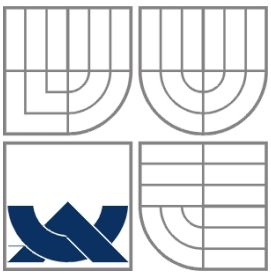
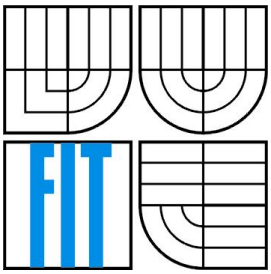


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OBLIČEJOVÝ ANONYMIZÉR

FACE ANONYMIZER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN PEŠA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2009

Abstrakt

V této bakalářské práci lze nalézt přehled klasifikačních algoritmů a jejich použití zejména pro prohledávání obrazových dat a detekcí tváří. V první části je nastíněn úvod do rozpoznávání obrazů, je popsáno teoretické pozadí těchto algoritmů a způsob jejich trénování. Představeny jsou i další použité prvky (například Kalmanův filtr nebo knihovna OpenCV). V druhé části se se nachází popis implementace a výstavby aplikace, která využívá těchto technologií pro vyhledávání, sledování a anonymizaci lidských obličejů ve video vstupu.

Klíčová slova

detekce obličeje, detekce očí, klasifikace, anonymizace, AdaBoost, WaldBoost, Haarovy příznaky, Local Rank Differences, Kalmanův filtr

Abstract

In this bachelor thesis you can find an overview of classification algorithms and their usage especially for searching image data and face detection. First part contains a brief introduction to a pattern recognition, a theoretical background of these algorithms and ways of training them. Other used components are also presented (e.g. Kalman filter or OpenCV library). Second part covers an implementation of the application which uses these technologies for searching, tracking and anononymization of human faces in a video stream.

Keywords

face detection, eye detection, classification, anonymization, AdaBoost, WaldBoost, Haar features, Local Rank Differences, Kalman filter

Citace

PEŠA, J.: *Obličejový anonymizér*. Bakalářská práce, Brno, FIT VUT v Brně, 2009. Ved. Aleš Láník

Obličejový anonymizér

Prohlášení

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

.....
Jan Peša
20. května 2009

Poděkování

Tímto bych chtěl poděkovat Ing. Aleši Láníkovi za odbornou pomoc a rady při tvorbě bakalářské práce. Dále bych chtěl poděkovat Bc. Martinu Malíkovi a mé sestře Daniele Pešové, kteří mi stáli za figuranty při vytváření některých snímků z fungování aplikace. A samozřejmě mým rodičům za jejich neustálou a plnou podporu v mých studiích.

Obsah

1 Úvod

Počítačové vidění a automatická analýza obrazu jsou dnes jedny z nejaktuálnějších témat. Tato oblast je – v porovnání s ostatními odvětvími informačních technologií – také poměrně mladá. Podléhá neustálému vývoji a jsou na ni zaměřeny značné lidské i finanční zdroje, neboť se u ní očekává velká budoucnost a slibné výsledky. Vše se odvíjelo od dostupné techniky, neboť zpracování velkých objemů obrazových dat je i značně výpočetně náročné. K většímu rozvoji došlo tedy až koncem sedmdesátých let dvacátého století.

V dnešní době, kdy máme dohledové kamerové systémy ve většině velkých měst na každém druhém rohu, se mnoho společností či vládních organizací zabývá zpracováním jejich výstupů a pokud možno co největší automatizací těchto úkonů z důvodu efektivity a rychlosti. I na akademické půdě vznikají neustále zajímavé a prospěšné projekty. Jedná se většinou a detekci různých objektů v obrazových záznamech. Tyto projekty pak slouží hlavně na veřejných prostranstvích – např. v metrech, kde hledají podezřelé předměty nebo nežádoucí chování (cestování s kolem, přeskakování turniketů, apod.). Po takovýchto aplikacích je velká poptávka.

I tato bakalářská práce se zabývá počítačovým viděním. Ne však z důvodu něčího odhalení, nýbrž naopak skrytí a anonymizace. Proto si myslím, že je to zajímavý protiproud v dnešní době, kdy se skoro na každém kroku setkáváme s někým nebo něčím, co nás chce identifikovat, určit kdo jsme, kam směřujeme a co děláme. Je to pomyslný ostrůvek soukromí v dnešním světě, který se v nadneseném slova smyslu začíná podobat tomu z románu G. Orwella.

Práce je rozdělena na dvě hlavní části. V té první je nastíněna potřebná teorie a problematika tohoto oboru, v další následuje již popis implementace. Z teorie je popsáno na jakých principech funguje rozpoznávání obrazů, vybrané druhy klasifikátorů, dále příznaky v obrazu používané při klasifikaci a další využité technologie jako např. Kalmanův filtr. V implementační části je popsán přístup při návrhu a následná realizace samotného programu a problémy, kterým bylo nutné čelit.

Program dokáže ve video vstupu nalézt lidské tváře a na vyžádání je buďto rozmazat nebo překrýt obě oči černou páskou, aby nebylo možné zjistit identitu dané osoby. Využívá volně šiřitelnou knihovnu OpenCV a je z části založen na práci Ústavu počítačové grafiky a multimédií (UPGM) na FIT VUT v Brně a jejich obličejovém detektoru. Pokud bychom se zamysleli nad reálným využitím podobné aplikace, lze přijít s několika možnými směry uplatnění:

- rozhovory s důležitými svědky, jejichž identita nesmí být prozrazena
- anonymizace osob na videozáznamech, kde autor nemá svolení k pořízení obrazového materiálu daných osob
- skrytí identity lidí ve videozáznamech, kde byl záznam určen pouze k statistickému sledování pohybu osob v daném místě
- aktuálně citlivé téma s radarovým měřením a kamerovým záznamem projíždějících vozů
- anonymizace osob v záznamu, který je volně dostupný a mohl by poškodit něčí soukromí či narušit osobní život

Jak ale bude zmíněno hned v úvodu následující kapitoly, úspěšnost počítačového vidění v současnosti závisí na několika klíčových faktorech, tudíž by nasazení takovéto aplikace do ostrého provozu bylo limitováno několika faktory, které jsou zmíněny v samotném závěru práce.

2 Rozpoznávání obrazů

Klasickou otázkou počítačového vidění je, zda-li obrazová data obsahují určitý objekt, vlastnost nebo činnost. Pro nalezení odpovědi dnes existuje mnoho metod, které ovšem řeší tento problém pouze pro specifické objekty, jako např. jednoduché geometrické útvary, lidské obličeje, tištěné nebo rukou psané písmo, automobily a další. Dále je třeba příznivých podmínek – dostatečné nasvícení scény, vhodné pozadí a vhodná poloha a natočení objektu vůči kameře. Různé přístupy v rozpoznávání jsou citlivé na odlišné faktory.

Rozpoznávání se tedy zabývá otázkou, jak pro zvolený popis objektu nejlépe navrhnout vhodný klasifikátor, jaké příznaky vyhledávat a jak je správně ohodnotit.

Celkově bychom tuto problematiku mohli rozdělit na 3 hlavní odvětví:

- **Rozpoznávání** – rozpoznáváme jeden nebo více objektů či objektových tříd.
- **Identifikace** – jednoznačná instance objektu, např. identifikace obličeje určité osoby, otisku prstu, konkrétního automobilu.
- **Detekce** – obrazová data jsou zkoumána na specifickou podmínku, např. detekce na shluky nádorových buněk v lékařských snímcích. Tyto oblasti obrazu nalezené jednoduchými a rychlými výpočty se dále postoupí více komplexním algoritmům pro hlubší analýzu.

2.1 Klasifikace

Klasifikace obrazu je proces, při kterém speciální funkce nebo modul hodnotí s jakou pravděpodobností se na daném místě nachází hledaný objekt. Klasifikaci můžeme rozdělit na:

- Binární – vrací hodnotu „ano“ nebo „ne“ v závislosti na výsledku hodnocení.
- Vícehodnotovou – vrací hodnotu v určitém rozsahu, který představuje pravděpodobnost, s jakou se hledaný objekt nachází na daném místě.

Klasifikátory nejčastěji hodnotí množinu tzv. příznaků, jež můžeme považovat za elementární popisy dané části dat (v našem případě obrazových), a jejichž správné vyhodnocení zodpoví otázku, zda-li vybraná data spadají do třídy objektu či nikoliv. Předmět nebo jev, který má být klasifikován, je tedy nejprve vhodně popsán. To znamená, že na předmětu (jevu) je vybrána množina příznaků, jejichž způsob získávání je apriorně znám. Klasifikátor má danou vstupní *n*-tici v podobě počtu příznaků a jeden výstup, který rozděluje klasifikovaný předmět do tříd.

Zatím neexistuje obecná metoda, která by dokázala určit, jaké příznaky na daném předmětu nebo jevu měřit. Příznaky musí volit konstruktér klasifikátoru například na základě konzultace s odborníkem z dané problémové oblasti. Teoreticky by šlo problém volby vhodných příznaků řešit tak, že na předmětu budeme měřit všechny představitelné vlastnosti, protože čím více příznaků na předmětu změříme, tím více informace bude klasifikátor mít a tím přesněji bude rozhodovat. Bohužel tento přístup je nereálný, protože s rostoucím počtem příznaků se značně komplikuje technická realizace klasifikátorů.

