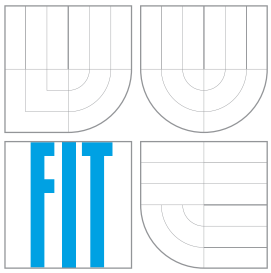
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ OBJEKTŮ V OBRAZE NA PLATFORMĚ ANDROID

IMAGE OBJECT RECOGNITION USING ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ŠIMON
AUTHOR

VEDOUCÍ PRÁCE — prof. Dr. Ing. PAVEL ZEMČÍK
SUPERVISOR

BRNO 2013

Abstrakt

Práce popisuje problematiku rozpoznávání objektů v obraze ve spojení s mobilní platformou Android. Rozebírá existující metody a postupy z oboru rozpoznávání objektů stejně jako dřívější řešení podobné problematiky. Z těchto základů čerpá a navrhuje praktickou aplikaci, která má ambice stát se schopným pomocníkem při houbaření. Při návrhu a implementaci bere v potaz také cílové zařízení, které je omezené jak výkonem, tak pamětí. Výsledky jsou vyhodnoceny na základě několika experimentů, včetně uživatelského testování. Implementace výsledné aplikace je součástí práce.

Abstract

This thesis describes an object recognition issue in connection with the Android mobile platform. It analyzes existing methods and procedures from the field of the object recognition as well as previous solutions of a similar topic. It builds on these basics and proposes a handy application which aspires to become a powerful assistant during mushroom picking. It considers the target device, which is memory and power limited, during the design and implementation phase. The results are evaluated by several experiments including an user experience. The implementation of the application itself is a part of this thesis.

Klíčová slova

Houby, Rozpoznávání, Klasifikace, Segmentace, Android, ARM.

Keywords

Mushrooms, Recognition, Classification, Segmentation, Android, ARM.

Citace

Martin Šimon: Rozpoznávání objektů v obraze na platformě Android, bakalářská práce, Brno, FIT VUT v Brně, 2013

Rozpoznávání objektů v obraze na platformě Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci na téma „Rozpoznávání objektů v obraze na platformě Android“ vypracoval samostatně pod vedením vedoucího bakalářské práce pana prof. Dr. Ing. Pavla Zemčíka a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

.....
Martin Šimon
14. května 2013

Poděkování

Velký dík patří zejména serveru Nahuby.sk za poskytnuté fotografie hub. Tímto bych chtěl poděkovat nepřímo i všem autorům zmíněných fotografií. Dále bych chtěl poděkovat vedoucímu práce, panu profesoru Pavlu Zemčíkovi, a to za jeho velké nasazení, ochotu a především podporu ve všech směrech, do kterých se tato práce vydala. Děkuji také Pavlíně Kotačkové za jazykové korektury.

© Martin Šimon, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Mobilní telefony dnes již neslouží pouze k telefonování či psaní zpráv. Stávají se z nich plnohodnotná multimediální zařízení umožňující například sledovat video ve vysokém rozlišení nebo hrát náročné hry. Jejich skutečný výkon jde však ještě dál. Operace, jako zpracování obrazu a rozpoznání objektů v obraze již často nejsou mimo možnosti těchto zařízení. Právě mobilita takových zařízení dodává těmto operacím bez nadsázky další rozměr. Tímto směrem se vydává i tato práce.

Zpracování obrazu je náročná úloha sama o sobě. Násobné průchody mnoharozměrných matic, složité operace jako je dělení, logaritmické či exponenciální funkce. Operace náročné jak z výpočetního, tak paměťového hlediska. Pokud se ke zpracování obrazu přidá ještě operace rozpoznávání a zpracovávaná data se dále identifikují a klasifikují, o složitosti snad již nemůže být pochyb.

Právě tato skutečnost však paradoxně vedla k této práci, neboť se zabývá rozpoznávání objektů v obraze na mobilní platformě Android. Pod tímto označením jsou v rámci této práce uvažována mobilní zařízení s omezeným výkonem, pamětí a mikroprocesorem s instrukční sadou ARMv7. Cílem je ověřit schopnosti těchto široce rozšířených zařízení na řešení problematiky z oboru počítačového vidění.

Cílem práce je vytvořit praktickou aplikaci, která bude běžet na mobilním zařízení s platformou Android a bude realizovat rozpoznávání objektů z obrazu. Tento obraz bude získán prostřednictvím vestavěné kamery a při práci budou maximálně využity vestavěné senzory. Tyto senzory dělají z takových mobilních zařízení multifunkční zařízení se širokým polem uplatnění.

Práce se dělí na několik kapitol. V kapitole 2 jsou rozebrány a popsány některé metody, které budou případně dále v práci použity. V žádném případě se však nesnaží pokrýt celou problematiku či zmínit všechny práce. Čtenář znalý problematiky může tuto kapitolu s klidným svědomím přeskočit.

Kapitola 3 obsahuje popis výběru vhodných metod, které byly popsány v kapitole 2. Tyto metody jsou zhodnoceny z pohledu vhodnosti, jsou navrženy některé jejich vylepšení vzhledem k cílovému zaměření této práce a je navržena samotná aplikace.

V kapitole 4 jsou popsány postupy a různé části implementace samotné aplikace. Tato kapitola obsahuje také využití znalostí získaných o optimalizaci kódu pro platformu Android a architekturu ARMv7.

Kapitola 2

Současný stav problematiky

Obsahem této kapitoly je popis problematiky a poznatků potřebných pro realizaci této práce. Kapitola je členěna na jednotlivé bloky, jejichž obsahem je popis konkrétních metod a postupů.

V podkapitole 2.1 jsou popsány základy počítačové grafiky, neboť práce v dalších částech na těchto základech staví. Podkapitola 2.2 se věnuje popisu metod a postupů z oboru počítačového vidění, které předchází samotnému rozpoznávání. Rozpoznávání, neboli klasifikace, je obsahem podkapitoly 2.3. Ta představuje některé základní způsoby klasifikace a klasifikátorů. V poslední podkapitole 2.4 jsou popsány a stručně představeny platforma a architektura, na kterou tato práce cílí.

Cílem žádné podkapitoly není utvořit encyklopedický seznam včech dílčích částí, které se k dané problematice mohou vázat. Obsahuje jsou pouze ty poznatky, které jsou ve vztahu k této práci relevantní.

2.1 Základy počítačové grafiky

Rozpoznávání objektů v obraze je jedno z mnoha témat problematiky známé jako *zpracování obrazu*. Je proto nutné dobře znát některé základní postupy a metody používané napříč celým spektrem této problematiky. Patří sem základy vektorové a rastrové grafiky, geometrické transformace bodu, použití filtrů s různými konvolučními jádry a jiné. Právě tyto základy budou obsahem této podkapitoly. V rámci práce se uvažuje pouze dvourozměrné zobrazení

Pro pochopení následujících řádků je nutné sjednotit reprezentaci bodu v prostoru. Bod ve 2D se souřadnicemi $[x, y]$ je reprezentován jako uspořádaná trojice homogenních souřadnic $[X, Y, w]$, pro kterou platí, že podělením homogenní souřadnice váhou w dostaneme vždy původní souřadnici (Rovnice 2.1a a 2.1b). Toto je velice výhodné, neboť nám to umožní aplikovat všechny transformace jednotně jako součit matic a vektorů. V případě lineárních transformací bude váha vždy $w = 1$ [1].

$$x = X/w \tag{2.1a}$$

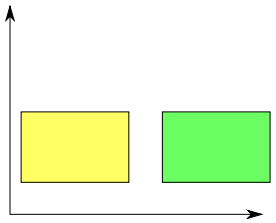
$$y = Y/w \tag{2.1b}$$

Geometrické transformace

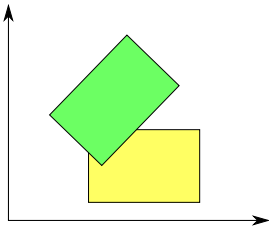
Transformace v prostoru, v tomto případě v rovině, jsou nejčastěji prováděné operace v oboru zpracování obrazu. Jelikož snad není řešení, ve kterém by se transformace nevyskytovaly, je vhodné je zde více přiblížit.

$$P = A \cdot P' \tag{2.2}$$

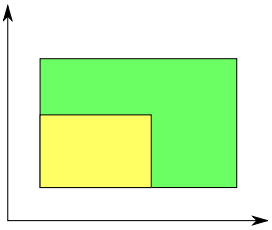
Jedná se o proces, při kterém se na každý bod v obraze aplikuje transformační matice, následkem čehož vznikne nový obraz bodu (Rovnice 2.2). Jelikož máme bod v rovině reprezentován trojicí, bude mít transformační matice vždy rozměr 3×3. V rámci práce budou popsány pouze některé lineární transformace, jmenovitě posunutí, otočení a změna měřítka.



Obrázek 2.1: Transformace posunutí



Obrázek 2.2: Transformace otočení



Obrázek 2.3: Transformace změny měřítka

Posunutí

Posunutí (angl. *Translation*) bodu s homogenními souřadnicemi $P = [X, Y, 1]$ v rovině podle vektoru posunutí $T = (dx, dy)$ je názorně popsán v rovnici 2.3. Opačné posunutí by bylo realizováno pomocí záporných hodnot dx a dy. Ilustrace posunutí je na obrázku 2.1.

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix} \tag{2.3}$$

Otočení

Otočení (angl. *Rotation*) bodu s homogenními souřadnicemi $P = [X, Y, 1]$ v rovině o úhel α se středem otočení v počátku souřadného systému je popsáno v rovnici 2.4. Kladná hodnota parametru α znamená otočení proti směru hodinových ručiček, otočení v opačném směru by pak zařídila hodnota záporná.

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Nové souřadnice x' a y' pak budou mít hodnoty podle rovnic 2.5a a 2.5b. Samotná ilustrace otočení je zobrazena na obrázku 2.2.

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha \tag{2.5a}$$
$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha \tag{2.5b}$$

Změna měřítka

Změna měřítka (angl. *Scale*) realizuje zvětšení či zmenšení obrázku. Pro bod reprezentovaný homogenními souřadnicemi $P = [X, Y, 1]$ je změna měřítka dána faktory změny měřítka S_x a S_y.

Faktor $S_{x,y}$, který je větší než 1, znamená zvětšení objektu. Pro faktor $0 < S_{x,y} < 1$ pak dochází ke zmenšení objektu. Pokud je faktor $S_{x,y}$ menší než 0, jedná se o zrcadlení - zvláštní případ, někdy uváděný jako samostatná transformační operace. Všechny tyto případy jsou ilustrovány v obrázku 2.3.

Tyto operace jsou popsány rovnicemi 2.7a a 2.7b. Maticový zápis je pak zobrazen v rovnici 2.6.

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$x' = x \cdot S_x \tag{2.7a}$$

$$y' = x \cdot S_y \tag{2.7b}$$

Skládání transformací

Velmi často je potřeba provést hned několik transformací zároveň, je vyžadována takzvaná obecná transformace. V takovém případě je nutné rozložit obecnou transformaci na jednotlivé lineární transformace, jejich transformační matice vynásobit a výsledná transformační matice je pak maticí výsledné obecné transformace. Je nutné ovšem dodržet pořadí násobení jednotlivých transformačních matic, neboť rozdílné pořadí má často naprosto nežádoucí výsledek..

Bodové operace

Geometrické transformace popsané výše pomůžou v případech, kdy je třeba nějakým způsobem upravit celý obraz bez ohledu na jeho obsah. V případě, kdy je třeba pozměnit obsah, hodnoty samotných pixelů, je třeba sáhnout po jiných metodách. Obsah této podkapitoly je zaměřen na práci s jedním bodem. Z toho důvodu zde nejsou popsány žádné operace, které by vlastnosti obrazu mohly získávat z více snímků.

První skupinou jsou různé lineární a nelineární filtry. Filtrace za pomoci těchto filtrů mohou odstranit šum, zvýraznit hrany v obraze či zdůraznit jiné aspekty obrazu. Jde v podstatě o úpravu vlastností bodu podle vlastností jeho okolí.

Tyto filtry jsou aplikovány pomocí diskrétní dvourozměrné konvoluce (Rovnice 2.8). Výsledek, kterého lze pomocí této aplikace dosáhnout, je možné ovlivnit volbou vhodného *konvolučního jádra*. Z toho důvodu jsou v rámci podkapitoly jednotlivé operace zapsány prostřednictvím matice, která představuje konvoluční jádro nebo také *konvoluční masku* [2].

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) \cdot f(x + s, y + t) \tag{2.8}$$

Druhou možností úpravy hodnot bodu jsou v podstatě extrémní případy filtrů, kdy se bere v potaz buď celý obraz nebo naopak jen jeden konkrétní bod. Naopak operace, která

počítá jen s hodnotou jednoho bodu a naprosto zanedbává jeho okolí, je třeba převod do stupňů šedi.

Redukce šumu

Obraz získaný z kamery často trpí šumem. Jde často o chybu snímání, jako je kvalita osvětlení, vlastnosti čočky i použitá technologie. Šum může v obraze vzniknout i při pozdějším zpracování, například jako následek komprese. Tento šum však může velmi ovlivnit výsledky dalšího zpracování obrazu, jakkoli jej lidé často ani nevidí.

Šum lze obecně rozdělit na dva druhy. První je takzvaný Gaussův šum. Tento šum lze odstranit lineárními filtry, často prostým průměrováním. Nejjednodušší způsob odstranění gaussovského šumu je použití gaussovského filtru. Tento filtr je daný konvolučním jádrem v rovnici 2.9.

$$H = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.9}$$

Druhý typem je pak šum „*sůl a pepř*“, který často vzniká při snímání. Někdy je nazýván také náhodným šumem. Obecně je vhodné na tento typ šumu použít nelineární filtry[3], které lépe zachovají ostrost obrazu. Jedním z nelineárních filtrů často používaných pro filtraci náhodného šumu je *mediánový filtr*.

Tento filtr spočívá v nahrazení bodu mediánem okolních hodnot. Medián lépe popisuje skutečnou tendenci okolních hodnot bodu a lokální extrémy, například extrémně světlé („sůl“) či tmavé („pepř“) body, jsou velice efektivně odfiltrovány.

Aplikace těchto filtrů je zobrazena na obrázku 2.4. Lze pozorovat rozmazání obrazu po aplikaci filtrů, které se liší dle použitého filtru.

Obrázek 2.4: Aplikace filtrů pro odstranění šumu. Vlevo je originální obrázek, nahoře po přidání a odstranění gausovského šumu, dole po přidání a odstranění šumu „*sůl a pepř*“.

Detekce hran

Filtry používáné k detekci hran realizují detekci hran například na základě jejich zvýraznění. Tyto filtry využívají předpokladu, že hrana je místo, kde je znatelný rozdíl hodnot mezi sousedními body. Proto je podstatou těchto filtrů tento rozdíl maximalizovat a detekci hran tak usnadnit.

Filtrů, které detekci hran realizují, je celá řada. Zde bude představen pouze Sobelův operátor, na němž bude demonstrována podstata filtrů pro detekci hran.

Sobelův operátor sestává z několika vzájemně otočených konvolučních jader, která odpovídají detekci hran v některém směru. Mezi nejčastěji používané patří kombinace jádra pro detekci horizontálních hran (Rovnice 2.10a) a jádra pro detekci vertikálních hran (Rovnice 2.10b). Výsledek použití Sobelova operátoru pro operaci detekce hran je zobrazen na obrázku 2.5.

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.10a}$$

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \tag{2.10b}$$

Obrázek 2.5: Použití filtru pro zvýraznění hran. Vlevo je obrázek po detekci horizontálních hran (rov. 2.10a), uprostřed po detekci vertikálních hran (rov. 2.10b) a vpravo po aplikaci obou filtrů. Barvy jsou invertovány.

Barevné modely

Barevný model, často také nazývaný *barevný prostor* či *barevný systém* (angl. *color model*, resp. *color space* nebo *color system*), je způsob, jak jednotně vyjádřit barvu. Jelikož se vnímání barev liší člověk od člověka a ani zobrazovací zařízení často nezobrazují barvy stejně, je nutné takto definovat a sjednotit způsob vyjádření barev [2].

Barevných modelů existuje hned několik. Hlavní rozdíl mezi těmito modely často spočívá v záměru, proč vznikly a k čemu se používají. V této sekci bude představen barevný model RGB, na němž budou demonstrovány podstatné vlastnosti barevných modelů.

Barevný model RGB

Barevný model RGB je jeden ze dvou základních barevných modelů. Je možné jej označit také jako nejčastější, neboť se hojně používá v počítačovém zobrazování a pro uložení fotografií. Název modelu symbolizuje způsob, jakým jsou skládány barvy, neboť se jedná o červenou (angl. *Red*), zelenou (angl. *Green*) a modrou (angl. *Blue*) barevnou složku.

Kombinací těchto tří složek tak vznikají různé barvy, jejichž množství je omezeno pouze pamětí určenou pro uložení každé barevné složky. Běžně se používá varianta RGB888, což značí 8 bitů na každou barevnou složku, a to je ve výsledku 16 777 216 barev ($(2^8)^3$). Je však možné používat i jiné bitové hloubky.

RGB model je zástupce takzvaných aditivních barevných modelů. Z podstaty jde o skládání barevných světel, z čehož plyne, že hodnota barvy (0,0,0), tzn. že $R = 0$, $G = 0$ a $B = 0$, značí černou, kdežto (255,255,255) značí v 24 bitovém RGB bílou (255 je maximální hodnota na 8 bitech, které jsou vyhrazeny pro barevnou složku).

Jak již bylo zmíněno, je možné použít i jiné bitové hloubky, což se také dělá. Jedinou zde zmíněnou bude varianta 8 bitového barevného RGB modelu, také zvaná RGB332. Hodnoty 3, 3 a 2 značí množství bitů vyhrazené pro jednotlivé barevné složky v pořadí RGB. Z toho je zřejmé, že na modrou barevnou složku není vyhrazen stejný prostor jako pro ostatní. Tato nerovnost je však tolerována, neboť vychází z faktu, že lidské oko je na modrou barvu nejméně citlivé a tedy není nutné používat tolik modrých odstínů jako červených či zelených (obr. 2.6).

Obrázek 2.6: Porovnání barevného obrazu v RGB888 (vlevo) a RGB332 (vpravo).

Ostatní barevné modely

Pro různé potřeby byly vytvořeny různé barevné modely. Těchto modelů je velké množství a liší se jak způsobem definice barvy, tak množstvím barev, které dokáže pokrýt. Barvy, které model dokáže reprezentovat, odpovídají právě potřebám, pro které byl model vytvořen.

Namátkou lze zmínit CMYK barevný model používaný pro tisk, či modely vycházející z modelu XYZ, který je zaměřený na intuitivní míchání barev s ohledem na skutečné vnímání lidského oka.

2.2 Příprava pro rozpoznávání

Rozpoznáváním je často nesprávně nazván celý systém, který rozpoznávání provádí. Samotnému rozpoznávání předchází několik kroků, které z podstaty nelze zahrnout názvem rozpoznávání, neboť se nic nerozpoznává, ale nelze je ani nijak přeskočit. Tyto kroky budou stručně popsány dále v následujících sekcích.

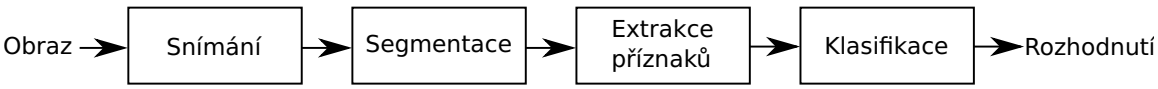
Schéma takového obecného rozpoznávacího systému lze vidět na obrázku 2.7. Z obrázku je patrné, že každé rozpoznávání začíná sběrem dat, takzvaným *snímáním*. V této fázi jsou nasnímána vstupní data, která budou rozpoznávána. Tomuto se práce věnuje v sekci Snímání.

Dalším krokem je pak *segmentace*. Tento krok je volitelný, záleží na aplikaci. Jeho úlohou je předzpracování vstupních dat tak, aby lépe odpovídala potřebám klasifikátoru. Některé způsoby segmentace jsou popané v sekci Segmentace.

Takto upravená data putují dál do části *extrakce příznaků*. V této části jsou ze vstupních dat definitivně vybrány pouze nejdůležitější informace a je vytvořen takzvaný *příznakový vektor*, který je vstupem pro další, předposlední část. Jaké nároky jsou na příznaky kladeny a jak jich může být dosaženo popisuje sekce Extrakce příznaků

Tou je již samotná *klasifikace*. Klasifikátor provadí na základě vstupních příznakových vektorů klasifikaci a pro každý produkuje výstup v podobě příslušnosti ke zvoleným třídám, často vyjádřené pravděpodobností. Klasifikaci a zejména některým konkrétním druhům klasifikátorů je věnována samostatná podkapitola podkapitola 2.3.

Poslední možným krokem je pak část zvaná *pozpracování* (angl. *postprocessing*). Úkolem této části je vhodně interpretovat výsledky klasifikátoru tak, aby co nejlépe a nejjasněji odpovídaly na otázku, která stála na počátku rozpoznávání.



Obrázek 2.7: Schéma procesu rozpoznávání od snímání po prezentaci výsledků

Snímání

Rozpoznávání obecně začíná snímáním (angl. *sensing*) dat. Jelikož rozpoznávání neznamená pouze rozpoznávání na základě obrazu, jedná se v tomto případě o mnoho různých snímacích senzorů. Těmito senzory může být v případě obrazu kamera, v případě zvuku pak mikrofon, ale také různé jiné senzory, jako je teploměr, barometr a další.

Pro rozpoznávání obrazu se používají snímací senzory jako kamera či fotoaparát. Velmi však závisí na kvalitě snímacího čipu, kvalitě čočky a dalších aspektech. Použití méně kvalitních komponent může vézt ke zkreslení obrazu. V případě čočky se může jednat o nejrůznější nerealistická zkreslení daná lokálními vadami čočky či celkové zkreslení dané zakřivením celé čočky.

