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

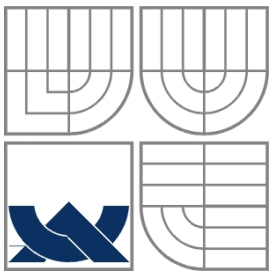
DETEKCE AUTOMOBILŮ V OBRAZE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

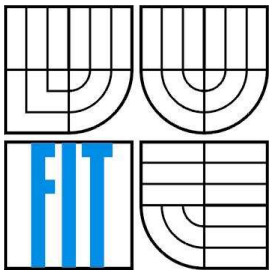
AUTOR PRÁCE
AUTHOR

ANTONÍN POMYKAL

BRNO 2010



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 AUTOMOBILŮ V OBRAZE

DETECTION OF VEHICLES IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ANTONÍN POMYKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2010

Abstrakt

Práce se zabývá možností detekce automobilů v obraze využívající charakteristických vlastností automobilů pomocí vlastních vytvořených obrazových příznaků, které jsou vytvořeny podle Haarových příznaků, a při použití metody AdaBoost pro trénování a vlastní detekci. Představíme si možnosti a typy vlastních obrazových příznaků, knihovnu OpenCV, která byla v implementaci programu využívána, a ukážeme si výsledky a úspěšnost této kombinace algoritmů při detekci.

Abstract

This work deals with the possibility of detection of cars in the image using the characteristics of cars with custom created image features , which are made pursuant to Haar-like features, and using methods of AdaBoost to train and their detection. We introduce the possibilities and types of custom picture features, OpenCV library, which was used in the implementation of the program, and we show the results and the success of this combination of detection algorithms.

Klíčová slova

detekce automobilů, OpenCV, Adaboost, Haarovy příznaky, obrazové příznaky

Keywords

car detection, OpenCV, Adaboost, Haar features, image features

Citace

Pomykal Antonín: Detekce automobilů v obraze, bakalářská práce, Brno, FIT VUT v Brně, 2010

Detekce automobilů v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Adama Herouta Ph.D. Další informace mi poskytl Bc. Vladimír Wrhel. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Antonín Pomykal
19. května 2010

Poděkování

Velmi děkuji vedoucímu bakalářské práce Ing. Adamovi Heroutovi, Ph.D. za trpělivost a pomoc při řešení problému. Také bych chtěl poděkovat Bc.Vladimíru Wrhelovi za cenné rady.

Obsah

1 Úvod

Počítačové vidění je oblastí informačních technologií, která se neustále rozvíjí a stále více vstupuje do popředí. A to ve všech možných reálných situacích. Usnadňuje práci v lékařství, výrobě, dopravě, bezpečnosti a slouží k automatizování mnoha dalších úkonů. Jako příklad počítačového vidění se nejvíce nabízí pohyb robota v jakémkoliv prostředí a jeho dovednost rozpoznávání překážek.

My se zaměříme hlavně na oblast dopravy, přesněji na detekci automobilů ve statickém obraze. Detekce automobilů nachází využití v aplikacích pro řízení provozu, zvýšení bezpečnosti provozu, asistenčních systémech automobilů nebo dokonce automatickém řízení vozidel. Existuje spoustu literatury popisující detekci lidského obličeje nebo i osob. A všechny jsou založeny na principu využívající stejných vlastností těchto objektů, jako dvě oči, ústa, ruce, nohy, tělo atd. Automobily se dají popsat podobně. Jsou to různorodé objekty s relativně stejnými vlastnostmi, jako jsou nárazníky, kola, světla, sloupky karoserie, nebo třeba i kapota. Ačkoliv některé tyto vlastnosti se mohou lišit v závislosti na typu automobilu, jsou zde geometrické tvary, podle kterých můžeme určit, že jde souhrnně o automobil. Tyto vlastnosti (tvary) se pokusíme využít k přesnějšímu vyhledání automobilu v obraze.

1.1 Cíle

K detekování tvarů automobilů použijeme podobný způsob, jako využívají Haarovy příznaky, které využívají kontrast mezi dvěma oblastmi a upravíme si je tak, aby obsáhly naše potřebné geometrické tvary automobilů. Vytvoříme si skupinu příznaků, které budou tvořit hlavně čáry a oblouky a určíme si jejich předpokládané pozice. K urychlení detekce použijeme trénovací metody Real AdaBoost, která nám vybere nejlepší kombinace příznaků [založeno na 10].

1.2 Obsah práce

Nejdříve trochu teorie. Řekneme si něco o detekci objektů. Popíšeme si metody, které se používají při detekci objektů. Zaměříme se také na Haarovy příznaky a integrální obraz. Zjistíme co je to metoda strojového učení zvaná AdaBoost, jak pracuje a jak se liší námi používaná podkategorie této metody Real AdaBoost. A pak se dostaneme do praktické části práce, kterou bude implementace programu detekce automobilů. Velkým přínosem je knihovna OpenCV, která je zde použita. A podíváme se na tvorbu vlastních příznaků.

2 Detekce objektů

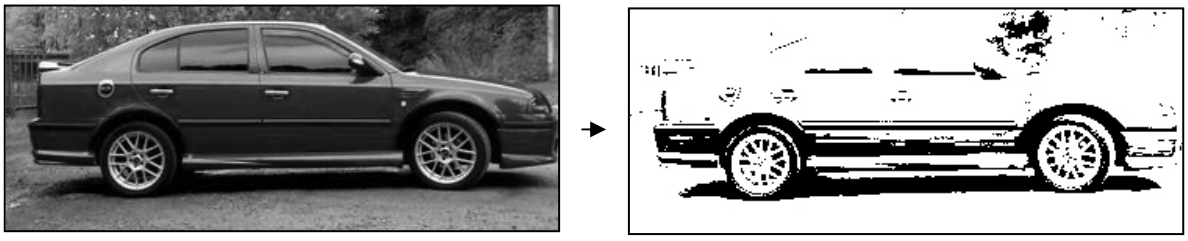
Detekce objektů je nejdůležitější a nejnáročnější funkcí v oblasti počítačového vidění. Vstupními daty jsou digitální rastrové obrazy. Rastrový obraz je jen množina obrazových bodů (pixelů), které nesou informaci jen o barvě bodu a o pozici tohoto bodu v daném obraze. Jak tedy z takového obrazu získat informaci, jaký objekt se zde nachází a kde? Nezbývá nám než se řídit podle vlastností, které víme o objektu, který se pokoušíme najít. Může to být barva objektu, tvar objektu nebo v sekvenci snímků to může být i to, že víme, že se náš objekt pohybuje. Při detekci se využívá několik postupů, založených právě na těchto vlastnostech objektu. Našli bychom spoustu prací, které se zaměřují jen na některé z nich a snaží se zlepšit jejich funkčnost, rychlost či přesnost. Podívejme se tedy, jaké základní metody se dají použít při detekci objektů.

