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

AUTOMATICKÁ SEGMENTACE DOKUMENTŮ

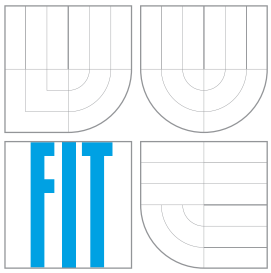
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DUŠAN JAKUB

BRNO 2012

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

AUTOMATICKÁ SEGMENTACE DOKUMENTŮ

AUTOMATIC SEGMENTATION OF DOCUMENTS STORED AS IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DUŠAN JAKUB
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZŐKE, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Práce se zabývá rozčleněním dokumentů uložených jako obrázek do segmentů trojího druhu – pozadí, text a grafické objekty. Představuje různé způsoby řešení a podrobněji popisuje postup využívající Gaborovy filtry a neuronové sítě. Je diskutována volba vhodných parametrů filtrů i trénování sítě. Pro zpřesnění výsledků je použita metoda hledání souvislých komponent. Součástí práce je klasifikátor v jazyce C++ vytvořený za použití knihovny OpenCV. Navržený postup byl koncipován pro segmentaci dokumentů publikovaných ve vědeckých časopisech a uložených jako obrázek např. po skenování. Vedle výsledků segmentace odborných textů jsou v práci prezentovány také experimenty se segmentací dokumentů jiného charakteru, např. reklamního letáku a slidů prezentace. V závěru je demonstrován přínos navrženého postupu při zapojení do procesu optického rozpoznávání znaků.

Abstract

This work deals with dividing the documents stored as images into three groups of segments – background, text and graphics. It introduces various solutions and the method using Gabor filters and artficial neural networks is described in detail. The selection of apropriate settings of the filters and training parameters of the network is discussed. Connected components searching is used for improving the results. A classifier writen in C++ and OpenCV library is part of the work. The designed procedure is applied for segmentation of scanned scientific papers, but also the results of segmentation of more complex documents (advertisements, presentation slides) are presented.

Klíčová slova

Segmentace dokumentu, detekce textu, texturové příznaky, Gaborovy filtry, umělá neuronová síť, OpenCV

Keywords

Document segmentation, text detection, texture analysis, Gabor filters, artificial neural network, OpenCV

Citace

Dušan Jakub: Automatická segmentace dokumentů, bakalářská práce, Brno, FIT VUT v Brně, 2012

Automatická segmentace dokumentů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Igora Szőka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dušan Jakub
15. května 2012

Poděkování

Děkuji svému vedoucímu Ing. Igoru Szőkovi Ph.D. za podporu a cenné rady při tvorbě práce.

Obsah

Kapitola 1

Úvod

Počítačové čtení, tedy převod obrazových dat do formy textu složeného ze znaků, je dnes již poměrně běžná záležitost. Snad i ten nejlevnější skener pro běžné domácí použití je distribuován s nějakým programem pro OCR (Optical character recognition), jehož výstupem je naskenovaný dokument v editovatelné textové podobě.

Různě modifikované OCR algoritmy mají ale mnohem širší využití. Najdeme je na poštách, kde pomáhají třídit zásilky podle směrovacích čísel a adres, v účetních odděleních firem, kde automaticky digitalizují faktury, či třeba na finančních úřadě, kde zpracovávají daňová přiznání. Koneckonců, jistě je využívá i naše fakulta při automatickém opravování našich půlsemestrálních a semestrálních testů.

Sebelepší algoritmus pro rozpoznání znaků je ale k ničemu, pokud nedokáže ve vstupním obrazovém souboru najít samotný text. Často je tomu napomáháno speciálním druhem formuláře – směrovací čísla na obálkách vpisujeme do předtištěných rámečků, stejně jako login na semestrálním testu.

V mnoha případech však nemáme tato umělá vodítka k dispozici – například při skenování knihy nebo vědeckého článku. Existují proto algoritmy, které v obrazu dokáží najít odstavce, řádky či samotné znaky a naopak vynechat ilustrační obrázky nebo grafy či tabulky. Každý OCR nástroj musí samozřejmě také implementovat některý z těchto algoritmů, jinak by nebyl prakticky použitelný. Problém spočívá v tom, že druhů rozvržení dokumentů existuje celá řada, takže vytvořit univerzální algoritmus je obtížné.

Tato práce se má zabývat zejména segmentací dokumentů, které byly publikovány ve vědeckých časopisech. Algoritmy tedy budou optimalizovány především pro dokumenty spíše jednodušší struktury, ať již po stránce rozvržení či barevnosti, které však obsahují řadu vnořených objektů, například schémat, grafů či tabulek, které se od prostého textu odlišují hůře než třeba fotografie.

V kapitole 2 této práce popisuji některé fáze procesu OCR a představuji výběr z metod, které se dají použít pro segmentaci dokumentu. Kapitola 3 pojednává o konkrétních postupech, která jsem si zvolil, a jsou zde popsány Gaborovy filtry a umělé neuronové sítě. V kapitole 4 popisuje aplikaci těchto postupů v mém projektu a konečně v kapitole 5 předvádím výsledky své práce.

Kapitola 2

Cíle práce a její zařazení do širšího kontextu

V souladu se zadáním je bakalářská práce zaměřena na vyhledávání textu ve vědeckých článcích. V podobných textech jsou detekovány textové a netextové objekty, například obrázky, tabulky či grafy. I netextové objekty mohou v sobě obsahovat text (např. popisky os grafu). Také tyto segmenty textu mohou být užitečné třeba pro vyhledávání klíčových slov. Výstupem zpracování vybraného objektu metodou popsanou v této práci je reprezentace pozic těchto objektů spolu s jejich typem a dalšími zjištěnými charakteristikami. Součástí práce je i nástroj pro vymaskování objektů určeného typu. Toto slouží například pro vyříznutí textových objektů a jejich následné převedení na text pomocí OCR.

2.1 Zpracování dokumentu

Segmentace je jen jednou z mnoha fází OCR. Než se zaměřím na samotnou segmentaci dokumentu, zmíním zde krátce operace, které segmentaci dokumentu předcházejí nebo na ni navazují.

Oprava vstupního obrazu Oskenované obrázky jsou často nekvalitní. Mohou obsahovat šum, mohou být různě otočené či jinak deformované. První fází proto často bývá náprava těchto škod. Ve své práci předpokládám, že vstupní obrázky již touto fází prošly.

Analýza dokumentu Po segmentaci obrazu (které je předmětem této práce) velmi často následuje fáze analýzy. Zde jsou objekty nalezené segmentační fází klasifikovány například na nadpisy, odstavce textu, popisy obrázků a podobně. Je zde také stanoveno pravděpodobné pořadí odstavců textu (jsou brána v úvahu třeba vícesloupcová rozvržení a podobně).

Ačkoli tato fáze není přímo předmětem mé práce, výstupem mého programu je soubor popisující každý nalezený textový segment. Kromě jeho polohy a velikosti v pixelech je zde také informace o velikosti písma a počtu řádků segmentu. Tyto informace mohou být dále při analýze použity.

2.2 Možné způsoby segmentace

Existuje mnoho metod, které se dají použít k segmentaci textů uložených jako bitmapové obrázky, v této kapitole představím některé z nich.

2.2.1 Rozdělení algoritmů

Segmentační algoritmy můžeme rozdělit na metody pracující shora dolů (top-down) a zdola nahoru (bottom-up). Top-down algoritmy začínají s jedním velkým segmentem pokrývajícím celou stránku, který postupně rozdělují. Oproti tomu v bottom-up algoritmech je výchozím stavem mnoho malých segmentů, často jednotlivých pixelů obrázku, které jsou spojovány.

2.2.2 Constrained run length (CRL) algoritmus

Algoritmus CRL patří mezi top-down. Lze ho aplikovat pouze na černobílé obrazy, které jsou binárně kódovány jako 1 pro černý a 0 pro bílý pixel [8].

Princip funkce algoritmu CRL spočívá v tom, že souvislé skupiny nul v každém řádku, respektive sloupci obrazu, které jsou kratší než předem stanovená mez (značíme C – constraint), jsou změněny na jedničky. Tímto dojde vlastně ke slití černých ploch, které jsou od sebe vzdáleny méně než C pixelů. Tuto operaci provedeme pro všechny řádky a pro všechny sloupce obrazu, čímž dostaneme dvě výsledné bitmapy: vzniknou horizontálně a vertikálně. Tyto výsledky sloučíme pomocí operace AND a tím získáme definitivní výsledek.

Souvislé černé plochy v tomto výsledku pak představují jednotlivé segmenty. Pro každý segment lze spočítat řadu příznaků (šířka, délka, jejich poměr, poměr černých pixelů ku počtu pixelů segmentu, poměr černých pixelů oproti ploše obdélníkové obálky segmentu a další). S využitím těchto příznaků lze každý segment zařadit do jedné ze 4 kategorií: text, obrázek vytvořený pomocí techniky halftoning a horizontální a vertikální čára.

Algoritmus vyžaduje zadání vhodných hodnot C (typicky se využívá jiné C pro vertikální průchod než pro horizontální). Jeho nevýhodou je také fakt, že nedetekuje text ve vložených netextových objektech.

2.2.3 X-Y řezy

Metoda X-Y řezů pracuje tak, že se snaží segment rozdělit na více menších segmentů [7]. Začíná s jediným segmentem pokrývajícím celou stránku, jedná se tedy také o top-down algoritmus.

Segmenty jsou vždy obdélníkové se stranami rovnoběžnými s osami obrázku. Metoda proto funguje pouze na nerotované vstupní obrázky. Je zvláště vhodná k segmentaci dokumentů, které samy o sobě obsahují obdélníkové bloky (např. noviny nebo vědecké články). Naopak obsahuje-li dokument složitá obtékání plovoucích objektů, může tato metoda selhat.

K určení hranic, podle kterých se segment rozdělí, se často používají projekční profily.

Projekční profily

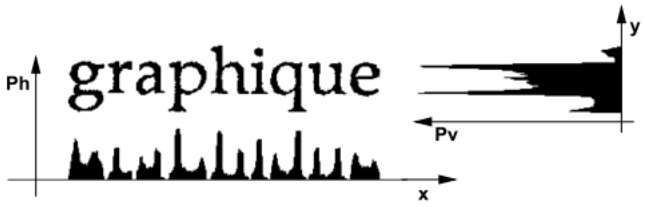
Projekční profily v základním tvaru jsou definované v [12]. Mějme černobílý obrázek o N řádcích a M sloupcích. Každému pixelu $S(M, N)$ je přiřazena hodnota 1, je-li černý, nebo 0, je-li bílý. Obrázek pak popisuje matice S o rozměrech $N \times M$.

Potom můžeme definovat vertikální profil jako

$$P_v[i] = \sum_{j=1}^{M} S[i, j] \tag{2.1}$$

a podobně horizontální profil jako

$$P_h[j] = \sum_{i=1}^{N} S[i,j] \tag{2.2}$$

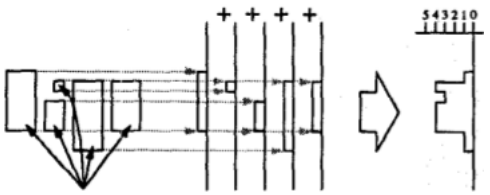


Obrázek 2.1: Projekční profily (převzato z [12])

Vidíme, že 2D matici jsme projektovali do dvou 1D vektorů, kdy hodnota prvku vektoru na konkrétní pozici je rovna počtu všech černých pixelů v odpovídajícím řádku či sloupci původního obrázku.

V [7] definují projekční profily pomocí nikoli pomocí počtu pixelů, ale počtu obálek spojitých ploch. Nejprve ve vstupním obrázku najdou všechny spojité černé plochy a zaznamenají si obalující obdélník každé z nich. Místo jednotlivých pixelů pak sčítají, kolik objektů, resp. jejich obálek je na řádku či v sloupci.

Podle citovaného článku [7] má tato verze profilů lepší vypovídající schopnost o původním obrázku než prosté sečtení pixelů, protože lidský mozek také reaguje na celá písmena, nikoli na jednotlivé body, kterými jsou tvořena.



Obrázek 2.2: Alternativní projekční profily, které uvažují jen obálky (převzato z [7])

2.2.4 Seskupování komponent

Jako příklad postupu zdola nahoru uvádím metodu z [5]. V navrženém algoritmu se nejprve najdou spojité černé plochy a pro každou se spočítá její obálka. Příliš velké obálky se odfiltrují, aby zbyly jen potencionální textové znaky.

Na středy těchto obálek je aplikována Houghova transformace (původně metoda pro detekci čar v obraze), pomocí které se dá odhadnout, která písmena leží na stejné přímce a mohou tak být součástí jednoho textu. Protože však písmena se liší ve velikosti a pozici na řádku, je třeba zavést určitou míru tolerance. Nakonec je spočítána vzdálenost mezi písmeny v získané skupině, která je porovnána s maximální povolenou vzdáleností mezi znaky a slovy.

