Application of projection learning to the detection of urban areas in SPOT satellite images

Konrad Weigl, Gerard Giraudon, Marc Berthod

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

Application of Projection Learning to the Detection of Urban Areas in SPOT Satellite Images

Konrad Weigl, Gérard Giraudon & Marc Berthod

N° 2143

December 1993

PROGRAMME 4

Robotique,

image

et vision

ISSN 0249-6399

Application of Projection Learning to the Detection of Urban Areas in SPOT Satellite Images

Konrad Weigl, Gérard Giraudon & Marc Berthod*

Programme 4 — Robotique, image et vision

Projet Pastis

Rapport de recherche n°2143 — December 1993 — pages

Abstract: We introduce a novel learning algorithm for neural networks, with the major feature of being rapid when compared to classical learning algorithms, offering misclassification rates of 5% and less after only a few iterations, i.e. 20-30 seconds of learning, depending on the task, if a suitable preprocessing has been done.
The algorithm is based on considering a neural network as a base in function space, base onto which the function to be learned is projected. We thus call our algorithm *projection learning*. We present the algorithm, show the application to the detection of inhabited areas in satellite images, discuss the various preprocessors used, compare to other approaches used, and outline further directions of research

Key-words: Learning in computer vision, segmentation and perceptual grouping, neural networks, texture analysis, pixel-based classification, low-level processing, feed-forward networks, satellite image analysis

(Résumé : tsvp)

*weigl@sophia.inria.fr giraudon@sophia.inria.fr berthod@sophia.inria.fr

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Application de l'apprentissage par projection à la détection de zones urbaines dans les images satellite SPOT

Résumé : Nous introduisons un nouvel algorithme d'apprentissage pour réseaux de neurones, dont la caractéristique principale est d'être rapide en comparaison des algorithmes classiques d'apprentissage, avec des taux d'erreur de 5% et moins après quelques itérations d'apprentissage, d'une durée de 20-30 secondes, en supposant un prétraitement convenable.
Nous considérons un réseau neuromimétique comme une base non-orthogonale dans un espace de fonctions, base sur laquelle la fonction à apprendre est projetée. C'est pourquoi nous appelons notre algorithme *Apprentissage par Projection*. Nous présentons l'algorithme, montrons l'application à la détection de zones urbaines dans des images satellite, discutons les prétraitements utilisés, comparons à d'autres approches, et indiquons les axes futurs de recherche.

Mots-clé : Apprentissage dans la vision par ordinateur, segmentation et groupement perceptuel, réseaux de neurones, analyse de texture, classification basée pixel, traitement bas niveau, réseaux à propagation directe, analyse d'images satellite

Contents

1 Introduction

Neural networks have been used for quite a few tasks in vision, and more specifically in early vision, and with appreciable success [1] [12] [25] [8] [16] [10] [5]. Some of the reasons for the use and the success are:
Firstly the characteristic nature of image data, forming a two-, three-, or higher-dimensional lattice of data, where higher-order information is often encoded in nearest-neighbor interaction, and can thus be treated in parallel by a distributed architecture, such as a neural network.
Secondly, the ability of such networks to *learn*, i.e. to exhibit a general behaviour if suitable examples of the behaviour wanted are given. The ability of the network to learn relieves the researcher in part from the tedious task of formulating the relationship between data input and relevant information to be extracted from it, using instead a neural network of the appropriate type, which will learn to associate specific inputs to specific outputs, if given the right examples, so that it can associate.
Thirdly, and specifically for classification tasks, the greater independence of neural networks from the statistical distribution of the data, as compared to classical classifiers [8]. This allows for a better performance of neural networks on these tasks, when the a priori assumption of classical classifiers concerning data distribution (E.g. normal distribution, etc.) are not met. Obviously, however, neural networks, like classical classifiers, expect the classification function to have some degree of smoothness: In a classification application, it is assumed that data close together in input space shall probably belong to the same class.
Fourthly, also, problems in early vision can be formulated as constraints expressable as an energy functional, and neural networks of other types, such as Hopfield Networks or similar [16], are suitable candidates to minimize such an energy functional and thus find an acceptable solution to the problem.

The present paper deals with a novel learning algorithm for a classical type of neural network called *feedforward neural network*, or *multilayer perceptrons*, short MLP [1]. These networks, used in about 70% of the industrial applications today, are usually taught via the backpropagation algorithm, easy to use, but rather slow to converge [18]. While there have been a multitude of successful attempts to improve the speed of

[1] It can also be used for other architectures, but that should not concern us here.

the algorithm, the velocity is still quite low. We are tackling the problem from another side: We consider neural networks such as Multi-layer perceptrons as non-orthogonal bases in a function space, bases which span submanifolds of that space. The basis functions of that base are the functions computed by the neurons of the hidden layer. A function to be approximated is then a vector in function space. The projection of that vector onto the submanifold spanned by the base is the function approximated by the neural network. That approximation is then optimal when the distance between the function to be approximated and its projection onto the submanifold is minimal by some metric. The metric classically used, which we shall also use, is the minimal squared error. We compute this distance in *sample space*, i.e. that subspace of function space the dimensions of which correspond to the input samples we have of the function to be approximated. The objective of learning in such a network is thus to minimize the distance between the function to be approximated and its projection onto the submanifold. This is achieved via dynamically rotating and shifting the base in such a way that the distance above is minimal. That rotation and shifting is executed through modification of the parameters of the basis functions of the network. A convenient way of computing the projection is with the help of *metric tensors*, a geometric object of differential geometry.
We call this new approach to learning *Projection Learning.*