Výsledkem procesu snímání obrazu je obecně pole hodnot jednotlivých barev, které je následně strukturováno dle některého formátu (viz sekce Barevné modely). Výstupem je tedy nejčastěji rastrový obraz, který nereprezentuje původní snímaný obraz, ale pouze jeho aktualní obraz v době sejmutí. Jelikož však takové pole obsahuje celý sejmutý snímek, včetně často nežádoucích částí jako je pozadí a podobně, není převážně vhodné jej použít přímo pro rozpoznávání. Dalším krokem je tedy segmentace.

Segmentace

Segmentace (angl. *segmentation*) je proces, při kterém jsou surová vstupní data upravena tak, aby byla vhodnější pro klasifikaci. Jedná se o rozdělení na menší části původního obrazu, tzv. *regiony* nebo *shluky* (angl. *regions* a *clusters*) [2]. Uspořádání doposud jednotlivých a významově nesouvisejících bodů v rastrové matici, která je výsledkem snímání (sekce Snímání), do těchto shluků, je podstatou segmentačních metod.

Segmentace je tedy mezikrok v procesu rozpoznávání, který oddělí pro rozpoznávání důležité regiony v obraze, jako je třeba objekt či významné body, od těch nedůležitých, jako třeba pozadí. Přínos je také v tom, že po takovém rozdělení algoritmus již pracuje s menšími částmi a je tedy často podstatně méně náročný.

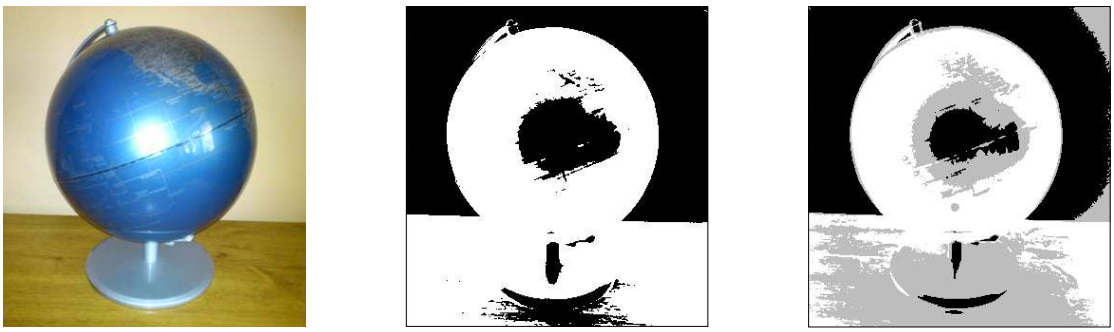
Jak již bylo uvedeno, segmentace vybírá pouze důležité regiony a ostatní ignoruje. Existuje mnoho technik, jak toho dosáhnout. Tyto techniky se liší efektivitou, složitostí implementace, ale zejména vhodností použití. Proto je nutné předem vědět, co se bude segmentovat a jaké výsledky se očekávají.

Prahování

Za nejzákladnější metodu segmentace lze považovat metodu tzv. *prahování* [2, 3]. Její podstatou je určení jednoho či více prahů (angl. *threshold*) a podle těchto prahů následně každý jednotlivý bod obrazu vyhodnotit. Matematický zápis je možno vidět na rovnici 2.11 pro případ jednoho prahu a na rovnici 2.12 pro prahy dva. Body, které mají podobné vlastnosti jsou tak sdruženy do shluků. Tyto vlastnosti bývají nejčastěji intenzita v bodě či hodnoty jednotlivých barevných složek. Podle způsobu, jakým je vybírán práh, pak existují různé metody prahování [4]. Podstata však zůstává stejná.

$$g(x,y) = \begin{cases} 1 & \text{pokud } f(x,y) > T \\ 0 & \text{pokud } f(x,y) \leq T \end{cases} \tag{2.11}$$

$$g(x,y) = \begin{cases} a & \text{pokud } f(x,y) > T_2 \\ b & \text{pokud } T_1 < f(x,y) \leq T_2 \\ c & \text{pokud } f(x,y) \leq T_1 \end{cases} \tag{2.12}$$



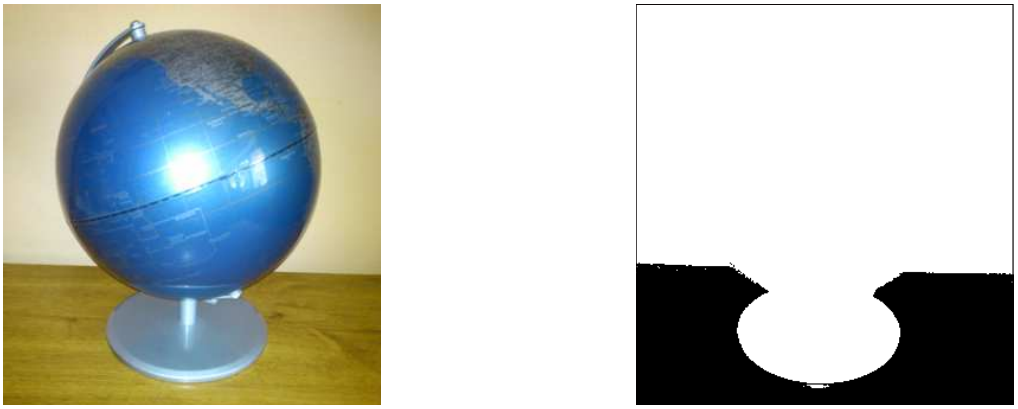
Obrázek 2.8: Příklad segmentace obrazu pomocí metody prahování. Vlevo je původní obrázek, uprostřed při použití jednoho prahu ($T = 150$) a v pravo při použití dvou ($T_1 = 120$ a $T_2 = 180$). Pro přehlednost jsou přidány černé okraje.

Metody shlukování

Jak název napovídá, tyto metody jsou založeny na vytváření shluků bodů podle určitých vlastností. Narozdíl od předchozí metody nemá pevně určený práh, ale novou hodnotu bodu získá podle vlastností sousedních bodů. Vzniknou tak různé shluky se stejnými nebo podobnými vlastnostmi a narozdíl od předchozí metody nevzniknou osamocené body, ale vždy větší shluky.

Příkladem takové metody je implementace algoritmu *k-means* [5]. Tato metoda při použití v segmentaci umožňuje segmentovat obraz pouze s informací o počtu shluků. Algoritmus je následující:

1. Pro každý shluk je určen střed. Toto je často prováděno náhodně, avšak je možné použít i některou heuristickou funkci a středy zvolit vhodněji.
2. Každý bod obrazu je přiřazen ke shluku, od jehož středu je nejméně vzdálen. Takto je zpracován celý obraz bod po bodu.
3. Po přiřazení všech bodů jsou přepočítány středy všech shluků, aby odpovídaly skutečné hodnotě bodů, které do nich spadají.
4. Pokračuje se bodem 2, dokud není každý bod přiřazený ke správnému shluku, tzn. po průchodu žádný bod nezmění svou příslušnost.

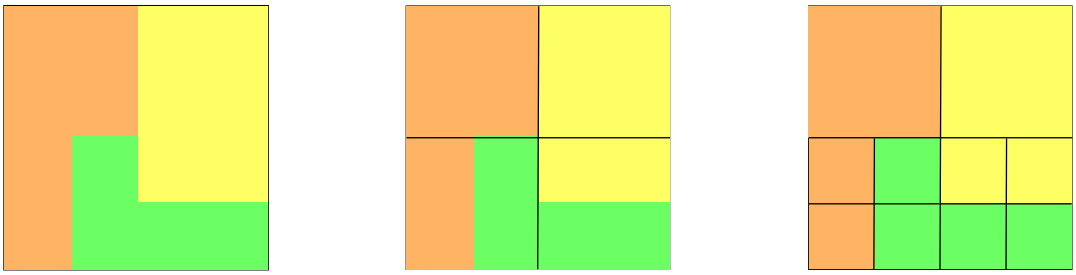


Obrázek 2.9: Příklad segmentace obrazu pomocí metody shlukování. Vlevo je původní obrázek, vpravo pak po segmentaci metodou shlukování. Pro přehlednost jsou přidány černé okraje.

Dělení a spojování oblastí

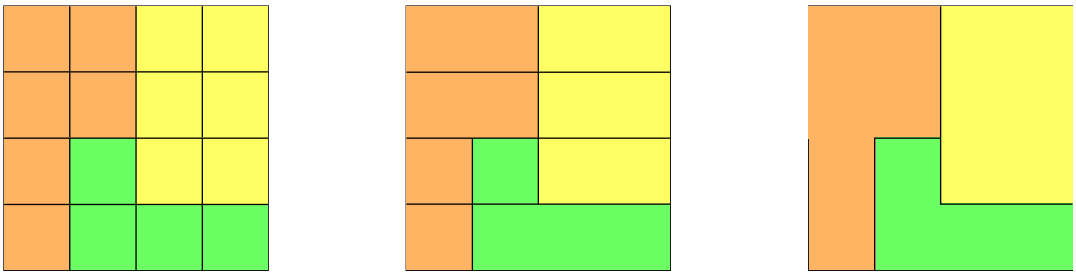
Metoda rozdělování a následného spojování oblastí (angl. *split and merge*) [6, 7] vznikla spojením starších metod dělení oblastí (angl. *region splitting*) [8] a spojování oblastí (angl. *region merging*) [9].

Dělení oblastí je metoda, jak název napovídá, založená na rozdělování oblastí na menší podoblasti. Oblast je rekurzivně dělena na čtvrtiny vždy, když je vyhodnocena jako nehomogenní. Algoritmus končí v momentě, kdy jsou již všechny podooblasti vyhodnoceny jako homogenní nebo jsou menší než určená minimální velikost. Homogennost oblasti je hodnocena na základě různých vlastností, podle kterých je pak možné rozlišit různé varianty metody. Ilustraci metody dělení oblastí je vidět na obrázku 2.10.



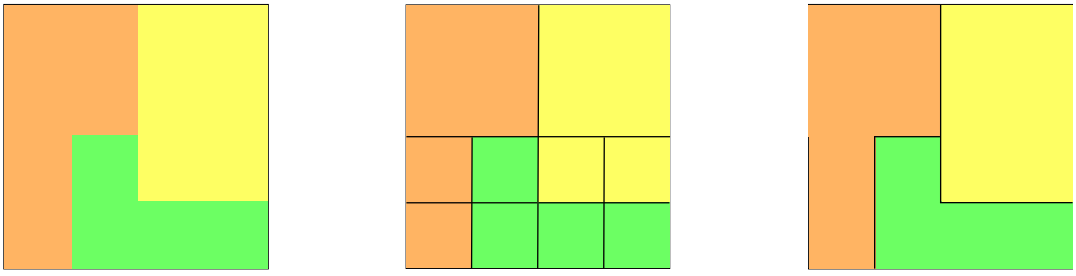
Obrázek 2.10: Segmentace metodou dělení oblastí. Vlevo je původní obrázek, uprostřed první krok metody, kdy je nehomogenní oblast rozdělena na čtvrtiny. Vpravo je vidět výsledek, kde všechny oblasti jsou již homogenní.

K jejich spojení slouží metoda spojování oblastí. Tato metoda ve své základní variantě určí každý bod jako samotnou podoblast. Následně opakovaně spojuje oblasti, které jsou si vlastnostmi nejbližší. Tato metoda končí v momentě, kdy si již nejsou žádné dvě oblasti dostatečně podobné a není tedy již co spojovat. Ilustraci metody spojování oblastí lze vidět na obrázku 2.11.



Obrázek 2.11: Segmentace metodou spojování oblastí. Vlevo je původní obrázek, který je již rozdělen na elementární části. Uprostřed první krok metody, kdy jsou spojeny podobné oblasti v řádcích. Vpravo je vidět výsledek, kde si již žádné sousední oblasti nejsou podobné.

Jak již bylo napsáno, tyto dvě metody dohromady tvoří metodu dělení a spojování oblastí. Když proběhne metoda dělení, obraz zůstane rozdělen na mnoho vzájemně sousedících oblastí, které však mají často úplně stejné či velmi podobné vlastnosti. Vlastnosti těchto oblastí jsou často tak podobné, že by byly funkcí na vyhodnocení podobnosti označeny jako homogenní. Z toho důvodu je vhodné tyto podobné sousedící oblasti opět spojit. Jak by tedy mohl vypadat průběh takové segmentace, lze vidět na obrázku 2.12. Hlavní výhodou této metody vůči prosté metodě spojování oblastí je rychlost, protože se nevychází z oblastí o velikosti jednoho bodu, ale z oblastí obecně podstatně větších.



Obrázek 2.12: Segmentace metodou dělení a spojování oblastí. Vlevo je původní obrázek, uprostřed je výsledek části dělení. Na ten navazuje metoda spojování (vpravo).

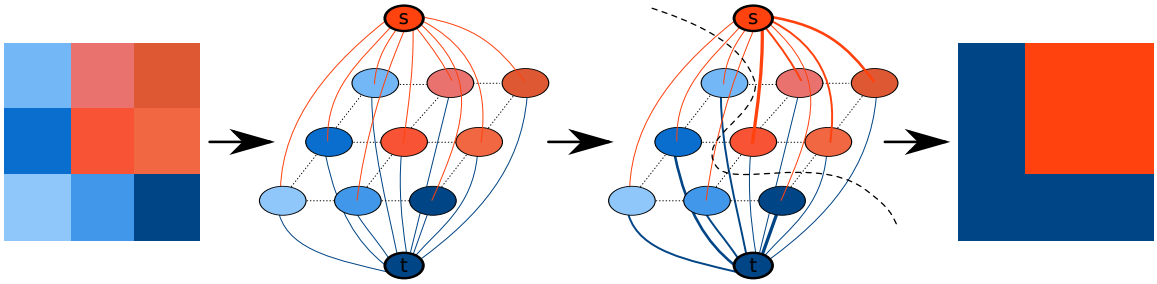
Grafové metody

Grafové metody vycházejí z problematiky teorie grafu [3]. Podstatou těchto metod je sestavení grafu, kde jednotlivé body v obraze jsou uzly grafu a hrany mezi uzly spojují sousedící body v obraze. Taková reprezentace obrazu je velmi výhodná, neboť umožňuje použití mnoha velmi efektivních činností spočívajících v průchodu grafem a grafových operacích obecně.

Jednou z metod, která využívá popis obrazu ve formě grafu a spadá tedy do kategorie grafových, je metoda *GraphCut*. Metoda hledá optimální globální řešení segmentace, takzvaný minimální řez, který je ekvivalentem maximálního toku v grafu [10]. Tento fakt vyplývá z *Fordovy-Felkersonovy věty* [11].

Řez grafem je taková operace, která rozdělí graf na dvě disjunktní množiny A a B tak, že jednoduše odstraní hrany, které tyto dvě množiny spojují (obr. 2.13). Hodnota takového řezu je pak dána součtem všech vah, které jsou tímto řezem zrušeny (Rovnice 2.13)

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{2.13}$$



Obrázek 2.13: Ilustrace řezu grafem.

Do grafu jsou tedy přidány další dva uzly, zdrojový uzel s (angl. *source node*) a cílový uzel t (angl. *sink node*), takzvané terminály. Každý uzel tedy obsahuje navíc vedle hran ke svým sousedům ještě jednu hranu vedoucí k cílovému uzlu t a jednu hranu vedoucí ke zdrojovému uzlu s.

Tyto terminály jsou tedy základy oblastí, které chceme od sebe řezem oddělit. Z předchozích odstavců je zřejmé, že energie takových částí musí být maximální, pokud řez bude minimální. Hledání minimálního řezu tedy algoritmicky přechází na hledaní maximálního toku.

Energie $E(A)$, kdy $A = (A_1, \ldots, A_p, \ldots, A_{|\mathcal{P}|})$, určuje pro každý bod $p \in \mathcal{P}$ příslušnost k jedné ze stran S, T, je dána dle vzorce 2.14. Cílem je najít takové A, jehož energie $E(A)$ je tedy nejmenší (minimální řez/maximální tok) [12].

$$E(A) = \sum_{p \in \mathcal{P}} R_p(A_p) + \sum_{\{p,q\} \in \mathcal{N}} B_{p,q}(A_p, A_q) \cdot \delta_{A_p \neq A_q} \tag{2.14}$$

kde $R_p(A_p)$ je funkce ceny oblasti, tedy každého bodu nezávisle na sousedech, a $B_{p,q}(A_p, A_q)$ je funkce ceny okolí, tedy dvou sousedních bodů. $\mathcal{P}$ je množina všech bodů obrazu a $\mathcal{N}$ je pak množina všech neseřazených sousedních bodů p, q. Pro funkci $\delta_{A_p \neq A_q}$ platí, že

$$\delta_{A_p \neq A_q} = \begin{cases} 1 & \text{pokud } A_p \neq A_q \\ 0 & \text{v jiným případech} \end{cases} \tag{2.15}$$

Funkce, které reprezentují výše zmíněné funkce oblasti a okolí, se liší podle oblasti použití metody. Metody GraphCut je často využívána právě pro její interaktivní variantu (obr. 2.14 uprostřed), která vyžaduje označení několika málo bodů na objektu a na pozadí. Na základě těchto označených bodů je pak stanovena cenová funkce, která je základem pro hodnocení všech zbývajících bodů obrazu.

Metody funkcí okolí jsou pak často odvozené, ne-li úplně stejné, pouze s tím rozdílem, že není sledována podobnost s žádným terminálem, ale se sousedními body.

Obrázek 2.14: Segmentace metodou GraphCut. Vlevo je původní snímek. Uprostřed jsou červeně označeny body objektu, modře body pozadí. Vpravo je zobrazen výsledek segmentace.

Způsobů, jak minimalizovat energii efektivním způsobem a nalézt tak v krátkém čase minimální a tedy optimální řez, je několik [12]. Není však cílem této práce je všechny podrobně rozebrat a popsat.

Výsledek segmentace metodou GraphCut je vidět na obrázku 2.14. Pro segmentaci byla použita implementace práce od autorů *O. Veksler*, *Y. Boykov* a *R. Zahib* [13] a funkcí oblasti je záporný logaritmus věrohodnosti v RGB332 (viz sekce Barevné modely) histogramu uživatelem určených bodů.

Extrakce příznaků

Příznak je v kontextu klasifikace a rozpoznávání chápán jako jakýsi zajímavý či významný prvek, případně přímo hodnota, která nějakým způsobem popisuje pro rozpoznávání podstatu klasifikovaného obrázku. V případě, kdy klasifikátor například klasifikuje na základě barev, stává se příznakem hodnota barevné složky či barevný histogram.

Výsledkem operací popsaných v sekci Segmentace je segmentovaný obraz, který rozlišuje části vstupu zajímavé pro klasifikaci a těch, které jsou pro klasifikaci zanedbatelné. Takto upravený obraz je tedy vhodným kandidátem pro klasifikaci.

Klasifikátor nicméně očekává čísla. Tato čísla můžou reprezentovat jakoukoli informaci, kterou do nich zakódujeme, nicméně pro klasifikátor to stále budou jen čísla. Často je velmi nevhodné a pro klasifikaci nepoužitelné použití celého obrázku jako vstup. Lze sice z podstaty počítačové teorie interpretovat obraz jako posloupnost čísel, nemá však smysl polemizovat o tom, že takový vstup nebude pro klasifikátor ideální.

Extrakce příznaků (angl. *features extraction*) tak vytáhne z obrazu, normalizuje a případně ještě zkomprimuje data tak, aby pro klasifikátor byla co možná nejvhodnější. Pokud nás tak zajímá barevná informace obrazu, není vhodné předkládat před klasifikátor informaci o barvě každého jednotlivého bodu obrazu, ale je lepší udělat jakýsi souhrn, který má vypovídající hodnotu stejnou, nebo dokonce vyšší.

Takovým souhrnem je třeba histogram. Histogram má pevný počet sloupců a po normalizaci známe i maximální hodnotu. Je tedy vhodným vstupem klasifikátoru. Vlastnosti, které dobrý příznak splňuje a které jsou pro klasifikátor nejvhodnější, jsou následující [14]:

- **Nekorelované.** Příznaky, které na sobě závisí a jejichž hodnoty stoupají a klesají obdobně, nejsou vhodné, protože nepřináší novou informaci, čímž pouze zbytečně zvyšují dimenzionalitu a tím i náročnost.
- **Mezitřídně variantní.** Příznak, který je napříč rozdílními třídami stejný nebo téměř stejný, pravděpodobně nebude nejlepší příznak. Je vhodné proto zvolit vždy takový příznak, který nejlépe rozlišuje mezi třídami, nebo jej alespoň vhodně otočit či deformovat, aby se jeho schopnost rozlišovat mezi třídami zlepšila.
- **Nízkodimenzionální.** Tento požadavek vychází z předpokladu, že většinou pouze zlomek dimenzí je skutečně rozhodujících. Každá další dimenze znamená vyšší výpočetní náročnost a navíc, pokud jde o dimenzi, ve které nejsou třídy separovatelné, přináší taková dimenze i větší zamíchání těchto tříd.
- **Transformačně invariantní.** Je vhodné počítat s různým natočením, posunutím či roztažením. Vůči všem těmto operacím by příznak měl být invariantní. Mimo tyto geometrické deformace by měl být invariantní i vůči změně osvětlení a mnoha dalším vnějším vlivům, které mohou nastat a případně negativně ovlivňovat výsledek rozpoznávání.

Příznaky, které jsou z obrazu extrahovány, však mohou být naprosto libovolné povahy. Jsou vybírány vždy podle největší vypovídající hodnoty a s ohledem na objekt klasifikace a rozpoznávání. Může jít o barevnou informaci, hrany, význačné body, různé grafické vzory, křivosti tvarů a další a další.

Analýza hlavních komponent

Analýza hlavních komponent (angl. *Principal Component Analysis*, zkrácené *PCA*), je metoda napomáhající snížení počtu dimenzí vektoru za minimální zráty informace. Tyto vlastnosti ji staví do pozice jedné z vůbec nejzákladnější úpravy příznakových vektorů.

