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

OCR NA PLATFORMĚ IOS

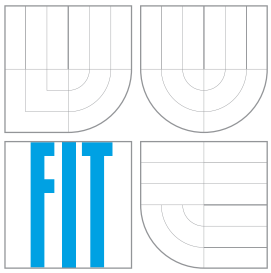
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ HAKULIN

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OCR NA PLATFORMĚ IOS

OCR ON IOS PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ HAKULIN
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAEL ANGELOV
SUPERVISOR

BRNO 2012

Abstrakt

Práce se věnuje problematice rozpoznávání textu v obraze na mobilní platformě iOS. Obsahuje principy a metody detekce textu, získávání příznaků a klasifikace. Popisuje návrh a implementaci jednoduché aplikace pro rozpoznávání informací o umístění nábytku ve skladu obchodního domu IKEA.

Abstract

This thesis is dedicated to text recognition in image on iOS mobile platform. It describes principles and methods for text location, feature extraction and classification. Portion of this work is devoted to design and implementation of simple application. With this application is possible to recognize information about location of furniture in IKEA's storeroom.

Klíčová slova

Počítačové vidění, strojové učení, klasifikace, extrakce příznaků, OCR, iOS, OpenCV, Objective – C, C++.

Keywords

Computer vision, machine learning, classification, feature extraction, OCR, iOS, OpenCV, Objective – C, C++.

Citace

Lukáš Hakulin: OCR na platformě iOS, bakalářská práce, Brno, FIT VUT v Brně, 2012

OCR na platformě iOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michaela Angelova.

.....
Lukáš Hakulin
12. května 2012

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Michaelu Angelovi za jeho odborné vedení, cenné rady a ochotu poskytnout pomoc, když jsem si něčím nebyl jistý.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Stejně jako se vyvíjí život a člověk, vyvíjí se s ním i technologie, které používá. Jedním z nejviditelnějších příkladů technologického pokroku jsou mobilní telefony. Tyto zázraky moderní doby nám ukazují, jak rychle se posouvají hranice našich možností. Na počátku jejich existence byly poměrně velké, těžké a zvládaly telefonní hovor a některé modely i textové zprávy. Proti tomu dnes tzv. chytré telefony umožňují telefonovat, psát, fotit, natáčet videa, připojit se k internetu, navigovat nás za pomoci GPS, prohlížet multimediální soubory a v podstatě téměř vše, co dokáže moderní počítač. Dá se říci, že tato zařízení jsou dnes už více počítači než telefony.

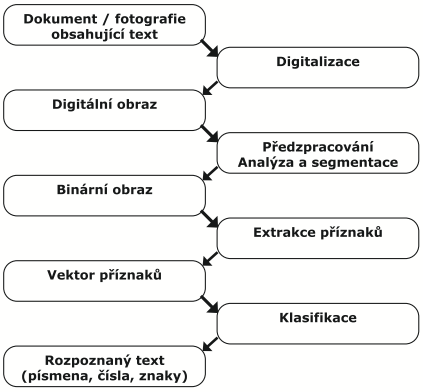
Ve své práci, která se zabývá realizací OCR systému na mobilní platformě iOS, chci těchto vlastností chytrých telefonů využít. OCR, tedy Optical Character Recognition, je realizace převodu digitálně pořízených záznamů obsahujících text do takové podoby, kdy s ním můžeme jako se skutečným textem pracovat (upravovat, mazat, dopisovat). Jistě není těžké si představit, jak široké může mít OCR využití. I v době, kdy jsou téměř všechny záznamy v elektronické podobě bezpečně uloženy a zálohovány, stále ještě existují rozsáhlé archivy obsahující důležitá data v textové podobě. Vlastní přepis takového rozsahu by znamenal nemalé požadavky na lidské zdroje, a tedy velkou časovou i finanční investici. Přepis samozřejmě můžeme nahradit prostým naskenováním, pak ale ztrácíme možnost s údaji pohodlněji pracovat. V této situaci je výhodné použít OCR systém. V současnosti, přestože vývojáři udělali mnoho pokroků v oblasti počítačového vidění, umělé inteligence a rozpoznávání, stále i komerční OCR řešení nedosahují 100 % úspěšnosti.

Mnou realizovaný OCR systém je o poznání jednodušší než komerční, protože si klade za cíl především zjistit, zda současné možnosti mobilních telefonů iPhone, vůbec umožní chod OCR aplikace, a získání výsledků v reálném čase. Pro potřeby mé bakalářské práce tedy omezím vstupní data pouze na obraz získaný kamerou mobilního telefonu a rozpoznávané znaky na čísla.

Obecně je možné model OCR rozdělit na několik podsystémů, jak je vidět na obrázku 1.1, které dohromady realizují celý proces rozpoznání znaků od jeho počátku až po konečný výstup. Jednotlivé kapitoly této práce pak reflektují tyto podsystémy. Jako první je vstup, ať už se jedná o dokument, fotografii obsahující text nebo například o video s titulky. Tento vstup, je-li to potřeba, je převeden do digitální podoby. Protože tato zařízení nejsou bezchybná, vnášejí do získaného obrazu šumy, deformují perspektivu nebo jinak znehodnocují data. Způsobům potlačení těchto vad se věnuje kapitola 2. Dále je potřeba v dokumentu nalézt oblast našeho zájmu, ať už se jedná o veškerý obsažený text nebo jen o jeho část. S nalezením textu je úzce spojeno jeho získání. Oběma těmto problematikám se věnuje kapitola 3 nazvaná Analýza a segmentace obrazu. Po segmentaci by měly být k dispozici

jednotlivé řádky, slova a znaky.

Dále zbývá je rozeznat a správně interpretovat. Nicméně stejně jako se člověk učí číst, psát a počítat, tak i počítač se rozeznávat znaky musí nejprve naučit. Tomuto tématu se věnuje kapitola 4 klasifikace. V kapitole 5 je pak popsán vlastní návrh, implementace a testování vyvíjené aplikace.



Obrázek 1.1: Jednotlivé kroky OCR systému [16]

Kapitola 2

Předzpracování obrazu

Obraz získaný fotoaparátem mobilního telefonu není dokonalým zachycením reálného světa. Každé takové zařízení vnáší do obrazu šumy, chyby a zkreslení. Úkolem metod pro zpracování obrazu je tyto zkreslení potlačit a dále uzpůsobit obraz pro další operace, které je nad ním potřeba provést.

2.1 Prahování obrazu

U velkého množství aplikací vycházejících z počítačového vidění je zapotřebí omezit informace, které jsou obsaženy v obraze, jen do dvou jednoduše navzájem odlišitelných skupin.

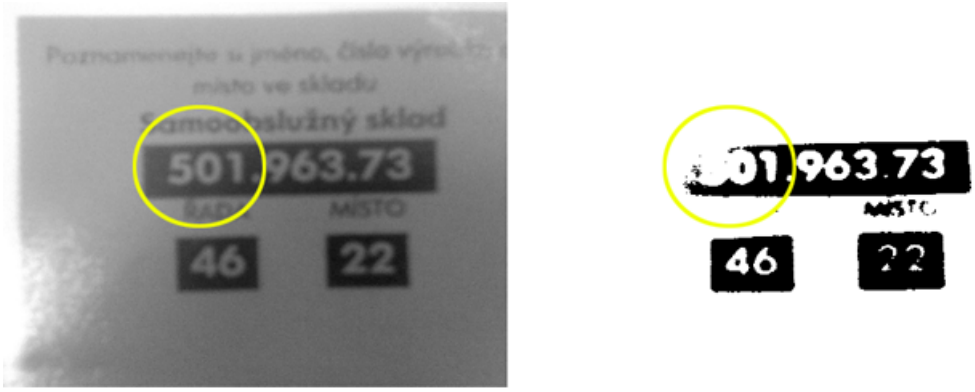
Prahování není tedy ničím jiným než transformací vstupního obrazu do jeho binární podoby na výstupu. Výstupní obraz bývá nejčastěji reprezentován černou a bílou barvou. Následující dvě vybrané metody prahování sloužily jako základ pro další, které z nich vycházejí.

2.1.1 Globální metoda prahování

Dá se označit jako intuitivní metoda. Jejím základem je určení si předem pevného prahu, a pak už dochází k porovnávání hodnoty jednotlivých pixelů obrazu. Pokud je hodnota pixelu větší než hodnota prahu, je zařazen do první skupiny, pokud je menší, je zařazen do druhé [14].

Tato metoda má jednu velkou nevýhodu a tou je nutnost znát dopředu hodnotu prahu. Co přináší pevná hodnota prahu je vidět na obrázku 2.1. Vstupem je obraz v odstínech šedi, který byl vyfotografován se zvýšeným množstvím šumu a jasu. Z důvodu přesvětlení v lokální oblasti obrazu došlo na některých místech k jasové nerovnováze a to způsobilo, že u metody globálního prahování došlo ke ztrátě informací a důsledkem toho ke ztrátě informací ve výstupním obraze. U systémů jako je OCR má taková ztráta naprosto fatální dopad na následnou schopnost rozpoznání znaků.

Jednou z možností jak těmto ztrátám předejít, je měnit hodnotu prahu podle hodnot získaných z histogramu obrazu. Podobná vylepšení v sobě zahrnují metody, které z tohoto modelu vycházejí.



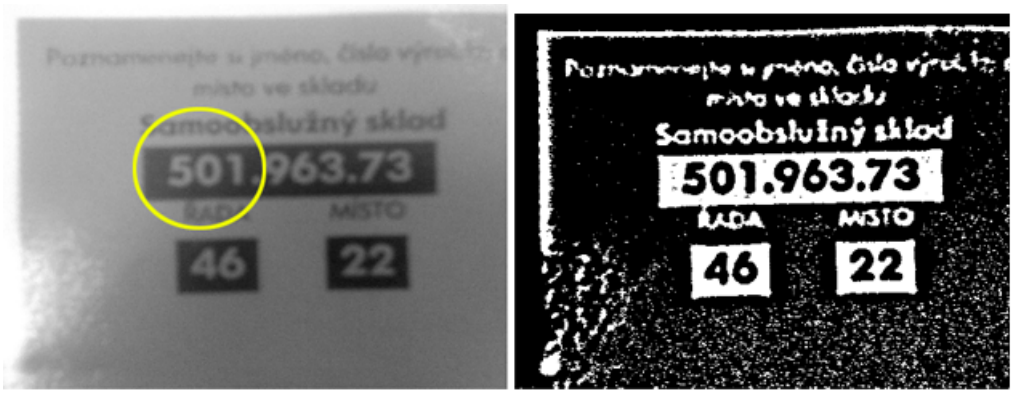
Obrázek 2.1: Globální metoda prahování

2.1.2 Adaptivní metoda prahování

Adaptivní metoda je příkladem pokročilejšího přístupu k volbě prahové hodnoty. Jak bylo dříve naznačeno, stíny, šumy, lokální přesvětlení mohou vést k tomu, že jedna prahová hodnota není vhodná pro celý obraz.

U adaptivní metody se setkáváme s přístupem dynamické změny prahové hodnoty pro jednotlivé pixely na základě vlastnosti daného pixelu nebo vlastností jeho blízkého okolí. Nejčastěji se spočítá průměr z hodnot okolních pixelů a ten se použije jako jeho prahová hodnota. Tímto přístupem se do prahové hodnoty dá začlenit i intenzita osvětlení.

