Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Aplikace pro rozpoznávání ručně psaného textu

2016

Tomáš Urbanec

Vedoucí práce: RNDr. Miroslav Kolařík, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Tomáš Urbanec
Název práce:	Aplikace pro rozpoznávání ručně psaného textu
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2016
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	79
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Tomáš Urbanec
Title:	Application for handwritten text recognition
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2016
Study field:	Computer Science, full-time form
Supervisor:	RNDr. Miroslav Kolařík, Ph.D.
Page count:	79
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Práce popisuje strojové zpracování ručně psaných dokumentů od okamžiku jejich získání až po základní možnosti exportu. Všechny popsané algoritmy jsou implementovány v přiložené aplikaci, která umožňuje jejich otestování na vlastním rukopise.

Synopsis

The thesis describes handwritten document processing by a computer. Algorithms for all of the main phases of the process are described and implemented together with the graphical interface which allows high level of user interaction.

Klíčová slova: OCR, HCR, ICR, HWR, ručně psaný text, digitalizace textu, digitalizace dokumentu

Keywords: HCR, OCR, ICR, HWR, handwriting recognition, handwritten character recognition, optical character recognition, cursive text recognition, text digitization, document digitization

Děkuji rodině za podporu během celého studia,
zejména matce a přítelkyni.
Vedoucímu práce děkuji za trpělivost.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Práce, kterou právě držíte v ruce, podává úvod do procesu zpracování ručně zpracovaných dokumentů od okamžiku jejich získání až po uložení v elektronické formě. Pro tuto činnost se nejčastěji používá zkratka OCR (z anglického Optical Character Recognition). Nejprve tedy uveďme, proč je tento problém zajímavý a co OCR vlastně přesně znamená.

1.1 Motivace

Téma práce jsem si vybral z několika důvodů. První motivací byl fakt, že bych rád měl k dispozici aplikaci pro digitalizaci svých poznámek nastřádaných během několika let studia. To mě vedlo už k volbě tématu bakalářské práce, která se zabývala rozpoznáváním písmen pomocí fuzzy logiky. Její výsledky a nabyté znalosti pak byly jedním z hlavních důvodů pro volbu tématu této práce.

V neposlední řadě je problém rozpoznávání ručně psaného textu velice zajímavý a v jeho obecné formě stále nevyřešený. Je to tedy dobrá příležitost k práci na něčem aktuálním.

1.2 Co a k čemu je OCR?

OCR je souhrnné označení technik používaných při digitalizaci textu z fotografií, skenů či jinak získaných tištěných nebo ručně psaných textů.

1.3 Historie OCR

Mohlo by se zdát, že potřeba počítačového zpracování textu je záležitost posledních pár desetiletí, nicméně opak je pravdou. První patent na myšlenku strojového zpracování textu si zaregistroval G. Tausheck v Německu již v roce 1929 těsně následován (1933) P. W. Handelem v USA. K dnešním, často sofistikovaným algoritmům tenkrát sice měli daleko – jednalo se o použití mechanických strojů hledajících přesnou shodu se vzorem – ale započali tím nový směr bádání, který v dnešní účinné algoritmy nakonec vyústil.[1]

1.3.1 Dělení OCR

Původně se jednalo pouze o tištěné texty, ale díky dnešnímu výkonu počítačů je do určité míry možné zpracovávat i ručně psané texty. Proto bylo zavedeno několik kategorií OCR. Prvním důležitým aspektem je, jestli se jedná o offline nebo online variantu.

Online rozpoznávání je prováděno během psaní textu člověkem na nějakém snímacím zařízení, například mobilním telefonu, tabletu či nějakém přímo pro tento účel navrženém zařízení.

Offline varianta je naopak prováděna na nějakém již dříve napsaném dokumentu, například novinové články, staré dokumenty nebo poznámky z přednášek.

Dalším důležitým faktorem je, jestli se jedná o text tištěný, nebo ručně psaný. Zkratka OCR se dnes nejčastěji používá pro zpracování tištěného textu. Zpracování ručně psaného textu se pak většinou označuje jako HCR (z anglického Handwritten Character Recognition), HWR (z anglického HandWritten Recognition) nebo ICR (z anglického Inteligent Character Recognition).

Pro ručně psaný text je pak problém dále rozdělen na zpracování textu psaného hůlkovým písmem s oddělenými znaky a zpracování textu psaného libovolně. Jendou z nejnovějších metod pro libovolně ručně psaný text je pak IWR (z anglického Intelligent Word Recognition), kdy jsou místo znaků rozpoznávána celá slova.

1.3.2 Použitelnost OCR

Z hlediska obtížnosti a použitelnosti je na tom online varianta mnohem lépe než offline varianta, neboť má k dispozici velké množství dat o rychlosti a tlaku jednotlivých tahů, které umožňují mnohem jednodušší rozdělení slov na písmena a jejich další zpracování.

Stejně tak je rozpoznávání tištěných textů mnohem snazší než rozpoznávání ručně psaných textů. Pro vstupní data dostatečné kvality dosahuje rozpoznávání tištěných textů vysoké úspěšnosti.

Na druhou stranu rozpoznávání ručně psaných textů je dnes použitelné jen pro text psaný oddělenými hůlkovými písmeny. Reálně je tato podoblast používána například pro detekci poštovních směrovacích čísel na dopisech, rozpoznávání státních poznávacích značek na fotografiích nebo při zpracování různých formulářů.

1.4 Existující řešení

Pro digitalizaci tištěných textů existuje mnoho nástrojů s vysokou úspěšností. Některé z umožňují i zpracování ručně psaného textu pro hůlkové písmo. Obecně ale výsledky zatím nejsou dostatečně dobré pro nasazení v reálných aplikacích.

Pravděpodobně nejznámějšími a nejpoužívanějšími nástroji v této oblasti jsou:

- Google Tesseract původně vyvinutý firmou HP, dnes volně dostupný.
- ABBYY finereader, placený produkt.
- Nuance Omnipage, placený produkt.

Pro zpracování libovolně ručně psaného textu jsem nenašel žádný dostupný nástroj. Existují programy, jako například NeurophOCR [2], pro rozpoznávání jednotlivých písmen. Trendem posledních let se zdá být použití „deep learning“ algoritmů.

1.5 Vymezení tématu

Tato práce si klade za cíl prozkoumat problém offline analýzy ručně psaného textu s libovolným rukopisem. Tedy nejobecnější variantou z výše vyjmenovaných.

1.6 Poznámka ke zdrojům

Před samotným popisem algoritmů a postupů použitých v práci bych ještě čtenáře rád upozornil, že v této oblasti je dnes publikováno mnoho prací. Naneštěstí ne všechny svou kvalitou odpovídají tomu, co uvádí v abstraktu či závěru. Při výběru zdrojů je tedy potřeba obezřetnosti, neboť studiem a implementací takových postupů člověk (jak jsem se sám přesvědčil) ztratí mnoho času. Doporučuji tedy věnovat zvýšenou pozornost zejména popisům testovacích dat, jejich množství a případné dostupnosti.

2 Úprava vstupu

Prvními kroky při zpracování dokumentu jsou úpravy vstupního obrázku. Ač se tento krok může jevit triviálním, tak je ve skutečnosti minimálně stejně důležitý a často i obtížný jako samotné určování znaků. Bez vhodně upraveného vstupu totiž pozdější algoritmy vůbec nemohou pracovat. Je-li to možné, pak je vhodné na pozdější rozpoznávání prvků dokumentu brát ohled již při získávání dokumentu, ať již skenováním, focením nebo jinak. Obecně platí, že čím více informací a detailů při digitalizaci zachováme, tím lepší budou výsledky rozpoznávání.

Mezi úpravy prováděné při samotném strojovém zpracování pak patří například binarizace vstupního obrázku, oprava pootočení způsobeného skenováním nebo filtrování chyb vzniklých při skenování (prach na skeneru, pokrčený papír, atp.). Důležité jsou pak i různé úpravy specifické pro jednotlivé algoritmy v pozdějších fázích: detekce hran, hledání koster objektů, hledání spojitých objektů, úpravy velikosti...

V práci je implementováno mnoho různých algoritmů a nástrojů používaných jak při samotné úpravě vstupu, tak později používanými algoritmy. Zde však budou popsány jen ty z autorova pohledu nejzajímavější a nejdůležitější.

2.1 Požadavky na vstup

Před samotným popisem použitých algoritmů uveďme požadavky na vstupní dokumenty pro použitelnost doprovodné aplikace:

1. Tmavý (ideálně černý) text na světlém (ideálně bílém) pozadí.

2. Dostatečná kvalita psacího nástroje a získaného skenu. Při testování se osvědčilo použití černé gelové propisky a skenování při 300 dpi.

3. Dokument nesmí mít černé okraje vzniklé například při skenování. Výjimkou je tabulka přes celý dokument.

4. Pro použití vyrovnávacího algoritmu musí mít dokument úhel rotace mezi $-40°$ a $40°$. V opačném případě musí uživatel sklon dokumentu nejprve upravit(lze provést v aplikaci) tak, aby byl v uvedeném rozmezí.

5. Dokument musí být rozložitelný na obdélníkové oblasti obsahující pouze jednotlivé logické celky (obrázky, tabulky, textové oblasti). Výjimkou je textová vrstva nejvyšší úrovně, která je zpracovávána odděleně od tabulek a obrázků.

6. Mezi každými dvěma řádky textu musí být mezera po celé šířce nadřazené oblasti. V případě textové vrstvy nejvyšší úrovně musí mezery procházet celu šířkou dokumentu (po případné extrakci obrázků a tabulek).

7. Mezi každými dvěma slovy v řádku musí být mezera přes celou výšku řádku.

8. Tabulky mohou obsahovat jen text.

9. Tabulky musí mít pravidelnou mřížkovou strukturu bez slitých sloupců či řádků.

10. Obsah tabulek se nesmí dotýkat jejích hranic.

2.2 Binarizace

Prvním důležitým krokem při zpracování vstupního obrázku je jeho převedení do vhodného barevného modelu. Ve většině případů se při detekci a rozpoznávání používá pouze černá a bílá barva a samotný proces převádění se nazývá binarizace. Nejinak je tomu v této práci.

Zde použitý algoritmus se po svém tvůrci nazývá Otsuova metoda. Poprvé popsána byla v práci [3]. Jedná se o statistickou metodu pro nalezení ideálního prahu pro převod šedotónového obrázku na černobílý. V závislosti na tom, jestli hledáme světlejší objekt na tmavším pozadí nebo naopak, je ve výsledném obrázku popředí bílé a pozadí tmavé nebo naopak. V našem případě vždy hledáme tmavé popředí na světlém pozadí.

Metoda je založena na postupné analýze všech možných prahových hodnot, kterých je typicky 256. Za ideální práh je pak vybrána taková hodnota t, která minimalizuje vztah

$$\sigma_W^2 = W_b \cdot \sigma_b^2 + W_f \cdot \sigma_f^2$$

W_b je poměr počtu pixelů, které při použití t budou v pozadí (jejich hodnota je blíže bílé), ku celkovému počtu pixelů; W_f je poměr počtu pixelů, které při použití t budou v popředí (jejich hodnota je blíže černé), ku celkovému počtu pixelů a σ_*^2 jsou příslušné výběrové rozptyly.

Při implementaci se využívá odvozeného vztahu

$$\sigma_b^2 = W_b \cdot W_f \cdot (\mu_b - \mu_f)^2$$

kde μ_b je průměrná hodnota pixelů pozadí při dané t a μ_f je průměrná hodnota pixelů popředí při dané t. Implementace s tímto vztahem je efektivnější a jeho hodnota je maximalizována, pravě když je hodnota σ_W^2 minimalizována. Při implementaci bylo čerpáno jak z původního zdroje [3] tak z [4].

Poznámka Otsuova metoda vyžaduje na vstupu šedotónový obrázek. Je-li tedy načten obrázek barevný, je třeba jej nejprve převést do stupňů šedi.

2.3 Filtrování šumu

Mezi důležité kroky úvodního zpracování vstupního obrázku patří také zlepšování jeho kvality. Zejména jde o vyčištění obrázku od případných naskenovaných prachových částeček a podobných nečistot, vyhlazení nejednotného tahu psacího nástroje, apod. V práci je pro tento účel použit soubor různých filtrů a utilit, kde

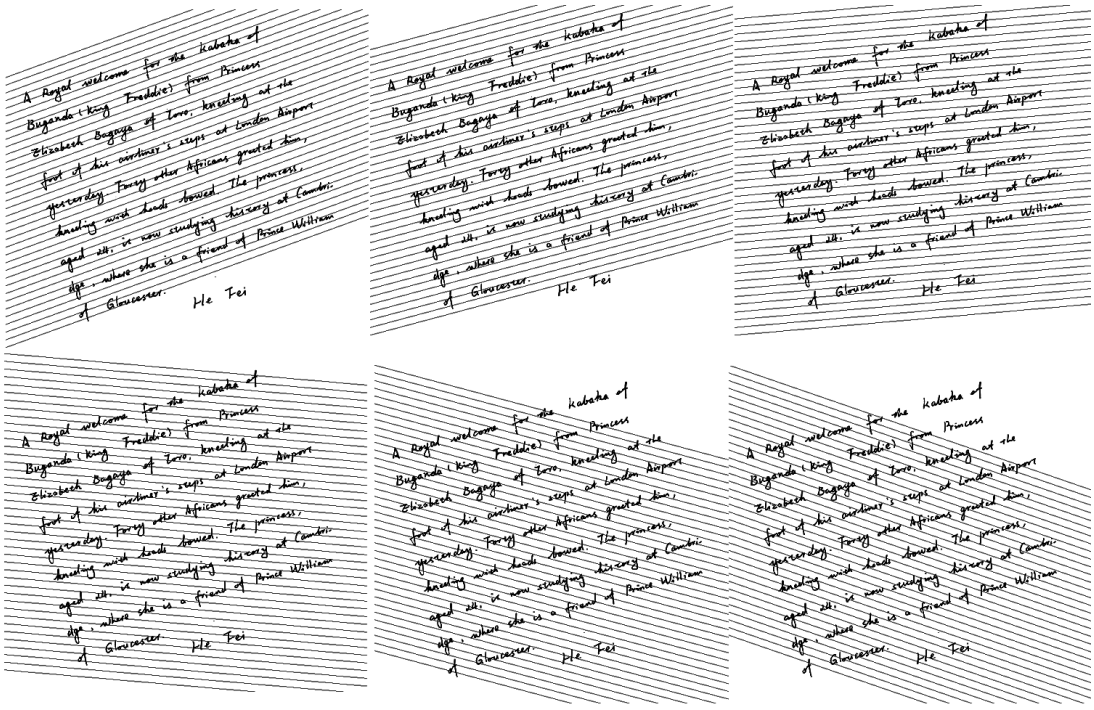
některé z nich pocházejí z práce [5]. Ve většině případů jde o použití (případně vícenásobné) konvoluce.

2.4 Úprava sklonu

Dalším důležitým krokem je oprava případné drobné rotace vzniklé při získávání obrázku. Vzhledem k použitým algoritmům by totiž pro některé takové obrázky nemusela fungovat segmentace textu na řádky a slova a tedy ani žádný z později používaných algoritmů.

Algoritmus použitý v práci používá přímočarou myšlenku, kterou lze najít v různých programech a pracích pracujících jak se strojovým, tak ručně psaným textem. Naneštěstí se mi nepodařilo najít původní zdroj. Nicméně postup již byl použit například v pracích [6] a [7]. Ani zde jsem ovšem nenašel žádný odkaz na dřívější zdroje.

Základní idea je proložit obrázkem rovnoběžné úsečky v různých úhlech a vybrat ten úhel, který maximalizuje rozptyl počtu černých pixelů, kterými jednotlivé úsečky prochází. Výsledný obrázek je pak vstup otočený o opačnou hodnotu takto detekovaného úhlu. Viz obrázek 1.



Obrázek 1: Myšlenka algoritmu úpravy sklonu. Vstupním obrázkem jsou proloženy rovnoběžky v různých úhlech. Úhel, u kterého je největší rozptyl mezi počty protnutých černých pixelů, je vybrán jako úhel rotace. Zde ukázka pro úhly: -20, -15, -5, 5, 15 a 20 stupňů. Výsledkem je pak úhel -15 stupňů.

Přesněji nejprve pro daný úhel α a frekvenci prokládání úseček určíme, které pixely jednotlivé úsečky pokryjí. Toho je v práci docíleno tak, že se nejprve najde jedna z takových úseček pomocí určení jejích krajních bodů (jeden bod zvolíme na levém okraji a za pomocí α a goniometrických funkcí dopočítáme druhý bod na pravém okraji) a použitím Bresenhamova algoritmu [8]. Ostatní úsečky bychom mohli najít stejným způsobem. Víme ale, že musí být rovnoběžné s tou, kterou již máme. Pro jejich nalezení tedy stačí posunout souřadnice každého z bodů nahoru nebo dolů. Takovýto postup získání každé rovnoběžky výrazně urychlí, a tedy nám umožní získat jich více. To se nám hodí, neboť čím více úseček obrázkem proložíme, tím přesnější data získáme.

Nyní, když máme souřadnice všech pixelů všech úseček, spočteme pro každou z nich počet černých pixelů, které protnula. Z těchto hodnot spočítáme rozptyl. Uvažujeme rovnoměrné rozdělení dat a použijeme vztah

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

kde $E(x)$ je střední hodnota.

Pro každý z možných úhlů otočení tedy známe rozptyl počtu černých pixelů. V námi hledaném případě, kdy jsou řádky rovnoběžné s danými úsečkami, bude rozptyl maximální, neboť každá úsečka protne buďto právě jeden řádek v celé délce (a tedy projde mnoha černými pixely) nebo mezeru mezi řádky (a tedy protne minimum černých pixelů). Viz obrázek 1, případ 2. Tedy úhel spojený s maximální rozptylem spočtených hodnot je úhel, o který byl původní obrázek otočen. Pro nápravu stačí otočit obrázek o opačný úhel.

2.5 Podpůrné algoritmy

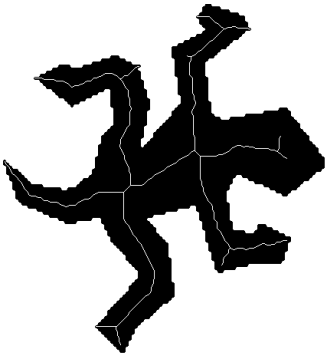
V práci je použito také několik podpůrných algoritmů, které jsou často využívány k připravení vstupu pro některé ze v pozdějších fázích používaných algoritmů.

2.5.1 Kostra objektu

Pro některé z dále používaných algoritmů je nezbytné znát kostru objektu. Pro koncept kostry objektu v obrázku existuje několik různých, více či méně formálních, definic.

Pro další pochopení je spíše než formální definice potřebný koncept kostry. Použijeme tedy neformální popis přirovnáním k prérijnímu ohni tak, jak je uveden v [9]: Hranice objektu v obrázku je celá v jeden okamžik zapálena, *kostrou objektu* jsou pak ta místa, na kterých se setká oheň ze dvou čí více směrů. Pro formální definice a různé pohledy na problém a jeho řešení vizte [9]. Příklad objektu a jeho kostry vidíme na obrázku 2.

Pro vytvoření kostry je v práci použit algoritmus z článku [10] s několika úpravami. Jde o iterační metodu, která v každé iteraci provede dva průchody obrázkem a při každém průchodu odstraní některé z pixelů.



Obrázek 2: Objekt a jeho kostra. Obrázek pochází z [9].

Obrázek 3: Indexování sousedních pixelů pixelu P_1 při hledání kostry.

$P_9(x-1, y-1)$	$P_2(x, y-1)$	$P_3(x+1, y-1)$
$P_8(x-1, y)$	$P_1(x, y)$	$P_4(x+1, y)$
$P_7(x-1, y+1)$	$P_6(x, y+1)$	$P_5(x+1, y+1)$

Při prvním průchodu odstraní přebytečné pixely na pravé a spodní straně objektu a případný přebytečný levý horní roh. Tedy pixely $P1$, které splňují všechny následující podmínky:

1. $2 \leq B(P_1) \leq 6$
2. $A(P_1) = 1$
3. $P_2 \cdot P_4 \cdot P_6 = 0$
4. $P_4 \cdot P_6 \cdot P_8 = 0$

kde P_* jsou pixely indexované dle obrázku 3; $B(p)$ značí počet sousedů bodu p různých od 0 a $A(p)$ značí počet přechodů z 0 do 1 v posloupnosti $P_2, P_3, P_4, \ldots, P_8, P_9$.

Podobně při druhém průchodu jsou odstraněny přebytečné pixely na horní a levé straně objektu a případně přebytečný pravý dolní roh. Tedy ty pixely, které splňují podmínky 1 a 2 z prvního průchodu a navíc:

1. $P_2 \cdot P_4 \cdot P_8 = 0$
2. $P_2 \cdot P_6 \cdot P_8 = 0$

Algoritmus skončí, když dokončená iterace neprovedla žádnou změnu.

Úpravy Původní algoritmus používá lehce odlišnou definici kostry, než očekávají algoritmy v dalších částech práce. V implementaci je tedy na konec přidána

ještě jedna iterace pro odstranění jednoho pixelu z každé trojice, kde každý je sousedem každého, tak, aby byla zachována propojenost.

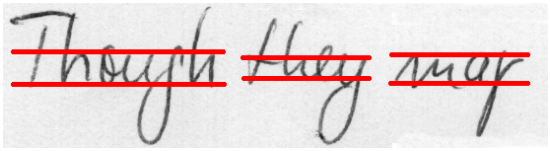
Další úpravou algoritmu je propojení nedotažených čar a blízkých volných konců ve výsledné kostře. Propojení nedotažených čar je provedeno, pokud prodloužení čáry o zlomek její délky protne další čáru. Spojení volných konců je provedeno, pokud jsou si dostatečně blízko.

Poznámka Algoritmus je, na rozdíl od většiny ostatních algoritmů popsaných v práci, definován a popsán pro binární obrázky, kde 1 znamená pixel popředí a 0 znamená pixel pozadí.

2.5.2 Jádro slova

Dalším algoritmem užitečným zejména při segmentaci slova na znaky je určení jádra obrázku slova. Rozdělíme-li obrázek slova (či řádku) pomocí dvou čar jako na obrázku 4, tak získáme tři zajímavé oblasti:

1. Oblast obsahující vrchní části vysokých znaků
2. *Jádro obrázku slova* – oblast obsahující většinu pixelů písmen.
3. Oblast obsahující spodní části znaků protínajících (pomyslný) vodící řádek.



Obrázek 4: Příklady jádra slova. Jádro každého slova je jeho oblast mezi červenými čarami.