2.1 Metody segmentace obrazu

Segmentace obrazu je skupina metod postavených na různých principech, digitálního zpracování obrazu, která slouží k automatickému rozdělení vlastního obrazu na oblasti se společnými vlastnostmi a které obvykle mají nějaký smysluplný význam. Typickým cílem segmentace obrazu je identifikace popředí a určení oblastí v obraze odpovídajícím významnému prvku zachycené scény. (převzato z [1])

2.1.1 Prahování

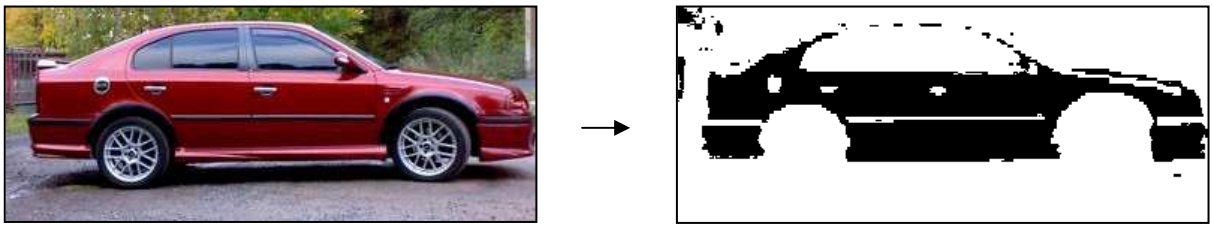
Metoda prahování se dá velmi dobře použít, když víme jakou barvu nebo jas bude mít náš hledaný objekt. Princip této metody je, že pouze ty obrazové body, které mají hodnotu nad určitou hranici (práh), nastavíme na bílou barvu a body pod tuto hranici (prahem), nastavíme na černou barvou. Příklad prahování jednou hodnotou (prahem) je možné vidět na Obrázek 2.1.



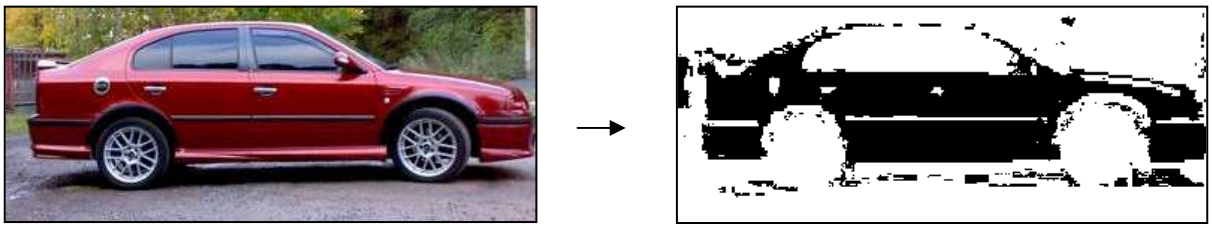
Obrázek 2.1 : Prahování černobílého obrázku prahem 28

Zde je vidět, že by se dalo snadno detekovat kola nebo podběhy nad koly. Obměnou této metody můžeme z obrázku extrahovat i barvu v určitém rozsahu. Zde se nabízí možnost extrakce barvy v různých barevných modelech, jako jsou například RGB (Red, Green, Blue) nebo HSV (Hue, Saturation, Value). Ukázky jak může extrakce vypadat v různých barevných modelech je znázorněna

na obrázcích Obrázek 2.2 a Obrázek 2.3. Barevný model HSV není tak náchylný na změnu jasu bodu jako model RGB. I když z obrázků by se spíše dalo říci, že je to naopak, ale vše záleží na výběru rozsahu hledané barvy. U modelu HSV je vybrána jakákoliv červená barva včetně všech odstínů, tmavých i světlých.



Obrázek 2.2 : Ukázka prahování červené barvy určitého rozsahu v barevném modelu RGB

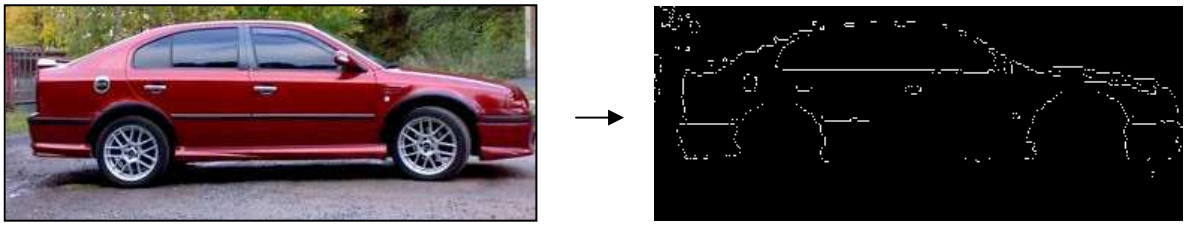


Obrázek 2.3 : Ukázka prahování jakékoliv červené barvy v barevném modelu HSV

Jak je z obrázků patrné, je nutné, aby s touto metodou byla použita metoda pro určení objektu. Z obrázku jsme sice vybrali možný barevný objekt, ale to ještě neurčuje, zda jsme uspěli v detekci našeho objektu. Na obrázku totiž může být i několik objektů stejné barvy a my potřebujeme zjistit, který objekt odpovídá tomu, který hledáme. Musíme tedy ještě aplikovat metodu, která nám zjistí, zda jde o náš objekt. Prahování ale výrazně zlepší možnosti vyhledání objektu.

2.1.2 Detekce hran

Další metodou segmentace obrazu je metoda detekce hran (*angl. edge detection*). Hranu můžeme definovat jako kontrastní informaci mezi dvěma oblastmi, nebo také jako derivaci jasové funkce v dané oblasti. V místě hrany bude mít derivace velkou hodnotu a největší hodnotu bude mít ve směru kolmém na hranu. Velká skupina metod na detekci hran aproximuje tuto derivaci pomocí konvoluce s vhodným jádrem (převzato z [2]). V současnosti nejlepším hranovým detektorem je Cannyho hranový detektor (ukázka použití na Obrázek 2.4).



Obrázek 2.4 : Ukázka Cannyho hranového detektoru s prahem 30

I tato metoda pouze zjednodušuje detekci. Zde také musíme aplikovat ještě některou další metodu detekce pro určení, zda se jedná o hledaný objekt.

2.2 Detekce pohybu

Metodu detekce pohybu využíváme tehdy, máme-li k dispozici sekvenci snímků, například z video souboru nebo přímo ze zdroje (videokamery). Většinou porovnáváme dva po sobě jdoucí snímky a vyhodnocujeme změny hodnot jednotlivých obrazových bodů (pixelů). V ideálním případě, kdy by detekce pohybu značila, že nenalezla žádný pohyb, dva po sobě jdoucí snímky budou stejné. To ale v reálném světě není vždy možné vzhledem ke změně osvětlení během dne, povětrnostním podmínkám ve snímaném okolí, nebo i vlivem šumu. Závisí to i na snímací technice. Tyto vlivy lze částečně omezit aplikací obrazových filtrů (Gaussovské rozostření, Medián,...), nebo zprůměrováním hodnot několika po sobě jdoucích snímků. Porovnávání provádíme nad snímky pozadí a popředí. Pozadí bývá předchozí snímek a popředí bývá snímek aktuální. Tyto metody můžeme rozdělit podle snímku pozadí. A to na pozadí statické, kdy kamera snímá pouze dané okolí, nebo pohyblivé, kde se kamera může třeba otáčet. Při statickém pozadí odečteme od aktuálního snímku snímek pozadí (angl. Background Subtraction) a je-li počet pixelů, které se změnily, větší než stanovená mez, můžeme říci, že se v daném okolí kamery pohybuje nějaký předmět a můžeme dokonce určit, kde se pohybující objekt nachází (Obrázek 2.5). Obdobou této metody detekce se statickým pozadím je vypočítání histogramu jasové složky snímku pozadí a popředí a jejich porovnání. Tyto metody mají výhodu v tom, že zjištění pohybu objektu je velmi jednoduché a poměrně rychlé. Nevýhoda je, že snímací kamera musí být umístěna napevno a také musíme odstranit výše zmíněné rušivé vlivy (stíny, vítr,...).