Výsledek takového přístupu můžeme vidět na obrázku 2.2. Další metody prahování podstatně vycházejí z myšlenky adaptivní metody a rozšiřují ji. Hlavní myšlenkou však nadále zůstává získat binární obraz s co největší mírou informace neboli bez ztrát způsobených nízkou kvalitou vstupního obrazu.



Obrázek 2.2: Adaptivní metoda prahování

2.2 Vyčistění obrazu

Při mnoha operacích nad vstupním obrazem je do něj, jako nechtěný sekundární produkt, zaneseno jisté množství šumu. Čistěním se snažíme tohoto šumu zbavit a potlačit tak jeho

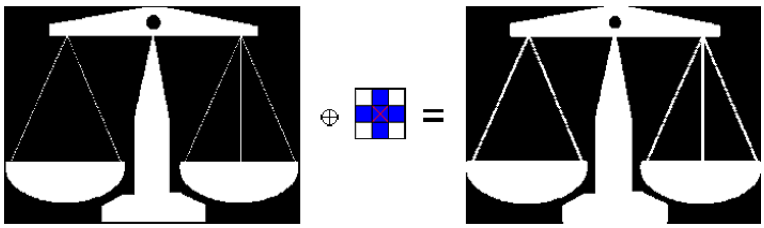
nepříjemný dopad na další úpravy.

Metody eroze a dilatace jsou jedny z nejzákladnějších metod používaných v počítačovém vidění a následující text stručně přibližuje, jakým způsobem fungují a kdy je vhodné je použít.

2.2.1 Dilatace

Operace dilatace je jednou ze základních v binární morfologii. Neformálně řečeno, dilatace zvětšuje objekty. Používá se pro odstranění jednopixelových chyb v obraze.

Vstupní obraz je binární. Nad tímto obrazem provedeme operaci dilatace s určitým strukturálním elementem. Tím je nejčastěji morfologický kříž reprezentovaný maticí o rozměru 3×3 obrazových bodů, který způsobí zaplnění jedno-pixelových děr a rozšíření objektů o jednu vrstvu. Na obrázku 2.3 je tento postup demonstrován.

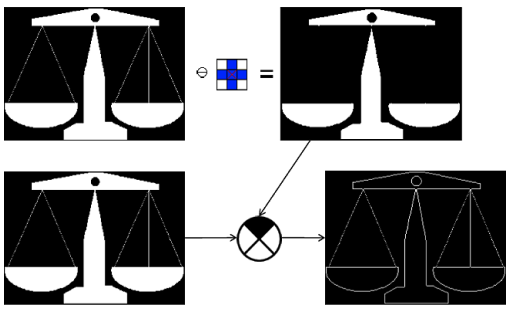


Obrázek 2.3: Operace dilatace obrazu [8]

2.2.2 Eroze

Operace eroze není inverzní operací k dilataci, jak by se mohlo z názvu zdát. Její aplikací se objekt zmenší o jednu vrstvu a dojde k odstranění jedno-pixelových čar.

Stejně jako u dilatace je vstupem binární obraz a tvar strukturálního elementu nám určuje výsledný tvar. Velmi zajímavou aplikací operace eroze je možnost získat obrysy objektu. Tento jev je znázorněn na obrázku 2.4. Vhodně zvolená křížová struktura objekt zmenší o jednu vrstvu na úkor okolí, pak rozdílem původního obrazu a obrazu erodovaného získáme obrysy objektu.



Obrázek 2.4: Obrys objektu operací eroze [8]

Tento postup může u jednoduchých objektů nahradit, na výpočet mnohem náročnější, Cannyho hranový detektor 2.4.1.

2.3 Vyhlazení obrazu

Vyhlazení potlačuje skokové změny v obraze, čímž vyhlazuje hrany a snižuje rozdíly jasu. Společnou myšlenkou pro všechny druhy vyhlazování obrazu je určit novou hodnotu jasu z omezeného lokálního okolí obklopujícího daný bod.

Tato operace najde uplatnění tam, kde potřebujeme potlačit šum, zjemnit obraz, ale stejně tak na místech, kde můžeme předpokládat, že jsou přítomny objekty, které byly během úprav obrazu poškozeny jako vedlejší produkt našich vlastních operací nad vstupním obrazem.

Následující tři metody demonstrují možné přístupy k problematice vyhlazování obrazu.

2.3.1 Průměrování

Základní metoda vyhlazování na základě konvoluce obrazu s maskou. Pokud je jas v bodě (i, j) dán lineární kombinací jasů v okolí O(velikosti $M \times N$) vstupního obrazu g s váhovými koeficienty h platí vztah

$$f(i,j) = \sum_{m=i-M/2}^{i+M/2} \sum_{n=j-N/2}^{j+N/2} h(m-i, n-j) g(m,n) \tag{2.1}$$

Výsledný jas bodu zásadně ovlivňuje velikost a obsah masky jak můžeme vidět tady:

Pro $h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ dostaneme aritmetický průměr okolí 3×3.

Pro $h = \frac{1}{10} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ zvýšíme váhu středu.

Jednou z možných obměn průměrování je určení maximálního rozsahu změn. Takovou úpravou docílíme omezení změn jasu proti původnímu obrázku [9].

2.3.2 Medián

Medián je číslo, které se v poli hodnot uspořádaných podle velikosti nachází uprostřed pole. Pokud je počet prvků sudý, zvolí se jako medián průměr z dvou prvků nejblíže středu.

Metoda vyhlazení založená na vlastnostech mediánu tolik nerozmazává hrany, na druhou stranu ale poškozuje tenké čáry a ořezává ostré rohy. Bývá využívána pro odstranění úzce specifikovaného druhu šumu, černých a bílých pixelů v obraze. Tato jeho vlastnost je demonstrována na obrázku 2.5.

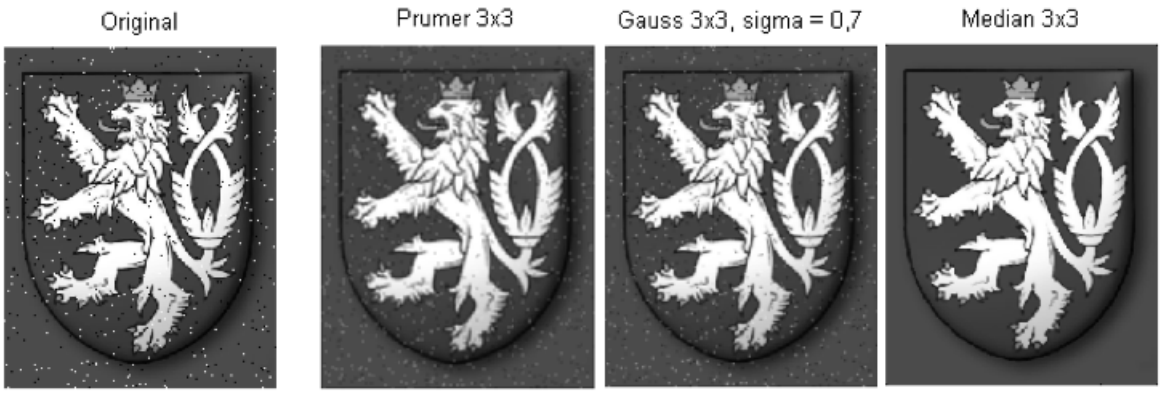
2.3.3 Filtr s Gaussovým rozložením

Metoda pro odstranění šumu konvolucí s maskou danou jednoznačně Gaussovou funkcí. Pro 2D obraz je Gaussova funkce [9]:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

Kde (x, y) jsou souřadnice obrazu a σ je směrodatná odchylka, která udává velikost okolí na kterém filtr pracuje.

Filtr s Gaussovým rozložením se používá pro odstranění šumu s normálním rozložením. Jedná se o jednu z nejrozšířenějších metod implementovanou v mnoha grafických aplikacích.



Obrázek 2.5: Porovnání metod vyhlazení (průměrování, Gaussův filtr, medián) [11]

2.4 Detekce hran

V problematice detekce a rozpoznávání objektů v obraze jsou některé prvky důležitější než jiné. Jedním z těchto prvků jsou hrany, které prokazatelně nesou větší míru informace než jiná místa. Pomocí rozpoznání hran můžeme určit, kde se v obraze objekty nacházejí, jaký mají tvar, kde začínají a končí a díky tomu je můžeme sledovat, počítat, odstranit aj.

Hrana je v obraze reprezentována jako oblast, kde se skokově mění hodnota jasu. Tato oblast má specifickou šířku a úroveň změny intenzity. Čím je šířka užší a změna jasu větší, tím je hrana ostřejší.

Na metody rozpoznávání hran klademe několik základních požadavků jako přesnost, jednoznačnost a nízkou chybovost. Dodržením těchto kritérií se vyhneme nechtěnému zdvojení hran, nepřesnému určení polohy a především detekci oblastí, které vůbec hranami nejsou.

2.4.1 Cannyho hranový detektor

Jedna z nejpoužívanějších metod detekce hran [4]. Za základní metodu se dá označit detekce za pomoci konvoluce speciálního operátoru s obrazem. Často používanými operátory jsou Sobelův, Robinsonův, Robertsův a Prewittové.

Cannyho hranový detektor byl navržen tak, aby splňoval tři hlavní požadavky:

Minimální chybovost: Detekovány jsou všechny hrany a pouze hrany, nikoliv místa, která hranami nejsou.

Přesná lokalizace: Poloha nalezené hrany musí přesně odpovídat skutečné hraně ve vstupním obraze.

Jednoznačnost: Rozpoznání hrany tak, aby nedocházelo ke zdvojení.

Pro splnění těchto podmínek v sobě zahrnuje následující čtyři základní kroky:

1. Odstranění šumu za použití filtru s Gaussovým rozložením.
2. Určení gradientu Sobelovým operátorem.
3. Ztenčení hran (Ztenčování oblastí na křivky o šířce právě jednoho obrazového bodu tak, aby výsledný tvar co nejlépe vystihoval tvar původní.).
4. Prahování s hysterezí.

Algoritmus tedy nejprve odstraní šum, pak určí gradient za pomocí konvoluce s maskou Sobelova operátoru, který je méně citlivý na šum. Velikost masky je jednou z možností jak ovlivnit výslednou detekci. Ztenčování je prováděno hledáním lokálních maxim prvních derivací tak, aby bylo zajištěno nalezení jen skutečných hran v místech největšího gradientu. Prahování s hysterezí slouží k rozlišení významných hran. Jsou použity dva prahy a ty určují, co je považováno za hranu a co ne. Pokud je gradient vyšší než druhý práh, je považován za hranu, pokud je větší než první práh a současně menší než druhý, je považován za hranu, pokud se v jeho okolí nachází bod, který byl hranou uznán. Pokud je gradient nižší než první práh, je zahozen. Výsledek algoritmu Cannyho hranového detektoru můžeme vidět na obrázku 2.6. Obrázek označený (a) je orginální předloha, na obrázcích označených (b) a (c) je vidět jaký vliv má změna hodnoty prahů na nalezení hran.