Metoda zjednodušeně funguje tak, že vypočítá směry, ve kterých se data nejvíce liší (takzvané *vlastní vektory*, angl. *eigenvectors*), a hodnoty těchto směrů (takzvané *vlastní čísla*, angl. *eigenvalues*). Následně tyto dvě struktury seřadí podle hodnot vlastních čísel, neboť největší vlastní číslo a jemu odpovídající vlastní vektor udávají, ve kterém směru mají data největší varianci a tedy kde je nejvíce informace. Ve výsledku se tudíž odstraní ty dimenze, v jejichž směrech je nejméně informace. Což je také nevýhoda této metody, neboť hledá varianci mezi všemi daty a ne mezi třídami, takže se může stát, že odstraněné dimenze jsou pro rozpoznávání nezbytné.

Aby však bylo možné PCA aplikovat a dimenzionalitu snížit, je třeba projít několika kroky. Analýza se počítá na všech datech bez ohledu na skutečnost, do kterých tříd spadají.

Nejdříve je tedy třeba od každého vzorku datové sady odečíst aritmetický průměr. Aritmetický průměr je vypočítán pro každou dimenzi napříč celou datovou sadou (rov. 2.16) a od každé dimenze každého data je následně odečten.

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{2.16}$$

Výsledkem je datová sada, jejíž průměr je nulový. Taková datová sada je vhodná pro výpočet kovarianční matice. Kovarianční matice je symetrická matice, na jejíž diagonále jsou variance dat a na ostatních místech kovariance dat. Výpočet kovarianční matice C je z dat X, jejichž průměr je nulový, je zobrazen na rovnici 2.17.

$$C = X \cdot X^{\tau} \tag{2.17}$$

Z předchozího zápisu je zřejmé, že pokud násobíme matici X o velikost $M \times N$ maticí X^{τ}, která musí mít velikost $N \times M$, výsledná kovarianční matice C bude mít rozměry $M \times M$. Taková matice je tedy vhodná pro výpočet vlastních čísel a vlastních vektorů.

Vlastní čísla se z kovarianční matice získají pomocí vypočtení determinantu z charakteristické rovnice 2.18, položení jej rovno nule a vyřešení. Řešená rovnice bude takového řádu, jako počet sloupců či řádků (matice je čtvercová) kovarianční matice. Vyřešením této rovnice se tedy získají vlastní čísla λ, kterých bude právě tolik, kolik má kovarianční matice sloupců či řádků.

$$\begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{vmatrix} = 0 \tag{2.18}$$

Jakmile jsou vlastní čísla vypočítána, pomocí obdobné rovnice 2.19 jsou vypočítány vlastní vektory. Jednoduše je za λ_n dosazeno vlastní číslo a z rovnice se pak již snadno vyčíslí vlastní vektor u_n.

$$\begin{vmatrix} a_{11} - \lambda_n & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda_n & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda_n \end{vmatrix} \cdot u_n = 0 \tag{2.19}$$

Vlastních čísel a vlastních vektorů je právě tolik, kolik bylo dimenzí původních dat. Další krokem, který je zároveň finální, je odvození nových dat z nových dimenzích podle vybraných vlastních vektorů. Jak již bylo popsané, největší variance je ve směru toho vlastního vektoru, který odpovídá největšímu vlastnímu číslu. Vybere se tedy tolik vlastních vektorů odpovídajících největším vlastním číslům, do kolika dimenzí je v plánu nová data promítnout.

$$X_{new} = X \cdot X^{\tau} \tag{2.20}$$

Nová data jsou dána násobením transponované datové sady X zleva vektorem vybraných vlastních vektorů EV (rov. 2.20). Vektor vlastních vektorů EV má v řádcích jednotlivé vlastní vektory, má tedy tolik řádků, kolik vlastních vektorů. Na základě tohoto bude mít i výsledná matice právě tolik sloupců, kolik vektorů bylo vybráno, což má za následek snížení dimenzionality.

2.3 Klasifikace

Klasifikace (angl. *classification*), respetktive klasifikátor, je jádro celého systému pro rozpoznávání. Obecně lze klasifikátorem nazvat algoritmus, který na základě předem určených vlastností rozhodne, do které třídy náleží vzorek popsaný příznakovým vektorem (viz sekce Extrakce příznaků). Tato podkapitola popisuje některé rozdíly mezi různými druhy klasifikátorů a následně některé důležité zástupce popisuje zevrubněji.

Aby se klasifikátor mohl rozhodovat, musí vědět podle čeho a jak se rozhodovat. K tomu slouží takzvaná *trénovací data*. Název je to poněkud zavádějící, neboť ne každý klasifikátor vyžaduje trénování. Tato data je třeba chápat jako jakousi ukázku dat, ať už roztříděných do tříd nebo ne, případně i s informací o počtu tříd.

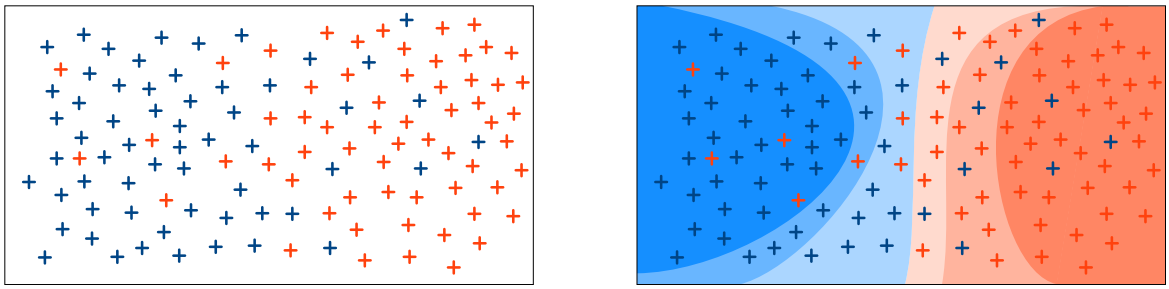
Jak již bylo zmíněno, klasifikátor může být trénovaný či netrénovaný. Rozdíl je v tom, že netrénovanému klasifikátoru předložíme dostatečný počet označkovaných vzorků z každé třídy a klasifikátor je okamžitě schopný na základě těchto dat rozlišit každý další vstupní vzorek. Takovým klasifikátorem je například klasifikátor zvaný *Nejbližší sousedé* (Nejbližší sousedé).

Opakem je klasifikátor trénovaný. Takový klasifikátor, když mu předložíme trénovací data, si zjednodušeně řečeno zapamatuje, čím se tato data liší či podobají, a vytvoří si odpovídající rozhodovací hranici. Na základě této rozhodovací linie je pak schopen rozlišovat každý další vstupní vzorek. Těchto klasifikátorů je drtivá většina.

Trénované klasifikátory se dále dělí podle způsobu trénování, a to na ty, které jsou trénovány takzvaně *s učitelem*, a na ty, které jsou trénovány bez něj. Přítomnost učitele v kontextu strojového učení, o které v tomto případě jde, znamená přítomnost informace o tom, co se má stroj naučit. V tomto konkrétním případě klasifikace přítomnost učitele znamená trénovací data, která jsou označkována (rozdělena do tříd). Klasifikace bez učitele se nazývá *shlukování* (angl *clustering*), v rámci této práce však již nebude dále rozebírana.

Nejbližší sousedé

Klasifikátor *Nejbližší sousedé* (angl. *k-nearest neighbors*) je jeden z vůbec nejjednodušších klasifikátorů vzhledem k tomu, že jej není třeba trénovat. Vstupem klasifikátoru je sada označkovaných dat, která slouží jako referenční data, a vzorek, který je třeba rozlišit. Na základě vzdálenosti klasifikovaného vzorku od referenčních dat je provedeno vyhodnocení [15]. Příklad klasifikace touto metodou lze vidět na obrázku 2.15.



Obrázek 2.15: Klasifikátor k nejbližších sousedů. Vlevo jsou zobrazena referenční data, vpravo pak vyhodocení celého prostoru, barevně je ilustrováno měkké rozhodnutí.

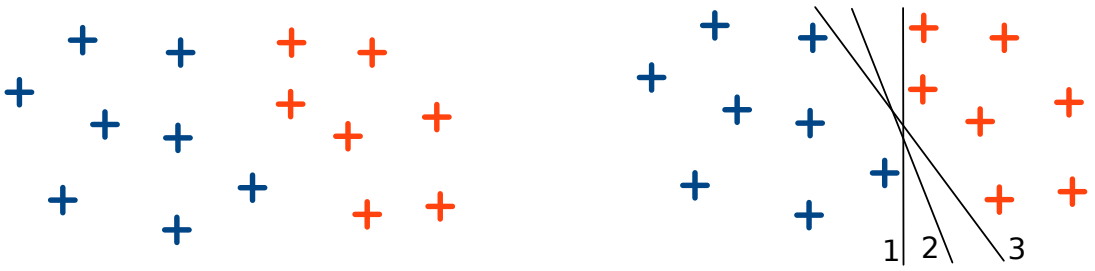
Z podstaty porovnávání vstupního vzorku se všemi referenčními daty plyne nutnost uchovávat všechna referenční data. To je také zřejmě největší nevýhoda této metody - paměťová náročnost.

Postup klasifikace pomocí této metody je tedy následující:

1. Trénovací data jsou vhodně seřazena. Je určena konstanta k, která určuje, kolik nejbližších sousedů se bude zjišťovat
2. Je změřena vzdálenost příchozího příznakového vektoru čekajícího na rozpoznání s každým vzorkem trénovacích dat.
3. Trénovací data jsou následně seřazena podle vzdálenosti od rozpoznávaného vzorku vzestupně.
4. Je vybráno k nejbližších sousedů a vyhodnocena třída, která se vyskytla nejčastěji. Případně je možné vyhodnotit procentuální příslušnost ke třídě vypočtením poměru tříd mezi k nejbližšími sousedy.

Lineární klasifikátor

Lineární klasifikátor je souhrnný název pro klasifikátory, které rozlišují mezi třídami na základě hranice určené lineární funkcí. Z tohoto důvodu bude správně fungovat pouze v případě lineárně separovatelných dat. V případě, kdy jsou data kupříkladu ve 2D ve formě mezikruží, lineární klasifikátor jednoduše není schopen najít vhodné řešení. Příklad rozhodovací hranice a rozhodnutí na jejím základě lze vidět na obrázku 2.16.



Obrázek 2.16: Lineární klasifikátor. Vlevo je příklad vstupních dat, na kterých je klasifikátor natrénován. Napravo je pak několik možných rozhodovacích linií vzniklých po natrénování takového klasifikátoru.

Rozhodovací hranice se získá trénováním. Trénování probíhá na základě trénovacích dat. Klasifikátor se opakovaně snaží vhodně umístit rozhodovací hranici mezi data, vyhodnotit zlepšení klasifikace a následně nalézt ještě lepší hranici [15].

Funkce, která definuje rozhodovací hranici, je obecně zapsaná jako v rovnici 2.21.

$$g(x) = w^\tau \cdot x + w_0 \tag{2.21}$$

kde x je vektor dat, $w = (w_1, \ldots, w_d)$ je váhový vektor v d dimenzích a w_0 je práh.

Práh w_0 určuje posunutí rozhodovací hranice od počátku, kdežto váhový vektor w její natočení. Do které třídy separovatelná data spadají, je provedeno na základě porovnání funkce $g(x)$ s nulou. Je-li větší, jedná se o první třídu, v opačném případě se jedná o třídu druhou.

Trénování perceptronu, což je jeden ze základních lineárních klasifikátorů, se provádí na základě hledání takových parametrů w a w_0, které spolehlivě rozdělí dvě třídy. Obecně je po každém kroku spočítána nějaká chybová funkce, která sdružuje všechny špatně klasifikované vzorky, podle jejich vzdálenosti od rozhodovací linie jim stanovuje váhu a po vyhodnocení

této funkce se v dalším kroku učení určí, jakým způsobem se změní klasifikační hranice, aby body klasifikovala lépe.

Z předchozího je zřejmé, že klasifikátor je trénován v iteracích. Počet těchto iterací je možné buď určit pevně či sledovat změny a zlepšení a učení ukončit například v době, kdy už učení nepřináší zlepšení. Může se totiž stát, že klasifikátor nenajde ideální řešení nikdy, nebo že se od určité chvíle začne přetrénovávat a od ideálního řešení se začne rychle vzdalovat.

Support Vector Machine

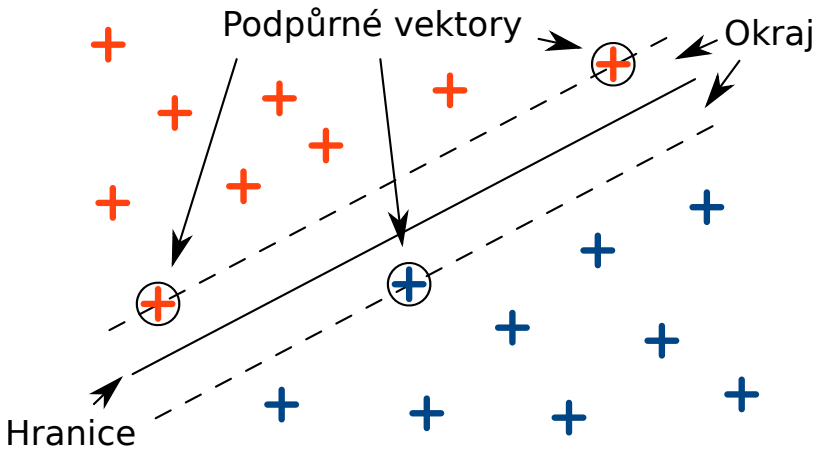
Podpůrný vektorový stroj (angl. *Support Vector Machine*, zkráceně *SVM*), je další z druhů klasifikátorů. Podstata metody spočívá v nalezené optimální vyšší dimenze, než ve která data jsou, která však umožňuje data efektivně rozdělit často i lineární funkcí. Tendence rozhodovací hranice však nemusí být vůbec lineární. Jde o metodu využívající takzvané *jádrové funkce*, které určují tvar rozhodovací hranice. Obecně lze jako rozhodovací hranici použít libovolnou funkci $g(x)$ od lineární funkce až po složité polynomy vysokého stupně.

Hranice, která je nalezena, je optimální. To znamená, že jsou splněny dvě podmínky. Zaprvé jsou všechna trénovacích data rozdělena do správných tříd s minimální chybou. Tato chyba se vypočítá ze vzorků, které se nacházejí na špatné straně rozhodovací hranice nebo v okraji, kde váha každého vzorku je dána vzdáleností od rozhodovací hranice (rov. 2.22) [15]. Druhou podmínkou je, aby okraj mezi rozdělenými daty byl maximální.

$$y_i(g(x)) \geq 1 - \xi_i \tag{2.22}$$

kde pro y_i platí, že pro data z $\omega_1 = 1$ a pro data z $\omega_2 = -1$, $g(x)$ je obecná rozhodovací funkce, $\xi_i = 0$ pro správně klasifikovaná data, $0 < \xi_i \leq 1$ pro data ležící na okraji a $\xi_i > 1$ pro data špatně klasifikovaná

Za okraj je v tomto případě považován prostor, na jehož každé straně se nachází jiná klasifikovaná třída a jehož středem prochází rozhodovací linie (obr. 2.17).



Obrázek 2.17: Okraj SVM Klasifikátoru. Lze vidět data rozdělená na dvě skupiny a ideální rozhodovací hranici, která je dána podpůrnými vektory.

Z rovnice 2.22 tedy vyplývá, že je žádoucí nalézt takovou hranici, pro kterou je počet bodů s $\xi > 0$ nejmenší. Algoritmicky tedy metoda přechází do minimalizace cenové funkce 2.23.

$$J(w, w_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} I(\xi_i) \tag{2.23}$$

kde ξ je vektor parametrů ξ_i a

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases} \tag{2.24}$$

Minimalizace rovnice 2.23 může být ovšem často velmi složitá operace. Existuje několik způsobů, jak tuto minimalizaci řešit, v této práci se jimi však dále nebudu zabývat.

Rozhodovací hranice je definována pomocí podpůrných vektorů, odkud má metoda název. Není třeba si tedy pamatovat všechny vzorky, ale pouze ty vektory, které skutečně určují rozhodovací hranici a její okraj. Metoda je tedy obecně velmi dobrá v *generalizaci* a je tak dostatečně robustní.

2.4 Platforma a architektura

Tato práce se zabývá implementací na platformě *Android* [16]. Implementace na této platformě má svá specifika, která je třeba brát v potaz, často již při návrhu aplikace, zejména pak ale v části řešící optimalizaci. Z tohoto důvodu je způsob vývoje aplikace na této platformě obsahem sekce Android.

Daleko důležitější je však pro vývoj výpočetně náročné aplikace, jako je rozpoznávání obrazu, nutnost znát arcihtekturu, na které taková aplikace poběží. Tato architektura má totiž právě tolik specifikací, jako platfroma. Navíc je to právě hardware, který výpočty provádí, a je proto jistě pochopitelné, že je třeba se věnovat jeho zvláštnostem a výhodám.

Architektura *ARM* [17] je architekturou, která dnes zaujímá přední místo v počtu mobilních zařízení, a která ve spojení s platformou Android vytváří základ pro většinu dnes vyrobených mobilních zařízení. Často je o nich uvažováno jako o jednom a tomtéž. Z toho důvodu bude sekce ARM věnována krátkému popisu této architektury tak, jak je pro tuto práci zajímavé.

Android

Platforma Android je mobilní platformou založenou na linuxovém jádru, za kterou dnes stojí uskupení *Open Handset Alliance* (zkráceně *OHA*) v čele se společností *Google* [16]. Systém je založenen na přístupu *otevřeného kódu* (angl. *Open Source*). Společně se systémem je distribuován i vývojový nástroj *Android SDK* pod stejnou licencí, čímž je podpořena velká obliba u vývojářů a velké množství vytvořených aplikací.

Vývojovým nástrojem však vstřícnost vůči vývojářům nekončí. OHA poskytuje k systému Android bohatou dokumentaci včetně podrobného popisu aplikačního rozhraní. Sám Android je navržen tak, aby maximálně umožnil a zjednodušil přístup ke všem součástím zařízení, které lze teoreticky využít.

Těmito součástmi jsou například mnohé senzory, kterými jsou taková zařízení doslova přeplněna. Výčet rozhodně nekončí u kamery či GSM a GPS modulu, ale obsahuje i další různé pohybové senzory, gyroskop, magnetický senzor, mikrofon, sensor na měření osvětlení

a další. Jistě, tyto senzory jsou doménou zařízení, na kterém systém Android běží, a ne systému samotného. Je ale jen zásluhou systému Android, že je přístup k nim vývojáři tak zjednodušen.

Vývoj na systému Android je prozatím uzpůsoben jazyku *JAVA*. Aplikační vrstva je řešena prostřednictvím virtuálního stroje *Dalvik*, který nahrazuje licencovaný *Java Virtual Machine*. Android však nabízí možnost pro potřeby většího výkonu nezatíženého během virtuálního stroje použít aplikace napsané v jazyce *C/C++* a přeložené přímo do nativního kódu. K tomuto účelu slouží nástroj *Android NDK*.

Problémem dnešní doby může být velká roztříštěnost, neboť se platforma Android v posledních letech velmi vyvíjela, a to včetně aplikačního rozhraní. V poslední době se zdá, že vývoj již není tak bouřlivý, nicméně spousta zařízení zůstává neaktualizována ve verzi *2.3*, přestože poslední stabilní verze je již *4.2* [16].

ARM

Architektura ARM není nová architektura (první mikroprocesor s architekturou ARM byl navržen v roce 1987 [17]), nicméně v poslední době je aktivně vyvíjen a stále častěji využíván. Jedná se o mikroprocesor s redukovanou instrukční sadou a velkou výhodou ve velmi nízké spotřebě energie a stále rostoucím výkonem. Právě snaha udržet spotřebovanou energii na konstantně nízkých hodnotách předurčuje architekturu ARM pro použití ve vestavěných a mobilních zařízeních.

Architektura ARM je od počátku navržena jako 32 bitová [18]. Výsledkem je jednodušší práce s pamětí, neboť všechny instrukce a bloky jsou zarovnány na šířku 32 bitů. To umožňuje mimo jiné například umístění podmíněných skoků či dokonce přímou přítomnost některých dat ve volání instrukce [17], jejichž šířka by jinak zůstala nevyužita.

Jelikož vývoj je v poslední době velmi aktivní, je vcelku běžné, že nové verze obsahují i některé nové specifické výpočetní jednotky. Namátkově lze zmínit například *VFP* (*Vector Floating Point*), která nahradila dřívější technologii *FPA* (*Floating Point Accelerator*). Zásadním rozdílem je, že technologie VFP přináší koprocesor manipulující s čísly v plovoucí řádové čárce, což je monohem efektivnější než FPA, která realizovala operace v plovoucí řádové čárce prostřednictvím jednotky *ALU* (*Arithmetic-Logic Unit*, pracující s celými čísly).

Další jsou pak rozšíření *NEON* a *DSP*. Tato rošíření pracují v řežimu *„Single Instruction, Multi Data“* (*SIMD*) a jsou tak přímo určená pro zpracování signálů, obrazu a obdobných. Bohužel se však jejich přítomnost liší zařízení od zařízení, nelze na ně tedy spoléhat stoprocentně. Výkon, kterého však jejich použitím lze dosáhnout, není zanedbatelný.

Kapitola 3

Analýza problému a návrh řešení

Rozpoznávání obrazu a tomu předcházející operace jsou problematiky, jimž se věnovalo mnoho času a jsou již na poměrně vysoké úrovni. Rozpoznávání hub z obrazu, konkrétně rozlišení, jestli je houba jedlá či nikoli, se věnoval v rámci semestrálního projektu *Damien matti* v práci s názvem *Mushroom Recognition* [19]. V rámci práce implementoval klient-server řešení, kdy na mobilním zařízení vyfotí houbu, provede segmentaci metodou GraphCut a data odešle na server, kde proběhne klasifikace. Matti doporučuje především dostatečně velkou databázi na trénování.

