VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

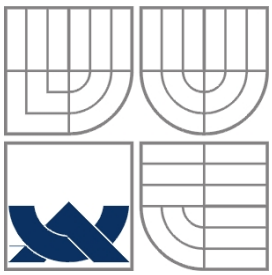
SEGMENTACE OBRAZU POMOCÍ NEURONOVÉ SÍTĚ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

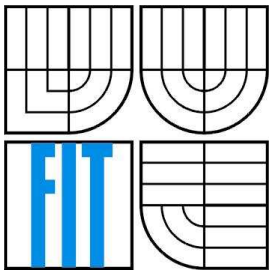
AUTOR PRÁCE
AUTHOR

BC. SOŇA JAMBOROVÁ

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SEGMENTACE OBRAZU POMOCÍ NEURONOVÉ SÍTĚ

NEURAL NETWORK BASED IMAGE SEGMENTATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. SOŇA JAMBOROVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PAVEL ŽÁK

BRNO 2011

Abstrakt

Tato práce se zabývá návrhem softwaru pro segmentaci obrazu pomocí neuronové sítě. Definuje nezbytné základní pojmy v této problematice. Zaměřuje se hlavně na přípravu obrazových informací pro segmentaci pomocí neuronové sítě. Popisuje a porovnává různé přístupy k segmentaci obrazu.

Abstract

This work is about suggestion of the software for neural network based image segmentation. It defines basic terms for this topics. It is focusing mainly at preperation imaging information for image segmentation using neural network. It describes and compares different aproaches for image segmentation.

Klíčová slova

Segmentace obrazu, neuronová síť, neuron, Kohonenova síť, radiální bázová funkce, k-means algoritmus, SOM, self-organized map, mediánový filtr, Gaussův šum, Gaussův filter, Gaborova funkce, HSV, RGB, klasifikace, shlukování.

Keywords

Image segmentation, neural network, neuron, Kohonen neural net, radial base function, k-means algorithm, SOM, self-organized map, median filter, Gaussian noise, Gaussian filter, Gabor function, HSV, RGB, classification, clustering.

Citace

Jamborová Soňa: Segmentace obrazu pomocí neuronové sítě, diplomová práce, Brno, FIT VUT v Brně, 2011

Segmentace obrazu pomocí neuronové sítě

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením Ing. Pavla Žáka.
Další informace mi poskytl Ing. Miroslav Švub.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Soňa Jamborová
27.7.2011

Poděkování

Děkuji vedoucímu diplomové práce Ing. Pavlu Žákovi a konzultantovi Ing. Miroslavu Švubovi za udělené rady a pomoc při vypracovávání této práce.

Obsah

1 Úvod

V dnešní době se lidé snaží čím dál více využívat počítačové systémy v běžných úkonech. Rádi by jim dali jistou míru nezávislosti a stability v činnostech, které člověk dělá s rutinou, ale jejichž provedení zabírá relativně hodně času nebo existuje mnoho ukazatelů, které mohou ovlivnit lidský úsudek. Proto se nejčastěji tyto systémy využívají hlavně v oborech, ve kterých záleží na lidském životě (lékařství, automobilismus, …). Dnešní počítačové systémy jsou založeny především na tom, aby daly prostředí určité objektivní měřítko, podle kterého se může člověk rozhodovat (vzdálenost překážky při parkování automobilu, krok pro posun robotického ramena při operaci apod.). Je však otázka času, kdy se počítačové systémy posunou opět na vyšší úroveň.

Klíčovým úkolem počítačových systémů je získávání informací o prostředí, ve kterém se daný systém pohybuje. Vezmeme-li si inspiraci v člověku, pak zjistíme, že se rozhoduje především na základě zraku. Zrakem vnímáme zhruba 80 % všech informací, které obdrží lidský mozek. Možná proto se říká, že jeden obrázek vydá za 1000 slov. Počítačové systémy dokáží podobně jako lidské oko reprezentovat obrazové informace z okolí. Pro statické scény využívá obrazových formátů nebo pro dynamické scény různé formáty videa.

Živé organismy po celém světě zvládají segmentaci obrazu ve zlomcích sekund. Tuto službu (mimo jiné důležité funkce) jim zajišťuje nervová soustava, která zpracovává i signály ze zrakových orgánů. V přírodě je hodně vjemů subjektivních. Každý organismus jinak vnímá stejné vjemy (psi – černobílé vidění; prvoci – světločivné skvrny; poruchy vidění u člověka – šedý zákal, barvoslepost apod.). Stroje by mohly přinést do oblasti vidění jistou dávku objektivity. Nabyté znalosti by se mohly přenášet po celém světě. Neuronová síť má pro tyto účely dobré předpoklady. Snaží se převést znalosti o nervové soustavě do digitální podoby a zajistit tak její přenositelnost.

Má práce bude zaměřena na zpracování obrazů, protože je základem i pro zpracování videa. Obraz však postrádá informace týkající se dynamiky pohybu. Zaměřuje se na lokalitu, rozměry a barevné vlastnosti vnímané scény.

Cílem mé diplomové práce je experimentovat s neuronovou sítí pomocí softwaru, který bude pro tento účel vytvořen. Předmětem experimentů bude standardizace práce s neuronovou sítí pro účely segmentace. Dále zjištění optimálních parametrů zvolených algoritmů segmentace pro uspokojivé výsledky a volba referenčního algoritmu, který je dnes běžně využíván. Cílem diplomové práce je určit, zda neuronové sítě mohou přinést lepší výsledky pro zpracování obrazu než dnes užívané metody.

Tento dokument se bude zabývat ujasněním pojmů, začleněním problematiky do širšího kontextu a návrhem vhodného segmentačního systému. Na závěr shrne výsledky experimentů, které byly pomocí něho získány, zařadí daný systém mezi již existující metody zabývající se touto problematikou, uvede jeho přínos pro uživatele a nastíní možnosti dalšího rozvoje.

2 Zpracování obrazu

Zpracování obrazu zahrnuje techniky, které dokáží analyzovat obsah obrazu. Mezi ně řadíme i segmentaci obrazu. Ta využívá základní operace nad obrazem, aby dosáhla stabilního výsledku. Základní operace umožňují zvýraznit nebo utlumit důležité charakteristiky obrazu a extrahovat některé vlastnosti, které nejsou na první pohled patrné, ale pro další zpracování jsou důležité.

Kapitola se bude věnovat definování základních pojmů potřebných při předzpracování obrazu pro jeho následnou segmentaci, která bude zařazena až v dalších kapitolách. Jde především o ujasnění, jaké informace může obraz reprezentovat a metodiku jejich získávání.

2.1 Reprezentace obrazu

Obraz je obecně jakékoliv grafické vyjádření. Obsah obrazu bývá zachycen pomocí optických soustav (fotoaparáty, kamery, oko, světločivné skvrny, …), které využívají optických jevů (odraz světla, lom světla, pohlcování světla, vyzařování světla, …), což umožní rozlišit jednotlivé materiály a tvary znázorněných objektů. Takto pořízený obraz lze dále zpracovávat a přiřadit mu význam pomocí nervové soustavy (přirozené pro člověka a zvířata – subjektivní vjem) nebo počítače (matematický model – objektivní vjem).

V dnešní době stále převládá ztvárnění obrazu ve 2D prostoru. Obraz tedy zachycuje 3D objekty a transformuje je do roviny (např. fotky, ...). Obsah obrazu můžeme vyjádřit pomocí barevného prostoru (C). Existují dva typy obrazu pro libovolný barevný prostor:

- **Analogový obraz** - obraz je definován jako spojitá funkce

$$I : \langle 0, w \rangle \times \langle 0, h \rangle \rightarrow C, \qquad 2.1.1$$

- **Digitální obraz** – obraz je definován jako diskrétní funkce

$$I : \{x \in N; 0 \leq x \leq w\} \times \{y \in N; 0 \leq y \leq h\} \rightarrow C, \qquad 2.1.2$$

kde w je šířka obrazu, h je výška obrazu a C je barevný prostor. Analogový obraz je nevhodný pro počítačové zpracování. Obsahuje nekonečně mnoho hodnot, tudíž nikdy nemůže být celý zpracován v reálném čase. Z toho důvodu bude v práci myšleno pod pojmem *„obraz“* vždy digitální obraz nebo bude implicitně uvedeno, že se jedná o *„analogový obraz“*.

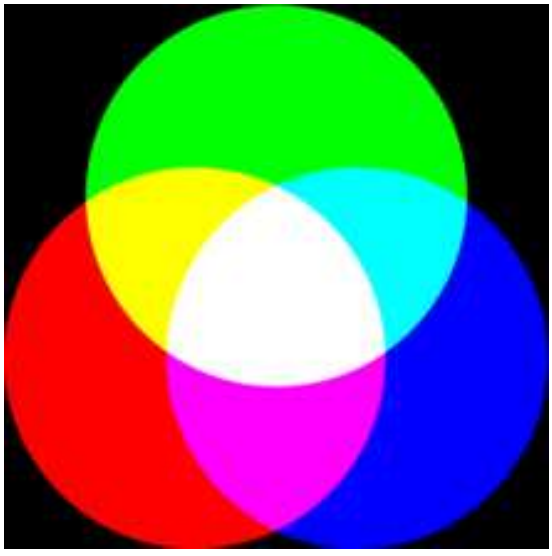
Barevný prostor reprezentuje rozložení barev v obraze. Existuje mnoho způsobů reprezentace barvy. V reálném světě je barva dána různou vlnovou délkou světla, kterou daná optická soustava zachytí. Člověk je schopen vnímat pouze od fialové barvy (400 nm) po červenou barvu (750 nm). Černá je reprezentována vlnovou délkou všech vnímaných složek světla mimo daný interval viditelnosti a bílá barva je dána tím, že obsahuje všechny vlnové délky světla z daného intervalu. Světelný paprsek tedy může obsahovat 0 až n složek, kde $n \in N$.

Pro účely strojového zpracování je tento model nepoužitelný (stejně jako analogový model obrázku). Proto byly vytvořeny barevné modely, které různě deformují barevný prostor:

- **Černobílý model**
  - „vše anebo nic“,
  - obsahuje pouze dvě barvy (černá – 0, bílá – 1),
  - možné reprezentovat na jednom bitu.
- **Model odstínů šedi**
  - obsahuje více hodnot,
  - znázorňuje úrovně intenzity světla,
  - označení C_G - často reprezentován na jednom bajtu, tzn. 256 úrovní šedi.
- **Barevný model**
  - zahrnuje mnoho dalších modelů,
  - převážně se skládají ze tří složek ,
  - např. RGB, HSV, YUV a další,
  - možná konverze různé složitosti z jednoho formátu do druhého,
  - označení C_G^3 - často reprezentováno na třech bajtech, pro každou složku jeden bajt s 256 úrovněmi.

Pro účely diplomové práce jsem si vybrala obrazy využívající barevné modely RGB a HSV.

Barevný model RGB ($C_{RGB} = C_G^3$) využívá tří barevných složek (červená - red; zelená - green; modrá - blue). Jedná se o adaptivní míchání barev, kdy se jednotlivé složky sčítají a vytváří světlo o větší intenzitě. Model je základem dnes všech zobrazovacích zařízeních a stal se standardem pro uchování obrazu v různých formátech (BMP, PNG, …). Dokáže zaznamenat přes 16 milionů odstínů. To umožní zachycení scény v kvalitě, která neomezuje lidské vnímání. Jeho nevýhodou je, že jakékoliv informace o původu objektu je potřeba vypočítat.

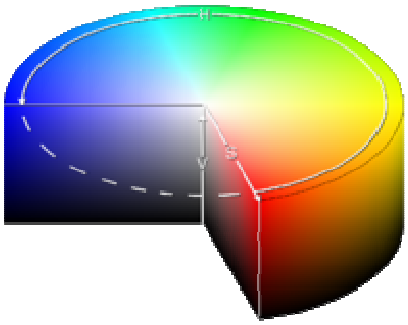


Obrázek 2-1: *Ukázka skládání barev pomocí modelu RGB [9]*

Barevný model HSV ($C_{HSV} = C_G^3$) využívá také tří složek, které dohromady určují výslednou barvu pixelu. Je však konstruován odlišně než RGB. Jednotlivé složky více odpovídají reálnému vnímání barvy objektu (na základě optických jevů):

- **Odstín (hue)**
  - spektrální barva objektu nebo světla procházející objektem,
  - měří se ve stupních: 0° - 360°.
- **Sytost (saturation)**
  - příměs jiné barvy, představuje sytost daného odstínu,
  - měří se v procentech: 0 % – 100 %.
- **Jas (value)**
  - kolik světla barva odráží,
  - měří se v procentech: 0 % - 100 %,
  - samostatná složka jasu (hue = 0, saturation = 0) představuje obraz v odstínech šedi.

Využití tohoto modelu pro účely segmentace obrazu se přímo nabízí. Model je schopen velmi jednoduše oddělit skutečnou informaci o barvě objektu od vlivu osvětlení. Nevýhodou tohoto modelu je, že není až tolik standardizovaný a je potřeba ho vypočítat z častěji používaných modelů, jako je třeba RGB model.



Obrázek 2-2: *Ukázka míchání barev pomocí modelu HSV [10]*

Transformace z RGB do HSV modelu je založena na přepočtu jednotlivých složek modelů. Nejdříve je potřeba vybrat minimální a maximální hodnotu ze složek r, g, b z modelu RGB. Maximální hodnotu budu označovat jako $\max(r,g,b)$ a minimální hodnotu jako $\min(r,g,b)$.

Složky h, s, v z modelu HSV se vypočítají podle vztahů:

$$h = \begin{cases} 60 \times \dfrac{g-b}{\max(r,g,b) - \min(r,g,b)} + 0°; \max(r,g,b) = r \wedge g \geq b \\ 60 \times \dfrac{g-b}{\max(r,g,b) - \min(r,g,b)} + 360°; \max(r,g,b) = r \wedge g \prec b \\ 60 \times \dfrac{b-r}{\max(r,g,b) - \min(r,g,b)} + 120°; \max(r,g,b) = g \\ 60 \times \dfrac{r-g}{\max(r,g,b) - \min(r,g,b)} + 240°; \max(r,g,b) = b \\ 0; \max(r,g,b) = \min(r,g,b) \end{cases} \quad 2.1.3$$

$$s = \begin{cases} 0; \max(r,g,b) = 0 \\ 1 - \dfrac{\min(r,g,b)}{\max(r,g,b)}; \textit{jinak} \end{cases} \quad 2.1.4$$

$$v = \max(r,g,b) \quad 2.1.5$$

Jednotlivé složky jsou v rozmezí, jak je uvedeno výše v popisu modelu. Složky s a v jsou udány v rozmezí 0 až 1, kdy $1 = 100$ %. Zkusíme-li převést odstíny šedi v RGB modelu, které jsou vyjádřeny stejnými hodnotami jednotlivých složek, na hodnoty HSV, pak zjistíme, že transformace se chová podle předpokladu:

- Složka $h = 0$, protože minimum a maximum jsou si rovny.
- Složka $s = 0$, protože podíl minima a maxima je roven jedné.
- Složka $v = odstin\_sedi$, protože maximum je rovno minimu a tudíž představuje daný stupeň.

Transformace z HSV do RGB modelu je založena na přepočtu jednotlivých složek modelů. Nejdříve je potřeba vypočítat její průběžné výsledky:

$$h' = \left\lfloor \frac{h}{60} \right\rfloor \bmod 6 \quad 2.1.6$$

$$f = \frac{h}{60} - h' \quad 2.1.7$$

$$p = v \times (1 - s) \quad 2.1.8$$

$$q = v \times (1 - f \times s) \quad 2.1.9$$

$$t = v \times (1 - (1 - f) \times s) \quad 2.1.10$$

Jednotlivé složky r, g, b z modelu RGB se určí následovně:

$$(r,g,b)=\begin{cases}(v,t,p);h'=0\\(q,v,p);h'=1\\(p,v,t);h'=2\\(p,q,v);h'=3\\(t,p,v);h'=4\\(v,p,q);h'=5\end{cases} \qquad 2.1.11$$

Složky r, g, b jsou v rozmezí hodnot 0 až 1. Transformace zachovává principy obou modelů. Hodnota h' určuje základní barvy a hodnota f je míra přechodu mezi sousedními barvami. Vzorce obou transformací jsou převzaty z [8].

2.2 Vlastnosti obrazu

Už víme, jak je možné reprezentovat obsah obrazu v digitální podobě. Je však třeba si uvědomit, že dva obrazy stejné scény nemusí mít nutně identický barevný prostor. Na základě tohoto poznatku můžeme rozdělit obrázky na:

- Syntetické obrázky
  - vznikají na počítači,
  - víme přesný tvar i barvu,
  - barevné prostory dvou obrazů stejné scény jsou identické.
- Reálné obrázky
  - vznikají snímáním fotoaparáty, kamerami, skenováním apod.,
  - dochází k nepřesnostem při převodu analogového signálu (světlo ze snímané scény) na digitální obraz (diskrétní obrazový prostor),
  - barevné prostory dvou obrazů stejné scény nejsou zpravidla identické.

Z tohoto rozdělení vyplývá, že syntetické obrázky jsou daleko lepší pro strojové zpracování. Bohužel však právě díky své povaze nejsou předmětem zkoumání v oboru zpracování obrazu. Jde-li scéna vytvořit na počítači, může se hned zanést informace o objektech (např. vektorovou grafikou) a využitých algoritmech pro zanesení kazů (např. typ a parametry naneseného šumu, úpravy jasu, rozostření atd.), aby výsledný syntetický obraz byl co nejvíce realistický. Proto se zpracování obrazu zaměřuje na reálné obrazy, u kterých není možné zajistit implicitní informace o objektech, a tudíž vzniká potřeba je dodat jinou cestou. Nejčastější nedokonalosti snímání u reálných obrazů jsou:

- Přesvětlený obraz
  - vznikne dlouhou dobou expozice,
  - na snímač dopadne hodně světla,
  - jsou dobře viditelné objekty ve stínu,

    - jsou špatně viditelné osvětlené objekty.
- Tmavý obraz
    - vznikne příliš krátkou dobou expozice,
    - na snímač dopadne málo světla,
    - jsou dobře viditelné osvětlené objekty,
    - jsou špatně viditelné objekty ve stínu.
- Zašuměný obraz
    - porucha snímání (vzorkování a kvantování obrazu),
    - odklon od skutečné hodnoty pixelu,
    - různé typy
        - pepř a sůl – náhodný pixel změněn na minimální nebo maximální hodnotu,
        - Gaussův šum – náhodný pixel je změněn v návaznosti na jeho originální hodnotu,
        - náhodný (bílý šum) – analogie k bílému světlu, obsahuje různé frekvence šumu,
        - aditivní bílý šum – bílý šum je přidán k originálnímu obrazu.
- Špatně zaostřený obraz
    - dochází k rozmazání hranic objektu.

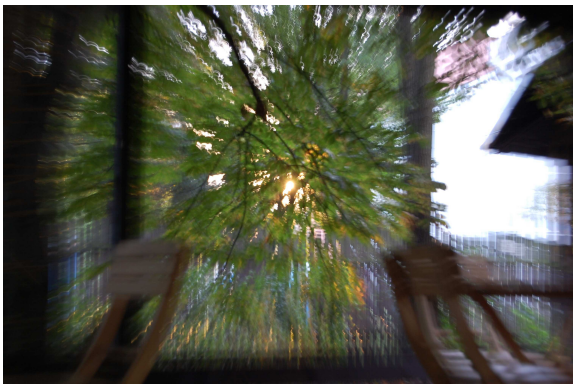
Na ukázku je přiloženo několik obrázků demonstrujících jednotlivé nedokonalosti.

Obrázek 2-3: *Přesvětlený obraz – nejsou vidět venkovní sloupy, ale je možné rozpoznat interiér budovy*

Obrázek 2-4: *Tmavý obraz - není vidět interiér budovy, ale jsou vidět detaily sloupu*

Lidské oko zvládne vidět ve scéně z Obrázek 2-3 a Obrázek 2-4 interiér i exteriér.

Obrázek 2-5: *Zašuměný obraz - Gaussův šum [11]*



Obrázek 2-6: *Rozmazaný obraz – vznik rotačním pohybem digitálního fotoaparátu v okamžiku snímání. Jev značně stěžuje lokalizaci a identifikaci objektů ve scéně.*

2.3 Operace nad obrazem

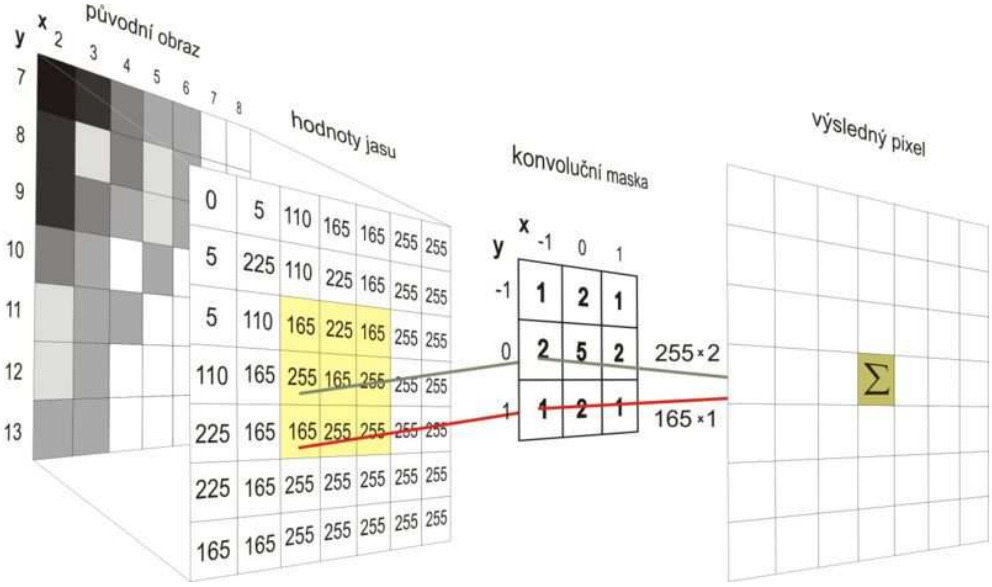
Tato podkapitola zachycuje možnosti explicitních úprav vlastností obrazu. Operace nad obrazem lze rozdělit do tří skupin podle velikosti okolí zpracovávaného pixelu:

- **Bodové**
  - průchod po bodech,
  - např. úprava jasu, převod barevného prostoru, prahování.
- **Lokální**
  - informace z malého okolí pixelu,
  - např. konvoluce (jádra 3x3, 5x5, 7x7 pixelů).
- **Globální**
  - informace z celého obrázku,
  - např. ekvalizace histogramu.

Dvojrozměrná konvoluce je důležitou operací při lineárním zpracování obrazu. Základem je tzv. konvoluční jádro neboli konvoluční maska ($h(x,y)$). Konvoluční jádro o velikosti $n \times n$ se aplikuje na diskrétní obrazovou funkci ($I(x,y)$) následovně:

$$I(x,y) * h(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot h(i,j), \text{ kde } k = \left\lfloor \frac{n}{2} \right\rfloor \qquad 2.3.1$$

Konvoluce umožňuje definovat okolí pixelu jedním číslem. Lze tak jednotným způsobem implementovat různé operace jen záměnou masky. Tyto operace mohou mít vysokou efektivitu (průměrování, Gaussův filtr, Gaborův filtr, hranové detektory apod.).



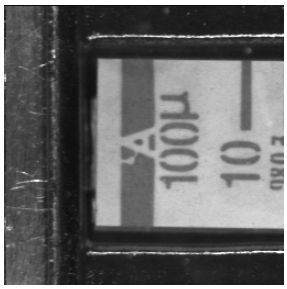
Obrázek 2-7: *Schéma algoritmu pro výpočet konvoluce nad pixelem obrázku [12]*

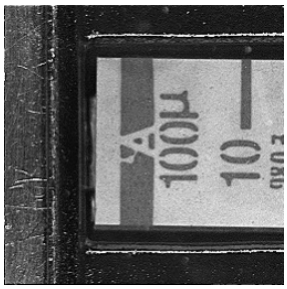
$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Obrázek 2-8: *Příklad použití Gaussova filtru – konvoluční jádro (vlevo), originální obrázek (uprostřed), výstup konvoluce (vpravo) [11]*

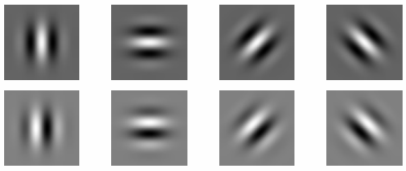
0	-1	0
-1	5	-1
0	-1	0





Obrázek 2-9: *Příklad zvýraznění hran pomocí konvoluce - konvoluční jádro (vlevo), originální obrázek (uprostřed), výstup konvoluce (vpravo) [13]*

Gaborovy filtry patří ke konvolučním filtrům pro klasifikaci a popis různých textur. Umožňují každému pixelu přiřadit charakteristiku okolí a vyjádřit ji jedním číslem. Jsou schopny zachytit frekvenci okolí pixelu v závislosti na jednotlivých směrech 0° až 360°, ovšem jedno konvoluční jádro představuje pouze jeden směr.



Obrázek 2-10: *Příklady jader Gabodrových filtrů [6]*

Pro tvorbu různých jader je možné použít jedno mateřské jádro a z něj odvodit pomocí rotace a změny velikosti jádra jiná. K vytvoření konvolučních jader Gaborových filtrů slouží Gaborova funkce, která je dána vztahem:

$$g_{\lambda,\theta,\psi,\sigma,\gamma}(x,y) = e^{-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}} \cdot \cos\left(2\pi\frac{x'}{\lambda}+\psi\right) \qquad 2.3.2$$

$$x' = x\cdot\cos\theta + y\cdot\sin\theta \qquad 2.3.3$$

$$y' = y\cdot\cos\theta - x\cdot\sin\theta \qquad 2.3.4$$

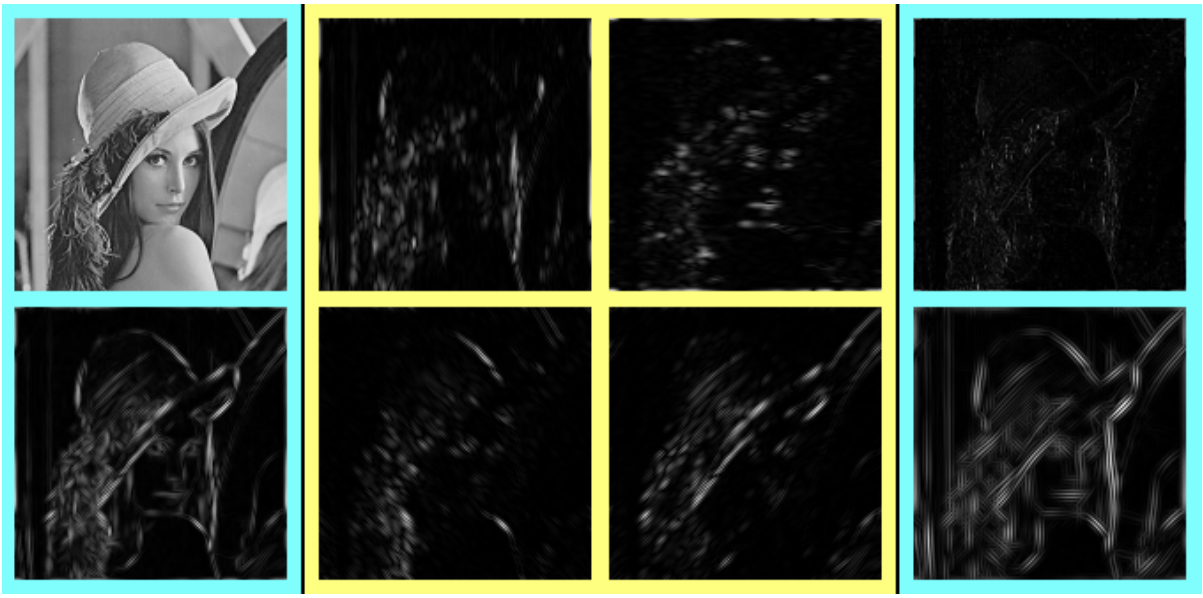
Parametry pro Gaborovu funkci jsou:

- vlnová délka (λ)

- zadává se v procentech,
- hodnoty musí patřit mezi reálná čísla větší než 2.

- natočení (θ)
  - udává natočení Gaborovy funkce,
  - hodnoty jsou v rozmezí 0 až 360°,
  - svislá orientace Gaborových vlnek je při 0°.
- fázový posun (ψ)
  - udává posun v ose natočení,
  - hodnoty jsou v rozmezí -180 až 180°.
- poměr stran (γ)
  - určuje tvar funkce (kružnice nebo elipsa),
  - hodnoty v rozmezí 0 až 1,
  - pro kružnici je hodnota rovna 1.

Vztah je uveden v [15].

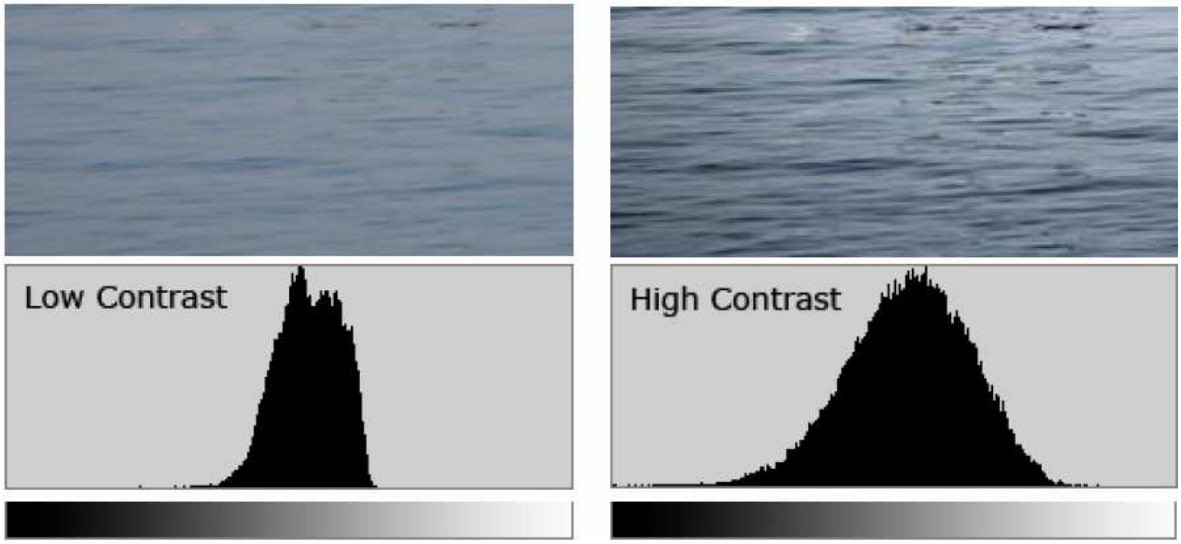


Obrázek 2-11: *Vliv Gaborových filtrů na vstupní obrázek (vlevo nahoře). Uprostřed jsou uvedeny odezvy na jednotlivé použití Gaborových filtrů pro natočení 0°, 90°, 45°,135° při nulovém posuvu a vlnové délce 7. Superpozicí těchto 4 filtrů získáme odezvu zachycenou v levém dolním obrázku. Další dva obrázky (v pravém sloupci) jsou superpozicí pro filtry 0°, 90°, 45°, 135°, kdy použité sady filtrů se liší vlnovou délkou. Pravý horní obrázek je odezvou na filtry s vlnovou délkou 2 a pravý dolní obrázek s vlnovou délkou 12. K filtraci byl použit software postupný z [15].*

Jak je patrné z Obrázek 2-11, při filtraci dochází k maximální odezvě (znázorněna bílou barvou) především na hranách, které odpovídají zvolenému profilu. Profil je reprezentován konvolučním jádrem. Při malé vlnové délce dojde k zachycení sebemenších nerovností. V tomto

případě může být maximální odezva způsobena i šumem. Při větších vlnových délkách dojde k eliminaci malých nerovností, ale vzroste vliv hrany na okolí a tím pádem i k detekci širší hrany.

