Connectionnists Methods for Human Face Processing

Emmanuel Viennet[1] and Françoise Fogelman Soulié[2]

[1]LIPN, Université Paris 13, 93430 Villetaneuse, France.
viennet@lipn.univ-paris13.fr
[2]Atos Ingéniérie Intégration,
1 avenue Newton, bp 207, 92 142 Clamart cedex, France.
ffogelman@atos-group.com

Abstract. We show in this paper how Neural Networks can be used for Human Face Processing. In Part I, we show how Neural Networks can be viewed as a particular class of Statistical models. We introduce learning as an estimation problem (1), then describe Multi-Layer Perceptrons and Radial Basis Function networks (2), widely used Neural Networks which we will use in Part II, for face processing. We further present Vapnik's framework for learning (3), show the capacity/generalization dilemma and discuss its implications for Neural Network training and model selection. Vapnik's ideas lead to a new interesting class of classifier, Support Vector Machines, presented in section 3.2. We then discuss the combination of models (4) and give a formalism which allows to cooperatively train multi-modular Neural Networks architectures. Finally, we present a multi-modular architecture to perform "Segmentation-Recognition in the loop" (5).

In Part II, we show how the presented models can be applied to build an efficient face localization and identification system. The face images are detected by scanning the scene with a retina feeding a hierarchical coarse-to-fine classifier. Detections are then identified in a small family of known persons.

Part I: Introduction to Connectionist Methods

Recent developments in the Neural Network (NN) field have shown the deep theoretical links between NN and Statistics [20, 41]. In practice, developing a NN requires the same care as for a statistical model. Some authors [37] have even proposed an equivalence list (table 1) trying to show that the differences between NN and statistics could be but a matter of vocabulary only.

In a way, *Neural Networks are indeed simple statistical models*: there are actually formal equivalences between some NN and conventional statistical techniques, such as Principal Component Analysis, Discriminant Analysis, Projection Pursuit, Ridge Regression...

On the other hand, *NN are not only statistical models*: they can be expressed in a unique formulation, which embodies both linear and non-linear models; it is easy to incorporate a priori knowledge into a NN architecture; NNs can be combined into complex multi-modular architectures. Yet, NN are by no means systematically better than statistical models: there is no "universaly best" model and one should always

Neural Networks	Statistics
learning	estimation
weight	parameters
knowledge	parameters value
supervised learning	regression / classification
classification	discrimination
non supervised learning	density estimation / clustering
clustering	classification / taxinomy
neural network	model
"large": 100 000 weights	"large": 50 parameters
learning set	sample
"large": 50 000 examples	"large": 200 cases

Table 1: Glossary: Neural Networks vs. Statistics.

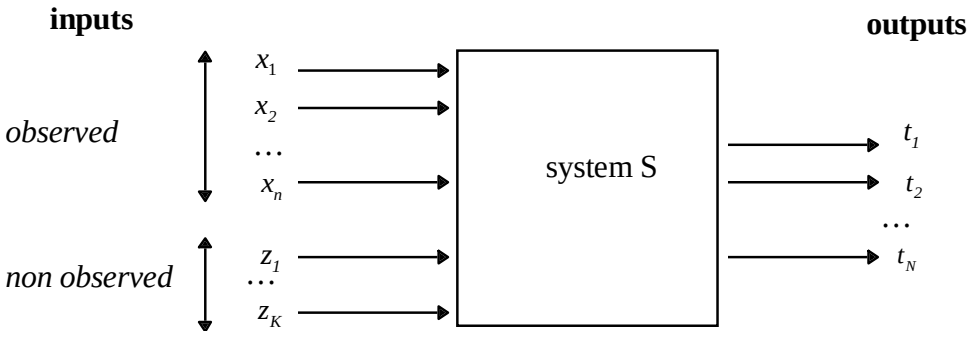

Figure 1: Input-output system.

compare various techniques in order to achieve the best performances on his problem.

In this paper, we will introduce some elements of the general statistical framework of NN, which we will then apply to face processing in Part II. The interested reader can refer to specialized books for more detailed presentations [5, 14, 37].

1 The problem

We will concentrate here on NNs trained through supervised learning.

1.1 The statistical model

Let us consider 3 random variables X, T and Z, of dimension n, N and K respectively, and an *input-output system* S (fig. 1), which, from inputs X and Z produces output T [20].

The system implements the mapping ("true" but unknown):

$$t_k = g_k(x_1, \ldots, x_n;\ z_1, \ldots, z_k) \quad k = 1, \ldots, N \tag{1}$$

However, since only inputs x can be observed, system S is modelled through a statistical model:

$$t_k = f_k(x_1, \ldots, x_n) + \epsilon_k \quad k = 1, \ldots, N \tag{2}$$

where ϵ_k is a random variable with (unknown) distribution P_ϵ, which models our ignorance of the non observable variable Z.

The problem consists in estimating function f_k.

1.2 Estimation

Let us suppose that we are given a *sample* D, of m observations of variables (X, T). From now on, for the sake of simplicity, we will assume that T is of dimension 1: $N = 1$. We denote f the function f_1 and $P(x, t)$ the joint distribution of (X, T):

$$D = \left\{(x^1, t^1), \ldots, (x^m, t^m)\right\} \tag{3}$$

We look for an estimator $\hat{f}$ of f. One can use $\hat{f}$ for:

1. *prediction / classification*: for any new data vector $x = (x_1, \ldots, x_n)$, one wants to predict what the output of system S will be: $t = \hat{f}(x)$.

   In this case, one will try to minimize the approximation error, i.e. $f(x) - \hat{f}(x)$.

2. *interpretation*: one wants to use $\hat{f}$ to understand the structure of system S.

NN are used mostly for prediction/classification, while statistical models are more often targetted at interpretation.

Let $L[t, g(x)]$ be the *loss function* which measures the cost resulting from replacing function f by function g. The risk R is defined as:

$$R(f, g) = \int L\left[t, g(x)\right] dP(x, t) \tag{4}$$

Risk R is the criterion used to compute estimator $\hat{f}$: $\hat{f}$ is chosen, from the set of all possible functions g, as that function which minimizes the risk:

$$\hat{f} = \arg\min_g R(f, g) \tag{5}$$

In practice, one cannot compute R since distribution P is unknown. One must thus use either of two methods:

- *density estimation*: estimate P, then replace it by its estimation in 5, and look for estimation $\hat{f}$ solution of 5;
- *regression*: define the *empirical risk*:

  $$R_E(f, g) = \frac{1}{m} \sum_{k=1}^{m} L\left[t^k, g(x^k)\right] \tag{6}$$

  which measures the average loss on sample D and find $\hat{f}$ which minimizes R_E.

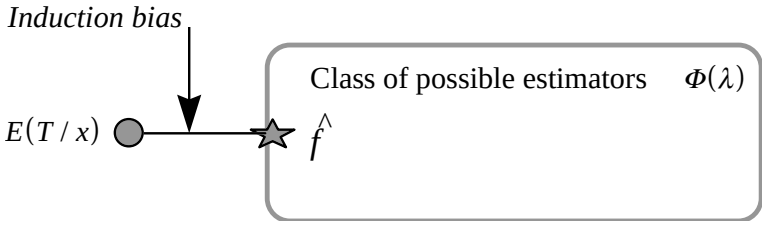

Figure 2: Estimator $\hat{f}$ is searched within space $\Phi(\lambda)$.

For example, if the loss is measured by the square error, R is the *Mean Square Error* criterion (MSE) and R_E the Empirical Mean Square Error (EMSE). That is, one have:

$$\begin{aligned} L[t, g(x)] &= [t - g(x)]^2 \qquad (7) \\ R(f, g) &= \text{MSE} = E\left([t - g(x)]^2\right) = \int [t - g(x)]^2 \, dP(x, t) \\ R_E(f, g) &= \text{EMSE} = \frac{1}{m} \sum_{k=1}^{m} \left[t^k - g(x^k)\right]^2 \end{aligned}$$

The following property holds true for any function g:

$$\begin{aligned} E\left([T - g(x)]^2\right) &= \int [t - g(x)]^2 \, dP(x, t) \\ &= E\left([T - E(T/X)]^2\right) + [E(T/X) - g(X)]^2 \end{aligned}$$

Hence, that estimator which minimizes R in the least mean square sense is:

$\hat{f}(x) = E(T/X)$

i.e. the conditional expectancy of T given X is the optimal estimator. Let us notice that, in the case where T is a discrete variable (e.g. in classification):

$\hat{f}(x) = P(T/x)$

which is the a posteriori probability of class T, given x.

1.3 Search Space for the Estimator

In practice, the estimator $\hat{f}$ is constrained to be part of a given class of functions $\Phi(\lambda)$ (where λ is the index of functions in $\Phi(\lambda)$, not necessarily a parameter). Then, when solving 5, one is restricted to searching space $\Phi(\lambda)$ and thus is not guaranteed to be able to reach the optimal solution $E(T/x)$: there is a systematic induction bias (fig. 2).

One should expect estimator $\hat{f}$ to be consistant: if class $\Phi(\lambda)$ grows, then $\hat{f}$ should become optimal.

In *non-parametric estimation*, there is no a priori assumption on the form of functions in $\Phi(\lambda)$. This is the case for example for *k-nearest neighbor* models (k-nn) or *Parzen windows*: λ is then a parameter equal to $1/k$ for k-nn and $1/\sigma$ for Parzen windows of width σ.

