VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

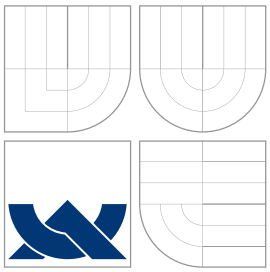
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVAČ PSANÉHO TEXTU PRO MOBILNÍ TELEFONY

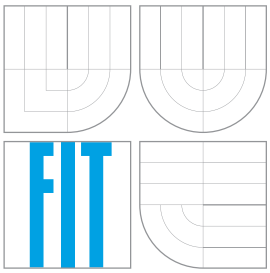
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE VLADIMÍR TALAŠ
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVAČ PSANÉHO TEXTU PRO MOBILNÍ TELEFONY

RECOGNITION OF HANDWRITING FOR MOBILE PHONES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — VLADIMÍR TALAŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR SCHWARZ
SUPERVISOR

BRNO 2007

Abstrakt

Cílem projektu bylo vytvoření aplikace pro mobilní telefon která by umožnila pomocí zabudovaného fotoaparátu telefonu vyfotografovat snímek v němž by nalezla a rozpoznala text. Tento text by následně bylo možno odeslat v textové zprávě. Aplikace je založena na implementaci algoritmů pro rozpoznávání textu z fotografií. Zejména se bude jednat o metody založené na skrytých Markovových modelech. Důraz je kladen na trénování modelu s cílem maximalizovat úspěšnost při rozpoznávání textu. Jsou prováděny experimenty s parametry modelu, díky čemuž se podařilo dosáhnout více jak 97% úspěšnosti při rozpoznávání textu

Klíčová slova

OCR, Rozpoznávání textu, skryté Markovovy modely, mobilní telefon, java, J2ME, Symbian

Abstract

The goal of this project is to create a mobile phone application, which can use phone camera to get a photography. This photography contains text, application has an ability to find a text, recognize all characters and send output as SMS. In this application there are implemented algorithms for text recognize from pictures based on Hidden Markovov Models. The particular stress is put on training of the model, to maximalise a succes of text recognition. There are some experiments model training with model variables, which are leading in better ability of text recognition. It was achieved a value of 97% succesfully recognized characters.

Keywords

OCR, text recognition, Hidden Markovov Models, mobile phone, java, J2ME, Symbian

Citace

Vladimír Talaš: Rozpoznávač psaného textu pro mobilní telefony, diplomová práce, Brno, FIT VUT v Brně, 2007

Rozpoznávač psaného textu pro mobilní telefony

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Schwarze

.....
Vladimír Talaš
22. května 2007

Poděkování

Děkuji Ing. Petru Schwarzovi za vedení této práce, Ing. Janu Tichému za poznatky a zkušenosti z jeho projektu, ale hlavně bych chtěl poděkovat mým rodičům za podporu, díky které jsem mohl studovat na tomto ústavu a vypracovat tuto diplomovou práci.

Obsah

Kapitola 1

Úvod

Optické rozpoznávání textu (Optical Character Recognition, OCR) je metoda která umožňuje převod digitálních dat obsahující text na prostý text. I přes dnešní výpočetní sílu počítačů a kvalitu digitálních optických snímačů je stále problematické vytvořit aplikace, která by se tohoto úkolu zhostila se stoprocentní úspěšností. Mezi příčiny tohoto nezdaru můžeme započítat velké množství fontů, rušivé jevy jako např. šum při snímání, různé deformace a poškození snímaného materiálu apod. V projektu je použit přístup založen na skrytých Markovových modelech (Hidden Markov models, HMM). Model, podle něhož se pak provádí samotné rozpoznávání, je potřeba natrénovat. Proces trénování a výsledný model má tedy zásadní vliv na úspěšnost rozpoznávání, provedeme proto několik málo analýz s cílem vytvořit optimální model. Tento projekt se zabývá studiem jak již existujících algoritmů pro rozpoznávání textu tak se pokusí přinést i nové algoritmy s ohledem na použití v mobilních telefonech.

Celý dokument můžeme rozdělit do dvou tematických celků. První, teoretická část přináší osvětlení použitých technologií a postupů. Najdeme zde jednotlivé fáze rozpoznávání textu jako je korekce vstupních dat, segmentace, parametrizace obrazu a následné rozpoznávání. Je zde zmíněn i přístup založen na neuronových sítích avšak důraz je kladen na metodu rozpoznávání založenou na skrytých Markovových modelech. Dále jsou zde popsány možné prostředky použitelné pro vývoj projektu a s tím související nároky na hardware (mobilní telefon).

Druhá část se zabývá implementací projektu a aplikací různých přístupů pro předzpracování obrazu, nalezení oblastí textu či segmentací řádků textu. Velký důraz je kladen na trénování modelu. Najdeme zde výstupy z mnoha experimentů prováděných za účelem vytvořit model s co největší schopností rozpoznávat text.

Jako samostatná část pak vystupuje závěr, kde jsou zhodnoceny dosažené výsledky, prodiskutovány nedostatky a zváženy možnosti dalšího postupu při tvorbě projektu.

Kapitola 2

Teorie optického rozpoznávání textu

Prioritou metody pro rozpoznávání textu je její účinnost, tzn. míra shody textu předlohy a výsledného textu. Tato shoda patrně nikdy nebude stoprocentní, avšak zvolením vhodného postupu při zpracování vstupních dat a rozpoznávacího algoritmu se však k této hodnotě můžeme velmi blízce přiblížit. Nejdříve je nutné vstupní data upravit tak, aby bylo co nejvíce zřetelné, které části snímku obsahují text. Tyto části je posléze potřeba rozdělit na menší části jako jsou řádky, slova či znaky a parametrizovat, tj. převést jasovou reprezentaci obrazu do fekvenční oblasti. Takto upravená data jsou pak vstupem pro samotný rozpoznávací algoritmus jehož výstupem je výsledný text. Proces rozpoznávání textu tedy můžeme rozdělit do těchto částí:

2.1 Předzpracování

Výsledná úspěšnost úzce souvisí s kvalitou vstupních dat. Aby úspěšnost rozpoznávání byla co možná nejvyšší je nutné vstupní data upravit. Tato fáze je velmi důležitá, vezmeme-li v potaz určení výsledné aplikace pro mobilní telefony, kde stále není kvalita snímacího zařízení optimální.

2.1.1 Korekce vstupních dat

Jedná se hlavně o eliminaci jevů způsobených optickým snímáním předlohy. Jako hlavní problémy se jeví osvětlení snímku a úhel snímání. Musíme počítat s tím, že snímání vstupních dat neprobíhá za konstantních světelných podmínek. Výsledný obraz může obsahovat světlejší a tmavší místa, proto není zřejmé, která místa ztmavit a která naopak zesvětlit. Za cíl při této úpravě si stanovíme konstantní osvětlení v celém rozsahu snímku. Další krokem ke zvýšení úspěšnosti rozpoznávacího algoritmu je zjištění pod jakým úhlem byl snímek pořízen. Stránku můžeme otočit tak, aby řádky textu byly vodorovně. Tento krok však není nezbytně nutný, použijeme-li posléze metody, které umí rozpoznávat text nezávisle na úhlu zkosení.

Segmentace

Cílem je odlišit objekty v obraze od jejich pozadí, je to vlastně funkce která upravuje barevné či jasové složky pixelů obrazu. Výsledkem je, že všechny body takto upraveného obrazu budou mít hodnotu 0 nebo 1. Práh (treshhold) je hodnota, podle které se rozhodne, zda pixel bude považován za pozadí (bude mít hodnotu 0) nebo zda bude považován za objekt (v našem případě text). Podle způsobů zvolení prahu můžeme dále metody prahování rozlišit [7, 4]. Budeme se zabývat metodami, které umí zvolit hodnotu prahu automaticky v závislosti na vstupním obraze.

- Konstantní hodnota prahu: pevnou hodnotu prahu můžeme určit jako aritmetický průměr či mediánu bodů obrazu. Jako zdrojová data není třeba použít celý obraz, stačí jen jeho část nebo dostatečné množství náhodně vybraných vzorků. Nevýhoda této metody je v tom, že je velmi závislá na kvalitě vstupních dat (snímaného obrazu), problémy by mohly nastat při výskytu šumu či nedostatečnému kontrastu textu vůči pozadí.

- Proměnlivá hodnota prahu: než o specifickou metodu se spíše jedná o vylepšení metody konstantního prahu. Jde o to, že vstupní obraz se rozdělí do několika částí a hodnota prahu se počítá pro každou část samostatně. V případě, že rozdíl mezi maximální a minimální hodnotou jasu je příliš malý, může se práh pro tuto oblast vypočítat jako průměr z okolních oblastí. Tato metoda eliminuje rozdílné osvětlení jednotlivých částí obrazu, nevýhodou je vyšší výpočetní náročnost.

- Metody automatického určení optimálního prahu: je to např metoda nalezení prahu podle Nobuyuki Otsu nebo nalezení prahu na základě entropie histogramu[7]. Vycházejí z relativního histogramu a jsou založeny na statistických výpočtech. Podle [7] se jedná o velice kvalitní metody, při jejich použití je dosahováno velmi dobrých výsledků, leč nevýhodou zůstává jejich matematická a s tím související výpočetní složitost.

Další možností jak obraz segmentovat je použití normalizace. Je to metoda kdy se hodnoty jasu mezi maximální a minimální hodnotou rozdělí do 255 stupňů šedi. Bylo již ovšem zjištěno [5], že použití této metody není výhodnější (nijak se nezvýšila úspěšnost rozpoznávání textu), navíc je výpočetně náročnější. Touto metodou se tedy dále již zabývat nebudeme.

2.1.2 Nalezení jednotlivých řádků

Jako jedna z ideálních metod pro rozlišení řádků se ukazuje metoda za použití obalových čar [5]. Její princip, jak už je z názvu patrné, použití čar obalujících jednotlivé řádky textu těsně kolem všech znaků. Kouzlo metody spočívá její jednoduchosti a účinnosti. Následné rozpoznávání je totiž prováděno pouze nad prostorem vymezeným obalovými čarami, zároveň metoda eliminuje vliv sklonu jednotlivých řádků.

Postup nalezení řádků je následující: v prvním kroku je vypočten horizontální/vertikální vektor změn v obraze. Každý prvek vektoru je součtem změn v sloupci/řádku obrazu. Změnou se rozumí přechod barvy pixelu z černé do bílé a naopak. Vypočtená střední

hodnota nám určuje lokální minima, dvě po sobě jdoucí minima nám určuje přibližnou polohu linek textu, postačí souřadnice levého horního a pravého dolního rohu. V další fázi za pomocí obalových čar přesně vymezíme jednotlivé řádky textu.

2.1.3 Parametrizace

Proces parametrizace se také nazývá jako extrakce příznakových vektorů a jeho cílem je, aby byly dostatečně popsány a vzájemně odlišeny jednotlivé části vstupního signálu, zároveň by měl redukovat datový tok vstupující do dekodéru (vektor by měl obsahovat dostatečný ale co nejmenší počet parametrů). Proces parametrizace (analýzy signálu) by neměl být příliš výpočetně náročný. Při zpracování řeči to může být např. prostředí, ve kterém byla řeč snímána, projev mluvčího atd. Při rozpoznávání textu má především vliv kvalita snímku, typ fontu, světelné podmínky při snímání obrázku (stejnoměrné nasvícení, ostrost, existence šumu apod.). Výsledkem parametrizace je maticová reprezentace vstupních dat, která je pak zpracována pomocí HMM. Je zřejmé, že parametrizace se provádí jak ve fázi trénování dat, tak ve fázi rozpoznávání. Pro obě fáze se musí jednat o stejný algoritmus.