Histogram je statistická reprezentace obrazu. Často se znázorňuje jako sloupcový graf, kde osa x označuje jednotlivé hodnoty jasu $x \in \langle 0,255 \rangle$ a osa y označuje počet pixelů, které mají společnou hodnotu jasu. Osa x nemusí označovat pouze hodnotu jasu, ale třeba i zastoupení jednotlivých složek v barevném modelu.



Obrázek 2-12: *Příklad histogramů u dvou obrázků s různým kontrastem*

3 Neuronové sítě

Umělá neuronová síť je výpočetní model, který našel své uplatnění v umělé inteligenci. Vychází z biologické nervové soustavy. Ta má za úkol zachytit a zpracovat podněty působící na organismus a zajistit odpovídající reakci na ně.

3.1 Biologický neuron

Biologický neuron je základní stavební prvek nervové soustavy živočichů. Je tvořen:

- tělem neuronu,
- výběžky neuronu
    - dendrity – šíří informaci od axionů předchozího neuronu k tělu neuronu,
    - axiony – šíří informaci od těla k dendritům následníka.

Spojení mezi axiony a dendrity se nazývá synapse. Šíření vzruchu v nervové soustavě se děje na základě akčního potenciálu, který je v neurální membráně a práh je dán propustností synapse.

3.2 Umělý neuron

Umělý neuron je výpočetní model biologického neuronu. Je tvořen:

- tělem neuronu ($f_{act}(u)$) – aktivační funkce neuronu,
- spoji mezi neurony
    - vstupy ($\vec{I}$) – spojeny s výstupy předchozích neuronů, každý vstup má svojí váhu (w),
    - výstup (O) – nese výsledek aktivační funkce dál.

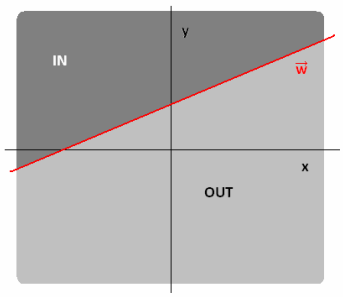
Synapse v tomto modelu nemá stejný význam jako v biologickém modelu. Je reprezentována pouze vahou daného vstupu neuronu. Každý vstup představuje hodnotu určité vlastnosti, kterou má přidělená váha za úkol potlačit nebo zvýraznit.

Lze říci, že stejně jako konvoluce ve zpracování obrazu, představuje neuron v umělé inteligenci jednotný algoritmus pro celou škálu úkolů. Modifikaci vstupu pak určuje typ použitých matematických funkcí a zapojení neuronů do sítě stejně, jako u konvoluce použití různých typů jader.

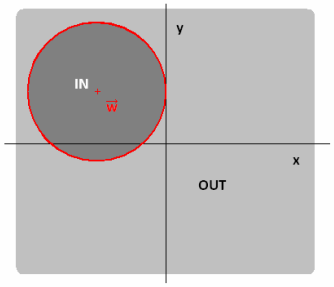
Neuron na základě vstupů a vah vypočítá tzv. bázovou funkci ($f_{baze}(\vec{I},\vec{w})$), na jejímž základě určí aktivační funkce výstup neuronu. **Matematický model neuronu** (N) je dán relací $N:\Re^{2n}\rightarrow\Re$, která je určena předpisem:

$$N: O = f_{act}\left(f_{baze}\left(\vec{I},\vec{w}\right)\right), \text{ kde } \vec{I} = (1, I_1, I_2, ..., I_n) \qquad 3.2.1$$

Aktivační a bázová funkce mohou mít různé předpisy podle zaměření úlohy, na kterou je neuron aplikován. Aktivační funkce je dána relací $f_{act}:\Re\to\langle -1,1\rangle$ nebo $f_{act}:\Re\to\{-1,1\}$, případně s hraničními hodnotami intervalu 0 a 1. Bázová funkce je dána relací $f_{baze}:\Re^{2n}\to\Re$ a rozhoduje o tom, jakým způsobem neuron rozdělí prostor na dvě množiny $IN=\left\{\vec{I}\,\middle|\,N\left(\vec{I},\vec{w}\right)=-1, resp. N\left(\vec{I},\vec{w}\right)=0\right\}$, $OUT=\left\{\vec{I}\,\middle|\,N\left(\vec{I},\vec{w}\right)=1\right\}$. Rozlišujeme lineární a radiální bázovou funkci. Aktivační funkce určuje, jak ostrá bude hranice mezi množinami (skoková, po částech lineární, sigmoidní, …). V případě radiální bázové funkce určuje vzdálenost, pro kterou je ještě prvek v prostoru zařazen do množiny IN.

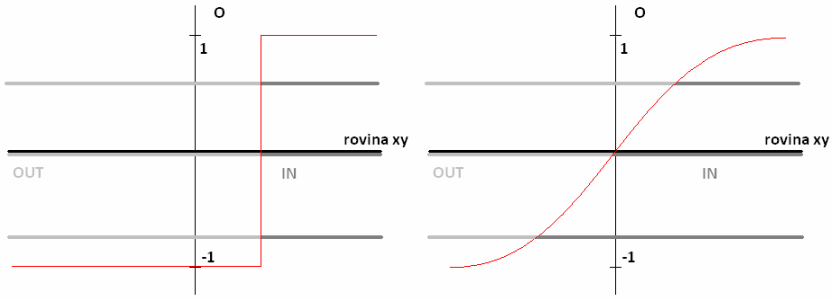


Obrázek 3-1: *Rozdělení roviny - neuron s lineární bázovou funkcí* $f_{baze}\left(\vec{I},\vec{w}\right)=\sum_{i=0}^{n} I_i\cdot w_i$

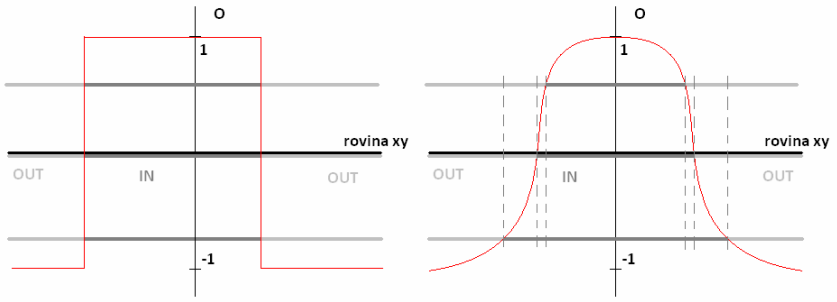


Obrázek 3-2: *Rozdělení roviny - neuron s radiální bázovou funkcí* $f_{baze}\left(\vec{I},\vec{w}\right)=\left\|\vec{I}-\vec{w}\right\|$

Z Obrázek 3-1 a Obrázek 3-2 je patrné, že pro obdobné rozdělení roviny, jaké zvládne jeden neuron s radiální bázovou funkcí, je potřeba alespoň tři neurony s lineární bázovou funkcí.



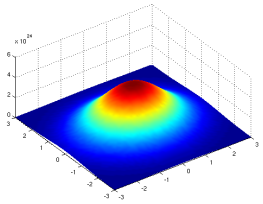
Obrázek 3-3: *Výstup neuronu a rozdělení roviny pomocí neuronu s lineární bázovou funkcí - aktivační funkce skoková (vlevo) nebo sigmoidní (vpravo)*



Obrázek 3-4: *Výstup neuronu a rozdělení roviny pomocí neuronu s radiální bázovou funkcí - aktivační funkce skoková (vlevo) nebo sigmoidní (vpravo)*

Z Obrázek 3-3 a Obrázek 3-4 lze zjistit, že sigmoidní aktivační funkce dává prostor pro prahování a odlazení menších nuancí, které se mohou objevit až v širším kontextu. Radiální bázová funkce společně se sigmoidní aktivační funkcí umožňuje aproximaci Gaussovského rozdělení.

V poslední době se začalo experimentovat i s jinými funkcemi a velmi zajímavé výsledky slibuje i přímé použití Gaussovy funkce pro 2D prostor. Využívá se různých rozptylů v daných osách. Pak jeden neuron s touto bázovou/aktivační funkcí dokáže nahradit tři neurony s klasickou radiální bázovou funkcí nebo čtyři a více neuronů s lineární bázovou funkcí.



Obrázek 3-5: *Znázornění průběhu Gaussovy funkce pro 2D prostor [21]*

3.3 Neuronové sítě

Spojením dvou a více neuronů můžeme kombinovat vlastnosti jednotlivých neuronů a umožnit tak rozdělení prostoru na dva a více podprostorů. Tato vlastnost má velký význam ve zpracování obrazu. Neuronové sítě jsou mocný nástroj a lze je použít pro následující oblasti:

- klasifikace,
- shlukování,
- asociace,
- funkční aproximace,
- predikce,
- optimalizace,
- a další.

Existují algoritmy pro úpravu topologie a váhových vektorů (tzv. učící algoritmy) neuronových sítí. Díky těmto algoritmům mají neuronové sítě schopnost postupné aproximace funkce. Podle její topologie je možné říci, které funkce dokáže aproximovat:

- perceptron (1 neuron) - aproximace logických funkcí (AND, OR, NOT),
- 1 skrytá vrstva - aproximace libovolné booleovské funkce (XOR, …) a spojité ohraničené funkce,
- 2 skryté vrstvy - aproximace libovolné funkce.

Jak je uvedeno v [7]. Následnou úpravou váhových vektorů vybíráme funkci z množiny funkcí, které lze danou topologií aproximovat. Z tohoto důvodu jsou velice důležité učící algoritmy.

Učení sítí je vhodné pro aproximaci dané funkce. Učení probíhá na základě tzv. trénovací množiny dat a testovací množiny dat, aby se předešlo k tzv. přeučení sítě. Při přeučení sítě dochází k tomu, že neuronová síť dobře rozpoznává data trénovací množiny, ale při použití na testovacích datech (neočekávaných datech) selhává a ztrácí schopnost generalizace. Podle struktury trénovacích dat rozlišujeme učení:

- S učitelem
    - Máme trénovací množinu dat, u kterých víme, kam je má neuronová síť zařadit.
    - Dochází k výpočtu odchylky výstupu od očekávaného výsledku a následnému přepočtu vah.
    - Pro testovací množinu dat existuje hodnotící funkce, která nám řekne, jak daná síť uspěla.
    - Je vhodné pro klasifikaci, asociaci, apod.
- Bez učitele
    - Trénovací množina dat neobsahuje správné zařazení.
    - Existuje však hodnotící funkce, jejíž výsledek říká, jak moc je daný výstup správný.
    - Je vhodné pro shlukování, optimalizaci apod.

Také tyto přístupy k učení mohou jednotlivé neuronové sítě různě spojovat. Je třeba dávat pozor na použité vstupní vektory. Jeden nepatřičný vstup je schopen narušit celý proces učení. Pro volbu trénovacích a testovacích dat neexistuje jednotný algoritmus výběru. Je třeba vycházet z problému, na který je neuronová síť aplikovaná. Existuje však způsob výběru správně natrénované sítě, který minimalizuje možnost výběru přeučené sítě a zaručuje jistou míru generalizace. Jedná se o cross-validaci.

Cross-validace se zaměřuje na způsob volby trénovací a testovací množiny dat. Mějme velkou množinu vstupních vektorů. Rozdělme ji na S podmnožin. Vybereme $S-1$ podmnožin a sloučíme ji do trénovací množiny. Zbylou podmnožinu označíme za testovací. Po naučení sítě a testování zaznamenáme úspěšnost sítě. Opakujeme postup tak dlouho, dokud každá podmnožina nebyla

testovací množinou (tzn. S krát). Na závěr označíme síť průměrnou úspěšností. Tento postup je zdlouhavý.

3.4 Použití neuronových sítí

V této souvislosti bych se ráda zmínila o několika typech neuronových sítí a rozvedla blíže jejich zaměření i způsob učení.

Dopředná neuronová síť se skládá z několika vrstev. Neurony jedné vrstvy mají na vstupu výstupy všech neuronů vrstvy předcházející. Spoje jsou jednosměrné a bez zpětné vazby (odtud název sítě - dobředná). Neurony v jedné vrstvě se vzájemně neovlivňují. Obecně platí, že dopředná neuronová síť implementuje funkce dané relací:

$$DNS : \Re^m \rightarrow \Re^n , \qquad 3.4.1$$

kde m značí počet vstupů (tedy počet neuronů ve vstupní vrstvě), n značí počet výstupů (tedy počet neuronů ve výstupní vrstvě).

Vícevrstevná dopředná neuronová síť obsahuje minimálně dvě vrstvy:

- vstupní,
- výstupní.

Kromě nich může obsahovat také vrstvy skryté. Přesný návrh topologie sítě je dán specifičností úlohy a její návrh vychází ze simulačních výpočtů a experimentů. Jedná se především o upřesnění:

- počtu vrstev,
- počtu neuronů v jednotlivých vrstvách.

Jak jsem se zmínila v předchozí podkapitole, k aproximaci libovolné funkce stačí 2 skryté vrstvy (dle zdroje [7]).

Nejčastěji se pro tento typ sítě volí jako učící algoritmus **Back Propagation**. Pro použití tohoto algoritmu je potřeba trénovací množina, která má k dispozici vstupní vektor sítě $\vec{x}$ a chtěný výstupní vektor sítě $\vec{y}$. Přepočet jednotlivých vah neuronů sítě vychází ze stanovení odchylky výstupu sítě od vektoru $\vec{y}$. Jedná se o chybu sítě. Algoritmus následně prochází jednotlivé neurony od poslední vrstvy (odtud název algoritmu - zpětné šíření chyby) a stanoví se odchylka δ_i^l pro každý neuron i ve vrstvě l. Vlastní přepočet vah $\vec{w}_i^l$ jednotlivých neuronů se opět děje průchodem sítě tentokrát od první vrstvy. Váha neuronů se nastaví podle vzorce:

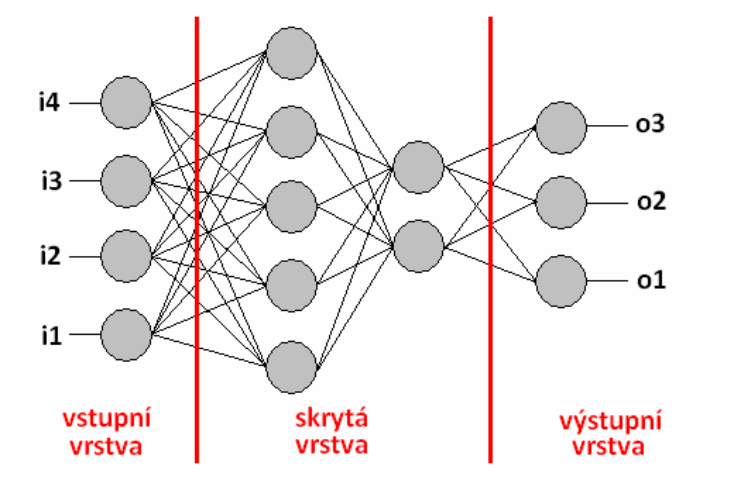
$$w_{i\,j}^l(k+1) = w_{i\,j}^l(k) + \mu \cdot \delta_i^l(k) \cdot x_j^l(k) , \qquad 3.4.2$$

kde j představuje index ve vektoru vah, k je iterace algoritmu, μ je učící konstanta (určuje míru konvergence) a $\vec{x}^l$ je vektor vstupů pro danou vrstvu l. Učení se provádí v:

- iteracích – jedna iterace odpovídá adaptaci na jeden trénovací vzorek,

- epochách – jeden průchod trénovací množinou.

Učení se provádí tak dlouho, dokud algoritmus nepřekročí stanovený počet epoch nebo síť nedospěje ke stanovené chybě sítě.



Obrázek 3-6: *Schéma dopředné neuronové sítě se dvěmi skrytými vrstvami*

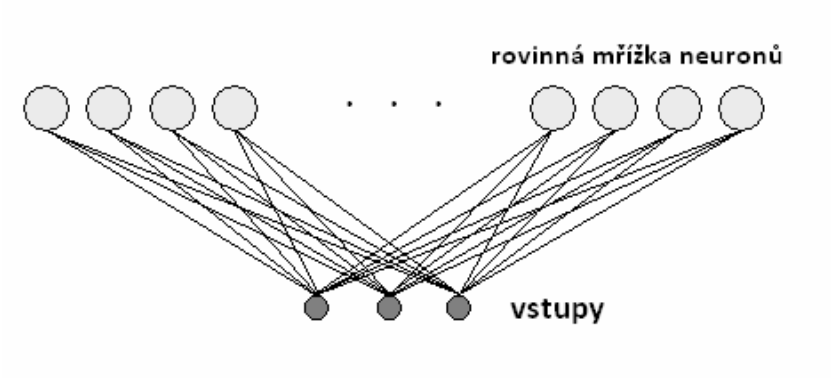
Kohonenova mapa je oproti dopředné neuronové síti jednovrstvá. Topologie sítě je tedy založena na jedné vrstvě, která je zároveň vstupní i výstupní. V jednom časovém okamžiku může být aktivní pouze jeden výstupní neuron. Neurony používají radiální bázovou funkci. To znamená, že váha neuronu představuje střed shluku a každý neuron ve vrstvě představuje jeden shluk. Při učení sítě se upravují váhy sousedních neuronů po získání nejlepšího neuronu pro daný vstupní vektor.

Pro neoptimalizovanou Kohonenovu mapu jsou počáteční váhy neuronů náhodně voleny. Pomocí učícího algoritmu se váhy jednotlivých neuronů rozprostřou do prostoru tak, aby pokrývaly všechna trénovací data. Kohonenovy mapy využívají soutěžní strategie učení.

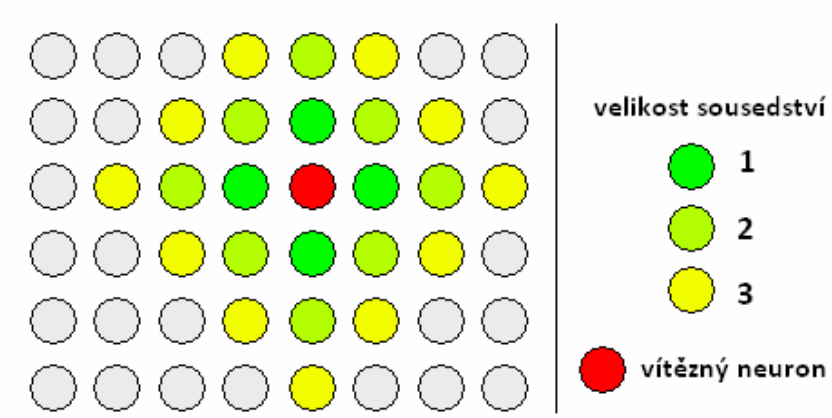
Soutěžní strategie učení je založena na učení bez učitele. Bázová funkce neuronů zastupuje hodnotící funkci. Algoritmus učení určí tzv. vítězný neuron. Jedná se o neuron, jehož váhový vektor má nejblíže vstupnímu vektoru $\vec{x}_i$. Neurony v okolí vítězného neuronu upraví svoji váhu $\vec{w}_j$ podle vztahu:

$$\vec{w}_j(k+1) = \vec{w}_j(k) + \mu(k) \cdot D(d) \cdot (\vec{x}_i(k) - \vec{w}_j(k)), \qquad 3.4.3$$

kde jednotlivé složky mají podobný význam jako u Back Propagation a funkce $D(d)$ určuje míru vlivu učící funkce na sousední neurony podle vzdálenosti d od vítězného neuronu. Vzdálenost d je blíže popsána na Obrázek 3-8.

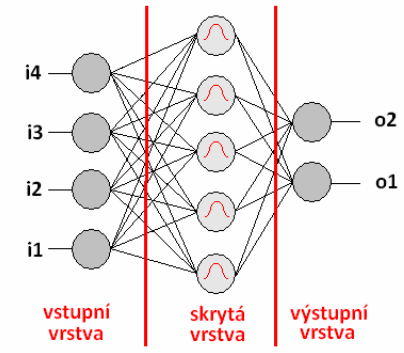


Obrázek 3-7: *Schéma Kohonenovy mapy (SOM)*



Obrázek 3-8: *Detail rovinné mřížky neuronů pro Kohonenovu mapu spolu se znázorněním, kam až sahá přepočet vah při zvolení vítězného neuronu v závislosti na velikosti sousedství*

Radial Basis Functions (dále již RBF) je druh dopředné neuronové sítě. Topologie sítě je založena na jedné skryté vrstvě obsahující neurony s radiální bázovou funkcí. Neurony v ostatních vrstvách používají lineární bázovou funkci. Tato síť se často používá pro klasifikaci vstupu. Počet výstupů sítě je založen na počtu tříd, které chceme klasifikovat. Skrytá vrstva je schopna aproximovat různé tvary podprostorů.

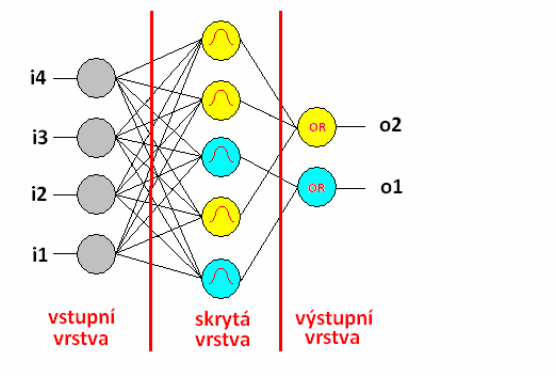


Obrázek 3-9: *Schéma RBF sítě*

Učení uvedené sítě probíhá na základě Back Propagation (zdlouhavé). Využívá učení s učitelem. Možné je použít algoritmus pro vytvoření topologie sítě **Restricted Coulomb Energy** (dále RCE) na základě trénovací množiny. RCE je druh sítě typu RBF. Začínáme tvořit topologii sítě pouze na základě vstupů. Při sestavování topologie musíme alespoň jednou projít celou trénovací množinu. Pro každý neuron ze skryté vrstvy spočítáme jeho vzdálenost od vstupu. Pokud vstup:

- náleží k neuronu ze stejné třídy – přejdeme k dalšímu vstupu,
- náleží k neuronu z jiné třídy – zmenšíme poloměr neuronu a přidáme nový skrytý neuron s váhou stejnou jako vstup,
- nenáleží žádnému neuronu – přidáme nový neuron s váhovým vektorem stejným, jako je vektor vstupu do skryté vrstvy,
- nový neuron skryté vrstvy napojíme na výstupní neuron dané třídy,
- neexistuje-li výstupní neuron k dané třídě vstupu, pak ho přidáme.

Trénovací množinu procházíme stále dokola tak dlouho, dokud už není potřeba měnit topologii sítě.



Obrázek 3-10: *Schéma RCE sítě*

4 Segmentace obrazu

Segmentace obrazu je jedna z metod zpracování obrazu, která dokáže označit v obraze plochy, které korespondují s objekty ve scéně na základě podobných vlastností jednotlivých pixelů. Jedná se nejčastěji o materiál a barvu, které objekt reprezentují. Stejně jako segmentace využívá konvoluci a lineární filtraci pro zpracování obrazu. Vyšší metody zpracování obrazu mohou využít výstup ze segmentace obrazu pro další zpracování (např. charakteristika tvaru, pojmenování objektů, zjištění rozměrů objektů atd.).

Kapitola se bude zabývat definicí základních pojmů pro segmentaci obrazu, vytyčení cíle segmentace a představení jednotlivých algoritmů, které mohou být implementovány v softwarové části diplomové práce.

4.1 Definice segmentace obrazu

Segmentace obrazu $f(x,y)$ je jeho dělení na podobrazy $R_1, R_2, ..., R_n$ tak, že podobrazy splňují následující kritéria:

- sloučením všech podobrazů vznikne původní obrázek
  - $\bigcup_{i=1}^{n} R_i = f(x,y),$
- dva různé podobrazy jsou vzájemně disjunktní
  - $R_j \cap R_i = \emptyset; i \neq j,$
- každý podobraz splňuje tvrzení (např. všechny pixely v podobraze R_i mají *„vlastnost“*).

Definice je uvedena v [1]. Pokud podobrazy zcela korespondují s objekty ve scéně, pak hovoříme o úplné segmentaci, jinak hovoříme o částečné segmentaci. Výsledkem segmentace je označení každého pixelu indexem segmentu (např. průměrné barvy segmentu, číslem, třídou, …), ke kterému patří.

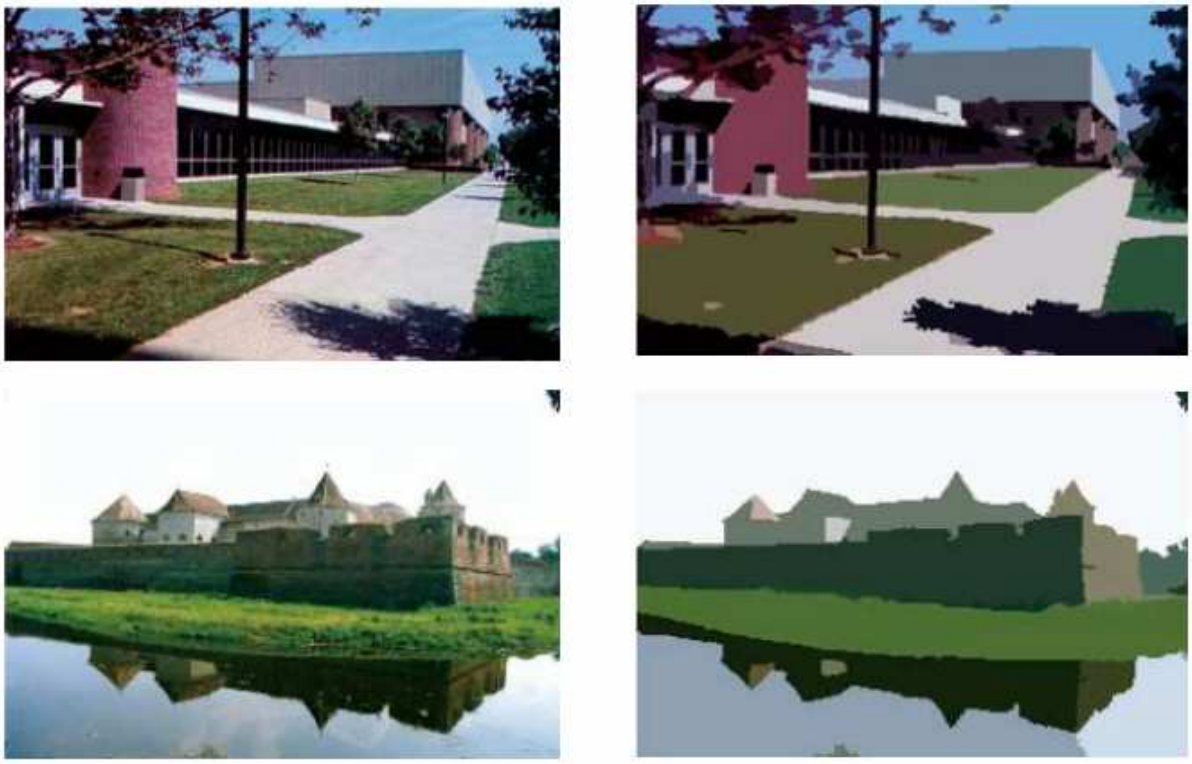
Poslední pravidlo z definice segmentace obrazu dává volnost v definování objektů. Lze tedy segmentovat obraz na základě různých předpokladů, např:

- objekt je tvořen jednou barvou,
- objekt má stejnou texturu na celém povrchu,
- objekt je kruhové těleso,
- a další.

Právě díky rozdílné definici objektů patří segmentace obrazu mezi náročné operace zpracování obrazu. Neexistuje jednotný algoritmus, který by se dal implementovat ve všech aplikacích. Využívá se různých přístupů k segmentaci obrazu:

- na základě hranových detektorů,
- na základě regionů v obraze,
- statistické metody,
- hybridní metody,
- znalostní segmentace.

Mezi další důvody, proč segmentační algoritmy řadíme k náročným, je různá kvalita snímání obrazů. Někdy i lidské oko velmi těžce rozpozná jednotlivé objekty.



Obrázek 4-1: *Příklad segmentace obrazu pomocí algoritmu Mean Shift [5]*

4.2 Definice objektu (segmentu)

Objekt může mít obtížnou definici. Jeho vlastnosti se často mění vzhledem ke směru zachycení 2D scény. Například obličej má jiné vlastnosti z profilu a jiné zepředu. Je však důležité, aby před začátkem segmentace bylo jasné, jak je objekt definován pro danou úlohu.

Pro segmentaci je důležité členění objektů. Objekty se vyznačují barvou (případně texturou) a tvarem. Objekty rozdělujeme na:

- Jednoduché (elementární)
    - nelze již dále dělit,

    - má atomické vlastnosti
        - jednotná barva (textura),
        - jednoduchý tvar (n-úhelník, kružnice, elipsa).
- Složené
    - obecná definice objektu,
    - skládá se z jednoduchých a složených objektů,
    - dá se popsat pomocí stromu
        - listy - jednoduché objekty,
        - uzly – jednotlivé podobjekty,
        - kořen - představuje vlastní složený objekt,
    - další vlastnosti mohou hrát roli
        - sousednost,
        - poloha,
        - a jiné.

