VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHIC AND MULTIMEDIA

ADABOOST IN COMPUTER VISION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. MICHAL HRADIŠ
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHIC AND MULTIMEDIA

ADABOOST V POČÍTAČOVÉM VIDĚNÍ
ADABOOST IN COMPUTER VISION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Michal Hradiš

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Igor Potúček, Ph.D.

BRNO 2007

Zadání diplomové práce/4987/2006/xhradi05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií   Akademický rok 2006/2007

Zadání diplomové práce

Řešitel:	**Hradiš Michal, Bc.**
Obor:	Inteligentní systémy
Téma:	**AdaBoost v počítačovém vidění**
Kategorie:	Počítačová grafika

Pokyny:

1. Prostudujte algoritmus AdaBoost a příbuzné motody se zaměřením na jejich využití v počítačovém vidění.
2. Navrhněte systém pro experimentální vyhodnocování vlastnotí různých modifikací daných metod se zaměřením na výpočetní efektivitu, jednoduchou konfigurovatelnost a modularitu kódu.
3. Navržený systém implementujte spolu s vybranými verzemi algoritmu AdaBoost.
4. Proveďte experimenty na různých typech dat a vyhodnoťte výsledky.

Literatura:

- Na vyžádání u školitele.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí:	**Potůček Igor, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání:	28. února 2006
Datum odevzdání:	22. května 2007

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

L.S.

[illegible]

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení:	**Bc. Michal Hradiš**
Id studenta:	49413
Bytem:	Jesenická 101, 790 81 Česká Ves
Narozen:	22. 03. 1983, Šumperk

(dále jen "autor")

a

2. **Vysoké učení technické v Brně**
   Fakulta informačních technologií
   se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
   jejímž jménem jedná na základě písemného pověření děkanem fakulty:

   ....

   (dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   diplomová práce

Název VŠKP:	AdaBoost v počítačovém vidění
Vedoucí/školitel VŠKP:	Potúček Igor, Ing., Ph.D.
Ústav:	Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP:	....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické formě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

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 pořizování 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: ....

....	[illegible]
Nabyvatel	Autor

Abstrakt

V této diplomové práci jsou představeny nové obrazové příznaky „local rank differences“ (LRD). Tyto příznaky jsou invariantní vůči změnám osvětlení a jsou vhodné k implementaci detektorů objektů v programovatelném hardwaru, jako je například FPGA. Chování klasifikátorů s LRD vytvořených pomocí algoritmu AdaBoost bylo otestováno na datové sadě pro detekci obličejů. LRD v těchto testech dosáhly výsledků srovnatelných s výsledky klasifikátorů s Haarovými příznaky, které jsou používány v nejlepších současných detektorech objektů pracujících v reálném čase. Tyto výsledky ve spojení s faktem, že LRD je možné v FPGA vyhodnocovat několikanásobně rychleji než Haarovy příznaky, naznačují, že by LRD příznaky mohly být řešením pro budoucí detekci objektů v hardwaru. V této práci také prezentujeme nástroj pro experimenty s algoritmy strojového učení typu boosting, který je speciálně uzpůsoben oblasti počítačového vidění, je velmi flexibilní, a přitom poskytuje vysokou efektivitu učení a možnost budoucí paralelizace výpočtů. Tento nástroj je dostupný jako open source software a my doufáme, že ostatním ulehčí vývoj nových algoritmů a příznaků.

Klíčová slova

Abstract

In this thesis, we present the local rank differences (LRD). These novel image features are invariant to lighting changes and are suitable for object detection in programmable hardware, such as FPGA. The performance of AdaBoost classifiers with the LRD was tested on a face detection dataset with results which are similar to the Haar-like features which are the state of the art in real-time object detection. These results together with the fact that the LRD are evaluated much faster in FPGA then the Haar-like features are very encouraging and suggest that the LRD may be a solution for future hardware object detectors. We also present a framework for experiments with boosting methods in computer vision. This framework is very flexible and, at the same time, offers high learning performance and a possibility for future parallelization. The framework is available as open source software and we hope that it will simplify work for other researchers.

Keywords

Boosting, AdaBoost, Local Rank Differences, LRD, Computer Vision, Face Detection, Haar-like Features, WaldBoost,

Citace

Michal Hradiš: AdaBoost in Computer Vision, diplomová práce, Brno, FIT VUT v Brně, 2007.

AdaBoost in Computer Vision

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Igora Potúčka, Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Michal Hradiš

Acknowledgements

First of all, I would like to express my thanks to Adam Herout for directing my interests towards computer graphic and later also towards computer vision and for helping me in many ways during my work on this thesis. I am very grateful to Igor Potúček, Víťa Beran, Jana Šilhavá, Michal Seman, Michal Španěl and all the other people at UPGM for providing great inspiration and for their help with increasing my knowledge of computer vision and machine learning. Especially, I would like to thank Pavel Zemcik for many interesting and guiding discussions and for the opportunity to participate in the research at UPGM. I would like to thank Roman Juránek for his help with the boosting framework and his great support for the experiments we have performed.
A part the work was done at the Spring School on Image Processing in Szeged. I would like to thank my team mates Ilona Jedyk, Ágoston Róth and Sándor Szabó for their great work on the facial expression recognition demonstration application.
The work presented in this thesis has been financially supported by the Czech Science Agency grant "Algorithms of image recognition" GACR GA201/06/1821, IST EU grant CARETAKER – "Content Analysis and REtrieval Technologies to Apply Knowledge Extraction to massive Recording" EU-6FP-IST 027231, Czech Academy of Sciences Grant Agency grant "Rapid prototyping tools for development of HW-accelerated embedded image- and video-processing applications" GA AVCR, 1ET400750408, and Ministry of Education, Youth, and Sport grant "Research of information technologies form the security point of view", CEZ MSMT, MSM 0021630528.

Contents

Contents 1

1 Introduction 3

1.1 Introduction to Classification 4

1.2 Classification Formalization 6

1.3 Limitations of Classification and Machine Learning 6

2 Boosting 8

2.1 AdaBoost 8

2.2 Real AdaBoost 9

2.3 AdaBoost Discussion 11

2.4 Improving Learning Process 13

2.5 Speed and Accuracy Tradeoff 15

2.6 Learning Speedup 17

3 Data Transformations in Computer Vision 18

3.1 Haar-like Features 18

3.2 Gabor Features 20

3.3 Local Binary Patterns 21

3.4 Linear Transforms 21

4 Weak Learning Algorithms 23

4.1 Histograms 23

4.2 Decision Trees and Decision Stumps 24

5 New Image Features 25

5.1 Min/Max Features 26

5.2 Local Rank Differences 27

5.3 LRD in FPGA 28

6 Experimental Boosting Framework 31

6.1 Requirements 31

6.2 Basic Principles 33

6.3 Framework Structure 34

6.4 Data Source 36

6.5 Features 38

6.6 Weak Learners and Hypotheses 38

6.7 The Boosting Algorithms and Cascade 39

6.8 Potential for Parallelization 39

6.9 Effective Computation of Features 40

6.10 Support Tools 42
7 Results 43
7.1 Comparison of Different Types of Features 44
7.2 Threshold Estimation Precisions 45
7.3 Restricting the Sizes of LRD 47
7.4 Prediction Value Quantization 49
7.5 Convolution Quantization 50
8 Conclusions and Future Work 52
References 53

1 Introduction

This thesis falls into the broad field of machine learning, which emerged from artificial intelligence research. Machine learning studies how to automatically extract information from data and possibly act according to that information. In the context of this work, learning is understood as inductive inference where rules and patterns are extracted from observed examples which incompletely represent some stochastic phenomenon. More specifically, the theme of this thesis is supervised machine learning, where each observed training example has an explicitly assigned label. The task of the learning method is then to create a prediction rule based on the information extracted form the training data and the corresponding labels. This prediction rule is then used to predict labels for unseen data samples. In general, the labels can be either discrete, in which case we speak about pattern classification – or real-valued in regression problems. Only classification is considered in this thesis.

Relatively recently, large margin classification methods emerged as practical results of statistical learning theory. Large margin classifiers search for such prediction rules which maximize distance of almost all examples from the decision boundary. The most theoretically and practically studied classes of large margin classifiers are support vector machines [1] (SVM) and boosting. In this thesis we focus on AdaBoost [2][3][4], which is one of the boosting methods, and its use in computer vision.

AdaBoost and its modifications have been successfully used in practical computer vision applications [5][6][7][8][9][10][11][12][13]. For example, the state of the art object detection classifiers are variations of cascade of boosted Haar-like features [7][8][9]. In object detection, image sub-windows on all positions, of different sizes and possibly rotations are scanned with a classifier. This results in very large number of classified image regions which places high demands on the computational effectiveness of the classifier. Haar-like features are very suitable for detection tasks as they can be computed very fast in constant time independent on their size using integral image [5][6]. In the considered approach, simple (weak) classifiers are each created based on a single Haar-like feature. AdaBoost then selects some of the weak classifiers and combines them into very accurate prediction rule. The prediction rule is a weighted majority vote of the weak classifiers and gets more accurate as more weak classifiers are added. This allows trade-off between classification accuracy and computational time. To optimize performance, some kind of cascade of consequently more complex classifiers is usually used. In such cascade, each stage rejects those image regions which are classified with enough confidence as background. Such cascade gradually lowers the false positive rate, which has to be very low in detection problems. In chapter 2, we give an overview of some boosting methods. Chapter 3 presents some of the data transformation techniques used in computer vision. Namely, the presented techniques are: principle component analysis (PCA), linear discriminant analysis (LDA), Haar-like features, Gabor features and local binary patterns (LBP). Some of the commonly used weak learners are described in chapter 4.

Haar-like features offer high performance if the classifier is evaluated on general purpose CPU. On the other hand, they are not very suitable for FPGA[1] and generally hardware implementation, which could be used in diverse embedded applications. The main issues preventing efficient implementation in FPGA are the need of normalization and the need of random memory access to each pixel of the integral image. Although, there exist some FPGA implementations of object detectors with Haar-like features [14][15][16], they provide relatively low performance. In chapter 5, we present new image features which could be effectively implemented in FPGA. Namely, the local rank differences [17] (LRD, see section 5.2) offer comparable discriminative power as Haar-like features, but the FPGA implementation [18] can evaluate one of these features in each clock cycle instead of six or eight cycles in the case of Haar-like features. Moreover, the LRD features do not require any explicit normalization as they implicitly normalize the results with local histogram equalization.

In the search for new features suitable for FPGA, it was necessary to create an experimental framework which could be used to evaluate performance of the newly suggested features. This framework has to be modular and has to offer high performance. Chapter 6 describes the framework. In chapter 7, we present many experimental results which were obtained using the framework. These results are mainly focused on the performance of the LRD. Finally, the achieved results are summarized and future work is suggested in chapter 0.

1.1 Introduction to Classification

Let's now look at an example of supervised machine learning approach to a simple classification task. Imagine we need to create a machine which can distinguish between horses and zebras. Do not occupy your mind with the question why should we do such thing and let's focus only on the question how to do it. Our classification machine will physically consist of a large black box with three doorways. One of the doorways will serve as entrance and the other two will serve as exits. Everything that leaves one exit will be considered a horse and everything that leaves the other exit will be further treated as a zebra. Inside the black box, there will be a computer to run our classification program (or prediction rule) and a hard-working but simple-minded[2] person who will be able to perform simple routine measurements on the animal inside the box and enter the results of the measurements into the computer. We will call the person Sensor in the further text. To sum it up, the

[1] Field-programmable gate array is a class of integrated circuits containing programmable logic components and programmable interconnects.

[2] He will not be able, for example, to answer questions like: "Is that a zebra?"

operation of our classification machine will consist of following steps. After an animal enters the box, Sensor performs some measurements and enters the results into the computer running classification software. The classification software then decides the fate of the animal based on the data obtained from Sensor.

Now, as we have the basic structure of the classification machine, we need to decide what Sensor will measure and how the classification software will work. Let's look at the measurements first. In machine learning, single measurement (or the measurable property of the phenomena being observed) is called a feature and a set of measurement results describing single object is called a feature vector. There are two basic requirements for the features used for classification. First, the set of features must be discriminative. Meaning the features should contain relevant information which can be used to distinguish between objects belonging to different classes. For example, a set of features consisting of the number of legs, the number of eyes and the weight of the animal does not contain much relevant information to distinguish between horses and zebras. On the other hand, a set of features describing all colors present on the animal skin and average length and width ratio of the colored patches should provide highly relevant information for our classification task. The features very much influence quality of the resulting classifier and their design offers great opportunity for human innovation.

The other usual requirement is to keep the number of features reasonably low. The obvious reason for doing so is computational complexity of the classifier. The computational complexity always grows – in some cases (e.g. artificial neural networks) even very fast – with the number of features for both classifier training and prediction rule evaluation. There is also the cost of the measurement itself, which can become very high in cases where specific sensors are needed or human involvement is necessary. An example of such area of applications is medical diagnostic, where each further examination can cost hundreds of Euros. In the case of our classification machine, there will be the problem with measurement cost too as we decided to use human to carry out the measurements. To reduce this cost, we will employ very simple-minded person. Because of this, we need to choose as simple and as well-defined features as possible, otherwise the measurement time may significantly reduce the throughput of the system. One of such simple features can be the number of light and dark transitions on the animal skin along a horizontal line. This feature itself should also be discriminative enough to distinguish between horses and zebras.

Next, we need to choose the classification method. This is another crucial point and the choice can influence the performance of the resulting application greatly. Classification methods can differ in the time needed for learning, computation complexity of the classifier, the complexity of the decision boundary, generalization properties (the performance on unseen data) and many other attributes. Some of the available classification methods are naive Bayes classifier, artificial neural networks, SVM, decision trees, AdaBoost, K-nearest neighbor and many others. In our case, we have the advantage of having only one dimensional feature space. Moreover, we can assume that zebras have more light and dark transitions along the horizontal line then most of the horses have. So, it is not unreasonable to think that we can find a threshold for this feature which can separate horses from zebras with relatively low error.

The example of the machine classifying horses and zebras is very simple and we could set the threshold ourselves using only our intuition and very few examples. This way we would, in fact, become the learning algorithm ourselves. Obviously, this is not possible in more complex tasks where automatic learning algorithms must be used. Since, we want to discus machine learning, we will set the threshold automatically. To do so, we need the training examples first. We have to gather two separate herds of zebras and horses and let Sensor measure and note the number of intensity transitions on the skin of each animal. We also have to note which measurements belong to zebras and which belong to horses. Having this training data, we can easily automatically set the threshold value such that the classification error on the training data will be minimal. At this point, we have all the parts of the machine ready.

1.2 Classification Formalization

The task of machine learning algorithm in supervised classification problem is to find a rule (or hypothesis) which assigns an object to one of several classes based on external observations. Such prediction rule can be formalized as a function $h: \mathrm{X} \to \mathrm{Y}$ where X is the input domain (the feature space) and Y is the set of possible labels. Let $S = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ be a set of training examples where each instance x_i belongs to X and each y_i belongs to Y. When referring to single example, a letter i is usually used. The training samples are usually supposed to be generated independent identically distributed (i.i.d) according to unknown probability distribution P on $\mathrm{Z} = \mathrm{X} \times \mathrm{Y}$. The task of the learning algorithm is then to estimate such function h which minimizes some objective error function on the training sample set S. Although, Y can be an arbitrary finite set, we consider only binary classification in this thesis where $\mathrm{Y} = \{-1, +1\}$.

1.3 Limitations of Classification and Machine Learning

There are many limitations on what can be achieved by machine learning methods in classification tasks. The first fundamental limitation is the fact that the true class-conditional probability density functions (PDF) of different classes can overlap. In other words it is possible that single data point in the feature space can be generated by objects from more than one class. The amount of the overlap depends on what we can measure about the observed object. More or better sensors can in many cases solve this kind of difficulties, but this may not be feasible considering the costs.

Another limitation arises from the size of the feature space. Considering, we always have only a finite set of training examples, we are not able to estimate the true class-conditional PDFs exactly. This problem becomes more profound with higher number of dimensions. For example, in a small feature spaces (two or three dimensions), it would be possible to discretize the feature space and estimate the class-conditional PDFs for each of the discrete points with high accuracy using only moderate number of training examples. With higher number of dimensions, this approach becomes infeasible, because the number of examples needed for reliable PDF estimation becomes extremely high. To be more precise the size of the feature space increases exponentially with the number of dimensions. This exponential increase is sometimes referred to as the "curse of dimensionality" [19].

The final limitation is caused by our ability to acquire suitable training set. In machine learning, it is generally assumed that the samples from training set are generated i.i.d. according to the same probability distribution as the unseen samples. In practice, this assumption is often violated as it is not possible or affordable to obtain training set in exactly the same conditions as the resulting application will work in. The performance of the classifier then largely depends on the degree of similarity of the probability distributions from which both training and unseen samples are generated.