Trénování klasifikátorů

Tato podkapitola byla částečně převzata a přeformulována z [4]. Existuje několik způsobů trénování klasifikátorů – učení bez učitele, kdy dělení vzorů dat do tříd probíhá na základě zadaného kritéria

kvality (též *clustering*) nebo učení s učitelem, kde on rozhoduje o rozdělení do tříd. Obyčejně se trénování provádí právě učením s učitelem. V principu máme různé vzorky dat, u nichž víme, do jaké třídy patří a jejich cíleným předkládáním klasifikátoru jej naučíme předpovědět třídu klasifikace pro neznámá data – tedy generalizovat. Při trénování se snažíme využit co nejlepší možnou sadu trénovacích dat. Samozřejmě není možné při učení obsáhnout všechny možné varianty vstupu, učení nebude nikdy dokonalé a výsledek bude mít tedy určitou chybu, která závisí právě na kvalitě trénovací sady. Komplexní klasifikátory mají chybu menší, ale tomu odpovídá i delší čas potřebný na natrénování a vyhodnocení.

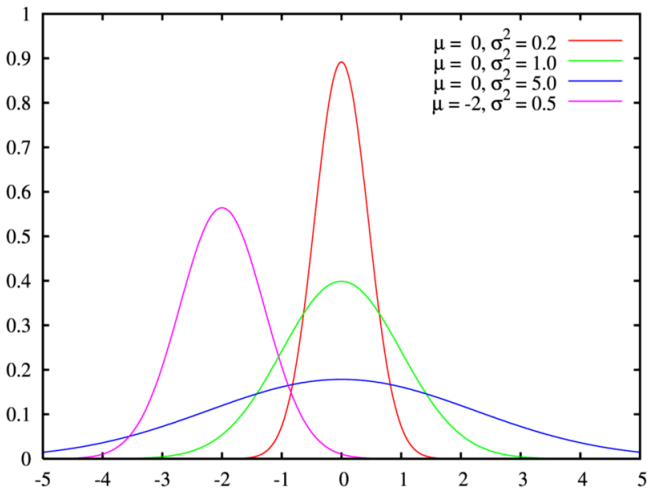
Při trénování se obyčejně používají dvě sady vzorků – trénovací a testovací. Na trénovací sadě se učí klasifikace a na testovací se ověřuje její chyba. Trénovací data obsahují obrazy objektů a jejich rozdělení do tříd. Vezměme například úlohu detekce obličejů, v niž jsou data rozdělena na dvě třídy – *obličej* a *pozadí*. Ve třídě *obličej* se nachází obrázky různých obličejů za různých podmínek. Ve třídě pozadí je vše, co nemá klasifikátor považovat za hledaný objekt. Před započetím trénovaní jsou ze vstupních dat extrahovány příznaky a vytvořen příznakový vektor.

Dále si představíme některé některé z druhů běžných klasifikátorů, z nichž některé jsou použity i v této bakalářské práci.

2.2 Gaussovské modely

V některých případech se pro klasifikaci používá Gaussův model, který je určen pro statistické rozpoznávání vzorů a založen na předpokladu, že objekty z jedné třídy budou mít navzájem podobné určité statistické vlastnosti. Jedná se tedy o učení bez učitele – *clustering*.

Jednou ze statistických technik je modelování hledaných tříd pomoci směsi gaussovských funkcí ve vektorovém prostoru příznaků (*Gaussian Mixture Models – GMM*). Tato technika se často používá například ve zpracování řeči pro detekci fenoménů a obecně je její použití velmi široké. Ve zpracování obrazu se obvykle využívá k detekci částí lidského těla na základě barvy kůže – rozděluje části obrazu do tříd *skin-color* nebo *background*.



Obrázek 2.1: Normální rozložení při různých hodnotách středu a rozptylu, zdroj: Wikipedie

Normální rozložení v jednorozměrném prostoru je definováno svým středem μ a rozptylem σ. Na obrázku 2.1 je znázorněno několik možných výsledků při různých hodnotách těchto veličin. Tímto způsobem lze modelovat pravděpodobnost, že nějaký jev hodnotou svého příznaku x náleží do modelované třídy (rovnice 2.1). Pokud máme vektor příznaků o rozměrech n, je třeba rovněž přejít do n-rozměrného prostoru. Pak lze modelovat pravděpodobnost Gaussovým rozložením, jehož parametry jsou vektor středních hodnot $\vec{\mu}$ matice rozptylů Σ (rovnice 2.2).

$$N(x;\sigma,\mu)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.1}$$

$$N(\vec{x};\vec{\mu},\Sigma)=\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})} \tag{2.2}$$

Bližší vysvětlení použití rovnic je převzato z [4]:

> Vztah (2.2) pro výpočet pravděpodobnosti je potřeba použít pouze v případě, že hodnoty příznaků jsou korelované a distribuce tedy může být v prostoru libovolně natočená. Taková distribuce má plnou kovarianční matici. Na druhou stranu rozložení, které je zarovnáno rovnoběžně s osami prostoru má definované rozptyly pouze ve směru jednotlivých os – jen na diagonále matice. Výpočet pravděpodobnosti lze v takovém případě zjednodušit na násobení jednorozměrných distribucí (2.3). Tímto lze podstatně urychlit výpočet pravděpodobnosti za cenu, že příznaky je před výpočtem pravděpodobnosti nutné dekorelovat například pomocí DCT (diskrétní kosinová transformace).

$$N(\vec{x};\vec{\mu},\vec{\sigma})=\prod_i N(x_i;\mu_i,\sigma_i) \tag{2.3}$$

V praxi data modelujeme součtem více Gaussových distribucí. Jejich počet se odvíjí od složitosti rozmístění příznaků v prostoru. S počtem použitých distribucí roste čas potřebný ke trénování, ale v závislosti na tom i přesnost výsledného klasifikátoru.

2.3 Klasifikátory trénované pomocí neuron. sítí

Umělá neuronová síť je paralelní systém, který se používá na modelování vztahu mezi vstupy a výstupy nebo na porovnávání pomocí vzoru. Funguje na způsob „černé skřínky“, kde minimální počet vstupů i výstupů je 1. Z názvu vyplývá, že je myšlenka principu fungování těchto klasifikátorů našla inspiraci v chování lidského mozku, což je přístup vhodný zejména tehdy, pokud je matematické řešení příliš složité. Jedinečná vlastnost těchto sítí je odolnost vůči poruchám. I když přestane fungovat několik neuronů, celková funkce sítě zůstane v podstatě nezměněna a může dál vykonávat (ač už s určitou nepřesností) svou činnost.

Činnost neuronové sítě je podle [10] charakterizována funkcí:

$$f: X \to Y \tag{2.4}$$

Síť se skládá z jednotlivých části zvaných *neurony* nebo *uzly*. Neuron přijímá N vstupů a vrací M výstupů. Definujeme tři typy neuronů:

- Vstupní – neurony, jejichž vstupy jsou signály z okolního prostředí.
- Výstupní – neurony, jejichž výstup vede do okolního prostředí.
- Skryté – spojovací mezičlánky mezi ostatními neurony.

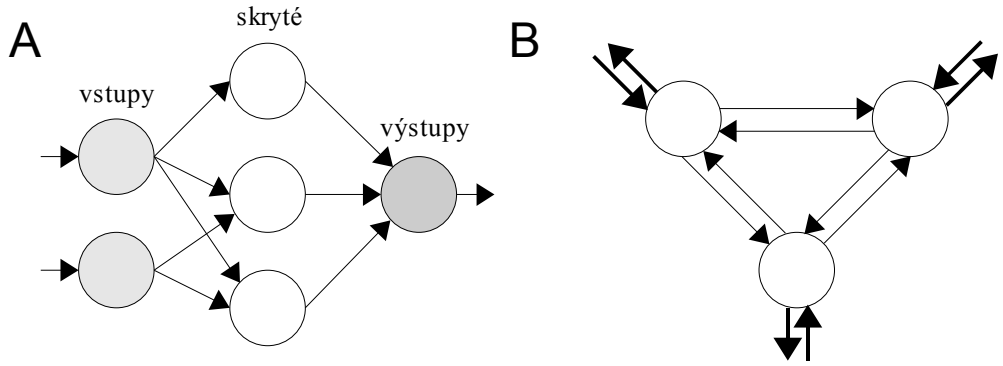
Je popsána celá řada modelů neuronů. Od těch velmi jednoduchých, používajících nespojité přenosové funkce, až po velmi složité, které popisující každý detail chování neuronu živého organismu. Jedním z nejužívanějších je model vytvořený McCullochem a Pittsem a popsaný v [12]:

$$Y = S\left(\sum_{i=1}^{N}(w_i x_i) - \Theta\right) \tag{2.5}$$

kde:
- Y je výstup neuronu
- x_i jsou vstupy neuronu
- w_i jsou synaptické váhy
- Θ je práh
- $S()$ je přenosová funkce neuronu (někdy aktivační funkce)

Samotná neuronová síť může být rozdělena do více vrstev, čehož se využívá při větší složitosti řešeného problému a vhodně tím rozdělíme simulaci do více skupin. V závislosti na toku dat je možné rozdělit neuronové sítě do dvou kategorií:

- **Dopředné** (*feed-forward*)– signál se šíří pouze od vstupních neuronů přes skryté k výstupním neuronům, příklad na obrázku 2.2 (A)
- **Rekurzivní** (*recurrent*)– signál se může pohybovat prakticky všemi směry, příklad na obrázku 2.2 (B)



Obrázek 2.2: A - dopředná neuronová síť s dvěma vstupy, třemi skrytými a jedním výstupním uzlem. Obsahuje 3 vrstvy. B - rekurentní neuronová síť, jejíž neurony jsou zároveň vstupními i výstupními. Neobsahuje skryté neurony a ani nejsou organizovány ve vrstvách.

