Flow Counting Using Realboosted Multi-sized Window Detectors

Håkan Ardö, Mikael Nilsson, and Rikard Berthilsson

Lund University, Cognimatics AB

Abstract. One classic approach to real-time object detection is to use adaboost to a train a set of look up tables of discrete features. By utilizing a discrete feature set, from features such as local binary patterns, efficient classifiers can be designed. However, these classifiers include interpolation operations while scaling the images over various scales. In this work, we propose the use of real valued weak classifiers which are designed on different scales in order to avoid costly interpolations. The use of real valued weak classifiers in combination with the proposed method avoiding interpolation leads to substantially faster detectors compared to baseline detectors. Furthermore, we investigate the speed and detection performance of such classifiers and their impact on tracking performance. Results indicate that the realboost framework combined with the proposed scaling framework achieves an 80% speed up over adaboost with bilinear interpolation.

1 Introduction

Object detection and tracking are central problems to computer vision. Finding objects from a specific object class and keeping track of them is often a first step that other computer vision tasks rely on. It is therefore important to find a solution to the problem that is fast, accurate and robust to the changes that naturally occur in real photographic situations. Preferably the method should also allow for low memory consumption and use only fix-point operations. This is a challenging task that allows the detector to be embedded within modern surveillance cameras that still typically lack floating point units. Concerning speed, most often the number of objects from the object class is several orders of magnitude lower than the number of objects not belonging to the class. This is for example the case for face detection and pedestrian detection. Here, the object classifier normally uses a sliding window that is scaled to different sizes and placed at all possible locations in the image. For each instance, the classifier should output true or false. It follows that the object detection must process several thousands of instances for a single image, and it is desired to process several such images per second.

In this paper we investigate object detectors connected to temporal tracking, in order to perform flow counting. In particular, counting of humans and bicycles. These kind of flow counts are often used by, for example, traffic planers and scientists to asses how public infrastructure is used.

A. Fusiello et al. (Eds.): ECCV 2012 Ws/Demos, Part III, LNCS 7585, pp. 193–202, 2012.

Next section presents the flow counting framework. Section 3 presents the object detection and corresponding classifier design. Section 4 discusses the temporal tracking employed. Section 5 presents experimental results.

2 Flow Counting Framework

The framework utilized here goes from video input to a flow counting report, see Fig. 1. The first step, which typically is the main bottleneck processing-wise, is the object detection. In this paper, we will investigate three classifiers used for object detection. Following the detection is temporal tracking conducted in order to achieve consistency over the temporal domain. The tracker employed is the classical Kalman filter. That means a linear model is used to model the dynamics. It fits nicely when modelling the motion of, for example, bicycles traveling along a straight road. However, it might be a more crude approximation when modelling, for example, the motion of faces in a general video.

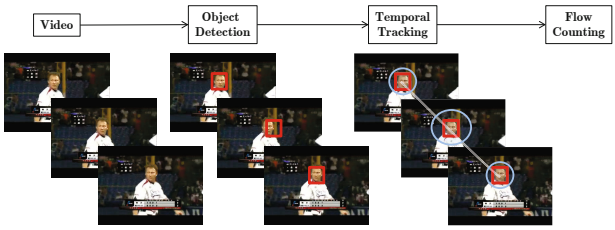

Fig. 1. Overview of framework for track counting in order to estimate flows. Images from Youtube faces database [1].

3 Object Detection

An early and successful approach to construct real-time detectors is to use a sliding window and a cascade of classifiers [2]. The detector use a set of Haar-like features, that can be computed relatively fast by using an integral image. The features are used as weak classifiers to train a cascade of adaboost [3] detectors. One problem is that the integral image requires a lot of memory. Furthermore, the features are sensitive to local changes in lighting. A later and more promising result was achieved by using features with a higher degree of invariance such as local binary patterns (LBP) [4] or local successive mean quantization transform (SMQT) [5,6]. These features take values from some discrete and finite set. Both use a sliding window as above and can be used in cascades. Compared to Haar-like features [2], they do not require any integral image and need fewer operations to compute feature values. Whereas, the detector in presented by Viola and Jones [2] can run on different scales, the LBP detector is however locked to one scale and the image itself has to be resized to different scales.