2 Our Algorithm

2.1 Major Advantages of the Algorithm

The major advantage of this algorithm is that it gives within two to four iterations, i.e. a few seconds, solutions which are close to the optimum possible, with an error decreasing with further convergence to between a third and a fifth of that initial value. This allows researchers to break off learning almost immediately if that initial error is too high, and to search for another combination of preprocessing steps, or increase the number of hidden-layer neurons; the latter is possible only up to a certain limit, of course.
Total time to convergence is also much smaller than with backpropagation and its derivates, as can be seen below in the section "Implementation Example and Comparison

to Backpropagation".
Another advantage is its robustness to the values of the parameters to initialize, such as learning rate, momentum terms, etc. The only initialization is the choice of weights from input to hidden layer, the weights from the hidden to the output layer are computed by the algorithm itself.

2.2 The Algorithm Intuitively

It is not our goal here to present a full mathematical treatment of the algorithm. We shall rather focus on an intuitive introduction and on major concepts. Fig. 1 shows, for demonstration purposes, a primitive feed-forward network, with one input node, two hidden-layer nodes, and one output node. The input node just accepts the input value x and sends it to the two hidden layer nodes indexed i, which compute the function $g_i(x)$ as follows:
a) Multiply x with a scalar a_i;
b) add a bias value θ_i;
c) apply to the result a non-linear function, such as a sigmoid $sigm(x) = \frac{1}{1+e^{-x}}$. Thus $g_i(x) = sigm(a_i x + \theta_i)$. $g_i(x)$ is the classical perceptron mapping function [24], though almost any other function is usable as well, depending on the task at hand.
Each hidden-layer node then sends its output to the output node. This latter again multiplies the value obtained from each node with a specific scalar A^i [2], sums over the two products, and outputs the result.
One input node and one output node allow thus such a network to compute a scalar function f(x) of one scalar input x. The values a_i, θ_i, and A^i are called *weights* in neural network terminology.

The aim of learning is now to compute the values a_i, θ_i, A^i in such a way that an error E as follows:

$$E = \sum_{x_k} (F(x) - \sum_i A^i g_i(a_i, \theta_i, x))^2 \qquad (1)$$

becomes minimal, where $\sum_{x_k}$ is the sum over all the examples x_k, and $F(x)$ is the function to be approximated. The present problem belongs thus to the class of separable non-linear least-squares [13] [14] [9].
The classical way is to use the backpropagation algorithm, i.e. initialize all weights to

[2] The superscript does *not* imply a power, it is just an index.

some small initial value, then to compute the dependence of the error E on each A^i, on each a_i, and on each θ_i, and to minimize each of these variables simultaneously, so that the error E decreases. The problem is that these variables are interdependent, so that the learning step has to be chosen small enough not to cause too many errors in the computation, but large enough to progress with the decrease of the error in an acceptable time.
The advantage of our algorithm, along with other approaches [19], is that the optimal weights A^i from hidden to output layer, for given weights a_i, θ_i, are computed *directly*, without a learning process, so that convergence is much faster. We only learn the weights from input to hidden layer. Furthermore, our algorithm allows for a universal choice of basis functions $g_i(x)$ to use, as long as they are differentiable in some sense with regards to their parameters[3].

Thus, indeed, our algorithm reduces to the following form, where $param_i$ is a weight of a hidden-layer neuron indexed i to be computed via gradient descent:

$$\frac{dparam_i}{dt} = -\frac{dE}{dparam_j} = -2\sum_x (F - \sum_\nu A^\nu g_\nu(x)) A^i \frac{dg_i(x)}{dparam_i}) \qquad (2)$$

The weights A^i are computed by a projection of the wanted function $f(x)$ onto the manifold spanned by the functions $g_i(x)$ computed by the hidden-layer neurons (Cf. fig 3). Let us assume that we have only three examples to learn (Cf. fig. 2). We known thus the three input values $x_k, k \in \{1,3\}$, the three wanted function values $F(x_k), k \in \{1,3\}$, and we can compute the outputs $g_i(a_i, \theta_i, x_k), k \in \{1,3\}$, once we have initialized the weights a_i, θ_i to some numbers. We can then consider the set of values $F(x_k), k \in \{1,3\}$ as a vector in a three-dimensional vector space, a discrete function space, and consider the two functions $g_0(a_0, \theta_0, x_k), k \in \{1,3\}$ and $g_1(a_1, \theta_1, x_k), k \in \{1,3\}$ as two vectors in that same space (Cf. fig. 3). These two vectors span a plane in that three-dimensional space. It is thus sufficient to project the function $F(x)$ to be approximated onto that plane to obtain the optimal approximation possible $F_{approx}(x)$ for a given set of functions g_0 and g_1 and the corresponding weights A^0 and A^1. This corresponds to a linear least-squares approach to the problem of computing the hidden- to output layer weights.
Iteratively, now, the plane will be shifted towards $F(x)$ in such a way that the distance $F(x) - F_{approx}(x)$, which corresponds to the error E, is minimized as much as possible.