Obrázek 2.6: Ukázka použití Cannyho hranového detektoru z knihovny OpenCV

Kapitola 3

Analýza a segmentace obrazu

Cílem analýzy obrazu je rozčlenit jeho obsah, který koresponduje s předměty, oblastmi z reálného světa tak, aby zůstaly jen ty, obsahující pro nás významné informace. Spojením této analýzy s následnou segmentací získáme na sobě nezávislé objekty, které použijeme jako vstup pro metody rozpoznávání a klasifikace OCR.

Analýza je založena na vlastnostech vstupního obrazu a našich znalostech jeho obsahu. Využíváme znalostí barev, velikostí objektů, tvarů, polohy, poměru velikosti jednoho známého objektu vůči druhému aj.

3.1 Detekce barev

Lidské oko vnímá jisté vlastnosti materiálů jako barvu. Přesněji světlo dopadající na povrch libovolného objektu je vlastnostmi povrchu ovlivněno a po odrazu nese informaci o objektu, kterou naše oko zpracuje a pošle do mozku, který ji interpretuje jako barvu.

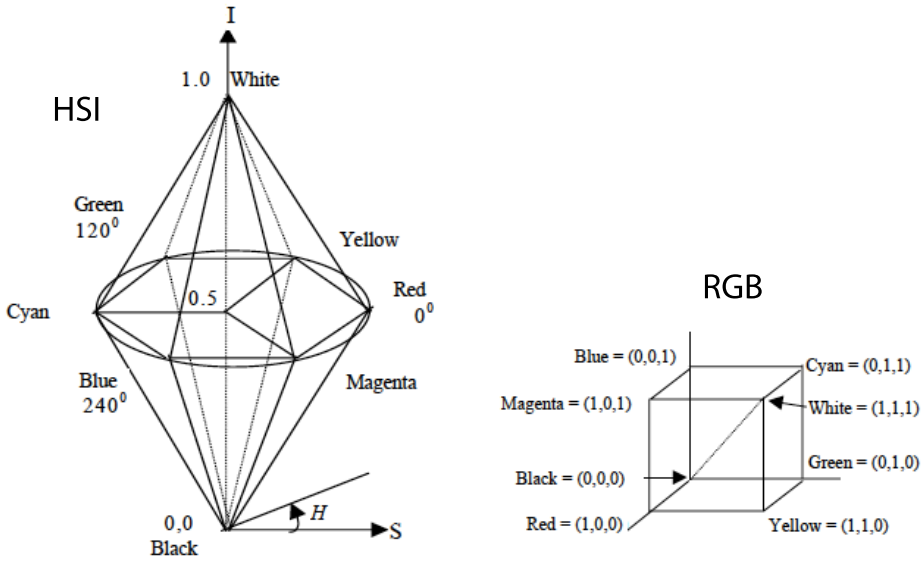
Pro různé účely se postupem času vyvinuly rozličné druhy metod, kterými reprezentujeme barvy. Těmto reprezentacím říkáme barevné modely. Nejrozšířenějším modelem v oblasti počítačových technologií je RGB. Tento model je založen na aditivním skládání barev a je reprezentován třemi základními barvami. Červenou, zelenou a modrou. Kombinací těchto tří barev dostatečně obsáhneme rozsah barev, které je naše oko schopno vnímat.

Pro účel rozpoznání určité specifické barvy, by tento model dostatečně vyhovoval, pokud však chceme rozeznávat nejen barvu, ale i všechny její odstíny, sytost a jas, je vhodnější zvolit jeden z modelů HSL (hue, saturation, lightness), HSV (hue, saturation, value) nebo HSI (hue, saturation, intensity). Dva příklady barevných modelů můžeme vidět na obrázku 3.1.

Například HSL, uživatelsky orientovaný model, dovoluje pracovat s barvou pomocí třech kanálů:

- Sytost (*saturation*)
- Barevný tón (*hue*)
- Jas (*lightness*)

Potom můžeme zvolit jen rozsah barevného tónu a tím určíme barvu. Jednou z oblastí, kde je tento barevný model využíván, je problematika detekce obličeje. Každý člověk má rozdílnou barvu kůže, přitom se však dá pro určitou skupinu specifikovat rozsah barev, který všechny jednice ve skupině zahrnuje [15], [18].



Obrázek 3.1: HSI a RGB model [15]

3.2 Detekce kontur v obraze

Pomocí nalezení kontur v obraze můžeme rozeznávat objekty, které obsahuje. Kontura v oblasti počítačového vidění je obrys hlavních linií předmětu v obraze, který nám dává představu o tom, jaký má objekt tvar. V případě, že očekáváme předem známý objekt, získáme také informaci o tom, zda je natočený, částečně deformovaný aj.

Detekce kontur je nepostradatelná pro segmentaci obrazu. Především v případech, kdy hledáme tvarově složitější objekt v členitém obraze, může být nalezení a správná interpretace kontur jediným řešením.

Největší překážkou v procesu nalezení kontur je nízká rozlišitelnost toho co je skutečně hlavní linií objektu a co může být například jen zastínění jiným objektem, kdy vznikne přechod vyššího kontrastu. Každá z metod detekce kontur se snaží tyto případy správně rozlišit a předejít chybám.

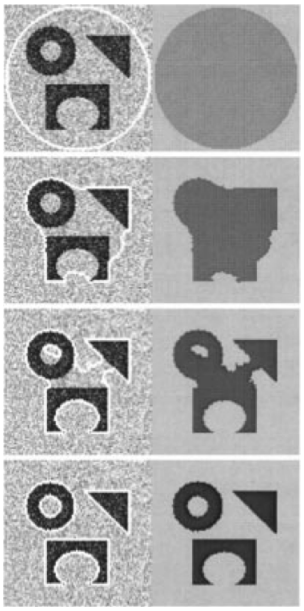
3.2.1 Aktivní kontury

Základní myšlenkou modelu aktivních kontur je vyvíjející se křivka omezená hranicí vstupního obrazu, jejímž úkolem je odhalit objekty v obraze [5].

V angličtině se u pojmu aktivních kontur často setkáme ještě s jiným názvem a to snake (had). Had proto, že tato metoda je používána především pro detekci složitějších tvarů a ty jsou reprezentovány křivkou skládající se z kontrolních bodů. Změny tvaru křivky abstraktně připomínají vlnícího se hada.

Na počátku je křivka inicializována kolem objektu a postupně se deformuje, dokud se nepřimkne k hranám objektu. Deformaci způsobuje vliv vnějších, vnitřních a obrazových energií. Vnitřní energie roste s velikostí hada a jeho zakřivením. Prudké zakřivení zvyšuje vnitřní energii. Vnější síly ovlivňuje umístění křivky. Obrazové síly kontrolují tvarování směrem k hranám, vnitřní hladkost průběhu. Výsledná pozice kontury je lokálním minimem součtu energií kontury. K řízení výše popsaných energií může být využito více postupů.

Metoda je demonstrována na obrázku 3.2.



Obrázek 3.2: Model aktivní kontury [5]

3.3 Segmentace znaků

Oddělení sekvence za sebou následujících znaků (písmen, číslic, obecných tvarů) je nedílnou součástí OCR systémů. Je to poslední krok před získáváním příznaků a následnou klasifikací. Existuje celá řada přístupů k této problematice. Jedním z intuitivních je použití horizontálního promítnutí. Při procházení vzniklé struktury pak hledáme lokální minima a považujeme je za hranici (mezeru) mezi znaky. Tento přístup je však nepoužitelný u textů s menšími mezerami mezi znaky (kerning). Kerning je dnes běžnou součástí fontů, kdy každá dvojice znaků má definovánu hodnotu kerningu tak, aby opticky působila shodně s ostatními mezerami, jak je vidět na obrázku 3.3. U těchto druhů textu může dojít k lokálnímu spojení po sobě jdoucích znaků.

Tento problém můžeme v některých případech vyřešit prostou úpravou vstupního obrazu pomocí morfologických operací jako ztenčování nebo přímo operací otevření, která je složením eroze následované dilatací. Morfologické otevření od sebe odděluje objekty spojené jen tenkou čarou a zároveň odstraňuje šum. Podobně jako u eroze a dilatace má na výsledek vliv velikost a tvar strukturálního elementu [8].

Pokud se ale některé znaky plně překrývají, mluvíme o záporném kerningu, může dojít k rozdělení znaků na více částí a následnému chybnému rozlišení. Jedním z algoritmů, který tento problém řeší je connected component labeling [13] [6]. Výsledkem aplikace tohoto algoritmu je rozčlenění znaků v obraze do tříd. Každá třída obsahuje oblasti, které jsou vzájemně propojeny.

Dvouprůchodový connected component labeling algoritmus pracuje ve třech různých fázích [17].

Příklad (Náhodná SLOVA)	Font příkladu
Náhodná SLOVA	Edwardian Script ITC
Náhodná SLOVA	Birch Std
Náhodná SLOVA	Lucida Calligraphy

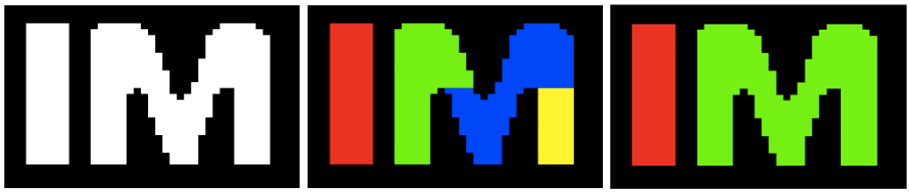
Obrázek 3.3: Ukázka překrývajících se znaků (záporný kerning) a lišících se velikostí mezer mezi znaky

Skenovací fáze: Obraz je procházen pro přiřazení prozatímních tříd pixelům objektů a je zaznamenána informace ekvivalence mezi prozatímními třídami.

Analytická fáze: Tato fáze analyzuje informace ekvivalence pro určení konečné třídy.

Označovací fáze: Třetí fáze určí konečné třídy pixelům jednotlivých objektů pomocí druhého průchodu obrazem.

V závislosti na struktuře dat používané pro zastupování ekvivalence informací, můžeme fázi analýzy začlenit do skenovací fáze nebo označovácí fáze. Na obrázku 3.4 vidíme zjednodušený model fází algoritmu.



Obrázek 3.4: Vstupní binární obraz (vlevo), obraz s barevně rozlišenými třídami (uprostřed) a výstupní obraz obsahující dva nezávislé regiony dat patřících do dvou tříd (vpravo) [10]

Kapitola 4

Klasifikace

Klasifikace je posledním nutným krokem pro funkci OCR systému a to krokem nejpodstatnějším. Tady dochází k určení znaku a to přiřazením ho k jedné z předem známých tříd. Podle příslušnosti k dané třídě, pak získáváme další informace o znaku.

Znak před klasifikací nese pouze informaci, že se jedná o libovolný znak z dané abecedy. Předpokládáme-li, že abeceda v sobě obsahuje všechny druhy písmen s i bez diakritiky a čísla plus některé speciální znaky, není tato informace příliš určující. Po zjištění příslušnosti k třídě, už ale víme, že se například jedná o číslo a ne písmeno a je to nula.