2 Boosting

The term boosting refers to a group of ensemble supervised learning algorithms. The basic idea of these algorithms is to iteratively combine relatively simple prediction rules (weak classifiers) into very accurate prediction rule (strong classifier). In most of the boosting algorithms, the weak classifiers are linearly combined. In the case of two-class classification, the strong classifier is the weighted majority of the votes. For introduction to boosting look at [3][4].

Boosting has its roots in the PAC (Probably Approximately Correct) machine learning model [20][21]. In this framework, the learner's task is to find – with a high probability – a bounded approximation of a classification function using only training samples which are labeled by this particular function. The PAC model constrains the learning methods in terms of their effectiveness – the learning time must be polynomial-bounded as well as the number of needed training samples. The question, if a learning algorithm which performs just slightly better then random guessing in the PAC model can be boosted into arbitrarily accurate learning algorithm was first suggested by Kearns and Valiant [22][23]. The first polynomial-time boosting algorithms were introduced by Freund [24] and Schapire [25]. These early algorithm, however, suffered from many drawbacks. For example, they needed some prior knowledge of the accuracies of the weak classifiers and the performance bound of the final classifier depended only on the accuracy of the least accurate weak classifier.

The AdaBoost algorithm, which was first introduced by Freund and Schapire [2], solved most of the practical drawback of the earlier boosting algorithms. In the original algorithm the output of weak classifiers is restricted to binary value and thus the algorithm is referred to as discrete AdaBoost. Schapire and Singer [26] introduced real AdaBoost, which allows confidence rated predictions and is most commonly used in combination with domain-partitioning weak hypotheses (e.g. decision trees). The following text introduces the original AdaBoost algorithms and discusses their general properties and performance. In the further text, some modifications of the original algorithms are discussed mostly focusing on convergence speed, accuracy-speed trade-off and noise resistance.

2.1 AdaBoost

AdaBoost calls a given weak learning algorithm repeatedly in a series of rounds $t = 1,\ldots,T$. In each iteration, the weak learning algorithm is supplied with different distribution D_t over the example set S. The task of the weak learner is to find a hypothesis $h_t : \mathrm{X} \rightarrow \{-1,+1\}$ minimizing a classification error in respect to the current distribution D_t:

$$\varepsilon_t = P_{i \sim D_t}\left[h_t(x_i) \neq y_i\right] \tag{1}$$

The weak hypothesis is then added to the strong classifier with a coefficient α_t which is selected according to the error ε_t of the hypothesis h_t on the current distribution D_t:

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

The final strong classifier is a linear combination of the selected weak hypothesis:

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

After a weak hypothesis is selected, new distribution D_{t+1} is generated in such way that the weights of the samples which are correctly (respective wrongly) classified by h_t decrees (respective increase):

$$D_{t+1}(x_i) = \frac{D_t(x_i)\cdot\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \qquad (2)$$

Maintaining the distribution D_t is one of the fundamental principles of AdaBoost. The weight $D_t(i)$ of sample i in step t reflects how well the sample is classified by all weak hypotheses selected in previous rounds. The complete discrete AdaBoost algorithm is shown in figure 1.

```
Given $S = \langle(x_1, y_1),\ldots,(x_m, y_m)\rangle$, $x_i \in X, y_i \in Y = \{-1,+1\}$
Initialize $\boldsymbol{D_1}(x_i) = 1/m$.
for $t = 1,\ldots,T$:
    Train weak learner using distribution $\boldsymbol{D_t}$.
    Get weak hypothesis $h_t : X \to \{-1,+1\}$.
    Choose $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    Update: $D_{t+1}(x_i) = \frac{D_t(x_i)\cdot\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
    where $Z_t$ is a normalization factor.
Output the final hypothesis: $H(x) = sign\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right)$
```

Figure 1: The original version of AdaBoost[2] with notation modified according to [26].

2.2 Real AdaBoost

Since the output of weak hypothesis in the original AdaBoost algorithm is binary, no information about how well the samples are classified by the weak hypotheses is available to the strong classifier. This way, valuable information which could otherwise improve the classification accuracy is discarded. Schapire and Singer [26] proposed a generalization of the original algorithm which can utilize prediction confidences. The authors have also shown how to generate the confidences of predictions and they have defined new function which should be minimized by the weak learner to obtain optimal predictions according to the speed of training error bound minimization. This generalization is sometimes referred to as real AdaBoost, since the output of a weak hypothesis can be any real number.

The real AdaBoost algorithm is in most aspects identical to discrete AdaBoost from figure 1. The only changes are that the weak hypotheses now have the form of $h_t : \mathrm{X} \rightarrow R$ and the selection of the α_t coefficients is not directly specified. The α_t coefficients can be selected in different ways depending on the type of the weak hypothesis. If no constraints are placed on the result of the weak hypotheses, the optimal α_t coefficients can not be found analytically. Schapire and Singer [26] present a general numerical method for choosing optimal α_t that uses binary search. However, it is not usually necessary to use this numerical method, since in the case of domain-partitioning weak hypotheses it is possible to find optimal α_t analytically.

To simplify notation, we will omit the t subscripts in further text as they will not be relevant. Moreover, let us fold the α coefficients into h. In other word, let's assume that the weak learner can freely scale any weak hypothesis h by any constant factor $\alpha \in R$.

Let us now explain the selection of the prediction values of h in the case of the domain-partitioning weak hypotheses. Each domain-partitioning hypothesis is associated with a partition of X into disjoint blocks $X_1, \ldots, X_N$ which cover all of X and for which $h(x) = h(x')$ for all $x, x' \in \mathrm{X}_j$. What this means is that the prediction of $h(x)$ depends only on which block X_j the instance x falls into. Let $c_j = h(x)$ for each $x \in \mathrm{X}_j$. For each j and for $b = \{-1, +1\}$ let

$$w_j^b = \sum_{i: x_i \in \mathrm{X}_j \wedge y_i = b} D(i) = P_{i \sim D}\left[x_i \in \mathrm{X}_j \wedge y_i = b\right]$$

be the weighted fraction of examples with label b which fall in block j. Then the optimal value of c_j is:

$$c_j = \frac{1}{2} \ln\left(\frac{w_+^j}{w_-^j}\right) \qquad (3)$$

The blocks of domain-partitioning hypothesis can be either implicitly given or variable (e.g. in decision trees). If the blocks are variable, some optimization criteria must be used to set the boundaries of the blocks. The weighted error criteria (1) can be used but does not provide optimal performance. Optimal in terms of training error bound minimization[1] is the criteria based on minimizing the normalization factor Z in the reweighing equation (2):

$$Z = 2\sum_j \sqrt{w_+^j w_-^j} \qquad (4)$$

[1] See section 2.3 for further information.

2.3 AdaBoost Discussion

The large advantage of the AdaBoost algorithm is that it provably and very fast converges to a hypothesis with low error on the training samples. This is true if the weak learner can constantly find weak hypotheses with an error which is lower then random guessing on the current distribution D_t. In [26], the authors show that the training error of the final classifier is bounded as follows:

$$\varepsilon = \frac{1}{m}\left|\{i : H(x_i) \neq y_i\}\right| \leq \prod_{t=1}^{T} Z_t = \frac{1}{m}\sum_i \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right)$$

The main consequence of this bound is that the weak learner should try to minimize Z_t on each round of boosting. This error bound is also the foundation for the choice of the prediction values in (3) and the minimization criteria for decision trees (4).

The effect of the reweighing equation (2) may not be fully clear. We will now look little bit closer on the effect it has in the training process. The first insight is that the equation makes weights of the wrongly classified samples larger and weights of the correctly classified samples smaller. This, intuitively, causes that the weak learner focuses more on hard examples (the examples mostly misclassified by the previously selected weak hypotheses). The hard examples are close to the decision boundary in the feature space. In this sense, they are very similar to the support vectors in SVM. Another, not so clear, effect of the re-weighting formula is that the currently selected weak hypothesis is totally independent on the hypothesis selected in the directly preceding round of boosting. This independence, however, does not hold for the other already selected weak hypothesis. Šochman and Matas [7] proposed a simple method to extend this independence to all previously selected weak hypotheses. Their Adaboost with totally corrective updates increases the convergence rate of learning without increasing the complexity of the combined hypothesis. We discuss this topic further in chapters 2.4.1.

Let's now look little bit closer at the learning algorithm – at its computational complexity and how it is practically implemented. We will assume a typical problem of image classification. In image classification, some wavelets are usually used to transform the original image data into more suitable representation. The number of features after such transformation can be very high. For example, in [5] the authors use 180,000 Haar-like features for samples with dimensions 24x24 pixels. In each iteration of the learning algorithm, the weak hypotheses have to be newly learned on the current distribution, which implies that the features are computed for each of the training examples. This is because the feature vectors for the training data usually don't fit into the memory. This is basically the most time consuming part of the AdaBoost algorithm. The other parts, the choice of α coefficient and reweighing of the examples, do not represent much computational burden as they already involve only single weak hypothesis. Based on this, the computational complexity of the learning algorithm is $\theta(N \cdot M \cdot T)$ where N represents the number of examples, M represents the number of the features and T is the number of algorithm iterations. This leads to a very nice property of the AdaBoost algorithm, which is that the computational complexity of the learning algorithm is independent on the number of previously selected weak hypotheses. Although, the computational complexity is relatively low, it can still be a limiting factor in some cases. In section 2.6 we discuss some methods to improve the learning speed.

Although, the minimization of error on training samples is necessary, the most important is the performance on unseen data. In [2], the authors propose an upper bound on the generalization error using the Vapnik-Chernonenkis theory. This upper bound gets looser with higher VC-dimension of the strong classifier, in other words, it depends on the number and complexity of the weak hypotheses. Such upper bound on generalization error suggests that an optimal length of the strong classifier can found. Classifiers shorter then the optimal length should be too simple to capture the structure of the data and longer classifiers should be too complex to be reliable learned from the data available and AdaBoost should overfit. Although, this method is theoretically sound, it is not consistent with experiments on real-world problems [29][30][31]. In the practical experiments, the training error often decreases or at lest does not increase even after hundreds of training rounds.

To fill the gap between theory and practice, Schapire et al. [32] proposed an alternative method to study the generalization properties of AdaBoost which is based on margins. The term margin refers to the distance of samples from the decision boundary of the classifier. In other words, it represents the degree of confidence of the classifier. The authors show that larger margins imply lower generalization error independent on the length of the classifier and they show that AdaBoost tends to increase the margins of the training examples. In [26], the authors extend the work of Schapire et al to real AdaBoost. They propose new upper generalization error bound based on margins. They conclude, according to this upper bound, that it is a bad idea to allow weak hypotheses which sometimes make predictions that are very large in magnitude. Such large predictions may dramatically reduce the margins of some of the training samples which can consequently have an adverse effect on the generalization error.

In the case of the domain-partitioning weak hypothesis, it is possible to obtain very large prediction values. It may even happen that one of the blocks contains samples only from single class. In such case the prediction value, according to equation (3), is equal to either positive or negative infinity. To smooth the prediction values, Schapire and Singer [26] propose to use smoothing parameter ε when choosing the prediction value:

$$c_j = \frac{1}{2}\ln\left(\frac{w_+^j + \varepsilon}{w_-^j + \varepsilon}\right)$$

where ε should take some appropriately small value. Because both w_+^j and w_-^j are bounded between 0 and 1, the addition of ε has the effect of bounding $\left|c_j\right|$ by

$$\frac{1}{2}\ln\left(\frac{1+\varepsilon}{\varepsilon}\right) \approx \frac{1}{2}\ln\left(\frac{1}{\varepsilon}\right)$$

The effect of the addition of ε on the convergence of the algorithm is negligible, since the value of Z is weakened only slightly if $\varepsilon << 1/2N$:

$$Z = 2\sum_j \sqrt{w_+^j w_-^j} + \sqrt{2N\varepsilon} \tag{5}$$

,where N stands for the number of the block in the partition. Schapire and Singer state that they have typically used ε on the order of $1/m$ where m is the number of training samples.

Although, the resistance of AdaBoost algorithm to overfitting is very high, on the other hand, it is highly susceptible to noise in the data. For example, if the training dataset contains two identical samples each belonging to different class, the algorithm gradually focuses only on these two samples ignoring the other samples. To be more precise, these two samples accumulate all the weights from other samples up to the machine numerical precision. This is caused by the fact that AdaBoost maximizes margins on all samples. This behavior is related to the term hard margins. When using hard margins, the size of margin depends on the sample closest to the decision boundary. In SVM, this problem was revealed very soon, as in the non-separable case some equations do not have a solution. On the other hand, the strong hypotheses found by AdaBoost are often still meaningful. In SVMs, the non-separable problem was solved by soft margins [33][34], which allow some samples to violate the margin. Gunnar Rätsch [35] used the ideas from SVMs and proposed one of the first modifications of AdaBoost with soft margins which still fits into a general boosting framework.

2.4 Improving Learning Process

The AdaBoost algorithm does not find optimal classifier in the terms of accuracy and the number of weak hypotheses. This is caused by the greedy nature of the algorithm. To find optimal classifier of given length could be vital in some applications, especially if real time performance is necessary. In this section, we introduce the totally corrective algorithm with coefficient updates [7] (TCAcu) which refines the prediction values of weak hypotheses and FloatBoost [8] which performs floating search in the space of weak hypotheses. Except these two algorithms, other solutions exist, for example, based on linear programming [36].

2.4.1 TCAcu

Šochman and Matas [7] proposed a modification of discrete AdaBoost algorithm which iteratively refines the predictions of previously selected weak hypotheses. The authors named the algorithm the totally corrective algorithm with coefficient updates (TCAcu). The idea behind this modification is that the predictions of selected weak hypotheses become suboptimal with additional rounds of boosting, but can be refined in each step of boosting in iterative process which requires only minor computational power. TCAcu assures that in each round of boosting the selected weak hypothesis is the most independent on all weak hypotheses selected in all previous rounds. This is a substantial improvement over the AdaBoost algorithm where the newly selected weak hypothesis is independent only on the weak hypothesis selected in the previous round. TCAcu provably tightens the bound on training error without increasing the classifier complexity.

TCAcu is almost identical to discrete AdaBoost from figure 1 except the totally corrective step (TCS) which is performed after each round of boosting. See figure 2 for pseudo code of the totally corrective step. TCS itself is basically the discrete AdaBoost algorithm. The major difference is that in TCS the set of weak hypotheses is limited to already selected ones and the weak hypotheses are not appended to the strong classifier, rather the corresponding α_t coefficients are summed.

```
Initialize $\boldsymbol{D'_1} = \boldsymbol{D_t}$ .
for $j = 1,\ldots,J_{\max}$ :
    Select weak hypothesis $q = \arg\max_{q=1..t} \left| P_{i\sim D'_j}\left[h_q(x_i) \neq y_i\right] - \frac{1}{2} \right|$
    If $\left| P_{i\sim D'_j}\left[h_q(x_i) \neq y_i\right] - \frac{1}{2} \right| < \Delta_{\min}$ then exit the loop.
    $\varepsilon_j = P_{i\sim D'_j}\left[h_q(x_i) \neq y_i\right]$
    Let $\alpha'_j = \frac{1}{2}\ln\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$
    Update: $D'_{j+1}(x_i) = \frac{D'_j(x_i)\cdot\exp(-\alpha'_j y_i h_q(x_i))}{Z_j}$
    where $Z_j$ is a normalization factor.
Assign $\boldsymbol{D_t} = \boldsymbol{D'_j}$
```

Figure 2: The totally corrective step [7].

2.4.2 FloatBoost

The FloatBoost algorithm [8] performs floating search in the space of weak hypotheses. This algorithm is based on AdaBoost and adds a backtracking phase. In this backtracking phase, those hypotheses which cause performance drops are deleted. The authors evaluated the performance of FloatBoost on face detection task and concluded that FloatBoost creates classifiers with lower number of weak hypotheses and lower error rates at the expense of longer training time. They report training time five times longer than that of AdaBoost.

2.5 Speed and Accuracy Tradeoff

For applications where the speed of classification is the most critical aspect, it is possible to train the classifiers in such way that for samples which are easy to classify, only low number of the weak hypotheses is evaluated. Here the term "easy to classify" refers to samples which can be at certain point of classifier evaluation assigned an appropriate label with sufficiently high confidence. This approach is mostly used in real-time face detection task. Generally, in object detection, image sub-windows on all positions, of different sizes and possibly rotations, are scanned with the classifier. This gives extremely high number of the classifier evaluations and places high demands on computational effectiveness of the classifier. Although, the classifiers used in face detection usually consist of hundreds of weak hypotheses, the average number of weak hypotheses evaluated per single sub-window can drop even to five.

The discussed approach does not have to necessary result in reduction of classification accuracy. In the case when large number of training samples is available, it is possible to discard the easy examples the same way as during classification and then replenish the training set with new samples which are not discarded by the current classifier. This technique is called bootstrapping and is commonly used in machine learning.