Obrázek 2.5 : Detekce pohybu za použití statické kamery (pozadí, popředí, výsledné odečtení)

U metody s pohyblivým pozadím, musíme zprůměrovat několik posledních snímků a vytvořit z nich nové pozadí. Poté můžeme porovnat jako v metodě se statickým pozadím. Nevýhodou této metody je, že se neustále musí provádět nové výpočty pozadí. Není možné použít staré hodnoty, protože již nemusí platit pro danou pozici kamery.

2.3 Detekce pomocí klasifikátorů

Metoda detekce pomocí klasifikátorů je dnes v oblasti počítačového vidění nejvíce používaná a bohužel je také velmi složitá a náročná. Abychom mohli detekovat touto metodou, musíme nejdřív detektor natrénovat na množině trénovacích dat. Už v první fázi této metody je vidět rozdíl v náročnosti oproti předchozím metodám. U nich jsme dopředu nepotřebovali nic znát. Samotné trénování je velmi náročnou operací a doba jeho zpracování taky není malá. Trénovací data obsahují dvě třídy snímků. V jedné třídě jsou pouze snímky obsahující objekt dané třídy a v druhé snímky bez tohoto objektu [3].

Klasifikace obrazu je vlastně zařazení testovaného obrazu právě do jedné z těchto tříd, vyhodnocením jednoho nebo několika klasifikátorů. Výsledkem je buď binární klasifikace, která pouze zjistí, zda se v tomto obrazu nachází hledaný objekt nebo ne, tedy množinu {-1,+1}. V případě klasifikace vícehodnotové, se nám jako výsledek vrací pravděpodobnost, s jakou se hledaný objekt v daném obraze vyskytuje. Formální popis klasifikace:

Vstup:

množina trénovacích dat $\{(x_1, y_1), \ldots (x_m, y_m)\}$, x_i náleží X, y_i náleží Y

Výstup:

klasifikátor $h : X \rightarrow Y$

Binární klasifikace

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

Klasifikátor může být slabý nebo silný. Slabým klasifikátorem zde může být jakákoliv funkce, která dokáže rozhodovat lépe než náhodná funkce (tedy její chyba je nižší než 0.5). Základem slabého klasifikátoru bývají obrazové příznaky (viz kapitola 2.6). Silný klasifikátor je tvořen lineární kombinací slabých klasifikátorů, ale pozor, není to lineární klasifikátor. Klasifikátor je množina příznaků, které popisují vlastnosti dané oblasti obrazu. Příznak není možné vytvořit nějakou funkcí, protože se nedají automaticky určit vlastnosti objektu, které má funkce očekávat. Proto příznaky tvoří autor implementace.

Trénování klasifikátorů se dá rozdělit do dvou částí:

- **učení bez učitele**, kdy dělení vzorů dat do tříd probíhá na základě zadaného kritéria kvality (též clustering)
- **učení s učitelem**, kde o rozdělení do tříd rozhoduje učitel.

Obyčejně se trénování provádí právě učením s učitelem. Jde o metodu zjišťování klasifikační funkce z dat, u kterých víme, ke které třídě náleží. Klasifikátor se během učení naučí podle dat předpovědět třídu klasifikace pro neznámá data (generalizovat). Při učení nelze v datech specifikovat všechny možné případy vstupu a výsledek proto bude mít určitou chybu. Při trénování se s použitím co nejlepších trénovacích dat snažíme, aby tato chyba byla co nejmenší. Komplexní klasifikátory mají menší chybu, ale tomu odpovídá i delší čas potřebný na natrénování a vyhodnocení vstupních dat. Při trénování se obyčejně používají trénovací a testovací data. Učení probíhá na trénovacích datech a chyba se kontroluje na testovacích datech. Trénovací data obsahují obrazy objektů a jejich rozdělení do tříd [5].

Mezi nejznámější metody detekce pomocí klasifikátorů patří v současné době neuronové sítě, SVM a Boosting. Tyto metody si více přiblížíme v dalších podkapitolách. Souhrnně ale o nich můžeme říci, že jsou velmi rozsáhlé, mají široké možnosti uplatnění a docela vysokou úspěšnost. Pro tyto své výhody jsou hodně používané. A také neustále zdokonalované.

2.3.1 AdaBoost

Tato kapitola byla převzata a upravena z [4 a 8]. AdaBoost vznikl už v roce 1995, ale jeho využití při detekci obličejů bylo představeno v roce 2001 (Viola & Jones). AdaBoost je zkratkou slov Adaptive Boosting. Adaptabilní proto, že využívá více slabých klasifikátorů a má schopnost při učení využívat množinu předešlých špatně zařazených objektů k natrénování slabých klasifikátorů. AdaBoost je algoritmus strojového učení, jehož vstupem je trénovací množina dat a výstupem klasifikátor, zařazující data do dvou tříd. Výsledný klasifikátor *H(x)* je lineární kombinací tzv. slabých klasifikátorů *ht(x)*

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Slabý klasifikátor nemá velkou přesnost. Ale musí být větší než náhoda, takže musí být větší než 50%. V každém kroku učení je do této lineární kombinace přidán jeden slabý klasifikátor z množiny klasifikátorů *H*. Výběr v každém kroku učení je realizován takovým způsobem, aby byl minimalizován horní odhad chyby klasifikátoru. Učení pomocí AdaBoostu je shrnuto níže v popisu algoritmu diskrétního AdaBoostu. Trénovací množina se skládá z dvojic (x_i, y_i), kde x_i je meření a y_i skutečná příslušnost měření x_i k jedné ze dvou tříd $\{-1, 1\}$. AdaBoost využívá při učení ovážení trénovací množiny váhami *Dt*. Ty jsou na začátku inicializovány rovnoměrně. Vlastní algoritmus učení probíhá ve smyčce.

V každém cyklu smyčky je třeba provést následující kroky:

1. Nalézt nejlepší slabý klasifikátor při daném ovážení *Dt* trénovacích dat,

2. Ověřit, že chyba tohoto klasifikátoru nepřekročila 0.5,

3. Spočítat koeficient slabého klasifikátoru v lineární kombinaci *H(x)*,

4. Aktualizovat váhy *Dt*.

Algoritmus diskrétního AdaBoostu [7]:

Vstup : $(x_1, y_1), ..., ((x_m, y_m); x_i \in X, y_i \in \{-1,+1\})$

Inicializuj váhy $D_1(i) = \frac{1}{m}$

Pro $t = 1, ..., T$:

1. Najdi $h_t = \arg\min_{h_j \in \mathrm{H}} \varepsilon_j ; \varepsilon_j = \sum_{i=1}^{m} D_t(i) I\left[y_i \neq h_j(x_i)\right]$
2. Když $\varepsilon_t > \frac{1}{2}$ potom skonči
3. $\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
4. Aktualizuj

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

kde $Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$

Výsledný klasifikátor:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Vybraný slabý klasifikátor z kroku 1 má nejmenší váženou chybu na trénovací množině. Výraz vrací 1 pokud je výrok pravdivý a 0 pokud je nepravdivý.

Podmínka v kroku 2 zajišťuje, aby byl nalezený slabý klasifikátor lepší než klasifikátor vracející náhodnou třídu. To je potřeba k zajištění konvergence algoritmu.

