela veća ili jednaka s prosječnom vrijednosti u hashu predstavljaju 1, dok manje vrijednosti predstavljaju 0. Dakle za sliku od 64 piksela se dobije 64-bitnu vrijednost hasha. Ova 64-bitna vrijednost hasha se kasnije može prikazati kao heksadecimalna ili decimalna vrijednost. U ovom radu mi smo za uspoređivanje koristili 64-bitnu vrijednost hasha.

2.2. Difference Hash

Difference Hash funkcija je jednostavna perceptualna hash funkcija koja je slična Average Hash funkciji. Glavna ideja ove funkcije je uspoređivati susjedne piksele slike. Ova metoda se može podijeliti u tri osnovna koraka u kojoj su prva dva koraka jednaka kao i kod Average Hash funkcije. Koraci ove funkcije prema [3] su:

1. **Promjena dimenzija slike**

   U ovom koraku slika se smanjuje na dimenziju 8x8 piksela iz bilo koje dimenzije slike bez obzira na omjer slike. Ovaj korak iz slike uklanja visoke frekvencije slike i detalje. Rezultat ovog koraka je slika u 64 piksela koja predstavlja sastav boja i raspodjelu izvorne slike. Smanjenjem dimenzije slike olakšava se rad sa hashiranjem.

2. **Smanjenje boja slike ili bojanje u sivo**

   U ovom koraku se pokušava smanjiti vrijednost crvene, zelene, plave (RGB) boje u jednu sivu vrijednost boje piksela. Rezultat ovog koraka je slika od 64 piksela, gdje svaki piksel ima vrijednost između 0 i 255.

3. **Kreiranje hasha**

   Hash se kreira tako da se prvo uspoređuju pikseli po redcima slijeva na desno sa svojim desnim susjedom. Ako je manji ili jednaki od svojeg desnog susjeda u hash se zapisuje 1, a u suprotnom 0. Nakon toga se kreira još jedan hash koji se kreira tako da se uspoređuju pikseli po stupcima odozgo prema dolje sa svojim donjim susjedom. Analogno kao i prije, ako je gornji susjed manji ili jednaki donjem susjedu u hash se zapisuje 1, u suprotnom 0. Difference hash nastaje spajanjem tih dva 64-bitna hasha pa tako Difference hash je zapravo 128-bitni hash. Ova 64-bitna vrijednost hasha se kasnije može prikazati kao heksadecimalna ili decimalna vrijednost. U ovom radu mi smo za uspoređivanje koristili bitovni zapis.

2.3. Perception Hash

Perception Hash funkcija je nešto kompliciranije od funkcija koje su ranije navedene. Ona je bolja funkcija za izračunavanje hasha koja nadograđuje Average Hash funkciju koristeći diskretnu kosinusovu transformaciju (eng. *Discrete cosine transform*, dalje u tekstu DCT) tako da dobije najosjetljivije informaciju o ljudskom vidnom sustavu. Kako navodi [4], DCT se široko koristi u kompresiji slike za smanjenje suvišnosti i povezanosti u slikama. Pretvara slike od piksela u frekvencijsku domenu odvajanjem slike u skup frekvencija, a koeficijent velikog dijela frekvencijskih komponenta je različita od 0. Prema [3] DCT se izračunava na sljedeći način:

$$X_k = \sum_{n=0}^{N-1} 2n * \cos(\pi * k * \frac{2n+1}{2N}) \quad \forall k \in [0, N]$$

Gdje X_k predstavlja DCT koeficijent. Implementacija ovog algoritma se svodi na nekoliko sljedećih koraka:

1. Smanjiti veličinu slike kao i kod Average Hash funkcije, ali na veličinu 32x32 piksela, zbog toga što pojednostavljuje izračunavanje DCT-a zadržavajući visoku frekvenciju.
2. Smanjiti boje slike, odnosno bojanje slike u sive nijanse kao i kod Average Hash funkcije.
3. Provođenjem DCT-a, dobije se matrica DCT koeficijenata veličine 32x32, gdje je energija slike sadržana u nekoliko koeficijenata.
4. Smanjiti DCT tako da zadržimo samo 8x8 gornje lijeve komponente kod matrice koeficijenata kako bi predstavljale najniže frekvencije slike.
5. Izračunava se prosječna DCT vrijednost koeficijenata.
6. Kreira se hash tako da se DCT koeficijenti uspoređuju s prosječnom DCT vrijednošću te ako je DCT koeficijent u matrici veći ili jednaki od prosjeka u hash se zapisuje 1, a ako je manji zapisuje se 0.

2.4. Wavelet Hash

Wavelet Hash funkcija je jako slična Perception Hash funkciji, samo što Wavelet Hash funkcija umjesto DCT-a, koristi diskretnu valnu transformaciju (eng. *Discrete wavelet transform*, dalje u tekstu DWT). DWT koristi signal koristeći wavelet funkcije s različitih mjesta i ljestvica te se one prikladne za predstavljanje signala velikom promjenom ulaznih signala, što rezultira manjom količinom informacija u frekvencijskoj domeni. Wavelet funkcija se implementira u sljedećim koracima:

1. Smanjiti veličinu na veličinu 32x32 piksela, zbog toga što pojednostavljuje izračunavanje DWT-a zadržavajući visoku frekvenciju
2. Smanjiti boje slike, odnosno bojanje slike u sive nijanse Provođenjem DWT-a, dobije se matrica DWT koeficijenata veličine 32x32, gdje je energija slike sadržana u nekoliko koeficijenata.
3. Smanjiti DWT tako da zadržimo samo 8x8 gornje lijeve komponente kod matrice koeficijenata kako bi predstavljale najniže frekvencije slike.
4. Izračunava se prosječna DWT vrijednost koeficijenata.
5. Kreira se hash tako da se DWT koeficijenti uspoređuju s prosječnom DWT vrijednošću te ako je DWT koeficijent u matrici veći ili jednaki od prosjeka u hash se zapisuje 1, a ako je manji zapisuje se 0.

