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

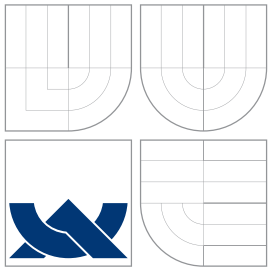
DETEKCE OBJEKTŮ V OBRAZE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

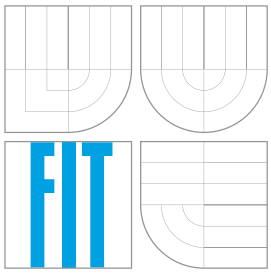
AUTOR PRÁCE
AUTHOR

TOMÁŠ PTÁČEK

BRNO 2008



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

DETEKCE OBJEKTŮ V OBRAZE

OBJECT DETECTION IN IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ PTÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MIROSLAV ŠVUB

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Ptáček Tomáš**
Obor: Informační technologie
Téma: **Detekce objektů v obraze**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte základy zpracování obrazu, seznamte se s možnostmi detekce objektů v obraze, případně jejich rozpoznávání.
2. Vyberte vhodné metody a navrhněte detektor objektů zvoleného typu.
3. Připravte trénovací data.
4. Experimentujte s vaší implementací. Případně navrhněte vlastní modifikace metod.
5. Diskutujte dosažené výsledky a možnosti budoucího vývoje. Zvažte další pokračování v rámci diplomové práce.
6. Vytvořte stručný plakát prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění prvních tří bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Švub Miroslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

Licenční smlouva je uvedena v archivním výtisku uloženém v knihovně FIT VUT v Brně.

Abstrakt

Tato práce pojednává o problematice detekce objektů a popisuje teoretická východiska detekce založené na boostingu, algoritmu AdaBoost a Haarových příznacích v roli slabých klasifikátorů. Dále se tato práce zabývá návrhem a implementací trénovací a detekční aplikace založené na knihovnách OpenCV a wxWidgets. K závěru popisuje test trénování a detekce obličejů provedený v implementované aplikaci.

Klíčová slova

Detekce objektů, detekce obličejů, zpracování obrazu, boosting, AdaBoost, Haarovy příznaky, OpenCV, wxWidgets.

Abstract

This work deals with the problem of object detection in images and describes theoretical backgrounds of detection based on boosting, AdaBoost algorithm and Haar-like features as weak classifiers. Further this work engages in design and implementation of a training and detection application based on OpenCV and wxWidgets libraries. To the end it shows a training and face detection test performed in the implemented application.

Keywords

Object detection, face detection, image processing, boosting, AdaBoost, Haar-like features, OpenCV, wxWidgets.

Citace

Tomáš Ptáček: Detekce objektů v obraze, bakalářská práce, Brno, FIT VUT v Brně, 2008

Detekce objektů v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Miroslava Švuba. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Ptáček
11. května 2008

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce panu Ing. Miroslavu Švubovi za poskytnutou odbornou pomoc a podporu při vývoji aplikace a psaní této technické zprávy.

Obsah

1 Úvod **3**
- 1.1 Motivace . . . 3
- 1.2 Obsah práce . . . 3

2 Základní pojmy **4**
- 2.1 Obraz a jeho vlastnosti . . . 4
- 2.2 Zpracování obrazu . . . 4
- 2.3 Klasifikace . . . 5
- 2.4 Boosting . . . 5
- 2.5 Slabý klasifikátor . . . 6
- 2.6 Haarovy příznaky . . . 6

3 Detekce objektů v obraze **7**
- 3.1 Význam a použití . . . 7
- 3.2 Třída objektů . . . 7
- 3.3 Principy detekce . . . 8
  - 3.3.1 Detekce podle barvy . . . 8
  - 3.3.2 Detekce podle pohybu . . . 8
  - 3.3.3 Detekce podle hran . . . 8
  - 3.3.4 Detekce podle vzoru . . . 9

4 Detekce obličejů a metoda AdaBoost **10**
- 4.1 Princip algoritmu AdaBoost . . . 10
- 4.2 Kaskáda klasifikátorů . . . 11
- 4.3 Falešné odezvy . . . 11
- 4.4 Postup detekce . . . 11

5 Návrh aplikace **13**
- 5.1 Trénování . . . 13
- 5.2 Nastavení trénování . . . 14
- 5.3 Detekce . . . 14
- 5.4 Testování úspěšnosti . . . 14
- 5.5 Vizualizace klasifikátoru . . . 15

6 Implementace aplikace **16**
- 6.1 Použité nástroje . . . 16
  - 6.1.1 OpenCV . . . 16
  - 6.1.2 Haartraining . . . 16

6.1.3 wxWidgets . . . 16
6.2 Popis vlastní implementace . . . 17
6.2.1 Trénovací část . . . 17
6.2.2 Detekční část . . . 18
6.2.3 Část testování úspěšnosti . . . 19
6.2.4 Část vizualizace klasifikátoru . . . 19

7 Testování **20**
7.1 Podmínky trénování . . . 20
7.2 Průběh trénování . . . 21
7.3 Výsledné detektory . . . 21
7.4 Ukázka detekce . . . 22
7.5 Testovací sady . . . 22
7.6 Úspěšnost detektorů . . . 22
7.7 Rychlost v detekci . . . 23

8 Závěr **24**

A Návod k použití aplikace **28**

B Snímky aplikace **31**

Kapitola 1

Úvod

1.1 Motivace

Detekce objektů v obraze je v současnosti aktuální a rozvíjející se vědní obor. Své uplatnění nalézá všude, kde po počítači nebo jiném výpočetním zařízení požadujeme schopnost „vidět“ objekty určitého druhu. Díky rostoucímu výkonu počítačů a cenové dostupnosti hardware se tento obor prosazuje v různých oblastech informačních technologií, mimo jiné v biometrii a kybernetice, a zastává nenahraditelnou roli v odlišných oblastech výzkumu a průmyslu.

1.2 Obsah práce

Cílem mé bakalářské práce bylo navrhnout a implementovat detektor určitého typu a testovat jeho vlastnosti. První část této technické zprávy je zaměřena na problematiku detekce objektů v obraze a její teoretická východiska. Druhá část se zabývá samotnou detekční aplikací. Zde je shrnutí toho, o čem pojednávají jednotlivé kapitoly:

1. kapitola - stránka, kterou právě čtete
2. kapitola - vymezení pojmů a teorie nutná k pochopení dalšího textu
3. kapitola - seznámení s různými přístupy k detekci objektů v obraze
4. kapitola - popis metody zvolené pro implementaci aplikace
5. kapitola - popis aplikace z hlediska jejího návrhu
6. kapitola - popis důležitých částí implementace aplikace
7. kapitola - představení provedeného testování a naměřených hodnot
8. kapitola - závěr a nástin možného budoucího vývoje aplikace

Kapitola 2

Základní pojmy

V této kapitole vymezím základní pojmy, na něž bude později v textu odkazováno. Jde o pojmy týkající se zpracování obrazu, klasifikace a boostingu. Pojmy na sebe navazují v pořadí, v jakém jsou uvedeny.

2.1 Obraz a jeho vlastnosti

V celé této práci je obrazem míněna *digitální rastrová grafika* získaná fotoaparátem, videokamerou nebo jiným snímacím zařízením. V oblasti detekce objektů je pojem obraz zcela běžně užíván se stejným významem a prakticky nikdy není myšlen jinak, takže nemůže dojít k nedorozumění.

Obraz v našem pojetí je grafická informace v podobě rastrové matice o určitém počtu řádků a sloupců. Nejmenší stavební jednotkou této matice je tzv. *pixel*. Každý pixel je nositelem barevné informace. Obraz lze vnímat také jako dvourozměrný diskrétní signál.

Pro téma detekce objektů je důležité uvědomit si, že rastrový obraz v sobě nenese žádnou informaci o vzájemném vztahu svých hodnot, pixelů. Není tedy možné přímou cestou z obrazu získat např. jednotlivé objekty na něm vyobrazené, jejich tvar ani jejich počet.

2.2 Zpracování obrazu

Jde o aplikaci algoritmů na obrazová data. Většinou tyto algoritmy pracují s jasovou či barevnou informací, nebo data filtrují. Mnoho aplikací uvažuje vstupní obraz jako dvourozměrný signál a tak s ním při zpracování i zachází. Výstupem může být nový obraz nebo zjištěná informace.

Obraz se často zpracovává za účelem snadnějšího získání informací z něj. To má zásadní význam v detekci objektů, kde algoritmy ze snímků potřebují extrahovat jen určité většinou jednoduché informace (např. hrany, oblasti, rozdíly světlosti) a obtížně by nakládaly s původním obrazem obsahujícím obrovské množství barevných informací.

Výhodami zpracování digitálního obrazu jsou nízké náklady a oproti analogovému zpracování nevzniká žádný šum.

2.3 Klasifikace

Statistická klasifikace je proces, při kterém se rozhoduje o příslušnosti položek k určeným třídám na základě kvantitativní znalosti jejich charakteristických vlastností [6]. Tato znalost vzniká trénováním na *trénovací množině* příkladů s předem známou příslušností. Jde tedy o formu učení s učitelem (anglicky *supervised learning*).

Totéž nyní vysvětlím formálně. Mějme například klasifikaci o dvou třídách:

$$Y = \{-1, +1\} \tag{2.1}$$

Taková klasifikace se nazývá *dvou-třídní* nebo *binární* klasifikace a může značit např. zda o položce platí (třída $+1$) či neplatí nějaké tvrzení (třída -1). Prvek této klasifikační množiny budeme značit y_i, nabývat může hodnot -1 a $+1$. Dále mějme množinu položek X a její prvky $\mathbf{x}_i$. Trénovací množina definovaná následující rovnicí je pak množinou dvojic položek a jejich příslušností k třídám.

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\mathrm{n}, y_n)\}, \mathbf{x} \in X, y \in Y \tag{2.2}$$

Výstupem trénování na této trénovací množině pak bude tzv. *klasifikátor*

$$h : X \to Y, \tag{2.3}$$

který promítá položku $\mathbf{x}$ do příslušné klasifikační třídy y. Jinými slovy, klasifikátor je hledaná funkce, která by pro dané vstupní $\mathbf{x}_i$ předepisovala správné výstupní y_i. Nalezení této funkce, nebo-li nalezení přesného klasifikátoru, je hlavní problém.

2.4 Boosting

Boosting je metoda strojového učení s učitelem. Myšlenkou boostingu je vytvořit velmi přesný klasifikátor (nazýván jako *silný*) kombinací relativně jednoduchých klasifikátorů (nazývaných jako *slabé*).

Boosting ke své činnosti vyžaduje popsanou trénovací množinu, tedy s předem známou příslušností jednotlivých příkladů do tříd. Na tuto množinu je aplikován algoritmus, jehož výstupem je natrénovaný silný klasifikátor.