2.2.5 Texturové příznaky

Při využití texturových příznaků se každému bodu na bitmapě přiřadí hodnota či častěji vektor hodnot, získaný pomocí filtru, jehož jádro je vycentrováno na tento bod. V literatuře [10] se toto číslo také nazývá „lokální energie." Vychází se z myšlenky, že při vhodně zvoleném filtru nebo sady filtrů budou mít pixely patřící do stejného typu segmentu podobnou energii.

V další fázi je třeba podle energií jednotlivých bodů rozhodnout, do které kategorie spadají. Toho lze dosáhnout více způsoby, od prostého shlukování po složitější klasifikátory.

2.2.6 Zhodnocení metod segmentace

Algoritmus CRL dokáže seskupit písmena do slov či odstavců, ale dosahuje toho pomocí spojení blízkých objektů bez rozlišení jejich druhu. Informace o druhu objektu jsou málo konkrétní a spolehlivé, proto pro účely této práce není příliš vhodný. V druhé fázi svého projektu (kdy už jsou vyfiltrovány jen textové objekty) používám ale velmi podobný způsob pro sestupování.

Metoda X-Y řezů vyžaduje, aby dokument byl členěn do obdélníkových segmentů. Na hlavní tok textu bychom mohli tento předpoklad aplikovat, avšak já zamýšlím detekovat i text vložený do grafu nebo třeba tabulku. Přesto ani tento postup nenechám bez využití, vertikální projekční profily totiž využívám pro detekci velikosti a počtu řádků v odstavci.

Algoritmus seskupování komponent se orientuje spíše na rozpoznání slov, řádků a odstavců. Podle velikosti obálek je sice možné získat určité povědomí o druhu objektu, ale pro kvalifikované oddělení textu od jiných objetů to nestačí.

Jako základní postup ve své práci jsem zvolil metodu texturových příznaků. Její velká výhoda spočívá ve faktu, že rozpoznává každý pixel jednotlivě podle jeho okolí, takže lze nalézt i vnořené textové objekty v obrázcích či grafech. To je ale zároveň i nevýhoda, protože tak jeden spojitý objekt může být rozdělen do více druhů. Proto jsou v druhé fázi využívány i postupy z dalších zde popsaných algoritmů.

Kapitola 3

Použité postupy

Pro svoji práci jsem si zvolil metodu texturových příznaků. V této kapitole představím podrobněji postupy a algoritmy, které jsem využil.

3.1 Gaborovy filtry

Jedním z používaných texturových příznaků jsou Gaborovy filtry. Jak již bylo řečeno, texturové příznaky využívají lokální energii v okolí každého pixelu obrázku.

3.1.1 Definice

V případě Gaborových filtrů je lokální energie definována takto [10]:

$$h(x, y) = s(x, y)w(x, y), \tag{3.1}$$

kde $s(x, y)$ je komplexní sinusoida, zvaná nosič, a $w(x, y)$ je 2D Gaussova funkce, zvaná obálka.

Nosič

Komplexní sinusoida je v tomto tvaru:

$$s(x, y) = e^{j(2\pi(u_0 x + v_0 y) + P)} \tag{3.2}$$

Z ní můžeme získat její reálnou a imaginární složku:

$$Re(s(x, y)) = cos(2\pi(u_0 x + v_0 y) + P) \tag{3.3}$$
$$Im(s(x, y)) = sin(2\pi(u_0 x + v_0 y) + P) \tag{3.4}$$

Parametry u_0 a v_0 označují horizontální a vertikální frekvenci filtru (v kartézských souřadnicích), P je počáteční fáze.

Pro účely této práce je však vhodnější počítat se souřadnicemi polárními (otočení ω_0 a vzdálenost od počátku F_0):

$$F_0 = \sqrt{u_0^2 + v_0^2} \tag{3.5}$$
$$\omega_0 = arctan\left(\frac{u_0}{v_0}\right) \tag{3.6}$$

a naopak

$$u_0 = F_0 cos\omega_0, \tag{3.7}$$
$$v_0 = F_0 sin\omega_0. \tag{3.8}$$

Obálka

Obálka může být použita více způsoby. Nejjednodušším z nich je Gaussova funkce, která klesá od středu ve všech směrech stejně rychle:

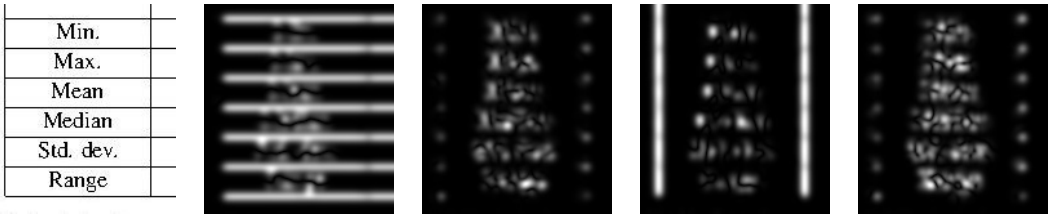
$$w_r(x, y) = K\ e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.9}$$

Parametr K ovlivňuje pouze amplitudu funkce a pro účely této práce jej položíme roven jedné. Parametr σ značí směrodatnou odchylku.

Existují i složitější varianty, ve kterých vertikální odchylka není rovna horizontální, takže plocha má eliptický tvar. Navíc může být rotovaná. Ve své práci však využívám základní tvar, kde obě odchylky jsou si rovny.

3.1.2 Aplikace

V zásadě jde tedy o modifikovaný Gaussův filtr, který však reaguje jen na určité prostorové frekvence. Tato vlastnost je pro rozlišení textu od jiných segmentů velmi důležitá, protože písmena latinky mají silnou odezvu na filtry s frekvencemi s vodorovnou a svislou orientací (a vhodným měřítkem), ale reagují i na jiné úhly. Naopak třeba čáry reagují pouze na jeden úhel, jak je vidět na obrázku 3.1.



(a) Originál (b) 0 (c) 0.25π (d) 0.5π (e) 0.75π

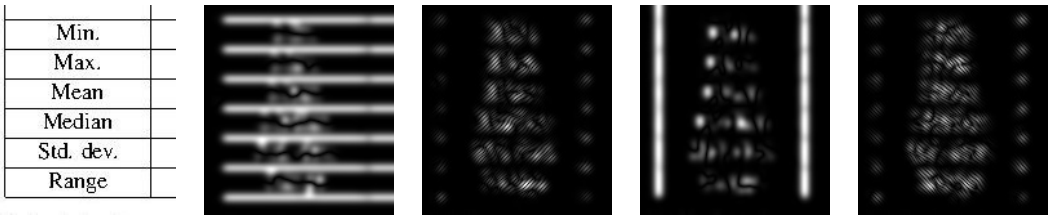
Obrázek 3.1: Ukázka výstupů Gáborových filtrů pro výřez z tabulky. Filtry mají periodu rovnu 2 pixelům, mění se jen úhel.

Odfiltrovat fotografie je nejsnazší – obsahují většinou tak nízké frekvence, že je mnou nastavené fitry vůbec nezachytí a vidí jen pozadí. Případný náhlý přechod sice může být zachycen, ale většinou je klasifikován jako netext či později odfiltrován postprocessingem.

V prvních verzích projektu jsem používal pouze reálnou část Gaborova filtru, což se však ukázalo jako zcela nedostačující (viz obrázek 3.2). Proto nyní využívám absolutní hodnotu vypočteného komplexního čísla (vlastně se jedná o dva filtry, po konvoluci s nimiž získám matice reálných, resp. imaginárních složek, ze kterých pak získám absolutní hodnotu).

3.1.3 Shrnutí parametrů

Gaborovy filtry ve tvaru, ve kterém jsou použity v mé práci, mají parametry uvedené v tabulce 3.1:



(a) Originál (b) 0 (c) 0.25π (d) 0.5π (e) 0.75π

Obrázek 3.2: Ukázka použití pouze reálné části Gaborových filtrů - zvláště na (c) a (e) je jasně patrné zvlnění

S	Velikost jádra
σ	Směrodatná odchylka obálky
F_0	Perioda sinusoidy
ω_0	Směr sinusoidy
P	Posunutí

Tabulka 3.1: Parametry Gaborových filtrů

Vynechán je parametr K obálky, který určuje pouze míru vychýlení a pro účely rozpoznávání je zbytečný.

Naopak přibyl parametr S. Ačkoli pro Gaussovu funkci platí, že

$$w_r(x, y) > 0 \text{ pro } x, y \in (-\infty, \infty), \tag{3.10}$$

bylo by náročné implementovat filtr, který by energii každého bodu počítal ze všech bodů obrázku. Používá se proto okno – čtvercová matice o straně S pixelů, do které se uloží hodnoty tak, aby odpovídaly hodnotám jádra filtru, pokud aktuálně zpracovávaný bod leží uprostřed této matice (je proto vhodné, aby S bylo liché číslo). Na koeficienty ležící mimo okno se pak pohlíží, jako by byly nulové, a do výpočtu se nezahrnují.

Vhodná volba S je důležitá, protože zde proti sobě stojí dva vlivy: Větší S znamená přesnější, ale náročnější výpočet a naopak.

Volba všech těchto parametrů je podrobněji diskutována v kapitole 4.2.3.

3.2 Klasifikace bodů

Po určení hodnot různých příznaků pro každý bod vznikne vektor hodnot. Jeho součástí je lokální energie z Gaborových filtrů, ale je možno použít i souřadnice pixelu v obrázku nebo barvu či jas. Podle vektoru příznaků pak klasifikujeme pixel do daných skupin.

V průběhu tvorby bakalářské práce jsem vyzkoušel dva možné způsoby klasifikace.

3.2.1 K-Means

Nejprve jsem experimentoval s jednoduchým shlukováním, jak navrhuje [8]. Jedním z nejznámějších shlukovacích algoritmů je **K-Means** (využíván i v [2]). Jeho vstupem je kromě sady bodů (zde příznakové vektory jednotlivých pixelů) také číslo K, které udává, kolik skupin má být identifikováno.

Algoritmus je pak následující:

1. Pro každou skupinu se zvolí počáteční bod. Může se využít nějaké heuristiky nebo například zvolit jako středy K náhodně vybraných vstupních bodů.

2. Každý vstupní bod se přiřadí do skupiny, k jejímuž středu má nejkratší eukleidovskou vzdálenost.

3. Střed každé skupiny se přepočítá jako průměr souřadnic bodů, které skupina obsahuje.

4. Body 2 a 3 se opakují tak dlouho, dokud dochází ke změnám přiřazení bodů do skupin.

Aplikováno na problém segmentace dokumentu, podle [2] stačí hledat 3 skupiny, které ve výsledku odpovídají pozadí, textu a netextovým regionům.

Přestože algoritmus skutečně dokázal nalézt většinu textu a naopak ignorovat ostatní objekty, výsledky nebyly příliš přesvědčivé.

Problémem se ukázal fakt, že nebylo možno klasifikátor natrénovat, uložit a spouštět opakovaně na různé vstupy obrázků, nýbrž musel se celý provádět při klasifikaci každého nového obrázku.

Postup v článku [2] navrhuje zavést váhu každému filtru a tím ovlivnit jeho důležitost při procesu shlukování, avšak touto cestou jsem se již nevydal. Algoritmus totiž vyžaduje, aby ve vstupu byly všechny hledané skupiny přítomny. V kontextu segmentace dokumentů to znamená, že pokud bych se pokusil zpracovat např. stránku z knihy, jednotvárně popsanou jedním typem písma, neobsahující žádné vnořené obrázky či jiné netextové objekty, K-Means by stále hledal tři kategorie.

Z tohoto důvodu jsem se rozhodl pro sofistikovanější způsob klasifikace.

3.2.2 Umělé neuronové sítě

Umělé neuronové sítě, anglicky Artificial Neural Networks (ANN), se při klasifikaci snaží napodobit fungování mozku živých organismů.

Podobně jako mozek i neuronová síť se skládá z neuronů, které jsou vzájemně propojeny. Modelů, jak jednotlivé neurony propojit, je mnoho, v této práci jsem použil schéma vícevrstvé neuronové sítě (MLP - multilayer perceptrons).

Formálně se pokoušíme aproximovat funkci $\mathbb{R}^x \rightarrow \mathbb{R}^y$, kde x, resp. y je počet vstupů, resp. výstupů sítě.

