VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

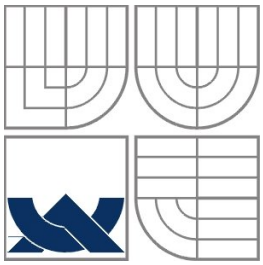
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ VZORŮ V OBRAZE POMOCÍ KLASIFIKÁTORŮ

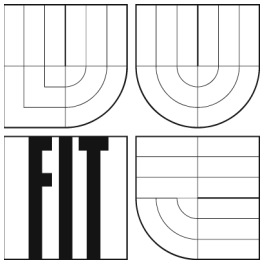
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ROMAN JURÁNEK
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNÉ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ VZORŮ V OBRAZE POMOCÍ KLASIFIKÁTORŮ

PATTERN RECOGNITION IN IMAGE USING CLASSIFIERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ROMAN JURÁNEK
AUTHOR

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

BRNO 2007

Zadání diplomové práce

Řešitel: Roman Juránek, Bc.

Obor: Počítačová grafika a multimedia

Téma: Rozpoznávání vzorů v obraze pomocí klasifikátorů

Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte algoritmy Ada/Wald-Bost případně další.
2. Důkladně popište studované algoritmy.
3. Vytvořte aplikaci pro experimentování s klasifikátory a pro vyhodnocení jejich vlastností.
4. Vyhodnoťte a popište vlastnosti studovaných algoritmů, srovnejte s jinými algoritmy rozpoznávání vzorů v obraze.
5. Demonstruje použití a použitelnost algoritmů v praktických úlohách.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Roman Juránek

Bytem: Sofijská 2797, Tábor 390 05

Narozen/a (datum a místo): 7.7.1983, Šumperk

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta ...Informačních Technologií.....

se sídlem ...Božetěchova 2, 612 66 Brno.....

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

Doc. Dr. Ing. Pavel Zemčík.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - ☒ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP:	Rozpoznávání vzorů v obraze pomocí klasifikátorů
Vedoucí/ školitel VŠKP:	Ing. Adam Herout, Ph.D.
Ústav:	Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP:	21.6.2007

VŠKP odevzdal autor nabyvateli v*:

- ☒ tištěné formě – počet exemplářů ...2.....
- ☒ elektronické formě – počet exemplářů ...2.....

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

V této práci bude představen algoritmus AdaBoost, který slouží k vytvoření silné klasifikační funkce z několika slabých hypotéz. Bude vyloženo teoretické pozadí algoritmu a způsob konstrukce silného klasifikátoru. Dále bude popsáno rozšíření algoritmu o sekvenční rozhodovací strategii nazývané WaldBoost. Práce se zabývá také obrazovými příznaky, které jsou v mnoha případech základem slabých klasifikátorů. Kromě popisu zmíněných algoritmů bude uveden základ rozpoznávání vzorů v kontextu počítačového vidění a budou uvedeny některé často používané metody trénování klasifikátorů. Součástí práce bylo vytvoření knihovny pro detekci objektů založené na klasifikátorech trénovaných metodou AdaBoost. Tato knihovna byla následně využita v implementaci programu, který prakticky demonstruje detekce obejktů ve videosekvencích. Poslední část práce popisuje nástroj pro trénování AdaBoost klasifikátorů.

Klíčová slova

Rozpoznávání vzorů, AdaBoost, WaldBoost, Klasifikace, Detekce, Obrazové Příznaky

Abstract

An AdaBoost algorithm for construction of strong classifier from several weak hypotesis will be presented in this work. Theoretical background of the algorithm and the method of construction of strong classifiers will be explained. WaldBoost extension to the algorithm will be described. The thesis deals with image features that are often used as element of weak classifiers. Brief introduction to pattern recognition in context of computer vision will be outlined in the begining of the work. Also some widely used methods of classifier training will be presented. An object detection library based on AdaBoost classifiers was developed as part of the work. The library was used in implementation of software that in praktice demonstrates object detection in videosquences. Last part of the work describes tool for training of AdaBoost classifiers.

Keywords

Pattern recognition, AdaBoost, WaldBoost, Classification, Detection, Image Features

Citace

Bc. Juránek R., Rozpoznávání Vzorů v Obraze Pomocí Klasifikátorů, diplomová práce, Brno, FIT VUT v Brně, 2007

Rozpoznávání Vzorů v Obraze Pomocí Klasifikátorů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Poděkování

Za veškeré podněty a trpělivé konzultace děkuji především vedoucímu diplomové práce Ing. Adamu Heroutovi, Ph.D. Dále také za a odbornou pomoc a jiné cenné rady při zpracování této práce.

Obsah

Seznam obrázků

1 Úvod

Oblast počítačového vidění (*computer vision*) je poměrně mladá v porovnání s některými jinými oblastmi zájmu v informačních technologiích. Větší rozvoj zde započal až koncem sedmdesátých let dvacátého století, kdy začaly být počítače dostatečně výkonné pro zpracování velkých objemů dat jako například obrazů.

Neexistuje žádná metoda jak obecně řešit problémy počítačového vidění, jsou jen více či méně vyvinuté metody řešení určitých přesně definovaných úloh. Mnoho metod je nyní ve fázi výzkumu, některé další si našly cestu ke komerčnímu využití, kde jsou součástí větších systémů (zpracování medicínksých dat, kontrola kvality, měření, ...). V současné době je většina praktických aplikací předprogramována pro řešení určitého problému, ale adaptivní metody založené na učení začínají být stále běžnější.

Podmnožinou počítačového vidění je rozpoznávání vzorů (*pattern recognition*), které se zabývá extrahováním informací ze signálů obecně. V mnoha případech se jedná o statistické metody založené na učení s učitelem (*supervised learning*). Tato práce se zabývá rozpoznáváním vzorů založeným na obrazových klasifikátorech a zaměřuje se hlavně na algoritmus AdaBoost.

Druhá kapitola se zabývá obecně problematikou rozpoznávání vzorů, klasifikace a vyhodnocením chyby klasifikátorů. Popisuje fungování neuronových sítí a zmiňuje některé jejich typy. Dále jsou zde zmíněny support vector machines a gaussovské modely. Ve 3. kapitole je z teoretického hlediska popsána metoda AdaBoost. Detailně se zde rozebírá trénovací algoritmus. Dále popisuje obrazové příznaky a zabývá se zejména Haarovými vlnkami, které jsou často základem slabých klasifikátorů. Rozebírá také algoritmus WaldBoost. 4. kapitola popisuje implementaci knihovny, která používá klasifikátory natrénované algoritmem AdaBoost a jednoduchý program používající API této knihovny pro detekci objektů. Předposlední kapitola popisuje program pro trénování AdaBoost klasifikátorů. V poslední kapitole jsou uvedeny některé experimenty, které byly s provedeny.

2 Rozpoznávání vzorů v obraze

Oblast rozpoznávání vzorů má za úkol získat informace ze signálu na základě nějaké znalosti nebo statistických informací. Klasifikované signály bývají obyčejně měření reprezentovaná v multidimensionálním prostoru vektorem příznaků (*feature vector*). Obecný klasifikační systém se skládá ze tří částí – získání dat, extrakce příznaků a klasifikace. Výsledek klasifikace záleží tedy především na kvalitě naměřených dat a příznaků z nich získaných. V případě počítačového vidění naměřená data představuje obraz (např. fotografie, obraz z kamery, ...) a příznaky jsou funkce počítané nad tímto obrazem. Klasifikace je použitelná obecně v prakticky libovolné úloze. Klasifikační metody lze nalézt například v dolování dat nebo zpracování řeči, kde se takto rozpoznávají fonémy a slova nebo se určuje jazyk, kterým je řeč mluvena. V počítačovém vidění má klasifikace za úkol zařadit obraz do jedné z definovaných tříd a tak identifikovat, zda se v obraze vyskytuje hledaný objekt. V binárních úlohách se tedy obvykle vyskytují dvě třídy – *hledaný objekt* a *pozadí*. Některé úlohy ale vyžadují větší počet klasifikačních tříd (např. OCR – každý znak má svoji třídu). Třída hledaných objektů není nijak omezena a hledat lze téměř cokoliv, např. určitý geometrický tvar, poznávací značka auta, znak, atd. Častá úloha pro použití klasifikátorů je detekce obličeje.

Existuje mnoho metod klasifikace. V praktických aplikacích je při výběru klasifikační metody nutné zohlednit kritéria jako množství dostupných trénovacích dat, dostupné výpočetní prostředky pro trénování a běh aplikace, atd. A podle nich zvolit použitou metodu, protože každá má jiné nároky a mají i různě kvalitní výstup.

Použití klasifikátorů

Klasifikace jako metoda rozpoznávání vzorů je použitelná v mnoha komerčních i průmyslových aplikacích. V mnoha případech se jedná o systémy kde je klasifikace poze jako jedna součást a celek využívá řadu dalších metod. Mnoho webkamer nebo digitálních fotoaparátů ma v sobě jednoduchý detektor obličeje, který používají pro automatické zaostřování, nastavení expozice (AF/AE) a odstranění červených očí. Další využití rozpoznávání vzorů lze nalézt v OCR programech, které musí na digitalizovaném dokumentu rozpoznat jednotlivá písmena (a řezy písma). OCR software se dnes běžně dodává k většině scannerů. Z průmyslových aplikací jde například o kontrolu dopravy, kde se klasifikace používá například při detekci a rozpoznávání poznávacích značek aut. Další využití lze nalézt v průmyslové kontrole kvality výrobků.

2.1 Trénování klasifikátoru

Obyčejně se trénování klasifikátoru provádí učením s učitelem (*supervised learning*). Jde o metodu zjišťování klasifikační funkce ze známých dat, u kterých víme, do které třídy patří (*ground truth*). Klasifikátor se tak podle dat, která během učení *viděl* naučí předpovědět třídu klasifikace pro neznámá data – tzv. generalizovat. Je zřejmé, že při učení nelze v datech popsat všechny možné případy vstupu (až na některé velmi speciální případy), které má klasifikátor umět zařadit a výsledek tedy bude mít vždy určitou chybu. Při trénování se s použitím co nejlepších trénovacích dat snažíme aby tato chyba byla co nejmenší. Platí, že se snižující se chybou roste komplexnost klasifikátoru a tady i čas potřebný pro jeho trénování a vyhodnocení. Jednodušší klasifikátory mají na druhou stranu chybu vyšší.

Při trénování se používají vzorky dat. Existují dvě sady vzorků – trénovací a testovací. Na trénovací sadě se učí klasifikační funkce a na testovací sadě se ověřuje její chyba. Obecně lze říci, že trénovací data obsahují obrazy objektů a jejich zařazení do třídy. Například v úloze detekce obličejů jsou dvě třídy objektů – *obličej* a *pozadí*. Třída *obličej* obsahuje obrázky různých obličejů za různých podmínek (obr. 1). Ve třídě *pozadí* je vše, co nemá klasifikátor považovat za detekovaný objekt. Čím lepší sadu vzorků reprezentující podmínky při použití klasifikátoru máme k dispozici, tím lepší klasifikátor lze natrénovat. Trénovací sada se někdy rozděluje na dvě části – vlastní trénovací sada a validační sada. Druhá jmenovaná se používá pro detekci přetrénování klasifikátoru.

Obr. 1: Část množiny vzorků pro detekci obličeje.

Před spuštěním trénování jsou ze vstupních dat extrahovány příznaky a vytvořen příznakový vektor – ten se v průběhu nemění. Samotné trénování je iterační proces, který lze popsat následujícím kroky.

1. *Klasifikace vzorků*
   Příznakový vektor se použije ke klasifikaci vzorků aktuální verzí klasifikátoru. Takto se zjistí jeho chyba (viz. kap. 2.2).
2. *Úprava klasifikátoru podle chyby*
   Trénovací algoritmus se pokusí opravit klasifikační funkci podle chyby klasifikace z předchozího kroku.

Tyto kroky se opakují dokud není splněno kritérium zastavení. Nejjednodušší kritérium je počet trénovacích kroků. Trénování se zastaví po dosažení stanoveného počtu iterací. Dalším kritériem může být velikost chyby. Trénuje se, dokud je chyba vyšší než stanovená hranice. Jiná možnost je, že pokles chyby mezi po sobě následujícími kroky musí být vyšší než určitá hodnota. Trénuje se tedy dokud klesá chyba na celé trénovací sadě. Metoda, která dosahuje nejlepších výsledků, je použití validační sady vzorků. Pokud chyba na trénovací i validační sadě klesá, lze trénovat dál. Ve chvíli, kdy začne chyba na validační sadě stoupat, znamená to, že klasifikátor začíná být přetrénovaný a je třeba trénování zastavit.

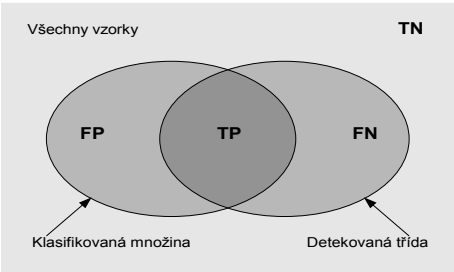
2.2 Vyhodnocení chyby klasifikátoru

Chyba klasifikátoru se určuje na testovací sadě vzorků u kterých je známo, do které třídy patří. Jde v podstatě o zjištění kolikrát se klasifikátor *netrefil* při určení třídy vzorku. Zjišťují se tyto hodnoty (obr. 2) *True Positive* (TP, správné přijetí), *True Negative* (TN, správné odmítnutí), *False Positive* (FP, chybné přijetí) a *False Negative* (FN, chybné odmítnutí). Vztahy (1) ukazují výpočet těchto hodnot.

$$\begin{aligned} TP &= \sum_i H(x_i) = y_i, \quad y_i = +1 \\ TN &= \sum_i H(x_i) = y_i, \quad y_i = -1 \\ FP &= \sum_i H(x_i) \neq y_i, \quad y_i = +1 \\ FN &= \sum_i H(x_i) \neq y_i, \quad y_i = -1 \end{aligned} \tag{1}$$

Kde $H(x)$ je klasifikační funkce, x_i vzorek a y_i jeho ohodnocení. Z těchto informací lze získat hodnotu chyby (2). Chyba je poměr chybně klasifikovaných vzorků k celkovému počtu vzorků.

$$error = \frac{FP + FN}{TP + TN + FP + FN} \tag{2}$$



Obr. 2: Míry správných nebo chybných přijetí a odmítnutí.

