Komprese obrázků metodou singulárního rozkladu - zápočtový program

Dominik Rathan

[OOP] - zimní semestr 2018/2019

1 Základní charakteristika problému

Matici $A \in \mathbb{T}^{m \times n}$ můžeme singulárním rozkladem (SVD) rozložit do tvaru $A = U\Sigma V^T$, kde matice U, V jsou unitární a Σ je diagonální matice se singulárními hodnotami A na diagonále. Pomocí tohoto rozkladu můžeme značně minimalizovat počet uchovávaných čísel matice a zpětným vynásobením získat slušnou aproximaci původní matice A, jejíž kvalita bude záviset na volbě počtu singulárních hodnot, se kterými budeme v rozkladu pracovat.

Singulární rozklad tedy navádí k použití při kompresi obrázků. U barevných obrázků bude ve stručnosti princip následující. Obrázek rozložíme do 3 barevných složek R, G, B a pro každou tuto složku (uchovanou v nějaké matici s hodnotami z množiny $\{0, 1, \ldots, 255\}$) provedeme singulární rozklad. Následně tyto složky zpětným vynásobením složíme dohromady a získáme R, G, B složky výsledného zkomprimovaného obrázku.

2 Popis programu

Je zajímavé pozorovat, jak komprese metodou singulárního rozkladu pracuje pro různé obrázky a různě zvolený počet singulárních čísel. Vytvořil jsem tedy program, který bude výše popsanou metodu komprese demonstrovat. Pracoval jsem ve Visual Studiu v jazyce C# a vytvořil Windows Form Aplikaci, která uživateli umožňuje nahrát obrázek, následně ho zkomprimovat, s úrovní komprese manipulovat (na základě počtu singulárních čísel užitých v kompresi), popřípadě kdykoliv v průběhu obrázek editovat a následně se vrátit ke kompresi, samozřejmě s možností obrázek kdykoliv uložit. Dále při spuštění programu se uživateli na ploše vytvoří složka `Verze` (neexistuje-li již), kam se budou ukládat jednotlivé verze manipulace s obrázkem ve formátu `.txt` (podrobnosti viz níže). Uživatel se tak může při opětovných spuštění programu vracet k jednotlivým verzím obrázků a dále s nimi pracovat, měnit úroveň komprese či ho dále editovat. V rámci jednoho spuštění programu je též k dispozici historie verzí, k nimž se uživatel může navracet, pokud si například uvědomí, že jedna z verzí byla lepší, ale už zapomněl, jaké parametry ji charakterizovaly.

3 Popis jednotlivých stavů programu z hlediska uživatele

Začátek

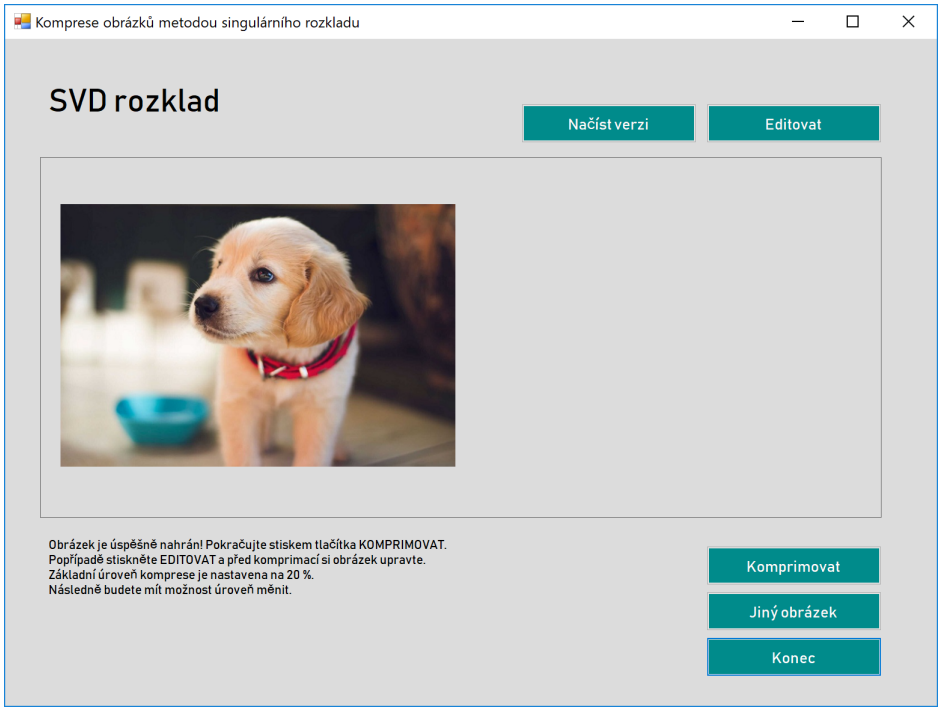
Na úvodní obrazovce je `Label` se stručným popisem funkčnosti programu, následně tlačítka