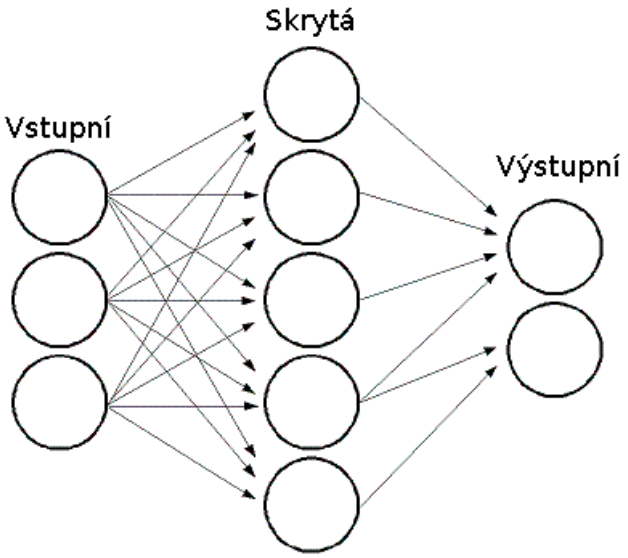
Vícevrstvá neuronová síť – MLP

V tomto modelu se síť skládá z několika vrstev neuronů: jedné vstupní vrstvy, kam je přivedený vstupní vektor (v našem případě vektor lokálních energií daného bodu), jedné výstupní vrstvy, kde získáme výsledek, a alespoň jedné skryté vnitřní vrstvy (viz obrázek 3.3).

Jednotlivé neurony jsou spojené tak, aby vstupem každého neuronu v každé vrstvě kromě první byly výstupy všech neuronů v předchozí vrstvě.

V některých případech je vhodné, že výstupem celé sítě je opět vektor čísel (stejně jako vstupem). Jindy, a to je i případ této práce, je potřeba získat kategorizovaný výstup – bod spadá právě do jedné z předem definovaných kategorií. Naskýtá se problém, jak tuto kategorickou informaci převést na vektor čísel.

Podle literatury [1] je nejlepším řešením vytvořit pro každou kategorii samostatný výstupní neuron, který nabývá hodnoty 1, pokud daný bod má spadat do dané kategorie, a



Obrázek 3.3: Příklad vícevrstvé neuronové sítě (převzato z [1]). Zde se jedná o příklad řídké sítě, kdy mezi druhou a třetí vrstvou nejsou všechna spojení. V mém případě jsou vždy všechny neurony sousedních vrstev spojeny.

0, pokud do ní spadat nemá. Při predikci pak získáme řadu čísel z intervalu $\langle 0, 1 \rangle$, ze kterých získáme konečný výsledek. V této práci bodu přiřadím jednoduše kategorii s nejvyšším ohodnocením.

Neuron

Obecně mohou být v ANN přítomny různé neurony, avšak v případě mnou použité implementace MLP jsou všechny stejného typu. Schéma je na obrázku 3.4. Každý z neuronů má několik vstupů a ke každému vstupu je přiřazena odpovídající váha, získaná trénováním. Kromě toho zde figuruje zvláštní „bias" vstup, na který je připojena konstanta.

Všechny váhované vstupy jsou sečteny a výstup neuronu se ze sumy získá pomocí aktivační funkce.

Formálně je tedy neuron definován takto:

$$u_i = \sum_j \left(w_{i,j}^{n+1} \cdot x_j \right) + w_{i,\text{bias}}^{n+1} \tag{3.11}$$

$$y_i = f(u_i) \tag{3.12}$$

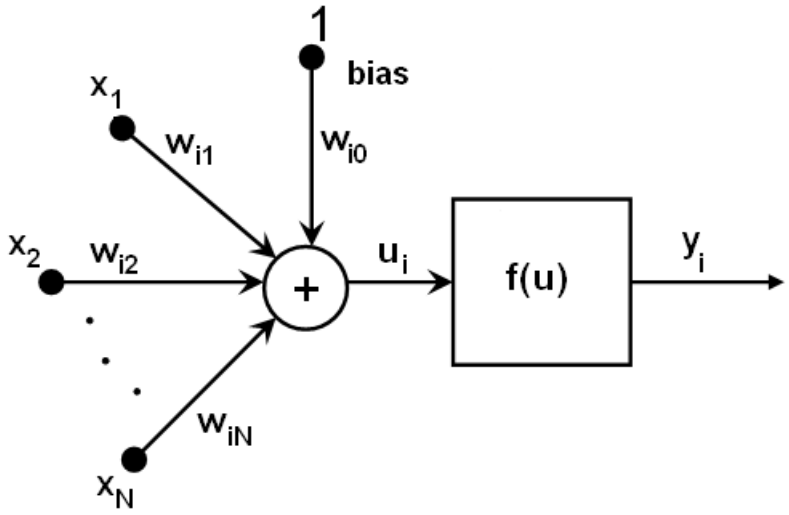
Aktivační funkce je nejčastěji sigmoida, kterou jsem použil i já. Je definovaná takto:

$$f(x) = \beta \cdot \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{3.13}$$

Získání klasifikátoru

Jak vyplývá z předchozího textu, umělá neuronová síť je definována dvěma faktory

- Topologií – Sem spadá počet vrstev a počet neuronů v každé z nich, stejně tak jako použité aktivační funkce a jejich parametry



Obrázek 3.4: Model neuronu (převzato z [1])

- Váhami vstupů neuronů

Univerzální pravidlo pro určení správné topologie neexistuje. U MLP je dán počet neuronů první vrstvy (rovná se počtu vstupů) a poslední vrstvy (počet výstupů, zde kategorií), avšak kolik má být skrytých vrstev a po kolika neuronech, je třeba zkusit. Obecně platí, že větší síť znamená přesnější aproximaci, ale delší trénování a je také potřeba mít dostatek trénovacích dat.

Váhy vstupů neuronů se naproti tomu získávají procesem zvaným trénování. Trénování vyžaduje mít předem připravenou sadu trénovacích dat, tedy vektory vstupů a k nim příslušných výstupů, které se síť naučí a dokáže pak predikovat výstupy na základě vstupů. Všechny trénovací algoritmy postupují iteračně, buď po jednotlivých vzorcích z testovací sady, nebo po dávkách.

Back Propagation

Základním algoritmem na trénování MLP je Back Propagation [9]. Jedná se o nejstarší algoritmus, ze kterého další algoritmy vycházejí a vylepšují ho.

Algoritmus můžeme popsat takto:

1. Inicializace vah neuronů náhodně
2. Přivedení vzorku z testovací sady na vstup
3. Vyhodnocení výstupu každého neuronu:
   (a) Váhovaná suma všech vstupů a biasu
   (b) Výpočet aktivační funkce
4. Postupně získáme výstupy celé sítě
5. Pro každý výstupní neuron vypočítáme chybu δ jako rozdíl získaného a očekávaného výstupu

6. Chybu propagujeme zpětně do sítě od výstupních po vstupní neurony. Postupujeme stejně jako v bodu 3.a, pouze v obráceném směru. Využíváme stejné váhy. Pro každý neuron tak získáme ochylku $\delta_{i,j}$.

7. Novou váhu pro j-tý vstup i-tého neuronu pak zjistíme podle tohoto vzorce:

$$w'_{i,j} = w_{i,j} + \eta\delta_{i,j}x_j, \tag{3.14}$$

   kde w je původní váha, w' je nová váha, $\delta_{i,j}$ je chyba získaná v předchozím postupu, x_j je hodnota příslušného vstupu a parametr η je konstanta zadaná před samotným učením, ovlivňující rychlost učení.

8. Ověříme koncovou podmínku. Dokud nekončíme, vracíme se na bod 2.

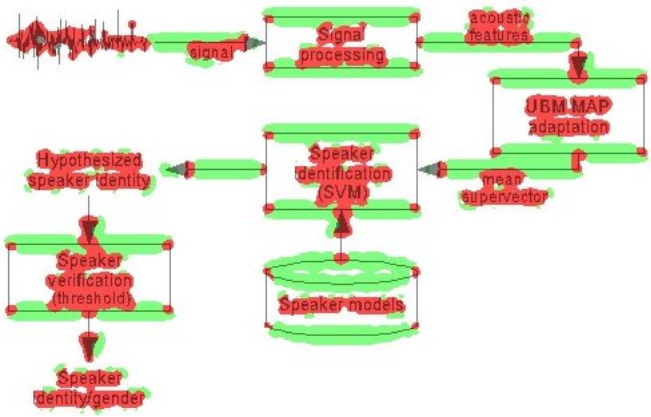
Je skoro jisté, že bude potřeba projít celou trénovací sadu vícekrát. Jedno projití nazýváme iterace, a po ukončení jedné iterace před započetím další je možné provést dílčí vyhodnocení procesu.

Implementace, kterou používám, umožňuje zadat koncovou podmínku dvěma způsoby: Maximální počet iterací a maximální chyba. Ta je určena jako $\sum \delta_i^2$ pro všechny neurony.

3.3 Vyhodnocení výsledků, postprocessing

Teoreticky by výše uvedený postup měl pro segmentaci postačovat. Každý pixel je klasifikován do jedné ze tří kategorií, takže lze snadno pomocí masky ze vstupního obrázku získat dokument, který obsahuje například pouze text.

Při provádění experimentů ale vyšlo najevo, že takto získaný výstup trpí některými systémovými vadami. Například konce úzkých čar a rohy fotografií jsou chybně označeny jako text (viz obrázek 3.5), protože mají s textem podobné charakteristiky.



Obrázek 3.5: Rohy a konce čar jsou chybně rozpoznány jako text

Při vyhodnocení primární metriky se tento nedostatek téměř neprojeví – jedná se jen o několik málo pixelů, množství zcela zanedbatelné vzhledem k jejich celkovému počtu.

Mnohem fatálnější jsou důsledky při pokusu o rozpoznání výsledného obrázku pomocí OCR. Vymaskované tvary mají velikost podobnou okolnímu textu a v rozpoznaném dokumentu se pak objevuje mnoho nežádoucích znaků typu plus, minus či malého r.

3.3.1 Hledání souvislých komponent

Jako řešení navrhuji použít postup inspirovaný algoritmem popsaným v kapitole 2.2.4. Stejně jako tam se nejprve detekují souvislé komponenty v obrázku a ty, jejichž obálka je příliš velká, jsou automaticky považovány za netext. U zbylých je porovnán počet textových a netextových pixelů (podle klasifikace získané předchozími kroky) a celá komponenta pak získá tu klasifikaci, která mezi pixely převládá. Pixely souvislé černé komponenty, které jsou přesto chybně klasifikovány jako pozadí, se zde považují za netextové.

Tato metoda přinesla subjektivně obrovské zlepšení výsledků, což potvrdily i primární i sekundární metrika. Je totiž přirozené, že souvislá komponenta je tvořena stejným druhem pixelů.

Přináší však také nové problémy. Zatímco Gaborovy filtry pracují se vstupními obrázky převedenými na stupně šedi, algoritmus hledání souvislých komponent vyžaduje binární obrázek. Je tedy nutné zavést prahování a určit vhodný práh. V této práci je zvolen konstantní práh s poměrně vysokou hraniční hodnotou, takže je preferována černá barva před bílou. Tento postup funguje dobře na vědecké články tištěné černé na bílém.

3.4 Klasifikace textových segmentů

Pomocí postupu popsaného v předchozím textu je již možné vymaskovat z například naskenovaného dokumentu text tak, aby mohl být dále rozpoznán pomocí OCR. V této práci jsem ale chtěl zamířit ještě dál, a to zjistit alespoň základní charakteristiky textových segmentů bez použití výstupů z OCR. Podle literatury (např. [4]) je možné pomocí Gaborových filtrů detekovat i řez písma a font, ale to již přesahuje zaměření této práce.

Zde tedy provádím pouze detekci řádků, že kterých pak zjišťuji počet řádků v odstavci a průměrnou velikost fontu.

3.4.1 Detekce odstavců a řádků

Z předchozích kroků můžu zjistit jednotlivé znaky či části znaků (např. diakritika) textu v obrázku. Prvním úkolem je spojit je do větších celků, v mém případě do odstavců. K tomu používám metodu zvanou eroze. Jedná se v podstatě o filtr, který každému bodu přiřadí minimum z jeho vhodně definovaného okolí. Vzhledem k tomu, že černé pixely mají hodnotu 0 a bílé hodnotu větší, dojde k roztažení a slití černých ploch. Výsledky jsou podobné jako v algoritmu CRL popsaném v kapitole 2.2.2, pouze prostředky k jeho dosažení jsou jiné.

Na výsledné plochy opět aplikuji algoritmus hledání souvislých komponent, abych našel masky jednotlivých odstavců.

Z každého odstavce spočítám vertikální projekci (viz 2.2.3), podle které již můžu počet a velikost řádků zjistit. Zda je řada s určitým počtem černých pixelů součástí řádku či prostoru mezi nimi, se určuje opět pomocí prahu, který je definován poměrně k maximu v projekci. Tato metoda není 100% spolehlivá, u kratších textů se někdy delší řádek rozdělí na dva, naopak u odstavců s více řádky zarovnanými do bloku, který končí neúplným řádkem, se tento poslední krátký řádek někdy nerozpozná, ale určitou představu to je schopno poskytnout.

3.4.2 Velikost fontu

Velikost fontu je spočítána jako průměr výšek detekovaných řádků.