3.1 Multi-sized Windows

To overcome this resizing limitation we propose to train three differently sized detectors, with 16×16, 20×20 and 25×25 windows, respectively. All three detectors are used to scan the test image at its original size. Then the image is reduced to half its original size and all detectors are used again. This gives more or less the same result as using a single detector and only reducing the image size a factor 1.26 each time, but it operates faster. This is due to the fact that only one third of the image resizing operations have to be performed and, furthermore, no interpolation is needed here as the scale factor is exactly two. Note that these three detectors can be trained simultaneously. Using different window sizes or more than 3 sizes would be possible, but was not investigated. However, note that as the window size gets larger the detector gets somewhat slower. Also, the LBP features are local features and would pick up different things at different scales.

3.2 Boosting Real Valued Functions

The LBP and SMQT methods use features that take values from some discrete and finite set $\mathcal{X}$. For the classical LBP $\mathcal{X} = \{0, 1, \cdots, 255\}$. Using a training set consisting of pairs, $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, m$, where $\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,n}) \in \mathcal{X}^n$ are feature vectors and $y_i \in \{-1, 1\}$ the corresponding labels. Note that the feature vectors consists of n features.

According to the anyboost framework [7], most boosting algorithms can be described as a gradient descent optimization in a function space, $\mathrm{lin}(\mathcal{F})$, consisting of all linear combinations of base classifiers $f \in \mathcal{F}$. Here, we will consider the case when $\mathcal{F}$ consists of all real valued lookup tables of a single discrete feature, $\mathcal{X} \to \mathbb{R}$, such as LBP. The values of the lookup table $f_t \in \mathcal{F}$ will be denoted $a_{t,v}$ with $f_t(v) = a_{t,v}$. The table f_t will be used for feature number t and we will use the notation $f_t(\mathbf{x}_i) = f_t(x_{i,t})$.

The objective of the training is to find a detector function, $F \in \mathrm{lin}(\mathcal{F})$, that as often as possible, classifies the training data correctly, i.e. $y_i = \mathrm{sign}(F(\mathbf{x}_i))$. The final detector is found by minimizing the cost function

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} c(y_i F(\mathbf{x}_i)), \tag{1}$$

where $c : \mathbb{R} \to \mathbb{R}$ is a non-negative, decreasing function. Different boosting algorithms can be derived by using different cost functions c. Adaboost uses $c(\alpha) = e^{-\alpha}$. Gradient descent is an iterative algorithm that in each iteration updates its current detector function F by finding the direction $f \in \mathcal{F}$ in which $C(F + \epsilon f)$ most rapidly decreases. In the case of adaboost, a line search is performed to find the optimal ϵ, which is used to update the current detector function F.

According to the anyboost framework [7], the optimal direction is found as the function, f, maximizing

$$-\langle \nabla C(F), f \rangle, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product

$$\langle F, G \rangle = \sum_{i=1}^{m} F(\mathbf{x}_i) G(\mathbf{x}_i) \tag{3}$$

on $\text{lin}(\mathcal{F})$. This is only true if the optimization is restricted to functions f of unit length, i.e $\langle f, f \rangle = 1$. In the anyboost setting [7], a scaled version of (3) is used so that $\langle f, f \rangle = 1$ for all $f : \mathcal{X}^n \to \{-1, 1\}$, which means that it is not necessary to constrain the optimization. In the present case of $f : \mathcal{X}^n \to \mathbb{R}$ there exists no scaled version of (3) with this property and the constraint has to be considered during the optimization.

By introducing a set of weights or a distribution over the training samples,

$$d_i = \frac{c'(y_i F(\mathbf{x}_i))}{\sum_{i=1}^{m} c'(y_i F(\mathbf{x}_i))}, \tag{4}$$

the target function (2) can be written as (see [7] for details)

$$\frac{1}{m} \sum_{i=1}^{m} y_i f(\mathbf{x}_i) d_i. \tag{5}$$

In the anyboost case where f only takes values in $\{-1, 1\}$, the optimal direction is the one that minimizes the weighted classification error

$$\sum_{i \mid y_i \hat{f}(\mathbf{x}_i) = -1} d_i. \tag{6}$$

Generalizing this result to the present case where f can take any real value is done in Lemma 2. For each fixed feature x_t, $1 \leq t \leq n$, Lemma 1 gives the optimal coefficients of f_t. Schapire and Singer [8] have provided the same result in form of a closed form solution for a case that is equivalent to the single feature setting here. That is, given a feature x_t, $1 \leq t \leq n$, how should the values $a_{t,v} \in \mathbb{R}$ be chosen? In Lemma 1 a different proof based on the anyboost [7] approach is given. This alternative derivation also allow the design of an algorithm for choosing the optimal feature t, $1 \leq t \leq n$, to add in each iteration, which is presented in Lemma 2.

