

Návrh konvolučního filtru pomocí evolučních algoritmů

Projekt MAPV

Vojtěch Vladýka a Martin Sehnoutka

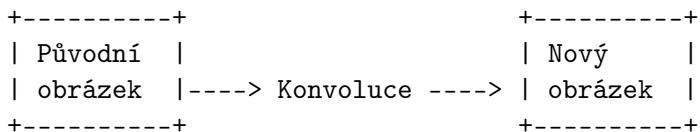
1. května 2016

Obsah

1	Zadání	3
2	Teoretický rozbor	3
2.1	Genetické algoritmy	3
2.2	Selekce	3
2.3	Křížení	4
2.4	Mutace	4
2.5	Zpracování obrazu	4
3	Implementace	5
3.1	Spouštění testů	6
4	Zhodnocení dosažených výsledků	7
4.1	Vizuální porovnání výsledků	7
4.2	Porovnání časové náročnosti	7
5	Závěr	8
6	Výsledná práce online	9
7	Použitá literatura	9
8	Přílohy	10

1 Zadání

Cílem projektu je navrhnout algoritmus pro indukci konvoluční masky na základě dvou šedo-tónových nebo barevných obrázků. Pro hledání řešení jsou použité genetické algoritmy. Jako prostředek realizace je zvolena knihovna OpenCV a programovací jazyk C++.



2 Teoretický rozbor

Práci je možné rozdělit do dvou základních oblastí. První jsou genetické algoritmy, které byly použity pro hledání optimálního řešení. Druhá oblast je zpracování obrazu, jelikož se zabýváme porovnáváním dvou obrázků, námi vytvořeného a zadánoho.

Tato sekce je pouhým náznakem principu daných algoritmů, jelikož předpokládáme, že toto není hlavním cílem projektu.

2.1 Genetické algoritmy

Jde o heuristický postup, který se snaží aplikovat evoluční principy na optimalizační úlohu. Na rozdíl od analytického výpočtu nepředpokládáme na výstupu exaktní řešení, proto tyto metody používáme v případě, že neznáme analytický postup jak úlohu vyřešit. Celý algoritmus se dá rozdělit na několik kroků, kterým říkáme genetické operátory. Jsou to selekce, křížení a mutace. Tyto operátory potom pracují nad množinou jedinců, kterou nazýváme populace. Jako poslední termín zavádíme fitness funkci, jenž slouží k ohodnocení "výkonnosti" jedince.

Schéma genetického algoritmu:

1. Vytvoření počáteční populace
2. Ohodnocení fitness funkcí
3. Ukončovací podmínka
 - +--> Splněna: Skok na konec
 - +--> Nesplněna: Pokračuj
4. Selekce
5. Křížení
6. Mutace
7. Elitismus - nejlepšího jedince ponech bez změny
8. Skok na fitness

2.2 Selekce

Naivní řešení by bylo vzít nejlepší jedince ze současné generace a stvořit generaci novou. Tímto postupem bychom se ovšem brzy dostali do lokálního minima, což není žádoucí. Proto zavádíme různé metody selekce, které mají tomuto jevu předcházet tak, že preferují lepší jedince, ale na bázi určité náhody dávají šanci i jedincům horším. V našem projektu implementujeme tyto metody selekce:

Vážená ruleta Tato metoda náhodně vybírá jednoho jedince z populace, nicméně lepší jedinci mají vyšší pravděpodobnost. Tato pravděpodobnost je určena poměrem jejich fitness funkcí.

Poziční selekce Funguje stejně jako vážená ruleta pouze místo fitness funkce používá pro určení poměrů pravděpodobnosti pozici jedince v pořadí od nejlepšího po nejhoršího. Tato metoda na rozdíl od rulety dává šanci jedincům i v případech, že jejich fitness je podstatně menší než nejlepších jedinců. Tím lze opět eliminovat možnost konvergence do lokálních minim.

Turnaj Metoda náhodně vybírá dvojice, ze které vybírá toho lepšího jedince.

2.3 Křížení

Metody křížení slouží k vytvoření dvojice potomků z dvojice původních jedinců.

Jednoduché Geny potomků jsou náhodně vybrány z matic rodičů. Příkladem implementace může být vygenerování náhodné matice binárních hodnot, která určí, zda se daný gen vezme z prvního nebo druhého rodiče.

