Neural Networks 22 (2009) 748–756

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2009 Special Issue

Boosting feature selection for Neural Network based regression

Kevin Bailly *, Maurice Milgram [1]

Institut des Systèmes Intelligents et de Robotique, Université Pierre et Marie Curie-Paris 6, CNRS, UMR 7222, 4 place Jussieu, 75005 Paris, France

ARTICLE INFO

Article history:
Received 5 May 2009
Received in revised form 11 June 2009
Accepted 25 June 2009

Keywords:
Input feature selection
Boosting
Regression
Fuzzy functional criterion

ABSTRACT

The head pose estimation problem is well known to be a challenging task in computer vision and is a useful tool for several applications involving human–computer interaction. This problem can be stated as a regression one where the input is an image and the output is pan and tilt angles. Finding the optimal regression is a hard problem because of the high dimensionality of the input (number of image pixels) and the large variety of morphologies and illumination. We propose a new method combining a boosting strategy for feature selection and a neural network for the regression. Potential features are a very large set of Haar-like wavelets which are well known to be adapted to face image processing. To achieve the feature selection, a new Fuzzy Functional Criterion (FFC) is introduced which is able to evaluate the link between a feature and the output without any estimation of the joint probability density function as in the Mutual Information. The boosting strategy uses this criterion at each step: features are evaluated by the FFC using weights on examples computed from the error produced by the neural network trained at the previous step. Tests are carried out on the commonly used Pointing 04 database and compared with three state-of-the-art methods. We also evaluate the accuracy of the estimation on FacePix, a database with a high angular resolution. Our method is compared positively to a Convolutional Neural Network, which is well known to incorporate feature extraction in its first layers.

1. Introduction

In a large number of regression problems, it is not easy to find relevant features from a huge set of potential input variables. In image based regression for instance, the number of pixels, which corresponds to the input dimension, is very large. Moreover, pixel level is not necessarily suitable for a particular problem and a huge set of potential features can be extracted from an image to complete or to supply the necessary information. Just few of them are relevant, while most of them are redundant. In this paper, we present a framework to learn simultaneously relevant features and the corresponding regressor.

A large number of solutions have been proposed in the feature selection literature. They can be divided into filter, wrapper and embedded approaches (Guyon & Elisseeff, 2003). In the filter based method, features are first selected using a specific measure of relevance such as mutual information or Pearson's correlation. The second stage consists of estimating parameters of the regression model on the selected subset. This approach is computationally efficient, but presents a major drawback (Kohavi & John, 1997): the regression model is not taken into account. Thus, a suboptimal set of relevant features tends to be selected rather than the complete set of useful features.

Wrapper approaches use the prediction performance of the model to quantify the relevance of a feature subset. In this category, greedy forward selection algorithms are frequently used: they progressively integrate new variables which optimize the regression score. This strategy is really time consuming and intractable when the regression learning algorithm is too complex.

Embedded approaches incorporate the feature selection directly into the learning algorithm. In ridge regression such as LASSO (Tibshirani, 1996) or Input Decay (Chapados & Bengio, 2001), a regularization term is introduced in the cost function. The aim is to train the model so that inputs which contribute poorly to the regression process are penalized. These methods are well suited when the number of potential features is quite restricted.

In Section 2, we introduce BISAR (Boosted Input Selection Algorithm for Regression) which combines a filter and a wrapper approach.

BISAR uses a kind of boosting in the feature selection process. In its simplest form, the boosting strategy aims to decrease the error rate of a classifier/regressor by concentrating at each iteration on examples that are particularly difficult to classify/regress. This is usually done by iteratively adapting the training samples weighting. At each iteration the weight of the training examples

* Corresponding author. Tel.: +33(0)144276309.
E-mail addresses: kbailly@gmail.com, kevin.bailly@isir.fr (K. Bailly), maurice.milgram@upmc.fr (M. Milgram).
[1] Tel.: +33(0)144275197.

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.06.039

depends on the performance of the classifier/regressor in the previous iteration.

AdaBoost (Freund & Schapire, 1995) is a popular algorithm to iteratively build a classifier as a linear combination of the so-called weak classifiers. At each step, a new weak classifier is added optimizing the classification error rate with a new weighting on training samples. AdaBoost.RT, proposed by Shrestha (Shrestha & Solomatine, 2006) for regression problems uses the so-called absolute relative error threshold ϕ to project training examples into two classes (poorly and well predicted examples) by comparing the absolute relative error with the threshold ϕ. The problem is that it is not obvious to choose the right value for the threshold ϕ. Therefore we have selected another boosting strategy, AdaBoost.R2 (Drucker, 1997), to compare several boosting strategies in BISAR. There are other methods like Redpath and Lebart (2005) and Xu and Zhang (2006) that also combine these two approaches in a different way. The algorithm described in Redpath and Lebart (2005) is clearly dedicated to solving classification and not regression problems and aims to reduce the number of features used by each weak classifier. In Xu and Zhang (2006), an approach which limits the correlations of the selected feature set for gene selection in microarray data analysis is presented. This approach seems to be specific to this kind of problem and the boosting strategy is completely different from BISAR.

