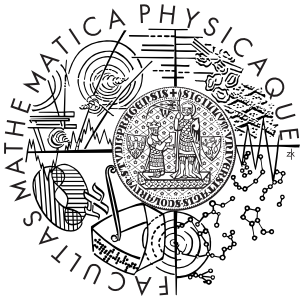
Univerzita Karlova v Prahe
Matematicko-fyzikálna fakulta

BAKALÁRSKA PRÁCA



Matej Klaučo

Rozpoznávanie textu

Katedra softwarového inžinierstva

Vedúci bakalárskej práce: Mgr. Jan Lánský
Študijný program: Informatika, Obecná informatika

2009

Ďakujem svojmu vedúcemu za to, že mi umožnil pracovať na tejto téme.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím s požičiavaním práce a jej zverejňováním.

V Prahe dňa 25. 5. 2009 Matej Klaučo

Obsah

Zoznam obrázkov

Názov práce: Rozpoznávanie textu
Autor: Matej Klaučo
Katedra (ústav): Katedra softwarového inžinierstva
Vedúci bakalárskej práce: Mgr. Jan Lánský
e-mail vedúceho: Jan.Lansky@mff.cuni.cz

Abstrakt: Cieľom tejto práce je implementovať prevod naskenovaného textového dokumentu do editovateľnej podoby. V práci sa venujeme podrobnej analýze tohto problému, jeho rozloženiu na menšie časti a ich riešeniu. Súčasťou práce je aj vytvorenie sady komplexných aplikácií s prepracovaným grafickým rozhraním, ktoré umožnia prispôsobenie samotného prevodu požiadavkám používateľa. Pri prevode sa zameriavame na obrázky, ktoré môžu obsahovať jemný šum spôsobený nekvalitou skeneru a môžu byť pootočené.

Kľúčové slová: rozpoznávanie textu, neurónová sieť, OCR, analýza členenia dokumentu

Title: Optical character recognition
Author: Matej Klaučo
Department: Department of Software Engineering
Supervisor: Mgr. Jan Lánský
Supervisor's e-mail address: Jan.Lansky@mff.cuni.cz

Abstract: The aim of this work is to implement a conversion from the scanned text document to the editable form. In this work we closely analyse this problem, its division into smaller parts and their solutions. An important part of the work is also a creation of a set of complex applications with a sophisticated graphical user interface, which allow accomodation of conversion to the user's requirements. During the conversion we specialize in images, which may contain insignificiant noise caused by a scanner of poor quality and which can be rotated.

Keywords: optical character recognition, neural net, document layout recognition

Kapitola 1

Úvod

1.1 Rozpoznávanie textu

Rozpoznávanie textu[1] je mechanický alebo elektronický preklad obrázkov, ktoré obsahujú text, do editovateľnej podoby.
Túto technológiu začala používať United States Postal Service na triedenie pošty už v roku 1965 a o šesť rokov neskôr začala používať OCR systémy aj Canada Post. Tieto systémy prečítajú meno a adresu adresáta v triediacom centre a vytlačia na obálku čiarový kód cesty založený na poštovom smerovacom čísle. Neskôr môžu byť dopisy triedené lacnejšími prístrojmi, ktorím stačí prečítať čiarový kód. V Európe prvýkrát použila rozpoznávanie textu British General Post Office, keď v roku 1965 začala pomocou OCR plánovať celý bankový systém známy ako National Giro. Tento proces vyústil vo Veľkej Británii do revolúcie platobných systémov.[1]

1.2 Súčasný stav

Presné rozpoznanie tlačenej latinskej abecedy je považované za vyriešený problém. Typická úspešnosť rozpoznania dosahuje 99%. Ostatné oblasti, vrátane rozpoznávania ručne písaného písma a tlačeného písma písaného v abecede s veľkým počtom znakov, sú stále predmetom aktívneho výskumu.

Musíme poznamenať, že úspešnosť rozpoznania je možné merať viacerými spôsobmi a spôsob, akým je meraná, môže veľmi zmeniť udávanú úspeš-

[1] tiež OCR - Optical Character Recognition

nosť. Napríklad, bez použitia slovníku na opravenie chýb, veľkosť chyby 1% (alebo 99% úspešnosť) meraná písmeno po písmene môže vyústiť do veľkosti chyby 5% alebo viac (alebo 95% úspešnosť), ak je meranie založené na správnosti celého slova.

Rozpoznávanie textu je niekedy zamieňané s on-line rozpoznávaním textu, hoci je inštanciou takzvaného off-line rozpoznávania textu, pri ktorom systém rozpoznáva statické tvary znaku, zatiaľ čo on-line rozpoznávanie využíva dynamické pohyby počas písania textu, napríklad môže využiť znalosť, či horizontálna čiara bola napísaná zľava doprava alebo sprava doľava. On-line rozpoznávanie je tiež známe ako dynamické rozpoznávanie textu, rozpoznávanie textu v reálnom čase alebo *Inteligentné rozpoznávanie textu*[2][1]

1.3 Cieľ

V našej práci sa zameriame na rozpoznávanie naskenovaných textových dokumentov s dostatočne veľkým rozlíšením. Optimálne rozlíšenie je 300dpi. Vzhľadom na spôsob implementácie, ktorý bude popísaný v ďalších kapitolách tejto práce, je vhodné, aby písmená v slovách neboli spojené so susednými písmenami. Pri súčasnej kvalite tlačiarní a skenerov to nepovažujeme za obmedzujúcu podmienku. Obrázky môžu obsahovať mierny šum pridaný nekvalitným skenerom a môžu obsahovať ľubovoľne pootočený text. Primárne predpokladáme tmavý text na svetlom podklade, keďže túto podmienku splňuje väčšina vytlačených dokumentov. Skenované dokumenty môžu obsahovať aj iné elementy ako text, napríklad obrázky alebo tabuľky, pričom rozpoznávaním pozície týchto elementov a v prípade tabuliek ich štruktúrou sa nebudeme v našej práci zaoberať. Text dokumentu môže obsahovať stĺpce, pričom veľkosť zanorenia je neobmedzená. Naša práca sa zaoberá aj rekonštrukciou tejto štruktúry textu. Implementácia procesu rozpoznávania bude premietnutá do sady komplexných aplikácií, ktoré umožnia používateľovi rozpoznávať dokumenty s čo najmenšou interakciou programu s používateľom. Na druhú stranu, tieto programy budú naprogramované tak, aby umožnili, v prípade potreby, prispôsobenie procesu rozpoznávania požiadavkám používateľa.

[2]Intelligent Character Recognition - ICR

1.4 Postup práce

Celý proces konverzie obrázka na text je vo všeobecnosti možné rozdeliť na niekoľko fáz, pričom každá fáza je pre dosiahnutie akceptovateľného výsledku kľúčová:

- predspracovanie obrázku do podoby vhodnej na rozpoznávanie textu. Táto fáza v našej práci zahŕňa konverziu obrázka do odtieňov šedej, odstránenie šumu a v prípade potreby zvýraznenie písma. Fáze predspracovania sa venujeme v Kapitole 2

- rozpoznanie štruktúry dokumentu, t.j. rozpoznanie pozície písmen v dokumente, priradenie diakritiky k príslušným písmenám a vytvorenie riadkov textu a odstavcov. V tejto časti tiež určíme správne poradie odstavcov. Tejto fáze sa venuje Kapitola 3

- samotné rozpoznanie písmen a oprava chýb po dokončení procesu. Touto problematikou sa budeme zaoberať v Kapitole 4 a 5

Kapitola 2

Predspracovanie obrázku

Prvou veľmi dôležitou fázou konverzie obrázku do textového dokumentu je jeho predspracovanie. Cieľom predspracovania je odstrániť z obrázku šum, ktorý vznikol pri skenovaní a v prípade menej výrazného textu zabezpečiť, aby nedošlo k označeniu textu ako šumu, a teda jeho odstráneniu. Výsledkom predspracovania je čierno-biely obrázok, kde čierne pixely sú pixely tvoriace písmená a biele pixely tvoria pozadie dokumentu.

2.1 Konverzia do odtieňov šedej

Pred samotným odstránením šumu prevedieme vstupný obrázok do odtieňov šedej. Pixely v takom obrázku nesú informáciu len o intenzite. Intenzitu je možné spočítať podľa vzorca 2.1

$$I = 0.31 \cdot R + 0.58 \cdot G + 0.11 \cdot B \tag{2.1}$$

kde R je veľkosť červenej, G veľkosť zelenej a B veľkosť modrej zložky. Váhy použité na výpočet intenzity majú dané ohodnotenie kvôli tomu, že pri rovnakých množstvách farby je ľudské oko najcitlivejšie na zelenú, potom na červenú a nakoniec na modrú farbu. Prevodom do odtieňov šedej pri väčšine skenovaných dokumentov nestratíme veľa informácie, keďže bežné dokumenty nie sú tlačené takou farbou a na takom farebnom podklade, ktoré majú rovnakú intenzitu v stupňoch šedosti. Dokumenty[1], ktoré obsahujú tmavé písmo na tmavom podklade prípadne veľmi svetlé písmo na svetlom

[1] v tejto fáze spracovávania pod pojmom dokument myslíme skenovaný obrázok obsahujúci textový dokument

podklade, majú po prevode do odtieňov šedej malý kontrast, čo je v procese odstraňovania šumu veľmi nežiadúce. Tento nedostatok opravuje histogramová ekvalizácia.

2.1.1 Histogramová ekvalizácia

Histogram digitálneho obrázka[2] s veľkosťami intenzít v rozsahu [0, L - 1] je diskrétna funkcia $h(r_k) = n_k$, kde r_k je k-ta hodnota intenzity a n_k je počet pixelov v obrázku s intenzitou r_k. V praxi je bežné normalizovať histogram vydelením každej hodnoty celkovým počtom pixelov v obrázku, označeným súčinom MN, kde, ako zvyčajne, M značí počet riadkov a N počet stĺpcov v obrázku. Normalizovaný histogram je teda daný ako $p(r_k) = r_k/MN$, pre $k = 0, 1, 2, ..., L - 1$. Voľne povedané, $p(r_k)$ je odhad pravdepodobnosti výskytu intenzity veľkosti r_k v obrázku. Suma všetkých komponent histogramu je rovná 1.

Poznamenáme, že v tmavom obrázku sú vysoké hodnoty v histograme koncentrované v dolnej časti škály intenzít. Podobne, histogram svetlého obrázku nadobúda vysoké hodnoty v hornej časti škály. Obrázok s malým kontrastom má úzky histogram umiestnený typicky blízko stredu škály. Obrázok s vysokým kontrastom je typický širokým histogramom, a tiež to, že distribúcia histogramu veľmi pripomína rovnomernú. Intuitívne, je rozumné usudzovať, že obrázok, ktorého intenzity pixelov majú tendenciu zaberať celý rozsah možných intenzít a zároveň rozloženie intenzít pripomína rovnomerné, bude mať vzhľad obrázka s vysokým kontrastom a bude obsahovať veľkú časť škály odtieňov šedej.

Označme premennou r intenzity v spracovávanom obrázku z rozsahu [0, L - 1], s $r = 0$ reprezentujúcou čiernu a $r = L-1$ reprezentujúcou bielu. Pre r spĺňajúce tieto podmienky sústredíme našu pozornosť na transformácie (mapovanie intenzít) tvaru

$$s = T(r) \qquad 0 \leq r \leq L - 1 \tag{2.2}$$

ktorých výstupom je intenzita veľkosti s pre každý pixel vo vstupnom obrázku, ktorý má intenzitu r. Predpokladáme, že:

1. $T(r)$ je neklesajúca funkcia na intervale $0 \leq r \leq L - 1$
2. $0 \leq r \leq L - 1$ pre $0 \leq r \leq L - 1$

[2]pod pojmom obrázok v tejto časti myslíme obrázok v odtieňoch šedej

Hodnoty intenzít v obrázku môžeme vnímať ako náhodné veličiny z intervalu [0, L - 1]. Základným deskriptorom náhodnej veličiny je jej hustota. Nech $p(r)$ a $q(s)$ označujú hustoty r a s. Dôležitým poznatkom zo základnej teórie pravdepodobnosti je, že ak $p(r)$ a $T(r)$ sú známe, a $T(r)$ je spojitá a diferencovateľná na celom požadovanom intervale, tak hustotu transformovanej veličiny s získame podľa vzorca

$$q(s) = p(r)\Big|\frac{dr}{ds}\Big| \tag{2.3}$$

Vidíme, že hustota veličiny výslednej intenzity, s, je určená hustotou vstupných intenzít a transformačnou funkciou T.
V spracovávaní obrázku je veľmi dôležitá transformačná funkcia tvaru

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw \tag{2.4}$$

kde w je pomocná premenná pri integrovaní. Pravá strana tejto rovnice je distribučnou funkciou náhodnej veličiny r. Aby sme našli hustotu q, ktorá korešponduje s touto transformačnou funkciou, použijeme 2.3 a Leibnizovo pravidlo, teda

$$\begin{aligned}\frac{ds}{dr} &= \frac{dT(r)}{dr} \\ &= (L-1)\frac{d}{dr}\Big[\int_0^r p(w)dw\Big] \\ &= (L-1)p(r)\end{aligned} \tag{2.5}$$

Dosadením do 2.3 dostaneme

$$\begin{aligned}q(s) &= p(r)\Big|\frac{dr}{ds}\Big| \\ &= p(r)\Big|\frac{1}{(L-1)p(r)}\Big| \\ &= \frac{1}{L-1} \qquad 0 \le s \le L-1\end{aligned} \tag{2.6}$$

Tvar $q(s)$ v poslednom riadku tejto rovnice poznáme ako rovnomerné rozdelenie hustoty. Tým sme dokázali, že vykonaním transformácie intenzít podľa 2.4 dostaneme náhodnú veličinu s charakterizovanú rovnomerným rozdelením, bez ohľadu na vstup do transformačnej funkcie. Pri diskrétnych hodnotách pracujeme s pravdepodobnosťami (hodnotami histogramu)

a sumami namiesto hustôt a integrálov. Ako bolo spomenuté vyššie, pravdepodobnosť výskytu intezity veľkosti r_k v obrázku je aproximovaná podľa

$$p(r_k) = \frac{n_k}{MN} \qquad k = 0, 1, 2, ..., L-1 \tag{2.7}$$

kde MN je celkový počet pixelov v obrázku, n_k je počet pixelov s intenzitou r_k, a L je celkový počet hodnôt intenzít, ktoré môžu byť nadobudnuté (napríklad 256 pre 8-bitový obrázok). Diskrétna forma transformácie 2.4 je

$$\begin{aligned} s_k = T(r_k) &= (L-1)\sum_{j=0}^{k} p_r(r_j) \\ &= \frac{L-1}{MN}\sum_{j=0}^{k}(n_j) \qquad k = 0, 1, 2, ..., L-1 \end{aligned} \tag{2.8}$$

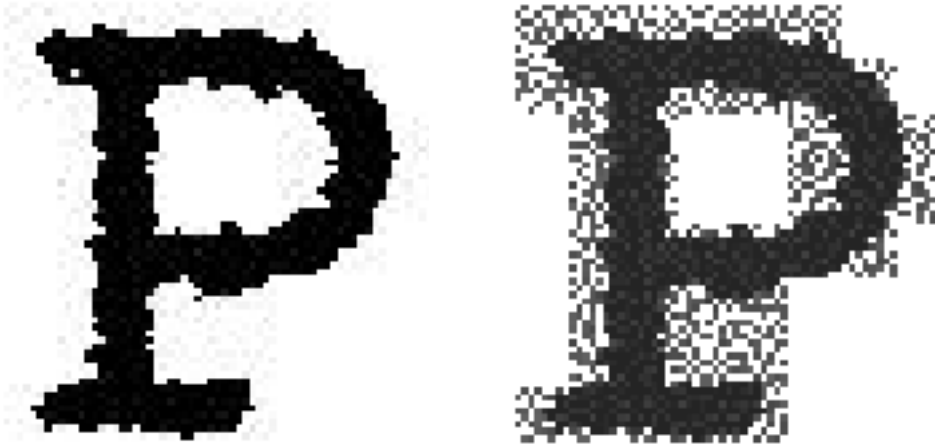
Výsledný obrázok teda získame transformáciou hodnoty intenzity r_k každého pixelu vo vstupnom obrázku podľa vzorca 2.8 na hodnotu s_k. Transformácia $T(r_k)$ v tejto rovnici sa nazýva *histogramová ekvalizácia*.[2]

Na rozdiel od spojitej ekvalizácie, nemôžeme dokázať, že diskrétna histogramová ekvalizácia vytvára rovnomerný histogram. Vo všeobecnosti však má tendenciu rozložiť hodnoty histogramu do väčsej škály hodnôt. Výsledkom je zlepšenie kontrastu potrebné pri ďalšom spracovaní obrázku - lokálnom prahovaní.

2.1.2 Automatická histogramová ekvalizácia

V praxi nie je vždy vhodné používať histogramovú ekvalizáciu. V našej práci sa pokúšame o automatickú histogramovú ekvalizáciu, to znamená, že podľa vlastností obrázka, konkrétne jeho histogramu, sa program sám rozhodne, či ekvalizovať histogram, alebo nie. Na obrázku 2.1a, ktorý predstavuje vystrihnutú a zväčšenú časť skenovaného obrázka, je možné vidieť, že po skenovaní dokumentu obrázok neobsahuje len biele a čierne pixely. V okolí písmen sa niekedy nachádzajú pixely iných odtieňov šedej. Tieto pixely by boli po vykonaní ekvalizácie zvýraznené a výstup by bol podobný tomu na obrázku 2.1b. V našej práci sa preto snažíme neprevádzať ekvalizáciu na tomto type obrázkov. Histogramy týchto obrázkov sa väčšinou podobajú histogramu na obrázku 2.2. Preto po konverzii obrázka do odtieňov šedej je vykonaný jednoduchý test podobnosti týchto dvoch histogramov. Úspešnosť správneho rozhodnutia je približne 70%. Samotná implementácia je veľmi jednoduchá

a bolo by možné nahradiť ju komplexným riešením založenom napríklad na neurónovej sieti. Rozhodovanie by potom fungovalo správne na viacerých typoch obrázkov. O neurónových sieťach sa čitateľ môže dozvedieť viac v 4.2.

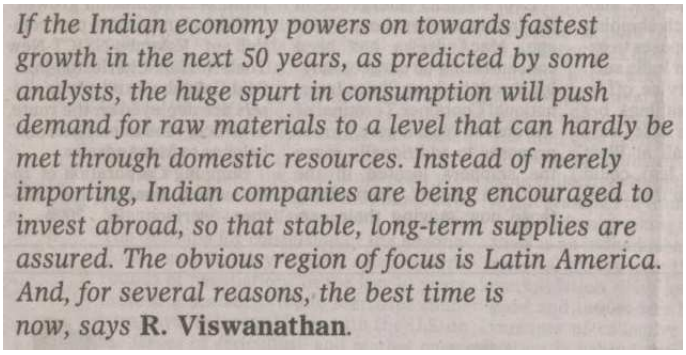


(a) Zväčšená a vystrihnutá časť skenovaného obrázku. Okolo kontúr písmena sa nachádzajú sivé pixely.

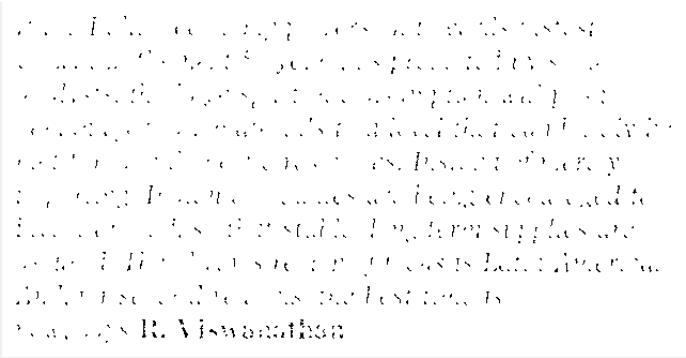
(b) Výsledok histogramovej ekvalizácie. Sivé pixely boli zvýraznené.

Obr. 2.1: Príklad obrázku nevhodného na histogramovú ekvalizáciu

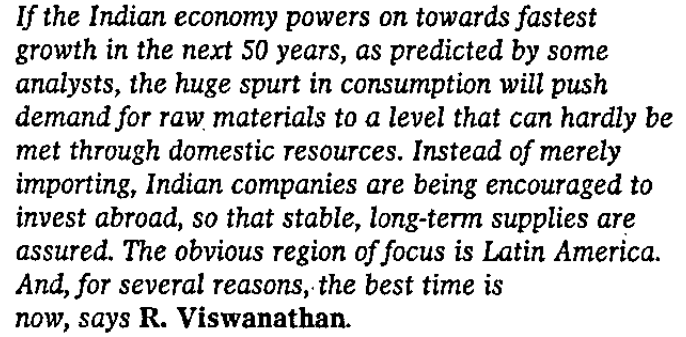
Obr. 2.2: Histogram obrázkov nevhodných na histogramovú ekvalizáciu



(a) Vstupný obrázok s menej výrazným textom.



(b) Odstránenie šumu z obrázku podľa 2.2 bez použitia histogramovej ekvalizácie



(c) Odstránenie šumu z obrázku podľa 2.2 s použitím histogramovej ekvalizácie

Obr. 2.3: Efekt histogramovej ekvalizácie

2.2 Lokálne prahovanie

Prahovanie[3] vo všeobecnosti patrí medzi základné transformácie intenzít pixelov v obrázku. Cieľom prahovania je oddeliť požadované objekty v obrázku od pozadia. Najzákladnejšou metódou je vybrať prah T, ktorý vytvorí dve skupiny pixelov, v našom prípade pixely reprezentujúce písmená textu a pixely pozadia:

- ak pre pixel na pozícii (x,y) platí, že $f(x, y) \leq T$, bude považovaný za pixel písmena, inak
- ak platí, že $f(x, y) > T$, bude považovaný za pixel pozadia

kde $f(x, y)$ je pixel (x,y) vo vstupnom obrázku.