Obsahem této kapitoly tedy je popis návrhu řešení vycházejícího z technologií a informací popsaných v předchozí kapitole a s ohledem na předchozí práci v oboru. Nejdříve bude popsán cíl, ke kterému práce směřuje, včetně všech potřebných informací (podkapitola 3.1). Následuje stručné shrnutí použitých technologií navržených k řešení jednotlivých dílčích částí (podkapitola 3.2). Na závěr této kapitoly budou představeny popisy jednotlivých experimentů, na jejichž základě bude provedeno vyhodnocení celého návrhu (podkapitola 3.3).

3.1 Cíl

Cílem práce je navrhnout a implementovat aplikaci, která za pomoci znalostí z oboru počítačového vidění vhodně demonstruje problematiku rozpoznávání objektů v obraze na platformě Android. Tato aplikace by navíc měla být prakticky použitelná.

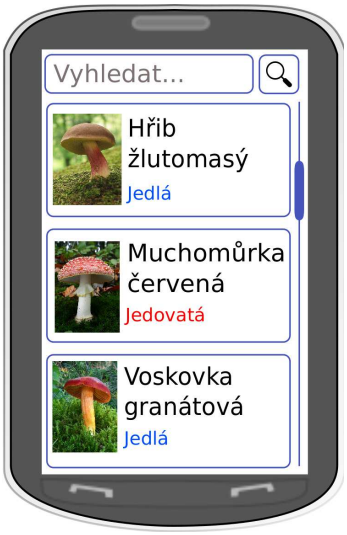
Aplikace si vytyčuje za cíl pomoci houbařům při rozpoznávání volně rostoucích hub prostřednictvím jejich mobilního telefonu. Vychází z velké rozšířenosti houbaření v zemích Střední Evropy, především pak v České a Slovenské republice a Polsku (stručný mykologický základ aplikace je popsán v sekci Mykologický základ). Platforma Android je z tohoto pohledu také ideální, neboť tato zařízení jsou dnes široce rozšířená a každému snadno dostupná. Navíc výkon těchto mobilních zařízení stále strmě stoupá.

Rozpoznání houby je realizováno na základě snímku houby. Tento snímek je získán za pomoci vestavěné kamery, která je dnes již takřka nedílnou součástí těchto zařízení. Tento snímek je následně zpracován přímo na mobilním zařízení. Odpadá tak nutnost odesílání dat ke klasifikaci na server za pomoci bezdrátového připojení, které je často v lese těžko dostupné, či nutnost houbu utrhnout a brát ji s sebou, aniž bychom si byli jisti, že ji skutečně chceme.

Po sejmutí houby kamerou je uživatel požádán, aby označil houbu a pozadí. Na základě tohoto označení je provedena segmentace, následně extrakce příznaků a samotná klasifikace. Výsledkem je pak seznam několika nejvíce podobných hub. Aby tento výsledný seznam byl

uživateli užitečný, bude se jednat o atlasové položky s informacemi o dané houbě, její vizuální charakteristice, informacemi o podobných houbách, snímkem houby a informací o jedlosti.

Z předchozího plyne, že aplikace bude obsahovat i atlas hub, neboť výsledek rozpoznávání je prezentován jako jeho podmnožina. Toto je také velmi užitečná vlastnost, neboť takový atlas může pokrýt daleko větší množství hub, než atlas papírový. Je tomu tak především kvůli faktu, že papírový atlas je často kvůli snaze o nízkou hmotnost pouze souhrnem několika nejčastějších hub. Bohužel, uživatel si často není jist právě těmi méně častými druhy, které však v takovém atlase již nenalezne.



Obrázek 3.1: Atlas hub.

Obrázek 3.2: Položka atlasu.

Jakkoli se aplikace jeví jako praktická a užitečná, je nutné zmínit také její nebezpečnost. Aplikace v žádném případě nemá za cíl suplovat houbařovo zdravé uvažování či za něj rozhodovat. Naopak, očekává se houbařův aktivní přístup, přinejmenším v podobě prostudování hub, které klasifikátor vyhodnotil jako nejpravděpodobnější. I tak však nelze spoléhat, že mezi navrženými houbami bude ta skutečná přítomna. V případě, kdy pak navrátí sadu hub, kde všechny jsou jedlé, a skutečná je jedovatá, může být aplikace nebezpečná i smrtelně.

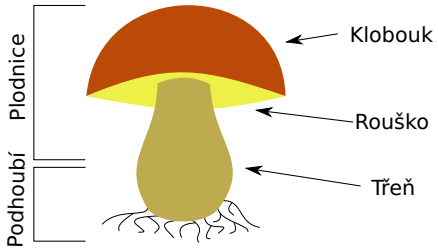
Mykologický základ

Rozpoznávaným objektem jsou tedy houby. Pro úspěšnou klasifikaci každého objektu je vhodné tento objekt znát a jinak tomu nebude ani v případě hub. Tato sekce se zabývá stručným popisem hub tak, jak je pro řešení dané problematiky nutné. Nesnaží se však o přílišné detaily, neboť se jedná pouze o informace doplňující, nikoli stěžejní.

Aplikace se zaměřuje na rozpoznávání hub. Jelikož si klade za cíl pomoci houbaři při houbaření, nesnaží se klasifikovat houby v celém rozsahu tohoto názvu, ale zaměřuje se pouze na podtřídu hub rouškatých (lat. *Agaricomycetidae*), konkrétněji na houby kloboukaté, kam patří všechny klasické houby. Ačkoli se sbírají i houby, které nespadají do podtřídy rouškatých, v této práci nebudou uvažovány.

Jedná se tedy o typické houby, jejichž tělo se skládá z podzemní částí, takzvaného *podhoubí*, a nadzemní části, takzvané *plodnice* (obr. 3.3). Tato plodnice se dále dělí na *třeň*

a *klobouk*. Klobouk má ze spodní strany *rouško*, na kterém můžou být buď *lupeny*, nebo *rourky*.



Obrázek 3.3: Stavba těla houby

Je zřejmé, že podzemní část nebude pro rozpoznávání relevantní. Klasifikace je založena na informaci získané z fotografie nadzemní části, konkrétně klobouku a třeně.

Mezi naše nejčastější houby patří bezesporu hřibovité. Vetšina hřibovitých hub je však jedlých, jen pár je jich nejedlých a naprosto výjimečně jde o houby jedovaté. Z tohoto pohledu je vhodné zaměřit se na rozpoznávání hub méně častých, většinou však také srovnatelně chutných a méně sbíraných. Jde o houby především *holubinkovité* (lat. *Russulaceae*), kam spadají holubinky, ale také třeba ryzce [20].

Dnešní mykologie provádí bezpečnou klasifikaci hub na základě mikroskopických znaků, jako je tvar či barva výtrusů, rozbor pachů a jiné. Vnější znaky, podle kterých rozpoznává běžný houbař, nemusí být vždy dostačující. Tato práce ale kopíruje postup rozpoznávání hub člověkem, a proto výchází pouze z obrazové informace. Pokud půjdeme ještě dál, je potřeba mít na vědomí, že dnešní mobilní zařízení sice mají kameru již zcela běžně, pachový senzor či mikroskop však nikoli.

Databáze

Pro kvalitní klasifikaci je třeba co možná největší trénovací databáze. Pro tento účel mi poskytl server *Nahuby.sk* [21] rozsáhlou databázi snímků hub pojmenovaných ve spolupráci s mykology. Tato databáze zahrnuje fotografie od mnoha autorů, takže pokrývá široké spektrum snímacích zařízení, světelných podmínek a dalších vlastností, které se dají očekávat při skutečném nasazení aplikace.

Velikost databáze přesahuje hodnotu 35 tisíc snímků. Množství snímků ke konkrétnímu druhu houby zhruba odpovídá rozšířenosti houby a její zajímavosti.

Databáze je tedy dostatečně velká, aby poskytla základ pro trénování, testování a validaci klasifikačního modelu. Na druhou stranu však obsahuje surová data a tak ji nelze použít pro trénování přímo, neboť segmentace vyžaduje aktivní přístup uživatele, a i na samotné snímání jsou kladeny požadavky, kterým snímky často nevyhovují.

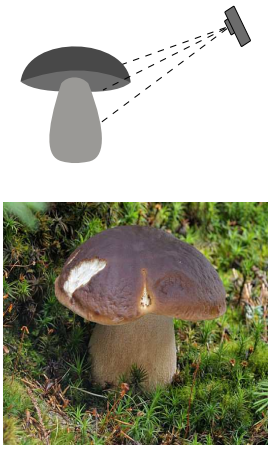
3.2 Technologický postup

Obsahem této podkapitoly je návrh způsobu realizace jednotlivých kroků a algoritmů, které budou použity. Sekce kopírují jednotlivé kroky tak, jak budou implementovány a jak je zobrazeno na obrázku 2.7.

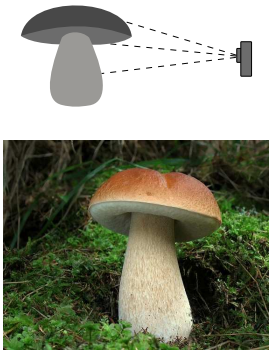
Snímání

Celý proces rozpoznávání začíná získáváním dat. Ta budou v tomto případě získána ve formě obrázku sejmutého vestavěnou kamerou mobilního zařízení. Pro snímání je však nutné stanovit několik omezení, aby sejmuté snímky byly pro klasifikaci co možná nejvhodnější. Tato omezení a způsob jejich řešení bude tedy následující:

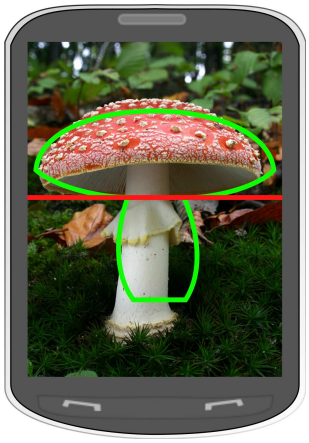
- **Osvětlení.** Jedním ze znaků, které budou použity ke klasifikaci, je barva. Z toho důvodu je nutné, aby bylo co možná nejvíce sjednoceno osvětlení, aby nevznikaly stíny či přehnaně světlé plochy. Tento problém z velké části může řešit přisvětlovací dioda fotoaparátu. Ta sice není přítomna na všech zařízeních, pokud však přítomna je, bude vynuceno její permanentní spuštění.
- **Sklon.** Dalším znakem, který bude při klasifikaci brán v potaz, je tvar. V případě tvaru je však nutno řešit problém, že samotná houba je trojrozměrná, kdežto obraz pouze dvourozměrný. Sklon, pod kterým bude houba snímána, je tedy nutné hlídat a omezit pouze na minimální rozsah, aby takto získané hodnoty tvaru nebyly příliš zkreslené. Ilustraci dvou příkladů sklonů snímání lze vidět na obrázcích 3.4 a 3.5. Způsob, jak vyřešit tento problém, je opět k nalezení v senzorech přítomných v cílových zařízeních, a to konkrétně v pohybovém senzoru (angl. *motion sensor*).
- **Pozice houby.** Pro navržené algoritmy segmentace je nutné, aby houba byla ve snímku uprostřed a aby byl klobouk v horní části snímku. Toho se využije také při automatickém zaostření na střed snímku, takže houba zůstane zaostřená, kdežto pozadí nikoli. Aby se vystředění houby dosáhlo, bude na promítací plochu nakreslena silueta a linka, jak je vidět na návrhu 3.6. Učelem siluety je vystředit celou houbu, linka pak odděluje klobouk od nohy a měla by pomoci tomu, aby byl klobouk v horní části snímku.



Obrázek 3.4: Ilustrace špatného úhlu při snímání.



Obrázek 3.5: Ilustrace správného úhlu při snímání.



Obrázek 3.6: Silueta a linka napomáhá správnému umístění houby v obraze.

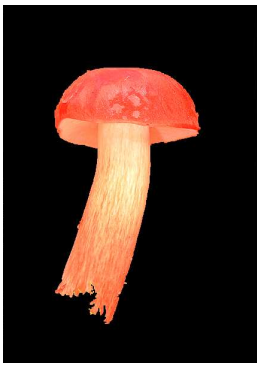
Segmentace

V momentě, kdy bude houba vhodně sejmutá, bude nutné ji segmentovat. Důvodem segmentace je oddělení nežádoucích částí od těch rozhodujících, konkrétně extrakci houby z pozadí a rozdělení na klobouk a třeň.

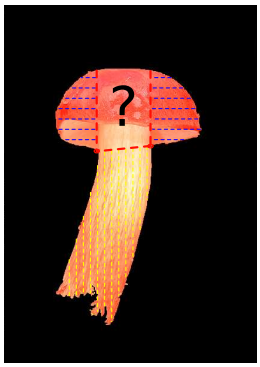
Pro segmentaci bude použita interaktivní metoda *GraphCut* (popsáno v sekci Grafové metody). Tento výběr byl podložen dobrými výsledky v práci *Mushroom recognition* [19] a také důvodu, že pro segmentaci hub lépe vyhovuje než ostatní metody popsané v sekci Segmentace. Interaktivita metody spočívá v tom, že je uživatel požádán, aby prostřednictvím dotykového displeje označil popředí a pozadí (obr. 3.7). Na základě tohoto vstupu bude provedena segmentace houby a pozadí (obr. 3.8).

Druhým krokem v segmentaci je pak rozdělení klobouku od třeně. To bude provedeno tak, že se naleznou dva body, kde se střetává klobouk a třeň, a mezi těmito body nahoru bude následně provedena segmentace stejnou metodou znova. Rozdílem je, že uživatel již nebude žádán o vstup, neboť je už známa dostatečná plocha klobouku a třeně, aby mohla segmentace proběhnout automaticky (obr. 3.9 a obr. 3.10).

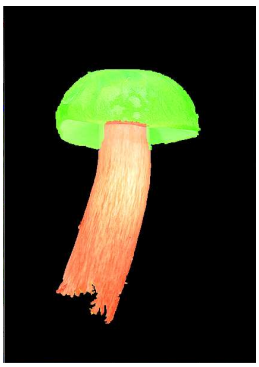
Obrázek 3.7: Vstup segmentace houby a pozadí



Obrázek 3.8: Výsledek segmentace houby a pozadí



Obrázek 3.9: Vstup segmentace klobouku a třeně

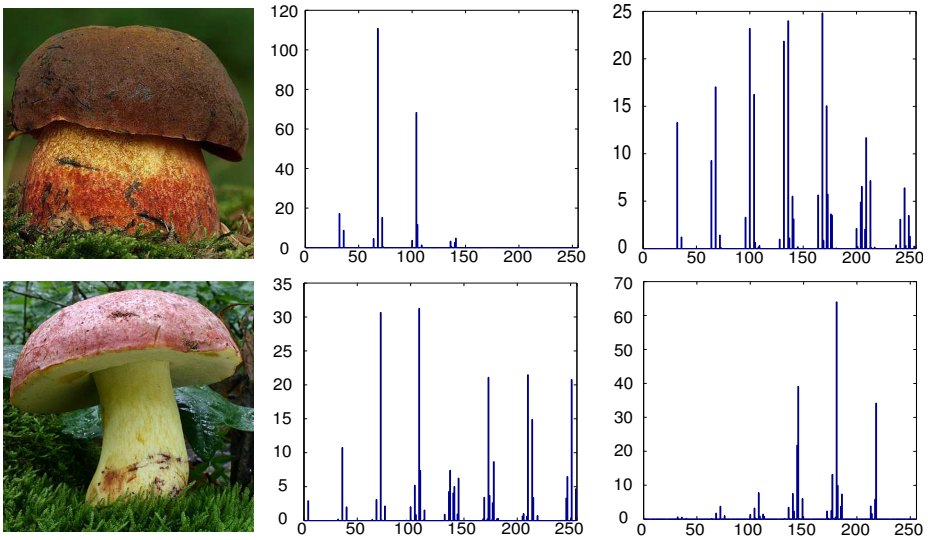


Obrázek 3.10: Výsledek segmentace klobouku a třeně

Extrakce příznaků

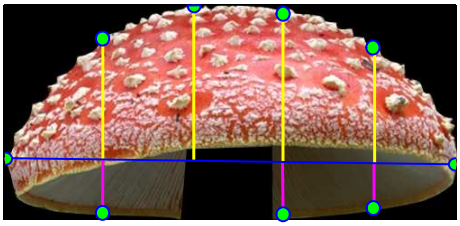
Znaků, na jejichž základě lze rozpoznávat houby, je mnoho. Tato aplikace však počítá jen s několika z nich. Konkrétně půjde o barvu klobouku, barvu třeně, tvar klobouku a tvar třeně. Tyto znaky byly zvoleny na základě jednoduchého získání a zájmu zjistit, jak úspěšná může klasifikace být pouze na jejich základě či na základě jejich kombinací.

Prvním extrahovaným příznakem bude tedy barevný histogram klobouku a třeně. Barevné histogramy budou počítány v 8 bitové hloubce modelu RGB, a to především kvůli menší velikosti za udržení podstatné části původní informace. Jedná se tak o první příznak, který bude použit (obr. 3.11).

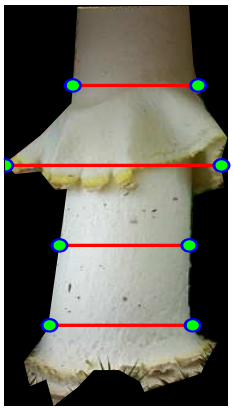


Obrázek 3.11: Horní řádek zleva: Hřib kovář, RGB332 histogram klobouku, RGB332 histogram třeně. Spodní řádek zleva: Hřib královský, RGB332 histogram klobouku, RGB332 histogram třeně.

Dalším příznakem bude tedy tvar. Pro získání tvaru klobouku bude vytvořena spojnice pravého a levého okraje klobouku. Tato vzdálenost bude rozdělena na několik dílů a na rozhraních mezi těmito díly bude změřena kolmá vzdálenost od zmíněné spojnice a horního, respektive spodního okraje klobouku. Tento způsob zjištění tvaru má navíc výhodu, že přináší částečnou invariantnost vůči otočení klobouku. Celou situaci lze vidět na obrázku 3.12.



Obrázek 3.12: Parametry tvaru klobouku



Obrázek 3.13: Parametry tvaru třeně

Tvar třeně bude reprezentován šířkou třeně v několika řezech. Výška třeně bude rozdělena na několik dílů a na hranicích mezi těmito díly bude změřena vzdálenost levého a pravého okraje. Metoda je opět částečně invariantní vůči otočení. Rozdělení lze opět vidět na obrázku 3.13.

Příznaky budou spojeny do jedno mnohadimenzionálního vektoru. Tento vektor však v případě, který byl právě popsán, reprezentuje houbu v $256 + 256 + 8 + 4$ dimenzích

(za předpokladu, že klobouk a třeň budou děleny na 5 dílů), což ovšem vůbec není málo. Jednou z možností, jak snížit počet dimenzí, je aplikovat na tento vysoce dimenzionální vektor metodu *PCA* (popsána v sekci Analýza hlavních komponent). Jelikož cílová aplikace poběží na mobilních zařízeních, lze předpokládat, že každé snížení dimenze bude mít za důsledek zvýšení rychlosti. V případě odstranění dimenzí, ve kterých mezi třídami není variance (zřejmě šum), lze očekávat i vyšší úspěšnost.

Jak již bylo řečeno, příznaků, které pomohou rozlišit houby, je mnoho. Namátkou lze zmínit například typ rouška (lupeny vs. rourky), textury povrchu klobouku a třeně, přítomnost prstence na třeni, přítomnost fleků na klobouku a mnoho a mnoho dalších. V rámci této práce a příslušné demonstrace budou však brány v potaz jen ty dosud popsané.

Klasifikace

Výsledkem extrakce příznaků bude tedy vektor příznaků. Tento vektor bude zároveň i vstupem klasifikátoru, který bude na množině těchto vektorů natrénován.

Za klasifikátor byl zvolem *SVM* klasifikátor (popsaný v sekci Support Vector Machine). Narozdíl od klasifikátoru *n-nejbližších sousedů* je trénovaný, takže nepotřebuje udržovat všechna data, a oproti *lineárnímu klasifikátoru* pak nabízí možnost různých rozhodovacích linií, ne pouze lineární.

Výhodou jsou i dobré výsledky a robustnost, která je pro SVM typická, především pak pro vícetřídní klasifikace, jako je tato.

Prezentace výsledků

Výsledkem klasifikátoru je pole věrohodností příslušnosti rozpoznávaného vzroku pro každou třídu (druh houby). Tento seznam bude seřazen a bude vybráno několik nejvíce pravděpodobných tříd, které se uživateli zobrazí. Zobrazený seznam se bude skládat z atlasových položek, čímž bude uživateli umožněno jednotlivé položky prostudovat a zjistit další rozhodovací znaky, na základě kterých se rozhodne.

Toto je velmi důležitý fakt, neboť uživatel nemá tušení, jak funguje klasifikátor a proč vybral právě ty houby, které vybral. Navíc se očekává uživatelův aktivní přístup a jeho schopnost se samostatně rozhodnout, o kterou houbu skutečně jde. Z toho důvodu je nutné, aby atlasové položky obsahovaly stručný popis znaků hub, podle který lze houby od sebe rozlišit. Náhled takového zobrazení výsledků je možné vidět na obrázcích 3.1 a 3.2.

3.3 Experiment a způsob vyhodnocení

Popis experimentu a způsob vyhodnocení je důležitou součástí práce, neboť stanoví podmínky, na základě kterých může být model vyhodnocen jako úspěšný. Obsahem této podkapitoly je tedy popis několika experimentů s různým zaměřením, na jejichž základě bude model vyhodnocen.