Lemma 1. *Given a classifier $F(\mathbf{x}) = \sum_t f_t(\mathbf{x})$ and a feature number, $1 \leq t \leq n$, the coefficients, $a_{t,v}$, of a lookup table, $f_t(v) = a_{t,v}$, that minimizes $C(F + f_t)$ from (1) with $c(\alpha) = e^{-\alpha}$ is given by*

$$a_{t,v} = \frac{1}{2} \log \frac{\sum_{i \mid x_{i,t} = v, y_i = 1} d_i}{\sum_{i \mid x_{i,t} = v, y_i = -1} d_i}. \tag{7}$$

Proof. To find the coefficients of f_t that minimizes the cost function

$$C(F + f_t) = \sum_{i=1}^{m} c(y_i F(\mathbf{x}_i) + y_i f_t(\mathbf{x}_i)), \tag{8}$$

the terms are reordered and grouped based on the feature value, which leads to

$$C\left(F+f_t\right)=\sum_{v\in\mathcal{X}}\sum_{i|x_{i,t}=v}c\left(y_iF\left(\mathbf{x}_i\right)+y_if_t\left(\mathbf{x}_i\right)\right). \tag{9}$$

Now $f_t\left(\mathbf{x}_i\right)=f_t(v)=a_{t,v}$ *is constant with regard to the inner sum. Each term of the outer sum can thus be optimized separately as there are no dependencies between them. Differentiating gives*

$$\frac{\partial C}{\partial a_{t,v}}=\sum_{i|x_{i,t}=v}y_ic'\left(y_iF\left(\mathbf{x}_i\right)+y_ia_{t,v}\right). \tag{10}$$

Using the same cost function as adaboost, $c\left(\alpha\right)=e^{-\alpha}$*, gives* $c'\left(\alpha\right)=-e^{-\alpha}$*, and*

$$\frac{\partial C}{\partial a_{t,v}}=-\sum_{i|x_{i,t}=v}y_ie^{-y_iF(\mathbf{x}_i)-y_ia_{t,v}}. \tag{11}$$

The sum can be separated into one sum for positive examples and one for negative. The factor depending on $a_{t,v}$ *can be factored out, so that*

$$\frac{\partial C}{\partial a_{t,v}}=\sum_{i|x_{i,t}=v,y_i=-1}e^{-y_iF(\mathbf{x}_i)}e^{a_{t,v}}-\sum_{i|x_{i,t}=v,y_i=1}e^{-y_iF(\mathbf{x}_i)}e^{-a_{t,v}}. \tag{12}$$

Solving $\frac{\partial C}{\partial a_{t,v}}=0$ *for* $a_{t,v}$ *gives the optimal coefficients,*

$$a_{t,v}=\frac{1}{2}\log\frac{\sum_{i|x_{i,t}=v,y_i=1}e^{-y_iF(\mathbf{x}_i)}}{\sum_{i|x_{i,t}=v,y_i=-1}e^{-y_iF(\mathbf{x}_i)}}. \tag{13}$$

Anyboost maintains a set of weights, d_i*, or a distribution, over the examples,*

$$d_i=\frac{c'\left(y_iF\left(\mathbf{x}_i\right)\right)}{\sum_{i=0}^{m}c'\left(y_iF\left(\mathbf{x}_i\right)\right)}=\frac{e^{-y_iF(\mathbf{x}_i)}}{\sum_{i=0}^{m}e^{-y_iF(\mathbf{x}_i)}}, \tag{14}$$

where the second equality holds for the adaboost cost function. Expressing $a_{t,v}$*, by using these weights, concludes the proof.*