- **Nahrát obrázek**, po jehož stisknutí se otevře `OpenFileDialog` a uživatel může vybrat obrázek v jednom z formátů `.bmp, .jpg, .png, .gif` ze svého počítače.
- **Konec**, po jehož stisknutí se program ukončí
- **Načíst verzi**, které zobrazí `OpenFileDialog` a uživatel může ze složky `Verze` vybrat příslušný textový soubor `.txt`, který se načte, spočte, a zobrazí příslušný původní a zkomprimovaný obrázek včetně jejich parametrů ve stavu *Zkomprimováno*.

Načteno

Zobrazuje stav, kdy máme načtený obrázek z počítače, ale ještě neproběhla žádná komprese. Zobrazuje `TextBox` s popsáním současného stavu a návodem pro pokračování a `PictureBox` s originálním obrázkem. Oproti předchozímu stavu zde máme navíc tlačítka

- **Komprimovat**, po jehož stisknutí se obrázek začne komprimovat. Základní úroveň komprimace je nastavena na 20%, tedy ve zkomprimované verzi bude užito 20% původního počtu singulárních čísel. Velikost obrázku se tak značně zmenší, ale současně bude jeho kvalita relativně zachována.
- **Jiný obrázek** pro výběr jiného obrázku, funguje stejně jako **Načíst verzi** ze stavu *Začátek* a navrátí se zpátky do stavu *Načteno*.
- **Editovat**, které vyvolá stav *Editace* a uživatel má možnost před provedením komprese obrázek jednoduchým způsobem editovat.