V následujících sekcích jsou popsány jednotlivé experimenty. Prvním experimentem je experiment na úspěšnost ropoznání, který je popsán v sekci Úspěšnost. Jelikož je uživateli jako výsledek prezentován seznam nejpravděpodobnějších hub, je v sekci 3.3 popsán experiment, který se soustředí právě na vyhodnocení tohoto seznamu. Dále pak experiment zaměřený na rychlost, neboť klasifikace běží na mobilních zařízeních a rychlost výpočtu je tedy také důležitá (Rychlost). A konečně, poslední experiment je zaměřený na uživatelské testování výsledné aplikace, popsaný v sekci 3.3.

Úspěšnost

Prvním experimentem, jehož výsledek je zřejmě i tím nejdůležitějším, je test ověření úspěšnosti modelu. Úspešnost samotná se bude měřit na úrovni jednotlivých příznaků a jejich kombinací. Údaj o úspěšnosti klasifikátoru ve formě poměru správně klasifikovaných výsledků ku celkovému počtu rozpoznávaných vzorků je jedním z nejvíce říkajích parametrů klasifikátoru vůbec. V případě této práce tomu není jinak.

Jak již bylo řečeno, změří se jako poměr správně klasifikovaných vzorků ku všem testovacím. Z toho důvodu jej lze interpretovat jako procentuální úspěšnost, neboť v případě, kdy klasifikátor správně rozpozná všechny vzorky, hodnota bude rovna jedné, kdežto v opačném případě bude výsledek nulový.

Samotný experiment bude mít následující průběh. Datová sada bude náhodně rozdělena v poměru 3:1 na úrovni každé třídy. Tři díly budou použity pro trénování, jeden pak pro vyhodnocení. Tento postup bude opakován pětkrát, celkový výsledek pak bude odpovídat aritmetickému průměru všech pěti výsledků.

Testování však proběhne pro každý příznak zvlášť a také pro jejich kombinaci. Výše popsaný postup bude tedy proveden samostatně pro barevný příznak, samostatně pro příznak tvaru a nakonec pro oba příznaky zřetězené dohromady. Tímto přístupem bude mít tento experiment 3 různé varianty.

Pozice

Další experiment, na jehož výsledcích je celkový model vyhodnocen, je velmi podobný prvnímu experimentu na úspěšnost. Také totiž hodnotí model na základě výsledku klasifikace. Bere však v potaz pouze ty případy, kdy klasifikátor nerozpozná správně třídy. V takovém případě zaznamenává pozici správné třídy v seznamu seřazeném dle pravděpodobnosti. Tyto pozice jsou následně průměrovány pro získání průměrné pozice.

Průběh experimentu je tedy obdobný průběhu prvního experimentu. Datová sada je rozdělena na čtvrtiny, kde tři jsou použity pro trénování a jedna na testování. Následně je model otestován a vyhodnocen. Tento postup je opakován pětkrát pro získání průměrné hodnoty neovlivněné výběrem vzorků, neboť rozdělení dat je provedeno náhodně.

Experiment je opět proveden na úrovni jednotlivých příznaků. Vzniknou tedy totožné varianty jako u experimentu předchozího, první zaměřená pouze na barevné příznaky, druhá zaměřená pouze na příznaky tvaru a třetí na jejich kombinaci.

Rychlost

Aplikace má ambice, aby běžela a prováděla klasifikaci na mobilním zařízení. Výkon těchto zařízení je omezený, přesto však musí uživatel získat výsledek v rozumném čase. V opačném případě by aplikace postrádala smysl. Měření rychlosti modelu, respektive délky výpočetních operací, je předmětem tohoto experimentu.

Testování rychlosti se soustředí na testování délky běhu klasifikace. Experiment je proveden na reálném mobilním zařízení několika druhů. Běh je měřen v aplikaci finální, tedy bez jakýchkoli testovacích a pomocných výstupů, které mohou ovlivňovat celkovou rychlost.

Samotné měření je vždy opakováno pětkrát, neboť první spuštění obahuje i alokaci zdrojů, což u dalších spuštění často není třeba (paměť zůstává nějakou dobu alokována a uvolněna je až v případě potřeby [16]). Výsledek je pak opět aritmetickým průměrem všech měření v odpovídajících konfiguracích.

Měření samotné je provedeno ve třech variantách pro každý příznak samostatně a pro jejich kombinaci.

Uživatelské testování

Předposledním experimentem, který ověří použitelnost samotné aplikace, je uživatelské testování. Tento experiment přímo závisí na vzorku uživatelů - testerů, na jejich vnímání, prioritách a zkušenostech. Z toho důvodu nejde o experiment exaktní, ale výsledky tímto experimentem dosažené jsou minimálně stejně důležité, jako výsledky experimentů doposud popsaných.

Samotný experiment je rozdělen do tří částí. První je zaměřena na uživatelské rozhraní a je provedena na vzorku dvaceti uživatelů, kteří byli vybráni ve snaze o rovnoměrné rozložení věku, technické zdatnosti a pohlaví. Experiment je vyhodnocen na základě reakcí a komentářů při provádění požadovaného úkolu přímo s aplikací. Všechny úkoly jsou provedeny po předchozím jednotném představení aplikace.

Cílů, které má uživatel tester v rámci testování za úkol dosáhnout, je několik. Tyto jednotlivé scénáře jsou jmenovitě *vyhledání houby v atlase*, *označení houby při segmentaci* a *správné vyfocení houby.*

Druhou součástí uživatelského testování je dotazník směřovaný na co nejširší vzorek uživatelů. Text, který dotazník uvadí, je následující: *„Představte si aplikaci, která je dostupná pro mobilní zařízení se systémem Android a která na základě fotografie houby pořízené pomocí vestavěné kamery Vám přímo v lese nalistuje v elektronickém atlase několik hub, které vyhodnotí jako nejpodobnější. Není tedy třeba listovat v omezeném papírovém atlase, není třeba ani žádného připojení k Internetu či snad nejvýkonějšího mobilního telefonu. Následující otázky jsou směřovány na právě takovou hypotetickou aplikaci, která je součástí mé bakalářské práce. Dotazník je anonymní.“*

Otázky dotazníku, včetně nabídnutých odpovědí, jsou k vidění v tabulce 3.1.

Znění otázky	**Možné odpovědi**
Jak často za rok chodíte na houby?	Vůbec / 0-2 / 3-6 / 7-10 / Častěji
Měl/a byste strach takovou aplikaci používat?	Ano / Ne
Používáte kapesní atlas při houbaření?	Ano / Ne
Kolik byste byl/a za takovou aplikaci ochoten zaplatit?	Číslo v KČ
Máte mobilní telefon s kamerou, dotykovým displejem a systémem Android?	Ano / Ne
Berete si svůj mobilní telefon do lesa na houby s sebou?	Ano / Ne
Kolik je vám let?	Do 15 / 16-18 / 19-25 / 26 - 35 / 36 - 50 / 51 a více
Jste žena nebo muž?	Žena / Muž

Tabulka 3.1: Otázky uživatelského dotazníku včetně možných odpovědí

Posledním součást uživatelského testování bude provedena dlouhodobě v reálném prostředí. Tento experiment počítá s tím, že uživateli, který aplikaci nainstaluje, použije a pokusí se o rozpoznání nějaké houby, nabídne po připojení zařízení k Internetu odeslání anonymních statistických dat.

Tato data budou sestávat z fotografie houby, parametrů získaných z obrazu, výsledku klasifikace a nastavení aplikace. Také budou obsahovat záznam o zařízení a době samotné klasifikace a segmentace.

Na základě těchto dat budou identifikována slabá místa a problémové operace.

Kapitola 4

Řešení a implementace

Obsahem kapitoly je popis řešení a samotné implementace. Pro aplikaci, běžící na Androidu a realizující rozpoznávání, byl vybrán pracovní název *Houbot* (zkráceně *HOUbařův roBOT*). Pro potřeby demonstrace v rámci této práce byla tato aplikace rozdělena na několik samostatných aplikací. Tyto části pak budou popsány v jednotlivých podkapitolách, včetně popisu problematiky, kterou z procesu rozpoznávání realizují.

První je aplikace suplující rozpoznávání na platformě Android (podkapitola 4.2). Druhou je aplikace, která pokrývá uživatelské rozhraní (podkapitola 4.3). Nakonec byla vytvořena aplikace, která běží na počítači a realizuje funkce nutné pro přípravu dat pro trénování (podkapitola 4.1).

4.1 Houbot Development

Aby odpovídalo pořadí jednotlivých kroků v procesu rozpoznávání (obr. 2.7), první popsanou implementovanou částí je podpůrný program běžící na systému linuxového typu, pracovně pojmenovaný *Houbot Development*. Tento program vznikl především kvůli potřebě konvertovat surová vstupní data (fotografie hub) na vektory příznaků vhodné pro trénování klasifikátoru, neboť jak bylo popsáno dříve (Databáze), databáze je ve formě skutečných fotografií a samotná segmentace vyžaduje uživatelovu spolupráci.

Pro tento účel byla využita kostra z předmětu *Základy počítačové grafiky - IZG* (vyučovaném na FIT VUT v Brně), používaná na cvičení, a to především pro již implementovanou funkci realizující načtení 24 bitového RGB obrazu a odchytávání událostí myší a klávesnice.

Snímání

Na snímek houby jsou kladena některá omezení. V první řadě je to vertikální úhel snímání. Jak je vidět na obrázcích 3.4 a 3.5, tento sklon vlivem promítnutí trojrozměrné houby do dvourozměrného prostoru může velmi ovlivnit rozhodovací schopnost příznaku tvaru.

Jednou z možností, jak toto zkreslení eliminovat, je pomocí zařízení omezit povolený sklon pro snímání. Pro tento účel je implementováno omezení založené na pohybovém senzoru, který v pravidelných nastavitelných intervalech generuje události s informací o orientaci zařízení. Výsledek je prezentován pomocí malé ikony signalizující naklonění vždy, když je zaznamenána nová událost. Uživatel je tedy průběžně informován a není mu dovoleno vyfotit houbu, pokud má špatný sklon zařízení.

Druhou věcí je svislá poloha zařízení vůči houbě. Toto bohužel nelze implementovat do aplikace, proto je nakreslena ve výšce tří pětin snímaného obrázku vodorovná čára, která

má uživateli pomoci, aby spodní hrana klobouku ležela vždy těsně nad touto čárou. Tato dvě omezení dohromady ve spolupráci s nakreslenou siluetou houby pro její vycentrování (obr. 3.6) vymezí a sjednotí snímání v dostatečné míře.

Dalším způsobem, jak je využito zařízení při snímání, je osvětlení a zaostření. Osvětlení je velmi důležité, obzvláště pak v tmavém lese, proto je vynuceno spuštění přisvětlovací diody. Zároveň provází sejmutí snímku automatické zaostření na střed, aby objekt - houba, zůstala ostrá. Obě tyto operace jsou v Androidu implementovány jako parametry kamery zařízení.

Další věcí, která je implementována, je uložení sejmutého snímku do stálé paměti zařízení. Toto má hned tři důvody. Jednak jde o otázku bezpečnosti. Pokud se totiž uživatel otráví houbou identifikovanou toutu aplikací, je dobré mít uložen její snímek. Jako u každé jiné otravy, tak i v případě hub je dobré vědět, čím se dotyčný otrávil. Druhým důvodem je modul zpětné vazby, který nabídne uživateli odeslat anonymní data na server, pro potřeby testování. Třetím důvodem je uložení obrázků pro případ uživatelova osobního alba.

Obecným problémem snímání je roztříštěnost zařízení. Android API sice nabízí možnosti, jak zjistit parametry dostupných senzorů či vůbec jejich přítomnost, pokud však zařízení některý sensor nenabízí, těžko může pomoci. V takových případech se snímání musí omezit na dobře informovaného uživatele a ponechat splnění omezení na něm.

Segmentace

Segmentace byla implementována pouze v aplikaci běžící na linuxovém systému, neboť jí bylo třeba především při trénování. Implementační detaily včetně problémů, se kterými se bylo třeba vyrovnat, jsou popsány níže.

Houba vs. pozadí

Segmentace je prováděna na základě grafové metody GraphCut. Bylo tedy nutné sestavit z obrazu graf, požádat uživatele aby označil několik bodů na houbě a pozadí a na základě funkce zjištěné z těchto bodů potom vypočítat cenu každého dalšího bodu.

Většinu popsaného dobře řeší implementace *GCoptimization* od autorů Olga Veksler a Andrew Delong [22, 23, 24, 13]. Tato implementace poskytuje potřebné struktury a funkce pro sestavení grafu, včetně minimalizace ceny a nalezení optimálního řezu. Na programátorovi zůstává už jen určit cenové funkce.

Cenová funkce každého jednotlivého bodu byla stanovena jako negativní logaritmická věrohodnost (angl. *negative log-likelihood*) příslušnosti bodu do barevného histogramu popředí nebo pozadí (rov. 4.1a a rov. 4.1b).

$$R_p(\text{obj}) = -\log P(I_p|\text{obj}) \tag{4.1a}$$

$$R_p(\text{bkg}) = -\log P(I_p|\text{bkg}) \tag{4.1b}$$

Barevný histogram je počítán z bodů označených uživatel a to v barevném modelu RGB332. Převod z RGB888 je proveden tak, že jsou zanedbány nejméně významné bity, zbylé jsou pak zřetězeny za sebe v pořadí odpovídající modelu RGB, čímž vznikne 8 bitů, což je 256 barev.

Následně je histogram pozadí a histogram popředí normalizován tak, aby pro stejné hodnoty vyjadřovaly pravděpodobnosti jedné či druhé třídy. Jinými slovy, součet pravděpodobností pro stejné hodnoty z obou histogramů je roven 1. Celá úprava je popsána pseudokódem v 4.1.

Algoritmus 4.1: Normalizace histogramů

```
// fgHist and bgHist are 256 items long arrays representing
// the foreground and the background histogram
// fgPoints and bgPoints are counts of points from which
// the histograms are made

for (i := 0; i < 256; i++)
{
    if (fgHist[i] == 0 && bgHist[i] > 0)
    {
        bgHist[i] := 1
    }
    else if (fgHist[i] > 0 && bgHist[i] == 0)
    {
        fgHist[i] := 1
    }
    else if (fgHist[i] == 0 && bgHist[i] == 0)
    {
        fgHist[i] := 0.5
        bgHist[i] := 0.5
    }
    else
    {
        fgHist[i] := fgHist[i] / fgPoints
        bgHist[i] := bgHist[i] / bgPoints

        diff := 1 / fgHist[i] + bgHist[i]
        fgHist[i] := fgHist[i] * diff
        bgHist[i] := bgHist[i] * diff
    }
}
```

Následně jsou provedeny dva kroky rozvoje hledání minimálního řezu, jehož výsledkem je hodnota výsledného označkování. Toto omezení bylo zvoleno kvůli faktu, že již po dvou krocích je minimalizace dostatečná a další kroky pak pouze znamenají další spotřebovaný výpočetní výkon, ne však již viditelné zlepšení výsledku.

Na základě výsledku rozvoje je vytvořena maska obrazu, která obsahuje shluky označené jako popředí a shluky označené jako pozadí. Tato maska obsahuje velké množství fragmentů, požadován je však právě jeden shluk pro pozadí a právě jeden shluk pro objekt. Tyto fragmenty je tedy nutné odfiltrovat a ponechat pouze jeden shluk pro každou hodnotu masky.

Odstranění fragmentů

Odstranění izolovaných shluků je implementováno na základě hledání cesty. V rámci snímání byla stanovena podmínka, že bod uprostřed musí patřit mezi body na popředí. Odstranění fragmentů tedy vychází z předpokladu, že z bodu, který je označený jako popředí, musí existovat cesta do středu masky pouze přes body, které jsou také označeny jako popředí. Pokud tato cesta neexistuje, jde o osamocený shluk, a bod je přeznačkován na pozadí. Obdobný postup je pak aplikován pro body pozadí.

Hledání cesty je relizováno prostřednictvím metody *Uniform cost search*, neboť přechod z jedoho bodu do vedlejšího má vždy cenu 1. Celková cena do cíle je pak dána jako vzdálenost podle *Manhattanské metriky*. Tato metrika byla vybrána z důvodu dosažení stejných

výsledků jako s *euklidovskou metrikou*, ovšem bez nutnosti výpočtu odmocniny a s použitím pouze celočíselné aritmetiky. Celý postup je popsán v algoritmu 4.2

Algoritmus 4.2: Hledání cesty

```
N := all points of the image
O~:= points marked as the object
B := points marked as the background
V~:= visited points
T := target of the path    // [0,0] for background, middle point for object
eventStack := sorted stack by the length of a path to the target
blobStack := stack of actual proceed points
pathExists := false        // bool if the path to the target exists

foreach (x in N)

    if (x in O)
        pathExists := false             // Initialize the path existence sign
        eventStack.push(x)

        while (! eventStack.empty())
            tmpPoint := eventStack.begin()   // The stack is sorted
            V.push(tmpPoint)           // Mark the point as visited
            blobStack.push(tmpPoint)  // Mark the point as actually processed

            foreach (n in neighborhood) // Real 4-neighborhood
                if (n == T)
                    pathExists := true // Path exist
                    eventStack.clear() // Clear the eventStack to exit a loop
                    break;
                else
                    if (n in O~AND n not in V)
                        // The neighbor has not been visited yet and it is
                        // also marked as the object
                        eventStack.push(n)

        if (! pathExists)
            // The path does not exists, mark all the points in blobStack
            // as the background and also mark them visited
        else
            // The path exists, mark all the points in blobStack as the
            // foreground and mark them visited as well

// Do the analogic operation for all the points marked as the background
```

Tento průchod algoritmu umožní odstranění jednotlivých fragmentů. Zůstane pouze jeden, označený jako objekt, pro nějž existuje cesta do středu, a jeden, označený jako pozadí, s existující cestou do bodu [0,0].

Klobouk vs. třeň

Dalším problémem, který bylo třeba vyřešit, bylo rozdělení klobouku a třeně, jelikož příznaky jsou počítány pro klobouk a třeň samostatně. Metoda GraphCut, která pro tento účel byla použita, však již nepočítá s uživatelovou spoluprací.

Levý a pravý bod klobouku jsou nalezeny jako nejlevější a nejpravější body v horní

polovině masky, protože při snímání je vyžadováno, aby klobouk houby byl v horní polovině obrazu. Mezi těmito body je vypočítána spojnice (alg. 4.3).

Algoritmus 4.3: Spojnice levého a pravého okraje klobouku

```
leftx := X coordinate of the left most point
lefty := Y coordinate of the left most point
rightx := X coordinate of the right most point
righty := Y coordinate of the right most point
dx := rightx - leftx;
dy := righty - lefty;

for (x := leftx; x < rightx; x++)
{
    y := lefty + dy * (x - leftx) / (dx);
    // The point [x,y] belongs to the connection
    // between [leftx,lefty] and [rightx,righty]
}
```

Body, kde se střetává klobouk a třeň, jsou nalezeny jako body nejbližší bodu `[ABS(rightx - leftx), ABS(righty - lefty)]` na levém a pravém okraji. Vzdálenost je opět počítána v Manhattanské metrice. Od těchto bodů jsou vedeny vzhůru přímky, od nichž směrem k okraji obrazu jsou body označené jako objekt přeoznačeny jako klobouk. Od spojnice těchto bodů jsou body směrem dolů přeoznačeny jako třeň. Tyto body jsou tedy základem pro výpočet histogramů třeně a klobouku.

Jelikož použitá implementace automaticky vytváří graf ze čtyřokolí, musí být segmentovaná plocha obdélníková. Proto je skutečná počítaná plocha segmentace určená na ose x nahoře nižším i dole nižším bodem. Zbývající plocha nad tímto prostorem je přeoznačena jako klobouk, protože nad kloboukem již třeň nebude. Plocha, která případně vznikne v tomto objektu pod spojnicí těchto dvou bodů je pak označena jako třeň a to až po segmentaci a bez ohledu na výsledek segmentace v těchto místech.

Extrakce příznaků

Příznaky extrahované a implementované jsou dva, barva a tvar. Oba jsou počítány samostatně pro klobouk a třeň a výsledek je pak zřetězení těchto příznaků do mnohadimenzionálního vektoru.

Prvním vektorem příznaků je tedy barva. Jak již bylo popsáno, barva je počítána v 8 bitovém modelu RGB. Histogram barvy tak tvoří 256 dimenzionální vektor pro klobouk a 256 dimenzionální vektor pro třeň. Dohromady tedy příznakový vektor barvy existuje v 512 dimenzionálním prostoru.

Druhým vektorem příznaků je tvar. Tvar popisuje okraje částí houby v dvourozměrném prostoru tak, jak je sejmut fotoaparátem. Proto je nutné omezit úhly a pozici snímání, aby se minimalizovalo zkreslení. Klobouk je rozdělen na pětiny ve směru osy x, opět je vytvořena spojnice s spojující nejvíc pravý a nejvíc levý bod klobouku (alg. 4.3) a v průsečících těchto přímek jsou změřeny vzdálenosti horního a spodního okraje houby po spojnici s. Vektor klobouku bude tedy existovat v 8 dimenzích a seřazen bude v pořadí nejdříve horních vzdáleností, následně spodních vzdáleností, vždy zleva doprava, jak je popsáno ve výrazu 4.2.

$$(levý\ nahoře,\ \ldots,\ pravý\ nahoře,\ levý\ dole,\ \ldots,\ pravý\ dole) \tag{4.2}$$

Obdobně je sestrojen příznakový vektor tvaru třeně. Je zjištěna délka třeně ve měru osy y, která je opět rozdělena na 5 dílů. Hodnoty vektoru v jednotlivých dimenzích odpovídají vzdálenostem levého a pravého okraje třeně v jednotlivých řezech. Vektor je seřazen v pořadí shora dolů, jak ilustruje výraz 4.3.

$$(nahoře,\ \ldots,\ dole) \tag{4.3}$$

Výsledný příznakový vektor tvaru je dán zřetězením vektoru tvaru klobouku a vektoru tvaru třeně v tomto pořadí, existuje tedy v 12 dimenzích. Výsledný příznakový vektor, který obsahuje barvu i tvar, je vytvořen opět zřetězením vektoru barvy a vektoru tvaru, dohromady tedy tento celkový vektor existuje v 522 dimenzích. V rámci testování byla testována funkčnost i na úrovni jednotlivých příznaků, proto jsou zde popsány jednotlivě.