□

Lemma 2. *The feature number,* $1\leq t^*\leq n$*, for which the cost function (1) decreases most rapidly is given by*

$$t^*=\underset{1\leq t\leq n}{\operatorname{argmax}}\sum_v\frac{\left(\sum_{i|x_{i,t}=v}y_id_i\right)^2}{\sum_{i|x_{i,t}=v}1}. \tag{15}$$

Proof. According to [7] the optimal t *is found by maximizing*

$$\frac{1}{m}\sum_{i=1}^{m}y_if_t\left(\mathbf{x}_i\right)d_i. \tag{16}$$

In the present setting where f_t can take any real value, the optimization has to be constrained to functions f_t for which $\langle f_t, f_t \rangle = 1$. The factor $1/m$ is constant and does not affect the position of the maximum and can thus be dropped. The terms of the remaining sum can be reordered and grouped based on the feature values, yielding

$$\sum_{v \in \mathcal{X}} \sum_{i|x_{i,t}=v} y_i f_t(\mathbf{x}_i) d_i. \tag{17}$$

The term $f_t(\mathbf{x}_i) = f_t(v) = a_{t,v}$ is constant with respect to the inner sum and can be factored out. That makes the inner sum constant and can be calculated from the current weights and the labels of the training data. The constraint $\langle f_t, f_t \rangle = 1$ implies that $\sum_{i=1}^{m} a_{t,x_{i,t}}^2 = 1$. By denoting the number of training examples with value v on feature t $c_{t,v} = \sum_{i|x_{i,t}=v} 1$, the constraint can be written $\sum_{v \in \mathcal{X}} c_{t,v} a_{t,v}^2 = 1$.

By introducing new coordinates, $\hat{a}_{t,v} = \sqrt{c_{t,v}} a_{t,v}$, it follows that $\|\hat{\mathbf{a}}_t\| = 1$, where $\hat{\mathbf{a}}_t = (\hat{a}_{t,0}, \hat{a}_{t,1}, \cdots)$. Introducing the new coordinates into (17) results in

$$\sum_{v \in \mathcal{X}} \frac{\hat{a}_{t,v}}{\sqrt{c_{t,v}}} \sum_{i|x_{i,t}=v} y_i d_i. \tag{18}$$

Let the inner sum normalized by $\sqrt{c_{t,v}}$ be denoted $b_{t,v}$ and note that it can be divided into negative and positive training samples. It follows that

$$b_{t,v} = \sum_{i|x_{i,t}=v} \frac{y_i d_i}{\sqrt{c_{t,v}}} = \sum_{i|x_{i,t}=v, y_i=1} \frac{d_i}{\sqrt{c_{t,v}}} - \sum_{i|x_{i,t}=v, y_i=-1} \frac{d_i}{\sqrt{c_{t,v}}}. \tag{19}$$

Equation (18) is the scalar product between the vectors $\hat{\mathbf{a}}_t$ and $\mathbf{b}_t = (b_{t,0}, b_{t,1}, \cdots)$, which can also be written $\|\hat{\mathbf{a}}_t\| \|\mathbf{b}_t\| \cos\phi$, where ϕ is the angle between $\hat{\mathbf{a}}_t$ and $\mathbf{b}_t$. The constraint $\langle f, f \rangle = 1$ implies that $\|\hat{\mathbf{a}}_t\| = 1$. Apart from that, $\hat{\mathbf{a}}_t$ can be chosen freely, which means it can be chosen parallel to $\mathbf{b}_t$ in which case $\cos\phi$ reaches its maximum, 1. The only factor left to maximize is $\|\mathbf{b}_t\|$, which means that the optimal feature t^* can be found by

$$t^* = \operatorname*{argmax}_t \|\mathbf{b}_t\|. \tag{20}$$

Substituting $\mathbf{b}_t$ for its definition (19) and removing the outermost square root (which does not affect the position of the maximum) concludes the proof.

□

A detector F can now be trained by iteratively choosing a feature t by using Lemma 2 and then select coefficients $a_{t,v}$ for f_t using Lemma 1. The weights d_i are maintained in the same manner as with adaboost. They are initiated to $1/m$ before the first iteration and then updated using $d_i := d_i e^{-a_{t,x_{i,t}} y_i}$.

4 Temporal Tracking

To count the number of tracks, the single frame detections are tracked across multiple frames. The tracking approach used is based on the Kalman filter.