Kako navodi [5],DWT se često koristi za uklanjanje suvišnih podataka, poput piksela u slikama. DWT se uspješno koristi za smanjenje buke, kompresiju slike, smanjenja slike te analize audio signala. Zbog prirode valne transformacije, očekuje se da će Wavelet Hash bolje djelovati na slikama s manjom količinom intenzivnih promjena, tj. Prostornim podacima visokog kontrasta. Također, razumno je pretpostaviti da će Perception Hash bolje djelovati na slikama sa suzdržanijim prostornim promjenama.

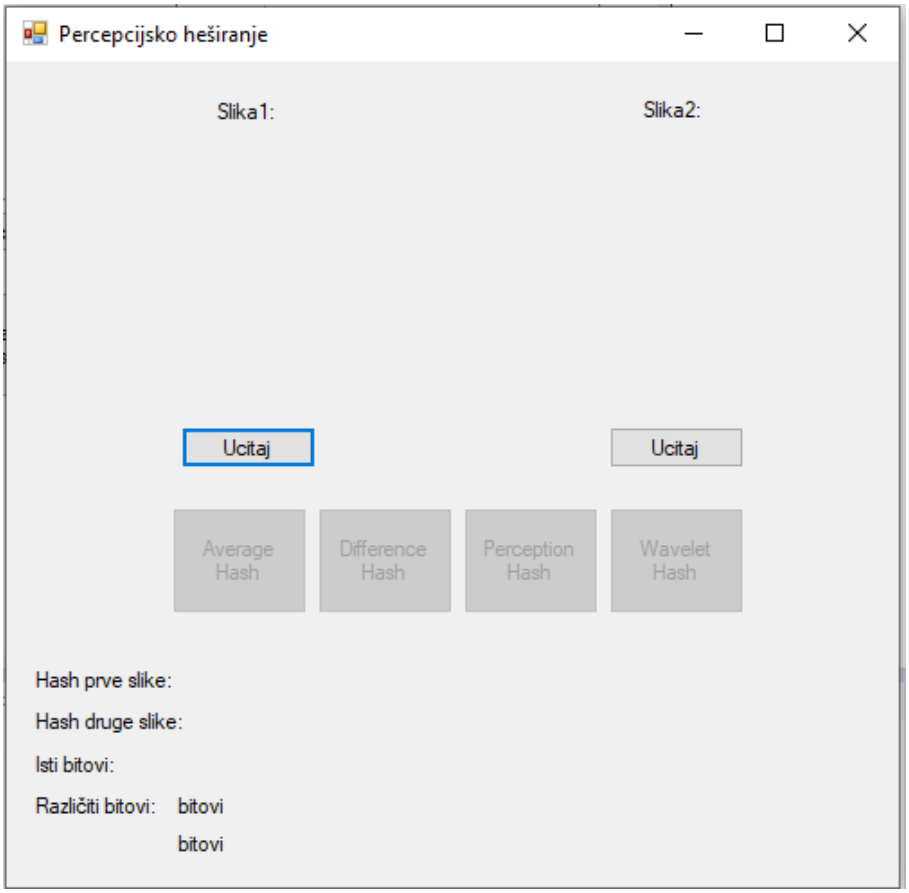
3. Aplikacija za određivanje sličnosti između dvije fotografije – Percepcijsko heširanje

U ovom poglavlju su opisani detalji o aplikaciji koju smo najavili na početku rada te su prikazane njezine funkcionalnosti.

Programsko rješenje za percepcijsko hashiranje izrađeno je na github sustavu za upravljanje izvornim kodom na sljedećem repozitoriju: https://github.com/jmigac/Percepcijsko-hesiranje.