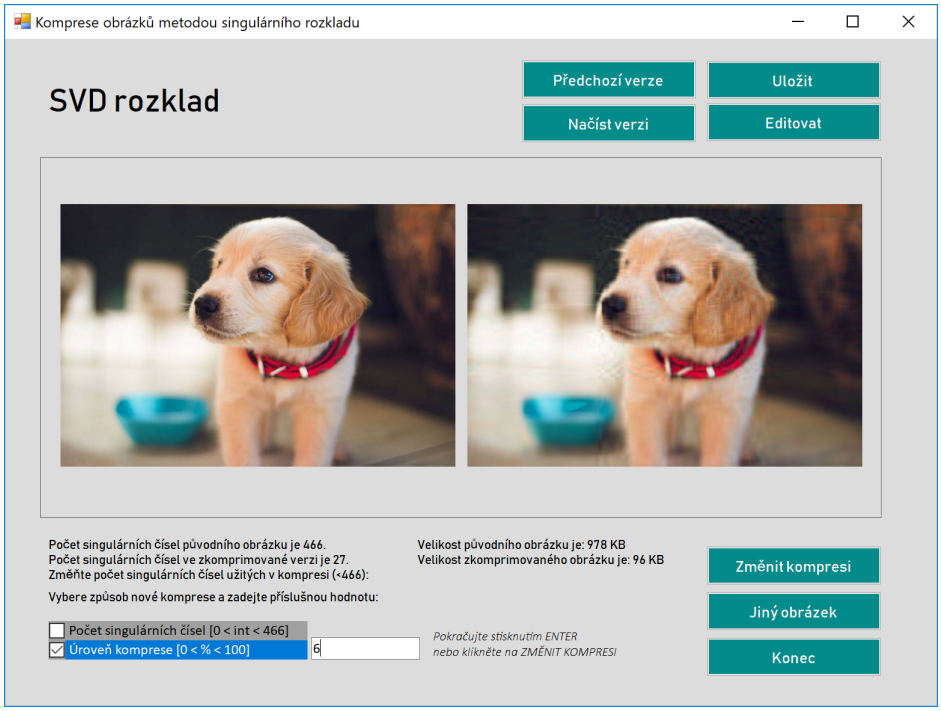
Zkomprimováno

Obrázek je úspěšně zkomprimovaný a my tak můžeme vidět dva `PictureBox`y, vlevo s originálním obrázkem a vpravo se zkomprimovaným. Následuje i jejich porovnání co do počtu singulárních čísel a velikostí. Podotýkám, že velikost původního obrázku je zde vypočtena na základě uchovávaného počtu čísel, nejde o velikost původního obrázku v nějaké kompresi (např. `.jpg`). Uživatel má dále možnost úroveň komprese měnit.

- **Změnu komprese** je možno provést pomocí `CheckedListBox`u, který nabízí dvě možnosti. Buď zadat nějaký vybraný počet singulárních čísel z intervalu (0, původní počet singulárních čísel) a nebo zadat percentuální úroveň komprese samozřejmě z intervalu (0, 100). V seznamu je možno vybrat pouze jednu z možností. Samotná hodnota je pak zadávána do `TextBox`u, který je ošetřen tak, že do něj lze zadávat pouze čísla. Pokud bude zadaná hodnota mimo intervaly popsané výše, zobrazí se chybová hláška. Stejně tak se zobrazí v případě, pokud jeden z parametrů (způsob komprese nebo číselná hodnota) nebude vybrán/zadán. Následně může uživatel v rámci `TextBox`u zmáčknout `ENTER` nebo pokračovat stiskem tlačítka **Změnit kompresi**. Po provedení všech výpočtů se vracíme zpět do současného stavu, pouze se změnou úrovně komprese, tedy změnou obrázku v pravém `PictureBox`u a příslušných popisných charakteristik.

- pokud je k dispozici více než jedna verze obrázku, po kliknutí na tlačítko **Předchozí verze** se zobrazí seznam verzí ve formátu `Verze_x`, po vybrání jedné z nich přejde program do stavu *Zkomprimovano* s načtenou příslušnou verzí.

- tlačítkem **Editace** se zobrazí nové okno, ve kterém se program zeptá, zda chce uživatel editovat původní či zkomprimovanou verzi obrázku. Po výběru program přejde do stavu *Editace*.