Budeme používat transformaci z oblasti časové do frekvenční. Obraz sice není funkcí času, leč v této situaci se dá považovat za dvourozměrný vzorkovaný signál. Do této kategorie transformací patří DFT (Diskrétní Fourierova transformace), DCT (Diskrétní Kosinová transformace) nebo KLT (Karhunen-Loéveho transformace). DCT je odvozena od DFT, narozdíl od DFT produkuje pouze reálné koeficienty. Obecně DFT produkuje komplexní matici frekvenčních koeficientů. Jádro má 2 složky - reálnou a imaginární část: cos + i*sin. Pokud je vstupní funkce sudá, tak při výpočtu vychází nulový imaginární člen i*sin. Tím pádem výstupem jsou jen reálné koeficienty a jedná se o DCT. Obrazová data můžeme bez problémů považovat za funkci sudou. Např. průběh jasových složek jednoho řádku si můžeme představit jako funkci s počátkem v 0, tato funkce je souměrná podle osy y a tedy sudá [8].

Na výběr máme z několika verzí DCT transformace (existuje 8 variant DCT, 4 z nich jsou nejpoužívanější), budu používat DCT typu II, pravděpodobně nejrozšířenější forma, často uváděna jen jako „DCT“ 2.1. Výpočet se provádí přes všechny sloupce vstupního obrázku. Později si uvedeme optimalizace, pomocí nichž se DCT nepočítá přes celou výšku obrazu ale jen z definovaných částí. Tyto optimalizace mají kladný vliv na kvalitu parametrizace obrazu (zvýší se úspěšnost rozpoznávání znaků).

$$X_k = \sum_{n=0}^{N-1} x_n \cos[\frac{\pi}{N}(n+\frac{1}{2})k] \quad k = 0, ...N-1 \tag{2.1}$$

2D DCT transformace, označovaná také jako transformace vícerozměrné funkce je vlastně série jednorozměrných DCT transformací provedených postupně v každém rozměru. Výpočet by se prováděl z širších oblastí než je pouze sloupec obrázku, tím pádem by se z obrazu dalo vyextrahovat širší spektrum informace (získáme vzájemný vztah mezi více pixely najednou). Také JPEG komprese využívá 2D DCT transformaci z toho důvodu, že 2D DCT vytváří frekvenční mapu, tzn. že v oblasti, nad kterou je transformace prováděna soustřeďuje vysoké frekvence do pravého dolního

rohu, nízké frekvence do levého horního rohu. Při JPEG kompresi se využívá toho, že lidské oko je méně citlivé na funkce vysokých kmitočtů a proto je možné tyto frekvence odstranit. Pro naše účely jsou naopak tyto vysoké frekvence důležité, udávají totiž rychlé přechody barev a tudíž možné výskyty textu. Pro další výpočet z matice DC koeficientů vhodným způsobem vybereme jen ty, které pro popis textu budou nejvhodnější.

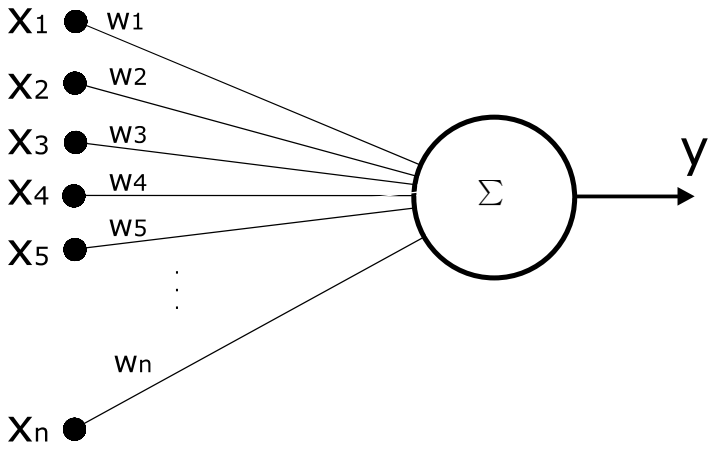
$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos[\frac{\pi}{N_1}(n_2 + \frac{1}{2})k_1] \ \cos[\frac{\pi}{N_2}(n_2 + \frac{1}{2})k_2] \tag{2.2}$$

2.2 Rozpoznávání

Proces rozpoznávání zjišťuje relaci nebo také míru shody parametrizovaného vstupního řádku textu se sekvencí základních symbolů. Základní symboly mohou být jednotlivá písmena, sekvence písmen či celá slova. Tuto skupinu označíme jako model, ten vznikne procesem trénování. Existují zde ale některé problémy. Záleží na typu fontu vstupního rozpoznávaného slova. Budeme me se zabývat dvěma v dnešní době nejpoužívanějšími přístupy pro rozpoznávání textu a to metodou s použitím neuronových sítí a metodou založenou na skrytých Markovových modelech.

2.2.1 Neuronové sítě

Metoda založená na neuronových sítích je založena na množství vzájemně propojených jednoduchých elementů, neuronů. Jejich matematický popis vychází ze z biologie a skutečných neuronů jako základních kamenů složitějších nervových systémů. Funkce spočívá v přenosu, zpracovávání a uchovávání informace. Formální popis neuronu byl navržen v roce 1943 Walterem Pitsem a Warenem McCullochem. Každá ta-



Obrázek 2.1: schema formálního neuronu

ková jednotka provádí pouze jednoduchou operaci, neuron má několik vstupů a pouze

jeden výstup a hodnotu označovanou jako potenciál neuronu (ξ). Vstupy x_1 až x_n jsou ohodnoceny váhami ω_1 až ω_n. [1] Potenciál každého neuronu se získá odečtením prahové hodnoty υ od váženého součtu svých vstupů.

$$\xi = \sum_{i=1}^{n} \omega_i x_i + \upsilon \tag{2.3}$$

Výstup z neuronu určuje jeho přenosovou funkci.

$$y = f(\xi) = f\ \sum_{i=1}^{n} \omega_i x_i + \upsilon \tag{2.4}$$

Přenosové funkce můžeme rozdělit do 3 hlavních skupin:

$$f(\xi) = 1 \quad \text{pro } \xi > 0, \quad 0 \quad \text{jinak} \tag{2.5}$$

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \tag{2.6}$$

$$f(\xi) = \xi \tag{2.7}$$

Rovnice 2.5 popisuje skokovou přenosovou funkce, která provádí velmi jednoduché prahování vstupních hodnot, nemá příliš dobré vlastnosti a proto je vhodnější spíše pro jednoduchý typ neuronu. Logická sigmoida popsaná rovnicí 2.6 je asi nejpoužívanější funkce. Hodnoty této funkce se pohybují v intervalu $\langle 0, 1 \rangle$, je spojitá, hladká, nelineární, jedná se v podstatě o vyhlazení skokové funkce v okolí bodu 0. Její derivace se dá vyjádřit pomocí vztahu

$$f'(x) = f(x)(1 - f(x)) \tag{2.8}$$

Poslední funkce definovaná vztahem 2.7 nechává potenciál neuronu na původní hodnotě a nazývá se identita.

Topologie sítí

Síť tvoří několik vzájemně propojených prvků. Obecnou Neuronovou síť je možndé definovat jako orientovaný graf v němž jednotlivé neurony tvoří uzly a orientované hrany tvoří vazby mezi nimi. Každá hrana má přiřazenou váhu ω. Rovněž definujeme neprázdnou množinu vstupních a výstupních neuronů. Skupiny neuronů se seskupují do vrstev, vstupní množina tvoří vstupní vrstvu, výstupní tvoří výstupní vrstvu, ostatní neurony jsou ve vrstvě (či vrstvách), které jsou mezi vstupní a výstupní vrstvou a nazývá se skrytá. Výstupní vrstva nemá další následníky a čte výstupy z celé sítě, přechodová funkce těchto neuronů obvykle bývá logická sigmoida. Naopak neurony vstupní vrstvy nemají žádné předchůdce, mají identickou přenosovou funkci a prahovou hodnotu rovnu nule. Jejich úkolem je pouze přenést vstupní hodnoty do sítě. Podle orientace hran ve skrytých vrstvách rozlišujeme 2 základní topologie neuronových sítí. V dopředných sítích (*feed-forward neural network*) vedou hrany pouze z vrstvy a_i do a_{i+1}, v sítích se zpětnou vazbou (*feed-back networks*)mohou neurony komunikovat i s prvky z předchozí vrstvy.

[1] Spoje se také někdy označují jako synapse a váhy jako synaptické váhy

Trénování sítí

Principem sítě je aproximovat vstupní data k nějaké vstupní funkci. Nastavení koeficientů sítě ovlivňuje jaká bude shoda výstupu ze sítě s referenční funkcí. Koeficienty jsou vyjádřeny maticí a vlastně určují cestu mezi jednotlivými neurony. Hledání ideální cesty (koeficientů) se provádí ve fázi trénovaní. Před započetím trénování je rovněž potřeba nastavit některé startovací parametry (počáteční hodnoty synaptických váh, počet neuronů ve skrytých vrstvách, množství trénovacích dat).

Vstupem pro učení je množina trénovacích dat složena ze vstupních vektorů nebo dvojic vektorů - vstupní vektor a požadovaný výstupní vektor. Podle toho rozlišujeme *soutěživé* učení nebo *adaptivní* učení. Adaptivní učení funguje tak, že srovnává výstup ze sítě s referenční výstupní hodnotou. Vypočítá se chyba všech neuronů ve výstupní vrstvě i chyby neuronů ve skryté vrstvě. Následně se upraví váhy synapsí tak, aby byla minimalizována celková odchylka od požadované hodnoty.

Soutěživé učení nemá na výstupu k dispozici žádnou srovnávací veličinu k určení správnosti. Neuron ve výstupní vrstvě s nejvyšší hodnotou je považován za vítěze a tedy správný výsledek. Upravují se pouze váhy tohoto neuronu nebo neuronů v jeho blízkém okolí.

2.2.2 Skryté Markovovy modely (HMM)

Skryté Markovovy modely byly s úspěchem použity v systémech jako např. rozpoznávání a modelování lidské řeči, rozpoznávání obličeje v neposlední řadě také rozpoznávání písma jak strojového tak ručně psaného, včetně podpisů. Princip rozpoznávacích systémů založených na skrytých Markovových procesech spočívá v tom, že dokážeme rozdělit vstupní data na segmenty, které jsme schopni popsat nějakou konfigurací z konečného počtu konfigurací. Uvedeme příklad na rozpoznávání řeči. U lidského hlasu můžeme tento segment pojmenovat jako foném. Je to krátký zvukový úsek (cca 20ms) a je generován jednou z konečných konfigurací hlasového ústrojí. To, jak bude foném bude vypadat(resp. jak bude vypadat jeho spektrální charakteristika), závisí právě na konfiguraci hlasového ústrojí. Je nutné, aby konfigurací byl konečný počet, čehož lze dosáhnout procesem vektorové kvantizace. Soubor všech konfigurací tvoří konečnou abecedu spektrálních vzorů (natrénované modely dat). Při rozpoznávání rozdělíme na zvukový vstup na fonémy a zparametrizujeme. Výsledné spektrum nahradíme vzorem ze souboru všech konfigurací, kterému je nejblíže. Při rozpoznávání textu bude situace podobná, blíže je znázorněna na obrázku 2.2.