In *parametric estimation*, functions in $\Phi(\lambda)$ are of a given form, parameterized by a (vector) parameter W: $\Phi(\lambda, W)$. This is the case, for example, of *Gaussian mixtures*

(λ is the number of Gaussians in the mixture and $W = (\mu, \sigma)$ is the parameter of the Gaussian), or of Multi-Layer Perceptrons (λ is the number of hidden neurons, W the weight vector of the network).

In this case, the estimation problem is decomposed into two successive steps:

1. *data approximation*: data are fitted by choosing the optimal parameter W:

$$\begin{aligned} W^* &= \arg\min_W \frac{1}{m} \sum_{k=1}^{m} \left[t^k - g(x^k; W; \lambda)\right]^2 \\ \hat{f}(x;\lambda) &= g(x; W^*; \lambda) \end{aligned} \tag{8}$$

2. *model selection*: one compares various families $\{\Phi(\lambda), \lambda\}$ and chooses an "optimal" value of λ (we will see later how).

1.4 The Bias / Variance Dilemma

We have denoted D the data sample $\left\{(x^1, t^1), \ldots, (x^m, t^m)\right\}$ which allowed us to compute the estimator $\hat{f}$ of f. $\hat{f}$ thus depends upon sample D, which we will denote $\hat{f}(x; D)$. Let us introduce this dependency in equation 7. We have:

$$\begin{aligned} E\left(\left[T - \hat{f}(X;D)\right]^2 / X, D\right) &= \int \left[T - \hat{f}(X;D)\right]^2 dP(x,t) \\ &= E\left(\left[T - E(T/X)\right]^2 / X, D\right) + \left[\hat{f}(X;D) - E(T/X)\right]^2 \end{aligned}$$

As we saw before, $\left[\hat{f}(X;D) - E(T/X)\right]^2$ is a measure of the quality of estimator $\hat{f}$. To obtain a measure independant from the particular sample D, we can average this measure on all possible data samples D (of size m). we obtain [21]:

$$\begin{aligned} &E_D\left(\left[\hat{f}(X;D) - E(T/X)\right]^2\right) \\ &= \underbrace{\left(E_D\left[\hat{f}(X;D)\right] - E\left[T/X\right]\right)^2}_{\text{bias}} + \underbrace{E_D\left(\left[\hat{f}(X;D) - E_D\left(\hat{f}(X;D)\right)\right]^2\right)}_{\text{variance}} \end{aligned}$$

The *bias* measures the mean distance of $\hat{f}(X;D)$ to the optimal value $E(T/X)$, whereas the variance measures how this distance varies with D.

A "good" estimator will have a good *accuracy*, i.e. a small bias, and a good *stability*, i.e. a small variance. However, these two objectives are contradictory: when λ parameter increases (for example in the above mentionned families: k-nn, Parzen windows, Gaussian mixtures, MLP), then the bias decreases and the variance increases (fig. 3). λ is thus used to control the bias / variance balance.

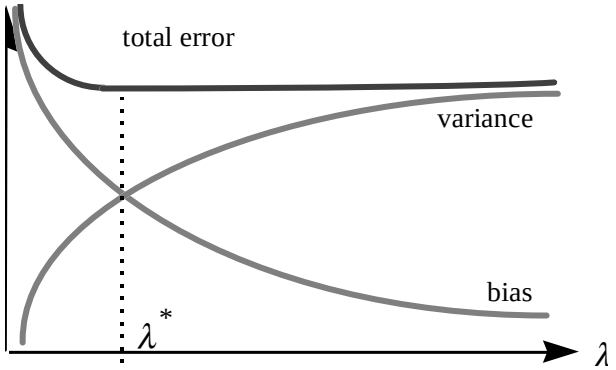

Figure 3: Bias / Variance Dilemma.

2 Multi-Layer Perceptrons

2.1 Neurons

A neuron is an elementary processor characterized by (fig. 4):

- an internal state $s_i \in \wp$;
- input signals $s_1, \ldots, s_n$;
- a state transition function g: $s_i = g(s_1, \ldots, s_n)$.

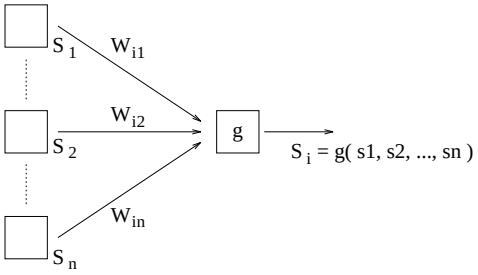

Figure 4: The neuron

There exist various sorts of neurons, in particular:

- *dot-product neuron.* Its transition function is: $s_i = f(A_i)$ with

$$A_i = \sum_{k=1}^{n} W_{ik} s_k \; = {}^t W_i . s \tag{9}$$

where coefficient W_{ik} is called the connection weight from k to i.
The dot-product neuron is thus composed of two successive modules: a linear transformation (the dot-product) followed by a (generally) non-linear transformation f. Function f can be the identity, a threshold function, a sigmoid function, ...

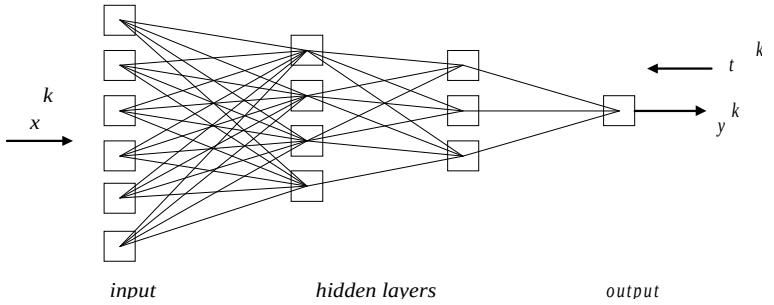

Figure 5: Multi-Layer Perceptron (MLP).

- *distance neuron.* Its transition function is again $s_i = f(A_i)$, here with
$$A_i = \|W_i - s\|^2 \tag{10}$$

The distance neuron is thus composed of two successive module: a distance evaluation module followed by a (generally) non-linear transformation f. Function f can be the identity or a kernel function. Distance neurons compare a *prototype* (their weight vector W_i) to the input signal.

A *Neural Network* (NN) is a set of interconnected neurons. It is fully caracterized by:

- its architecture: the number of neurons and their interconnection scheme;
- the neurons transition functions: weights W_{ik} and functions f.

2.2 The Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) (fig. 5) is a network composed of successive *layers* (a layer is a set of neurons which are not connected):

- an input layer (where the inputs to the system are presented) and an output layer (where the outputs of the system are read out);
- one or more hidden layers.

Each neuron is a dot-product neuron (eq. 9) where function f is a sigmoid function (in general). A MLP implements a transformation:

$$y = F(x_1, \ldots, x_n; W) \tag{11}$$

MLPs can be connected in various ways, depending upon the problem (fig. 6):

- full connections: the neuron receives inputs from all neurons in the previous layer;
- local connections: a neuron receives inputs from only some of the neurons in the previous layer, called its "receptive field". The various neurons in one layer can have receptive fields of the same size, covering -possibly with some overlap- the input layer. This arrangement is similar to having a sliding "window" on the input layer.
- shared weights: all neurons in one layer are locally connected to their inputs through a receptive field of the same size, and their weights are identical. This

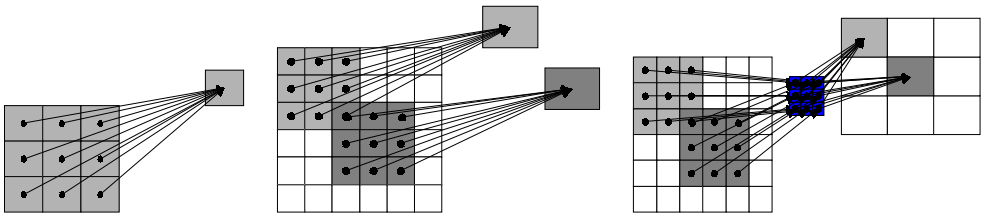

Figure 6: Connections in Multi-Layer Perceptron: full connections (left), local connections (middle) and shared weights (right).

arrangement is similar to having a filter passed through the input layer, the coefficients of the filter being the common weights vector.

It is to be noted that, depending upon the connection scheme, the number of weights can be very different: for an input layer with N neurons and a layer with P neurons, the number of weights on connections between these 2 layers will be $P \times N$ for full connections, $P \times k$ for local connections with a receptive field of size k, and just k for local connections with shared weights.

One particular case of MLP is the non-supervised or auto-associative MLP (fig. 7): its input and output layers have the same number of neurons. The desired output is identical to the input: the MLP is expected to reproduce the input at its output layer. In general, the hidden layer has less neurons than the input or output layers: the MLP thus has to encode the input signal into the hidden neurons, and thus decode it to produce the output.

In the general case where the output layer is different from the input layer, one has a hetero-associative MLP. This is, in particular the case in classification problems: the output layer then has as many neurons as there are classes; for class i, all neurons will be "off" (state 0) except neuron i which will be "on" (state 1).

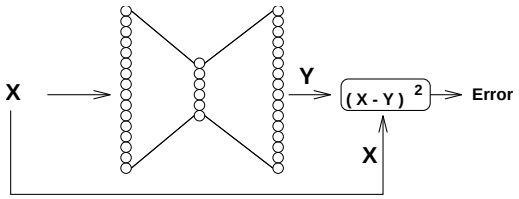

Figure 7: Multi-Layer Perceptron in auto-associative mode.