It maintains one set of confirmed tracks and one set of tentative tracks. Both kind of tracks are maintained using Kalman filters with the position and velocity vectors as state and the position as observation.

The detections from a new frame are first matched to the confirmed tracks using the Hungarian method [9]. The Mahalanobis distance between the observation predicted by the Kalman filter and the detection is used as cost-function. The detections not matching any of those are then matched to the tentative tracks, again using the Hungarian method. New tentative tracks are produced for the detections not matching any existing or tentative tracks.

When enough detections have been assigned to a tentative track it becomes confirmed. If the variance of a track becomes too large it is considered lost and is removed. If such a track was confirmed it is counted and accumulated into the flow count report.

5 Experimental Results

Experiments were performed using a five stage cascade and a 3×3 neighborhood LBP feature and executed on a 2.93 GHz Core 2 Duo. Three detectors were trained using the classical adaboost (with binary weak classifiers) [3], realboost as presented in Section 3.1 and split up SNoW [6]. All three produce detectors of the same form, although the lookup tables produced by the classical adaboost are more restricted as the week classifiers are only allowed the values of -1 or 1 and not any real value as is the case with the other two classifiers. The cascades were formed by requiring a True Positive Rate (TPR) of 99.5% and a False Positive Rate (FPR) of 5% for each step.

5.1 Face Detection

The training data consists of 10620 manually annotated faces and 1713 high resolution images containing no faces. The test data consists of 1183 faces and 191 negative images. The test data described above was used to verify the detection performance of the resulting detector, and the run time was evaluated on a 6000×2585 image[1] Results are shown in Table 1.

Note that the realboost detector has similar accuracy to adaboost or the split up SNoW, but it is more than 60% faster. This is a significant speedup, and stems from the fact that fewer features can be used in the initial steps effectively discarding a lot of negative examples at an early stage. Note that the exact same implementation were used for each of the detectors. The only difference is the values of the lookup-tables and the number of features used in each cascade step.

5.2 CMU+MIT

The three detectors trained on our database was also tested using the CMU+MIT test dataset (set A, B and C), with results presented in Table 2. This dataset

[1] `http://www.flickr.com/photos/kitty-kat/6049220331/`

Table 1. Results comparing the performance of three different detectors. The number of features used in each stage of the cascade is listed as well as the run time needed to process an image. The true positive rate (TPR) shows the amount of faces detected among the 1183 faces in the training database and the false positive rate (FPR) shows the number of false detections made per mega-pixel.

Training	Size	Features					Run time (s)		TPR	FPR
Adaboost	16×16	59	156	196	196	196	2.81	8.17	96.53 %	0.147 %
	20×20	55	141	264	324	324	2.80			
	25×25	58	139	249	431	418	2.98			
Split Up SNoW	16×16	81	57	44	14	0	3.00	8.94	95.86 %	0.071 %
	20×20	65	64	79	91	25	2.63			
	25×25	101	52	137	180	59	3.74			
Realboost	16×16	36	81	159	196	196	1.76	**4.93**	96.28 %	0.136 %
	20×20	32	75	128	232	278	1.72			
	25×25	37	72	121	180	195	1.85			

contain noisier images as compared to our training dataset, which seams to lead to a high false negative rate as LBP is noise sensitive. However this test confirms the increased speed of the realboost approach, which here is 50% faster than adaboost and 70 % faster than split up SNoW.

Table 2. The detection results from testing the trained detectors on our database on the CMU+MIT database together with the runtime for processing the entire database. In the Prop. columns results from using 3 window sizes and no interpolation, and in the Bilin. columns results from using a single window size and bilinear rescaling are shown.

	True Positives		False Negatives		False Positives		Runtime	
Training	Prop.	Bilin.	Prop.	Bilin.	Prop.	Bilin.	Prop.	Bilin.
adaboost	373	407	137	102	11	4	9.65 s	11.56 s
split up SNoW	333	356	177	154	5	5	10.85 s	14.9 s
realboost	356	403	154	107	9	7	**6.39** s	7.59 s

5.3 Single Sized Detector

The rescaling approach proposed was compared with the standard approach of using a single window size and rescaling the image using bilinear interpolation. A rescaling factor of $2^{1/3} \approx 1.26$ was used together with a step size of 1/16 of the window side. This gives approximately the same number of windows tested in both cases. Results on the CMU+MIT dataset is presented in Table 2. Realboost shows a 19 % speedup, split up SNoW 37 %, while adaboost is 20 % faster.