3.1. Opis aplikacije

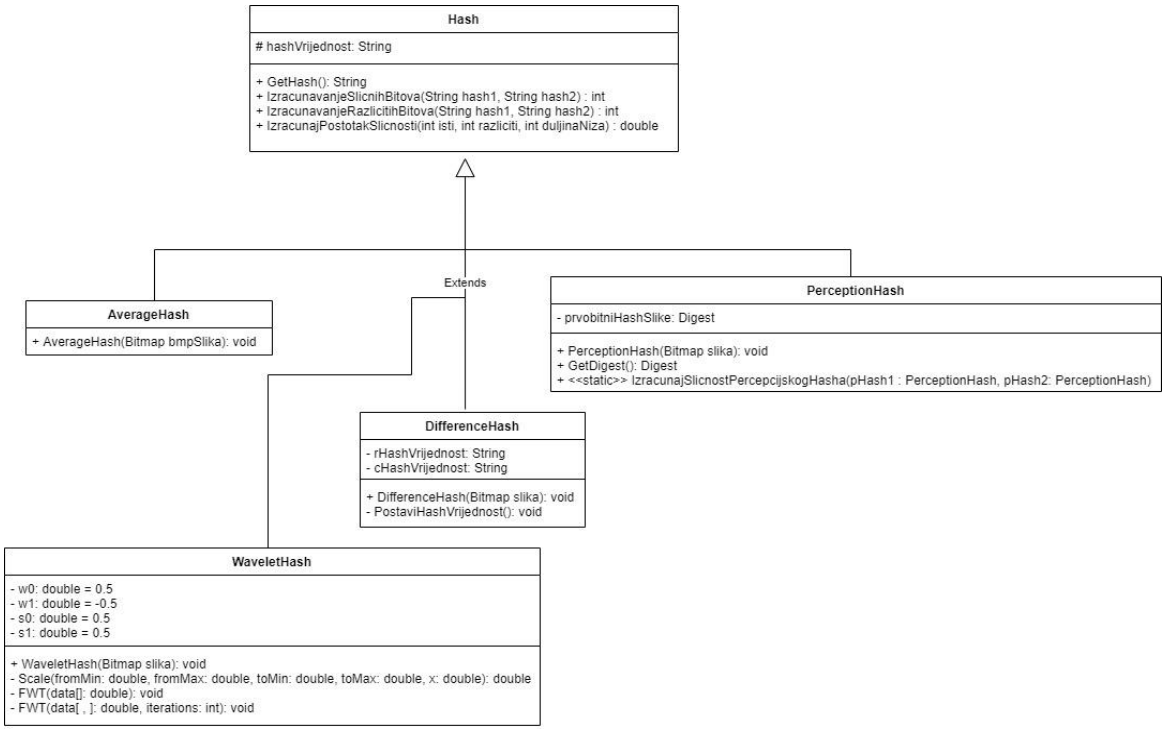
Aplikacija Percepcijsko heširanje je aplikacija koja određuje postotak sličnosti dviju fotografija na temelju izračunatih hasheva. Aplikacija omogućuje izračunavanje hasha po četiri različitih hash algoritma, a to su Average Hash, Difference Hash, Perception Hash te Wavelet Hash. Aplikacija je izrađena u programskom alatu Microsoft Visual Studio-u, u C# jeziku. Aplikacija sadrži jedan obrazac koji omogućuje korisniku da učita željene fotografije te mogućnost izračunavanja željenog hasha. Početna forma je prikazana na sljedećoj slici (Slika 2.)



Slika 2. Početna forma aplikacije Percepcijsko heširanje

3.2. Dijagram klasa

Dijagram klasa se sastoji od pet različitih klasa. Klase AverageHash, DifferenceHash, PerceptionHash, WaveletHash proširuju klasu Hash. Dijagram klasa je prikazan na sljedećoj slici.



Slika 3. Dijagram klasa aplikacije Percepcijsko heširanje

Opis svih metoda klase Hash:

- **GetHash()** - Funkcija vraća hash vrijednost za određenu sliku proslijeđenu preko objekta
- **IzracunavanjeSlicnihBitova(String hash1, String hash2)** - Funkcija izračunava broj istih bitova na istim pozicijama.
- **IzracunavanjeRazlicitihBitova(String hash1, String hash2)** - Funkcija izračunava broj različitih bitova na istim pozicijama u nizu.
- **IzracunajPostotakSlicnosti(int isti, int razliciti, int duljinaNiza)** - Funkcija izračuna prosjek sličnosti između istih i različitih bitova.
- **GetBitsFromBytes(BitArray bitZapis)** - Funkcija vraća bitove iz niza bajtova u string obliku.

Opis svih metoda klase AverageHash:

- **AverageHash(Bitmap bmpSlika)** - Konstruktor funkcije kreira hash vrijednost na temelju izračunatih vrijednosti prosječne *brightness* vrijednosti na temelju slike koja je pretvorena u *grayscale* te kasnije s izračunatim vrijednostima uspoređena za izračun finalne hash vrijednosti.

Opis svih metoda klase DiffferenceHash:

- **DifferenceHash(Bitmap slika)** - Konstruktor klase DifferenceHash generira hash vrijednost slika na temelju izračunatih vrijednosti rHashVrijednost i cHashVrijednost koji su zapravo usporedba prethodnika i sljedbenika za određeni stupac, odnosno redak.
- **PostaviHashVrijednost()** - Funkcija zbraja rHashVrijednost i cHashVrijednost, odnosno spaja navedene stringove.

Opis svih metoda klase PerceptionHash:

- **PerceptionHash(Bitmap bitmap)** - Konstruktor izračunava hash vrijednost u string obliku i učahuruje unutar klase.
- **GetDigest()** - Funkcija vraća originalni zapis hash vrijednosti slike.
- **IzracunajSlicnostPercepcijskogHasha(PerceptionHash pHashPrveSlike, PerceptionHash pHashDrugeSlike)** - Funkcija izračunava sličnost između dviju percepcijskih hasheva.

Opis svih metoda klase WaveletHash:

- **WaveletHash(Bitmap bitmap)** - Funkcija generira hash vrijednost za ulaznu vrijednost Bitmap slike.
- **FWT(double[] data)** - Funkcija računa podatke pojednostavljanja vrijednosti boja na određenim lokacijama za određeni stupac ili redak ovisno o lokaciji.
- **FWT(double[,] data, int iterations)** - Funkcija generira diskretni wavelet koeficijent od ulaznog signala data koristeći "iterations" iteracija od jednostavnog wavelet filtera definiranog prema Mallatovom algoritmu. Glavna zadaća je demonstrirati rastavljanje osnovnih boja na s više inačica na manji broj istih dimenzija boja ovisno o iteraciji. Primjerice Crvena boja --> (početna inačica 128 prikaza crvene) svesti kroz 10 iteracija na recimo 6.