2.3 Training MLPs

An MLP is "trained" by using a data sample D (3) so as to determine those weights W^* solutions of (8) where $g(x; W; \lambda)$ is the function $F(x; W)$ implemented by the MLP:

$$W^* = \arg\min_W \frac{1}{m} \sum_{k=1} m \left[t^k - F(x^k; W)\right]^2 \tag{12}$$

In order to solve (12), one uses numerical optimization techniques: gradient descent, conjugate gradients, second order methods, ..., which are called, in this context, *learning algorithms*. The most well known algorithm, the *gradient back-propagation* algorithm is based upon the following idea: in any point W, the gradient vector of the empirical risk, ∇R_E, points in the direction of increasing empirical risk. To decrease R_E, one thus has just to move in the opposite direction to the gradient ∇R_E. The back-propagation algorithm is an iterative algorithm, which consists in modifying weights according to the following rule:

$$W_{ij}(t) = W_{ij}(t-1) + \Delta W_{ij}(t) \tag{13}$$

where $\Delta W_{ij}(t)$ is proportional to the opposite of the gradient:

$$\Delta W_{ij}(t) = -\epsilon(t)\frac{1}{m}\sum_{k=1}^{m}\frac{\partial C_E(x^k;W)}{\partial W_{ij}} \tag{14}$$

where $C_E(x^k;W) = \left[t^k - F(x^k;W)\right]^2$.

This algorithm depends upon all the examples, which have thus to be made available for each weight update: this is why it is called the *off-line* algorithm. The *on-line* algorithm performs one weight update after each presentation of an example:

$$\Delta W_{ij}(t) = -\epsilon(t)\frac{\partial C_E(x^k;W)}{\partial W_{ij}} \tag{15}$$

Usually, especially when the data sample D is large, and thus redundant, the on-line algorithm is much faster than the off-line algorithm [29].

In practice, one uses the given data sample D_m as follows: it is separated in 3 sub-samples: $D = D_m^l$ is the training set used in (14) or (15). D_m^v is a validation set used for model selection, i.e. to compare the performances of models (e.g. compare MLPs with various numbers of hidden units). Finally, D_m^t is used to estimate the performances of the selected optimal model on an independent data set which has never been used during training or validation.

One should be careful when training MLPs: learning algorithms must, like every other numerical algorithms, be implemented with care. There are many tricks described in the litterature: one must be cautious and make a clear distinction between theoretical results and the results which are actually achieved after numerical training.

2.4 Radial Basis Function Network

A Radial Basis Function Network -RBF- (fig. 8) is a MLP with one hidden layer of distance neurons. It is trained in much the same way as MLPs, the parameters to be adapted through learning are W_{ij}^1 the parameters of the function used in the hidden layer (10), and W_i the weights from hidden to output layer. For example, if the hidden neurons are Gaussian:

$$F(x_1,\ldots,x_n) = \sum_{i=1}^{p} W_i \exp\left[-\frac{1}{\sigma_i^2}\sum_{j=1} n(x_j-\mu_i)^2\right] \tag{16}$$

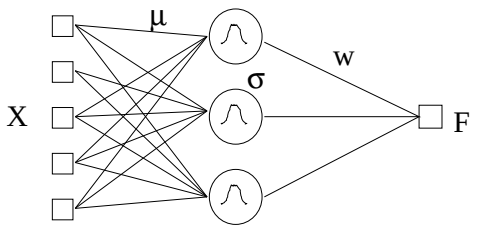

Figure 8: Radial Basis Function Network.

and $W_{ij}^1 = (\mu_i, \sigma_i^2)$.

These parameters are updated by applying (14) or (15) with the derivative of C_E being computed with respect to W_{ij}^1 and W_i. In the same way as nearest neighbor classifiers, RBFs tend to have poor performances on high-dimensional inputs.

3 Vapnik's model and Support Vector Machines

3.1 Risk and Generalization bounds

We have previously defined the risk R and the empirical risk R_E by:

$$\begin{aligned} R(f,g) &= \int L\left[t, g(x)\right] dP(x,t) \\ R_E(f,g) &= \frac{1}{m}\sum_{k=1}^{m} L\left[t^k, g(x^k)\right] \end{aligned}$$

Since $P(x,t)$ is unknown, one cannot compute R nor thus minimize it. In practice, one thus tries to minimize R_E: this is the *minimum empirical risk principle* (MRE). Before accepting such a principle, one should be able to answer the following questions [40]:

- is the MRE principle consistant? i.e. does function $\hat{f}_m$ which minimizes R_E for sample D_m converge to function $\hat{f}$ which minimizes R?
- is convergence fast?

The answers to these questions depend upon the *Vapnik Chervonenkis dimension* (VC dimension) of the class of functions $\Phi(\lambda)$ where the solution $\hat{f}_m$ is looked for. We will not define here the VC dimension (see for instance [39, 11]): suffices it to say that it is an estimation of the class *complexity*. Denoting h this dimension, Vapnik [40] has shown that:

1. the MRE principle is consistant if dimension h is finite;
2. the speed of convergence is: $\sqrt{\ln(m/h)/(m/h)}$;
3. with probability $1-\eta$:

$$R(f,\hat{f}) \leq R_E(f,\hat{f}) + C(m,h;\eta) \quad (17)$$

$$C(m,h;\eta) = 2\sqrt{\frac{1+\ln(2m/h)}{m/h} - \frac{\ln\eta}{m}} \quad (18)$$

These properties, and in particular (18), show that it is reasonable to minimize the empirical risk R_E instead of risk R, *as long as h is finite*. However, there is a systematic error which depends upon m, the size of sample D_m, and h the VC dimension of class $\Phi(\lambda)$. This error is bounded by $C(m, h; \eta)$ (fig. 9). However, Vapnik's result (18) is a worst case analysis: the bound is not very tight.

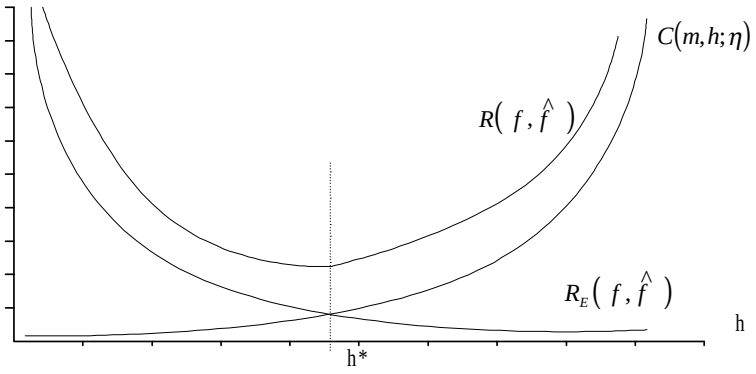

Figure 9: Risk R, Empirical risk R_E and Vapnik's bound C.

The expression of (18) shows that there exists an optimal value h^* of h. If m is fixed, then when h increases:

- the empirical risk $R_E(f, \hat{f})$ decreases;
- the error bound $C(m, h; \eta)$ increases.

This is the usual bias-variance dilemma. As h grows larger, it becomes easier to fit the data and the bias becomes smaller. However, when h grows larger, the estimator depends more on the data and the variance grows larger too. As a result there exists a class of functions $\Phi(h^*)$ of optimal complexity h^* for which $R(f, \hat{f})$ is minimal (fig. 9): h^* must be large enough to fit the data, but not too much with respect to the number m of available data.

Vapnik's results allow to apply the MRE principle, by ensuring a bound on the generalization error $C(m, h; \eta)$. However:

- this bound is not tight: as we have said before, it comes from a worst case analysis;
- the VC dimension is defined through theoretical developments [39]. Its value is known (table 2) for only a few classes $\Phi(\lambda)$, but not for the simplest classes of MLPs.

3.2 Support Vector Machines

Support Vector Machines (SVM) are new classifiers proposed by Vapnik [41] and implementing the Structural Risk Minimization principle.

In the following, we will briefly recall the SVM principles. The interested reader can find more information in the paper of V. Cherkassky in this volume, or in a tutorial by C. Burges [11]. In section 9, we will present a few preliminary results obtained with SVM on our face detection application.

Network	VC dimension	Reference
threshold neurons N neurons, W weights	$h \leq 2W \log_2 eN$	[3]
$k_0 \times k_1 \times 1$ W weights from input to others neurons	$h \geq W + 1$	[2]
$k_0 \times k_1 \times 1$	$h \leq 2W \log_2 eN$	[3]
$k_0 \times k_1 \times k_2 \times 1$ if $k_0 \geq k_1$ and $k_2 \leq \frac{2^{k_1}}{k_1^2/2+k_1/2+1}$: if $k_1 > k_0 > 1$ and $k_1 \geq k_2$:	$h \geq k_0 k_1 + 1 + k_1(k_2 - 1)$ $h \geq k_0 k_1 + 1 + \frac{k_1(k_2-1)}{2}$	[2]

Table 2: Vapnik Chervonenkis Dimension for some classes of MLPs. The notation $k_0 \times k_1 \times s$ represents a MLP with k_0 input neurons, k_1 hidden neurons, and s output neurons. $k_0 \times k_1 \times k_2 \times s$ is a MLp with 2 hidden layers with k_1 and k_2 neurons.