V práci je jádro získáno ve třech krocích. Nejprve je nalezen první odhad středu slova a to jako průměrná ypsilonová souřadnice černého pixelu slova. Přesněji průměr průměrů ypsilonových souřadnic černých pixelů jednotlivých pixelových sloupců. Následně je nalezen pixelový řádek s maximálním počtem černých pixelů v okolí tohoto prvního odhadu. Tímto získáme pixelový řádek s nejvíce černými pixely v okolí středu slova nezávisle na tom, jestli slovo obsahuje nějaká vysoká či vodící řádek protínající písmena. Posledním krokem je pak nalezení hranice nad získaným středem, kde se prudce snižuje počet černých pixelů, a tím určení vrchní hranice jádra slova. Obdobně, jen směrem dolů od nalezeného středu, pro spodní hranici jádra slova.

2.5.3 Spojité komponenty

Hledání spojitých komponent v obrázku je další široce využívaný algoritmus při segmentaci jak celého dokumentu, tak tabulek a různých textových částí. *Spojitá*

komponenta v obrázku je (podobně jako v teorii grafů) množina všech pixelů takových, že z každého do každého lze přejít průchodem jen přes sousední pixely stejné barvy (z každého do každého existuje cesta).

Pro hledání spojitých komponent je v práci implementován klasický flood–fill algoritmus, který je typicky využíván k implementaci „kbelíku“ v grafických editorech. Jméno autora algoritmu se mi nepodařilo vypátrat, ale informace o algoritmu lze nalézt například v [11]. Původně byla zvažována známější rekurzivní verze, nicméně ta má při větších komponentách (typicky tabulky a obrázky) zbytečně velké nároky na zdroje. Nakonec tedy byl místo rekurze použít zásobník.

3 Segmentace dokumentu

V okamžiku, kdy máme černobílý, vyrovnaný a od případného šumu vyčištěný dokument, můžeme přistoupit k další fázi zpracování: segmentaci dokumentu na základní logické celky, jako jsou textové oblasti, tabulky a obrázky.

3.1 Segmentace základních oblastí

Při zpracování je v dokumentu důležité nejprve detekovat obrázky a tabulky. Pro tuto část je implementován základní algoritmus využívající toho, že jak obrázky tak tabulky bývají zpravidla mnohem větší než slovo či znak. Nejprve jsou tedy nalezeny všechny spojité oblasti v dokumentu. K tomu je použit algoritmus spojitých komponent popsaný výše (2.5.3).

Po získání všech komponent jsou u každé z nich určeny opsané obdélníky a spočítán jejich obsah. Komponenty, u kterých je obsah větší než několikanásobek průměrného obsahu obdélníku opsaného nějaké komponentě, jsou vyhodnoceny jako netextové oblasti a jsou odstraněny z textové vrstvy a postoupeny dalšímu zpracování.

Nyní je nalezena komponenta označena za tabulku a dojde k pokusu o její segmentaci (viz dále 3.2). Pokud segmentace vrátí smysluplný výsledek, je komponentě status tabulky ponechán a navíc je dle dodaného výsledku rozdělena na buňky. V opačném případě je komponenta označena za obrázek. Po zpracování všech příliš velkých komponent tedy obdržíme část dokumentu obsahující jen data podobná textu a kolekci ostatních částí.

V doprovodné aplikace je alternativou ruční segmentace dokumentu uživatelem (viz 7.5.4).

3.2 Segmentace tabulek

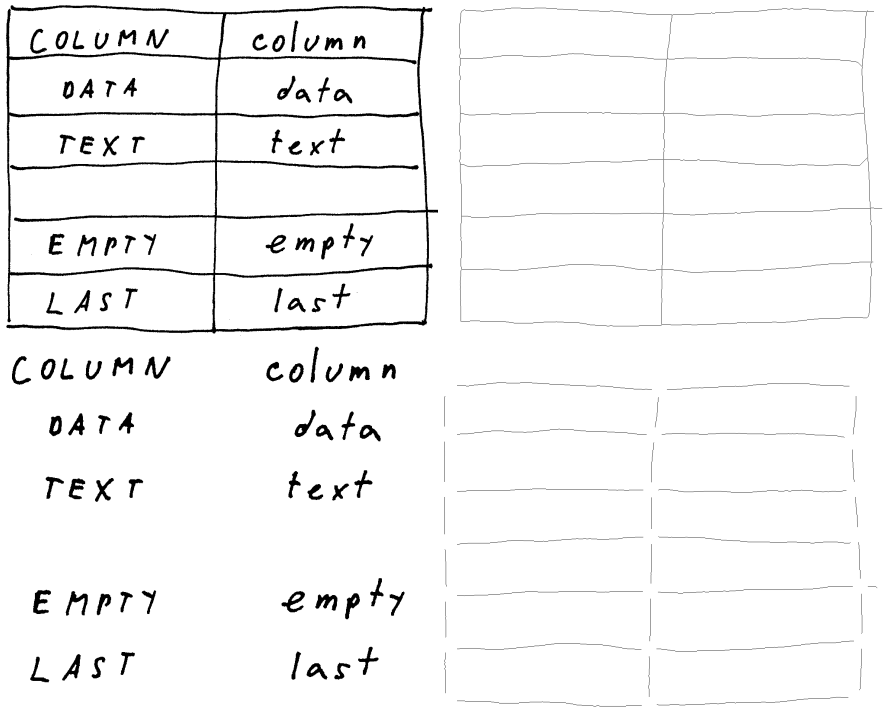
Máme–li identifikovány všechny tabulky, přejdeme k hledání jednotlivých buněk. Pro tento účel jsem si zkusil navrhnout vlastní algoritmus. Výsledný postup je nicméně natolik intuitivní, že jej nelze označit za nový.

Algoritmus na vstupu očekává tabulku splňující podmínky uvedené v 2.1. Použitý postup lze popsat v šesti krocích:

1. Nejprve musíme najít a oddělit mřížku a data tabulky (viz obrázek 5 vpravo nahoře a vlevo dole). K tomu je použit algoritmus pro hledání spojitých komponent popsán výše.

2. Dále je nutné určit důležité body mřížky. To znamená všechna křížení a rohy. V doprovodné aplikaci je tohoto docíleno takto:

   - Nejprve najdeme kostru mřížky (2.5.1).
   - Nyní detekujeme všechny body mřížky, které mají více než dva sousedy. Toto jsou body křížení. Každý z nich uložíme a následně spolu



Obrázek 5: Postup segmentace tabulky. Pro vstupní tabulku (první obrázek) oddělíme její mřížku (druhý obrázek) a data (třetí obrázek). V mřížce detekujeme významné body – přerušení mřížky na posledním obrázku – a ty postoupíme dalšímu zpracování.

s jeho okolím odstraníme z mřížky. Tím zamezíme několikeré detekci jednoho bodu křížení, způsobené občasnými artefakty ze skenovacího a binarizačního procesu.

- Na mřížce tedy zůstávají jen nezajímavé hrany a některé typy rohů (to závisí na vytvořené kostře). Projdeme tedy upravenou mřížku znovu a extrahujeme ještě ty body, v jejichž okolí dochází k protnutí dvou sousedních hranic okolí (např pravá a spodní, nebo levá a horní, ...).
- Postup je ilustrován na obrázku 5.

3. Máme oddělena data od mřížky a známe důležité body na mřížce. Stačí tedy body rozdělit do skupin definujících jednotlivé buňky. V práci je toto provedeno detekováním sloupců důležitých bodů. Následuje vytváření řádků.

4. Posledním krokem je dříve získaný obrázek dat tabulky (bez mřížky) dle výsledných buněk rozdělit.

Pokud jsou data v předposledním kroku nesmyslná vzhledem k vstupním omezením (např. různé počty řádků v jednotlivých sloupcích), tak výsledek nemusí odpovídat realitě. Náprava takové situace je v doprovodné aplikaci ponechána na uživateli (viz 7.5.5).

3.3 Segmentace textu na řádky a slova

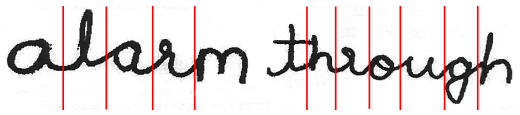
Máme-li izolované jednotlivé textové oblasti, musíme je rozdělit na řádky a ty pak na slova. Pro obě tato rozdělení je v práci použit stejný princip analýzy součtů hodnot pixelů v jednotlivých sloupcích či řádcích. Podobně jako u úpravy sklonu dokumentu (2.4) je i tento algoritmus široce používaný, ale původní zdroj mi není znám. První zdroj, ve kterém jsem jej našel, je [12].

Při hledání textových řádků nejprve sečteme hodnoty pixelů v každém pixelovém řádku. Čím více černých pixelů (a tedy i písmen) pixelový řádek protíná, tím menší bude odpovídající součet (předpokládáme, že hodnota bílé barvy je větší než hodnota barvy černé). Jako prahovou hodnotu použijeme šířku dané textové oblasti s případnou tolerancí pro rukopisy s protínajícími se spodními částmi písmen vrchního řádku a vrchními částmi písmen spodního řádku. Spojité oblasti pixelových řádků se součty menšími než prahová hodnota určují textové řádky a naopak spojité oblasti pixelových řádku se součty většími než prahová hodnota určují mezery mezi textovými řádky. Spojitou oblastí pixelových řádků zde myslíme posloupnost po sobě jdoucích pixelových řádků $\langle i, i+1, i+2, \ldots, j \rangle$ takovou, že součty příslušné každému z nich jsou ve stejné relaci větší/menší vůči prahové hodnotě.

Po získání jednotlivých řádků je úplně stejný postup, jen na pixelových sloupcích, použit pro rozdělení řádku na slova.

3.4 Segmentace slov na znaky

Dělení slov na znaky se ukázalo být jedním z největších problémů při digitalizaci ručně psaného textu. Ideální stav, kdy člověk píše každé písmeno odděleně (a tedy lze použitý algoritmus spojitých komponent), nastává zřídkakdy. Zejména při používání psacího písma jsou téměř všechna písmena spojena s minimálně jedním dalším. Dále různé výskyty téhož písmene mohou být odlišné a to jak tvarem, tak rozměry. Stejně tak spojnice mezi písmeny mohou mít různé rozměry, pozice a tvary. Obrázek 6 obsahuje příklady takových písmen.



Obrázek 6: Typické problémy při segmentaci. Červené čáry označují ideální místa rozdělení. Vlevo si lze všimnout dvou velmi odlišných „a“ v jednom slově; vpravo pak nastává podobný případ pro „h“. Také se zde nachází několik různých typů spojnic mezi písmeny, nejvýznačnější je pak dvojitá spojnice mezi „t“ a „h“ ve slově „through“. Obrázky slov pochází z [13].

Pro segmentaci slov bylo navrženo mnoho algoritmů. Dle [14] je však lze rozdělit do tří základních skupin

1. Klasický přístup, kdy jsou výsledné segmenty určeny na základě jejich vlastností. Tyto vlastnosti jsou obecné vlastnosti segmentu, které určují

jeho „znakovost“. Příkladem mohou být výška a šířka segmentu, jeho separace od sousedních segmentů, atp.

2. Přístup založený na rozpoznávání znaků, kdy algoritmus prochází vstup a hledá segmenty s nejlepším výsledkem dle nějakého algoritmu rozpoznávání znaků.

3. Holistické metody, kdy nám nejde o rozdělení slova na znaky, ale o určení slova jako celku. Často jsou nazývané také IWR (viz 1.3.1).

V této práci, kvůli povaze zadání, nemohou být holistické metody použity, neboť by to od každého uživatele vyžadovalo naučení všech slov, která by někdy v budoucnu mohla být rozpoznávána. To by aplikaci činilo prakticky nepoužitelnou. Na druhou stranu naučení několika verzí od každého znaku lze dosáhnout zpracováním několika dokumentů.

V úvahu tedy připadá klasický přístup nebo přístup založený na rozpoznávání znaku. U čistě klasického přístupu by byl problém nalézt množinu vlastností, které specifikují znak při různých rukopisech a velikostech písma. Toto je ostatně vidět již na obrázku 6, kde i stejná písmena psána stejným autorem v rámci jednoho slova vypadají jinak. Na druhou stranu u přístupu založeném jen na rozpoznávání znaků je mnoho možností, které je potřeba ověřit (tzn. zkusit rozpoznat), což může vést k dlouhé době běhu a mnoha falešným detekcím dělení.

Volba tedy padla na kombinaci klasického přístupu a přístupu založeného na rozpoznávání znaku, kdy klasický přístup nejprve najde vhodné kandidáty na místa rozdělení slova a přístup založený na rozpoznávání z těchto kandidátů následně vybere ty nejlepší.

Nejprve popíšeme algoritmus vybírající z kolekce dodaných možných míst dělení ta vhodná, a to s dopomocí rozpoznávacího algoritmu. Poté se budeme věnovat omezení této kolekce (kandidátů na místa dělení) algoritmem klasického přístupu.

Pro porozumění následujícím několika stranám je nezbytné, aby čtenář měl základní přehled o teorii grafů. Pro osvěžení potřebných pojmů lze použít například [15].

3.4.1 Výsledná segmentace jako cesta grafem

Segmentačních algoritmů založených na výsledcích rozpoznávání jednotlivých segmentů existuje mnoho. Asi nejčastějším přístupem je použití pomyslného okna, které je posouváno vstupním obrázkem a pomocí rozpoznávání aktuálního obsahu identifikuje kandidáty na znaky. Ačkoliv článek [16] popisující pro práci zvolený algoritmus o posuvném okně vůbec nemluví, tak svým principem pokryje všechna možná okna na vstupním obrázku, pouze tak činí skrytě, v pozadí.

K této volbě mě vedlo několik důvodů. Prvním důvodem byla elegance navrhované přístupu, druhým pak vysoká úspěšnost popsaná v citované práci, téměř 80%. Tato hodnota je dle srovnání různých technik segmentace slov na úrovni

dnešních možností [17]. V neposlední řadě hrál roli i fakt, že práce, ze které metoda pochází, je napsána kvalitně a obsahuje dostatečné množství testů, příkladů a výsledků – toto pro mě bylo důležité zejména vzhledem k diskuzi v odstavci 1.6. Mezi další zajímavé vlastnosti pak patří: snadná propojitelnost s libovolným algoritmem klasického přístupu, což umožní další zlepšení výsledků; libovolný počet znaků na slovo, přičemž tento počet nemusí být předem znám; není nutná předchozí normalizace výšky či šířky znaku (tj. pomyslné posuvné okno má pružné rozměry).

Myšlenka
Nejprve popíšeme základní myšlenky algoritmu a každou z nich později rozvedeme. Segmentace probíhá ve čtyřech krocích. Na začátku je slovo (spojitý text) hustě a rovnoměrně rozděleno pomocí vertikál, které tvoří kandidáty na hranice segmentů.

Následně je vytvořen orientovaný graf, který reprezentuje právě možné hranice a znaky mezi nimi. Kandidáti tvoří vrcholy a mezi každými dvěma vrcholy je vytvořena hrana reprezentující podobnost daného segmentu nějakému znaku.

V tomto grafu je pak problém segmentace textu redukován na problém hledání průměrně nejdelší cesty mezi prvním a posledním vrcholem, kdy optimální řešení lze najít v polynomiálním čase. Graficky jsou jednotlivé kroky algoritmu zobrazeny na obrázku 7.

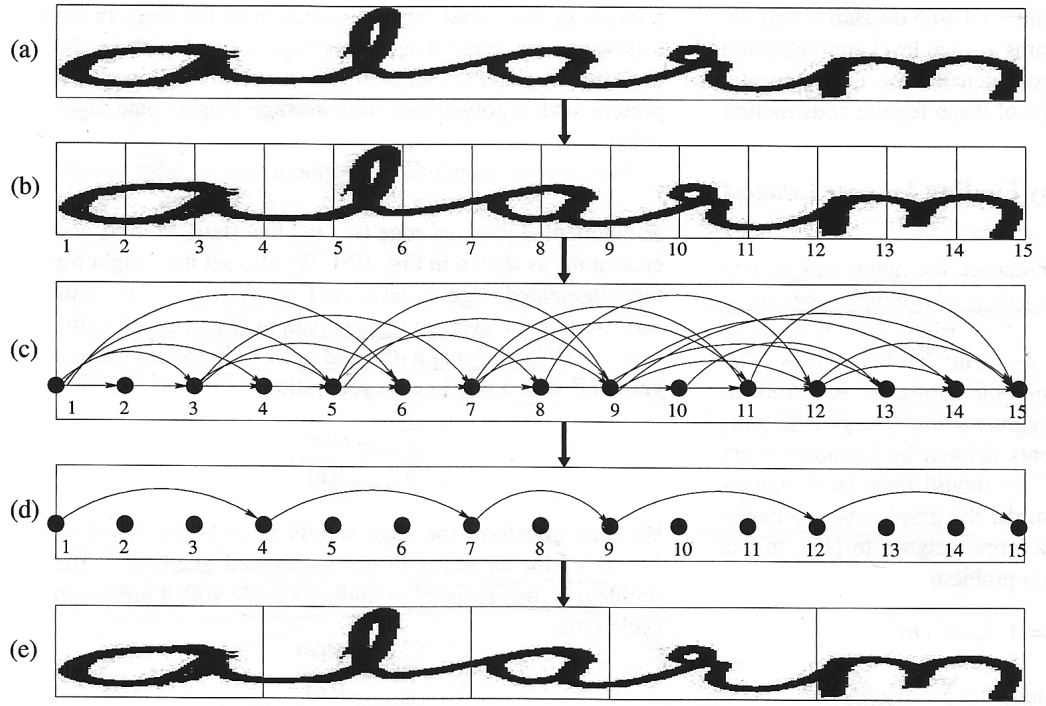
Úvodní segmentace
Nejprve potřebujeme vytvořit kandidáty na hranice jednotlivých znaků. Je důležité, aby všechny skutečné hranice byly pokryty kandidáty, a to včetně nejpravější a nejlevější. Na druhou stranu je algoritmus schopen zavrhnout ty kandidáty, kteří jsou na špatné pozici. Nejvhodnější je tedy slovo rozdělit rovnoměrně a hustě. Dle [16] lze v extrémním případě použít jako kandidáta každý pixelový sloupec. Tento postup ovšem vede k velké potřebě zdrojů. V doprovodné aplikaci je tento krok obohacen o použití segmentačního algoritmu klasického typu a tím je množství kandidátů výrazně redukováno (viz 3.4.2).

Grafová reprezentace
Když máme k dispozici kandidáty na hranice znaků, tak problém určení hranic znaků převedeme do grafové reprezentace.

Pro množinu kandidátů $\{S_1, S_2, \ldots, S_n\}$ uspořádaných zleva doprava dle vstupního obrázku vytvoříme graf $G = \langle V, E \rangle$ následovně:

1. Každý kandidát S_i je reprezentován pomocí $v_i \in V$.
2. Pro každou dvojici v_i, v_j, kde $i < j$ vytvoříme hranu $e_{ij} = \langle v_i, v_j \rangle \in E$. Hrany tedy odpovídají možným znakům.

Výsledný graf G plně popisuje všechny doposud získané informace. Jeho vrcholy odpovídají kandidátům a hrany mezi nimi potencionálním znakům ve slově



Obrázek 7: Průběh segmentace slova „alarm“: (a) Vstupní obrázek. (b) Množina úvodních kandidátů. (c) Graf reprezentující možné znaky (nejsou zobrazeny všechny možnosti). (d) Cesta grafem reprezentující výslednou segmentaci. (e) Výsledná segmentace. Obrázek pochází z [16].

tak, že hrana $\langle v_i, v_j \rangle$ reprezentuje segment vstupu mezi pixelovými sloupci S_i a S_j.

Ohodnocení hran

Nyní je potřeba určit, jak moc jednotlivé segmenty odpovídají znakům. K tomu je použit externí OCR (ICR) algoritmus A. Pro každou hranu $e = \langle v_i, v_j \rangle$ aplikujeme A na odpovídající segment (pro hranu e je to segment mezi S_i a S_j). Na výstupu A pak očekáváme nejen znak nejpodobnější danému segmentu, ale hlavně výsledný stupeň podobnosti. Tento je pak použit jako váha hrany e. Motivace je taková, že pouze hrany reprezentující segmenty odpovídající znakům budou mít vysokou váhu.

Ideální cesta grafem

Máme–li vybudovaný graf a ohodnocené všechny jeho hrany, tak musíme vybrat, kteří z kandidátů tvoří průměrně nejdelší cestu grafem.
Průměrně nejdelší cesta grafem je taková cesta z v_1 do v_n, kde průměrná váha

hrany v cestě je maximální. Tedy hledáme takové m^* a $v_i^*; i \in 1, \ldots, m^*$, kde

$$(m^*, \{v_i^*\}) = \arg\max_{m, v_i} \frac{\sum_{i=1}^{m-1} w_{v_i, v_i+1}}{m-1}$$

kde m je počet vrcholů v dané cestě a w_{v_i,v_i+1} je váha hrany $e = \langle v_i, v_{i+1} \rangle$. Jinými slovy hledáme takové vrcholy $v_1, \ldots, v_{m^*}$, kde průměrná váha hrany mezi v_i a v_{i+1} je maximální.

Takovou cestu lze dle [16] najít redukcí na maximalizační problém hledání orientovaného cyklu v grafu G_2 odvozeného z G následovně:

1. Nejprve do G přidáme hranu z v_n do v_1.
2. Dále zavedeme druhou váhu každé hrany $l_{e_{i,j}} = \begin{cases} 1 & \text{pro } i < j \\ 0 & \text{pro } i = n \text{ a } j = 1 \end{cases}$

Takto získáme modifikovaný graf G_2. Nalezení hledané cesty v G je ekvivalentní nalezení orientovaného cyklu C s nejvyšším poměrem

$$\frac{\sum_{e \in C} w(e)}{\sum_{e \in C} l(e)}$$

v grafu G_2.
Za tím účelem graf G_2 ještě jednou upravíme a to na graf G_3, kde

$$w(e_{i,j}^{G_3}) = 1 - w(e_{i,j}^{G_2})$$

Nalezení cyklu C v grafu G_2 je ekvivalentní nalezení cyklu C_2 v grafu G_3 tak, že

$$\frac{\sum_{e \in C_2} w(e)}{\sum_{e \in C_2} l(e)}$$

je minimální.