Ak je T konštanta aplikovateľná na celý obrázok, proces prahovania nazývame *globálne prahovanie*. Ak sa hodnota T mení s pozíciou v obrázku, používame pojem *variabilné prahovanie*. Niekedy sa používa názov *lokálne prahovanie*, ak chceme zdôrazniť, že ide o variabilné prahovanie pri ktorom hodnota T na ľubovoľnej pozícii (x,y) v obrázku závisí na vlastnostiach okolia (x,y). Ak T závisí od súradníc (x,y) v priestore, hovoríme o *dynamickom* alebo *adaptívnom prahovaní*.

Globálne prahovanie je možné použiť len na obrázkoch v ktorých je rozloženie intenzít pixelov objektov a pozadia zreteľné. Navyše je nutné počítať aj s variabilitou vstupov, ktorá si vynucuje použitie algoritmu na automatické určenie prahu pre ten ktorý obrázok. V našej práci je preto oveľa vhodnejšie použiť lokálne prahovanie.

Základným postupom v lokálnom prahovaní je použitie smerodatnej odchýlky a priemeru intenzít v okolí S_{xy} bodu (x,y). Bežné formy lokálnych prahov majú tvar

$$T_{xy} = a\sigma_{xy} + bm_{xy} \tag{2.9}$$

kde a a b sú nezáporné konštanty a

$$T_{xy} = a\sigma_{xy} + bm_G \tag{2.10}$$

kde m_G je globálny priemer intenzít pixelov. Výsledný čiernobiely obrázok vypočítame nasledovne:

$$g(x, y) = \begin{cases} 1 & \text{ak } f(x, y) > T_{xy}, \text{ (pixel (x,y) je súčasťou pozadia)} \\ 0 & \text{ak } f(x, y) \leq T_{xy} \end{cases} \tag{2.11}$$

[3]angl. thresholding

kde $f(x, y)$ je pixel vstupného obrázka. Táto rovnosť je aplikovaná na každý pixel vstupu a na každej pozícii (x,y) je vypočítaný nový prah s použitím okolitých pixelov S_{xy}.

V našej práci sme použili modifikáciu špeciálneho prípadu lokálneho prahovania, ktorý je nazývaný *prahovanie s pohybujúcimi sa priemermi*. Pôvodná metóda je založená, ako je z názvu zrejmé, na priebežnom výpočte priemeru intenzít pozdĺž spracovávaných línií v obrázku. Výpočet je veľmi rýchly a aj preto sa tento spôsob lokálneho prahovania používa veľmi často:

Nech z_{k+1} označuje intenzitu pixelu v spracovávanej línii v kroku k + 1. Priemer v tomto bode je daný

$$\begin{aligned} m(k+1) &= \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i \\ &= m(k) + \frac{1}{n}(z_{k+1} - z_{k-n}) \end{aligned} \tag{2.12}$$

kde n označuje počet bodov použitých pri výpočte priemeru a $m(1) = z_1/n$. Pretože je priemer počítaný pre každý pixel v obrázku, výsledný obrázok dostaneme použitím 2.11 s $T_{xy} = bm_{xy}$, kde b je konštanta a m_{xy} je priemer z 2.12 v bode (x,y) vstupného obrázku.[2]

2.2.1 Modifikácia

Označme z_{k+1} ako priemer intenzít n bodov kolmých na aktuálne spracovávaný bod, tak že vzdialenosť najvzdialenejšieho bodu od aktuálneho bodu je maximálne $\lceil n/2 \rceil$. Pri počítaní pohybujúceho sa priemeru tak berieme do úvahy štvorec veľkosti $n \times n$ so stredom v aktuálnom bode. Kladom tejto modifikácie je fakt, že prahová hodnota ľubovoľného bodu viac korešponduje s jeho okolím a vie sa teda lepšie vysporiadať s prítomným šumom a teda oddeliť pozadie od písmen. Cenou tejto modifikácie je počítanie dodatočného priemeru intenzít v čase O(n). Táto modifikácia je teda n-krát pomalšia ako bežná verzia lokálneho prahovania s pohybujúcimi sa priemermi. Výsledok aplikácie tejto modifikácie lokálneho prahovania je zobrazený na obrázku 2.4.

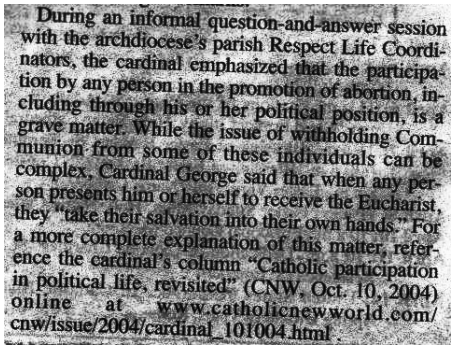
2.3 Odstránenie zbytkového šumu

Ďalším krokom pri predspracovaní obrazu je odstránenie tých pixelov písmen alebo pozadia, ktorých najbližšie okolie obsahuje minimálne K pixelov s rovnakou hodnotou intenzity[4]. Najbližšie okolie obsahuje práve 8 pixelov, platí teda $0 \leq K \leq 8$. Pri predspracovávaní nemá zmysel položiť $K < 6$. Pripomíname, že teraz spracovávame čiernobiely obrázok. Pre $K = 8$ dochádza k odstráneniu osamotených pixelov a pri $K = 7$ dosiahneme odstránenie čiar širokých jeden pixel. Odstránením týchto elementov sa zbavíme príliš malých oblastí, ktoré sú označené ako písmeno, prípadne vyhladíme kontúry písmen, ako je možné vidieť na obrázku 2.5b.

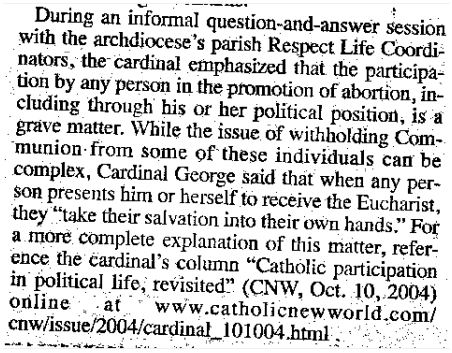
2.4 Vlastné filtre

Naša aplikácia umožňuje pokročilú správu filtrov. Každý filter je plugin, ktorý je načítaný na začiatku aplikácie. Veľmi skúsený užívateľ si môže vytvoriť vlastný filter, prípadne celú sadu filtrov. Manažér filtrov umožňuje vyberať zo zoznamu dostupných filtrov tie, ktoré sa majú použiť pri predspracovaní. Každý filter môže byť aplikovaný ľubovoľne veľa krát a v ľubovoľnom poradí. Každý filter môže obsahovať ľubovoľne veľa nastavení, ktoré je možné v manažéri meniť. Nastavenia sú individuálne pre každú použitú inštanciu filtru. Jedinou podmienkou pre správnu funkčnosť je to, že obrázok musí byť po prechode všetkými zvolenými filtrami čiernobiely, kde čierna má hodnotu 0 a biela hodnotu 255. Podrobnosti o vlastných filtroch a ich použití je možné nájsť v užívateľskej príručke k aplikácii, v kapitole Filtre.

[4] tento filter sa anglicky nazýva K-fill filter

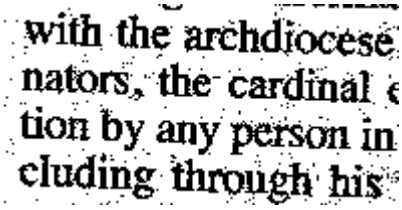


(a) Obrázok obsahujúci šum na pozadí písmen. Obrázok bol upravený pomocou histogramovej ekvalizácie

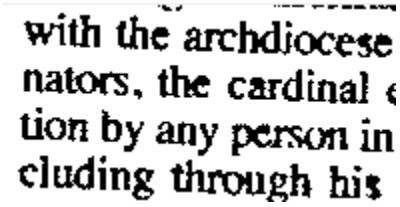


(b) Obrázok upravený pomocou lokálneho prahovania. Veľkosť prahovacieho okna je 17.

Obr. 2.4: Aplikácia modifikovaného lokálneho prahovania



(a) Obrázok obsahujúci zbytkový šum.



(b) K-fill filter aplikovaný na obrázok. K = 6

Obr. 2.5: Aplikácia K-fill filtru

Kapitola 3

Analýza členenia dokumentu

3.1 Základné poznatky

Analýza členenia dokumentu je vykonávaná na zistenie fyzickej štruktúry dokumentu, to znamená na rozpoznanie komponent dokumentu. Komponenty dokumentu môžu byť:

- *samostatné súvislé oblasti* - oblasti navzájom pospájaných čiernych pixelov, ktoré vytvárajú základné komponenty ako zbytkový šum, bodky, čiarky, písmená bez diakritiky a iné
- *skupiny základných komponentov* napríklad kompletné znaky vrátane diakritiky (napr. í,á,é,ä), slová, riadky textu alebo bloky textu.

Dominantná orientácia riadkov textu určuje *uhol otočenia*. Skutočný dokument má uhol otočenia nulový, ale pri skenovaní sa môže tento uhol zmeniť.

Analýzu členenia môžeme vykonať spôsobom *zhora nadol*, pri ktorom môže byť dokument rozdelený na viac stĺpcov, každý stĺpec je rozdelený na odstavce, každý odstavec na riadky atď. Alternatívne, analýza môže byť vykonaná technikou *zdola nahor*, pri ktorej sú súvislé oblasti spájané do znakov, slov, riadkov textu atď. Po vykonaní analýzy môže nasledovať označenie funkcií jednotlivých blokov textu na základe vlastností zistených pred ňou. V našej práci sa touto problematikou nezaoberali.

3.1.1 Detekcia uhlu otočenia

Uhol otočenia dokumentu je možné zistiť viacerými spôsobmi. V stručnosti popíšeme tieto často používané:

- použitie profilu projekcie
- zoskupovanie najbližších susedov

Profil projekcie je histogram počtu čiernych pixelov zozbieraných pozdĺž paralelných línií v celom obrázku. V dokumente s vodorovnými riadkami textu bude mať profil horizontálnej projekcie vrcholky so šírkami rovnajúcimi sa výškam písmen a profil vertikálnej projekcie bude mať šírky vrcholkov zodpovedajúce stĺpcom textu. Aby sme zistili uhol otočenia, profil projekcie je počítaný pod niekoľkými rôznymi uhlami blízko očakávaného uhlu otočenia. Pre každý tento uhol sa vypočíta celková veľkosť zmeny v profile projekcie. Maximálna zmena odpovedá najlepšiemu zarovnaniu s riadkami a tak je určený uhol otočenia. Obmedzením tejto metódy je to, že je vhodná len na dokumenty s malým uhlom otočenia, typicky menším ako $\pm 10°$.

Druhá spomínaná metóda využíva techniku *zdola-nahor*, ktorá je založená na vyhľadaní dominantného uhlu v histograme uhlov medzi najbližšími susedmi každého komponentu súvislosti. Výhodou tejto metódy je to, že nie je limitovaná na žiadny rozsah uhlov.

3.1.2 Spôsoby analýzy členenia dokumentu

Po zistení uhlu otočenia je obrázok zvyčajne pootočený do nulového uhlu a je na ňom vykonaná analýza členenia. Hlavným rozdielom medzi nami použitou technikou (viď 3.1.3) a technikami *zhora-nadol* je ten, že naša technika nie je obmedzená na *Manhattanské rozloženie* dokumentu, t.j. rozloženie ktorého bloky textu sú oddeliteľné vertikálnymi alebo horizontálnymi rezmi. Medzi techniky analýzy patria napríklad:

- varianty algoritmu *RLSA*[1]. Táto metóda zoskupuje písmená do slov, slová do riadkov textu a (niekedy) riadky textu do odstavcov "rozmazaním" textu aby sa vytvorili spojené oblasti textu. To je možné dosiahnuť použitím vyhladovacích filtrov ktorých veľkosti jadier sú odvodené od veľkostí medzier medzi jednotlivými komponentami. Táto metóda vyžaduje obrázok s nulovým sklonom textu a známymi a rovnomernými veľkosťami medzier.
- použitie profilov vertikálnej a horizontálnej projekcie na rozdelenie dokumentu na menšie bloky obdĺžnikového tvaru.

[1] Run-length smoothing algorithm

- technika typu zhora nadol založená na analýze medzier na izolovanie blokov textu a použití profilu projekcie na zistenie riadkov textu. Ako pri všetkých technikách zhora-nadol, dokument musí mať Manhattanské rozloženie.

3.1.3 Docstrum

V našej práci sme použili na analýzu členenia dokumentu metódu, ktorú autor nazval *Docstrum*. Docstrum[2] je reprezentácia dokumentu, ktorá popisuje vlastnosti jeho štruktúry a môže byť použitá na analýzu jej rozloženia.

Metóda je založená na technike zdola-nahor - na zlučovaní k najbližšich susedov súvislých oblastí. k najbližších susedov komponentu i je k najbližších komponentov, kde vzdialenosť dvoch komponentov je počítaná pomocou Euklidovskej vzdialenosti v obrázku. Pre $0 ¡ j ¡ k$ je každý pár najbližších susedov i,j popísaný dvojicou $D_{ij}(d, \phi)$, kde d je vzdialenosť a ϕ uhol medzi stredmi komponentov i a j. Dva susedné znaky v slove budú mať teda relatívne malú vzdialenosť a ich uhol bude približne taký veľký ako uhol pootočenia dokumentu. Pre $k = 5$ bude mať znak dvoch až troch susedov z toho istého slova alebo zo susedného slova v rámci jedného riadku a zvyšné dvojice budú vytvárať znaky z riadkov pod alebo nad riadkom, v ktorom sa nachádza aktuálny znak. Tieto medziriadkové dvojice majú d väčšie ako tie vytvorené v rámci jedného riadku, a uhol približne zodpovedá uhlu priamky kolmej na hlavnú líniu textu.

Docstrum je graf $D_{ij}(d, \phi)$ pre všetky nájdené dvojice. Priamo z grafu je možné zistiť uhol pootočenia a vlastnosti ako priemernú vzdialenosť v rámci riadku a priemernú vzdialenosť medzi dvoma riadkami.[3]

Pri stránkach obsahujúcich text rôznej veľkosti, ako napríklad veľké znaky v nadpisoch a menšie znaky v texte, by mala byť zmieňovaná metóda aplikovaná buď na každú podmnožinu veľkostí textu alebo na každú veľkosť textu samostatne. Dôvodom je fakt, že samotná analýza je založená na údajoch zistených vo vyššie spomínanom grafe. Prítomnosť širokého rozsahu veľkostí textu zväčší odchýlku od priemeru, čo môže viesť k chybných výsledkom. Rozdelenie komponentov podľa ich veľkosti je však nutné len pri veľkých rozsahoch veľkostí, rozsah veľkostí veľkých a malých písmen je zanedbateľný.

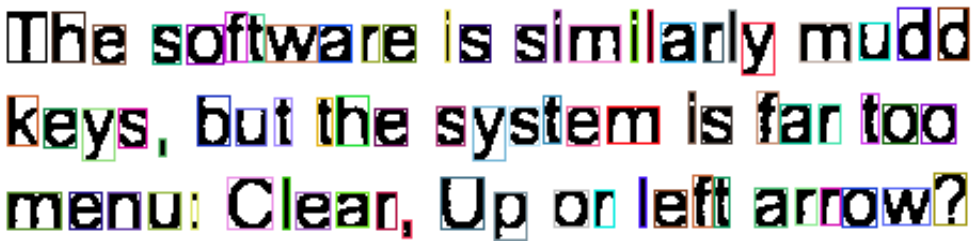
V práci pre zjednodušenie predpokladáme malé rozsahy veľkostí komponentov za cenu čiastočne chybných výsledkov pri dokumentoch s veľkými

[2]skratka z *document spectrum*

nadpismi a malým písmom.

3.2 Hľadanie znakov

Pred samotnou analýzou je nutné označiť súvislé oblasti čiernych pixelov, aby sme vedeli, ktoré pixely patria ku ktorému písmenu, prípadne inému elementu textu (diakritika, interpunkcia). V tejto fáze už predpokladáme, že obrázok neobsahuje šum, keďže bol spracovaný postupom popísaným v kapitole 2. Pri každom písmene nás zaujíma najmenší obdĺžnik v ktorom sa daný znak nachádza. Predpokladáme, že počet komponentov v obrázku nepresiahne 65535. Táto hodnota nie je obmedzujúca, nakoľko priemerne hustý text sa skladá z približne 2000 písmen. Nami použitý algoritmus značkovania oblastí je popísaný v [4] a v našej práci ho nebudeme rozoberať. Tento algoritmus patrí medzi jednopriechodové algoritmy, to znamená, že už po prvom prejdení celého obrázka je známy výsledok. Pracuje s kontúrami značkovaných oblastí a patrí medzi najrýchlejšie zo značkovacích algoritmov. Príklad označených súvislých oblastí je zobrazený na obrázku 3.1



Obr. 3.1: Príklad označených súvislých oblastí v obrázku

Úprava vstupného obrázka

Náš software umožňuje vo vstupnom obrázku označiť oblasti, ktoré sa majú alebo nemajú spracovávať. Rozlišujeme tri typy oblastí:

- rozpoznávaná oblasť
- oblasť označujúca obrázok
- nespracovávaná oblasť

Rozpoznávaná oblasť je tá časť obrázka, v ktorej má dôjsť k rozpoznaniu písmen a štruktúry textu. Oblasť označujúca obrázok slúži na označenie tej časti obrázka, ktorú chceme skopírovať na výstup. Nespracovávaná oblasť je oblasť, ktorá sa bude v ďalších krokoch konverzie obrázka na text ignorovať. Ak je označená nejaká oblasť na rozpoznávanie, tak zvyšná časť obrázku bude považovaná za nespracovávanú oblasť, okrem oblastí označujúcich obrázok. Pred značkovaním písmen je vytvorená na základe označených oblastí maska M, ktorá sa aplikuje na obrázok O nasledovne:

$$ak(M_{xy} == kód\ nespracovávanej\ oblasti) potom\ O_{xy} := 255$$

kde M_{xy} je kód oblasti v obrázku na pozícii *(x,y)* a O_{xy} je intenzita na pozícii *(x,y)*. Hodnota *255* označuje bielu farbu, teda farbu pozadia.

Podrobné informácie o oblastiach je možné nájsť v užívateľskej príručke, kapitola *Práca s obrázkom a dávkou*, podkapitola *Výber oblastí na spracovanie*.

3.3 Hľadanie najbližších susedov

Prvým krokom algoritmu docstrum je nájdenie k najbližších susedov každého znaku. Predtým však musí byť vybratá hodnota k. Ideálne, chceli by sme nájsť susedov napravo, naľavo, nad a pod aktuálnym znakom. Znalosť týchto susedov nám dá informáciu o veľkosti medzier medzi znakmi v riadku a medzi riadkami. Náš program má preto štandardne nastavenú hodnotu k = *4*. Táto hodnota sa však dá zmeniť v nastaveniach programu (viď užívateľská príručka, kapitola *Nastavenia programu*). Ak sú vzdialenosti medzi riadkami v dokumente veľké, je nutné zvoliť väčšie k, typicky 6 až 7. Keďže je priemerný počet znakov v dokumente veľký (1000 až 5000 v závislosti od typu a textu), tento krok je časovo najnáročnejší zo všetkých krokov, ktoré nespracovávajú pixely samostatne. Najpriamejšie implementácie hľadania k najbližších susedov majú časovú zložitosť $O(n^2)$. V našej práci však používame *kD stromy*, ktoré zmenšujú časovú zložitosť. Podrobnejšie sa im budeme venovať v podkapitole 3.3.1

3.3.1 2D stromy

2D strom je špeciálnym prípadom *kD stromu*. *kD strom* je dátová štruktúra, ktorá slúži na organizáciu bodov v k-dimenzionálnom priestore. Je to

binárny strom, v ktorom každý uzol reprezentuje bod k-dimenzionálneho priestoru. Každý vnútorný uzol vytvára deliacu nadrovinu, ktorá rozdeľuje priestor na dva podpriestory (v našom prípade rovinu na polroviny). Body naľavo od nadroviny sa nachádzajú v ľavom podstrome, body napravo v pravom podstrome. Smer nadroviny je určený nasledujúcim spôsobom: každý koreň podstromu je asociovaný s jednou z dimenzií tak, že nadrovina je kolmá na vektor danej dimenzie. V prípade 2D stromov, ak je s nejakým uzlom asociovaná prvá dimenzia, tak sú body v rovine rozdelené na dve skupiny: skupina s x-ovou súradnicou menšou ako je x-ová súradnica daného uzlu a skupina s väčšou x-ovou súradnicou ako je x-ová súradnica uzlu.

Vytvorenie 2D stromu

Existuje mnoho spôsobov ako konštruovať tento typ stromu, keďže existuje veľa možností v akom poradí vyberať smer nadroviny v uzloch. Základná metóda konštrukcie spĺňa nasledujúce podmienky:

- Pri prechode od koreňa k listu sú uzlom priradené deliace nadroviny podľa cyklicky vyberaných osí (napríklad koreň bude mať deliacu nadrovinu rovnobežnú s x-ovou osou, jeho potomkovia s y-ovou osou, ich potomkovia opäť s x-ovou osou atď).
- Body sú vkladané do stromu výberom mediánu z aktuálne vkladaných bodov zoradených podľa aktuálnej súradnice použitej pri vytvorení deliacej nadroviny. To vedie k vytvoreniu vyváženého stromu.