4.2 Houbot Benchmark

Druhou samostatnou aplikací, tentokrát již běžící na mobilním zařízení se systémem Android, je aplikace pracovně nazvaná *Houbot Benchmark*. Tento název byl zvolen s ohledem na fakt, že prvotním cílem aplikace je provádět experimenty na základě předem nastavených konfiguracích. Jako takový však také obsahuje implementované klasifikační jádro a pokrývá tak samotnou problematiku rozpoznávání na platformě Android.

Jedním z experimentů, které tato aplikace vyhodnocuje, je i experiment zaměřený na rychlost segmentace. Tento experiment je však stejně jako ostatní přednastavený, což znamená, že označené oblasti jsou předem dané a tudíž není třeba žádné další uživatelské aktivity.

Klasifikace

Klasifikace je rozdělena na dvě části. První částí je trénovací modul, který však nebyl implementován samostatně, protože je již součástí použité knihovny LIBSVM [25]. Pomocí nástrojů *svm-train*, *svm-scale* a *svm-predict* je natrénován model na stolním počítači. Pro klasifikaci je tedy použit klasifikátor SVM s jádrovou funkcí RBF (*Radial Basis Function*). Parametry tohoto klasifikátoru byly zjistěny pomocí nástroje *grid.py*, který je rovněž součástí knihovny LIBSVM. Klasifikátor byl navíc natrénován s přepínačem `-b 1`, který zaručuje, že je model natrénován tak, aby byl při klasifikaci schopný generovat pravděpodobnostní skóre pro jednotlivé třídy. To je velmi důležité, neboť na základě tohoto skóre jsou výsledky následně řazeny.

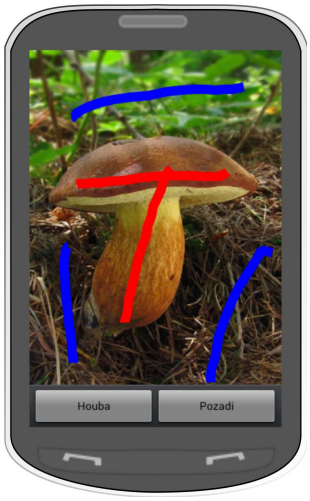
Na stolním počítači byl tedy modul natrénován, čímž vznikl trénovací dataset, validační dataset a natrénovaný model. Poslední dvě části, uložené v samostatných souborech, byly nepozměněné přeneseny na paměťovou kartu zařízení, odkud jsou při klasifikaci načítány. Uložení na paměťovou kartu bylo zvoleno z důvodu, že se očekává velká paměťová náročnost natrénovaného modelu a atlasu samotného, tudíž je v plánu systém známý z navigací, kdy je nainstalována jen základní aplikace a v rámci aplikace bude uživateli umožněno vybrat si atlas dle počtu hub a ten si samostatně stáhnout do zařízení, respektive na paměťovou kartu.

Pro klasifikaci, prováděnou v rámci dílčí aplikace Houbot Benchmark, byla použita implementace *libsvm-androidjni*[1]. Tato implementace pouze upravuje původní LIBSVM pro běh v nativním prostředí na platformě Android, nijak však neupravuje funkčnost či samotné algoritmy. Z toho důvodu není potřeba model mezi těmito implementacemi nijak upravovat.

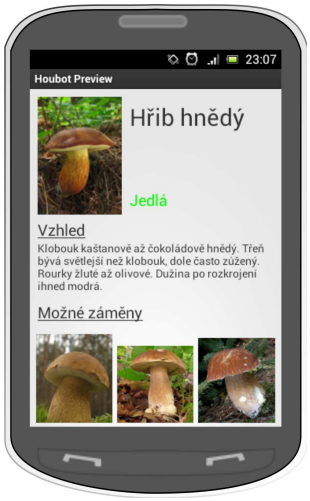
Klasifikační jádro tedy běží v prostředí nativního kódu. V rámci aplikace je v jazyce Java zastoupena klasifikace třídou `Classification`, která realizuje načítání souborů podle zvoleného experimentu a volá nativní funkci `doClassificationNative`. Té jsou předány v parametrech odkazy na datové struktury, kam budou uloženy výsledky.

4.3 Houbot Preview

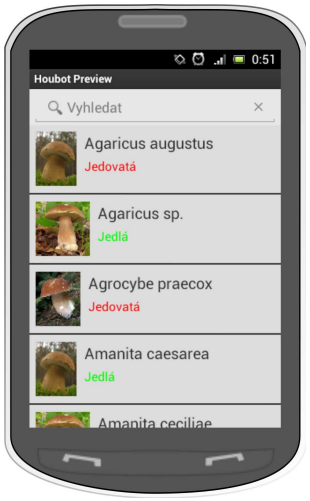
Poslední aplikace, pracovně nazvaná *Houbot Preview*, vznikla pouze jako představení vzhledu možné aplikace. Jejím cílem je pouze demonstrovat uživatelské rozhraní aplikace tak, jak bylo popsáno, včetně všech prvků. Neklade si však za cíl funkčnost těchto prvků, ale jen a pouze jejich vzhled. Jedná se tak o demo aplikaci s několika přednastavenými scénáři použití. Náhled uživatelského rozhraní lze vidět na obrázcích 4.1, 4.2 a 4.3.



Obrázek 4.1: Houbot Preview. Segmentace.



Obrázek 4.2: Houbot Preview. Položka atlasu.



Obrázek 4.3: Houbot Preview. Atlas hub.

Tato aplikace také cílí na experiment popsaný v sekci Uživatelské testování. Tento experiment, přesněji jeho scénáře, byly provedeny prostřednictvím této aplikace.

Aplikace byla implementována podle zvyklostí typických pro aplikace na zařízení Android. Základní stavební kámen, třída, která reprezentuje zobrazitelnou část aplikace, se nazývá *Aktivita*. Každá aplikace by tedy měla mít nějakou hlavní Aktivitu, která ve své podstatě reprezentuje výchozí bod aplikace, neboť se spustí jako první.

Další aktivity pak odpovídají zobrazeným částem aplikace. Aktivita `CameraActivity` je spuštěna v momentě, kdy se uživatel rozhodne rozpoznávat. Zajišťuje snímání kamerou a všechna zmíněná omezení. Po vyfocení je spuštěna aktivita `SegmentationActivity`, která umožňuje uživateli dotykem kreslit na prezenční vrstvu a tak určit body, které spadají do

[1] Kun Li, https://github.com/cnbuff410/Libsvm-androidjni

houby a které do pozadí. Následně provede segmentaci a uživateli zobrazí výsledek, aby měl zpětnou vazbu a mohl případně značit body jiné.

Pokud je spokojen se segmentací, je spuštěna aktivita `ClassificationActivity`, které je předána segmentovaná maska a původní obraz. Ta zavolá extrakci příznaků na základě obrazu a masky. Výsledky extrakce předá klasifikátoru, který provede klasifikaci a navrátí výsledky. Po celou dobu jsou všechny mezikroky hlášeny uživateli, aby na základě zpětné vazby měl přehled, co je právě prováděno, a nedostal pocit, že aplikace neodpovídá.

Na konci je zavolána aktivita `AtlasActivity`, pomocí které jsou výsledky klasifikace zobrazeny a prezentovány uživateli. Tato aktivita je použita také v případě, kdy se uživatel rozhodne použít atlas a procházet houby ručně. Vlastností aktivity je tedy zvolit jaké houby a v jakém pořadí budou zobrazovány v seznamu.

Další aktivitou je pak `AtlasDetailActivity`, která pokrývá potřeby zobrazení atlasové položky se všemi potřebnými informacemi. Tato aktivita se spustí po žádosti o otevření konkrétné položky v atlase, tedy i při otevření položky po prezentaci výsledků rozpoznávání.

Poslední aktivitou je `ConfigurationActivity`, která je přítomna pro doplnění funkčnosti aplikace jako takové. Aktivita umožňuje nastavení několika parametrů snímání, segmentace a klasifikace.

Všechny výše zmíněné části dohromady tvoří celkovou aplikaci, prostě nazvanou *Houbot*. Tato podkapitola však popisuje aplikaci podpůrnou a nazvanou *Houbot Preview*. Z toho důvodu zmíněné aktivity pouze demonstrují svou funkčnost na předem nastavených scénářích, ale ve skutečnosti nemají žádné sofistikované výpočetní jádro. Pro zjednodušení je také možné, že přibudou některé aktivity, které budou vhodně a jednodušeji demonstrovat vzhled a chování celkové aplikace, ve výsledné aplikaci by však postrádaly význam a jejich účel by převzala některá ze zmíněných aktivit. Z toho důvodu jim nebude věnován žádný další prostor.

Kapitola 5

Výsledky a vyhodnocení

Obsahem této kapitoly je vyhodnocení navrženého a implementovaného modelu na základě experimentů popsaných v podkapitole 3.3. Výsledky testování jsou ukazatalem toho, jak je model funkční a jak je použitelný při skutečném nasazení. Také odhalují některé slabiny a nabízejí cesty pro další vylepšení.

Jednotlivé experimenty jsou vyhodnoceny v následujících podkapitolách. Každá podkapitola je rozdělena na dvě části, kdy jsou nejdříve vhodně prezentovány výsledky všech variant experimentu a následně je na základě výsledků utvořeno vyhodnocení experimentu.

Podkapitoly kopírují pořadí toho, jak byly experimenty stanoveny. Nejdříve jsou prezentovány výsledky experimentu na úspěšnost (podkapitola 5.1), následuje experiment na pozici (podkapitola 5.2), dále pak experiment mapující rychlost operací (podkapitola 5.3) a nakonec experiment vycházející z uživatelského testování (podkapitola 5.4).

5.1 Úspěšnost

Testování bylo provedeno na základě 6 tříd, konkrétně jde o houby **Hřib smrkový** (lat. *Boletus edulis*), **Hřiv dubový** (lat. *Boletus reticulatus*), **Křehutka** (lat. *Psathyrella sp.*), **Hřib kovář** (lat. *Boletus luridiformis var. luridiformis*), **Hřib královský** (lat. *Boletus regius*) a **Hřib koloděj** (lat. *Boletus luridus*). Tyto druhy byly vybrány jako houby s největším množstvím snímků v databázi. Toto množství však bylo následně ještě upraveno, neboť zdaleka ne všechny snímky odpovídaly podmínkám snímání. Počet použitých vzorků ke každé třídě a jejich rozdělení na testovací a trénovací lze vidět v tabulce 5.1.

Třída	**Český název**	**Počet vzorků**	**Trénovací data**	**Testovací data**
1	Hřib smrkový	54	40	14
2	Hřib dubový	30	22	8
3	Křehutka	14	10	4
4	Hřib kovář	26	19	7
5	Hřib královský	65	49	16
6	Hřib koloděj	45	34	11

Tabulka 5.1: Počet vzorků dat ke třídě a jejich rozdělení na trénovací a testovací množiny

Jak tyto houby vypadají lze vidět na skupině obrázků 5.1. Je vidět, že houby jsou si převážně velmi podobné.

Obrázek 5.1: Houby vybrané k testování.

První variantou bylo tedy testování pouze na základě barvy. Barevných příznaků je 256 pro klobouk a 256 pro třeň, dohromady 512 dimenzionální vektor celého barevného příznaku. Výsledky testování lze vidět v tabulce 5.2. V řádcích jsou správné třídy a ve sloupcích pak výsledky klasifikace. Na diagonále lze vidět úspěšnost klasifikace konkrétní třídy, ostatní hodnoty odpovídají pravděpodobnostem špatného rozpoznání ve prospěch konkrétní třídy. Všechny hodnoty jsou aritmetickým průměrem pěti stejných experimentů nad náhodně rozdělenými daty.

Správně \ Výsledek	**1**	**2**	**3**	**4**	**5**	**6**
1	0.902	0.042	0.014	0.028	0.000	0.014
2	0.525	0.050	0.350	0.000	0.000	0.075
3	0.700	0.050	0.250	0.000	0.000	0.000
4	0.086	0.000	0.000	0.600	0.000	0.314
5	0.015	0.000	0.000	0.000	0.985	0.000
6	0.000	0.113	0.017	0.113	0.033	0.724

Tabulka 5.2: Výsledek klasifikace na základě barvy. V řádcích jsou korektní třídy, ve sloupcích pak výsledek klasifikace. Lze pozorovat, mezi kterými třídami dochází často k záměně.

Další variantou je testování úspěšnosti na základě příznaku tvaru. Příznak tvaru se skládá z 8 hodnot pro klobouk a 4 hodnot pro třeň, dohromady 12 dimenzionální vektor příznaků. Výsledky jsou opět zobrazeny pomocí tabulky, která zobrazuje i informaci o tom, která třída se se kterou často plete (tab. 5.3). Opět se jedná o průměrnou hodnotu ze všech

pěti samostatných běhů experimentu.

Správně \ Výsledek	**1**	**2**	**3**	**4**	**5**	**6**
1	0.057	0.000	0.000	0.000	0.914	0.029
2	0.175	0.000	0.000	0.000	0.775	0.050
3	0.050	0.000	0.650	0.000	0.300	0.000
4	0.029	0.000	0.000	0.000	0.971	0.000
5	0.108	0.000	0.015	0.000	0.877	0.000
6	0.117	0.117	0.033	0.000	0.733	0.000

Tabulka 5.3: Výsledek klasifikace na základě tvaru. V řádcích jsou korektní třídy, ve sloupcích pak výsledek klasifikace. Lze pozorovat, mezi kterými třídami dochází často k záměně.

Poslední variantou je tedy spojení dvou předchozích. Pro ilustraci bylo spojení provedeno dvěma způsoby. Prvním je způsob, který byl popsán v experimentu, tedy zřetězení vektoru příznaků barvy a vektoru příznaků tvaru (tab. 5.4). Druhým je pak situace, kdy jsou oba samostatné klasifikátory zřetězeny tak, že výsledná klasifikace je dána aritmetickým průměrem dvou původních (tab. 5.5). Výsledky jsou opět prezentovány pomocí tabulky znázorňující pravděpodobnosti včetně záměn.

Správně \ Výsledek	**1**	**2**	**3**	**4**	**5**	**6**
1	0.786	0.043	0.086	0.000	0.071	0.014
2	0.575	0.300	0.025	0.000	0.000	0.100
3	0.400	0.250	0.300	0.000	0.000	0.050
4	0.029	0.029	0.000	0.484	0.029	0.429
5	0.000	0.000	0.000	0.000	1.000	0.000
6	0.017	0.083	0.000	0.050	0.017	0.833

Tabulka 5.4: Výsledek klasifikace na základě tvaru i barvy. Varianta získaná zřetězením příznaků barvy a tvaru do jednoho vektoru, pouze jeden klasifikátor.

Správně \ Výsledek	**1**	**2**	**3**	**4**	**5**	**6**
1	0.914	0.043	0.000	0.000	0.029	0.014
2	0.675	0.200	0.000	0.000	0.025	0.100
3	0.500	0.050	0.450	0.000	0.000	0.000
4	0.086	0.000	0.000	0.571	0.000	0.343
5	0.000	0.000	0.000	0.000	1.000	0.000
6	0.017	0.083	0.033	0.067	0.083	0.717

Tabulka 5.5: Výsledek klasifikace na základě tvaru i barvy. Varianta se dvěma klasifikátory, kdy je výsledek získán jako aritmetický průměr pravděpodobností klasifikátoru barvy a pravděpodobností klasifikátoru tvaru.

Celkovou úspěšnost klasifikátoru lze získat dvojím způsobem. Zaprvé je možné vypočítat aritmetický průměr hodnot na diagonále předchozích tabulek, čímž se získá celková úspěšnost s ohledem na úspěšnost klasifikace jednotlivých tříd. Druhý způsob je získat celkovou úspěšnost jako úspěšnost klasifikace testovací sady. Tento způsob je však náchylný na nestejné množství vzorků jednotlivých tříd v této testovací sadě. Oba způsoby jsou popsány v tabulce 5.6, kde celková úspěšnost získaná druhým způsobem je uvedena v závorkách.

Varianta	**Celková úspěšnost**
Barva	0.585 (0.724)
Tvar	0.264 (0.259)
Kombinace - příznaky	0.617 (0.707)
Kombinace - klasifikátory	0.642 (0.728)

Tabulka 5.6: Shrnutí celkové úspěšnosti klasifikátorů v závislosti na příznacích.

Z uvedených tabulek je patrné, že klasifikace na základě barvy a kombinovaná je poměrně úspěšná. Přibližných 70 % je pro potřeby demonstrace funkčnosti určitě dostačující. Na druhou stranu, úspěšnost klasifikace na základě tvaru úspěšná určitě není, 25 % pro klasifikaci úspěšnou klasifikaci nestačí.

5.2 Pozice

Výsledky a vyhodnocení experimentu zkoumajícího pozici správné houby v seřazeném seznamu hub podle pravděpodobnosti v případě, kdy není správná houba na prvním místě, jsou obsahem této podkapitoly. Stejně jako předchozí experiment, je vyhodnocen na datové sadě popsané v tabulce 5.1.

Jak je popsáno v zadání experimentu, experiment samotný je vyhodnocen ve variantách zvlášť pro příznak barvy, tvaru a jejich dvou kombinací, obdobně jako předchozí experiment. Jelikož však nelze dost dobře výsledky interpretovat jako v předchozím případě, jsou výsledné pozice vidět v tabulce 5.7.

Varianta	**Průměrná pozice**
Barva	3.191
Tvar	2.513
Kombinace - příznaky	2.425
Kombinace - klasifikátory	2.556

Tabulka 5.7: Průměrná pozice správné houby při neúspěšné klasifikaci v závislosti na variantě experimentu

Výsledné hodnoty pozice jsou aritmetickým průměrem pěti samostatných běhů experimentu s náhodně rozdělenou testovací a trénovací množinou. Výsledek tedy není ovlivněn konkrétním nastavením testovací a trénovací sady. Minimální, a tedy nejlepší hodnota pozice, je 2. Této hodnoty by bylo dosaženo v situaci, kdy by se v případě neúspěšné klasifikace správná třída nacházela vždy na druhé pozici.

Z výsledků je také jasně patrné, že výsledná pozice je lepší při použití obou příznaků. Zanedbatelný není ani úspěch experimentu při použití pouze tvarového příznaku, jehož

výsledek je takřka stejný jako při použití kombinace příznaků. Za zmínku stojí ještě relativní neúspěch pozice na základě příznaku barvy, narozdíl od úspěchu v předchozím experimentu.

5.3 Rychlost

Experiment na testování rychlosti základních operací je důležitý z důvodu, že je cílem aplikaci implementovat a přizpůsobit pro běh na mobilním zařízení. Vyhodnocení proběhlo na třech rozdílných zařízeních, která jsou pro přehlednost s výčtem některých parametrů zobrazena v tabulce 5.8.

	SE Xperia Mini Pro	**HTC One V**	**Asus Transformer**
Model	SK17i	T320e	TF101
Architektura CPU	ARMv7	ARMv7	ARMv7
Frekvence CPU	1x1000MHz	1x1000MHz	2x1000MHz
Typ SoC	Qualcomm MSM8255	Qualcomm MSM8255	NVIDIA Tegra 2
Verze OS	4.0.4	4.0.3	4.0.3
RAM	512MB	512MB	720MB

Tabulka 5.8: Přehled zařízení na nichž byla rychlost testována

Jak je popsáno v sekci Rychlost, je sledována rychlost klasifikace. Délka běhu klasifikace ja změřena pětkrát pro každé zařízení a variantu příznakového vektoru a výsledek je dán aritmetickým průměrem těchto pěti hodnot. Výsledky jsou zobrazeny v tabulce 5.9.

Model zařízení	**Varianta**	**Naměřená doba [s]**
SK17i	Barva	1.858
	Tvar	0.162
	Kombinace	1.943
T320e	Barva	1.543
	Tvar	0.121
	Kombinace	1.554
TF101	Barva	1.293
	Tvar	0.103
	Kombinace	1.345

Tabulka 5.9: Rychlost klasifikace v závislosti na variantě experimentu a zařízení, na kterém experiment proběhl

Z výsledků je zřejmé, že bez ohledu na zařízení je rychlost klasifikace silně ovlivněna především dimenzí příznakového vektoru. Taktéž vyplynulo, že množství výpočetních jader procesoru nehraje ve výsledku významnou roli. Toto je však nejspíše způsobeno faktem, že knihovna LIBSVM v základním stavu běží pouze na jednom jádru. Zrychlení je tedy pravděpodobně dáno jen tím, že se zátěž klasifikace a zbytku systémových operací rozloží na dvě jádra.

5.4 Uživatelské testování

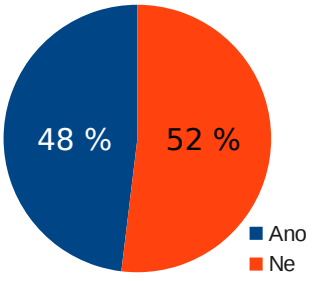
Uživatelské testování sestává ze dvou nezávislých testování na dvou různých skupinách uživatelů. První varianta je prováděna prostřednictvím aplikace Houbot Preview a sleduje počet kroků, dobu provádění a reakce uživatelů při plnění jednoho ze tří scénářů. Vzorek lidí byl vybrán náhodně za účelem rovnoměrně rozloženého vzorku co se týče věku, technické zdatnosti a pohlaví uživatelů.

Celý experiment s jedním uživatelem netrval déle než pět minut. Uživateli bylo představeno v pár větách prostředí aplikace, její cíl a výhody s důrazem na jednotnost v představení aplikace všem uživatelům. Byly zmíněny některé vlastnosti, které uživateli pomohou dosáhnout cíle, například že vyhledávat v atlase lze pomocí barvy či listováním, při segmentaci mají být označené plochy v objektu či pozadí, nesmí však přecházet přes hranice, a že pro správné sejmutí je nutné zaměřit houbu do siluety na displeji a hlídat její správnou výšku za pomoci linky.