3.3. Funkcionalnost aplikacije

3.3.1. Usporedba slika pomoću Average Hash algoritma

Nakon što se odaberu dvije fotografije koje se žele usporediti, klikom na gumb „Average Hash“ izračunava se Average hash za obje fotografije te se dobije postotak koliko je bitova jednako na istim pozicijama, odnosno slično fotografija. Average hash se izračunava na sljedeći način :

1. Slika se smanji na dimenziju 8x8 piksela, koji se omogućuje sljedećom linijom koda:

```
Bitmap slika = new Bitmap(bmpSlika, new Size(8, 8));
```

2. Smanjenje boje slike ili bojanje u sivo omogućuje se sljedećim linijama koda

```
Bitmap grayscale;
            int x, y, avg, a, r, g, b;
            for (x = 0; x < slika.Width; x++)
            {
                for (y = 0; y < slika.Height; y++)
                {
                    Color bojaPiksela = slika.GetPixel(x, y);
                    a = bojaPiksela.A;
                    r = bojaPiksela.R;
                    g = bojaPiksela.G;
                    b = bojaPiksela.B;
                    avg = (r + g + b) / 3;
                    slika.SetPixel(x, y, Color.FromArgb(a, avg, avg,
avg));
                }
            }
            grayscale = slika;
```

3. Izračunavanje prosječnog hasha omogućuje se sljedećim kodom:

```
int avgCrvena = 0, avgZelena = 0, avgPlava = 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;
                    avgCrvena += r;
                    avgZelena += g;
                    avgPlava += b;
                }
            }
            Color prosjekBoja = Color.FromArgb(avgCrvena / 64,
avgZelena / 64, avgPlava / 64);
```

4. Upoređivanje piksela sa prosječnom vrijednošću:

```
for (x = 0; x < grayscale.Width; x++)
            {
                for (y = 0; y < grayscale.Height; y++)
                {
                    if (grayscale.GetPixel(x, y).GetBrightness() <
prosjekBoja.GetBrightness())
                    {
                        hashVrijednost += "1";
                    }
                    else
                    {
                        hashVrijednost += "0";
                    }
                }
            }
```

Nakon što su se izračunali hashevi slika, slični bitovi se izračunavaju pomoću metode IzracunavanjeSlicnihBitova(String hash1, String hash2) čiji kod izgleda ovako:

```
public static int IzracunavanjeSlicnihBitova(String hash1, String hash2)
        {
            int brojIstihBitova = 0;
            int duljinaPrvogHasha = hash1.Length;
            int duljinaDrugogHasha = hash2.Length;
            if (!duljinaPrvogHasha.Equals(duljinaDrugogHasha))
            {
                throw new Exception("Veličine hash vrijednosti nisu
jednake!");
            }
            var arraySlovaHash1 = hash1.ToCharArray();
            var arraySlovaHash2 = hash2.ToCharArray();
            for (int i = 0; i < arraySlovaHash1.Length; i++)
            {
                if (arraySlovaHash1[i].Equals(arraySlovaHash2[i]))
                {
                    brojIstihBitova++;
                }
            }
            return brojIstihBitova;
        }
```

A različiti bitovi metodom IzracunavanjeRazlicitihBitova(String hash1, String hash2):

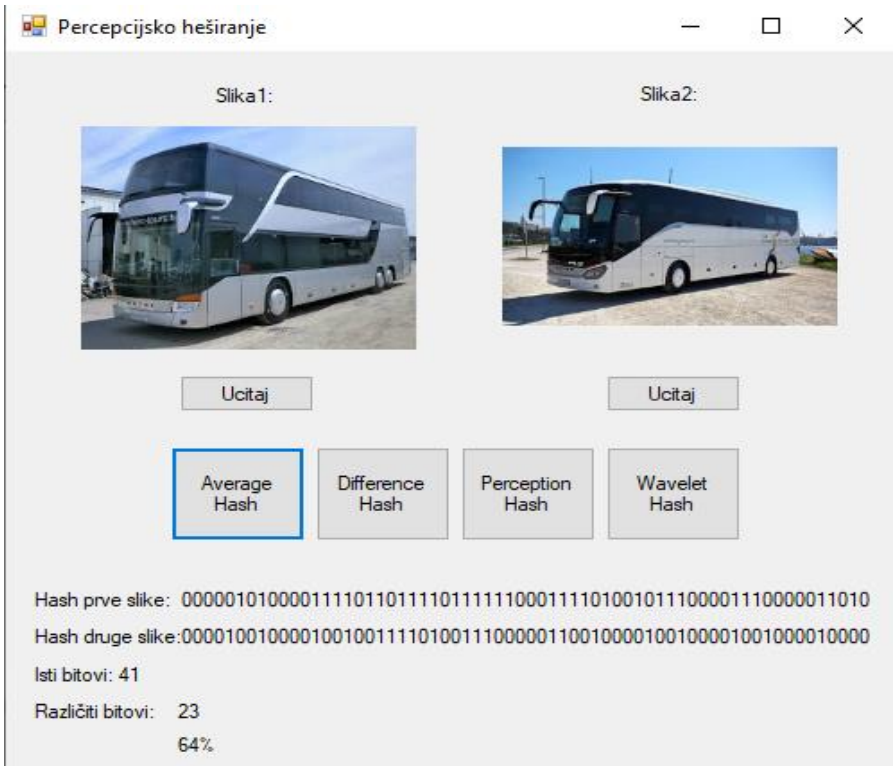
```
public static int IzracunavanjeRazlicitihBitova(String hash1, String hash2)
        {
            int brojRazlicitihBitova = 0;
            var arraySlovaHash1 = hash1.ToCharArray();
            var arraySlovaHash2 = hash2.ToCharArray();
            int duljinaPrvogHasha = hash1.Length;
            int duljinaDrugogHasha = hash2.Length;
            if (!duljinaPrvogHasha.Equals(duljinaDrugogHasha))
            {
                throw new Exception("Veličine hash vrijednosti nisu
jednake!");
            }
            for (int i = 0; i < arraySlovaHash1.Length; i++)
```

```
            {
                if (!arraySlovaHash1[i].Equals(arraySlovaHash2[i]))
                {
                    brojRazlicitihBitova++;
                }
            }
            return brojRazlicitihBitova;
        }
```