Nasledujúci kód popisuje vytvorenie 2D stromu:

```
Uzol Vytvor2Dstrom (ZoznamBodov pointList, TypOsi aktualnaOs)
{
    if (pointList je prázdny zoznam)
      return null;
    else
    {
        zoraď pointList podľa aktualnaOs;
        prostrednyBod := median(pointList);
        Uzol novyUzol;
        novyUzol.bod := prostrednyBod;
        TypOsi novaOs :=ZiskajDalsiuOs(aktualnaOs);
        novyUzol.lavyPotomok := Vytvor2Dstrom(body z pointList pred mediánom,novaOs);
        novyUzol.pravyPotomok := Vytvor2Dstrom(body z pointList za mediánom,novaOs);
        return novyUzol;
    }
}
```

Časová zložitosť vytvorenia stromu je v našom prípade $O(Nlog^2N)$, keďže na zistenie mediánu triedime zoznam bodov v čase $O(nlogn)$.

Vyhľadanie najbližších susedov v 2D strome

Vyhľadanie najbližšieho suseda daného znaku v 2D strome prebieha nasledovne:

Začínajúc v koreni stromu, algoritmus postupuje stromom rekurzívne smerom k listu takým istým spôsobom, akým by bol daný bod do stromu vkladaný (t.j. postúpi do pravého alebo ľavého podstromu aktuálneho uzlu podľa toho, či je súradnica pozície daného znaku, podľa ktorej sa rozdeľuje, menšia alebo väčšia ako je súradnica aktuálneho uzlu)

Keď algoritmus dôjde do listu, uloží bod, ktorý daný list reprezentuje ako doteraz najbližší bod. Pri vynáraní sa z rekurzie sú vykonávané v každom uzle nasledujúce kroky:

- Ak je aktuálny uzol bližšie ako doteraz najbližší bod, potom sa stane doteraz najbližším bodom.

- Algoritmus overí, či sa môžu nachádzať nejaké body, ktoré ležia v druhej polrovine, bližšie k zadanému bodu ako aktuálne najbližší bod. Keďže sú deliace nadroviny rovnobežné s osami, tento krok je implementovaný ako jednoduchý test, či rozdiel deliacej súradnice uzlu a tej istej súradnice hľadaného bodu je menší ako vzdialenosť hľadaného bodu od doteraz najbližšieho bodu.

  Ak je rozdiel menší, môžu sa nachádzať v druhej polrovine ešte bližšie body, takže algoritmus musí pokračovať prechodom druhého podstromu aktuálneho uzlu rovnakým spôsobom ako funguje celé vyhľadávanie.

  Ak rozdiel nie je menší, algoritmus pokračuje vo vynáraní sa z rekurzie a teda eliminuje celý nepreskúmaný podstrom aktuálneho bodu.

Vyhľadávanie je dokončené, keď algoritmus spracuje koreň stromu.[5]

Tento základný algoritmus musíme modifikovať, aby sme ho mohli v práci použiť. Pri vyššie zmienenom postupe by algoritmus vrátil pre akýkoľvek znak v dokumente ten istý znak, keďže každý znak z dokumentu sa vo vytvorenom strome nachádza. Ďalší problém je ten, že potrebujeme nájsť k najbližších susedov bez toho aby sme menili štruktúru stromu. Obidva

problémy vyriešime pomerne jednoduchým spôsobom: použijeme pomocné pole identifikátorov znakov, ktoré nemôžeme pri hľadaní najbližšieho suseda použiť. Pred vyhľadaním prvého najbližšieho suseda do tohto poľa vložíme aj identifikátor aktuálneho znaku. Tým vyriešime prvý problém. Po každom vyhľadaní najbližšieho suseda vložíme do tohto poľa aj identifikátor nájdeného znaku a zabezpečíme tak nájdenie k rôznych susedov.

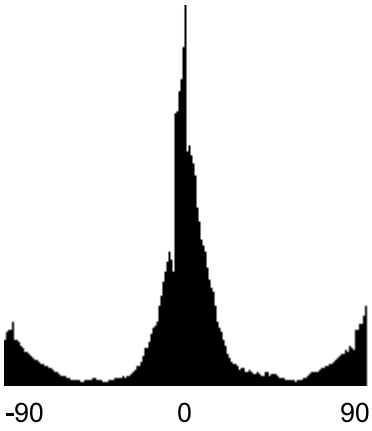
3.3.2 Určenie vzdialeností písmen a riadkov

Z docstrum grafu je možné zistiť veľkosť pootočenia textu, vzdialenosti medzi stredmi písmen v rámci riadku a medzi dvomi riadkami. Orientáciu textu odhadneme z histogramu uhlov medzi nájdenými dvojicami susedov. Pri výpočte veľkosti uhlu nezáleží na poradí znakov, a preto spadá do rozsahu $[0, 180°]$. Histogram je vytvorený z uhlov zaokrúhlených na celé čísla, pretože zatiaľ nepotrebujeme väčšiu presnosť. Ďalej histogram cyklicky vyhladíme oknom šírky 10, pretože dáta o uhloch sú cyklicky spojité. Vo vyhladenom histograme nájdeme maximum, pričom jeho pozícia určuje priemernú orientáciu textu. Táto hodnota je len prvý odhad, presnejší odhad zistíme neskôr (viď 3.6). Na obrázku 3.2 je zobrazený typický histogram nepootočeného skenovaného dokumentu.

Po zistení odhadu uhlu vytvoríme histogramy vzdialeností. Histogram vzdialeností v rámci riadku vytvoríme zo vzdialeností tých dvojíc susedov, ktorých uhol spadá okolia odhadnutého uhlu. Obdobne vytvoríme histogram vzdialeností medzi riadkami zo vzdialeností susedov, ktorých vzájomný uhol spadá do takého istého okolia uhlu kolmého na odhadnutý uhol. Obidva histogramy sú vyhladené, ak obsahujú dostatočný počet vzoriek. Minimálny počet vzoriek potrebný na vyhladenie, veľkosť okolia aj šírku vyhladzovacieho okna je možné v nastaveniach programu zmeniť (viď užívateľská príručka, kapitola Nastavenia programu). Ak celkový počet písmen kriticky malý (menší ako počet hľadaných najbližších susedov), tak program predpokladá nulový uhol otočenia dokumentu a každý znak je považovaný za samostatný blok. Príklady vytvorených histogramov sú zobrazené na obrázku 3.3.

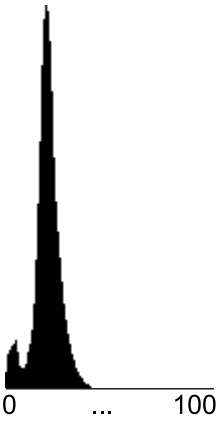
3.4 Spracovanie diakritiky

Pôvodný algoritmus docstrum nepočíta s príliš malými znakmi. Tie sú vopred odstránené. Medzi tieto znaky patrí napríklad bodka, čiarka, bodka nad *i*, mäkčene, dĺžne, elementy tvoriace dvojbodku alebo bodkočiarku, prípadne



Obr. 3.2: Typický histogram uhlov k najbližších susedov nepootočeného skenovaného dokumentu. Z obrázka je vidieť, že najviac susedných dvojíc zviera uhol približne 0°

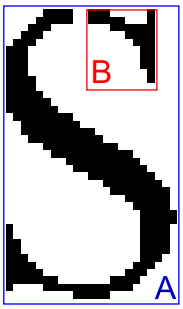
(a) Príklad histogramu vzdialeností susedov v riadku



(b) Príklad histogramu vzdialeností susedov medzi riadkami

Obr. 3.3: Príklad histogramov vzdialeností v pixeloch medzi k najbližšími susedmi.

bodka, ktorá je súčasťou otázniku alebo výkričníku. V našej práci pracujeme aj s týmito znakmi, pretože sú nevyhnutné pri rozpoznávaní textu. Popíšeme teda náš algoritmus, ktorý zabezpečí správne priradenie vyššie spomínaných znakov k ich nadradeným znakom a zároveň odstráni tie znaky, ktoré sú celé obsiahnuté v obdĺžniku, ktorý obklopuje nejaký susedný znak, ako ukazuje obrázok 3.4. Vo všeobecnosti platí, že diakritika sa nachádza v blízkosti nadradeného písmena v smere približne kolmom na smer textu. To isté platí aj o výkričníku a otázniku a túto podmienku spĺňa aj dvojbodka alebo bodkočiarka, ak budeme považovať jeden z elementov, z ktorých sa skladajú, za nadradený. Vopred predpokladáme, že spomínané znaky sa nachádzajú medzi k najbližšími susedmi nadradeného znaku, alebo naopak. Algoritmus vykoná pre každého najbližšieho suseda všetkých znakov nasledovné:



Obr. 3.4: Príklad písmena S rozdeleného na dve časti A a B, kde A obsahuje B. Časť B je v ďalšom procese spracovania písmen ignorovaná.

Označme Z ako aktuálny znak a S nech je jeho sused a D nech je priemerná vzdialenosť medzi písmenami získaná podľa 3.3.2, C a K nech sú konštanty.

```
if (S je obsiahnutý v Z a zároveň je jeho vzdialenosť menšia ako D * C)
{
    nastav S ako neviditeľný znak;
}
else
{
    if (S je viditeľný a jeho vzdialenosť od Z je menšia ako D * C &&
       && uhol ktorý zviera S a Z je kolmý na dominantný uhol  K)
    {
        T := menší znak z S a Z;
        U := väčší znak z S a Z;
```

```
        if(T ešte nemá vlastníka)
        {
            nastav Z ako vlastníka T;
        }
        else
        {
           if(vzdialenosť pôvodného vlastníka od T > vzdialenosť S od Z)
           nastav U ako vlastníka T;
        }
    }
}
```

Po vykonaní tohto algoritmu už len stačí prejsť všetkých vlastníkov nejakých znakov a upraviť im veľkosť a pozíciu obdĺžnika, ktorý daný znak obklopuje, podľa znakov ktoré vlastní. Znaky, ktoré vlastní, sa označia ako neviditeľné. Pri vlastníkovo si zapamätáme, že je to znak zložený z viacerých častí. Tieto neviditeľné znaky sú v ďalších krokoch konverzie ignorované.

Štandardne je nastavená C na hodnotu *1,3* a K na hodnotu *15*. Ak je text písaný veľkými písmenami, hodnotu C je potrebné zmenšiť približne na *1,1*. V opačnom prípade sa môže stať, že algoritmus začne zlučovať interpunkciu a znaky medzi dvoma rôznymi riadkami v dokumentoch s malými medzerami medzi riadkami.

3.5 Predbežné vytvorenie slov

Ďalší hlavný krok v analýze rozloženia dokumentu po nájdení písmen je vytvorenie slov a riadkov. Slová sú vytvorené vykonaním tranzitívneho uzáveru susedov nachádzajúcich sa v tom istom riadku, ktorých vzdialenosť je menšia ako *konštanta* $C \times$ *priemerná vzdialenosť medzi písmenami*. Test či sa dve písmená nachádzajú v tom istom riadku je vykonaný na základe kolmej vzdialenosti priamok, ktoré prechádzajú stredmi týchto písmen. Ak je vzdialenosť menšia ako *konštanta* $V \times$ *priemerná vzdialenosť medzi riadkami*, tak sa písmená nachádzajú v tom istom riadku.

V tejto fáze nemusíme hľadať presnú konštantu C ktorá zabezpečí vytvorenie slov tak ako sú v originálnom dokumente. Stačí aby bola dostatočne veľká na to aby zlučovala susediace písmená v jednom riadku a aby bola dostatočne malá na to aby nezlučovala písmená, ktoré sú síce v jednej línii, ale nachádzajú sa v rôznych odstavcoch. Štandardne má táto konštanta veľkosť *1,2*.

Konštanta V má hodnotu *0,3*, ale pri atypických textoch (napríklad vlnitý text) je ju možné zmeniť v nastaveniach programu.

3.6 Vytvorenie riadkov

Riadky textu sú vytvorené zlučovaním nájdených slov. Každé slovo je na začiatku považované za samostatný riadok textu. Dva slová sú zlúčené ak platia nasledujúce podmienky:

- existuje písmeno v slove ktoré má suseda s ktorým zviera uhol blízky dominantnému uhlu
- vzdialenosť medzi týmto písmenom a jeho susedom je menšia ako *konštanta* $B \times$ *priemerná vzdialenosť medzi písmenami*
- sused nie je súčasťou aktuálneho slova

Tento algoritmus končí po prejdení všetkých susedov všetkých písmen a výsledné slová sú považované za riadky textu. Hodnota konštanty B by mala byť väčšia ako hodnota vyššie spomenutej konštanty C a zároveň dostatočne malá na to, aby neboli zlúčené do jedného riadka textu dve slová z rôznych odstavcov. Hodnota tejto konštanty bola empiricky nastavená na *2,5*. Nastavenia programu ju samozrejme umožňujú zmeniť. Jej zmena (zmenšenie) je vhodná napríklad pri dokumentoch, ktoré majú malé medzery medzi stĺpcami.

Musíme poznamenať, že v jednom skutočnom riadku textu sa môže nachádzať viac výsledných slov (ktoré by mali reprezentovať riadok textu pri vhodnej veľkosti konštanty B) vytvorených vyššie spomenutým algoritmom.

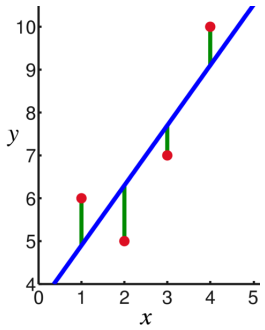
Po skončení algoritmu dopočítame ku každému riadku koeficienty priamky, ktorej celková vzdialenosť od stredov písmen je najmenšia. Túto priamku určíme metódou najmenších štvorcov.

Nech má priamka vyjadrenie $y = ax + b$, potom pre n zadaných bodov nájdeme koeficienty priamky nasledovne:

$$a = \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - n \sum_{i=1}^{n} x_i y_i}{(\sum_{i=1}^{n} x_i)^2 - \sum_{i=1}^{n} (x^2)}$$

$$b = \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} y_i)(\sum_{i=1}^{n} x_i^2)}{(\sum_{i=1}^{n} x_i)^2 - n \sum_{i=1}^{n} (x_i^2)} \tag{3.1}$$

Priamka vyjadrená pomocou týchto koeficientov je priamka, ktorej súčet *vertikálnych* vzdialeností od zadaných bodov je najmenší, ako ukazuje obrázok 3.5 [6]. Takýto výpočet je zlý pre obrázky, ktoré sú otočené o viac ako 45°. Pri tomto type obrázkov vymeníme súradnice a pôvodné vzorce sa teda zmenia na:



Obr. 3.5: Metóda najmenších štvorcov. Priamka je vytvorená na základe vertikálnych vzdialeností od zadaných bodov

$$a = \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - n \sum_{i=1}^{n} x_i y_i}{(\sum_{i=1}^{n} y_i)^2 - \sum_{i=1}^{n} (y_i^2)}$$

$$b = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i^2)}{(\sum_{i=1}^{n} y_i)^2 - n \sum_{i=1}^{n} (y_i^2)} \tag{3.2}$$

Pre uhol rovný 90° položíme a rovné nekonečnu a b priemernej veľkosti y-ovej súradnice písmen. To zaručí funkčnosť na akomkoľvek type obrázkov. Z uhlov otočenia týchto priamok zistených pomocou ich koeficientu a určíme konečný odhad otočenia dokumentu. Tento odhad je presnejší ako prvotný odhad, pretože dlhé priamky podstatne redukujú vplyv písmen zo spodným doťahom a vplyv šumu na vytváraný histogram uhlov.

3.7 Vytvorenie blokov textu

Vytváranie odstavcov pozostáva z testovania všetkých dvojíc nájdených riadkov, či sa nachádzajú v tom istom bloku. Dva riadky textu sa nachádzajú v tom istom bloku:

- ak sú približne rovnobežné
- ak majú malú kolmú vzdialenosť
- ak sa prekrývajú do istej miery alebo sú oddelené malou medzerou.

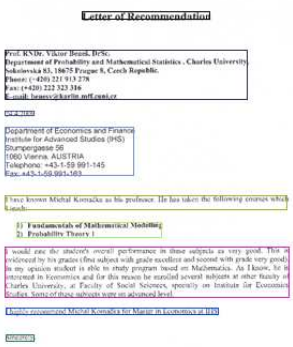
Ak nejaká dvojica riadkov textu spĺňa tieto podmienky a ani jeden z nich nie je priradený žiadnemu bloku, obidva sú priradené novému bloku. Ak je práve jeden z riadkov priradený nejakému bloku, tak aj druhý riadok je priradený tomu istému bloku. Ak oba riadky sú už priradené rôznym blokom, tak sú tieto bloky zlúčené do jedného. Po otestovaní všetkých dvojíc je každý riadok textu priradený nejakému bloku. Tieto bloky popisujú štruktúru dokumentu.

Implementácia výpočtu vzdialeností medzi riadkami nie je veľmi priamočiara, keďže riadky textu sú podľa výpočtov v 3.6 len približne paralelné. Presný postup je popísaný v [3].

Nech E vyjadruje dominantnú vzdialenosť písmen z rôznych riadkov zistenú v 3.3.2 a D dominantnú vzdialenosť medzi písmenami v rámci jedného riadku, ktorú sme zistili tiež v 3.3.2. Potom samotný test, či dva riadky X a Y patria do toho istého bloku má v pseudokóde nasledujúci tvar:

```
if(rozdiel uhlov pootočenia X a Y < Q &&
   && kolmá vzdialenosť medzi X a Y < U * E
   && rovnobežná vzdialenosť medzi X a Y < V * D)
{
    X a Y patria do toho istého bloku
}
else
{
   X a Y nepatria do toho istého bloku
}
```

kde Q, U, V sú konštanty. Q vyjadruje maximálny prípustný rozdiel uhlov v stupňoch a štandardne má hodnotu *30*, veľkosť konštanty U je štandardne nastavená na *1,4*. V má predvolenú hodnotu *1,6*. Tieto konštanty je možné v programe zmeniť(viď užívateľská príručka, kapitola Nastavenia programu). Hodnoty U a V je vhodné zmeniť napríklad v dokumentoch s veľkými medzerami medzi riadkami. Na obrázku 3.6 sú zobrazené bloky textu vytvorené spomenutým alogritmom.



Obr. 3.6: Príklad vytvorených blokov textu

3.8 Zoraďovanie písmen a riadkov

Po tom, ako pre každé písmeno zistíme, do ktorého predbežného slova patrí, je nutné tieto písmená zoradiť podľa orientácie textu. Taktiež je nevyhnutné, aby boli riadky textu v blokoch zoradené v takom poradí v akom by sa čítali. Dominantný uhol zistený v 3.3.2 je však určený nejednoznačne. Ako príklad dobre poslúži text, ktorý má dominantný uhol 0°. Program nevie rozlíšiť, či je tento text "dole hlavou" alebo nie. Túto informáciu musí dodatočne poskytnúť užívateľ.

Predpokladajme teda, že poznáme skutočný uhol otočenia textu, ktorý je z rozsahu [0, 360]. Z uhlu kolmého na tento uhol a pozície stredu písmena vypočítame koeficient b priamky, ktorá prechádza týmto stredom písmena a je kolmá na dominantný smer textu. Podľa týchto koeficientov a znalosti orientácie textu zoradíme písmená v slovách.

Pri triedení zlúčených slov, ktoré by mali reprezentovať riadky textu, môžu nastať pri porovnávaní pozícií dvoch slov tieto dva prípady:

- slová sa nachádzajú v jednom riadku a postupujeme podobne ako pri triedení písmen

- slová sa nachádzajú v rôznych riadkoch a triedime podľa b koeficientov priamky, ktorá je určená dominantným uhlom v texte a štartovacím bodom úsečky prechádzajúcej daným slovom. Slová sa nachádzajú v rôznych riadkoch, ak ich kolmá vzdialenosť vzhľadom k dominantnému uhlu otočenia je väčšia ako *konštanta* $P \times$ *dominantná vzdialenosť medzi riadkami* zistená v 3.3.2.

Predvolená hodnota konštanty P je *0,3*. Túto hodnotu je možné zmeniť v nastaveniach programu. Výsledok zoraďovania slov a riadkov je možné vidieť na obrázkoch 3.7 a 3.8.