The two main contributions of this paper are:

1. The Fuzzy Functional Criterion (FFC): a new filter used to select *relevant* features (Section 2.1)
2. A new boosting strategy that selects incrementally new *complementary* inputs for the regressor (Section 2.3).

A large number of experiments (Section 3) with BISAR on a classical pattern recognition problem is covered in this article (Section 4). Comparisons are drawn between BISAR and state-of-the-art methods (Section 5). Finally we present conclusions and prospects in Section 6.

2. Boosted Input Selection Algorithm for Regression (BISAR)

The regression we propose is based on two modules working together:

1. A boosted feature selection algorithm based on a new fuzzy criterion.
2. A neural network with growing input set according to the boosted incremental choice of features.

In order to select useful features to perform neural network based regression, we adapt both a filtering paradigm based on our new criterion FFC (Fuzzy Functional Criterion) independent from the regression engine and a boosting paradigm based on the neural network error.

We have a set of examples $\mathbf{x_i} \in \mathbb{R}^d$. To each $\mathbf{x_i}$ is associated a value $y_i \in \mathbb{R}$ that we want to predict. Data are divided into a training set A, a validation set V and a test set E. A set F of features H_k ($1 \leq k \leq N$) can be computed for each $\mathbf{x_i}$. F can be extremely large (typically more than 10 000 elements as in our case). The main objective of our method is to select a subset of features $FS \subset F$ adapted to a specific regressor. We can summarize our method as follows:

1. Initialize all examples at the same weight; FS is empty.
2. Compute the fuzzy criterion for all features in F using the training set and the current weights. The best feature according to this criterion is added to FS.
3. Train a new regressor taking as input all previously selected features.

1. initialisation : $t = 0$
   - Set initial values for $w^0 = (\frac{1}{M}, \ldots, \frac{1}{M})$ and $FS = \emptyset$
   - Compute Q used to evaluate the FFC criterion :
     $$Q(i) = \sum_j L_a(|h_{k,i} - h_{k,j}|).(1 - L_b(|y_i - y_j|))$$
2. iteration : $t = 1 \ldots T_{max}$
   - Evaluate the FFC criterion for each feature :
     $$FFC(H_k) = -\sum_i w_i Q(i)\ b_t = \arg\max_{k \in \{1,\ldots,N\}}(FCC(H_k))$$
   - Add the best feature H_{b_t} to the set of selected features. $FS \leftarrow FS \cup H_{b_t}$ and $F \leftarrow F \backslash H_{b_t}$.
   - Train R_t taking $(h_{b_1,i}, \ldots, h_{b_t,i})$ as input and y_i as output for each example $\mathbf{x_i}$.
   - Update the weight vector using one of the proposed boosting strategy (cf. 2.3)
3. The final regressor is the one with the lowest error on the validation set V during the $Tmax$ iterations

Fig. 1. BISAR algorithm.

4. Compute the new weights using the error of the regressor for each example.
5. Repeat from 2 to 5 until the maximum number of input is reached.
6. Select the regressor with the lowest error on the validation set V during all iterations.

Hence, our method combines an iterative filtering approach for feature selection and a neural network. The selection process uses weights provided by the network error at each step. It means that, given a set of pairs $(\mathbf{x_i}, y_i)$ of training input–output and weights on example, we determine a new score for each feature H_k. This score has to reflect the relevance of feature H_k for our problem, that is the matching between $\mathbf{x_i}$ (a pattern) and y_i (the target value associated to $\mathbf{x_i}$ that we want to predict) (Fig. 1).

2.1. Feature selection criterion

We have to determine a criterion to select features. This criterion must measure how much the output y depends functionally on this feature H_k. Moreover, this criterion should support a weighting function over the example set. Usual statistical measures (correlations, covariance, etc.) do not fit to our situation because of the linear hypothesis that we do not want to assume.

We are going to examine two criterions: the well-known Mutual Information (MI) and our new Fuzzy Functional Criterion (FFC) in the next sections.

