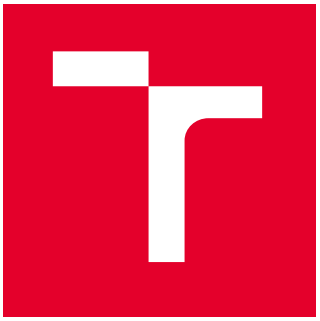
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2023 Bc. Peter Kopec



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

HASHOVÁNÍ OBRÁZKŮ POMOCÍ KOMPRIMAČNÍHO VZORKOVÁNÍ

IMAGE HASHING USING COMPRESSED SENSING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Peter Kopec**

AUTHOR

VEDOUCÍ PRÁCE — **prof. Mgr. Pavel Rajmic, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Peter Kopec ***ID:*** 211570

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Hashování obrázků pomocí komprimačního vzorkování

POKYNY PRO VYPRACOVÁNÍ:

Hashování obrázků je technika, kdy je k obrázku přiřazena relativně krátká číselná posloupnost, přičemž dva rozdílné obrázky by měly mít dva rozdílné hashe. Student se seznámí s problematikou hashování nejprve obecně, posléze konkrétněji pro oblast digitálních obrazů. Udělá si přehled o současných používaných metodách. Seznámí se s aktuální metodou dle zdroje [1] a implementuje tento postup v MATLABu, nebo v jiném dohodnutém programovacím jazyku. Implementuje rovněž jiné vybrané hashovací techniky a porovná vše na volně dostupných databázích obrázků. Srovnání bude zahrnovat studii odolnosti hashe vůči modifikaci obrazu komprimováním a vůči editaci obrazu. To proběhne na základě objektivních kritérií. Student neopomene srovnat metody z hlediska výpočetní náročnosti.

DOPORUČENÁ LITERATURA:

[1] Tang, Z., Zhang, H., Lu, S. a kol.: Robust image hashing with compressed sensing and ordinal measures. J Image Video Proc. 2020, 21.

[2] C. Lu a kol.: Robust mesh-based hashing for copy detection and tracing of images, 2004 IEEE International Conference on Multimedia and Expo, 2004

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: prof. Mgr. Pavel Rajmic, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Táto diplomová práca sa venuje analýze a implementácii hashovania obrázkov, ktoré vychádza z článku „Robust image hashing with compressed sensing and ordinal measures“[3]. Hashovanie obrázkov využíva tzv. vnímavé hashovacie metódy. Tieto metódy majú veľké využitie vo vede o počítačovom videní a vlastnosti týchto metód nám umožňujú porovnávať podobnosť hashovaných obrázkov a rozdeľovať tieto obrázky do skupín. Toto porovnávanie vieme využiť napríklad na vyhľadávanie obrázkov na internete z rôznych dôvodov. V teoretickej časti si povieme bližšie o vlastnostiach týchto hashovacích metód a popíšeme si spôsob hashovacej metódy podľa spomenutého článku, zameráme sa najviac na to čo to je kompresné vzorkovanie, saliency mapa a ako to dosiahneme. V praktickej časti si pomocou skriptovacieho jazyku Python pripravíme testovací dataset a implementujeme hashovaciu metódu podľa spomenutého článku. Následne túto hashovaciu metódu otestujeme na tomto datasete a na koniec ju porovnáme s inou hashovacou metódou.

KĽÚČOVÉ SLOVÁ

hashovanie, kompresné vzorkovanie, porovnávanie, Python, saliency mapa, vlastnosti

ABSTRACT

This thesis is devoted to the analysis and implementation of image hashing based on the article „Robust image hashing with compressed sensing and ordinal measures“[3]. Image hashing uses so-called perceptual hashing methods. These methods have great applications in computer vision science, and the properties of these methods allow us to compare the similarity of hashed images and classify these images into groups. We can use this comparison, for example, to search images on the Internet for various reasons. In the theoretical part, we will talk more about the properties of these hashing methods and describe the hashing method according to the mentioned paper, we will focus most on what is compressive sampling, saliency map and how we achieve it. In the practical part, we will prepare a test dataset using Python scripting language and implement the hashing method according to the mentioned article. Then we test this hashing method on this dataset and finally compare it with another hashing method.

KEYWORDS

comparison, compressed sensing, hashing, properties, Python, saliency map

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Peter Kopec
VUT ID autora:	211570
Typ práce:	Diplomová práca
Akademický rok:	2022/23
Téma závěrečnej práce:	Hashování obrázků pomocí komprimačního vzorkování

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi prof. Mgr. Pavlu Rajmicovi Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Táto práca sa venuje analýze a implementácií hashovaniu obrázkov, ktoré vychádza z článku „Robust image hashing with compressed sensing and ordinal measures“[3]. Hashovanie obrázkov je robustná metóda v oblasti porovnávania obrázkov. Táto vlastnosť má široké využitie vo vede o počítačovom videní. Túto a ďalšie vlastnosti si ukážeme, otestujeme a porovnáme.

Skladá sa z dvoch častí, teoretickej a praktickej.

Teoretická časť sa venuje teoretickému vysvetleniu pojmov ako sú hashovanie a kompresné vzorkovanie a opisuje rôzne metódy pre hashovanie a porovnávanie obrázkov. Taktiež opisuje bližšie niektoré funkcie ktoré sa používajú pri hashovaní a použitý dataset.

Praktická časť obsahuje funkčný skript, ktorý dokáže vytvoriť hash z obrázka pomocou techník v spomenutom článku a následne tento hash porovnať s iným hashom. Taktiež sa venuje úprave obrázkov pre dataset a testovaniu hashovacej metódy a rôznych parametrov pomocou tohoto datasetu.

Ciele práce:

1. Zoznámiť sa s kompresným vzorkovaním
2. Implementovať kompresné vzorkovanie
3. Zoznámiť sa s hashovaním obrázkov a metódami pre hashovanie
4. Implementovať postup hashovania obrázkov podľa spomenutého článku
5. Vytvoriť dataset obrázkov pre testovanie
6. Otestovať dataset a hashovaciu metódu
7. Porovnať našu hashovaciu metódu s inou metódou

1 Kompresné vzorkovanie – CS

V tejto kapitole si opíšeme, čo je to kompresné vzorkovanie – CS, ako funguje a aké má využitie.
Skratka CS pochádza z anglického názvu „Compress sampling“ čo vieme preložiť ako Kompresné vzorkovanie. Môže byť tiež referované ako „Compressed sensing“.

1.1 Čo to je CS

CS je technika spracovania signálu pomocou ktorej vieme signál skomprimovať a zároveň navzorkovať na nižšej frekvencii ako nám káže Nyquist teorém [9]. Nyquist teorém nám hovorí, že vzorkovacia frekvencia musí byť aspoň dva krát väčšia ako je frekvencia signálu, ktorý chceme vzorkovať. Ale ako môžeme vidieť na obrázku 1.1, dvojnásobok je absolútne minimum. V skutočnosti je potreba použiť ešte vyššiu vzorkovaciu frekvenciu.

1.2 Teória CS

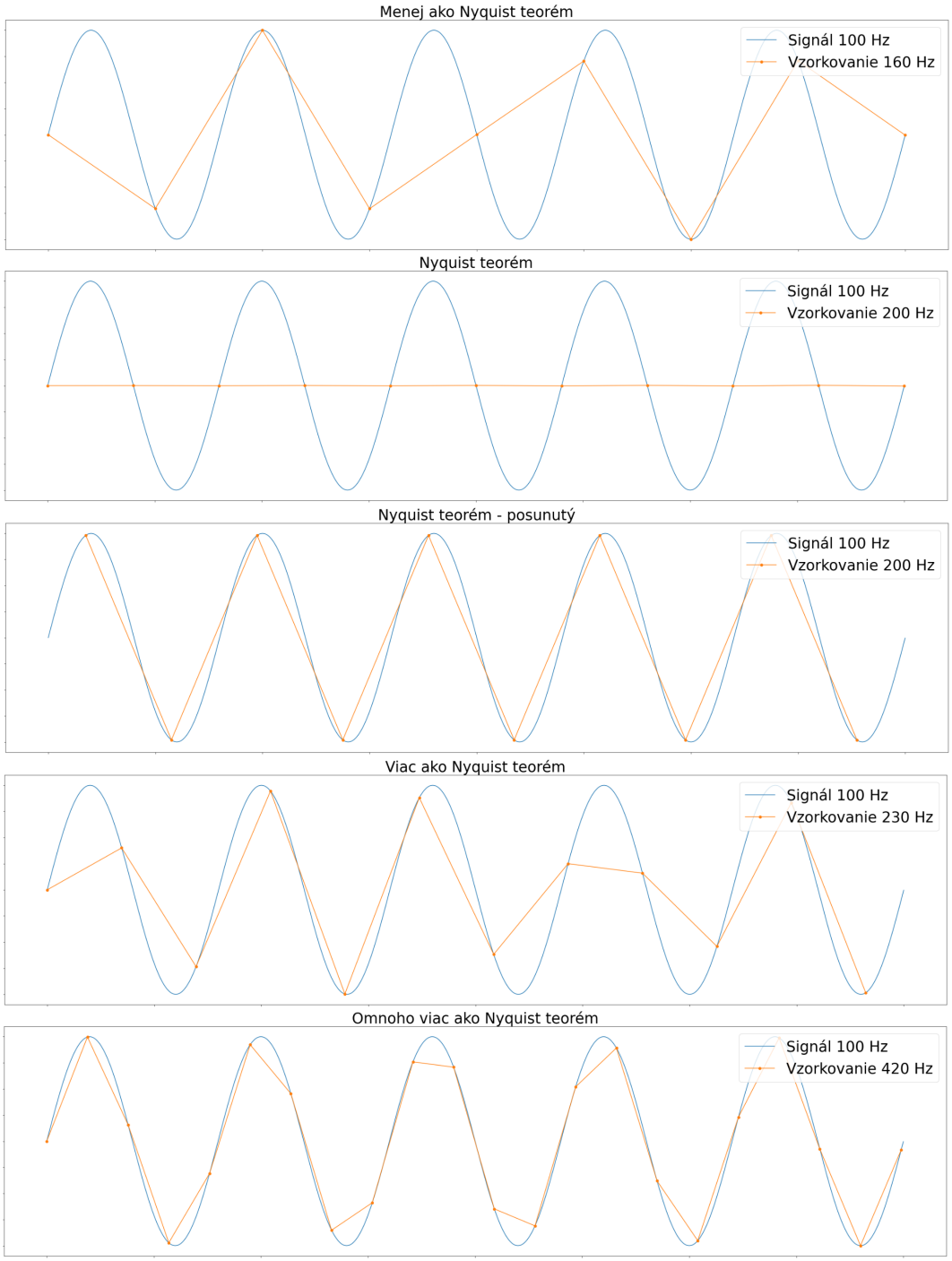
Kompresné vzorkovanie je postavené na lineárnej rovnici 1.1

$$Y = C \times \Psi \times S \tag{1.1}$$

Na obrázku 1.3 môžeme vidieť aplikovaný CS na obrázok 1.2, kde Y je skomprimovaný signál, C je vzorkovacia matica naplnená náhodnými 32bit hodnotami s Gaussovím rozdelením, Ψ je transformačná báza, na obrázku DCT a S je sparse vektor. $\Psi \times S$ = obrázok.
Z obrázkov si môžeme všimnúť rozmery jednotlivých častí. Náš obrázok X má rozmery x,y, v našom prípade $5 * 5$, vektor S má dĺžku x, Ψ ma rozmery $x * y$ na $x * y$ a matica C má rozmery $x * y$ a druhy rozmer záleží na veľkosti kompresii, dĺžke vektora Y. Pre väčšie obrázky môžeme naraziť na problémy s veľkými maticami, napríklad, vzorkovacia matica pre obrázok s veľkosťou 1920x1080, povedzmez, že chceme 10% vzoriek a naplníme ju opäť 32bit hodnotami, výsledok je približne 1,7TB, aj iba pre 1bit hodnoty je to stále približne 53GB.

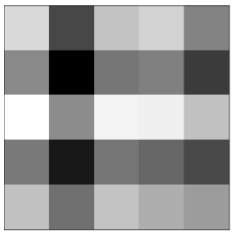
1.3 Rekonštrukcia dát

Pri znalosti vektoru Y a matice C je možné dáta zrekonštruovať do pôvodnej podoby, za predpokladu, že vektor Y má dostatočnú dĺžku, ak sú splnené nasledovné podmienky:

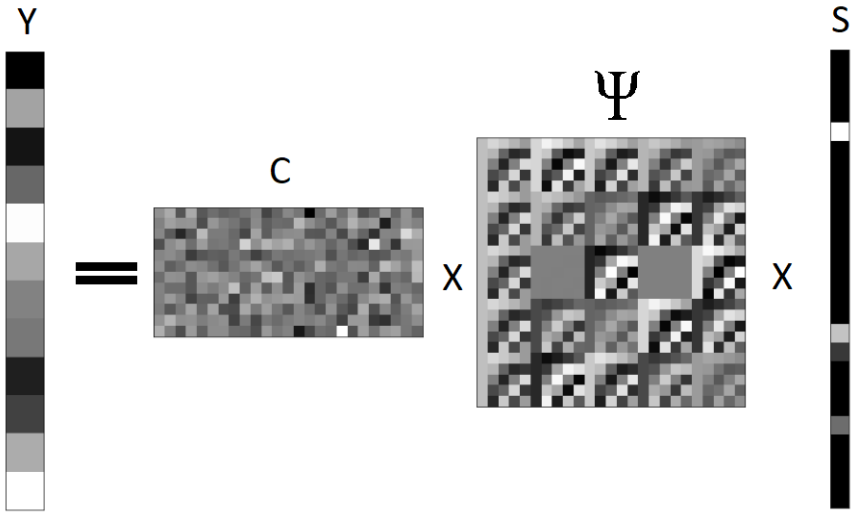


Obr. 1.1: Vplyv vzorkovacej frekvencie na odmeraný signál

1. Signál je sparse, čo znamená, že existuje nejaká dimenzia v ktorej je signál riedky, obsahuje nulové hodnoty. Inou dimenziou môžeme rozumieť napr. vyjadrenie signálu pomocou Diskrétnej kosínusovej transformácie (DCT).
2. Druhou podmienkou je inkoherencia meracej matice a DCT báze. To znamená, že riadky meracej matice sú ortogonálne k stĺpcom báze.



Obr. 1.2: Pôvodný obrázok X



Obr. 1.3: CS ukážka

Pre zrekonštruovanie potrebujeme vyriešiť rovnicu 1.1 pre najredšie S. Najredšie S nájdeme hľadaním riešenia pomocou L1 normy, ktorej vlasnosť je nájdenie najjednoduchších, najredších, riešení.

1.3.1 L1 vs L2

L2 norma je jedna z najviac bežne používaných noriem. Preto si v tejto časti definujeme a porovnáme L1 normu s L2 normou.

L1 norma, tiež známa ako „Manhattan Distance“ alebo „Taxicab norm“ je súčet absolútnych hodnôt veličín vo vektore. Je to najprirodzenejší spôsob merania vzdialenosti medzi vektormi, teda súčtom absolútneho rozdielu zložiek vektorov. V tejto norme majú všetky zložky vektora rovnakú váhu [20]. Vzorec na výpočet môžeme vidieť na vzorci 1.2

$$||X||_1 = \sum_{j=i}^{M} |X(j)| \tag{1.2}$$

L2 norma, tiež známa ako euklidovská norma. Je to najkratšia vzdialenosť na prechod z jedného bodu do druhého. Vzorec na výpočet môžeme vidieť na vzorci 1.3

$$||X||_2 = \sqrt{\sum_{j=i}^{M} X(j)^2} \tag{1.3}$$

Na obrázku 1.4 môžeme vidieť dva grafy. Na grafe vľavo vidíme dáta bez chyby a dve priamky, jedna vytvorená pomocou L1 normy a druhá pomocou L2 normy. V tomto prípade sa jedná o takmer identické priamky.

Na druhom grafe je umelo vnesená chyba do dát a môžeme jasne pozorovať, že priamka vytvorená v L1 rovine pretína maximálny možný počet bodov. Priamka vytvorená v L2 rovine je veľmi ovplyvnená dvoma chybnými hodnotami a nepretína takmer žiadne body. Chyba je v tomto prípade prehnaná pre zvýraznenie rozdielu.

Táto odchýlka je spôsobená tým, že L2 norma umocňuje hodnotu dát a tým pádom aj chyby zatiaľ čo L1 používa absolútnu hodnotu.