In following text, we introduce two techniques to tradeoff strong classifier speed and accuracy. First presented method is a cascade of consequently more complex classifiers which has been used, for example, by Viola and Jones [5] in their face detection system. Second presented method to tradeoff between classifier speed and accuracy is the WadlBoost algorithm [37] which introduces early termination thresholds of the strong classifier sum.

2.5.1 Cascade of Boosted Classifiers

A cascade of boosted classifiers was first used by Viola and Jones in their real-time face detection system [5]. This solution was approximately fifteen times faster than any other face detector of that time and modifications of their solution still keep the status of the state of the art real-time object detectors. Viola and Jones used a cascade of consequently more complex AdaBoost classifiers with decision stumps as weak classifiers and Haar-like features[1].

The scheme of the classifier cascade can be found on the figure 3. The main idea of the detection cascade is that smaller, and therefore more efficient, boosted classifiers can reject many of the background sub-windows while keeping almost all face sub-windows. This is achieved by adjusting the threshold of the boosted classifier so that the false negative rate is close to zero. The cascade is in principle a degenerated decision tree where, in each node, is decided if the sample

[1] For more information about the Haar-like features see section 3.1.

probably belongs to background or further information is necessary to classify it. The positive result form one cascade stage triggers the evaluation of consequent classifier. This approach benefits from the fact that in detection task the overwhelming majority of sub-windows belongs to background, thus the classification speed depends almost only on the average classification time for background samples.

The cascade is trained using bootstrapping – the subsequent classifiers are trained using only those training samples which pass through all of the previous stages. This allows the training process to effectively and precisely estimate the weak hypotheses even on the very hard and extremely rare examples which pass to the final stages of the cascade. The bootstrapping requires enormous supply of background samples as the rejection ratio in the later stages can reach 1:1000 and less.

One of the disadvantages of the detection cascade is that the results of previous stages are discarded, even thou they can provide relatively good prediction on the samples which pass the stage. This results in longer classifiers in the later stages then would be possible to achieve if this information was used. This fact was addressed by Xiao at al. [9]. In their boosting chain they use the classifier of previous stage as the first weak hypotheses of current classifier.

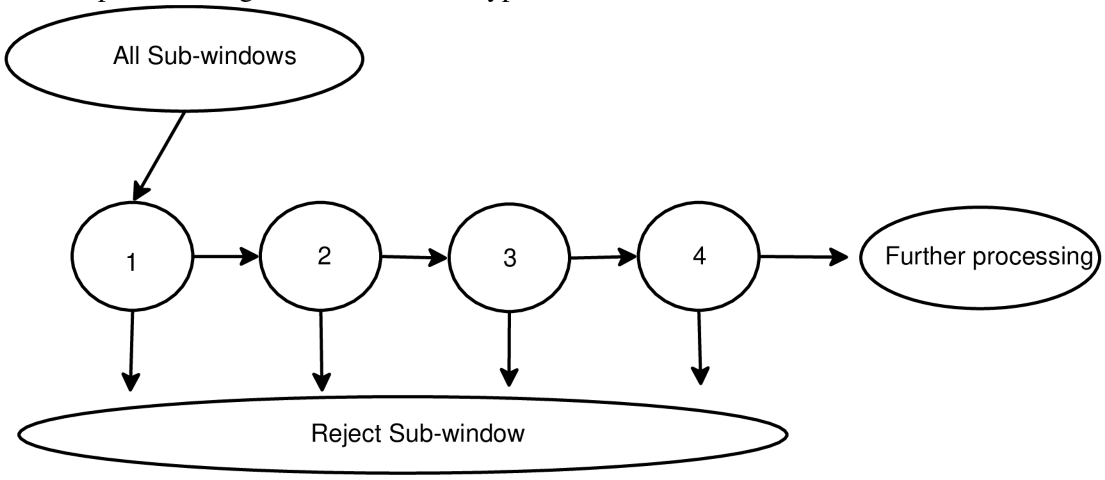


Figure 3: The detection cascade of boosted classifiers.

2.5.2 WaldBoost

Šochman and Matas in their WaldBoost algorithm [37] combine AdaBoost with Wald's sequential probability ratio test which solves the problem of creating optimal classification strategy in terms of the shortest average decision time subject to a constraint on error rates. WaldBoost classifier is almost identical to the AdaBoost classifier, except each weak hypothesis can be assigned two (for each class) early termination thresholds. If the strong classifier sum exceeds one of these thresholds, the sample is classified to corresponding class, otherwise the evaluation of the classifier continues. The termination thresholds are selected to achieve desired false positive and false negative rates. The classifier is trained with bootstrapping as in the case of cascaded classifiers. This implies that in detection tasks, only thresholds for rejecting background samples are used, as it is usually not possible to get more face examples. The authors note that they use independent validation set to select the thresholds.

2.6 Learning Speedup

As noted in section 2.3, the computational complexity of the AdaBoost learning algorithm in the detection tasks is $\theta(N \cdot M \cdot T)$ where N represents the number of examples, M represents the number of the features and T is the number of algorithm iterations. Although the computational complexity seams reasonable, the learning time may still reach many ours or even days in cases with high number of samples or many weak hypotheses to choose form. Although this may not be a problem when creating new classifiers for particular practical applications, it may significantly constrain the possibilities of experimenting with new variations of learning algorithms and features.

One way to reduce the learning time, which was proposed by Friedman et. al [38], is to use only a fraction of examples in each iteration which have currently the highest weights. This approach has its justification in the fact that samples with higher weights influence the result more then those with low weights. Therefore it is reasonable to use the available computational power on the samples with higher weights. The suggested size of the fraction of samples used is between 0.99 and 0.9 of the total weight mass. This approach does not reduce the performance of resulting strong classifier much; however, it shifts the distribution over the examples. This may be solved by resampling which eliminates some of the samples with low weights and appropriately raises weights for other samples with originally low weights.

3 Data Transformations in Computer Vision

One of the important tasks in classification is to extract a set of suitable features from the available data. The features should have high classification-related information content compared to the data in its original form to enable the machine learning algorithms to achieve better results. The transformation of the original data, obviously, does not add any additional information, but it can make the relevant information much easier to be found by the learning algorithm. In general case, linear transforms, such as principal component analysis (PCA) or linear discriminant analysis (LDA), can be used to extract relevant features. When additional prior knowledge about the data is available, it should also be utilized in the feature generation. In computer vision, the data usually represents two dimensional discrete signals which exhibit strong spatial relations. Some particular knowledge about the structure of the data – about the spatial/frequency relations – can be utilized e.g. by using linear transforms with suitable fixed basis vectors such as Fourier transform, discrete cosine transform (DCT), or wavelet transforms [39]. More specialized features are used in some specific task such as optical character recognition where the image is usually preprocessed and then some features describing shape are extracted.

When using AdaBoost in computer vision problems, it is possible to use highly over-complete set of features based, for example, on some kind of simple wavelets. If the weak hypotheses are simple (e.g. decision stumps[1]) and each is based on a single feature, then AdaBoost essentially tries to select the most discriminative and, at the same time, compact sub-set of the features. This approach results in higher classification precision and lower number weak hypotheses in the final classifier, then if a classic wavelet transform was used. In the following text, we describe some of the most common data transforms used in computer vision in connection with AdaBoost.

3.1 Haar-like Features

Haar-like features were used in combination with AdadBoost for the first time by Viola and Jones in their face detection system [5]. Since then, many authors continued this work [6] [7] [8] [9] [10] [11] [13] [37]. Also, other applications of these features emerged. For example, in [41] the authors use WaldBoost classifier with Haar-like features as an approximation of Hassian-Laplace detector to detect points of interest in images. The Haar-like features are generally very suitable for detection tasks as they can be computed very fast and in constant time using a structure called integral image

[1] For more information about decision stumps see section 4.2.

[5]. One disadvantage of Haar-like features and all other features based on wavelets is that they need some normalization to achieve intensity scale invariance. Normalization by standard deviation of intensity of the sample is usually used. This normalization, although simple and effective on CPU, can be problematic on other platforms like GPU and FPGA.

Haar-like features are derived from Haar wavelets which were proposed already in 1909 by Alfred Haar [40]. Note that this was even before the term wavelet was established. The Haar wavelets are the simplest possible wavelets. They are basically a localized step function (see figure 4). The Haar-like features extend the wavelets to 2D and some of them are little bit more complex. The most basic Haar-like features are composed of two adjacent, axis-aligned rectangular areas of equal size. The result is then the difference of the average intensity value in the two areas. Also more complex features exist. Some of the Haar-like features which have been used in practical applications are shown in figure 5.

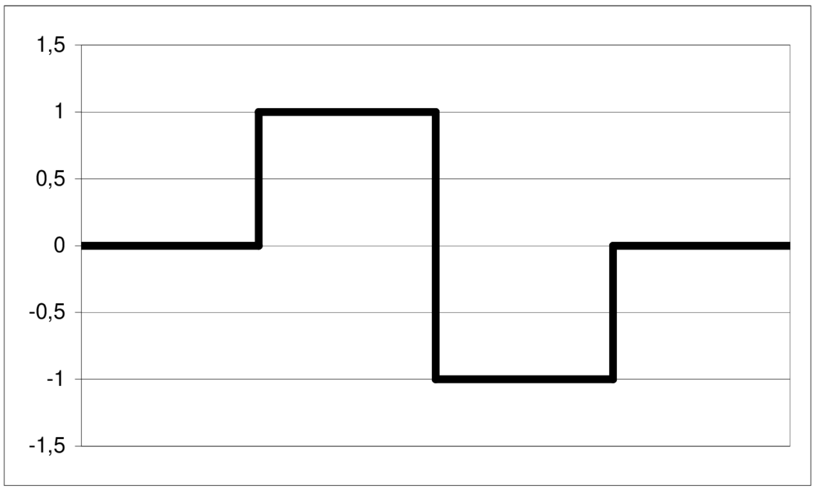


Figure 4: The Haar wavelet.

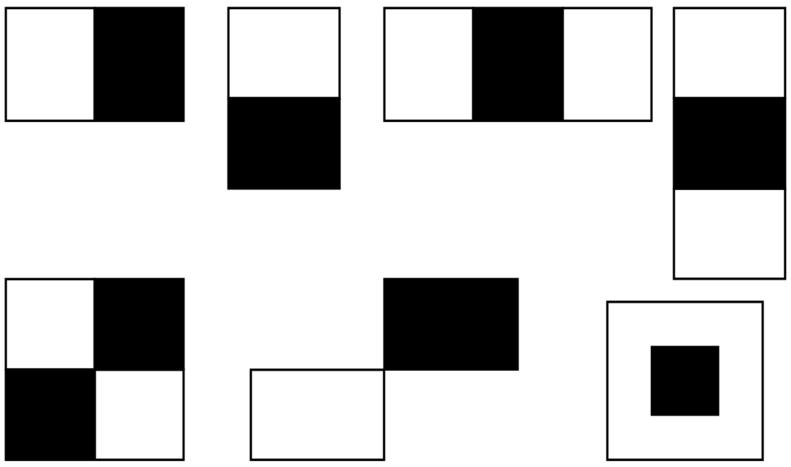


Figure 5: The Haar-like features which were used in practical applications.

The term integral image was first used by Viola and Jones in [5]; however, similar structure called summed area tables was used earlier in computer graphic. The integral image is an intermediate representation which makes it possible to compute sums of values in arbitrary sized axis-aligned rectangular areas in constant time with only four accesses to memory. The integral image at location x, y contains the sum of the pixels above and to the left of x, y, inclusive:

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

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. Also some modifications to the integral image exist. In [6], the authors use extended set of Haar-features which use integral image rotated by 45°.

3.2 Gabor Features

Gabor wavelets [44] are preferred for their higher descriptive power in applications where the computational time in not so critical [42][43]. Gabor wavelets provide ideal trade-off between frequency resolution and spatial resolution. Another interesting motivation for using 2D Gabor wavelets in computer vision is that they are closely related to how the images are processed in the human visual cortex [44]. Gabor function is a Gaussian-modulated complex exponential (see Fig 6). Similarly to the Haar-like features, the Gabor wavelets also need normalization to achieve intensity scale invariance.

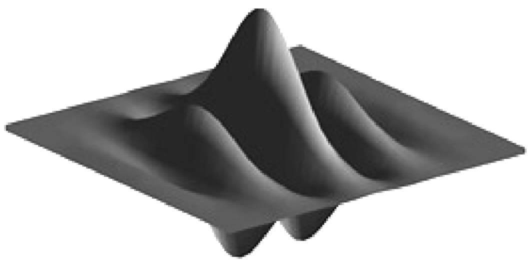


Figure 6: The 2D Gabor wavelet.

3.3 Local Binary Patterns

Local Binary Pattern (LBP) is a texture analysis operator which provides information about local texture structure invariant to monotonic changes in gray-scale and possibly to rotations. LBP creates a binary code by thresholding a small circular neighborhood by the value of its center (see Fig 3). The original definition of LBP [45] was extended to arbitrary circular neighborhoods in [46]. Invariance to rotations can be achieved by merging appropriate code values [47]. Rotation invariance can be further improved by distinguishing only uniform patterns [47] – patterns with at most two transitions between 0 and 1 in the corresponding binary code.

LBP operator was used in many practical applications mostly tightly connected to static texture analysis [45][47][48][49] and dynamic texture analysis [50], but also in face recognition[51][52] and authentication [53], facial expression recognition [54] and palmprint identification [54]. For further information on LBP and examples of successful application see [56].

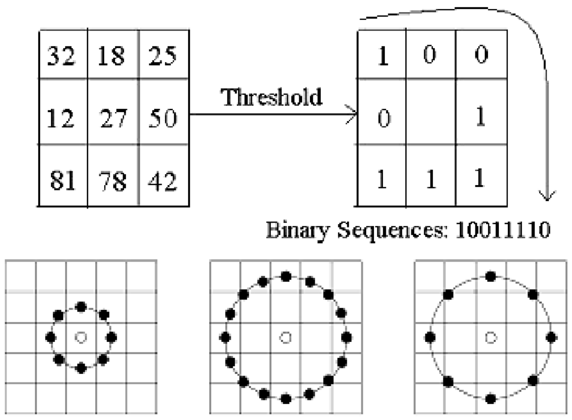


Figure 7: Fig 3: Local binary patterns (LBP) as presented in [54].

3.4 Linear Transforms

Linear transform is a function between two vector spaces that preserves the operations of vector addition and scalar multiplication. When considering finite-dimensional vector spaces, every linear transformation can be represented as a matrix multiplication. When considering the use of linear transforms in classification applications, they can be used to reduce the data dimensionality, they can transform the data to a vector space where the distinct classes can be easily separated and/or they can provide a way how to utilize some knowledge about the data.

Linear transforms have many uses in the field of computer vision and image processing. Probably the most widely used linear transform in image processing is the discrete Fourier transform (DFT), which transforms the original two dimensional image signal into a discrete spectrum of its frequency components. The basis vectors of the discrete Fourier transform are complex exponentials with rising frequency. It may be beneficial to use DFT to transform images before classification, as DFT decorrelates the original data using our knowledge about some inherent structure of the data. For practical classification application, it is more suitable to use the discrete cosine transform (DCT) which provides real-valued results. Except these two transformations, many other exist. Also wavelet transformations are linear.

Many general linear transforms are used to support classification such as principle component analysis (PCA), linear discriminant analysis (LDA) and independent component analysis (IDA), etc. These transforms do not have fixed basis vectors as in the case of DFT and DCT, but the basis vectors are rather estimated from the data based on some objective criteria. In PCA, the bases vectors are computed in such way that the first one reflects the direction of the largest variability in the original data and this variability decreases for further bases vectors. For example, in [10], the authors use a cascade of AdaBoost classifiers where the first stages use Haar-like features and the later stages use features derived from PCA. This approach is beneficial, since the PCA features offer enough discriminative power even in the later stages of the cascade where the Haar-like features are too weak to discriminate the hard examples. On the other hand, the PCA features are too computationally expensive to be used in the first stages of the cascade. This way, the cascade preserves the high classification speed while increasing its accuracy. LDA is related to PCA, but, in this case, the bases vectors represent directions in which samples from different classes can be best separated. The goal of ICA is to find such linear transform of non-gaussian data so that the resulting features are statistically independent, or as independent as possible.

4 Weak Learning Algorithms

Weak learning algorithm in the context of PAC learning framework is any learning algorithm which can achieve at least slightly better results then random guessing on arbitrary distribution over the training samples. Although the weak learning algorithms which can be boosted by AdaBoost are not restricted in any other way, in practice, only very simple weak learners are usually used. The commonly used weak learners include histograms, decision stumps and decision trees. All of these weak learners are members of a group of so-called domain-partitioning weak hypotheses[1] and use only single feature to form their prediction. The domain-partitioning weak hypotheses divide the feature space X into disjoint blocks which cover all of X and the prediction values of the hypotheses depend only on which block a sample falls into.