Segmentace by především měla odhalit elementární objekty. Lze pak přiřadit těmto objektům třídu (židle, tužka, tlačítko, sklenice, písmena, …). Následně při hledání určitého objektu aplikovat jeho strom popisu na nalezené elementární objekty a rozhodnout, zda daný objekt je v obraze nebo ne.

Definice elementárního objektu se mění vzhledem k rozlehlosti scény. Pokud obraz obsahuje například pohled na psací stůl, pak jako elementární objekty bereme kalkulačku, tužky, pera, krabici od cartrige apod. Tvoří-li však větší část pohledu kalkulačka s krabicí od cartrige, pak jednotlivé elementární objekty jsou tlačítka, display, písmena, číslice.

Při segmentaci reálného obrázku budu klást důraz na barevné vlastnosti objektu. Tvar lze určit dodatečně na základě výstupu ze segmentace. Vlastnosti jednotlivých segmentů lze vyjádřit následovně:

- všechny pixely v podobraze mají podobný odstín barvy,
- všechny pixely v podobraze mají podobnou charakteristiku textury.

4.3 Segmentační algoritmy

Jedná se o algoritmy, které implementují vlastní segmentaci obrazu. Jak je popsáno výše, algoritmy mohou být vystavěné na různých principech. Nejčastěji však na shlukování nebo klasifikaci. Obojí se provádí nad tzv. prostorem příznaků.

Prostor příznaků představuje n-dimenzionální prostor určený hodnotami jednotlivých vlastností hledaného objektu. Vlastní segmentační algoritmus tento prostor rozdělí do podprostorů, které odpovídají jednotlivým objektům zájmu.

Klasifikace se děje nad známou množinou objektů, např. chceme-li na automatické lince propustit pouze objekt s určitými vlastnostmi, které známe (červené jablko, zcela okrájenou

bramboru, …). Pomocí jednotlivých algoritmů namodelujeme chtěné třídy (podprostory) a určujeme, zda obraz patří do této třídy nebo ne. Lze využít následující algoritmy:

- Gaussian Mixture Models,
- RBF.

Shlukování lze provádět nad neznámou množinou objektů. V daném prostoru příznaků spojí ty body, které mají k sobě nejblíže. Samozřejmě algoritmus jde optimalizovat pro jistý typ scén tak, aby shlukování proběhlo, co nejrychleji. Se vzrůstajícím počtem informací o segmentovaném obraze lze jejich činnost urychlit (vhodné zvolení středů shluků, …). Algoritmy, které se užívají při shlukování, jsou:

- k-means,
- Mean shift,
- shlukování na základě spojitosti,
- Path-based.

Jedná se především o iterační algoritmy, které časem zmenšují svoji chybu. Větší prostor bych tady chtěla věnovat algoritmu k-means.

Algoritmus k-means implementuje shlukování bodů v příznakovém prostoru. Je založen na rozdělení prostoru do k shluků. Každý shluk má svůj střed (bod v prostoru příznaků). Každý vstupní vektor z množiny dat je zařazen do jednoho shluku, jehož střed je mu nejblíže. Po přiřazení všech vstupních vektorů do jednotlivých shluků jsou středy shluků přepočítány tak, aby byly znovu středem shluku. Opět se projdou všechny vstupní vektory z množiny dat a jsou přiřazeny do jednotlivých shluků. Vše se opakuje tak dlouho, dokud se středy shluků nemění nebo je překročen počet iterací.

Algoritmus musí předem vědět, kolik shluků má vytvořit. Středy shluků se inicializují náhodně.

4.4 Kvalita segmentace

Důležitou částí segmentace je určení její kvality a celkové zhodnocení zvoleného algoritmu. Hodnotí se především:

- Vlastnosti segmentace
  - správnost zařazení.
- Vlastnosti algoritmu
  - robustnost,
  - schopnost generalizace,
  - apod.

Jak je uvedeno výše, od segmentace očekáváme, že jeden segment bude obsahovat všechny pixely, které představuje odpovídající objekt. Toto je ideální stav. Během segmentace však může dojít k problémům v klasifikaci/shlukování, proto je důležité zhodnotit podle vzoru segmentace výsledek

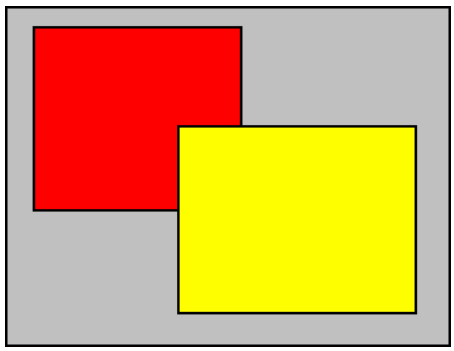
segmentace. Z pohledu klasifikace pixelu mohou nastat tyto případy:

- pixel je identifikován správně
    - true positive (objekt byl a měl být identifikován)
    - pixel odpovídá popsanému objektu
- pixel je identifikován na jiný objekt
    - false positive (objekt byl a neměl být identifikován)
    - pixel patří k jinému objektu

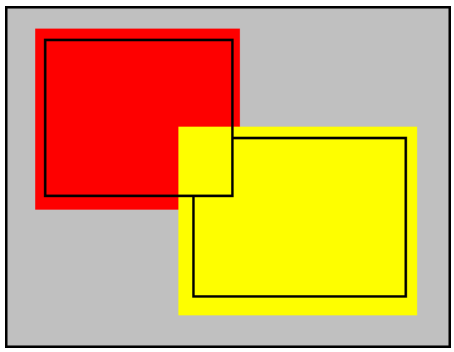
Z pohledu klasifikace objektu (segmentu) mohou nastat tyto případy:

- objekt je zcela obsažen v segmentu,
- segment je zcela obsažen v objektu,
- jednomu objektu přísluší více segmentů,
- více objektů přísluší jednomu segmentu.

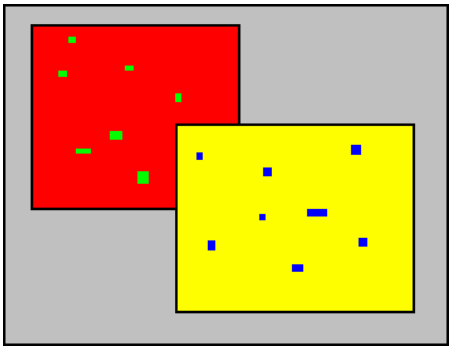
Následující série obrázků se bude věnovat možným chybám segmentace obrazu. Barevné plochy odpovídají nalezeným segmentům, černé tvary označují hranice reálných objektů.



Obrázek 4-2: *Ukázka správné segmentace – objekty jsou zcela obsaženy v segmentu a zároveň segment je zcela obsažen v objektu*



Obrázek 4-3: *Porucha segmentace – šedý segment je zcela obsažen v pozadí, pozadí zasahuje do tří segmentů; červený segment zasahuje do pozadí scény a také pokrývá pouze část objektu; žlutý segment zcela obsahuje objekt a navíc zasahuje do dvou dalších objektů.*



Obrázek 4-4: *Porucha segmentace - hranice objektu jsou definovány správně, vlivem šumu a textury je uvnitř objektu nalezeno více stejných segmentů, které nepřekračují hranici objektu, ale narušují výsledek segmentace*

Bylo by vhodné vytvořit způsob hodnocení, který dokáže zachytit všechny poruchy kvality segmentace číselně. Při hodnocení segmentace je rozhodující, na jaký objekt se daný pixel segmentu mapuje. Jedná se tedy o statistickou informaci, jaká část pixelů segmentu odpovídá jednotlivým objektům ze scény.

Tabulka 4-1: *Statistická informace o tom, kolik procent pixelů daného segmentu z* Obrázek 4-4 *odpovídá stejnému objektu*

Barva segmentu	Objekt	Odpovídající část segmentů
šedá	pozadí	100%
červená	objekt1	100%
žlutá	objekt2	100%
zelená	objekt1	100%
modrá	objekt2	100%

Je vidět, že tahle metrika úplně nestačí. Segmentace by se mohla zdát v pořádku. Je třeba mít i pohled opačný.

Tabulka 4-2: *Statistická informace o tom, kolik procent pixelů daného objektu z* Obrázek 4-4 *odpovídá stejnému segmentu*

Objekt	Barva segmentu	Odpovídající část objektu
objekt1	červená	95%
objekt1	zelená	5%
objekt2	žlutá	96%
objekt2	modrá	4%
pozadí	šedá	100%

Zde je již vidět jasné nedostatky v segmentaci.

5 Návrh segmentačního systému

Diplomová práce obsahuje i praktickou část, jejímž úkolem je vyvinout softwarové zázemí pro segmentaci obrazu pomocí neuronové sítě. Cílem je vytvořit přehlednou a intuitivní aplikaci, která umožní práci s jednotlivými segmentačními algoritmy.

Tato kapitola má čtenáři vysvětlit vyšší princip segmentačního systémů, představit vlastnosti jednotlivých algoritmů v širším kontextu a nahlížet na ně jako na tzv. černé skříňky. Rozebírá jen možnosti propojení algoritmů a očekávanou modifikaci vstupů na výstupy. Vlastní implementace bude popsána v další kapitole.

5.1 Zhodnocení problematiky a stanovení cílů

Základním cílem je vytvoření zázemí pro segmentaci obrazu a optimalizaci zvolených algoritmů. Implementovaná aplikace by měla nabízet:

- intuitivní ovládání,
- přehledné GUI,
- klást minimální nároky na znalost uživatele,
- dovolit měnit parametry pro zkušené uživatele.

Pro účely této diplomové práce jsem zvolila 2 segmentační metody. Jedna zastupuje neuronové sítě a druhá je zástupcem klasických metod. Segmentační algoritmy jsou jádrem celého systému a od jejich činnosti se odrážejí další části systému. Oba algoritmy by měly být v celém postupu zaměnitelné, aby bylo možno porovnat jejich výsledek za stejných podmínek. Celý segmentační systém je založen na čtyřech fázích segmentačního procesu:

- příprava trénovací množiny,
- příprava testovací množiny,
- adaptace zvoleného algoritmu na trénovací množinu,
- zhodnocení algoritmu na testovací množině.

Důležité je uvědomit si, že není předem znám objekt zájmu a jedná se o obecnou skladbu obrázků určených k segmentaci (krajina, portréty, zátiší atd.). Z tohoto důvodu by měl mít uživatel možnost zasáhnout do výběru trénovací množiny, aby byl schopen si algoritmus optimalizovat pro svoje data. Volnost uživatele ve volbě trénovacích dat pro učení algoritmu by měla začínat výběrem trénovacích obrázků, konvolučních filtrů pro předpřipravení vstupních obrazů a končit vytvořením vektoru příznaků (barevný model, textura).

Jak bylo uvedeno dříve, existují dvě metody učení neuronových sítí, které ovlivňují celkový proces segmentace. Při volbě učení bez učitele je závěrečné zhodnocení spíše statistické než pevně dané. Přesto je tato část velmi důležitá pro dodání objektivního charakteru celému segmentačnímu

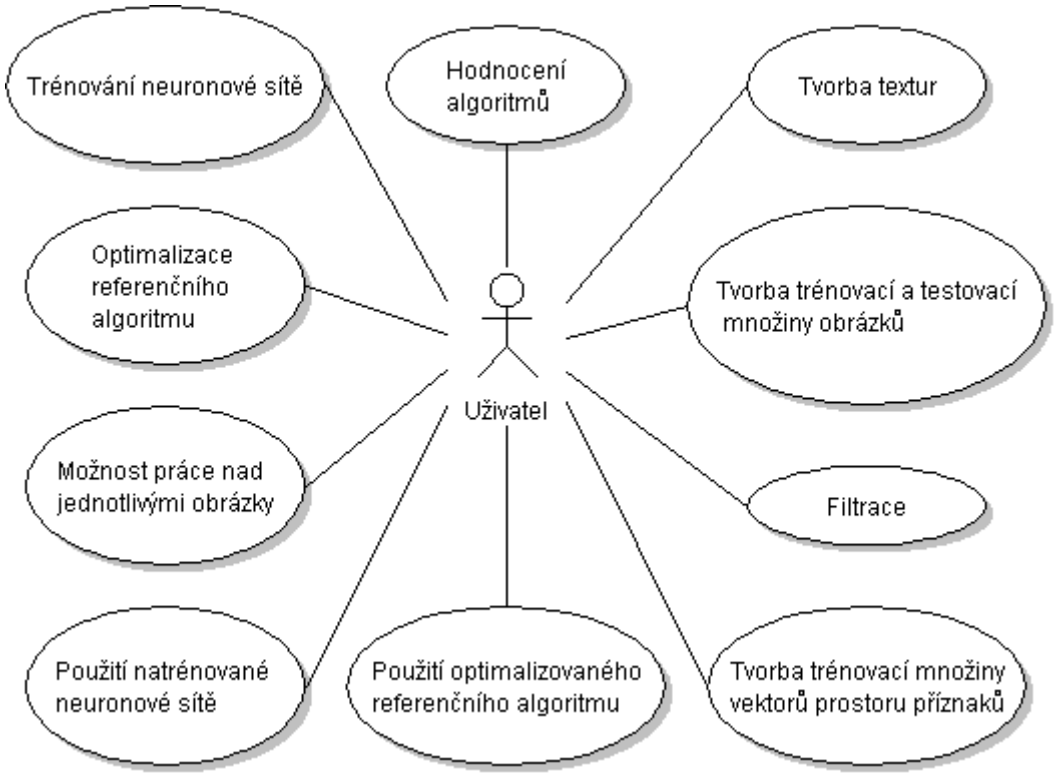
systému. Další významnou roli hraje rozhraní pro jednoduché nastavení učících algoritmů, aby bylo možné dané segmentační algoritmy zkoumat z různých úhlů a co nejlépe připravit na danou úlohu.

Od navržených segmentačních metod se odvíjí i množina úkonů, které je nutné implementovat pro pohodlí uživatele. Samozřejmě se jedná o experimentální činnost a neuronové sítě nabízí širokou škálu zajímavých algoritmů a nastavení. Vytvořit efektivní grafické rozhraní a přitom neomezovat pokročilého uživatele je velmi obtížný úkol. Proto bych segmentační systém chtěla rozdělit do dvou částí:

- grafické rozhraní pro segmentační systém a
- knihovna pro segmentaci pomocí zvolených algoritmů.

V tomto dokumentu se především zaměřím na první část a to z důvodu, že představuje nástavbu pro ovládání segmentačních algoritmů podle potřeb vzniklých při jejich základním porovnání a mapování možností dalšího rozvoje. Vlastní knihovna je pak pouze prostředek, který je nutný implementovat. Knihovna sama o sobě neříká nic o tom, jak se jednotlivé algoritmy propojí ve výsledný segmentační systém, protože musí zaručit její maximální rozšiřitelnost a znovu použitelnost algoritmů pro účely dalších systémů.

Pro vhodný návrh grafického rozhraní a základních rysů knihovny bylo použito některých diagramů patřících ke standardu UML, který slouží pro návrh aplikací v různých úrovních abstrakce. Tato fáze návrhu je důležitá především pro stanovení důležitých rysů.



Obrázek 5-1: *UML diagram případů použití pro navrhovaný segmentační systém*

Aplikace musí být dále schopna spolupracovat s jinými standardními aplikacemi pro lepší kontrolu a pro změnu dat v procesu učení přes uložené soubory obsahující:

- obrazové informace,
- nastavení segmentační metody,
- použité prostory příznaků a testovací data,
- automatické nastavení formulářů.

Pro tyto účely je potřeba využit volně dostupné formáty jako jsou bmp, jpeg, png pro obrazové formáty, XML pro nastavení aplikace a CVS pro statistická data.

Shrnutí jednotlivých cílů:

- Vybrat a implementovat segmentaci založenou na:
  - neuronových sítích,
  - referenčních metodách.
- Vytvořit dostatečně velkou množinu syntetických dat s informací o objektech ve scéně.
- Automaticky optimalizovat metody nad určitou sadou dat.
- Navrhnout formát pro uložení/načtení obrazu, optimalizovaných metod a množiny dat.
- Vybrat metody pro předzpracování segmentovaného obrazu.
- Vhodně zvolit metriky pro segmentaci obrazu.
- Vytvořit nejenom jednu komplexní aplikaci, ale vytvořit i moduly, které se dají použít samostatně (například jako knihovna).

5.2 Výběr segmentačních metod

V této podkapitole se jedná především o správnou volbu topologie neuronové sítě a vhodný algoritmus učení tak, aby rychle ustálil. V druhé řadě jde o volbu referenčního segmentačního algoritmu pro srovnání neuronové sítě s běžně využívanou metodou. Z toho důvodu by obě metody měly být založeny na obdobném principu. Aplikace není zaměřená na určitý typ scén (kontrola kvality produktu na automatickém pásu, …), ale jde o to zmapovat, jak se jednotlivé algoritmy chovají v různých scénách. Pro výběr segmentačních metod si musíme nejdříve zodpovědět několik otázek a na základě jejích odpovědí můžeme volit jednotlivé metody.

Jsme schopni vědět předem, kolik objektů je ve scéně? Pokud se bude jednat o syntetické obrázky, je možné omezit počet objektů na určité číslo a dodržet barevnou skladbu objektů (příroda – zelená, hnědá, modrá; ulice – šedá, modrá, červená, zelená). Pokud se jedná o reálné obrázky, pak bez šablony není možné určit počet objektů, leda bychom osobně vybrali takové obrazy, které jsou si tématicky blízké a mají podobný počet objektů.

Jsme schopni vědět předem, jaké typy objektů jsou ve scéně obsaženy? Pokud se bude jednat o syntetické obrázky, je možné dodat informaci o typech objektů, které jsme vykreslili. Ale

vzhledem k předpokládanému užití na reálné obrazy, není možné dostatečně přesně popsat všechny objekty, které se ve scéně vyskytují.

Vzhledem k odpovědím na tyto dvě otázky je vhodné využít algoritmy implementující shlukování. Sem patří např. k-means a Kohonenovy mapy.

Algoritmus k-means jsem zvolila jako **referenční algoritmus** pro účely diplomové práce. Jedná se o základní algoritmus v umělé inteligenci pro implementaci shlukování, jeho výhody a nevýhody jsou zmapovány a různé segmentační metody jsou porovnávány převážně s tímto algoritmem. Důvody pro použití k-means jako referenčního algoritmu ke Kohonovým mapám jsou následující:

- K-means algoritmus je dobře znám.
- Má obdobné rysy klasifikace vstupních dat k jednotlivým shlukům.
- Má odlišný přístup k adaptaci sítě $\Rightarrow$ šance na vylepšení segmentace.

Porovnáme-li oba algoritmy, zjistíme, že používají stejnou hodnotící funkci, a to vzdálenost vstupu od středu shluku. Po přiložení trénovacího vektoru na vstupy algoritmu proběhne stejná fáze a jejím výsledkem je určení vítězného shluku neboli určení shluku, který je nejblíže vstupnímu vektoru. Po této fázi nastane fáze učení. Učení je v obou případech bez učitele. V dalším bodě se však jednotlivé algoritmy už liší. Kohonenova mapa mění středy vítězného neuronu po každém vstupu, navíc přidává tzv. sousedství. To znamená, že se žádný shluk nemůže jen tak odtrhnout od sousedů a pokud se přesune střed vítězného neuronu, pak se sousední shluky přesunou směrem k vítěznému shluku, což umožňuje odstupňovat náležitost k danému segmentu a zachovat určitou generalizaci. U k-means algoritmu je přístup k adaptaci odlišný. Středy shluků se mění až po projití všech prvků z trénovací množiny. Algoritmus odkládá rozhodnutí o změně středu, dokud nezná všechny potřebné informace o vstupních vzorcích. Algoritmus není schopen interpolace, protože každý shluk jedná sám za sebe.

V předchozím odstavci jsem použila rozdělení do dvou fází. Podíváme-li se na ně detailněji, pak první fáze představuje vlastní klasifikaci vstupních vektorů do určitého shluku, ke kterému naleží. Pro účely aplikace použijeme tuto fázi, abychom určili, jak úspěšné byly jednotlivé učící algoritmy na testovacích datech. Druhou fázi pak budeme nazývat optimalizace algoritmu, protože v ní dochází k adaptaci shluků na trénovací množinu.

Co tedy můžeme od Kohonenovy mapy očekávat? Přepočet po každé iteraci může vést k menšímu celkovému počtu iterací, které potřebujeme na adaptaci algoritmu, na úkor výpočetní náročnosti. Dále jemnější segmentaci a možnost rozdělení jednoho objektu na více segmentů, protože se objekt rozloží i na sousední shluky. Kohonenova mapa by tedy při segmentaci měla podle předpokladu mít úspěšnost srovnatelnou s k-means algoritmem, ale k její optimalizaci by mělo docházet rychleji.

5.3 Příprava trénovací a testovací sady

Pro úspěšnou optimalizaci algoritmů je potřeba početná množina trénovacích obrazových dat. Nezáleží jen na početnosti, ale také na optimálním složení a pořadí vybraných dat. Avšak základem pro vygenerování množiny je dostatečný počet obrázků, které odpovídají danému účelu . K vytvoření těchto obrázků vede několik cest:

- Snímání dostatečného počtu obrázků (reálné obrazy)
    - zdlouhavé,
    - vyžaduje přítomnost uživatele,
    - nelze do nich jednoduše zanést informaci o objektech.
- Vytvoření dostatečného počtu obrázků (syntetické obrazy)
    - lze automatizovat,
    - možnost vytvoření vzoru pro segmentaci.

Modul generátor obrazů má vytvořit syntetické obrázky, které by co nejvíce odpovídaly realitě, společně s vytvořením šablony pro určení objektů. Jak bylo uvedeno výše, barva pixelů v reálných obrazech často zcela neodpovídá barvě objektu. Navíc některé elementární objekty mohou mít na sobě nějaký vzor, dále označen jako textura (dřevěný objekt – letokruhy; atd.), u kterého nechceme, aby byl detekovaný jako složený objekt. Proto je potřeba ještě počítat s nanesením zvolené textury na syntetizovaný objekt. Abych nechala uživateli volnost v sestavení vhodných textur, rozhodla jsem se učinit generátor obrázků nezávislý na textuře. Generátor dostane k dispozici pouze složku v souborovém systému, kde jsou uložené jednotlivé textury, které se mají zobrazit. Popsaný přístup umožní uživateli použít i nafocené textury nebo vybrat nejvhodnější textury, které byli vygenerovány jinou aplikací. Z tohoto důvodu se mi jeví výhodné nechat generátor obrázků a generátor šumu odděleně pro případ, že by uživatel chtěl využít jiný software vhodnější pro jeho potřeby.

Při segmentaci obrazu jde hlavně o detekci objektu na základě jeho barevných vlastností (barva, textura). Pokud bereme v potaz okolí pixelu (např. při použití texturních příznaků), vždy dojde k modifikaci vektoru příznaků na hranách segmentu. Proto je důležité nebrat příznaky pouze z textur, ale vytvořit i scénu, kde spolu textury sousedí. Čím větší okolí pixelu zvolíme, tím větší bude vliv tohoto rozhraní. Protože je nutné použít v syntetických obrazech objekty různých tvarů pro simulaci rozhraní, zvolila jsem základní geometrické útvary:

- elipsa - zvláštní případ elipsy je kruh,
- obdélník – zvláštní případ obdélníku je čtverec,
- n-úhelník – zvláštní případ n-úhelníku je obdélník, ale také výsledek rasterizace elipsy.

Díky tomu, že v reálném světě se mohou předměty překrývat, lze toho využít i v syntetickém obraze a získat tak daleko rozmanitější tvary. Volba geometrických tvarů bude záviset na podpoře programovacího jazyka. Vzhledem k tomu, že při rasterizaci je většina tvarů vykreslena jako

n-úhelník, můžeme využít libovolného výše zmíněného tvaru. Vliv konvexních a nekonvexních útvarů by měl být zcela eliminován segmentací na základě barvy pixelu a u texturních příznaků je dán charakteristikou rozhraní. Přesto však zvolená metoda generování objektů umožní simulovat oba jevy.

Obrázek 5-2: *Příklady syntetických obrazů vygenerovaných podle návrhu. Obrazy pocházejí ze stejné sady generující 4 objekty ve scéně. Díky metodě překrývání je možné modelovat jak konvexní tvary (třetí obraz zleva), tak nekonvexní tvary (první a druhý obraz zleva) a navíc i nižší počet objektů než je zadáno.*

Implementovaný generátor obrázků by měl být schopen vytvořit náhodnou scénu o určitém počtu objektů a určitých rozměrech. Dále nanést libovolnou texturu na objekt a vytvořit šablonu (masku) tak, aby bylo možné později detekovat polohu a rozsáhlost objektů umístěných ve scéně. Označení segmentu pro objekt stejné textury by mělo být ve vygenerovaném souboru jednotné. Pozdější modifikace by byla složitá. Tento přístup umožní v budoucnu (aspoň částečně) zařadit mezi segmentační algoritmy metody založené na učení s učitelem.

Modul pro zanesení šumu do syntetického obrázku bude pracovat nad obrázky umístěné ve složce. Protože systém umožňuje několik variant pro přiblížení syntetického obrázku realitě, je tento modul v systému nepovinný a může být použit na různá obrazová data. Má však za úkol zanést do vstupního obrázku různé druhy šumu. Jednotlivé druhy šumu byly uvedené výše. Nyní se zaměřím jen na určení jejich parametrů. Při nanesení šumu typu *„pepř a sůl“* se jedná o náhodně zvolené pixely obrázku, které se změní buď na černou nebo na bílou barvu. Parametrem tohoto typu šumu je, kolik procent pixelů je takto změněno. Oproti tomu „Gaussův šum“ postihuje úplně všechny pixely a mění jeho hodnotu podle Gaussova náhodného rozložení. To znamená, že malé změny hodnoty pixelu jsou pravděpodobnější než změny velké. Parametrem Gaussova šumu je rozptyl. Příliš velký rozptyl může změnit celkový charakter scény.

Modul pro generování textury nebude mít žádné speciální parametry. Bude náhodně generovat textury zadaného rozměru podle níže uvedeného postupu a ukládat textury do jedné složky souborového systému. Jedním z parametrů bude formát uložení textury (BMP, JPEG, PNG atd.).

Vlastní generování textury může být značně složité. Existuje mnoho typů a nelze je všechny jednotně popsat. Pro účely aplikace jsem zvolila několik procedurálních textur. Tento modul je spíše podpůrný. Vzhledem k použití aplikace na obecné scény nelze do modulu zahrnout všechny druhy.

Jedna z možností pro generování textur je spíše logicky odvozena od volby prostoru příznaků. Již dříve jsem zmínila, že pro klasifikaci a analýzu textur se využívá Gaborových filtrů, které extrahují informaci o frekvenční charakteristice textury ve zvoleném směru. Je možné proces obrátit

a podle Gaborova filtru zmodifikovat vlastnosti jednobarevného obrazu v různých směrech a vytvořit tak texturu, která by byla nanesena na příslušný objekt.

Pro generování procedurální textury lze využít nejrůznější matematické funkce pro 2D vstupní prostor. Kombinací, posunem, rotací lze vygenerovat odlišné textury. Nejvhodnější k tomuto účelu jsou periodické funkce, které je možné nanášet do nekonečna.

Další z možností je využít Perlinova šumu. Ten se používá v počítačové grafice pro generování různých textur (dřevo, mramor, mraky, turbulence, vlasy atd.). Předmětem této diplomové práce není generování reálných textur pomocí Perlinova šumu, a tak bude implementována jen základní myšlenka.

Úkolem tohoto modulu je vytvořit textury, které pokryjí různé odezvy na zvolené Gaborovy filtry pro prostor příznaků, a připravit data pro automatickou tvorbu trénovací a testovací množiny obrázků.

5.4 Volba prostoru příznaků

Má vliv na průběh segmentačního algoritmu. Je třeba vědět, jaké vlastnosti jsou důležité pro danou segmentaci obrazu, abychom je mohli převést na číselnou reprezentaci a použít jako vstup algoritmu. Již dříve jsem se zmínila o atributech jednotlivých objektů ve scéně. Nyní se k nim vrátíme a zjistíme, jaké hodnoty barevného prostoru obrázku je reprezentují.

Barva objektu je základním identifikátorem. Člověk je schopen velice jasně vnímat hranici mezi dvěmi odlišnými barvami. Až při určení významu objektu se dívá na tvar a skládá barevné segmenty do větších celků. Barevný prostor může být reprezentován různými modely. Dříve jsem se zmínila o modelech RGB a HSV.

Model RGB potřebuje všechny složky k definici barvy objektu. Je vhodný pro syntetické obrázky, kde nedochází k výrazné modifikaci barvy v rámci daného objektu. Bohužel však v reálných obrazech nemá valný význam. Scény často mají nehomogenní osvětlení. Tudíž část objektu je vystavena světlu a část je ve stínu. Dochází k tomu, že takto osvětlené objekty jsou rozděleny na dva segmenty (světlejší a tmavší). Tento jev je pro segmentaci nežádoucí a způsobuje ho jasová složka, která nám říká, kolik světla z dané části scény snímač zachytil.

Model HSV umožní oddělit jasovou informaci, aniž by došlo ke ztrátě barvy objektu. Je tedy vhodné použít pro segmentaci v reálných systémech prostor HS k dosažení uspokojujících výsledků.

Bohužel objekty mohou být vytvořeny z různých materiálů (dřevo, žula, mramor, …), které nemají jednotnou barvu. Lidské oko však dokáže eliminovat barevné rozdíly a vnímat je jako jeden objekt. V počítačové grafice se pro reprezentaci barevných vlastností těchto materiálů využívá textur. Jsou těžko popsatelné, ale existují různé modely, které nám umožní rozlišit jednotlivé textury. Lze použít například tzv. Gaborových filtrů.

Při převodu textury na prostor příznaků lze využít **Gaborovy filtry** a aplikovat je v různých směrech. Například pro čtyři úhly 0°, 45°, 90° a 135°. Gaborovy filtry jsou osově (2 osy na sebe kolmé) i bodově symetrické. Proto lze použít natočení jen do 180° k pokrytí všech směrů. Volba okolí pro filtraci záleží převážně na vlnové délce. Při zvolení malého jádra a velké vlnové délky nemusí dojít k zachycení potřebné frekvence. Naopak při zvolení malé vlnové délky a velkého jádra dojde sice k zachycení frekvence, ale na úkor nadbytečné výpočetní náročnosti. Aplikace by tedy měla umožnit spravovat tyto filtry, aby uživatel mohl vhodně volit parametry.