[3]Otherwise, of course, Simulated Annealing or other Monte-Carlo methods are applicable.

When we have reached a minimum for that distance, the network has learned the function as well as it can.

2.3 The Algorithm Step by Step

While there are quite a few different ways to implement the algorithm [30] [31] [32] [33], we shall not dwell on the different possibilities here, but focus on the simplest one, explaining it step by step: Refer to figure 5.

Let $x_k, k \in 1, .., M$ be the set of input examples, where M is the number of examples. if we have only one input line, x_k is a scalar, otherwise, it is a vector of values, one per input line. Let $F(x_k)$ be the wanted output to be associated with x_k, i.e. the output we would want the network to produce for the input given being x_k. We also have thus M such wanted output samples, $k \in 1, .., M$. In the case shown in figure 5, the number of examples is 3; we have thus 3 dimensions, tagged by x_0,x_1,x_2.

The first step is initialization of the weights a_i and θ_i of each hidden-layer neuron: Like any other network with supervised learning, we initialize these weights. The speed of convergence depends on the initial weights, but we could observe no difference beyond a double to fourfold increase in convergence time for real-world problems with thousands of examples. A random initialization is helpful. Something which might be critical for small number of examples, but again not for real-world problems, is the required linear independence of the basis functions in sample space; if they are not linearly independent in sample space, the metric tensor described below under 2) is singular and then only invertible by Singular Value Decomposition, Miller-Penrose Pseudo-inverse, or gradient descent as above. Random initial weights should take care of the problem. We have never had any problem with real-world applications using any random initialization with an even probability distribution, a mean of 0.0, and a maximal span of values of up to $+/-0.5$. If we use a higher span of values, up to $+/-1.5$, we obtain slightly faster convergence, but at the risk of falling into local minima.

The steps in the learning loop, thus per iteration, are then the following:

1) Compute the output values $g_i(x_k)$ of all the hidden-layer neurons i for all the input samples $x_k, k \in 1, .., M$;

2) Multiply pairwise $g_i(x_k)$ with $g_j(x_k)$, and sum over all these products $g_i(x_k)g_j(x_k)$, $k \in 1, .., M$; this gives us the scalar g_{ij}:

$$g_{ij} = \sum_k g_i(x_k)g_j(x_k) \tag{3}$$

Do this for all hidden-layer neurons i, j; These scalars g_{ij} are the components of the covariant metric tensor, which is a symmetric and real matrix. Invert the matrix by any method of your choice. This gives us the contravariant metric tensor g^{ij} .

3) Multiply all the output samples given $F(x_k), k \in 1, .., M$ with the corresponding outputs of the hidden-layer neurons $g_i(x_k), k \in 1, .., M$ computed above. Sum over all these products $F(x_k)g_i(x_k)$, $k \in 1, .., M$; this gives us the covariant component A_i:

$$A_i = \sum_{x_k} F(x_k)g_i(x_k) \tag{4}$$

Repeat for all the hidden-layer neurons.

4) Multiply the contravariant metric tensor obtained above, which is a matrix, with the vector formed by all the covariant components A_i; this gives us the vector of contravariant components A^i:

$$A^i = \sum_j g^{ij}A_j \tag{5}$$

Multiplying now all the hidden-layer neurons $g_i(x_k)$ with the corresponding A^i, and summing up over all the indices i, gives us the function approximation for the network and input sample x_k, called $F_{approx}(x_k)$:

$$F_{approx}(x_k) = \sum_i A^i g_i(x_k) \tag{6}$$

Thus we can now compute the distance D:

$$D = \sum_{k=1}^{M} (F(x_k) - F_{approx}(x_k))^2 \tag{7}$$

5) We have to differentiate D with regards to the parameters/weights of the nodes/basisfunctions; thus we need, by the chain rule, for example, to differentiate w.r.t. the parameters of

the hidden-layer neuron j, called $params_j$:

$$\frac{dD}{dparams_j} = \sum_x (F(x_k) - F_{approx}(x_k))^2 \frac{d(F(x_k) - F_{approx}(x_k))^2}{dparams_j} \tag{8}$$

implies[4] from equation 6 above:

$$\frac{dD}{dparams_j} = \sum_x 2(F(x_k) - \sum_i A^i g_i(x_k))(A^j \frac{dg_j(x_k)}{dparams_j}) \tag{9}$$