Te se postotak sličnosti izračunava metodom IzracunajPostotakSlicnosti(int isti, int razliciti, int duljinaNiza):

```
public static double IzracunajPostotakSlicnosti(int isti, int razliciti,
int duljinaNiza)
        {
            double prosjek = 0;
            if (isti != 0 && razliciti != 0)
            {
                prosjek = (isti * 100 / duljinaNiza);
            }
            if (isti != 0 && razliciti == 0)
            {
                prosjek = 100;
            }
            return prosjek;
        }
```

Na slici (Slika 4.) je prikazana usporedba dviju fotografija autobusa. U aplikaciji su nam prikazani izračunati hashevi prve i druge fotografije, broj istih i različitih bitova te postotak sličnosti. Vidimo da je postotak sličnosti fotografija 64%.



Slika 4. Primjer rada Average Hash algoritma

3.3.2. Usporedba slika pomoću Difference Hash algoritma

Nakon što se odaberu dvije fotografije koje se žele usporediti, klikom na gumb „Difference Hash“ izračunava se Difference hash za obje fotografije te se dobije postotak koliko je bitova jednako na istim pozicijama, odnosno slično fotografija. Difference hash se izračunava na sljedeći način :

1. Slika se smanji na dimenziju 9x9 piksela

```
Bitmap pomocnaSlika = new Bitmap(slika, new Size(9, 9));
```

2. Smanjenje boje slike ili bojanje u sivo

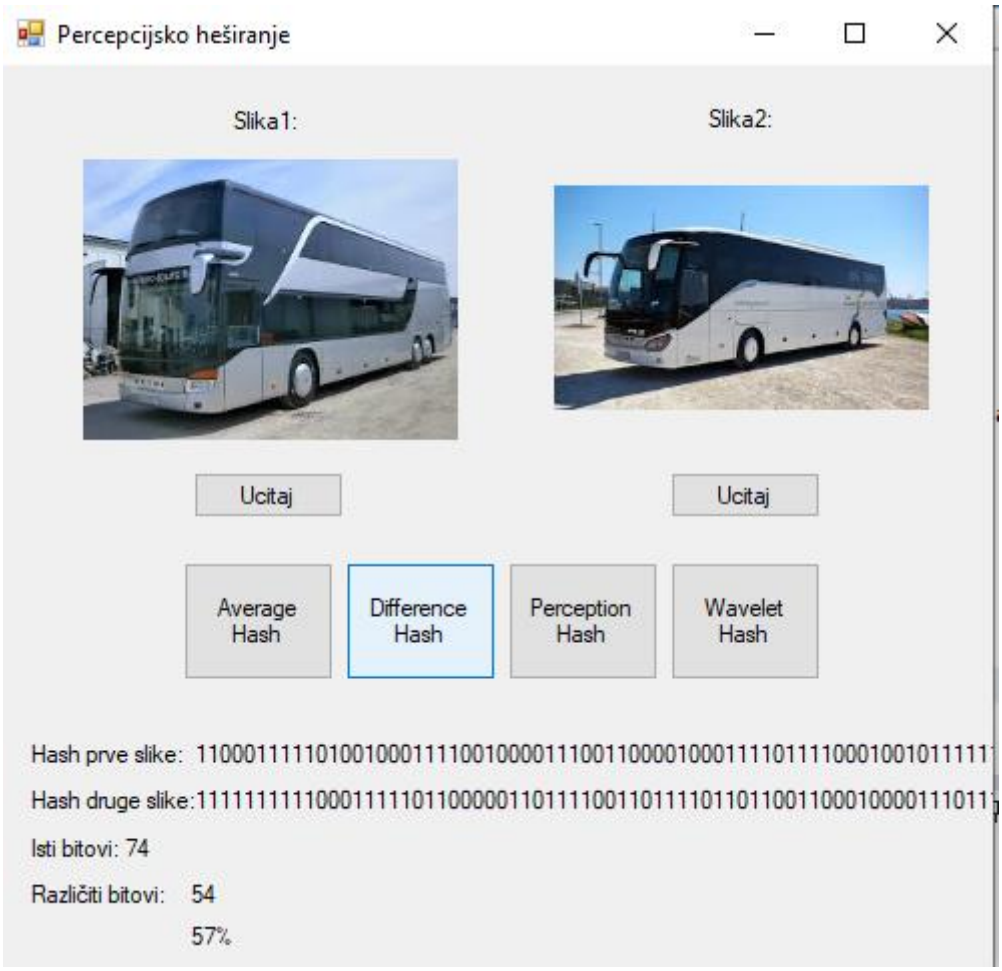
```
Bitmap grayscale;
            int x, y, avg, a, r, g, b;
            for (x = 0; x < pomocnaSlika.Width; x++)
            {
                for (y = 0; y < pomocnaSlika.Height; y++)
                {
                    Color bojaPiksela =
pomocnaSlika.GetPixel(x, y);
                    a = bojaPiksela.A;
                    r = bojaPiksela.R;
                    g = bojaPiksela.G;
                    b = bojaPiksela.B;
                    avg = (r + g + b) / 3;
                    pomocnaSlika.SetPixel(x, y,
Color.FromArgb(a, avg, avg, avg));
                }
            }
            grayscale = pomocnaSlika;
```