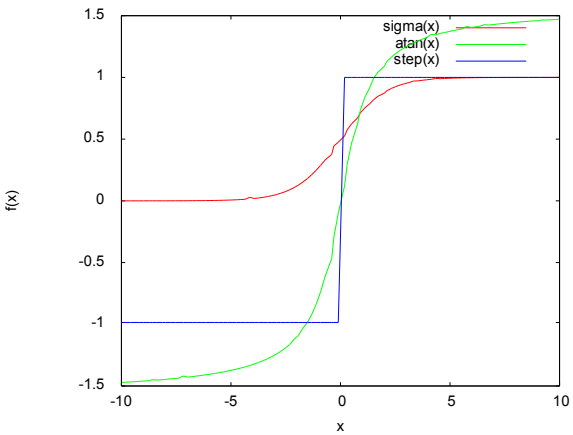
Nejběžnějším způsobem vyhodnocení kvality klasifikátoru je v současné době ROC křivka (*Receiver Operating Characteristic*). Jde o závislost míry *false positives* na *false negatives* při různých nastaveních klasifikátoru. Na této křivce lze určit hodnotu EER (*Equal Error Rate*), kdy má klasifikátor stejnou míru chybných přijetí a chybných odmítnutí.

2.3 Neuronové sítě

Myšlenka neuronových sítí byla inspirována funkcí mozku, který se skládá z různě propojených neuronů, které přenášejí vzruchy. Umělé neuronové sítě jsou tvořené množstvím propojených uzlů – umělých neuronů. Neurony jsou propojeny tak, aby síť jako celek prováděla určitou funkci. Jsou založené na spoluprací mnoha uzlů, které dohromady tvoří jeden systém. Jedinečná vlastnost těchto sítí je odolnost proti poruchám. I když přestane pracovat několik neuronů, celková funkce sítě zůstane v podstatě neporušena a může vykovávat svoji činnost (s určitou nepřesností způsobenou chybějícími uzly). Využití neuronových sítí je mnohem širší než oblast rozpoznávání vzorů a klasifikace. Jedná se například o aproximaci funkcí, predikci, shlukování, dolování dat, atd.

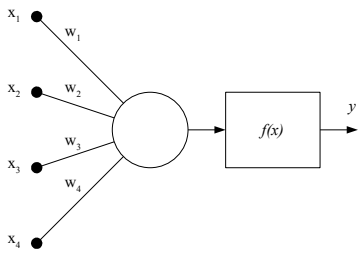
Neuron

Umělý neuron (obr. 4) je jednoduchá funkce provádějící skalární součin vektoru svých vstupů a vektoru vstupních vah. Výsledná hodnota je nakonec transformována výstupní (aktivační) funkcí. Při trénování neuronové sítě se provádí nastavení vektoru vah u každého neuronu tak, aby síť jako celek prováděla požadovanou funkci.



Obr. 3: Některé používané aktivační funkce.

$$y = f\left(\sum_{i}^{m} w_i x_i\right) = f\left(\vec{w} \cdot \vec{x}\right) \tag{3}$$



Obr. 4: Model umělého neuronu se čtyřmi vstupy.

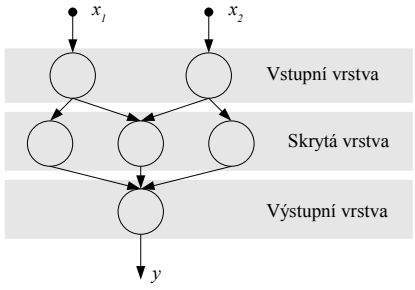
Výstupní hodnota neuronu (3) je definována vektorem vstupů, vektorem vah a výstupní funkcí *f(x)*. V mnoha případech se jako výstupní funkce volí *sigmoida* (obr. 3). Další používaná aktivační funkce je např. *step*. Důležitá vlastnost aktivační funkce je její nelinearita.

Typy neuronových sítí

Základním typem sítě je dopředná síť. Vyznačuje se tím, že informace proudí od vstupů přímo k výstupům a nejsou v ní tedy žádné smyčky. Nejjednodušším druhem dopředné sítě je jednovrstvý perceptron. Jedná se pouze o jednu vrstvu výstupních neuronů. Tento typ sítě nedokáže vyřešit například XOR problém, který je řešitelný až vícevrstvou sítí.

Vícevrstvý perceptron (obr. 5). Skládá se z několika vrstev neuronů typicky propojených tak, že výstupy jedné vrstvy jsou připojené na vstupy vrstvy následující. Vrstvy lze rozdělit do tří typů - vstupní, skryté a výstupní. Síť obsahuje jednu vstupní a výstupní vrstvu a jednu nebo více skrytých vrstev. Tento typ sítě dokáže aproximovat libovolnou reálnou funkci $f(x)$ definovanou na intervalu $\langle 0;1\rangle^n$ (4) libovolně přesně [1]. $f'(x)$ je výstupní hodnota sítě.

$$|f'(x)-f(x)|<\epsilon,\quad \epsilon>0, x\in\mathbb{R}^n \tag{4}$$



Obr. 5: Vícevrstvá síť.

Jiným typem sítí jsou rekurentní sítě. Na rozdíl od dopředných sítí, ve kterých data proudí od vstupů k výstupům, je zde možné nalézt zpětné vazby. Jedním druhem reku-

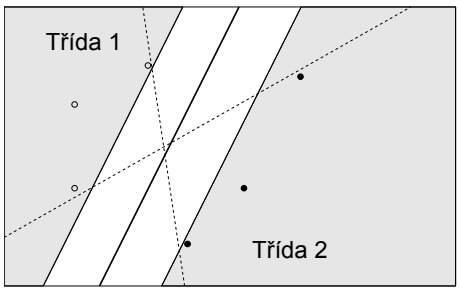
rentní sítě je Hopfieldova síť. Obsahuje skupinu neuronů, které jsou vzájemně propojeny každý s každým. Tato síť funguje jako asociativní paměť a je tedy vhodná pro úlohy jako restaurace poškozené informace nebo rozpoznávání.

Trénování

U dopředných sítí se pro trénování často používá technika zpětného šíření chyby (*back-propagation*). Algoritmus pracuje následujícím způsobem [2]. Na vstupní vrstvu se postupně vkládají vzorky trénovacích dat a výstup sítě se porovnává s požadovaným výstupem. V každém výstupním neuronu se vypočítá lokální chyba. Nastavením vstupních vah neuronu se chyba sníží na minimum – tímto se zvyšuje chyba neuronů v předchozí vrstvě Postup se opakuje postupně s neurony předchozích vrstev dokud se nedojde ke vstupní vrstvě.

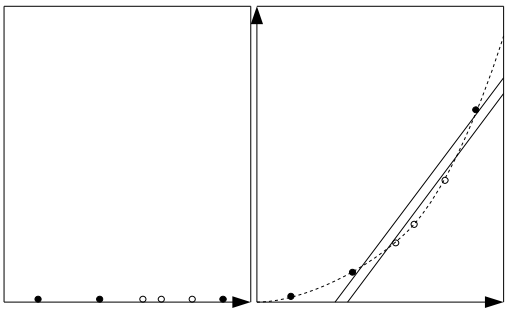
2.4 Support vector machine

Support vector machine (SVM) je metoda klasifikace lineárních dat [3]. Velmi často se používají například v aplikacích pro vyhledávání obrazů (*image retrieval*). V případě klasifikace nelineárních dat je možné data transformovat do prostoru s vyšší dimenzí a klasifikovat je jako lineární. Trénovací proces se za pomocí trénovacích dat snaží nalézt hyperplochu nejlépe rozdělující data v prostoru (*maximum margin*). Na obrázku 6 jsou data ze dvou tříd. Čárkované čáry jsou hyperplochy s minimální dělící vzdáleností. Plnou čárou je zvýrazněna hyperpolocha, která data dělí nejlépe.



Obr. 6: Ukázka maximalizace vzdálenosti dělící hranice od dat.

Data, která není možné lineárně separovat, je možné nejprve transformovat do prostoru s vyšší dimenzí pomocí jádra (*kernel*). Transformovaná data pak lze lineárně rozdělit. Na obrázku 7 vlevo jsou jednorozměrná data, pro která není možné nalézt jednu dělící hyperplochu.

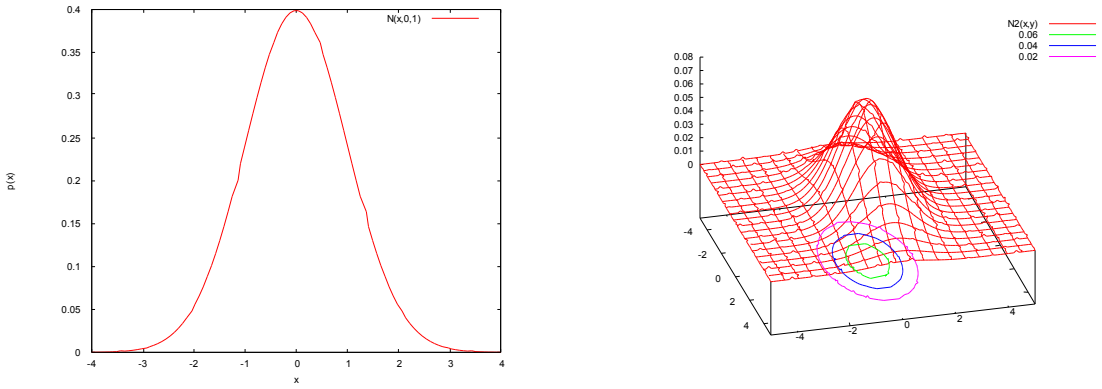


Obr. 7: Lineárně neseparovatelná data a jejich transformace.

Vpravo jsou data transformována do dvou rozměrů s naznačenou dělící plochou. Použití jader umožňuje i vícetřídní klasifikaci nelineárních dat v mnoharozměrném vektorovém prostoru příznaku při zachování vysoké přesnosti klasifikátoru [4]. Nevýhoda SVM je, že jádra a parametry je nutné pro konkrétní úlohu zvolit ručně.

2.5 Gaussovské modely

V některých případech se ke klasifikaci používá statistické rozpoznávání vzorů. To je založeno na předpokladu, že objekty z jedné třídy budou mít navzájem podobné určité statistické vlastnosti. Jednou statistickou technikou je modelování hledaných tříd pomocí směsi gaussovských funkcí ve vektorovém prostoru příznaků (G*aussian Mixture Models, GMM*). Využití této techniky je široké. Používá se například ve zpracování řeči pro detekci fonémů. Ve zpracování obrazu se tato metoda používá velmi často k detekci částí lidského těla pomocí barvy kůže – klasifikace barev do tříd *skin-color*/*background*.



Obr. 8: Normální rozožení v jednorozměrném a dvourozměrném prostoru

Normální rozložení v jednorozměrném prostoru je definováno svým středem μ a rozptylem σ (5). Levá část obrázku 8 ukazuje rozložení středem 0 a rozptylem 1. Pomocí

něj lze modelovat pravděpodobnost, že nějaký jev hodnotou příznaku x náleží do modelované třídy. V pravé části je dvourozměrný případ s rozptylem 1 (ve směru x) a 2 (ve směru y).

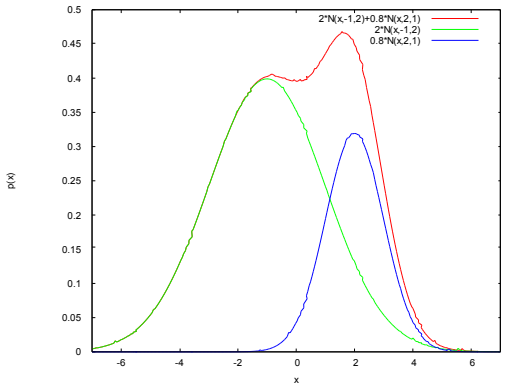
V případě vícerozměrného prostoru, kdy modelujeme jev s větším počtem příznaků, lze pravděpodobnost modelovat Gaussovým rozložením jehož parametry jsou vektor středních hodnot $\vec{\mu}$ a kovarianční matice (matice rozptylů) $\boldsymbol{\Sigma}$ (6).

$$N(x;\sigma,\mu)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5}$$

$$N(\vec{x};\vec{\mu},\boldsymbol{\Sigma})=\frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T\boldsymbol{\Sigma}^{-1}(\vec{x}-\vec{\mu})} \tag{6}$$

Vztah (6) 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 (nekorelované příznaky, obr. 8 vpravo) má definované rozptyly pouze ve směru jednotlivých os – jen na diagonále matice. Výpočet pravděpodobnosti lze v takovém případě zjednodušit na násobení jednorozměrných distribucí (7). Tímto lze podstatně urychlit výpočet pravděpodobnosti za cenu, že příznaky je před výpočtem pravděpodobnosti nutné dekorelovat například pomocí DCT.

$$N(\vec{x};\vec{\mu},\vec{\sigma})=\prod_i N(x_i;\mu_i,\sigma_i) \tag{7}$$



Obr. 9: Dvě distribuce a jejich součet.