$\frac{dg_j}{dparams_j}$ depends on the type of basis function used, obviously; for a sigmoid, it is fo¡r example [24]:$\frac{dg_j(x_k)}{dparams_j} = g_j(x_k)(1 - g_j(x_k))x_k$ for the computation of a non-bias input, and $\frac{dg_j(x_k)}{dparams_j} = g_j(x_k)(1 - g_j(x_k))$ to compute the bias weight. 6) Once we have computed $\frac{dD}{dparams_j}$ for all parameters, we can compute:

$$\frac{dparams_j}{dt} = -\frac{dD}{dparams_j} \tag{10}$$

and compute one step via gradient descent or other; then we reiterate the loop with steps 1) - 6), until the minimal distance has been found.
Of course, if the distance, seen as an energy, is not convex, other methods, such as Simulated Annealing, must be used.
Our algorithm has intrinsically *no* learning parameters or learning rates to set; the decisions which might be critical are two:
a) The initial set of parameter values, where a suitable choice influences learning time by a factor of two to four, but has no effect beyond that; if an initialization beyond +/- 0.5 is chosen, the system has been observed to fall into local minima. Below that level, no local minima were observed for a realistic number of examples, i.e. beyond 50-100.
b) If a direct inversion of the matrix is chosen, the maximum span between largest and smallest eigenvalue: We used universally 1.0^{18} as the ratio, which gave fast and good results.

[4] We have shown in [33] that the terms depending upon $\frac{dA^j}{dparams_j}$ cancel out.

2.4 Implementation Example and Comparison to Backpropagation

In our implementation, we used the most primitive gradient descent: We choose initially an arbitrary stepsize, move along the gradient direction, compute the energy there, and reduce iteratively the stepsize until the energy found is smaller than the initial one. There we compute the gradient, choose a stepsize which is a function of the previous one and the difference between the energy levels, and reiterate the procedure.
Fig. 6 shows a comparison of the implementation of our algorithm described above with backpropagation, both implemented with the same standard gradient descent as described above: Note the increase in computation time for backpropagation with rising number of hidden-layer neurons and samples, while the increase for the projection learning network is much slower: This is due to the fact that the higher the number of hidden-layer neurons, the greater the manifold spanned by them in function space, and thus the greater the chance that the function to be approximated is close to the manifold: Therefore the distance/error to be minimized is already smaller from the start, and the network converges thus faster. We have also made a run with backpropagation on the application described below: Under the same external criteria, i.e. 20 hidden-layer neurons, same initialization values for the input-to hidden-layer weights, the projection learning algorithm took 400 seconds CPU total time to converge to an error rate of 2%, with an error rate of 2.5% after the first iteration, while the backpropagation algorithm offered at first iteration an error rate of 18%, and had not converged to 2% after several days.

3 The Application to the Detection Task

We have applied the algorithm to a low level image processing task: The recognition of inhabited areas on satellite images. The images stem from the french satellite SPOT, and have been taken with a panchromatic camera. The classification is pixel based, i.e. we classify each pixel as being either a pixel from an inhabited area or from the countryside, as a function of measures of the greyscale values of itself and its neighbors. The approach taken was to preprocess the image in one or several steps with one or several suitable preprocessors, and then present the resulting preprocessed images as inputs to the network. The aim of the preprocessing steps was twofold:

a) Retain as much relevant information as necessary, while removing irrelevant data, thereby reducing also input dimensionality, though that did not play such a big role in our present case;
b) Incorporate neighborhood information to a suitable degree: Not enough neighborhood information, and we obtain a large number of isolated misclassifications, too much neighborhood information, and isolated pixel clusters of one class within the other class will get ignored. We shall show below in the section "Results" two examples resulting from that tradeoff.
The task ahead was thus the following:
a) Choose suitable preprocessing steps and their sequence and combination;
b) Choose a suitable number of hidden-layer neurons;
c) Choose a suitable set of samples to train the network on.

3.1 General Approach

Experience showed that whatever the error rate was after the first iteration, which took roughly ten to fifteen seconds to compute, the final error rate which was achievable by the network was never less than a third to a fifth of that first iteration. We thus tested a variety of preprocessor steps and combinations, as described below, as well as different numbers of hidden-layer neurons, computing only the few first iterations, and breaking off if the initial error computed was too high. This allowed us to find rather rapidly the optimal combination, described under "Further Preprocessing"

3.2 Preprocessing

The preprocessing, crucial for a successful classification, evolved from two sources:
a) previous work done by Gertner et al. [7], focusing on local derivatives and histogram measures, and
b) previous work done by Ryan et al. [25], based on orientation-invariant local power-spectral measures.

The objective of the preprocessing, as described above, was chosen in such a way that wanted discrimating criteria were kept and irrelevant characteristics removed. Gertner et al. [7] found preprocessing operators that, while not fully discriminating,

nevertheless removed the majority of irrelevant variablility: Their preprocessing was fourfold: Computation of the laplacian, computation of the distance of the local gradient from a local gradient average, computation of phase and amplitude of a local grey-scale value histogram. All the resulting images were then averaged with an anisotropic local operator.
The following two sections describe the operators used by Gertner et al. [7], the next after that the operator of Ryan et al. [25], while further common processing is described beyond.