Konvexní Upravuje předcházející metodu tak, že výsledný gen není vybrán pouze ze dvou hodnot genu rodičů, ale z intervalu těchto hodnot. Nicméně i tato metoda má nevýhodu v tom, že může konvergovat do lokálních maxim, proto zavádíme následující metodu.

BLX- α Metoda křížení BLX- α [3] je založená na principu výběru hodnoty z rozsahu většího, než je rozsah rodičů, takže se může potomek vyhnout lokálnímu extrému.

Výsledná hodnota jednoho prvku po křížení je určena takto:

$$u = \min(x, y) - \alpha\Delta; \max(x, y) + \alpha\Delta > \quad (1)$$

kde α je parametr rozšíření intervalu, Δ je absolutní hodnota rozdílu hodnot rodičů. Hodnota u je hodnota nového potomka která je náhodně vybraná z intervalu.

2.4 Mutace

Swap Mutace metodou swap je pravděpodobně nejjednodušší možné řešení. Spočívá ve výměně několika dvojic prvků mezi sebou.

Dynamická Dynamická mutace na rozdíl od swap bere v potaz počet iterací a s blížícím se koncem běhu genetického algoritmu postupně snižuje velikost mutace. Tento způsob lze popsat rovnicí:

$$\Delta(t, y) = y \left(1 - r^{1 - \frac{t}{T}^B} \right) \quad (2)$$

Kde y je jeden konkrétní gen, r je náhodné číslo z intervalu $< 0, 1 >$, t je pořadí iterace, T je maximální možný počet iterací a B je empirická konstanta s doporučenou hodnotou 5.

2.5 Zpracování obrazu

Chybová funkce Slouží k ohodnocení podobnosti dvou obrazů, tzn. v našem případě je použita jako fitness funkce pro genetický algoritmus.

Kvadrát rozdílu obrazových funkcí Jedná se o naivní implementaci porovnání dvou obrazů, která se dá popsat tímto vzorcem:

$$error = \sqrt{\sum_{\forall r} \sum_{\forall c} (O_{r,c} - Y_{r,c})^2} \quad (3)$$

Kde O je vzor upravený neznámým konvolučním jádrem a $Y = I * k$ je vzor po konvoluci s kandidátním řešením. r, c jsou obrazové souřadnice.

Tato funkce ovšem vykazuje opačné chování, než jsme od fitness funkce chtěli, klesá s lepším kandidátem a roste s horším. Proto jsme vzorec upravili, aby vyhovoval našim potřebám.

$$fitness = \sqrt{\frac{rows \cdot cols}{\sum_{\forall r} \sum_{\forall c} (O_{r,c} - Y_{r,c})^2}} \quad (4)$$

Nicméně ani tato fitness funkce nebyla vyhovující neboť dostatečně nerozlišovala dobré a špatné kandidáty, proto jsme ji upravili do finální podoby:

$$fitness_{new} = e^{5 \cdot fitness} \quad (5)$$

Kde hodnota 5 byla určena experimentálně.

SSIM SSIM (Structural Similiarity Image Assesment) je metoda spadající pod Image Quality Assesment (IQA), která vyjadřuje podobnost dvou obrazů na škále od -1 pro naprosto odlišné obrazy po 1 pro identické obrazy. Tato metoda bere vychází ze způsobu jak obraz vnímá člověk, a to že sleduje strukturu obrazu.

Definice podle původní práce[2] je následující:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (6)$$

kde μ je průměr hodnot pixelů obrazu, σ je standartní odchylka obrazu. Konstanty C_1 a C_2 jsou odvozeny od rozsahu hodnot pixelu L , v našem případě 255 a konstanty $k << 1$. V práci jsou použity hodnoty $k_1 = 0,01$ a $k_2 = 0,03$. Posléze se konstanta C počítá takto: $C = k \cdot L$. V naší implementaci, která byla převzatá ze zdroje [?] jsou použity konstanty $C_1 = 6,5025$ a $C_2 = 58,5225$. Průměr μ je řešen pomocí gaussiánu 11x11. Standartní odchylka σ je realizována jako rozdíl gaussiánu a μ . Poté je všechno ve formě matice, tím pádem je potřeba pro skalární výsledek spočítat průměrnou hodnotu matice.

Tato metoda vykazuje vyšší výpočetní náročnost ale zároveň i lepší výsledky, obzvláště u nedostatečného množství významných bodů.