Ďalej si popíšeme vlastnosti noriem ako robustnosť, stabilita, zložitosť a riedkosť.

Robustnosť

Robustnosť je definovaná ako odolnosť voči odľahlým hodnotám v súbore údajov. Čím viac je model schopný ignorovať extrémne hodnoty v údajoch, tým je robustnejší.

Norma L1 je robustnejšia ako norma L2 čo môžeme vidieť na obrázku 1.4, ktorý bol opísaný vyššie.

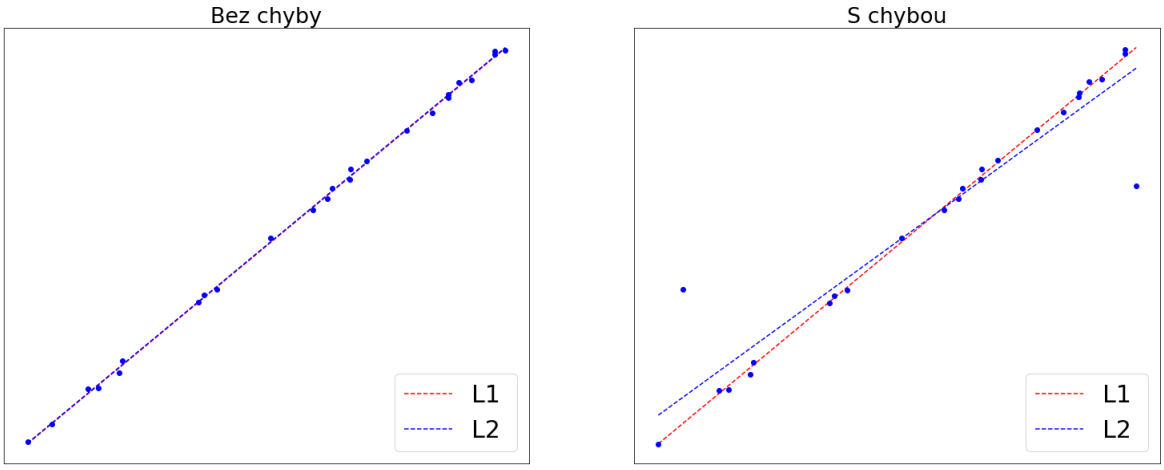
Stabilita

Stabilita v matematike je stav, kedy malá zmena na vstupe nevyvolá veľkú zmenu na výstupe.

Regulácia L1 využíva vzdialenosti Manhattan na dosiahnutie jedného bodu, takže existuje veľa trás, ktorými sa možno dostať do určitého bodu. Regulácia L2 využíva euklidovské vzdialenosti, ktoré nám povedia najrýchlejší spôsob, ako sa dostať k bodu. To znamená, že norma L2 má len 1 možné riešenie a tým pádom je stabilnejšia [12].

Výpočetná zložitosť

Pre L2 normu existuje analytické riešenie, čo znamená, že riešenie problému spracovávame v dobre porozumenej podobe, využíva efektívne algoritmy a vypočíta presné riešenie. Vďaka tejto dobre porozumenej podobe vieme túto normu vypočítať veľmi



Obr. 1.4: Regresná priamka pomocou L1 a L2 normy [**?**]

rýchlo a efektívne. L1 norma nemá analytické riešenie avšak vieme použiť niektoré algoritmy ktoré nám uľahčia výpočet [12].

Riedkosť

L2 regularizácia zmenšuje váhy iba na hodnoty blízke 0, namiesto toho, aby boli v skutočnosti 0. Na druhej strane, L1 regularizácia zmenšuje hodnoty na 0 [12]. Táto vlastnosť je žiadaná pre CS.

1.4 Využitie CS

Fakt, že komprimovateľný signál je možné efektívne odmerať použitím množiny nekoherentných meraní, ktorá je úmerná k jeho informačnej úrovni $S << n$, kde S sú nekoherentné merania a n množina všetkých meraní, má dôsledky ktoré sú ďalekosiahle a týkajú sa množstva možných aplikácií ako napr:

1. **Kompresia údajov** V niektorých situáciách transformačná báza Ψ môže byť pre kódovač neznáma alebo nepraktická na implementáciu pre kompresiu dát. Ale náhodnú meraciu maticu C však možno považovať za univerzálnu kódovaciu maticu, keďže nemusí byť navrhnutá s ohľadom na bázu Ψ. Táto univerzálnosť môže byť užitočná najmä pre distribuované zdroje, ako sú senzorové siete [13].
2. **Zber dát** V niektorých situáciách môže byt nepraktické alebo nemožné odmerať 100% dát, ktoré chceme. V takom prípade môžeme použiť meraciu maticu C na odmeranie dát. Za predpokladu, že sú splnené podmienky pre rekonštrukciu sme schopní získať všetky dáta za kratšiu dobu merania, samozrejme

za výpočtovú cenu neskôr. Jeden z príkladov využitia je magnetická rezonancia [14].

3. **Kanálové kódovanie** Princípy CS (sparsitosť, náhodnosť a konvexná optimalizácia) možno upraviť a aplikovať na návrh kódov na rýchlu opravu chýb a ochranu pred chybami počas prenosu [14].

2 Hashovanie obrázkov

V tejto kapitole si povieme čo to je hash, popíšeme si metódy pre vytváranie hashov z obrázkov, ich výhody, nevýhody a využitie.

2.1 Čo to je hashovanie

Hashovanie je vytváranie odtlačku o fixnej veľkosti z dát s rôznymi rozmermi. Metódy pre hashovanie obrázkov, niekedy nazývané vnímavé hashovanie, po anglicky „perceptual hashing“, sú lokálne senzitívne hashe alebo zkratou LSH. LSH je algoritmická technika, ktorá hashuje podobné vstupy do jednej skupiny s vysokou pravdepodobnosťou. Množstvo týchto skupín je omnoho menšie ako množstvo vstupných kombinácií. Inak povedané, táto technika má vysokú mieru stability takže malá zmena na vstupe vyvolá malú zmenu na výstupe. Tieto metódy sú rozdielne od metód použitých pre kryptografické hashe ktoré využívajú lavínového efektu. Lavínový efekt zaručuje veľmi nízku mieru stability

2.2 Využitie

Hashovanie obrázkov sa využíva na porovnávanie alebo na zgrupovanie podobných obrázkov. Toto vieme využiť na hľadanie obsahu chráneným autorskými právami, na hľadanie nelegálneho obsahu alebo duplicitných dát. Niektoré implementácie si popíšeme v sekcii 2.2.2.

2.2.1 Výhody a nevýhody

Porovnávanie obrázkov na základe hashu má svoje výhody aj nevýhody. Začneme výhodami hashov:

- **Úložisko** Databáza hashov zaberá výrazne menej priestoru na úložisku ako databáza obrázkov.
- **Rýchlosť** Hľadanie hashov, znakových reťazcov, je rýchlejšie ako hľadanie obrázkov.
- **Anonymita** Hľadaný aj uložený obrázok je len reťazec znakov a už neposkytuje vizuálny úžitok. Veľmi vhodná vlastnosť pre PhotoDNA nakoľko ich cieľom sú nelegálne obrázky [5].

Na druhej strane, z faktu, že hash vytvára výstup o konštantnej veľkosti, čo znamená, že máme limitovaný počet unikátnych výstupov tak môže nastať kolízia. Kolízia znamená, že dva rozdielne obrázky budú mať podobný alebo rovnaký hash.

Pravdepodobnosť kolízie záleží na veľkosti hashu, použitom algoritme pre vytvorenie hashu a na spôsobe a citlivosti porovnávania ktoré zvolíme.

2.2.2 Implementácie

Hashovanie obrázkov využíva niekoľko aplikácií. O niektorých z nich si povieme nižšie.

- **TinyEye** TinyEye je vyhľadávací nástroj, ktorý na základe hashu obrázku dokáže nájsť jeho kópie na internete. Majú databázu vyše 56 miliard odtlačkov [4].
- **PhotoDNA** Nástroj od Microsoftu ktorého úlohou je vyhľadávanie a odstraňovanie nelegálnych obrázkov na základe hashu obrázku z internetu [5]
- **Undouble** Jedná sa o python knižnicu ktorá implementuje rôzne hashovacie metódy za účelom vyhľadávania podobných obrázkov v súborovom systéme alebo priečinku [3].

2.3 Hashovanie pomocou CS

V tejto časti si opíšeme spôsob hashovania obrázkov navrhnutý v článku „Robust image hashing with compressed sensing and ordinal measures“ [3]. Tento postup sa neskôr snažíme implementovať, zreprodukovať a porovnať s ďalšími metódami.
Tvorenie hashu podľa tohto postupu sa skladá z následujúcich krokov:

1. Obrázok je pomocou bikubickej interpolácie upravený na fixnú veľkosť 512×512 pixlov.
2. Vypočíta sa saliency mapa pomocou metódy navrhnutou Itti et. al. v článku „A Model of Saliency-Based Visual Attention for Rapid Scene Analysis“ [8].
3. Výstup kroku 1 je prevedený do čiernobielej podoby a pomocou Canny operator sú detekované kraje v obrázku.
4. Vypočíta sa vážená reprezentácia obrázku vynásobením výstupu z kroku 2 a 3 do jedného obrázku.
5. Obrázok je rozdelený na 32 blokov i o veľkosti 64×64 pixlov. Na jednotlivé bloky je aplikované kompresné vzorkovanie s výstupom Y_i. Presná metóda a množstvo kompresie nie je špecifikované. Pre ukázanie fluktuácie elementov vo vektore Y_i, je použitý rozptyl, ktorý sa vypočíta pomocou vzorca 2.1. Kde $Y_i(j)$ je hodnota na pozícií j vektoru Y_i. m je priemer vektoru Y_i, ktorý sa vypočíta pomocou vzorca 2.2.

$$v_i = \frac{1}{M-1}\sum_{j=1}^{M}[Y_i(j) - m_i]^2 \tag{2.1}$$

$$m_i = \frac{1}{M}\sum_{j=1}^{M} Y_i(j) \tag{2.2}$$

Po vypočítaní fluktuácie pre všetky bloky dostaneme vektor v 2.3, ktorý pozostáva z 32 hodnôt o veľkosti 32bitov.

$$v = [v_1, v_2, \ldots, v_{32}]^T \tag{2.3}$$

6. Posledným krokom vytvárania hashu je zmena na ordinálne dáta. Táto zmena nám umožní vymeniť 32bit hodnoty za celé čísla ktorých veľkosť záleží od dĺžky vektora. V tomto prípade sa jedná o hodnoty 1 až 32. Vďaka tejto zmene vieme zmenšiť veľkosť hashu na úložisku a vylepšiť jeho klasifikačné vlastnosti.

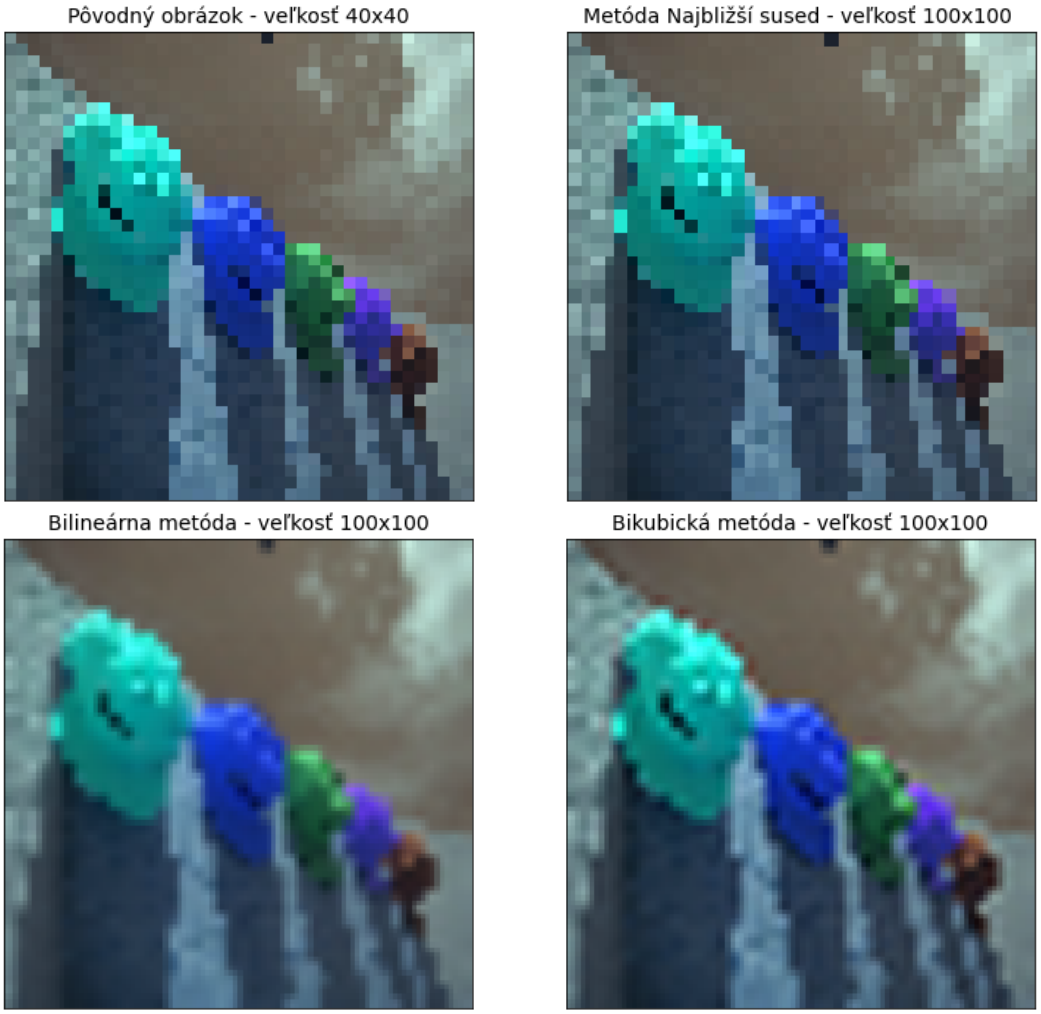
Na koniec, navrhnutý spôsob pre porovnávanie podobnosti dvoch hashov je použitá L2 norma dvoch hashov, ktorú vypočítame pomocou vzorca 2.4 kde $h_1(j)$ a $h_2(j)$ sú elementy hashov h_1 a h_2 na pozícií j. Hashe dvoch podobných obrázkov by mali mať malú veľkosť d, L2 normy. Identické obrázky majú nulovú hodnotu d.

$$d(h_1, h_2) = \sqrt{\sum_{j=1}^{M}[h_1(j) - h_2(j)]^2} \tag{2.4}$$

2.3.1 Interpolácia

Interpolácia je metóda predpokladania neznámych hodnôt na základe známych hodnôt. Interpolácia obrázkov sa používa pre úpravu ich rozlíšenia. Na interpoláciu obrázkov sa prevažne používa jedna z nasledujúcich metód: najbližší sused, bilineárna interpolácia alebo bikubická interpolácia. Metóda najbližšieho suseda je najjednoduchšia a najrýchlejšia z uvedených metód. Nevytvára žiadne nové nové dáta keďže použije hodnotu ktorá má najmenšiu vzdialenosť k novej. Táto metóda môže spôsobiť blokové artefakty. Je vhodná pre retro hry, ktoré používajú tzv. blokovú grafiku.

Bilineárna metóda predpokladá novú hodnotu na základe váženého priemeru štyroch najbližších pixlov. Jedná sa o najbežnejšiu metódu.
Bikubická metóda predpokladá novú hodnotu na základe váženého priemeru šestnástich najbližších susediacich pixelov. Výstup tejto metódy je hladší a má lepšiu reprezentáciu vstupného obrázka za cenu zvýšenej zložitosti. Táto metóda taktiež spôsobuje „overshoot“, nad tyrkysovým klobúkom 2.1, v signálovom spracovaní to znamená, že hodnota signálu presiahne požadovanú, priemernú, hodnotu. Táto vlastnosť môže niekedy zvýšiť vnímanú ostrosť. [10]



Obr. 2.1: Ukážka interpolácie

2.3.2 Saliency map

Vo vede o počítačovom videní, saliency mapa je obraz, ktorý zvýrazňuje oblasť, na ktorú sa ľudské oči zameriavajú ako prvé. Cieľom saliency mapy je odrážať stupeň dôležitosti pixlu pre ľudský vizuálny systém. Vizuálny systém človeka sa vie zamerať na tieto dôležité časti a vďaka tomu vie relatívne rýchlo a s malo námahou interpretovať rôzne zložité scény. Umelo vytvorené saliency mapy sa môžu líšiť od skutočných máp vytvorenými ľudmi. Ľudia tieto dôležité časti nachádzajú za pomoci rôznych vizuálnych máp ktoré sa nachádzajú v mozgu.