Obrázek 3.6: Velikosti písma v typografii (převzato z [11])

Problémem je, že v typografii existuje velikostí několik a není jak zjistit, která byla detekována, viz obrázek 3.6. Ve většině případů je detekována tzv. střední výška, což odpovídá výšce malého x. U nadpisů psaných verzálkami je však detekována velikost kuželky a ve zvláštních případech, kdy by například řádek obsahoval mnoho znaků, které přesahují pod řádek, by mohl být výsledek úplně jiný. Detekovaná velikost je tak spíše orientační.

Kapitola 4

Aplikace postupů

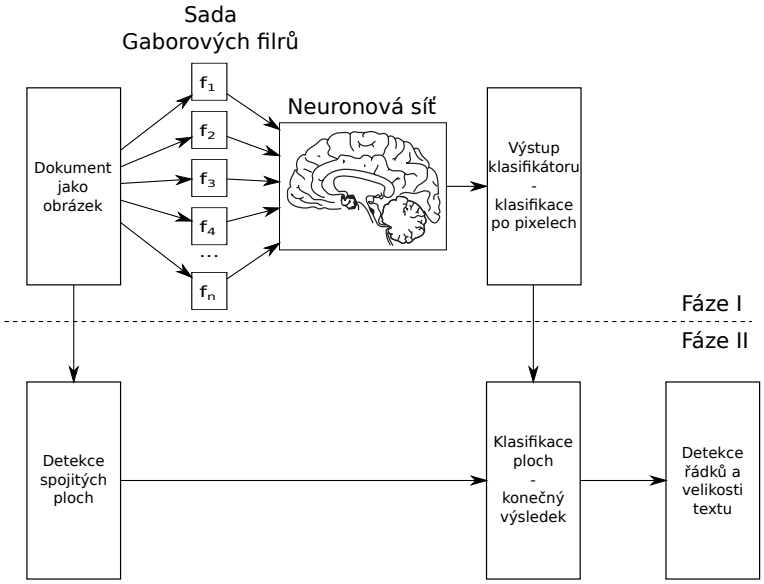
V této kapitole již popíši konkrétní využití zde představených algoritmů. Kapitola je řazena chronologicky tak, jak na sebe jednotlivé kroky navazují.

Program je implementován v jazyce C++ za použití knihovny OpenCV[1] ([1], [3]).

4.1 Schéma systému

Systém jsem rozdělil na dvě fáze: V první fázi se klasifikuje každý pixel zvlášť podle neuronové sítě. Druhá fáze obsahuje postprocessing s detekcí spojitých ploch a rozpoznávání řádků a odstavců textu.

Schéma systému je na obrázku 4.1



Obrázek 4.1: Schéma systému

[1] http://opencv.willowgarage.com/

4.2 Výběr filtrů

Jak bylo předesláno v kapitole 3.1.3, Gaborovy filtry mají nejméně 5 volitelných parametrů. Ačkoli obor hodnot každého parametru se liší, jeho mohutnost je vždy neomezená. Je tedy třeba nějak vybrat takovou sadu či sady parametrů, které nejlépe poslouží účelu separace jednotlivých tříd.

4.2.1 Metrika úspěšnosti filtru

Nejprve je třeba číselně ohodnotit úspěšnost jednotlivých filtrů při rozdělení pixelů do tříd. Uvažujeme-li jen dvě třídy (text a netext), poslouží nám následující funkce [2]:

$$J(w) = \frac{|m_1 - m_2|^2}{s_1^2 + s_2^2}, \tag{4.1}$$

kde m_1 a m_2 jsou aritmetické průměry lokálních energií bodů patřících do jednotlivých tříd a s_1 a s_2 jejich směrodatné odchylky. Čím vyšší je hodnota této funkce, tím lépe daný filtr odděluje od sebe dané dvě třídy.

Pozadí zde záměrně není uvažováno. Ve všech mých trénovacích datech bylo pozadí bílé, tedy jeho detekce je mnohem jednodušší než odlišení textu od ostatních objektů. Předpokládám tedy, že vybraná sada filtrů, která je úspěšná při oddělení textu od netextu, dokáže oddělit i pozadí.

4.2.2 Stanovení rozsahu parametrů

Rozsah některých parametrů bylo třeba zjistit experimentálně opakovanou evaluací filtrů s různým nastavením. Některé parametry je však možné určit konstantní nebo v závislosti na jiných parametrech.

Posunutí Je výhodné, aby funkce, která generuje jádro filtru, byla sudá. Gaborův filtr totiž rozostří obraz podél svého směru. Pro účely klasifikace je mnohem vhodnější, když ono rozostření je centrováno kolem středu filtru. Posunutí tedy stanovíme vždy nulové.

Sigma Směrodatná odchylka obálkové Gaussovy funkce vlastně určuje, kolik period či jak velká část periody sinusoidy bude uvažována. Abych ji mohl ukládat v celých číslech a přitom zachoval rozumný krok, uvádím ji v násobcích periody.

Velikost jádra Velikost jádra souvisí s parametrem σ. V mém případě je stanovené na $4\sigma + 1$, čímž je zároveň zajištěno, že velikost bude lichá.

Perioda Perioda se zadává v pixelech a přímo udává, na jaké velikosti objektů bude filtr reagovat.

Směr Vzhledem k periodicitě funkce kosinus dává smysl zadávat hodnoty z intervalu $\langle 0, 2\pi)$. Navíc, jelikož se jedná o funkci sudou, stačí interval $\langle 0, \pi)$. Vnitřně je tento parametr uchováván v násobcích π, avšak při volbě správných filtrů je vhodné definiční obor rozdělit rovnoměrně. Proto se algoritmu zadává jako parametr počet kroků, ze kterých se pak podělením získá délka kroku a posléze úhel otočení.

4.2.3 Postup výběru filtrů

Celý postup provedeme na předem vybrané sadě obrázků. Je možné použít stejnou sadu, které bude použita i pro trénování neuronové sítě.

Postup je následující:

1. Vygenerujeme množinu filtrů vzniknou kartézským součinem předem daných rozsahů parametrů.
2. Každý z takto získaných filtrů aplikujeme na každý obrázek ze sady a spočítáme úspěšnost podle funkce J (vzorec 4.1).
3. Úspěšnosti každého filtru na všech obrázcích sečteme.
4. Filtry, které mají periodu a otočení stejné a liší se pouze σ a velikostí jádra, nepřinášejí žádnou informaci navíc. Proto z těchto filtrů vybereme pouze ten nejlepší a ostatní ze sady vyřadíme.
5. Výslednou sadu seřadíme podle úspěšnosti a uložíme.

4.3 Trénování neuronové sítě

K vlastnímu trénování používám algoritmus RPROP (resilient backpropagation), který je vylepšenou verzí Back Propagation. Implementaci jsem využil z knihovny OpenCV.

Bohužel tato implementace trpí několika nevýhodami:

- Vyžaduje, aby celá trénovací sada byla najednou nahraná v paměti. Vzhledem k tomu, že počet bodů jednoho obrázku trénovací sady se počítá na miliony, znamenalo to, že velikost sady je značně omezená.
- Jedná se o černou skříňku. Lze nastavit parametry trénovacího algoritmu, ale nelze do jeho průběhu nijak zasáhnout, například kvůli průběžné validaci či tisku aktuální chyby.

Z těchto důvodů jsem z algoritmu v OpenCV použil vždy jen jednu iteraci a učící cyklus jsem naprogramoval vlastní. Naštěstí trénování lze v OpenCV spustit i opakovaně, aniž by se stav sítě pokaždé znovu náhodně inicializoval.

Trénovací smyčka je následující:

1. Načtení rovnoměrné dávky z disku
2. Zamíchání dávky
3. Trénování pomocí jedné iterace OpenCV implementace RPROP algoritmu
4. Výpočet chyby na části trénovací sady
5. Výpočet chyby na validační sadě
6. Vyhodnocení ukončovací podmínky, případně návrat na začátek

4.3.1 Načítání rovnoměrné dávky

První pokusy s trénováním načítaly body jeden po druhém tak, jak se nacházely v obrázcích trénovací sady.

Výsledky byly překvapivé a evidentně nesmyslné: Přibližně 94% všech bodů bylo přiřazeno do správné kategorie a zároveň 100% bodů bylo přiřazeno do kategorie pozadí.

Toto zjištění mne přivedlo jednak ke změně metriky (body, které mají být pozadí, se do výpočtu nezahrnují) a jednak k vyrovnání počtu bodů jednotlivých kategorií v jedné dávce.

Důsledkem tohoto nového požadavku je fakt, že nebudou v jedné iteraci použity všechny body (přesněji řečeno budou použity všechny body právě té kategorie, která je zastoupena nejmenším počtem bodů).

Je tedy třeba použité body nějakým způsobem vybírat. Požadavky na tento způsob byly, aby každý bod stejné kategorie měl stejnou šanci, že bude vybrán. Pravděpodobnost, že kterýkoli z n prvků bude vybrán do sady o velikosti k tedy musí být

$$P = \frac{k}{n}. \tag{4.2}$$

Toho se podařilo dosáhnout tímto postupem:

1. Začneme s prázdnou sadou.
2. Načteme i-tý vstupní bod, číslováno od 0.
3. Máme již kompletní sadu – $i \geq k$?
   (a) Ne: Vložíme na pozici i.
   (b) Ano: Vygenerujeme náhodné číslo $x = \text{Uniform}(0, i)$.
      i. $x < k$? Pokud ano, přepíšeme pozici x.
4. Opakujeme pro všechny prvky na vstupu.

Například matematickou indukcí se dá dokázat, že pro každý bod platí pravděpodobnost z rovnice 4.2.

Každá dávka obsahuje vlastně 3 rovnoměrně zastoupené složky, jednu pro každou kategorii, získané způsobem popsaným výše. Před vlastním tréninkem je celá dávka zamíchána. Jinak by totiž síť při učení jedné části dávky pravděpodobně zapomněla na ostatní části.

Chtěl bych na tomto místě upozornit na fakt, že náhodný výběr bodů pro dávku, stejně tak jako zamíchání dávky se děje v každé iteraci, tedy během tréninku. Může se tedy stát, je to skoro jisté hlavně u bodů pozadí, kterých je zdaleka nejvíce, že každá trénovací dávka obsahuje úplně jiné body, potencionálně s jinými výstupy filtrů. Tento fakt může při trénování způsobovat drobné odchylky v míře průběžné chybovosti od teoretického průběhu.

4.3.2 Velikost dávky a počet dávek

Velikost dávky je volitelná a je to jeden z parametrů, který určuje rychlost učení. Počet dávek se získá z počtu prvků ve třídě, která je zastoupena nejméně.

Jednotlivé kategorie jsou načítány každá v samostatném streamu nezávisle na sobě a počet bodů každé kategorie, které jsou načteny pro získání jedné dávky, je dán poměrně vůči celkovému počtu bodů dané kategorie.

Celkově je ale pro trénování použito jen 90% trénovacích dat. Zbylých 10% slouží pro cross validaci po natrénování každé iterace.

4.3.3 Výpočet chyby a ukončovací podmínka

Po každé iteraci jsou vypočteny dvě metriky úspěšnosti:

- Úspěšnost trénovací sady – počítá se ze zbývajících 10% trénovacích dat. Předpokládá se, že trénovací data jsou rozložena rovnoměrně, takže úspěšnost spočítanou pro část můžeme vztáhnout na celou sadu. Využívá implicitní metriku (5.1.1) a jejím účelem je prokázat, že síť se skutečně učí. Její hodnota by tedy měla růst s přibývajícími iteracemi.
- Úspěšnost na validační sadě – druhá metrika se počítá z kompletně jiné validační sady a i jiným způsobem, jedná se o primární metriku (5.1.2). Z důvodu odlišného výpočtu o jejím průběhu nemáme žádné apriorní informace, pouze předpokládáme, že pro malý počet iterací poroste a pro počet iterací jdoucí k nekonečnu bude opět klesat. Z experimentů plyne, že často má více než jedno lokální maximum (více viz kapitola 5.5).

Ukončovací podmínka bere v potaz úspěšnost na validační sadě. Protože však podle experimentů nemá tato funkce předvídatelný tvar, nemůžu např. ukončit výpočet v bodě, kdy poprvé začne klesat. Proto je ukončovací podmínka definována takto: Výpočet se ukončí po 10 iteracích od posledního maxima. Jako výsledek trénování je pak považován stav sítě v iteraci s maximální úspěšností (tedy nikoli v té poslední).