Some work has been also done to explore the possibilities of using more complex weak learners such as artificial neural networks and SVM. However, these weak learners are not widely used.

There are two main reasons to use simple weak learning algorithms. The first reason, which is most relevant in real-time applications, is the low computational complexity of such algorithms. The simple weak hypotheses are very fast and more complex hypotheses do not usually provide adequate speed-up to justify their computational cost. In this context it is more effective to use some data transformation technique (Haar-like features, PCA, …) then to use more complex classifiers. The second reason is connected with generalization properties of the strong classifier, as it is not fully clear how the boosting algorithms will perform with such complex hypotheses.

4.1 Histograms

Histograms are the simplest weak classifiers. When considering only histograms based on single feature, the partition blocks are formed by equidistant hyper planes which are perpendicular to one of the dimensions of the feature space. In other words, this is equal to dividing the real line of the possible feature values into connected intervals which have equal width. In further text, we will call such partition blocks which are based on single feature bins.

As noted in [57], these weak learners are still used by many authors [58] [59], although they suffer from many drawbacks. The first drawback is that there is no general rule how to set the number of bins. The appropriate number of bins is usually chosen according to experiments. Another drawback is that the bins are, in every, case far from optimal. In areas where the probability distribution functions change rapidly, the equidistant bins are not able to capture the rapid changes.

[1] See section 2.2 for definition of domain-partitioning weak hypotheses.

On the other hand, in stable regions the number of bins is unnecessarily high and reduces the prediction power of the weak hypothesis due to the smoothing coefficient (5).

4.2 Decision Trees and Decision Stumps

These weak hypotheses eliminate the main drawback of the histograms which are the fixed bin boundaries. The decision stumps were historically the first weak hypotheses used with AdaBoost, because they are suitable for discrete AdaBoost as they inherently divide the samples into two bins. The decision stumps have been also used in many successful practical applications [5][6][11]. A decision stump can be viewed as a degenerated decision tree with only the root node and two leaf nodes. As such, the decision stumps contain only single threshold which divides the samples into two bins.

The weak learner's task in the case of decision stumps is to find suitable position for the threshold. The threshold should be set to such position which assures the best classification performance of the resulting hypothesis. In the case of discrete AdaBoost, weighted classification error (1) was used as the criteria to place the threshold. In real AdaBoost, the optimal criteria is the minimization of the Z_t value (4).

The decision trees are basically recursive decision stumps. They perform greedy optimization of some criteria. Again, the optimal criteria which assures the fastest minimization of the bound on the training error is based on the Z_t value. Some authors, however, propose different optimization criteria. For example, in [57] the authors use criteria based on entropy and select the best weak hypothesis based on Kullback-Liebler divergence. In the case of decision, the number of leaf nodes needs to be somehow controlled. There is the possibility to explicitly restrict the depth of the tree or the number of nodes. It is also possible to define some stopping criteria. Such criteria was used in [57]. In real applications, the need to limit the number of leaf nodes does not pose a problem, because even very low number of leaf nodes is sufficient (6-20) to achieve best possible performance and in such case the smoothing coefficient does not significantly weaken the predictions.

As noted in the previous text, the decision trees perform a greedy minimization. The problem of finding optimal bins can be, however, solved more precisely. For example, it is possible to use dynamic programming to find the optimal thresholds.

5 New Image Features

In this section, we present newly developed image features which are suitable for implementation of object detection classifiers in FPGA.

The contemporary state of the art real-time object detection classifiers are modifications of cascade[1] of boosted[2] decision trees or decision stumps[3] based on Haar-like features[4]. Such classifiers benefit from the effective computation of the Haar-like features which takes constant time for all sizes of the features. This allows scanning sub-windows of different sizes without the need to scale the image (the classifier is scaled instead). An intermediate image representation called integral image is used for evaluation of the features. This integral image is also used to efficiently compute standard deviation of pixel values in the classified area, which is used to normalize the features. Another reason why these classifiers are so fast is that the weak hypotheses (histograms, decision trees or decision stumps) are very simple and fast. Finally, the cascaded classifiers make early decisions for most of the background image areas and thus reduce the mean number of weak classifiers which need to evaluate (to 5-20 weak classifiers). All of these facts enable the classifier to scan all the sub-windows needed to reliably find the object even in high-resolution video. To sum it up, the speed of the classifier depends on how fast the weak classifiers are computed and how much discriminative power they offer. The demand for high discriminative power arises from the fact that higher discriminative power of the weak classifiers implies lower average number of evaluated weak hypotheses needed to make reliable classification decision.

The classifiers discussed in the previous text are optimized for general purpose CPUs which can not be used in many applications due to high power consumption, high cost and/or space limitations. Such applications include, for example, digital cameras, camcorders, surveillance, traffic monitoring and mobile robots. The solution for such applications can offer programmable hardware – FPGA. In contrast to the CPUs, FPGAs offer better energy consumption – computational power ratio, but only if the algorithms can suitably parallelized and mapped to the device. The limitations of FPGAs include limited numerical precision, low local memory capacity and limited resources for the algorithm itself.

The classic boosted classifiers based on Haar-features are not much suitable for implementation in FPGA for number of reasons. First, relatively high precision is needed for the integral image (16-18b for features 32x32 pixels large) and the access to the integral image is absolutely random.

[1] See section 2.5 for more information.

[2] See section 2.2 for more information.

[3] See section 4.2 for more information.

[4] See section 3.1 for more information.

Second, on CPU, the features are normalized by the standard deviation of intensity in the classified window and the integral image of squared pixels which is used to compute the standard deviation effectively needs even higher bit precision (cca. 22-24 bits). Further, the computation of the standard deviation involves square root which is itself problematic to implement in FPGA. Moreover, it is not possible to fully take advantage of the fact that the evaluation time of the Haar-like features is independent on the size of the feature. This advantage, which is used in the CPU to effectively scale the classifier, can not be used in FPGA as the precision of the integral image is limited and there is not enough memory on most of the FPGA chips to store significantly large portion of the image. Despite all these properties make it difficult to implement these classifiers in FPGA, some implementations exist [14][15][16]. However, they provide only relatively low performance.

In the following text, we introduce new image features which we have developed for classifiers in FPGA. The Min/Max features, which are presented first, were the first development step in the search for efficient and discriminative features. As such, they do not excel in classification performance, but are still significant as they have led to more powerful features. During their description, we also present the principles which are used in the local rank differences (LRD). The LRD features, which we have presented for the first time in [17], may be the solution for AdaBoost classifiers in FPGA as they can be efficiently evaluated [18] and as they performed similarly to the Haar-like features in our face detection experiments[1].

5.1 Min/Max Features

The Min/Max features basically search for minima or maxima of intensity in some local neighborhood and return the position of the extrema. This could be done in many ways differing in the shape of the neighborhood, how the position of extrema is encoded and how the scalling of the feature is performed. In our work, we consider neighborhoods of rectangular shape and arbitrary size and we smooth the image with rectangular filters when scaling the feature – the position encoding will be explained further.

To make it clear, we will now describe specifically our variation of the Min/Max features which you can see on figure 8. The Min/Max features form a grid over the pixels of the original image. Both the number of cells in the grid and the size of the cells can be arbitrary. Each of the cells inside the grid is assigned a unique index i. The order of the indexes is, in fact, not important. Let V_i be mean value of intensity in cell i and let $\mathbf{V} = \{V_1, V_2, \ldots, V_N\}$ then the result of the feature is simply the index of the cell with minimum or maximum V_i:

$$f_{\max}(\mathbf{V}) = \arg\max_i (V_i); \; f_{\min}(\mathbf{V}) = \arg\max_i (V_i)$$

[1] The results of the experiments are presented in chapter 7.

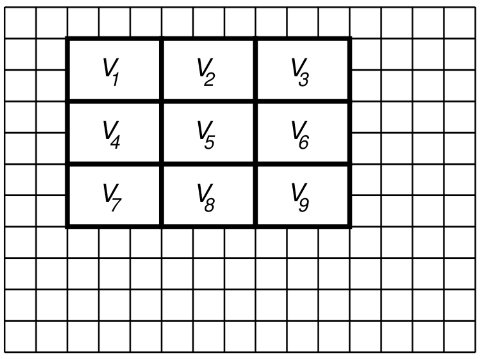


Figure 8: The Min/Max features.

5.2 Local Rank Differences

Similarly to the Min/Max features, the LRD features operate with a set of values $\mathbf{V} = \{V_1, V_2, \ldots, V_N\}$ derived from a local image neighborhood. In our case the values represent mean intensity value inside a corresponding grid cells (see figure 9). But instead of searching for extrema in this set of values, the LRD features sort the values V_i and then assign each of the cells a rank $R_i = Rank(V_i, \mathbf{V})$ according its position in the ordered sequence. The output of the feature is then a difference of ranks R_i of two specifically marked cells i' and i'':

$$LRD(\mathbf{V}, i', i'') = Rank(V_{i'}, \mathbf{V}) - Rank(V_{i''}, \mathbf{V})$$

Each of the LRD features is fully specified by the size of the grid, size of the cells, position in the image and the indexes i' and i'' of the two special cells.

The output of the LRD features is in its meaning much similar to the Haar-like features except the difference of intensity is replaced by the difference of the ranks of the values. The results of the LRD features are, in fact, differences of the intensity of the two areas normalized by equalization of local histogram. The equalization of histogram is generally considered the best possible normalization method for preprocessing of images for machine learning algorithms, and thus it can be intuitively expected that the LRD features will provide good performance. Moreover, the number of all possible LRD features in an image is higher then the number of all possible simple Haar-like features. For example, the number of possible LRD features with the grid consisting of 3x3 cells in an image 24x24 pixels large is 304,704. The number of Haar-like features with only two areas is only 86,400 in the image of the same size. The higher number of the features may be beneficial as it increases the probability that some of the features will perform well on the training distributions generated by AdaBoost (see section 2.2)

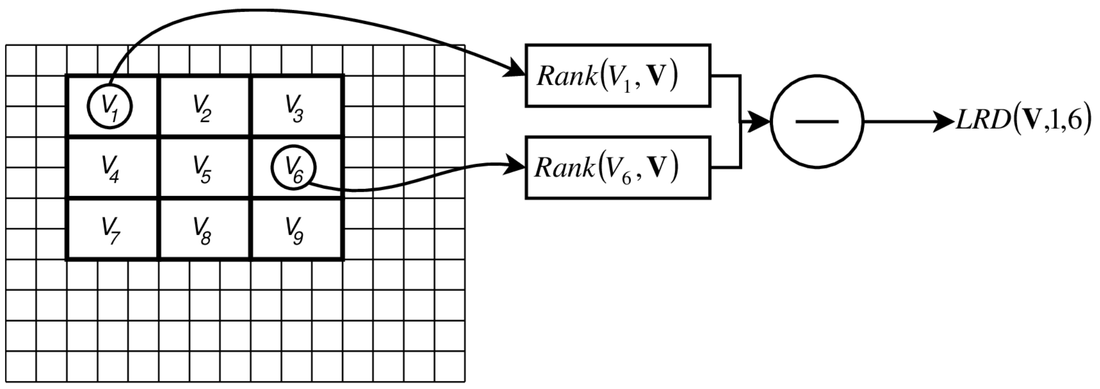


Figure 9: The local rank differences.

5.3 LRD in FPGA

It is obvious that, on general purpose CPU, classifiers with the LRD will not be able achieve classification speeds that would be competitive with classifiers based on the Haar-like features. The LRD are simply too complex compared to the Haar-like features. However, this complexity is compensated by a great potential for parallelization. In fact, an FPGA implementation of a classifier evaluation engine already exist which is able to evaluate two LRD each clock cycle running at 100MHz when synthesized for a small and low cost FPGA Virtex II 250. This gives effective speed of 200,000,000 evaluated features per second. For more information about this classification engine see [18].

Let's now look little bit closer at the concepts which are used at the FPGA. The "pseudocode" implementing the features can be e.g. as follows (figure 10):

```
1) Compute a set of mean values $\mathbf{V}$ based on the grid over the image area
2) Compare $V_{i'}$ to all items in $\mathbf{V}$ - $A = \sum_{v \in \mathbf{V}} [V_{i'} >= v]$
3) Compare $V_{i''}$ to all items in $\mathbf{V}$ - $B = \sum_{v \in \mathbf{V}} [V_{i''} >= v]$
4) Return $A - B$
```

Figure 10: Pseudocode for computation of the LRD in FPGA.

Let's note that the classifiers are not resized in the case of LRD features, the images are rather scaled. Further, the image is not processed at once, but in 128x31 image stripes. Also, the size of the grid is restricted to 3x3 cells in the current implementation. This size of the grid gives 17 possible predictions.

First of all, the mean values V_i are not computed during the evaluation of the feature, but are computed in a preprocessing step for every possible location. In the current engine, this restricts the number of possible distinct sizes of the grid cells to four due to memory limitations[1]. We use, for example, sizes 1x1, 2x2, 2x4 and 4x2. Although, this approach may, at the first glance, seem as a waste of computational power, it is, in fact, done very efficiently and does not slow-down the engine as it is done parallel to the evaluation of the features.

These convolved images are stored in the block rams of the FPGA in such way, that it is possible to read values of a 3x3 grid with single memory access. This is, in fact, the most important property of the engine as the number of accesses to memory per feature limits the performance. For example, the simplest Haar-like features with only two areas need six memory accesses when using the integral image. This effectively makes even these simplest Haar-like features at least six times slower then the LRD. The comparisons are done in parallel by block of comparators. A simplified diagram of the evaluation engine can be seen on figure 11. As can be seen on the figure, the positive results of the comparators are summed and the two values are then subtracted. The result of this subtraction is used to directly address appropriate prediction value in the table of predictions. The predictions are accumulated to get the final classification result. The engine is also able to test after evaluation of each feature if the current accumulated sum of the weak predictions is lower then some threshold and if the result of the test is positive the classification of the current window is terminated with negative result. The threshold for this early termination can be set by the WaldBoost algorithm[2].

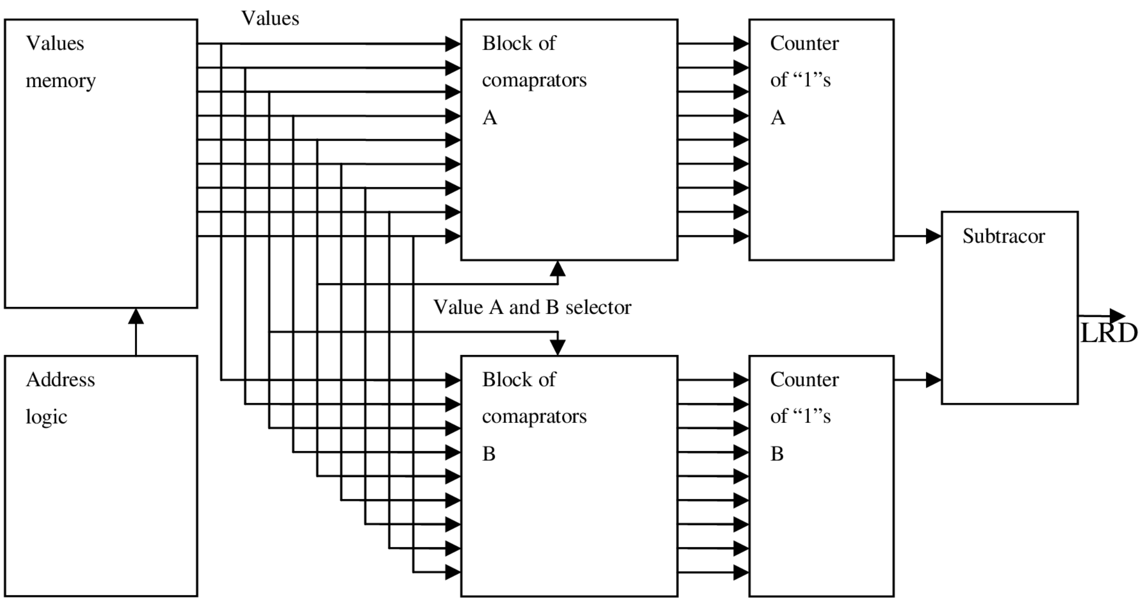


Figure 11: A simplified diagram of the FPGA engine for evaluation of the LRD as presented in [17].

[1] The experiments do not show any significant drop in discriminative power due to the restriction of the sizes of the grid – see section 7.3 for the results of the relevant experiments.

[2] For detail on the WaldBoost algorithm see section 2.5.2.

The numeric precision of the prediction values is restricted only to eight bits. However, this restriction does not lower the classification power of the features. Also, the V_i values are rounded to eight bits, but this does not pose any problem as information is lost only for larger sizes of the cells, and even there the lost bits should be mostly noise.