Pro klasifikaci se v oblasti počítačové grafiky používá algoritmus zvaný klasifikátor. Správná volba klasifikátoru je důležitou částí návrhu, obecně založenou na zkušenostech a pokusech. Vstupem velké skupiny klasifikátorů je příznakový vektor, který v sobě nese informace (příznaky) důležité ke klasifikaci. Právě i na jejich vlastnostech můžeme vybrat správný klasifikátor.

Jakmile máme zvolen klasifikátor, přejdeme k procesu učení nad předem zvolenými trénovacími daty. Tato data by měla pojmout co možná nejobsáhlejší škálu příkladů, které bude později v reálném nasazení klasifikovat. Po natrénování klasifikátoru dochází k otestování jeho schopností na jiné množině, tentokrát testovacích dat. Tento test určí, jestli bylo učení úspěšné. Mohlo by se zdát, že čím je trénovací sada obsáhlejší, tím musí být pozdější klasifikace úspěšnější, ale to není vždy pravda. Nedílnou součástí klasifikátoru je jeho schopnost generalizace předem neviděných dat. Tato vlastnost mu umožňuje klasifikovat i data, které v trénovací množině nebyla zastoupena, a tím se stát obecnějším. Pokud tedy natrénujeme klasifikátor na příliš obsáhlé trénovací sadě se specifickými vlastnostmi, může dojít k přetrénování a klasifikátor bude schopen jen klasifikace na úzké skupině trénovacích dat.

Jak tedy bylo uvedeno výše, před použitím klasifikátoru v praxi je třeba jej otestovat na testovacích datech. Není neobvyklé, že je proces učení prováděn opakovaně a měněn počet trénovacích dat, charakter příznaků aj.

4.1 Příznaky

Příznaky nám popisují objekty získané při segmentaci. Problematika příznaků spočívá v nalezení takových informací, které nejlépe daný objekt popisují. Přitom jsou na příznaky kladeny tyto požadavky:

- Invariantnost na změnu měřítka, jasu, kontrastu, rotaci a pod.
- Spolehlivost na podobné hodnoty získané z dvou objektů též třídy.

- Diskriminabilita, tedy různé hodnoty příznaku pro různé objekty.
- Efektivita výpočtu, protože dokonalý příznak získaný hodiny trvajícím výpočtem je nám k ničemu.
- Časová invariance při zpracování dynamických obrazů.

Zvolené příznaky mohou mít například charakter radiometrický (tvar, velikost, obvod, ...) nebo fotometrický (lokální jasové úrovně objektu, histogram, ...). Výběr příznaků je pro správnou klasifikaci stejně důležitý, jako výběr vhodného klasifikátoru [1],[2].

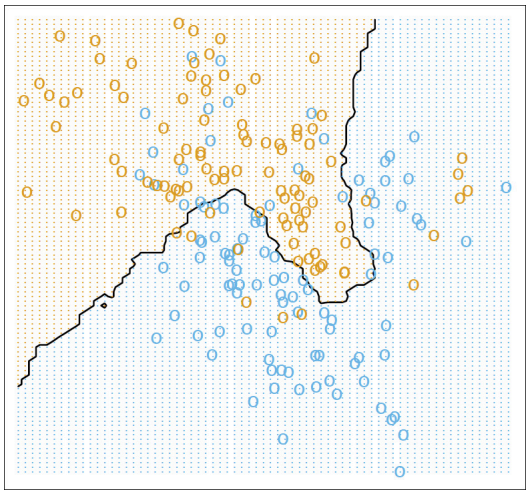
4.2 K–nejbližších sousedů

K–nejbližších sousedů je jedním z nejjednodušeji implementovatelných klasifikačních algoritmů, které se používají. Čím je metoda známa je ale především to, že u ní nedochází k procesu učení. Pouze jsou pro její funkci z trénovací sady vybráni reprezentanti, kteří ji nejlépe charakterizují. Tyto reprezentanty si pak metoda pamatuje a na základě porovnání vzdáleností vzorů od jednotlivých reprezentantů klasifikátor určí do jaké třídy patří.

Reprezentantů může být i více a jejich počet a výběr má vliv na míru úspěšnosti klasifikace. Při klasifikaci algoritmus K–nejbližších sousedů hledá k reprezentantů ze všech tříd, které jsou nejblíže vzoru. Z k nalezených reprezentantů se spočítá do jaké třídy připadá nejvíce nalezených reprezentantů a tato třída je vzoru přiřazena. Pro $k = 1$ je vzoru přiřazena třída, do které připadá nejbližší nalezený reprezentant.

Pro zjištění vzdálenosti je nejčastěji volena *Euklidovská* nebo *Hammingova metrika*. Právě výběr a počet reprezentantů společně se zvolenou metrikou a konstantou k mají vliv na rychlost a úspěšnost klasifikace.

Tato metoda bývá také používána pro srovnání s ostatními metodami klasifikace [1],[2]. Ukázku použití klasifikátoru K-nejbližších sousedů můžeme vidět na obrázku 4.1.



Obrázek 4.1: Použití klasifikátoru K-nejbližších sousedů na dvou třídach pro $k = 15$ [7]

4.3 Naivní bayesovský klasifikátor

Klasifikátor vycházející z Bayesovy věty o podmíněných pravděpodobnostech. Základní Bayesův vztah pro výpočet podmíněné pravděpodobnosti zní:

$$\text{posterior probability} = \frac{\text{likelihood} \times \text{prior probability}}{\text{evidenc}}$$

Matematický zápis by byl:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{4.1}$$

Tento vztah říká, že platí předpoklad H, pokud pozorujeme stav (událost) E

Apriorní pravděpodobnost hypotézy $P(H)$ odpovídá znalostem o zastoupení jednotlivých tříd bez ohledu na další nformace. Podmíněná (aposteriorní) pravděpodobnost $P(H|E)$ vyjadřuje změnu pravděpodobnosti hypotézy, pokud víme že nastal stav E. $P(E)$ vyjadřuje pravděpodobnost události E.

Protože nás zajímá pouze nejvyšší pravděpodobnost (MAP, *maximum aposteriory*) pro stav E a současně nás nezajímá konkrétní hodnota, můžeme vzorec upravit:

Hypotéza H_t pro $t = 1, \ldots, T$

$$H_{MAP} = H_j, \text{ právě když } P(E|H_j)P(H_j) = max_t(P(E|H_t)P(H_t)) \tag{4.2}$$

Dalším předpokladem je, že všechny stavy jsou nezávislé. Právě díky tomu získal své pojmenování naivní (v anglické literatuře také *idiotics*). Tento předpoklad nám umůžňuje počítat aposteriorní pravděpodobnost pro stavy E_1 až E_k jako:

$$P(H|E_1, \ldots, E_K) = \frac{P(H)}{P(E_1, \ldots, E_K)} \times \prod_{k=1}^{K} P(E_k|H) \tag{4.3}$$

Při klasifikaci pak hledáme hypotézu s největší aposteriorní pravděpodobností H_{MAP}:

$$H_{MAP} = H_j \text{ právě když } P(H_j) \times \prod_{k=1}^{K} P(E_k|H_j) = max_t(P(H_t) \times \prod_{k=1}^{K} P(E_k|H_t)) \tag{4.4}$$

Tato statistická klasifikační metoda se často používá, a i přesto, že předpoklad nezávislosti nebývá u skutečných dat příliš častý, dosahuje dobrých výsledků. Tuto skutečnost literatura vysvětluje tím, že hledáme jen největší hodnotu a pro klasifikaci používáme více atributů, než je třeba. Pracujeme tedy s velkým informačním přebytkem [1],[2].

4.4 Rozhodovací stromy

Klasifikátor reprezentující znalosti ve formě rozhodovacího stromu. Díky této struktuře je jeho prohledávání a tedy celá klasifikace velmi rychlá.

Principem je průchod stromem od kořene k uzlům a v každém uzlu se algoritmus rozhodne, kam bude dále pokračovat podle hodnoty atributu. Jakmile se při rozhodování dostaneme do listu stromu, je proces skončen a jako výsledek je převzata hodnota z listu stromu (výsledná třída).

Samotná klasifikace je tedy intuitivní proces. To, čím se u klasifikátoru rozhodovacích stromů především zabýváme, je proces učení. Ten je založen na metodě *rozděl a panuj*. Množina trénovacích dat je opakovaně rozdělována na menší podmnožiny vytvářející uzly stromu tak, aby v nich převládala jedna třída. Na začátku tedy máme všechny data pohromadě a na konci listy tvořené vždy jednou třídou.

Algoritmus reprezentující tento postup se nazývá *down induction of decision trees* (TDIDT).

Algoritmus 4.4.1 algoritmus TDIDT:

1. *zvol jeden atribut jako kořen dílčího stromu,*
2. *rozděl data v tomto uzlu na podmnožiny podle hodnot zvoleného atributu a přiděl uzel pro každou podmnožinu,*
3. *existuje-li uzel, pro který nepatří všechna data do stejné třídy, pro tento uzel opakuj postup od bodu 1, jinak skonči.*

Jedná se o základní algoritmus, který bývá často v reálných implementacích upravován. Klíčovým prvkem algoritmu je otázka, jak správně vybrat atribut pro větvení stromu. Cílem je nalézt takový atribut, který od sebe odliší příklady různých tříd. Pro tuto volbu se využívá informace o entropii, informačním zisku a dalších znalostí z teorie informace.

Protože se u TDIDT často setkáváme s vytvořením příliš košatého stromu, přeučením a následnými problémy při klasifikaci, zavádíme pojem prořezávání stromu. Tento algoritmus vhodně redukuje úplný strom a snaží se dosáhnout co nejlepšího poměru úspěšnosti klasifikace k velikosti stromu.

Rozhodovací stromy bývají využívány u úloh reprezentovaných hodnotami atributů, při zatížení dat šumem nebo když data neobsahují všechny hodnoty [1],[2].

4.5 Boosting

Jedna z nejmladších metod založená na kombinaci více slabších klasifikačních modelů do jednoho. Tato metoda vznikla jako odpověď na otázku, který z klasifikátorů je nejlepší. Při klasifikaci pak všechny modely váženě hlasují o jakou třídu se jedná.

Při procesu učení se postupně vytvářejí modely s rostoucí vahou hlasu. Každý nový model je ovlivněn modely předcházejícími a zaměřuje se na příklady, které se zatím nepodařilo správně klasifikovat. Na začátku mají všechny příklady stejnou hlasovací váhu. Na základě výsledků klasifikace se váhy mění. Úspěšně klasifikovaným se váha sníží, neúspěšným zvýší. Tento postup umožňuje od sebe odlišit složité a méně složité příklady.

Nejznámějším příkladem boostovacích metod je algoritmus *AdaBoost*. Tento klasifikátor umožňuje řešit pouze dvoutřídní problémy a předpokládá použití stejného algoritmu u všech modelů.