3 Implementace

Jak již bylo zmíněno, implementace byla provedena v jazyku C++ a to s pomocí knihovny OpenCV. Samozřejmě jsme nepoužili pouze tuto knihovnu, ale rovněž STL, Google test framework, boost atd. I přes použitý jazyk, jsme implementaci neprovedli čistě objektově. Většina základních funkcí (jako jsou na příklad genetické operátory) jsou definovány jako prosté funkce, u kterých byl důraz kladen na ortogonalitu a modularitu, abychom mohli jednoduše zkoušet různé kombinace námi implementovaných operátorů. Nicméně pro pohodlné použití jsme vytvořili jednu třídu příhodně nazvanou Worker, která zapouzdřuje celý genetický algoritmus, ta obsahuje parametry generace a ukazatele na použité operátory.

Testování programu je možné pomocí dvou spustitelných souborů. První z nich je „gekon_run“, který jako parametry přebírá velikost konvolučního jádra a vstupní obrázky, nicméně nelze měnit parametry algoritmu neboť jsou, jak by se řeklo v IT žargonu, „hardcoded“ ve zdrojovém souboru. Tento problém řeší druhý spustitelný soubor, který je pojmenován „automated_tests“ a přebírá konfigurační soubor napsaný v jazyku TOML, ten pak definuje parametry testů jako jsou např. velikost generace, použité operátory a počet vláken.

Struktura projektu:

```
. -- bin -- spustitelné soubory
|- docs -- dokumentace
|- gekon -- zdrojové kódy
|- samples -- testovací vzory
|- tests -- unit testy
```

Ukázka rozhraní genetických operátorů:

```
typedef std::function<std::vector<candidate_t>(candidate_t, candidate_t)> crossover_fcn_t;
typedef std::function<void(candidate_t&, unsigned int, unsigned int)> mutation_fcn_t;
typedef std::function<double(tr_sample_t, candidate_t)> fitness_fcn_t;
typedef std::function<population_t(population_t)> selection_fcn_t;
```

A implementace jednoho konkrétního operátoru:

```
void m_dynamic(candidate_t &X, unsigned int t, unsigned int T) {
    /*
     * Dynamic mutation (Michalewicz):
     * Defined as  $\Delta(t, y) = y \left(1 - r^{1 - \frac{t}{T}^B}\right)$ 
     * where  $B = 5$ 
     */
    int rows = X.rows;
    int cols = X.cols;
    for (int r = 0; r < rows; ++r) {
        for (int c = 0; c < cols; ++c) {
            if(random(0,1) < m_swap_prob) {
                X.at<ker_num_t>(r,c) = X.at<ker_num_t>(r,c)*
                    (1-pow(random(0,1), pow(float(1)-float(t)/T, m_dynamic_B)));
            }
        }
    }
}
```

3.1 Spouštění testů

Program během výpočtu vypisuje průběžné výsledky. Ideální je výstup (stdout) přesměrovat do souboru a následně filtrovat požadované informace. Jako příklad uvádíme srovnání časové náročnosti běhu jednotlivých algoritmů:

```
$ cat example1.log | grep '\[TEST\]' | grep 'Time elapsed\|Test name'
[TEST] Test name: tournament-blx_a-ssim-swap
[TEST] Time elapsed: 1021
[TEST] Test name: tournament-simple-mse-dynamic
[TEST] Time elapsed: 579
[TEST] Test name: tournament-blx_a-ssim-dynamic
[TEST] Time elapsed: 1050
[TEST] Test name: tournament-blx_a-mse-swap
[TEST] Time elapsed: 597
```

```
[TEST] Test name: tournament-simple-mse-swap
[TEST] Time elapsed: 611
[TEST] Test name: tournament-simple-ssim-dynamic
```

4 Zhodnocení dosažených výsledků

Během testování jsme vyzkoušeli všechny možné kombinace námi napsaných operátorů nad všemi zadanými vzory. Časová náročnost byla přibližně jeden den na výkonné domacím počítači při fixní délce běhu 500 iterací.

4.1 Vizuální porovnání výsledků

Při pohledu na výsledky lze usuzovat vhodnost některých metod pro tuto úlohu. Např. fitness funkce kvadrátu odchylky dává stabilně horší výsledky než SSIM, to samé lze tvrdit o mutaci metodou swap oproti dynamické mutaci, jejíž výsledky jsou z pravidla lepší. Naopak vliv selekce a křížení je spíše menší. Ten by se projevil hlavně na rychlosti konvergence k výsledku.