3. Kreiranje hasha uspoređivanjem susjednih polja

```
for(int redak = 0; redak < 8; redak++)
            {
                for(int stupac = 0; stupac < 8; stupac++)
                {
                    if (grayscale.GetPixel(stupac, redak + 1).R
>= grayscale.GetPixel(stupac, redak).R)
                    {
                        rHashVrijednost += "1";
                    }
                    else
                    {
                        rHashVrijednost += "0";
                    }
                }
            }
            for (int stupac = 0; stupac < 8; stupac++)
            {
                for (int redak = 0; redak < 8; redak++)
                {
                    if (grayscale.GetPixel(stupac+1, redak).R
>= grayscale.GetPixel(stupac, redak).R)
                    {
                        cHashVrijednost += "1";
                    }
                    else
```

```
                {
                    cHashVrijednost += "0";
                }
            }
        }
        PostaviHashVrijednost();
```

Na slici (Slika 5.) su prikazani rezultati hashiranja i uspoređivanja istih dviju fotografija pomoću Difference Hash algoritma. Možemo primijetiti da je postotak sličnosti 57%, što je manje od postotka izračunatim pomoću Average Hash algoritma.



Slika 5. Primjer rada Difference Hash algoritma

3.3.3. Usporedba slika pomoću Perception Hash algoritma

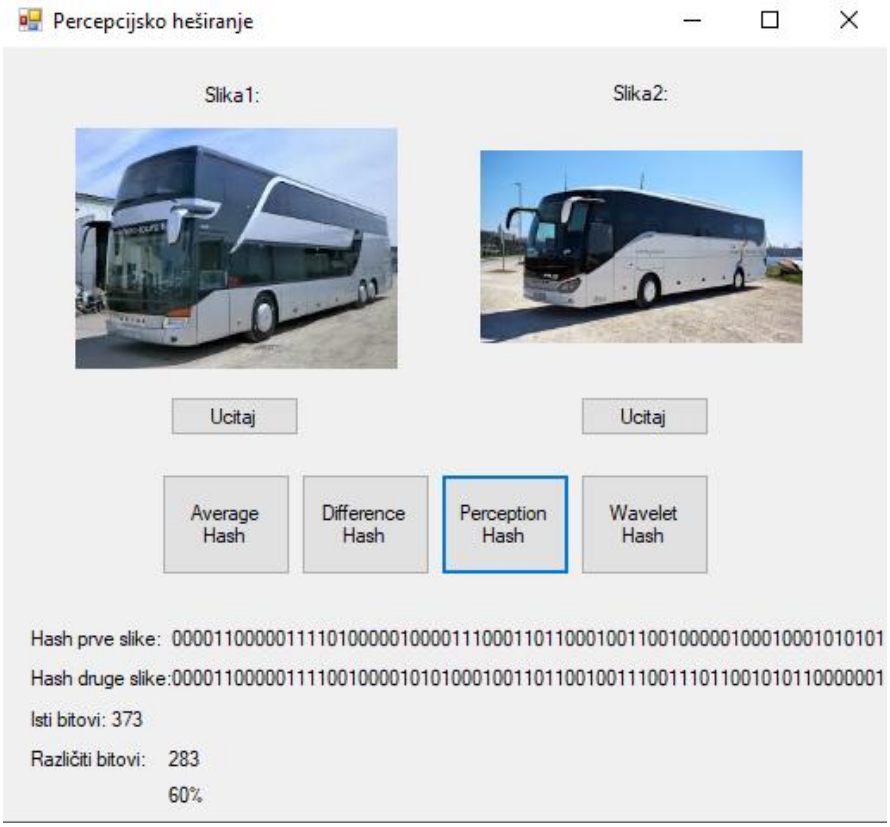
Nakon što se odaberu dvije fotografije koje se žele usporediti, klikom na gumb „Perception Hash“ izračunava se Perception hash za obje fotografije te se dobije postotak koliko je bitova jednako na istim pozicijama, odnosno slično fotografija. Perception hash nećemo izračunavati algoritmom koji prati korake koji su navedeni u poglavlju 2.3. Perception Hash, zbog toga što koristimo klase Shipwreck.Phash.Bitmaps i Shipwreck.Phash u kojima su sadržane funkcije za izračunavanje hasheva pomoću DCT-a. Pa programsko rješenje u kodu izgleda ovako:

```
public PerceptionHash(Bitmap bitmap)
        {
            this.prvobitniHashSlike =
ImagePhash.ComputeDigest(bitmap.ToLuminanceImage());
            byte[] hashSlikeBytes =
Encoding.ASCII.GetBytes(this.prvobitniHashSlike.ToString());
            this.hashVrijednost = GetBitsFromBytes(new
BitArray(hashSlikeBytes));
        }
```

Dok se sličnost slika izračunava sljedećim kodom :

```
public static double IzracunajSlicnostPercepcijskogHasha(PerceptionHash
pHashPrveSlike, PerceptionHash pHashDrugeSlike)
        {
            return ImagePhash.GetCrossCorrelation(pHashPrveSlike.GetDigest(),
pHashDrugeSlike.GetDigest()) * 100;
        }
```

Na slici (Slika 6.) su prikazani rezultati Perception Hash algoritma. Možemo primijetiti da svaki hash ima 656 bitova, broj tih bitova nije bilo moguće smanjiti jer smo koristili gotove klase. Također se može primijetiti da su prema Perception hashiranju slike slične 60%, što je između rezultata Average i Difference Hash algoritma.