Algoritmus 4.5.1 algoritmus AdaBoost:

Učení

1. *přiřaď stejnou váhu všem trénovacím příkladům,*
2. *pro každou iteraci (vytvářený model)*
   (a) *vytvoř model*
   (b) *spočítej chybu Err na vážených datech*
   (c) *pokud $Err = 0$ nebo $Err \geq 0{,}5$ skonči*
   (d) *pro každý příklad pokud klasifikace je správně, potom váha $w := w\frac{Err}{1-Err}$*
   (e) *normalizuj váhy příkladu (součet nových vah stejný jako součet původních)*

Klasifikace jednoho příkladu

1. *přiřaď váhy 0 všem třídám*
2. *pro každý model přiřaď třídě určené modelem váhu $w := w - \log\frac{Err}{1-Err}$*
3. *vydej třídu s nejvyšší váhou*

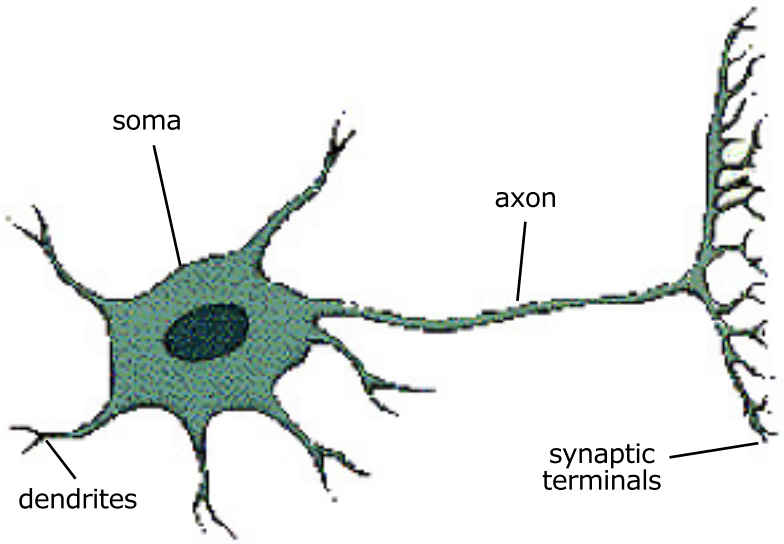
U boostovacích metod se nabízí otázka, kterému modelu více věřit. Vážené hlasování předpokládá, že všechny klasifikátory dávají stejné výsledky, ale je tomu skutečně tak? Protože pokud není, hlasováním špatných modelů můžeme získat horší výsledek než použitím jednoho průměrného. Otázkou spolehlivosti jednotlivých modelů se zabývá metoda *stacking* [1],[2].

4.6 Neuronové sítě

Neuronové sítě jsou jedním z hojně používaných klasifikátorů v moderním počítačovém vidění a nejen tam. Vznikly inspirací funkce lidského mozku, který je tvořen více jak deseti miliardami neuronů. Neuron se stal později základním kamenem pro matematické modely neuronových sítí. Zjednodušený model biologického neuronu je na obrázku 4.2. Tady můžeme vidět, že se skládá z těla (*soma*), ze kterého vybíhají výběžky (*dendrity*) a jeden delší výčnělek *axon*. Axon je ukončen jedním až několika terminály (*synapse*), které slouží jako spojení na další neurony. Přes tato spojení se šíří vzruchy, které aktivují další neurony. Tuto schopnost synapsí šířit vzruchy nazýváme *excitací* a opačně schopnost tlumit vzruchy *inhibicí*.

Matematický model neuronu přímo vychází z biologického. Jedním z těchto modelů je i *Adeline*, Widrowův adaptivní lineární neuron. Představit si jej můžeme jako black box, jehož vstupem jsou numerické hodnoty $x_1, \ldots, x_n$ a výstupem je číslo 0 nebo 1. Uvnitř Adeline je pak systém vah $w_1, \ldots, w_n$, kterými jsou vstupní hodnoty násobeny. Nad nově získanými hodnotami je proveden vážený součet a nově získaná suma je porovnána s vahou w_0. Pokud je suma větší než w_0, je výstup roven 1 a pokud je menší než w_0, je roven 0.

Modelů je velké množství a mohou nabývat obecně libovolných hodnot z daného intervalu. Těchto hodnot bývá dosaženo přenosovými funkcemi jako:



Obrázek 4.2: Zjednodušený model biologického neuronu [12]

sigmoidání funkce $f(SUM) = \frac{1}{1+e^{-SUM}}$, výstup pak nabývá hodnot z intervalu $[0, 1]$

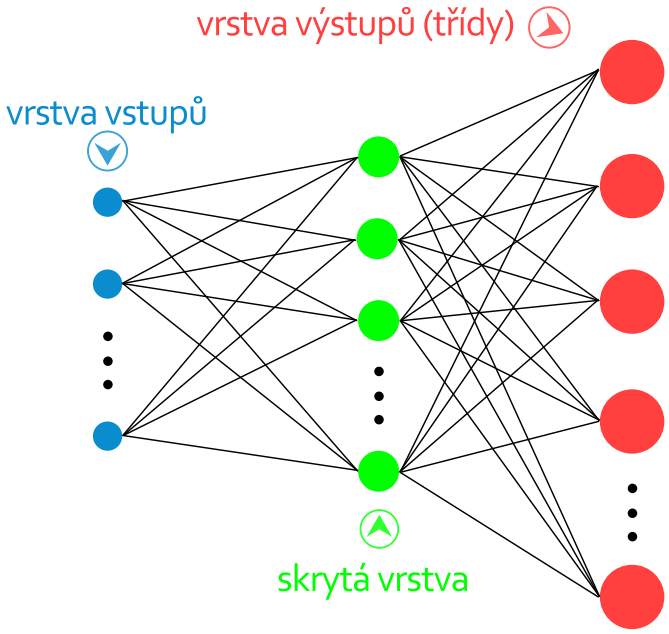
nebo

hyperbolický tangens $f(SUM) = tgh(SUM)$, pak získáme hodnoty z intervalu $[-1, 1]$.

Poté co byl popsán model jednoho neuronu byl jen pomyslný krok ke spojení více neuronů do neuronové sítě. První sítí byl *Roseblattův Perceptron*, který byl navržen jako model zrakové soustavy. Je ironií, že tento krok kupředu byl díky oprávněné námitce M. Minskeho a S. Paperta, že Perceptron si není schopen poradit s tak jednoduchou funkcí jako je XOR, zároveň důvodem stagnace vývoje neuronových sítí až do poloviny 80. let, kdy se podařilo dokázat, že libovolnou spojitou funkci lze s libovolnou přesností aproximovat pomocí třívrtsvé sítě, která je tvořena n neurony na vstupu, $2n + 1$ neurony ve druhé tzv. skryté vrstvě a m neurony ve výstupní vrstvě.

Současnou asi nejpoužívanější topologií neuronové sítě je *vícevrstvá síť* (multi-layer perceptron) 4.3, v níž každý neuron dané vrstvy je napojen na všechny neurony vrstvy následující. Klasicky se s touto topologií setkáme v podobě sítě s jednou skrytou vrstvou vycházející z jednoduchého perceptronu, kdy přejímá schopnost aproximovat libovolnou spojitou funkci.

Pro natrénování této sítě se používá algoritmus *zpětné propagace* (zpětné šíření chyby, error backpropagation). Cílem algoritmu je změnou hodnot jednotlivých vah dosáhnout stavu, kdy je síť schopna klasifikovat vstupní data s co nejmenší chybou. Toho se snaží dosáhnout opakovaným přepočítáváním chyby definované jako druhá mocnina rozdílu mezi skutečným a očekávaným výstupem sítě. Název *zpětná propagace* je odvozen od směru postupu algoritmu, protože nejprve je vypočítána chyba na výstupu, a to pak umožní spočítat chybu na skryté vrstvě následovanou změnou jednotlivých vah u neuronů. Konec algoritmu je dán jednou z ukončovacích událostí jako ustálení chybové funkce (v minimu), dosažení určitého počtu iterací nebo pokles chybové funkce pod určitou hranici.



Obrázek 4.3: Topologie více vrstvého perceptronu [7]

Jak je z algoritmu patrné, neuronová síť během učení provádí množství iterací a to může být při velkém množství trénovacích dat časově náročné.

Neuronová síť má tedy v zásadě dvě činnosti. První je učení, kdy se na základě trénovací sady naučí klasifikovat daný problém a druhou je vykonávání této naučené schopnosti, kdy je paměť této sítě reprezentována váhami, které byly během učení nastaveny na určité hodnoty. Takto naučená síť má pak schopnost rozpoznávat jak trénovací data, tak i data reálná, díky vlastnosti generalizace [1],[2].

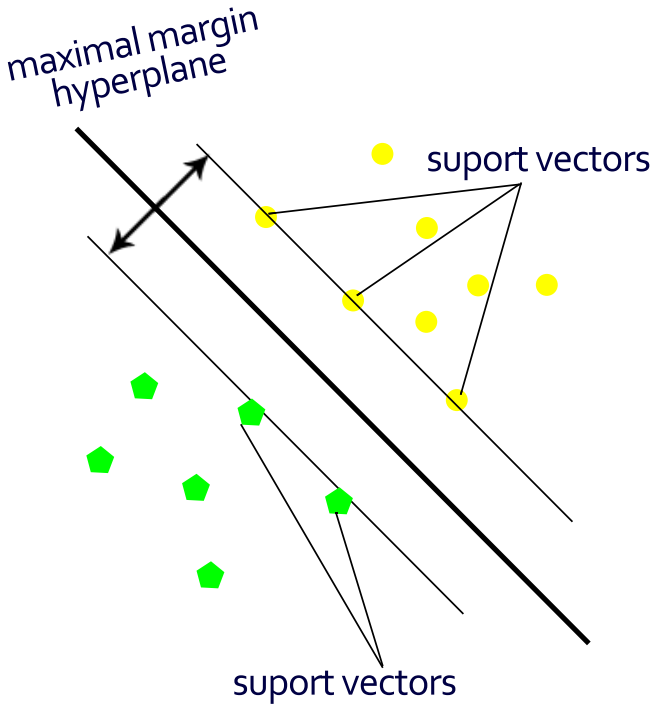
4.7 SVM

SVM (*Support Vector Machine*) je v oblasti počítačového vidění mladou metodou klasifikace. Zatím co neuronové sítě hledají takové nastavení vah, aby se co nejvíce přiblížily optimálnímu řešení klasifikace, SVM se snaží řešení explicitně vypočítat. Tato metoda přišla s myšlenkou nesnažit se nalézt optimální řešení ve složité úloze, kdy není možné nalézt lineární separátor, ale transformovat vstupní atributy tak, aby bylo možné řešit problém v lineární rovině.

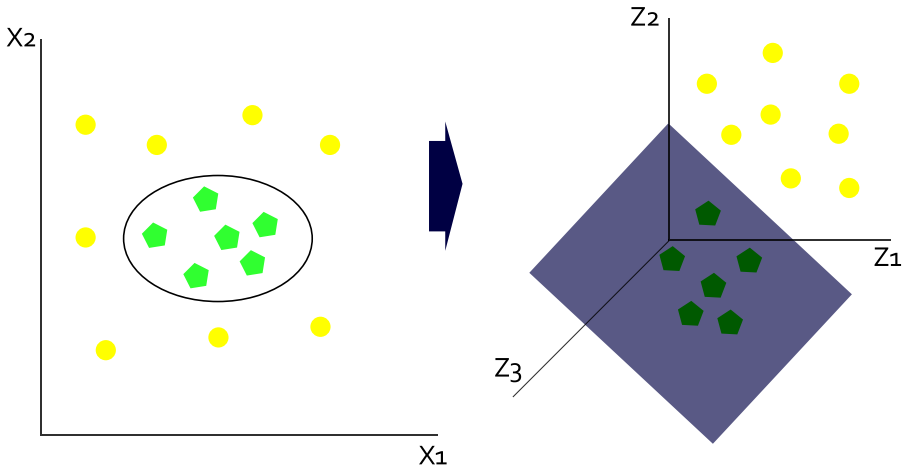
Tímto přístupem se sice zvětšuje dimenze vstupů, ale získáváme mnohem jednodušší úlohu na řešení. Když tedy získáme transformované vstupy, hledáme v dané dimenzi rozdělující nadrovinu, která má největší odstup (*maximal margin hyperplane*) od trénovacích dat 4.4.