Podstatnou vlastností neuronových sítí je schopnost učit se. Učení je důležitá fáze, kterou musí obsahovat každý proces tvorby sítě. V této fázi se upravují jednotlivé váhy spojení neuronů tak, aby se dosáhly požadované výsledky, nebo se k nim výstup aspoň co nejvíce přiblížil. Správným naučením se zabývá oblast umělé inteligence *strojové učení* (podrobnosti je možné nalézt v [11]). Využití neuronových sítí je mnohem širší než oblast rozpoznávání vzorů a klasifikace. V oblasti detekce tváře se využívá např. metoda zpětného šíření chyby (*back-propagation*). Příklad klasifikátoru vytvořeného pomocí neuronové sítě je v [5] popsán např. takto:

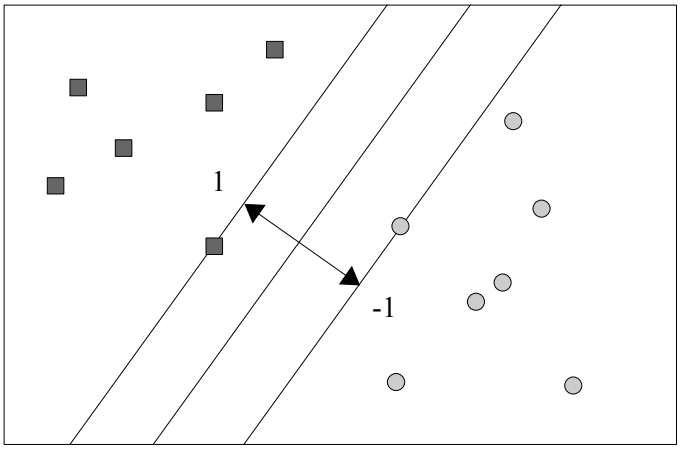
Obsahuje 3 typy skrytých uzlů: 4-krát typ, který zkoumá oblasti 10×10 pixelů, 16-krát typ, který zkoumá oblasti 5×5 pixelů a 6-krát typ, který zkoumá překrývající se 20×5 pixelové horizontální pruhy. Horizontální pruhy umožňují skrytým neuronům detekovat rysy jako jsou ústa nebo pár očí, zatímco skryté neurony čtvercového typu mohou detekovat rysy tváře jako jednotlivé oči, nos nebo koutky úst.

2.4 Klasifikátory trénované pomocí SVM

Support Vector Machine (dále jen SVM) je metoda klasifikace lineárních dat (při klasifikaci nelineárních dat je možné data transformovat do prostoru s vyšší dimenzí a klasifikovat je jako lineární). Velmi často se používá například v aplikacích pro vyhledávání obrazů. SVM metoda je postavená tak, aby mohla co nejlépe generalizovat informací při trénování. Samotné trénování je však časově i paměťově náročné.

Podstatou trénování s SVM je vytvoření hyperroviny, která ze stupních dat vytvoří 2 sady vektorů v n-rozměrném prostoru. Pro každou sadu se potom vytvoří takováto hyperrovina, která se znovu použije jako vstupní sada. Data, která není možné separovat lineárně mohou být nejprve transformována do prostoru s vyšší dimenzí pomocí jádra (*kernel*).

Vysvětlení SVM klasifikátoru se nachází v [6]. V tomto dokumentu je popsán jednoduchý příklad, kde se nachází 2 spojené množiny, které jsou lineárně oddělitelné. Cílem SVM je ve vstupní sadě $D=\{(X_i, y_i)\}_{i=1}^{n}$, kde $y \in (-1, 1)$, nalézt optimální lineární řešení na základě minimalizace rizika. Řešením je hyperrovina, která nechává mezi těmito třídami co největší okraj. Okraj roviny je definován jako součet vzdáleností hyperroviny od nejbližšího bodu obou tříd. Ukázka fungování SVM je na obrázku 2.3. Objekty, které byly ohodnoceny -1 patří do jedné skupiny, zatímco objekty v té druhé mají SVM klasifikátorem danou výstupní hodnotu 1. Nevýhodou SVM je, že parametry je nutné pro konkrétní úlohu volit ručně.



Obrázek 2.3: Ukázka maximalizované vzdálenosti při dělení dat do dvou tříd

2.5 Klasifikátory trénované pomocí AdaBoost

Název metody je zkrácením Adaptive Boosting. Adaptabilní se nazývá z důvodu využívání více slabých klasifikátorů a schopnosti při učení využívat množinu předešle špatně zařazených objektů na natrénování slabých klasifikátorů. AdaBoost algoritmus pracuje iterativním způsobem. V každé iteraci mění důležitost (váhy) jednotlivých vstupních vzorků a vybírá ze všech jednoduchých klasifikátorů ty, které vykazují nejlepší výsledky a vzájemně se vhodně doplňují. Jeho výsledkem je jeden silný klasifikátor, sestavený jako vážená kombinace vybraných jednoduchých klasifikátorů.

Vytváříme tedy silný klasifikátor *H(x)* pro klasifikaci dat x_i jako kombinací slabých klasifikátorů $h(x_i)$, které též můžeme chápat jako příznaky.

$$h(x_i) \in \{-1, 1\} \tag{2.6}$$

$$H(x) = sign \sum_{i=1}^{N} \alpha_i h(x_i) \tag{2.7}$$

kde α_i je váha, kterou nastavujeme trénováním. Při detekci objektů v obraze AdaBoost rozhoduje, jak vybrat z filtrů. Vstupem jsou příklady objektů a ne-objektů (trénovací data). Algoritmus je uveden níže. Konstrukce silného klasifikátoru *f(x)* tvořeného lineární kombinací

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{2.8}$$

jednotlivých slabých klasifikátorů $h_t(x)$.

Vstup: sekvence vstupů $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$, *kde* $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Inicializace: $D_1(i) = \frac{1}{m}$, *kde* $i = 1, \ldots, m$

Pro: *t = 1..T*

1. spočti chybu pro εt pro každý filtr

$$\varepsilon_t = \sum_i w_{t,i} [h_t(x_i) \neq y_i] \tag{2.9}$$

2. vyber filtr s nejmenší chybou
3. nastav α_t

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right) \tag{2.10}$$

4. převaž příklady (boosting), aby špatně zařazená data měla větší váhu

$$w_{t+1,i} = w_{t,i} e^{-\alpha_t y_i h_t(x_i)} \tag{2.11}$$

5. přidej slabý filtr s váhou αt

Výstup: váhové ohodnocení slabých klasifikátorů

$$H(x) = sign \sum_{i=1}^{N} \alpha_i h(x_i) \tag{2.12}$$

2.6 WaldBoost

Nevýhodou AdaBoostu je, že není možné rozhodnout, zda daná prohledávaná oblast patří nebo nepatří do určité skupiny dříve, než se propočítají hodnoty ze všech slabých klasifikátorů, i když je výsledek zřejmý již během propočtu jen jejich určité části. Na řešení tohoto nedostatku vznikla modifikace AdaBoostu s názvem WaldBoost. Klasifikátor natrénovaný pomocí WaldBoostu nemusí být vyhodnocen celý. V případě, že se během vyhodnocování slabým klasifikátorem překročí určitá spodní nebo horní hranice, oblast je zařazena do dané skupiny. V opačném případě se oblast předá následujícímu klasifikátoru. Tato výhoda umožňuje použití klasifikátoru v real-time aplikacích, od kterých požadujeme okamžité zpracování dat bez zpoždění.

2.7 Haarovy příznaky

Při klasifikaci obrazu můžeme použít několik přístupů. Jedním z nich je vykonávat klasifikaci přímo na základě hodnot intenzit klasifikovaného obrazu. Jiným přístupem je klasifikace pomocí příznaků. Výhoda spočívá v tom, že příznaky v sobě obsahují informaci o charakteru vybrané oblastí obrazu. Pakliže se tato informace dá vyjádřit jednoduchým způsobem, dostáváme velmi rychlé metody.

Haarovy příznaky jsou jednoduché obdélníkové oblasti (obrázek 2.8), jejichž hodnota se vypočítává z intenzity obrazu pod danou plochou. Výpočet příznaku se provádí tak, že se od sumy bílých oblastí odečte suma těch černých:

$$f(x) = \sum_{w \in W} x(w) - \sum_{b \in B} x(b) \tag{2.13}$$

kde pixel o intenzitě x nacházející se v oblasti zkoumaného Haarova příznaku patří do skupiny W v případě, že se nachází v bílé oblasti. V opačném případě, leží-li v černé oblasti, patří do skupiny B. Haarovy příznaky jsou založeny na sumách obdélníkových oblastí. Tento výpočet můžeme vykonávat prostým sčítáním, které je citlivé na velikost sčítané oblasti, nebo můžeme na výpočet použít tzv. integrální obraz.

Haarovy příznaky nejsou jediné, které lze použít při trénování AdaBoost či WaldBoost. Existuje celá řada příznaků, například spojitých – Gáborovy vlnky nebo diskrétních – *Local Binary Patterns*, *Local Range Differences* (popsány v 2.12), a další.