Obrázek 2 ukazuje výsledky různých kombinací operátorů. Výsledek 2c je vizuálně nejshodnější, zatímco výsledek 2d se v jistých bodech odlišuje více (byť to není v těchto malých náhledech vidět). Naopak výsledek 2e je evidentně špatně, ačkoli možná by se po delším čase dostal též k výsledku.

Jak bude vidět na dalších ukázkách, kombinace operátorů Roulette pro selekci, BLX- α pro křížení, SSIM pro fitness a Dynamická mutace se jeví jako nejlepší kombinace univerzálně pro všechny zadанé vzory.

Zajímavá situace je vidět na obrázku 3. Správný výsledek je opět na obrázku 3c, ale rozdíl mezi ním a obrázkem 3d je pouze v použitém operátoru mutace, kde v prvním případě je použit lepší operátor Dynamické mutace zatímco ve druhém případě je operátor Swap. Na obrázku 3e je evidentně špatný výsledek, který pouze zvýraznil hrany.

Výsledek na šedotónovém obrázku 4 je obecně lepší, ale i zde jsou výsledky, které jsou naprostě špatně. Správný výsledek je opět na obrázku 4c se stále stejnou kombinací jako všechny předchozí příklady. Zajímavá situace nastala u výsledků 4d a 4e. První jmenovaný je poměrně obstojný. Nemá na správný výsledek, protože je poněkud tmavší, ale řešení je to správné. Ale od špatného výsledku na druhém zmíňovaném se liší pouze způsobem křížení, kde v prvním případě byla použita metoda Simple crossover a u druhého Convex crossover. V tomto případě by bylo zajímavé nechat testy běžet opakováně, zda se tento výsledek projeví znovu.

Zajímavá změna je na obrázku 5. Zde ve vizuální podobnosti vyšel lépe vzorek využívající selekci Turnaj oproti Ruletě. Opět jako několikrát předtím, špatný výsledek je na 5e.

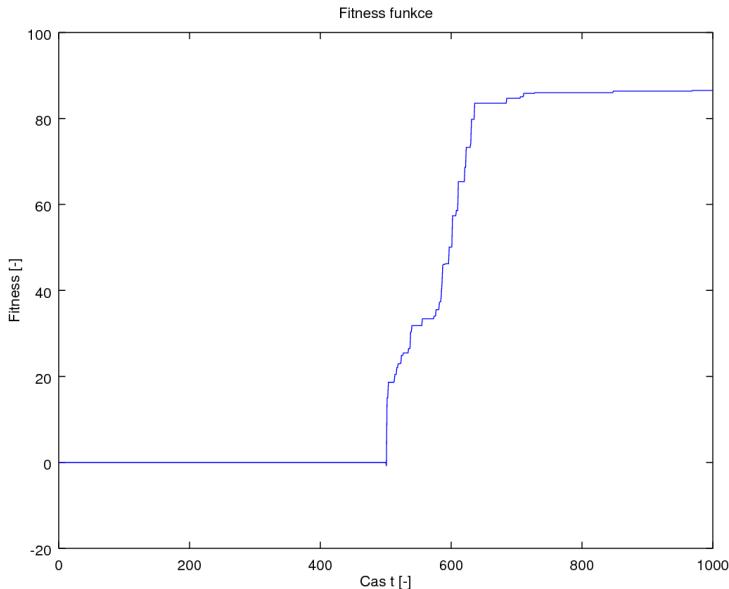
Časový průběh fitness funkce vzorového běhu algoritmu je zobrazen na obrázku 1. Jak je vidět, používáme elitismus, abyhom zajistili neklesající průběh.

4.2 Porovnání časové náročnosti

Tabulka 1 uvádí vliv jednotlivých operátorů na celkovou časovou náročnost; tzn. nejedná se o náročnost pouze daného operátoru, ale celého algoritmu s použitím tohoto operátoru.

Jak je vidět, vliv na celkový čas mají prakticky pouze metody fitness funkce. U genetických operátorů by bylo třeba měřit pouze čas provádění daného operátoru.