Existuje několik boostingových algoritmů. Většina z nich pracuje v iteracích. V každé iteraci je hledán nový slabý klasifikátor, který by minimalizoval chybu na trénovací množině. Když je takový klasifikátor nalezen, je mu na základě jeho chyby přiřazena určitá váha. Pak je klasifikátor přidán do skupiny budoucího silného klasifikátoru, obvykle provedením operace lineární kombinace. Na konci iterace jsou ještě spočteny nové váhy jednotlivých prvků trénovací množiny.

Váhy prvků trénovací množiny mají podstatný vliv na to, jakým způsobem jsou vybírány slabé klasifikátory. Boostingové algoritmy tyto váhy utváří postupně v průběhu celého trénování a reflektují v nich úspěšnost klasifikace každého prvku trénovací množiny. Čím menší je tato úspěšnost, tím větší získává prvek váhu a naopak. To umožňuje algoritmu zaměřit se v trénování na prvky dříve chybně klasifikované a nalézt v nich nové určující vlastnosti zlepšující jejich klasifikaci.

2.5 Slabý klasifikátor

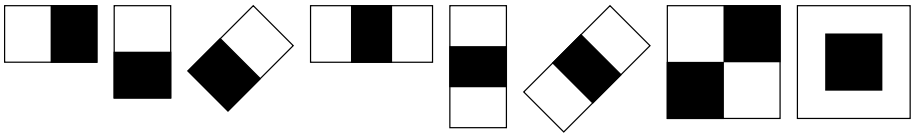
Pojem slabý klasifikátor se užívá ve spojení s klasifikací a boostingem. Jde o klasifikátor, který jen velmi vzdáleně odpovídá správné klasifikaci, ale zato je obvykle relativně jednoduchý a rychle spočitatelný.

K detekci objektů se používají různé druhy slabých klasifikátorů citlivých na určité informace extrahované z obrazu (např. světlost, barva). V této práci bude řeč pouze o tzv. *Haarových příznacích*, které reagují na vzájemný vztah sousedních pixelů.

2.6 Haarovy příznaky

Haarovy příznaky (anglicky *Haar-like features*) jsou velmi rychlou alternativou k dříve používané klasifikaci pracující se světlostí obrazu. Nevýhoda tohoto přístupu spočívala v jeho časové složitosti.

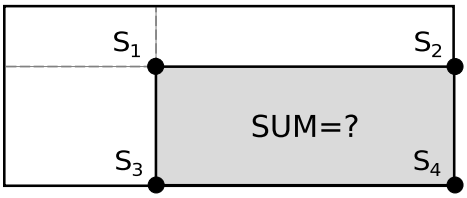
Haarovy příznaky pracují s obdélníkovými oblastmi původního obrazu. V nich jsou počítány sumy pixelů a výsledek příznaku je dán jako rozdíl těchto sum. Na obrázku 2.1 jsou znázorněny některé Haarovy příznaky. Nejjednodušší varianta se skládá ze dvou osově souměrných obdélníků stejných velikostí. Složitější se pak skládají ze tří nebo čtyř obdélníků, případně jsou navíc otočeny o 45°.



Obrázek 2.1: Ukázka Haarových příznaků včetně některých diagonálních variant.

Vysoká rychlost Haarova příznaku souvisí s výpočtem sum pixelů a s pojmem *integrální obraz*, který byl poprvé zaveden v publikaci autorů Viola a Jones [4]. Integrální obraz lze definovat jako dvourozměrnou matici o rozměrech původního obrazu. Každý prvek této matice obsahuje sumu všech pixelů původního obrazu v obdélníku umístěném nad a vlevo od tohoto prvku. Integrální obraz takto umožňuje v konstantním čase spočítat sumu pixelů kterékoliv obdélníkové části obrazu pouhým vyhledáním 4 hodnot a jejich sečtením/odečtením. Například suma pixelů ve zvýrazněném obdélníku z obrázku 2.2 je vypočtena následovně:

$$SUM = S_1 - S_2 - S_3 + S_4 \tag{2.4}$$



Obrázek 2.2: Výpočet sumy pixelů vnitřního obdélníku integrálním obrazem. Body S_1 až S_4 představují hodnoty v integrálním obraze.

Kapitola 3

Detekce objektů v obraze

Nyní se zaměřím na popis detekce objektů v obraze, jejích principů a existujících metod.

3.1 Význam a použití

Detekce objektů v obraze je počítačová technologie vyhledávání objektů určité třídy v obraze. Cílem je zjistit pozici a velikost každého nalezeného objektu.

Detekce objektů se užívá hlavně ve spojení s videokamerou nebo jiným snímacím zařízením. Takto vybavený počítač je schopen ve snímaném obraze automaticky lokalizovat daný objekt a na jeho výskyt nebo vlastnost dále reagovat. Často se používají detektory lidských obličejů v identifikačních systémech, kde detektor slouží jako předstupeň rozpoznávání obličejů. Dále jsou detektory objektů uplatňovány v oblasti automatizace výrobních procesů, kde mohou sloužit jako zařízení kontroly výstupní kvality. Jinou aplikací jsou inteligentní roboti s videokamerami schopní interakce se svým okolím nebo například inteligentní videokamery sledující pohyb snímané osoby. V mnohých těchto aplikacích probíhá detekce v reálném čase, což umožňuje stále rostoucí výkon výpočetní techniky a její vysoká dostupnost.

Ačkoliv detekce objektů poskytuje počítači schopnost „vidět“ určité objekty, zdaleka nejde o napodobení lidského vidění, které je ve své podstatě mnohem složitější a komplexnější. Hlouběji se tímto zabývá obor počítačové vidění, kde detekce objektů figuruje jako jeden z mnoha otevřených problémů.

3.2 Třída objektů

Žádný univerzální postup jak detekovat libovolné objekty není. Proto se soustřeďujeme vždy na konkrétní třídu objektů, které chceme detekovat.

Objektem dané třídy může být např. automobil nebo lidský obličej, obecně jakýkoliv objekt, který lze při snímání opticky rozlišit od okolí a který je charakteristický svým vzhledem. Stejně jako většina automobilů je typických svým tvarem, mají i obličeje své charakteristické znaky, jež sdílí a díky nimž je lze zároveň odlišit od objektů, které obličeji vůbec nejsou. Povaha těchto společných znaků má také vliv na výběr vhodného typu detektoru.

3.3 Principy detekce

Hlavním problémem je, jak objekty v obraze detekovat. Obraz je totiž pouhou maticí číselných hodnot bez jakékoliv přímé informace o objektech v něm.

K detekci využíváme různých principů zpracování obrazu a znalostí o lidském vidění objektů.

3.3.1 Detekce podle barvy

Pokud je objekt význačný svou barvou a za předpokladu, že máme k dispozici barevné snímací zařízení, lze použít tuto jednoduchou metodu detekce. Její princip se zakládá na zpracování obrazu filtrací barevné informace s vhodnou tolerancí odchylky barvy a světlosti. Jakákoliv obrazová data mimo určený rozsah barev a světlosti jsou z obrazu odstraněna a hledaný objekt zůstává.

Problém samozřejmě nastává v momentě, kdy se na snímcích objeví jiný objekt stejné barvy. V případě, že něco takového běžně nenastane, lze detektor použít bez potíží. V ostatních případech je možné tuto metodu spojit s nějakou jinou metodou, která dokáže v množině zjištěných objektů eliminovat nežádoucí případy.

Detekce podle barvy objektu může být využita například v kombinaci s detektory obličejů, kde předem známe barvu lidské pokožky. V praxi se ale tato kombinace příliš nepoužívá. Jednak proto, že nefunguje se všemi typy pokožky, a také kvůli zkreslení barev vlivem různých světelných podmínek.

3.3.2 Detekce podle pohybu

V případě aplikace s videosekvencí nebo videokamerou a obrazem snímaným v reálném čase lze použít detekci objektů podle pohybu. Jeden z možných principů této metody je založen na porovnávání dvou posledních zachycených snímků a vyhodnocování změn hodnot jednotlivých pixelů [2]. Objekty jsou nalezeny vyhledáním celistvých oblastí, ve kterých došlo ke změně dostatečně velkého množství pixelů.

Podobně jako u metody detekce podle barvy zde nastává problém, když se v obraze může pohybovat více objektů. Řešením je opět použití tohoto detektoru v kombinaci s další detekční metodou.

Ve snímaném obraze je potřeba zohlednit i šum, který zde může vzniknout a který by mohl být detektorem mylně vyhodnocen jako pohybující se objekt. K řešení je možné použít jednoduchý filtr šumu, který z vyhodnocených změn pixelů odstraní malé osamocené oblasti.

3.3.3 Detekce podle hran

Tato metoda je založena na zpracování obrazových dat pomocí segmentace. Výstupem segmentace jsou nalezené významné oblasti v obraze zjištěné pomocí detektoru hran nebo např. zkoumáním oblastí s přibližně stejnými vlastnostmi pixelů. Zjištěné oblasti jsou obvykle dále zkoumány jiným algoritmem. Ten se uplatní v případě členitého obraz, kde je potřeba rozhodnout, která oblast obsahuje hledaný objekt.

Existuje mnoho metod segmentace, detekce hran je jednou z nich. Společným cílem těchto metod je zjednodušit další zpracování obrazu. Když je obraz rozdělen do několika celků, je následná analýza snazší. Lze se tak namísto celého obrazu věnovat konkrétním vzniklým segmentům a ty pak třeba dále klasifikovat.

3.3.4 Detekce podle vzoru

V oblasti počítačového vidění je detekce objektů podle vzoru tou nejvýše postavenou a zároveň nejsložitější metodou. Využívá se zde principů strojového učení na množině trénovacích dat.

Detektor musí být nejdříve na trénovacích datech nacvičen, v čemž se tato metoda liší od ostatních. Obvykle se detektor trénuje na sadě obrazových dat rozdělené do dvou tříd s předem známým významem: třída snímků daného objektu a třída snímků bez výskytu tohoto objektu.

Obecně slouží trénovací data v procesu trénování k vyhledání informací, většinou založených na statistickém přístupu. V oblasti detekce objektů jde o vyhledávání informací popisujících vzory v trénovacích datech. Tyto vzory jsou z obrazu získávány pomocí matematicky spočitatelných příznaků. Pokud jsou informace o vzorech nalezeny, je na jejich základě vytvořen výsledný klasifikátor použitelný k detekci objektů. Běžně jde o binární klasifikátor rozlišující pouze dvě třídy: nacvičený objekt a něco jiného než nacvičený objekt.

Mezi nejznámější přístupy k této metodě detekce patří v současné době neuronové sítě a boosting. Charakteristické jsou především svou robustností, širokým uplatněním a vysokou úspěšností. Díky těmto i dalším přednostem si metody detekce objektů podle vzoru získaly ve světe velkou oblibu. Jejich vývoj je stále předmětem výzkumu.