Takovýto cyklus C_2 je invariantní k libovolné transformaci hran grafu G_3 ve tvaru

$$w(e) = w(e) - b \cdot l(e)$$

pro libovolné $b \in \mathbb{R}$.
Navíc existuje optimální konstanta b^* tak, že její použití vede k

$$\sum_{e \in C_2} w(e) = 0$$

Tedy celý problém byl zredukován na hledání cyklu C_2 s $\sum_{e \in C_2} w(e) = 0$. V citované práci je tento cyklus nalezen algoritmem sekvenčního vyhledávání, který pracuje následovně:

1. Nastav $b = \max_{e \in E} w(e) + 1$. Víme, že optimální $b^* < b$.

2. Transformuj váhy hran pomocí $w(e) = w(e) - b \cdot l(e)$ a detekuj negativní cyklus C. Takový cyklus bude existovat právě tehdy, když $b \neq b^*$. Pokud tedy cyklus nebyl nalezen, vrať cyklus nalezený v předchozí iteraci a ukonči výpočet.

3. Pro nalezený C spočti $\frac{\sum_{e \in C} w(e)}{\sum_{e \in C} l(e)}$ za použití netransformovaných vah hran a výsledek nastav jako b pro další iteraci. Pokračuj krokem 2.

Výsledný cyklus obsahuje právě ty vrcholy, které tvoří průměrně nejdelší cestu v původním grafu a tedy právě hledanou nejvhodnější segmentaci slova.

3.4.2 Výběr kandidátů

Jak již bylo zmíněno výše, mohli bychom při hledání segmentace začít s množinou kandidátů hustě a rovnoměrně rozprostřenou přes celý vstupní obrázek. To ale reálně vede k velkému množství volání rozpoznávacího podsystému a tedy i k velké spotřebě zdrojů. Navíc to může vést k velké míře falešných detekcí, neboť části některých písmen mohou opět vypadat jako písmena. Cílem několika dalších odstavců je tedy popsat, jak množinu kandidátů co nejvíce zmenšit a přitom nezhoršit výsledky. Zejména je třeba se vyhnout zrušení některé ze skutečných hranic, protože bez nich by výše popsaný algoritmus nemohl určit správnou segmentaci (sám už žádné kandidáty nepřidává).

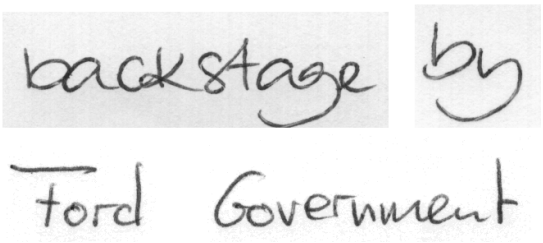
Hledání kandidátů probíhá v několika fázích a při jeho navrhování jsem použil část postupů z práce [18].

Ještě před popisem fází uveďme motivace pro jednotlivé kroky. Společnou vlastností všech rukopisů, které jsem během vytváření této práce viděl, je to, že písmena ve slovech jsou většinou propojena jen na jednom místě a to někde ve střední části slova (ve vertikálním směru). Navíc tato spojení mají téměř vždy podobu jednoduché čáry od jednoho písmene ke druhému. Už toto pozorování nám umožní hodně omezit hledané kandidáty. Když k tomu navíc přidáme základní omezení na rozměry znaků dle těch, které byly dostupné při trénování, můžeme se dostat na zlomek původního počtu kandidátů a přitom zachovat vysokou spolehlivost pokrytí všech správných možností.

Jak již napovídá předchozí odstavec, nejprve musíme najít střední část slova, ve které se nalézají spojnice mezi jednotlivými znaky. Tato oblast je právě jádro znaku, jehož hledání jsme popsali již v části 2.5.2.

Máme–li k dispozici jádro slova, můžeme přejít k hledání pixelových sloupců, ve kterých se vyskytují možné spojnice mezi znaky. Toto provedeme ve dvou krocích:

- Nejprve pro každý pixelový sloupec spočítáme, kolik přechodů z černé na bílou nebo naopak se v něm nachází. Je–li takových přechodů více než dva, pak tímto sloupcem nutně prochází více než jedna čára a tedy se (jak bylo diskutováno výše) téměř jistě nejedná o spojnici dvou znaků, ale o tělo některého ze znaků typu 'a', 'o', 'p', 'c', ... Zde vidíme důležitost první



Obrázek 8: Písmena zasahující svými částmi do okolních písmen slova. Obrázky slov pochází z [13].

fáze – kdybychom pracovali s celým slovem a ne pouze s jeho jádrem, tak by písmena jako 'T', 'P', 'y' nebo 'g' mohla, díky svým rozsáhlým částem nad či pod úrovní ostatních písmen, způsobit ignorování platných kandidátů na spojnice (viz obrázek 8). Máme tedy sloupce, ve kterých se nachází nejvýše jedna čára.

- Ve druhém kroku tyto sloupce znovu projdeme a spočteme tloušťku procházejících čar. Určíme průměr a všechny sloupce s čárou tlustší než průměr vyřadíme. Tím se zbavíme kandidátů, kteří procházejí vertikálními částmi znaků jako jsou 'i', 'k', 'b', ...

Nyní máme k dispozici posloupnost sloupců, které by mohly obsahovat místa dělení slova na znaky. Tato posloupnost ale může obsahovat (a většinou obsahuje) dlouhé podposloupnosti po sobě jdoucích sloupců. To je dané tím, že spojnice se svou šířkou často blíží šířce samotných znaků a předchozí kroky určí jako kandidáta každý sloupec obsahující pouze část nějaké spojnice. Pokud si uvědomíme, že vliv na správnost segmentace má pouze rozdělení spojnice v jejím krajním bodě nebo kdekoliv uprostřed, tak můžeme každou takovou podposloupnost redukovat na tři prvky – první, prostřední a poslední sloupec.

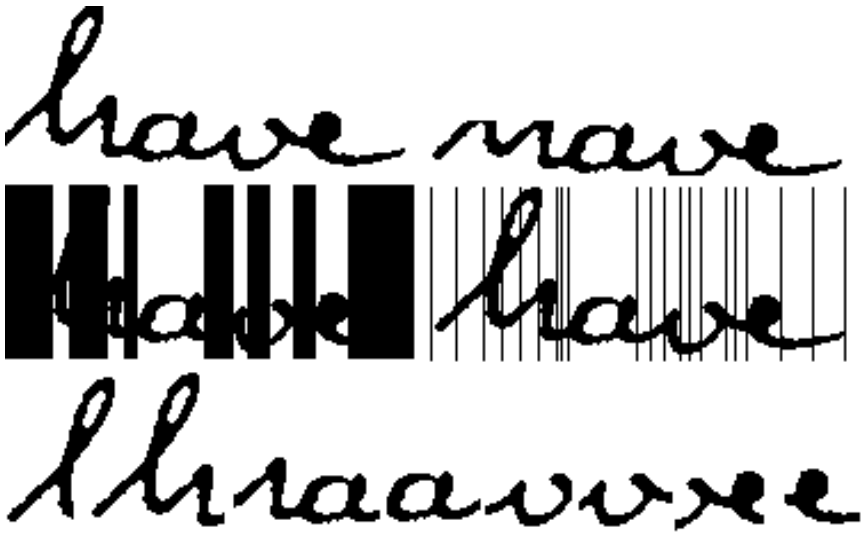
Filtrování segmentů Další úpravou, která je v práci provedena, je filtrování segmentů (hran), které díky své velikosti nemohou obsahovat známý znak. Při tvoření grafu jsou tedy vytvořeny jen hrany mezi segmenty, jejichž vzdálenost je mezi šířkou nejužšího a nejširšího známého znaku.

3.4.3 Vliv úprav na výkon

Výše popsané úpravy mají výrazný vliv na výkon, neboť řádově zmenšují velikost vstupu pro samotný segmentační algoritmus. Typicky bylo, při zpracování slova za pomocí původního algoritmu s rovnoměrným hustým pokrytím vstupního obrázku, rozpoznávacím algoritmem zkontrolováno několik set segmentů (až několik tisíc pro delší slova), kdy valná většina z nich obsahovala zjevně nesmyslná data (obrázky se šířkou 5 pixelů, obrázky několikanásobně širší než libovolný znak...). Po redukci možností pomocí popsaných postupů počet kontrolovaných segmentů řádově klesl.

Příklad průběhu výsledného algoritmu lze vidět na obrázku 9. V tomto případě jsme obdrželi 22 kandidátů a vyhodnocovali 67 segmentů. Kdybychom použili původně navrhované rovnoměrně hustě rozmístěné kandidáty s periodou pět pixelů a bez filtrace příliš velkých či malých segmentů, tak bychom měli 36 kandidátů a 666 segmentů ke kontrole. Perioda pět pixelů byla během testování přístupu s rovnoměrným rozložením vypozorována jako největší možná pro zachování správných segmentačních kandidátů pro většinu testovacích dat.

Pro zajímavost uveďme, že v případě, v původní práci zmíněného extrémního přístupu s kandidátem v každém pixelovém sloupci, bychom obdrželi 182 kandidátů a 16653 segmentů ke kontrole.



Obrázek 9: Průběh segmentace slova „have“. První řádek obsahuje vstupní slovo a jeho jádro. Druhý řádek kandidáty (svislé čáry) na segmentaci před a po závěrečné filtraci. Poslední řádek pak obsahuje některé ze segmentů vyhodnocované rozpoznávacím algoritmem.

3.4.4 Tiskací a psací text

Výše popsaný algoritmus lze použít na libovolný typ textu. Pro tiskací text s mezerami mezi písmeny to ale může být zbytečné až kontraproduktivní, neboť pro oddělení hůlkového písma existují jiné, spolehlivější a výpočetně nenáročné metody. Proto je v práci naimplementováno i dělení takovéhoto typu textu pomocí hledání spojitých komponent. Hlavní myšlenka tohoto přístupu je zřejmá: je-li ve slově psaném tiskacím písmem spojitá oblast, pak to bude právě jedno písmeno. Tato idea byla dále rozvedena i pro případy, kdy některá písmena zasahují nad či pod jiná (obvykle „ocásky“ pod písmenem y či g nebo vrchní část písmene T) a na druhou stranu jiná písmena jsou rozpadlá na více komponent (typicky E, K, H, ...). Řešením druhého případu je spojit ty komponenty, které se alespoň částečně překrývají ve vertikálním směru. To ale zase vede ke slití písmen

z prvního případu. Ve výsledku to vedlo k postupu, kdy jsou spojité komponenty hledány (a vertikální překryvy slévány) pouze v jádře slova (2.5.2). Mezi takto získanými „jádry znaků“ se už jen najdou mezery a ty se zvolí jako dělící body pro segmentaci slova.

3.4.5 Poznámka ke kurzívě

Jedním z dalších významných problémů ovlivňujících výsledky segmentace je sklon písmen při psaní kurzívou. Kvůli povaze zadání, přesněji kvůli možnosti učení vlastního rukopisu, jsem se rozhodl tento problém řešit právě učením od uživatele. Při úvodním učení má uživatel možnost ručně nastavit úhly, o které jsou jednotlivá slova skloněna. Tyto si pak aplikace zapamatuje a při rozpoznávání získaná data použije pro automatickou segmentaci. Více informací lze nalézt v 7.5.8.

4 Rozpoznávání znaků

Další fází zpracování dokumentu je určení jednotlivých znaků. Je to další z části procesu, ve které stále probíhá čilý výzkum a je publikováno mnoho prací. Pro určení hodnoty znaku tedy existuje velké množství přístupů, myšlenek a algoritmů. Jejich velká část se však skládá ze dvou základních kroků:

- Nalezení a popis specifických vlastností znaku.
- Klasifikace znaku na základě získaného popisu.

Nejinak tomu je i ve třech algoritmech, které si popíšeme v této části textu. Prvním z nich je algoritmus VHE využívající přístupu obalů oblastí znaku a jejich zaplnění. Dalším je neuronová síť neocognitron pokoušející se imitovat vidění savců. Posledním z rozpoznávacích algoritmů je pak segmentační algoritmus, který znaky dále rozkládá na jednotlivé tahy.

4.1 VHE

Prvním z algoritmů pro rozpoznávání znaků implementovaným v práci je tedy algoritmus VHE. Jeho původní verze vznikla v rámci bakalářské práce autora [19]. Aktuální verze staví na původní myšlence detekce důležitých vlastností, rozvíjí však zpracování takto získaných dat o znaku. Při analýze vstupu využívá některé z konceptů teorie fuzzy množin a fuzzy logiky. Následujících pár stránek tedy počítá s tím, že se čtenář orientuje v základních pojmech zmíněných oborů. Pokud tomu tak není, doporučuji nahlédnout například do [20].

Jméno algoritmu bylo odvozeno od jeho principu používání vertikálních a horizontálních obalů (Vertical and Horizontal Envelopes).

4.1.1 Základní idea

Myšlenka algoritmu je prostá – pro určení znaku je důležitý nejprve jeho celkový tvar. Až pokud tento není jednoznačný, zaměříme se na jeho menší části.

Získání vlastností znaku
Způsob zachycení vlastností znaku je motivován poznatky, které jsem vypozoroval sám na sobě při identifikaci náhodně vybraného znaku – tedy tím, že nejdůležitější je pro mě celkový tvar znaku, až poté jednotlivé části. Příkladem mohou být znaky M a I, které jsou na první pohled odlišné od všech ostatních. Na druhém konci spektra pak jsou například znaky D a O, C a G nebo N a H, u kterých je potřeba věnovat větší pozornost i jednotlivým detailům.

Potřebujeme tedy nejprve kvantifikovat základní tvar znaku, a pokud tento nebude jedinečný, tak přejít k dalším detailům, dokud jej nebude možné odlišit od ostatních znaků.

Analýza znaku
Analýza znaku se skládá ze dvou kroků:

1. Zachycení tvaru znaku obalem.
2. Kvantifikace vlastností obalu.
3. Odlišení dalších detailů, je–li to nutné.

Zachycení tvaru znaku
Problém zachycení základního tvaru znaku lze vyřešit pomocí konstrukce obalu znaku. Přesněji zaplnění celé části, která je znakem nějakým způsobem obalena, pomocí černých pixelů. Tím se zbavíme všech nedůležitých informací – těch, které se netýkají celkového tvaru.

Musíme ale zajistit, abychom s tímto obalem neodstranili i část pro tvar důležitých informací. Tedy použití například konvexního obalu vhodné není. U většiny znaků by to vedlo k nechtěné ztrátě informace (např. M, v, x, ...). Ve výsledném postupu jsou použity čtyři různé obaly, kde každý z nich zachycuje detaily tvaru pouze z jednoho ze čtyř základních směrů – shora, zdola, zleva nebo zprava.

Klasifikace vlastností obalů
Kdybychom měli jistotu, že všechny výskyty daného znaku budou totožné (ideální případ pro tištěné znaky), stačilo by si jen pamatovat, které pixely byly černé (tedy patřily znaku). Pro ne úplně stejné (přesto stále podobné) exempláře téhož znaku bychom pak rádi zavedli nějakou toleranci. Můžeme tedy místo jednotlivých pixelů kontrolovat obsah obdélníků o rozměrech $n \times m$ pixelů.

Právě takto funguje kvantifikace vlastností obalu. Rozdělíme obaly na oblasti zájmu, v každé z nich určíme nakolik je plná a tím, ve výsledné kombinaci hodnot pro všechny čtyři směry, zachytíme i typ jejího obsahu. Pro zajištění dostatečné tolerance vůči odchylce v ručně psaném textu jsou tyto hodnoty navíc fuzzifkovány.

Odlišení dalších detailů
Předchozí fáze jsou schopny dodat dobrou představu o celkové podobě mnoha znaků. Existují ale i skupiny znaků, kterým takovýto postup přiřadí velice podobnou reprezentaci. Například již výše zmíněné B a D.

Tento problém algoritmus adresuje opakováním klasifikačního procesu i pro některé podoblasti znaku. Například jeho střed či dolní polovinu. Tím zachytíme i další detaily, které se v původních obalech ztratily a pro správnou klasifikaci mohou být stěžejní.

4.1.2 Konstrukce obalů

Algoritmus na vstupu očekává černobílý obrázek (popředí je černé) s jedním znakem (viz obrázek 10 vlevo). Nejprve pro obrázek vytvoříme výše popsané

obaly. Obaly tedy budou čtyři – pohled zdola, shora, zleva a zprava. Jejich tvorbu popíšeme jen pro pohled shora, ostatní obaly tvoříme analogicky.

Pohled shora Pohled shora vytvoříme průchodem jednotlivých pixelových sloupců od horního okraje ke spodnímu. V každém sloupci necháváme bíle pixely, dokud nenarazíme na první černý. Od toho okamžiku budou všechny další pixely až po spodní okraj černé. Lze si to představit jako nalití vody na vrchní okraj obrázku se znakem z hydrofobního materiálu. Obrázek se pak opláchne (bude bílý) všude tam, kudy proteče voda.

Po vytvoření takovéhoto obalu je vhodné jej ještě upravit, uzavřít případné malé mezery vzniklé nedotaženou čarou a podobně (viz obrázek 10).

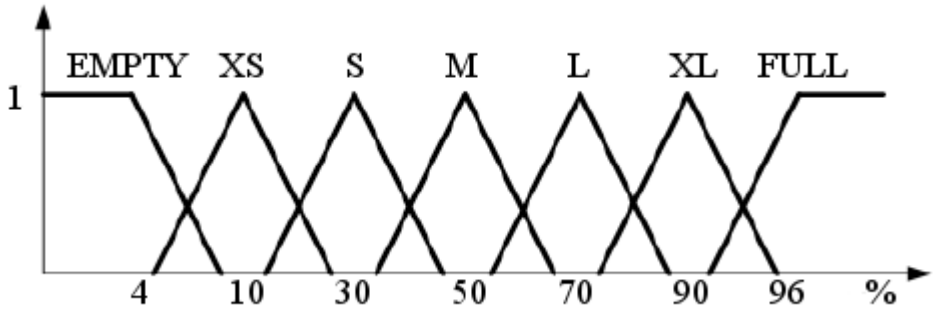
Obrázek 10: Vstupní znak A s chybějícími pixely, přenos chyby do horního obalu a jeho oprava.

Jak již bylo zmíněno, ostatní obaly tvoříme stejným způsobem, jen z ostatních tří směrů. Při návrhu bylo uvažováno i o dalších směrech pro obaly, ty ale dodaly další informaci jen v několika málo specifických případech, jako jsou znaky C a G při pohledu z pravého horního rohu a dostatečné mezeře mezi spodní a horní částí. Tento přístup byl tedy nakonec nahrazen analýzou podoblastí popsanou výše. Ta dodatečných informací zachová více.

Nyní, když máme obaly, můžeme přejít k jejich analýze a získávání výsledných reprezentací znaků.

4.1.3 Analýza obalů

Předpokládejme, že máme k dispozici obal nějakého znaku nebo jeho oblasti a víme, vlastnosti kterých částí obalů nás zajímají. Pak výsledná reprezentace obalu je kolekce dvojic (oblast, jméno fuzzy množiny), kde oblast definuje své umístění a rozměry v obalu a jméno fuzzy množiny identifikuje fuzzy množinu, ve které má nejvyšší stupeň příslušnosti procentuální vyjádření zaplnění této oblasti. Použité fuzzy množiny jsou na obrázku 11.



Obrázek 11: Fuzzy množiny použité v algoritmu VHE při analýze obalů k fuzzifikaci procentuálního zaplnění jednotlivých oblastí.

V této fázi už k možnosti porovnávání znaků potřebujeme jen určit, jakým způsobem porovnat reprezentace jednotlivých obalů. Pro úplnost podotkněme, že se vždy budou porovnávat jen sobě odpovídající obaly. Tedy stejný typ obalu (levý, pravý spodní, horní) stejné oblasti (umístění a velikost) ve dvou různých znacích.

Nejprve tedy definujme, jak porovnávat nejmenší dostupné informace, zaplnění oblastí obalu. V praxi se ukázalo, že je prospěšné přidat další toleranci i na této úrovni. Tedy umožnit nenulovou podobnost i pro některé páry hodnot daných jmény použitých fuzzy množin. Stupně podobnosti použité v doprovodné implementaci lze najít v tabulce 4.1.3.

Máme–li určeny podobnosti všech odpovídajících si oblastí v daných obalech téže části dvou znaků, můžeme přistoupit k spočítání podobnosti těchto dvou obalů.

Pro pár stejných obalů dané části znaků získáme podobnost jako průměrnou podobnost odpovídajících si párů oblastí zájmu. Poznamenejme, že průměr nebyla jediná zvažovaná možnost. Vyzkoušeno bylo několik dalších včetně např. Łukasiewizcovy či produktové T–normy. Nicméně v prvním případě docházelo

	EMPTY	XS	S	M	L	XL	FULL
EMPTY	1	0.8	0.3	0	0	0	0
XS	0.8	1	0.8	0.3	0	0	0
S	0	0.8	1	0.8	0.3	0	0
M	0	0.3	0.8	1	0.8	0.3	0
L	0	0	0.3	0.8	1	0.8	0.3
XL	0	0	0	0.3	0.8	1	0.8
FULL	0	0	0	0	0.3	0.8	1

Tabulka 1: Stupně podobnosti hodnot oblastí. Hodnoty byly určeny experimentálně, zachycují ale intuitivní představu o podobné velikosti.

k příliš striktní kumulaci chyby (ve smyslu odlišnosti), což se pro ručně psaný text ukázalo nevhodné. Druhý případ na tom byl o poznání lépe, ale i ten za průměrem, co do správné klasifikace, zaostával.

Analýza části znaku Máme–li analyzovány všechny obaly dané části znaku, můžeme přistoupit ke spočtení výsledné podobnosti pro celou podoblast, tedy některý z detailů znaku (viz 4.1.1). I zde bylo vyzkoušeno více možností, například zdůraznění pravého obalu (tj. větší důraz na pravou stranu oblasti/znaku) pro lepší odlišení dvojic znaků jako E a C, nebo použití minima pro striktnější či naopak maxima pro volnější klasifikaci. Opět se ale jako nejlepší volba ukázal aritmetický průměr hodnot.

4.1.4 Porovnávání znaků

Nyní už známe všechny potřebné údaje pro samotné porovnání celých znaků. To probíhá v šesti krocích. Každý krok se odlišuje tím, jak jemné detaily v jaké oblasti hledá, přičemž se, v duchu diskuze z úvodu, postupuje od nejhrubší klasifikace (celkový tvar) až po detaily v částech znaku. Výsledné porovnání znaků tedy probíhá takto:

1. První krok analyzuje celé obálky celého znaku a jejich poloviny ve vertikálním i horizontálním směru.
2. Druhý krok analyzuje vertikální a horizontální třetiny obálek celého znaku.
3. Třetí krok analyzuje vertikální, horizontální a mřížkové čtvrtiny obálek celého znaku.
4. Čtvrtý krok analyzuje mřížkové pětadvacetiny obálek celého znaku.
5. Páty krok provádí první, druhý a třetí krok na obálkách mřížkových čtvrtin znaku.
6. Šestý krok nakonec provádí první, druhý a třetí krok na obálkách čtvrtiny znaku okolo jeho středu.