6 Experimental Boosting Framework

The development of efficient classifiers is to a great extent an experimental science. All new ideas have to be tested on representative testing sets and/or some of the parameters have to be tuned for specific applications. Although the boosting algorithms themselves are mostly relatively simple, things get complicated with the weak learners, the data transformations and a need for computational efficiency. It becomes even more complex when learning object detectors for images. In such case, it is necessary to manage the huge number of background samples which is needed to train classifiers with the early decision termination[1]. When considering all of these necessary parts, it is difficult and costly to create an application for experimenting with boosting classifiers in computer vision tasks from the scratch. On the other hand, all of the mentioned parts are relatively independent and therefore it should be possible to create a framework where it would be possible to add any new boosting algorithms, features, etc. To our knowledge, such framework does not exit or it is not publicly available. On account of these facts, we have decided to develop a boosting framework for computer vision applications. This boosting framework is intended for our own experiments focusing mostly on the development of new features, as well as, a contribution to the global research efforts.

In this chapter, we present the experimental boosting framework which we have developed. First, we summarize all of the requirements which are relevant for the framework. Next, we explain the basic structure of the framework and basic ideas which led to this structure. In further text, we explain all parts of the framework in more detail. We also discuss the possibility of parallelization. At the end of this chapter, we describe some of the interesting approaches we have taken during the realization of the framework and we also present some of the supporting tools which we have developed to manage image datasets and to process the results obtained during the experiments.

6.1 Requirements

The experimental boosting framework has to satisfy three main requirements to be generally usable. First, the framework should be feely available without any additional expenses. Second, the framework should be as flexible as possible. And last but not least, the framework should provide high performance so the time and resources needed for experiments are reduced to minimum. From this thee basic requirements, we have derived following set of more concrete simple requirements:

[1] See section 2.5

Public availability without additional expenses:

- The framework should be distributed under such license which allows to use the framework for non-commercial and research purposes.
- All external libraries used by the framework should be also free of charge.
- All adopted source codes, if any, should be distributed under the GNU General Public License or similar one.
- No commercial support tools should be needed.
- The framework should be platform-independent – at least Windows, Linux and Unix should be supported.

Flexibility:

- It should be possible to add any new learning algorithm which observes the boosting paradigm.
- It should be possible to add data sources.
- It should be possible to add new image features or other data transformations.
- It should be possible to add new weak classifiers.
- It should be possible to add new types of performance evaluations and printed outputs.
- All of the parts of the framework should be easily configurable.

High performance learning:

- The samples should be represented in such way to allow efficient computation of the features.
- It should be possible to efficiently exchange the samples in the training datasets during training.
- The framework should try to minimize computational burden arising from frequent calls of simple functions and from data transfers.

Additionally, we have formulated few requirements which arise specifically from our area of research. We needed that it was possible to experiment with specific properties of the classifiers in FPGA. This includes mostly quantization of some values. The specific requirements are:

- Possibility to limit the precision of the weak prediction values.
- Possibility to limit the precision of computations during evaluation of the features.

As the framework is primarily intended to be used in the area of computer vision, it can not work with precomputed feature vectors. The feature vectors in computer vision classification tasks are often so long[1] that even moderate number of feature vectors does not fit into memory of

[1] For example the length of feature vector derived form image 24x24 pixels large could be 180,000 in the case of Haar-likefeatures or 300,000 in the case of LRD.

contemporary computers. Due to this fact, the feature vectors have to be computed during the learning from the original data. The possibility to transform the data during the learning could be also beneficial because it may enable to estimate some transformations like PCA directly on current distribution over the learning data samples which is generated by AdaBoost.

6.2 Basic Principles

During the design of the framework, we considered the most important requirement to be the need for flexibility and extensibility. We wanted to reduce the number of changes needed when adding any new code to the framework. Quite naturally, we decided to use object oriented design to reduce the dependencies between distinct parts of the code and polymorphism to achieve almost plug and play extensibility.

The requirements for flexibility and extensibility are deeply connected with the requirement for transparent configurability, as it is needed that the new parts of code manage their configuration themselves without the need to communicate with other parts of the code. We have solved this requirement by using XML for configuration. We use XML in such way, that constructor of each object in the framework takes an XML node as a parameter and the constructor itself decides if it should process the XML node. If the XML node is recognized, the object processes the configuration information contained in the XML node and optionally passes some of the child nodes to globally defined functions which distribute the child nodes to other constructors. This solution is possible due to the fact, that the objects in the boosting framework form a tree structure.

Let’s now illustrate this concept on a practical example. Consider a boosting algorithm which uses some weak learners. In our framework, the boosting algorithm as well as the weak learners is represented each by XML nodes in the configuration file. Moreover, the nodes representing the weak learners are children of the node representing the boosting algorithm. During the initialization of the framework, constructor of the boosting algorithm receives the relevant XML node. It recognizes this node, processes the configuration information and passes all child nodes to a global function which is responsible for initialization of the weak learners. This global function passes the child nodes to constructors of all weak learners in the framework and if any of the nodes are recognized then the function returns the successfully created instances of the weak learners. The instance of the boosting algorithm receives the weak learners and inserts them into its pool of weak learners.

The result of this approach is that it is really simple to add new features to the framework. It is only needed to create a class which implements corresponding well-defined interface and add a call of its constructor to a global function which initializes this kind of objects.

The use of XML goes even further then the configuration of the framework. Each object in the framework is able to store its content in the form of XML. This is used to export the resulting classifiers, but also, for example, to copy the objects and other manipulation with the objects. This ability to store itself into XML could be also used to send the objects between different computational nodes during eventual parallel execution[1].

6.3 Framework Structure

As described earlier, the framework is object oriented and uses polymorphism. The interfaces which form the framework mirror the basic objects from boosting learning in computer vision tasks. There are also some additional interfaces and classes which provide access to data sets.

We will first introduce the part of the framework concerning the data sets. The interfaces and classes which provide access to the datasets are shown in the figure 12. This part of the framework consists of three levels of classes. The top-level class unifies access to different subsets of samples in the dataset. These subsets can represent samples from different classes, samples obtained from different sources, etc. The subsets unify access to various physical data sources which are here called repositories. The repositories provide access directly to the physical data and provide a way how to add new sources of data in the future. At the present state, all of the repositories access data in form of feature vectors or images. In the future, it is possible to add, for example, generators of artificial datasets. For more information about this part of the framework and how the samples are loaded see section 6.4.

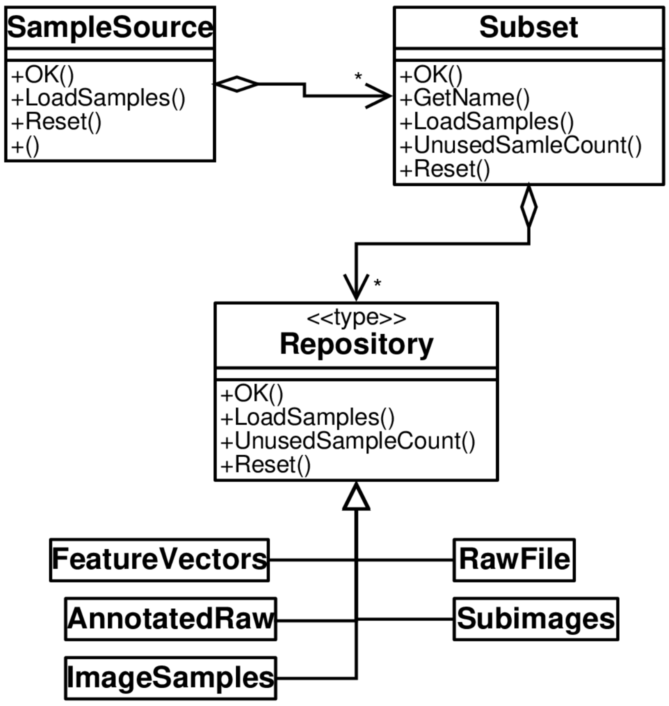


Figure 12: The part of the framework which is responsible for accessing the datasets.

[1] For discussion about possibilities for parallel execution see section 6.8.

The structure of classes which perform learning is shown in figure 13. If we consider only simple classifiers without early termination of evaluation, the learning is performed by an object implementing the BasicBoosting interface. Such object receives training set of samples and returns learned strong classifier. In each iteration of the boosting algorithm, the object passes the training examples to the weak learners together with current distribution over the examples. The weak learners may use some data transformations provided by an object implementing the interface Features. There are two types of the transformations. The results of the first type of features are real numbers – for example the Haar-like features, Gabor features, linear combinations, etc. The results of the second type of features are integer values – for example the LRD, LBP and Min/Max features. Each of the weak learners produces the best weak hypothesis possible on the current distribution. The boosting algorithm then selects from the weak hypotheses that which performs the best and adds it into the strong classifier (class LinearTwoClassClassifiier). The prediction values can be set directly by some of the weak learners or are adjusted by the boosting algorithm.

The samples for the boosting algorithm can be provided by some simple top level routine or they can be obtained form a cascade learning algorithm (implementing interface CascadeLearning). The cascade algorithm loads the training samples and calls the boosting algorithm. The cascade algorithm then receives a classifier and adds it to the cascade of classifiers (interface CascadeCalssifier) which is then used to prune the training set. After the training set is pruned and replenished to desired size, the process repeats again.

The framework consists of even more classes and interfaces which are, however, not so important for the learning part. To mention some, an interface for evaluation of the classifiers and output of the results is defined.

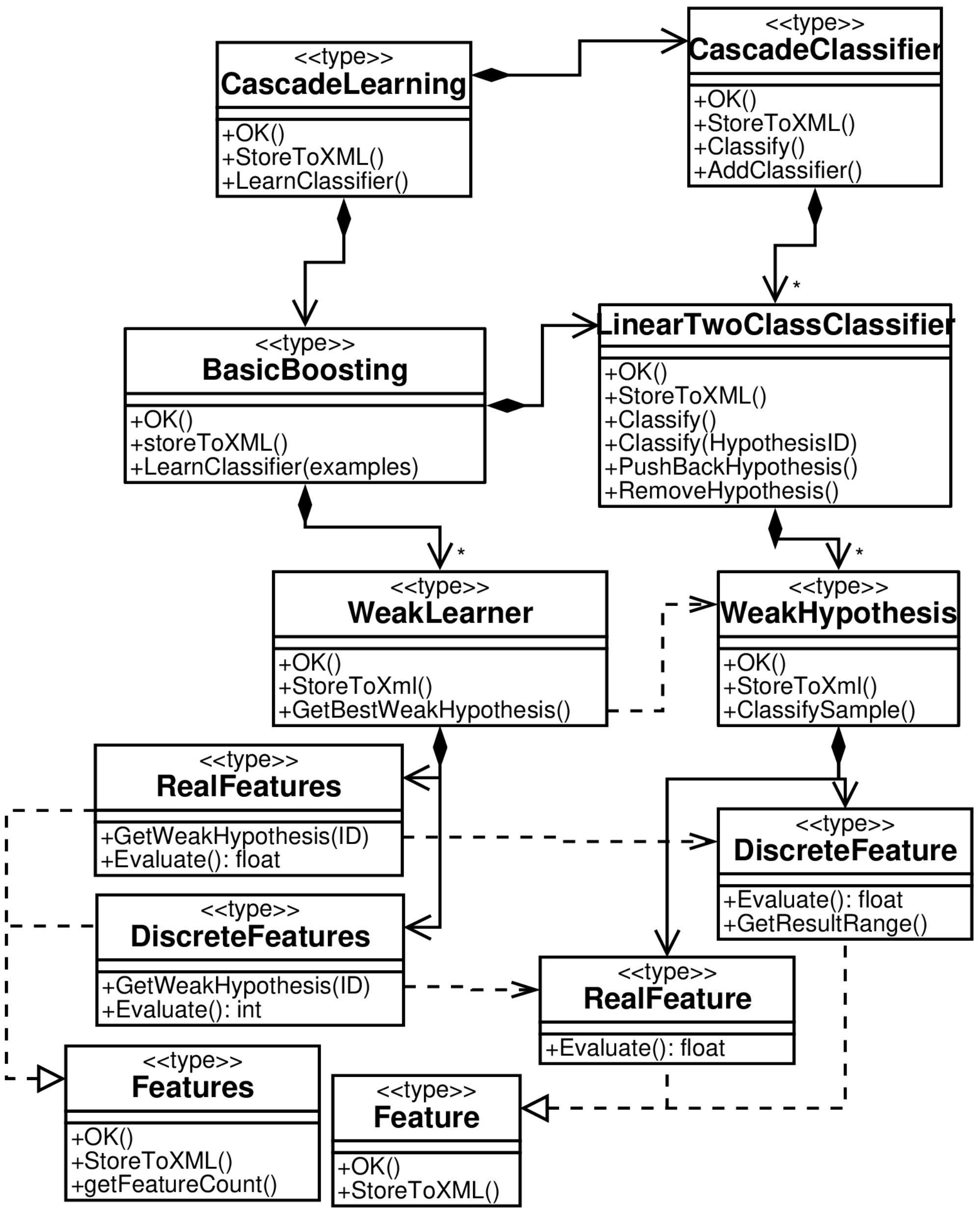


Figure 13: The classes and interfaces forming the learning part of the framework.

6.4 Data Source

The structure of classes responsible for loading samples is presented on figure 12. Each dataset is defined by single XML file. This file defines the logical structure of the dataset and contains references to the physical data which is stored in separate data files. The dataset consists of subsets of samples. These subsets are not identical with classes assigned to the samples during learning, but samples from more subsets can be combined into single class. This can be used, for example, to merge training and validation sets for some experiments only by slight modification of the configuration file and without any change in the dataset file. Each of the subsets has a unique textual identifier and may contain multiple physical sample sources which are called repositories in the framework. These repositories abstract access to various sources of samples like image files, raw data files, text files with feature vectors, annotated images and annotated raw data files, generators of artificial data, etc.

For experiments in classification, it is often needed to load the samples in random order. Further, some experiments require that a single dataset is repeatedly randomly divided into disjoint training, testing or validation sets. To make such experiments more efficient and the preparation of the experiments easier, these possibilities are integrated directly into the framework. When loading samples in random order from multiple subsets, the number of samples is distributed between the relevant subsets proportionally to the number of unused samples in each of the subsets. The number of loaded samples is then further distributed between the repositories again according to the number of unused samples. It is also possible to load the samples in linear order. The boosting algorithms can be stopped in any iteration, while the learned strong classifier effectively holds all information about the previous rounds of the algorithm. This implies that it is possible to stop boosting at any time and later continue learning from that point. To continue correctly, it is necessary to load exactly the same training samples as in the suspended learning. In the framework, this is assured by custom pseudo-random number generator which generates the same sequences of numbers regardless which compiler and operating system is used. The seed number for this generator is stored together with exported classifiers.

The possibility to divide the data randomly into disjoint training, testing and possibly validation sets is tightly connected with the possibility to repeatedly load samples from a single subset. In such case, the previously loaded samples must be marked and not available next time. This in combination with the random loading of samples allows assigning random samples into disjoint training and testing set.

The samples are not loaded separately, but preferably multiple samples are loaded at once. This increases efficiency and makes it possible to use simple and efficient algorithm to distribute the loading of the samples between subsets and repositories. When loading samples in random order, random access to files can significantly slow-down the loading process. A way to eliminate this would be to load all data into memory in advance, but this could be memory consuming and inefficient. In the framework, only the content of those files which have been accessed is cached in memory. It is possible to explicitly clear this cache as well as the lists of used samples. This caching is provided by the repositories on local level as the repositories can use their knowledge about the data to store it and access it in the most efficient way.

In the section describing the requirements, we noted that the samples should be represented in such way to allow efficient computation of the features. According to this, each sample may contain the original data, integral image, the result of Fourier transform and, possibly, other representations of the original data. Each sample also contains information about its class and weight currently assigned by the boosting algorithm.

6.5 Features

There are two fundamental types of features in the framework which differ in the type of their results. The results of the first type of features are real numbers and the results of the second type of features are integer values. Moreover, each of these groups has two interfaces to be implemented. The first interface is designed to be used for classes implementing the features for the training part of the framework. The instances of these classes hold multiple features of the same kind. This enables to compute multiple features with only single method call which reduces the overhead caused by the method call. More importantly, this allows using values computed for one feature to compute the result of other features. This reduces the learning time by 85 % in the case of LPB[1] and in some cases even more. For example, it is possible to compute responses of Gabor features on all positions of a sample by simple multiplication of the image and the Gabor wavelet in the frequency domain and then use IFFT to get the results. Such approach can speed up the learning with Gabor features significantly especially for larger sizes of samples. Within the scope of the objects, single features are addressed with unique integer index.

The second interface is designed to be used by classes whose instances are part of the strong classifier. These objects hold only single feature. Of course, it is possible that both interfaces are implemented by single class, but it is usually more suitable to have two separate classes.