Formálně definujeme Markovův model **G** jako pětici:

$$G = (Q, V, A, B, \pi) \tag{2.9}$$

Kde:

- Q - specifikuje stavy modelu. I když jsou skutečné stavy skryté, ve většině případů existuje určitý vztah mezi stavy modelu a nějakou skutečnou událostí. Množina stavů Q je definována:

$$Q = \{q_1, q_2, ..., q_N\}, \quad |Q| = N \tag{2.10}$$

- V - udává počet symbolů pro každý stav modelu. Je to vlastně konečná abeceda výstupních symbolů modelovaného systému definována jako konečná množina

$$V = \{v_1, v_2, ..., v_M\}, \quad |V| = M \tag{2.11}$$

- A - je to matice přechodů mezi jednotlivými stavy. Její prvky určují s jakou pravděpodobností systém ve stavu q_i v čase t přejde do stavu q_j v čase $t + 1$.

$$A = \{a_{ij}\}, \quad a_{ij} = P[q_{t+1} = j | q_t = i], \quad \leq i, j \leq N \tag{2.12}$$

  Jedná se o skutečnou pravděpodobnost a proto platí, že součet všech pravděpodobností vystupujících z jednoho stavu musí být jedna.

$$\sum_j a_{ij} = 1$$

- B - rozložení pravděpodobnosti generovaných vzorů ve stavu j. Vytvářejí matici, jejíž prvky určují s jakou pravděpodobností je v jakémkoliv čase t generována m-tý prvek z konečné abecedy výstupních symbolů V, je-li systém ve stavu q_j.

- π - sloupcový vektor pravděpodobností počátečního stavu.

Markovovy slouží k modelování procesů generujících náhodné sekvence stavů (stochastické procesy). Přechody mezi jednotlivými stavy jsou ohodnoceny pravděpodobnostmi. Avšak z vnějšku nejsou tyto stavy přímo viditelné, lze pouze zjistit jejich popis pomocí pravděpodobnostních funkcí, proto název skryté Markovovy modely [9]. Je zde jistá podobnost s konečným automatem. Má taktéž pevně stanovený počet stavů, do kterých se může během výpočtu dostat a přechodová funkce je tvořena maticí přechodů.

Rozpoznávání

Formálně můžeme obecný proces rozpoznávání modelu definovat takto: vstupní zparametrizovaný řádek textu označíme jako sekvenci vektorů parametrů

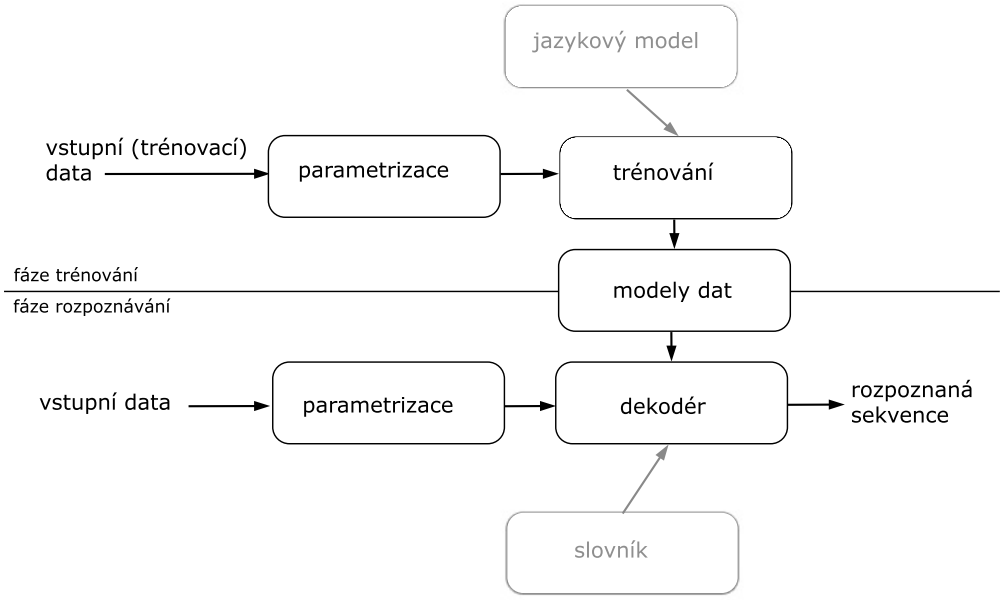
$$O \ = o_1, o_2, o_3, ...o_T \tag{2.13}$$

Pak tedy můžeme říci, že hledáme slovo ω_i s největší hodnotou pravděpodobnosti $P(\omega_i|O)$z Bayesova vztahu kde

$$P(\omega_i|O) = \frac{p(O|\omega_i)P(\omega_i)}{p(O)} \tag{2.14}$$

Pravděpodobnost $P(\omega_i|O)$ určuje, s jakou pravděpodobností odpovídá sekvence vektorů O na vstupu slovu ω_i. A právě nalezení parametrů skrytého Markovova modelu je vlastně odhad pravděpodobnosti $P(\omega_i|O)$. Budeme předpokládat, že $p(O)$ bude stejná pro všechny třídy a rovněž i $P(\omega_i) = \omega$ takže proces nalezení slova bude spočívat v nalezení maximální věrohodnosti (likehood) mezi vstupním vektorem a sekvencí základních symbolů [3].

$$\omega_i^* = max \ p(O|\omega_i) \tag{2.15}$$



Obrázek 2.2: model rozpoznávacího systému

Skrytý Markovův model

Na obrázku 2.3 je znázorněn jednoduchý Markovův model. Tento model v každém čase t přejde ze stavu q_i do stavu q_j s pravděpodobností a_{ij} přičemž vygeneruje vektor o_t podle rozložení $b_j(o_t)$ [3]. Stavy q_2 až q_5) zpracovávají vstupní vektor a označují se jako stavy vysílací (emitting), okrajové stavy q_1 a q_6) mají pouze funkci napojování na sousední modely. Součet pravděpodobností průchodu modelem přes všechny stavy $X = x_1, x_2...x_t$ určuje pravděpodobnost $P(O|M)$, že vektor O je generován modelem M.

$$P(O|M) = \sum_{X} a_{x(0)}a_{x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t)a_{x(t)}a_{x(t+1)} \tag{2.16}$$

Protože přímý výpočet z 2.16 je příliš náročný, lze použít zjednodušení uvedené ve vztahu 2.15 a to spočívá v nahrazení sumy maximální hodnotou.

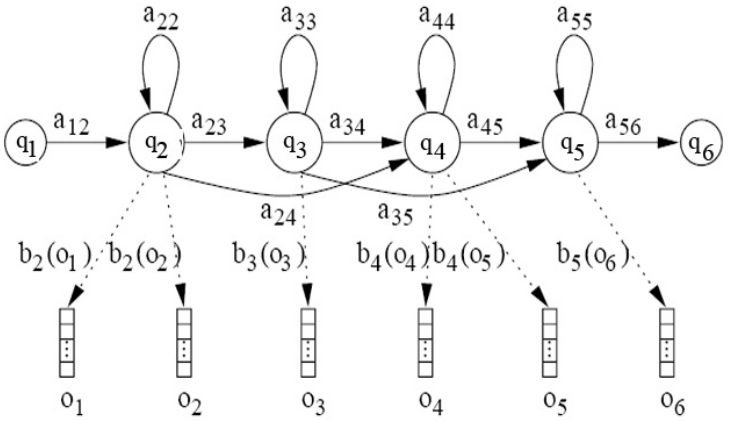
$$\hat{P}(O|M) = \max_{X}\{a_{x(0)}a_{x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t)a_{x(t)}a_{x(t+1)}\} \tag{2.17}$$

Viterbiho algoritmus

Máme k dispozici neznámou posloupnost O_i a sadu natrénovaných modelů M_i. Úkolem je zjistit maximální shodu mezi modelem a vstupním vektorem. To provedeme zjištěním *Viterbiho pravděpodobnosti* pro každý model a vybereme ten s nejvyšší pravděpodobností. Tento model odpovídá rozpoznané sekvenci (v našem případě se bude jednat o písmeno).

2.2.3 Trénování modelu

Při trénování jsou ke vstupním, obrazovým datům přidána rovněž i jejich textová reprezentace. Na obrázku 2.2 je rovněž znázorněn slovník a ortografická pravidla (pomocí ortografických pravidel je možné eliminovat některé nesmyslné kombinace znaků) rozpoznávaného jazyka. Nejsou to nutné součásti, při jejich použití trpí obecnost celého rozpoznávacího systému protože je nutné pro každý jazyk dodat vlastní slovník a ortografická pravidla. Na druhou stranu ovšem roste úspěšnost rozpoznávání (systém bude schopen opravit některé chyby ve slovech).



Obrázek 2.3: jednoduchý HMM

Schematicky je proces trénování zobrazen na obrázku 2.4. Hodnota funkce hustoty rozdělení pravděpodnosti může být definovaná jako suma hustot normálního rozdělení (Gaussovo rozložení), postačující je použití pouze jednoho Gaussova rozložení.

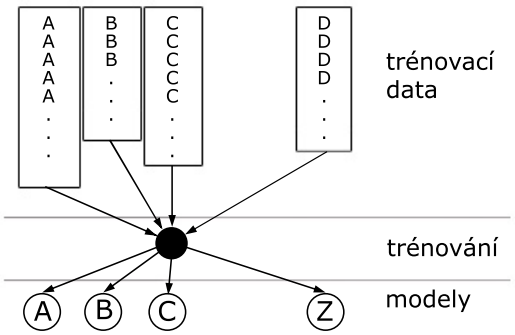
$$b_j[o(t)] = \aleph(o(t); \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^P |\Sigma|}} e^{-\frac{1}{2}(o(t)-\mu_j)\Sigma^{-1}(o(t)-\mu_j)} \tag{2.18}$$

Σ je diagonální kovarianční matice, k jejímu popisu postačí vektor směrodatných odchylek σ o velikosti P. μ udává střední hodnotu.

Samotný proces trénování je vlastně výpočet parametrů μ a σ. Nejprve se provede hrubý odhad parametrů, posléze jsou parametry upřesňovány. Prvotní nastavení je globální, tedy pro všechny stavy ale během trénování se parametry upřesňují pro každý model zvášť. Pro první odhad stačí použít průměrný hodnota:

$$\hat{\mu}_j = \frac{1}{T} \sum_{t=1}^{T} o_t \tag{2.19}$$

$$\hat{\Sigma}_j = \frac{1}{T} \sum_{t=1}^{T} (o_t - \mu_j)(o_t - \mu_j)' \tag{2.20}$$



Obrázek 2.4: proces trénování modelu

Baum-Welshova reestimace

Upřesňování parametrů probíhá prostřednictvím Baum-Welshova algoritmu. Obvykle se provádí několik iterací, jejich počet může být volen buď konstantní, nebo může být stanoven z výpočtů mezi jednotlivými iteracemi (sledují se rozdíly předchozího a nového modelu). Algoritmus se řídí podle tzv. Baum-Welshových vztahů:

$$\hat{\mu}_j = \frac{\sum_{t=1}^{T} L_j(t) o_t}{\sum_{t=1}^{T} L_j(t)} \tag{2.21}$$

a

$$\hat{\Sigma}_j = \frac{\sum_{t=1}^{T} (o_t - \mu_j)(o_t - \mu_j)'}{\sum_{t=1}^{T} L_{j(t)}} \tag{2.22}$$

$L_j(t)$ je pravděpodobnost, že se model v čase t nachází ve stavu j. Výpočet $L_j(t)$ se provádí rekurzivně pomocí tzv. dopředné a zpětné pravděpodobnosti či pomocí Viterbiho algoritmu, podle hodnoty se určují nové parametry modelu [6].