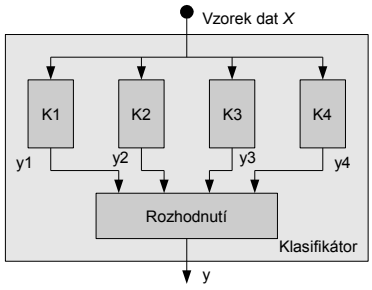
V praxi se data modelují součtem více Gaussových distribucí. Jejich počet závisí na složitosti rozmístění příznaků v prostoru. S počtem použitých distribucí roste čas potřebný ke trénování a také přesnost výsledného klasifikátoru. Parametry distribucí je

nejprve nutné odhadnout z trénovacích dat. Velmi často se používá MLE (*Maximum Likelihood Estimation*) odhad parametrů, které maximalizují celkovou likelihood na trénovacích datech [5]. Dále pak algoritmus EM (*Expectation Maximization*), nebo diskriminativní trénovací algoritmy. MMI (*Maximum Mutual Information*), MCE (*Minimum Classification Error*) nebo MAP (*Maximum a-posteriori*).

Pokud jsou odhadnuty parametry distribucí je možné pro libovolný bod v prostoru příznaků spočítat pravděpodobnosti pro jednotlivé třídy a bod zařadit do té, která má pravděpodobnost nejvyšší.

3 AdaBoost

Algoritmus AdaBoost (*Adaptive Boosting*) byl představen v roce 1995. Jeho hlavním přínosem je schopnost exponenciálně snižovat chybu výsledného klasifikátoru na libovolně nízkou úroveň (s danou množinou vzorků a slabých klasifikátorů). Formální důkaz tohoto faktu lze nalézt v článku [6]. AdaBoost dokáže produkovat v relativně krátkém trénovacím čase klasifikátory s velmi malou chybou i za použití jen velmi jednoduchých slabých klasifikátorů. Takové klasifikátory pak mají výhodu, že jsou při použití v reálných podmínkách velice přesné a jejich vyhodnocení lze provést ve velmi krátkém čase. Ukazuje se, že jsou vhodné pro úlohy jako například detekce obličeje ve videosekvenci v reálném čase [7, 8].



Obr. 10: Struktura silného klasifikátoru.

Základní varianta algoritmu [6, 7] je určena ke kombinování několika klasifikátorů do jednoho tak, aby výsledná klasifikační funkce byla přesnější než všechny použité klasifikátory. Výsledkem je silný klasifikátor (*strong classifier*), který se skládá z několika slabých klasifikátorů (*weak classifier*). Slabé klasifikátory mohou mít libovolnou složitost, ale v mnoha případech se volí jen jednoduché funkce. Algoritmus každému přiřadí váhu na základě jeho chyby. Varianty, kterými se zabývají následující podkapitoly jsou omezeny na použití slabých klasifikátorů, které se skládají pouze z jednoho obrazového příznaku.

3.1 Algoritmus

Základním úkolem algoritmu je vybrat z velkého množství slabých klasifikátorů malou podmnožinu tak, aby tyto klasifikátory co nejlépe rozdělovaly dané vzorky do svých tříd. V článku od autorů Freund a Schapire [6] je popsána základní varianta algoritmu pro klasifikaci do dvou tříd. Článek také obsahuje dvě rozšíření algoritmu pro klasifikaci do více tříd.

Vstupem algoritmu je sada trénovacích vzorků *x* a jejich ohodnocení *y* (třída, do které náleží) – $(x_1, y_1)\ldots(x_m, y_m)$, kde každé $x \in X$ je instance hledaného vzoru a $y \in Y$ je jeho ohodnocení. V našem případě platí, že $Y = \{-1, +1\}$. Hodnotou 1 jsou označeny hledané objekty, -1 mají vzorky protipříkladů. Hlavní myšlenkou algoritmu je, že uchovává distribuci trénovacích vzorků neboli jejich váhu. Díky této distribuci se algoritmus přizpůsobuje těžko klasifikovatelným vzorkům v trénovacích datech a postupně opravuje svou funkci tak, aby i tyto vzorky dokázal správně zařadit. Váha vzorku *i* v trénovacím kroku *t* je $D_t(i)$. Na začátku jsou váhy všech vzorků nastaveny stejně. V každém dalším kroku je váha chybně klasifikovaných vzorků zvýšena a u správně klasifikovaných snížena. To dovoluje algoritmu opravovat rozhodnutí u chybně klasifikovatelných vzorků.

Úkolem algoritmu je v každém kroku nalézt právě jeden slabý klasifikátor z dané množiny, který pro distribuci $D_t(i)$ nejlépe klasifikuje vzorky v trénovací sadě. Slabý klasifikátor je funkce $h_t : X \to \{-1, +1\}$, která každému vzorku z množiny *X* přiřadí jeho ohodnocení. Vhodnost klasifikátoru h_t se zjišťuje pomocí jeho chyby ϵ_t. Jedná se o součet vah špatně klasifikovaných vzorků.

$$\epsilon_t = \sum_{i : h_t(x_i) \neq y_i} D_t(i) \tag{8}$$

Pro každý slabý klasifikátor h_t vypočítán parametr α_t, který znamená důležitost klasifikátoru – čím nižší je jeho chyba, tím vyšší je α_t. Aktualizace distribuce pro další iteraci algoritmu (krok 3 v následujícím pseudokódu) má za úkol zvýšit důležitost u chybně klasifikovaných vzorků a snížit u správně klasifikovaných. Tímto se algoritmus zaměřuje na obtížné vzorky v trénovací sadě.

Celý algoritmus lze pak zapsat pseudokódem.

Vstup: $(x_1, y_1)\ldots(x_m, y_m)$, $x \in X$, $y \in \{-1, +1\}$

Inicializace: $D_1 = \frac{1}{m}$ (pro všechny vzorky)

Pro t = 1 … T

1. Učení klasifikátorů. Nalezení optimálních parametrů (pokud nějaké má) každého slabého klasifikátoru tak, aby měl co nejmenší chybu ϵ_j na aktuální distribuci *w*.
2. Nalezení klasifikátoru s nejmenší chybou ϵ_t pro distribuci D_t. $h_t : X \to \{-1, +1\}$

3. Výpočet váhy klasifikátoru $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
4. Aktualizace vah vzorků v trénovací množině

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

   Kde Z_t je normalizační faktor zvolený tak, aby funkce D_{t+1} zůstala pravděpodobnostním rozložením.

Výsledkem je lineární klasifikátor. Funkce $H(x): X \to \{-1, +1\}$ je po T krocích trénování vyhodnocena jako součet odezev klasifikátorů násobených jejich váhou. Lze ji vyjádřit i vektorově jako skalární součin $\vec{\alpha}\cdot\vec{h}$, kde $\vec{\alpha}$ je vektor vah slabých klasifikátorů a $\vec{h}$ je vektor jejich odezev.

$$H(x) = \mathrm{sgn}(\vec{\alpha}\cdot\vec{h}) = \mathrm{sgn}\left(\sum_{t=0}^{T}\alpha_t h_t(x)\right) \qquad (9)$$

Je zřejmé, že chyba výsledného klasifikátoru v kroku *t+1* je v nejhorším případě stejná jako v kroku *t*. Pokud lze nalézt slabý klasifikátor takový, že jeho chyba je nižší než 0.5 (což odpovídá náhodné funkci), chyba vždy klesá.

Viola-Jones

Jiná forma algoritmu od autorů Viola a Jones [7]. Vstupem této modifikace algoritmu jsou opět vzorky a jejich ohodnocení $(x_1, y_1)\ldots(x_m, y_m)$. Kde $x \in X$, $y \in \{0,1\}$. Oproti předchozí variantě se liší v rozdílné inicializaci distribuce vah *w* a jiné aktualizaci vah na konci iterace. Rozložení *w* zde nemusí být skutečným rozložením pravděpodobnosti a proto se na začátku každé iterace normalizuje.

Inicializace: $w_{1,i} = \frac{1}{2\mathrm{m}}, \frac{1}{2\mathrm{l}}$ pro $y_i = 0, 1$, kde *m* a *l* je počet negativních, resp. pozitivních vzorků.

Pro t = 1 ... T

1. Normalizace vah, aby *w* bylo pravděpodobnostní rozložení.

   $$w_{t,i} = \frac{w_{t,i}}{\sum_j w_{t,j}}$$

2. Učení klasifikátorů. Nalezení optimálních parametrů (pokud nějaké má) každého slabého klasifikátoru tak, aby měl co nejmenší chybu ϵ_j na ak-

tuální distribuci *w*.

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$$

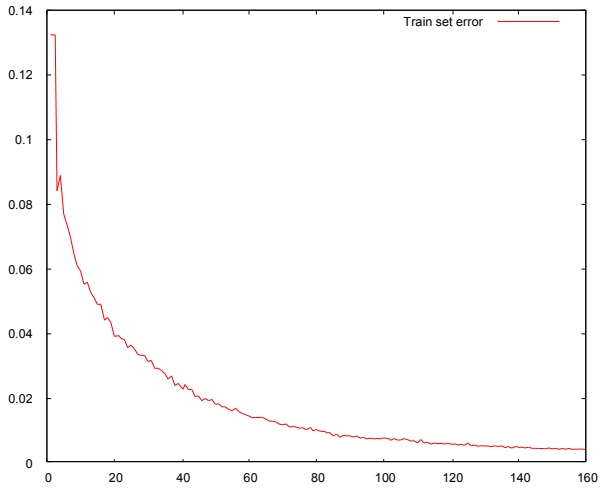
3. Výběr slabého klasifikátoru h_j s nejnižší chybou.
4. Aktualizace vah.
   $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$, kde $e_i = 0$ pokud je vzorek i klasifikován správně a $e_i = 1$, pokud je klasifikován špatně. $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

Hodnota výsledného silného klasifikátoru se vypočítá podle vztahů (10). V každém trénovacím kroku se tedy vybírá příznak $h_t(x)$ s naučenými parametry a jeho váha α_t.

$$\begin{aligned} H(x) &= 1 \quad pro \sum_T \alpha_t h_t(x) > \frac{1}{2} \sum_T \alpha_t \\ H(x) &= 0 \quad jinak \\ \alpha_t &= \frac{1}{\beta_t} \end{aligned} \tag{10}$$

3.2 Chyba trénování

Základní vlastností algoritmu AdaBoost je schopnost exponenciálně snižovat chybu na trénovací sadě. Klasifikátor, který vybírá třídu pro každou instanci náhodně má u binárních problémů chybu 0.5.



Obr. 11: Pokles chyby s počtem slabých klasifikátorů.

Pokud se chyba klasifikátoru h_t vyjádří jako $\epsilon_t = \frac{1}{2} - \gamma_t$, pak γ_t vyjadřuje jakou mírou

je h_t lepší než náhodný výběr. Freund a Schapire [6] dokázali, že chyba silného klasifikátoru *H* je z hora ohraničena (11). Na obrázku 11 je ukázka chování chyby silného klasifikátoru na trénovací sadě vzorků s přibývajícím počtem slabých klasifikátorů.

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2} \tag{11}$$

V případě, že se klasifikátor *H(x)* skládá ze slabých klasifikátorů, které jsou alespoň trochu lepší než náhodná funkce ($\gamma_t > 0$), chyba na trénovací sadě vzorků exponenciálně klesá s počtem slabých klasifikátorů.

3.3 WaldBoost

V předchozí části byl popsán algoritmus AdaBoost. Jeho jistou nevýhodou byl fakt, že předtím, než bylo možné vydat rozhodnutí, do které třídy patří vzorek bylo *nutné* vyhodnotit všechny slabé klasifikátory, kterých může být velké množství. Tato kapitola popisuje algoritmus WaldBoost, který vychází z AdaBoostu. Klasifikátor natrénovaný touto metodou nemusí být vyhodnocen vždy celý. Pokud v průběhu klasifikace vzroste jistota, že vzorek bude klasifikován do určité třídy, vyhodnocení je možné ukončit. Tato výhoda umožňuje použití klasifikátorů v real-time aplikacích, které netolerují zpoždění ve zpracování dat.

WaldBoost je kombinací algoritmu AdaBoost pro výběr a řazení slabých klasifikátorů a metody SPRT (*Sequential Probability Ratio Test* [8, 9]). Algoritmus je založen na sekvenční rozhodovací strategii.

Sekvenční rozhodovací strategie

Při vyhodnocení se v každém kroku se provede jedno měření (vyhodnotí jeden slabý klasifikátor). SPRT definuje dva prahy *A* a *B*. Klasifikace *S* (12) je vyhodnocena jako $+1$ pokud R_t, je vyšší nebo rovno *B*, a pokud je nižší než *A* je klasifikováno -1. V případě, že se R_t nachází mezi prahy *A* a *B*, je potřeba provést přinejmenším ještě jeden krok a vyhodnotit další měření. R_t (13) vyjadřuje likelihood klasifikace do jedné z tříd.

$$\begin{aligned} S &= +1 \quad R_t \leq B \\ S &= -1 \quad R_t \geq A \\ S &= 0 \quad B < R_t < A \end{aligned} \tag{12}$$

$$R_t = \frac{p(x_1, ..., x_t | y=-1)}{p(x_1, ..., x_t | y=+1)} \tag{13}$$

Hodnoty prahů A a B se pro konkrétní aplikaci stanoví podle maximální přípustné míry chyb prvního (vzorek $+1$ a je klasifikován jako -1, *false negative*) a druhého (vzorek -1 a je klasifikován jako $+1$, *false positive*) druhu [8]. Tyto prahy je komplikované přesně vypočítat, ale lze je určit na základě odhadu [9].

Trénování

WaldBoost potřebuje při trénování dva parametry α (*false negative rate*) a β (*false positive rate*), ze kterých se vypočítají prahy $\theta_A^{(t)}$ a $\theta_B^{(t)}$. Trénování probíhá v iteracích. Prvním krokem je standardní AdaBoost výběr slabého klasifikátoru. Následuje odhad R_t a nalezení prahů $\theta_A^{(t)}$, $\theta_B^{(t)}$. Na konci každé smyčky jsou z trénovacích dat odstraněny vzorky u kterých již stávající klasifikátor dokáže rozhodnout, do které třídy patří a jsou přidány nové (*bootstraping*). Následuje pseudokód algoritmu.