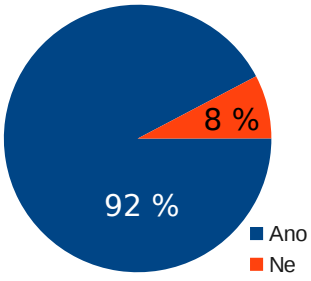
Experiment byl prováděn na vzorku 20 lidí. Výsledky z experimentu ukázaly, že před segmentací je třeba uživatele dobře informovat, například prostřednictvím průvodce. V atlase se uživatelé pohybovali s přehledem, opět však nastal problém s nedostatečně představenými inovacemi v ovládání. Celkově byla hodnocena aplikace jako přehledná.

Druhou částí je uživatelský dotazník nad hypotetickou aplikací. Tito respondenti aplikaci nikdy neviděli a jedinou informaci, kterou o aplikaci měli, jim byla sdělena v záhlaví dotazníku (viz sekce Uživatelské testování). Dotazník byl realizován prostřednictvím formuláře ve službě *Google Disk* a byl rozšířen mezi dotázané především prostřednictvím sociálních sítí, e-mailu a osobního upozornění.

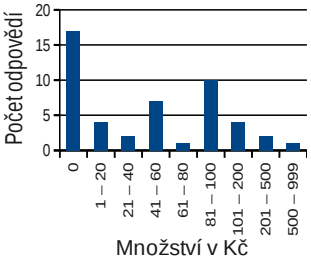
Z tohoto dotazníku vzešlo hned několik zajímavých faktů. Na obrázku 5.2 je jasně vidět, že polovina dotázaných vlastní mobilní zařízení, na které tato práce cílí, a téměř všichni dotázaní si pak telefon sebou berou do lesa (obr. 5.3). Detailní rozbor pak odhalil, že z těch, kdo si s sebou telefon do lesa nebere, nikdo nevlastní telefon cílového typu (Android, dotykový displej, kamera).



Obrázek 5.2: Vlastníte cílové zařízení?



Obrázek 5.3: Berete si s sebou mobil do lesa?

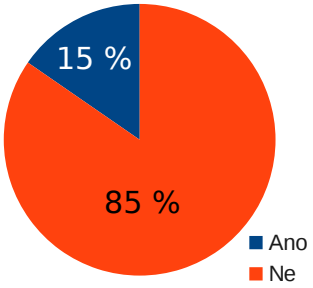


Obrázek 5.4: Zaplatili byste za aplikaci?

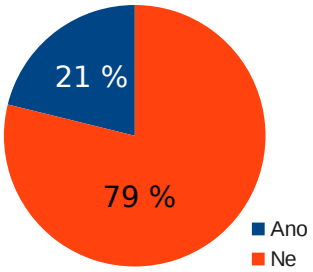
Dalším zajímavým zjištěním, které z dotazníku vyplynulo, byla skutečnost, že dvě třetiny odpovídajících by byly ochotny za aplikaci zaplatit, a to průměrně okolo **50Kč** (obr. 5.4). Vyskytly se i odpověďi, ve kterých dotázaní uváděli, že by zaplatili mnohem víc (často tisíce), tyto odpovědi byly však vyřazeny kvůli zjevnému vychýlení od reality a dnešních trendů.

Posledním faktem, který stojí za pozornost je, že jen přibližně **15 %** dotázaných odpovědělo, že při houbaření používá atlas (obr. 5.5). Tento fakt není příliš příznivý, neboť tato

aplikace se snaží atlas nahradit lepší formou. Otázkou tedy zůstává důvod, proč uživatelé atlas nepoužívají. Může jít o nepoužívání atlasu z důvodu složitosti v něm houbu nalézt, špatnými zkušenostmi, ale také třeba nechutí používat další pomůcky a v lese se zabývat něčím jiným, než sběrem několika známých hub.



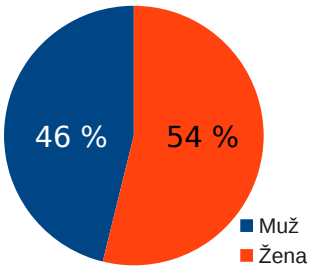
Obrázek 5.5: Používáte kapesní atlas?



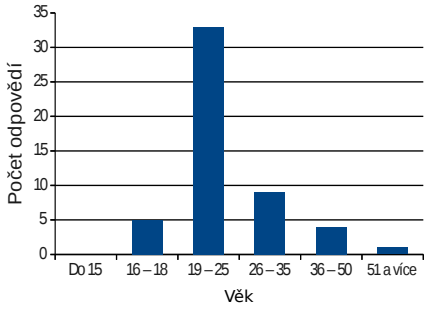
Obrázek 5.6: Měli byste strach aplikaci používat?

V prvním případě by tak měla aplikace šanci uživatele oslovit, neboť její interaktivita může být přinejmenším v začátcích pro uživatele zajímavá. Také odpadá nutnost brát s sebou něco víc, než by si uživatel do lesa stejně s sebou vzal. V druhém případě by bylo těžší uživatele oslovit, i zde ale může pomoci to, že si uživatel s pomocí aplikace může rozšířit znalosti z mykologie.

Posledním výsledkem získaných z dotazníku je převážná nebojácnost respondentů vůči aplikaci. Opět může jít například o nedomyšlení důsledků či přílišné důvěry v techniku, ale stejně tak například o víru ve vlastní svědomitost a zdravý rozum při používání takové aplikace. V každém případě, aplikaci by se bálo používat pouze **21 %** dotázaných (obr. 5.6), což je alespoň pro začátek určitě dobrá zpráva.



Obrázek 5.7: Jste žena nebo muž?



Obrázek 5.8: Kolik máte let?

Vzorek respondentů tak, jak byl bez řízení získán, je rovnoměrně rozdělen mezi muže a ženy (obr. 5.7). Věk však bohužel rovnoměrně rozdělen není (obr. 5.8), což je dáno především informačními kanály, kterými byl dotazník šířen, a faktem, že dotazník existuje pouze v elektronické podobě.

Poslední součást experimentu počítá s daty získanými na základě širokého používání aplikace. Tato data nejsou dostupná z dvou důvodů. První je, že navržená aplikace prozatím nebyla implementována v plné funkčnosti, ale pouze v rozsahu nutném pro demonstraci.

Druhým důvodem je, že vývoj probíhal přes zimu a jaro, tudíž ani v případě, kdy by aplikace byla plně funkční, nebylo by co rozpoznávat.

Kapitola 6

Závěr

Cíl této práce, demonstrovat rozpoznávání objektů v obraze na platformě Android prostřednictvím praktické aplikace, byl splněn.

Jako objekt, který se v rámci demonstrace v obraze rozpoznával, byly zvoleny houby. Aplikace měla ambice stát se praktickou pomůckou pro houbaře, která do houbaření přinese další rozměr a napomůže tak rozšíření povědomí od houbách a hlavně o rozmanitosti jejich druhů.

Jako segmentační metoda byla zvolena interaktivní metoda GraphCut. Příznaky, na základě kterých byla provedena klasifikace, byly zvolena barva a tvar. Příznakový vektor byl vstupem klasifikátoru SVM.

Výsledky rozpoznávání jsou prezentovány formou seřazeného seznamu několika nejvíce pravděpodobných tříd. Z toho důvodu není nutné, aby správná třída byla nezbytně na prvním místě.

Pro potřeby demonstrace byly vytvořeny tři aplikace, které dohromady realizují všechny popsané kroky. Aplikace **Houbot Development** běží na stolním počítači a realizuje segmentaci a extrakci příznaků. Další aplikace, **Houbot Benchmark**, provádí na mobilním zařízení samotnou klasifikaci. Poslední aplikace, **Houbot Preview**, pokrývá problematiku uživatelského rozhraní případné celkové aplikace.

Výsledky testování a experimentů ukázaly, že aplikace má s to dostát ambicím. Úspěšnost klasifikace dosahuje nejlepší hodnoty téměř **73 %**, pozice houby v seřazeném seznamu pak je v nejlepším případě okolo **2.5** a délka klasifikace se pohybuje v blízkosti **1.3 s**.

Do budoucna je vhodné přidat některé další příznaky, na jejich základě může být klasifikace úspěšnější. Dále je nutné rozšířit klasifikaci na větší množství tříd, aby se dostálo zmíněným výhodám vůči kapesnímu knižnímu atlasu. Nutnost vystředit houbu a správně ji ve fotografii umístit zvyšuje náchylnost na uživatelskou chybu, takže je v plánu toto omezení nahradit nějakou metodou detekce houby s automatickou segmentací. Na aplikaci je v plánu pokračovat a zmíněné nedostatky odstranit.

Z práce přímo výchází článek *Support of Acquisition of Recognition of Mushroom Images* na prezentovaný na konferenci *Central European Seminar on Computer Graphics 2013*, který byl publikován ve sborníku. Celé znění článku je k vidění v příloze. Obdobný článek založený na této práci byl také přihlášen na soutěž *EEICT 2013*, na základě čehož byl publikován ve sborníku této soutěže a po úspěšné obhajobě byl také oceněn prvním místem v kategorii *Zpracování signálů, obrazů a dat, Bakalářská kategorie.*

Literatura

[1] Kršek, P.: *Základy počítačové grafiky: IZG*. Brno: Fakulta informačních technologií, 2006, 89 s., studijní opora [online].

[2] Gonzalez, R. C.; Woods, R. E.: *Digital Image Processing*. Upper Saddle River: Pearson, třetí vydání, 2008, ISBN 0-13-168728-x, 954 s.

[3] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Texts in computer science, London: Springer, 2010, ISBN 978-1-84882-934-3, 812 s.

[4] Al-amri, S. S.; Kalyankar, N. V.; Khamitkar, S. D.: Image Segmentation by Using Thershold Techniques. In *Journal of computing*, ročník 2, Květen 2010, ISSN 2151-9617, s. 83 – 86.

[5] Zbořil, F.: *Základy umělé inteligence: IZU*. Brno: Fakulta informačních technologií, 2008, 142 s., studijní opora [online].

[6] Horowitz, S.; Pavlidis, T.: Picture Segmentation by a Tree Traversal Algorithm. *Journal of the ACM*, ročník 23, č. 2, Duben 1976: s. 368 – 388.

[7] Horowitz, S.; Pavlidis, T.: Picture Segmentation by a Directed Split and Merge Procedure. In *Computer Methods in Images Analysis*, 1977, s. 101 – 11.

[8] Ohlander, R.; Price, K.; Reddy, D. R.: Picture Segmentation Using a Recursive Region Splitting Method. *Computer Graphics and Image Processing*, ročník 8, č. 3, Prosinec 1978: s. 313–333, ISSN 0146-664X.

[9] Brice, C. R.; Fennema, C. L.: Scene Analysis Using Regions. Technická Zpráva 17, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Duben 1970.

[10] Wu, Z.; Leahy, R.: An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, ročník 15, č. 11, Listopad 1993: s. 1101–1113, ISSN 0162-8828, doi:10.1109/34.244673.

[11] Papadimitriou, C. H.; Steiglitz, K.: Mineola (New York): Dover Publications, 1998, ISBN 0-486-40258-4, 496 s.

[12] Boykov, Y.; Jolly, M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, ročník 1, 2001, s. 105 – 112, doi:10.1109/ICCV.2001.937505.

[13] Boykov, Y.; Veksler, O.; Zahib, R.: Efficient Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 23, 2001: s. 1222–1239.

[14] Duda, R. O.: *Pattern classification*. New York: Wiley, druhé vydání, 2001, ISBN 0-471-05669-3, 654 s.

[15] Theodoridis, S.; Koutroumbas, K.: *Pattern Recognition*. Amsterdam: Academic Press, druhé vydání, 2003, ISBN 0-12-685875-6, 689 s.

[16] Open Handset Alliance: Android. [online 11.3.2013]].
URL http://android.com/

[17] ARM Holdings: ARM - The Architecture For The Digital World. [online 11.3.2013]].
URL http://arm.com/

[18] Sloss, A. N.; Symes, D.; Wright, C.: *ARM system developer's guide: designing and optimizing system software*. Amsterdam: Elsevier, 2004, 689 s.

[19] Matti, D.: Mushroom Recognition. Technická zpráva, École polytechnique fédérale de Lausanne, 2010.
URL http://mmspg.epfl.ch/files/content/sites/mmspl/files/shared/Semesterproject_mushroomrecognition.pdf

[20] Smotlacha, M.: *Smotlachův atlas hub: oficiální příručka pro určování jedlých a jedovatých hub*. Praha: Cesty, 1999, ISBN 80-718-1311-7, 271 s.

[21] Baranovic, R.: Nahuby.sk. [online 11.3.2013]].
URL http://nahuby.sk/

[22] Boykov, Y.; Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 26, č. 9, Září 2004: s. 1124 – 1137, ISSN 0162-8828, doi:10.1109/TPAMI.2004.60.

[23] Delong, A.; Osokin, A.; Isack, H.; aj.: Fast approximate energy minimization with label costs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, ISSN 1063-6919, s. 2173–2180, doi:10.1109/CVPR.2010.5539897.

[24] Kolmogorov, V.; Zabin, R.: What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 26, č. 2, Únor 2004: s. 147 – 159, ISSN 0162-8828, doi:10.1109/TPAMI.2004.1262177.

[25] Chang, C.-C.; Lin, C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, ročník 2, 2011: s. 27:1–27:27.
URL http://www.csie.ntu.edu.tw/~cjlin/libsvm

Dodatek A

Článek na konferenci CESCG 2013

Příspěvek s názvem *Support of Acquisition of Recognition of Mushroom Images* je založený na této práci. Byl publikován v rámci sborníku při konferenci *Central European Seminar on Computer Graphics 2013*, kde byl také odprezentován. Příspěvek je napsán v anglickém jazyce.

Support of acquisition of recognition of mushroom images

Martin Simon*
Supervised by: prof. Dr. Ing. Pavel Zemcik†

Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

Proper recognition of mushrooms is one of the key safety issues in mushroom picking activities widely spread in Czech Republic, Slovak Republic, and other countries in Central Europe. This contribution proposes a novel approach to support at recognition of the mushrooms through mobile communication devices, such as cellphones. The user - mushroom picker - is supposed to take a picture of a mushroom in question through the mobile device which then attempts to recognize the mushroom or to suggest a set of possible similar mushrooms in order to simplify their recognition by the users. The recognition process itself uses shape parametrization as well as texture and color properties of the mushroom. The mushroom picking prototype application runs on Android devices that are widely spread and inexpensive enough to enable wide exploitation by users. The contribution contains description of the basic principles and presents the content of the future Bc. thesis of the author.

Keywords: Object Recognition, Mushroom Recognition, Mushroom Picking, Mobile Device, Android

1 Introduction

Mushrooming is very popular in Central European countries, specifically in Czech Republic, Slovak Republic and few others. Although the mushrooming has a long history, even the experienced mushroom pickers can recognize only between 50 and 300 species of mushrooms. Other mushrooms they search in atlases, which are often only limited pocket versions. In order to minimize the weight, this kind of atlases are focused only on the most common mushrooms, whose count is at most hundreds. Moreover, the most common mushrooms are often known to the mushroom pickers. One way or another, the amount of mushrooms that grow here quite commonly is more than 1500.

Nowadays, mobile devices are not supposed only to call or text. There are still more and more statefull multimedial devices that allow for example watch high resolution video or play games. However, their real usable power is much better. Operations such as image processing or image recognition are no longer out of possibilities of these devices. Furthermore, the mobility of these devices is for operation such as image recognition - in this case mushroom recognition - of a great benefit. This contribution aims just this way.

The target platform is wide spread and powerful enough mobile *Android* device. Such a device can contain an electronic atlas allowing user to search and filter among much more samples than with the paper atlas. However, this contribution is aimed at the recognition based on the images acquired by the device camera. To use the mobility of these devices, the recognition does not depend on any internet or other connection which could be hardly reachable deep in the forest.

The main purpose is to design and implement an application running on a mobile Android device. This application sets the goal to take a mushroom image directly in the nature - it is a good habit to not pick any mushroom you do not want to take with you but to leave it grow. The segmentation is done on basis of the user interaction. Then the mushroom is recognized and the results are presented as a list of the most similar mushrooms. In an ideal case, the application returns only one mushroom, the one correctly identified.

It is obvious, from the previous paragraph, that the atlas of mushrooms is a part of the proposed application. This atlas consists mainly from brief description and an image of a mushroom and the main purpose is to present results. The results could be also the list, the subset of the original atlas, so the description is quite important to user to find the right mushroom. Of course, the user can browse in the electronic atlas as well as in paper atlas.

Such an application can be quite dangerous if the user fully trust its results. This application, especial;y the recognition machine, should be always used with the knowledge of its experimental character. Reckless relying on given results could cause injuries and in extreme case even the death.

Section 1.1 describes materials and algorithms on which this contribution is based. This section also includes an

*xsimon14@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

overview of the previous similar projects. The approach itself is described in Section 2. That section is divided into a subsection about a preprocessing (2.1) which includes also a sensing and a segmentation, a subsection about feature extraction (2.2) and a subsection about classification itself (2.3), where the classification is described as is used in this contribution. Section 3 addresses design draft and an implementation of the application as well as results presentation. At the end, in Section 4, a possible way to evaluate this contribution will be proposed with expected results.

1.1 Related work

The basic approach of this contribution is to help to recognize a mushroom by its shape. For this purpose, the parametric model [12, 14] has been chosen. Since the shape of a mushroom is often not reliable, this contribution reflects also color histograms [13] of a cap and a stem as well as few other specific mycological features, as describes Matti [11], or some independently chosen ones.

Not many contributions have been published about this issue despite the fact of widely spread mobile devices, mushrooming in this region, and computer vision upswing. This contribution is mainly based on a master semester project of *Damien Matti's Mushroom recognition* at Ecole polytechnique federale de Lausanne[1] from 2010 [11]. The approach of that project is to classify mushrooms with Android mobile telephone. One of the differences between mine and his solution is where the recognition part is done. He uses a client-server solution, so he uses the mobile device only as a client which performs just the segmentation and then he sends the preprocessed image to the server, where the recognition is actually done. Another difference is that he does not aim to identify specific kind of mushroom, but wants only to find out if the mushroom is edible or not.

For the segmentation part and an extracting mushroom from its background, he uses the GraphCut method [3, 4] based on a graph theory. After taking a photo, he asks the user to mark a background and a foreground.

In the conclusion of that project, Matti proposes a way where to go as well as methods which to use or not use. He mentions mainly the necessity of a big enough database of training data. For the purpose of this contribution, I contacted the server *nahuby.sk* [1] which provided me for this academic purposes a huge database[2] of high quality images of mushrooms, which are safely marked with assistance of specialists. He proposed also to use mushroom specific features such as ring detection, white blob detection, gills and pores recognition, etc.

[1]The report from the project is available at <http://mmspg.epfl.ch/files/content/sites/mmspl/files/shared/Semesterproject_mushroomrecognition.pdf>

[2]Over 600 of single mushroom species with minimal amount of 20 images for every kind - totally about 35 thousand of images - or over 1000 of single mushroom species with minimal amount of 10 images for every mushroom kind - about 45 thousand of images

2 Recognition

The recognition of a mushroom based on taken image consists from several parts. At the beginning, preprocessing is described, including the sensing and the segmentation of taken image. In this section, methods and ways how to take an image are proposed with usage of possibilities of mobile Android device. It also contains the extraction of the mushroom from the background and creating the mask of the cap, the stem and unused background. This mask combined with the original sensed image are used in the very next part.

The second part of this paper handles features extraction for classification itself. All the preprocessed parts of sensed image are used there and the features are extracted to use in parametric model. Also color histograms are computed. All of these parameters are then decorrelated using method called *Principal Component Analysis (PCA)* [2, 8] which helps to select only decorrelated values and significantly decreases the amount of the dimensions of the original vector. All of these changes are made with only an unimportant loss of information.

Finally, the last part concerns the recognition itself. The features obtained in the previous part are an input of the *Support Vector Machine (SVM)* [2, 10, 14] classifier which is used in this contribution. Actually, this contribution does not include an implementation of SVM but only uses an already implemented system.

2.1 Sensing, Preprocessing and Segmentation

Many of the features typical for mobile Android devices are used for a preprocessing in general and also specifically for the recognized objects - mushrooms. The application uses auto-focus feature which is built-in in the most of nowadays mobile devices with camera. It uses a stability sensor which allows us to restrict the sensing angle, too. This is done because the angle could be quite important. Sensing under a correct angle (Figure 1) shows us the top of the cap, the bottom of the cap as well as curves of the stem or the cap. On the contrary, a wrong sensing angle (Figure 2) could skew the curves of the edges or totally hide for example the bottom of the cap.

Another important feature is to have the LED light constantly on. That can help with unification of brightness of sensed images. It also helps to eliminate shadows which could eventually look like edges occuring where they are not present. The profile of a mushroom is, in case of simplification, drawn on device preview surface to help the user to center the sensed mushroom (Figure 3).

Also, a line is supposed to divide the cap and the stem. This line is not really used for the stem-cap split algorithm, but with the requirement for the correct angle helps to get the real, not skewed, shape parameters. To center the mushroom, is quite important because the post-

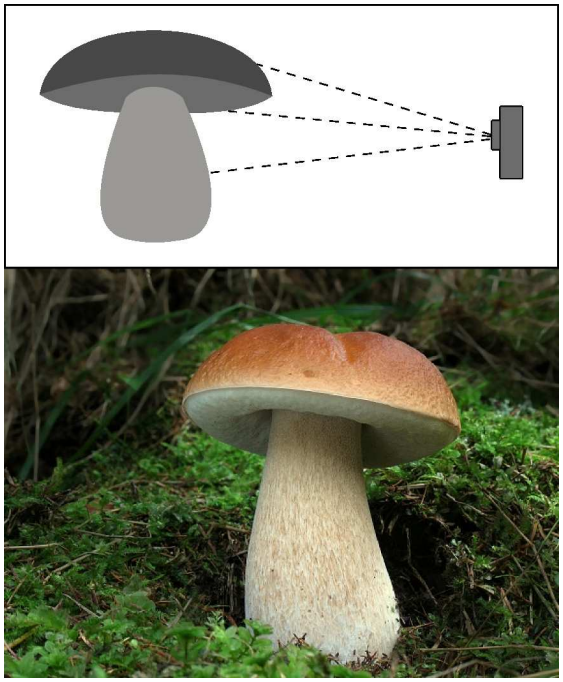


Figure 1: Illustration of the correct sensing angle with the taken image. On the example picture you can see clearly all parts of the mushroom body.