Kapitola 3

Vývoj aplikací pro mobilní telefony

Pro vývoj aplikací pro mobilní telefony můžeme v dnešní době využít dva patrně nejrozšířenější přístupy. Pokud má telefon operační systém (např. Palm OS, Symbian, Windows CE) lze pro takové zařízení za pomoci specifických vývojových nástrojů vytvořit aplikaci přeložitelnou kompilátorem jazyka C++. Přeložená aplikace pak běží v prostředí mobilního telefonu a přímo využívá výpočetní síly procesoru. Nevýhoda tohoto přístupu je poměrně značná nekompatibilita jak zařízení, tak i programů. Druhá možnost je použití programovacího jazyka Java, kdy stačí pouze to, aby na přístroji byl k dispozici interpret Javy, tzv. virtuální stroj Javy (JVM - Java Virtual Machine). Velká výhoda takto vytvořené aplikace spočívá v její přenositelnosti, protože běh virtuální stroje Javy je nezávislý na platformě a použité architektuře. Na druhou stranu Java je jazyk interpretovaný, tzn. že před startem programu je nutná kompilace, která se podepisuje na pomalejším startu aplikace.

3.1 Java pro mobilní telefony

3.1.1 Technologie

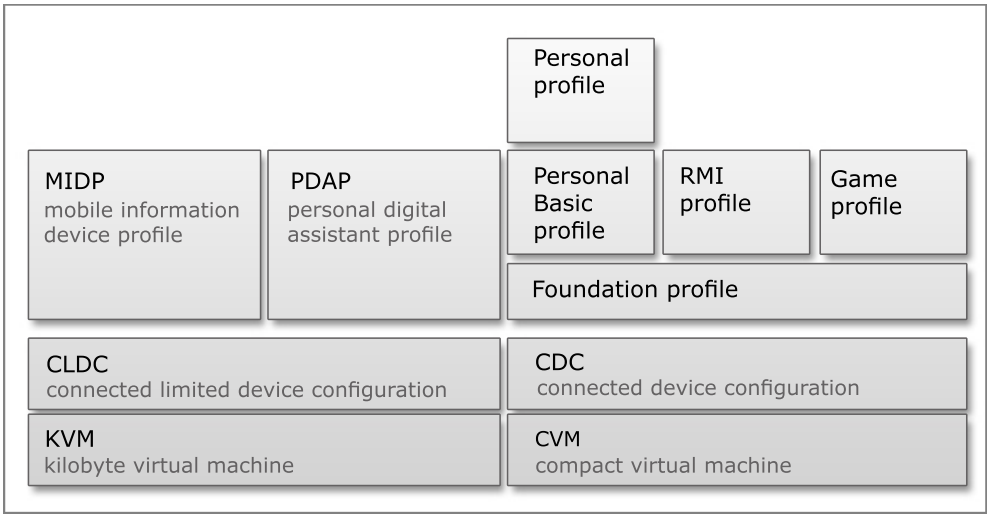
Java existuje v mnoha distribucích [1], pro vývoj aplikací pro mobilní telefony v se používá distribuce označovaná jako J2ME (Java 2 Microedition) [2]. Pomocí profilů a konfigurací můžeme základní J2ME balík v závislosti na typu koncového zařízení dále specifikovat. Konfigurace definují základní vlastnosti pro jednotlivá zařízení, podmnožiny programového vybavení pro danou úroveň jsou dány profily. Vše je patrné z obrázku 3.1.

Virtuální stroj: sestává ze interpreta jazyka Java a systému realizující vazbu na hardware. Tento systém má 251význam jako procesor a správce paměti.

Konfigurace: Zahrnují virtuální stroj definují programové vybavení pro danou skupinu zařízení. Jednotlivé skupiny zařízení jsou definována rychlostí a typem procesoru, velikostí paměti atd. V J2ME rozlišujeme 2 konfigurace:

- CLDC (Connected Limited Device Configuration) definuje nejmenší standardní konfiguraci Javy se zaměřením na zařízení s omezenými zdroji. Obsahuje virtuální

[1] Další edice jsou Java SE (Standard Edition) a Java EE (Enterprise Edition)



Obrázek 3.1: J2ME - architektura

stroj a minimální množinu tříd. Virtuální stroj pro pro CLDC mimo jiné verze CLDC 1.0 neobsahuje podporu plovoucí řádové čárky (ta je podporována až ve verzi 1.1) či je zde omezeno zpracování chyb. Konfigurace je určena pro zařízení disponující minimálně 128kB stálé paměti (ROM, flash) pro uložení virtuálního stroje, alespoň 32kB paměti RAM použitelné za běhu programu a 16-ti či 32 bitový procesor s minimální taktovací frekvencí 25MHz. CLDC konfiguraci dále upřesňují profily MIDP a PDAP.

  - MIDP(Mobile Information Device Profile) Upřesnění tkví hlavně v požadavcích na hardware. Je vyžadován displej s minimální velikostí 96x54 bodů a s jednobitovou barevnou hloubkou. Dále pak 8kB stálé paměti pro uložení MIDP aplikací. V oblasti síťí je požadavek na umožnění obousměrného provozu s omezenou rychlosti přenosu. Vstupy jsou realizovány za pomocí klávesnice telefonu, QWERTY klávesnice či dotykového displeje. Ve verzi MIDP 2.0 přibývá požadavek na 256kB stálé paměti a 128kB dočasné paměti. Verze 2.0 dále přináší podporu zvuku a prostředky pro vývoj her. Aplikacím napsaným za pomoci tohoto profilu se říká midlety podle základní třídy tohoto profilu. Co v MIDP profilu chybí je podpora komunikace se telefonním seznamem telefonu a možnost odesílání SMS zpráv. To je však řešeno buď individuálně v závislosti na výrobci telefonu nebo pomocí přídavného balíku (např. WMA - Wireless Messaging API).
  - PDAP(Personal Digital Assistant Profile) Jedná se o upřesnění pro PDA zařízení a specifické mobilní telefony. V zásadě zařízení s lepšími hardwarovými prostředky (větší paměť, kvalitnější displej atd.)

- CDC (Connected Device Configuration) Je to konfigurace cílená na zařízení s připojením k síti, 32-bitovým procesorem, alespoň 512kB paměti ROM a 256kB dynamické paměti. Obsahuje virtuální stroj funkčností stejný s virtuálními

stroji vyšších edicí Javy[1]. CDC konfigurace určena spíše pro větší zařízení typu set-top-box, internetová televize, videotelefony atd.. Pro tento projekt je tato konfigurace víceméně nezajímavá, avšak pro úplnost uvedu stručnou charakteristiku jednotlivých profilů a konfigurací.

- Foundantion Profile - tvoří základ pro další rozšiřující profily. Vyžaduje 1MB ROM a 512kB RAM, přidává většinu základních tříd, které CDC chybí oproti standardní edici. Neobsahuje žádné uživatelské rozhraní a také neobsahuje některé knihovny jako např.java.beans, java.rmi ani java.sql.
- RMI (Remote Method Invocation) - pro dané zařízení přidává k Foundation profilu vzdálené volání metod kompatibilní s rozhraním standardní edice.
- Game Profile - podpora pro hry
- Personal Basic Profile - přidává základní uživatelské rozhraní, avšak použití tohoto rozhraní je omezeno pouze na jednu instanci.
- Personal Profile - sem se přesunuje edice Personal java, která je již rozšířená na spoustě zařízení, jako jsou PDA nebo komunikátory. Jedná se o kompletní prostředí s plnou podporou AWT [2]. Vyžaduje 2.5MB ROM a alespoň 1MB RAM.

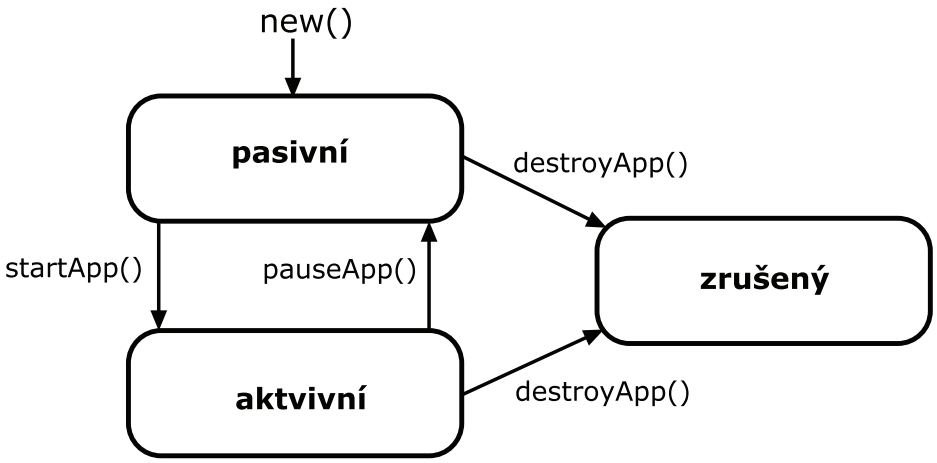
3.1.2 Midlet

Běh aplikace pro mobilní telefony (Midlet) se řídí jistými specifickými pravidly. Životní cyklus Midletu je znázorněn na obrázku 3.2. O běh aplikace a o přechody mezi jednotlivými stavy se stará aplikační manager. Rozdělení na stavy aplikace má zajistit kontrolu nad zdroji aplikace. Při vytvoření instance je aplikace v pasivním režimu kdy by neměla používat ani vlastnit žádné zdroje. Rovněž při uspání přechází aplikace do pasivního stavu, zároveň uvolňuje používané prostředky. Pouze v aktivním režimu má aplikace možnost využívat zdroje jako jsou např. vlákna nebo připojení k síti.

3.1.3 Vývojová prostředí pro J2ME

K vývoji a testování budu používat Sun Java Wireless Toolkit for CLDC. Je to balíček pro vývoj aplikací pro zařízení vyhovující specifikacím konfigurace CLDC a MIDP profilu. Standardně obsahuje nástroj pro emulaci mobilních zařízení vyvinutý společností Sun. Pro lepší otestování kompatibility je dobré stáhnout software pro emulaci konkrétního mobilního zařízení(popř. modelové řady) přímo od jeho výrobce.