Díky matematickým výpočtům, na kterých je SVM založena, můžeme brát v potaz jen ty data, která jsou nejblíže hledané hranici. Podle těchto podpůrných vektorů (*support vectors*) dostala metoda své jméno [1],[2].

Příklad transformace vstupních dat můžeme vidět na obrázku 4.5.



Obrázek 4.4: Maxim margin hyperplane[7]



Obrázek 4.5: Transformace do vícerozměrného prostoru a nalezeni oddělovací hranice [7]

Kapitola 5

Návrh a implementace

Jednou z částí této bakalářské práce je i vytvoření jednoduchého OCR systému pro platformu mobilního operačního systému iOS od firmy Apple. Tento systém si klade za cíl rozpoznávání identifikačního kódu samoobslužného skladu v prodejnách nábytku IKEA.

K implementaci aplikace jsou použity programovací jazyky *C++* a *Objective-C 2.0* a knihovna pro zpracování obrazu *OpenCV*. Politika firmy Apple implicitně neumožňuje vývoj software pro jejich výrobky na jiných platformách než jsou jejich vlastní. Proto je nutné mít potřebné hardwarové vybavení. Především počítač s operačním systémem OS X, na který je pak možné si zdarma stáhnout vývojové prostředí *XCode*.

5.1 Návrh

Jednou z částí správné tvorby programů je návrh samotné této procedury. Právě určením jak postupovat, aby nakonec hotový program jako celek správně pracoval, se bude zabývat tato část bakalářské práce.

5.1.1 Parametry mobilního telefonu

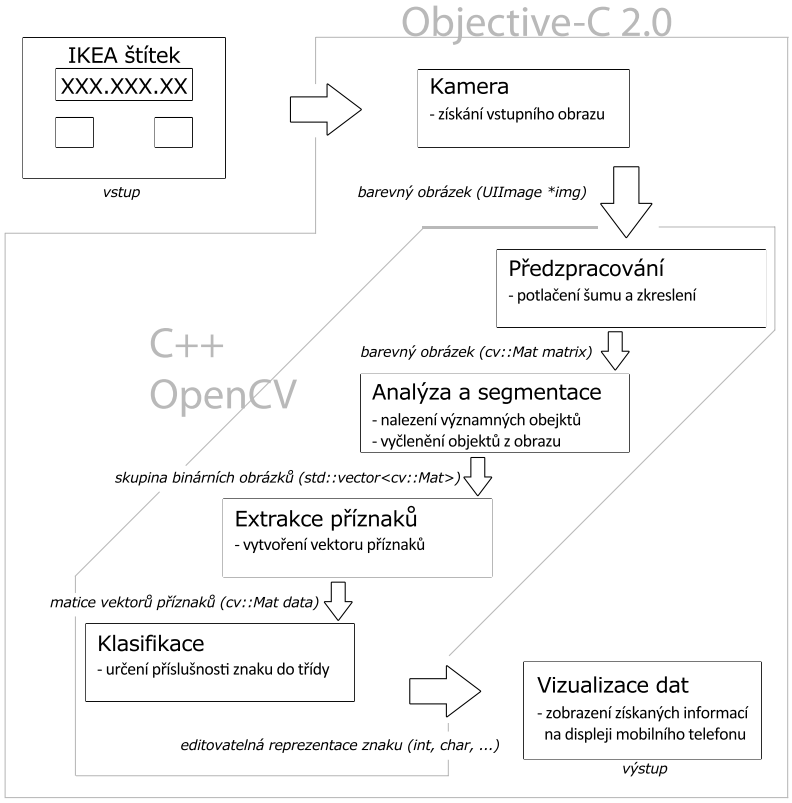
Při návrhu aplikace pro mobilní zařízení je nutné počítat s jistými hardwarovými omezeními. Jako výchozí zařízení byl stanoven model *iPhone 3G* s následujícími důležitými parametry pro vývoj:

Procesor	412 MHz
Paměť	128 MB
Disk	8 GB (flash)
Kamera	2.0 megapixel
OS	iOS v 4.0.1

Tabulka 5.1: Tabulka důležitých parametrů pro iPhone 3G

5.1.2 Zpřesnění požadavků na aplikaci

Pro správný postup implementace bylo důležité zpřesnit požadavky na výslednou aplikaci, jakož i upřesnit co je očekávaným vstupem aplikace a jaké má vlastnosti důležité pro jednotlivé kroky OCR systému.



Obrázek 5.1: Schéma postupu vývoje

Jako vstup je tedy očekáván snímek IKEA štítku, který se nachází u vybraných kusů nábytku v obchodním domě IKEA, a který určuje jeho lokaci v samoobslužném skladu. Tento štítek má následující, pro aplikaci důležité, parametry:

- Tvar štítku je obdélník o vždy stejném poměru velikosti stran.
- Barva štítku je odstínem červené.
- Jednotlivé prvky na štítku (nápisy, čísla, obdélníková pole) mají černou barvu.
- Počet rozeznávaných číslic je vždy osm.
- Číslice jsou umístěny v obdélníkovém poli o vždy stejném poměru stran.

Jako výstup aplikace se pak očekává osmice čísel v editovatelném tvaru, přičemž na displeji mobilního telefonu by uživatel měl vidět jak snímek ze kterého byla data získána, tak i osm obrázků, z nichž každý obsahuje jedno číslo. Dále by měl mít možnost zadat název nábytku a počet kusů, které si bude chtít ze skladu vyzvednut. Po stisku odpovídajícího tlačítka se získané informace uloží do seznamu, který si může uživatel kdykoliv zobrazit.

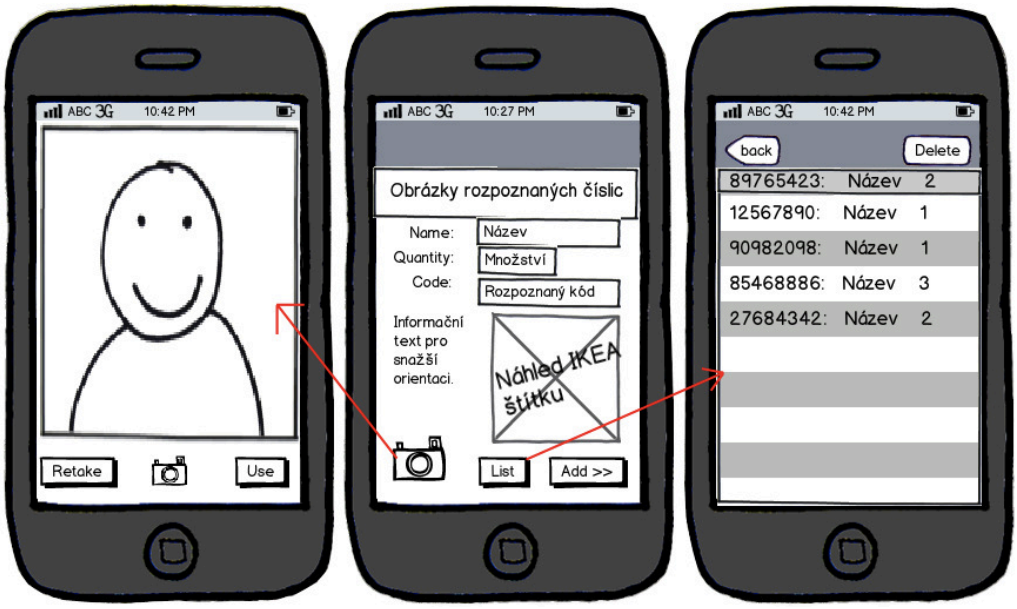
5.1.3 Nástroje pro implementaci

Při návrhu bylo dále důležité zamyslet se nad jednotlivými nástroji, které budou při implementaci použity a zjistit, zda s sebou nepřináší případné komplikace. Jazykem implicitně

používaným k vývoji na platformě *iOS* je *Objective–C*. Pro tento jazyk je vývoj přímo přizpůsoben a proto není pravděpodobné, že by s sebou přinášel jakékoliv problémy. Tento jazyk naneštěstí není podporován knihovnou *OpenCV*, která je stěžejním prvkem celé aplikace. Knihovna v použité verzi 2.3.2 však obsahuje plnou podporu pro jazyk *C++*, se kterým jde bez větších obtíží na platformě *iOS* také pracovat a kombinovat jej s *Objective–C*. Dalším problémem je to, že *OpenCV* není oficiálně platformou *iOS* podporována, a je proto nutné ji zkompilovat. Pokud nechceme knihovnu samostatně kompilovat, lze využít některé z předem zkompilovaných předpřipravených verzí, které se dají dohledat na internetu. Pak s knihovnou pracujeme jako s frameworkem, dodatečná konfigurace (kromě importu u tříd, kde ji chceme použít) není nutná, pouze prostým přetažením přidáme OpenCV mezi ostatní frameworky. Posledním významným problémem spojeným s knihovnou *OpenCV* je její vlastní spolupráce s podporovaným formátem obrázku získaného kamerou telefonu *UIImage*. Pro převod z tohoto formátu do *cv::Mat* podporovaného *OpenCV* je potřeba vlastní kód.

5.1.4 Grafický návrh

Pro lepší představu a utřídění všech potřebných aspektů dalších kroků nutných k vývoji aplikace byl za pomocí softwaru *Balsamiq Mockups* vytvořen návrh uživatelského rozhraní pro mobilní telefon iPhone. Na obrázku 5.2 (uprostřed) je vidět hlavní pohled aplikace, který obsahuje základní ovládací prvky. Ikona fotoaparátu slouží k přepnutí na pohled starající se o získání snímku IKEA štítku (vlevo). Tlačítko *List* přepne pohled na tabulku obsahující informace pro lokalizaci jednotlivých kusů nábytku ve skladu obchodního domu IKEA (vpravo). Tlačítko *Add* přidá název, množství a rozpoznaný kód do tabulky.



Obrázek 5.2: Grafický návrh uživatelského rozhraní

Základní model chování očekávaný od uživatele v sobě zahrnuje čtyři jednoduché kroky:

1. Uživatel se nachází v hlavním pohledu aplikace. Klikne na *ikonu fotoaparátu*, která přepne obrazovku na pohled umožňující získat snímek IKEA štítku.

2. Uživatel stiskne tlačítko *Use*, čímž dá signál, že snímek obsahuje štítek a započne proces vedoucí k nalezení a získání kódu ze štítku. Současně je přesunut zpět na hlavní pohled a vidí nalezená čísla získaná při segmentaci a kód získaný klasifikací.