Except the actual features and data transformation, there are two classes implementing the interface Feature and Features which only manipulate the results of other features. First of these is a class which converts the real results of features into integer values and second is a class which manipulates the integer results – it can be used to reduce the number of possible outputs of a feature.

6.6 Weak Learners and Hypotheses

The weak learner's task is to learn a weak hypothesis which is successful on the current distribution over the training samples provided by the boosting algorithm. Although, any classification algorithm can be used as a weak learner, the framework currently includes only simple weak learners which are

[1] See section 6.9 for more information about efficient computation of the LPB and the Haar-like features during learning.

frequently used in computer vision tasks[1]. These weak learners produce weak hypotheses which fit the definition of domain partitioning weak hypotheses[2]. The class representing weak hypotheses produced by these weak learners contains public variable which holds the prediction values for each block of the partition. Although, the weak learners could choose the prediction values themselves, the prediction values are rather computed by the boosting algorithm. This solution was chosen for the reason that different boosting algorithms can choose the prediction values in different ways or they can adjust the predictions in further iterations of boosting[3].

6.7 The Boosting Algorithms and Cascade

Most learning algorithms which fit the boosting paradigm can be integrated into the framework. Such classes must implement the **BasicBoosting** interface. The boosting algorithms take as an input set of examples and, optionally, validation set. There is a simple routine in the framework which takes care of loading the data sets and which also can evaluate the performance of resulting classifiers and print the results.

Classifiers with early termination of evaluation[4] take care of loading samples themselves. They also implement different interface then the basic boosting algorithms – CascadeLearning. The name can be little misleading as these classifiers with early termination of evaluation do not need to be really some form of classifier cascade. For example, the WaldBoost algorithm also implements this interface. In the framework, there is a simple routine which can evaluate the performance of these classifiers.

6.8 Potential for Parallelization

The boosting algorithms are, in fact, very suitable for parallelization as they can be effectively partitioned. The natural way to parallelize the algorithm is to distribute the estimation of weak hypotheses on more computational nodes. This choice is quite natural as the learning of weak hypotheses is significantly the most time consuming part of leaning. When using such parallelization of the learning algorithm, no information needs to be transferred between the computational nodes

[1] The decision trees, decision stumps and histograms are currently part of the framework. For more information about these weak learners see chapter 4.

[2] For definition of domain partitioning weak hypotheses see section 2.2.

[3] For example, the TCAcu adjusts the prediction values in each iteration of the boosting algorithm. For more information about this algorithm see section 2.4.1.

[4] Some of the classifiers with early decision termination are cascade of boosted classifiers, boosting chain and WaldBoost algorithm. More information about these algorithms can be found in section 2.5.

except the information about the best weak hypothesis. The computation of the new distribution D_t can be performed separately on each of the nodes. This way the speedup factor can grow almost linearly with the number of computational nodes even for relatively high number of the nodes[1].

Our framework, although it was not one of the initial goals, provides simple and at the same time effective support for this parallel learning of weak hypotheses. If more weak learners are defined in the configuration file, the boosting algorithm can send each of the weak learners to different computational node and then receive the best weak hypothesis from each of the nodes and choose the overall best hypothesis. This hypothesis can be then sent to the nodes which compute the new weights. This kind of communication between the nodes does not require any changes in the weak hypotheses and learners as these are already able to store themselves in XML. This representation can be easily sent over the computer network and then used to create corresponding objects at the receiver node. The training samples can be also transferred via network or the computational nodes cam load the samples themselves according to XML configuration received from the master node.

Although, this approach is very simple, it is efficient only under certain conditions. This kind of distribution of work load between the nodes can be effective only if the granularity of weak learners is adequate. If there is low number of weak learners then it is not possible to balance the work load between the nodes. On the other hand, if there are too many weak learners, the communication needed to transfer the weak hypotheses and possibly weak learners significantly slows-down the learning process. Moreover, the fragmentation of weak learners has to be defined in the configuration file which may be annoying for the users.

The mentioned problem with granularity can be, however, easily solved by slight modification of the weak learners. As the weak learners which are currently used in the framework construct each of the weak hypotheses on top of a single feature and then only chose the best, it is possible to add a new method which restricts the weak learner only to certain features. The boosting algorithm could then assign the computational nodes not only the weak learners, but also the portion of features which they should consider during learning.

6.9 Effective Computation of Features

As the feature vectors used in computer vision classification tasks are usually very long[2], it is not possible to precompute the training feature vectors in advance and store them in RAM. The feature vectors must be instead computed on the fly during learning. Even thou some of the features can be

[1] This depends on the time needed to estimate the weak hypotheses which depends on the number of samples and weak hypotheses. If this time is low, the speedup factor may drop as a result of delays during synchronization of the computational nodes.

[2] See chapter 3 for more information about the data transforms used in computer vision.

computed very efficiently, their computation still consumes most of the learning time[1]. That is why any attempts to reduce the learning time should focus on effective computation of the features. The techniques which are discussed in the following text provide significant speed-up (for example, approximately 400% the case of Haar-like features).

Generally, it is a good idea to reduce access to RAM. To achieve this, we can improve data locality during learning and take advantage of fast cache or we can even try to reduce the amount of data which needs to be loaded from memory by computing some of the data on the fly. The data which needs to be loaded from memory is represented by the samples and some information about the features. There are usually also some temporary variables used by the weak learner. For example the histogram weak learners use an array of accumulators to accumulate the weights of samples. In such case, the best data locality is achieved when multiple features are computed for single sample and after the weak learner accumulates the weight of the sample features are evaluated for the next sample. The number of features must be reasonably low so all the data needed by the features and the accumulators used by the weak learner fit into the cache. This way, cache misses happen only when accessing a sample for the first time. In our framework we use this technique with good results. For example, in the case of Haar-like features we compute the positions of the features on the fly which in the combination with the improved data locality provides approximately 400% speed-up compared to the original solution where the positions of the features were stored in RAM. The performance during learning is now 37M Haar-like features computed per second on Intel Core 2 CPU running at 1.66GHz.

Other features can benefit form the fact that multiple features are computed from a single sample at the same time by sharing some of the computations. For example, in the case of LRD[2] features, we compute the sums of intensity in the cells of the grid and then compute the ranks of each cell. With these ranks, we can compute rank differences of all possible combinations of the cells. There are 36 cell combinations for grid size 3x3. The fact that the 36 features share the computation of the ranks provides significant additional speed-up. In fact the LRD achieve almost the same performance during learning like the Haar-like features, even thou, they are mum more complex.

[1] The computation of features usually consumes between 80 % and 98 % of the total learning time for basic boosting algorithms.

[2] For detailed information about the LRD see section 5.2.

6.10 Support Tools

During the development of the framework, many support tools were created[1] which either manage datasets or allow more effective work with the framework. One of these tools is an image annotation tool which can be used to define regions of interest in images. Next, there is a dataset generator which is able to process these annotations and create datasets compatible with the framework. The dataset generator can add random geometric transformations and noise to the data. There is also a tool which is able to read a dataset and convert it back to annotated image. Finally, we have created a tool which can be used to automatically generate configuration files for experiments. All these tools are also available with the framework.

[1] These tools were created by various authors. In particular, the image annotation tool as well as part of the dataset generator was not created by the author of this thesis.

7 Results

We have used the boosting framework to perform many experiments focusing mainly on performance of the new LRD features compared to the classic Haar-like features. Further experiments studied how the LRD features perform when implemented in FPGA where the sizes of LRD features as well as the computational precision are limited. Some experiments were also concluded to estimate influence of some parameters of the learning algorithms on performance of the resulting classifiers.

We considered a face detection problem as a bases for all the experiments. We chose this problem, because the boosted classifiers are mainly used for face detection in the area of computer vision. There are, however, some differences between the face detection classifiers and those created in our experiments. Although, we used datasets which are suitable for face detection, we did not use classifiers with early termination of evaluation and we did not try to achieve asymmetric classification errors with extremely low false positive rate, which would have been necessary if we had intended to usee the classifiers in a real detection application. This was not necessary in the experiments as we only needed to mutually compare different versions of the classifiers at this stage of the research. We used the real AdaBoost algorithm in all of the experiments.

For the experiments all samples were resized to 24x24 pixels. A dataset of 5,000 hand-annotated face images was divided into training and testing set of equal sizes. This face dataset was originally used as training and validation sets for face detection system described in [37]. The samples were mirrored to effectively double their number. The test set was additionally supplemented with 470[1] faces from MIT+CMU dataset and 1,200 faces from annotated group photos. The non-face samples were randomly selected from a pool of sub-windows from a large set of non-face images. If not stated otherwise, 10,000 non-face samples were used for learning and the performance of the classifiers was estimated on 300,000 non-face samples in the experiments. As this testing set is highly unbalanced (1:45), we report error rates which are recomputed as if the testing set was balanced:

$$\acute{E} = \frac{FP + FN}{2}$$

where FP is the false positive rate and FN is the false negative rate.

Except the classification error on independent dataset set, we also report how the classifiers are able to reduce the Z normalization factor[2]. Although the classification error gives information about the generalization properties of the classifier and about its real-world performance, the Z value carries information about how well examples from individual classes are separated by the classifier.

[1] Only images of real human faces were used from the MIT+CMU dataset.

[2] This is the normalization factor used in the reweighing formula in the AdaBoost algorithm. For more information see section 2.1.

To achieve good separation of the training data is important for classifiers with en early termination of decision, where faster separation of the training data implies faster average decision. Thus, the speed of minimization of the Z value can be used to predict properties of a classifier with early termination of decision learned with the same parameters.

7.1 Comparison of Different Types of Features

In this experiment, we compared performance of five classifiers with different sets of features. First classifier uses only the simplest Haar-like features with two areas. These simplest Haar-like features are supplemented with the features with three areas for the second classifier. Another two classifiers use LRD. The first one uses all possible LRD with grid size 3x3 and the second uses only restricted set of the LRD with possible sizes of cells in the grid 1x1, 2x2, 2x4 and 4x2. The last classifier is based on LBP. Decision tree with eight leaf nodes was used as a weak learner for the Haar-like features as well as for the LRD. Since the LBP provide only eight possible output values, histogram weak learner was used in this case.

The speed of Z minimization of the classifiers is shown in figure 14. The full set of Haar-like features provides the best performance in terms of Z minimization and the restricted set of LRDs is comparable to the reduced set of Haar-like features. The full set of LRD provides only slight improvement over the reduced set. The LBP do not seem to be suitable for the face detection task as they provide the worst results. The reason of this bad performance is probably the fact that the LBP discard large portion of the available information.

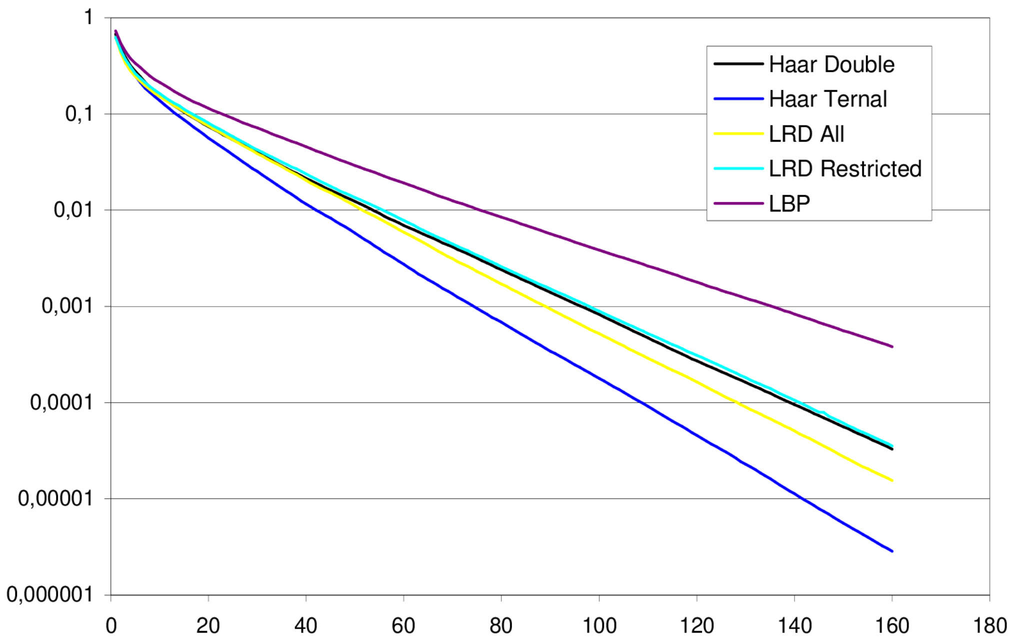


Figure 14: The speed of Z normalization factor minimization. The x-axis represents the number of weak hypotheses and the y-axis represents the value of Z.

The classification error of the classifiers is shown in figure 15. Both LRD and LBP seem to generalize better compared to the Haar-like features considering their speed of Z minimization. Here the LRD slightly outperform the full set of Haar-like features and the LBP outperform the reduced set of Haar-like features. This phenomenon is quite unexpected and requires further analysis.

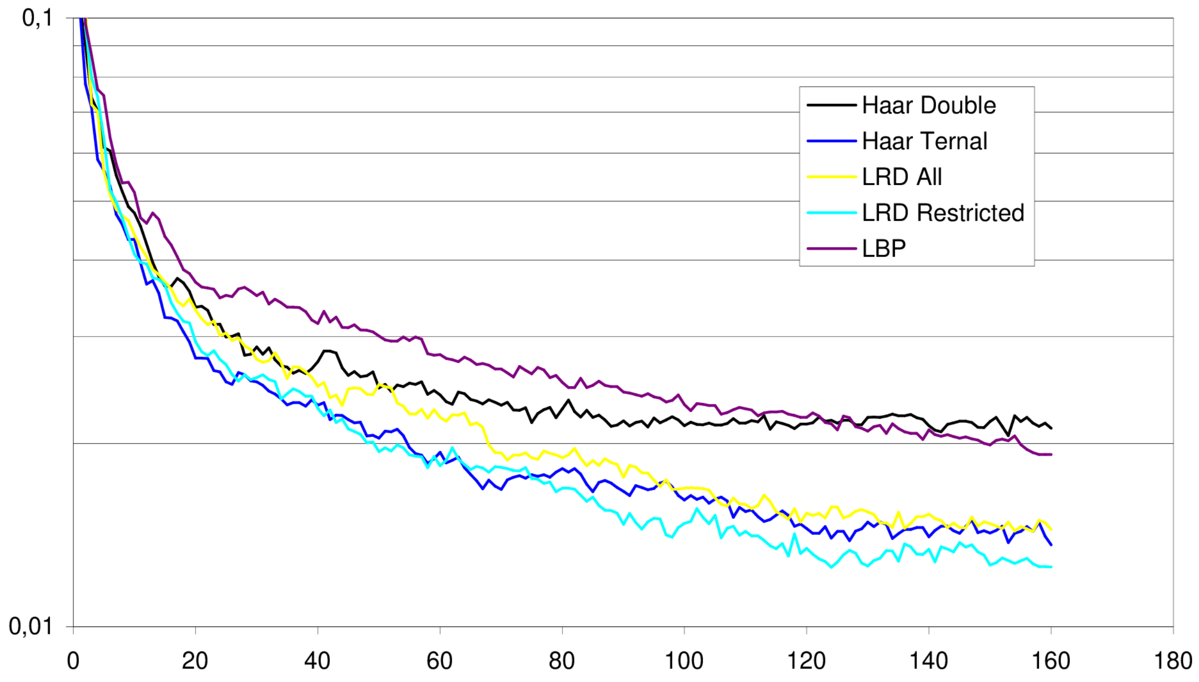


Figure 15: The classification error for different numbers of weak hypotheses. The x-axis represents the number of weak hypotheses and the y-axis represents the classification error.

7.2 Threshold Estimation Precisions

The decision stumps and decision trees weak learners could estimate thresholds directly on the samples. This would provide best precision of the estimated positions of threshold, but it would be also computationally very inefficient as the samples need to bee sorted according to the result of the feature before the estimation. Another possibility to estimate the thresholds is to discretize the results of the features and estimate the thresholds on a histogram. This approach is computationally efficient, but could reduce the performance of the resulting classifiers. The results of this experiment should answer the question how the discretization influences performance of the weak learner and how fine the disretization should be to prevent degradation of the weak learner. Haar-like features and decision tree weak learner were used in the experiment and the number of leaf nodes was set to eight.

The results, which are presented on figure 16, show that there is no measurable improvement in classification error of the resulting classifiers when there are more then 32 bins in the histogram. However, the speed of Z minimization still slightly increases even with higher number of bins. According to these results it should be safe to use quantization with 64 or more bins.