Výběr příznaku podle minima vážené chyby společně s výpočtem koeficientu pro lineární kombinaci α_t (krok 3) jsou navrženy tak, aby byl v každém kroku učení minimalizován horní odhad chyby výsledného klasifikátoru:

$$\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t = \frac{1}{2^T} \prod_{t=1}^{T} \sqrt{\varepsilon_t (1-\varepsilon_t)}$$

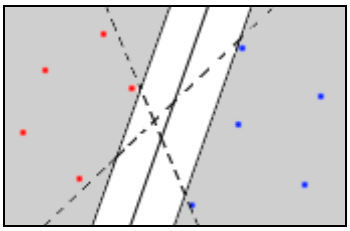
Aktualizace vah v kroku 4 způsobí to, že váha špatně klasifikovaných měření se zvětší a váha dobře klasifikovaných se zmenší. V následujícím kroku bude tedy hledán slabý klasifikátor, který bude muset lépe určit doposud špatně klasifikovaná měření.

2.3.2 Real AdaBoost

Jak už název napovídá jedná se o metodou trénování klasifikátorů odvozenou z výše popsané metody AdaBoost. Její hlavní funkce je stejná, tak si ji jenom trochu popíšeme. Rozdíl oproti diskrétnímu AdaBoostu je, že slabé klasifikátory dávají reálné hodnoty. Už ne {-1,+1}, ale například 4,2342. Metoda má mnohem rychlejší konvergenci (kratší klasifikátor) a lepší úspěšnost na testovacích datech. Velmi často má v praktických úlohách lepší výsledky. Existují ale i algoritmy, které by měli mít lepší vlastnosti.

2.3.3 SVM – Support Vector Machine

SVM (Support Vector Machine) (viz [6]) je metoda klasifikace lineárních dat. Používá se například v aplikacích pro vyhledávání obrazů. Tato metoda klasifikuje data pomocí nadroviny, která rozděluje trénovací data do dvou kategorií. Snahou je získat takovou rozdělovací nadrovinu, která je od jednotlivých kategorií trénovacích dat co nejvíce vzdálená. Klasifikační úloha zahrnuje trénování a testování dat. Trénovací data jsou tvořena dvojicemi (xi, yi), i = 1, ..., k kde xi $\in$ Rn a y $\in$ {1, -1}k . Xi je vektor dimenze n popisující vlastnosti daného prvku. Yi určuje do které množiny vektor patří. Metoda SVM je ze své přirozenosti binární. Rozděluje data do dvou množin s příznaky 1 a -1. Testovací data tvoří pouze vektory x. Cílem klasifikátoru je podle hodnot vektoru určit jeho příznaky. Jedním z problémů je nalezení takové dělící nadroviny, která by správně rozdělila trénovací data, a přitom byla od těchto dat co možná nejvíce vzdálená. Větší vzdálenost od trénovacích dat zajišťuje lepší výsledky nad testovacími daty. Vektorům, které jsou nejblíže nadrovině a jsou proto důležité při její konstrukci, se říká support vectors. Trénovací data obvykle není možné takto jednoduše rozdělit. Místo toho, aby se použila nějaká nelineární křivka, namapují se data do vyšší dimenze, kde se data rozdělí nadrovinou. Původně lineárně neseparovatelná data se tak stanou lineárně separovatelná.



Obrázek 2.6 : Ukázka maximalizace vzdálenosti dělící hranice od dat

2.3.4 Neuronové sítě

Neoronová síť je výpočetní model, který se používá v oblasti informačních technologií zaměřené na umělou inteligenci, ale také se používají pro rozpoznávání a kompresi obrazů nebo zvuků.. Princip činnosti je založen na fungování lidského mozku. Umělá neuronová síť je určena k paralelnímu zpracování dat.

Skládá se z umělých neuronů, které kopírují vlastnosti biologických neuronů. Tyto neurony jsou vzájemně propojeny a navzájem si předávají signály a přeměňují je pomocí určitých přenosových funkcí. Neuron má libovolný počet vstupů, ale pouze jeden výstup.

Popisů modelů neuronu je spoustu. Od jednoduchých, které používají nespojité přenosové funkce, až po velmi složité, které popisují každý detail chování neuronu. Model neuronu je znázorněn na Obrázek 2.7. Jedním z nejpoužívanějších je model popsaný McCullochem a Pittsem (popis převzat z [9]):

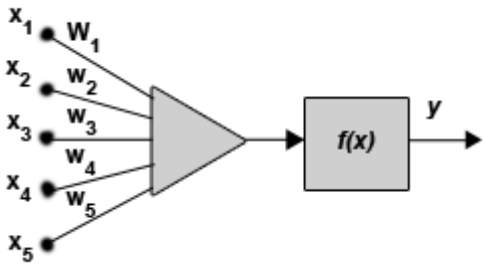
$$Y = S\left(\sum_{i=1}^{N}\left(w_i x_i\right) + \Theta\right)$$

kde:

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

Velikost vah wi vyjadřuje uložení zkušeností do neuronu. Čím je vyšší hodnota, tím je daný vstup důležitější. V biologickém neuronu práh Θ označuje prahovou hodnotu aktivace neuronu.

Tzn. $\sum_{i=1}^{N}\left(w_i x_i\right)$ je-li menší než práh, neuron je v pasivním stavu.



Obrázek 2.7 : Model umělého neuronu

Podle povahy vstupních (a výstupních) dat můžeme neurony dělit na binární a spojité. Podle typu neuronu a typu neuronové sítě se použije vhodná přenosová funkce.

Typy přenosových fukcí:

Skoková přenosová funkce - vrací pro vstup menší než daná mez nulu, pro větší vrací jedna

Sigmoidální přenosová funkce - Její hodnoty se blíží nule v minus nekonečnu a jedničce v nekonečnu. Pro nulu je hodnota 0,5.

Přenosová funkce hyperbolické tangenty - Její hodnoty se blíží -1 v minus nekonečnu a jedničce v nekonečnu. Pro nulu je hodnota 0.

Přenosová funkce radiální báze - hodnoty se blíží nule v minus nekonečnu a nule v nekonečnu. Pro nulu je maximální hodnota 1.

Typy neuronových sítí

Základním typem neuronových sítí je dopředná síť. Informace v ní proudí od vstupů přímo k výstupům a nejsou v ní žádné smyčky. Typickým příkladem této sítě je jednovrstvý perceptron. Jedná se pouze o jednu vrstvu výstupních neuronů. Vícevrstvý perceptron se skládá z několika vrstev neuronů, které mají výstupy jedné vrstvy připojené na vstupy vrstvy následující. V tomto typu perceptronu nalezneme několik vrstev. Určitě musí obsahovat vstupní a výstupní vrstvu a jednu nebo více skrytých vrstev.

Jiným typem sítí jsou rekurentní sítě. Na rozdíl od dopředných sítí je zde možné nalézt navíc zpětné vazby. Zástupcem rekurentní sítě je Hopfieldova síť. Neurony v této síti jsou vzájemně propojeny každý s každým. Tato síť má vlastnosti asociativní paměťi a jde tedy použít pro úlohy obnovení poškozené informace nebo rozpoznávání.

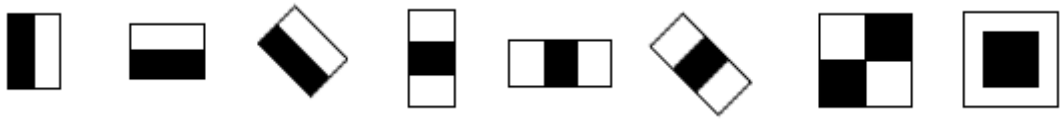
2.4 Haarovy příznaky

Příznaky nesou informaci o intenzitě dané oblasti, na kterou je příznak aplikován. Podaří-li se nám při tvorbě příznaku tuto vlastnost jednoduše získat, výsledná metoda je pak velmi rychlá.