Jedna z metód pre výpočet tejto mapy počítačom, ktorá je inšpirovaná ľudským vizuálnym systémom je metóda podľa článku „A Model of Saliency-Based Visual Attention for Rapid Scene Analysis“ [8]. Pre výpočet saliency mapy, táto metóda najprv vypočíta 42 rôznych tzv. „feature“ máp. Kde 6 máp je pre intenzitu, 12 pre farbu a 24 pre orientáciu. Tieto mapy sú následne skombinované do troch tzv.

„conspicuity maps“, ktoré spoločne vytvoria saliency mapu.

Ďalej si bližšie opíšeme jednotlivé kroky tejto metódy a začneme intenzitou.

Intenzita

Vstupný farebný obrázok je prevedený na intenzitu I pomocou vzorca 2.5, kde r, g, b sú farebné kanály obrázku, červený, zelený a modrý. Na túto intenzitu sa aplikujú Gaussove pyramídy výsledkom čoho je 9 obrázkov ktorých veľkosť sa postupne znižuje, od mierky 1:1 čo je pôvodná veľkosť až ku 1:256 zmenšenine. Tieto menšiny majú postupne silnejšie a silnejšie rozmazanie. Vytvorí sa 6 rôznych kombinácií týchto obrázkov ktoré sa medzi odčítajú a získame 6 máp pre intenzitu. [8]

$$Intenzita(I) = (r + b + g)/3 \tag{2.5}$$

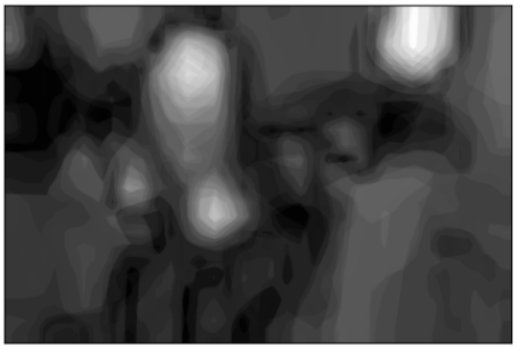
Obr. 2.2: Zložka intenzity obrázku podľa Itty

Farba

Farebné kanály r, g, b sú normalizované podľa I aby sa oddelil odtieň od intenzity. Nakoľko ale zmeny odtieňu pri nízkej intenzite nie sú človekom veľmi vnímané, normalizácia je aplikovaná len na miesta kde I je väčšie ako 1/10 maxima. Ostatné lokácie majú nulovú hodnotu r, g, b. Následne sú vytvorené 4 farebné kanály R, G, B, Y. Červený $R = r - (g + b)/2$, zelený $G = g - (r + b)/2$, modrý $B = b - (r + g)/2$ a žltý $Y = (r + b)/2 - (r - g)/2 - b$. Na každý farebný kanál sú štyrikrát aplikované Gaussove pyramídy. Následne pomocou vzorcov 2.6 a 2.7 je vytvorených 6 máp pre RG a 6 máp pre BY kde c, s sú indexy Gaussovej pyramídy. [8]

$$RG(c, s) = |(R(c) - G(c)) - (G(s) - R(s))| \tag{2.6}$$

$$BY(c, s) = |(B(c) - Y(c)) - (Y(s) - B(s))| \tag{2.7}$$

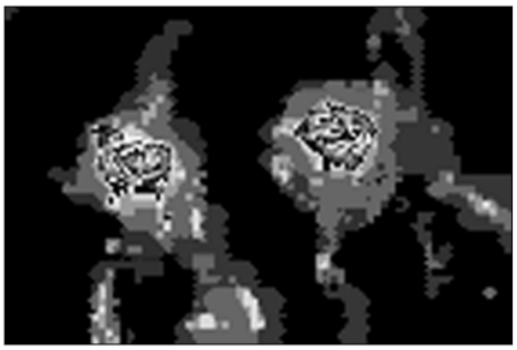


(a) BY farebná zložka obrázku podľa Itty

(b) RG farebná zložka obrázku podľa Itty

Orientácia

Posledný súbor máp je vytvorených pre orientáciu za pomoci Gaborových filtrov. Tieto filtre sa aplikujú v 4 rôznych orientáciách na 9 obrázkov intenzity ktoré boli vytvorené v časti 2.3.2. Následne sú opäť vytvorené dvojice ako v prípade intenzity, ale tentokrát sú použite obrázky na ktoré bol aplikovaný Gaborov filter. Tieto dvojice sú od seba odčítané a získame 24 máp pre orientáciu. [8]



Obr. 2.4: Zložka orientácie obrázku podľa Itty

Sčítanie a normalizácia

Pre získanie výslednej saliency mapy z týchto 42 máp, je potreba všetky tieto mapy skombinovať do jednej. Nakoľko ale tieto mapy majú rôzne rozsahy dát a môžu obsahovať šum je potreba tieto mapy normalizovať. Táto normalizačná funkcia N uprednostňuje mapy, ktoré majú menší počet výraznejších vrcholov a zároveň potlačuje mapy, ktoré majú veľa porovnateľných vrcholov. Túto vlastnosť funkcia dosiahne tak že, nájde globálne maximum mapy M a vypočíta priemer $\overline{m}$ z lokálnych maxím danej mapy. Následne je celá mapa vynásobená hodnotou $(M-\overline{m})^2$. Výsledné skombinovanie prebehne pomocou vzorca 2.8 kde N je spomenutá normalizačná funkcia a $\overline{I}, \overline{C}, \overline{O}$ sú sčítané conspicuity mapy pre intenzitu, farbu a orientáciu.

$$S = 1/3(N(\overline{I}) + N(\overline{C}) + N(\overline{O})) \tag{2.8}$$

(a) Obrázok použitý pre ukážku (b) Saliency mapa obrázku podľa Itty

2.3.3 Canny Edge detection

Canny edge detection je algoritmus na detekciu hrán. Tento algoritmus pozostáva z niekoľkých krokov.

1. **Noise Reduction** Pomocou Gaussovho filtra je odstránený šum z obrázka.
2. **Finding Intensity Gradient of the Image** Po odstránení šumu je obrázok filtrovaný v horizontálnej G_x a vertikálnej G_y orientácii pomocou Sobel kernel. Z týchto dvoch obrázkov sa vypočíta Edge_Gradient pomocou vzorca 2.9 a Angle pomocou vzorca 2.10. Smer prechodu (gradient) je vždy kolmý na hrany. Je zaokrúhlená do jedného zo štyroch uhlov reprezentujúcich vertikálny, horizontálny a dva diagonálne smery.

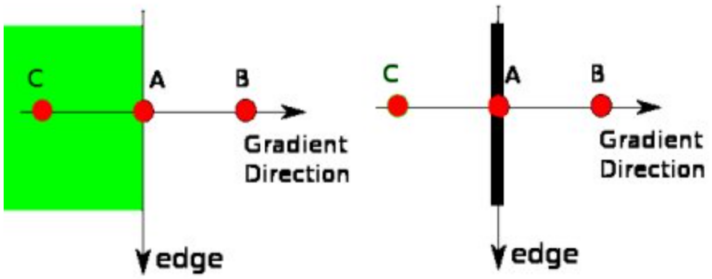
   $$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2} \tag{2.9}$$

   [11]

   $$Angle(\Theta) = tan^{-1}(\frac{G_y}{G_z}) \tag{2.10}$$

   [11]
3. **Non-maximum Suppression** Po získaní magnitúdy a smeru gradientu sa vykoná úplné skenovanie obrázku, aby sa odstránili všetky nežiaduce pixle, ktoré nemusia predstavovať okraj. Na tento účel sa pri každom pixeli kontroluje, či ide o lokálne maximum v jeho susedstve v smere gradientu. Pozri obrázok 2.6 kde bod A je na okraji (v zvislom smere). Smer prechodu je kolmý na okraj. Body B a C sú v smere sklonu. Takže bod A sa kontroluje s bodom

B a C, aby sa zistilo, či tvorí lokálne maximum. Ak áno, zvažuje sa pre ďalšiu fázu, v opačnom prípade sa potlačí (vynuluje sa).



Obr. 2.6: Non-maximum Suppression [11]

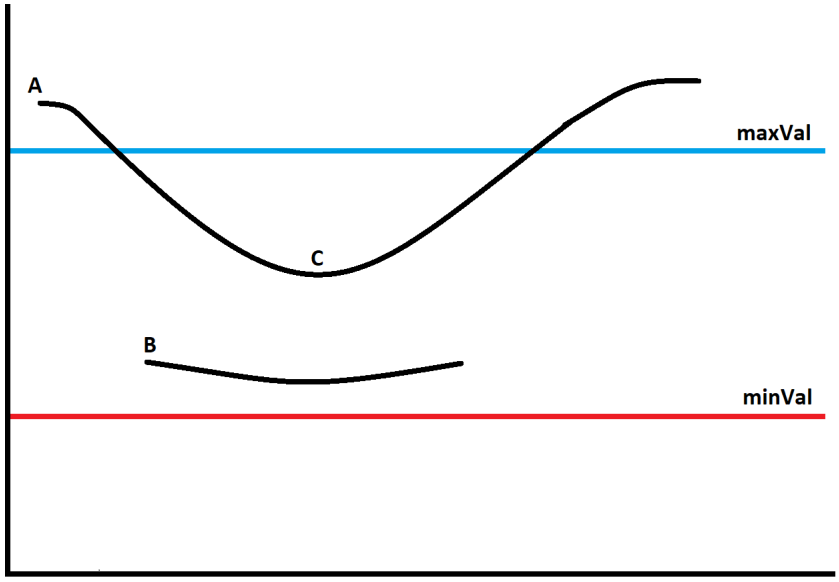
4. **Hysteresis Thresholding** Táto fáza rozhoduje o tom, ktoré hrany sú skutočne hrany a ktoré nie. Na to potrebujeme dve prahové hodnoty, minVal a maxVal. Akékoľvek hrany s gradientom intenzity väčším ako maxVal budú určite hranami a tie pod minVal určite nebudú hranami, takže sa zahodia. Tie, ktoré ležia medzi týmito dvoma prahmi, sú klasifikovaní ako hrany alebo nie hranami na základe ich konektivity. Ak sú pripojené k pixlom s „istým okrajom“, považujú sa za súčasť hrán. V opačnom prípade sú tiež vyradené. Pozri obrázok 2.7 kde hrana A je nad maxVal, takže sa považuje za „istú hranu“. Hoci hrana C je pod maxVal, je spojená s hranou A, takže sa to tiež považuje za platnú hranu a dostaneme celú krivku. Ale hrana B, aj keď je nad minVal a je v rovnakej oblasti ako hrana C, nie je pripojená k žiadnej „istej hrane“, takže sa zahodí. Preto je veľmi dôležité, aby sme podľa toho vybrali minVal a maxVal, aby sme dosiahli správny výsledok.
   Táto fáza tiež odstraňuje šum malých pixlov za predpokladu, že okraje sú dlhé čiary [11].

2.3.4 Ordinálne dáta

Ordinálne dáta majú široké využitie v rôznych aplikáciach ako sú video podpisy, rozpoznávanie dúhoviek v oku, rozpoznávanie tváre vďaka ich robustnosti a kompaktným vlastnostiam. Vo všeobecnosti, ordinálne dáta prvku v sekvencii môžeme vypočítať nasledujúcim postupom.

- Prvky usporiadame do novej sekvencie vo vzostupnom poradí.
- Pre každý prvok z pôvodnej sekvencie nájdeme jeho pozíciu v zoradenej sekvencii a táto pozícia je jeho nová reprezentujúca pozícia.

Ukážku ordinálnych dát môžeme vidieť na tabuľke 2.1. Na vrchu tabuľky môžeme vidieť pôvodnú sekvenciu ktorá obsahuje hodnoty 1 až 14 a na spodku ordinálne



Obr. 2.7: Hysteresis Thresholding [11]

dáta ktoré reprezentuje danú sekvenciu s hodnotami 1 až 10 [3].

Tab. 2.1: Ukážka ordinálneho merania

Pozícia	1	2	3	4	5	6	7	8	9	10
Pôvodná sekvencia	2	8	6	4	9	12	3	10	14	1
Zoradená sekvencia	1	2	3	4	6	8	9	10	12	14
Oridnálne meranie	2	6	5	4	7	9	3	8	10	1

2.4 Ďalšie metódy hashovania

V tejto sekcii si povieme o ďalších používaných metódach na vytváranie hashov, ktoré sú implementované v python knižnici ImageHash [19].

- **Average hash** Obrázok je najprv prevedený do čiernobielej verzie, následne prebehne redukcia veľkosti obrázka. Veľkosť je závislá od požadovanej dĺžky hashu.

  Z tohto upraveného obrázka vypočítame priemernú hodnotu pixelov, následne porovnáme každý pixel s priemernou hodnotou. Ak je hodnota pixlu väčšia alebo rovná ako priemer dostane hodnotu 1 ak menšia tak 0. Nakoniec pole hodnôt prevedieme na vektor [1].
- **Perceptual hash** Na čiernobielu verziu obrázka aplikujeme diskrétnu kosínusovú transformáciu (DCT) ktorá prevedie nízke frekvencie do ľavého horného

rohu. Z tohto rohu vezmeme požadovaný počet pixelov, podľa veľkosti hashu. Každý pixel porovnáme s mediánom, ak je hodnota pixlu väčšia tak dostane 1 inak 0. Nakoniec pole prevedieme na vektor [1].

- **Difference hash** Obrázok je prevedený do čiernobielej podoby, veľkosť je opäť upravená podľa veľkosti hashu. Následne sa porovnáva pixel X s pixlom X+1, ak je pixel X menší ako bit X+1 dostane hodnotu 1 inak hodnotu 0. Nakoniec je pole prevedené do vektoru [1].
- **Haar wavelet hash** Na čierno bielu verziu obrázka v požadovanej veľkosti aplikuje Haar wavelet transformáciu, následne porovná každý pixel s mediánom, ak je pixel väčší ako medián dostane 1 ináč 0. Nakoniec pole prevedie do vektoru [1].
- **Daubechies wavelet hash** Táto metóda je podobná Haar wavelet metóde, jediný rozdiel je v použitej transformácií. Ako môžeme vidieť z názvu, v tejto metóde sa používa Daubechies wavelet transformácia.
- **Crop-resistant hash** Na čierno biely obrázok je aplikovaná transformácia podobnej watershed transformácií. Táto transformácia je schopná v obrázku nájsť objekty a následne pre najväčšie objekty sa vypočíta vlastný hash. Výstupom tejto metódy je hash pre každý objekt namiesto jedného hashu pre celý obrázok. [7].

2.5 Iné metódy porovnávania obrázkov

V tejto časti si opíšeme niektoré metódy pre porovnávanie obrázkov z článku Ïmage Comparison Methods & Tools"[18]. Niektoré z týchto metód by sme mohli považovať za hash nakoľko vytvárajú odtlačok obrázku na základe rôznych algoritmov.