2.1.1. Mutual Information

The MI between two discrete random variables x and y is defined to be:

$$I(x; y) = -\sum_{x,y} P(x, y) \log_2 \frac{P(x, y)}{P(x).P(y)} \quad (1)$$

and:

$$I(x; y) = H(x) + H(y) - H(x, y) = H(x)H(x|y) = H(y)H(y|x) \quad (2)$$

where $H(x)$ is the entropy of the random variable x and $H(x, y)$ is the joint entropy of these variables.

There is a continuous definition of the MI for continuous variables but it is necessary to approximate the MI before using it practically in a program. So, the main problem is the estimation of the joint probability distribution function $P(x, y)$. We use the Parzen windows estimator to evaluate $P(x, y)$ which is a popular non-parametric approach and which is also very close to the computation performed by our FFC criterion. It is well known

that the size of Parzen windows can be critical and has to be tuned finely. Unfortunately, in our case, we do not have any prior information about the probability distribution function $P(x, y)$. We will present in Section 4.1 a comparison between the MI and the FFC, however, we show here some arguments against the MI, even if this measure is still a very good and broadly used tool:

1. MI computation lays on the probability distribution function (pdf) $P(x, y)$ which can be difficult to estimate, particularly if the pdf has a high dynamic.
2. MI is symmetric: $I(x, y) = I(y, x)$; in our case, we want to measure the fact that y is a function of x and not the converse.
3. We shall have to deal with weights on samples (see the further section on boosting) and the criterion has to incorporate these weights. Due to its structure, we have to fully compute the MI for each feature and for each step of our algorithm.
4. To transpose our method to higher dimension feature or output, it becomes exponentially harder to estimate the pdf due to the famous "curse of dimensionality".

We propose in the next paragraph a new criterion, the so-called Fuzzy Functional Criterion (FFC):

1. FFC does not need the joint pdf of (x, y)
2. FFC is not symmetric
3. FFC can be precomputed independently of the weights and be very quickly computed when the weights are known
4. FFC can be easily transpose to higher dimension feature or output.

2.1.2. Fuzzy Functional Criterion (FFC)

Our Fuzzy Functional Criterion is like an energy function. The basic idea is the following: we want to evaluate the relation between two variables u and v (u is a measure and v a target to be predicted). We call P the logical proposition "u_1 and u_2 are close" and Q the logical proposition "v_1 and v_2 are close". If a functional smooth relation between u and v exists (i.e. $v = F(u)$ and F is a smooth function), we can say that the logical implication $P \Rightarrow Q$ is true. If the variable v takes a value in a discrete subset of the real numbers set, we have to adapt the criterion slightly. This is done using the truth table of the logical implication. We find that it is equivalent to "$\neg P$ or Q" and also to $\neg(P$ and $\neg Q)$. We take the fuzzy logic formulation of P and Q based on classical triangular shaped functions (Zadeh, 1994) denoted by L and defined as:

$$L_a(e) = \begin{cases} 1 - \dfrac{|e|}{a} & \text{if } |e| < a \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

To quantify the fact that "u_1 and u_2 are close" is true, we just take the value $L_a(|u_1 - u_2|)$ into account, where a is the spread of the triangular function and its value is discussed later. We can do the same for v and write:

$$Z = 1 - L_a(|u_1 - u_2|)(1 - L_b(|v_1 - v_2|)). \tag{4}$$

Z is a fuzzy evaluation of our implication $P \Rightarrow Q$. To build our criterion, we have to sum up Z over all (u_1, u_2, v_1, v_2) (the constant "1" can be dropped):

$$FFC = \sum_i \sum_j -L_a(|u_i - u_j|)(1 - L_b(|v_i - v_j|)). \tag{5}$$

The sum is taken over all quadruples. In our case, we replace u_i by $h_{k,i}$ (for some feature H_k) and v_i by the target y_i. We also introduce the weighting function on examples, w_i, which is a non-negative function with $\sum w_i = 1$. These weights will be useful in the boosting process. So we can reformulate our criterion as:

$$FFC(H_k) = -\sum_i w_i \sum_j L_a(|h_{k,i} - h_{k,j}|)(1 - L_b(|y_i - y_j|)). \tag{6}$$

The criterion $FFC(H_k)$ has to be maximized. Two parameters (a and b) control the spread of the L functions and are used to normalize the criterion over all features. In many problems, the range of u and v variables are bounded and can be determined. We shall tune a and b as a fixed proportion of these ranges. Experiments have shown that the criterion is not too sensitive to this tuning. Another way to cope with the normalization issue is to replace u values by their rank. In this option, the rank can be seen as a simple normalization: if there are N examples, the number $rank(x)/N$ is always in the interval [0 1]. Notice that the difference of ranks divided by the number of examples has the same distribution for all features.