[2] Abstract Window Toolkit (AWT) je rozhraní umožňující tvorbu grafického uživatelského rozhraní (GUI)



Obrázek 3.2: životní cyklus midletu

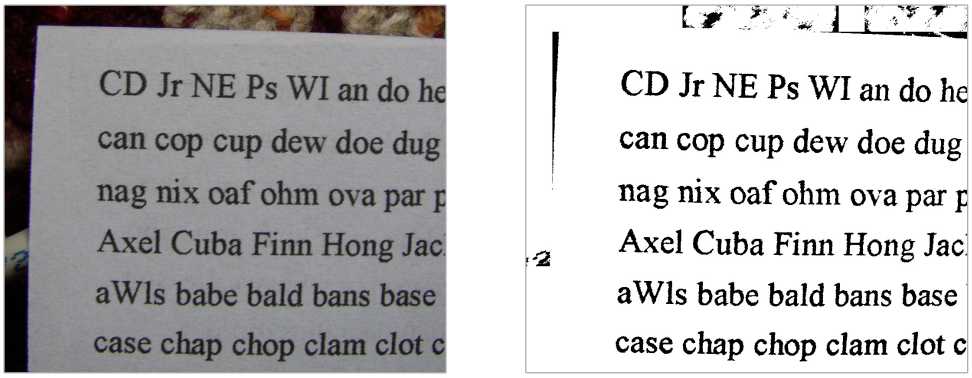
3.2 Implementace

V průběhu práce na projektu jsem došel ke zjištění, že programovací jazyk java není pro tento úkol příliš vhodný, je potřeba operovat s relativně velkými objemy dat což v jazyce java není zrovna ideální co se týče rychlosti zpracování. Proto je celá aplikace naprogramována v jazyce C++ krom jiného také proto, že v minulých letech ing. Jan Tichý vytvořil OCR aplikaci pro mobilní telefony s operačním systémem Symbian (C++)[5]. Jeho aplikace poskytuje množství ověřených dat a postupů z kterých se dá čerpat, zároveň poskytuje jistou referenční hodnotu. V mém projektu se zaměřím hlavně na proces trénování modelu, provedu několik experimentů s cílem dosáhnout co nejvyšší úspěšnosti rozpoznávání a zjistit optimální parametry pro trénování. Moje aplikace není primárně určená pro mobilní telefony, priorita tkví ve studiu a optimalizaci postupů použitých v projektu mého kolegy. V případě nutnosti nebude těžké aplikovat mnou použité algoritmy do zmíněné aplikace ing. Tichého. Pro operace s obrazovými daty používám knihovnu Cimg [1]. Je to open-source projekt Davida Tschumperlé. Tato knihovna implementuje mnohé operace s obrazovými daty jako je načítání/ukládání v různých formátech, úprava barevné hloubky, zobrazování a mnoho dalších operací. Navíc je spustitelná na mnoha operačních systémech (Unix/X11, Windows, MacOS X, *BSD)

3.2.1 Předzpracování obrazu

Do fáze předzpracování v první řadě patří úprava vstupního obrázku a to odstranění případného šumu či vlivů rozdílného osvětlené části obrazu. Použiji metodu prahování obrazu s vhodným zvolením prahové hodnoty, výsledkem bude obrázek s binární barevnou hloubkou (0 pro bílou, 1 pro černou baru). Ještě před prahováním převedeme barevnou hloubku obrazu na 256 stupňů šedi. Následně nám půjde o to, najít vhodný

algoritmus pro nalezení textu v obraze a segmentace na jednotlivé řádku textu. Zde se osvědčil algoritmus obalových čar. Tato metoda rozloží text na jednotlivé řádky, zároveň si umí poradit s různým sklonem řádků textu.



Obrázek 3.3: původní obrázek a po adaptivním prahování

Prahování

Účel prahovaní spočívá v oddělení textu od pozadí obrazu. Vhodně zvolená metoda prahování může do jisté míry zlepšit úspěšnost rozpoznávání textu. Pokud není zvolena optimálí hodnota prahu, může dojít k poškození znaků textu, či naopak k nedokonalému oddělení textu od pozadí. Rovněž můžeme docílit částečné eliminace okrajů jak je zobrazeno na 3.3. Vstup je černobílý obrázek s 256-ti stupni šedi (pokud tomu tak není, je automaticky provedena konverze), výstupem obraz s binární barevnou hloubkou.

Metoda prahování se neprovádí přes celý obraz najednou, provádíme ji po oblastech (adaptivní prahování, proměnné prahování). Tato metoda je schopna odstranit vliv rozdílného nasvícení jednotlivých částí obrazu, zároveň do jisté míry detekovat části, kde je obsažen text. Pro každou oblast je vypočten práh zvlášť, jeho hodnota je stanovena jako střední hodnota přes všechny pixely v oblasti. V každé sekci je také sledováno přípustné minimum a maximum střední hodnoty. Experimentálně jsem zjistil, že střední hodnota pro části s textem se pohybuje kolem hodnoty 200, proto sektory s extrémními středními hodnotami (menší než 70, větší než 245) jsou považovány za oblasti bez textu a je zde volen práh 1 (oblasti budou mít bílou barvu). Velikost rozměru sekcí byla rovněž stanovena experimentálně, jako optimální se zdá být čtverec o straně 50 pixelů. Při řádově menších hodnotách (menší než 10) nedocházelo k výrazným zlepšením v poměru k nárustu složitosti výpočtu. V některých případech po provedení prahování nedošlo k ideální eliminaci okrajů snímaného obrazu 3.3, je to defekt, kterému se budeme věnovat v pozdějších fázích zpracování [3].

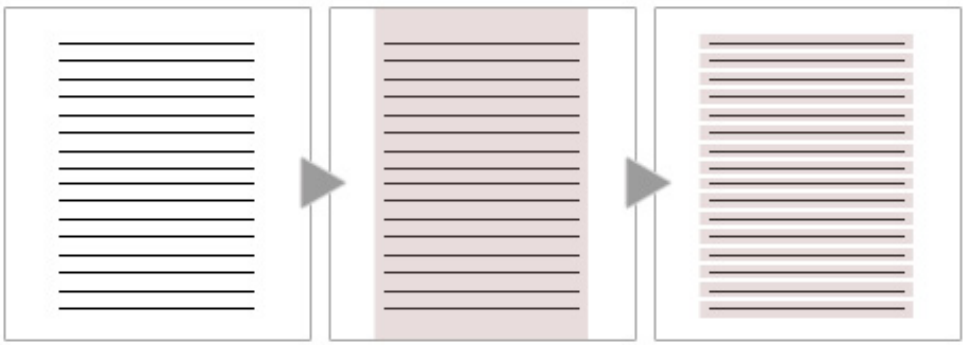
[3]Experimenty byly prováděny na obrázcích o rozlišení 1984x1488pixelů, bez blesku, denní světlo, řádky měly výšku cca 30 px.

Hrubé vyhledání řádků textu

Vstupem pro tuto část zpracování je upravený, prahovaný obraz, kde už jde poměrně snadno odlišit text. Přesto i zde mohou nastat komplikace v podobě různého sklonu jednotlivých řádků. Tento jev může nastat díky vadě na snímacím objektivu kamery (např. „soudkový efekt“) nebo prostým špatným otočením snímaného materiálu. Postup při hledání řádků bude následující: nejdříve určíme řádky velmi zhruba, pomocí vektorů změn a poté provedeme upřesnění pomocí metody obalových čar. Dále zde figuruje přibližná hodnota výšky řádku textu. Hodnota je defaultně nastavena (20px), uživatel má možnost si nastavit vlastní hodnotu minimální výšky textu. Tato konstanta je používána hlavně při prvnotním, hrubém vyhledávání řádků.

Změnové vektory jsou horizontální a vertikální, jejich prvky udávají počet změn v řádcích a ve sloupcích obrazu. Každá hodnota vertikálního vektoru je tvořena součtem změn v řádcích obrazu, analogicky pro horizontální vektor změn. Jako změna je vnímán přechod z černé barvy na bílou a naopak.

Nyní zjistíme střední hodnotu z každého vektoru zvlášť. Vypočtené střední hodnoty udávají práh, hodnoty z vektorů vyšší než tento práh tvoří řádek textu. Pro nalezení sloupce textu stačí zjistit první a poslední hodnotu, kde je prvek větší než práh. Aby se zabránilo detekování lokálních změn (a tedy špatného odhadu oblasti s textem) je použito upřesnění, kdy v případě, že narazím na text (hodnota vyšší než práh) tak kontroluji i následující bod v určité vzdálenosti (např. může být použita hodnota minimální výšky řádku). obrázek 3.4 naznačuje postup detekce řádků.



Obrázek 3.4: postup hrubého odhadu řádků textu

Pro následné zpracování metodou obalovacích čar je nejdůležitější získat co nejpřesněji výchozí bod odkud se bude poloha řádku dále upřesňovat, tedy polohu kde text začíná, začátky řádků. Tuto polohu zjistíme za pomoci vektoru vertikálních změn. Hodnoty prvků vektoru můžeme získat průchodem přes všechny body každého řádku (průchod přes celou šířku obrazu). Ovšem v situaci, kdy linky textu nejsou vodorovné, přesnost určení řádku rapidně klesá. Výpočet vertikálních změn pouze z části obrázku je slušným řešením této situace. Nabízí se možnost nastavit napevno např. První čtvrtinu šířky obrázku, avšak optimálnější je využít toho, že již máme k dispozici údaj o vzdálenosti bloku textu od pravého a levého okraje (zjištěno pomocí vektoru horizontálních změn). Zjistil jsem, že dostačující je výpočet horizontálních změn

jen z první čtvrtiny bloku textu. Docílíme tak relativně přesného odhadu začátku řádků.

Metoda obalovacích čar

Tato metoda má za úkol co možná nejpřesněji separovat jednotlivé řádky textu. Jako největší problém se ukázal sklon řádků textu a také malé vzdálenosti mezi řádky. Vstupem je prahovaný obrázek s hrubě rozlišenými řádky textu (známe přibližnou polohu prvního písmene řádku a výšku řádku). Výstupem jsou dvě posloupnosti hodnot definující horní a dolní obalovací linky.

V projektu jsou implementovány dva přístupy metody obalovacích čar. První a jednodušší je založena na tom, že v rozpoznávaném obraze je pouze jeden řádek textu. V této situaci se metoda značně zjednodušuje a zároveň je i přesnější. Nepřesnosti vznikají v obrazech, kde je více řádků textu, kde je potřeba kromě jednotlivých znaků oddělit i řádky textu. Pokud víme, že v obraze je jen jeden řádek textu (zjistíme metodou hrubého odhadu řádků textu), můžeme detekci znaků provádět jednoduchými průchody přes výšku obrázku, při kterém hledáme výskyty černých pixelů značících text. Pokud existuje v obraze více řádků tak je nutné definovat referenční bod a maximální odchylku. Referenční bod je místo, odkud provádíme detekci okrajů znaků. Na počátku je to horní levý okraj textu, zjištěný v předchozím kroku zpracovávání, ovšem v průběhu zpracovávání řádku se tato hodnota mění v závislosti na sklonu textu. Maximální odchylka udává, do jaké vzdálenosti (ve vertikálním směru) od referenčního bodu se bude hledat, snižuje se tak riziko, že se při hledání znaků dostanu na okolní řádky textu. Díky této metodě jsme schopni přesně zjistit polohu textu a to i při sklonu řádků či jiné deformaci řádku. Ušetřilo se tak několik operací a čas potřebný k metodám pro srovnání sklonu textu.