Poznámka: Zmíněnou metodou boostingu se budu podrobněji zabývat v následující kapitole *na straně 10.*

Kapitola 4

Detekce obličejů a metoda AdaBoost

Pro svou práci jsem jako třídu detekovaných objektů zvolil obličeje. Důvodů je několik. Ze všech tříd objektů je detekce obličejů pravděpodobně nejrozšířenější, což znamená také existenci většího množství literatury na toto téma. Dalším důvodem jsou volně dostupné, kvalitní a rozsáhlé databáze trénovacích dat s předzpracovanými snímky obličejů. Díky existenci těchto databází odpadá potřeba vytvářet vlastní trénovací data, což je s ohledem na potřebné množství a variabilitu vzorků časově náročný proces. Jiným důvodem je samotná zajímavost obličejů jako třídy objektů daná specifickými vlastnostmi, které se u jiných tříd obtížně hledají.

Obrázek 4.1: Ukázka trénovací množiny se snímky obličejů (zdroj: Yale Face Database)

Jako metodu jsem zvolil detekci podle vzoru založenou na algoritmu AdaBoost a na detekčním systému Viola & Jones [4]. Tento systém má několik klíčových vlastností: klasifikaci provádí přes relativně rychlé Haarovy příznaky, k jejich výběru a trénování využívá silný algoritmus AdaBoost a místo jediného výsledného klasifikátoru vytváří *kaskádu klasifikátorů*. Tyto vlastnosti zajistily detekčnímu systému Viola & Jones znatelně vyšší rychlost detekce a umožnily jeho nasazení v real-time aplikacích, což bylo u podobných systémů dříve nemyslitelné.

4.1 Princip algoritmu AdaBoost

AdaBoost, aneb adaptivní boosting, je algoritmus strojového učení. Jeho vstupem je popsaná trénovací množina a výstupem silný klasifikátor sestavený ze slabých klasifikátorů nalezených v průběhu trénování. Adaptivnost tohoto algoritmu spočívá v jeho schopnosti přizpůsobit trénování nových klasifikátorů dříve chybně klasifikovaným případům.

Mějme trénovací množinu $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in X$, $y_i \in Y = \{-1, +1\}$. Nejdříve je inicializována distribuční funkce $D_1(i)$ váhující příklady v trénovací množině. Pak pro iterace $t = 1, \ldots, T$ je prováděna následující posloupnost operací [3, 5]:

1. Najdi slabý klasifikátor h_t minimalizující chybu s ohledem k distribuční funkci $D_t(i)$
2. Spočítej chybu tohoto klasifikátoru ϵ_t jako pravděpodobnost chybné klasifikace
3. Podle chyby ϵ_t urči váhu α_t tohoto klasifikátoru
4. Spočítej novou distribuční funkci D_{t+1} pro příští iteraci

Nakonec je vytvořena lineární kombinace slabých klasifikátorů $h_t(\mathbf{x})$ a jejich vah α_t. Vznikne suma:

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{4.1}$$

Výsledný klasifikátor je pak dán znaménkem této sumy:

$$H(\mathbf{x}) = sign\left(f(\mathbf{x})\right) \tag{4.2}$$

4.2 Kaskáda klasifikátorů

Algoritmus Viola & Jones je charakteristický použitím kaskády klasifikátorů. Není zde pouze jeden silný klasifikátor, ale celá jejich řada (běžně 10 až 30).

Důvodem k použití kaskády klasifikátorů je dosažení vysoké rychlosti detekce objektů. Obraz je totiž v naprosté většině případů tvořen velkým množstvím pozic, kde se nenachází žádný hledaný objekt. Detektor obličejů může mít až tisíce rozhodovacích elementů. Kdyby všechny tyto elementy musely být spočteny v každé pozici v obraze, zpracování celého obrazu by bylo náročnější.

Kaskáda klasifikátorů umožňuje velmi rychle přeskočit nežádoucí pozice v obraze a soustředit se tak na oblasti s potenciálním výskytem hledaného objektu. V prvních stádiích kaskády bývá obvykle jen několik málo klasifikátorů a s dalšími stádii jejich počet stoupá. První stádia jsou tedy vyhodnocena rychleji, díky čemuž lze aktuální pozici dříve označit za nežádoucí a přeskočit ji bez vyhodnocení zbývajících stádií, což je klíčová vlastnost.

4.3 Falešné odezvy

Jedním z důležitých cílů při trénování je dosažení co nejnižšího podílu falešně pozitivních a také falešně negativních odezev. Pokud uvážíme, že v celém obraze je například jeden hledaný obličej, znamená to pro detektor velké množství pozic bez obličeje, kde by tudíž neměl být žádný obličej falešně detekován. Z tohoto pohledu má velký význam nízký podíl falešně pozitivních odezev. Pro reálné použití detektoru jde o důležitý parametr, který se v konečném důsledku může projevit i v jeho vyšší rychlosti.

Nízkého podílu falešně pozitivních odezev se dosahuje volbou kvalitní a rozsáhlé trénovací množiny a dostatečně dlouhým trénováním. Pro kvalitní detektor by mělo být trénování ukončeno až po dosažení podílu falešně pozitivních odezev alespoň 0,000005 [1]. Takový detektor v milionu pozicích bez obličeje chybně detekuje jen 5 pozic.

4.4 Postup detekce

Objekty jsou v daném obraze detekovány postupným procházením obrazu pomyslným oknem (viz obrázek 4.2), přičemž obsah tohoto okna je v každém kroku vyhodnocen klasifikátorem. Pokud klasifikátor v nějakém kroku pro dané okno vyhodnotí kladnou odezvu,

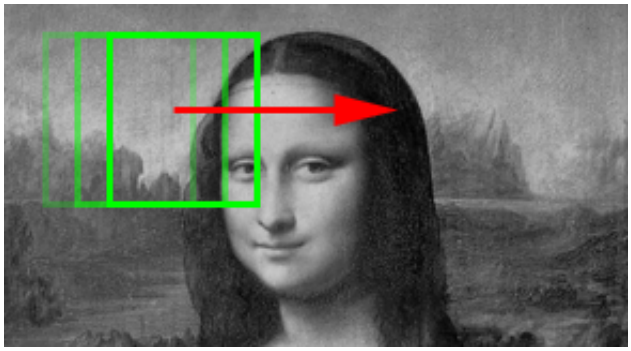
znamená to, že toto okno představuje hledaný objekt. V takovém případě je pozice a velikost okna zaznamenána nebo i vyznačena přímo v obraze.

Protože však neznáme velikost hledaného objektu a chceme, aby detektor byl k velikosti objektu invariantní, musí být procházení obrazu provedeno několikrát v různých měřítkách. Nemění se však velikost okna, kterým obraz procházíme, ale transformuje se velikost celého obrazu. Obraz je po každém průchodu podvzorkován (viz obrázek 4.3) s daným poměrem nové a předchozí velikosti a když už je tak malý, že jej nelze naším oknem procházet, detekce skončí.

Poměr podvzorkování má vliv na úspěšnost detekce. Pokud je jeho hodnota nastavena příliš nízko, je obraz zmenšován po velmi malých krocích. To sice zvyšuje pravděpodobnost správné detekce všech objektů v obraze, ale detekce je časově náročnější, protože je nutné průchod obrazem provést vícekrát. Nastavení příliš vysoké hodnoty zase vede k rychlé detekci, ale detektor častěji nějaký objekt vůbec nezaznamená, protože jej zkrátka „přehlédne“.

Při implementaci detektoru je snaha minimalizovat počet kroků nutných k průchodu obrazem. K urychlení detekce se využívá např. poznatku, že v místě, kde již byl detekován objekt, nebude přítomen další objekt stejného druhu. Takovou oblast tedy lze bez obav přeskočit.

Jiným přístupem k urychlení detekce je využití poznatku, že pokud detekční okno právě obsahuje příliš členitou strukturu (např. větvě stromů), určitě nepůjde o hledaný objekt. Takové oblasti v obraze jsou místem s vysokým potenciálem falešně pozitivních odezev, kde může být klasifikace pomalá. Zde popisovaný přístup spolupracuje s vhodným detektorem hran, který dokáže zmíněné oblasti rychle rozpoznat a umožní jim při detekci předejít.



Obrázek 4.2: Průchod obrazu pomyslným oknem při detekci.

Obrázek 4.3: Podvzorkování v poměru 0,5. Vlevo: poměr velikostí. Vpravo: vliv na obraz.

Kapitola 5

Návrh aplikace

V předchozích kapitolách jsem probral teoretický základ problematiky detekce objektů v obraze. Také jsem uvedl a zdůvodnil své rozhodnutí věnovat se detekci obličejů s použitím metody AdaBoost. V této a dalších kapitolách se již budu věnovat praktické části své práce, formulaci cíle, jeho realizaci a testování.

Cílem mé bakalářské práce bylo navrhnout a implementovat detektor objektů určitého typu a testovat jeho vlastnosti. Jelikož existují kvalitní a volně dostupné implementace různých detekčních algoritmů, bylo již od začátku počítáno s použitím jedné z nich. Mým úkolem tudíž bylo soustředit se na návrh a provedení aplikace, která by nad použitou knihovnou vytvářela intuitivní grafické rozhraní poskytující funkce spojené s příslušnou metodou detekce objektů.

Celá aplikace byla v návrhu rozdělena do několika samostatných částí. Hlavní jsou části „Trénování“ a „Detekce“. Mimo tyto byly v aplikaci navrhnuty ještě části „Nastavení trénování“, „Testování úspěšnosti“ a „Vizualizace klasifikátoru“. Vyjmenované části zde nyní popíši.

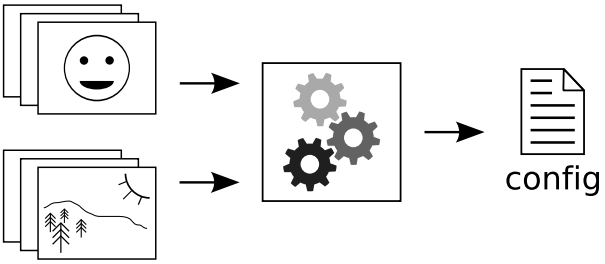
5.1 Trénování

Účelem této části je umožnit uživateli vytvořit vlastní klasifikátor. Trénování vyžaduje množinu trénovacích dat s předem známým přiřazením prvků do tříd. Předpokládá se existence dvou adresářů s obrazovými soubory, jeden s pozitivními příklady - snímky daného objektu - a druhý s negativními příklady - snímky bez výskytu daného objektu. Aplikace nabídne uživateli možnost volby těchto adresářů.

Pomocí příslušného tlačítka dá uživatel pokyn k přípravě na trénování. Aplikace provede kontrolu a načte obrazové soubory ze zvolených adresářů. O případných problémech s načítáním snímků je uživatel informován prostřednictvím výpisu. Jakmile tato příprava skončí, informuje aplikace uživatele kolik bylo úspěšně načteno snímků.

Dále aplikace nabídne uživateli volbu cíle, kam bude po dokončení trénování uložen výsledný klasifikátor v podobě konfiguračního souboru. Trénování pak uživatel spustí tlačítkem. V průběhu trénování budou zobrazovány podrobné informace, např. aktuální chyba klasifikátoru.