Obrázek 1: Průběh fitness funkce v závislosti na čase



Tabulka 1: Porovnání časové náročnosti

Metoda	Průměrný čas [s]
SSIM	448
mse	246
Turnaj	347
Poziční selekce	346
Ruleta	348
Jednoduché křížení	349
Konvexní křížení	347
BLX- α	345
Mutace swap	346
Mutace dynamická	347

5 Závěr

Cílem projektu bylo navrhnut algoritmus pro indukci konvoluční masky při znalosti původního a výsledného obrázku. Toto zadání se podařilo splnit beze zbytku pomocí implementace genetických algoritmů. Jako nejvhodnější kombinace se jeví použití SSIM algoritmu pro fitness funkci, selekce pomocí rulety, BLX- α pro křížení a mutace dynamická. Tato kombinace operátorů vykazuje stabilně nejlepší výsledky při shodných časech, který jsou sice obecně delší oproti jednodušším operátorům, ale tato cena je daná za jejich výkonnost.

Důležitým faktorem je výkonnost právě fitness funkce. Ta musí být schopná i z relativně špatného výsledku co nejlépe určit, zda je aktuální výsledek lepší nebo horší. Proto jsou obecně lepší metody které porovnávají strukturální podobnost jako například SSIM.

Dalším kritickým faktorem je mutace. Při pokusech bez mutace se algoritmus zasekával v lokálních extrémech a správného výsledky je možné dosáhnout pouze při správné konstelaci první generace. Na druhou stranu nechávat mutovat jedince po celou dobu výpočtu není vhodné

z důvodu možného odchýlení od správného výsledku. Tato situace nastala právě u obrázku 3d, který měl mutaci Swap, která má stejnou pravděpodobnost mutace po celou dobu výpočtu oproti výsledku na obrázku 3c, který má mutaci dynamickou. Té se pravděpodobnost mutace snižuje jak se přibližuje správnému výsledku.

Vliv operátora křížení je zásadní pro rychlosť konvergence k výsledku. Možnosti jsou několik, základní jsou ale vždy omezené na rozsah hodnot v genomu rodičů. To ovšem sebou nese riziko konvergence do lokálního extrému. Proto se zavedla metoda BLX- α , která vybírá hodnotu z rozsahu rodičů, který je rozšířený o konstantní velikost na obě strany. Díky tomu se může vyhnout lokálnímu extrému. Právě tato metoda vykazuje nejlepší výsledky při křížení.

Vliv operátora selekce se projevuje převážně v rychlosti konvergence ke správnému výsledku. V našem případě nejlepších výsledků dosahovala ruleta i přes svou šanci konvergovat k lokálním extrémům.

6 Výsledná práce online

Výsledek práce i s kompletní historií postupu lze najít v repozitáři na Githubu:

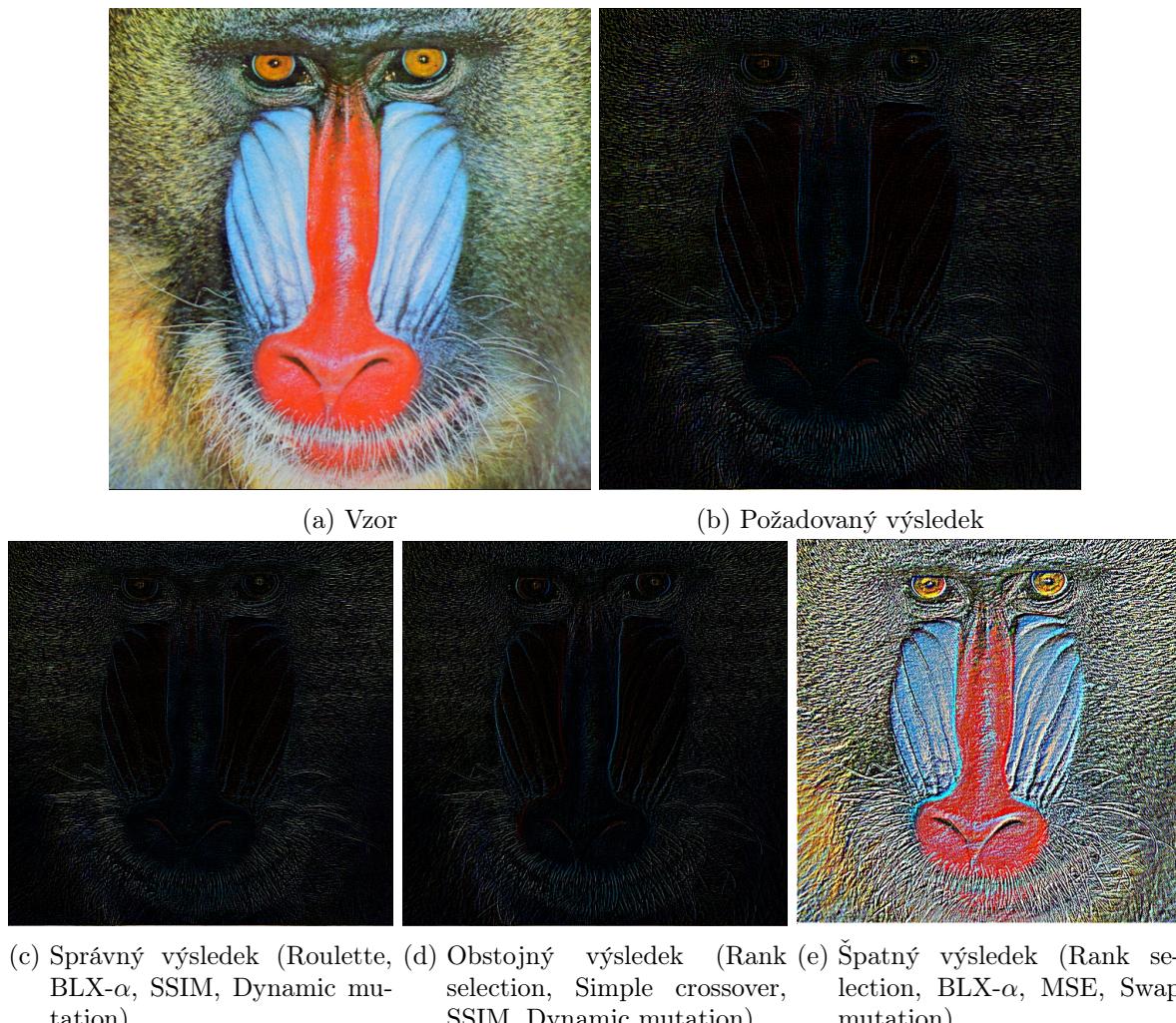
<https://github.com/argorain/GeKon>.

Předpokladem pro úspěšný překlad programu je C++ ve verzi 14, CMake a OpenCV verze 3 a vyšší.

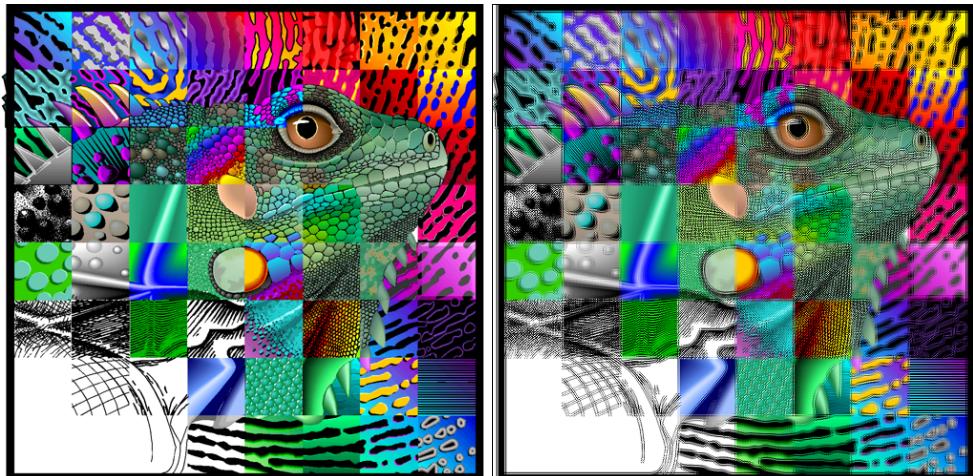
7 Použitá literatura

- [1] HONZÍK, Petr. *Strojové učení* [online]. Brno, 2006. Dostupné také z: www.vutbr.cz
- [2] WANG, Zhou, Alan C. BOVIK, Hamid R. SHEIKH a Eero P. SIMONCELLI. *The SSIM Index for Image Quality Assessment*. The Center for Neural Science at NYU [online]. New York: NY University, 2003 [cit. 2016-04-18]. Dostupné z: <http://www.cns.nyu.edu/lcv/ssim/>
- [3] *Blend (BLX) Crossover*. Tomasz Gwiazda e-books [online]. Warsaw: Warsaw University, 2006 [cit. 2016-05-01]. Dostupné z: <http://www.tomaszgwiazda.com/blendX.htm>