The most interesting aspect is the weighting function on the examples does not imply to compute at each step a new FFC from scratch: it is possible to compute an intermediary form $Q(i)$ of FFC and to modulate it with the weighting function on examples based on the regression error. Eq. (6) can be written as:

$$FFC(H_k) = -\sum_i w_i Q(i) \tag{7}$$

where:

$$Q(i) = \sum_j L_a(|h_{k,i} - h_{k,j}|)(1 - L_b(|y_i - y_j|)) \tag{8}$$

and Q can be precomputed once.

2.2. Regressor

The regressor is the second important element of our system. We have tested different regressor from the Radial Basis Function (RBF) network family. They are known to be very efficient in function interpolation. An important feature is that their output becomes close to zero when the input is very far from any training sample; this can be used, for example, for a rejection purpose. RBF networks have three layers but are quite different from classical multilayer perceptron (MLP): the hidden layer is made of RBF units (and not sigmoid) learned via a specific algorithm. Output layer is fully connected to the hidden layer and is a linear neuron (or several linear neurons). There are two parts in the training of a RBF network: during the first one, hidden cells are chosen using the EM algorithm (or in our case the K-means algorithm). In the second part, a classical gradient descent algorithm finds the optimal weights for the connections between hidden layer neurons and the output layer neuron. A hidden neuron N_i can be viewed as a couple made of a point x_i in the input space and a spread value s_i. For an input x, the output y of the neuron N_i is computed by the spherical radial function:

$$y = \mathrm{e}^{\frac{-\|x - x_i\|}{s_i}}. \tag{9}$$

It is also possible to replace the spherical function by the Gaussian function which uses the full covariance matrix S_i:

$$y = \mathrm{e}^{-(x - x_i)^T S_i^{-1} (x - x_i)}. \tag{10}$$

To learn hidden neurons' values, we can use either an EM or a simple K-means algorithm. It has been shown that the latter is a more robust procedure. In EM approach, a covariance matrix of each cluster is computed at each step and it happens frequently that this matrix is poorly conditioned. This fact leads to great instabilities during the learning process. In K-means approach, covariance matrix of each cluster is computed once at the end of the process. The Euclidean distance is used in the clustering step. After the choice of the hidden neuron number N, a random initialization of centers x_i is done. Then, at each step, we select for each training sample x the nearest center x_i producing a cluster C_i of all training samples close to x_i. Then, for each cluster, we compute

m_i the center of gravity of C_i and set x_i to m_i. The process runs until x_i are stable or the maximum number of iterations is reached. Once hidden neurons are learned, we determine other weights (between hidden and output layers) via a gradient descent to minimize the Mean Square Error between desired output and network output.

We also used a Generalized Regression Neural Network (GRNN) which can be considered as a simplified RBF network (Wasserman, 1993). Each center of the hidden unit corresponds to an example of the training set and weights between hidden and output layers are set to the target values. It means this network does not need any training step but instead just keeps all prototypes (in the hidden layer) and all targets (in the second layer weights). This network can be viewed as a Parzen window procedure applied to an interpolation task.

2.3. Boosting strategy

At the beginning, the neural network starts with only one input cell which corresponds to the best feature given by the criterion using a uniform weighting function. In the second step, a new network is trained with 2 input cells: the new input corresponds to the second best feature which is selected among all features (except the feature already selected) using weights provided by the error of the first network. It means that examples that are poorly processed by the first network will receive higher weights than others. By doing that, we believe that the resulting weights, when fed to the criterion computation, will lead the criterion to choose a new feature better adapted to these examples. It is clear that our boosting paradigm enhances the regression system. Contrary to AdaBoost (Schapire, 2003), iterative reweighting is only used to select the best complementary feature. Features combination is entirely handled by the regressor and weights are taken into account. It can be seen as a regularization to prevent overfitting.

Several strategies are possible to drive the selection process. One has to tune the weights according to the error and this weight adaptation has to be defined carefully.