Jelikož je trénování dlouhodobý proces, který může trvat i na výkonném stroji několik dní, předpokládá se, že do cílového adresáře budou průběžně ukládány hotové části trénovaného klasifikátoru. Díky tomu bude možné po přerušení později v trénování pokračovat. Aplikace tedy nabídne uživateli tuto možnost, pokud nalezne data z předešlého trénování.



Obrázek 5.1: Blokové schema trénovací části aplikace.

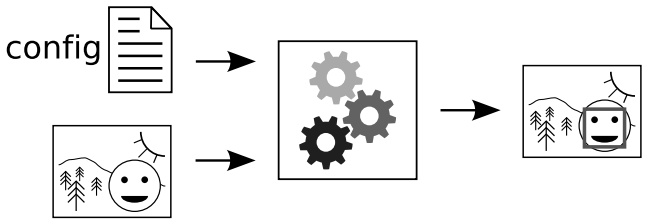
5.2 Nastavení trénování

Trénování klasifikátoru lze ovlivnit několika parametry, jako počet stádií, cílový podíl falešně pozitivních odezev, rozměry vzorku a dalšími. Aplikace poskytne uživateli přehled o těchto parametrech a umožní mu tyto před trénováním upravit dle potřeby. Aplikace musí kontrolovat správnost zadaných parametrů. V případě chybného parametru upozorní uživatele a znemožní mu spuštění trénování, dokud nebude parametr zadán správně.

5.3 Detekce

Detekční část aplikace slouží k provedení detekce objektů v konkrétním obraze. Použije se zde již hotový natrénovaný klasifikátor. Aplikace nabídne uživateli volbu soubor s konfigurací klasifikátoru a volbu vstupního obrazového souboru. Prostřednictvím zvoleného klasifikátoru jsou následně ve zvoleném obraze detekovány všechny výskyty objektů té třídy, na které byl klasifikátor nacvičen. Obraz je pak uživateli zobrazen s vyznačením všech zjištěných pozic a jejich velikostí. Aplikace pro uživatelův přehled uvede celkový počet objektů detekovaných v obraze.

Cílem této části aplikace je umožnit uživateli otestovat natrénovaný klasifikátor na konkrétních snímcích a ověřit tak jeho správné chování. K testování úspěšnosti detekce nad více snímky je vhodné použít část „Testování úspěšnosti“.



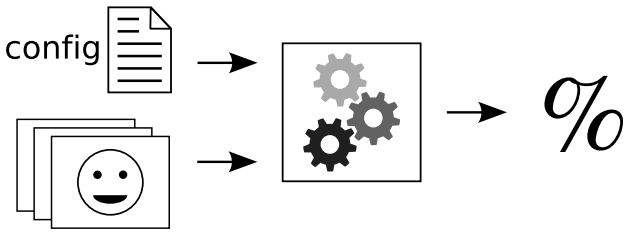
Obrázek 5.2: Blokové schema detekční části aplikace.

5.4 Testování úspěšnosti

Tato část slouží k změření úspěšnosti daného klasifikátoru. Měření se provádí na množině testovacích pozitivních příkladů. Předpokládá se existence adresáře s obrazovými soubory,

na kterých má být testování provedeno. Uživatel v aplikaci zvolí tento adresář a soubor s konfigurací klasifikátoru, který má být testován. Kliknutím na tlačítko je testování spuštěno a po jeho skončení je uživateli zobrazena informace o naměřené úspěšnosti.

Zpracování většího množství obrazových dat může trvat delší dobu. Proto bude v průběhu testování zobrazena informace o postupu, aby uživatel měl přehled o počtu již zpracovaných snímků a mohl odhadnout dobu zbývající do konce testování.

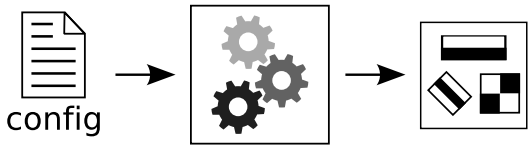


Obrázek 5.3: Blokové schema testování úspěšnosti.

5.5 Vizualizace klasifikátoru

Tato funkce aplikace je určena spíše k výukovým nebo demonstračním účelům. Slouží k znázornění toho, co je výsledkem celého procesu trénování klasifikátoru. Zobrazí totiž vnitřní stavbu kaskády klasifikátoru a znázorní rozložení jednotlivých příznaků, tak jak budou použity při detekci objektů.

Aplikace si od uživatele vyžádá volbu souboru s konfigurací klasifikátoru. Po jeho načtení nabídne seznam jednotlivých stádií kaskády, v každém stádiu jednotlivé klasifikátory a pro každý z nich příslušné příznaky. Uživatel má možnost mezi jednotlivými stádii, klasifikátory a příznaky libovolně přepínat, přičemž mu vždy bude zobrazena vizualizace momentálně zvoleného příznaku a další údaje jako levá a pravá hodnota prvku rozhodovacího stromu, prahová hodnota a váhy jednotlivých částí příznaku.



Obrázek 5.4: Blokové schema vizualizace klasifikátoru.

Kapitola 6

Implementace aplikace

Navrženou aplikaci jsem implementoval v programovacím jazyce C++ s použitím knihovny OpenCV pro detekci objektů v obraze a s použitím GUI knihovny wxWidgets pro realizaci uživatelského rozhraní. Pro implementaci části „Trénování“ jsem použil zdrojový kód programu haartraining, který je dodáván s knihovnou OpenCV. Všechny části, jichž nejsem autorem, byly použity v souladu s licenčními ujednáními těchto částí.

Popsané řešení jsem zvolil kvůli mé znalosti jazyka C++ a kvůli předešlé kladné zkušenosti s knihovnou wxWidgets. Knihovnu OpenCV jsem zvolil na základě doporučení vedoucího této práce.

Výhodami zvoleného řešení jsou přenositelnost kódu mezi různými operačními systémy a snadná použitelnost, případně i možnost dalšího rozšíření. Aplikaci lze provozovat na majoritních operačních systémech Linux, Windows a zřejmě i Mac OS X. Funkčnost byla ověřena pouze pod operačním systémem Linux, ve kterém byla aplikace vyvíjena, a částečně pod systémem Windows.

6.1 Použité nástroje

6.1.1 OpenCV

OpenCV (Open Source Computer Vision) je rozsáhlá volně použitelná knihovna zaměřená na aplikace v oboru počítačového vidění. Kromě mnoha jiných účelů, ke kterým se používá, ji lze využít k detekci objektů v obraze. Použita byla knihovna OpenCV verze 1.0.0. Aktuální verzi lze získat na URL `http://opencvlibrary.sourceforge.net/`.

6.1.2 Haartraining

Knihovna OpenCV je dodávána se zdrojovým kódem programu haartraining. Tento program slouží k natrénování klasifikátoru pomocí některé z variant algoritmu AdaBoost. Knihovna OpenCV samotná nedisponuje funkcemi pro trénování klasifikátoru. Aby vlastní aplikace mohla uživateli nabídnout tuto funkcionalitu, využívá právě programu haartraining a jeho definovaného rozhraní.

6.1.3 wxWidgets

Knihovna wxWidgets je volně použitelný multiplatformní nástroj pro tvorbu grafického uživatelského rozhraní (GUI - Graphical user interface). Poskytuje jednotné aplikační roz-

hraní (API - Application programming interface) a umožňuje tak vytvářet aplikace pro různé operační systémy, aniž by bylo potřeba pro každý systém psát odlišný zdrojový kód.

Při vývoji aplikace byla použita knihovna wxWidgets (konkrétně wxGTK) verze 2.8.7. Aplikace je však přeložitelná i s nižšími verzemi knihovny.

Knihovnu wxWidgets lze získat na URL http://www.wxwidgets.org/.

6.2 Popis vlastní implementace

V této podkapitole proberu jednotlivé části aplikace a způsob jejich implementace. Použité funkce a jiné symboly jsou v textu vyznačeny zvláštním způsobem. V jejich názvech se lze orientovat podle počátečních písmen. Více viz následující tabulka.

prefix	význam
cv	funkce knihovny OpenCV
icv	funkce programu haartraining
wx	funkce nebo třída knihovny wxWidgets
jiné	vlastní funkce nebo standardní funkce C/C++

Tabulka 6.1: Významy počátečních písmen (prefixů) v názvech symbolů.

6.2.1 Trénovací část

Tato část využívá rozhraní programu haartraining k vytvoření konfiguračních souborů trénovací množiny a ke spuštění trénování, jehož výstupem je konfigurační soubor klasifikátoru. Program haartraining vyžaduje k trénování data trénovací množiny, ale neumožňuje použít přímo adresáře s obrazovými soubory. Je potřeba vytvořit dva pomocné soubory - VEC soubor pro pozitivní příklady a BG soubor pro negativní příklady.

VEC soubor obsahuje předzpracované vzorky pozitivních příkladů. Dělí se na hlavičku s údaji o svém obsahu a tělo, které obsahuje vzorky pozitivních příkladů, např. snímky obličejů. Tyto vzorky jsou vytvořeny z původních snímků jejich převedením do šedé škály a zmenšením na požadované rozměry. Rozměry vzorků postačují velmi malé, například jen 20×20 pixelů (lze nastavit). Díky tomu je VEC soubor i při velkém množství snímků relativně malý. Pro zápis vzorků využívám dvou funkcí z rozhraní programu haartraining - funkce *icvWriteVecHeader* zapíše hlavičku a *icvWriteVecSample* zapíše konkrétní vzorek. Celý postup vytvoření VEC souboru, od získání seznamu souborů, přes jejich převod a změnu velikosti, až po zápis, obstarává vlastní funkce *MakeVecFile*. Jako argumenty tato funkce přijímá cestu k adresáři s pozitivními příklady, cestu k výslednému VEC souboru a požadované rozměry vzorků.

BG soubor obsahuje výčet souborů adresáře s negativními příklady. Oproti binárnímu VEC souboru je v textovém formátu. K vytvoření BG souboru slouží vlastní funkce *MakeBgFile*. Ta ze zadaného adresáře získá seznam souborů, ověří, zda jde o obrazové soubory, a do výstupního souboru zaznamená jejich cesty. Argumenty této funkce jsou pouze cesta k adresáři s negativními příklady a cesta k výstupnímu BG souboru.

Jakmile jsou soubory VEC a BG vytvořeny, lze přistoupit k vlastnímu trénování. Trénování je spuštěno zavoláním jediné funkce *cvCreateTreeCascadeClassifier*, která je deklarována v rozhraní programu haartraining. Tato funkce požaduje jako argumenty především

cesty k připraveným VEC a BG souborům. Dále funkce přijímá velké množství jiných argumentů ovlivňujících průběh trénování. Těmito parametry jsou zejména počet pozitivních a negativních příkladů, počet stádií trénování, rozměry vzorků ve VEC souboru, cílový poměr falešně pozitivních odezev, varianta algoritmu AdaBoost, sada Haarových příznaků a další. Všechny parametry je možné nastavit v grafickém rozhraní aplikace v části „Nastavení trénování“. Funkci jsou pak tyto parametry předány při jejím zavolání.