Začíná se prvním krokem. Ten spočte podobnost ke všem naučeným znakům, z výsledných podobností vybere ty nejlepší a předá je kroku druhému, který opakuje stejný postup na těchto předvybraných kandidátech. Takto se postupuje dále, dokud některý z kroků nenajde výsledek (tj. zůstane jediný znak, nebo několik exemplářů téhož znaku). Pokud proces projde všemi kroky a stále zůstane více vhodných kandidátů, tak poslední krok vybere toho nejpodobnějšího dle svého výpočtu.

4.2 Neocognitron

Druhým algoritmem, který si zde popíšeme je neuronová síť neocognitron. Pro plné pochopení dalších několika stránek je vhodné mít alespoň základní vhled do problematiky neuronových sítí a lineární algebry. Pro osvěžení těchto znalostí bych čtenáři doporučil [21] a [22].

Při samotném popisu a implementaci neocognitronu jsem pak čerpal zejména z prací [23] a [24], dále pak z [25] a [26].

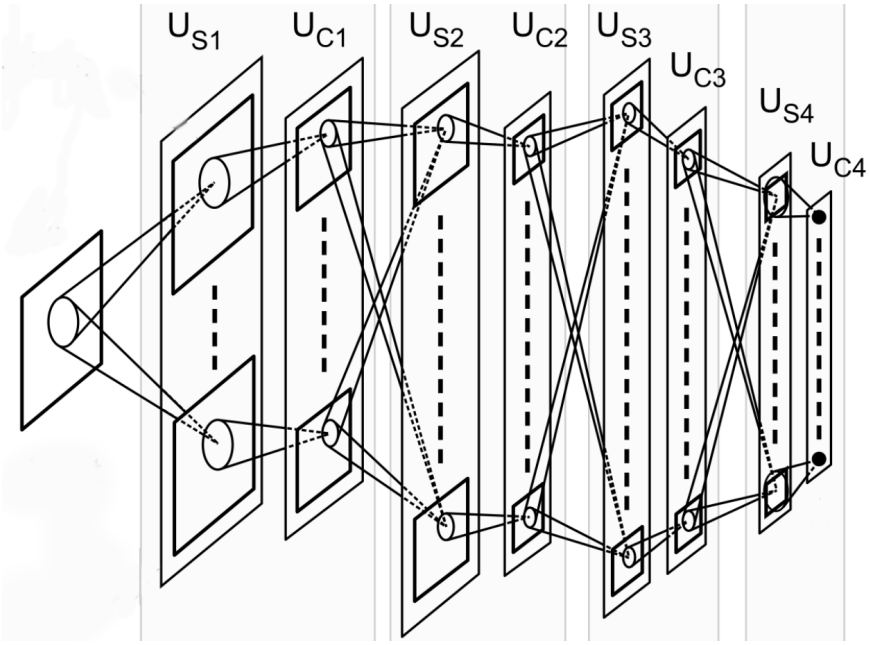
Neocognitron je model neuronové sítě navržen Kunihiko Fukushimou v roce 1980 [27]. Jde o hierarchickou více vrstvou síť založenou na zjednodušeném modelu zraku u savců, se schopnosti robustně rozpoznávat vzory v obraze. Mezi nejčastější využití modelu patří OCR u ručně psaných čísel, může však být využit pro rozpoznávání i jiných objektů v dvourozměrném obraze. Jeho největší výhodou je právě robustnost, díky které je schopen identifikovat nejen naučené objekty, ale i jejich pootočené, posunuté, zmenšené či jinak transformované verze. Další výhodou je také schopnost samoorganizace sítě při jejím vytváření. Na druhou stranu při použití Neocognitronu hrozí, že síť, kterou získáme, bude mít v každé z vrstev mnoho neuronů a mnoho spojení, což může vést ke zvýšené náročnosti na zdroje a čas. Model také má mnoho parametrů, které mohou silně ovlinit výsledek.

4.2.1 Struktura Neocognitronu

Neocognitron je víceúrovňová síť se dvěma (hlavními) typy vrstev, S–vrstvy a C–vrstvy (viz dále). Tyto vrstvy jsou střídavě a hierarchicky uspořádány. Přesněji každá úroveň obsahuje po jedné z těchto vrstev, nejprve S–vrstvu (spolu s doplňkovou V–vrstvou) a poté C–vrstvu. Dále použité značení je U_{Ck} pro k–tou C–vrstvu a U_{Sk} pro k–tou S–vrstvu. Mezi každým párem sousedících vrstev jsou retinoopticky uspořádána spojení. Retinooptické uspořádání znamená, že každá buňka dané vrstvy dostává vstupní signály od buněk předchozí vrstvy, které se nachází pouze v omezené oblasti, tedy od buněk pracujících s omezenou části vstupu (viz obrázek 12). Právě tato vlastnost je inspirována zrakovou dráhou u savců.

První (vstupní) úroveň obsahuje pouze vstupní vrstvu U_0, která je v podstatě dvourozměrným polem buněk odpovídajícím receptorům sítnice (pixelům vstupu). Výstup této vrstvy se stává vstupem první skryté úrovně (přesněji vrstvy U_{S1}), která reaguje na lokální charakteristiky objektu na vstupu, tj. křížení čar, konce čar, atp. Každá další vrstva poté reaguje na globálnější charakteristiky (formace čar, vzájemná pozice bodů, ...). Poslední (výstupní) úroveň poté reaguje na všechny informace o všech charakteristikách celého vstupu.

Neurony pozdějších vrstev tedy pokrývají větší oblast vstupu (viz obrázek 12) a tedy se jejich hustota v pozdějších vrstvách snižuje. Vrstvy jsou navíc dále rozděleny na roviny (v originále plane). Každá z těchto rovin reaguje na jinou charakteristiku a skládá se z několika neuronů reagujících na odpovídající charakteristiku na různých pozicích v předchozí vrstvě (a tedy i ve vstupu).



Obrázek 12: Typická architektura neocognitronu. Alternující S a C–vrstvy obsahují roviny neuronů, kdy každá S–vrstva přebírá vstupy od všech rovin předchozí vrstvy a každá C–vrstva přebírá vstupy vždy od jedné roviny z předchozí S–vrstvy. Upraveno z [25].

Neurony v nejvyšší S–vrstvě jsou učeny pomocí předem klasifikovaných tréninkových dat. Pro každou z možných tříd je v nejvyšší vrstvě většinou generováno více rovin – pro každou z detekovaných transformací. Nejvyšší C–vrstva nakonec vybere třídu s největší aktivací v nejvyšší S–vrstvě jako výsledek klasifikace.

Neurony všech S–vrstev kromě poslední jsou trénovány bez učitele tak, aby pro podobné vstupy generovaly podobný výstup, a tím postupně stíraly rozdíly mezi podobnými vstupy pro poslední vrstvu.

4.2.2 S–neurony a V–neurony

S–neurony (neurony tvořící S–vrstvy) neboli jednoduché (S z anglického simple) neurony fungují jako detektory charakteristik ve vstupu. S–neurony ve vyšších vrstvách detekují globálnější charakteristiky (např. části znaku) než S–neurony v nižších vrstvách (body, čáry, ...). Každý S–neuron je spojen s V–neuronem, který má stejné zdroje vstupních synapsí a jehož výstup směřuje právě do daného S–neuronu a přenáší průměrnou intenzitu vstupů přes inhibiční synapsi. Průměr zde může být počítán více statistikami, dle [23] je nejvhodnější použít vážený kvadratický průměr:

$$v = \sqrt{\sum_n c_n x_n^2} = ||\overline{x}||$$

kde c_n jsou váhy vstupních synapsí a x_n souřadnice tréninkového/vstupního vektoru.

Váhy vstupních hran S–neuronů mohou být měněny při tréninku – tímto je určena extrahovaná vlastnost pro daný S–neuron. Po natrénování každý S–neuron reaguje výhradně na danou charakteristiku v jeho oblasti působnosti.

4.2.3 Výpočet S–neuronů

Nechť $\overline{X} = (X_1, \cdots, X_n)$ je tréninkový vektor naučený S–neuronem (*referenční vektor*). Váha vstupní synapse a_n je pak dána jako

$$a_n = c_n \cdot X_n / ||\overline{X}|| \tag{1}$$

Dále nechť $\overline{x}$ je vstupní vektor (*stimulační vektor*), $\varphi(x) = \max(x, 0)$ a Θ práh (tj váha inhibiční hrany z V–neuronu; hodnota určující toleranci ke změnám). Podobnost stimulačního a referenčního vektoru je pak dána jako

$$s = \frac{(\overline{X}, \overline{x})}{||\overline{X}|| \cdot ||\overline{x}||}$$

Pro S–neuron existuje více možných aktivačních funkcí, nicméně nejnovější z výše uvedených zdrojů doporučují použít tuto:

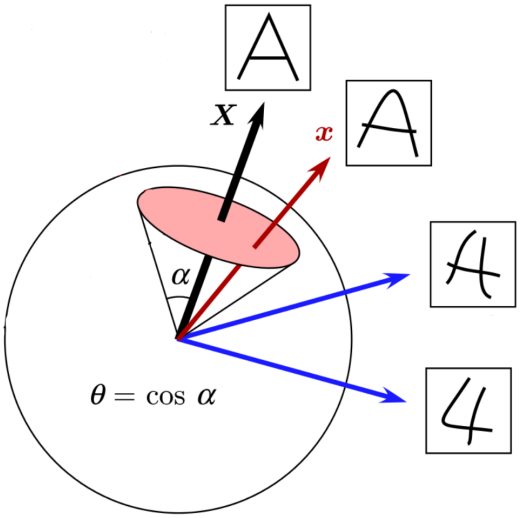
$$s_{out} = ||\overline{x}|| \cdot \frac{\varphi[s - \Theta]}{1 - \Theta}$$

S–neuronu tedy zahoří pouze, pokud je podobnost s mezi $\overline{x}$ a $\overline{X}$ větší než práh Θ. Vektory pro které platí $s > \Theta$ jsou nazývány oblast tolerance S–neuronu (viz obrázek 13). To znamená, že můžeme ovlivnit toleranci/reakci S–neuronu na vstup změnou prahové hodnoty Θ. Vyšší Θ zajistí menší toleranci k transformacím vstupu a nižší naopak. Z pohledu rozpoznávání vzorů v obraze to znamená, že snížení prahu zvýší toleranci k deformacím.

Dva různé prahy pro S–neurony
Popisujeme–li funkci prahové hodnoty u S–neuronu, zmiňme použití různý prahových hodnot při učení a klasifikaci. Přesněji vyšší práh při tréninku a nižší při klasifikaci. Použití nižšího prahu při klasifikaci nám umožní vyšší toleranci k deformacím při použití méně neuronů a zároveň pomůže zabránit příliš úzce specializovaným neuronům reagujícím pouze na přesnou verzi naučeného vstupu.

4.2.4 Trénink S–neuronů

Během tréninku jsou vzorová data předávána síti jeden vstup za druhým. Aktivace vrstvy U_{Cl-1} je použitá jako tréninkový vstup pro U_{Sl}. Při tréninku každé vrstvy předpokládáme, že trénink předchozích vrstev byl již dokončen. Tedy trénink probíhá po vrstvách.



Obrázek 13: Oblast tolerance S–neuronu a grafický význam prahové hodnoty Θ. Bílý kruh reprezentuje prostor možných vstupů, menší červený kruh pak ukazuje, které vektory padnou do tolerance S–neuronu s prahem Θ. Naučený vektor je označen černě, tolerované vektory červeně a ignorované vektory modře. U každého z vektoru je příklad generujícího vstupu. Modifikováno z [23].

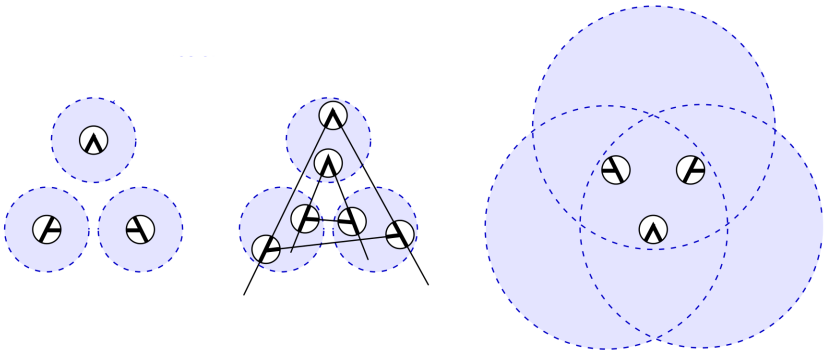
Váhy vstupních synapsí do S–neuronů nižších úrovní (úrovně 1 až $n-1$) jsou stanoveny učením bez učitele. Učení poslední úrovně je popsáno v další části textu. Je více možnosti, jak učit S–neurony nižších úrovní. Rozdíl mezi nimi je zejména v upravování vah vstupních hran do S–neuronu po jeho vytvoření. Dle [25] je nejnovějším a nyní nejpoužívanějším pravidlo „přidat pokud mlčí“ (orig. add–if–silent). Toto pravidlo je použito i v doprovodné aplikaci, popíšeme si jej tedy blíže.

Přidat pokud mlčí

Pokud žádný z neuronů vrstvy U_{Sl} při tréninku nového referenčního vektoru nezahoří, tak je vytvořen nový S–neuron v U_{Sl}. Váhy jeho vstupních hran jsou nastaveny tak, aby reagoval na tento referenční vektor, a zároveň tak, aby odpovídaly síle výstupu z předchozí vrstvy (tedy dle vztahu (1)). Po přidání neuronu do vrstvy už jeho váhy nebudou nadále měněny.

Ve skutečnosti přidání nového neuronu znamená přidání celé roviny neuronů – pro různé pozice právě naučené charakteristiky (referenčního vektoru). Všechny neurony vytvořené v této nové rovině mají stejné váhy vstupních synapsí jako první generovaný neuron – ten se stává jádrem (orig. seed cell) nové roviny.

Tento proces se opakuje, dokud každý vstup nevyvolá v dané vrstvě nějakou reakci, poté se přejde k další vrstvě. Použití této myšlenky nám zajišťuje, že celý prostor možných vstupů (specifikovaný tréninkovou množinou) bude v dané vrstvě vyvolávat odpovídající reakci.



Obrázek 14: Tolerance posunu pomocí C–neuronů. Vlevo vidíme části naučeného vzoru, uprostřed správně detekované části deformovaného znaku a vpravo chybu klasifikace způsobenou příliš velkou tolerancí. Modifikováno z [23].

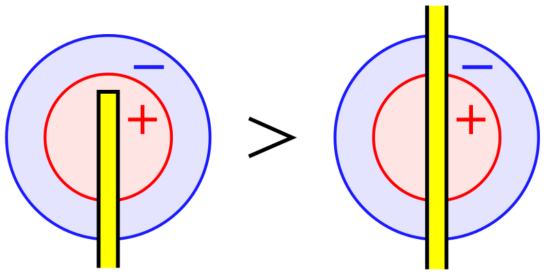
4.2.5 C–neurony

C–neurony neboli komplexní neurony (z anglického complex) umožňují toleranci posunu dané charakteristiky. Každá rovina C–neuronů dostává stimuly od roviny z jedné předchozí S–vrstvy, která detekuje jeden typ charakteristiky v nějaké oblasti vstupu. Toto je jeden z velkých rozdílů oproti S–neuronům, které vstupy dostávají od dané oblasti neuronů ze všech rovin předchozí vrstvy.

Váhy vstupních hran k C–neuronu jsou zafixovány a nemění se během tréninku. C–neuron zahoří, pouze když zahoří alespoň jeden z jeho vstupních S–neuronů. Úkolem C–neuronu je zprůměrovat vstupy z S–neuronů předchozí vrstvy. K tomu mohou být použity různé statistiky: (vážený) aritmetický průměr, (vážený) kvadratický průměr nebo maximum. Toto v globálním měřítku způsobuje toleranci sítě k deformacím. Každá C–vrstva totiž umožní malou změnu v každé z dříve získaných charakteristik a zároveň tuto toleranci propaguje do další úrovně získávání charakteristik S–vrstvou. Na proces se lze dívat ze dvou úhlů pohledu:

Snížení citlivosti k posunu
První možností je interpretovat funkci C–neuronu jako absorbování části poziční odchylky jednotlivých charakteristik každou z C–vrstev. Výsledkem pak je, že každá z vyšších S–vrstev robustněji detekuje danou charakteristiku – toleruje větší deformace. Na druhou stranu nesmí být tolerance posunu příliš velká, neboť při velké toleranci u nízkoúrovňových charakteristik může celá síť vykazovat chybné výsledky (obrázek 14). Je důležité, že tento proces probíhá v každé C–vrstvě (tedy jednou v každé úrovni). Díky tomu mají vyšší vrstvy možnost tolerovat větší změny jednotlivých charakteristik.

Rozmazání charakteristik C–neuronem
Na napojení S–neuronu do C–neuronu lze také pohlížet jako na operaci rozma-



Obrázek 15: Inhibiční okolí kolem synapse C–neuronu. Neuron s váhami kladnými uprostřed a zápornými na okrajích jím pokryté oblasti bude reagovat silněji na význačný bod (vlevo), než na běžný bod (vpravo). Modifikováno z [23].

zání části vstupu. Zde využijeme schopnost S–neuronu změřit podobnost mezi dvěma vstupními vektory (pomocí jeho aktivační funkce). K tomu je použit skalární součin, a tedy je výsledek závislý na „přesahu“ těchto vektorů. Když dané charakteristiky rozmažeme, tak se podobnost jejich vektorů zvýší a zvýší se tedy i podobnost vypočítaná S–neuronem. Důležité je, že rozmazání C–vrstvou je provedeno až po extrakci charakteristik S–vrstvou, což umožní přesnou identifikaci lokálních charakteristik a zároveň vyšší toleranci podobnosti těchto charakteristik se vzorem. Díky rozmazání je také vyhlazen případný šum ve vstupu.

Inhibiční okolí
Další z novějších úprav popsaných autorem neocognitronu, kterou používají jeho modernější verze, je koncept inhibičního okolí kolem vstupní synapse C–neuronů (obrázek 15). Nosnou myšlenkou je zde vynucení silnější reakce na důležité charakteristiky. Koncové body čar způsobují silnější reakci než vnitřní body, a tedy tyto (pro klasifikaci zjevně důležitější body) ovlivní výsledek klasifikace více než ostatní.

4.2.6 Nejvyšší S–vrstva

S–neurony v nejvyšší S–vrstvě (U_{Sl}) jsou trénovány s učitelem pomocí předem klasifikované tréninkové množiny. Pokud nějaký vstup způsobí aktivaci v předchozí C–vrstvě, ale žádnou reakci v této S–vrstvě, tak je vygenerována nová rovina, která přebírá vstupní vektor z U_{Cl-1} jako svůj referenční vektor a třídu vstupního obrázku jako výsledek. Většinou je vygenerováno více než jeden neuron pro každou možnou třídu.

Během klasifikace je poté výstup U_{Sl} analyzován v U_{Cl} a třída s nejsilnější aktivací zvolena za výsledek.

Je více možností jak klasifikovat výstup z U_{Sl}. Tréninková metoda musí být vybrána dle klasifikační metody. Dle citovaných prací je jedna z nejčastějších použitých metod pro nejvyšší vrstvu metoda „vítěz bere vše s učitelem“ (orig. supervised winner–take–all).

Vítěz bere vše s učitelem

Během tréninku jsou poslední vrstvě předkládány jeden za druhým výstupy předchozí vrstvy. Pokaždé, když vrstva nový vstup klasifikuje špatně, nebo pro něj nezahoří žádný z jejích neuronů, tak je vygenerována nová S–rovina, která se naučí právě tento vektor a zapamatuje si jeho třídu.

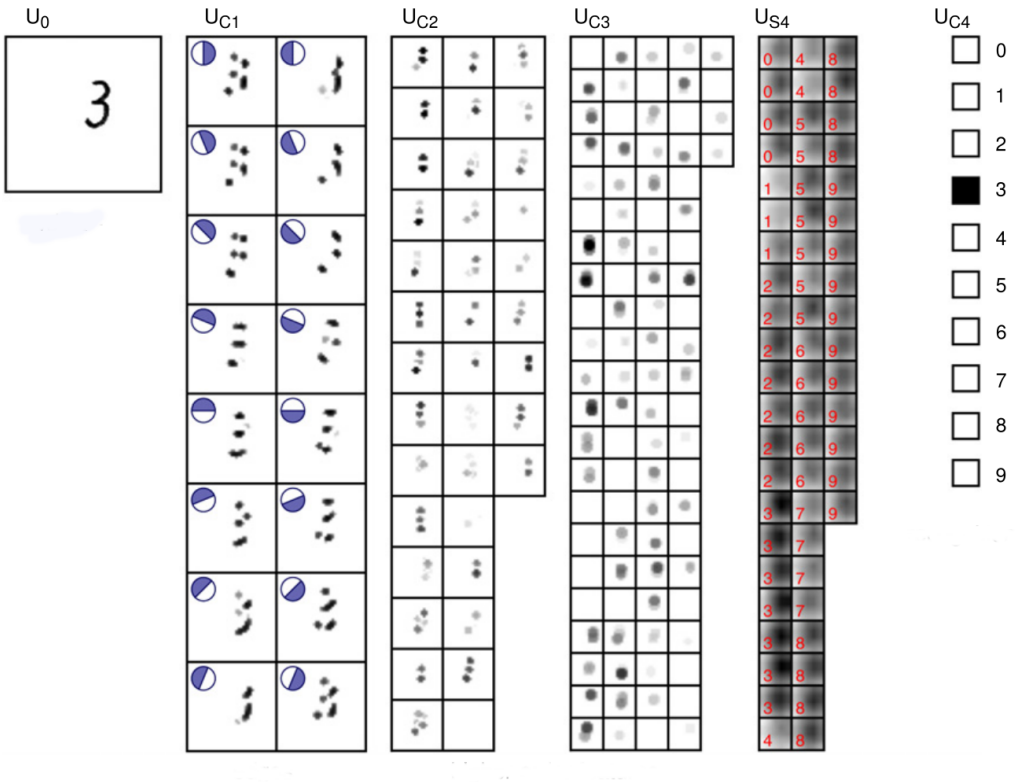
Během klasifikace je pak jako výsledná třída vybrána ta, která odpovídá rovině s nejsilněji aktivovaným neuronem. Odtud pochází název vítěz bere vše.