Modul správce filtrů umožní uživateli vytvářet, měnit a prohlížet konvoluční filtry pro extrakci příznaků, ale i pro prefiltraci obrazů. Modul zapouzdřuje menší moduly pro generování konvolučních jáder různých typů filtrů. Tyto moduly mají společné rozhraní, které umožní zadávat parametry a získávat jádro filtru, které je uchováno pro další použití nebo zobrazeno. Správce filtrů je využíván vždy, když je potřeba zvolit vhodný prostor příznaků.

Modul generátoru množiny dat je používán pro převod obrazových informací na vektor příznaků. Je schopen uložit a načíst vygenerovanou množinu dat v textovém formátu. Modul je využíván převážně segmentačními algoritmy pro uchování trénovací množiny během učení. Je nositelem informace o zvolených příznacích a prefiltraci. Je schopen zajistit kompatibilitu dat s optimalizovaným segmentačním algoritmem a tím zabránit záměně vektorů příznaků a následné chybné segmentaci. Modul bude schopen generovat následující druhy příznakového prostoru:

- HS
    - 2D prostor příznaků,
    - reprezentuje plnohodnotnou informaci o barvě,
    - odstraňuje vliv nehomogenního osvětlení.
- RGB
    - 3D prostor příznaků,
    - podléhá vlivu nehomogenního osvětlení.
- Gaborovy filtry
    - vícedimenzionální prostor příznaků,
    - zaměřuje se na frekvenční analýzu okolí pixelu,
    - pomáhá v rozpoznávání podobných typů textur (dřevo, zeď, …),
    - je náchylnější na zvýraznění hrany objektu.
- HS nebo RGB s Gaborovými filtry
    - vícedimenzionální prostor příznaků,
    - přejímá vlastnosti předchozích prostorů,
    - pomáhá v rozpoznávání podobných typů textur (dřevo, zeď, …).

5.5 Optimalizace algoritmů

Při použití neuronových sítí se optimalizací algoritmu rozumí proces učení. Dochází k adaptaci středů shluků v algoritmu na předem danou sadu vstupních dat tak, aby byl výsledek této operace co nejlepší. Jedná se především o počáteční rozestavení shluků v prostoru. Díky vhodně zvolené inicializaci středů shluků, dokáží algoritmy rychle konvergovat a tím umožní rychlý průběh segmentace na odpovídající množině obrázků. V neposlední řadě může optimalizace vylepšit i stabilitu algoritmu. Tento proces je hlavním předmětem zkoumání mé diplomové práce.

Jednou z největších komplikací při učení algoritmů je přeučení, kdy dochází k příliš velké adaptaci na vstupní vzorky, které však nepokrývají celý vstupní prostor, ale pouze jeho část. Zbylá část prostoru je aproximována. Přeučení se projevuje tím, že opakovaným použitím vzorků mimo trénovací množinu dojde k chybným klasifikacím. Je potřeba, aby si segmentační algoritmus zachoval určitou generalizaci. Pro odstranění tohoto jevu se doporučuje využití cross-validace a dostatečně velká množina vstupních dat. Postup cross-validace byl vysvětlen v kapitole 3.3.

Učící algoritmus pro k-means je pevně daný, kdežto učící algoritmus pro Kohonenovy mapy, který se řídí vzorcem 3.4.3, umožňuje využití skoro libovolných matematických funkcí, které určí na základě iterace učící konstantu $\mu(k)$ a na základě vzdálenosti od vítězného neuronu $D(d)$. Vzhledem k experimentální části, která bude popsána níže, by bylo zajímavé srovnat vliv některých elementárních typů funkcí. Volené funkce by měly mít následující vlastnosti:

- Definiční obor funkce je v přirozených číslech $N = \{0,1,2,3,...\}$.
- Obor hodnot funkce je v intervalu $\langle 0,1 \rangle$, protože chceme, aby se váha středu přenesla směrem k vstupnímu vektoru, ale nikdy ne za něj.
- Průběh funkce je nerostoucí. Pro učící funkci $\mu(k)$ to znamená, že v prvních iteracích učení se nejvíce mění poloha středů shluků, a pak už dochází jen k menším úpravám, a tím pádem je zajištěná konvergence naučenosti sítě. Pokud by byla tato funkce rostoucí, pak by konvergence sítě nebyla zajištěna. Obdobné důvody platí pro volbu nerostoucího průběhu funkce $D(d)$. Zde by rostoucí průběh znamenal, že neurony, které jsou nejdále od vítězného neuronu, se posunou nejvýrazněji k němu, a to způsobí kolaps mřížky do jednoho bodu a nikoliv její úplné rozprostření po prostoru příznaků.

Segmentační modul je rozhraní pro použití libovolné segmentační metody. Umožní tak jednotný přístup k libovolné segmentační metodě a zajistí její nastavení parametrů, klasifikaci testovací sady a adaptaci na trénovací sadu. Modul umožní lepší statistické porovnání metod.

Cílem automatické optimalizace je vytvořit postup, který bude pro danou množinu dat stabilní, bude vykazovat nejlepší výsledky segmentace a bude dobře statisticky porovnatelný s libovolnou segmentační metodou.

5.6 Předzpracování segmentovaného obrazu

Předzpracování segmentovaného obrazu zaručí určitou stabilitu segmentace na reálných datech, které obsahují různé artefakty. Vzhledem k tomu, že optimalizace algoritmu se provádí na syntetických datech, umožní také přiblížení reálných dat k datům syntetickým. V této sekci jde především o odstranění šumu, a tak minimalizovat rozdíly pixelů v rámci jednoho objektu. K tomuto účelu slouží lokální filtry.

Prefiltrační modul lze přiřadit v systému na kterékoliv místo, kde je výstupem nebo vstupem obraz. Tento modul má na vstupu obrazovou informaci, tu modifikuje podle zadané množiny filtrů a na výstup vrací informaci ve stejném formátu. Využívá lokální filtry, a proto potřebuje na vstupu celé obrázky. Zaměřuje se především na konvoluční filtry, ale musí být schopen využít i mediánový filtr, který nepatří mezi konvoluční filtry, ale často se ve zpracování obrazu používá. Pro zadání konvolučních filtrů využívá modul správce filtrů.

Gaussův filtr je založen na konvoluci. Implementuje vážený průměr pomocí Gaussova rozložení. Parametry pro vytvoření konvolučního jádra jsou:

- střední hodnota v ose x,
- střední hodnota v ose y,
- rozptyl,
- velikost konvolučního jádra.

Pro Gaussovo rozložení, kde je střední hodnota $(0,0)$, dává filtr velkou váhu aktuálnímu pixelu a menší váhy pixelům okolním. Při použití tohoto filtru se nemění celková světlost obrazu, ale dochází k rozmazání hran a ztrátě detailů.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Obrázek 5-3: *Konvoluční jádro o velikosti 5x5 implementující Gaussův filtr s* $\sigma = 1$ *[11]*

Pro Gaussovo rozložení, kde není střední hodnota $(0,0)$, dojde navíc k posunu obrazu opačným směrem než ukazuje vektor střední hodnoty. Tento typ Gaussova filtru nemá velké použití, ale pro úplnost nechávám uživateli volbu pro jeho vygenerování.

Mediánový filtr je založen na průchodu okolí pixelu. Neaplikuje se konvoluce, ale z okolí je spočítán medián, který je přiřazen aktuálnímu pixelu. Medián je statistická hodnota, která říká, že 50 % prvků okolí je menší než vybraná hodnota a 50 % prvků okolí je větší. Díky tomu dokáže lépe reprezentovat střední hodnotu daného souboru než čisté průměrování. Časová a prostorová složitost tohoto filtru je dána výběrem řadícího algoritmu. Při použití tohoto filtru dochází k odstranění šumu a zaoblení ostré hrany. Jediným parametrem filtru je velikost okolí, které bereme v úvahu.

5.7 Zhodnocení výsledků segmentace

Cílem diplomové práce není jen vytvoření segmentačního algoritmu, ale i její objektivní zhodnocení úspěšnosti při použití. Proto je velice důležité implementovat metriky pro porovnání obrazových dat.

Vzhledem k tomu, že v rozsáhlejších scénách nebo při nepovedené segmentaci může být výčet objektů a segmentů podle metrik z kapitoly 4.4 velice zdlouhavý, vzniká potřeba vyjádřit kvalitu v jednom čísle. Pro experimentální část je nejdůležitější určit kvalifikaci pixelů. Zda byl pixel zařazen správně nebo špatně. Naštěstí počet správně zařazených pixelů a počet špatně zařazených pixelů musí být roven počtu pixelů v daném univerzu a celkovou úspěšnost lze vyjádřit jedním číslem.

Modul pro vyhodnocení segmentace se zaměří na implementaci popsané metriky. Modul je aplikován na výstup ze segmentačního algoritmu. Důležité si je uvědomit, že modul může být použit jen na syntetickou množinu dat, kde je vytvořena i šablona segmentace. Vstupem jsou dva obrázky:

- výstup ze segmentace,
- maska obrázku.

Výstupem je pak číselná informace o kvalitě segmentace.

5.8 Vytvoření struktury dat pro segmentaci

Vzhledem k tomu, že aplikace by měla implementovat segmentační systém a poskytovat plný komfort pro experimentální část systému, vzniká potřeba uchování výstupů z hlavních modulů a znova je použít pro jiný experiment s odlišnými parametry. V této fázi návrhu je vhodné uvažovat nad způsobem propojení modulů do jednoho celku. Může být přímé, zvláště pro účely grafického rozhraní, nebo přes souborový systém. Pro správnou funkci automatického systému nad souborovým systémem je nutné vědět, kde jsou umístěny potřebné informace, stanovit způsob ukládání a načítání informací. Informace v segmentačním systému mohou být dvojího typu:

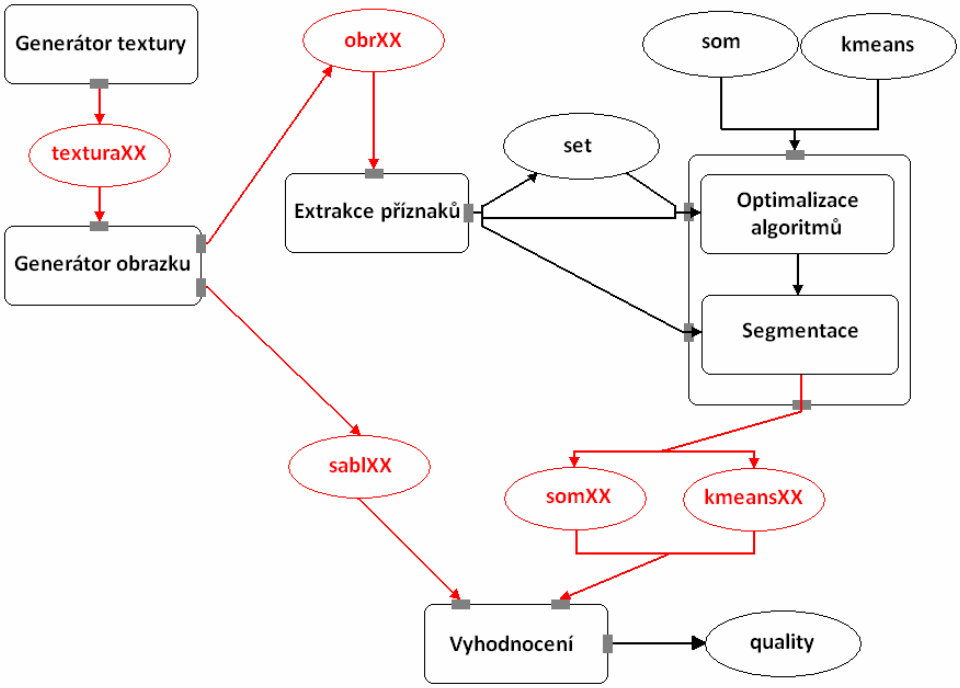
- Obrazové informace
    - množina obrazových dat (soubor obrázků),
    - množina kontrolních dat,
    - množina segmentovaných dat.

- Textové informace
    - množina trénovacích a testovacích dat,
    - informace o optimalizovaných algoritmech,
    - výsledky segmentace.

Je vidět, že bude potřeba na disku uložit mnoho informací a je potřeba jim dát řád.

Z hlediska experimentů budou navrženy dva způsoby pro práci se souborovým systémem. První způsob vyplývá z potřeby uložení výstupů jednotlivých modulů pro kontrolu dat a cílenou optimalizaci systému. Druhý způsob je pro provedení velké škály testů, které mapují stavový prostor segmentačního algoritmu na základě jeho parametrů.

Při cílené optimalizaci systému se předpokládá, že uživatel ví, jaké parametry využít a jen chce otestovat optimalizovaný algoritmus na cílové množině obrázků a případně upravit pár parametrů, pokud výsledek nevyjde podle očekávání. V tomto případě není nutné využít mnoho nastavení a lépe než dodatky v názvech souboru bude lepší, když se vytvoří nové adresáře, pro lepší orientaci uživatele. Jde tedy o vytvoření adresářové struktury projektu, a tudíž musí být stanovena pravidla integrity, která ochrání data před přepsáním jiným modulem. Vzhledem k použití automatického systému se při novém spuštění stejného modulu data přepíšou. Někdy to je vítaná vlastnost, jindy to je spíše na škodu. Každopádně uživatel si může data zálohovat i překopírováním zajímavých dat do jiného adresáře. Dost modulů v systému funguje na náhodné volbě, a tak výstupy dvou běhů stejného modulu nejsou zcela totožné. Pro stanovení pravidel je třeba si uvědomit, v jakém pořadí budou jednotlivé moduly volány, aby umožnily optimální segmentaci obrazu.

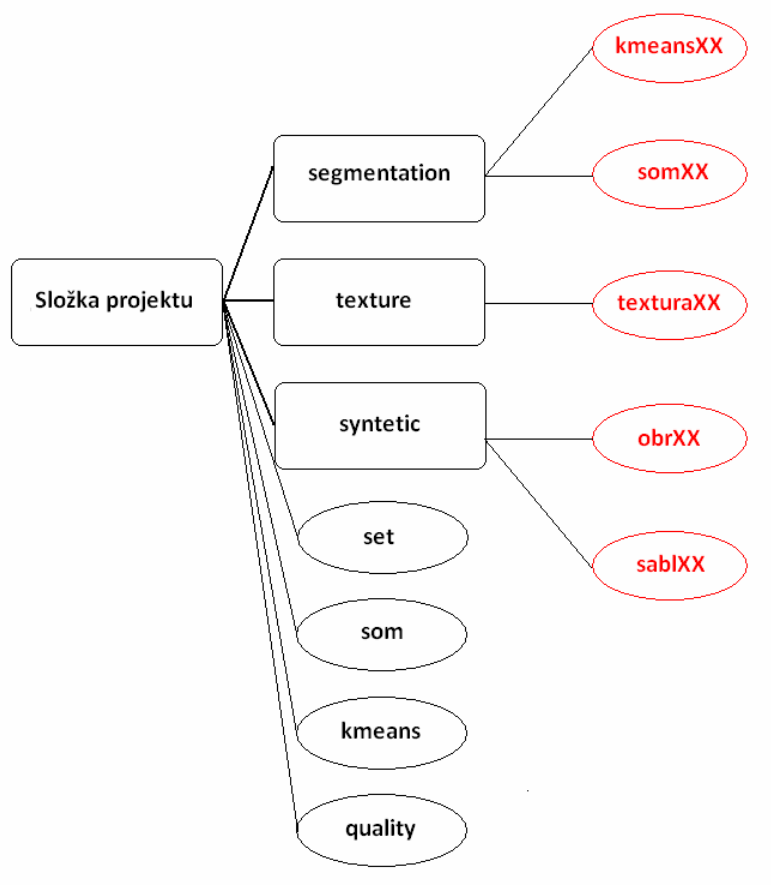


Obrázek 5-4: *Schéma procesu segmentace – znázorněno pomocí modulů (zaoblený obdélník) a jejich komunikace nad souborovým systémem (elipsa: červená – obrázek, černá – textový soubor)*

Víme již, jaké soubory jsou v celém systému segmentace využity a kdy. Můžeme tedy navrhnout pravidla přístupu a formátu jednotlivých typů souborů:

- texturaXX
    - představuje texturu, která bude nanášena na objekt v generátoru obrázku,
    - je potřeba, aby měla stejnou nebo větší velikost jako generovaný obrázek,
    - je v obrazovém formátu (BMP, JPEG atd.);
- obrXX
    - syntetický obrázek,
    - modifikovaný tak, aby odpovídal reálnému obrázku,
    - vstup do modulu extrakce příznaků,
    - ke každému obrázku musí být vygenerována i šablona,
    - je v obrazovém formátu (BMP, JPEG atd.);
- sablXX
    - syntetický obrázek,
    - má stejný obsah jako obrXX, ale bez použití textury a šumu,
    - slouží k vyhodnocení segmentace,
    - je v obrazovém formátu (BMP, JPEG atd.),
- set
    - textový soubor s vlastním formátem,
    - slouží pouze k optimalizaci algoritmů,
- som, kmeans
    - textový soubor s vlastním formátem,
    - slouží pro inicializaci segmentačního algoritmu,
- somXX, kmeansXX
    - segmentovaný obrázek,
    - vznikl použitím optimalizovaného segmentačního algoritmu (produkt segmentace),
    - je v obrazovém formátu (BMP, JPEG atd.),
- quality
    - textový soubor,
    - sdružuje výpis kvality segmentace nad jednotlivými obrázky pro všechny segmentační algoritmy.

Každý modul může modifikovat pouze složku pro svůj výstup. Jména souborů jsou daná. Z toho vyplývá, že při dalším běhu daného modulu, jsou data přepsána.



Obrázek 5-5: *Schéma adresářové struktury dat při segmentaci*

Pro účely diplomové práce je však třeba využít i jiného zapojení jednotlivých modulů. Ze začátku bude potřeba prohledat stavový prostor pro jednotlivé metody segmentace. Prohledávání bude probíhat zcela automaticky podle příslušného nastavení. Ukládány jsou jen výsledky segmentace:

- optimalizovaný algoritmus,
- jejich zhodnocení.

Zde by sice šlo použít pro každé nastavení i adresářovou strukturu popsanou výše, ale byla by příliš komplikovaná. Záleželo by na zvolené hodnotící funkci, učící funkci, parametrech učící funkce, sousednosti shluků a iterací. Vzhledem k tomu, že soubor s metrikou by měl propojovat všechny tyto složky s předpokládanou adresářovou hloubkou 5, je lepší, když výsledkům jen přidáme přízviska a uživatel si je může vyfiltrovat dle potřeby pomocí regulárních výrazů různých vyhledávačů. Přízviskem rozumím jednoduché pojmenování přiřazené k originálnímu jménu souboru. Více se tím zabývá Tabulka 5-1.

Tabulka 5-1: *Popis použitých přízvisek v názvech souborů pro experimentální režim aplikace*

přízvisko	hodnota		popis
	integer	double	
_top	A	N	uspořádání shluků dle interní reprezentace
_met	A	N	hodnotící funkce dle interní reprezentace
_lf	A	N	učící funkce dle interní reprezentace
_rad	A	N	velikost okolí
_pr1	N	A	první prametr funkce
_pr2	N	A	druhý parametr funkce
_it	A	N	zachyceno v iteraci

Soubor s metrikou by měl být ve formátu CVS, pro lepší import do tabulkových programů, které dokáží snadno dělat grafy a filtrovat parametry (např. MS Excel, OpenOffice Calc). Moduly by měly být schopny načíst libovolně pojmenovaný soubor a pokračovat tak v učení za pomoci jiných parametrů, například pro zjemnění kroku v intervalech.

6 Implementace segmentačního systému

Tato kapitola se bude zabývat vlastní implementací segmentačního systému. Podá podrobný popis výše navržených modulů. Nebude se jen zaměřovat na jejich propojení, ale již také na vlastní provedení a představí vlastnosti konkrétních algoritmů.

Nejdříve se zaměří na obecné řešení softwarové části diplomové práce. Následně uvede vlastní implementaci jednotlivých modulů tak, aby v experimentální části bylo jasné, jak se daný algoritmus chová a aby bylo možné už jen shrnout výsledky.

6.1 Vývojové prostředí

Pro účely této diplomové práce jsem se rozhodla využít programovací jazyk Java ve verzi JDK 1.6.0_24. Důvodů pro použití tohoto programovacího jazyka bylo více:

- přenositelná aplikace (pro spuštění stačí mít nainstalován Java Virtual Machine obsažený v JRE 1.6.0_24, který je dostupný pro nejrůznější operační systémy),
- podpora objektově-orientovaného programování,
- jednotně zdokumentované knihovny pomocí Java Doc,
- v základní implementaci podporuje většinu důležitých nástrojů pro práci s GUI, obrazy, videem, XML formátem atd. a není tudíž potřeba používat další knihovny,
- dostupná knihovna s otevřeným kódem pro Kohonenovy mapy.

Hlavní nevýhodou tohoto programovacího jazyka je ale horší správa paměti. Systém alokuje a uvolňuje paměť automaticky (Garbage collector) nemá explicitní příkaz pro tyto úkony. Dobrým programátorským stylem je možné přiblížit časovou složitost aplikací napsaných v Javě, stejným aplikacím napsaných v C++. Časová složitost je sice u zpracovávání obrazů důležitý parametr, ale vzhledem k tomu, že oba segmentační algoritmy budou implementovány ve stejném programovacím jazyce, je jejich časová složitost srovnatelná. Ukáže-li se, že jeden algoritmus je rychlejší v této implementaci, pak pravděpodobně bude rychlejší i v jiných implementacích. Navíc v této diplomové práci jde především o kvalitu segmentace než o její časovou složitost.

Aplikace byla vytvořena jako projekt ve vývojovém prostředí NetBeans 6.9.1. pod operačním systémem Windows 7. Aplikace byla testována na operačních systémech Windows XP.

6.2 Knihovny

V aplikaci využívám dvě knihovny. První knihovna implementuje Kohonenovu mapu a druhou knihovnu používám k ukládání a načítání souborů ve formátu XML.

Java Kohonen Neural Network Library (JKNNL) [16] je knihovna pro práci s Kohonenovou mapou. Implementuje různé učící algoritmy, topologie, hodnotící funkce. Výhodou knihovny je distribuce i se zdrojovými soubory a BSD licence. Tato knihovna je v implementovaném systému zapouzdřena do objektu SegmentsByKohonen.

Knihovna se skládá ze šesti balíčků zdrojových kódů:

- Topology
    - Balíček sdružuje objekty implementující sousednost neuronů v síti.
    - Umožní navolit počet sousedů: Matrix (4 sousedé) a Hexagonal (6 sousedů).
    - Rozhraní pro implementaci libovolné funkce $D(d)$, která je použita ve vztahu 3.4.3.
    - Implementace $D(d)$ pomocí Gaussovy funkce.
- Network
    - Balíček sdružuje třídy a rozhraní pro vytvoření funkční neuronové sítě.
- Metrics
    - Balíček sdružuje třídy implementující hodnotící funkce neuronů.
    - Zahrnuje i rozhraní pro implementaci nových typů funkcí.
- LearningFactorFunctional
    - Balíček sdružuje implementace funkcí, které určují faktor $\mu(k)$ (použit ve vztahu 3.4.3).
    - Zahrnuje i rozhraní pro implementaci nových typů funkcí.
- Kohonen
    - Balíček sdružuje implementace různých algoritmů pro učení sítí (Winner Take All a Winner Take Most).
    - Zahrnuje i rozhraní a implementaci pro trénovací sadu, která je používaná v učících algoritmech.
- ActivationFunction
    - Balíček sdružuje implementace aktivačních funkcí neuronů.
    - Zahrnuje i rozhraní pro jejich implementaci.
    - Tyto zdrojové kódy nebyly použity k implementaci popisovaného systému.

Java-based Dokument Object Model (JDOM) [17] je knihovna pro tvorbu XML dokumentů. Zajišťuje komplexní řešení pro přístup, manipulaci a výstup XML dat z instancí Java objektů. Využívá již existující standardy pro práci s XML dokumenty jako jsou SAX (Simple API for XML) a DOM (Document Object Model). XML formát je reprezentován několika objekty. Celý dokument reprezentuje třída Document. Přes tuto třídu je možné přistupovat k jednotlivým elementům, které reprezentuje třída Element, jejich atributům, které reprezentuje třída Attribute, a jmenným prostorům,

které reprezentuje třída Namespace. Načítání dokumentů zajišťuje třída SAXBuilder a výstup do souboru je realizován přes třídu XMLOutputter.

Knihovna je vyvíjená v rámci Java Community Process a vztahuje se k ní licence Apache-style open source license.

6.3 Přínos objektově orientovaného přístupu programovacího jazyka Java vzhledem k návrhu systému

Objektově orientované paradigma programovacího jazyka Java přináší do kódu větší přehlednost a snadnou čitelnost. Snažila jsem se využít naplno všech možností, které Java nabízí pro lepší rozšiřitelnost aplikace do budoucna. Jedná se hlavně o využití statických knihoven, výčtových typů na bázi objektu, dědičnosti a rozhraních. V této podkapitole bych se chtěla věnovat hlavně organizaci API a jeho využití pro vytvářenou aplikaci s grafickým uživatelským rozhraním.

Vytvořené třídy jsou rozděleny do balíčků podle funkčnosti. Základní balíček se jmenuje „segment“. Obsahuje třídy pro práci s grafickým rozhraním a grafické formuláře, které pomáhají hlídat rozsah vstupních hodnot. V balíčku se přímo nachází i třída „SegmentationApp“, která má na starosti spuštění aplikce. Tyto třídy využívají balíčku „api“, jenž sdružuje všechny moduly popsané v předchozí kapitole a algoritmy, které je implementují. Třídy obsažené v balíčku „api“ jsou rozděleny na „generators“, “metrics“, “segmentalg“, “tools“. Budou popsány v dalších podkapitolách.

Java umožňuje vytvořit třídy, které sdružují operace stejného druhu bez nutnosti vytváření instance třídy. Jedná se o funkce, u kterých není potřeba uchovávat vnitřní stav pro další volání. To umožní efektivní volání často využívaných metod v různých třídách pomocí následujících knihoven:

- ImageOperation
    - Třída obsahuje funkce související s operacemi nad obrázky.
    - Jedná se především o otevírání a ukládání obrazů do grafických formátů, konvoluci, nanesení šumu, mediánový filtr, normalizace vektorů atd.
- MyMath
    - Třída obsahuje funkce související s jednoduchými matematickými úkony, které nejsou implementovány ve třídě Math.
- XMLoperation
    - Třída řeší nástavbu nad knihovnou JDom.
    - Zabývá se především otevíráním a ukládáním XML souborů.
- Utils
    - Třída sdružuje statické funkce, které nezapadají do výše popsaných profilů.

    - Jedná se o převod polí na řetězce, vyhledávání a převod čísel v řetězcích, ukládání textových souborů apod.

Výrazné ulehčení pro implementaci a rozšíření aplikace představuje definice výčtového typu v Javě. Je k němu rovněž přistupováno jako k objektu, což umožní přiřadit nejen jméno prvku pro použití v kódu, ale také jiné objekty (index v poli, jméno zobrazované v grafickém rozhraní atd.) a třeba i metody pracující s tímto prvkem. Toho jsem využila pro proměnlivé nastavení jednotlivých algoritmů. Pro sestavení odpovídajícího API jsem připravila následující výčtové typy:

- ColorMod
    - Představuje typy barevného prostoru, který jsem použila pro implementaci příznakového vektoru a pro filtraci obrazu.
    - Tento výčtový typ uchovává informaci o dimenzi jednotlivých barevných prostorů, unikátní název pro textové vyjádření zvoleného barevného prostoru a uspořádání jednotlivých prvků tak, aby se dalo snadno projít všechny barevné prostory.
    - Jsou zde implementovány také dvě statické metody, jedna pro získání instance výčtového typu na základě řetězce použitého v textovém souboru a druhá pro získání všech identifikačních řetězců, což je vhodné například pro inicializaci JComboBoxů.
- HSV, RGB
    - Zde lze najít informace o jednotlivých složkách barevného modelu HSV.
    - Jedná se hlavně o index v poli, které určuje barvu podle zadaného modelu.
    - V celém API je používáno získání barvy na základě tohoto indexu, což umožní snadnou implementaci přeuspořádání nebo naopak, pokud si není uživatel API jistý indexem, pak použít právě tento výčtový typ.
- EnumDistance
    - Představuje typy pro volbu hodnotící funkce v algoritmech.
    - Umožňuje jednoznačný překlad jak na index v poli, tak na textovou hodnotu.
- EnumLearnF
    - Představuje typy pro volbu funkce na určení učícího faktoru.
    - Sdružuje informace nejen o indexu v poli a textové hodnotě, ale také o parametrech jednotlivých funkcí
- EnumTopology
    - Představuje typy pro volbu topologie a sousednosti u Kohonenovy mapy.
    - Má stejné využití jak EnumDistance.

Hlavní výhodou použití výčtových typů ve výše popsaných případech je opravdu snadná rozšiřitelnost kódu nebo případná reorganizace aplikace v datové oblasti.

Použití výčtových typů je nasnadě, pokud je potřeba mít pod kontrolou různé implementace stejného rozhraní (například z důvodu výběru správné instance objektu v GUI). Jak je zřejmé

z dřívějšího popisu „EnumDistance“ umožňuje držet si velmi jednoduše informaci o instanci objektu, který implementuje rozhraní „MetricModel“, stejně jako „EnumLearnF“ vychází z rozhraní „LearningFactorFunctionalModel“ a jiné. Použití rozhraní v tomto typu aplikace je velice běžné. V předchozí kapitole jsem zdůrazňovala potřebu zachovat stejný přístup k modulům. Ta vychází z experimentální části diplomové práce, kdy je nutné zachovat počáteční podmínky pro různé implementace, a tím efektivněji porovnat jednotlivé metody. Z toho důvodu jsem naimplementovala následující rozhraní:

- Segmentation – jednotný přístup k algoritmům k-means a Kohonenova mapa.
- KernelCreator – jednotný přístup ke generátorům filtrů.
- Metrics – jednotný přístup pro všechny metody měřící úspěch algoritmů.
- Generator – jednotný přístup ke všem generátorům obrazu.

Využití dědičnosti v systému není na první pohled patrné jako použití rozhraní. Vyskytlo se zde pouze pár případů. Pokud pominu dědičnost grafických prvků pro přizpůsobení jejich vzhledu. Pak jsem dědičnost použila pouze v případě, kdy bylo nutné k reprezentaci obrazu pomocí třídy „BufferedImage“ přidat i informaci o jméně souboru, ve kterém byl jeho obsah uložen, a i nadále se chovat k objektu jako k obrazu. Pro tuto příležitost jsem vytvořila třídu „MyBufferedImage“.