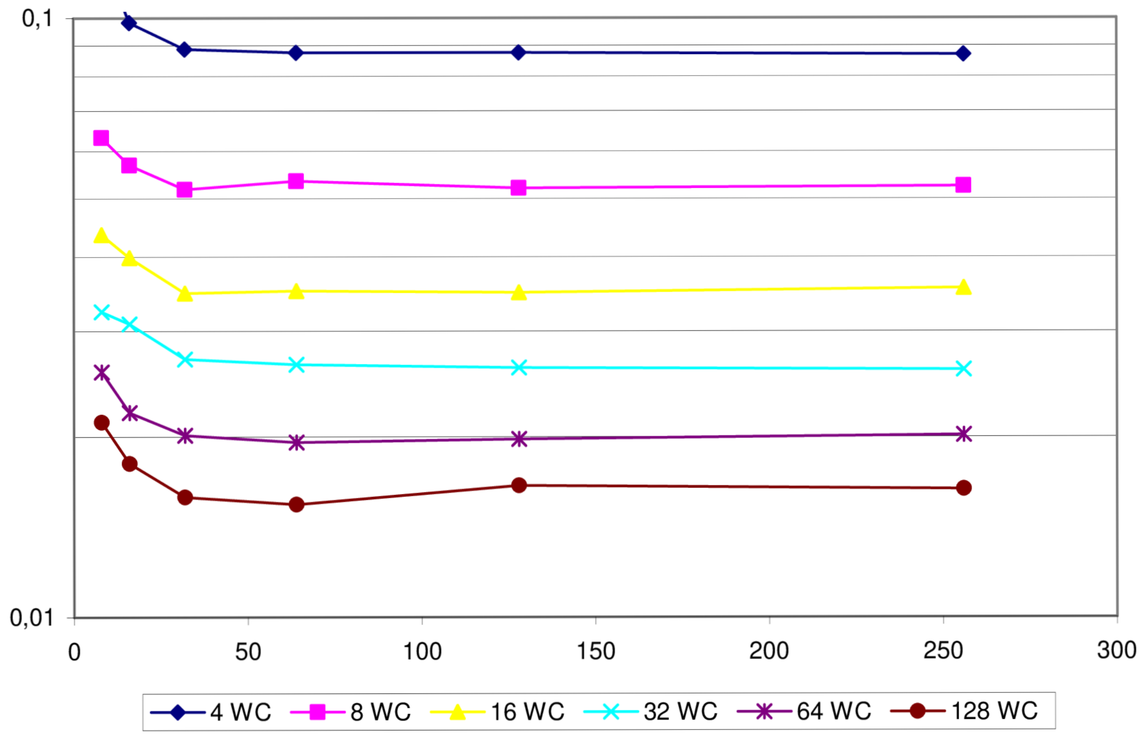


Figure 16: Relation between classification error and precision of threshold estimation for the decision tree weak learner. Each of the lines represents single length of the boosted classifier and the x-axis represents the number of bins in the histogram.

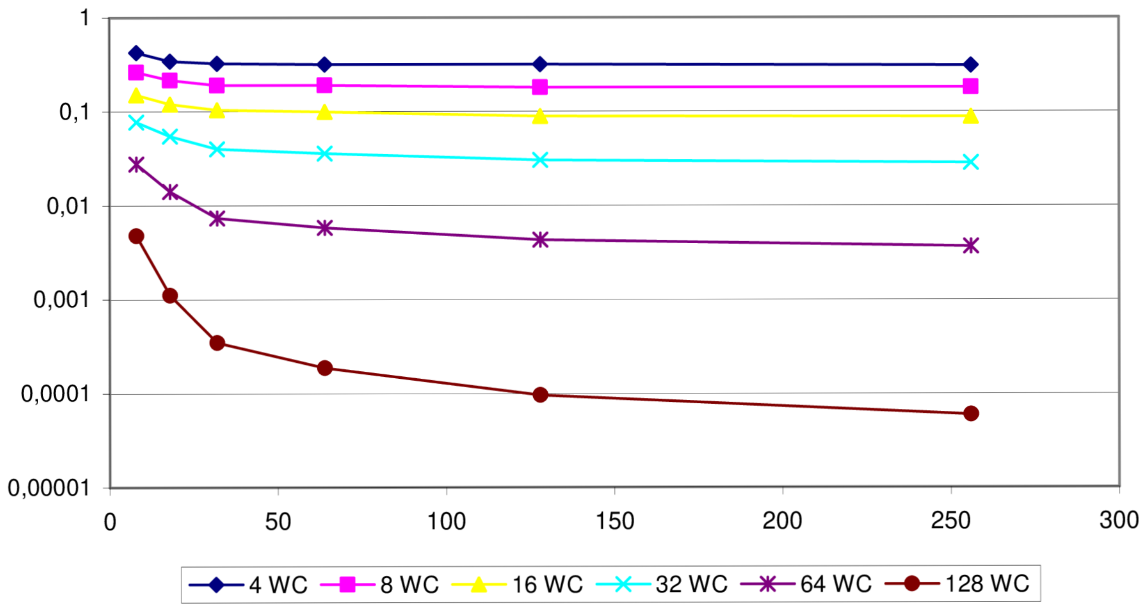
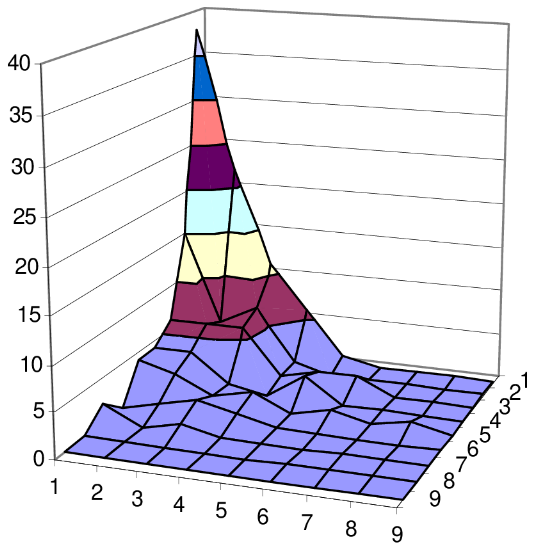


Figure 17: Relation between speed of Z minimization and precision of threshold estimation for the decision tree weak learner. Each of the lines represents single length of the boosted classifier and the x-axis represents the number of bins in the histogram.

7.3 Restricting the Sizes of LRD

When using the LRD in FPGA, it is necessary to limit the number of distinct sizes of the grid cells. In the current FPGA evaluation engine [18], only four sizes of grid cells can be used at the same time. The important question is what combination of sizes is the best for certain application.

The figure 18 shows histograms of grid cell sizes of features in a face detection classifier. It is clearly visible that majority of the features has small sizes of cells – more then 80 % of the features have both sizes of the cells smaller then 4. This fact suggests that it should be possible to use only the small sizes without a need to increase the number of features in the classifiers.



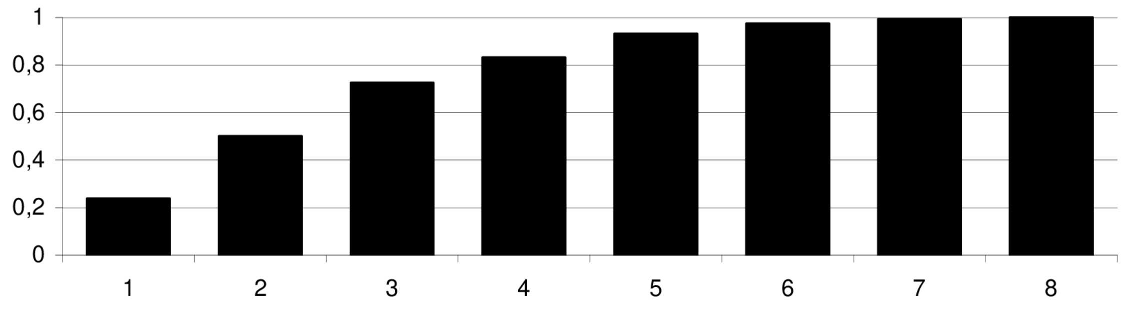


Figure 18: The upper chart shows 2D histogram of the cell sizes of a face detection classifier, where both horizontal and vertical size of the cells is considered. The lower chart shows portions of features in a face detection classifier which have grid cells with horizontal and vertical sizes smaller then certain value. The x-axis represents the maximum size of the grid cell.

To answer the question, what is the best combination of sizes for face detection, we have compared several classifiers with different combinations of the cell sizes. The results are shown on figures 19 and 20. According to this data, there is no clearly best combination of sizes. The differences in classification performance are only slight and rather random. Also the speed of Z minimization is not conclusive. According to the results, there is no significant degradation of classification performance when the sizes of grid cells are restricted.

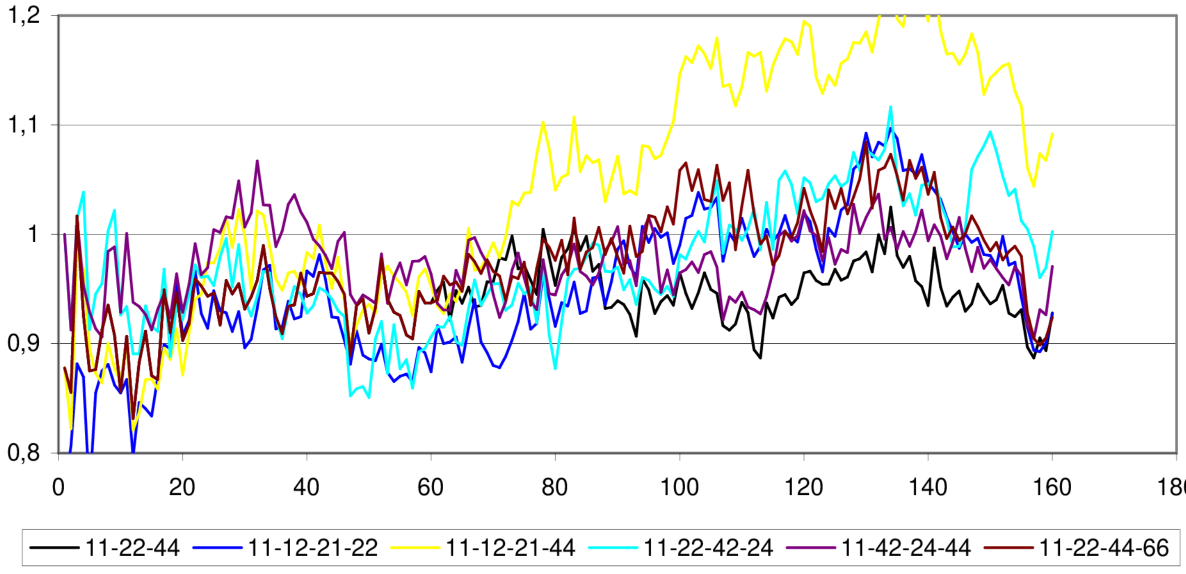


Figure 19: Classification error of face detection classifiers with restricted sizes of LRD relative to the classification error of a classifier with full set of LRD. The x-axis represents the number of weak hypotheses and the y-axis represents relative classification error (lower numbers are better). The pairs of digits in the description of the classifiers correspond to the sizes of the grid cells.

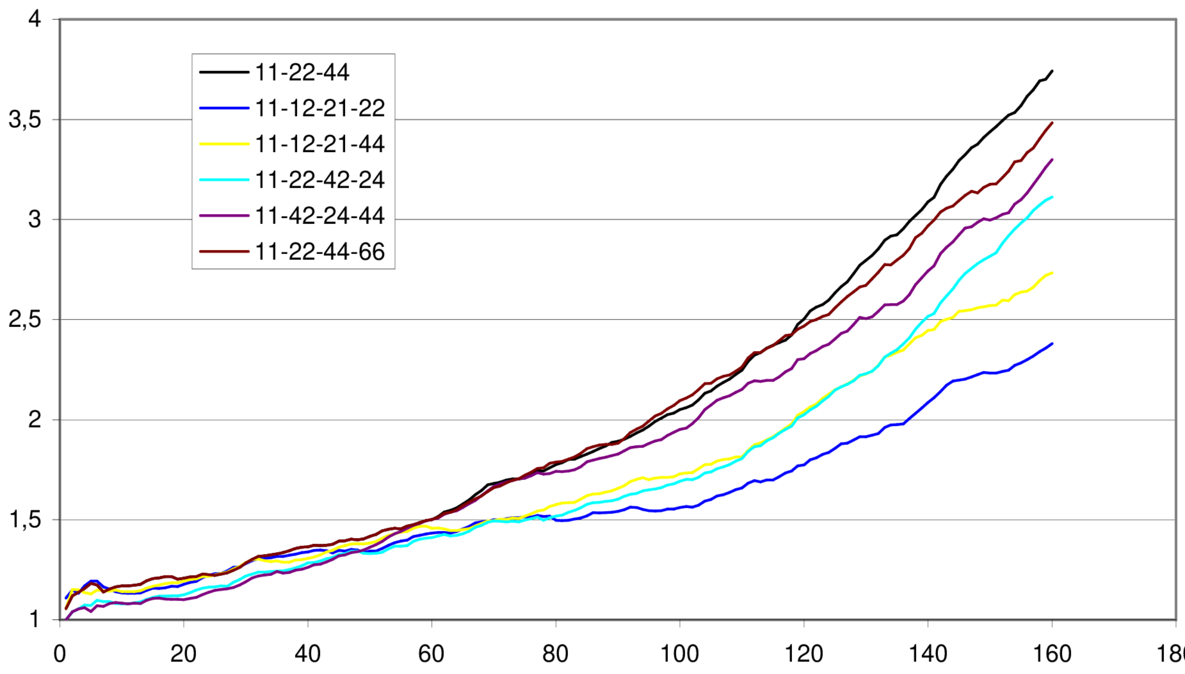


Figure 20: Speed of Z minimization of face detection classifiers with restricted sizes of LRD relative to the speed of Z minimization of a classifier with full set of LRD. The x-axis represents the number of weak hypotheses and the y-axis represents relative Z (lower numbers are better). The pairs of digits in the description of the classifiers correspond to the sizes of the grid cells.

7.4 Prediction Value Quantization

One of the limitations of an LRD evaluation engine in FPGA is the memory capacity needed to store the prediction values. Each LRD with grid size 3x3 needs 17 prediction values. In this context, lower precision of the prediction values implicates a possibility to use higher number of features in a single classifier. On the other hand, the reduced precision of the prediction values could result in higher error rate of the classifier or a need for longer classifiers to achieve the same error rate.

The figures 21 and 22 show the influence of reducing the precision of the prediction values on the classification error. The results show that even very low precision is sufficient. There is no measurable improvement of the classification error with more then 4b precision and the speed of Z minimization also almost does not increase with higher precision.

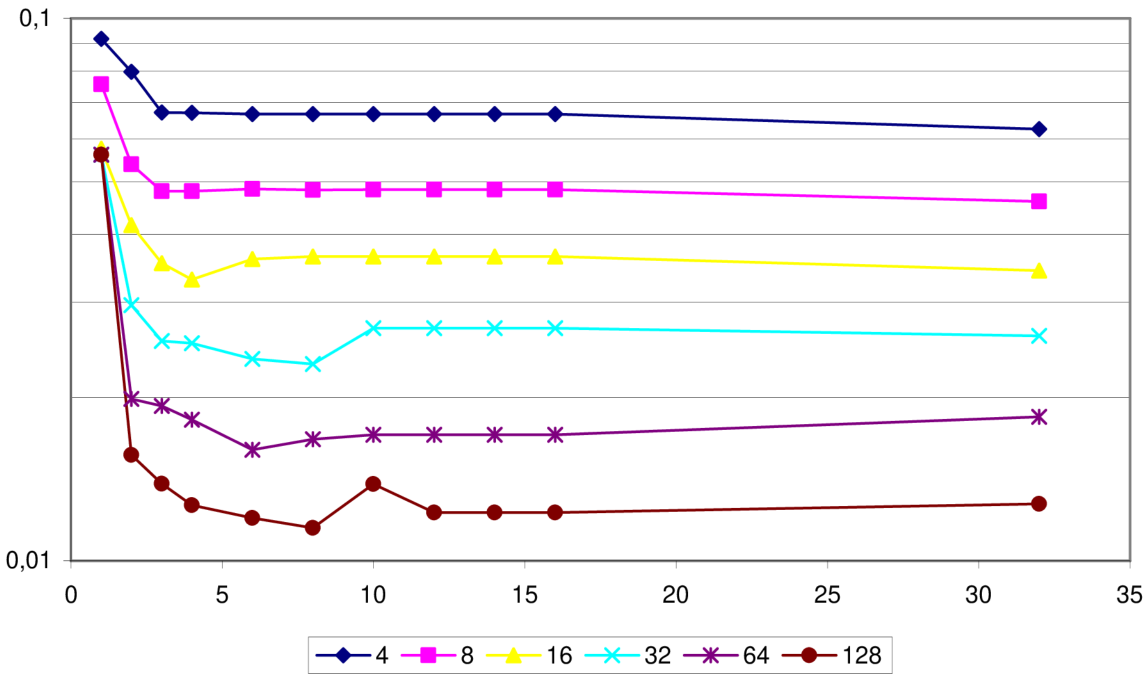


Figure 21: Relation between classification error and precision of the predictions. Each of the lines represents single length of the boosted classifier and the x-axis represents the number of bits used to represent the prediction values.