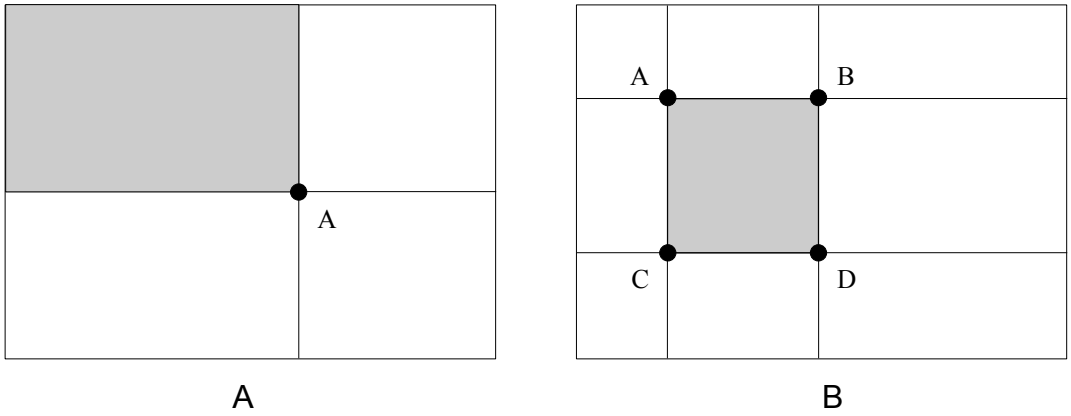
2.8 Integrální obraz

Integrální obraz může výrazně urychlit sumarizaci obdélníkových oblastí. Jeho hlavní funkcí je v konstantním čase vypočítat libovolně velký a jakkoliv umístěný obdélníkový výřez obrazu. Vlastní hodnoty integrálního obrazu II jsou součtem hodnot intenzit pixelů obrazu I nalevo a nahoru od pozice pixelu v obraze (obrázek 2.4 A). Z toho vyplývá, že pozice pro výpočet umístěná v pravém dolním rohu obrazu nám je součtem intenzit v celém obrazu, zatímco první sloupec a řádek mají hodnotu 0.

$$II(x, y) = \sum_{i,j}^{x,y} I(i, j) \tag{2.14}$$

Pokud tedy známe pozici rohových bodů obdélníku, značených jako **A, B, C, D**, můžeme pomocí jednoduché rovnice 2.15 vypočítat intenzitu obdélníkové oblasti (obrázek 2.4 B).

$$I_{obd} = A - B - C + D \tag{2.15}$$



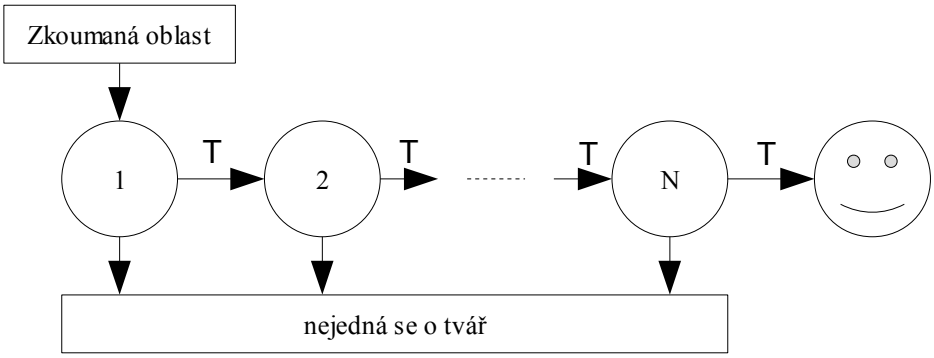
Obrázek 2.4: Integrální obraz: A) Hodnota bodu A představuje šedou oblast. B) Oblast vypočtená rovnicí I_{obd}

2.9 Kaskádové zapojení

Ve snaze snížení chybovosti a zvýšení efektivity se začalo využívat kaskádového zapojení.

Stupně kaskády jsou natrénovány pomocí AdaBoostu, přičemž každý stupeň je schopen rychle rozhodnout zda-li je daný výřez obrazu hledaný objekt či ne. Tyto klasifikátory nemusí dosahovat vysoké bezchybnosti. Vstupní výřez je nejdříve zkoumán jedním klasifikátorem a pokud jej ten označí za oblast, kde se objekt nenachází, se tato hodnota vrátí. V opačném případě se předá dalšímu stupni kaskády – dalšímu klasifikátoru. Pokud se zkoumaná oblast dostane až k poslednímu klasifikátoru, můžeme rozhodnout, zda se v daném výřezu opravdu nachází to, co hledáme. Postup je schematicky znázorněn na obrázku 2.5.

Výhodou tohoto přístupu je vyřazení oblastí, které neobsahují objekt již na začátku, kdy mohou být klasifikátory výrazně jednodušší a rychlejší, než klasifikátory na konci, které mají již podstatně větší složitost. Právě díky tomu se šetří čas potřebný na detekci.



Obrázek 2.5: Schematické zobrazení kaskádového zapojení

2.10 Kalmanův filtr

Kalmanův filtr je soubor matematických rovnic, který poskytuje efektivní výpočetní (rekurzivní) prostředky pro odhad stavu systému, a to způsobem, který minimalizuje průměr kvadratické chyby. Tento filtr je velmi silný v několika ohledech – umožňuje odhady minulosti, přítomnosti i budoucnosti a jeho schopen tak činit dokonce i v případě, kdy neznáme přesnou povahu daného modelovaného systému.

Kalmanův filtr se zabývá obecným problémem snahy odhadnout stav procesu běžícího v diskrétním čase, který se řídí lineární stochastickou diferenciální rovnicí

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.16}$$

s měřitelným výstupem systému $z \in \Re^m$, který je

$$z_k = Hx_k + v_k \tag{2.17}$$

Náhodné proměnné w_k a v_k představují stavový bílý šum v rovnici měření. Jsou na sobě nezávislé a s normálním rozložením pravděpodobnosti:

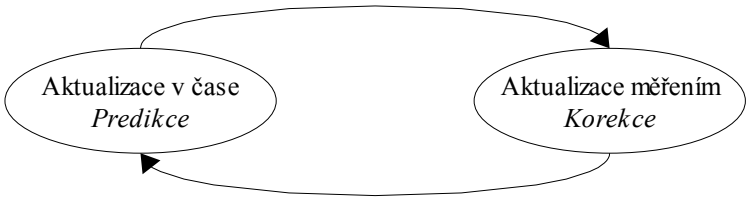
$$\begin{aligned} p(w) &\sim N(0, Q) \\ p(v) &\sim N(0, R) \end{aligned} \tag{2.18}$$

V praxi to znamená, že matice kovariance stavového a měřeného šumu se mohou měnit s každým časovým krokem nebo měřením, nicméně zde se předpokládá, že jsou konstantní.

Matice A $n \times n$ v diferenciální rovnici v úvodu této kapitoly představuje vztah mezi předešlým $k-1$ a aktuálním časovým krokem k, při neuvažování řídící funkce nebo procesního šumu. V praxi je třeba si uvědomit, že se matice A může měnit v každém kroku, zde ji ale považujeme za konstantní. Matice B $n \times 1$ představuje volitelný kontrolní vstup $u \in \Re^l$ systému ve stavu x. Matice H $m \times n$ rovnice měření se opět v praxi může měnit v každém kroku, pokládáme ji ale za konstantní.

Kalmanův filtr předvídá svůj stav za užití zpětné vazby. Filtr odhadne stav procesu v určitém čase a následně obdrží zpětnou vazbu v podobě měření (se šumem). Z tohoto důvodu rozdělujeme rovnice Kalmanova filtru do dvou hlavních skupin: *aktualizační časové* a *aktualizační měřící*. Časové rovnice spolu s kovariancí chyb jsou zodpovědné za promítání současného stavu vpřed po časové ose a apriorní odhad dalšího kroku. Měřící naopak zodpovídají za zpětnou vazbu, tedy za zavedení nového měření do apriorního odhadu, čímž získáme již korigovaný odhad.

Tyto dvě skupiny rovnic můžeme označit též jako *predikční* a *korekční* a jejich vzájemná interakce v algoritmu pro řešení numerických problémů je znázorněna na obrázku 2.6.



Obrázek 2.6: Cyklus diskrétního Kalmanova filtru. Predikce odhadne změnu aktuálního stavu v dalším časovém kroku. Korekce tento projektovaný odhad zkoriguje pomocí vlastních hodnot naměřených v daném čase.

Další a podrobnější popis Kalmanova filtru, který by byl již nad rámec této bakalářské práce, lze nalézt v [8].

2.11 OpenCV

OpenCV (Open Source Computer Vision) je multiplatformní knihovna funkcí orientovaných na počítačové vidění, původně vyvinutá společností Intel. Dokáže pracovat na systémech Windows, Linux i MacOS. Knihovna je zaměřená zejména na zpracování obrazu v reálném čase a obsahuje k tomuto účelu pestrou škálu funkcí.

V programové části této bakalářské práce je využívána zejména na zachytávání videa, jednoduchou práci s obrazem (anonymizační efekty) a zobrazování výsledků. Větší prostor dostala v doplňkových funkcích pro predikci pohybu obličeje v obraze pomocí Kalmanova filtru a zpracování Haar-kaskád při detekci očí v obličejové části.