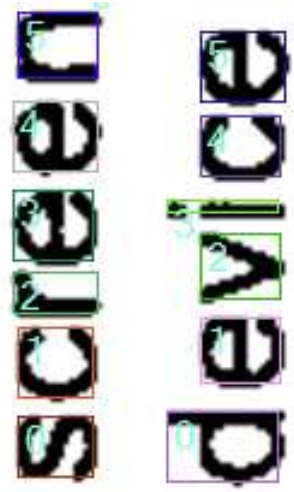
3.9 Zoraďovanie blokov textu

Dôležitý krok, ktorý nasleduje po vytvorení blokov, je ich zoradenie do takého poradia, v akom by to čitateľ s najväčšou pravdepodobnosťou čítal. Vo všeobecnosti nie je možné dosiahnuť stopercentnú úspešnosť správneho zoradenia, ale pri bežných textoch je možné sa k nej priblížiť. V našej práci sme sa zamerali na správne zoradenie textu, v ktorom sa nachádzajú stĺpce, pričom hĺbka zanorenia stĺpcov v texte je ľubovoľná. Špeciálne elementy

(a) nepootočený dokument

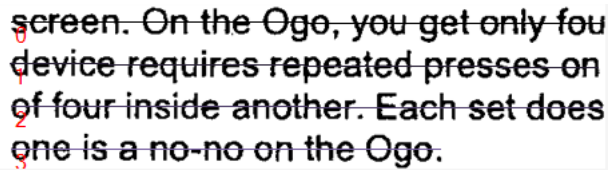
(b) dokument otočený o 180°

(c) dokument otočený o −90°

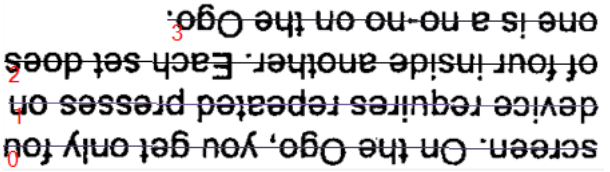


(d) dokument otočený o 90°

Obr. 3.7: Zoraďovanie písmen v riadku



(a) nepootočený dokument



(b) dokument otočený o 180°

Obr. 3.8: Zoraďovanie riadkov v blokoch textu

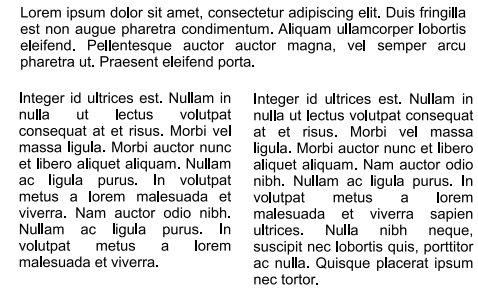
dokumentu ako hlavičky a päty, číslovanie strán a nadpisy pre zjednodušenie chápeme ako súčasť textu. Na obrázku 3.9 sú zobrazené rozpoznávané usporiadania blokov.

Zoradenie blokov do správneho poradia je dôležité hlavne pri ukladaní výsledného dokumentu do formátu, ktorý nepodporuje stĺpce, napríklad textový súbor. Preto sa v našej práci venujeme aj tomuto problému, keďže vytvorený program štandardne umožňuje uloženie dokumentu do textového súboru alebo ako HTML stránku.

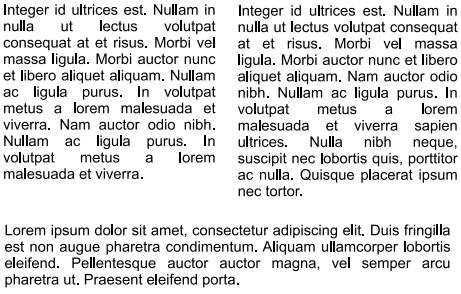
3.9.1 Princíp fungovania algoritmu

Algoritmus zoraďovania blokov je pomerne komplikovaný a technicky náročný na implementáciu. Na vstupe predpokladá postupnosť blokov usporiadanú podľa ich skutočnej šírky od najužšieho po najširší. Definujeme dva typy štruktúry dokumentu, alebo časti dokumentu:

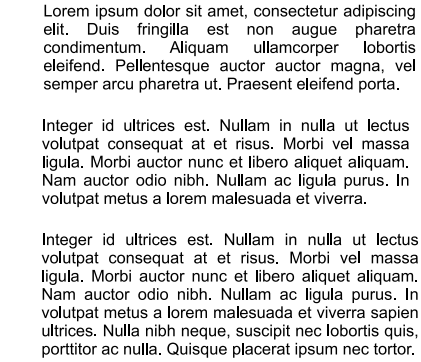
- štruktúra typu *stĺpce*, ktorá môže obsahovať ľubovoľne veľa štruktúr typu *stĺpec*.
- štruktúra typu *finálny text*, ktorá obsahuje samotný vkladaný blok textu.
- štruktúra typu *stĺpec*, ktorá môže obsahovať ľubovoľne veľa štruktúr typu *stĺpce* a *finálny text*.



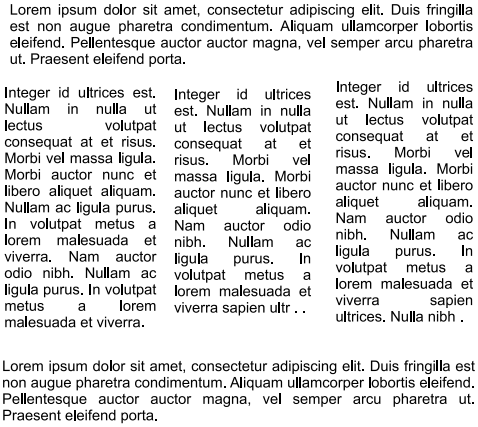
(a) ľubovoľný počet stĺpcov pod spájacím blokom



(b) ľubovoľný počet stĺpcov nad spájacím blokom



(c) ľubovoľný počet blokov pod sebou



(d) ľubovoľný počet stĺpcov medzi blokmi

Obr. 3.9: Rozpoznávané typy štruktúr dokumentu

Algoritmus prejde zoradenú postupnosť na vstupe a každý blok z postupnosti vloží do počiatočnej štruktúry. Predpokladáme, že dokument má stĺpcovú štruktúru a preto je počiatočná štruktúra práve štruktúra typu *stĺpce*, ktorá neobsahuje žiadny *stĺpec*. Po spracovaní vstupu získame konečnú štruktúru dokumentu. Poradie blokov získame jej priechodom metódou in-order. Na obrázku 3.10 je možné vidieť príklad výsledného usporiadania blokov.

Obr. 3.10: Príklad zoradenia blokov textu. Keďže sa predpokladá stĺpcová štruktúra textu, číslo strany má poradové číslo 4, pretože vytvára samostatný stĺpec uprostred stĺpcov s hlavným textom.

3.9.2 Príprava vstupu

Každý blok si štandardne uchováva informáciu o pozícii ľavého horného a pravého dolného rohu najmenšieho nepootočeného obdĺžnika, ktorý daný blok obklopuje. Táto informácia je však pri vytváraní štruktúry dokumentu nepostačujúca, keďže v tom istom obdĺžniku sa môže nachádzať nekonečne veľa blokov rôznych skutočných veľkostí. Preto musíme zistiť súradnice ľavého horného a pravého dolného rohu najmenšieho obdĺžnika, ktorý je otočený o rovnaký uhol ako text, ktorý obklopuje. Pred zistením týchto súradníc najprv zistíme skutočnú šírku bloku, z ktorej vypočítame aj jeho výšku. Šírku bloku zistíme prechodom cez zoradené riadky textu, a z nich vyberieme tej najširší. Výšku bloku zistíme pomocou rozdielu b koeficientov priamok, ktoré prechádzajú cez prvú a poslednú položku zoradených riadkov textu v smere otočenia textu. Z výšky bloku a uhlu otočenia textu zistíme s pomocou trigonometrických funkcií aj rohy pootočeného obdĺžnika, ktorý obklopuje daný blok.

Súradnice rohov sú vypočítané s prihliadnutím na skutočnú orientáciu textu. Pri texte, ktorý je otočený o 180° sa teda ľavý horný roh nachádza na mieste pravého dolného a naopak.

3.9.3 Vkladanie bloku do štruktúry typu *stĺpce*

Štruktúra typu *stĺpce* slúži na horizontálne zatriedenie vkladaného bloku s prihliadnutím na orientáciu textu. Ak sa v tejto štruktúre nenachádza žiadny *stĺpec*, vytvorí sa nový a vloží sa do neho aktuálny blok postupom popísaným v 3.9.4. V opačnom prípade nájdeme postupnosť po sebe nasledujúcich stĺpcov, ktoré aktuálny blok zasahuje, t.j. pokrýva minimálne polovicu šírky stĺpca. Ak sa nachádza v tejto postupnosti práve jeden stĺpec, zatriedime blok do tohto stĺpca podľa postupu v 3.9.4. Ak tvoria postupnosť aspoň dva stĺpce, rozdelíme každý z nich na dve časti. V prvej sa nachádzajú všetky časti dokumentu, ktoré sú nad aktuálnym blokom (s prihliadnutím na skutočný uhol otočenia dokumentu) a v druhej sa nachádzajú ostatné. Z hornej časti vytvoríme stĺpec a uchováme v zozname horných častí, podobne postupujeme pre dolné časti. Z týchto zoznamov vytvoríme dve štruktúry typu stĺpce. Stĺpce zasiahnuté aktuálnym blokom odstránime a namiesto nich vložíme jeden stĺpec skladajúci sa z troch častí - štruktúra typu stĺpce obsahujúca vyššie spomenuté horné časti, aktuálny blok, štruktúra stĺpce obsahujúca dolné časti.

Pri rozdeľovaní stĺpcov sa môže stať, že aktuálny blok zasahuje do po-

ložky typu stĺpce. Tú je potom nutné rozdeliť popísaným algoritmom, pričom sa delia všetky stĺpce tejto štruktúry.

3.9.4 Vkladanie bloku do štruktúry typu *stĺpec*

Štruktúra typu *stĺpec* slúži na vertikálne zatriedenie vkladaného bloku opäť s prihliadnutím na orientáciu textu. Ak sa v stĺpci nenachádza žiadna položka, vytvorí sa nová položka typu *finálny text* s aktuálnym blokom. Ak sa v stĺpci nachádzajú nejaké položky a aktuálny blok je možné zatriediť tak, že nezasahuje ani do jednej z nich, t.j. že sa "zmestil do medzery", tak sa vytvorí nový *finálny text* na nájdenej pozícii.

Ak vkladaný blok zasahuje do nejakej položky, tak je v prípade nutnosti táto položka skonvertovaná na položku typu *stĺpce*. Do tejto položky sa pokúsime pridať nový blok, pričom za úspešné považujeme pridanie len vtedy, ak sa vytvorí v položke (štruktúra typu *stĺpce*) nový *stĺpec*. V opačnom prípade by došlo k zacykleniu algoritmu a je nutné zatriediť vkladaný blok do zoznamu položiek v *stĺpci* tentokrát na základe pozícií stredov položiek.

3.10 Vytváranie slov

Vytváranie predbežných slov sme popísali v 3.5. Tento spôsob vytvárania slov nezaručuje správne výsledky ani pri zvolení optimálnej hodnoty konštanty C. Príklad, kedy vytváranie slov touto metódou zlyhá, sú slová v dokumente písané veľkým fontom v porovnaní s ostatným textom. Vzdialenosti stredov susedných písmen v takýchto slovách sú oveľa väčšie ako vzdialenosti stredov písmen v slovách s bežnou veľkosťou fontu. Taktiež sa môže stať, že šírka medzery medzi slovami v bežnom texte je menšia ako vzdialenosť stredov veľkých písmen.

Lepší spôsob, ako určovať slová, je zamerať sa na medzery medzi písmenami. Šírka medzier v slove napísanom ľubovoľne veľkým fontom sa veľmi nemení. To je dôležitý poznatok, na ktorom je založený náš algoritmus vytvárania slov popísaný v 3.10.1.

3.10.1 Algoritmus vytvárania slov

Samotný algoritmus vytvárania slov je pomerne jednoduchý. Nech D je priemerná vzdialenosť medzi písmenami získaná podľa 3.3.2, G je konštanta a S

je aktuálne slovo, potom nasledujúci kód sa vykoná pre každý riadok textu v dokumente:

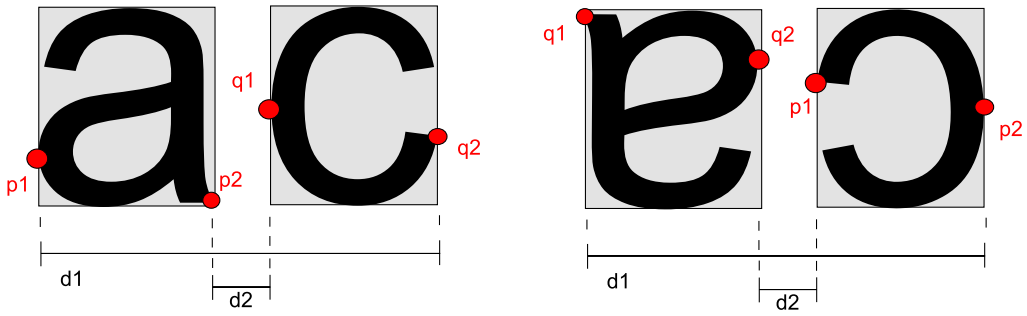
```
S:=prvé písmeno v riadku textu;

foreach(písmeno P a nasledujúce písmeno Q v riadku textu)
{
    p1,p2 := VráťKoncovéBodyPísmena(uhol otočenia textu, P);
    q1,q2 := VráťKoncovéBodyPísmena(uhol otočenia textu, Q);
    if(min(vzdialenosť p2 a q1, vzdialenosť p1 a q2) >= G * D)
    {
       pridaj S do zoznamu slov v riadku;
       S:= prázdne slovo;
    }
    else
       pridaj Q na koniec S;
}
```

Konštanta G má predvolenú hodnotu *0,5* a je ju možné zmeniť v nastaveniach programu. Na obrázku 3.11 je možné vidieť, že funkcia *VráťKoncovéBodyPísmena* vracia koncové body bez ohľadu na skutočné otočenie textu. Preto pri výpočte šírky medzery počítame minimum zo vzdialeností možných medzier. Vzdialenosť bodov sa je v algoritme počítaná ako kolmá vzdialenosť priamok prechádzajúcich týmito bodmi v smere kolmom na dominantný smer textu.

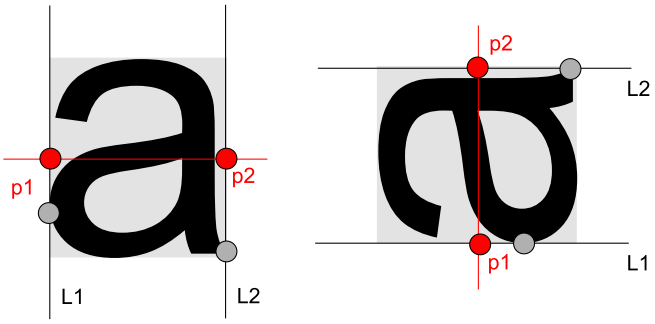
3.10.2 Zistenie pozícií koncových bodov

Problém, ktorý nastáva pri tomto vyššie zmienenom algoritme je ten, že funkcia *VráťKoncovéBodyPísmena* musí pracovať rýchlo. Ak by sme požadovali nájdenie presných koncových bodov v danom smere, funkcia by musela vykonať projekciu cez písmeno v smere kolmom na zadaný smer a pomocou jej profilu hľadať vzdialené čierne pixely, ktoré sú súčasťou písmena. Tento postup je však výpočetne náročný. Projekciu však nie je nutné vykonávať v triviálnych prípadoch, keď je uhol otočenia dokumentu $0°$ alebo $90°$. V takýchto prípadoch použijeme ako koncové body stredy strán obklopujúceho obdĺžnika, ako je ukázané na obrázku 3.12. Tento odhad je pomerne presný vďaka vyššie spomenutému výpočtu vzdialeností medzi bodmi. V skutočnosti s rovnakou úspešnosťou používame tento odhad aj na uhly z rozsahu $[-5°, 5°]$ a $[85°, 95°]$.

Pre zvyšné uhly zistíme koncové body ďalšou vhodnou aproximáciou. V procese označovania oblastí popísanom v 3.2 si budeme pri vytváranom písmene pamätať aj koncové body v smere projekcie $45°$ a $-45°$. Pri vkladaní nového pixelu do písmena zároveň zistíme pomocou b koeficientov priamky,

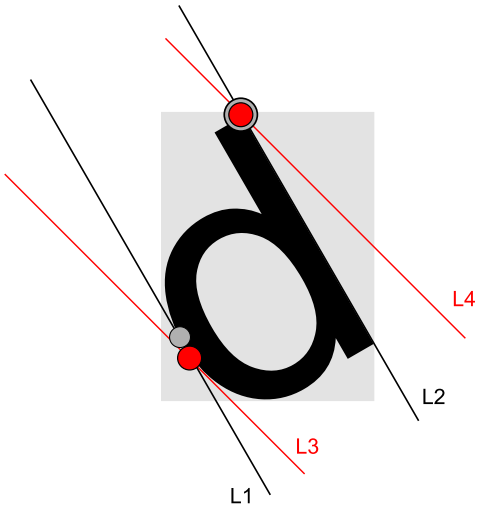


Obr. 3.11: Vzdialenosti medzi písmenami



Obr. 3.12: Triviálne prípady určenia koncových bodov písmena. Na obrázku je možné vidieť, že priamka L1 pretína skutočný sivý koncový bod aj odhadnutý koncový bod p1. Podobne oba koncové body pretína aj priamka L2

či nie je vkladaný pixel koncovým bodom v niektorom zo smerov projekcie. V procese vytvárania slov potom vyberieme podľa uhlu otočenia dokumentu vypočítané koncové body jednej z projekcií. Tieto koncové body sú veľmi dobrým odhadom skutočných koncových bodov, ako ukazuje obrázok 3.13 a navyše, časová zložitosť tohto spôsobu riešenia je O(počet pixelov, ktoré tvoria písmeno).



Obr. 3.13: Na obrázku kde je písmeno "d" pootočené o 30° je vidieť, že pozície získaných červených koncových bodov ležiacich na priamkach L3 a L4 (projekcia pod uhlom −45°) sa takmer zhodujú s pozíciami skutočných koncových bodov, ktoré ležia na priamkach L1 prípadne L2 kolmých na smer textu.

Kapitola 4

Rozpoznávanie písmen

Tretia fáza rozpoznávania textu je samotné rozpoznávanie písmen a je najdôležitejšou časťou spracovania dokumentu. Úspešnosť rozpoznania jedného písmena je kritická, keďže vytvorený dokument, ktorý obsahuje veľa chýb, je nepoužiteľný, pretože oprava týchto chýb bude trvať takmer rovnaký čas ako ručné prepísanie skenovaného dokumentu. Preto je nutné použiť spoľahlivý spôsob rozpoznávania písmen. Zďaleka najpoužívanejší a čo do úspešnosti jeden z najlepších spôsobov rozpoznávania je použitie neurónových sietí. V našej práci sme použili práve tento spôsob, konkrétne viacvrstvovú neurónovú sieť s učiacim algoritmom spätného šírenia chyby (backpropagation).

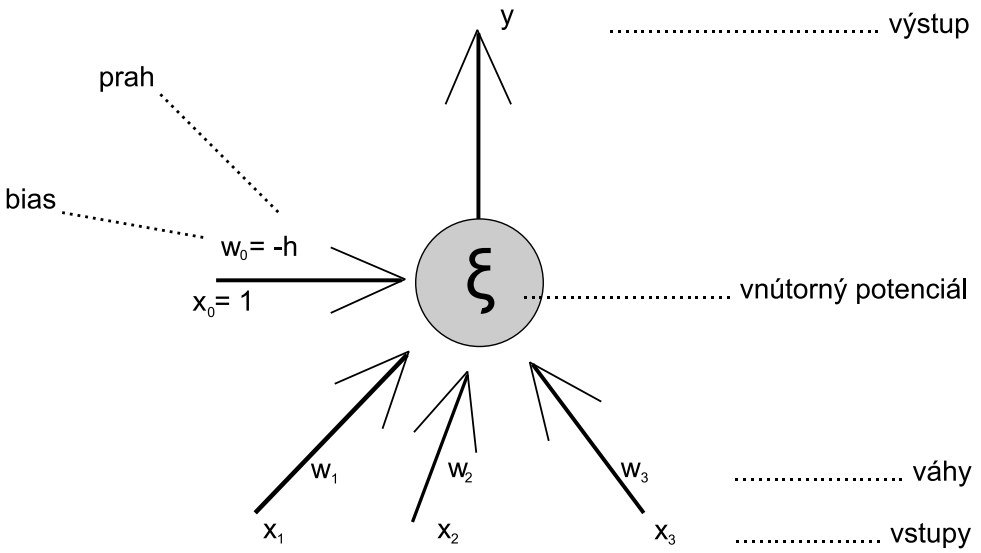
4.1 Neurón

Základom matematického modelu neurónovej siete je *formálny neurón*, ktorého štruktúra znázornená na obrázku 4.1 je zjednodušenou formou biologického neurónu[7].

Formálny neurón má n obecne reálnych *vstupov* $x_1, ..., x_n$. Vstupy sú ohodnotené *váhami* $w_1, ..., w_n$. Zvážená suma vstupných hodnôt predstavuje *vnútorný potenciál* neurónu:

$$\xi = \sum_{i=1}^{n} w_i x_i. \tag{4.1}$$

Hodnota vnútorného potenciálu ξ po dosiahnutí takzvanej *prahovej* hodnoty h indukuje *výstup (stav)* neurónu y. Nelineárny nárast výstupnej hodnoty $y = \sigma(\xi)$ pri dosiahnutí prahovej hodnoty potenciálu h je daný tzv. *aktivačnou (prenosovou) funkciou* σ. Najjednoduchším typom aktivačnej funkcie



Obr. 4.1: Formálny neurón

je tzv. *ostrá nelinearita*, ktorá má tvar:

$$\sigma(\xi) = \begin{cases} 1 & ak \xi \geq h \\ 0 & ak \xi < h. \end{cases} \tag{4.2}$$

Formálnou úpravou dosiahneme to, že funkcia σ bude mať nulový prah a vlastný prah neurónu so záporným znamienkom chápeme ako váhu, tzv. *bias* $w_0 = -h$ ďalšieho formálneho vstupu $x_0 = 1$ s konštantnou jednotkovou hodnotou, ako je naznačené na obrázku 4.1. Matematická formulácia funkcie neurónu je potom daná vzťahom:

$$\sigma(\xi) = \begin{cases} 1 & \text{ak } \xi \geq 0 \\ 0 & \text{ak } \xi < 0. \end{cases}, \qquad \text{kde } \xi = \sum_{i=0}^{n} w_i x_i. \tag{4.3}$$

Medzi iné používané funkcie patrí:

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}} \qquad \text{štandardná sigmoida} \tag{4.4}$$

$$\sigma(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}} \qquad \text{hyperbolický tangens} \tag{4.5}$$

$$\tag{4.6}$$

4.2 Neurónová sieť

Neurónová sieť sa skladá z formálnych neurónov, ktoré sú vzájomne prepojené tak, výstup neurónu je vstupom všeobecne viacerých neurónov. Počet neurónov a ich vzájomné prepojenie určuje tzv. *architektúru* neurónovej siete. Z hľadiska využitia rozlišujeme v sieti *vstupné, skryté a výstupné* neuróny. Stavy všetkých neurónov v sieti určujú tzv. *stav neurónovej siete* a váhy všetkých spojov predstavujú *konfiguráciu* neurónovej siete.