3.2.1 Computation of Laplacian and Difference to Local Average

The Laplacian was computed in the classical manner, i.e. the curvature operator was implemented in a discrete way both in the x and y direction. Afterwards, the module of the two values was taken. The assumption was that landscapes would either have a very small (Within fields) or very large (Boundaries of fields) module, while cities would be characterized by a medium-valued module. The second preprocessing consisted in taking the scalar distance of the vector-valued local gradient to the local average of that same vector-valued gradient: This operator is a measure of the local directional variance of operators, variance which is expected to be large in cities (Since the gradient shifts fast in fine textures such as those characterizing cities) but small in fields and other characteristic non-inhabited areas. The gradient was computed using the classical Canny-Deriche operator, the local average of that operator was computed over a window 15x15 pixels.

3.2.2 Computation and Measures of Local Histograms

A local histogram of grey-scale values was computed, and two measures of it taken (Cf. fig. 7): The phase, which is simply the greyscale value which occurs most often in the neighborhood, and thus can be identified as the largest peak of the histogram, and the amplitude of the histogram, which is simply the deviation of the histogram from a histogram of equally distributed greyscale values. Again, the phase of the histogram is a variable which allows us to differentiate between dark-colored non-inhabited sites such as water bodies and lighter areas such as cities. The histogram amplitude also has discriminating function: It can be interpreted as the distance to a fully noisy signal; it

will thus be relatively large for rather uniformly colored areas such as fields, and rather small for areas with many different grey-scale values, such as towns, where the entropy of the distribution is higher.

3.2.3 Local Orientation-invariant Power Spectra

Ryan et al. used other operators to discriminate between land and water, and so detect shorelines [25]. The operator he used is similar to the one shown in figure 8, which is the one we applied: We first took a local FFT of an 8x8-pixels patch around the pixel to be classified, then computed the corresponding moduli of the spectral components of the power spectrum. Figure 8 shows the subsequent processing: The frequency zero is shown on the upper left, and then from there the frequencies increase. As did Ryan, we took averages over the moduli, to get a more or less isotropic output. We computed six values, which were the averages over the fields indexed with the corresponding numbers: The value at the frequency zero, by far the largest, was divided by 64, but only for purposes of representation.
For our purposes it turned out to be enough to compute that operator for each second pixel horizontally and vertically of the image.

3.2.4 Anisotropic Smoothing

In order to minimize isolated misclassifications, neighborhood information was incorporated by submitting all of the images resulting from the preprocessing above to a nonlinear anisotropic smoothing. That smoothing aimed at spreading values around without destroying too much contour (Cf. fig. 9):
The operator consists of two steps:
a) Computation of four averages a through d, where averaging runs over the pixels marked respectively a through d. Pixels marked with several letters are counted in all of the corresponding averages.
b) Once the four averages have been computed, the four differences to the grey-scale value of the pixel to be classified are computed. The smallest difference found will then be associated to the pixel to be classified, marked black in fig. 9.
This anisotropical smoothing enables strong smoothing without destroying too much contour.

3.2.5 Further Preprocessing

Since using the preprocessing by Gertner above for learning and classification yielded in our case unsatisfying results, being for example sensitive to some type of ridges and contours, we tried out further additional alternative preprocessing operators, as well as further preprocessing of images already treated by the steps as described above.
Here the inherent speed of our algorithm to be able to judge rapidly the quality of a specific combination of preprocessing steps was very helpful.
We found two combinations to give good results: Combination a was based on removing one of the preprocessing operators above, namely the amplitude of the histogram, and treating the image with the three remaining preprocessing steps of Gertner et al. as described above, with anisotropic smoothing. Then, for each pixel, average and variance was taken over pixels distanced +/- 6 pixels crosswise from the pixel to be classified and the pixel itself. Each of the three preprocessed images was treated that way, resulting in six images which were then used as input for the network.
Combination b consisted of treating the image with the FFT-preprocessing above, anisotropic smoothing, and then taking only the average over the five pixels as in combination a. With six spectral components, we obtain thus also for combination b a six-dimensional input.

3.3 Training

The training for the present application was done as follows: We cut out initially six sample images of dimension 128x128 resp. 64x64 from the full 1024x1024 image of the neighborhood of the french town of Larochelle, containing each either an example of a town or of landscape, and attributed to each sample image the label TOWN or LAND, according to the content. It turned out that these samples had not captured all of the variability of the landscape: We chose thus four more images as samples, also from other source images.
The full number of sample images used to find the optimal preprocessing combination and number of hidden-layer neurons was thus ten.
At each iteration of the gradient descent, we chose randomly another set of pixels to be learned from the set of sample images; this guarantees generalization at least within the set of sample images. The number of samples taken at each step was chosen to be between 2000 to 3000, which yielded good results.

In the cases where we let the network converge all the way to the preset maximal tolerated classification error, we verified the generalization capacity of the trained network at least for the sample images: We took five further sets of 2000-3000 pixel examples each from the sample images, and computed the error rates for each of these five sets of samples. These error rates were on the order of +/-1% around the threshold set for maximal error.
Note that we did not scale or normalize the values obtained before using them as inputs for the network. For backpropagation networks, this is usually done, and is not necessary in our approach.