2.12 UPGM detektor a OpenCV detektor

Tato bakalářská práce staví velkou měrou na obličejovém klasifikátoru, který byl vyvinut v Ústavu počítačové grafiky a multimédií FIT VUT v Brně (dále jen UPGM). Ten byl natrénován pomocí knihovny využívající metody WaldBoost a tzv. *Local Range Difference* (dále LDR). Tuto knihovnu je možné stáhnout na stránkách http://www.fit.vutbr.cz/research/prod/index.php?id=43&notitle=1 a její zevrubný popis zní:

> Softwarová knihovna napsaná v C/C++, která provádí efektivní dvoutřídní klasifikaci pomocí klasifikátorů trénovaných metodou WaldBoost založených na Haarových příznacích. Klasifikátor je uložen jako XML soubor, který definuje pozice a parametry jednotlivých obrazových příznaků, které je potřeba vyhodnotit pro výpočet odezvy klasifikátoru. Knihovna nabízí klasifikaci jednotlivých obrázků a také skenování obrazu pro získání odezvy klasifikátoru na všechna podokna obrazu. Dále knihovna nabízí generování jednoduché pyramidální struktury obrazu a tedy podporu detekce různě velikých objektů.

Local Rank Differences

Tato podkapitola je převzata z [1]. Předpokládejme skalární obraz $I(x, y) \rightarrow \boldsymbol{R}$. V takovémto obraze můžeme definovat následující vzorkovací funkci

$$S_{xy}^{mn} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x + m(u-1) + i, y + n(v-1) + j) \tag{2.19}$$

Tato funkce je parametrizována vzorkovacím blokem o rozměrech m, n a pozici vzorku (x, y), což je pixel v obraze. Na základě této funkce lze definovat čtvercovou masku

$$M_{xy}^{mnwh}=\begin{bmatrix} S_{xy}^{mn}(1,1) & S_{xy}^{mn}(2,1) & \cdots & S_{xy}^{mn}(w,1) \\ S_{xy}^{mn}(1,2) & S_{xy}^{mn}(2,2) & \cdots & S_{xy}^{mn}(w,2) \\ \vdots & \vdots & \ddots & \vdots \\ S_{xy}^{mn}(1,h) & S_{xy}^{mn}(2,h) & \cdots & S_{xy}^{mn}(w,h) \end{bmatrix} \tag{2.20}$$

Maska má rozměry *w*, *h* a je parametrizována stejně jako vzorkovací funkce *S*, tedy rozměry $m \times n$ a pixelem v obraze o souřadnicích *x*, *y*. Dle [2] experimenty prokázaly, že vzhledem k detekci metodami AdaBoost a WaldBoost, jsou zcela dostačující masky o rozměrech 3×3 ($w = 3$, $h = 3$). Pro různé klasifikátory je potřeba bloků různých rozměrů: pro obličejový detektor operující nad čtvercovými obrazovými okny o rozměrech strany 24 pixelů, jsou vhodné vzorkovací rozměry 1×1, 2×2, 2×4, 4×2.

Pro každou pozici v masce může být definován *rank*

$$R_{xy}^{mnwh}(u,v)=\sum_{i=1}^{w}\sum_{j=1}^{h}\begin{matrix} 1, když\ S_{xy}^{mn}(i,j)<S_{xy}^{mn}(u,v) \\ 0, ostatní \end{matrix} \tag{2.21}$$

to jest, *rank* je pořadím daného členu masky v její postupně seřazené množině. Tato hodnota je nezávislá na lokální intenzitě obrazu, což je důležitá vlastnost pro chování LRD, které definujeme:

$$LRD_{xy}^{mnwh}(u,v,k,l)=R_{xy}^{mnwh}(u,v)-R_{xy}^{mnwh}(k,l) \tag{2.22}$$

Tento zápis můžeme zčásti zjednodušit vektorizací matice *M* – zřetězením jejích řádku (je pouze nepsanou konvencí, že se řetězí řádky a ne sloupce):

$$V_{xy}^{mnwh}=[\,S_{xy}^{mn}(1,1) \quad S_{xy}^{mn}(2,1) \quad \cdots \quad S_{xy}^{mn}(w,h)\,] \tag{2.23}$$

Pozice členu ve vektoru je tedy dána ($V_{xy}^{mnwh}(i)$ značí *i*-tý člen vektoru):

$$R_{xy}^{mnwh}(a)=\sum_{i=1}^{w\times h}\begin{matrix} 1, když\ V_{xy}^{mnwh}(i)<V_{xy}^{mnwh}(a) \\ 0, ostatní \end{matrix} \tag{2.24}$$

LDR dvou pozic *a*, *b* v rámci vektoru je pak:

$$LRD_{xy}^{mnwh}(a,b)=R_{xy}^{mnwh}(a)-R_{xy}^{mnwh}(b) \tag{2.25}$$

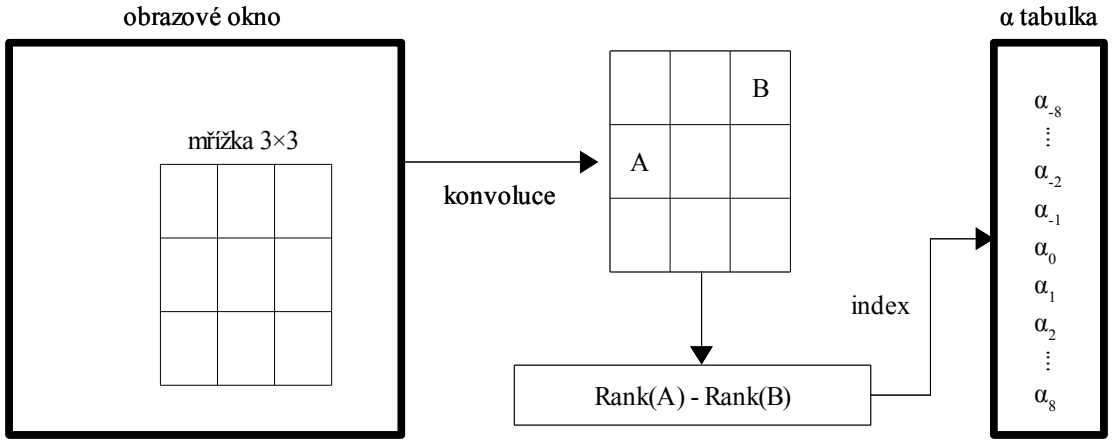
Pro zvýšení efektivity LDR hodnocení můžeme předpočítat funkci S_{xy}^{mn} definovanou na vstupním obraze. Jak již bylo řečeno, pro natrénování objektového klasifikátoru je třeba pouze malého počtu kombinací $m \times n$. Vstupní obraz může být konvoluován pomocí

$$\begin{aligned} h_{2\times 2}&=\begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix} \\ h_{4\times 2}&=\begin{bmatrix} 1/8 & 1/8 & 1/8 & 1/8 \\ 1/8 & 1/8 & 1/8 & 1/8 \end{bmatrix} \\ h_{w\times h}&=\begin{bmatrix} 1/wh & \cdots & 1/wh \\ \vdots & \ddots & \vdots \\ 1/wh & \cdots & 1/wh \end{bmatrix} \end{aligned} \tag{2.26}$$

díky čemuž bude výsledný obraz na dané pozici (x, y) obsahovat hodnoty vzorkovací funkce. Takové předpočítání lze provést velice efektivně pomocí paralelismu (platformy MMX, GPU, FPGA) a LRD

vyhodnocení se potom sestává pouze z 9 (v případě 3×3 LRD masky) nahlédnutí do vhodně předpřipraveného obrazu a poté vyhodnocení dvou prvků masky.

Na obrázku 2.7 vidíme zjednodušený postup při vyhodnocování jednoho LRD klasifikátoru. Detekce probíhá v obrazovém okně (o rozměrech např. 30×30 pixelů), na které umisťujeme pravoúhlou masku M_{xy}^{mn33} (v tomto případě 3×3). Každé políčko masky zabírá několik pixelů, které je třeba konvoluovat.



Obrázek 2.7: Použití LDR v klasifikátoru

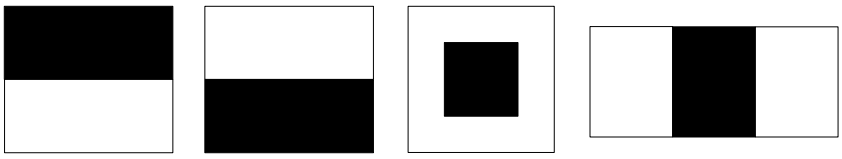
Následně jsou vyhodnoceny *ranky* a jejich rozdíl je použit jako index v alfa tabulce, která rozhodne o výsledku daného příznaku. Více se o LRD i jeho implementaci na platformě MMX můžeme dočíst např. v [1].

Jelikož LRD využívá WaldBoost, volíme i určitou hranici (*threshold*), při jejímž překročení je daná oblast zařazena do vybrané skupiny. V opačném případě je předána následujícímu klasifikátoru.

OpenCV a Viola-Jones

Detektor v knihovně OpenCV je naproti tomu postaven na detekci Viola-Jones. Konkrétně jde tedy o klasifikátor využívající Haarovy příznaky zapojené v kaskádě, který pro zrychlení detekce příznaků pracuje nad integrálním obrazem. Klasifikátor je natrénován pomocí několika stovek pozitivních (objekt se v obraze nachází) i negativních (objekt v obraze není) příkladů, které jsou zpracovány pomocí robustní metody AdaBoost.

Haarovy příznaky (2.7), jejich kaskádové zapojení (2.9), integrální obraz (2.8) i metoda AdaBoost (2.5) již byla popsána výše. Metoda Viola-Jones tedy používá sérii slabých klasifikátorů zapojených v řetězci (obrázek 2.5), přičemž každý klasifikátor představuje jeden Haarův příznak. Příklady těch použitých v OpenCV vidíme na obrázku 2.8.



Obrázek 2.8: Příklady Haarových příznaků použitých v OpenCV.