- **Pixel by pixel** Táto metóda porovnáva dva obrázky pixel po pixli, vezme pixel na pozícií X,Y z obrázka A a pixel na pozícií X,Y z obrázka B a porovná ich farbu. Táto metóda podporuje niekoľko vstupných parametrov. Jeden z nich je Color Tolerance, tento parameter nám určuje ako veľmi podobné dva pixely musia byť pre vyhodnotenie zhody. Ďalší parameter je Transparent Color, táto možnosť vezme farbu prvého pixely a všetky pixle s touto farbou nebudú mať efekt na výslednú zhodu obrázkov. [18]
- **Color Coherence Vectors and joint histograms** Každý pixel v danej farebnej skupine klasifikujeme ako koherentný alebo nekoherentný na základe toho, či je alebo nie je súčasťou veľkej podobne sfarbenej oblasti. Vektor koherencie farieb (CCV) ukladá počet koherentných verzus nekoherentných pixlov s každou farbou. Oddelením koherentných pixlov od nekoherentných pixlov poskytujú CCV jemnejšie rozlíšenie ako farebné histogramy. Farebné histogramy

sú široko používané na získavanie obrázkov na základe obsahu kvôli ich účinnosti a robustnosti. Farebný histogram však zaznamenáva iba celkové zloženie farieb obrázka, takže obrázky s veľmi odlišným vzhľadom môžu mať podobné farebné histogramy. Na tento účel máme alternatívu k farebným histogramom nazývanú „joint histogram“, ktorý zahŕňa dodatočné informácie bez toho, aby bola obetovaná robustnosť farebných histogramov. Joint histogram vytvoríme výberom množiny lokálnych pixelov a vytvorením viacrozmerného histogramu. Každý záznam v joint histograme obsahuje počet pixelov na obrázku, ktoré sú opísané konkrétnou kombináciou hodnôt vlastností. [18]

- **Feature-Based** V počítačovom videní vlastnosť(feature) obrázka je matematické vyjadrenie nespracovaných dát obrázku. Vo všeobecnosti, porovnávanie vlastností je efektívnejšie a presnejšie ako porovnávanie nespracovaných obrázkov. Táto metóda sa skladá z troch krokov. Z detekcie vlastností, ako napríklad hrany obrázka. Druhý krok je opis detekovaných vlastností a nakoniec zo samotného porovnávania, napríklad pomocou euklidovej vzdialenosti. [18]
- **Defect Detection Algorithm for Gray level Digital Images using Median Filters Gabor Filter and ICA** Operácia extrakcie prvkov sa vykonáva v každom riadku obrázka, aby sa získali presné výsledky o vlastnostiach pixlov. Algoritmus sa porovnáva s existujúcimi metódami, ako je metóda analýzy nezávislých komponentov (ICA) a metóda optimálneho Gaborovho filtra. Tento algoritmus je rýchly a jednoduchý a využíva extrakciu funkcií a segmentáciu na identifikáciu defektov v digitálnych obrázkoch s úrovňou šedej. Chyby je možné identifikovať pomocou podrobných matíc, ktoré pozostávajú zo stredových, maximálnych a minimálnych bodov. Na detekciu defektov textúry možno použiť vlnové subpásmo a optimalizované filtre Gabor. Gabor filtre môžu byť nasadené na detekciu defektov tkaniny pomocou Bernoulliho pravidla kombinácie. Segmentácia používaná na detekciu defektov v systéme výroby odevov. Metóda odčítania obrazu používaná v oblasti detekcie defektov vzorovaných textílií. Minimálne, maximálne a stredné hodnoty sa vypočítajú pre každý riadok obrázka, aby sa zarámoval vektor prvku. Vysokofrekvenčné zložky sa eliminujú pomocou strednej hodnoty každého radu a nakoniec sa obraz nízkofrekvenčnej zložky spolu so strednou hodnotou každého radu použije na detekciu chybných bodov s náhlou intenzitou od pôvodného obrazového prvku alebo náhlou odchýlkou od strednej hodnoty. [18]

3 Dataset

Pre testovanie vlastností hashovacích metód bol vytvorený dataset obrázkov z ktorých budeme následne robiť hashe. Ako základ pre náš dataset je použitá databáza 24 nekomprimovaných obrázkov od kodaku [2].

3.1 Modifikácia obrázkov

Pre otestovanie robustnosti a diskriminácie boli na obrázkoch prevedené zmeny. Všetky zmeny s parametrami môžeme vidieť v tabuľke 3.1

Tab. 3.1: Parametre úprav obrázkov

Úprava	Parameter										Jednotka
Jas	0,50	0,75	1,5	2							Faktor
Kontrast	0,5	0,75	1,5	2							Faktor
Gama	0,75	0,9	1,1	1,25							Faktor
Veľkosť	0,5	0,75	0,9	1,1	1,5	2					Násobok
Gaus. filter	0,1	0,4	0,5	0,6	0,7	0,8	0,9	1			Odchýlka
S&P	1	2	3	4	5	6	7	8	9	10	PPT
Speckle	1	2	3	4	5	6	7	8	9	10	Rozptyl 10^{-3}
Jpeg	30	40	50	60	70	80	90	100			Kvalita %
Vodoznak											
Orezanie	0,85	0,95	1,05	1,15							Pomer
Otočenie	1	3	8	20	45	90	180				Stupeň

Všetky zmeny boli prevedené za pomoci python skriptov vďaka čomu je veľmi jednoduché rozšíriť dataset o ďalšie obrázky.

Zmena jasu a kontrastu bola prevedená za pomoci funkcie „ImageEnhance“ z knižnice Pillow. Ako parameter berie pozitívne číslo, faktor, kde 0 vráti čierny obrázok pre jas a šedý pre kontrast, 1 vráti pôvodný obrázok.

Zmena gammy je vykoná pomocou funkcie na z obrázku 3.1, vstupom pre túto funkciu je obrázok a číslo, faktor. Toto číslo vyjadruje zmenu kde 1 vráti pôvodný obrázok a číslo menšie ako 1 vráti svetlejší. Využíva knižnice numpy pre výpočet pola ktoré sa následne za pomoci funkcie „LUT“ z knižnice openCV spojí s obrázkom, toto spojenie vykoná zmenu gammy.

Výpis 3.1: Funkcia na zmenu gammy obrázka

```
def gammaCorrection(src, gamma):
    invGamma = 1 / gamma
    table = [((i / 255) ** invGamma) * 255 for i in range(256)]
    table = np.array(table, np.uint8)
    return cv.LUT(src, table)
```

Zmena veľkosti obrázku bola vykonaná za pomoci funkcie z obrázku 3.2, vstupom pre túto funkciu je obrázok a násobok veľkosti. Z násobku si vypočítame novú veľkosť a pomocou funkcie „resize“ z knižnice openCV zmeníme veľkosť obrázku.

Výpis 3.2: Funkcia na zmenu veľkosti obrázku

```
def scale(img,ratio):
  x,y,c = img.shape
  nx = round(x*ratio)
  ny = round(y*ratio)
  newdim = (ny,nx)
  return cv.resize(img,newdim,interpolation=cv.INTER_CUBIC)
```

Gausov filter bol na obrázok aplikovaný za pomoci funkcie „GaussianBlur“ z knižnice openCV, kde ako vstup si vieme zvoliť veľkosť matice a odchýlku.

S&P šum bol aplikovaný pomocou iterátoru, ktorý iteroval cez každý pixel obrázku, pre PPT 1 je pre každý pixel šanca 1 k 1000, že bude upravený, ak sa bude jednať o upravený pixel tak ja šanca 50%, že pixel bude čierny a 50% biely.

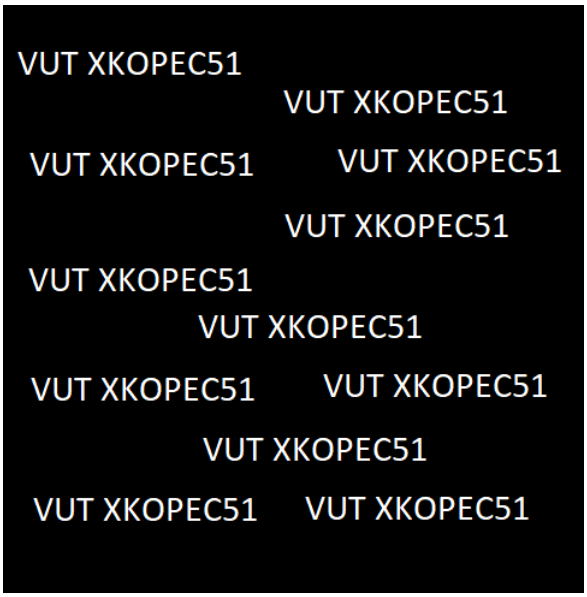
Speckl šum bol aplikovaný pomocou funkcie „random_noise“ z knižnice skimage. V tejto funkcii si vieme priamo zvoliť rozptyl.

JPEG kompresia bola na obrázok aplikovaná pomocou funkcie „save“ z knižnice Pillow kde ako parameter vstupu pre túto funkciu je kvalita kompresie.

Vodoznak v tabuľke nemá žiadne špecifické parametre. Bolo vytvorených 5 rôznych obrázkov. Prvý obrázok obsahuje okom neviditeľný vodoznak, ktorý bol vložený pomocou funkcie „WaterMark“ z knižnice „blind_watermark“ kde ako vstup vieme použiť textový reťazec. Ďalšie 3 obrázky obsahujú viditeľný vodoznak, ktorý vznikol pričítaním vodoznaku, obrázku, 3.2 k obrázku, ktorý chceme označiť, výsledok môžeme vidieť na obrázku 3.3. Toto sčítanie bolo prevedené za pomoci funkcie „addWeighted“ z knižnice openCV. Jeden z parametrov pre túto funkciu ktorý si vieme nastaviť je váha sčítania ktorá určuje viditeľnosť druhého obrázku. Posledný

Obr. 3.1: Ukážka speckl šumu

obrázok obsahuje malý viditeľný vodoznak v rohu obrázku vytvorený rovnakou metódou ako predošlé.



Obr. 3.2: Obrázok vodoznaku

Obr. 3.3: Obrázok s vloženým vodoznakom

Orezanie obrázku bolo vykonané pomocou kódu, ktorý môžeme vidieť na obrázku 3.3, pomer, ratio, číslo určuje rohy obrázku. Čo znamená, že ak máme obrázok široký 100 pixlov, po orezaní v pomere 0,95 bude obsahovať pixle 5 až 95. Toto orezanie je prevedené pomocou funkcie „crop“ z knižnice Pillow.

Výpis 3.3: Kód pre orezanie obrázku

```
ratio = 0.85
cropx = round(nx*ratio)
cropy = round(ny*ratio)
im1 = im.crop((nx-cropx,ny-cropy,cropx,cropy))
im1.save(name+"crop085.png")
```

Na koniec, otočenie obrázku bolo prevedené pomocou funkcie „rotate“ z knižnice Pillow kde ako parameter použijeme uhol o koľko chceme otočiť.

4 Riešenie

V tejto kapitole opíšeme prostredie, ktoré bolo použité pre riešenie spolu s hlavnými knižnicami, ktoré sme použili. Taktiež si následne opíšeme samotné riešenie a dosiahnuté výsledky.

4.1 Prostredie

Táto práca je riešená v jazyku Python. Python je interpretovaný, objektovo orientovaný programovací jazyk na vysokej úrovni s dynamickou sémantikou. Jeho vysokoúrovňové vstavané dátové štruktúry v kombinácii s dynamickým písaním a dynamickým viazaním ho robia veľmi atraktívnym pre rýchly vývoj aplikácií, ako aj pre použitie ako skriptovací alebo tzv. „lepiaci jazyk“ na prepojenie existujúcich komponentov. Jednoduchá syntax jazyka Python, ktorá sa dá ľahko naučiť, zdôrazňuje čitateľnosť, a preto znižuje náklady na údržbu programu. Python podporuje moduly a balíky, čo podporuje modularitu programu a opätovné použitie kódu. Interpret Pythonu a rozsiahla štandardná knižnica sú k dispozícii v zdrojovej alebo binárnej forme bez poplatku pre všetky hlavné platformy a možno ich voľne šíriť [15].

Programátori často preferujú Python kvôli zvýšenej produktivite, ktorú poskytuje. Keďže neexistuje žiadny kompilačný krok, cyklus úprav-test-ladenie je neuveriteľne rýchly. Ladenie programov v Pythone je jednoduché: chyba alebo zlý vstup nikdy nespôsobí chybu segmentácie. Namiesto toho, keď interpreter objaví chybu, vyvolá výnimku. Keď program nezachytí výnimku, interpret vytlačí stopu zásobníka. Debugger na úrovni zdroja umožňuje kontrolu lokálnych a globálnych premenných, vyhodnocovanie ľubovoľných výrazov, nastavovanie bodov prerušenia, prechádzanie kódom po riadkoch atď. Debugger je napísaný v samotnom Pythone, čo svedčí o introspektívnej sile Pythonu. Na druhej strane, často najrýchlejším spôsobom ladenia programu je pridanie niekoľkých tlačových príkazov do zdroja: rýchly cyklus úprav-test-ladenie robí tento jednoduchý prístup veľmi efektívnym [15].

Na druhej strane, Python nie je najrýchlejší jazyk preto implementuje rôzne knižnice ako OpenCV a Numpy ktoré sú napísane v efektívnejšom jazyku ako C a C++.

Knižnice Opencv a Numpy tvoria veľkú časť nášho riešenia.

4.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) je open source softvérová knižnica počítačového videnia a strojového učenia. OpenCV bol vytvorený s cieľom poskyt-

núť spoločnú infraštruktúru pre aplikácie počítačového videnia a urýchliť využitie strojového vnímania v komerčných produktoch. Knižnica má viac ako 2500 optimalizovaných algoritmov, ktoré zahŕňajú komplexnú sadu klasických aj najmodernejších algoritmov počítačového videnia a strojového učenia [16].

4.1.2 Numpy

NumPy je projekt s open-source kódom, ktorého cieľom je umožniť numerické výpočty v jazyku Python. Je to základný balík pre vedecké výpočty v Pythone. Je to knižnica Pythonu, ktorá poskytuje viacrozmerný objekt poľa, rôzne odvodené objekty (ako sú maskované polia a matice) a rad rutín pre rýchle operácie s poľami, vrátane matematických, logických, manipulácií s tvarmi, triedenia, výberu, I/O, diskrétne Fourierove transformácie, základná lineárna algebra, základné štatistické operácie, náhodná simulácia a mnohé ďalšie [17].

4.2 Implementácia

Za pomoci spomenutých knižníc bol vytvorený skript v jazyku Python, ktorý vytvorí hash z obrázka podľa postupu, ktorý je opísaný v 2.3. Interpolácia a detekcia hrán boli implementované za pomoci funkcií zo spomenutej knižnice OpenCV, ako môžeme vidieť na výpise 4.1 v kroku 1 a 3. Krok 2, Saliency Map, získame pomocou python kódu z [22]. Krok 5, Compress sampling, je vykonaný za pomoci spomenutej knižnice numpy pomocou ktorej sme v kroku 0 vytvorili náhodnú vzorkovaciu maticu s Gausovím rozložením. Následne pomocou volania „matmul“ sme vykonali násobenie matice bloku obrázka a meracej matice, compress sampling. Kroky 4 a 6 sú vytvorené s pomocou základných python funkcií ako násobenie, vzostupné zoradenie hodnôt v liste, nájdenie hodnoty v liste a pridanie hodnoty do listu.