8 Přílohy

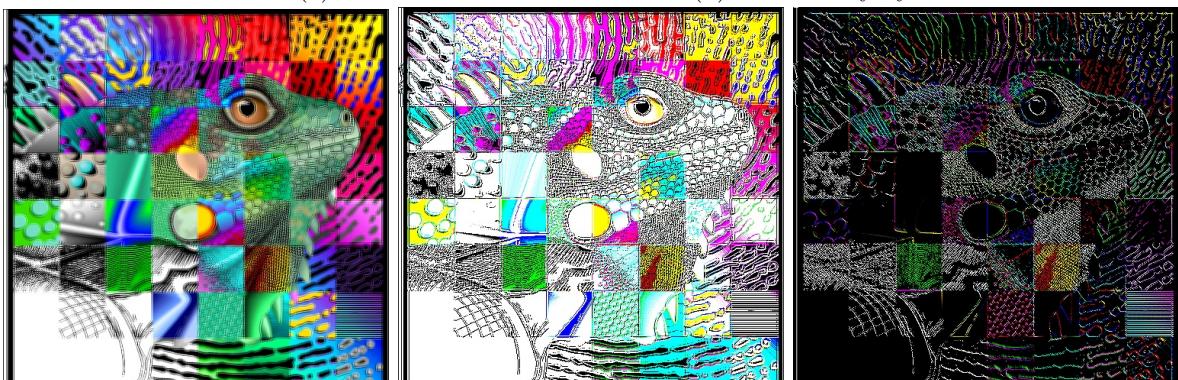


Obrázek 2: Srovnání na vzoru 1



(a) Vzor

(b) Požadovaný výsledek



(c) Správný výsledek (Roulette, BLX- α , SSIM, Dynamic mutation)

(d) Špatný výsledek (Roulette, BLX- α , SSIM, Swap mutation)

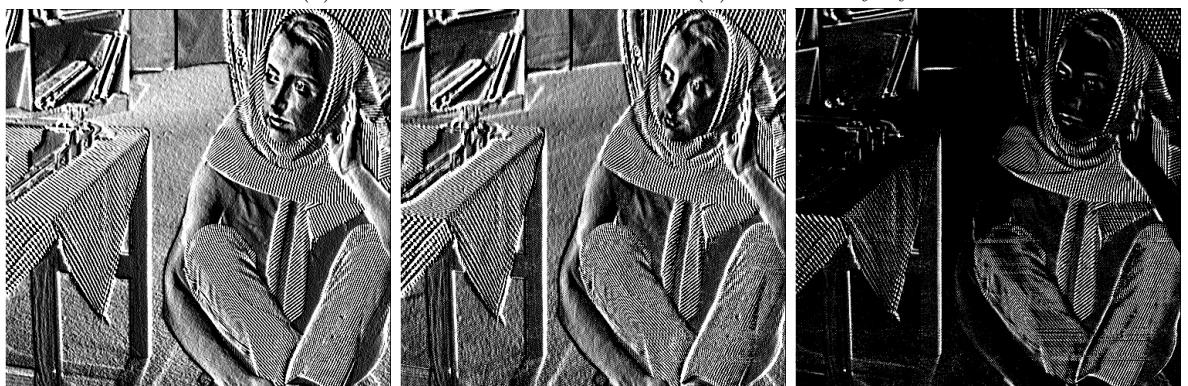
(e) Špatný výsledek (Rank selection, Convex, MSE, Swap mutation)

Obrázek 3: Srovnání na vzoru 2



(a) Vzor

(b) Požadovaný výsledek

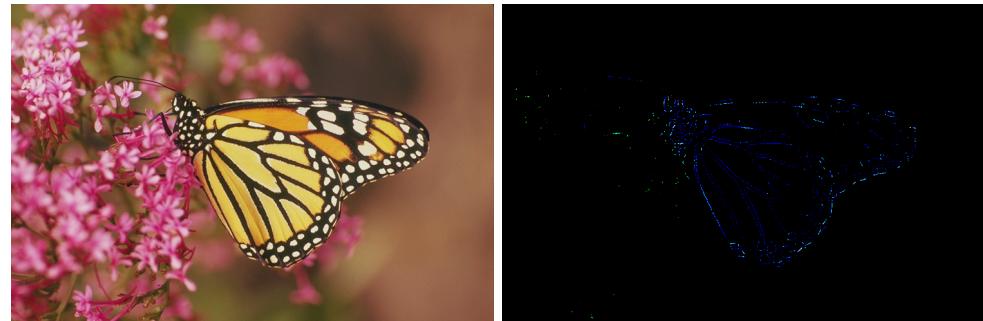


(c) Správný výsledek (Roulette, BLX- α , SSIM, Dynamic mutation)