3.4 Results and Discussion

For the trial of the optimal combination of the preprocessors, error tolerance was set each time to a low level, in our case 3%, since we stopped in most cases the process after a few iterations anyway, until the optimal combination of preprocessors was found. We found after trial and error the optimal combinations described above in the section "Further Preprocessing", which yielded a classification error of only 3%-4% on the very first iteration, where the usual first classification error was on the order of 5% to 12% for the other preprocessor combinations. We thus decided to let that network run to 1%-2% classification error.
Classification error was based on thresholding: Since the network was trained to associate a 1.0 output to a town pixel, and 0.0 to a countryside pixel, after classification, we thresholded the results: An output above 0.5 was classified as a pixel belonging to a town, an output below 0.5 was classified as a pixel representing country.
After convergence, the image, of size 1024 by 1024 pixels, was thus correctly classified on the basis of roughly 10,000-50,000 pixel samples, which shows the generalization ability of the algorithm and of the network.

The figures 10, 11, 12, and 13 show inputs and results: On the images of classification results, we marked in white the pixels classified by the network as belonging to towns, the pixels belonging to the countryside are left unchanged.
Figures 11 and 13b show results after preprocessing with combination a as described in section "Further Preprocessing" above: The basic triple preprocessing proposed by Gertner et al., then forming average and variance over five pixels, i.e. the pixel itself

with the four pixels distanced +/6 pixels from it. Classification was then done with 15 hidden-layer neurons and 3000 samples per iteration: Note the short time of 82 secs CPU on Sparc 10 to convergence, which is a result, as already mentioned, of the relatively high number of neurons in the hidden layer. This preprocessing combination is able to detect even small villages, as seen in the image 13b on the left, but reacts also to false local signals, i.e. what is presumably a road bordered by textural elements, as seen in the same image in the upper middle.
Figures 12 and 13c show results using preprocessing with combination b from section "Further Preprocessing" above: The preprocessing combination consisting of the averaged local powerspectra with multiple subsequent spatial averaging and smoothing.
As can be seen in the figures 13c, the detection system is less sensitive to small textural misclassifications, but smaller villages are also in danger of being ignored.
Note that if the number of hidden-layer neurons varies between 10 and 15, we were not able to detect its influence on the final quality of results.
Also here, in contrast, we used only 10 hidden-layer neurons, resulting in a total computation time to convergence of 43 secs CPU. These convergence times are short even on the time scale of our algorithm: On the average, time to full convergence was between 400-600 secs, with an extreme case of 2400 seconds CPU for a trial with an extremely low error rate. 1-3 iterations, usually sufficient to estimate the effectiveness of a given preprocessor combination, took roughly 20-30 secs.

We have not applied any postprocessing to the results obtained: In practice, one would use a postprocessing to remove small isolated specks, if that is deemed necessary. The potential operators would be e.g. mathematical morphology, as done in [7], or coupled neuro-oscillators, as in [29].

3.5 Further Discussion

3.5.1 The Neighborhood Dilemma

To sum the results up, the vast majority of the preprocessors used had no difficulty detecting the large urban areas; the difficulties lay rather in the small isolated spots of texture which is similar to urban texture, but of countryside origin, such as crossroads,

roads crossing rivers, etc.: If we use a large spatial neighborhood to detect urban areas, we shall have little misclassification there, as can be seen in images 12 and 13c; however, we risk not detecting small villages, as can be seen in the same images.
If we insist on having to detect these small villages, we have two options:
a) accept misclassification of small countryside patches as villages, and do a subsequent analysis by hand or maybe a training with other preprocessing operators on that particular subproblem;
b) Use higher-level information, e.g. use the heuristics that villages are linked by roads, apply a specific road detector, and fuse the information at a higher level. A possible architecture for such an approach would be e.g. the one proposed in [4], or one based on a higher-level neural-network architecture.

3.5.2 How to Define a Quality Metric?

The combinations discussed above were not the optimal ones in the strict Mean-Least-Squares error sense: Here the additional preprocessing operators described below, namely the abovementioned regression on the local power spectrum, yielded lower rates. That regression preprocessor was, however, more sensitive to ridges and strongly-contrasted contours, as was also the original combination of preprocessing operators. Thus we consider the two combinations presented as the optimal ones for the concrete task presented. It might thus be a worthwhile goal for the future to define cost functions which reflect better the task at hand, e.g. cost functions which increase if the neighboring pixel does not belong to the same class, and to find ways to implement them in classification algorithms, by combining them for example with image restoration approaches such as in [34] [29].