Introduction When building a pattern classifier, one wants to minimize the *risk* R or generalization error. During learning, we can only measure the *empirical risk* R_{emp}, the error on the training sample. We mentionned above (17) a bound on the risk functional, which can also be written as follows: with probability $1 - \eta$,

$$R \leq R_{\text{emp}} + C(l/h, \eta) \tag{19}$$

where C depends on the Vapnik-Chervonenkis dimension h of the set functions implemented by the classifier and the size l of the training sample. C gives a confidence interval on the generalization error, knowing the error on the learning sample.

Empirical Risk Minimization (ERM) strategies [41] aim to minimize the bound on R by minimizing the first term R_{emp} of eq. 19. Neural Networks are good examples of ERM application. In order to control the generalization (C), a classifier with small h must be chosen. This can be achieved by data preprocessing [22], regularization techniques, or network architecture selection.

SVM implements another strategy : keep the empirical risk fixed ($R_{\text{emp}} = 0$) and minimize $C(l/h, \eta)$. A SVM is a two-class classifier which simply computes $f(x, w, b) = \text{sign}(w.x + b)$ If the set $(x_i, y_i), i = 1, ..., l$ (where $y_1 = \pm 1$ gives the class of each example) can be separated without errors by f, one can show that the VC dimension h of the family of functions f is bounded by

$$h \leq \min(R^2 A^2, n) + 1 \tag{20}$$

where n is the dimension of the input space, R is the radius of the smallest sphere enclosing all the examples x_i, and A a constant such that $\|w\| < A$.

Thus, if we minimize $\|w\|$, we also minimize the bound on the generalization error $C(l/h, \eta)$ in eq. 19. Minimizing $\|w\|$ while separating the classes has a convincing geometric interpretation : it leads to the hyperplane whose distance to the closest example (the so-called *margin*) is maximal.

Optimal Hyperplane The problem stated above can be formulated as follows : minimize $\Phi = \frac{1}{2}\, w.w$ subject to the constraints : $y_i.f(x_i, w, b) \geq 1,\ i = 1...l$. By expressing the Lagrangian $L(w, b, \alpha) = \Phi - \sum \alpha_i \{f(x, w, b) - 1\}$, it can easily shown [41] that the solution w_0 is a linear combination of the examples x_i : $w_0 = \sum y_i \alpha_i\, x_i$. Lagrange multipliers $\Lambda = (\alpha_i)$ are maximizing the functional

$$Q(\Lambda) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j x_i.x_j \tag{21}$$

under the constraints : $\sum_{i=1}^{l} \alpha_i^0 y_i = 0$ and $\alpha_i \geq 0$.

All constraints which are *active* in the solution lead to zero multipliers, $\alpha_k = 0$. Thus, only a few patterns appear in expression of w_0. They are called *support vectors* (SV).

The derivation above can be generalized to non-linear separators simply by changing the scalar product $u.v = K(u, v)$. Among useful scalar products, we decided to use the polynomials of degree d : $K(u, v) = (\frac{u.v}{n})^d$ which allow to draw SVM non-linear frontiers between classes.

Non separability If the training set is not separable by the hyper-surface, the problem given by eq. 21 has no solution. Vapnik [41] suggests to use a modified formulation which leads to maximize the same functional $Q(\Lambda)$ under one more constraint $\alpha_i \leq C$. The constant C prevents divergence of Q and controls the capacity of the classifier. The choice of C is done empirically.

Optimization technique In order to implement the SVM, we developed an optimization method based on the algorithm proposed by More and Torraldo [30]. This algorithm is oriented to solve large quadratic programming problems with bound constraints. Given a quadratic function $q : I\!R^n \rightarrow I\!R$, the problem is to find

$$\min_x \{q(x) :\ l \leq x \leq u\}, \tag{22}$$

where $l \in I\!R^n$ and $u \in I\!R^n$ define the feasible region. The idea of the algorithm is to use a conjugate gradient to search the face of the feasible region defined by the current iterate and the projected gradient method to move to a different face. It is proved that for strictly convex problems the algorithm converges to the solution in a finite number of steps.

To build a SVM, we have the minimize the quadratic form $Q(\Lambda)$ under bound constraints $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{l} \alpha_i^0 y_i = 0$. We can use a penalization technique to express this problem in the form of equation 22. This introduces a term $\frac{\gamma}{2} \left(\sum_{i=1}^{n} \alpha_i y_i\right)^2$, $\gamma > 0$ and we define q as

$$q(\Lambda) = \frac{1}{2} \Lambda' (D + \gamma F) \Lambda - \sum_{i=1}^{l} \alpha_i, \tag{23}$$

where $D_{ij} = y_i y_j K(x_i, x_j)$ and $F_{ij} = y_i y_j$ are positive matrices. As γ grows, the solution converges to the solution of the original problem (eq. 21).

Incremental optimization During optimization with l examples, we have to store a l by l matrix. In our case, the total number of examples (faces + backgrounds) reach 50 000, so we have to optimize incrementally the SVM, using the following procedure:

1. Let A be of set of N_a patterns, initialized randomly from the training set, and B the set of the remaining examples.
2. Optimize the SVM on A. A subset of the patterns of A is selected as support vectors and kept in A, while the other examples, for which $y_i.f(x_i, w, b) > 1$ are moved to B.
3. Select some patterns from B such that $y_i.f(x_i, w, b) \leq 1$, and put them in A.
4. Repeat from step 2, until convergence.

At the end of this procedure, A contains the SV. Note that N_a must be larger that the total number of SV. This number is related to the generalization error rate [11, 41].

4 Models Combination

It very often happens that we have a family of models to choose from: one can try to *select* the best one for his particular problem, or to *combine* them all. When combining models, one hopes that the errors of one model will not be done by another, so that the overall performances of the combination will be improved. There exist various ways to combine models.

- *ensemble*: all models are computed *in parallel* and the various estimations produced are then combined, e.g. by taking their average [23];
- *stacking*: all models are computed in sequence and the final estimation is the result of the last model [45, 9];
- *task decomposition*: one sequence of models is executed, depending on the input. Input space is decomposed in various "zones", each corresponding to a particular sub-task: the corresponding sequence of models is the "expert" dedicated to solving that task. The decomposition can be done by hand or obtained through training [25].

These combination methods can be used to build complex multi-modular architectures, which embody the knowledge available about the particular problem at hand.

A technique was introduced [6, 7] to train such multi-modular architectures. Let us suppose that we have an architecture composed of $M_1, \ldots, M_N$ interconnected in such a way that their interconnection graph has no cycle.

One can show that it is possible to train such an architecture through an algorithm very similar to the Gradient Back-propagation algorithm. In practice, each module is trained separately and then the whole architecture is trained cooperatively. This technique has proved very efficient in many applications [4, 42, 19].

5 Segmentation - Recognition

In many image processing problems, objects must be first localized and then identified. This is the case for example in Optical Character Recognition (OCR) and of course

in face recognition, but it occurs in speech processing as well. The two problems are heavily inter-twinned: one can easily segment a character which he has already identified, but isolating an object without knowing what it is is much harder. Hence the idea of "segmentation-recognition in the loop", which has been first proposed in the OCR domain [27, 28].

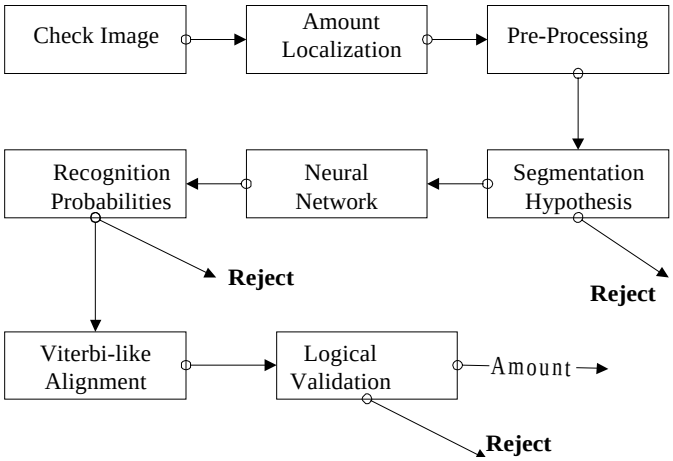

Figure 10: Segmentation-Recognition in the Loop.

For example, we have developed an OCR application for check-reading [18] with a succession of steps (fig. 10): a segmentation hypothesis is generated which is then passed for recognition to a multi-modular architecture. The result of this recognition is scored by a Viterbi algorithm: and the result with best score is chosen as the final result. The process of segmentation-recognition is thus iterative: if the score is not high enough, further segmentation hypothesis can be generated.

The multi-modular architecture has two modules (fig. 11):

- an MLP for feature extraction: the MLP is an auto-associative MLP (see fig. 7);
- an RBF for recognition: the RBF uses as input the "code" extracted in the hidden layer of the MLP.

Such an architecture is very efficient: the MLP reduces the dimensionality of the input, which allows good performances of the RBF. When the two modules have been separately trained, they are further cooperatively trained, along the process described in section 4.

Part II: Localization and identification of faces in indoor images

We have just shown that Statistics provide a theoretical framework well suited to introduce Neural networks architectures and training methodology. Multi-modular

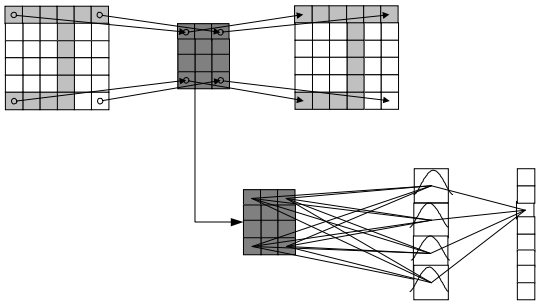

Figure 11: Multi-Modular Architecture for OCR.

architectures and "segmentation-recognition in the loop" can be used to solve problems of image processing, involving complex objects. We will now see how these techniques can be applied to Face Processing.