Vstup: ohodnocené vzorky $(x_{1,} y_1), ..., (x_n, y_n)$ $x \in X$, $y \in \{-1, 1\}$. Požadované míry chyb α a β.

Inicializace: Váhy vzorků $w_{1,i} = \frac{1}{n}$, pro $i \in (1, n)$,

$A = \frac{1-\beta}{\alpha}$, $B = \frac{\beta}{1-\alpha}$

Pro $t = 1 ... T$

1. Vyběr slabého klasifikátoru s nejmenší chybou (viz algoritmus AdaBoost).
2. Odhad R_t.

$$R_t = \frac{p(H_t(x) | y=-1)}{p(H_t(x) | y=+1)}$$

3. Nalezení prahů $\theta_A^{(t)}$ a $\theta_B^{(t)}$.
4. Odstranění vzorků z trénovací množiny, které odpovídají podmínce $H_t \geq \theta_B^{(t)}$ nebo $H_t \leq \theta_A^{(t)}$.
5. Nahrazení odstraněných vzorků novými.

Výstup: Klasifikátor H_T a prahové hodnoty $\theta_A^{(t)}$ a $\theta_B^{(t)}$.

3.4 Příznaky a slabé klasifikátory

V předchozích podkapitolách bylo popsáno, jak AdaBoost vybírá slabé klasifikátory, ze kterých postupně skládá silnou klasifikační funkci. Slabým klasifikátorem zde může být jakákoliv funkce, která dokáže rozhodovat lépe než náhodná funkce (tedy její chyba je nižší než 0.5). Původní verze AdaBoost počítala s kombinací několika složitých slabých klasifikátorů jako jsou například neuronové sítě. Ve zpracování obrazu jsou velmi často základem slabých klasifikátorů obrazové příznaky (*features*).

Příznaky lze typicky rozdělit do dvou skupin – spojité a diskrétní. Spojité mohou dávat jako výsledek jakoukoliv reálnou hodnotu. Diskrétní příznaky jako výsledek vrací přirozená čísla. Ze spojitých příznaků vytvořit jejich diskrétní verze, jako například u Haarových. Existují ovšem diskrétní příznaky, které nemají svůj spojitý ekvivalent.

Spojité příznaky obyčejně bývají konvoluce na různé pozici v obraze o různé velikosti např. Haarovy nebo Gáborovy [1] vlnky. Typ příznaku je definován tvarem (bází) konvolučního jádra a každá instance příznaku jeho pozicí v obraze a velikostí. AdaBoost nejprve vygeneruje množinu příznaků (na všech pozicích a ve všech velikostech), které použije jako klasifikátory, ze kterých pak vybírá ty, které vykazují nejmenší chybu. Vyčerpávající množina Haarových vlnek (obr. 12) v okně 19×19px obsahuje 45144 příznaků, tedy mnohokrát více než pixelů!

Důvodem používání příznaků místo pixelů je rychlost výpočtu a univerzálnost takového přístupu. Příznakově orientované systému jsou obyčejně rychlejší a flexibilnější než pixelově orientované. Tato podkapitola se zaměřuje na popis slabých klasifikátorů založených na Haarových příznacích. Tyto klasifikátory byly použity například v pracích [7, 8] pro úlohu detekce obličeje.

Haarovy příznaky

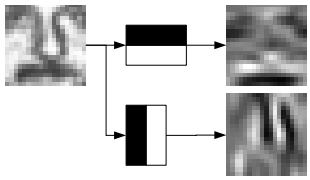
Na obrázku 12 jsou zobrazeny 4 typy Haarových bází. Výpočet hodnoty konkrétního příznaku na obraze je pak realizován podle vzorce (14), kde x je vzorek dat a W a B jsou množiny pixelů příslušející k bílé resp. černé oblasti příznaku.. Jeho interpretace pro konkrétní příznak umístěný v obraze je rozdíl pixelů pod černou a bílou oblastí příznaku. Na obrázku 13 jsou pak ukázány 2 instance stejného typu příznaku v obraze.

Obr. 12: Tvary Haarových příznaků.

$$f(x)=\sum_{w\in W} x(w)-\sum_{b\in B} x(b) \tag{14}$$

Obr. 13: Různé instance jednoho typu příznaku v trénovacím vzorku.

Obrázek 14 ukazuje odezvy dvou typů Haarových příznaků na vzorku. Příznaky byly vyhodnoceny ve všech pozicích, velikost příznaků byla $5\times 2\text{px}$ (nahoře) a $2\times 5\text{px}$ (dole). Každý pixel výsledného obrazu je tedy hodnota patřičného příznaku.



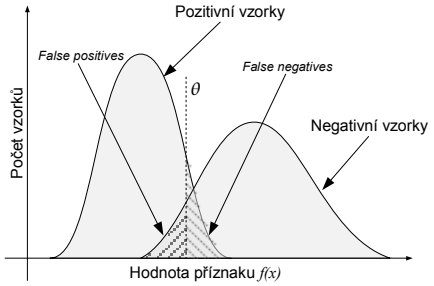
Obr. 14: Odezvy příznaků na vzorku.

Slabý klasifikátor, kromě obrazového příznaku, obsahuje také parametry. V případě Haarových příznaků se jedná o klasifikační práh θ a polaritu p. Práh je rozhodovací hodnota pro klasifikaci do jedné nebo druhé třídy. Polaritou lze prohodit význam klasifikovaných tříd (obrácení znaménka nerovnosti). Takové slabé klasifikátory jsou vyhodnocovány podle vzorce (15), kde $f(x)$ je hodnota příznaku. Výstupní hodnota slabého klasifikátoru může být v závislosti na definici AdaBoost z množiny $\{0,1\}$ nebo $\{-1,+1\}$.

$$\begin{aligned} h_j(x)&=1 \quad pro\ p_j f_j(x) < p_j\theta_j \\ h_j(x)&=0 \end{aligned} \tag{15}$$

Algoritmus AdaBoost obsahuje krok učení klasifikátorů. Jedná se o nalezení optimálních parametrů slabých klasifikátorů. V tomto případě jde o práh a polaritu. Při implementaci se může postupovat například tak, že se vytvoří histogram s výstupem klasifikátoru na vodorovné ose a součtem vah vzorků se stejnou odezvou na svislé (obr. 15) . V průběhu trénování se udržuje statistika (počty *false positives* a *false negatives*) pro obě možnosti nastavení polarity, postupně se nastavuje práh a počítá chyba klasifikátoru. Nakonec se vyberou parametry, se kterými má klasifikátor nejmenší chybu. Jedná se velmi výpočetně a paměťově náročnou operaci, kterou je potřeba provést pro všechny přítomné slabé klasifikátory. Chyba je charakterizována součtem vah chybně klasifikovaných vzorků (16).

$$\epsilon_j = \sum_i w_i |h_{j,\theta,p}(x_i) - y_i| \tag{16}$$



Obr. 15: Histogram pozitivních a negativních vzorků v závislosti na hodnotě příznaku.

Učení slabého klasifikátoru tedy provádí minimalizaci jeho chyby na aktuální distribuci vah vzorků pomocí nastavení jeho vnitřních parametrů.

Jiné typy slabých klasifikátorů

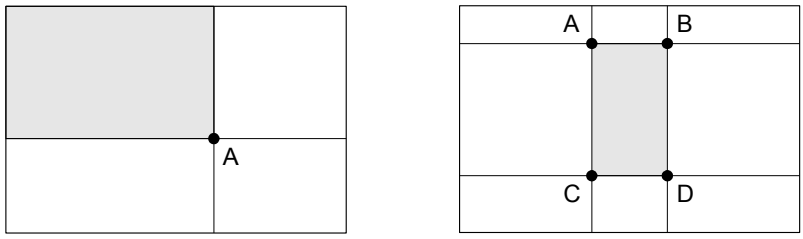
Haarovy příznaky nejsou jediné, které lze použít při trénování AdaBoost. Je možné trénovat s různými typy spojitých příznaků jako například Gáborovy vlnky. Existuje i celá řada diskrétních příznaků – Local Binary Patterns, Local Rank Difference, a další.

Různé typy příznaků se liší náročností svého vyhodnocení. Rychlost vyhodnocení Gáborovy vlnky je oproti Haarově vlnce mnohonásobně vyšší. Některé typu příznaků (zejména diskrétní) je také možné snáze implementovat v hardware (např. v FPGA).

Integrální obraz

Haarovy příznaky jsou založeny na sumách pixelů v obdélníkových oblastech vzorku. Tyto sumy lze implementovat buď jako obyčejné sčítání pixelů, nebo lze použít integrální (sumární) obraz. Přístup s postupným sčítáním pixelů není příliš efektivní, protože s velikostí oblasti vždy roste i doba potřebná ke zjištění sumy. Integrální reprezentace obrazu [7] obsahuje v každém pixelu A hodnotu, která představuje součet všech pixelů vlevo a nahoru od A (6). Pixel v pravém dolním rohu reprezentuje tedy součet všech pixelů v obraze. Díky tomuto uložení obrazu lze získat součet jakékoliv obdélníkové oblasti obrazu bez ohledu na její velikost v *konstantním* čase pouze s použitím reference rohových bodů této oblasti. Je také samozřejmě možné získat zpět původní hodnoty pixelů. Rovnice (17) představuje matematickou reprezentaci integrálního obrazu.

$$ii(x,y) = \sum_{x' \le x,\, y' \le y} i(x', y') \tag{17}$$



Obr. 16: Integrální obraz.

Pokud jsou tedy rohové pixely A, B, C a D (na obrázku 16 vpravo), lze sumu pixelů v šedé oblasti vyjádřit jako $A+D-B-C$.

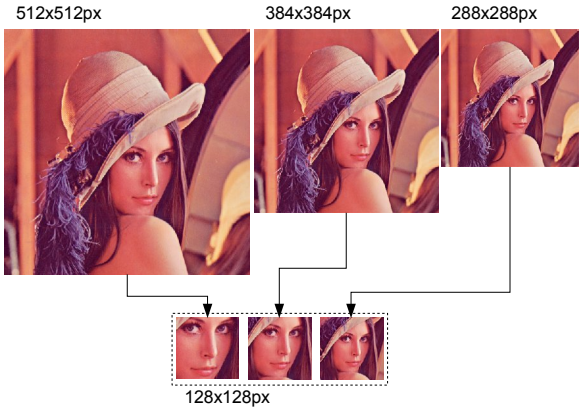
4 Knihovna pro detekci objektů

Výsledkem trénování AdaBoost je vždy klasifikátor, který s určitou chybou dokáže zařadit objekt v obraze do své třídy. Schopnost klasifikace lze využít pro detekci objektu. Pro experimentování s natrénovanými klasifikátory byla vytvořena knihovna, která provádí detekce v obraze. Následující podkapitoly popisují možnosti realizování detekcí klasifikátorem, vnitřní implementaci detektoru a API knihovny, které lze použít pro detekci objektů v dalších aplikacích.

Podkapitola 4.3 pak popisuje ovládání programu, který používá API pro detekci objektů ve videosekvencích. Program dokáže načíst několik klasifikátorů z XML souborů a zobrazovat jejich detekce ve videu. Umožňuje povolit nebo zakázat použití každého klasifikátoru a nastavit jejich parametry. Lze také měnit parametry vstupního obrazu – počet úrovní, do kterých bude každý obraz převzorkován a poměr velikosti jednotlivých úrovní. Pro reprezentaci obrazu byla použita knihovna DigILib a pro vstup videa knihovna AVFile. Obě knihovny jsou vyvíjeny na FIT VUT Brno v rámci skupiny Graph@FIT.

4.1 Implementace detektoru

Detektor je vlastně klasifikátor, který se v obraze posunuje a na každé své pozici provede klasifikaci. Velikost okna klasifikátoru je konstantní, ale velikost objektů, které budou detekovány může být různá. Proto je potřeba měnit velikost klasifikačního okna nebo měnit velikost obrazu, ve kterém budou detekce prováděny.



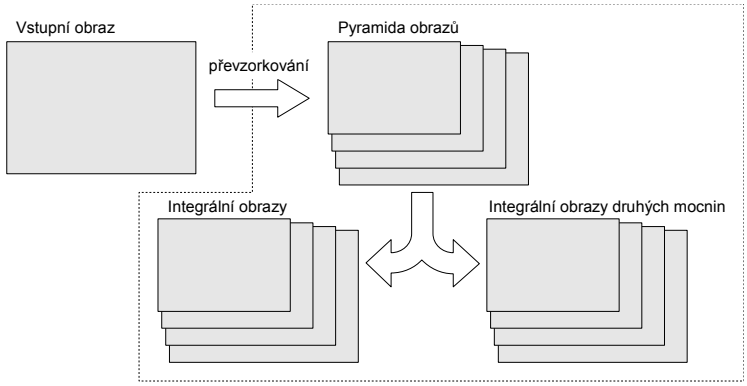
Obr. 17: Subokna stejné velikosti braná z obrazů s různým rozlišením.

V následujících podkapitolách je popsána realizace přípravy obrazu pro detektor, imple-

mentace příznaků a slabých klasifikátorů a vlastní realizace detektoru.

Příprava obrazu pro detektor

Zde je použit přístup, který nejprve vytvoří několik převzorkovaných verzí v postupně se snižujících rozlišeních z originálního obrazu (pyramida obrazů, obrázek 18) a ke každému následně vytvoří integrální obraz. Tento přístup je univerzální a dovoluje v klasifikátorech použít i příznaky u kterých nelze měnit jejich velikost.



Obr. 18: Fyzická realizace pyramidy obrazů.

Velikost obrazu v každé úrovní pyramidy lze vypočítat z velikosti vstupního snímku x_0 a z hodnoty faktoru převzorkování f (18).