4.4 Formát dat

Vstupní obrazová data očekává program ve formátu JPEG, ve stejném formátu jsou i výstupy. Informace o kategoriích jednotlivých pixelů (ať získaných z již zadaných vstupů či vypočtených klasifikátorem) jsou ukládány ve zvláštních souborech ve formátu PNG ve stupních šedi tak, že hodnota barvy pixelu odpovídá jeho kategorii podle tabulky 4.1:

0	Pozadí
1	Netext
2	Text

Tabulka 4.1: Kategorie v souboru

Matice vstupů pro klasifikátor, tedy vektory lokálních energií získaných zvolenou sadou filtrů pro každý bod obrázku, je potřeba při trénování načítat proudově. Proto nemůžou být uloženy v komprimované podobě ve formě obrázku, jako jsou uložena jiná data. Místo toho je jejich formát přímý otisk paměťové reprezentace této matice v OpenCV. Každý soubor obsahuje binární hlavičku, definující typ a rozměry matice, a dále jsou po řádcích uloženy jednotlivé hodnoty.

Protože datové typy v OpenCV mají definovánu velikost v bitech, je tento soubor přenositelný mezi 32 a 64 bitovými variantami architektury x86. Nešel by však přenést na architekturu, která používá kódování Big Endian.

Výsledné soubory jsou uloženy s koncovkou .bin.

Textové informace (parametry filtrů, natrénovaný klasifikátor a informace o textových segmentech) jsou ukládány ve formátu YAML.

Kapitola 5

Trénování, experimenty a jejich výsledky

V této kapitole popíši metodiku testování svého projektu a zhodnotím jeho výsledky.

5.1 Metriky úspěšnosti

Jelikož algoritmy použité v této práci mají mnoho volitelných nastavení, je potřeba je nastavit pro získání co nejlepšího výsledku. Kvalitu výsledku je třeba umět objektivně kvantifikovat. K tomuto účelu se používají vhodné metriky.

5.1.1 Implicitní metrika

Je definována takto:

$$\text{Implicitní metrika} = \frac{\text{Počet správně rozpoznaných pixelů}}{\text{Počet pixelů celkem}} \tag{5.1}$$

Všechny pixely jsou posuzovány stejně bez ohledu na jejich očekávané či zjištěné zařazení. Protože však počet pixelů jednotlivých typů segmentů se výrazně liší a stejně tak se liší jejich důležitost, není výsledek implicitní metriky příliš vypovídající. Tato metrika se tak používá pouze vnitřně při trénování neuronové sítě.

5.1.2 Primární metrika

Cílem práce je především odlišit od sebe textové a netextové objekty. V článcích ve vědeckých časopisech obvykle bývá černý (tmavý) text na bílém (světlém) pozadí. Odlišení pozadí je zde triviálním úkolem. Maskování spočívá v nahrazení pixelů jiné než textové, resp. netextové kategorie barvou pozadí. Je-li tedy pixel, který má být součástí pozadí, rozpoznán chybně a přiřazen k textu či netextu, po maskování to nepůsobí žádné obtíže, protože na jeho místě v každém případě zůstane pozadí.

Navíc experimenty ukázaly, že bodů pozadí je v dokumentech řádově mnohem více než pixelů jiných kategorií. Proto výsledkem implicitní metriky mohou být velmi vysoká čísla (90% a více), i když klasifikátor vůbec dobře nefunguje.

Proto je navržena primární metrika, která body pozadí nezapočítává. Je definována takto:

$$\text{Primární metrika} = \frac{\text{Počet správně rozpoznaných černých pixelů}}{\text{Počet černých pixelů celkem}} \tag{5.2}$$

5.1.3 Sekundární metrika

Cílem projektu je správně detekovat text pro jeho následné rozpoznání pomocí OCR programu. Sekundární metrika se snaží vyčíslit přesnost rozpoznání.

Pro účely vypočítání sekundární metriky je třeba mít dokument v textové formě, který je převeden na obraz, segmentován a znovu rozpoznán. Sekundární metrika je pak definována takto:

$$\text{Chybovost podle sekundární metriky} = \frac{L(\text{orig}, \text{ocr})}{count(\text{orig})}, \tag{5.3}$$

kde $L(\text{orig}, \text{ocr})$ značí Levenshteinovu vzdálenost mezi původním a rozpoznaným textem a $count(\text{orig})$ je počet znaků původního textu.

Levenshteinova vzdálenost určuje minimální počet znaků, které musíme vložit, odebrat nebo nahradit (jeden za jeden), abychom z jednoho řetězce získali řetězec jiný.[6]

Na rozdíl od předchozích metrik zde platí čím méně, tím lépe.

Sekundární metriku je třeba vyhodnotit dvakrát: jednou bez použití mého projektu a jednou s jeho využitím pro vymaskování textu. Porovnáním získaných čísel lze pak vyhodnotit zlepšení (či zhoršení).

Je třeba si dát pozor na fakt, že po odstranění netextových objektů může selhat analyzátor toku textu v OCR programu, čímž může dojít k záměně některých bloků textu. Je tedy vhodné provádět tento test na dokumentech s jednoduchým (nejlépe jednosloupcovým) rozvržením, kde je riziko záměny menší.

5.2 Datové sady

Jako zdroj testovacích a validačních dat posloužil webový archiv článků vědeckého časopisu Radioengineering.[1]

Jako trénovací sadu jsem použil vybrané stránky, kde se vyskytují jak různé řezy a velikosti písma, tak tabulky, grafy schémata, fotografie a další druhy netextových objektů.

Validační sadu tvoří 5 jiných, náhodně vybraných stránek z celého archivu.

5.3 Příprava dat

Aby bylo možno články použít pro trénování a validaci, bylo potřeba je jednak převést do obrázku, ale navíc ke každému pixelu zjistit, do jaké kategorie (pozadí, netext, text) náleží.

Časopis Radioengineering publikuje své články na webu ve formátu PDF. Ačkoli text je v tomto formátu uložen skutečně jako text a vektorové objekty dokonce jako vektor, je obtížné se v binárním formátu PDF zorientovat a tuto informaci z něj vypreparovat.

Proto byly soubory PDF nejprve převedeny do formátu PostScript pomocí příkazu `pdf2ps`. Tento je již textový, a dokonce všechny z mých převedených souborů obsahovaly v úvodu jakousi hlavičku s definicemi maker (pravděpodobně kvůli úspoře místa), mimo jiné i těch pro kreslení vektorů a vložených bitmap. Pomocí skriptu šlo tedy vymazat těla

[1] http://www.radioeng.cz

	Velikost jádra (px)	Sigma (px)	Perioda (px)	Úhel (rad)	Posun (px)
1.	61	15	3	0	0
2.	41	10	2	0	0
3.	101	25	5	$5/12\pi$	0
4.	101	25	5	$7/12\pi$	0
5.	81	20	4	$5/12\pi$	0
6.	41	10	5	$8/12\pi$	0
7.	33	8	4	$8/12\pi$	0
8.	81	20	4	$7/12\pi$	0
9.	41	10	2	$2/12\pi$	0
10.	101	25	5	0	0
11.	41	10	5	$4/12\pi$	0
12.	121	30	6	$7/12\pi$	0
13.	65	16	4	$9/12\pi$	0
14.	49	12	6	$8/12\pi$	0
15.	33	8	4	$4/12\pi$	0
16.	101	25	5	$6/12\pi$	0
17.	41	10	2	$3/12\pi$	0
18.	61	15	3	$3/12\pi$	0
19.	121	30	6	$5/12\pi$	0
20.	49	12	4	0	0

Tabulka 5.1: Sada filtrů použitá pro trénování

těchto maker, čímž lze snadno získat dokument s identickým rozložením jako originál, ale bez netextových prvků.

Makra, která se starají o kreslení vektorů a jejichž těla (ale ne definice) je třeba vymazat, jsou tyto: `/m, /l, /c, /re, /h, /S, /Sf, /f, /f*, /W, /W*, /Ws`

Dále je třeba upravit funkci pro zobrazení bitmapy, která je zakódována přímo v souboru. Tělo této funkce je třeba změnit na:

```
/pdfIm {
  { currentfile pdfImBuf readline
    not { pop exit } if
    (%-EOD-) eq { exit } if } loop
} def}
```

Tím se zajistí, že data, která v souboru zůstanou, jsou korektně načtena, ale nic se s nimi neprovede.

Výsledné soubory jsem pak převedl do formátu JPEG v rozlišení 144 ppi.

5.4 Použité filtry

Sadu filtrů jsem získal postupem popsaným v kapitole 4.2.3. Použil jsem trénovací sadu a rozsahy parametrů Gaborových filtrů podle tabulky 5.2.

Výsledkem byla sada filtrů na tabulce 5.1. Je seřazena podle získaného skóre od nejlepších k nejhorším a cituji zde pouze prvních 20 filtrů, které budou použity dále.

Parametr	Rozsah
Směrodatná odchylka obálky (σ)	2 až 5
Velikost jádra	$4\sigma + 1$
Perioda sinusoidy	2 až 8
Směr sinusoidy	krok $1/12\pi$
Posunutí	vždy 0

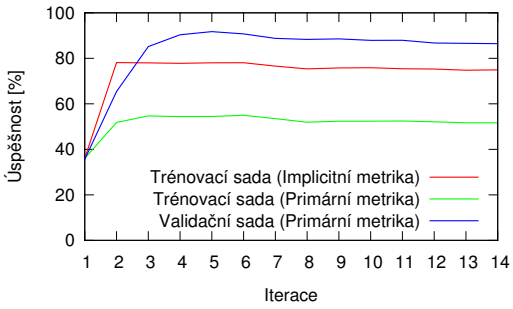
Tabulka 5.2: Použité rozsahy parametrů Gaborových filtrů

5.5 Parametry trénování a jejich vliv

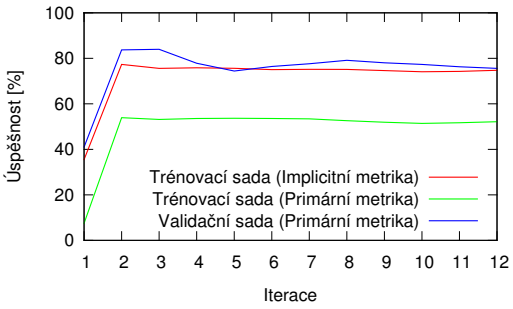
Počet vstupů a velikost střední vrstvy má přímý dopad na úspěšnost trénování.

5.5.1 Počet vstupů a velikost střední vrstvy

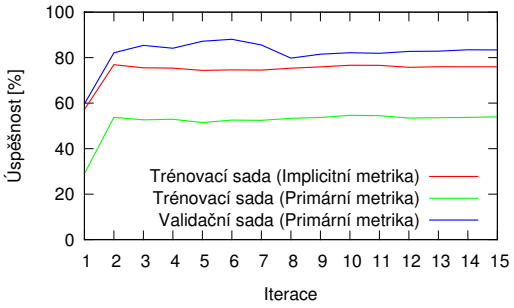
Na grafech 5.1, 5.2 a 5.3 jsou znázorněny průběhy trénování pro různé počty vstupních filtrů a velikost střední vrstvy je udána jako jedno-, dvou- a trojnásobek tohoto počtu. Při tomto trénování byla použita velikost dávky 1000 prvků.



(a) 5 filtrů a 5 neuronů



(b) 5 filtrů a 10 neuronů

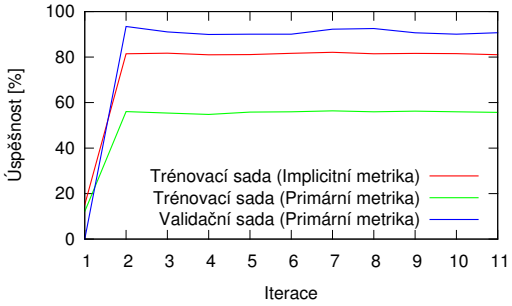


(c) 5 filtrů a 15 neuronů

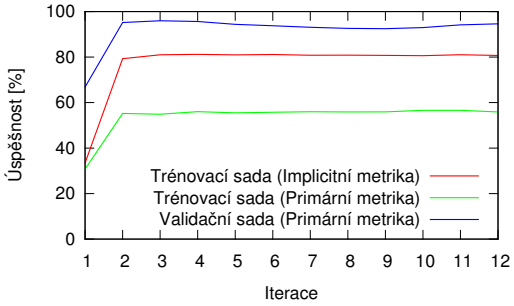
Obrázek 5.1: Průběh trénování s nejlepšími 5 filtry a různým počtem neuronů střední vrstvy

Uvedené grafy pocházejí vždy z jednoho trénování na každý graf. Vzhledem k metodice výběru dat pro jednotlivé dávky (viz kapitola 4.3.1) a také kvůli tomu, že síť je inicializována náhodně, vypadá každé trénování mírně odlišně.