Neurónová sieť sa v čase vyvíja, menia sa prepojenia a stavy neurónov, menia sa váhy. V súvislosti s touto zmenou je vhodné *dynamiku* neurónovej siete rozdeliť do troch dynamík a uvažovať tri režimy práce siete:

- *organizačný* (zmena topológie)
- *aktívny* (zmena stavu)
- *adaptívny* (zmena konfigurácie)

Organizačná dynamika špecifikuje architektúru siete a jej prípadnú zmenu. Rozlišujeme dva typy architekúry:

- *cyklická* - v sieti existuje skupina neurónov zapojená do kruhu
- *acyklická* - neuróny je vždy možné rozdeliť do tzv. *vrstiev*, ktoré sú usporiadané (napríklad nad sebou) tak, že spoje medzi neurónmi vedú len z nižších vrstiev do vyšších a vo všeobecnosti môžu preskočiť jednu alebo viac vrstiev.

Špeciálny prípad acyklickej architektúry je *viacvrstvová neurónová sieť*. V tejto sieti je nultá (dolná), tzv. vstupná vrstva tvorená vstupnými neurónmi a posledná (horná) vrstva, tzv. výstupná vrstva sa skladá z výstupných neurónov. Ostatné, skryté vrstvy sú zložené zo skrytých neurónov. Vrstvy číslujeme od nuly, ktorá odpovedá vstupnej vrstve. Tú potom nepočítame do počtu vrstiev siete. V topológii viacvrstvovej siete sú neuróny spojené so všetkými neurónmi bezprostredne nasledujúcej vrstvy. Preto je možné zadať architektúru takejto siete počtom neurónov v jednotlivých vrstvách.

Aktívna dynamika špecifikuje *počiatočný stav* siete a spôsob jeho zmeny v čase pri pevnej topológii a konfigurácii. V aktívnom režime sa na začiatku nastavia stavy vstupných neurónov na tzv. *vstup siete* a ostatné neuróny sú v uvedenom počiatočnom stave. Všetky možné vstupy, alebo stavy siete tvoria tzv. *vstupný alebo stavový priestor* neurónovej siete. Po inicializácii stavov

prebieha samotný výpočet. Všeobecne sa uvažuje spojitý vývoj stavu siete v čase a hovorí sa o *spojitom modeli*, Väčšinou sa však predpokladá diskrétny čas. V každom časovom kroku je vybraný jeden alebo viac neurónov, ktoré aktualizujú svoj stav na základe svojich vstupov. Stav výstupných neurónov je tzv. *výstupom* neurónovej siete. Obvykle sa uvažuje taká adaptívna dynamika, že výstup siete je po istom čase konštantný a neurónová sieť realizuje nejakú funkciu na vstupnom priestore, tj. ku každému vstupu siete vypočíta práve jeden výstup. Túto funkciu nazývame *funkcia neurónovej siete*.

Adaptívna dynamika špecifikuje *počiatočnú konfiguráciu* siete a spôsob akým sa menia váhy v čase. Všetky možné konfigurácie tvoria tzv. *váhový priestor* neurónovej siete. Cieľom adaptácie je nájsť takú konfiguráciu siete vo váhovom priestore, ktorá by v aktívnom režime realizovala predpísanú funkciu. Ak sa aktívny režim používa k výpočtu funkcie siete pre daný vstup, potom adaptívny režim slúži k *učeniu* tejto funkcie. Existuje mnoho učiacich algoritmov. Najznámejší a najpoužívanejší je *backpropagation* pre viacvrstvovú neurónovú sieť[7]. Učenie siete predstavuje zložitý nelineárny optimalizačný problém, ktorého riešenie je časovo veľmi náročné.

Požadovaná funkcia je zadaná tzv. *tréningovou množinou* dvojíc vstup / výstup siete (tzv. *tréningový vzor*). Tento spôsob popisu požadovaného chovania siete modeluje učiteľa, ktorý pre vzorové vstupy siete informuje adaptívny mechanizmus o správnom výstupe siete. Tomuto typu adaptácie sa preto hovorí *učenie s učiteľom*. Niekedy učiteľ hodnotí kvalitu výstupu siete pre daný vstup pomocou známky, ktorá je zadaná miesto požadovanej hodnoty výstupu siete. V takom prípade ide o *klasifikované učenie*.

Iným typom adaptácie je *samoorganizácia*. V tomto prípade obsahuje tréningová množina len vstupy siete. Tomuto spôsobu adaptácie sa hovorí *učenie bez učiteľa*.

4.2.1 Backpropagation

Učiaci algoritmus backpropagation sa používa približne v 80% všetkých aplikácií neurónových sietí. Cieľom algoritmu je minimalizácia chyby siete

$$E(\mathbf{w}) = \sum_{k=1}^{p} E_k(\mathbf{w}) \tag{4.7}$$

kde p je počet tréningových vzorov, $\boldsymbol{w}$ je konfigurácia siete, $E_k w$ je parciálna chyba siete vzhľadom ku k-temu tréningovému vzoru a je úmerná súčtu moc-

nín odchýliek skutočných hodnôt výstupu siete pre vstup k-teho tréningového vzoru od odpovedajúcich požadovaných hodnôt výstupu u tohto vzoru:

$$E_k(\mathbf{w}) = \frac{1}{2}\sum_{j\in Y} E_k(y_j(\mathbf{w}, \mathbf{x}_k) - d_{kj})^2 \tag{4.8}$$

kde Y je množina výstupných neurónov. Chyba môže byť počítaná po predložení každého vzoru, alebo po predložení všetkých vzorov tréningovej množiny. Predloženie všetkých vzorov sieti nazývame *epocha*. V tejto práci nebudeme uvádzať presný popis algoritmu.

4.3 Trénovanie siete

Cieľom trénovania siete je nájsť také ohodnotenie váh v sieti, ktoré bude realizovať predpísanú funkciu neurónovej siete. V našom prípade požadujeme, aby funkcia siete pre každý vstup reprezentujúci nakreslené písmeno vrátila práve jeden výstup reprezentujúci dané písmeno. To znamená, že veľkosť výstupnej vrstvy našej siete musí byť práve počet trénovaných znakov. Aby sme dosiahli čo najväčšiu úspešnosť rozpoznania znaku, musí byť tréningová množina čo najväčšia. To znamená, že pre každý trénovaný znak musí poskytovať čo najviac rôznych vzorov. V prípade rozpoznávania písmen je teda vhodné použiť takú tréningová množinu, ktorá bude napríklad obsahovať ako vstupy písmená nakreslené rôznymi fontami.

4.3.1 Tréningová množina

Pred trénovaním siete je nutné vytvoriť tréningovú množinu, v ktorej sa nachádzajú tréningové vzory. Pred jej vytvorením vyberieme v nami vytvorenom programe (viď 4.3.3) sadu znakov, ktorá má byť trénovaná. Štandardne by sa mali trénovať tieto znaky (v 4.5 však ukážeme, ktoré znaky sa smú trénovať a ktoré nie):

a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 ( ) / \ [ ] { } *? !* + − ∗ @ # $ % & § . ; : ,

Nový vzor vytvoríme tak, že špecifikujeme výstup, t.j. jeden znak z trénovanej sady znakov. K nemu vygenerujeme obrázok zadaných rozmerov, v

ktorom je tento znak nakreslený. Veľkosť nakresleného znaku je vždy maximálna možná vzhľadom k rozmerom generovaného obrázka. Z tohto obrázka následne vyextraktujeme určitý počet nejakých vlastností, ktoré sú vyjadrené ako reálne čísla. Tieto vlastnosti sú označené ako vstup tréningového vzoru.

Najjednoduchší prípad extrahovaných vlastností[1] je postupnosť intenzít pixelov vytvorená rozložením riadkov obrázka. Výhodou je rýchlosť vyextrahovania vlastností avšak tieto vlastnosti sú veľmi náchylné na šum a aj na malé pootočenie písmena v obrázku. K ďalším typicky extrahovaným vlastnostiam patria kontúry písmena charakterizované úsečkami. Problémom pri tomto type extrakcie je to, že pri reálnych dátach sa nemusí zhodovať počet nájdených úsečiek charakterizujúcich kontúru s počtom úsečiek trénovaných písmen. V našej práci sme použili na extrakciu vlastností diskrétnu kosínovú transformáciu (DCT).

Diskrétna kosínová transformácia

DCT je špeciálny prípad Diskrétnej Fourierovej Transformácie, ktorý pracuje len s reálnymi zložkami komplexných čísel. DCT je aplikovaná na vstupný obrázok. Jej výsledkom sú koeficienty v matici rovnakej veľkosti ako je vstup. Najjednoduchšia podoba výpočtu koeficientov matice x rozmerov $N_1 \times N_2$ je nasledovná:

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos\big[\frac{\pi}{N_1}\big(n_1 + \frac{1}{2}\big)k_1\big] cos\big[\frac{\pi}{N_2}\big(n_2 + \frac{1}{2}\big)k_2\big] \qquad (4.9)$$

$$k_1 = 0, ..., N_1 - 1$$
$$k_2 = 0, ..., N_2 - 1$$

Výhodou tejto transformácie je fakt, že hlavné charakteristiky obrázka sú uchované len v niekoľkých koeficientoch. Navyše sa tieto koeficienty veľmi nemenia pri posunutí a pootočení obrázka. To umožňuje rozpoznávať aj pootočený text a zároveň zmenšiť veľkosť vstupnej vrstvy neurónovej siete a urýchliť tak proces trénovania.

[1] angl. features

4.3.2 Architektúra neurónovej siete a parametre učenia

Pred samotným tréningom siete je nutné zvoliť vhodnú architektúru siete tak, aby odpovedala zložitosti riešeného problému, tj. počtu tréningových vzorov, ich vstupov, výstupov a štruktúre vzťahov, ktoré popisujú. Je zrejmé, že malá sieť nedokáže riešiť komplikovaný problém. Pri učení pomocou algoritmu backpropagation sa príliš malá sieť zastaví v plytkom lokálnom minime chybovej funkcie[7] a je potrebné doplniť topológiu o ďalšie skryté neuróny. Na druhú stranu, bohatá architektúra umožní nájsť globálne minimum chybovej funkcie. Táto konfigurácia však obvykle príliš zohľadňuje tréningové vzory a pre neučené vzory dáva chybné výsledky, t.j. zle generalizuje. Tomuto presnému naučeniu sa tréningovej množiny sa hovorí *preučenie* siete.

Okrem architektúry siete je potrebné zvoliť vhodné parametre učenia metódou backpropagation. Parametre ktoré sa dajú v programe (viď 4.3.3) nastaviť je rýchlosť učenia, perióda pridávania a veľkosť šumu pridávaného do siete. Taktiež je možné zadať veľkosť chyby siete, pri ktorej sa trénovanie zastaví, prípadne maximálny počet epôch, ak chyba siete neklesne pod zadanú hranicu.

V našej práci sme natrénovali neurónovú sieť na štandardnú sadu znakov (viď 4.3.1). Každý znak je nakreslený do obrázku veľkosti 20×20. Z tohto obrázku je pomocou DCT vyextrahovaných 250 koeficientov z ľavého horného rohu transformovaného obrázku postupne diagonálne smerom ku stredu. Spôsob vyberania je daný tým, že v tomto rohu sa nachádzajú koeficienty, ktoré uchovávajú najviac informácií o obrázku. Na vytvorenie trénovacej množiny sme použili 13 štandardných fontov. V sieti sme použili jednu skrytú vrstvu s 200 skrytými neurónmi. Ako prahové funkcie boli použité štandardné sigmoidy. Úspešnosť tejto siete po natrénovaní je viac ako 97% na tréningovej množine. Na reálnych dátach dostatočnej kvality je úspešnosť rozpoznania písmena približne 75%.

4.3.3 Program OCR trainer

Súčasťou vytvoreného balíčku programov je program, ktorý slúži na trénovanie rozpoznávania znakov. Program umožňuje zvoliť, ktoré znaky sa majú trénovať a tak isto umožňuje nastaviť architektúru siete a parametre tréningu. Tréningová množina je vytvorená na základe užívateľom vyznačených fontov. V programe je možné testovať aktuálnu úspešnosť siete na zvolených

fontoch písma a zvolenej testovacej množine. Taktiež umožňuje automatické ukladanie neurónovej siete počas trénovania. Podrobný opis programu sa nachádza v užívateľskej príručke kapitola B.2.

4.4 Rozpoznanie písmena

Po predložení vygenerovaného alebo reálneho vstupu sieti sa podľa z výstupov výstupných neurónov siete určí, ktorý výstupný neurón (ktorý odpovedá práve jednému písmenu z trénovanej sady) má najvyššiu hodnotu na výstupe. Znak, ktorý tento neurón reprezentuje sa považuje za rozpoznaný znak. Po rozpoznaní znaku však musíme vyriešiť niekoľko problémov ktoré popíšeme v nasledujúcich podkapitolách.

4.5 Určenie malých a veľkých písmen v slove

Pri niektorých písmenách štandardnej sady trénovaných písmen ako *c, o, p, s, v, w, z* je problém rozpoznať, či sú v skutočnom texte písané tieto písmená veľkým písmom alebo nie. Taktiež pri trénovaní vnáša tento typ písmen do siete konštantnú chybu a sieť preto nie je možné natrénovať na sto percent. Preto by tréningová sada mala obsahovať len jednu verziu z každého takého písmena.

Ak sieť vráti ako výsledok nejaké písmeno zo spomínaných písmen, tak program musí podľa skutočnej výšky písmena určiť, či sa jedná o veľké alebo malé písmeno. Skutočnú výšku písmena zistíme podľa koncových bodov určených v 3.10.2. Táto výška v pixeloch však neurčuje to či je písmeno malé alebo veľké. To určíme pomocou štatistických dát zozbieraných v jednom riadku textu. Konkrétne, písmeno budeme považovať za veľké (C,O,S ...), ak:

$$|d| < 0,1 * W \tag{4.10}$$

kde d je *skutočná výška písmena v pixeloch - výška najvyššieho písmena v riadku*, W je priemerná výška znakov v riadku s bežnou veľkosťou, t.j. znakov, ktoré nepatria do množiny nízkych znakov T: $\tilde{}$. ,

Táto podmienka pracuje na bežných dokumentoch veľmi spoľahlivo. V dokumentoch, ktoré v rámci jedného riadku obsahujú text písaný rôznymi veľkosťami fontu, sa však môže stať, že bude zle odhadnutá veľkosť písmena.

4.6 Určenie niektorých znakov interpunkcie

Pri rozpoznávaní textu je veľmi dôležité správne rozpoznať znaky interpunkcie. Znaky ako :;!? sieť s veľkou pravdepodobnosťou správne rozpozná. Štandardne trénovaná sada znakov však neobsahuje apostrof a uvodzovky. Apostrof nie je obsiahnutý v sade kvôli veľkej podobnosti s čiarkou. Uvodzovky sa skladajú z dvoch častí, ktoré sa podobajú na apostrof, prípadne čiarku a v procese analýzy štruktúry dokumentu nedochádza k ich zlúčeniu do jedného znaku. Tieto podobnosti znakov, spolu s ich skutočnou výškou a zo znalosťou ich pozície vzhľadom k priamke prechádzajúcej stredom riadku, využijeme k správnemu určeniu týchto znakov v texte:

```
if (výška znaku L <= I * H)
{
    if (L nie je v T)
        L := čiarka;
}
else
{
    if (L je čiarka)
        L := 'r';
}

if (L je nad priamkou prechádzajúcou stredom riadku a L je čiarka alebo bodka)
    L := apostrof;
```

kde H je priemerná výška takých znakov v slove, ktoré nepatria do T a I je konštanta s predvolenou hodnotou *0,5*. Hodnotu tejto konštanty je možné v nastaveniach programu zmeniť.

4.7 Určenie zameniteľných znakov

Ďalší problém pri rozpoznávaní znakov je ten, že niektoré dvojice, prípadne k-tice, písmen, sú si veľmi podobné. Medzi tieto k-tice patria:

- *O* a *0*
- *l* a *1* a *I*

Pre správne určenie, ktorý znak z danej k-tice má byť považovaný za rozpoznaný znak, je nutné použiť niektoré štatistické údaje o danom slove. Medzi tieto štatistické údaje patrí pomer p počtu číslic k počtu znakov v slove, pomer q počtu číslic a znakov podobných nule (o,O) k počtu znakov v slove a pomer r počtu veľkých písmen k počtu znakov v slove. Potom:

- ak $p > U_1$, tak l je zmenené na *1*
- ak $p < U_2$, tak *1* je zmenená na l
- ak $p < V_1$, tak O je zmenené na *0*
- ak $q > V_2$, tak *0* je zmenená na O
- ak $r > W$, tak l je zmenené na I

kde U_1, U_2, V_1, V_2, W sú konštanty. Ich hodnoty je možné zmeniť v nastaveniach programu.

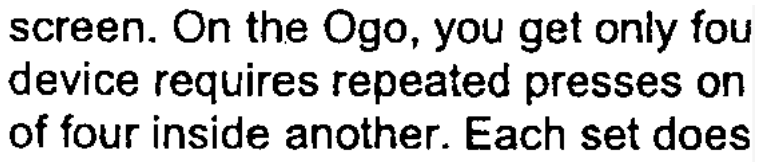
4.8 Pomocné neurónové siete

Pri rozpoznávaní znakov musí jedna sieť vedieť rozpoznať veľa znakov kreslených rôznymi fontami a je nemožné natrénovať nami používanou metódou túto sieť tak, aby mala úspešnosť rozpoznania znaku 100%. Navyše sa môže stať, že koeficienty DCT obrázkov s rôznymi písmenami môžu byť podobné a sieť potom nevie presne odlíšiť tieto písmená. Preto v našej práci používame pomocné neurónové siete, ktoré sú natrénované len na vybrané navzájom podobné písmená. Tréningové vzory sú vytvárané typicky z obrázkov väčších rozmerov ako v hlavnej sieti a počet vyextrahovaných koeficientov sa taktiež líši. Najbežnejšie problémové množiny písmen sú:

- i,j,l,t
- o,c,e,a,n,D,G,Q
- E,F
- i a bodkočiarka

Každá pomocná sieť natrénovaná na dané množiny písmen má svoju prioritu použitia. Sieť je použitá vtedy, ak aktuálne rozpoznané písmeno patrí do množiny písmen, ktoré daná sieť rozpoznáva. Priorita použitia je daná jej názvom v adresári s pomocnými sieťami. Cestu k tomuto adresáru je možné v nastaveniach programu zmeniť.

Celková úspešnosť rozpoznania písmena sa s použitím pomocných neurónových sietí dramaticky zvýšila až na úroveň 90%. Táto úspešnosť nie je konečná. Na jej vylepšenie sme použili metódu popísanú v nasledujúcej kapitole. Na obrázku 4.2 uvádzame príklad rozpoznaného textu s použitím doposiaľ zmienených algoritmov.



(a) Časť naskenovaného obrázku.

scraen. On the Ogo, you get only fou

device requires repeated presses on

of four inside anothar. Each get does

(b) Výsledný rozpoznaný text, ktorý obsahuje tri chyby.

Obr. 4.2: Príklad rozpoznaného textu bez kontroly textu

Kapitola 5

Automatická kontrola rozpoznaného textu

Doposiaľ sme sa venovali rozpoznaniu samotného písmena, ktorého úspešnosť je približne 90%. Štatisticky to znamená, že pri priemernej dĺžke slova 6 písmen sa v každom druhom slove v texte nachádza chyba. Za tohto predpokladu sa v práci pokúšame zvýšiť celkovú úspešnosť rozpoznania textu tým, že na rozpoznaných slovách vykonáme kontrolu hláskovania[1] a v prípade jednoznačnej chyby v slove nahradíme slovo tým správnym slovom zo slovníka. Na to aby sme to mohli vykonať, potrebujeme vhodný slovník a vhodnú metódu na vyhľadanie podobných slov.

5.1 Slovník

Výber vhodného slovníka je v procese automatickej kontroly textu kritický. Slovník musí obsahovať taký počet slov, aby označil čo najviac bezchybných slov ako skutočne správne slová a aby označil čo najviac chybných slov ako nesprávne a ak je možné, zamenil dané slovo za to ktoré užívateľ očakáva. Slovník by nemal obsahovať v danom jazyku málo používané slová, napríklad odborné slová, archaizmy atď. V našej práci sme použili slovník, ktorý obsahuje vyše 80000 anglických slov.

[1] angl. spell check

5.2 Priebeh kontroly

Každé slovo textu je vyhľadané v slovníku. Ak sa v slovníku nachádza, je toto slovo označené ako správne slovo. Ak sa tam nenachádza, program nájde zadaný počet najpodobnejších slov. Podobnosť slov je možné počítať viacerými spôsobmi. Medzi najčastejšie používané miery podobnosti patrí Hammingova a Levenshteinova vzdialenosť.