Slika 6. Primjer rada Perception Hash algoritma

3.3.4. Usporedba slika pomoću Wavelet Hash algoritma.

Nakon što se odaberu dvije fotografije koje se žele usporediti, klikom na gumb „Wavelet Hash“ izračunava se Wavelet hash za obje fotografije te se dobije postotak koliko je bitova jednako na istim pozicijama, odnosno slično fotografija. Wavelet hash se izračunava na sljedeći način:

1. Smanjenje slike na 32x32 piksela

```
Bitmap slika = new Bitmap(bitmap, new Size(32, 32));
```

2. Smanjiti boje slike, odnosno bojanje slike u sive nijanse Provođenjem DWT-a, dobije se matrica DWT koeficijenata veličine 32x32, gdje je energija slike sadržana u nekoliko koeficijenata

```
Bitmap pomocnaBitmapa;
            int x, y, avg, a, r, g, b;
            for (x = 0; x < slika.Width; x++)
            {
                for (y = 0; y < slika.Height; y++)
                {
                    Color bojaPiksela = slika.GetPixel(x, y);
                    a = bojaPiksela.A;
                    r = bojaPiksela.R;
                    g = bojaPiksela.G;
```

```
                    b = bojaPiksela.B;
                    avg = (r + g + b) / 3;
                    slika.SetPixel(x, y, Color.FromArgb(a, avg, avg, avg));
                }
            }
            pomocnaBitmapa = slika;
            double[,] Crvena = new double[pomocnaBitmapa.Width,
pomocnaBitmapa.Height];
            double[,] Zelena = new double[pomocnaBitmapa.Width,
pomocnaBitmapa.Height];
            double[,] Plava = new double[pomocnaBitmapa.Width,
pomocnaBitmapa.Height];

            Color boja;
            for (int j = 0; j < pomocnaBitmapa.Height; j++)
            {
                for (int i = 0; i < pomocnaBitmapa.Width; i++)
                {
                    boja = pomocnaBitmapa.GetPixel(i, j);
                    Crvena[i, j] = (double)Scale(0, 255, -1, 1, boja.R);
                    Zelena[i,j] = (double)Scale(0, 255, -1, 1, boja.G);
                    Plava[i,j] = (double)Scale(0, 255, -1, 1, boja.G);
                }
            }

            FWT(Crvena, 4);
            FWT(Zelena, 4);
            FWT(Plava, 4);
```

Kod FWT funkcije :

```
   private void FWT(double[,] data, int iterations)
           {
               int rows = data.GetLength(0);
               int cols = data.GetLength(1);

               double[] row = new double[cols];
               double[] col = new double[rows];

               for (int k = 0; k < iterations; k++)
               {
                   for (int i = 0; i < rows; i++)
                   {
                       for (int j = 0; j < row.Length; j++)
                           row[j] = data[i, j];

                       FWT(row);

                       for (int j = 0; j < row.Length; j++)
                           data[i, j] = row[j];
                   }

                   for (int j = 0; j < cols; j++)
                   {
                       for (int i = 0; i < col.Length; i++)
                           col[i] = data[i, j];

                       FWT(col);

                       for (int i = 0; i < col.Length; i++)
                           data[i, j] = col[i];
                   }
```

```
                }
```

3. Smanjiti 32x32 matricu na gornju lijevu matricu veličine 8x8 piksela

```
for (int j = 0; j < pomocnaBitmapa.Height; j++)
            {
                for (int i = 0; i < pomocnaBitmapa.Width; i++)
                {
                    pomocnaBitmapa.SetPixel(i, j, Color.FromArgb((int)Scale(-1,
1, 0, 255, Crvena[i, j]), (int)Scale(-1, 1, 0, 255, Zelena[i, j]), (int)Scale(-
1, 1, 0, 255, Plava[i, j])));
                }
            }
            Bitmap bitmapPromjena = new Bitmap(pomocnaBitmapa, new Size(8, 8));
```

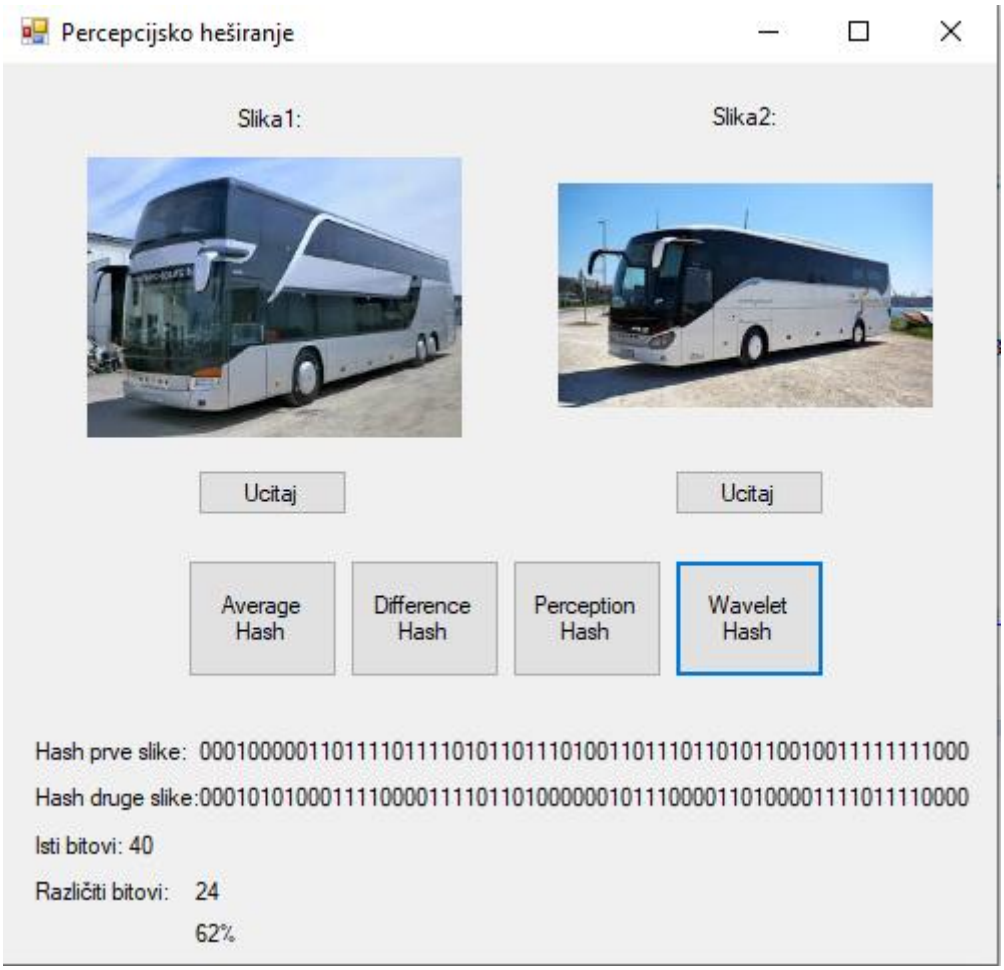
4. Izračunava se prosječna DWT vrijednost koeficijenata

```
int avgr = 0, avgg = 0, avgb = 0;
            for (x = 0; x < bitmapPromjena.Width; x++)
            {
                for (y = 0; y < bitmapPromjena.Height; y++)
                {
                    r = bitmapPromjena.GetPixel(x, y).R;
                    g = bitmapPromjena.GetPixel(x, y).G;
                    b = bitmapPromjena.GetPixel(x, y).B;
                    avgr += r;
                    avgg += g;
                    avgb += b;
                }
            }
            Color prosjekBoja = Color.FromArgb(avgr / 64, avgg / 64, avgb / 64);
```