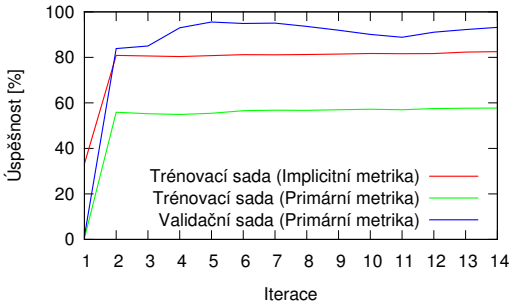
Z grafů je také patrné, že úspěšnost na trénovací sadě vždy neroste, jak by podle teorie měla (i když nikdy ani výrazně neklesá). Tento fakt může být zapříčiněn dvěma vlivy:



(a) pro 10 filtrů a 10 neuronů



(b) 10 filtrů a 20 neuronů



(c) 10 filtrů a 30 neuronů

Obrázek 5.2: Průběh trénování s nejlepšími 10 filtry a různým počtem neuronů střední vrstvy

- Dávky v každé iteraci nejsou stejné, jak bylo popsáno výše.
- Cross-validační sada není reprezentativním vzorkem sady trénovací. Pixely jednotlivých kategorií jsou čteny sekvenčně a teprve v rámci dávky jsou zamíchány.

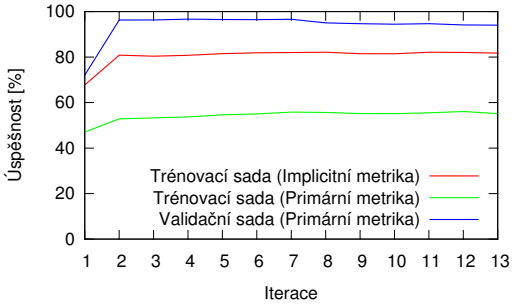
  Pokud by například poslední z deseti souborů (tedy posledních 10%) byl výrazně odlišný, byl by i výsledek cross-validace neplatný. V mém případě si byly obrázky velmi podobné, avšak i drobné rozdíly mohly způsobit malé nepřesnosti, které vidíme v grafech.

Je také vidět, že na všech grafech je síť natrénovaná velmi brzy. Na trénovací sadě je citelný nárůst pouze mezi první a druhou iterací, pak se již mění především poměr mezi jednotlivými kategoriemi, což způsobuje změny ve validační, ne však v testovací sadě. Tento trend se nezměnil ani při přidání velkého množství neuronů střední vrstvy (zkoušel jsem i řádově stovky), ani při změně velikosti trénovací dávky. Mohu z toho tedy vyvozovat pouze tolik, že algoritmus RPROP konverguje velmi rychle. Je také pravděpodobné, že ještě lepších výsledků by se dalo dosáhnout s větší trénovací sadou.

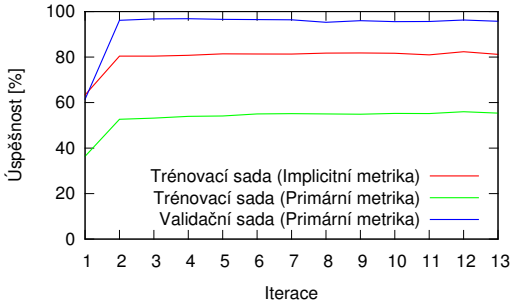
Tři nejlepší konfigurace shrnuje tabulka 5.3.

5.5.2 Parametry klasifikátoru textu

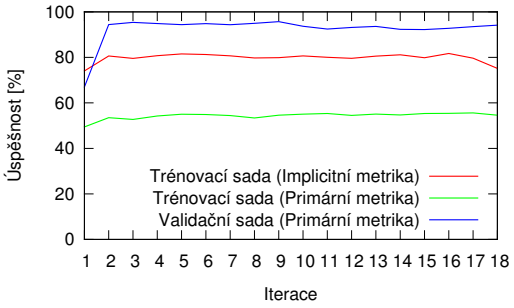
Do parametrů textového klasifikátoru patří především hodnota prahu pro převedení šedého obrázku do binární podoby a hraniční hodnota ve vertikální projekci, která od sebe rozděluje řádky.



(a) 20 filtrů a 20 neuronů



(b) 20 filtrů a 40 neuronů



(c) 20 filtrů a 60 neuronů

Obrázek 5.3: Průběh trénování s nejlepšími 20 filtry a různým počtem neuronů střední vrstvy

Graf	Filtry	Stř. vrstva	Úspěšnost
5.3(b)	20	40	96, 87%
5.3(a)	20	20	96, 68%
5.2(b)	10	20	95, 96%

Tabulka 5.3: Nejlepší konfigurace při trénování

Vzhledem k tomu, že tato práce je cílena na vědecké články, které jsou přímo tištěny černo-bíle, není hodnota prahu příliš podstatná. Je nastavena přímo v programu na hodnotu 220 (maximum – bílá barva – je 255).

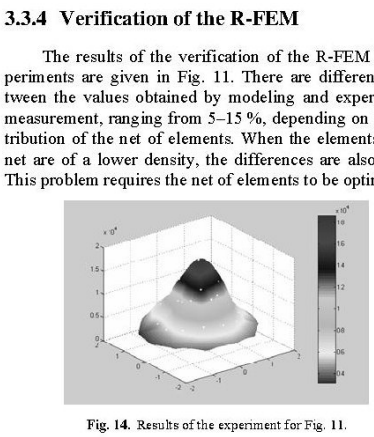
Prahová hodnota pro oddělení řádků byla stanovena na jednu desetinu z maxima vertikální projekce, nejméně však 3 pixely. Tento postup vychází z ručního zkoušení různých hodnot.

5.6 Výsledky

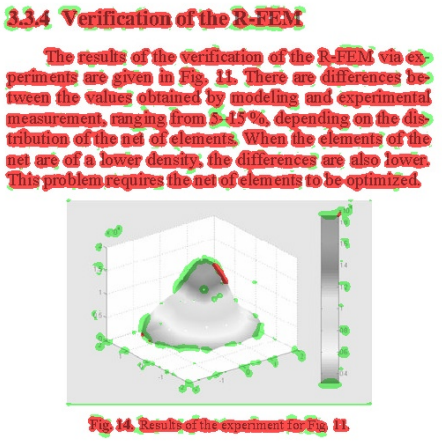
Následující obrázky ukazují vždy originál, kategorizaci pixelů pomocí samostatné neuronové sítě, výstup postprocessingu a vizualizaci výstupu klasifikátoru textu.

5.6.1 Vstupy z validační sady

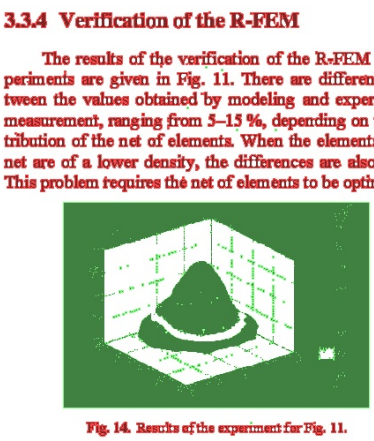
Nejprve předvedu činnost klasifikátoru na části validační sady (obrázky 5.4 a v příloze B.1 až B.4).



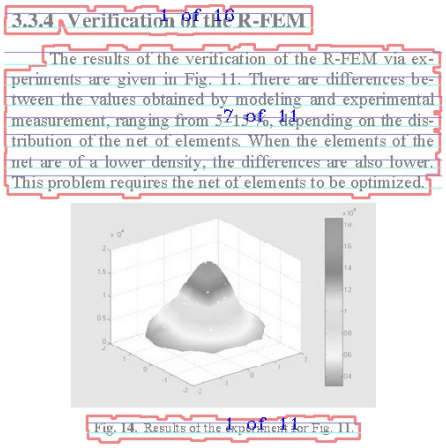
(a) Originál



(b) Klasifikováno neuronovou sítí



(c) Konečný výsledek



(d) Textový klasifikátor

Obrázek 5.4: Výsledky klasifikátoru

Všechny výsledné obrázky v příloze B mají tuto strukturu: Na první obrázku je originální dokument. Na druhém je výstup z neuronové sítě. Červenou barvou jsou vyznačeny pixely textu, zelenou netextových objektů a pixely pozadí nejsou vybarveny nijak. Třetí obrázek ukazuje výstup druhé fáze klasifikátoru, kdy jsou posuzovány spojité černé plochy (barevný kód zůstává stejný). Poslední obrázek vizualizuje výstup textového klasifikátoru: Červeně jsou ohraničeny jednotlivé bloky textu, uprostřed nichž je modrou barvou napsáno, kolik obsahují řádků a jaká je průměrná velikost textu. Horní a dolní hranice detekovaných řádků jsou navíc vyznačeny modrými a azurovými linkami.

Je patrné, že s většinou textu si klasifikátor poradil bezchybně. Problémy dělají některé znaky, například pomlčka nebo znak rovná se, které jsou klasifikovány jako netext (připomínají totiž čáru), stejně tak jako některé části vzorců. Naopak správně je klasifikována tabulka a graf, ve kterém jsou správně rozpoznány dokonce některé vnořené texty.

Na druhém obrázku je vidět, že klasifikátor není příliš robustní vůči velikosti textu – v nadpisu se některé znaky nerozpoznají, týká se to hlavně těch znaků, které obsahují delší svislé části.

Postprocessing na primární metriku už nemá výrazný vliv (zlepšení je přibližně 1%),

avšak je nezbytný pro úspěch metriky sekundární.

5.6.2 Experimenty s jinými vstupy

Zajímalo mne, jak si klasifikátor natrénovaný na vědeckých článcích z jednoho časopisu poradí s jinými druhy vstupů.

Prvním příkladem je reklamní leták, který je designově odlišný od vědeckých článků, avšak stále zachovává formát A4.

Asi největším rozdílem je dominantní černá barva pozadí, na které je vytištěn světlý text. Neuronová síť v obrázku i tak poměrně spolehlivě našla text. Selhal však algoritmus hledání souvislých komponent, proto jsem musel zavést podmínku, že v případě nadpolovičního počtu černých pixelů v prahovaném vstupu se vstup invertuje. Výsledek ze zobrazen na B.5.

Jako druhý pokus jsem použil slidy z prezentace. Výsledky jsou na obrázcích B.6 a B.7. Zde byl největší problém velikost písma, které je obrovské ve srovnání s písmem v dokumentech určených k tisku. Pro získání výsledků zde prezentovaných jsem tak musel slidy převádět do formátu JPEG s rozlišením nikoli 144 ppi jako u ostatních dokumentů, ale pouhých 50 ppi.

Klasifikátor není tedy robustní vzhledem k velikosti použitého písma, zatímco ani složitější layout či barevné složení (pokud je dost kontrastní) mu problém nedělá. Pro slidy by tedy bylo nutné natrénovat klasifikátor, který by využíval Gaborovy filtry s delší periodou.

5.7 OCR test

Účelem tohoto testu je ohodnotit, jak se detekce a vymaskování textových segmentů projeví v celém procesu optického rozpoznávání znaků, jak je popsáno v kapitole 5.1.3. Jako OCR nástroj byl vybrán volně dostupný program Tesseract verze 3.02.[2]

Test jsem prováděl na desetistránkovém vědeckém článku, který obsahoval mnoho rovnic, tabulek a obrázků. Původně dvousloupcový layout jsem převedl na jednosloupcový, abych se vyvaroval chyb způsobených špatnou detekcí pořadí odstavců.

Výsledky jsou shrnuty v tabulce 5.4.

Velikost originálu ve znacích	26307
OCR bez maskování	
Levenshteinova vzdálenost	4473
Chybovost	17%
OCR s maskováním	
Levenshteinova vzdálenost	4366
Chybovost	12%

Tabulka 5.4: Výsledky OCR testu

OCR převod byl spuštěn v obou případech s implicitním nastavením, což mimo jiné znamená, že Tesseract prováděl segmentaci i sám vnitřně. Přesto jsem naměřil zlepšení převodu o 5%.

[2] http://code.google.com/p/tesseract-ocr/

Kapitola 6

Závěr, možnosti vylepšení

Ve své práci jsem se věnoval rozdělení dokumentu zadaného jako bitmapa do segmentů, které mohu označit jako pozadí, text a jiné objekty. V úvodních kapitolách jsem popsal několik vybraných postupů segmentace dokumentů, které jsem nalezl v literatuře.

Navrhl jsem klasifikátor, který využívá sady Gaborových filtrů a neuronové sítě k rozřazení pixelů vstupního obrázku. Projekt jsem implementoval v jazyce C++ za použití knihovny OpenCV, ale protože implementace trénování neuronových sítí v této knihovně neodpovídala mým požadavkům, musel jsem vytvořit vlastní postup.

Na vědeckých článcích, na které byla tato práce především cílena, jsem dosáhl úspěšnosti více než 96% správně určených černých bodů.

Ukázalo se však, že není vhodné použít tuto metodu jako jedinou.

Proto jsem zařadil ještě jednu fázi, která je založena na detekci souvislých černých ploch ve vstupním obrázku, kdy celá plocha je zařazena do stejné kategorie. Tato fáze ještě vylepšila přesnost klasifikátoru. Byl také proveden test, jak vymaskování textu ovlivní výsledky OCR programu Tesseract, a ukázalo se, že výsledky jsou o 5% lepší, než když necháme segmentaci jen na tomto programu samotném.