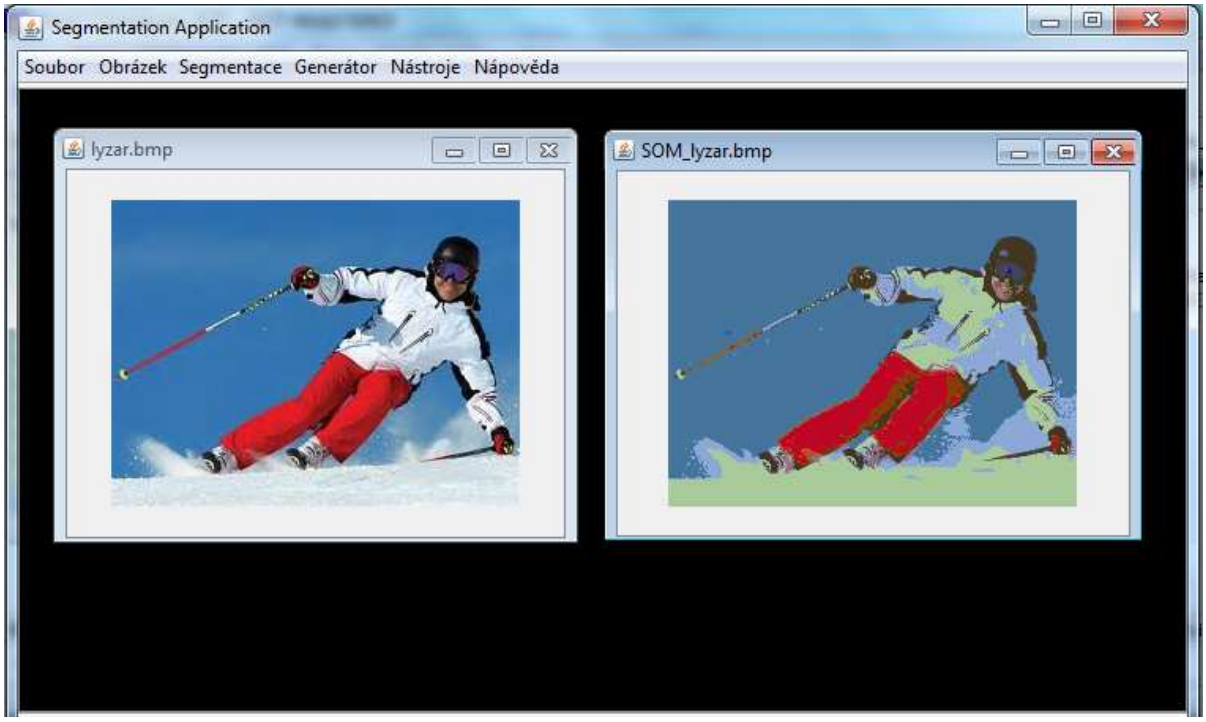
6.4 Implementace grafického rozhraní

Pro snadnější adaptaci segmentačních algoritmů na malou množinu testovacích obrázků a možnost vizuálního porovnání výsledků segmentace jsem se rozhodla vytvořit grafické rozhraní pro navrženou knihovnu. Toto rozhraní umožňuje pro uživatele, který nemá přístup ke zdrojovým kódům, zadání potřebných parametrů přes formulář, který pomáhá dodržet jejich požadovaný rozsah a typ. Po potvrzení formuláře dojde k automatickému sestavení potřebných modulů podle zadaných parametrů. Aplikace využívá grafické třídy definované v balíčku javax.swing a knihovny podporované vývojovým prostředím Netbeans pro snadnější uspořádání grafických prvků v JPanelu.

Grafický vzhled aplikace se odráží na potřebě zobrazení více obrázků v jeden časový okamžik. Toho je docíleno pomocí JDesktopPanelu. JDesktopPanel je kontejner, který uchovává informace o všech otevřených oknech (instance třídy JInternalFrame) v rámci aplikace a dokáže určit aktivní okno. V horní části aplikace se nachází menu, které se snaží dodržet zaběhnuté zvyky uživatele. Nejlevějším prvkem je sekce „Soubor“, kde uživatel najde volby pro otevření obrazového souboru, nastavení projektu a zavření aplikace. Následují jednoduché operace nad obrazem v sekci „Obrázek“, dále je možné začít s navolením parametrů pro segmentační algoritmy v sekci „Segmentace“. Sekce „Generátor“ umožňuje vygenerování syntetických dat pro realistické zobrazení a konečně v sekci „Nástroje“ jsou rozšířené volby, jako například správce filtrů a prohledávání stavového prostoru algoritmů pro účely experimentů. Aplikace nenabízí zkrácenou verzi menu po kliknutí pravým

tlačítkem myši. Je však možné se dostat k jednotlivým modulům pomocí klávesových zkratek, které jsou uvedeny v menu.

Grafické rozhraní zpracovává jen část možností pro využití implementovaného API.



Obrázek 6-1: *Ukázka pracovní plochy aplikace vyvinuté pro práci se segmentačními algoritmy*

6.5 Generátor textury

Modul je určený k náhodnému výběru procedurální textury a k jejímu uložení pomocí standardního formátu BMP. Jedná se o první modul, který by měl být využit pro sestavení syntetické množiny trénovacích i testovacích obrázků. Je ho však možné nahradit i ručním výběrem a extrakcí reálných textur z obrazů.

Generátor vytvoří ve složce projektu složku „texture“, do které následně uloží textury pod názvem vygenerovaným na základě následujícího vzorce:

jméno_souboru = „texture“+číslo_obrazu+přípona, 6.5.1

kde číslo_obrazu určuje pořadí, ve kterém byl obraz vygenerován, s oborem hodnot 0,1,2, … počet_obrazů – 1 a přípona je textový řetězec určující formát souboru, který je momentálně nastaven na hodnotu „.bmp“. Vzor textury je vytvářen ve stupni šedi. Generátor pak vybere náhodně barvu a podle textury zmodifikuje její jas. Procedurální textura je trojího typu.

Goniometricky založená textura implementuje náhodný výběr parametrů θ pro úhel otočení, f pro frekvenci goniometrické funkce a k pro posun funkce v daném směru. Volba výsledného jasu val pixelu na pozici x a y je určena následujícími vztahy:

$$val'_x = x \cdot \cos\theta + y \cdot \sin\theta \qquad 6.5.2$$

$$val'_y = y \cdot \cos\theta - x \cdot \sin\theta \qquad 6.5.3$$

$$val_x = \frac{\sin\theta(val'_x * 2 * \pi * f)}{8} + 0{,}375 + k \qquad 6.5.4$$

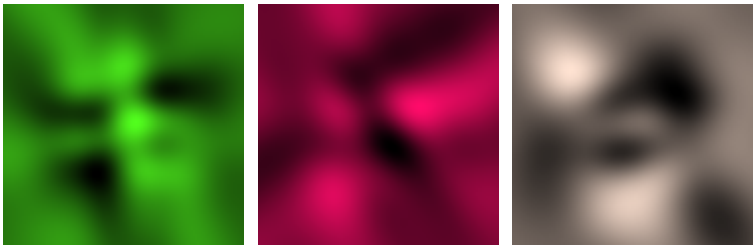
$$val_y = \frac{\sin\theta(val'_y * 2 * \pi * f)}{8} + 0{,}375 + k \qquad 6.5.5$$

$$val \in \left\{ val_x, val_y, \frac{val_x + val_y}{2} \right\} \qquad 6.5.6$$

Textura je schopna vytvořit simulaci různých frekvencí v různých směrech, bohužel v jedné textuře jsou z zachyceny maximálně dva směry.

Obrázek 6-2: *Příklady textur vygenerovaných pomocí goniometrických funkcí*

Skládání Gaborových filtrů má oproti předchozí textuře výhodu v jisté nepravidelnosti. Umožňuje zachytit v jedné textuře více frekvencí a směrů. Jedná se vlastně o analogii k vytvoření prostoru příznaků, který je použit v tomto systému. Pomocí třídy „GaborFilter“ vytvořím náhodný počet jader filtrů o velikosti výsledného obrazu. Vytvořené filtry sečtu do jednoho jádra, které pak převedu na stupně šedi.



Obrázek 6-3: *Příklady textur vygenerovaných pomocí Gaborových filtrů*

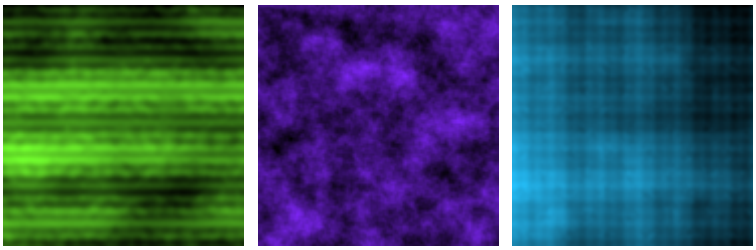
Perlinův šum je hojně využívaný v počítačové grafice pro reprezentaci náhodných textur, které po vhodné modifikací mohou vypadat realisticky. K výpočtu Perlinova šumu je zapotřebí vygenerování jednorozměrného náhodného pole permutací a náhodně vygenerovaného pole gradientů, které je v dimenzi podle prostoru scény (v tomto případě 2D).

Implementace této procedurální textury vyžaduje funkci $noise(x, y)$, která interpoluje polohu bodu ve scéně s uloženými gradienty. Ty jsou zvoleny na základě pole permutací. Hodnota pixelu na pozici x a y je určena následujícím vztahem:

$$PerlinNoise_{a,b,n}(x,y) = \sum_{i=0}^{n} \frac{noise(x \cdot b^i, y \cdot b^i)}{a^i}, \quad 6.5.7$$

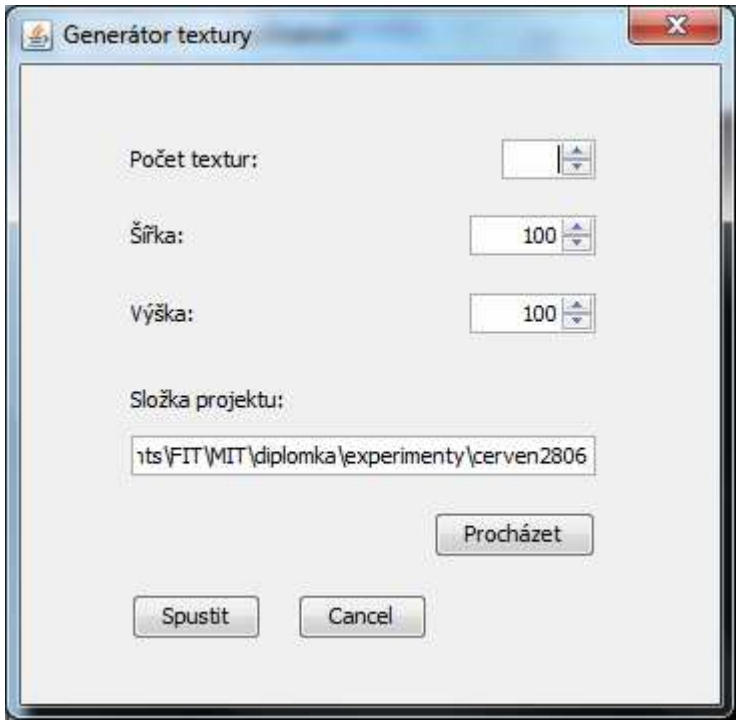
kde a představuje amplitudu a b je frekvence.

Skládáním této funkce s dalšími matematickými funkcemi lze dosáhnout různých efektů, které mohou vést k realistickému zobrazení mraků, vody, lesů a dokonce i k západu slunce.



Obrázek 6-4: *Příklady textur vygenerovaných pomocí Perlinova šumu*

Tento modul je implementovaný třídou „segment.api.generators.GenTexture“ a používá rozhraní „Generátor“. Třída obsahuje dva atributy, které je potřeba zadat v konstruktoru, jedná se o výšku a šířku obrázku. Dále obsahuje pouze jednu veřejnou metodu, která vygeneruje zadaný počet obrázků a uloží je do složky předané přes argument funkce a konečně více privátních metod, které implementují procedurální textury. Z grafického rozhraní je modul přístupný přes formulář „segment.GenTextureDialog“, kde je potřeba zadat výše popsané parametry.



Obrázek 6-5: *Ukázka grafického formuláře pro zadání parametrů do generátoru textury*

6.6 Generátor syntetických obrazů

Modul je určen k sestavení scén z objektů. Má dva režimy. První režim pouze přiřadí objektu barvu a druhý režim načte textury ze složky „texture“, které jsou přiřazeny k jednotlivým objektům. Zároveň vytváří i šablonu obrázků, která neobsahuje žádnou texturu a stejná hodnota barvy značí jeden objekt. Generátor ukládá vygenerované obrázky do složky „syntetic“, kterou vytvoří ve složce projektu. Jak již bylo zmíněno, výstupem modulu jsou vždy dva typy obrazových souborů:

- syntetický obrázek – jeho název je dán následujícím vzorcem:

  jméno_obrázku = „obr“+číslo_obrázku+přípona 6.6.1

- šablona (maska) obrázku – jeho název je dán následujícím vzorcem:

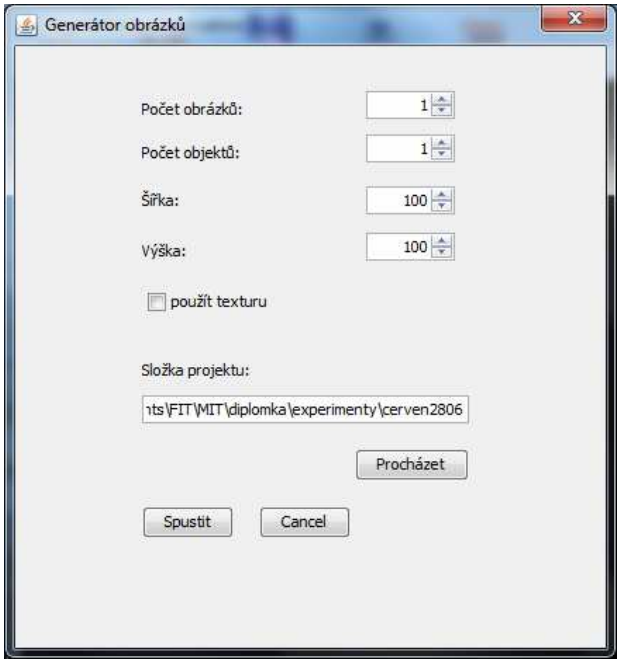
  jméno_šablony = „sabl“+číslo_obrázku+přípona 6.6.2

Je zřejmé, že obrázek se šablonou je propojen pomocí čísla_obrázku. Z toho důvodu bylo potřeba implementovat v knihovně Utils funkci pro načtení celého čísla z řetězce za zadaným identifikátorem a před oddělovačem „.“.

Vlastní generování scény je pak založeno na náhodném výběru tvaru objektu (obdélník, elipsa) a na pseudonáhodném výběru umístění a velikosti objektu. Pseudonáhodným výběrem se rozumí použití třídy „GridImage“, která rozdělí obrázek do mřížky o zadaném počtu buněk. V tomto případě se jedná o mřížku, jejíž počet buněk v mřížce, je blízký počtu objektů v obraze, s maximálním velikostí 8 sloupců a 7 řádků. Pomocí této mřížky jsou vygenerovány levý horní a pravý dolní roh bounding boxu objektu. Smyslem tohoto pseudonáhodného výběru je zajištění alespoň jednoho rohu v každé buňce mřížky.

Šablona obrázku je vytvářena zároveň s obrazem. Pokud je zvolena jen volba barvy, pak je šablona obrázku shodná s originálem. Jedná-li se však o texturované objekty, pak je barva segmentu určena pomocí hashovací tabulky na základě pořadí načtení textury. Tím je zajištěna jednoznačná identifikace textury ve vygenerovaném souboru.

Tento model je implementovaný třídou „segment.api.generators.GenColor“ a používá rozhraní „segment.api.generators.Generator“. Stejně jako generátor textury definuje pouze jednu veřejnou metodu danou rozhraním a více soukromých metod pro její implementaci. Při vytváření instance třídy je potřeba zadat počet objektů ve scéně a její rozměry. Pro generování příslušného počtu obrazů je nutné volané metodě zadat počet obrazů, jejich umístění a pole textur. Není-li pole textur zadané, pak dojde pouze k přiřazení jednotné barvy objektu. Z grafického rozhraní je modul přístupný přes formulář „segment.GenColorDialog“, který je zodpovědný za správné zadání parametrů.



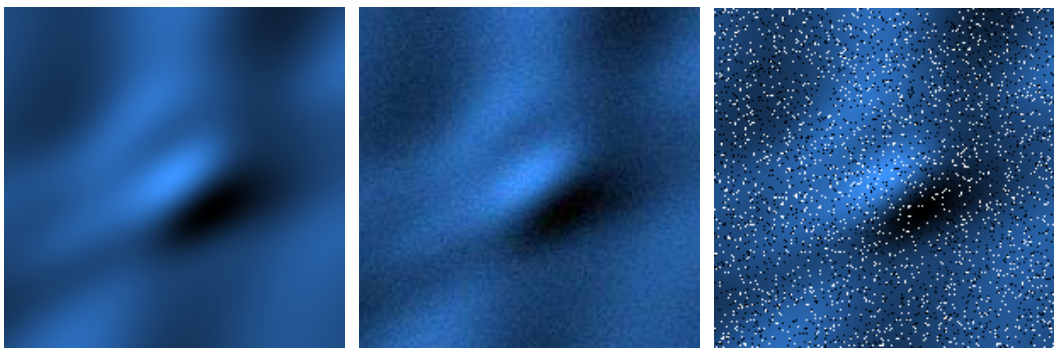
Obrázek 6-6: *Ukázka grafického formuláře pro zadání parametrů generátoru obrázků*

6.7 Generátor šumu

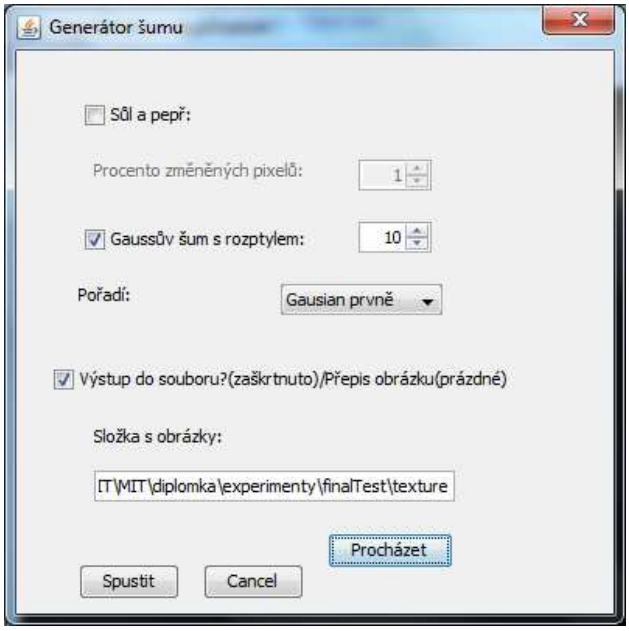
Modul je určen k zanesení šumu do obrazové informace. Implementuje výše popsané typy šumu „Sůl a pepř“ a „Gaussův šum“. Vstupem je adresář, ve kterém se nachází jen obrazy, do kterých má být zanesen šum. Výstup je dvojího typu:

- vstupní obrazy jsou přepsány výstupem,
- v adresáři je vytvořena nová složka s názvem určující parametry šumu a obsahující výstupní obrazy se stejným názvem, jako měly vstupní obrazy.

Hlavní algoritmy jsou velmi jednoduché. Proto jsou implementovány pomocí příslušné funkce v knihovně ImageOperation. Přes grafický formulář „segment.GenNoiseDialog“ je možné zadat platné vstupní hodnoty. Parametr šumu „Sůl a pepř“ je zadán v procentech (1-100) a parametr „Gaussova šumu“ určuje, o kolik hladin je možné změnit hodnotu každé složky v RGB modelu.



Obrázek 6-7: *Ukázka použití generátoru šumu na texturu (vlevo je originální obrázek, uprostřed „Gaussův šum“ s rozptylem 10 pixelů, vpravo šum „Sůl a pepř“ s parametrem 10 %)*



Obrázek 6-8: *Ukázka grafického formuláře pro zadání parametrů generátorů šumu*

Použití modulu je jen na volbě uživatele a může být aplikován na textury nebo na syntetické obrázky.

6.8 Správce filtrů

Modul je důležitý pro snadnou správu konvolučních filtrů určených k filtraci příznaků. Jedná se o to, aby nebylo potřeba znovu generovat ty samé filtry a po vytvoření je znovu použít v nových běžích aplikace. To je důvod propoužití souboru ve formátu XML, který obsahuje všechny filtry vytvořené aplikací. Každý filtr má:

- jméno – jednoznačně identifikuje filtr,
- velikost – šířka i výška konvolučního jádra je shodná,
- hodnoty konvolučního jádra – pole, které nese hodnoty konvoluce.

Celý modul se skládá z několika částí. První zahrnuje prostředí pro definici libovolného konvolučního jádra, několik generátorů známých filtrů (Gabor, Gauss), které podle parametrů vygenerují jádro, nástroje pro normalizaci filtrů a algoritmus konvoluce v knihovně ImageOperation.

Konvoluční mechanismus byl popsán již dříve. Nyní se zaměřím pouze na jeho implementaci. Využití konvoluce nabízí i sama Java pomocí třídy ConvolveOp. Protože jsem si nebyla jistá, zda obsahuje i normalizaci výstupu filtru, kterou bylo potřeba použít pro Gaborovy filtry, implementovala jsem konvoluci sama podle algoritmu popsaného v kapitole2.3. Pro hraniční pixely obrazu umožňuje:

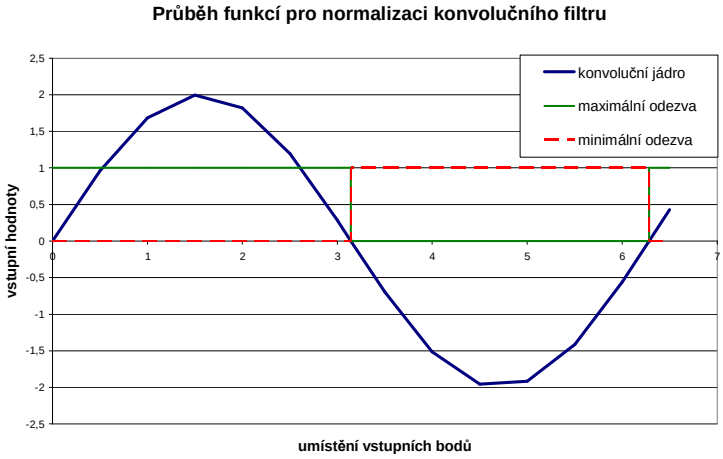
- doplnění okolí o nuly
    - přidává do obrazu hranu,
    - mění barevné i texturní vlastnosti obrazu za hranicí scény,

- nevhodné pro použití obou typů příznaků zvolených pro segmentaci obrazu;

- šíření posledního pixelu v daném směru
  - mění texturní vlastnosti obrazu za hranicí scény,
  - frekvence je uchována alespoň v jednom směru (osa x nebo y),
  - zůstává referenční informace o barvě,
  - vhodnější způsob pro segmentaci obrazu.

Je možné provádět konvoluci nad jednotlivými složkami RGB modelu (vhodné pro prefiltraci) nebo jen nad stupni šedi (vhodné pro texturní příznaky).

Normalizace filtru je vhodná pokud potřebujeme zaručit výstup filtru v určitém rozmezí. Konvoluce sama neklade žádnou podmínku pro dodržení výstupního intervalu. Proto je potřeba ji dodat explicitně. Ve zpracování obrazu je nutné dodržet interval $\langle 0,1 \rangle$ nebo $\langle 0,255 \rangle$. Normalizace filtru se provádí zjištěním maximální a minimální odezvy pro maximální a minimální vstupy reprezentované skokovou funkcí a následným převodem tohoto intervalu do potřebných mezí. Tento proces má v API na starosti třída „segment.api.tools.filter.FilterCharacter“.

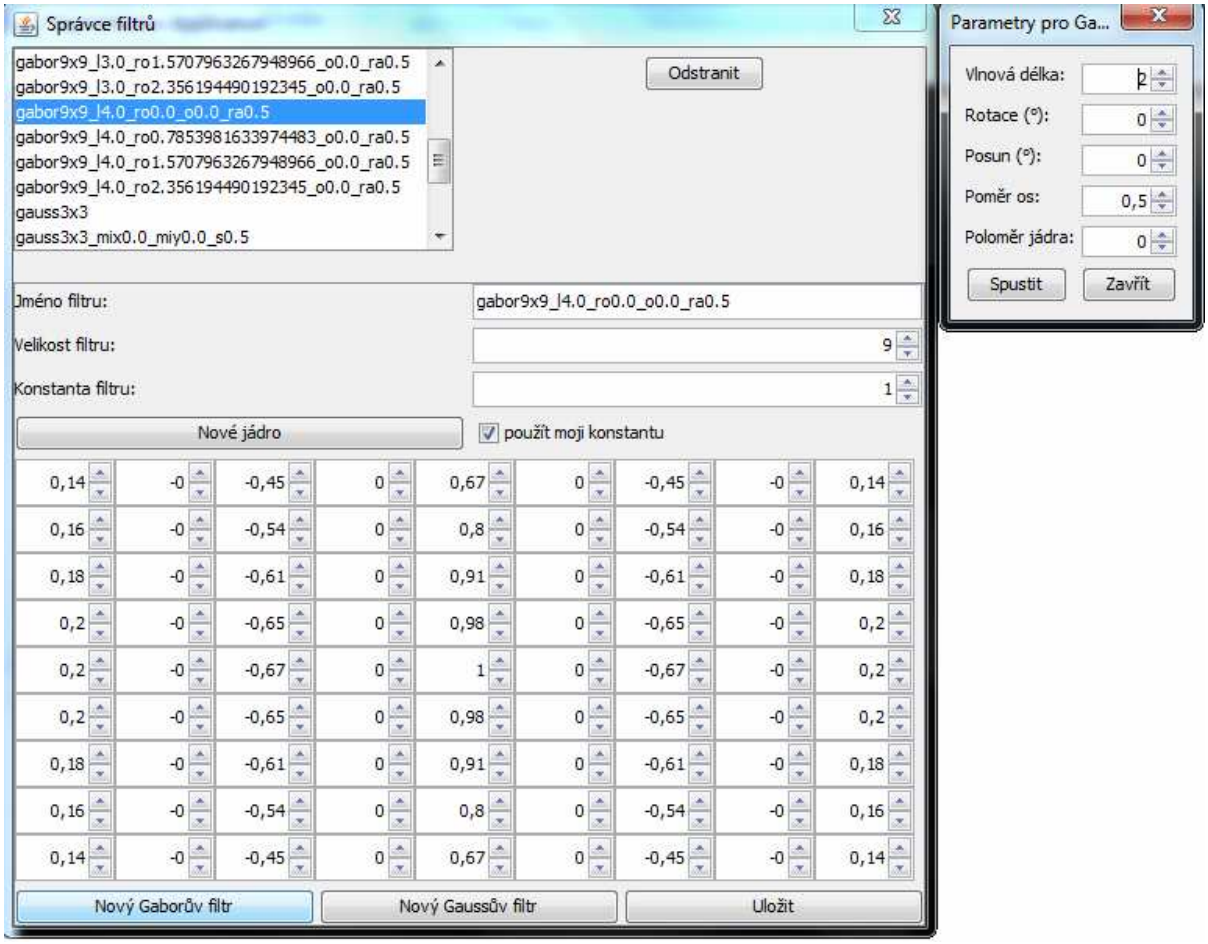


Obrázek 6-9: *Návod pro sestavení skokové funkce pro dané konvoluční jádro (1D) a požadovaný rozsah výstupních hodnot v rozmezí 0 a 1*

Generátory konvolučních filtrů umožňují navzorkování spojité funkce definované předpisem filtru. Pro účely API musí používat rozhraní „segment.api.tools.filter.KernelCreator“. V aplikaci jsou implementovány celkem dva generátory konvolučního jádra podle zadaných parametrů. Jeden je pro Gaborovy filtry a druhý pro Gaussův filtr. Jejich průběh je podrobněji popsán v kapitole operace nad obrazy. S těmito generátory spolupracuje formulář „segment.api.tools.filter.KernelCreationDialog“.

Vlastní správce filtrů obsahuje všechny aktuálně definované filtry v hashovací tabulce. Umožňuje přístup k definovanému filtru pomocí jména filtru. Definice filtru je zapouzdřená ve třídě „segment.api.tools.filter.Filter“. Správce řídí přiřazování, změnu, přidávání a odebírání filtrů. Mezivýsledky ukládá do XML souboru uloženého v místě spuštění aplikace pod názvem

„filters.xml“. Pokud není nalezen takový soubor, pak ho vytvoří. Grafický formulář opět zaštiťuje případy použití výše popsaného API.



Obrázek 6-10: *Ukázka grafických formulářů pro správu filtrů (vpravo je dialog pro vygenerování nového jádra na základě příslušných parametrů)*

6.9 Generátor množiny dat

Generátor množiny dat je stěžejní modul pro celý systém. Připravuje obrazová data pro segmentační algoritmy a může výrazně ovlivnit jejich výsledek. Extrahuje důležité charakteristiky z obrázků, a tím určuje ráz segmentace. Je nutné, aby umožnil snadno měnit jednotlivé charakteristiky a přitom zaručil určitou míru kompatibility. Odpovídá na jednu z nejzávažnějších otázek celé segmentace: „Jak zvolit trénovací data, aby se segmentace povedla?“ Má na starosti nejen extrakci příznaků, ale i prefiltraci obrazových dat, výběr vhodných pixelů a jejich pořadí v trénovací sadě.

Modul je implementován jako jediná třída „segment.api.segmentalg.MyTrainData“. Jejím vstupem jsou obrazové informace a na výstupu je trénovací množina aplikovatelná přímo na segmentační algoritmy. Trénovací množina je implementována především jako vektor vstupních polí pro segmentační algoritmy. Třída umožňuje přistupovat ke vstupům podle indexu. Pro uchování

získané trénovací množiny je použito textového souboru. Třída nabízí rozhraní pro načtení dat z textového souboru, nebo z pole obrázků. Důležitou součástí pro získání trénovacích dat ze souboru je správce filtrů, který umožňuje identifikaci filtrů na základě uloženého jména. Získání vhodného příznakového vektoru probíhá ve dvou fázích:

- prefiltrace – používá konvoluční filtry a mediánový filtr nad RGB barevným modelem,
- extrakce příznaků – používá barevný model a konvoluční filtry nad stupni šedi.