Dalším rozdílem oproti nižším vrstvám je nastavení obou prahů Θ s–neuronů (učícího a klasifikačního) na nulu. Důsledkem je změna aktivační funkce na

$$s_{out} = ||\overline{x}|| \cdot \frac{\varphi[s-0]}{1-0} = ||\overline{x}|| \cdot s,$$

tedy na podobnost vstupního a naučeného vektoru proporční k síle vstupu.

Celý proces rozpoznávání v poslední vrstvě si tedy lze představit jako hledání tréninkového vektoru nejpodobnějšího dodanému stimulu s přihlédnutím k jeho síle. Grafické zobrazení je na obrázku 16. Pro vstup „3“ jsou zde postupně znázorněny výsledky výpočtu čtyřúrovňového neocognitronu. U jednotlivých úrovní vidíme výstup jejich C–vrstev, tedy rozmazané charakteristiky detekované předchozí S–vrstvou. Pro první úroveň toto odpovídá různě orientovaným čarám. V dalších úrovních už interpretace není tak snadná, nicméně každá z rovin detekuje některou z kombinací dříve nalezených vlastností. Čím tmavší je barva v dané oblasti roviny, tím silněji zahoří okolní neurony. V poslední úrovni pak vidíme, že nejsilněji reagovala rovina s třídou „3“, ta je tedy vybrána jako výsledek.



Obrázek 16: Grafické znázornění výpočtu neocognitronu. Čím tmavší barva, tím silněji reagují neurony v dané oblasti. Modifikováno z [23]

4.2.7 Popis implementace

Při implementaci necoongitronu jsem se snažil, co nejvíce přiblížit postupům popsaným ve výše zmíněných pracích. Nicméně verzí neocogitronu již existuje celá řada, neboť jeho autor jej kontinuálně vyvíjí již od jeho prvního návrhu v roce 1980. Učící pravidla a parametry (počty úrovní, prahy, aktivační funkce, rozměry vrstev a počty jejich napojení do předchozích vrstev, ...) jsem nikde nenašel v ucelené podobě. Příkladem za všechny může být speciální učení pro první vrstvu, které je použito v mnoha verzích neocognitronu, ale v článcích mu je zpravidla věnována jen jedna věta v úvodní části. Tyto detaily výsledné implementace jsou tedy spíše kombinací náznaků z různých zdrojů a výsledku experimentů. Pro detailnější popis situace vizte diskuzi v části o výsledcích (6.3.5).

4.3 Segmentační algoritmus

Třetím a posledním algoritmem pro rozpoznávání písmen implementovaným v práci je algoritmus pracující na principu další segmentace znaků. Tento pochází z práce [28]. Implementace pak vychází z implementace v mé bakalářské práci. Pro získání charakteristických vlastností segmentů používá základní principy teorie

fuzzy množin. Chce–li si čtenář tyto koncepty nejprve připomenout, může k tomu použít například [20].

Stejně jako oba výše zmíněné algoritmy i tento první hledá charakteristiky znaků, které využívá k následnému porovnávání. Hledání charakteristik probíhá ve dvou krocích:

1. Hledání segmentů znaku.

2. Kvantifikace jejich vlastností.

4.3.1 Hledání segmentů znaku

Segment znaku je libovolná jeho část od ostatních částí oddělena význačnými body. Algoritmus rozpoznává pět skupin význačných bodů: osamocený bod, koncový bod, křižovatka tvaru X, křižovatka tvaru T a změna směru.

Pro nalezení těchto bodů potřebujeme znát kostru znaku, tu tedy dodáme za pomocí externího algoritmu (2.5.1). Při znalosti kostry lze význačné body získat dle počtu černých sousedů každého pixelu. Každý bod, který má jiný počet černých sousedů než dva, je význačným bodem (bod bez černých sousedů je osamocený, bod s jedním černý sousedem je koncový, atd.).

Známe–li všechny význačné body, můžeme přistoupit k hledání jednotlivých segmentů. Zajímá nás šest typů segmentů: bod, vertikální čára, horizontální čára, zleva doprava stoupající čára, zprava doleva stoupající čára a cyklický segment.

Extrakce segmentů

Nejprve najdeme a ze vstupní kostry vyjmeme všechny osamocené body. Pokračujeme hledáním necyklických segmentů, tedy vždy najdeme koncový bod a od něj přecházíme ze souseda na souseda, dokud nedojdeme do jiného význačného bodu. Navštívené pixely vyjmeme a uchováme pro pozdější zpracování. Tento postup opakujeme, dokud v kostře zbývá nějaký koncový bod.

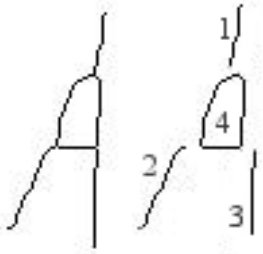
Jakmile v kostře žádné další koncové body nejsou, tak přejdeme k hledání cyklických segmentů. Začneme libovolným ze zbylých černých pixelů a opět přecházíme ze souseda na souseda, dokud se nedostaneme zpět do původního pixelu. Tím jsem našli cyklický segment. Jeho body z kostry vyjmeme.

Odstraněním cyklického segmentu se mohl otevřít nějaký další cyklický segment na něj napojený, a tedy objevit nový koncový bod. Proto se musíme vrátit zpět ke hledání necyklických segmentů a opakovat jej, dokud v kostře existují nějaké koncové body.

Tyto dva kroky střídáme, dokud je v obrázku nějaký bod kostry.

Zpracování nalezených segmentů

V okamžiku, kdy jsme z kostry získali všechny segmenty, musíme určit jejich typy a vlastnosti. Určení typu segmentu je přímočaré: je–li délka segmentu rovna jedné, pak jde o osamocený bod; pokud segment začíná a končí na stejném místě a není to bod, pak jde o cyklický segment; pokud segment začíná a končí v blízkých



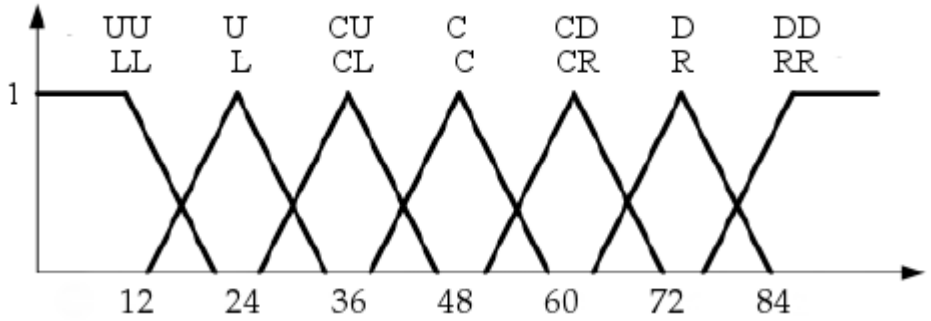
Obrázek 17: Kostra znaku A a jeho segmentace. Segmenty v pravé části jsou určeny jako: 1 – vertikální, 2 – zleva stoupající, 3 – vertikální a 4 – cyklický.

pixelových sloupcích a není to bod ani cyklus, pak je to vertikální čára; podobně pro horizontální čáru a řádky; nejedná–li se o žádný z předchozích segmentů, tak musí jít o zleva stoupající nebo zprava stoupající segment – to určíme dle souřadnic jeho konců. Výslednou segmentaci pro znak A včetně typů jeho segmentů ukazuje obrázek 17.

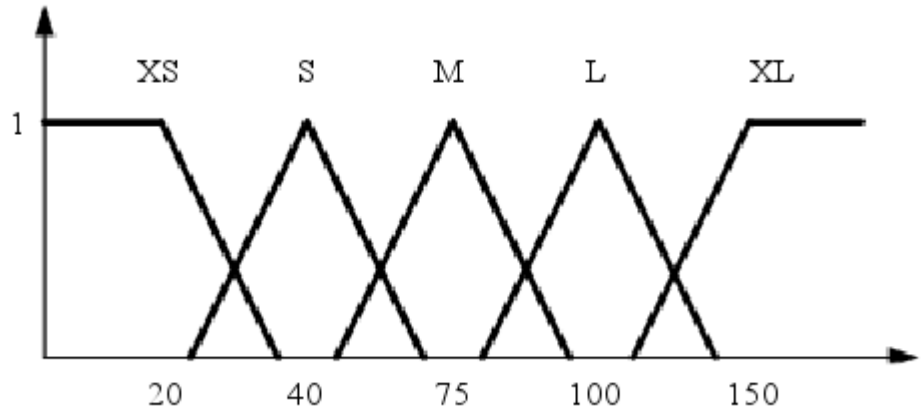
4.3.2 Kvantifikace vlastností segmentů

Nyní, když máme dostupné všechny segmenty vstupního znaku, musíme zvolit množinu vlastností, kterou jsou popsány. Původní zdroj používá délku segmentu a jeho pozice ve vertikálním a horizontálním směru. V implementaci je do množiny charakteristických vlastností přidán i typ segmentu, neboť občas docházelo ke spárování různých typů segmentů se stejnou délkou a pozicí, a tedy výrazné změně výsledku. Dále jsou přidány pozice koncových bodů segmentů (mají–li je).

Všechny vlastnosti kromě typu segmentu jsou následně fuzzifikovány pomocí fuzzy množin z obrázků 18 a 19. Výsledná hodnota dané vlastnosti je jméno fuzzy množiny, do níž náleží v maximálním stupni.



Obrázek 18: Fuzzy množiny pro vertikální/horizontální pozici charakteristik segmentů. Na horizontální ose je procentuální vyjádření pozice daného elementu vzhledem k výšce/šířce vstupního obrázku. Na vertikální ose je pak stupeň náležení dané hodnoty do jednotlivých fuzzy množin. Významy jmen jednotlivých fuzzy množin jsou odvozeny z anglických slov Left, Right, Center, Up a Down (vlevo, vpravo, uprostřed, vzhůru a dolů) a vyjadřují míru posunu charakteristiky od středu vstupního obrázku.



Obrázek 19: Fuzzy množiny pro délku segmentů. Na horizontální ose je procentuální vyjádření délky daného segmentu vzhledem k většímu z rozměrů vstupního obrázku. Na vertikální ose je pak stupeň náležení dané hodnoty do jednotlivých fuzzy množin.

4.3.3 Porovnávání znaků

V okamžiku, kdy máme dva znaky, jejich rozklady na segmenty a známe i vlastnosti těchto segmentů, můžeme přistoupit k porovnání znaků. Stupeň podobnosti znaků odvodíme od stupňů podobností jednotlivých segmentů pomocí aritmetického průměru. Přesněji postupujeme tak, že z prvního znaku vybereme nepoužitý segment a v druhém znaku k němu najdeme nejlepší shodu mezi nepoužitými segmenty. Hodnotu si uložíme jako sim_i, kde i je počet aktuálně zpracovaných segmentů prvního znaku, a oba segmenty označíme jako použité. Postup opakujeme, dokud v každém ze znaků zůstává alespoň jeden nepoužitý segment.

Jakmile jsme použili všechny segmenty z jednoho (obou) znaků, můžeme určit celkovou podobnost znaků jako:

$$sim_{Z_1}^{Z_2} = \sum_{i=1}^{\min(Z_1^N, Z_2^N)} \frac{sim_i}{\max(Z_1^N, Z_2^N)}$$

kde sim_i je získaná podobnost jednotlivých segmentů a Z_1^N (resp. Z_2^N) je počet segmentů v prvním (resp. druhém) znaku. Tento přístup umožní porovnat i znaky s různými počty segmentů. Umožníme tím tedy větší toleranci pro chybu (odlišnost) ve vstupních kostrách znaků. Na druhou stranu je tato tolerance dostatečně nízká (přímo úměrná počtu segmentů), takže se na výsledku projeví až u vícesegmentových znaků, kde je větší prostor pro chyby.

Poslední chybějící částí výpočtu podobnosti je tedy podobnost dvou segmentů. Ta je definována jako průměr podobnosti jednotlivých vlastností. V citované práci je porovnávání bivalentní, v implementaci byla (podobně jak u algoritmu VHE) přidána tolerance i při porovnávání již fuzzifikovaných hodnot. Použité stupně lze nalézt v tabulkách 2 (pro typy segmentů) a 3 (pro vertikální/horizontální pozice a délky). Výsledná podobnost dvou segmentů je tedy dána takto:

$$sim_{S_1}^{S_2} = \frac{sim_h(S_1, S_2) + sim_v(S_1, S_2) + sim_l(S_1, S_2) + sim_t(S_1, S_2)}{4}$$

kde $sim_h(S_1, S_2), sim_v(S_1, S_2), sim_l(S_1, S_2)$ a $sim_t(S_1, S_2)$ značí popořadě podobnost horizontální pozice, podobnost vertikální pozice, podobnost délky a podobnost typu segmentů S_1 a S_2.

	vertikální	horizontální	zleva stoupající	zprava stoupající	cyklus	bod
vertikální	1	0	0.5	0.5	0	0
horizontální	0	1	0.5	0.5	0	0
zleva stoupající	0.5	0.5	1	0	0	0
zprava stoupající	0.5	0.5	0	1	0	0
cyklus	0	0	0	0	1	0
bod	0	0	0	0	0	1

Tabulka 2: Stupně podobnosti jednotlivých typů segmentů. Hodnoty byly určeny experimentálně a zachycují intuitivní pohled na částečnou podobnost mezi šikmými a rovnými čarami.

	LL/UU/XS	L/U/S	C/C/M	R/D/L	RR/DD/XL
LL/UU/XS	1	0.8	0.3	0	0
L/U/S	0.8	1	0.8	0.3	0
C/C/M	0.3	0.8	1	0.8	0.3
R/D/L	0	0.3	0.8	1	0.8
RR/DD/XL	0	0	0.3	0.8	1

Tabulka 3: Stupně podobnosti fuzzifikovaných hodnot pro horizontální pozici/-vertikální pozici/délku objektu. Hodnoty byly určeny experimentálně.

5 Úprava výstupu

Posledními kroky při zpracování vstupního dokumentu je úprava a předložení výsledků uživateli. Toto odvětví je opět velmi široké a mohlo by vydat na samostatnou práci. Jeho základním předpokladem je ale dostatečná úspěšnost všech předchozích kroků rozpoznávání.

V doprovodné aplikaci můžeme najít tři základní možnosti na zpracování či ovlivnění výstupu. Je to kontrola oproti slovníku, učení se z dokumentu a export dokumentu. Z hlediska popisu je relativně zajímavá jen část první – kontrola oproti slovníku. Ostatní možnosti jsou pak spíše implementačního charakteru, a tedy jim v této části textu nebude věnován prostor. Jejich použití a popisy lze najít v 7.5.

5.1 Kontrola oproti slovníku

Pro slovníkovou kontrolu je v práci implementován základní algoritmus editační vzdálenosti. Při jeho implementaci a popisu jsem čerpal zejména ze svých po-

známek z výuky na katedře informatiky.

Editační vzdálenost

Algoritmus editační vzdálenosti pro vstupní určí jeho vzdálenost od každého slova ze slovníku. *Vzdálenost* je zde definována jako počet elementárních kroků nutných ke změně slova z jeho aktuální podoby do podoby odpovídající slovníku. Rozpoznáváme tři typy elementárních kroků:

1. Vložení znaku do slova (LED → SLED)
2. Odebrání znaku ze slova (SLED → LED)
3. Nahrazení znaku ve slově (LED → MED)

Nejprve je za použití principu dynamického programování spočtena vzdálenost vstupního slova od každého slova ze slovníku. Výsledkem je následně určeno to slovo, pro které byla nalezená vzdálenost minimální. Má–li čtenář zájem o přesný postup, doporučuji nahlédnout do zdrojových kódů přiložené aplikace.

6 Výsledky

V posledních čtyřech částech tohoto textu stručně popíšeme doprovodnou aplikaci, její použití, vnitřní strukturu a výsledky a také zmíníme možné směry dalšího vývoje. Nejprve se tedy zaměříme na míru úspěšnosti jednotlivých algoritmů popsaných na předchozích stranách.

6.1 Testovací data

Pro testování byla použita data z několika zdrojů. Prvním, a díky možnosti vytvářet data dle potřeby tím nejzásadnějším, jsou dokumenty psané autorem. Mezi ostatní zdroje pak patří „IAM Handwriting Database“, tedy databáze ručně psaných dokumentů mnoha pisatelů, která je po emailové registraci dostupná ze stránek Univerzity v Bernu [13] a databáze Andrew Seniora dostupná z [29]. Pro příklady použitých dokumentů nahlédněte do přílohy (A) nebo do dat přiložených k doprovodné aplikaci.

6.2 Metodika testování

Testování probíhalo odděleně pro každou z částí a každý z odpovídajících algoritmů.

Pro rozpoznávací algoritmy byla úspěšnost testována na, za pomoci doprovodné aplikace, ručně segmentovaných dokumentech, neboť segmentace slov na písmena je úzkým hrdlem celého procesu zpracování dokumentu, a tedy by výsledky v případě jejího použití neměly pro rozpoznávací algoritmy žádnou výpovědní hodnotu.

Tréninková množina nebyla nijak filtrována. Vzorový dokument byl pouze rozdělen na znaky a ty pak byly vloženy do tréninkové množiny.

6.3 Algoritmy rozpoznávání písmen

Počítání procentuální úspěšnosti u algoritmů rozpoznávání písmen probíhalo následovně:

1. Dokument byl ručně rozdělen na písmena.
2. Na takto získaných písmenech byl spuštěn algoritmus naučený na písmenech z jiných dokumentů.
3. Byly spočítány počty správně rozpoznaných, špatně rozpoznaných a až na velikost správně rozpoznaných písmen.
4. Písmena, která byla určena správně, nebo až na velikost správně, byla započítána jako úspěch, ostatní jako neúspěch (viz 6.3.1).
5. Úspěšnost byla vyjádřena v procentech.

6.3.1 Tolerance pro velikost písmen

U autorova rukopisu často splývá podoba velkých a malých verzí téhož písmene (například o a O či v a V). Totéž platí pro dvojici písmen l a I. Podobnost je často tak velká, že bez kontextu správný výsledek neurčí ani člověk. Znaky, u kterých pouze nesedí velikost a případ l a I, jsou tedy ve výsledných procentech pro autorův rukopis započteny jako správně určené.

Reálně se ukázalo, že ani některé odlišné znaky jako J a d nebo t a f autor sám nedokáže ve svém rukopisu bez kontextu odlišit. Použité algoritmy u nich tedy nemají naději na úspěch. Přesněji mají při stejném počtů vzorů obou znaků zhruba poloviční pravděpodobnost, že vybraný vzor bude ten správný. Tyto případy už jsou ale do výsledku započítány jako chyba, neboť by jejich tolerování mohlo vést k ovlivnění výsledku závislém na aktuální náladě a míře fantazie autora při vizuální kontrole vstupního obrazu a výsledného znaku.

6.3.2 Spojité písmo

Ještě horší je pak situace u psacího písma, kdy často splývají znaky: e a c; h a k; e a l; r a z; t a l... Také se objevuje problém s extrakcí jednotlivých znaků. Požíváme–li pro oddělení znaků pouze svislé čáry, tak pro text psaný bez ohledu na případnou analýzu často nelze ideálně oddělit jednotlivé znaky. To vedlo k výraznému zhoršení výsledků pro psací text u všech algoritmů. Zároveň u psacího textu není (z důvodu zmíněného výše) žádná tolerance chyb.

Také to do značné míry osvětlilo, proč mnoho prací o segmentaci slov na písmena používá relativně malé množiny testovacích slov.

Poznámka Výše uvedená pozorování vedla k tomu, že z ostatních zdrojů mohlo být použito jen velmi málo dat, neboť dokumenty byly psány bez větších ohledů na strojové zpracování. Konkrétní příklady lze nalézt v doprovodné aplikaci.

6.3.3 Algoritmus VHE

Algoritmus VHE se ukázal být reálně použitelným pro hůlkové písmo. Je ale nutné, aby předchozí fáze procesu kvalitně oddělily jednotlivé znaky. Pro přesné výsledky vizte tabulku 4.

VHE	Celkem testů	Znaků správně	Tolerance l/i a velikost	Znaků špatně	% správně
Autor hůlková malá	1342	1193	13	136	89.9
Autor hůlková velká	1387	1221	–	166	88.0
Autor hůlková všechna	801	598	90	113	85.9
Autor psací	415	267	–	148	64.3
Ostatní zdroje	1715	994	–	721	58.0

Tabulka 4: Výsledky dosažené za pomocí algoritmu VHE a ručního rozdělení dokumentu. Nejsou tedy ovlivněny rozdělením slov na písmena.

Porovnání výsledků VHE se starší verzí
Původní verze VHE má oproti této jednu značnou nevýhodu – musela být učena ručně, znak po znaku. Jinak docházelo k velkému ovlivnění výsledků v případě špatné kvality vzorů. Nynější verze se již dokáže učit z libovolných znaků dodaných uživatelem. Její rozšířený rozpoznávací proces také dokáže tolerovat více odchylek. Pro přímé srovnání uveďme, že původní verze dosahuje na stejných datech (hůlkové písmo) úspěšnosti okolo 50%. Pro data vybíraná ručně se pak blíží hranici 82%. To ale vyžaduje velkou časovou investici.

6.3.4 Segmentační algoritmus

Pro segmentační algoritmus byly provedeny podobné testy jako pro algoritmus VHE. Výsledky však odpovídaly tomu, že algoritmus byl navržen pouze pro velká tiskací písmena. Relativně úspěšný pak byl i pro malá tiskací písmena (viz tabulka 5). Pro psací písmo je algoritmus v podstatě nepoužitelný. Důvodem by mohl být celkově mnohem zaoblenější styl písma při spojitém psaní. Nelze tedy najít tolik specifických segmentů.

Zároveň se ve velké míře ukázala jeho slabá stránka – potřeba přesné kostry znaku. Při testování s pečlivě ručně vybíranými tréninkovými znaky (včetně kontroly jejich kostry) je algoritmus schopen dosáhnout až k 80% hranici úspěšnosti. Toto ale po uživateli nelze požadovat. Zde zobrazené výsledky tedy používají stejnou metodiku jako testy u VHE.

Segm. alg.	Celkem testů	Znaků správně	Tolerance l/i a velikost	Znaků špatně	% správně
Autor hůlková malá	1342	683	19	640	52.3
Autor hůlková velká	1387	839	–	548	60.5

Tabulka 5: Výsledky dosažené za pomocí segmentačního algoritmu pro rozpoznávání znaků a ručního rozdělení dokumentu. Nejsou tedy ovlivněny rozdělením slov na písmena. Pro jiné typy písem se algoritmus ukázal být nepoužitelný.