Obrázek 3.5: ideální selekce řádku textu

V experimentech s trénováním HMM modelu jsem zjišťoval i vliv toho, jak je text z obrazu selektován. Později se k tomuto problému ještě vrátím, momentálně stačí říci, že se budeme snažit specifikovat hranice textu co možná nejpřesněji (nejblíže textu), zároveň hranice by měla mít charakter přímky. Na obrázku 3.5 je znázorněna (podle mého názoru) ideální varianta detekce textu. Nevýhodou metody obalovacích čar používající referenční bod a maximální odchylku je ta, že v některých specifických případech nedokáže zachytit některé detaily v textu. Je velmi obtížné rozlišit vertikální hranice textu počítáme-li s jistým sklonem. Aby nedocházelo k prolínání obalových čar do sousedních řádků je nutné sledovat linii jednotlivých znaků co nejtěsněji. Důsledkem toho se v jistých případech může objevit situace jako je na obrázku 3.6. Každopádně i tak metoda z globálního pohledu dává dobré výsledky, navíc některé nedostatky se dají odstranit optimalizací (zvolením, vhodné minimální

Obrázek 3.6: chybné rozpoznání písmena ï¨

výšky apod.) nebo přidáním dalších podpůrných operací. V případě, že prostor mezi řádku není extrémně malý, dá se využít toho, že při dalším zpracování (může to být uložení řádku do souboru nebo výpočet feature vektoru) se přidá určitá konstantní hodnota nad i pod obalovými čarami a to buď v celém rozsahu (pro všechny prvky obalové linky), nebo pouze v případech, kdy není zvýšená pravděpodobnost kontaktu s okolními řádky. Větší pravděpodobnost kontaktu je v oblastech lokálních maxim, takže např. oblast nad velkými písmeny, znaky jako „l“, „k“ nebo pokud by se jednalo o spodní obalovou linku tak se bude jednat o znaky „p“„y“ nebo „j“. V tomto projektu jsem implementoval metodu přidání konstantní hodnoty po celém rozsahu obalových čar. Výsledek prozatímního snažení vidíme na obrázku 3.7. Text je v tuto

Obrázek 3.7: obalové čáry bez optimalizace

chvíli už dobře ohraničen, avšak charakter obalových linek není zrovna ideální pro další zpracování. Jak už bylo řečeno, je vhodné obalové linky co nejvíce vyrovnat. Na obrázku 3.7 je příklad textu, ve kterém znaky tvoří vcelku sourodou struktury, přesto obalovací linky jsou velmi zvlněné. Průchodem přes prvky obalových čar s určitým krokem zjistíme vzájemnou polohu dvou bodů ve vzdálenosti dané velikostí kroku a pokud mají prvky stejnou hodnotu, přiřadíme tu samou hodnotu i prvkům mezi nimi. Výsledek této činnosti je na obrázku 3.8. Postup můžeme provést několikrát a s jinou hodnotou kroku. Výstupem selekce řádků pomocí obalových čar je dvojice vektorů obsahující polohu horní a dolní obalující linky. Protože je každý řádek zpracováván nezávisle, je tato metoda si poradit i s různým sklonem textu v rámci jednoho bloku textu, tak jak je to zobrazeno na obrázku 5.2. Takže v této fázi máme již obrázek prahován, jsou nalezeny oblasti s textem, text je segmentován na řádky, můžeme tedy přistoupit k parametrizaci jednotlivých řádků.

3.2.2 Parametrizace

V této fázi se provádí parametrizace neboli extrakce feature vektorů. Je to proces, kdy se na jednotlivé řádky textu aplikuje DCT transformace, vypočtené hodnoty se

Obrázek 3.8: optimalizovaná metoda obalových čar

buďto uloží do souboru ve speciálním formátu (*.fea), požadovaném HTK Toolkitem (to v případě, že připravujeme data pro trénování HMM modelu)nebo se mohou poslat na vstup procesu rozpoznávání (v mé aplikaci myocr tato možnost není implementována). Cílem parametrizace je normalizovat výšku řádků na specifikovanou hodnotu, šířka řádku zůstává nezměněna. Při použití 1D DCT transformace je to počet DCT koeficientů. Výstupem tedy bude matice, která bude mít shodný počet řádků s délkou řádku v obrázku, počet sloupců matice bude shodný se zvoleným počtem koeficientů nezávisle na výšce řádku. Optimální volba počtu DCT koeficientů není triviální operací, podrobněji se této problematice budu zabývat v následující části věnované trénování HMM modelu.

K trénování používám HTK toolkit [9], součástí projektu jsou skripty pro přípravu trénovacích dat, jako je např. vytváření lab souborů. HTK Toolkit je primárně určen pro trénování modelů pro rozpoznávání řeči, pro účely ORC je ale rovněž použitelný.

3.2.3 Trénování HMM modelu

Proces trénování generuje matici přechodů, která je pak používána během rozpoznávání posloupnosti segmentů neznámého slova. Pro trénování modelu je potřeba mít připraveny vstupní data v podobě feature vektorů (zparametrizované řádky textu) a jejich textovou reprezentaci (*.lab soubory). Tuto skupinu označíme jako trénovací data. Je také vhodném připravit si data testovací, na kterých se bude ověřovat s jakou přesností je systém schopen rozlišovat text. Není vhodné volit data z trénovací množiny, výsledky by pak nebyly směrodatné poněvadž při trénování jsou tyto data přesně zapamatována. Vhodnější je použít soubor dat, které se na trénování nepodílely, tuto skupinu označíme jako testovací.

Dalším krokem prováděným ještě před zahájením procesu trénování je nastavení parametrů modelu. Nastavení parametrů má klíčový vliv na celkové rozpoznávací schopnosti systému. Je ovšem důležité volit parametry s rozmyslem s ohledem na výkonnost systému (např. úspěšnost systému ale i výpočetní náročnost roste s rostoucím počtem DCT koeficientů). Mezi tyto parametry patří především délka feature vektoru a související počet DCT koeficientů, dále počet stavů Markovova modelu, množství a typ trénovacích dat a také počet iterací při trénování modelu. Kromě těchto parametrů má vliv na výslednou úspěšnost rozpoznávání také způsob předzpracování vstupních dat, selekce řádků textu atd.. Následující experimenty s trénováním modelu mají za úkol zjistit optimální nastavení vstupních parametrů pro trénování. Také budu zkoumat ostatní faktory ovlivňující schopnost systému rozpoznávat text jako je způsob selekce či parametrizace jednotlivých řádků.

Postup trénování modelu vypadá následovně: Extrakce feature vektorů Vytvoření prototypu modelu Iterativní zpřesňování modelu Baum-Welshovou metodou pomocí funkce *HERest* z balíku HTK Toolkit [9], provádím konstantní množství iterací (25).

Pro každý typ parametru provádím sérii testů tj. pro každý typ parametru trénuji nový model. Výsledné úspěšnosti jsou udávány v procentech správně rozpoznaných znaků, jedná se o výstup funkce *HVite* zpracované programem *HResults*. HResults na základě shody předpisu se získaným výsledkem vyhodnocuje chybovost modelu, je to hodnota v procentech určující správně rozpoznané znaky, procentuální hodnoty jsou v následujících formátech: úspěšnost označovaná jako *Acc* (Accuracy) a *Corr* (Correct). Corr vypočteme ze vztahu

$$Corr = \frac{H}{N}100\ [\%] \tag{3.1}$$

kde H je počet správně rozpoznaných a N je počet všech znaků. Do výpočtu Acc navíc zasahuje parametr I udávající tzv. *insertion error*.

$$Acc = \frac{H - I}{N}100\ [\%] \tag{3.2}$$

Výpočet feature vektorů

Feature vektory jsou vlastně matice vypočtených DCT koeficientů ve speciálním formátu. Výpočet se provádí pro každý řádek textu ve vstupním obrázku. Vstupem do procesu extrakce feature vektorů je tedy řádek textu definovaný ve fázi předzpracování metodou obalových čar.

Prvotní myšlenkou pro zavedení obalových čar bylo úspěšné rozlišování řádků pomocí této metody. Máme-li už ovšem takto poměrně přesně definován text, je možné toho s úspěchem využíti i při extrakci feature vektorů. A to tak, že výpočet DCT koeficientů se bude provádět pouze z oblasti definované horní a dolní obalovou čarou. To sebou nese výhody plynoucí z menší ztráty informace-výpočet je prováděn z menšího počtu pixelů. Ovšem nevýhodou zůstává fakt, že feature vektory velkých a malých písmen jsou si velmi podobné, liší se pouze šířkou a s tím je spojena mírné zhoršení rozlišování malých a velkých písmen. I přesto však je výsledná úspěšnost při tomto postupu výpočtu vyšší než v případě získávání feature vektorů z celé výšky řádku.

Delta koeficienty

Zavedení derivace okolí sloupečku (delta koeficientů) jsou dalším prostředkem pro zvýšení úspěšnosti rozpoznávání. Z mých experimentů je vidět, že zlepšení může při vhodném zvolení parametrů dosáhnout i několikaecto procent (při použití delta koeficientů se zvýšila úspěšnost o 4% z 90 na 94%). Vezmeme-li v potaz, že jde o velice jednoduchou a výpočetně ne-příliš náročnou operaci je takové zvýšení úspěšnosti velmi dobré. Derivaci počítám z každého bodu každého sloupce obrázku po DCT transformaci (přesnější je říci, že se provádí z každého sloupce textu).

$$x_i = (c_1 x_{i+1} - c_1 x_{i-1}) + (c_2 x_{i+2} - c_2 x_{i-2}) + ... + (c_N x_{i+N} - c_N x_{i-N}) \tag{3.3}$$

Výpočet delta koeficientů je uveden ve vztahu 3.3 kde N udává vzdálenost od aktuálního bodu x_i, takže při výpočtu delta koeficientů v jednom rozměru je počet bodů podílejících se na výpočtu $2N$ (levé a pravé okolí). Množina $C = \{c_1, c_2, ...c_N\}$ jsou definované konstanty (váhové konstanty) pro jednotlivé bocy okolí.

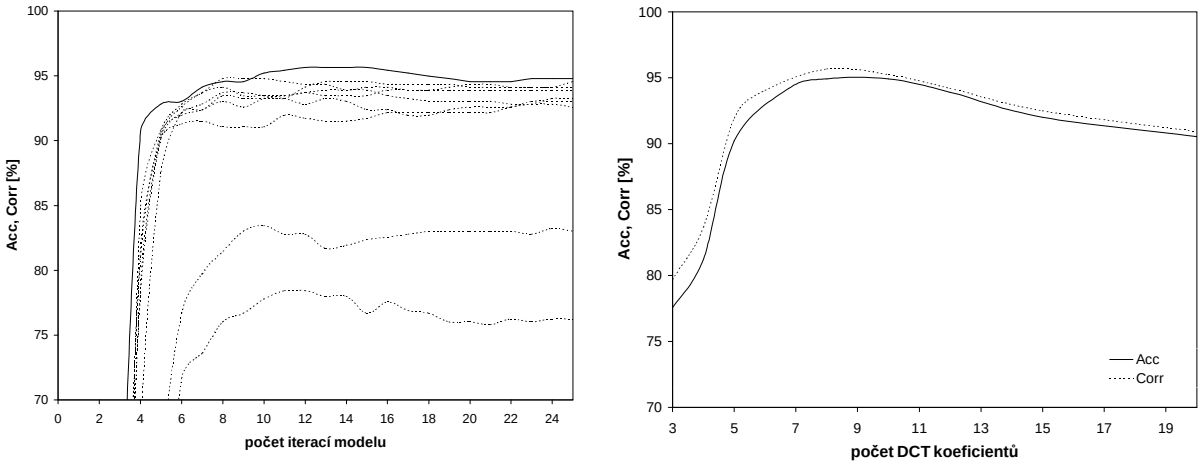
	DCT [n]	stavy modelu [n]	trénovací data řádky [n] / znaky [n]	testovací data řádky [n] / znaky [n]	Corr [%]	Acc [%]
bez delta koef.	8	10	300 / 6000	25 / 750	91,49	90,20
delta koef.	8	10	300 / 6000	25 / 750	95,42	94,12

Obrázek 3.9: tabulka optimálního nastavení delta koeficientů

V tabulce 3.9 jsou vidět výsledky experimentů s delta koeficienty (kompletní tabulka výsledků i s parametry modelu se nachází v přílohách). Je zde vidět zmiňovaný nárůst úspěšnosti rozpoznávání, nejlepších výsledků bylo dosaženo při okolí velikosti 2 pixely. Při výpočtu jsou problematické okrajové sloupce a pokud jsem se řídil zkušenostmi z rozpoznávání řeči a zkopíroval jsem první a poslední sloupce, tak úspěšnost místo očekávaného nárůstu klesla. V tomto případě je tedy vhodnější přístup, kdy okrajové sloupce obrazu nejsou do výpočtu derivací okolí zahrnuty, pouze se zkopírují z původní matice. Napadlo mě kromě výpočtu derivace v horizontálním okolí přidat pro výpočet i pixely ve vertikálním okolí, tím pádem jeden bod je upřesňován pomocí osmi sousedních bodů, po několika pokusech s nastavením parametrů jsem našel hodnotu, při které se úspěšnost zvýšila o více než 2%. Hledání parametrů a zvýšení úspěšnosti se povedlo na jednom specifickém modelu. Avšak při tomtéž nastavení derivace použité na jiné modely (různé počty DCT koeficientů, stavů modelu atd.) nedošlo k tak velkému zvýšení, mnohdy úspěšnost o něco málo poklesla. Nabízí se ještě možnost aplikovat výpočet delta koeficientů vícekrát za sebou. Avšak ani po několika různých nastavení již nedošlo ke zvýšení úspěšnosti.Rozhodně však má tato metoda rezervy a domnívám se, že hlubší studium tohoto problému by vedlo k dalšímu zvyšování úspěšnosti rozpoznávání.