We test three boosting mechanisms. The first one lies on a memoryless process. Weights at iteration $t+1$ are only related to the quadratic error ϵ_k^t at the previous iteration. In our case, we adopted this simple relation:

$$w_{t+1}(i) = \frac{(\epsilon_t(i))^2}{\sum_{i=1}^{M}(\epsilon_t(i))^2}. \tag{11}$$

The second boosting strategy is cumulative and the weight of each example depends on the regression model error and the previous weights.

$$\tilde{w}_{t+1}(i) = \begin{cases} w_t(i) & \text{if } \epsilon_t(i) < median_i(\epsilon_t(i)) \\ \max\{\alpha w_t(i), w_{\max}\} & \text{otherwise} \end{cases} \tag{12}$$

$$w_{t+1}(i) = \frac{\tilde{w}_{t+1}(i)}{\sum_{i=1}^{M}(\tilde{w}_{t+1}(i))}. \tag{13}$$

Weights of half the examples with the highest prediction error are multiplied by a constant accumulation factor α. Here $w_{\max}$ is a constant used to avoid overfitting. Typically, α is set to 1.1 and $w_{\max}$ to 0.1.

The third reweighting procedure is the same as in AdaBoost.R2 (Drucker, 1997). The performance of the regressor is measured using a loss function:

$$L_t(i) = \left(\frac{\epsilon_t(i)}{\max_{i=1\ldots M}\epsilon_t(i)}\right)^2. \tag{14}$$

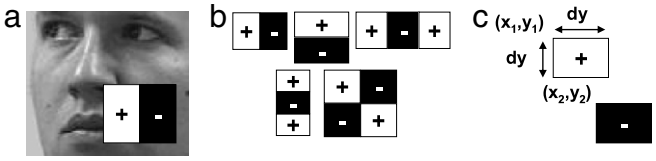

Fig. 2. Image features. (a) A feature example. (b) Representation of the Haar-like features used in our experiment. (c) The last type of features are the difference between the sum of the pixels within two non-connected rectangular regions.

This function is averaged over all the weighted examples

$$\bar{L_t} = \sum_{i=1}^{M} w_t(i)L_t(i). \tag{15}$$

Knowing $\bar{L_t}$, the weight updating parameter denoted β_t is computed as follows:

$$\beta_t = \bar{L_t}/(1-\bar{L_t}). \tag{16}$$

Finally, new weights on examples are calculated as:

$$\tilde{w}_{t+1}(i) = w_t(i)\beta_t^{(1-L_t(i))} \tag{17}$$

And normalized as:

$$w_{t+1}(i) = \frac{\tilde{w}_{t+1}(i)}{\sum_{i=1}^{M}(\tilde{w}_{t+1}(i))}. \tag{18}$$

Contrary to other boosting algorithms for regression, no parameter has to be calibrated and tests on different data sets have shown good results in Shrestha and Solomatine (2006).

In the three reweighting procedure, weights are normalized to make their set a distribution. In the rest of this paper, the three boosting strategies will be referred to as the memoryless (11), median (12) and AdaBoost.R2 (17) boosting strategies respectively.

3. Experimental setup

In order to test our approach, BISAR is applied to a head pose estimation problem. Research in head pose determination is now an important topic in pattern recognition and many face analysis applications such as face recognition systems and human–computer interaction need this information. Many solutions have been suggested in Murphy-Chutorian and Trivedi (2008). One of them considers head pose estimation as a nonlinear regression problem. These methods try to learn a functional mapping between an image (or a set of image features) and the head direction. This section will describe the features and the data sets used in our experiments.

3.1. Image features

Faces are down-sampled to a fixed size of 32×32 pixels. Moreover, color images are converted to gray scale images and histograms are dynamically adjusted. We use four types of features. The first descriptors correspond to the popular Haar-like features (Viola & Jones, 2002) as depicted in Fig. 2. We choose these features because of their good performance obtained in related area such as face detection (Viola & Jones, 2002) and alignment (Wu, Liu, & Doretto, 2008). Moreover they are simple and very easy to compute using the integral image (Viola & Jones, 2002).

Another kind of features, we propose here, are the difference between the sum of the pixels within two non-connected rectangular regions. Features are parameterized by four values x_1, y_1, dx and dy. x and y correspond respectively to the horizontal

Fig. 3. Pointing 04 database samples.

and vertical position of the feature in the image. *dx* and *dy* are the width and height of a rectangle. Two extra parameters x_2 and y_2 are needed to describe the features of the fourth category. They correspond to the position of the second rectangle in the image.

3.2. Data set 1: Pointing 04

3.2.1. Data set description

The first database used for this evaluation is the publicly available Face Pointing 04 database. It was used to evaluate head pose estimation systems in the Pointing 2004 Workshop on Visual Observation of Deictic Gestures. It was also included in the International Evaluation on Classification of Events Activities and Relationships (CLEAR 2006). The corpus consists of 15 sets of images. Each set contains 2 series of 93 images of the same person at 93 different poses. The first series is used for learning. Tests are carried out on the second series.