6 Introduction

During the last decade, a lot of research has been done on automatic face processing, leading to thousands of publications. We can see two main reasons: first, face processing is a nice academic problem, for which no satisfying solution exists but which seems tractable: there is room for a lot of improvements of current technologies. Second, there is pressure from companies wishing to develop commercial applications. Among the major applications of face processing systems, let's cite automatic surveillance, access control, teleconferencing systems, indexing, etc.

In this paper, we present a system allowing to count and identify people in a scene image. This system, developed during the last five years, is suitable for surveillance tasks and offers state of the art performances. We discuss the methodology used and compare various approaches for face/background discrimination: face-space unsupervised modelization, discriminant neural networks, hierarchical classification system. Finally, we present recent developments using Vapnik's Support Vector Machines classifiers.

6.1 Face localization methods

There are so many teams working on face processing systems that it would be difficult to cite all relevant publications in a few pages. The interested reader can find a good review in [13].

Face localization algorithms can be roughly divided in tow classes: feature-based and face-based methods. Feature based methods [15, 24, 32] looks for individual components of the face, such that eyes, mouth, face outline, and so on. The main problem with these methods is their lack of robustness: some features are often missing in faces images due to lighting conditions or hidden parts.

Face-based methods try to build global characterization of a face image, allowing to tell if an image is or not a face image. Such methods are expected to offer more

robustness. They usually do not depend on explicit a-priori knowledge about faces, but make use of a set of example images (*training set*) to extract model parameters.

We proposed a few years ago [8, 43, 19, 42] to use Space Displacement Neural Networks architectures (see section 2.2) with supervised training for face detection. More recently, Rowley et al. at CMU developed a face detection system based on similar principles [33, 34].

In the following, we present various improvements of our previous system, using new hierarchical architecture. We will show how an efficient face detector can be built using several multi-modular classifiers in cascade.

6.2 The application

The goal of the application described in this paper is to count and identify the people in a scene image. We restrict the search to faces looking approximately towards the camera (with a tolerance of $\pm 40°$ in rotation). The detected faces is identified in a group of previously known persons (a *family*), or detected as unknown.

The input of the system is a digitized indoor scene image (760x580 pixels, 256 grey levels). The distance to the camera is unconstrained, the illumination and apparent size of faces are very variable. Generally speaking, the images are of poor quality : low resolution and contrast on faces.

We gathered a database of 5000 scene images, including around 6500 faces from 40 distinct persons[1]. For all the experiments described in this paper, we have used 4000 scene images to train the systems, and the remaining 1000 to evaluate the performances. All scenes have been manually labeled, registering the coordinates of boxed faces (as in figure 12).

As usual in detection problems, two kinds of errors may occur :

- *Non detection* (false negative) : face not detected;
- *Alarm* (false positive) : detection of an object which is not a face;

For any detection system, the decision usually depends on a threshold parameter Θ. All results should then be presented using "detection curves", plotting the non detection rate versus false positive rate.

6.3 System's overview

Figure 13 gives an overview of our system. As mentioned above, we adopt a global approach, and don't introduce in the algorithms any a priori knowledge about faces. In principle, the presented methodology can be applied to other object detection tasks [16].

In order to detect all faces with a single fixed-size classifier, we decompose the scene image on a multiresolution pyramid. Such a pyramid can easily be computed using a low pass filter (smoothing) associated to a sub-sampling operation (figure 14). Each resulting image is scanned by the classifier's retina. Large faces will be detected in the lower levels, while small ones will be detected in the upper ones.

[1] Unfortunately, this database is part of an industrial project and is currently not publicly available.

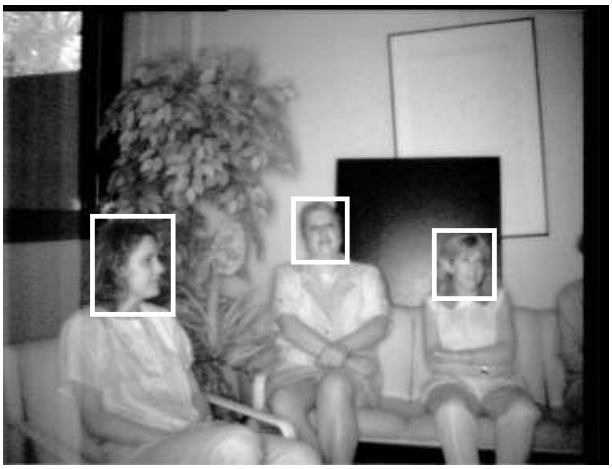

Figure 12: A labeled scene. Note that the detection should be tolerant to strong rotations (leftmost face).

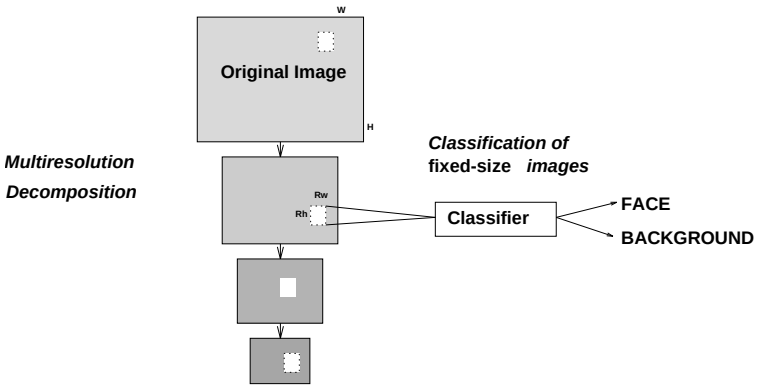

Figure 13: The input image is decomposed in a multiresolution pyramid; each level of the pyramid is scanned by the retina of a classifier.

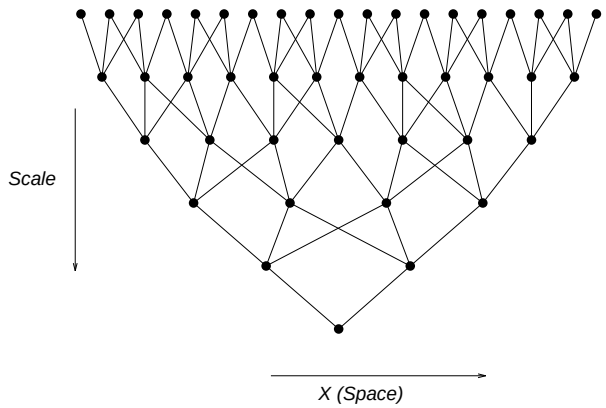

Figure 14: A multiresolution pyramid can be built using filtering and sub-sampling. For simplicity, the operation is presented here in one dimension. Points of S_i are given by local averaging of the preceding level S_{i-1}, allowing fast computation. Index i can be seen as a *scale* parameter.

7 Modelization versus Discrimination

In this section, we briefly discuss two usual ways to image detection: unsupervised modelization and supervised classification.

The (huge) set of all possible face images is called the "face space". Linear Principal Component Analysis and variants have widely been used for face detection. Turk et al. [38] proposed to use a variant of Principal Component Analysis (PCA) to modelize the face space. After computing the PCA on a training set of face images, the distance from a new image to the face space can be estimated.

These approaches suffer two weaknesses: first, the image is considered as a one dimensional vector (pixels values), so the computation does not take advantage of the bidimensional structure of the face (correlations between neighboring pixels). Second, the estimation is *unsupervised*: the face space is modelized using only examples of face images. As we will see in this section, a discriminant system trained using examples of faces and non-faces images can obtain much more accurate results.

7.1 PCA and auto-associative networks

It is well known that PCA can be implemented by an auto-associative multi layer perceptron with one hidden layer and linear transfer functions [1], as represented in figure 7. MLPs with sigmoidal transfer function are expected to slightly improve PCA. In fact several authors used auto-associative MLP networks for face detection (e.g. [17]).

In order to check the performances of this approach, we trained an auto-associative network on a set of 5000 faces. Faces are normalized to 20x25 pixels and, after a few tries, we choose a hidden layer has 48 cells.

Results measured on a separate test set (1500 faces, 100000 backgrounds images)

are presented in figure 16. From the left curve, we can see that the faces images are better reconstructed that backgrounds. There's a significant overlap between the two distributions. The detection curve (right) shows that to get 1 % of non detected faces, we have to accept about 17 % of false positives. It seems that auto-associative systems are unable to deal with complex backgrounds.

PCA-like methods basically compute the distance between the input pattern and its projection on the subspace $\mathcal{S}$ spawned by principal axis. This subspace is estimated using examples of faces only, but may contain also other patterns, as illustrated by figure 15.

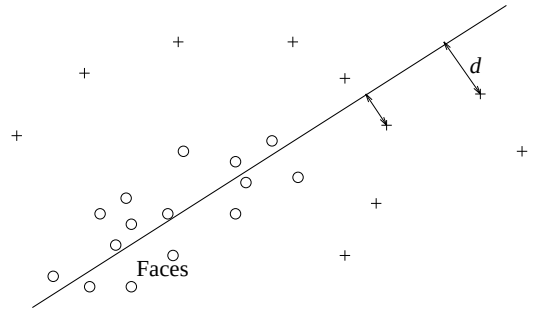

Figure 15: Why PCA doesn't work for detection: PCA subspace is the line, estimated using examples of faces (circles). The detection is then based on the distance d from a point to the line. Background images (crosses), even far from faces region, can lie very close to the subspace.