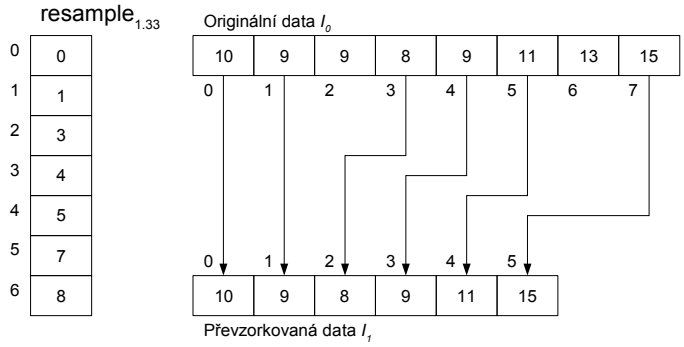
$$x_k = \frac{x_0}{f^k} \quad k \in \langle 0; N \rangle \tag{18}$$

Pro každou úroveň pyramidy je následně vytvořena vyhledávací tabulka, která urychluje převzorkování (19), k je úroveň pyramidy, pro kterou se tabulka počítá, x je pozice v tabulce. Převzorkování se pak provede pomocí vztahu (20), kde I_0 je vstupní obraz.

$$\text{resample}_k(x) = \text{round}(x\, f^k) \tag{19}$$

$$I_k(x, y) = I_0(\text{resample}_k(x), \text{resample}_k(y)) \tag{20}$$

Na obrázku 19 je ukázána fyzická realizace převzorkování jednorozměrných dat. Vlevo je tabulka, jejíž hodnoty se použijí jako indexy do pole originálních dat. V případě obrazů se tabulka použije pro indexování sloupců a řádků vstupního obrazu. Takto lze obraz převzorkovat do několika rozlišení aniž by byl za potřebí velký výpočetní výkon.



Obr. 19: Ukázka převzorkování jednorozměrných dat s faktorem 1,33.

Ke každé úrovni v pyramidě je vytvořen odpovídající integrální obraz a integrální obraz z druhých mocnin originálních hodnot. Každý jeho pixel odpovídá součtu všech hodnot vlevo a nahoru od něj. Výpočet integrálního obrazu lze implementovat jedním průchodem obrazem. Následující vztahy vyjadřují výpočet jednoho řádku y integrálního obrazu. I je vstupní obraz, J výstupní obraz a *tmp* pomocná proměnná pro součet hodnot na řádku.

$$\begin{aligned} tmp(x) &= \sum_{i=0}^{x} I(i, y) \\ J(x, y) &= tmp(x) + J(x, y-1) \end{aligned} \tag{21}$$

Haarovy příznaky potřebují po svém vyhodnocení normalizovat standardní odchylkou hodnot ve zkoumaném okně. Výpočet standardní odchylky pro každé zkoumané subokno by byl časově velmi náročný, proto se ke každému obrazu vytváří integrální obraz druhých mocnin hodnot, který spolu s obyčejným integrálním obrazem dovoluje provést výpočet standardní odchylky v libovolném subokně obrazu v konstantním čase. Podle vztahu 22 je potřeba pro výpočet standardní odchylky znát průměr zkoumaných hodnot a průměr jejich druhých mocnin.

$$\sigma^2 = \frac{1}{N}\left(\sum_{N} x^2\right) - \bar{x}^2 \tag{22}$$

Vzorec 23 ukazuje výpočet, kde A, B, C a D jsou hodnoty v rozích okna v obyčejném integrálním obraze a E, F, G a H jsou hodnoty v rozích okna v integrálním obraze druhých mocnin. N je počet pixelů v okně.

$$\sigma^2 = \tfrac{1}{N}(E - F - G + H) - \left[\tfrac{1}{N}(A - B - C + D)\right]^2 \tag{23}$$

Každá úroveň pyramidy je následně skenována a u každého zkoumaného subokna je provedena klasifikace.

Příznaky a klasifikátory

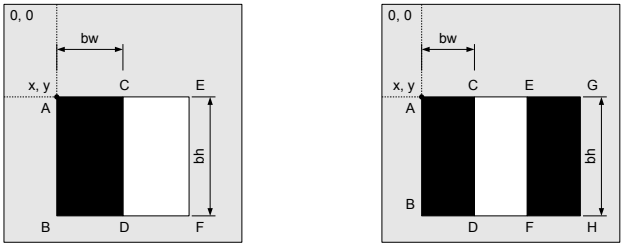
V současné době jsou podporovány pouze spojité Haarovy příznaky, které lze implementovat v software velmi efektivně. V kapitole 3.4 byly matematicky popsány jejich parametry. Zde bude uvedeno, jak je v knihovně řešeno jejich vyhodnocení a vyhodnocení slabých klasifikátorů.

Všechny struktury jsou implementovány objektovým přístupem. Každý typ příznaku musí být potomkem třídy `TFeature`, která definuje rozhraní příznaků. Různé typy slabých klasifikátorů jsou potomkem třídy `TWeakClassifier`. V případě slabých klasifikátorů založených na Haarových příznacích pak struktura obsahuje ukazatel na příznak a další parametry. Třída silného klasifikátoru `TClassifier` pak obsahuje vektor slabých klasifikátorů.

Třída `TFeature`, definuje rozhraní které zajišťuje načtení příznaku z XML struktury a vyhodnocení příznaku nad obrazem. Potomky této třídy jsou třídy, které definují vyhodnocení různých typů Haarových příznaků.

```
THaarDoubleHorizontal
THaarDoubleVertical
THaarTripleHorizontal
THaarTripleVertical
```

```
class TFeature {
public:
      virtual ~TFeature() { }
      virtual float eval(const TSampleImage&) = 0;
      virtual int loadFromXML(xmlNodePtr) = 0;
};
```



Obr. 20: Horizontální příznaky umístěné ve vzorku a jejich parametry.

Konkrétní implementace Haarových příznaků pak obsahují parametry, které určují, na kterém místě vzorku se mají vyhodnotit – pozice a velikost.

Na obrázku 20 jsou zobrazeny dva typy horizontálních příznaků a jejich parametry. Hodnota *x*, *y* udává pozici levého horního rohu ve vzorku. Hodnoty *bw* a *bh* udávají

šířku a výšku jednoho bloku příznaku. Tyto údaje se použijí pro výpočet umístění všech rohových bodů (na obrázku jsou označeny jako A až *H*), ze kterých se vypočítá odezva příznaku. Vyhodnocení příznaků se provádí nad integrálním obrazem. Pokud jsou vyhodnocované bloky spojené, není třeba počítat sumy bloků samostatně, ale je možné celý vztah zjednodušit. Vzorce 24 a 25 ukazují výpočet odezvy `THaarDoubleHorizontal` a `THaarTripleHorizontal` příznaků. Analogicky je možné optimalizovat výpočet vertikálních verzí příznaků.

$$f(X) = -A + B + 2(C - D) + E - F \tag{24}$$

$$f(X) = A - B - 3(C + D + E - F) - G + H \tag{25}$$

Podobně jako `TFeature` je pro slabé klasifikátory definováno rozhraní `TWeakClassifier`. Slabý klasifikátor může obsahovat kromě příznaku další parametry, podle kterých rozhoduje o zařazení vzorků do svých tříd. Abstraktní třída definující rozhraní vypadá následovně.

```
class TWeakClassifier {
      TFeature * feature;
public:
      TWeakClassifier(TFeature * f);
      virtual ~TWeakClassifier();
      virtual float eval(const TSampleImage&) const = 0;
      virtual int loadFromXML(xmlNodePtr) = 0;
};
```

Potomkem této třídy jsou pak implementace různých typů slabých klasifikátorů obsahující parametry potřebné pro jejich funkci. Je možné například vytvořit spojité a diskrétní verze Haarových příznaků nebo různé typy příznaků založených na lokálních změnách v obraze. Potomkem této třídy je zde pouze třída `THaarWeakClassifier`, která definuje vyhodnocení slabých klasifikátorů založených na spojitých Haarových příznacích. Obsahuje parametry *alpha* (váha klasifikátoru), *parity* (parita) a *threshold* (klasifikační práh).

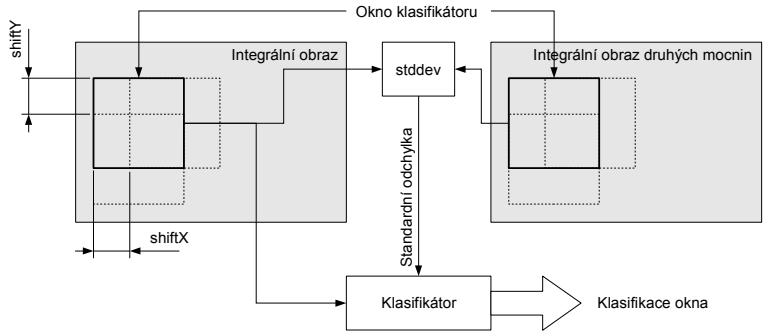
Poslední součástí je silný klasifikátor. Ten obsahuje vektor se slabými klasifikátory a parametry, které řídí proces detekce (velikost skenovacího okna, posuv v obraze, detekční práh). Slabé klasifikátory ve vektoru jsou jsou seřazené podle pořadí v jakém byly vybrány během trénování (tedy podle jejich důležitosti při vyhodnocení). Třída umožňuje načíst celý klasifikátor z XML struktury a provést scanování obrazu tímto klasifikátorem.

```
class TClassifier {
```

```
      std::vector<TWeakClassifier*> wClassifier;
      float T, msd;
      int sizeX, sizeY, sx, sy;
public:
      unsigned scanImage(ImageStruct*, ImageStruct*, TRe-
sultVector&, unsigned);
      void addWeakClassifier(TWeakClassifier*);
      int loadFromXML(xmlNodePtr);
      // ...
};
```

Uvedené třídy a jejich potomci zajišťují provedení detekce v obraze. Samotné scanování obrazu klasifikátorem zařizuje metoda `scanImage`. Při klasifikaci vzorku je v metodě `TClassifier::scanImage()` volána metoda `TWeakClassifier::eval()` každého slabého klasifikátoru, ze které se volá příslušná metoda `TFeature::eval()` konkrétního typu příznaku, který slabý klasifikátor obsahuje. Výsledky slabých klasifikátorů se akumulují a pokud hodnota přesáhne klasifikační práh `T` je klasifikace pozitivní.

Výsledek je pak předán ve struktuře `TResultVector`, která obsahuje předalokované pole pro výsledky. Struktura `TClassifier` umožňuje také nastavit parametry skenování obrazu. Je možné nastavit horizontální a vertikální posun skenovacího okna (`sx`, `sy`) a minimální standardní odchylku (`msd`), jakou obraz v okně musí mít aby byla provedena klasifikace. Ve většině případů není potřeba se zabývat plochými místy v obraze jako například obloha nebo stěny, které maji standardní odchylku nízkou. Tato místa jsou automaticky vynechána, což se projevuje na celkové rychlosti zpracování obrazu.



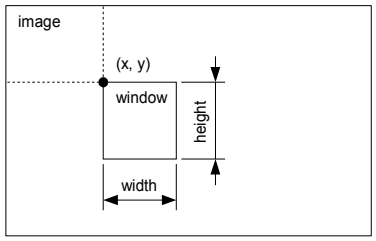
Obr. 21: Princip skenování obrazu klasifikátorem.

Činnost metody `TClassifier::scanImage()` vyjadřuje obrázek 21. Vstupem jsou dva integrální obrazy. Jeden je použit pro vyhodnocení klasifikátorů a druhý k výpočtu standardní odchylky hodnot v klasifikovaném okně.

Implementace skenovacího okna je velmi efektivní. Místo kopírování dat z obrazu do ji-

ného obrazu, který bude ohodnocen klasifikátorem je použita funkce DigILibu, která dovoluje vytvořit plnohodnotný obraz jako referenci do jiného obrazu. Takto vytvořený obraz se pak chová jako běžná struktura `ImageStruct`. A je možné v ní provádět reference pixelů relativně k pozici okna v původním obraze.

```
ImageStruct * image; // původní obraz
ImageStruct   window; // obdélníková oblast
SetTempReference(&window, image, x, y, width, height);
```



Obr. 22: Obdélníková oblast v obraze

Během skenování je okno umisťováno na všechny souřadnice v obraze tak aby se pokryla celá jeho plocha a každé takové okno je klasifikováno (s výjimkou těch, která mají standardní odchylku menší než je definováno). Pokud je výsledek klasifikace pozitivní a je tedy nalezen hledaný objekt, je souřadnice okna a jeho velikost zapsána do pole s výsledky.

4.2 Rozhraní knihovny

Nad výše popisovanými strukturami a algoritmy je postaveno vysokoúrovňové API, které poskytuje jednoduše použitelné funkce pro detekování objektů. Jsou zde funkce pro načítání klasifikátorů ze souboru, nastavení parametrů klasifikátorů a skenování obrazu. Knihovna podporuje i použití více klasifikátorů. Klasifikátory lze načítat ze souborů XML. Každý klasifikátor je pak pro ostatní funkce identifikován ukazatelem typu `hClassifier`.

V současné době je k dispozici dynamická verze pro překladače GCC a C++ Builder. Knihovna byla vyzkoušena v prostředí OS Windows a Linux. Zde je popis exportovaných funkcí pro práci s klasifikátory.