There are 15 peoples in the database, male and female, wearing glasses or not and with various skin color and facial hair. The pose is determined by 2 angles, ranging from $-90°$ to $+90°$ for the horizontal angle (pan) and from $-60°$ to $+60°$ for the vertical orientation (tilt). Fig. 3 depicts the variety of this database.

Ground truth was obtained by asking people to look at some specific markers in the room. In fact, this ground truth is strongly more related to the gaze direction than to the head direction. This acquisition process leads to a very challenging database with ambiguities and variations on pan angle result in very similar images.

3.2.2. Face localization

Faces in Pointing 04 are just a small area of the whole image. Thus a lot of information are not relevant and can disturb the regression process so we need somehow to crop the face area. However, because no generic multi-pose face localizer is available, we decided to manually crop the faces. We also proposed an ad hoc method similar to Gourier, Maisonnasse, Hall, and Crowley (2007) for two main reasons:

- To reduce the bias introduced by the manual cropping and test the approach on noisy data.
- To compare the results with other methods which integrate a face localization step.

Our face localization is based on skin color. First, frontal faces are detected in the training database using the Viola–Jones face detector (Viola & Jones, 2002). Pixels within the detected area are used to build a histogram on H (Hue) and S (Saturation) channels of HSV color space. Other pixels are randomly picked outside the face area in order to build a histogram of the background. Face pixels are detected in the test image set with a Bayesian rule based on these skin and non-skin histograms. The face bounding box size is proportional to the standard deviation of skin pixels along *X* and *Y* axes. In this method, only frontal faces of the training set are needed and no manual labelling is required. The main drawback is the weak accuracy of this method in few cases. For example, the neck can be included in the bounding box. The lack of precision in localization can greatly degrade head pose estimation results. In result section, results are presented for both manual and automatic face localization.

3.3. Data set 2: FacePix

The second data set used for this evaluation is the publicly available FacePix database (Little, Krishna, Black, & Panchanathan, 2005) from the Center for Cognitive Ubiquitous Computing (CUbiC). The FacePix database consists of three sets of face images: one set with pose angle variations, and two sets with illumination angle variations. Each of these sets are composed of a set of 181 face images (representing angles from $-90°$ to $+90°$ at $1°$ increments) of 30 different subjects, with a total of 5430 images. All the face images (elements) are 128 pixels wide and 128 pixels high. These images are normalized, such that the eyes are centered on the 57th row of pixels from the top, and the mouth is centered on the 87th row of pixels. The pose angle images appear to rotate such that the eyes, nose, and mouth features remain centered in each image.

In our experiment, we use the set with pose angle variation ranging from $-90°$ to $+90°$ with a step of $5°$. 20 faces were randomly picked from the original data for training, 6 for testing and 3 for cross-validation. Contrary to Pointing 04, only the pan parameter is modified. Moreover, labels on each image are reliable since the acquisition process is more accurate.

4. Experimental results

4.1. Feature selection criterion

In this first experiment we compare the Fuzzy Functional Criterion to the Mutual Information. In order to restrict the impact of outer factors, a GRNN is used and no boosting is applied, i.e. weights on example remains constant during iterations. The optimal number of inputs is chosen using the score on the validation set. Table 1 shows FFC has the edge over MI. Moreover FFC is faster to use Eq. (7) when boosting is introduced. These two reasons led us to give priority to FFC criterion in the next experiments.

4.2. Boosting strategy

In this section, we compare several boosting strategies. Tests are carried out using a GRNN. Thus no parameter needs to be tuned and results do not depend on random initialization. Results are presented for both FacePix and Pointing 04 databases. For this latter data set, tests are performed on manually and automatically cropped images (cf. 3.2.2). In this way, we can evaluate the

Table 1
FFC vs. MI: mean absolute error in degree on the test sets. The value in brackets indicates the corresponding number of features.

	Pointing 04				FacePix
	Manual cropping		Automatic cropping		
	Pan error	Tilt error	Pan Error	Tilt error	Pan error
FFC	**8.1**° (580)	**8.4**° (560)	**12.5**° (480)	15.9° (100)	**6.2**° (191)
MI	8.4° (600)	8.9° (560)	12.8° (460)	**15.7**° (140)	6.4° (180)

Table 2
Mean absolute error in degree on the test set for different boosting strategies. The value in brackets indicates the corresponding number of features.