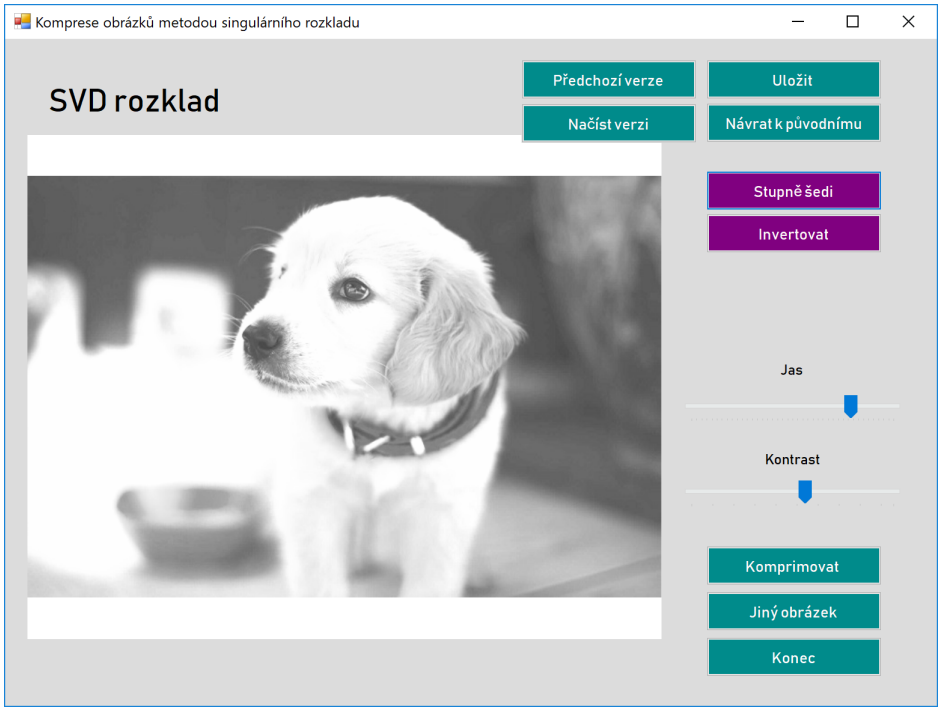


Editace

Uživatel má možnost několik jednoduchých úprav:

- Jas
- Kontrast
- Stupně šedi
- Invertovat

Kdykoliv v průběhu se může vrátit k původní verzi stiskem tlačítka **Návrat k původnímu**, které stornuje všechny úpravy. Stiskem tlačítka **Komprimovat** přecházíme do stavu *Zkomprimováno* se zkomprimovaným vyeditovaným obrázkem. Zbytek tlačítek již byl popsán výše a funguje stejným způsobem.



4 Implementace SVD

Jelikož se problém hledání singulárního rozkladu matice dá převést na problém hledání vlastních čísel matice, a ten se zase dá převést na hledání kořenů polynomů, z Abel-Ruffiniho věty víme, že neexistuje řešení takového problému vyjádřitelné v radikálech a tedy se musíme spokojit s iteračními metodami. Z toho plyne značný počet problémů, především z důvodu obtížnosti algoritmů a stran numerické stability. Proto jsem byl nucen použít volně dostupnou knihovnu `MathNet.Numerics.LinearAlgebra`, která v sobě SVD rozklad obsahuje (a jeho přesnost je pro potřeby programu více než uspokojující).

5 Popis nejdůležitějších tříd a metod

Vše je popsáno ve zdrojovém kódu ve formě komentářů, zde popíšu pouze nejdůležitější z nich.

Verze

Třídu `Verze` užívám pro ukládání jednotlivých verzí v průběhu manipulací s nahraným obrázkem. Pro každou takovou úpravu se pomocí konstruktoru vytvoří její instance, jejími nejdůležitějšími členy, které zde zmíním, jsou `string source`, tedy odkaz na původní obrázek a `int pouzity_pocet_SC` udávající počet singulárních čísel užitých v kompresi. Pomocí metody `Uloz()` se daná instance uloží do již zmíněné složky `Verze` ve formátu `yyyy-dd-MM_hhmmss.txt` (rok, den, měsíc a čas), kde v textovém souboru bude na první řádce cesta k obrázku a na druhé počet singulárních čísel. Instance se dokáže i načíst pomocí metody `Zkomprimovany Nacti(string filePath)`, která vrací příslušný zkomprimovaný obrázek. Je samozřejmě nutné všechny výpočty a SVD rozklad spočítat znova, protože jediné údaje, které máme k dispozici, jsou počet singulárních čísel a původní obrázek.