Výpis 4.1: Ukážka kódu

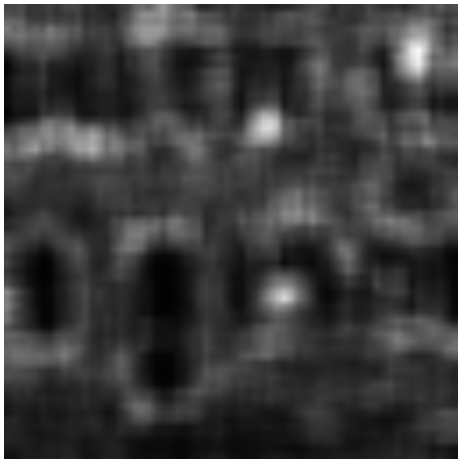
```
# krok 0 - input parameters
interpolation_size = 512
image_path = "kodak_default/kodim01.png"
image = cv.imread(image_path)
block_size=32
compression = 0.2  #1= nulova kompresia 0=maximalna CS
measurement_matrix = np.random.normal(loc=100, scale=40,
  size=(round(block_size**2 * compression),block_size**2))

# krok 1 - interpolation
image = cv.imread(image_path)
int_image = cv.resize(image,(interpolation_size,interpolation_size),
                      interpolation=cv.INTER_CUBIC)

#krok 2 - saliency map
intensty = intensityConspicuity(int_image)
gabor = gaborConspicuity(int_image, 4)
im = makeNormalizedColorChannels(int_image)
rg = rgConspicuity(int_image)
by = byConspicuity(int_image)
c = rg + by
nsize = resize_size
saliency = 1./3 * (cv2.resize(N(intensty),nsize) + cv2.resize(N(c),nsize) + N(gabor))
saliency_image = cv.resize(saliency,(interpolation_size,interpolation_size),
                      interpolation=cv.INTER_CUBIC)

# krok 3 - edge detection
edges_image = cv.Canny(int_image,100,200)

#krok 4 - weighted representation
edges_image = edges_image /255
weighted_image = edges_image * saliency_image

#krok 5 - Compress sampling
def m_formula(y):
  return 1/y.size * sum(y)

def v_formula(y):
  sum = 0
  m = m_formula(y)
  for i  in range(y.size):
    sum += (y[i] - m)**2
  return 1/(y.size - 1) * sum

long_hash = []
bs = block_size
for x in range(0,interpolation_size,block_size):
  for y in range(0,interpolation_size,block_size):
    mv = np.matmul(measurement_matrix,
              weighted_image[y:y+block_size,x:x+block_size].flatten('F'))
    long_hash.append(v_formula(mv))

#krok 6 - ordinal measures
sorted_hash = long_hash.copy()
sorted_hash.sort()
ordinal_hash = []
for i,s in enumerate(long_hash):
  ordinal_hash.append(sorted_hash.index(s))
print(ordinal_hash)
```

4.2.1 Saliency mapa

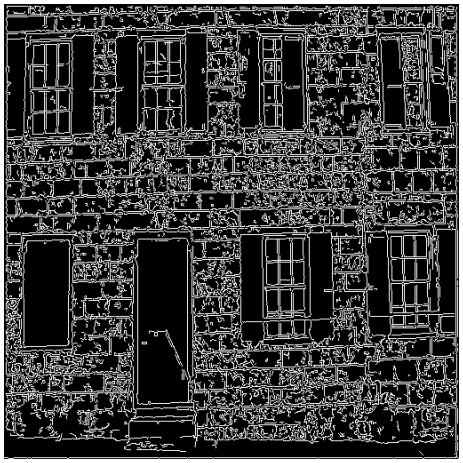
Saliency mapu podľa Itty získame implementáciou python kódu z [22]. Táto implementácia sa primárne opiera o knižnice Opencv, Numpy a matematické funkcie. Výstupná mapa má ľubovolné rozlíšenie. Pre výpočet saliecny mapy využívame aj druhú implementáciu Itty modelu, tzv. „Saliency toolbox“ [23]. Tento model je implementovaný v matlab kóde a v základnej konfigurácií vráti mapu o veľkosti 1/16 vstupného obrázku. Výstupné mapy týchto dvoch modelov nie sú zhodné, bližšie

ukážeme v sekcii 5.3, preto použijeme oba modely pri testovaní. Pri testovaní hashovacej metódy použijeme ešte tretí model, ktorý bol použitý aj v článku [8] pre testovanie výkonnosti. Jedná sa o tzv. „Spectral Residual“ model, ktorý získame z knižnice Opencv. Táto metóda vracia saliency mapu v identickej veľkosti vstupného obrázku.

(a) Krok 1 - interpolácia



(b) Krok 2 - saliency mapa



(c) Krok 3 - detekcia hrán

(d) Krok 4 - vážená reprezentácia

[8255806523.346277, 1490141755.2414305, 15760812297.71295,
3994991350.6757402, ... 2491129912.5684676, 5732355570.655183,
10101129401.488144, 725837709.4479252]

(e) Krok 5 - kompresné vzorkovanie

[48, 6, 58, 23, 10, 36, 54, 0, 32, 5, 61, 18, 31, 15, 39, 2, 60, 14, 53, 45, 35, 3, 20, 8,
50, 22, 52, 41, 17, 12, 33, 7, 24, 13, 62, 55, 25, 49, 37, 1, 43, 40, 34, 57, 28, 30, 21,
9, 44, 47, 46, 56, 27, 29, 26, 4, 63, 59, 51, 38, 11, 42, 19, 16]

(f) Krok 6 - ordinálne dáta

Obr. 4.1: Ukážka výstupov krokov z kódu 4.1

5 Výsledky práce

Výsledkom tejto práce je testovanie našej hashovacej metódy podľa článku [3], jej parametrov a jej následné porovnanie s hashovacou metódou „average hash“ z knižnice ImageHash, ktorú sme spomenuli v sekcii 2.4. Otestovanie modifikovaných obrázkov v našom datasete. Taktiež výstupom sú skripty ktoré dokážu automatizovane vytvoriť tieto modifikované obrázky pre náš dataset a ďalšie skripty ktoré slúžia pre vytvorenie hashu a ich následne porovnávanie.

Výsledky testovania môžeme vidieť v nasledujúcich sekciách. Pre hodnotenie výkonnosti budeme používať „receiver operating characteristic curve“ v skratke ROC krivku. Pre výpočet „True positive rate“ TPR a „True negative rate“ TNR použijeme vzorce 5.1 a 5.2, kde TP je počet skutočne pravdivých prípadov, P je celkový počet pravdivých prípadov, TN je počet skutočne nepravdivých prípadov a N je celkový počet nepravdivých prípadov. TP sú všetky hodnoty menšie ako testovacia hodnota a TN sú všetky hodnoty väčšie ako testovacia hodnota. Na grafe 5.10 modré body predstavujú pravdivé prípady, červene body predstavujú nepravdivé prípady, testovacia hodnota je L2 norma a testujeme pre všetky normy v distribúcii.

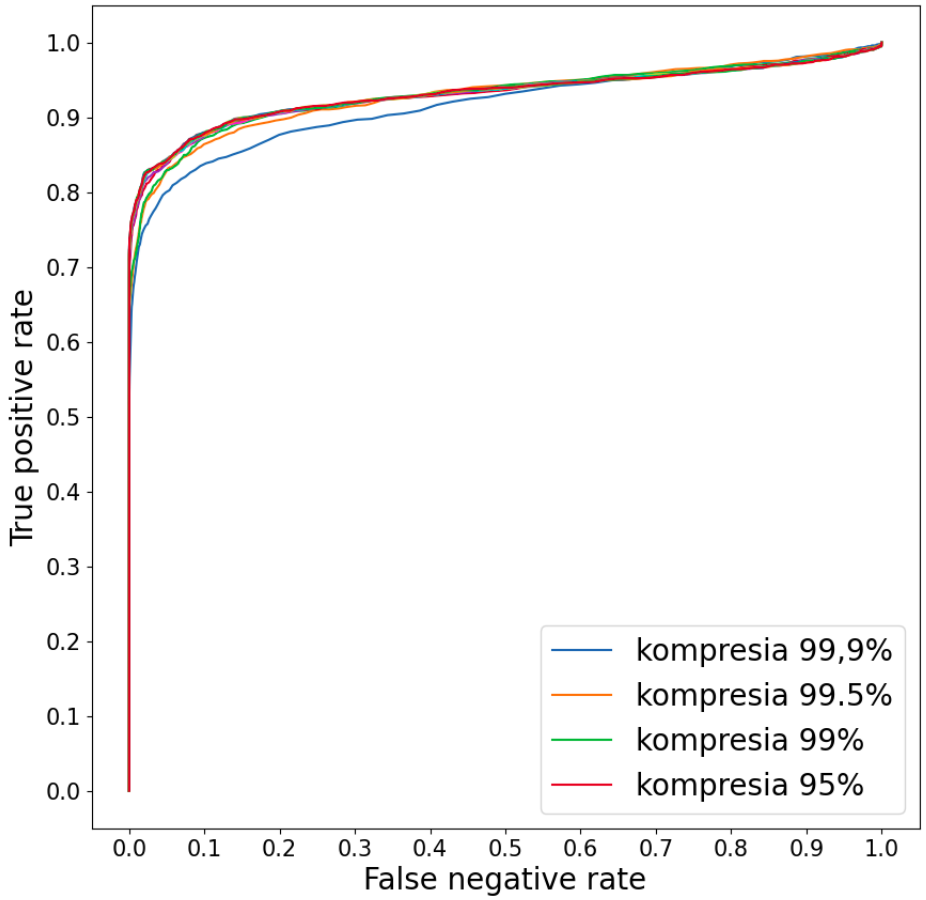
$$TPR = TP/P \tag{5.1}$$

$$TNR = TN/N \tag{5.2}$$

5.1 Vplyv kompresného vzorkovanie

V tejto časti si ukážeme vplyv kompresného vzorkovania na rozlišovaciu schopnosť hashovacej metódy a na jej rýchlosť. Kompresia pre vzorkovaciu maticu je definovaná ako pomer stĺpcov a riadkov, takže napríklad 90% kompresia pre 100 stĺpcovú maticu obsahuje 10 riadkov. Kompresia neovplyvňuje výslednú veľkosť hashu. Pri testovaní sme použili našu hashovaciu metódu s python implementáciou Itty saliency mapy, s datasetom, ktorý sme vytvorili v kapitole 3, L2 normou pre porovnávanie hashov a s veľkosťou bloku 64. Pre každé meranie kompresného vzorkovanie sme museli vygenerovať novú vzorkovaciu maticu čo môže samo o sebe ovplyvniť rýchlosť a rozlišovaciu schopnosť hashu ale nie v takej miere ako zmena kompresného vzorkovania. Meranie vyobrazené na nasledujúcich grafoch zobrazuje iba jedno meranie nakoľko sa jedná o časovo náročný test. Pri tvorbe a testovaní funkcií bolo prevedených niekoľko meraní a vyobrazené meranie na nasledujúcich grafoch je reprezentatívne týchto výsledkov.

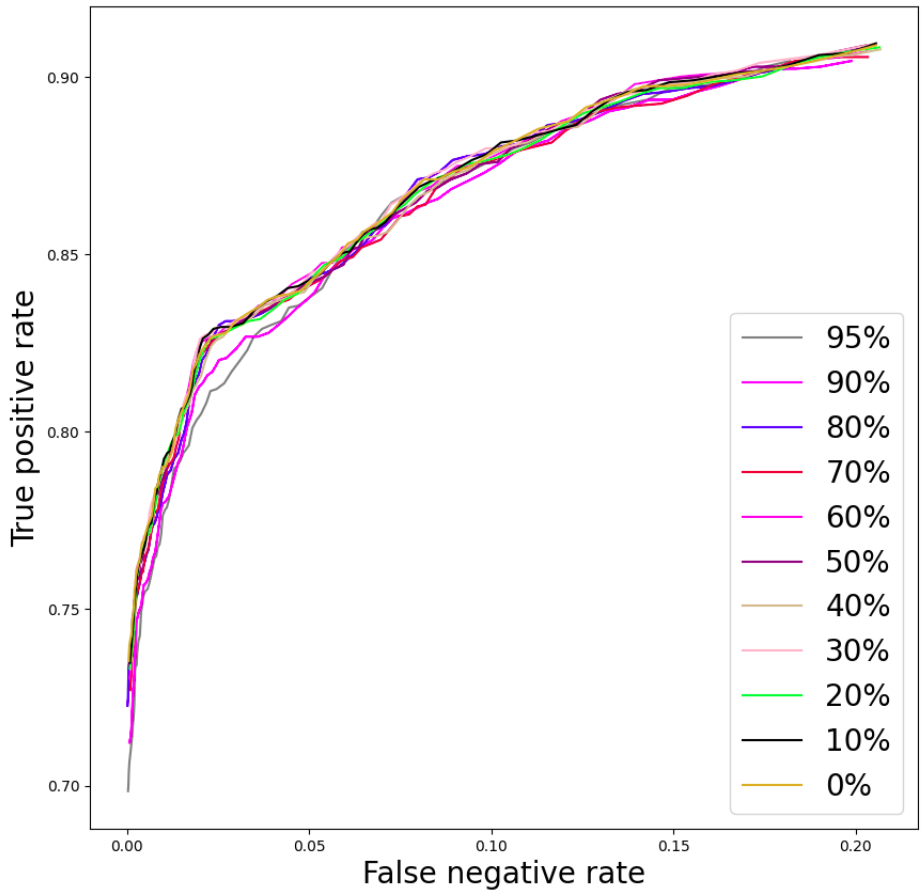
Na grafe 5.1 môžeme vidieť ROC krivky pre všetky testované kompresie. Tu môžeme vidieť, že kompresie väčšie ako 99% majú viditeľne horšiu ROC krivku ako ostatné kompresie.



Obr. 5.1: ROC krivky všetkých kompresií

Tieto ostatné kompresie okrem troch najväčších môžeme bližšie vidieť na grafe 5.2. Tu si môžeme všimnúť, že kompresia 95% a 90% má o niečo horšiu ROC krivku ako zvyšok. Tieto ostatné kompresie v rozsahu 0% až 80% sa pohybujú blízko seba. Nakoľko sa tieto krivky pretínajú a striedajú, môžeme považovať tieto kompresie za rovnocenné z hľadiska ich výkonnosti na ROC krivke. Rozdiely medzi nimi môžeme považovať za odchýlku merania spôsobenou meracou maticou.

Ďalšiu vlastnosť ktorú sme merali je čas ako dlho trvalo vytvoriť hashe. Toto meranie môžeme vidieť na grafe 5.3. Aby sme vylúčili výkonnosť prostredia sú časy vyjadrené ako násobok času, ktorý bol potrebný pre kompresiu 99.99% ktorej čas bol 90 sekúnd bez saliency mapy a 30 minút s python Itty saliency mapou. Na grafe sú dve krivky, modrá a červená, ktoré majú skoro identický tvar ale používajú inú mierku na ose Y. Červená krivka vyjadruje časy ktoré boli potrebné pre celú hashovaciu metódu a modrá krivka tiež vyjadruje čas pre hashovaciu metódu ale



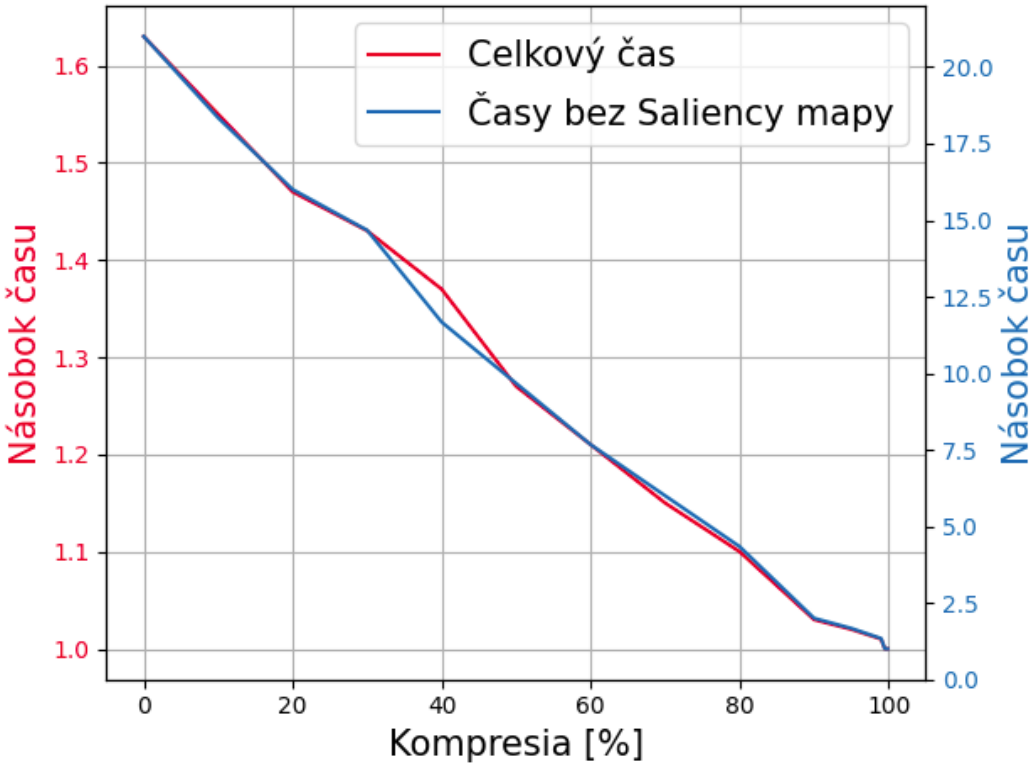
Obr. 5.2: ROC krivky kompresií - priblížené

tento krát bez výpočtu saliency mapy. Nakoľko výpočet saliency mapy zaberá väčšinu času dokážme lepšie odmerať čas potrebný pre kompresné vzorkovanie. Z ROC kriviek sme videli, že už pri 80% kompresii sme dosahovali optimálne výsledky. Čas pre 80% kompresiu bez výpočtu saliency mapu je 4,34 násobok čo je v porovnaní s 2 násobkom pre 90% kompresiu o niečo viac ale získame lepšie vlastnosti.

Nakoľko 80% kompresia je najrýchlejšia zo skupiny kompresií ktoré sú od seba rozdielne odchýlkou merania môžeme považovať túto 80% kompresiu najideálnejšiu z hľadiska rýchlosti a výkonností.