Způsob použití je zřejmý z obrázku 2.9, který zachycuje první dva příznaky ve Viola-Jones kaskádě. Vlevo vidíme vstupní fotografii lidské tváře. Prostřední zachycuje horizontální Haarův příznak, kde dojde ke shodě díky tomu, že lícní část obličeje je ve většině případů světlejší než oči. Vpravo je zase světlejší kořen nosu. Vyhodnocení takovýchto dvou situací by úspěšně posunulo klasifikaci v kaskádovém zapojení blíže k pozitivnímu vyhodnocení a detekci obličeje v obraze.

Obrázek 2.9: Originální snímek a první dva Haarovy příznaky používané metodou Viola-Jones

Detektor knihovny OpenCV poskytuje také několik ladících parametrů, díky nimž je možné omezit počet falešných detekcí nebo použít zprůměrovanou hranici pro oblast, v níž bylo učiněno několik detekcí velmi blízko sebe a tyto detekce se překrývají. Jelikož je OpenCV klasifikátor použit v aplikaci na detekci očí v obličejové části, jsou tyto vlastnosti podrobněji popsány v druhé části práce v 3.4.

3 Implementace programu

3.1 Použité klasifikátory

Při implementaci programu jsem použil dva klasifikátory:

- **LRD klasifikátor** (poskytnutý UPGM) – Natrénován pro detekci obličejů čelem ke kameře, s výsledky trénování uloženými v souboru `faces.h`.
- **OpenCV klasifikátor** – Použil jsem již hotový XML soubor `haarcascadeeye.xml`, který byl vytvořen pomocí volně přístupných databází snímků obličejů. Seznam těchto databází je uveden v příloze.

3.2 Vyhledání a ověření obličeje

Protože cílem aplikace bylo umožnit sledovat teoreticky neomezené množství obličejů v obraze, bylo nutné zvolit i vhodný způsob, jak tyto obličeje v aplikaci udržovat a efektivně aktualizovat jejich polohu a velikost.

Logickým důsledkem bylo tedy vytvořit jednoduchý objektový model, kde každý obličej v obraze vytvoří vlastní instanci a jejich množina bude uchovávána ve vhodném kontejneru. Dalším faktorem, který jsem chtěl ovlivnit byly falešné a nevhodné detekce a chování toho, která detekce má ovlivnit jaký obličej. Navrhl jsem proto třídu `Face`, která udržuje základní atributy jako jsou souřadnice, velikost obličeje a dále i důležité statické parametry, které do značné míry ovlivňují výsledné chování aplikace:

- `AGE_LIMIT` – Počet snímků, po které musí být v dané oblasti detekován klasifikátorem obličej, aby došlo k jeho potvrzení a následnému zobrazení ve výstupním obraze.
- `TTL_NEW` – *Time To Live* pro nově detekovaný obličej, tedy počet snímku, po nichž jsou uchovávány informace o novém potencionálním obličeji v obraze a čeká se na naplnění `AGE_LIMIT`, aby se obličej potvrdil a zobrazil.
- `TTL_CONFIRMED` – Počet snímku, po nichž se již potvrzený obličej zobrazuje v obraze a čeká se, než bude definitivně odstraněn, pokud v jeho oblasti nedojde k žádné nové detekci; tato hodnota je dle nastavení několikanásobně větší než `TTL_NEW`.

Třída poskytuje funkce `getRect()`, `getCenter()`, `isConfirmed()`, jejichž účel je zřejmý z jejich názvu. Dále `isActive()`, která odečítá nastavený TTL a zároveň sděluje, zda již životnost objektu nevypršela. Funkce `isConfirmed()`, která vrací pravdivostní hodnotu, zda je již obličej potvrzen a má se vykreslit a nejdůležitější funkci `updateFace()`, která aktualizuje informace o poloze a rozměrech objektu.

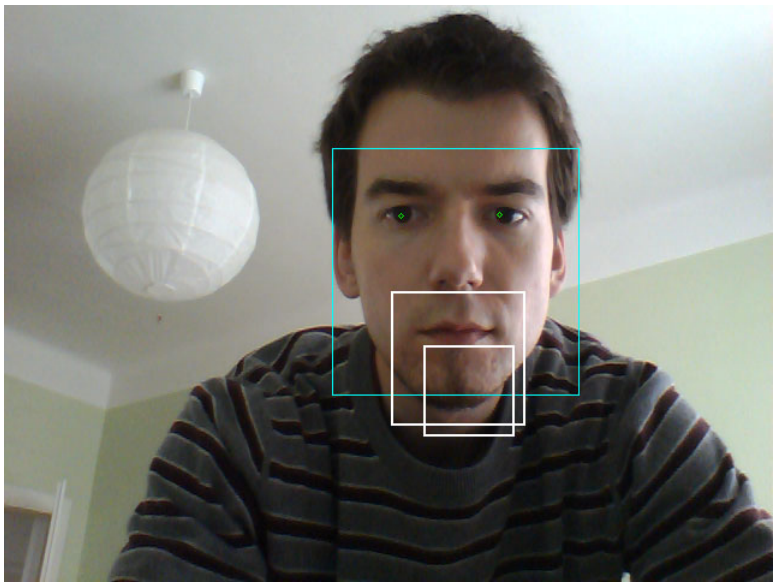
Nové detekce jsou získány od LRD klasifikátoru ve formě struktur typu `CvRect`, což je interpretace čtyřúhelníku v OpenCV. V případě nových a dosud neregistrovaných detekcí je vytvořena nová instance třídy `Face` a uložena do C++ kontejneru typu `vector`. Ostatní detekce jsou roztříděny mezi již registrované obličeje a dále profiltrovány, jak je popsáno v následující kapitole.

Po každém kompletním průchodu klasifikátoru snímkem je navíc všem instancím odečtena jedna jednotka TTL, jsou odstraněny zaniklé instance a případně obličeje, které se příliš překrývají a jeden z nich může být tedy bez závady na případné anonymizaci zrušen.

3.3 Sledování obličeje

Jak jsem zmínil výše, problémem při dosažení co nejlepších výsledků sledování a anonymizace obličeje byly falešné nebo značně vychýlené detekce. Jednou z možností by bylo takovouto situaci řešit požadavkem na minimální počet sousedících detekcí. Byl by nastaven práh, kolik detekcí se musí minimálně nacházet v blízkém shluku, aby byla jejich skupina spojena a ohraničena v jedné detekci, přičemž ostatní shluky by byly zahozeny. Tento způsob používá např. detektor v OpenCV, jak je popsáno níže v 3.4. Já jsem se rozhodl zvolit odlišný přístup a využít několik filtrů na roztřídění detekcí mezi jednotlivé obličeje a zahození těch nevhodných.

Již v samotné hlavní smyčce programu jsem vložil první filtr podle poměrných vzdáleností, který pošle na aktualizaci jednotlivým obličejům jen takové detekce, které svůj střed mají vzdálen maximálně do jedna násobku rozměrů dané instance třídy `Face`. Tím je zajištěno to, že detekce ovlivňují pouze obličeje ve svém bezprostředním okolí. Uvnitř funkce `updateFace()` je dále důležitý mediánový filtr, který si pamatuje poslední předané rozměry a u potvrzených obličejů odfiltruje ty, které by rozměr změnily o více jak jednu osminu vůči vypočtenému mediánu. Tak nedochází k příliš velké skokové změně velikosti, což nebyl zas takový problém při celoobličejovém rozmazání, ale často způsoboval vychýlení černé pásky z oblastí očí. Postupně jsem přidal i podobný filtr pro velké skokové změny polohy. Problémové detekce vznikaly nejčastěji v oblasti dolní části obličeje a úst. Příklad je vidět na obrázku 3.1. Do budoucna by bylo vhodné uvažovat o přístupu nahrazování všech překrývajících se detekcí jednou společnou zprůměrovanou hranicí, což se mi ke konci testování jevilo jako pravděpodobně lepší přístup, než filtrovat detekce nevhodné.



Obrázek 3.1: Příklad typických nevhodných detekcí, které jsou v aplikaci odfiltrovány

Po nalezení a potvrzení obličeje v obraze je výhodné jeho polohu i pohyb upravovat jistým korelačním prostředkem, čímž můžeme částečně eliminovat některé výchylky od jeho reálné polohy a lépe sledovat jeho pohyb. Tímto prostředkem je v aplikaci Kalmanův filtr (2.10), který je nasazen pro sledování polohy středu obličeje.

Programová třída obličeje `Face` proto obsahuje 3 důležité struktury (v prvním případě ukazatel na ni): `CvKalman`, což je samotný Kalmanův filtr a dále 2 matice typu `CvMat`, v kterých se uchovává měření z aktuální detekce a korigovaný výsledek, který přímo určuje výsledné zobrazení. Pro jednoduchost je Kalmanův filtr vypočítáván pro matematický střed obličeje, třída tedy obsahuje bod `CvPoint center`.

Při instanciaci objektu třídy `Face` je volána vlastní funkce `initKalman()`, která inicializuje Kalmanův filtr a jeho potřebné vnitřní struktury. Pří každém volání funkce `updateFace()` je poloha středu obličeje predikována pomocí funkce `cvKalmanPredict()` a následně korigována reálným měřením uloženým v matici `measurement` a funkcí `cvKalmanCorrect()`. Veškerou funkcionalitu týkající se Kalmanova filtru poskytuje knihovna OpenCV.