3. Uživatel doplní pole název a množství následované stiskem tlačítka *Add*, které přidá údaje do seznamu.

4. Stisk tlačítka *List* přepne pohled na tabulku obsahující všechny informace obsažené v seznamu.

Tento model v sobě zahrnuje jen základní chování předpokládající pozitivní nalezení informací ze snímku. V případě, že dojde k nestandardní situaci, je uživatel informován prostřednictvím *informačního textu*, který nabídne další postup a řešení situace.

5.1.5 Postup vývoje

Další fází návrhu je schéma postupu vývoje, které lze vidět na obrázku 5.1. Toto schéma přímo vychází z obecného diagramu OCR systému, který je ilustrován v úvodu dokumentu. Nová verze je přizpůsobena specifickým požadavkům mobilního telefonu. V další části implementace jsou blíže popsány jednotlivé kroky vývoje.

5.1.6 Příprava testů

Poslední fází návrhu je příprava očekávaných testů jak jednotlivých částí OCR systému, tak konečné úspěšnosti klasifikace. V části testování níže by se tak mělo objevit testování:

- Rychlost aplikace (segmentace objektů, klasifikace dat).
- Úspěšnost (nalezení dat k segmentaci, klasifikace dat).

V této části jsou blíže popsány jednotlivé trénovací a testovací sady.

5.2 Implementace

Prvním krokem nutným pro veškerou práci aplikace je získání snímku pro další zpracování. Velkou výhodou platformy iOS je její API rozhraní, které usnadňuje manipulaci s kamerou telefonu. Veškerou prací programátora je tedy pouze nastudovat a použít třídu *UIImagePickerController*. Jakmile je získán vhodný snímek, uloží se do proměnné *lastFrame*, která slouží jako předloha originálního obrázku pro ostatní třídy.

5.2.1 Předzpracování, analýza a segmentace

O veškeré operace nutné pro předzpracování, analýzu a segmentaci dat ze snímku se stará třída *Preprocessing*.

Přístup, který jsem si zvolil, je předpokládat v každém získaném obraze určité množství šumu. Tento přístup není bezchybný, protože se může stát, že do velmi kvalitního snímku sám vnese menší množství šumu, ale tento zápor převyšuje přínos v podobě zlepšení kvality středních a nekvalitních snímků, kterých je obecně více.

V prvním kroku zpracování se snažím nalézt štítek za pomocí vhodně zvoleného rozsahu barev. Využívám při tom specifické barvy štítku IKEA. Pro zlepšení síly rozeznání převádím

dočasně implicitní BGR reprezentaci barev do HSV reprezentace. Výstupem této operace je binární obraz obsahující pouze ty data, která se nacházejí v určeném rozsahu.

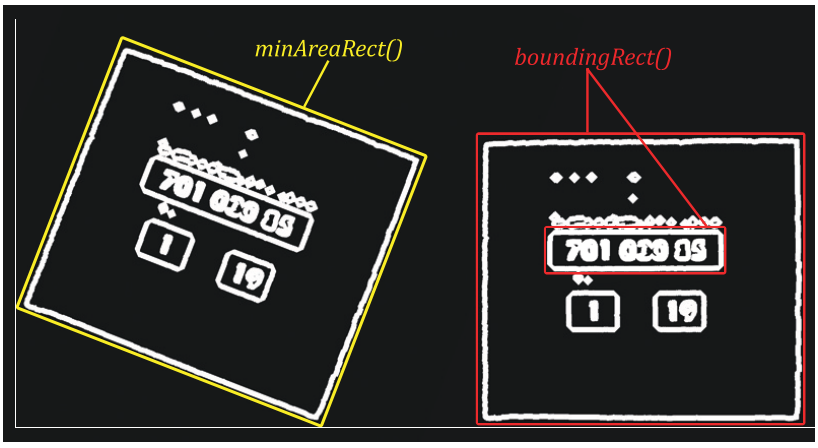
Dále provádím nalezení hran. Původně jsem používal Cannyho hranový detektor, ale později jsem jej nahradil přístupem s odečtením erodovaného obrazu od původního, jak bylo popsáno v 2.2.2.

Jakmile jsou získány hrany, pokračuji nalezením kontur. K tomu využívám metodu *findContours* z knihovny OpenCV, která umožňuje určit hierarchii, ve které jsou rozeznané kontury vráceny. Použitá stromová struktura dovoluje zjistit, zda ta která kontura má svého předka, následovníka, případně i kolik jich má.

V dalším kroku metodou *minAreaRect* z knihovny OpenCV, která nalezne nejmenší možný pravoúhlý čtyřúhelník (libovolně natočený) kolem zadaného objektu, obalím vhodně zvolené kontury. Za vhodně zvolenou konturu v tomto případě považuji takovou, která nemá žádného předka a současně má nejméně jednoho následovníka, tento popis totiž sedí na okrajovou linii kolem štítku. V ideálním případě by byla nalezena jen jedna taková kontura, ale musíme počítat s možností šumu, který by mohl být interpretován jako jiná kontura se stejnými parametry. Informace z těchto čtyřúhelníků umožní zjistit velikost, poměr stran a úhel natočení nalezených objektů. Tyto hodnoty používám jako ukazatele, zda se skutečně jedná o štítek. Pokud ano, tak úhel natočení čtyřúhelníku použiji k opravě natočení.

Pokud jde vše správně (na snímku je skutečný IKEA štítek a podařilo se jej rozeznat), mám v této části implementace správně natočený snímek a vím, že obsahuje štítek s informacemi potřebnými pro klasifikaci.

Další postup kopíruje předchozí kroky až po obalení čtyřúhelníky. Toto obalení se sice provede, ale čtyřúhelníky již nejsou natočitelné (metoda *boundingRect* z knihovny OpenCV). Z informací z hierarchie kontur a vlastností čtyřúhelníků se snažím získat obdélník obsahující čísla udávající polohu hledaného objektu ve skladu a čtyřúhelník kopírující obvod štítku. Na obrázku 5.3 můžeme vidět ilustraci obalení štítku čtyřúhelníky. Pokud je nalezen obdélník s čísly, pokračuje algoritmus dále k segmentaci čísel, pokud se ale nepodařilo tento obdélník nalézt, vystřihne ze snímku pouze část s nalezeným štítkem a pokusí se o opětovné nalezení obdélníku s čísly později, po aplikování operací jako *eroze* a *dilatace* za účelem odstranění pravděpodobného šumu.



Obrázek 5.3: Ilustrace obalení významných kontur štítku čtyřúhelníky a oprava natočení na základě jejich vlastností

Tento postup sice zvyšuje náročnost na hardware a prodlužuje dobu nalezení, ale zvyšuje

pravděpodobnost nalezení čísel ke klasifikaci.

Pokud se podařilo nalézt obdélník s čísly, přecházíme k segmentaci znaků. Původně jsem postupoval vytvořením horizontálního a vertikálního promítnutí a za rozdělující úseky znaků jsem považoval lokální minima. Tento postup se však ukázal jako neefektivní a to hned ze dvou důvodů. Prvním byla nízká kvalita vstupního obrazu, kdy docházelo ke spojování znaků a následnému špatnému označení dvou případně i více znaků jako jeden. Druhým problémem bylo horizontální promítnutí znaku nula, který při snížené kvalitě vstupního obrazu vytvořil uprostřed promítnuté oblasti takové minimum, které bylo považováno za mezeru. Proto jsem segmentaci doplnil o vyplňování uzavřených oblastí a kontrolu výšky a šířky znaku. Tento postup je symbolicky demonstrován na obrázku 5.4.

Každý takto získaný znak je pak vystřihnut a normalizován na velikost 20×20 px. Takto musí být získáno osm znaků, které jsou předány ve formátu vektoru obsahujícího jednotlivé matice dál ke zpracování.

Obrázek 5.4: Segmentace znaků

5.2.2 Extrakce příznaků a klasifikace

Na základě informací z knihy [2], která uvádí klasifikátor na principu neuronových sítí *multi-layer perceptron* z knihovny *OpenCV* jako vhodný pro rozpoznávání znaků, jsem se rozhodl jej využít.

Jako příznak jednoho znaku je použit vektor o rozměru 400, který je reprezentován jako jednokanálová matice, kdy každý řádek reprezentuje jeden příznakový vektor. Příznakový vektor je pak získán jako posloupnost po sobě jdoucích obrazových bodů z jednoho znaku (20×20 px). Každá z 400 proměnných nabývá hodnoty buď 1 nebo 0, podle hodnoty pixelu v binárním obrázku. Tento přístup jsem zvolil na základě práce *Tobyho Breckona* z Cranfieldské univerzity [3].

Všechny metody použité při získávání příznaků a klasifikaci jsou implementovány ve třídě *MachineLearning*. Pro natrénování klasifikátoru jsou použita reálná data, tedy fotografie štítků z obchodu IKEA, pořízené kamerou iPhoneu. Z těchto fotografií jsem získal 200 vzorků pro natrénování klasifikátoru.

Pro urychlení práce s trénovacími a testovacími daty jsem implementoval několik metod speciálně k tomuto účelu. Metoda *getFeaturesVectorFromMat...* získá příznakový vektor z obrázku, metoda *writeDataTocsvName...* uloží tento vektor do *CSV* souboru a metoda *readDataFromcsvName...* je pak dokáže rychle načíst. Tato technika je vhodná především při opakovaném testování úspěšnosti klasifikace, protože odstraňuje nutnost zpracovávání obrázků až po fázi získávání příznaků.

Po naučení klasifikátoru bylo jeho nastavení uloženo do *XML* souboru, který je načten a používán při klasifikaci na reálných datech.

5.2.3 Vizualizace dat

Uživatelské rozhraní je realizováno prostřednictvím nástrojů prostředí XCode. Ihned po zpracování snímku se zobrazí osm obrázků obsahujících znaky ke klasifikaci. Vizuální vzhled lze vidět na obrázku 5.5. Posloupnost jednotlivých pohledů simuluje průchod aplikací.

Obrázek 5.5: Vizuální vzhled aplikace

5.3 Testování aplikace

Testování implementované aplikace probíhalo v několika fázích. V první řadě byla testována rychlost aplikace na mobilním telefonu, poté úspěšnost rozeznání znaků v obraze a nakonec úspěšnost klasifikace znaků.

5.3.1 Rychlost aplikace

Pokud jde o rychlost, s jakou je schopna aplikace rozeznat štítek a vyseparovat z něj oblast potřebných dat, dostává se v průměru na $13,56$ *s*. Tato hodnota není nijak vysoká, ale už se nedá mluvit o real-time aplikaci. Samozřejmě je nutné vzít v úvahu, že model *iPhone 3G*, na kterém byla aplikace testována, je již zastaralý. Jeho procesor o frekvenci 412 MHz, ve srovnání například s modelem *iPhone 4S* osazeným *dvou-jádrovým procesorem Apple A5* o *frekvenci 800MHz*, je pomalý. Tabulka 5.2 obsahuje vybrané výsledky testování rychlosti mobilního telefonu iPhone v modelech 3G, 3GS, 4 a 4S. Testování bylo provedeno aplikací *Geekbench 2* uživateli z celého světa, kteří odeslali své výsledky na webové stránky aplikace pro srovnání.