5.2 L1 vs L2 pre porovnávanie hashov

V tejto časti si ukážeme rozdiel medzi normou l1 a normou l2 pre porovnávanie hashov. Zameráme sa opäť na výkonnosť na ROC krivke a čas, ktorý je potrebný pre vypočítanie týchto noriem. Pri testovaní sme použili rovnaké parametre ako pri testovaní kompresného vzorkovanie spolu s 80% kompresiou. Pre testovanie sme použili jednu, spoločnú, vzorkovaciu maticu.

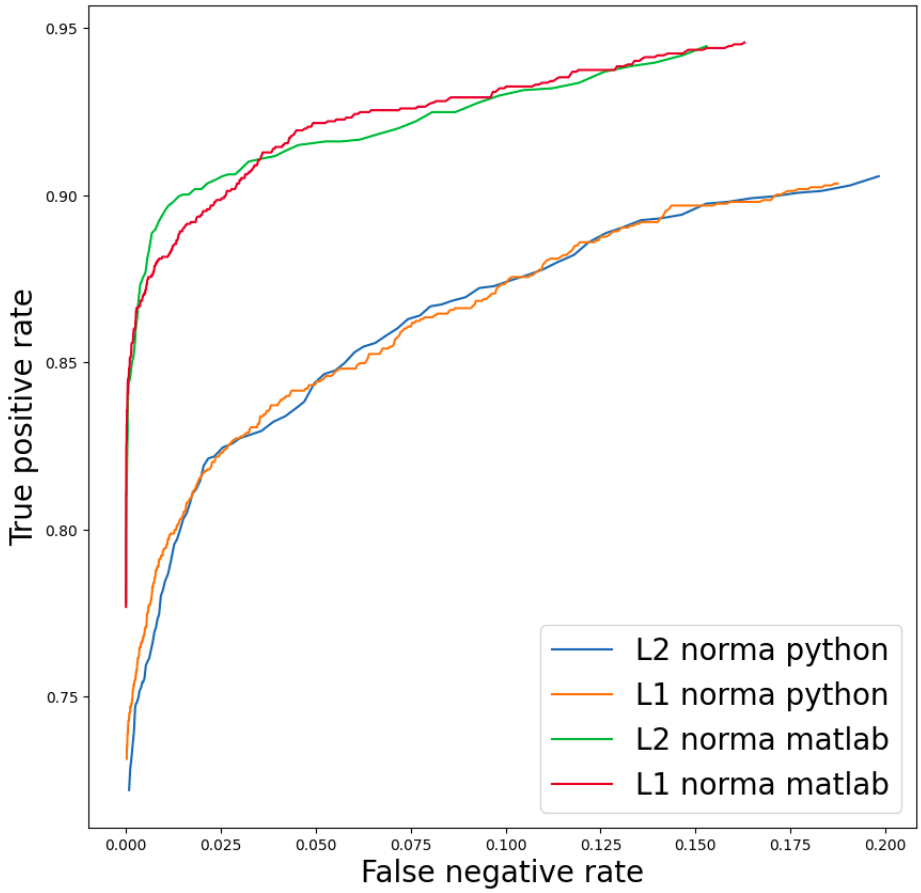


Obr. 5.3: Časy pre rôzne kompresie

Začneme rýchlosťou, L2 norma potrebuje v priemere približne 2 násobok času ako L1 norma. Dôvod prečo L2 norma je pomalšia je ten že miesto absolútnej hodnoty počíta druhú mocninu čo je komplikovanejšie a na koniec ešte potrebuje vypočítať druhú odmocninu. V praxi to znamená, že čas pre vypočítanie 43776 L2 noriem, počet noriem ktoré počítame pre našu databázu, je približne 2 sekundy a 1 sekundu pre L1 normu. To znamená, že najväčšie zrýchlenie aké môžeme získať použitím L1 normy je 1%.

Na grafe 5.4 môžeme vidieť 4 krivky. Najprv sa zameráme na krivky „L1/L2 norma python“. Tieto krivky sú si veľmi podobné, dokonca krivka L1 normy vyzerá byť efektívnejšia. Nakoľko tieto normy majú z ich definície veľmi rozdielne vlastnosti, očakávali sme rôzne výsledky, preto sme do grafu vykreslili aj meranie s inými parametrami. Toto meranie je vykreslené krivkami „L1/L2 norma matlab“ ktoré používajú Matlab implementáciu Itty saliency mapy. Na týchto krivkách môžeme vidieť väčšie rozdiely medzi L1 a L2 normou.

Z tohoto môžeme usúdiť, že pre náš model hashovacej metódy s python implementáciou Itty saliency mapy nezáleží či použijeme L1 alebo L2 normu. Nakoľko ale L2 norma má lepšie vlastnosti pri použití matlab Itty saliency modelu, ktorý sám o sebe zlepšuje výkonnosť na ROC krivke a časový rozdiel je skoro zanedbateľný, budeme naďalej používať L2 normu.



Obr. 5.4: ROC krivky L1 vs L2 norma pre porovnávanie hashov

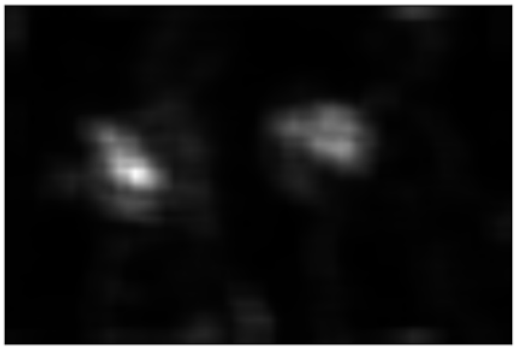
5.3 Porovnanie saliency máp

Ako sme už mohli vidieť na grafe 5.4 rozdiel medzi rôznymi implementáciami saliency mapy môže byť veľký, preto si ďalej porovnáme rôzne saliency mapy. Ako bolo spomenuté v 4.2.1, používame 2 implementácie Itty modelu pre výpočet saliency mapy a druhý, Spectral Residual, model pre testovanie. Na obrázkoch 5.5 môžeme vidieť ukážku saliency máp z rôznych modelov a ich vstupný obrázok. Tieto saliency mapy sú od seba veľmi rozličné. Pre rôzne vstupné obrázky môže mať vždy iný model najprirodzenejšiu saliency mapu. Pre testovanie použijeme rovnaké parametre ako do teraz, veľkosť bloku 32 a opäť spoločnú vzorkovaciu maticu. Databáza obsahuje 1824 obrázkov ale pozostáva iba z 24 obrázkov ktoré ale obsahujú ľudí, prírodu, zvieratá a ľudmi postavené štruktúry. To ale neznamená, že vieme univerzálne určiť najlepší model.

Keď sa pozrieme na ROC krivky na grafe 5.6 môžeme jasne vidieť, že jedna implementácia je výrazne lepšia a to je Matlab Itty implementácia zo „Saliency toolbox“. Nakoľko naša hashovacia metóda je implementovaná v Python, tieto Matlab saliency mapy boli pred počítané v Matlabe. Toto pred vypočítanie 1848 máp v on-

(a) Pôvodný obrázok

(b) Matlab Itty



(c) Spectral Residual

(d) Python Itty

Obr. 5.5: Ukážky saliency máp rôznych modelov

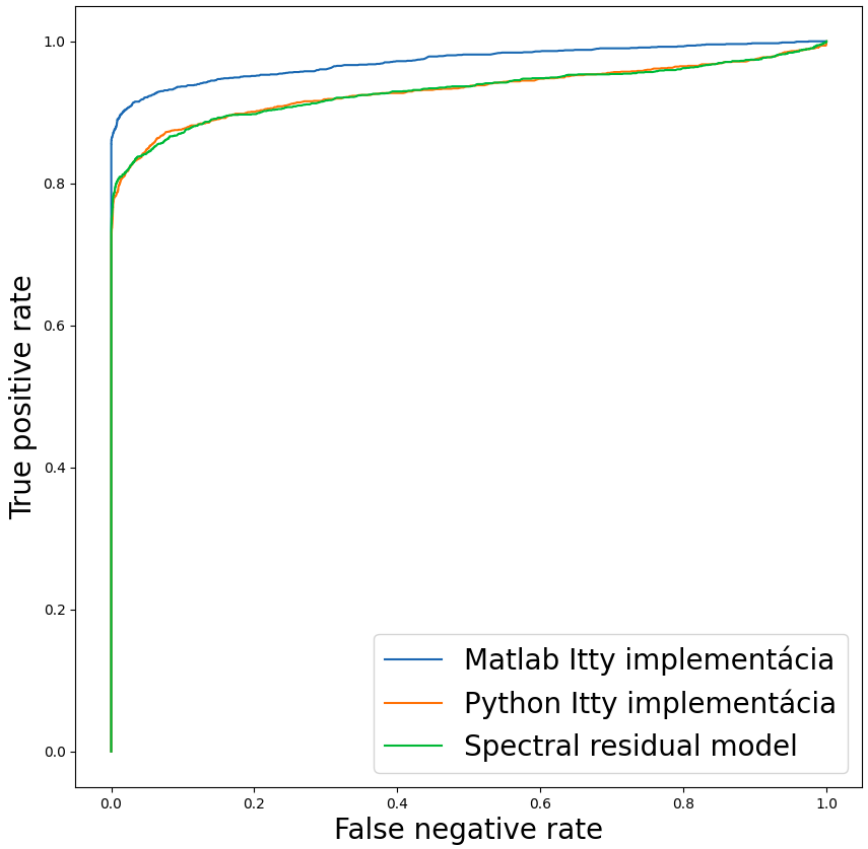
line Matlabe trvalo približne 10 minút. Keď to porovnáme s Python impelentáciou Itty modelu, ktorá potrebovala približne 28 minút je tento rozdiel taký veľký, že môžeme jednoznačne povedať, že Matlab implementácia je rýchlejšia. Jeden z dôvodov je ten že Python nie je najrýchlejší jazyk ale opiera sa o knižnice v iných jazykov ako C o rýchle výpočty. Nakoniec „Spectral residual“ metóda potrebovala približne 90 sekúnd v Python.

Napriek tomu, že „Spectral residual“ metóda je najrýchlejšia, Matlab implementáciu môžeme považovať za najlepšiu nakoľko má výrazne lepšiu výkonnosť na ROC krivke.

5.4 Vplyv veľkosti bloku hashovacej metódy

V tejto časti si ukážeme vplyv veľkosti bloku na hashovaciu metódu, konkrétne vplyv na ROC krivku a rýchlosť hashovania. Pri testovaní sme použili 80% kompresiu, veľkosť bloku 32 a Matlab implementáciu Itty modelu pre saliency mapu.

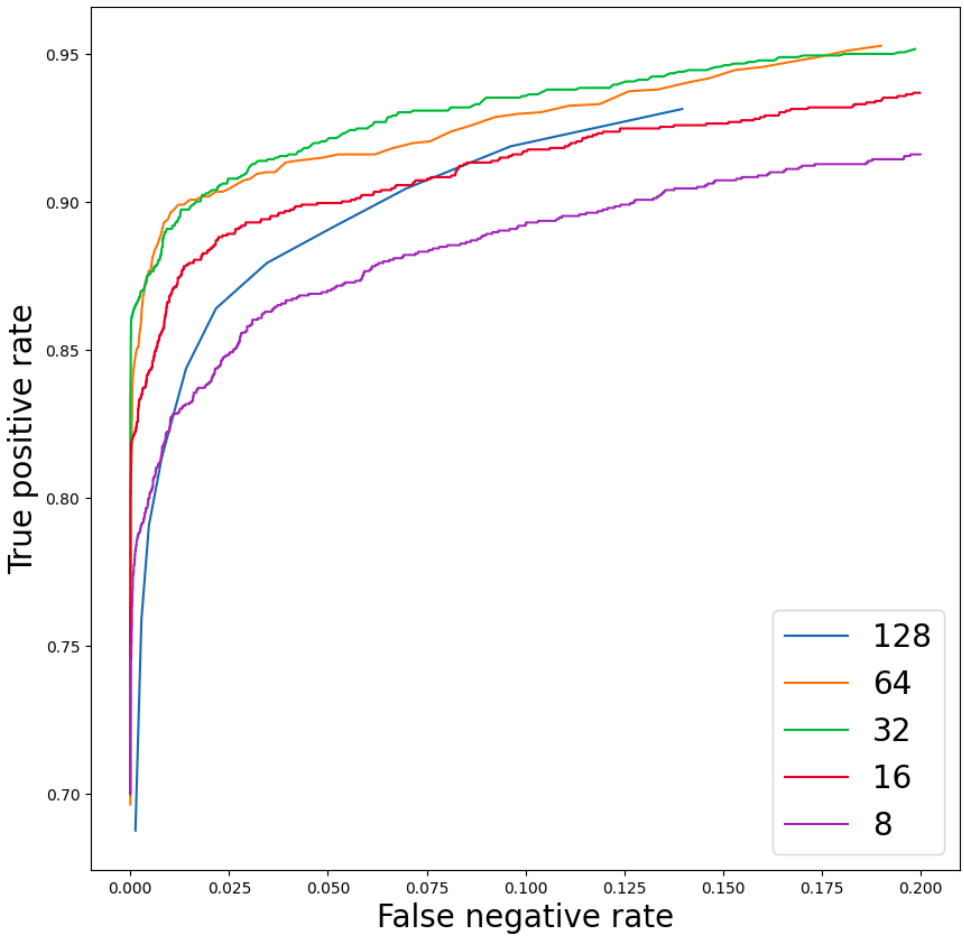
Na grafe 5.7 môžeme vidieť ROC krivky pre rôzne veľkosti blokov ktoré sa používajú pri hashovani. Môžeme si všimnúť, že pri zmenšovaní veľkosti bloku existuje



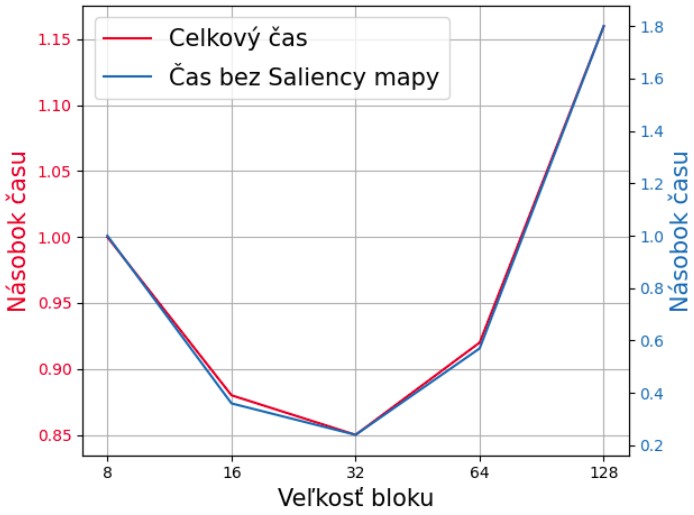
Obr. 5.6: ROC krivky rôznych saliency implementácii a modelov

bod zvratu kde prestávame získavať lepšie vlastnosti. Tento bod je niekde okolo veľkosti bloku 64 a 32 kde dosahujeme najlepšie vlastnosti.

Keď sa pozrieme na časy, ktoré sú v grafe 5.8, ktorý využíva rovnaké typy os z rovnakého dôvodu ako graf, ktorý sme použili pre časy pri porovnávaní noriem, môžeme si všimnúť že hashovacia funkcia pracuje najrýchlejšie pri použití blokov o veľkosti 32. Veľkosť bloku ovplyvňuje počet blokov na ktoré je potreba aplikovať kompresné vzorkovanie a zároveň ovplyvňuje veľkosť vzorkovacej matice. Menšia veľkosť bloku vygeneruje väčší počet blokov čo znamená je potreba previesť viacej operácií vzorkovania ale zároveň potrebuje menšiu vzorkovaciu maticu čo znamená jednoduchšie vzorkovanie. Preto tieto časy nemajú lineárny charakter ale existuje tzv. „sweet spot“ v strede kde je rovnováha medzi zložitosťou a množstvom operácií. Nakoľko veľkosť bloku 32 je najrýchlejšia a má jednu z najlepších alebo najlepšiu výkonnosť na ROC krivke, môžeme usúdiť, že táto veľkosť bloku je najlepšia pre našu hashovaciu metódu.



Obr. 5.7: ROC krivky rôznych veľkostí blokov



Obr. 5.8: Časy pre rôzne veľkosti blokov

5.5 Vplyv modifikácie obrázkov

V tejto časti si ukážeme aký veľký negatívny vplyv majú zmeny ktoré sme aplikovali na naše obrázky na ich podobnosť s nemodifikovaným obrázkom. Pre testovanie použijeme rovnaké parametre ako do teraz, ale použijeme Matlab implementáciu Itty modelu.