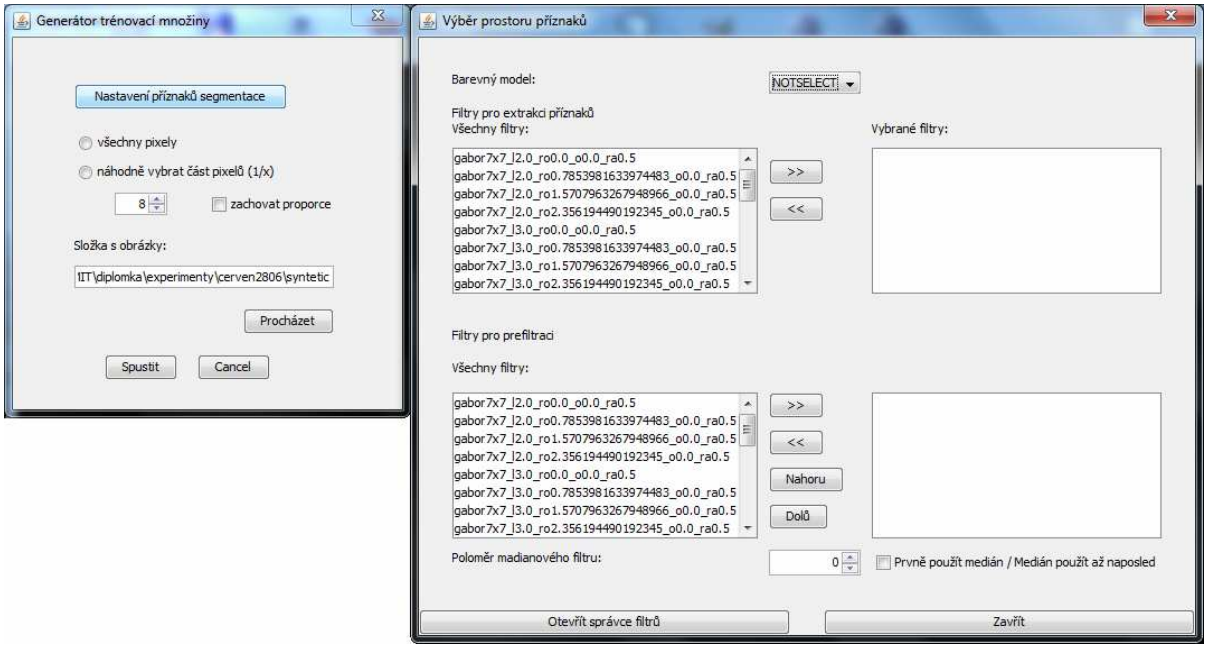
Různou kombinací těchto filtrů vznikají různé trénovací množiny. Kompatibilní trénovací sady jsou definovány podle jmen použitých filtrů a pořadí, ve kterém byly aplikovány v obou fázích.

Pro experimentální účely jsem vytvořila několik režimů výběru trénovací množiny, které by mohly mít vliv na celý průběh segmentace. Každý z režimů přistupuje k trénovací sadě obrázků na vstupu trochu odlišně:

- režim 1 (MyTrainData.DATA_ALL)
  - Zpracovává všechny pixely.
  - Pro každý pixel připraví vektor příznaků.
  - Vektory příznaků jsou uchovány v pořadí načtení.
- režim 2 (MyTrainData.DATA_RANDOM)
  - Zpracovává jen zadanou část pixelů (např. $\frac{1}{2}, \frac{1}{30}$ apod).
  - Výběr pixelů se provádí pomocí třídy „GridImage“.
  - Jen pro vybrané pixely se připraví vektor příznaků.
  - Vektory příznaků jsou uchovány v pořadí načtení.
- režim 3 (MyTrainData.DATA_SEGM)
  - Zpracovává jen zadanou část pixelů.
  - Rozdělí pixely podle náležitosti k jednotlivým objektům ve scéně podle šablony.
  - Náhodně vybere příslušnou část z každé skupiny.
  - Vypočítá pro ně vektor příznaků a náhodně mu přiřadí pořadí v trénovací množině.
- Režim 4 (metoda featureFromPosition)
  - Na základě vstupního obrázku vrátí příznakový vektor pro zadaný pixel.

Všechny režimy mají v systému své použití. V experimentální části si však slibuji dobré výsledky od režimu 3, a to z důvodů vynuceného zachovávání proporcí jednotlivých textur a předcházení tomu, aby se síť adaptovala delší čas pouze na jeden objekt. Tato situace může nastat především v režimu 1. Vektory jsou zpracovány sekvenčně a je velká pravděpodobnost, že dva nebo více trénovacích vektorů po sobě náleží stejnému segmentu. Režim 2 má nejmenší časovou náročnost, ale není zde implicitně garantované zachování proporcí. Režim 4 se uplatní v segmentační části algoritmu.

Modul je z grafického rozhraní přístupný přes třídu „segment.GenFeatureDialog.java“ a výběr příznaků, nebo případné prefiltrace je implementován přes „segment.FeatureManagerDialog“.

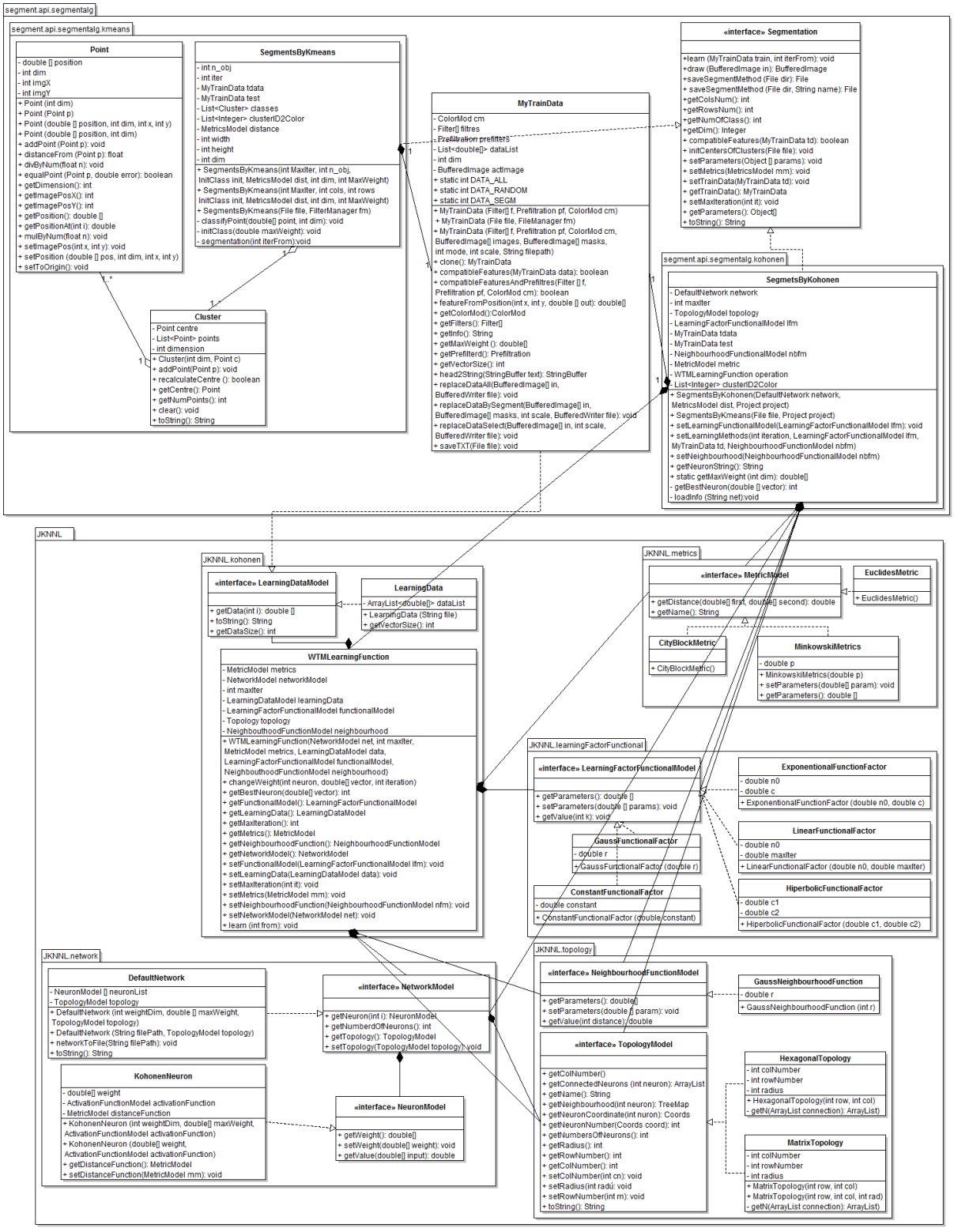


Obrázek 6-11: *Ukázka grafického rozhraní pro přípravu trénovací množiny (vpravo lze vybírat příslušný prostor příznaků a prefiltrace vstupních obrazů)*

6.10 Segmentační algoritmus

Celý segmentační systém se odvíjí od segmentačních algoritmů. Lze je označit za jádro celého systému. Dle návrhu je třeba k nim zařídit jednotný přístup, což zaručuje použití rozhraní „segment.api.segmentalg.Segmentation“ u tříd, které je implementují. Rozhraní podporuje funkce pro učení, nastavení parametrů, inicializaci středů ze souboru a segmentaci obrázků podle aktuálního rozestavení středů shluků.

Vlastní implementace algoritmů dodržuje principy zmíněné výše. Pro Kohonenovy mapy je použita knihovna JKNNL. Topologie sítě a učící algoritmus definované v JKNNL jsou zapouzdřeny do třídy „segment.api.segmentalg.kohonen.SegmentsByKohonen“. Algoritmus k-means jsem implementovala sama podle kapitoly 4.3. Potřebné třídy jsou k nalezení v balíčku „segment.api.segmentalg.kmeans“.



Obrázek 6-12: *Třídní diagram standardu UML pro zachycení vztahů mezi implementovanými objekty a objekty obsaženými v knihovně Java Kohonen Neural Network Library.*

Definice algoritmů se v některých případech vztahuje pouze na obecné označení typu použité funkce a pro její implementaci je možné využít nejrůznější předpisy. Jedná se o následující typy:

- hodnotící funkce

- jsou implementovány knihovnou JKNNL,
- představují výpočet vzdálenosti dvou bodů v prostoru,
- jsou implementována pomocí:
  - Euklidovy vzdálenosti (pro interní potřebu označena 0)

$$d(\vec{x},\vec{w}) = \sqrt{\sum_{i=0}^{n}(x_i - w_i)^2}; |\vec{x}| = |\vec{w}| = n \qquad 6.10.1$$

  - Vzdálenosti CityBlock (pro interní potřebu označena 1)

$$d(\vec{x},\vec{w}) = \sum_{i=0}^{n}|x_i - w_i|; |\vec{x}| = |\vec{w}| = n \qquad 6.10.2$$

  - Vzdálenosti Minkowski (pro interní potřebu označena 2)

$$d_p(\vec{x},\vec{w}) = \sqrt[p]{\sum_{i=0}^{n}|x_i - w_i|^p}; |\vec{x}| = |\vec{w}| = n, p \in R_0^+ \qquad 6.10.3$$

- jsou společné pro Kohenovy mapy i pro k-means.

- učící funkce
  - jsou implementovány knihovnou JKNNL,
  - představují učící faktor v závislosti na iteraci učícího algoritmu,
  - jsou implementována pomocí:
    - Konstantní funkce (pro interní potřebu označena 0)

$$\mu_c(k) = c; c \in \langle 0,1 \rangle \qquad 6.10.4$$

    - Exponenciální funkce (pro interní potřebu označena 1)

$$\mu_{n,c}(k) = n \cdot e^{-c \cdot k}; n \in \langle 0,1 \rangle, c \in R^+ \qquad 6.10.5$$

    - Gaussovy funkce (pro interní potřebu označena 2)

$$\mu_r(k) = e^{-\frac{k^2}{2 \cdot r^2}}; r \in R^+ \qquad 6.10.6$$

    - Hyperbolické funkce (pro interní potřebu označena 3)

$$\mu_{a,b}(k) = \frac{a}{b+k}; a,b \in R^+ \qquad 6.10.7$$

    - Lineární funkce (pro interní potřebu označena 4)

$$\mu_{a,b}(k) = \frac{a}{b+k}; a,b \in R^+ \qquad 6.10.8$$

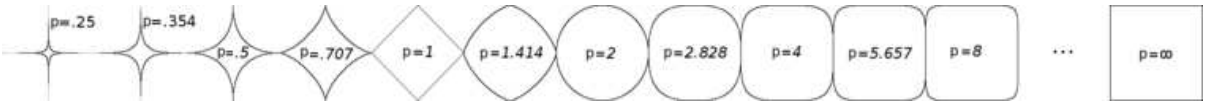
  - mají význam jen ve spojitosti s neuronovou sítí.

- funkce sousednosti
  - je implementována pomocí JKNNL,
  - představuje učící faktor pro sousedy vítězného neuronu,
  - je implementována pomocí Gaussovy funkce:

$$\mu_r(d) = e^{-\frac{d^2}{2 \cdot r^2}}; r \in R^+ \qquad 6.10.9$$

- má význam jen ve spojitosti s neuronovou sítí.

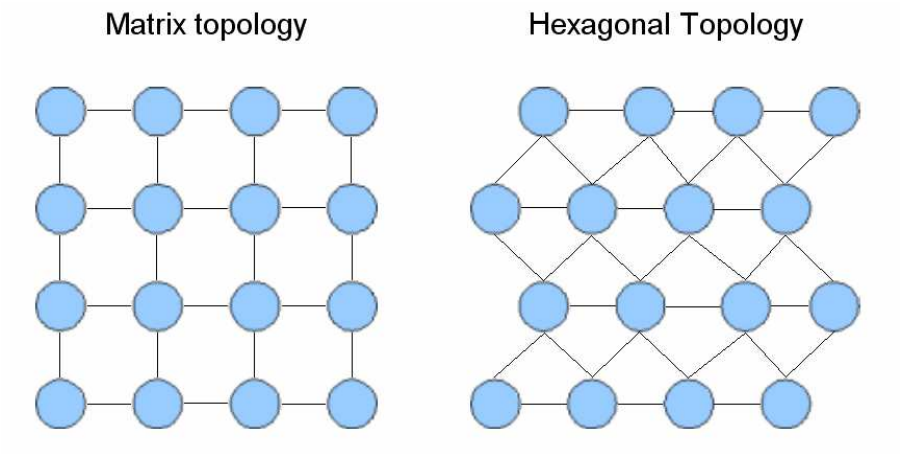
Třídy implementující výše popsané funkce využívají rozhraní umožňující zadávat parametry pomocí pole nebo v konstruktoru a metodu pro získání hodnoty na základě vstupní hodnoty.



Obrázek 6-13: *Závislost jednotkového prostoru pro výpočet vzdálenosti pomoci metriky Minkowski na parametru* p

Ze schématu vyplývá, že pokud je $p = 1$, pak je metrika totožná s CityBlock metrikou a pokud je $p = 2$, pak je metrika podobná Euklidově. Hodnota parametru implicitně použita v aplikaci je $p = 1$ [18].

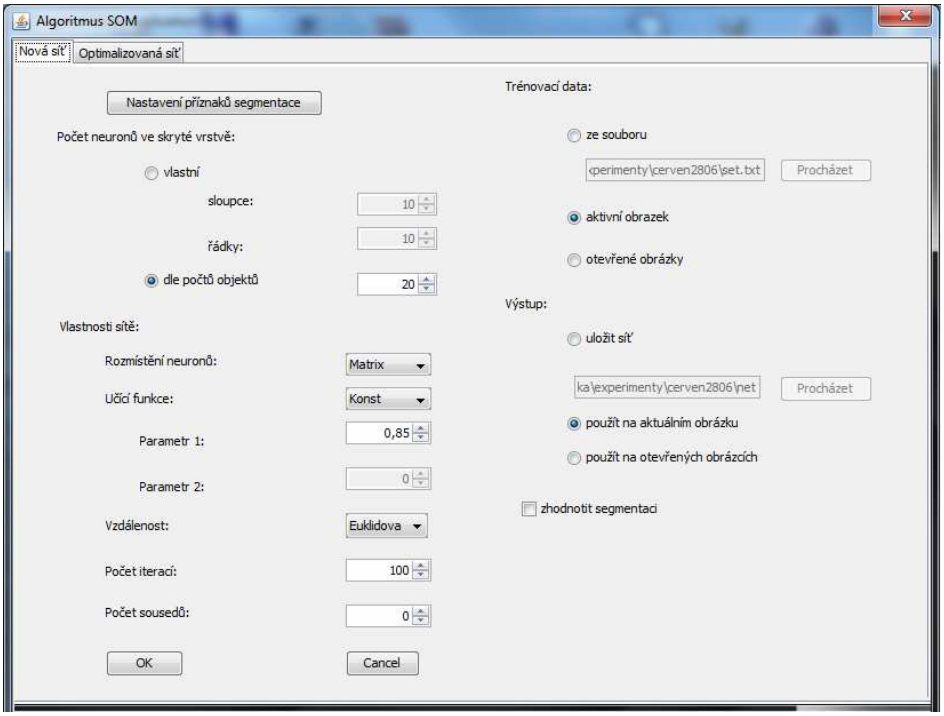
U Kohonenovy mapy má význam ještě uspořádání neuronů pro výpočet funkce sousednosti a určení vzdálenosti neuronu od vítězného neuronu. V aplikaci rozeznávám celkem 2 typy propojení neuronů. Prvním typem je pravoúhlá mřížka, u které je počet přímých sousedů roven 4. Druhým typem je hexagonální mřížka, u které je počet přímých sousedů roven 6. Oba typy jsou implementovány pomocí knihovny JKNNL. Pro použití příslušných tříd došlo k zásahu do kódu knihovny a pro správnou funkčnost je vyžadována upravená verze, která je přiložena k aplikaci.



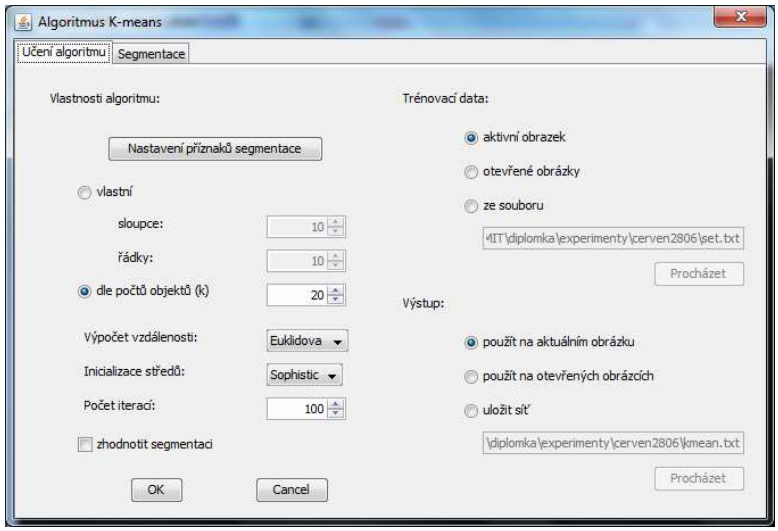
Obrázek 6-14: *Schéma implementovaných topologií Kohonenovy mapy pro rozměry 4x4 [16]*

Přístup k segmentačním algoritmům je zajištěn přes několik typů formulářů, které odlišně pracují s výše popsanými moduly. Předpokládám, že nejvíce používané zapojení systému je implementované pomocí tříd „segment.Kmean“ a „segment.Kohonen“. Je určen pro aplikaci naučených segmentátorů na jednotlivá data a pro optimalizaci algoritmů podle odzkoušených parametrů. Implementace se skládá pouze ze seskládání potřebných modulů podle vyplněného

formuláře. Pokud dojde k výběru trénovacích dat z obrázků, které jsou otevřeny v aplikaci, pak je použit režim 2 a trénovací sada obsahuje 25 % vektorů z celkového počtu pixelů.



Obrázek 6-15: *Ukázka grafického formuláře pro optimalizaci Kohonenovy mapy.*



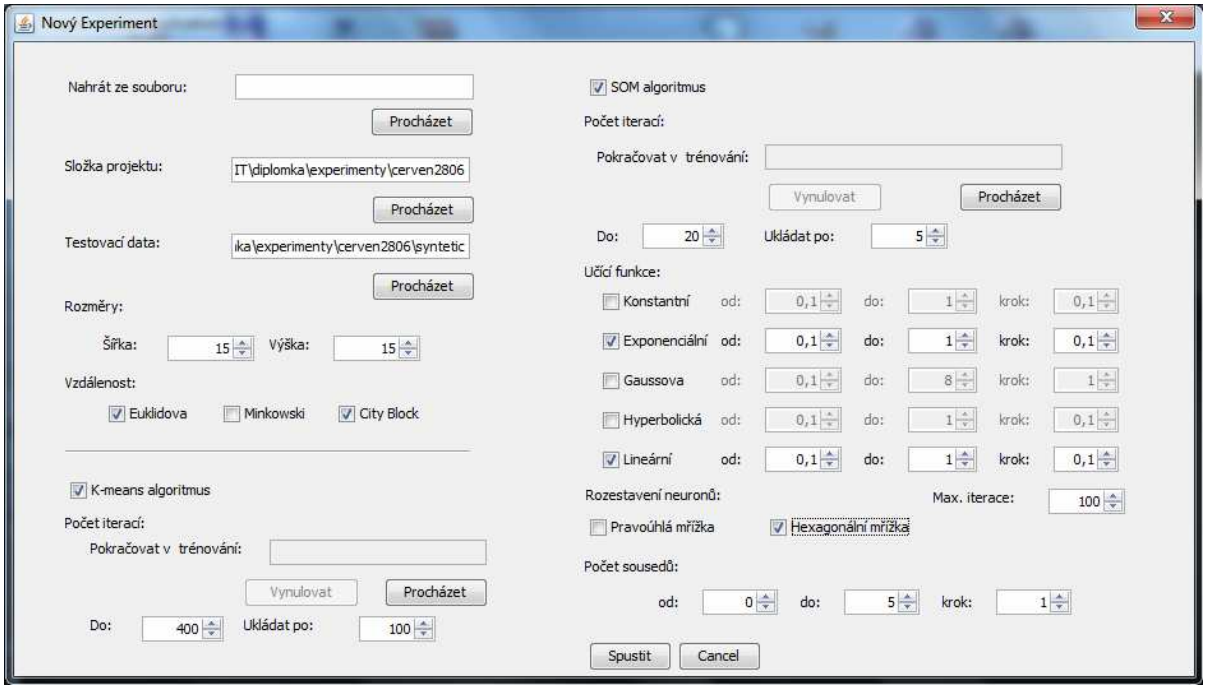
Obrázek 6-16: *Ukázka grafického formuláře pro optimalizaci k-means algoritmu*

Druhý přístup je určen pro experimentální účely diplomové práce, kdy hledám optimální nastavení a mapuji chování obou algoritmů. Formulář „segment.ExperimentDialog“ provádí zapojení jednotlivých modulů podle zadaných parametrů a stará se o dodržení integrity dat, aby výsledné statistické hodnoty mohly být porovnatelné. Integrita dat je zaručena v případě:

- Inicializace středů shluků, kdy algoritmus změní pouze pozici středů shluků.

- Výstupu do souborového systému, uloží vlastnosti algoritmu bez změny vnitřní reprezentace a následně pokračuje v učení.
- Záznamu pořizovaného po vykonání určité sumy iterací nazvaných krokem.
- Procházení stavového prostoru, kdy objekty pro nastavení algoritmů jsou uloženy v polích a pomocí výčtového typu je jednoznačně určena poloha objektu. Pokud dané nastavení není použito, je to reprezentováno ukazatelem „null“ a přejde se k dalšímu nastavení. Pole nastavení jsou procházena sekvenčně pomocí „for cyklů“.
- Kontroly kompatibility při inicializaci počátečního rozložení středů shluků ze souboru, kdy je kontrolován rozměr shluků a sémantika příznakových vektorů. Uživatel může tuto kontrolu obejít záměnou informací v textovém souboru, ale to je již ponecháno na jeho úsudku.

Cílovým souborem je pouze záznam o vyhodnocení segmentací. Ostatní soubory jsou jen podpůrné pro kontrolu průběhu experimentů. Pro opakování stejných experimentů, nebo případné úpravy jen části parametrů, je do složky projektu vygenerován soubor ve formátu XML. Ten obsahuje nastavení formuláře, které je možné znova nahrát.



Obrázek 6-17: *Ukázka grafického formuláře pro zadání nového experimentu (ve složce projektu se objeví nové složky kmeans a SOM, ve kterých budou uloženy mezivýsledky pro zvolené parametry)*

6.11 Hodnocení segmentace

Pro úspěšné zhodnocení segmentačních algoritmů je potřeba spolehlivé metodiky na zhodnocení výsledků. Příslušná metodika byla popsána v předchozích kapitolách. Vstupem modulu jsou výstupy

ze segmentačních algoritmů společně se jménem segmentovaného obrázku. Modul provede identifikaci obrázku a vygeneruje si identifikátor šablony, kterou se pokusí načíst. Pokud dojde k selhání, pak model skončí.

Jednotlivé metodiky musí používat rozhraní „segment.api.metrics.Metrics“. Nyní je implementována pouze jedna metodika. Pokud by bylo vytvořeno více metrik, doporučují jejich reprezentaci pomocí výčtového typu. Metriky musí být jednoznačně identifikovány pomocí řetězce. Lze dodatečně nastavit vstupní obrazy a přepočítat metriku. Dále rozhraní umožňuje získat podrobnou informaci o stavu metriky v řetězci a redukovat vše na jedno číslo. Dostupná implementace je zajištěna třídou „segment.api.metrics.StatisticOfMap“, která využívá hashovací tabulku barev odpovídajících pixelů. Hashovacím klíčem je barva v obrázku uloženém v proměnné „from“ a hodnotou je seznam barev odpovídajících pixelů v obrázku, který je uložený v proměnné „to“.

Základním problémem pro implementaci metriky je určení pixelů, které patří do množiny true positive. Z hashovací tabulky sice známe odpovídající si barvy, ale v další fázi je třeba přiřadit náležitost pixelů do množin „true positive“ a „false positive“. Z hlediska dostupných informací se jedná především o statistické přidělení. Nemáme informaci o tom, která barva má být originální. V případě klasifikačního algoritmu by tento problém odpadl.

Předpokládejme, že máme úspěšný algoritmus. Pak by platilo, že každý klíč v hashovací tabulce by obsahoval list o velikosti 1. V případě existence nenulové množiny „false positive“ dojde k nárůstu velikosti jednotlivých listů. **Kde se však nachází hranice mezi „false positive“ a „true positive“?** Ze statistických údajů vyplývá, že správně odpovídající si barvy daného objektu mají maximální četnost v namapování obrázků z pohledu „from – to“ i „to – from“.

V daném algoritmu počítám četnost množiny „true positive“. Na začátku algoritmu je rovna nule. V cyklech procházím hashovací tabulku, abych našla maximální prvek, který je zaznamenám v proměnné. Po projití celé hashovací tabulky uvolním list náležející barvě „from“ zaznamenaného prvku a smažu prvek se zaznamenanou barvou „to“ z ostatních listů. Počet pixelů zaznamenaný v maximálním prvku přičtu k množině „true positive“. Celý postup opakuji, dokud hashovací tabulka obsahuje nějaké prvky.

7 Experimentální část

Účelem experimentální části je otestovat navržený systém na syntetických i reálných datech, porovnat obě implementované segmentační metody (k-means a Kohonenovu mapu) a zjistit jejich nedostatky či výhody. Jak již bylo několikrát řečeno, diplomová práce se zabývá obecnou segmentací. Pro ni je velmi těžké stanovit konkrétní výsledky, protože může být vykonávána nad různou množinou vstupů. Proto jsem experimentální část vedla spíše směrem, který by umožnil zmapovat chování učícího algoritmu pro Kohonenovy mapy.

Při procházení studijní literatury jsem nenašla shrnutí základních rysů Kohonenovy mapy. Zvláště při prvním spuštění rozsáhlejších experimentů, jsem se potýkala s nevhodným nastavením vstupních parametrů. Samozřejmě vše se odvíjí od charakteristiky obrázků, na kterých chceme algoritmus použít, ale přesto se Kohonenova mapa řídí určitými pravidly, které je vhodné znát při navrhování konkrétnějších segmentačních systémů, např.:

- co určuje rychlost konvergence sítě,
- jakým způsobem dochází k rozdělení prostoru příznaků,
- jak se síť chová v návaznosti na počet neuronů,
- a další.

Jednotlivé experimenty byly voleny tak, aby dokázaly odpovědět na uvedené otázky.

V kapitole bude podán záznam o průběhu jednotlivých experimentů a jejich výsledků. Každý experiment bude popsán v samostatné podkapitole, která bude mít následující strukturu:

- Motivace – proč jsem zařadila tento experiment.
- Cíle – co od experimentu očekávám.
- Průběh experimentu – popis metodiky experimentu.
- Výsledky – přehled nejdůležitějších výsledků.
- Závěr – naplnění nebo vyvrácení cílů experimentu.

7.1 Experiment 1 – zmapování vlastností implementované Kohonenovy mapy I

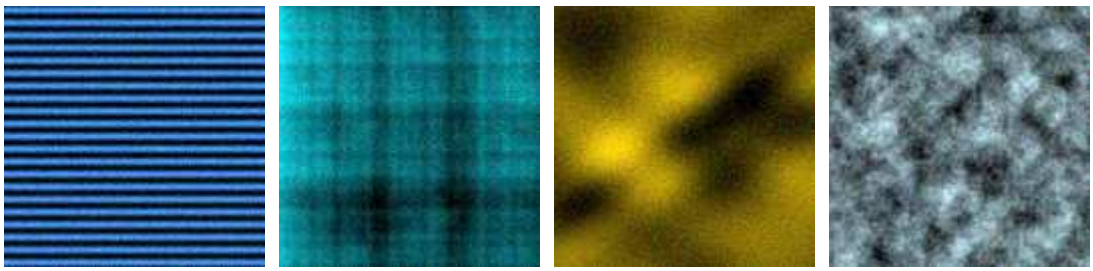
Motivací k experimentu byla různá nastavení, která nabízí zvolená knihovna pro práci s Kohonenovou mapou (Java Kohonen Neural Network Library). Rozhodla jsem se vyzkoušet většinu z dostupných možností, abych zvolila nejvhodnější nastavení pro účely segmentace.