	3G	**3GS**	**4**	**4S**	
				single-core scalar	multi-core scalar
Img. Compress [Mpix/sec]	1.07	2.22	2.71	3.22	6.40
Img. Decompress [Mpix/sec]	1.66	3.63	4.90	5.35	10.5
Sharpen Img. [Kpix/sec]	766	908	1210	1350	2680
Blur Img. [Kpix/sec]	357	459	572	532	1060
Read Sequential [MB/sec]	130	297	316	319	–
Write Sequential [MB/sec]	144	535	737	806	–

Tabulka 5.2: Tabulka výsledků testování programem Geekbench 2

Naproti tomu klasifikace je proces nesmírně rychlý. Třívrstvá neuronová síť typu *multi-layer perceptron* je schopna klasifikovat všechna data získaná ze štítku v rozmezí 400 až 800 ms.

5.3.2 Rozeznání znaků

Úspěšnost rozeznání znaků (osmi číslic) na štítku je shrnuta v tabulce 5.3 níže. Aby bylo možné si udělat představu o významu dat, je nutné si představit trénovací a testovací sady, na kterých bylo testování provedeno. Tyto testovací vzorky byly postupně převedeny do podoby *CSV* souborů, které obsahují na každém řádku vektorový příznak a jako poslední znak třídu, do které vektor patří.

Sada A: Tato sada se skládala ze snímků focených z různých vzdáleností, s proměnlivým natočením a na mnoha barvách a druzích pozadí. Jedná se, co do rozeznání, o nejsložitější vzorek. Sada A byla z části (200 znaků, pro každou třídu 20 vzorků) použita v trénovací sadě a z části (216 znaků) použita v primární testovací sadě. Žádný z 416 vzorků není kopií jiného.

Sada B: Tato sada je tvořena snímky štítků focených ze vzdálenosti ne větší než 15 cm, s natočením do 5 stupňů a s jednoduchým, většinou jednobarevným, pozadím.

Sada C: Tuto sadu tvoří snímky focené ze vzdálenosti od 20 do 30 cm, jinak se jedná o stejné vzorky jako u sady B.

Sada D: Sada D je tvořena snímky se štítky natočenými doleva o úhel v rozmezí 5 až 30 stupňů, jsou foceny ze vzdálenosti 10 až 15 cm. Opět se jedná o stejné vzorky jako u sad B a C.

Sada E: Sada E je tvořena snímky se štítky natočenými doprava o úhel v rozmezí 5 až 30 stupňů, jsou foceny ze vzdálenosti 10 až 15 cm. Opět se jedná o stejné vzorky jako u sad B, C a D.

Sada	**Množství vzorků**	**Rozeznaných**	**Nerozeznaných**	**Úspěšnost**
Sada A	65	52	13	80,00 %
Sada B	59	57	2	96,61 %
Sada C	59	49	10	83,05 %
Sada D	59	52	7	88,14 %
Sada E	59	54	5	91,53 %

Tabulka 5.3: Tabulka rozezání znaků z testovacích sad

5.3.3 Klasifikace znaků

Úspěšnost klasifikace znaků byla testována na sadách získaných během fáze rozeznání znaků doplněných o znaky tzv. falešně nerozeznané, kdy bylo plně rozeznáno osm znaků a jeden navíc jako oddělující tečka mezi číslicemi, která byla manuálně odstraněna. Ze sady *A* bylo odebráno 200 vzorků, od každé třídy (0 až 9) pro trénovací sadu. Jak je vidět z tabulky 5.4, neuronová síť dosahuje výborných výsledků na testovacích datech ze skupiny focené z blízka (v takové vzdálenosti od objektivu, aby byl štítek na displeji vidět celý a současně se blížil okrajům displeje - sada B), ale rovněž ze skupin focených z blízka a k tomu s různým natočením (sady C a D). U fotek focených ve vzdálenosti vyšší (sada C) dochází k menšímu poklesu schopnosti správně klasifikovat, což je pravděpodobně způsobeno horší kvalitou vstupních dat (snímků). Zde se opět projevují nevýhody zastaralého modelu 3G, který oproti novějším není vybaven tak kvalitní kamerou a především funkcí automatického zaostření.

		Klasifikováno	
Sada	**Množství vzorků**	**Korektně**	**Chybně**
A	219	218 (99,543%)	1 (0,457%)
B	464	462 (99,569%)	2 (0,431%)
C	344	340 (98,837%)	4 (1,163%)
D	424	422 (99,528%)	2 (0,472%)
E	431	431 (100%)	0 (0,000%)

Tabulka 5.4: Tabulka úspěšnosti klasifikace z testovacích sad

V další tabulce 5.5 je vidět rozložení falešně pozitivní klasifikace u jednotlivých tříd všech testovacích sad. Z tabulky můžeme vidět, že chyby se výrazněji neopakují a i celková úspěšnost klasifikace je vysoká.

		Sada				
	Třída	**A**	**B**	**C**	**D**	**E**
Falešně pozitivní	0	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	1	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	2	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	3	1 (0,457%)	0 (0%)	0 (0%)	1 (0,236%)	0 (0%)
	4	0 (0%)	0 (0%)	1 (0,291%)	0 (0%)	0 (0%)
	5	0 (0%)	0 (0%)	1 (0,291%)	1 (0,236%)	0 (0%)
	6	0 (0%)	1 (0,216%)	0 (0%)	0 (0%)	0 (0%)
	7	0 (0%)	1 (0,216%)	0 (0%)	0 (0%)	0 (0%)
	8	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	9	0 (0%)	0 (0%)	2 (0,581%)	0 (0%)	0 (0%)

Tabulka 5.5: Tabulka falešně pozitivní klasifikace u jednotlivých tříd daných sad

Kapitola 6

Závěr

V rámci této práce byly popsány metody pro detekci a rozpoznání znaků v obraze a diskutována jejich použitelnost na mobilní platformě iOS. Dále byl implementován jednoduchý OCR systém, jehož účelem je rozpoznání identifikačního kódu samoobslužného skladu v prodejnách nábytku IKEA.

Během práce jsem prostudoval metody pro zpracování obrazu, segmentaci objektů z obrazu, metody klasifikace a naučil jsem se pracovat v prostředí XCode na vývoj aplikací pro mobilní platformu iOS.

Za pomoci získaných znalostí jsem vybral vhodné metody pro každou část vývoje mnou implementované aplikace a tím se mi podařilo dosáhnout dobré úspěšnosti segmentace a klasifikace OCR systému.

Při programování jsem využíval především již implementovaná řešení z knihovny OpenCV (multi-layer perceptron, neuronová síť), ale naproti tomu segmentaci znaků řeší můj vlastní algoritmus popsaný v kapitole 5.2.1.

Výsledky testování aplikace potvrdily předpokládané problémy mající vliv na úspěšnost jak segmentace, tak pozdější klasifikace dat. Je to především snížená kvalita vstupního obrazu.

Jako možné rozšíření aplikace, by mohlo být zavedení paralelní lokace IKEA štítku v obraze jiným způsobem než na základě rozsahu barev, který se například při vyfocení snímku na pozadí podobné barvy nemusí podařit rozeznat. S touto změnou však vyvstává problém zvýšení výpočetního nároku na mobilní zařízení a s tím spojené prodloužení doby segmentace.

Mnou implementovaný jednoduchý OCR systém se komerčním OCR systémům nevyrovná především v rozsahu rozeznávaných znaků, nicméně požadavky této bakalářské práce, a tedy i účel pro který byl vytvořen, splňuje.

Literatura

[1] Berka, P.: *Dobývání znalostí z databází*. Praha: Academia, 2003, ISBN 80-200-1062-9.

[2] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library*, ročník 1. OReilly Media, 2008, 555 s.

[3] Breckon, T.: Machine Learning : MSc Course [online]. 2012 [cit. 2012-04-28].
URL http://public.cranfield.ac.uk/c5354/teaching/ml/

[4] Canny, J.: A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 8, č. 6, 1986: s. 679–698.

[5] Chan, T. F.; Vese, L. A.: Active contours without edges. *IEEE Transactions on Image Processing*, ročník 10, č. 2, 2001: s. 266–77.

[6] Chang, F.: A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, ročník 93, č. 2, 2004: s. 206–220.

[7] Hastie, T.; Tibshirani, R.; Friedman, J.: *The Elements of Statistical Learning*, ročník 18. Springer, 2001, 764 s.

[8] Horák, K.: Matematická morfologie [online]. 2010 [cit. 2011-11-16].
URL http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/mpov.html

[9] Horák, K.; Kalová, I.; Petyovský, P.; aj.: Počítačové vidění [online]. 2008 [cit. 2011-11-18].
URL http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/mpov.html

[10] Jankowski, M.; Kuska, J.-P.: Connected components labeling - algorithms in Mathematica, Java, C++ and C#. In *International Mathematica Symposium*, Srpen 2004.
URL http://www.izbi.uni-leipzig.de/izbi/publikationen/publi_2004/IMS2004_JankowskiKuska.pdf

[11] Kalová, I.: Předzpracování obrazu [online]. 2010 [cit. 2011-11-16].
URL http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/mpov.html

[12] Lobb, C. J.; Chao, Z.; Fujimoto, R. M.; aj.: Parallel Event-Driven Neural Network Simulations Using the Hodgkin-Huxley Neuron Model. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, PADS '05, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2383-8, s. 16–25, doi:10.1109/PADS.2005.18.

[13] Parker, J. R.: *Algorithm for Image Processing and Computer Vision*. New York: Wiley Computer Publishing, 1997, ISBN 0-471-14056-2, 417 s.

[14] Sahoo, P. K.; Soltani, S.; Wong, A. K. C.; aj.: A SURVEY OF THRESHOLDING TECHNIQUES. *COMPUTER VISION GRAPHICS AND IMAGE PROCESSING*, ročník 41, č. 2, 1988.

[15] Singh, S. K.; Chauhan, D. S.; Vatsa, M.; aj.: A Robust Skin Color Based Face Detection Algorithm. *Science*, ročník 6, č. 4, 2003: s. 227–234.

[16] Trier, O. D.; Jain, A. K.; Taxt, T.: Feature extraction methods for character recognition-a survey. *Pattern Recognition*, ročník 29, č. 4, 1996: s. 641–662.

[17] Wu, K.; Otoo, E.; Suzuki, K.: Optimizing two-pass connected-component labeling algorithms. *Pattern Anal. Appl.*, ročník 12, č. 2, Únor 2009: s. 117–135, ISSN 1433-7541, doi:10.1007/s10044-008-0109-y.

[18] Žára, J.; Beneš, B.; Felkel, P.: *Moderní počítačová grafika*. Brno: Computer Press, a.s, druhé vydání, 2004, ISBN 80-251-0454-0, 578 s.

Příloha A

Obsah CD

Disk přiložený k této práci obsahuje tyto položky:

- **OCR_for_iOS/** – Tato složka obsahuje zdrojové kódy a soubory potřebné pro překlad a spuštění aplikace.
- **doc/** – Složka obsahuje dokumentaci ve formátu PDF.
- **sady_img/** – Složka obsahující obrázky použité na vytvoření trénovací sady a testovacích sad.
- **sady_csv/** – Složka obsahující CSV soubory použité při trénování a testování klasifikátoru.
- **poster/** – Tato složka obsahuje plakát prezentující tuto práci.

Příloha B

Plakát