Constraints on the model must be added by using counter-examples, leading us to discriminant systems.

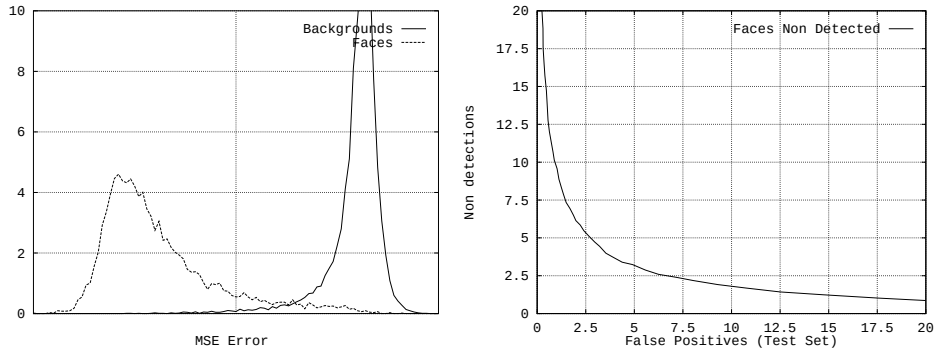

Figure 16: Results of auto-associative system. Left: reconstruction errors repartition (on test sets); right: corresponding detection curve.

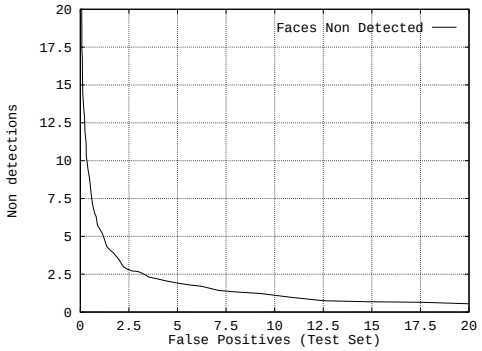

Figure 17: Detection curve obtained after supervised MLP training.

7.2 Discriminant systems

To learn a discriminant classifier, one needs a set of patterns of each class. For a detection problem, this means that we have to collect a set of background (non face) images. This set is virtually unlimited, so we have to design a strategy to collect relevant counter-examples.

In the following, we will respectively denote by N_{faces} and N_{back} the numbers of faces and non-faces (background images) examples.

A simple MLP classifier Just to put in evidence the gain obtained by using a supervised training scheme, we did a very simple experience: we *randomly* selected $N_{\text{back}} = N_{\text{faces}} = 5000$ background images and trained a classifier on the obtained examples. Here again, we normalized the images to 20x25 pixels. The classifier is a multi-layer perceptron with one hidden layer of 48 cells, 2 output cells (one per class). All layers are fully connected.

Figure 17 shows the detection curve measured on the test set. For 1 % of non detected faces, we get approximately 10 % of false positives. This is obviously a poor result, but can be compared to the 17 % of false positives obtained by the auto-associative network (figure 16). The discriminant system is clearly better, but in order to improves its performances we have to improve the selection of counter examples.

Adding more examples: modifying the cost function Before looking at more sophisticated ways to find counter-examples, let's try to add more random background images.

During learning, the average error is minimized. This error (or *cost*) for a pattern from class i is

$$E_i = \frac{1}{2} \|o - t_i\|^2$$

where o is the NN output, and t_i the desired output for class i. If the number of examples in a class is much larger than is the others classes, the training will be

biased towards this class. This problem can be solved by weighting the cost :

$$E'_i = \frac{N}{4N_i} \|o - t_i\|^2$$

where N_i is the number of patterns in class i, and $N = \sum_i N_i$ the training set size. Thus, each class contributes equally to the total error, which is minimized during learning.

Experimental results are summarized in table 3 (The set of faces is the same as before, with $N_{\text{faces}} = 5000$).

NN	N_{faces}	N_{back}	Non detection	False Positive
MLP auto-associative	5000	0	1 %	17.5 %
MLP classif	5000	5000	1 %	10.0 %
MLP classif	5000	40 000	1 %	14.3 %
MLP *cost*	5000	40 000	1 %	6.5 %

Table 3: Results obtained by various simple systems, measured on the same test set.

Adding more counter-examples without modifying the cost function degrades performances. Training with a modified cost allows to reduce the false positive rate down to 6.5 %.

Remember that the classifier will be applied in each position of the multiresolution decomposition of the scene to analyze. In our case, this leads to around 100 000 calls by scene. A false positive rate of 6.5 % per call gives 6500 objects detected in the scene!

8 Hierarchical system and Incremental learning

8.1 Incremental selection of examples

Various incremental strategies have been proposed in the literature [44, 33], based on the idea of incrementally re-inject false positives in the training set during the learning procedure. The main idea is:

1. choose a classifier (e.g. NN architecture);
2. let F be the set of faces examples;
3. randomly select a set B of non face examples;
4. Repeat :
   - supervised training using F and B;
   - collect false positives on scenes images and add them to B;

   Until false positive rate $< \epsilon$.

This kind of approach raise several problems :

- Size of B grows (fast), since we only add examples. Heuristics may be used to limit the number of non faces examples, by selecting only counter examples "far" from the existing ones.
- After each step of the procedure, the classification problem gets harder ("hard" false positives are selected). This is the most serious problem. From modern learning theory [41], we know that for each given problem, one has to choose a model with a complexity related to the training set. The "complexity" of a model can be quantified by its *capacity* (or VC dimension, see section 3). After each modification of the training set, the architecture of the NN classifier should be adjusted accordingly, in order to increase its capacity.
- To get good performances, one has to choose a classifier able to solve the "harder" problem, leading to large computing times.

8.2 Hierarchical approach

A modular hierarchical system can solve the problems mentioned above. The main principle is illustrated by figure 18. Each classifier evaluates the probability that the presented image is a face, *given* that all previous classifiers consider it's a face. If the probability is below a threshold, the image is rejected as "non face". The classifier at

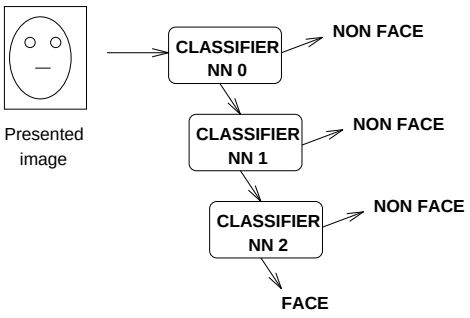

Figure 18: A 3-level hierarchical detection system.

level i is trained using only patterns (faces and non faces) selected at previous levels, in a fashion similar to incremental selection schemes discussed above. After each stage, the classification problem gets harder: only background images looking like faces are selected. The complexity of each level can be adjusted accordingly. Here, processing time considerations does not limit the choice, since the vast majority of the images will be filtered by the first classifier, which can be very quick, as we will show in next section.

8.3 The 3 stages system

In this section, we present our face detection system based on 3 hierarchical classifiers. In preliminary experiments (section 7), we used only simple multi-layer perceptrons (MLP). This kind of architectures are clearly not optimal for image classification problems. As mentioned above, they don't take advantage of the 2D structure of the input pattern.

A classifier is classically decomposed in two parts: extraction of features, and classification. For face detection, extracted features may be "low level" or "high level". Low levels features are computed by standard image processing techniques: filtering, edge detection, linear data analysis and so on. They do not carry specific knowledge about the object to detect. In contrast, high level features need a model of the object. For faces, we could use eyes positions, mouth, hair outline, etc. As stated above, such high level features are often difficult to use in real (noisy) images.

Some connectionnists architectures allow to integrate low level feature extraction with the classification process. A well known example is the Space Displacement Neural Networks (SDNN) "LeNet" proposed by Le Cun [26] for handwritten digits recognition. SDNNs use local connection patterns and shared weights (see figure 19).

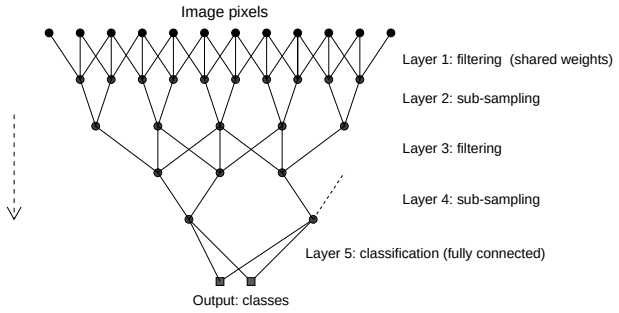

Figure 19: Part of SDNN LeNet, shown in one dimension. Processing goes downwards. This architecture uses 4 layers of "feature extraction", alternating filtering and sub-sampling to reduce dimension of the data. Last layer can be seen as a quasi-linear classifier of the features. Each set of similar weights share the same value, giving translation-invariant processing.

MLP-RBF architectures for image detection As explained in section 2.2, MLP networks neurons compute their activies using dot-product neurons, leading to a non local behaviour. The obtained class frontiers are thus unbounded, as illustrated by figure 20. Even after an extensive training procedure, our image classifier will have to handle lots of *outliers* patterns, very distinct from all previously seen images. The RBF networks, as stated in section 2.4, are based on distance computations and localized kernels, naturaly leading to bounded activity regions: output on outliers patterns is zero.