Haarovy příznaky jsou jednoduché obdélníkové oblasti. Skládají se z černé a bílé oblasti. Ukázky Haarových příznaků jsou na Obrázek 2.8 . První dva příznaky jsou základní tvary těchto příznaků. Mají jen dvě oblasti. Další příznaky jsou z rozšířené sady, která již obsahuje pootočené příznaky a příznaky skládající se i ze tří oblastí. Aplikací Haarova příznaku na obrazová data, získáme odezvu příznaku, která se skládá z rozdílu černé a bílé části. Každá část je součet intenzit pixelů pod touto částí. Výslednou odezvu můžeme tedy spočítat podle vzorce:

$$f(p) = \sum I(w) - \sum I(b)$$

kde *I(...)* je intenzita pixelu, a *w* je pixel z bíle oblasti a *b* je pixel z černé oblasti. Rychlost výpočtu je závislá na velikosti daných oblastí. Ke zrychlení výpočtu těchto příznaků se používá Integrální obraz.



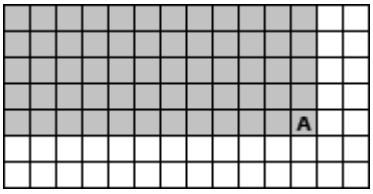
Obrázek 2.8 : Ukázka základních a rozšířených Haarových příznaků

2.5 Integrální obraz

Jak bylo napsáno výše, integrální obraz se používá ke zrychlení výpočtu intenzit pixelů v Haarových příznacích. Je to vlastně matice hodnot se stejnými rozměry jako má testovaný obraz, kde každý prvek této matice obsahuje sumu všech intenzit pixelů testovacího obrazu v oblasti nad a vlevo tohoto prvku (viz Obrázek 2.9). Integrální obraz můžeme formálně popsat rovnicí:

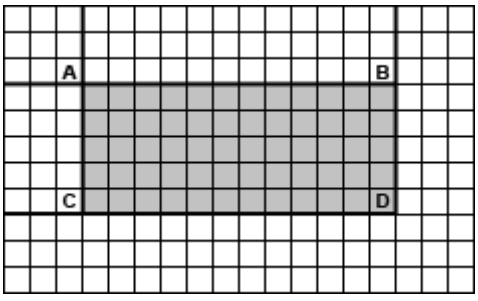
$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$

Integrální obraz tak umožňuje v konstantním čase spočítat sumu intenzit jakkoliv velkého a kdekoliv v obraze umístěného obdélníku pouhým vyhledáním hodnot 4 prvků integrálního obrazu. Ukázka hodnot pro výpočet je na Obrázek 2.10.



Obrázek 2.9 : Ukázka integrálního obrazu

(prvek A má hodnotu rovnou součtu intenzit všech prvků v označené oblasti)



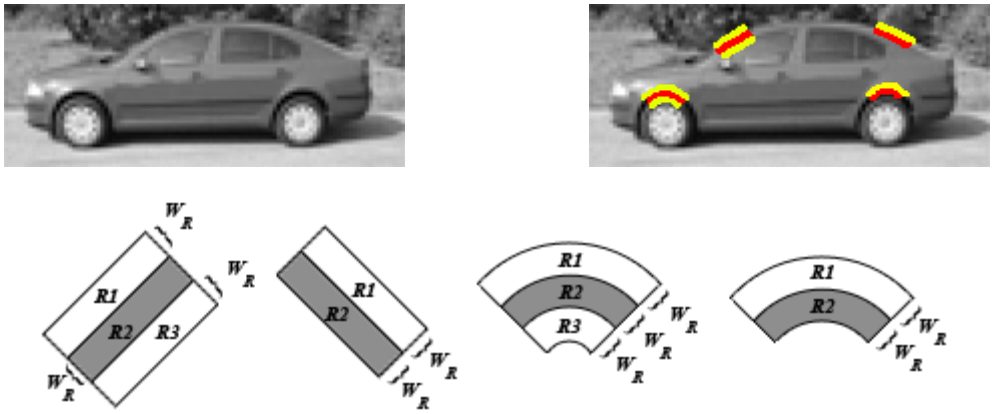
Obrázek 2.10 : Ukázka výpočtu v integrálním obraze

Výslednou sumu intenzit prvků *I* v označené oblasti spočítáme pomocí vzorce:

$$I = A - B - C + D$$

2.6 Obrazové příznaky

Auta mají ve své podstatě relativně stejné tvary. I když hodně záleží na typu automobilu, ale všechny mají dané vlastnosti jako jsou kola, kapota, světla, nárazníky a další. Proto můžeme k určení automobilu použít několik geometrických útvarů. My budeme pracovat s čárami a oblouky. Příznaky, které jsme vytvořili, se velmi podobají Haarovým příznakům. Mají také dvě nebo tři oblasti (Obrázek 2.11). Jen je bohužel kvůli tvaru obloukových částí nemůžeme tak snad vypočítat s integrálního obrazu. Ukázka skutečných oblastí obloukových příznaků, je na Obrázek 2.12



Obrázek 2.11 : Ukázka nových příznaků a jejich možná aplikace na trénovací obrázek

Odezvu jednoduchého hranového (2-oblastního) příznaku můžeme vypočítat podle vzorce:

$$f_{hrany} = \left| \frac{\sum_{(x,y)\in R1} I(x,y)}{\|R1\|} - \frac{\sum_{(x,y)\in R2} I(x,y)}{\|R2\|} \right|$$

kde I(x,y) je intenzita pixelu, a ||..|| je počet pixelů v dané oblasti. Obdobně můžeme vypočítat i odezvu pro hřbetový (3-oblastní) příznak, a to:

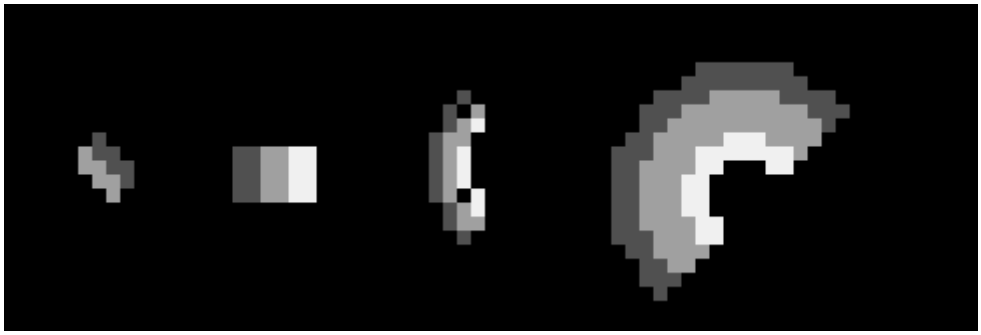
$$f_{hrbetu} = \left| \frac{\sum_{(x,y)\in R1} I(x,y)}{\|R1\|} + \frac{\sum_{(x,y)\in R3} I(x,y)}{\|R3\|} - \frac{2\sum_{(x,y)\in R2} I(x,y)}{\|R2\|} \right|$$

V obou případech vzorců je použita absolutní hodnota na výsledek. Je to v důsledku kontrastních přechodů u automobilů, které jsou mnohem komplexnější na rozdíl od obličejů. Obličeje mají většinou kontrastní informací stejnou, například u očí. Oči bývají většinou tmavé a pokožka světlá.

Ale u automobilů to může být i obrácené. Podívejme se třeba na přechod z kapoty do pozadí. Jaká by byla odezva, kdyby auto bylo tmavé, a jaká by byla, kdyby auto bylo světlé a pozadí tmavé. Použitím absolutní hodnoty při výpočtu popíšeme kontrastní informaci, která je invariantní vůči barvám automobilu a pozadí.