Funkce *cvCreateTreeCascadeClassifier* spouští proces trénování. Během něj jsou funkcí *printf* průběžně tisknuty informace na standardní výstup. Aby mohly být tyto informace místo standardního výstupu zobrazeny v okně vlastní aplikace, musel jsem mírně zasáhnout do zdrojových kódů programu haartraining. Do dvou souborů jsem přidal definici makra C++ prekompilátoru, které při kompilaci přepíše volání standardní funkce *printf* na volání mé vlastní připravené funkce *MyPrintf*. Tato funkce se stará o předávání veškerého výstupu do příslušné komponenty *wxTextCtrl*.

Prosté zavolání tak náročné funkce jako *cvCreateTreeCascadeClassifier* není v grafické aplikaci vhodné. Došlo by k přerušení zpracovávání veškerých událostí rozhraní a aplikace by dlouhou dobu nereagovala na uživatelské vstupy ani by nedocházelo k překreslování jejího okna. Tento problém jsem se rozhodl vyřešit vyhrazením nového vlákna v běžící aplikaci pouze pro potřeby trénovací funkce. Vlákno je definováno jako vlastní třída odvozená z třídy *wxThread*. Vytvořeno a spuštěno je při každém trénování. Když trénování skončí, vlákno samo zanikne a uvolní využité prostředky. Pozastavení ani přerušení běžícího vlákna jsem však neimplementoval, ačkoliv by to bylo velmi užitečné. Implementace těchto funkcí by totiž vyžadovala větší zásah do zdrojových kódů programu haartraining nebo úplně jiný přístup.

6.2.2 Detekční část

Detekční část aplikace vykonává detekci objektů v daném obraze. K činnosti je potřeba natrénovaný klasifikátor. Soubor s konfigurací klasifikátoru je nejdříve načten funkcí *LoadClassifier*, která jen zapouzdřuje funkci *cvLoad*. Dále musí být načten samotný obraz, což je provedeno funkcí *cvLoadImage*. Detekce objektů je pak vyvolána vlastní funkcí *DetectObjects*, která při úspěchu vrátí souřadnice všech nalezených objektů. Do obrazu tyto souřadnice vyznačí v podobě obdélníků funkce *DrawRectangles*. Výsledný obraz je nakonec zobrazen uživateli v okně odvozeném od třídy *wxScrolledWindow*.

V této části aplikace je zásadní funkce *DetectObjects*. V původním návrhu aplikace se počítalo s vlastní implementací detekce objektů postupným průchodem obrazu, ale zvolená knihovna OpenCV disponuje jednoduše použitelnou funkcí *cvHaarDetectObjects*, která tento úkon již implementuje. Funkce *DetectObjects* tedy jen vytváří podmínky pro použití knihovní funkce (připraví požadované proměnné, struktury, alokuje a uvolní paměť), volá tuto funkci s přednastavenými parametry a výstup, tedy seznam souřadnic, předává do snadno zpracovatelného C++ typu *vector*.

Přednastavenými parametry funkce *cvHaarDetectObjects* jsou poměr podvzorkování obrazu, minimální počet sousedů detekovaných oblastí pro vyřazení několikanásobné detekce stejného objektu v různých měřítkách a zapnutí či vypnutí detektoru hran, který předchází oblastem v obraze s příliš členitou strukturou.

Vlastní funkce *DetectObjects* ve svém těle vstupní obraz převzorkuje na poloviční velikost. Dle OpenCV manuálu [7] je díky tomu možné zvýšit rychlost detekce.

Jako argumenty funkce *DetectObjects* přijímá ukazatel na načtený klasifikátor, ukazatel na vstupní obraz a ukazatel na *vector* pro zaznamenání získaných souřadnic.

6.2.3 Část testování úspěšnosti

Testování úspěšnosti klasifikátoru je založeno na provedení detekce nad množinou pozitivních testovacích příkladů. Měří se počet snímků, v nichž byl detekován právě jeden objekt (obličej). Úspěšnost klasifikátoru je pak dána poměrem celkového počtu snímků k počtu správně detekovaných snímků.

Implementace této části aplikace je podobná části „Detekce“. Nejdříve je načten daný klasifikátor, opět pomocí funkce *LoadClassifier*. Dále je získán seznam všech souborů v adresáři s testovacími snímky a na každém z nich se provede detekce funkcí *DetectObjects*, která vrací počet detekovaných objektů. Zároveň je měřena statistika úspěšnosti detekce, jak byla vysvětlena v tomto textu výše.

6.2.4 Část vizualizace klasifikátoru

V této části aplikace je úkolem znázornit vnitřní strukturu souboru s konfigurací natrénovaného klasifikátoru. Tento soubor je programem haartraining ukládán v obecném XML formátu. K získání struktury klasifikátoru je potřeba tento soubor přečíst a obsah vhodným způsobem interpretovat. Nemusíme k tomu však použít další knihovnu pro zpracování XML souborů. Knihovna OpenCV má vestavěn potřebný interpret, což vlastně vyplývá i z toho, že umí klasifikátor z XML souboru načíst a zpracovat sama. Tuto funkcionalitu skrývá víceúčelová funkce *cvLoad*, která daný XML soubor načte do předem definovaných datových struktur. Odtud je již možné získat popis klasifikátoru přímočaře. Jednotlivé datové struktury jsou zdokumentovány v manuálu OpenCV [7] a získaní jejich údajů je jen otázkou přístupu k záznamům a polím.

Kaskádový klasifikátor se v OpenCV skládá ze stádií, stádium z jednotlivých klasifikátorů a klasifikátor z Haarových příznaků. Jednotlivá stádia jsou získána ze struktury *CvHaarClassifierCascade*, jednotlivé klasifikátory příslušné kaskády nabízi struktura *CvHaarStageClassifier*, Haarovy příznaky klasifikátoru jsou k nalezení ve struktuře *CvHaarClassifier* a konkrétní části jednoho příznaku obsahuje struktura *CvHaarFeature*. Informacemi z těchto struktur jsou naplněny ovladatelné prvky *wxList*, z kterých si uživatel může vybírat položky.

Vizualizace Haarových příznaků spočívá v načtení údajů konkrétního příznaku a jejich převedení do grafické formy. O každém příznaku jsou v příslušných strukturách zaznamenány souřadnice, velikosti a váhy jeho dílčích obdélníků. Obdélníky můžou být dva nebo tři. Dále jsou k příznakům vedeny informace o jejich pootočení o 45°, tedy zda jsou diagonální či nikoliv. Všechny tyto informace kromě váhy (ta není pro vizualizaci potřebná) jsou využity vlastní funkcí *DrawFeature* k zakreslení obdélníků do rastrového obrazu. Kreslení je umožněno provést v libovolném měřítku. Funkce *DrawFeature* vrací obraz daných rozměrů jako objekt třídy *wxBitmap*. Obraz je v aplikaci dále zobrazen v okně odvozeném od třídy *wxScrolledWindow*.

Kapitola 7

Testování

Na implementované aplikaci nyní představím provedené testování. Jedná se o srovnání dvou detektorů: jednoho s použitím kaskády klasifikátorů a druhého bez této kaskády. Cílem bylo ověřit, zda kaskáda v praxi opravdu přináší znatelně vyšší rychlost detekce, a případně změřit, jak je toto zrychlení veliké.

7.1 Podmínky trénování

Trénovací množina zvolená k trénování obsahovala asi 2400 pozitivních příkladů obličejů o rozměrech 19 x 19 pixelů a zhruba 23500 negativních příkladů o stejných rozměrech. Tento téměř desetinásobný počet negativních příkladů jsem zvolil z toho důvodu, že v době testování jsem neměl k dispozici jinou databázi snímků s větším rozlišením. Jinak by postačilo například asi 4000 až 5000 negativních příkladů s rozlišením 384 x 288 pixelů.

Před samotným trénováním bylo nutné nastavit parametry obou detektorů a kritérium ukončení trénování. Při volbě parametrů jsem vycházel z přednastavených hodnot programu haartraining. Pro oba detektory jsem zvolil jako boostingový algoritmus obecně doporučovaný *Gentle AdaBoost*, množinu Haarových příznaků jsem nastavil na svislé a vodorovné (diagonální příznaky se projevily jako podstatně náročnější na výkon počítače), rozměry detekčního okna jsem nastavil na 19 x 19 pixelů (celá trénovací množina je v tomto rozlišení), povolil jsem optimalizaci pro symetrické objekty (obličeje lze považovat za symetrické) a zbylé parametry jsem ponechal ve výchozím nastavení.

Kritéria pro ukončení trénování byla nastavena s cílem dosažení přibližně stejné úspěšnosti detekce obou detektorů. První detektor jsem tedy nastavil tak, aby poměr falešně pozitivních odezev v každém stádiu byl menší jak 0,5 a počet stádií jsem zvolil 10. To by mělo ve výsledku dát detektor s poměrem falešně pozitivních odezev minimálně $0{,}5^{10}$, tedy 0,000976562. Nejde o žádnou vysokou přesnost, pro testování však postačuje. Druhý detektor byl nastaven pro vytvoření jediného stádia s tím, že by trénování mělo v tomto stádiu dosáhnout stejného poměru falešně pozitivních odezev jako celá kaskáda prvního detektoru.

Trénování jsem provozoval na počítači s procesorem Intel Celeron M 1,3 GHz a operační pamětí 512 MB. Pro trénování detektoru nejde o příliš vhodnou konfiguraci. K dispozici jsem sice měl výkonnější stroj, ale na něm se mi nepodařilo trénovací aplikaci včas zprovoznit.

7.2 Průběh trénování

Trénování prvního detektoru s kaskádami bylo v pozdějších stádiích problematické. Jak se dalo očekávat, potýkal jsem se především s nedostatečným výkonem počítače a nedostatkem operační paměti. Vlivem toho byl postup v trénování pomalý. Především ve fázích mezi stádii, kde trénovací algoritmus zpracovával snímky, docházelo k neúnosně dlouhým prodlevám. Po přerušení a opětovném spuštění trénování od místa přerušení však k žádné prodlevě již nedošlo. Příčinu se mi nepodařilo objasnit. Nicméně i přesto jsem trénování dokončil. Jen pro zajímavost uvádím, že doba zpracování negativních příkladů se s každým novým stádiem zdvojnásobovala a v desátém stádiu dosahovala téměř 4 hodin. Pro další dvě stádia by jen toto zpracování trvalo téměř celý den. A v tomto čase ještě není započteno samotné trénování.

Abych trénování urychlil, experimentoval jsem s nižším počtem negativních příkladů (stále na stejné množině snímků 19 x 19 pixelů), ale v tomto případě trénování pravidelně selhávalo, protože algoritmus po nějaké době již nenalézal dobré slabé klasifikátory, které by dále minimalizovaly chybu detektoru.