`hClassifier LoadClassifier(hClassifier filename);`

Načte klasifikátor z XML souboru. Pokud se objeví chyba, vrací nulový ukazatel, v opačném případě handle nového klasifikátoru. Handle se používá jako identifikace každého klasifikátoru v ostatních funkcích.

```
void ReleaseClassifier(hClassifier handle);
```

Zruší klasifikátor definovaný hodnotou `handle` a uvolní jeho paměť. V případě, že žádný takový neexistuje, nic se nestane.

```
int ScanImage(hClassifier handle, ImageStruct * i, int length,
    TResultVector & results);
```

Skenuje obraz klasifikátorem definovaným pomocí handle. `length` je maximální počet slabých klasifikátorů, které se při klasifikaci vyhodnotí. Tato funkce nepoužívá možnosti pyramidy obrazů a obraz skenuje pouze v jednom rozlišení (s jednou velikostí klasifikátoru) a vnitřně si alokuje pomocné integrální obrazy.

```
void GetClassifierSize(hClassifier handle, int & sx, int & sy);
```

Uloží velikost skenovaného okna v pixelech do `sx` a `sy`.

```
void GetClassifierShift(hClassifier handle, int & sx, int & sy);
```

Uloží horizontální a vertikální posun skenovacího okna v pixelech do `sx` a `sy`.

```
unsigned GetClassifierLength(hClassifier handle);
```

Vrátí počet slabých klasifikátorů, které klasifikátor obsahuje.

```
float GetClassifierEvalStddev(hClassifier handle);
```

Vrátí minimální hodnotu standardní odchylky, kterou zkoumané okno musí mít aby byl klasifikátor vyhodnocen.

```
float GetClassifierThreshold(hClassifier handle);
```

Vrátí detekční práh klasifikátoru.

```
void SetClassifierShift(hClassifier handle, int sx, int sy);
```

Nastaví horizontální a vertikální posun skenovacího okna. Pokud je hodnota 0, odpovídající posun zůstane nezměněn. `SetClassifierShift(handle, 0, 8)` tedy nastaví pouze vertikální posuv na 8px a horizontální zůstane nezměněn.

```
void SetClassifierEvalStddev(hClassifier handle, float stddev);
```

Nastaví minimální hodnotu standardní odchylky, kterou musí mít zkoumané okno aby byl klasifikátor vyhodnocen.

```
void SetClassifierThreshold(const int handle, float t);
```

Nastaví detekční práh klasifikátoru na hodnotu `t`.

```
hPyramid CreatePyramid(float f, unsigned depth, unsigned x, un-
    signed y);
```

Vytvoří strukturu obrazové pyramidy a vrátí její ukazatel. `f` je poměr velikostí dvou po sobě následujících obrazů pyramidy, `depth` je počet vytvářených úrovní. `x` a `y` jsou výška a šířka vstupního obrazu.

`void ReleasePyramid(hPyramid handle);`
Zruší strukturu pyramidy a uvolní ji z paměti.

`void InsertImage(hPyramid handle, ImageStruct * image);`
Vloží nový obraz do pyramidy a provede jeho převzorkování na všechny úrovně a vytvoření integrálních obrazů a integrálních obrazů druhých mocnin.

`unsigned GetPyramidDepth(hPyramid handle);`
Vrátí počet úrovní pyramidy.

`float GetPyramidScaleFactor(hPyramid handle);`
Vrátí poměr velikostí dvou sousedních úrovní pyramidy.

`void SetPyramidDepth(hPyramid handle, unsigned depth);`
Nastaví počet úrovní pyramidy. Její obsah je pak nedefinovaný.

`void SetPyramidScaleFactor(hPyramid handle, float factor);`
Nastaví poměr velikosti sousedních úrovní pyramidy.

`int ScanImagePyramid(hPyramid pHandle, hPyramid cHandle, TResultVector & results);`
Provede skenování všech úrovní pyramidy klasifikátorem `cHandle`. Výsledky jsou uloženy do pole `results`. Funkce vrací počet detekcí. Pokud identifikace `pHandle` nebo `cHandle` není platná nic se neprovede. Činnost funkce se skládá z vložení vstupního obrazu do pyramidy a následném skenování všech úrovní klasifikátorem.

Typické použití API se skládá z vytvoření struktury pyramidy obrazů, načtení klasifikátoru z XML a alokace pole pro výsledky. Každý vstupní snímek se vloží do pyramidy funkcí `InsertImage()` a funkce `ScanImagePyramid()`, pak zpracuje obraz a uloží výsledky do pole. Detekce lze provádět i bez vytvoření pyramidy pomocí funkce `ScanImage()`.

```
#include <detector.h>
...
hClassifier c = LoadClassifier("classifier.xml");
hPyramid p = CreatePyramid(1.33, 8, 720, 576);
TResultVector results(20000);
...
// Získání nového obrazu z videa, kamery, atd...
InsertImage(p, image);
int d = ScanImagePyramid(p, c, results);
// Zápis/vykreslení detekcí
...
```

```
ReleasePyramid(p);
ReleaseClassifier(c);
```

Po volání funkce `ScanImage()` nebo `ScanImagePyramid()` jsou detekce uložené ve struktuře `TResultVector`. Ke každé detekci je možné přistoupit indexováním této struktury a lze získat pozici a velikost detekce pomocí přístupu k vnitřním položkám struktury. Pozice je dvourozměrný vektor `pos`, velikost `size`.

```
TResultVector results(1000);
int x = results[0].pos[0];
int y = results[0].pos[1];
int width = results[0].size[0];
int height = results[0].size[1];
```

Překlad programů

Pro bezproblémový překlad programů využívajících dynamickou knihovnu (`detector.dll` pro Windows a `detector.so` pro Linux) je potřeba mít přeloženou knihovnu umístěnou v adresáři, na který ukazuje systémová proměnná `PATH`. Při překladu je pak potřeba k programu přilinkovat rozhraní knihovny `libdetector.a`. Pokud není umístěno v adresáři s knihovnami kompilátoru, je potřeba cestu při překladu zadat. Takto může vypadat překlad programu `test.cpp` překladačem GCC, když je hlavičkový soubor i rozhraní knihovny přímo v adresáři se zdrojovými kódy.

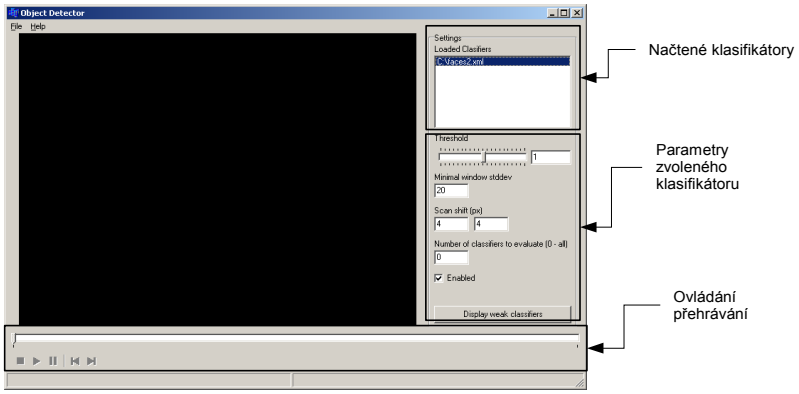
```
$g++ test.cpp -o test -ldetector -L. -I.
```

V prostředí *Borland C++ Builder* je nutné do projektu přidat rozhraní knihovny `detector.lib`.

4.3 Program pro detekci objektů

Implementovaný program používá výše popsané API k detekci objektů. Pro vstup videa byla použita knihovna AVFile s podporou DigILib obrazů. GUI bylo realizováno v prostředí Borland C++ Builder. Tato podkapitola popisuje ovládání programu z uživatelského pohledu.

Okno programu obsahuje prostor pro zobrazení videa. V pravé části nahoře je seznam načtených klasifikátorů, pod ním jsou parametry, klasifikátoru a ve spodní části je ovládání přehrávače.



Obr. 23: Hlavní okno programu.

Činnost programu

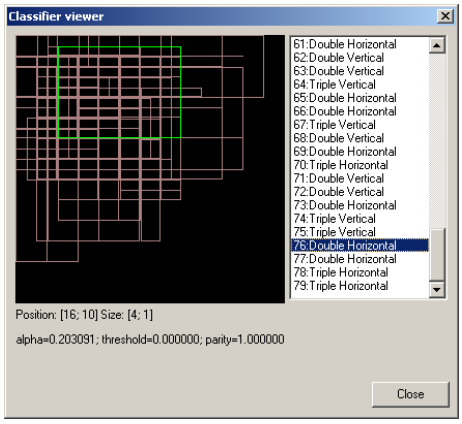
Program udržuje seznam načtených klasifikátorů a jejich vlastností. Pro každé video vždy vytvoří novou obrazovou pyramidu pro patřičnou velikost vstupního obrazu a alokuje nové pomocné obrazy.

Při přehrávání program vždy načte jeden snímek videa a převede jej do stupňů šedi. Tento obraz je následně vložen do pyramidy. Pro každý klasifikátor je volána funkce `ScanImagePyramid()` a je provedeno skenování všech úrovní pyramidy. V poli výsledků jsou pak vráceny detekce, které se vyznačí do původního obrazu. Nakonec se zobrazí obraz s označenými detekcemi.

Nastavení klasifikátorů a pyramidy

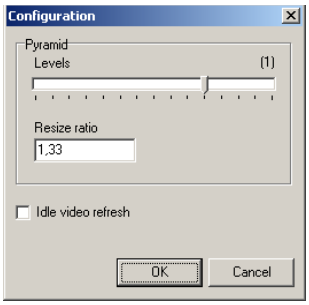
Program umožňuje použít několik různých klasifikátorů pro detekci různých objektů. Každý klasifikátor podporuje několik nastavení, která je možné měnit v pravé části okna. Mění se vždy parametry klasifikátoru, který je označen v seznamu vpravo nahoře. Všechny změny se ihned po zadání projeví v činnosti klasifikátoru. Je možné měnit tyto vlastnosti.

1. Povolení/Zakázání detekcí
2. Nastavení detekčního prahu
3. Nastavení minimální standardní odchylky okna pro vyhodnocen klasifikátoru
4. Nastavení posunu skenovacího okna v obraze
5. Počet vyhodnocovaných slabých klasifikátorů



Obr. 24: Obsah silného klasifikátoru.

Tlačítko *Display weak classifiers* zobrazí okno s informacemi o slabých klasifikátorech obsažených v označeném klasifikátoru (obrázek 24). Zobrazí jejich pozice a velikosti ve skenovacím okně. V seznamu vpravo se vypisuje typ příznaku. Označením slabého klasifikátoru v seznamu se provede jeho zvýraznění v náhledu vlevo a ve spodní části okna se zobrazí detailní informace – práh, alpha a parita. Tyto hodnoty jsou jen informační a nelze je nijak měnit.



Obr. 25: Dialog pro konfiguraci pyramidy.

Na obrázku 25 je dialog, který umožňuje měnit parametry pyramidy obrazů. Lze měnit počet úrovní a poměr velikstí dvou sousedních úrovní. Změny se projeví okamžitě po potvrzení dialogu.

Rychlost detekcí

Na každém vstupním snímku je potřeba vyhodnotit klasifikátor ve všech pozicích. Počet těchto vyhodnocení přímo ovlivňuje rychlost detektoru. Faktory, které nejvíce ovlivňují počet nutných klasifikací jsou: Rychlost procesoru, na kterém detekce běží, velikost zpracovávaného obrazu, počet úrovní, ve kterých se obraz skenuje a velikost klasifikátoru. Paměťové nároky detektoru nejsou příliš veliké. Pokud je dostatek paměťi na uložení převzorkovaných a integrálních obrazů, není rychlost výpočtu nijak ovlivněna. Většinou

není potřeba více než několik desítek MB paměti.

Například máme-li obraz o velikosti $640 \times 480\text{px}$, klasifikátor $24 \times 24\text{px}$ a skenujeme jej v krocích po 2px (horizontálně i vertikálně) bude potřeba 70 224 vyhodnocení silného klasifikátoru. Pokud tento klasifikátor obsahuje 100 slabých klasifikátorů bude potřeba 7 022 400 vyhodnocení příznaků! Počet vyhodnocení silného klasifikátoru na jednom obraze lze zobecnit na tvar (26), kde *W* a *H* jsou rozměru obrazu, *cw* a *ch*, rozměry klasifikátoru a *sx* a *sy* posun klasifikátoru v obraze v horizontálním a vertikálním směru.

$$n = \text{round}\left(\frac{W - cw}{sx}\right)\text{round}\left(\frac{H - ch}{sy}\right) \tag{26}$$

Ze vztahu vyplývá, že s velikostí okna klasifikátoru a velikostí skenovacího kroku klesá počet vyhodnocení. Rostoucí velikost obrazu naopak počet vyhodnocení zvyšuje.

Dalším faktorem ovlivňujícím rychlost detektoru je počet úrovní pyramidy obrazů a faktor pro převzorkování jednotlivých úrovní. S počtem úrovní stoupá i počet vyhodnocení klasifikátoru. Dále, v pyramidě s hodnotou faktoru blízkou 1 se tvoří obrazy o velikosti blízké původnímu obrazu a tedy počet vyhodnocení stoupá rychleji než kdyby byla hodnota faktoru vyšší a v pyramidě vy se tvořily menší obrazy.

Rychlost zpracování obrazu je výrazně ovlivněna počtem slabých klasifikátorů, ze kterých se detektor skládá. V klasifikátoru s nízkou chybou (v řádu desetin procent) mohou být až stovky klasifikátorů, které je nutné vyhodnotit všechny (díky statickému klasifikačnímu prahu). Když uvážíme klasifikátor trénovaný metodou WaldBoost je průměrný počet vyhodnocených klasifikátorů v řádu jednotek až desítek, což by znamenalo výrazné urychlení zpracování obrazu.

5 Nástroj pro trénování klasifikátorů

Program popsaný v předchozí části slouží k detekci objektů pomocí natrénovaného klasifikátoru. Pro trénování klasifikátorů byla použita implementace algoritmu AdaBoost vytvořená na UPGM VUT Brno v rámci projektu *Hardwarová akcelerace AdaBoost*. Následující kapitoly popisují možnosti konfigurace této implementace trénování. Nakonec je uveden formát výsledného klasifikátoru, který je možné využít pro detekci objektů.

Konfigurace trénování se programu předává v XML souboru, ve kterém je určeno na jakých datech se bude klasifikátor trénovat, jaké příznaky se mají při trénování použít a jaké mají být vlastnosti klasifikátoru. Jiný XML soubor popisuje data a rozděluje je do pojmenovaných skupin.

Podporované typy klasifikátorů

Tato implementace AdaBoost podporuje v současné době dva druhy slabých klasifikátorů – spojité a diskrétní. Nástroj podporuje následující typy příznaků.