Počet DCT koeficientů

Počet DCT koeficientů definuje velikost feature vektoru a s tím související výpočetní náročnost při extrakci příznaků a rovněž udává ztrátu informace z rozpoznávaného řádku. S klesajícím počtem koeficientů roste ztráta informace. V této sérii pokusů nejdříve zjistím z prvního grafu na obrázku 3.10 ideální počet iterací při trénování. S tímto počtem iterací provádím provádím trénování pro různé hodnoty DCT koeficientů, výsledek je patrný z druhého grafu 3.10. Referenční model z dostupných dat [5] má hodnotu Acc=89,70%, hodnota Corr=92,9%, viz tabulka 3.11. Těchto výsledků bylo dosaženo s použitím 5-ti DCT koeficientů a 10-ti stavů modelu, 25 iterací. Aby byla data srovnatelná, použil jsem stejných trénovacích i testovacích dat. Při stejných parametrech (5x DCT, 10 stavů) avšak s 9-ti iteracemi jsem dosáhl výsledku 94,77%. Z druhého grafu 3.10 je však patrné, že úspěšnost kumuluje v oblasti kolem 8 DCT koeficientech. Tohoto zlepšení jsem dosáhl použitím delta koeficientů (optimální nastavení z předchozího měření) a také využitím optimalizované metody obalovacích



Obrázek 3.10: nastavení referenčního modelu

čar. Možnosti dalšího zlepšování ovšem zdaleka nejsou vyčerpány, kromě následující série testů jež směřuje k nalezení optimálního počtu stavů modelu, jsou k dispozici ještě experimenty s množstvím trénovacích dat či jemného nastavení některých funkcí HTK toolkitu. Výsledky této fáze testování máme v tabulce na obrázku 3.12. Z grafu je taktéž patrné, že nad hodnotu osmi DCT koeficientů už další zvyšování počtu nemá smysl.

DCT [n]	stavy modelu [n]	trénovací data řádky [n] / znaky [n]	testovací data řádky [n] / znaky [n]	Corr [%]	Acc [%]
5	10	150 / 4500	25 / 750	92,90	89,70

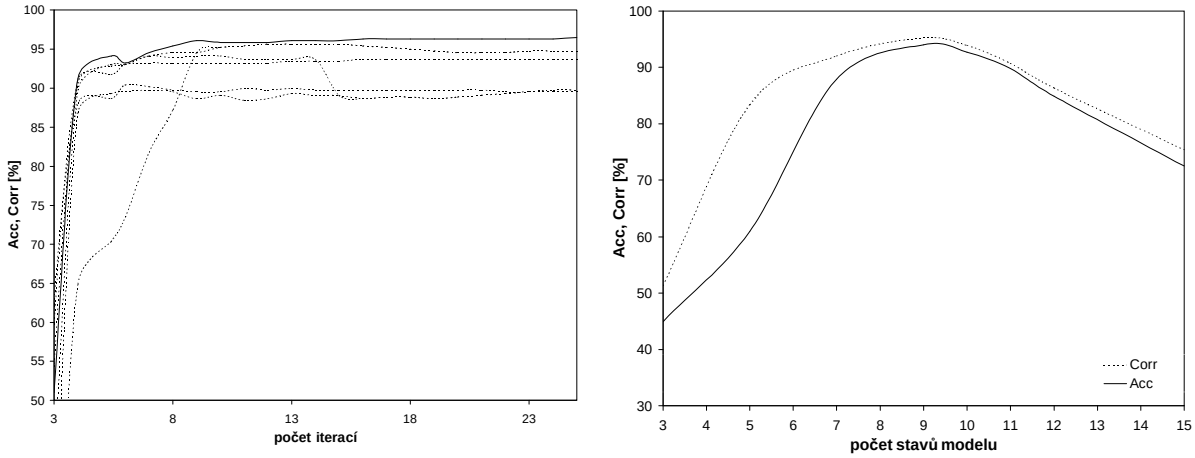
Obrázek 3.11: nastavení referenčního modelu

DCT [n]	stavy modelu [n]	trénovací data řádky [n] / znaky [n]	testovací data řádky [n] / znaky [n]	Corr [%]	Acc [%]
5	10	150 / 4500	25 / 750	93,56	92,90
8	10	150 / 4500	25 / 750	95,42	94,77

Obrázek 3.12: optimální nastavení počtu DCT koeficientů

Počet stavů modelu

V experimentech se stavy modelu sleduji výsledky při rozpoznávání textu s nastavením různých počtů stavů Markovova modelu. Zde již máme k dispozici výstupy z předešlých testů. Víme, že optimální počet DCT koeficientů je roven 8, vstupní feature vektory proto také budou mít délku 8. Výsledky pozorování jsou vyneseny



Obrázek 3.13: Graf závislosti úspěšnosti na počtu stavů modelu

do grafu 3.13. Emitujících stavů modelu je N-2, kde N je celkový počet stavů, krajní stavy slouží pouze k napojení na sousední modely. První graf 3.13 ukazuje závislost úspěšnosti rozpoznávání na počtu iterací při trénování pro množství stavů nastavených od 3 do 12-ti stavů. V druhém grafu je závislost úspěšnosti na počtu stavů při optimálních 15-ti iteracích. Maximální úspěšnost má hodnotu 95,90%, a to při 9-ti stavech modelu. Došlo tedy ke zvýšení úspěšnosti při současném snížení počtu stavů modelu. Kompletní nastavení parametrů je shrnuto v tabulce 3.14

DCT [n]	stavy modelu [n]	trénovací data řádky [n] / znaky [n]	testovací data řádky [n] / znaky [n]	Corr [%]	Acc [%]
8	9	150 / 4500	25 / 750	96,12	95,90

Obrázek 3.14: optimální nastavení stavů modelu

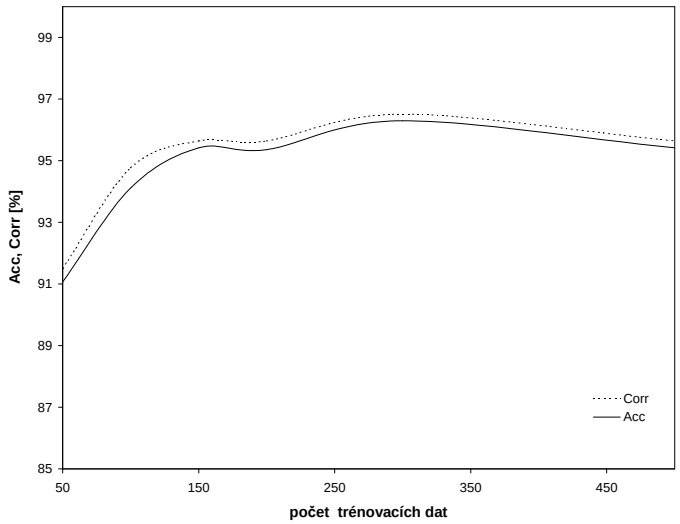
Trénovací data

Data, kterými se provádí trénování modelu. Volil jsem jak stejná data jako v referenčním projektu [5], (to aby bylo zřejmé jisté srovnání), tak i vlastní vzorky dat. Pro prvotní testy s DCT koeficienty a stavy bylo shodné i množství dat. Leč pro rozpoznávací schopnosti je dobré použít trénovacích dat více. Je ovšem třeba volit data s rozmyslem, příliš mnoho dat stejného typu by mohlo mít za následek „specializaci“ modelu na tento typ dat. Tím je myšleno, že model trénovaný na příliš podobných vzorcích by mohl mít dobré výsledky pouze na datech tohoto typu.

Dalším aspektem je obsah trénovacích dat. Při použití metody selekce řádků založené na obalovacích čarách by se dal předpokládat fakt, že smysluplný obsah vstupních, trénovacích textů bude mít kladný vliv na úspěšnost. Opak je pravdou, bylo zjištěno, že úspěšnost rozpoznávání klesá při volbě reálných textů jako trénovacích

dat [5]. Vhodnější přístup je použití generátoru náhodných znaků, který umí sledovat četnost jednotlivých znaků nebo ještě lépe i sekvencí znaků. Pro každý jazyk existují sekvence znaků, které pro daný jazyk neexistují a nebo naopak jsou velmi časté. S touto znalostí by se daly vytvářet modely nejen pro jednotlivé znaky ale i pro často se vyskytující se sekvence a také by bylo možno sestrojit omezující pravidla pro neexistující složeniny znaků. Toho by se využilo při rozpoznávání textu, sekvence rozpoznané jako neexistující by se automaticky označily jako chyba či by mohl být zaveden nápravný mechanismus schopen tuto chybu opravit. Logicky se tím zvyšuje rozpoznávací schopnost aplikace, avšak s tím dopadem, že aplikace bude použitelná jen pro daný, konkrétní jazyk.

Co se týče množství trénovacích dat, výstupy z pokusů s množstvím trénovacích dat jsou v grafu 3.15. Vstupem do trénovací fáze jsou řádky textu, každý uložen ve zvláštním souboru, součástí je i jeho textová reprezentace. V každém řádku je průměrně 30 znaků. Z grafu je jasně patrné jev označován jako „přetrénování“ modelu, nastává při volbě příliš velkého množství trénovacích dat, jako optimální množství se jeví 300 řádků, tj. cca 9000 znaků. Připomínám, že se jedná o vzorek s náhodně generovanými sekvencemi znaků.



Obrázek 3.15: Graf závislosti úspěšnosti množství trénovacích dat

Další optimalizace

Přidáním modelů pro sekvence znaků je další možností jak optimalizovat model. Provedl jsem několik úspěšných testů na jednoduchých modelech s tímto nastavením, úspěšnost rozpoznávání se při použití modelů pro často vyskytující se sekvence znaků zvýšila, ovšem zvýšení bylo patrné jen při vyšším počtu stavů modelu. Nabízí se ještě možnost opačného přístupu, kdy jsou ošetřeny neexistující sekvence znaků pro daný jazyk.