Ačkoli klasifikátor byl natrénován na vědeckých článcích, ukázalo se, že může být využit poměrně úspěšně i na složitějších dokumentech, které dokonce nejsou tištěné černé na bílém. Důležitá je především podobná velikost textu. Klasifikátor, tak jak je natrénován, je velmi citlivý na velikost textu, proto nešel přímo využit na slidy určené k promítání, kde je text mnohem větší než v tištěných dokumentech. Určitě by ale bylo možné natrénovat zde popsanými postupy klasifikátor určený přímo na slidy.

Při použití větší trénovací sady, více filtrů a větší neuronové sítě by se mohlo dosáhnout větší robustnosti klasifikátoru vůči velikosti písma, ovšem za cenu delšího rozpoznávání a trénování.

Celý program by také bylo možné optimalizovat, zvláště druhá fáze (detekce spojitých ploch) poskytuje prostor pro zrychlení. Zde by se dalo dosáhnout lepších výsledků použitím například adaptivního prahování pro získání binárního obrazu, či větší rychlosti, kdyby se použil jiný algoritmus pro segmentaci obrazu podle spojitých ploch.

Také by bylo možné tuto fázi předřadit výpočtu hodnot filtrů a ty získávat pouze pro pixely, o kterých už víme, že nejsou pozadí. Je zde tedy dostatek prostoru pro další modifikace a experimenty.

Tato práce byla řešena samostatně bez návaznosti na jiné projekty řešené v tomto roce.

Literatura

[1] OpenCV Documentation. [online]. Cit. 1. 5. 2012. Dostupné z WWW: http://opencv.itseez.com/

[2] Asirvatham, A. P.: *Script Segmentation of Multi-Script Documents*. Semestrální projekt, International Institute of Information Technology, Gachibowli, India, 2002.

[3] Bradski, G.; Kaehler, A.: *Learning OpenCV*. Sebastopol: O'Reilly Media, první vydání, 2008, ISBN 9780596516130.

[4] Doermann, D.: Gabor filter based multi-class classifier for scanned document images. *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, 2003: s. 968–972, doi:10.1109/ICDAR.2003.1227803.

[5] Fletcher, L.; Kasturi, R.: A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 10, č. 6, 1988: s. 910–918, ISSN 01628828, doi:10.1109/34.9112.

[6] Gilleland, M.: Levenshtein Distance. [online]. Cit. 1. 5. 2012. Dostupné z WWW: http://www.merriampark.com/ld.htm

[7] Ha, J.; Haralick, R.; Phillips, I.: Recursive XY cut using bounding boxes of connected components. *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR'95)*, 1995: s. 952–955, doi:10.1109/ICDAR.1995.602059.

[8] Jain, A.; Bhattacharjee, S.: Text segmentation using Gabor filters for automatic document processing. *Machine Vision and Applications*, ročník 5, č. 3, 1992: s. 169–184, ISSN 14321769.

[9] Jamborová, S.: *Segmentace obrazu pomocí neuronové sítě*. Diplomová práce, FIT VUT v Brně, 2011.

[10] Movellan, J.: Tutorial on Gabor filters. 2002. [online]. Cit. 15. 1. 2012. Dostupné z WWW: mplab.ucsd.edu/tutorials/gabor.pdf

[11] Ott, V.: Základní typografické pojmy: Co byste měli vědět o písmu. 2010. [online]. Cit. 1. 5. 2012. Dostupné z WWW: http://www.scribus.cz/zakladni-typograficke-pojmy-co-byste-meli-vedet-o-pismu/

[12] Zramdini, A.; Ingold, R.: Optical font recognition from projection profiles. *Electronic Publishing*, ročník 6, č. 3, 1993: s. 249–260, ISSN 0894-3982.

Příloha A

Obsah CD

Přiložené CD obsahuje zdrojové kódy programu vytvořeného v rámci práce a elektronickou verzi této technické zprávy, včetně zdrojových souborů.

Jsou zde také trénovací a validační obrázky, stejně tak jako další obrázky, na kterých lze činnost klasifikátoru vyzkoušet. Ke každému z nich jsou vygenerovány všechny mezikroky i konečné výsledky, kromě surových vstupů pro trénování neuronové sítě (ty mají několik GB a na CD by se nevešly). Před vlastním trénováním je třeba je nejdříve vygenerovat nebo použít přiložený skript, který automatizuje celý proces.

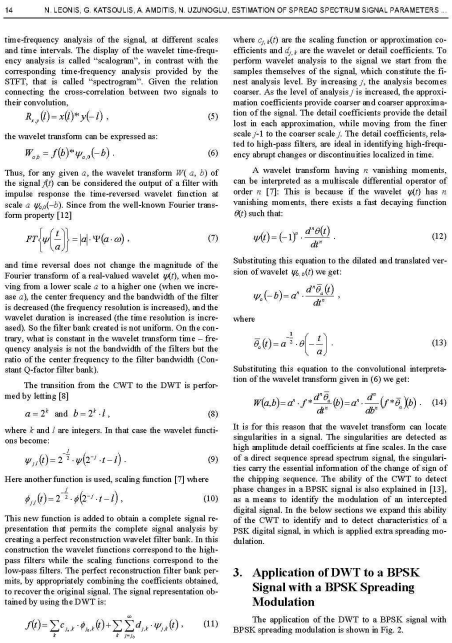
CD také obsahuje nastavení filtrů a neuronových sítí, které vykazovaly nejlepší výsledky.

Podrobnější informace o obsahu CD lze nalézt v souboru **readme.txt** v kořenovém adresáři média.

Příloha B

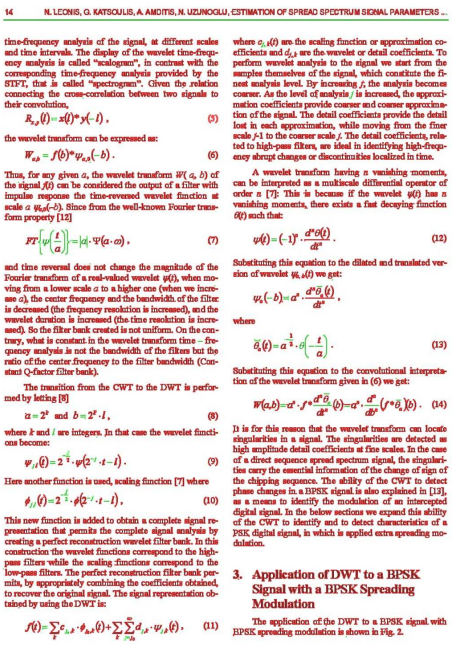
Experimenty

Zde uvádím další výsledky ze vstupů validační sady. Kompletní rozpoznaná validační sada je k dispozici na přiloženém CD.



(a) Originál

(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext)



(c) Konečný výsledek (červená – text, zelená – netext)

(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.1: Validační obrázek 1

Dual Band a-Si:H Solar-Slot Antenna for 2.4/5.2GHz WLAN Applications

SHYNU S.V [1], Maria J. ROO ONS [1], Max J. AMMANN [1], Brian NORTON [2]

[1] Antenna & High Freq. Research, School of Electronic and Communications Engg., Dublin Inst. of Technology, Dublin-8,
[2] Dublin Energy Lab, Focas Institute, Dublin-8, Ireland

max.ammann@dit.ie

Abstract. *A simple and compact design of solar-slot antenna for dual band 2.4/5.2GHz wireless local area networks (WLAN) applications is proposed. The design employs amorphous silicon (a-Si:H) solar cells in polyimide substrate with an embedded twin strip slot structure to generate dual resonant frequencies. A T-shaped microstripline feed is used to excite the twin slot in the a-Si:H solar cell. The measured impedance bandwidths for the proposed solar antenna are 25.9% (642 MHz) centered at 2.482 GHz and 8.2% (420 MHz) centered at 5.098 GHz. The measured gain at 2.4 and 5.2 GHz are 3.1 dBi and 2.1 dBi respectively.*

Keywords

Solar antenna, slot antenna, amorphous silicon.

1. Introduction

The idea of integration of photovoltaic solar cells with microwave antennas offers a wide range of advantages in terms of surface coverage, volume, mass, cost and electric performance when compared with a simple juxtaposition of antennas and solar cells. Recently, communication systems integrated with photovoltaic technology for low cost and stand alone applications received much interest [1-5]. The photovoltaic systems of power generation when combined with communications systems can provide compact and reliable autonomous communication systems for many applications.

In most of the reported attempts of integration of solar cells with printed antennas, commercial solar cells are glued or placed next to the radiating patch or in the ground plane of slot antennas [6]. Other combinations like placing the solar cells behind the reflectarray antennas have also been studied [7]. Successive development of an amorphous-Si cell on a flexible thin film polymer substrate realized an improved photovoltaic performance at lower cost. Here a higher level of integration was then made possible by integrating amorphous silicon solar cells with microstrip slot antennas [8]. The slot antennas were selected in order to minimize the effect of solar cells on the RF performance of the antenna. But this choice of slot antenna introduced drawbacks such as narrow bandwidth and poor circular polarization performance. Complex laser cutting of the solar cells is required to achieve desired shapes of slots. This makes the design of dual-frequency and multi-frequency solar slot antennas difficult, as complex slot structures in solar cells are hard to engrave. Moreover, the large dimension of the slot structures will degrade the solar cell efficiency. Printed slot antennas are widely studied for WLAN operation [9, 10]. The key to provide a flexible design is to make the slot in the solar cell as small as possible with a basic geometric shape to excite the dual resonant modes.

In the present approach, flexible amorphous silicon solar cells are used to design a compact dual band microwave slot antenna operating at 2.4/5.2GHz WLAN application. The proposed design consists of an amorphous silicon solar cell in polyimide substrate where a twin strip embedded rectangular slot is imprinted at the centre of the solar cell. The performance of the proposed solar antenna is optimized using a finite integral equation based electromagnetic simulator. Details of the proposed solar-slot antenna design are described, and experimental results for the dual broadband performance are presented and discussed.

2. Solar-Slot Antenna Design

The photograph of the proposed a-SiH solar-slot antenna design for 2.4/5.2GHz WLAN application is shown in Fig. 1. To realize total integration of the photovoltaic solar cell and the antenna, amorphous silicon solar cells of dimension 72.58 x 68.76 mm were used as the ground plane for the microstrip slot antenna. The a-SiH solar cell consists of a p-i-n silicon layer of thickness 0.39 μm and ε_r = 11.7 sandwiched between two zinc oxide (ZnO) layers of thickness 1.5 μm. An aluminum layer of thickness 1 μm acts as the back contact. The transparent and conductive ZnO layer (1.5 μm) on the top acts as the collector. Finger patterns with silver forms (Ag-bus bars) the top layer of the cell. The bottom and top layers of polyimide (ε_r = 3.4, tanδ = 0.0018) and silver electrodes are 50 μm and 0.7 μm respectively.

A rectangular shaped slot of length L=30 mm and width W =17.7 mm is located on the solar cell at its centre as shown in Fig. 2(a). Two rectangular PEC strips of

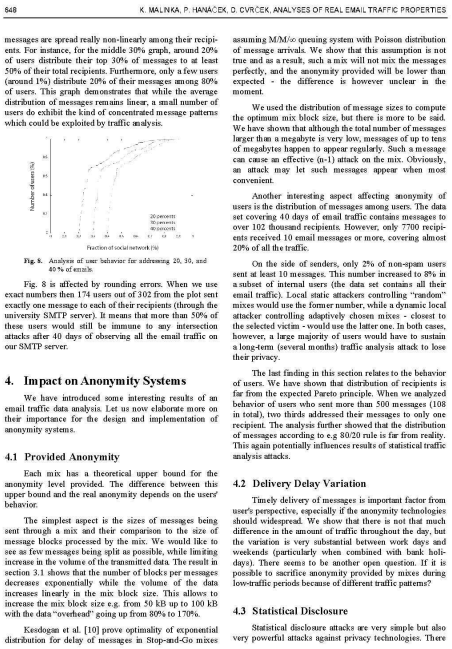
(a) Originál

(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext)

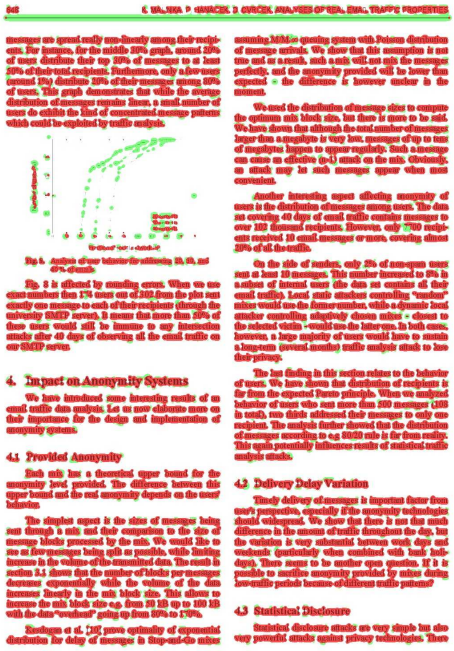
(c) Konečný výsledek (červená – text, zelená – netext)