1. Spojité *Haarovy* příznaky
2. Diskrétní *Haarovy* příznaky
3. Diskrétní *Local Rank Difference* (LRD) příznaky několika typů
4. Diskrétní *Parity Change* příznaky
5. Příznaky reprezentující jednotlivé pixely
6. a další

Všechny tyto typy lze ve výsledném klasifikátoru libovolně kombinovat.

5.1 Dataset a konfigurace

Data pro trénování jsou specifikována v XML souboru, ve kterém jsou jména obrazových souborů se vzorky sdruženy do pojmenovaných skupin (*subsets*). Jedna skupina může obsahovat mnoho souborů různých typů. Jména skupin jsou později potřeba v konfiguračním XML, kde se musí určit která data budou načtena jako příklady a protipříklady pro trénování. Následuje ukázka definice datasetu, ve které se definují dvě množiny a jim příslušející odkazy na soubory s daty.

```
<dataset name="Test" directory="/mnt/data/test/">
      <params>
```

```
            <sampleInfo sizeX="24" sizeY="24">
      </params>
      <subsets>
            <subset name="subset1">
                  <raw name="data1.raw"/>
                  <raw name="data2.raw"/>
            </subset>
            <subset name="subset2">
                  <raw name="data3.raw"/>
                  ...
            </subset>
      </subsets>
</dataset>
```

Data jsou zde umístěna v adresáři `/mnt/data/test` a velikost vzorků z tohoto datasetu bude 24×24px . Dataset obsahuje dvě množiny `subset1` a `subset2`. Odkazy na data se definují jako subelementy v části `subset`. Zde jsou vyjmenovány možnosti definice různých typů dat.

`raw` – Vstup dat z `raw` souboru, kde jsou uložena čistá obrazová data.

`name` – jméno souboru s daty.

Pokud k souboru existuje anotace dat, lze volitelně specifikovat soubor s anotací a množinu dat, která se mají použít.

`annotation` – Jméno souboru s anotací.

`set` (`int`) – Identifikace množiny, která se má nahrát.

`samples` – Vstup z jednotlivých obrázkových souborů.

`subimages` – Subobrazy větvětšího obrazu jsou načteny jako vzorky.

`name` – jméno obrazového souboru.

`step` – krok s jakým jsou vybírány jednotlivé subobrazy (1 znamená celá šířka vzorku v horizontálním směru (výška ve vertikálním)).

Podporované vstupní obrazové formáty jsou v současné době `pgm` a `raw`. Každý soubor `pgm` musí obsahovat právě jeden vzorek o určené velikosti. Soubory `raw` obsahují mnoho vzorků stejné velikosti. `Raw` soubory jsou tedy vhodné pro uložení tisíců vzorků na místo aby existovaly tisíce malých souborů.

Dalším souborem vstupujícím do trénování je konfigurace. Zde jsou určena data a obrazové příznaky, se kterými se bude trénovat.

```
</trainingAdaBoost
            description="name"
```

```
            randSeed="000000000">
      <dataset path="\mnt\data\test.xml">
            <!-- definice trénovacích
            a testovacích vzorků -->
      </dataset>
      <classifier
            <!-- parametry trénování -->
      />
      <features
            <!-- parametry příznaků -->
      >
            <!-- definice použitých příznaků -->
      </features>
</trainingAdaBoost>
```

Parametr kořenového elementu `randSeed` vyjadřuje inicializační hodnotu generátoru náhodných čísel. Tento parametr neni povinný a pokud neni zadán je generátor inicializován náhodně při startu programu. Jeho hodnota se zapíše do výsledného XML aby bylo možné pokračovat v trénování se stejnými podmínkami s jakými bylo spuštěno.

Element `dataset` obsahuje definici dat vstupujících do trénování. Parametr `name` obsahuje jméno XML souboru s datasetem. Uvnitř tohoto elementu je specifikováno, která data budou použita pro trénování a která pro testování.

```
<dataset name="...">
      <samples
            purpose="training" name="subset"
            amount="1000" classValue="1.0"/>
      <samples ... />
</dataset>
```

Element `samples` definuje ze které množiny v použitém datasetu se budou vzorky nahrávat, k jakému účelu budou vzorky sloužit (`training`, `testing`), kolik jich bude nahráno a do jaké třídy budou patřit. Hodnota třídy se pro dvoutřídní problému volí typicky 1.0 (příklady) a -1.0 (pro protipříklady).

V elementu `classifier` jsou definovány parametry výsledného klasifikátoru a jména výstupních souborů.

```
<classifier
      length="150" output="classifier.xml"
      featureOutput="features.txt"
      trainFeatureVectorsOutput="train_features.txt"
```

```
      testFeatureVectorsOutput="test_features.txt"
      weightNormalize="classes"
/>
```

`length` – `int`

Určuje počet slabých klasifikátorů, které budou natrénovány.

`Output` – `string`

Výstupní soubor s natrénovanými slabými klasifikátory.

`trainFeatureVectorsOutput`, `testFeatureVectorsOutput` – `string`

Jména souborů s odezvami vybraných příznaků na jednotlivé trénovací a testovací vzorky – vektory příznaků. Tyto vektory lze následně použít pro trénování klasifikátorů jiného typu – například neuronových sítí nebo support vector machines. Tento přístup byl použit ve frameworku pro srovnání klasifikátorů [10].

`weightNormalize` – `classes`, `entire`

Určuje jakým způsobem budou váhy načtených vzorků normalizovány. Hodnota `classes` udává, že normalizace jednotlivých tříd proběhne nezávisle na sobě, hodnota `entire`, že všechny váhy budou normalizovány společně.

Parametry elementu `features` definuje vlastnosti použitých příznaků, subelementy pak definují konkrétní příznaky. Některé parametry se vztahují jen ke spojitým příznakům (CF – *continuous feature*), některé k diskrétním (DF – *discrete feature*), a některé k oběma typům. Zde jsou vyjmenovány některé parametry, které ovlivňují vlastnosti použitých příznaků.

`useCDTree` (CF) – `true`/`false`

Použití rozhodovacího stromu u spojitých příznaků.

`useThresholds` (CF) – `true`/`false`

Pokud je hodnota `true`, jsou použity příznaky s prahem a paritou. Při `false` je použita pouze parita a práh je vždy 0. Tento parametr má smysl jen pokud `useCDTree="false"`.

`thresholdEstimationPrecission` (CF) – `int`

Přesnost nastavení prahu slabého klasifikátoru a rozhodovacího stromu.

`maxPrestoredResultCount` (DF) – `int`

Velikost paměti (v bytech) alokované jako cache pro výsledky diskrétních příznaků. Pokud je alokována dostatečně velká paměť trénovací proces se může výrazně urych-

lit.

`restrictAlpha` (DF) – `true`/`false`
Povolení omezení hodnoty alpha u diskrétních slabých klasifikátorů. Maximální hodnotu udává parametr `maxAlpha`.

`maxAlpha` (DF) – `float`
Maximální hodnota alpha. Parametr má vliv pouze při `restrictAlpha="true"`.

`mergeBinsMode` (DF) – `blank`, `equalize`, `subsample`, `Dtree`
Metoda použitá pro snížení počtu *binů*.

`equalize` – vyváží počet vzorků v jednotlivých binech.

`subsample` – Podvzorkuje biny faktorem `binSubsampleRatio`.

`DTree` – Použije rozhodovací strom.

`treeDepth` (CF, DF) – `int`
Maximální hloubka rozhodovacího stromu.

`alphaQuantization` (DF) – `true`/`false`
Udává, zda se budou hodnoty alpha diskrétních příznaků kvantizovat.

`alphaQuantizationResolution` (DF) – `int`
Udává počet bitů, na které se budou alphy kvantizovat při `alphaQuantization="true"`

`maxAlphaQuantizationValue` (DF) – `float`
Maximální hodnota alpha při zapnuté kvantizaci.

Subelementy elementu features obsahují definice různých typů příznaků vstupujících do trénování. Každý typ příznaku je definován svým jménem a parametry, které mohou se u různých typů lišit.

```
<Typ_Příznaku>
      <Parametr value="hodnota"/>
      ...
</Typ_Příznaku>
```

Program podporuje následující typy.

Spojité Haarovy příznaky
`DoubleHorizontal`, `DoubleVertical`, `TernalHorizontal`, `TernalVertical`

Diskrétní Haarovy příznaky

`DoubleHorizontalBinary`, `DoubleVerticalBinary`, `TernalHorizontalBinary`, `TernalVerticalBinary`

Local Rank Difference příznaky

`TFeaturesABDistance`, `TFeaturesMax`, `TFeaturesMin`

Kromě zde jmenovaných program podporuje některé další experimentální typy příznaků. U každého typu jsou pak definovány parametry, podle kterých budou příznaky při startu programu generovány.

Příznaky, které umí zpracovat program popsaný v kapitole 4.3 jsou pouze spojité Haarovy příznaky. Všechny jejich typy mají stejné parametry.

`MinWidth`
: Minimální šířka celého příznaku v pixelech. Mělo by se jednat o celočíselný násobek počtu bloků příznaku v horizontálním směru.

`MinHeight`
: Minimální výška příznaku v pixelech. Mělo by se jednat o celočíselný násobek počtu bloků příznaku ve vertikálním směru.

`MaxWidth`
: Maximální šířka příznaku.

`MaxHeight`
: Maximální výška příznaku.

`ShiftStepX`
: Krok v pixelech, se kterým budou generovány pozice příznaků v horizontálním směru.

`ShiftStepY`
: Krok v pixelech, se kterým boud generovány pozice příznaků ve vertikálním směru.

`SizeStepX`
: Krok, se kterým bude generována šířka příznaků.

`SizeStepY`
: Krok, se kterým bude generována výška příznaků.

Tyto parametry pak použije generátor příznaků k vytvoření všech možných instancí konkrétního typu. Celá definice jednoho typu příznaku pak může vypadat následovně.

```
<DoubleHorizontal>
```

```
      <minWidth value = "2"/>
      <minHeight value = "1"/>
      <maxWidth value = "20"/>
      <maxHeight value = "20"/>
      <shiftStepX value = "1"/>
      <shiftStepY value = "1"/>
      <sizeStepX value = "2"/>
      <sizeStepY value = "1"/>
</DoubleHorizontal>
```

5.2 Klasifikátor

Po natrénování je klasifikátor uložen jako subelement kořenového elementu konfiguračního XML. Tento přístup dovoluje například pokračovat v trénování klasifikátoru, testovat vlastnosti klasifikátoru na různých datech, vypsat odezvu klasifikátoru na testovací nebo trénovací vzorky, atd.

Kořenový element klasifikátoru je `AdaBoostClassifier`, který obsahuje definice jednotlivých slabých klasifikátorů (`WeakClassifier`). Každý slabý klasifikátor pak může obsahovat buď spojitý nebo diskrétní příznak. Element `DiscreteFeature` a `ContinuousFeature` definují parametry slabých klasifikátorů a jako subelementy mají definice jejich příznaků.

```
<AdaBoostClassifier description="">
<WeakClassifier index="0">
      <DiscreteFeature>
            <Feature/>
      </DiscreteFeature>
</WeakClassifier>
<WeakClassifier index="1">
      <ContinuousFeature>
            <Feature/>
      </ContinuousFeature>
</WeakClassifier>
...
</AdaBoostClassifier>
```

Vlastnosti slabého klasifikátoru se definují pomocí parametrů elementů `Continuous/DiscreteFeature`. V případě spojitých Haarových příznaků se jedna o parametry `threshold`, `alpha` a `parity`. Ostatní typy klasifikátoru zde mohou mít uložené jiné informace. Jako subelement je pak uložen konkrétní příznak. Haarovy příznaky zde

definují pouze svou pozici a velikost. Jiné typy příznaků zde mohou mít další informace v závislosti na parametrech s jakými byly trénovány. Zde je ukázka jednoho slabého klasifikátoru.

```
<WeakClassifier index="0">
      <ContinousFeature
            alpha="0.953803" error="0.12925"
            threshold="-0.40678" parity="-1">
            <HorizontalTernalFeature
                  positionX="8" positionY="4"
                  blockWidth="3" blockHeight="6"/>
      </ContinousFeature>
</WeakClassifier>
```

5.3 Spuštění trénování

V předchozích částech bylo popsáno, jaké vstupy program očekává a jaká nastavení podporuje. Tato podkapitola rozebírá, jak se program spouští a co se děje při běhu programu.

V současné době existuji funkční verze programu pro OS Windows i Linux. Obě ke svému běhu potřebují knihovnu `libxml2`, která zajišťuje vstup a výstup XML dat. Jediný parametr příkazového řádku je `-i` a jméno konfiguračního XML souboru, kde jsou specifikovány všechny potřebné informace pro trénování.

```
$AdaBoost -i jméno_konfigurace.xml
```

Při spuštění program načte konfiguraci, vzorky z datasetu a vygeneruje slabé klasifikátory podle nastavení v konfiguraci a spustí proces trénování. Při běhu program vypisuje na standardní výstup informace o průběhu. Na začátku se vypisují data o průběhu generování slabých klasifikátorů a při trénování data o přesností každého vybraného klasifikátoru na trénovací a testovací sadě, které lze později použít pro vyhodnocení chyby klasifikátoru. Po skončení program zapíše číselně svou odezvu na načtené vzorky. Z těchto informací lze pak získat ROC křivku a z ní odhadnout optimální klasifikační práh.

Hromadné spuštění

Pokud potřebujeme natrénovat klasifikátor pro konkrétní účel (detekce obličeje, ocr,...), není většinou potřeba spouštět více než jedno nebo několik málo trénování. V případě, že potřebujeme spustit trénování více, je zbytečné je všechny pouštět na jednom stroji, ale je možné využít distribuované zpracování na výpočetním clusteru, pokud je

dostupný. Důvody ke spuštění mnoha trénování mohou být různé. Například vyhodnocení vlivu různého nastavení příznaků na kvalitu klasifikátoru, porovnání různých typů příznaků nebo vzájemné porovnání klasifikátoru s klasifikátory jiného typu [10]. Takových trénování můžou být desítky až stovky – pro podobné případy je použití distribuovaného výpočtu velmi žádoucí hlavně z hlediska času. Některá trénování jsou velmi náročná a trénovací časy mohou dosáhnout až desítek hodin (v závislosti na velikosti trénovacích dat a počtu příznaků).