Trénování druhého detektoru bez kaskády bylo podstatně rychlejší a bezproblémové. Došlo zde jen k jednomu zpracování negativních příkladů (pokud nepočítám závěrečné měření přesnosti detektoru), takže problémy spojené s trénováním prvního detektoru se nestihly projevit. Během trénování byly stále nalézány nové slabé klasifikátory a chyba úspěšně klesala až pod stanovenou mez.

Pokud zanedbám zmíněné potíže s prvním detektorem, trvalo jeho natrénování odhadem asi 16 hodin. Druhý detektor se podařilo na stejném stroji natrénovat bez přerušování za necelé 4 hodiny.

7.3 Výsledné detektory

Výsledky trénování jsou popsány v tabulce 7.1. Sloupec HR (hit rate) v tabulce označuje poměr kladných odezev pro pozitivní příklady trénovací množiny (ideálně 1,0) a sloupec FA (false alarm rate) označuje poměr falešně pozitivních odezev pro negativní příklady trénovací množiny (ideálně 0,0).

detektor	**stádií**	**příznaků**	**HR**	**FA**	**čas trénování**
1. s kaskádou	10	89	0,958419	0,000127269	16h?
2. bez kaskády	1	43	0,995060	0,000593915	4h

Tabulka 7.1: Výsledné parametry trénovaných detektorů.

V tabulce stojí za povšimnutí rozdíl v poměru kladných odezev (HR), na nějž má vliv použití kaskády klasifikátorů. Před trénováním byl nastaven tento poměr na minimální požadovanou hodnotu 0,995. Detektoru bez kaskády se tento požadavek podařilo dodržet, protože po celou dobu trénování pracoval s kompletní trénovací množinou a algoritmus mohl vybírat nové příznaky s ohledem na dodržení této hodnoty. Detektor s kaskádou ovšem tento poměr nedodržel. Je to dáno tím, že při vytváření nových stádií kaskády dochází k redukci trénovací množiny. Jakmile dřívější stádia nějaký příklad klasifikují záporně, je tento snímek z trénovací množiny vyřazen[1]. Chyba zanesená každým novým stádiem pak

[1] Při jeho ponechání by další stádia byla natrénována na snímku, který nikdy nebudou klasifikovat.

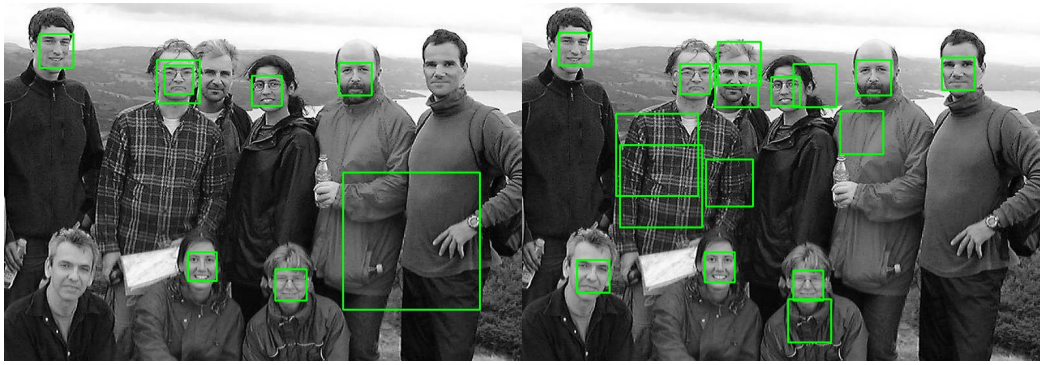
kumuluje a v pozdějších stádiích ji nelze ovlivnit ani tím nejlepším výběrem příznaků.

Odlišné hodnoty dosažených poměrů falešně pozitivních odezev (FA) jsou dány jen tím, jaké se podařilo nalézt poslední příznaky před dosažením kritéria pro ukončení trénování.

Dále tabulka 7.1 ukazuje počty stádií, které odpovídají nastaveným parametrům, a celkové množství příznaků v detektorech. Prozkoumání příznaků ve vizualizační části implementované aplikace ukázalo, že příznaky v prvním stádiu detektoru s kaskádou přesně odpovídají prvním příznaků detektoru bez kaskády. Ze začátku totiž probíhalo trénování obou detektorů na stejné kompletní trénovací množině. Další příznaky se již vzájemně liší, protože detektor s kaskádou byl od druhého stádia trénován na redukované množině.

7.4 Ukázka detekce

Na následujícím snímku je ukázka výstupu detekce obličejů na skupinové fotografii. Vlevo je výstup detektoru s kaskádou a vpravo výstup detektoru bez kaskády. Na snímku vpravo je vidět větší množství falešně pozitivních odezev (dáno téměř 5× vyšším poměrem falešně pozitivních odezev tohoto detektoru).



Obrázek 7.1: Ukázka výstupu detektorů (zdroj původního snímku: Bao Face Database)

7.5 Testovací sady

K otestování detektorů jsem použil tři různé sady pozitivních příkladů. Pro všechny sady platí, že na každém snímku je právě jeden obličej.

- Sada **A** - 149 barevných snímků, téměř žádné pozadí, průměrně cca 170×170 pixelů
- Sada **B** - 1521 šedotónových snímků, 384×286 pixelů
- Sada **C** - 450 barevných snímků, mnoho pozadí, 896×592 pixelů

7.6 Úspěšnost detektorů

Zde pro zajímavost uvádím orientační úspěšnosti detektorů změřené v implementované aplikaci na zmíněných testovacích sadách (viz tabulka 7.2). Za úspěšně detekovaný snímek byl považován takový, v němž byl nalezen právě jeden objekt. Nejde o nijak přesný způsob

měření, protože detektor může ve snímku detekovat i něco, co ve skutečnosti není obličej, ale věřím, že naměřené výsledky se blíží pravdě alespoň ve vzájemných vztazích.

Úspěšnost jsem měřil jednou s podvzorkování všech snímků na poloviční rozměry a podruhé bez tohoto podvzorkování s původními rozměry snímků.

Výsledky v tabulce ukazují, že oba detektory jsou na tom s úspěšností většinou podobně. Za nízkou úspěšnost u sady **C** může přítomnost větších oblastí s pozadím ve snímcích, kde se projevují falešně pozitivní odezvy detektorů. Dále je ve výsledcích vidět, že podvzorkování snímků na poloviční rozměry pomáhá lepší detekci. Zřejmě je to způsobeno ztrátou části informací z podvzorkovaného obrazu.

testovací sada	**s podvzorkováním**		**bez podvzorkování**	
	s kaskádou	bez kaskády	s kaskádou	bez kaskády
Sada A	42 %	48 %	52 %	43 %
Sada B	56 %	47 %	33 %	12 %
Sada C	30 %	12 %	1 %	0 %

Tabulka 7.2: Výsledné úspěšnosti detektorů změřené na testovacích sadách s a bez podvzorkování snímků na poloviční rozměry.

7.7 Rychlost v detekci

Na závěr představím výsledky měření rychlosti detekce (tabulka 7.3). Tyto výsledky prokázaly, že v praxi dosahuje detektor s kaskádou skutečně vyšší rychlosti detekce. Průměrně detektor s kaskádou detekoval $1,35\times$ rychleji, než detektor bez kaskády.

V tabulce byly naměřené doby detekce přepočítány na počet snímků za vteřinu a kvůli různým velikostem snímků testovacích sad také na počet megapixelů za vteřinu (Mpx/s).

testovací sada	**rozměry snímku**	**s kaskádou**		**bez kaskády**	
		snímků/s	Mpx/s	snímků/s	Mpx/s
Sada A	cca 170×170 px	53,2	1,5	48,1	1,4
Sada B	384×286 px	25,1	2,8	18,8	2,1
Sada C	896×592 px	5,2	2,8	3,2	1,7

Tabulka 7.3: Rychlosti detektorů změřené na testovacích sadách.

Kapitola 8

Závěr

Navrhovaná aplikace byla úspěšně implementována a všechny její části odzkoušeny. Aplikace je plně funkční, vše co bylo navrženo, se podařilo včas implementovat a v rámci této vývojové etapy nezůstala žádná část nedokončená.

V aplikaci by samozřejmě šlo navrhnout a implementovat celou řadu dalších rozšíření. V budoucí práci bych se tak mohl pokusit například o vytvoření nástroje pro přípravu trénovacích dat. Jednalo by se o součást aplikace umožňující vytvořit z dostupných snímků sadu výřezů konkrétního objektu. Většinou se totiž nepodaří k trénování získat nebo vytvořit snímky, které by obsahovaly pouze daný objekt bez rušivého pozadí. Dalším rozšířením by mohla být hlouběji propracovaná část „Testování úspěšnosti“. Tato část by disponovala nastavitelnými parametry, např. kolik výskytů objektu se předpokládá na testovacích snímcích. Také by přišel vhod detailní popis výsledků testování, např. toho, v kterých konkrétních snímcích testovací sady detektor selhal. Další rozšíření by se mohlo týkat detekční části aplikace a jejího vybavení ovládacím prvkem pro rychlé přepínání mezi snímky v aktuálním adresáři. Někdy je potřeba kontrolovat úspěšnost detektoru manuálně prohlížením jeho výstupů na více snímcích. Současná jediná cesta volby vstupního snímku přes dialog výběru souboru je pro tyto účely příliš pomalá. Poslední rozšíření, které zmíním, spadá do jiné kategorie. Jednalo by se o modifikaci detekčního algoritmu umožňující detekovat obličeje či jiné objekty v polohách, s jakými původní detektor nepočítá. Například detekce obličejů nakloněných o různé úhly může být zajímavým námětem k řešení.

Literatura

[1] Florian A. *How-to build a cascade of boosted classifiers based on Haar-like features* [online], 2003-09-02. Dostupné na URL http://robotik.inflomatik.info/other/opencv/OpenCV_ObjectDetection_HowTo.pdf (duben 2008).

[2] Hjelmås E., Lerøy C. B., Johansen H. *Detection and Localization of Human Faces in the ICI System: A First Attempt* [online]. Report #6, pages 2–3. Gjøvik College, 1998. Dostupné na URL http://www.ansatt.hig.no/erikh/papers/hig98_6.pdf (květen 2008).

[3] Hradiš M. *AdaBoost v počítačovém vidění*. Diplomová práce, Ústav počítačové grafiky a multimédií FIT VUT v Brně, 2007. Vedoucí diplom. práce Ing. Igor Potúček, Ph.D.

[4] Viola P., Jones M. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 511–518, 2001. ISSN 1063-6919, ISBN 0-7695-1272-0.

[5] Adaboost. *Wikipedia: The Free Encyclopedia* [online]. Dostupné na URL http://en.wikipedia.org/wiki/AdaBoost (květen 2008).