6.3.5 Neocognitron

Neocognitron implementovaný v doprovodné aplikaci se mi nepodařilo nastavit tak, aby měl jakoukoliv rozumnou míru úspěšnosti pro větší počet možných tříd dat (písmena, čísla). Pro čtyři třídy se pohybujeme na úspěšnosti okolo 68%. S rostoucím počtem tříd pak úspěšnost velmi rychle klesá a už pro číslice (deset tříd) je implementovaná verze nepoužitelná. Analýzou dostupných možností jsem došel k několika možným příčinám:

- Jak již bylo uvedeno v části textu o implementaci neocognitronu (4.2.7), tak v některých mně dostupných článcích jsou zmínky o různých speciálních pravidlech pro některé z vrstev (práce s kontrastem, speciální učící data...). Nicméně nikde jsem nenašel jejich kompletní popis. Kombinace informací ze zdrojů [23] a [25] mě přivedla k jedné z takových úprav, a to speciální učení první skryté vrstvy pomocí úseček v šestnácti směrech. Dle různých zmínek v uvedených zdrojích to ale vypadá, že autor neocognitronu ve svých implementacích používá takovýchto úprav více.

- Dalším důvodem pro neúspěch této implementace by mohl být fakt, že většina částí o výsledcích v uvedených zdrojích hovoří pouze o rozpoznávání číslic. Výjimkou je práce [30] – ta ale pochází z roku 1991 a odpovídající neocognitron v mnoha ohledech neodpovídá dnešnímu modelu. Jejím největším problémem je pak skutečnost, že trénink je prováděn za velké asistence zvnějšku – pro každou rovinu v jednotlivých vrstvách jsou referenční vektory vybírány ručně.

- Možným důvodem je i velká ztráta informace při úvodním zmenšení obrázků. Toto nemusí vadit pro několik tříd, ale pro množinu 30 či více znaků tím můžeme ztratit potřebné odlišnosti jednotlivých tříd. Použití velkých vstupních obrázků sice částečné zlepšení přineslo, nicméně doba běhu se stala neúnosnou.

V rámci hledání chyby jsem zkoušel najít i nějakou jinou implementaci, se kterou bych své postupy a výsledky porovnal. To mě přivedlo k práci [31]. Jedná se o implementaci neocogitronu v jazyce Java za použití zdrojů, které z velké části obsahují stejné postupy, jako zdroje použité v této práci. V sekci výsledků její autor pak konstatuje stejný problém, který jsme popsali výše – náročnost správného nastavení parametrů sítě. Tato implementace dosahuje úspěšnosti 73% pro 5 tříd znaků (posunuté či lehce deformované číslice 0 – 4).

Poznámka Předchozím zamyšlením v žádném případě nemíním zpochybnit či snížit funkčnost neocognitronu jako takového – jeho autor ve svých pracích pravidelně dosahuje úspěšnosti 90% a více pro číslice (10 tříd). Chyba je tedy zjevně na mé straně.

6.4 Segmentace slov

Určení objektivních výsledků segmentace pomocí algoritmu počítajícího průměrně nejdelší cestu grafem je v aktuálním stavu v podstatě nemožné. Důvodem je jeho závislost na rozpoznávacím algoritmu. Původním plánem bylo pro tuto část použít neocognitron, ale implementací dosažené výsledky toto znemožňují.

Nabízí se možnost použít algoritmus VHE, ale dle testů to vypadá, že ten pro takovéto použití vhodný není. I přes relativně vysokou úspěšnost při odlišování znaků má totiž problémy s označením tvarů, které znakem nejsou. Zejména pak u části písmen typu w či m. Extrémním příkladem může být slovo minimum, kde má vysokou váhu mnoho segmentací. Jednotlivé části všech jeho písmen jsou totiž velice podobné znaku i.

Nicméně potenciál algoritmu se zdá být vysoký. U slov, která takto dále dělitelná písmena neobsahují, jsou jeho výsledky zajímavé i s algoritmem VHE. Pro objektivní zhodnocení je ale nutné mít k dispozici OCR algoritmus, který dokáže s vysokou úspěšností říct jestli daný segment je znakem, či nikoliv.

Zde bych čtenáře odkázal na doprovodnou aplikaci, která poskytne užitečný náhled na situaci.

6.5 Ostatní algoritmy

U některých z ostatních algoritmů implementovaných a popsaných v práci má, spíše než popis úspěšnosti, smysl diskuze vstupů, pro které fungují. U všech z nich má uživatel v aplikaci možnost výsledky pro nepodporované či chybně vyhodnocené typy dokumentů upravit ručně.

6.5.1 Segmentace tabulek

Segmentace tabulek tak, jak je popsána a implementována v doprovodné aplikaci, je funkční pouze pro tabulky omezené podmínkami v části 2.1. Pro otestování je několik takových tabulek přiloženo k doprovodné aplikaci. Pokud bychom chtěli podporu rozšířit i pro jiné typy tabulek, tak bychom museli upravit zejména

fázi shlukování význačných bodů do skupin definujících buňky a jejich následné řazení do mřížky tabulky.

6.5.2 Úprava sklonu

Algoritmus pro úpravu sklonu se během testování ukázal, vzhledem k jeho přímočarosti, až překvapivě účinný. Našel jsem pouze dva typy dokumentů, se kterými si neporadí – dokumenty s velmi malým množstvím textových řádků a dokumenty, kde řádky nejsou rovnoběžné. Poznamenejme, že je předpokládáno, že z dokumentu byly předem odstraněny netextové oblasti.

6.5.3 Základní segmentace dokumentu

Algoritmus pro základní segmentaci dokumentu použitý v práci je, podobně jako algoritmus pro tabulky, plně funkční pouze pro dokumenty splňující omezení daná v 2.1. Problémy mu také mohou činit velmi malé obrázky a tabulky či jejich nespojitě kreslené verze.

6.5.4 Segmentace na řádky a slova

Principiálně stejná omezení platí i pro segmentaci textu na řádky a slova. Nejsou–li mezi řádky či slovy mezery, algoritmus je detekuje jako jednolité celky. Toto je v použité implementaci částečně řešeno možností učení statistik o dokumentu (viz 7.5.11).

7 Uživatelská příručka

Nyní popíšeme instalaci, rozhraní, dostupnou funkčnost a ovládání doprovodné aplikace. Začněme tedy jejími požadavky na prostředí.

7.1 Požadavky

7.1.1 Hardware

Aplikace žádné specifické hardwarové požadavky nemá, nicméně obsahuje mnoho výpočetně náročných algoritmů a díky uchovávání velkého množství grafických informací, potřebuje relativně hodně paměti. Pro bezproblémový běh tedy doporučuji stroj s vícejádrovým procesorem o taktu alespoň 2.5 GHz a 4 GB operační paměti. Většina testování pak proběhla na počítači se čtyřjádrovým procesorem Intel i5 pracujícím na frekvenci 3.4 GHz a 8 GB dostupné operační paměti.

7.1.2 Software

Z hlediska softwarových závislostí je k běhu aplikace potřeba aktuální verze Java Runtime Environment. Toto je důležité, neboť aplikace obsahuje některé komponenty, které ve starších verzích Javy nemusí být dostupné. Týká se to zejména některých prvků grafického rozhraní (viz 8.1).

Díky použití Javy je aplikaci možno provozovat na různých operačních systémech, nicméně pro tuto chvíli pro její běh doporučuji použití Windows 8.1, ve kterých byla aplikace testována před odevzdáním práce. Vývoj a testování sice z velké části probíhaly na Linuxu (konkrétně Debian 7 a Kubuntu 15.10 za použití Oracle JRE 8), ale grafické prostředí KDE vykazovalo nestandardní chování při práci s JavaFX okny.

7.2 Instalace

Samotná aplikace je, spolu s dalšími daty nezbytnými pro její běh, distribuována jako jar archiv v adresáři *bin* na přiloženém CD (viz B). Tento adresář stačí nakopírovat na libovolný disk, na který uživatel může zapisovat a dvojklikem na „dp.jar“ archiv práci spustit. Další možností je zvolený adresář otevřít v konzoli a zadat `java -jar dp.jar`.

7.3 Nové písmo – nový profil

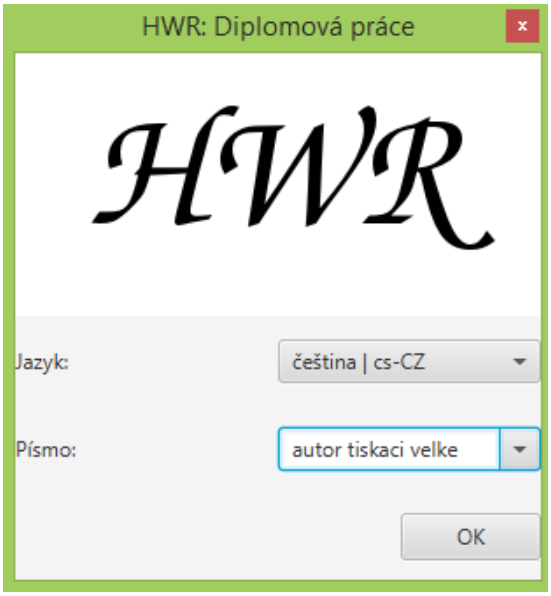
Chceme–li aplikaci naučit více různých druhů vlastního písma (tiskací, psací, kapitálky...), tak pro každé z nich musíme vytvořit vlastní profil. Počet profilů je limitován pouze omezením plynoucím z použitých technologií. Prakticky tedy omezen není.

7.4 Uživatelské rozhraní

Po spuštění aplikace se nám nejprve zobrazí okno se základním nastavením (viz obrázek 20), kde si můžeme vybrat jazyk uživatelského rozhraní a některé z již naučených písem. Je také možné vytvořit písmo nové. Z jazyků je na výběr čeština a angličtina.

Připraveno je i několik profilů s demonstračními daty. Pro vytvoření profilu pro nové písmo stačí do textového pole zadat zvolené jméno a stisknout OK. Pokud písmo se zadaným jménem ještě neexistuje, budeme dotázáni, zda si jej přejeme vytvořit. V opačném případě budou načtena jeho data.

Libovolnou z těchto možností završíme úvodní nastavení a objeví se hlavní okno aplikace.



Obrázek 20: Úvodní okno aplikace umožňující výběr jazyka a písma.

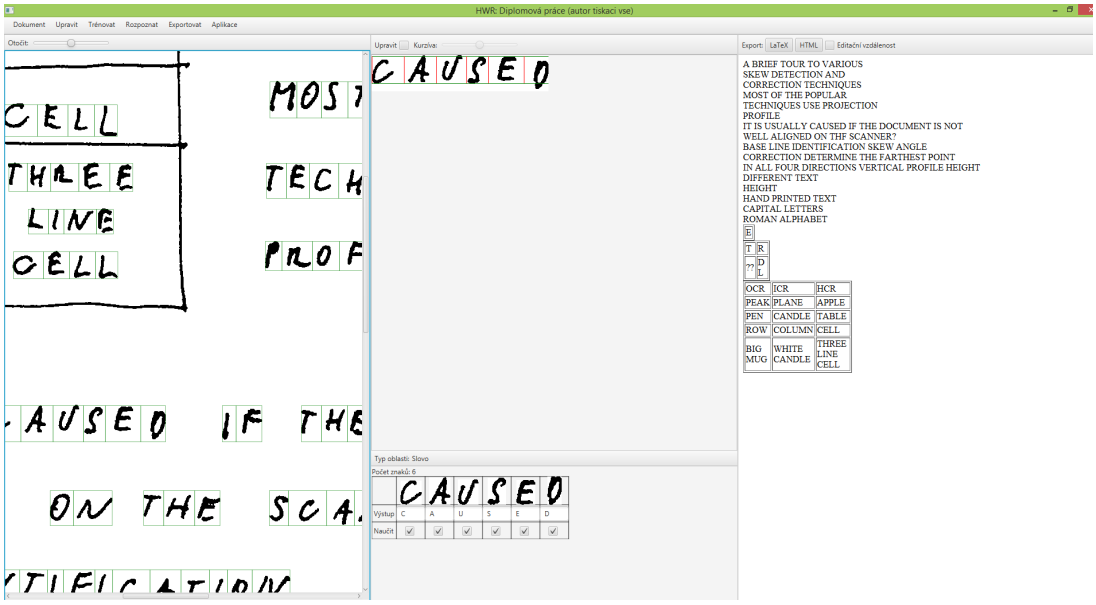
7.4.1 Hlavní okno aplikace

Hlavní okno aplikace se skládá ze čtyř částí: menu, navigační modul, modul úprav a modul výsledku (viz obrázek 21). Jednotlivé položky menu jsou popsány v další části příručky spolu s dostupnou funkcionalitou. Zde tedy popíšeme pouze moduly.

Navigační modul

Navigační modul je v levé části aplikace. Jeho účelem je umožnit navigaci ve zpracovávaném dokumentu. Druhotnou úlohou je pak vizuální zpětná vazba o aktuálním stavu dokumentu.

Máme–li otevřený rozpracovaný dokument, pak pomocí klikání do jeho obrázku v tomto modulu, můžeme vybírat jednotlivé již nalezené oblasti. Vybraná oblast pak bude zobrazena v modulu úprav (viz dále).



Obrázek 21: Hlavní okno aplikace. V horní části je hlavní menu, pod ním pak zleva doprava: navigační modul, modul úprav a modul výsledku.

Každé kliknutí vybere oblast pod kurzorem o jednu úroveň hlouběji, než se právě nacházíme. Tedy, máme–li vybrán řádek a klikneme na některé z jeho slov, tak bude vybráno. Obdobně pro tabulky a jejich buňky, textové oblasti a jejich řádky atd.

V okamžiku, kdy už vybraný objekt neobsahuje další objekty k vybrání, způsobí kliknutí vybrání celého dokumentu. Analogicky bude dokument vybrán i pokud klikneme někam mimo aktuálně zvolenou oblast.

Navigační modul nám také podává vizuální informaci o aktuálním stavu zpracovávaného dokumentu:

- Červeně podbarvené oblasti jsou oblasti textu se zatím neznámým významem znaků.
- Znak, který červeně podbarven není, má již určen výsledek.
- Zelený rámeček okolo znaku značí, že je připraven pro přidání do tréninkové množiny.
- Znaky označené malým zeleným čtverečkem v jejich levém horním rohu jsou již přidány do tréninkové množiny.

Modul úprav

Modul úprav (uprostřed okna) slouží ke změně segmentace a významu jednotlivých oblastí dokumentu. Jeho přesné použití pro jednotlivé úpravy je popsáno

v odpovídajících částech popisu dostupné funkcionality (7.5). Zde zmiňme pouze prvky v jeho horní části.

- Zaškrtávací políčko *Upravit* přepíná mezi pohybem a úpravami v právě zobrazené části.
- Posuvník *Kurzíva* umožňuje změnit náklon právě vybraného slova.

Modul výsledku
Pravý modul, tedy modul výsledku, využijeme v samotném závěru zpracování dokumentu pro jeho kontrolu a export. Kromě plochy pro zobrazení výsledku obsahuje tlačítka *LaTeX* a *HTML* sloužící pro export do odpovídajících formátů a zaškrtávací políčko *Editační vzdálenost*, které aktivuje porovnávání se slovníkem před exportem.

7.5 Dostupná funkcionalita

Nyní na příkladech použití jednotlivých funkcí pro zpracování dokumentu popíšeme, jak s aplikací pracovat.

7.5.1 Práce s dokumenty

Začneme otevřením nějakého dokumentu pro zpracování. Podporovány jsou různé obrázkové formáty (např. *png*, *jpg*,*bmp* či *tiff*). Požadavky na obsah vstupního obrázku jsou pak popsány v části 2.1. Dokument načteme pomocí položky menu *Dokument–Nový...*

Při načítání obrázku automaticky dojde k jeho binarizaci, filtraci a dalším úpravám popsaným v sekci 2. Toto může v případě větších obrázků chvíli trvat. Výsledek se zobrazí v navigačním modulu.

Je–li obrázek větší než oblast dostupná pro modul, bude pohyb v něm umožněn pomocí posuvníků nebo pomocí myši. Zároveň budou červeně zvýrazněny všechny neznámé textové části dokumentu – ve výchozím stavu tedy celý dokument.

Dalšími možnostmi pro práci s dokumentem jsou uložení jeho aktuálního stavu pro pozdější zpracování (*Dokument–Uložit...*), načtení dříve uloženého rozpracovaného dokumentu (*Dokument–Otevřít...*) a zavření aktuálně otevřeného dokumentu (*Dokument–Zavřít*).

7.5.2 Úprava sklonu dokumentu

Druhým krokem, nezbytným pro úspěšné zpracování, je vyrovnání případně křivě naskenovaného obsahu. Toho lze docílit dvěma způsoby. Buďto ručně, za pomoci posuvníku v levé horní části navigačního modulu, nebo automaticky pomocí jedné z položek v menu *Upravit–Vyrovnat dokument*. Vybrat si lze ze dvou položek – *Smíšený dokument* a *Čistý text*.

Rozdíl je v typu dokumentů, se kterými pracují. Možnost *Čistý text* je mnohem rychlejší, ale ve většině případů si neporadí s dokumenty obsahujícími tabulky či obrázky. Možnost *Smíšený dokument* pak nejprve oddělí obrázky a tabulky od textu. Teprve poté přistoupí k určení úhlu pro otočení. Cenou je pak čas a výkon potřebný pro tuto analýzu dokumentu.

Také je zde potřeba upozornit, že účel funkce je opravit drobný sklon dokumentu způsobený skenováním či nepřesným psaním. Kontrolovány jsou tedy pouze úhly mezi -45 a 45 stupni. Pokud je náš dokument otočený o úhel mimo tento rozsah, je nutné jej nejprve ručně pootočit.

Je důležité, abychom správné natočení určili hned na začátku práce s dokumentem. Změna rotace totiž vede ke zrušení aktuální segmentace.

7.5.3 Navigace v dokumentu

Při dalším zpracování již budeme potřebovat možnost navigace mezi jednotlivými částmi dokumentu. K tomuto účelu slouží navigační modul (7.4.1). V aktuálním stavu, tedy hned po načtení a případném natočení dokumentu, bude klikání pouze přepínat mezi celým dokumentem a výchozí verzí jeho textové vrstvy.

7.5.4 Základní segmentace

Nyní, když máme načtený, upravený a vyrovnaný dokument a víme, jak se v něm pohybovat, můžeme přistoupit k jeho rozdělení na textové oblasti, tabulky a obrázky. Pokud jde o dokument obsahující pouze text, tak tuto část zpracování můžeme vynechat.

Opět máme na výběr ze dvou možností: ruční nebo automatická segmentace. Ruční segmentace se, stejně jako všechny ostatní ruční úpravy, provádí v modulu úprav.

Kliknutím tlačítka do navigačního modulu vybereme k úpravě celý dokument. Toto je indikováno nápisem *Typ oblasti: Dokument* na panelu ve spodní části modulu úprav. Je–li zde jiný nápis, pak do obrázku klikáme znovu, dokud nevybereme celý dokument.

Pomocí myši nebo posuvníků se posuneme k objektu, který chceme označit a pro aktivaci možnosti změn zaklikneme volbu *Upravit* v horní části modulu úprav.

Označení tabulky (resp. obrázku) provedeme stisknutím levého (resp. pravého) tlačítka myši, jeho přidržením a tahem. Tím dojde k vykreslení zeleného (resp. modrého) obdélníku okolo vybraného objektu. Při puštění tlačítka se pak ve struktuře dokumentu vytvoří záznam o novém objektu ve vybrané oblasti. Oblast také bude odstraněna z textové vrstvy dokumentu.

Je–li vybraný objekt větší než oblast dostupná pro modul úprav, lze se v něm při výběru pohybovat pomocí šipek.

Stejný úkon opakujeme pro všechny tabulky a obrázky, které dokument obsahuje. Jak již bylo uvedeno výše (2.1), jednotlivé oblasti obsahující tabulky

a obrázky se nesmí překrývat. Pokud se takovou překrývající oblast pokusíme vytvořit, tak všechny překryté oblasti budou smazány.

Uděláme–li během segmentace chybu, lze pro smazaní již vytvořených částí použít prostřední tlačítko. To při stisknutí a tahu vykreslí červený obdélník, který odstraní segmentaci všech tabulek a obrázků, se kterými má průnik. Nemáme–li prostřední tlačítko, lze mazání aktivovat přidržením klávesy *Ctrl*.

Druhou možností základní segmentace dokumentu je možnost menu *Upravit–Rozdělit–Dokument na části*. Při jejím zvolení se spustí automatická analýza základní struktury dokumentu, která detekuje všechny objekty, které neodpovídají slovům (viz 3.1). Pokusí se také určit, jestli se jedná o tabulky či obrázky. Pokud výsledek neodpovídá očekáváním, lze jej ručně upravit pomocí postupu popsaného výše.

7.5.5 Zpracování tabulek

Máme–li detekovány případné tabulky, je nutné je rozdělit na jednotlivé buňky. I zde toho můžeme docílit ručně nebo pomocí položky menu *Upravit–Rozdělit–Tabulky na buňky*.

Ruční segmentace probíhá úplně stejně jako ta u tabulek. Přidržením levého tlačítka postupně označíme oblast každé z buněk. Jediný rozdíl je, že při vytvoření nové buňky budeme dotázáni na její pozici v tabulce, tedy pořadí řádku a sloupce do kterého nová buňka patří.

Druhou možností je výše zmíněná položka v menu. Ta spustí algoritmus, který se pokusí všechny tabulky rozdělit automaticky. Omezení na typ tabulek, které algoritmus rozpozná, opět najdeme v 2.1.

7.5.6 Segmentace textu

Po označení všech netextových částí dokumentu přichází na řadu segmentace textu na řádky, slova a písmena. Stejně jako v předchozích případech i toto lze udělat automaticky nebo ručně.

Pokud náš dokument splňuje požadavky na vstup, tak je nejlepší možností provést segmentaci textu na slova pomocí položky menu *Upravit–Rozdělit–Texty na slova*.

Vyžaduje–li dokument ruční zásah, pak jej opět provádíme v prostředním modulu. Nejprve vybereme textovou oblast, kterou chceme rozdělit (tj textovou vrstvu dokumentu nebo obsah některé z tabulek). Nyní popíšeme jednotlivé úrovně dělení.

Textová oblast na řádky

Kliknutím na řádek levým tlačítkem dojde, je–li to možné, k jeho oddělení od ostatních řádků. Toto je možné vždy, když je mezi řádky mezera přes celou šířku nadřazené oblasti. Alternativně můžeme pravým tlačítkem označit tuto mezeru mezi řádky.