5. Kreira se hash tako da se DWT koeficijenti uspoređuju sa prosječnom DWT vrijednošću te ukoliko je DWT koeficijent u matrici veći ili jednaki od prosjeka u hash se zapisuje 1, a ukoliko je manji zapisuje se 0.

```
for (x = 0; x < bitmapPromjena.Width; x++)
            {
                for (y = 0; y < bitmapPromjena.Height; y++)
                {
                    if (bitmapPromjena.GetPixel(x, y).GetBrightness() <
prosjekBoja.GetBrightness())
                        this.hashVrijednost += "1";
                    else
                        this.hashVrijednost += "0";
                }
            }
```

Na slici (Slika 7.) prikazani su rezultati Wavelet Hash algoritma. Možemo primijetiti da Wavelet hash ima 64 bitova te da je sličnost između fotografija 62%, a taj postotak se nalazim između Perception Hash (60%) i Average Hash (64%) algoritma.



Slika 7. Primjer rada Wavelet Hash algoritma

4. Zaključak

Percepcijsko hashiranja je algoritam bitno različit od kriptografskog hashiranja, koji čak i najmanjom promjenom piksela u bitmapi rezultira sličnom vrijednosti hasha, dok bi za razliku od korištenja kriptografskog hashiranja, vrijednost hasha se značajno promijenila u odnosu na original. Stoga možemo zaključiti da percepcijsko hashiranje je algoritam stvaranja "otiska prsta" nad multimedijskim sadržajem, koristeći različite algoritme za izradu istih. Najbolji primjer sve većeg korištenja percepcijskog hashiranja bi bila baza podataka koja bi pratila sadržaj zaštićen autorskim pravima te spremala hash vrijednosti proizvedene nekim od algoritama percepcijskog hashiranja te tako prilikom objave nekog sadržaja može pokrenuti provjeru hasha vrijednosti s novoobjavljene objave s podacima iz baze podataka te dobiti podatke o mogućem plagijatu dobivene u postotcima ili sličnim korelacijskim vrijednostima. Percepcijsko hashiranje uvelike ima značaj u digitalnoj forenzici, zbog svoje mogućnosti izrade hash vrijednosti nad digitalnim sadržajem te dobivanja korelacijskih vrijednosti u odnosu na ispitivani objekt. Daljnjim razvojem online medija i objavljivanja sadržaja na internetu dolazimo do značajnog interesa i velike implementacijske zone za korištenje algoritama percepcijskog hashiranja koji su djelotvorni i efikasniji od standardnih algoritama provjera multimedijskog sadržaja s povećim sadržajem zapisanog u bazi podataka.

Popis literature

[1] „Bertolami“, *Bertolami*, 14.012020. [Na internetu]. Dostupno na: http://bertolami.com/index.php?engine=blog&content=posts&detail=perceptual-hashing.

[2] „Average Hashing for Perceptual Image Similarity in Mobile Phone Application“, *Average Hashing for Perceptual Image Similarity in Mobile Phone Application*. [Na internetu]. Dostupno na: https://media.neliti.com/media/publications/92774-EN-average-hashing-for-perceptual-image-sim.pdf.

[3] „TESTING DIFFERENT IMAGE HASH FUNCTIONS“, *TESTING DIFFERENT IMAGE HASH FUNCTIONS*. [Na internetu]. Dostupno na: https://content-blockchain.org/research/testing-different-image-hash-functions/.

[4] „Visual tracking based on improved foreground detection and perceptual hashing“. [Na internetu]. Dostupno na: https://www.labxing.com/files/lab_publications/4962-1556263250-yQ2uUQ5A.pdf.

[5] „Evaluating Robustness of Perceptual Image Hashing Algorithms“, *Evaluating Robustness of Perceptual Image Hashing Algorithms*. [Na internetu]. Dostupno na: https://bib.irb.hr/datoteka/881114.Evaluating_Robustness_of_Perceptual_Image_Hashing_Algorithms.pdf.

Popis slika