5.2.1 Hammingova vzdialenosť

Hammingova vzdialenosť dvoch rovnako dlhých slov je definovaná ako počet písmen ktoré sa na korešpondujúcich pozíciách v týchto slovách líšia [8]. Ako príklad uvedieme anglické slová *number* a *member*, ktorých Hammingova vzdialenosť je 2.

Použitie tohto typu vzdialenosti je vhodné vtedy, keď nepotrebujeme hľadať slová inej dĺžky ako zadané slovo, tj. vtedy, keď je skenovaný obrázok veľmi kvalitný a v texte sa nevyskytujú spojené písmená.

5.2.2 Levenshteinova vzdialenosť

Levenshteinova vzdialenosť je daná ako minimálny počet operácií potrebných na transformáciu jedného reťazca na druhý, kde operácia je vloženie, odstránenie alebo nahradenie jedného písmena[9]. Štandardný algoritmus na výpočet tejto vzdialenosti používa dynamické programovanie a v tejto práci ho nebudeme rozoberať.

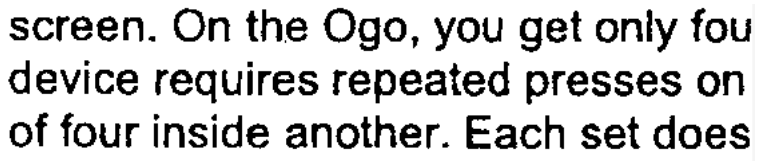
Tento typ vzdialenosti je vhodné použiť pri štandardnej kvalite dokumentov, kde sa môže stať, že dve písmená sú zlúčené do jedného. Pri takýchto slovách je však menej pravdepodobné, že najpodobnejšie slovo bude práve to, ktoré bolo v obrázku, pretože vzdialenosť tohto slova je kvôli zlúčeniu dvoch písmen minimálne 2. To znamená, že ak sa v slovníku nachádza slovo so vzdialenosťou 1, je toto slovo vybrané ako najpodobnejšie.

5.2.3 Automatické opravy

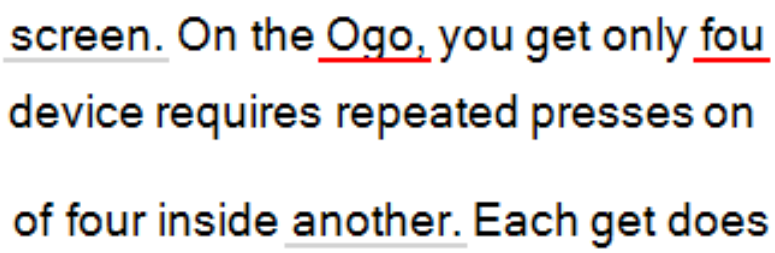
Automatická oprava chybného slova je možná len vtedy, ak je jednoznačne určené najpodobnejšie slovo. V našej práci definujeme jednoznačnosť ako existenciu práve jedného slova, ktorého vzdialenosť od chybného slova je 1. Takéto chybné slovo je potom nahradené nájdeným slovom. Táto definícia

jednoznačnosti umožňuje opravovať len slová s práve jedným zameneným písmenom, ku ktorým existuje jedno najpodobnejšie slovo.

Použitie automatickej kontroly v našom prípade zvýšilo úspešnosť rozpoznania písmena na 95%. Úspešnosť rozpoznania slova ako celku je približne 80%. Na obrázku 5.1 je zobrazený výsledok rozpoznania textu s použitím kontroly textu.



(a) Časť naskenovaného obrázku.



(b) Výsledný rozpoznaný text, ktorý obsahuje tri chyby.

Obr. 5.1: Príklad rozpoznaného skontrolovaného textu. Tento výsledok v porovnaní s tým na obrázku 4.2 obsahuje len jednu chybu. Červenou sú podčiarknuté slová, ktoré neboli nájdené v slovníku a sivou tie, ktoré boli chybné, ale v slovníku pre ne existuje jednoznačne určené najpodobnejšie slovo. Slovo zo slovníka bolo vypísané na výstup. Slovo "get" v treťom riadku je síce rozpoznané nesprávne, ale v slovníku sa takéto slovo nachádza a preto bola v tomto prípade kontrola neúčinná.

Kapitola 6

Záver

Cieľom práce bolo implementovať konverziu skenovaného obrázka s textom do editovateľnej podoby. Priemerná úspešnosť našej implementácie konverzie je 95%, ak počítame počet správne rozpoznaných písmen v texte a približne 80%, ak berieme do úvahy počet správne rozpoznaných slov. Výsledok použitia automatickej kontroly textu mal za následok zlepšenie úspešnosti približne 5%. Táto hodnota predčila naše očakávania. Pôvodný zámer však bol rozpoznávať text naskenovaný pod ľubovoľným uhlom. Tu sme sa spoliehali na diskrétnu kosínovú transformáciu, ktorej koeficienty by sa nemali príliš meniť na pootočenom vstupe. Počas trénovania neurónovej siete sme však zistili, že na túto vlastnosť sa nemôžeme spoliehať pri našej architektúre siete a použitých tréningových vzoroch. Tento nedostatok nepovažujeme za vážny, nakoľko bežné pootočenie skenovaného dokumentu činí v drvivom množstve prípadov maximálne $2°$. Vyššie spomenuté výsledné hodnoty úspešnosti boli namerané len na minimálne pootočených obrázkoch (uhol otočenia $(-5°, 5°)$ a $(85°, 95°)$) a sú dostatočne vysoké na to, aby sme mohli považovať náš softvér za použiteľný v praxi.

Za úspešnú taktiež považujeme analýzu štruktúry dokumentu. Tá je invariantná na rotáciu a pri štandardných dokumentoch so stĺpcovou štruktúrou textu funguje veľmi dobre. Pri analýze sme nezanedbateľné úsilie venovali implementácii zoraďovania odstavcov do takého poradia, v akom by sa čakalo, že ich bude čitateľ čítať. Tento zdanlivo triviálny krok patril k implementačne náročnejším častiam projektu. Nami vytvorený algoritmus zoraďovania funguje veľmi spoľahlivo na všetkých bežných štruktúrach rozloženia dokumentu a s jeho spôsobom implementácie sme spokojní.

Patričné úsilie sme venovali aj vytvoreniu grafického používateľského ro-

zhrania. Grafické rozhranie hlavného programu je koncipované tak, aby v prvom rade umožnilo používateľovi čo najjednoduchšie použitie programu. Pre skúseného používateľa však ponúka množstvo nastavení. Program na trénovanie rozpoznávania písmen je zameraný na prehľadné vytváranie, trénovanie a testovanie neurónových sietí. S použitím automatického zálohovania je vhodný aj na časovo veľmi náročné trénovanie veľkých sietí.

V práci by sme chceli vylepšiť celkovú úspešnosť rozpoznávania aspoň na 99%, ako sú toho schopné profesionálne OCR programy. Taktiež nie sme spokojní s rýchlosťou konverzie. Tá je bohužiaľ pomerne malá a je priamym dôsledkom spôsobu analýzy štruktúry dokumentu metódou "zdola nahor". Užívateľské rozhranie hlavného programu by sme rozšírili o plnohodnotný editor výsledného textu a nakoniec by sme chceli rozpoznávať textové dokumenty obsahujúce tabuľky, obrázky a aj matematické vzorce. Po splnení týchto bodov bude náš softvér schopný konkurovať tým najlepším OCR programom súčasnosti.

Literatúra

[1] Wikipedia, The Free Encyclopedia: *Optical character recognition* http://en.wikipedia.org/wiki/Optical_character_recognition

[2] Rafael C. Gonzales, Richard E. Woods: *Digital Image Processing*, Pearson Prentice Hall, Upper Saddle River, 2008

[3] Lawrence O'Gorman: *The Document Spectrum for Page Layout Analysis*, IEEE Transactions On Pattern Analysis and Machine Intelligence,IEEE Computer Society, Los Alamitos, November 1993

[4] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu: *A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique*, Elsevier Science Inc., New York, 2004

[5] Wikipedia, The Free Encyclopedia: *Kd-tree* http://en.wikipedia.org/wiki/Kd-tree

[6] Wolfram Mathworld: *Least Squares Method* http://mathworld.wolfram.com/LeastSquaresFitting.html

[7] Jiří Šíma, Roman Neruda: *Teoretické otázky neuronových sítí* Matfyzpress, Praha, 1996

[8] Wikipedia, The Free Encyclopedia: *Hamming Distance* http://en.wikipedia.org/wiki/Hamming_distance

[9] Wikipedia, The Free Encyclopedia: *Levenshtein Distance* http://en.wikipedia.org/wiki/Levenshtein_distance

[10] NeuronDotNet Library http://neurondotnet.freehostia.com

[11] FFTW, The Fastest Fourier Transform In The West
http://www.fftw.org/

Dodatok A

Obsah CD

Priložené CD obsahuje:

- *dokumentacia* - adresár s vygenerovanou dokumentáciou pomocou programu Doxygen
- *programy* - adresár obsahujúci archív *programy.zip* s vytvorenými aplikáciami
- *text prace* - tento text vo formáte PS, DVI a PDF
- *ukazkove data* - adresár, ktorý obsahuje naskenované obrázky s textom
- *zdrojove kody* - adresár so všetkými zdrojovými kódmi, vrátane knižnice NeuronDotNet
- súbor *info.txt* - kontakt na autora

Dodatok B

Užívateľská príručka

B.1 Inštalácia balíčka programov

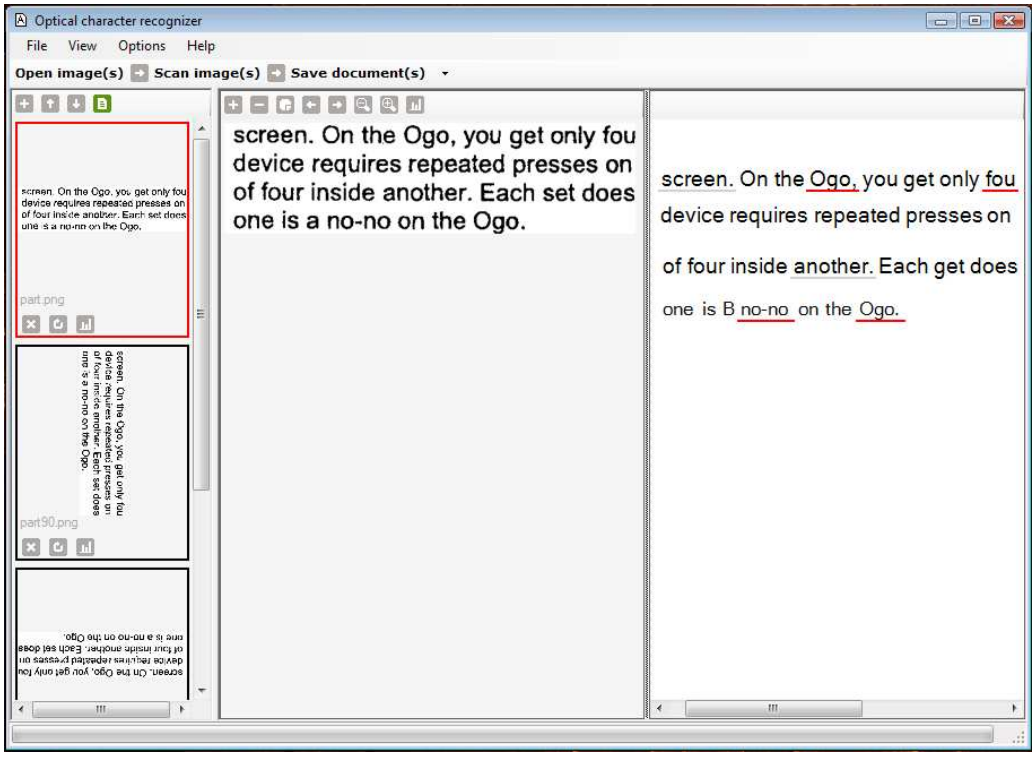
Inštalácia balíčka je veľmi jednoduchá. Stačí rozbaliť archív s programami na ľubovoľné miesto na disku. Programy vyžadujú Microsoft .NET Framework 3.5. Ak tento framework nie je nainštalovaný, je potrebné ho stiahnuť a nainštalovať:

```
http://www.microsoft.com/downloads/details.aspx?
FamilyId=333325FD-AE52-4E35-B531-508D977D32A6\&displaylang=en
```

B.2 Program Optical Character Recognizer

Program Optical Character Recognizer[1] slúži na prevod skenovaných obrázkov s textom do editovateľnej podoby. Štandardne umožňuje uložiť výsledný text do textového alebo HTML súboru. Program spustíme pomocou súboru *OCR.exe*, ktorý sa nachádza v adresári *Optical Character Recognizer*. Na obrázku B.1 je zobrazené hlavné okno súboru s niekoľkými načítanými a rozpoznanými obrázkami. Hlavné okno sa skladá z viacerých častí. V hornej časti sa nachádza hlavné menu. Pod hlavným menu je menu pre konverziu. V ľavej časti okna sa nachádza panel pre prácu s dávkou. Stred okna slúži na zobrazenie aktuálne vyznačeného obrázka a informácií, ktoré budú z neho počas konverzie vyextrahované. V pravej časti sa zobrazuje výsledný text približne v takom rozložení, v akom sa nachádza v obrázku.

[1] skr. OCR

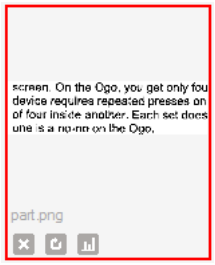


Obr. B.1: Hlavné okno programu Optical Character Recognizer

B.2.1 Práca s obrázkom a dávkou

Pridanie a odstránenie obrázka

Obrázok je možné pridať do dávky kliknutím na položku v menu *File → Add file(s)*. Podporované formáty súboru sú *JPG, PNG, BMP, GIF, EXIG*. Je možné pridať viac obrázkov do dávky naraz označením viacerých obrázkov. Rýchlejší spôsob, ako pridať obrázok do dávky, je kliknúť na obrázok so znamienkom *"+"* v paneli pre prácu s dávkou, alebo kliknúť na tlačidlo *Open image(s)* pod hlavným menu. Ak je obrázok poškodený, alebo je jeho formát nepodporovaný, vypíše sa chybová hláška a tento obrázok bude pri vkladaní do dávky ignorovaný. Obrázok by mal obsahovať tmavý text na svetlom pozadí. Príklad načítaných skenovaných obrázkov je možné vidieť na obrázku B.1. Pre každý načítaný obrázok je vytvorená v paneli pre prácu s dávkou položka s náhľadom obrázku a jeho názvom tak ako je to zobrazené na obrázku B.2. Kliknutím na položku sa zobrazí veľký náhľad obrázku v



Obr. B.2: Položka v paneli pre prácu s dávkou

strednej časti okna. Položka obsahuje aj tri tlačidlá, v poradí zľava: tlačidlo na odstránenie položky z dávky, tlačidlo na zvýšenie kontrastu v obrázku a tlačidlo na zobrazenie histogramu.

Kliknutím na prvé tlačidlo zľava dôjde k okamžitému odstráneniu položky z dávky. Podobne, položku je možné odstrániť aj pomocou *File → Close current file*, položka však musí byť aktívna, to znamená, že v strednej časti hlavného obrázka sa nachádza práve odstraňovaný obrázok. Pomocou *File → Close all files* je možné odstrániť všetky položky z dávky.

Kliknutím pravým tlačidlom myši na položku sa daná položka stane aktívnou a rámček okolo nej bude červený. Kliknutím na šípku hore alebo dole v menu panelu pre prácu s dávkou aktivujeme predchádzajúcu alebo nasledujúcu položku.

Zvýraznenie textu a informácie o obrázku

V prípade obrázku, v ktorom je text málo výrazný, je možné aby sa program pokúsil text zvýrazniť kliknutím na prostredné tlačidlo v danej položke dávky. Tlačidlo je zvýraznené sivým rámčekom, ak už automaticky došlo ku zvýrazneniu textu. Pri automatickom zvýrazňovaní sa však môže stať, že program zle odhadne, či má alebo nemá zvýrazniť text a preto je vždy vhodné skontrolovať viditeľnosť textu označením oblasti (B.2.1) v obrázku a zobrazením jej náhľadu po aplikácii nastavených filtrov (B.2.2).

Kliknutím na tlačidlo najviac vpravo v danej položke dávky sa zobrazí histogram intenzít odtieňov šedej daného obrázka prevedeného do odtieňov šedej. Dáta sú zobrazené do stĺpcov, pričom stĺpec najviac vľavo reprezentuje výskyt čiernej farby a stĺpec najviac vpravo reprezentuje výskyt bielej farby v obrázku.

Obr. B.3: Menu pre prácu s obrázkom

Výber oblastí na spracovanie

Štandardne je počas konverzie spracovávaný celý naskenovaný dokument. To je však nežiadúce, ak sa v dokumente nachádzajú obrázky, prípadne chceme previesť na text len isté časti dokumentu. Software umožňuje vo vstupnom obrázku označiť oblasti, ktoré sa majú alebo nemajú spracovávať. Rozlišujeme tri typy oblastí:

- rozpoznávaná oblasť
- oblasť označujúca obrázok
- nespracovávaná oblasť

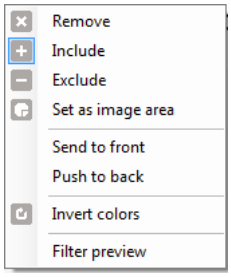
Rozpoznávaná oblasť je tá časť obrázka, v ktorej má dôjsť k rozpoznaniu písmen a štruktúry textu. Oblasť označujúca obrázok slúži na označenie tej časti obrázka, ktorú chceme skopírovať na výstup. Nespracovávaná oblasť je oblasť, ktorá sa bude v ďalších krokoch konverzie obrázka na text ignorovať. Ak je označená nejaká oblasť na rozpoznávanie, tak zvyšná časť obrázku bude považovaná za nespracovávanú oblasť. To akú oblasť chceme označiť

je možné vybrať v menu pre prácu so skenovaným dokumentom zobrazeným na obrázku B.3.

Kliknutím pravého tlačidla myši na nejakú oblasť sa zobrazí menu pre prácu s oblasťou (B.4). V menu je možné zmeniť typ oblasti, odstrániť danú oblasť a presúvať oblasť nad všetky oblasti alebo pod všetky oblasti, pričom dominantný typ oblasti v nejakej časti obrázku je daný typom najvrchnejšej oblasti. Taktiež je možné zobraziť náhľad, ako vyzerá daná oblasť po aplikovaní filtrov (viď B.2.2). Obrázok by mal vyzerať tak, že každé písmeno textu by malo byť samostatné a výrazné. Okrem písmen by sa v obrázku nemali nachádzať žiadne elementy považované za šum, ktorý by mohol veľmi ovplyvniť kvalitu výsledku.

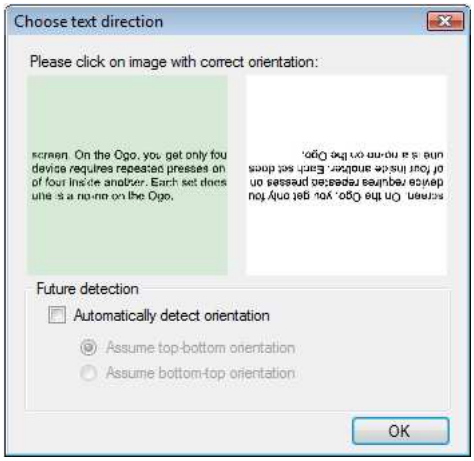
Rozpoznanie

Po tom ako vhodne nastavíme filtre (viď B.2.2) a prípadne označíme oblasti v obrázku, môžeme začať konverziu obrázka na text. V prípade viacerých obrázkov je možné vybrať v paneli pre prácu s dávkou položky, ktoré sa majú konvertovať. Tieto položky vyberieme držaním tlačidla CTRL a klikaním na položky. Vybrané položky budú mať červený rámček. Ak chceme konvertovať len vybrané položky, klikneme na zelené tlačidlo v paneli pre prácu s dávkou. Ak chceme konvertovať všetky obrázky v dávke bez ohľadu na to či sú označené alebo nie, klikneme na tlačidlo *Scan image(s)* pod hlavným menu. Priebeh konverzie je zobrazený v dolnej časti okna. Ak chceme konverziu zastaviť, klikneme na tlačidlo *Abort scanning*. V priebehu konverzie bude, ak je nastavené, užívateľ dotázaný na výber správnej orientácie textu tak, ako je ukázané na obrázku B.5, pretože program si ju automaticky nevie zistiť. Vo zobrazenom formulári môžeme zaškrtnúť voľbu *Automatically detect orientation*, aby program predpokladal jednu z možných orientácií.



Obr. B.4: Menu pre prácu s obrázkom

Pri konverzii ďalších obrázkov už nebudeme dotázaní na správnu orientáciu textu. Toto nastavenie je možné zmeniť v nastaveniach programu, záložka *Text direction*.