The problem with RBF networks (and with all distance-based classifiers) is that they poorly perform on high-dimensionnal inputs (the so-called curse of dimensionality). These classifiers are usually used on low dimensionnal *features*, extracted by some algorithm, instead of raw data. We propose another solution : use a multi-modular connexionnist architecture, with a MLP module for feature extraction and dimensionnality reduction, and a RBF module for classification, taking the best of

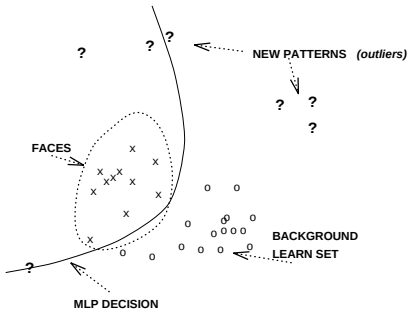

Figure 20: Multi Layer Perceptrons (MLP) separate classes using unbounded hypersurfaces.

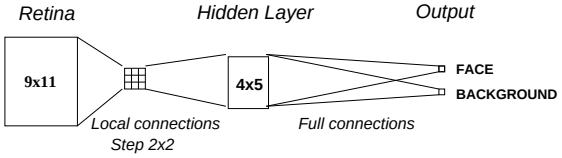

Figure 21: MLP for first stage: use a single hidden layer with local connections to the small retina.

both systems. This architecture is very similar to the one presented in section 5 for digit recognition.

Description of architectures Let's now describe the classifiers used for our 3-stage hierarchical system:

1. *First stage* is applied to every position of the multiresolution pyramid, so it has to be very fast. We use a very simple MLP, working with a low resolution (9x11 retina), as shown in figure 21.

2. *Second stage* use a higher resolution to distinguish between faces and other objects. We use a MLP+RBF architecture: the MLP layer reduce the input dimension from 20x25 to 6x8 cells (see figure 22). Average number of calls per scene is approximately 3000.

3. *Third stage*. Various proposed systems use "high level" features search to confirm detections : presence of eyes, mouth, etc.

   Such features are often hard to detect in our images (hidden or rotated faces, low contrast due to flash), as can be seen in figure 24. Once again, we use a MLP+RBF module, but with a higher resolution and information from the body of the detected person. In effect, multimodular connexionnists architectures provide a very convenient way to combine different sources of informations and to optimize jointly the corresponding modules. In this case, we have three

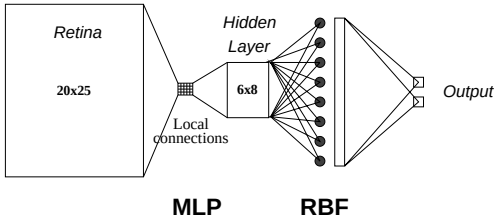

Figure 22: MLP+RBF for second stage of detection.

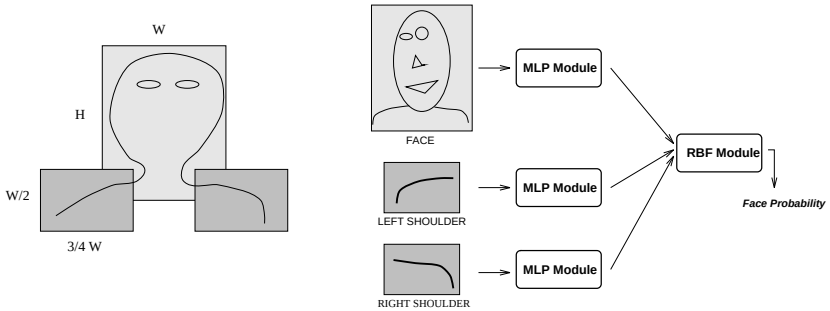

Figure 23: Third classification module uses information from face and shoulders regions. MLP modules extracts features from each regions. Those features are combined by a RBF classifier which take the final decision.

retinas (figure 23): one for the face image, and two smaller ones receiving information from the shoulders regions. Of course, these regions do not always carry useful information: the shoulders may not be observable. Nevertheless, using this source of information allowed us to reduce the false positive rate by 25%. In a first phase, each MLP module is independently trained to reconstruct the presented images (as in our standard MLP+RBF architectures), then the RBF classifier is initialized and the whole MLPs+RBF system optimized by gradient descent.

8.4 Results of the localization system

After the third stage of classification, we usually get several detections for each face. These detections are grouped by a simple clustering algorithm.

We plotted in figure 25 the detection curve measured on an independent set of 1000 scenes (test set). This curve has been obtained by varying the decision threshold of the third stage (the other threshold are fixed during the incremental learning procedure). If we allow 2% of the face to be non detected, we get less than one false positive in 100 scenes.

In the current (non-optimized) implementation, a 760x580 image is processed in 20 seconds on a 133MHz PC. Note that we do not make any assumptions on the structure of the scene and don't use any color or movement information to speed up

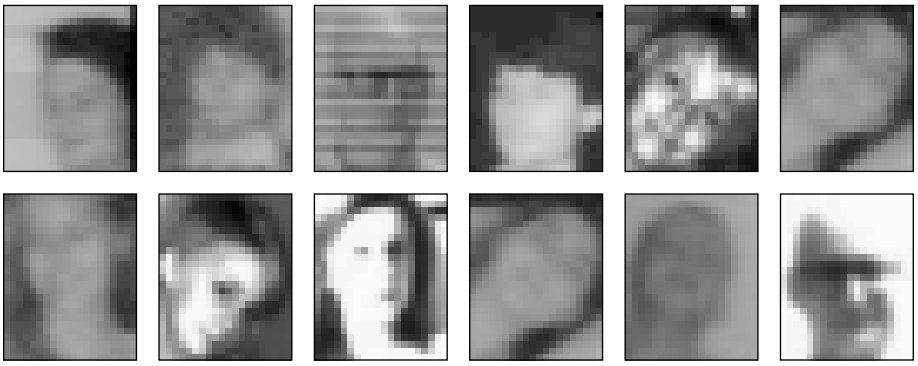

Figure 24: Some faces images of bad quality...

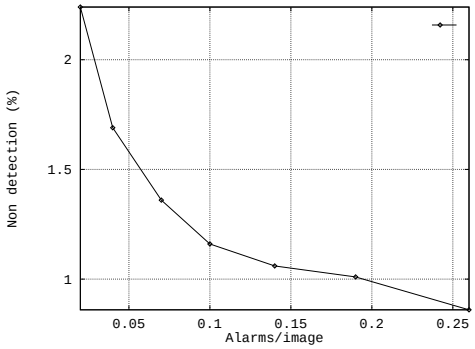

Figure 25: Non detection rate vs false positives/image, obtained with the presented MLP+RBF 3 stage system.

the search. The presented system could be used to process any complex static image.

Instead of specializing the system to gain speed, we will try in the next section to gain more accuracy (i.e. to reduce further the false alarm rate) by introducing a new type of image classifier, Support Vector Machines.

9 Improving the accuracy with Support Vector Machines

Very recently, SVM classifiers (see section 3.2) have been compared to RBF on digit image recognition, clearly demonstrating the better accuracy of SVM [36]. An application to face detection has also been presented last month [31], with interesting results.

Here, we present a few preliminary results obtained by replacing the last MLP+RBF module of our face detection system by a SVM machine. We used a Gaussian kernel as dot product.

The best machine obtained so far has about 2000 support vectors. This number can be reduced by 10% without significant loss of performance (pruning the SV associated to low values of multiplier α. The performances obtained during this preliminary experiments were very similar to our best NN classifier, which were selected after a lot of time consuming trials and errors.

Note that the technique used is quite simple and leaves room for improvements:

- invariances to translations, rotations and illumination changes should be added, either by using a specialized dot-product or with the "virtual support vector" technique [35].
- speedups by a factor 50 have been reported by reducing the number of SV via the "reduced set" method [10, 12].

During optimization, SVM extract "important" examples from the learning set. The importance of each support is quantified by its associated Lagrange multiplier α_i. This can be used to quickly inspect a large pattern database by visualizing the support vectors. For instance, incorrectly labeled pattern will almost surely be selected as support, with high α_i value. The images shown in figure 26 are faces in the training set which were originally labeled as "non-faces". Removing these patterns and re-training the system decrease the error rate while reducing the number of SV.

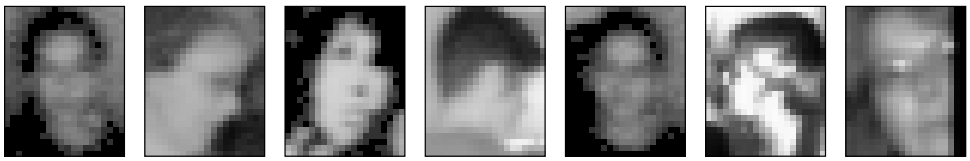

Figure 26: Incorrectly labeled images found among the Support Vectors.

10 Identification of detected faces

After localization, we want to identify the detected face to a member of a small group of known persons, which we called a "family". Such a system is for instance useful for automatic people tracking, in surveillance or telecommunication applications. For this kind of tasks, a large number of examples is usually available; faces images of each person can for instance be automatically extracted from a video sequence. On the other hand, images are of bad quality, the identification system need to be very robust to changes of light, rotations, imperfect segmentation, etc. Figure 27 shows some faces images.