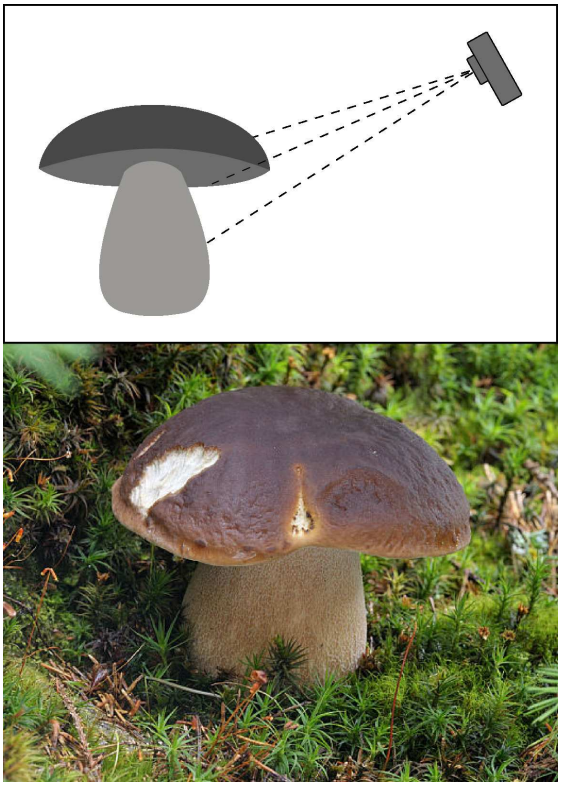


Figure 2: Illustration of the wrong sensing angle with the taken image. On the example picture you can not see clearly all parts of the mushroom body. E.g. the bottom of the cap is not visible at all.

segmentation algorithm counts with the fact that the middle point belongs to the mushroom body.

The extraction of the mushroom from its background is based on GraphCut, similarly to how Matti [11] performs it. This contribution uses a library implemented by *O. Veksler* [5, 7, 9, 16]. The user is asked to mark foreground - the mushroom, and the background. These marked areas are being hard constraints in the segmentation. The data costs are set as negative log-likelihood of color histograms of the hard constraints marked by the user. Then, the primary mask is created.

This mask is smoothed by an algorithm to reduce of isolated fields - island. The intention is to have only one coherent area marked as the mushroom and the rest marked as the background. The algorithm mentioned above is an implementation of the *Uniform-cost search* with the cost set to the *Manhattan distance* from the point to the point-root. The aim is to search the path from the isolated *island* to the middle point if the island is marked as the foreground, or the path to the [0,0] point, which is supposed to be a part of the background, in case the island is marked as a background. One way or another, if the path exists, the island is not isolated. Otherwise, the island is isolated and it should be re-marked.

This way, the smooth masks of the foreground and the background are created. The next step is to split the mask of the mushroom into two parts - the stem and the cap. To achieve that, the border points between the cap and the stem are found and the GraphCut algorithm is re-run on the area given by these two points and corresponding intersections of the top of the foreground mask and lines in direction of y-axis and determined by these two points. This GraphCut run does not have any hard constraints, is fully automated and uses the rest of the mask to compute histograms of the cap and the stem to set data costs (See the Figure 4).

We have got the mask of the stem and the cap. Also, we have the original color image of mushroom of the same size as the mask. These two matrices are inputs of the feature extraction part.

2.2 Feature extraction

The goal of this section is to show which features are used and how they are extracted, so this section follows the previous one. After applying the mask, we have an array of points that belong to the cap, or, alternatively, to the stem. These two arrays are now used to extract and compute appropriate features that can be inputs of the classifier.

The first feature what is extracted, is the shape information. For this purpose, the left most and the right most points of the cap are connected by a straight line, which is divided into fifths. The lengths from this line to the top border of the cap, or the bottom border of the cap, are the shape parameters. These values are computed from the line, so it should be rotational invariant (Figure 5).

Figure 3: Illustration of taking photo of a mushroom with a visual help

A very similar operation is done with a stem of a mushroom. We have the top and the bottom point of the stem and the length between them is divided into fifths, again. Then the lengths from the left edge point to the right edge point on every intersection are measured (Figure 6).

Another gathered features are the color histograms of the stem and the cap. Also the color depth is reduced to 3-3-2 bit RGB fixed palette, which means 256 colors. The reason for this color reduction is that we use the whole histogram as a part of the multidimensional vector used in the SVM classifier. As such, the information compression is very desired. The advantages of this reduction are much bigger than disadvantages, namely loss of information.

The reduction is done by ignoring the least significant bits. We use only the 3 most significant bits from the red value of the color, the 3 most significant bits from the green value and only 2 most significant bits of the blue value. These values are then concatenated in order RGB, so the palette is 256 color length. All histograms computed in case of this contribution, use this 3-3-2 bit color palette.

The last but finally not the least important feature is presence of gills or pores on the bottom side of a cap. Only two classes exist, gills and pores, so it can be possible to train the SVM classifier on the whole dataset and let it to distinguish between them.

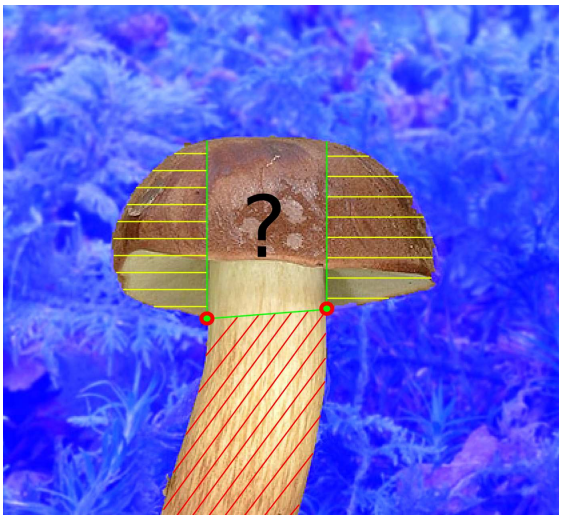


Figure 4: Split of the cap and the stem. Two mentioned points are marked. The *red* area is a stem, so it is used to compute the stem color histogram. The *yellow* areas are parts of the cap, so there are analogically used to compute the cap color histogram. The area marked by the question mark is going to be segmented and divided into the cap and the stem. The unused background in blue colored.

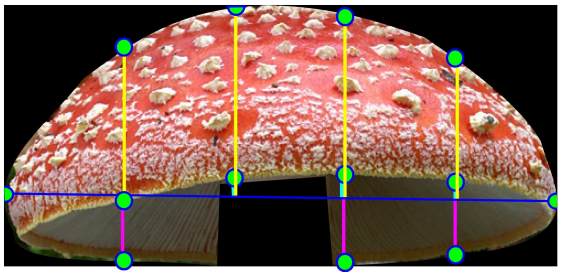


Figure 5: Illustration of the cap segmentation for parameters extraction

2.3 Mushroom classification

All the features described in previous section combined together form a feature vector. Actually, the values obtained from the histograms feature extraction, the shape feature extraction, the ring detection and the gills or pores presence, are concatenated into one single multidimensional vector. This vector is supposed to be an input of the SVM classes. Unfortunately, this feature vector could be very high dimensional and probably correlated. To decorrelate this vector and decrease the dimensionality, the PCA method is used. This can help us to find witch features or what dimensions of the whole feature vector are really important and where the classes are the most variant.

The classification itself is done using the SVM classifier implemented in *LIBSVM* [6]. The input of this classifier, as it is written above, is the whole feature vector consisting of single features extracted from the widespread database of marked mushroom images [1].

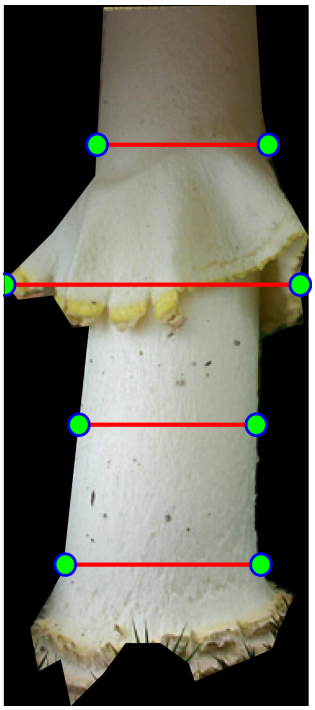


Figure 6: Illustration of the stem segmentation for parameters extraction

The database consists of about 35 thousand of mushroom images from many photographers. Although the fact that all mushrooms in this database are safely identified, not all images are suitable to the classifier training. The recognition requires to take a photo under specific angle and with the middle point on the mushroom body. All of this is needed to not have skewed the real shape of the mushroom. It is obvious that the very same requirement is need to be requested to the training set.

Every mushroom image needs to be manually reviewed for the proper sensing angle and mushroom centering, then manually segmented using graph cut and after that, finally, features can be extracted and used for the training.

3 Implementation

The mushroom recognition in images proposed in this contribution is realized using mobile device on Android platform. The aim is to design and implement an application running on Android mobile device to support recognition of mushroom in the nature. Individual parts of this task are the following ones:

- **Take an image of the mushroom.** This part allows the mushroom picker to take an image using built-in camera in correct position and situation with unified brightness and auto-focused to make the recognition as much stable as possible. To achieve this purpose this application uses the sensors so typical for such devices as much as possible. The used sensors are the motion sensor for sensing under correct angle, LED light source for shadow elimination and color unification and possible auto-focus aimed in the middle of the sensing area to eliminate the foreground - mushroom - blur. The draft of the sensing screen you can see on Figure 3.

- **Extract the mushroom from its background.** The application does not consider the background (even though it might be also a feature useful for a recognition), so for the next steps of recognition, it is necessary to extract the mushroom from its background. The user is asked to mark the foreground and the background and based on these seeds the segmentation is done and the mask is created. The second pass of graph cut splits the cap and the stem.

- **Feature extraction.** From the original image of the mushroom and the mask the earlier mentioned features are extracted and computed to describe the given mushroom as much as possible. These features are concatenated into one multidimensional feature vector.

- **Classification.** The feature vector is an input of the already trained classifier. It determines classes, into which the feature vector most probably fits.

- **Show the most probable results.** The results are presented as a subset of the atlas. In the ideal case, only one mushroom with maximum probability is shown. The proposed design of result presentation is shown in Figure 7 and Figure 8.

This section also includes a draft of user interface with the aim to simplicity and usability including focus on few important elements. Some aspects of the implementation itself are mentioned, too.

3.1 User interface

The application ambition is to help a mushroom picker with the classification of the found mushroom. To achieve such a goal, it is also good to spend some time with the design of the user interface and primarily the presentation of the results.

The application itself is not only recognition machine, but also the mushroom atlas with brief description, mainly of the visual elements, information about an edibility, eventually with what mushroom can be mistaken. The atlas itself is designed as a list of mushrooms with names and images (Figure 7). The list allows to search by name and possibly filtering by features. If the classifier result is a list of mushrooms and not the single one, the list will be subset of this list. Some percentage or edibility sign should be visible in this case, too.

The appearance of the mushroom screen (Figure 8) includes an image, name, brief description, edibility sign,

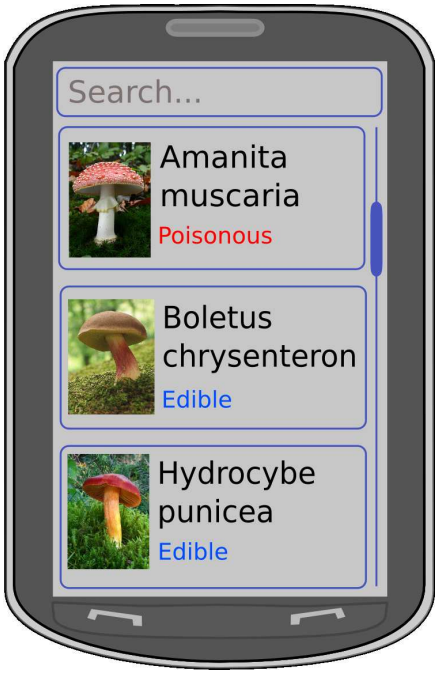


Figure 7: Draft of design: List of mushrooms

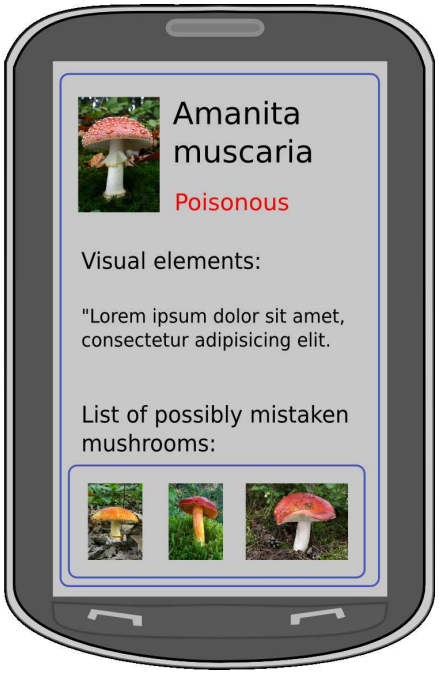


Figure 8: Draft of design: Detail of the mushroom

and the list of the most similar mushrooms generated automatically after the classifier learning. Such list of mushrooms is very important because the classifier in ideal case returns only one result but the user should know whether there are any similar mushrooms, even though the classifier excludes them for any reason.

The final important design element is a screen of a camera where the user takes an image of mushroom and where he is then informed about the recognition progress (Figure 9). In that case, the preview surface is spread to the fullscreen for the reason of maximizing of feedback. Before the user takes a photo, he/she should be informed about the correct sensing angle with signal icon and in the middle of the screen should be a profile of mushroom to help the user with centering the sensed mushroom. In this screen, the user should be informed about status of recognition parts. This is important in cases when the recognition takes more time and the user might think the device is not responding.

3.2 Implementation

The implementation on mobile Android device must be in this contribution with a knowledge of an implementation on a system based on ARM instruction set. Of course, we use the interface that the Android system give us, but it is needed to know where the computation is really done.

To develop a successful application, it is good to follow these specifics. On one hand, it is obvious that the computing power can not be compared with powerful desktop computers but on the other hand we can not say it is not powerful enough. A similar situation occurs in the field of memory possibilities of these devices. On one hand, the size of these memory units is still quite limited and it is not reasonable to waste it. On the other hand, the Moore's law is still valid.

One way or another, such devices have several limitations which could ruin the whole experiment. The first one can be the memory requirement. To overcome this, it is good to realize that the application is not supposed to be a mushroom gallery. Yes, the atlas without images is not an atlas at all, but one bright image per every mushroom kind is more than enough.

The second one could be the memory again, but this time the operation memory. The *static* memory described above could be solved with an external storage, but this *dynamic* memory could be much bigger problem. But even in this case the problem is not unsolvable. For example, in case of segmentation, we need to build a graph with $width \times height$ of nodes. This can be a lot of allocated memory, probably more than such devices can offer. But this issue can be solved for example by resizing the original image to lesser matrix, computing the mask and resizing the mask back. The borders should be preserved and it is not a big deal that the borders are not so smooth. The important fact is, that if we compute this mask in an image of a half width and a half size, the count of nodes in the graph will be $(width \times height)/4$.

4 Testing procedure

The methods proposed in this contribution were chosen on basis of studying and they have not been fully imple-

Figure 9: Draft of design: Recognition screen

mented and verified yet. The implementation itself is a part of future Bc. thesis of the author.

The main aim is to choose the most similar mushrooms as the subset of the whole atlas. The one hundred percent success is not expected and the application is not supposed to recognize mushrooms instead of the mushroom picker. It only offers the most similar mushrooms and expects and active approach of the user. Very real scenario is that the mushroom picker suspects that he found some kind of Boletus, but he does not know which exactly. If the result of the recognition in that model situation would be a subset of ten different Boletus mushrooms, it would not be useless. The user gets the subset which would have to complicatedly search in other case.

The user interface has been tested informally on an adhoc sample of almost 20 people of my surrounding. This sample consists mainly from amateur mushroom pickers, but the age, technical skills or expectation has been spread into whole spectrum.

The first determined result from the user interface testing, is the time length of classification itself. Almost everyone said that the time longer than one minute is on the edge of usability, because they are able to find the mushroom in the atlas faster.

Other conclusions have been used in the implementation design. It was namely the necessity of more significant visual help during the sensing (Figure 3) and an interest to be more involved in the recognition. The last interest has two reasons. The first reason is that in the recognition would be more interactive, it can take more time and user will not register that. The second reason is that the user know that the segmentation is based on his interaction and is afraid that he could do it wrong and make the recognition failed. To solve this, one more screen is added just after the image is segmented on basis of user interaction. The screen is an easy feedback about the segmentation and allows to the user to go a step back and do the segmentation once more.

It is interesting to monitor the informative value of single extracted features. Parametrization used to shape recognition is one of the most interesting. Unfortunately, the shape of mushroom seems not to be often determinative at all. Mushrooms tend to be deformed or broken or they grow askew, so the sensing angle often is incorrect rather than correct. Despite that, I hope in its robustness.

The rest of the proposed features, such as gills and pores recognition, or color histograms of cap and stem, are quite distinguishable. Their success could be expected, but it is not good to rely only on these features completely. The greatest success is expected only after proper combination of all the features.

The testing will be done in real environment. It is planned to create a beta version of the application. The user will be asked to send anonymous statistical data such as sensed images and classification results. The evaluation is going to reveal, whether it will be better to train the classifier to distinguish among individual mushroom species or only among their families, or what features are really valuable and properly chosen.

There are some unmistakable species such as Amanita Muscaria or Boletea family. The expected success of these species is almost 100%. What is really important, is to recognize correctly the lesser frequently occurring mushrooms. The expected success is to have the correct mushroom among the top ten returned mushrooms. Otherwise, it could be quite dangerous, especially in the case when the sensed mushroom is poisonous and in the returned list are only edible mushrooms.

5 Acknowledgments

I would like to thank especially to the server *Nahuby.sk* [1] and to all people who made it real in such a quality. They provided me thousands of high quality and well marked images of mushrooms. I would like to thank to my supervisor, Pavel Zemcik, for his very valuable support and many corrections of this contribution.

6 Conclusions and Future work

In this contribution, the approach of support of mushroom machine recognition using mobile Android device was proposed. The previous contributions and projects of similar topic have been used as well as the mycological knowledge in part of choosing proper features.

It was described that the mobile devices are quite powerful but they need another approach than desktop computers

and the high level of optimization as well. The optimization is the first part of what can be done better.

Another step how to improve results - and possibly quite radical one - is to choose other features or choose more of them. However, this could have a significant influence of power and time requirements of such a recognition. For now, it seems wise to choose computationally easier methods and focus properly on the optimization process. But the computing power of such devices is still increasing.

There are many of features that could refine the classification. It could be white blobs on the cap or ring presence on the stem (as Matti propsed). These features are not related to the whole spectrum of the mushrooms, but only to the narrow subset. As such they were not chosen to this contribution. However, the smart set of such detectors focused to specific features could be really valuable.

The application could, in the future, involve the mushroom picker to higher extent. In case when the classifier returns a list of possible candidates, the application could generate a decision tree and ask the user for a help.

Following the previous point, there is a possibility to develop recognition on sequence of questions and return the list of most probable candidates. This could be useful in case of devices without a built-in camera. This method has a nice side effect - it can spread more information about the mushroom classification to the users and teach them how to recognize them.

There are also some possible improvements that proposed Matti already. It can be, e.g. seasons or weather forecast. These informations can be easily available in such a device.

References

[1] Roland Baranovic. *Nahuby.sk*, [online 19.2.2013], A mycologic web with a lot of information and a large atlas of mushrooms with a lot of photos of every mushroom kind. Available at http://nahuby.sk/

[2] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer Science Business Media, New York, 2006.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124 – 1137, september 2004.

[4] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1, pages 105 – 112, 2001.

[5] Y. Boykov, O. Veksler, and R. Zahib. Efficient Approximate Energy Minimization via Graph Cuts. In *I*EEE TPAMI, pages 1222–1239, 2001.

[6] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

[7] A. Delong, A. Osokin, H. N. Isack, and Y. Boykov. Fast Approximate Energy Minimization with Label Costs. In *I*EEE CVPR, pages 2173–2180, 2010.

[8] Richard O Duda. *Pattern classification*. Wiley, New York, second edition, 2001.

[9] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):147 – 159, february 2004.

[10] Y. Lee, Y.; Lin and G. Wahba. Multicategory support vector machines. In *Computing Science and Statistics, 2001. Proceedings of the 33rd Symposium on the Interface*, june 2001.

[11] Damien Matti. Mushroom recognition. Master semester project, Ecole polytechnique federale de Lausanne, 2010.

[12] K. Mikolajczyk, B. Leibe, and B. Schiele. Multiple object class detection with a generative model. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, pages 26 – 36, june 2006.

[13] C.L. Novak and S.A. Shafer. Anatomy of a color histogram. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 599 – 605, june 1992.

[14] Konstantinos Koutroumbas and Sergios Theodoridis. *Pattern Recognition*. Academic Press, Amsterdam, second edition, 2003.

[15] Andrew N Sloss, Dominic Symes, and Chris Wright. *ARM system developers guide: designing and optimizing system software*. Elsevier, Amsterdam, 2004.

[16] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in computer science. Springer, London, 2010.