Pokud taková mezera ale neexistuje, tak přidržením tlačítka *Ctrl* aktivujeme plnou kontrolu nad dělením. Tedy každé kliknutí rozdělí danou část vybrané textové oblasti horizontální čarou na dvě. Můžeme použít jak levé tlačítko vytvářející začátky řádků (signalizováno červenou barvou vzniklé čáry), tak pravé tlačítko vytvářející začátky mezer mezi řádky (signalizováno zelenou barvou).

Při postupné segmentaci oblasti si můžeme všimnout, že nové řádky jsou červeně zvýrazněny i v navigačním modulu. To znamená, že jde o textovou oblast, pro kterou ještě není známá výsledná podoba.

Řádek na slova
Dělení řádků na slova funguje analogicky k dělení textových oblastí na řádky. Jen dělící čáry vytváříme ve vertikálním směru.

Slova na znaky
Pro dělení slov na písmena je implementováno několik algoritmů závisejících na typu písma. Pro hůlkové písmo s mezerami mezi písmeny lze použít položku menu *Upravit–Rozdělit–Slova na znaky–Hůlkové písmo*. Toto spustí algoritmus popsaný v části 3.4.4.

Pro libovolné písmo pak lze použít algoritmus z části 3.4.1, který spustíme pomocí položky menu *Upravit–Rozdělit–Slova na znaky–Použít VHE*, nebo *Upravit–Rozdělit–Slova na znaky–Použít Segm. alg.*. Rozdíl je v algoritmu rozpoznávání znaků volaném z dělícího algoritmu. Pro jeho aktuální možnosti, prosím, před spuštěním konzultujte část textu o výsledcích (6.4).

Chceme–li dělit slovo na znaky ručně, pak pro vytvoření/zrušení nového bodu dělení stačí kliknout do obrázku slova v modulu úprav. Změny se promítnou do informací o slově ve spodní části modulu. Při ručním dělení je vnitřně využíváno dělení za pomoci jádra slova popsané v textu práce.

Poznamenejme, že pro aktivaci těchto možností je nutné, aby použitý algoritmus byl předem naučen na daný typ písma a aby v dokumentu byla nějaká zatím nerozdělená slova.

7.5.7 Učení

Pro úspěšné zpracování dokumentu je všechny z algoritmů nejprve potřeba naučit, jak naše písmo vlastně vypadá. To provedeme pomocí zpracování několika dokumentů se vzory. Vhodný soubor dokumentů se vzory je takový, který obsahuje několik (čím více, tím lépe) výskytů každého znaku, který budeme chtít v budoucnu rozpoznávat. Pro jednoduchost zde předpokládáme, že máme jeden dokument obsahující vše (viz 7.5.12). Pro vzory rozdělené přes více dokumentů je postup úplně stejný, jen se zpracovává jeden dokument po druhém.

Nejprve tedy provedeme základní segmentaci, ať už ručně, nebo s dopomocí některých z výše popsaných funkcí. Poté musíme ke každému vytvořenému znaku zadat jeho hodnotu. Hodnota znaku je libovolný textový řetězec, na který si přejeme, aby se daný znak přeložil při generování výstupu (viz 7.5.15).

Jsou důležité jak prázdné znaky (mezery, tabulátory. . .), tak velikost písmen. Tedy například slova „Alpha“ a „alpha“ budou považována za různé hodnoty. Hodnoty znaků se zadávají ve spodní části prostředního modulu do polí označených *Výstup* (zobrazeno pouze při výběru některého ze slov). Při zadávání si můžeme všimnout, že hned pod polem *Výstup* je pole *Naučit*, které určuje jestli se daný znak má přidat do množiny vzorů, nebo ne (viz dále).

Je dobré si uvědomit, že pokud naše písmo obsahuje například nějaké specifické, těžko rozdělitelné slitky, tak je můžeme naučit jako celek. Stačí hodnotu pouze nastavit na požadovanou skupinu znaků nebo využít možnosti automatického překladu (viz 7.5.15).

7.5.8 Kurzíva

Jak již bylo diskutováno dříve (3.4.5), jsou slova psaná kurzívou při segmentaci problematická. Aplikace umožňuje použití algoritmů i pro text psaný kurzívou, její možnosti jsou ale omezené. Při segmentaci slova na znaky můžeme u slova upravit sklon pomocí posuvníku v horní části prostředního modulu. Volíme takovou možnost, aby znaky byly v co nejsvislejší poloze. Je–li náš rukopis celkově skloněn (nejedná se tedy jen o pár slov v rámci dokumentu), pak pro naučení míry sklonu využijeme možnosti učení vlastností dokumentů (7.5.11). Získaná data pak budou použita při segmentaci dalších dokumentů se stejným rukopisem.

7.5.9 Tréninková množina

V okamžiku, kdy je celý vzorový dokument zpracován a máme vyplněny hodnoty všech znaků vybraných pro učení, můžeme přistoupit k naplnění tréninkové množiny. Pro práci se vzorovými daty slouží položky v menu *Trénovat–Tréninková DB*.

První z nich je *Spravovat. . .* , která zobrazí okno se všemi vybranými znaky, jejich hodnotami a možnostmi pro jejich úpravu a mazání. Změna hodnoty se provádí rozkliknutím příslušné buňky a přepisem hodnoty, mazání pak kliknutím na tlačítko smazat.

Další položkou menu pro tréninkovou množinu je *Přidat vybrané znaky*. Její volbou zapříčiníme přidání všech volbou *Naučit* označených znaků dokumentu do tréninkové množiny.

Účel ostatních možností, tedy *Uložit data*, *Načíst data* a *Vymazat data*, je zřejmý. Jen poznamenejme, že data uložena na disk (získána z předchozích sezení a zachována i po zavření aplikace) se nezmění dokud explicitně nevybereme volbu *Uložit data*. Tím je zajištěna možnost návratu do původního stavu (volbou *Načíst data* nebo zavřením aplikace bez uložení), kdyby se nám změny nepodařily dle našich představ.

7.5.10 Učení dle tréninkové množiny

Máme–li vytvořenu databázi vzorových znaků splňující naše požadavky, můžeme přistoupit k učení jednotlivých algoritmů. Uvedeme zde popis učení jen pro algoritmus VHE, u zbylých dvou jsou možnosti obdobné.

K tréninku algoritmu slouží položky v menu *Trénovat–VHE*. První z nich je *Trénovat na DB*, která spustí trénink algoritmu na datech z tréninkové množiny. Tato činnost může chvíli trvat. Po jejím dokončení je algoritmus připraven k použití.

Dalšími možnostmi jsou položky *Uložit data*, *Načíst data* a *Vymazat data*. Jejich účel je opět zřejmý. Data na disku opět nejsou změněna, dokud si to nevyžádáme volbou možnosti *Uložit data*.

Učení lze samozřejmě pouštět opakovaně. To, mimo jiné, dává možnost algoritmy v budoucnu přeučit s tréninkovou množinou obohacenou o nové vzorové znaky.

7.5.11 Učení vlastností písma

V aplikaci je i několik možností pro zlepšení výsledků většiny segmentačních algoritmů na základě vlastností zvoleného dokumentu. Myšlenka je taková, že máme–li několik dokumentů napsaných stejnou osobou při stejné příležitosti, pak jsou si pravděpodobně velice podobné. Toho lze využít k minimalizaci efektu specifičnosti písma každého uživatele na výsledky segmentace.

Pokud se do takové situace dostaneme, tak nejprve zpracujeme první z těchto dokumentů do požadované podoby. A následně aplikaci naučíme vlastnosti tohoto dokumentu pomocí položky menu *Trénovat–Vlastnosti dokumentu–Trénovat na dok.*

Pokud kdykoliv poté aktivujeme možnost *Trénovat–Vlastnosti dokumentu–Aktivovat*, tak budou všechny segmentační algoritmy využívat hodnoty získané z tohoto dokumentu místo svých výchozích.

Aplikace si pro dané písmo pamatuje statistiky vždy o nejvýše jednom dokumentu.

Tato možnost je také stěžejní pro zpracování kurzívy (viz 7.5.8).

Ostatní položky menu (*Uložit data*, *Načíst data* a *Vymazat data*) pak fungují ve stejném duchu jako u trénování rozpoznávacích algoritmů a tréninkové množiny.

7.5.12 Vzorové dokumenty

Jak již bylo zmíněno výše, základní učení probíhá na vzorových dokumentech. To jsou dokumenty obsahující všechny možné znaky s dostatečným počtem exemplářů od každé možnosti. Pokud je sami vytváříme, pak jsou dobrou volbu anglické pangramy, tedy věty obsahující všechna písmena anglické abecedy. Pro inspiraci zde uveďme některé ze známých pangramů:

1. Quick brown fox jumps over the lazy dog.

2. Six big devils from Japan quickly forgot how to waltz.
3. Grumpy wizards make a toxic brew for the jovial queen.
4. Jack quietly move up front and seized the big ball of wax.
5. The wizard quickly jinxed the gnomes before they vapourized.
6. Public junk dwarves quiz mighty fox.
7. Painful zombies quickly watch a jinxed graveyard.

Typický vzorový dokument autorova písma lze vidět na obrázku 22 v příloze A.

7.5.13 Jak tedy aplikaci učit?

Nyní známe všechny postupy potřebné pro učení aplikace. Můžeme tedy stručně zrekapitulovat učící proces. Nejprve získáme vzorové dokumenty. V ideálním případě takové, které mají vlastnosti popsány výše (7.5.12). Nemáme–li je k dispozici, použijeme dostatečné množství jiných dokumentů tak, abychom pokryli všechny možné vstupní znaky.

Tyto dokumenty tedy jeden po druhém načítáme, případně vyrovnáváme a za použití popsaných segmentačních technik získáváme jednotlivé znaky. Ty z nich, které si přejeme naučit, přidáme do tréninkové množiny. V okamžiku, kdy jsme s tréninkovou množinou spokojeni, můžeme přistoupit k učení jednotlivých algoritmů.

Doporučuji se při segmentaci spojitých slov na znaky chovat co nejpředvídatelněji, tedy segmentovat vždy na začátku nebo konci písmene. V případě, kdy segmentujeme vždy někde náhodně mezi písmeny, může množství potřebných vzorových znaků značně narůst. V případě ruční segmentace vertikálně oddělených písmen na přesném místě nezáleží.

Zároveň v aktuální verzi aplikace doporučuji neučit interpunkci a nepoužívat diakritiku.

7.5.14 Rozpoznávání znaků

Máme–li ukončený proces učení, je rozpoznávání znaků otázka pár kliknutí. Přesněji pro otevřený dokument nejprve provedeme vyrovnání a segmentaci dle pokynů výše a samotné rozpoznávání následně spustíme některou z položek v menu *Rozpoznat*. Význam položek je zřejmý z jejich názvů, snad jen upozorněme, že položka *Vymazat výsledky* smaže nastavené hodnoty ze všech znaků v aktuálním dokumentu.

Zde bych jen upozornil, že z důvodu popsaných v části o výsledcích (6.3.5), nelze při spuštění neocognitronu očekávat rozumné výsledky.

7.5.15 Generování výstupu

K závěrečným fázím práce s dokumentem patří vytvoření a uložení souboru s nalezenými daty. K tomuto účelu slouží modul výstupu a položka menu *Exportovat*.

Při práci na dokumentu lze v modulu výstupu pozorovat, jak se postupně vyvíjí průběžný výsledek, od segmentace až po hodnoty jednotlivých slov a znaků.

V okamžiku, kdy jsme s výsledky spokojeni, přistoupíme k jejich exportu do některého z výstupních formátů. Na výběr máme ze tří formátů:

- HTML – pomocí položky menu *Exportovat–Do HTML* nebo pomocí tlačítka *HTML* v pravém modulu.
- LaTeX – pomocí položky menu *Exportovat–Do LaTeXu* nebo pomocí tlačítka *LaTeX* v pravém modulu.
- Interní formát – pomocí položky *Exportovat–Do intern. formátu* v menu.

Použití formátů LaTeX a HTML je zřejmé, interní formát pak slouží k přenosu rozpracovaného dokumentu. Lze jej zpětně načíst pomocí položky menu *Dokument–Otevřít...*.

Další funkcí související s výstupními formáty je překlad nalezených prvků na řetězce odpovídající potřebám jednotlivých formátů. K tomu slouží slovník formátů dostupný z menu *Exportovat–Slovník formátů*. Volba *Spravovat...* otevře okno s možností definovat překlady jednotlivých záznamů.

Levý sloupec obsahuje hodnoty odpovídající těm, které při učení zadáváme v textových polích pro hodnoty znaků. Ostatní sloupce pak obsahují řetězce, na které se mají hodnoty přeložit při exportu do jednotlivých formátů. Poslední sloupec obsahuje možnost daný záznam o překladu smazat. Sloupce pro hodnotu a její překlady jsou editovatelné.

Pro přidávání nových záznamů lze použít textová pole a tlačítko ve spodní části okna.

Při exportu pak každá hodnota každého znaku uvedeného v tomto slovníku bude nahrazena odpovídajícím přepisem pro zvolený formát. Pokud se nějaká hodnota ve slovníku nevyskytuje, bude použita původní hodnota bez překladu. Není tedy nutné uvádět do slovníku písmena, která jsou v obou podporovaných formátech reprezentována stejně. Naopak je potřeba uvádět speciální znaky jako třeba „%“, které v LaTeXu uvozuje komentář. Správná podoba pro % je tedy \% pro LaTeX a % pro HTML.

Poslední možností exportu je porovnání výsledných slov se slovníkem. K tomu slouží možnost *Editační vzdálenost* v pravém modulu. Je–li aktivní, budou data před exportem porovnána se slovníkem. Každé slovo upravené dle slovníku pak bude do výstupu pro odlišení zapsáno kapitálkami. Chceme–li změnit použitý slovník, můžeme k tomu použít položku menu *Exportovat–Editační vzdálenost–Načíst slovník...*.

8 Implementace

Předposlední částí této práce je stručný popis implementace doprovodné aplikace. Jedná se o desktopovou aplikaci pro rozpoznávání ručně psaného textu psanou za použití objektově orientovaného paradigmatu.

8.1 Použité technologie

Aplikace je napsána v jazyce Java verze 8 za použití vývojového prostředí Netbeans 8, verzovacího systému Git a testovacího frameworku JUnit 4. Pro implementaci uživatelského rozhraní pak byla zvolena platforma JavaFX. Tato platforma sice zatím není tak „dospělá“ jako Swing, ale díky pozornosti, kterou jí Oracle věnuje, tento nedostatek pravděpodobně rychle dožene. Toto, spolu s přáním naučit se něco nového, nakonec vedlo k závěrečné volbě.

8.2 Použité knihovny

V práci jsou použity dvě externí knihovny. Jmenovitě Jai image IO, dostupná z [32] pod BSD licencí s dodatkem pro nukleární průmysl, používaná pro práci s obrazovým formátem *tiff*, ve kterém je jedna z použitých databází ručně psaných slov. A JGraphT, dostupná z [33] pod EPL licencí, která slouží pro práci s grafy. Znění zmíněných licencí lze najít v adresáři *lib/licence*.

8.3 Struktura aplikace

Kód aplikace je členěn do sedmi logických balíků (patnáct skutečných Java balíků) souvisejících jak s logickou strukturou aplikace, tak se strukturou samotného procesu zpracování ručně psaných dokumentů.

Balík Core

Logický balík *Core* je dále rozdělen na balík *Core.DataWorkers*, obsahující různé třídy pro práci s různými typy dat, a balík a *Core.ImageManipulations* obsahující různé funkce pro transformaci obrázků.

Balík Data

Logický balík *Data* je pak dále rozdělen na balíky *Common, Core, DocumentParts* a *Exceptions*. Z nich nejzajímavější je asi balík *DocumentParts* obsahující datový model pro členění dokumentu (viz 8.4.1).

Balík OCR

Dalším logickým celkem je balík *OCR*. V něm jsou obsaženy balíčky jednotlivých algoritmů rozpoznávání znaků. Tedy *Neocognitron*, *SegmentingAlgorithm* a

VHE spolu s balíkem *API* definujícím požadavky na rozhraní rozpoznávacích algoritmů.

Balík GUI

Balík GUI obsahuje kompletní uživatelské rozhraní aplikace.

Balík Preprocessing

Předposledním logickým balíkem je balík „Preprocessing“ obsahující implementace všech segmentačních algoritmů v balíku „Segmentation“ a implementace algoritmů pro vyrovnání obrázků dokumentů v balíku *SkewCorrection*.

Balík Postprocessing

Zůstává už jen balík *Postprocessing* obsahující algoritmy pro závěrečné úpravy zpracovaného dokumentu.

8.4 Vybrané implementační detaily

Na závěr popisu implementace ukažme některé ze zajímavějších částí samotné aplikace.

8.4.1 Datový model

Jednou z prvních výzev při postupném zpracovávání dokumentu byla volba datového modelu pro jeho uložení. Bylo potřeba stanovit, jaké dokumenty vlastně očekáváme a co vše o nich budeme chtít uložit.

Už od počátku byla zvažována stromová (případně grafová) struktura. Nějakou dobu byla zvažována reprezentace rozkladu dokumentu pomocí dvou dimenzionálního k–d stromu, nakonec se ale ukázalo, že by to bylo zbytečně omezující řešení a byla tedy zvolena vlastní stromová struktura.

Výslednou reprezentací je tedy dvojice stromů (pro textovou vrstvu a ostatní části), kde platí:

- Každý uzel je ve stromu jedinečný svou pozicí ve vstupním obrázku.
- Mimo pozice každý uzel nese informaci o svých rozměrech (obdélník) a svém rodiči.
- Každý uzel může obsahovat libovolné množství potomků, ti se však nesmí vzájemně překrývat.
- Každý uzel s výjimkou kořene má právě jednoho rodiče. Kořen nemá rodiče.
- Má–li uzel rodiče, pak je celý obsažen v jeho obdélníku.
- Oba kořeny stromů obsahují informaci o kořenu druhého stromu.

Význam těchto omezení je následující:

- Každá oblast dokumentu je ve stromu dokumentu (textové vrstvy) nejvýše jednou.
- Každá oblast ví, kde přesně se nachází, jakou oblast pokrývá a zná nadřazenou oblast v dokumentu (textové vrstvě).
- Každá oblast může být rozdělena dále, její podoblasti ovšem musí být disjunktní. Nelze tedy říct, že např. jedno písmeno patří do dvou slov v řádku, nebo že jedno slovo patří do dvou řádků v textu.
- Každá oblast patří právě do jedné oblasti. Nelze tedy říct, že např. jedno slovo patří do dvou různých textových oblastí, nebo že jedno písmeno patří do dvou různých slov. Kořen reprezentuje největší oblast – samotný dokument (textovou vrstvu celého dokumentu).
- Dokument zná svou textovou vrstvu a textová vrstva ví, ke kterému dokumentu patří.

Takto zavedený model dokáže popsat všechny oblasti dokumentu i jejich vzájemné vztahy. Výhodou je, že jej do budoucna lze snadno modifikovat pro oblasti s libovolným tvarem oblastí. Stačí nadefinovat predikáty pro průnik oblastí, náležení bodu do oblasti a jejich sjednocení. V takovém případě je ale nutné dávat pozor na složitost implementace takových predikátů. Byly by totiž hojně volány při navigaci v dokumentu a mohly by tedy ovlivnit odezvu aplikace.

8.4.2 Jazykové mutace

Aplikace je plně připravena k překladu do libovolného jazyku. Aktuálně jsou dostupné dva překlady: český a anglický. Uživatelé také mají možnost vytvářet vlastní překlady.

Práce s jazyky je založena na třídě *java.util.ResourceBundle*, tedy v podstatě mapě klíčů na hodnoty s širokou podporou pro práci s *java.util.Locale* – informacemi o specifickém „geografickém, kulturním či politickém regionu“ (viz [34]).

Pro přidání nového jazyku je potřeba udělat tři věci:

- Nejprve přeložit všechny záznamy z *LabelsBundle_x_y.properties*, *StringsBundle_x_y.properties* a *HelpBundle_x_y.properties* v adresáři *lang*, kde *x_y* je kód jazyka, ze kterého překládáme.
- Dále pro získané hodnoty vytvořit odpovídající *LabelsBundle_a_b.properties*, *StringsBundle_a_b.properties* a *HelpBundle_a_b.properties*, kde *a_b* je kód jazyka, do kterého překládáme.
- Nakonec o novém jazyce vytvořit záznam v souboru *supported_locales* tamtéž. Záznam vytvoříme přidáním řádku s kódem nového jazyka.

Kód jazyka musí být uveden dle standardu IETF BCP 47. V názvu souborů použijeme jako oddělovač jednotlivých informací podtržítko, v záznamech v *supported_locales* pak pomlčku.

Pro jednotlivé soubory s bundly pak musí být použito kódování ISO-8859-1. K docílení tohoto požadavku lze použít například utilitu *native2ascii* [35].

9 Budoucí vývoj

Pro budoucí vývoj aplikace existuje mnoho možností. Nejprve je důležité zvýšit úspěšnost stávajících algoritmů (nebo přidat nové) pro všechny druhy rukopisu. To znamená věnovat se jak samotné fázi rozpoznávání, tak zejména fázi segmentace slov na písmena, neboť tato se ukázala být největší výzvou při rozpoznávání ručně psaného textu. To ostatně dokládá i pozornost, která je těmto algoritmům dnes věnována (viz například [17]).

V oblasti dalších algoritmů bych se rád blíže podíval na některé z těch modernějších, jako jsou algoritmy využívající HOG (z anglického Histogram of Oriented Gradients) reprezentaci vlastností objektů, různé konvoluční sítě, které lze považovat za rozvedení a zdokonalení myšlenek za neocognitronem (např. LeNet) či dnes velice moderní „deep learning“ metody.

Vzhledem k diskuzi u metodiky testování v části o výsledcích (6.3.1) by dalším přístupem slibujícím zajímavé výsledky mohlo být přidání dalšího propojení mezi jednotlivé fáze určování znaků, podobně jako je tomu mezi algoritmem segmentace slova na znaky a algoritmy určování samotných znaků. Příkladem by mohlo být použití určování znaku s informací o okolních znacích. Například s přihlédnutím ke slovníku, statistickým vlastnostem jazyka, nebo nějakému kontextovému analyzátoru.

Také bych se rád věnoval zvýšení úrovně automatizace, kde bych jednou rád dosáhl dávkového zpracování mnoha souborů. Pak by mělo smysl uvažovat i o vytvoření serverové verze aplikace. První fází v tomto směru je odstranění omezení na vstup popsaných v části 2.1.

Zamyslíme–li se nad obsahem dokumentů, pak by nejvyšší metou bylo asi zpracování a následný export matematických výrazů.