Na obrázkoch 5.9 a 5.10 môžeme vidieť jednotlivé grafy pre každú modifikáciu, ktorú sme aplikovali. Hodnoty na grafoch sú priemerné hodnoty zo všetkých obrázkov na ktoré bola aplikovaná zmena. To znamená, že každý bod na grafe reprezentuje hodnotu z 24 obrázkov. Osa X reprezentuje parameter aplikovanej modifikácie a Y ukazuje veľkosť L2 normy.

Pre lepšie získanie predstavy aká veľkosť L2 normy je „dobrá/zlá“ si ukážeme distribúciu L2 noriem na grafe 5.11. Osa X reprezentuje L2 normu a Y reprezentuje jeden zo základných kodak obrázkov, kodim01 až kodim24. Modré body sú L2 normy zhodných obrázkov, čiže sme porovnali každý základný obrázok zo všetkými jeho modifikáciami čo nám dá 1824 noriem. Červené body reprezentujú L2 normy nezhodných obrázkov, čiže každý základný obrázok sme porovnali zo všetkými ostatnými modifikovanými a základnými obrázkami čo nám dá 41952 noriem. Priemerná hodnota zhodných obrázkov je 873, nezhodných 2194. Najväčšia a najmenšia hodnota L2 normy pre zhodné obrázky je 2504 a 0, pre nezhodné obrázky to je 2726 a 1441. To znamená, že L2 normu o veľkosti 1441 môžeme považovať za hranicu, ak ja L2 norma väčšia ako táto hranice začneme nesprávne vyhodnocovať zhodu obrázkov.

Na základe tejto znalosti vieme napríklad povedať, že všetky testované zmeny typu „orezanie“ budú nesprávne vyhodnotené keďže všetky hodnoty sú väčšie alebo veľmi blízke našej hranici 1441. Zmien ktoré majú hodnoty väčšie alebo veľmi blízke ako táto hranica je 15 čo je 360 obrázkov a to je približne 20% nášho datasetu. Niektoré zmeny ako napríklad Gausov šum nemajú taký veľký vplyv na podobnosť ako napríklad, už spomenuté, orezanie.

5.6 Porovnanie hashovacej metódy

Na koniec si v tejto časti porovnáme našu hashovaciu metódu podľa článku [3] s hashovacou metódou „average hash“ z voľne dostupnej Python knižnice „ImageHash“, ktorú sme spomenuli v 2.4, na našej databáze obrázkov. Pre našu hashovaciu metódu použijeme parametre ktoré sme v predošlých častiach určili za najlepšie a to 80% kompresiu, veľkosť bloku 32 a Matlab implementáciu Itty saliency mapy. Pre porovnávanie hashov opäť použijeme L2 normu. Aby sme mohli použiť rovnaké funkcie pre porovnávanie hashov z funkcie „Average Hash“ musíme výstupný hexadecimálny

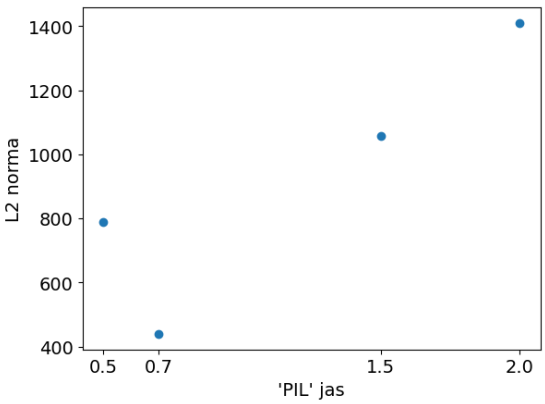
hash prekonvertovať do decimálnej podoby. Vstupné obrázky pre hashovaciu metódu „average hash“ nie sú nijak upravované, jediný vstupný parameter, ktorý používame je „hash_size“ s hodnotou 128.

Začneme časovou náročnosťou, pre vytvorenie hashov pomocou našej hashovacej metódy sme potrebovali približne 12 minút. Metóda „average hash“ potrebovala približne 41 sekúnd pre vytvorenie hashov a 25 sekúnd pre prevod do decimálnej podoby, tento prevodník by sa dal ďalej zefektívniť.

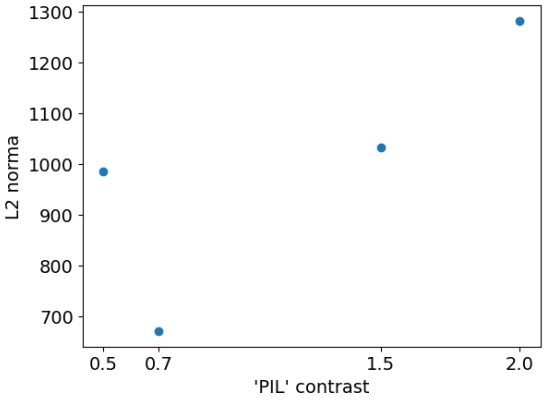
Ďalej si môžeme všimnúť graf 5.12 kde môžeme vidieť porovnanie týchto metód za pomoci ROC krivky. Rozdiel nie je veľký ale je jasne viditeľné, že modrá krivka našej hashovacej metódy má lepšie vlastnosti.

Na koniec si ešte ukážeme distribúciu L2 noriem z „Average hash“ metódy. Túto distribúciu môžeme vidieť na grafe 5.13. Túto distribúciu môžeme porovnať s distribúciou noriem pre našu hashovaciu metódu na grafe 5.11. Ale hodnotu L2 noriem nemôžeme priamo porovnávať medzi sebou nakoľko dĺžky hashov a rozsah hodnôt v hashoch nie je rovnaký takže maximálna hodnota L2 normy je rozdielna. Na grafe z funkcie „Average Hash“ si môžeme všimnúť veľmi výraznej skupiny modrých bodov, medzeru a potom skupinu červených bodov. Na grafe distribúcie noriem z našej hashovacej jasne vidíme skupinu červených bodov, skupina modrých bodov je omnoho viacej rozprestrená a nemá žiadnu jednu výraznú skupinu. Taktiež nemôžeme vidieť žiadnu výraznú medzeru medzi modrými a červenými bodmi ako sme mohli vidieť na predošlej distribúcii. Ako sme už spomenuli pre našu hashovaciu metódu, približne 1464 modrých bodov má menšiu hodnotu L2 normy ako 1441 čo predstavuje 53% hodnotu maximálnej L2 normy na grafe. Keď to porovnáme s „average hash“, 1237 modrých bodov majú hodnoty L2 normy menšie ako 100 a táto hodnota je len 12% maximálnej L2 normy na grafe a ešte máme ďalších 218 čísel medzeru k najbližšej norme červeného bodu čo reprezentuje 25% medzeru zo spektra.

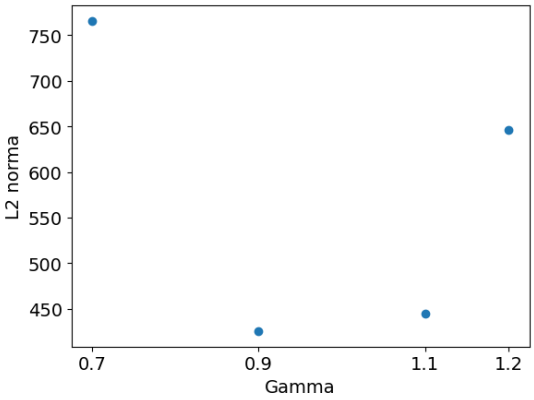
Na základe týchto nájdení je komplikované rozhodnúť ktorá metóda je lepšia. Ak budeme súdiť len na základe ROC krivky, naša metóda má malú ale jasnú výhodu oproti „Average Hash“ metóde. Keď sa pozrieme na čas, ktorý bol potrebný na výpočet je jasné, že naša metóda je niekoľko násobne pomalšia, ale niekedy aj násobne pomalšia ale presnejšia metóda je vhodnejšia ak potrebujeme maximálnu presnosť a máme dostatok času. Na druhej strane metóda „Average Hash“ má omnoho lepšiu zgrupovaciu vlasnosť ako sme mohli vidieť na grafoch distribúcie noriem. Táto vlastnosť môže byt veľmi žiadaná, kedy táto metóda má výhodu.



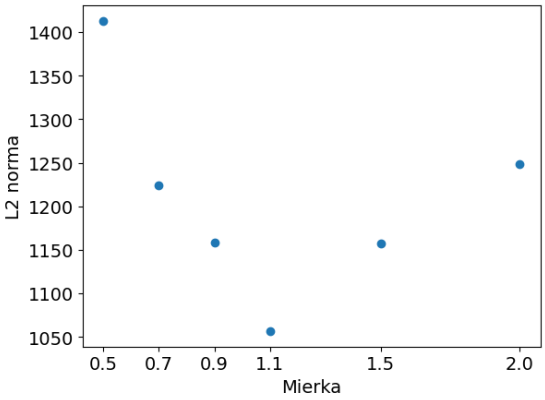
(a) jas



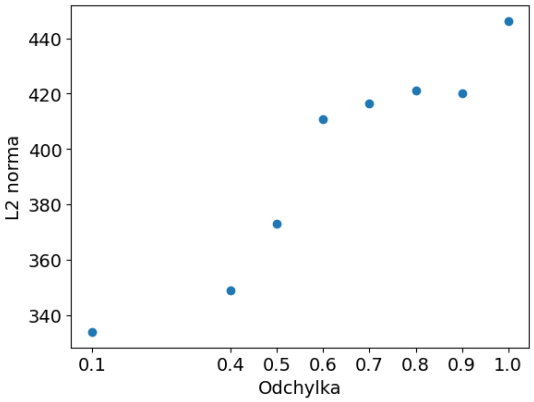
(b) Kontrast



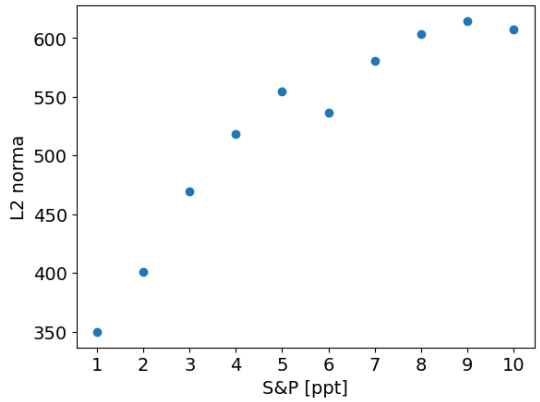
(c) Gamma



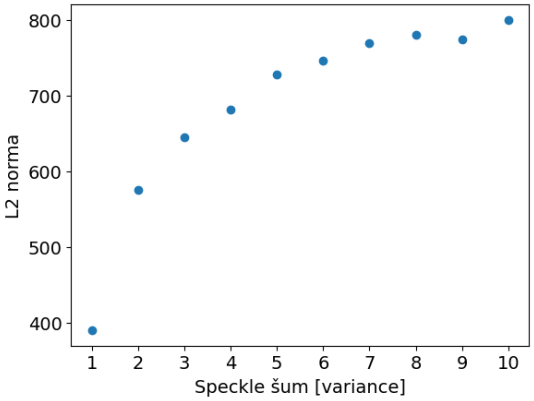
(d) Veľkosť



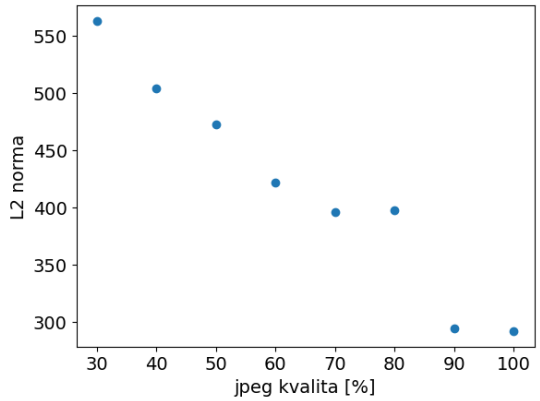
(e) Gaus šum



(f) Salt and Pepper šum

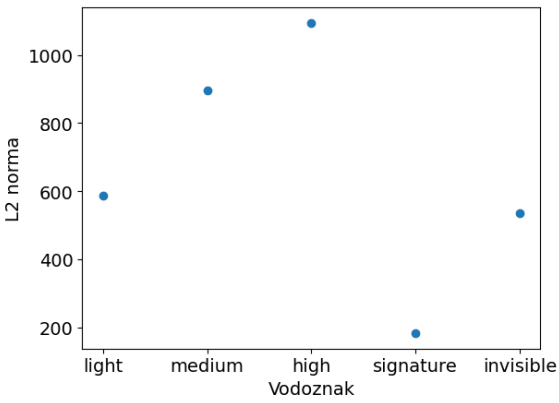


(g) Speckle šum

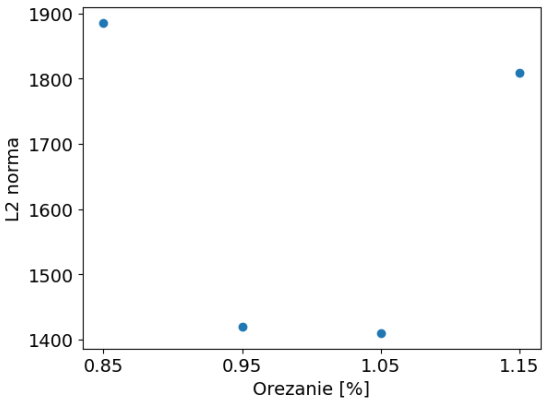


(h) Jpeg kompresia

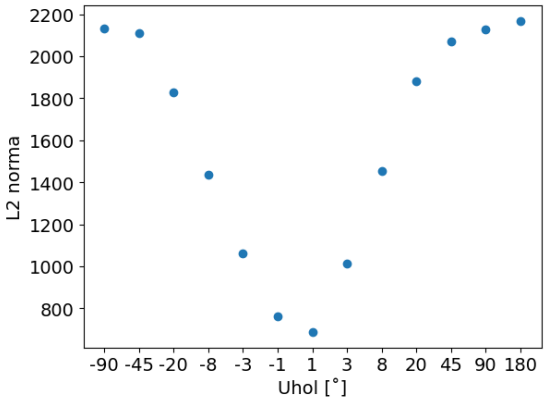
Obr. 5.9: Zmeny obrázkov (strana 1)