Pro popis automobilů se zdají být nejvhodnější geometrické tvary příznaků podle čar a podle oblouků. Oblouky jsou široký pojem, proto se omezíme na půlkruhy, čtvrtiny kruhů a osminy kruhů.

Zde trochu předbíhám, ale nastíním velikosti a natočení pro naše příznaky. Pro linky se osvědčilo použití délky linky v rozmezí od 3 do 6 pixelů, natočení linky od 0° po 15° až do 165°, a šířka oblasti od 2 do 3 pixelů. U obloukový tvarů to byla délka od 3 do 8 pixelů, natočení od 0° po 22,5° až do 360°, a šířka oblasti od 1 do 3 pixelů. Příklad tvarů mých příznaků:



Obrázek 2.12 : Ukázky skutečných tvarů aplikovatelných příznaků

2.7 OpenCV

OpenCV (Open Source Computer Vision) je multiplatformní knihovna funkcí zaměřená na počítačové vidění. Vyvinula ji společnost Intel. Je distribuována pod licencí BSD. Byla vytvořena, aby byla schopna pracovat na systémech Windows, Linux i MacOS. Knihovna je zaměřená zejména na zpracování obrazu v reálném čase a obsahuje k tomuto účelu již více než 500 funkcí. Právě pro usnadnění práce s obrazem, detekcí a trénováním klasifikátorů, jsem ji použil v implementaci programové části práce.

3 Implementace

V předchozích kapitolách jsem popsal možnosti, které se při detekci objektů v obraze dají použít a jaké mají vlastnosti a použití. Každá z nich se hodí na jinou práci, podle toho co a jak chceme hledat.

Mě zaujala možnost detekce objektů natrénováním určitých dat. Nejdříve jsem se zaměřil na detekci Haarovými příznaky. Poté jsem se zaměřil na vytváření vlastních příznaků. K trénování těchto příznaků jsem se mě zdálo být vhodné použít algoritmus AdaBoost, a také mě zajímalo, jaké dosahuje tento algoritmus výsledků nad testovacími daty. Vzhledem k vytváření vlastním příznaků, jsem použil trénovací metodu odvozenou právě od AdaBoostu, kterou byl Real Adaboost. Ten je přiblížen v kapitolách výše. Mým koníčkem jsou automobily, proto jsem se zaměřil na detekci automobilů. Ale detektor, který jsem chtěl vytvořit, by měl jít použít, při natrénování potřebných dat, i k detekci jiných objektů než jen automobilů.

Vlastní program byl implementován v jazyce C++, a vývoj a testování probíhalo na operačním systému Windows XP Professional v prostředí programu Microsoft Visual Studio 2005. Pokusy probíhali i na operačním systému Windows 7, ale po mnoha neúspěšných pokusech implementace a testování na tomto systému, jsem se vrátil k Windows XP. Co se týká hardwarových prostředků, které jsem použil, tak je to notebook s procesorem AMD Mobile Sempron 1,8GHz s pamětí 1024 MB RAM.

Program ke své funkčnosti využívá multiplatformní knihovny pro zpracovávání obrazu OpenCV, která obsahuje kromě funkcí pro zpracování obrazu i funkce pro zobrazování výsledků a jejich prezentaci na vyšší úrovni. Nejdříve jsem používal verzi 1.1, ale po vydání nové verze jsem přešel na 2.1, kvůli vylepšení stávajících funkcí. Mimo jiné i funkce pro detekování obličejů a trénování dat pro detekci. Tato funkce využívá Haarovy příznaky.

K testování správné funkčnosti aplikace byla použita databáze automobilů obsahující 550 automobilů a 500 pozadí (viz Přílohy). Vzhledem k malému počtu obrázku, negativních i pozitivních, se nedalo očekávat dobré výsledky, ale pro odzkoušení trénování a detekce tato databáze postačovala. Vytvořil jsem i databázi moderních evropských aut, ale vinou chyby disku jsem bohužel o tyto data přišel. Novou datovou sadu jsem vytvořil znovu, ale již jsem ji nestihl otestovat na programu. Proto jsou jako testovací výsledky použity obrázky ze stažené databáze (viz výše).

Implementace programu se kvůli použití trénovacího algoritmu dělí na dva samostatné programy. Na program, který trénuje klasifikátor na daných datech a na program, který se pokouší detekovat správné objekty v testovacích datech.

3.1 Třídy příznaků

Na začátku programování jsem se rozhodoval, co bude nejlepší pro jednoduchou práci s příznaky. Vytvořil jsem si tedy dvě třídy. Jednu pro popis vlastního příznaku a druhou pro funkce, které budu nad množinou příznaků používat. Funkce pro popis vlastního příznaku se nazývá `feature`. Obsahuje jenom hodnoty pro nastavení vlastností příznaku. Její strukturu vidíme zde:

```
class feature
{
 public:
       int type;
       int region_type;
       int single_region_width;
       int length;
       float angle;
       bool region_generated;
       vector<CvPoint> R1;
       vector<CvPoint> R2;
       vector<CvPoint> R3;
       ~feature();
};
```

Hodnota `type` určuje o jaký příznak se jedná (čára nebo jeden z typů oblouků). Proměnná `region_type` popisuje, zda se jedná o příznak s dvěma nebo třemi oblastmi, `single_region_width` určuje šířku jedné oblasti, `length` její délku, `angle` značí úhel natočení příznaku. Jen pro kontrolu je tu proměnná `region_generated` , která ukazuje, že daný příznak je již vygenerován celý. Ještě nesmím zapomenout na nejdůležitější část a to jsou vektory bodů jednotlivých oblastí `R1,R2,R3`.

Druhá třída s názvem `features`, je používána jak při trénování tak při detekci. Při detekci ji používáme proto, že musíme vyhodnotit všechny příznaky, se kterými byl natrénován klasifikátor, nad každým obrazem, ve kterém chceme zjistit, zda se tam nachází hledaný objekt. První funkcí, která nás v této třídě zajímá jako první je funkce `generate_features()`. Ta si zavolá dvě podfunkce `generate_line_features()` a `generate_arc_features()` , jak je z názvu zřejmé,

každá vytvoří příznaky určitého druhu. Je nutné volat pro každý typ jinou funkci, kvůli rozdílné náročnosti a požadavkům umístění v trénovaném obraze. Po úspěšném proběhnutí těchto funkcí se nám naplní seznam příznaků `list<feature> list_features`. Ke zjištění kolik se nám vytvořilo vlastně příznaků je zde funkce `get_count_features()`. Nyní máme příznaky vytvořeny. Vzhledem k tomu, že generování příznaků někdy trvá docela dlouho vytvořil jsem si funkci pro ukládání příznaků do souboru `save_to_file(string file_name)` a pro načítání příznaků ze souboru `load_from_file(string file_name)`.

Po načtení příznaků se můžeme pustit do trénování dat. Zvlášť si vytvoříme matici `CvMat`, která bude mít jako sloupce odezvy všech příznaků a jako řádky jednotlivé příklady. Postupně budeme načítat obrazy trénovacích dat a pro každý zavoláme funkci `add_train_image(IplImage* test_image, CvMat* examples_matrix, long position_in_matrix)` s pozicí, na kterou má daný příklad uložit. Jakmile máme takto vytvořenou matici příkladů, vytvoříme si matici klasifikace daných příkladů do tříd, neboli matice má jeden sloupec a počet řádků je stejný jako počet trénovacích příkladů. Každý řádek této matici určuje, dokteré třídy můžeme zařadit daný příklad, tedy zda má příklad pozitivní odezvu na daný objekt nebo ne. Když už máme vytvořenou i tuto matici, nic nám nebrání se pustit do samotného trénování. K trénování byla použita třída, implementovaná v knihovně OpenCV, vytvořená právě kvůli trénování dat. Jmenuje se `CvBoost`. Ta obsahuje funkci `train(…)`, která po zadání všech parametrů spustí trénování. Jakmile skončí, uložíme si výsledný klasifikátor do souboru, abychom jej mohli použít při detekci. To je dá se říct vše, co potřebujeme k trénování z hlediska mé třídy příznaků.