	Pointing 04				FacePix
	Manual cropping		Automatic cropping		
	Pan error	Tilt error	Pan error	Tilt error	Pan error
None	8.1° (582)	8.4° (573)	12.5° (467)	15.9° (102)	6.2° (191)
Memoryless	**7.3**° (553)	8.5° (267)	11.6° (389)	13.5° (78)	5.4° (243)
Median	7.6° (569)	8.9° (223)	11.9° (464)	16.5° (55)	**4.5**° (599)
AdaBoost.R2	8.2° (382)	**7.6**° (457)	**10.9**° (430)	**12.3**° (120)	6.0° (265)

robustness to noise and outliers. Results on the test set for the three boosting strategies (cf. 2.3) are reported in Table 2. We also present results obtained without any boosting (denoted by 'none' in the table). We investigate the performance of BISAR with up to 600 iterations and we select the classifier with the best performance on the validation set.

The effect of boosting is evident as any reweighting method outperforms results without boosting. In addition, the AdaBoost.R2 reweighting function seems slightly better since it gives better results for three of the five test sets. Fig. 4 depicts the evolution of the mean absolute error against number of selected features. It shows that the error decreases faster for Adaboost.R2 reweighting method in the first few iterations. The median strategy gives better results on FacePix but it requires up to three times the number of features as compared to other methods. The memoryless procedure has good results and outperforms the AdaBoost.R2 strategy on the pan estimation when images are manually cropped but it tends to overfit on the noisy data (cf. Fig. 4(b)).

The results are better on FacePix than on Pointing 04. This is most probably due to the differences underlined in 3.3 (only pan variation, bias on the cropping and the greater accuracy of the ground truth).

4.3. Regressor architecture

In this section, we experiment other RBF networks. We performed 300 iterations for each regressor on both Pointing and FacePix databases. We kept AdaBoost.R2 weighting since it was found to be the best in the previous section. Test are carried out on two RBF networks which differ on the number of hidden units. RBF 1 has half the number of training examples while RBF 2 has one fifth of this number. Hidden cells are chosen using a K-means algorithm and the covariance matrix of each cluster is only computed at the end of the process. Table 3 shows that performance is directly related to the number of units in the hidden layer. Although the number of units is really large especially in the GRNN, no overfitting was observed. It may be partly due to the spherical radial functions. We have chosen spherical functions because they have only d independent parameters (where d is the dimensionality of the input space) as compared with the $d(d+3)/2$ independent parameters of a full Gaussian basis function. Moreover, training process with RBF is sometimes unstable since initial centers are randomly specified.

4.4. Multiresponse regression

In this experiment, pan and tilt were jointly trained. To achieve this, we slightly modified the FFC. The output is a vector and we use the L_1 norm instead of the absolute difference of the outputs in Eq. (6). Learning simultaneously pan and tilt information seems, a priori, more difficult since the network has to learn two responses instead of one. However, the two pieces of information are interlinked. Table 4 shows that the joint learning performed slightly better on pointing 04. It can also be observed that the joint learning needs to extract a smaller number of features.

Table 3
Mean absolute error in degree on the test set for different regressor architectures. The value in brackets indicates the corresponding number of features.

	Pointing 04		FacePix
	Pan error	Tilt error	Pan error
GRNN	**11.4**° (288)	**12.3**° (120)	**6.0**° (265)
RBF 1	14.0° (217)	15.8° (256)	7.2° (120)
RBF 2	15.2° (161)	16.0° (284)	8.5° (237)

5. Comparison with other methods

5.1. Convolutional Neural Networks

Convolutional Neural Networks (CNN) are bioinspired (Hubel & Wiesel, 1962) multilayered neural networks and were first proposed by Le Cun, Boser, Denker, Howard, Habbard, Jackel, and Henderson (1990) for handwritten character recognition. The idea is to build an architecture using both local receptive fields with share weights and subsampling. In a typical architecture, there are convolutional layers where each unit receives input from a set of cells located in a small rectangular neighbourhood in the previous layer. With local receptive fields, cells can extract basic features (oriented edges for example). Layers extracting feature are called convolutional because of the weight sharing: all units in this layer perform the same linear operation (before applying the sigmoid function) and the process can be viewed as a simple convolution. These features are then combined by the subsequent layers in order to detect high-order features. Before this new extraction, the network reduces the resolution of the feature maps (by averaging and subsampling). This reduction has two justifications: to reduce the size of the layer and to bring some robustness against small distortions. Fig. 5 describes the CNN architecture used in our experiments, following Simard recommendations (Simard, Steinkraus, & Platt, 2003).

CNN can be trained with the classical backpropagation algorithm and we have used an adaptive learning rate and a momentum regularization. Table 5 summarizes comparative results between BISAR and CNN. It is worth noting that performance are quite the same on FacePix, but BISAR outperforms CNN on Pointing 04.