3.5.3 Further Unsuccessful Preprocessing

While for our application, other preprocessing operators we tested did not yield good results, we shall mention them:
a) two measures of the local Fourier transform of the image: Taking the FFT of 8x8-blocks of pixels of the image, we fitted via least-squares a regression to the modulus of

the power spectrum of the image as a linear function of the logarithm of the frequency, making a separate vertical and horizontal fit. We actually used two measures of that fit: The coefficient itself, and the residual, which is a measure of the fit of the data to the model. We averaged over the vertical and horizontal slope of that regression slope, and input that scalar as a local measure of coarseness. The second input was the residual from that computation, which indicates how well the model was fit to the data, and which was revelatory in itself, while not improving the results;
b) the fractal local dimension of the data, by a similar approach as above, taken from Pentland [21]: He used a regression model of the log of the local powerspectrum to the log of the frequency;
c) the FFT of a circle of 16 pixels around the pixel to be classified; the aim was to be able to differentiate between higher spectral components characterizing towns and lower spectral components characterizing smoother grey-scale values such as fields;
d) the study of the phase differences between these spectral components, expecting that structures such as step edges characterizing field limits would exhibit more regular phase shifts than the rather random grey-scale value changes characterizing towns.
e) the blurring of the original image with an isotropic gaussian with a standard deviation of 1.5 pixels, to which simply the preprocessing operators above were applied;
f) computing the average over a circle of 16 pixels around the pixel to be classified, and processing the resulting image with the preprocessing by Gertner [7].
All of the above preprocessing steps, while most of the time rendering good results, did not appreciably improve them.

3.6 Outlook and Future Work

In conclusion, we have shown in this section that the algorithm is able to solve a real-world problem within a reasonable amount of time and obtain acceptable results. We hope that the algorithm, along with other algorithms which are more effective than backpropagation [19] [3], is a further step towards a wider acceptance of neural networks in the image processing task: It enables the researcher to focus more on the quest for suitable preprocessing operators and combinations, relieving him of the tedious labor of finding heuristically optimal sets of parameters, moment values, initial weights, and other untreatable criteria of a successful backpropagation application.
While the algorithm works successfully on the original image, as is e.g. state of the art

for classification of multispectral images via feed-forward neural nets [18], the preprocessing chosen depends strongly on the luminance values and variance of the image, and can thus be expected to be successfully to other images of same first and second-order statistics. However, in order to be able to classify images with other statistics using the parameters learned on one image, we have to look at other approaches. There are basically four approaches possible:
a) The network itself learns to ignore luminance and variance as discriminatory criteria, which implies training it with a representative data base of images, with a wide variety of luminance and variance values, as well as the use of suitable preprocessing operators;
b) a suitable preprocessor removes luminance and variance, and offers to the preprocessors used, such as the ones above, therefore images which are normalized in that sense;
c) using a preprocessor that does not rely on luminance and second-order statistics, such as fractal or multifractal measures [21] [15];
d) a combination of the above.
We have chosen the second approach, using a strongly non-linear operator that removes most of the variability of luminance and variance among different images. Using other operators, e.g. a histogram matching, as is often done to recalibrate luminance and variance of different images, will only be effective if we know that the images have similar contents, i.e. for example similar percentages of town and countryside pixels, etc.
Applying our operator before applying any of the operators above renders quite promising initial results: We are currently able to classify new images with weights learned on one image; however, classification error rate is still too high, roughly 10%.

References

[1] Anderson, J., A., Rossen, M. L., Sicuso, S., R., Sereno, M.E., Experiments with Representation in Neural Networks: Object Motion, Speech, and Arithmetic, in: Synergetics of Cognition, H. Haken, Ed., Springer, Berlin.

[2] Baldi, P., et Meir, R., Computing with Arrays of Coupled Oscillators: An Application to Preattentive Feature Discrimination, Neural Computation, 2, 458-471, (1990)

[3] Biegler-Koenig, F., and Baermann, F., A Learning Algorithm for Multi-Layered Neural Networks Based on Linear Least Squares, in: Neural Networks, Vol. 6, no. 1, pp. 127-131, 1993, Pergamon Press

[4] Clément, V., Giraudon G., Houzelle S., and Sandakly, F., Interpretation of Remotely Sensed Images in a Context of Multisensor Fusion using a Multi-Specialist Architecture, in: IEEE Transaction on Geoscience and Remote Sensing, FUng A.K., and Blanchard A.J., eds., vol. 31., no. 4, July 1993

[5] Daugman, J., Complete Discrete 2-D Gabor Transform by Neural Network for Image Analysis and Compression, in: IEEE trans. ASSP, vol. 36, No. 7, July 1988, pg. 1169-1179

[6] Ditzinger, T., et Haken, H., Oscillations in the Perception of Ambiguous Patterns, Biol. Cybern., vol 61,, pg. 279-287, (1989)

[7] Gertner, Valerie, and Herve, Andre, Extraction d'éléments texturés dans les images SPOT (Extraction of Textured Elements from SPOT Images), ISTAR/INRIA-Sophia Antipolis/ESSI 3, Research Report, May-July 1990, in french

[8] Gish, S.L. and Blanz W.E., Comparing the Performance of Connectionist and Statistical Classifiers on an Image Segmentation Problem, 1990?