Cílem je srovnat jednotlivá nastavení Kohonenovy mapy s výsledky dosaženými pomocí algoritmu k-means. Experiment by měl poskytnout první náhled na problematiku segmentace pomocí zvoleného algoritmu.

Pro účely experimentu jsem vygenerovala 50 textur pomocí Generátoru textury. Z nich jsem vybrala 15 textur tak, aby byly od sebe relativně dobře oddělitelné a rovnoměrně zastupovaly různé barevné a texturní příznaky. Na zvolené textury jsem nanesla Gaussův šum s rozptylem 12 a vygenerovala množinu 50 obrázků o velikosti 120x120 pixelů s maximálně čtyřmi objekty ve scéně. Z nich jsem vytvořila následující množiny:

- trénovací množina 23 obrázků,
- testovací množina 19 obrázků,
- trénovací množiny vektorů příznaků, které:
    - jsou vygenerovány z trénovací množiny obrázků,
    - jsou voleny proporcionálně s redukcí na 1/25 (zhruba 13195 vektorů),
    - obsahují texturní příznaky pomocí Gaborova filtru s velikostí konvolučního jádra 7x7 pixelů a vlnovou délkou 2, 6 a 12, kdy pro každou vlnovou délku bylo zvoleno natočení 0°, 45°, 90°, 135°,
    - obsahují následující příznaky:
        - HS – složku hue a saturation z HSV modelu,
        - RGB – složky red, green a blue z RGB modelu,
        - HS + textura (popsaná výše),
        - RGB + textura (popsaná výše),
        - textura (popsaná výše).

Odezva Kohonenovy mapy na testovací sadu obrázků bude porovnávána v návaznosti na volbu funkce určující učící faktor, charakteristiky okolí vítězného neuronu a počtu iterací. Záznam o zvolených parametrech, optimalizované Kohonenovy mapy pro jednotlivá nastavení a výsledky segmentace jsou uloženy na přiloženém DVD médiu. Maximální iterace pro SOM síť byla stanovena na 50 se záznamem výsledků po 10 iteracích. Zkoumán byl vliv učících funkcí, jejichž parametry byly voleny od 0,5 do 1 se záznamem po 0,1, a vliv velikosti okolí, které bylo voleno následovně 0, 1, 2, 3. Oba algoritmy rozdělují vstup do 7x7 shluků.



Obrázek 7-1: *Ukázka textur použitých v syntetických obrazech*

Následující tabulky a grafy uvádí výběr nejzajímavějších dat z experimentu, protože zde není prostor pro uvedení všech dat. Úspěšnost nad testovací sadou v daném okamžiku učení je udávána

jako průměrná a střední (medián) úspěšnost z jednotlivých obrazů. Pro každou funkci definující učící faktor určuji maximální dosaženou úspěšnost nad testovací sadou.

Tabulka 7-1: *Shrnutí úspěšnosti algoritmu k-means nad danou testovací sadou za použití různých vztahů pro výpočet vzdálenosti*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	34,89	42,76	56,29	60,95	50,18
	CityBlock	33,24	37,85	57,46	55,18	53,98
Medián	Euclides	30,05	41,28	48,33	50,36	36,94
	CityBlock	30,96	38,33	47,91	46,27	47,82
Naučen v iteraci		do 125	do 250	do 125	do 125	do 125

Tabulka 7-2: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při konstantním učícím faktoru a nulovém počtu sousedů vítězného neuronu*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	36,11	57,01	59,59	60,59	62,59
	CityBlock	35,61	57,06	56,76	53,83	63,07
Medián	Euclides	35,76	63,32	57,66	65,25	66,81
	CityBlock	33,14	63,14	49,71	59,56	69,45
Maximum		10 iter pro 0,9	10 iter pro 1,0	10 iter pro 1,0	10 iter pro 0,9	10 iter 0,6

Tabulka 7-3: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při konstantním učícím faktoru a počet sousedů vítězného neuronu je roven třem*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	38,39	70,08	34,66	56,12	35,72
	CityBlock	35,16	70,28	33,13	53,31	34,52
Medián	Euclides	39,49	81,92	30,58	57,10	36,19
	CityBlock	34,38	81,37	32,76	57,53	35,38
Maximum		10 iter pro 0,9	10 iter pro 0,9	10 iter pro 0,8	10 iter pro 0,9	10 iter 0,9

Tabulka 7-4: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při exponenciálním učícím faktoru a nulovém počtu sousedů vítězného neuronu*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	36,66	39,92	56,10	60,10	51,59
	CityBlock	36,42	40,13	54,62	58,87	51,30
Medián	Euclides	37,88	41,26	42,83	53,95	45,44
	CityBlock	36,87	41,53	42,42	50,96	45,81
Maximum		10 iter 0,9	10 iter pro 0,8	10 iter pro 0,7	30 iter pro 1,0	10 iter 0,9

Tabulka 7-5: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při exponenciálním učícím faktoru a počet sousedů vítězného je roven třem*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	24,88	41,32	24,52	29,44	19,14
	CityBlock	25,08	41,05	24,00	27,62	19,07
Medián	Euclides	27,03	45,01	25,74	26,59	17,22
	CityBlock	24,85	44,35	23,64	24,59	17,47
Maximum		10 iter pro 0,6	20 iter pro 0,9	20 iter pro 0,5	20 iter pro 0,7	20 iter pro 0,6

Tabulka 7-6: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro Gaussův učící faktor a nulovém počtu sousedů vítězného neuron*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	38,76	58,76	57,30	62,54	64,55
	CityBlock	38,18	59,13	54,11	59,03	64,86
Medián	Euclides	37,78	67,22	54,18	63,35	68,24
	CityBlock	39,56	68,06	49,58	61,35	69,84
Maximum		50 iter pro 45	10 iter 45	50 iter pro 30	10 iter pro 30	40 iter pro 35

Tabulka 7-7: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro Gaussův učící faktor a počet sousedů vítězného je roven třem*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	40,90	69,52	36,99	62,17	34,58
	CityBlock	36,34	69,65	32,26	64,09	33,21
Medián	Euclides	41,09	81,60	38,94	73,31	34,33
	CityBlock	39,80	81,47	32,95	69,27	34,96
Maximum		20 iter pro 35	30 iter pro 30	40 iter pro 45	10 iter pro 40	20 iter pro 45

Určení učícího faktoru podle Gaussovy funkce má velice stabilní výsledky. V jednotlivých kategoriích je úspěšnější než ostatní funkce. Při použití Euklidovy vzdálenosti je vždy lepší než algoritmus k-means. Bohužel jsem v daném testu zjistila, že nemá stabilní maximum při určitém nastavení, což může být způsobeno velkým krokem v počtu iterací.

Učící faktor daný exponenciální funkcí je v sérii testů nejhorší ve většině případů. Pro nulový počet sousedů je srovnatelný s k-means algoritmem, ale při větším okolí rapidně ztrácí účinnost. V tomto ohledu jsou si s hyberbolickou funkcí podobné. To může být způsobeno nevhodnou volbou druhého parametru, který ovlivňuje rychlost klesání funkce.

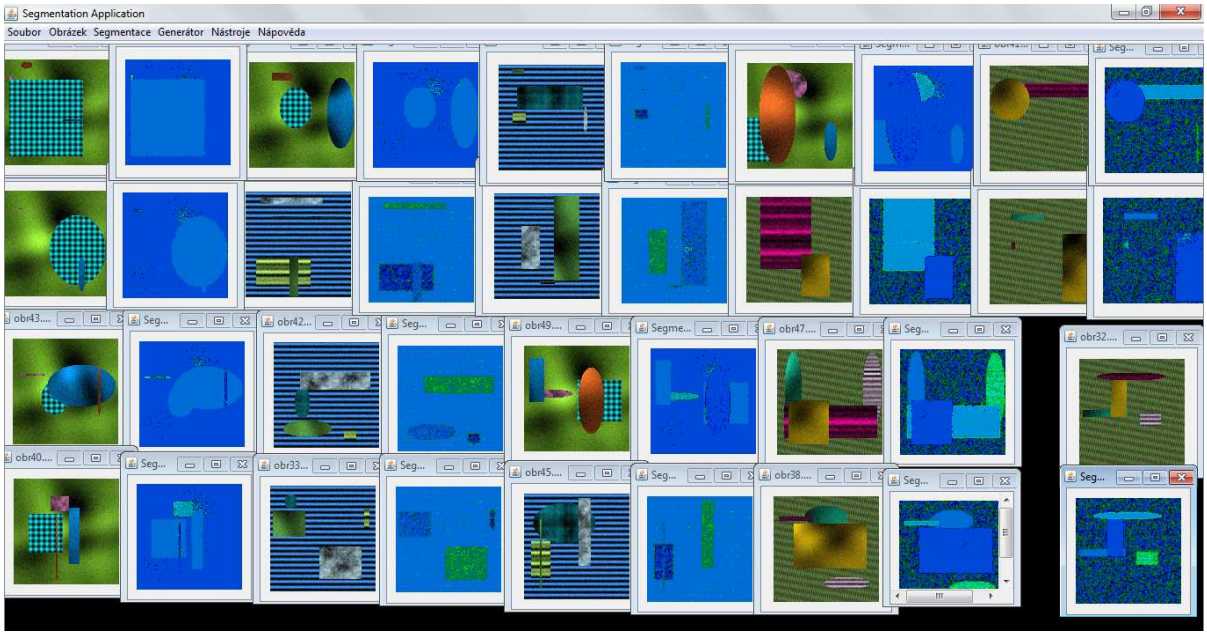
Ostatní učící faktory jsou srovnatelné a často lepší než k-means algoritmus. Pouze pro příznaky RGB s texturou vyniká k-means algoritmus.

Tento test odhalil „bug“ v Java Kohonen Neural Network Library, který byl následně odladěn. Avšak z důvodů velké výpočetní náročnosti už nebyl experiment opakován. „Bug“ se týkal změny topologie sítě po jejím vytvoření. To se v experimentu projevilo neplatnými údaji pro hexagonální topologii, které byly shodné s pravoúhlou mřížkou.

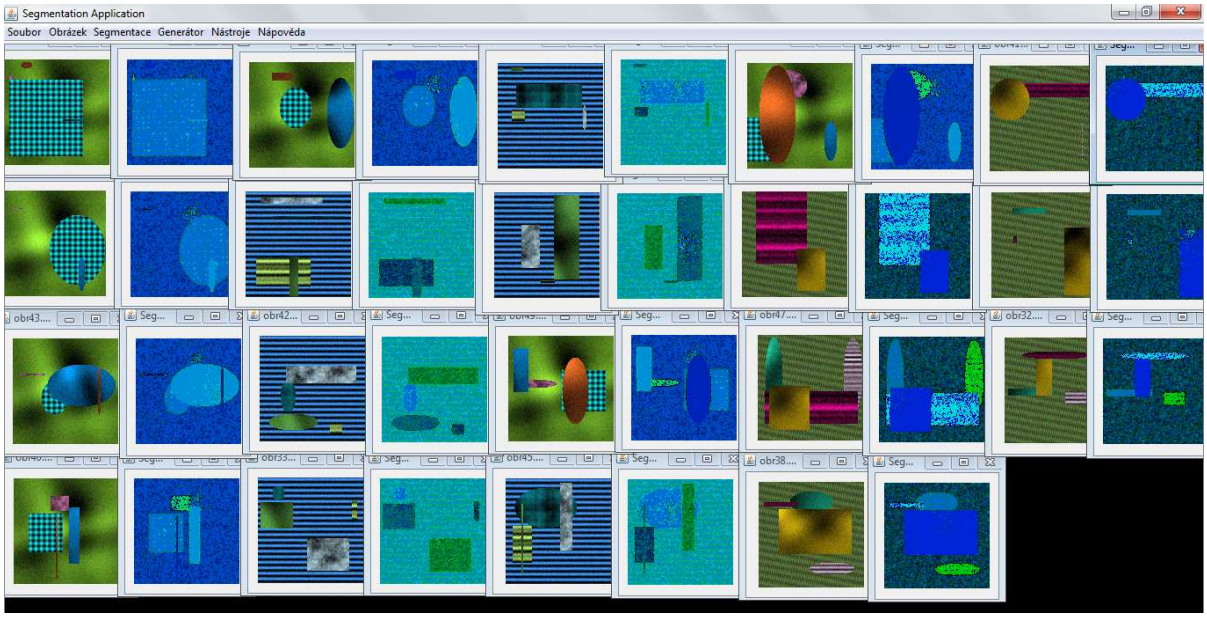
Z testů vyplývá, že Kohonenovy sítě mají maximální účinnost okolo desáté iterace a občas se projeví i periodičnost. Většina učících algoritmů dosáhne maxima pro parametr učícího faktoru okolo 0,9. To je nejspíše ideální poměr pro zachování obecnosti. Vítězný neuron zcela nesplyne se vstupem, jen se dostatečně posune jeho směrem.

Na přiloženém DVD jsou uvedeny průběhy jednotlivých funkcí. Experiment prokázal výhodu pro funkce, které v prvních iteracích mají vysoký učící faktor a jeho změna je minimální.

Kohonenova mapa dosahuje lepších průměrných hodnot na zvolené testovací sadě při Euklidově bázové funkci, oproti tomu CityBlock bázová funkce dosahuje srovnatelných nebo lepších výsledků pro mediánové hodnoty.



Obrázek 7-2: *Srovnání testovací sady a její odezvy pro optimalizovanou SOM síť (konstantní učící faktor s parametrem 1,0, HS příznaky, v 10 iteraci, pravoúhlá mřížka, okolí velikosti 3, Euklidova vzdálenost)*



Obrázek 7-3: *Srovnání testovací sady a její odezvy pro optimalizovaný k-means algoritmus (HS příznaky, Euklidova vzdálenost)*

Z Obrázek 7-2 a Obrázek 7-3 vyplývá, že konstantní funkce získala vysoké hodnocení, protože je odolná vůči Gaussovu šumu. Bohužel však ztrácí rozlišitelnost jednotlivých objektů, které mají obdobnou barvu. K-means opravdu odlišil jednotlivé objekty, ale bohužel se na výsledku projevil vliv Gaussova šumu. Těžko říci, která varianta je vhodnější. Rozhodnutí záleží na dalším zpracování.

7.2 Experiment 2 - zmapování vlastností implementované Kohonenovy mapy II

Motivací k tomuto experimentu jsou výsledky z předchozích testů, kdy sice Kohonenova mapa byla dle zvolené metriky povětšinou lepší než k-means, ale maximální úspěšnost byla docela nízká. Proto jsem zkusila zvolit jinou texturní charakteristiku výběrem dalších Gaborových filtrů.

Cílem je zmapovat vlastnosti Kohonenovy mapy se zaměřením na jinou testovací a trénovací sadu obrázků a využít poznatků z prvního experimentu.

Postup je obdobný jako u prvního experimentu. Bylo vygenerováno 50 syntetických obrázků ze 17 textur o rozměrech 100x100 pixelů a Gaussovým šumem s rozptylem 8. Pro vytvoření trénovací množiny vektorů prostoru příznaků bylo vybráno 30 obrázků. Testovací množina se opět skládá z 19 obrázků. Počet shluků se rovná 7x7.

Krok pro parametry učícího faktoru je 0,05 pro jemnější zachycení rozdílů. Pro lepší představu o průběhu učícího algoritmu jsem volila záznam po 2 iteracích u Kohonenovy mapy, ale tentokrát s maximální iterací rovnou 34. V předchozím experimentu se osvědčily dvě velikosti pro okolí vítězného neuronu, a to segmentace bez okolí nebo s maximálním zapojením celé mřížky, což pro síť 7x7 je velikost odpovídající třem.U k-means bylo nastaveni trochu odlišné. Maximální iterace je rovna 120 s krokem po 10 iteracích.

Tabulka 7-8: *Shrnutí úspěšnosti algoritmu k-means nad danou testovací sadou za použití různých vztahů pro výpočet vzdálenosti*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	**Euclides**	60,45	68,79	74,74	90,24	66,92
	CityBlock	57,29	60,40	65,18	80,83	63,41
Medián	**Euclides**	56,95	77,53	73,98	89,75	74,66
	CityBlock	55,27	52,80	57,20	87,50	69,46
Naučen v iteraci		do 10	do 50	do 20	do 30	do 30

Tabulka 7-9: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při konstantním učícím faktoru a nulovém počtu sousedů vítězného neuronu*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	**Euclides**	52,73	76,39	83,05	89,82	70,98
	CityBlock	53,21	77,04	67,50	86,88	74,84
Medián	**Euclides**	51,35	81,07	88,16	92,72	72,09
	CityBlock	50,51	80,02	61,91	86,91	75,30
Maximum		2 iter pro 0,95	2 iter pro 0,95	2 iter pro 0,9	2 iter pro 0,85	2 iter pro 0,95

Tabulka 7-10: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce při konstantním učícím faktoru a počet sousedů vítězného neuronu je roven třem*

		Příznak	RGB	HS	RGB, textura	HS, textura	textura
Matrix	Průměr	Euclides	48,09	69,64	51,90	67,11	35,71
		CityBlock	49,01	70,00	46,72	68,13	33,57
	Medián	Euclides	45,31	77,49	47,85	71,19	32,42
		CityBlock	45,48	77,70	42,72	70,92	32,51
	Maximum		8 iter pro 0,9	2 iter pro 0,95	10 iter pro 0,8	4 iter pro 0,8	16 iter pro 0,8
Hexagonál	Průměr	Euclides	51,42	73,80	51,41	69,43	37,03
		CityBlock	43,81	72,98	40,98	69,78	33,91
	Medián	Euclides	50,65	81,37	53,51	82,55	35,27
		CityBlock	42,35	82,32	38,27	77,46	32,16
	Maximum		2 iter pro 0,85	2 iter pro 0,85	6 iter pro 0,85	4 iter pro 0,8	6 iter pro 0,95

Tabulka 7-11: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro Gaussův učící faktor a nulovém počtu sousedů vítězného neuronu*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	52,92	73,43	84,26	92,13	75,14
	CityBlock	57,36	75,08	78,74	86,87	74,98
Medián	Euclides	51,59	79,33	87,76	95,57	75,49
	CityBlock	51,41	79,81	84,58	89,60	75,30
Maximum		32 iter pro 29	20 iter pro 29	34 iter pro 26	26 iter pro 29	28 iter pro 35

Tabulka 7-12: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro Gaussův učící faktor a počet sousedů vítězného neuronu je roven třem.*

		Příznak	RGB	HS	RGB, textura	HS, textura	textura
Matrix	Průměr	Euclides	55,42	75,09	53,82	74,26	37,10
		CityBlock	54,10	74,03	43,87	70,97	33,31
	Medián	Euclides	52,69	84,68	51,92	82,96	34,37
		CityBlock	51,32	80,87	41,14	77,29	32,19
	Maximum		2 iter pro 32	2 iter pro 26	8 iter pro 32	26 iter pro 29	8 iter pro 29
Hexagonál	Průměr	Euclides	55,35	75,81	52,38	74,24	35,39
		CityBlock	52,58	73,25	48,39	75,64	33,28
	Medián	Euclides	56,70	90,06	50,02	79,49	33,96
		CityBlock	49,20	82,55	42,53	86,32	31,68
	Maximum		8 iter pro 41	2 iter pro 35	12 iter pro 26	14 iter pro 41	2 iterpro 32

Tabulka 7-13: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro hyperbolický učící faktor o nulovém počtu sousedů vítězného neuronu*

	Příznak	RGB	HS	RGB, textura	HS, textura	textura
Průměr	Euclides	63,44	69,68	82,05	88,43	75,26
	CityBlock	62,39	73,26	81,47	87,92	70,53
Medián	Euclides	56,73	76,06	86,82	91,36	76,03
	CityBlock	58,40	79,27	89,37	90,80	72,15
Maximum		8 iter pro 0,7	24 iter 0,9	2 iter pro 0,75	34 iter pro 0,65	18 iter pro 0,85

Tabulka 7-14: *Shrnutí maximální úspěšnosti SOM sítě pro různé bázové funkce pro hyperbolický učící faktor a počet sousedů vítězného neuronu je roven třem*

		Příznak	RGB	HS	RGB, textura	HS, textura	textura
Matrix	Průměr	Euclides	57,41	74,65	57,14	72,59	34,42
		CityBlock	57,50	70,12	56,49	66,71	32,38
	Medián	Euclides	63,03	84,13	57,86	82,21	31,95
		CityBlock	54,10	80,47	54,70	77,45	29,63
	Maximum		6 iter pro 0,8	4 iter pro 0,65	16 iter pro 0,75	2 iter pro 0,85	14 iter pro 0,8
Hexagonál	Průměr	Euclides	60,94	75,65	63,32	72,89	34,87
		CityBlock	62,18	74,04	60,47	70,85	31,89
	Medián	Euclides	62,99	80,66	67,84	80,14	32,69
		CityBlock	64,16	85,16	61,52	85,58	29,59
	Maximum		18 iter pro 0,9	4 iter pro 0,65	6 iter pro 0,65	10 iter pto 0,65	8 iter pro 0,65

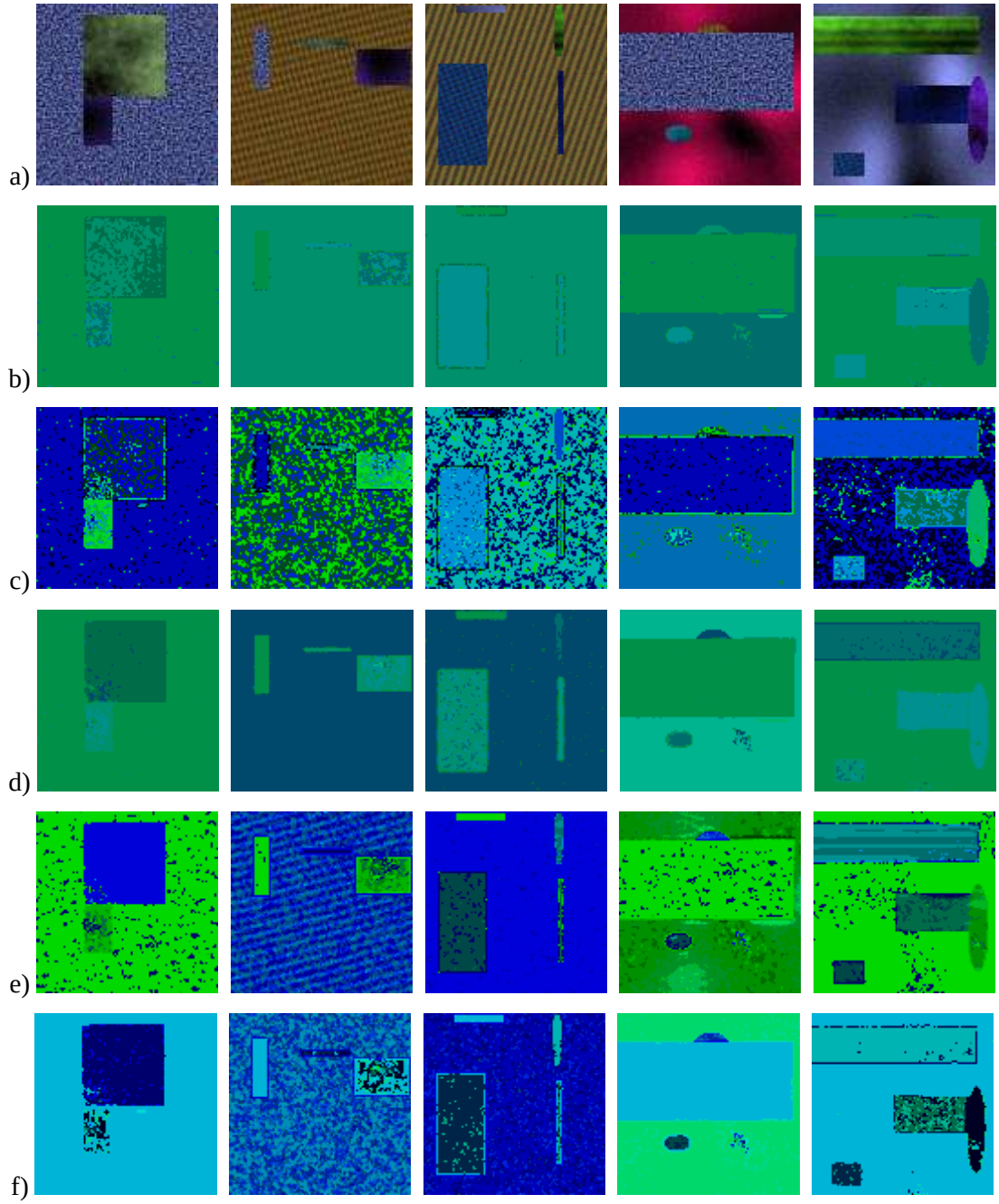
Zaměřím se nyní na porovnání úspěšnosti segmentace vzhledem k volbě prostoru příznaků a velikosti okolí. Z hodnot vyplývá určitá pravidelnost. Pokud nebereme v potaz okolí vítězného neuronu, pak texturní příznaky přináší vyšší úspěšnost segmentace. Samostatný texturní prostor je o něco méně úspěšnější než HS prostor, ale má lepší výsledky než RGB prostor. Výjimečně se stane, že HS prostor je méně úspěšný než texturní příznaky nebo RGB příznaky. Pro hybridní příznaky vždy platí, že jsou úspěšnější než segmentace pouze na zadaném barevném modelu.

Pro okolí vítězného neuronu o velikosti tři je situace poněkud odlišná. Použití samotného texturního prostoru je velice neefektivní a použitím hybridního prostoru dojde k nepatrné ztrátě úspěšnosti oproti barevnému modelu. Největší úspěšnost dosahuje HS model. Situaci vystihuje Obrázek 7-4.

Co se týče účinnosti algoritmu podle učícího faktoru, pak své prvenství znovu obhájila Gaussova funkce, která ve většině případů dominuje oproti ostatním funkcím. Konstantní učící faktor se drží stále ve středních hodnotách, pokud jde o segmentaci bez sousednosti. Při zahrnutí sousednosti neuronů SOM síť častěji dosahovala minimální účinnosti. Nejhorších výsledků pro segmentaci bez sousednosti dosahovalo opět exponenciální určení učícího faktoru.

Pro průběh učení v závislosti na natrénovanosti sítě se dá říci, že konstantní funkce konverguje do dvou iterací a pak se již nemění. Srovnáme-li rozestavení neuronů před začátkem učení a po jeho ukončení, pak zjistíme, že se z celkového počtu neuronů změnila jen taková množina neuronů, která je nejblíže vstupním vzorkům při první iteraci, zbytek neuronů nikdy nevyhraje a tudíž nemění svůj střed. Při zapojení sousednosti neuronů pak dochází k periodickému opakování maximální úspěšnosti v závislosti na počtu iterací. Pro exponenciální, hyperbolickou a lineární funkci platí, že úspěšnost sítě se vždy ustálí na určité hladině. O počtu iterací, která je potřeba pro ustálení sítě, rozhoduje druhý parametr. Pro hodnotu druhého parametru exponenciální funkce rovné 0,5 dojde k ustálení okolo 20. iterace. Při lineární funkci dojde k ustálení pro iterace, které jsou větší než druhý parametr. Při zkoumání Gaussovy funkce se nepodařilo najít charakteristický průběh učícího algoritmu.

Tentokrát bylo do experimentu zahrnuto i hexagonální uspořádání mřížky neuronů. Tento prvek dosahuje srovnatelných nebo lepších výsledků oproti pravoúhlé mřížce.

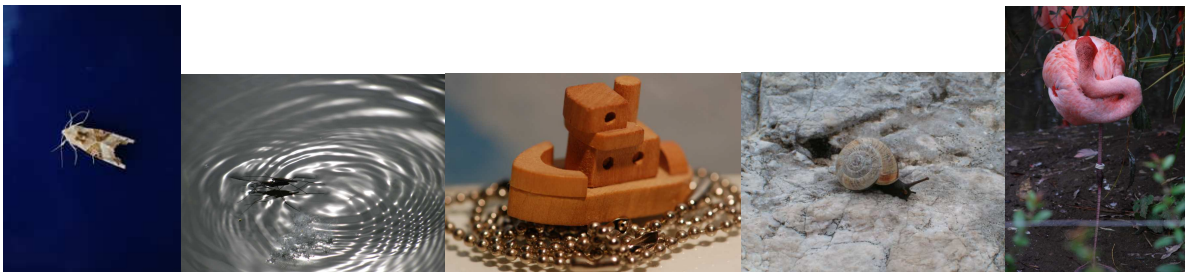


Obrázek 7-4: *Vliv velikosti okolí pro vítězný neuron na úspěšnost segmentace v závislosti na volbě prostoru příznaků (řada a) znázorňuje vstupy segmentace; řada b) je výstup ze SOM sítě s prostorem HS+textury a bez zahrnutí okolí; řada c) je výstup ze SOM sítě s prostorem HS a bez zahrnutí okolí; řada d) je výstup z k-means algoritmu pro prostor HS+textura; řada e) je výstup ze SOM sítě s prostorem HS+textury a pro velikost okolí rovnou 3; řada f) je výstup ze SOM sítě s prostorem HS a pro velikost okolí rovnou 3)*

7.3 Experiment 3 – segmentace reálných obrazů

Nyní již víme charakteristické chování pro jednotlivé algoritmy na syntetické množině obrazů. Účelem segmentace je však aplikace na reálná data. Proto jsem do experimentální části zařadila testování algoritmů na základě reálných dat.