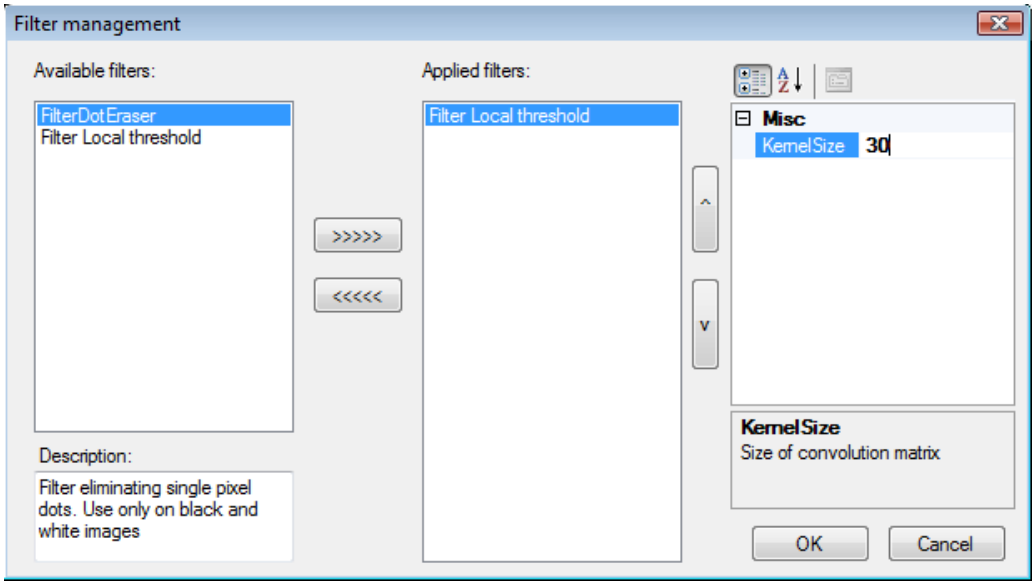
Obr. B.5: Formulár slúžiaci na výber správnej orientácie textu

B.2.2 Filtre

Pred samotnou konverziou sú na obrázok aplikované filtre, ktorých cieľom je odstrániť šum z obrázku. Nastavenia filtrov sú individuálne pre každý obrázok a preto je nutné pred konverziou nastaviť filtre tak, aby ich výsledkom bol čiernobiely obrázok obsahujúci len text. Obrázok nesmie obsahovať žiadne iné elementy inej farby ako farby pozadia, teda bielej farby. Na nastavovanie filtrov slúži manažér filtrov (viď obrázok B.6), ktorý otvoríme kliknutím na *Options* → *Filters*.

Manažér filtrov obsahuje zoznam filtrov, ktoré sú k dispozícii a zoznam filtrov ktoré sú aplikované. Každý použitý filter môže byť samostatne nastavovaný v pravej časti okna, kde sa nachádza zoznam jeho vlastností. Pri každej vlastnosti by mal byť vpravo dole jej popis, na čo slúži.

Ak chceme použiť nejaký dostupný filter, označíme ho v zozname dostupných filtrov a klikneme na tlačidlo obsahujúce šípky doprava. Filter bude umiestnený na koniec zoznamu použitých filtrov. Ak chceme zmeniť pozíciu filtra, ktorá udáva aj poradie aplikácie na obrázok (čím vyššie je filter v zozname, tým skôr bude aplikovaný), označíme filter v zozname použitých filtrov a klikneme na tlačídlo indikujúce smer nahor, prípadne nadol.



Obr. B.6: Manažér filtrov

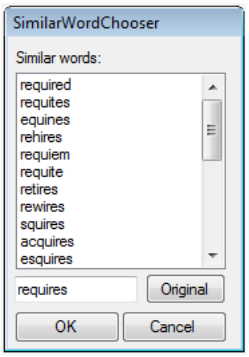
Štandardná distribúcia softwaru poskytuje dva filtre. FilterLocalThreshold slúži na hlavné odstránenie šumu z obrázka. Filter odstraňuje oblasti s jemnými prechodmi a ponecháva časti s veľkou zmenou intenzity farby. Pri tomto filtri je možné nastaviť hodnotu KernelSize. Táto hodnota by sa mala pohybovať v rozmedzí od 10 do 100, v závislosti na rozlíšení obrázka, pri ktorom bol skenovaný. Pre obrázky skenované pri rozlíšení 300dpi odporúčame hodnotu 50. Platí, že čím vyššie je rozlíšenie, tým vyššia by mala byť táto hodnota.

FilterDotEraser slúži na odstraňovanie čiernych bodiek v obrázku a na vypĺňanie bielych bodiek čiernou farbou v oblastiach s čiernymi pixelmi. Obsahuje nastaviteľnú hodnotu Threshold, ktorá indikuje, koľko bielych (čiernych) pixelov v najbližšom susedstve čierneho (bieleho) pixelu, je potrebných na to, aby bol daný pixel nastavený na opačnú farbu, teda aby zmizla čierna alebo biela bodka. Nie je vhodné nastavovať túto hodnotu na menej ako 6.

Ak máme k dispozícii iný filter ako vyššie spomenuté, stačí dll súbor s filtrom vložiť do adresára Filters, ktorý sa nachádza v adresári so spúšťacím súborom a reštartovať aplikáciu. Ak tento filter skutočne spĺňa podmienky na to aby mohol byť použitý ako filter, zobrazí sa v manažéri filtrov v zozname dostupných filtrov.

B.2.3 Kontrola správnosti rozpoznania

Text, ktorý je výsledkom konverzie je, ak je tak nastavené, následne skontrolovaný voči aktuálnemu slovníku. Ak sa rozpoznané slovo nenachádza v slovníku a nebolo nájdené jednoznačne určené najpodobnejšie slovo, je označené ako chybné a na výstupe je podčiarknuté červenou farbou. Ak sa slovo nenchádza v slovníku, ale bolo nájdené jednoznačné najpodobnejšie slovo, je dané slovo nahradené tým zo slovníka a na výstupe bude podčiarknuté sivou farbou. Slová je možné na výstupe zmeniť. Stačí, ak klikneme na rozpoznané slovo a zobrazí sa formulár so zoznamom nájdených najpodobnejších slov. Ak chceme slovo zameniť za niektoré z tohto zoznamu, klikneme na vybrané slovo a potom na OK, alebo vykonáme dvojklik myšou na vybrané slovo zo zoznamu. Ak chceme slovo zameniť za také, ktoré sa nenachádza v zozname, je možné ho zameniť za vlastné, stačí ho napísať do texového poľa v spodnej časti okna. V spodnej časti formulára sa tiež nachádza tlačidlo, ktorým sa vrátime k pôvodnému textu. Na obrázku B.7 je zobrazený tento formulár. Použitý slovník môžeme zmeniť v nastaveniach programu (záložka *Spell checker*), tak isto aj spôsob vyhľadávania podobných slov a počet slov, ktoré majú byť vyhľadané. Štandardne je k dispozícii anglický slovník, ktorý obsahuje približne 85000 slov.



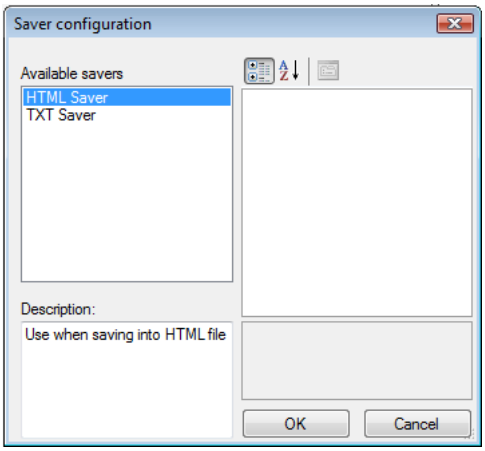
Obr. B.7: Formulár na výber podobných slov

B.2.4 Uloženie výsledku

Výsledok konverzie je možné uložiť štandardne do textového alebo HTML súboru. Stačí kliknúť na tlačidlo *Save document(s)* pod hlavným menu. Kliknutím naň sa vysunie menu, kde si zvolíme formát súboru z dostupných

formátov. Ak bolo skonvertovaných viac dokumentov, sú tieto dokumenty ukladané do jedného súboru za sebou.

Program podporuje ukladanie do ľubovoľného formátu, stačí uložiť knižnicu na to určenú do adresára *Savers*, ktorý sa nachádza na úrovni spúšťacieho súboru. Každý "ukladač" môže mať vlastné nastavenia. Tieto nastavenia je možné meniť v manažéri ukladačov (obrázok B.8), ktorý zobrazíme kliknutím na *Options* → *Savers*.



Obr. B.8: Manažér ukladačov

B.2.5 Nastavenia programu

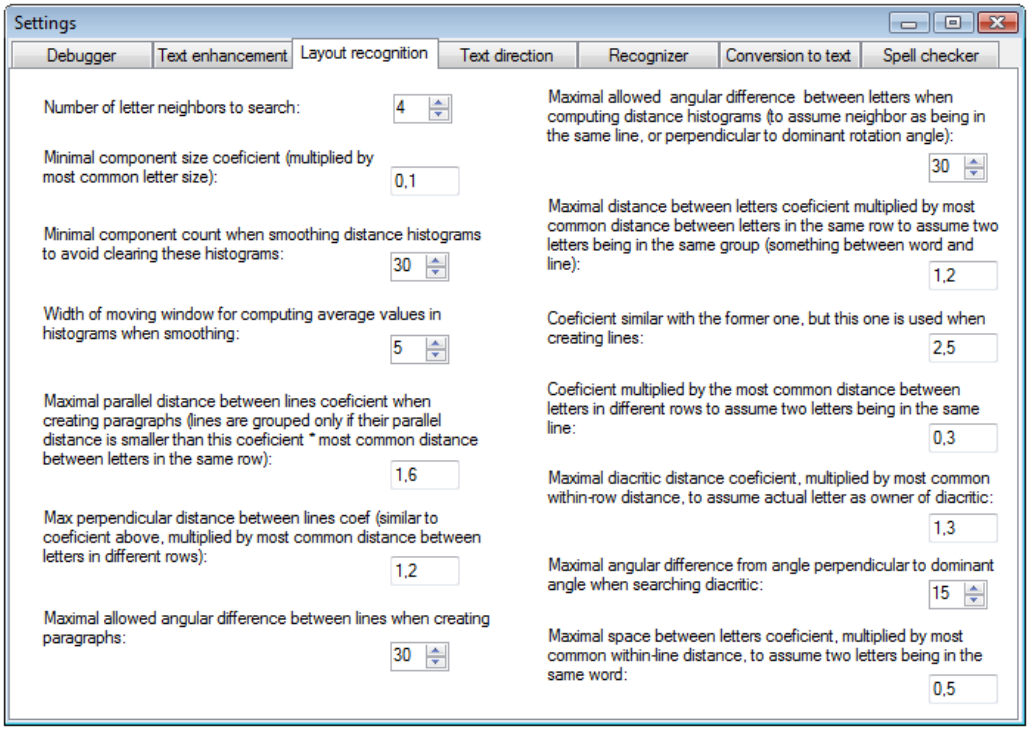
Program OCR obsahuje množstvo nastavení. Formulár s nastaveniami (viď obrázok B.9) zobrazíme kliknutím na *Options* → *Settings*.

Zvýrazňovanie textu

V záložke *Text enhancement* je možné nastaviť, či sa má pri načítaní obrázku automaticky vykonávať zvýraznenie textu. V prípade, že sa nemá, je možné nastaviť či sa má prevádzať vždy, alebo nikdy.

Smer textu

Počas konverzie program nevie, či sa obrázok nachádza "hlavou dole" alebo nie. Štandardne bude užívateľ dotázaný na výber správnej orientácie. Nastaveniami v záložke *Text direction* môžeme nechať program, nech automaticky



Obr. B.9: Nastavenia programu

predpokladá zadanú orientáciu. Počas konverzie už užívateľ nebude dotazovaný.

Rozpoznávače textu

V záložke *Recognizer* môžeme nastaviť cestu k hlavnej neurónovej sieti, ktorá bude rozpoznávať písmená a taktiež adresár, v ktorom sa nachádzajú pomocné neurónové siete slúžiace na vylepšenie úspešnosti rozpoznania.

Kontrola textu

V záložke *Spell checker* môžeme zmeniť používaný slovník, zmeniť metódu na vyhľadávanie najpodobnejších slov a počet vyhľadávaných slov. Taktiež môžeme automatickú kontrolu úplne vypnúť.

Pri výbere Hammingovej vzdialenosti budú vyhľadávané len slová rovnakej dĺžky ako zadané slovo. Táto metóda je rýchla, ale jej úspešnosť vyhľadania je menšia ako v prípade Levenshteinovej metódy, ktorá vyhľadáva najpodobnejšie slová, ktorých dĺžka sa môže líšiť od dĺžky zadaného slova o 1. Vyhľadávanie pomocou Levenshteinovej vzdialenosti je preto pomalšie ako pomocou Hammingovej.

Zobrazenie informácií o obrázku

Počas konverzie program zistí o štruktúre dokumentu veľa informácií. V záložke *Debugger* si môžeme vybrať, ktoré informácie chceme zobraziť. Môžeme zobraziť označenie nájdených písmen, riadkov, blokov a ich poradové čísla po zoradení. Taktiež je možné zobraziť uhol otočenia dokumentu, histogram uhlov medzi vyhľadanými susedmi, histogram vzdialeností písmen v jednom riadku a medzi riadkami. Taktiež si môžeme nechať zobraziť zistenú najčastejšiu vzdialenosť medzi písmenami v jednom riadku a medzi riadkami. Znalosť týchto hodnôt môžeme použiť pri nastavovaní vlastností rozpoznania textu v prípade, že počas prvej konverzie sme nedosiahli úspešné výsledky pri zisťovaní štruktúry textu.

Rozpoznanie členenia dokumentu

Rozpoznanie členenia dokumentu je možné ovplyvniť množstvom koeficientov. Tieto koeficienty nastavíme v záložke *Layout recgnition*. Táto záložka obsahuje nastavenia postupne po stĺpcoch zľava doprava:

- počet susedov, ktoré majú byť vyhľadané pre každé písmeno. Počet susedov by sa mal pohybovať v rozpätí 4-5 pri bežnom dokumente a 6 až 7 pri dokumente, ktorý má veľké riadkovanie.

- koeficient určujúci minimálnu veľkosť nájdeného písmena(alebo zhluku čiernych pixelov, šum). Pri dokonalom dokumente, ktorý obsahuje len písmená môže byť táto hodnota nastavená na ľubovoľne nízku kladnú hodnotu, pri viac zašumených dokumentoch, ak sú písmená dostatočne veľké, môže byť hodnota nastavená až na 0,15. Tento koeficient je násobený najbežnejšou veľkosťou písmena v dokumente, kde veľkosť písmena je počítaná ako veľkosť uhlopriečky najmenšieho obdĺžnika, ktorý obklopuje písmeno.

- minimálny počet komponentov potrebných na vykonanie vyhladenia histogramov vzdialeností medzi písmenami. Tento počet by mal byť minimálne 20°.

- šírka vyhladzovacieho okna použitá pri vyhladzovaní histogramov. Širka určuje počet po sebe idúcich hodnôt v histograme, ktoré budú spriemerované. Čím väčšia šírka, tým viac bude histogram vyhladený. Optimálne hodnoty šírky okna sú od 3- 10, v závislosti od tvaru daného histogramu.

- koeficient maximálnej rovnobežnej vzdialenosti medzi priamkami vzhľadom k najbežnejšej vzdialenosti medzi písmenami v riadku, ak chceme aby boli tieto priamky priradené do jedného bloku. Optimálna hodnota sa pohybuje v rozmedzí 1,5 až 2,5 v závislosti od hustoty textu.

- koeficient maximálnej kolmej vzdialenosti medzi priamkami vzhľadom k najbežnejšej vzdialenosti medzi písmenami v rôznych riadkoch, ak chceme aby boli tieto priamky priradené do jedného bloku. Optimálna hodnota je v rozmedzí 1,2 až 5, podľa toho aké je veľké riadkovanie v dokumente.

- maximálna hodnota rozdielu uhlov pootočenia dvoch priamok, aby boli tieto priamky priradené do jedného bloku. Hodnota tohto koeficientu by mala byť z rozmedzia $10° - 30°$

- maximálny odchýlka uhlu priamky prechádzajúcej cez susediace písmená, aby boli v procese vytvárania histogramov vzdialeností označené

ako patriace do jedného riadku (odchýlka od dominantného uhlu), prípadne označené ako patriace do rôznych riadkov (odchýlka od uhlu priamky kolmej na priamku nakreslenú pod dominantným uhlom). Čím hustejší text, tým menšia môže byť hodnota. Optimálna hodnota sa pohybuje v rozmedzí $5° - 30°$.

- koeficient vzdialenosti medzi písmenami vzhľadom k najbežnejšej vzdialenosti medzi písmenami v jednom riadku, aby boli tieto písmená zlúčené do jednej skupiny písmen (podľa veľkosti koeficientu môžu byť vytvorené skupiny práve slová textu, prípadne riadky textu v rámci jedného bloku). Pre vytvorenie skupín veľkosti slov je vhodná hodnota z rozmedzia 1,2 - 1,4. Pre vytvorenie skupín veľkosti riadkov je vhodná hodnota z rozmedzia 2 - 3.

- koeficient podobný predchádzajúcemu. Slúži na vytváranie riadkov textu a jeho hodnota by mala byť z rozmedzia 2 - 3 v závislosti od veľkosti medzier medzi slovami a vetami v texte.

- koeficient maximálnej kolmej vzdialenosti dvoch písmen vzhľadom k najbežnejšej vzdialenosti medzi riadkami, aby boli tieto písmená považované za písmená z jedného riadku.

- koeficient maximálnej vzdialenosti diakritiky od nadradeného písmena vzhľadom k najbežnejšej vzdialenosti dvoch písmen v rámci jedného riadku.

- maximálna odchýlka uhlu medzi diakritikou a písmenom od uhlu priamky kolmej na priamku nakreslenú pod dominantným uhlom, aby mohla byť táto diakritika priradená danému písmenu. Optimálna hodnota sa pohybuje v rozmedzí od $10° - 20°$.

- maximálna veľkosť medzery medzi písmenami v jednom slove vzhľadom k najbežnejšej vzdialenosti medzi písmenami vrámci riadku. Optimálna hodnota závisí od veľkosti fontu a veľkosti medzier medzi slovami a mala by sa pohybovať v rozmedzí 0,2 - 0,8. Čím väčší font a menšie medzery, tým menšia by mala byť hodnota.

Nastavenie vlastností rozpoznania textu

Po rozpoznaní písmen môžeme upraviť niektoré ťažko rozoznateľné písmená dodatočne, vzhľadom k slovám v ktorých sa nachádzajú. Tieto nastavenia

môžeme zmeniť kliknutím na záložku *Conversion to text*. Môžeme nastaviť:

- minimálny podiel veľkých písmen v slove, aby boli výškou nestabilné písmená(vyzerajú rovnako, keď sú veľké aj malé, napr. s,v,x,o,c,w,p,z) považované za veľké písmená.

- maximálny podiel nealfanumerických znakov a znakov "o" a "O" v slove, aby bola nula považovaná za "O" alebo "o".

- minimálny podiel čísel v slove aby boli znaky "o" a "O" považované za nulu.

- minimálny podiel čísel v slove aby bol znak "l" považovaný za jednotku.

- maximálny podiel čísel v slove aby bola jednotka považovaná za "l".

- koeficient, ktorý určuje maximálnu veľkosť znakov ".,_ ". Hodnota závisí od veľkosti spomenutých znakov. V prípade "veľkých čiarok a malých písmen" je vhodné nastaviť túto hodnotu na 0,5. V opačnom prípade je optimálna hodnota v rozmedzí od 0,2 do 0,4 v závislosti od veľkosti fontu (čím väčší font, tým menšia hodnota)

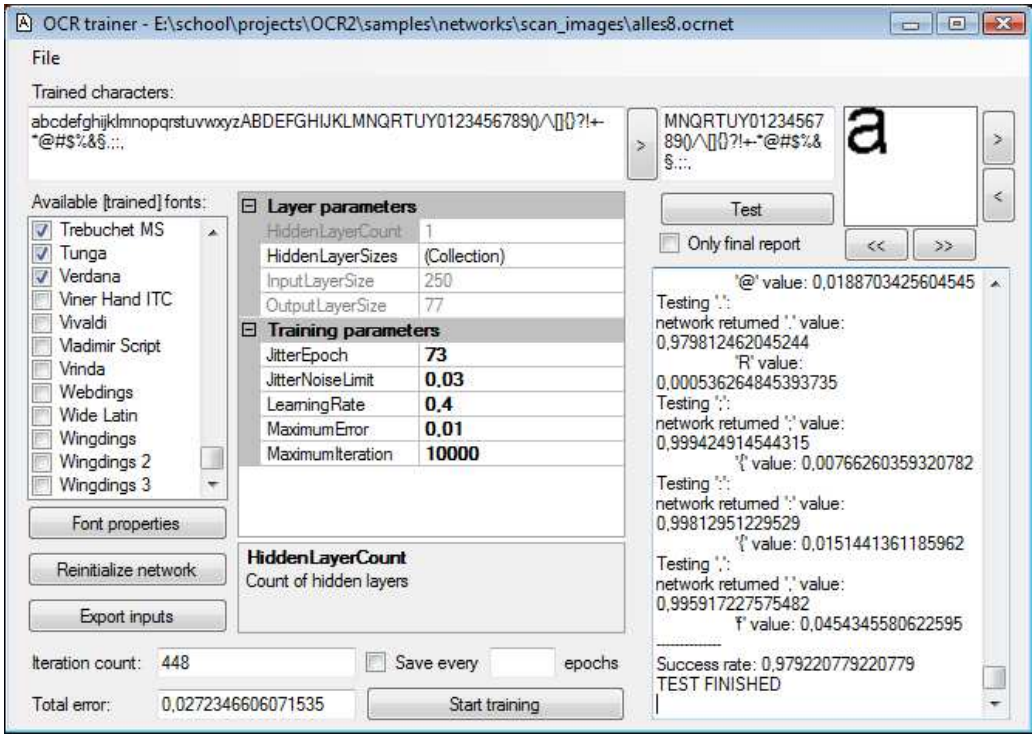
B.3 Program OCR trainer

Program OCR trainer (na obrázku B.10) slúži na vytvorenie, trénovanie a testovanie neurónovej siete slúžiacej na rozpoznanie písmen v obrázku. Hlavné okno sa skladá z dvoch hlavných častí - tréningovej a testovacej.