A TDNN classifier network can get good performances on this task, if it is trained with enough examples [42]. The average generalization error rate measured for families of eleven persons is 2 %. Larger error rates are observed if a person change its appearance (e.g. hairs). A known problem with such discriminant system is their inability to accurately detect and reject patterns from unknown classes (in our case non faces images and faces from previously unknown persons) [19]. This is illustrated by the curve 28, which plots the the rejection rate on both faces and non faces as a function of the face identification error rate. With an error rate of 1 %, about 40 % of the unknown faces are accepted as images of faces from one of known classes.

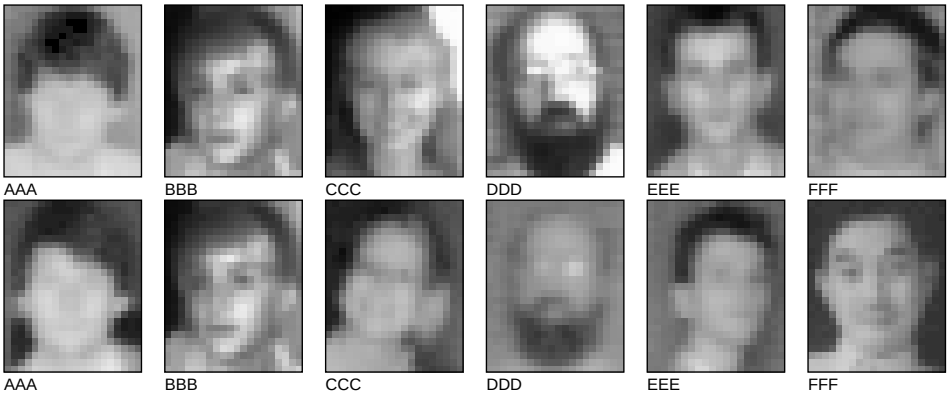

Figure 27: A typical "family". Images from the first line belongs to the training set, and from the second line to the test set.

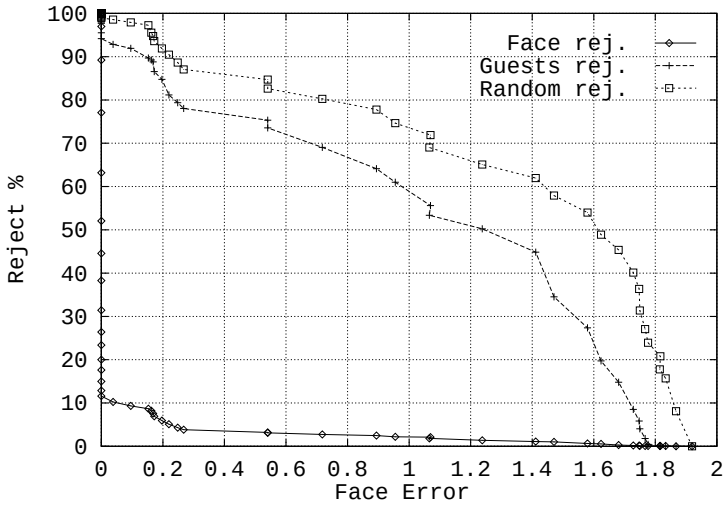

Figure 28: Rejection rate as a function of the face identification error rate, measured on three sets of images: known faces, unknown faces (labeled "Guests") and random noise images.

11 Conclusion

The results discussed in this paper clearly demonstrate that connexionnist methods allow to build robust and accurate systems for real world image processing problems. We shown how different kind of models can be combined to improve performances, both in terms of computing speed and error rates. The presented system offers very good performances for face detection in complex images, and is suitable for real world applications. Our recent results using Vapnik's Support Vector Machines lead to further reduction of error rates.

References

[1] P. Baldi and K. Hornik. Neural networks and principal component analysis: learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

[2] P. L. Bartlett. Vapnik-chervonenkis dimension bounds for two- and three-layer networks. *Neural Computation*, 5(3):371–373, 1993.

[3] E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[4] Y. Bennani. *Approches Connexionnistes pour la Reconnaissance Automatique du Locuteur: Modélisation et Identification.* PhD thesis, Université Paris XI, 1992.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, 1995.

[6] L. Bottou and P. Gallinari. A formalism for neural net algorithms. In EC2, editor, *Proceedings of Neuro-Nîmes'91*, 1991.

[7] L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, 3, pages 781–788, 1991.

[8] H. Bouattour, F. Fogelman Soulié, and E. Viennet. Neural nets for human face recognition. In *Proceedings of IJCNN Baltimore*, volume 3, page 700, 1992.

[9] L. Breiman and J. Friedman. A new approach to multiple outputs through stacking. Technical Report LCS 114, Standford University, 1994.

[10] C. C. J. Burges. Simplified support vector decision rules. In L. Saitta, editor, 13^{th} *International Conference on Machine Learning*, pages 71–77. Morgan Kaufman, 1996.

[11] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Submitted to Data Mining and Knowledge Discovery*, 1997. Available from `http://svm.research.bell-labs.com/SVMdoc.html`.

[12] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, 9, pages 375–381, 1997.

[13] R. Chellapa, C. L. Wilson, and S. Sirohey. Human and machine recognition of faces: A survey. *Proceedings of the IEEE*, 83(5):705–740, May 1995.

[14] V. Cherkassky, J. Friedman, and H. Wechsler, editors. *From Statistics to Neural Networks: Theory and pattern recognition Applications*, volume F136 of *NATO ASI Series in Computer and System Sciences*. Springer Verlag, 1994.

[15] I. Craw, H. Ellis, and J. Lishman. Automatic extraction of face-features. *Pattern Recognition Letters*, 5:183–187, Feb. 1987.

[16] R. Fernandez, E. Viennet, E. Goles, R. Barrientos, and M. Telias. On-line coarse ore granulometric analyser using neural networks. In *Proceedings of ICANN'95 Paris*, volume industrial session, Oct. 1995.

[17] M. Fleming and G. W. Cottrell. Categorization of faces using unsupervised feature extraction. In *Proceedings of IJCNN Washington*, volume 2, 1990.

[18] F. Fogelman Soulié, D. Gureghian Gastinel, J. Loncelle, M. E., and J. Gallinari. A check reading system. In F. F. Soulié and P. Gallinari, editors, *Industrial Applications of Neural Networks*, pages 363–368. World Scientific, 1997.

[19] F. Fogelman Soulié, B. Lamy, and E. Viennet. Multi-modular neural networks architectures for pattern recognition: Applications in Optical Characters Recognition and Human Face Recognition. *Int. J. Pattern Recognition and Artificial Intelligence*, 7(4), 1993.

[20] J. Friedman. Adaptive techniques for machine learning and function approximation. In Cherkassky et al. [14], page 61.

[21] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[22] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S. A. Solla. Structural risk minimization for character recognition. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, 4, pages 471–479, 1992.

[23] L. Hansen, C. Liisberg, and P. Salamon. Ensemble methods for handwritten digit recognition. In *NNSP'92*, pages 333–342. IEEE, 1992.

[24] C. Huang and C. Chen. Human facial feature extraction for face interpretation and recognition. *Pattern Recognition*, 25(12):1435–1444, 1992.

[25] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[26] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In Touretzky, editor, *Advances in Neural Information Processing Systems*, 2, pages 396–404, 1990.

[27] G. L. Martin and M. Rashid. Recognizing overlapping hand-printed characters by centered-object integrated segmentation and recognition. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, 4, pages 504–511, 1992.

[28] O. Matan, C. J. Burges, Y. Le Cun, and J. S. Denker. Multi-digit recognition using a Space Displacement Neural Network. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, 4, pages 488–495, 1992.

[29] M. Møller. Supervised learning on large redundant training sets. *Internation Journal of Neural Systems*, 4(1):15–25, 1993.

[30] J. More and G. Toraldo. On the solution of large quadratic programming problems with bounds constraints. *SIAM Optimization*, 1(1):93–113, 1991.

[31] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proceedings of CVPR'97*, Puerto Rico, June 1997.

[32] N. Roeder and X. Li. Accuracy analysis for facial feature detection. *Pattern Recognition*, 29(1):143–157, 1996.

[33] H. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. Technical Report CMU-CS-95-158R, School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213, 1995.

[34] H. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. In *Advances in Neural Information Processing Systems*, 8, 1995.

[35] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Proceedings ICANN'96 Int. Conf. on Artifical Neural Networks*, Berlin, 1996. Springer Verlab.

[36] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. Signal Processing*, 45:2758–2765, 1997. Also available from `ftp://publications.mit.edu`, A.I. Memo No. 1599.

[37] S. Thiria, Y. Lechevallier, O. Gascuel, and S. Canut, editors. *Statistiques et méthodes neuronales*. Dunod, 1997. (In french).

[38] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[39] V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[40] V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, 4, pages 831–838, 1992.

[41] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[42] E. Viennet. *Architectures Connexionnistes Multi-Modulaires. Application à l'analyse de scène*. PhD thesis, Université Paris XI, june 1993.

[43] E. Viennet and F. Fogelman Soulié. Multi-resolution scene segmentation using MLPs. In *Proceedings of IJCNN Baltimore*, volume 3, page 55, 1992.

[44] J. Vincent. Facial feature location in coarse resolution images by multi-layered perceptrons. In *Artificial Neural Networks*, pages 821–826. Elsevier Science, 1991.

[45] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):242–259, 1992.