Byly vyvinuty nástroje a skripty pro automatické generování různých konfigurací a datasetů a spuštění trénování s mnoha konfiguracemi v systému SGE (Sun Grid Engine) [11]. SGE se skládá z hlavního uzlu (*master host*) který udržuje frontu úloh a z výpočetních uzlů (*execution hosts*), na kterých se úlohy spouští. Uživatelé přidávají úlohy, které chtějí zpracovat, do hlavní fronty. Hlavní uzel z nich automaticky vybírá a úlohy posílá ke zpracování na volné výpočetní uzly. Takto je možné spouštět standardní úlohy ale i úlohy, které běží paralelně na několika počítačích. Uživatel má nad svými úlohami plnou kontrolu a může kdykoliv u každé změnit nastavení nebo ji odstranit z fronty. Další výhodou SGE je že uživatel ovládá celý systém z jednoho místa.

Systém SGE dostupný na FIT VUT obsahuje desítky dedikovaných serverů a umožňuje připojit do clusteru i počítačové učebny v době, kdy se nepoužívají. Takto je možné získat systém sestávající se až z několika stovek výpočetních slotů.

6 Experimenty

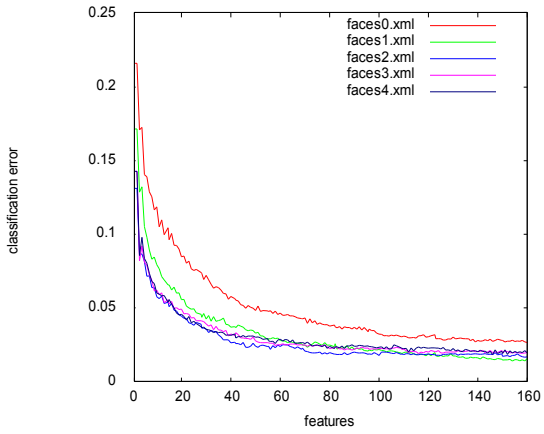
Bylo provedeno několik experimentů, které zahrnovaly trénování klasifikátorů programem popsaným v předchozí kapitole s různými nastaveními. První experiment se zaměřil na zjištění toho, jak moc použitá množina příznaků ovlivňuje průběh poklesu chyby klasifikátoru. V druhém experimentu bylo vytvořeno několik datových sad se stejnými vzorky, které byly rotovány o náhodný úhel v určitém rozsahu (který byl v každé datové sadě jiný). Tento experiment testuje, jaký vliv má rozptyl rotace trénovacích vzorků na chybu klasifikátoru. Výsledky AdaBoost jsou zde porovnány s neuronovou sítí.

Detektor obličeje

Velmi častou praktickou úlohou je detekování obličeje, které je zde použito jako ukázková úloha. Bylo natrénováno pět klasifikátorů s různou sadou příznaků a různou velikostí datových sad. Použitá datová sada vzorků byla z ČVUT. Velikost vzorků byla 26×26px .

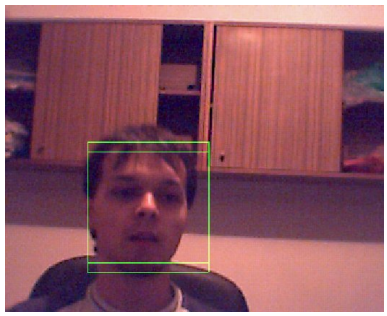
Klasifikátor	Počet příznaků	Doba trénování	Chyba
faces0.xml	1588	3min 26s	2,6 %
faces1.xml	5812	10min 54s	1,53 %
faces2.xml	14928	1h 9min 4s	1,67 %
faces3.xml	37088	2h 6min 57s	1,95 %
faces4.xml	53508	5h 4min 9s	1,94 %

Tabulka 1: Statistiky trénování.



Obr. 26: Chyba na různých klasifikátorech.

Počet použitých vzorků při trénování byl 24 000 (trénovací sada) a 8 000 (testovací sada). Poměr příkladů a protipříkladů byl 1/7 (tedy sedm krát více protipříkladů). Všechna trénování byla provedena s Haarovými příznaky. Generovaly se instance od minimální velikosti po určitou maximální hranici. Tedy vždy se generovaly příznaky s vysokými frekvencemi. Maximální velikosti příznaků byly 2×2px (`faces0.xml`), 4×4px (`faces1.xml`), 8×8px (`faces2.xml`), 16×16px (`faces3.xml`) a 26×26px (`faces4.xml`). Úplná sada příznaků byla tedy použita jen v klasifikátoru `faces4.xml`.

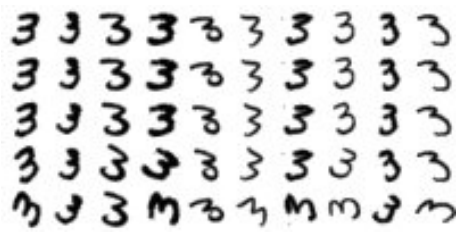


Obr. 27: Ukázka detekce pomocí klasifikátoru `faces4.xml`

Z obrázku 26, který zobrazuje pokles chyby klasifikátorů na testovací sadě vzorků, je zřejmé, že pro natrénování klasifikátoru s nízkou klasifikační chybou není nutné generovat všechny možné instance příznaků, ale stačí jejich určitá podmnožina. Všechna trénování skončila s přibližně stejnou chybou. Rozdíl je pouze v rychlosti, s jakou chyba klesá na testovací sadě – pokud je dostupná větší sada příznaků, chyba klesá rychleji, protože algoritmus má lepší možnost výběru. Počet generovaných slabých klasifikátorů také významně ovlivňuje rychlost trénování. Z tabulky 1 je vidět, že čas trénování stoupá, ale výsledná chyba klasifikátorů je ve všech případech srovnatelná.

Rozpoznávání číslic

Pro článek [10] byla provedena série testů zkoumající vliv rotace na přesnost klasifikátoru. Jako základní datová sada byl použit dataset s ručně psanými číslicemi MNIST. Velikost vzorků byla 20×20px .



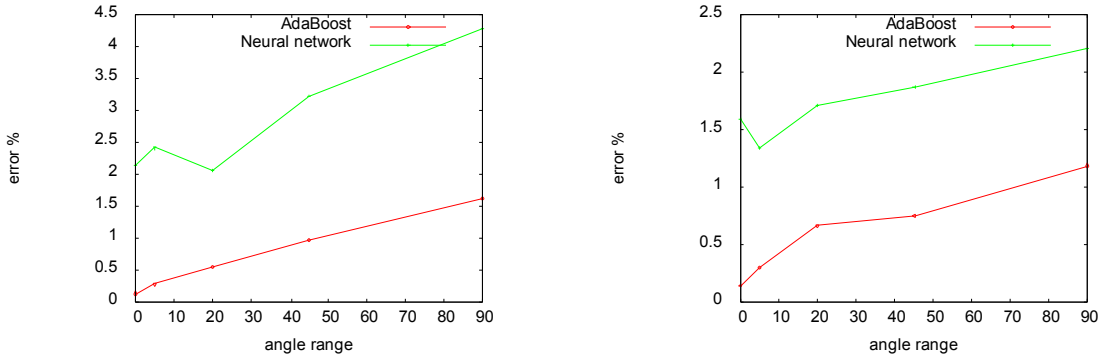
Obr. 28: Ukázka datasetu s číslicí 3.

Bylo vygenerováno 5 datasetů. V každém byly vzorky rotovány o náhodný úhel v určitém rozsahu (0, 5, 20, 45 a 90 stupňů). Na obrázku 28 je malá část z různých datasetů se vzorky číslice 3. V každém řádku jsou vzorky rotované v jiném rozsahu.

Trénovaly se vždy klasifikátory pro rozpoznání jedné číslice (0 nebo 1) od všech ostatních číslic. Velikost datových sad v experimentech byla 40 000 vzorků (trénovací data 20 000 příklady, 20 000 protipříklady) a 15 000 testovací sada (10 000 příklady, 5 000 protipříklady). Byla vždy použita kompletní sada Haarových příznaků. Počet vybíraných příznaků byl ve všech případech 120.

číslice/rozsah rotací	**0°**	**5°**	**20°**	**45°**	**90°**
0	0,12 %	0,27 %	0,55 %	0,97 %	1,62 %
1	0,14 %	0,30 %	0,66 %	0,75 %	1,18 %

Tabulka 2: Chyby klasifikátorů.



Obr. 29: Srovnání chyb AdaBoost a neuronové sítě.

Tabulka 2 ukazuje chyby klasifikátorů. Na obrázku 29 je srovnání s výsledky neuronových sítí (číslice 0 vlevo, 1 vpravo). Neuronové sítě byly trénovány s příznakovými vektory vybranými metodou AdaBoost. Je vidět, že klasifikátory mají nízkou chybu i při velkých rozsazích rotací.

Klasifikátory trénované se zde použitými datovými sadami dokáží odlišit pouze jeden typ číslic od ostatních a jsou proto samostatně nepoužitelné pro rozpoznání textu při OCR a nelze je tedy použít ani pro detekci číslic v obraze jako předchozí klasifikátory pro detekci obličeje. Jejich využití by mohlo být například v systému, kde je nutné rozpoznání ručně zadávaného textu (např. v PDA).

7 Závěr

V této práci byly shrnuty některé metody rozpoznávání vzorů v obraze. V druhé kapitole byly přiblíženy základy neuronových sítí, support vector machines a gaussovských modelů. Tyto metody dnes mají své využití v různých oblastech počítačové grafiky a počítačového vidění, avšak pro řešení problémů v časově omezených podmínkách nebo v real time aplikacích jsou mnohdy nepoužitelné.

V kapitole 3 je popsán algoritmus AdaBoost, který odstraňuje některé nedostatky jiných metod metod. Bylo uvedeno teoretické pozadí algoritmu. Tento algoritmus slouží k výběru klasifikátorů z dané množiny a jejich kombinaci do jednoho silného klasifikátoru. Jednou z velmi kladných vlastností algoritmu je to, že se během trénování postupně adaptuje na data v trénovací množině a s přibývajícími slabými klasifikátory opravuje chyby v silné klasifikační funkci. Bylo dokázáno, že celková chyba se s počtem použitých slabých klasifikátorů exponenciálně snižuje. V kapitole 3.3 byl popsán algoritmus WaldBoost, který používá AdaBoost k výběru a řazení slabých klasifikátorů podle jejich chyby a navíc k němu přidává sekvenční rozhodovací strategii, která zajišťuje, že pro provedení klasifikace není nutné vyhodnotit všechny slabé klasifikátory. Tato vlastnost umožňuje implementovat velmi rychlé detektory objektů.

Byla implementována knihovna pro detekci objektů v obraze pomocí obecných klasifikátorů založených na AdaBoost. Kapitola 4 popisuje detaily této knihovny. Nakonec je popsáno API, které je možné použít v dalších aplikacích. Lze zde také nalézt popis vlastností programu, který API využívá pro detekci objektů ve videu. Kapitola 5 detailně popisuje nástroj pro trénování klasifikátorů který je vyvíjen na UPGM FIT. Popisují se zde možnosti vstupu dat do trénování a možnosti konfigurace trénovacího procesu.

Poslední kapitola popisuje některé realizované experimenty s klasifikátory. Na úloze detekce obličeje bylo ukázáno, že pro rychlé natrénování poměrně přesného klasifikátoru postačuje pouze malá podmnožina příznaků. Počet dostupných příznaků ale ovlivňuje rychlost poklesu chyby. Druhý experiment ukazuje, že algoritmus AdaBoost je robustní proti rotacím vzorků a dokáže trénovat kvalitní klasifikátory i když jsou trénovací vzorky otočené o náhodný úhel.

Při pokračování projektu bude implementován modul s algoritmem WaldBoost v trénovacím nástroji a podpora WaldBoost klasifikátorů v knihovně. Vývoj knihovny se dále zaměří na podporu dalších typů příznaků, detekci rotovaných objektů a celkovou optimalizaci skenování obrazu.

Literatura

[1] Španěl M., *Rozpoznávání Gest ve Video Sekvencích*, Vysoké Učení Technické v Brně, 2003

[2] *Back-Propagation*, 2006, http://en.wikipedia.org/wiki/Back_propagation

[3] Vapnik V. N., Lerner A.,, *Pattern Recognition Using Generalized Portrait Method*, Automation and Remote Control, 1963

[4] Boser B. E., Guyon I. M., Vapnik V. M., A *Training Algorithm for Optimal Margin Classifiers*, 5th Annual ACM Workshop on COLT, 1992

[5] Myung J., *Tutorial on Maximum Likelihood Estimation*, Journal of Mathematical Psychology, 2003

[6] Freund Y., Schapire R., A ***Decision-theoretic Generalization of On-line Learning and an Application to Boosting***, Journal of Computer and System Sciences, 1997

[7] Viola P., Jones M., ***Robust Real Time Object Detection***, SCTV, Vancouver, Canada, 2001

[8] Šochman J., Matas J., ***WaldBoost - Learning for Time Constrained Sequential Detection***, Center for Machine Perception, Dept. of Cybernetics, Faculty of Electrical Engineering, CVUT Prague, 2005

[9] Wald A., *Sequential Analysis*, Dover, New York, 1947

[10] Šilhavá J., Herout A., Zemčík P. et al., *Platform for Evaluation of Image Classifiers*, Spring Conference on Computer Graphics, 2007

[11] Sun Microsystems, *N1 Grid Engine User Guide*, Sun Microsystems, 2005