Podívejme se ještě na detekci. Jak už bylo popsáno výše, nejdříve si načteme příznaky z již dříve vytvořeného souboru `load_from_file(string file_name).` Musíme také zajistit načtení souboru s klasifikátory. To se děje ve třídě CvBoost. Jakmile máme tyto dva soubory, můžeme začít s detekcí. Postupně načítáme obrazy pro detekci a aplikujeme na ně funkci `test_on_image(IplImage* test_image, CvMat* examples_matrix).` Tato funkce aplikuje všechny příznaky na daný obrázek a uloží jejich odezvy do matice dané druhým parametrem. Poté nastává práce opět pro třídu `CvBoost`, které dáme jako parametr právě tuto matici, kterou naplnila hodnotami naše funkce. Třída `CvBoost` vyhodnotí odezvy a vrátí nám odpověď, zda se v daném obraze nachází hledaný objekt. Tento postup je nastíněn pro obrazy stejně veliké jako trénovací. V opačném případě musíme vstupní obraz projít posuvným oknem, které nám vyřízne části obrazu. S těmito části potom voláme funkci `test_on_image(….)` a určíme, zda se v daném výřezu objekt nachází. Takový průchod obrazem dá velké množství kladných odpovědí (tzv. falešné detekce) a my je musíme redukovat. K tomu můžeme použít třeba metodu požadavku minimálních sousedních detekcí. To je, že si nastavíme práh, od kolika sousedících detekcí budeme považovat detekci za správnou a ty sloučíme.

3.2 Trénování

V této kapitole si popíšeme nejdůležitější parametry programu pro trénování příznaků. Program se jmenuje `train` a má dva povinné parametry. Jsou to seznamy pozitivních a negativních obrazů:

`-list_pos` soubor - seznam obrázku (soubor obsahuje na každém řádku cestu k souboru)

`-list_neg` soubor - seznam obrázku neobsahujících automobily (opět jeden na řádek)

Toto jsou dva důležité a povinné parametry. Nyní si objasníme parametry volitelné. Při použití výše psané třídy, zde máme parametry pro výběr souboru s prvky, které budou načteny bez předchozího generování:

`-save_boost` soubor - uložení výsledných natrénovaných dat, defaultně nastaven soubor "boost.xml"

`-load_features` soubor - načte již vygenerované příznaky ze souboru (defaultně je program bude generovat)

`-save_features` soubor - uloží vygenerované příznaky do souboru (defaultně je nastaven soubor "features.xml "

`-width` - šířka trénovacího okna

`-height` - výška trénovacího okna

`-skip` - přeskočí chybu při špatném načtení obrázku, jde nastavit i za běhu programu

`-h` - vypíše programovou nápovědu, jestli je tento parametr zadán ještě s nějakými, jsou ostatní ignorovány a je vypsána nápověda

Nyní máme za sebou parametry pro správnou funkci programu a nyní se podíváme na volitelné parametry, které slouží k nastavování trénování dat. Tyto parametry jsou odvozené právě ze třídy CvBoost. Přesná definice parametrů trénování je:

```
CvBoostParams( int boost_type, int weak_count, double weight_trim_rate, int max_depth, false, 0 )
```

Parametry zadávané programu jsou stejné:

-boost_type typ	- jako typ algoritmu pro boosting můžeme vybírat z možností REAL (defaultně), DISCRETE, LOGIT a GENTLE
-weak_count celé číslo	- počet slabých klasifikátorů
-weight_trim_rate reálné číslo	- úspěšnost (reálné číslo od 0 do 1)
-max_depth celé číslo	- maximální zanoření

Program kontroluje vstupní hodnoty. Například nepodaří-li se načíst příznaky ze souboru, nabídne uživateli jejich vygenerování. Další ochranou je, že když se nepodaří, při vytváření matice příkladů, načíst nějaký trénovací obrázek, tak po ukončení načtení všech obrázků upraví trénovací matici, jen pro data, která se podařilo načíst.
Získávání odezvy na jednotlivé příklady probíhá, jak bylo popsáno výše, aplikací obrazových příznaků.

K trénování jsme si toho už řekli hodně, snad jen se můžeme podívat, jak vlastně vypadají trénovací příklady:

Obrázek 3.1 : Datová sada negativních obrázků (pozadí), rozměr 100x40

Obrázek 3.2 : Datová sada pozitivních obrázků, rozměr 100x40

Obrázek 3.3 : Nová datová sada automobilů, bohužel nenatrénovaná, rozměr 64x32

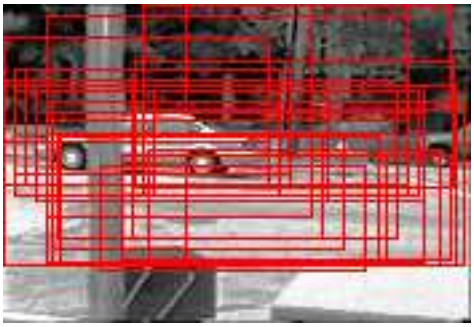
3.3 Detekce

Natrénovaná data už máme a nyní je odzkoušíme na testovacích datech. Popíšeme si parametry a funkce programu pro detekci objektů s názvem `detect`. Všechny zde vypsané jsou volitelné:

`-boost` soubor	- parametr pro zadání cesty k souboru s klasifikátory, defaultně je nastaven soubor „boost.xml“
`-features` soubor	- parametr pro zadání cesty k souboru s klasifikátory, defaultně nastaven soubor „features.xml“
`-save` adresář	-nastaví adresář, kam se budou ukládat výstupní snímky
`-list` soubor	- bude postupně detekovat všechny obrázky ze souboru
`-noshowresults`	- parametr vhodný ke kombinaci s parametrem `–save`, výsledky se nezobrazují, jen se ukládají
`-h`	- vypíše nápovědu k programu
Soubor	- soubor určený k detekci, je možné jakékoliv množství těchto parametrů

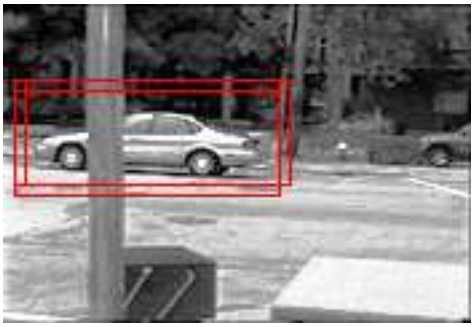
Program `detect` se dá spustit i bez jakéhokoliv parametru. Jakmile to ale zjistí, nabídne uživateli možnost zadat cestu k souboru pro detekci až za běhu programu.