SVD

Jedná se o třídu uchovávající SVD rozklad dané matice. Obsahuje členy structu `Matice` a to `U, SIGMA, V`, kde `U` je unitární matice levých singulárních vektorů, `SIGMA` je diagonální (obecně obdélníková) se singulárními čísly původní matice na diagonále a `V` je unitární matice pravých singulárních vektorů. Konstruktor bere za parametr dvourozměrné pole (matici), kterou za pomocí knihovny `MathNet` rozloží a vytvoří příslušné 3 matice. Další metodou je `void ZmenPocetSinglarnichCisel(int d)`, která za argument bere nový počet singulárních čísel a upraví dané 3 matice tak, aby používaly pouze daných d singulárních čísel. Vlastně upraví pouze jejich dimenze (rozsah, se kterým pracujeme) a pole prvků nechává nedotčené. Musí zůstat zachované pro případ, že bychom v dalším kroku chtěli pracovat s vyšším počtem s.č.

RGB

Je třída na ukládání R, G, B složek obrázku v podobě 3 dvourozměrných polí, dále zná i šířku a výšku obrázku. Konstruktor bere za argumenty všechny členy a pouze je přiřazuje. Dále máme metody `RGB VytvorRGB(Bitmap bmp)`, která z bitmapy daného obrázku vytvoří instanci RGB (užíváme ji pří načtení původního obrázku pro vytvoření RGB matic) a `Bitmap VytvorObrazek(RGB rgb)`, která naopak z instance RGB vytvoří bitmapu (tu zase používáme, když máme vypočtené aproximace RGB matic a chceme vykreslit zkomprimovaný obrázek).

Obrazek

Uchovává šířku, výšku, zdroj obrázku, bitmapu a jeho RGB. Třída je abstraktní a neexistují tedy její instance, dědí od ní však třídy

- **Komprimovany** která představuje původní obrázek. Konstruktor na základě zdroje obrázku vytvoří instanci s tím, že pokud je jeden z rozměrů větší než 900px, obrázek zmenší (přímo úměrně). Je to nutné kvůli SVD rozkladu, výpočty by pak trvaly příliš dlouho (číslo 900 nalezeno heuristicky).
- **Zkomprimovany** uchovává navíc 3-rozměrné pole SVD rozkladů, počet původních singulárních čísel obrázku a počet užitých s.č. v dané zkomprimované verzi. Konstruktor bere za parametry původní komprimovaný obrázek a úroveň komprese (číslo, které pokud je < 1 chápe jako percentuální změnu původního počtu s.č., pokud je > 1 tak jako nový počet s.č.). Základní úroveň komprese je nastavena na 20%. Vytvoří instanci SVD pro jednotlivé složky RGB a tyto aproximované $U\Sigma V^T$ pak pro každou složku vynásobí a užitím `Bitmap VytvorObrazek(RGB rgb)` vytvoří danou bitmapu. Metoda `Zkomprimovany Zmena_komprimace(int nova_SC)` příslušný již zkomprimovaný obrázek změní dle zvoleného nového počtu s.č.
- **Editovany** je třída editovaného obrázku, která uchovává informace o jeho jasu, kontrastu, zda je ve stupních šedi či invertovaný a dále původní obrázek.

6 Závěr

Závěrem bych řekl, že i přes neošetřené nedostatky, které se pravděpodobně v programu najdou, se jedná o nejrozsáhlejší a nejkomplexnější program, jaký jsem kdy napsal. V průběhu jeho psaní jsem si utvrdil a vyzkoušel v praxi užití metod OOP, které jsme probírali na přednášce, ale třeba neměli příležitost je implementovat. Takových věcí, jak se nakonec ukázalo, bylo více, než jsem čekal. Jedna z výzev bylo například responsivní GUI v průběhu SVD rozkladu, který pro větší obrázky trval celkem dlouho a bylo tedy třeba užít vláken (implementováno pomocí `BackGroundWorker`). Naopak, co mě mrzí a nepodařilo se byla vlastní implementace SVD algoritmu, který jsem po dvou dnech snahy musel vzdát - byl to zkrátka moc velký oříšek a i doporučení nalezená na různých fórech zněla "radši nezkoušejte".