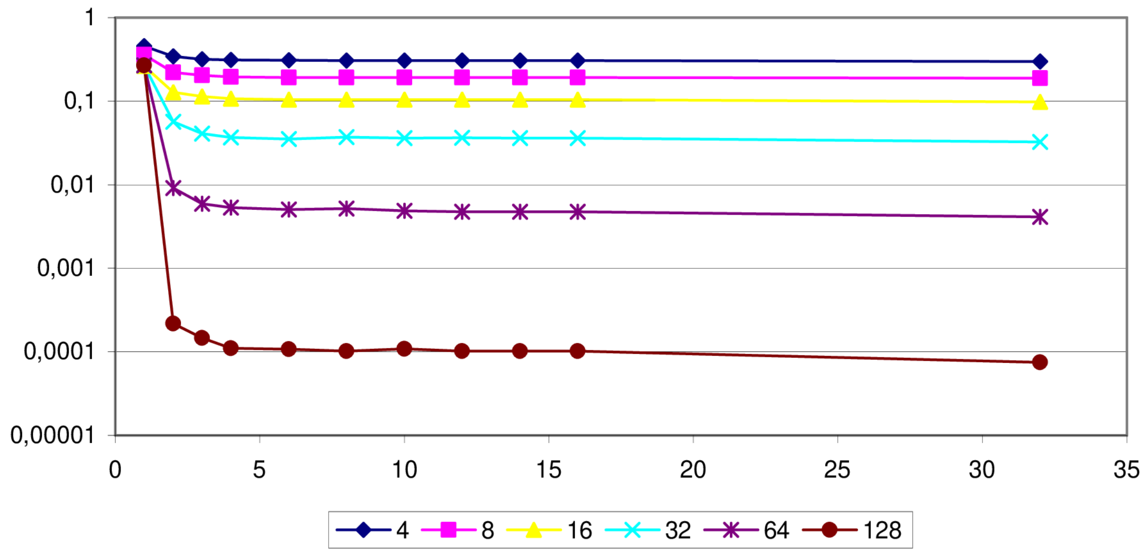


Figure 22: Relation between speed of Z minimization and precision of the predictions. Each of the lines represents single length of the boosted classifier and the x-axis represents the number of bits used for the prediction values.

7.5 Convolution Quantization

The value of each grid cell of the LRD features is based on mean intensity value inside the cell[1]. These mean intensities can be viewed as a convolution with a rectangular filter. In the FPGA, these convolutions are precomputed and stored locally on the FPGA chip. The lower the bit resolution of these convolutions is the larger part of the image fits in the memory of FPGA and, consequently, lager part of the image can be processed at single time. With lower bit resolution it is also possible to increase the number of distinct sizes of the convolution filters (the grid cells). On the other hand, lower bit precision results in loss of information especially for higher sizes of the cells.

To estimate the impact of reducing the precision of the convolutions, we created classifiers with artificially restricted convolution precision. The results of these classifiers are shown on figure 23. The convolution sizes used in this experiment were 1x1, 2x2, 2x4 and 4x2. The results show that there is no significant degradation of performance with precision 5b and higher.

[1] For more information about LRD features see section 5.2.

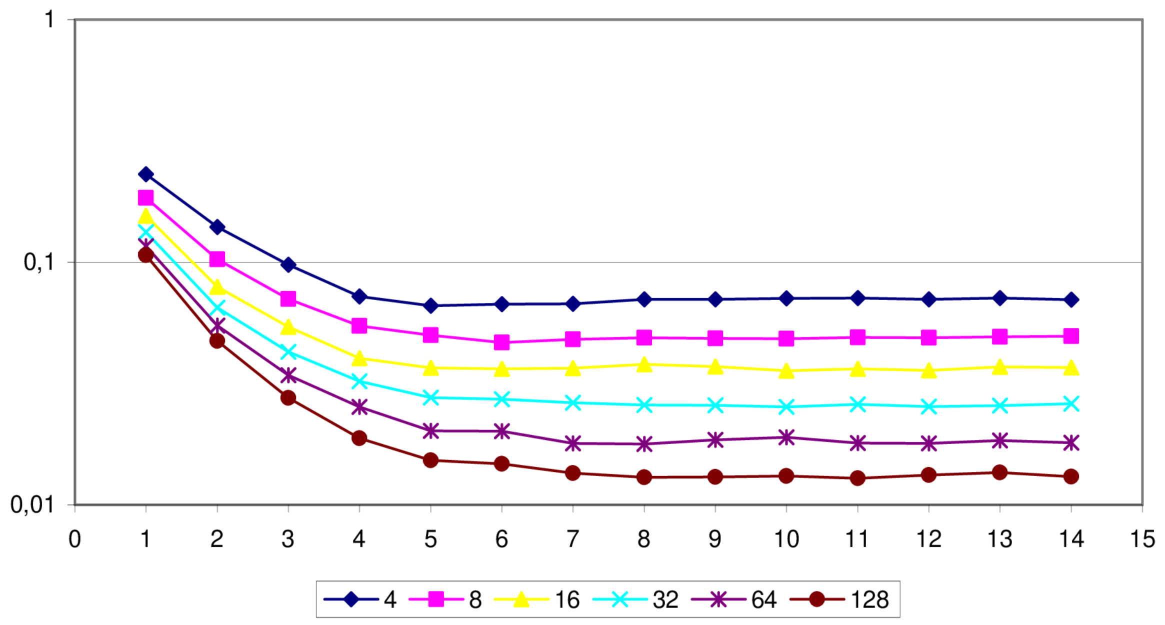


Figure 23: Relation between classification error and precision of the convolutions in LRD. Each of the lines represents single length of the boosted classifier and the x-axis represents the number of bits used for the convolutions.

8 Conclusions and Future Work

In this thesis, novel image features are presented. The LRD are suitable for object detection classifiers which can be efficiently evaluated in FPGA. The performance of AdaBoost classifiers with the LRD was tested on a face detection dataset with results which are similar to the state of the art Haar-like features. Other experiments were performed to estimate performance of the classifiers with LRD when evaluated in FPGA. This was done because some limitations of the FPGA evaluation engine could degrade the classifier performance. The presented results show that the limitations do not degrade the classification performance in any significant way. These results are very encouraging and suggest that the LRD may be a solution for object detectors in hardware.

The thesis also presents a framework for experimenting with boosting methods focusing mainly on computer vision applications. This framework is very flexible and makes it possible to simply plug in a new code with boosting algorithm, weak learner, features or data source. The framework also offers high efficiency when learning classifiers and a possibility for future parallelization. The framework is available as open source software and we hope that it will simplify work for other researchers.

In the feature, we plan to add other boosting algorithms into the framework as well as some of the early decision termination classifiers. Further, we plan to explore possible applications of classifiers with the LRD in FPGA. For example, we want to experiment with emulation of corner detectors.

References

[1] Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In D. Haussler, editor, 5th Annual ACM Workshop on COLT, Pittsburgh, PA, ACM Press, 1992, s. 144-152.

[2] Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119--139, August 1997.

[3] Freund, Y., Schapire, R..: A short introduction to boosting. J. Japan. Soc. for Artif. Intel. 14(5), 1999, s. 771-780.

[4] Schapire, R.: The boosting approach to machine learning: An overview. In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, Mar. 2001.

[5] Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In CVPR, 2001.

[6] Lienhart, R., Maydt, J.: An extended set of Haar-like features for rapid object detection. ICIP (1) 2002, s. 900-903.

[7] Sochman, J., Matas, J.: Adaboost with totally corrective updates for fast face detection. AFGR04, 2004, s. 445-450.

[8] Li, S.Z., Zhang, Z.Q., Shum, H, Zhang, H.J.: FloatBoost learning for classification. In S. Thrun S. Becker and K. Obermayer, editors, NIPS 15. MIT Press, December 2002.

[9] Xiao, R., Zhu, L., Zhang, H.J.: Boosting Chain Learning for Object Detection. ICCV03, Nice, France, 2003.

[10] Dong Zhang, Li, S.Z., Gatica-Perez, D.: Real-time face detection using boosting in hierarchical feature spaces. Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, Vol.2., 2004, s. 411-414.

[11] Takeshi Mita, Toshimitsu Kaneko, Osamu Hori: Joint Haar-like Features for Face Detection. ICCV 2005, s. 1619-1626.

[12] Yubo Wang, Haizhou Ai, Bo Wu, Chang Huang: Real time facial expression recognition with AdaBoost. In Proceedings of the 17th International Conference on Pattern Recognition, 2004.

[13] Xinwen Hou, Cheng-Lin Liu, Tieniu Tan: Learning Boosted Asymmetric Classifiers for Object Detection. Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1, 2006, s. 330 – 338.

[14] Theocharides, T., Vijaykrishnan, N., Irwin, M.: A Parallel Architecture for Hardware Face Detection. isvlsi, IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06), 2006, s. 452-453.

[15] Ming Yang, Ying Wu, Crenshaw, J., Augustine, B., Mareachen, R.: Face detection for automatic exposure control in handheld camera. icvs, Fourth IEEE International Conference on Computer Vision Systems (ICVS'06), 2006, s. 17.

[16] Granát, J., Herout, A. Hradiš, M., Zemčík, P.. Hardware Acceleration of AdaBoost Classifier. Poster MLMI, 2007.

[17] Zemčík, P. Hradiš, M., Herout, A.: Local Rank Differences - Novel Features for Image Processing. Poster MLMI, 2007.

[18] Zemčík, P., Žádník, M.: AdaBoost Engine. In FPL, 2007.

[19] Koeppen, M.: The Curse of Dimensionality. 5th Online World Conference on Soft Computing in Industrial Applications (WSC5), held on the internet, September 4-18, 2000.

[20] Valiant, L. G.: A theory of the learnable. Communications of the ACM 1984, s. 1134-1142.

[21] Haussler, D.: Overview of the Probably Approximately Correct (PAC) Learning Framework. 1995.

[22] Kearns, M., Valiant, L.: Learning Boolean Formulae or Finite Automata is as Hard as Factoring. Technical Report TR-14-88, Harvard University Aiken Computation Laboratory, August 1988.

[23] Kearns, M., Valiant, L.: Cryptographic limitations on learning Boolean formulae and finite automata. Journal of the Association for Computing Machinery, 41(1), January 1994, s. 67-95.

[24] Schapire, R.: The strength of weak learnability. Machine Learning, 5(2), 1990, s. 197-227.

[25] Freund, Y.: Boosting a weak learning algorithm by majority. Information and Computation, 121(2), 1995, s. 256-285.

[26] Schapire, R., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. In: Machine Learning, 37(3), 1999, s. 297-336.

[27] Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In D. Haussler, editor, 5th Annual ACM Workshop on COLT, Pittsburgh, PA, ACM Press, 1992, s. 144-152.

[28] Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.: Learnability and the Vapnik-Chervonenkis Dimension. Journal of the ACM, 36(4), 1989, s. 929-965.

[29] Breiman, L.: Arcing classifiers. The Annals of Statistics, 26(3), 1998, s. 801-849.

[30] Drucker, H., Cortes, C.: Boosting decision trees. In Advances in Neural dings Information Processing Systems 8, 1996, s. 47-485.

[31] Quinlan, J.: Bagging, boosting, and C4.5. In Proceedings of the Thirteenth international Conference on Artificial Intelligence, 1996, s 725-730.

[32] Schapire, R., Freund, Y., Bartlett, P., Wee Sun Lee: Boosting the margin: A new explanation for the effectiveness of voting methods. In Machine Learning: Proceedings of the Fourteenth International Conference, 1997.

[33] Bennett, K., Mangasarian, O.: Robust linear programming discrimination of two linearly inseparable sets. Optimization Methods and Software, 1, 1992, s. 23-34.

[34] Cortes, C., Vapnik, V.: Support vector networks. Machine Learning, 20, 1995, s. 273-297.

[35] Rätsch, G.: Robust boosting via convex optimization. Dissertation Universität Potsdam, Institut für Informatik, Potsdam, 2001.

[36] Demiriz, A., Bennett, K., Shawe-Taylor, J.: Linear programming boosting via column generation. J. of Mach. Learning Research, 46(1), 2001, s. 225-254.

[37] Sochman, J., Matas, J.: WaldBoost - Learning for Time Constrained Sequential Detection. CVPR, (2), 2005, s. 150-156.

[38] Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. The Annals of Statistics, 28(2), April 2000, s. 337-374.

[39] Chui, C.: An introduction to wavelets. Elsevier, 1992. ISBN 0121745848

[40] Haar, A.: Zur Theorie der orthogonalen Funktionensysteme. Mathematische Annalen, 69, 1910, s. 331-371,

[41] Matas, J., Sochman, J.: Wald’s Sequential Analysis for Time-constrained Vision Problems. In ICRA’07, 2007.

[42] Bartlett, M. et al.: Recognizing Facial Expression: Machine Learning and Application to Spontaneous Behavior. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, 2005, s. 56-573.

[43] Zhou, M., Hong, W.: Face Verification Using GaborWavelets and AdaBoost. icpr, 18th International Conference on Pattern Recognition (ICPR'06), 2006, s. 404-407.

[44] Lee, T.S.:.Image representation using 2D Gabor wavelets. IEEE Transaction of Pattern Analysis and Machine Intelligence. Vol. 18, No. 10, 1996, s. 959-971.

[45] Ojala, T., Pietikäinen, M.: Unsupervised texture segmentation using feature distributions. Pattern Recognition, 32(3), 1999, s. 477-486.

[46] Mäenpää, T., Pietikäinen, M.: Multi-scale binary patterns for texture analysis. In: Image Analysis, SCIA 2003 Proceedings, Lecture Notes in Computer Science 2749, Springer, 2003, s. 885-892.

[47] Ojala, T., Pietikäinen, M., Mäenpää, T.: Gray scale and rotation invariant texture classification with local binary patterns. In: Computer Vision, ECCV 2000 Proceedings, Lecture Notes in Computer Science 1842, Springer, 2000, s. 404-420.

[48] Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on feature distributions. Pattern Recognition, 29, 1996, s. 51-59.

[49] Ojala, T., Pietikäinen, M., Mäenpää, T.: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence. 24(7), 2002, s. 971-987.

[50] Zhao, G., Pietikäinen, M.: Dynamic texture recognition using volume local binary patterns. Proc. ECCV 2006 Workshop on Dynamical Vision, Graz, Austria, 2006, s. 12.

[51] Ahonen, T., Hadid, A., Pietikäinen, M., Face description with local binary patterns: application to face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(12), 2006, s. 2037-2041.

[52] Heusch, G., Rodriguez, Y., Marcel, S.: Local Binary Patterns as an Image Preprocessing for Face Authentication. 7th International Conference on Automatic Face and Gesture Recognition (FG2006), Southampton, UK, April 2006, s. 9-14.

[53] Rodriguez, Y., Marcel, S.: Face Authentication Using Adapted Local Binary Pattern Histograms. Proc. 9th European Conference on Computer Vision (ECCV 2006), Graz, Austria, Vol. 4, May 7 – 13 2006, s. 321-332.

[54] Liao, S., Fan, W., Chung, A.C.S., Yeung, D.Y.: Facial expression recognition using advanced local binary patterns, Tsallis entropies and global appearance features. IEEE International Conference on Image Processing (ICIP 2006), 2006.

[55] Xianji Wang, Haifeng Gong, Hao Zhang, Bin Li, Zhenquan Zhuang: Palmprint Identification using Boosting Local Binary Pattern. Proc. 18th International Conference on Pattern Recognition (ICPR 2006), Hong Kong, 2006.

[56] Pietikäinen, M.: Image analysis with local binary patterns. In: Image Analysis, SCIA 2005 Proceedings, Lecture Notes in Computer Science 3540, Springer, 115-118, plenary presentation, 2005, s. 115-118.

[57] Ce Liu, Hueng-Yeung Shum.: Kullback-Leibler Boosting. cvpr, 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03) - Volume 1, 2003, s. 587.

[58] Chang Huang, Haizhou Ai, Bo Wu, Shihong Lao: Boosting Nested Cascade Detector for Multi-View Face Detection. icpr, 17th International Conference on Pattern Recognition (ICPR'04) - Volume 2, 2004, s. 415-418.

[59] Duy-Dinh Le, Shinichi Satoh: Ent-Boost: Boosting Using Entropy Measure for Robust Object Detection. icpr, 18th International Conference on Pattern Recognition (ICPR'06), 2006, s. 602-605.