(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

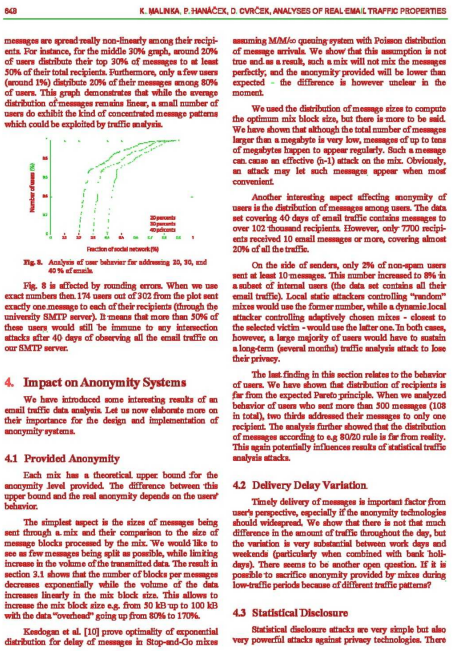
Obrázek B.2: Validační obrázek 2



(a) Originál



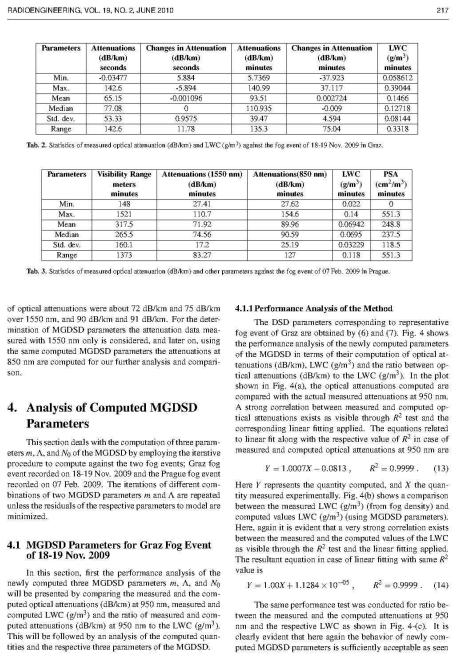
(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext)



(c) Konečný výsledek (červená – text, zelená – netext)

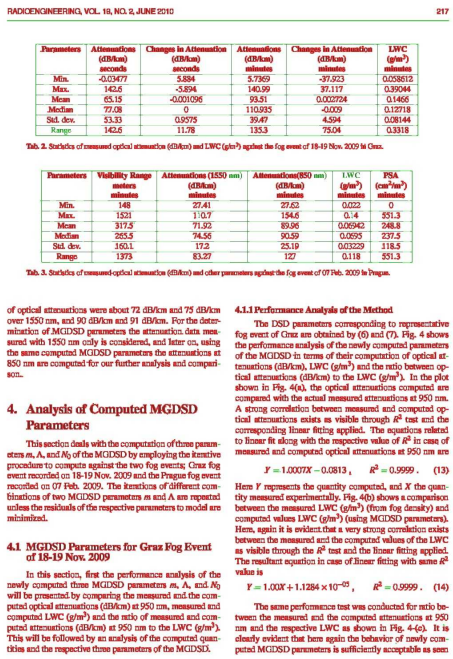
(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.3: Validační obrázek 3

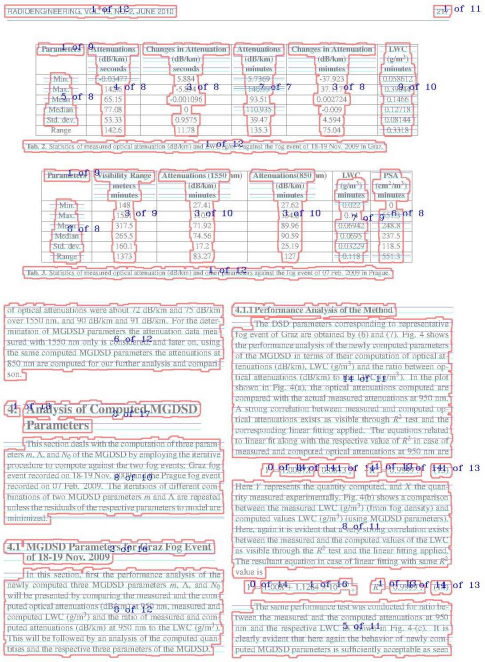


(a) Originál

(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext)



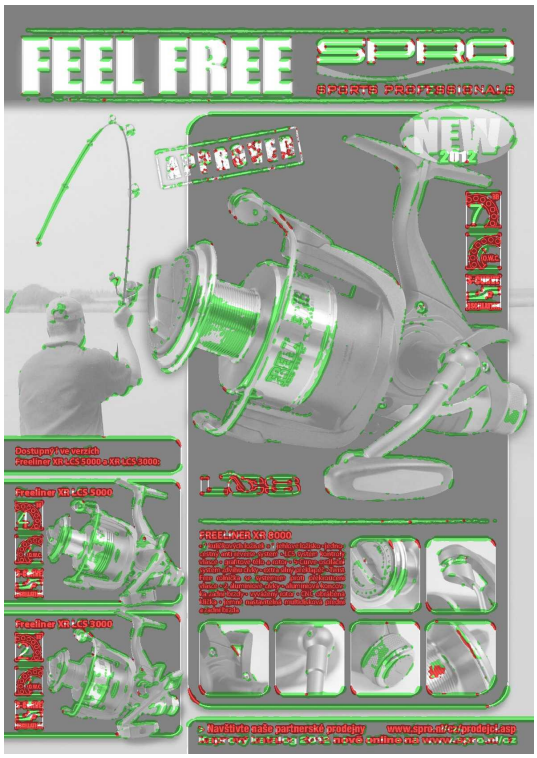
(c) Konečný výsledek (červená – text, zelená – netext)



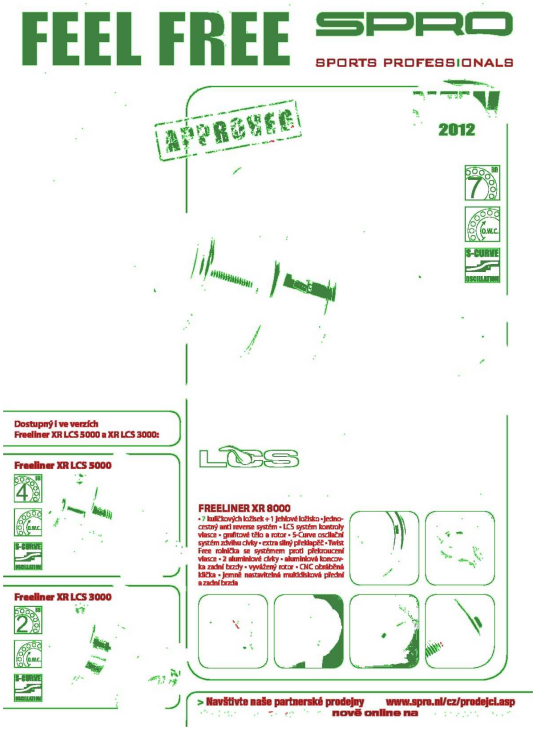
(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.4: Validační obrázek 4

(a) Originál



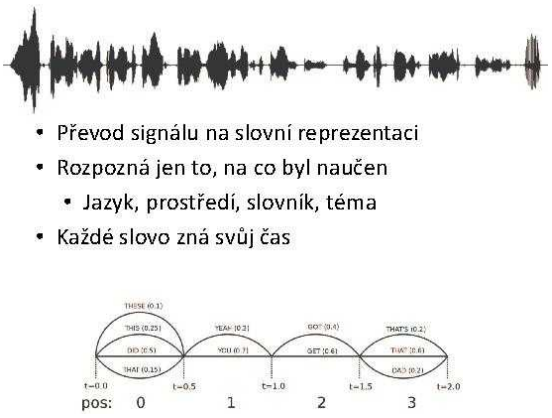
(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext)



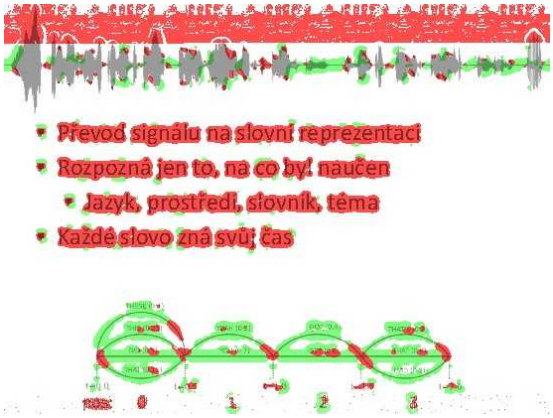
(c) Konečný výsledek (červená – text, zelená – netext)

(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.5: Experiment: reklamní leták



(a) Originál



(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext) – červený obdélník vznikl pravděpodobně vlivem JPG komprese

(c) Konečný výsledek (červená – text, zelená – netext)

(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.6: Experiment: slide 1

(a) Originál

(b) Klasifikováno neuronovou sítí (červená – text, zelená – netext) – červený obdélník vznikl pravděpodobně vlivem JPG komprese

(c) Konečný výsledek (červená – text, zelená – netext)

(d) Textový klasifikátor (červeně orámované bloky textu, modře napsaný počet řádků a velikost textu, světle modrou vyznačeny řádky)

Obrázek B.7: Experiment: slide 2

Příloha C

Popis implementace

Jednotlivé části jsou navrženy s ohledem na co největší znovupoužitelnost a za použití objektového paradigmatu. Protože tato práce má celou řadu fází, je jejím výstupem také řada jednoúčelových programů pro příkazový řádek.

C.1 Třídy programu

Zde popíši nejdůležitější třídy použité v mém programu.

Gabor Třída, která se stará o vlastní filtrování obrázku Gaborovým filtrem o zadaných parametrech. Instance si uchovávají obálkovou část filtru (Gaussovu křivku) zvlášť a pokud je stejná, negenerují ji znovu. Proto je výhodné při filtrování více filtry seřadit tyto filtry tak, aby ty se stejnou obálkou šly po sobě.

ANN Obálka nad CvANN_MLP – třídou OpenCV pro umělé neuronové sítě. Oproti ní obsahuje metody pro trénování a predikci specifickou pro klasifikaci obrázků.

GaborClassifier Jedna z nejdůležitějších tříd celého projektu. Zde jsou implementovány algoritmy pro trénování neuronové sítě, jak je popsáno v kapitole 4.3, i pro klasifikaci.

TextClassifier Druhý v pořadí klasifikátorů, který provádí další analýzu obrazu podle souvislých komponent. Zde jsou implementovány algoritmy pro detekci odstavců a řádků, jak jsou popsány v kapitole 3.3.

ResourceManager Třída, která má za úkol nahrávání a ukládání veškerých obrazových i textových dat použitých v projetu podle adresářové struktury, jak je popsaná v kapitole C.2.

C.2 Adresářová struktura

Většina programů projektu vyžaduje jako jeden ze svých argumentů adresář. Tento adresář má pevně danou strukturu, pod kterou programy očekávají jednotlivé mezivýsledky zpracování souborů. Struktura je popsána v tabulce C.1.

Jména vlastních souborů mohou být libovolná, ale musejí mít danou koncovku a pochopitelně jména všech souborů vztahujících se k jednomu obrázku musejí být stejná. V tabulce je základní jméno souboru nahrazeno znaky *jméno*.

`full/`*jméno*`.jpg`	Původní nemodifikovaný vstupní obrázek, který slouží za vstup většině programů
`textOnly/`*jméno*`.jpg`	Vstupní obrázek s vymazanými netextovými objekty. Slouží k vytvoření souboru s kategoriemi.
jméno`.png`	Soubor obsahující kategorie každého pixelu. Slouží jako správné řešení při trénování, při výběru pixelů a evaluaci získaných rozdělení.
jméno`.resultANN.png`	Kategorie získané pomocí klasifikátoru s Gaborovými filtry a neuronovou sítí
jméno`.result.png`	Konečný výsledek získaný z předchozího pomocí analýzy spojitých komponent
jméno`.text.jpg`	Výstupní obrázek, který obsahuje jen pixely klasifikované jako text
filtry`_`*počet*`/`*jméno*`.bin`	Vstupní matice pro trénování, získaná z obrázku pomocí sady ze souboru *filtry*, ze kterého bylo použito *počet* nejlepších filtrů
filtry`_`*počet*`/filters.yml`	Kopie sady filtrů použité pro generování matic
jméno`.tags.yml`	Výstupní soubor, který obsahuje informace o každém textovém segmentu v obrázku.
jméno`.labelled.jpg`	Vizualizace předchozího

Tabulka C.1: Adresářová struktura