[9] Golub, G.H., and Pereyral, V., The Differentiation of Pseudo-inverses and Nonlinear Least Squares Problems whose Variables Separate, in SIAM J. Numer. Anal., Vol 10, NO. 2, April 1973, pp. 413-432

[10] Fuchs, A., and Haken, H., Pattern Recognition and Associative Memory as Dynamical Processes in a Synergetic System, in: Biol. Cybern. No 60, pg 17-22 (1988)

[11] Heeger, D.J., Optical Flow from Spatiotemporal Filters, ICCV 1987, pg. 181-190

[12] Heermann P.D. and Khazenie, N., Classification of Multispectral Remote Sensing Data Using a Back-Propagation Network, IEEE Trans. on Geoscience and Remote Sensing, Vol., 30, No. 1, Jaunuary 1992

[13] Kaufman, L., A Variable Projection Method for Solving Separable Nonlinear Least Squares Problems, BIT 15 (1975), pg. 49-57

[14] Krogh, F.T., Efficient Implementation of a Variable Projection Algorithm for Nonlinear Least-Squares Problems, Numerical Mathematics, Communications of the ACM, March 1974, Vol. 17, Number 3.

[15] Levy Vehel, J., About Lacunarity, some Links between Fractal and Integral Geometry, and an Application to Texture Segmentation, INRIA, Domaine de Voluceau, Rocquencourt, B.P. 105, F-78153 Le Chesnay France

[16] Lin, W., Tsao E.C., and Chen, C., Constraint Satisfaction Neural Networks for Image Segmentation, in Proc. ICANN 1991, Helsinki, Kohonen et al. eds, Elsevier Publ. Co., 1991, pg. 1087-1090

[17] Lumsdaine, A., Wyatt, J. L., Jr., et Elfadel, I. M., Nonlinear Analog Networks for Image Smoothing and Segmentation, A.I. Memo No. 1280, M.I.T. Lab of Artificial Intelligence, January 1991

[18] Paola, J.D., and Schowengerdt, R.A., A Review and Analysis of Neural Networks for Classification of Remotely Sensed Multispectral Imagery, RIACS Tech. Rep. 93-05, June 1993, Research Institute for Advanced Computer Science, NASA Ames Research Center

[19] Pati, Y.C., and Krishnaprasad, P.S., Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformation, in: IEEE Trans. on Neural Networks, Vol. 4, No. 1, January 1993

[20] Pattison, Tim R., Relaxation Network for Gabor Image Decomposition, Biological Cybernetics, 67, pg. 97-102, 1992

[21] Pentland, A., Fractal-Based Description of Natural Scenes, IEEE Trans. PAMI, Vol. PAMI-6, No. 6, Nov. 1984

[22] Pollen, D.A., and Ronner, Visual Cortical Neurons as Localized Spatial Frequency Filters, in: IEEE trans. SMC, vol SMC-13, No. 5, Sept./Oct. 1983

[23] Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., Numerical Recipes in C, Cambridge University, Cambridge, New York, Melbourne, 1988

[24] Rumelhart, D.E., McClelland, J.L., et al., Parallel Distributed Processing, Vol. 1, MIT-press, 1986

[25] Ryan, T.W., Sementilii, P.J., and Hunt, B.R., Extraction of Shoreline Features by Neural Nets and Image Processing, in: Photogrammetric Engineering and Remote Sensing, Vol. 57, NO. 7, July 1991, Pg. 947-955

[26] Theimer, W.M., and Mallot, H.A., Binocular Vergence Control and Depth Reconstruction using a Phase Method, in: : Artificial Neural Networks, Proc. ICANN 92 Brighton, Elsevier Science Publ., Amsterdam, 1992, vol. 1, pg. 517-521

[27] v.d.Malsburg, C, et Buhmann, J., Sensory Segmentation with Coupled Neural Oscillators, in: Biol. Cybern., 67, pg. 233-242, (1992)

[28] Waxman, A. M., Seibert, M., Cunningham, R., Wu, J., Neural Analog Diffusion-Enhancement Layer and Spatio-Temporal Grouping in Early Vision, Proceedings of ICANN 91, Helsinki, Elsevier Science Publishers, North Holland, (1991)

[29] Weigl, K., and Berthod, M., The Restauration of Images with a System of Locally-coupled Limit-cycle Oscillators, in: Proc. of the 9th IAICV, Ramat Gan, Israel, 1992

[30] Weigl, K., and Berthod, M., Metric Tensors and Dynamical Non-Orthogonal Bases: An Application to Function Approximation, in Proc.

WOPPLOT 1992, Workshop on Parallel Processing: Logic, Organization and Technology, Springer Lecture Notes in Computer Sciences, to be published.

[31] Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors: An Application to Artificial Neural Networks, in New Trends in Neural Computation, Proc. IWANN'93, International Workshop on Artificial Neural Networks, Springer Lecture Notes in Computer Sciences, vol. 686.

[32] Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors, in: Workshop on Neural Networks, Huening H. et al. eds, Aachen 1993, Verlag der Augustinus Buchhandlung, ISBN 3-86073-140-8

[33] Weigl K., and Berthod, M., Neural Networks as Dynamical Bases in Function Space, Technical Report INRIA, to be published.

[34] Zerubia, J., et Chellappa, R., Mean Field Annealing for Edge Detection and Image Restoration, European Signal Processing Conference, Barcelona, Spain (1990)