5.4 Tracking Faces

The output of the three detectors described above were passed to the Kalman tracker framework in order to perform face tracking. The Youtube faces [1] database were used for testing. It consists of 3425 video clips with one face trajectory marked out in each clip. In total there are 619139 frames. The tracks produced by the tracker were matched to the ground truth tracks as favorably as possible and the number of frames where the tracker placed the correct object close to the ground truth object were considered correct and counted. The results are presented in Table 3.

Table 3. The number of frames in which the detector detected (Matches) or missed (Misses) the annotated face as well as the same data after tracking. After tracking, the id-number of the detection also has to be correct for a frame to be considered matched.

	Detector		Tracker		
Training	**Matches**	**Misses**	**Matches**	**Misses**	**Runtime**
Adaboost	592723	28403	602503	16636	29.54 h
Split Up SNoW	580385	40741	599301	19074	29.17 h
Realboost	589174	31952	601142	17662	**26.16 h**

Note that while the number of frames where the realboost detector misses a face have increased with 11% as compared with the adaboost. However, the number of frames where the tracker misses a face have only increased by 5% as the tracker is able to fill in some missing detections.

5.5 Counting Bicycles

The proposed framework were tested on a bicycle counting scenario where a bicycle road was filmed for two days. The bicycles from the first day were marked manually and the detectors trained on that data. The bicycles from the second day was counted both manually and automatically and the results are compared in Figure 2. Processing 1000 frames required 10.76 s for realboost, 13.25 s for adaboost and 16.97 for the single window bilinear scaled adaboost. The mean absolute counting error over 30 min parts is 17.6 % for realboost, 9.8 % for adaboost and 12.3 % for bilinear scaling.

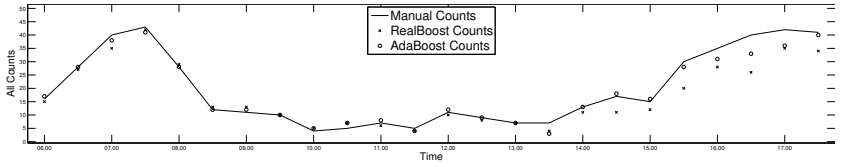

Fig. 2. Comparing automatic and manual bicycle counts

6 Conclusions

In this paper we present a novel approach to speed up real time object detections. This by exploring real valued weak classifiers and detectors with multiple sized windows. Results indicate that the realboost framework archives a 65% speedup over adaboost and a further 19% speedup by utilizing the proposed scaling framework compared to the common bilinear interpolation. In a tracking scenario the proposed solution can run close to 60% faster than standard adaboost implementation, with only a 5.3 percentage points loss in performance.

It should be possible to improve on those results by also optimizing over the window size of the detector, the number of window sizes used as well as the parameters of the Kalman filters. Also, in many flow counting applications, especially traffic studies, somewhat lower accuracy can be compensated for by increasing the amount of measurements.

References

1. Wolf, L., Hassner, T., Maoz, I.: Face recognition in unconstrained videos with matched background similarity. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 529–534 (2011)
2. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, pp. 511–518 (2001)
3. Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning and An Application to Boosting. In: Vitányi, P.M.B. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 23–37. Springer, Heidelberg (1995)
4. Ojala, T., Pietikainen, M., Harwood, D.: Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In: Proceedings of the 12th IAPR International Conference on Pattern Recognition - Conference A: Computer Vision Image Processing, vol. 1, pp. 582–585 (October 1994)
5. Nilsson, M., Dahl, M., Claesson, I.: The successive mean quantization transform. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 429–432 (March 2005)
6. Nilsson, M., Nordberg, J., Claesson, I.: Face detection using local smqt features and split up snow classifier. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (April 2007)
7. Mason, L., Baxter, J., Bartlett, P.L., Frean, M.R.: Boosting algorithms as gradient descent. In: NIPS, pp. 512–518 (1999)
8. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Mach. Learn. 37, 297–336 (1999)
9. Kuhn, H.W.: The hungarian method of solving the assignment problem. Naval Res. Logistics Quart. 2, 83–97 (1955)