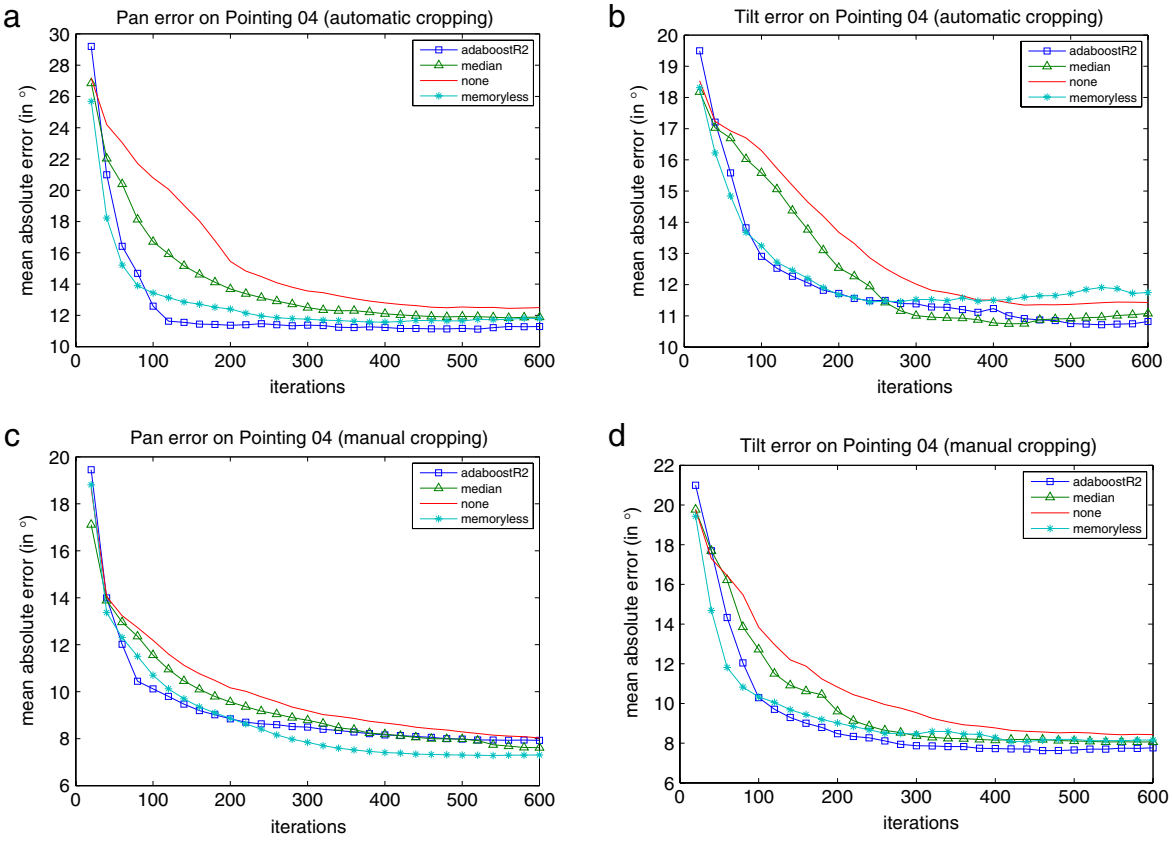

Fig. 4. Comparison of boosting effect against iterations.

Table 4
Mean absolute error in degree on the test set. The value in brackets indicates the corresponding number of features.

	Manual cropping		Automatic cropping	
	Pan error	Tilt error	Pan error	Tilt error
Joint learning	**7.5**° (600)	8.0° (600)	**10.3**° (294)	**12.0**° (294)
Separate learning	8.2° (382)	**7.6**° (457)	10.9° (430)	12.3° (120)

Fig. 5. Architecture of the Convolutional Neural Networks.

Table 5
Comparison with convolutional neural networks.

	Pointing 04		FacePix
	Pan error	Tilt error	Pan error
BISAR	**7.3**°	**7.6**°	4.5°
CNN	8.7°	11.5°	**4.2**°

5.2. State-of-the-art methods on Pointing 04

Performance are compared with three standard methods for this database (Murphy-Chutorian & Trivedi, 2008). The first one proposed by Voit, Nickel, and Stiefelhagen (2007) uses a multilayer perceptron (MLP) to estimate the pose. First, head is localized using a linear boundary decision classifier to detect skin color. Head area corresponds to the bounding box surrounding the biggest