Jak bylo již popsáno v kapitole 3.1, detekce nám může dávat spoustu kladných odezev (falešných detekcí). Příklad těchto falešných detekcí je zde:



Obrázek 3.4 : Výstup detekce bez redukce falešných detekcí

Ukážeme si, jak může vypadat, když tyto falešné detekce redukujeme:



Obrázek 3.5 : Výstup detekce s redukcí falešných detekcí

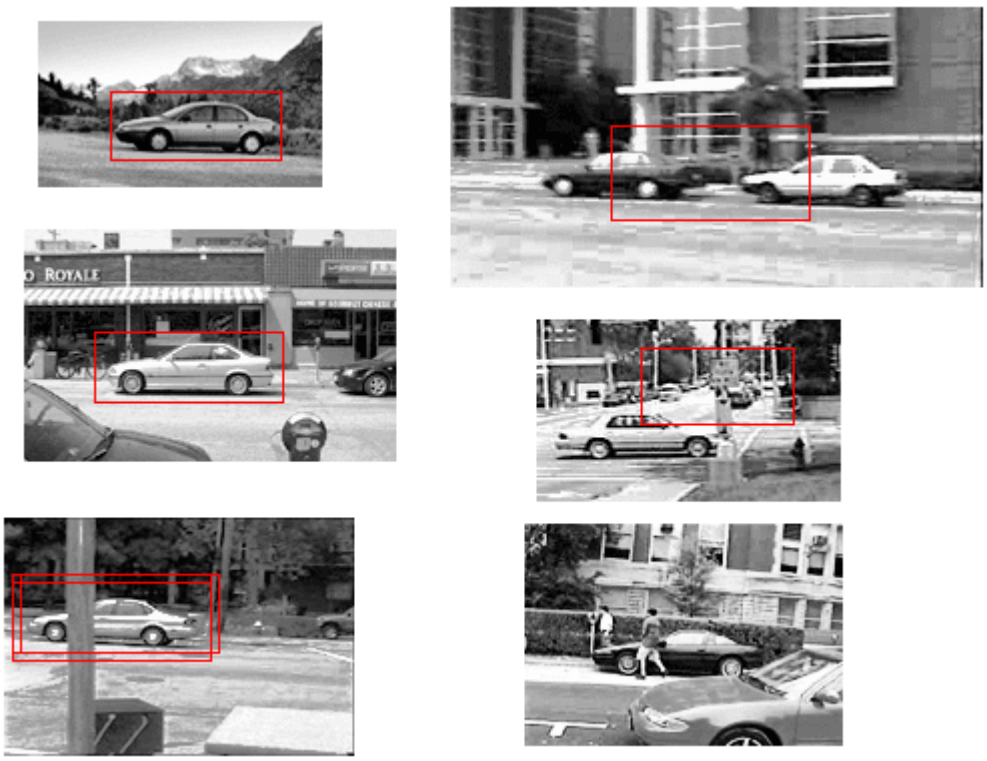
3.4 Soubor pro ukládání příznaků

Pro zrychlení vytváření struktury příznaků, při načítání programu, si ukládám všechny vygenerované příznaky do souboru. Odtud si je potom kdykoliv načtu bez pracného generování. Toto je ukázka obsahu souboru s příznaky. Obsahuje dva příznaky (jeden je linka a druhý osmina oblouku. Linkový příznak má dvě oblasti a obloukový tři):

```
<?xml version="1.0"?>
<opencv_storage>
<f1>
  <t>1</t>
  <rt>2</rt>
  <srw>2</srw>
  <a>0.</a>
  <l>3</l>
  <R1>
    <X>5 6 7 5 6 7</X>
    <Y>5 5 5 6 6 6</Y>
  </R1>
  <R2>
    <X>5 6 7 5 6 7</X>
    <Y>7 7 7 8 8 8</Y>
  </R2>
 </f1>
<f2>
  <t>4</t>
  <rt>3</rt>
  <srw>2</srw>
  <a>67.50000000000000000</a>
  <l>7</l>
  <R1>
    <X>82 83 84 85 86 87 </X>
    <Y>18 18 18 18 18 18 </Y>
  </R1>
  <R2>
    <X>83 84 85 86 87</X>
    <Y>20 20 20 20 20</Y>
  </R2>
  <R3>
    <X>84 85 86</X>
    <Y>23 23 23</Y>
  </R3>
 </f2>
</opencv_storage>
```

4 Výsledky

Detektor byl testován na 170 testovacích obrázcích a z toho správně určil 124 jako auto a zbytek byli špatné detekce nebo auto vůbec nenašel. Jeho úspěšnost byla 73 %. Zde uvádím několik obrázků dobrých, špatných i žádných detekcí:



Obrázek 4.1 : Ukázka výsledků detekce

5 Závěr

Jak jsem se v této práci přesvědčil, algoritmus Real AdaBoost je dobrou volbou v trénování dat, a detekci objektů samotné. Bohužel, ale v kombinaci s mnou vytvořenými příznaky se rychlost detekce zhoršila. Možná to bylo i pomalejším počítačem, protože celkový počet příznaků, které jsem na začátku vytvořil, bylo přes 200 000 a to jsem už příznaky omezoval. Celkově se mě je podařilo omezit, a s udržením docela dobrých výsledků, až na 24 731. Myslím si ale, že by tento počet šel ještě více snížit a to úpravou generování příznaků tak, aby lépe vystihovali geometrické tvary automobilu. Dalším zrychlením by se podle mě dalo dosáhnout použití jiné metody učení. Real AdaBost je velmi dobrý, ale jeho nevýhoda je v tom, že se pokaždé musí vyhodnocovat všechny příznaky. A to značně zpomaluje. Uznejte sami, na obrázku o rozměrech 140x120 pixelů můžeme aplikovat posuvné okno o rozměrech 64x32 pixelů dokonce až 6864 výřezů a na každý musím aplikovat všechny příznaky.

Mým závěrem tedy je, že aplikace geometrických příznaků s použitím metod trénování je velmi přínosná a při jejím dalším vývoji by mohla byt používána i v real-time aplikacích.

Literatura

[1] *Wikipedie: Otevřená encyklopedie: Segmentace obrazu*[online]. c2010 [citováno 2.5.2010].Dostupný z WWW: <http://cs.wikipedia.org/iwik/Segmentace_obrazu>

[2] *Wikipedie: Otevřená encyklopedie: Detekce hran* [online]. c2010 [citováno 4.5.2010].Dostupný z WWW: <http://cs.wikipedia.org/wiki/Detekce_hran>

[3] Bc. Juránek R., *Rozpoznávání vzorů v obraze pomocí klasifikátorů*, diplomová práce, Brno, FIT VUT v Brně, 2007

[4] Šochman Jan, Matas Jiří, *Adaboost*, cvičení, Praha, fakulta elektrotechnická ČVUT, Katedra kybernetiky, Centrum strojového vnímání, 2005 <http://cmp.felk.cvut.cz/cmp/courses/recognition/Labs/adaboost/adaboost.pdf>

[5] *Wikipedie: Otevřená encyklopedie: Machine leasing* [online]. c2010 [citováno 10.5.2010]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Machine_learning>

[6] *Wikipedie: Otevřená encyklopedie: Support Vector Machine*[online]. c2010 [citováno 11.5.2010]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Support_Vector_Machine>

[7] *Wikipedie: Otevřená encyklopedie: AdaBoost* [online]. c2010 [citováno 23.11. 2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/AdaBoost>

[8] *Wikipedie: Otevřená encyklopedie: Boosting* [online]. c2010 [citováno 12.1. 2010]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Boosting>

[9] *Wikipedie: Otevřená encyklopedie: Neuronová síť* [online]. c2010 [citováno 8. 5. 2010]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=Neuronov%C3%A1_s%C3%AD%C5%A5&oldid=5338992>

[10] Wei Zheng; Luhong Liang; , "Fast car detection using image strip features," *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* , vol., no., pp.2703-2710, 20-25 June 2009, doi: 10.1109/CVPR.2009.5206642 Dostupné z WWW: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5206642&isnumber=5206488>

Seznam obrázků

Seznam příloh