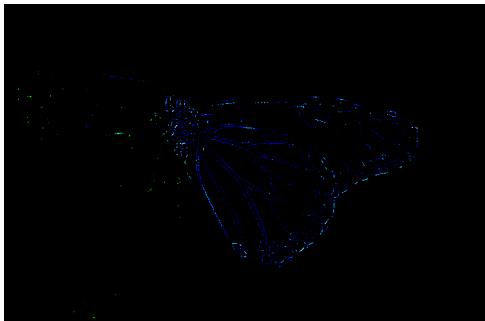
(d) Obstojný výsledek (Rank selection, Simple, MSE, Swap mutation)

(e) Špatný výsledek (Rank selection, Convex, MSE, Swap mutation)

Obrázek 4: Srovnání na vzoru 6



(a) Vzor



(b) Požadovaný výsledek



(c) Správný výsledek (Tournament mutation (!), BLX- α , SSIM, Dynamic mutation)

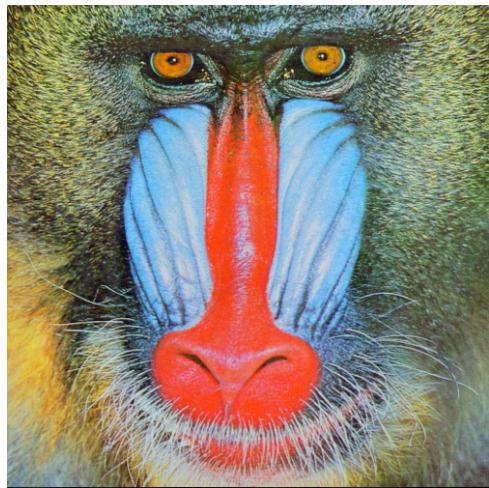


(d) Správný výsledek (Roulette, BLX- α , SSIM, Dynamic mutation)

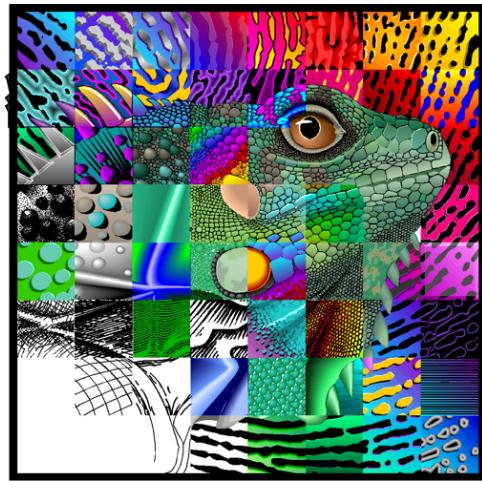


(e) Špatný výsledek (Roulette, Convex, MSE, Swap mutation)

Obrázek 5: Srovnání na vzoru 3



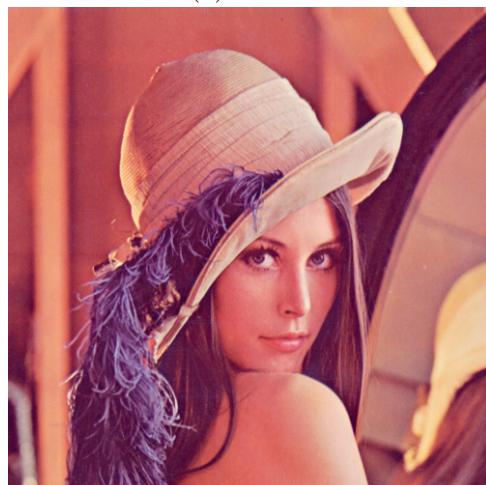
(a) Vzor 1



(b) Vzor 2



(c) Vzor 3



(d) Vzor 4



(e) Vzor 5



(f) Vzor 6

Obrázek 6: Vzory