[6] Statistical classification. *Wikipedia: The Free Encyclopedia* [online]. Dostupné na URL http://en.wikipedia.org/Statistical_classification (květen 2008).

[7] *OpenCV Reference Manual* [online]. Dostupné na URL http://opencvlibrary.sourceforge.net (květen 2008).

Seznam zkratek

AdaBoost	Adaptive boosting
API	Application programming interface
BG	Background file
FA	False alarm rate
GTK	GIMP Toolkit
GUI	Graphical user interface
HR	Hit rate
OpenCV	Open Source Computer Vision
URL	Uniform resource locator
VEC	Vector file
XML	Extensible markup language

Seznam příloh

A Návod k použití aplikace
B Snímky aplikace
C Disk CD (v zadní části výtisku)

Příloha A

Kompilace a spuštění

Aplikaci je možné zprovoznit na systémech Windows, Linux a údajně i Mac OS X (nebylo vyzkoušeno). Tyto systémy jsou podporovány knihovnami OpenCV a wxWidgets současně. S aplikací je dodáván Makefile, ale pouze pro operační systémy Linux. Pro ostatní systémy bude nutné Makefile nejdříve vyrobit, například s pomocí nástroje Bakefile.

Vyžadovány jsou nainstalované knihovny OpenCV verze 1.0.0 a wxWidgets nejlépe verze 2.8.7. Pro zkompilování pod systémem Linux je potřeba GNU kompilátor GCC nejlépe verze 3.x a vyšší. Kompilaci lze spustit příkazem `make`.

Aplikaci lze po zkompilování v Linuxu spustit příkazem `./adaboost`. Detekci je možné vyzkoušet na již existujících klasifikátorech, které jsou dodávány s knihovnou OpenCV a nacházejí se v jejím adresáři v podadresáři „data“. Jinak lze v aplikaci vytvořit vlastní klasifikátor pomocí trénování.

Trénovací data

Pro trénování jsou potřeba dvě sady snímků - sada s pozitivními příklady a sada s negativními příklady. Pozitivní příklady by měly být snímky s výřezy zvoleného objektu, které chceme detektor naučit detekovat. Negativní příklady můžou být libovolné snímky bez výskytu zvoleného objektu.

Například pro natrénování detektoru hrušek bude sada pozitivních příkladů tvořena snímky hrušek ze stejného úhlu pohledu a při stejné poloze hrušek (hruška např. nesmí být nasnímána jednou naležato a jednou nastojato). Negativní příklady budou libovolné snímky, v kterých nejsou vyobrazeny podobné nebo pro jistotu žádné hrušky.

Dobré pozitivní příklady obsahují jen daný objekt a co nejméně pozadí. Umístění objektu na snímcích by mělo být neměnné a jeho významné znaky by měly být vždy na přibližně stejných místech v obraze v rámci celé sady snímků. Snímky nemusejí být příliš rozměrné. Před trénováním se totiž převádějí do šedotónových vzorků o malé velikosti kolem 24×24 pixelů (záleží na nastavení a konkrétním případu). Negativní příklady mohou mít rozměry např. 384×288 pixelů i více. Všechny snímky jsou aplikací podporovány v následujících obrazových formátech:

- Windows bitmap (*.bmp, *.dib)
- JPEG (*.jpeg, *.jpg, *.jpe)
- Portable Network Graphics (*.png)
- Portable image format (*.pbm, *.pgm, *.ppm)
- Sun rasters (*.sr, *.ras)
- TIFF (*.tiff, *.tif)
- OpenEXR HDR (*.exr)
- JPEG 2000 (*.jp2)

Snímky je potřeba rozdělit do dvou adresářů - jeden s pozitivními a druhý s negativními příklady.

Nastavení trénování

Ještě před trénováním lze v aplikaci pod záložkou **Tr.Nastavení** vybrat parametry trénování. Případně lze ponechat přednastavené hodnoty. Zde vysvětlím některé parametry.

Položky **Pozitivních příkladů** a **Negativních příkladů** určují, kolik má být použito příkladů z příslušných adresářů s trénovacími daty. Pokud je některá položka nastavena na nulu, použijí se všechny příklady. **Počet stádií** určuje maximální počet stádií, jaký může být v kaskádě klasifikátorů vytvořen. **Využití paměti v MB** určuje maximální množství operační paměti využitelné k předzpracování trénovacích dat. Tuto hodnotu je potřeba volit přiměřeně podle aktuálního množství volné paměti v systému. Vhodným nastavením lze dosáhnout vyšší rychlosti trénování. Nastavení příliš vysoké hodnoty (např. celá kapacita RAM) však může trénování naopak zpomalit. **Minimální podíl pozitivních odezev** určuje minimální podíl pozitivních příkladů, jaký musí být v trénování pozitivně klasifikován. **Maximální podíl falešně pozitivních odezev** pak určuje maximální podíl případů falešně pozitivní klasifikace v množině negativních příkladů. **Množina Haar featur** nastavuje sadu Haarových příznaků použitých jako slabé klasifikátory. **Symetrické objekty** lze nastavit na ANO, pokud budou trénovány symetrické objekty jako třeba hrušky, obličeje, dopravní značky, aj. **Šířka** a **Výška okna v pixelech** určují rozměry detekčního okna a tím i velikost vzorků, na jakou budou před trénováním převedeny pozitivní příklady. Parametrem **Typ boostingu** lze nastavit konkrétní variantu boostingového algoritmu použitého k trénování.

Spuštění trénování

V záložce **Trénování** je nutno zvolit adresář s pozitivními příklady, adresář s negativními příklady a dále adresář, kam bude uložen výstupní XML soubor s natrénovaným klasifikátorem. Poté kliknutím na tlačítko **Konfiguruj** se prohledají zvolené adresáře a z nalezených snímků se vytvoří pomocné konfigurační soubory. Nakonec je možné tlačítkem **Start** spustit samotné trénování. Výstupy trénování se budou postupně objevovat v okně pod záložkou **Tr.Výstup**.

Aplikace neumožňuje zastavit běžící trénování. Přerušit jej lze jen tlačítkem pro zavření aplikace v záhlaví jejího okna nebo násilným ukončením.

Navázání trénování

V aplikaci lze návázat na předchozí trénování, které bylo přerušeno uživatelem nebo pádem aplikace. Pokračovat lze i v trénování, které již úspěšně skončilo. Postup je podobný jako při spuštění nového trénování. Opět je nejdříve nutno nastavit parametry trénování pod záložkou **Tr.Nastavení**. Parametry se můžou lišit od těch předchozích. Pod záložkou **Trénování** je dále potřeba znovu zvolit adresáře s pozitivními a negativními příklady. Výstupní adresář se musí nastavit stejný jako v předchozím trénování. Pak lze pokračovat dvěma způsoby. Aplikace si pamatuje poslední provedenou konfiguraci trénovacích dat, takže pokud nebyly v části **Tr.Nastavení** změněny rozměry detekčního okna a nebyly

zvoleny jiné adresáře s příklady, je možné vynechat konfigurační proceduru a rovnou spustit trénování tlačítkem **Pokračovat**. V opačném případě je před pokračováním nutná nová konfigurace tlačítkem **Konfiguruj** a až poté lze spustit trénování tlačítkem **Pokračovat**.

Pokud při vynechání konfigurace ihned po kliknutí na tlačítko **Pokračovat** aplikace skončí, je to pravděpodobně kvůli starým konfiguračním souborům, které nejsou v souladu s nově nastavenými parametry. V takovém případě je řešením před spuštěním trénování pomocí tlačítka **Konfiguruj** vytvořit novou konfiguraci.

Pozn.: Náhlé ukončení není způsobeno chybou v aplikaci. Jde o nevýhodu spojenou s použitím zdrojových kódů programu haartraining, který při problému okamžitě ukončí celou aplikaci. Odstranění tohoto nepříjemného chování je námětem pro budoucí vývoj aplikace.

Vizualizace klasifikátoru

Pod záložkou **Klasifikátor** lze zobrazit strukturu natrénovaného klasifikátoru včetně vizualizace příznaků. Stačí zvolit XML soubor klasifikátoru. Ten je následně načten, zpracován a zobrazen v podobě seznamu stádií, klasifikátorů a příznaků, mezi nimiž je možné přepínat a tím zobrazovat další informace.

Testování úspěšnosti

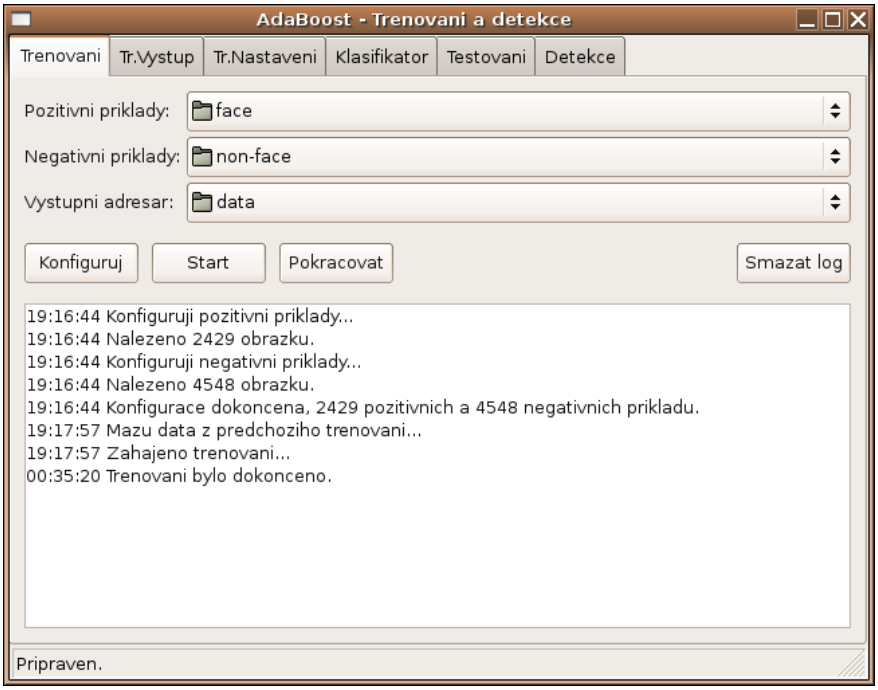
Testování úspěšnosti detektoru vyžaduje adresář s pozitivními testovacími příklady. Mělo by jít o sadu snímků s objektem, na němž byl detektor natrénován. Každý snímek musí obsahovat právě jeden tento objekt, ale nemusí to být přesný výřez. Snímky mohou být větších rozměrů, mohou obsahovat více pozadí a objekty v nich mohou být různě umístěny. Tato testovací sada by ale neměla být totožná se sadou pozitivních příkladů, na které byl detektor trénován. Testování by tak nevypovídalo o skutečné úspěšnosti detektoru.