Poslední sérii testů jsem provedl s různým nastavením kovarianční matice modelu. Místo diagonální kovarianční matice jsem použil matici plně kovarianční. Pro její popis nestačí pouze vektor směrodatných odchylek, je potřeba matice a tím roste složitost modelu. Avšak zavedením plně kovarianční matice se dosáhlo nejvyšší hodnoty úspěšnosti rozpoznávání a to již při 6-ti DCT koeficientech. Model dosáhl úspěšnosti 97,34% (Acc).

Kapitola 4

Závěr

Cílem projektu bylo hlouběji se seznámit s optickým rozpoznáváním textu pomocí metody založené na skrytých Markovových modelech. K dispozici je projekt z minulých let, napsaný pro mobilní telefony s operačním systémem Symbian, obsahující mnoho cenných a ověřených postupů tvořící základ pro další vývoj a optimalizace. Rozpoznávací schopnost textu tohoto projektu činí přibližně 90%, tuto hodnotu beru jako referenční pro moji aplikaci. Díky experimentům s trénováním modelu se mi podařilo zvýšit úspěšnost na více jak 97%.

Rozpoznávání můžeme rozdělit do několika fází. Vstupní obraz nejdříve prochází úpravami s cílem odstranit různé rušivé vlivy jako je šum či rozdílné osvětlení částí obrazu. Zde se osvědčilo adaptivní prahování po částech obrazu, ideální velikost oblastí je čtverec o přibližné velikosti strany rovné dvojnásobku výšky jednoho řádku textu. Hodnota prahu je stanovena z střední hodnoty intenzity pixelů v každé oblasti.

Nalezení oblastí textu a následná segmentace řádků se jeví jako problém, který se stále nepodařilo úplně ideálně vyřešit. Při snímání reálných textů digitálním fotoaparátem vznikají komplikace hlavně v podobě rozdílného sklonu řádků textu a relativně malé vzdálenosti mezi řádky. Optimalizovaná metoda obalovacích čar, myslím si, udává dobrý směr vývoje při eliminaci zmíněných problémů a je velice dobře použitelná pro kvalitně nasnímaný materiál (bez velkého sklonu řádků). Tato metoda spočívá v hrubém nalezení počátků řádků textu a jeho následném postupném „obalení“ horní a dolní obalovou linkou. Tento postup jsem úspěšně popsal i aplikoval, leč stále zůstává obtížným úkolem přesně stanovit hranice řádků, při velkém sklonu textu se objevují nepřesnosti v hrubém odhadu počátků řádků.

Dalším úspěšným počinem bylo zavedení derivace okolí parametrizovaných bodů textu (delta koeficienty). Díky experimentům se podařilo nalézt vhodné nastavení parametrů delta koeficientů. Rovněž experimenty s parametry modelu, jako je počet DCT koeficientů, počet stavů modelu, typ kovarianční matice atd. ukázaly optimálnější nastavení těchto parametrů jež vedlo ke zvýšení úspěšnosti rozpoznávání vzhledem k referenčnímu modelu.

Prostor pro další zlepšování systémy vidím v již zmíněné optimalizaci pro vyhledání oblastí s textem a selekce řádků (především hrubé stanovení řádků) nebo vyřešit přidání slovníku do procesu rozpoznávání.

Na přiloženém CD je okomentovaný kód mé aplikace napsané v C++. Je to apli-

kace zaměřená na přípravu dat pro trénování a pro samotné trénování modelu. Postupy z tohoto projektu jsou použitelné v aplikaci určené pro mobilní telefony od Ing.Jana Tichého[5].

Literatura

[1] Cimg, *c++ template image processing library*. Dokument dostupný na URL `http://cimg.sourceforge.net` (leden 2007).

[2] Java 2 platform, *micro edition (j2me)* . Dokument dostupný na URL `http://java.sun.com/j2me/index.jsp` (listopad 2006).

[3] Černocký Jan. *Zpracování řečových signálů*. VUT Brno, 2006.

[4] Viktor Haškovec. *SEGMENTACE OBRAZU S VYUŽITÍM HYSTEREZNÍHO PRAHOVÁNÍ*. Dokument dostupný na URL `http://dsp.vscht.cz/konference_matlab/matlab04/haskovec.pdf` (listopad 2006).

[5] Tichý Jan. *Rozpoznávač psaného textu pro mobilní telefony*. VUT Brno, 2006. diplomová práce.

[6] Rajnoha Josef. *Rozponávání řeči v reálných podmínkách na platformě standardního PC*. ČVUT Praha, 2006. diplomová práce.

[7] Martin Šmat. *Metody digitální holografické interferometrie*. Plzeň, 2002. diplomová práce.

[8] Tišnovský Pavel. Diskrétní kosinová transformace (dct). Dokument dostupný na URL `http://www.root.cz/clanky/programujeme-jpeg-diskretni-kosinova-transformace-dct/` (leden 2007).

[9] L. Rabiner. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proc. of the IEEE, 1989. vol.77, no.2.

Kapitola 5

Přílohy

5.1 Delta koeficienty (derivace okolí)

N [n]	c(n) (zleva doprava od aktuálního)	Acc [%]
0	bez delta koef.	90,20
1	8	88,24
1	1	90,63
2	1,1	91,94
2	1,2	92,59
2	1,3	92,16
2	1,6	93,68
2	2,3	92,16
2	**2,8**	**94,12**
2	4,8	92,59
2	2,12	93,68
2	2,9	93,79
3	1,2,3	91,07
3	1,1,2	91,29
3	1,2,8	89,54

Obrázek 5.1: kompletní tabulka výsledků experimentů s delta koeficienty

5.2 Metoda obalových čar

Na obrázku 5.2 je znázorněn výstup z algoritmu obalových čar

5.3 Uživatelská příručka

myocr.exe - program pro úpravu obrázků s textem, detekci bloků textu, segmentaci řádků, umožňuje uložení řádků textu do jednotlivých souborů, výpočet feature

vektorů pomocí DCT. Kompilace se provede příkazem „make“. Přeložený program běží v příkazovém řádku, jako vstup může být jakýkoliv nekomprimovaný obrázek, rozšíření o podporu standardních komprimovaných souborů získáme použitím knihoven ImageMagic, libjpeg, libpng nebo XMedCon. Všechny tyto podpůrné knihovny jsou k dispozici pro jakoukoliv platformu.

Použití: myocr.exe vstupní_obrázek [parametry]
parametry:

-N int počet výstupních členů DCT transformace (default 5)

-W vykreslí obalové čáry (přesná nalezení řádků textu)

-L vykreslí nalezené řádky textu (hrubé nalezení řádků textu)

-D zobrazí zpracovávaný obrázek

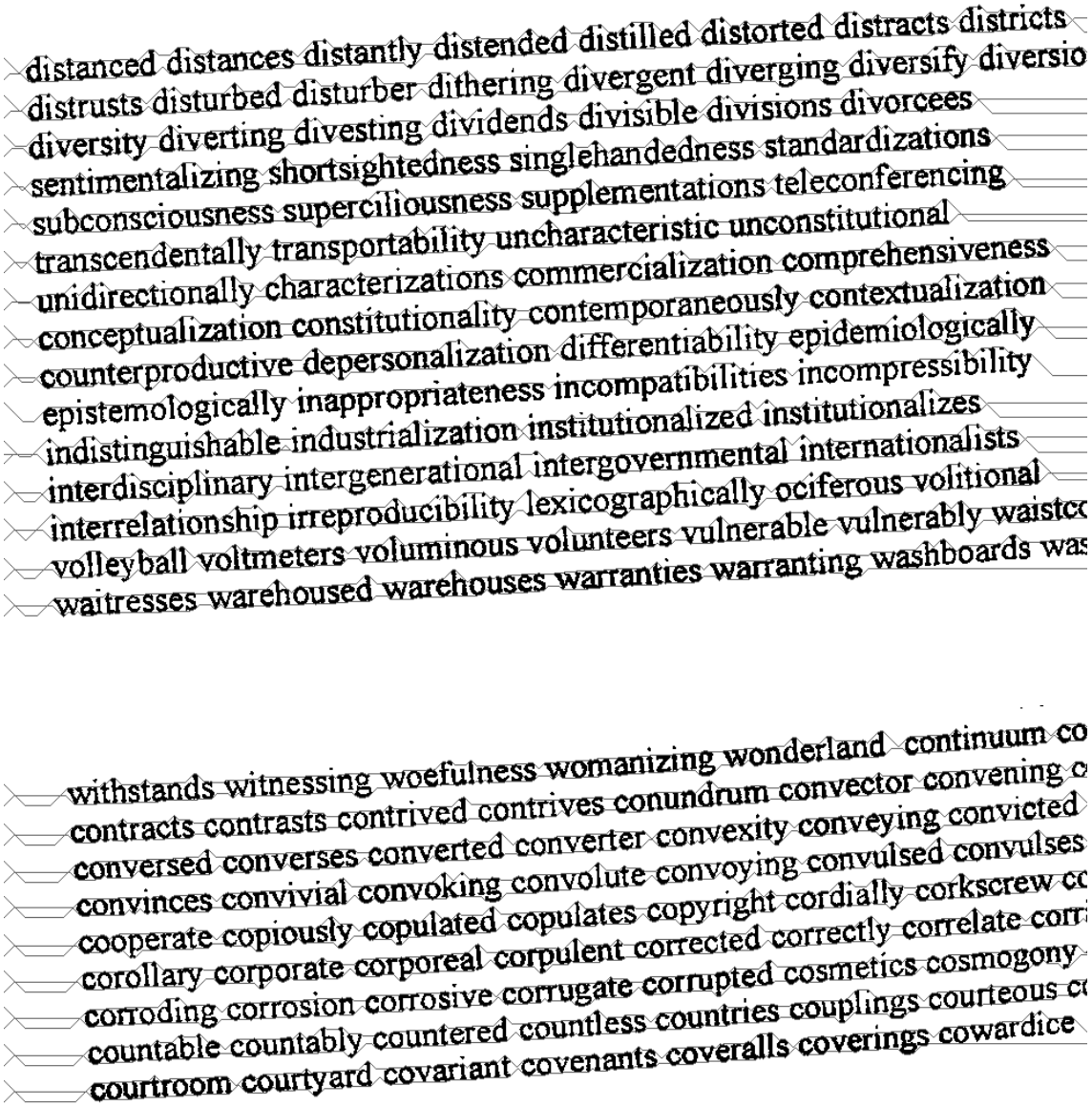
-Q tichý mód (nezobrazuje informační výpisy)

-f uložení jednotlivých řádků textu do bmp souboru. Jméno výstupního souboru je jméno vstupního souboru+označení řádku.bmp (pokud je v obraze nalezeno více než jeden řádek textu)

-F name to samé jako -f, „name“ specifikuje název výstupního souboru ve formátu bmp. Pokud je nalezeno více řádků, je k názvu výstupního souboru přidána definice řádku

-C výpočet feature vektorů a uložení do souboru ve formátu HTKToolkitu. Jméno výstupního souboru je shodné s jménem vstupního, přípona je změněna na „fea“

V případě, že ve stejném adresáři jako vstupní obrázek existuje také jeho textový přepis (stejný název, přípona TXT), tak je vytvořena sada LAB souborů.



Obrázek 5.2: Demonstrace metody obalových čar na deformovaném textu