Posledním zajímavým směrem vývoje, který zde zmíníme, je využití možností dnešních víceprocesorových (vícejádrových) počítačů a co nejvíce výpočtů paralelizovat. To by přineslo možnost práce s většími vzorovými množinami bez zvýšení nároků na čas uživatele.

Závěr

V práci byl krok za krokem popsán proces strojového zpracování ručně psaných dokumentů. Mimo jiné obsahuje popis a implementaci tří algoritmů pro rozpoznávání znaků založených na různých přístupech, z nichž jeden je vlastního návrhu. Dále pak několik algoritmů pro segmentaci dokumentu z původního obrázku až na úroveň znaků a obrázků (diagramů), včetně možnosti zpracovat základní typ tabulek. Pro výsledek je naimplementována možnost exportu do HTML a LaTeXu. V případě hůlkového písma a dokumentu členěného s ohledem na strojové zpracování je doprovodná aplikace schopna dosáhnout téměř 90% úspěšnosti. Aplikace je rovněž plně internacionalizována. Aktuálně existuje český a anglický překlad.

Za největší přínos práce považuji právě výše zmíněný algoritmus VHE, který je v současné verzi reálně použitelný pro rozpoznávání jednotlivých písmen. Užitečná je i samotná aplikace, která umožňuje poměrně velkou míru interakce mezi uživatelem a procesem rozpoznávání. Zároveň je připravena na přidání nových algoritmů v libovolné fázi rozpoznávání. Plánuji tedy její další vývoj. Za zajímavý považuji i doprovodný text, který přináší ucelený pohled na nejdůležitější fáze problému zpracování dokumentů s odkazy na mnohé užitečné zdroje.

Conclusions

The process of handwritten document segmentation was described in the thesis. Three character recognition algorithms working with different ideas are described and implemented together with several other algorithms useful during document segmentation. One of them was designed by author of the thesis.

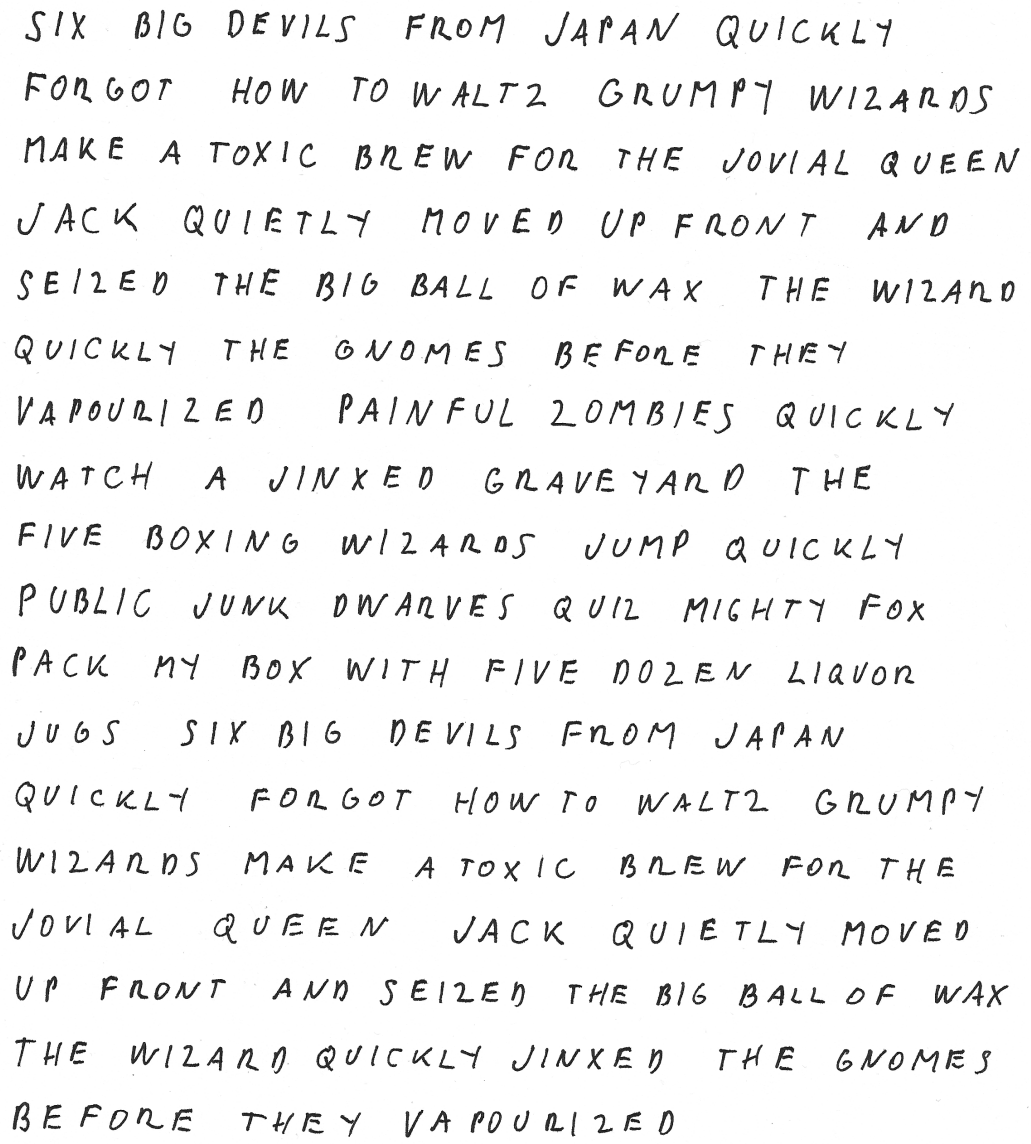
The application is able to detect and process logical document parts such as basic tables, images and text areas. The result can be exported to LaTeX and HTML formats. For the case of handprinted writing style and reasonable document layout, the application is able to recognize up to 90% of characters correctly. The application is also fully internationalized. Czech and English translations are included.

I consider the VHE algorithm to be the most interesting result of the thesis. Its proposed version can be used with satisfactory results.

The application itself is also useful as it allows quiet high level of interaction between the user and the recognition process. The author would like to continue with its development.

A Příklady testovacích dokumentů

Tato příloha obsahuje některé z dokumentů použitých k učení a testování algoritmů v přiložené aplikaci.



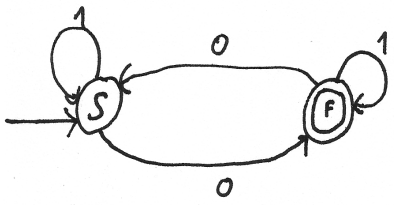
Obrázek 22: Příklad dokumentu se vzory (velká tiskací písmena) napsaného autorem práce.

QUICK BROWN FOX JUMPS OVER THE LAZY DOG
quick brown fox jumps over the lazy dog

THE FIVE BOXING
WIZARDS JUMPS
QUICKLY
the five boxing
wizards jumps
quickly

COLUMN	column
DATA	data
TEXT	text
EMPTY	empty
LAST	last

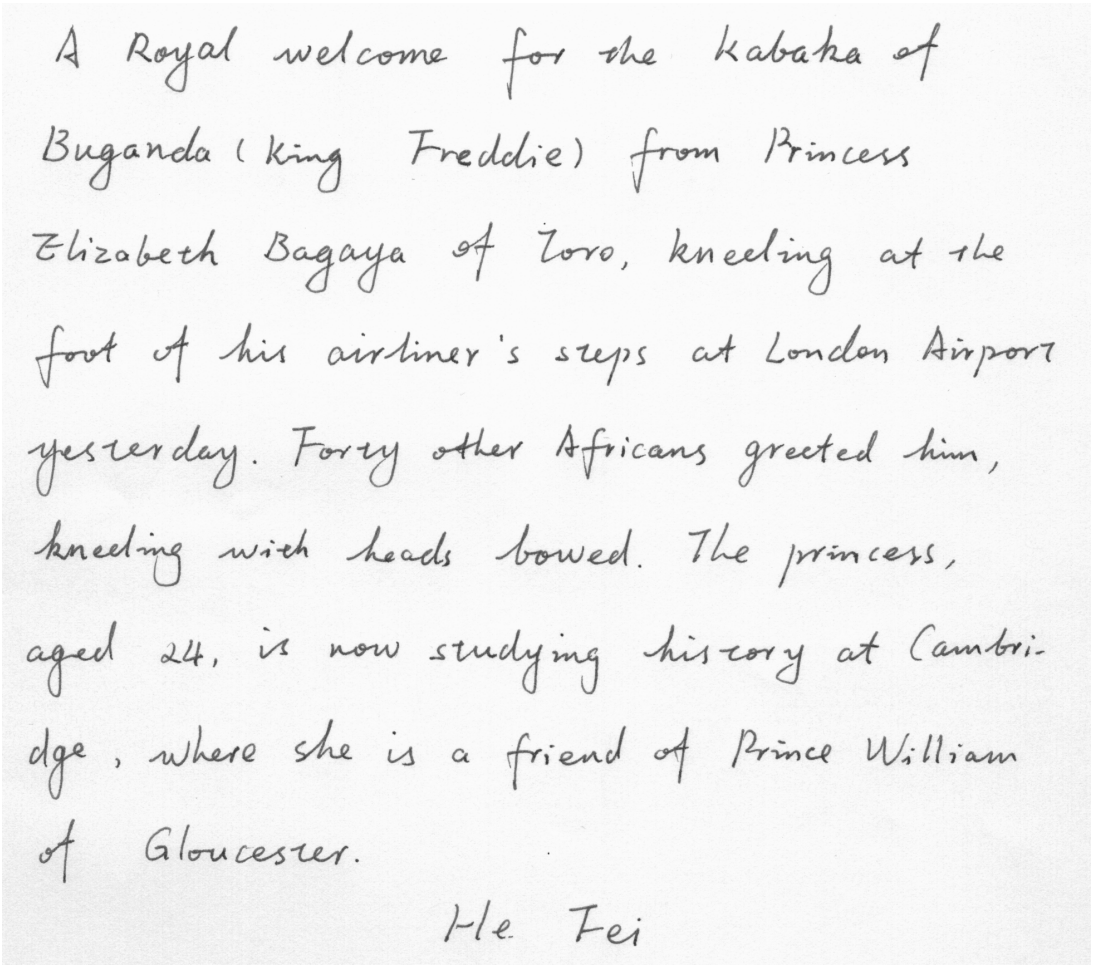
JUMPY HALFLING DWARVES PICK QUARTZ
BOX jumpy halfling dwarwes pick quartz
box



FIX PROBLEM QUICKLY
WITH GALVANIZED JETS

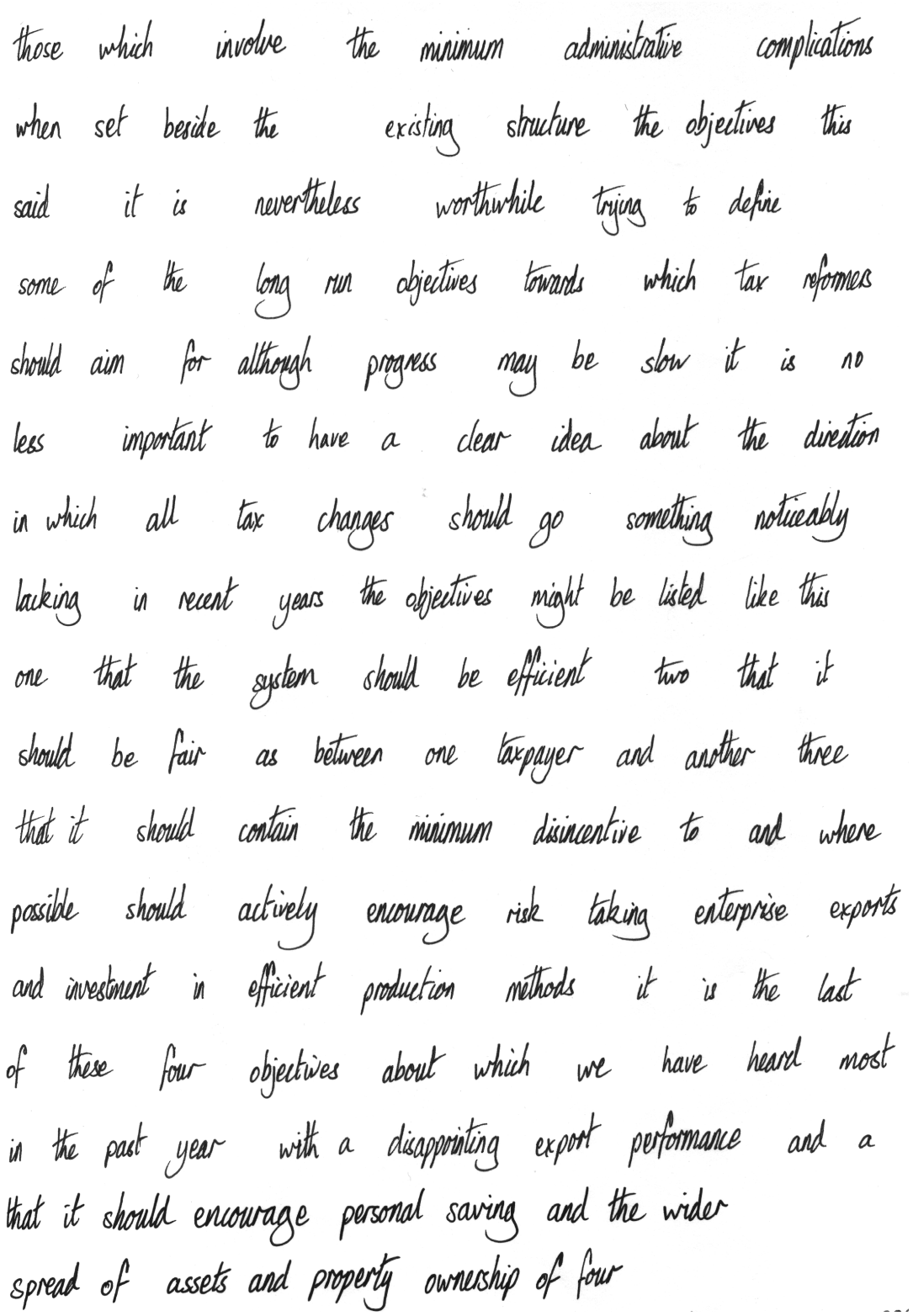
fix problem quickly with
galvanized jets

Obrázek 23: Příklad testovacího dokumentu napsaného autorem práce.


A Royal welcome for the Kabaka of Buganda (King Freddie) from Princess Elizabeth Bagaya of Toro, kneeling at the foot of his airliner's steps at London Airport yesterday. Forty other Africans greeted him, kneeling with heads bowed. The princess, aged 24, is now studying history at Cambridge, where she is a friend of Prince William of Gloucester.

He Fei

Obrázek 24: Příklad testovacího dokumentu z IAM Handwriting database [13].



Obrázek 25: Příklad testovacího dokumentu z databáze Andrew Seniora [29].

B Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD.

bin/
: Adresář *bin* obsahuje spustitelný jar archiv aplikace spolu se všemi potřebnými daty k jejímu běhu. Data použitelná v aplikaci lze najít v podadresáři *test*. Přesněji *test/img* obsahuje různé dokumenty ke zpracování a *test/work* obsahuje některé aplikací již zpracované a uložené dokumenty.

  Pro práci s aplikací je nutné tento adresář nakopírovat na disk, na který můžeme zapisovat.

doc/
: Adresář *doc* obsahuje tento text práce spolu s jeho zdrojovými soubory.

src/
: Adresář *src* obsahuje kompletní zdrojové kódy doprovodné aplikace.

Literatura

[1] MORI, S.; SUEN, C. Y.; YAMAMOTO, K. Historical Review of OCR Research and Development. *Proceedings of IEEE*. 1992, roč. 80.

[2] SEVARAC, Z.; CONTRIBUTORS. *NeurophOCR* [online]. [cit. 2016-05-08]. Dostupný z: ⟨https://sourceforge.net/projects/hwrecogntool/⟩.

[3] OTSU, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, roč. 9, č. 1, s. 62–66. ISSN 0018-9472.

[4] GREENSTED, A. *Otsu Thresholding Explained* [online]. [cit. 2016-05-03]. Dostupný z: ⟨www.labbookpages.co.uk/software/imgProc/otsuThreshold.html⟩.

[5] TARÁBEK, P. Morphology Image pre-processing for thinning algorithms. *Journal of Information, Control and Management Systems*. 2007, roč. 5.

[6] MAKKAR, N.; SINGH, S. A Brief tour to various Skew Detection and Correction Techniques. *International Joural for Sciecnce and Emerging Technologies with Latest Trends*. 2012, roč. 4, č. 1. ISSN 2250-3641.

[7] MARSDEN, K. *Skew Detection and Correction* [online]. [cit. 2016-05-05]. Dostupný z: ⟨http://http.cs.berkeley.edu/~fateman/kathey/skew.html⟩.

[8] BRESENHAM, J. E. Algorithm for computer control of a digital plotter. *IBM Systems Journal*. 1965, roč. 4, č. 1, s. 25–30. ISSN 0018-8670.

[9] PALÁGYI, K. *Skeletonization* [online]. [cit. 2016-05-05]. Dostupný z: ⟨http://www.inf.u-szeged.hu/~palagyi/skel/skel.html⟩.

[10] ZHANG, T. Y.; SUEN, C. Y. A Fast Parallel Algorithm for Thinning Digital Patterns. *Communications of the ACM*. 1984, roč. 27, č. 3, s. 236–239. ISSN 0001-0782.

[11] VANDEVENNE, Lode. *Flood fill* [online]. [cit. 2016-05-06]. Dostupný z: ⟨http://lodev.org/cgtutor/floodfill.html⟩.

[12] ČERNÝ, M. *Rozpoznávání registračních značek motorových vozidel*. 2010.

[13] MARTI, U.; BUNKE, H. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition. *Int. Journal on Document Analysis and Recognition*. 2002, roč. 5, s. 39–46.

[14] CASEY, R. G.; LECOLINET, E. A survey of methods and strategies in character segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1996, roč. 18, č. 7. ISSN 0162-8828.

[15] VEČERKA, A. *Grafy a grafové algortimy* [online]. 2007 [cit. 2016-05-10]. Dostupný z: ⟨http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf⟩.

[16] SALVI, D.; ZHOU, J.; WAGGONER, J.; WANG, S. Handwritten text segmentation using average longest path algorithm. In. *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*. 2013, s. 505–512. Dostupný také z: ⟨http://dx.doi.org/10.1109/WACV.2013.6475061⟩.

[17] REHMAN, A.; SABA, T. Off-line Cursive Script Recognition: Current Advances, Comparisons and Remaining Problems. *Artif. Intell. Rev.* 2012, roč. 37, č. 4, s. 261–288. Dostupný také z: ⟨http://dx.doi.org/10.1007/s10462-011-9229-7⟩. ISSN 0269-2821.

[18] GOMATHI ROHINI, S.; UMA DEVI, R. S.; MOHANAVEL, S. Character Segmentation for Cursive Handwritten Text Using Ligature Classification and Transition Feature. *Proceedings of the Fourth International Conference on Signal and Image Processing*. 2012.

[19] URBANEC, T. *Rozpoznávání ručně psaného textu pomocí fuzzy logiky*. 2013.

[20] NGUYEN, H. T.; WALKER, E. A. *A first course in fuzzy logic*. BOCA RATON: CHAPMAN & HALL, 2000.

[21] ROJAS, R. *Neural Networks: A Systematic Introduction*. New York, NY, USA: Springer-Verlag New York, Inc., 1996. ISBN 3-540-60505-3.

[22] KRUPKOVÁ, O. *Lineární algebra* [online]. 2008 [cit. 2016-05-10]. Dostupný z: ⟨http://phoenix.inf.upol.cz/esf/ucebni/Algebra.pdf⟩.

[23] FUKUSHIMA, K. Artificial Vision by Multi-layered Neural Networks: Neocognitron and Its Advances. *Neural Netw.* 2013, roč. 37, s. 103–119. Dostupný také z: ⟨http://dx.doi.org/10.1016/j.neunet.2012.09.016⟩. ISSN 0893-6080.

[24] KUKAČKA, M. Neocognitron: A Survey of a Classical Hybrid Neural Network Model. *WDS'11 Proceedings of Contributed Papers*. 2011, s. 112–118.

[25] FUKUSHIMA, K. Training Multi-layered Neural Network Neocognitron. *Neural Netw.* 2013, roč. 40, s. 18–31. Dostupný také z: ⟨http://dx.doi.org/10.1016/j.neunet.2013.01.001⟩. ISSN 0893-6080.

[26] FUKUSHIMA, K. Interpolating Vectors for Robust Pattern Recognition. *Neural Netw.* 2007, roč. 20, č. 8, s. 904–916. Dostupný také z: ⟨http://dx.doi.org/10.1016/j.neunet.2007.06.003⟩. ISSN 0893-6080.

[27] FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 1980, roč. 36, s. 193–202.

[28] MAHASHUKHON, P.; MOUSAVINEZHAD, H.; SONG, J. Y. Hand-printed English character recognition based on fuzzy theory. *IEEE International Conference on Electro/Information Technology (EIT)*. 2012.

[29] SENIOR, A. *Handwriting Database* [online]. [cit. 2016-05-15]. Dostupný z: ⟨ftp://svr-ftp.eng.cam.ac.uk/pub/data/handwriting_databases.README⟩.

[30] FUKUSHIMA, K.; WAKE, N. Handwritten alphanumeric character recognition by the neocognitron. *IEEE Transactions on Neural Networks*. 1991, roč. 2, č. 3, s. 355–365. Dostupný také z: ⟨http://dx.doi.org/10.1109/72.97912⟩. ISSN 1045-9227.

[31] CONN, N. J. Character Recognition Using a Neocognitron. [online]. [Cit. 2016-05-20]. Dostupný z: ⟨https://github.com/nicholasjconn/Neocognitron-Java/blob/master/Final-Paper-NJC-20120520.pdf⟩.

[32] *The Java Advanced Imaging Image I/O Tools API Core*. [online]. [cit. 2016-05-16]. Dostupný z: ⟨https://java.net/projects/jai-imageio-core⟩.

[33] NAVEH, B.; CONTRIBUTORS. *JGraphT – Java graph library* [online]. [cit. 2016-05-16]. Dostupný z: ⟨http://jgrapht.org/⟩.

[34] *Java 8 API*. [online]. [cit. 2016-05-16]. Dostupný z: ⟨https://docs.oracle.com/javase/8/docs/api/⟩.

[35] *Interactive native2ascii*. [online]. [cit. 2016-05-20]. Dostupný z: ⟨http://native2ascii.net/⟩.