V aplikaci pod záložkou **Testování** stačí zvolit XML soubor testovaného klasifikátoru, zvolit adresář s testovacími snímky a dále kliknout na tlačítko **Start**, které zahájí měření. Výsledná úspěšnost je po měření zobrazena ve spodní části aplikace. Vyjádřena je procentuálně a také počtem úspěšně detekovaných snímků z celkového počtu snímků.

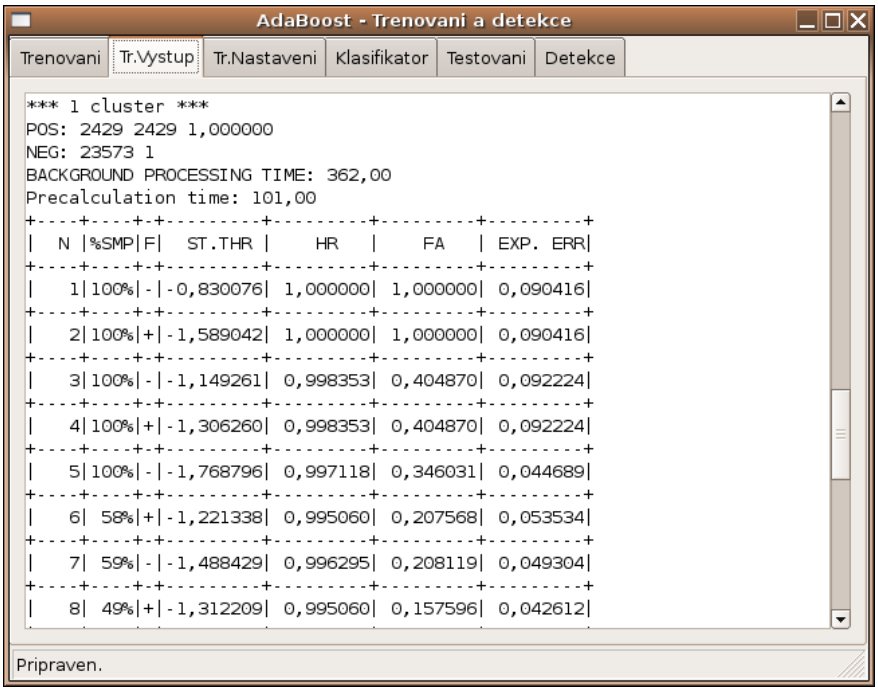
Detekce

V této části, pod záložkou **Detekce**, lze zobrazit výstup detektoru na konkrétním snímku. Nejdříve se zvolí XML soubor klasifikátoru a dále vstupní snímek, v němž má být detekce provedena. V okně níže je pak zobrazen zvolený snímek s vyznačením všech detekovaných objektů. Spodní část aplikace indikuje celkový počet nalezených objektů.

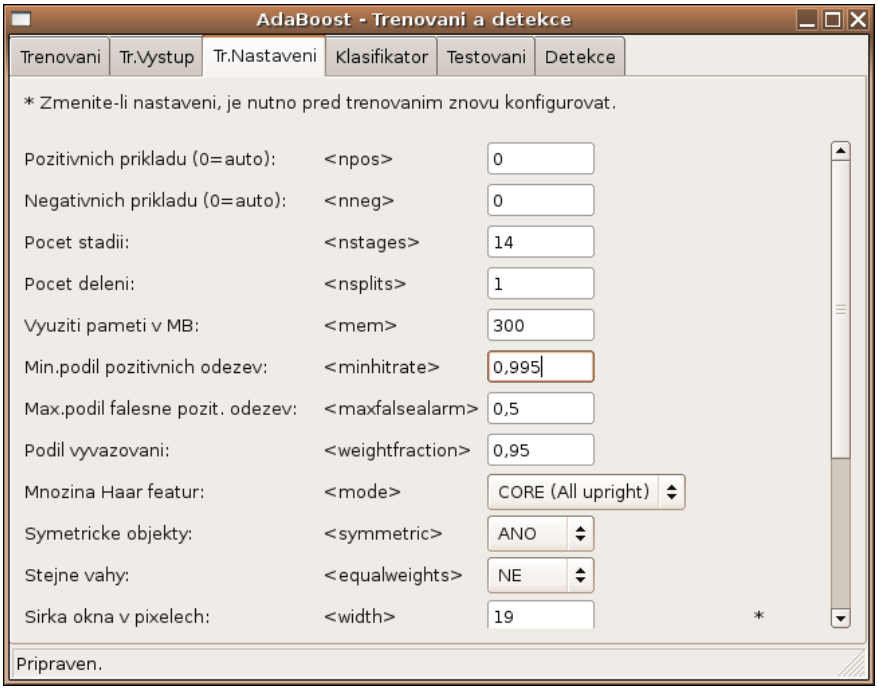
Příloha B



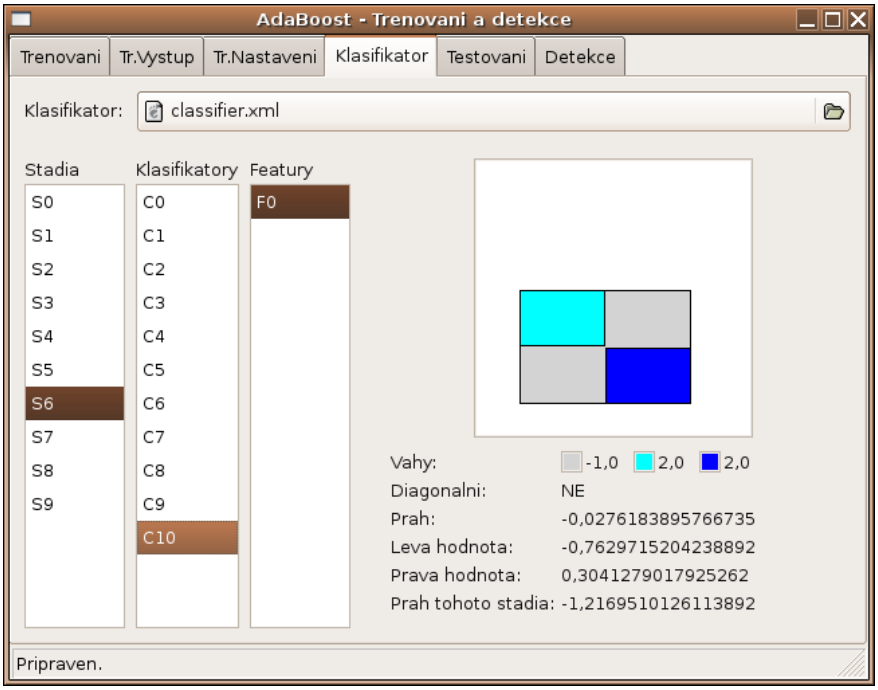
Trénovací část: S výpisem po konfiguraci a trénování.



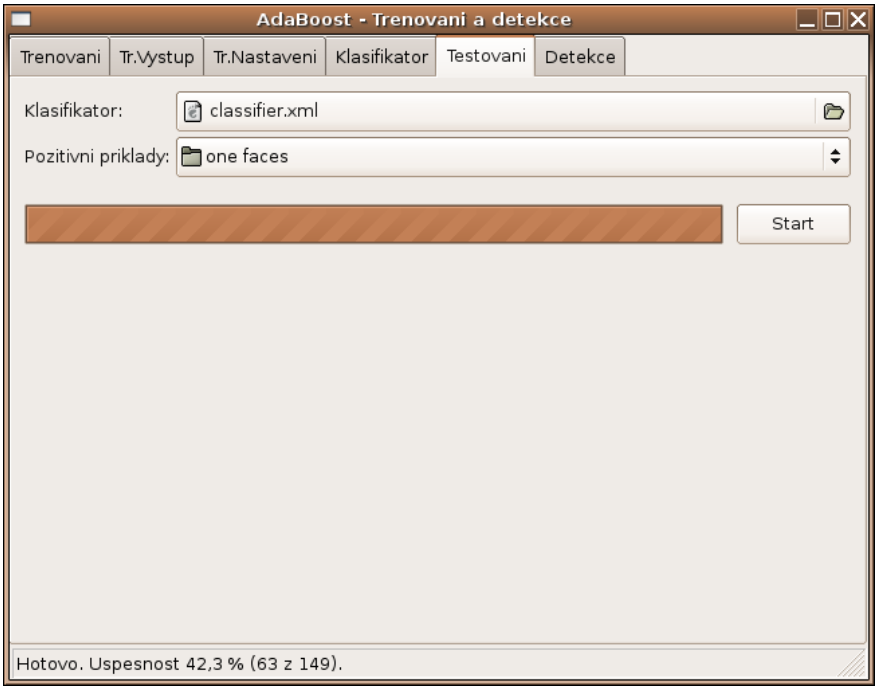
Trénovací část: Výstup trénování produkovaný programem haartraining.



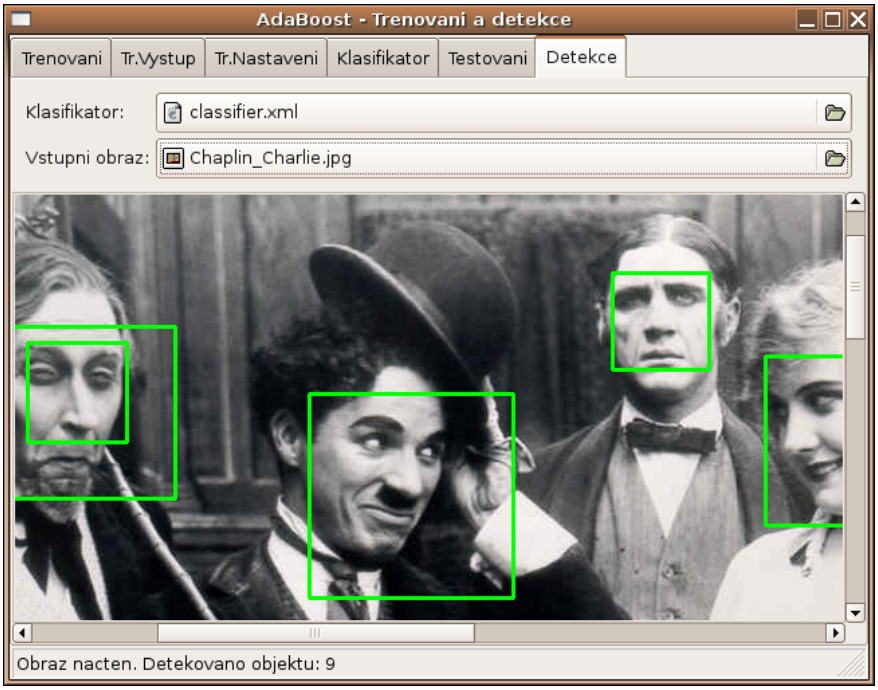
Trénovací část: Nastavení parametrů trénování.



Vizualizace klasifikátoru: Zobrazuje stádia, klasifikátory a příznaky.



Testovací část: Ve spodní části zobrazuje naměřenou úspěšnost.



Detekční část: Výstup detekce obličejů v černobílém snímku.