Poznámka: trénovanie neurónovej siete je zložitý proces. V nasledujúcich podkapitolách budeme predpokladať užívateľa s aspoň základnou znalosťou neurónových sietí.

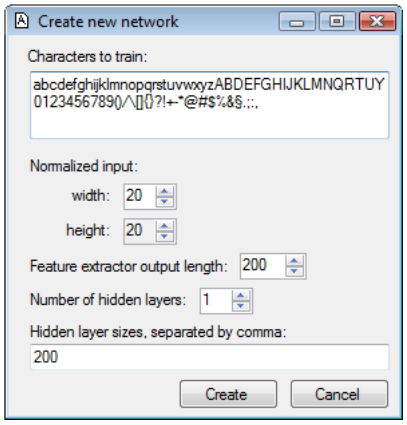
B.3.1 Vytvorenie novej siete

Novú sieť vytvoríme kliknutím na *File → New network*. Zobrazí sa dialógové okno (obrázok B.11), kde nastavíme množinu znakov, ktoré chceme trénovať a veľkosť generovaného obrázka, do ktorého majú byť tieto znaky postupne vykreslené. Ďalej nastavíme počet koeficientov, ktoré majú byť z obrázka extrahované. Tieto koeficienty charakterizujú daný obrázok. Čím viac koeficientov zvolíme (až do počtu pixelov vo vygenerovanom obrázku). Počet



Obr. B.10: Program OCR trainer

koeficientov by nemal byť príliš malý, aby nedošlo k zamieňaniu písmen pri trénovaní z dôvodu nedostatku informácií. Optimálny počet je polovica počtu pixelov v generovanom obrázku. Ďalej nastavíme počet skrytých vrstiev v sieti. Pri trénovaní písmen stačí jedna alebo dve skryté vrstvy. Čím viac skrytých vrstiev obsahuje sieť, tým dlhšie bude trvať trénovanie. Pre každú vrstvu nastavíme jej veľkosť (počet neurónov) tak, že tieto veľkosti napíšeme do textového poľa v dolnej časti dialógového okna. Veľkosti oddelíme čiarkou. V prípade nezhody zadaného počtu vrstiev a počtu veľkostí vrstiev bude vypísaná chybová hláška.



Obr. B.11: Dialógové okno na vytvorenie novej siete

B.3.2 Nastavenie tréningovej množiny

Tréningová množina je zoznam tréningových vzorov, ktoré budú predkladané sieti pri jej učení. Tréningový vzor je vytvorený z koeficientov vyextraktovaných z vygenerovaného obrázka s písmenom. To, akými fontami chceme vykresľovať písmeno, si môžeme vybrať zo zoznamu dostupných fontov. Čím viac fontov zaškrtneme, tým lepšie môžeme natrénovať sieť, fonty však musia byť podobné. S počtom vybraných fontov však stúpa časová náročnosť trénovania. Každý font má svoje vlastnosti. Po označení fontu a kliknutí na tlačidlo *Font properties* môžeme zmeniť hodnotu *Kernel size*. Táto hodnota je používaná pri prevádzaní vygenerovaného obrázku v odtieňoch šedej na čiernobiely obrázok, a pri fontoch kde písmená pozostávajú z "tenkých čiar" by mala byť táto hodnota väčšia. Optimálna hodnota je 20.

B.3.3 Nastavenie parametrov trénovania

Pred začatím tréningu siete musíme nastaviť parametre, ktoré charakterizujú jeho priebeh. Cieľom trénovania je minimalizovať celkovú chybu siete. Táto chyba je vypisovaná v textovom poli označenom ako *Total error*.

Pri tréningu sú sieti postupne predkladané tréningové vzory. Predloženie všetkých vzorov nazývame *epocha*. Na to aby sme úspešne natrénovali sieť potrebujeme vykonať rádovo tisíce epôch. Aktuálny počet vykonaných epôch je zobrazený v textovom poli označenom ako *Iteration count*.

Kedykoľvek pred alebo aj počas trénovania siete môžeme zmeniť hodnotu všetkých parametrov. *LearningRate* udáva rýchlosť učenia. Hodnota by mala byť z intervalu [0,1]. Vo všeobecnosti platí, že čím menšiu hodnotu LearningRate nastavíme, tým pomalšie sa bude sieť učiť. Ak však zvolíme príliš vysokú hodnotu, môže sa stať, že sieť sa nedokáže naučiť rozpoznávať zadané písmená. Optimálna hodnota LearningRate je 0,4.

Niekedy je vhodné (aby sa sieť neučila príliš rýchlo, prípadne ak sa sieť prestane učiť a pritom je veľká celková chyba siete), aby boli váhy spojení medzi neurónmi počas trénovania málo pozmenené. Parameter *JitterEpoch* udáva veľkosť periódy v epochách, kedy má dôjsť k zmene váh. Ak je hodnota záporná alebo nula, nebude dochádzať k dodatočnej zmene váh.

Parameter *JitterNoiseLimit*, ktorého hodnota H udáva interval [-H,H]. Náhodná hodnota z tohto intervalu bude každú *JitterEpoch* pripočítaná k váham spojení medzi neurónmi. Hodnota parametru by sa mala nachádzať v blízkosti nuly. V opačnom prípade sa môže stať, že sa sieť prestane učiť.

Parameter *MaximumError* určuje maximálnu dovolenú celkovú chybu siete. Tréning je zastavený, ak aktuálna chyba siete klesne pod túto hranicu. Hodnota tohto parametra by mala byť malá, ale nie príliš malá, aby nedošlo k natrénovaniu siete len na tréningovú množinu. Dosiahnutie stanovenej hranice závisí od tréningovej množiny. Optimálna hranica sa pohybuje v rozpätí 0,00001 až 0,01.

Parameter *MaximumIteration* určuje maximálny počet epôch. Trénovanie sa zastaví ak je dosiahnutá táto hodnota počtu epôch a zároveň celková chyba siete neklesla pod stanovenú hranicu.

Tréning začneme kliknutím na tlačidlo *Start training* a môžeme ho kedykoľvek zastaviť kliknitím na to isté tlačidlo.

B.3.4 Automatická záloha

Trénovanie neurónových sietí je spravidla veľmi časovo náročné. Aby sme nemuseli venovať pozornosť trénovaniu, prípadne ukladať doterajšie výsledky, program OCR trainer umožňuje automatické vytváranie zálohy. Stačí zadať do textového poľa nad tlačidlom *Start training* hodnotu, po koľkých epochách sa má vytvoriť záloha aktuálneho stavu siete. Stav siete je uložený do nového súboru, ktorého názov je vytvorený z názvu súboru so sieťou a poradového čísla zálohy. Záloha je ukladaná do adresára kde je pôvodná sieť uložená. Ak vytvárame zálohu neuloženej siete, tak záloha je vytvorená do adresára so spúšťacím súborom programu OCR trainer.

B.3.5 Testovanie siete

Pravá časť hlavného okna slúži na testovanie natrénovanej siete. Do textového poľa v hornej časti obrazovky zadáme písmená, ktoré chceme otestovať. Z týchto písmen bude vygenerovaný obrázok tak ako bol generovaný pri učení. Výsledný generovaný obrázok je zobrazený v pravej hornej časti hlavného okna. Medzi vygenerovanými obrázkami s rôznymi písmenami a rôznymi fontami sa môžeme pohybovať pomocou tlačidiel, ktoré sa nachádzajú vedľa a pod obrázkom. Pred spustením testu si môžeme pomocou voľby *Only final report* zvoliť, či chceme zobraziť len celkovú úspešnosť siete, alebo či chceme zobrazovať počas testovania podrobné informácie o predkladaných vzoroch a výstupoch zo siete. V druhom prípade sa vypíše písmeno, ktoré je predkladané sieti a k nemu prvé dva najpravdepodobnejšie rozpoznané písmená spolu s "mierou istoty" (hodnota od 0 do 1, kde 1 je úplná istota, t.j. sieť je na 100% presvedčená o správnosti výsledku).

B.3.6 Ukladanie a export

Natrénovanú sieť je možné uložiť a kedykoľvek znova otvoriť a pracovať s ňou ďalej. Sieť môžeme uložiť pomocou *File* → *Save network*. Zavretím hlavného okna budeme taktiež dotázaní na uloženie siete.

Program poskytuje možnosť exportovania tréningovej množiny kliknutím na *Export inputs*. Pri exportovaní sú vytvorené tri súbory. Prvý súbor obsahuje koeficienty tréningových vzorov. Druhý súbor obsahuje maticu jednotiek a núl. Šírka matice je počet písmen. Výška matice je počet tréningových vzorov. Jednotka sa nachádza v matici na tom mieste, kde pre daný vzor požadujeme dané písmeno (číslo písmena závisí od poradia v reťazci

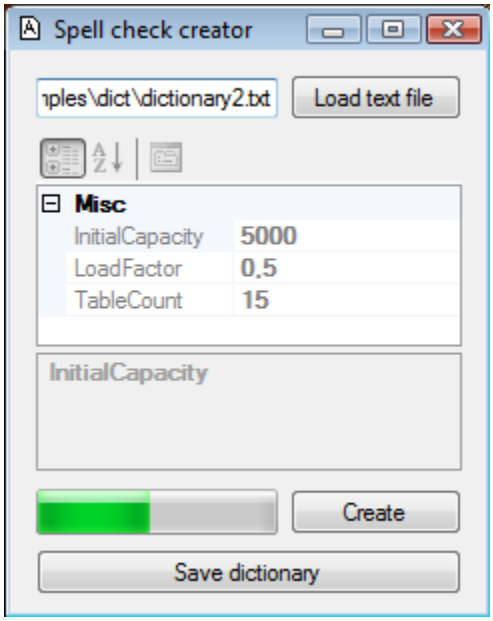
trénovaných znakov). Tretí súbor obsahuje rovnaké informácie ako druhý súbor, ale v inej reprezenácií. Konkrétne, obsahuje čísla, ktoré reprezentujú požadované výstupy (písmená) tréningových vzorov. Táto reprezentácia je po malých úpravách vhodná napríklad pre program Matlab a trénovanie pomocou algoritmu LVQ.

B.4 Program Spell check Creator

Program Spell check Creator (na obrázku B.12) slúži na vytvorenie slovníka používaného v programe OCR pri kontrole textu. Slovník vytvoríme zo zoznamu slov oddelených aspoň jednou medzerou alebo novým riadkom. Súbor so zoznamom slov načítame kliknutím na tlačidlo *Load text file*.

Po načítaní zoznamu slov je možné nastaviť vnútornú reprezentáciu slovníka. Slová v slovníku sú rozdelené podľa ich dĺžky do viacerých skupín. Počet skupín je možné nastaviť zmenou hodnoty *TableCount*. V prvej skupine sa budú nachádzať slová dĺžky 1, v druhej slová dĺžky 2 atď. V poslednej (n-tej) skupine sa nachádzajú slová dĺžky n a viac. Hodnota konštanty *TableCount* by preto nemala byť malá (menšia ako 7). Na druhú stranu, je zbytočné vytvárať samostatné skupiny pre extra dlhé slová, ktorých je len veľmi málo. Preto by hodnota *TableCount* nemala presiahnuť 15. Konštanta *LoadFactor* udáva pomer medzi rýchlosťou vyhľadávania a spotrebou pamäte. Čím bližšie je hodnota k nule, tým rýchlejšie budú slová vyhľadané a tým viac sa spotrebuje pamäte. Optimálna hodnota sa pohybuje od 0,2 do 0,7 v závislosti od veľkosti slovníka. Čím väčší slovník, tým menšiu hodnotu by mala konštanta nadobúdať, aby bolo vyhľadávanie rýchlejšie. Hodnota *InitialCapacity* by mala byť približne *počet slov v slovníku / TableCount* a udáva očakávanú priemernú veľkosť skupiny slov.

Po nastavení vlastností slovníka tento slovník vytvoríme kliknutím na *Create*. Tento proces môže trvať v závislosti od veľkosti slovníka aj niekoľko minút. Počas vytvárania slovníka nie je možné meniť jeho vlastnosti. Po dokončení procesu je možné slovník uložiť. Uložený súbor potom môžeme načítať v programe OCR v *Nastaveniach*, záložka *Spell checker* (viď B.2.5). zxczxc



Obr. B.12: Program SpellChecker Creator

Dodatok C

Detaily implementácie

Balíček programov je napísaný v programovacom jazyku C#. Aby správne fungoval, musí byť na počítači nainštalovaný Microsoft .NET Framework 3.5. Na preloženie zdrojových súborov je potrebné mať nainštalované Microsoft Visual Studio 2008. Hlavný .sln súbor sa nachádza v adresári *zdrojove kody/OCR*.

Pri práci sme použili tieto knižnice:

- *NeuronDotNet*[10], ktorá slúži na vytvorenie, trénovanie a používanie neurónovej siete. Túto knižnicu sme museli upraviť, aby spĺňala naše podmienky na použitie. Zdrojové kódy je možné nájsť na priloženom CD v adresári *zdrojove kody/NeuronDotNet*. Táto knižnica je napísaná v jazyku C#.

- *FFTW*[11], ktorá okrem iného implementuje diskrétnu kosínovú transformáciu, ktorú sme použili na vytvorenie tréningových vzorov pri trénovaní neurónovej siete. Táto knižnica je napísaná v jazyku C. My sme vytvorili C++ knižnicu, ktorá poskytuje základnú funkčnosť knižnice FFTW. Tá je potom použitá v knižnici *ImageFeatureExtractor* napísanej v C#, ktorá implementuje extrahovanie informácií z obrázka. V našom prípade vracia koeficienty transformácie získané z obrázku.

C.1 Reprezentácia dát

Vytvorený balíček programov obsahuje množstvo viac alebo menej dôležitých štruktúr a nebudeme popisovať všetky. Popíšeme len niekoľko najzaujímavejších a zároveň najdôležitejších v celom programe.

C.1.1 Vstupný obrázok

Každý vstupný obrázok je uchovávaný v objekte typu *ScannedImage*. Samotný obrázok je reprezentovaný ako matica hodnôt typu *short*. Okrem tejto matice obsahuje trieda ScannedImage aj vlastný manažér označovaných oblastí - objekt typu *RegionManager*. S dávkou sa pracuje pomocou singleton objektu typu *BatchManager*.

C.1.2 Reprezentácia štruktúry dokumentu pri spracovávaní obrázku

Pri spracovávaní obrázku vytvárame zoznam označených oblastí typu *LabeledComponent*. Tieto oblasti sú združené do riadkov typu *Line*. Riadky sú združené do blokov textu typu *Paragraph*. Triedy *Line* a *Parahraph* sú odvodené od triedy *DocumentItem*, ktorá uchováva informácie o najmenšom nepootočenom obdĺžniku, v ktorom sa daný objekt na obrázku nachádza. V priebehu konverzie sú zhromažďované potrebné dáta o bloku textu v objekte typu *ParagraphInfo*. Podľa informácií, ktoré tento objekt zhromažďuje je vytvorená štruktúra dokumentu. Predpokladá sa, že je stĺpcová, a preto sme využili návrhový vzor *Composite* tak ako sme to popísali v 3.9.1. Každý element tejto štruktúry je odvodený od *BaseColumn*, ktorý obsahuje informácie o umiestení bodov reprezentujúcich skutočne najmenší obdĺžnik, ktorý obsahuje daný blok, otočený o dominantný uhol.

C.1.3 Reprezentácia štruktúry dokumentu pri ukladaní

Nedostatok predchádzajúcej štruktúry je ten, že zachytáva iba polohy blokov textu. Tieto polohy však nemôžeme použiť po tom ako dôjde k rozpoznaniu textu a vytvoreniu slov a riadkov textu. Preto je táto štruktúra skonvertovaná do štruktúry, ktorá sa používa na zobrazenie a uloženie rozpoznaného textu. Táto štruktúra (*FinalDocumentStructure*) je zložená zo stĺpcov textu (*FinalColumns*) a zoznamu obrázkov (*ImageInfo*) vystrihnutých z pôvodného obrázku (ak užívateľ označil nejakú oblasť ako obrázok). Trieda *FinalColumns* obsahuje zoznam stĺpcov textu (*FinalColumn*). Každý stĺpec textu obsahuje zoznam položiek typu *FinalBaseElement*. Všetky elementy štruktúry okrem obrázkov sú odvodené práve od tejto triedy. Blok textu v predchádzajúcej štruktúre je skonvertovaný na elementy typu *FinalParag-*

raph. Ten obsahuje zoznam elementov typu *FinalText*, ktoré reprezentujú riadky textu. Nakoniec, každý element typu *FinalText* obsahuje zoznam slov typu *FinalWord*. Každý element typu slovo obsahuje dva reťazce. Prvý reprezentuje aktuálne použité slovo a druhý reprezentuje slovo, ktoré vzniklo po rozpoznaní písmen.

Nová štruktúra akceptuje visitor typu *DocumentStructureVisitor* a je stále pootočená tak ako bola pootočená tá pôvodná. Na jej pootočenie do uhlu 0° slúži *RotateStructureVisitor*. Na vykreslenie tejto štruktúry je použitý *PaintStructureVisitor*.

C.1.4 Rozpoznávanie textu

Konverzia textu prebieha po blokoch. Každý blok typu *ParagraphInfo* obsahuje zoznam riadkov textu (text myslíme zoznam písmen zatiaľ reprezentovaných ako *LabeledComponent*). Tie sú pri konverzii pomocou metódy *Convert* v triede *ParagraphConverter* skonvertované na skutočný text. Konverzia písmena prebieha v metóde *Recognize* v triede *CharacterRecognizer*. Skutočné slová sú uchovávané v objektoch typu *Word*.

C.1.5 Reprezentácia slovníku

Slovník, voči ktorému sa kontroluje správnosť rozpoznaných slov, je zložený zo zoznamu K hashovacích tabuliek v C# reprezentovaných prostredníctvom vstavanej triedy Hashtable. V každej hashovacej tabuľke sú zahashované indexy slov rovnakej dĺžky s výnimkou poslednej tabuľky, kde sú zahashované indexy slov dĺžky K a viac. Zahashované indexy sú indexami do patričného poľa slov (v ktorom sa nachádzajú opäť slová danej dĺžky). Pri vyhľadávaní podobných slov, kde vzdialenosť dvoch slov je počítaná pomocou Hammingovej vzdialenosti, sa prehľadáva práve jedno pole, v ktorom sú uložené slová tej istej dĺžky. Ak používame Levenshteinovu vzdialenosť, je najprv prehľadané pole so slovami rovnakej dĺžky L a v prípade potreby aj polia so slovami dĺžky $L-1$ a $L+1$. Tento spôsob určite nie je najefektívnejší, ale je pomerne jednoduchý na implementáciu.

C.2 Vlákna

Vykonávané výpočty sú časovo náročné a preto ich musíme vykonávať v inom vlákne než je to, v ktorom sa spracováva vstup od užívateľa. Na pre-

sunutie výpočtov sme použili vstavanú triedu *BackgroundWorker*. Pomocou jej funkcionality je možné výpočty zastaviť a taktiež je možné zobrazovať aktuálny stav spracovania v progressbare v dolnej časti hlavného okna.

Rýchlosť programov sme sa snažili zvýšiť použitím *unsafe* kódu a ukazovateľov. V prípade filtrov tak došlo k značnému posunu v rýchlosti spracovania.

C.3 Rozšíriteľnosť programu

Program OCR je vytvorený tak, aby bolo možné pridávať ďalšie filtre na predspracovanie obrázku a aby bolo možné ukladať výsledok do ďalších formátov. To sme dosiahli využitím reflexie. Každý filter alebo "ukladač" je plugin, ktorý má svoje meno a popis a musí implementovať rozhranie IPlugin:

```
public interface IPlugin
{
    string GetName();
    string GetDescription();
}
```

Každý filter musí implementovať rozhranie IFilter, aby bolo možné daný filter použiť a uložiť jeho nastavenia:

```
public interface IFilter : IPlugin,ISerializable
{
    void Apply(ref short[,] image, int width, int height);
}
```

Podobne, každý ukladač musí implementovať rozhranie ISaver, aby mohol byť v programe použitý:

```
public interface ISaver: IPlugin,ISerializable
{
    void Save(Stream s, FinalDocumentStructure document);
    void Save(StreamWriter w, FinalDocumentStructure document);

    void PreSave(StreamWriter w, string saveDir, string fileName);
    void PostSave(StreamWriter w);

    string GetButtonText();
    string GetExtension();
}
```

Metóda PreSave je zavolaná pred začatím ukladania výsledných textov. Ako parameter dostane adresár, kde sa má uložiť výsledok a názov súboru. Po skončení ukladania je zavolaná metóda PostSave. Metódy GetButtonText a

GetExtension sú používané pri výpise dostupných ukladačov. Pri ukladaní sa spracováva inštancia objektu typu FinalDocumentStrucure. Tú rozoberieme v kapitole C.1.

Vlastnosti pluginov sú zobrazené pomocou komponentu *PropertyGrid*. Ten zobrazuje vlastnosti objektu v jazyku C#.

C.4 Nastavenia programu

Program Optical Character Recognizer obsahuje množstvo nastavení. Tie sú sústredené v triede *Settings*. Všetky nastavenia sú pomocou serializácie uložené do súboru *settings*, ktorý sa nachádza v adresári so spúšťacím súborom. Nastavenia filtrov a ukladačov sú ukladané do osobitných súborov.