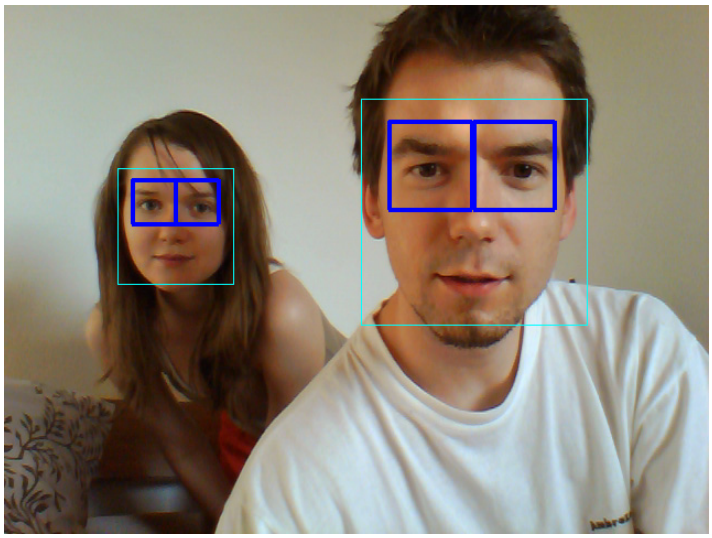
3.4 Vyhledání očí v obraze

Jelikož program má umět i anonymizaci pomocí překrytí očí černou páskou, bylo nedílnou součástí realizace i spolehlivá detekce očí v nalezeném obličeji. Mým záměrem bylo minimalizovat nezbytně nutné zatížení počítače, aby vyhledávání očí zbytečně nebrzdilo celou aplikaci a zároveň umožnit spolehlivé fungování anonymizace i za předpokladu, že detekce skončí neúspěchem.

Klíčem k tomuto cíli je vymezit prohledávanou oblast tak, aby zabírala co nejpravděpodobnější místo, v jehož středu by se mělo nalézat oko – za předpokladu, že se klasifikace provádí pro obě oči zvlášť a na obličeji natočeném ke kameře čelem. Vzhledem k pozorování chování již implementovaného detektoru, prozkoumání několika sad fotek lidských tváří a následnému ladění programu, jsem zvolil pro levé oko oblast 3/8 šířky obličeje × 4/10 výšky obličeje, posunutou o 1/8 šířky od levého okraje tváře (vzhledem k pozorovateli aplikace) a 1/10 výšky od horního okraje. Pro pravé oko se jedná o tutéž oblast tentokráte s rozdílem posunutí od pravého okraje. Názorně jsou oblasti vyznačeny v obrázku 3.2.

Tímto způsobem je dosaženo výsledku, že pokud se oko nepodaří spolehlivě detekovat např. kvůli problematickému nasvícení jedné strany obličeje (může být příliš zastíněna), dojde k automatickému umístění nejpravděpodobnější pozice oka do středu prohledávané oblasti. Za každých okolností tedy máme v obličeji 2 body, které můžeme spojit černou páskou. Buď se jedná o přímou detekci nebo o odhad umístění.

Detekce očí je realizována pomocí knihovny OpenCV a jejího klasifikátoru, který je založen na metodě Viola-Jones, jak je již zmíněno v (2.12). Detekci i případnou anonymizaci očí má v aplikaci na starost funkce `detectEyes()`, jejímiž nejdůležitějšími parametry jsou vstupní obraz, oblast detekovaného obličeje a kaskáda Haarových příznaků. Funkce se zaměří na již dříve zmíněnou oblast s pravděpodobným výskytem oka, nastaví tedy odpovídající *range of interest* (oblast zájmu, dále jen ROI). ROI oblast se pomocí funkce knihovny OpenCV `cvHaarDetectObjects()` prozkoumá a z případného pozitivního výsledku se uchová jeho střed v bodu `leftCenter` případně `rightCenter`, neboť klasifikace probíhá zvlášť pro levé i pravé oko.



Obrázek 3.2: Oblasti, na které je omezeno detekování očí pro celkové urychlení výpočtu

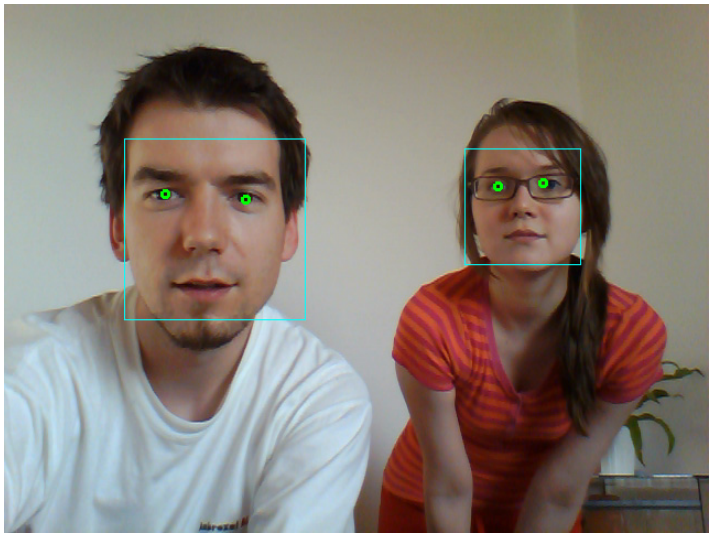
Jednou z věcí, se kterou je třeba počítat je, že detektor po svém zavolání vyprodukuje velké množství pozitivních nálezu (ohraničených čtverců), které prošly Haar-kaskádou. Oblast těsně kolem samotného oka generuje jejich největší množství. Ty se z větší míry vzájemně překrývají. Osamocené detekce nejčastěji představují falešné nálezy, takže je vhodné je zahodit. Naopak překrývající se detekce se spojí do jedné, která ohraničuje celý jejich region. Knihovna OpenCV dělá obě tyto věci ještě před vrácením seznamů finálních detekcí.

Mezi osamocenými detekcemi a jejich velkými seskupeními ještě existují shluky několika detekcí, kterou mohou či nemusí být ve skutečnosti okem. Práh pro vyhodnocení této skutečnosti představuje určitý počet překrývajících se sousedů a nastavuje se parametrem `min_neighbors`, jehož výchozí hodnota je 3. Všechna seskupení s počtem detekcí pod tento nastavený limit jsou zahozena. Kdybychom nastavili parametr na 0, funkce vrátí kompletní seznam nefiltrovaných detekcí získaných Haar-kaskádou. V mém případě se mi osvědčily hodnoty v rozmezí 6-8.

Pro zajímavost čtvrtý parametr funkce `cvHaarDetectObjects()` specifikuje, jak rychle má OpenCV škálovat velikost zkoumané oblasti obrazu s každým dalším průchodem. Výchozí hodnotou je 1.1, což představuje zvětšení o 10% v každém průchodu. Šestý parametr se nastavuje buď na 0 nebo na `CV_HAAR_DO_CANNY_PRUNING`, který způsobí, že se obraz nejprve protáhne hranovým detektorem a tím se vyřadí oblasti, které s největší pravděpodobností nemohou obsahovat hledaný objekt. Klasifikace je tak méně výpočetně náročná, ovšem existuje riziko, že se nepodaří zachytit některé objekty. Z tohoto důvodu, a protože je oblast stejně relativně velmi malá, tuto možnost nevyužívám. Konečně posledním a sedmým parametrem funkce je nejmenší možná velikost objektu, který hledáme. Tento rozměr můžeme nastavit buď sami, nebo ponechat rozměr 0×0, což způsobí použití výchozích hodnot z XML souboru ohraničených značkou `<size>`.

Jádrem klasifikátoru je samozřejmě Haar-kaskáda uložená v XML souboru, který je načten v aplikaci. Dále je třeba zajistit paměťový buffer, do kterého detektor bude ukládat nalezené výsledky, a který se bude automaticky dle potřeby rozšiřovat. K tomuto účelu knihovna OpenCV poskytuje vhodný objekt `CvMemStorage`.

Po proběhnuté klasifikaci nám zůstanou vyplněné dva body představující středy očí, stačí je tedy již jen buď označit v obraze nebo rovnou anonymizovat v závislosti na odpovídajícím parametru funkce `detectEyes()`.



Obrázek 3.3: Oči detekované v obličejové části

3.5 Anonymizace

Dle zadání má program umět celkové rozmazání obličeje a překrytí očí černým proužkem. Způsobů, jak spolehlivě znemožnit identifikaci celého obličeje, je celá řada. Z televizních obrazovek je určitě nejznámější "rozkostičkování" a dále různé černé geometrické útvary přes hlavu.

Nejdříve jsem uvažoval o Gaussově rozostření, ale z důvodu subjektivně lepších výsledků jsem nakonec zvolil dilatační filtr, který poskytuje funkce `cvDilate()`. Několik průchodu tohoto filtru spolehlivě rozmaže obraz a znemožní identifikaci člověka. Výsledek rozmazání je vidět v obrázku 3.4. Černá páska přes oči se dá velice snadno realizovat díky jednoduché primitivě z knihovny OpenCV – dostatečně silnou úsečkou vykreslenou pomocí funkce `cvLine()` – a spojuje dva body získané funkcí `detectEyes()`. Tloušťka čáry se mění proporčně vůči velikosti obličeje (aktuálně nastavena na jeho jednu čtvrtinu), čímž se vhodně přizpůsobuje a zakrývá pouze tu část, kde se nachází oči. Příklad je v obrázku 3.5.