V první části jsem vybrala různorodou skupinu reálných obrazů a na ní jsem natrénovala jak Kohonenovu mapu, tak k-means. Objektivní porovnání bohužel není možné.



a) b) c) d) e)

Obrázek 7-5: *Sada reálných obrazů, které byly vybrány pro trénovací i testovací množinu*

Na vybrané sadě reálných obrazů bych ráda ukázala vlastnosti obou algoritmů. Je vidět, že všechny obrazy obsahují jeden objekt zájmu (motýl, vodoměrka, loď, hlemýžď, plameňák), které budeme chtít detekovat. Objekty se nacházejí na různém pozadí a budeme chtít jejich oddělení. Motýl a plameňák jsou výrazně vidět a u segmentace by mělo dojít k jejich úplné detekci. U plameňáka je velmi složité pozadí a nejspíše bude detekováno jako více objektů. Vodoměrka se na hladině celkem ztrácí, nejspíše budou detekovány vlnky a ráda bych zjistila, jestli se mezi detekovanými vlnkami objeví i vodoměrka. Loď by měla být bez potíží detekovaná. Problém může být s hnědým tónem desky a řetízku. Ty by však měly mít odlišnou odezvu na texturu. Hlemýžď bude velice těžko oddělen od pozadí (zejména ulita, která se liší pouze pravidelností uspořádání „letokruhů“). Jejich odezva na zvoleném okolí bude minimální.

a) b) c) d) e)

Obrázek 7-6: *Výstup z Kohonenovy mapy (velikost 5x5, Gaussův učící faktor 10, počet iterací 16, okolí 0, HS+Gaborovy filtry o vlnové délce 2, 3, 7 a rotaci 0°, 45°, 90°, 135°, Euklidova vzdálenost)*

a) b) c) d) e)

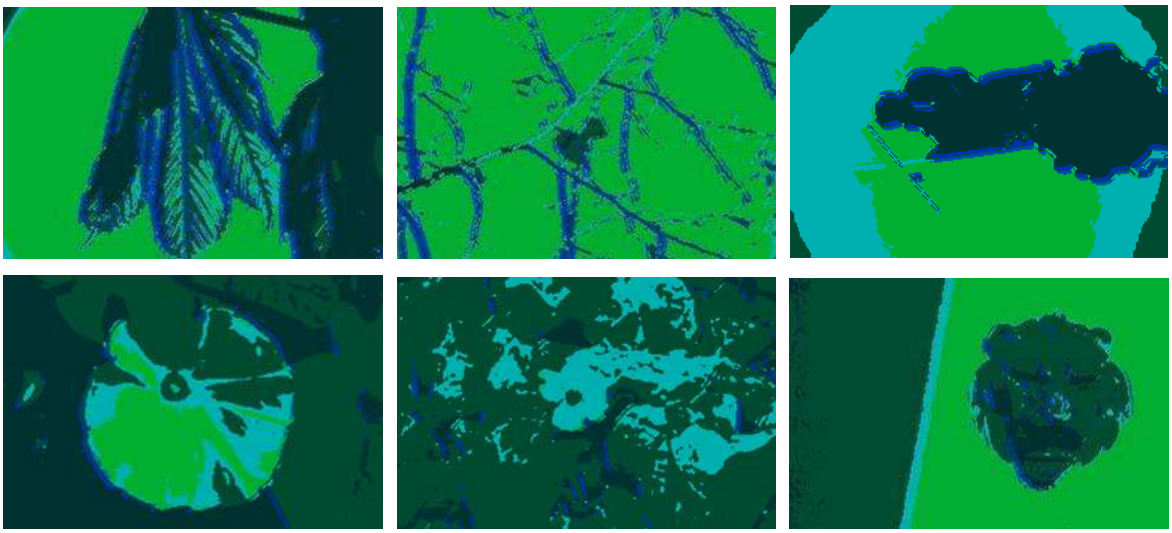
Obrázek 7-7: *Výstup z k-means algoritmu (velikost 5x5, 50 iterací – ustálený stav, HS+Gaborovy filtry o vlnové délce 2, 3, 7 a rotaci 0°, 45°, 90°, 135°, Euklidova vzdálenost)*

Z výstupů na Obrázek 7-6 a Obrázek 7-7 můžeme vyvodit následující závěry. Pro obrázek a) se velice zdařila segmentace pomocí Kohonenovy mapy. Pozadí zde může být zcela eliminováno. Oproti tomu k-means je silně závislé na odstínu pozadí a okolo motýla přidává nepříjemný artefakt stínu. Pro obrázek b) došlo v obou případech k detekci vodoměrky, ale pouze jejího odrazu na vodě. Kohonenova mapa výrazně potlačila vliv vlnek na detekci vody oproti algoritmu k-means. Loď na obrázku c) je pomocí k-means algoritmu segmentována jako jeden objekt, což je vítaný bonus oproti Kohonenově mapě, která je závislá na kresbě dřeva. Bohužel oba algoritmy detekovaly loď společně s řetízkem. Pozadí na Obrázek 7-6 c) je rušeno šumem, ale na Obrázek 7-7 c) je detekováno jako více celistvých segmentů. Segmentace obrázku d) se bohužel nezdařila. Pouze Kohonenova mapa dokázala oddělit část ulity od pozadí. K-means algoritmus přiřadil ulitu k pozadí, ale za to oddělil nohu hlemýždě. SOM síť označila nohu hlemýždě stejně jako tmavou trhlinu ve skále. Plameňák z obrázku e) byl lépe oddělen od okolí pomocí k-means algoritmu. Při použití SOM sítě došlo k detekci stejného segmentu i na pozadí, a tím pádem znemožnil přesný výběr plameňáka. Ani jedna z metod nedokázala na celé sadě obrazů přesně vybrat ústřední motiv.

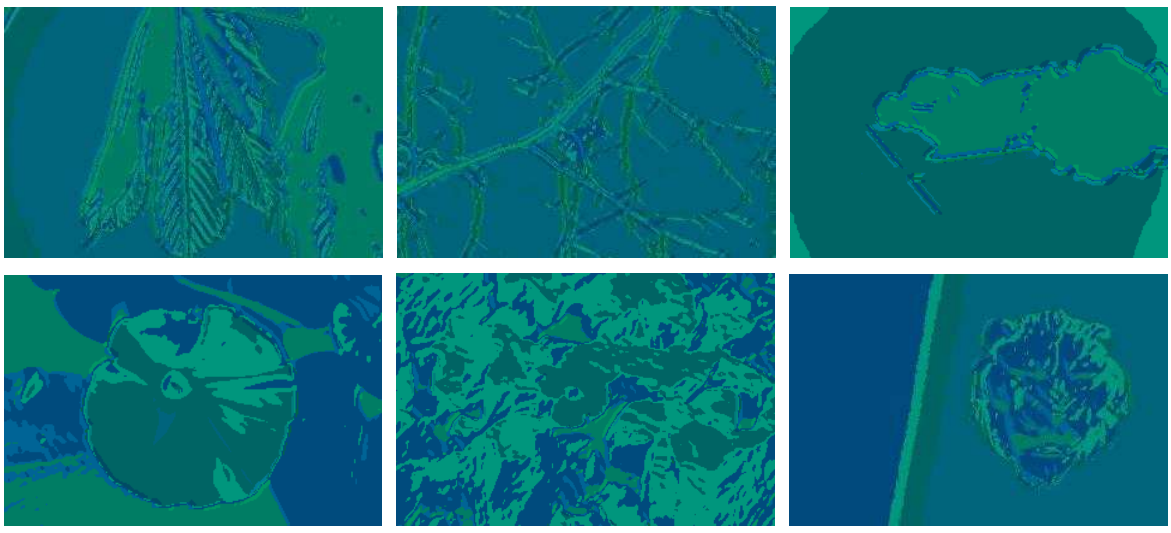
V druhé části jsem si připravila sadu obrazů, které v popředí mají jeden nebo dva objekty a jejichž pozadí je relativně stálé. Pro segmentaci jsem volila různé prostory příznaků a vždy využila stejný počet shluků (10x10).

Trénování algoritmu probíhalo na stejné sadě obrazů, na kterou byla i aplikována. Výběr trénovací množiny byl náhodný s implicitní redukcí množiny.

Obrázek 7-8: *Trénovací a testovací sada reálných obrazů*

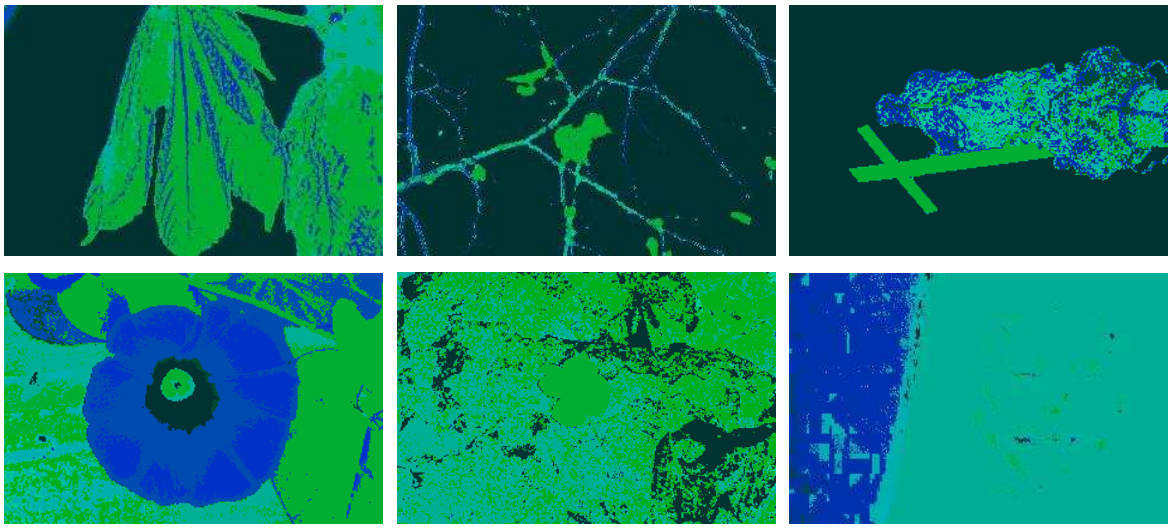


Obrázek 7-9: *Výstup z k-means algoritmu při použití pouze texturních příznaků (Gaborovy filtry o vlnové délce 2, 4, 7 a natočení 0°, 45°, 90°, 135°)*

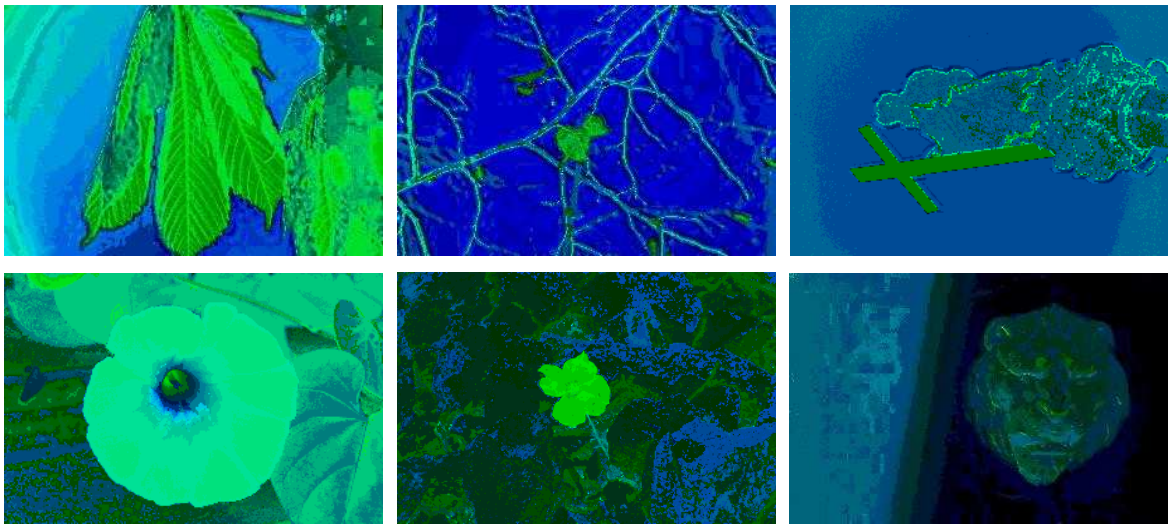


Obrázek 7-10: *Výstup ze SOM sítě při použití texturních příznaků (využívá Gaussovu funkci s parametrem 6 a výsledek je zachycen po 12 iteracích)*

Výstup z k-means algoritmu působí po segmentaci na základě textury rozmazaně. U SOM sítě došlo k větší generalizaci na plošná vstupní data. Bohužel v obou případech jsou segmentované obrazy nepoužitelné z hlediska detekce více objektů jako jeden segment.

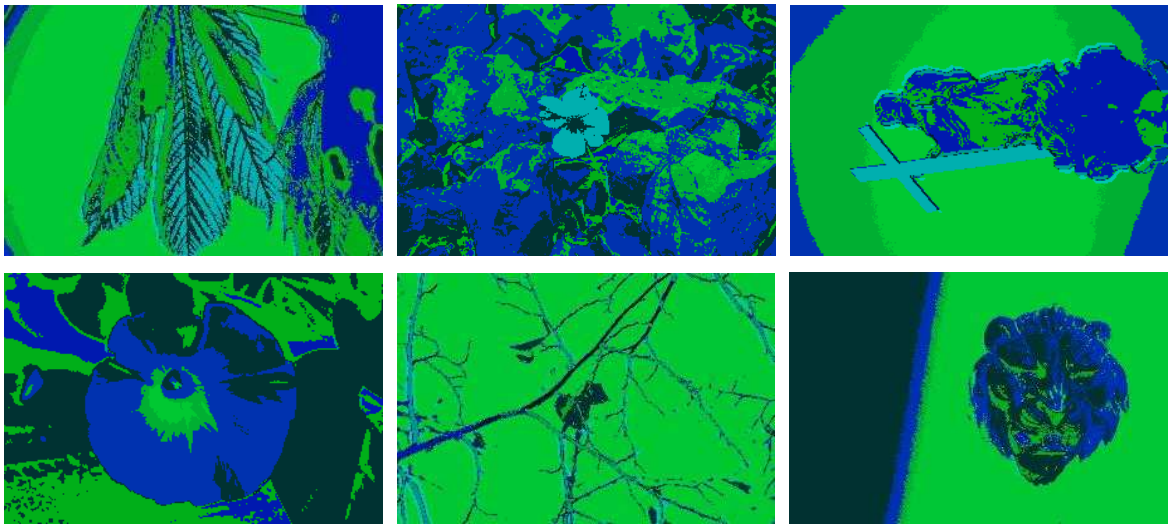


Obrázek 7-11: *Segmentace pomocí k-means algoritmu na základě přidání barevného modelu HS do prostoru příznaků*

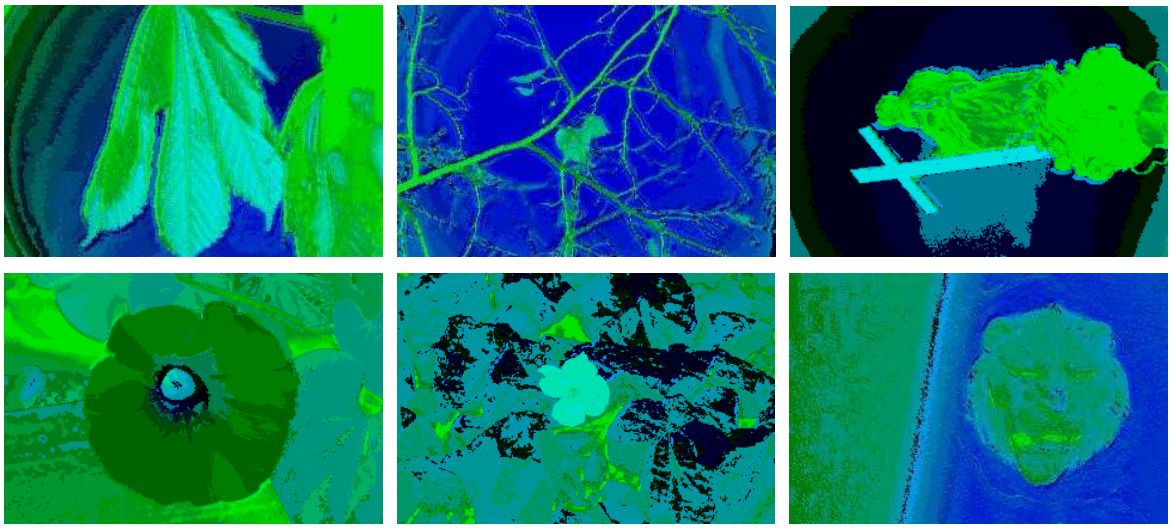


Obrázek 7-12: *Segmentace pomocí SOM sítě na základě stejného prostoru příznaků (využita hexagonální topologie sítě s okolím 1, učící faktor je exponenciální s parametry 0,8 a 0,5, výstup po 12 iteracích)*

Propojíme-li texturní příznaky s informací o barvě aktuálního pixelu na základě HSV modelu, získáme již větší rozlišitelnost jednotlivých objektů ve scéně. SOM síť byla úspěšnější v detekci květinky mezi kamením nebo detekci lví hlavy. Bohužel díky použití sousednosti mezi neurony došlo k rozdělení pozadí do více segmentů.



Obrázek 7-13: *Segmentace na základě k-means algoritmu s RGB modelem a texturou*



Obrázek 7-14: *Segmentace pomocí SOM sítě na základě prostoru RGB + textura (hexagonální topologie, sousednost rovna 1, exponenciální učící faktor s parametry 0,8 a 0,5, ukončeno ve 12. iteraci)*

Při použití RGB modelu místo HSV modelu získáme vyšší závislost na barvě, což může prospět segmentaci, jako například při detekci lví hlavy pomocí k-means algoritmu, nebo uškodit, jako v případě neidentifikovatelného tvaru poblíž kříže světce v případě aplikace SOM sítě.

8 Další možnosti rozvoje systému

Jak je vidět z popisu návrhu a implementace segmentačního systému, jedná se o API, které pokrývá různé oblasti využití počítačů (např. syntéza scény, zpracování obrazu, inteligentní systémy apod.). Diplomová práce se zaměřuje obzvláště na oblast zpracování obrazu s použitím inteligentních systémů. Ostatní oblasti jsou zde využity jen podpůrně, přestože jejich vylepšení by mohlo výrazně přispět k lepším vlastnostem optimalizovaných systémů a umožnit tak aplikaci stát se mocným nástrojem pro přípravu segmentačních algoritmů.

Kapitola obsahuje seznam nejprospěšnějších optimalizací, jejich návrh a diskuzi o tom, jakým směrem se budou dále rozvíjet.

8.1 Generování realističtějších textur

Článek [19] vypovídá o možnosti generování složitých textur za pomoci Perlinova šumu. Diskutuje jeho modifikace pro vytváření procedurálních textur, které mají realistický vzhled. Popsaný algoritmus je však náročný na výběr vhodného generátoru pseudo-náhodných čísel tak, aby výsledný dojem nebyl narušen nepředpokládanou pravidelností generovaných čísel.

Lze experimentovat se skládáním nejrůznějších matematických funkcí pro získání potřebné textury. Za pomoci Perlinova šumu lze vygenerovat mraky, vodu, oheň, mramor, dřevo, lesy, skály, vlasy a další. Vezmeme-li v potaz, že při generování textur nám jde především o simulaci reálného prostředí, pak Perlinův šum může přinést do systému výrazné zlepšení ve složení trénovací množiny a lepší použití optimalizovaného segmentátoru na reálné obrazy.

8.2 Paralelní zpracování algoritmů

Za posledních pár let došlo k výraznému posílení výpočetní síly paralelních počítačových sestav a jejich dostupnosti pro širokou veřejnost. Nejdříve se objevily vícejádrové výpočetní procesory, které byly následovány vysoce paralelními procesory na grafických kartách (např. grafické karty od nVidia). Pro další rozvoj systému a jeho urychlení se výrazně nabízí využití standardu CUDA od společnosti nVidia, případně nástavbové knihovny, např. OpenCL. Ta využívá knihovnu nad grafickými kartami společnosti nVidia, ale také i nad konkurenčními kartami od společnosti ATI. Dále umožňuje spolupráci s grafickými API jako jsou DirectX nebo OpenGL. Pro využití CUDA architektury je zajímavé její rozšíření i na další programovací jazyky jako je Java (JCuda) nebo Python (PyCuda).

Vlastní konstrukce segmentačních algoritmů se přímo nabízí k paralelnímu zpracování. Jedná se především o výpočet vítězného shluku, kdy by bylo možné vypočítat vítězný neuron pouze v čase

potřebném pro výpočet hodnotící funkce pro jeden neuron. Zaměřím se nyní pouze na neuronové sítě, jejichž výpočetní náročnost je složitější než u k-means. V další fázi by se stanovila vzdálenost od vítězného neuronu a naposled by se přepočítaly váhy pro vybrané okolí vítězného neuronu. Natrénování sítě by proběhlo v časové složitosti odpovídající přepočtu jednoho neuronu, alespoň pro menší počet shluků.

Přestože je prostředí CUDA vysoce paralelní a uživatelsky přívětivé, může se nesprávným použitím dospět k výraznému zpomalení oproti optimalizovanému kódu. Neoptimalizovaný kód může být klidně 14x pomalejší než kód optimalizovaný, jak uvádí [20]. Hlavním zdrojem zpomalení jsou:

- různé typy pamětí, které se liší
    - dobou přístupu,
    - HW ochranou přístupu při čtení/zápisu dat,
    - velikostí paměti;
- časté přesuny dat po sběrnici spojující CPU a GPU;
- nevyvážený objem instrukcí v jednotlivých vláknech (skoky, smyčky apod.)
    - pro if konstrukci se vždy vykonají oba případy, zápis do paměti však proběhne jen v oprávněném případě.

Učící algoritmus pro menší Kohonenovy sítě by bylo možné docela výrazně urychlit. Učení neobsahuje žádné smyčky, jejichž počet opakování by byl odlišný pro jednotlivé neurony. Obsahuje pouze jednu podmínku, a to pro přepočítání vah okolních neuronů. Podle velikosti okolí se určuje loadbalance. Každý neuron má přidělený privátní prostor, do kterého se provádí zápis (vektor vah) a žádný jiný neuron nemá čtecí práva. Výstup neuronové sítě při učení není důležitý z pohledu okolních aplikací, a tudíž není potřeba přenášet data po sběrnici.

Zde je jen lehký nástin důvodů Kohonenovy mapy pro paralelizaci. Při podrobnějším návrhu pro určitou architekturu může být nalezeno další „úzké hrdlo“ nebo některé z výše zmíněných problémů eliminovány. V kapitole jsem se zaměřila na nejčastější problémy spojované s CUDA architekturou.

8.3 Rozšíření množiny segmentačních algoritmů

Jak jsem se zmínila v teoretické části diplomové práce, existuje mnoho různých typů neuronových sítí, lišící se použitými funkcemi a propojením. Pro účely segmentace mohou být využity různé typy sítí, například na bázi učení s učitelem. Tento typ učení by bylo vhodné implementovat po zdokonalení generátoru textur, kde by bylo možné jasněji určit typ textury.

Zajímavým prvkem by bylo samostatné využití jiných neuronových sítí. Ke zlepšení výsledků segmentace by mohlo vést použití RBF a případně její modifikace neuronů na neurony s bázovou funkcí implementující Gaussovu funkci, která umožní přiřazení vstupu v eliptickém okolí středu neuronu a váhováním příslušnosti.

Dalším rozšířením použitelnosti aplikace by mohl být správce matematických funkcí, čímž by bylo možné navolit libovolný průběh jednotlivých funkcí v neuronových sítích. Stejný mechanismus byl zařazen pro konvoluční filtry a přineslo to větší možnosti ve výběru prefiltrace a volbě prostoru příznaků. Tento správce by mohl výrazně přispět k optimální adaptaci sítě na trénovací množinu.

9 Závěr

Cílem diplomové práce bylo shrnout požadavky na vhodný segmentační systém, najít v široké škále neuronových sítí algoritmus, který by daný problém dokázal řešit se srovnatelnou úspěšností jakou lze dosáhnout pomocí konvenčních metod. Hlavním úkolem experimentální části bylo ověření předpokladů v praxi za pomoci implementovaného systému a odhalit případné nedostatky využití neuronových sítí pro segmentaci dat. Velká část diplomové práce byla věnována automatické přípravě vstupních a trénovacích dat pro úspěšnou optimalizaci segmentačních metod.

V rámci diplomové práce se podařilo navrhnout aplikaci, která umožňuje uživateli různorodou skladbu možností pro optimální natrénování segmentačního algoritmu na základě Kohonenovy mapy. Analýzou segmentačních metod jsem zjistila, že Kohonenova mapa má velice blízko k algoritmu k-means, který byl pro účely této práce vybrán jako referenční algoritmus. Kohonenova mapa však svojí strukturou umožňuje různé přístupy k segmentaci. Zatímco k-means algoritmus je pevně daný a možnosti jeho modifikace jsou omezené.

Právě otevřenost Kohonenových map k různým modifikacím je jejich největší výhodou, ale zároveň skrývá i různé nástrahy, na které si musí dát uživatel pozor. Špatná volba vstupních parametrů může snadno znehodnotit celou adaptaci algoritmu na trénovací množinu. Oproti tomu k-means algoritmus je robustní a striktní. Standardně dává přijatelné výsledky a rozhoduje se na základě získaných informací během jedné iterace.

V experimentech se mi podařilo doložit, že Kohonenova mapa může být optimalizována zhruba po pěti až deseti iteracích a její výstup je srovnatelný a často i předčí výsledek k-means algoritmu, který k optimalizaci potřebuje zhruba třicet iterací. Přívětivý průběh optimalizace Kohonenovy sítě přinesla volba učícího faktoru podle Gaussovy funkce. Standardně vykazovala kvalitu segmentace o 5 až 15 procent vyšší než dosáhl algoritmus k-means při stejné inicializaci středů a stejném prostoru příznaků. Pro tuto funkci je vhodné volit parametr, který je menší než počet chtěných iterací, ve kterých se má síť ustálit. Volba prostoru příznaků obstála v experimentech a propojení informace o barvě spolu s texturou.

Kohonenova mapa má další výhodu a to v možnosti interpolace příznakového prostoru použitím sousednosti. Statistická správnost segmentace sice prudce klesá, ale zaměříme-li se na objekt, pak zjistíme, že Kohonenova mapa rozdělila objekt do tematických segmentů, které umožní vyhledávání drobných nuancí na sadě reálných obrazů. Pro tuto metodu segmentace je nutná postfiltrace, která dokáže sestavit menší segmenty zpět do logických celků.

Při testování segmentačních algoritmů na reálných datech dokázala Kohonenova mapa extrahovat objekty zájmu od pozadí poměrně úspěšně. V některých situacích to zvládá daleko lépe než k-means algoritmus. Hlavní výhodou Kohonenovy mapy je, že když získáme nevhodnou segmentaci, je možné síť znova optimalizovat a upravit průběh učení dle potřeby.

Literatura

[1] Španěl, M., Beran, V.*: Obrazové segmentační techniky. Přehled existujících metod.* aktualizováno 19.1.2006. [online] Dostupné na URL: <http://www.fit.vutbr.cz/~spanel/segmentace>

[2] Zbořil, F.: *Biologický a umělý neuron, neuronové sítě. Perceptron,Adaline, Madaline a BP (Back Propagation).* [online] Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/SFC/private/10sfc_2.pdf> (26. 12. 2010)

[3] Zbořil, F.: *Neuronové sítě RBF a RCE. Topologicky organizovanéneuronové sítě, soutěživé učení, Kohonenovy mapy.* [online] Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/SFC/private/10sfc_4.pdf> (26.12.2010)

[4] Španěl, M.: *Statistické rozpoznávání a shlukování.* [online] Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/POV/private/lectures/pov_02_statisticke_rozpoznavani.pdf> (26. 12. 2010)

[5] Španěl, M.: *Segmentace obrazu, analýza histogramu, analýza barev, shlukování.* [online] Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/POV/private/lectures/pov_04_segmentace_obrazu.pdf > (26.12.2010)

[6] Zuzaňák, J.: *Analýza a extrakce příznaků z textur.* [online] Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/POV/private/lectures/pov_08_analyza_a_extrakce_priznaku_z_textur.pdf> (26.12.2010)

[7] Janoušek, V.: *Machine Learning and Neural Networks.* Vytvořeno 2007. [online] Dostupné na URL: <http://www.fit.vutbr.cz/study/courses/SIN/private/current/SIN-NN.pdf> (26.12.2010)

[8] Agoston, Max K.: *Computer Graphics and Geometric Modeling: Implementation and Algorithms.* London: Springer. 2005. ISBN 1-85233-818-0. str. 303–304.

[9] *blog.orflex.sk: RGB alebo CMYK?.* [online] Dostupné na URL: <http://blog.orflex.sk/2006/11/rgb-alebo-cmyk.html> (26.12.2010)

[10] *Digital Camera & Photography Information posted by Jusanji - HSL and HSV.* [online] Dostupné na URL: <http://www.jusanji.co.kr/67633> (26.12.2010)

[11] Španěl, M.: *Chyby v obraze, typy šumu, restaurace obrazu a optimální filtrace.* [online] Dostupné na URL: <http://www.fit.vutbr.cz/study/courses/ZPO/private/lectures/zpo_11_sum_v_obraze.pdf> (26.12.2010)

[12] *File:Konvoluce 2rozm diskretni.jpg - Wikimedia Commons.* [online] Dostupné na URL: <http://commons.wikimedia.org/wiki/File:Konvoluce_2rozm_diskretni.jpg> (26.12.2010)

[13] Zemčík, P.: *Lineární filtrace obrazu.* [online] Dostupné na URL: <http://www.fit.vutbr.cz/study/courses/ZPO/private/lectures/zpo_05_filtrace.pdf> (26.12.2010)

[14] Potůček, I.: *Bodové transformace (funkce) obrazu.* [online] Dostupné na URL: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZPO-IT/lectures/zpo_03_bodove_tran.cs.pdf > (26.12.2010)

[15] Petkov, N., Wieling, M.B.: *Gabor filter for image processing and computer vision.* [online] Dostupné na URL: <http://matlabserver.cs.rug.nl/edgedetectionweb/web/edgedetection_params.html> (26.12.2010)

[16] *Java Kohonen Neural Network Library.* [online] Dostupné na URL: <http://jknnl.sourceforge.net/> (20.7.2011)

[17] *JDOM.* [online] Dostupné na URL: <http://www.jdom.org/index.html> (20.7.2011)

[18] *Minkowski distance - Wikipedia, the free encyclopedia.* Dostupné na URL: <http://en.wikipedia.org/wiki/Minkowski_distance> (20.7.2011)

[19] Perlin, K.: *An Image Synthesizer.* 1985. Dostupné na URL: …. (20.7.2011)

[20] Nikolopoulos, D. S: *Topics on CUDA Optimization.* Dostupné na URL: <http://www.csd.uoc.gr/~HY529/lectures/lecture08-handout.pdf> (20.7.2011)

[21] *File:Gaussian 2d.png - Wikipedia, the free encyclopedia.* [online] Dostupné na URL: <http://en.wikipedia.org/wiki/File:Gaussian_2d.png > (23.7.2011)

Seznam obrázků

Seznam tabulek

Seznam zkratek

RGB ... barevný model pojmenovaný podle názvu svých složek Red, Green, Blue

HSV ... barevný model pojmenovaný podle názvu svých složek Hue, Saturation, Value

SOM ... Self-Organized Map

CPU ... Compute Processor Unit

GPU ... Graphics Processor Unit

HW ... Hardware

SW ... Software

Seznam příloh

DVD