(a) Vodoznak

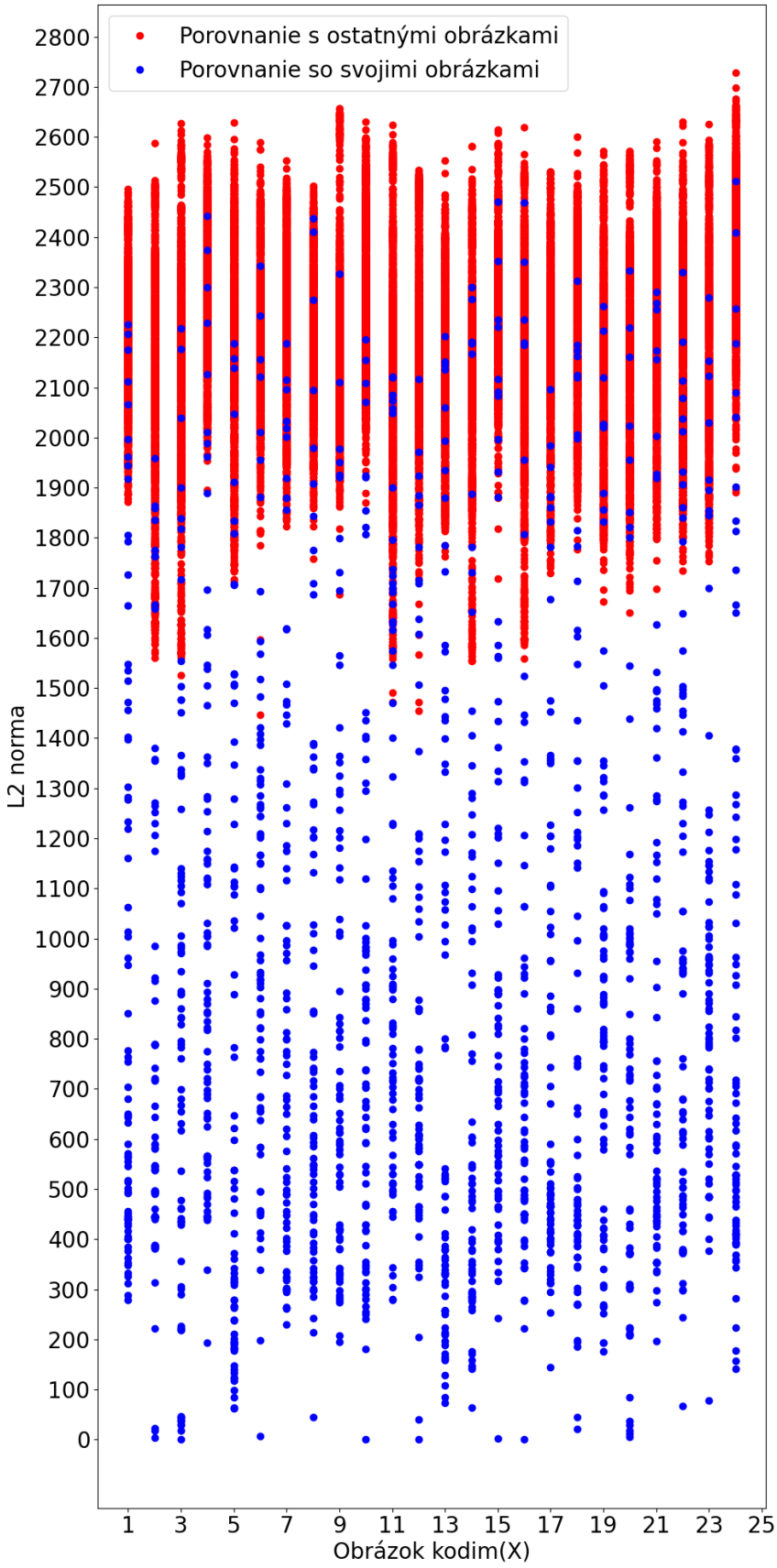


(b) Orezanie

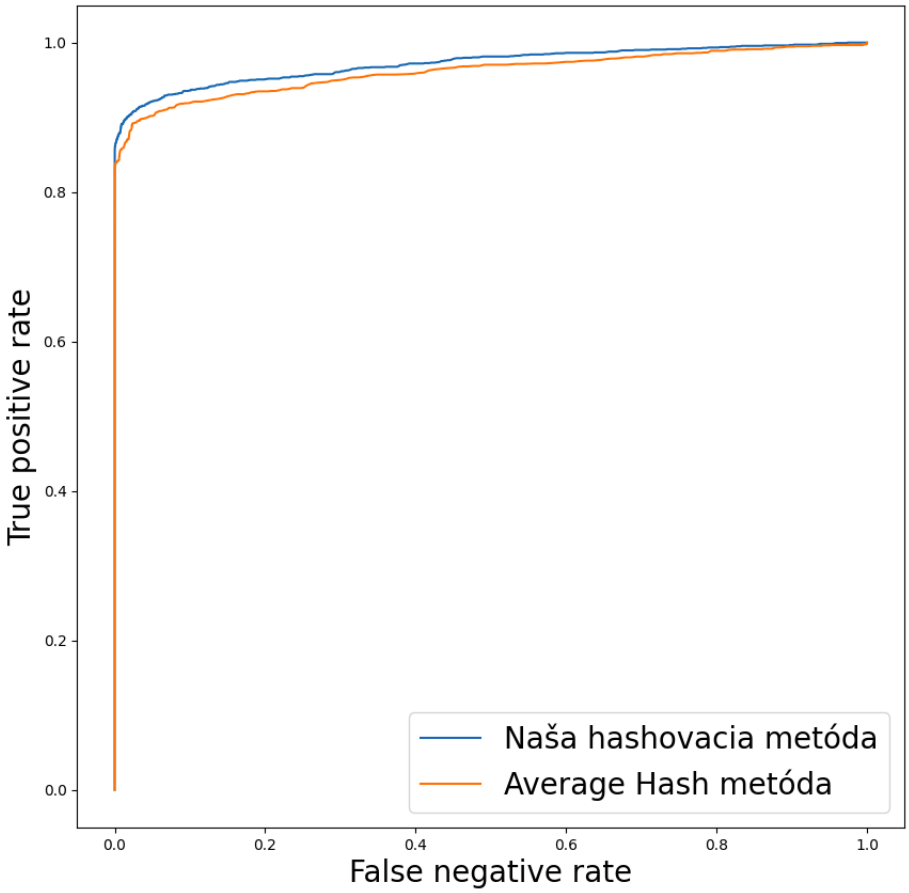


(c) Otočenie

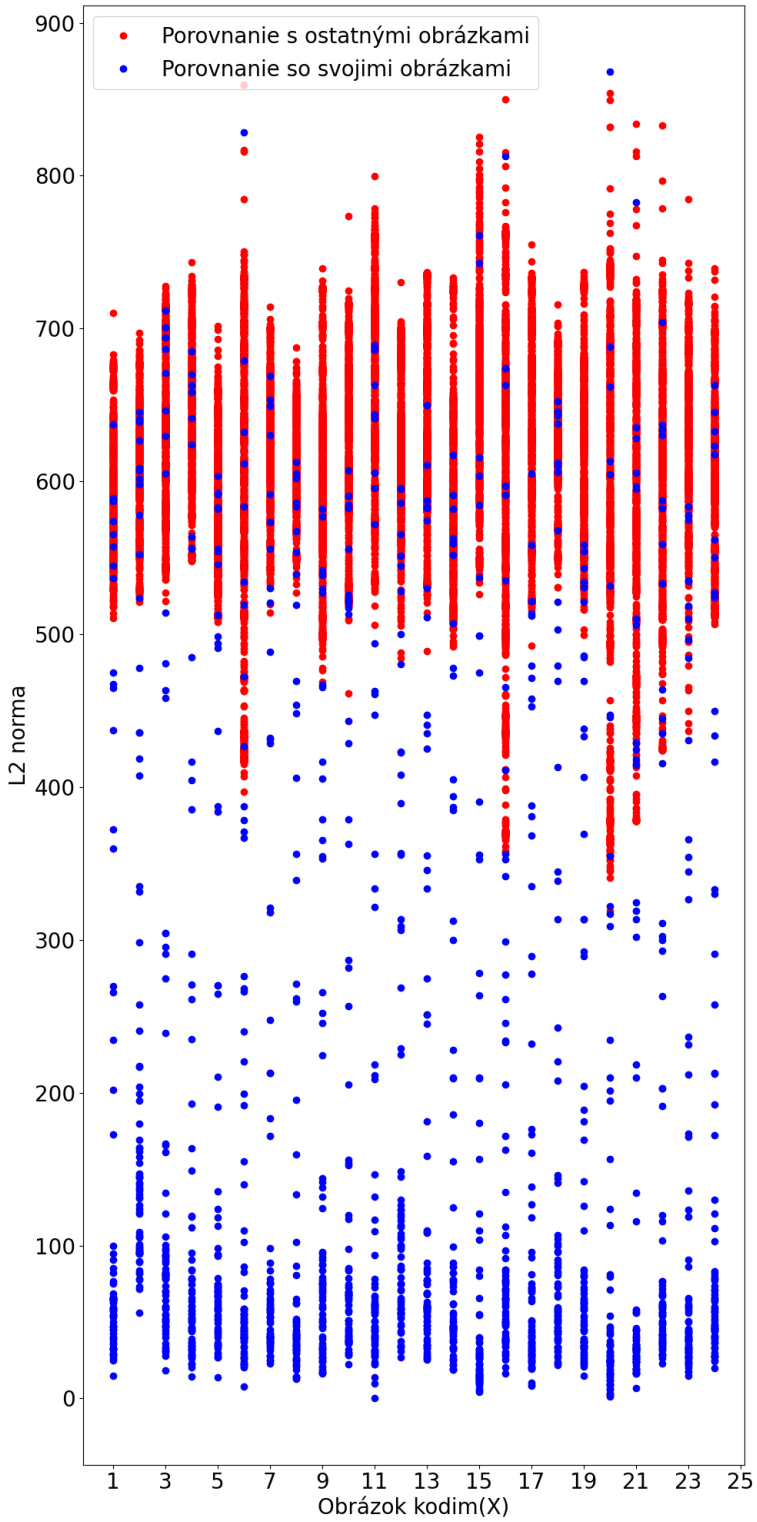
Obr. 5.10: Zmeny obrázkov (strana 2)



Obr. 5.11: Distribúcia L2 noriem našej hashovacej metódy



Obr. 5.12: ROC krivka Average hash



Obr. 5.13: Distribúcia L2 noriem metódy Average hash

Záver

Práca sa zaoberala teoretickým úvodom o kompresnom vzorkovaní, hashovaní obrázkov a opisom rôznych funkcií ktoré implementujú.

Cieľom bolo zoznámiť sa s kompresným vzorkovaním a implementovať model kompresného vzorkovania. Bolo vysvetlené ako funguje a aké má využitie kompresné vzorkovanie. Taktiež bol tento postup aplikovaný v jazyku Python na bloky obrázka pri vytváraní hashu.

Ďalším cieľom bolo sa zoznámiť s hashovaním obrázkom, s metódami pre vytváranie hashov z obrázkov a implementovať postup pre hashovanie obrázkov podľa článku. Opísali sme si rôzne metódy ktoré je možné použiť pre hashovanie a tiež sme si povedali ich využitie. Metóda pre hashovanie obrázkov podľa spomenutého článku bola a je schopná úspešne vytvoriť hash obrázku.

Ďalším cieľom bolo vytvoriť dataset pre testovanie. Tento dataset bol vytvorený za pomoci Python skriptov vďaka čomu je tento dataset možné prípadne automatizovane rozšíriť.

Ďalším cieľom bolo otestovať bolo otestovať dataset a hashovaciu metódu. Boli otestované rôzne paramatre pre našu hashovaciu metódu a ná základe týchto parametrov sme otestovali aký vplyv mali prevedené modifikácie na podobnosť obrázkov.

Posledným cieľom bolo porovnať našu hashovaciu metódu s inou hashovacou metódou. Našu hashovaciu metódu sme porovnali s hashovacou metódou „AverageHash“ z knižnice ImageHash pomocou rôznych metrík ako ROC krivka. Pri porovnávaní sme zistili, že tieto dve metódy majú ROC krivky blízko pri sebe, ale aj napriek tomu časová náročnosť a distribúcia hashov na grafe je veľmi rozdielna. Každá metóda ma svoje silné a slabé stránky, ktoré spomenuli.

Literatúra

[1] ERDOGAN TASKESEN: *Detection of Duplicate Images Using Image Hash Functions*[online]. Zverejnené dňa 28.1.2022. Citované dňa 1.11.2022. Dostupné z URL:
<https://towardsdatascience.com/detection-of-duplicate-images-using-image-ha

[2] *Kodak Lossless True Color Image Suite*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<http://r0k.us/graphics/kodak/>.

[3] TANG, Z., ZHANG, H., LU, S. ET AL.: *Robust image hashing with compressed sensing and ordinal measures*[online]. Zverejnené dňa 8.7.2020. Citované dňa 1.11.2022. Dostupné z URL:
<https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-020-00509-3>.

[4] *Tiny Eyey*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://help.tineye.com/article/233-how-does-tineye-work>.

[5] *PhotoDNA*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://www.microsoft.com/en-us/photodna>.

[6] *undouble*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://erdogant.github.io/undouble/pages/html/index.html>.

[7] M. STEINEBACH, H. LIU AND Y. YANNIKOS: *Efficient Cropping-Resistant Robust Image Hashing,"2014 Ninth International Conference on Availability, Reliability and Security, 2014, pp. 579-585, doi: 10.1109/ARES.2014.85.*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://ieeexplore.ieee.org/document/6980335>.

[8] L. ITTI, C. KOCH AND E. NIEBUR: *A model of saliency-based visual attention for rapid scene analysis,ïn IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 11, pp. 1254-1259, Nov. 1998, doi: 10.1109/34.730558.*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://ieeexplore.ieee.org/document/730558>.

[9] E. J. CANDES AND M. B. WAKIN: *An Introduction To Compressive Sampling,ïn IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 21-30, March 2008, doi: 10.1109/MSP.2007.914731.*[online]. Citované dňa 1.11.2022. Dostupné z URL:
<https://ieeexplore.ieee.org/document/4472240>.

[10] SUJITHA JULIET DEVARAJ: *Image Interpolation*[online]. Zverejnené v roku 2019. Citované dňa 1.11.2022. Dostupné z URL: <https://www.sciencedirect.com/topics/engineering/image-interpolation>.

[11] *Canny Edge Detection*[online]. Citované dňa 1.11.2022. Dostupné z URL: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>.

[12] KURTIS PYKES: *Fighting Overfitting With L1 or L2 Regularization*[online]. Zverejnené dňa 14.11.2022. Citované dňa 24.11.2022. Dostupné z URL: <https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization>.

[13] D. BARON, M.B. WAKIN, M.F. DUARTE, S. SARVOTHAM, AND R.G. BARANIUK: *"Distributed compressed sensing," 2005, Preprint.* Citované dňa 1.11.2022.

[14] E. CANDÈS AND T. TAO: *"Decoding by linear programming," IEEE Trans. Inform. Theory, vol. 51, no. 12, pp. 4203-4215, Dec. 2005.* Citované dňa 1.11.2022.

[15] *What is Python? Executive Summary*[online]. Citované dňa 1.11.2022. Dostupné z URL: <https://www.python.org/doc/essays/blurb/>.

[16] *OpenCV - About* [online]. Citované dňa 1.11.2022. Dostupné z URL: <https://opencv.org/about/>.

[17] *What is NumPy?*[online]. Citované dňa 1.11.2022. Dostupné z URL: <https://numpy.org/devdocs/user/whatisnumpy.html>.

[18] RAVI KATUKAM: *Image comparison Methods & Tools: A Review*[online]. Citované dňa 1.11.2022. Dostupné z URL: <https://www.academia.edu/19962797/Image_comparison_Methods_and_Tools_A_Review>.

[19] *ImageHash*[online]. Citované dňa 1.11.2022. Dostupné z URL: <https://pypi.org/project/ImageHash/>.

[20] SARA IRIS GARCIA: *0 Norm, L1 Norm, L2 Norm & L-Infinity Norm*[online]. Zverejnené dňa 1.5.2018. Citované dňa 1.11.2022. Dostupné z URL: <https://montjoile.medium.com/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18

[21] *Compressed Sensing in Python*[online]. Zverejnené dňa 26.5.2016. Citované dňa 1.11.2022. Dostupné z URL:

<http://www.pyrunner.com/weblog/2016/05/26/compressed-sensing-python/>.

[22] *Simple Itty-Koch-Style Saliency Maps*[online]. Zverejnené v roku 2016. Citované dňa 11.4.2022. Dostupné z URL: <https://gist.github.com/tatome/d491c8b1ec5ed8d4744c>.

[23] *Saliency toolbox*[online]. Zverejnené dňa 23.2.2007, aktualizované 21.1.2016. Citované dňa 11.4.2022. Dostupné z URL: <https://www.saliencytoolbox.net/index.html>.

Zoznam symbolov a skratiek

LSH	Lokálne senzitívne hashovanie(Locality-sensitive hashing)
CS	Kompresné vzorkovanie (Compress sampling)
DCT	Diskrétna kosínusová transformácia
S&P	Soľ a korenie (Salt and Pepper)
PPT	Častíc na tisíc (Parts per thousand)
ROC	Receiver operating characteristic
TPR	True positive rate
TNR	True positive rate

Zoznam príloh

A Riešenie - príloha

Odkaz na google colab kde sú uložené všetky skripty:
`https://colab.research.google.com/drive/1FMWD1wyUSOrWPfm7oxxx29o9aqsnZAi0?usp=sharing`

A.1 Definície všetkých funkcií

Blok s týmto nadpisom obsahuje všetky použité funkcie, jedna z nich je `hasher`, ktorú vieme zavolať, ako vstup jej dáme cestu k obrázku a vráti nám hash obrázku podľa zdroja [3] pomocou Python implementácie Itty mapy. Alebo funkcia `hasher_mat`, ktorá využíva Matlab implementáciu Itty mapy.

A.2 Average Hash

Blok s názvom `imagehash hashovanie` obsahuje implementáciu metódy Average hash.

A.3 Modifikácia obrázkov

Bloky s prefixom „dataset“ obsahujú skripty, ktoré slúžia na výrobu datasetu.

A.4 Testovanie

Bloky s prefixom „testovanie“ obsahujú skripty, ktoré sme použili na testovanie.