Obrázek 3.4: Rozmazání celého obličeje dilatačním filtrem

Obrázek 3.5: Příklad anonymizace černou páskou, která spojuje detekce očí v obličeji a přizpůsobuje svou šíři jeho velikosti

Při samotné anonymizaci jsem se setkal s celou řadou problémů, které jsem se snažil postupně řešit a odstraňovat. Co se týče samotné anonymizace, ta funguje spolehlivě pro relativně neomezené množství obličejů (ovlivněno samozřejmě šířkou záběru kamery, jejich velikostí a vzájemnou vzdá-

leností). Podmínkou je ovšem jejich frontální natočení vůči snímači, aby aplikace mohla získávat detekce od LRD klasifikátoru, který detekuje tento typ tváří. Problémem není chvilkové natočení snímané osoby, neboť detekované obličeje setrvávají ještě nějakou dobu na daném místě do vypršení TTL. Takovýto obličej ale již není možné bohužel danými prostředky sledovat a vhodný postup by představoval kombinaci několika přístupů. Možné řešení jsem proto popsal a představil v závěru této práce. Množství předávaných detekcí od obličejového klasifikátoru je ovlivněno nastaveným prahem (*threshold*) – viz 2.6 a 2.12, který lze měnit za běhu aplikace a tím do značné míry i pozitivně ovlivnit výsledný anonymizační efekt.

3.6 Ovládání aplikace

Aplikace má velmi jednoduché ovládání. Po úspěšném spuštění se rovnou zobrazí obraz snímaný kamerou a začne vyhledávání obličejů, které jsou po úspěšném nalezení v obraze ohraničeny čtverci spolu s vyznačenými kružnicemi kolem očí.

V aplikaci je možné nastavit tzv. *threshold*, tedy hranici pro rozdělování částí obrazu do jednotlivých tříd (2.6 a 2.12). Klávesami 'N' a 'M' se provádí snížení respektive zvýšení této hranice. Klávesa 'T' vypíše její aktuální hodnotu. Konečně klávesa 'F' spouští anonymizaci všech potvrzených obličejů a klávesa 'E' má na starosti to samé pro oči. Program se standardně ukončuje klávesou 'Q'.

4 Závěr

Jak jsem se v mé práci přesvědčil, v principu lze sledovat anonymizovaný obličej snímek po snímku pomocí detektorů popsaných výše. Pokud se obličej jakkoliv vychýlí z ideální polohy, např. nakloní k rameni nebo vytočí více z profilu, detektor naučený na frontálních obrázcích obličejů přestane fungovat. Řešením by samozřejmě bylo procházet obraz několikrát a hledat různé typy obličejů. To je ovšem zbytečně výpočetně náročné řešení a myslím, že existují řešení lepší. Jedním z nich je kombinace přístupu realizovaného v této bakalářské práci s jinou metodikou sledování, která nám ošetří situace, kdy klasifikátor ztratí přehled o pozici sledovaného objektu.

Při rešerších a studování dané problematiky jsem narazil na zajímavý přístup sledování obličeje, který využívá barevnou informaci v obraze a nespoléhá pouze na barvu jedinou, ale sleduje jejich celou skupinu. Jelikož sleduje barvu, není ovlivněn změnami orientace obličeje tam, kde jiné detektory selhávají. Jednu ze zajímavých implementací podobného detektoru obsahuje i knihovna OpenCV a je zjednodušeně založena na následujícím principu:

1. Vytvoří barevný histogram reprezentující oblast obličeje.
2. Vypočítá pro každý pixel následujícího snímku pravděpodobnost, že náleží právě obličeji.
3. Změní odpovídajícím způsobem polohu ohraničení obličeje v každém snímku.

Vytvoření histogramu probíhá pouze jednou z HSV barevného modelu a jeho jednotlivé sloupce představují údaj, kolik pixelů v obraze má daný barevný tón. Pravděpodobnost toho, že pixel patří obličeji, odpovídá hodnotě, již získáme jako procentuální poměr z celkové výšky, když sloupce histogramu naskládáme na sebe. Pravděpodobnostní ohodnocení obrazu podle správně vytvořeného histogramu z obličejové části je vidět na obrázku 4.1.

Obrázek 4.1: Pro jednotlivé pixely vypočítaná pravděpodobnost, že odpovídají svým barevným tónem obličeji (kůži). Černé – nejnižší hodnota, bílé – nejvyšší.

Jak jsem si ověřil při testování tohoto způsobu sledování, je velice rychlý a spolehlivý. Samozřejmě má své nevýhody. Problémem je, že je ovlivněn jakoukoliv jinou části těla pokrytou kůží a dále má značné problémy při atypickém nebo slabém osvětlení. Také je třeba pokaždé ručně označit část obrazu (obličej), z kterého chceme vypočítat sledovací histogram a poté důsledně rozlišovat mezi více potencionálními plochami, které obsahují stejný barevný tón, aby se nám neslévaly do jedné, apod. Nabízí se ovšem ideální kombinace s přístupem použitým v mém anonymizátoru.

Dle mého názoru řešením a ideálním pokračováním vývoje by byla kombinace klasifikátoru s doplňkovým sledováním pomocí barevného modelu. Klasifikátor by v obraze našel obličeje, čímž bychom získali ohraničené oblasti pro výpočet histogramů a tento proces by se zautomatizoval. Barevná informace by se připojila k objektovému návrhu obličeje. Sledování by mohlo probíhat pomocí barevného modelu, protože je méně náročné a klasifikátor by mohl být využíván již jen pro vyhledávání nových obličejů ve volných místech obrazu, nebo při sporných situacích, kdy bychom si potřebovali s jistotou ověřit, že sledovaná osoba opustila obraz nebo při křížení trajektorií pohybů. Jelikož by klasifikátory byly využívány v daleko menší míře, nebyl by problém přidat další ohodnocení příznaků i pro profily, obličeje z nadhledu i podhledu.

Konečně aktuální výsledek mého anonymizéru si je možné prohlédnout na obrázku 4.2. Pro tento test byla použita statická fotografie s velkým počtem obličejů. Při samotném anonymizování videa zpracovávaného v reálném čase dosahuji dobrých výsledků, ale s omezeními, která byla zmíněna. Výsledky dále např. na obrázcích 3.4 a 3.5. Anonymizace obličejů ve video vstupu v reálném čase by musela v ostrém nasazení zaručit dokonalé zpracování každého snímku (stačí byť jen jeden na odhalení identity) a k tomu by mohl dopomoci právě navržený další možný vývoj.

Obrázek 4.2: Výstup anonymizátoru pro slavnou fotografii nejmenované firmy z roku 1978, zdroj Microsoft

Literatura

[1] HEROUT, A., et al.: *Implementation of the "Local Rank Differences" Image Feature Using SIMD Instructions of CPU*. Graph@FIT, Brno University of Technology

[2] HEROUT, A. – HRADIŠ, M. – ZEMČÍK, P.: Local Rank Differences – Novel Features for Image. In *Proceedings of SCCG 2007*. Budmerice, SK, 2007

[3] HEWITT, R.: Seeing With OpenCV: A Five-Part Series. *SERVO Magazine*, 2007. T&L Publications, Inc. Dostupné z URL: <http://www.cognotics.com/opencv/servo_2007_series/index.html>

[4] JURÁNEK, R.: *Rozpoznávání vzorů v obraze pomocí klasifikátorů*. Diplomová práce, Brno, FIT VUT v Brně, 2007. Ved. Adam Herout

[5] ROWLEY, H. – BALUJA, S. – KANADE, T.: Neural Network-Based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Ročník 20, č. 1, Leden 1998

[6] ROOHI, M. – MIRJALILY, G. – SADEGHI, M. T.: Face Detection Using a Modified SVM-Based Classifier. In *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007).* Washington, DC, USA: IEEE Computer Society, 2007, ISBN 0-7695-3050-8

[7] ŠTRBA, M.: *Detekce obličeje*. Bakalářská práce, Brno, FIT VUT v Brně, 2008. Ved. Michal Hradiš

[8] WELCH, G. – BISHOP, G.: *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill, Department of Computer Science

[9] Wikipedia: AdaBoost - Wikipedia, The Free Encyclopedia. 2009, [online; cit. 2009-05-15]. Dostupné z URL: <http://en.wikipedia.org/w/index.php?title=AdaBoost&oldid=290174411>

[10] Wikipedia: Articial neural network - Wikipedia, The Free Encyclopedia. 2009, [online; cit. 2009-04-24]. Dostupné z URL: <http://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=285933987>

[11] Wikipedia: Machine learning - Wikipedia, The Free Encyclopedia. 2009, [online; cit. 2009-04-25]. Dostupné z URL: <http://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=286067808>

[12] Wikipedie: Neuronová síť - Wikipedie, otevřená encyklopedie. 2009, [online; cit. 2009-02-07]. Dostupné z URL: <http://cs.wikipedia.org/w/index.php?title=Neurovová_síť&oldid=3595710>

Seznam příloh

Příloha 1. CD – struktura:

- \src – zdrojové kódy aplikace a další potřebné soubory
- \bin – zkompilovaný program (pod OS Windows)
- \poster – plakát prezentující aplikaci
- \text – text práce v PDF a zdrojovém ODT
- \install – instalační soubory pro použité OpenCV

Příloha 2. Seznam volně přístupných databází obrázků:

- http://vis-www.cs.umass.edu/lfw/
- http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html
- http://cvc.yale.edu/projects/yalefaces/yalefaces.html
- http://vasc.ri.cmu.edu/idb/html/face/
- http://www.bioid.com/downloads/facedb/index.php
- http://vis-www.cs.umass.edu/~vidit/IndianFaceDatabase/
- http://www.cs.cmu.edu/~cil/v-images.html
- http://vision.ai.uiuc.edu/mhyang/face-detection-survey.html